LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Faculty of Technology
Electrical Engineering

Marko Kupiainen

READOUT ELECTRONICS FOR GAS ELECTRON MULTIPLIER DETECTORS

Examiners: Professor Jero Ahola
Professor Tuure Tuuva

Supervisor: Ph.D. Paul Aspell

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology
Electrical Engineering

Marko Kupiainen

Readout Electronics for Gas Electron Multiplier Detectors

Master's Thesis

2013

73 pages, 38 figures, and 5 tables.

Examiners: Professor Jero Ahola
Professor Tuure Tuuva

Keywords: Data communications electronics, Detector electronics, Particle physics.

The European Organization for Nuclear Research (CERN) operates the largest particle collider in the world. This particle collider is called the Large Hadron Collider (LHC) and it will undergo a maintenance break sometime in 2017 or 2018. During the break, the particle detectors, which operate around the particle collider, will be serviced and upgraded. Following the improvement in performance of the particle collider, the requirements for the detector electronics will be more demanding. In particular, the high amount of radiation during the operation of the particle collider sets requirements for the electronics that are uncommon in commercial electronics.

Electronics that are built to function in the challenging environment of the collider have been designed at CERN. In order to meet the future challenges of data transmission, a GigaBit Transceiver data transmission module and an E-Link data bus have been developed. The next generation of readout electronics is designed to benefit from these technologies. However, the current readout electronics chips are not compatible with these technologies. As a result, in addition to new Gas Electron Multiplier (GEM) detectors and other technology, a new compatible chip is developed to function within the GEMs for the Compact Muon Solenoid (CMS) project.

In this thesis, the objective was to study a data transmission interface that will be located on the readout chip between the E-Link bus and the control logic of the chip. The function of the module is to handle data transmission between the chip and the E-Link. In the study, a model of the interface was implemented with the Verilog hardware description language. This process was simulated by using chip design software by Cadence. State machines and operating principles with alternative possibilities for implementation are introduced in the E-Link interface design procedure. The functionality of the designed logic is demonstrated in simulation results, in which the implemented model is proven to be suitable for its task. Finally, suggestions that should be considered for improving the design have been presented.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknillinen tiedekunta
Sähkötekniikan koulutusohjelma

Marko Kupiainen

Lukuelektroniikka Gas Electron Multiplier -ilmaisimille

Diplomityö

2013

73 sivua, 38 kuvaa ja 5 taulukkoa.

Tarkastajat: Professori Jero Ahola
Professori Tuure Tuuva

Avainsanat: Tietoliikenne-elektroniikka, Ilmaisinelektroniikka, Hiukkasfysiikka.

Euroopan hiukkasfysiikan tutkimuskeskus (CERN) operoi maailman suurinta hiukkastörmäytintä, jonka on määrä läpikäydä huoltotauko vuosina 2017–2018. Tämän tauon aikana hiukkaskiihdyttimen varrella toimivia hiukkasilmaisimia huolletaan ja parannetaan. Hiukkaskiihdyttimen suorituskyvyn parannusten myötä ilmaisinelektroniikalle asetetut vaatimukset kasvavat. Erityisesti suuri säteilyn määrä hiukkaskiihdyttimen toiminnan aikana asettaa vaatimuksia, jotka ovat kaupallisessa elektroniikassa epätavallisia.

CERN:ssä on kehitetty hiukkaskiihdyttimen haastavaan ympäristöön sopivaa elektroniikkaa. Jotta voitaisiin vastata tulevaisuudessa kasvavan mittausdatan määrän tuomiin tiedonsiirtohaasteisiin, on CERN:ssä kehitetty GigaBit Tranceiver-tiedonsiirtomoduuli sekä E-Link-tiedonsiirtoväylä. Seuraavan sukupolven lukuelektroniikan on suunniteltu käyttävän näitä teknologioita, sillä tämän hetken lukuelektroniikkasirut eivät ole yhteensopivia näiden teknologioiden kanssa. Tästä syystä GEMs for the Compact Muon Solenoid (CMS) -projektissa kehitetään uusien Gas Electron Multiplier-ilmaisimien (GEM) ja muun tekniikan lisäksi uusi yhteensopiva siru.

Tässä työssä tavoitteena oli tutkia tiedonsiirtorajapintaa, joka tulee lukuelektroniikkasirun ohjauslogiikan ja E-Link-väylän välille. Moduulin tehtävänä on vastata tiedonsiirrosta sirun eri osien ja E-Linkin välillä. Tutkimuksessa toteutettiin Verilog-kuvauskielellä rajapintamoduuli, jota simuloitiin käyttämällä Cadencen sirusuunnitteluohjelmistoa. E-Link-rajapinnan suunnittelussa on esitelty sen osien tilakoneet ja logiikan toimintaperiaate. Näiden lisäksi on tuotu esille mahdollisia vaihtoehtoja toteutuksille. Suunnitellun logiikan toimintaa havainnollistetaan simulointituloksin, joissa on todettu toteutetun mallin soveltuvan tarkoitukseensa. Lopuksi rajapinnan toimintaan on esitetty parannusehdotuksia, jotka tulisi ottaa huomioon jatkokehityksessä.

Acknowledgments

The research conducted in this Master's Thesis was done at CERN under the employment of the Faculty of Technology at Lappeenranta University of Technology.

Firstly, I would like to thank Professor Tuure Tuuva for granting me this unique chance to work at CERN in Switzerland. It has been an experience to remember for the rest of my life. Secondly, I want to give my utmost thanks to my supervisor Ph.D. Paul Aspell. Working at CERN was a great experience because of his advice and encouragement.

Also, the whole microelectronics group at CERN deserves my thanks. I am grateful for the comfortable working environment and I enjoyed the way that people were easy to approach when I had something to ask. I would especially like to thank Ph.D. Sandro Bonacini and Ph.D. Kostas Kloukinas from the top floor for helping me get started with the design tools and Verilog. Additionally, I would like to say thank you to Ph.D. Xavi Llopart Cudie and Ph.D. Massimiliano De Gaspari for helping me with the later stage of design that did not make it into the thesis. There are also individuals who helped me out with smaller problems during the work. You know who you are.

Lastly, I want to thank my parents for all of their support over the years.

Helsinki 7.6.2013

Marko Kupiainen

Contents

Abbreviations

7B8B	7 Bit-to-8 Bit
8B10B	8 Bit-to-10 Bit
ADC	Analog-to-Digital Converter
ALICE	A Large Ion Collider Experiment
ASIC	Application-Specific Integrated Circuit
ASM	Algorithmic State Machine
ATLAS	A Toroidal LHC Apparatus
CAD	Computer-Aided Design
CERN	European Organization for Nuclear Research
CMS	Compact Muon Solenoid
CRC	Cyclic Redundancy Check
CSC	Cathode Strip Chamber
DC	Direct Current
DLL	Delay Locked Loop
DSP	Digital Signal Processor
DT	Drift Tube
ECAL	Electromagnetic Calorimeter
FCS	Frame Check Sequence
FPGA	Field-Programmable Gate Array
GBT	GigaBit Transceiver
GEM	Gas Electron Multiplier

HCAL	Hadron Calorimeter
HDL	Hardware Description Language
HDLC	High-Level Data Link Control
LEIR	Low Energy Ion Ring
LEP	Large Electron-Positron Collider
LHC	Large Hadron Collider
LHCb	LHC beauty
LHCf	LHC forward
LSB	Least Significant Bit
LV1	Level 1 trigger
LV1A	Level 1 Accept trigger
LV2	Second Level trigger
MRU	Maximum Receive Unit
MSB	Most Significant Bit
PCB	Printed Circuit Board
PISO	Parallel-In/Serial-Out
PS	Proton Synchrotron
PSB	Proton Synchrotron Booster
RPC	Resistive Plate Chamber
SC	Slow Control
SIPO	Serial-In/Parallel-Out
SPS	Super Proton Synchrotron
SRAM	Static Random-Access Memory

TOTEM	Total Elastic and Diffractive Cross-Section Measurement
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits

1 Introduction

Research on fundamental physics is conducted in several places around the world. For instance, examples of such places are Fermilab in the United States and DESY in Germany. There are several reasons for studying particle physics and so far many questions in fundamental physics are without answers. We can ask: what did the universe consist of in the first instants after the big bang?; why is there more matter in the universe than anti-matter?; or why do fundamental particles have mass?

European research on particle physics is concentrated at the European Organization for Nuclear Research, CERN. CERN employs just over 2400 persons, but approximated 10000 visiting scientists come to CERN for their research. CERN is known for having the largest particle accelerator in the world, the Large Hadron Collider (LHC), which has a length of 27 kilometers in circumference. Both the LHC and CERN are located on the Franco-Swiss border close to Geneva (CERN 2013).

The research conducted at CERN aims to answer the questions listed above.

1.1 European Organization for Nuclear Research

CERN was established in 1954 and it inherited its name from the European Council for Nuclear Research (in French: Conseil Européen pour la Recherche Nucléaire). The European Council for Nuclear Research was founded two years earlier with the goal of establishing a world-class fundamental physics research organization in Europe. The 12 founding member states are Belgium, Denmark, France, the Federal Republic of Germany, Greece, Italy, the Netherlands, Norway, Sweden, Switzerland, the United Kingdom, and Yugoslavia. In 2013 the number of member states has increased to 20 and 5 countries have been granted observer status (Ibid.).

The research at CERN is divided into more than 20 different experiments. Each experiment has an objective of its own, and these objectives differ depending on the field of particle physics research, for example, matter and anti-matter, dark matter, or undiscovered particles. The research takes place in the LHC as well as other accelerators and facilities (Ibid.).

1.2 Particle Accelerators at CERN

In addition to the LHC, there are several other particle accelerators and decelerators at CERN. More than one accelerator is needed because protons and ions (lead nuclei) must be accelerated to an operating level of several teraelectronvolts before they are forced to collide in the LHC. For this purpose, a cascade consisting of multiple accelerators is used to increase the energy of particles before they are injected into the LHC, Figure 1. During the accelerator chain, the particles are arranged into bunches that eventually make a beam. A bunch is a cluster of particles and a beam is a stream of bunches, each beam consisting of 592 bunches of ions or 2808 bunches of protons (Evans 2009). The particles travel in the accelerators in a vacuum because collisions with gas molecules are undesirable (Lefevre 2008). Most of the magnets in the accelerators are used to guide the particles instead of accelerating them. Liquid helium cooled superconducting magnets and wiring are used and they operate at 4 Kelvin (Evans 2009).

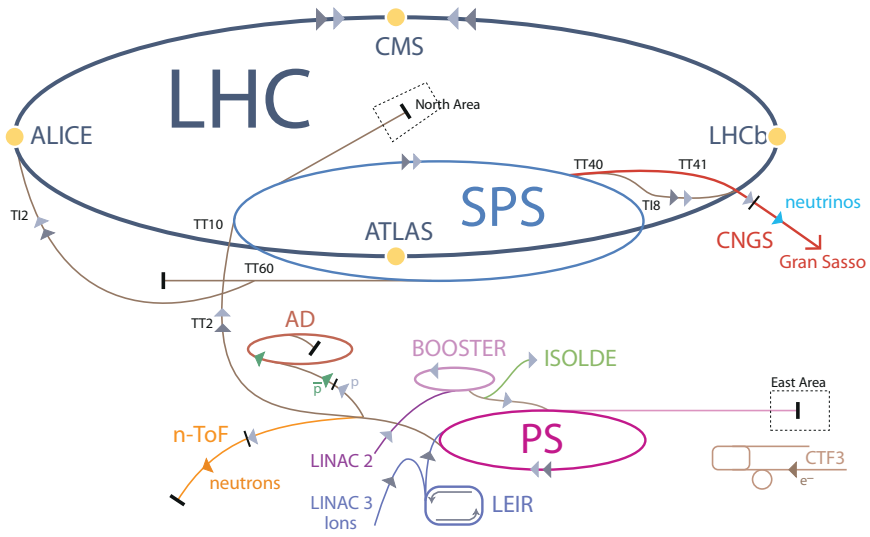


Figure 1. An illustration of the accelerator chain at CERN. The figure shows all the particle paths between the accelerators and decelerators (Lefevre 2008).

The acceleration process for protons and ions differ in the beginning of the chain, Figure 1. Protons are generated from hydrogen gas by leading the hydrogen through an electric field in which the hydrogen is stripped of its electrons. Thus, only protons continue on to the linear accelerator (Linac 2) where the energy of the protons will be increased. The acceleration of the protons is done by leading them through conductors, which are alternately charged positive or negative. The conductors behind the protons are pushing the protons and the conductors in front of the protons are pulling them. Small quadrupole magnets sustain the protons as a tight beam. From the Linac 2, the beam is passed on to the Proton Synchrotron Booster (PSB). At this point, the protons are accelerated to 50 megaelectronvolts. In the

PSB, the protons are accelerated further up to 1.4 gigaelectronvolts and delivered to the Proton Synchrotron (PS) (CERN 2013).

The ions also eventually proceed to the PS. The electrons of lead nuclei are stripped before and during acceleration. The ions are accelerated in Linac 3 in a similar way as protons are accelerated in Linac 2. This is done by using cylindrical conductors that have an alternating charge. Once the ions are accelerated in Linac 3, they are passed on to the Low Energy Ion Ring (LEIR) where each pulse from Linac 3 is split into four shorter pulses each containing two hundred and twenty billion lead ions. In the LEIR, the ions are accelerated from 4.2 megaelectronvolts to 72 megaelectronvolts. Ions are accumulated from multiple pulses before they are sent to the PS (Ibid.).

The next phase for protons and ions in the accelerator chain is the PS, which was the first synchrotron at CERN. In addition to accelerating the particles to 26 gigaelectronvolts, the proton beams are grouped into bunches in the PS. One bunch contains one hundred billion protons; a bunch is about 1.2 meters long and there is a 7 meter long gap between bunches. The distance between bunches is kept constant until the bunch crossing. A bunch crossing is a collision between two bunches in the LHC (Evans 2009). The distance between bunches is defined so that they will collide at the frequency of 40 MHz, which is 40 million times per second. For practical reasons, the actual collision frequency is smaller because there are holes in the bunch pattern on purpose (Lefevre 2008).

The protons and ions are passed on from the PS to the Super Proton Synchrotron (SPS) where the particles are accelerated to 450 gigaelectronvolts. One function of the SPS is to provide beams for the LHC and the associated experiments. The SPS has been used to research the inner structure of protons, investigate why matter manifests over anti-matter, study matter during the first instances of the universe, and search for exotic forms of matter. The highlight of the SPS came in 1983 when W and Z particles were discovered (CERN 2013).

1.3 Large Hadron Collider

Currently in 2013, the LHC is the largest particle accelerator in the world with a diameter of 27 km. It is a similar size to its predecessor the Large Electron-Positron Collider (LEP) that was located in the same tunnel that the LHC is now in. The LEP was dismantled in 2000 to make way for the construction of the new accelerator. The LEP was built in a tunnel because excavating a tunnel was easier than purchasing land on the surface. Furthermore, the crust of the earth provides a good shield from radiation (Lefevre 2008).

The LHC is designed to accelerate particles coming from the SPS to 7 teraelectronvolts. When proton beams are collided into each other at almost the speed of light their collision energy is 14 teraelectronvolts. Lead nuclei have several protons, therefore their collision energy is considerably higher compared to colliding protons. The beams colliding in the accelerator are always either protons or ions. Proton and ion beams are not collided into each other (Ibid.).

Thousands of magnets of different types are used to guide the beam in the LHC. 15 meter long dipole magnets are used to bend the beam and quadrupole magnets that are 5–7 meters in length are used to focus the beam. Magnets further along in the process squeeze the beam together before collision to increase the probability of collision between particles. The beams in the LHC travel in two separate tubes in opposing directions. There are four points along the LHC where the beam can be collided. The experiments done with the LHC are located at these points, shown in Figure 1. The four experiments are: A Large Ion Collider Experiment (ALICE), A Toroidal LHC Apparatus (ATLAS), the Compact Muon Solenoid (CMS), and the LHC beauty (LHCb) (CERN 2013).

1.4 Detectors and Experiments in LHC

The experiments in the LHC aim to search for answers to the questions in physics that are still unanswered. The four detectors at the LHC and their respective experiments are introduced below. The CMS detector is covered in more detail for the reason that the work done in this thesis is involved in the development of future detector electronics for the CMS experiment. In addition to the four experiments, there are the Total Elastic and Diffractive Cross-Section Measurement (TOTEM) experiment next to the CMS detector and the LHC forward (LHCf) experiment at the ATLAS detector (Lefevre 2008).

1.4.1 ALICE

The ALICE detector is designed to study matter at extreme energy densities, thus it is specialized for the collisions of lead nuclei. The task of the ALICE experiment is to detect and investigate a phase of matter called quark-gluon plasma that is formed under very high temperatures and densities. The phase of matter in question is thought to have formed right after the big bang. Protons and neutrons, which are both hadrons, are composed of quarks, which in turn are held together by gluons. In quark-gluon plasma, quarks and gluons are no longer inside of hadrons (CERN 2013; Lefevre 2008).

In the ALICE detector, the collisions between the lead nuclei inflict temperatures that are 100000 times hotter than the core of the Sun. In these extreme conditions, the quarks are unbound from the ties of gluons and quark-gluon plasma is allowed to form (CERN 2013). The plasma expands, cools down, breaks apart and condensates back into ordinary particles (Evans 2009). This is an abnormal state of matter because quarks have never been observed in isolation. In short, the ALICE experiment studies the state of matter that existed in the moments after the birth of the universe (CERN 2013).

1.4.2 ATLAS

The ATLAS detector is a general-purpose detector that is used for a wide range of things in physics research. Protons are collided in the heart of the detector in order to discover new physics. These include the discovery of the Higgs boson, extra dimensions, and particles that could make up dark matter (CERN 2013; Evans 2009).

Particles collide in the core of the ATLAS detector and this results in new particles that escape from the point of collision in every direction. There are six different subsystems for detecting the trajectories, momenta, and energies of the particles. Momentum can be determined by using the huge magnet system of the ATLAS detector (CERN 2013). Positively and negatively charged particles can be distinguished from each other because they spiral in opposite directions. Additionally, particles of very high momentum travel very near to a straight line. This is contrary to weaker particles that make spirals (Lefevre 2008).

1.4.3 LHCb

The LHCb experiment investigates the small asymmetry between anti-matter and matter. This manifests in interactions between B-particles that have a beauty quark. The aim of this research is to understand these interactions, and thus possibly also understand why the universe consists of observed matter (CERN 2013; Lefevre 2008)

Instead of covering the point of collision in every direction like in the ATLAS and the CMS experiments, the detectors for the LHCb are along the beam tube to detect forward particles. The reason for doing this is that the collisions resulting from the b and the anti-b quarks travel mostly in small angles with respect to the beam. The detectors are used to detect both the b quarks and its decays (CERN 2013).

1.5 CMS

The CMS experiment is a general purpose detector like ATLAS, and it is the largest of the experiments with regards to personnel; approximately 4300 people are working on the collaboration. Although the CMS and the ATLAS experiments share the same main goals (to research a wide range of physics), the CMS experiment uses different technical solutions than the ATLAS experiment and its magnet system is designed differently than the ATLAS experiment. The detector is built around a huge super conducting solenoid magnet that is able to generate a 4-Tesla magnetic field. The magnetic field is contained inside the detector by the bulk of the detector (Ibid.).

The point of collision in the heart of the CMS detector is surrounded by detectors in different layers like in an onion. Figure 2 shows one quarter of the cross-section of the entire CMS detector where the point of collision is. The point of collision is the point where the bunches of particles collide and it can be seen in the bottom left corner of the diagram. From there, different particles escape through various layers of detectors. The x-axis represents the beam travel path (Sharma 2012).

The pink area in the bottom left corner is the tracker detector. The darker area is the pixels and the lighter area corresponds to the silicon tracking system. Light green presents the electromagnetic calorimeter (ECAL) surrounded by the hadron calorimeter (HCAL) in blue (Ibid.).

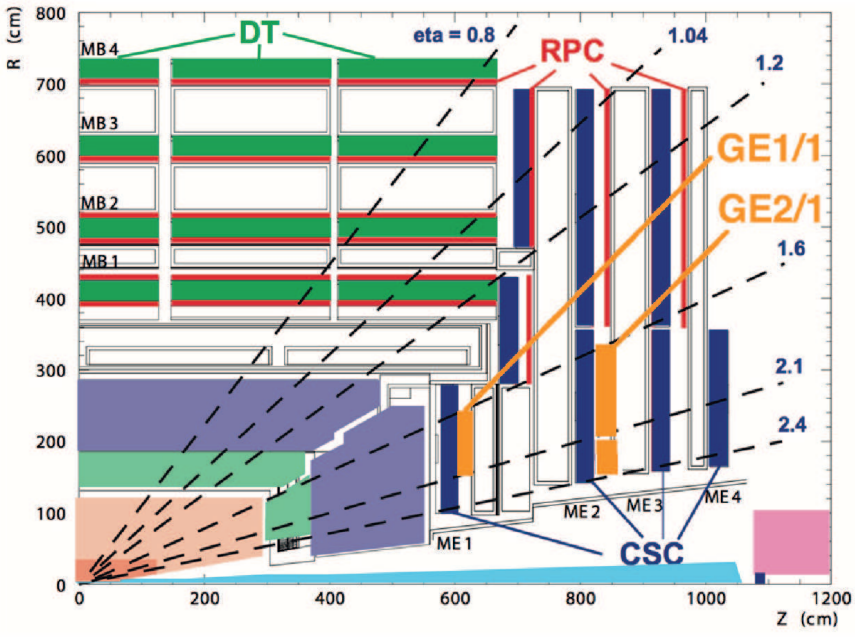


Figure 2. One quarter of the entire cross-section of the CMS detector (Sharma 2012).

In Figure 2 there are also Gas Electron Multiplier (GEM) detectors located in the endcaps at the ends of the barrel, shown as GE1/1 and GE2/1 in orange. These detectors are currently under development and are to be installed in the future. The detectors of the CMS experiment can be divided into four groups: the tracker detector, the electromagnetic calorimeter, the hadron calorimeter, and the muon detectors. The superconducting solenoid that is used to generate the magnetic field that bends the trajectories of particles can be seen between the HCAL and the muon chambers (Ibid.).

There are three different types of muon detectors: Drift Tubes (DT), Cathode Strip Chambers (CSC), and Resistive Plate Chambers (RPC) (CMS Experiment 2013). The DTs are located in the barrel section of the CMS detector and the CSCs are in the endcaps. The RPCs are located in both the barrel and the endcaps. In Figure 2, the DTs are marked in dark green, the CSCs are in dark blue, and the RPCs are in red.

1.5.1 Tracker Detector

The first detector, the silicon-strip inner-tracking system, is located right in the core of the CMS detector near the point of collision. It consists of a silicon detector called the pixels and a silicon microstrip detector surrounding the pixels. The tracker detector is used to track the flight path of a particle. The flight path is important because it can be used to derive the momentum of the particle. The whole design is made of silicon because the tracker should record trajectory data in high detail while not have any effect on the paths of particles (CMS Experiment 2013). The tracker detector has three pixel barrel layers and ten siliconstrip tracker barrel layers. The detector layers are presented in Figure 3 which shows how particles travel through the silicon detector. The endcaps have these layers so that they also cover the low angle trajectories (Evans 2009).

1.5.2 Calorimeters

The tracker detector is surrounded by the ECAL. The calorimeter consists of a barrel and two endcaps that are located between the tracker and the HCAL. The ECAL is used to measure the energy of electrons and photons during the collisions. The functionality of the calorimeter is based on the use of lead tungstate ($PbWO_4$) crystals, which are heavier than stainless steel but transparent like glass. When detecting electrons and photons it is important that the crystal scintillates when electrons and photons pass through the crystal (CMS Experiment 2013). In scintillation, a high-energy electron or photon collides into the crystal and gives its energy to the crystal. As a result, a shower of secondary photons, electrons, and positrons

emerges. The particles of the shower excite the atoms which instead emit light during de-excitation. The energy of the colliding particle can be determined from the light (Evans 2009). Figure 3 illustrates how the ECAL absorbs electrons and photons while letting other particles pass through.

The HCAL surrounds the ECAL forming a tight shell around it because the HCAL has to capture every particle that emerges from the collisions. The calorimeter detects the energy of hadrons, for instance protons and neutrons, and through in-direct measurement, non-interacting uncharged particles can also be detected, for example neutrinos. Figure 3 shows how charged and neutral hadrons are absorbed into the detector layer. The HCAL is built using brass and uses the same crystal as in the ECAL. The brass and crystal layers alternate in the calorimeter, where brass functions as an absorber and crystal emits light when a particle passes through it (Ibid.).

1.5.3 Muon Detectors

The outermost layer of detectors in the CMS detector consists of muon detectors because muons are the only particles that can be detected on the edge of the detector; they will not leave a trace in the inner layers of detectors. This is seen in Figure 3 where muons penetrate all the other detectors unnoticed. The three types of muon detectors are DTs, CSCs, and RPCs. All three types are gaseous particle detectors (Ibid.).

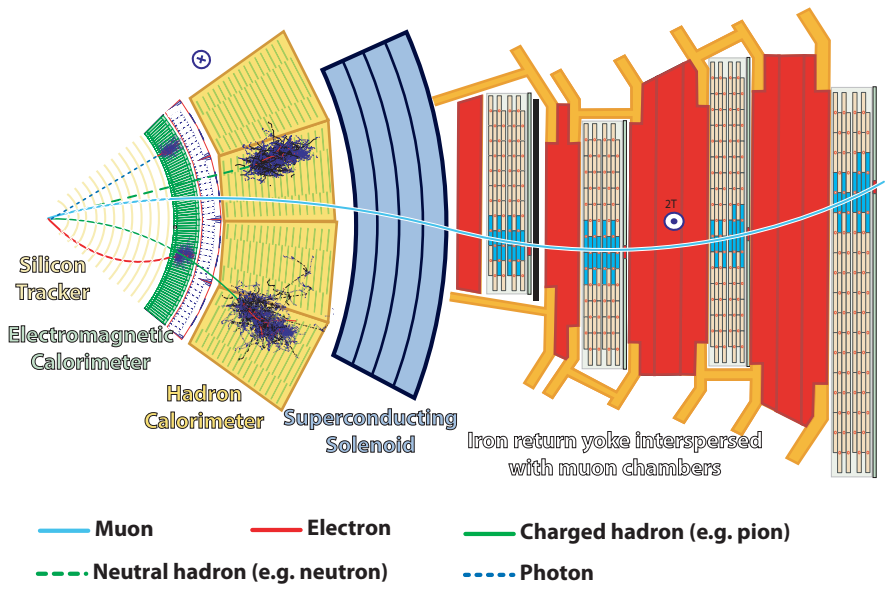


Figure 3. A cross-section in the plane perpendicular to the LHC beams of the CMS detector. The figure illustrates how different particles are observed in the different layers of detectors (CMS Experiment 2013).

The muon DTs measure the position of a muon in the barrel of the CMS. A DT is a gas-filled 4 cm wide tube inside of which is a positively charged wire. A muon or another charged particle ionizes gas atoms within the tube when it passes through. When a particle collides with the electrons of gas atoms, it initiates an avalanche, the electrons are guided by the electric field and they end up in the wire. This generates a pulse from a penetrating particle that can observed (CMS Experiment 2013).

The CSCs are located in the endcap of the CMS detector where the magnetic field is uneven and the rate of particles is high. The CSC is an array of positively charged wires which are perpendicular to negatively charged copper strips. As in the DTs, the wires and strips of CSCs are in a gas volume. When a particle passes through, it ionizes the gas by colliding with the gas atoms, thus freeing electrons. As a result, the electrons travel to the wires and the positive ions travel to the copper. Both the electrons and the ions cause a measurable pulse, and thus every particle gives two coordinates (Ibid.).

The third type of gaseous detectors are the RPCs, which are located in the barrel and the endcap. The RPCs have a worse position resolution than the DTs and CSCs but they offer a fast response. Thus the RPCs are used to improve the time resolution of the detectors (Ibid.). Because the detectors produce a very large amount of measurement data, the uninteresting events have to be disregarded. This is done with a trigger and data acquisition system that consists of detector electronics, L1 trigger processors, a readout network, and an online event filter system. The events that pass the Level 1 trigger (LV1) are read out from the readout electronics (Evans 2009).

1.6 Background

The LHC is planned to go through the second Long Shutdown sometime in 2017 or 2018. Maintenance and detector upgrades will be done during the shutdown. Therefore, the CMS experiment is now looking for new detector candidates and options for the high-eta region (the region with a high rate of particles) of the CMS detector. The detectors in the high-eta region are to trigger and track muons which are produced by colliding bunches of protons together in the heart of CMS detector. One of the aforementioned candidates for detectors is a Triple-GEM based detector and this thesis touches upon the electronics of a such detector.

The GEM chambers that are planned for installation in the CMS experiment are segmented into three columns each. The preliminary plans are to include 10 front-end chips in each column, 30 in total. On the edge of a GEM chamber, there would be three GigaBit Transceivers (GBT), one for every 10 front-end chips in a column. This would make a very high data rate

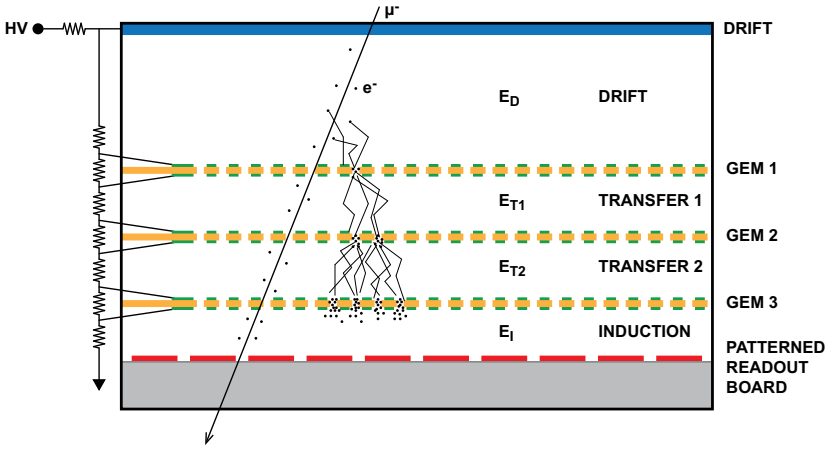


Figure 4. The structure of the GEM detector. A muon passes through the detector module and releases electron/ion pairs by colliding with atoms in the gas volume. Electrons drift to the holes in the GEM layers and trigger an avalanche. The avalanche is further amplified in the following two GEM layers and the resulting charge is read out from the PCB board. (GDD 2011)

transmission possible. The GBT allows for a data rate of 4.8 Gbps, but considering the error correction method (and other things) the effective bandwidth drops down to 3.2 Gbps. Data is optically transmitted from the GBTs to the counting room through a fiber connection.

Simply put, the Triple-GEM based detectors are built from five layers that have some distance between them, Figure 4. The three innermost layers are the GEM layers, hence the name Triple-GEM based detector (hereinafter GEM detector). The GEM layers are made of a thin, metalclad polymer foil that is pierced by a high density arrangement of holes. On top of the GEM layers is a drift cathode and under them is a readout Printed Circuit Board (PCB). A gas mixture flows between the layers and there is a large electric field between the drift cathode and the readout PCB that works as an anode (Sauli 1997; GDD 2011).

Now, when a charged particle, a muon in this case, enters the GEM detector it will release electron/ion pairs by ionization in the topmost gas volume. Ions will recombine at the drift cathode and electrons will drift into the holes in the GEM layers. The electric field is higher in the holes and when electrons drift to the high field region they will initiate an avalanche and leave towards the lower GEM layer. This way the amount of electrons is multiplied in the holes of the three GEM layers. The electrons will finally land on the PCB for detection (Sauli 1997).

When the readout PCB has collected the charge caused by the muons, the signal is conducted to the front-end electronics for preamplification and shaping. What happens in the next phase after shaping depends on architecture that is still to be decided upon. This phase is either based on an Analog-to-Digital Converter (ADC) and a Digital Signal Processor (DSP) or a comparator, Time-over-Threshold, synchronization and trigger logic. The results are stored

in Static Random-Access Memory (SRAM) and eventually transmitted onwards to the GBT through an E-Link. The E-Link consists of differential electrical wires that run between master and slave E-Ports (hereinafter referred to as E-Link interfaces).

1.7 Motivation

Currently, the front-end chip architecture has two candidates: VFAT3 and GdSP. Both are the next generation of earlier front-end chips, the VFAT2 and S-Altro respectively. In the past, the VFAT2 has been used with the GEM detectors in the TOTEM experiment (Aspell et al. 2008). By the beginning of 2013, the S-Altro chip has not been applied yet in any of the experiments (De Gaspari 2013). The S-Altro's predecessor ALTRO is used in the ALICE experiment (ALICE 2007).

The front-end chip architectures are being studied as part of a collaboration between CERN and several institutes. The collaboration is working on GEMs for the CMS project. The aim of the GEMs for CMS is to provide electronics and GEM detectors; the detectors are planned to be installed during 2017 or 2018.

Because the currently developed front-end chip architecture has similarities with the previous generation chip, it can be questioned if it is necessary to develop a new chip; this is especially true if the previous ones work well. The reason why new chips are needed is that neither the S-Altro nor the VFAT2 are compatible with the GBT. Moreover, a chip with better performance needs to be developed so that the new requirements that are a result of the higher luminosity of the LHC are met. The chips also have to be designed to accommodate the higher granularity and the higher data rate demands that will be needed in the future. Additionally, research on the possibility of sending both control and tracking data in the same transmission medium was needed.

1.8 Research Objectives and Methods

The main objective of the study is to carry out a feasibility analysis for an interface between the E-Link and a front-end chip. This is done by first analyzing what has been done before, and then the analysis is done by studying the previous generation of gaseous detector front-end chips. The literature used for this part includes several publications and reports as well as websites of institutes and organizations involved in particle detector development. The emphasis of the references is heavily based on what has been documented in previous

projects. Therefore, the previous generation front-end chips are introduced in Chapter 2 and this will serve as a brief introduction to gaseous detector electronics.

The foundation of the study itself is an implemented model of the data transmission system that is done in Verilog Hardware Description Language (HDL). This data transmission system acts as the E-Link interface in the front-end chip design. Before studying the results obtained from simulations, Chapter 3 describes the design and the functionality of the implemented code in close detail. Also, the technology that is used is introduced because its functionality heavily affects the design itself.

The fourth chapter is dedicated to the results obtained from the simulations. This part aims to find out the correlation between design and the simulated functions of the E-Link interface. Based on these results, the feasibility study is summarized in Chapter 5 and this is followed by a conclusion in Chapter 6. This includes studying possible limitations, alternatives, and further research ideas of the design.

2 Front-end Electronics for GEM Detector

As previously mentioned, there are two options for the architecture of the front-end electronics. The decision to be made will be between the VFAT3 chip and the GdSP chip; these chips differ in the way they process the signal after preamplification and shaping. However, it has been decided that the data transmission part of the chip will be designed to suit both options for the front-end architecture.

2.1 Previous Generation Gaseous Detector Front-end Electronics

The earlier implementations of front-end electronics are studied in this chapter. The examples being studied are the previous generation chips, VFAT2 and S-Altro. The description of the chips is limited to their data flow and how the data is transmitted from the chip because these are the issues covered in this thesis.

2.1.1 VFAT2

The VFAT2 chip is a front-end Application-Specific Integrated Circuit (ASIC) primarily built for the needs of the TOTEM experiment. Of its two functions, trigger and tracking, the first is to help with the generation of an LV1 by providing fast regional hit information. The second function is to provide precise spatial information for an event that has been triggered (Aspell et al. 2007).

What follows is a simple description of the data flow. The VFAT2 chip has 128 identical analog input channels and its sensors are sampled at a frequency of 40 MHz, which corresponds to the rate of collisions. Sampled data is put through a preamplifier and shaper, and then put through a comparator. When the amplitude of an input signal rises above a programmable threshold value, the comparator produces a logic 1. The following monostable generates a pulse out of the comparator output that lasts for one clock cycle. The structure is illustrated in Figure 5 (Aspell 2006; Aspell et al. 2007).

The monostable is followed by two SRAMs. The first SRAM saves the logic 1s and 0s from the monostable output. If the VFAT2 chip receives a Level 1 Accept trigger (LV1A) command, the data corresponding to the triggered time slot is transferred to the second SRAM along with the addition of labeling headers. Otherwise the data is discarded. As soon as the second SRAM contains data, a data formatting block begins to read the data to be sent

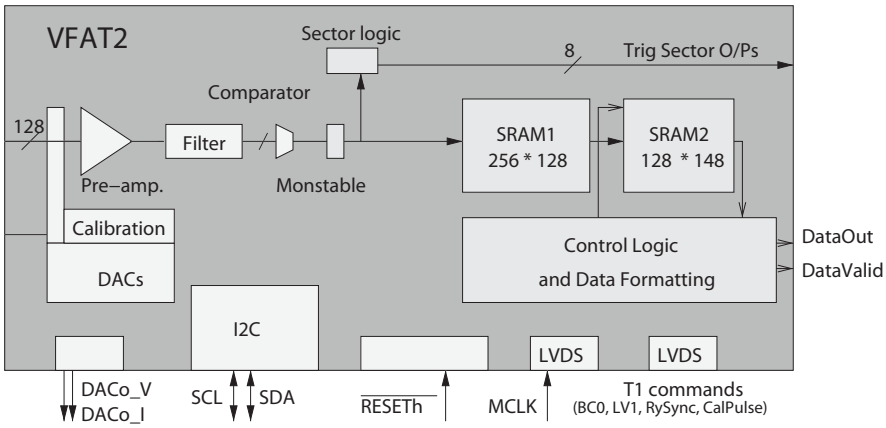


Figure 5. A block diagram of the VFAT2 chip (Aspell et al. 2007).

serially outbound from the chip. While the serial data is being sent, a separate *DataValid* output is held high, which indicates when the data in the second SRAM is to be read (Ibid.).

The input signals of the VFAT2 chip, in addition to the channels from sensors, are the 40 MHz master clock and the *T1* input. The commands to the VFAT2 chip are received through *T1*. There are four available signals: *BC0*, *LV1A* (shown as *LV1* in Figure 5), *ReSync*, and *CalPulse*. These signals are explained in Table 1 because they are also used in the VFAT3/GdSP (Aspell 2006).

Table 1. The T1 commands of the VFAT2 chip (Aspell 2006).

Command Name	Function
LV1A	Level 1 Accept trigger
CalPulse	Timing of calibration pulse
ReSync	Resynchronization of all state machines
BC0	Bunch crossing zero identifier

The data is stored in SRAM1 when the trigger LV1 is applied, and the data that is stored in SRAM1 is stored in SRAM2 when the *LV1A* trigger is used. *CalPulse* is a 200 ns long pulse for calibration. *ReSync* is used to reset all the counters and state machines. It is also used to clear the data in the VFAT2 chip. The *BC0* command is used to reset a bunch crossing counter that is used to recognize the bunch crossing event that the data comes from (Ibid.).

2.1.2 S-Altro

The S-Altro chip has been designed particularly as the readout electronics for gaseous detectors. It is designed to be used for reading the prototype of the time projection chamber in the linear collider. Currently it is in its demo phase, and for that reason it is not used in any

of the CERN experiments. As opposed to the VFAT2 chip, which only sends hit data and spatial information, the S-Altro chip sends hit data while also sending data on the shape of the charge pulse produced by a gaseous detector (Aspell et al. 2012).

The block diagram of the chip is shown in Figure 6. The S-Altro chip has 16 input channels that have a preamplifier and shaper that are similar to the VFAT2 chip. The major difference between the two is that the S-Altro is a mixed-signal chip, and thus the analogous signal is converted into digital by an ADC. The ADC has a 10-bit resolution and its maximum sampling frequency is 40 MHz (Aspell et al. 2010). Subsequently, the data is streamed to a DSP that is programmable, and different DSP sub-blocks can be skipped if desired. It is also possible to skip all the blocks and feed the data straight to the data formatter (Aspell et al. 2012; De Gaspari 2013).

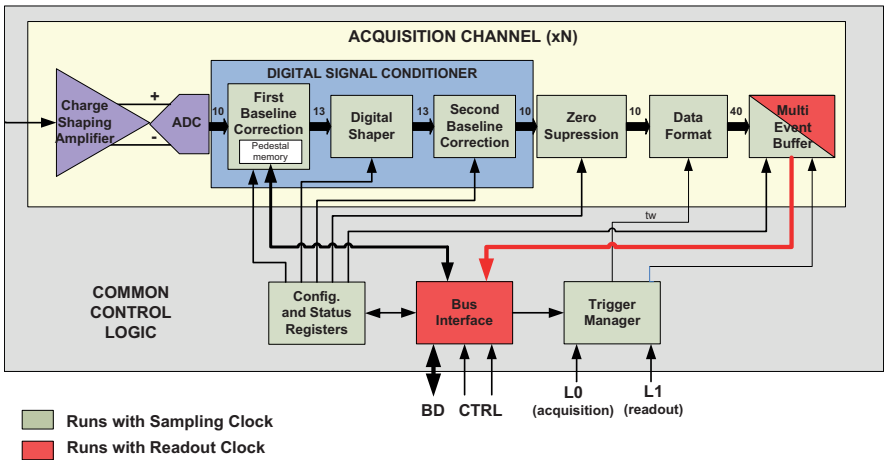


Figure 6. A block diagram of the S-Altro chip (Aspell et al. 2010).

The DSP is used for several different signal processing operations. The first block in the DSP corrects the baseline of the input signal. The baseline corrector removes the systematic offsets in a signal. These offsets are caused by the noise pickup of the clock signal and the switching of the gating grid of the detector. Afterwards, the baseline corrector signal is processed with a digital shaper. This is done so that undershoots caused by long ion tails are removed. The digital shaper is followed by another baseline corrector that reduces non-systematic changes in the baseline (Aspell et al. 2010; Aspell et al. 2012).

The aforementioned steps are followed by zero suppression. This method removes the samples that are left below the programmable threshold. Furthermore, the pulses shorter than the defined length are taken out. After the signal processing, the data formatter converts 10-bit data to 40-bit by including a header, tailer, timestamp, configuration information, and error flags. Finally the data is stored in the multi-event buffer (Ibid.).

Data acquisition is initiated with the LV1 trigger when an acquisition window is started. The

process is finished with the Second Level trigger (LV2), which is followed by storing the acquired data in the multi-event buffer. In case the LV1 trigger is followed by another LV1, the data acquired after the first LV1 is discarded. After the LV2 trigger, a readout command can be sent to the chip. Then, the contents of the multi-event buffer are sent outbound from the chip (ALICE 2007; Aspell et al. 2010).

In addition to the 16 input channels, the S-Altro chip has eight control lines and 40 bidirectional lines. These lines are digital. 20 bits of the 40-bit bus are used for address and the remaining 20 bits for data. This bus is used to access configuration and state registers (Aspell et al. 2010).

2.2 Proposed architecture of VFAT3/GdSP

The next version of the current GEM detector front-end chips is still under study. The reason is that there are no desired properties nor performance requirements defined for the chip yet. As the final chip design has not been determined, the study is based on the previous GEM detector front-end chips that have been implemented. Therefore, the newest gaseous detector electronics will be a new generation of either one of the two previous options. However, the choice will likely be the VFAT3 chip.

As was mentioned before, the VFAT3 chip will be based on the VFAT2 chip and the GdSP will be based on the S-Altro chip. Although these two chips function in different way, the architecture of the chips will be kept as similar as possible. This way it is possible to begin the study even though the objective is not totally clear. This is also done to enable an opportunity to benefit from earlier work no matter which one of the chips is chosen. A comparison between the VFAT2 chip and the S-Altro chip can be made with the help of the block diagrams in Figure 7.

As Figure 7 illustrates that with a VFAT3 chip, the input signal will not be converted to digital with an ADC, but instead the signal is followed by a comparator and the operations involving signal processing would be Time-over-Threshold, synchronization, and trigger logic. As a side note, one could see this as a 1-bit AD conversion. Conversely with a GdSP chip, the signal received from a GEM detector is converted to digital and the necessary operations are done on the DSP. The use of the DSP would enable signal processing operations, such as subtraction of background artifacts for a cleaner signal, which are not possible on VFAT3 because it is based on analog technology.

In the case of both the VFAT3 and GdSP chips, the input signal is stored in the SRAM and

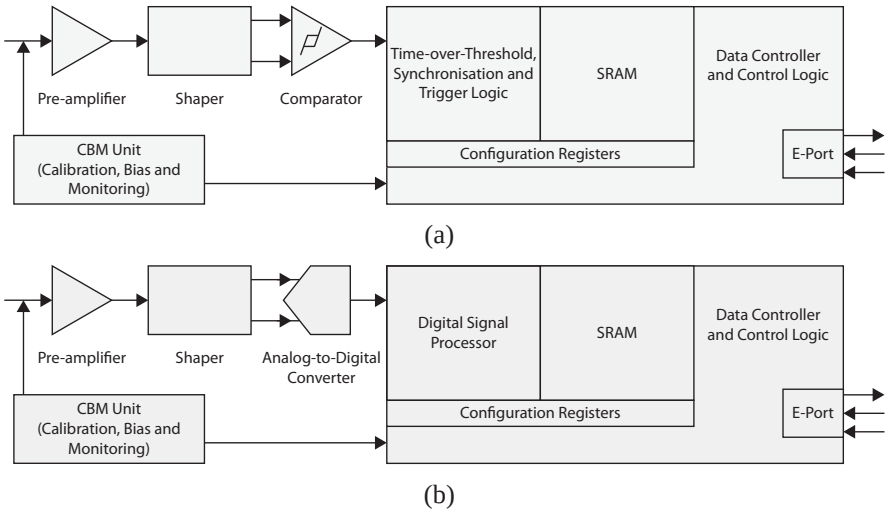


Figure 7. Block diagrams of the (a) VFAT3 and (b) GdSP front-end architectures.

the SRAM is controlled by the control logic block. The control logic block is responsible for the functions of the chip. From this point, the data flow would follow the same path as it does in the previous generation chips. In the end, on both chips, data is conveyed from the SRAM to a data controller that converts the data so that it is ready for transmission and then transmits it onwards to the GBT and counting room electronics as described in Figure 8. It is important to notice that neither the VFAT2 chip nor the S-Altro chip had a data controller or an E-Link. This is the topic of this thesis; to study if it is possible to implement a block that handles tracking and control data transmission and if so, how should it be done.

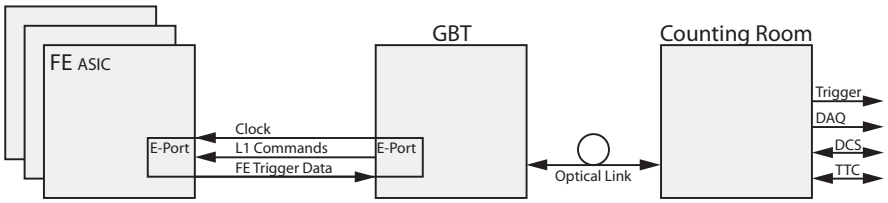


Figure 8. A block diagram of the whole data transmission system.

3 Logic Design of E-Link Interface

This chapter discusses the details of what has been planned for the E-Link interface system and its logic. This includes the features desired for the system, how they are implemented, and possible options. Also, some details about the technologies being used are discussed. Because the gaseous detectors of the previous generation did not have an E-Link at their disposal, the documentation for the existing chips will not help with the planning of the design. The interfaces of the system define a large portion of the functionality. As such, the interfaces of E-Link and other blocks are described next. An illustration of the interfaces and the internal signaling is presented in Figure 9. Later, the block diagrams of the separate blocks are presented to describe the functionality and show the state machine and Algorithmic State Machine (ASM) diagrams. All of the blocks have active high resets, even though they are not presented in the block diagrams.

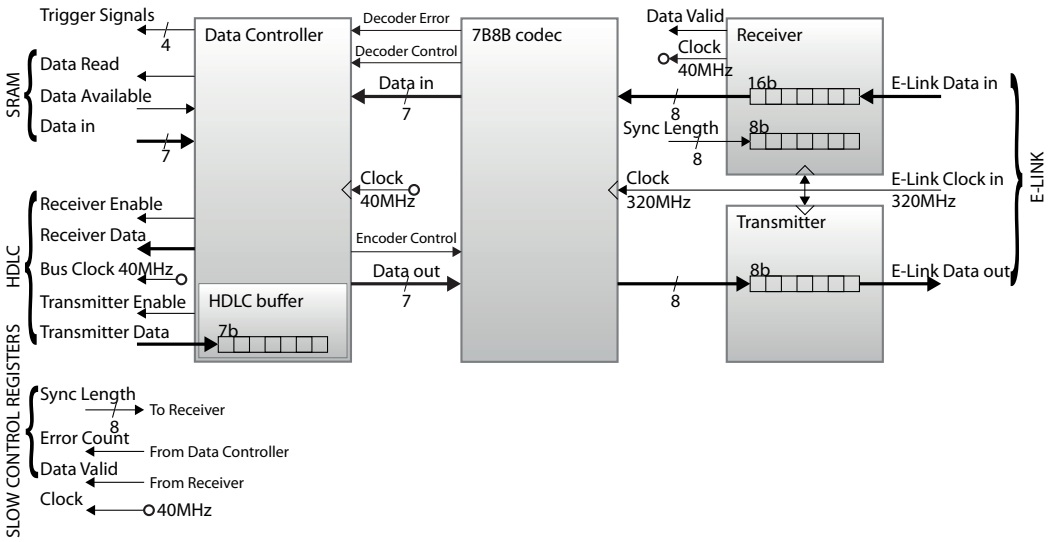


Figure 9. An illustration describing the inputs and outputs of the E-Link interface and the surrounding logic.

The system, the E-Link interface for the front-end chip, is neighbored by three other logic blocks that have their own functions on the chip. The interface for the E-Link consists of data lines for inbound data (*RXDATA*), outbound data (*TXDATA*), and a 320 MHz clock signal (*RXCLK*). This *RXCLK* is used as the system clock in the blocks responsible for data transmission and reception, and also in the 7 Bit-to-8 Bit (7B8B) codec. The receiver block has a clock divider that divides the 320 MHz clock signal by 8 and produces a 40 MHz clock signal for the rest of the E-Link interface and the front-end chip. Also, other ways to produce the system clock for the front-end chip have been discussed. For example, feeding a Delay Locked Loop (DLL) with a 320 MHz clock signal and producing a 40 MHz clock signal for the front-end chip from the delay locked clock signal. This way it would be possible for both

Table 2. A list and description of the inputs and outputs of the E-Link interface.

Interface	Name	Type	Width	Description
E-Link	RXDATA	Input	1	Serial data input of the E-Link.
	RXCLK	Input	1	320 MHz clock signal of the E-Link. Used as the system clock in the transmitter, receiver, and 7B8B codec.
	TXDATA	Output	1	Serial data output of the E-Link.
SRAM	r_en	Input	1	Read enable control signal. Indicates that data is available in the data formatter when set high.
	read	Input	1	Data read control signal. Strobed high when data is read.
	in	Output	7	Parallel data input of SRAM data.
HDLC	RX_EN	Output	1	Data enable control signal. Set high when an HDLC data bit is available in the data controller.
	RXD	Input	1	Serial data output of the HDLC bus.
	CLK	Output	1	40 MHz clock signal of the HDLC bus.
	TX_EN	Output	1	Data enable control signal. Set high when the data controller can read a bit from the HDLC bus to the buffer.
	TXD	Input	1	Serial data input of the HDLC bus.
Triggers	LV1A	Output	1	Level 1 Accept trigger. Strobed when the data controller receives command.
	BC0	Input	1	Bunch crossing zero identifier. Strobed when the data controller command.
	Resync	Output	1	Resynchronization command. Strobed when the data controller receives command.
	CalPulse	Output	1	Calibration pulse command. Strobed when the data controller receives command.

SC registers	SyncLength	Input	8	Input for an external register to define the used syncronization interval. 00000000 disables the resynchronization interval function.
	ErrorCount	Output	8	Output for an external register to count decoding errors. Stores the amount of occurred decoding errors.
	DataValid	Output	1	Output for an external register to tell the control logic about the state of synchronization of the E-Link interface.
	Clock	Output	1	40 MHz clock signal for the front-end chip.

the 40 MHz and 320 MHz clock signals to be distributed to the chip. The DLL would be used to adjust the phase of the clock signal accordingly.

Data transmitted from the E-Link interface is acquired from either an SRAM or a High-Level Data Link Control (HDLC) bus. The HDLC is used to access the Slow Control (SC) logic that is used to control the front-end chip and the contents of the external registers that the E-Link interface accesses. The interface of the SRAM, a data formatter, has a read enable output (*SRAM_r_en*) that is used to indicate the availability of data. Data is read from a 7-bit input *SRAM_in* and when 7 bits are read, the control input *SRAM_read* tells the data formatter that the data is read. *SRAM_read* is implemented to avoid the need for a buffer.

The interface of the HDLC bus has data lines for incoming and outgoing data (*TXD* and *RXD*, respectively), data enable outputs (*TX_EN* and *RX_EN*), and the 40 MHz clock signal (*CLK*). All HDLC lines are serial mode and only *TXD* is an input, the others are all outputs.

3.1 Hardware Description Languages

Integrated circuit design by hand is time-consuming in connection with complex designs and is practically impossible due to programmable logic devices. However, Computer-Aided Design (CAD) tools have been introduced in the past to help with the design process. For example, the schematic capture CAD tools were introduced around the 1970s. The HDLs for circuit design were introduced a decade later in the 1980s. The HDLs allowed flexibility in the design, and furthermore added verification of circuits at the simulation and synthesis level without having to test an actual prototype of a circuit (Kaur 2011).

Of all the HDLs, the most popular ones are Very High Speed Integrated Circuits (VHSIC) HDL (VHDL) and Verilog HDL (Ibid.). VHDL and Verilog are similar languages, but Verilog is a bit different because it closely resembles the C programming language. The major difference to consider between HDLs and conventional programming languages is that the HDLs follow combinational logic whereas programming languages are sequential. Furthermore, the functionality of the HDL must be able to be implemented on hardware.

The design implemented in Verilog has several properties that are common to all of the blocks. There are only clock and reset signals on the sensitivity lists. All of the E-Link interface uses the rising edge of the clock with the exception of the HDLC buffer. According to Reese and Thornton (2006), asynchronous inputs are generally reserved for the power-on logic, therefore the E-Link has an asynchronous reset because the reset is not part of normal operation. When the reset signal is applied, the logic resets all the registers and state machines to default values.

The state machines in the Verilog code are implemented based on what Bhasker (1998) suggests. The states are declared as parameters and used in case statements. Also, the states are declared by using one-hot encoding. In this context, one-hot encoding means having one dedicated bit in a register for one state. When a state machine is working accordingly, only one bit of the register has the value of logic 1. When using one-hot encoding, it is only possible to use one flip-flop per state (Bhasker 1998). In Field-Programmable Gate Array (FPGA) designs, one-hot encoding allows also faster operation when compared to highly encoded state machine (Xilinx 1995). However, the reason for using one-hot encoding is that the electronics in the high-eta region of the CMS detector are prone to single event upsets caused by particles passing through the detectors and electronics. A single event upset is defined as a measurable effect on a circuit caused by a nuclear particle hit. (Meggyesi et al. 2004)

A hit can result in data corruption that is due to energy deposited on silicon when an ionizing particle passes through (Ibid.). Therefore, a single event upset in a state register can change the state of a state machine. The state machines have a default state that they return to when an illegal state is encountered. Therefore, if a state register has two logic 1s, the state machine ends up in this default state. The functionality of the default state is to bring the state machine to the right state and to prevent deadlocking (Xilinx 1995).

The registers described later are big endian by default, unless mentioned otherwise. Big endian bytes have their Most Significant Bit (MSB) on the left and their Least Significant Bit (LSB) on the right. To clarify the terminology, hereinafter any binary sequence is a character unless its length is 8 bits, then it is a byte. The reason being that in the literature an 8-bit byte

is sometimes referred to as an octet. This is done if the term byte is ambiguous.

3.2 Receiver and Transmitter

The E-Link is not a commercial solution, therefore it is necessary to describe its functionality. The aforementioned E-Link is an electrical interface that is suitable for data transmission over a distance of a few meters through PCB or electrical wires in an environment with a high level of radiation. Results have proven that a data rate of up to 480 Mbit/s is possible for a data communications link, but the data rate used in this system is fixed at 320 Mbps. The E-Link is full duplex, therefore the data transmission occurs simultaneously in both directions. That is the case if both the up and the downlink have separate transmission media (Bonacini et al. 2009). Differential pairs are used as the transmission medium.

The functions of the transceiver are divided into two modules: a transmitter and a receiver, Figure 10. For data transmission, the E-Link interface has *TXDATA* for outbound data and *RXDATA* for inbound data. Both are serial mode. The transceiver obtains its 320 MHz clock frequency from the E-Link *RXCLK*, which is sent by a E-Link master, and the transceiver shares it with the 7B8B codec. The receiver divides the 40 MHz clock frequency from *RXCLK* and the divided frequency is used throughout the rest of the VFAT3/GdSP chip. The 40 MHz clock signal is synchronized to the synchronization characters received from the E-Link master.

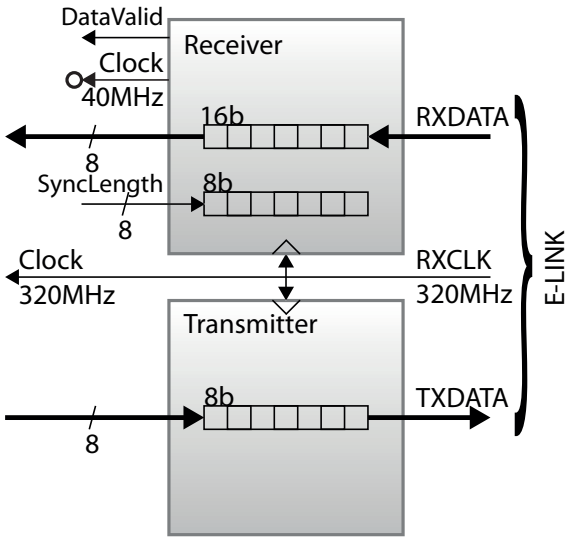


Figure 10. A block diagram of the receiver and transmitter illustrating the data and control connections between neighboring blocks.

The transmitter is a Parallel-In/Serial-Out (PISO) shift register without any other functionality. It reads 8-bit byte data available from the 7B8B encoder and sends the byte bit-by-bit to

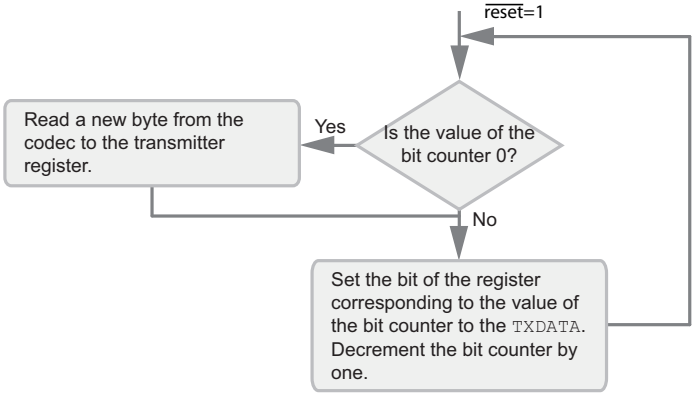


Figure 11. An ASM diagram of the transmitter logic of the transceiver block.

the E-Link on every rising edge of *RXCLK*.

The functionality of the transmitter block is described in the ASM diagram shown in Figure 11. The functionality is based on the use of a 3-bit counter that is decremented on every rising edge of *RXCLK*. If the counter has a value of 0, the transmitter reads the input byte from the encoder and sets the MSB of the new byte to the E-Link output *TXDATA*. If the counter has any other value, only the bit of the register corresponding to the value of the counter is sent to the E-Link. As a result, the transmitter sends the data in the register bit-by-bit, and when the byte has been sent, the transmitter reads a new byte to be sent. Whether it is synchronous or not does not influence the operation of the transmitter.

The receiver block has inputs for the inbound data (*RXDATA*), the clock signal (*RXCLK*), and an 8-bit *SyncLength* for a programmable synchronization interval that reads the value from an external register. There are three outputs in the receiver block. *DataValid* indicates that the receiver is synchronous when it is set to high. Then, there is the generated 40 MHz clock signal and an 8-bit data output. The receiver functions as a Serial-In/Parallel-Out (SIPO) shift register. It has a 16-bit shift register through which one bit is read from *RXDATA* on every rising clock edge of *RXCLK*. The new bit is stored to the LSB of the shift register. On the next rising edge, the contents of the register are shifted left and a new bit is added from *RXDATA*. Unless the receiver is synchronous with the transmitter on the other side of the E-Link, the data is not sent to the 7B8B decoder. An 8-bit register is used to store the current interval value.

An ASM diagram describing the functionality of the receiver block is presented in Figure 12. First, the receiver shifts the register. Then it reads a new bit from *RXDATA*, and increments a bit counter that is used to determine when a whole byte has been read. To synchronize the receiver, the contents of the register are observed. The receiver is synchronous if the two control characters (*IDLEa* and *IDLEb*) are found. By using 7B8B encoding, it is possible

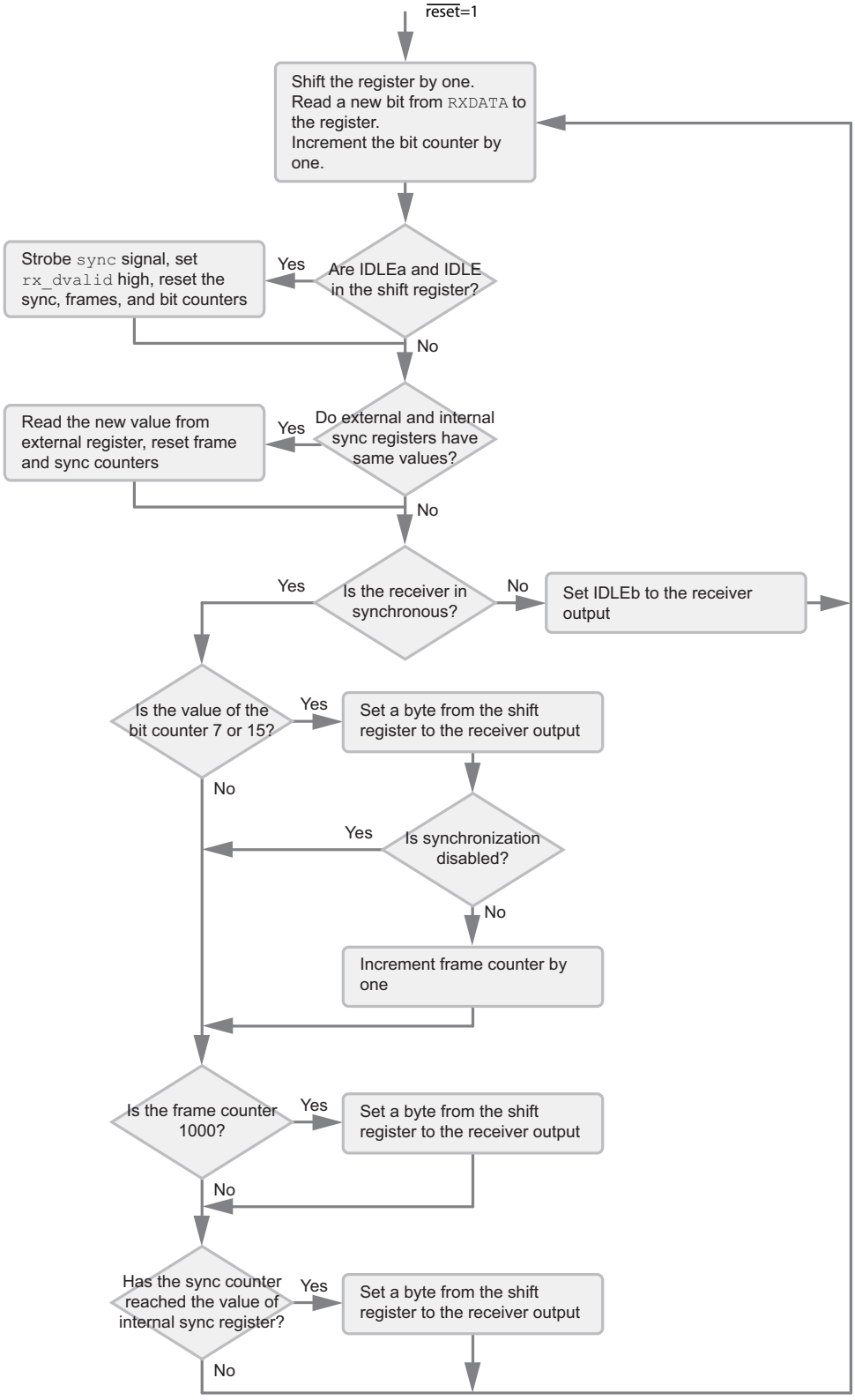


Figure 12. An ASM diagram of the receiver logic for synchronization and data transmission.

to use bytes that are not recognized as data for control characters. The synchronization is followed by strobing an output *sync*, setting the output *DataValid* to high, and resetting the bit and synchronization counters.

After checking if *IDLE*s are received, the receiver checks if the *SyncLength* value matches the value of an external register. If the values differ, the receiver reads the new value from the external register and stores it in an 8-bit register. If the values match the receiver checks to see if it is synchronous. If it is not synchronous, the receiver sets *IDLEb* to the output for the decoder. This is done because *IDLEb* has a parity of 0, as does *IDLEa*, and the decoder produces a zero sequence when decoding an *IDLE*. This way the operation of the data controller is not affected if the receiver is not synchronous.

If the receiver is synchronous, it will go through data transmission functionality. The first thing that is checked is the value of the bit counter. If the value is 7 or 15, the receiver has completed reading a byte and the byte can be set to the decoder input. Furthermore, if synchronization is not disabled, a frame counter is incremented by one.

The synchronization procedure is implemented a way where the frame counter first counts until 1000 bytes. At 1000 frames, another counter called a counter *sync* is incremented by one. The *sync* is the counter that is compared to the *SyncLength*. 1000 bytes should be enough to send a slow control command and prevent a situation where the receiver drops out of sync before the SC command is finished. If synchronization is disabled, the *sync* counter is not incremented. When the frame counter reaches a value of 1000, it is reset and the counter *sync* is incremented by one.

The last thing the receiver does on one clock cycle is to check that the value of the *SyncLength* is not exceeded. If not, normal operation continues. In other cases, the receiver interprets the situation as being without synchronization for too long and lowers the *DataValid* to drop out of sync. This is done so that the system does not continue to run out of sync for too long and to make sure that byte alignment is maintained. This could possibly be avoided by using a Cyclic Redundancy Check (CRC). The loss of byte alignment would be noticed when the CRC check sum would fail because of byte misalignment during decoding.

The receiver block also generates the 40 MHz clock signal for the E-Link interface by dividing the 320 MHz from *RXCLK* by 8. The ASM diagram of the clock divider is in Figure 13. The clock signal is generated by using a counter that is incremented on every rising edge of *RXCLK*. When the counter has a value of 3 or 7, the clock signal changes state.

Next, the functional logic for the transceiver is described. An ASM diagram for the functionality is presented in Figure 12. For data transmission, the E-Link interface has *TXDATA*

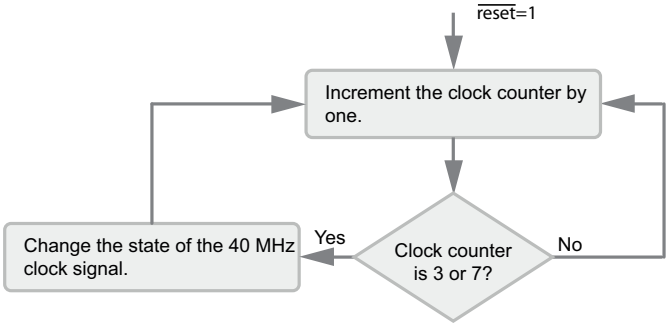


Figure 13. An ASM diagram of the 40 MHz clock generator of the receiver block.

for outbound data and *RXDATA* for inbound data. Both are serial mode. The receiver and transmitter of the E-Link interface are relatively simple in their functions. The receiver has a 16-bit shift register through which one bit is read from *RXDATA* on every rising clock edge of *RXCLK*. The new bit is stored in the LSB of the shift register. On the next rising edge, the contents of the register are shifted left and a new bit is added from *RXDATA*.

After startup, the system is not in sync with the E-Link master, hence the first thing to happen is synchronization. The 7B8B encoding is used to carry out synchronization by making use of the control characters that the encoding offers. The 7B8B encoding is discussed in more detail in Section 3.3. In the case of the receiver, two control characters *IDLEa* and *IDLEb*, which are 8-bit bytes each, are used for synchronization. The receiver is synchronous when its 16-bit register contains sequential *IDLEa* and *IDLEb* bytes. When this condition is reached, the receiver gives a signal indicating synchronous operation. If the receiver is not in sync with the E-Link master, the data sent to the 7B8B decoder is only zeros.

Because the receiver cannot be certain if it is in sync, the synchronization has to be carried out at a defined interval. The interval is read from the external register to the receiver's internal register. If the value in the external register is 0, the interval is disabled. An internal counter is incremented on every received byte. Once this counter is equal to the interval, the receiver sets a signal indicating that synchronous operation is low. After this, the receiver has to receive *IDLEa* and *IDLEb* again in order to reset the synchronization counter and to set the synchronous signal to high. If *IDLEa* and *IDLEb* are received while the receiver is synchronous, the receiver synchronizes itself to these bytes, resets the synchronization counter, and sets the synchronous signal to high. The synchronous signal is fed to an external register so that the control logic is aware of the current status of the receiver.

3.3 7 Bit-to-8 Bit Codec

The necessity of data coding was studied and the conclusion is that a line coding method is necessary. One major reason for this is the synchronization of the receiver. Several existing technologies were studied to get an idea of what is needed for the E-Link data transmission. In the end, it was proposed that the 7B8B encoding be used because it had been implemented before. The 7B8B encoding is similar to the 8 Bit-to-10 Bit (8B10B) encoding that is used in Gigabit Ethernet (IEEE 2008), Serial ATA (Hewlett-Packard 2011), and USB 3.0 (Hewlett-Packard Company et al. 2008), for example.

The 7B8B encoding used in the E-Link interface is a transmission code like its better known counterpart the 8B10B encoding. The 8B10B encoding translates a byte to a 10-bit character and the 7B8B encoding translates a 7-bit character to a byte. If 7 bits are translated to 8 bits, there are some extra bytes that cannot be interpreted as the original 7-bit characters containing data. These characters are outside the normal data encoding process. However, some bytes can be translated to 7-bits and the result is a command character that can be distinguished from normal data. Also, there are bytes with a high disparity that are considered to be corrupt data (Brooks 1980).

Similarly to 8B10B encoding, 7B8B encoding is also direct current (DC) balanced, which means that in a long bit stream the amount of logic 1s and 0s is kept as close to equal as possible. Because all the bytes cannot have an equal amount of 1s and 0s, a term of binary disparity has to be defined. The disparity is a value that is the difference between 1s and 0s. The disparity is calculated by assigning a logic 0 with a disparity of -1/2, and a logic 1 with a disparity of 1/2 (Ibid.). An easy way to calculate the disparity is to calculate the amount of 1s in a byte and subtract 4 (Bonacini 2008). For example, the byte 11000000 has a disparity of -2, 00001111 has 0, and 01011101 has +1.

The 7B8B codec used in the system was not implemented by the author. Instead the codec was done for an earlier project at CERN. The Verilog code of the codec is based on the method presented by Alexander and Nagra (1979), but has been further improved. The codec consists of an encoder and a decoder part. The encoder translates 7-bit data to 8-bit bytes and the decoder does the same thing vice versa. A block diagram illustrating the inputs and outputs of the codec is presented in Figure 14. From the block diagram, it can be seen that the codec receives 7-bit data from the data controller; the data controller also drives the control signal of the encoder. The encoded data is then passed on to the transmitter and sent to the E-Link.

The encoded 8-bit data coming from the receiver to the codec is decoded to a 7-bit character

for the data controller. The codec provides outputs indicating a command character or a decoding error. It uses the 320 MHz signal *RXCLK* from the E-Link as the clock signal. Although the data is flowing through at 40 MHz, the codec needs to run at a higher frequency to be able to update its internal variables in time. A failure in updating the variables can result in decoding errors.

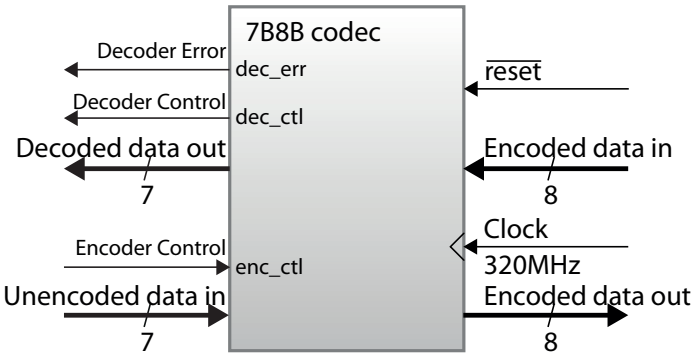


Figure 14. A block diagram of the inputs and outputs of the 7B8B codec.

Next, the functionality of the 7B8B codec is described in detail. When a 7-bit character is input into the codec, it can be encoded to a command character by setting the control input *enc_ctl* to high. Otherwise the 7-bit data is encoded as normal data. The list of all the command characters is presented in Table 3. Decoding produces a 7-bit character from a byte. If a command character is recognized, the output *dec_ctl* is set to high. In case of an error or a disparity that is too high, the decoder sets the error output *dec_err* to high. Bytes resulting in an error cannot be encoded by the encoder, therefore an error is the result of a transmission error of an unknown reason or interference with the E-Link bus. When an error occurs, the output is set to 0000000. The bytes resulting in an error are listed in Table 4.

3.4 Data Controller

The data controller acts as a multiplexer between the codec and the data sources. The data controller receives tracking data from the SRAM, slow control data from the HDLC bus, and different types of commands from the codec. In addition to sending and receiving data, the data controller is responsible for sending control characters as well as for receiving and acting upon them.

The data controller is divided into three blocks based on the type of functions. The receiver block receives data from the codec, looks for commands, and executes them accordingly. It also relays HDLC data from the codec to the HDLC bus. The transmitter block receives data either from the SRAM or the HDLC bus and relays the data to the codec. The third

Table 3. A list of the available 7B8B command characters. The first ones are command characters for controlling the E-Link interface. The second group of commands (that are used to report back) are separated by a line, and the commands in the third group are unused.

	7-bit		8-bit		
#	Binary	Hex	Binary	Hex	Description
A	0001111	0F	00001111	0F	IDLEa, synchronization
B	1110000	70	11110000	F0	IDLEb, synchronization
C	0011111	1F	01011111	5F	Filler character, this is sent when there is no data available in SRAM nor HDLC
E	0110111	37	01110111	77	Error character, used to tell master E-port that an error has occured in decoding
F	0111011	3B	10000100	84	SRAM character, a header for following data inbound from SRAM
G	0111101	3D	01111101	7D	HDLC character, a header for following data inbound from HDLC
D	0101111	2F	10010000	90	
H	0111110	3E	10000001	81	
I	1001111	4F	11001111	CF	
J	1010111	57	00101000	28	
K	1011011	5B	11011011	DB	
L	1011101	5D	00100010	22	
M	1011110	5E	00100001	21	
N	1100111	67	11100111	E7	
O	1101011	6B	11101011	EB	
P	1101101	6D	00010010	12	
Q	1101110	6E	00010001	11	
R	1110011	73	11110011	F3	
S	1110101	75	11110101	F5	
T	1110110	76	00001001	09	
U	1111001	79	11111001	F9	
V	1111010	7A	00000101	05	
W	1111100	7C	00000011	03	

block is a buffer for HDLC data because during each clock cycle only one bit of HDLC data is received. This is the reason why the inbound data is stored in a register. The buffer block also handles the signaling of data transmission coming from the HDLC bus. The block diagram is illustrated in Figure 15. The functionality of the blocks is described in more detail next.

Table 4. A list of the 7B8B characters resulting in a decoding error.

8-bit byte	Complement
00000000	11111111
00000001	11111110
00000010	11111101
00000100	11111011
00001000	11110111
00010000	11101111
00100000	11011111
00111111	11000000
01000000	10111111
01111111	10000000

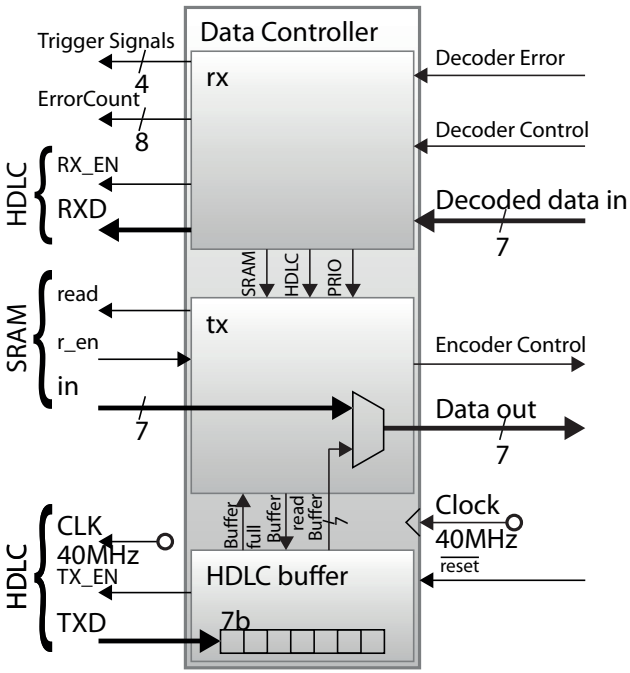


Figure 15. A block diagram of the data controller illustrating the data and control connections between neighboring blocks.

3.4.1 Command Data Handling

When the data controller receives decoded data from the codec through a 7-bit parallel input, first the codec outputs are polled for an error (*dec_err*) or a control character (*dec_ctl*). If both inputs are low, it means that the data controller is free to process the received data. The MSB is sent to the HDLC bus, however the HDLC data transmission is described in more detail in the section that covers the HDLC interface (Section 3.4.3).

If an error has occurred in decoding, the data controller is informed of this by the codec when it gives the output *dec_err*. The data controller keeps control output *RX_EN* low to prevent data from being read by the HDLC port, but the MSB is set in the data output *RXD* in any case. Also, the data controller increments data to a specific register called *ErrorCount* to keep track of the number of errors that have occurred. The value of the register is sent to an external register that can be accessed by the control logic of the front-end chip. Furthermore, the value can be read from the register by using an SC read command.

In a case where the codec has decoded a 7B8B command character, the data controller sets the MSB on *RXD* and keeps *RX_EN* low. An alternative way to do this is to actually use the command characters instead of the four LSB bits dedicated for the trigger and other commands. However, this type of implementation will not be shown in this thesis, albeit it was implemented in the code as an option. The commands delivered using the four bits include the trigger commands previously presented in Table 1. The 4-bit field is referred to as the L1 field. In addition to the trigger commands, there are commands reserved for the transmission modes of the data controller.

The commands for the data controller are listed in Table 5. The first four commands in Table 5 are explained in Section 2.1.1. When one of the trigger commands is received, the corresponding bit of the output *Triggers* rises to high for one clock cycle. The *modePRIO* command is used to switch the transmission mode so that it sets SRAM as the primary source for data. The following commands *modeHDLC* and *modeSRAM* are used to relay data only from the corresponding source and ignore the other one. As each command has a fixed bit pattern, it is obvious that it is not possible to send multiple commands in one character. Compared to the proposed alternative where these commands are fixed to the 7B8B command characters, this way is more prone to executing wrong commands because of the possible interference introduced when transmitted data. However, the benefit is the possibility of sending an SC bit to the HDLC bus in the same character as the commands for the data controller.

Table 5. The T1 and mode commands of VFAT3/GdSP.

Command	4-bit Binary	Description
LV1A	0001	Level 1 Accept trigger
BC0	0010	Timing of calibration pulse
ReSync	0100	Resynchronization of all state machines
CalPulse	1000	Bunch crossing zero identifier
modePRIO	0110	Set the transmission mode of the data controller to SRAM priority
modeHDLC	0011	Set the data controller to send HDLC data only
modeSRAM	1100	Set the data controller to send SRAM data only

3.4.2 SRAM Interface and Data Transmission

The data controller relays data that it has received from either SRAM or HDLC bus to the codec. The codec encodes the data and sends it to the E-Link. If no data is available, the 7B8B code command character *C* (the filler command character) is sent. There are three different transmission modes for the data controller: *PRIO* (default), *HDLC*, and *SRAM*. *PRIO* stands for priority (SRAM as the priority) and in this mode the data controller sends data from SRAM as long as the SRAM read enable input (*SRAM_r_en*) is high; a high *SRAM_r_en* indicates that there is data available in SRAM. Once *SRAM_r_en* is low, the data controller switches its state and reads the HDLC buffer and sends the contents to the codec if the buffer is full. The *HDLC* mode prohibits sending data from SRAM, thus only the data from HDLC is sent. The *SRAM* mode works in the same way as *HDLC*, but only data from SRAM is sent.

These modes function around a state machine that is made up of three states. This is illustrated in Figure 16. When the data controller starts functioning, the initial state is *interrupt* and the initial mode is *PRIO*. In the *interrupt* state, the data controller sends the command character *F* (the SRAM data header) to indicate the beginning of a data stream from the SRAM. Then, the state machine switches to the state *txSRAM*. If there is no data available from the SRAM, the state machine returns back to *interrupt*, sends the command character *G* (the HDLC data header), and switches to the state *txHDLC*. Once the HDLC buffer is full, the 7-bit data is read from the buffer and sent for encoding. If data comes available in the SRAM, it is indicated by the data formatter setting *SRAM_r_en* high. In this case the state machine switches first to *interrupt*, sends the command character *F*, and enters the state *txSRAM*.

The *SRAM* mode restricts the data controller so that it only sends data from the SRAM. In

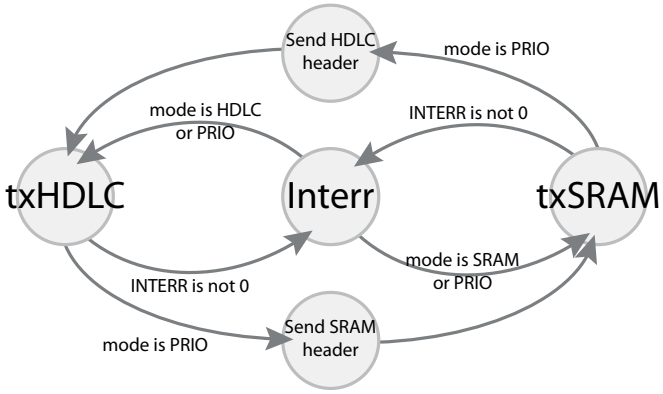


Figure 16. A state machine diagram of the transmitter block of the data controller.

this mode, the state *txHDLC* is unavailable. The *HDLC* mode is similar to *SRAM* but it only accesses HDLC data, as opposed to *SRAM*. The state *interrupt* is also used when the receiver block of the data controller receives an error from the codec. In case of error, the state machine halts the current data transmission, sends the command character *E* (report error), and returns to the state it was previously in.

The *interrupt* state is accessed in any of the transmission modes. Switching between the states is controlled by the signals *modePRIO*, *modeSRAM*, and *modeHDLC*. Once any of the signals strobes, it is registered by setting a corresponding bit high in a register *INTERR*. The logic constantly observes that the content of the register *INTERR* is zero. In other cases, the state machine switches to the *interrupt* state and does one of four operations based on which bit in *INTERR* is high. The four operations are illustrated in Figure 17. The first priority is the error reporting that is used to send the command character *E* (report error) while the encoder control input *enc_ctl* is set to high. Also, the error bit in *INTERR* is set to 0 to mark that the error has been reported. The next state is either *txHDLC* or *txSRAM* depending on what the current state is.

The other three operations are related to mode changing. *INTERR* has bits for *SRAM*, *PRIO*, and *HDLC*. In each of the operations, a header is sent first: the SRAM header in the modes *SRAM* and *PRIO*, and the HDLC header in the mode *HDLC*. The encoder control input *enc_ctl* is strobed on high during control character transmission. Also, the corresponding bit is set to zero to finish the operation. The next state is *txSRAM* for the modes *SRAM* and *PRIO*, and *txHDLC* for the mode *HDLC*. If the content of *INTERR* is something other than logic 1s and 0s, the ASM goes to the default state and sends the command character *C* (filler).

The ASM for the state *txSRAM* is illustrated in Figure 18. In the state *txSRAM*, the data controller sends SRAM data as long as *SRAM_r_en* is high (data is available). The encoder control input *enc_ctl* is kept low. To read the last available bit after SRAM data is no longer

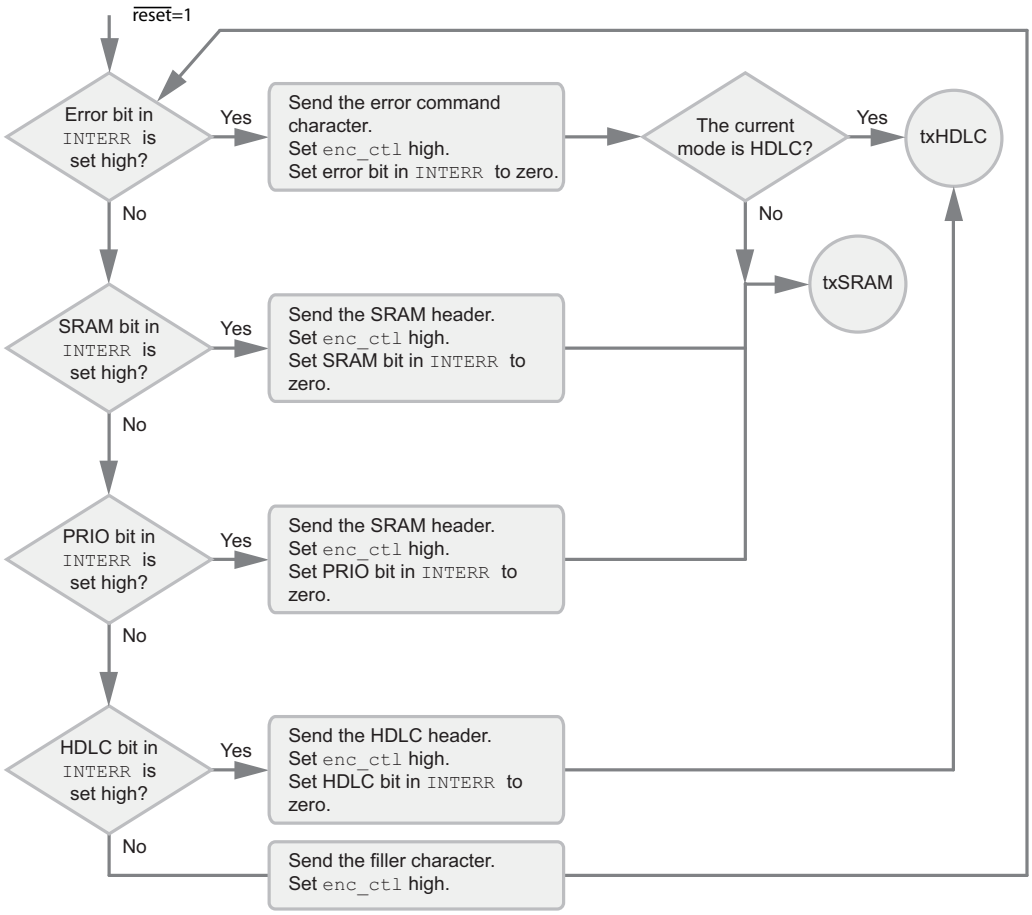


Figure 17. An ASM diagram of the state *interrupt* of the data controller transmitter block.

available, the ASM reads data while *SRAM_read* is high. The output *SRAM_read* is high when *SRAM_r_en* is high and the state is *txSRAM*. In practice, *SRAM_read* is high for one clock cycle after *SRAM_r_en* has been lowered.

When SRAM data is not available, the transmitter block checks if it is in priority mode. If the current mode is *PRIO*, the transmitter sends the HDLC header and switches to the state *HDLC*. If the mode is not *PRIO*, and is *SRAM* instead, the transmitter sends a filler character.

3.4.3 HDLC Interface

The HDLC buffer works as the interface between the data controller and part of the HDLC bus. As was already mentioned, the data controller takes care of the outbound SC data that goes to the HDLC bus, and the HDLC buffer handles inbound SC data. Because *TXD* is a serial mode input, the buffer receives only one bit per clock cycle. This is the reason why it was decided to implement a 7-bit buffer as a place to store 7 bits of SC data while waiting for an opportunity for the data controller to transmit SC data.

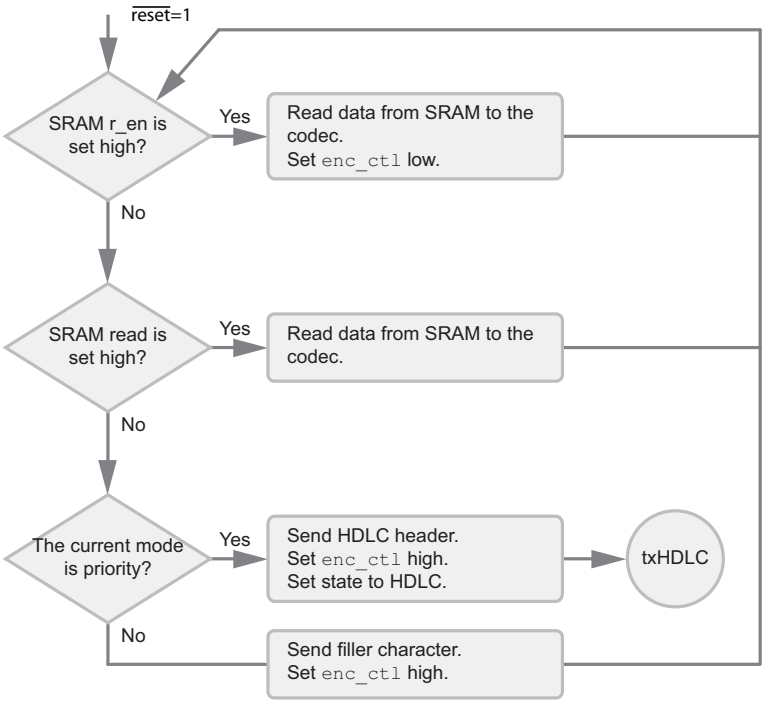


Figure 18. An ASM diagram of the state *txSRAM* of the data controller transmitter block.

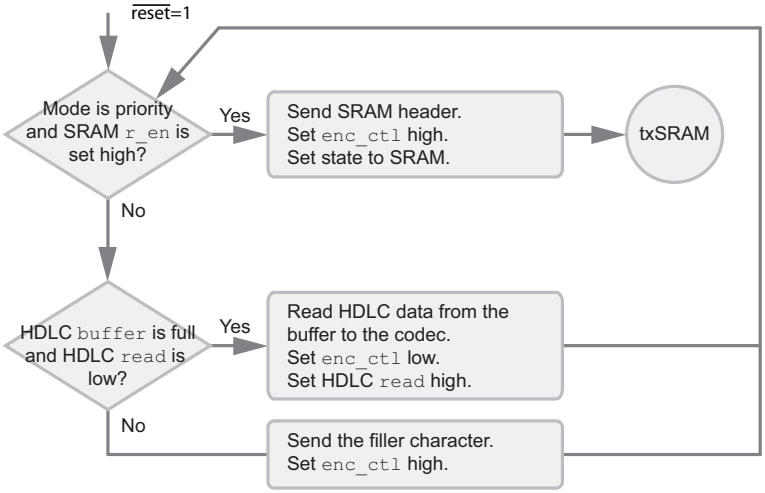


Figure 19. An ASM diagram of the state *txHDLC* of the data controller transmitter block.

The HDLC data frame structure is shown in Figure 20. The data frame begins with a flag byte that is used for synchronization and as an inter-frame fill. The *FLAG* is followed by an address byte, which is set as 11111111 for all stations, and a control byte, which can define different types of frames according to its content. Both the address and control fields can be omitted if needed (Simpson 1994a). The protocol byte, which can be either 8 or 16 bits, defines the used protocol. The information field has a varying length. For both the information and padding fields, the maximum length is defined by a Maximum Receive Unit (MRU), which is 1500 by default. The padding field is optional. If the padding field is in

use and the length of the information field is less than the MRU, it can fill up to the MRU with padding bytes (Simpson 1994b). The last field is the Frame Check Sequence (FCS); this application is a 16-bit CRC. The CRC is a common error-detecting code used in digital networks and storage devices. After the FCS, a flag byte that marks the end of the frame will follow. The next byte is either the next address or another flag (Simpson 1994a).

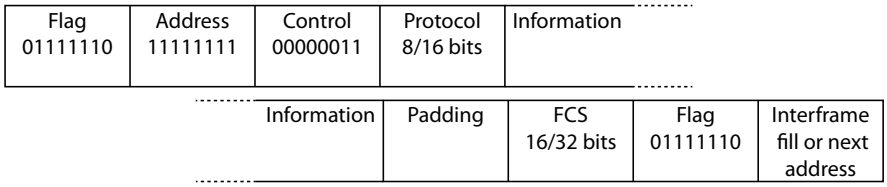
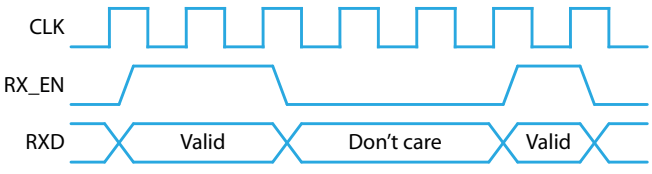


Figure 20. The structure of an HDLC data frame according to the HDLC protocol (Simpson 1994a).

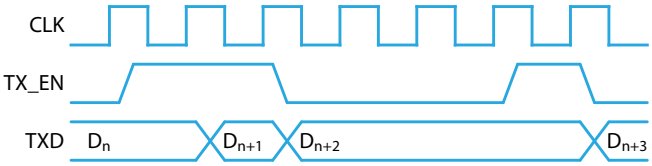
The data transmission in the HDLC bus is controlled by the enable outputs. The enable signals are named from the point of view of the HDLC driver on the other end of the bus. For this reason, the read enable signal is the output *TX_EN* and the write enable is the output *RX_EN*. In the design, the HDLC buffer controls the read enable output *TX_EN* and the receiver block of the data controller controls the output *RX_EN*. When *TX_EN* is set to high, a bit from the input *TXD* is read on the rising edge of the *CLK* (40 MHz). If an enable signal is low, the bit is ignored. Due to difficulties with timing the signals right, the *TX_EN* is set to both high and low on the falling edge of the *CLK*. This is the only case where the falling edge of the clock signal is used.

The data controller sets the MSB of the 7-bit character from the 7B8B codec to the HDLC serial output *RXD* and the transmit enable output *RX_EN* is set to high for one clock cycle. This tells the HDLC port on the other end of the bus to read the bit. In case either of the inputs *dec_err* and *dec_ctl* are high, the data controller keeps the *RX_EN* output low. The bit however is set to *RXD* regardless of the state of *RX_EN*. An illustration of the HDLC signals is shown in Figure 21.

The state machine of the HDLC buffer is a rather simple one (see Figure 22). It has two states: *WAIT* for waiting for the buffer to be read and *READ* for reading a bit from the input *TXD*. During power-on, the initial state of the buffer is *READ*. The buffer sets *TX_EN* to high, shifts an internal 7-bit shift register, reads a bit from the *TXD* to the register, and increments a counter. This is repeated until seven bits are read from the *TXD*. Then the counter is reset, the contents of the register are set to the buffer output, and a control output *buffer_full* is set to high so that the data controller knows that SC data is available. Once the data is read by the data controller, a *buffer_read* signal is strobed and the HDLC buffer knows to set the *buffer_full* to low and switch the state to *READ*. If the state machine goes into an unknown state, the default state is defined so that it becomes *READ*.



(a) HDLC data reception



(b) HDLC data transmission

Figure 21. An illustration of the HDLC signals in data communication.

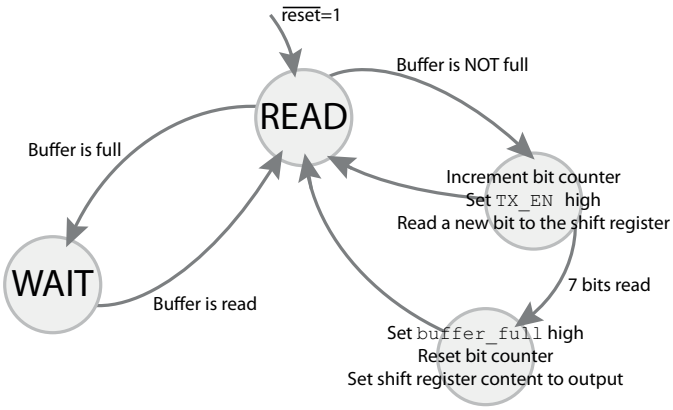


Figure 22. A state machine diagram of the HDLC buffer of the data controller.

4 Simulation Results

In this chapter the logic described in the previous chapter, which is implemented in Verilog, is simulated using Cadence CAD tools. The simulation results (presented later) are produced from Cadence SimVision outputs. The results are purged of irrelevant data, such as timestamps and filenames that have been added to the digital printouts by the program. The chapter begins with a description of the test bench environment for the E-Link interface. After explaining the test bench that has been implemented and used, the different features of the interface are presented using the results obtained from the simulations.

The values in the simulation results are presented in hexadecimal or binary forms. The hexadecimal values have an h' prefix and binary values a b'. Some of inputs and outputs are one bit long, and therefore these can have only a value of logic 0 or 1.

4.1 Description of Test Bench

The implemented test bench for the E-Link interface covers all the available interfaces with the external registers, the SRAM data formatter, the HDLC bus, and the E-Link. Considering the system as a whole, there is a GBT between the E-Link and the counting room electronics. However, because the GBT protocol is transparent to the user, the GBT can be omitted from the simulations.

The test bench blocks for SRAM and HDLC provide the data to be transferred; this data is observed again at the E-Link master. The SRAM is simply an array from which 7-bit characters are input into the *SRAM_in* output and read by the data controller when *SRAM_r_en* is set to high. When the data controller sets *SRAM_read* to high, the SRAM block inputs another character into the *SRAM_in* bus. The *SRAM_r_en* output is sometimes lowered to mimic an actual SRAM that might not have data available all the time.

The HDLC block is based on a preliminary Verilog code of an HDLC bus controller implemented by another group collaborating on the GEMs for CMS project. The HDLC controller functions only as a source of data that would function accordingly to the enable signals. In the test bench, the HDLC controller only sends the HDLC flag characters. This is sufficient enough for simulations because the controller lacks the functionality to access registers.

The major part of the test bench consists of the E-Link module, which mimics the E-Link master and its control logic. The function of the master is to provide the necessary signals that the E-Link interface needs to function. Generation of the 320 MHz master clock *RXCLK* is

done in the E-Link master block. The data generation is implemented in a similar way to the SRAM block. In addition to the data array, the master has a 7B8B codec and almost identical transmitter and receiver blocks compared to the E-Link interface. The 7B8B codec is used to encode the transmitted data, but this can be bypassed. Bypassing is a suitable approach when the E-Link interface is tested against corrupted data; the 7B8B encoder cannot provide corrupt data by itself. The data sent from the E-Link interface to the master is decoded and makes it possible to see whether or not the data is valid.

Several different profiles were implemented in the test bench to make the testing of different features easier. The profiles are basically settings for the data array content because the E-Link interface responds to the command it receives. This way, the interface can be tested for things like mode changes, trigger commands, and erroneous data. All of the profiles share a common feature: the *IDLE* characters are sent first and various commands, which control the E-Link interface, follow.

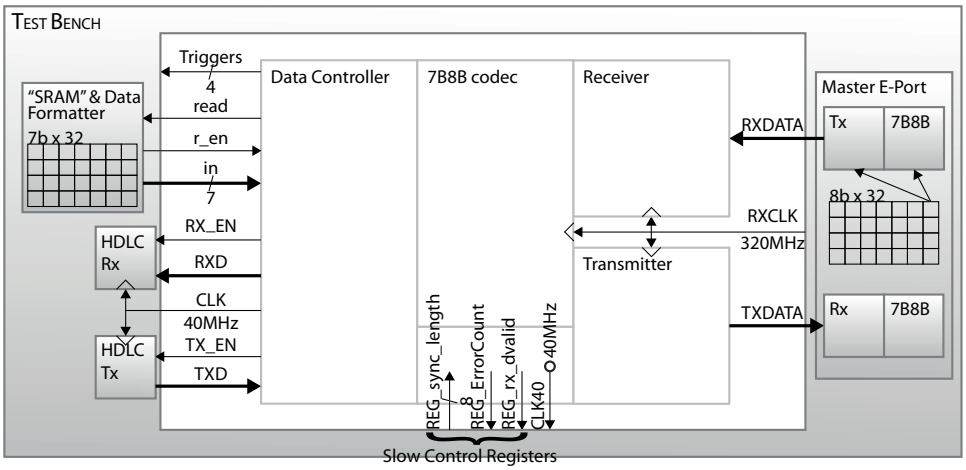


Figure 23. A block diagram describing the implemented test bench for the E-Link interface. Tx stands for transmitter and Rx stands for receiver.

4.2 Receiver and Transmitter

The simulation results for the receiver and transmitter are presented in this section.

4.2.1 Receiver Synchronization

The functionality of the receiver is presented in Figure 24. In the beginning, it can be seen that the generated clock signals *CLK40* and *HDLC_CLK* are in an unknown state. This state is shown as the signal having both a high and a low state simultaneously. However, this

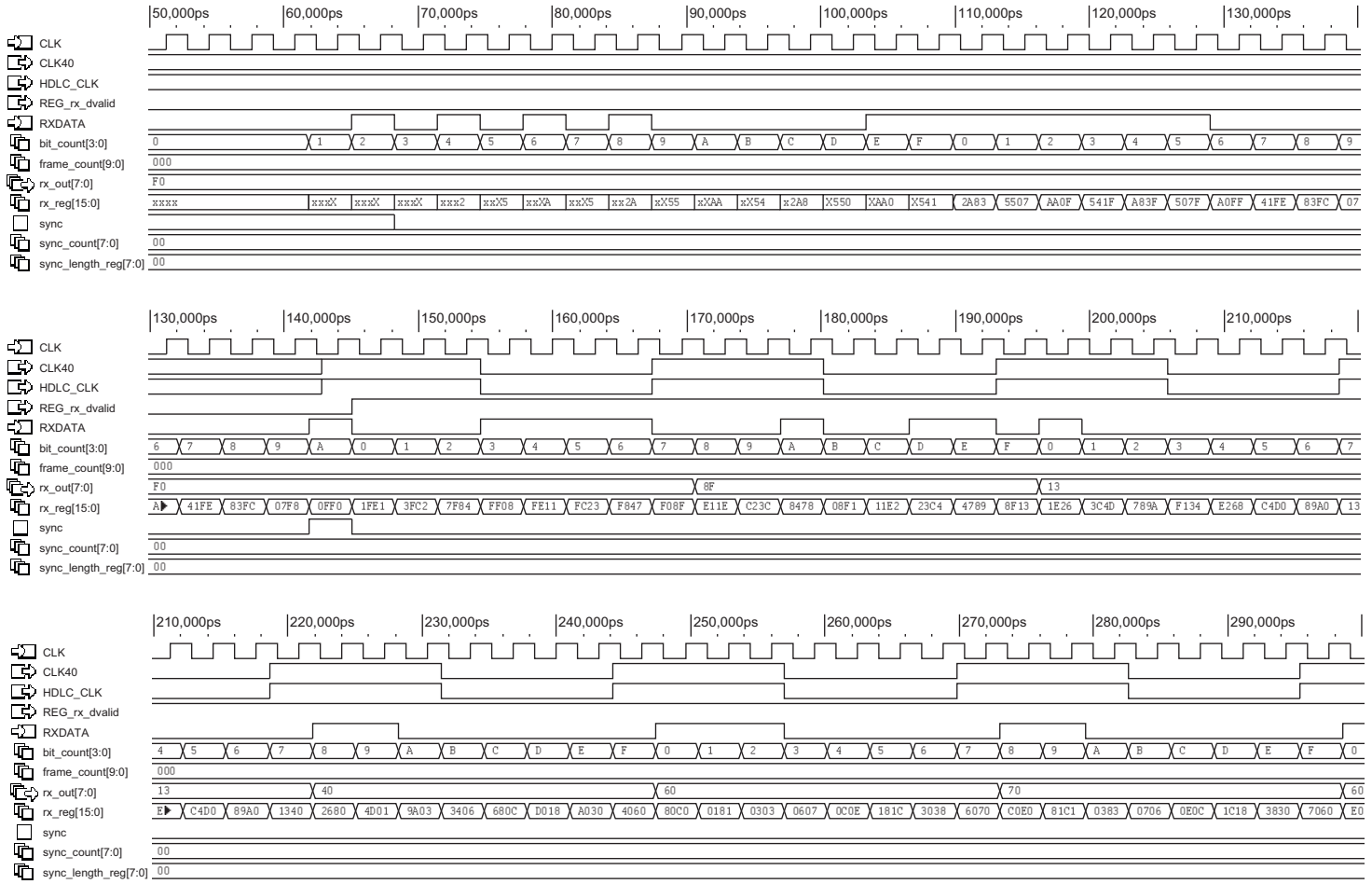


Figure 24. Simulation results illustrating the functionality of the receiver block. This describes how the receiver gets in sync with the transmitter on the E-Link master by reading *RXDATA* and receiving *IDLEa* and *IDLEb*. The results also show how the clock divider begins to function.

changes when the system synchronizes by locking on to the *IDLEa* and *IDLEb* characters. First a bit stream b'010101010 is received from E-Link input *RXDATA* and read to the 16-bit shift register *rx_reg*, but clock generation will not begin yet. The bit stream in *RXDATA* is followed by the *IDLE* characters b'00001111 and b'11110000, which are used to synchronize the system. The lock-on is recognized when both bytes (seen as a value of h'0FF0) are in the shift register of the receiver; lock-on happens at 140.30 ns.

Once the receiver is synchronous, the signal *sync* strobes for one clock cycle. This indicates that the receiver has received the *IDLE* characters that reset clock signal generation. The initialization of clock generation can be seen in Figure 24. This happens after synchronous operation is reached at 140.30 ns when the clock signals *HDLC_CLK* and *CLK40* shift from the unknown state to 40 MHz oscillation. When the system is synchronous, the 4-bit counter *bit_count* is reset to read the incoming bytes correctly.

Synchronization also resets the synchronous counters, but this functionality is seen later in Figures 25 and 26. In Figure 24, the synchronous counter has been disabled; this can be seen when the bit counter is not incremented when a byte is read to the shift register from *RXDATA*. Also, the external register value *REG_rx_dvalid* is set as logic 1 to indicate that received data can be decoded and processed. As seen in Figure 24, at 170.00 ns, the received byte is set to the receiver output *rx_out*. The data set to the output will be decoded in the 7B8B codec. A new byte will be set to the output *rx_out* when *bit_count* is h'7 or h'F and *REG_rx_dvalid* is set to high.

4.2.2 Receiver Resynchronization

Maintaining synchronous operation is important and the corresponding functionality is shown in Figure 25. In the figure, it is shown how the receiver functions when it receives *IDLE* characters in varying phases. The first phase of synchronization adjustment happens at 7.11 µs. It can be seen that the clock signal rises three clock cycles earlier compared to the signal before 7.11 µs. The clock signal phase adjustment can be seen better at 7.21 µs where the clock signal from the previous phase is seen for a longer time. At 7.21 µs, the clock signal is advanced by two clock cycles.

Finally, the third adjustment is seen at 7.30 µs where the clock signal is adjusted by four clock cycles. This four clock cycle adjustment basically produces a 180 degree phase shift. A rapid high-low-high shift is seen in the clock signal at the synchronization point. The source of this behavior is the reset mechanism of the clock divider, which sets the clock signals to low on reset.

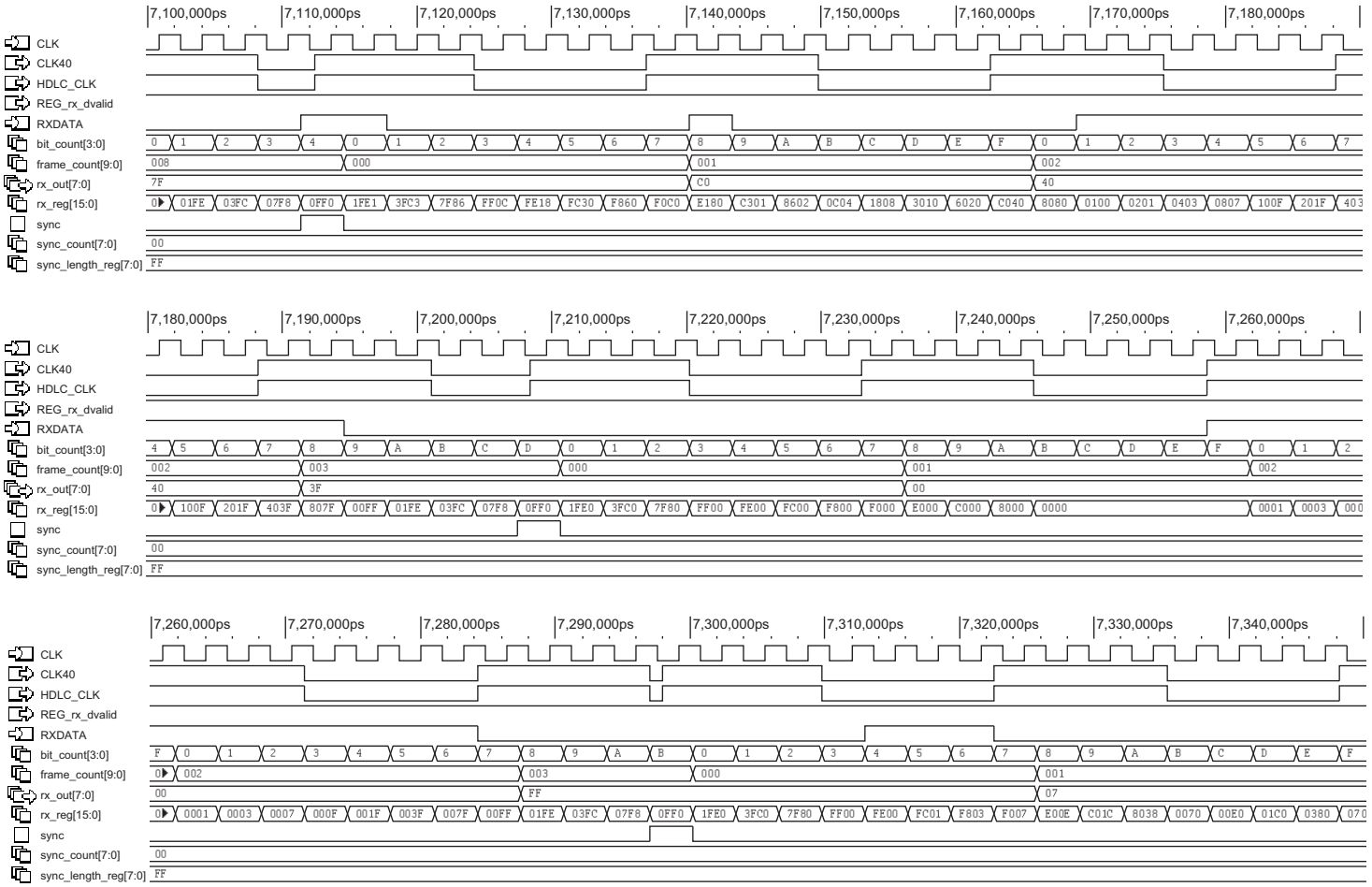


Figure 25. Simulation results illustrating how the receiver functions when it receives *IDLE* characters with adjusted phase. This simulation result shows how the phase of the generated clock can be shifted by inbound synchronization characters.

In a similar way to the system start up, the resynchronization of the bit counter *bit_count* is reset. Unlike Figure 24, the synchronization counter is now enabled. On every byte read, the counter is incremented by one.

Figure 26 shows how the receiver goes out of sync after 16.17 µs; this results in the output *REG_rx_dvalid* being set to low. The previously received byte is kept in the output *rx_out*, but after next byte is received at 16.19 µs the output is changed to *IDLEb*. At 25.59 µs, the receiver has received *IDLEa* and *IDLEb* and has regained a synchronous state. The output *REG_rx_dvalid* is set back to high and the first data byte is set to the output *rx_out* after 25.61 µs.

The functionality resulting from changing the value of the synchronization interval is illustrated in Figure 27. The new value h'FF is set to the external register *REG_sync_length* at 1.06 µs. The previous value was h'00. This means that synchronization was disabled before and now it is enabled. The new value h'FF is read to the register *sync_length_reg* in one clock cycle.

4.2.3 Transmitter

The transmitter of the E-Link interface functions as a simple serializer. This is shown in Figure 28: the transmitter reads a byte from the codec to the register *tx_reg* when the 3-bit counter *bit_count* is h'0 and begins to transmit the byte to *TXDATA*. The byte is sent starting from the MSB and the bit counter is decremented on every clock cycle.

At the bottom of Figure 28, there are 4-bit counter *bit_count* and 7-bit shift register *rx_out*. These two are part of the test bench and are used to illustrate that the data sent from the transmitter is read correctly on the other end of the simulated E-Link.

4.3 Data Controller

The data controller simulations include command and error handling, transmission modes, and the HDLC bus functionality. The command handling shows how the data controller responds to any commands it receives. When a decoding error occurs, the data controller is expected to respond to the errors by incrementing the error counter and sending a dedicated command character. This functionality is shown in the simulations. The three data transmission modes are simulated and their properties are illustrated by simulation snapshots. Finally, the HDLC functionality is simulated and the operation of the HDLC bus is closely tied to the

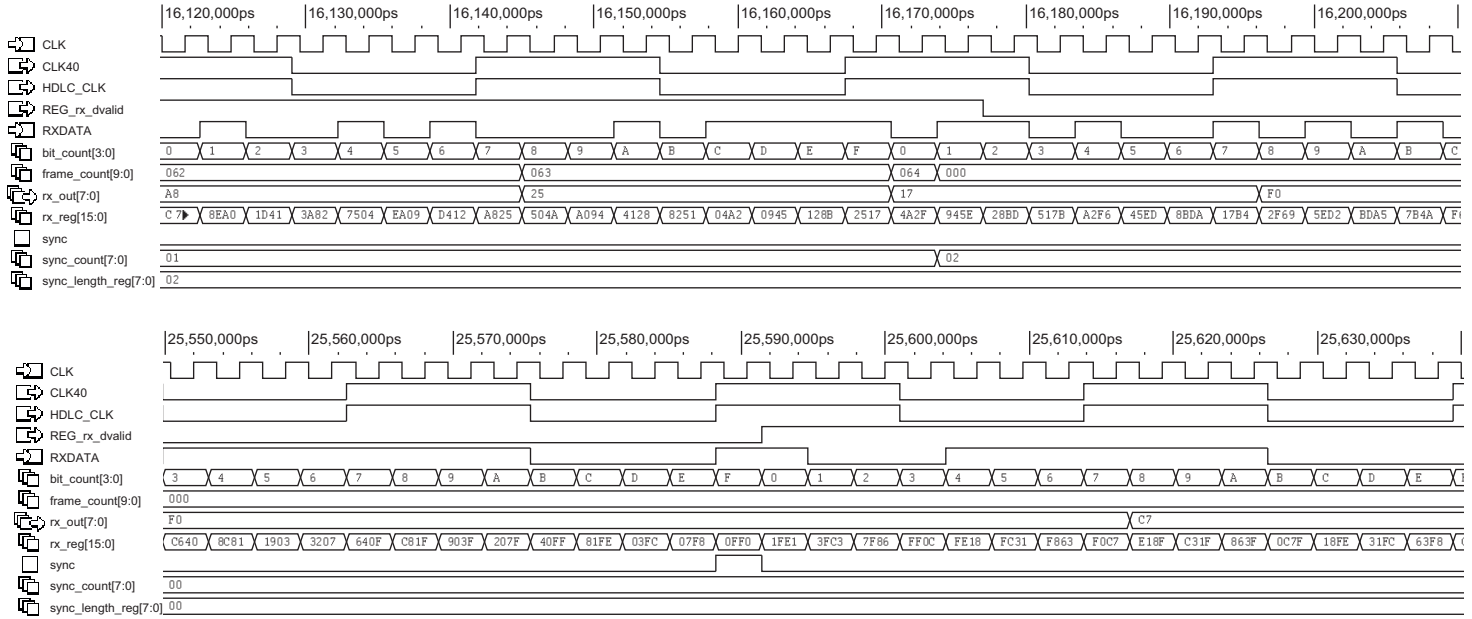


Figure 26. Simulation results of the receiver when it first loses sync after 16.17 µs and later regaining it at 25.59 µs.

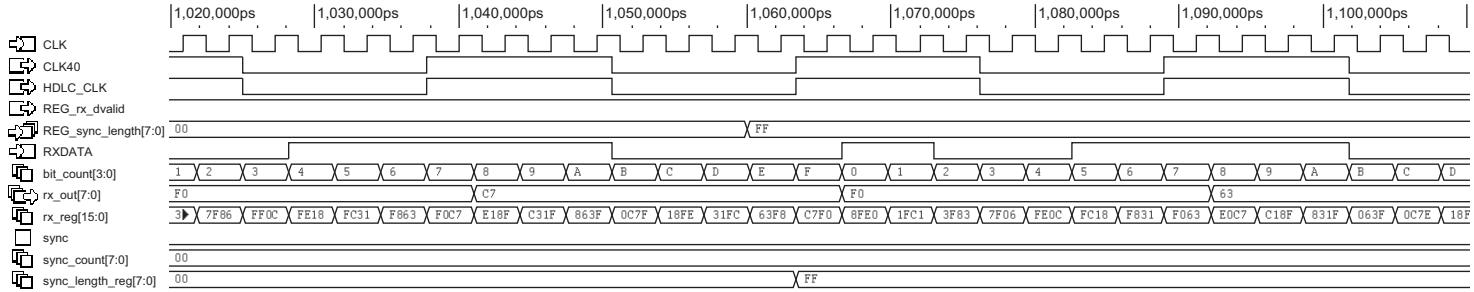


Figure 27. A simulation snapshot showing the synchronization interval value change. A new value is set to the external register *REG_sync_length* at 1.06 µs and it is read one clock cycle later.

transmission modes.

4.3.1 Command Handling

Figure 29 shows how the receiver block of the data controller responds to the trigger commands in *eport_in*. The setup in the test bench sends a burst of trigger commands to show that the data controller can handle such an extreme situation. Sending a burst of trigger commands to the E-Link interface is not considered to be normal functionality. At the bottom of Figure 29, the input *dec_in* shows the codec input from the receiver. At 46.90 µs, three incoming command characters can be seen: *IDLEa*, *IDLEb*, and *IDLEa* (again). The command characters are followed by h'1B (b'00011011 in binary). h'1B is an example of characters that have the same value when encoded because of the parity of 0.

As illustrated in Figure 29, it is not possible to send multiple trigger commands in one character. However, it is possible to send multiple trigger commands following each other as seen in the output *trigger_out*. The burst of trigger commands begins at 47.00 µs and is finished by 47.50 µs. During the period all the triggers are enabled and at 47.12 µs, it can be seen that an L1 command pulse can be lengthened to span out over multiple clock cycles if needed. By default, a trigger command has a duration of one clock cycle.

There are several mode commands accompanying the trigger commands. It can be seen that it is not possible to send a mode command and L1 command in the same character. Furthermore, sending trigger commands and mode commands right after each other does not negatively affect the functioning of the data controller. More information on the mode command functionality is shown in Figures 30 and 31.

Figure 29 also presents some other functionality of the receiver part. Although the figure does not show its response to errors, it shows how the data controller handles incoming control characters. The signal *dec_ctl* comes from the decoder and when it is set to high, the received 7-bit character is a command character. When the data controller receives a command character, the output *RX_EN* is set to low. This is done to tell the HDLC controller in the HDLC bus not to read the bit that the data controller just received. However, the received bit is set to *HDLC_out*, which is the *RXD* output of the HDLC bus.

Figures 30 and 31 show the functionality of the data controller when mode commands are received in a uncommonly fast fashion. The purpose of the figures is to illustrate how the data controller responds to mode commands and how it performs when mode commands are received faster than expected in normal operation. In this figure, there is SRAM data

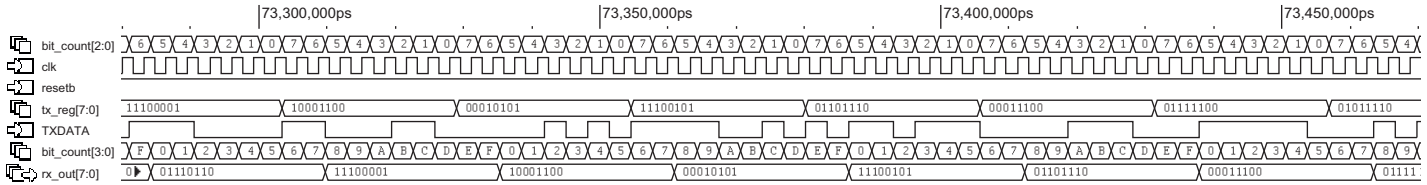


Figure 28. A simulation snapshot illustrating transmitter functionality.

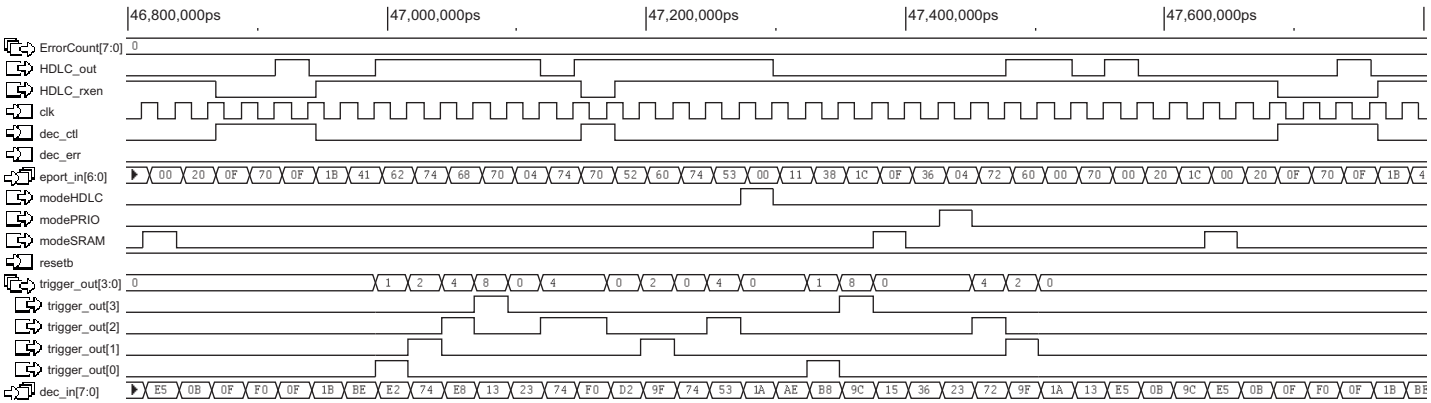


Figure 29. Simulation results describing the trigger command behavior of the data controller block. The triggers are handled by the receiver part of the data controller.

available for transmission.

Figure 30 illustrates how the data controller responds to a faster than normal rate of mode commands. The most important input to observe is the data controller input *eport_in* from the codec and especially the four LSBs. Around 233.30 µs an h'4C is seen and the four LSBs of this character are h'C (b'1100), which corresponds to mode command for *SRAM*. Following the h'4C, it can be seen that the output *modeSRAM* of the receiver block of the data controller strobes to mark the changing of the transmission mode. Furthermore, the transmission mode change is acknowledged with a command character h'3B (SRAM data header) at 233.38 µs. After the data header, the data from the SRAM input *SRAM_in* is sent 7 bits at a time until the next mode command is received.

The second mode command, h'43 for *HDLC* (four LSBs are b'0011), arrives after 233.40 µs. This makes the receiver part strobe the *modeHDLC* and the mode command is acknowledged at 233.52 µs with the command character h'3D (HDLC data header). At the same moment, the mode command h'46 (LSBs are b'0110) is received, priority mode is indicated and the *modePRIO* is strobed. This leads to unexpected behavior because switching to priority mode, and later to *SRAM* mode, results in three SRAM headers being sent. At the bottom of the figure, there is a register *state* that indicates the state that the transmitter part of the data controller is in. The state for *SRAM* is h'2, the state for *interrupt* is h'4, and the state for *HDLC* is h'8 . From the register *state*, we can see that the transmitter part goes to the state *HDLC* around 233.50 µs. After the state *HDLC*, the transmitter part switches quickly to *SRAM* and again to the state *interrupt*. During this, the transmitter part sends a header and a character. Next, the transmitter part enters the *SRAM* state again to send a header and characters. This is followed by an *SRAM* mode command at 233.60 µs that is acknowledged correctly. As a result, there is one extra SRAM data header sent. The reason for this is that the state machine sends an extra SRAM header when it is sending SRAM data in priority mode when the state is switched to SRAM data only.

The necessity of the extra header is questionable, but it can be used to see if the mode command was received by the data controller and if the transmission mode was changed accordingly. However, it is a simple detail to modify it to not send an SRAM header if the state machine is in priority mode and has been sending SRAM data. It is a matter of adding an 'if' condition so that an SRAM header is not sent when the current mode is *PRIO* and the state is *SRAM*. In any other case, send the header.

Despite the extra header presented in Figure 30, the data controller is functioning as it is designed to. More importantly, the data transmission is not interfered with because of this. The bandwidth efficiency of the E-Link decreases slightly because of this feature, but the data

is sent the right way; if the mode changes, a header is sent and it is followed by corresponding data.

In Figure 31, the mode commands arrive at a faster rate than in the previous figure. Furthermore, this time there is no SRAM data available to be sent. As can be seen in the figure, the commands arrive as a burst that begins at 235.54 µs and finishes by 235.85 µs. The first mode command is for priority mode and this is acknowledged with an HDLC header (h'3D) at 235.60 µs. The second header that is sent after the h'3D is the SRAM header h'3B at 235.65 µs. At this point, the *HDLC* mode command received between the priority and *SRAM* mode commands has been ignored.

As seen in Figure 31, the data controller cannot keep up with the extremely fast mode changing rate. This, however, is not normal operation of the system because there should not be a need for mode change on every clock cycle. Furthermore, by observing the output *eport_out*, it can be seen if the data transmission is executed correctly. Data sent follows a corresponding header and as such, the receiving E-Link master on the other side of the E-Link will be able to recognize the source of the data.

4.3.2 Error Handling

One of the simulated features is how the E-Link interface responds to possible decoding errors. Once the data controller is notified by the codec of an erroneous byte, the data controller reports this to the E-Link master by sending the command character h'37 (b'00110111). Because of the limited nature of the error detection of 7B8B encoding, it is not expected to receive very many errors during normal operation. In the test bench setup of this simulation, the E-Link master sends data containing bytes that result in an error. The 7B8B encoder cannot produce these erroneous bytes and therefore the data sent from the test bench is not encoded.

Figure 32 illustrates the error handling capabilities of the data controller. In the figure, the input *dec_err* indicates an error in the received byte when it is set to high. For instance, one error is indicated by the codec at 1.14 µs and the error is reported to the E-Link master by sending the command character h'37 at 1.20 µs.

Also, a feature worth paying attention to is that the value of external register *REG_ErrorCount* is incremented by one for every error that occurs. If an error occurs, HDLC data from the E-Link is not relayed to the HDLC controller. In fact it can be seen from the input *eport_in* that the decoder gives a zero sequence as an output when an error happens in decoding. The

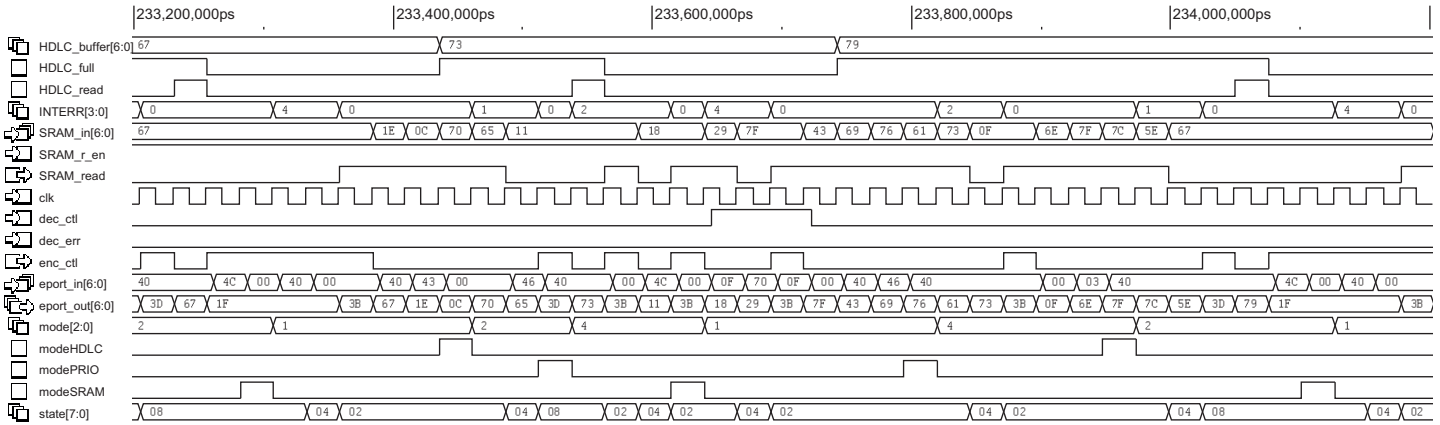


Figure 30. Simulation results illustrating the responses to the mode commands (SRAM data available).

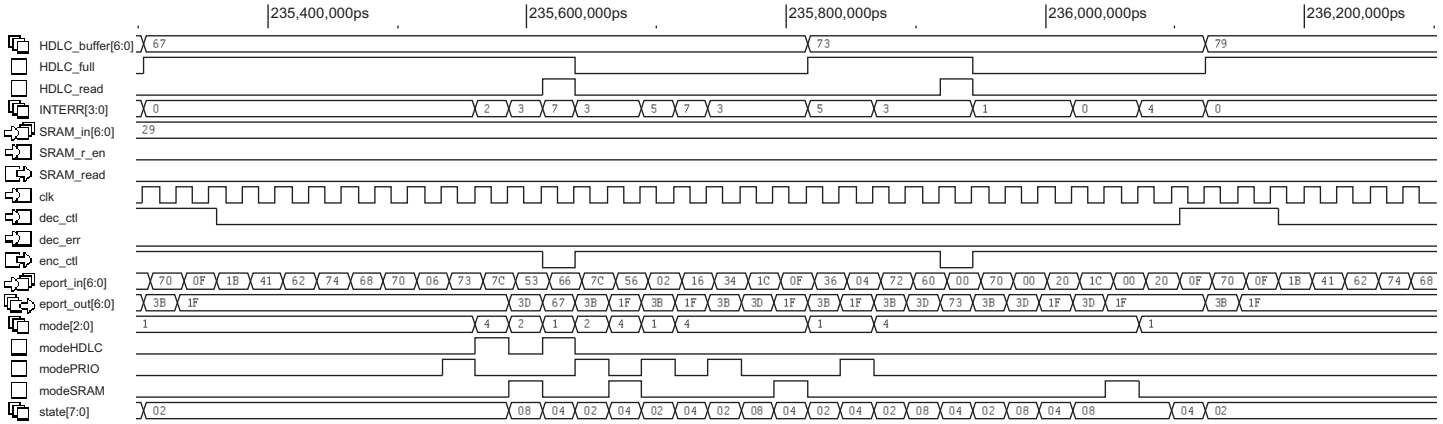


Figure 31. Simulation results illustrating the responses to the mode commands (SRAM data not available).

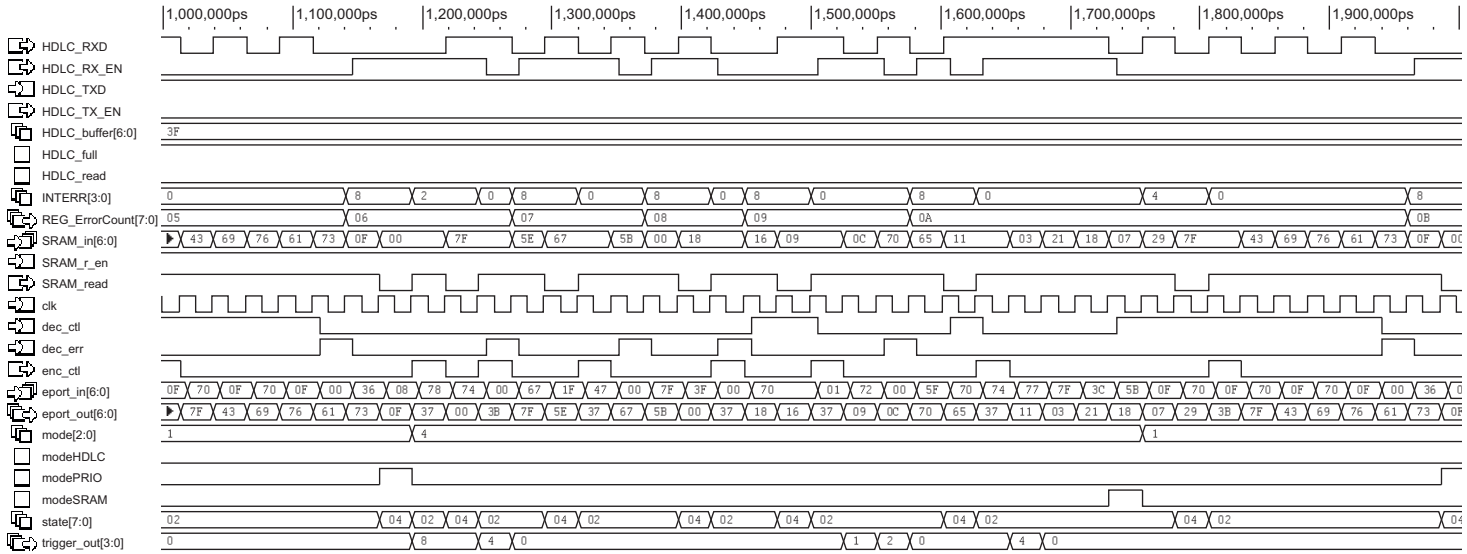


Figure 32. Simulation results of the data controller receiving a lot of errors from the decoder.

bit is set to the output *RXD*, but output *RX_EN* is held low to prevent the HDLC controller from reading the bit, which would be zero anyway. In Figure 32, six errors are reported by the codec and five command characters are sent. One command character is omitted because its location is outside the simulation snapshot.

Error handling and mode changes are the only functionalities of the data controller that report back to the E-Link master. And because the error handling produces error reports to be sent to the E-Link master, it is necessary to be able to see if error reporting interferes with data transmission. In the simulation snapshot, the data controller is in priority and *SRAM* modes, and thus no HDLC data is sent. The data transmission functions the way it is meant to regardless of error reporting. The command characters sent are put in between the headers and SRAM data, thus not replacing data characters with the command characters. Simply put, the data reading from *SRAM_in* is interrupted when an error command character or a header is being sent. However, sending the error command characters reduces the effective bandwidth for the data being sent.

Seeing that the error handling has the desired functionality with more than the expected amount of decoding errors, the next step is to test how the data controller will handle incoming data containing almost nothing other than errors. In Figure 33, the errors begin to arrive at 1.15 µs and the burst of errors can be considered to be finished by 1.76 µs. The burst begins with five erroneous bytes that are decoded to zeros by the decoder. Consequently, the data controller sets the output *RX_EN* to low and starts sending the error command characters (h'37). Between 1.15–1.76 µs, a total of 14 erroneous bytes are decoded, but only six command characters are sent. It is obvious that the data controller is not capable of responding to a high rate of errors. However, the data controller's ability to count the number of errors (*REG_ErrorCount*) functions well.

One way to correct this inability to respond would be by implementing a counter that would be the same size as the register *REG_ErrorCount*. This counter would count the number of error command characters sent and once the state machine would enter the state for sending the command characters, it would compare the numbers between *REG_ErrorCount* and the counter. In addition, the state machine would not proceed to the next state unless the compared numbers are equal. It should be taken into account that if the external register *REG_ErrorCount* is reset, then the counter must also be reset. Otherwise, the functionality of the state machine would result in an inability to respond to errors.

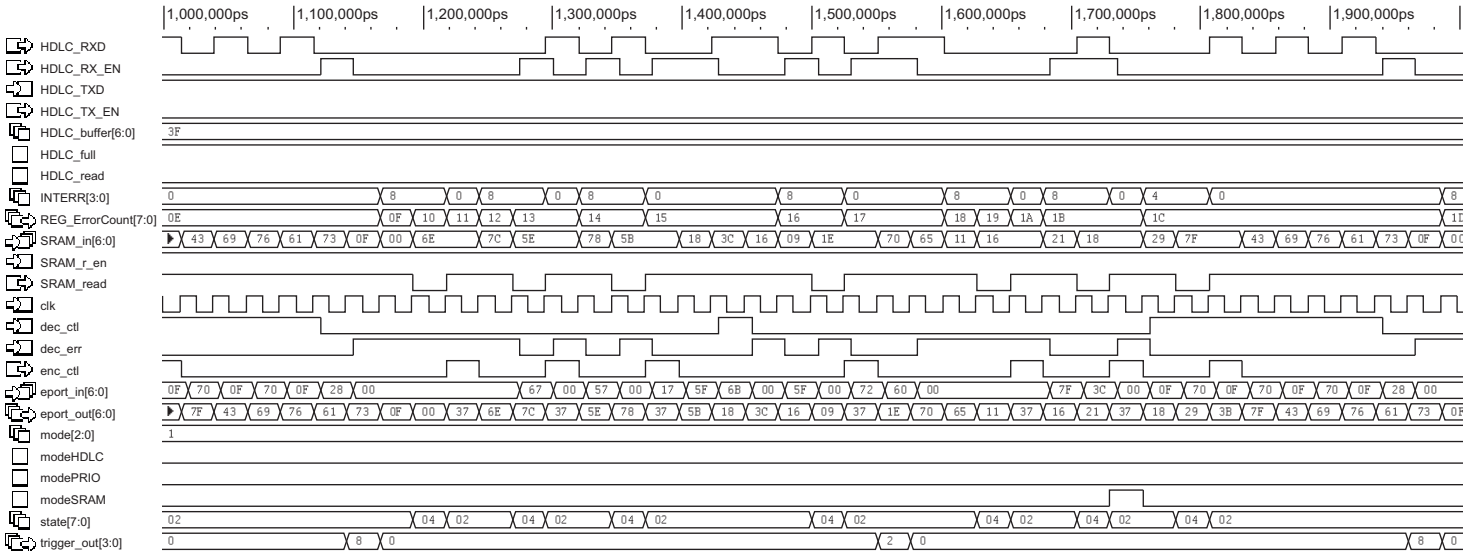


Figure 33. Simulation results of the data controller receiving a burst of errors from the decoder.

4.3.3 Data Transmission Modes

The last things to simulate and study with regards to the data controller are the transmission modes. The simulation results for the priority mode are illustrated in Figure 34. The purpose of priority mode is to send HDLC data if no SRAM data is available, and therefore make use of the E-Link bandwidth more efficient. Simulation results for *SRAM* mode, *HDLC* mode and HDLC buffer are presented later in this thesis.

The setup for priority mode simulations is to have both SRAM and HDLC data available. This way, the typical operation of the data controller in priority mode can be observed. In Figure 34, the simulation snapshot begins with showing typical SRAM data transmission until 24.77 µs when there is no SRAM data available. This is indicated by the data formatter by setting the input *SRAM_r_en* to low. At the same time, the data controller sets the output *SRAM_read* to low to tell the data formatter that it has read the last available character.

At the bottom of Figure 34, the output *enc_out* shows how the data sent to the encoder is encoded and how some of the characters have two possible bytes when encoded and some have only one. The output *rx_data* is the byte received by the receiver in the test bench on its way to be decoded. The output of the decoder is *dec_out* and this shows that the data sent from the data controller is received correctly. Finally there are the decoder outputs *dec_ctl* and *dec_err* indicating a command character or an error.

After setting *SRAM_read* to low, the data controller sends the last character read from input *SRAM_in* (h'29) at 24.77 µs that is followed by an HDLC header (h'3D) and HDLC data (h'3F). This marks the beginning of the HDLC data transmission as the data controller switches to HDLC mode. Subsequent data sent are filler command characters (h'1F) indicating that there is neither SRAM nor HDLC data available. It takes seven clock cycles to fill the HDLC buffer and one more clock cycle for the data controller to read and send another character.

HDLC mode continues until 32.96 µs when the data controller sets *SRAM_r_en* to high as a sign that there is SRAM data available. It takes one clock cycle for the data controller to respond, send an SRAM header, and read the first character. Starting from 33.01 µs, the data controller sends only SRAM data.

The simulation results for *SRAM* mode are presented in Figure 35. In this simulation, only SRAM data is being sent and HDLC data is disregarded. In the beginning of the simulation, the data controller is sending the 7 bits of data that the data formatter provides to the input *SRAM_in*. The data is read from *SRAM_in* and sent to *eport_out*; just like in priority mode.

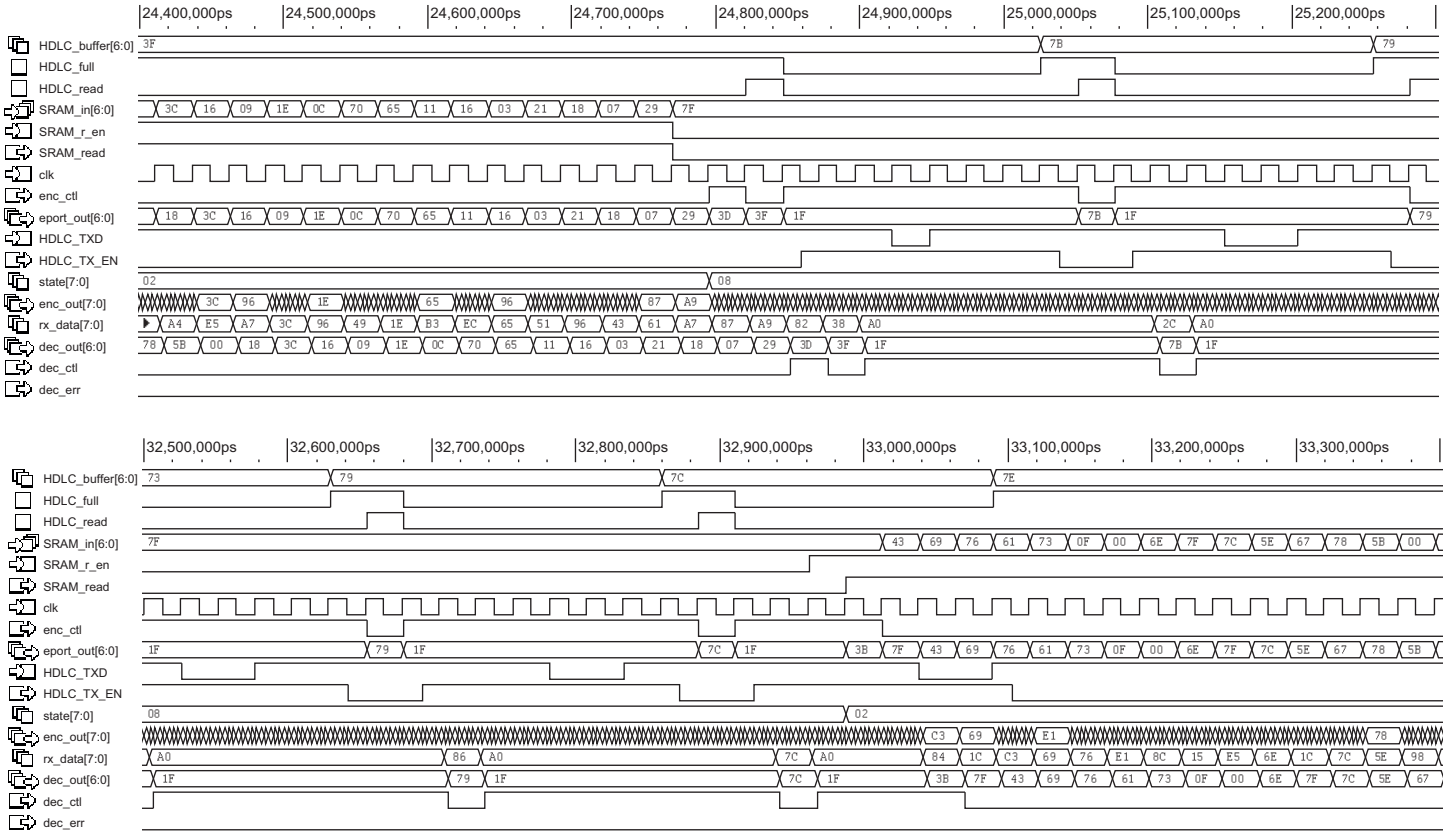


Figure 34. Simulation results illustrating priority mode. At first there is data available in the SRAM, and after a while the SRAM is empty. Later at 32.96 µs, the SRAM has data available again.

Furthermore, like previously, the figure also presents the outputs of the test bench side so that the sent and received data correlate with each other.

The data controller sends SRAM data until 24.77 µs when the data formatter sets the output *SRAM_r_en* to low. At the same moment, the data controller sets *SRAM_read* to low to tell the data formatter that the last available character was read. On the next clock cycle, the data controller sends the last character (h'29) and it is followed by filler control characters (h'1F).

The filler characters are sent until the SRAM data becomes available again at 32.96 µs and when the data formatter sets *SRAM_r_en* to high. The data formatter responds to this in one clock cycle, reads the available character h'7F, and sends it to the codec to be encoded. Compared to priority mode, the *SRAM* mode is one clock cycle faster in responding to available data. The character that has been sent is received in the test bench after decoding at 33.05 µs. After SRAM data is available again, the data controller continues to transmit the data from the data formatter to the encoder.

4.3.4 HDLC Buffer and HDLC Transmission Mode

In Figure 34, HDLC data transmission in priority mode was briefly covered. However, because the functionality of *HDLC* mode is closely related to the HDLC buffer, the simulation results of both of them are presented in Figure 36. In this simulation, the HDLC bus, the HDLC buffer, and the data controller transmitter block are being observed.

The simulation begins with the data controller inputting a 7-bit character from the HDLC buffer output *HDLC_buffer* into the output *eport_in* at 2.01 µs. At the end of the clock cycle, the input *HDLC_full* and the output *HDLC_read* are set to low to indicate that there is no data available at the moment. This initiates data reading from the HDLC bus. A new 7 bits are read from the input *HDLC_TXD* (previously *TXD*) to the shift register buffer when the output *HDLC_TX_EN* (*TX_EN*) is set to high. *HDLC_TX_EN* stays high for seven clock cycles and by then the buffer is full and *HDLC_TX_EN* is set to low. Once the buffer is full, *HDLC_full* is set to high, followed by the data controller reading the character, sending it to the codec, and strobing *HDLC_read*.

4.4 Reset and Clock Functionality

The functionality of the E-Link interface is synchronous to the 320 MHz clock and it will not start functioning if the reset is low. This is presented in Figure 37 in which the *resetb* signal

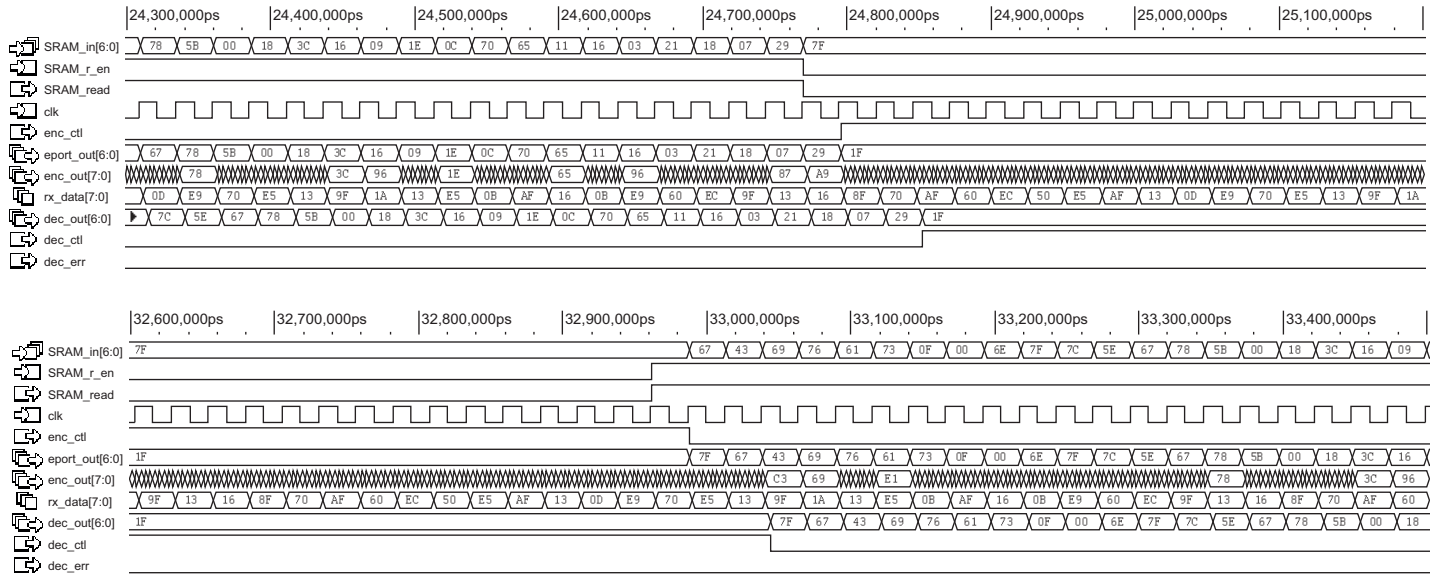


Figure 35. Simulation results illustrating the functionality of the *SRAM* mode.

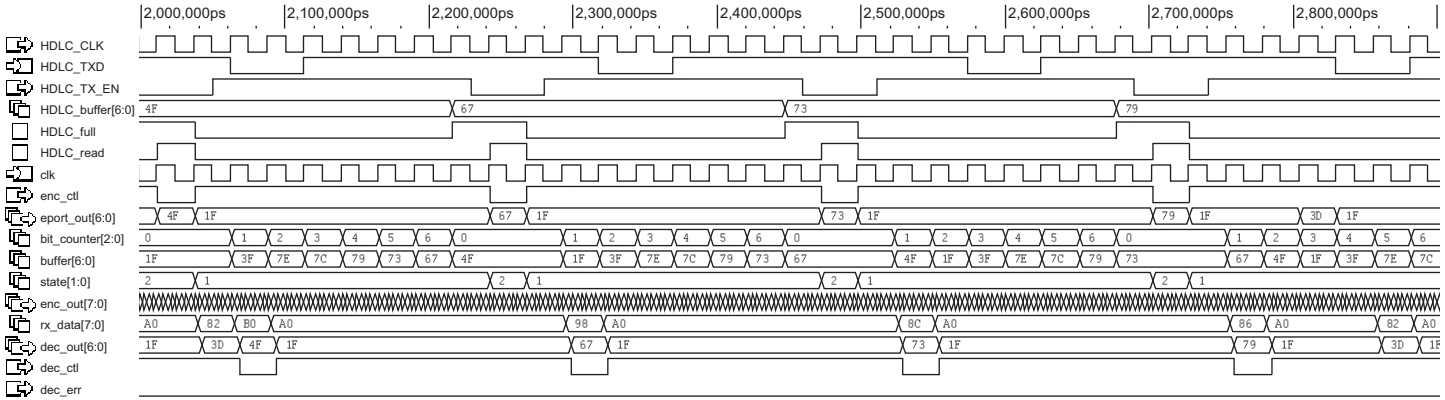


Figure 36. Simulation results presenting the operation of both the HDLC buffer and the *HDLC* mode.

is kept low during the simulation. The simulation snapshot shows that all the signals, with the exception of *RXCLK*, are static and as such, they register values that remain invariable. Therefore, the system is functioning as intended when it comes to reset behavior.

Another simulation snapshot is shown in Figure 38. In this simulation, the clock signal is omitted, thus it has the value of 0. The reset signal is set to high at 0.10 μs, but it does not have any effect on the interface functionality. Therefore, it is safe to say that the E-Link interface is synchronous to the clock and no asynchronous functionality is included that would be observable in the inputs and outputs. Normal behavior is achieved by applying both the clock *RXCLK* and by setting the reset signal (*resetb*) to high.

Figure 37. A simulation snapshot of the E-Link interface when the reset signal *resetb* is not set to high.

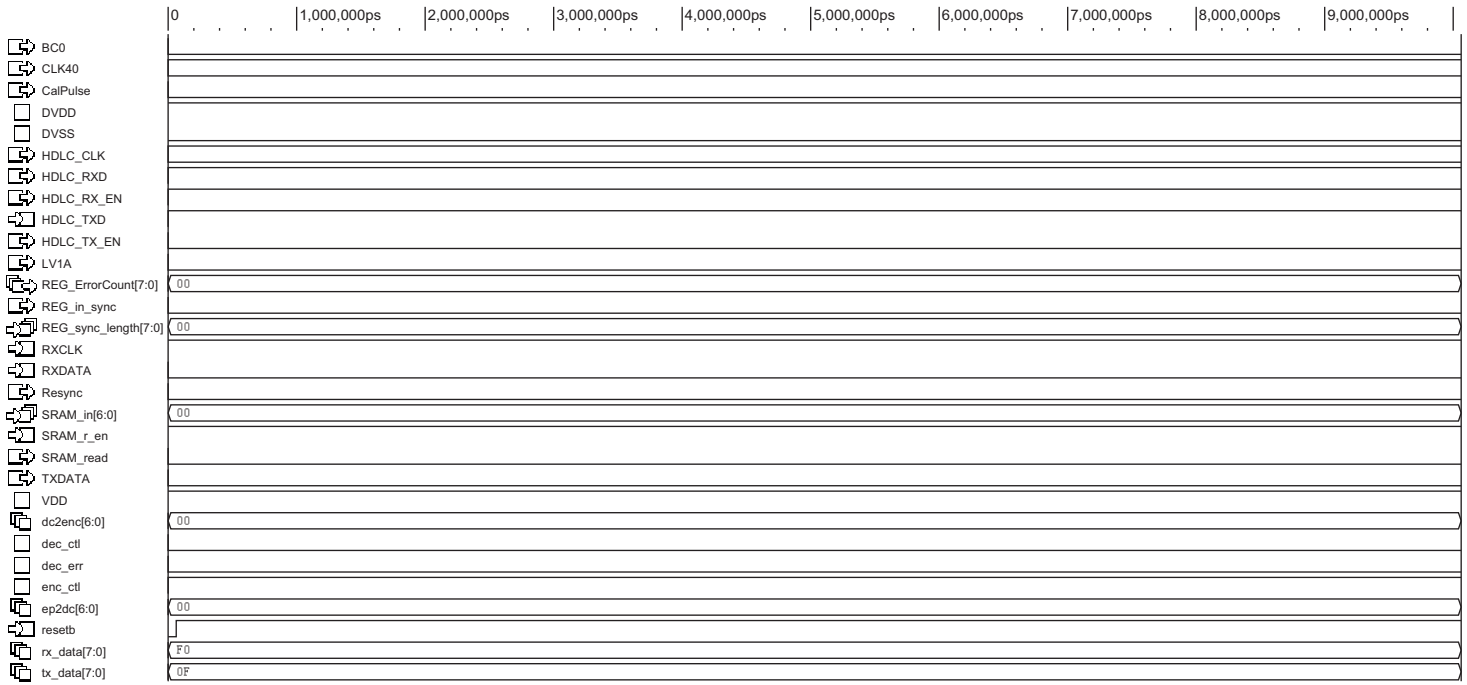


Figure 38. The simulation results presenting E-Link interface functionality when no clock signal is applied to the system.

5 Summary

The starting point of the study was to investigate an E-Port module between read-out electronics and a data transmission link. This module would be responsible for both the inbound and outbound data transmission of a front-end chip. These front-end chips are planned to be used in the GEM detectors of the CMS detector at CERN. The implemented module has interfaces to both a front-end chip and a data transmission link; these chips set the requirements for the functionality of the module. One requirement is to use the bandwidth of the E-Link as efficiently as possible. A model implemented in the Verilog hardware description language was done for the E-Link interface, and the model was simulated using chip design software by Cadence.

The work was begun by studying the technologies that would be needed. This process started with the needs of the E-Link interface and progressed towards the SRAM and HDLC interfaces. It was decided that 7-bit-to-8-bit encoding would be used for the data transmission because data outbound from the front-end chip is defined so that it is split into 8-bit bytes. For this reason, the more popular encoding method 8B10B was not suitable for the design. Additionally, the choice in favor of the 7B8B was affected by the fact that the encoding module had already been implemented in the past. 7B8B encoding translates a 7-bit character into an 8-bit byte so that a long data stream will be DC balanced as much as possible. The bytes encoded in 7B8B can be decoded back to the original characters.

Making the choice to use 7B8B encoding greatly affected the design of the E-Link interface. The encoding method allows 8-bit command characters to be used; 8-bit command characters cannot be decoded into data. Therefore, several of these command characters were utilized to synchronize the E-Link interface and report its status. The receiver part was designed to synchronize with two particular consecutive command characters that were used to define the phase of the generated 40 MHz clock signal. This 40 MHz clock signal is generated from the 320 MHz clock signal provided by the E-Link bus. Furthermore, a bit misalignment is avoided by synchronizing with inbound data. After reading the serial data to its shift register, the receiver sends the data in parallel mode to the 7B8B decoder, thus acting as a SIPO shift register. The implemented transmitter is a simple PISO shift register that reads data from the output of the 7B8B encoder and sends the data to the E-Link.

Seen from the direction of the E-Link, the receiver and transmitter parts are followed by the 7B8B codec that encodes data into a suitable form for transmission and then decodes data back into the original form. Inbound and outbound data is handled in the data controller, which makes up the largest part of the logic. The desired functionality is to send data from

two sources: the tracking data of the GEM detector from the SRAM and slow control data from the HDLC bus. Data from the SRAM comes through the data formatter that tells the data controller when data is available to be sent, and if no data is available in the SRAM, then data from the HDLC is read instead. Three different transmission modes for controlling the data transmission were implemented; the transmission modes can be used to decide where data is read from. The transmission modes are: SRAM data only, HDLC data only, and priority mode. Priority mode primarily sends SRAM data, but switches to HDLC data if no SRAM data is available.

All three transmission modes were implemented to allow either SRAM or HDLC to be ignored when necessary. The third mode was implemented so that the other two modes would have the option to work as a default mode that allows access to both of the data sources. An HDLC buffer was done for HDLC data transmission to buffer serial data through a shift register because it is imperative to read 7 bits before encoding. This is absolutely necessary for maximizing bandwidth efficiency.

6 Conclusions

The E-Link interface allows data transmission between the front-end chip and external systems at 320 Mbps. To synchronize the E-Link interface of the front-end chip using bytes that are not understood as data, the interface uses 7B8B encoding. The command characters, made possible by the encoding, are also used as data headers for outbound data transmission. As a result of using the encoding, the effective data rate of the E-Link drops by 12.5% to 280 Mbps.

Based on the simulation results presented in this thesis, the next generation of front-end chip can utilize E-Link. The implemented Verilog code was done so that its development can be continued from its current state and ultimately, a chip layout of the interface can generated. It was observed from the results that the implemented E-Link interface functions as intended and that the desired requirements for functionality were achieved.

When analyzing the simulation results, two ways to improve the current functionality of the interface were identified. The first thing to improve is the priority mode of the state machine of the data controller. Currently, the state machine changes to the state *txHDLC* if there is no available data in the SRAM. To make the SRAM data reading faster by one clock cycle, the state machine should enter the state *txHDLC* only if there is data available in HDLC buffer.

The second thing to be improved on would be the buffer for HDLC bus. At the moment, 7 bits are received in 8 clock cycles. This is one clock cycle too many: a one clock cycle long delay affects the bandwidth efficiency of the E-Link. Functionality would be improved if the buffer is allowed to read data even if it is full and the previously read data is stored in the output. If the buffer is full, which means that the data from the output is not read, the buffer would pause when 7 bits are read. This would allow an efficient use of both the register and the output. In terms of functionality, this would be seen as a rapid data transmission of 14 bits when the data controller changes the state from *txSRAM* to *txHDLC*. Also, the buffer fill-up time could be reduced to 7 clock cycles.

With regards to future development, the code has to be optimized and the improvements mentioned in the previous paragraph should be implemented. With optimization, the objective is to implement the described functionality by using the most beneficial amount of microelectronic components possible. Additionally, the logic of the interface should be triplicated to make it radiation hard so that it can withstand the high radiation of the environment. This means implementing all of the functionality three times and determining the value of the output by a majority vote.

References

Alexander, J.R. and Nagra, A.S.T. (1979), "Transformation of Binary Coded Signals into a Form Having Lower Disparity," Patent specification.

ALICE (2007), "ALICE TPC Front End Electronics," URL http://ep-ed-alice-tpc.web.cern.ch/ep-ed-alice-tpc/index.htm. Accessed 22.1.2013.

Aspell, P. (2006), "VFAT2 Operating Manual," URL http://totem.web.cern.ch/Totem/work_dir/electronics/totelwork_files/PDFgeneral/VFAT2Manual.pdf. Accessed 23.1.2013.

Aspell, P., Anelli, G., Chalmet, P., Kaplon, J., Kloukinas, K., Mugnier, H., and Snoeys, W. (2007), "VFAT2: A front-end system on chip providing fast trigger information, digitized data storage and formatting for the charge sensitive readout of multi-channel silicon and gas particle detector," in *Proceedings of TWEPP-07, Topical Workshop on Electronics for Particle Physics*, pp. 292–296.

Aspell, P., Avati, V., Bialas, W., Kaspar, J., Kopal, J., Petäjäjärvi, J., Radicioni, E., Rouet, J., Snoeys, W., and Vichoudis, P. (2008), "The VFAT production test platform for the TOTEM experiment," in *Proc. Topical Workshop on Electronics for Particle Physics 2008*, pp. 544–548.

Aspell, P., De Gaspari, M., França, H., García García, E., and Musa, L. (2012), "Super-Altro 16: a Front-End System on Chip for DSP Based Readout of Gaseous Detectors," in *IEEE Transactions on Nuclear Science*, draft.

Aspell, P., França, H., García García, E., De Gaspari, M., Mager, M., Musa, L., Rehman, A., and Trampitsch, G. (2010), "Description of the SAltro-16 chip for gas detector readout," .

Bhasker, J. (1998), *Verilog® HDL Syntesis: A Practical Primer*, Star Galaxy Publishing.

Bonacini, S. (2008), "Verilog HDL code for a 7B8B codec," .

Bonacini, S., Kloukinas, K., and Moreira, P. (2009), "E-link : A Radiation-Hard Low-Power Electrical Link for Chip-to-Chip Communication," in *Topical Workshop on Electronics for Particle Physics*, pp. 422–425.

Brooks, R. (1980), "7B8B Balanced Code with Simple Error Detecting Capability," in *Electronics Letters*, vol. 16, pp. 458–459.

CERN (2013), "About CERN," URL http://home.web.cern.ch/about. Accessed 22.4.2013.

CMS Experiment (2013), "What is CMS," URL http://cms.web.cern.ch/news/what-cms. Accessed 29.4.2013.

De Gaspari, M. (2013), “A conversation about technical details of S-Altro chip,” 22.1.2013.

Evans, L. (2009), *The Large Hadron Collider: a Marvel of Technology*, Fundamental Sciences, EPFL Press, Lausanne.

GDD (2011), “The Gas Detectors Development Group,” URL http://gdd.web.cern.ch/GDD/. Accessed 10.1.2013.

Hewlett-Packard (2011), “Serial ata technology: Technology brief, 4th edition,” URL http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00301688/c00301688.pdf. Accessed 13.3.2013.

Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, NEC Corporation, ST-NXP Wireless, and Texas Instruments (2008), “Universal serial bus 3.0 specification, revision 1.0,” URL http://www.usb.org/developers/docs/. Accesssed 13.3.2013.

IEEE (2008), “Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications,” URL http://standards.ieee.org/about/get/802/802.3.html. Accessed 13.3.2013.

Kaur, G. (2011), *VHDL: Basics to Programming*, Pearson Education India.

Lefevre, C. (2008), “LHC: the guide,” http://cds.cern.ch/record/1092437?ln=en. Accessed 25.4.2013.

Meggyesi, Z., van der Bij, E., and McLaren, R. (2004), “FPGA Design in the Presence of Single Event Upsets,” CERN publication service.

Reese, R.B. and Thornton, M.A. (2006), *Introduction to Logic Synthesis using Verilog HDL*, Morgan & Claypool, 1st edition.

Sauli, F. (1997), “GEM: A New Concept for Electron Amplification in Gas Detectors,” in *Nuclear Instruments and Methods in Physics Research*, vol. A386, pp. 531–534.

Sharma, A. (2012), “A GEM Detector System for an Upgrade of the CMS Muon Endcaps,” Intended for CMS internal use and distribution only.

Simpson, W. (1994a), “PPP in HDLC-like Framing,” RFC: 1662.

Simpson, W. (1994b), “The Point-to-Point Protocol (PPP),” RFC: 1661.

Xilinx (1995), “Appendix A: Accelerate FPGA Macros with One-Hot Approach,” in *HDL Synthesis for FPGAs Design Guide*, URL http://www.xilinx.com/txpatches/pub/documentation/xactstep6/hdlsynth.pdf. Accesssed 9.4.2013.