Irradiation Resistivity and Mitigation Measurement Design for Xilinx Kintex-7 FPGAs

Master Thesis in Microelectronics

Lukas On Arnold

Institute of Microelectronics, School of Engineering, University of Applied Sciences and Arts Northwestern Switzerland, Windisch, Switzerland

Institute for Microelectronics, School of Engineering, University of Applied Sciences of Eastern Switzerland, Rapperswil, Switzerland

March 16, 2015

Home University:	Institute of Microelectronics, School of Engineering **University of Applied Sciences and Arts Northwestern Switzerland** Steinackerstrasse 5, Windisch, Switzerland
Research University:	High Energy Physics Group, Cavendish Laboratory, Department of Physics **University of Cambridge** JJ Thomson Avenue, Cambridge, United Kingdom
Experiment:	LHCb, Ferney-Voltaire, France
Location:	European Organization for Nuclear Research (CERN), Geneva, Switzerland
Project Supervisor:	Dr Stephen Wotton, Cambridge
Institutional Supervisor:	Michael Pichler, Windisch
Expert:	Prof Dr Markus Hufschmid, Windisch

CERN-THESIS-2015-372
27/04/2015

Abstract

For the upgrade in the Ring-Imaging Cherenkov Detector (RICH) in the LHCb detector, Xilinx Kintex-7 FPGAs are projected to be taken into operation as data acquisition devices for the measurement taken by the photosensors. They will be placed in the immediate surrounding area of the particle collision and hence have to face severe radiation. In order to have a prediction on the working reliability of the devices, irradiation tests will be performed to determine upset probability and effectiveness of mitigation techniques.

The thesis deals with the question of designing an irradiation resistivity and mitigation measurement FPGA, its simulation, implementation and verification. To ensure reliable communication during radiation tests between the device-under-test and the DAQ module despite radioactive environment and limited cable resources, a custom protocol is examined and defined.

Acknowledgements

I want to express special thanks to Stephen Wotton, who was supervising me during the project. He has provided me with strong expertise in the field of Microelectronics and with great help and inspiration for the project. I want to thank Michael Pichler for advising me on FPGA programming and Prof Markus Hufschmid for his valuable contributions in the field of communication theory, which have been of vital importance. Also, I want to thank Prof Val Gibson and the Cambridge group for supporting me and giving me the opportunity to work on this very interesting project in their group. I am grateful to everyone else whom I was working with for the experience gained.

Contents

List of Figures

List of Tables

1 Introduction

The LHCb experiment, based at CERN, Geneva, Switzerland, measures parameters of proton-proton collisions produced by the Large Hadron Collider (LHC). It aims at gaining information about Charge Parity violations in b-hadron interactions, which can lead to an understanding of Matter-Antimatter asymmetry in the universe. Various sub-detectors are used to track and identify the particles produced at and immediately after the collision.

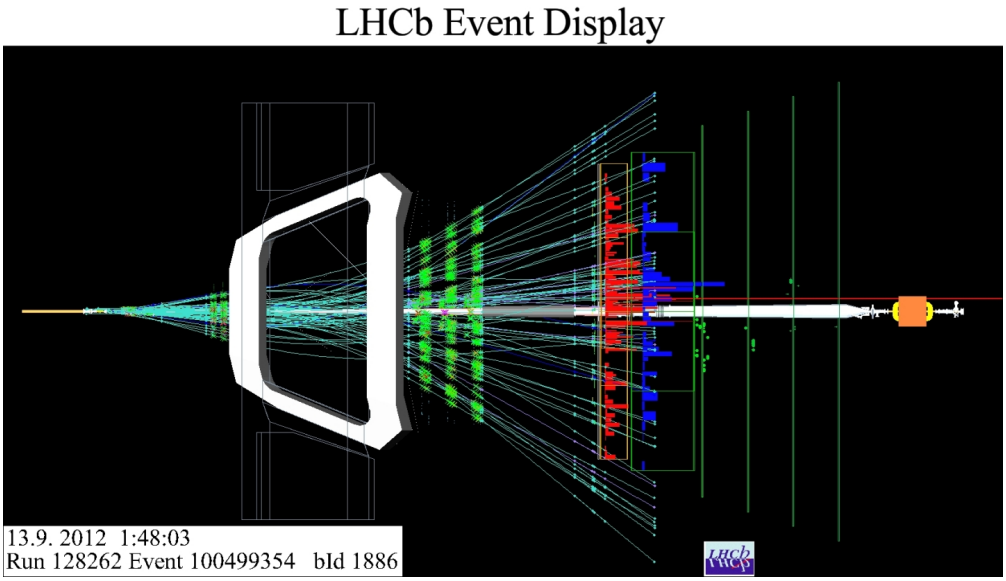

Figure 1.1: Reconstructed event of a proton-proton collision detected by the LHCb detector. Ring-imaging Cherenkov detection comes into action before and after passing the magnet(*white*). *Source:* CERN.

The identification of electrically charged particle is done by the Ring-Imaging Cherenkov Detector (RICH). It uses the Cherenkov effect to derive the particles' velocity and by that to ascribe them to a particle type. Cherenkov radiation is emitted due to the fact that particles can travel faster than light in non-vacuum (what they cannot obviously in vacuum); in this case, an electromagnetic cone is formed at the travelling particle, similar to the Mach wave formed at an object travelling faster than acoustic waves. Two RICH detectors are used within LHCb, one for identifying low- and high-momentum charged particles each. Since the particle travelling paths should not be interrupted by detectors, photons emitted as Cherenkov radiation are reflected by surrounding mirrors; the deflected Cherenkov photon emission is detected in RICH by highly sensitive photomultiplier arrays which are triggered by reflected photons.

Field-Programmable Gate Arrays (FPGAs) are used for reading out the produced data due to their reconfigurability, reliability, high flexibility in functionality and their low cost. Data are taken from the

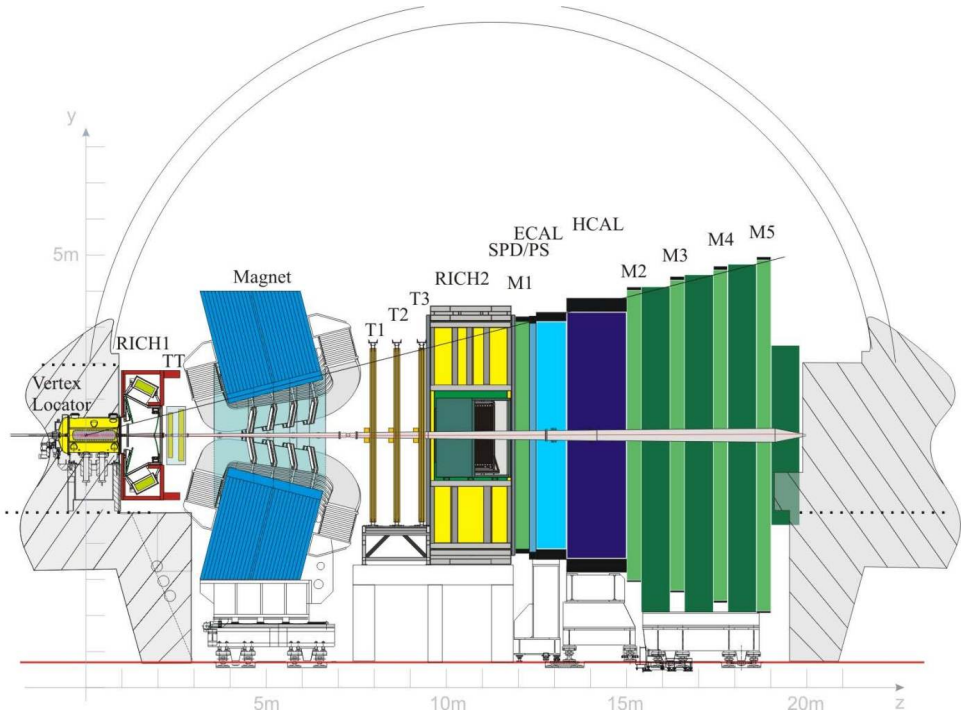

Figure 1.2: Layout of the LHCb detector showing – besides the other parts – the two RICH sub-detectors. *Source:* CERN.

silicon arrays, then multiplexed and handed over to a Multi-Gigabit Transceiver ASIC, which then passes a data stream via optical fibers to the controls outside the detector. Particle collisions cause high irradiation; thus, FPGAs working inside the detector have to be able to withstand high ionizing doses in order to be properly working.

Since access to the detectors is very limited besides the upgrade phases, the FPGAs not only have to meet high performance standards, but also high reliability in time. Thus, verification is required to prove the FPGA types used suitable for the given task. In order to achieve that, the Devices-under-Test (which are in this case, as described subsequently, Xilinx Kintex-7 FPGAs) are put into a test beam where they are exposed to a certain amount of irradiation. This allows to measure the irradiation effects on these devices and by that to form a base to estimate reliability and suitability of the test devices.

In this thesis, a VHDL test design for a Kintex-7 is presented, based upon effect simulation and common mitigation techniques. Whereas the main effort was lead on the practical work carried out at CERN, the written documentation should give a comprehension of the design and of the effects which have to be faced when working in a radiation environment.

1.1 Overview

The first part of the thesis will give an introduction to the methods and structure of the work, whereas the second part documents the design for its usage. Each chapter contains information on both the Device-under-Test, which also is the main task of the thesis, and the communication FPGA, which will be of use to provide radiation-hard communication.

2 General aims and methods

The LHC particles accelerator collides sub-atomic particles after Long Shutdown 1 in early 2015 with an energy up to 14TeV. Peak luminosity of about $5.4\frac{\text{Hz}}{\text{mb}}$ has been reached at the LHCb experiment recently [6].

Luminosity will by increased by a factor of 10 after Long Shutdown 2, which is planned to end in late 2018. This also means, that data production and detector irradiation will be increased in the same order. This makes it necessary to upgrade the readout FPGAs used in the RICH detector to make them suitable for the new – more challenging – operating environment in terms of data volume and irradiation.

2.1 FPGAs in irradiation environment

Radiation environment cause working errors on FPGAs, from which the most are induced by Single Event Upsets (SEUs). SEUs are usually created by transition of a particle (usually neutrons, protons and pions) above a certain energy threshold of about 20MeV [7].

On FPGA devices, SEUs target the configuration bitstream by flipping or altering single bits. Configuration bitstreams generate the connection between the logic cells of the FPGA; thus, SEUs affect connections of the FPGA cells and their functionality themselves [8]. In order to provide faultless operation of the readout FPGA nevertheless, measures have to be undertaken as consequence. On the hard-wired level, industrial radiation-hardened FPGA solutions exist. These devices, however, are much more costly than standard devices, why it is more preferable to use the latter if appropriate reliability can be ensured [9] [10]. Thus, methods (which will be described later on) have to be used, which allow standard FPGAs to work in high-irradiation environment.

To evaluate and implement a system well-adapted on the given feature of the readout FPGA which both guarantees reliability and compromises with scrubbing time loss will be the task of the current thesis.

2.2 Hardware specifications and working environment

2.2.1 FPGA

In the readout electronics system of the Cambridge LHCb group, Xilinx Kintex-7 FPGAs will be used. The Kintex-7 family is a good compromise between performance and cost: like the sister family Virtex-7, it relies on 28nm CMOS technology. On the performance side, the Kintex-7 provides up to 478k logic cells (Virtex: 2M logic cells); Digital Signal Processing (DSP) performance can reach up to 2800GMAC/s, while up to 12.5Gb/s of data can be transceived (Virtex: 5300GMAC/s peak DSP performance and 12.5Gb/s

peak transceiver speed) [11] [5]. This performance – which meets the requirements for the given task – can be achieved while the costs are merely the half of the high-performance VIRTEX family, which makes it favorable to use KINTEX-7 type devices in the specific readout system.

KINTEX FPGAs contain all the features usually available in state-of-the-art FPGAs, including configuration readback and verification, dynamical change between different configuration sources on the fly (MULTIBOOT) and choice of loading devices (e.g. SRAM, JTAG), which will be of particular importance for our task [12].

To achieve full exposure of the board within the test setup, the upper layer of the FPGA is stripped off. It will be placed on a test PCB designed by the Horia Hulubei National Institute for R&D in Physics and Nuclear Engineering in Bucharest, Romania.

2.2.2 Working environment

As mentioned before, the readout FPGAs used by the LHCb Cambridge group are placed in a high-irradiation area. Irradiation exposure usually is given as the exposure to Total Ionizing Dose (TID) and is expressed in Rad [rd], whereas 1rd equals 10nJ. In the readout part, about 30krd TID are estimated for the time being; this amount usually can be handled by implementing fault detection and scrubbing systems into the FPGA [13].

2.3 Irradiation measurements

In order to have a basic quantification of irradiation effects to FPGAs, measurements have to be undertaken. The basic idea is to gain the number of upsets per time unit (and logic cell) ($\frac{\#\text{SEU}}{\text{s}}$) as a function of irradiation dose (TID). Such measurements have been conducted on various FPGA devices, (e.g. ALTERA ARRIA, XILINX VIRTEX), but in the considered literature not on KINTEX-7 FPGAs [8] [9] [13] [14] [15] [16]. Therefore it is important to acquire information about irradiation-resistivity of the KINTEX-7 type.

Irradiation measurements of FPGAs consist of two parts: The measurement of the ionizing dose itself, and the measurements of the SEUs when being exposed to irradiation. Different methods to create a test irradiation environment exist: One can take radioactive material – such as ^{60}Co [9] – or target the FPGA by an accelerated test beam. The first encounters the problem that it mainly generates γ-ray, which does not correspond to the real environment in the detector; a proton test beam would be more realistic.

Parameters of proton test beams – such as particle energy – can be configured. This makes generation of certain TID levels possible; to verify TID, dosimeters can be attached to the FPGA board [13].

For the SEU measurement, serial shift registers can be used, which pipeline a defined data stream. Shift registers are built by serial connections of Flip-Flops (FF). When a SEU occurs, there is a chance that the value registered in the flip-flop changes. As shown in the block diagram in figure 2.1, an input data stream – ideally consisting of equal amount of 0 and 1 – is given as input for the FPGA. The input stream could be generated by Xilinx ChipScope software [17]. This stream then is pipelined through the device and its output compared to the (by the number of Flip-Flops times clock cycle delayed) input. If comparison proves inequality between the value, one can state that an error has occurred.

In order to get a general derivation out of the measurements, one has to know how much logic cells are used within the device and how they are distributed, and how much radiation is targeted at the device.

Irradiation measurements will not be part of the thesis; it is rather the task, to provide an FPGA which contains functionality that significant results can by achieved by irradiation tests.

2.3.1 Test beam environment

Measurements for determination of irradiation resistivity and efficiency of n-modular mitigation techniques are carried out under a test beam, which is a particle beam simulating the environment which has to be faced at the detector's working environment.

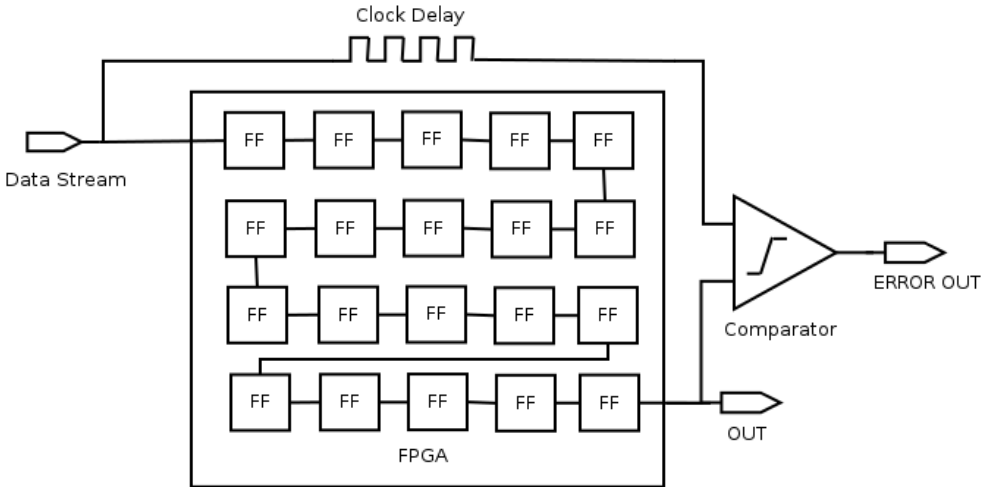

Figure 2.1: Block diagram of a shift register for measuring SEUs.

2.4 Fault detection and correction

Errors caused by irradiation should be overcome by an error detection system with the result that flawless data processing within the FPGA used by the Cambridge LHCb group is granted. The basic concept for

data fault detection is to use redundant logic combined with a voting detection system.

2.4.1 Triple modular redundancy

An established and reliable fault prevention system is Triple Modular Redundancy (TMR) [18]. A logic block of a well-defined size is built redundantly three times within the FPGA. Then, a voter compares the three signals forwarded by the logic blocks – if an error occurs, the voter forwards only the value which appears two times. Obviously, fault risk can be decreased exponentially when TMR is applied.

In the FPGA readout system used by the Cambridge LHCb group, faults could also be handled by reloading the configuration: The voter not only lets the majority signal pass, but informs a voter controller that an error has occurred. As soon as the voter controller notices the fault, it sends a command to the configuration controller to reconfigure the FPGA. The configuration controller then loads a configuration file (or sof-file) from Flash memory. A representation of the system can be seen in figure 2.2.

Restarting the FPGA and reconfiguring it takes some time, usually some seconds. Therefore it is preferable to determine the size of the gated logic blocks by desired fault risk and reconfiguration frequency; also a higher-order redundancy system can be thought about if specifications demand an alternative scrubbing system.

Automatic TMR synthesis tools do exist; however it is preferred to build custom TMR blocks because of their higher adaptability to specific needs and the fact that Xilinx has discontinued their TMR synthesis tool for the 7-Series.

In this thesis, focus will be lead on TMR, and – moreover – N-modular redundancy.

2.5 Communication

A problem faced when working under test beam conditions is the one that error rate is measured by the Device-under-Test itself, thus the error rate measurement blocks and communication blocks are facing irradiation effects as well. This problem has to be solved by designing these blocks in such a way that they are exempt from irradiation effects by applying appropriate mitigation techniques. Since certain parts of communications, like input and output nodes, cannot be subject to Triple or N-Modular Redundancy, a communication protocol has to be used which allows radiation-hard communication. For this reason, a custom protocol has been defined according to the test conditions, from which the two main points are asynchronisity of the two communicating boards (*Device-under-Test* and *Communication board*) and radiation-hardness of the communication.

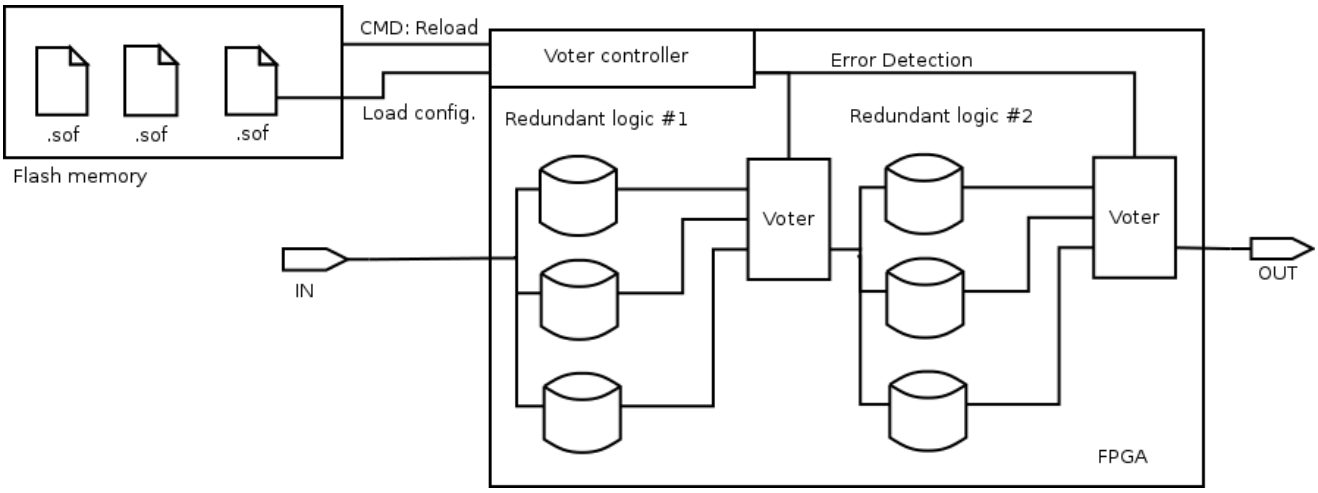

Figure 2.2: Example block diagram of a radiation-hard FPGA design using Triple Modular Redundancy and scrubbing methods.

3 Theory

3.1 Radiation-induced faults on FPGAs

Radiation-induced faults on FPGAs happen primarily due to the electrical disturbances on a silicon layer caused by a charged particles ionization [1] [10]. Particle travelling through silicon layers of a digital electronics device can cause both transient and permanent flipping of the bit held by the Complementary metal-oxide-semiconductor (CMOS) element (see figure 3.2). The different types of errors induced by charged

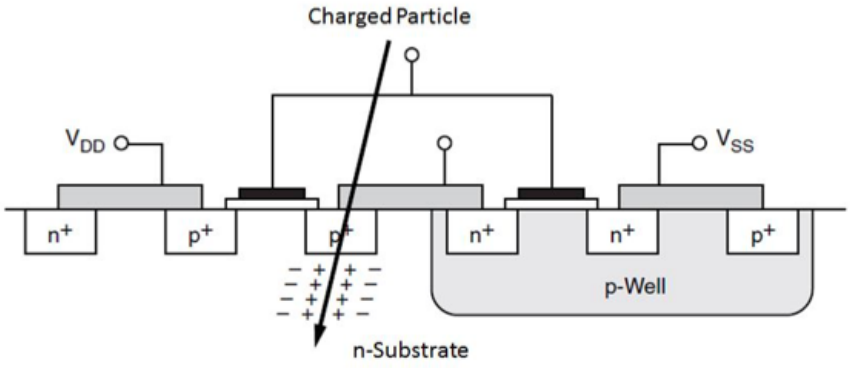

Figure 3.1: Principle of a charged particle travelling through silicon substrate of a metal-oxide transistor. *Source*: [1].

particles passing can be ascribed to one of the following categories: [10]

- *Single Event Upsets (SEU)* denote the non-transient, permanent change of a single bit or multiple bits by a charged particle passing.
- *Single Event Transients (SET)* are, unlike Single Event Upsets, transient changes-of-state of cells.
- *Single Event Latch-ups (SEL)* are high-current states caused by feedbacked bi-stable PNPN structures. They also can be caused by SETs in the case that the changed cell states have propagated into the circuit via Flip-Flops. They only can be removed by power cycling.
- *Single Event Functional Interrupts (SEFI)* are caused by upsets either of an internal memory element or of the circuit which lead to the loss of the FPGA's functionality.
- *Single Event Gate Rupture (SEGR)* is the destruction of the gate oxide by a high electric field during radiation exposure [19].

3.1.1 Device Cross Section

For the estimation of irradiation resistivity of a digital electronics device, its cross section σ is used as a crucial parameter, giving the error probability as a function of particle fluency. It is given by: [2] [20]

$$\sigma = \frac{n_e}{I_p} = \frac{n_e}{\frac{n_p}{A}} \tag{3.1}$$

where:

- $\sigma \left[\mathrm{cm}^2\right]$: Device cross section
- n_e: Number of errors
- $I_p \left[\mathrm{cm}^{-2}\right]$: Particle fluency
- n_p: Number of particles
- $A \left[\mathrm{cm}^2\right]$: Area through which the particles pass

Device Cross Section is used for the number of errors of the whole device (here: FPGA). But, of course, the cross section – or error probability – also can be determined for a single bit within a device. Device Cross Section can be seen as a function of various parameters and can fit to different models [2]. Obviously, the Device Cross Section functions vary for each device and for each type of passing particle.

Weinbull curves are most widely used to represent the cross section functions [10]; they are fit for protons passing as a function of proton energy (E_p) by: [2]

$$\sigma_b(E_p) = \sigma_{\text{sat}} \cdot \left(1 - \exp\left(\left(\frac{E_p - E_0}{W}\right)^s\right)\right) \tag{3.2}$$

where:

- $\sigma_b \left[\frac{\text{cm}^2}{\text{bit}}\right]$: Bit cross section
- $\sigma_b \left[\text{cm}^2\right]$: Limiting or plateau cross section
- E_p [MeV]: Proton energy
- E_0 [MeV]: Onset energy
- W: Width parameter
- s: Exponent parameter

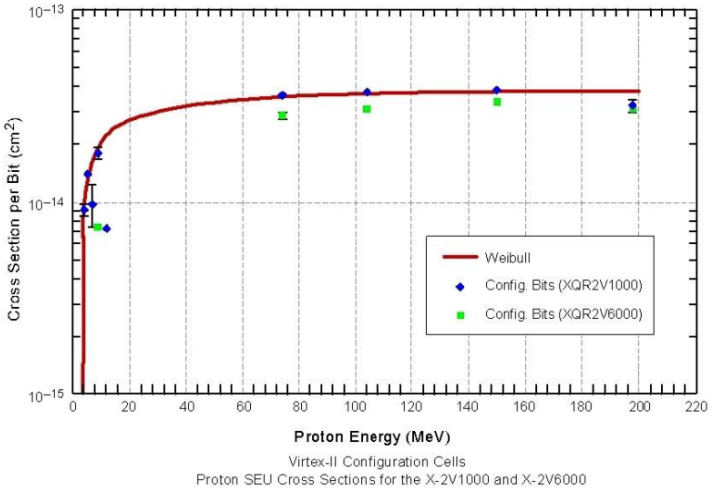

Figure 3.2: Typical Weinbull curve for a XILINX VIRTEX-II FPGA under proton irradiation. *Source*: [2] [3].

Based on equations 3.1, the cross section for Flip-Flops in the Device-under-Test (XILINX KINTEX-7) can be calculated as subsequently follows:

Assuming we know the number of errors n_e for a given time for the non-voting shift register, the size of the shift register S and the number of logic bits (logic size) L of the whole FPGA, its cross section will be,

if particle fluency I_p is known:

$$\sigma_{\mathrm{Kintex-7}} \approx \frac{L}{S} \cdot \frac{n_e}{I_p} \tag{3.3}$$

3.2 Error Mitigation

A major part of the thesis will be to design a measurement device which does not only measure error rates (and by doing so, device cross section), but also measure efficiency and capability of N-modular mitigation techniques, in which the main focus is lead on Triple Modular Redundancy.

N-modular redundancy does not only play a crucial role in electronics application in the field of Particle Physics, but is of high importance in aeronautics, computer science and electrical engineering as well.

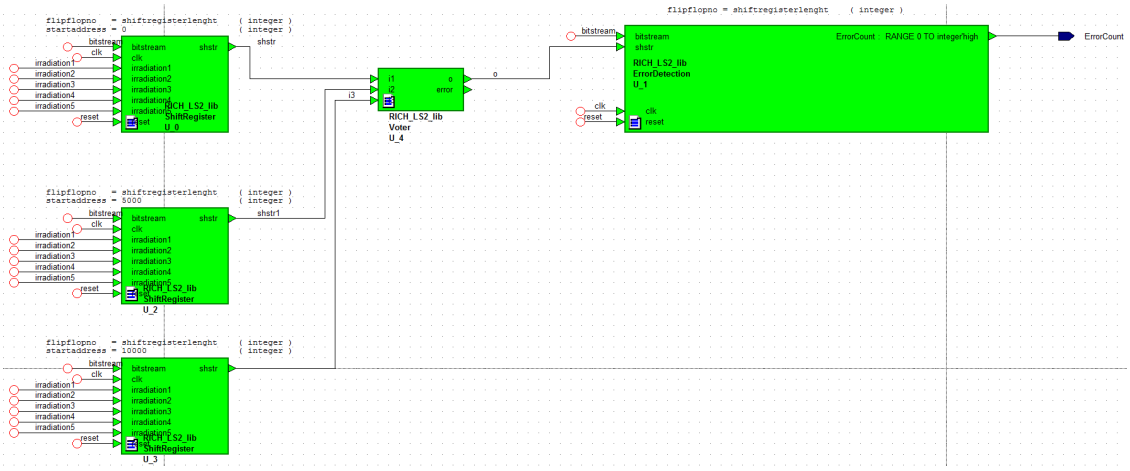

Figure 3.3: Example of Triple Modular Redundancy implementation.

Triple Modular Redundancy follows a simple principle, which is illustrated in figure 3.4: The input signal is sent to three logic blocks of the same type *(left)*, processing the input signal ideally in the same manner. All the signals are sent to a voter block *(middle)*, which verifies that all signals are the same; then it passes the signal on *(right)*. However, if one block happens to be disrupted, the voter will only pass the signals which are handed to the voter block by two of the three signals, thus preventing the disrupted block to take any effect on system functionality.

This principle, of course, is not only limited to three redundant logic blocks, but can be implemented with any number of redundancy which allows majority voting (therefore, the redundancy number has to be odd).

3.2.1 Logical expression

The logical formulation of a Triple Redundancy Voter is given by:

$$D = (A \wedge B \wedge C) \vee (A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C) \tag{3.4}$$

D	**A**	**B**	**C**
1	1	1	1
1	1	1	0
1	1	0	1
0	1	0	0
1	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

Table 3.1: Truth table for triple majority voting.

for inputs A, B and C and output D. This can be reduced, among many versions, to:

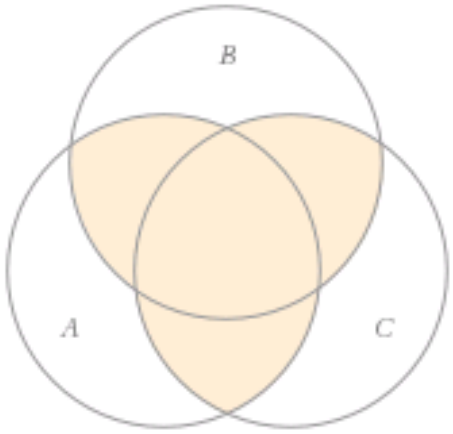

Figure 3.4: Venn diagram of the logical expression for triple majority voting.

$$D = (A \wedge B) \vee (B \wedge C) \vee (A \wedge C) \tag{3.5}$$

Similar voting structures can be easily calculated for N-modular redundancy voter when respecting the principle that the signal state which is in majority is identical with the output signal state.

3.2.2 System reliability

Error mitigation efficiency and capability of N-modular redundancy can be influenced by mainly two parameters: Redundancy order and size of voted logic.

Redundancy order N describes how many times a logic block should be multiplied and ideally is an odd number in order to enable majority voting. Size of voted logic in relation to logic size of the device gives the frequency of majority voting, i.e. after how many logic cells majority voting is performed.

The Data fault rate DFR as a function of sub-system (logic element) reliability σ and size of voted logic A is approximated by the equation: [21] [22]

$$DFR = \sum_{i=\frac{N+1}{2}}^{N} \binom{N}{i} (\sigma \cdot A)^i \cdot (1 - \sigma \cdot A)^{N-1} \tag{3.6}$$

We see that DFR behaves quite linearly to the size of voted logic A, but decreases exponentially with the increase of redundancy order N (see also figure 3.5).

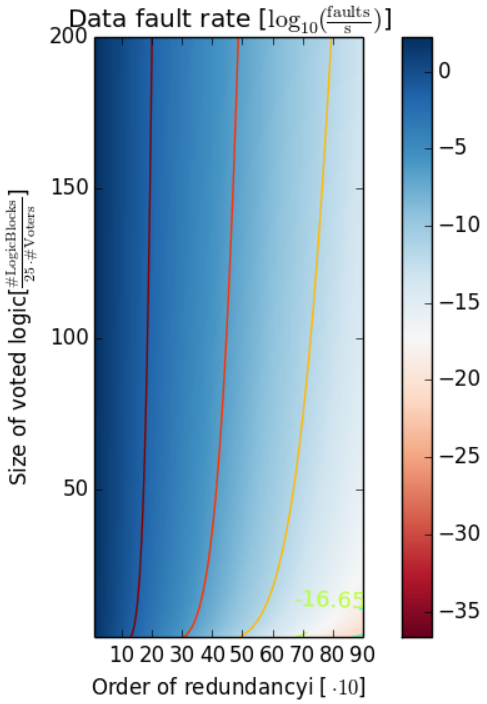

Figure 3.5: Data fault rate as a function of redundancy order and size of voted logic. Data fault rate corresponds to the colour scheme. A more comprehensive plot of the same function can be seen in figure 4.3.

The maximum redundancy order MRO is constrained by how many logic blocks are available on a device (S_A) and how many logic blocks are taken by the basic, non-redundant logic to be voted (S_B).

The order MRO can be determined calculating the maximum redundancy number MRN first:

$$MRN = \frac{S_B}{S_A} \tag{3.7}$$

And then, after having obtained the maximum redundancy number by the simple formula 3.7, applying – in

order to get an odd natural number:

$$MRO = \begin{cases} \text{round down } MRN, & \text{if this value is odd} \\ \text{round down } MRN - 1, & \text{otherwise} \end{cases} \tag{3.8}$$

3.3 Communication

In this chapter, an introduction into the basics of how the asynchronous communication protocol should be given. A general introduction on the basics is held before coming to the specific needs of our task.

3.3.1 General

Speaking about communication in the most general way, we can describe it as a method delivering information I from an information source A to an information sink B, where the information received at the information sink B (denotation: I') should be as close as possible to the information initially sent by the information source A (denoting: I^*).

Mathematically, information can be described by a status vector $\vec{I}$; of course, the status space cannot be defined generally, but is spanned by the specific information. For example, a detector for particles might span a status space by a vector acting in the dimensions $(\text{Velocity}, \text{Charge})$. The distance between the vector of the initial vector $\vec{I^*}$ and the received status vector $\vec{I'}$ gives an indication of how well-performing the communication behaves, while the less the distance between these two vectors, the better communication quality. So what we want is (or what would be an ideal communication system):

$$d(\vec{I^*}, \vec{I'}) \stackrel{!}{=} 0 \tag{3.9}$$

Source A and sink B are most likely to be distinct from each other, not being able to exchange the information directly. Somehow, a transformation has to be made of the information vector I^* to be able to be reconstructed by the sink. In reality, we face in most cases, that information I^* is transformed to encoded data D^*, which then is modulated into a signal U^* to be sent across some kind of channel. It is possible that the sent signal is transformed due to various disturbance sources. On the sink side, this (probably slightly different) signal U' will be received, which then is demodulated to data (or code) D' and is interpreted to information I'. Thus: [23]

$$I^* \to D^* \to U^* \xrightarrow{\text{Channel}} U' \to D' \to I' \tag{3.10}$$

Please note that the arrows are used to denote functions, not logical relationships.

Example 1 Let's assume that Alice (A, information source) wants to share with Bob (B, information sink) the colour of a thing she saw, red, which in this example is the source information I^*. She will first

transform her impression of the colour she saw into transferable data D^*, which is the word **r-e-d**, since language is the means we communicate by. Alice modulates then by moving her lips, making usage of her vocal cords and so on the code – an acoustic wave – onto the air, corresponding to U^*. This acoustic wave is sent across the media 'air' across a large hall. The acoustic wave then will be transformed due to distance, resonance etc, and Bob hears a slightly modified acoustic wave U'. His brain will transform the acoustic wave back to the word **r-e-d** (data/code) D', and he will – as the word **r-e-d** is somewhat a pointer to the abstract of an electromagnetic wave with the specific wavelength of red – imagine something which is near of what Alice has seen (I').

Example 2 More technical and therefore more interesting for us is a Physics application. Let's look at a particle detector (A), which detects particle velocity (I^*) which should be sent to a data acquisition module (B). The information I^* might be $v = 0.9 \cdot c$. Since machines do not compute very well with fractional expressions, the data sent will be an array of bits; this, in this case, might be the byte $(1, 0, 0, 1)$. The data then will be encoded. Let's assume the detector's encoder uses HAMMING code, then the generated code is the sequence of bits $\{1, 0, 0, 1, 1, 0, 0\}$ [23] [24]. This data code D^* will be modulated onto an analog channel (cable) and the sequence will be a time-dependent function based on high- and low-voltage states, i.e. the signal U^* (which can be modeled by a multiplication of the sequence of Dirac impulses (= Dirac comb) with a time-continuous function). In the channel, bits might get changed due to induced noise. The receiver DAQ/B then will convolve the received signal U' with a series of Dirac impulses and apply filters and HAMMING decoding to get the code D', which in our case most probably can be decoded to the initial data $(0, 0, 0, 0, 1, 0, 0, 1)$, from which the velocity $v = 0.9 \cdot c$ (=I') can be calculated. For an approximating illustration, see figure 3.6. This also is the ideal case, where $d(\vec{I^*}, \vec{I'}) = 0$ (for the one-dimensional vectors given, see definition in equation 3.9). However, if an error in encoding or decoding has happened, or in the case that the channel has induced too much noise, it might be that the deduced value I' is $v = 0.8 \cdot c$, and the distance between sent and received information is $d(I^*, \vec{I'}) = 0.1 \cdot c > 0$, therefore the transmission is not ideal and for sure too inaccurate for most particle detectors.

As we can see, the received and the sent code do not have to be the same so that sent and received information has to be the same when sufficient coding is applied.

To find sufficient coding hence is a crucial task when defining a communication protocol.

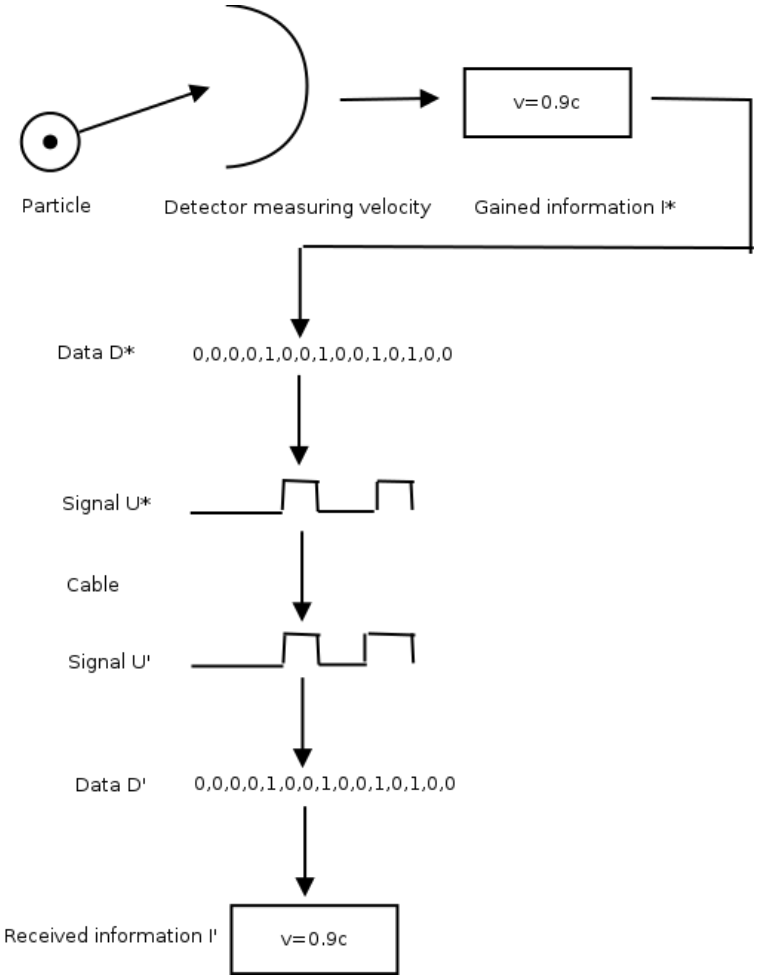

Figure 3.6: Approximating illustration of Example 2, referring to equation 3.10.

3.3.2 Specific towards asynchronous communication

We want to take more specific steps to look at the problems arisen by asynchronous communication: We will discuss the concrete implementation in section 5, and look here only on theoretical aspects.

In our case, we know that the information from source is the numbers of errors, thus the information to be sent is a natural number ($\in \mathbb{N}$) which has to be represented by a Boolean vector ($\in \mathbb{B}^n$). This transformation of a number $\mathbb{N} \to \mathbb{B}^n$ is trivial (converting the value of the number to binary) and shall not be subject to further discussion.

Assumptions We will assume that the encoded signal carries redundant information, which enables us to look not only on the custom protocol defined in this thesis, but on redundant code in general. Redundant information is used for error detection and reconstruction of the source information. It has to be noted

that redundant information does not have to be of the same size or larger than the source information; in most cases it is even the clue that it is of smaller size. For example an 8-bit vector with 1 parity bit carries redundant information only $\frac{1}{8}$ the size of the source information. Advanced codes, like Reed-Solomon code [25] [23] or HAMMING code [24] [23] generally are able to reconstruct disturbed signal while carrying not too much of redundant information.

The signal sent and the code used we regard as set, so we will look at the problems on the receiver side in order to obtain statement independent of which code and modulation type is used.

Asynchronisity Problems caused by asynchronous communication using a channel exposed to disturbance act in two dimensions:

- Periodical errors caused by acquisition due to the asynchronisity of modulation and demodulation Dirac combs
- Stochastic errors caused by disturbance (which in the case of this thesis mainly consists on radiation)

Demodulation How these two error causes effect decoding? It is assumed that a disturbed, time-continuous signal $u(t)$ is received, which corresponds to the signal U' in the equation 3.10. For demodulation, a multiplication is made with the Dirac comb, i.e. sampling function: [26]

$$\mathrm{III}_T(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT) = \frac{1}{T}\,\mathrm{III}\left(\frac{t}{T}\right) = \frac{1}{T}\sum_{k=-\infty}^{\infty} e^{2\pi i k \frac{t}{T}}. \tag{3.11}$$

$$(\mathrm{III}_T * u)(t) = \sum_{k=-\infty}^{\infty} u(t)\delta(t - kT) = \sum_{k=-\infty}^{\infty} u(kT)\delta(t - kT). \tag{3.12}$$

Decoding We then can map the received normalized Dirac impulses to a Boolean vector $\vec{D} \in \mathbb{B}^n$ (most likely one byte or several bytes), which corresponds to data D' in the block equation 3.10.

We assume that the receiver knows the decoding algorithm, hence would be able to find the information I' from the data D' without any problems. However, since this code only consider stochastic errors – only one of two error types – decoding would not be optimal. For decision making, thus, not only stochastic error should be recognized, but also periodical clock shift. Instead of the mapping

$$f(\vec{D}) : \mathbb{B}^n \to \mathbb{B}^m \tag{3.13}$$

Symbol	Meaning
$\leftarrow$	bit is sampled two times, shift backward
$\parallel$	bit is sampled one time, no shift
$\rightarrow$	bit is lost, shift forward

Table 3.2: Definitions of symbols used for set $\mathbb{S}$.

which should represent classical decoding, we get the mapping to a probability p (here P is denoting the set of possible probabilities p for all coding solutions I) of a two dimensional vector from which one basis vector spans a Boolean space $\mathbb{B}^m$, and the other one a 'clock shift set' $\mathbb{S}$ (*'likelihood' would be the more correct term then 'probability', but since likelihood directly emerges from probability in this case, these terms might be interchanged here*):

$$g(\vec{D}) : \mathbb{B}^n \rightarrow P\left(\mathbb{B}^m, \mathbb{S}\right) \tag{3.14}$$

We define clock shift set $\mathbb{S}$ as follows: Asynchronisity can cause shifts, i.e. that an information is read two times or that an information bit is lost, depending on the sending and sampling frequency of transmitter and receiver. We can define now $\mathbb{S}$ consisting of the set $\{\leftarrow, \parallel, \rightarrow\}$, with the meaning shown in table 3.2.

Based on the values which result from classical decoding and from clock shift estimation, a decision can be made which type of error – if any – has happened, and by that finding the final result:

$$h(p(\vec{I})) : P\left(\mathbb{B}^m, \mathbb{S}\right) \rightarrow \mathbb{B}^m \tag{3.15}$$

with p being here the actual probability function.

Working in two dimensions and problem fields, in the one of value decoding and in the one of clock shift recognition due to the asynchronous communication, we want to find the optimum function in order to find an optimum receiver. When knowing that equation 3.14 means the probability function as a function of which error might have happened, and equation 3.15 decides on these probabilities which error constellation is the most probable/the most likely, it is easy to see that the decoding and decision process $e(\vec{D})$ is the same as the composition of these functions:

$$e(\vec{D}) = h \circ g(\vec{D}) \tag{3.16}$$

$g(\vec{D})$ (equation 3.14) depends on the code used and the nature of asynchronisity, thus depend on the concrete setup. For the function $h(p)$ (equation 3.15) however there is a general statement to be made: this vector is

the most likely to have been sent, which is the most probable:

$$h(p) : \vec{I'} = \vec{I}(\max(P)) \tag{3.17}$$

With this scheme, we can perform further calculations to define our decoding procedure for our custom protocol.

4 Simulation and Calculation

Based on the theory provided in section 3, simulations and calculations have carried out in order to gain crucial information about the FPGA's behaviour and the efficiency and capability of the error mitigation techniques applied. The section contains both (statistical) simulations and (analytic) calculations.

The *Encyclopedia of Computer Science* states: [27]

> Simulation is the process of designing a model of a real or imagined system and conducting experiments with that model. The purpose of simulation experiments is to understand the behavior of the system or evaluate strategies for the operation of the system. Assumptions are made about this system and mathematical algorithms and relationships are derived to describe these assumptions – this constitutes a *model* that can reveal how the system works. If the system is simple, the model may be represented and solved analytically.

The goal of the simulation therefore is not only to verify the design, but also to understand the system. This is why the section on simulation is placed before the (more technical) design section. Also the calculations carried out may be seen, as the quote from the *Encyclopedia* suggest, as a form of simulation.

4.1 Simulation tools

As simulation tools, Altera Quartus and ModelSim have been used to set up the simulation environment and to simulate the FPGA in irradiation environment, and Python script for calculations. Simulation and data analysis scripts have been written both in VHDL and Python.

4.2 Simulation environment

A simulation environment has been set up for having a model of the environment which has to be faced when performing the test beam measurements and within the LHCb detector. It is difficult to predict which level of radiation actually has to be faced, making it necessary to have free parameters. The basic concept can be seen in figure 4.1: Parameters can be set manually, which then generate an environment which causes effects onto the FPGA.

Monte Carlo method is used to perform the simulations: random, but well-distributed events take place, as it is expected in a real irradiation environment [28].

A random generator is available in non-implementable VHDL which we make use of for Monte Carlo simulation. According to parameters which can be defined, mean upset rate and standard deviation is

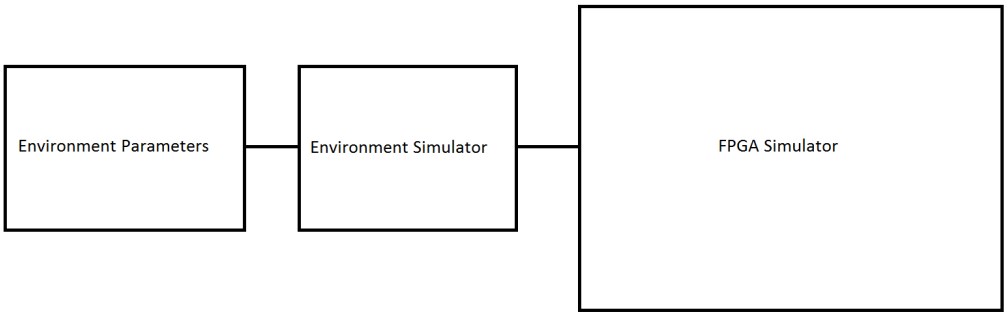

Figure 4.1: Block diagram showing the basic principle of the irradiation simulations performed.

calculated according to equation 3.1 and events are triggered accordingly. These events are detected by the FPGA simulation representation, which then flips bits. The structure of implementation can be seen in figure 0.1

The simulation has been verified using Monte Carlo simulation and checking plausibility of the data. Results for the simulation verification tests are shown in figure 4.2.

4.3 FPGA simulation

It might be of some interest how upset events have been implemented into the FPGA VHDL code. The principle is quite simple: each Flip-Flop is assigned an identification number. If an event occurs, the irradiation simulator randomly sends a number in the range of the identification numbers to the FPGA representation. By non-synthesized `if else`-statements it is checked whether a logic value is affected or not, and if so, the Flip-Flop value is being flipped, which causes a transient upset.

4.4 Data Fault Rate and Logic Size Calculations

Data Fault Rate In the theory section 3 we have shown dependencies of N-modular redundancy mitigation (see equation 3.6). However, we want to have more specific predicates for the data fault rate DFR. We can limit our area of work both in the order-of-redundancy dimension and in the size of voted logic, which allows us to determine a *window* in which we expect DFR to be (see figure 4.3 with the mentioned window in red). DFR is calculated for non-transient errors.

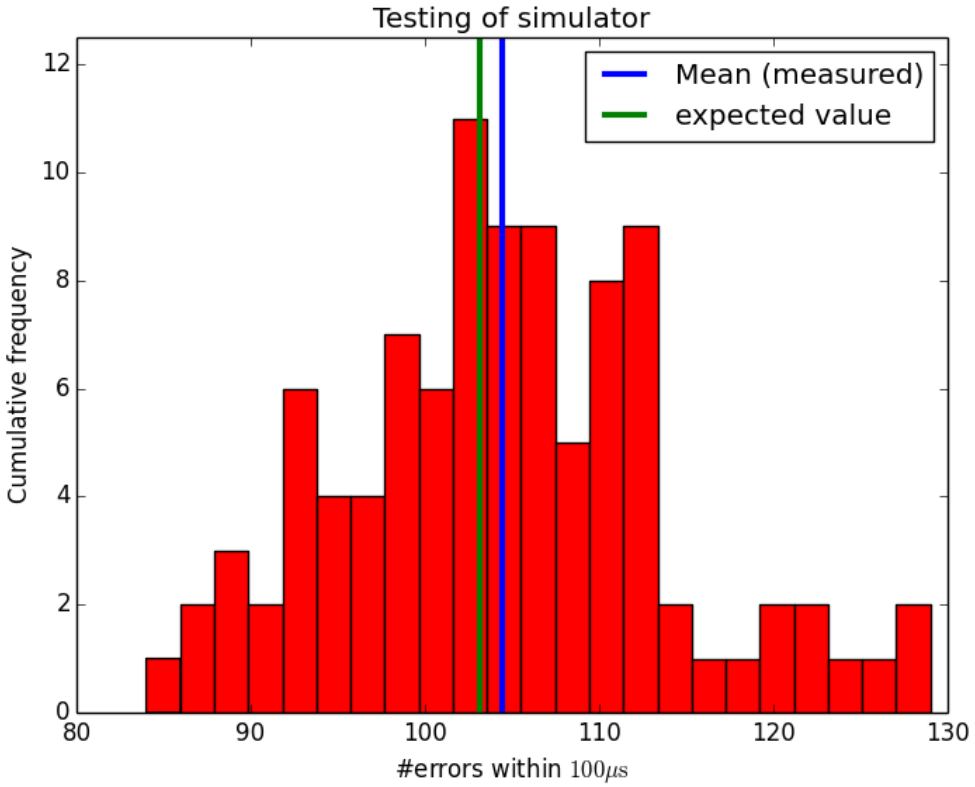

Figure 4.2: Simulator testing at $\sigma = 7 \cdot 10^{-7} \frac{\text{cm}^2}{\text{bit}}$, 4780 logic cells, particle flux $= 1.11 \cdot 10^{12} \frac{\text{n}}{\text{cm}^2 \cdot \text{h}}$, 100 samples

Logic size The size of the logic of the FPGA, obviously, depends linearly on the order-of-redundancy, since the basic logic simply is multiplied. The voted size does not play a major role for determination of the device's logic size, because it is only then a considerable factor when the size of the voted logic is very small and – by that – a lot of voters have to be used (this easily can be seen in figure 4.4).

4.5 Simulation results

Monte Carlo simulations have been performed for unvoted, triple modular and quintuple (5×) modular redundancy and are applied to shift registers; this causes the error to be only of transient nature. Simulation results are depicted in figure 4.5; it is to note that the calculated values refer to non-transient upsets, unlike the simulates values.

We see that Triple Modular Redundancy reduces Data Fault Rate by one order of magnitude in the current setup at least. Quintuple Modular Redundancy reduces Data Fault Rate even more, whereas the reduction is constant for transient upset and increases non-linearly for non-transient upsets. Therefore we can

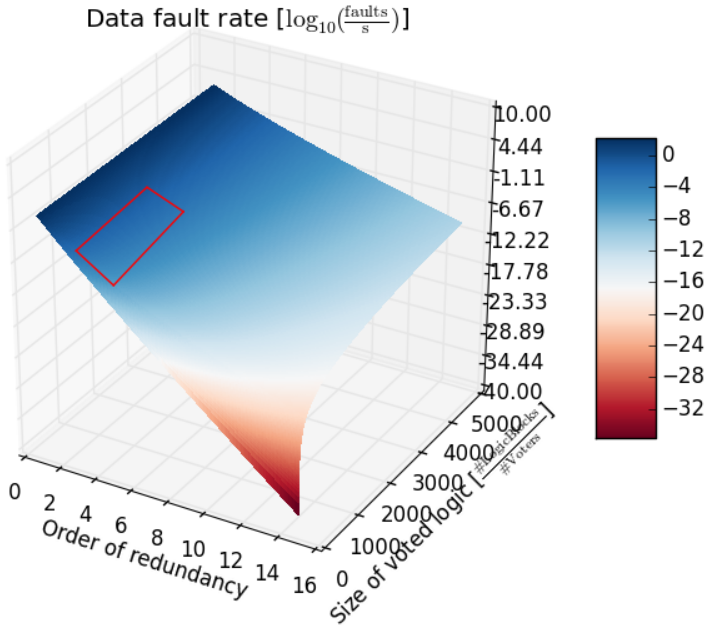

Figure 4.3: Data fault rate DFR as a function of order-of redundancy and the size of voted logic. The red windows corresponds to the area the FPGA is expected to work in. Note that the error resulting from the increased number of voters when working with smaller sizes of voted logic are not taken into account, as these effects were regarded as being negligible.

state that N-modular redundancy techniques seem to be appropriate measures against irradiation-induced faults; upon the simulation results it was decided to implement Triple and Quintuple Modular redundancy techniques – besides the non-voted register – for irradiation measurements into the test design.

4.6 Communication

In has been stated in section 3.3.2 that the Boolean-vector-to-Probability function $g(\vec{D})$ in equation 3.14 is different for each environment and has to be adapted to the concrete circumstances. In this section, we want to calculate such adapted functions $g(\vec{D})$ in order to get the decision function $e(\vec{D}) = g \circ h(\vec{D})$ (see equation 3.16).

The function $g(\vec{D})$ gives all information about probabilities of certain constellations to happen or – respectively – the likelihood of certain constellations to be for all possible Boolean vectors $\vec{D} \in \mathbb{B}^n$.

We define axiomatically an initial code according to table 4.1. It is just the initial value $\in \mathbb{B}$ multiplied by three, resulting in a code word $\in \mathbb{B}^3$.

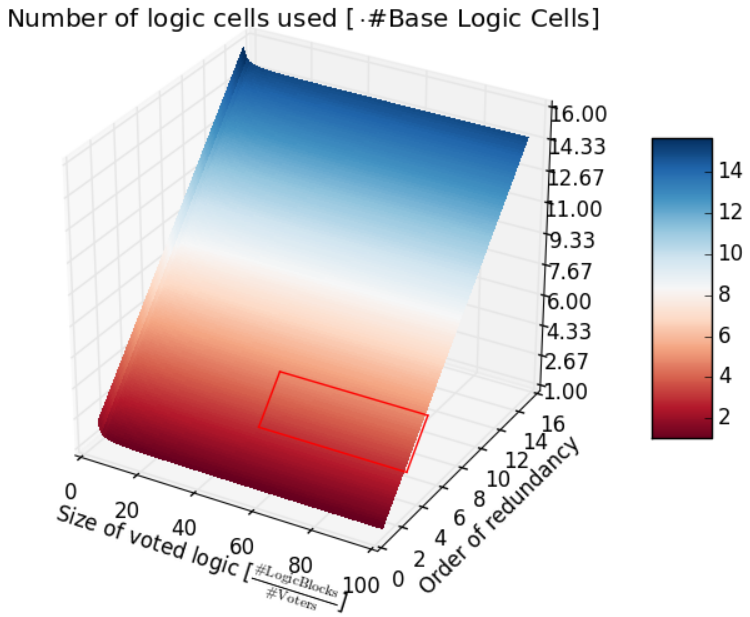

Figure 4.4: Logic size is plotted against order-of redundancy and the size of voted logic. The red windows shows the expected working area and has the same limits as the window in figure 4.3. The size of the voted logic seems to have no considerable influence to the logic size in the window area.

Value I	**Symbol/code word** $\vec{D}$
0	$(0,0,0)$
1	$(1,1,1)$

Table 4.1: Axiomatically defined coding table.

We also assume that the vector $\vec{D}'$ received is of length $3 \in \mathbb{B}^3$ and should be decoded back to a value. Now we want to calculate a distance between all code words $\vec{D}$ for all possible vectors received $\vec{D}'$ in order to calculate a distance $d(\vec{D}, \vec{D}')$. Both Euclidean and HAMMING distance can be used [23]. The results are listed in table 4.2. For our application, we will focus on HAMMING distance due to its higher countability [29].

Assuming that we have a given stochastic probability p_s of an error to happen within a defined time of one symbol acquisition cycle, we can calculate the probability p as a function of $\vec{D}'$ and $\vec{D}$ as a binomial distribution:

$$p(\vec{D}', \vec{D}) = \binom{n}{d(\vec{D}, \vec{D}')} \cdot p_s^{d(\vec{D}, \vec{D}')} \cdot (1-p)^{n-d(\vec{D}, \vec{D}')} \tag{4.1}$$

with n being the dimension of the Boolean space, which in our case is 3.

For exemplifying the calculation, we set p_s to 5% (which would be a much too high value in the actual

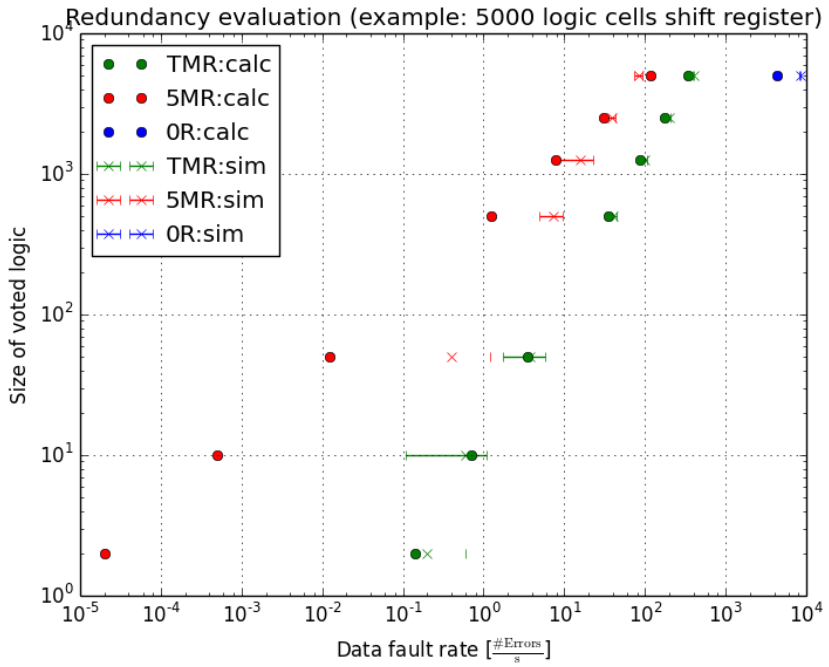

Figure 4.5: Simulation results. *calc* values refer to calculated values for non-transient upsets for non-redundant (*0R*), triple modular redundant (*TMR*) and quintuple modular redundant (*5MR*) shift registers, *sim* to the respective simulation results for transient upsets.

working environment). We then get the probabilities as a function of HAMMING distance as seen in table 4.3 when applying equation 4.1.

One might have noted that when applying equation 3.15 to the probability function shown in table 4.2, one gets the same logical expression as the one for the Triple Voter in section 3.2.1.

Now we have values for stochastic errors, still it is needed to have probabilities for clock shift. It is assumed that clock shifts happen periodically, thus clock shift probability primarily is a function of phase φ of the respective periodicity. Most likely it will meet the condition

$$1 = \int_{-\alpha}^{\alpha} p(\varphi)\mathrm{d}\varphi \tag{4.2}$$

meaning that the clock shift will happen within the phase interval $(-\alpha, \alpha)$, ideally determinable as one acquisition cycle.

If such an acquisition cycle is occurring, HAMMING distance will be calculated only in $\mathbb{B}^{n-1}$, in our case $\mathbb{B}^2$, where one dimension of $\vec{D}'$ is *shifted away* and the comparison vector $\vec{D}$ is reduced accordingly. A simplified model can be seen in table 4.4, in which it is not taken into account that an oversampled bit will

$\vec{D}$	$\vec{D}'$	**Hamming distance**	**Euclidean distance**
$(0,0,0)$	$(0,0,0)$	0	0
$(0,0,0)$	$(0,0,1)$	1	1
$(0,0,0)$	$(0,1,0)$	1	1
$(0,0,0)$	$(0,1,1)$	2	$\sqrt{2}$
$(0,0,0)$	$(1,0,0)$	1	1
$(0,0,0)$	$(1,0,1)$	2	$\sqrt{2}$
$(0,0,0)$	$(1,1,0)$	2	$\sqrt{2}$
$(0,0,0)$	$(1,1,1)$	3	$\sqrt{3}$
$(1,1,1)$	$(0,0,0)$	3	$\sqrt{3}$
$(1,1,1)$	$(0,0,1)$	2	$\sqrt{2}$
$(1,1,1)$	$(0,1,0)$	2	$\sqrt{2}$
$(1,1,1)$	$(0,1,1)$	1	1
$(1,1,1)$	$(1,0,0)$	2	$\sqrt{2}$
$(1,1,1)$	$(1,0,1)$	1	1
$(1,1,1)$	$(1,1,0)$	1	1
$(1,1,1)$	$(1,1,1)$	0	0

Table 4.2: $d(\vec{D}, \vec{D}')$ in HAMMING weight and Euclidean distance.

Hamming distance $d(\vec{D}, \vec{D}')$	**Probability** p[%]
0	85.74
1	13.54
2	7.13
3	0.01

Table 4.3: Probabilities of HAMMING distance values for the example $p_s = 5\%$.

$(\vec{D}_{\text{reduced}}, S)$	$\vec{D}'$	**Hamming distance**
$((0,0), \leftarrow)$	$(0,0,\not{0})$	0
$((0,0), \leftarrow)$	$(0,0,\not{1})$	0
$((0,0), \leftarrow)$	$(0,1,\not{0})$	1
$((0,0), \leftarrow)$	$(0,1,\not{1})$	1
$((0,0), \leftarrow)$	$(1,0,\not{0})$	1
$((0,0), \leftarrow)$	$(1,0,\not{1})$	1
$((0,0), \leftarrow)$	$(1,1,\not{0})$	2
$((0,0), \leftarrow)$	$(1,1,\not{1})$	2
$((1,1), \leftarrow)$	$(0,0,\not{0})$	2
$((1,1), \leftarrow)$	$(0,0,\not{1})$	2
$((1,1), \leftarrow)$	$(0,1,\not{0})$	1
$((1,1), \leftarrow)$	$(0,1,\not{1})$	1
$((1,1), \leftarrow)$	$(1,0,\not{0})$	1
$((1,1), \leftarrow)$	$(1,0,\not{1})$	1
$((1,1), \leftarrow)$	$(1,1,\not{0})$	0
$((1,1), \leftarrow)$	$(1,1,\not{1})$	0

Table 4.4: $d(\vec{D}, \vec{D}')$ in HAMMING weight if a backwards shift $\leftarrow$ (see table 3.2) has happened.

Hamming distance $d(\vec{D}, \vec{D}')$	**Probability** $p[\%]$
0	90.25
1	9.50
2	0.25

Table 4.5: As in table 4.3, but in an example case where clock shift has happened.

generate two bits of the same kind.

Equation 4.1 now can be applied (where n is $n_{\text{old}} - 1$ to get – for the example of $p_s = 5\%$ – the probability results in table 4.5).

In our case, the composite function from equation 3.16 is not an injective function, no unique decision can be made. In the implementation therefore, as will be explained in section 5, $\vec{D}$ not only consists on the three Boolean values obtained during one acquisition cycle, but also one bit value from the next cycle (which only can be achieved with latency), and the last bit value from the previous cycle.

Undoubtedly, it would be very interesting to look on mapping functions of this kind, since the acquisition vector spans two additional Boolean subspaces with the input vectors (in $\mathbb{B}^3$: $\vec{D}_{\text{new}} = \vec{D}_1 + \vec{D}_2 + \vec{D}_3$). We will omit to do so however since it is neither part nor necessary for this project.

5 Design

In this section implementation is discussed onto synthesis level. An overview over concepts should be given, rather than providing documentation (which is done in section 6). It is intended that the design methods and design structure can be comprehended to allow modification, extension and reproduction of the design. For implementation, VHDL code and ALTERA tool chain (Quartus, PrecisionSyntesis, ModelSim) are used.

5.1 Basic concept

It is recalled that the measurements take place under test beam conditions, with the Device-under-Test being the measurement FPGA at the same time. Therefore it is necessary to communicate via a communication board, which acquires the signal from the Device-under-Test in a radiation-free place and hands it to the end acquisition device.

An example of a test beam location is shown in figure 5.1.

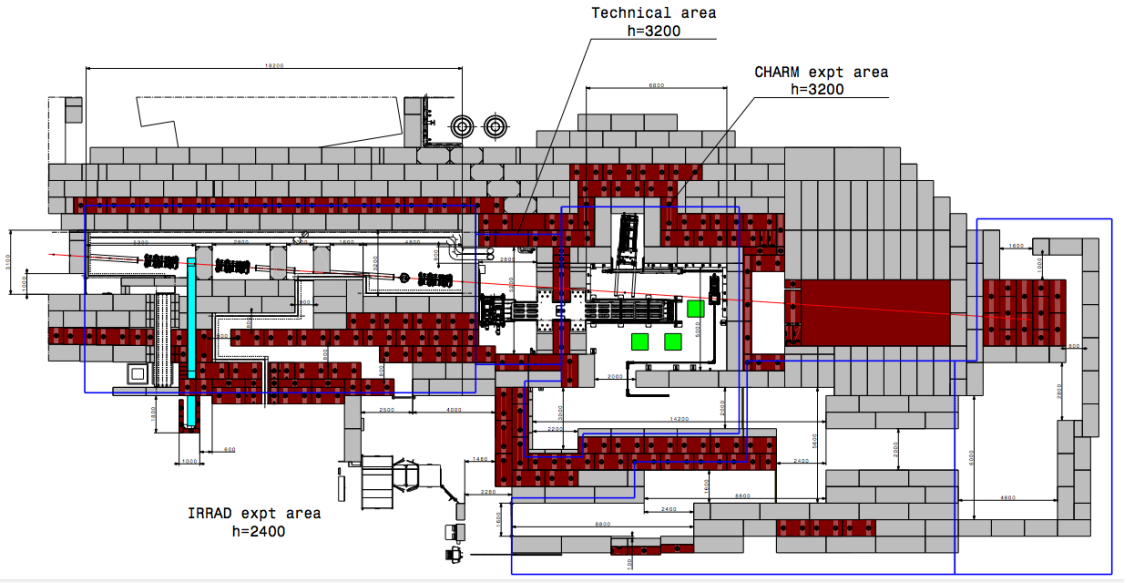

Figure 5.1: CHARM test beam facility for irradiation measurements at CERN. Initially foreseen to be taken at this place, the measurements will most likely take place at Horia Hulubei National Institute for R&D in Physics and Nuclear Engineering in Bucharest, Romania. *Source:* [4].

Since necessary cable length from the Device-under-Test board to the communication board is not known and it is likely that the number of signal wires is limited, it has been decided to use only one pair of Low-Voltage Differential Signalling (LVDS) cable to carry the signals because of their excellent Electromagnetic Compatibility (*EMC*) performance [30]. Maximum length to be faced has been estimated to 10m, which

without any doubt lies within the capabilities of LVDS signalling [30].

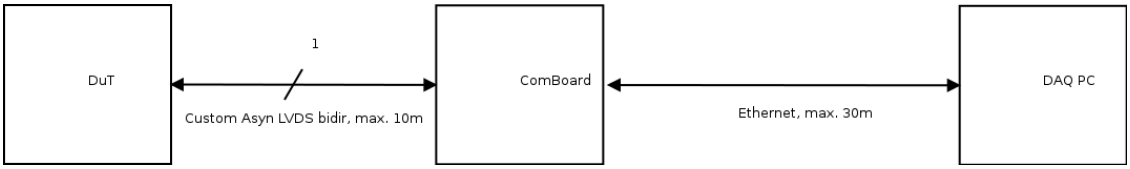

Figure 5.2: Block diagram of the basic FPGA system.

5.2 Device-under-Test

Core device in the measurement apparatus is the Device-under-Test (DuT) FPGA. Not only it will measure device cross section, but also efficiency and capability of mitigation techniques (Triple and Quintuple Modular Redundancy). It also will send the measurement results via LVDS to a communication board. In the appendix, an idea of implementation is given (figure 0.2).

5.2.1 Measurement design

All measurements should take place at the same time. Therefore non-redundant, Triple Modular and Quintuple Modular measurement blocks are implemented into one design, making it necessary to split the FPGA's logic blocks into three parts, as shown in figure 5.3. Each block is connected to an error counter which then will place the current value into the register read out by the communication interface.

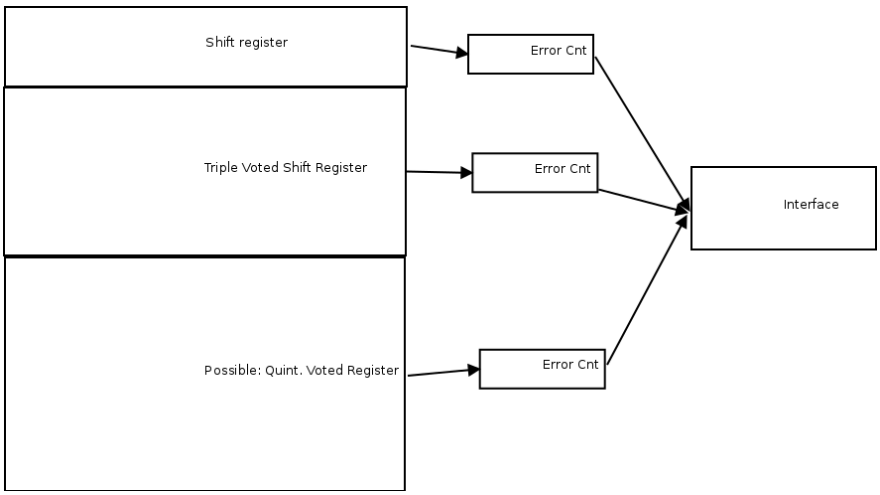

Figure 5.3: The Device-under-Test consists on three separate parts.

It is now shown how the shift register measurements take place:

Step 1 A bitstream generator injects a bit of a defined bit pattern into a shift register (figure 5.4). The value is also injected into a shift register not effected by radiation (clock delay line).

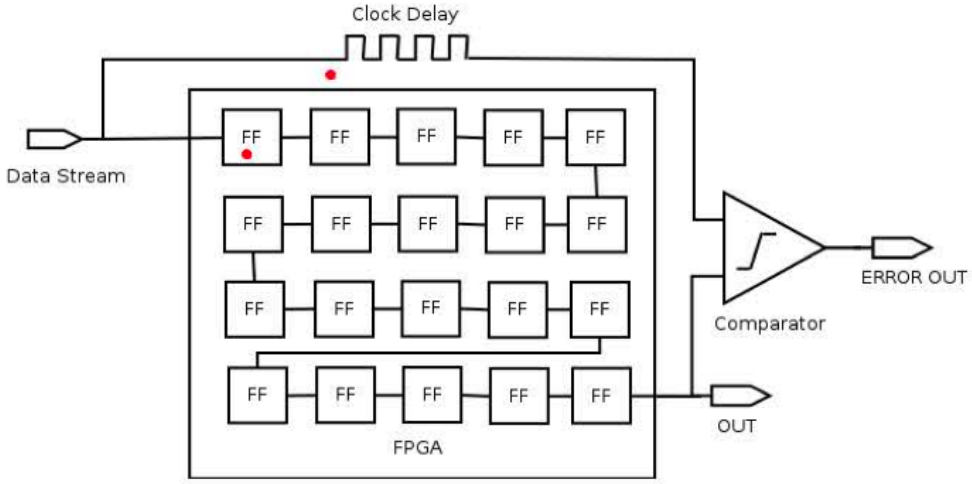

Figure 5.4: A value (red = '0') is injected into a Flip-Flop shift register by a bitstream generator.

Step 2 The value is shifted one value ahead as a clock cycle has passed; the first Flip Flop of the shift register receives a new bit from the bitstream generator (figure 5.5). The clock delay line behaves accordingly.

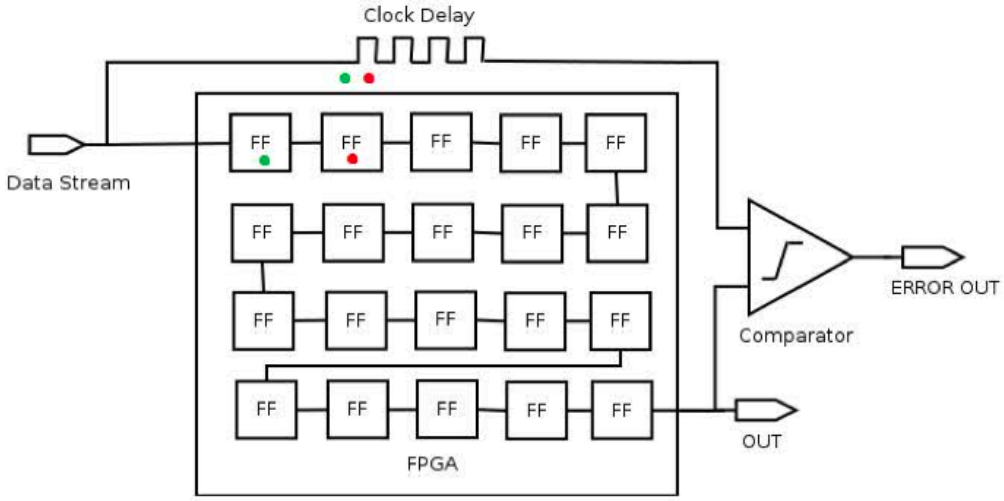

Figure 5.5: The value from Step 1 is handed to the second Flip-Flop; the first Flip-Flop receives a new value.

Step 3 Behaviour from Step 2 similarly takes into effect on shift register and delay line for a new clock cycle (figure 5.6).

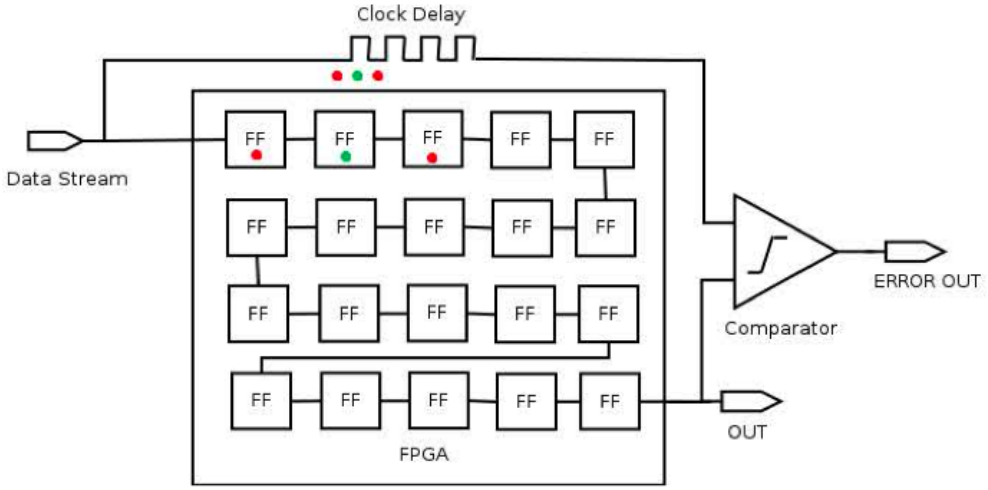

Figure 5.6: Similar to figure 5.5.

Step 4 Now a Single-Event Latch-up happens. A value from one of the Flip-Flops will be flipped in the next step. There is no effect on the radiation-tolerant delay line (figure 5.7).

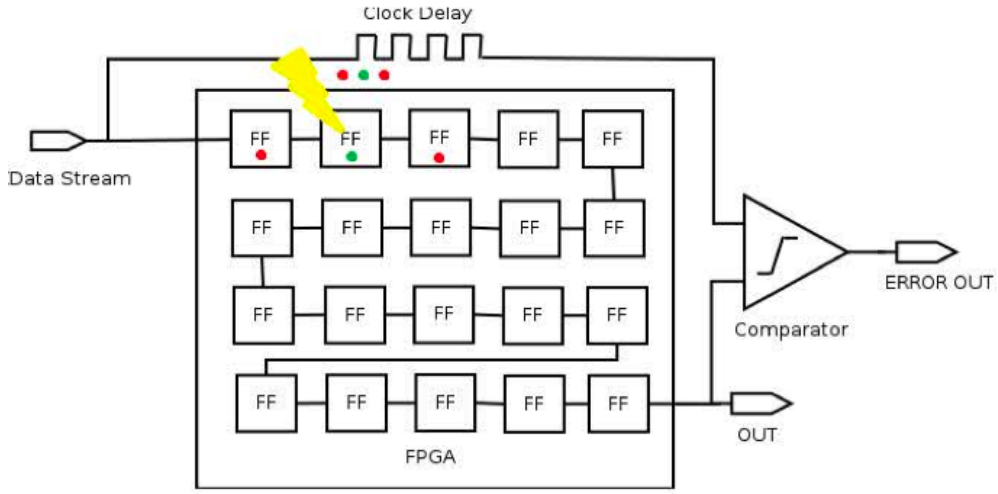

Figure 5.7: A Single-Event Latch-up happens on one of the Flip-Flops of the shift register.

Step 5 The radiation effect now has flipped a Flip-Flop value (figure 5.8).

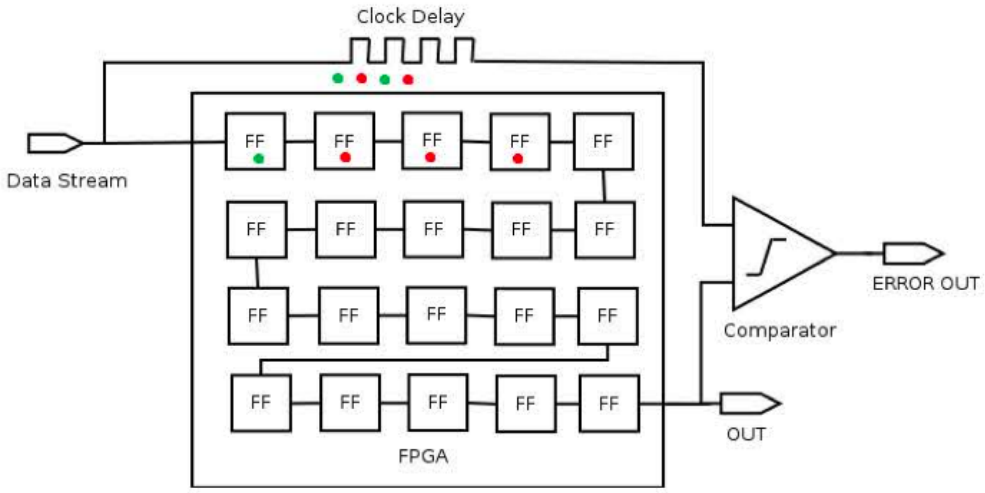

Figure 5.8: A Flip-Flop value is flipped and propagates.

Step 6 When the values have reached the comparator (not shown in figure 5.9), the comparator compares delay line signal and shift register signal and increases the error counter if they do not match.

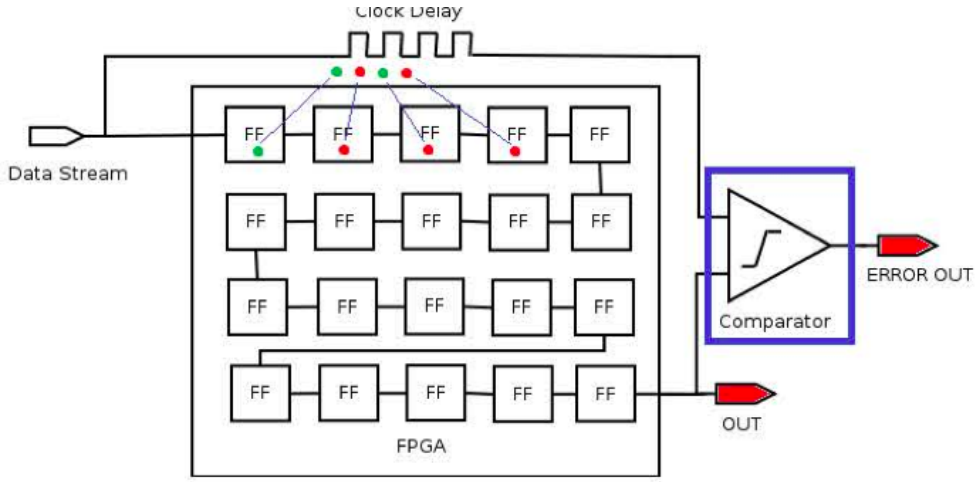

Figure 5.9: Delay line and shift register signal are compared when reaching the comparator (not shown).

5.2.2 Design paradigms

The design has been programmed in a generic way due to higher flexibility [31]. In that way, it is possible to determine shift register size according to the FPGA type used.

Another design paradigm crucial for functionality is to design every block which is not a measurement shift register (comparator, error counter, delay line, communication etc.) in a radiation-hard way, which means that they are subject to Quintuple Modular Redundancy.

5.3 Communication board

The communication board simply aims at receiving the data from the Device-under-Test and writing them into a register in order to be read out by an $I^2$2/Ethernet interface. The main part of the board is the communication interface, which we will discuss in some detail in the next subsection (the design is shown in figure 0.3).

5.4 Communication interface

The communication interface (see figure 0.4) has various constraining properties:

- Asynchronisity *(and moreover, only using one LVDS pair)*
- Radiation-hardness
- Bidirectionality

On the other hand, only low transmission rate is needed.

In this subsection it should be discussed how these constraints were met.

5.4.1 Asynchronisity

Asynchronisity means that no clock is provided with the signal. Since we do not have to meet any data transmission requirements, we use simple coding, as we have defined axiomatically in table 4.1. An initialization pattern is sent in order to allow the receiver to determine when a symbol start and ends.

A problem which might occur is that sending and receiving frequency mismatch. For this reason the three bits forming the symbol are read out with double sampling rate, thus twice the number of symbols is received than has been sent. This allows to detect if sampling has to speed up or to wait, as can be seen in figure **??**, and due to oversampling, even lost bits can be recovered.

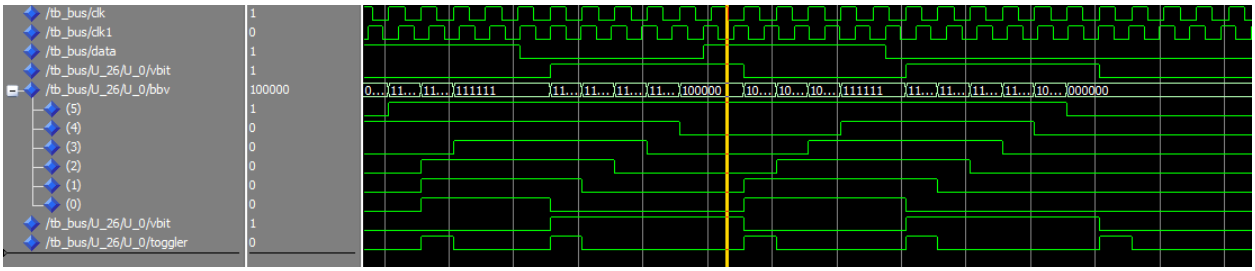

Figure 5.10: Example of asynchronous one-wire communication.

I	Clock shift	A	B	C	D
0	‖	0	1	0	0
ndef	ndef	0	1	0	1
1	ndef	0	1	1	0
1	←	0	1	1	1
0	‖	0	0	0	0
0	→	0	0	0	1
0	‖	0	0	1	0
ndef	ndef	0	0	1	1
ndef	ndef	1	1	0	0
1	‖	1	1	0	1
1	‖	1	1	1	0
1	‖	1	1	1	1

Table 5.1: Excerpt from the result LUT for the given assumptions for an input series A, B, C, D. 'ndef' values mean not defined values. For symbol meaning of clock shifts see table 3.2.

5.4.2 Radiation-hardness

Radiation-hardness means tolerance towards radiation-induced faults. A look-up table (LUT) has been defined to provide a mapping corresponding to equation 3.14 and equation 3.16.

Probabilities have been determined by estimation, using the rules that

- clock shift is much more probable than one radiation-induced error,
- but that one radiation-induced error is much more probable than two clock shifts.

An excerpt from the results can be seen in table 5.1.

5.4.3 Bidirectionality

Since only one differential pair is used, only one of the two communication transceivers can communicate at the same time. Therefore, nodes can be in 'sending' s, 'receiving' r or 'idle' i state, while they can change

Feature	Value
Logic Cells	65600
Configurable Logic Block Slices	10250
I/O banks	6

Table 5.2: Features of the KINTEX-7 XC7K70T. *Source:* [5].

states only in the manner described in equations 5.1 to 5.4 below:

$$s \xrightarrow{\text{if nothing to send}} i \tag{5.1}$$

$$i \xrightarrow{\text{if data is received}} r \tag{5.2}$$

$$i \xrightarrow{\text{if data should be sent}} s \tag{5.3}$$

$$r \xrightarrow{\text{if nothing is received}} i \tag{5.4}$$

5.5 Synthesis and Verification

5.5.1 Overview

The main focus on synthesis and verification is led on verifying that the critical parts sensitive to radiation are of negligible size on the one hand, and on verification of the feedbacked communication interface on the other. Both have been verified onto the level performed; a more detailed view will be lead on the radiation tolerance part.

All timing constraint have been met, working on 40MHz clock frequency.

KINTEX-7 XC7K70T FBG676 devices have been used on a PCB designed by the research group. Features are shown in table 5.2.

5.5.2 Radiation tolerance

It has to be ensured that radiation effects are not likely to happen on the voters itself. Quintuple Modular Redundancy is regarded to be sufficient for our needs, resulting from the simulations performed, why we look at the voters as the most critical parts. Since KINTEX-7 offer 6-input look-up table (LUT) technology [32], even quintuple voters only use one Configurable Logic Block (CLB) slice. The number of CLBs used for voters therefore can be well estimated (see table 5.3). We see in table 5.4 that the total number of voter LUTs (173) is less then 0.5% of the total number of used LUTs, from which it can be concluded that sufficient

Area	**Number of Critical CLBs**
Clock	1
Upset Measurement	49
Communication	123
Total	173

Table 5.3: Voter CLBs estimation for the Device-under-Test.

Shift Register Length	**Used Slice Registers**	**Used LUTs**
Total	82000	41000
0	1958	2212
100	2900	2432
200	3810	2491
1000	11030	3065
8000	74060	35648
8200	75870	37217
8500	–	–

Table 5.4: Number of logic cells used for different lengths of the measurement shift register. Maximum length has been reached at 8200, with 92% of the Slice Registers and 90% of the LUTs used.

radiation tolerance can be provided.

6 Documentation

6.1 Device-under-Test: Top-Level entity

6.1.1 File name

`fpga_system_struct.vhd`

6.1.2 Task

The task of this design is

1. to measure transient upsets caused by irradiation,
2. to drive this information via LVDS to the communication board.

6.1.3 Description

(go to the blocks for detailed description of the blocks)

Upset Measurement Upsets are measured by comparing the output of a faultless shift register to a shift register targeted by irradiation.

1. *Pulse Divider:* The Pulse Divider divides the 80 MHz clock from the board oscillator to the 40 MHz working frequency.
2. *Bitstream Generator:* Creates the bitstream injected to the shift registers.
3. *Shift register:* Carries the injected bitstream through its Flip-Flops. There will be three versions (non-voted, triple voted and quintuple voted) synthesized onto the board.
4. *Error Counter* The error counter counts the number of bits which have been flipped by comparing it to a (probably) faultless bitstream, for each of the three shift register versions.

Voting structure Every element which is crucial for operation and can be multiplied is quintuple voted (some elements e.g. buffers cannot be multiplied.) The in- and output signals of the voters are asserted to be kept in order to prevent them to be removed during compilation and synthesis.

Communication The communication part consists of the asyn and the I^2C interface. The asyn interface is the one prepared for usage in the tests.

1. *Data Mux* Data Mux routes the data into the registers read out by the two interfaces. The routing is described in the block, and might be changed (the generic "lengthofdata" of the Asyn Interface has to be matched, though.

2. *Asyn Interface* The Asyn Interface communicates by a differential signal pair with the communication board. It has quintuple voted architecture and thus can be used in radiation environment.

3. *I^2C Interface* Depending on the type of the FPGA, either the Kintex or the Spartan version of the I^2C Interface is generates. The I^2C Interface is not radiation-hard and is used to read the board status in the preparatory phase.

6.1.4 Generics

`integer` **logicsize** The number of Flip-Flops in 1 shift register. The larger the number, the more likely an upset will occur. Maximum is between 8200 and 8500. When using 8200 Flip-Flop shift register, 92% of all slice registers and 90% of the look-up tables on the Kintex-7 FPGA are used.

`std_logic` **kintex7** Determines the configuration according to the table below.

Value	Device used
'1'	Kintex-7 FPGA
'0'	Spartan-6 FPGAs

Table 6.1: Device-under-Test top-level entity generic 'kintex7'.

`std_logic` **faults** Faults are injected for testing purposes if faults = '1',

6.2 Device-under-Test: Voters

6.2.1 Triple Redundancy Voter

is RTL.

File name `Voter_rtl.vhd`

Task Triple Modular Redundancy Voter

Description LUT for Triple Modular Redundancy. If there is mismatch on (at least) one of the signals, error is driven out.

6.2.2 Quintuple Redundancy Voter

is RTL.

File name `VoterFive_rtl.vhd`

Task Quintuple Modular Redundancy Voter

Description LUT for 5th-Order (Quintuple) Modular Redundancy. Is designed for 5 or 6-input LUTs If there is mismatch on (at least) one of the signals, error is driven out.

6.2.3 Multibit Quintuple Redundancy Voter

is RTL.

File name `VoterFiveMultiple_rtl.vhd`

Task Quintuple Modular Redundancy Voter for std_logic_vector input

Description LUT for 5th-Order (Quintuple) Modular Redundancy. Is designed for 5 or 6-input LUTs. If there is mismatch on (at least) one of the signals, 'error' is driven out.

Generics

numberofbits length of voted std_logic_vector

6.3 Device-under-Tests: Bitstream generation

6.3.1 Bitstream generator

is RTL.

File name `Bitstream_Generator_rtl.vhd`

Task Streams a pattern synchronously with clock. The pattern is sent through the shift register, whose output then will be compared to the original pattern.

Description The pattern is a sequence of subsequences, whereas the subsequence is alternating each period length, and whereas the pattern consist of subsequences whose period are, also alternating, increasing with time until the maximum period "maxperiod" has been reached, or, respectively the period '1' has been reached.

Generics

maxperiod see description 6.3.1.

6.3.2 Fault injector

is RTL.

File name `Bitstream_Generator_rtl.vhd`

Task Passes the bitstream to the output when 'fault' is '0', creates a stream only consisting of '1' when 'fault' is '1'.

Generics

`std_ulogic` **faults** If set to '1', faults are injected to the bitstream.

6.4 Device-under-Tests: Shift register, error counting and data registers

6.4.1 Shift register

is RTL.

File name `ShiftRegister_rtl.vhd`

Task Shiftregister, Flip-Flops addressable by the irradiation generator.

Description Simple shift register. A Flip-Flop addressed by the irradiation value will flip its value.

Generics

`integer` **flipflopno** number of Flip-Flops (shift cells) within the shift register.

`integer` **startaddress** the addresses of the Flip-Flops range from (startaddress) to (startaddress + fliflopno).

`std_logic` **sel_or_seu** this parameter has only to be set by performing a simulation and determines whether a Flip-Flop suffers from a transient Single Event Latch-up or a Single Event Upset.

6.4.2 Error counter

is RTL.

File name `ErrorDetection_rtl.vhd`

Task Counts the number of upsets which have occurred within the shift register.

Description Delays the original bitstream by the amount of Flip-Flops and compares the delayed stream with the shift register output. If inequality has been detected, the error counter increments.

Generics

`integer` **flipflopno** number of Flip-Flops (shift cells) within the shift register

`integer` **add** if several voted shift register are pipelined, the voter delay has to be taken into consideration and is expressed in the "add" value.

6.4.3 Data registers *(SixteenToEightBits)*

is RTL.

File name `SixteenToEightBits_rtl.vhd`

Task Writes relevant information in the 255 8bit status registers for the I^2C interface to 'status'. Writes relevant information in the 48bit register for the asyn interface to 'OutputTx'.

Description Note that integers are written in Little Endian mode.

Status	**Information**
1 down to 0	errors in unvoted logic
3 down to 2	errors in triple voted logic
5 down to 4	errors in quintuple voted logic
6	bit 0: error in triple voted logic
	bit 4: error in quintuple voted logic
7	0x00
8	0xFF
9	bit 0: dscrim (discriminator)
10	length of received sequence (Asyn Rx)
18 down to 11	received sequence
19	bit 0: error from receiver

Table 6.2: Status bits of I^2C register.

Status	**Information**
15 down to 0	erros in unvoted logic
31 down to 16	errors in triple voted logic
47 down to 32	errors in quintuple voted logic

Table 6.3: Status bits of Output register.

6.5 Communication board: Top-level entity

6.5.1 File name

`fpga_communication_struct.vhd`

6.5.2 Task

The task of this design is to receive information from the Device-under-Test, decode it and write it into a register readable by an USB-I^2C interface.

6.5.3 Description

(click the blocks for detailed description of the blocks)

The communication board is designed to pass the information received from the DuT in the irradiation area to the data acquisition computer in the control/user room of the test facility. It is assumed that it will be placed outside an area where it has to face high radiation doses. The information received is written into the register as show in the table below:

Status	**Information**
0	Bit '0': error
	Bit '1': error
1	Data length
9 down to 8	Errors in unvoted logic
7 down to 6	Errors in triple voted logic
5 down to 4	Errors in quintuple voted logic

Table 6.4: Communication board top-level entity status bytes.

6.5.4 Generics

`std_logic` **kintex7** Determines the configuration according to the table below.

Value	**Device used**
'1'	Kintex-7 FPGA
'0'	Spartan-6 FPGAs

Table 6.5: Communication board top-level entity generic 'kintex7'.

6.6 Communication interface

6.6.1 Top-level: Asynchronous Bidirectional Communication Interface

is Structural

File name `asynintlvdsvoted_struct.vhd`

Task This module is a bidirectional 1-differential pair asynchronous quintuple voted transceiver. Data to be transmitted is handed over as a `std_logic_vector` to 'data_in', whereas the size of the vector has to be specified in 'lengthofdata'.
The decoded received data is given out as the vector 'data_out', whereas only the bits 0 to ('data_length'-1) of the vector contains the information received.
The discriminator 'dscrim' is set to high each cycle a new data package has been decoded (and the value of 'data_decoded' might have changed). That an error in data reception has occurred and the package has not been decoded is indicated by the output 'error'. 'data_n' and 'data_p' are the XILINX differential output ports.

Description The Pulse Divider generates the pulse for the transmitter. The transmitter hands the output sequence over to the output buffer, with the inverted 'sending' signal indicating when the buffer is in receiving mode.
Bidir LVDS passes data only if the transmitter is not sending, otherwise driving HIGH (= '1').
The receiver processes the data received; the processed (decoded) data are given to the output port described.
The 'bus_prepare' module determines how the communication between the two modules behave. Receiver and transmitter logic form two blocks, which are separately quintuple voted.

Generics

`integer` **lengthofdata** Denotes the length of the vector to be transmitted.

`integer` **lenght_waiting** This value defines the number of clock cycles, for which a wait request is sent to the transmitter. The integer has to be higher or equal than (81·lut).

`integer` **length_sending** This number defines the number of clock cycles, after which a wait period is sent. The request will cause the transmitter stop sending data after it has finished sending its current data

package. The value should be long enough to ensure that the transmitter sends at least one initialization pattern during the period.

`std_ulogic` **masterslave** The term 'Master' is misleading, since there is not really a Master/Slave system. The bit solely indicates, when the two board are started up simultaneously, which board will be transmitting in the initial state and whose wait period occurrence will be shifted. Of course, if the master of one board is set to '1', the one on the other board has to be set to '0'.

`integer` **length_init** This integer value states after how many packages an initialization pattern should be sent, so that the receiver recognizes the data packages.

`integer` **lut** The value states how much the sending frequency is relative to the data sampling frequency (Please note: sending frequency is NOT necessarily the same as the clock frequency of the transmitter module). If it is set to '1', one has to know which of the clocks of the two communication modules goes faster (unless both have accurately the same frequency), stated in mmdir. The look-up table is chosen according to this option, whereas '3' being the most complex, and '1' the most simple LUT. The value has to be the same for both modules!

`std_ulogic` **mmdir** If the multiplier is 1 and runaway direction cannot be determined therefore, the information about runaway direction has to be given manually (this does not apply if exactly the same clock is used for both communication modules). '0' stated that the receiver has a slower, '1' that it has a faster clock than the transmitting block.

`string` **io_standard** Is a string, which denotes which LVDS IO standard is used at the port.

`std_ulogic` **bidirectional** The value 'bidirectional' determines how the interface between the modules is used, according to the table below:

Module 1	Module 2	Transmission code
1	1	Bidirectional communication
1	0	Transmission from Module 2 to Module 1
0	1	Transmission from Module 1 to Module 2
0	0	Dependent*

Table 6.6: Communication interface generic 'bidirectional'.

* Dependent on which module start communicating first, or, if both start communicating at the same time, which is the master.

6.6.2 Bus Prepare

is RTL.

File name `bus_prepare_rtl.vhd`

Task Asyn Transmitter (BusTx) requires a vector for 'data_length' and a 64bit-vector for the data. Bus prepare fits the OutputTx vector into this scheme. Moreover, it sets the waiting length (which defines how long a module waits for a response before sending again) and the frequency of waiting periods (which (says how often the module will stop sending data to wait for the other module to respond). Counts the number of upsets which have occurred within the shift register.

Description The bits from 0 to ('datalength'-1) of the 64-bit-vector is filled with the 'data_in'-bits. The other bits of the 64-bit-vector are set to '0'.
After a defined period ('comtogfreq'), the module will send the signal to stop sending data (which is indicated by 'datalength' = "000000"). During a defined time ('comlenfreq'), the transmitter is requested not to send any data, in order to wait for the partner module to start sending data.
To prevent that the two communication modules' waiting periods interfere with each other, the start of the defined periods can be shifted by ('comlenfreq'+'comtogfreq')/2 clock cycles.

Generics

`integer` **datalength** Defines the length of the input data. Can be 1 to 64. The 'data_in' vector has to match 'datalength'.

`std_ulogic` **toggling** Enables wait periods. When disabled, only unidirectional communication is assumed.

`integer` **comlenfreq** This value defines the number of clock cycles, for which a wait request is sent to the transmitter. The integer has to be higher or equal than (81·'multiplier').

`integer` **comtogfreq** This number defines the number of clock cycles, after which a wait period is sent. The request will cause the transmitter stop sending data after it has finished sending its current data package. The value should be long enough to ensure that the transmitter sends at least one initialization pattern during the period.

`std_ulogic` **shift** If the communication module on both sides use the same comlenfreq and comtogfreq values, and both modules are started up at the same time, a delay has to be imposed on one of the modules to prevent both modules from sending at the same time.

6.6.3 Asynchronous Custom Interface Transmitter (BusTx)

is RTL.

File name `BusTx_rtl.vhd`

Task (see block Asynchronous Bidirectional Communication Interface for the whole environment)
This block sends data defined in 'data_in' of the length 'data_lenght'. The 'data_in vector will only be sent from bits 0 to ('data_length'-1), the rest will be ignored.

The block only transmits, when 'receiving' is zero; this enables the tristate use in the block "Asynchronous Bidirectional Communication Interface". It indicates that it is in sending mode by the bit 'sending'. The output data is sent on 'data_out'.

Every ('initfreq' · 'clkperiod'), an initialization pattern is sent, to let the receiver know the start point of each package.

Data sampling is double the clock frequency of the transmitter module, thus 'clk' of the transmitter has to have half the frequency than 'clk' of the receiver.

Description

Package A data package consists of the elements below (one character representing 1 bit)

`SLLLLLLYDDDDDDDDYDDDDDDDDYDDDDDDDDYE`:

`S`: Start bit, to indicate the start of the package

`L`: 6 bit for the length of the data package

`Y`: Synchronization bit, to prevent long sequences of same bit values, preventing the signal of 'running away'

`D`: Data, split in 8bit = 1 byte subpackages.

`E`: End bit, to indicate the end of the package.

The sequence `DY` is repeated as many times as the length vector says.

Initialization After a defined period ('initfreq'), an initialization pattern is sent.

Generics

`std_ulogic` **master** The term 'Master' is misleading, since there is not really a Master/Slave system. The bit solely indicates, when the two board are started up simultaneously, which board will be transmitting in the initial state. Of course, if the master of one board is set to '1', the one on the other board has to be set to '0'.

`integer` **initfreq** This integer value states after how many packages an initialization pattern should be sent, so that the receiver recognizes the data packages.

`integer` **multiplier** The value states how much the sending frequency is relative to the data sampling frequency (Please note: sending frequency is NOT necessarily the same as the clock frequency of the transmitter module). If it is set to '1', one has to know which of the clocks of the two communication modules goes faster (unless both have accurately the same frequency). The look-up table is chosen according to this option, whereas '3' being the most complex, and '1' the most simple LUT. Multipliers of Tx and Rx have to be equal.

6.6.4 Asynchronous Custom Interface Receiver (BusRx)

is RTL.

File name `BusRx_rtl.vhd`

Task This block reads packages sent by an Asynchronous Custom Interface Transmitter.

The data received has to be connected to the 'data_Rx' port. The block then drives out the length of the package ('data_length') and the data which have been transmitted ('data_decoded'). 'data_decoded' is a 64-bit vector, and obviously only the bits 0 to ('data_length'-1) represent the received data.

The discriminator 'dscrim' is set to high each cycle a new data package has been decoded (and the value of data_decoded might have changed).

That an error in data reception has occurred and the package has not been decoded is indicated by the output 'error'.

Whether or not data is received is indicated by 'receiving'. To go from receiving to non-receiving state, at least (81 · 'multiplier') clock cycles, '0' have to be received continuously.

Data sampling is half the clock frequency of the transmitter module, thus 'clk' of the transmitter has to have half the frequency than 'clk' of the receiver.

Description Data is read out with double the frequency then the bit sending frequency. Moreover, bits can be multiplied by 2 or 3. Thus, for each information bit, 2 to 6 readout bits are registered (saved in the bbv array). A look-up table now relates these readout bits to an output information bit value, and also extrapolates the information which of the two communication module's clock is faster. Readout then is adjusted according to that information. Regular adjustment (ensured by synchronization bits) has to occur in order that neither signal nor readout run away. The package then is read out according to the underlying package pattern:

Package A data package consists of the elements below (one character representing 1 bit)

`SLLLLLLYDDDDDDDDYDDDDDDDDYDDDDDDDDYE`

`S`: Start bit, to indicate the start of the package

`L`: 6 bit for the length of the data package

`Y`: Synchronization bit, to prevent long sequences of same bit values, preventing the signal of 'running away'

`D`: Data, split in 8bit = 1 byte subpackages.

`E`: End bit, to indicate the end of the package.

Generics

`integer` **multiplier** The value states how much the sending frequency is relative to the data sampling frequency (Please note: sending frequency is NOT necessarily the same as the clock frequency of the transmitter module). If it is set to '1', one has to know which of the clocks of the two communication modules goes faster (unless both have accurately the same frequency). The look-up table is chosen according to this option, whereas '3' being the most complex, and '1' the most simple LUT. Multipliers of Tx and Rx have to be equal.

`std_ulogic` **mismatchdirection** If the multiplier is 1 and runaway direction cannot be determined therefore, the information about runaway direction has to be given manually (this does (not apply if exactly the same clock is used for both communication modules). '0' stated that the receiver has a slower, '1' that it has a faster clock than the transmitting block.

7 Outlook

The design for the irradiation measurement FPGAs has been concluded. Though, one may find one important part missing in the thesis: The performance of measurement and its results, which have not been part of the project which focused on the Engineering side. While talking with people in industry and research, I became aware that there seems to be a vital interest about possible usage of Xilinx7 Series FPGAs in radiation environment applications, be it in the field of Particle Physics, Space Science or Medicine. Having provided a design able to perform such measurements, I hope that I have contributed – with the help of my supervisors and colleagues – to this evaluation.

The measurements are likely to start in Summer 2015, which I am looking forward to.

References

[1] Bridgford B., Carmichael C., and Tseng C.W. *Single-Event Upset Mitigation Selection Guide*. Xilinx, v1.0 edition, March 2008.

[2] Engel J.D. and Withlin M.K. Predicting on-orbit static single event upset rates in Xilinx Virtex FPGAs, 2006.

[3] Xilinx SEE Consortium. `http://www.xilinx.com/products/silicon_solutions/marketspecific_devices/aero_def/capabilities/see.htm`.

[4] Mekki J *(responsible)*. CHARM Layout – East Area. `http://charm.web.cern.ch/CHARM/Documents/Layout/layout_est.pdf`, 2014.

[5] Xilinx. *7 Series FPGAs Overview*, v1.15 edition, February 2014.

[6] Pinget R. and MacPherson A. LHC performance and statistics. `http://lhc-statistics.web.cern.ch/LHC-Statistics/`, March 2014.

[7] Bartsch V., Postranecky M., Targett-Adams C., Warren M., and Wing M. Estimation of radiation effects in the front-end electronics of the electromagnetic calorimeter using physics events. *EUDET-Report*, 3, 2007.

[8] Cassano L. M. *Analysis and Test of the Effects of Single Event Upsets Affecting the Configuration Memory of SRAM-based FPGA*. PhD thesis, University of Pisa, 2013.

[9] Bo G., Yu X., Ren D., Li Y., Sun J., Cui J., Wang Y., and Li M. Total dose ionizing irradiation effects on a static random access memory field programmable gate array. *Journal of Semiconductors*, 33(2), March 2012.

[10] Schmidt F. Fault tolerant design implementation on radiation hardened by design SRAM-based FPGAs. Master's thesis, Massachusetts Institute of Technology, 2013.

[11] Xilinx. *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics*, v2.8 edition, March 2014.

[12] Xilinx. *7 Series FPGAs Configuration – User Guide*, v1.7 edition, October 2013.

[13] Faerber C., Uwer U., Wiedner D., Leverington B., and Ekelhof R. Radiation tolerance tests of SRAM-based FPGAs for the potential usage in the readout electronics for the LHCb experiment. *Journal of Instrumentation*, 2013.

[14] Ceschia M., Bellato M., Menichelli M., Papi A., Wyss J., and Paccagnella A. Ion beam testing of altera apex FPGAs. *Proceedings of the 2002 IEEE Radiation Effects Data Workshop*, pages 45–50, July 2002.

[15] Berg M. *Field Programmable Gate Array Single Event Effect Radiation Testing.* NASA Electronic Parts and Packaging, 2012.

[16] Quinn H., Grahan P., Krone J., Caffrey M., and Rezgui S. Radiation-induced multi-bit upsets in SRAM-based FPGAs. *IEEE Transactions on Nuclear Science*, 52(6):2455–2461, December 2005.

[17] Xilinx. *ChipScope Pro Integrated Bit Error Ratio Test for Kintex-7 FPGA GTX*, v2.01.a edition, October 2011.

[18] Rezgui S., Wang J.J., Sun Y., Cronquist B., and McCollum J. New reprogrammable and non-volatile radiation tolerant FPGA: RTA3P. *Aerospace Conference, IEEE*, 2008.

[19] Soos C. SEU effects in FPGA – how to deal with them? June 2009.

[20] Edmonds L. Recommendations regarding the use of CREME96 for heavy-ion SEU rate calculations.

[21] Amiri M. and Přenosil V. Reliability of digital systems. N-modular redundancy. 2013.

[22] Shooman M.L. *Reliability of Computer Systems and Networks.* Wiley, 2002.

[23] Hufschmid M. *Information und Kommunikation. Grundlagen und Verfahren der Informationsübertragung.* Teubner, 2006.

[24] Hamming R.W. Error detecting and error correcting codes. *The Bell System Technical Journal*, XXVI(2):147–162, April 1950.

[25] Blömer J. Algorithmische Codierungstheorie. Universität Paderborn, 2007.

[26] Córdoba A. Dirac combs. *Letters in Mathematical Physics*, 17(3):191–196, 1989.

[27] Smith R.D. Simulation article. *Encyclopedia of Computer Science*, 4th Edition, 1998.

[28] Weinzierl S. Introduction to Monte Carlo methods. 2000.

[29] Wegner P. A technique for counting ones in a binary computer. *Communications of the ACM*, 3(5):322, May 1960.

[30] Parkes S.M. High-speed, low-power, excellent EMC: LVDS for on-board data handling. *LVDS Application Workshop*, 2011.

[31] Brantschen S., Pichler M., and Schenk K. *Digitale ASIC Schaltungstechnik. Ein Lehrgang zur Entwicklung von integreierten digitalen Schaltungen und FPGA*. Institut für Mikroelektronik, 2008.

[32] Xilinx. *7 Series FPGAs Configurable Logic Block*, v1.7 edition, November 2014.

Appendix

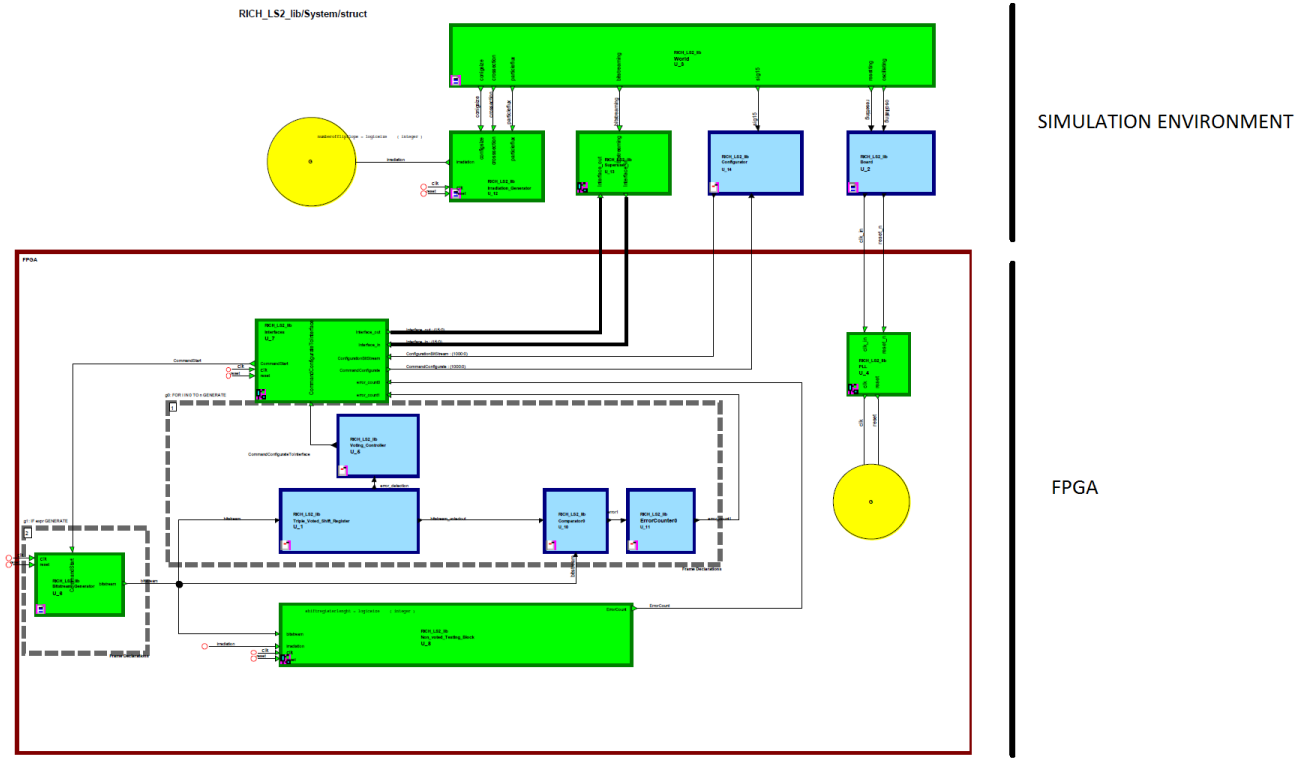

Figure 0.1: Implementation of the simulation in ALTERA Quartus.

Figure 0.2: Implementation of the Device-under-Test in ALTERA Quartus.

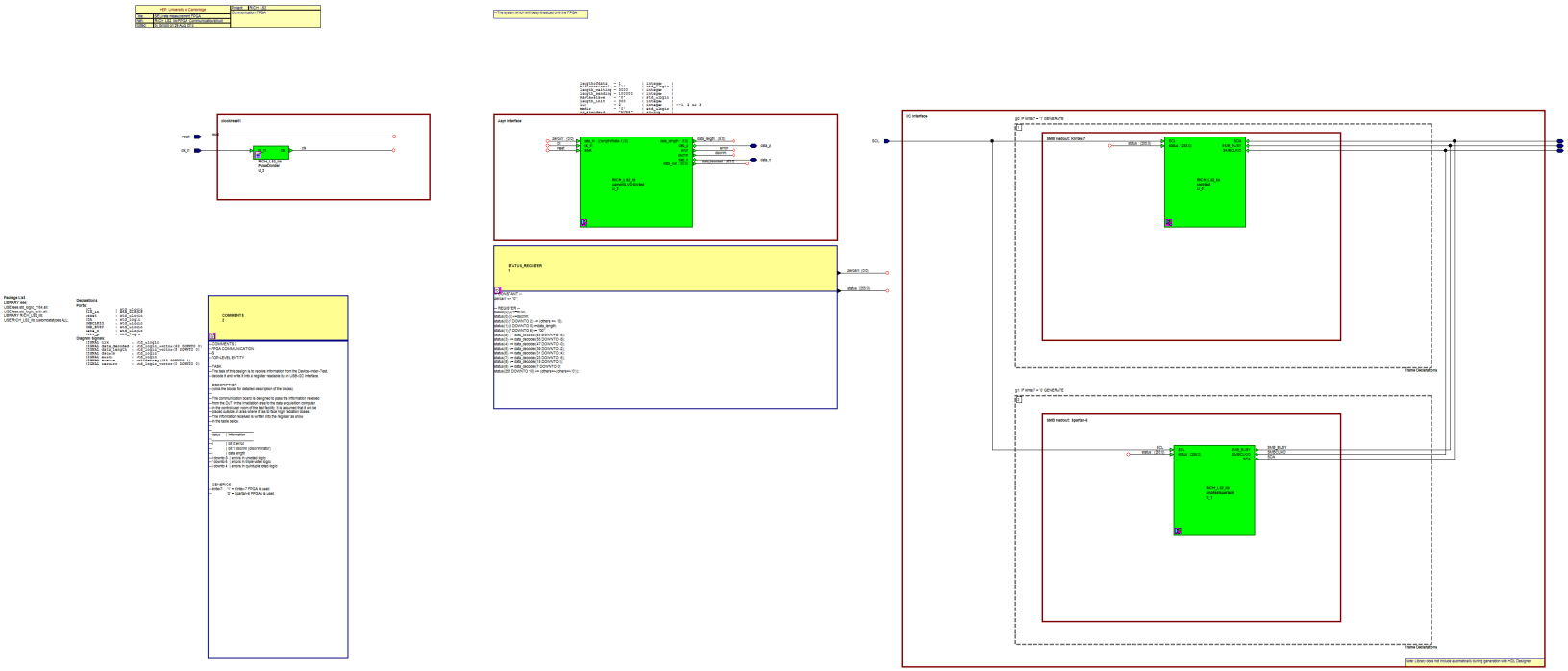

Figure 0.3: Implementation of the communication board in ALTERA Quartus.

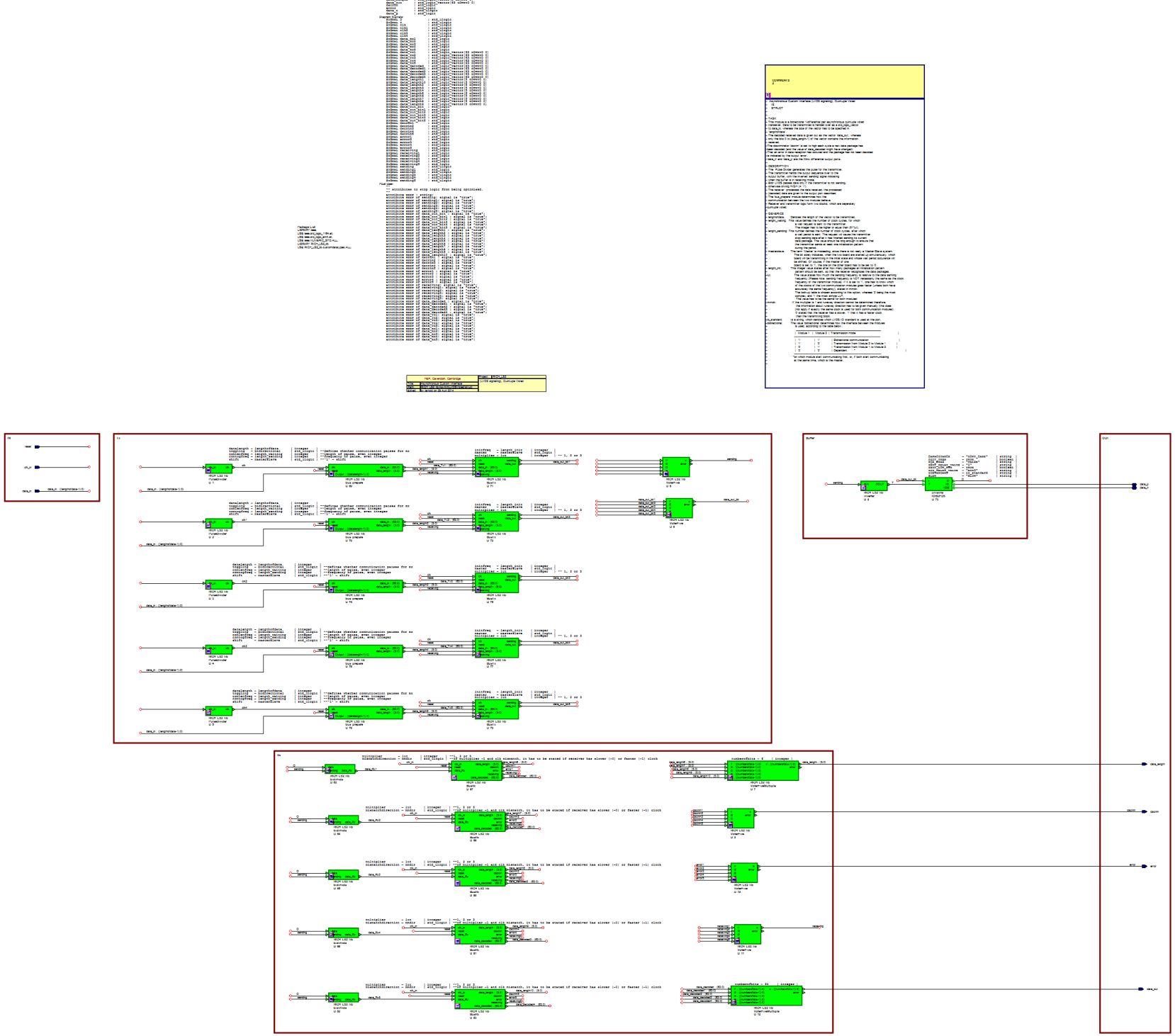

Figure 0.4: Implementation of the communication interface in ALTERA Quartus.