Università di Pisa
Facoltà di Ingegneria

Corso di Laurea Specialistica in
Ingegneria Elettronica

Tesi di Laurea

Definition and design of a new communication protocol and interfaces for data transmission in High Energy Physics experiments

Relatori

Prof. Luca Fanucci

Ing. Sergio Saponara

Ing. Guido Magazzù

Candidato

Claudio Tongiani

Table of Contents

Introduction

Particle physics, often referred to as High Energy Physics (HEP), is the branch of physics that studies the most basic constituents of matter, i.e. subatomic particles, and their interactions in order to answer key questions about nature and origin of the Universe and improve the understanding of the fundamental laws that regulate it. The main instruments for High Energy Physics are particle accelerators, complex machines that produce beams of particles and provide them with the high energies needed by High Energy Physics experiments, that consist in colliding particle beams and studying the results of the collisions using particle detectors that surround the interaction point. Most of the attention, in the world of particle physics, is nowadays focused on the Large Hadron Collider (LHC) at CERN (the European Organization for Nuclear Research) that is now the world's largest and most powerful particle accelerator and promises, by means of collision energies never reached before, to make new particle events observable that could answer fundamental questions of nowadays physics, such as the existence of the Higgs boson, that could explain the origin of mass, and supersymmetric particles, related to dark matter and dark energy.

Current HEP experiments have very similar architectures with respect to systems for acquisition of data from sensors and for control and management of the detector. Signals generated by sensors in particle detection are handled by Front-End (FE) electronics embedded in the detectors and transferred to remote data acquisition (DAQ) systems, that are placed far away from the experiment's area to keep them in an environment that is free from the intensive levels of radiation that is present in the proximities of the interaction point: typically the transfer is carried out by means of electrical links for a first stretch inside the detector, and then through optical links that allow to cover the long distances (hundred of meters) from the experiment's area to the remote DAQ system, and provide the large bandwidth needed (up to tens of Gbit/s). The DAQ systems constantly receive a stream of data from a subset of the sensors which is used to detect significant events: if such an event is detected the DAQ system sends a trigger to the Front-End electronics which commands the start of a full-scale DAQ process from all the detectors: this way the very large amount of data generated in the detector is reduced to rates that can be handled by the readout system (a typical order of magnitude is hundreds of MB/s from each FE device), and only the interesting events are selected. A remote control system, also called TTC (Timing, Trigger and Control) system, manages trigger distribution and the configuration and monitoring processes in the Front-End electronics.

Also in future HEP experiment, such as the ones for the two-phase upgrade (in 2013 and 2018) of LHC that will lead to the so-called Super LHC, DAQ and TTC systems will have similar architectures, and therefore they will still share most of the requirements with respect to data rate, trigger latency, robustness against transmission errors and component failures, radiation hardness and power dissipation of hardware components; hence, common solutions for data transmission systems that can be shared and reused in several applicative contexts can minimize time, risks and effort for the development of new experiments. In particular, electrical serial links between Front-End electronics and Electrical to Optical Converters (EOCs) will be required; these links will have to cover a wide range of data rates from tenths of Mbit/s from each FE through intermediate stages of data concentration to one Gbit/s. The literature on electrical links is mostly about communications below one Mbit/s for configuration/control tasks, and standard protocols used in telecommunications, consumer electronics and automotive industry are not suitable for this field of application, failing to satisfy the stringent requirements above mentioned especially about radiation hardness and trigger latency.

Moving from these considerations, the FF-LYNX (Fast and Flexible links) project was started in January 2009 by a collaboration between INFN-PI (Italian National Institute for Nuclear Physics, division of Pisa) and the Department of Information Engineering (DII_IET) of the University of Pisa, with the aim of defining a new serial communication protocol for integrated distribution of TTC signals and Data Acquisition, satisfying the typical requirements of HEP applications and

providing flexibility for its adaptation to different scenarios, and of its implementation in radiation-tolerant, low power interfaces. The work presented in this thesis constituted a phase of the FF-LYNX project working plan and was carried out at the Pisa division of INFN: in particular, it dealt with the definition of a first version of the FF-LYNX protocol and the design of hardware transmitter and receiver interfaces implementing it. A description of the contents of this thesis follows.

In chapter 1 the FF-LYNX project is presented: after a brief description of High Energy Physics experiments and typical requirements of data transmission systems for this field of application, the purposes of the project are presented and the methodology defined for the project work is outlined.

Chapter 2 describes the FF-LYNX protocol, and in particular the version 1 that was defined and implemented in this thesis work: the basic issues about trigger and data transmission that were considered in the definition of this version of the protocol are outlined, as well as the solutions that were adopted to address these issues. Also, new features that are foreseen to be included in future versions of the protocol are briefly described, and finally the results of simulations in a high-level model of the link, intended to estimate various aspects of the protocol performance, are presented.

Chapter 3 illustrates the architecture that was defined for the interfaces implementing the FF-LYNX protocol version 1: a functional description of input and output ports of the FF-LYNX Transmitter and Receiver is given first, and then their internal functional architecture is outlined.

Chapter 4 presents the VHDL models of the transmitter and receiver blocks that was created in the design phase of the FF-LYNX interfaces implementing the protocol version 1.The functionality of each component is described in detail also showing the results of VHDL simulations on it. Finally, the structure and the operation of the VHDL test bench built to test the functionality of the complete transmitter-receiver system are described, and results of simulations on it are reported.

In chapter 5 an FPGA based emulator for the FF-LYNX transmitter-receiver system, foreseen as the final result for the FF-LYNX project first year of activity, is outlined in its functional architecture, and the development board chosen for its implementation is briefly described. Finally, the results of preliminary synthesis trials of the designed TX and RX blocks onto the target FPGA are reported.

This thesis ends with the exposition of conclusions about the work that has been done, and the outlining of next developments that are foreseen in the future activity for the FF-LYNX project.

1 The FF-LYNX project

In this chapter the FF-LYNX project is presented. The FF-LYNX project, approved and funded by INFN V Commission, started in January 2009 and, in a three-year foreseen activity, aims at the design of an integrated and scalable system for data acquisition and control in High Energy Physics experiments. To this end, the definition of a novel communication protocol is foreseen, as well as its implementation in radiation tolerant and low power interfaces to be provided to ASIC designers for inclusion in their systems.

The project genesis is outlined first in section 1.1, with a brief introduction to the field of High Energy Physics experiments and the description of the requirement for data transmission systems in this area of application; then the goals of the project are illustrated in section 1.2, and finally section 1.3 presents the methodology that was settled for the project activities and the design flow.

1.1 Genesis of the project

The FF-LYNX project was born from the experience of INFN in the field of High Energy Physics (HEP) experiments and collaborations with the most important research centers for particle physics (such as CERN, Fermilab, etc.) and in particular from considerations about the opportunity of a research and development activity focused on the definition of an innovative data transmission system that could meet the typical requirements of HEP scenarios, intended to become a new flexible standard for application to different experiments thus minimizing development costs and efforts. With this aim, the FF-LYNX project was started as a collaboration between the experience of INFN in design and development of Silicon and Gas detectors and radiation tolerant Integrated Circuits, and the experience of DII-IET in communication protocols and radiation tolerant interfaces for space applications.

1.1.1 High Energy Physics experiments

Particle physics is the branch of physics that studies the most basic constituents of matter and their interactions. Modern particle physics research is focused on subatomic particles, i.e. particles with dimensions and mass smaller than atoms, including atomic constituents such as electrons, protons and neutrons and particles produced by radiative and scattering processes, such as photons, neutrinos and muons. Since many elementary particles do not occur under normal circumstances in nature, to allow their study they are created and detected by means of high energy collisions of other particles in particle accelerators: therefore, particle physics is often referred to as *High Energy Physics* (HEP). The purpose of particle physics is to investigate the fundamentals of matter in order to address unanswered key questions about nature and origin of the Universe, such as symmetries in physical laws, the origin of mass, the nature of dark matter and dark energy, possible existence of extra dimensions and so on; the final, ambitious objective would be the creation of a general theoretical model that is able to describe and explain all physical phenomena in an unified and coherent vision.

The main instruments for High Energy Physics are therefore particle accelerators, large and complex machines that produce beams of particles and provide them with the high energies needed for HEP experiments. Accelerators typically employ electric fields to increase kinetic energy of particles and magnetic fields to bend and focus the beam, which is then collided against a fixed target or with another particle beam: the high energy collision produces the new particles and events that must be detected and studied. Beside the use in particle physics, the applications of accelerators nowadays span from industry (e.g. ion implantation in electronic circuits) to medicine

(radiotherapy), with different ranges of energy for the different fields application. Current accelerators for High Energy Physics work in the GeV and TeV energy range (referring to the energy provided to particle beams) and typically treat beams of electrons, hadrons[1] and heavy atomic nuclei; the structure can be linear (LINAC, LINear ACcelerator), with the particle beam traveling from one end to the other and colliding against a fixed target, or circular (cyclotrons, synchrotrons), with the beams traveling repeatedly around a ring, gaining more energy at every loop and colliding with other beams running in opposite direction (Fig. 1.1).

The major research centers for High Energy Physics nowadays include, among the others:

- The European Organization for Nuclear Research (CERN), located near Geneva, Switzerland: its main facilities included LEP (Large Electron Positron collider), which was dismantled in 2001 and substituted with LHC (Large Hadron Collider) that is now the world's most energetic accelerator;
- DESY (Deutsches Elektronen Synchrotron), located in Hamburg, Germany: its main structure is HERA, which collides electrons or positrons and protons;
- the Fermi National Accelerator Laboratory (Fermilab), located near Chicago, USA: its main facility is the Tevatron, which collides protons and antiprotons;

(a)

(b)

Fig. 1.1 – Aerial view of two particle accelerators for HEP experiments: (a) the Stanford Linear Accelerator Center (SLAC), an example of LINAC; (b) the Tevatron at Fermilab, that is an example of circular accelerator.

[1] *Hadrons* are the subnuclear particles that are subject to the strong force, and are constituted by quarks. The family of hadrons is then divided in two subsets: *baryons*, comprising neutrons and protons, and *mesons*, including pions and kaons.

- the Stanford Linear Accelerator Center (SLAC), located near Palo Alto, USA: it hosts the longest linear accelerator in the world, colliding electrons and positrons;
- the INFN (Istituto Nazionale di Fisica Nuleare) center in Frascati, Italy, that hosts DAΦNE (Double Annular ring for Nice Experiments) a circular accelerator for the collision of electrons and positrons.

High Energy Physics experiments thus consist in colliding particle beams in accelerators and studying the results of the collisions by means of particle detectors that surround the interaction point. *Particle detectors* are devices used to track and identify high-energy particles produced in collisions, also measuring their attributes like momentum, charge and mass. A particle detector is typically made up of different layers of sub-detectors, each specialized in revealing and measuring different particles and properties of particles: normally the innermost layer (i.e. the nearest to the interaction point) is the tracking device, that has the task of revealing the paths of electrically charged particles through the trails they leave behind; in the outer layers calorimeters are typically placed, that measure the energy lost by particles that go through them. To help identify the particles produced in the collisions, the detector usually includes a magnetic field that bends the path of charged particles: from the curvature of the path, it is possible to calculate the particle momentum which helps in identifying its type. Particles with very high momentum travel in almost straight lines, whereas those with low momentum move forward in tight spirals.

To record and analyze events produced by collisions in an experiment, information about particles detected by sensors in the detector are converted into electric signals, which are then collected by dedicated electronic components embedded in the detector and located in close contact with the sensors themselves: these devices, usually called Front-End (FE) electronics, deal with the proper conditioning of signals (e.g. amplification, shaping, buffering, analog to digital conversion) and their transmission to remote data acquisition systems that perform data analysis and storage. However, some means is needed to reduce the amount of data that must be transferred from the detector to the remote system, that is extremely large in every HEP experiment. In fact, to increase the probability of occurrence of rare, interesting events, the number of interactions per second is made very high (a typical order of magnitude is billions of particle interactions per second); a measure of collision rate is the so-called *luminosity*, which is usually expressed in $cm^{-2}\ s^{-1}$ and for a two-beam collider is defined as the number of particles per second in one beam multiplied by the number of collisions per unit area in the other beam at the crossing point. Collision rates of this order of magnitude produce amounts of raw data that range from tens of terabyte to a petabyte per second, which is beyond the possibility of any data acquisition and storage system. Therefore, since the interesting events are a very small fraction of the total, the total amount of data is filtered by means of a *trigger system*: raw data are temporarily buffered in the FE electronics while a small amount of key information is used by trigger processors (located at various hierarchical level inside the detector and in the remote elaboration center) to perform a fast, approximate calculation and identify significant events: the result of this processing is a trigger signal that is sent back to FE electronics to command a data readout, i.e. the transferring of a selection of the buffered data towards the remote system. This way, the amount of data to be transferred is reduced to rates that can be handled by the readout system (a typical order of magnitude is hundreds of MB/s from each FE device), and only the interesting events are selected.

A typical control and readout system for a HEP experiment can be schematized as in Fig. 1.2:

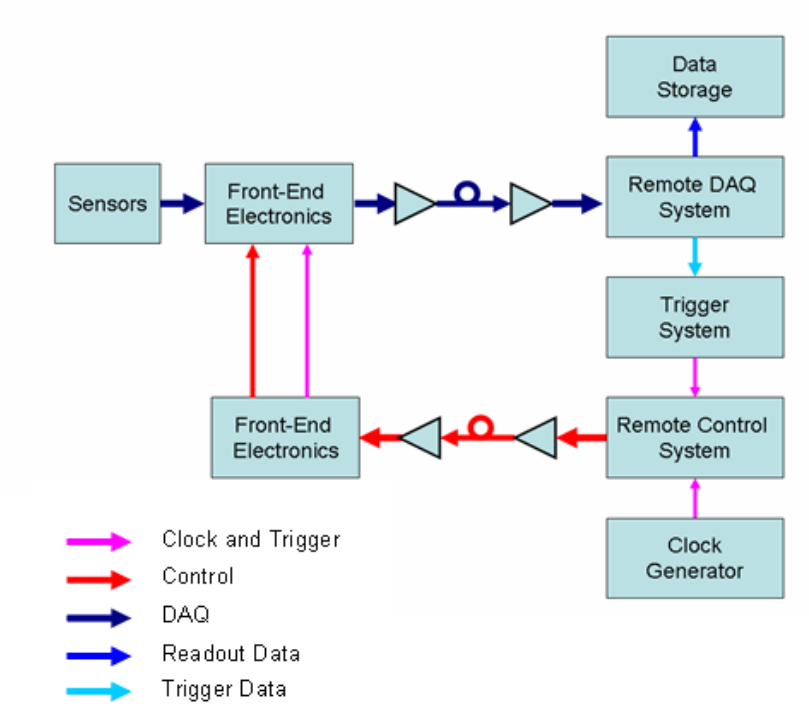


Fig. 1.2 – Typical architecture of the control and readout system for a HEP experiment.

Signals generated by the interaction with sensors of particles produced in the beam collisions are handled by Front-End electronics embedded in the detectors and transferred to remote data acquisition (DAQ) systems, that are placed far away from the experiment area to keep them in an environment that is free from the intensive levels of radiation that are present in the proximities of the interaction point: typically the transfer is carried out by means of electrical links for a first stretch inside the detector, and then through optical links that allow to cover the long distances (hundred of meters) from the experiment area to the remote DAQ system, and provide the large bandwidth needed (up to tens of Gbit/s). A subset of the transferred data is used to perform trigger calculation, and the generated trigger command is sent back to FE electronics along with timing (clock) and control signals by a remote control system, also called TTC (Timing, Trigger and Control) system, that manages the configuration and monitoring processes in the Front-End electronics.

High Energy Physics experiments constitute a very challenging application for electronics, since the equipment must deal with large amounts of data and high data rates, with tight timing and data integrity constraints and operate in an environment that is intrinsically hostile due to the high levels of radiation. Typical requirements for detector electronics and data transmission links in a HEP experiment are:

- radiation hardness: electronic devices and systems must tolerate high levels of ionizing radiations and the associated Total Dose Effects (e.g. threshold voltage drift and sub-threshold current increase in MOS devices) and Single Event Effects (e.g. Single Event Upset in flip-flops and SRAM cells);
- small size: the space available for devices and cabling inside a particle detector is usually very limited due to the large amount of different components (readout and control systems, cooling systems, mechanical structures and so on) that must be integrated in a small area around the interaction point; additionally, bulky equipments are undesired because any non-sensor material interferes with the measure by deflecting and absorbing the particles that must be detected (the amount of material surrounding the interaction point, characterized with the radiation thickness of each component/layer, is usually referred to as *material budget*);

- low power dissipation: due to the high concentration of electronic equipment inside the detector, power density is a major issue because it dictates the cooling requirements; cooling system is a critical aspect in HEP experiments because it complicates the material budget and the mechanical requirements;
- constant trigger latency: the trigger signal must be delivered to all Front-End devices with a fixed and known latency, to command the readout of selected data thus allowing precise reconstruction of significant events;
- capability of handling high data rates: readout electronics must be able to elaborate and transfer large amount of data in a limited time to allow a continuous data flow during the running of the experiment, with minimal loss of information; for the same reason, adequately high bandwidth in electrical and optical links are required;
- data integrity: appropriate methods must be employed in transmission links to protect data against transmission errors, and in storage elements to deal with data corruption due to radiation.

1.1.2 The Large Hadron Collider at CERN and its experiments

The Large Hadron Collider (LHC) at CERN is the world's largest and most powerful particle accelerator. It is a circular structure of 27 km of circumference, located in an underground tunnel (100 m of average depth) near Geneva, Switzerland. It is constituted by two beam pipes, kept at ultrahigh vacuum, carrying two separate particle beams that travel in opposite directions. They are guided around the accelerator ring by a strong magnetic field, achieved using over 1600 superconducting electromagnets operating at a temperature of −271°C, that generate a magnetic field of about 8 Tesla.

LHC accelerates beams of protons and heavy ions (lead nuclei) and make them collide in four points of the accelerator ring, where large cavities are built around the tunnel to house the detectors of the LHC experiments; the rate of collisions is 40 millions per second, and hence 40 MHz is the frequency of the LHC master clock that synchronizes all the detectors. The energy of proton beams is 7 TeV for each particle, leading to a collision energy of 14 TeV which is the highest level ever reached in a laboratory; lead nuclei are accelerated to an energy of 2.7 TeV per nucleon. The design luminosity is 10^{34} cm^{-2} s^{-1} for protons.

The goal of LHC is to observe, by means of collision energies never reached before, new particle events that could answer most of the key questions of nowadays physics: in particular, experimental evidences are expected for the existence of the Higgs boson, that is theorized as the explanation to the origin of mass, and supersymmetric particles, that could clarify the nature of dark matter and dark energy.

LHC hosts six experiments: A Large Ion Collider Experiment (ALICE), A Toroidal LHC ApparatuS (ATLAS), the Compact Muon Solenoid (CMS), the Large Hadron Collider beauty (LHCb), the Large Hadron Collider forward (LHCf) and the TOTal cross section, elastic scattering and diffraction dissociation Measurement (TOTEM). The two larger experiments, ATLAS and CMS, are based on general-purpose detectors and are designed to investigate the largest range of physics possible. Two medium-size experiments, ALICE and LHCb, have specialized detectors for analyzing the LHC collisions in relation to specific phenomena. ATLAS, CMS, ALICE and LHCb detectors are installed in four huge underground caverns located around the beam crossing points of the LHC (Fig. 1.3). TOTEM and LHCf experiments are smaller in size, and are positioned near the CMS detector and the ATLAS detector, respectively.

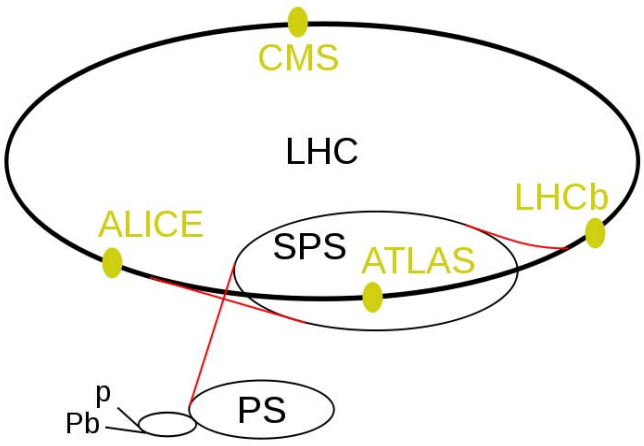


Fig. 1.3 – The LHC complex and its four main experimental points. Pb and p denote the linear accelerators that generate the ion and proton beams, which are then injected into the preaccelerators PS and SPS and finally into the main ring.

LHC operation was started for a first test beam circulation on 10 September 2008, but was then stopped few days later due to a severe fault in a superconducting magnet; after repairs and addition of supplementary safety features, the new turn-on is scheduled for November 2009. However, a two-phase luminosity upgrade for the accelerator is already planned, to increase the probability of observing rare events and improve the measurement precision: the upgraded accelerator will be called Super LHC (SLHC) and will have a luminosity increase by a factor of 2-3 for phase I, planned for 2013, and by a factor of 10 for phase II in 2018, reaching the value of 10^{35} cm^{-2} s^{-1} [1]. The consequently larger rate of events to be detected and the resulting larger amount of data to be handled will impact the detectors, which will have to be upgraded as well.

The work of requirement analysis inside the FF-LYNX project focused on the two main experiments at LHC: CMS and ATLAS, that are briefly described in the following sections.

1.1.2.1 CMS

The Compact Muon Solenoid (CMS) experiment uses a general-purpose detector to investigate a wide range of physics, including the search for the Higgs boson, extra dimensions, and supersymmetric particles. It is located in an underground cavern at Cessy in France, close to the Swiss border and Geneva.

The experimental apparatus is 21 m long, 15 m wide and 15 m high, and comprises different kinds of detectors laid out in concentric layers around the beam pipe, where the collision point is located (Fig. 1.4). From the interaction point to the outside, these detectors are:

- the *tracker*, a silicon detector that reveals the trajectories of charged particles;
- the *electromagnetic calorimeter* (ECAL), made up of lead tungstate sensors that measure the energies of electrons and photons;
- the *hadronic calorimeter* (HCAL), that has the task of measuring the energy of hadrons produced in each event: it is composed of alternating layers of absorbing materials and fluorescent “scintillator” materials;
- the *muon chambers*, for the detection of muons.

The detecting devices are enclosed inside a huge solenoid magnet that has the form of a cylindrical coil of superconducting cable and generates a magnetic field of 4 Tesla. The magnetic field is confined by a steel 'yoke' that forms the bulk of the detector weight of 12500 tons. As in most HEP experiment, the magnet is an essential part of the experimental apparatus, since the bending of the particle trajectory in the magnetic field allows to measure the momentum of a particle by tracing of its path.

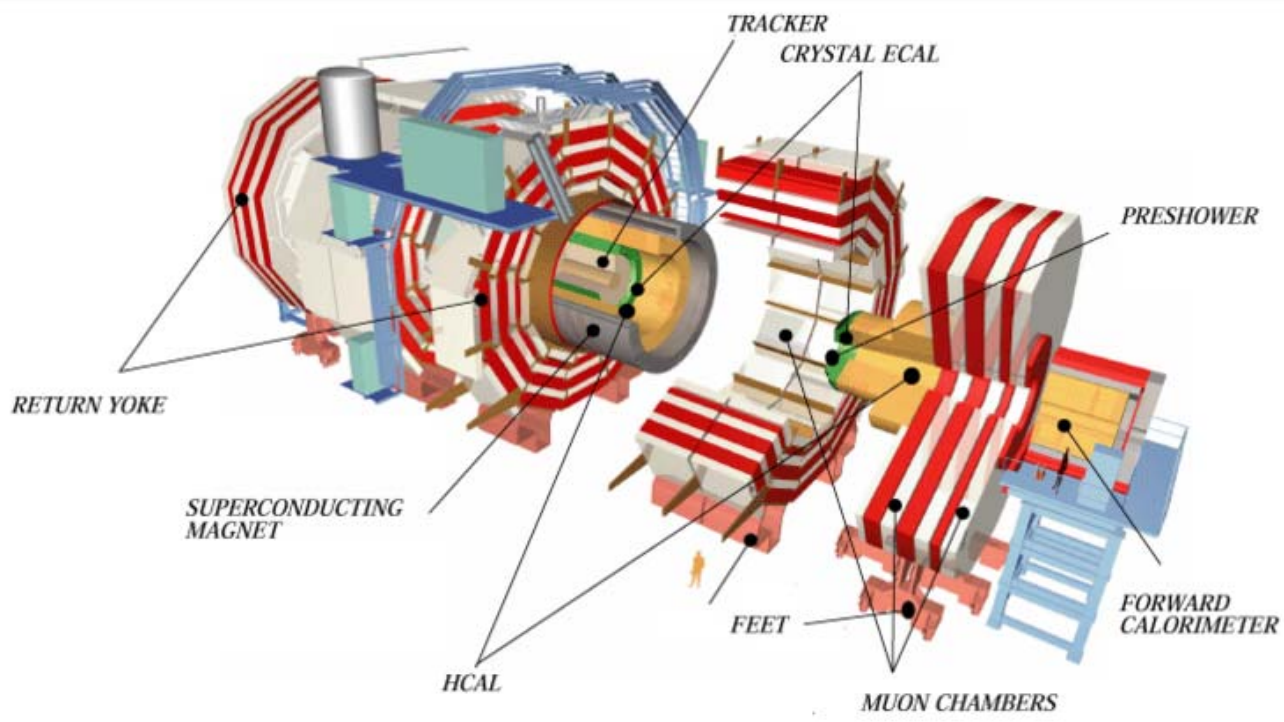


Fig. 1.4 – The experimental apparatus of CMS.

At full operation, about one billion proton-proton interactions will take place every second inside the CMS detector. To allow data storage and processing, the event rate must be drastically reduced selecting only the potentially interesting events: this task is performed by the trigger system, that carries out the reduction in two steps called Level-1 Trigger (L1T) and High-Level Trigger (HLT). The Level-1 Trigger consists of custom-designed electronics that use coarsely segmented data from the calorimeters and the muon system, while holding the high-resolution data in pipelined memories in the Front-End electronics; the L1T calculation is carried out in about 1 μs, and reduces the event rate from the original 1 GHz to some tens of kHz: the design output rate limit of the L1 trigger is 100 kHz. The readout data of events that pass the L1T filter are sent is sent over fiber-optic links to the High Level Trigger, which is a software system implemented in a farm of commercial processors. The HLT has access to the complete read-out data and can therefore perform complex calculations to select specially interesting events, thus reducing the event rate to the final value of about 100 per second.

For requirement analysis our attention focused on the CMS tracker system, that is the detector with highest data flow rates being the innermost layer of the CMS apparatus. The tracker is made up of two sub-systems: a *pixel detector*, with three cylindrical layers located in the central section of the CMS detector around the beam pipe (the so called *barrel*), at radii between 4.4 cm and 10.2 cm; and a *silicon strip tracker* with 10 barrel detection layers extending outwards to a radius of 1.1 m. Each system is completed by endcaps which consist of 2 disks in the pixel detector and 3 plus 9 disks in the strip tracker on each side of the barrel.

The pixel detector is composed of several silicon sensor plates, segmented in 150 μm x 150 μm sensitive elements (the pixels): each pixel consists in a reverse-biased p-n junction, that reveals the passage of charged particles by generating an electric current. The sensor signal is collected and handled by a silicon Read-Out Chip (ROC), bump-bonded to the pixel sensor (Fig. 1.5).

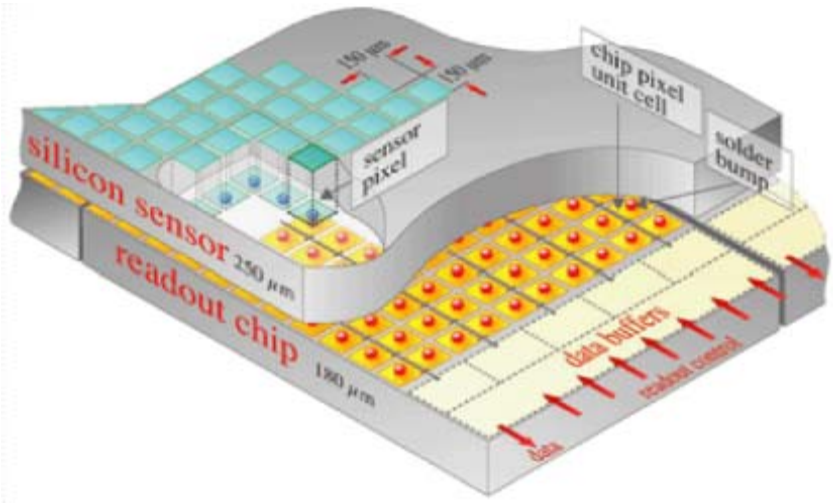


Fig. 1.5 – Structure of a CMS Pixel Detector.

Each sensor plate is mounted on a module, together with ROCs and a printed board that hosts circuits for monitoring voltage, current and temperature, for clock synchronization and trigger decoding, for concentrating data to be sent to the remote DAQ system. Data transmission is made on electrical links from each module to electro-optic conversion units (EOC) that are located at the end of the barrel, and on optical links from EOCs to the remote data gathering system. The central TTC system sends trigger and clock through optical links to distribution devices, that in turn deliver them to Front End modules through electrical links.

The strip detector has a structure that is similar to the pixel detector, but the silicon sensor plates are here segmented in thin strips (10 cm x 180 μm in the inner four layers, and 25 cm x 180 μm in the remaining six layers). The charge on each microstrip is read out and amplified by an Analogue Pipeline Voltage (APV25) chip. Four or six such chips are housed within a "hybrid", which also contains electronics to monitor key sensor information, such as temperature, and provide timing information in order to match "hits" with collisions. The APV25 stores the signals in a memory for several microseconds and then processes them before sending to a EOC unit, from which data are sent to the remote DAQ system through optical fibers.

In both pixel and strip detectors, when a Front-End circuit receives a trigger signal from the centralized control system, the stored samples corresponding to the triggered event are serialized and sent to the concentrator, and from here to the optical transmitter. Currently this readout process is partially analog: readout data are stored in Front-End chips, concentrated and transmitted to a first stage of collection systems in analog form, and only subsequently they are digitized, processed and transmitted to the remote elaboration center. Analog readout data transmission is serial and synchronous with the 40 MHz LHC clock. For example, a ROC in the pixel detector transmits the data relative to a single "hit" (i.e. a pixel being hit by a particle), comprising the pixel address and the hit amplitude, in six successive 40 MHz clock cycles with an analog signal that can assume one of 6 different levels at each cycle. When digitized, the data for a pixel hit in a ROC correspond to 23 bits [5][6]. Considering then the occupancy, i.e. the probability for a channel to generate a significant event in correspondence to a trigger command, that is currently about 5.4 hits/ROC at each trigger event [6], we can estimate the readout data rate from each ROC in 23 bit x 5.4 x 100 kHz = 12.42 Mbit/s. Considering then the occupancy for a 16-ROC module, that is around 6.5 hit ROCs per module [6], we have a data rate from each module that is estimable as 6.5 x 12.42 Mbit/s = 80.73 Mbit/s.

The analog readout scheme was chosen mainly for resolution reasons, since the availability of an analog measure of hit amplitudes in pixel and strips permits, in case of charge sharing between adjacent sensor elements, interpolation techniques that in turn increase position resolution [9]; also, material budget is reduced as the analogue to digital conversion and its power needs are shifted out

of the tracker volume. However, in real operation the analog readout proved to have unsatisfactory performance mainly due to signal degradation in the readout chain and temperature sensitiveness of analog to optical converters [10]. This fact, together with the need for higher transmission speed to handle the larger data rate that will result from luminosity increase, dictates the switch to a fully digital readout scheme for the LHC upgrade, already in phase I. Considering again the case of the CMS pixel detector, for the phase I upgrade an increase factor of about 3 is foreseen for data rates from ROCs and from modules [6], leading to foresee a 160 Mbit/s digital link from each ROC to the data concentrator in the module, and a 320 Mbit/s digital link from each module to the EOC unit [6]. Similar requirements will show up for the strip detector.

1.1.2.2 ATLAS

ATLAS is the other general-purpose detector at the LHC; it is located in an underground cavern at Meyrn, near Geneva. It has the same physics goals as CMS, recording similar sets of measurements on the particles created in the collisions: their paths, energies, and their identities; however, the two experiments have adopted different technical solutions and designs for their detectors' magnet systems. The comparison of the results of the two experiments will be important for cross-confirmation of new discoveries.

The experimental apparatus is 46 m long, 25 m high and 25 m wide and is based, similarly to CMS, on a large superconductive magnet generating a magnetic field of 2 Tesla. Integrated with the magnet, the following concentric layers of detectors are displaced around the beam pipe (Fig. 1.6), listed here from the interaction point to the outside:

- the *inner tracker*, that measures the momentum of each charged particle;
- the *electromagnetic and hadronic calorimeters*, that measure the energies carried by the particles;
- the *muon spectrometer*, that identifies and measures muons.

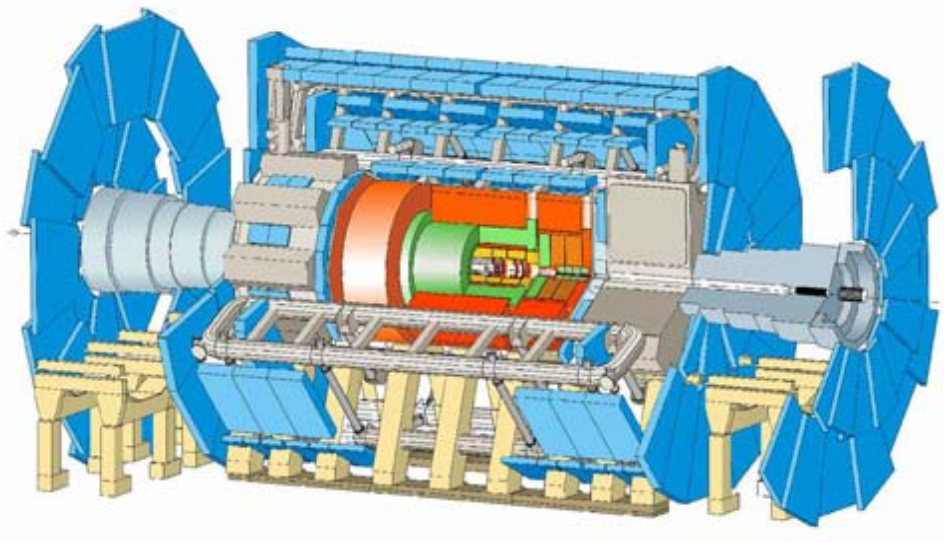


Fig. 1.6 – The ATLAS detector.

The inner tracker is made up of three parts: the Pixel Detector, the Semi-Conductor Tracker (SCT) and the Transition Radiation Tracker (TRT). The Pixel Detector is the innermost part of the detector, and consists of 250 μm thick silicon sensors divided in 50 μm x 400 μm pixels and distributed in modules; each module contains 16 readout chips and other electronic components. The SCT is the middle component of the inner detector: it is similar in concept and function to the Pixel Detector but with long, narrow silicon strips (80 μm x 12.6 cm) rather than small pixels,

making coverage of a larger area practical. The TRT, the outermost component of the inner detector, is a combination of a straw tracker and a transition radiation detector. The detecting elements are drift tubes (straws), filled with gas that becomes ionized when a charged particle passes through. Between the straws, materials with widely varying indices of refraction cause ultra-relativistic charged particles to produce transition radiation and leave much stronger signals in some straws.

The detector generates about 25 Mbytes of raw data per event, that multiplied for 40 million beam crossings per second give a total of 1 petabyte/s of raw data. The trigger system, structured in two levels, uses simple information to identify, in real time, the most interesting events to retain for detailed analysis. The first-level (LVL1) trigger works on a subset of information from the calorimeter and muon detectors. It requires about 2 μs to reach its decision, including the propagation delays on cables between the detector and the underground counting room where the trigger logic is housed. All of the information from the detector must be stored in pipeline memories until the LVL1 decision is available.. After the first-level trigger, about 100,000 events per second have been selected. The LVL2 trigger refines the selection of candidate objects, using full-granularity information from all detectors, including the inner tracker which is not used at LVL1. In this way, the rate can be reduced to about 1kHz.

The collection and readout of sensor data and the distribution of TTC signals are done with modalities and features that are similar to the ones in the CMS detector, and therefore also the requirements of data transmission systems about speed, radiation hardness, material budget and so on are similar, and will be similar for the LHC upgrade.

1.1.3 Requirements of DAQ/TTC systems for HEP experiments

From the scenario depicted in the preceding sections clearly appears that most High Energy Physics experiments, although different for the adopted technologic solution and implementative choices, share basic architectures and operating principles, and have similar requirements for the Data Acquisition and TTC systems; and the same will be in the near future. These requirements can be summarized in the following way, analyzing the different aspects of data readout and control processes.

- **Data readout**. Data collected from sensors by Front-End electronics must be transferred to the remote acquisition and elaboration system in response to trigger signals, that command data readout; this direction of data transfer is usually called *uplink*. The links that operate the transmission of readout data must provide a bandwidth that is adequate to the expected data rate, in order to minimize data losses due to buffer overflows and situations of busy data transfer systems. Usually data from different readout circuits are then gathered by a data concentrator inside a module, to allow their multiplexing to only one or two high-bandwidth links per module thus reducing the total number and length of cablings. The values required for link speed commonly range from 40 Mbit/s to 160 Mbit/s for the links between single readout chips to the data concentrator inside a module, and from 160 Mbit/s to about one Gbit/s for the links departing from each module. Packets of readout data must also be characterized with timing specifications in order to make their association to specific events possible by the elaboration system: typically a “time stamp” (i.e. a binary word whose value specifies a timing information) is attached to each data packet to associate it to a specific trigger command, and later in the readout chain the so called *event building* process is performed by grouping together all the data with a same time stamp thus reconstructing the physics event sensed by the detector.
- **Trigger system**. Trigger commands have to be transmitted from the central elaboration system to the Front-End circuits (*downlink* direction) with two fundamental requisites: the number of triggers that are lost due to transmission errors or other malfunctions must be minimum (possibly zero), and the latency of each trigger command from the central system to

each Front-End circuit must be fixed and known, in order to allow the correct time stamping of readout data and thus the correct reconstruction of events. These constraints must be fulfilled through a trigger transmission scheme that guarantees particular properties of high priority and error protection to the trigger signal; at the same time, however, this scheme must allow the transmission of triggers that are close in time up to a limit foreseen by the experiment: for example, the minimum interval between two consecutive triggers in the CMS and ATLAS experiment is respectively 3 and 5 cycles of the 40 MHz clock [11][13].
In the uplink, trigger data (i.e. the information from a subset of sensors that is used for trigger calculation) must be transmitted; the requirements here are again fixed latency and high bandwidth: for example, in the CMS upgrade it is foreseen that data from the microstip tracker will participate to the L1 trigger generation, with data rates that are foreseen to be in the range 40-120 Mbit/s from each FE ASIC and up to about one Gbit/s from each module [14].

- **Timing and control system**. The master clock of the experiment (e.g. the 40 MHz accelerator clock for the LHC experiments) must be delivered to all Front End circuits with controlled skew, in order to synchronize the detectors with the beam crossings. The downlink has also to transmit so called "slow control" commands to control and configure Front End devices; here the link speed requirements are more relaxed, with 40 Mbit/s being an adequate value for most cases. Currently, slow control is performed on dedicated links with custom or commercial protocols such as I^2C [5], and a dedicated uplink is used for monitoring and response data from FE to the central control system.
- **Error control**. Critical data must be protected against transmission errors, that can arise from noise in the transmission lines and from Single Event Effects affecting the decision devices (e.g. photodiodes at the end of optical links): the environment of HEP experiment is particularly critical under this point of view. Transmission errors can occur in single events or in bursts, due to *jets* of particles hitting the lines. Error control coding scheme therefore must be adopted to limit data losses. The most critical signal under this point of view is the trigger, because a trigger command not reaching a FE circuit due to corruption means the loss of readout data that is associated to a potentially interesting event.
- **Radiation hardness**. Having to operate for years in an extremely radiation-full environment, all the electronic components must tolerate high levels of ionizing radiations (up to tens of Mrad [11]) and the associated Total Dose Effects and Single Event Effects without major malfunctions. Electronic systems hence must be designed and realized with proper rad-hard techniques and technologies, and submitted to irradiation tests.
- **Material budget**. The amount of non-sensor material (mechanical structures, cablings, cooling systems, etc.) inside the detector is a major concern in any HEP experiment, because it absorbs and deflects particles interfering with the measurement. The material budget is usually analyzed in detail for each component of the system, and for each part it is expressed as X/X_0, i.e. the thickness of the material in units of its radiation length[2] X_0. For data transmission systems the desirable thing is to reduce the number of physical links as much as possible, thus alleviating also the problem of fitting the required cabling in the limited space available inside the detector. Some ways to achieve this are the integration of more services (timing and trigger distribution, data transmission, etc.) on the same links, the use of serial transmission schemes rather than parallel ones, and the concentration of data streams from more sources (FE circuits) into few, high-bandwidth physical links. In some cases, the number of available cables is strictly limited due to available space constraints, and this poses the most stringent requirement for data transmission systems: for example, in the CMS upgrade for 2013 the number of pixel modules will be increased from 784 to1216 but only the existing fibers can be used due to space limitation [6]; therefore, just one fiber per module will be employed (currently there are 2 fibers from each module in the first two barrel layers of the

[2] The *radiation length* of a material is the mean path length required to reduce the energy of relativistic charged particles by the factor 1/e as they pass through matter.

pixel detector) and the bandwidth of the link departing from each module must be increased to 320 MHz.

- **Power dissipation**. Power dissipation of electronics inside the detector is closely related to material budget, and it must be minimized mainly to relieve the requirements for power distribution and cooling system, that typically contributes to the overall material budget for the most part. In general, high-power consuming parts should be placed as farther as possible from the interaction point in order to minimize the cooling cabling in the center of the detector. As for data transmission systems, in current experiments typical values of power dissipation range from some mW to 10 mW per channel for electrical links from FE circuits to EOCs, and up to some watts per channel for optical links [15].

The FF-LYNX project moved from the analysis of these requirements, performed in close collaboration with physicists and engineers involved in the design of future detectors and Front-End electronics, and aims at the definition and implementation of innovative data transmission systems that could meet the common requirements of DAQ and TTC systems for HEP experiments, offering at the same time the necessary degree of flexibility to make the system adaptable to the specific needs and conditions of the different applications. Indeed, common solutions that can be shared and reused in several applicative contexts can maximize performance (allowing the observation of new physical phenomena) and minimize time, risks and effort for the development of new experiments. Besides, it was noted in the previous sections that most HEP experiments, for example ATLAS and CMS at LHC, and different components of the detector inside a single experiment (e.g. the pixel tracker and the microstrip tracker) share basic requirements and architectures; nevertheless, the groups of physicists and engineers that carried out the development of each experiment and are involved in the design of future upgrades appear in most cases as separate communities with little coordination, multiplying efforts while working on similar issues. Instead, a key lesson from the past ten years of activity in the design and construction of the large experiments for the LHC is that the use of common solutions for different detectors and experiments reduces efforts, resources and risks.

In particular, our attention focused on flexible electrical links, possibly integrating DAQ and TTC functions in a single protocol and relative communication interfaces: the availability of electrical serial links, with a scalable bandwidth from tens to hundreds of Mbit/s, low power consumption (<10mW/channel) and high rad-tolerance (tens of Mrad) would allow to reduce the number of interconnections within the detectors to move power-consuming EOC units away from the active regions. At the present moment, several research activities are ongoing about Gbit/s transmissions over optical fiber between EOC unit and the remote DAQ system: e.g. Gigabit Bi-directional Transceiver (GBT) by CERN and SpaceFiber by ESA [18][19]. On the contrary, for the electrical communication between sensors and the EOCs (from tens to hundreds of Mbit/s) custom solutions exist for the specific experiments and there is not an approach that allows for scalability of bandwidth, power consumption, fault- and rad- tolerance, interoperability of future experiments. The literature on electrical links is mostly about communications below one Mbit/s for configuration/control tasks, such as I^2C and CAN; on the other hand, protocols for consumer electronics, telecommunications, industrial automation, avionics, automotive, space craft applications (e.g. Ethernet, Firewire, Fiberchannel, SpaceWire) are little suitable for the stringent requirements of high rad-tolerance, low power consumption and low material budget: for instance, SpaceWire uses 8 wires for each bidirectional channel; a basic requirement that these protocols fail to satisfy is the possibility of transmitting high priority commands, such as the trigger, with fixed latency and robustness against errors.

1.2 Purposes of the project

Moving from the considerations about requisites outlined in the previous sections, the FF-LYNX project aims at the definition of a serial communication protocol that could integrate the distribution of TTC signals and Data Acquisition, satisfying the typical requirements of HEP applications and providing at the same time a degree of flexibility that allows its adaptation to different scenarios, and its implementation in radiation-tolerant interfaces designed and developed in a standard CMOS technology. The intended application for this protocol is the field of future HEP experiments, but the tailoring of the protocol features to match the constraints of this field will make it suitable for possible applications also in the area of astrophysics and space remote sensing, where the increasing use of CCD and CMOS-pixel image sensors in detectors poses similar requirements to data acquisition systems.

The proposed protocol is intended to offer the following key features:

- **Integrated distribution of TTC signals and DAQ data**. The FF-LYNX protocol is designed to allow the management of both these fundamental functionalities of data transmission in HEP experiments, applying the same interfaces to both the downlink and the uplink. In the downlink the fundamental functionality is the transmission of clock, triggers and control commands. The basic function of the uplink is the transmission of data from the Front-End systems (readout data in response to triggers or monitoring data in response to commands), but the availability of a high-priority command such as the trigger could be useful to allow the constant-latency transmission of special, fixed-length frames that carry data to be used for the generation of the Level 1 trigger. Furthermore, the protocol is intended to be applicable also to ring architectures (see later on) where there is no distinction between down-link and up-link, and triggers have to be propagated from one FE circuit to another along with readout data. Hence, the FF-LYNX protocol proposes common features to meet the requirements of the two transmission directions: the transmission on a single serial link of clock and triggers for the FE circuits, and user data (that can be configuration/control commands or readout data); user data are transmitted by encapsulating them into data frames.
- **Availability of a high-priority command**. The trigger command transmission is integrated on the same link with generic data transmission, but the protocol gives highest priority to the trigger in order to ensure its delivery to the receiver with fixed and known latency between transmission and receiving in all the conditions of traffic on the link.
- **Robustness of critical signals/commands against transmission errors**. The trigger and other signals that are critical for the protocol operation, that as will be explained in the following are frame headers and frame descriptors, are protected against transmission errors by means of an appropriate encoding that must allow the correct recognition of the transmitted command and the reconstruction of its timing.
- **Flexibility**. The FF-LYNX protocol is intended to offer a high degree of flexibility with respect to various parameters and aspects of transmission systems, such as data rate, data format, system architecture. Regarding data rate, the protocol is proposed in different versions with different values of the data transmission speed, chosen as multiples of the master clock frequency of the application (e.g. the 40 MHz clock for LHC experiments) to facilitate the generation of the transmission clock and the distribution of the master clock through the link itself. As for data format, the chosen approach is that of transparency towards the user data transported by the link, i.e. the protocol transfers user information from the transmitter to the receiver without any structuring of the transported payload, thus accepting any kind of data format. As for system architecture, finally, the protocol is designed for a basic point-to-point application, but with the possibility of applying the interfaces to different architectures of the readout and control system, being the “ring” and the ”star” topologies the most common in the detectors for HEP experiments (Fig. 1.7 and Fig. 1.8). To that end, beside transmitter and receiver interfaces implementing the protocol, the development is foreseen of dedicated units

to concentrate data coming from different sources over a single, high-speed uplink and to transmit the TTC signals of a single downlink to several destinations; the concentrator could also have the function of organizing readout data on the basis of some characterization (e.g. time stamp), thus performing a first stage of event building inside the detector. Furthermore, for a given architecture the bandwidth requirement for each branch of the topology can be fulfilled by selecting the FF-LYNX link with the appropriate speed, or by grouping more low-speed links to reach the required total speed.

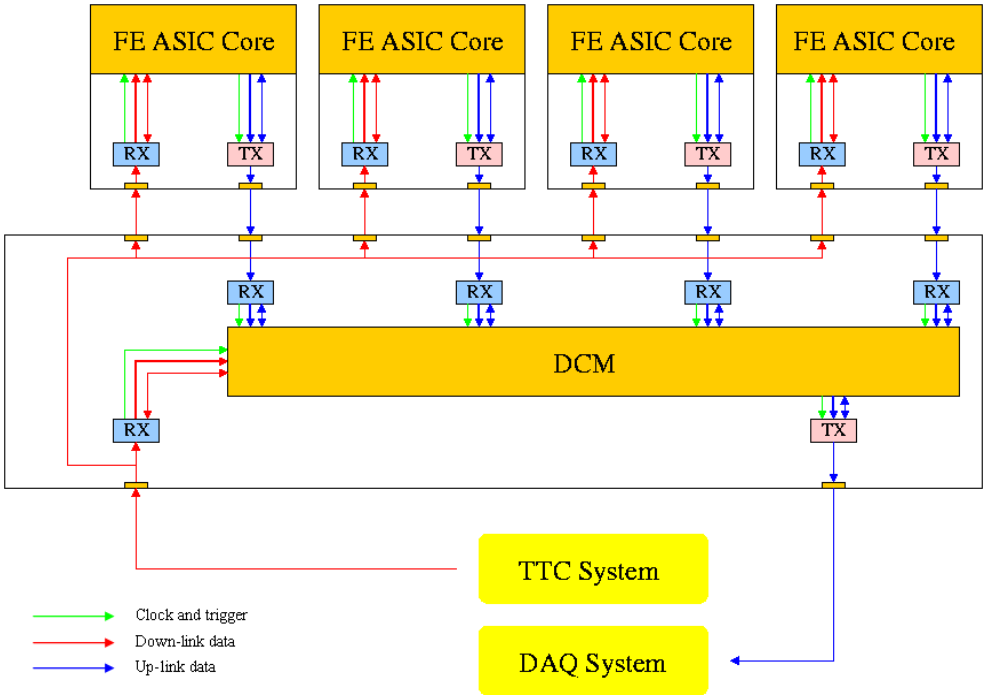


Fig. 1.7 – "Star" architecture: Front-End circuits are connected to a Data Concentrator Module (DCM) that merges data streams, optionally performing event building (i.e. aggregating data identified by the same time stamp), and distribute TTC signals to the connected Front-End circuits.

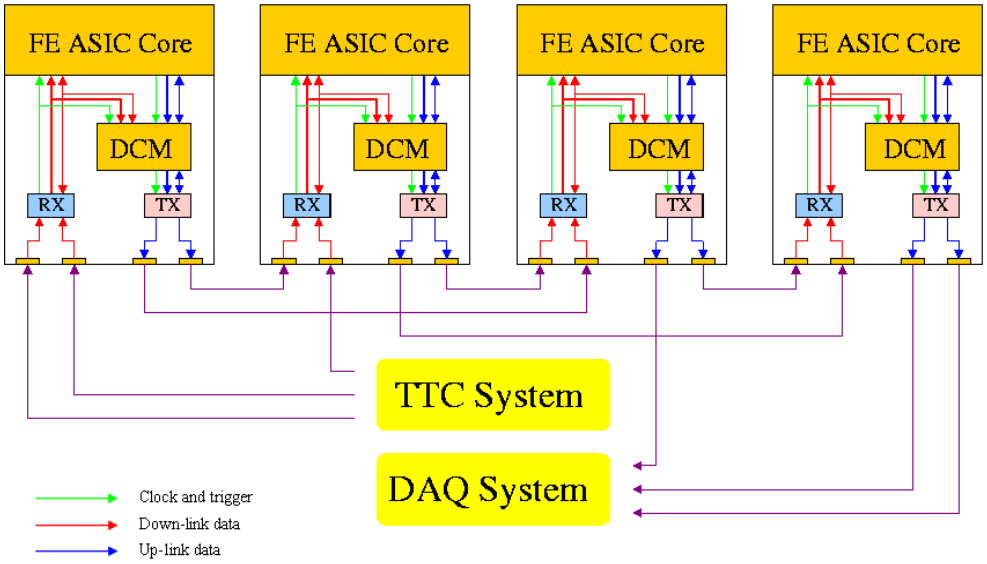


Fig. 1.8 – "Ring" architecture: Front-End circuits are daisy chained with redundant connections to provide robustness against component failures. TTC signals and data will propagate along the chain and the data merging will be distributed among the nodes of the chain (i.e.: each node will merge its own data with the data received from the previous nodes).

- **Compatibility with different link technologies**. The FF-LYNX protocol is defined at the data-link layer of the ISO/OSI protocol stack ,and is intended to be compatible with different technologies for the implementation of the physical layer, such as standard or “low power” LVDS (Low Voltage Differential Signaling)..

As for the protocol implementation, the goal is to design and develop low-power, radiation-tolerant interfaces, targeted to a standard CMOS technology and make them available to the designers of the integrated circuits for future HEP experiments as fully characterized and tested Intellectual Property (IP) cores, that will be parametric and partially configurable to ensure flexibility. The required tolerance to Total Ionization Dose (TID) and Single Event Effects (SEE) will be obtained by using CMOS standard technologies below 180 nm (that proved to be intrinsically robust to some TID effects, such as threshold voltage drift because of the reduced thickness of the oxide) and applying appropriate solutions at architectural, circuital and layout level (e.g. triple redundancy, hardened flip-flops). With respect to the use of custom rad-hard technologies, this approach offers the possibility to benefit from the high performance of standard processes (logic density, power consumption, speed, availability of pre-designed library cells) and yet to achieve the required radiation tolerance.

1.3 Methodology and design flow

As a first phase in the FF-LYNX project, a methodology was established dividing the design activity in successive phases with various intermediate verification steps, in order to validate the results obtained in each stage. A particular characteristic of this methodology is the use of a high-level software model of the link as a first stage of evaluation of the defined protocol, allowing the comparison of different choices about the various aspects of the protocol itself on the basis of conveniently chosen cost functions and figures of merit (e.g. bandwidth efficiency, robustness to errors, etc.). A diagram illustrating the project work flow is reported in Fig. 1.9; the main phases of the flow are here described:

- **Protocol definition**. Moving from the requirement analysis outlined in section 1.1, the first stage of the activity is the definition of the FF-LYNX protocol in a first, tentative version to be submitted to successive verification, and open to refinements following suggestions and requests from FF-LYNX collaborators in the field of HEP experiments (physicists and engineers involved in the design of detectors). This first version of the protocol is defined trying to meet the requirements about data rate, trigger latency and error robustness by means of a mix of custom and standard solutions typical of the data-link layer of the ISO/OSI model.
- **Validation in a high-level simulation environment**. A high-level software simulation environment is created that models the transmitter and receiver interfaces, implementing the defined protocol, and a surrounding test bench providing the expected stimuli and measuring various aspects of the link performance. This environment is realized in SystemC [20] and offers a highly configurable model of the link in order to allow the test, during the phases of protocol definition and refinements, of different versions of the protocol. To validate the protocol and allow comparisons between different versions, the basic task of the high-level simulator is the evaluation of specific cost functions and figures of merit (e.g. bandwidth efficiency, data loss rate, data latency) that has been defined on the basis of system requirements; furthermore, by evaluating these parameters the high-level simulator also gives valuable information about hardware aspects of the interfaces, such as optimum size of buffers.
- **Definition of the interfaces architecture**. After the phase of protocol validation, the architecture of the hardware interfaces implementing the first version of the protocol is

defined, following indications from the high-level simulation phase. The operation of the transmitter and receiver interfaces is divided into functional blocks (e.g. buffer, encoder, serializer and so on) in order to simplify the implementation and to allow the separate verification, in the successive HDL simulation phase, of the correctness of each sub-function.

- **VHDL modeling and simulation**. Once defined their architecture, a model of the interfaces using a Hardware Description Language (HDL) is created for functional simulation and successive synthesis. This model is built in a highly-parameterized form in order to allow quick changes to follow indications coming from the high-level simulator. Each block of the architecture is separately modeled and tested in a specific test bench through functional simulation to verify its functionality, so to progressively build an overall model that was verified in all its parts. Finally, the complete transmitter-receiver model is tested in a test bench including emulators of the transmitting and receiving hosts to verify the overall functionality of the system.
- **FPGA prototyping**. Next, a phase of FPGA-based emulation is foreseen, to provide additional verification of the system functionality (especially about the aspects that are difficult to assess in the software simulator, due to excessive simulation time: for example, the evaluation of the effect of transmission errors on data integrity with realistic bit error rates – in the order of 10^{-9} or less – requires some days of simulation) and to evaluate different solutions for the physical layer. The HDL models of the interfaces are synthesized on an FPGA together with the model of a surrounding test system that provides test vectors to the interfaces model and records the test results. The FPGA is mounted on a development board chosen so as to contain all the resources required for the test, such as memories, connectors, network interfaces etc. Furthermore, by choosing an FPGA with I/O ports in different standards (LVDS, LVCMOS, etc.), the operation of the protocol with different link technologies can be tested.
- **Test chip**. As a final phase, the design and realization of a test chip is foreseen to complete the verification and the characterization of the FF-LYNX interfaces. The chip will contain the transmitter and receiver interfaces (in all the different versions in terms of transmission speed) and the test bench architecture that has been already synthesized on the FPGA emulator, and will be realized in a commercial CMOS technology, below 180 nm, accessible to INFN through CMP, MOSIS or Europractice programs. The chip will be then tested in order to characterize the interfaces about electrical and thermal properties, and also radiation tests will be performed to evaluate tolerance to TID and SEE: the final goal is the creation of IP hardware macrocells, fully characterized and tested, to be made available to designers of ASICs for future High Energy Physics experiments for integration in their systems.

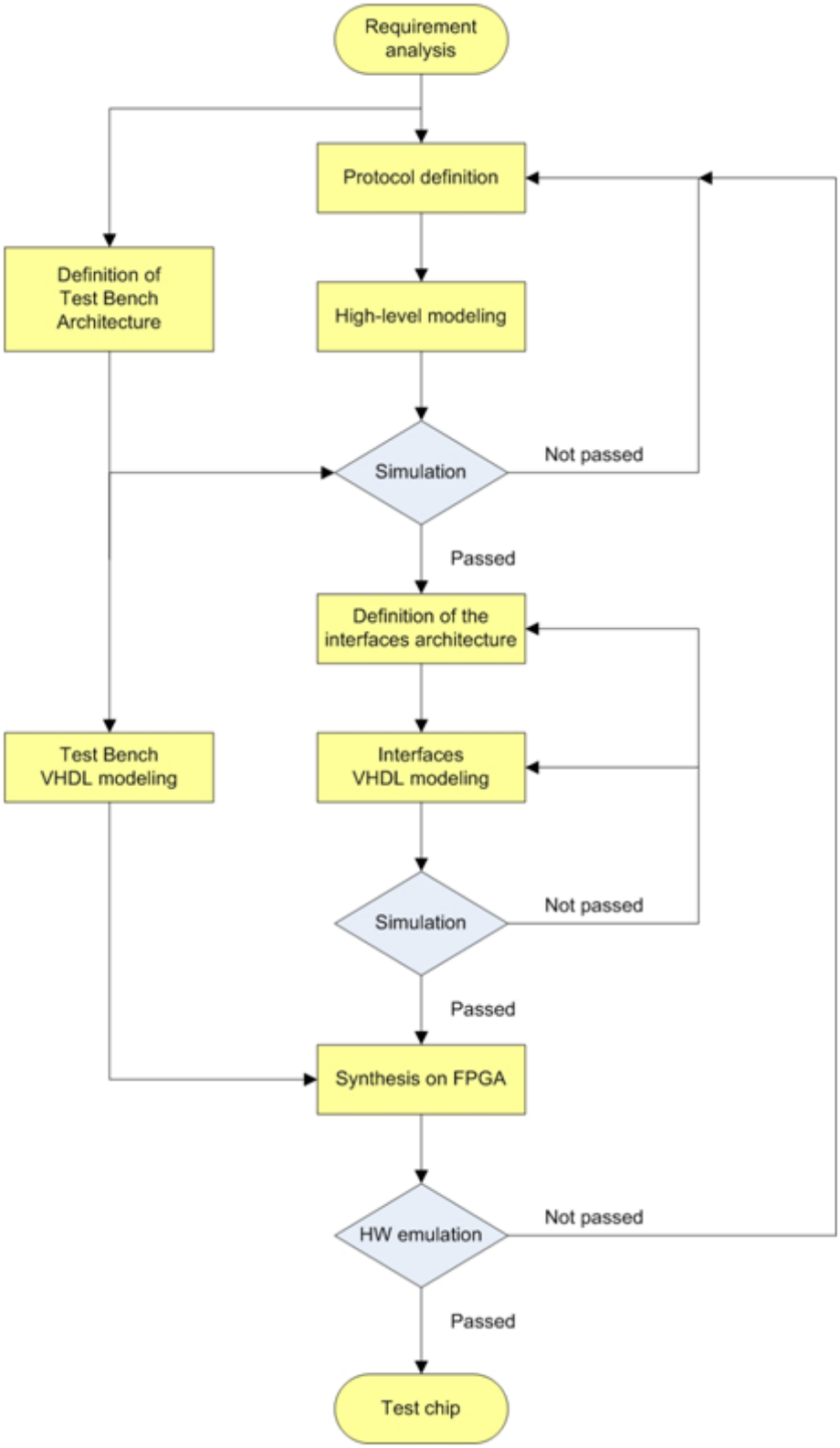


Fig. 1.9 – Project work flow diagram.

2 The FF-LYNX protocol

This chapter describes in detail the first version of the FF-LYNX protocol, called version 1 or v.1, i.e. the one that was tested in the high-level software simulator and was implemented in the VHDL model of the interfaces, which is the main subject of the present thesis work. The procedure of validation in the high-level simulator is then described and the results are reported, and finally the foreseen successive versions of the FF-LYNX protocol are outlined.

2.1 General characteristics

The first version of the FF-LYNX protocol (protocol v.1) was defined, as anticipated before, at the data-link layer of the ISO/OSI model, dealing therefore with three basic issues:

- stream multiplexing and synchronization: as will be explained in the following, two channels are multiplexed into the transmitted serial data stream by means of a Time Division Multiplexing (TDM) technique: the protocol must then define the way this multiplexing is performed by the transmitter and the demultiplexing is carried out by the receiver, acquiring the synchronization on the two channels of the received stream;
- framing: the transfer of user data (called payload) is carried out by means of information units called frames, that are sequences of bits in the stream with defined limits (start and end): the protocol has to specify how frames are built by the transmitter and how their beginning and ending in the stream are recognized by the receiver;
- error control: the protocol must define a way to detect and possibly correct transmission errors that affect critical data.

The FF-LYNX protocol v.1 was defined trying to satisfy some of the most basic requirements of TTC and DAQ systems for HEP experiments, starting from requisites about clock and trigger transmission.

First of all, a serial transmission was chosen to minimize the number of required wires, and a separate clock transmission scheme was preferred for the moment to simplify the receiver circuits. Therefore, in protocol v.1 the FF-LYNX physical link comprises two wires: a CLK line, carrying the transmission clock, and a DAT line that delivers serial data. The values foreseen for the transmission speed are 4, 8 and 16 times the frequency F of the reference clock in the FE electronics: considering for example the LHC scenario, where the master clock frequency F is 40 MHz, the transmission speed will be then 160, 320 or 640 Mbit/s: the CLK line hence will carry a 160, 320 or 640 MHz clock. These values are chosen to allow an easy reconstruction, in the receiver, of the reference clock (frequency F) from transmission clock received on the CLK line, as will be explained later. For ease of exposition, in the following the LHC value of 40 MHz is considered for the master clock frequency F, although it's understood that the protocol (and all the following considerations) is applicable to any value of F.

To transmit the trigger command onto the serial DAT stream, it must be encoded as a bit sequence of some length, and this must be done respecting the two fundamental requisites about trigger: robustness to transmission errors and constant latency between transmission and reception. Transmission errors due to noise or radiation can affect the digital stream altering one or more of the transmitted bits so that a transmitted logic '0' is received as a logic '1' or vice-versa (bit-flip): these errors on the link can occur as single events (i.e. bit errors occur "fairly distanced" one from the other, so that the probability of having more than an error in each sequence of consecutive bits of a certain length is negligible) or as bursts, i.e. more consecutive bits are corrupted: however, not to complicate things too much, in protocol v.1 only single bit-flips are considered, while the

protection against bursts will be addressed in successive versions of the protocol by means of interleaving techniques, as will be explained later on. Therefore, just one bit-flip is considered to possibly happen on each sequence of bits with which a trigger command is encoded. As for the constant latency requirement, this means that the sequence of bits encoding the trigger must be transmitted with highest priority, i.e. without waiting, every time a trigger command arrives to the transmitter interface: this can happen at every cycle of the 40 MHz master clock. Furthermore, the receiver must recognize the trigger sequence in the received serial stream with exact reconstruction of timing, i.e. the 40 MHz clock cycle of occurrence. If the trigger sequence is transmitted into the serial bit stream "embedded" with normal data, a possibility for addressing these requirements is to adopt a character-oriented protocol, that is a protocol in which data is transmitted as divided in fixed-length groups of bits called "characters": examples of this kind of protocol are Bisync (Binary Synchronous Communication) by IBM and PPP (Point-to-Point Protocol) [21]. In this approach, the trigger could be sent as a special character, providing that it can never occur in "normal" data. However, it must be guaranteed at the same time that the "trigger character" is recognized by the receiver even in case of bit-flip: this complicates this approach very much, since some mechanism must be foreseen to exclude the possibility of presentation in the normal data not only of the trigger sequence, but also of the sequences that results from the trigger one after the flipping of a bit.

To overcome these problems, the idea was that of reserving to trigger sequences a dedicated time-division channel, inside the DAT serial stream, that is free from other data transmission so that triggers can be encoded as bit sequences in a way that allows to satisfy the requirements about latency and error robustness. This reserved channel is obtained through time division multiplexing: as said before, considering a transmission speed of 160, 320 or 640 Mbit/s, 4, 8 or 16 bits are transmitted in the DAT line in each 40 MHz clock cycle. This way the line can be regarded as a 40 MHz parallel link and conveniently split into two logical channels: the THS channel, so called because it is reserved, as will be explained later, to the transmission of Triggers, frame Headers and Synchronization patterns, and the FRM channel that carries data frames.

The next step is to decide the number of bits that are reserved to the THS channel (here and in the following, "in each 40 MHz clock cycle" is understood) and, related to this, the length of the bit sequence that represents the trigger. To this end, the following considerations/constraints must be kept in mind:

- to minimize the bandwidth fraction that is subtracted to data transmission, it would be preferable to have a "narrow" THS channel. i.e. the number of bits for the THS channel should be as small as possible;
- the trigger sequence must last not more than three 40 MHz clock cycle, since this is the minimum interval between two consecutive trigger commands; this condition was chosen considering the CMS requirements, that appears to be the most restrictive regarding this aspect [14];
- even in case of (single) bit-flip, the trigger sequence must be always recognized by the receiver with correct timing: that is, if a corrupted trigger sequence arrives to the receiver at the 40 MHz clock cycle number i, the receiver must not detect a trigger sequence at the 40 MHz clock cycle number i−1 or i+1.

To satisfy at best the first requirement, one could think to assign just one bit to the THS channel: however this would imply, for the second constraint, that the sequence representing the trigger is just 3 bits long. But this is not sufficient to fulfill the last requirement: supposing for example that all zeroes are transmitted onto the THS channel in absence of triggers, and that the trigger sequence is 111, if ...0000110000... is received because of a bit-flip then it is not possible to say with certainty where is the beginning of the sequence (it could be ...000011<u>0</u>0000... or ...0000<u>1</u>10000... depending on whether the bit preceding the two 1s or the bit following them is considered to be corrupted); the same problem arise with all the other possible choices of the 3-bit sequence encoding the trigger. Therefore, it was decided to assign 2 bits to the THS channel, and consequently to encode the trigger as a 6-bit sequence: this allows to fulfill all the previously

mentioned requisites, as will be shown later on. Consequently, the FRM channel has 2, 6 or 14 bits in each 40 MHz clock cycle for a transmission speed of 160, 320 or 640 Mbit/s, respectively: this is schematically illustrated in Fig. 2.1 for the cases 160 and 320 Mbit/s:

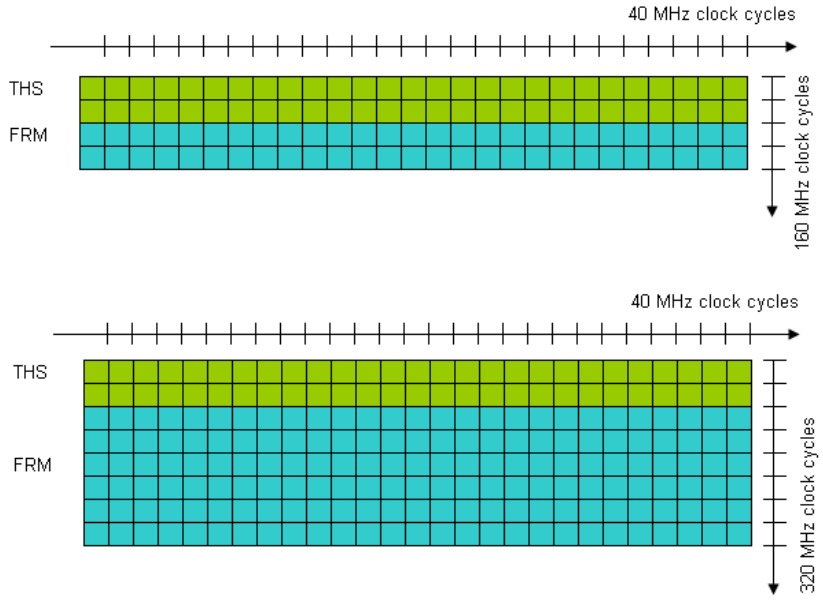


Fig. 2.1 – The division of the serial DAT stream in the THS channel and the FRM channel, in the cases of transmission speed equal to 160 and 320 Mbit/s.

The FRM channel is used instead for the transmission of frames, that carry general, low priority user data and whose beginning is marked by a header, that is a special sequence of bits. As will be explained in section 2.2.3, the 6-bit encoding used for trigger allows to represent up to three different patterns (in addition to the "no operation" sequence, i.e. the one that is transmitted when the channel is idle) with the desired properties of robustness to bit-flip: therefore, beside the trigger, also the frame header is encoded as one of these patterns and transmitted in the THS channel as well, thus reducing the framing overhead in the FRM channel; the third available sequence can be used to encode "sync" patterns, i.e. sequences that are transmitted to maintain synchronization of the receiver on the THS channel of the received stream. The THS channel indeed gets its name from the names of the patterns that are transmitted on it: Trigger, Header and Sync.

The operation of the THS channel, the synchronization mechanism and the structure of FF-LYNX frames are described in more detail in the following sections.

2.2 The THS channel

The THS channel is therefore a 2-bit wide sub-stream in the DAT serial bit stream, reserved for the transmission of trigger and header 6-bit sequences, that will be called in the following TRG and HDR; this is the basic function of the THS channel although, as will be explained in the following, in a preliminary version of the protocol also a third sequence, denoted as SYN, was foreseen to be transmitted into the THS channel, and used as a synchronization pattern for the receiver.

Two main issues arise about the function of the THS channel: TRG and HDR scheduling, i.e. the timing organization of the transmission of these two sequences so that the TRG has always a fixed latency between transmission and reception, and synchronization of the receiver, that is the mechanism by which the receiver interface is able to distinguish, in each 40 MHz clock cycle, the 2 bits that belong to the THS channel from the bits of the FRM channel. These two issues are treated

in the next two sections; after which, the encoding scheme chosen for THS sequences is described in section 2.2.3.

2.2.1 TRG and HDR scheduling

Since TRG and HDR sequences are both transmitted onto the THS channel, a method must be provided to avoid the overlapping of their transmissions. In fact, the commands for the transmission of TRG and HDR arrive from different sources (the trigger command arrives to the transmitter interface as an input from the host, while the beginning of a header transmission is decided by the interface itself when a data frame is ready to be sent) and a 6-bit TRG or HDR sequence occupies the channel for three consecutive 40 MHz clock cycles: therefore, if no further control is foreseen the possibility exists that TRG and HDR transmissions overlap in time. In addition, the method to avoid this must guarantee that TRG always has higher priority with respect to HDR, so that a trigger transmission command never has to wait the ending of the transmission of a HDR: this way, fixed latency for triggers is assured.

To guarantee the minimum interval of three 40 MHz clock cycles between consecutive TRG and HDR commands, respecting at the same time the fundamental requisite of constant latency for the TRG commands, the FF-LYNX transmitter employs the following strategy: a delay of three 40 MHz cycles is introduced between every input TRG command and the start of its transmission into the THS channel, thus creating a temporal "observation window" that can be used to determine the intervals in which the starting of the transmission of a HDR sequence is not allowed, because it would overlap with an upcoming TRG sequence or with a past TRG transmission that hasn't been completed yet. More in detail, given the (40 MHz) clock cycle of starting of a TRG sequence and considering that TRG and HDR sequences occupy 3 clock cycles, there is a "forbidden interval" of 5 clock cycles in which the beginning of a HDR sequence's transmission would cause a "collision", as illustrated in Fig. 2.2:

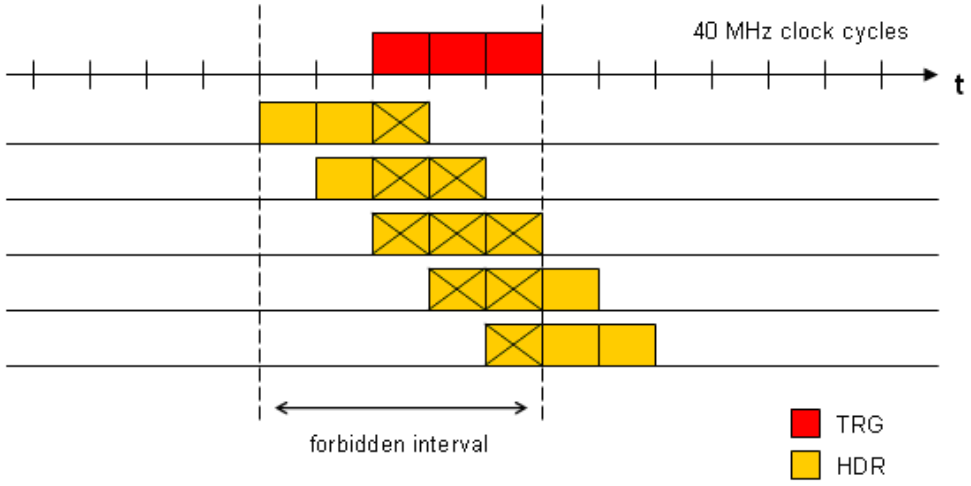


Fig. 2.2 – TRG and HDR scheduling: the transmission of a TRG sequence (shown in red) creates a 5-clock-cycles "forbidden interval" during which the starting of a HDR transmission can't be allowed, because it would cause the two sequences to overlap (as represented by crossed blocks).

Therefore, when a trigger command arrives, the transmitter must schedule the TRG transmission for three (40 MHz) clock cycles later and start a forbidden window of 5 clock cycles, delaying the transmission of any possible HDR until the end of this window. More details about the hardware implementation of this scheduling mechanism will be given in section 4.2.4.

2.2.2 Synchronization

To properly receive the transmitted triggers and data, and also to reconstruct correctly the 40 MHz clock from the transmission clock, the receiver has to synchronize on the THS channel, i.e. to distinguish, in the received serial bit stream, the bits that belong to the THS channel from the ones that belong to the FRM channel. The receiver has to acquire this synchronization at the beginning of the transmissions (sync lock); but then, during transmission, it is possible that sync is lost due to transmission errors on the CLK line (that cause the non-recognition by the receiver of one or more edges of the transmission clock, or the detection of spurious edges) so that the receiver is no more correctly aligned on the THS channel, thus generating a reconstructed 40 MHz clock that is not in phase with the 40 MHz master clock feeding the transmitter and not being able to correctly detect THS sequences and deserialize data. In such a case, the sync mechanism must detect the loss of synchronization, abandon the previous alignment (sync unlock) and acquire the new sync lock.

The mechanism that was chosen for synchronization is based on THS sequences recognition, and works as follows. In the received serial bit stream all the possible 2-bit channels are considered; these are 4, 8 or 16 in case of transmission speed equal to 160, 320 or 640 Mbit/s respectively, and are distinguished one from the other on the basis of the position of the two considered bits among the 4, 8 or 16 bits that are transmitted in each 40 MHz clock cycle: this is illustrated in Fig. 2.3 for the example of 320 Mbit/s:

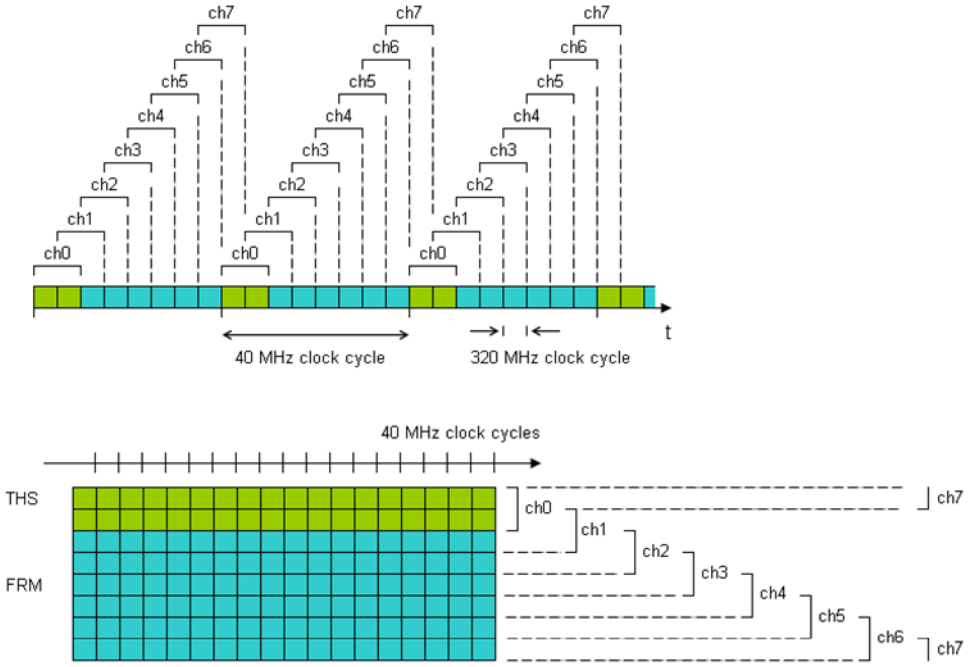


Fig. 2.3 – The eight 2-bits channels that can be identified in the 320 Mbit/s serial stream.

For each of this channel, the receiver looks for transmitted THS sequences (TRG or HDR) and counts them: every time a THS sequence is detected in a channel, the count for that channel is incremented. When the count for one of the channels, say channel i, reaches a threshold N_{lock}, the sync is declared as acquired (sync lock) and channel i is considered to be the THS channel (let's say that channel i has become the channel "in charge"); the count for all the other channels is reset, and the reconstructed 40 MHz clock is generated by the receiver with a phase that is aligned with channel i. From the moment of sync lock, the counting of THS sequences goes on in all the channels, but every new THS sequence detected on the channel in charge resets the count for all the other channels. If instead the count for a channel that is not in charge reaches a second threshold

N_{unlock}, the sync is declared lost, and a new sync lock is done on the first channel that reaches the locking threshold N_{lock}. The synchronization algorithm state diagram is reported in Fig. 2.4.

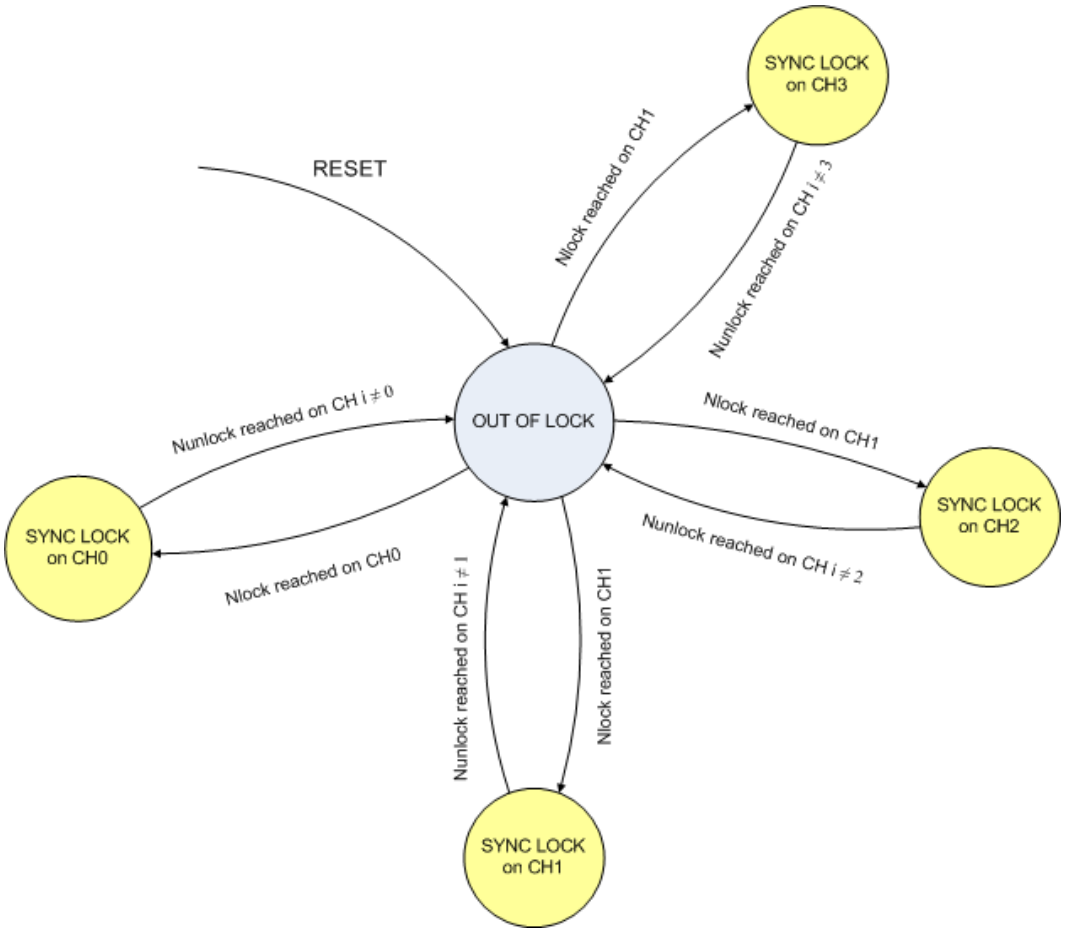


Fig. 2.4 – Synchronization algorithm.

Obviously, even when the receiver is correctly synchronized on the THS channel, it is possible that "fake" THS sequences appear in other channels, but statistically they will appear less frequently than real THS sequences in the real THS channel, and so this mechanism allows the receiver to maintain the correct sync. However, to this end it must be provided that THS sequences are sent into the real THS channel with sufficient frequency, since otherwise the receiver will lock on some other channel due to spurious THS sequences detected on it. For this reason SYN patterns were introduced: they are sent by the transmitter into the THS channel when no TRG or HDR transmission has occurred for a certain period, so that the THS channel is never free from transmissions for a too long time. Of course, SYN transmissions are scheduled by the transmitter interface with lowest priority with respect to TRG and HDR.

If instead during transmission the receiver goes out of sync, THS sequences will begin to be counted in the channel that is now the THS one, and so this mechanism allows the receiver to recover the correct synchronization.

In conclusion, the receiver uses TRG, HDR and SYN sequences as synchronization patterns. As will be explained in the next section, once synchronization is acquired the receiver detects TRG and HDR sequences in the THS channel also if they are corrupted by a single bit-flip; for synchronization, instead, only "pure" (i.e. not corrupted by bit-flips) TRG, HDR and SYN sequences are considered, in order to limit the number of spurious sequences that are detected in other channels.

2.2.3 The THS encoding

A custom encoding scheme, called THS encoding, was chosen to represent THS commands (TRG, HDR and SYN) as 6-bit sequences to be transmitted on the THS channel with desired characteristics of robustness to transmission errors (bit-flip). It is here described after a brief introduction on error control coding theory.

2.2.3.1 Elements of coding theory

Error control coding is a branch of information theory that deals with methods of delivering information from a source to a destination reducing at a minimum the errors that are introduced by noise on the channel. A communication channel can be schematized as in Fig. 2.5:

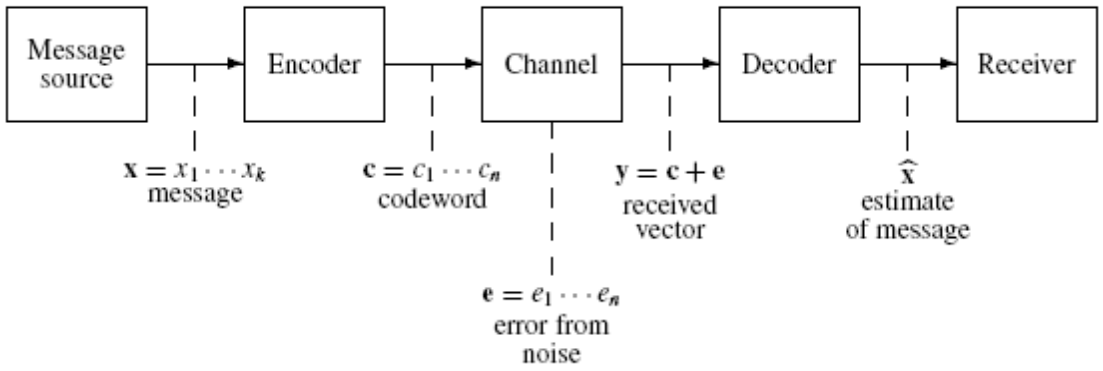


Fig. 2.5 – Model of a communication channel.

The source (transmitter) has to send a message **x**, i.e. a sequence of symbols x_1, ... , x_k. If no modification is made to the message and it is transmitted directly over the channel, any noise would make the message unrecoverable by altering some of its symbols. The basic idea of error control coding is to add some redundancy to the message so that hopefully the received message is the original message that was sent. The redundancy is added by the encoder and the modified message, called a *codeword* (**c** in the figure), is sent over the channel where noise in the form of an error vector **e** distorts the codeword producing a received vector **y**. The received vector is then sent to the decoder that removes errors and strips off the redundancy, and produces an estimate $\hat{\mathbf{x}}$ of the original message: if the error control scheme has worked well, it will be $\hat{\mathbf{x}} = \mathbf{x}$.

Definition of **code**. Let $\mathbb{F}_q$ be a finite field of q elements, for example the binary field $\mathbb{F}_2 = \{0, 1\}$, and let $\mathbb{F}_q^n$ with $n > 0$ be the vector space of all n-tuples over the finite field $\mathbb{F}_q$. $\mathbb{F}_q$ is also called a *finite alphabet of q letters* (or symbols), and each vector of $\mathbb{F}_q^n$, i.e. a n-tuple of letters of $\mathbb{F}_q$, is called a *word* over the alphabet $\mathbb{F}_q$. A *code C* over $\mathbb{F}_q$ (*q-ary code*) is any non-empty subset of $\mathbb{F}_q^n$; more precisely, a *(n, M) code C* over $\mathbb{F}_q$ is a subset of $\mathbb{F}_q^n$ of size M, i.e. containing M elements (vectors) called *codewords*.

A *binary code* is thus a code over $\mathbb{F}_2$, and its codewords are sequences of n bits. In the binary field $\mathbb{F}_2$ addition (+) and multiplication (·) are defined by the following tables:

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

the $\mathbb{F}_2$ addition is thus a modulo-2 addition, or XOR operation.

Definition of **distance** and **weight**. Consider the set $\mathbb{F}_q^n$ of all the words of n letters over the alphabet $\mathbb{F}_q$. The *Hamming distance* (or *distance* tout court) d(**x**, **y**) between two words **x**, **y** $\in\mathbb{F}_q^n$ is defined as the number of position in which **x** = (x_1, x_2, ... , x_n) and **y** = (y_1, y_2, ... , y_n) have different letters:

$$\mathrm{d}(\mathbf{x},\mathbf{y}) = \left|\{i : x_i \neq y_i; 1 \leq i \leq n\}\right| \tag{2.1}$$

where | . | denotes the number of elements of the set inside. It can be easily verified that d(**x**, **y**) is a metric on the vector space $\mathbb{F}_q^n$.

The smallest distance between two distinct words of a code is called the *minimum distance* of the code; a (*n*, *M*) code over $\mathbb{F}_q$ with minimum distance *d* is called a (*n*, *M*, *d*) code over $\mathbb{F}_q$.

The *(Hamming) weight* wt(**x**) of a word **x** $\in\mathbb{F}_q^n$ is the number of nonzero letters in **x**.

From the definitions of distance and weight it follows immediately that d(**x**, **y**) = wt(**x** – **y**) and wt(**x**) = d(**x**, 0).

Definition of **sphere**. Given a **x** $\in\mathbb{F}_q^n$ and a positive integer *r*, the *sphere* of radius *r* centered at **x** is defined to be the set

$$S_r(\mathbf{x}) = \left\{\mathbf{y} \in \mathbb{F}_q^n : d(\mathbf{x},\mathbf{y}) \leq r\right\} \tag{2.2}$$

Linear codes. If *C* is a *k*-dimensional subspace of $\mathbb{F}_q^n$, then *C* is said to be a *linear code* of length *n* and dimension *k* (with $k \leq n$) over $\mathbb{F}_q$, and is denoted as a [*n, k*] code; the number of its codewords is q^k. From the definition it follows immediately that a linear code always comprises the zero word, i.e. the word whose each letter is the zero element of $\mathbb{F}_q$.

A [*n*, *k*] code with minimum distance *d* is called a [*n*, *k, d*] code. Since d(**x**, **y**) = wt(**x** – **y**), in a linear code *C* the minimum distance is the minimum weight of the non-zero codewords of *C*.

Given a [*n, k*] code *C*, any $k \times n$ matrix *G* whose rows are a basis of the vector subspace *C* is called a *generator matrix* for the code.

For any set of *k* independent columns of a generator matrix *G*, the corresponding set of coordinates forms an *information set* for *C*; the remaining $r = n - k$ coordinates are called a *redundancy set* and *r* is called the *redundancy* of *C*. For example, in a [*n, k*] binary code each codeword will have k information bits and $r = n - k$ redundancy bits.

A generator matrix *G* is said to be in *standard form* if

$$G = \left[I_k \middle| A \right] \tag{2.3}$$

where I_k is the $k \times k$ identity matrix and A is a $k \times (n - k)$ matrix. Therefore, if G is in standard form, an information set of the code is formed by the first k coordinates.

Two $[n, k]$ codes C_1 and C_2 are said to be *equivalent* if one can be obtained from the other through a finite number of operations of the following kind:

- permutation of the coordinates, i.e. a given permutation of the n coordinates is applied to all the codewords;
- multiplication of all the symbols in a given position of the codewords for a non-zero element of $\mathbb{F}_q$.

Passing from a code C_1 to an equivalent code C_2 many properties of are maintained, in particular the weights of the codewords and hence the minimum distance. In this sense, two equivalent codes are essentially the same code.

It easily follows from the definition of equivalence that, if C_1 and C_2 are equivalent codes, then a generator matrix G_2 of C_2 can be obtained from a generator matrix G_1 of C_1 through a finite number of operations of the following kind:

- permutation of the columns;
- multiplication of a column for a non-zero element of $\mathbb{F}_q$;

and also of the following kind:

- permutation of the rows;
- multiplication of a row for a non-zero element of $\mathbb{F}_q$;
- sum of a row with a scalar multiple of another row;

since these three last operations simply correspond to a change of the basis of the subspace.

From these considerations, it also follows that any generator matrix of a code C can be brought into standard form, i.e. transformed into a standard form matrix that generates a code C' equivalent to C.

For a $[n, k]$ code C, a $(n - k) \times n$ matrix H defined by

$$C = \left\{ \mathbf{x} \in \mathbb{F}_q^n : H\mathbf{x}^{\mathbf{T}} = 0 \right\} \tag{2.4}$$

is called a *parity check matrix* for C (note that here, as will be in the following, the vectors are intended as row vectors). In other words, a parity check matrix for a linear code C is a $(n - k) \times n$ matrix H such that C is the kernel of H. Since from linear algebra it is known that, for every $m \times n$ matrix A,

$$\dim\{\ker(A)\} = n - \operatorname{rank}(A) \tag{2.5}$$

then it follows that

$$\operatorname{rank}(H) = n - k \tag{2.6}$$

that is, the rows of H are independent. From the definition of H it also follows immediately that, for any generator matrix G and for any parity check matrix H of a code,

$$GH^{\mathrm{T}} = HG^{\mathrm{T}} = 0 \tag{2.7}$$

because the rows of G are codewords; and using this result, if $G = \left[I_k \middle| A \right]$ is a generator matrix in standard form of a code C, it can be easily verified that a parity check matrix H for C is

$$H = \left[-A^{\mathrm{T}} \middle| I_{n-k} \right] \tag{2.8}$$

Encoding and decoding. Given a [*n, k*] code C, the transformation of each k-letter vector $\mathbf{x}$ of $\mathbb{F}_q^k$, called *message*, into a n-letter codeword $\mathbf{c}$ of C is called *encoding*. For linear codes, the usual encoding method is to encode a message $\mathbf{x}$ as the codeword

$$\mathbf{c} = \mathbf{x}G \tag{2.9}$$

In particular, if G is in standard form it will result that:

$$\mathbf{c} = (c_1, ..., c_n) = \mathbf{x}G = \mathbf{x}\left[I_k \middle| A \right] = \left[\mathbf{x} \middle| \mathbf{x}A \right] = (x_1, ..., x_k, c_{k+1}, ..., c_n) \tag{2.10}$$

that is, the message $\mathbf{x}$ appears unchanged in the first k coordinates of $\mathbf{c}$, while the remaining $n - k$ coordinates c_{k+1}, ... , c_n are the redundancy symbols added by the code.

When using an error control method to transmit information from a sender to a receiver, a message $\mathbf{x}$ is first encoded as a codeword $\mathbf{c}$, and then $\mathbf{c}$ is sent over the communication channel. The noise in the channel has the effect of changing some symbols of the sent codeword $\mathbf{c}$, so that the receiver gets an altered word $\mathbf{y} \in \mathbb{F}_q^n$

$$\mathbf{y} = (y_1, ..., y_n) = (c'_1, ..., c'_n) \tag{2.11}$$

where c'_i denotes the i-th symbol of $\mathbf{c}$ possibly altered by an error. This received vector can be seen as $\mathbf{y} = \mathbf{c} + \mathbf{e}$ where $\mathbf{e}$ is an error vector, with non-zero elements in the coordinates corresponding to the altered symbols. The *decoding process* is that of reconstructing the transmitted codeword $\mathbf{c}$, and hence the message $\mathbf{x}$, from the received $\mathbf{y}$: the task of the decoder is thus to find the error vector $\mathbf{e}$ so that it can find $\mathbf{c}$ by performing $\mathbf{c} = \mathbf{y} - \mathbf{e}$.

A common way to perform the decoding process (because it can be proved that, under certain hypotheses [23], it maximizes the probability of reconstructing the correct message) is the so called *nearest neighbor decoding*, which consists in finding the codeword $\mathbf{c}$ that has the minimum Hamming distance from the received vector $\mathbf{y}$; equivalently, an error vector $\mathbf{e}$ must be found with minimum weight such that $\mathbf{y} - \mathbf{e}$ belongs to the code.

Let's now consider the spheres of a given radius t centered at the codewords of the code. If t or less errors occur in the transmitted codeword $\mathbf{c}$ (i.e. the error vector e has t or less non-zero components) then the received word $\mathbf{y}$ will belong to the sphere of radius t centered at $\mathbf{c}$; and if in addition all the spheres are pair-wise disjoint, i.e. no vector of $\mathbb{F}_q^n$ belongs to more than a sphere, then a single codeword exists with minimum distance from $\mathbf{y}$: that is the center of the sphere that $\mathbf{y}$ belongs to. This means that under these conditions nearest neighbor decoding will correctly decode the received vector as $\mathbf{c}$.

Since it can be easily proved [23] that for a code C with minimum distance d the spheres of radius

$$t = \left\lfloor (d-1)/2 \right\rfloor \tag{2.12}$$

centered at the codewords of C are pair-wise disjoint, then we can conclude that a code C with minimum distance d can correct (through nearest neighbor decoding) up to t errors in each received word, with $t = \lfloor (d-1)/2 \rfloor$: the code is said to be a *t-error-correcting* code. If on the contrary more than t errors hit a transmitted codeword **c**, then a t-error-correcting code can fail in reconstructing the correct **c**, since the received vector **y** can fall out of the sphere of radius t centered at **c**.

When a vector **y** is received, to realize nearest neighbor decoding with a t-error-correcting code the most obvious way would be to examine all codewords until one is found with distance t or less from **y**: however, this method is clearly impractical if the number of codewords is large. A more efficient method to implement nearest neighbor decoding is the so called *syndrome decoding*. If H is a parity check matrix for a code C, then the *syndrome* **s** of a vector $\mathbf{y} \in \mathbb{F}_q^n$ with respect to H is defined as

$$\mathbf{s}(\mathbf{y}) = \mathbf{y}H^{\mathbf{T}} \tag{2.13}$$

hence, **s** is a vector of $\mathbb{F}_q^{n-k}$.

Supposing that a codeword **c** is transmitted, and a vector **y** = **c** + **e** possibly affected by a number of errors comprised in the correcting capability of the code is received, syndrome decoding consists in calculating the syndrome of the received vector **y**: if **y** belongs to the code C, i.e. there were no errors in the transmission so that **y** is equal to the transmitted codeword **c**, then it will be $\mathbf{s}(\mathbf{y}) = H\mathbf{y}^{\mathbf{T}} = 0$ from the definition of H. If instead an error vector **e** ≠ 0 has affected **y**, then it will be

$$\mathbf{s}(\mathbf{y}) = \mathbf{s}(\mathbf{c} + \mathbf{e}) = \mathbf{s}(\mathbf{e}) \tag{2.14}$$

that is, a syndrome will result that depends only on the error vector. It can be proved [23] that if $\mathbf{y} \notin C$ the syndrome **s**(**y**) is always non-zero, and that from **s**(**y**) it is possible to find **e** as the minimum weight error vector such that **y** − **e** belongs to the code, thus performing nearest neighbor decoding and reconstructing correctly the transmitted codeword as **c** = **y** − **e**.

The advantage of syndrome decoding over the straightforward method of examining all the codewords is that only q^{n-k} possible values of the syndrome must be scanned to find the corresponding **e**: this could be done for example with a table of $n - k$ entries, that would instead require n entries if one would have to find the codeword at minimum distance from the knowledge of **y**.

Hamming codes. For the sake of simplicity, in the following the focus will be restricted to binary codes, although the result can be extended to the general case of codes over $\mathbb{F}_q$.

Let r be an integer ≥ 2, and let $n = 2^r - 1$. Then let H be the $r \times n$ matrix whose columns are the numbers 1, 2, ... , $2^r - 1$ in binary representation: for example, if $r = 3$ and hence $n = 7$, H will be

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

A matrix H constructed in this way will have rank equal to r, and so it is the parity check matrix of a $[n = 2^r - 1,\ k = n - r]$ binary code. Any permutation of the columns of H produces an equivalent code, and therefore any of these codes is called the *binary Hamming code of length* $n = 2^r - 1$, and is denoted as H_r or $H(n,k)$. A $H(n,k)$ binary code is thus made up of 2^k n-bit codewords, each carrying k bits of information and r bits of redundancy. For example, the H matrix shown

above gives the H(7,4) binary Hamming code, that encodes 4 bits of information into a 7-bit codeword.

Two important facts about Hamming codes can be proved [23]:

- the minimum distance of a Hamming code, regardless of the length n, is equal to 3;
- any $[2^r - 1, 2^r - 1 - r, 3]$ binary code is equivalent to the binary Hamming code H_r.

The fact that $d = 3$ implies, referring to equation (2.12), that Hamming codes are 1-error-correcting codes. Hamming codes have thus a limited correction capability, but have the advantage of a relatively simple hardware implementation and are therefore employed in many application where hardware complexity is an important issue, such as error protection in RAM memories. Anyway, as will be shown in the following, Hamming codes can be *extended* (as every other code) by adding an extra symbol (a bit in the binary case) to every codeword in such a way that the minimum distance of the code is increased from 3 to 4, thus providing the extended code with the additional capability of detecting (but not correcting) also double errors in each word. Such a code is called a SECDED (Single Error Correction, Double Error Detection) code.

2.2.3.2 TRG/HDR/SYN encoding

In the approach that was initially adopted for the THS encoding, an encoding with the abovementioned characteristics of error robustness was sought for three THS sequences (TRG, HDR and SYN) under the hypothesis that the NOP (no operation) sequence was an all-zero sequence: i.e. all zeroes are transmitted into the THS channel when neither TRG nor HDR are being transmitted. The procedure for finding such an encoding was the following.

Two commands (TRG and HDR) or possibly three (TRG, HDR and SYN) must be encoded in 6 bits, belonging to 3 successive cycles of the 40 MHz clock; these commands are transmitted in the THS channel, which carries all '0' when no command is being transmitted. Hence the data stream, for example in the case of transmission speed = 160 Mbit/s, appears as in Fig. 2.6:

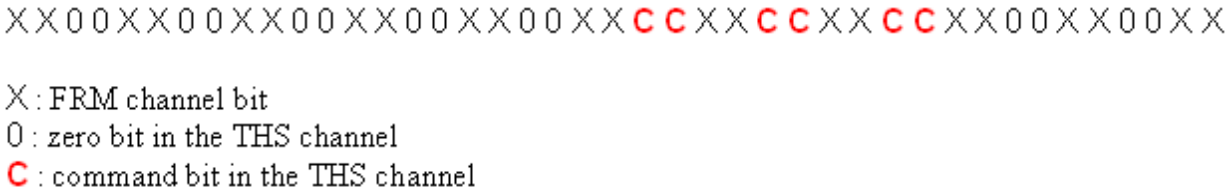


Fig. 2.6 – Transmission of a THS command in the THS channel.

Supposing that we will be able to encode 3 commands with the desired requirements, let's represent the codewords associated to these commands with the following notation:

Codeword 1 : $\alpha_1\ \beta_1\ \gamma_1$
Codeword 2 : $\alpha_2\ \beta_2\ \gamma_2$
Codeword 3 : $\alpha_3\ \beta_3\ \gamma_3$

where each α_i , β_i , γ_i is a 2-bit "character" of the set {00, 01, 10, 11}.

The command encoding must satisfy the following requirements:

- Condition 1 : the commands must not be confused one with another and must be distinguishable from the "zero" sequence 000000 (i.e. absence of command) if a (single) bit-flip occur in the 6-

bit sequence received. In other words, each sent command (including the sequence 000000 meaning "no command") must be exactly recognized by the receiver even if one of its 6 bits is inverted.
This means that, given a certain command cmd1 encoded for example as 100101, the receiver must recognize as cmd1 all the 6-bit sequences that differ from cmd1 by 1 bit at most, i.e.:

100101
100100
100111
100001
101101
110101
000101

Thus, using the coding theory terminology, the receiver recognizes as cmd1 all the 6-bits words that have a distance of 1 or less from the cmd1 word 100101. The same will happen with the other codewords: the receiver detects as codeword **x** all the words falling into the sphere of radius r = 1 centered at **x**. This in turn requires that the chosen code (i.e. the set of the two or three commands/codewords and the zero codeword 000000) has a minimum distance of 3, so that all the spheres of radius 1 centered at each codeword are pair wise disjoint, i.e. a word belonging to two or more different spheres does not exist: this guarantees that if a single bit-flip occurs in a transmitted codeword, the received word is always correctly interpreted as the original codeword. This condition can thus be summarized as the following:

$$\text{Condition 1:} \quad d(\alpha_i\, \beta_i\, \gamma_i\, ,\, \alpha_j\, \beta_j\, \gamma_j) \geq 3 \text{ with } i, j \in [0,3]$$

where $\alpha_0\, \beta_0\, \gamma_0$ is the zero codeword 00 00 00.
Note that with this approach the words at distance 2 from all the codewords are ignored by the receiver (that is, the double bit error in a codeword is detected but not corrected in this case), but a double bit-flip in a codeword can move the resulting received word in the sphere of another codeword, leading to a wrong interpretation of the received sequence: therefore, it must be guaranteed that a maximum of one bit-flip can occur in each 6-bit received word.

- Condition 2 and 3 : a possible bit-flip in the stream of 00 characters (in the THS channel) preceding a command must not cause the detection of a command in a wrong position of the THS stream. That is, referring to Fig. 2.7: if a bit-flip occurs in the green 00 characters preceding the command CC CC CC, the sequences 00 00 CC (seq 2) and 00 CC CC (seq 3) must never be recognized as valid codewords because this would cause the receiving of a command that differs from the transmitted one in type and/or timing (i.e. the 40 MHz clock cycle at which the command must have effect).

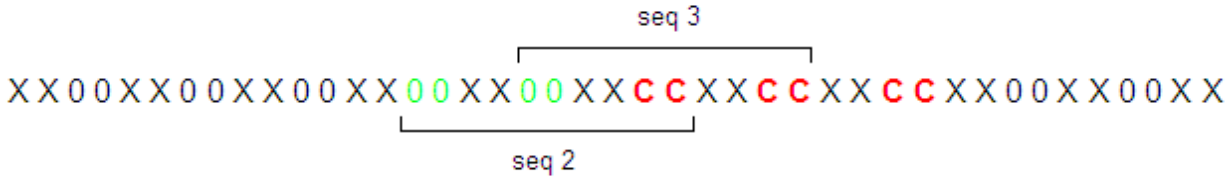


Fig. 2.7 – Sequences that could be recognized as THS commands.

The conditions 2 and 3 are therefore:

$$\text{Condition 2:} \quad d(00\ 00\ \alpha_i\, ,\, \alpha_j\, \beta_j\, \gamma_j) \geq 3 \text{ with } i, j \in [0,3]$$

Condition 3: $d(00\ \alpha_i\ \beta_i\ ,\ \alpha_j\ \beta_j\ \gamma_j) \geq 3$ with $i, j \in [0,3]$

The sequences CC CC 00 and CC 00 00 following a command are not an issue because when the receiver detects a codeword it is then insensitive to other commands for the next three 40 MHz clock cycles, since commands cannot overlap.

When beginning to search for a code that meet these requirement, starting from condition 1, we might first of all wonder how many codewords can be represented with a 6-bit code, satisfying the condition of minimum distance 3. To that end, the coding theory offers a number of upper bounds to the quantity $A_q(n, d)$, defined as the maximum number of codewords that a q-ary code of length n and minimum distance d can have:

$$A_q\left(n,d\right) = \max\left\{M : \text{an } (n,M,d) \text{ code over } \mathbb{F}_q \text{exists}\right\}$$

with the disadvantage that all these bounds represent only necessary conditions to the feasibility of a code with desired parameters n, M, d, without guaranteeing its actual existence; nevertheless, it's worth to check one or more of these bounds to verify that we are not looking for an impossible result. A bound that in most cases proves more stringent (and hence precise) than others is the *Plotkin bound*, which for the case q = 2, d odd and n < 2d + 1 (as in the present case) states that

$$A_2\left(n,d\right) \leq 2\left\lfloor \frac{d+1}{2d+1-n} \right\rfloor$$

therefore, for the present case n = 6 and d = 3, the condition is $A_2(6, 3) \leq 8$, that leaves an open possibility to represent 4 codewords (the zero sequence + 3 commands) using a 6-bit code with d = 3.
A way to satisfy easily the condition of minimum distance d = 3 (condition 1) is to choose a 6-bit binary "shortened" Hamming code. Since we need (possibly) 4 codewords, we can consider the [7, 4, 3] binary Hamming code (obtained by choosing r = 3, while with r = 2 we have a [3, 1 ,3] code with only two codewords), that is constructed by adding 3 parity bits (p_1 , p_2 , p_3) to 4 information bits (d_1 , d_2 , d_3 , d_4) with the following rule:

$p_1 = d_1 + d_2 + d_4$
$p_2 = d_1 + d_3 + d_4$
$p_3 = d_2 + d_3 + d_4$

(where "+" indicates the $\mathbb{F}_2$ addition, i.e. the modulo-2 addition or XOR operation); thus obtaining $2^4 = 16$ codewords that have with minimum distance 3 between each other, being a Hamming code. As we must use only 6 bits, we can *shorten* the 7-bit Hamming code now described. Shortening of codes will be described in more detail in section 2.3.1.1: here it suffices to say that the 6-bit binary shortened Hamming code is the code that is obtained from the 7-bit Hamming code by removing one information bit from the codewords, thus leaving 3 information bits (d_1 , d_2 , d_3) and 3 parity bits (p_1 , p_2 , p_3) following the rule

$p_1 = d_1 + d_2$
$p_2 = d_1 + d_3$
$p_3 = d_2 + d_3$

It can be proved (see section 2.3.1.1) that this code has minimum distance equal to 3, as the original H(7,4) code.

As the condition 1 can be satisfied with this choice, the next step is to find, among the eight words of the 6-bit shortened Hamming code, one or more set of four codewords (including the zero word 000000) that satisfy the conditions 2 and 3. Since no practicable analytical solution to this problem appears readily (the required conditions lead to a system of 18 non-linear equations in the variables α_1 β_1 γ_1 , α_2 β_2 γ_2 , α_3 β_3 γ_3) the remaining solution is to try the conditions 2 and 3 on every combination of 3 non-zero words out of the above code: furthermore, every permutation of this code should be tried.

The resulting enormous number of trials that must be done dictates the use of a calculator, and in this point of view it is preferable to try generic 6-bit sequences, abandoning the limitation to the Hamming code to explore also the possibility of non-linear codes: this means of course that the condition 1 on d = 3 must be verified again for every examined set of three 6-bit words.

Therefore, a simple program in C language has been written to find the desired code; the execution of the program returns the following two threesomes of codewords:

01 10 10	10 01 01
11 01 11	11 10 11
11 11 00	11 11 00

It can be easily verified that both satisfy all the requirements illustrated above. Any possible performance difference (false command detections, difficulty of synchronization, …) between these two codes, that are equally valid regarding the robustness against single bit-flip, must be examined through simulations.

The above program was also modified to seek possible sets of 4 words satisfying the three conditions, but it produced no result: we can therefore conclude that the maximum number of commands that can be encoded in 6 bits complying with the above conditions 1, 2 and 3 is three.

2.2.3.3 TRG/HDR balanced encoding

The one just described was the very first version of the THS encoding, that turned out as a first attempt of finding an encoding with the desired features. However, in the FF-LYNX protocol v.1 a modification of this encoding has been adopted, with better synchronization performance and properties of balancing and frequent bit transitions.

The starting point for this modification was the consideration that, in the foreseen “single wire” version of the protocol (i.e. clock and data are transmitted on a single wire, instead than the two wires CLK and DAT of the current version v.1), a clock recovery circuit will be needed in the receiver to extract the bit timing from the received stream. For reliable operation of this clock recovery circuit, a minimum frequency of transitions $0 \rightarrow 1$ or $1 \rightarrow 0$ in the received bit stream must be guaranteed, that is, the number of consecutive equal bits (0s or 1s) in the stream must be limited. Another desirable property, since it avoids baseline wander problems in AC-coupled receivers, would be DC balancing, that is the property of having an equal number of transmitted 0s and 1s in each portion of the stream of given length. For example, a popular line code that addresses these two requirements is 8b/10b [24], that maps 8-bit symbols to 10-bit symbols in such a way that the so called *running disparity* (that is the difference between the total number of 0s and 1s that has been transmitted) is always bounded between -1 and +1 and the maximum number of consecutive 0s or 1s in the stream is limited to five.

The 8b/10b code (or similar block codes) cannot be employed directly in the FF-LYNX protocol because of the division of the stream into two channels; however, the THS encoding can be modified to provide a bounded running disparity and a minimum frequency of transitions at least in the THS channel.

To this end, the first thing that can be done is to change the NOP sequence: instead of transmitting all zeroes into the THS channel when no THS command is being sent, which causes a constant

increase of the running disparity, the 2-bit pattern “01” is continuously transmitted, thus introducing frequent transitions and keeping the running disparity constant. This corresponds to consider “010101” as the 6-bit NOP sequence, instead than “000000” (note however that also “101010” could be considered as the NOP sequence, with the same effects on DC balancing and transitions; this choice would lead to TRG and HDR sequences that are complementary to the ones that will be found in the following, but obviously with the same balancing properties). Then, new sequences can be sought to represent TRG and HDR commands, with the requirement that they are balanced, i.e. have the same number of 0s and 1s. For this purpose, a modified version of the C program used for the first THS encoding was employed, considering “010101” as the NOP sequence and looking for two (or possibly more) 6-bit sequences that satisfy the three conditions exposed in section 2.2.3.2.

The execution of this program returns the following two pairs of sequences, in the hypothesis of NOP = 010101:

00 10 11 10 00 11
11 10 00 10 11 00

that are of course equivalent as for balancing. The program wasn’t able to find a third balanced sequence with the desired properties, so the SYN command can’t be encoded with this scheme: however, the NOP sequence itself can be used as a synchronization pattern by the receiver, with the advantage that it is continuously transmitted when the THS channel is free from TRG or HDR, thus speeding up the sync locking process and aiding the maintenance of sync lock on the correct channel when spurious THS sequences occur on other channels.

In conclusion, in the FF-LYNX protocol v.1 the following THS encoding was implemented:

NOP = 01 01 01
TRG = 10 00 11
HDR = 10 11 00

2.3 The FF-LYNX frame

As in any other data-link layer protocol, in the FF-LYNX protocol the frame is the data unit employed to carry user information from transmitter to receiver. In protocol v.1 only a basic kind of frame is implemented, used to transfer generic low-priority data: that is, commands and configuration data in the downlink, and readout data or responses to commands or monitoring data in the uplink. Trigger data frames, used to transport high-priority trigger data in the uplink (to be implemented in a successive version of the protocol), will be introduced in section2.3.3.

The structure of the basic FF-LYNX frame is illustrated in Fig. 2.8:

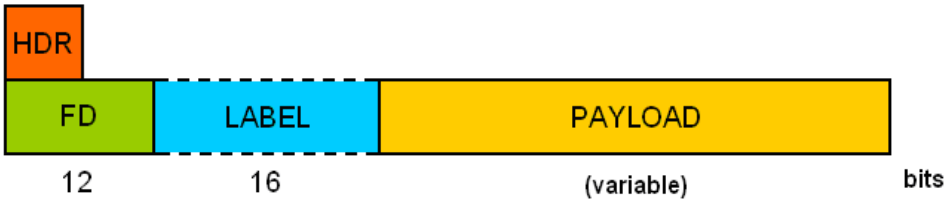


Fig. 2.8 – Basic FF-LYNX frame structure; the dashed-line field is optional.

This very simple frame structure comprises the following fields:

- Header: marks the beginning of the frame;

- Frame Descriptor (FD): contains information about the frame such as the frame length, expressed in number of 16-bit words, and the optional presence of the "label" field following the frame descriptor itself;
- Label: optional field can carrying additional information associated to the payload.
- Payload: carries user data, and is divided in words of 16 bits each.

A fundamental concept underlying to this frame structure, common to protocol stack models in networking, is encapsulation of user data: to obtain maximum flexibility, the frames can transport in their payload any type of information, with its own structure, that is simply unpacked from the frame by the receiver interface and delivered as it is to the receiver host; in other terms, the link is transparent to the user hosts with respect to the carried information, not imposing any particular structure to it. The only characterization given to the data contained in the payload is the label, that as will be explained in the following is intended to contain additional qualification that the user wants to associate to the data transported by the frame, such as a timing information.

The beginning of the frame is marked by the header, that is transmitted as a HDR sequence in the THS channel beginning in the same 40 MHz clock cycle in which the transmission of the frame itself begins in the FRM channel, as shown in Fig. 2.9:

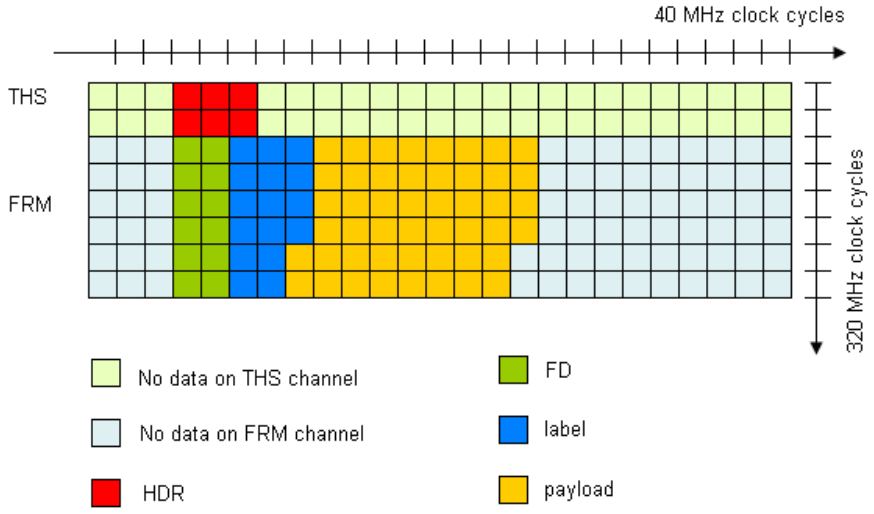


Fig. 2.9 – A HDR sequence in the THS channel marking the beginning of a frame transmission in the FRM channel.

To allow the receiver to locate the end of the frame, a trailer (or end-of-frame flag) could have been used, i.e. a special bit sequence transmitted after the last bit of the payload. With this approach, though, it must be guaranteed that the trailer sequence can never appear in the payload, since otherwise the receiver would suppose the end of the frame by detecting it. This issue is typically resolved in character-oriented protocols through character stuffing, i.e. if the end-of-frame sequence occurs in the payload, an "escape" character is added before it in the stream, so that the receiver understands that the following character is just data and not the real trailer, marking the end of the frame; furthermore, if the escape character occurs inside the payload, another escape is added before it, and so on. In the FF-LYNX protocol, however, this solution is impractical, since the possibility of bit-flips would oblige to let the receiver to detect as the trailer not only the "pure" trailer sequence but also bit-flip-corrupted trailer sequences: this in turn would mean that character stuffing should be done for all the bit-flip-corrupted trailer sequences occurring inside the payload, and also for all the bit-flip-corrupted escape sequences, with the result that most of the transmitted bits would be character-stuffing overhead instead than real information. Therefore, information about the length of the frame is transmitted instead: the payload is structured in 16-bit words, and its length is specified in the Frame Descriptor as the number of words constituting the payload.

The fields of the FF-LYNX frame are described hereafter in more detail.

2.3.1 The Frame Descriptor

The first field of the frame after the header, called Frame Descriptor (FD), specifies some important characteristics of the frame itself and the payload. In protocol v.1, the FD consists of 7 bits of information, divided into the following sub-fields (Fig. 2.10):

- Frame Length (FL), 4 bits: indicates the length of the frame expressed in number of 16-bit words, including the label if it's present; more precisely, the number of words in the frame is obtained as FL + 1: the minimum length of a frame is therefore 1 word (FL = 0) and the maximum length is 16 words (FL = 15).
- Label On (LO): 1 bit: indicates whether the label is present or not; if a data packet is fragmented in multiple frames (see later on), only the first has the label.
- Data Type (DT), 1 bit: specifies the type of data carried in the payload; it is intended to discriminate between DAQ data (e.g.: raw data or trigger data) and responses to commands (e.g.: configuration or monitoring data) in the uplink, so that a possible data concentrator system can identify the frame on which event building has to be performed.
- Last Frame (LF), bit: it is used in case of fragmentation of a data packet into multiple frames, as will be explained in section 2.3.2.

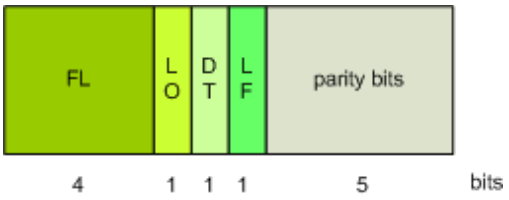


Fig. 2.10 – Frame Descriptor structure.

Since it carries the frame length information, that is crucial for the correct reception of the frame, the Frame Descriptor must be protected against transmission errors: this is done by means of a H(12,7) code, that is a shortened and extended Hamming code capable of detecting and correcting single bit-flip and detecting double bit-flips.

2.3.1.1 The H(12,7) encoding

A Hamming code was chosen for error protection of the Frame Descriptor because of its simple implementation. More precisely, a H(12,7) code was chosen, that is a modified version of a H(15,11) Hamming code with SECDED capability.

To derive the H(12,7) code and define the architecture of its encoder and decoder, one has to start from a standard Hamming code. Since the bits to be encoded in the Frame Descriptor are 7, a Hamming code with $k \geq 7$ must be considered first: this is H(15,11), obtained by setting $r = 4$ and therefore $n = 15$ and $k = 11$. A standard form parity check matrix for this code, constructed in accordance to the definition of binary Hamming code, is the following 4 × 15 matrix:

$$H_{(15,11)} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} = \left[-A^{\mathrm{T}} \middle| I_4 \right]$$

and a corresponding generator matrix will be $G_{(15,11)} = \left[I_{11} \middle| A \right]$, which is a 11 × 15 matrix.

The encoding of 11-bit messages **x** through the operation $\mathbf{c} = \mathbf{x}G_{(15,11)}$ will then produce 15-bit codewords **c** containing $k = 11$ information bits and $r = 4$ redundancy bits:

$$\mathbf{c} = (c_1, ..., c_{15}) = (x_1, ..., x_{11}, p_1, ..., p_4)$$

The redundancy bits p_i are also called *parity bits* since each of them can considered as the even parity bit of a group of information bits, being the result of the modulo-2 sum of the information bits of that group: for example, p_1 is obtained as the multiplication of **x** for the 12th column of *G* and so it is:

$$p_1 = c_{12} = (x_1, ..., x_{11}) \begin{pmatrix} g_{1,12} \\ \vdots \\ g_{11,12} \end{pmatrix} = x_1 + x_2 + x_4 + x_5 + x_7 + x_9 + x_{11}$$

Since we need only 7 bits of information, this H(15,11) code can be shortened. The *shortening* of a code is the removal of some information bits, keeping the number of the redundancy bits the same: the length *n* of the code is thus reduced by the same amount. The removal of information bits from a code can be regarded as the result of two steps [25]:

1. set those bits to 0 in all the codewords and recalculate the parity bits;
2. remove these zero bits;

(also, duplicate codewords that possibly originated from the previous steps must be obviously eliminated). Now, since the first step changes the original codeword into another codeword of the same code, the minimum distance is not reduced, nor it is decreased by the removal of zero bits in the same position from all the codewords. Hence, we can conclude that the minimum distance of a binary code is not reduced by shortening (while it could be even increased); this guarantees that a shortened Hamming code has minimum distance of 3 at least.

To find the *H*' and *G*' matrices of the shortened code, we observe that the two steps above respectively correspond to the following operations on the *H* and *G* matrices (supposed to be in standard form) of the original code:

1. remove the rows of *G* corresponding to the positions of the bits that were set to zero;
2. remove the columns of *G* corresponding to the same positions, since after step 1 these columns contain now only zeroes.

We can therefore shorten the H(15,11) code by removing for example the information bits in position 8, 9, 10 and 11 from each codeword, and hence canceling the same rows and columns from $G_{(15,11)}$ thus obtaining the following generator matrix:

$$G_{(11,7)} = \left[I_7 \middle| \tilde{A} \right] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

where $\tilde{A}$ is the 7 × 4 matrix resulting from the elimination of the last four rows of the matrix A in $G_{(15,11)}$. This $G_{(11,7)}$ generates a H(11,7) code that, for what has been said, has minimum distance of 3 at least; however, it can be verified that d is exactly equal to 3 as in the original Hamming code.

In the H(11,7) code a message $\mathbf{x} = (x_1, \ldots, x_7)$ gets encoded as

$$\mathbf{c} = (c_1, ..., c_{11}) = \mathbf{x}G_{(11,7)} = (x_1, ..., x_7, p_1, ..., p_4)$$

where

$$\begin{aligned} p_1 &= x_1 + x_2 + x_4 + x_5 + x_7 \\ p_2 &= x_1 + x_3 + x_4 + x_6 + x_7 \\ p_3 &= x_2 + x_3 + x_4 \\ p_4 &= x_5 + x_6 + x_7 \end{aligned} \tag{2.15}$$

These formulas therefore specify the operations that must be performed to find the four parity bits that must be appended to the message in the H(11,7) encoding.

The parity check matrix associated to $G_{(11,7)}$ is

$$H_{(11,7)} = \left[-\tilde{A}^{\mathbf{T}} \middle| I_4 \right] = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and its knowledge allows us to derive the formulas for calculating the syndromes for the decoding of a received word **y**:

$$\mathbf{s}(\mathbf{y}) = \mathbf{y}H_{(11,7)}^{\mathbf{T}} = (s_1, s_2, s_3, s_4)$$

with

$$\begin{aligned} s_1 &= y_1 + y_2 + y_4 + y_5 + y_7 \qquad + y_8 \\ s_2 &= y_1 + y_3 + y_4 + y_6 + y_7 \qquad + y_9 \\ s_3 &= y_2 + y_3 + y_4 \qquad + y_{10} \\ s_4 &= y_5 + y_6 + y_7 \qquad + y_{11} \end{aligned} \tag{2.16}$$

Each syndrome bit s_i is therefore calculated as the even parity of the group of bits that includes the message bits that originated p_i, and p_i itself: it can be thus verified again that if no error has occurred in the transmitted codeword, i.e. $\mathbf{y} = \mathbf{c}$, then all the syndrome bits will be zero.

If instead a single bit error in position i has affected the transmitted codeword, then we can write $\mathbf{y} = \mathbf{c} + \mathbf{e}_i$ where $\mathbf{e}_i$ is the n-bit vector with a 1 in position i and zeroes elsewhere, and therefore we have

$$\mathbf{s}(\mathbf{y}) = \mathbf{s}(\mathbf{c} + \mathbf{e}_i) = \mathbf{s}(\mathbf{e}_i) = \begin{pmatrix} h_{1,i} \\ \vdots \\ h_{n-k,i} \end{pmatrix} \tag{2.17}$$

that is, the syndrome is equal to the i-th column of H. This is a property of all Hamming codes, and allows us to identify the position of the single bit error that has occurred: indeed, being the columns of H all different by construction, each possible value of the syndrome is associated to one

and only one error vector $\mathbf{e}_i$: the correct codeword can thus be reconstructed by performing $\mathbf{c} = \mathbf{y} - \mathbf{e}_i$, i.e. by flipping the i-th bit of the received word **y**. To carry out the decoding, the receiver has thus to hold a table that associates each possible value of the syndrome to the corresponding error position: this table has a number of entries equal to the number of the columns of H, and for the H(11,7) code is the following:

Syndrome	Error position
0000	no error
1100	1st bit
1010	2nd bit
0110	3rd bit
1110	4th bit
1001	5th bit
0101	6th bit
1101	7th bit
1000	8th bit
0100	9th bit
0010	10th bit
0001	11th bit

Table 2.1 – Correspondence between syndrome value and error position.

Note that a syndrome equal to a power of 2 (1000, 0100, 0010, 0001 in this case) means that the erroneous bit is one of the parity bits.

If instead more than one error occur in a single word, this decoding method will fail: in fact, considering for example two errors, if $\mathbf{y} = \mathbf{c} + \mathbf{e}_i + \mathbf{e}_j$ ($j \neq i$) then the syndrome calculated on **y** will be recognized as one resulting from a single error, different from the two that actually occurred; it is also possible that $\mathbf{s}(\mathbf{y})$ falls out of the set of the recognized syndromes (i.e. the ones listed in Table 2.1), since H(11,7) is a shortened code and so the recognized syndromes are not all the possible binary numbers of r bits ($r = 4$ in this case).

Besides, the H(11,7) code has minimum distance equal to 3, and hence its error correction capability is limited to single bit errors in each word. However, as it has been anticipated in section 2.2.3.1, H(11,7) can be provided with an additional protection against double bit-flips by extending it to a H(12,7) code, that is a SECDED code: the extension is made by adding an extra parity bit to each codeword of H(11,7), so that the length n of the code is increased by one (from 11 to 12) while the number k of the information bits remains unchanged (k = 7).

The additional redundancy bit is calculated as the overall even parity bit of the codeword: hence, if $\mathbf{c} = (c_1, c_2, \ldots, c_{11})$ is a codeword of H(11,7), the corresponding codeword of H(12,7) will be $(c_1, c_2, \ldots, c_{11}, c_{12})$ with $c_{12} = c_1 + c_2 + \ldots + c_{11}$. When a linear binary code C is extended with this method to create an extended code $\hat{C}$, the following facts can be easily realized:

- $\hat{C}$ is still a linear code;
- $\hat{C}$ has only even-weight vectors, i.e. vectors whose weight is an even number;
- if the minimum distance d of C is odd, then the minimum distance of $\hat{C}$ is $d + 1$.

We can therefore conclude that H(12,7) has minimum distance equal to 4, and so it is a SECDED code: if a single bit error affects a transmitted codeword **c**, the receiver can correctly reconstruct **c** as the codeword at distance 1 from the received vector **y**; if instead **c** is corrupted by a double bit

error then **y** will be at distance 2 from all the codewords, and so the double error can be detected, although not corrected.

If G is a generator matrix for a linear code C then a generator matrix $\hat{G}$ for the code $\hat{C}$, that is the extension of C according to the method of adding an overall parity check, can be obtained from G by adding to it an extra column so that the sum of the elements of each row of $\hat{G}$ is zero. Therefore, a generator matrix for the H(12,7) code will be

$$G_{(12,7)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{2.18}$$

and so, according to the usual encoding formula $\mathbf{c} = \mathbf{x}G$, the parity bits that in this code must be appended to the messages (x_1, x_2 , ... , x_7) are the p_1, p_2, p_3, p_4 of the H(11,7) code (formulas (2.15)) plus the additional parity bit p_5 calculated as

$$p_5 = x_1 + x_2 + x_3 + x_5 + x_6 \tag{2.19}$$

From its definition, this additional parity bit can be obviously calculated also as $p_5 = x_1 + ... + x_7 + p_1 + ... + p_4$, but this would require to calculate p_1, ... , p_4 first, and so the way offered by (2.19) is faster because it allows to obtain p_5 directly from the message.

The resulting codeword **c** = (c_1, ... , c_{12}) is then transmitted, and a vector **y** is received which is the transmitted **c** possibly corrupted by errors:

$$\mathbf{y} = (y_1,...,y_{12}) = (c'_1,...,c'_{12}) = (x'_1,...,x'_7,p'_1,...,p'_5) \tag{2.20}$$

For the decoding, a 4-bit syndrome (s_1, ... , s_4) is calculated on the received vector **y** in the same way as in the H(11,7) code (formulas (2.16)), and also an additional bit s_5 is calculated as the overall parity bit on the whole received vector:

$$s_5 = \sum_{i=1}^{12} y_i \tag{2.21}$$

If no error has occurred in the transmitted codeword, then both the syndrome (s_1, ... , s_4) and the overall parity bit s_5 will be zero. In case of a single bit error, there are two sub-cases: if the error is in the message bits x'_1, ... , x'_7 or in the first 4 parity bits p'_1, ... , p'_4 then the syndrome (s_1, ... , s_4) will be non-zero and s_5 will be 1, and the decoder can use the H(11,7) table to find the error position from the syndrome value; if instead the error is in the overall parity bit $y_{12} = p'_5$ then the syndrome (s_1, ... , s_4) will be zero and s_5 will be 1, and in this case the decoder can deliver the received message (x'_1, ... , x'_7) unchanged. Finally, if the received word is corrupted by a double bit error then the syndrome (s_1, ... , s_4) will be non-zero but the overall parity bit s_5 will be zero: in this case the decoder must declare the detection of the double error, although it can't correct it.

To sum up, the H(12,7) encoding/decoding procedure chosen for error protection of the Frame Descriptor comprises the following operations:

1) The 7-bit Frame Descriptor **x** = (x_1, ... , x_7) is encoded as a 12-bit codeword

$$\mathbf{c} = (c_1,...,c_{12}) = (x_1,...,x_7,p_1,...,p_5)$$

appending to the message (FD) five parity bits calculated as

$$\begin{aligned} p_1 &= x_1 + x_2 + x_4 + x_5 + x_7 \\ p_2 &= x_1 + x_3 + x_4 + x_6 + x_7 \\ p_3 &= x_2 + x_3 + x_4 \\ p_4 &= x_5 + x_6 + x_7 \\ p_5 &= x_1 + x_2 + x_3 + x_5 + x_6 \end{aligned} \tag{2.22}$$

and codeword **c** (that will be also called FDC, Coded Frame Descriptor) is transmitted.

2) On the received vector **y**, that will be called also FDC', four syndrome bits s_1, ... ,s_4 and an overall parity bit s_5 are calculated as

$$\begin{aligned} s_1 &= y_1 + y_2 + y_4 + y_5 + y_7 + y_8 \\ s_2 &= y_1 + y_3 + y_4 + y_6 + y_7 + y_9 \\ s_3 &= y_2 + y_3 + y_4 + y_{10} \\ s_4 &= y_5 + y_6 + y_7 + y_{11} \\ s_5 &= \sum_{i=1}^{12} y_i \end{aligned} \tag{2.23}$$

3) The number of errors in FDC' (supposed to be not greater than 2) is deduced from the value of (s_1, ... ,s_4) and s_5 according to the following table:

Case	(s_1, ... ,s_4)	s_5	N° of errors
A	0000	0	0
B	0000	1	1 (on p'$_5$)
C	≠ 0000	0	2
D	≠ 0000	1	1

Table 2.2 – Possible cases arising from the value of the syndrome bits and the overall parity bit.

Note however that case B can also stem from the occurrence of an odd number of errors greater than 2, but this goes beyond the detecting capability of the H(12,7) code.

4) In cases A and B the received message bits x'$_1$, ... , x'$_7$ are not corrupted, and hence they are delivered by the decoder without modifications; if instead a double error is detected (case C), the decoder declares the non-correctable corruption of the FDC field; finally, if a single error is revealed (case D) the decoder calculates a 7-bit correction vector **ε** (corresponding to the first 7 bits of the occurred error pattern **e**) in accordance with the following table, that is readily derived from Table 2.1:

$(s_1, \ldots, s_4)$	**ε**
1100	1000000
1010	0100000
0110	0010000
1110	0001000
1001	0000100
0101	0000010
1101	0000001

Table 2.3 – Association of the possible syndrome values to the 7-bit error vector.

The correction vector **ε** is then added to the received vector **y** to correct the error and reconstruct the transmitted FDC. Note that in Table 2.3 the last four values of the syndrome that appear in Table 2.1 are not considered, since they correspond to errors in the parity bits of **y** and therefore the message bits $x'_1, \ldots, x'_7$ must not be corrected.

In conclusion, the logic architecture of the encoder is described by equations (2.22), while the one of the decoder is defined by equations (2.23) and by Table 2.2 and Table 2.3.

2.3.2 Payload and label

As said before, user data are carried as payload of the frame without constraints about the internal structure. The only characterization of the payload in a frame is its subdivision into 16-bit words: this width was chosen to fit typical size of commands and elementary readout packets in the studied HEP experiments. For example, in the CMS pixel detector the slow control is carried out by means of a modified I^2C protocol, with 10-bit long commands [5]; in the ATLAS pixel detector the "fast" control commands are 9-bit long [13].

A feature about data structuring that is offered by FF-LYNX protocol v.1 is the possibility of organizing the data stream in packets. In this sense, a data packet is here defined as a series of data words that have an unitary meaning for the host, that delivers them to the TX interface one consecutively (more details about data delivery to the TX interface by the host will be given in chapter 3); a data packed is then transmitted by the TX interface encapsulating it in a single frame if it is short enough, or fragmenting it into successive frames if its length exceeds the maximum frame length (16 words). If a packet is contained in a single frame, the Last Frame bit of that frame is set to '1'; when instead a packet is fragmented into multiple frames, the Last Frame bit is set only for the last frame of the series, so that the receiver host can reassemble the packet by examining the value of the Last Frame bit, that is outputted by the RX interface contemporaneously with the frame. However, the hosts can use the FF-LYNX interface also for transmitting a *stream* of data without any packet structure: to this end, the receiver host simply doesn't consider the value of the Last Frame bit of the frames, and receives the transmitted data words one after the other reconstructing the data stream.

If the Label On bit in the Frame Descriptor field is set to '1', the first 16-bit word of the payload is considered to be a "label" tagging the frame or the data packet. The label is intended to contain information that characterize the data carried by the frame, to be used for sorting or aggregating data packets. The typical use should be that of event building: in response to a trigger command, each Front End circuit sends a data packet containing its hit data through one or more FF-LYNX frames, appending information about the timing of the hits (time stamp) as the label for the packet;

data packets from many FEs are gathered by a data concentrator unit, that aggregates them on the basis of the label thus constructing longer packets containing all the hit data relative to a single event (i.e. the ones with the same value of the time stamp); finally, the data concentrator sends aggregated data packets to the remote DAQ system, thus performing a first stage of data compression and event reconstruction.

The label must be provided to the transmitter interface (TX) by the host as the first word of the data packet it delivers to TX (see chapter 3 for details); the TX interface will then encapsulate that packet into a frame with the Label On bit set to '1'. If the data packet is fragmented into multiple frames, only the first frame will have the label at its beginning and the Label On bit set.

2.3.3 Trigger data frames

Besides basic, variable length frames intended for generic, low priority data, in future versions of the protocol a special kind of frame is foreseen to carry trigger data, i.e. information from a subset of FE circuits that is used for trigger calculation (e.g. coordinates of clusters of pixels/strips that has been hit at a given clock cycle). Since a fundamental requirement of trigger data, in order to allow identification of interesting events with precise timing, is fixed (and small) latency, trigger data frames have a fixed length (so that the time required for transmission is constant) and are marked with a TRG sequence as their header, so that they are transmitted and routed with the highest priority in any node of the network.

The transmission of a trigger data frame can be started when a hit is detected in the FE electronics, and hits that are detected in the clock cycles used to transmit the frame will be included in the same frame up to a maximum number limited by the frame length: a relative timing information is associated to each hit to allow reconstruction of the exact hit timing. The structure of the trigger data frame is shown in Fig. 2.11; different possibilities are still under study for the size of the fields of this frame, that are the following:

- Number of hits: specifies the number of hits (Nh) transmitted in the frame; it is Hamming-encoded for robustness against bit-flip (e.g.: 1 bit encoded with Hamming in 3, if up to two hits can be transmitted in the frame, 2 bits encoded in 5, if up to 4 hits can be transmitted).
- Payload: contains the information associated to the transmitted hits: Nht bits for the relative hit timing (e.g.: Nht = 2 if the frame length is ≤ 4, Nht = 3 if the frame length is ≤ 8) and Nhd bits for the hit address, i.e. the position of the hit sensor (e.g.: Nhd = 5 if the hit information is the coordinate of the 4-strip cluster where the hit has been detected in a FE ASIC connected to 128 silicon strips).
- Parity bit: for detection of single bit-flips in the payload.

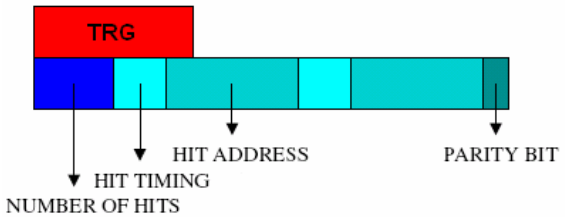


Fig. 2.11 – Trigger data frame structure.

For example, with a link speed of 320 Mbit/s a frame length of three (40 MHz) clock cycles (i.e. 18 bits for data) would allow the transmission of up to 2 hits generated in the 3 clock cycles ($1 \leq Nh \leq 2$; Nht = 2; Nhd = 5) while a frame length of 5 clock cycles (i.e.: 30 bits for the data) would allow

the transmission of up to 3 hits generated in the 5 clock cycles ($1 \leq Nh \leq 3$; Nht = 3; Nhd = 5). Hence, different versions can be defined with respect to Link Speed (LS), frame length (NC) and maximum Number of Hits (NH) per frame. Preliminary comparisons of some of these variants have been carried out on the basis of the following figures of merit: Trigger Transmission Efficiency (TTE), defined as the ratio between the average number of transmitted hits and the average number of generated hits, and Bandwidth Occupancy (BO), that is the fraction of the bandwidth used to transmit trigger data; the results, obtained with high-level C models and Monte Carlo simulation, are shown in Table 2.4.

			C models	Monte Carlo sim.		
LS	NC	NH	TTE	TTE	BO	
320	3	2	96,57%	96,54%	28,32%	(a),(b)
320	5	3	98,53%	98,44%	39,78%	(a),(b)
640	4	7	97,29%			(a),(c)
640	8	13	98,94%			(a),(c)

(a) Poisson distribution for hits (mean = 0.125 hits/bx)
(b) hits generated in 1 FE ASIC
(c) hits generated in 4 FE ASIC

Table 2.4 – Evaluation of Trigger Transmission Efficiency (TTE) and Bandwidth Occupancy (BO) resulting from four possible versions of the trigger data frame, different for the values of the parameters Link Speed (LS), frame length in number of clock cycles (NC), and maximum Number of Hits (NH).

2.4 Future protocol versions

As already mentioned, additional features about error control, line coding and data handling are foreseen to be introduced in future versions of the FF-LYNX protocol, to improve performance and to address the indications that will come out from complete tests of version 1. The features that are currently under study are the following:

- **Interleaving**. If the channel is affected by burst errors, more than one bit-flip can occur in a THS sequence, thus making the THS encoding ineffective; similarly, more than two bit-flips can hit the Frame Descriptor field of a frame, thus deceiving the H(12,7) decoder and preventing the correct reception of the frame. Interleaving can be a solution to this problem: transmission of data is arranged is such a way that bits of different codewords are sent one after the other (for example, the transmission order can be: 1^{st} bit of the 1^{st} codeword, 1^{st} bit of the 2^{nd} codeword, ... , 1^{st} bit of the n^{th} codeword, 2^{nd} bit of the 1^{st} codeword, 2^{nd} bit of the 2^{nd} codeword and so on), and the original order is recovered after reception (deinterleaving): this way, a burst of m consecutive bits in the serial stream can be made to affect only one bit in m different codewords, so that a 1-error-correcting code can still work [25]. Interleaving has a rather simple hardware implementation, but has the drawback of increasing the transmission latency of data: the possible effects of this on trigger commands and data have to be carefully evaluated.
- **CRC**. In version 1 the FF-LYNX protocol doesn't offer any error protection on the payload, leaving it to the upper layer (user level); in next versions, instead, the implementation of a Cyclic Redundancy Check (CRC) covering the payload and the label is foreseen, optionally activated by the host system: if the CRC option is selected, the transmitter interface will calculate the CRC checksum on the payload and the label and will append it to the frame to be transmitted, and the RX interface will perform the CRC check on the received frame signaling possible check failure to the receiver host. The structure of the FF-LYNX frame with the optional CRC checksum field is shown in Fig. 2.12:

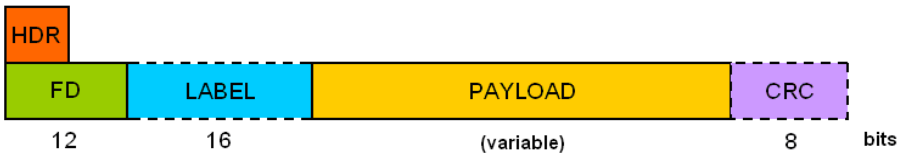


Fig. 2.12 – Structure of the FF-LYNX frame with an 8-bit CRC field; the dashed-line fields are optional.

CRC is considered because of its simple hardware implementation and excellent performance about error detection; various CRC versions, differing in checksum length and generator polynomial (e.g. CRC-8-CCITT (8 bits), used in 1-Wire standard; CRC-16-CCITT (16 bits), used among others in X.25 and Bluetooth protocols [21]) will be compared through simulations to choose the one that fits best to the FF-LYNX application.

- **Single wire transmission**. Transmission of serial data stream and clock on the same wire is an upgrade that is considered of major importance for the FF-LYNX protocol, in order to reduce the cablings in the target application and to avoid possible skew problems between the CLK and the DAT lines arising in the double wire version. With single wire transmission, a Clock and Data Recovery circuit (CDR), usually comprising a PLL (Fig. 2.13), is needed in the receiver to extract the timing signal from the received stream and use this for data sampling: in order for this scheme to work, the data stream must have sufficiently frequent transitions to allow the CDR to follow any jitter of the incoming stream.

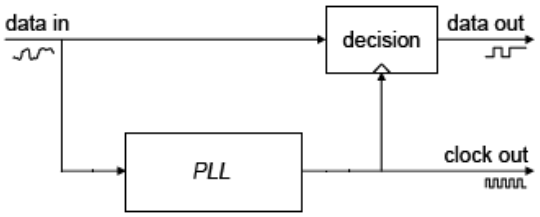


Fig. 2.13 – PLL-based Clock and Data Recovery circuit.

Some block encoding is therefore needed to ensure frequent transitions; as already explained in section 2.2.3.3, character-oriented codes like 8b/10b are widely employed to address this issue, but they are not suitable for the FF-LYNX protocol due to the division of the stream into two channels: a custom block encoding is hence currently under study to be applied to the FRM channel only (thus preserving the THS encoding) and that is able to guarantee a limited maximum number of consecutive identical digits (CID) in the stream, so that the CDR scheme can work properly.

- **DC balancing**. The same custom block encoding used to ensure frequent transitions for the CDR operation can be designed so that it also guarantees a bounded running disparity of the serial bit stream: this way, a stream with negligible DC component can be generated, thus making the FF-LYNX protocol suitable for applications with AC-coupled receivers and high-pass transmission channels.
- **Trigger data handling**. The implementation of trigger data frames, described in section 2.3.3, in future versions of the protocol will allow the transmission of trigger data in the uplink using the same physical links as the readout data transmission. The associated mechanism will be implemented in uplink TX and RX interfaces: when the FE circuit requires the transmission of trigger data, the TX interface builds the special fixed-length frame containing those data, tags it with a TRG sequence as its header and sends it in onto the link with highest priority.

2.5 Protocol validation

A phase of validation of the defined protocol is advisable before proceeding to the interfaces design, in order to verify the correctness of the choices that have been made and to foresee the protocol performance. This is made by means of a high-level software model of the transmitter and receiver interfaces, implementing the defined protocol, and a surrounding test controller that stimulates the model, reads its outputs and measures appropriate performance figures. The model is realized in SystemC language. SystemC is a library of C++ classes and macros which provide an event-driven simulation kernel that makes it possible to simulate concurrent processes, thus resembling hardware description languages such as VHDL and Verilog: when compared to true HDL languages, SystemC has the drawback of inferior performance if used for register transfer level simulation, but greater freedom of expressiveness is offered in return. These features make SystemC optimal for system-level modeling, and for the modeling of hardware at a higher abstraction level than HDL languages, thus allowing faster and more flexible description. On the other hand, SystemC was preferred to other system-level modeling and simulation tools like Simulink or Network Simulator for its better granularity, as a high level of detail (e.g. clock cycle accuracy) is required to evaluate all the aspects of the protocol operation.

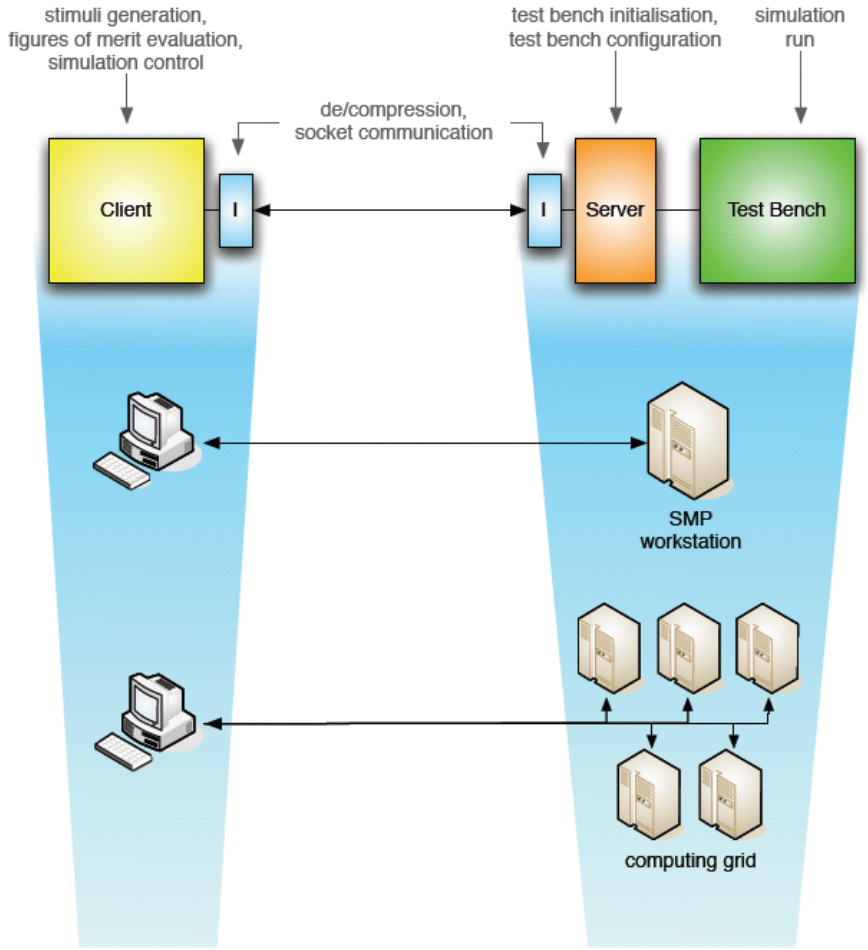


Fig. 2.14 – Client/Server architecture of the test controller system.

The test controller was realized with a client/server architecture that allows different operating configurations (Fig. 2.14). The Client section commands the start of simulation runs, generates the test vectors and calculates the performance figures; the Server instantiates and manages the Test

Bench system that contains the interfaces model. Communication between Client and Server takes place by means of TCP/IP sockets that determine the decoupling of the two sides: this way the Server can be hosted on a physical machine that is different from the one running the Client, such as a Symmetric Multi Processing (SMP) workstation or even a computing grid, to increase the simulation performance.

The SystemC model was built with the aim of maximum extensibility and configurability, in order to allow the test of different versions of the protocol, thus guiding the phase of protocol definition and refinement. To permit comparisons between different versions, the basic task of the high-level simulator is the evaluation of specific cost functions and figures of merit that has been defined on the basis of system requirements. The following figures have been identified as some of the most significant performance metrics for a data transmission system for HEP experiments, in relation to the FF-LYNX protocol:

- lost packet rate (lpr): it is defined as the number of data packets that are lost over the number of packets that are sent during all the test:

$$lpr = \frac{npkt_{lost}}{npkt_{sent}}$$

  a packet is lost if the header of the frame that carries it is not recognized by the receiver due to transmission errors, so that the reception of the frame is not started;
- corrupted packet rate (cpr): it is defined as the number of corrupted packets over the number of packets that are transmitted:

$$lpr = \frac{npkt_{corrupted}}{npkt_{sent}}$$

  a received packet is considered as corrupted if some of its data words are missing (because the Frame Descriptor field of the frame that transports the packet is received with errors) or contain errors;
- minimum, maximum and mean packet latency (min_pl, max_pl, mean_pl): the latency of a data packet is defined as the interval (in number of clock cycles) between the start of the introduction of the packet into the transmitter interface and the end of its delivery to the receiver host by the receiver interface;
- lost trigger rate (ltr): the number of triggers that are lost during transmission over the number of triggers that are sent during all the test:

$$ltr = \frac{ntrg_{lost}}{ntrg_{sent}}$$

  a trigger is deemed as lost if a trigger command is sent at the clock cycle i, but no trigger is received at the cycle $i + tl$ where tl is the trigger latency;
- fake trigger rate (ftr): the number of fake triggers that are received over the total number of received triggers:

$$ftr = \frac{ntrg_{fake}}{ntrg_{rcv}}$$

  a fake trigger reception is counted when a trigger command is received at the clock cycle i, but no trigger was sent at the cycle $i - tl$ where tl is the trigger latency;

In addition, the two-channel structure of the FF-LYNX protocol presents the issue of synchronization. To evaluate this aspect, the following statistics have been considered:

- mean sync time: measures the mean time the receiver needs to recover channel synchronization after losing it as a consequence of a clock error. It depends both on the promptness of the synchronization algorithm in detecting the sync loss, and on its rapidity in acquiring the new lock;
- false sync rate: counts how many times the synchronization algorithm incorrectly reports a loss of synchronization condition over the total number of synchronization loss events. This statistic analyses the sensitivity of the algorithm to spurious THS sequences in the FRM channel.

The evaluation of the above figures in simulations on the SystemC model served as a guide for the choice between architectural alternatives during the protocol definition. As an example, the choice of the synchronization algorithm is here described.

2.5.1 Synchronization algorithm selection

As described in section 2.2.2, a synchronization mechanism based on the counting of THS sequences in each channel and the reaching of two counting thresholds (a high threshold and a low one) was chosen to provide a means of distinguishing between a sync lock state (when synchronization is considered as acquired) and an out-of-lock state (when synchronization is being looked for). This mechanism is hence called Dual Threshold (DT): the transition from the out-of-lock state to the sync lock state takes place when one of the counters reaches the high threshold (thus becoming the in-charge counter) while the inverse transition occurs when a counter that is not in charge reaches the low threshold. Three variations of this algorithm have been devised:

- Fair Dual Threshold (FDT): as soon as a counter hits the higher threshold, it resets all the counters including itself; this ensures that every counter has the same opportunity of becoming the new in-charge.
- Privileged Dual Threshold (PDT): when a counter hits the high threshold, it resets all the other counters but not itself. Whenever the sync lock is confirmed, meaning the in-charge detects another THS sequence, again, all but the in-charge are reset. This approach privileges (hence the algorithm's name) the in-charge counter considering that most of the sequences other counters report are likely spurious and so, to avoid their buildup to the thresholds, a reset is issued often. This way, only tightly spaced bursts of THS sequences (which statistically occur only in the true THS channel) can reach the thresholds thus changing the locked channel.
- Mixed Dual Threshold (MDT), which combines the two previous variations by giving an intermediate level of privilege to the in-charge counter. As soon as it reaches the high threshold, in fact, like in FDT, it is reset together with the other counters. Like in PDT, however, the in-charge has an edge over the contenders: it needs not hit the higher threshold again to confirm the sync lock, as the lower suffices. While the contenders need to reach the higher threshold to relock, the in-charge resets itself and the others upon hitting the lower. It is evident that more leeway is given to other counters to get to the high threshold, than in PDT.

The specific synchronization mechanism described in section 2.2.2 is therefore the PDT.

These three algorithms have been compared on the basis of three of the above defined figures: mean sync time, false sync rate and lost packets rate. These statistics have been evaluated for each algorithm in a benchmark constituted of several simulation runs, with test conditions chosen to represent a peak level situation, where the interfaces are handling an above-average quantity of data

and the clock error rate is over typical levels (a BER in the order of 10^{-9} can be considered for a LVDS link, with a cable length of 1m, a data rate of 100 Mbit/s, operating in a strongly radiation-full environment), in order to have significant results in a relatively short simulation time.

The details of the test conditions are reported in Table 2.5:

Parameter	Value
architecture	4xF
number of runs	9 per algorithm
run duration	0.1 s
packet size	random, 5 to 10
inter-packet idle time	random, 10 to 100 cycles
trigger rate	10^{-2}
missing clock rate	10^{-6}
spurious clock rate	10^{-6}

Table 2.5 – Test conditions for the comparison of the FDT, PDT and MDT synchronization algorithms .

In the nine runs for each algorithm, different values for the thresholds were selected to find out the optimum choice also for this aspect: the tested values are listed in Table 2.6:

	Run								
	1	2	3	4	5	6	7	8	9
Low threshold	2	2	2	3	3	3	4	4	4
High threshold	3	4	5	4	5	6	5	6	7

Table 2.6 – Threshold values used in each simulation run .

The results of the nine simulation runs about mean sync time, false sync rate and lost packets rate are shown in Fig. 2.15, Fig. 2.16 and Fig. 2.17.

As for mean sync time, the best algorithm is FDT due to its fairness, that implies a faster response to clock errors; PDT, on the other hand, has longer time of recovery from a sync loss event, because if spurious sequences arrives on the in-charge channel before the counter on the true THS channel can reach the high threshold, the sync is incorrectly reconfirmed.

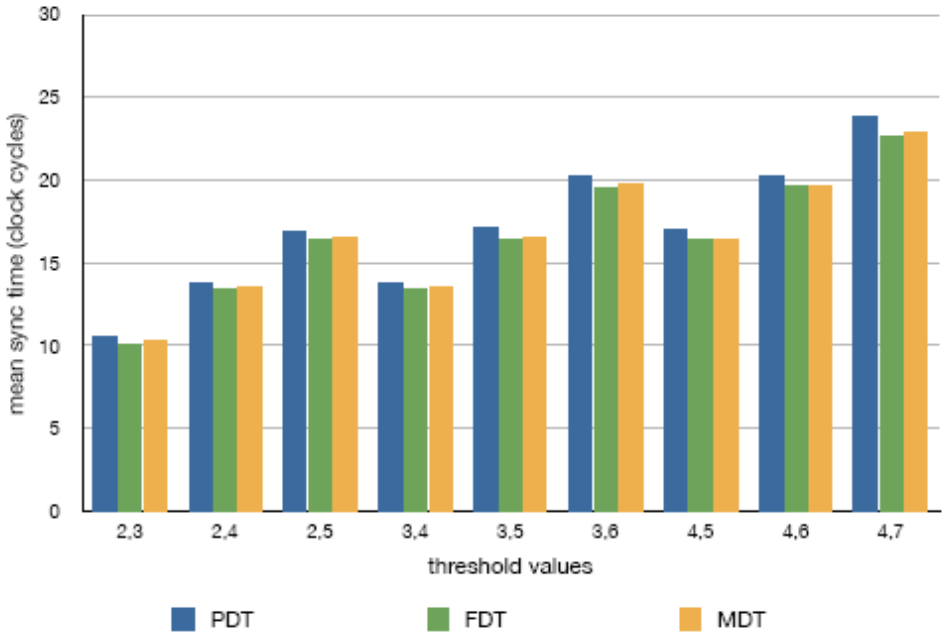


Fig. 2.15 – Mean sync time resulting from simulations for the PDT, FDT and MDT synchronization algorithms and for different threshold values.

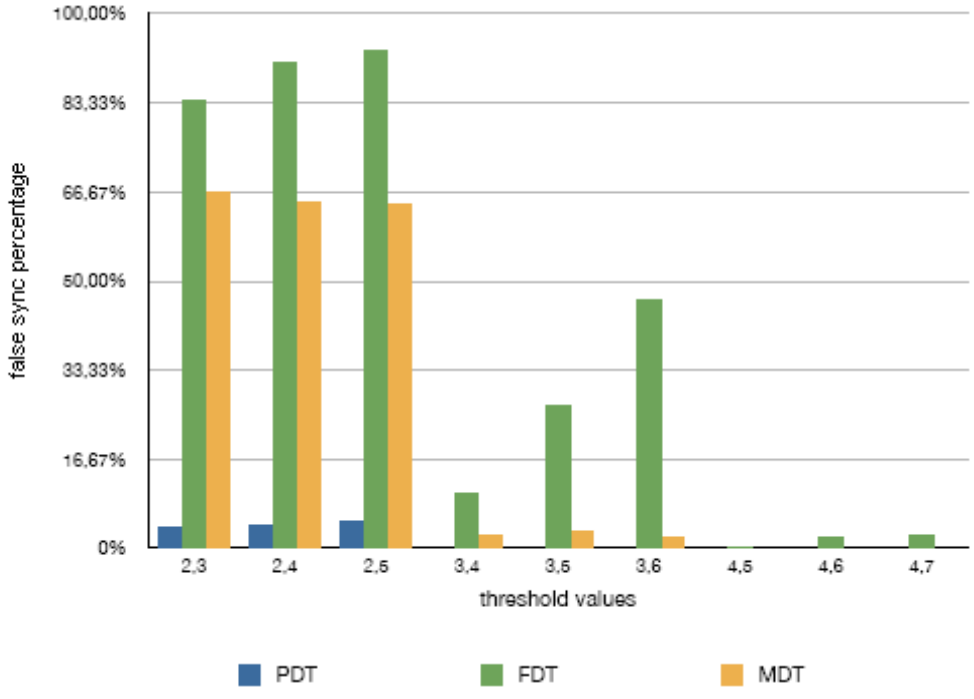


Fig. 2.16 – False sync rate resulting from simulations for the PDT, FDT and MDT synchronization algorithms and for different threshold values.

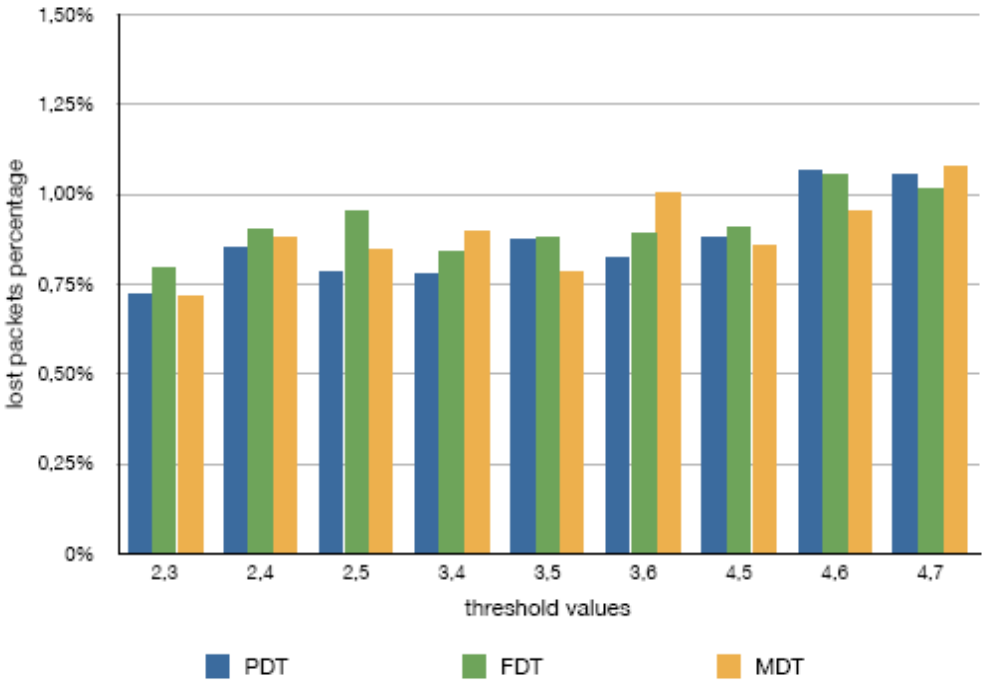


Fig. 2.17 – Lost packets rate resulting from simulations for the PDT, FDT and MDT synchronization algorithms and for different threshold values.

About false sync rate, PDT is markedly the winner: by giving privilege to the in-charge channel, it strongly limits the spurious sequences buildup phenomenon (even avoiding it completely for low threshold values greater than 2); FDT is instead the worse choice here because its fairness makes it particularly defenseless against spurious sequences.

Regarding lost packet rate, finally, PDT globally behaves better than the other two algorithms. This figure, in fact, depends primarily on the time the receiver stays out of the correct sync, which in turn results from the reactivity of the algorithm in recovering the sync after clock errors but also from its tendency to false synchronization: the PDT algorithm, as seen above, is the slower in sync recovery but is very robust against the false sync phenomenon, so that its lost packets rate is the best at least for small threshold values.

In conclusion, PDT is chosen as the best synchronization algorithm after the above considerations, and lower threshold values are preferable since they ensure better reactivity in case of clock errors, thus leading to a lower lost packets percentage. In particular, the values 3 and 4 are chosen for the thresholds so to have a zero false sync rate; this fact makes PDT 3-4 preferable to PDT 2-3 despite the slightly higher packet loss (0.77% versus 0.72%). Preferring a 0% false sync rate over a lower packet loss might seem counterintuitive, but it must be remembered that the one simulated is a peak level situation, with an artificially high error rate. In more common situations, where errors occur less frequently, the advantage of having no chance of losing data or triggers due to false syncs compensates the slightly increased mean sync recovery time.

3 FF-LYNX Interfaces

In this chapter the transmitter and receiver interfaces implementing the FF-LYNX protocol v.1 are described in their architectural and external aspects. For the FF-LYNX Transmitter (FF_TX) and the FF-LYNX Receiver (FF_RX) a description of input and output ports is given first, and then the internal functional architecture is briefly outlined. For ease of exposition, the value of 40 MHz (LHC scenario) is considered for the reference clock frequency F, and consequently the three values of the transmission speed that have been implemented in the interfaces models are referred to as 160, 320 and 640 Mbit/s.

3.1 FF_TX

3.1.1 External specifications

The FF-LYNX Transmitter (FF_TX) is the interface that allows the host system to send trigger commands and data towards a receiver system according to the FF-LYNX protocol. The symbol of the FF_TX showing its inputs and outputs is reported in Fig. 3.1, while Table 3.1 lists the details of these terminals.

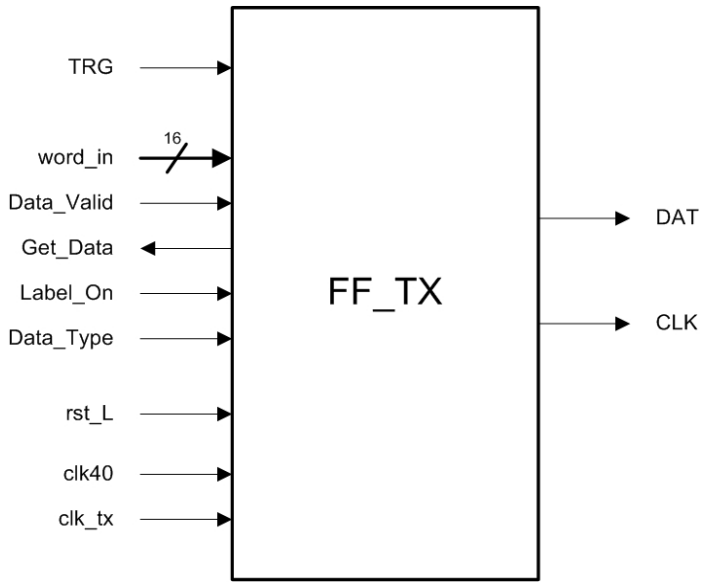


Fig. 3.1 – Input and output terminals of FF_TX.

The signals that constitute the interface with the host system (TRG, word_in, Data_Valid, Get_Data, Label_On, Data_Type) and the active low reset rst_L are all synchronous with the 40 MHz clock (clk40).

The functionalities of the FF_TX interface are:

- to accept trigger commands on the TRG input;
- to read and store data packets, provided by the host through the word_in port in a word-by-word fashion and characterized with the bits Label_On and Data_Type;
- to generate a serial data stream onto the DAT output to transmit the TRG commands and the stored data packets, in accordance with the FF-LYNX protocol.

These functionalities are described hereafter from an external point of view.

Name	Direction	# Bits	Active level	Description
TRG	IN	1	high	Input trigger command
word_in	IN	16	-	Parallel data input
Data_Valid	IN	1	high	Indicates that the value present on the word_in port is valid and must be read by FF_TX
Get_Data	OUT	1	high	Signals that FF_TX is able to read a new word from the word_in input
Label_On	IN	1	high	Specifies the Label_On bit for the current data packet
Data_Type	IN	1	-	Specifies the Data_Type bit for the current data packet
rst_L	IN	1	low	Synchronous reset
clk40	IN	1	-	40 MHz input clock
clk_tx	IN	1	-	Transmission-speed input clock
DAT	OUT	1	-	DAT output line, carrying serial data
CLK	OUT	1	-	CLK output line, carrying transmission clock

Table 3.1 – Description of input and output terminals of FF_TX.

Trigger commands

When a trigger must be sent, the host has to pull up the TRG input of the FF_TX interface for one cycle of the 40 MHz clock: the timing is shown in Fig. 3.2.

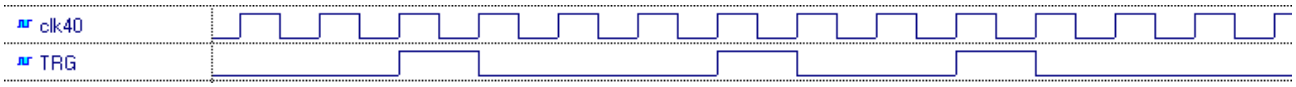


Fig. 3.2 – Input TRG command timing. Three TRG commands are issued in this example.

Each command detected on the TRG input is transmitted onto the DAT stream three clk40 cycles after, in order to be able to schedule header transmissions without overlapping them with the TRG sequence. The minimum interval between two consecutive triggers is three clk40 cycles: that is, an high level on the TRG input one or two cycles after a previous TRG command is ignored by FF_TX, that will send an encoded TRG sequence on the DAT line for the first TRG command only. This behavior is illustrated by the example in Fig. 3.3, where the downmost line shows the effective TRG commands that are recognized by FF_TX interface if the shown pattern is applied to the TRG input.

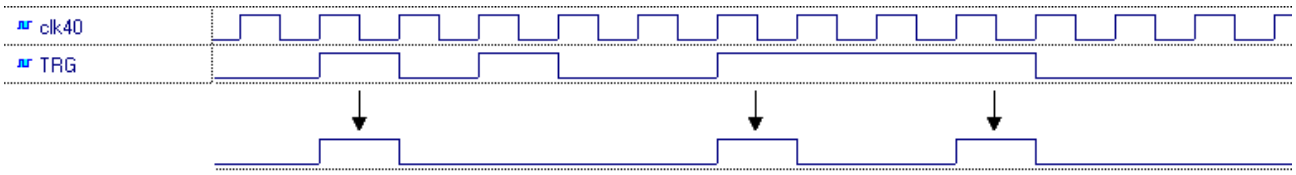


Fig. 3.3 – Example of effective TRG command recognition.

Reading of data packets

To start the storing of a data packed into the FF_TX interface, the host has to pull up the Data_Valid input and to put the first 16-bit word of the packet onto the word_in input. As long as Data_Valid remains high, a new word is read from the word_in input at each clk40 cycle and stored into the internal TX Buffer as a part of the packet, so the host must pull down Data_Valid only at

the end of the data packet. At the beginning of each packet reading, i.e. when the rising of Data_Valid is detected, the value of the Label_On and Data_Type inputs is sampled as well and stored as the value of the Label_On and Data_Type bits for that packet.

If FF_TX is not ready to accept a new input word (because either the TX Buffer, that stores the data packets, or the FDC_FIFO, that stores the Frame Descriptor for the frames to be sent, is full) it will pull down the Get_Data output, keeping it low until the FF_TX becomes available again to store new words: therefore, while Get_Data is low the word presented at the word_in input is not read by FF_TX, and hence the host must hold it until Get_Data goes high again.

Fig. 3.4 shows an example of the timing of the signals concerning data packet reading by FF_TX:

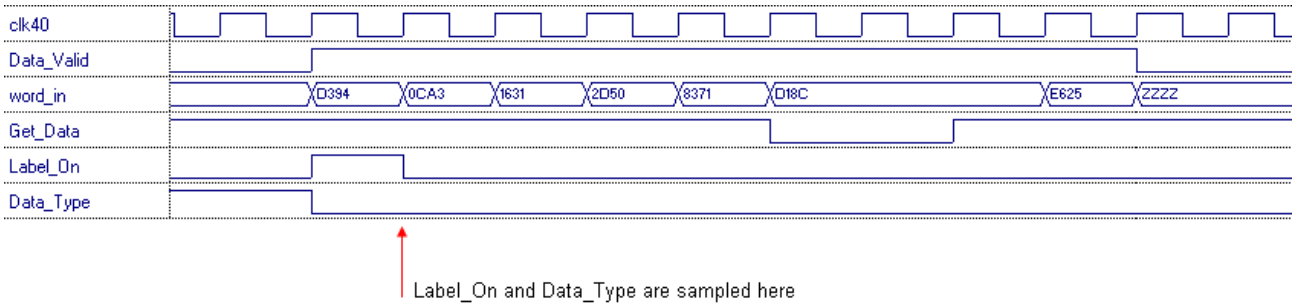


Fig. 3.4 – Example of timing for the reading of a 7-words data packet.

Generation of serial data stream

To transmit the TRG commands and the stored data packets, the FF_TX interface generates on to the DAT output a serial stream, synchronous with the transmission clock (160, 320 or 640 MHz) that is provided to the interface through the clk_tx input and is replied onto the CLK output. In accordance with the FF-LYNX protocol, two bits of the DAT stream in each clk40 cycle are reserved for the THS channel, while the other bits constitute the FRM channel (2, 6 or 14 bits depending on the transmission speed): in particular, the THS bits occupy the second and the third cycle of the transmission clock in each clk40 cycle, as shown in Fig. 3.5.

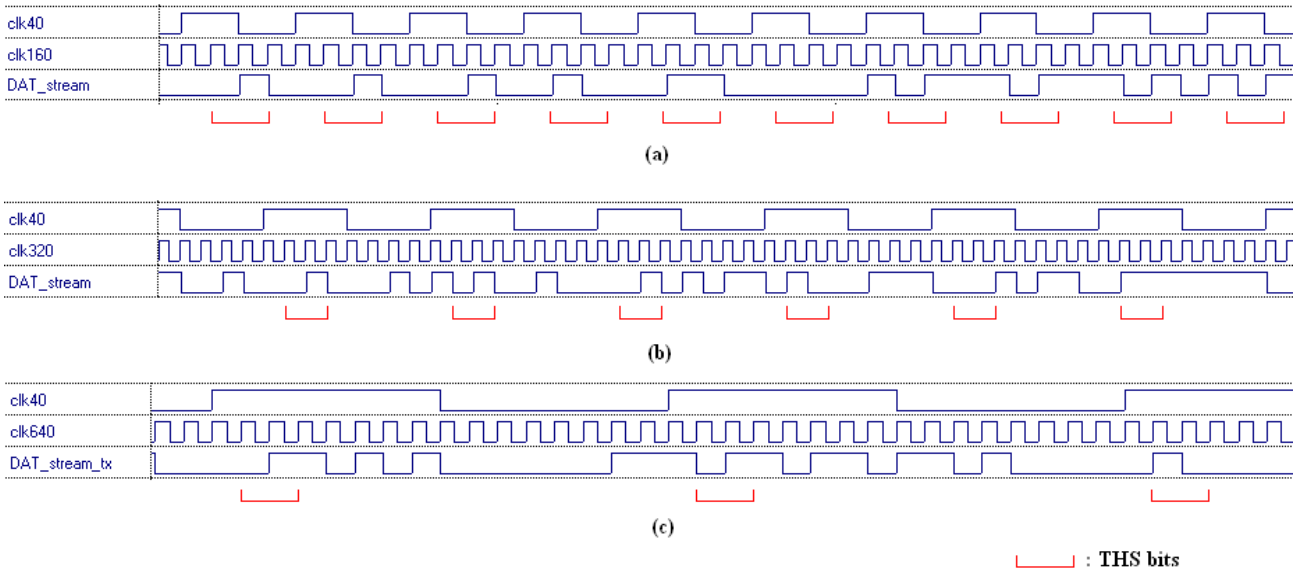


Fig. 3.5 – Example of DAT stream generated by FF_TX in the case of transmission speed equal to 160 (a), 320 (b) and 640 Mbit/s (c).

The THS channel carries TRG sequences to transmit trigger commands to the receiver and HDR sequences to mark the beginning of frames in the FRM channel; when free from TRG or HDR transmission, the THS channel is filled with NOP sequences, constituted by the repetition of the pair of bits “01” at each clk40 cycle. Fig. 3.6 shows the transmission of a TRG sequence in the THS channel occurring, as said before, three clk40 cycles after the external command on the TRG input of FF_TX; in this example the transmission speed is 320 Mbit/s.

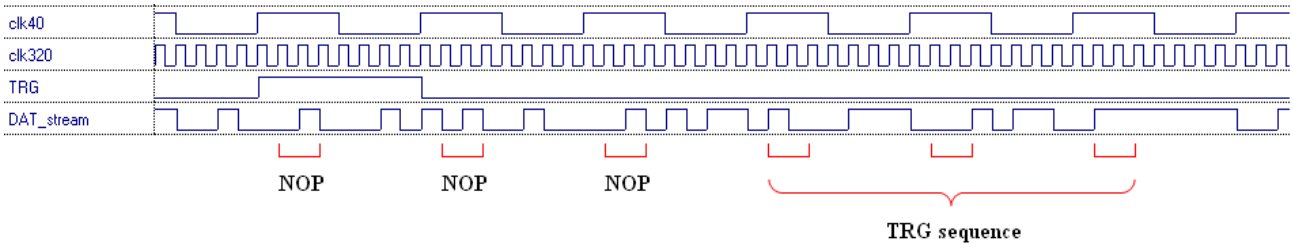


Fig. 3.6 – Transmission of a TRG sequence on the THS channel of the DAT stream.

Reset conditions

When the reset input rst_L of FF_TX is activated, all the internal blocks including the TX Buffer are reset (see later on): the external result is that the DAT output is forced to the low logic level.

3.1.2 Internal architecture

The functional architecture of the FF_TX interface is illustrated by the block diagram reported in Fig. 3.7:

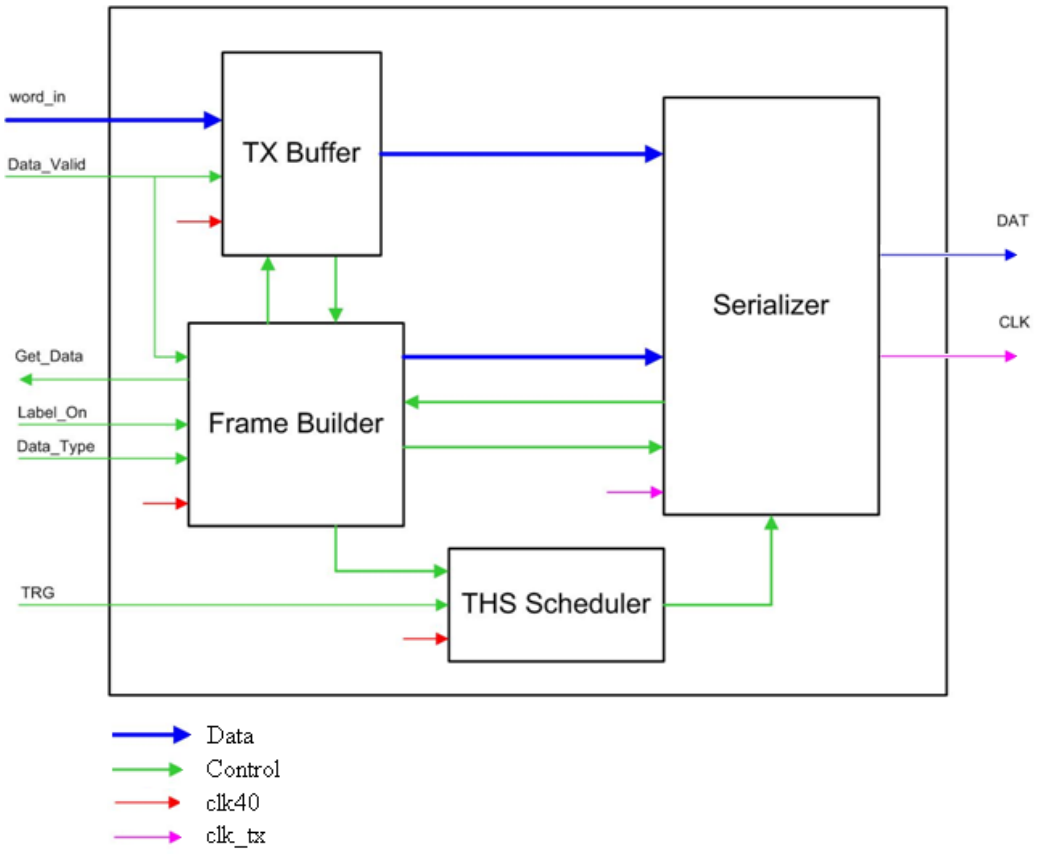


Fig. 3.7 – Functional architecture of the FF_TX interface.

A brief description of the function of these blocks follows:

- **TX Buffer –** The TX Buffer is the memory that stores the data packets awaiting to be transmitted. It is managed as a circular FIFO structure, with 16-bit locations: the data packets are stored at one word per location. The full condition of the TX Buffer determines the pulling down of the Get_Data output, to pause the arrival of packet words by the host. The TX Buffer is clocked by the 40 MHz clock.
- **Frame Builder –** This is the block that controls the assembly of frames for the transmission of data stored in the TX Buffer. In more detail, the tasks of the Frame Builder are: to control the storing of data packets into the TX Buffer, managing the Data_Valid/Get_Data handshake with the host system; to create frames for the transmission of the data packets stored in the TX Buffer, preparing the Frame Descriptor for each frame and encoding it to create the FDC (Coded Frame Descriptor) that is then passed to the Serializer; to start frame transmissions by sending HDR commands to the THS Scheduler; to control the flow of data from the TX Buffer to the Serializer during the transmission. This block is clocked by the 40 MHz clock.
- **THS Scheduler –** This block works out the arbitration between triggers and frame headers: it receives TRG and HDR commands, respectively from the TRG input port of FF_TX and from the Frame Builder, and passes them to the Serializer organizing their arrival in such a way that the transmission of the THS sequences never overlaps. The THS Scheduler is clocked by the 40 MHz clock.
- **Serializer –** The Serializer is the block that generates the DAT output stream with the THS channel and the FRM channel: it is the only FF_TX block that is clocked by the transmission clock . It receives the FDC field from the Frame Builder and frame words from the TX Buffer, and serializes these data into the FRM channel of the DAT output stream; into the THS channel, instead, the Serializer transmits TRG and HDR sequences according to the commands that arrive from the THS Scheduler.

3.2 FF_RX

3.2.1 External specifications

The FF-LYNX Receiver (FF_RX) takes as inputs the DAT and CLK transmission lines, carrying the serial data stream and the transmission clock generated by FF_TX, and delivers to the receiver host the transmitted data packets, trigger commands and a reconstructed 40 MHz clock. Fig. 3.8 shows the input and output terminals of FF_RX, that are then described in Table 3.2.

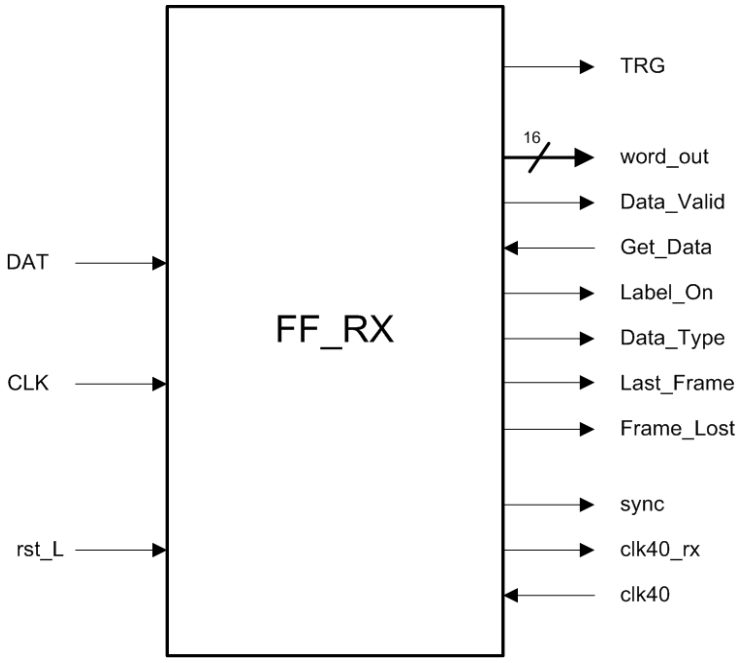


Fig. 3.8 – Input and output terminals of FF_RX.

The signals that constitute the interface with the host system (TRG, word_out, Data_Valid, Get_Data, Label_On, Data_Type, Last_Frame, Frame_Lost) and the active low reset rst_L are all synchronous with the 40 MHz clock (clk40).

The tasks of the FF_RX interface are:

- to synchronize with the THS channel of the DAT stream, generating a reconstructed version of the 40 MHz system clock (clk40_rx); this signal is then processed by an external Delay Locked Loop (DLL) that produces a 40 MHz clock signal (clk40) phase-aligned with the 40 MHz clock of the DAT stream: this clk40 signal is provided to the host chip and to the FF_RX interface itself;
- to detect TRG sequences in the THS channel of the incoming DAT stream and generate trigger commands on the TRG output in correspondence with the detected TRG sequences;
- to receive and store data frames transported by the DAT stream;
- to deliver stored data frames to the receiver host through the word_out port.

These functionalities are described hereafter from an external point of view.

Name	Direction	# Bits	Active level	Description
DAT	IN	1	-	DAT input line, carrying serial data
CLK	IN	1	-	CLK input line, carrying transmission clock
word_out	OUT	16	-	Parallel data output
Data_Valid	OUT	1	high	Indicates that the value present on the word_out port is valid and can be read by the receiver host
Get_Data	IN	1	high	Signals that the host is able to read a new word from the word_out port
Label_On	OUT	1	high	Specifies the Label_On bit for the data packet that is currently being delivered to the host
Data_Type	OUT	1	-	Specifies the Data_Type bit for the data packet that is currently being delivered to the host
Last_Frame	OUT	1	high	Indicates that the frame that is currently being delivered to the host is the last of a packet
Frame_Lost	OUT	1	high	Indicates that a frame has been lost due to corruption of its Frame Descriptor
sync	OUT	1	high	Indicates the achieved synchronization of the receiver on the THS channel of the DAT stream
TRG	OUT	1	high	Output trigger command
rst_L	IN	1	low	Synchronous reset
clk40_rx	OUT	1	-	40 MHz clock reconstructed by FF_RX
clk40	IN	1	-	40 MHz input clock regenerated by DLL

Table 3.2 – Description of input and output terminals of FF_RX.

Synchronization

At the start of transmission, the receiver interface examines the DAT stream searching for THS sequences in order to locate the position of the THS channel in the received serial stream. When a certain number of THS sequences is detected in one of the 2-bit channels of the DAT stream, synchronization is acquired on that channel: at this time FF_RX pulls up the sync output and starts generating the reconstructed 40 MHz clock (clk40_rx output). This is shown in Fig. 3.9.

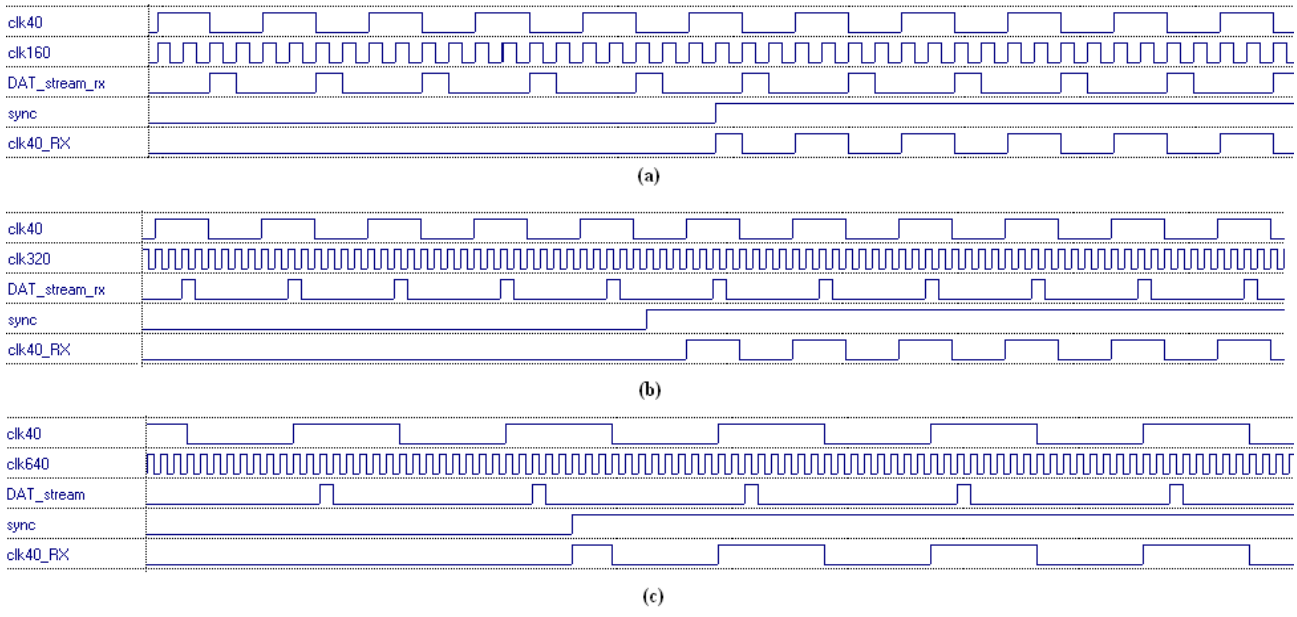


Fig. 3.9 – Acquisition of synchronization by FF_RX at different transmission speed: 160 (a), 320 (b) and 640 Mbit/s (c).

If synchronization between the THS channel in the DAT stream and the receiver is lost during the transmission (due to transmission errors on the CLK line), FF_TX pulls down the sync output and starts again the sync acquisition procedure, at the end of which the sync output when sync is pulled up and the reconstructed clock clk40_rx is adjusted to the new sync lock (Fig. 3.10).

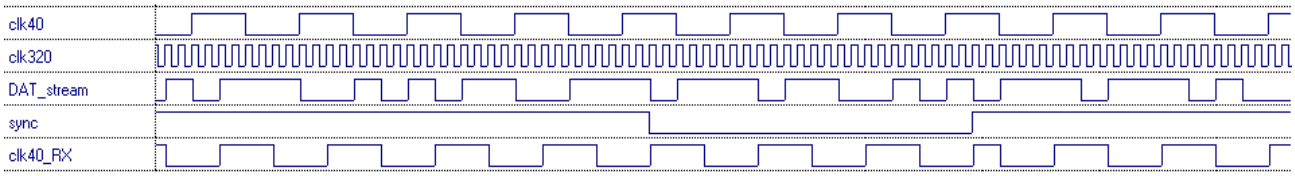


Fig. 3.10 – New acquisition of synchronization by FF_RX after a sync unlock (transmission speed = 320 Mbit/s).

Trigger commands

When a trigger is detected in the DAT stream, the FF_RX interface pulls up the TRG output for one clk40 cycle. The detection occurs when the last bit of the TRG sequence in the stream is sampled by the receiver, and the TRG output command is issued on the next clk40 cycle: Fig. 3.11 illustrates this behavior (the transmission speed is 320 Mbit/s in this example).

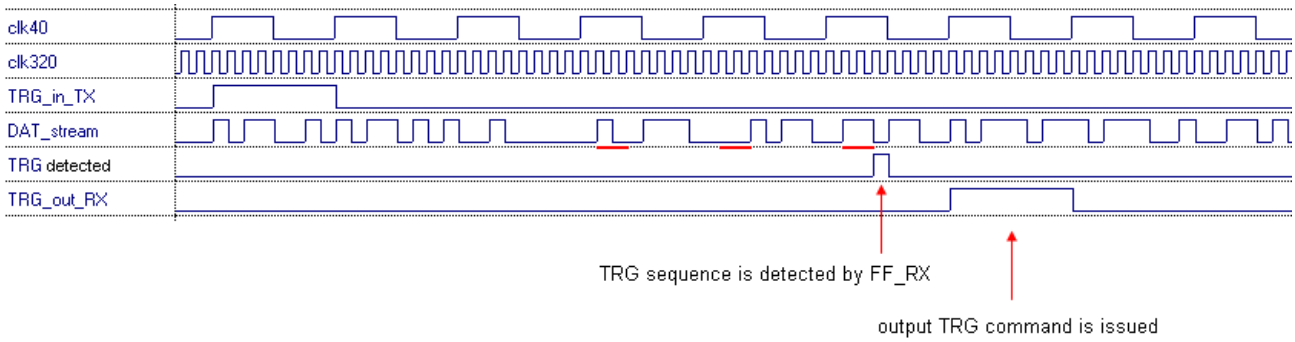


Fig. 3.11 – Detection of a TRG sequence by FF_RX and issuing of the relative TRG output command. The 6-bit TRG sequence in the DAT stream is underlined in red.

Fig. 3.11 also shows that the latency between the TRG input command in the transmitter (TRG_in_TX) and the TRG output command in the receiver (TRG_out_RX) is equal to 6 cycles of the 40 MHz clock, excluding the delay on the transmission line: this is true for all the three values of the transmission speed (160, 320 and 640 Mbit/s).

Delivery of data packets

Data packets travel from the transmitter to the receiver on FF_LYNX frames: the FF_RX interface receives these frames from the link and stores carried data in an internal buffer. Data packets stored in the buffer are finally delivered to the receiver host through the word_out parallel port: the outputting of a data packet is carried out delivering one word per clk40 cycle and one data block at a time by means of a Data_Valid/Get_Data handshake, where the term “data block” is used to designate the sequence of data words that were carried by a single frame .

When a data block is present in the receiver buffer, FF_RX pulls up the Data_Valid output and starts the block delivery to the host, that goes on with the outputting of a data word onto the word_out port at each clk40 cycle in which the Get_Data input is active; hence, the block delivering process can be put on hold by the external circuit by pulling down the Get_Data signal. At the end

of the block delivery, Data_Valid is pulled down for at least one clk40 cycle to indicate the finish of the block to the host.

During the delivery of a data block, the value of the Data_Type and Label_On bits characterizing the frame that transported that block are presented at the Data_Type and Label_On output of FF_RX. If the data packet delivered by the transmitter host to the FF_TX interface was fragmented into more frames, the FF_RX interface will deliver the corresponding data blocks activating the Last_Frame output only for the last block of the packet, while the preceding blocks will have Last_Frame set low: the host can thus reconstruct the data packet from the observation of the Last_Frame signal. If a packet is tagged with a label, the Label_on output will be high for all the blocks of that packet, but only the first word of the first block must be considered as the label.

Fig. 3.12 shows two examples of the timing of these signals.

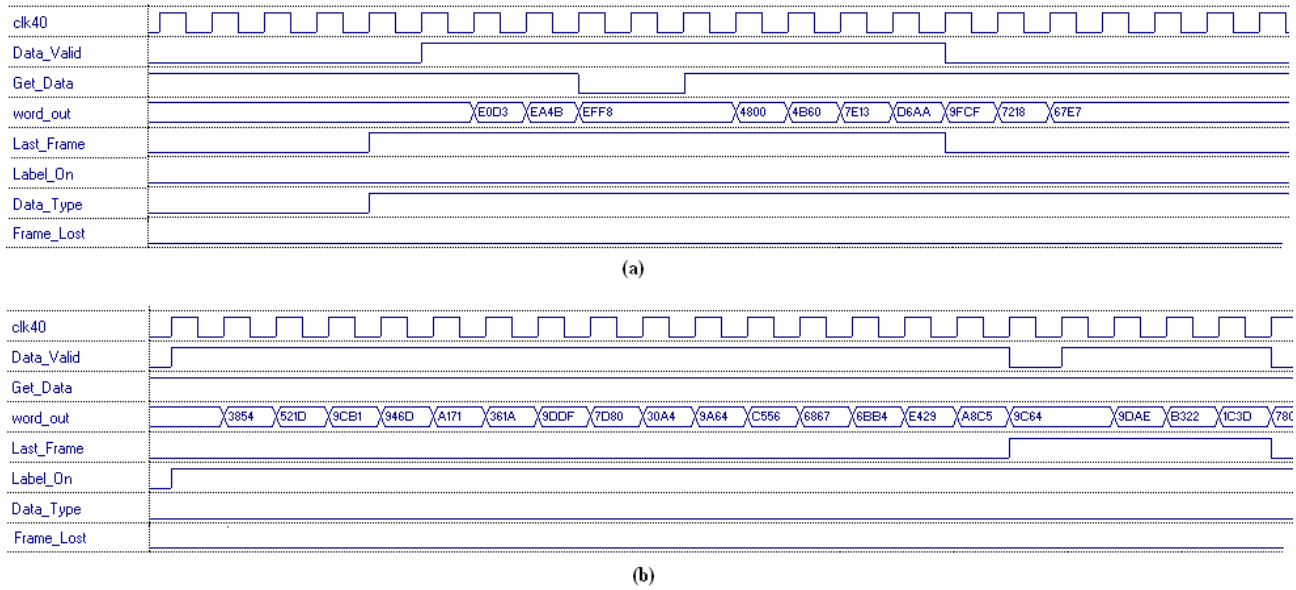


Fig. 3.12 – Two examples of data delivery by FF_RX; in (a) a single data block constituting a data packet (Last_Frame = '1') is delivered, one word per clk40 cycle: the falling of Get_Data puts the delivery on hold, with the word_out kept stable until it is actually read by the host (i.e. Get_Data returns high); in (b) two data blocks are delivered, belonging to a data packet: the first has Last_Frame = '0', while the second has Last_Frame set.

If a frame is lost due to a double bit error in its Frame Descriptor, it is indicated to the host by means of a one-clk40-cycle long Data_Valid pulse, with the Frame_Lost output being high at the same time(Fig. 3.13).

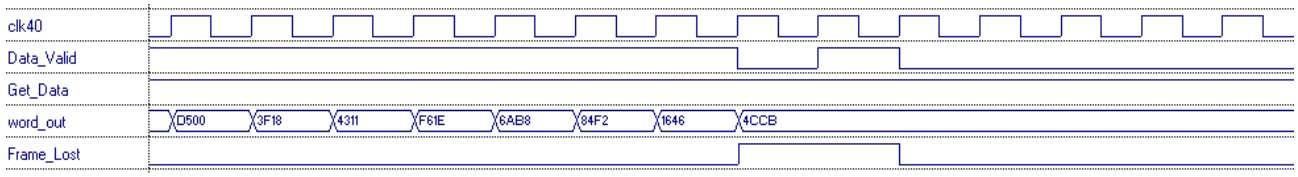


Fig. 3.13 – Signaling of a lost frame by FF_RX.

Reset conditions

When the reset input rst_L of FF_RX is activated, all the internal blocks including the RX Buffer are reset (see internal architecture): the external result is that the all the output are forced to the low logic level.

3.2.2 Internal architecture

The functional architecture of the FF_RX interface is illustrated by the block diagram reported in Fig. 3.14:

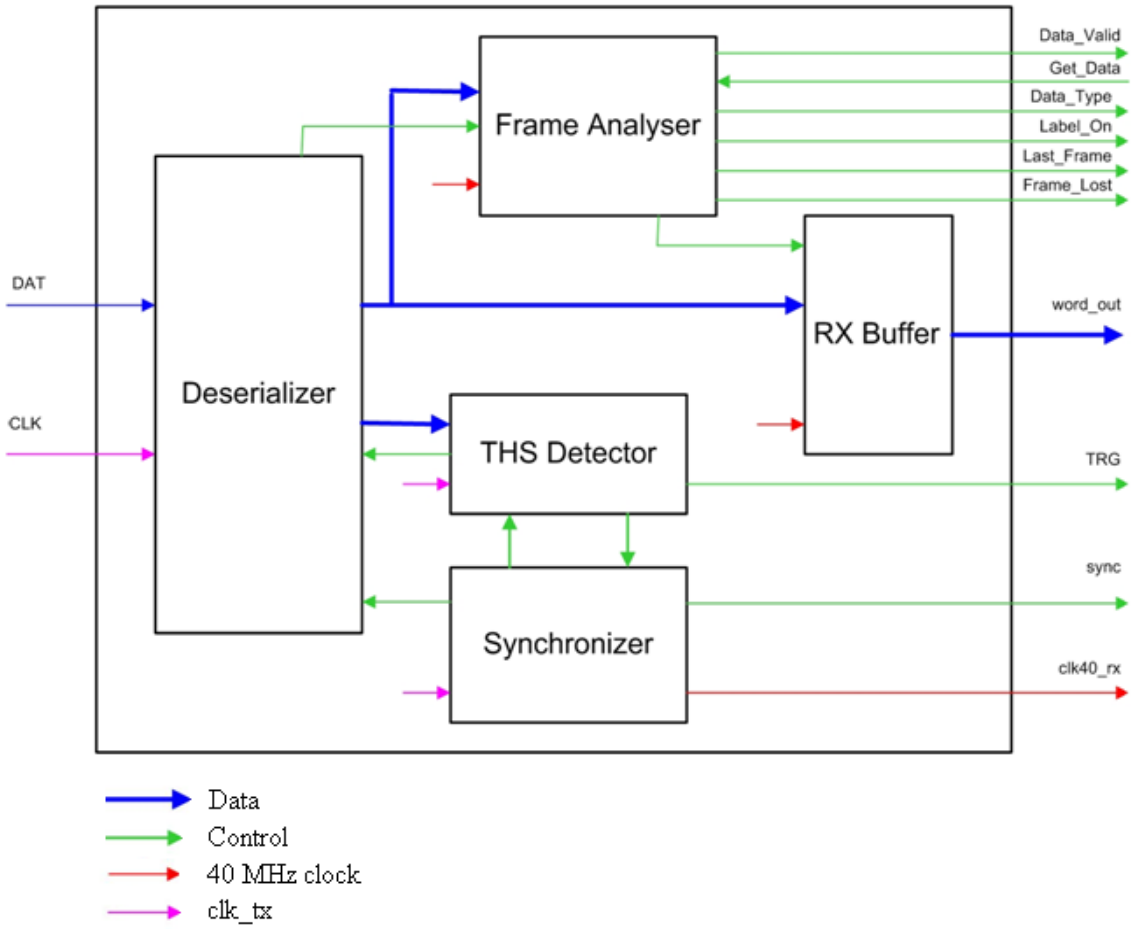


Fig. 3.14 – Functional architecture of the FF_RX interface.

A brief description of the function of these blocks follows:

- **Deserializer –** This block converts back the DAT stream into parallel form, separating the THS channel and the FRM channel and providing the data words for storing into the RX Buffer and the FDC field of incoming frames for decoding by the Frame Analyzer. It is clocked by the transmission clock (CLK input).
- **THS Detector** – It detects the arrival of TRG, HDR and NOP sequences in the THS channel examining the THS bits provided by the Deserializer. The THS Detector is clocked by the transmission clock.
- **Synchronizer** – This block generates the 40 MHz reconstructed receiver clock (clk40_rx) synchronized on the THS channel of the received DAT stream: the synchronization is obtained according to the information coming from the THS Detector about the detected THS sequences. The Synchronizer is clocked by the transmission clock.
- **Frame Analyser –** This is the block that controls the reception of data frames, their storing into the RX Buffer and the delivery of stored data to the receiver host. The task of the Frame Analyser is to decode and store the Frame Descriptor of received frames, to command the writing of the received frame words in the RX Buffer and to manage the reading of the stored

frames by the host, through the Data_Valid/Get_Data handshake. This block is clocked by the 40 MHz clock (clk40).

- **RX Buffer –** The RX Buffer is the memory that stores the received data packets awaiting to be delivered to the receiver host. It is managed as a circular FIFO structure, with 16-bit locations: the data packets are stored at one word per location. It is clocked by the 40 MHz clock.

4 VHDL modeling and simulation

The phase of the design flow that has constituted the main part of the present thesis work is the creation of a VHDL model of the FF-LYNX interfaces for functional verification of the chosen architecture and successive synthesis. VHDL (Very High Speed Integrated Circuits Hardware Description Language) is, together with Verilog, the most widespread language for describing digital electronic systems. As any other Hardware Description Language, VHDL allows to build software models of a digital system, specifying the structure of a design and the function of its building blocks: the result is a Register Transfer Level (RTL) description of the system, that can be verified through functional simulation (after elaboration by a VHDL compiler) and finally passed to a synthesis tool for implementation in a target technology. The typical VHDL modeling and simulation flow is shown in Fig. 4.1:

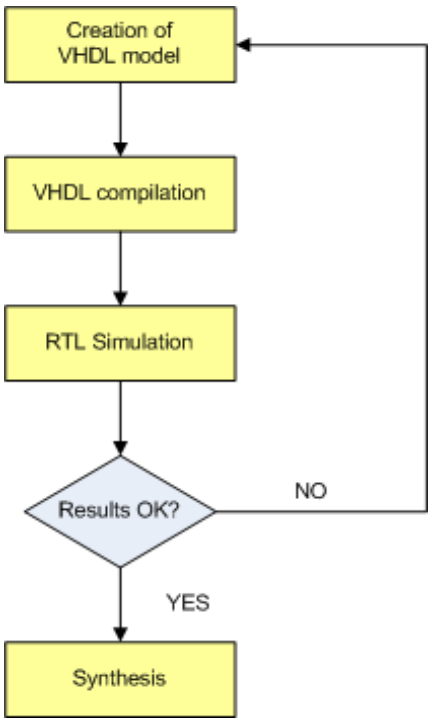


Fig. 4.1 – VHDL modeling and simulation flow.

The development environment used for this phase was Active-HDL by Aldec [27], an integrated environment comprising various design entry tools (HDL text editor, block diagram editor, state diagram editor), VHDL and Verilog compilers, a simulation kernel, debugging tools and graphical and textual simulation output viewers; the version employed was Active-HDL 7.2 Student Edition.

The VHDL model was created according to the interfaces architecture described in chapter 3. To ease the task of building a functionally correct model, the function of each block in the FF_TX and FF_RX architecture was first of all partitioned in simpler sub-blocks; then each sub-block was described in textual form or by means of a graphic entry tool (block diagram editor, for schematic descriptions, or state diagram editor, to describe state machines) and its functionality was verified in a specific test bench. Afterwards, each block of the architecture was built connecting its constituting sub-blocks, and functional simulations on the whole block were carried out. Finally, the complete transmitter-receiver system was built and the overall functionality was verified in a global test bench comprising also external VHDL modules for the generation of test vectors (data and trigger commands to be transmitted) and for the reading of the system outputs (received data and trigger commands).

This chapter describes the VHDL model of the FF-LYNX interfaces in section 4.2, 4.3 and 4.3, showing the implementation details and the results of functional simulation of each sub-block; finally, section 4.4 illustrates the structure of the test bench for the complete TX-RX system and reports the results of functional simulations on it.

4.1 General description of the model

The VHDL model of the FF_LYNX interfaces consists of two entities: FF_TX, that models the transmitter interface, and FF_RX, that is the receiver interface; these two entities are then instantiated in the top-level test bench entity TX_RX_tb that will be described in section 4.4.

The model was created in a strongly parameterized form to allow reusability of building blocks in different parts of the design and fast adaptability to changes about various aspects of the protocol and of its hardware implementation, such as transmission speed, length of the frame fields, word length, buffer size and so on. To this end, a VHDL package (called interfaces_pkg) has been created that defines the numeric constants that are referred to through all the model and specifies their values; Table 4.1 lists these parameters (the type is positive integer for all of them), and the value that has been assigned to them in the first version of the protocol/VHDL model:

Name	Description	Value in model v.1
N_WORD	Length (in bits) of a data word.	16
N_WORD_b	Number of bits of counters that count transmitted/received bits of data words: must be $\geq \log_2$(N_WORD)	4
N_THS	Length (in bits) of THS sequences.	6
N_THS_b	Number of bits of counters that count transmitted/received bits of THS sequences: must be $\geq \log_2$(N_THS)	3
N_FD	Length (in bits) of the Frame Descriptor field, prior to Hamming encoding.	7
N_FDC	Length (in bits) of the coded Frame Descriptor field (i.e. after Hamming encoding).	12
N_FL	Length (in bits) of the Frame Length field inside the Frame Descriptor.	4
N_FLpwr	Maximum number of words carried by frame: N_FLpwr = 2^{N_FL}	16

Table 4.1 – Constants defined in the package “interfaces_pkg” and used as parameters in the interfaces VHDL model.

In addition, the parameters described in Table 4.2 (the type is positive integer for all of them) have been used in FF_TX and FF_RX entities as generics (i.e. parameters that in VHDL are used to pass information to an instance of an entity):

Name	Description	Value for tx speed equal to (Mbit/s)		
		160	320	640
N_CYC	Number of clk_tx (transmission clock) cycles in a clk40 (40 MHz clock) cycle.	4	8	16
N_CYC_b	Number of bits of counters that count tx_clock cycles in each clk40 cycle: must be $\geq \log_2$(N_CYC).	2	3	4
N_FRM	Number of FRM bits in each clk40 cycle.	2	6	14

Table 4.2 – Generics used to configure FF_TX, FF_RX entities and their sub-blocks.

The values of these parameters for the FF_TX and FF_RX entities are set in the top-level test bench by means of a configuration statement, and are then passed through FF_TX and FF_RX to their constituting blocks that use these parameters. There is one configuration statement for each of the three considered transmission speeds: this way, the simulation for the interfaces architecture

relative to 160, 320 or 640 Mbit/s can be set up by simply selecting the appropriate configuration, that passes to FF_TX and FF_RX the values of N_CYC, N_CYC_b and N_FRM needed to simulate a specific transmission speed.

The following sections describe some VHDL entities that are instantiated in various points of the design. The type of all single-bit signals is std_logic, and std_logic_vector for all multi-bit signals.

4.1.1 FIFO_N_D

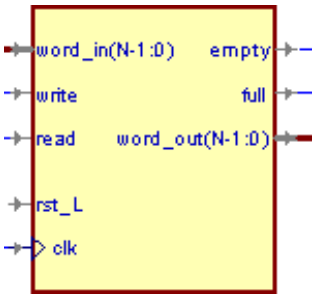


Fig. 4.2 – Symbol of entity FIFO_N_D.

Name	Type	Description
N	integer	Length (in bits) of a each location (word) of the FIFO.
D	integer	Number of locations (depth) of the FIFO.

Table 4.3 – Generics of entity FIFO_N_D.

Name	Direction	# Bits	Active level	Description
word_in	IN	N	-	Parallel data word input.
write	IN	1	high	Synchronous write command.
read	IN	1	high	Synchronous read command.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Input clock..
word_out	OUT	N	-	Parallel data word output.
empty	OUT	1	high	Empty flag: signals that the FIFO is empty.
full	OUT	1	high	Full flag: signals that the FIFO is full.

Table 4.4 – Description of input and output terminals of entity FIFO_N_D.

FIFO_N_D is a FIFO memory with D locations of N bits each. It consists of a D-deep array of N-bit words (D and N are integer generics), two integer pointers W_pnt and R_pnt and two flags for the empty and full conditions. The write pointer W_pnt points to the FIFO location to be written next, while the read pointer R_pnt points to the location to be read next.

The operation of the FIFO is implemented through a writing process and a pointer updating process. If the active low reset input rst_L is active at the rising edge of clk, the writing process writes '0' in all the locations of the FIFO, and the pointer updating process resets W_pnt and R_pnt to 0, sets the empty flag and resets the full flag.

If rst_L is inactive at the rising edge of clk, the write command is active and the FIFO is not full, the writing process stores the word_in value in the location pointed by W_pnt, while the pointer updating process increases W_pnt (resetting it back to 0 if it reaches the value D). If in addition the

read command is inactive, the pointer updating process resets the empty flag and sets the full flag if after its increase W_pnt has reached the same value as R_pnt.

If at the rising edge of clk the reset is inactive, the read command is active and the FIFO is not empty, the pointer updating process increases R_pnt (resetting it back to 0 if it reaches the value D); if in addition the write command is inactive, the full flag is reset and, if after its increase R_pnt has reached the same value as W_pnt, the empty flag is set.

The word_out output is always the value of the FIFO array location pointed by R_pnt.

4.1.2 Counter_N

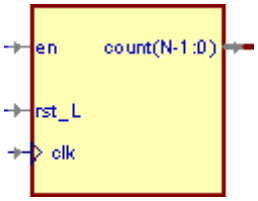


Fig. 4.3 – Symbol of entity Counter_N.

Name	Type	Description
N	integer	Number of bits of the counter.
modul	integer	Modulus of the counter.

Table 4.5 – Generics of entity Counter_N.

Name	Direction	# Bits	Active level	Description
en	IN	1	high	Count enable input.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Input clock..
count	OUT	N	-	Count output.

Table 4.6 – Description of input and output terminals of entity Counter_N.

This is a N-bit binary synchronous up counter with synchronous active-high count enable (en) and synchronous active-low reset (rst_L). The modulus of the counter is specified by generic “modul”: clearly it must be modul $\leq 2^N$.

4.1.3 ffd

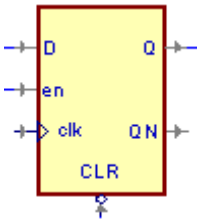


Fig. 4.4 – Symbol of entity ffd.

Name	Direction	# Bits	Active level	Description
D	IN	1	-	Data input.
en	IN	1	high	Enable input.
CLR	IN	1	low	Synchronous reset.
Q	OUT	1	-	Data output..
QN	OUT	1	-	Inverted data output.

Table 4.7 – Description of input and output terminals of entity ffd.

This is a D-type flip-flop with synchronous active-low reset (CLR) and synchronous active-high enable (en): when en is asserted, the value present at the D input is sampled at the rising edge of clk.

4.1.4 Register_N

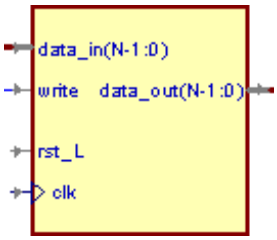


Fig. 4.5 – Symbol of entity Register_N.

Name	Type	Description
N	integer	Number of bits of the register.

Table 4.8 – Generics of entity Register_N.

Name	Direction	# Bits	Active level	Description
data_in	IN	N	-	Data input.
en	IN	1	high	Enable input.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Input clock..
data_out	OUT	N	-	Data output.

Table 4.9 – Description of input and output terminals of entity Register_N.

Register_N is N-bit, D-flip-flop-type register with synchronous active-low reset (rst_L) and synchronous active-high write command (write): if write is asserted, the value present at the data_in input is latched at the rising edge of clk.

4.1.5 Shift Registers

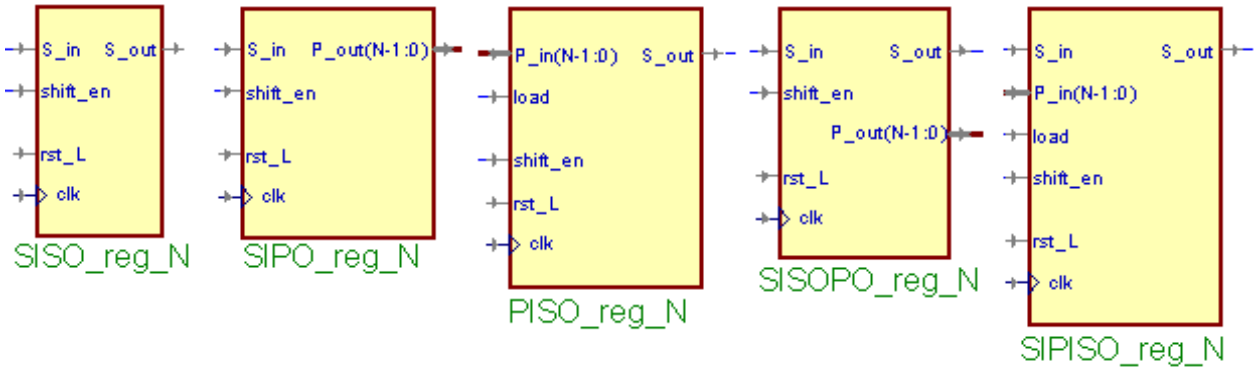


Fig. 4.6 – Symbol of entities SISO_reg_N, SIPO_reg_N, PISO_reg_N, SISOPO_reg_N, SIPISO_reg_N.

Name	Type	Description
N	integer	Number of bits of the register.

Table 4.10 – Generics of entities SISO_reg_N, SIPO_reg_N, PISO_reg_N, SISOPO_reg_N, SIPISO_reg_N.

Name	Direction	# Bits	Active level	Description
S_in	IN	1	-	Serial data input.
P_in	IN	N	-	Parallel data input.
load	IN	1	high	Load command for parallel data input.
shift_en	IN	1	high	Shift enable input.
rst_L	IN	1	low	Synchronous clear input.
clk	IN	1	-	Input clock..
S_out	OUT	1	-	Serial data output.
P_out	OUT	N	-	Parallel data output.

Table 4.11 – Description of input and output terminals of entities SISO_reg_N, SIPO_reg_N, PISO_reg_N, SISOPO_reg_N, SIPISO_reg_N.

SISO_reg_N, SIPO_reg_N, PISO_reg_N, SISOPO_reg_N, SIPISO_reg_N are N-bit shift registers with serial input (SI), parallel input (PI), serial output (SO), or parallel output (PO). They all have synchronous active-low clear input (rst_L) and synchronous active-high shift enable command (shift_en); the shift direction is rightward, from bit N-1 (MSB) to bit 0 (LSB). The shift registers with parallel input also have active-high, synchronous load command (load).

4.2 FF_TX

The transmitter interface FF_TX has the logic internal architecture described in section 3.1.2; the Active-HDL block diagram for this entity, showing also the internal signals that connect the various blocks, is reported in Fig. 4.7.

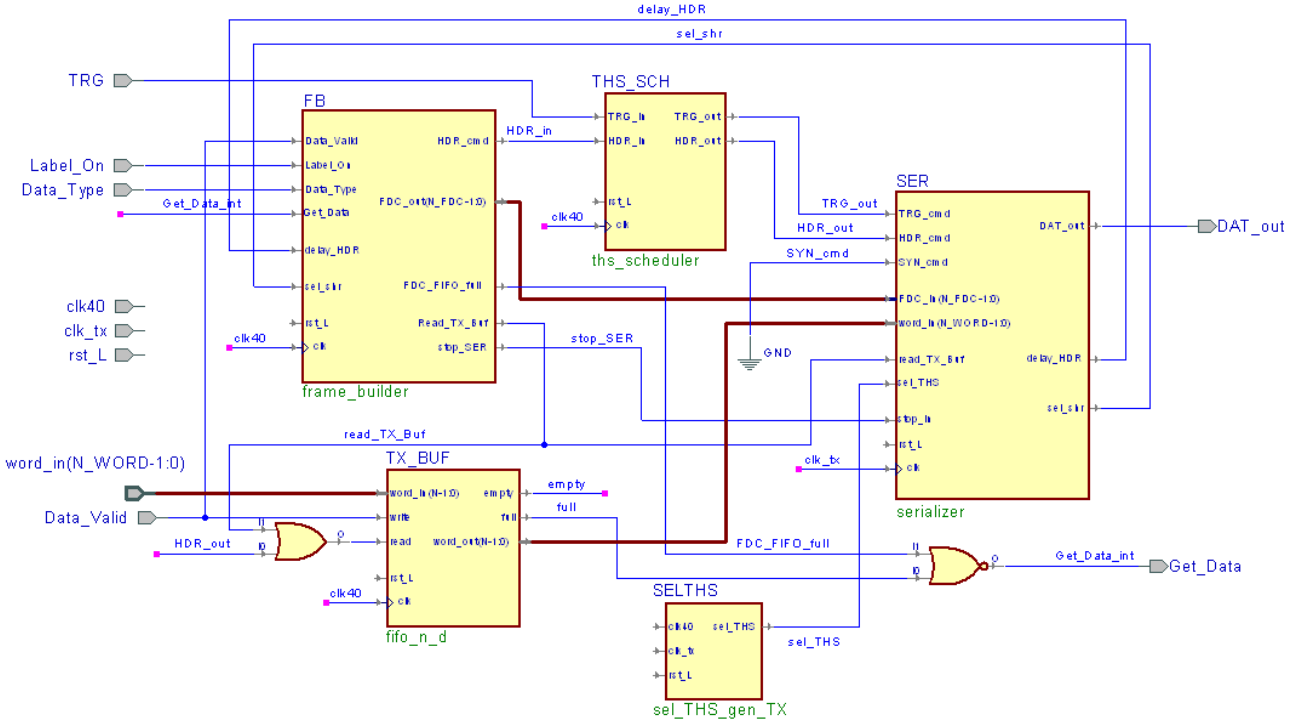


Fig. 4.7 – Block diagram of FF_TX.

As in the logic architecture, the four functional blocks TX Buffer, Frame Builder, THS Scheduler and Serializer are present; the addition here is the auxiliary block called sel_THS Generator TX (entity sel_THS_gen_TX) that has the task of generating the sel_THS signal needed by Serializer (see section 4.2.5).

These blocks are described in more detail hereafter.

4.2.1 TX Buffer

TX Buffer is the FIFO memory that stores the frame words provided by the external circuit awaiting to be transmitted by the TX interface. It is realized with a FIFO_N_D entity (see section 4.1.1): the write command is the Data_Valid input of FF_TX, and the read command is obtained by OR gating the read_TX_Buf output of the Frame Builder and the HDR_out output of the THS Scheduler (see sections 4.2.3 and 4.2.4). It is clocked with the 40 MHz clock (clk40).

4.2.2 sel_THS Generator TX

Fig. 4.8 – Symbol of sel_THS_gen_TX.

Name	Direction	# Bits	Active level	Description
clk40	IN	1	-	40 MHz input clock.
clk_tx	IN	1	-	Transmission input clock.
rst_L	IN	1	low	Synchronous reset.
sel_THS	OUT	1	high	sel_THS signal for the Serializer: when it's high, the Serializer sends the THS channel bits into the output DAT stream.

Table 4.12 – Description of input and output terminals of sel_THS_gen_TX.

This block uses the 40 MHz clock (clk40 input) to generate the sel_THS signal to be used by the Serializer (see section 4.2.5). The internal architecture is shown in Fig. 4.9:

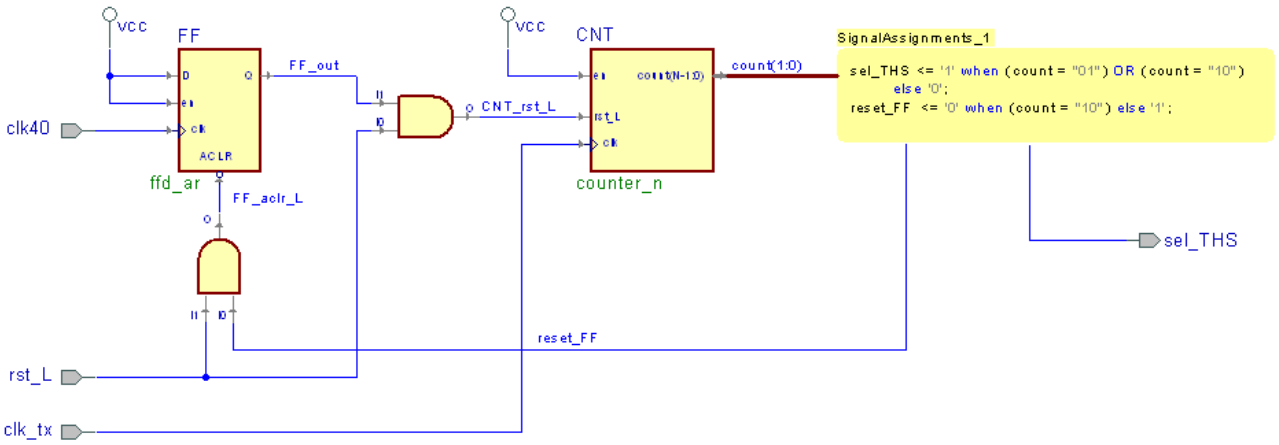


Fig. 4.9 – Internal block diagram of sel_THS_gen_TX.

The component with instance name CNT is an instance of the entity counter_N, that is a N bit synchronous counter (N is an integer generic) with active-low reset (rst_L) and active-high count enable (en). The counter_N entity is used extensively in many parts of the interfaces model; here the actual value of parameter N is 2.

The active low reset rst_L resets the 2-bit counter CNT and FF, that is a D flip-flop with asynchronous, active low reset ACLR.

At the rising edge of the 40 MHz clock the flip-flip output FF_out goes high, thus deactivating the reset of the counter that therefore starts to count at the first clk_tx rising edge after the rising edge of clk40. The sel_THS output is pulled high when the count equals 1 or 2 (i.e. in the second and third clk_tx cycle after the rising edge of clk40). When the count is equal to 2, also, the reset_FF signal is pulled down, and this resets the flip-flop through its asynchronous clear input: the FF output therefore goes low again resetting the counter, so that sel_THS is pulled low again. The circuit has thus returned to its initial state, waiting for the next rising edge of clk40.

A simulation on the sel_THS_gen_TX block is shown in Fig. 4.10 for tx speed = 160, 320 and 640 Mbit/s:

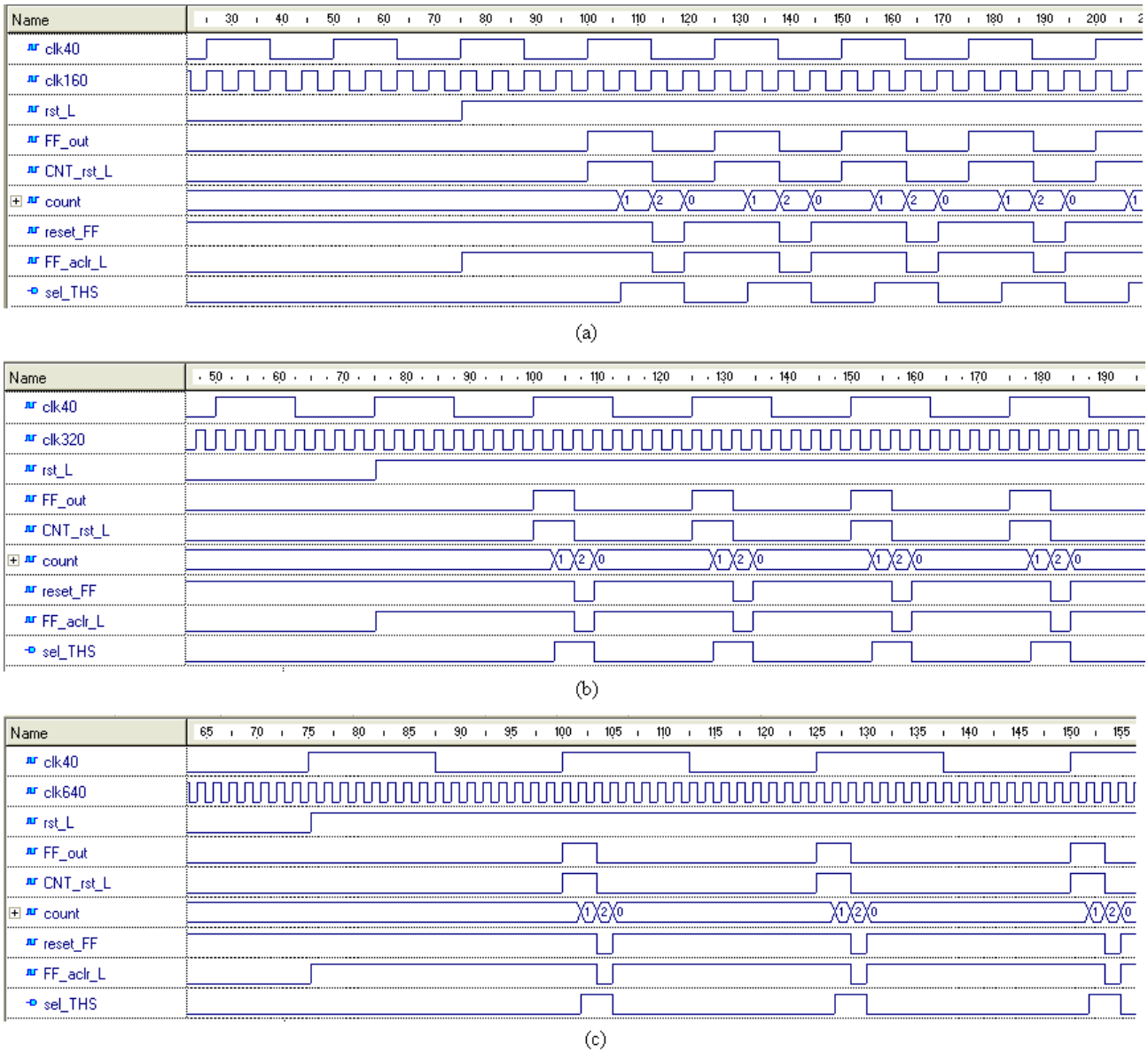


Fig. 4.10 – Simulation showing the behavior of sel_THS_gen_TX for tx speed = 160 (a), 320 (b) and 640 Mbit/s (c).

4.2.3 Frame Builder

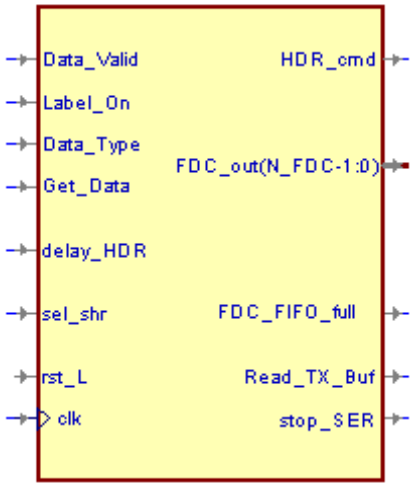


Fig. 4.11 – Symbol of Frame_Builder.

Name	Direction	# Bits	Active level	Description
Data_Valid	IN	1	high	Data_Valid input of FF_TX.
Label_On	IN	1	high	Label_On input of FF_TX.
Data_Type	IN	1	-	Data_Type input of FF_TX.
Get_Data	IN	1	high	Get_Data output of FF_TX; directed to FB_Control1 inside the Frame Builder.
delay_HDR	IN	1	high	Signal generated by Serializer to indicate that a frame transmission is ongoing, and directed to FB_Control2 inside the Frame Builder.
sel_shr	IN	1	-	Signal coming from Serializer, where it indicates the shift register in use (see Serializer description), and directed to FB_Control2.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	40 MHz input clock.
FDC_out	OUT	12	-	Parallel output to send the FDC field to the Serializer.
HDR_cmd	OUT	1	-	HDR command for the THS Scheduler.
Read_TX_Buf	OUT	1	high	Read command for the TX Buffer.
stop_SER	OUT	1	high	Stop command for the Serializer.

Table 4.13 – Description of input and output terminals of Frame_Builder.

This block, working with the 40 MHz clock, prepares the FDC field for each frame stored in the TX Buffer, starts frame transmissions by sending HDR commands to the THS Scheduler and commands the readings of the TX Buffer to provide the Serializer with the frame words during the transmission. The internal architecture is shown in the Fig. 4.12.

The operation of the Frame Builder is controlled by two state machines. The FB_Control1 FSM (FB_CTRL1), using the Word Counter1 (WORD_CNT1) and the FD registers FL_reg, LO_reg, DT_reg and LF_reg (that keep respectively the Frame Length field and the Label On, Data Type and Last Frame bits of the Frame Descriptor), deals with the construction of the FD field of the frame that is being loaded into the TX Buffer; the FD is then encoded by the FD Encoder (FD_ENC) and queued into the FDC FIFO Buffer (FDC_FIFO).

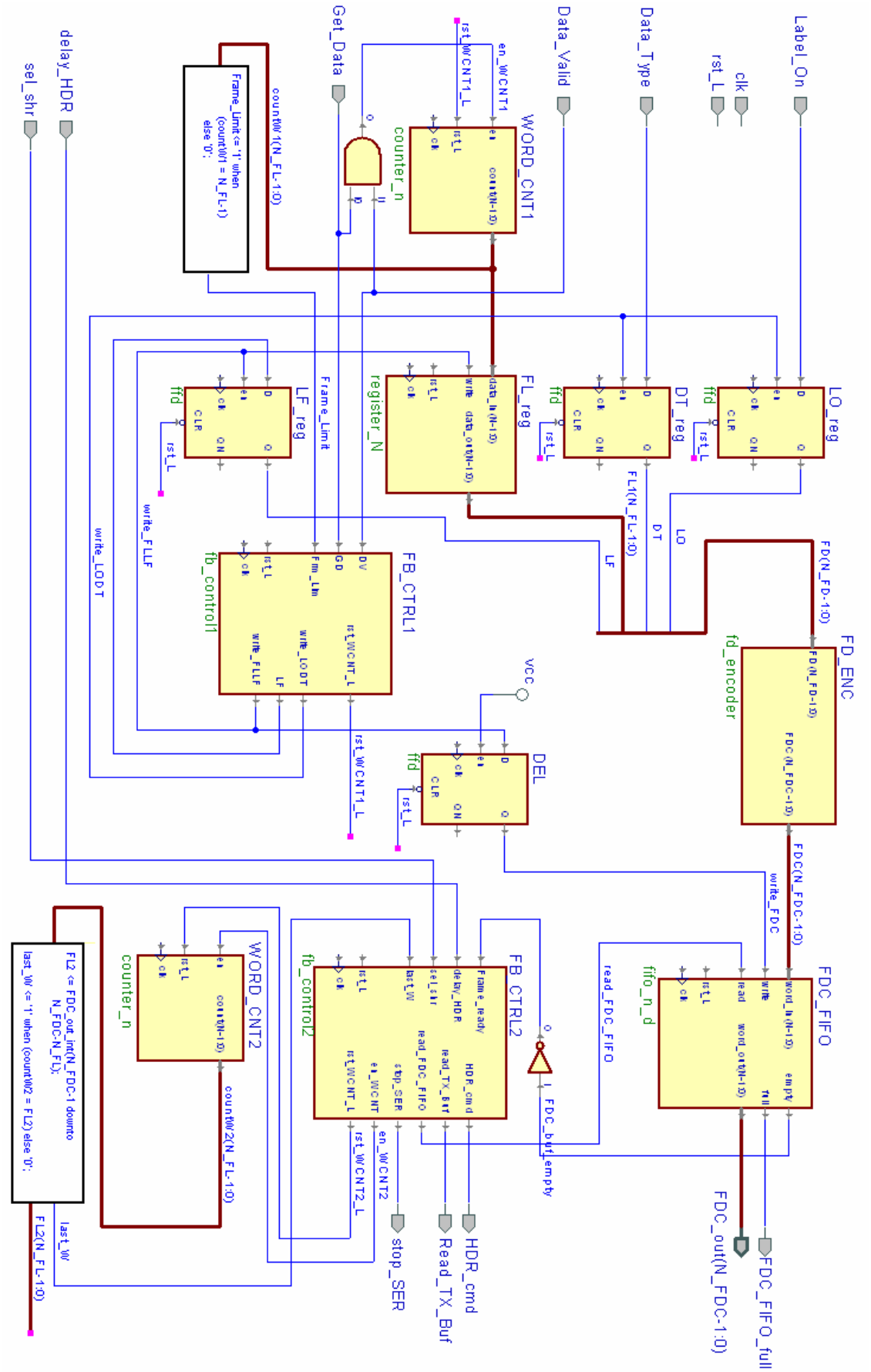


Fig. 4.12 – Internal block diagram of Frame_Builder.

The Word Counter1 is enabled by Data_Valid AND Get_Data, so that it counts only the words that are actually written into the TX Buffer; when all the bits of its output are '1' the Frame_Limit signal is pulled up to indicate that the maximum frame length has been reached. The flip-flop DEL is used to generate the write command for the FDC FIFO (write_FDC) by delaying of a clk40 cycle the write_FLLF signal originated by FB_Control1.

The FB_Control2 FSM (FB_CTRL2) instead has the task of issuing HDR commands (HDR_cmd output) to start frame transmissions, commanding FDC_FIFO readings to send the FDC information to the Serializer and commanding TX Buffer readings (through the Read_TX_Buffer output signal) to send the frame words to the Serializer; it uses the Word Counter 2 (WORD_CNT2) to count the words that has been read from the TX Buffer, so as to determine the end of the frame that is being transmitted by comparison with the Frame Length (signal FL2) available at the output of the FDC_FIFO (the last_W signal is pulled up when countW2 = FL2).

The internal components of Frame Builder are described in further detail hereafter.

4.2.3.1 FB_Control1

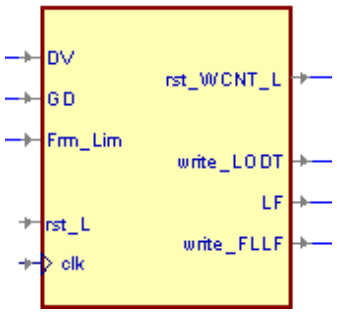


Fig. 4.13 – Symbol of FB_Control1.

Name	Direction	# Bits	Active level	Description
DV	IN	1	high	Data_Valid input.
GD	IN	1	high	Get_Data input.
Frm_Lim	IN	1	high	Connected to the Frame_Limit signal inside the Frame Builder.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	40 MHz input clock.
rst_WCNT_L	OUT	1	low	Reset command for Word Counter 1
write_LODT	OUT	1	high	Write command for LO and DT registers.
LF	OUT	1	high	Imposes the value of the Last Frame bit in the LF register.
write_FLLF	OUT	1	high	Write command for FL and LF registers.

Table 4.14 – Description of input and output terminals of FB_Control1.

This block is a Mealy machine that controls the Word Counter 1 and generates the write commands for the FD registers and for the FDC FIFO. The state diagram is reported in Fig. 4.14.

At the reset, the FSM is in the IDLE state resetting the Word Counter 1 (rst_WCNT_L <= '0'). When a '1' on the DV (Data_Valid) input is detected, meaning that the host has started a packet delivery, FB_Control1 moves to the state PKT_LOADING: on this transition the FSM starts the Word Counter by removing its reset and activates the write_LODT command, that in the Frame Builder writes the value of the Label_On and Data_Type inputs into the corresponding FD registers LO_reg and DT_reg (the write_LODT output of FB_Control1 is not registered to allow the sampling of the Label On and Data Type inputs on the first cycle of the frame loading, i.e. when DV goes high).

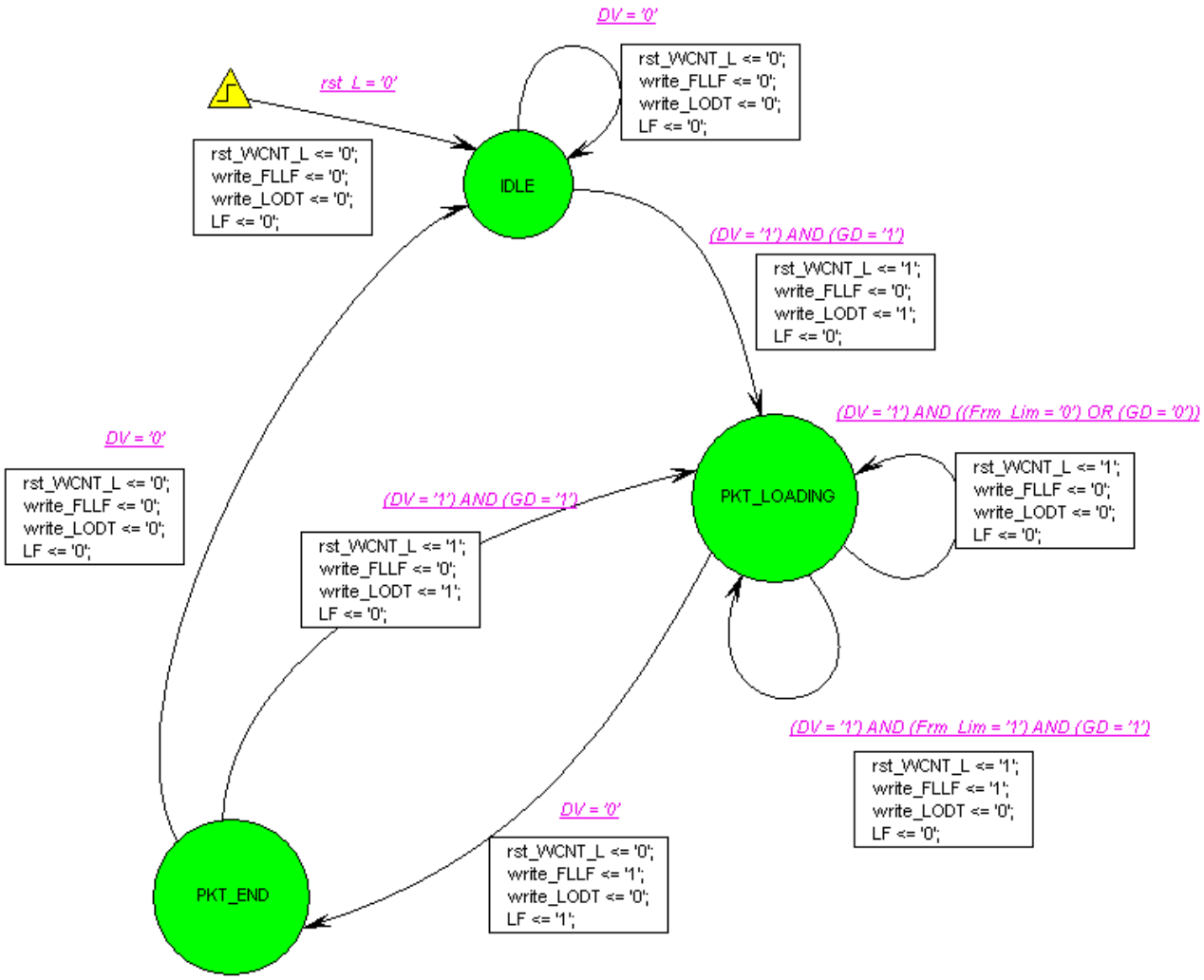
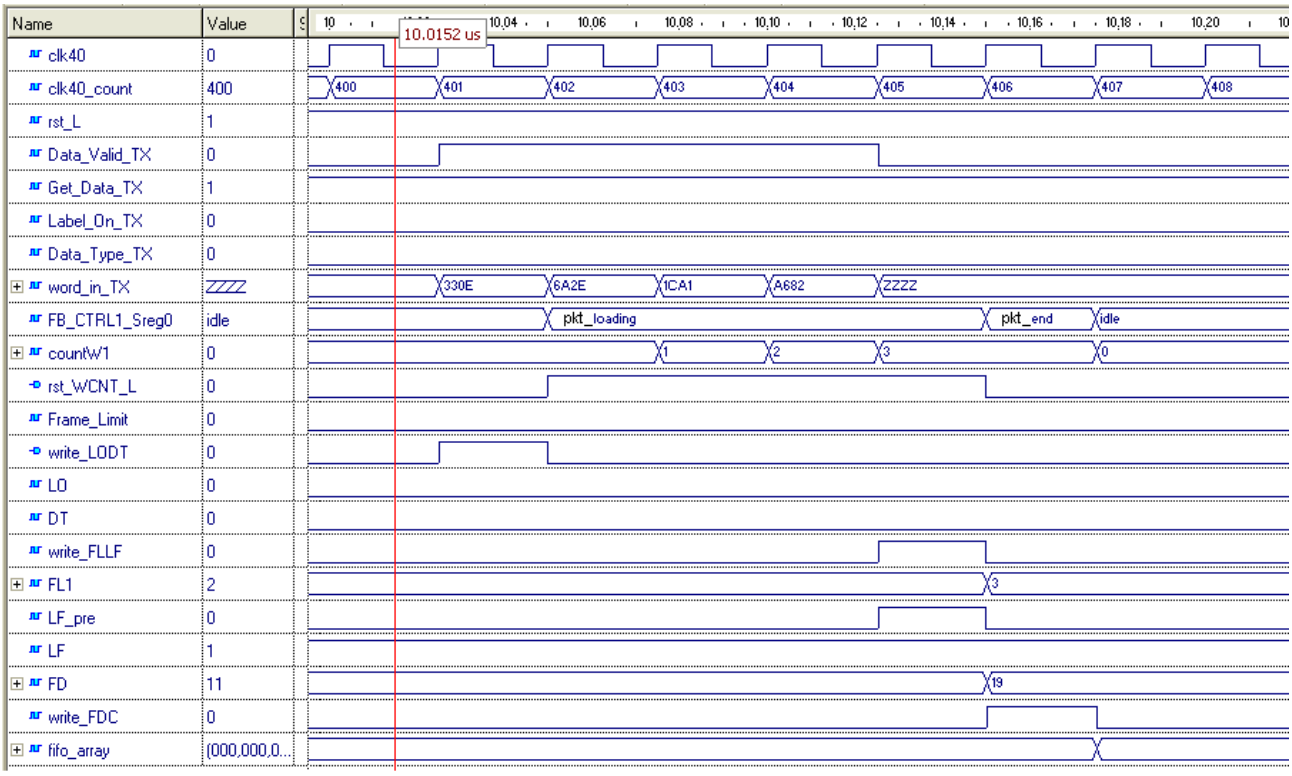


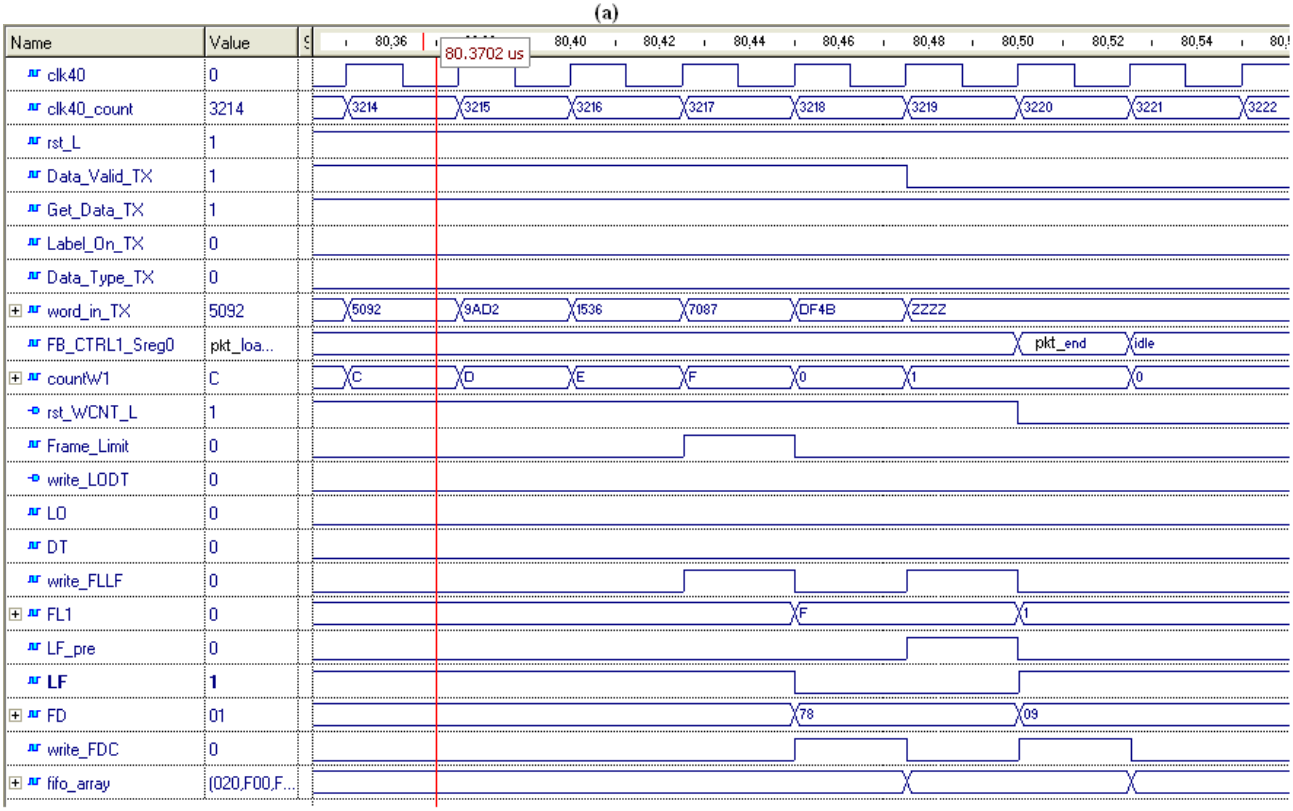
Fig. 4.14 – State diagram of FB_Control1.

The FSM remains in the state PKT_LOADING as long as DV = '1', i.e. until the end of the packet, carrying on the counting of the words: since the Word Counter 1 is enabled by the signal Data_Valid AND Get_Data, only the words that are actually written into the TX Buffer are counted (that is, the words that are proposed by the host in the clock cycles when Get_Data is high).

Consider first the case in which GD stays high during all the packet loading. When DV returns to 0, indicating that the packet loading in the TX Buffer is completed, the FSM goes to the state PKT_END resetting the Word Counter and activating the write_FLLF command: this writes into the FL register the value reached by the Word Counter before its reset, equal to the number of clk40 cycles that DV has been high (diminished by one to conform to the convention "number of words of the frame = value of FL +1"). The write_FLLF command also stores in the LF register the value of the Last Frame bit imposed by FB_Control1 itself through the LF output: on the passage from PKT_LOADING to PKT_END, LF is set to 1 since a transition of DV from 1 to 0 always correspond to the ending of a data packed, that hence won't be fragmented into further frames. If instead the maximum number of words in the frame is reached (i.e. the Frame_Limit signal goes high) while DV is still asserted, FB_Control1 sets LF to '0', writes this value in the LF register by activating the write_LFFL command (thus writing also the Frame Length, now equal to the maximum possible value "111....111", into the FL register) but remains in the PKT_LOADING state to continue the loading of the packet words as a new frame. The word counter doesn't need to be reset in this case because it has reached its maximum value and so it will start again from zero with the first word of the new frame.



(a)



(b)

Fig. 4.15 – Simulation showing the operation of FB_Control1 in case of loading of a 4-word data packet (a) and of a 17-word data packet (b). The signal named "fifo_array" is the content of the FDC FIFO.

Now let's consider the possibility of Get_Data going low during the loading of a packet, i.e. when FB_Control1 is in the state PKT_LOADING and DV is asserted. If this happens for, say, one clk cycle, the word proposed by the host in that cycle will not be written in the TX Buffer and will not be counted by the Word Counter 1; the condition (DV = '1') AND ((Frm_Lim = '0') OR (GD = '0'))

will be true and therefore FB_Control1 will remain in the state PKT_LOADING without activating the write_FLLF command. This behavior is correct because, even if the frame limit has been reached and so Frm_Lim (Frame_Limit) is high in this cycle, the first word of the second frame of the fragmented packet hasn't been written yet in the TX Buffer. The fragmentation, i.e the activation of the write_FLLF command, must be carried out only when the condition (DV = '1') AND (Frm_Lim = '1') AND (GD = '1') is true: this corresponds to the second loop transition of the state PKT_LOADING in the FB_Control1 state diagram.

At the next clock cycle after the arrival into the PKT_END state, if DV is still low the FSM goes back to the IDLE state keeping the Word Counter in reset; if instead DV has been set again, FB_Control1 moves directly to the PKT_LOADING state activating the write_LODT command again and starting the Word Counter to begin the FL calculation for the new frame.

Being the write_FDC command within the Frame Builder originated by delaying the write_FLLF command, the FDC field (calculated by the Hamming Encoder) is written into the FDC_FIFO one (40MHz) clock cycle after the writing of FL_reg and LF_reg, that is the moment when the complete FD is available as output of the FD registers.

The operation of FB_Control1 was tested with packets of length ranging from 0 to 39, to verify the correctness of the fragmentation function. Fig. 4.15 shows two examples of the evolution of the FB_Control1 signals: in the first case a 4-word data packet is loaded, causing the creation of a single frame with the LF bit set to 1 in the Frame Descriptor; in the second case the loading of a 17-word data packet gives rise to its fragmentation in two consecutive frames, the first with LF = 0 and the second with LF = 1.

Simulations where performed also for the case of Get_Data going low (with various patterns) during the packet loading. Fig. 4.16 shows as an example the "critical" case of GD going low in the cycle in which the Frame_Limit signal goes high: as it can be seen, the fragmentation (write_FLLF command) is postponed to the next clk cycle, when GD is pulled up again.

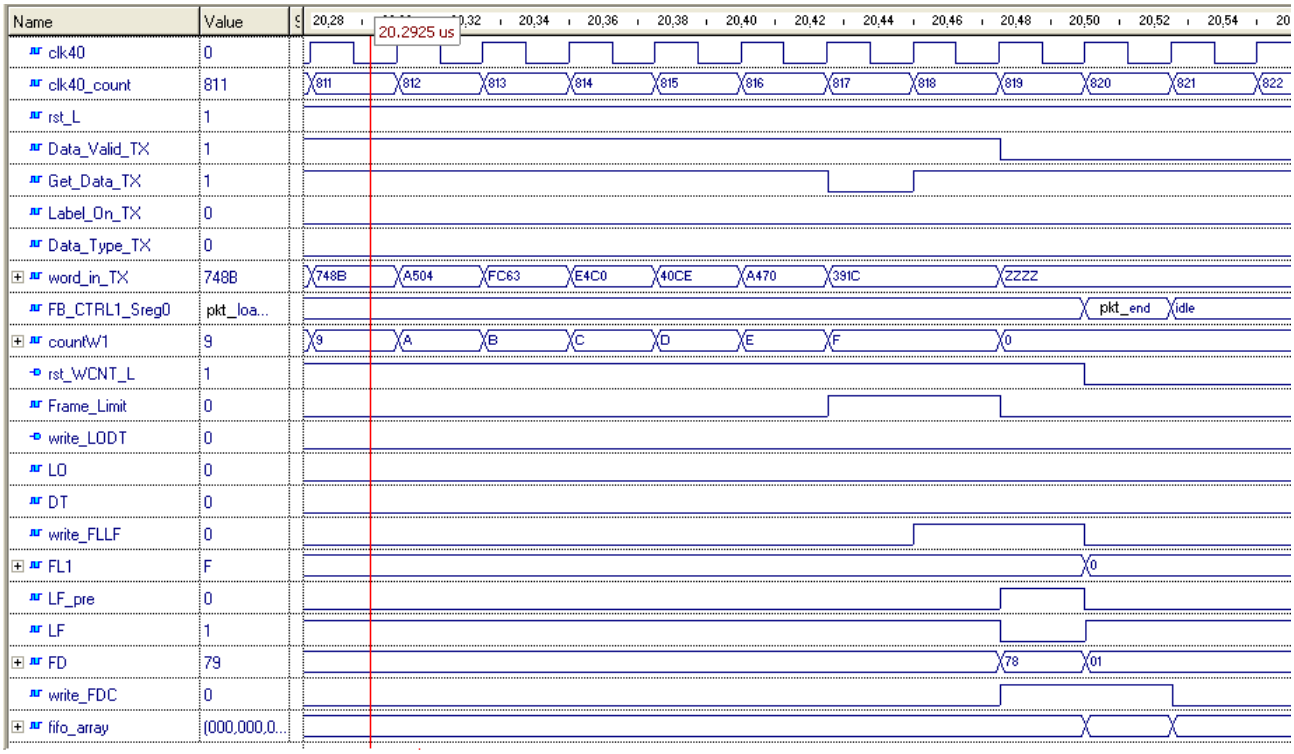


Fig. 4.16 – Simulation showing the operation of FB_Control1 during a packet loading, in the case of GD going low in the cycle in which the Frame_Limit signal goes high.

4.2.3.2 FB_Control2

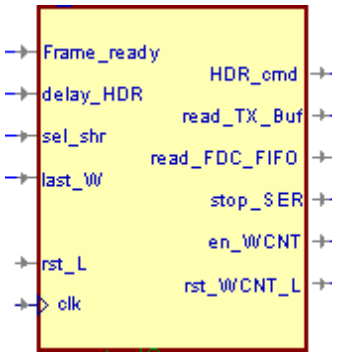


Fig. 4.17 – Symbol of FB_Control2.

Name	Direction	# Bits	Active level	Description
Frame_ready	IN	1	high	Signals that the FDC_FIFO is not empty.
delay_HDR	IN	1	high	Signal coming from Serializer and used by FB_Control2 to determine the moment when a HDR command can be sent.
sel_shr	IN	1	high	Signal coming from Serializer: indicates to FB_Control2 when the transmission of each frame word is finished.
last_W	IN	1	high	Last_W signal generated from the output of WCNT2.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	40 MHz input clock.
HDR_cmd	OUT	12	low	HDR command for the THS Scheduler.
read_TX_Buf	OUT	1	high	Read command for the TX Buffer.
read_FDC_FIFO	OUT	1	high	Read command for the FDC_FIFO.
stop_SER	OUT	1	high	Stop command for the Serializer.
en_WCNT	OUT	12	low	Enable command for Word Counter 2
rst_WCNT_L	OUT	1	low	Reset command for Word Counter 2.

Table 4.15 – Description of input and output terminals of FB_Control2.

This entity is a Mealy machine that reads the FDC FIFO (read_FDC_FIFO output command), generates the HDR commands (HDR_cmd), issues the read commands for the TX Buffer (read TX_Buf) and controls the Word Counter 2; besides, it generates the "stop" signal for the Serializer (stop_SER) at the end of a frame transmission. The state diagram is shown in Fig. 4.18.

At the reset, the FSM begins it activity in the READY state, waiting for the Frame_ready signal. When a frame is ready to be transmitted, its FDC is written in the FDC_FIFO and thus the Frame_ready signal (that is the negation of the FDC_FIFO empty signal) goes to '1': this event, if the delay_HDR signal is low (meaning that the THS channel is free and the Serializer has completed any previous frame transmission, see Serializer description), causes the FB_Control2 to move to the START_FRAME state issuing a HDR_cmd to start the frame transmission in the Serializer.

The HDR command will be processed by the THS Scheduler and will then arrive to the Serializer, that as a consequence will start the transmission of the frame words alternating the use of its two shift registers by means of the sel_shr signal (see Serializer description): FB_Control2 uses this signal to determine the end of each word transmission, thus resolving the problem of the interfacing between the 40 MHz-clocked Frame Builder and TX Buffer and the clk_tx-clocked Serializer (note that sel_shr remains in each state, low or high, for at least one clk40 cycle, providing that the number of bits of a frame word is equal or greater than the number of FRM bits transmitted in each clk40 cycle; with N_WORD = 16, this is true for all the three considered values of the tx speed).

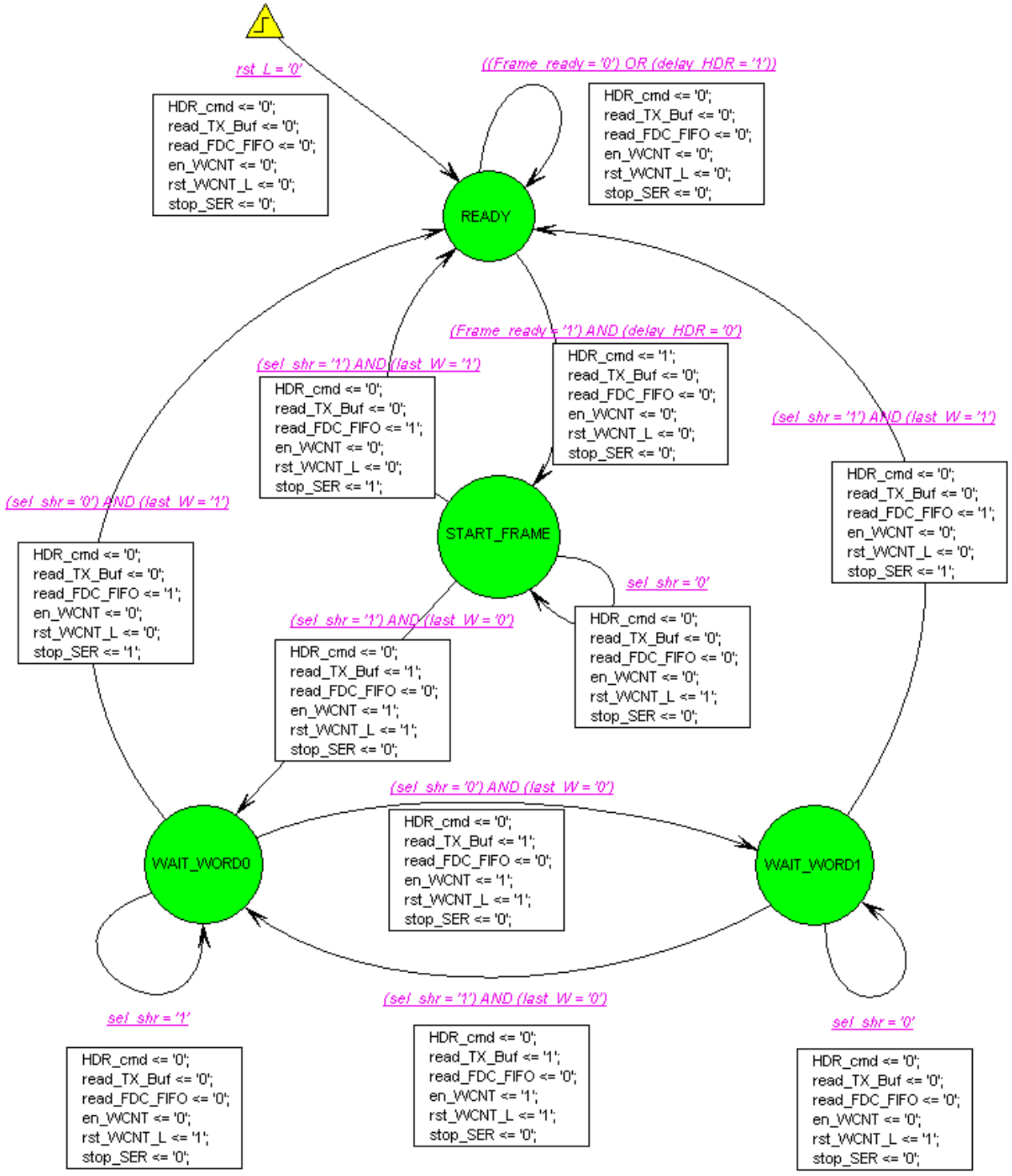


Fig. 4.18 – State diagram of FB_Control2.

Hence, FB_Control2 remains in the START_FRAME state as long as sel_shr is low, which means that the Serializer is still transmitting the FDC field of the frame; when the FDC transmission is finished sel_shr goes high and this in turn (supposing that the frame consist in more than one word, so that the "last word" flag last_W is low at this time) causes FB_Control2 to move to the state WAIT_WORD0. On this transition the FSM commands a reading of the TX Buffer to load the next frame word into the Serializer.

Once in the state WAIT_WORD0, the FSM waits that the Serializer has finished transmitting the first word of the frame. When this happens, sel_shr goes low again and FB_Control2 moves to the state WAIT_WORD1 commanding another reading of TX Buffer and incrementing the value of the

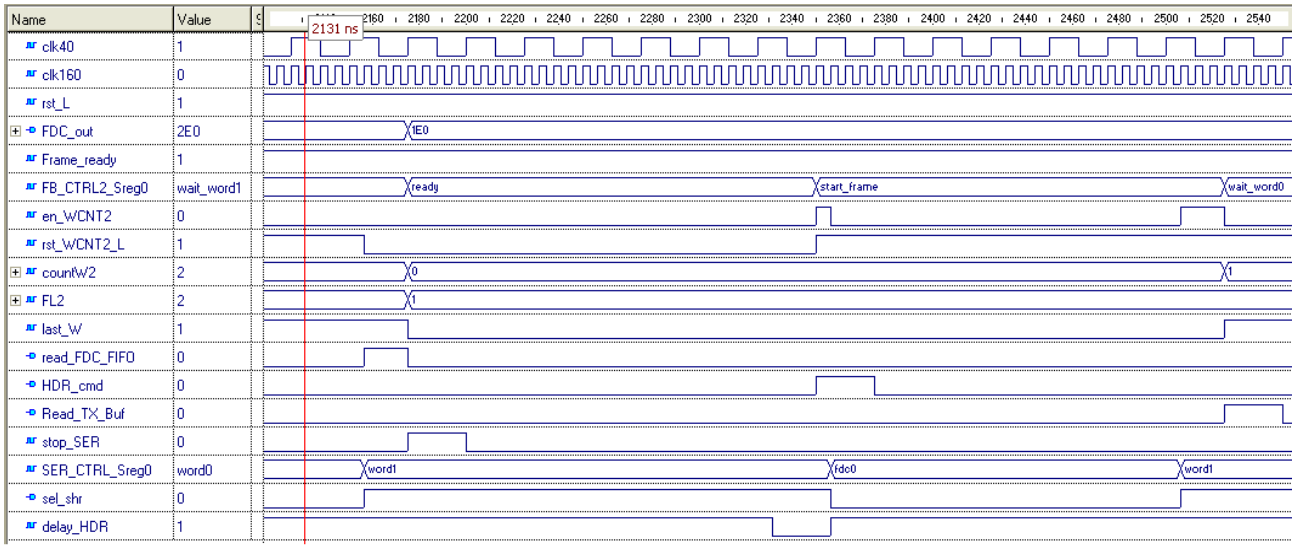
Word Counter 2 by activating its enable (en_WCNT <= '1'). In the state WAIT_WORD1 the FSM waits for sel_shr to go high again (indicating that another word's transmission has been completed), and when this happens it moves back to the state WAIT_WORD0 reading another word from TX Buffer and incrementing again the word counter.

The alternation between WAIT_WORD0 and WAIT_WORD1 goes on until the flag last_W goes high, that happens when the value of the Word Counter 2 reaches the value of the signal FL2, which is the Frame Length of the frame that is currently being transmitted. When, being FB_Control2 in the state WAIT_WORD0/1, the signal sel_shr changes state and last_W is high, the FSM returns into the READY state because the Serializer is now sending the last word of the frame. On this transition a reading of the FDC_FIFO is commanded (read_FDC_FIFO <= '1') so that FB_Control2 knows, once returned into the READY state, if there are more frames ready to be sent: if this is the case the signal Frame_ready will be high and so FB_Control2 will start a new frame transmission (moving again into the state START_FRAME and issuing a new HDR command) as soon as the signal delay_HDR will be detected low, meaning that the Serializer has completed the transmission of the last word of the preceding frame. On the transition from the state WAIT_WORD0/1 to the READY state the stop_SER output command is activated too: it is used to command the Serializer to return back to its IDLE state, thus stopping the transmissions in the FRM channel, at the end of each frame.

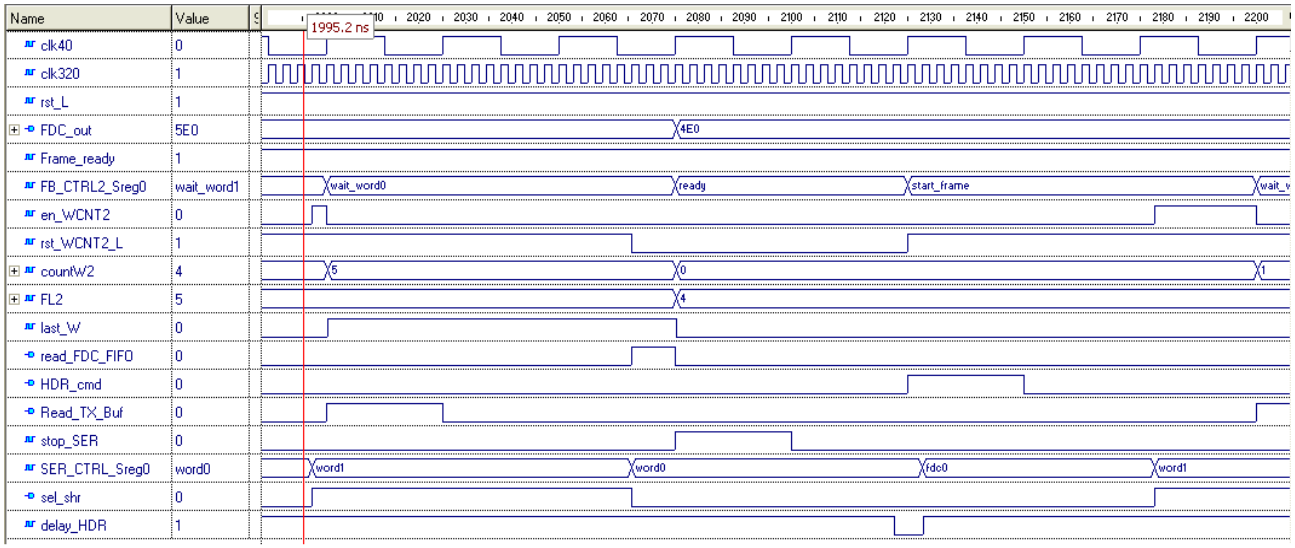
In the case of a single-word frame, the signal FL2 is equal to 0 and consequently the signal last_W is high when FB_Control2 is in the state START_FRAME: hence, when sel_shr is detected as high, meaning that the Serializer has finished sending the FDC field and is now transmitting the first (and last) word of the frame, FB_Control2 goes back directly to the READY state commanding a reading of the FDC_FIFO and issuing a stop_SER command.

The HDR_cmd output is registered because the HDR command must be issued when FB_Control2 enters the START_FRAME state and not before; the read_TX_Buffer output is registered too, because this signal has also the effect of loading the shift registers inside the Serializer and so it seemed more safe (for possible post synthesis timing problems) to load them in the middle of their "idle" interval (i.e. the interval during which a shift register is not enabled to shift out) rather than at its beginning, i.e. at the first tx_clk edge after the transition of sel_shr. On the contrary, read_FDC_FIFO and the word counter control signals (en_WCNT and rst_WCNT_L) are not registered: indeed, if read_FDC_FIFO were registered, the FB_Control2 would know if there are other frames in the FDC_FIFO only one clk40 cycle after the detection of last_W = '1', and at tx speed = 640 Mbit/s this is too late because there is only one clk40 cycle from the detection of last_W = '1' and the detection of delay_HDR = '0', that marks the clk40 edge at which a possible new HDR_cmd must be issued. As for en_WCNT and rst_WCNT_L, they must have their effect on the clk40 edge at which FB_Control2 changes state (moving into the state WAIT_WORD0/1) to have the words counted properly, ad so they must be not registered.

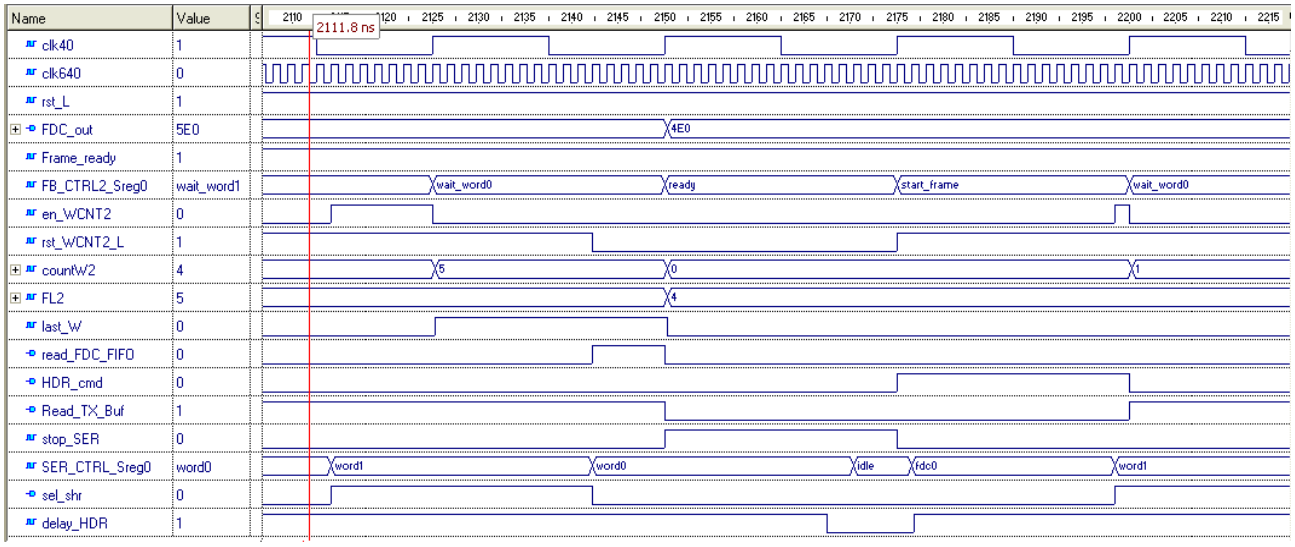
An example of the evolution of the FB_Control2 signals at the end of a frame transmission (with the beginning of a new transmission immediately after) is shown in Fig. 4.19 for the three values of the tx speed.



(a)



(b)



(c)

Fig. 4.19 – Simulation showing the operation of FB_Control2 at the end of a frame transmission with the beginning of a new transmission immediately after; tx speed = 160 (a), 320 (b) and 640 Mbit/s (c).

4.2.3.3 FD Encoder

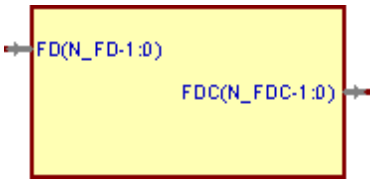


Fig. 4.20 – Symbol of FD_Encoder.

Name	Direction	# Bits	Active level	Description
FD	IN	7	-	Uncoded Frame Descriptor.
FDC	OUT	12	-	Coded Frame Descriptor.

Table 4.16 – Description of input and output terminals of FD_Encoder.

This block performs the H(12,7) encoding on the Frame Descriptor (FD input) producing the coded FD to be included into the frame for transmission (FDC output). Its architecture is defined by the encoding algorithm described in section 2.3.1.1: the 7 most significant bits of FDC are the bits of FD without changes, while the remaining FDC(4), ..., FDC(0) are the five parity bits calculated according to equations (2.22).

4.2.4 THS Scheduler

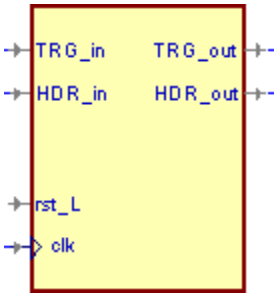


Fig. 4.21 – Symbol of THS_Scheduler.

Name	Direction	# Bits	Active level	Description
TRG_in	IN	1	high	TRG input command.
HDR_in	IN	1	high	HDR input command.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	40 MHz input clock.
TRG_out	OUT	1	high	TRG output command.
HDR_out	OUT	1	high	HDR output command.

Table 4.17 – Description of input and output terminals of THS_Scheduler.

This block, clocked by the 40 MHz clock, manages TRG and HDR commands, arriving through the TRG_in and HDR_in inputs respectively from TRG input port of FF_TX and from the Frame Builder, and passes them to the Serializer through the TRG_out and HDR_out outputs: the task of the THS_Scheduler is to organize the arrival of the TRG and HDR commands to the Serializer in such a way that the transmission of the THS sequences never overlap, i.e. there is always an interval

of at least 3 clk40 cycles (i.e. the duration of the THS sequences) between two successive THS commands.

The scheduling strategy is the one that was described in section 2.2.1: when a TRG_in command arrives, the THS_Scheduler starts the forbidden window of 5 clock cycles and generates the TRG_out pulse at the third clock cycle of this window; if a HDR_in command arrives inside the forbidden interval, it is put on hold and the corresponding HDR_out pulse is generated at the end of the interval, as shown in Fig. 4.22. (To be more precise, the Serializer simply ignores THS commands that arrive when a preceding THS sequence's transmission is still ongoing: hence, the arrival of a HDR command in the forbidden interval would not lead to an overlap with the TRG sequence but would cause the loss of the TRG command or of the HDR command itself, depending on which is the last of the two).

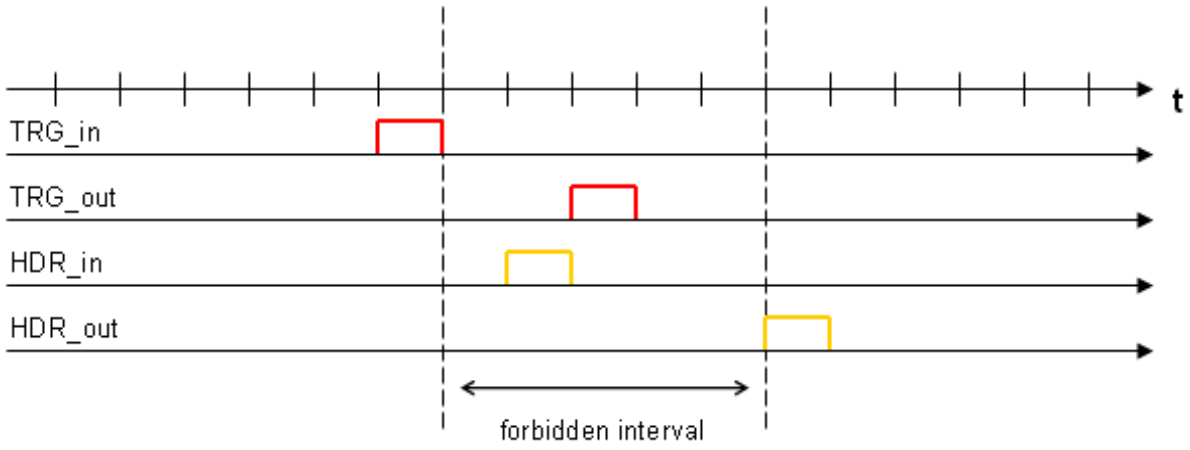


Fig. 4.22 – Scheduling of TRG and HDR commands: if a HDR_in command arrives inside the 5-clock-cycles "forbidden window", the corresponding HDR_out command is issued at the end of this window.

The internal architecture of the THS_Scheduler is shown in Fig. 4.23:

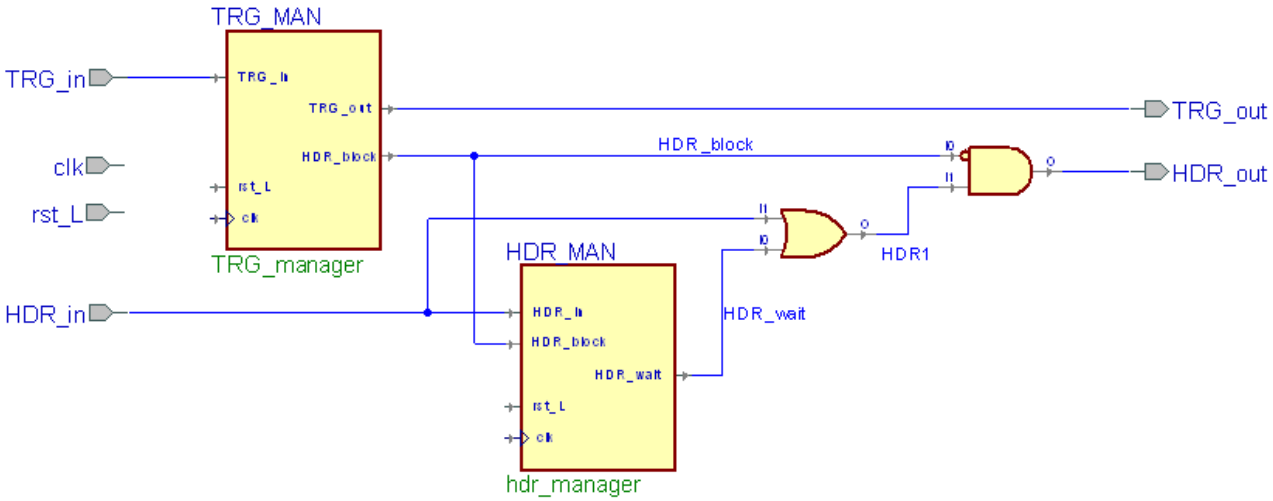


Fig. 4.23 – Internal block diagram of THS_Scheduler.

TRG_manager is a registered-output Mealy machine with the state diagram shown in Fig. 4.24. Being in the IDLE state at the reset, when a '1' on the TRG_in input arrives (TRG command pulse) the FSM starts to move at every next clk cycle through the states TRG1, ... , TRG5, setting the output HDR_block to '1': this signal is the one that determines the forbidden interval, stopping a possible HDR command as will be explained next. On the transition between TRG2 and TRG3 the

TRG_out pulse is generated. After completing the cycle in TRG5, the FSM returns to the IDLE state, but if a second TRG_in command arrives while the FSM is in TRG3, TRG4 or TRG5, the machine jumps directly to the state TRG1 to extend the forbidden window (HDR_block = '1') and produce another TRG_out pulse once arrived again to the TRG3 state.

Considering again the THS_Scheduler architecture, if a HDR_in command arrives in a clk cycle that doesn't belong to the forbidden interval, the HDR_block signal will be low and the HDR command will be passed through the OR and the AND gates generating a pulse on HDR_out in the same clk cycle. If, on the contrary, the HDR_in pulse occur in the forbidden window, it will be blocked by the AND gate being HDR_block = '1' and the HDR_manager will intervene to generate a HDR_out pulse at the end of the forbidden interval.

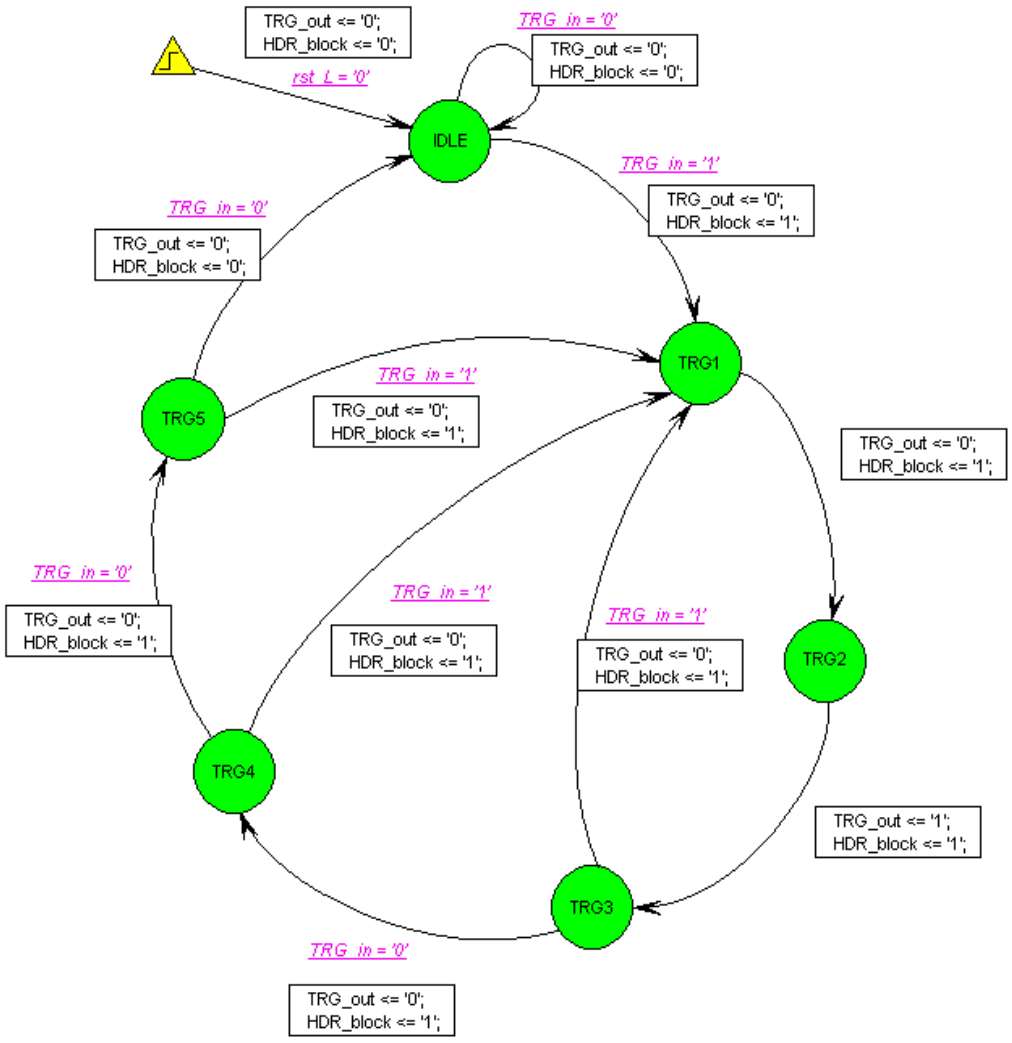


Fig. 4.24 – State diagram of TRG_manager.

HDR_manager is a registered-output Mealy machine with the state diagram shown in Fig. 4.25.

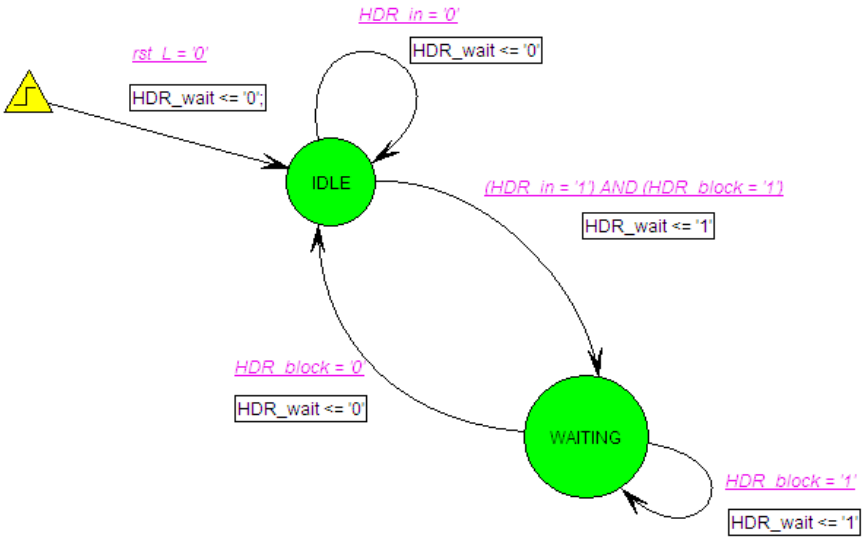


Fig. 4.25 – State diagram of HDR_manager.

This FSM stays in the IDLE state until a HDR_in command arrives and, at the same time, HDR_block = '1', meaning that the HDR_in command has been stopped: when this happens, the HDR_manager moves to the WAITING state and sets HDR_wait to '1'. This signal is used to extend, by means of the OR gate following the HDR_manager, the HDR pulse until one clk cycle after the end of the forbidden interval, and indeed it is reset when the HDR_manager leaves the WAITING state having detected the resetting of HDR_block.

The simulation shown in Fig. 4.26 illustrates the mechanism of TRG and HDR command management described above:

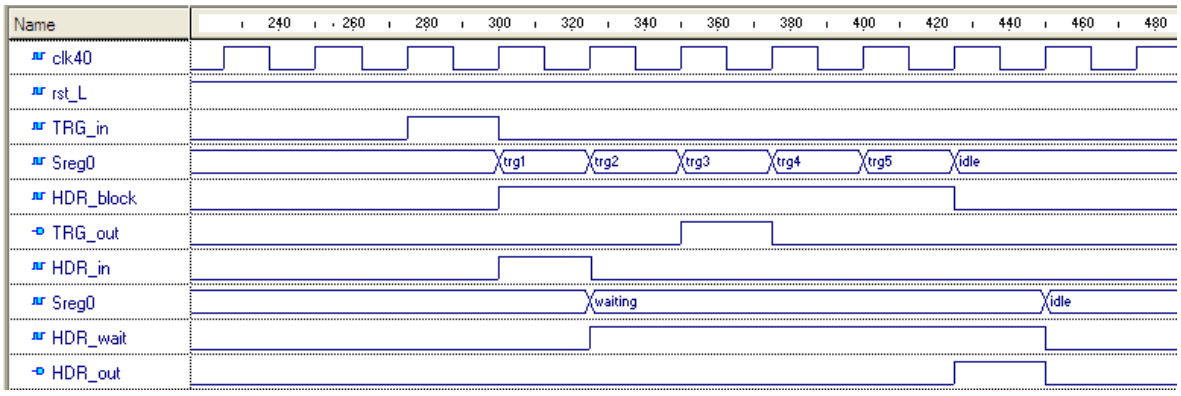


Fig. 4.26 – Simulation showing the operation of the THS Scheduler .

4.2.5 Serializer

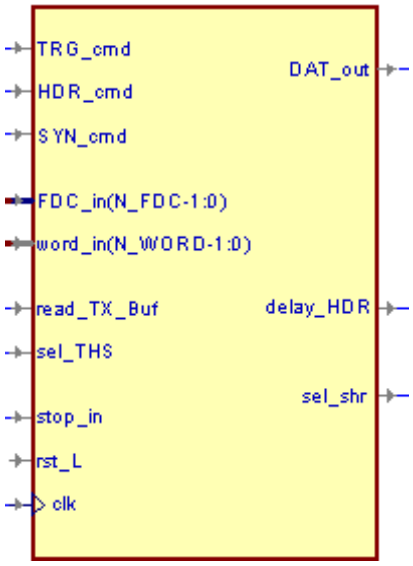


Fig. 4.27 – Symbol of Serializer.

Name	Direction	# Bits	Active level	Description
TRG_cmd	IN	1	high	Commands the transmission of a TRG sequence.
HDR_cmd	IN	1	high	Commands the transmission of a HDR sequence.
SYN_cmd	IN	1	high	Commands the transmission of a SYN sequence.
FDC_in	IN	12	-	Input for the FDC field coming from the Frame Builder.
word_in	IN	16	-	Input for data words coming from the TX Buffer.
read_TX_Buf	IN	1	high	Read command for the TX Buffer.
sel_THS	IN	1	high	Signal coming from the sel_THS Generator: identifies the THS channel bits inside each clk40 period.
stop_in	IN	1	high	Stop signal for Serializer coming from the Frame Builder.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input, connected to the transmission clock (clk_tx).
DAT_out	OUT	1	-	DAT serial stream output.
delay_HDR	OUT	1	high	Signal directed to Frame Builder to indicate that a frame transmission is ongoing.
sel_shr	OUT	1	-	Signal directed to Frame Builder to indicate which shift register is in use inside the Serializer.

Table 4.18 – Description of input and output terminals of Serializer.

This component is clocked by the transmission clock clk_tx (160, 320 or 640 Mbit/s) and produces the DAT serial stream with the FRM channel and the THS channel. The internal architecture is shown in Fig. 4.28:

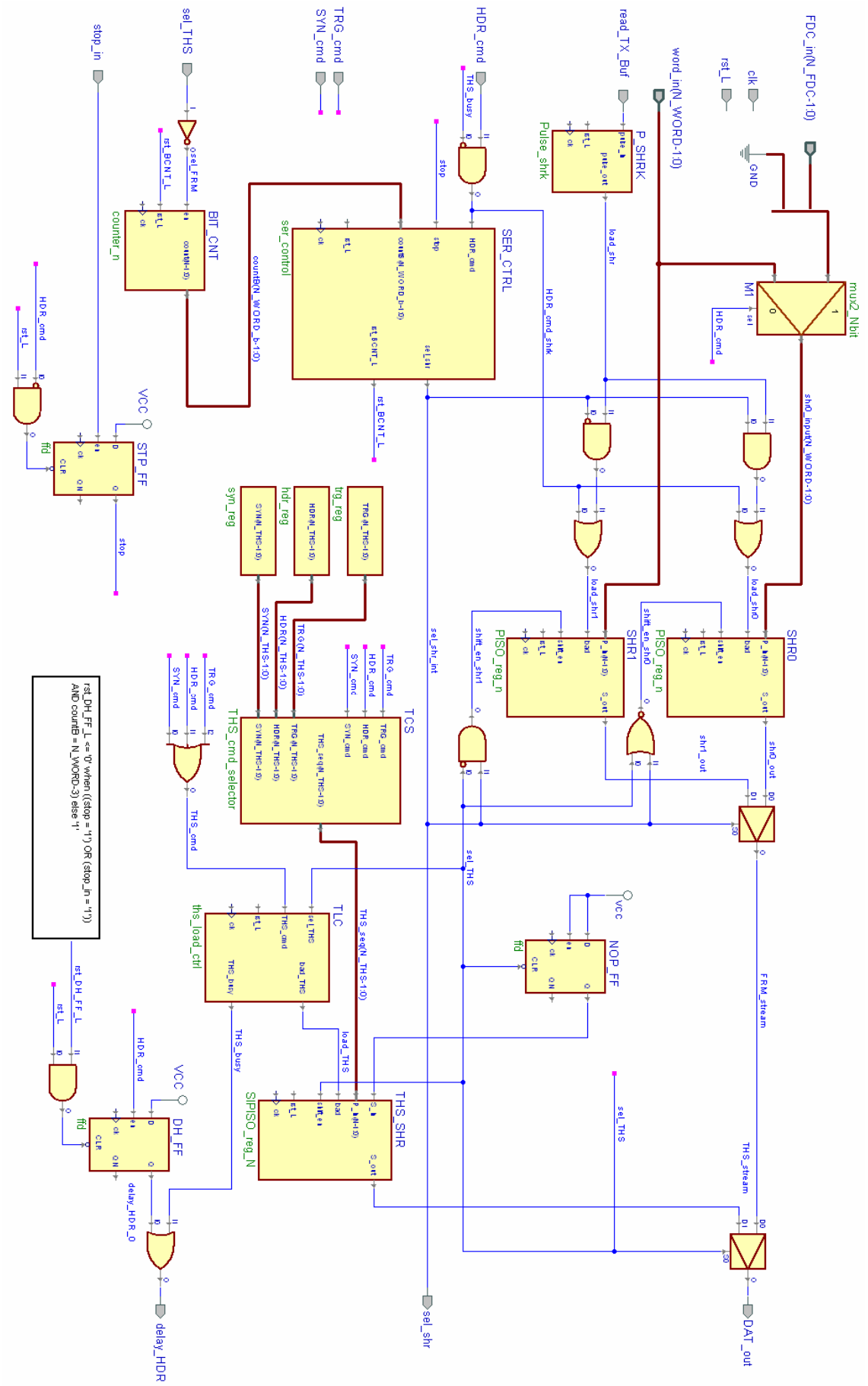


Fig. 4.28 – Internal block diagram of Serializer.

The DAT_out stream is provided by a mux that switches between the THS and the FRM streams, controlled by the sel_THS signal that is high in the THS part of each clk40 cycle, and low in the FRM part: the sel_THS signal arrives to the Serializer from the Sel_THS Generator.

The THS stream is produced by the THS shift register (THS_SHR) that is loaded with the THS command sequence that has to be transmitted and is enabled to shift out its content in the THS part of each clk40 cycle, being enabled by the sel_THS signal. The THS sequences are kept in three registers (TRG_reg, HDR_reg and SYN_reg): the sequence to be loaded in the THS_SHR is chosen by the THS_cmd_selector, while the load command (load_THS) is generated by the THS_load_ctrl block when a pulse on the THS_cmd input occur. The leftmost bit of THS_SHR is shifted in from the output of the flip-flop NOP_FF, that is reset when the sel_THS signal goes low and therefore produces the NOP pattern 0101010101... into the THS stream. The encoding that was implemented in this version of the Serializer is the TRG/HDR balanced encoding (described in section 2.2.3.3) that doesn't have a SYN command, but the possibility of transmitting a third THS sequence as a "spare" command besides TRG and HDR was left in this Serializer: however, the SYN_cmd input of Serializer is connected to ground (see Fig. 4.7).

The FRM stream is produced by the two shift registers SHR0 and SHR1, that are loaded alternatively with the words to be transmitted into the FRM channel (coming from the Frame Builder through the FDC_in input and from the TX Buffer through the word_in input) and alternatively are shifted out in the FRM part of each clk40 cycle. In principle, a single shift register for the FRM stream could have been used, but in the 320 and 640 Mbit/s configurations the transmission of a word can finish in the FRM part of the clk40 cycle and so the shift register should be loaded with the new word while the shifting is ongoing: this can cause timing problems that can be eliminated with the choice of using two shift registers, and loading one of them while the other is being shifted out and vice versa.

The source of the FRM stream is chosen between SHR0 and SHR1 by the mux controlled by the sel_shr signal. This latter also determines, together with the sel_THS signal, the intervals in which the two shift registers are enabled to shift: the shift enable for SHR0 (shift_en_shr0) is high when sel_THS is low (i.e. in the FRM part of the clk40 cycle) and sel_shr is low (i.e. when SHR0 is selected as the source of the FRM stream), while the shift enable for SHR1 (shift_en_shr1) is high when sel_THS is low and sel_shr is high (i.e. when SHR1 is the source of the FRM stream).

The loading of SHR0 and SHR1 is controlled by the load enable signals load_shr0 and load_shr1, that are generated from the HDR_cmd and Read_TX_Buf inputs. At the beginning of a frame transmission, a clk40 cycle-long pulse arrives on the HDR_cmd input: from this pulse a clk_tx cycle-long pulse is obtained by AND-gating it with the THS_busy signal generated by the THS load Control block (see its description below), and this "shrunk" HDR command (HDR_cmd_shrk) causes the loading of both SHR0 and SHR1. SHR1 is loaded with the first word of the payload (from the word_in input) while SHR0 is loaded with the FDC field coming from the FDC_in input since the mux M1, that selects the input of SHR0 between FDC_in and word_in, is controlled by the HDR_cmd signal and at this time HDR_cmd is high. Then, during the frame transmission, the Frame Builder issues a Read_TX_Buf command for each word that must be transferred from the TX Buffer to the Serializer: in the Serializer the clk40 cycle-long pulse on the Read_TX_Buf input is first narrowed by the Pulse Shrinker (see its description below) to create a clk_tx cycle-long pulse (the signal load_shr), and this latter is combined with the sel_shr signal to create two load command signals for SHR0 and SHR1 (the upper inputs of the two OR gates) that alternate at each successive frame word transferred from the TX Buffer to the Serializer.

The signal sel_shr is generated by the FSM Ser_Control, that uses the Bit Counter (BIT_CNT) to determine the end of the transmission of the FDC field and of each word of the frame. The Bit Counter reset is commanded by Ser_Control, and its enable is the inversion of the sel_THS signal (sel_FRM) to have it counting only the bits that are transmitted into the FRM channel of the output stream.

The D-type flip-flop STP_FF is used to generate the stop command for SER_CTRL: the stop signal is set to '1' when the stop_in input goes high, and is then reset when a HDR_cmd arrives. Indeed the stop_in command, generated by the Frame_Builder, must be extended to guarantee the proper operation of SER_CTRL in all cases (see SER_CTRL description).

The delay_HDR output signal (that is used by the Frame Builder to determine the moment at which a HDR command can be sent) is generated by means of the flip-flop DH_FF and the following OR gate: the arrival of a HDR_cmd sets the flip-flop, that is then reset when the Serializer is transmitting the third last bit of the last word of the frame (condition (stop = '1' OR stop = '1') AND countB = N_WORD-3): therefore, the DH_FF output (delay_HDR_0 signal) is set to '1' at the beginning of a frame and is reset almost at its end in a moment such that, for N_WORD = 16 and in all the cases of tx speed and frame length, delay_HDR_0 is low around the rising edge of clk40 at which a possible new HDR command must be sent by the Frame Builder to start a new frame transmission, back-to-back with the preceding. To obtain the actual delay_HDR output, the delay_HDR_0 signal is then OR-gated with the THS_busy signal coming from the THS_load_ctrl block, so that the Frame Builder is forced to delay a HDR generation also if a preceding HDR or SYN transmission is not completed. Indeed, as seen in the description of the THS_Scheduler, it is possible that a SYN is generated one or two clk40 cycles before the moment in which the Frame Builder would produce a HDR command, and thus in this case the HDR generation must be delayed; additionally, in the case tx speed = 640 Mbit/s and with a frame consisting in only one word, the frame transmission is completed in two clk40 cycles, i.e. one cycle before the completion of the HDR sequence transmission: the new HDR command generation must be delayed also in this situation, and this is done again by means of the delay_HDR signal.

A more detailed description of the internal blocks of the Serializer follows.

4.2.5.1 THS load controller

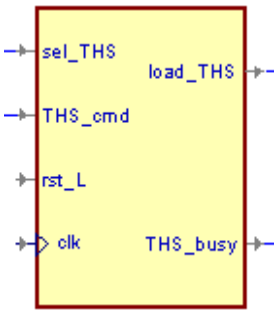


Fig. 4.29 – Symbol of THS_load_ctrl.

Name	Direction	# Bits	Active level	Description
sel_THS	IN	1	high	Signal coming from the sel_THS Generator: identifies the THS channel bits inside each clk40 period.
THS_cmd	IN	1	high	Signals that a TRG, HDR or SYN command is active.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input (connected to clk_tx).
load_THS	OUT	1	high	Load command for the THS shift register.
THS_busy	OUT	1	high	Signals that the transmission of a THS sequence is ongoing.

Table 4.19 – Description of input and output terminals of THS_load_ctrl.

The THS commands arriving to the Serializer from the THS Scheduler are one clk40 period long, so if the THS_cmd was used as the load command for the THS shift register this latter would be blocked for a clk40 cycle without shifting out properly the loaded sequence: the THS load controller generates a 1-clk_tx long load command for the correct operation of the THS shift

register when a THS command arrives. The internal architecture of this component is shown in Fig. 4.30:

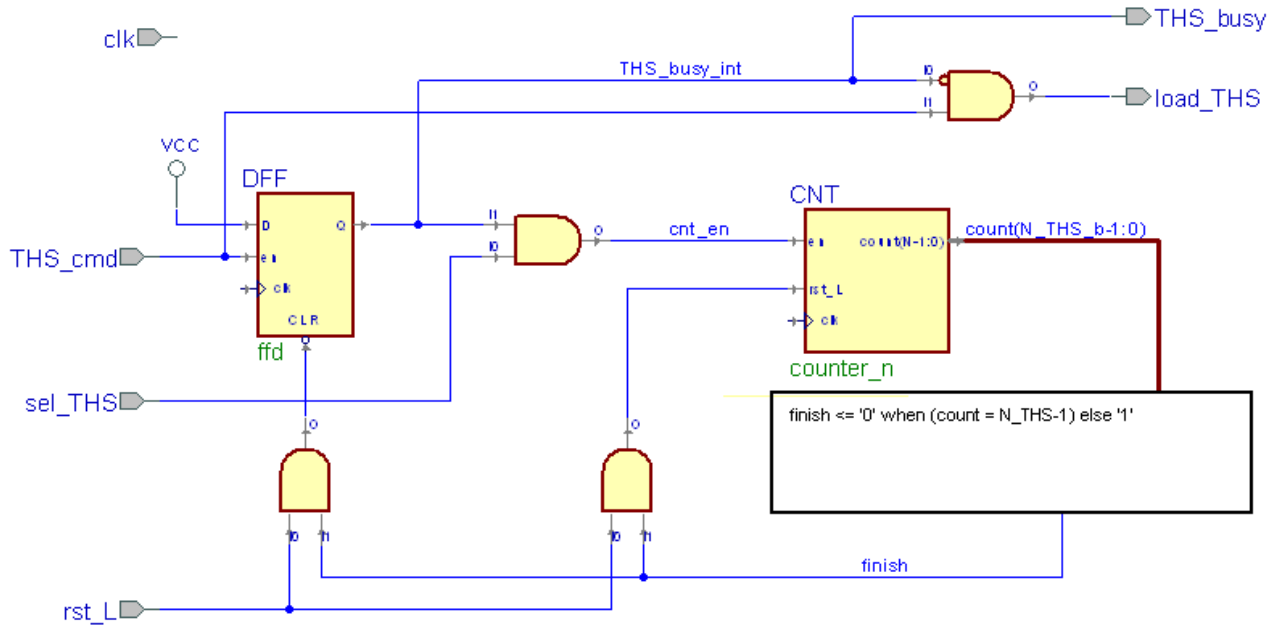


Fig. 4.30 – Internal block diagram of THS_load_ctrl.

The rst_L input clears the content of both the D flip-flop (DFF) and the counter (CNT). When rst_L is inactive and a '1' on the THS_cmd input is sampled on the rising edge of the clock, the DFF sets its output to '1' and thus the counting of CNT is enabled during the THS intervals of the transmission, i.e. when sel_THS is high. The load_THS output initially follows the THS_cmd to '1', but then it is reset to '0' when the DFF output is set: hence, the THS shift register will be loaded with the THS sequence only at the first clk_tx edge after the THS_cmd arrival, as it's correct. When the count reaches N_THS-1, where N_THS is the number of bits of the THS sequences, the DFF and the counter are reset and the circuit returns into its initial state, waiting for the next THS_cmd. The THS_busy output, being the DFF output, is high during the counting of the N_THS transmitted bits, and so it means that the transmission of a THS sequence is ongoing.

The behavior of the THS_load_ctrl on response of a THS_cmd is shown in Fig. 4.31 (for the case tx speed = 320 Mbit/s, as an example).

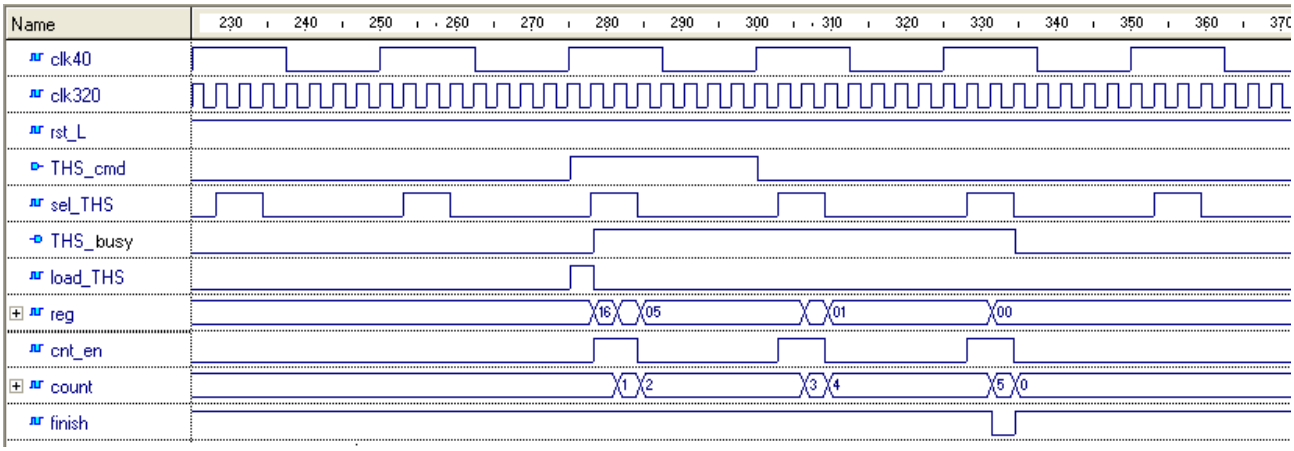


Fig. 4.31 – Simulation showing the behavior of THS_load_ctrl when a THS command arrives (tx speed = 320 Mbit/s). "Reg" is the content of the THS shift register.

4.2.5.2 THS command selector

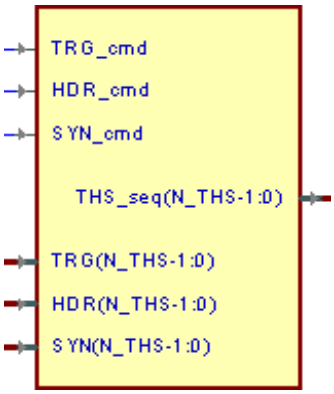


Fig. 4.32 – Symbol of THS_cmd_selector.

Name	Direction	# Bits	Active level	Description
TRG_cmd	IN	1	high	TRG command.
HDR_cmd	IN	1	high	HDR command.
SYN_cmd	IN	1	high	SYN command.
TRG	IN	6	-	TRG sequence from TRG_reg.
HDR	IN	6	-	HDR sequence from HDR_reg..
SYN	IN	6	-	SYN sequence from SYN_reg..
THS_seq	OUT	6	-	Selected THS sequence to be loaded in THS shift register.

Table 4.20 – Description of input and output terminals of THS_cmd_selector.

This component is a combinatorial logic that assigns to the THS_seq output the value of the TRG, HDR, or SYN input depending on which of the 3 input command (TRG_cmd, HDR_cmd or SYN_cmd) is active. If more than a input command is active the priority order is TRG, HDR, SYN.

4.2.5.3 Pulse Shrinker

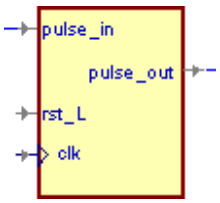


Fig. 4.33 – Symbol of Pulse_shrk.

Name	Direction	# Bits	Active level	Description
pulse_in	IN	1	-	One-clk40-cycle long input pulse.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input (connected to clk_tx).
pulse_out	OUT	1	-	One-clk_tx-cycle long output pulse.

Table 4.21 – Description of input and output terminals of Pulse_shrk.

This block, clocked by clk_tx, has the task of generating a one-clk_tx cycle long pulse (pulse_out) when a one-clk40 cycle long pulse arrives on its input pulse_in. It's internal architecture, similar to the one of THS load controller, is shown in Fig. 4.34:

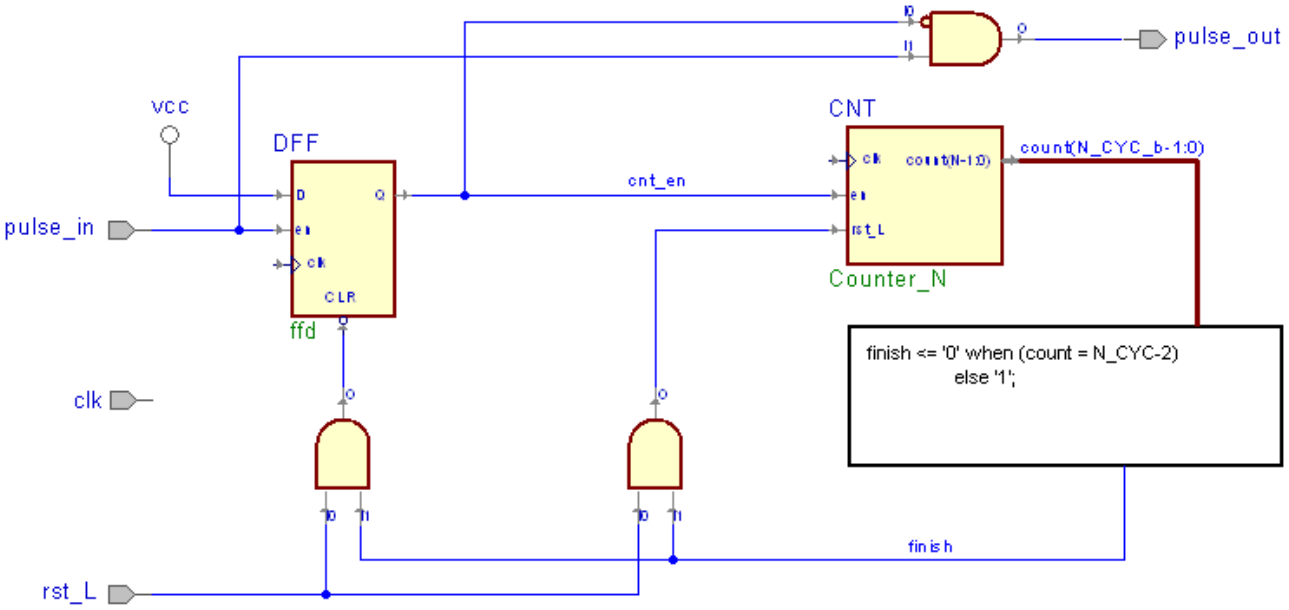
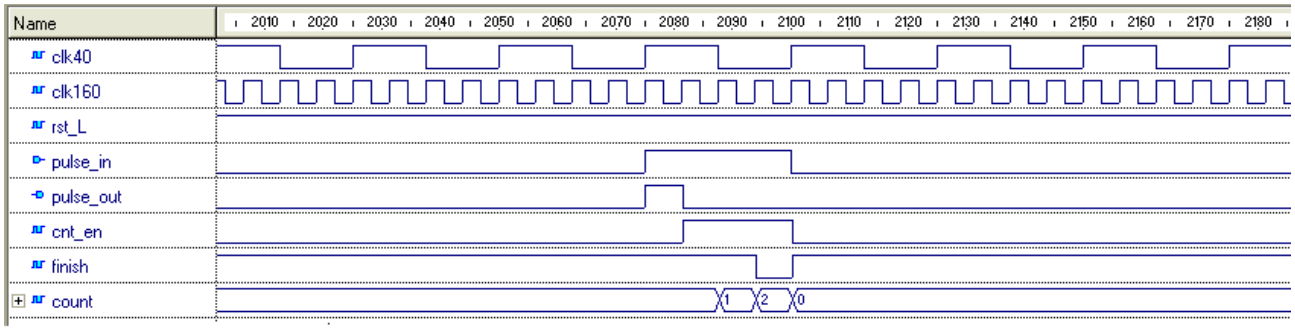


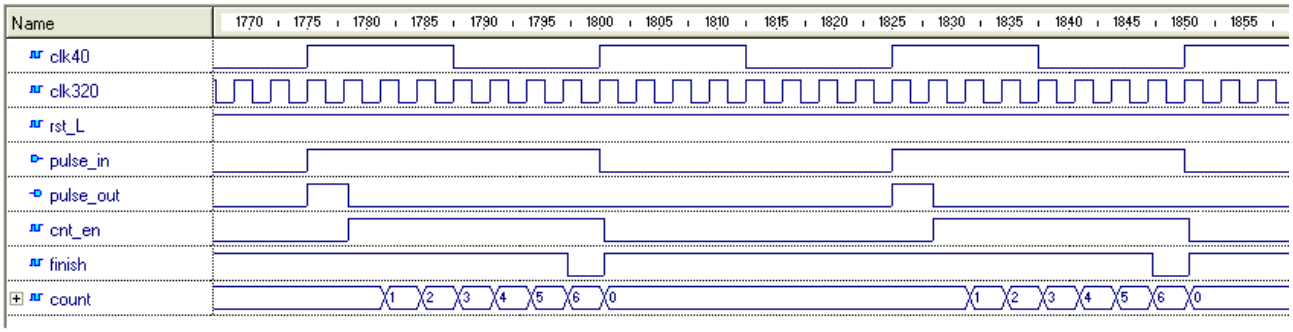
Fig. 4.34 – Internal block diagram of THS_load_ctrl.

The operation is similar to the one of the THS load controller, with the difference that here the sel_THS signal doesn't take part in the enabling of the counter: when rst_L is inactive and a high level of the pulse_in input is sampled on the rising edge of the clock, the DFF sets its output to '1' thus enabling the N_CYC_b-bit counter (whose modulus is N_CYC). The pulse_out output initially follows the pulse_in to '1', but after one clk_tx cycle (when the DFF output is set) it is reset to '0' and all the circuit becomes insensitive to the value of pulse_in for N_CYC-1 bits, i.e. for a clk40 cycle. Finally, when the count reaches N_CYC-2 the DFF and the counter are reset and the circuit returns into its initial state, waiting for the next pulse on the input (that can arrive at the next clk40 cycle, back to back with the preceding pulse).

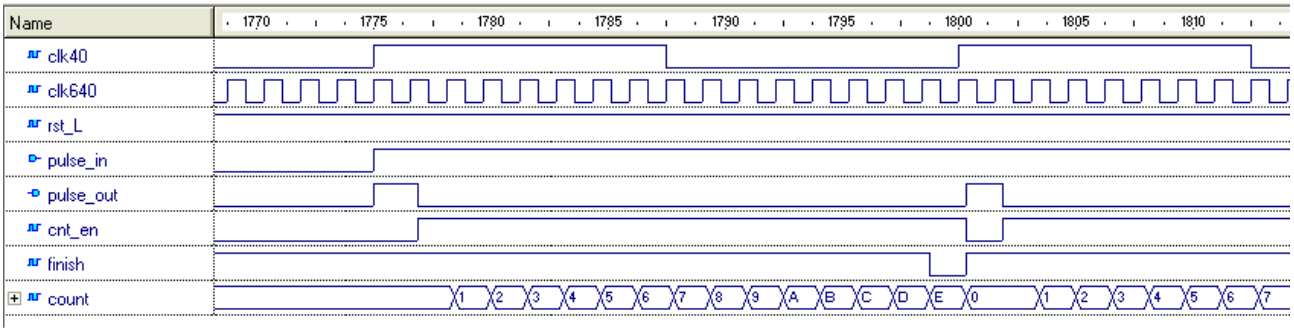
Fig. 4.35 shows an example of the operation of the Pulse Shrinker for tx speed = 160, 320 and 640 Mbit/s:



(a)



(b)



(c)

Fig. 4.35 – Simulation showing the behavior of Pulse_shrk for tx speed = 160 (a), 320 (b) and 640 Mbit/s (c)..

4.2.5.4 Ser_Control

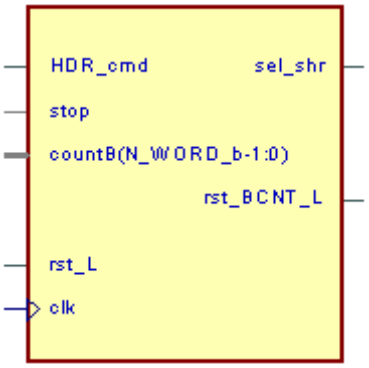


Fig. 4.36 – Symbol of Ser_Control.

Name	Direction	# Bits	Active level	Description
HDR_cmd	IN	1	high	Connected to HDR_cmd_shrk (shrunk version of HDR_cmd).
stop	IN	1	high	Stop signal generated by STP_FF.
count_B	IN	4	-	Output of Bit Counter.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input, connected to the transmission clock (clk_tx).
sel_shr	OUT	1	-	Selects the shift register (SHR0 or SHR1) to enable for serialization.
rst_BCNT_L	OUT	1	low	Reset command for the Bit Counter.

Table 4.22 – Description of input and output terminals of Ser_Control.

This entity is a Mealy machine that controls the operation of the shift registers SHR0 and SHR1, using the Bit Counter (BIT_CNT) to determine the end of the transmission of FDC and of each frame word. The state diagram of this FSM is shown in Fig. 4.37:

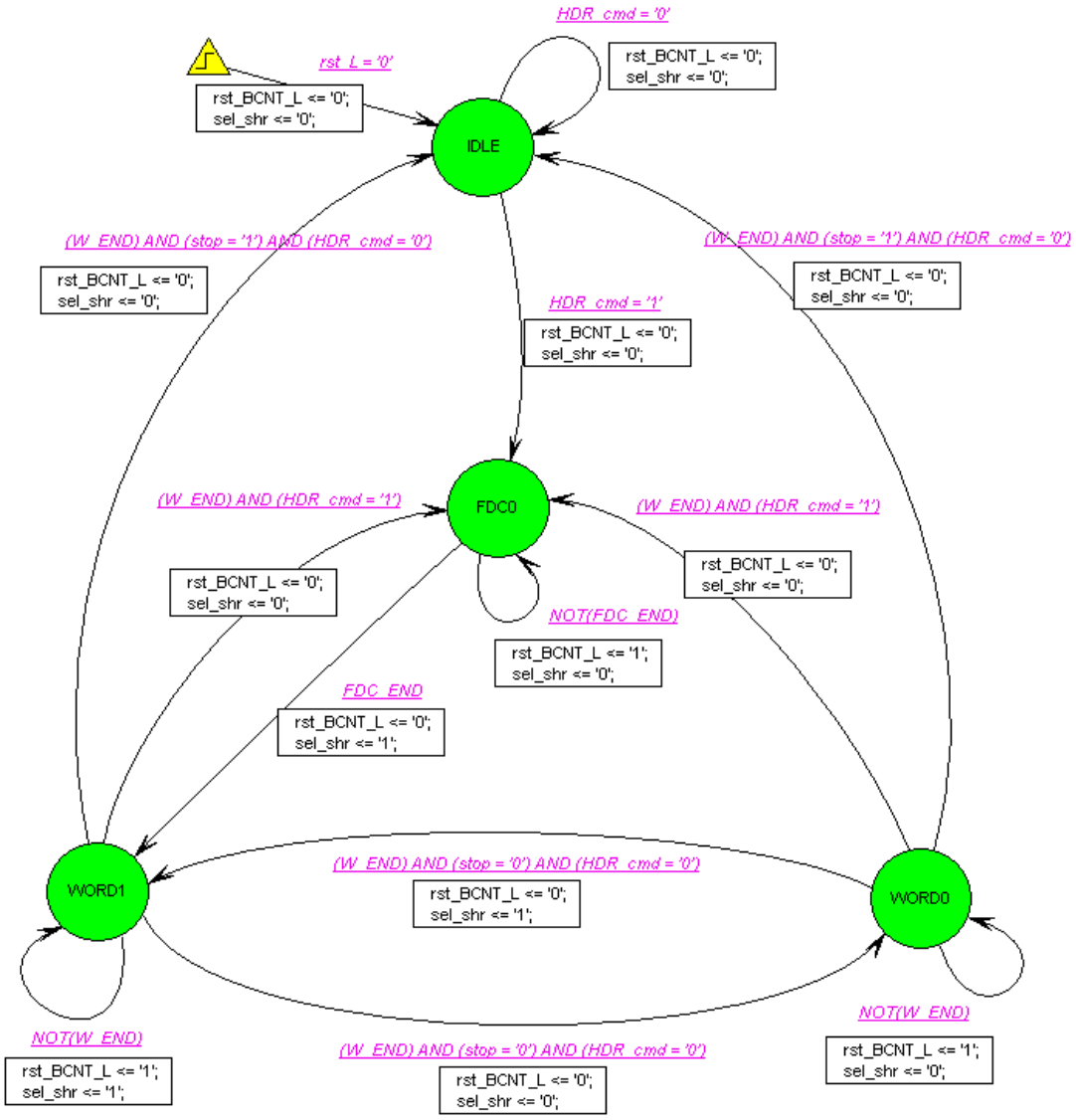


Fig. 4.37 – State diagram of Ser_Control.

At the reset (rst_L input = ‘0’) the machine goes into the IDLE state, resetting the bit counter (rst_BCNT_L <= ‘0’).

When a logic ‘1’ is detected on the HDR_cmd input, the FSM moves to the FDC0 state to transmit the FDC field by means of the shift register SHR0, which is selected as the source of the FRM stream (sel_shr <= ‘0’); as described above, at this time SHR0 is loaded with the value of the FDC_in input and SHR1 is loaded with the value of the word_in input, that is the first word of the frame (or its label). While the FSM is in the FDC0 state the transmission of the FDC sequence (that has been loaded in the SHR0 during the THS part of the first clk40 cycle of the frame) goes on, and the bit counter counts the bits transmitted in the FRM channel, being enabled by the negation of the sel_THS signal.

When the bit count reaches N_FDC-1 (condition FDC_END), meaning that the FDC sequence has been transmitted, the FSM goes to the state WORD1 to transmit the first word of the frame from SHR1: hence, on this transition SHR1 is selected as the source of the FRM stream (sel_shr <= ‘1’) and the bit counter is reset.

When the bit count reaches N_WORD-1 (condition W_END), meaning that the first word has been transmitted, if the frame is not finished (i.e. if stop = 0) Ser_Control goes to the state WORD0 to transmit the second word of the frame that has been loaded into SHR0 by the Frame Builder during the transmission of the first word: on this transition sel_shr is set to '0' again and the bit counter is reset (if the modulus of the counter is set equal to N_WORD, it resets itself at this point, but the situation can arise in which the bit counter reaches its limit value N_WORD-1 at the beginning of the THS interval: in this case, if Ser_Control didn't reset the counter, its value would be stuck at N_WORD-1 for all the duration of the THS interval, causing Ser_Control to bounce repeatedly between the states WORD0 and WORD1). When the second word has been completed, the FSM returns to the state WORD1 to transmit the third word, and so on until the end of the frame.

Once the last word of the frame is reached, the Frame Builder activates the stop_SER command that arrives to Ser_Control through its stop_in input and in turn activates the stop signal by means of the flip-flop STP_FF. If the last word transmission terminates inside the FRM interval of a clk40 cycle, the condition (W_END) AND (stop = '1') AND (HDR_cmd = '0') will become true and so Ser_Control goes back to the IDLE state. However, depending on the tx speed and the number of words of the frame, the situation can arise in which the frame transmission is completed exactly at the end of the FRM interval of a clk40 cycle: as shown in Fig. 4.38, this happens at tx speed = 160 Mbit/s with frames consisting in any number of words, at tx speed = 320 Mbit/s with frames consisting in k·3 words (with k positive integer), and at tx speed = 640 Mbit/s with frames consisting in 1 + k·7 words (with k positive integer).

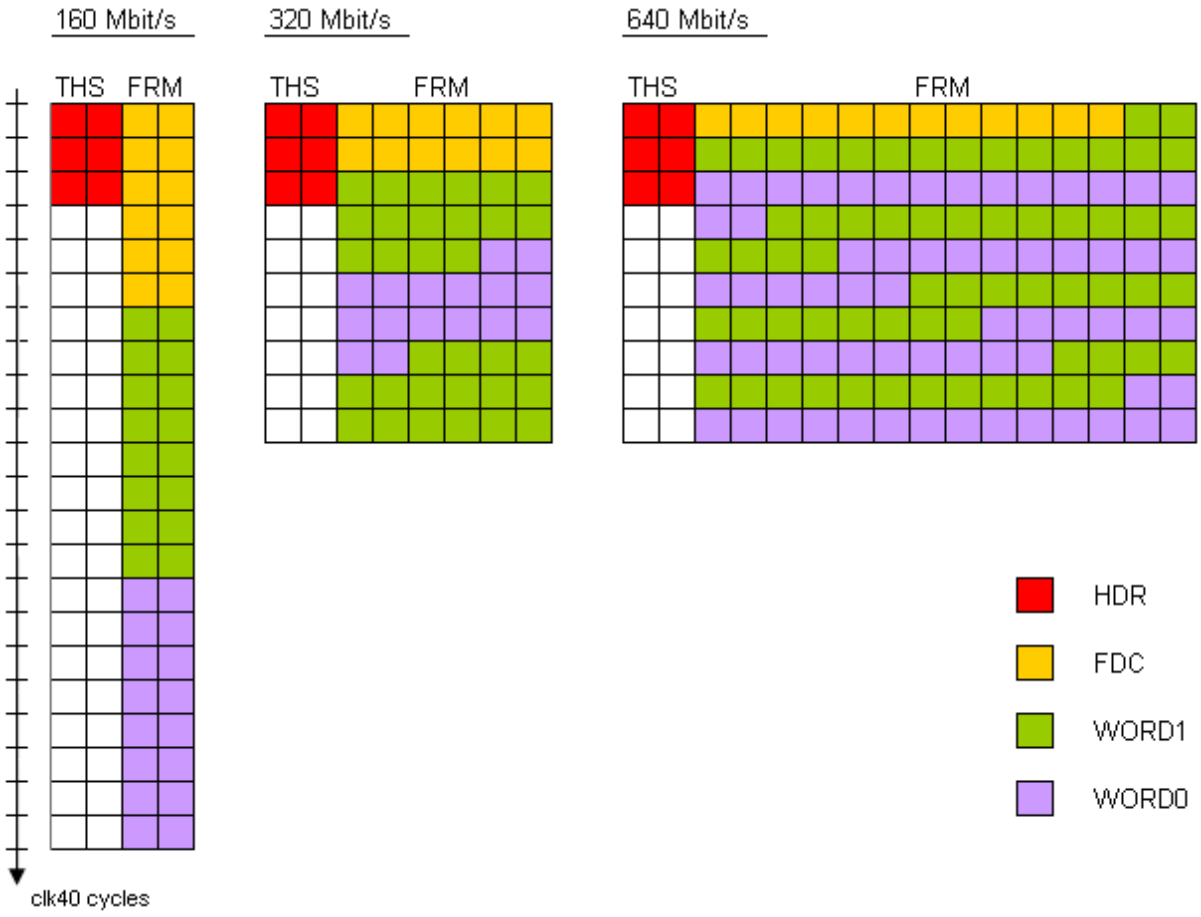


Fig. 4.38 – Possible situations of transmission being completed at the end of the FRM interval of a clk40 cycle.

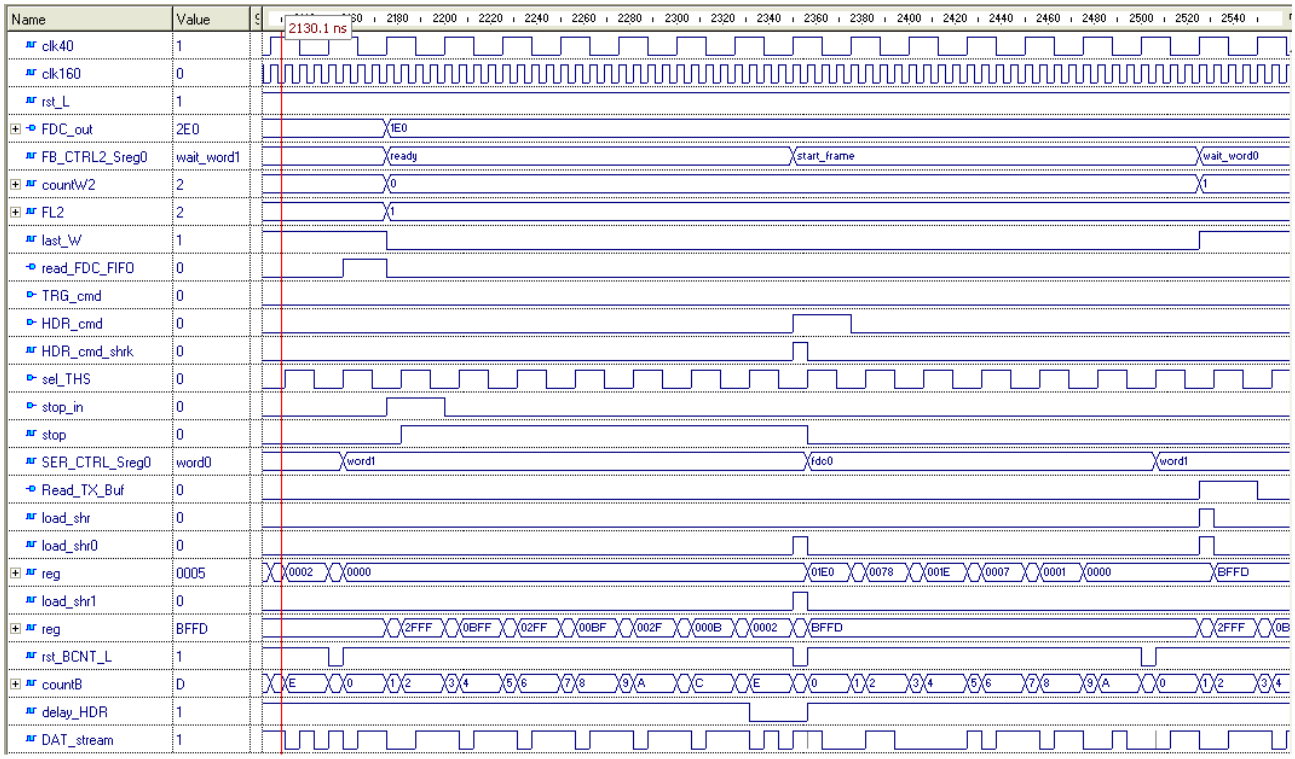
In this situation, the last bit of the last word of the frame is transmitted during the first clk_tx cycle of a new clk40 cycle. If there aren't more frames to be transmitted, the stop signal will continue to be high and Ser_Control goes back to the IDLE state. If on the contrary a new frame has to be transmitted (back-to-back frames), the Frame Builder will issue a HDR in this clk40 cycle, and two cases can occur:

1) the HDR is not delayed by the THS Scheduler: in this case Ser_Control will find the condition (W_END) AND (HDR_cmd = '1') to be true and will move directly to the state FDC0 to begin the transmission of the new frame;
2) the HDR is delayed by the THS Scheduler (because of an upcoming TRG): in this case Ser_Control will find the condition (W_END) AND (stop = '1') AND (HDR_cmd = '0') to be true and will go back to the IDLE state, to wait for the transmission of the frame to be started by the Serializer when the HDR will be scheduled.

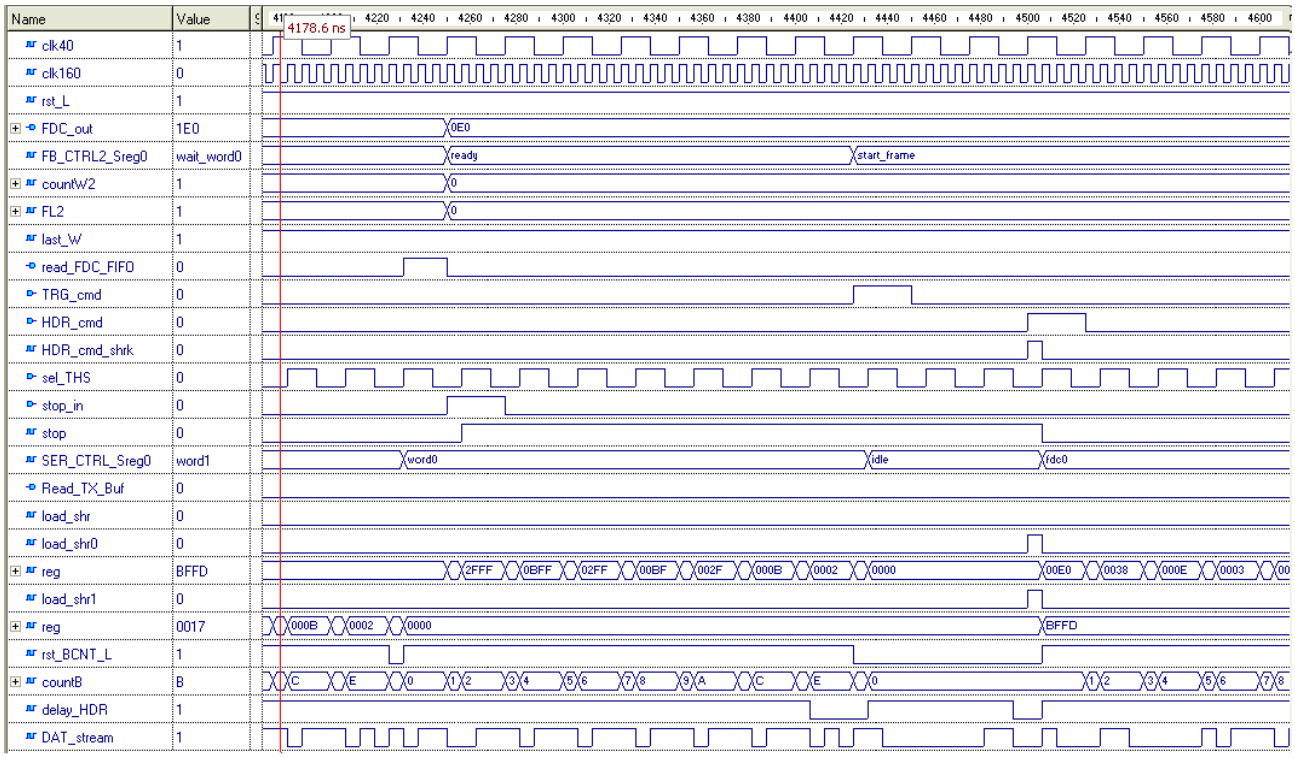
In the second case the stop command needs to extend to the first clk_tx cycle of the new clk40 cycle, or Ser_Control would not go back to IDLE: this explain the operation of the STP_FF flip-flop in the Serializer.

Of the Ser_Control outputs, sel_shr is registered while rst_BCNT_L is not: this assures that at the moment of the state change of Ser_Control (from FDC0 to WORD1 and then between WORD1 and WORD0) the signal sel_shr changes state too, and the Bit Counter starts again from count 0.

Fig. 4.39, Fig. 4.40 and Fig. 4.41 show an example, for tx speed = 160, 320 and 640 Mbit/s, of the evolution of the Ser_control signals at the end of a frame transmission, with a new frame transmission following immediately after or delayed by a TRG. Some FB_Control2 signals are also reported, to show the joint behavior of the Frame Builder and the Serializer during the frame transmission.

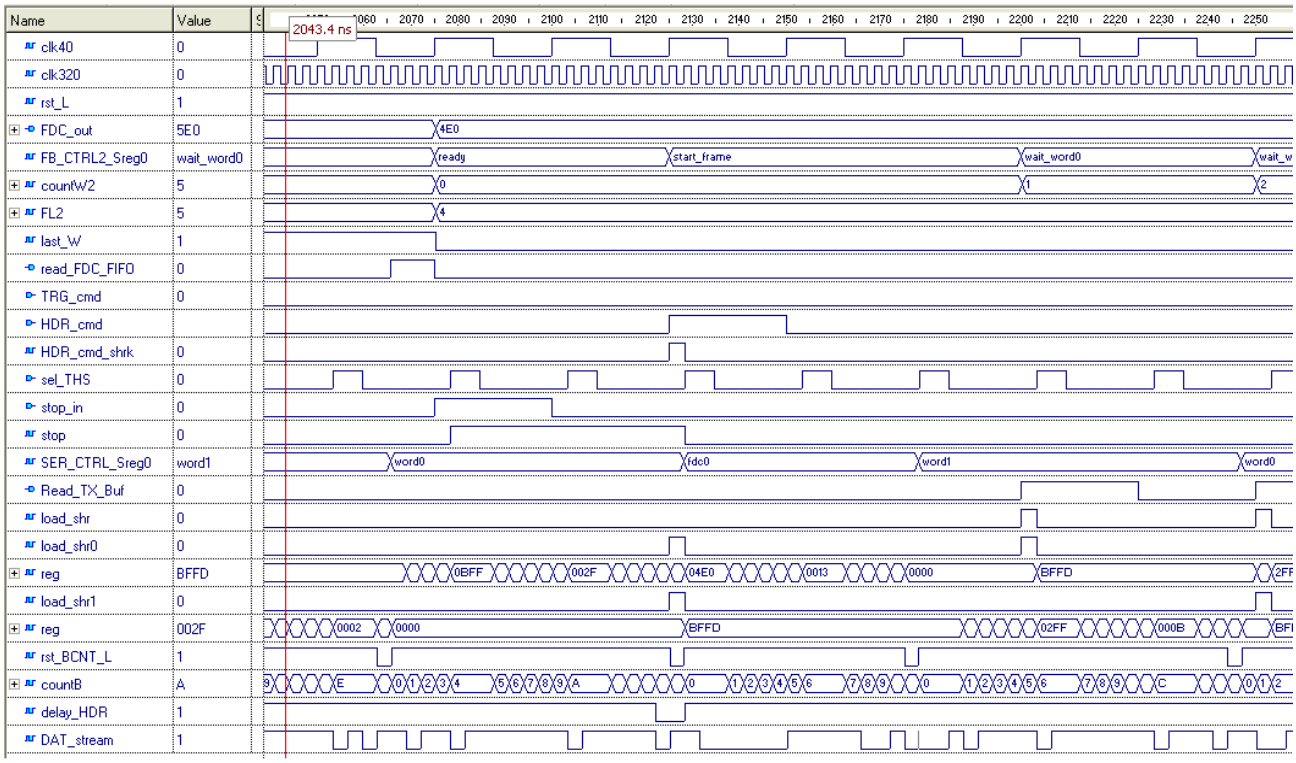


(a)

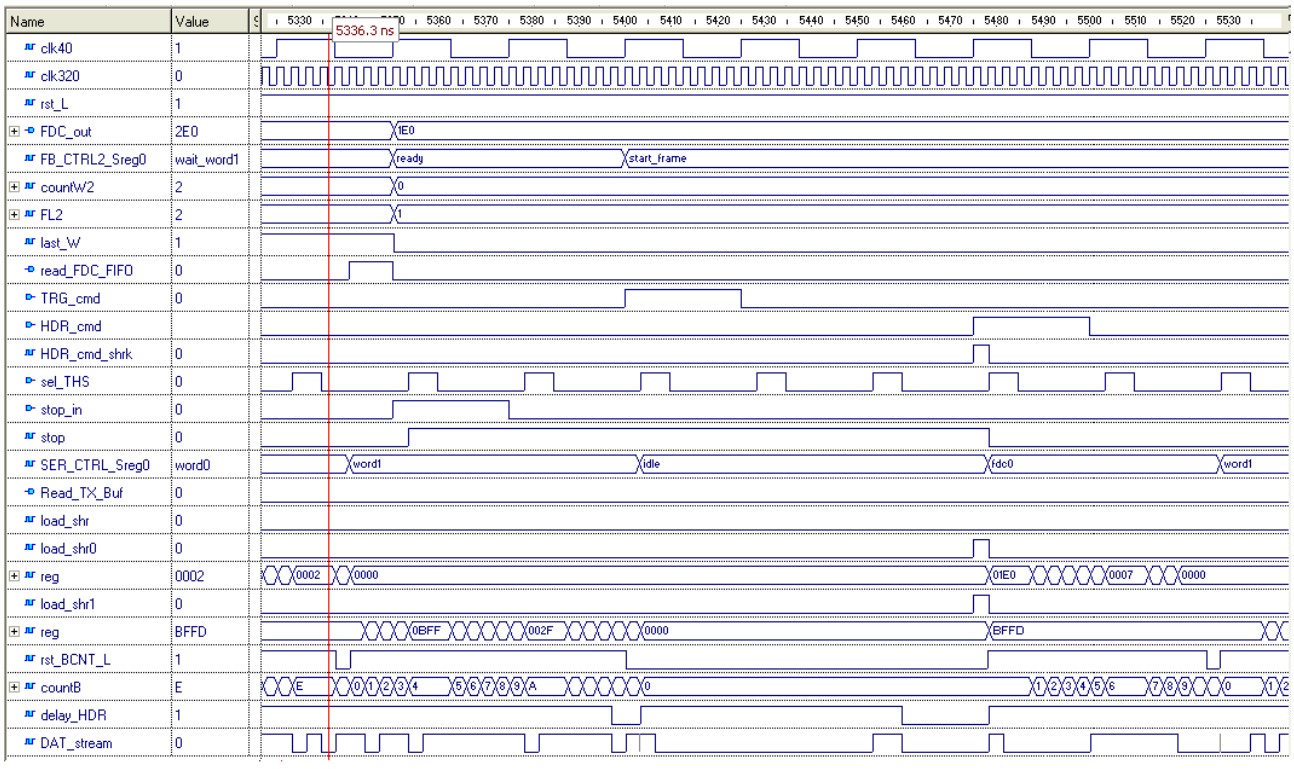


(b)

Fig. 4.39 – Simulation showing the evolution of the Ser_control signals at the end of a frame transmission, with a new frame transmission following immediately after (a) and delayed by a TRG (b); tx speed = 160 Mbit/s. The two signals "reg" are the content of SHR0 and SHR1.

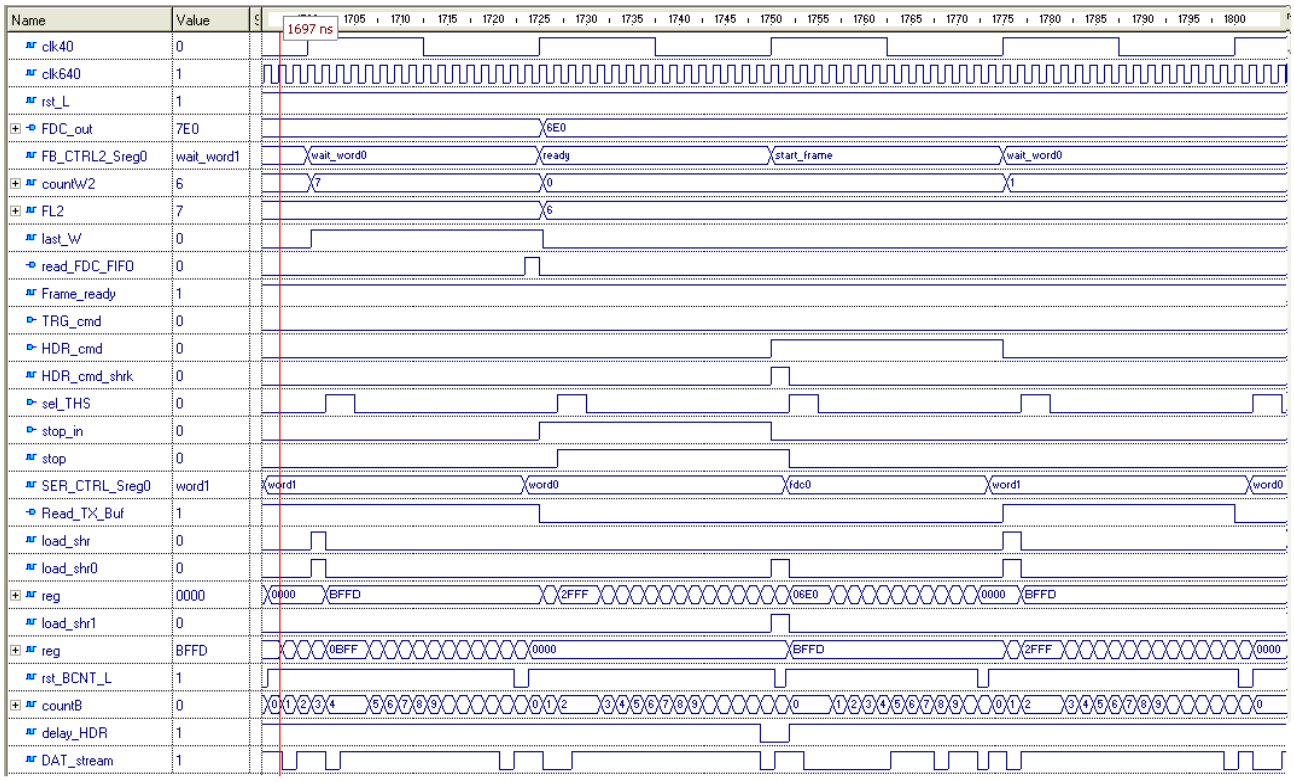


(a)

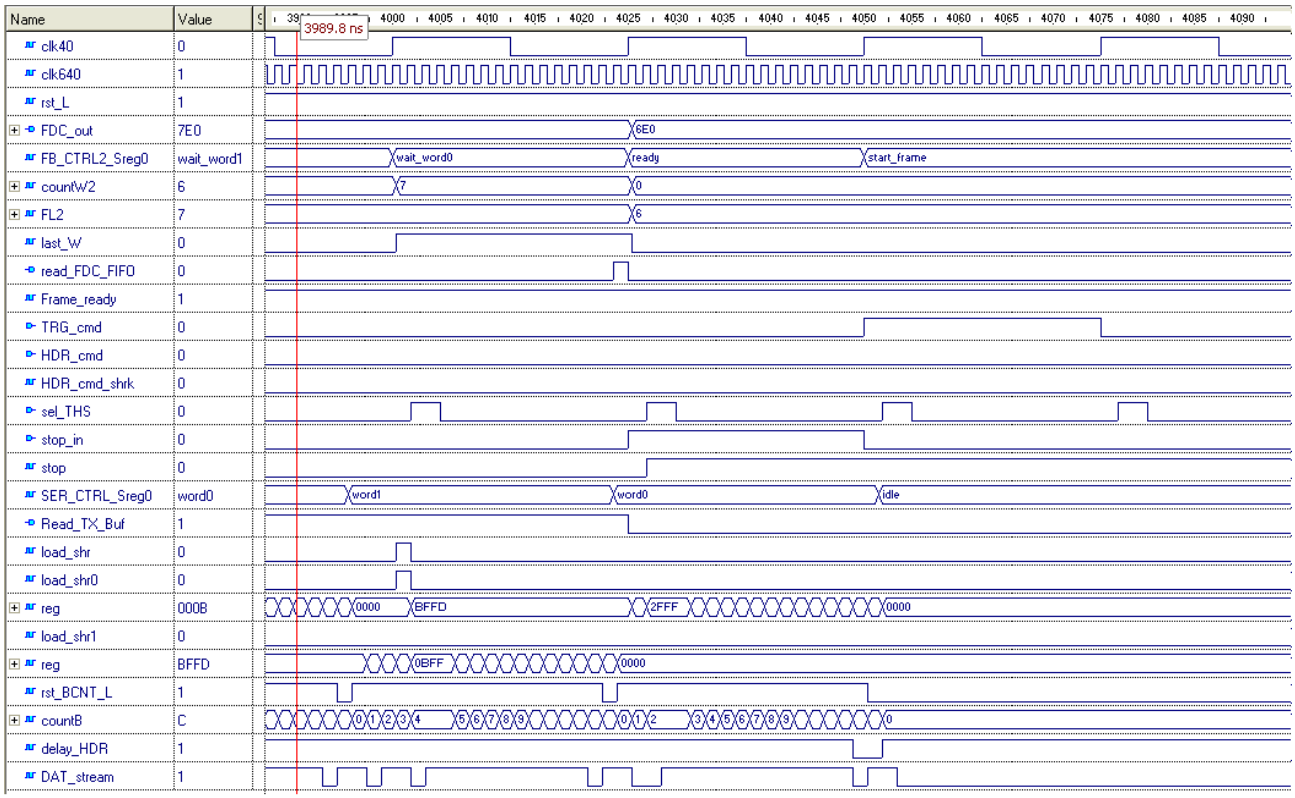


(b)

Fig. 4.40 – Simulation showing the evolution of the Ser_control signals at the end of a frame transmission, with a new frame transmission following immediately after (a) and delayed by a TRG (b); tx speed = 320 Mbit/s. The two signals "reg" are the content of SHR0 and SHR1.



(a)



(b)

Fig. 4.41 – Simulation showing the evolution of the Ser_control signals at the end of a frame transmission, with a new frame transmission following immediately after (a) and delayed by a TRG (b); tx speed = 640 Mbit/s. The two signals "reg" are the content of SHR0 and SHR1.

4.3 FF_RX

The receiver interface FF_RX has the logic internal architecture described in section 3.2.2; the Active-HDL block diagram for this entity, showing also the internal signals that connect the various blocks, is reported in Fig. 4.42.

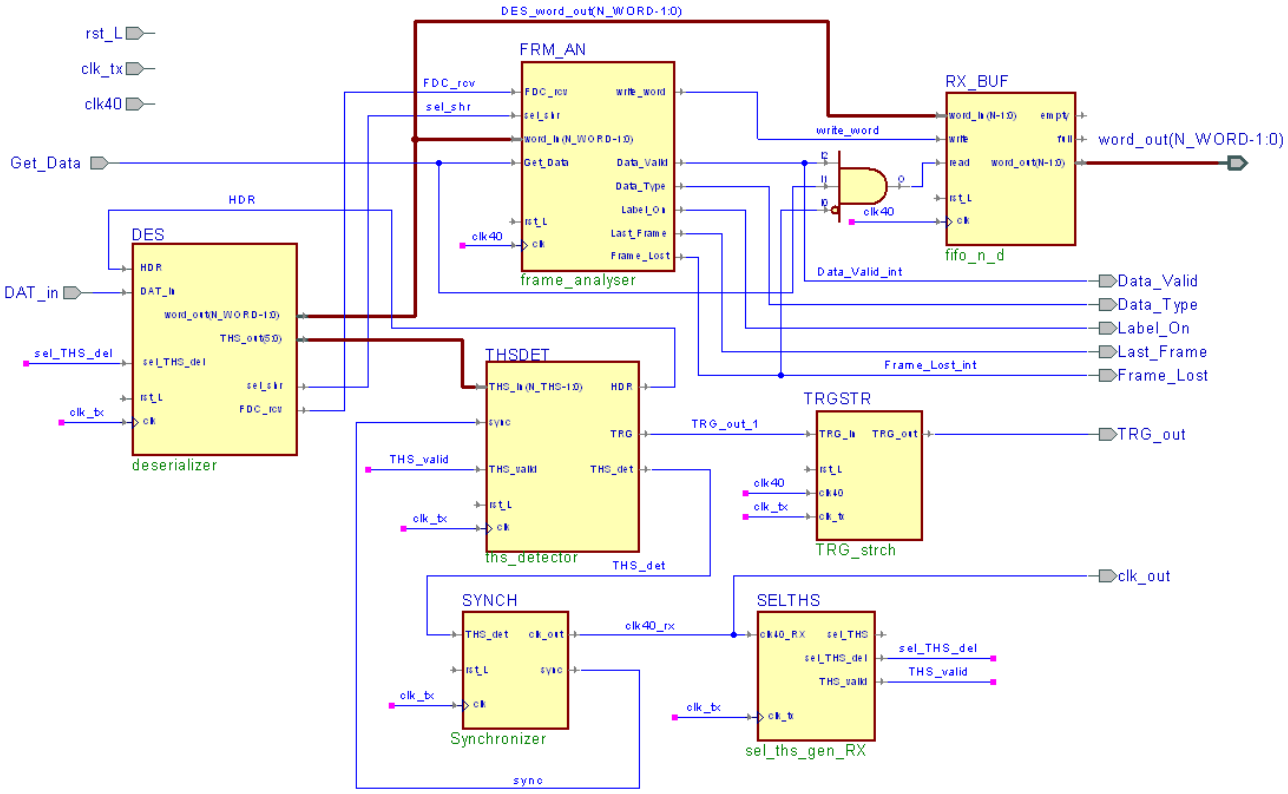


Fig. 4.42 – Block diagram of FF_RX.

Besides the five blocks Deserializer, THS Detector, Synchronizer, Frame Analyser and RX Buffer that were described in the functional architecture, two auxiliary blocks are present: sel_THS Generator RX (entity sel_THS_gen_RX) that has the task of generating the sel_THS signal needed by Deserializer and the THS_valid signal needed by THS Detector (see section 4.3.2), and TRG Stretcher, that generates a one-clk40-cycle long pulse (TRG_out output) for TRG pattern revealed by THS Detector. The internal components of FF_RX are described in more detail hereafter.

4.3.1 Deserializer

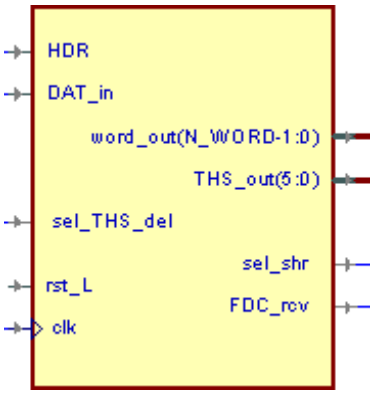


Fig. 4.43 – Symbol of Serializer.

Name	Direction	# Bits	Active level	Description
DAT_in	IN	1	-	DAT serial stream input.
HDR	IN	1	high	HDR signal from THS Detector.
sel_THS_del	IN	1	high	Delayed sel_THS signal (from sel_THS_gen_RX).
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input, connected to the transmission clock (clk_tx).
word_out	OUT	16	-	Parallel data output for Frame Analyser and RX Buffer.
THS_out	OUT	6	-	THS bits for THS Detector.
sel_shr	OUT	1	-	Signal directed to Frame Analyser to indicate which shift register is in use inside the Deserializer.
FDC_rcv	OUT	1	high	Signal directed to Frame Analyser to indicate that the FDC field of a frame is being received by Deserializer.

Table 4.23 – Description of input and output terminals of Deserializer.

The Deserializer receives the DAT serial bit stream (DAT_in) and converts it in parallel form, providing the THS bits (THS_out) for the THS Detector and the frame words and FDC field (word_out) for the RX Buffer and the Frame Analyzer. The block diagram is pictured in Fig. 4.45:

The block of SISO and SISO/PO registers R1, …, R5 constitutes the shift register (let's call it shift register A) that has the task of providing the THS bits in parallel form for the THS detector, and is driven by the clk_tx shifting in a new DAT stream bit at each clock edge. In the hypothesis of THS sequences consisting in 6 bits (in 3 successive clk40 cycles, 2 bits per cycle), the registers R1, R3 and R5 have a dimension of 2 bits each one and their parallel output, as a whole, makes up the THS output; R2 and R4, instead, have only the function of appropriately separating the THS bits of the incoming DAT stream and have a dimension that depends on the tx speed, being equal to the N_FRM parameter (passed as generic to the Deserializer entity): N_FRM = 2, 6, 14 for tx speed = 160, 320, 640 Mbit/s respectively. The complete shift register A, thus consisting in 10, 18 or 34 bits, is shown in Fig. 4.44 for the three possible tx speeds:

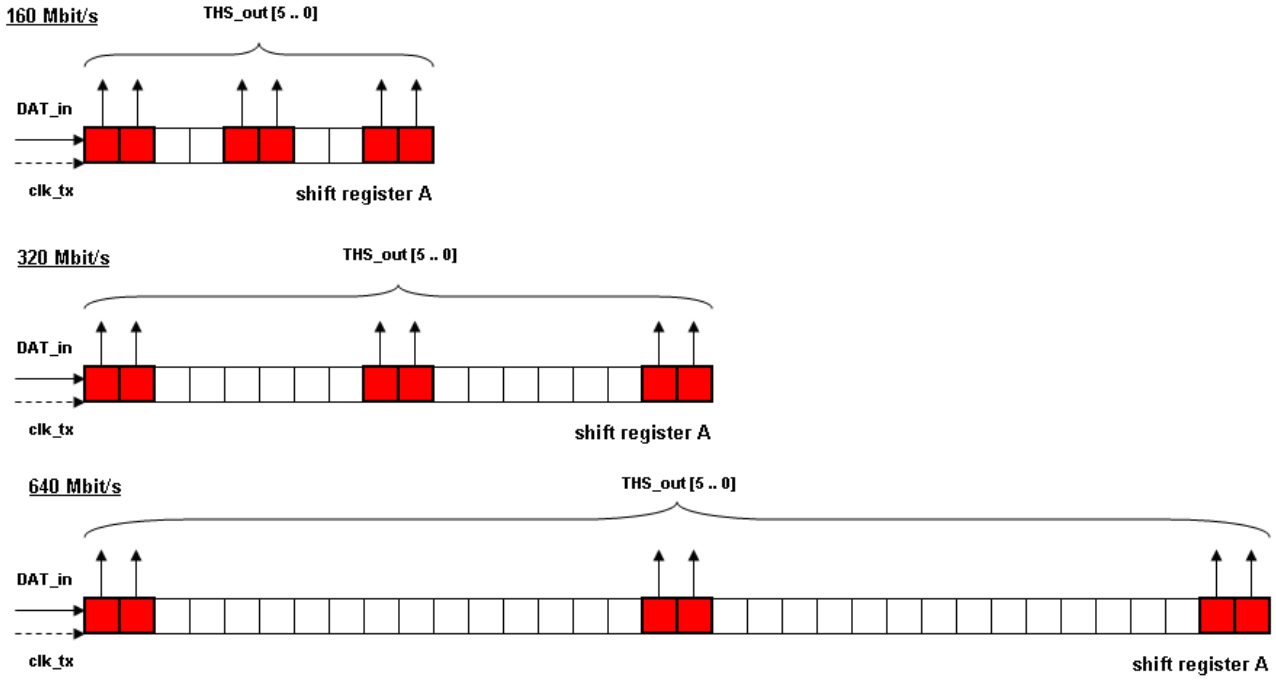


Fig. 4.44 – Shift register A of Deserializer in the three different architectures (for tx speed = 160, 320 and 640 Mbit/s).

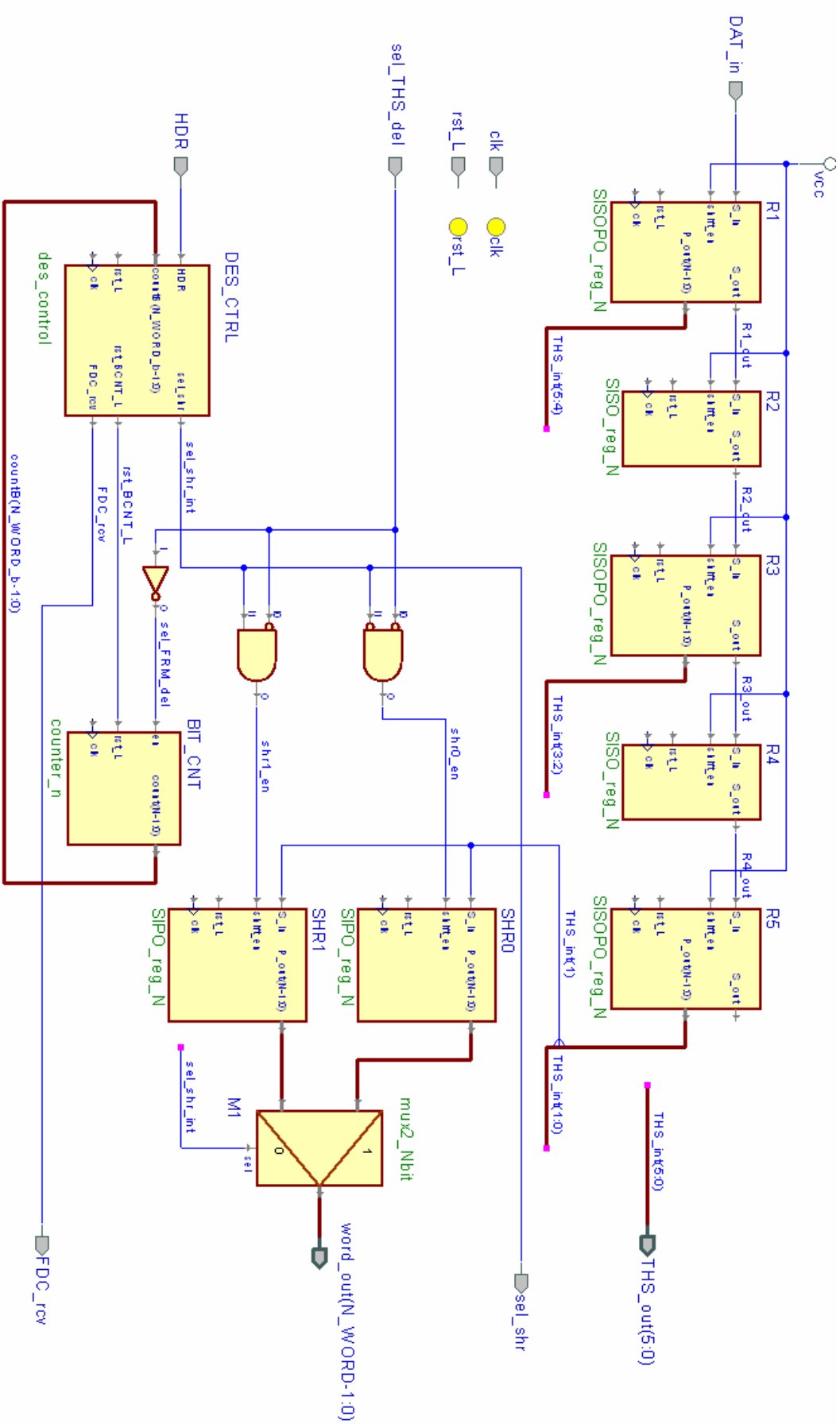


Fig. 4.45 – Internal block diagram of Deserializer.

When a HDR sequence is present in the THS bits, the THS detector recognizes it and notifies the event to the Deserializer through its HDR input: at this point, the Deserializer must start collecting the FDC field and the words of the incoming frame and making them available in parallel form for the Frame Analyzer and the RX Buffer. Since these latter blocks work with the 40 MHz clock, each deserialized word must be available for at least a clk40 cycle: this dictates the need of at least one buffering register in the Deserializer to hold the last received word stable for some time while the next word is being shifted in from the DAT stream at one bit per clk_tx cycle. To minimize the total number of flip flops in the registers, one could think to add as buffer a single register, of dimension = N_WORD, in addition to the shift register A (extended to have at least N_WORD bits, excluding the THS locations), using the latter to deserialize the words and transferring each word in a parallel way from the shift register A to the buffer register. However, in the cases tx speed = 320 and 640 Mbit/s this approach would imply the need of quite complex combinatorial logic to move the data from the shift register A to the buffer register, since the words appear in the shift register A in different positions: for example, in the case tx speed = 640 Mbit/s, if at a certain clk40 cycle a word is present in the FRM bits 0 to 15 of the shift register A (the FRM bits of the shift register A are the ones represented in white in Fig. 4.44), then at the next clk40 cycle the new word occupies the FRM bits 2 to 17 and so on, as shown in Fig. 4.46:

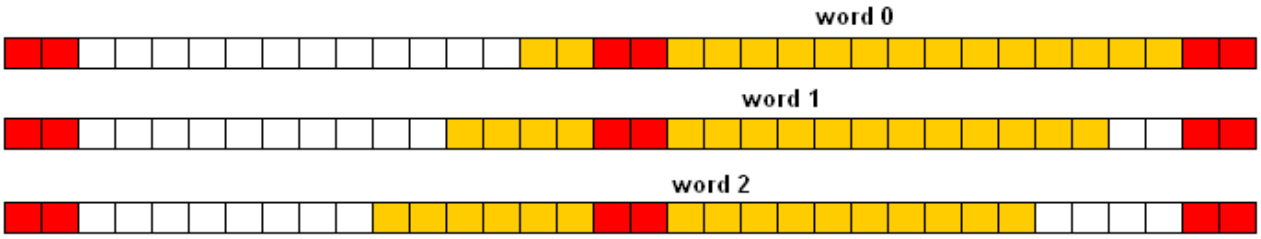


Fig. 4.46 – Positions occupied by frame words in shift register A in successive clk40 cycles, in the case tx speed = 640 Mbit/s.

Therefore, to avoid the use of complex combinatorial logic and the probable associated timing problems, an architecture was chosen with two buffer shift registers, SHR0 and SHR1, that are alternatively loaded with the serial output of the shift register A: when SHR0 is loaded, SHR1 maintains its data stable to be read by the Frame Analyzer or by the RX Buffer, and vice versa. The alternate writing of SHR0 and SHR1 is controlled by the state machine DES_Control by means of the signal sel_shr; to generate the effective enable signals for the shift registers, en_shr0 and en_shr1, sel_shr is then AND gated with the signal sel_THS_del (delayed sel_THS) so that SHR0 and SHR1 actually shift in only the FRM bits of the input DAT stream. The Des_control FSM is started by the arrival of a pulse on the HDR input from the THS_Detector, indicating the beginning of a new frame in the DAT stream, and uses the Bit Counter BIT_CNT to determine the moment when a word has been written in SHR0 [SHR1] to switch sel_shr and start writng a new word in SHR1 [SHR0]: the Bit Counter is enables by the same signal sel_THS_del to count only the bits that are actually routed into SHR0 or SHR1.

In the Serializer block of the TX Interface, sel_THS is a signal that stays high during the THS part of each clk40 cycle (2 bits) and low in the FRM part: in the Deserializer, to have SHR0 and SHR1 load the FRM bits of the DAT stream, their input could be connected to the leftmost FRM bit of the shift register A using the sel_FRM signal (the inverse of the sel_THS signal) as shift enable for SHR0 and SHR1. However, since the HDR pulse produced by the THS_detector occurs in the first clk_tx cycle of the low period of sel_THS, i.e. the first clk_tx cycle of the high period of sel_FRM, there would be no time for Des_Control to start the Bit Counter and begin the frame acquisition. So, a one-clk_tx-cycle delayed version of sel_THS and sel_FRM, sel_THS_del and sel_FRM_del, is used instead to enable SHR0/1 and the Bit Counter, and the same delay is introduced also into the

data path from DAT_in to SHR0/1 by picking up the input of SHR0/1 from the THS bit 1 of the shift register A. The resulting conceptual architecture is shown in Fig. 4.47, for the case of tx speed = 320 Mbit/s as an example:

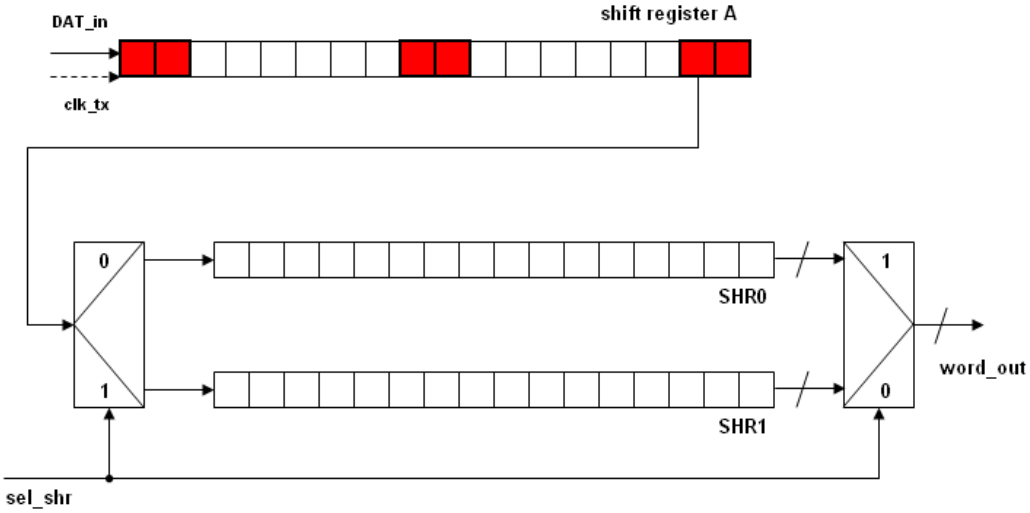


Fig. 4.47 – Conceptual architecture of Deserializer, in the case of tx speed = 320 Mbit/s.

The internal blocks of the Deserializer are described in more detail hereafter.

4.3.1.1 Des_control

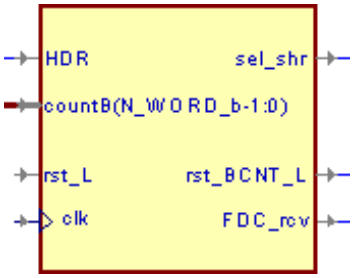


Fig. 4.48 – Symbol of Des_Control.

Name	Direction	# Bits	Active level	Description
HDR	IN	1	high	HDR signal from THS Detector.
count_B	IN	4	-	Output of Bit Counter.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input (connected to clk_tx).
sel_shr	OUT	1	-	Selects which shift register (SHR0 or SHR1) is enabled for deserialization.
rst_BCNT_L	OUT	1	low	Reset command for the Bit Counter.
FDC_rcv	OUT	1	high	Signal directed to Frame Analyser to indicate that the FDC field of a frame is being received by Deserializer.

Table 4.24 – Description of input and output terminals of THS_load_ctrl.

This block is a Mealy FSM that has the task of controlling, by means of the sel_shr signal, the alternate loading of SHR0 and SHR1 with the words of the incoming frame. Through the active low reset signal rst_BCNT_L this FSM also controls the Bit Counter, that is used to count the bits that enter SHR0 and SHR1. Its state diagram is reported in Fig. 4.49:

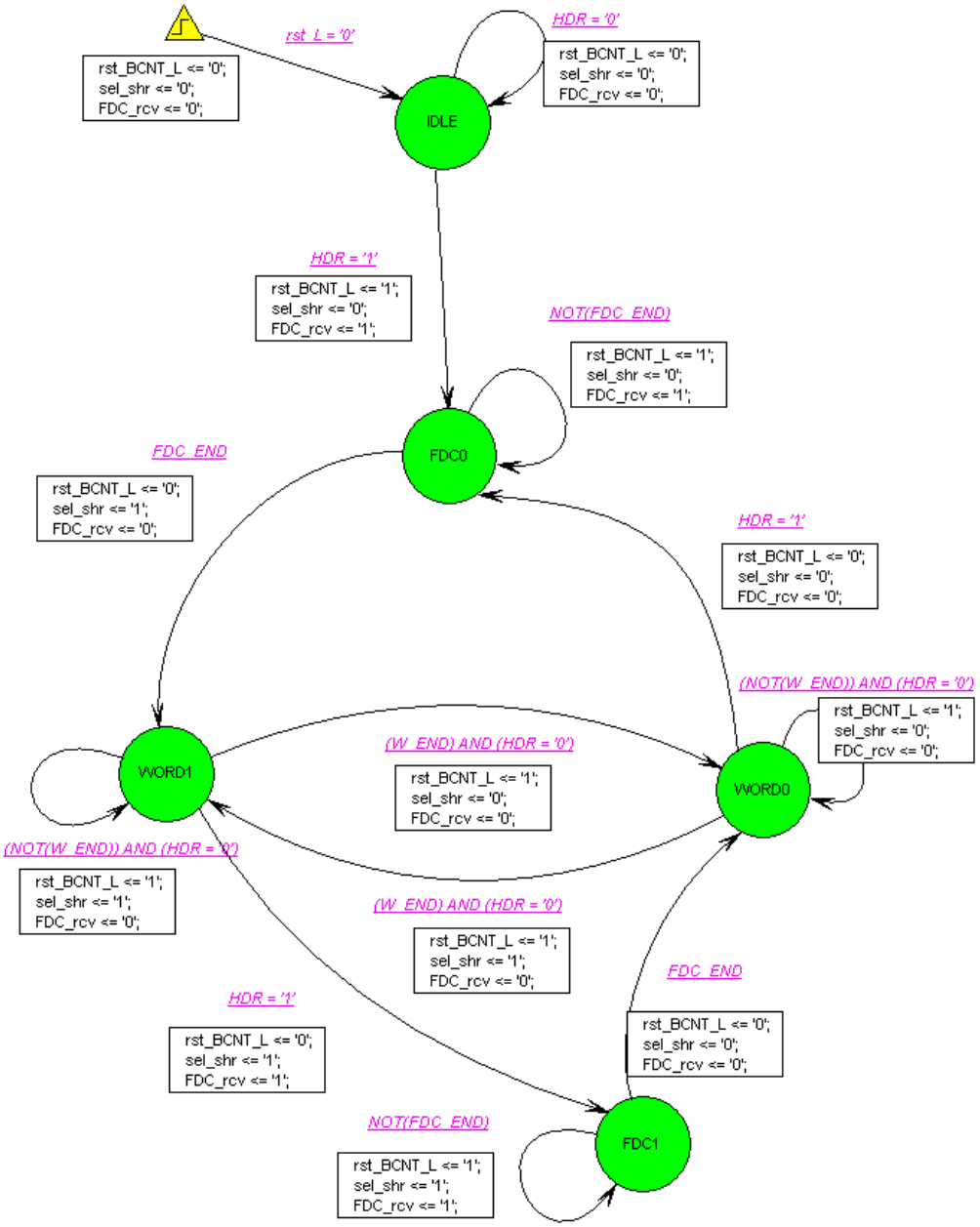
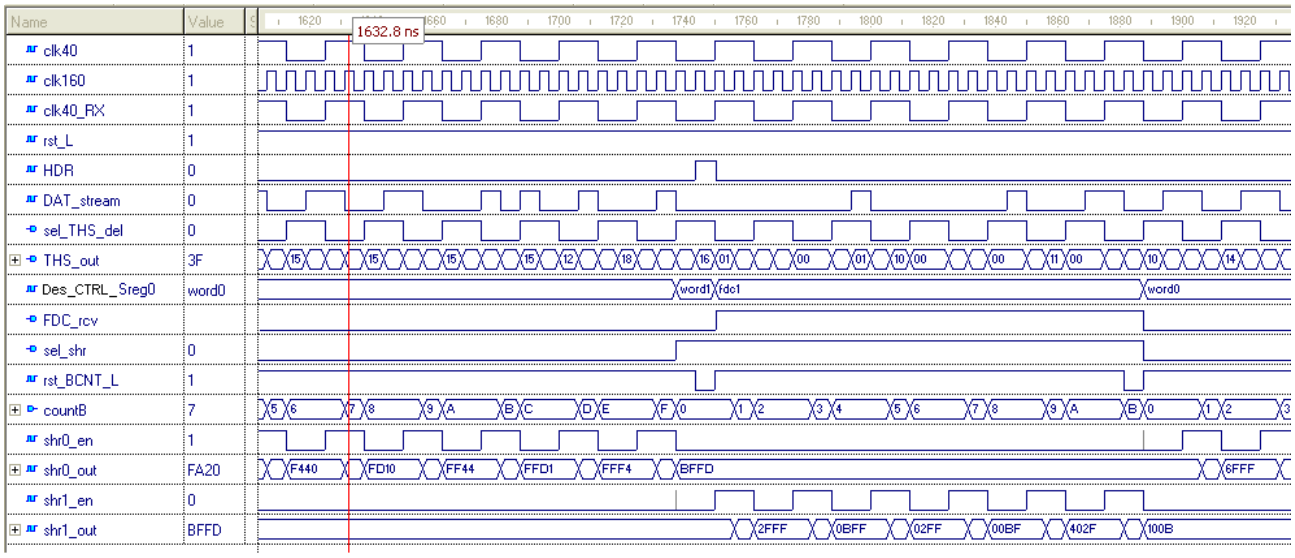
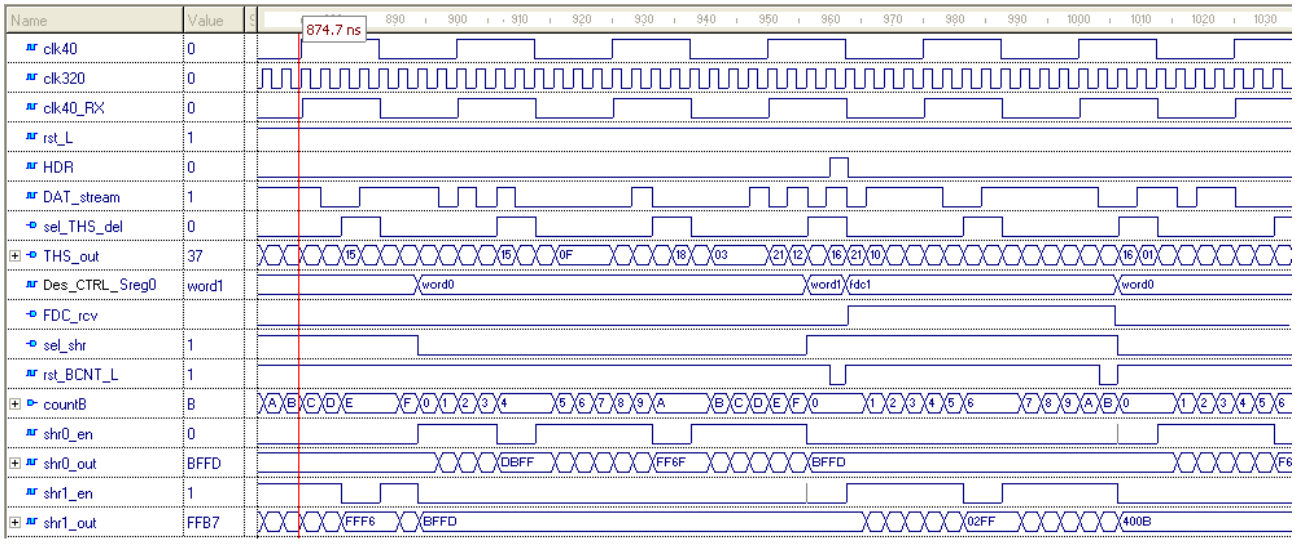


Fig. 4.49 – State diagram of Des_Control.

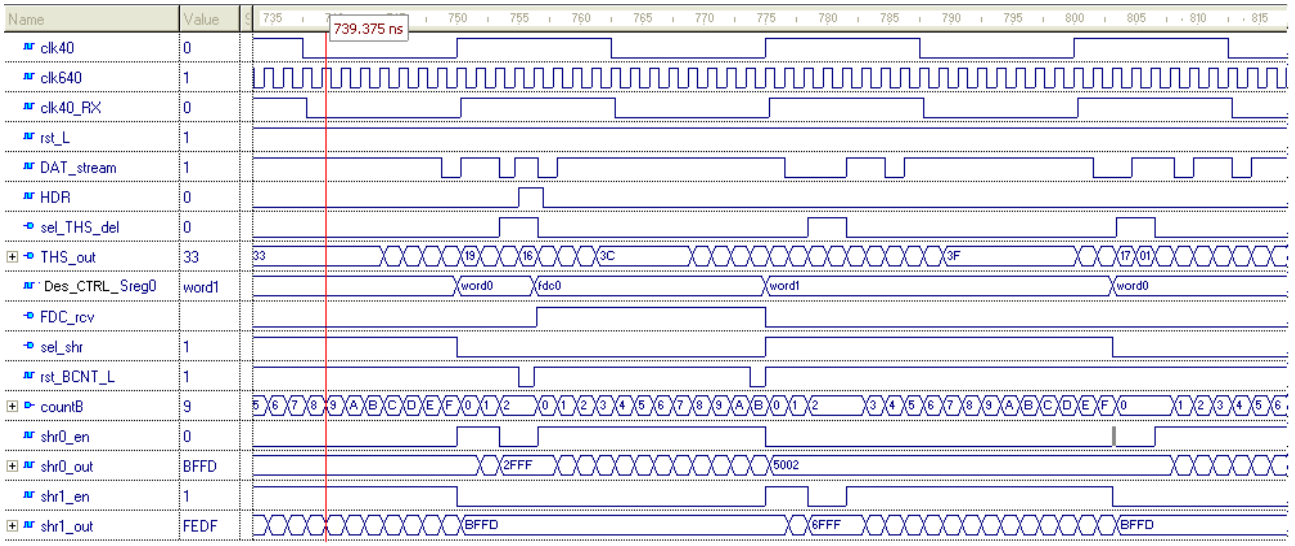
The FSM begins at the reset in the state IDLE, resetting the Bit Counter (rst_BCNT_L <= ‘0’). When a HDR arrives (HDR = ‘1’), the FSM goes to the FDC0 state starting the Bit Counter (rst_BCNT_L <= ‘1’) that counts the bits of the FDC field of the frame that are shifted in SHR0 (sel_shr <= ‘0’). When all the FDC field has been written in SHR0 (condition FDC_END), the FSM moves to the state WORD1 resetting the Bit Counter and enabling SHR1 to shift in the bits of the first word of the frame; during the loading of the word in SHR1, the FDC field in SHR0 is stable and can be read by the Frame Analyzer. As soon as the first word is completed (condition W_END) the Des_Control goes to the WORD0 state switching sel_shr to 0 again to write the second word in SHR0 (the Bit Counter resets itself automatically when the count reaches N_WORD, so Des_control doesn’t need to pull down rst_BCNT_L). When the second word is finished, the FSM goes back to WORD1 to deserialize the third word and so on. This loop between the states WORD0 and WORD1 is interrupted when a new HDR arrives, marking the beginning of a new frame: when this happens, the FSM goes to the state FDC0 or FDC1 whether the state occupied at the moment of the HDR arrival was WORD0 or WORD1: doing so, the last word of the preceding frame can be



(a)



(b)



(c)

Fig. 4.50 – Simulation showing the evolution of the Des_Control signals at the passage between two consecutive frames; tx speed = 160 (a), 320 (b) and 640 Mbit/s (c).

kept stable for at least one clk40 cycle because the shift register where it was written is not the one where the new FDC is written now. From the FDC0 or FDC1 state the FSM then moves to WORD1 or WORD0, respectively, and the cycle for the acquisition of the new frame is started.

The output FDC_rcv (FDC receiving) is pulled up when Des_Control is in the state FDC0 or FDC1: it signals to the Frame Analyser that a FDC is being received (see Frame Analyser description).

An example of the evolution of the Des_Control signals at the moment of passage between two consecutive frames is shown in Fig. 4.50.

4.3.2 THS Detector

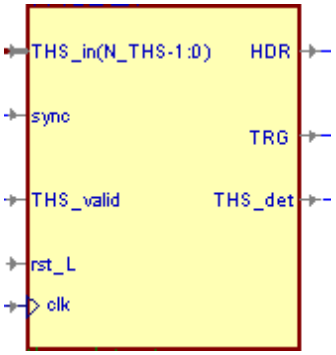


Fig. 4.51 – Symbol of THS_Detector.

Name	Direction	# Bits	Active level	Description
THS_in	IN	6	-	THS bits from Deserializer.
sync	IN	1	high	Signal generated by Synchronizer: indicates acquired synchronization.
THS_valid	IN	1	high	Signal generated by sel_THS_gen_RX: defines the moments when the input signal THS_in contains the THS channel bits.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input, connected to the transmission clock (clk_tx).
HDR	OUT	1	high	Signals that a HDR has been detected in the THS channel.
TRG	OUT	1	high	Signals that a TRG has been detected in the THS channel.
THS_det	OUT	1	high	Signals that a TRG, HDR or NOP sequence has been detected in the THS channel. It is directed to Synchronizer.

Table 4.25 – Description of input and output terminals of THS_Detector.

The THS_Detector checks the THS bits of the received DAT stream (provided by Deserializer) and detects the presence of TRG, HDR and NOP sequences, generating the HDR, TRG, and THS_det output signals. Its internal architecture is shown in Fig. 4.52.

When a TRG, HDR or NOP exact (i.e. without bit-flips) sequence is present in the THS_in input (coming from the Deserializer), the TRG_det, HDR_det or NOP_det internal signal, respectively, is pulled up (it has a one-clk_tx-cycle long pulse), and this in turn causes a pulse on the THS_det output. When a TRG or HDR sequence with one bit-flip is present in the THS_in input, the TRG_det_BF or HDR_trg_BF internal signal, respectively, is pulled up. When TRG_det or TRG_det_BF is high (meaning that a TRG command is detected, exact or with one bit-flip), and THS_valid and THS_enable are also high, the TRG output is pulled up for one clk_tx cycle. When HDR_det or HDR_det_BF is high (meaning that a HDR command is detected, exact or with one bit-flip), and THS_valid and THS_enable and sync are also high, the HDR output is pulled up for

one clk_tx cycle. The filtering of HDR with the sync signal is done to avoid the start of frame reception by the Frame Analyser due to detection of false HDR sequences while the receiver is not synchronized on the THS channel.

The THS_valid signal is used to filter the TRG and HDR detection signals validating only the ones that actually belong to the THS channel: THS_valid is high only in the 4th clk_tx cycle (160, 320 or 640 MHz) of each clk40 cycle, as in that clk_tx cycle the THS_out output of the Deserializer actually contains the THS channel bits. The THS_enable signal, produced by the THS_enable generator, is pulled down when a TRG or HDR sequence (confirmed by THS_valid) is found and set to 1 again after 3 clk40 cycles, to disable the possible recognition of spurious sequences while the last detected one is still to be completed in the THS channel.

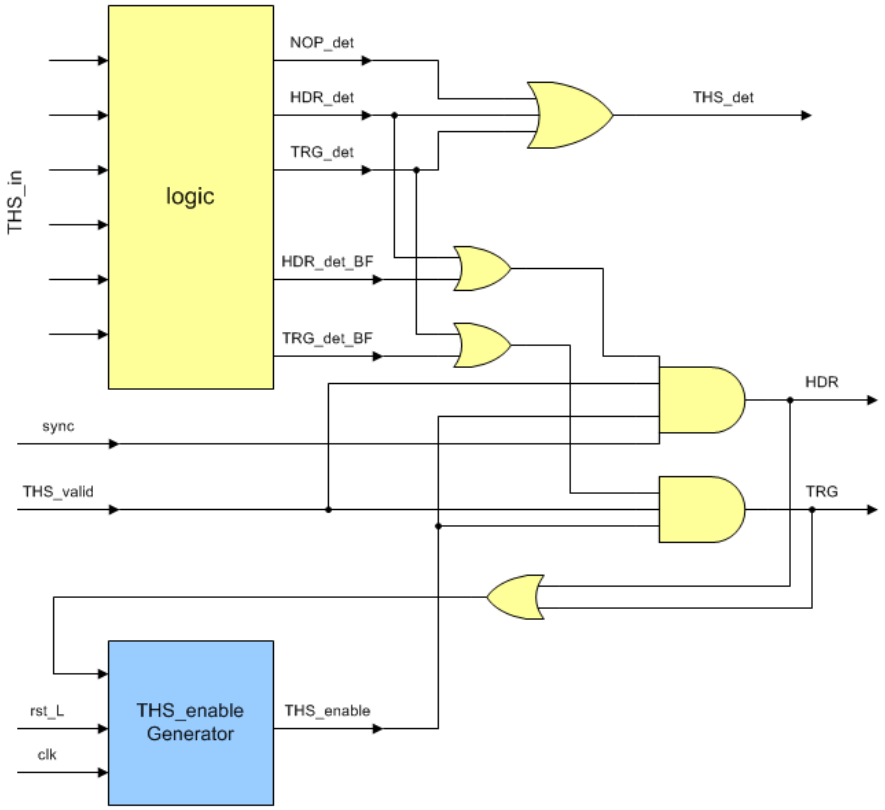


Fig. 4.52 – Internal architecture of THS_detector.

4.3.3 Synchronizer

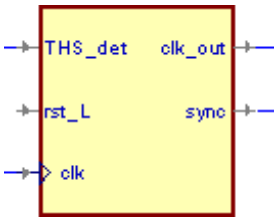


Fig. 4.53 – Symbol of Synchronizer.

Name	Direction	# Bits	Active level	Description
THS_det	IN	1	high	Signal coming from THS Detector.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input, connected to the transmission clock (clk_tx).
clk_out	OUT	1	-	40 MHz reconstructed clock.
sync	OUT	1	high	Signals acquired synchronization.

Table 4.26 – Description of input and output terminals of Synchronizer.

The Synchronizer is the block that keeps the FF_RX synchronized with the THS channel of the DAT stream, reconstructing the 40 MHz clock with the correct phase from the transmission clock (CLK line). The internal architecture of Synchronizer comprises a Clock Splitter, 4, 8 or 16 Sync_Counters (depending on the tx speed configuration) and some combinatory logic arranged as in the block diagram reported in Fig. 4.54, that shows the architecture for the case tx_speed = 160 Mbit/s (the architectures for tx_speed = 320 and 640 Mbit/s are analogous, with 8 and 16 Sync_Counters respectively).

The THS_det input (from the THS_Detector) activates one of the four Sync_Counters, enabling its counting for a clk_in cycle, through the demultiplexer that switches the connection of the input to a different output at every clk_in cycle, being controlled by the count output of the Clk_Splitter. The overall operation is the same as each counter checked one of the four 2-bit channels of the input stream, separated by 1 bit one from another, and counted the THS sequences appearing in that channel.

Consider first an initial situation with all the Sync_Counters reset and the sync signal, that indicates the obtained synchronization, being low. When the transmission starts one of the Sync_Counter (say, the counter i, with i = 0,...,3) begins to detect real THS sequences on the THS channel, while the other Sync_Counters can detect spurious sequences on the other channels. Every time a Sync_Counter detects a THS sequence in the channels it controls, its internal count is incremented: when one of the Sync_Counters (most likely the Sync_Counter i, being aligned with the THS channel) reaches the value N2 it pulls up its set_sync output, that has the effect of setting the sync signal (through the sync logic), and sets its sel output that in turn selects the output clock clk_out as clk_split(i) through the clk_out selector: the Sync_Counter i has become the counter "in charge".

From this moment, each detection of a THS sequence by the Sync_Counter in charge will cause that counter to pull down its clr_all_CNT_L output: this in turn will issue, through the 4-input AND gate, a rst_all_CNT_L command that resets the count in all the non-in-charge Sync_Counters.

If any non-in-charge Sync_Counter reaches the value N1 with its count it pulls up its reset_sync output, that commands the sync logic to reset the sync output in order to signal the (possible) lost of synchronization. After this, the first Sync_Counter that reaches the threshold N2 (it can be the in-charge one or the others) becomes the new in-charge counter, so it set its sel output and sets the sync signal again by means of its set_sync output: this has also the effect, through the leftmost OR gate, of activating the rst_all_FF_L command that resets the sel output in all the other Sync_Counters.

The description of the inner blocks of Synchronizer follows.

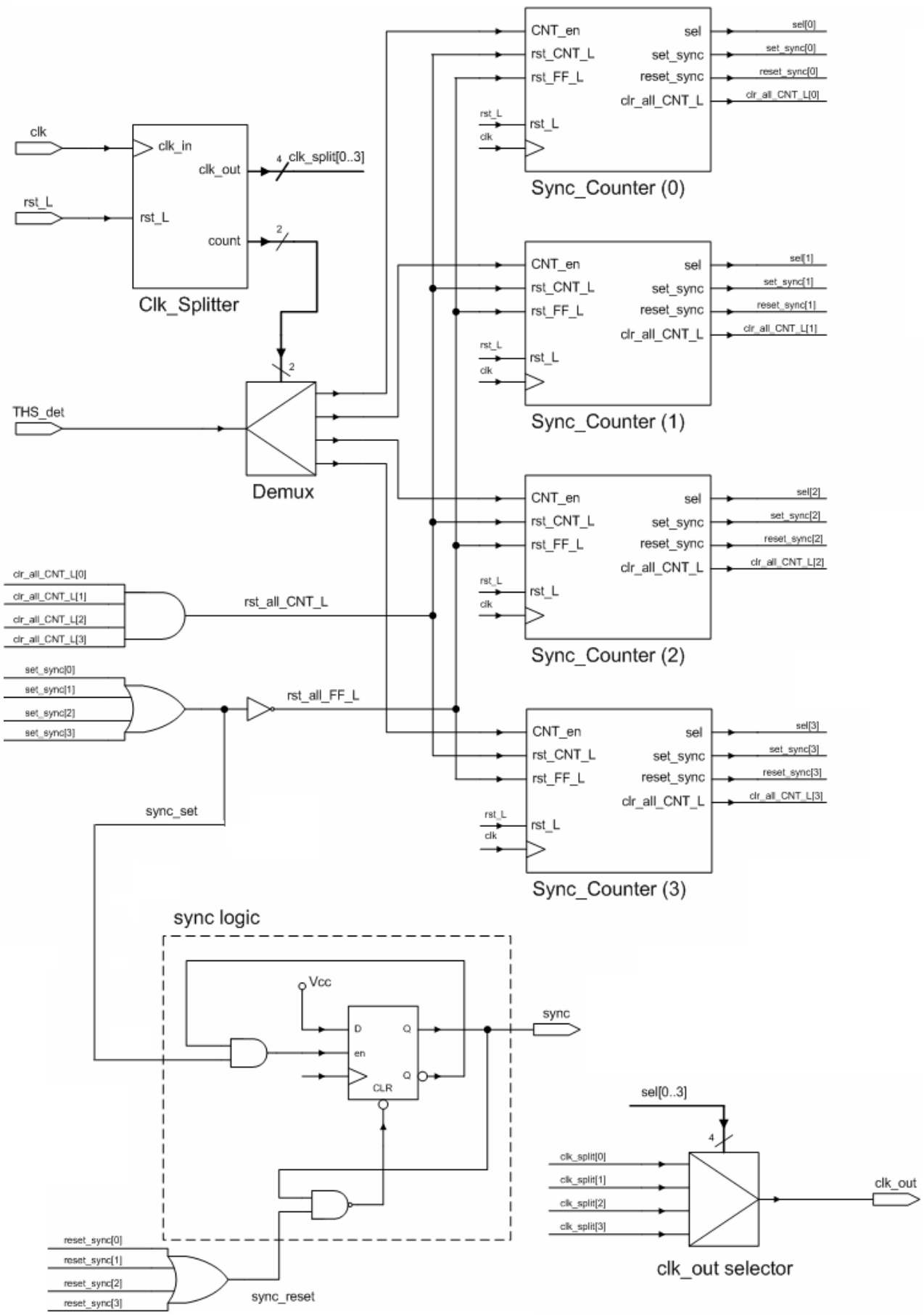


Fig. 4.54 – Internal block diagram of Synchronizer.

4.3.3.1 Clock Splitter

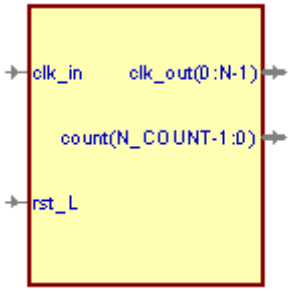


Fig. 4.55 – Symbol of Clk_Splitter.

Name	Direction	# Bits	Active level	Description
clk_in	IN	1	-	Clock input (connected to clk_tx).
rst_L	IN	1	low	Synchronous reset.
clk_out	OUT	N_CYC	-	40 MHz output clocks.
count	OUT	N_CYC_b	-	Output of internal counter, directed to the demux inside the Synchronizer.

Table 4.27 – Description of input and output terminals of Clk_Splitter.

This entity takes an input clock (clk_in) and, by counting its rising edges, generates N output clocks with clk_out_period = clk_in_period·N and phase shifted of one clk_in cycle one from another. The count output has N_COUNT bits and counts the edges of clk_in. If the rst_L input is activated, the count is reset to 0 and all the clk output are set to 0.

The values of N and N_COUNT are passed as generics (integer) to the entity Clk_Splitter: their actual values are N_CYC and N_CYC_b respectively.

4.3.3.2 Sync Counter

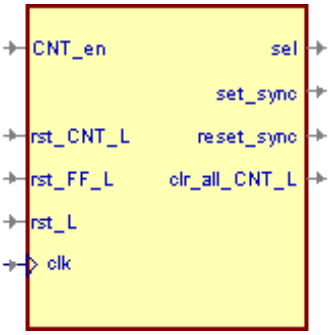


Fig. 4.56 – Symbol of Sync_Counter.

This entity has the following generics:

- CNT_mod (integer): to be passed to the Counter_N as its modulus;
- CNT_b (integer): to be passed to the Counter_N as its number of bits;
- N1 (integer): first threshold;
- N1 (integer): second threshold.

The internal architecture is shown in Fig. 4.57.

When the CNT_en input is high at the rising edge of the clock (clk input, that is connected to the transmission clock) the counter CNT increments its count value. When the count is equal to N2, the N2_reach signal is pulled up which in turn sets the flip-flop FF and thus the sel output: the Sync_Counter becomes in this way the counter in charge. The N2_reach signal also coincides with

the set_sync output, so the reaching of the threshold N2 has also the effect of setting the sync output of the Synchronizer. If the present Sync_Counter is in charge (sel output high), each pulse on the CNT_en input generates also a negative pulse on the clr_all_CNT_L output command that has the effect of resetting all the other Sync_Counters. In fact, in a Sync_Counter that is not in charge the signal sel is low and thus the rst_CNT_L input active low command effectively resets the counter through the OR and the AND gates.

Name	Direction	# Bits	Active level	Description
CNT_en	IN	1	high	Signal coming from the sel_THS Generator: identifies the THS channel bits inside each clk40 period.
rst_CNT_L	IN	1	low	Reset command for the internal counter.
rst_FF_L	IN	1	low	Reset command for the internal flip-flop.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input (connected to clk_tx).
sel	OUT	1	high	Selection output.
set_sync	OUT	1	high	Commands the setting of sync.
reset_sync	OUT	1	high	Commands the resetting of sync.
clr_all_CNT_L	OUT	1	low	Reset command for all other counters.

Table 4.28 – Description of input and output terminals of Sync_Counter.

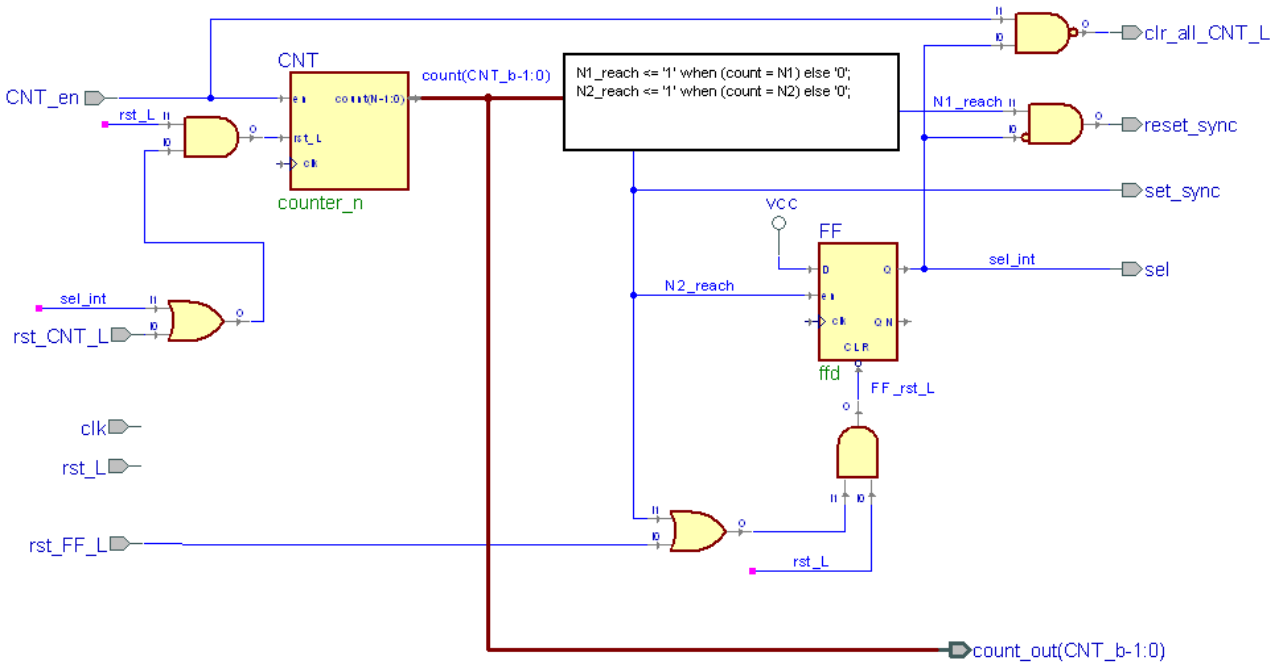


Fig. 4.57 – Internal block diagram of Sync_Counter.

The flip-flop FF can be reset, thus removing the Sync_Counter from its in-charge condition, by the active low input rst_FF_L that is activated when one of the Sync_Counters in the Syncronizer reaches the threshold N2 (see Synchronizer diagram): if the Sync_Counter reaching N2 is not this Sync_Counter, the signal N2_reach will be low too at that time and therefore the rst_FF_L command will effectively reset the FF and thus the sel output.

When the count signal is equal to N1, the N1_reach signal is pulled up, which in turn activates the reset_sync output if the sel output is not set, i.e. if this Sync_Counter is not the counter in-charge:

this way, the sync output of the Synchronizer is reset. The active low rst_L input resets both the counter CNT and the flip-flop FF.

Fig. 4.58 shows two examples of the evolution of the Synchronizer signals in the case tx_speed = 320 Mbit/s:

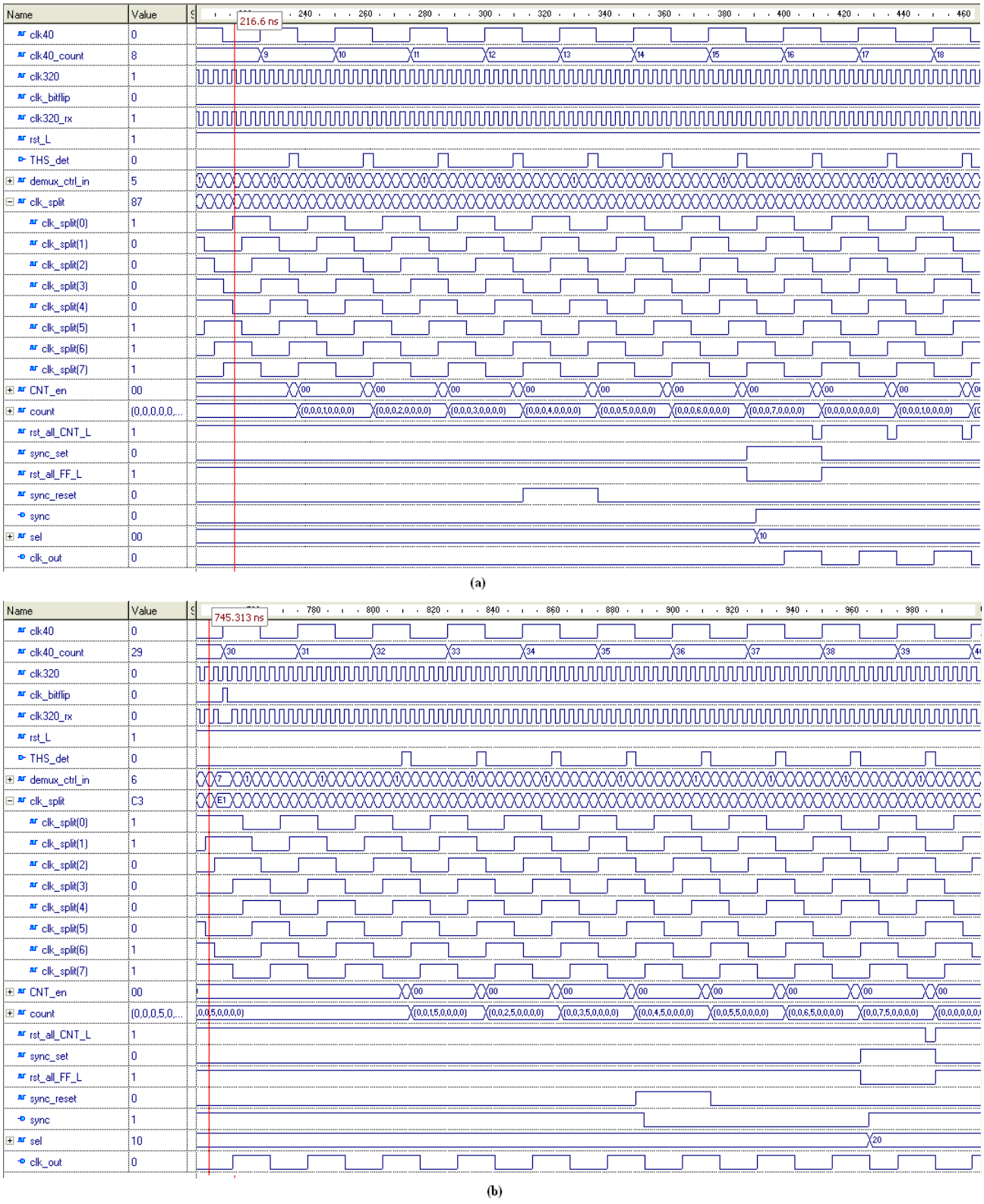


Fig. 4.58 – Simulation showing the evolution of the Synchronizer signals (tx_speed = 320 Mbit/s) at the beginning of the transmission (first sync lock acquired) (a) and after the loss of sync lock due to a bit-flip on the CLK line (clk_bitflip signal going high) (b): the threshold are set as N1 = 4, N2 = 7.

4.3.4 Frame Analyser

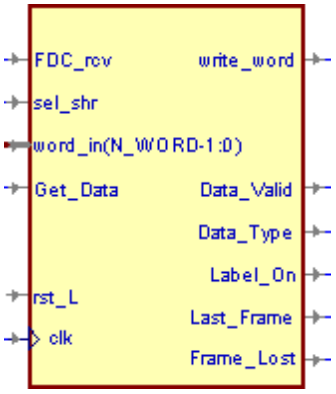


Fig. 4.59 – Symbol of Frame_Analyser.

Name	Direction	# Bits	Active level	Description
FDC_rcv	IN	1	high	Signal generated to by Deserializer to indicate that the FDC field of a frame is being received.
sel_shr	IN	1	-	Signal generated to by Deserializer to indicate which shift register is in use.
word_in	IN	16	-	Input for FDC data coming from Deserializer.
Get_Data	IN	1	high	Get_Data input of FF_RX; directed to FA_Control2 inside the Frame Analyser.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	Clock input, connected to the 40 MHz clock (clk40).
write_word	OUT	1	high	Write command for RX Buffer.
Data_Valid	OUT	1	high	Data_Valid output of FF_RX (see Table 3.2).
Data_Type	OUT	1	-	Data_Type output of FF_RX (see Table 3.2).
Label_On	OUT	1	high	Label_On output of FF_RX (see Table 3.2).
Last_Frame	OUT	1	high	Last_Frame output of FF_RX (see Table 3.2).
Frame_Lost	OUT	1	high	Frame_Lost output of FF_RX (see Table 3.2).

Table 4.29 – Description of input and output terminals of Frame_Analyser.

The task of the Frame Analyser is to decode and store the Frame Descriptors of received frames, to command the writing of the received frame words in the RX Buffer and to manage the delivery of the frames stored in the RX Buffer to the outside world, through the Data_Valid/Get_Data handshake. To simplify the interfacing with the RX Buffer and the external circuits, this block has been designed so as to work with the 40 MHz clock; the internal architecture is shown in Fig. 4.60.

The operation of the Frame Analyser is managed by two state machines: FA_Control1 and FA_Control2. FA_Control1 deals with the writing of the received frame words into the RX Buffer, by means of the write_word command, and the storing of Frame Descriptors. Each Frame Descriptor (FDC) received by the Deserializer, available at the input word_in, is decoded by the Hamming Decoder, temporarily stored in the FD_reg register and finally saved in the FD_FIFO when the reception of the frame is completed. The FD_reg register and each location of the FD_FIFO are N_FD+1 bit long, since they contain the FD field and, in addition, a “Frame_Lost” bit that comes from the error output of the FD Decoder (FD_err signal) and indicates that the frame has not been received because its FDC was corrupted by a double-bit error.

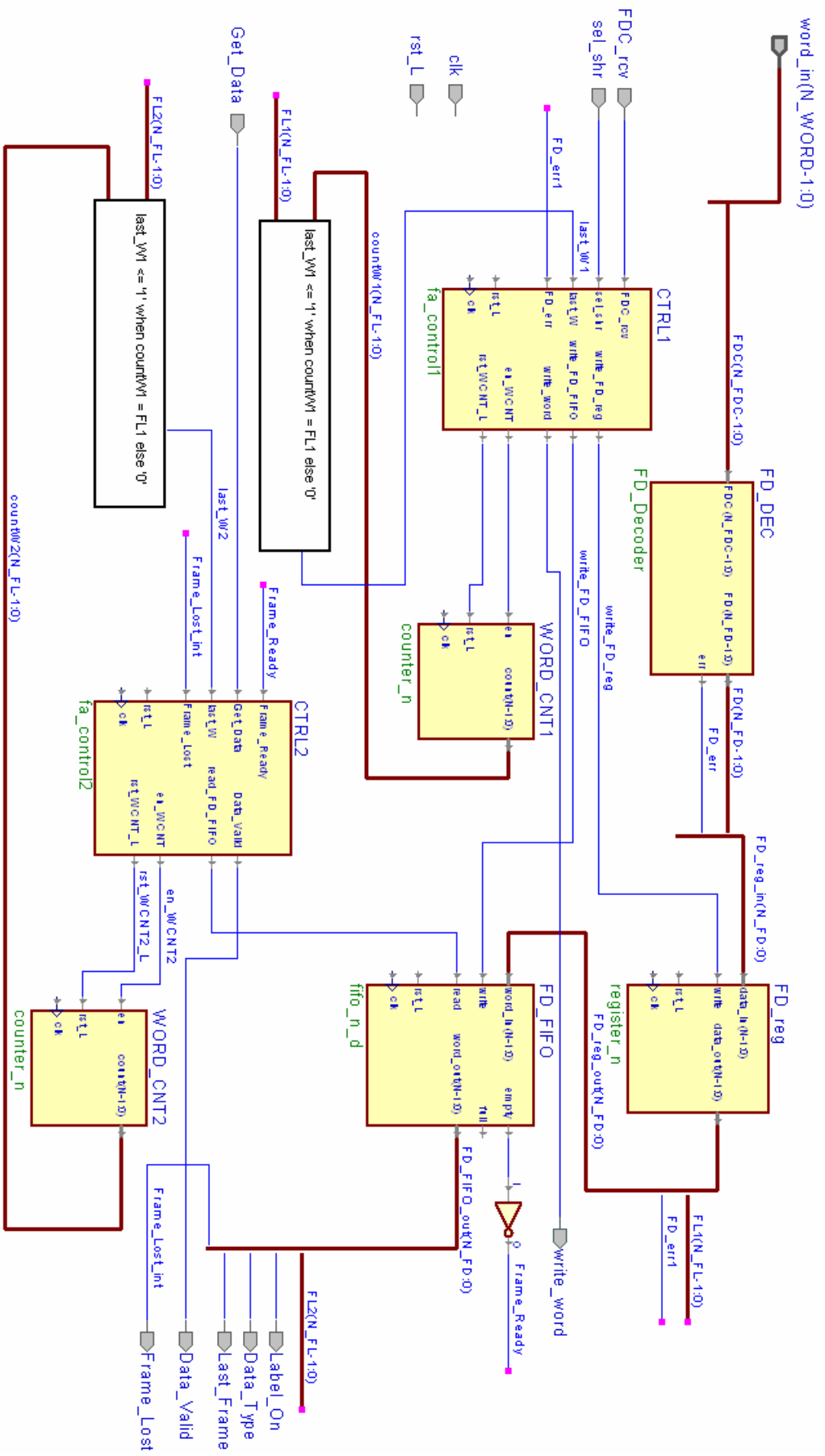


Fig. 4.60 – Internal block diagram of Frame_Analyser.

During the reception of a frame, FA_Control1 uses the Word Counter n°1 (WORD_CNT1) to count the received words thus being able to determine the frame's end through the last_W1 signal: this latter is obtained by comparing the word count (count_W1) with the Frame Length of the currently received frame (FL1) available from the Frame Descriptor stored in the FD_reg.

FA_Control2 handles the reading of the frames stored in the RX Buffer by the external circuits: when some Frame Descriptor is available in the FD_FIFO, FA_Control2 pulls up the Data_Valid output to notify to the outside the presence of a frame to be delivered, and during the delivery of the frame it uses the Word Counter n°2 (WORD_CNT2) to determine the ending of the frame (similarly to last_W1, the last_W2 signal is now used for this purpose, obtained by comparing the second word count (count_W2) with the Frame Length of the delivered frame (FL2) available from the Frame Descriptor that is being read from the FD_FIFO). At the end of the frame, Data_Valid is pulled down for at least one clk40 cycle to indicate the finish of the frame to the external reader, and the FD_FIFO is read to look for possible other frames to be delivered. The frame reading process can be put on hold by the external circuit by pulling down the Get_Data signal, thus stopping the FA_Control2 state machine; the read command for the RX Buffer is obtained as (Data_Valid AND Get_Data), so that a new word is read only when the external circuit is ready to receive it.

A more detailed description of these components follows.

4.3.4.1 FA_Control1

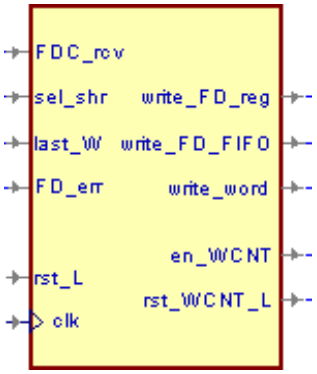


Fig. 4.61 – Symbol of FA_Control1.

Name	Direction	# Bits	Active level	Description
FDC_rcv	OUT	1	high	Signal directed to Frame Analyser to indicate that the FDC field of a frame is being received by Deserializer.
sel_shr	IN	1	-	Signal generated to by Deserializer to indicate which shift register is in use.
last_W	IN	1	high	Last_W1 signal generated from the output of WCNT1.
FD_err	IN	1	high	Error signal from FD Decoder.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	40 MHz input clock.
write_FD_reg	OUT	1	high	Write command for FD register.
write_FD_FIFO	OUT	1	high	Write command for FD FIFO.
write_word	OUT	1	high	Write command for RX Buffer.
en_WCNT	OUT	1	high	Count enable command for Word Counter 1.
rst_WCNT_L	OUT	1	low	Reset command for Word Counter 1.

Table 4.30 – Description of input and output terminals of FA_Control1.

This block is a Mealy state machine that controls the Word Counter 1 and generates the write commands for the FD_reg, the FD_FIFO and the RX Buffer. The state diagram is shown in Fig. 4.62:

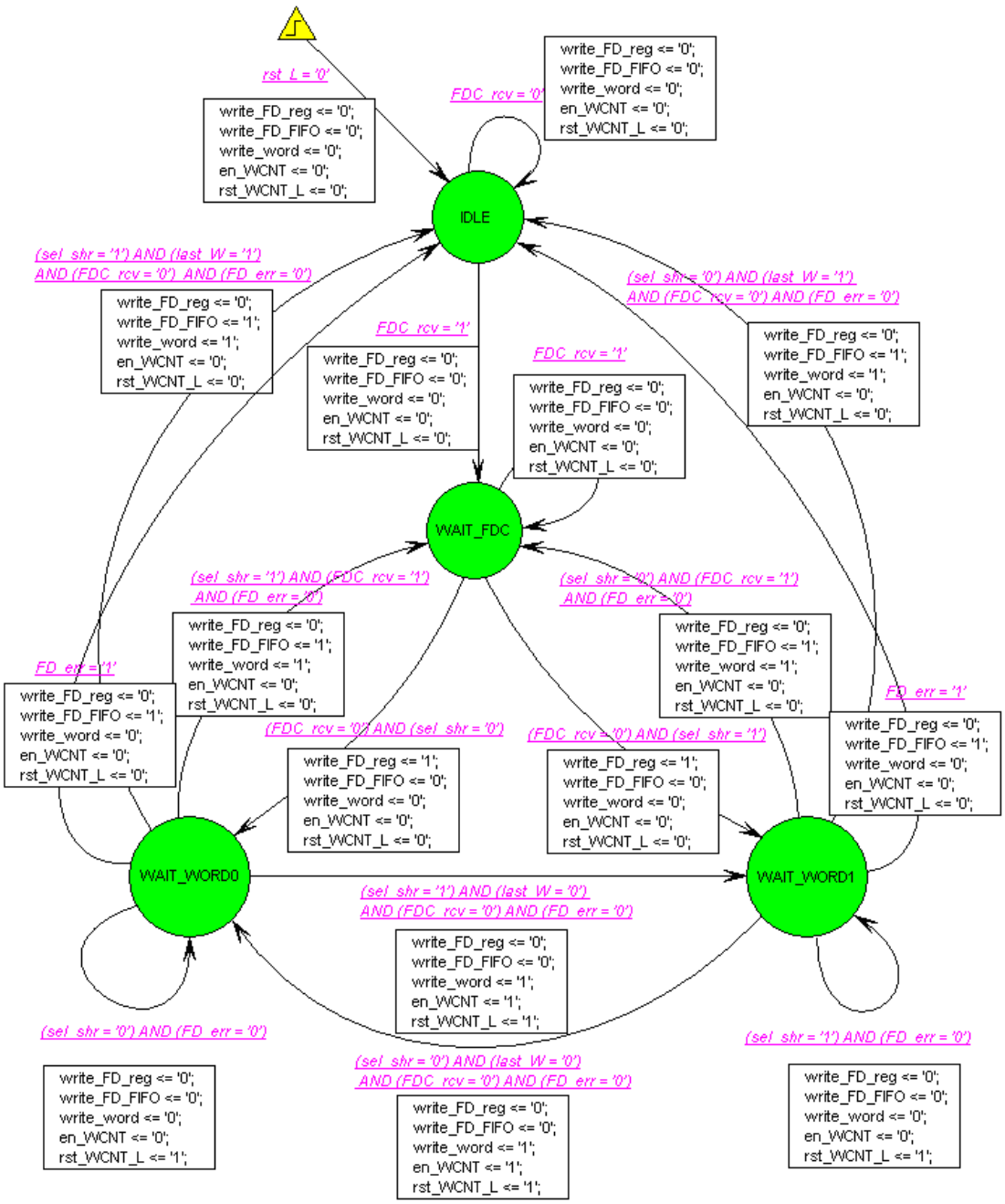


Fig. 4.62 – State diagram of FA_Control1.

At the reset, the FSM goes to the IDLE state resetting the word counter (W_CNT1), and in this state it waits for the signal FDC_rcv, that comes from the Deserializer and indicates, when it is high, that a FDC is being received. When FDC_rcv becomes high, FA_Control1 moves to the state WAIT_FDC: a frame reception is started in the Deserializer and so the FA_Control1 prepares to begin its activity. When FDC_rcv is pulled down again, meaning that the reception of the FDC field by the Deserializer is completed, FA_Control1 goes to the state WAIT_WORD0 or WAIT_WORD1 whether the sel_shr signal (coming from the Deserializer as well) is 0 or 1, writing the FD field into the FD_reg (write_FD_reg <= '1'). The FD field has been calculated in the meanwhile by the Hamming Decoder from the FDC field, that is made available at the word_in input of the Frame Analyser when FDC_rcv goes low. The sel_shr signal was used in the

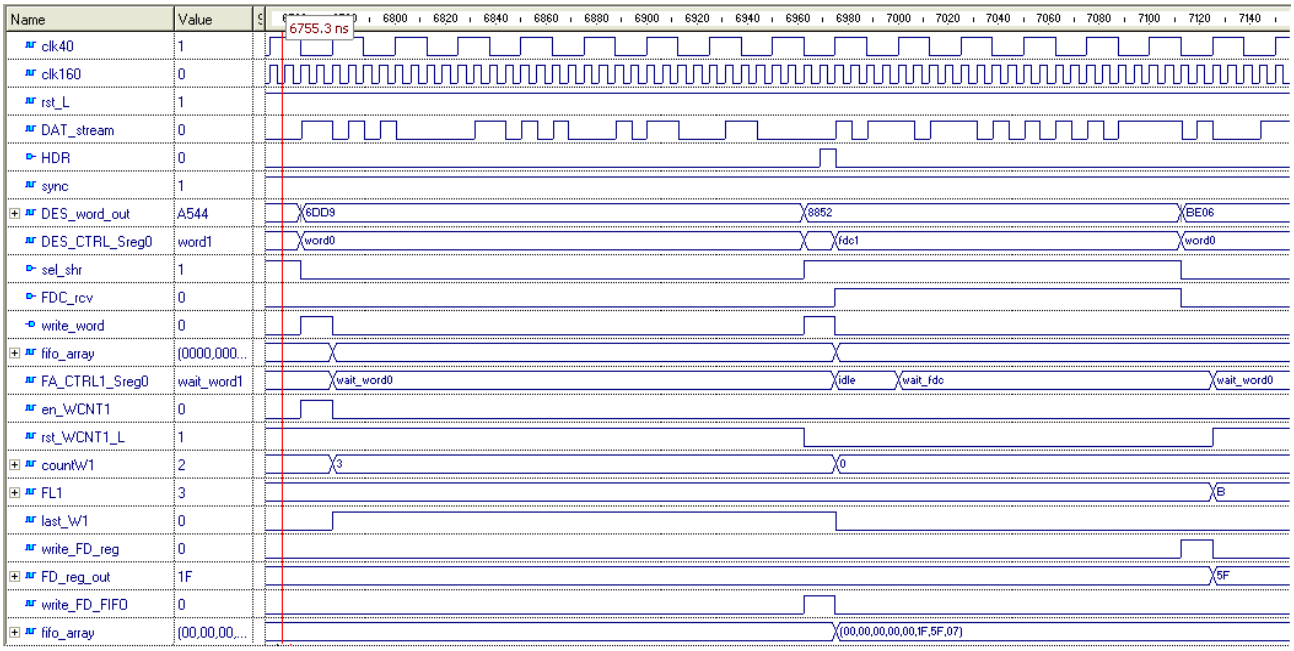
Deserializer to select the shift register to be loaded with the incoming bit stream: here sel_shr is simply used to figure out the moments when each word has been completely received by the Deserializer and is thus available for writing in the RX Buffer.

Supposing for the moment that FD_err = '0' (i.e. the FDC has been decoded without errors), in the state WAIT_WORD0 the FA_Control1 FSM waits for the reception of a word in the SHR0 of the Deserializer to be completed: this is indicated by the transition of sel_shr from 0 to 1, so when this happens the FSM moves into the state WAIT_WORD1 enabling the word counter to count a word (en_WCNT <= '1') and writing the received word in the RX Buffer (write_word <= '1'). The same behavior is carried on in the state WAIT_WORD1, with the difference that here the transition of sel_shr from 1 to 0 is awaited. In both the states WAIT_WORD0 and WAIT_WORD1, at the end of the reception of a word (i.e. when sel_shr = 1 and 0, respectively), the last_W input (connected to the last_W1 signal) and FDC_rcv are checked as well: if last_W is high and FDC_rcv is low, meaning that the word that has been received is the last of the frame but the Deserializer is not already receiving a new FDC, the FSM goes back to the IDLE state, writing the last word in the RX Buffer and resetting the word counter; if instead FDC_rcv is high, indicating that the FDC of a new frame is being deserialized, FA_Control1 goes directly to the state WAIT_FDC to start the new frame reception.

If instead a double-bit error has been detected in the received FDC by the FD_Decoder, the input FD_err will be high when the decoded FD is written into the FD_reg, that is, when FA_Control1 moves from the state WAIT_FDC to the state WORD0/1: if this is the case, FA_Control1 goes back immediately to the IDLE state renouncing to the frame reception (no word is written into the RX Buffer) but writing into the FDC_FIFO the FDC of the lost frame (with the Lost_Frame bit set).

The FA_Control1 outputs are not registered because in the case tx speed = 640 Mbit/s there is less than two clk40 cycles of time for FDC and for each incoming word to be stored, and so the delay introduced by the registering of the FA_Control1 output would not allow a correct functionality of the Frame Analyser; at 160 and 320 Mbit/s, instead, the FA_Control1 outputs can be registered, and in particular this would have the advantage of more time for the Hamming Decoder to perform its calculation (from the moment when FDC is available at the Deserializer output to the moment when FD is written in the FD_reg).

An example of the evolution of the FA_Control1 signals is shown Fig. 4.63:



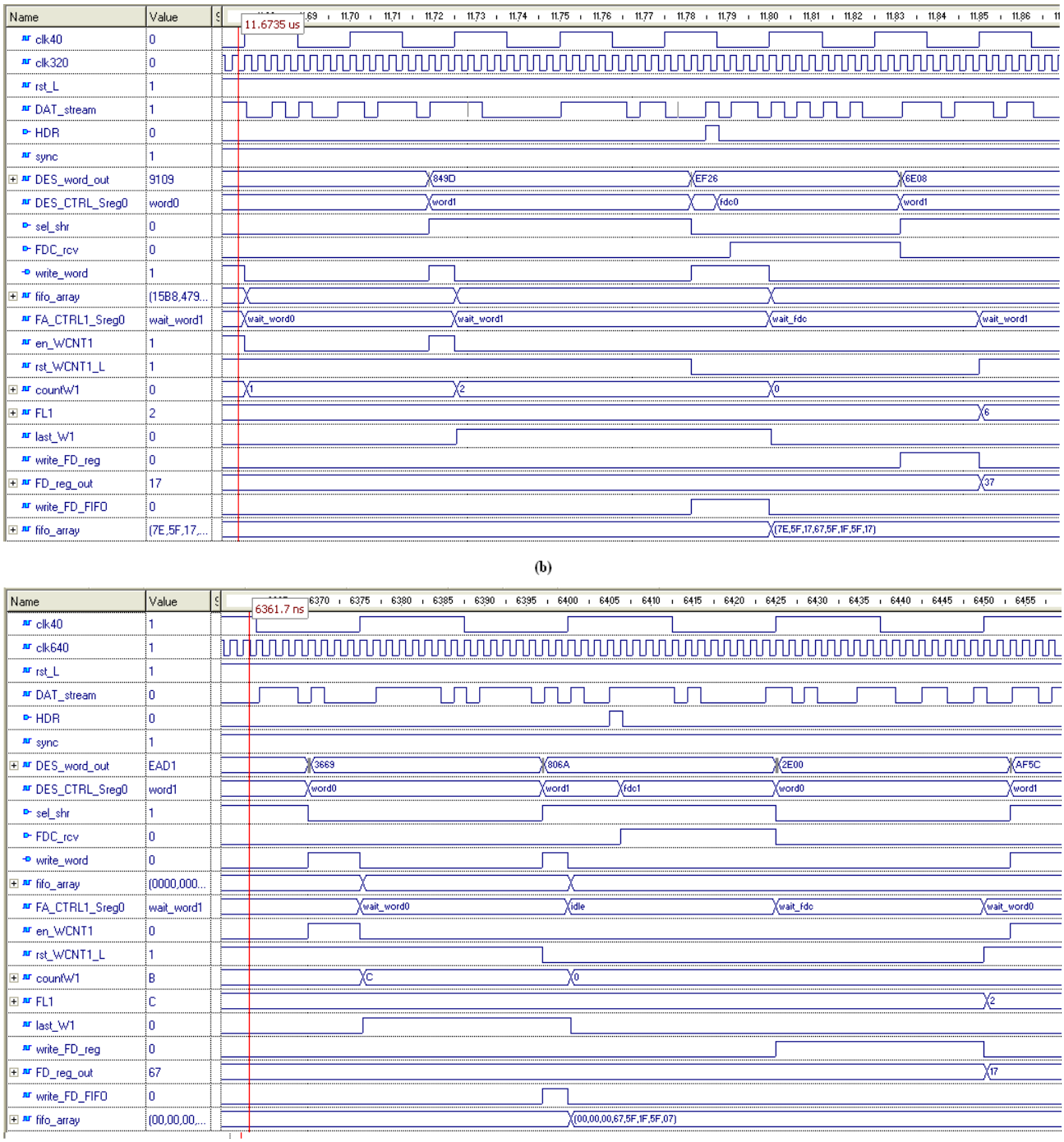


Fig. 4.63 – Simulation showing the evolution of the FA_Control1 signals at the end of a frame reception, with a new frame reception following immediately after; tx speed = 160 (a), 320 (b) and 640 Mbit/s (c). The first signal named "fifo_array" is the content of RX Buffer; the second is the content of FD_FIFO.

4.3.4.2 FA_Control2

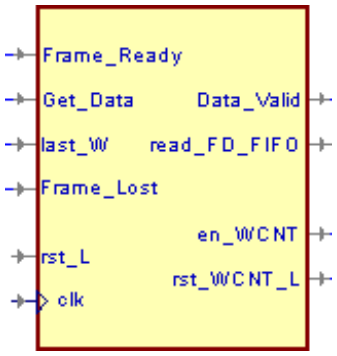


Fig. 4.64 – Symbol of FA_Control2.

Name	Direction	# Bits	Active level	Description
Frame_Ready	IN	1	high	Signals that the FDC_FIFO is not empty.
Get_Data	IN	1	high	Signals that the host is able to read a new word from the word_out port
last_W	IN	1	high	Last_W2 signal generated from the output of WCNT2.
Frame_Lost	IN	1	high	LSB of signal FDC_FIFO_out: indicates that the current Frame Descriptor is relative to a frame that has been lost due to the corruption of its FDC.
rst_L	IN	1	low	Synchronous reset.
clk	IN	1	-	40 MHz input clock.
Data_Valid	OUT	1	high	Data_Valid output of FF_RX (see Table 3.2).
read_FD_FIFO	OUT	1	high	Read command for FD FIFO.
en_WCNT	OUT	1	high	Count enable command for Word Counter 2.
rst_WCNT_L	OUT	1	low	Reset command for Word Counter 2.

Table 4.31 – Description of input and output terminals of FA_Control2.

This block is a Mealy machine that controls the Word Counter 2 and manages the handshake with the external reader of the received frames. The state diagram is shown in Fig. 4.65.

At the reset, the FSM goes to the IDLE state resetting the word counter (W_CNT2), and in this state it waits for the signal Frame_Ready to become high: Frame_Ready is the inversion of the FD_FIFO empty output, so it indicates that at least one Frame Descriptor is present in the FD_FIFO, and hence that a frame is present in the RX Buffer.

When Frame_Ready goes high, FA_Control2 moves to the state DELIV_FRAME (delivering frame) pulling up Data_Valid to signal that a word is available at the RX Buffer for reading. Then, as long as the frame delivering is not finished (last_W = 0) the FA_control2 remains in the state DELIV_FRAME increasing the word counter if Get_Data is high (meaning that a word has been read by the external circuit) or leaving it unchanged if instead Get_Data is low. When the frame reading is completed (last_W = 1) the FSM returns to the IDLE state pulling down Data_Valid and commanding a reading of the FD_FIFO, so that in the next clock cycle it will know, through the Frame_Ready signal, if there is another frame to be delivered, in which case a new frame delivery cycle is started by going to the DELIV_FRAME state again.

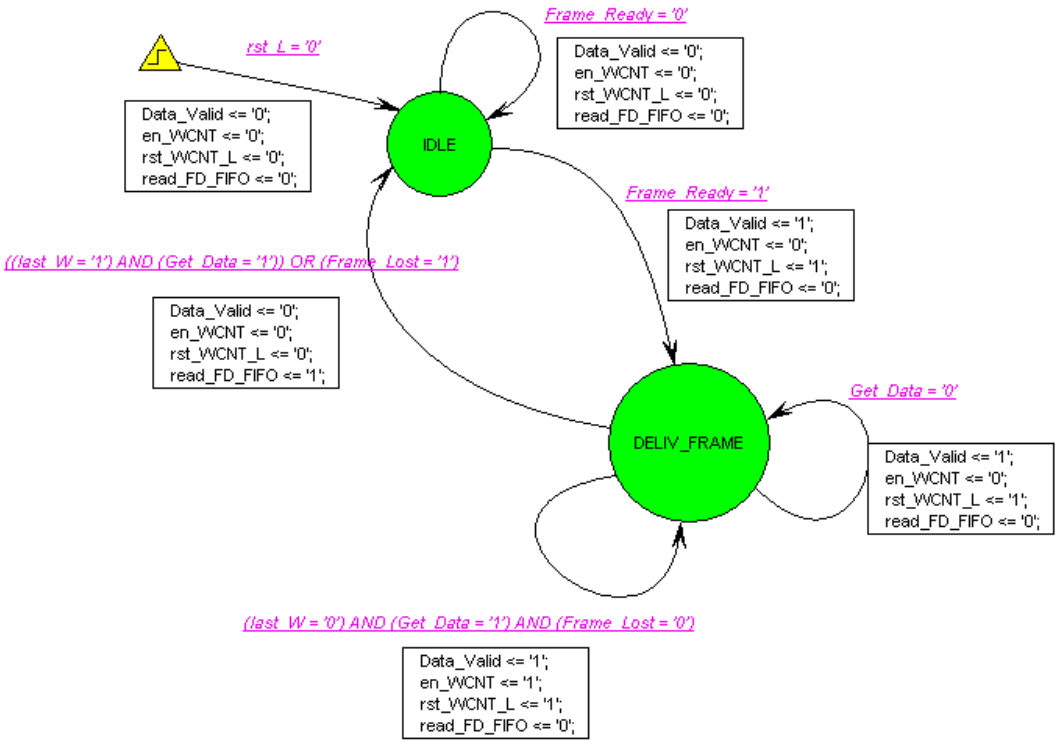


Fig. 4.65 – State diagram of FA_Control2.

If the Frame Descriptor present in the FD_FIFO is relative to a frame that has been lost due to the corruption of its FDC, the Frame_Lost signal will be high when FA_Control2 enters the state DELIV_FRAME: in this case, provided that the host is ready to read (i.e. Get_Data is high), FA_Control2 goes back to the IDLE state after just one cycle of permanence in the DELIV_FRAME state, still issuing a read_FD_FIFO command to pass on to the next FD. Therefore, the lost frame is indicated to the host by means of a one-clk40-cycle long Data Valid pulse, with the Frame_Lost output being high at the same time. No word is read from the RX Buffer in this case, since its read command is given by (Data_Valid AND Get_Data AND NOT(Frame_Lost)) (see FF_RX block diagram).

Note that, since a FD is written into the FD_FIFO only when the reception of the relative frame is completed, the delivery of a frame to the external user can go on without interruption by the Frame Analyser because all the frame is already available in the RX Buffer.

An example of the evolution of the FA_Control2 signals during the delivery of a 7-words frame followed by a 3-words frame is shown in Fig. 4.66.

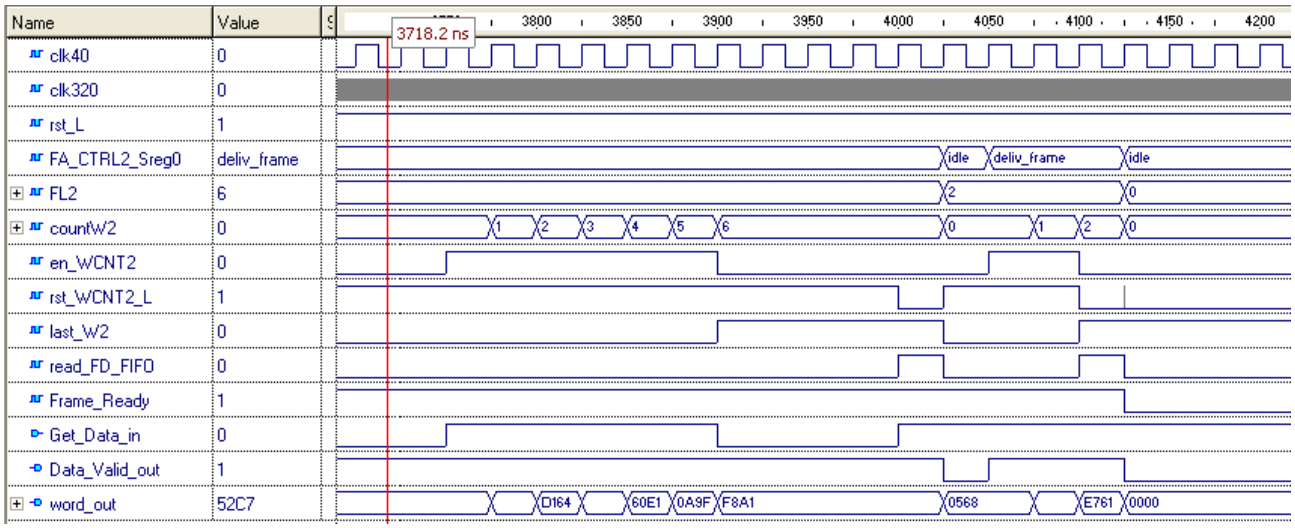


Fig. 4.66 – Simulation showing the behavior of FA_Control2 during the delivery of a 7-words frame followed by a 3-words frame (tx speed = 320 Mbit/s).

4.3.4.3 FD Decoder

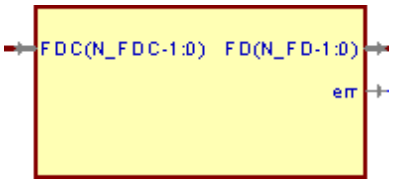


Fig. 4.67 – Symbol of FD_Decoder.

Name	Direction	# Bits	Active level	Description
FDC	IN	12	-	Coded Frame Descriptor.
FD	OUT	12	-	Decoded Frame Descriptor.
err	OUT	1	high	Signals a double bit error in FDC.

Table 4.32 – Description of input and output terminals of FD_Decoder.

This block performs the H(12,7) decoding of the received coded Frame Descriptor (FDC input) reconstructing the transmitted FD (FD output). Single bit errors are corrected, while double bit errors are signaled on the err output. Its architecture is defined by the decoding algorithm described in section 2.3.1.1, in particular by equations (2.23) and by Table 2.2 and Table 2.3.

4.3.5 sel_THS Generator RX

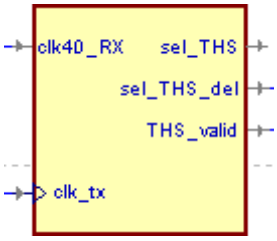


Fig. 4.68 – Symbol of sel_THS_gen_RX.

Name	Direction	# Bits	Active level	Description
clk40_RX	IN	1	-	40 MHz reconstructed clock.
clk_tx	IN	1	-	Transmission input clock.
sel_THS	OUT	1	high	Identifies the THS channel bits inside each clk40 period.
sel_THS_del	OUT	1	high	sel_THS signal delayed by one clk_tx cycle: it's directed to Deserializer.
THS_valid	OUT	1	high	Signal directed to THS Detector: defines the moments when the input signal THS_in contains the THS channel bits.

Table 4.33 – Description of input and output terminals of sel_THS_gen_RX.

This block uses the 40 MHz reconstructed clock (clk40_RX input) to generate the sel_THS_del and THS_valid signals needed by Deserializer and the THS_Detector. The internal architecture is shown in Fig. 4.69:

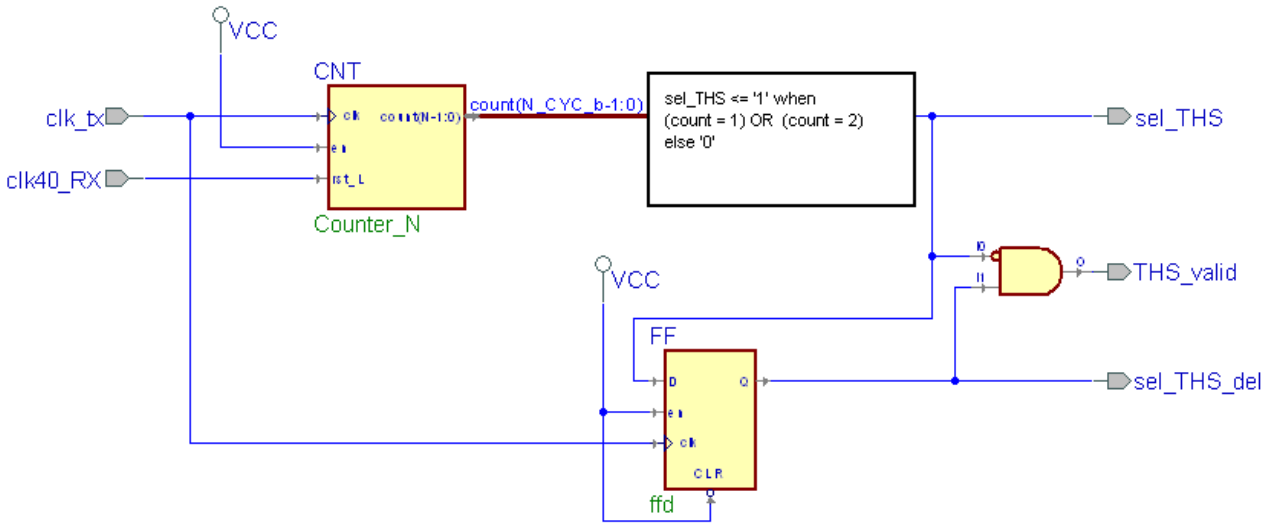


Fig. 4.69 – Internal block diagram of sel_THS_gen_TX.

The counter CNT, driven by clk_tx, is reset when clk40_RX is low, so it will start its counting after the clk40 rising edge. The sel_THS output is high only when the count equals 1 or 2 (i.e. in the second and third clk_tx cycle after the rising edge of clk40); by delaying sel_THS with the flip-flop FF the sel_THS_del output is generated, and THS_valid is obtained as (sel_THS_del) AND NOT(sel_THS), so it is high only in the fourth clk_tx cycle after the rising edge of clk40, that is the time when a real THS sequence can be present in the THS_out output of the Deserializer.

A simulation on this block is shown in Fig. 4.70.

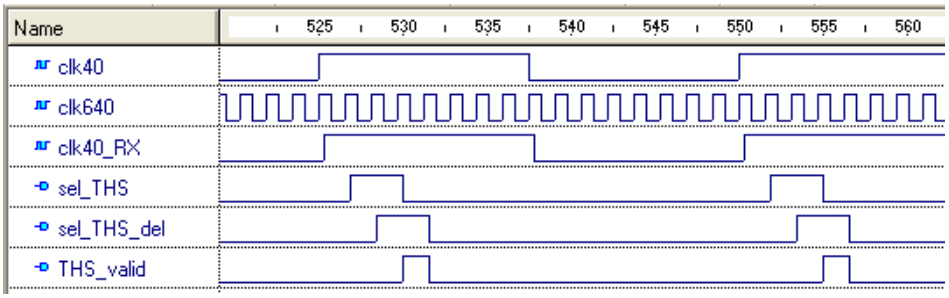


Fig. 4.70 – Simulation showing the operation of sel_THS_gen_RX (tx speed = 640 Mbit/s).

4.3.6 TRG stretcher

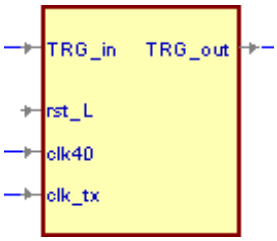


Fig. 4.71 – Symbol of TRG_strch..

Name	Direction	# Bits	Active level	Description
TRG_in	IN	1	high	One-clk_tx-cycle long input pulse.
rst_L	IN	1	low	Synchronous reset.
clk40	IN	1	-	40 MHz input clock.
clk_tx	IN	1	-	Transmission input clock.
TRG_out	OUT	1	high	One-clk40-cycle long output pulse.

Table 4.34 – Description of input and output terminals of TRG_strch.

The task of this block is to generate a one-clk40-cycle long pulse for the TRG output in response to each one-clk_tx-cycle long pulse that arrives on the TRG_in input. There are two different internal architecture for entity TRG_strch: one for the 160 Mbit/s configuration (TRG_strch160) and one for the 320 and 640 Mbit/s configurations (TRG_strch320_640).

The TRG_strch160 architecture is shown in Fig. 4.72. It simply consists of a D-flip-flop, clocked by clk40; when a TRG_in pulse arrives from the THS detector (this occurs in the fourth 160 MHz clock cyle of a clk40 cycle, and hence the high level of TRG_in is sampled by the rising edge of clk40) the TRG_out signal is set, and is then pulled down again at the next clk40 positive edge, when the QN signal resets the flip-flop. A simulation is shown in Fig. 4.73.

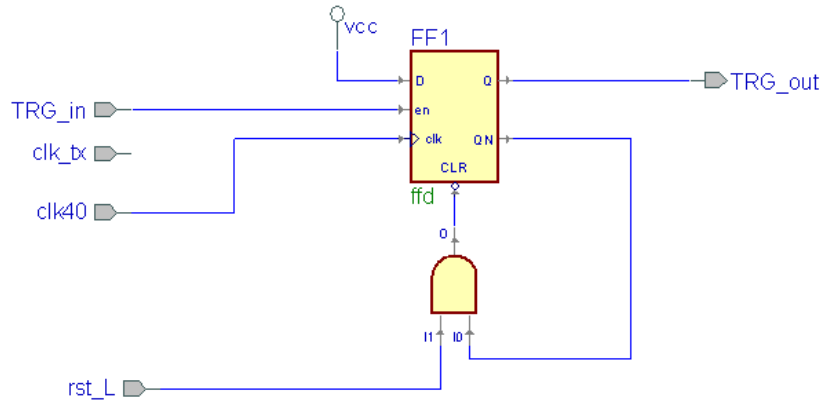


Fig. 4.72 – Internal block diagram of TRG_strch160 architecture.

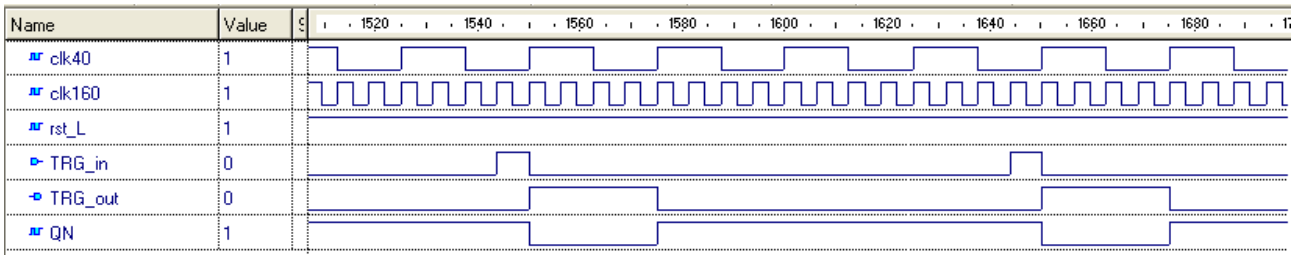


Fig. 4.73 – Simulation showing the operation of TRG_strch160 architecture.

The TRG_strch320_640 architecture is shown in Fig. 4.74. When a TRG_in pulse arrives from the THS detector the first flip-flop (FF1) is set, so that at the next clk40 rising edge also the second flip-flop (FF2) is set by Q1 being high. This in turn causes the reset of FF1 (by QN2 being low) at the next clk_tx positive edge and consequently the reset of FF2, too, at the next clk40 positive edge.

A simulation is shown in Fig. 4.75.

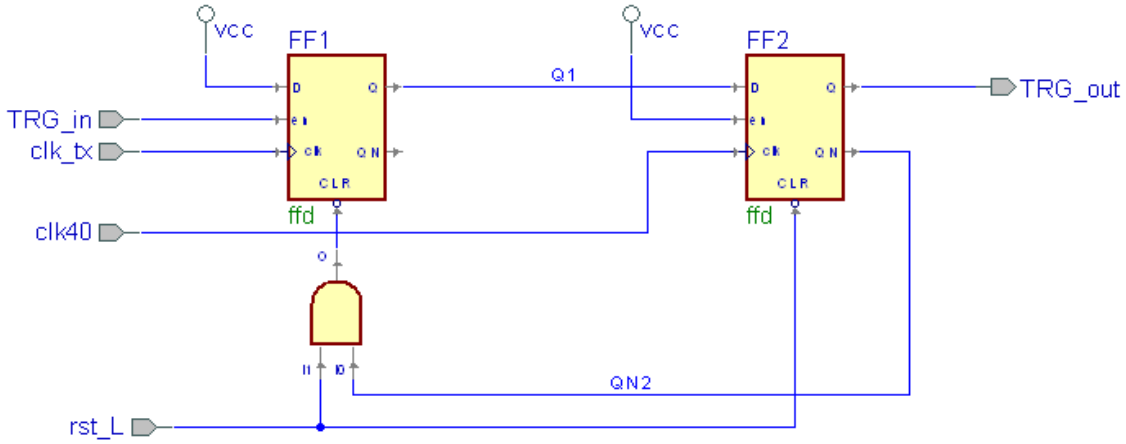


Fig. 4.74 – Internal block diagram of TRG_strch320_640 architecture.

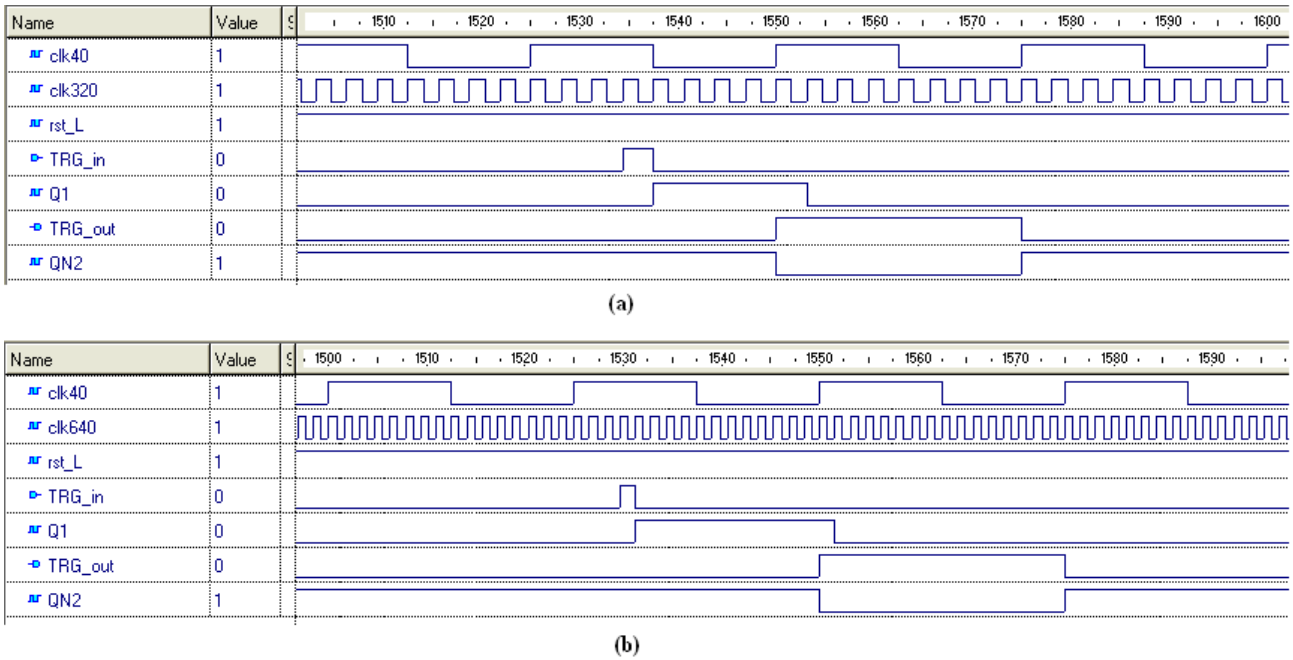


Fig. 4.75 – Simulation showing the operation of TRG_strch320_640 architecture, for tx speed = 320 Mbit/s (a) and 640 Mbit/s (b) .

4.3.7 RX Buffer

RX Buffer is the FIFO memory that stores the received frame words awaiting to be delivered to the receiver host. It is realized with a FIFO_N_D entity (see section 4.1.1): the write command is provided by the Frame Analyser, while the read command is obtained as Data_Valid AND Get_Data AND NOT(Frame_Lost), so that a new word is actually read when the Frame Analyser is delivering a not corrupted frame (i.e. when Data_Valid is activeand Frame_Lost is not) and the host is able to read the word (Get_Data active). It is clocked with the 40 MHz clock (clk40).

4.4 Test bench

To test the functionality of the transmitter-receiver system, a VHDL test bench was created with the structure shown in Fig. 4.76:

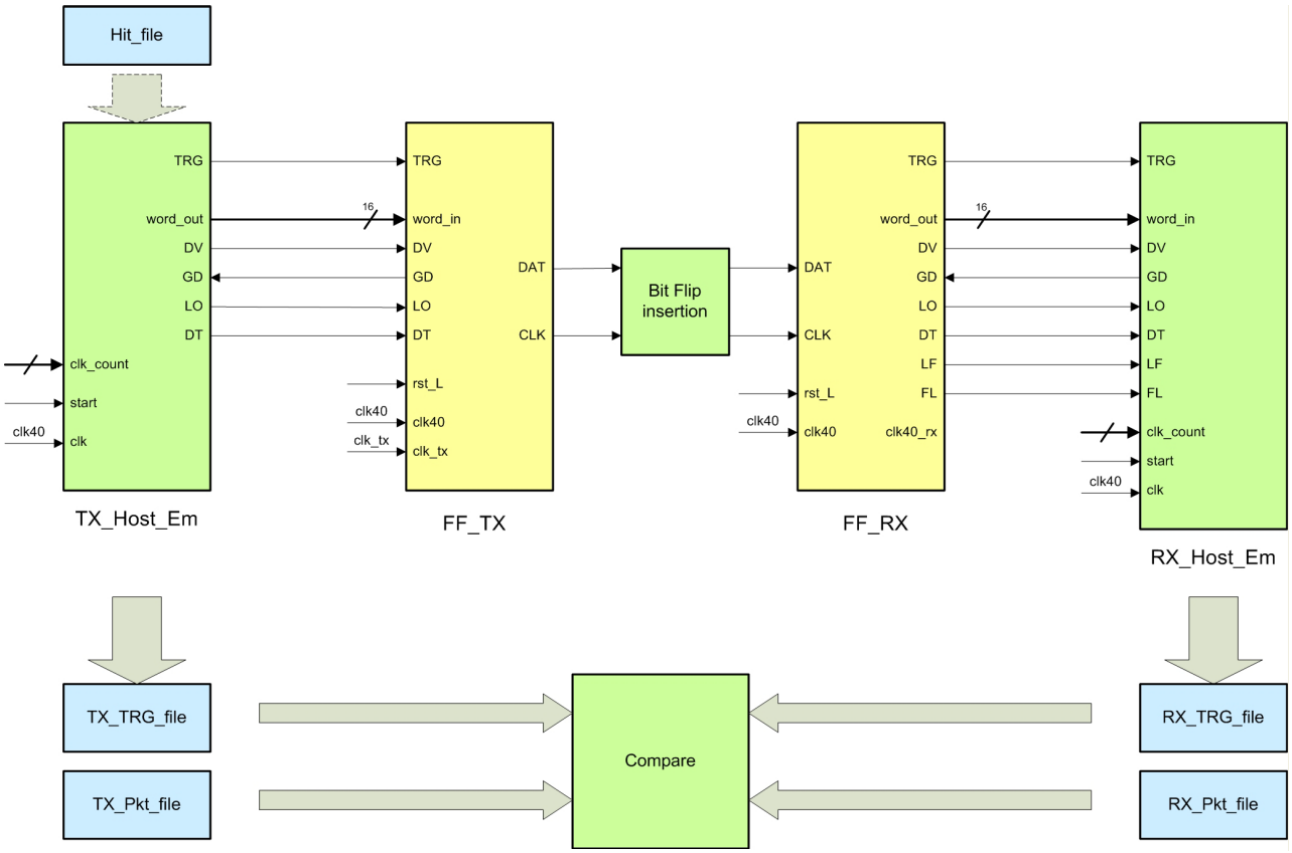


Fig. 4.76 – Architecture of the test bench for FF_TX-FF_RX.

The module named TX Host Emulator (TX_Host_Em) generates trigger commands and data packets for the FF_TX interface, and produces two text files listing the generated triggers and packets (TX_TRG_file and TX_Pkt_file); the RX Host Emulator (RX_Host_Em) reads TRGs and data packets outputted by FF_RX and writes two text files listing the received triggers and packets (RX_TRG_file and RX_Pkt_file); both these modules are clocked by the 40 MHz clock (clk40) and their operation is synchronous with this clock. TX_Host_Em can generate triggers and data packets using an internal random generator or reading an input text file (Hit_file), originated from physics simulations, that describes particle hits on a sensor. At the end of the test, the TX files and the RX files are compared to verify the correct operation of the link. Transmission errors corrupting bits on the DAT line and the CLK line are simulated by a Bit-flip insertion block: bit-flips on the DAT line have been inserted to test the THS and the Frame Descriptor encoding; bit-flips on the CLK line have been inserted to test the operation of the synchronization mechanism (see sections 2.2.2 and 4.3.3). The tests with TRG and Pkt files instead were carried out without bit-flip insertion, to check the basic functionalities of data transmission of the link A test controller, not shown in the figure, generates the 40 MHz and the transmission clock (clk40 and clk_tx), the reset signal rst_L, the start command for TX_Host_Em and RX_Host_Em and the clk_count signal, that counts the clk40 cycles and whose value is included as time stamp in the trigger and packet files.

4.4.1 TX Host Emulator

The TX Host Emulator generates trigger commands and data packets to be used as test vectors for the link, and creates the TX_TRG_file and TX_Pkt_file to be compared with the analogous files that are produced by the RX Host Emulator on the receiver side of the link. TX_Host_Em can operate in two modes: random mode and hit file mode. In the first, triggers and data packets are internally generated in a random way, while in the second mode they are generated from the information contained in an input text file (hit file) that describes the hits occurring in a sensor module (supposed to be connected to the TX host circuit in this case).

TRG and data packet generation in random mode.

In random mode, after the start of test (start input), at each clk40 cycle TX_Host_Em generates a random real number TRG_rand with uniform probability distribution in the range [0, 1), and if the generated value is below a settable threshold a TRG command is outputted (i.e. a one-clk40-cycle long pulse on the TRG output is produced). After a TRG command, the TRG generator is suspended for 3 clk40 cycles (to respect the constraint about minimum interval between consecutive triggers) and then it is resumed.

In random mode, at each clk40 cycle TX_Host_Em generates also a random real number DV_rand with uniform probability distribution in the range [0, 1), and if the generated value is below a settable threshold the Data_Valid (DV) output is pulled up to start the delivery of a data packet tot FF_TX. At this moment, other random number are generated to decide with the same modality the value of the Label_On and Data_Type bits for the current packet, and the generated values are placed onto outputs LO and DT in the same clock cycle as the rising of DV; also, the length of the data packet (i.e. the number of its words) is chosen at the same time as a random integer uniformly distributed between 1 and a settable maximum length (for example 20, to test also the packet fragmentation functionality). From now on, a new data word (randomly generated as well) is put onto the word_out output at each clock cycle as long as its reading by FF_TX is confirmed by the Get_Data signal being asserted. If instead GD is low at some clock cycle, the word outputted by TX_Host_Em in that cycle is kept unchanged until GD goes high again, meaning FF_TX has correctly read it. Generation of words goes on until the end of the packet (i.e. when the previously generated packet length is reached): at this time, the DV output is pulled down and the packet generation algorithm is started again at the next clock cycle.

TRG and data packet generation in hit file mode.

In this operating mode, TX_Host_Em reads a text file that lists, for each value of a bunch crossing counter (time index), the coordinates of hits occurred in a sensor module and the amplitude of each hit. In particular, the CMS Pixel case was considered and the file, provided by Read Out Chips (ROC) designers at Paul Scherrer Institut (PSI), was originated from physics simulations on a 4-cm layer module with 16 ROCs at the full LHC phase I upgrade luminosity: it is a text file with each line having the following format (the various fields are enclosed in square brackets) :

[Time index] – [L1T] – [1st Hit Data] [2nd Hit Data] [3rd Hit Data] ...[last Hit Data] ;

Time index is an integer that expresses the clk cycle number (bunch crossing number) at which the events described in the line occur; lines are ordered by increasing time index. Next, enclosed between two dashes "–", is the field L1T: a character that can be either "0" (meaning no trigger for the current event) or "T" (trigger present). Then hit data follow, separated with spaces one from another: each hit data block specifies the address of a single pixel and the amplitude of the hit it gets, and has the following format:

([ROC number] , [column address] , [row address] , [hit amplitude])

Finally, the line must be ended with a semicolon. So, an example of a line of the hit file could be:

23 – T – (3, 15, 142, 2) (5, 0, 63, 5);

meaning that at clk cycle n° 23 a L1T is present and two hits occur: one with amplitude 2 for pixel number 142, in the n° 15 Column of ROC number 3, and the other of amplitude 5 for pixel n° 63, in the n° 0 Column of ROC number 5.

In hit file mode, TX_Host_Em reads each line containing a trigger and emulates the behavior of a ROC or the data concentrator inside the module (called TBM, Token Bit Manager): when the clock counter of the test bench reaches the value of the time index of the line, it creates a packet with the data describing the hits listed in that line that refer to a specific ROC (in ROC mode) or for all the ROCs in the module (in TBM mode), following the format foreseen for the CMS Pixel digital readout in the phase I upgrade:

ROC number:	4 bits
column address:	5 bits
row address:	8 bits
hit amplitude:	8 bits

therefore, 25 bits for each hit. In TBM mode, a 16-bit time stamp (containing the current time index value) is also added as the label for the packet. TX_Host_Em then divides this packet into 16-bit words, stores it into an internal buffer and delivers it word-by-word to FF_TX, according to the Data_Valid-Get_Data handshake.

The trigger commands read from the hit file are sent to the FF_TX interface as well, to test the simultaneous transmission of readout data and triggers with realistic rates.

Writing of TX_TRG_file and TX_Pkt_file.

Regardless of the generation method, trigger commands and data packets that are delivered to the FF_TX interface are also written into the TX_TRG_file and TX_Pkt_file.

For each TRG command produced, a line is added to the text file TX_TRG_file simply containing the time stamp of the transmitted TRG command, that is the value of the clk_count signal in the cycle when the TRG pulse was delivered to the FF_TX interface. At the end of the test, the TX_TRG_file will appear as the example in Fig. 4.77:

```
77
90
114
130
151
163
167
176
189
199
239
243
252
261
275
280
301
```

Fig. 4.77 – Example of TX_TRG_file (a portion is shown here).

When a packet delivery by TX_Host_Em begins, that is when DV goes high, a line is added to the text file TX_Pkt_file with the string “Packet #” followed by a progressive integer used to identify the packet and the value of the LO and DT bits for that packet. Then, into the following lines the words belonging to the packet are written, one word per line preceded by its time stamp, i.e. the value of the clk_count signal in the cycle when that word was read by the FF_TX interface. At the end of the test, the TX_Pkt_file will appear as the example in Fig. 4.78:

```
Packet # 41, LO = 1, DT = 1
1298 60C8
1299 C7C2
1300 D415
1301 EEA4
1302 8101
1303 9426

Packet # 42, LO = 0, DT = 1
1310 5589
1311 5F00
1312 1231
1313 7517
1314 4F9C
1315 0DB5
1316 2498
1317 5CE6
1318 2624
1319 E2AC
1320 739C
1321 279F
1322 C919
1323 3DB1
1324 A7C6
1325 CACB
1326 8A8E
1327 6054
1328 269B
1329 EFB4

Packet # 43, LO = 1, DT = 1
1367 7D2C
1368 7569
1369 65AD
1370 1AAD
```

Fig. 4.78 – Example of TX_Pkt_file (a portion is shown here).

4.4.2 RX Host Emulator

The RX Host Emulator reads trigger commands and data packets outputted by FF_RX and writes them into two text files to be compared with TX_TRG_file and TX_Pkt_file.

Writing of RX_TRG_file.

After the start of test (start input), at each clk40 cycle RX_Host_Em checks the TRG input, connected to the trigger command output of FF_RX: if a TRG command is present, RX_Host_Em writes a line into the text file RX_TRG_file simply containing the time stamp of the received TRG, that is the value of the clk_count signal in the cycle when the TRG pulse was delivered by the FF_RX interface. Therefore at the end of the test, if the link operation was correct, the

RX_TRG_file will appear as the TX_TRG_file, but with each time stamp incremented of an amount equal to the TRG latency in the link, that is six clk40 cycles; for example, the RX_TRG_file portion relative to the TX_TRG_file portion of Fig. 4.77 is the one shown in Fig. 4.79:

```
83
96
120
136
157
169
173
182
195
205
245
249
258
267
281
286
307
```

Fig. 4.79 – Portion of RX_TRG_file relative to the portion of TX_TRG_file shown in Fig. 4.77.

Writing of TX_Pkt_file.

After the start of test (start input), at each clk40 cycle RX_Host_Em checks the DV input, connected to the Data_Valid output of FF_RX: if a DV is asserted, meaning that a packet delivery is started by FF_RX, RX_Host_Em writes a line into the text file RX_Pkt_file with the string "Packet #" followed by a progressive integer used to identify the packet and the value of the LO and DT bits for that packet, read from the Label_On and Data_Type output of FF_RX. Then, into the following lines the words belonging to the packet are written, one word per line preceded by its time stamp, i.e. the value of the clk_count signal in the cycle when that word was delivered by the FF_RX interface. When DV goes low, meaning the end of the delivery of a frame by FF_RX, if the packet is not finished (i.e. the Last_Frame output of FF_RX was unasserted for the frame that has just ended) RX_Host_Em writes a blank line in the file before writing the words of the next frame of the packet: this is done to visually represent the fragmentation of long packets. At the end of the test, if the link operation was correct, the RX_Pkt_file will appear as the TX_Pkt_file, but with different time stamps and blank lines inside the fragmented packets; for example, the RX_Pkt_file portion relative to the TX_Pkt_file portion of Fig. 4.78 will be the one shown in Fig. 4.80.

4.4.3 Comparison of files

At the end of the test, a procedure is started that compares the TX_TRG_file with the RX_TRG_file and the TX_Pkt_file with the RX_Pkt_file. For the TRG files, it is verified that for each time stamp with value ts_i in TX_TRG_file a correspondent time stamp exists in RX_TRG_file with value $ts_i + 6$ (i.e. the TRG latency), otherwise an error is reported. For the Pkt files, it is checked that for each data packet in TX_Pkt_file a correspondent packet in RX_Pkt_file exists, possibly fragmented into more frames but with the same value of the LO and DT bits and the same words, otherwise an error is reported.

```
Packet # 41, LO = 1, DT = 1
1422 60C8
1423 C7C2
1424 D415
1425 EEA4
1426 8101
1427 9426

Packet # 42, LO = 0, DT = 1
1468 5589
1469 5F00
1470 1231
1471 7517
1472 4F9C
1473 0DB5
1474 2498
1475 5CE6
1476 2624
1477 E2AC
1478 739C
1479 279F
1480 C919
1481 3DB1
1482 A7C6
1483 CACB

1486 8A8E
1487 6054
1488 269B
1489 EFB4

Packet # 43, LO = 1, DT = 1
1531 7D2C
1532 7569
1533 65AD
1534 1AAD
```

Fig. 4.80 – Portion of RX_Pkt_file relative to the portion of TX_Pkt_file shown in Fig. 4.78.

4.4.4 Test results

Test runs were performed in hit file mode to check the link functionality with realistic data and trigger rates. Test runs with an input hit file of 100000 clock cycles (bunch crossings), relative to a 4-cm layer module with 16 ROCs at the full LHC phase I upgrade luminosity, were carried out for different link speeds (160, 320 and 640 Mbit/s) and in both ROC mode and TBM mode, to test the application of the FF-LYNX interfaces to the ROC-TBM link and to the module-DAQ system link. The average trigger frequency in this file is about 125 kHz, and the average data rate, considering the 25-bit-per-hit format described in previous section, is about 5.7 Mbit/s from each ROC and about 34.5 Mbit/s from the entire module.

The result of all the test runs was positive, i.e. the TRG and Pkt file comparison was successful for all the different link speeds and for both ROC mode and TBM mode: obviously, the occupancy of the TX buffer, i.e. the maximum number of occupied locations, that dictates the depth that the buffer itself must have not to have data overflows, is different for the various cases: for example, a 160 Mbit/s link can be applied to the module-DAQ system stretch, but a large TX buffer is required in this case to avoid overflows. The occupancy of the TX buffer measured in the hit file mode simulations is reported in Table 4.35: in these simulations the depth of the TX_Host_Em buffer (representing the buffering capability of the ROC/TBM) was set to a value such that it can contain the largest hit data packet generated in a single bunch crossing.

Mode	Average data rate	TX Buffer occupancy		
		link speed =160 Mbit/s	link speed = 320 Mbit/s	link speed = 640 Mbit/s
ROC	5.7 Mbit/s	24	22	20
TBM	34.5 Mbit/s	108	57	31

Table 4.35 – TX Buffer occupancy for different link speed and application (ROC or TBM).

These results show that, even if the average data rate is well under the link capability, because of data rate peaks (in correspondence with triggered events) it is preferable to use high speed links (320 or 640 Mbit/s) in the module-DAQ system stretch in order to limit the size of buffers that are needed in the transmitter interfaces to avoid overflows.

Many test runs were also carried out in random mode to verify the functionality of the interfaces with very high trigger and data rates: by appropriate setting of random generators inside the TX_Host Emulator, trigger rates up to about 3,3 MHz (one TRG every 12 clk40 cycles on average) and (average) data rates up to about 230 Mbit/s were simulated in several test runs at different link speed, each with a duration of about 100000 clk40 cycles: the test result was positive in all the runs, thus confirming the correct functionality of the interfaces VHDL model at least in absence of errors on the line.

5 FPGA prototyping

As final result for the first year of activity in the FF-LYNX project, the realization of an FPGA-based emulator for the FF-LYNX transmitter-receiver system is foreseen. This emulator will consist in an FPGA development board on which the VHDL models of the FF_TX and FF_RX interfaces are synthesized, together with a surrounding test system that drives the interfaces with stimuli and reads the results; the development board provides all necessary resources, such as memories, interfaces and connectors to create a physical link between FF_TX and FF_RX and interfaces for communication with a PC that controls the emulator operation, provides the test vectors and analyzes the test results. The architecture foreseen for the FF-LYNX Emulator is briefly outlined in section 5.1, while the development board chosen to implement the emulator is described in section 5.2. In view to design and realize the emulator, as the final step of this thesis work the synthesis of the FF-LYNX interface VHDL model was carried out on the Stratix III FPGA included in the chosen development board, to have a first complexity estimate of the designed interfaces. Synthesis results are reported in section 5.3.

5.1 FF-LYNX Emulator

The conceptual architecture that is foreseen for the FF-LYNX Emulator is depicted in Fig. 5.1.

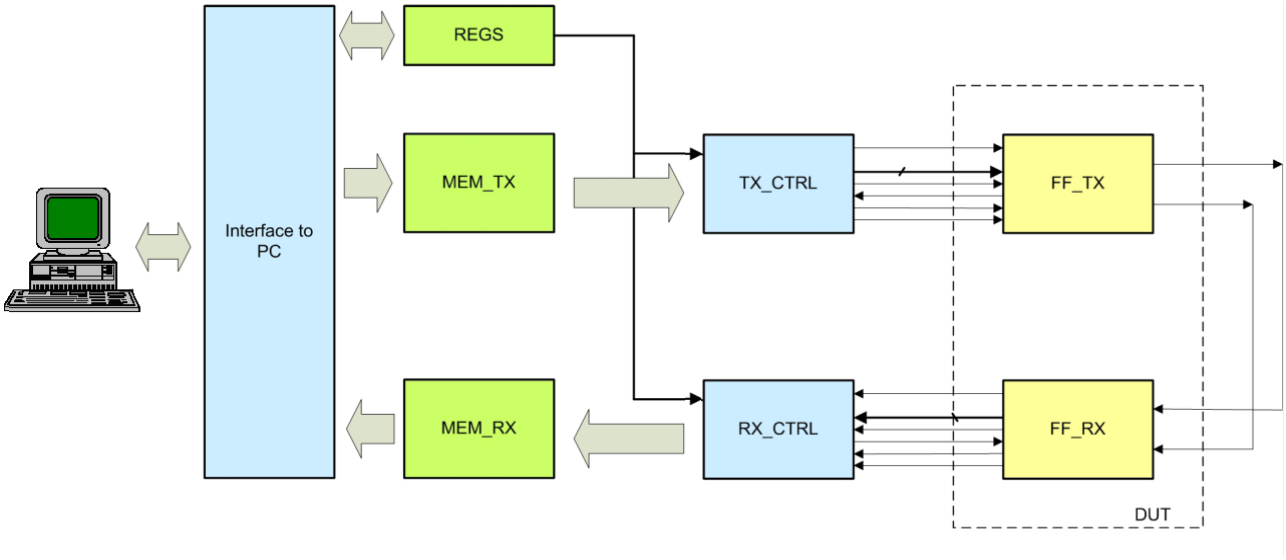


Fig. 5.1 – Conceptual architecture of the FF-LYNX Emulator.

The Device Under Test (DUT) is the FF-TX-FF_RX pair. The test input signals are provided to the FF_TX interface by a TX Controller block (TX_CTRL) that reads test vectors (trigger commands and data packets) stored in a transmitter memory (MEM_TX). An RX Controller (RX_CTRL) reads triggers and data received by the FF_RX interface and writes them into a receiver memory (MEM_RX). Test vectors are generated by a software application running on a Personal Computer, and then written into MEM_TX through an interface module that deals with data exchange between the PC and the Emulator system using a communication port of the board (Ethernet). The same software application reads DUT output data from MEM_RX and elaborates them, performing comparison with test vectors and calculating predefined performance figures; also, it controls the test procedure by setting the content of the emulator control registers (REGS) that drive the operation of TX_CTRL and RX_CTRL. The client/server architecture of the high-level test controller used for the protocol validation phase (section 2.5) reveals particularly useful

here: the server side can be now replaced with the FPGA emulator without changing the client, since the TCP/IP socket interface remains the same.

In a first development phase, all the Emulator system will be implemented into the FPGA; successively, on-board RAM modules will be used as memory blocks MEM_TX and MEM_RX to provide longer test capability, and the link between FF_TX and FF_RX will be realized with an external cable between connectors of the development board to test the interfaces operation with a physical link technology (LVDS in particular).

5.2 Development board

The development board chosen for the FF-LYNX Emulator is a Stratix III EP3SL150 from Altera [31]. It is based on a EP3SL150F1152C2 device, that is a 1,152-pin Altera Stratix III FPGA in a ball-grid array (BGA) package featuring:

- 142,000 logic elements (LEs)
- 5,499 Kbits of memory
- Eight phase locked loops (PLLs)
- 16 global clock networks
- 736 user I/Os
- 1.1 V core power

The board also includes a variety of memory modules, displays, buttons and interfaces as shown in Fig. 5.2.

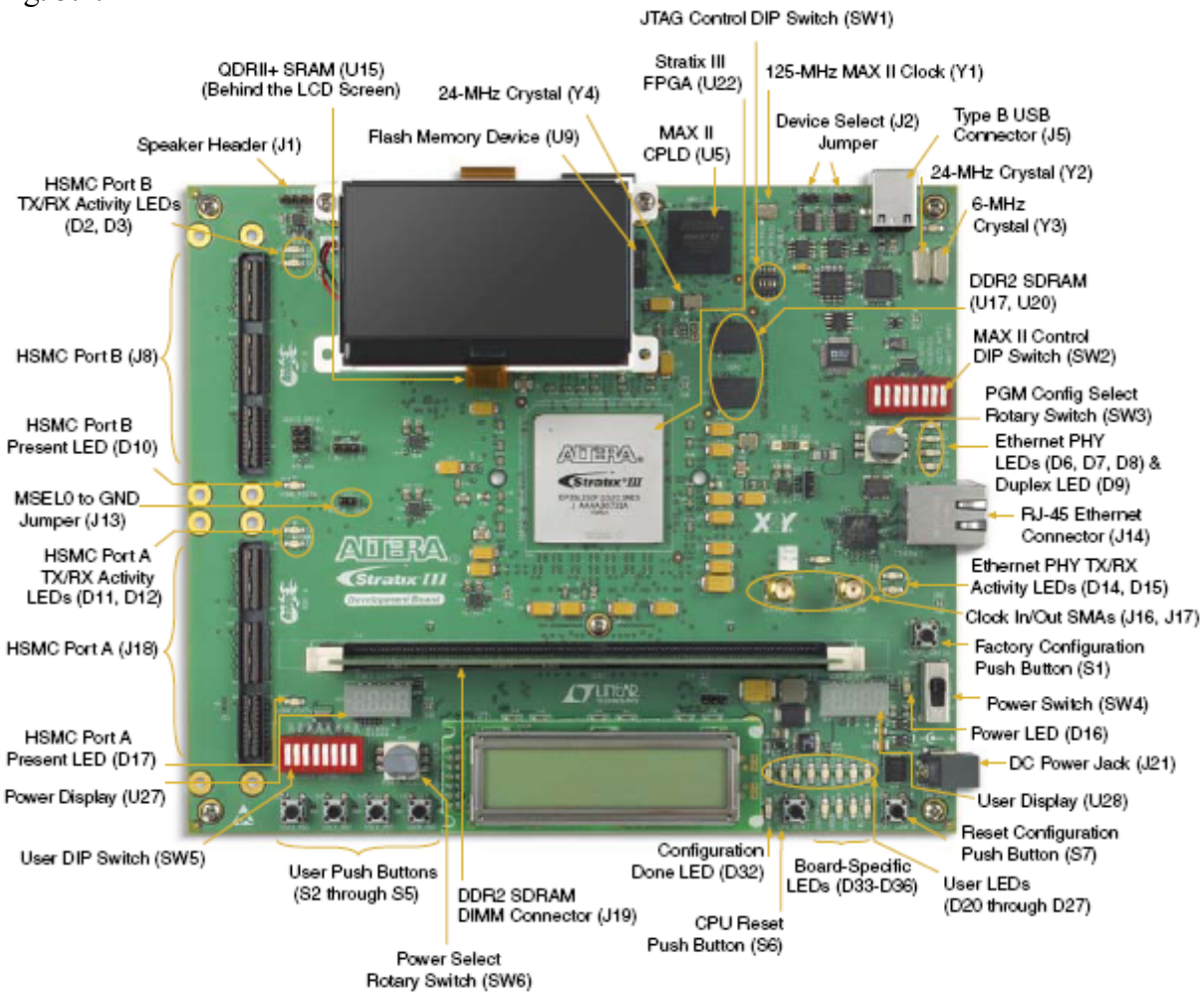


Fig. 5.2 – The Altera Stratix III EP3SL150 development board.

In particular, the following components are useful for the implementation of the FF-LYNX Emulator:

- two 16-MByte DDR2 SDRAM devices, to be used as TX and RX test memories;
- Ethernet interface for connection to PC;
- two HSMC (High Speed Mezzanine Card) interfaces, supporting both single-ended and differential signaling: these two ports have a direct connection with I/O pins of the FPGA and a loopback HSMC daughter card can be used to establish a loop link in different technologies, such as LVDS or LVCMOS.

5.3 FPGA synthesis of FF-LYNX interfaces

Using the Altera Quartus II design software, synthesis has been carried out of the interfaces VHDL model on the Stratix III FPGA included in the development board.

To his end, a top-level schematic was created including the FF_TX and FF_RX entities together with one of the eight Phase Locked Loops (PLL) available on the Stratix III, to generate the reference clock at frequency F (F = 40 MHz nominally) and the transmission clock for both the receiver and transmitter interfaces from an FPGA input clock. Then, using the Quartus II synthesis tool several synthesis attempts were made for the three modeled architectures (4xF, 8xF and 16xF, nominally 160, 320 and 640 Mbit/s) to find the maximum value of F for which the modeled architecture can be mapped onto the FPGA respecting the internal timing constraints, i.e. without violation of the hold and setup time conditions on any of the internal paths between registers: this analysis is carried out by the Timing Analyzer tool included in the Quartus II design processing flow. The synthesis results are reported in Table 5.1.

		clk40/clk_tx frequency	Logic utilization		
			Combinational ALUT	Registers	Block memory bits
4xF	Fnom	40/160 MHz	479 /113,600 (0.42%)	313 /113,600 (0.27%)	2,688 /5,630,976 (0.04%)
	Fmax	100/400 MHz	453 /113,600 (0.39%)	316 /113,600 (0.28%)	2,688 /5,630,976 (0.04%)
8xF	Fnom	40/320 MHz	508 /113,600 (0.45%)	353 /113,600 (0.31%)	2,688 /5,630,976 (0.04%)
	Fmax	50/400 MHz	503 /113,600 (0.44%)	360 /113,600 (0.32%)	2,688 /5,630,976 (0.04%)
16xF	Fnom	40/640 MHz	-	-	-
	Fmax	25/400 MHz	585 /113,600 (0.51%)	425 /113,600 (0.37%)	2,688 /5,630,976 (0.04%)

Table 5.1 – Synthesis results.

Logic utilization, that gives an estimate of the interfaces area occupation, is reported as the number of combinational ALUT, registers and block memory bits utilized by the synthesis tool to map the design into the Stratix III FPGA. Devices of this family have a core architecture that is based on Logic Array Blocks (LAB) surrounded by an interconnect matrix (Fig. 5.3). Each LAB consists of ten Adaptive Logic Modules (ALM), that are the basic building blocks of the FPGA logic, carry chains, shared arithmetic chains, LAB control signals, local interconnect, and register chain connection lines. The local interconnect transfers signals between ALMs in the same LAB. Paired with each LAB there is a Memory LAB (MLAB) that is a LAB with SRAM capability: each MLAB can be configured to work as a regular LAB or as a memory block.

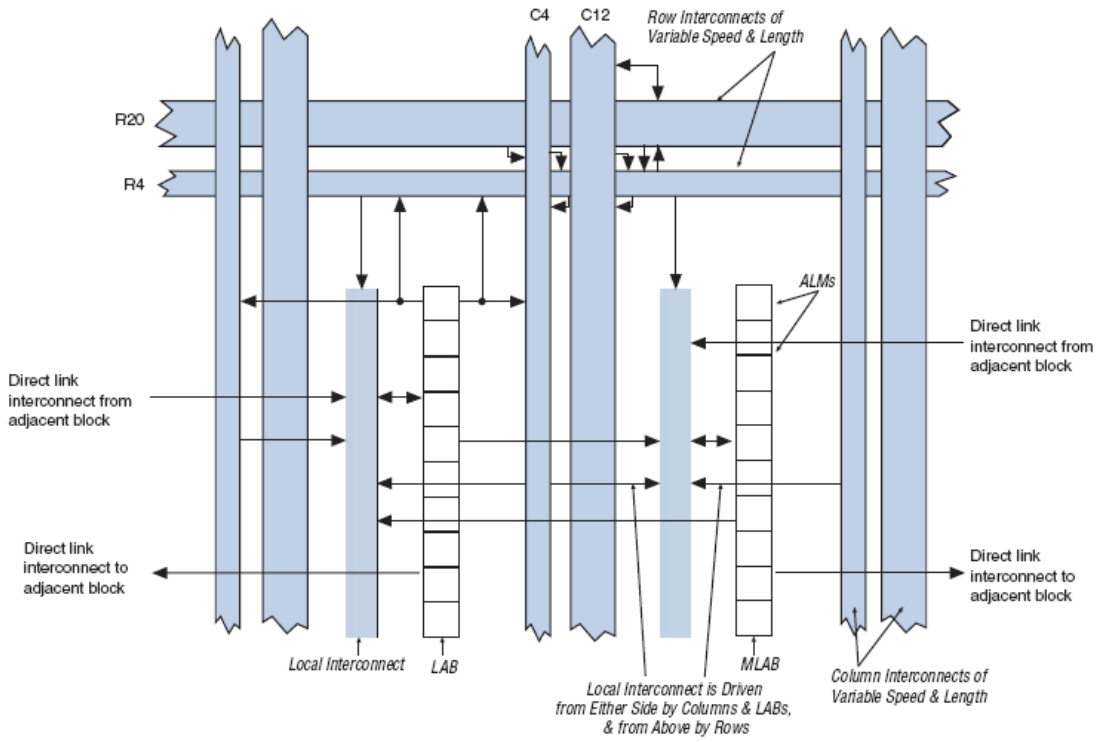


Fig. 5.3 – The Stratix III LAB structure.

The structure of an Adaptive Logic Module is shown in Fig. 5.4. The basic logic functionality is based on two Adaptive Look-Up Tables (ALUT) and two registers (flip-flops); in addition to these, each ALM contains two dedicated full adders, a carry chain, a shared arithmetic chain, and a register chain. ALUTs are combinational in normal LABs, while in MLABs they function as SRAM blocks.

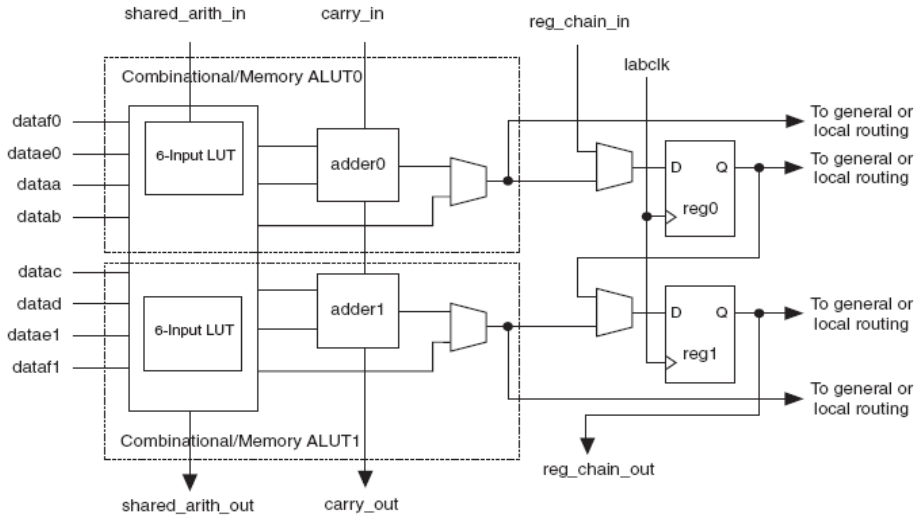


Fig. 5.4 – Internal architecture of the Stratix III ALM.

Therefore, the combinational ALUT utilization result in the synthesis report refers to the number of ALUT that are used for combinational functions in normal LABs, while the block memory bits result refers to the total number of memory bits used in MLABs, plus the ones used in other dedicated SRAM blocks that are included in the Stratix III architecture.

Conclusions and future developments

In High Energy Physics experiments, systems for data acquisition and distribution of Timing, Trigger and Control signals have similar architectures and similar requirements about data rate, trigger latency, robustness of critical data against transmission errors, radiation hardness and power dissipation and of hardware components and material budget. The same will be in the near future, for example in the scenario of the experiments for the Super LHC, that is the upgrade of the Large Hadron Collider at CERN to be carried out in two phases foreseen for 2013 and 2018. Therefore, the use of common solutions that can be reused in different applicative contexts can bring many advantages such as the reduction of costs, risks and time needed for the development of new experiments. In particular, a research and development activity appeared as useful in the field of electrical links that are employed for data transmission to and from Front End circuits inside the detectors to move power-consuming optical converters away from the interaction point: indeed, analog readout schemes currently used in some contexts (e.g. the CMS pixel readout system) need to be replaced with digital and serial links in order to improve the noise performance, and standard protocols used in telecommunications, consumer electronics and automotive industry don't fulfill satisfactorily the requirements about radiation hardness and trigger distribution imposed by these applications.

These consideration constituted the motivations for the FF-LYNX project, that was started in January 2009 by a collaboration between INFN-PI (Italian National Institute for Nuclear Physics, division of Pisa) and the Department of Information Engineering (DII_IET) of the University of Pisa. In a three-year foreseen activity, the project aims at the definition of a serial communication protocol for the integrated distribution of TTC signals and Data Acquisition that could meet typical requirements of HEP applications and be adaptable to different applicative scenarios inside this field, and its implementation in radiation-tolerant interfaces designed and developed in a standard CMOS technology.

In the first nine months of activity, a first version (v.1) of the FF-LYNX protocol has been defined with features that address some of the basic requirements of HEP applications, such as constant trigger latency and robustness of critical data against transmission errors: this version of the protocol was then tested in a SystemC simulation environment to validate the choices that have been made and implemented in a transmitter and a receiver interface, for which a VHDL model was created and simulated. This work of thesis, in particular, concerned the protocol definition and the VHDL modeling of the transmitter and receiver interfaces in three architecture, distinguished for the value of the transmission speed: 4xF, 8xF and 16xF, where F is the reference frequency of the applicative context, i.e. the master clock frequency of the experiment; in the LHC scenario F = 40 MHz and the featured transmission speed values are 160, 320 and 640 Mbit/s.

The first version of the FF-LYNX protocol was defined at the data-link layer of the ISO/OSI model, dealing with the issues of stream multiplexing and synchronization, framing and error control of critical data. To minimize the number of required wires a serial transmission was chosen, and a separate clock transmission scheme was preferred to simplify the receiver circuits: the transmission is therefore carried out on two wires:, a DAT line that delivers serial data and a CLK line that carries the transmission clock with frequency 160, 320 or 640 MHz (referring to the LHC scenario, where the reference clock frequency is 40 MHz); the reference clock to be delivered to the receiver host circuit is reconstructed by a synchronizer circuit inside the receiver interface.

To guarantee the fundamental requisites of error protection of the trigger command and constant latency from its transmission to its reception, a dedicated time-division channel, called THS channel, inside the DAT serial stream was reserved to transmission of Triggers, frame Headers and Synchronization patterns, while the other channel (FRM channel) carries data frames.

The THS channel comprises two bits in each 40 MHz clock cycle and carries THS sequences of six bits each. Two of these sequences, denoted as TRG and HDR, are used to encode triggers and frame headers respectively: TRG transmission is always scheduled with highest priority in order to

guarantee fixed latency to the trigger command; a third sequence (SYN) to be used as a synchronization pattern, was initially considered, but afterwards a balanced version of the THS encoding was chosen comprising just the TRG and HDR sequences and a NOP pattern that is continuously sent onto the THS channel when no TRG or HDR is being transmitted. Synchronization of the receiver on the THS channel, i.e the individuation of the 2 bits of this channel among the 4, 8 or 16 bits (depending on the transmission speed) that arrive on the DAT line in each 40 MHz clock cycle, is obtained by detecting and counting THS sequences inside the received DAT stream. A custom 6-bit encoding for the TRG, HDR and NOP sequences was adopted to allow a correct recognition of each pattern even in presence of single-bit transmission errors.

The FRM channel is used to transmit user data, encapsulated in data frames. The basic FF-LYNX frame has a simple structure that comprises, besides the HDR sequence that is transmitted into the THS channel to mark the frame beginning, a Frame Descriptor field, an optional "label" and the payload, structured in 16-bit words; to have maximum flexibility towards data formats, no other structure is imposed for the user data carried in the payload. The optional, 16-bit label field is intended to contain information that characterize the data carried by the frame, to be used for sorting or aggregating data packets: the typical use should be that of carrying a timing information (time stamp) relative to the particle hit data produced by the sensors, for event building purposes. The Frame Descriptor specifies some important characteristics of the frame itself and the payload: the frame length, expressed in number of 16-bit words, the presence of the label, the data type and a "last frame" bit that is used in case of fragmentation of a long data packet into more consecutive frames. Since it carries the frame length information, that is crucial for the correct reception of the frame, the Frame Descriptor is protected against transmission errors by means of a H(12,7) code, that is a shortened and extended Hamming code capable of detecting and correcting single-bit errors and detecting double-bit errors.

Besides basic, variable length frames intended for generic, low priority data, in future versions of the protocol a special kind of frame is foreseen to carry trigger data, i.e. information from a subset of FE circuits that is used for trigger. These trigger data frame have a fixed length and are marked with a TRG sequence as their header to guarantee a fundamental requirement of trigger data, that is fixed and small latency.

The architecture of a transmitter (FF_TX) and a receiver (FF_RX) interface has been defined to implement the FF-LYNX protocol v.1. FF_TX is the interface that allows the host system to send trigger commands and data towards a receiver system according to the FF-LYNX protocol. It comprises the following functional blocks: TX Buffer, that is the FIFO memory that stores the data packets awaiting to be transmitted; Frame Builder, that controls the assembly of frames for data transmission; THS Scheduler, that manages the scheduling of triggers and frame headers transmission; Serializer, that generates the DAT output serial stream with the THS channel and the FRM channel. FF_RX takes as inputs the DAT and CLK transmission lines, carrying the serial data stream and the transmission clock generated by FF_TX, and delivers to the receiver host the transmitted data packets, trigger commands and a reconstructed 40 MHz clock. The functional blocks of FF_RX are: Deserializer, that is the block that converts back the DAT stream into parallel form providing the received data words; THS Detector, that reveals the arrival of TRG, HDR and NOP sequences in the THS channel; Synchronizer, that gets the sync with the THS channel of the received DAT stream thus generating the 40 MHz reconstructed receiver clock; Frame Analyser, that controls the reception of data frames, their storing into the RX Buffer and the delivery of stored data to the receiver host; RX Buffer, that is the FIFO memory that stores the received data packets awaiting to be delivered to the receiver host.

For the design phase of the FF-LYNX interfaces, a model of FF-TX and FF_RX was built using the VHDL hardware description language, employing the development environment Active-HDL by Aldec to write the model and to perform functional simulations on it. The function of each block in the FF_TX and FF_RX architecture was partitioned in simpler sub-blocks; then each sub-block

was described and its functionality was verified in a specific test bench. Afterwards, each block of the architecture was built connecting its constituting sub-blocks, and functional simulations on the whole block were carried out. Finally, the complete transmitter-receiver system was built and the overall functionality was verified in a global test bench comprising also external VHDL modules for the generation of test vectors (data and trigger commands to be transmitted) and for the reading of the system outputs (received data and trigger commands). Of these modules, the one on the transmitter side, named TX Host Emulator, can generate triggers and data packets to be delivered to FF-TX by using an internal random generator or reading an input text file, originated from physics simulations, that describes particle hits on a sensor; it also produces two text files listing the generated triggers and data packets. On the receiver side, the RX Host Emulator module reads triggers and data packets outputted by FF_RX and writes two text files listing the received triggers and packets. At the end of the test, the TX files and the RX files are compared to verify the correct operation of the link.

Test runs were performed with an input hit file, originated by physics simulations on the CMS Pixel system and provided by Read Out Chips (ROC) designers at Paul Scherrer Institut (PSI), to check the link functionality with realistic data and trigger rates. Test runs with an input hit file of 100000 clock cycles (bunch crossings), relative to a 4-cm layer module with 16 ROCs at the full LHC phase I upgrade luminosity, were carried out for different link speeds (160, 320 and 640 Mbit/s); the average trigger frequency in this file is about 125 kHz, and the average data rate is about 5.7 Mbit/s from each ROC and about 34.5 Mbit/s from the entire module. Test runs were also carried out with random generation of test vectors to verify the functionality of the interfaces with very high trigger and data rates: trigger rates up to about 3,3 MHz and average data rates up to about 230 Mbit/s were simulated in several test runs at different link speed, each with a duration of about 100000 40 MHz clock cycles. The result of all the test runs was positive, thus confirming the correct functionality of the interfaces VHDL model.

As the final result for the first year of activity in the FF-LYNX project, the realization of an FPGA based emulator for the FF-LYNX transmitter-receiver system is foreseen.. This emulator will consist in an FPGA development board on which the VHDL models of the FF_TX and FF_RX interfaces are synthesized, together with a surrounding test system that drives the interfaces with stimuli and reads the results; the test vectors generation, the test results analysis and the control of the test system operation will be carried out by a software application running on a personal computer. The conceptual architecture for the FF-LYNX Emulator has been defined and the Altera Stratix III EP3SL150 development board has been chosen to realize the emulator: this board offers all the resources needed for our system (such as SDRAM modules, Ethernet interface for connection to PC, and various types of external connectors to test the protocol with different physical technologies, LVDS in particular) and for possible future developments.

As the final step in this thesis work, synthesis has been carried out of the interfaces VHDL model (for the 160, 320 and 640 Mbit/s architectures) on the Stratix III FPGA using the Altera Quartus II design software: the maximum working frequency (referring to the transmission clock) resulted to be 400 MHz for all the three architectures.

As for next developments of the FF-LYNX project, the first activity to be carried out is the completion of the design and the implementation of the FPGA emulator, and a thorough testing activity on it to evaluate the performance aspects that have been defined and explored with the SystemC simulator.

Next versions of the FF-LYNX protocol will be studied and defined to include new features that appear as useful to improve the link performance: interleaving, to make the adopted error control techniques effective also with burst errors; error protection of the payload and the label of data frames with Cyclic Redundancy Check (CRC); a custom block encoding (similar to 8b/10b) to be applied to the FRM channel only (thus preserving the THS encoding) and that is able to guarantee DC balancing of the stream (thus making the FF-LYNX protocol suitable for applications with AC-coupled receivers and high-pass transmission channels) and frequent transitions, so that a Clock and

Data Recovery circuit scheme can work properly and hence a single-wire (clock + data) transmission scheme can be adopted; trigger data frames handling, to allow transmission of trigger data from Front End circuits with small and fixed latency. The definition of a second version (v.2) of the FF-LYNX protocol, including fixed length frames for trigger data transmission, and its implementation in the SystemC and VHDL models is foreseen for the last three months of 2009; in the first months of 2010 the FF-LYNX protocol v.2 will be upgraded to version 3 by including the single-wire feature with the "8b/10b-like" encoding. For each new protocol version the relative interfaces will be designed and tested in a VHDL modeling phase and a FPGA prototyping phase, upgrading the FF-LYNX Emulator that has been realized for the version 1.

For the first half of 2010 the design and realization of a test chip is also foreseen to complete the verification and characterization of the FF-LYNX interfaces implementing the protocol version 2. The chip will contain the transmitter and receiver interfaces (in all the different versions in terms of transmission speed) and will be interfaced with the test bench architecture that has been already synthesized on the FPGA emulator; will be realized in a commercial CMOS technology (below 180 nm to profit of intrinsic robustness of these technologies to Total Ionization Dose effects) and will be tested in order to characterize the interfaces about electrical and thermal properties; also radiation tests will be performed to evaluate tolerance to Total Ionization Dose and Single Event Effects.

The FF-LYNX project and the work described in this thesis was presented at the following international workshops and conferences:

- 2nd Common ATLAS CMS Electronics Workshop for SLHC (ACES '09), Geneva, Switzerland, 3-4 March 2009: oral and poster presentation [32].
- XI Pisa Meeting on Advanced Detectors: Frontier Detectors for Frontier Physics, La Biodola, Isola d'Elba, Italy, 25-30 May 2009: poster presentation and paper on Nuclear Instrumentation and Methods in Physics Research A [33].
- IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS '09), 21-23 September 2009, Rende, Italy: poster presentation and paper on conference proceedings [34].

Bibliography

[1] O. Bruning, “LHC challenges and upgrade options”, J. Phys.: Conf. Ser. 110, 2008.

[2] O. Bruning et al., “LHC Design Report 2004 Volume 1”, CERN-2004-003.

[3] E.Bartz, The 0.25μm Token Bit Manager Chip for the CMS Pixel Readout, Proceedings of the 11th Workshop on Electronics for LHC and Future Experiments, Heidelberg, Germany, 12-16 September 2005.

[4] H. C. Kästli et al., “Design and Performance of the CMS Pixel Detector Readout Chip”, arXiv:physics/0511166 v2 24 Feb 2006.

[5] K. Gabathuler, “PSI46 Pixel Chip - External Specification”, 8 Dec 2005.

[6] H. C. Kästli, “CMS Pixel Upgrade”, presentation at 2nd ACES workshop, CERN, 2009.

[7] G. Hall, "Super LHC: Detector and Electronics Upgrade," presentation at RD50 workshop, May 2004.

[8] G. Papotti, A. Marchioro, P. Moreira, “An Error-Correcting Line Code for a HEP Rad-Hard Multi-GigaBit Optical Link”, Proc. 12th Workshop on Electronics for LHC and Future Experiments, Valencia, Spain, 25-29 September 2006.

[9] AA. VV. “CMS Tracker Project, Technical Design Report”, CERN/LHCC 98-6.

[10] R. Horisberger, “Progress on the CMS Pixel front-end system”, presentation at 2nd ACES workshop, CERN, 2009.

[11] The CMS Collaboration, S Chatrchyan et al., “The CMS experiment at the CERN LHC”, Journal of Instrumentation, 2007.

[12] The ATLAS Collaboration, “ATLAS Inner Detector Technical Design Report”, Volume 1 and 2, ATLAS TDR 4, CERN/LHCC/97-16.

[13] R.Beccherle et al., “MCC: the Module Controller Chip for the ATLAS Pixel Detector”, Nucl. Instr. and Meth. in Phys. Res. A 492 (2002) 117-133.

[14] T. Virdee et al., “CMS High Level Trigger”, CERN-LHCC-2007-021, Geneva, 2007.

[15] J. Troska et al., "Optical readout and control systems for the CMS tracker," IEEE Trans. Nucl. Sci, vol. 50, pp. 1067-1072, 2003.

[16] G. Anelli et al., “Radiation tolerant VLSI circuits in standard deep submicron CMOS technologies for the LHC experiments: practical design aspects”, IEEE Transactions on Nuclear Science, vol.46, no.6, December 1999, pp.1690-1696.

[17] F. Faccio et al., “Radiation tolerance of commercial 130nm CMOS technologies for High Energy Physics Experiments”, FEE 2006 Perugia.

[18] P. Moreira, A. Marchioro, K. Kloukinas, “The GBT: a proposed architecture for multi-Gb/s data transmission in high energy physics”, Topical Workshop on Electronics for Particle Physics, Prague, Czech Republic, 03 - 07 Sept 2007.

[19] M. Suess, “From SpaceWire to SpaceFibre”, 2006 MAPLD International Conference, Washington, USA, Sept 2005.

[20] T. Grötker et al., “System Design with SystemC”, Springer, 2002.

[21] J. F. Kurose, K. W. Ross, “Computer Networking”, Addison Wesley, 2000.

[22] F. Halsall, “Computer Networking and the Internet”, Addison Wesley, 2005.

[23] W. C. Huffman, V. Pless, “Fundamentals of Error-Correcting Codes”, Cambridge University Press, 2003.

[24] A. X. Widmer, P. A. Franaszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code", IBM Journal of Research and Development 27 (5), 1983.

[25] P. Sweeney, “Error Control Coding. From Theory to Practice”, John Wiley & Sons, 2002.

[26] A. Papoulis, “Probability, random variables, and stochastic processes”, 3rd ed., McGraw-Hill, 1991.

[27] http://www.aldec.com/ (Aldec website).

[28] D. L. Perry, “VHDL Programming by example”, McGraw-Hill, 2002.

[29] “IEEE Standard VHDL Language Reference Manual”, IEEE, 2002.

[30] B. Cohen, “VHDL: Coding Styles and Methodologies”, Kluwer, 1995.

[31] http://www.altera.com/ (Altera website).

[32] G. Bianchi, R. Castaldi, L. Fanucci, G. Magazzù, S. Saponara, C. Tongiani, P. G. Verdini, “FF-LYNX: Fast and Flexible protocols and interfaces for data transmission and distribution of timing, trigger and control signals”, poster presented at 2nd Common ATLAS CMS Electronics Workshop for SLHC (ACES ‘09), Geneva, Switzerland, 3-4 March 2009.

[33] G. Bianchi, R. Castaldi, L. Fanucci, G. Magazzù, S. Saponara, C. Tongiani, P. G. Verdini, “Fast and Flexible Electrical Links for Data Acquisition and Distribution of Timing, Trigger and Control Signals in Future High Energy Physics Experiments”, Nuclear Instrumentation and Methods in Physics Research section A, 2009.

[34] G. Bianchi, R. Castaldi, L. Fanucci, G. Magazzù, S. Saponara, C. Tongiani, P. G. Verdini, “The FF-LYNX Project: Design of Fast and Flexible Protocols and HW Interfaces for Data Acquisition and TTC Distribution in High Energy Physics Experiments”, poster presented at IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS ‘09), 21-23 September 2009, Rende, Italy.