Department of Physics and Astronomy

University of Heidelberg

Diploma thesis

in Physics

submitted by

Robert Weidel

born in Lörrach

2011

Development of a Knowledge-based Processor Controller for High Speed Track Trigger Applications

This diploma thesis has been carried out by Robert Weidel

at the

Physikalisches Institut Heidelberg

under the supervision of

Prof. Dr. André Schöning

Development of a Knowledge-based Processor Controller for High Speed Track Trigger Applications :

The Large Hadron Collider (LHC), at which protons of $E_p = 7$ TeV (nominal) are collided at a frequency of 40 MHz, is planned to be upgraded in the year 2020 to increase the luminosity by a factor 5. Recording of events in the ATLAS detector is limited to about 200 Hz and a multi-level trigger system is used to reduce the event rate. One option to keep the L1 trigger rate at the present level of about 75 kHz is to integrate a Fast Track Trigger, which could be implemented using fast lookups based on Knowledge-based Processors (KBP).
In this thesis the functionality and performance of KBPs is investigated and a controller is developed for usage in a Fast Track Trigger. To study the hardware performance an evaluation kit equipped with a NL9000 chip is used. The latency and the result rate of the chip is measured and found to be compatible with the specifications of the upgraded ATLAS experiment. In context of this work it is shown that a realisation of a Fast Track Trigger is possible by using the NL9000. The functional behaviour of the NL9000 chip is described by a Verilog model. This model is verified and demonstrated to be reliable.

Entwicklung eines Controllers für Knowledge-based Processors als Anwendung in einem High Speed Track Trigger :

Der Large Hadron Collider (LHC), in welchem Protonen mit $E_p = 7$ TeV (nominell) in einer Frequenz von 40 MHz kollidieren, wird voraussichtlich im Jahr 2020 aufgerüstet, um die Luminosität um Faktor 5 zu erhöhen. Da das Speichern der Ereignisse mit maximal 200 Hz geschehen kann, wird ein mehrstufiger Trigger verwendet um die Ereignisrate zu verringern. Eine Möglichkeit um die Triggerrate auf dem bestehenden Niveau zu halten ist es, einen Fast Track Trigger auf dem ersten Triggerlevel zu integrieren, welcher schnelle Lookups von Knowledge-based Processors (KBP) benutzt.
In dieser Arbeit werden die Funktionen und die Leistungsfähigkeit eines KBPs in Hinblick auf die Nutzung in einem Fast Track Trigger untersucht. Um die Leistungsfähigkeit der Hardware zu analysieren, wurde eine Testumgebung mit einem NL9000 Chip genutzt. Die Kompatibilität mit dem aufgerüsteten ATLAS Experiment wurde anhand von Messungen der Latenz und der Ergebnisrate des Chips festgestellt. Im Rahmen dieser Arbeit wird gezeigt, dass es möglich ist, ein Fast Track Trigger mit Hilfe von NL9000 Chips zu realisieren.
Die Funktionsweise des NL9000 Chips wird durch ein Verilog-Simulationsmodell beschrieben. Die Zuverlässigkeit dieses Modells wurde durch Tests verifiziert.

Contents

1 Introduction

One of the main goals of modern physics is to obtain a deeper understanding of the structure of matter. The currently most successful model to describe elementary particles and their interactions is the Standard Model. It includes spin 1/2 particles called Fermions consisting of 6 Leptons and 6 Quarks divided in 3 families. For each particle exists an corresponding anti particle. Fermions carry fractional electric charges, which are multiples of 1/3. The Quarks have an additional property called color. Color is, in comparison to electromagnetism, the charge of the strong force but with three aspects instead of one. The Weak Interaction couples to left-handed particles or right-handed anti particles. Furthermore, there are 12 gauge bosons with spin 1 including the photon, for the electromagnetic exchange, the $W^{\pm}$ and Z^0 for the weak decay and the 8 gluons for the strong force. A scalar boson called Higgs particle is also predicted by the Standard Model but not yet experimentally confirmed.

The Higgs particle is necessary within the Standard Model to explain why particles are massive. Searches for the Higgs particle have been performed at LEP and Tevatron and were recently started at LHC. Since the cross section for the production of events with the Higgs particle is supposed to be very low and the background is higher by several orders of magnitude, there are a large amount of events required to detect a significant signature for a discovery.

One of the most promising approaches to detect new fundamental particles is the LHC accelerator at CERN. It allows to search for new physics in a wide range of energy with high statistics by colliding protons. In addition to the search for the Higgs boson, the LHC is able to discover other particles proposed in a variety of models.

Although the standard model predicts the current research results very well, the possible existence of a fourth generation cannot explicitly be excluded. There are models which predict at least another family of fermions beyond the standard model. The most promising way of detecting a new generation is analysing the cross section of decay topologies, which are sensitive to new heavy particles. In order to examine possible new physics phenomena with high sensitivity, proton-proton collisions are studied at high centre of mass energies (nominal 14 TeV) and high luminosities $L = 10^{34} cm^{-2} s^{-1}$ at the experiments ATLAS and CMS.

The LHC was also built to address fundamental questions of todays physics. One of these questions is concerning the mass distribution of the universe. Known matter contributes to only about 4 % of the observed objects in the universe. The rest is referred to as dark matter and dark energy, whose existence were discovered by cosmological and astrophysical observations. Measuring potential candidates for dark matter, e.g. the lightest supersymmetric particles, would fill a huge gap in cosmological models. Supersymmetry could also be extended to a Grand Unification theory, which unifies leptons and quarks and the electroweak and strong interaction. The asymmetry of matter and anti matter in our universe is also not understood and is therefore investigated by the LHCb experiment, where CP violation in B-decays is studied. By studying collisions at very high energy densities, it is possible to simulate conditions similar to short time after the 'big bang', also providing information about the theoretically predicted quark-gluon-plasma. For this purpose, the LHC is also used to produce lead ion collisions, which are analysed by the dedicated ALICE detector.

To use the potential of the LHC more efficiently, an upgrade is planned in the year 2020. By this, the luminosity of the LHC will be increased by an order of magnitude providing higher statistics for detection of new physics. This will also challenge the detectors, since more events need to be traced in the same time interval. In the ATLAS experiment the inner detector will be replaced by a new one with more readout channels and higher granularity. To process data with higher rates, it will be also necessary to improve the trigger system of ATLAS. One option is to include a Fast Track Trigger at the first trigger level to keep the rates at L2 low enough.

The aim of this diploma thesis is to investigate a Knowledge-based Processor (KBP) of Netlogic [1] in regard to the possible usage in a Fast Track Trigger. In the course of the thesis two tools were available: The encrypted Verilog simulation model of the used KBP and the Development Platform HTG-100GIG. The main task was to design a controller for the KBP to test the chip in a hardware application. Since there was also a simulation model available the tests were applied using both tools and the results were compared.

In the next chapter the LHC and the ATLAS detector will be described. The trigger system and the L1 track trigger, which is the main focus of this thesis, are discussed in chapter 3. The technology used in the KBPs and the chip family itself will be topic of chapter 4. Then, the simulation (chapter 5) and the hardware setup (chapter 6) will be described followed by an overview of the developed controller (chapter 7). In chapter 8 the results of the hardware application will be presented and compared to the simulation.

2 Collider Experiments in CERN

In this chapter the Large Hadron Collider (LHC) and its accelerator chain will be described. The ATLAS detector will be discussed in more detail and the future plans to upgrade the LHC and the ATLAS detector will be introduced.

2.1 Large Hadron Collider

The Large Hadron Colllider is a particle accelerator and collider operating with two beams with counter wise rotating proton or lead bunches based on the synchrotron principle [2]. It is located at CERN (Conseil Européen pour la Recherche Nucléaire) close to Geneva in Switzerland. CERN is the world's largest research center with 20 Member States and about 3400 employees plus 8000 visiting scientist. Its main purpose is to gain information about the fundamental constituents of matter.

The LHC, which replaced the Large Electron-Positron Collider (LEP) in the 26.7 km long tunnel system at CERN, collides protons or lead ions at four collision points. At each of these four points a detector is located to reconstruct the tracks and energy deposition of particles generated by the collisions. The design centre of mass energy when protons are used is 14 TeV. This would not be possible with electrons because of radiation losses. Although protons accelerated in LHC are at $99,9999991$ % of the speed of light, the radiation losses are not the most dominant limitation of the maximum energy. Instead of that, the design energy of protons is limited by the strength of the dipole magnets (8.33 T) to bend the protons. There are 1232 of these magnets arranged at the storage ring in addition to 392 quadrupole magnets, which are necessary to focus the particle beams. Both sorts of magnets are superconducting devices cooled to 1.9 K.

For accelerating particles to 7 TeV, there are multiple pre-accelerators necessary, as displayed in Figure 2.1. The first step is the linear accelerator Linac2 to get the protons to an energy of 50 MeV. Afterwards, they are injected into the Proton-Booster Synchrotron (PSB) to increase the energy to 1.4 GeV, followed by the Proton Synchrotron (PS) with an output energy of 26 GeV. Then, the protons are injected into the Super Proton Synchrotron (SPS) to get them finally to the injection energy of 450 GeV needed to bring them to their maximum energy in the LHC storage ring within 20 minutes.

In order to achieve a luminosity of 10^{34} cm^{-2}s^{-1} the particles have to be focused in bundles of up to 115 billion protons each. LHC can be filled with a maximum of 2800 of these packages at the same time, giving the accelerator a total collision frequency of 40 MHz. The bundles themselves are compacted to a diameter of 16 μm and a length of 8 cm, to get a precise collision point inside the detectors and stable

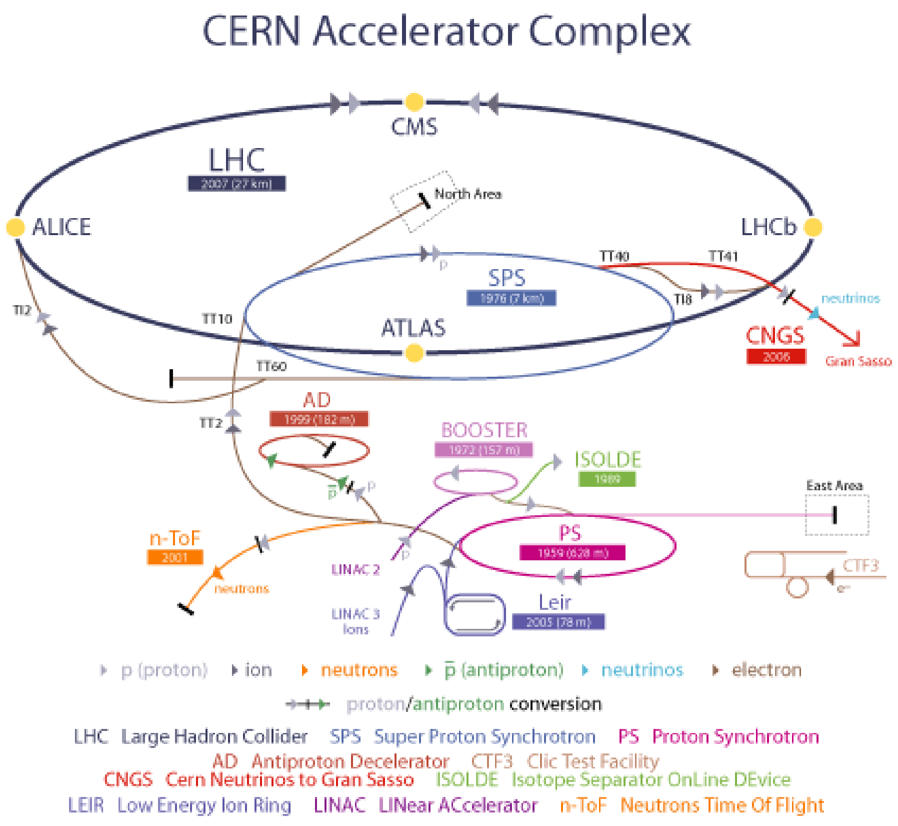


Figure 2.1: The LHC Accelerator and Detector Complex and its preaccelerators to illustrate the acceleration steps of proton or lead ion bunches. The proton and lead ion bunches start with different linear accelerators and different first synchrotron accelerators. Both of them are injected into the PS afterwards, following the accelerator chain to the SPS and finally to the LHC.(Image from [3])

beams for several hours.

To detect and measure the produced particles of the collisions there are 4 detectors with different functionality at the LHC. LHCb is an asymmetric detector built in only one direction close to the beam pipe with the primary usage for B-physics. These measurements can help to explain fundamental questions like the Matter - Antimatter asymmetry in the universe. ALICE is especially designed to study the lead ion collisions and has therefore a special configuration. ATLAS and CMS are 4π multi purpose detectors built to handle high luminosities. These types of detectors are used to maximize the sensitivity to find any signs of new physics in the experiment. One of the advantages of building two similar detectors is that discoveries in one of the two can easily be checked in the second. In the following chapter the ATLAS experiment will be described in detail.

2.2 The ATLAS detector

The ATLAS detector, which is an acronym for 'A Toroidal LHC ApparatuS', is located at one of the four collision points of the LHC [4]. It is a cylindric detector with a length of 44 m and a diameter of 25 m as shown in Figure 2.2. The detector can be cut down to 4 important devices: the inner detector, the calorimeters, the muon chambers and the magnetic system.

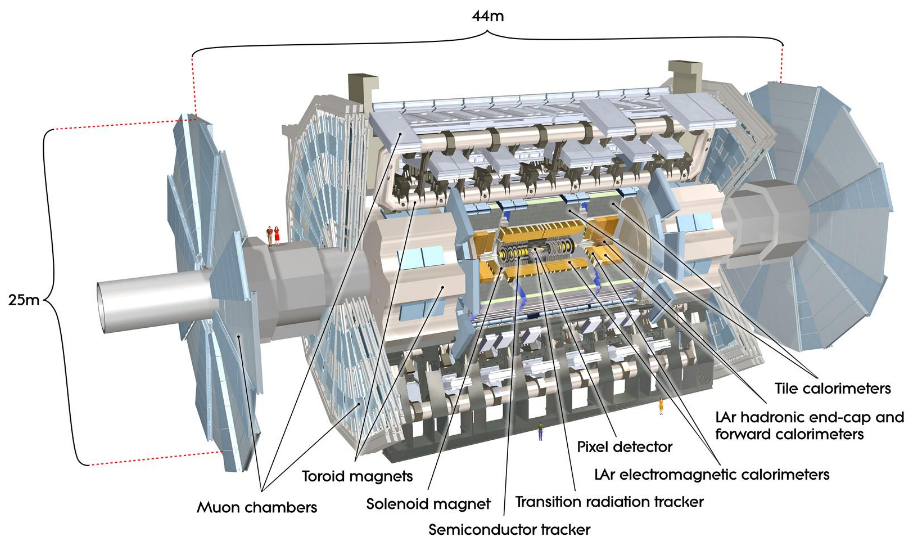

44m
25m
Tile calorimeters
LAr hadronic end-cap and forward calorimeters
Pixel detector
LAr electromagnetic calorimeters
Toroid magnets
Muon chambers
Solenoid magnet
Transition radiation tracker
Semiconductor tracker

Figure 2.2: Overview of the ATLAS experiment and its detector components. In the center is the inner detector surrounded by the electromagnetic and hadronic calorimeters. The outer parts is occupied by the magnet systems and muon chambers.(Image from [5])

The Inner Detector

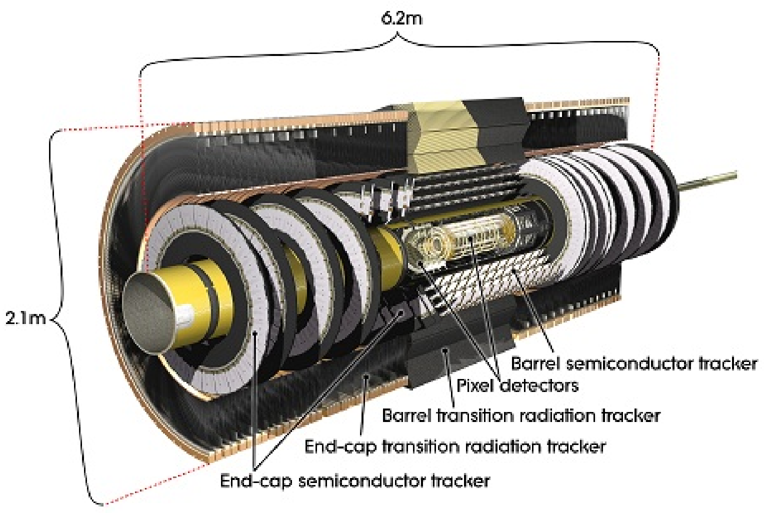

6.2m
2.1m
Barrel semiconductor tracker
Pixel detectors
Barrel transition radiation tracker
End-cap transition radiation tracker
End-cap semiconductor tracker

Figure 2.3: The profile of the inner detector. In the center is the the inner detector barrel with pixels, SCT and TRT layers arranged in cylinders around the beam axis. At the end-caps the pixel and SCT detectors are aligned radially on discs surrounded by also radially straightened TRT detectors.(Image from [5])

The inner detector (shown in figure 2.3) is used to reconstruct tracks, primary and secondary vertices, momentum and charges and is therefore in a 2 T B-field. It consists of three devices located in the barrel (see figure 2.4) and the end-caps: the semiconductor pixel detector, the semiconductor microstrip (SCT) detector and the Transition Radiation Tracker (TRT). As shown in figure 2.5 the innermost part is the pixel detector and covers $|\eta| < 2.5$[1]. It has the smallest granularity to produce an excellent resolution of the primary vertex of the particles created by the collisions. Because there are about 1000 particles to be expected with a frequency of 40 MHz, the pixel detector is designed as three cylinders around the beam axis and in 2 x 3 discs in the end-cap in very small-sized dimensions of 50 μm x 400 μm in (R - ϕ) x z direction. Overall, this results in 80.4 million readout channels for the pixel detectors.

The SCT consists of 4 pairs of detector layers in the barrel region of $|\eta| < 2.5$ gaining 8 hits per particle crossing it. The orientation of the strips are tilted by 40 mrad, one of them being in parallel with the beam axis. Furthermore, they are used in 2 x 9 discs radially aligned to the beam in the end-caps. The dimension of each strip is 80 μm in R - ϕ and 6.4 cm in z direction producing a total of 6.3 million readout channels.

[1]In the coordinate system of the ATLAS detector the collision point it defined as origin. The pseudorapidity η is defined as $\eta = -ln\tan(\theta/2)$ with θ being the polar angle from the beam axis. The azimuthal angle ϕ is defined as the angle around the beam axis

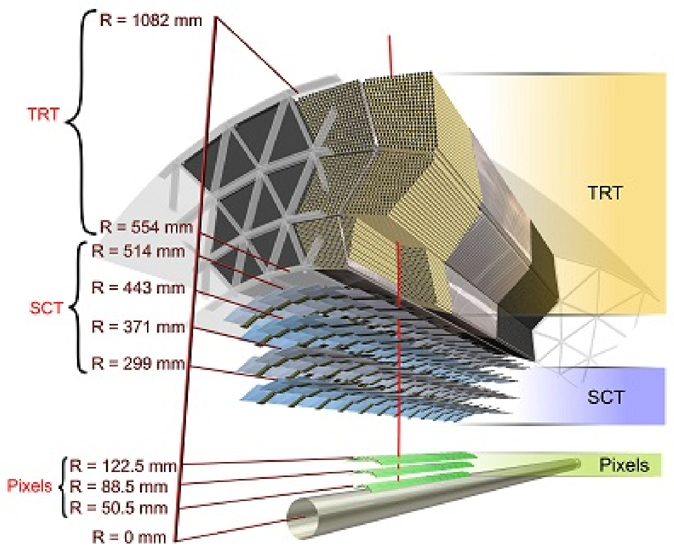
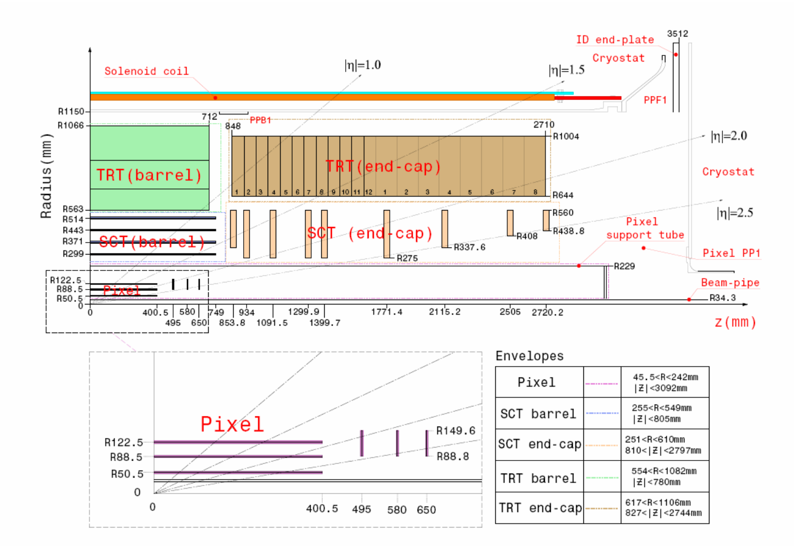


Figure 2.4: Illustration of the interior composition of the inner detector barrel including the radii of the Pixel, silicon microstrip (SCT) and Transition Radiation Tracker (TRT) layers. (Image from [5])

The outermost part of the inner detector is the TRT consisting of 4 mm diameter straw tubes. In the barrel region they are 144 mm long, cover $|\eta| < 2.0$ and due to being in parallel to the beam axis, they do not produce z information. There are also straw tubes in the end-caps, arranged radially in wheels with a length of 37 cm. Altogether, the TRT has about 351 thousand readout channels resulting in about 36 hits per track leading to a good R - ϕ resolution after reconstruction.

Calorimeters

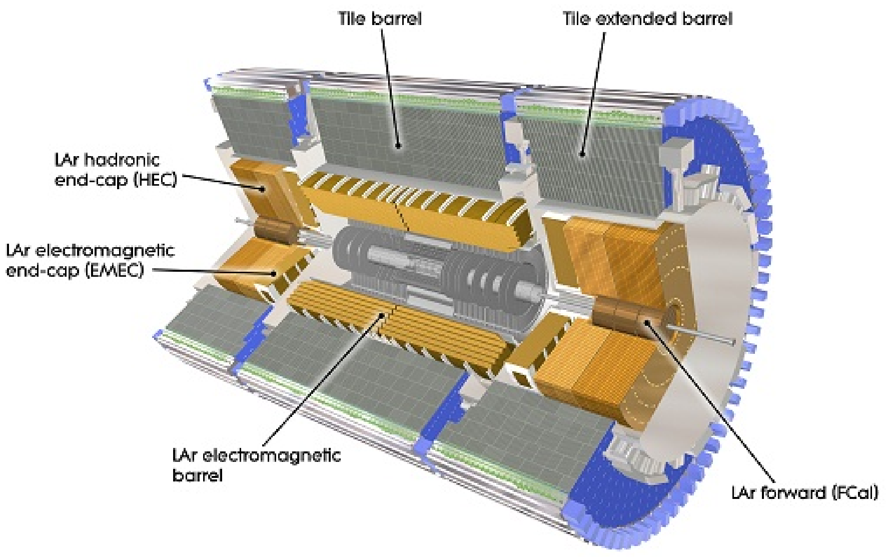
The calorimeters, displayed in figure 2.6, cover overall $|\eta| < 4.9$ with different techniques. Since it is very important to absorb most of the particles before reaching the muon system and gaining information about their deposited energy, the electromagnetic calorimeters thickness is more than 22 radiation lengths. The hadronic calorimeter is about 10 interaction length. There are different calorimeters devices next to the inner detector barrel and in the end-caps of the detector. While the calorimeters next to the barrel give a good resolution to measure electrons and photons, the other calorimeters provide coarser information. The electromagnetic calorimeter at the barrel consists of two parts separated at z = 0 with a small gap. It covers $|\eta| < 1.475$ while the two end-cap parts cover $1.375 < |\eta| < 3.2$. The electromagnetic calorimeters are *lead-LAr* detectors with accordion-shaped kapton electrodes and lead absorber plates.



Envelopes

Pixel	▭	45.5<R<242mm |Z|<3092mm
SCT barrel	▭	255<R<549mm |Z|<805mm
SCT end-cap	▭	251<R<610mm 810<|Z|<2797mm
TRT barrel	▭	554<R<1082mm |Z|<780mm
TRT end-cap	▭	617<R<1106mm 827<|Z|<2744mm

Figure 2.5: Schematic picture of the setup of the inner detector of ATLAS. All barrel and end-caps devices are outlined with radial and longitudinal position. (Image by [4])



Tile barrel
Tile extended barrel
LAr hadronic end-cap (HEC)
LAr electromagnetic end-cap (EMEC)
LAr electromagnetic barrel
LAr forward (FCal)

Figure 2.6: Profile of the ATLAS calorimeter system consisting of electromagnetic and hadronic barrel and extended barrel calorimeters. Furthermore it shows the electromagnetic (EMEC) and hadronic (HEC) end-cap and forward (FCAL) calorimeter. (Image from [5])

The hadronic calorimeter is divided into four parts. The Tile barrel covering $|\eta| < 1.0$, the Tile extended barrel $0.8 < |\eta| < 1.7$ and the end-caps and forward calorimeter with a maximum range up to $|\eta| = 3.2$. The Tile barrels are sampling calorimeters with layers of lead absorbers and scintillating active materials. In the end caps the absorber material are copper plates and LAr technology is used as active material. The forward calorimeter has three parts, with the first one consisting of copper absorber material for electromagnetic measurements and the other two of tungsten for mostly hadronic interactions. LAr is used in all three parts as active material in the FCal.

Muon Chambers

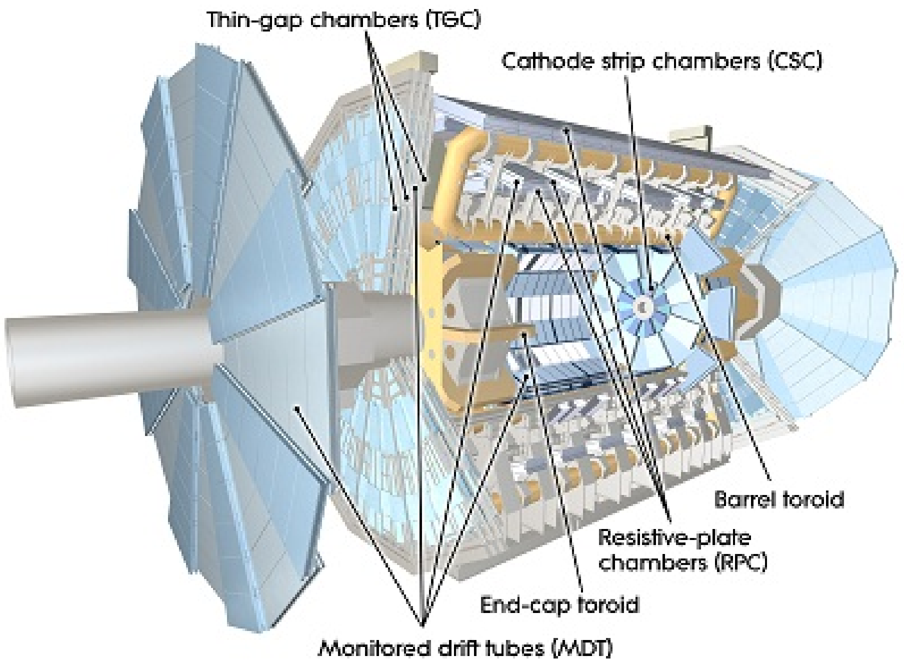

Thin-gap chambers (TGC)
Cathode strip chambers (CSC)
Barrel toroid
Resistive-plate chambers (RPC)
End-cap toroid
Monitored drift tubes (MDT)

Figure 2.7: Profile of the ATLAS detector focusing the muon system and magnets. In the barrel region RPCs are used for triggering, while in the end-caps TGCs are integrated for this purpose. In both regions regions MDTs provide information about particles momentum using fields of the the barrel or end-cap toroidal magnets. (Image from [5])

Since muons are not absorbed by the previous detector layers, they can be measured with high precision in the outer parts of the detector (see figure 2.7). Therefore, there are large magnetic fields provided by the magnet system to bend the particles. Over a range of $|\eta| < 1.4$ it is set up by the barrel toroidal magnet, while in the region $1.6 < |\eta| < 2.7$ smaller end-cap magnets are used. In the transition region the field is considered to be a combination of barrel and end-cap fields. The deflected muons are detected in muon chambers arranged in cylindric layers around the beam axis in the barrel region and in discs at the end-cap. Both systems consist of three layers. The muon system in its huge dimension is mainly responsible for the size of the ATLAS detector.

To achieve good resolutions for the precision momentum measurements there are detectors with wire drift tube technology (Monitored Drift Tube and Cathode-Strip Chambers) used in the muon system. While these two detector types provide excellent position and time information, they are not very fast. Despite of precision measurements the Muon Chambers are also used for triggering. For this purpose there are also Resistive Plate Chambers (RPC) and Thin Gap Chambers (TGC) installed, which deliver their signals with a timing precision of 25 ns.

Magnet System

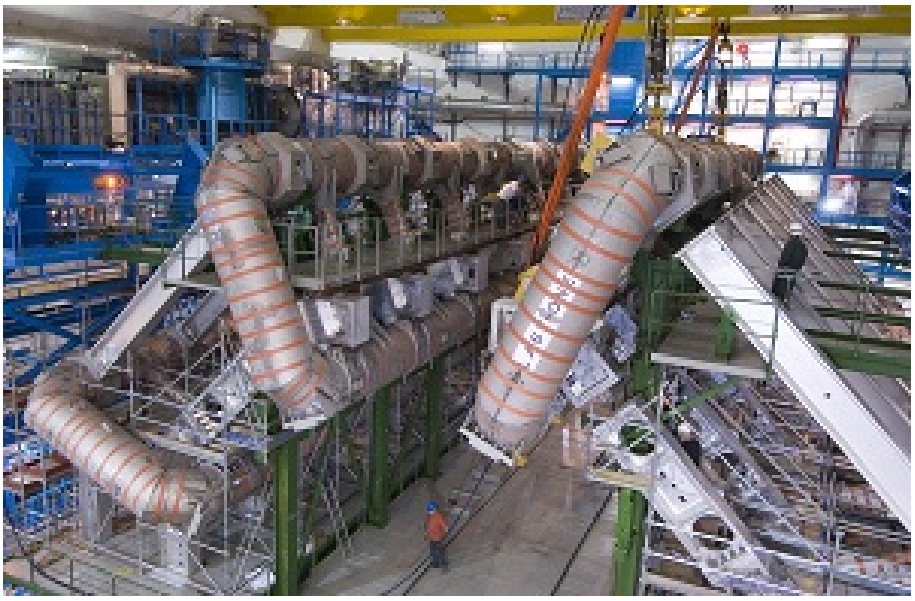


Figure 2.8: The ATLAS toroidal magnet system in its construction phase. (Image from [5])

The ATLAS detector consists of four superconducting magnets, three toroidal and one solenoidal systems. The solenoidal magnets are designed to provide a homogeneous magnet field of 2 T for the inner detector, while the barrel toroidal (figure 2.8) and the two end-cap toroidal magnets produce a field of 1 T and 0.5 T for the muon system. The magnetic fields are aligned to deflect the particles in the inner detector in ϕ and at the muon system in η direction. The field of the overlap region between the 3 toroidal magnets is very complicated but hall probes distributed all over the detector measure the field components to get precise information about the magnetic fields.

2.3 LHC upgrade

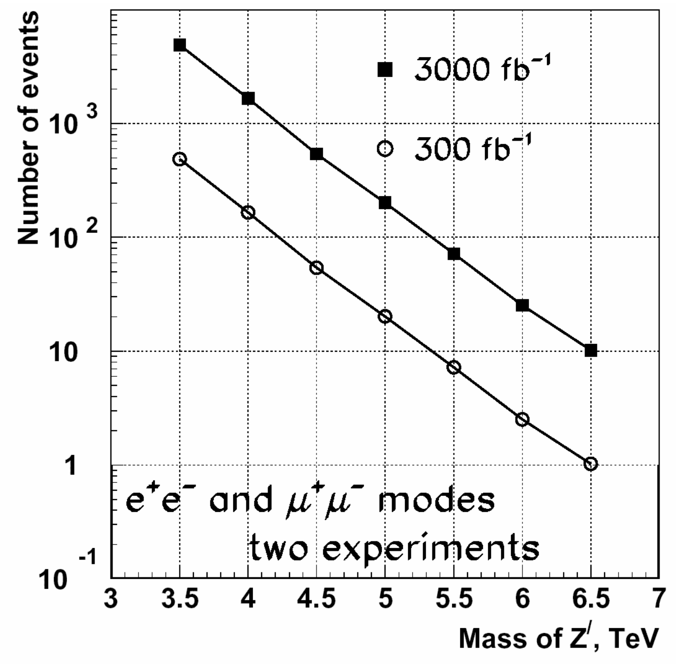


Figure 2.9: Number of events for Z' for LHC and HL-LHC with Standard Model couplings (Image from [4])

The LHC is considered to find answers to several open questions in different fields of physics or to provide some hints for further investigations. In order to exploit fully the potential of the LHC and to increase the mass reach and precision to detect new physics processes, several upgrades of the LHC are planned. It is planned to integrate a new linear accelerator at the beginning of the accelerator chain in about 2014. In about 2020, there will be a bigger shut down to upgrade the LHC.

Basically, there are two possibilities for an LHC upgrade: Either increasing the maximum energy or the luminosity. Doubling the design energy would give the chance to investigate new energy regions and find new physics which could not be detected at lower energy. But the cost of a project like this would be in the region of the LHC itself [6]. Since low statistics and huge background effects are an other limitation for detection of new physics it is possible to improve the particle accelerator by increasing the luminosity, which would be much more cost efficient. Figure 2.9, for example, demonstrates the expected event rate of a hypothetical Z' boson with Standard Model couplings for nominal LHC luminosity and for high luminosity LHC (HL-LHC) with integrated luminosities of 300 fb^{-1} and 3000 fb^{-1}. Assuming that at least 10 events are needed to discover the signal in the data, the discovery limit would be raised from about 5.3 TeV to 6.5 TeV.

To achieve the planned luminosity of $5 \cdot 10^{34}$ cm^{-2}s^{-1}, the injector chain and the interaction regions have to be improved. After the Linac2 will be already replaced by Linac4 earlier, it is necessary to increase the performance of PSB and PS. The upgrades of PSB and PS are still in the planning phase but increasing their injection energy is under discussion. An enhancement of the injection energy from 23 GeV to 50 GeV allows higher bunch intensities up to $3.6 \cdot 10^{11}$ protons per bunch when keeping the interaction rate of 40 MHz. A higher injection energy also leads to less injection losses, shorter injection and acceleration times and reduced filling times [6].

Furthermore, it is important to change the setup at the interaction points when changing the bunch intensities or shapes. This will basically result in replacing the quadrupole magnets by stronger devices at the interaction regions to retain stable beams after the upgrade.

Another point is the refill time, which needs to be improved. With ATLAS and CMS active, there will be about 10^{14} protons lost due to collisions per hour, draining the beam consisting of $5 \cdot 10^{14}$ protons very fast. The aim is therefore to reduce the turnaround time from 10 to 5 hours to reduce the dead times.

Another improvement is 'luminosity leveling'. In this approach the beam parameters are dynamically adjusted in order to run at a constant luminosity during a fill. This would decrease the peak power deposition in the interaction region and reduce the peak pile-up in the detectors [7].

2.4 ATLAS upgrade

The detector system was built to survive the high radiation environment at the LHC. Nevertheless some parts of the ATLAS detector will be damaged due to the radiation, so it will be necessary to maintain and exchange some parts due to defects in regular time intervals.

A major ATLAS upgrade will take place when the LHC is upgraded to HL-LHC as discussed before. Some parts of the ATLAS detector are not designed to operate at higher luminosities and will be replaced by components which are more adequate for the new setup. To demonstrate the difference of requirements of the detector due to higher luminosity, Figure 2.10 shows how the same event is seen when additional proton proton collisions take place at the same bunch crossing, describing luminosities of 10^{33} cm^{-2}s^{-1} and 10^{35} cm^{-2}s^{-1}.

One consequence of the higher amount of tracks in the detector is a rise in occupancy. This can cause problems by resulting in false track reconstruction especially at lower radii. An obvious step is to replace the inner detector [9]. Giving the new design a higher granularity would solve most of the problems, but causes other problems by adding more material, extra heat dissipation and more readout channels leading to higher computation times in the detector. Especially the extra material should be added very carefully since it could cause more secondary interactions, which would complicate the reconstruction of the event. Another problem occurs at the jet reconstruction due to higher pile-up. In the cone of a detected jet most probably some other particles will be found coming from different events. These

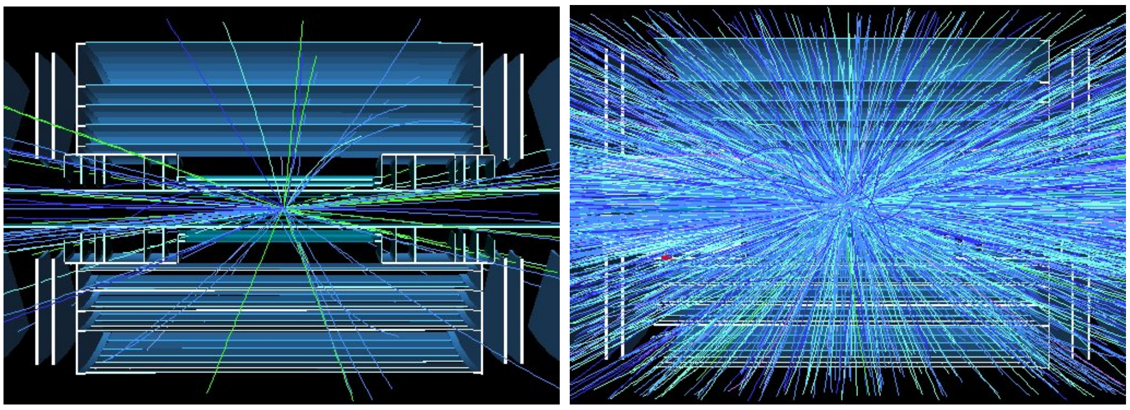


Figure 2.10: Illustration of the ATLAS detector occupancy while running at different luminosities resulting in 5 pile-up events (left) and 400 pile-up events (right). (Image from [8])

effects could be subtracted, but it would downgrade the energy resolution of the single jet event and cause generally extra noise. Because of these issues it is very important to keep the detector performance at the same level even if going to higher luminosities.

To achieve this aim, there are several upgrade geometries for the inner detector of ATLAS investigated [10]. One of the currently used designs for simulation is shown in Figure 2.11. Additional to the end-caps, there are 4 layers of pixel detectors at the barrel planned at a distance of 3.7 cm, 7.5 cm, 15.6 cm and 19.5 cm from the beam line with a pixel size of 50 μm x 150 μm. The semiconductor strip detectors will be replaced by double layer short strip and long strip detectors. The three short strip double layers are at 38 cm, 50 cm and 62 cm distance from the beam pipe with a size of 75.6 μm x 2.438 cm. The two long strip double layers can be found at 74 cm and 100 cm distance from the beam pipe with a size of 75.6 μm x 9.754 cm. There is no additional *Transition Radiation Tracker* intended like in the current ATLAS inner detector.

The calorimeters and muon chambers will basically stay the same and just get maintained while the LHC is shut down. At the moment, there are no changes planned for the magnetic fields. In contrast, the development of the trigger system, as described in the following chapter, will be a big challenge in the ATLAS upgrade.

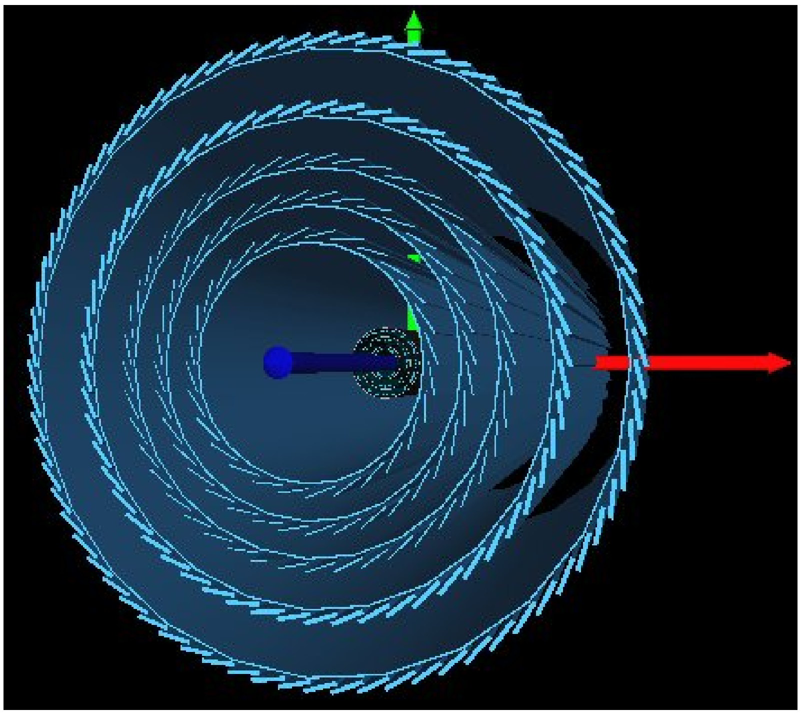


Figure 2.11: Profile of the inner detector of ATLAS with upgraded geometry after the LHC high luminosity upgrade.(Image from [10])

3 ATLAS Trigger System and Data Reduction

In this chapter, the current trigger system consisting of L1, L2 and event filter is described. The plans for upgrading the trigger system for HL-LHC are discussed followed by a description of the L1 Track Trigger realized with Content Addressable Memory chips. The chips themselves will be discussed in detail in the next chapter.

3.1 Current trigger system

The LHC has a design collision frequency of 40 MHz, meaning that there are 25 ns between two collisions of proton bunches. Since there are multiple proton-proton collision per bunch crossing the interaction rate is approximately 1 GHz [4]. Because of the large number of read-out channels, it is not possible to record data with a frequency higher than 200 Hz resulting in a required rejection factor of $5 \cdot 10^6$. This reduction is performed by the trigger with the aim of selecting a high fraction of events relevant for further analysis. The other events are dismissed and cannot be restored. The event data are stored in a buffer while the decision is made. This decision has to be made fast due to physically limited storage capacity and the high experiment frequency. To meet these constraints a special trigger system has been designed for the ATLAS experiment.

The ATLAS trigger system is operating in a hierarchical method. The data recorded by the detector systems are required to pass the L1 trigger, the L2 trigger and the event filter before being stored at the CERN computer centre. Multiple trigger chains exist searching the data for different signatures. These chains are treated individually by the three trigger systems. An overview of the trigger process is given by Figure 3.1.

3.1.1 The L1 trigger

The first step is the L1 trigger, which is illustrated in Figure 3.2. The aim of the L1 trigger is to reduce the event rate from 40 MHz to 75 kHz [4]. For this purpose, coarsened information from the muon system and the calorimeters is used. Since it is impossible to make a decision within 25 ns, the data is stored in a buffer. This buffer is physically limited and cannot store the data longer than 2.5 µs. About 1 µs is allocated by signal transition through cables in the detector complex. Another 0.5 µs is reserved for contingency, leaving 1 µs for decision making by the L1 trigger hardware. Considering that there are signals with a width of several bunch-crossings

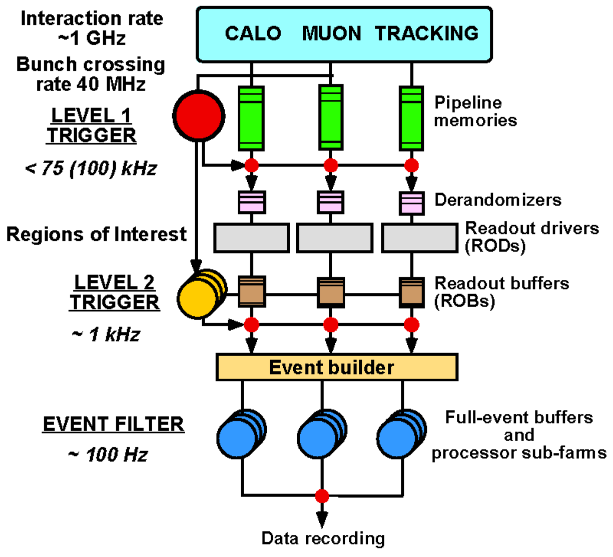


Figure 3.1: Diagram to give an overview of the ATLAS trigger and data acquisition system. The L1 trigger is using the information from muon and calorimeter system to reduce the event rate to 75 kHz. The L2 trigger reduces the event rate further down to about 1 kHz, leaving the event filter to scale the event rate down to about 100 Hz for data storage.(Image from [11])

coming from the calorimeter and other signals from the muon detector being delayed due to the huge dimension of ATLAS, the timing is very tight. To reach the decision in the given time constraint nonetheless, it is necessary to use custom-made electronics for this purpose.

The fast devices of the Resistive Plate Chambers (RPC) and Thin-Gap Chambers (TGC) are used to select events based on signals coming from the muon chambers. The L1 trigger is able to extract events with hit patterns of tracks, pointing to the interaction region, in the muon chambers with six different energy thresholds. Furthermore, data from the calorimeters are used to include information about energy deposition in the trigger decision. In this case the L1 trigger is detecting events with total high transverse energy E_T, high E_T from electrons or photons, high missing transverse energy E_T^{miss}, jets and τ lepton decays into hadrons. Also for the calorimeters different trigger thresholds are available. The final decision is made by the central trigger processor (CTP). All trigger information from the detector is sent to the CTP, which can in addition to using the decision from the sub-systems also trigger on combined signals for example by combining several muon and calorimeter signals.

If an event passes the L1 trigger, the data acquisition system (DAQ) will send the data stored in the buffer to the L2 trigger together with the coordinates of the detector region, where the signature is found. This extra information is used in the

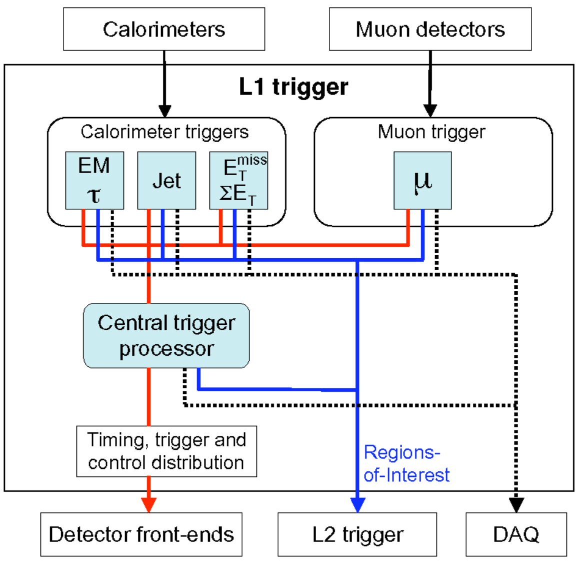


Figure 3.2: Block diagram of the ATLAS L1 trigger to demonstrate its operation method. The Muon detectors and calorimeters are used as input for the decision making, investigating the data for high p_T muons, high E_T, high E_T^{miss}, jets and τ decays. If the event passes the filter, the data will be forwarded to the DAQ, the L2 trigger and the detector front-ends.

L2 trigger to define Regions-of-Interest (ROI) and compute the next filter step there with full detector granularity.

3.1.2 The L2 Trigger

The L2 trigger operates on a part of the detector, the Region-of-Interest (ROI), defined by the L1 trigger [4]. The information, where the L1 trigger requirements have been fulfilled are sent to the ROI builder. This step is also implemented by special designed hardware to minimize the time needed by the building step. The L2 trigger itself uses commercial processing technology. The idea is to search in the ROIs for physical signatures, such as predefined shower-shapes, match tracks to calorimeter clusters or use higher thresholds and better granularity. The detector information, on which the decision is computed, can be improved step-wise to reduce the event rate to less than 3.5 kHz. The events passing the L2 trigger are sent to the event builder and event filter for the final trigger step.

3.1.3 The Event Filter

If an event passes the L2 filter, the event builder will start to collect the data assigned to the investigated event from the detector components [4]. The event builder is designed to work on multiple events in parallel to avoid timing conflicts.

The event data is sent to the event filter to scale down the event rate another order of magnitude, to less than 200 Hz. The event filter is running on a processing farm and operates similar to the L2 trigger but using more detailed detector information and more sophisticated algorithms, closer to the software used for offline analysis. The events passing this last filter step are forwarded to the CERN computer centre to store the data for further offline analysis.

3.2 Plans for trigger systems for HL - LHC

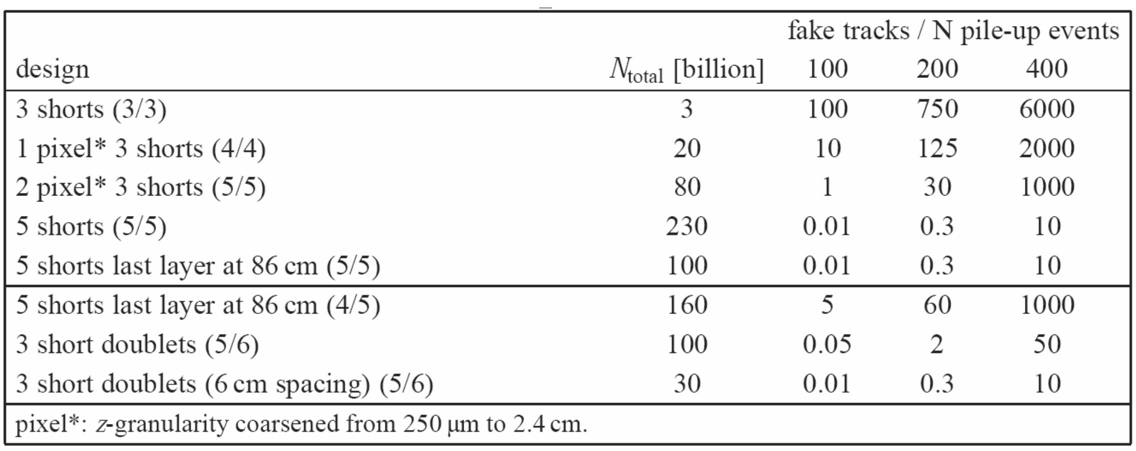
When the luminosity is increased by an order of magnitude due to the upgrade of LHC to HL-LHC, the trigger system has to be reconsidered because of the extra data rate. Since the inner detector will probably have to improve resolution to keep the occupancy below 2 %, there will be more readout channels per event and more events per time. One possibility to handle the extra pressure on the trigger system is to improve the L1 trigger to keep the rates at L2 low enough. There are two schemes to integrate an inner detector tracking device in the L1 trigger system [12]. One possibility is to use track trigger information only within the ROIs based on the L1 Calorimeter and Muon decision. This would result in longer delays, which would be necessary to be compensated with larger buffers. The second possibility is a Standalone Fast Track Trigger, which would use the information of the whole inner detector and could be combined with the muon and calorimeter system afterwards.

For example, two approaches for Standalone Track Triggers are currently investigated in Heidelberg. One approach is the 'on-detector filtering', which examines the reduction rates achieved by applying a cut on offsets or cluster sizes of tracks within the SCT [13, 14]. Another approach will be discussed in this work using Content Addressable Memory (CAM) chips to compare hit patterns of tracks in the inner detector to prestored reference patterns.

3.3 Implementation of L1 Track Trigger with CAM chips

As discussed in Section 3.1 the time constraints are very tight for the L1 trigger. There is only 1 µs available for decision making. When designing a matching scheme the easiest approach would be to build all possible combinations of detector signals and store a single bit to validate or decline the signature. The '1's and '0's returned by the silicon detectors could be used as address in a big RAM containing the validation bits. The problem of this approach is that the address size is scaling with 2^n, where n is the number of bits involved [12]. In this method every physically meaningless combination of bits is considered, which makes the total amount of patterns too high to be implemented in RAM memories.

To meet the timing constraints and to reduce the amount of data, ternary Content Addressable Memory (CAM) chips can be used. This hardware is designed for fast



design	N_{total} [billion]	fake tracks / N pile-up events 100	200	400
3 shorts (3/3)	3	100	750	6000
1 pixel* 3 shorts (4/4)	20	10	125	2000
2 pixel* 3 shorts (5/5)	80	1	30	1000
5 shorts (5/5)	230	0.01	0.3	10
5 shorts last layer at 86 cm (5/5)	100	0.01	0.3	10
5 shorts last layer at 86 cm (4/5)	160	5	60	1000
3 short doublets (5/6)	100	0.05	2	50
3 short doublets (6 cm spacing) (5/6)	30	0.01	0.3	10
pixel*: z-granularity coarsened from 250 μm to 2.4 cm.				

Table 3.1: Number of templates and fake rates for $p_T^{min} = 10$ GeV and a pile-up of 100, 200 and 400. Standard detector setup is 4 pixel layers at radii of 5, 11, 16, 21 cm and 5 short strip layers at radii of 38, 50, 62, 74, 100 cm if not otherwise stated in the entries. (Table from [12])

look-ups providing besides '0's and '1's also a *don't care bit* 'X'. This extra bit can be used to reduce the number of patterns by 2^m, where m is the number of *don't care bits* in the pattern. Additional to that the number of patterns can be reduced further by coarsening the detector granularity. In the SCT for example up to 32 strips in ϕ could be interconnected reducing the total pattern number by a factor of about $\frac{1}{32^2} = \frac{1}{1024}$ [12].

Another problem of the pattern matching approach is the risk to reconstruct tracks incorrectly due to several low energy particles producing a hit pattern, which could be interpreted as a pattern of a high energy particle. This is referred to as fake tracks. To cope with this background the total number of templates and fake tracks for different detector geometries have been investigated as a function of pile-up summarised in Table 3.1. Best results are achieved by 5 short layers with the last layers distance reduced to 86 cm and 3 short strip double layers with spacing reduced to 6 cm instead of 12 cm since the total number of templates are not above 100 billion and fake tracks below 10.

In one design option a commercial CAM chip from Netlogic (see Chapter 4.2) can be used to store more than 50000 templates with 640 bit each [15]. Assuming the detector granularity has been reduced by combining adjacent strips to lower the number of templates by a factor of 1000, there will be 100 million templates to be stored in the chips if the design of 3 short doublets with 5/6 coincidence is used. The tracking system can be divided into 2000 geometrical regions with a CAM chip in each sector to provide the L1 track trigger information [16]. Since the output of the CAM chip is the highest matching address more than one trigger threshold could be realized.

To achieve lowest fake rates possible in this scheme full detector granularity is necessary. Therefore, a refinement step consisting of two big RAMs and multiple

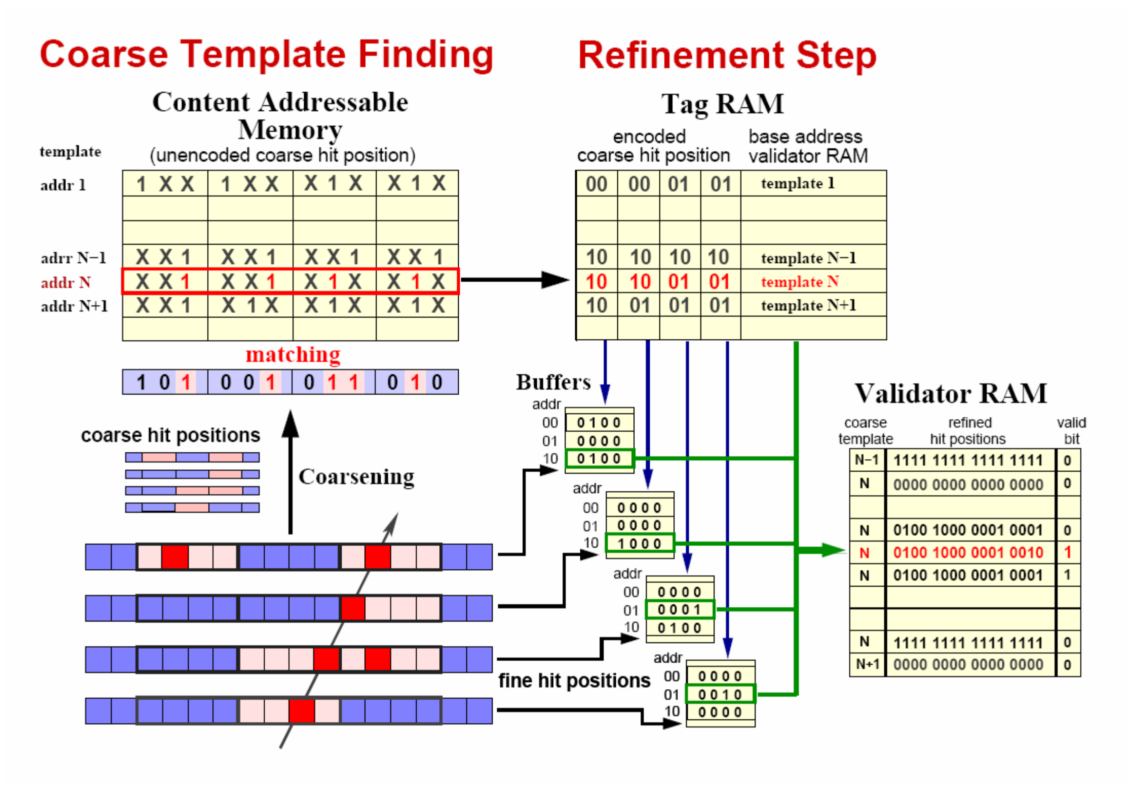


Figure 3.3: Hardware implementation of a fast track trigger with coarsened detector granularity followed by a refinement step using buffers and RAM memories. (Image from [12])

buffers could be realised after the first matching step like demonstrated in Figure 3.3. After the hit pattern is coarsened in a first step and matched to an entry with address N in a CAM chip, the address output can be used to get the encoded coarse hit position stored in a *Tag RAM*. With this information the precise hit pattern in the interesting regions could be read-out from the buffers to perform the final look-up with full detector granularity. For this purpose the base matching address coming from the CAM can be combined with the precise hit patterns to an address to get access to the final valid bit written in a Validator RAM. This process could be repeated multiple times to reduce the address space with the cost of additional decison time delays. Since CAM chips are specialised in these applications the first step could be realized within 200 ns leaving some tolerance for further operations before the final decision has to be made.

Another option for a Fast Track Trigger with KBPs is to use the full granularity of the detector in first processing level (lookup). A detector design with 3 double layers can be used, where front-end filtering on the double layers reduces the number of hits in the proposed design by a factor of about 50 [13]. By this, the fake rate is lowered to a controllable level. $3 \cdot 10^9$ templates are necessary to cover the 3/3 coincidence. Therefore, the detector should be divided into at least 10000 geometrical regions with $3 \cdot 10^5$ templates each. The number of KBPs needed to store the templates is dependent on the width of the database entries in the chip. Configuring the KBP to 80 bit entries would allow to store about $5 \cdot 10^5$ entries in one chip. Thus, a Fast Track Trigger could be realised with 10000 KBPs to provide the trigger decision in full granularity. A more detailed description of CAM chips can be found in the following chapter.

4 Knowledge-based Processors of Netlogic

In this chapter the Content Addressable Memory (CAM) chips will be introduced. Netlogic is producing memories of this technology naming them Knowledge-based Processors (KBP) [1]. An overview of available chips is given afterwards. For this work a NL9000 chip is used, which will be presented in more detail in the following sections, in particular with regard to the basic structure, the functionality and the performance.

4.1 Introduction to Content Addressable Memory

Content Addressable Memory (CAM) is a memory chip designed to perform look-up operations at the fastest possible way. Figure 4.1 shows the basic functionality of these chips. A search key is applied at the input. The search data register is connected via search lines to a database, where it is compared to each entry in parallel. The output of the CAM is the encoded highest matching address. A ternary CAM has also a *don't care bit* 'X' in addition to the bits '0' and '1'. This third bit is helpfull to reduce the necessary amount of entries since it allows to mask bits in the stored words. This means both '0' and '1' will cause a match if the searched bit is assigned to 'X'.

In order to compare the database in parallel the compare logic needs to be integrated in the basic memory cells consuming about double silicon space and extra power per cell. Considering modern semiconductor technology is aiming for smallest

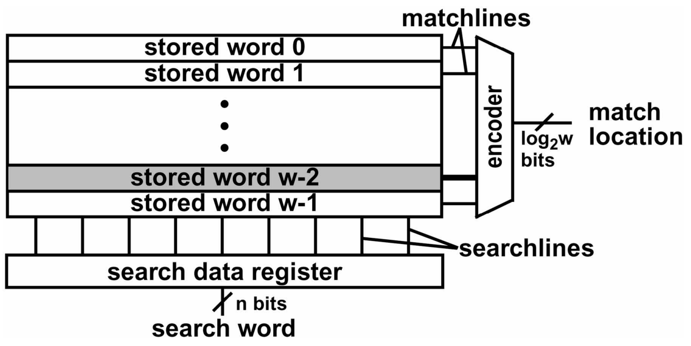


Figure 4.1: Basic components of a CAM chip. The database is searched for the key applied to the input. On the output the highest matching address is assigned. (Image from [17])

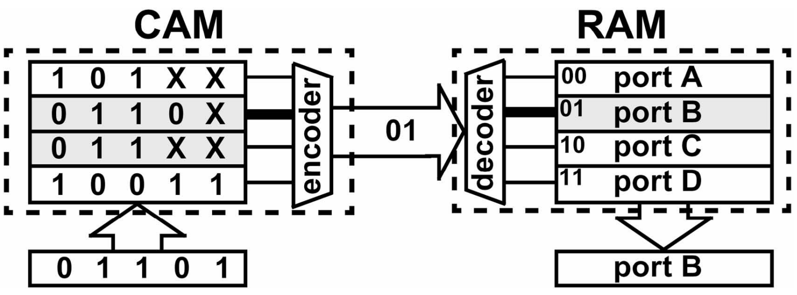


Figure 4.2: Routing system using a CAM and a RAM chip. The address is applied to the searchlines of the CAM and the highest matching result used to get the output port stored in the RAM memory. (Image from [17])

cell sizes and low power consumption to implement big memories on small scales, CAMs are only useful for high specialized applications. For example, they are used in Hough transformation, Huffman coding/decoding, Lempel–Ziv compression and image coding [17]. The main commercial usage are fast look-ups in routers. For data transfer in network processes, the data are parted into packages and transferred individually. Each package holds the destination address besides a part of the data. Thus, routers have to decide for each package to which port it need to be forwarded with lowest possible delays. For this purpose the address is applied to the searchlines of a CAM as shown in Figure 4.2. The search pattern matches two patterns of the database in this case, resulting in ’01’ on the output since it is the highest address. The match result is sent to a RAM to get the searched port of the router. Similarly the Fast Track Trigger needs the same hardware feature to perform fastest possible maths. This makes the CAM technology, which is already optimized for commercial implementations, an promising candidate for track triggers.

4.2 Netlogic Chip Families

This section is supposed to give a short overview of commercial CAM from Netlogic. Netlogic is market leader in producing specialized silicon chips for a variety of network application. For this work especially the newest generations of their knowledge-based processors (KBP) are interesting since they are basically operating as the CAM described in the section before. In addition to normal CAM functionality the KBP chips of Netlogic have some extra features. Table 4.1 and 4.2 are summarising the performance benchmarks of the NL6000, NL7000 and NL9000 chip family. The KBPs of Netlogic can be used in two modes of operation: Standard Speed and High Speed. A detailed description can be found in Section 4.3.2 for the NL9000. For each family only one chip with highest memory size is displayed. The latency of all three families, describing the time delay between search key insertion and result available at the output, can be below 200 ns depending on the mode of operation. Another important parameter is the result rate since it has to be above the 40 MHz collision frequency of the LHC. This constraint can also be kept by all

Family	Latency		Frequency	
	Standard Speed	High Speed	Standard Speed	High Speed
NL6000	71 ns	54 ns	266 MHz	500 MHz
NL7000	71 ns	54 ns	266 MHz	500 MHz
NL9000	156 ns	259 ns	300 MHz	(400/200) MHz[1]

Table 4.1: Latency and frequency of Netlogic Knowledge-based Processors while operating at Standard Speed or High Speed.

Family	Result Rate		Memory Size	
	Standard Speed	High Speed		
NL6000	33 MHz	62.5 MHz	256k x 72 bit	32k x 576 bit
NL7000	33 MHz	62.5 MHz	512k x 72 bit	64k x 576 bit
NL9000	75 MHz	100 MHz	512k x 80 bit	64k x 640 bit

Table 4.2: Result Rate of Netlogic Knowledge-based Processors while operating at Standard Speed or High Speed. The memory size is displayed in two possible configurations for the largest chip available in each family.

three families.

In this work the NL9000 chip was investigated because it represents the newest generation of KBPs of Netlogic and has the biggest memory size. Additional to that it has more features, being introduced in the following sections.

4.3 NL9000

The NL9000 is a Knowledge-based Processor (KBP) from Netlogic developed for high speed look-up operations on databases. It is able to perform searches in parallel and has four result outputs [15]. There are error detection methods implemented on the database and at the interface. It is commercially used for network applications, for example routers and switches. The communication has to be performed by an ASIC or FPGA, which will be discussed in detail in chapter 6.

4.3.1 Basic Structure

The basic structure of the NL9000 will be discussed in the following section. Figure 4.3 gives a short overview of its construction including the interface, database, registers, Context Buffer and the Key Processing Unit (KPU) of the chip.

[1]The interface operates with a frequency of 400 MHz and the core with a frequency of 200 MHz

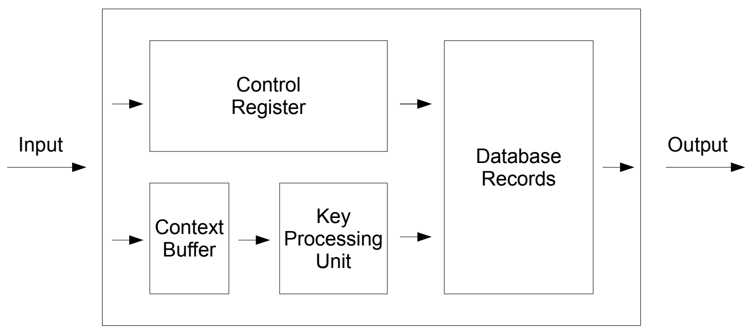


Figure 4.3: This image illustrates the basic structure of the NL9000. The most important components are the interfaces, the Control Registers, the Database, the Context Buffer and the Key Processing Unit. A compare operation always starts by writing into the Context Buffer. From there the search keys are committed to the Key Processing Unit and finally compared to the database entries.

Interface

For communication with other devices the interface of the NL9000 has 6 input buses and 4 output buses displayed in Table 4.3. To send data to the chip the signals have to be synchronous to the Data Clock (DCLK). The Data Valid (DV) pins have to be pulled high when a new operation is to be started. The type of operation is defined by the Instruction Bus (DIBUS) while the data is applied to the 80 pins of the Data Bus (DBUS). To optimize the performance during compare operations the search key buffer has an address bus, the Data Context Address (DCTX). Error detection is provided on all input buses by 2 extra parity bits, which have to be applied on the Data Bus Parity (DPR) each cycle. The results on the output buses are sent synchronous to the Result Bus Clock (RCLK), which is provided by the chip, and are flagged by the Result Bus Valid (RV) bits. The result data can be found on the Result Bus (RBUS), which are like the input buses secured by 2 parity bits on the Result Bus Parity (RPR). Up to four matching results can be returned by the RBUS in response to one compare operation. Therefore, the 48 bit of the RBUS are split up into two sections of 24 bit. Each section can provide a match flag and the address of an database entry.

Database

As shown in Figure 4.4 the database is arranged in 80 bit entries. On each address there are two entries to implement a ternary data structure, one Vality Bit (VBIT) and two parity bits for each entry. Two possibilities for ternary data structure are supported by the chip: the X - Y format and the Data - Mask format. In the Data -

Name	Type	Amount
Data Clock (DCLK)	Input	6
Data Valid (DV)	Input	3
Data Bus (DBUS)	Input	80
Instruction Bus (DIBUS)	Input	5
Data Context Address (DCTX)	Input	5
Data Bus Parity (DPR)	Input	2
Result Bus Clock (RCLK)	Output	4
Result Bus Valid (RV)	Output	2
Result Bus (RBUS)	Output	48
Result Bus Parity (RPR)	Output	2

Table 4.3: In- and Outputs on the interface of the NL9000.

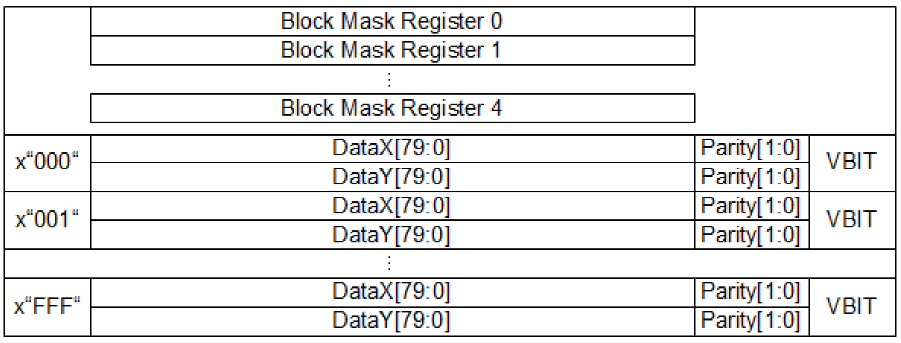


Figure 4.4: This block diagram is demonstrating the database block organisation of the NL9000. On each address there are two records to implement the ternary data structure. The entries are protected with two parity bits and can be activated by a valid bit (VBIT). Each bit can be individually masked by the block mask registers.

Mask format the data will be written into the bits of the first entry and participate in the compare operations if there is a '0' in the corresponding mask bit. If there is a '1' assigned in the mask bit the data bit will be ignored in the compare operation. In the encoded X-Y system the bit combination of '01' corresponds to an unmasked bit entry of '0' and '10' to an entry of '1'. '00' is corresponding to a *don't care* bit causing a match independent on the input bit and '11' is always a mismatch.

The entries are arranged in blocks with a 12 bit address space and five block mask registers to define a mask for all data entries in the block. The blocks are configurable into several organisations by grouping 2, 4 or 8 of the 80 bit entries to get a block setup of 4k x 80 bit, 2k x 160 bit, 1k x 320 bit or 512 x 640 bit. Four blocks are put together to a super block. Overall a NL9000 chip with 512k entries is organised into 60 blocks and 16 super blocks. To write data in all entries of a 512k x 80 bit device with a clock frequency of 300 MHz, it takes about 1.71 ms.

Device Register

Since the NL9000 is designed to be flexibly used in multiple applications some registers are necessary to store the configuration setup. The registers consist of 80 bit, which can be read by the user. For most entries the user has write permissions to change the configuration for his application. In this work only a short description of the most important registers will be given. For example, the Configuration Register defines if the chip is a single device or part of a cascaded chain. Furthermore error detection can be enabled for the input and the database. The form of the error scans and how to proceed with problematic entries can be defined for the database. In addition to that, the power mode can be appointed.

For the error detection additional registers exist for identifying where the errors occurred and which constraint failed. Other setup registers are designed to configure compare operations and the range matching function, which will be discussed Section 4.3.2.

Context Buffer

When the configuration of the NL9000 is completed, compare operations can be started. Since the width of the Input Data Bus is 80 bits but compare operations on 640 bit entries are foreseen, the search key needs to be stored temporarily. Because of that, the Context Buffer, consisting of 512 entries with 160 bits each, is implemented in the chip. For 640 bit usage search keys can be written into the buffer in multiple steps with the last step starting the compare operation.

Key Processing Unit

In the Key Processing Unit the master key in the Context Buffer can be used to create up to four search keys. For this reason the corresponding registers have to be configured and selected in the compare operation command. There are 40 bit sections selectable from the master key to be added to each of the four search keys. The keys can be applied to an unequal selection of super blocks and will be computed in parallel. To get more than one result each block can be directed to one of the four output slots returning the highest match address of the compare operation.

Error Flags and Codes

To signal the user that a problem occurred during a valid operation, there are three flags available on separate output pins. The Parity Error Output flag (PEO_L) indicates a parity error on the interface, the General Interrupt Output flag (GIO_L) indicates an error detection on the Context Buffer or database and the Phase Error Output (PHSEO_L) will be assigned to indicate a phase error. When error flags were asserted, the Result Bus will provide an error code to specify the error. Additional information can be found in the error status register with precise information about the occurrence.

Power Consumption

Information about the power consumption of the core of the NL9000 cannot be found in its manual and have to be measured (chapter 8). The power consumption of the output drivers are dependent on the time the pins are pulled high. Assuming the output pins are all used without idle cycles and randomly pulled to different values, the power consumption is about 2 W [15].

In the NL9000 a feature is implemented called Low Power Mode. By using lower granularity in the Database Mask the active power consumption can be significantly reduced [15]. The results of the measurement of this feature can be found in chapter 8.

4.3.2 Setup and Functions

Error Protection

As mentioned in the previous section, this chip is able to protect itself from bit errors. For the input, even parity is used with two bits, one protecting the even bits and the other one the uneven bits of the input buses. For the even bits, for example, this is done by sum up all bits and choosing the parity bit to get the sum even. If, due to disturbances, an uneven amount of bits flip, this will be detected and marked by setting an error flag. In analogy to that, the uneven bits of the input buses and the bits on the output buses are protected.

A scan algorithm for the database is implemented scanning the database continuously. In addition to the parity bits, which are written into the database in each write cycle, there is the possibility to activate an error correction algorithm. The algorithm uses idle cycles of the NL9000 to scan the database for errors, occurred during operation. Two options for error correction can be applied: 1 bit detect and

Categories of errors allocated by the NL9000
Data Valid Extraneous Error
Device Address Mismatch Error
Cascade Result Input Valid Irregular Error
Data Valid Irregular Error
Illegal Instruction Error
Sahasra Engine Features Parity Error
Context Buffer Parity Error
Cascade Input Parity Error
Data Input Parity Error
Database Soft Error FIFO Full
Database Parity Error

Table 4.4: Categories of errors allocated by the NL9000.

1 bit correct or 2 bit detect and 1 bit correct. To provide the error correction, the algorithm uses the 8 most significant bits of the database entries, which cannot be used for data storage anymore if the option is activated. If the scan finds an error in the database, the GIO_L flag will be set, the entry will be corrected or invalidated and the information about where the error occurred will be stored in the error FIFO[2]. This information can be read back from the FIFO.

In Table 4.4 other error sources the chip is able to recognise, are listed. If one of them occurs, an error flag will be set and the more precise information about the error source will be saved in a register.

Range Matching

The X - Y format of the database (Section 4.3.2) allows a reduction of needed entries to store similar data by means of *don't care* bits. A further reduction for some application can be achieved by the Range Matching function of the NL9000. It is designed to scale down the storage space of entries describing a range of numbers. For this function it is necessary to allocate 16 bit per 320 bit sequences. For 640 bit entries this would result in a loss of 32 bits, which might be useful for encoding IPs or Ports but less useful in a pattern match application.

Standard Speed and High Speed Mode

Aside from the operation frequency there are two modes of operation selectable: High Speed Interface Mode and Standard Speed Interface Mode. On the one side table 4.2. states that both modes of operation are able to handle the collision frequency of LHC. The result latency, on the other side, is lower when the Standard Speed is used. This is caused by the fact that the core frequency of the NL9000

[2]FIFO is the short cut for a memory chip with First In First Out policy.

Instruction	Description
idle	Idle cycle causing no reaction of the chip
NOP (No Operation)	No operation used during initialisation
Register Write and Database Write	Performs a write operation to register or database entry
Register Read and Database Read X	Performs a read operation to register or database encoded 'X' value
Database Read Y	Performs a read operation to database encoded 'Y' value
Context Buffer Write	Writes data to Context Buffer
Context Buffer Write and Compare 1	Writes data to Context Buffer and starts a compare operation of up to 320 bit wide
Context Buffer Write and Compare 2	Writes data to Context Buffer and starts a compare operation of up to 640 bit wide

Table 4.5: Instruction Set of the NL9000.

is lower when operating in High Speed mode. Since the Standard Speed Mode fits best to the requirements of a Fast Track Trigger the further applications will be discussed mainly for Standard Speed Mode.

Instruction Set

In Table 4.5 the Instruction Set of the NL9000 is listed. There are overall 8 valid instructions to write and read the database, configure the chip and perform compare operations. Since the *Context Buffer Write* and the *Context Buffer Write and Compare2* instruction are the most important instructions for this work, they will be discussed in more detail.

Figure 4.5 shows the *Context Buffer Write* timing diagram. The NL9000 is designed to get input data with a double data rate, meaning that data are taken on the rising and the falling edge of the Data Clock (DCLK). It is necessary to start the operation on the rising edge of the DCLK by applying a '1' to the Data Valid (DV) pins and the corresponding signals on the rest of the input buses. The data has to be stable when the DCLK toggles from '0' to '1', otherwise an error flag will be set. To define the storage space where the search key is to be placed, the address is applied to the DCTX bus. On the DBUS the bit pattern is committed in two 80-bit-slices. The operation is completed within one clock cycle and since the chip is pipelined, the input buses can be used for other operations. After having passed the result latency, the Result Valid pins are pulled high on the rising edge of the Result Clock (RCLK). On the Result Bus 48 bits of '0' are applied. This instruction

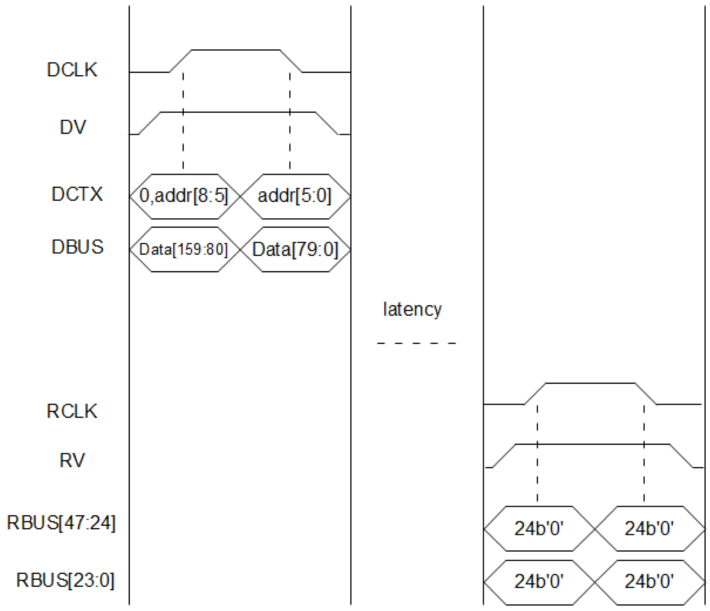


Figure 4.5: Timing diagram of the *Context Buffer Write* instruction. When the Data Clock (DCLK) is on the rising edge the bits of the buses need to be stable. The beginning of the instruction is identified by the Data Valid (DV) bits and the corresponding instruction code on the Instruction Bus.

is important for the implementation of a Fast Track Trigger because a 640 bit search key cannot be written into the buffer within one instruction. The procedure will be to use the *Context Buffer Write* two times to write the first 320 bit into the buffer and use the *Context Buffer Write and Compare2* instruction afterwards.

The *Context Buffer Write and Compare2* instruction is shown in Figure 4.6. In contrast to the *Context Buffer Write* instruction it needs two clock cycles. On the first cycle the signals are basically analog to Figure 4.3, but in the second cycle another 160 bit are transmitted to the Context Buffer. After the result latency has passed, the RV bits are set to '1' and the results are available on the RBUS.

To configure the chip and write the database the *Register Write and Database Write* instruction has to be used. For the compare operation, multiple modes of operation are available, which have to be set up before:

- The database can be organized in 80, 160, 320 and 640 bit sequences. Search keys need the same width as the data to be applied to the corresponding block. The organisation of both can be adjusted in their configuration registers.

- The master search key, which is committed to the Key Processing Unit with the compare operation, can be used to construct up to 4 search keys to perform parallel searches. A search key has to be assigned to each activated super block.

- On the output up to four results are returned. The default setup will link all blocks to result port 0 and, if the search matches to one or more entries, it will return the highest matching address. Each blocks can be allocated to a result port individually, providing the option to divide the database into four parts with a unique output of each part. The mapping of blocks to output ports is independent of the search key mapping.

- For each block a Block Mask has to be selected, which allows to compare only to a selected part of the entries or disable problematic regions.

Device Initialisation

Before the normal operations on the NL9000 can be started, the device needs an initialisation sequence. Two initialisation sequences exist for the chip: the *System-On* reset is necessary at the start up and resets everything to default values, while the *Core Logic Reset* only resets the control registers. Both resets need a stable clock and applied reset signals to the pins followed by a sequence of consecutive operations on the Data Bus. The *System-On* reset takes about 5 ms, the *Core Logic Reset* about 2.5 ms.

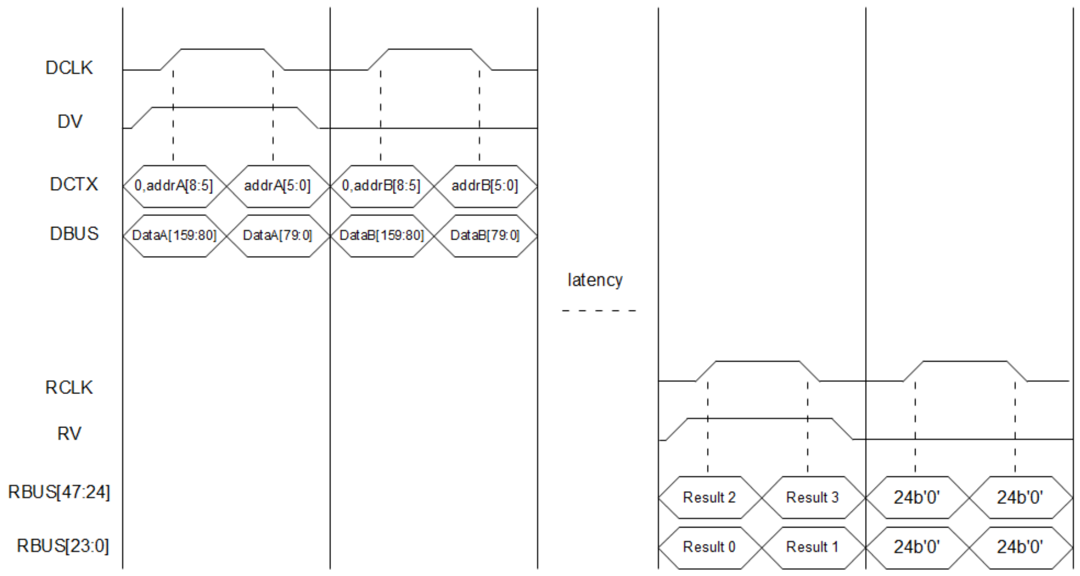


Figure 4.6: Timing diagram of the *Context Buffer Write and Compare2* instruction. When the Data Clock (DCLK) is on the rising edge the bits of the buses need to be stable. The beginning of the instruction is identified by the Data Valid (DV) bits and the corresponding instruction code on the Instruction Bus.

4.3.3 Performance and Uncertainties

The frequency, memory size, result rate and latency of the NL9000 is shown in Table 4.1 and Table 4.2. To explain these values in more detail, it is important to know that the device is operating in a double data rate. This means that valid data can be applied on the rising and the falling edge of the clock. Together with the DCLK maximum frequency, 640 bit usage and the width of the DBUS (80 bit) this defines the decision rate of 75 MHz for Standard Speed and 100 MHz for High Speed Mode. Highest reachable decision rate would be possible by using only 80 or 160 bit search keys with four parallel compares resulting in 1200 million decisions per second.

The latency is basically given in clock cycles the operation needs to get through the pipelined electronics. The number of clock cycles is dependent on the memory size of the NL9000 chip. For a device with 1024k records of 40 bit the latency is 43 ± 4 for Standard Speed and 74 ± 4 for High Speed. In this thesis a chip with 512k records is used with a latency of 31 ± 2 for Standard Speed and 56 ± 2 for High Speed Mode. Depending on the frequency this translates in a minimum latency of 143 ns $\pm$ 13 ns for Standard Speed and 246 ns $\pm$ 13 ns for High Speed Mode of the chip with 1024k entries. The chip with 512k records has a latency of 103 ns $\pm$ 6.6 ns for Standard Speed and 186 ns $\pm$ 6.6 ns for High Speed Mode.

5 Simulation of the NL9000 Processor

This chapter describes the simulation of the behaviour of the NL9000. After giving an overview of the simulation tools and the encrypted Verilog model, the test runs and results will be discussed.

5.1 Simulation Tools

For simulating the behaviour of integrated circuits *ModelSim* of Mentor Graphics can be used [18]. It is able to calculate the behaviour of hardware models based on VHDL[1] or Verilog[2] close to reality. Since the coding of hardware programs is commonly done in VHDL but the model of Netlogic is in Verilog, *ModelSim* is the optimal choice because it can connect VHDL and Verlilog codes.

In addition to an editor and compiler for VHDL files, ModelSim provides a graphical display named *Waveform* to monitor the output and input signals produced in the process. The visualisation of test results is helpful for debugging applications during the development phase.

5.2 Encrypted Verilog Model

The encrypted Verilog model is coded by Netlogic and provided under non-disclosure agreement. The model is designed to have the same inputs, outputs and configuration pins like the chip in hardware. It is supposed to show realistic reactions when signals are applied to its interface. How close the behaviour of the model gets to reality will be discussed in Chapter 6.

5.3 Setup and Test Runs

To start a simulation of a component in VHDL, it is necessary to write a testbench file. In this file the input and output slots of the component, which is to be tested, has to be described. The functionality of the component will be generated by the separate file of Netlogic. Furthermore, internal signals can be defined to apply data on the input ports and monitoring the output ports. These signals must be connected to the corresponding ports of the component in the port map. After the component

[1]VHDL is a short cut for Very High Speed Integrated Circuit Hardware Description Language. It is a programming language especially designed for hardware.

[2]A hardware description language similar to VHDL

Parameter	Value
Clock Frequency	300 MHz
PLL Range	250 MHz to 400 MHz
Mode Select	Standard Speed
Database Error Correction	disabled
Parity Checks	enabled
Low Power Mode	disabled
RBUS config	dual RBUS enabled [3]
Single Device / Cascaded	Single Device
Device ID	"00"
Data Alignment to Clock	Center Aligned
Input Buses Default	all '0'

Table 5.1: Default Settings for the NL9000 simulation.

is defined, it can be instantiated multiple times and connected individually. For this thesis, it is sufficient to instantiate it once and connect the signals to the ports to test a single device.

Default Settings

During this work several functions of the NL9000 were tested and, therefore, multiple files with different setups were constructed. Nevertheless some adjustments like the system clock, the configuration applied to the pins and the setup of control registers can be used for all tests. The default settings are listed in Table 5.1.

Instruction Library

When the functionality of the NL9000 is tested in a VHDL simulation, a lot of instructions have to be applied multiple times. In VHDL a library with functions can be developed to avoid mistakes by generating repeatedly appearing code parts. In addition to that, it is also reasonable to implement an algorithm to calculate the parity bits automatically.

In context of the diploma thesis, a library file 'function_package.vhd' has been designed. It provides a function to calculate the parity bits for the input bus plus procedures for the initialisation and all valid instructions of the NL9000. An extraction of this code can be found in the appendix (C.3).

The simulation of the NL9000 is performed by a process in VHDL. In this process any combination of bits can be applied to the ports of the tested component. By including the library into the VHDL code of the simulation, the initialisation and the instructions can be executed with one command. Naturally, the simulation process starts by initialising the chip with the function call initial. The clock period,

[3]if disabled, RBUS[47:24], RV_1 and RPR[1] are tri-stated

Test Run	Description
Database configuration	Simulates setup of the control registers, the database and the Context Buffer of the NL9000
80 bit compare	Writes 80 bit database entries and starts 80 bit *Context Buffer Write and Compare1* instructions.
640 bit compare	Writes 640 bit database entries and starts 640 bit *Context Buffer Write and Compare2* instructions.
parallel compare	Writes 80 bit database entries and starts *Context Buffer Write and Compare2* instructions with four parallel searches of 80 bit each.

Table 5.2: Overview of NL9000 test runs.

DV, DIBUS, DCTX, DBUS, DPR and the reset signals have to be committed as arguments to the function. After that the simulated NL9000 is ready to get instructions on its input buses. The instructions can also be applied by a function call. For example, a *Register Write* instruction can be generated by the function call of 'reg_write'. The arguments are the clock period, the input buses, the address of the register and the data. Analog to that the rest of the instruction set can be used.

Overview of Test Runs

In Table 5.2 the four basic tests, which are necessary to get an overview of the functionality of the NL9000 are listed. The configuration test is about reading the default settings of the chip and writing own configurations into the registers. Furthermore, the Database and Context Buffer are tested. The *80 bit compare* test needs, besides the setup of the previous test, the configuration of the Key Processing Unit. Entries written into the database can be searched by the *Context Buffer Write and Compare1* instruction. The *640 bit compare* test is basically analog to the previous but using 640 bit entries and search keys. The *parallel compare* test will use a different configuration of the Key Processing Unit to search the database with four 80 bit keys in parallel. These keys will be constructed from a 320 bit master search key.

The simulation process of the database configuration test starts with the initialisation of the NL9000. After that, the control and identification registers are read. This gives the user information about the chip and its default setups. The next step is to write the configuration of Table 5.1 into the configuration register. Each write instruction will be read back to check correct operation. Then, the blocks are configured to 80 bit usage and enabled. Database Write X - Y and Read Instructions are performed to test operations on the database. Finally, Context Buffer Write instructions are performed.

The simulation process of the *80 bit compare* test also starts with initialising the chip and configuring the default setup of Table 5.1. Here, multiple blocks are used and are therefore activated and set up for 80 bit usage. Furthermore, the block mask registers, the block mask select and key construction register are configured for a simple 80 bit compare. In the activated blocks of the database multiple 80 bit patterns are written matching to the search keys applied by the *Context Buffer Write and Compare1* instruction. Some entries are inserted into the database several times to provoke multiple matches.

The setup of the *640 bit compare* test is done similar to the tests before, but the database and search key are configured to be used with 640 bit width. When 640 bit patterns are supposed to be written into the database, there are 8 write instructions necessary for each of them. Since the *Context Buffer Write and Compare2* instruction writes 320 bit into the Context Buffer there are 320 bit left, which have to be written by two *Context Buffer Write* instructions before. This results in three instructions applied to start one compare operation.

At the *parallel compare* test, 320 bit search keys are used but the database is configured to 80 bit usage. The parallel search register is set up to split up the 320 bit master search key into 4 equal parts of 80 bit. These four parts are used as search keys on four parts of the database. Each of these database parts will be mapped to an individual output port. The parallel compare register and the key construction register have to be configured for this usage in addition to the previously introduced setups.

5.4 Results

The simulation of the NL9000 will be helpful when the chip is tested in hardware because the correct operation of signals on the input pins and the configuration of the device pins and control registers can be tested before. The configuration of the chip can therefore be excluded as error source when debugging code for the hardware. The library developed for this simulation can be used as blueprint to integrate the Instruction Set into the code for the hardware. Furthermore, latencies and result rates can be extracted for different setups. The performance data will be compared to the performance of the hardware tested in Chapter 6 to judge about the quality of the encrypted Verilog file.

From the database configuration test, the time necessary to initiate the NL9000 can be extracted. Figure 5.1 shows the 'Waveform' of ModelSim when the test is completed. The green lines are signals of the input and output buses, the blue ones are the three error flags. The end of the initialisation process is marked after 5.202 ms.

An extraction of the *80 bit compare* test is displayed in Figure 5.2. In the 'Waveform' the start of the compare operations and the start of results on the RBUS are

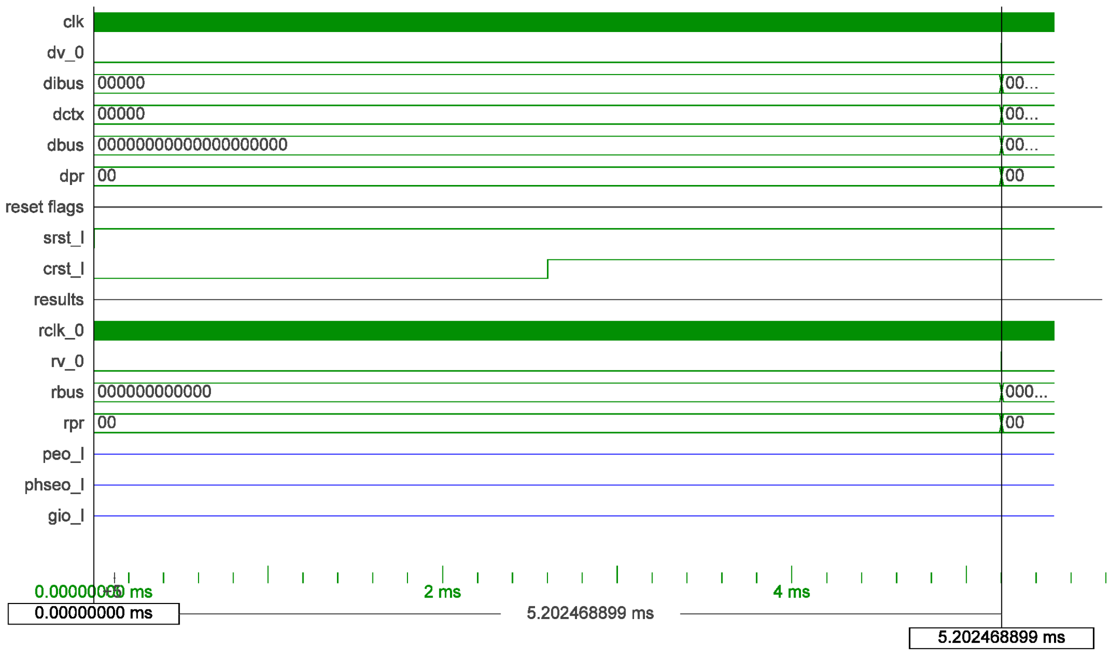


Figure 5.1: Waveform of ModelSim showing the initialisation process. The start and end of the initialisation are marked with the black markers. The time until the chip finished the initialisation is 5.202 ms. The green signals are the inputs and outputs of the NL9000.

marked. The time delay is 96.645 ns, which is within the lower limit of the 512k NL9000 device. The read instruction on the database and mask registers returns 'X' values before something is written into it. 'X' stands for 'forcing unknown' in *std_logic* signals meaning that the database is uninitialised. The response of the read instruction on the database mask register can be seen in Figure 5.2 in the lower left. It is not initialised by default and returns 'X', which is marked in red. The result rate is illustrated in Figure 5.3. The beginning of each response of a compare operation is tagged by a black marker. The time between two responses is 3.332 ns corresponding to a frequency of 300 MHz.

Figure 5.4 and Figure 5.5 are showing the result latency and the result rate of the *640 bit compare* test. The result latency is constant as expected while the result rate is dependent on the number of clock cycles needed to apply the search key to the NL9000. Since there are four clock cycles necessary instead of one in this case, the result rate dropped to 75 MHz.

If the silicon technology improves further in the next years, the width of the input buses can be large enough to compute more than one region of the detector. Therefore, the parallel compare test demonstrates that it is possible to divide the search pattern into four patterns of 80 bit. Each of these patterns could represent a region of the detector on which the track triggering is performed. The result latency would stay constant in this scenario but the result rate would improve to 600 MHz.

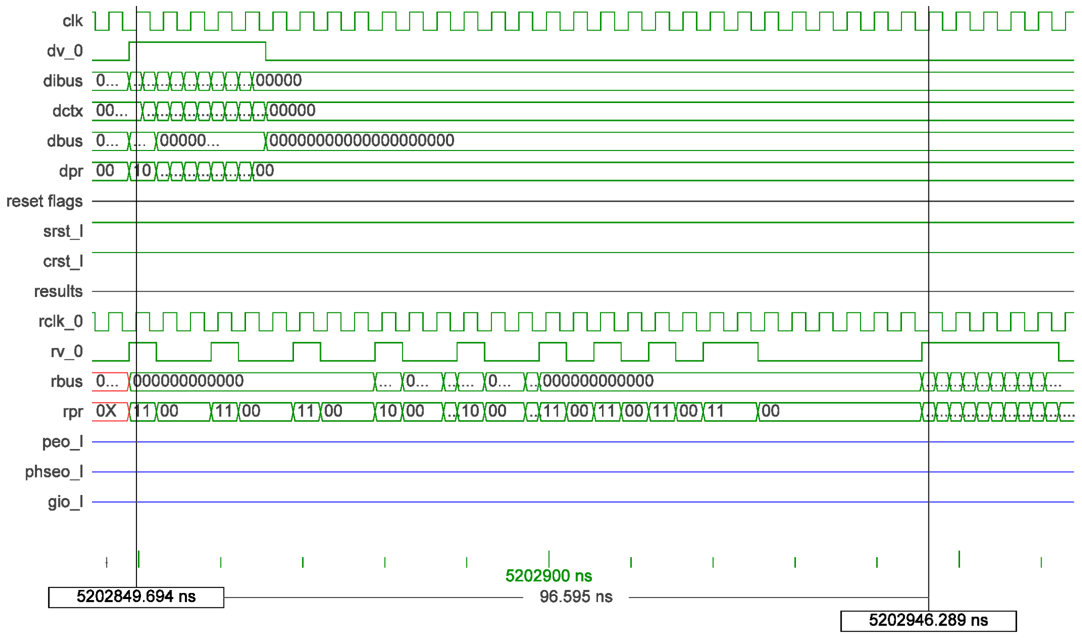


Figure 5.2: Timing simulation of the 80 bit compare. The markers are at the beginning of the compare operation and at the beginning of the response. The response time is 96.645 ns.

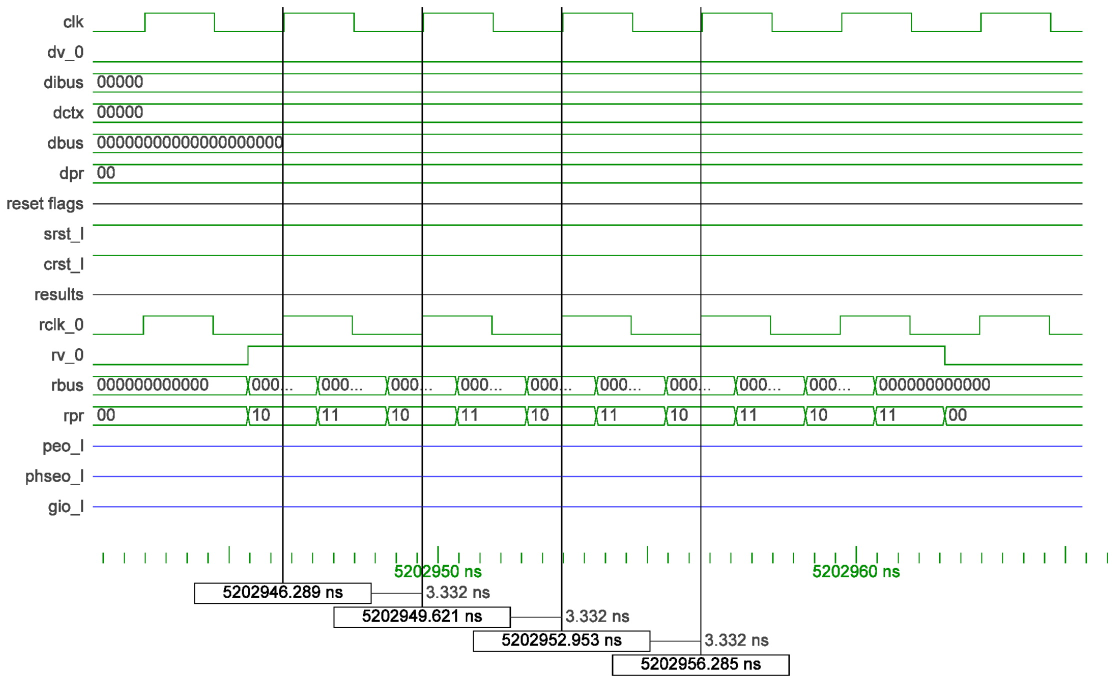


Figure 5.3: Timing simulation of the 80 bit compare. The beginning of each response to a compare operation is marked. The time delay between two results is 3.332 ns.

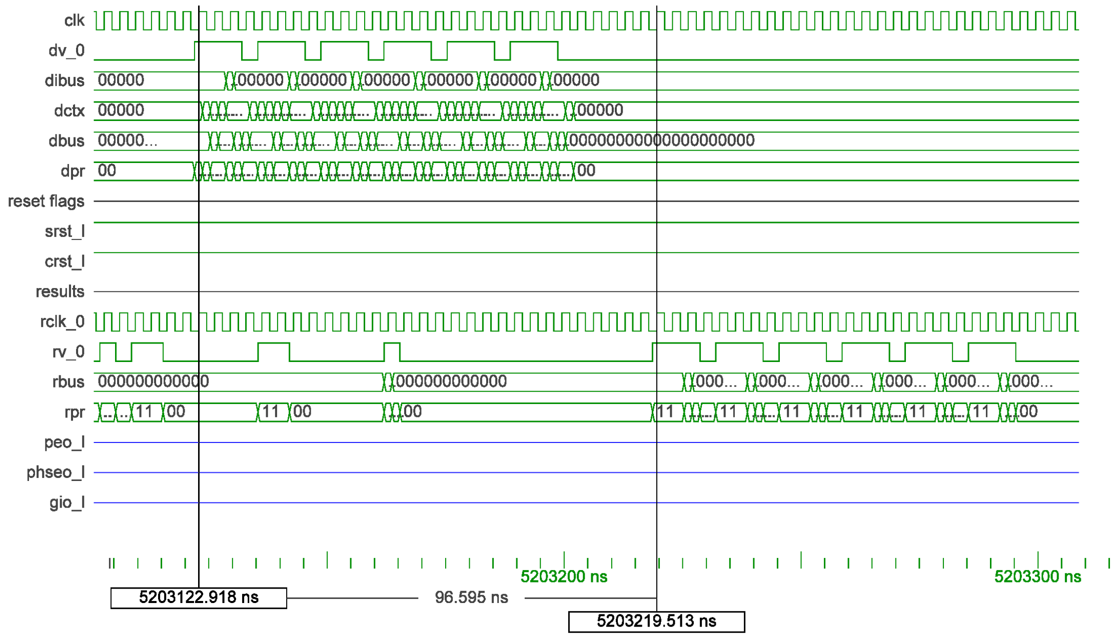


Figure 5.4: Timing simulation of the 640 bit compare. The markers are at the beginning of the compare operation and at the beginning of the response. The response time is 96.595 ns.

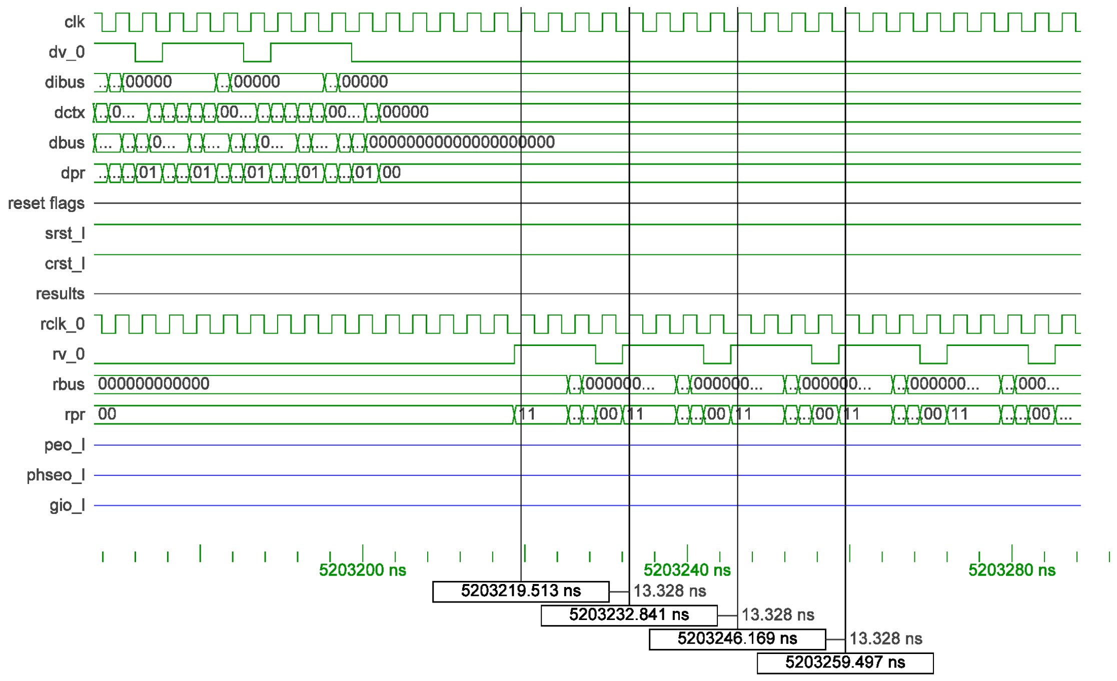


Figure 5.5: Timing simulation of the 640 bit compare. The beginning of each response to a compare operation is marked. The time delay between two results is 13.328 ns.

By this, one chip would compute the track triggering of four regions on the detector to reduce the amount of chips also by a factor of 4. Theoretically, another factor of 2 could be gained by using only one clock cycle instead of two to apply the search key into the chip. This would presume the input buses being large enough to apply data of four detector regions into the chip within one clock cycle.

6 Desription of Test Hardware Components

This chapter details the hardware necessary to implement a L1 track trigger with a NL9000. Since the chip needs special signals on its input pins, further hardware devices are needed to communicate with the chip. For this purpose a FPGA[1] or an ASIC[2] can be used.

To perform tests with the NL9000 the Development Platform HTG-100GIG [19] has been used. There is a FPGA of Xilinx on this platform, which needs to be programmed to perform the tests with the NL9000. The software and tools for that are topic of the next chapter.

6.1 Field Programmable Gate Array

A Field Programmable Gate Array (FPGA) is a commercially fabricated chip allowing flexible implementation of logic operations. The chip provides I/O blocks with pins supporting various I/O standards [20]. The core of the chip consists of programmable logic cells, programmable routing systems and clock resources. Figure 6.1 shows an internal FPGA design used to implement re-programmable hardware. The logic blocks are connected to wires routing the signals through the chip. These wires can be connected to other wires or logic blocks allowing the user to implement complex logical operations in the chip.

Figure 6.2 illustrates the inner structure of a logic cell. Basically, logic cells consist of look up tables (LUT), flip flops (FF) and multiplexers (MUX). Control signals direct the input signals to the components with the intended functionality. From there, they are applied to the outputs of the block and routed, if necessary, to the next block. In a Xilinx Virtex 5, for example, a SLICEM logic block consists of 4 LUTs with 6 inputs each, routing resources and 4 FF.

In addition to that, there are separate clocking wires and special resources to edit clocks within the FPGA. Another feature of FPGAs are block RAM components integrated into the core. They can be used to store data in memory resources or to build components like FIFOs or CAMs in the FPGA. The memory space of these embedded resources are limited by the size and number of components in the FPGA

[1]Field Programmable Gate Array: A silicon chip with integrated circuits, which are individually programmable

[2]Application-Specific Integrated Circuit : A silicon chip specially designed for an specific application

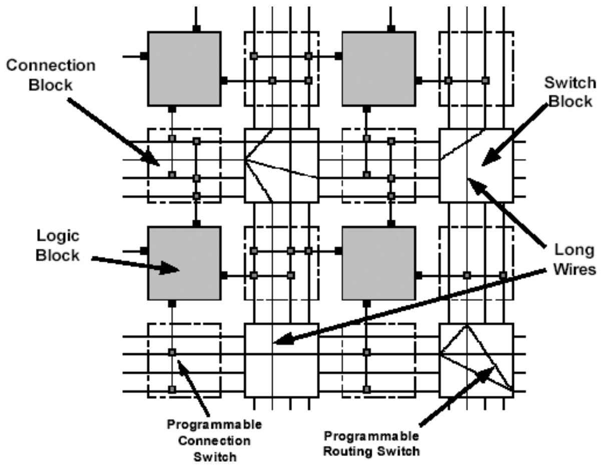


Figure 6.1: Core of a FPGA consisting of logic blocks and routing system.(Image from [21])

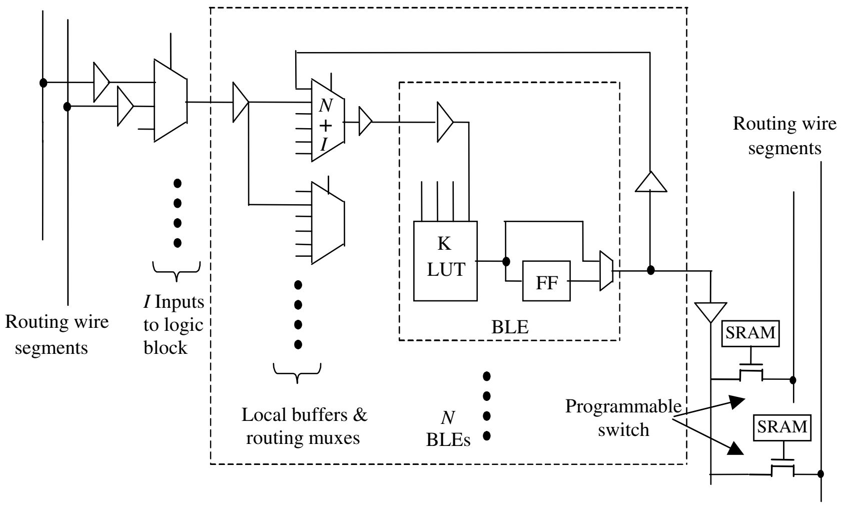


Figure 6.2: Basic structure of a logic cell in a FPGA.(Image from [20])

and cannot compete with specialised hardware. Nevertheless, a FPGA is the optimal module to develop a controller for the NL9000.

6.2 Development Platform HTG-100GIG

The HTG-100GIG [19] is a development platform produced by HiTech Global to alleviate the draft of high speed subsystems. It provides functional blocks to design user specific solutions for high performance applications. In addition to that the components of the board like for example the Knowledge-based Processor of Netlogic can be accessed easily by using one of the implemented interfaces.

6.2.1 Components and Architecture

An overview of the components integrated into the development board is given in Table 6.1. The main component on this board is the Virtex 5 FPGA of Xilinx [22], which allows to develop prototypes for a wide range of applications. Due to the flexible usage of the logic cells in the FPGA, these applications can be tested and debugged safely. Table 6.2 shows the list of the integrated circuits of the Virtex 5. The aim of this work is to build up a track trigger system by using the features of

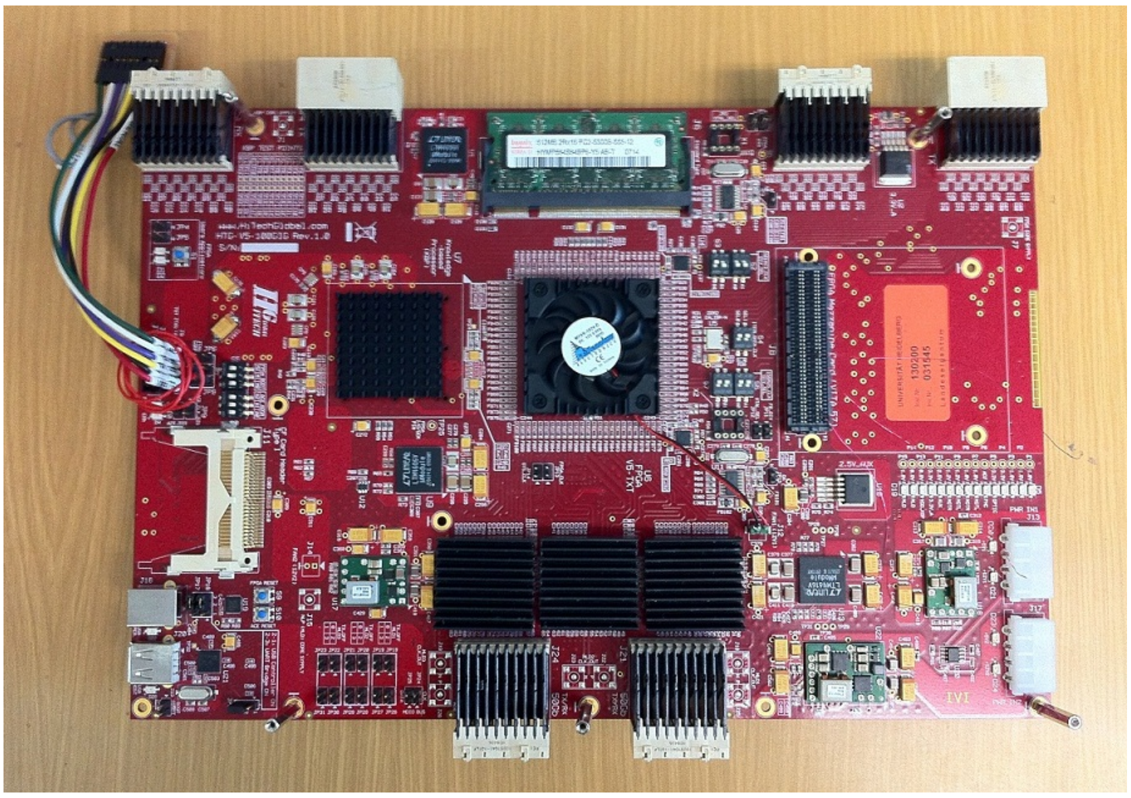


Figure 6.3: Picture of the front side of the Development Platform HTG-100GIG [19].

the FPGA combined with the Knowledge-based Processor of Netlogic. The function of the FPGA is to control the signals on the pins of the NL9000, while the NL9000 is considered to provide the search results. The pins of these two devices are therefore connected directly on the board. Furthermore, memory devices and several interfaces are provided by the development platform. For this thesis, the USB to UART Bridge was used since it is the simplest way to establish the communication between the computer and the FPGA.

Number	Component
1	Xilinx XC5VTX240T-2FF1759 FPGA
1	Knowledge-based Processor - NL9000
3	Netlogic 10GIG PHY (NLP10142)
4	FCI Airmax Header and Receptacle Connectors (Interlaken interface)
2	FCI Airmax Receptacle Connectors (Extender Module interface)
2	DDR-II SO-DIMM
1	SystemACE (Configuration)
1	USB 2.0 Host/Device
1	USB To UART Bridge
1	FMC (FPGA Mezzanine Connector)

Table 6.1: List of components on the Development Platform HTG-100GIG.

Feature	Number
Slices	37440
Logic Cells	239616
CLB Flip-Flops	149760
Maximum Distributed RAM (Kbits)	2400
Block RAM/FIFO w/ECC (36Kbits each)	324
Total Block RAM (Kbits)	11664
Digital Clock Managers (DCM)	12
Phase Locked Loop (PLL)/PMCD	6
Maximum Single-Ended Pins	680
Maximum Differential I/O Pairs	340
DSP48E Slices	96
PCI Express Endpoint Blocks	1
10/100/1000 Ethernet MAC Blocks	4
RocketIO^{TM} GTX High-Speed Transceivers	48
Configuration Memory (Mbits)	65.8

Table 6.2: List of features of the Virtex 5 TX240T FPGA of Xilinx.

7 Software for the Controller and the Interface

The challenge of this work is to send search patterns for the NL9000 at high frequency with the correct timing. The NL9000 is controlled by the Virtex 5 FPGA of Xilinx, which can operate up to 500 MHz maximum speed. This FPGA can be programmed to provide the instructions to the NL9000 like simulated in chapter 5. Furthermore, it is necessary to communicate with the FPGA to control its operation and read back the results. In context of this thesis, software for this purpose has been developed and will be introduced in the following sections.

7.1 Software Tools

In this work the software tool *Xilinx ISE* has been used. The program includes an editor, can compile VHDL code and implements its functionality into a FPGA [23]. In addition to that, it can perform timing analysis to help the user to find critical timing paths. The routed program in the FPGA can be displayed by the integrated FPGA editor to show the paths and used components in the chip. In addition to that, the integrated simulation tool *ISim* in *Xilinx ISE* can simulate programs in different stages of development.

Labview has been used for coding an user interface for the computer [24]. This is a development environment from National Instruments using a graphical programming language. It is made to create special user interfaces for data acquisition and instrument control.

7.2 Software Modules for the Development Platform

For the implementation of a test application, simulating the behaviour of a Fast Track Trigger in hardware, the components of the development board are used as shown in Figure 7.1. The user can control the behaviour of the components by an user interface on a connected computer. The information transmission from PC to FPGA is realised with USB communication. The USB port is connected to a USB to UART Bridge on the board to simplify the signals received by the FPGA. The FPGA communicates directly with the Knowledge-based Processor (KBP) of Netlogic. A program is installed on the FPGA, which can apply signals to the NL9000 and read back the responses flexibly.

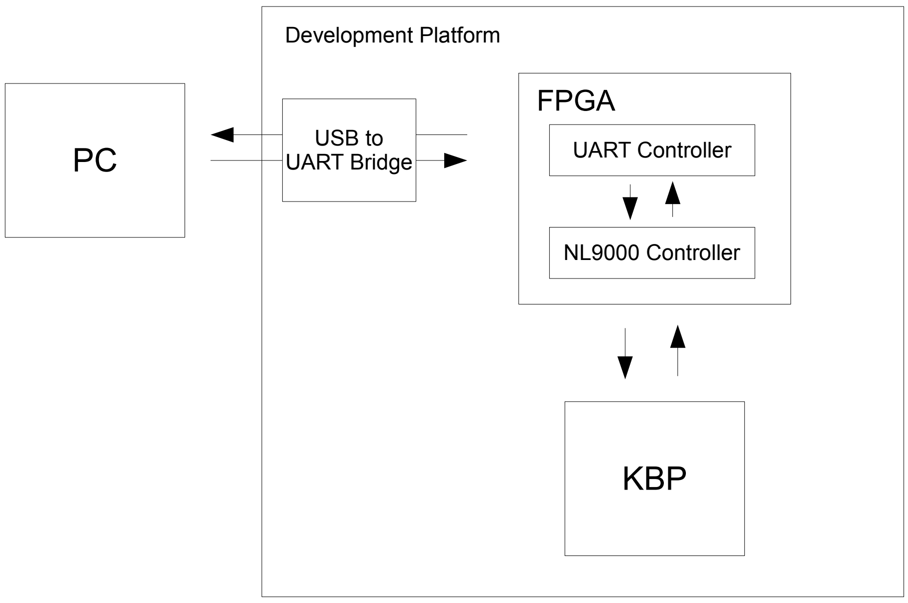


Figure 7.1: Block Diagram showing the hardware devices of the test application. It can be controlled by the computer of the user. The interface between board and PC is the USB to UART Bridge. The function of the FPGA is to apply signals with correct timing to the NL9000 of Netlogic (KBP).

7.2.1 Initial Startup of the Xilinx Virtex 5 and the NL9000

For the initial startup several modules are necessary to be programmed in the FPGA. In this section the elementary modules are described to get first responses from the NL9000.

Description of the UART Controller

The first issue, during successful implementation of a controller into the Virtex 5, is to establish the communication between computer and FPGA. The USB to UART Bridge on the development platform provides UART signals to the FPGA like demonstrated in Figure 7.2. It is a serial signal beginning with a low start bit followed by 8 bits of data. The signal is pulled low for a '0' and pulled high for a '1' when sending data. Between each bit a specific time passes, which is referred to as *bit time*. The end of the data transmission is marked by a high stop bit. If more data is available on the Bridge the next 8 bits can be sent after applying another start bit.

The signals, transferred by the USB to UART Bridge need to be interpreted by the FPGA. For this purpose an UART controller has been developed (Appendix). It is a small state machine consisting of five states: 'idle', 'start', 'receive', 'stop' and 'send'. The state machine remains in the 'idle' state until it recognizes the start bit by observing the input signal to change to 'low'. The state transits to 'start' and waits by using a counter until half of the *bit time* passed. Then the state changes to 'receive'. In this state the 8 bits transmitted by the bridge will be written into a 'std_logic_vector' after waiting for one *bit time* each. The state then transits into 'stop' signalising to the NL9000 Controller that there are data available

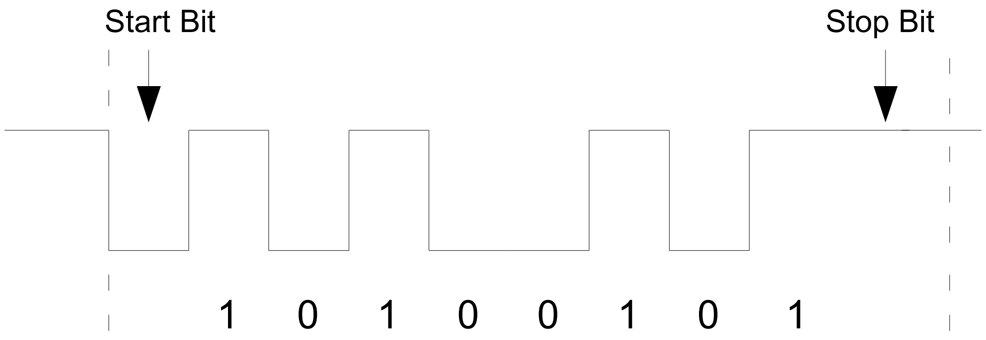


Figure 7.2: Data format of the USB to UART Bridge. The UART signal is a serial communication format starting with a low start bit followed by 8 bits of data. It ends with a high stop bit.

and to the Bridge to wait until the data were transmitted to the Controller. This works unproblematically because the NL9000 controller is faster by several orders of magnitude than the USB to UART Bridge meaning that the NL9000 Controller waits always for new UART data when the state machine is in the state 'stop'. Next, the state is changed into 'send' and the UART controller waits until the NL9000 Controller sends a signal to confirm the receipt of the data. The state will be changed back to 'idle' until it recognises a new start bit. A similar state machine is implemented to send data received from the NL9000 Controller to the Bridge.

Initialisation Design of the NL9000

After the implementation of the UART Controller the communication between PC and FPGA is established. Data can be sent from Virtex 5 to the computer and vice versa. Furthermore, a method to exchange data with the NL9000 is necessary. Before any instruction can be sent to the KBP, it has to be initialised. The initialisation process needs, after setting the reset signals, a stable DCLK in combination with a set of instructions on the input buses of the NL9000. The clock has to be generated in the FPGA and applied to the output pins corresponding to the DCLK bus of the NL9000. In the default setup of the development platform there is one clock routed to the FPGA with a frequency of 200 MHz and one to the NL9000 with 225 MHz as shown in Figure 7.3. Since it would be technically problematic to convert the 200 MHz clock into a 225 MHz clock matching to the core frequency of the NL9000, the RCLK has to be used to generate the DCLK.

The NL9000 delivers the 225 MHz RCLK after the reset signals are applied. It is routed to a clock input pin of the Virtex 5 for read-in. Due to construction constraints it is routed into a Digital Clock Manager (DCM) before the jitter can be removed by using an internal Phase Locked Loop (PLL). The instructions have to be applied to the output pins of the Virtex 5 on the rising and the falling edge of the DCLK. Since the timing is very tight it is necessary to store the bit sequence physically close to the output pins. This can be done by two FIFOs, one storing the bits for the rising and one for the falling edge of the clock. The FIFOs can be read out with the internal 225 MHz clock applying its content to the output pins. To have the data stable when the DCLK toggles the phase of the internal clock has to be shifted before being applied to the pins of the DCLK. Therefore, a DCM close to the output pins can be used. This setup is illustrated in Figure 7.4.

The first test for the development platform needs a NL9000 controller consisting of the elements described in Figure 7.4 and a state machine. This state machine should be connected to the UART controller and will basically wait for a command of the user to start the initialisation process. As described before it will apply the reset signals, route the clocks and send the required instructions. After it received a response from the NL9000, it can send a reply byte-wise to the UART controller, which will forward it to the USB port of the PC. By using this method the basic functionality of the involved components can be validated.

To use full functionality of the NL9000 a more complex NL9000 controller has to be used, which is described in the next passage.

7.2.2 Final Program for the Xilinx Virtex-5

In this section the final version of the program for the FPGA is described. It is supposed to give an overview of the structure of the NL9000 Controller and its implemented features.

Data Paths

The software design for the FPGA has to be created with the aim of running pattern matchs in real time. Since the NL9000 clock is running with a frequency of 225 MHz on default settings and can be increased to up to 300 MHz, the timing of the signals on the pins connected to the NL9000 has to be considered carefully. Therefore, the bit sequences describing instructions on the NL9000 have to be built before and stored close to the I/O banks of the Virtex 5. In this work two FIFOs have been used for the output signals and another two for the input buses. The FIFOs are connected to double data rate flip flops, one FIFO providing the data for the rising edge and the other one for the falling edge of the clock. The FIFOs have to be designed large enough to store data for multiple instructions. Otherwise it would force idle cycles if it was necessary to wait for data coming from parts further away from the I/O banks. In the design developed in course of this diploma thesis the

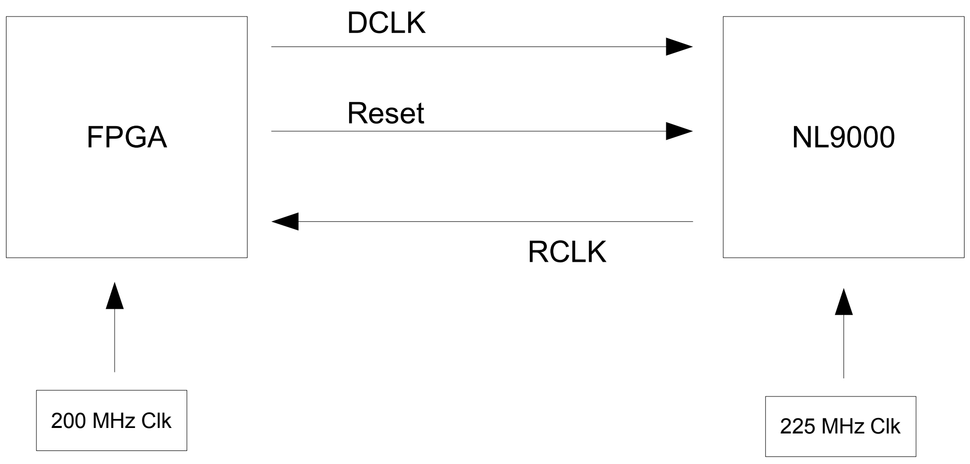


Figure 7.3: Overview of the Clocks necessary to run the application. The 200 MHz clock for the FPGA and the 225 MHz clock for the NL9000 are implemented on the development platform. The RCLK from the NL9000 will be available after the reset signals are applied. To start operations on the NL9000 a stable DCLK has to be delivered matching to the 225 MHz core frequency.

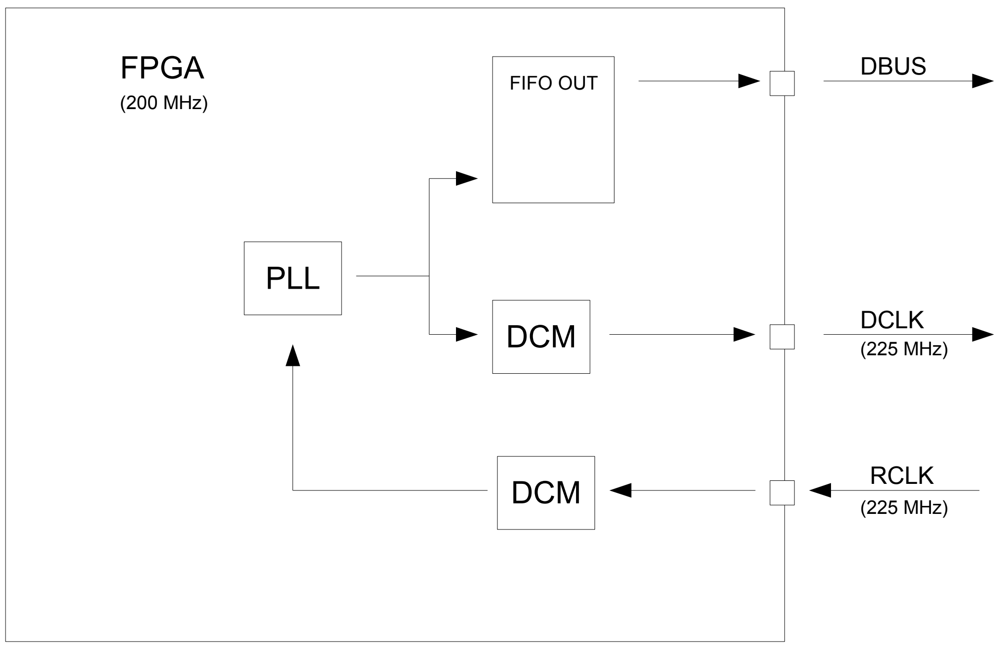


Figure 7.4: Implementation of a stable DCLK by using the RCLK. The clock has to be routed to a DCM on the input pin before the jitter can be removed by the PLL. This internal clock can be used to read out the data stored in a FIFO but the phase has to be shifted in a DCM afterwards to have stable data when the DCLK toggles.

width of the output FIFOs has been chosen to 128 bit and the depth to 2048. The FIFOs of the Xilinx design library allow depths of up to 16384 if bigger FIFOs are needed. Furthermore, the FIFOs provide three flags: full, overflow and empty. The input FIFOs are similar to the output FIFOs but using a width of 64 bit fitting best to the input data.

An overview of the data paths is given in Figure 7.5. For easier utilization a Labview user interface has been designed, which will be described more detailed in Section 7.3. The USB to UART Bridge converts the USB data to an UART signal and sends it to the FPGA. The UART Controller implemented in the FPGA can read this data format and forward it byte-wise to the state machine of the NL9000 controller. This state machine communicates with the input and output FIFOs and starts the test runs. The FIFOs are connected to the I/O banks where the signals will be applied to the buses of the NL9000.

Communication Protocol

The state machine of the NL9000 Controller has a variety of tasks and can be controlled by the user by utilising the protocol introduced in Figure 7.6.

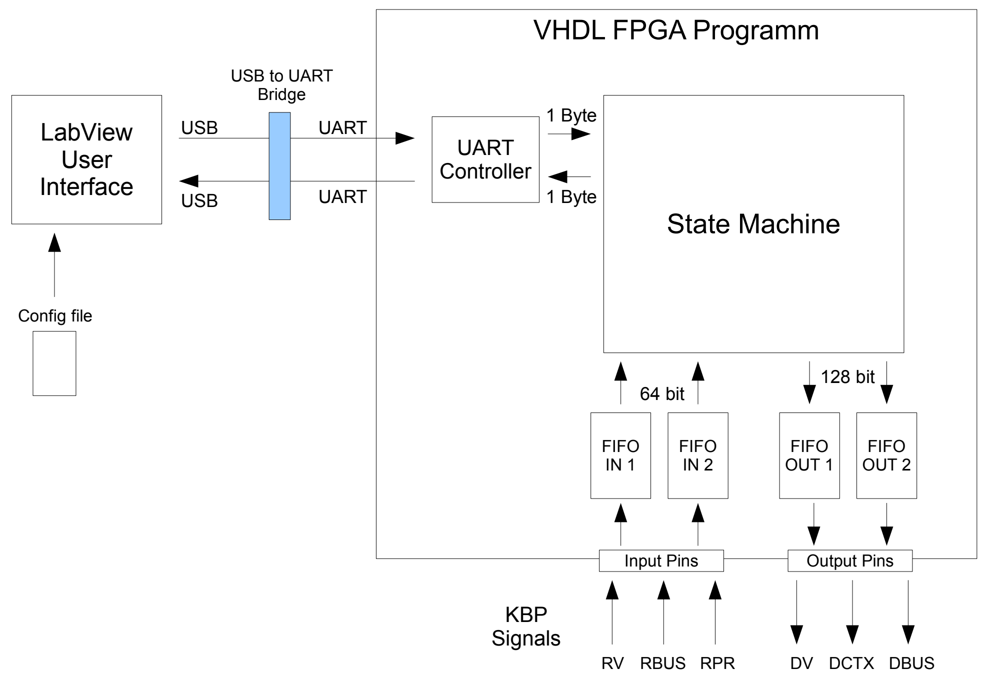


Figure 7.5: Block Diagram showing the communication paths of the NL9000 test application. The data are transferred from the user interface to the development platform using the USB port of the PC. On the platform the USB to UART Bridge converts the USB signal into a UART signal and send it to the FPGA. In the FPGA an UART Controller can interpret these signals and forward them byte-wise to the state machine of the NL9000 Controller. The state machine can communicate with the NL9000 using FIFOs, which are routed to the I/O banks used as connection to the NL9000 buses.

CMD1	CMD2	SIZE	ADDRESS + DATA
Table 7.1	Table 7.2	3 - 45	

Figure 7.6: Protocol used to send commands to the NL9000 Controller. The CMD1, CMD2 and SIZE block of the protocol are 1 Byte each. The size of the address and data block is defined by the content of the size block.

The first byte arriving at the state machine will be interpreted as a command of Table 7.1. If the byte is ’01’ in hexadecimal, the state machine will write a NL9000 instruction in the output FIFOs after it received additional data specifying the instruction. This data have to be sent by a second block of one byte to choose an instruction of the NL9000 (Table 7.2), one byte to describe the length of the last block containing the instruction specific data. If, for example, a register with the address ’000001’ shall be read, the CMD1 will have to be ’01’ for writing a NL9000 instruction and CMD2 ’02’ for *Register Read*. The ’Address+Data’ block is three byte (’03’) long since there is only the address sent resulting in a total data sequence of ’010203000001’.

If the user wants to read the status of the FIFOs, the second command in Table 7.1 can be used. This will return 3 bytes consisting of the status flags of the FIFOs and the error flags of the NL9000. Table 7.3 shows the response format. The three error flags of the NL9000 are low active meaning that they would be ’1’ if no error occurred.

The *Setup Data Recording* command can be used to configure the number of clock cycles the data are recorded by the input FIFOs. In default settings the FIFOs will write 3 clock cycles when the Result Valid (RV) signal is assigned. This will be necessary if the user wants to read records of the database or registers since the RV signal will be set to ’0’ after the first clock cycle although data can be applied for another two clock cycles. If the chip is used for a test run consisting of compare instructions only, it will be sufficient to write just one clock cycle after RV is assigned. This will reduce the memory space used in the input FIFOs without losing data.

DCLK Timing

As described in Figure 7.4 a DCM is necessary to ensure correct timing between data and DCLK on the output pins. This DCM can be configured by using the *Shift DCLK* command. After the NL9000 Controller received the command code it will wait for another byte which defines the shift direction: ’00’ will cause a shift left and ’01’ a shift right, and a third byte with a hexadecimal number. The shift will be applied by multiplying the received number with 1.406 degrees. In this work the default value of the phase shift is 48 (67.488 Degrees).

Organisation of the VHDL program

The files of the VHDL program are summarised in Table 7.4. The first six files are designs from the Xilinx library. They describe the functionality of the FIFOs and the clocking resources. The next six files contain the VHDL code of the program. In ’top.vhd’ the top component is described. It defines the input and output signals of the FPGA and connects the sub-modules of the VHDL program. The last two files are used for the behavioral simulation of the Controller, which will be described in the next section.

Command Code 1	Operation	Description
01	Write NL9000 Instruction	Operation to write a NL9000 instruction into the output FIFOs. The instruction will be specified with the next byte (Table 7.2).
02	Read Status	Returns the FIFO status flags and the three error flags of the NL9000. (Table 7.3)
03	Start NL9000 Test	Starts the read-out of the output FIFOs. The content will be applied to the input buses of the NL9000.
04	Start Initialisation	Executing the Initialisation process of the NL9000.
05	Read input FIFO	Reads the records of the input FIFOs and sending them to the user.
06	Reset FIFOs	Resets the input and output FIFOs.
07	Setup Data Recording	For compare operations only one clock cycle of the NL9000 response must be recorded. Reading the database or registers will return responses of three clock cycles.
08	Shift DCLK	The DCLK will be shifted by $x \cdot 1.406$ degrees. x is defined by the next bytes.
09	Latency Readout	If an instruction is sent to the NL9000, a counter will be started and stopped when the response of the NL9000 is available.

Table 7.1: List of commands for the CMD1 block. The *Write NL9000 Instruction* will enable a second command block (see Table 7.2). Otherwise the state machine will directly execute the Operation and wait for the beginning of the next command code.

Command Code 2	NL9000 Instruction	Address+Data Block
01	Register Write	3 bytes address 10 bytes data
02	Register Read	3 bytes address
03	Database Write XY	3 bytes address 10 bytes data X 10 bytes data Y
04	Database Read X	3 bytes address
05	Database Read Y	3 bytes address
06	Context Buffer Write	2 bytes address 20 bytes data
07	Context Buffer Write and Compare1	2 bytes address 20 bytes data 1 byte compare setup
08	Context Buffer Write and Compare2	2×2 bytes address 2×20 bytes data 1 byte compare setup

Table 7.2: List of commands for the CMD2 block. After sending the Command Code the state machine waits for the 'Address+Data' block. When all bytes are received the state machine writes the instruction in the output FIFO and waits for the next command.

Byte	Bit	Content
1	7	full FIFO_OUT1
	6	empty FIFO_OUT1
	5	overflow FIFO_OUT1
	4	'0'
	3	full FIFO_OUT2
	2	empty FIFO_OUT2
	1	overflow FIFO_OUT2
	0	'0'
2	7	full FIFO_IN1
	6	empty FIFO_IN1
	5	overflow FIFO_IN1
	4	'0'
	3	full FIFO_IN2
	2	empty FIFO_IN2
	1	overflow FIFO_IN2
	0	'0'
3	7	PEO_L
	6	GIO_L
	5	PHSEO_L
	4:0	'00000'

Table 7.3: Response data of the NL9000 Controller to the *Read Status* command

Name	Description
dclk_dcm.xaw	The Digital Clock Manager (DCM) is a design element from the Xilinx library. This DCM is used to modify clocks received on the clock input pins. It has the RCLK of the NL9000 on its input and forwards it as internal clock (nl9k_clk_intern).
dclk_dcm2.xaw	DCM used to shift the clock phase before it is applied to the output pins of the DCLK.
dclk_pll.xaw	The Phase Locked Loop (PLL) is a design element from the Xilinx library. It is used to reduce the jitter of the clock signal coming from the NL9000.
dcm_150MHZ.xaw	DCM to create a 150 MHz clock for internal procedures from a 200 MHz clock coming from a Oscillator on the platform.
fifo_in.xise	The FIFO is a design element from the Xilinx library. It is used to store responses coming from the NL9000 and is configurable in width and depth.
fifo_out.xise	This FIFO is used to store instructions for the NL9000. It is also configurable in width and depth and will be read out to the output pins when the test run is started.
func_lib.vhd	Library with a function to calculate the parity bits for the input buses of the NL9000.
state_machine.vhd	Internal State Machine controlling the NL9000.
testboard.vhd	VHDL-File managing the data paths and connects the State Machine to the FIFOs and I/O - Blocks.
uart_controller.vhd	Controller converting UART signals into 'std_logic_vector' of 1 Byte size and and vice versa.
top_ucf.ucf	User Constraint File (UCF) connecting signals of the VHDL code to the correct pins of the FPGA.
top.vhd	Top Entity of the VHDL program connecting the modules of the VHDL code.
top_tb.vhd	Testbench file to simulate the behaviour of the program on the Xilinx Virtex 5.
sim_func.vhd	Library of the simulation to simplify the data input.

Table 7.4: List of Files for the VHDL program of the Xilinx Virtex 5. In the upper section components from the Xilinx library are used. These components are already implemented clock devices in the FPGA or FIFOs, which can be adapted to specific usage. The second section includes VHDL files with different functions describing the Controller implemented on the FPGA. The last section shows files to simulate the behaviour of the VHDL program.

7.3 ISim Simulation

During the development process of a program designed to run on a FPGA it will be necessary to simulate the behaviour of the program if problems occur. Usually the FPGA will give no feedback about why it is not operating in the intended way if there is an error in the VHDL code. Therefore, it is useful to simulate the behaviour of the program to be able to look into internal signals and memories, which are not accessible in modern chip designs. A simulation tool, ISim, is integrated in Xilinx ISE. This tool has been used for the simulations of the FPGA program.

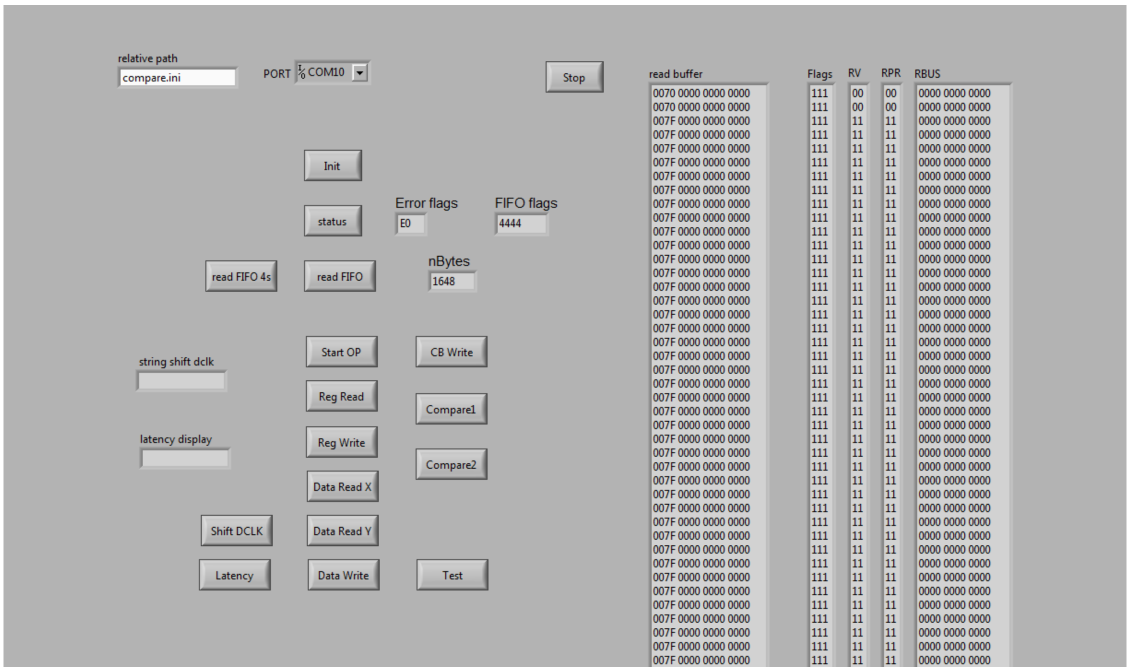
To create the simulation environment as simple as possible the library 'sim_func' was developed. The purpose of this library is to apply a function generating the input UART signal for the controller. The arguments of the function are the number of bytes to send, the data in hexadecimal, the UART bit rate and the UART signal where the data shall be applied. By using this function in the testbench the user can send test samples to the simulated FPGA program and observe the internal components and responses on the output graphically in the ISim Waveform. The simulation can also be used to simulate the behaviour of the controller when the NL9000 applies data on the result buses.

7.4 Functions of the Labview Program

As discussed before, a test run on the NL9000 needs a lot of instructions to configure the chip and write the database. In contrast to the simulation described in chapter 5, the situation gets even more complicated since the instructions cannot be applied directly to NL9000 and involves the the communication with the Xilinx Virtex 5, which has another instruction set and the USB connection. Applying the data to the USB port manually is not a convenient solution for bigger tests.

Therefore, a graphical user interface has been designed in LabView, shown in Figure 7.7. In the top of the picture is a drop down box to define the port of the PC to send the data from. Next to it is a stop button to release the port after finishing the usage. The other buttons are used to start different operations on the NL9000 Controller. The gray fields can display the response coming from the NL9000 Controller. At the top left, there is a field to define the configuration file used. A configuration file is structured in different sections with keys for each entry. A picture of a configuration file is shown in Figure 7.8. An entry of this configuration file can be read out by defining for example section 'CB Write' and key 'CBAddress0'. This will return '0002'.

The user interface uses this principle by reading out the 'NumberEntries' position at the section of the operation the program is working on. With this information the program knows how much entries are in the section and can read them out systematically. The data will be then merged with predefined instruction formats for the Controller and sent to the USB port. Another challenge is to handle the different data formats. The configuration file is written in ASCII code, while the

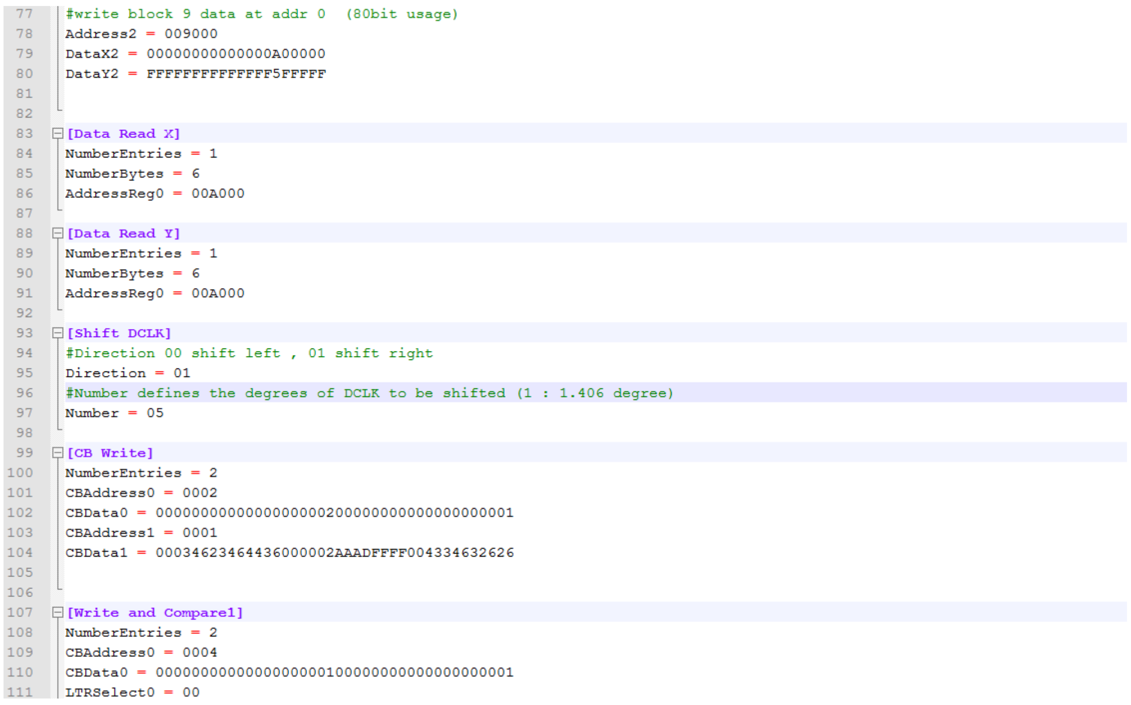


relative path: compare.ini
PORT: COM10
Error flags: E0
FIFO flags: 4444
nBytes: 1648
string shift dclk:
latency display:

```
read buffer          Flags  RV  RPR  RBUS
0070 0000 0000 0000  111    00  00   0000 0000 0000
0070 0000 0000 0000  111    00  00   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
007F 0000 0000 0000  111    11  11   0000 0000 0000
```

Figure 7.7: User interface designed to simplify the communication with the NL9000 Controller. At the top left is a field to define the path of the Config File. Next to it is a drop down box to specify the port used to send the USB data. The buttons can be used to send instructions to the NL9000 Controller. The gray fields are displaying results coming from the development platform after the initialisation process was performed.

USB to UART Bridge expects hexadecimal data. Labview has no function to convert these data formats into each other directly. Due to that some extra loops have to be added to convert the data into the correct format. An extraction of the LabView code can be found in the appendix.



```
#write block 9 data at addr 0  (80bit usage)
Address2 = 009000
DataX2 = 00000000000000A00000
DataY2 = FFFFFFFFFFFFFF5FFFFF


[Data Read X]
NumberEntries = 1
NumberBytes = 6
AddressReg0 = 00A000

[Data Read Y]
NumberEntries = 1
NumberBytes = 6
AddressReg0 = 00A000

[Shift DCLK]
#Direction 00 shift left , 01 shift right
Direction = 01
#Number defines the degrees of DCLK to be shifted (1 : 1.406 degree)
Number = 05

[CB Write]
NumberEntries = 2
CBAddress0 = 0002
CBData0 = 0000000000000000000020000000000000000001
CBAddress1 = 0001
CBData1 = 00034623464436000002AAADFFFF004334632626


[Write and Compare1]
NumberEntries = 2
CBAddress0 = 0004
CBData0 = 0000000000000000000010000000000000000001
LTRSelect0 = 00
```

Figure 7.8: Extraction of a sample config file for the LabView user interface. The file is structured in sections with keys for each entry. In these files, entries can be read out by choosing a section and key.

8 Test Runs and Results

In this chapter the test runs performed on the Development Platform HTG-100GIG are discussed. The NL9000 was configured with different setups, which could be useful to implement a fast track trigger. In this context the performance was measured and the functionality examined. The tests for the hardware are orientated on the tests already presented in chapter 5 to have the ability to compare the results and judge about the quality of the simulation model.

8.1 Test Configuration Files

For running tests on the development platform the defining information has to be written into configuration files. For each NL9000 instruction there is one section in the file. Depending on the instruction, address and data have to be written into the file. For example, read instructions need only an address while *Database Write* instructions need address, data X and data Y values.

The structure of the tests is analog to the tests of chapter 5, which are described in Table 5.2. The *database configuration* test examines the default setup of the chip and tests the basic operation of the database and registers of the chip. For the *80 bit compare* and *640 bit compare* the database and Key Processing Unit are configured and multiple compare operations are sent to measure the latency and result rate. The *parallel compare* test configures the width of the database to 80 bits and performs four searches in parallel.

In addition to the tests of chapter 5, a test is performed on the hardware confirming its reliability. For this test 1000 *Database Write* instructions have been generated by a random number generator of Labview and have been written into a configuration file. These instructions are applied to the NL9000 repeatedly to confirm stable and reliable functionality in long-term applications.

8.2 Measurement of the Power Consumption

For the measurement of the power consumption of the NL9000 two modifications were necessary. The first modification had to be made to measure the current and voltage the KBP is consuming. For this reason an external power supply had to be integrated into the circuits of the development platform. The second modification had to be made on the software design of the NL9000 Controller because it has no feature to send compare instructions repeatedly to the NL9000. This feature was added by writing a compare instruction into the last entry of the output FIFOs instead of an idle instruction by default. This way the Controller will always send a compare instruction to the NL9000 if no other instruction is sent by the user. This Controller design is used to measure the power consumption of the NL9000 if used permanently, while the normal Controller can be used to measure the power consumption during idle times.

8.3 Latency measurement

To measure the result latency of the NL9000 while operating on 225 MHz and 300 MHz, two approaches have been used. One approach is the direct measurement of the time difference between the start and the response of the compare operation by using an oscilloscope. Since it is not possible to get access to the input and output pins of the NL9000 directly, pins of the FMC Connector (section 6.2.1) can be used. In the current design the pins of the connector are unused and easily accessible. The start of each compare instruction is marked by a high DV signal and the response is marked by a high RV signal. This makes these two signals ideal candidates for the latency measurement. To forward them to the pins of the FMC Controller, extra features have been added to the NL9000 Controller. The output FIFO is the closest point to the output pins where the DV signal can be duplicated. To have similar routings between the FIFO the FMC pin on one hand and the FIFO and the NL9000 pin on the other hand, an *output double data rate flip flop* has been added in front of the FMC pin. The RV signal is forwarded directly to the corresponding FMC pin to have the minimum time delay.

Despite the efforts made to have the routes almost identically, there are time delays between the different paths, which can be estimated by the software tools of Xilinx. For the NL9000 design operating on 225 MHz the RV signal is delayed by maximal 14.0 ns, for the 300 MHz design the delay is maximal 13.4 ns. The DV signal is routed into a flip flop leading to an uncertainty of one clock cycle.

The second approach is to start a counter when the instruction is sent and to stop it after the response is available on the result bus. This function is implemented in the NL9000 Controller and the result can be displayed on the user interface. The routing of the stop signal in this method causes delays which have to be corrected by subtracting 2 clock cycles in the 225 MHz design. In the 300 MHz design the NL9000 Controller has to be modified to keep the timing constraints on the output pins. The most critical path is the DV signal because it is used on two output pins and the counter. Therefore, an extra flip flop has been added between counter and output FIFO. This delays the start signal by one clock cycle. To correct for it, it is necessary to subtract this time span from the measured latency.

8.4 Results

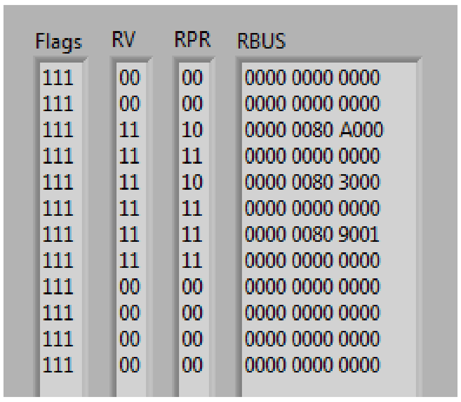
The first operation sent to the NL9000 is the initialisation process. This process operates as described in the manual and in agreement with the simulation. The database and the Block Mask Register have no initial value and will return random bit sequences as long as nothing is written into it. The control registers of the NL9000 can be accessed and configured as done in the simulation. The *reliability* test provides a response of the tested hardware similar to Figure 7.7. Applying several thousand *Database Write* instructions to the NL9000 does not violate any timing constraints of the communication between FPGA and NL9000 nor causing errors of the NL9000 itself.

The power consumption of the NL9000 was measured with regard to three different conditions. While the FPGA was programmed the power consumption had the lowest value with 0.64 W due to missing signals on its input pins. Using the chip mainly with idle cycles, the power consumption was measured to be 2.267 W. When the Controller design was used to send a compare instruction to the NL9000 repeatedly, the power consumption reached the highest value of 2.360 W. The measurements were performed on the chip at ambient temperature. It is to be expected that the power consumption is dependent on temperature. The Low Power Mode of the NL9000 had no measurable effect on the results.

Figure 8.1 displays the response of the development platform after performing the *80 bit test* in hexadecimal values. Providing the instructions to the NL9000 with 300 MHz did not cause an error flag and returns the correct search results in the interval of one clock cycle. The first result at the NL9000 is a match in block 'A' address '000'. The '8' is the system match flag signaling that the search key was found in at least one entry in the database. The next line is filled with '0' since there were no results available on the falling edge of the result clock. The following line represents another match in the next compare instruction. In this case it is a match in block '3' address '000'. After the next clock cycle has passed, the third result is available on the RBUS, signaling a match in block '9' address '001'. Thus, the *80 bit compare* test of the hardware components is consistent with the simulation results of chapter 5 including functionality and result rate.

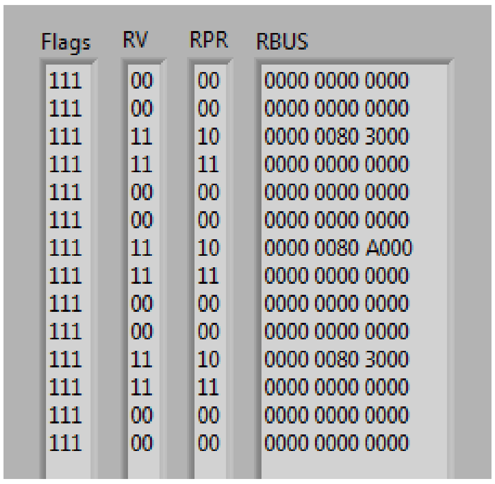
Figure 8.2 is showing the response of the tested hardware of the *640 bit compare* test. Similar to the test before, the NL9000 returns the expected results of the compare instructions. In the *640 bit compare* test the *Context Buffer Write and Compare2* instructions have to be sent in intervals of four clock cycles, leading to results received also in intervals of four clock cycles. In Figure 8.2 only five half-clock cycles can be seen between two results since the NL9000 Controller does not record data of several clock intervals without valid data being on the RBUS. In this case, a direct measurement of the result rate is not necessary, because the NL9000 is constructed to send results in fixed time intervals depending on the mode of operation (see section 4.3.3).

Figure 8.3 illustrates the response of the platform to the *parallel compare* test. When the RV bits are set to '11' the results of the four parallel searches are available



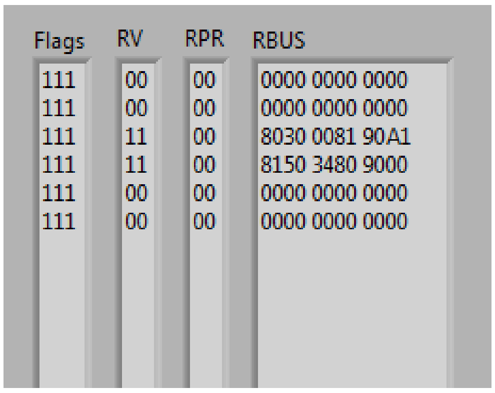
Flags	RV	RPR	RBUS
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000
111	11	10	0000 0080 A000
111	11	11	0000 0000 0000
111	11	10	0000 0080 3000
111	11	11	0000 0000 0000
111	11	11	0000 0080 9001
111	11	11	0000 0000 0000
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000

Figure 8.1: Snapshot of the user interface showing the response of the development platform after applying the *80 bit compare* test. The error flags are constant '111' meaning that no errors occurred. Valid results are allocated when the RV bits are '11'. The RBUS shows three successful matches at the third, fifth and seventh half-clock cycle.



Flags	RV	RPR	RBUS
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000
111	11	10	0000 0080 3000
111	11	11	0000 0000 0000
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000
111	11	10	0000 0080 A000
111	11	11	0000 0000 0000
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000
111	11	10	0000 0080 3000
111	11	11	0000 0000 0000
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000

Figure 8.2: Snapshot of the user interface showing the response of the development platform after applying the *640 bit compare* test. The error flags are constant '111' meaning that no errors occurred. Valid results are allocated when the RV bits are '11'. On the RBUS three compare results are shown. Block '3' address '000' is matching to the first and third compare instruction in this case.



Flags	RV	RPR	RBUS
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000
111	11	00	8030 0081 90A1
111	11	00	8150 3480 9000
111	00	00	0000 0000 0000
111	00	00	0000 0000 0000

Figure 8.3: Snapshot of the user interface showing the response of the development platform after applying the *parallel compare* test. The error flags are constant '111' meaning that no errors occurred. Valid results are available on the RBUS when the RV bits are '11'. On the RBUS one compare result with four parallel results is illustrated. The matches are in block '3' address '000', block '19' address '0A1', block '15' address '034' and block '9' address '000'.

on the RBUS. Block '3' address '000' and block '19' address '0A1' are applied to the RBUS on the rising edge of the RCLK and block '15' address '034' and block '9' address '000' on the falling edge of the RBUS. As discussed in chapter 5, this would permit four searches in parallel with a frequency of up to 600 MHz if the maximum input bus of 80 bit is sufficient for a track trigger application. In this example the search key width of 320 bit was used leading to a result rate of 150 MHz with four 80 bit compares in parallel.

The measurement of the result latency was performed for the 225 MHz and the 300 MHz design of the NL9000 Controller by using two different methods. The results of the two implementations of the measurement are shown for two examples in Figure 8.4 and Figure 8.5.

Figure 8.4 illustrates the method using the oscilloscope on the 225 MHz design. Figure 8.5 shows the the response of the method using a counter in the Xilinx Virtex 5 for the 300 MHz design. An overview of all results are summarised in Table 8.1.

For the 225 MHz design, the time interval between start and stop signal is 139.4 ns in the oscilloscope method. 14.0 ns have to be subtracted due to delays within the Xilinx Virtex 5 resulting in a measured latency of 125.4 ns $\pm$ 4.4 ns. The counter method provides a value of 31 clock cycles between start and stop signal. 2 clock cycles have to be subtracted due to delays within the FPGA leading to a result latency of 128.89 ns $\pm$ 8.89 ns.

The oscilloscope method measures a time interval of 109.0 ns in the 300 MHz design. After correcting for the delays within the FPGA, the measurement results in 95.6 ns $\pm$ 3.3 ns. The counter method returns the value '1E' in hexadecimal as pictured in Figure 8.5. This translates into 29 $\pm$ 2 clock cycles after the necessary correction corresponding to 96.67 ns ± 6.67 ns. This can be compared to the simulation result of chapter 5, where the Verilog model predicts a latency of 96.595 ns fitting well to the results of the hardware tests.

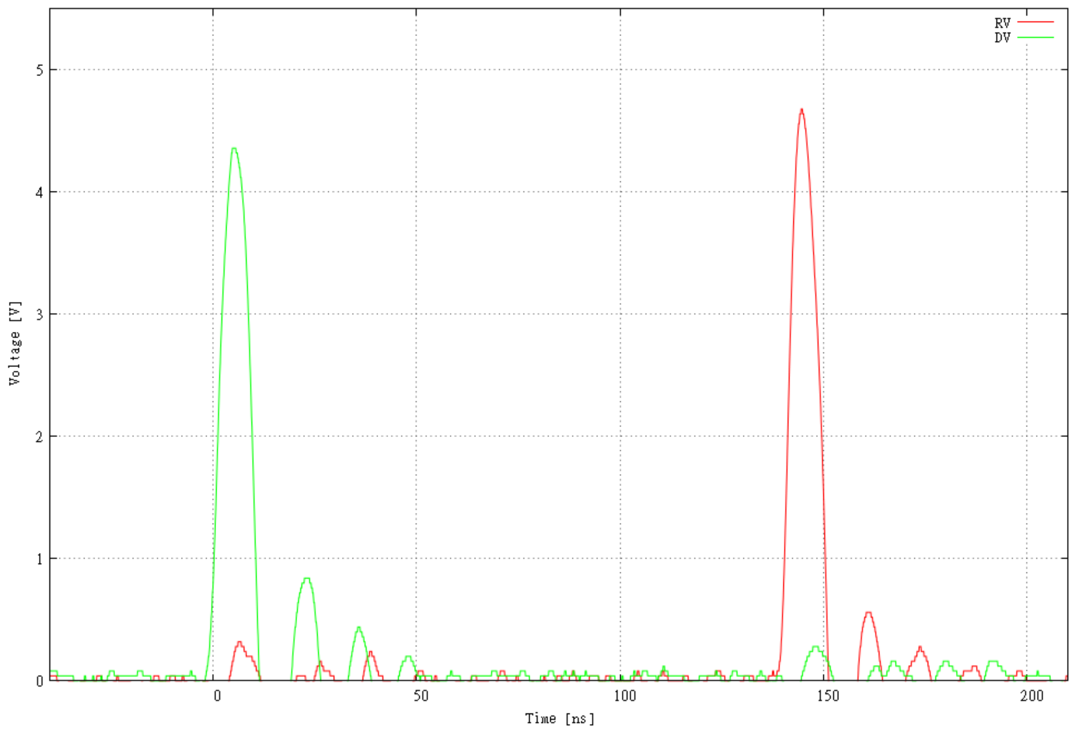
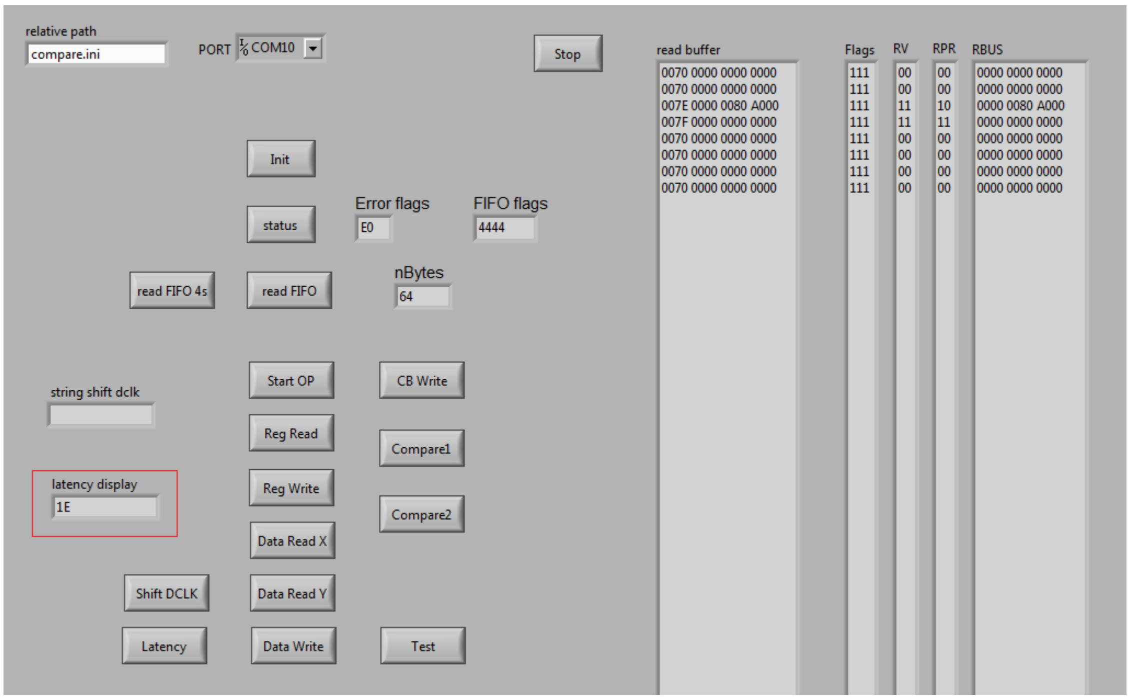


Figure 8.4: Measurement of the result latency using the oscilloscope. The green line represents the DV signal, the red line is the RV signal. Both signals show reflections of the cable connection.



relative path: compare.ini

PORT: COM10

Error flags: E0

FIFO flags: 4444

nBytes: 64

string shift dclk:

latency display: 1E

```
read buffer               Flags  RV  RPR  RBUS
0070 0000 0000 0000       111    00  00   0000 0000 0000
0070 0000 0000 0000       111    00  00   0000 0000 0000
007E 0000 0080 A000       111    11  10   0000 0080 A000
007F 0000 0000 0000       111    11  11   0000 0000 0000
0070 0000 0000 0000       111    00  00   0000 0000 0000
0070 0000 0000 0000       111    00  00   0000 0000 0000
0070 0000 0000 0000       111    00  00   0000 0000 0000
0070 0000 0000 0000       111    00  00   0000 0000 0000
```

Figure 8.5: Measurement of the result latency using a counter in the FPGA. The counter returns the value of '1E' in hexadecimal displayed in the latency display in the bottom left. In the top right the response to the single compare instruction can be seen, which was used to perform the measurement.

Clock Frequency	Oscilloscope	Counter
225 MHz	125.4 ns ± 4.4 ns	128.89 ns ± 8.89 ns
300 MHz	95.6 ns ± 3.3 ns	96.67 ns ± 6.67 ns

Table 8.1: Results of the latency measurement with clock frequencies of 225 MHz and 300 MHz.

9 Conclusion

In the year 2020 a luminosity upgrade is planned for the LHC increasing its design luminosity by an order of magnitude. The inner detector and the trigger system have to be reconsidered because of the higher data rates. The inner detector will be replaced by a new one with higher granularity and with more read-out channels. To keep the L1 trigger rate below 75 kHz after the upgrade, a L1 track trigger is considered. Because of the stringent timing constraints at the first level trigger of 2.1 µs a track trigger has to be implemented using fast lookup techniques. In this context, the functionality of Knowledge-based Processors (KBPs) have been investigated in this work. A NL9000 chip from Netlogic is considered and studied for a possible L1 track trigger application.

To test the functionality and performance of the NL9000 with respect to its usage in a Fast Track Trigger, a Controller for the Development Platform HTG-100GIG was designed in VHDL. This Controller is implemented in the FPGA on the platform and is able to communicate with the PC by using the USB to UART Bridge. The complete instruction set of the NL9000 is supported by the Controller and can be used to operate tests on the chip. To run the tests under most realistic conditions without being slowed down by the data transfer between PC and FPGA, the instructions can be stored in the FIFOs of the Controller. The tests are applied to the NL9000 running at 300 MHz clock frequency. For a Fast Track Trigger application the latency of the NL9000 is an important parameter and was therefore measured. It was found to be of the order of 100 ns. Furthermore, a Labview user interface was written. To define the test procedures configuration files are used. This way the user can describe the instructions to be applied to the NL9000 simply by writing the test data into the configuration file. The user interface can interpret these files and provide the data in the required format to the Controller.

Tests of the investigated chip were performed in a software simulation using a model from Netlogic and in the hardware design. Features as parallel compare operations and the configuration of the database in different widths have been examined and found to be consistent in both approaches. The performance of the NL9000 on the development platform fits also to the results of the simulation qualifying the chip for a possible usage in a Fast Track Trigger. Due to quickly evolving semiconductor processes, the industry will probably be able to provide faster and bigger devices in the next years. This will allow integration of a high number of hit patterns for track trigger application and reduce the number of needed chips.

Acknowledgments

At the end of my diploma thesis it is time to thank all the people who have been a great help in the last year. I want to say 'thank you' to:

first of all, André Schöning who made this work possible and supported me during my time in his working-group at the 'Physikalisches Institut' in Heidelberg.

Esteban Rubio for being always there for discussion and supervision while working with either hardware or software of the used electronics.

Sebatian Schmitt, Gregor Kasieczka and Rohin Narayan for answering all my physical questions and proof-reading large parts of this thesis.

Arno John, Patricia Sauer and Moritz Kiehn for being kind and entertaining companions during our time together.

last but not least, my parents, my family and friends and especially Anne Ludin for support and encourage during all the time of my studies.

Part I

Appendix

A Lists

A.1 List of Figures

A.2 List of Tables

B Bibliography

[1] Netlogic Microsystems, July 2011, www.netlogicmicro.com.

[2] L. Evans and P. Bryant, *LHC Machine*, Journal of Instrumentation, 3:S08001, 2008.

[3] CERN, 2011/05/25, http://public.web.cern.ch/.

[4] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, Journal of Instrumentation, 3:S08003, 2008.

[5] ATLAS Collaboration, 2011/05/25, http://atlas.ch/.

[6] M.L. Mangano, *The super-LHC*, arXiv:0910.0030, 2009.

[7] F. Zimmermann, *HL-LHC: parameter space, constraints and possible options*, Proceedings of Chamonix workshop, CERN, 2011.

[8] A. Abdesselam, June 2010, http://atlas.web.cern.ch/Atlas/GROUPS/UPGRADES/.

[9] R.L. Bates, *Upgrading the ATLAS barrel tracker for the super-LHC*, Nuclear Instruments and Methods in Physics Research A, 607, 24 - 26, 2009.

[10] ATLAS Upgrade Group, 2011/05/25, https://twiki.cern.ch/twiki/bin/viewauth/Atlas/UpgradeDetectorGeometryVersions.

[11] ATLAS Collaboration, *ATLAS detector and physics performance technical design report*, ATLAS TDR 14, CERN/LHCC 99-14, 1999.

[12] S. Schmitt and A. Schöning, *A Fast Track Trigger for ATLAS at Super-LHC*, Journal of Instrumentation, 5:C07013, 2010.

[13] A. John , *Data Reduction at Module Level in a Level-1 Track Trigger for the ATLAS High-Luminosity Upgrade*, Diploma thesis, Universität Heidelberg, 2011.

[14] S. Pirner , *Intelligent Filtering Algorithms for an upgraded Inner Tracker at ATLAS*, Bachelor thesis, Universität Heidelberg, 2010.

[15] Netlogic Microsystems, *NL9000 RA Knowledge-based Processors 2M, 1M, 512k, and 128k Records*, Datasheet, Revision 2.6, May 2010.

[16] D. Glodeck , *Simulation eines L1-Spur-Triggers für ATLAS*, Bachelor thesis, Universität Heidelberg, 2010.

[17] K. Pagiamtzis and A. Sheikholeslami, *Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey*, IEEE Journal of Solid-State Circuits, 41:712 - 727, 2006.

[18] ModelSim SE, Version 6.6b, May 2010, www.modelsim.com.

[19] HiTech Global, *HTG-V5-100GIG User Manual*, December 2009.

[20] D. Chen, J. Cong and P. Pan, *FPGA Design Automation: A Survey*, Foundations and Trends in Electronic Design Automation, Vol. 1, No 3, 195 - 330, 2006.

[21] V. Betz, J. Rose and A. Marquardt, *Architecture and CAD for Deep - Submicron FPGAs*, Kluwer Academic Publishers, 1999.

[22] Xilinx, *Virtex-5 FPGA User Guide*, May 2010.

[23] Xilinx ISE Design Suite, Version 12.1, April 2010, www.xilinx.com.

[24] Labview, Version 9.0, 2009, www.labview.com.

C Programs

C.1 VHDL program codes for the FPGA

UART Controller

This Part of the UART Controller is described in 7.2.1. It is a small state machine converting the UART signals in 1 byte 'STD_LOGIC_VECTOR'.

```
architecture Behavioral of uart_controller is
entity uart_controller is
generic(
    baud : natural := 115200;
    sys_freq : natural := 150  -- MHz
);
port(
   CTS : out STD_LOGIC := '0';
   TXD : in STD_LOGIC;
   RXD : out STD_LOGIC := '1';
   RTS : in STD_LOGIC;

   sysclk : in STD_LOGIC;

   d_valid_out : out STD_LOGIC := '0';
   to_nl9k_data : out STD_LOGIC_VECTOR (7 downto 0) := (others=>'0');
   uart_wait : out STD_LOGIC := '0';
   nl9k_rd : in STD_LOGIC;
   d_valid_in : in STD_LOGIC;
   from_nl9k_data : in STD_LOGIC_VECTOR (7 downto 0)
);
end uart_controller;

signal uart_counter, uart_counter2 : natural := 0;
signal bit_counter, bit_counter2 : natural := 0;
signal data_in : STD_LOGIC_VECTOR ( 7 downto 0);
signal data_out : STD_LOGIC_VECTOR (7 downto 0);

type state_type is ( idle, start, receive, stop, send);
signal currentState  : state_type := idle;
constant bit_time : natural := (sys_freq *1000000 / baud -1);
```

```
begin

   receive_data : process (sysclk)
   begin
     if rising_edge(sysclk) then
       case currentState is

       when idle =>
       if TXD = '0' then
         currentState <= start;
       end if;

       when start =>
         if uart_counter = (bit_time /2) then
           currentState <= receive;
           uart_counter <= 0;
         else
           uart_counter <= uart_counter +1;
         end if;

       when receive =>
         if uart_counter = bit_time and bit_counter < 8 then
           data_in (bit_counter) <= TXD;
           bit_counter <= bit_counter +1;
           uart_counter <= 0;
         elsif bit_counter > 7 then
           currentState <= stop;
           uart_counter <= 0;
           to_nl9k_data <= data_in;
         else
           uart_counter <= uart_counter +1;
         end if;

       when stop =>

         if uart_counter = bit_time then
            currentState <= send;
            d_valid_out <= '1';
            CTS <= '1';
         else
            uart_counter <= uart_counter +1;
         end if;
```

```
      when send =>
         if nl9k_rd = '1' then
            d_valid_out <= '0';
            CTS <= '0';
            currentState <= idle;
            uart_counter <= 0;
            bit_counter <= 0;
         end if;
      end case;
   end if;
end process;
```

Section of the NL9000 Controller writing an instruction into FIFOs

This program code demonstrates the implementation of a NL9000 Instruction in the state machine of the NL9000 Controller. When the state machine received the commands to get into the 'cb_write_compare2' state, it waits for the necessary data. The data will be submitted by the UART Controller byte-wise as long as the 'rd_usb_in' signal is set to '1'. The received data are saved temporarily in the FPGA in different 'STD_LOGIC_VECTOR' to process them in a *CB Write and Compare2* Instruction and write them into the two output FIFOs.

```
when cb_write_compare2 =>

    rd_usb_in <= '1';

    if d_valid_in = '1' then
        if programm_counter3 < 2 then
            received_cb_address1 ( ((2 - programm_counter3) *8 -1)
             downto ((1 - programm_counter3) *8 )) <= usb_data_in;
            programm_counter3 <= programm_counter3 +1;

        elsif programm_counter3 < 4 then
            received_cb_address2 ( ((4 - programm_counter3) *8 -1)
             downto ((3 - programm_counter3) *8 )) <= usb_data_in;
            programm_counter3 <= programm_counter3 +1;

        end if;

        if programm_counter3 < 24 and programm_counter3 > 3 then
            received_cb_data2 ( ((24 - programm_counter3)*8 -1)
             downto ((23 - programm_counter3)*8 )) <= usb_data_in;
            programm_counter3 <= programm_counter3 +1;
        elsif programm_counter3 < 44 and programm_counter3 > 23 then
            received_cb_data3 ( ((44 - programm_counter3)*8 -1)
             downto ((43 - programm_counter3)*8 )) <= usb_data_in;
            programm_counter3 <= programm_counter3 +1;

        end if;

        if programm_counter3 = 44 then
                received_LTRSelect  <= usb_data_in;
                programm_counter3 <= programm_counter3 +1;

        end if;
    end if;

    if    programm_counter3 = 45 then
        rd_usb_in <= '0';

        data_fifo_out1 <= "0" & x"00000000" & "111" & "01000" &
            ("0" & received_cb_address1 (8 downto 5))
            & received_cb_data2(159 downto 80)
            & parity ( "01000", "0" & received_cb_address1 (8 downto 5)
            , received_cb_data2 (159 downto 80), '1', '1', '1');
        wr_fifo_out1 <= '1';

        data_fifo_out2 <= "0" & x"00000000" & "111" &
            received_LTRSelect(4 downto 0)
            & received_cb_address1 (4 downto 0)
            & received_cb_data2(79 downto 0)
            & parity ( received_LTRSelect(4 downto 0),
            received_cb_address1(4 downto 0),
            received_cb_data2 (79 downto 0), '1', '1', '1');
        wr_fifo_out2 <= '1';
        programm_counter3 <= programm_counter3 +1;

    end if;

    if    programm_counter3 = 46 then
        rd_usb_in <= '0';

        data_fifo_out1 <= "0" & x"00000000" & "000" & "00000" &
            ("0" & received_cb_address2 (8 downto 5))
            & received_cb_data3(159 downto 80)
            & parity ( "00000", "0" & received_cb_address2 (8 downto 5)
            , received_cb_data3 (159 downto 80), '0', '0', '0');
            wr_fifo_out1 <= '1';

        data_fifo_out2 <= "0" & x"00000000" & "000" & "00000"
            & received_cb_address2 (4 downto 0)
            & received_cb_data3(79 downto 0)
            & parity ( "00000", received_cb_address2(4 downto 0)
            , received_cb_data3 (79 downto 0), '0', '0', '0');
            wr_fifo_out2 <= '1';
            programm_counter3 <= programm_counter3 +1;

    end if;

    if programm_counter3 > 46 then
        rd_usb_in <= '0';
        wr_fifo_out1 <= '0';
        wr_fifo_out2 <= '0';
        programm_counter3 <= 0;
        currentState <= idle;
    end if;
```

C.2 VHDL program codes for the ModelSim simulation

Program to calculate the parity bits of the NL9000

This program is part of the Instruction Library of the ModelSim simulation. It was similarly used in NL9000 Controller.

The parity bits are calculated by using two boolean values and toggle them for each '1' found on the even/uneven bits in the input buses of the NL9000.

```
package body functions_pack is

  function parity (   constant DIBUS   : in STD_LOGIC_VECTOR (4 downto 0);
                      constant DCTX    : in STD_LOGIC_VECTOR (4 downto 0);
                      constant DBUS    : in STD_LOGIC_VECTOR (79 downto 0);
                      constant DV_0 ,DV_1 ,DV_2 : in STD_LOGIC
                  )

                      return STD_LOGIC_VECTOR is

    variable parity0    : boolean;
    variable parity1    : boolean;
```

```
begin

  parity0 := false;
  parity1 := false;

  for parity_counter0 in 0 to 2 loop
    if DIBUS(parity_counter0 * 2) = '1' then
      parity0 := not parity0;
    end if;
  end loop;

  for parity_counter1 in 0 to 1 loop
    if DIBUS(parity_counter1 *2 + 1) = '1' then
      parity1 := not parity1;
    end if;
  end loop;

  for parity_counter2 in 0 to 2 loop
    if DCTX(parity_counter2 *2) = '1' then
      parity0 := not parity0;
    end if;
  end loop;

  for parity_counter3 in 0 to 1 loop
    if DCTX(parity_counter3 *2 +1) = '1' then
      parity1 := not parity1;
    end if;
  end loop;

  for parity_counter4 in 0 to 39 loop
    if DBUS(parity_counter4 *2) = '1' then
      parity0 := not parity0;
    end if;
  end loop;

  for parity_counter5 in 0 to 39 loop
    if DBUS(parity_counter5 *2 +1) = '1' then
      parity1 := not parity1;
    end if;
  end loop;

  if DV_0 = '1' then
    parity0 := not parity0;
    end if;

    if DV_1 = '1' then
      parity1 := not parity1;
    end if;

    if DV_2 = '1' then
      parity0 := not parity0;
    end if;

    if parity1 and parity0 then
      return "11";
    elsif parity1 and not parity0 then
      return "10";
    elsif (not parity1) and parity0 then
      return "01";
    else
      return "00";
    end if;
```

C.3 LabView program codes

Configuration File of the 80 bit compare test

In the following section the content of the configuration file of a 80 bit compare test is shown.

```
[Reg Read]
NumberEntries = 13
#NumberBytes including command bytes for State Machine
NumberBytes = 6
#Device Identification Reg
AddressReg0 = 000000
#Device Config Reg
AddressReg1 = 000001
#LTR Block Select Reg
AddressReg2 = 000200
#block x config reg
AddressReg3 = 000400
AddressReg4 = 00040A
AddressReg5 = 000403
AddressReg6 = 000409
#block x BMR reg
AddressReg7 = 0008A0
```

```
AddressReg8 = 000830
AddressReg9 = 000890
#parallel search reg
AddressReg10 = 000202
#range matching reg
AddressReg11 = 080400
#key constr reg
AddressReg12 = 000206


[Reg Write]
NumberEntries = "11"
#NumberBytes including command bytes for State Machine
#only even number of bytes allowed for this LabView programm
NumberBytes = 16
#write dev config reg (select single device)
AddressReg0 = 000001
DataReg0 = 00000000000000000092
#write block select reg activate block 3+9+A
AddressReg1 = 000200
DataReg1 = 00000000000000000608
#write block A config reg  (80bit usage and activate) !!!
AddressReg2 = 00040A
DataReg2 = 00000000000000000001
#write block 3 config reg  (80bit usage and activate) !!!
AddressReg3 = 000403
DataReg3 = 00000000000000000001
#write block 9 config reg  (80bit usage and activate) !!!
AddressReg4 = 000409
DataReg4 = 00000000000000000001
#write block A BMR reg
AddressReg5 = 0008A0
DataReg5 = 00000000000000000000
#write block 3 BMR reg
AddressReg6 = 000830
DataReg6 = 00000000000000000000
#write block 9 BMR reg
AddressReg7 = 000890
DataReg7 = 00000000000000000000
#parallel search reg
AddressReg8 = 000202
DataReg8 = 00000000000000000000
#write range matching reg
AddressReg9 = 080400
```

```
DataReg9 = 00000000000000000000
#key construction reg
AddressReg10 = 000206
DataReg10 = 00000000000000000003

[Data Write]
NumberEntries = 4
#write block A data at addr 0  (80bit usage)
Address0 = 00A000
DataX0 = 00000000000000000001
DataY0 = 000FFFFFFFFFFFFFFFFE
#write block 3 data at addr 0  (80bit usage)
Address1 = 003000
DataX1 = 00000000000000A00000
DataY1 = FFFFFFFFFFFFFF5FFFFF
#write block 9 data at addr 0  (80bit usage)
Address2 = 009000
DataX2 = 00000000000000A00000
DataY2 = FFFFFFFFFFFFFF5FFFFF
#write block 9 data at addr 1  (80bit usage)
Address3 = 009001
DataX3 = 18F17D526E0280C721D9
DataY3 = 0000000000FD7F300000


[Data Read X]
NumberEntries = 1
NumberBytes = 6
AddressReg0 = 00A000

[Data Read Y]
NumberEntries = 1
NumberBytes = 6
AddressReg0 = 00A000

[Shift DCLK]
#Direction 00 shift left , 01 shift right
Direction = 01
#Number defines the degrees of DCLK to be shifted (1 : 1.406 degree)
Number = 05

[CB Write]
NumberEntries = 2
CBAddress0 = 0002
```

```
CBData0 = 00000000000000000020000000000000000001
CBAddress1 = 0001
CBData1 = 00034623464436000002AAADFFFF004334632626


[Write and Compare1]
NumberEntries = 1
CBAddress0 = 0004
CBData0 = 12300000000000000001123000000000000000001
LTRSelect0 = 00
```

Reliability Test and User Interface Database Write function

The following two snapshots of Labview program code are supposed to demonstrate the basic functionality of the Reliability Test and the user interface. Figure C.1 is a part of the user interface. It is programed to read out the section 'Data Write' of the configuration file and sends the data to the USB port in hexadecimal values. The program in Figure C.2 uses a random number generator of Labview to write 1000 *Database Write* instructions into a configuration file with random data entries.

Figure C.1: The program is part of the Labview user interface and reads out the section of the Configuration File, which is used to store the address and data of all *Database Write* instructions.

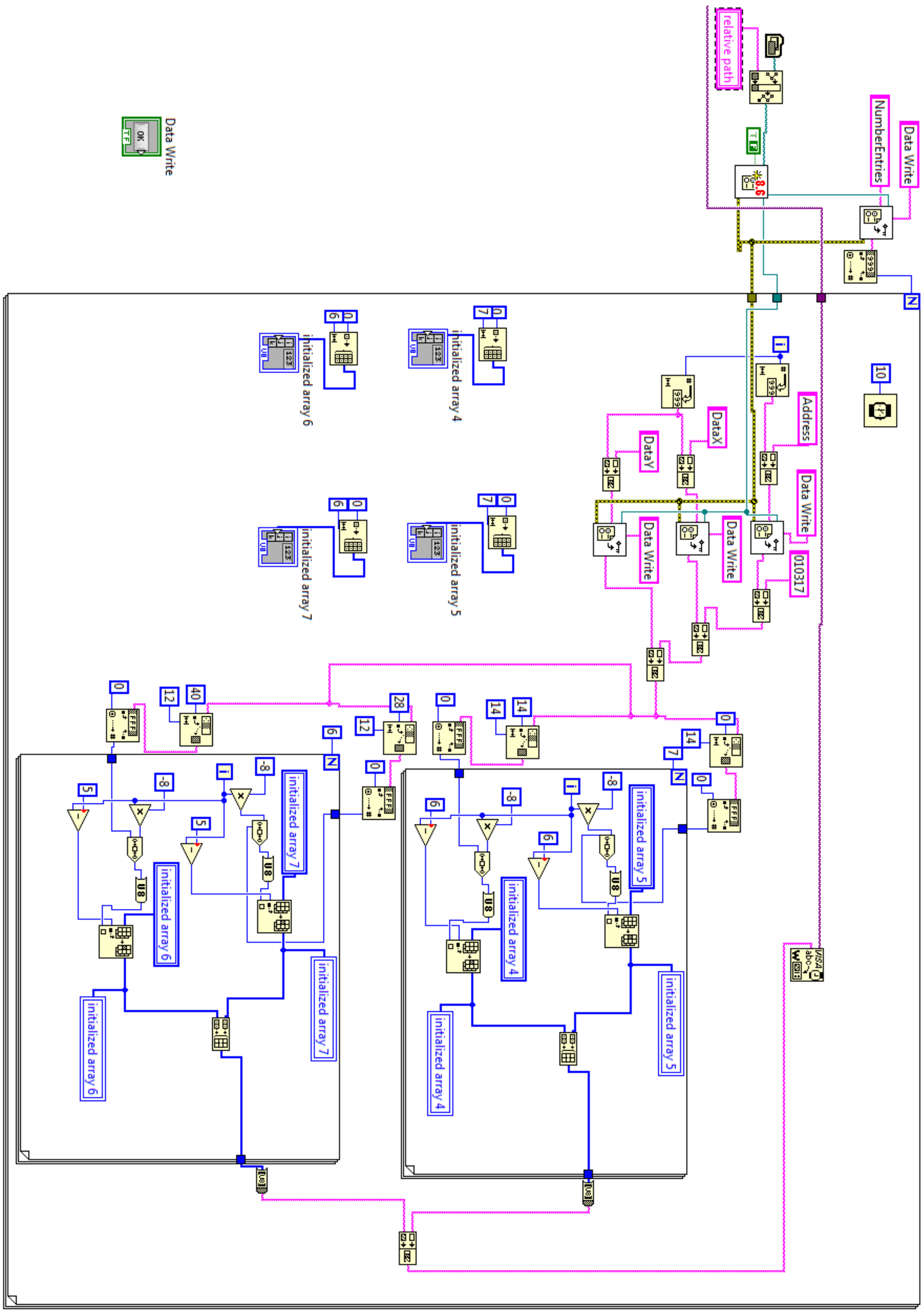
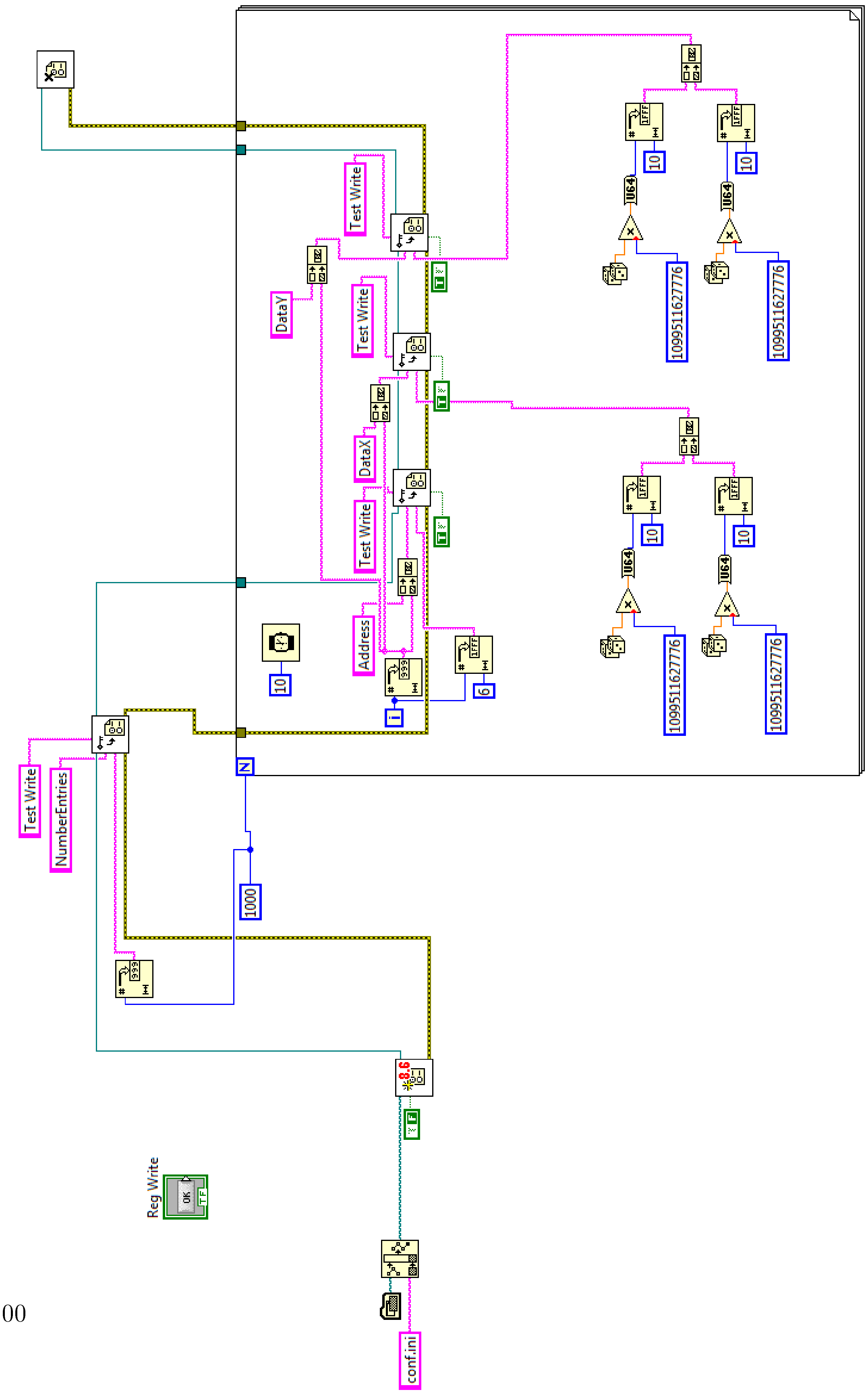


Figure C.2: The Reliability Test program generates 1000 *Database Write* instruction in a configuration file. The DataX and DataY values are generated with a random number generator.



Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den (Datum) ....