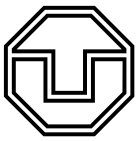


TECHNISCHE UNIVERSITÄT DRESDEN

Optimisation of the FPGA Firmware Implementation of Convolutional Neural Networks for the ATLAS LAr Calorimeter Signal Processing

Master-Arbeit
zur Erlangung des Hochschulgrades
Master of Science
im Master-Studiengang Physik

vorgelegt von

Johann Christoph Voigt
geboren am 15.07.1995 in Dresden

Institut für Kern- und Teilchenphysik
Fakultät Physik
Technische Universität Dresden
2021

Eingereicht am 28. April 2021

1. Gutachter: Prof. Dr. Arno Straessner
2. Gutachter: Prof. Dr. Kai Zuber

Abstract

The planned Phase-II upgrade of the LHC will increase the number of proton-proton collisions that happen simultaneously to improve the statistics for rare events. The drastically increased pile-up poses new challenges to the readout electronics of the detectors. At the same time, new hardware components enable more sophisticated real-time processing solutions. In this master's thesis, convolutional neural networks are implemented on FPGA firmware for the purposes of the signal readout for the LAr calorimeter of the ATLAS detector.
The focus lies on the accurate reproduction of the reference results from the software implementation in Keras and the performance optimisation of the firmware. This will enable the transfer of the neural networks under development into detector firmware with minimal effort, while fulfilling all hardware performance and latency requirements.
The result is a firmware implementation based on fixed-point numbers with 18 bit precision, that can run a convolutional neural network with around 100 parameters in four layers at a latency of 125 ns and a clock frequency of 480 MHz on a Stratix 10 FPGA. Based on this, the energy reconstruction on a detector cell level is possible in real-time for the trigger systems.

Zusammenfassung

Das geplante Phase-II Upgrade des LHC wird die Zahl der gleichzeitigen Proton-Proton-Kollisionen erhöhen, um eine bessere Statistik für seltene Ereignisse zu ermöglichen. Der damit verbundene starke Anstieg des Signal-Pileup stellt eine neue Herausforderung für die Ausleseelektronik der Detektoren dar. Neue Hardwarekomponenten ermöglichen aber zeitgleich die Nutzung von komplexeren Methoden zur Echtzeitdatenverarbeitung. Diese Masterarbeit beschäftigt sich mit der Implementierung von Convolutional Neural Networks in FPGA Firmware für die Signalauslese des LAr-Kalorimeters des ATLAS Detektors.
Der Fokus liegt dabei auf der exakten Reproduktion der Ergebnisse der Software-Referenzimplementierung in Keras, sowie der Performanceoptimierung der Firmware. Dies soll den Transfer der in Entwicklung befindlichen Netzwerke in Detektor-Firmware so einfach wie möglich machen und dabei alle Hardware- und Latenzanforderungen erfüllen.
Das Resultat ist eine Firmwareimplementierung, die unter Nutzung von Fixkommazahlen mit einer Bitbreite von 18 bit, ein solches neuronales Netz mit etwa 100 Parametern auf vier Ebenen mit einer Latenz von 125 ns bei einer Frequenz von 480 MHz auf dem Stratix 10 FPGA ausführen kann. Damit ist eine Energierekonstruktion auf Detektorzellenlevel in Echtzeit für das Triggersystem möglich.

Contents

1 Introduction

The Large Hadron Collider (LHC) [16] is the largest particle accelerator currently in operation and will stay in this position for a while yet. Its biggest success so far has been the discovery of the Higgs boson in 2012 by its experiments ATLAS [51] [1] and CMS [52] [10]. Since then the focus has moved towards characterizing the Higgs boson further and looking for signs of Beyond Standard Model physics. Because the relevant events are very rare, a higher luminosity is required and a better discrimination of background events. The planned Phase-II upgrade of the ATLAS detector is aimed towards raising the luminosity and updating the relevant systems to keep up with the higher rate of particles and reactions. A part of this upgrade effort is the overhaul of the liquid argon (LAr) calorimeter readout. Here, the higher luminosity introduces new challenges because of the increased signal pile-up. This master's thesis explores the feasibility of implementing machine learning solutions for the energy reconstruction of the LAr calorimeters on field-programmable gate arrays (FPGAs).
The real time energy reconstruction is used as input for the trigger system of ATLAS as well as later physics analysis. The LAr calorimeter systems in particular are responsible for providing energy measurements of photons, electrons, tau lepton decays, hadronic jets and general spatial resolution of the energy flow of the collision events. Especially in regard to physics beyond the Standard Model, the energy reconstruction is of high importance. Many of the predicted particles are not directly detectable and can only be seen in the detector as missing energy.

1.1 The Standard Model of Particle Physics

The Standard Model [19] [54] [45] [48] [17] [44] [21] [55] is the current mathematical description of elementary particles and their interactions. It evolved in the 1960s and 1970s as the unification of different theories and can currently describe three of the four fundamental interactions. The remaining force of gravity could not yet be described on the level of quantum physics.
There are three main categories of particles in the Standard Model. The leptons are fermions that interact with the weak and electromagnetic interaction. They exist in three generations. The most well known is the electron, since it is a component of normal matter. In each generation there is an electrically charged and an uncharged particle. The uncharged particle, referred to as neutrino, only

interacts through the weak interaction and is thus very hard to detect directly. The equivalent to the electron for the second and third generation are the muon and tau leptons.
Quarks are the fermions that interact through the strong, weak and electromagnetic interactions. They also come in three generations. Each generation contains a quark with positive and negative electric charge of $+\frac{2}{3}$ and $-\frac{1}{3}$ respectively. They cannot exist individually due to the gluon self interaction and the coupling, which increases at low energy scales. This is called confinement. Their most common bound states are the proton and neutron, which make up the main part of normal matter and can be found in the nuclei of atoms. Both of them belong to the group of baryons, which describes the particles consisting of three quarks. The superordinate group of quark compounds are the hadrons, which also includes the mesons consisting of two quarks. Exotic hadrons with more quarks are rare and the subject of ongoing investigations.
The gauge bosons in the Standard Model are the particles responsible for the interactions between other particles. All gauge bosons in the Standard Model have spin one and are therefore vector bosons. The photon is responsible for the electromagnetic interaction, the W and Z bosons belong to the weak interaction and the gluons convey the strong interaction.
Separate from the gauge bosons is the only elementary spin zero boson, the Higgs particle discovered in 2012 [1] [10]. It is an excitation of the Higgs field, which gives mass to elementary particles through spontaneous electroweak symmetry breaking [23] [24] [25] [15] [22].
An overview of all the particles of the Standard Model can be seen in figure 1.1.

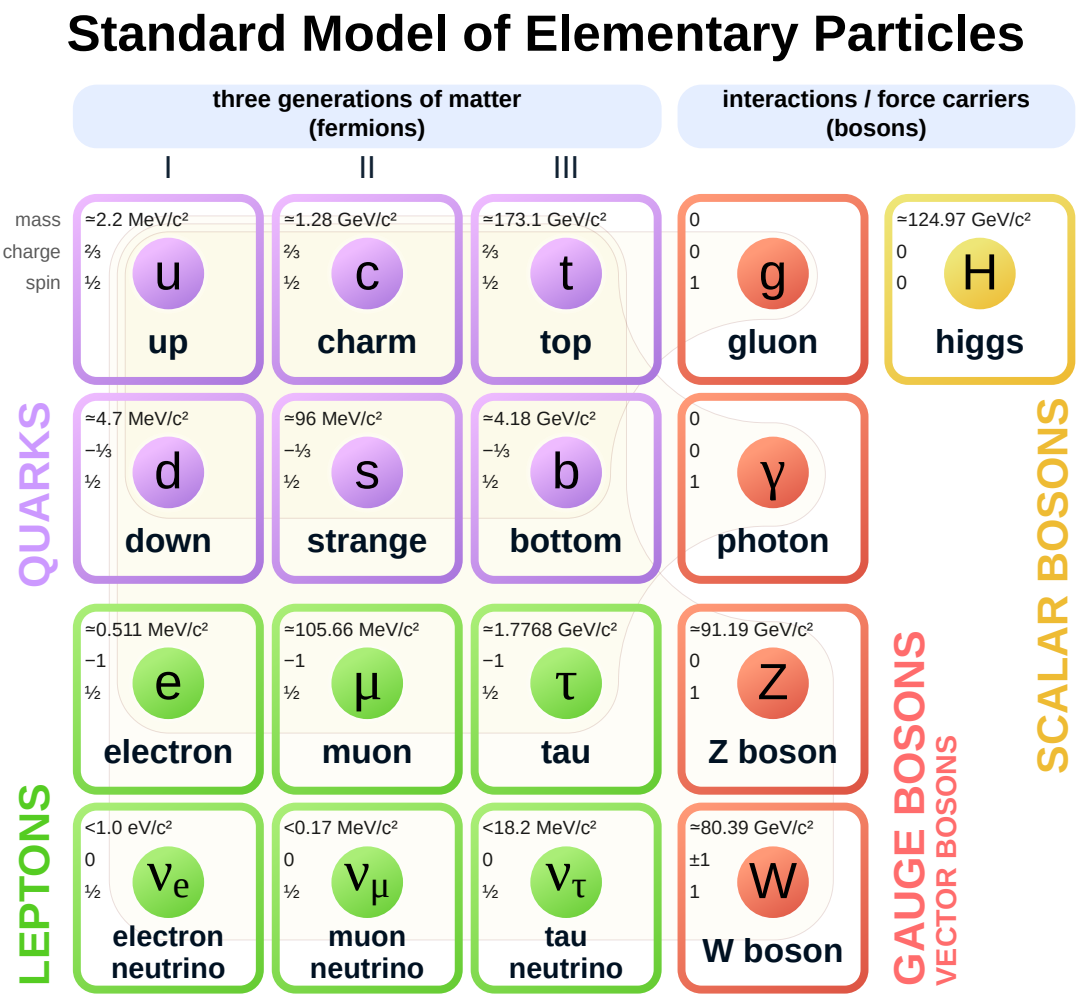


Figure 1.1: Overview of the particles in the Standard Model [41]

1.2 Beyond Standard Model Physics

There are several limitations of the Standard Model which hint at physics beyond this description. The most obvious problem is the exclusion of gravity, the fourth fundamental interaction. However, it is not expected that this problem will be solved at the LHC, since it becomes relevant at energies in the range of 10^{18} GeV [39, p. 3].

Another limitation of the Standard Model is the lack of a first order electroweak phase transition [34], which is required by the matter-antimatter asymmetry of the universe. Further examples are the gauge coupling unification problem, the lack of promising dark matter candidates and the fine-tuning problem [13, p. 5].

A popular theory that may solve these issues is Supersymmetry (SUSY). It introduces partners for all known particles with a spin of $\frac{1}{2}$ different from their Standard Model equivalent. Furthermore, the Higgs sector needs to be extended, for example from one to two doublets. This leads to further new particles, like a second scalar Higgs boson, a pseudoscalar A boson and two charged Higgs bosons. [13, pp. 7 sq.] This may solve the gauge coupling problem, since the model influences the gauge couplings and may lead to a proper unification at high energies. The R-parity commonly found in supersymmetric models leads to a stable lightest supersymmetric particle, that is a candidate for dark matter. The fine-tuning problem that leads to divergent terms in the Higgs mass is solved as well, since the new particles introduce loop corrections, which directly cancel out the problematic terms of the Standard Model particles. [13, p. 6] The search for hints of these new physics has been a major consideration in the design of the LHC and its experiments [51, pp. 2 sq.].

1.3 ATLAS and the LHC

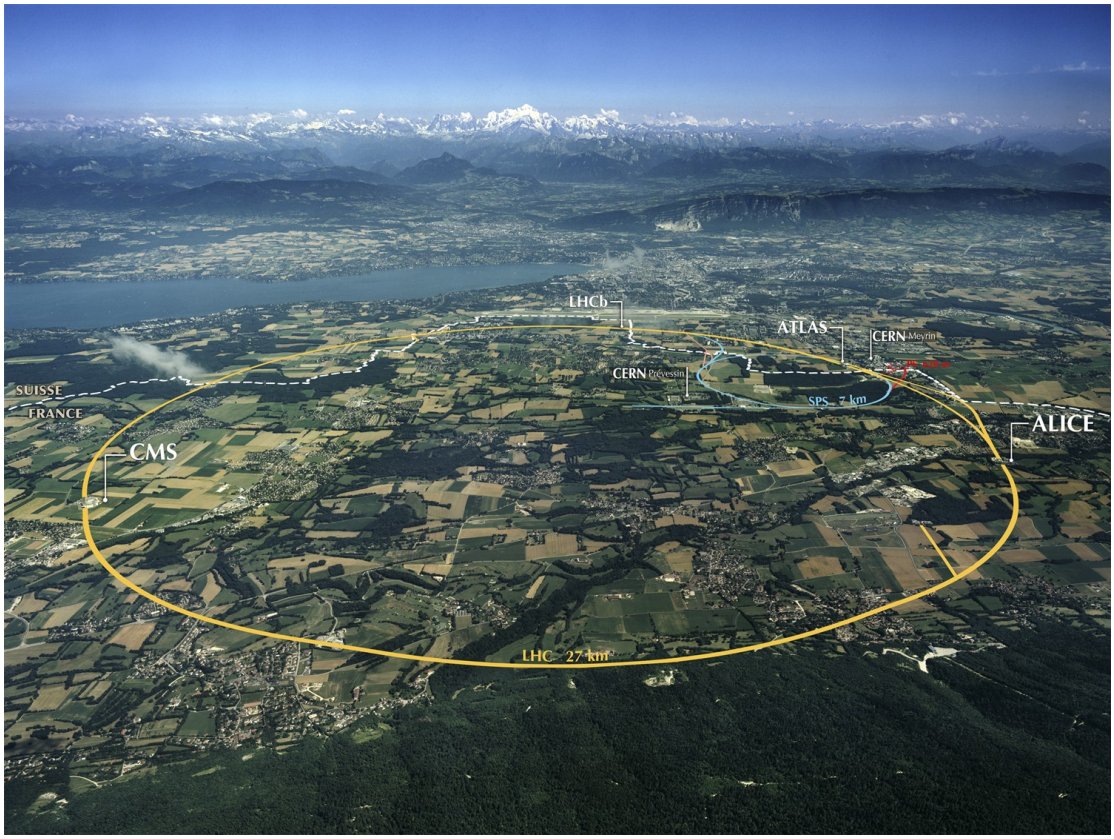


Figure 1.2: Overview of the LHC [8]

The LHC is a proton-proton particle collider located at European Organization for Nuclear Research (CERN) in Switzerland. The main ring with a length of 27 km allows the acceleration of two proton beams running in opposite directions to a centre-of-mass energy of 13 TeV and a design energy of 14 TeV. [16, p. 1] At four collision points the two beams are diverted to intercept. Around these points, the four experiments ATLAS [51], CMS [52], ALICE [49] and LHCb [53] are located.
Figure 1.2 shows an aerial view of the region above the LHC. The accelerator ring is located underground and shown in the picture by the yellow circle. The main experimentation sites are also marked. The accelerated beam is not continuous, but split into bunches. This means, that at the interaction points, there will be so-called bunch crossings at pre-defined times. At those times, the packets of protons intercept and interactions happen. The frequency of bunch crossings is 40 MHz. At each bunch crossing there have been $\mu = 36$ reactions on average in the latest run in 2018 [50, p. 3]. Due to technical reasons, the real bunch train structure is more complex. There is not a collision at each of these time steps, which has implications for the background handling in the detectors.

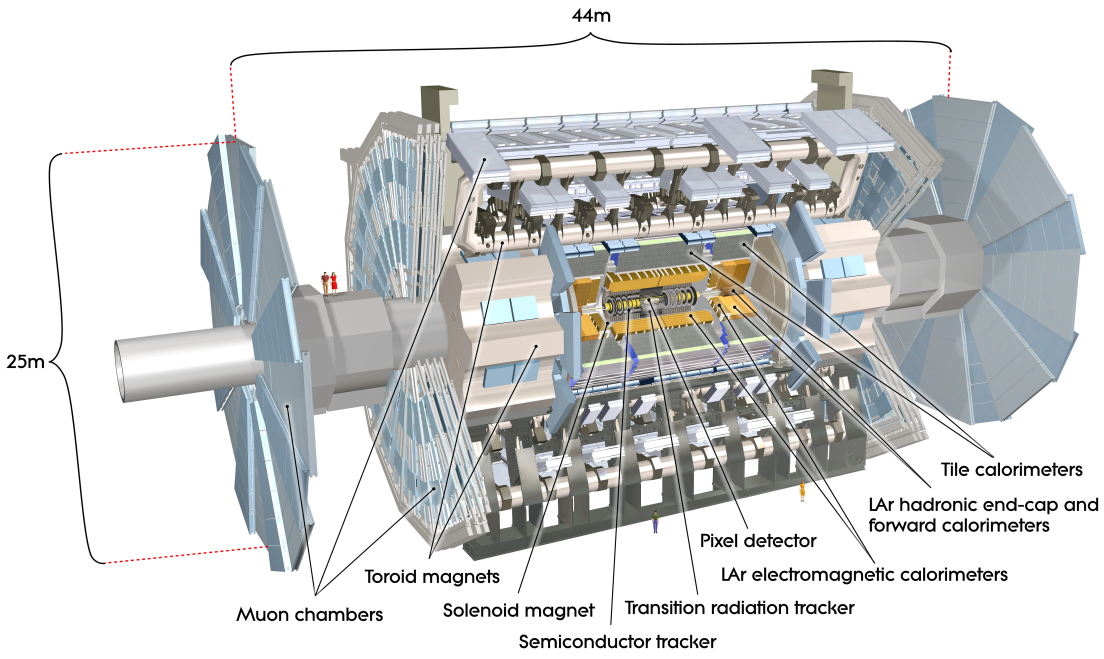


Figure 1.3: Overview of the ATLAS detector [43]

Figure 1.3 shows a schema of the ATLAS detector. It is one of the two general-purpose detectors located at the LHC. Therefore it consists of the main components found in most particle detectors in an onion-like structure: an inner tracking

system, electromagnetic and hadronic calorimeters and a muon spectrometer.

1.4 The LAr-Calorimeter

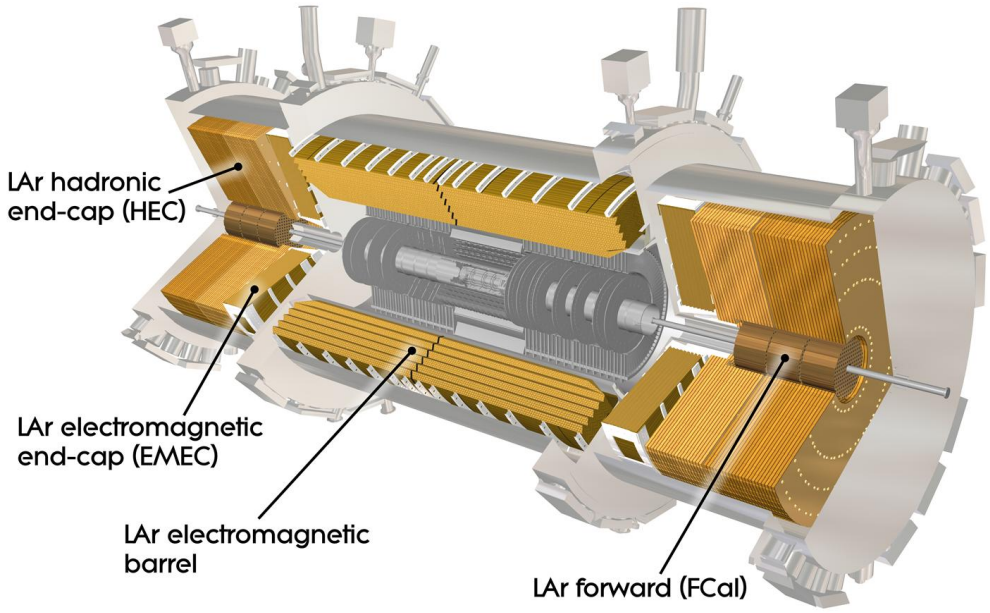


Figure 1.4: Overview of the LAr calorimeter [42]

An overview of the LAr calorimeter systems at ATLAS can be seen in figure 1.4. The electromagnetic barrel part of the LAr calorimeter is of most interest for this thesis, but the results obtained here can also be applied to the other LAr calorimeter sections, like the electromagnetic and hadronic end-caps, as well as the forward calorimeters.

The LAr calorimeter is a sampling calorimeter, where layers of absorber material and active medium are alternated.

When an electrically charged particle passes through the detector material, it looses energy through three main mechanisms: pair production, Bremsstrahlung and Compton effect. This leads to the production of a shower of secondary electromagnetically interacting particles. The goal of a calorimeter is the full absorption of the target particles. The amount of produced charges in the detector material

within this shower of secondary particles is then proportional to the deposited energy.
Figure 1.6 shows the structure of the LAr calorimeter in the barrel region. The absorber material is lead in a mantle of stainless steel for better structural integrity [37, p. 161]. As the name of the system suggests, liquid argon is used as the active medium.
Electrodes create an electric field across the thin LAr gaps. Therefore, the charged particles begin to drift towards the electrodes and induce a triangular pulse in the readout electronics depending on the voltage, active medium and detector layout. The electronic pulse in the readout can be seen in figure 1.5. This is then formed in an analogue pulse shaper into a bipolar pulse with a quickly rising peak and a longer undershoot, which is shown in the same plot. The integral over this pulse is zero, which means, that no net current is flowing.
This pulse is then digitized and a digital filter is used for the energy reconstruction. This is complicated by the fact, that there are a lot of low energy particles being absorbed all the time, which produce the same signal shape, just with lower amplitudes. Also, if a particle is absorbed during the undershoot of the previous pulse, the amplitude will be reduced accordingly. As can be seen in the plot, previous pulses influence the signal for a time much larger than the time between bunch crossings (BCs). The amount of expected pile-up is driven by the number of proton-proton collisions per BC. Therefore, its average $\langle\mu\rangle$ is an important parameter that characterizes the pile-up background. In-time pile-up occurs in the same BC as a high energy hit in a detector cell and can only be corrected on average, but not for an individual data sample. Out-of-time pile-up is the influence of previous pile-up events on the measured hit energy in the detector cell. With sufficient knowledge of the characteristics of the detector cell and its readout electronics this can be corrected, since there is separate information available about the pile-up hits from the previous time step. Apart from this, normal thermal noise of the electronics exists. To compensate these two main noise sources, a combination of the analogue shaping and digital filtering as introduced here is used.
In the current detector readout the analogue pulse shaper is a $CR-RC^2$ filter with a shaping time of 13 ns. This is done for three different gains to get a total dynamic range of 16 bit using 12 bit analog-to-digital converters (ADCs). [5, pp. 735 sq.]
The energy reconstruction through digital filtering is done by an Optimal Filter (OF). Its coefficients are chosen using the characteristics of the expected noise and pile-up for that detector region. This allows the filter to minimize the impact of both random noise and out-of-time pileup.

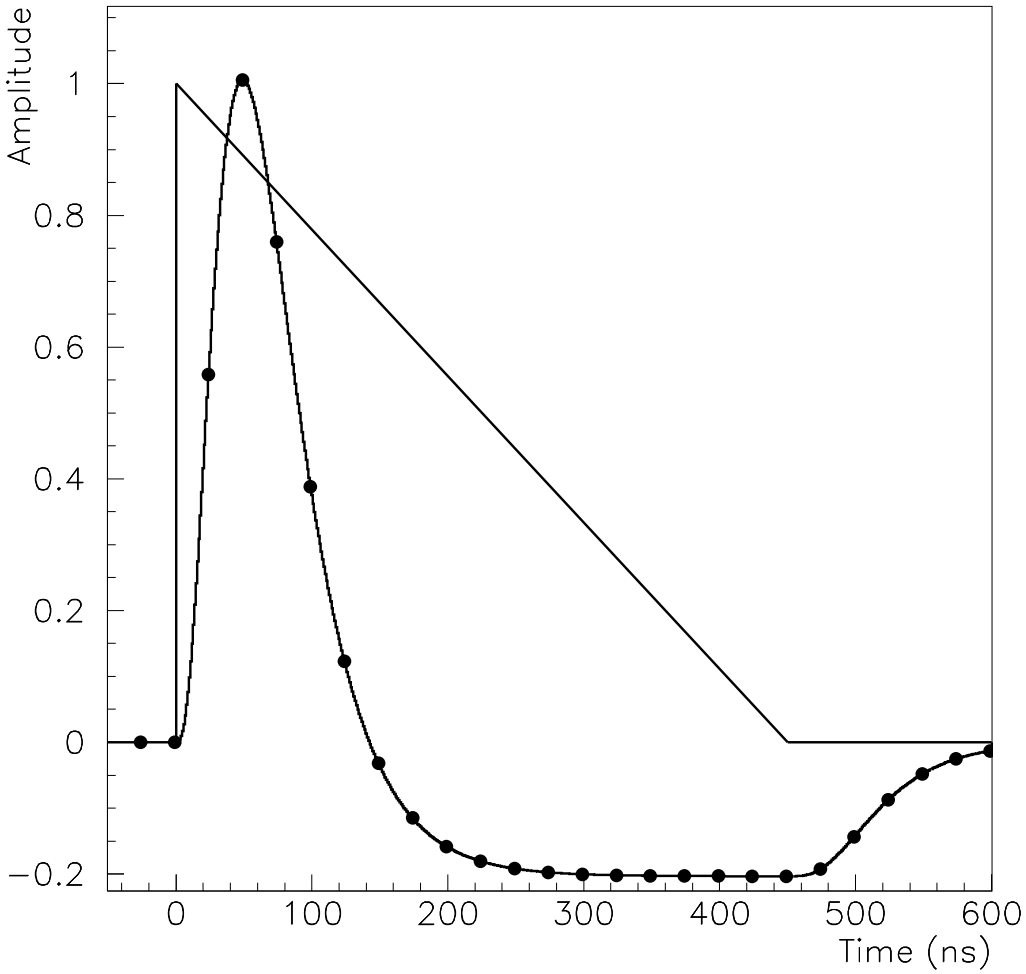


Figure 1.5: LAr detector signal shape before and after analogue pulse shaper [4]

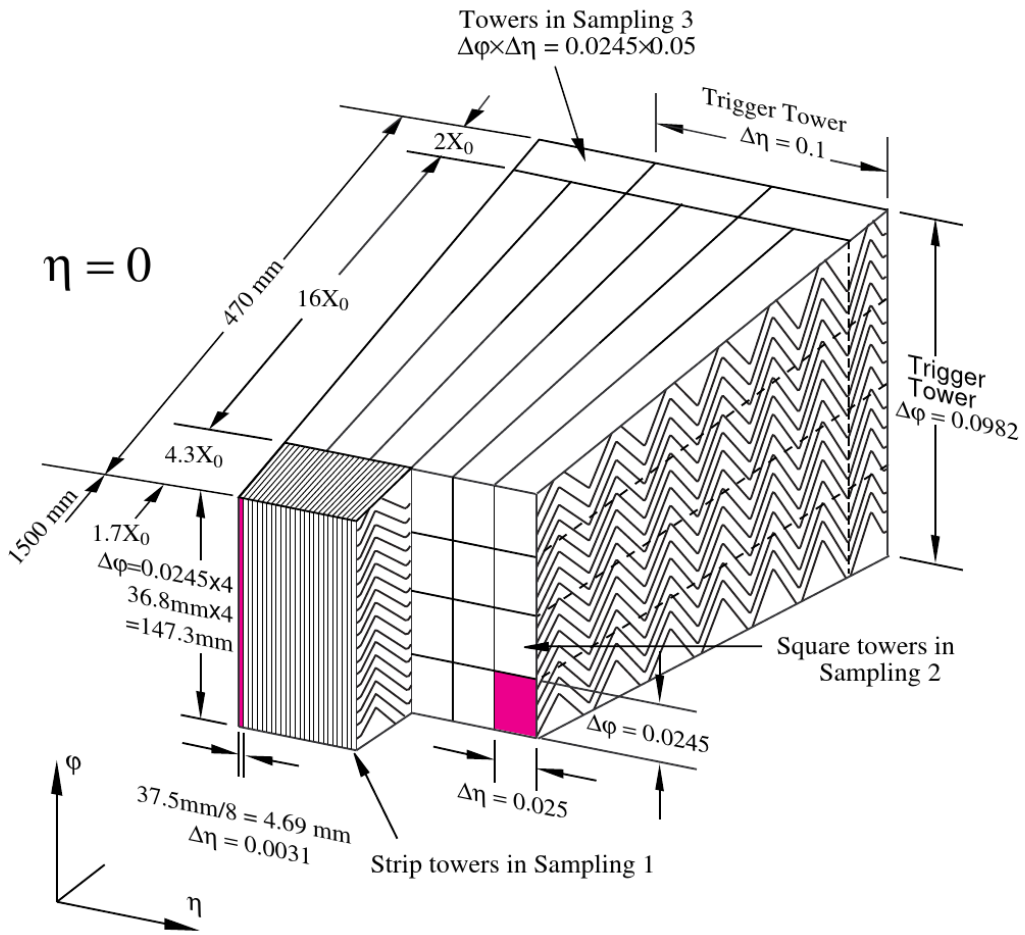


Figure 1.6: Structure of the LAr calorimeter [37, 5, Fig 1-2]

1.5 Current Optimal Filter Approach

Currently, the energy from the LAr-Calorimeter is reconstructed using an OF as described by Cleland and Stern in [12]. Its goal is to extract the signal amplitude and time of a hit in a detector cell with the maximum signal-to-noise ratio in an environment with random noise and pile-up. The filter itself uses a linear combination of a certain number of consecutive input samples S_i to reconstruct the energy A and time deviation of the hit [12, pp. 474 sq.]:

$$A = \sum_i a_i \cdot S_i \tag{1.1}$$

$$A\tau = \sum_i b_i \cdot S_i \tag{1.2}$$

The filter depth is usually set to five. The weights $\vec{a} = a_i$ and $\vec{b} = b_i$ are defined through

$$\vec{a} = \lambda V \vec{g} + \kappa V \vec{g'} \tag{1.3}$$

$$\vec{b} = \mu V \vec{g} + \rho V \vec{g'} \tag{1.4}$$

based on the unit amplitude shaper response $\vec{g}$ and its derivative $\vec{g'}$. This is a known property of the detector readout chain and the analogue pulse shaper in particular. The helper variables can be calculated according to

$$\lambda = \frac{Q_2}{\Delta} \tag{1.5}$$

$$\kappa = \frac{-Q_3}{\Delta} \tag{1.6}$$

$$\mu = \frac{Q_3}{\Delta} \tag{1.7}$$

$$\rho = \frac{-Q_1}{\Delta} \tag{1.8}$$

$$\Delta = Q_1 Q_2 - Q_3^2 \tag{1.9}$$

$$Q_1 = \vec{g}^{\mathrm{T}} V \vec{g} \tag{1.10}$$

$$Q_2 = \vec{g'}^{\mathrm{T}} V \vec{g'} \tag{1.11}$$

$$Q_3 = \vec{g'}^{\mathrm{T}} V \vec{g} \tag{1.12}$$

$$V = R^{-1} \tag{1.13}$$

R is the noise autocorrelation function and can be calculated as the sum of the thermal electronic and pile-up noise autocorrelation functions. These values are known from the expected pile-up in the detector cell as well as the characteristics

of the readout electronics.

1.6 Phase-II Upgrade Plans

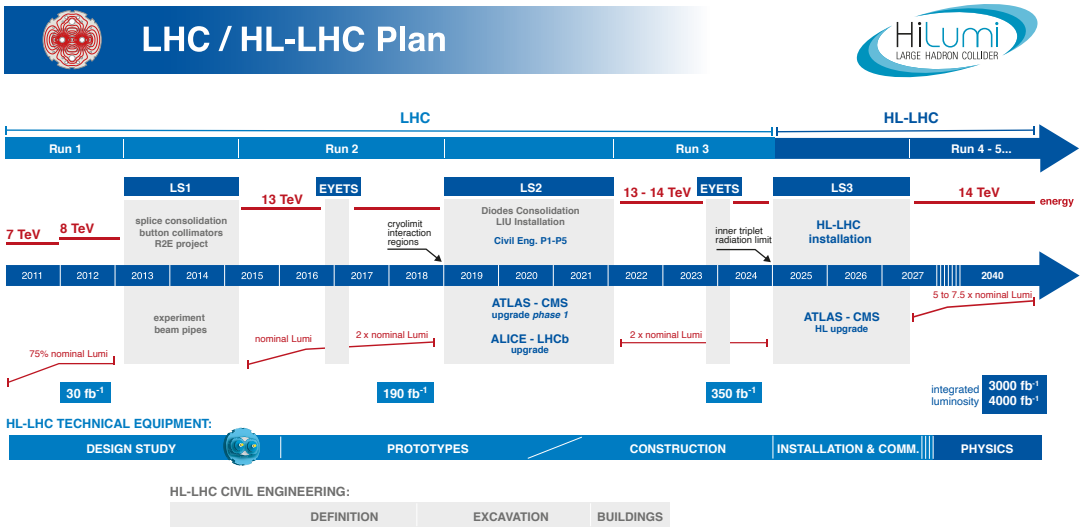


Figure 1.7: Current schedule for the Phase-II upgrade [9]

The LHC did not start in its final configuration right away. Both energy and luminosity are being increased incrementally over the years. Especially during the long shutdown periods, there are major upgrade works going on. The first such period increased the centre of mass energy to 13 TeV and the luminosity to the nominal values. At the time of this master's thesis, the LHC is undergoing modifications during the long shutdown 2, which is scheduled to end in early 2022. Together with the LHC itself, the experiments have their own upgrade plans during these phases. For example, the ATLAS detector is undergoing modifications under the name of Phase-I upgrade. The next long shutdown is planned for 2025 and will enable the Phase-II upgrade of ATLAS. Similar work is being done on CMS. An overview of the LHC upgrade steps can be seen in figure 1.7. The Phase-II upgrade will increase the expected total integrated luminosity to 10 times the predicted value of the original design [6, p. 3]. The increase in luminosity leads to more interactions per bunch crossing, which happen at same frequency of 40 MHz as before. This means, that a lot more particles are produced at those time steps and the detectors need to be able to handle this increased rate accordingly. Current projections estimate the possible pile-up at around $\langle\mu\rangle = 140 - 200$, meaning the average number of events per bunch crossing. This will allow the collection of an

integrated luminosity of $300\,\mathrm{fb}^{-1}$ to $400\,\mathrm{fb}^{-1}$ per year. [6, p. 4] For comparison, the 2018 run delivered an integrated luminosity of $63.4\,\mathrm{fb}^{-1}$ [50, p. 3].
The hardware and software algorithms developed in this master's thesis are aimed towards the Phase-II upgrade. This upgrade will replace the current readout systems completely to make the front-end electronics more radiation hard and make the readout compatible with the new trigger system. [36, p. 3] The new system should also be able to handle signal pulses in consecutive or very close BCs. The machine learning approach presented here is a promising alternative to the traditional OF for the digital energy reconstruction to solve the arising issues. The new readout system will enable the use of more complex digital filter systems inside the Liquid Argon Signal Processor (LASP) component [36, p. 37]. This module will be the framework for the firmware developed in this master's thesis. The detector readout will contain 1524 Front-End Board 2 (FEB2), which can process 128 calorimeter readout channels each. Here, the analogue pulse shaping is applied before the pulses are digitized at 40 MHz. To cover the desired dynamic range of 16 bit, two separate gains will be used together with 14 bit ADCs. The output is then converted into an optical signal, that can be transmitted via approximately 32 000 optical $10.24\,\mathrm{Gbit\,s}^{-1}$ links to the off-detector electronics. The data is then processed on the FPGAs of the LASP system. Each FPGA has to process the data received from up to four FEB2s, meaning a total of 512 detector readout channels with two separate gains each. A total of 372 FPGAs will be mounted on 186 boards. The processed data is sent to the Global Event Trigger Processors and the Level-0 calorimeter trigger Feature Extractor (FEX), as well as the data acquisition (DAQ) data readout. [36, pp. 77 sqq.]
While the total delay for signals sent to the L0Calo FEX can be up to 1.7 µs [36, p. 85], the available time for the energy reconstruction via digital filtering on the FPGAs is significantly shorter and budgeted at 125 ns [36, p. 163]. For the purposes of this project it is assumed, that a moderate increase of the latency budget for the energy reconstruction can be achieved through optimisation of other steps and use of the available margins. A value of up to 150 ns is therefore deemed acceptable at this stage of the project.

1.7 FPGA and VHDL

FPGAs are reconfigurable hardware chips, that can be used to represent different digital circuits in firmware even after the initial hardware production is finished. The layout of the digital circuit can be changed any time, like a software device can be reprogrammed. This is possible through a programmable interconnect matrix, that allows flexible connections between the internal logic elements. This can be supported through programmable or configurable logic elements to gain more flexibility.

Modern FPGAs use look-up tables (LUTs) as their main logic units with a bit width of about 7 bit. For specialized purposes they also contain other elements like clocks, phase-locked loops (PLLs), digital signal processors (DSPs) as native multiplication units and dedicated faster long distance connections to distribute key signals.
Their main advantages over the more common central processing units (CPUs) and graphics processing units (GPUs) lies in their very high possible transfer rates of input and output data and the extensive use of parallelisation and pipelining in the data processing. Since the desired design is implemented in hardware, a much more specialized circuit will be used to process the data. Instead of a fixed general purpose circuit, that reads instructions at runtime to determine how to process the data, here a circuit is produced, that is specifically tailored to the desired application and cannot be used for anything else until it is reprogrammed. While enabling the full flexibility to program any circuit design onto the FPGA, the adopted firmware can only offer the runtime functionality it was designed for. In this regard, FPGAs are similar to application-specific integrated circuits (ASICs). Those, however, cannot be reconfigured after the manufacturing is complete. For this reason, FPGAs are used during development and prototyping as well as in small series productions. ASICs are better suited for large production series and will be cheaper, more performant, need less power and be less susceptible to radiation than FPGAs, since their design is less complex due to the missing interconnect matrix, reprogrammable LUTs and connected circuitry. All this flexibility adds a lot of potential failure points in the case of radiation exposure and also degrades the performance, since it adds extra logic elements in comparison to a fixed ASIC. However, this is not an issue in the context of this work, since the developed firmware is targeted towards the off-detector electronics, which are not subjected to high radiation doses.
The real time energy reconstruction for the LAr-Calorimeter has high bandwidth requirements, since there are 182 468 [36, p. 9] readout channels. They all need to be processed. This data rate cannot be channelled to a traditional software based processing device like a CPU or GPU easily. An FPGA instead allows much higher and more flexible connections to the data transfer lines from the detector.
This work will make use of an Intel Stratix 10 FPGA [31]. The available development kit [30] can be seen in figure 1.8. This board contains the specific FPGA model with the reference number 1SG280HU2F50E2VG. This is the *GX 2800* version. Most relevant for the following considerations is, that the device contains 5760 DSPs, which are specialized units used for multiplications, and 933 120 adaptive logic moduless (ALMs), which are the main logic units the FPGA uses to represent different circuit layouts. The speed grade *E2V* determines the maximum frequency the DSPs can operate at. Depending on the mode of operation, this is at least 578 MHz [29, p. 32].
The circuit design to be prorgammed onto the FPGA can be described in a hard-

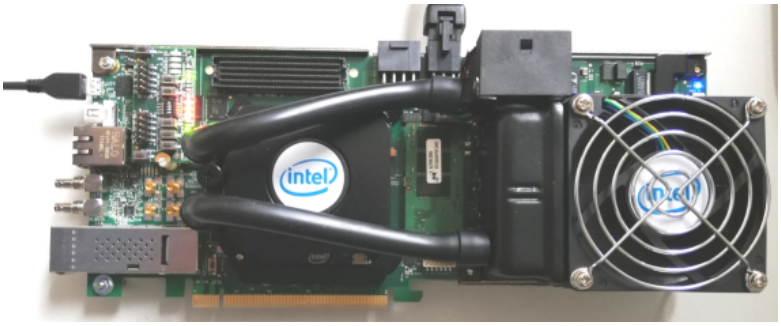


Figure 1.8: Stratix 10 Development kit [18, p. 39 (Fig. 19)]

ware description language (HDL), which is in some regards similar to a programming language. Here, the language Very High Speed Integrated Circuit Hardware Description Language (VHDL) is used. To translate this very abstracted VHDL code to a digital circuit design and a format that can be programmed onto the FPGA, a synthesis tool is necessary. For this FPGA model this is the Quartus [28] software by Intel. During development, it is also helpful to simulate the design, since the full synthesis is time and resource consuming and data extraction from the FPGA in real time for debug purposes is challenging. For this purpose, specialized simulation software like Modelsim or Questasim [46] exists. Here, a version delivered with the Intel Quartus development tools is used [33].

1.8 Artificial Neural Networks

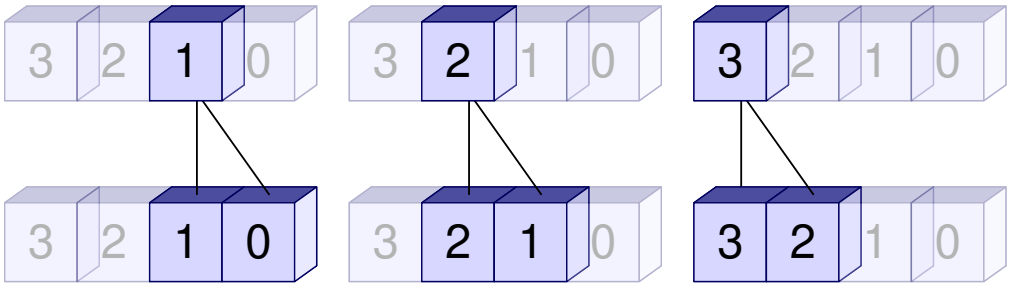
1.8.1 Overview of Network Types

Artificial neural networks (ANNs) in general are a group of machine learning algorithms loosely modelled after the biological neurons found in living organisms, albeit severely modified and adapted to the necessities of digital data processing. Goodfellow, Bengio, and Courville define a machine learning algorithm as "an algorithm that is able to learn from data" [20, p. 97]. They proceed to cite the definition for what learning means in this context by Mitchell (1997): "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."
In general, the network itself is a layered structure, that operates on some input data and calculates output values based on this. Each layer represents a mathematical function that is applied to the output of the function of the previous layer. As long as there is no recursive or feedback structure, the network is classified as a feedforward network [20, p. 164]. The other case is a recurrent network, but those are not relevant in the context of this work. Usually, the connections between

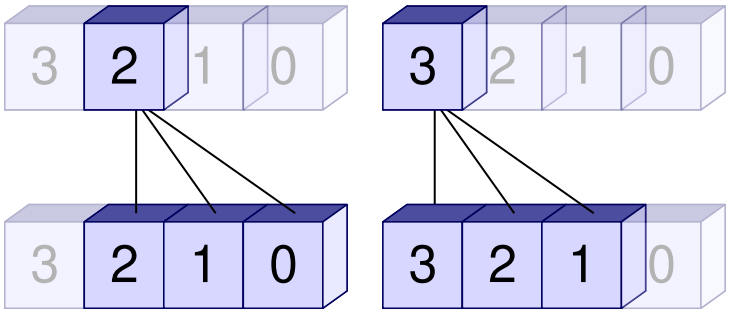
those layers and the architecture of the layers itself are fixed by the developer. Generally, each layer consists of a number of neurons which operate on the layer input in parallel. They process their input by multiplying it with fixed weights, summing the results and applying a nonlinear activation function, which has been defined by the developer.
What sets ANNs apart from classical signal filter solutions is the way how the weights used in the calculation of the output are obtained. The learning process itself is transformed into a minimisation problem with the help of a loss function. An example for such a learning algorithm, that is used for the networks for this master's thesis, is the Adam optimiser [20, pp. 305 sq.]. While the architecture, provided data samples and the general settings of this learning algorithm are handled by the developer, he does not take any direct action during the learning process itself and lets the algorithm figure out on its own how to make the network fit the provided training data. The parameters that the training algorithm cannot change during this training process are called hyperparameters.
In contrast to linear models, like fitting a linear function or system of linear equations, a nonlinear activation function is used in neural networks. In consequence, iterative algorithms that make use of the gradient are used in training. This does not guarantee a global minimum of the loss function. [20, p. 173].
Machine learning algorithms can be categorized as either supervised or unsupervised during the training. Here, supervised algorithms are used. This means that the training data set contains the desired output value for each data sample. The learning algorithm then optimizes the network parameters to be able to predict the associated output value with the provided input. In unsupervised learning, the learning algorithm would be tasked with finding interesting features in the training data itself. [20, p. 103].
There are a lot of possible architectures for these networks. For this work the most important type is the convolutional neural network (CNN). This type is very well suited for the specific purposes of this application and the implementation on FPGAs.
The network training is done using the TensorFlow [40] and Keras [11] libraries for Python.
The training and validation data is generated using the ATLAS Readout Electronics Upgrade Simulation. (AREUS) software [38] [47]. This program provides a modular toolbox to simulate the entire readout of the LAr calorimeter from the detector cell to the digital energy reconstruction. This can be used to generate digitized samples like the real ADC would provide them. AREUS includes all relevant noise sources, like electronic noise and out-of-time pile-up. The result is a realistic representation of the electronic noise and pile-up energy distribution. It supports a wide range of properties of the accelerator and detector. For example, the proper bunch train structure can be simulated. For the verification data, a sequence with regular high energy hits and a typical pile-up background is used.

1.8.2 Convolutional Neural Networks

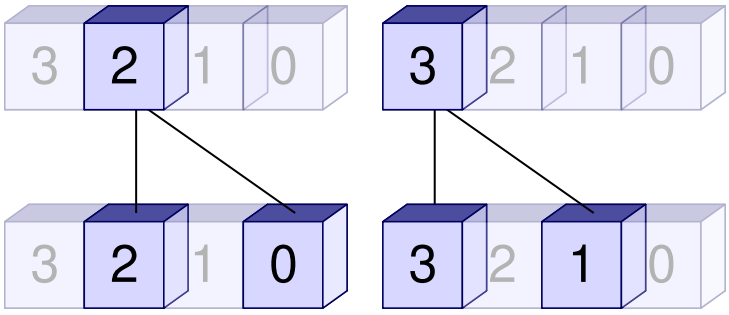
CNNs are typically used in applications for processing structured 2D or 1D data [20, p. 326]. This network type provides several key benefits for the planned application. In comparison to fully connected networks, there are a lot fewer connections. This reduces the number of parameters and thus the amount of resources needed to calculate the network output. This is achieved by effectively reusing the same parameters over the entire input data. The network operates like a sliding window over the input sequence. This is similar to the mathematical convolution operation. The size of the sliding window is the kernel size, which has to be smaller than the input data size. [20, pp. 329 sqq.]
A special benefit regarding the time series data from the LAr calorimeter readout is, that CNNs can process continuous data streams. This means, that it can be applied to an input data stream for any timespan without altering the network structure. Of special note here is the equivariance to translation of this network type [20, pp. 334 sq.]. The expected pulses from the detector mostly have the same shape independent of when in the time series they occur. This property of CNNs means, that it reacts to two identical pulses at different points in time in exactly the same way.
Figure 1.9 shows the basic working principle of a convolutional neural network layer operating on a continuous series. In figure 1.9a the kernel size is two and the dilation one, which means that no input values are skipped. The lower chain of blocks represents the input, the upper chain the output values. The numbers in the boxes represent the time step this input or output value belongs to. When moving to the second and third subpicture, the time step increases by one respectively. As the kernel size suggests, there are always two input values contributing to one output value, which is highlighted. Each connecting line represents a multiplication by a fixed weight. Before the output value is obtained, those intermediate results need to be summed up, the bias added and the activation function applied. Figure 1.9b shows the process for a layer with kernel size three. The only difference here is, that three input values are considered per output value as the kernel size suggests. It should also be noted, that the networks here are all causal, which means, that only input values of the current and past time steps can be considered. The last subplot 1.9c shows the principle for a kernel size of two again, but this time with a dilation of two. This means that there are still two input values considered per output value, but between all consecutive input sample one value will be skipped.
To understand the benefit of dilation, one has to look at the range of input values considered for one output values of the network. This is called receptive field. For the single layers shown in 1.9, the receptive field is two for the first case without dilation in 1.9a, and three for the same kernel size and thus number of parameters in 1.9c. This effect increases in influence for multiple layers, since each input



(a) Kernel 2, dilation 1



(b) Kernel 3, dilation 1



(c) Kernel 2, dilation 2

Figure 1.9: Working principle of a layer in a causal CNN for different kernel sizes and dilations

values for the shown layer may again be calculated using the spread induced from the dilation. The benefit here is, that the network can look further into the past, while retaining the same amount of parameters. There may be more information in the spread out input samples than just the consecutive sequence of the latest input values.
Each layer may consist of multiple feature maps, which all operate on the same input, but operate independently and have their own sets of weights. The next layer then gets the output of every feature map of the previous layer.
CNNs are well suited for the purposes of this project, since they recycle their weights a lot. Where other networks need weights for every input value, CNNs can perform well with just a small set of weights, which get applied to all the input values. This is perfect for applications, where the resources during application of the network to data are severely limited.

1.9 Trigger Performance of the Convolutional Neural Networks

The trigger performance of the network can be characterised using receiver operating characteristic (ROC) curves. They show the background rejection as a function of signal efficiency. This is done by varying the threshold over which the trigger output is considered as a hit between zero and one. For each value in this interval, the amount of correctly identified signal hits and correctly rejected background events is calculated by comparing the neural network prediction to the true result and added as a data point to the plot. This means that the perfect trigger would have a square ROC curve that reaches the top right corner of the plot. In practice this cannot be achieved, but it is desirable to get as close to that corner as possible. Both of the axes are important, since signal efficiency without background rejection is useless and vice-versa. A perfect signal efficiency can be achieved by triggering for every event, a perfect background rejection by never triggering. Both of these cases are obviously undesirable. Similar plots can be obtained from energy reconstruction algorithms by varying an energy instead of trigger threshold. Everything above the threshold is classified as a signal hit detection by the algorithm, for everything below the threshold it is assumed that the algorithm classified this as a background signal.
Such a ROC curve can be seen in figure 1.10 comparing the trigger performance of different trigger and energy reconstruction algorithms for a pile-up parameter of $\mu = 140$ as it is expected after the Phase-II upgrade of the LHC. The red line shows the performance of the OF, which is the benchmark and baseline scenario for all alternative energy reconstruction methods under investigation. The yellow and blue lines show the performance of a CNN with three and four layers respectively.

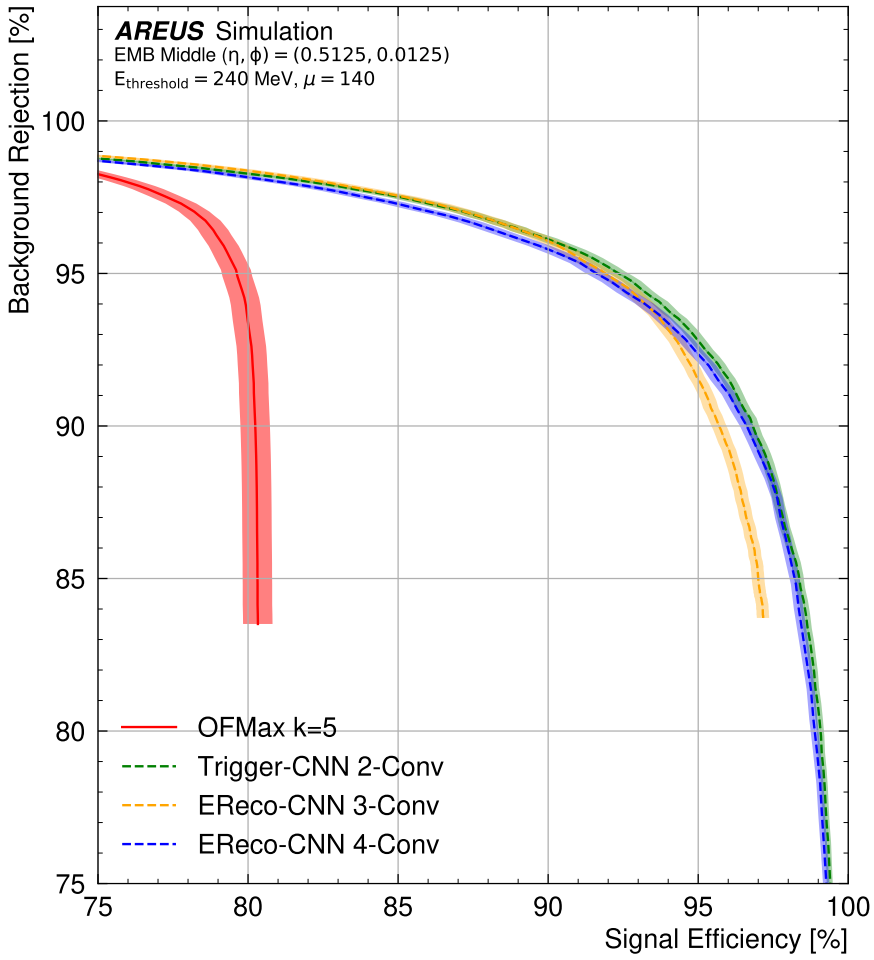


Figure 1.10: ROC curves showing the trigger performance of the trigger and energy reconstruction neural networks compared to the Optimal Filter. [7, p. 8]

Those networks are trained to reconstruct the hit energy. They both contain a trigger subnetwork with two convolutional layers responsible for detecting signal hits over an energy of 240 MeV. The performance of this part as a separate network is shown in the plots in green. The training procedure for these networks is described in [2]. More details about the network architectures are presented in section 4.1.

It is very clear, that all the neural network solutions have a much better signal efficiency. This means that true signal hits are much more likely to be correctly classified as signal by the CNNs than the traditional OF. While the OF barely reaches 80 %, the CNNs only drop of after 95 %. In the displayed interval, all neural networks outperform the OF at all signal efficiencies.

The best performance is reached by the isolated trigger network, closely followed by the four layer energy reconstruction network. The three layer energy reconstruction network has similar performance up to signal efficiencies of 93 %, but drops off afterwards and the curve ends much sooner without reaching any lower values of background rejection independent of the signal efficiency. This is due to the fact, that this network has more problems reconstructing very low energies below the threshold of the trigger subnetwork due to its reduced number of layers. Since this is only a software implementation, the next challenge is to port these neural networks onto the FPGA platform and evaluate the performance of the resulting firmware implementation compared to the software reference. The accuracy of the energy reconstruction or trigger performance may degrade depending on the approximations necessary in the firmware version to be able to meet the latency and resource requirements.

1.10 HLS4ML

An alternative approach to bring machine learning solutions onto FPGAs is the High-Level Synthesis for Machine Learning (HLS4ML) package. It allows the description of neural networks on a higher level language, which is then ported to code synthesizeable on the FPGA. It offers a flexible and highly customizable implementation. HLS4ML has been written specifically with the application in high energy physics trigger systems in mind and is thus similar to the context of this work [14, pp. 2 sq.].
A very notable feature of HLS4ML is the ability to reuse DSPs a specified number of times. This allows a trade-off between latency and DSP usage. It is possible to tweak the reuse parameter from a dedicated DSP for every multiplication all the way to only using a single one for the entire network. [14, p. 13].
Other techniques used to reduce the resource usage are the use of fixed point numbers and compression techniques on the network training level, like pruning. [14, p. 11]
However, this package is only beginning to support Intel FPGAs. Also, many of the advanced features, like using DSPs multiple times, are not useful in the ATLAS LAr readout scenario due to the very stringent latency requirements. Therefore it was decided, that the ANNs are to be directly implemented in firmware using VHDL instead of using this pre-made framework. This will allow much more optimisation towards the specific use-case than the HLS4ML solution.
An alternative approach for the LAr energy reconstruction with more complex networks, that uses HLS4ML for the conversion into FPGA firmware, is also presented in [2].

2 Trigger Networks

2.1 Previous Work

The firmware developments presented here are the continuation of previous work [18] by Nick Fritzsche on the implementation of neural networks on FPGA.

The adopted VHDL code consists of several components, which, when put together, model the network and necessary control logic. The basic building block is the *connections.vhd* file, which contains the entity that calculates the output of one feature map of one network layer. This module has generic input ports for the basic configuration of that layer, like the kernel size, dilation and other network parameters, as well the bit widths and number of fractional bits for the different signals used. The signal input and output ports are used to get access to the weights and input of the respective layer and output the result of this feature map.

The entity in *ann.vhd* constructs the network out of the building blocks of *connections.vhd*. Each layer consists of a number of those entities according to the number of feature maps in that layer. This module creates those entities using the VHDL *for-generate* statement. This allows a flexible structure, which can be configured from a config file read in during compilation or simulation time. The instances of the *connections* module are linked together and provide the appropriate weights and necessary configuration values in the superordinate *ann* component.

Depending on whether the network is to be simulated or synthesized, this is included into a test bench or a wrapper with some additional control logic. However, while the code can be executed on an FPGA, it is not possible to input or output values to and from the FPGA yet, except when using the utility program Signaltap from Intel. To control the functions of the module later, a slow control module is under development by Rainer Hentges within the LASP project. While the weights can be loaded with the help of the available random-access memory (RAM) on the development board, there is currently no signal injector available to simulate the ADC input stream. Therefore, the verification is mostly done by simulation in Modelsim or Questasim and the performance is estimated using the Intel Quartus compilation reports.

2.2 Piecewise Linear Approximation of the Sigmoid Activation Function

Due to the very low level nature of the logic components on the FPGA, the nonlinear activation function cannot simply be calculated on the chip. The DSPs are also not capable of calculating a function as complex as this directly and the resource usage would be prohibitive. The simplest solution is a LUT with the required bit width. This is easy enough to implement on the FPGA, since it contains a high number of programmable LUTs anyway, albeit with a much lower bit width. These can be composed together to form a larger LUT without using up the more limited supply of DSPs. The necessary program code for the LUT can easily be generated by a Python script.

Different approaches have been considered on how to potentially replace the LUT for the activation functions with a more elegant, more performant or a solution using up less logic cells. One promising option is a composition of linear approximations in a series of subintervals. One such approach is presented by Amin, Curtis, and Hayes-Gill [3] under the name piecewise linear approximation of a nonlinear function (PLAN). They approximate the sigmoid function as constant for x-values greater than five. The interval $[0, 5]$ is divided into three distinct subintervals. In each of those intervals a linear function is used with the salient property of having a slope that can be expressed as a power of two term, including negative exponents. The boundaries between intervals are chosen to achieve a continuous function. For negative values of x the symmetry of the function is utilized and the result can be calculated according to $f(-x) = 1 - f(x)$. The result of this approximation approach can be seen in Figure 2.1.

The special conditions for the slope are chosen to allow the replacement of all multiplications by bit shifts. In a further step of abstraction, the entire approximation can be replaced by a logical function mapping input bits to output bits when using fixed point numbers [3, p. 314]. This is possible, since all required multiplications are just bit shifts and can therefore be easily replaced by logical functions. Similarly, the check for the appropriate approximation subinterval for the current input value can be expressed like this. A subtraction step is still required for negative input values, but the use of the symmetry otherwise halves the required resources.

The suggested direct transformation for a bit width of eight is presented in [3, p. 315] and can be implemented in VHDL without issues. However, this is a very rigid solution, since it provides a solution only for the fixed bit width including four fractional bits for the input and seven for the output.

It is possible to extend this approach for higher bit widths, when using fixed point numbers with more fractional bits. This requires only some modification of the transformation mappings. The result is a better discrete approximation of the

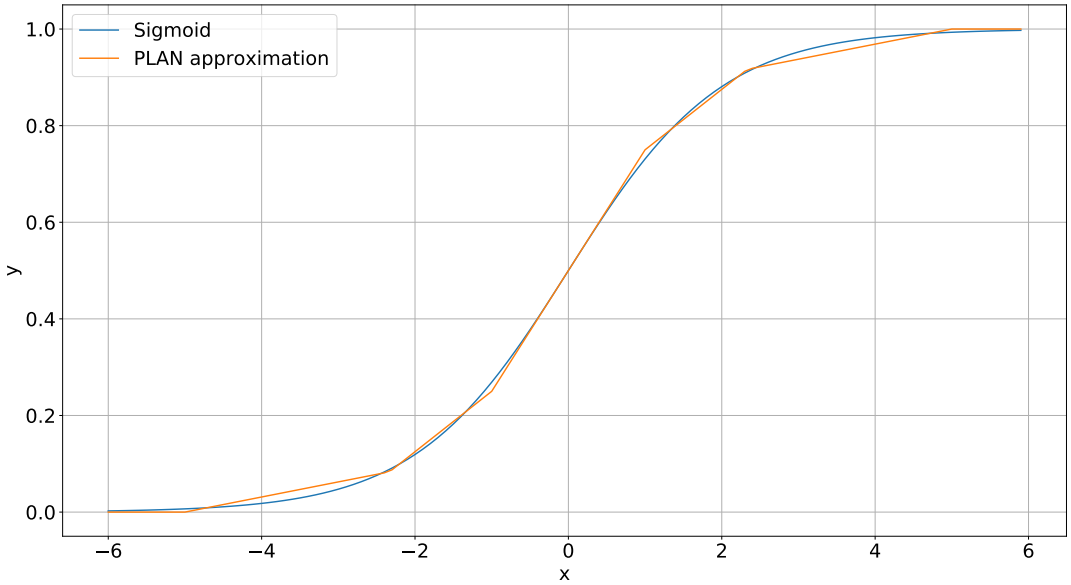


Figure 2.1: Sigmoid function and PLAN approximation according to [3]

linear functions that make up PLAN itself. The number of subintervals and the linear functions themselves remain unchanged for this. This means that only the discrete implementation increases in accuracy, while the inherent deviations due to the linear approximation and intervals remains the same. However, this has potential if PLAN is already used during network training. The derivative can be calculated from the approximation itself by the following property of the sigmoid function [3, p. 313]:

$$f'(x) = f(x) \cdot (1 - f(x)) \tag{2.1}$$

This ensures a continuous derivative, although the exact derivative of the approximation would have discontinuities. Then the higher bit width allows the hardware implementation to more closely match the results of the reference implementation, which uses floating point numbers. One definite improvement of the more flexible bit width is the easier integration with the rest of the code, which uses a configurable number of bits and fractional bits for most signals. The fixed nature of the original PLAN did not fit this concept well. And, while the accuracy may not improve much due to the inherent deviations of the linear functions used to approximate the sigmoid function, the results will have a higher granularity, meaning that different input values will lead to different output values more so than with the original PLAN, since there is less rounding necessary before the application of the activation function. Of course one could argue, that this higher granularity is purely the illusion of a higher accuracy, since the inherent deviations from the

approximation are dominant.

2.3 Dilation Support

Some trigger networks make use of dilation, as explained in section 1.8.2. It helps to increase the receptive field, without the need for more network parameters. This feature was previously not supported in the VHDL implementation. To add this, a clever buffer of the input values is necessary. Previously, each neuron buffered its input values depending on the kernel size. In each clock cycle, the buffer is shifted by one step forward and the latest input value inserted. To implement a dilation d, this same buffer can be used, but its size needs to be increased. Instead of kernel size k the buffer now needs a size of at least

$$s_{\text{buffer}} = d \cdot (k - 1) + 1 \tag{2.2}$$

not considering other technical reasons that may require more delay for the buffer. This is due to the fact, that there will still be k values required for the calculation, but they are spaced out more due to the dilation d.

2.4 Automatic Conversion of Keras Files to Configuration for VHDL Implementation

The network architecture can be specified in a configuration file. This contains a package with the definition of a number of VHDL constants, which define the network. To facilitate this process, an automated configuration tool has been created. It is able to read the Keras model files obtained from the network training and generate the configurations necessary for the VHDL representation of the network. This allows to reliably and reproducible configure larger networks without errors in the translation. All the automated tasks are controlled by a central Python script. A command line flag tells the script to either prepare the files for simulation or synthesis, since the target directories and files differ slightly between those options. If an input sequence is provided, the tool is able to prepare the simulation environment for the network so it can be tested directly without further manual interventions.
Global parameters like the bit width of the internal signals and the number of fractional bits for input values and weights can be specified in a configuration file. If this file does not exist, it is automatically generated with a set of default values. This file has the standard *ini* file structure to be easily accessible.
In the first step, the script searches for the necessary input files in the provided directories. This is the JavaScript Object Notation (JSON) file with the network

description, as well as the Hierarchical Data Format 5 (HDF5) file with the network weights from the Keras training. The directory of the Keras model files and the input sequence in the HDF5 file format can be specified as command line options. The script identifies the necessary files from those directories automatically. It can also read in multiple input sequence files and combine them into one longer sequence. This is required to apply the network to larger datasets, since the available sequences generated by AREUS are fragmented into smaller parts, while the VHDL simulation works better with a single large sequence.
Keras itself is used to read in the model files. To be exact, the Python library *TensorFlow.Keras* is utilized, since it was also used to write those files after the training. Despite Keras not being needed for anything else here, this is still the most elegant solution, since this script does not need to be designed with the exact structure of the Keras files in mind. It will also be compatible with different versions of Keras, where the file structure may change in subtle ways, as long as the application programming interface (API) functions remain the same.
The conversion is only possible for a limited range of network architectures and features, as they are required by the specific networks under investigation for this project. If necessary, new features and network types can be added to this procedure at a later time. However, the script does support all features of the VHDL implementation and is thus complete for its specific purpose.
Several decimal input files are generated afterwards. This is most importantly the input sequence based on the provided HDF5 files, but also stimulus files that govern the enable signal for the calculation as well as the reset signal and a signal required to begin the process of reading the network weights and bias values from the input array. The network weights are written into such a decimal text file as well and shifted by the correct number of steps to match the timing of the other provided control signals. For an alternative version of the code that reads in the weights in the VHDL model using the on-board RAM of the FPGA, a *mif* file with the weights is also provided. This can be used to initialize the RAM with the correct values. Whether the decimal sequence file or this RAM initialisation file are used in the simulation can be chosen in the test bench code. The conversion script here provides the necessary input for both options. In case of synthesis, only the RAM initialisation file can be used. The same goes for all other decimal input and stimulus files: They cannot be used in the synthesized project. Since this information is not necessary during the synthesis, the project can still be compiled. However, at this point it is not possible to really test it on an actual FPGA, since no proper input sample injector and wrapper framework exists yet. When the VHDL project is embedded into such a framework later on, the respective file formats necessary at that point will need to be added to this conversion script to be able to still use it.
After all the network parameters have been identified and sorted, they are applied as patches to a provided template configuration file using a regular expression

search and replace approach. The template file can be any valid configuration file. No special preparation, like masking all the configuration values, is required. The function used to prepare and buffer the patches beforehand already takes care of whether a specific variable is a single value or an array and formats it accordingly, so that a valid VHDL file is output. The template file is only edited in very specific positions, as only the values of the patched generics are replaced by the new values. This means that the rest of the file remains largely unchanged, including all comments and formatting. This approach results in a separation of VHDL code and the automation script, since no actual code needs to be generated. The resulting code on both sides is therefore much more readable.
The script outputs a warning for all generics in the template, for which no patch has been defined. This helps with identifying problems in the conversion process. It is desirable, that the script defines patches for all generics in the file to really be independent of the values defined in the template file. If variables were not overwritten by the script, the initial values from the template file would be kept, which is not intended behaviour.
After this string manipulation, the configuration is written to the target directory used by the simulation and a shell script with the correct command line options for a script to later compare the results between the VHDL implementation and the Keras reference model is generated.

2.5 Comparison and Verification

The networks implemented in VHDL need to be verified to identify potential issues with the code and judge the influence of the inherent deviations due to the simplifications made in the model. The verification can be directly integrated with the automation tools presented in the previous section. Since the simulation software is able to write the output of the network into files, it can be easily accessed. This will not be possible when the firmware is tested on an actual FPGA. With the help of this simulation setup, issues in the implementation can be identified and fixed during development more effectively. This will be replaced by a proper process verification framework in later stages of the project.
The output from the VHDL networks needs to be processed before it can be compared to the results from Keras, which provides the reference model run on a regular CPU based computer.
To read the input sequence, the same functions used by the conversion script can be reused. The results from the VHDL simulation are read from the hex file and converted to floating point numbers using the provided number of fractional bits. The Keras model and its weights are loaded by Keras and the model applied to the input sequence. This means, that there are now the simulation results as well as the reference results available for comparison.

However, the VHDL results are still shifted by a certain number of bunch crossings. The exact shift depends on the implementation and the latency of the test bench, as well as architecture of the simulated network. While this shift can in theory be calculated by this evaluation script, it can also be determined by comparing it to the reference results. The latter approach is more robust against changes to the VHDL code, since it does not have to be changed every time, the latency of the model changes. This concept is therefore chosen. Starting with a shift of zero, the sequences are shifted relative to each other step by step. For each step, the root mean square (RMS) of the difference is calculated. The final result for the shift is the value where the minimal RMS is observed. To reduce the effort of the calculation, it is sufficient to only consider positive shifts of the VHDL sequence relative to the Keras results, as well as set a reasonable upper limit for the shift. The RMS is also output and can be used as a first indicator of the quality of the simulation results and determine whether it is likely, that any problems or bugs occurred somewhere.
To verify the results from the VHDL simulation, a plot of the differences to the reference model is an obvious choice. Of special interest is a plot of those differences as a function of the true deposited energy in the detector cell. This helps to separate problems in the reconstruction of low energy hits and pile-up events at or below the noise threshold from high energy hits and see how the influence of the numerical simplifications of the VHDL implementation influence the trigger performance at different energy scales. To get access to this information, the original input files for the ADC samples can be revisited. Since they were generated by an AREUS simulation, they also contain further information like the actual deposited energy in the detector cell for each BC.
To judge the performance of the trigger networks after the training process, ROC curves as described in section 1.9 can be used. They show, how effective the trigger network can separate signal and noise events. The previous comparison between VHDL and Keras results is of a more technical nature. A comparison of the ROC curves from the two implementations will show the actual influence of the observed numerical inaccuracies on the trigger performance and thus the expected physics results. The pre-processing of the results described in this section already provides all necessary information to generate these plots for both models. The plotting itself can be done using the same code as it is used to evaluate network performance after the training and is provided by Anne-Sophie Berthold.

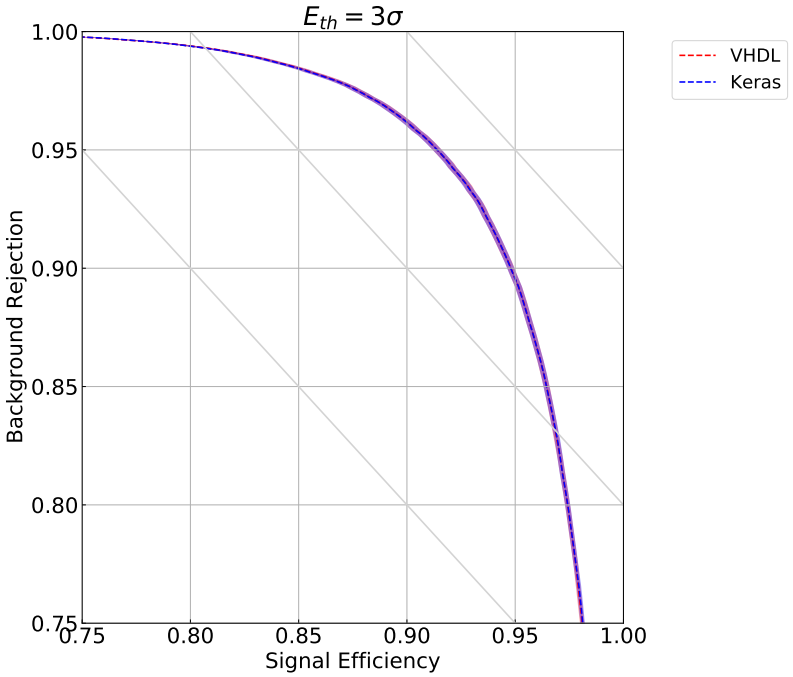
2.6 Integration into the LASP Framework

The LASP system is the detector readout component responsible for the digital filtering and energy reconstruction in the ATLAS LAr calorimeter. This firmware does not only contain the signal processing components for the energy reconstruc-

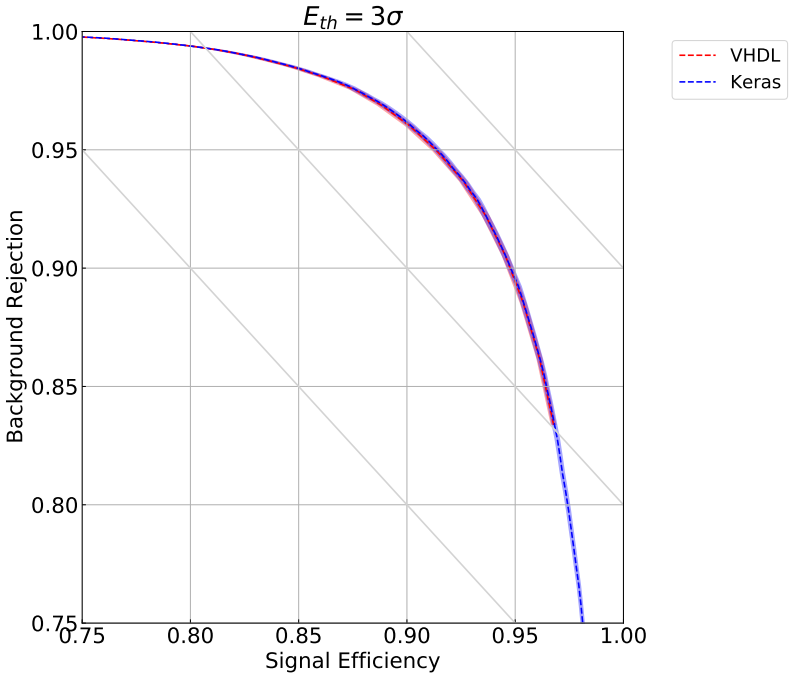
tion, but also additional features necessary for the operation of the energy readout. This includes functionalities to receive the ADC samples, buffer the data until it is requested by the following readout and trigger systems and transmit the data in the required formats and protocols.
In order to integrate the neural network component into this larger firmware project, several technical modifications were necessary. Apart from stricter requirements regarding coding style and interfaces, it also uses its own building pipeline and environment. The project was ported into this framework largely by Nick Fritzsche in parallel to the feature extensions described here. Those extensions were than added afterwards to the code inside the framework in a modified version as a merge request on the GitLab version control system. The fundamental structure of the neural network code did not change in this process. The principle of the introduced automation tools for the configuration of the VHDL implementation directly from trained Keras models was also unaffected.

2.7 Trigger Network Performance

A ROC curve for an example trigger network can be seen in figure 2.2. The classification between hit and background is done with the known true deposited energy based on the energy threshold of 240 MeV. The plot is for a trigger network with two convolutional layers, where the first layer uses ten feature maps. The kernel size is three for the first and two for the second layer. The second layer also has a dilation of two. Plot 2.2a uses the LUT for the sigmoid activation function and 2.2b uses the PLAN approximation with four fractional bits in the input to the activation function and seven bits for the output.
The plots show the curves for the reference model in blue. For this the network was used on the PC directly in Keras on the same input sequence. The red curves shows the result from the VHDL-model results using the Questasim simulation software. Both lines are in good agreement with the VHDL-model and are nearly indistinguishable. The most notable difference is, that the curve for the VHDL-model using the PLAN sigmoid function ends earlier and does not reach higher signal efficiencies. This is due to the rounding involved with the fixed point numbers and the PLAN approximation of the sigmoid activation function. The greatly decreased bit width of the PLAN approximation has only a very limited influence on the ROC curve. It is very remarkable, that the trigger performance does not suffer from either the limited bit width of the VHDL implementation in general or the approximated sigmoid activation function. This is due to the fact, that the network is only used to make a binary decision using the current threshold. Little variations due to rounding have very little influence. The general performance of the VHDL models is still significantly better than for the OF, since that had much worse trigger characteristics as discussed in section 1.9. The good



(a) VHDL results when using LUT for activation function



(b) VHDL results when using PLAN for activation function

Figure 2.2: ROC curve of a two layer convolutional network. Comparison between the simulated VHDL-model and the Keras reference for two different implementations of the activation function.

agreement of the VHDL and Keras models here therefore shows that the networks on the FPGA can also reach a much better trigger performance than the OF. It remains to be seen, how the performance will look for the energy reconstrution, as the reduced bit width will have more influence there.

3 Optimisation

3.1 Approaches in Optimisation

The optimisation of an FPGA design can happen with regard to different performance metrics. When talking about speed, there are three different variables that can be optimised for in this context: throughput, latency and timing [35, p. 1].
Throughput regards the amount of data that can be processed in a certain time. This value is predefined for this project, since the bit width and number of input channels per FPGA is largely fixed by other project requirements already [36].
The latency is of high interest for this project, since the energy and timing information is required by the trigger system. A reduction of the latency per layer of the neural network means that potentially more layers could be used. This means, that even after the latency requirements are met, a further reduction of latency gives more flexibility for the possible network architectures. This flexibility may not be needed though, since more layers do not automatically result in higher quality output, especially if the number of parameters in the network is fixed. The latency optimisation has therefore a very high priority, until the CNNs currently under evaluation meet the requirements. A further reduction is beneficial, but should not be prioritized in trade-offs.
The timing optimisation is of special interest in conjunction with the plans for time domain multiplexing. If one network implementation is to process the data of n input streams in sequence, it needs to run at n times the speed of the input. With the ADC sampling frequency of 40 MHz and a planned multiplexing factor of 12, a maximum clock frequency of 480 MHz is needed. This reuse of FPGA resources increases the number of detector cells, which can be processed per FPGA, by a factor of 12. Further improvements in the timing are a good safeguard against future changes that degrade the timing, but not in itself necessary. A higher multiplexing factor is not planned, even if this could save more resources, since it would allow more data streams to be processed per network implementation. With a fixed number of inputs per FPGA, this could allow the use of networks with more parameters. However, the CNN code is embedded into a larger design with other subsystems, which also have a target frequency oriented towards the 12 times multiplexing. Currently, there are no plans for an additional clock to drive a faster CNN data processing. The goal here remains to reach the frequency of 480 MHz with all features implemented. Some margin here remains very beneficial

though.
A key concept for reaching high throughputs on FPGAs and other hardware devices is the use of pipelining. Kilts [35, p. 2] compares this to the assembly line used in automotive production. This means, that a high number of specialized processing stations are combined to a chain. On the way to the finished product, each input value or car passes through all those stations in the same order. At each station, a specific processing or manufacturing step is executed. At a high level, this concept is used in the structure of the CNN code. The data take a linear path and there are no multiplication units that are passed twice. This means, that all loops regarding the network structure are unrolled as suggested by Kilts [35, p. 2]. Further optimisations can only be achieved with a higher execution frequency. Reusing multiplication units could potentially allow networks with more parameters. However, it is highly unlikely, that the desired execution frequencies can be met then, especially with the already introduced multiplexing feature. This fully unrolled structure also reduces the amount of control logic inside each layer. The use of pipelining does not automatically increase the latency, since the maximum clock frequency increases together with the latency measured in clock cycles. However, the maximum clock frequency is driven by the critical path. The longest time necessary for the connection between any two consecutive registers is therefore crucial. Since all other registers in the module will be using the same clock, their connections among each other will have the same delay as the slowest connection. Therefore, the use of more pipelining stages does degrade latency, while improving the maximum clock frequency [35, p. 5].
The optimisation of timing and thus the maximum clock frequency is more complex. It has already been established, that this can be improved by the introduction of additional pipelining stages. But there are other possibilities to improve the signal delay on the critical path. Many of these optimisations are very dependent on the specific logic operations between the involved registers. The measures to improve that timing include parallelisation, flattening logic, balancing registers and reordering of paths [35, p. 16]. Many of those methods can be applied by the synthesis tool itself. But it is hard to generalize, in which situation the specific tool will succeed and where more manual attention is required. This is also a very dynamic field, since the synthesis tools are in constant development and new and improved optimisation features are added.
Another approach is to reduce the number of parameters in the network. In contrast to the previously discussed solutions, this has a direct impact on the quality of the network output itself. For example, one can prune all the weights below a certain threshold. However, this technique is not well suited for the LAr data processing. When the CNNs are programmed onto an FPGAs, all the calculation processes will be mapped to certain areas and actual physical locations on the chip. It is simplest to implement the network in such a way, that each multiplication unit is assigned to one specific weight in the network and all the

connections in the ANN exist as actual connections on the FPGA and not just as a rule for a higher level controller on how to assign data to multi-purpose calculation units. This is somewhat connected to the described unrolled loops, but such an abstraction would be necessary for a dynamic model, that can handle pruned weights.
While being easier to implement, the rigid approach also reduces the overhead between layers, since there is no higher level control logic required to assign input values to the calculation units. Their input and output ports can just be connected by fixed paths with some static logic cells and registers in between. This reduces the number of interconnections required between far away regions on the FPGA and promotes short and direct paths in the mapping. However, the key benefit is that the weights are not required during synthesis and can be loaded at a later time, since they have no influence on the architecture. This makes it easier to program the same network architecture onto all the FPGAs dedicated to the LAr calorimeter, but use a slow control to provide the weights depending on the detector region or even the specific detector cell as necessary. The weights can later be varied without recompiling the networks. It is not necessary to run the synthesis tool for every single FPGA separately, which would require immense effort in either server infrastructure or time independent of the time required to train the networks for specific cells.
As a summary it can be noted, that pruning can be applied, but has no performance benefits, since all the necessary calculation units for the pruned weights will still be present on the FPGA and included in the mapping structure. Therefore, pruning makes no difference and may actually require more logic blocks depending on the implementation. Furthermore, all networks considered here have already been optimized to make good use of the available number of parameters.

3.2 Results of Optimisations

The most pressing concern regarding network performance is the maximum clock frequency. Since it is planned to use time domain multiplexing, the network is expected to run at 12 times the frequency of the LHC bunch crossings, which translates to 480 MHz. While the execution speed of the firmware may be different once the multiplexing is implemented, it will not be better than the current version with only one network instance and no multiplexing. Therefore it is crucial to improve the design to a point, where the desired frequencies can be reached.
The multiplexing itself is a clever way to fully utilize the potential of the FPGAs used. Since it can run at frequencies up to 1 GHz, it would be a waste to let them run at only 40 MHz. With the multiplexing, fewer FPGAs are needed to process all detector readout channels. Or inversely, if the number of available FPGAs is already fixed, the size of the ANNs that can be implemented, can be increased

accordingly.
The maximum clock frequency can be estimated by using the Quartus Timing Analyzer [27] reports from the synthesis step. Quartus estimates the frequency for four different scenarios, of which the model that assumes 900 mV supply voltage and 100 °C operating temperature is chosen. For the initial design, this is also the scenario with the worst performance. However, this not consistently the case for all following optimisation steps. It is still beneficial to stick to one test case as a performance measure. Further analysis of the expected environment can be done with the final firmware or when the performance reaches sufficiently high levels. The FPGA model used is the Stratix 10 version 1SG280HU2F50E2VG [31]. As a standard benchmark, a trigger network with four convolutional layers, a kernel size of two and three feature maps in all but the last layer is used.
With those settings, an initial performance value of 54.45 MHz is obtained. This is very far from the desired target value of 480 MHz. The design is further improved through incremental changes and recompilation, of which only the more important intermediate steps will be listed here. Since the clock frequency is always limited by the slowest connection between two registers, the identification of this critical path is very useful for targeted optimisations. The slowest data paths can also be seen in the Quartus synthesis reports. This helps the incremental optimisation process, since it is clear where an improvement will very likely result in better performance and where this is not immediately the case. The two easiest ways to address the slowest data path are to either remove logic from the path or introduce new pipelining registers and thus distributing the work between multiple clock cycles.
The calculation within one layer consists of four main steps here: First the correct input samples are identified, then they are multiplied by their weights, added up and finally the activation function is applied. The first step in optimisation taken was therefore to split the activation function into a separate clock cycle. This yields a frequency of 62.7 MHz. To decrease the compilation time, from now on the high effort optimisation of the compiler is turned off. The equivalent frequency of this step in comparison to the following ones is therefore 61.3 MHz.
In the current code, there are some overflow checks. While this is in itself a useful feature, it adds additional logic in the calculation path and slows down the processing. The information is also not used in a meaningful way at this stage. While overflows are an important topic, this should be addressed later and possibly be prevented by design for intermediate results. A check can also be reimplemented, once a procedure exists on how to handle the cases where this occurs. Therefore, the overflow checks are removed.
The multiplication and summation happen in separate steps in the current code, but still the same clock cycle. By removing the intermediate variable and doing both in one line, the compiler might be incentivised to merge the two operations together. This can yield a better timing, since the FPGA has dedicated multiply-

add units in the form of the DSPs. With these two changes, the clock frequency could be increased to 100.84 MHz.
This design still does all the necessary multiplications in sequence within one clock cycle. This is an approach commonly used in software solutions for CPUs. But this sequential approach, while being good for a single, fast processing unit, is not ideal on FPGAs. Here, intermediate steps can be parallelized by just splitting up the data path and merging it again afterwards. The bottleneck is expected in the multiply-add calculation block. Therefore, all the multiplications are now done in parallel and only summed up at the next clock cycle. Since the multiplications are now executed at the same time, the results of one step are not available anymore for the next one. Therefore, the merged multiply-add feature cannot be utilized anymore. However, the critical path gets much shorter, since now only one multiplication has to be done in a clock cycle instead of all multiplications defined by the kernel size of the layer. This results in a performance jump to 260.69 MHz. Now three of the mentioned four main steps in the calculation of the layer output are split into individual clock cycles. If the identification of the necessary input samples is also split from the multiplication, only a small increase in performance to 264.41 MHz can be observed.
At this point, all high level steps have been pipelined and no obvious solution for further performance increases exists without taking a look at the specific logic on the critical signal path. This can be done in the register-transfer level (RTL) viewer of the Quartus tool. This approach helps to identify further bottlenecks. Firstly, there are some globally shared signals. Those are always a challenge for performance, since the synthesis tool needs to maintain the proper timing and synchronisation of these signals. Potentially, this has to be done over the entire chip design in one clock cycle. Therefore, these should only be used with caution and removed where possible. Of these globally shared signals there exist four in the current design. These are the clock, reset signal, an enable signal for the calculation and the weights vector.
A look at the logic inside the network calculation reveals that the reset and enable signals drive more logic of internal registers than necessary. There is also an unnecessary feature for pruning networks. It compares the weights with a configurable threshold and sets them to zero in case the threshold is not passed. If this feature proves necessary, it can be replicated by already applying the threshold check offline to the weights, before they are uploaded onto the FPGA. But it has already been established, that pruning does not provide any performance benefits. Those issues have been addressed and the results are presented in table 3.1. Most notable is, that the performance actually decreases for some of those measures. This shows that the routing is a non-trivial task and may react in unexpected ways, especially to changes not directly on the critical signal path. The important part here is, that the critical path changes from the multiplication to the summation stage. And in contrast to the multiplication, the latter can actually be pipelined

Table 3.1: Maximum clock frequency and critical path for incremental changes in control logic of the CNN layer for an example network

Incremental modification	$f_{\max}$ in MHz	Slowest step
Initial pipelined model	264.41	Multiplication
Enable signal only drives output	265.89	Multiplication
Remove weight threshold check	255.75	Multiplication
Buffer weights before multiplication	251.83	Multiplication
Reset only output and input to sample buffer	245.04	Summation
Delay reset signal by one cycle	233.54	Summation
Delay enable signal by one cycle	239.23	Summation

further.
Therefore, the summation is further split into two stages. This results in a maximum clock frequency of 375.8 MHz when combined with all the modifications presented in table 3.1. When the 2-stage summation is implemented on top of the initial pipelined model without those other changes, the performance results look worse. A similar incremental modification based on this is shown in table 3.2. It can be seen, that the previously mediocre improvements or degradations are now actually useful and all contribute to the improved performance of the 2-stage summation model.
For higher kernel sizes or a lot of feature maps in the previous layer and thus more input values to a neuron, the number of intermediate results to be added up in one clock cycle increases to a level, where this is the critical path instead of the multiplication when using one summation stage even without the other modifications. In case of the 2-stage summation, this can still lead to a severe performance degradation, if many additions are required. This could be compensated for by further splitting the summation into multiple sub-steps depending on the actual

Table 3.2: Performance and critical path for control logic changes combined with two stage summation. The brackets in the critical path column indicate that there is no clear slowest step in the calculation.

Incremental modification	$f_{\max}$ in MHz	Slowest step
Initial pipelined model (w/o split Ident.)	260.69	Multiplication
Two stage summation	257.27	Multiplication
Delay reset and enable signals	276.7	(Multiplication)
Buffer weights	285.88	(Summation)
Enable signal only drives output	323.94	Multiplication
Remove weight threshold check	330.91	(Multiplication)

number of required add operations.
The multiplication is now running in a separate clock cycle without much other logic between the respective registers before and after it. But still, the maximum clock frequency is very far from the target of 480 MHz. This means that some other approach has to be taken to still gain significant improvements. It is therefore in order to take a closer look at the calculation units responsible for the multiplications, the DSPs.

3.3 Pipelining over Input Values

The trivial implementation until now only starts with the calculation for one specific output value, when all the necessary input values are available. This approach is shown in table 3.3 for an unpipelined model. By splitting those four main steps of the calculation inside a layer into four clock cycles and using a parallel approach for the multiplication, a significant performance increase measured in the maximum clock frequency has been observed.
However, even this improved calculation scheme does not make good use of the time delay between input values inherent to processing a time series in real time. The calculation for a certain output value can be started when the first necessary input value is available instead. This allows chaining the DSPs and makes use of their combined multiply-add functionality again. This works by multiplying the first input value by its weight and pushing the result to the next DSP, where it should arrive at the same time as the next input value from the previous layer or the ADC is available. The new input value is then multiplied with its weight and added to the intermediate result from the previous DSP in the chain. This means that only a single multiplication has to be done between the time the last necessary input value arrives and the output is calculated. An overview of this scheme is shown in table 3.4. All the cells of the same colour contribute to the same output value. The result of the operation in one cell is therefore handed over to the next unit, where this intermediate result is summed with the next operation of the same colour. The pipelining concept makes it possible, that all the intermediate results for the end result of different time steps exist in the DSP

Table 3.3: Previous structure of the calculation. The calculation for one output value only starts after all input values are available.

Timestep	Input	Calculation
n	x_0	$y_0 = b + w_0 * x_0 + w_1 * x_1 + w_2 * x_2$
$n-1$	x_1	$y_1 = b + w_0 * x_1 + w_1 * x_2 + w_2 * x_3$
$n-2$	x_2	$y_2 = b + w_0 * x_2 + w_1 * x_3 + w_2 * x_4$

Table 3.4: General concept of pipelining over input values. Cells of one colour contribute to the same output value.

Time Step	Input	Calculation		
n	x_0	$y_0 = w_0 * x_0 +$	$w_1 * x_0 +$	$w_2 * x_0 + b$
$n-1$	x_1	$y_1 = w_0 * x_1 +$	$w_1 * x_1 +$	$w_2 * x_1 + b$
$n-2$	x_2	$y_2 = w_0 * x_2 + ...$	$w_1 * x_2 + ...$	$w_2 * x_2 + b$

chain at same time. The example shows, how a kernel size of three with one input stream can be processed with only three multiply-add units.

It is clear how all but one multiplication can be done before the last sample arrives. There may be more than one input stream, for example when the previous layer has more than one feature map. In that case the DSP chain structure exists independently for each input stream and the results of all chains need to be added up before the activation function is applied.

When comparing this concept to the previous approach, one can observe that the use of pipelining is much better integrated with the calculation structure now. One important improvement is also, that even for higher kernel sizes, the latency should not change, since there are still only one multiplication and the sum over all input streams necessary. In the initial model, all the multiplications needed to be done in sequence, as well as the summation. In the improved pipeline model, all multiplications were at least executed in parallel, but the summation over all intermediate results still needed to happen afterwards.

The model using pipelining over input values, as presented here, only needs to sum over the final results per input stream afterwards. The DSP chain structure already does a large part of the summation steps automatically. This means that a slight improvement in either latency or maximum clock frequency can be expected for this model, at least for some networks. The specific architecture of the network needs to be considered for this, since the number of additions depends on it. For small kernel sizes the addition step will probably not be the critical path in the entire neural network design and thus have no influence on the maximum clock frequency. For larger kernel sizes, this can still be achieved by further pipelining the summation stage at the cost of increased latency. The performance of the approach using pipelining over input values is less dependent on the specific architecture, since fewer values need to be added up after the multiplications are finished. This path can still become the critical path depending on the architecture, since the effort depends on the number of input paths, whose intermediate results need to be added. Albeit, the dependence on the architecture is much smaller.

3.4 Manual Assignment of DSPs

The approach shown in table 3.4 can be further improved in terms of maximum clock frequency. This is due to the fact that the DSPs actually requires more than one clock cycle to calculate their output. If the result is expected after a single clock cycle nonetheless, the remaining components will only be able to operate at half the speed or even less. To solve this in a clean and consistent way, the DSPs need to be assigned manually. For this, a intellectual property (IP)-core like library function can be used. Better accessibility is provided by a wrapper for this function [26, pp. 76 sq.]. This allows the inclusion of the DSPs like any other VHDL component. The numbers to be multiplied are assigned as input signals and the result is available as an output signal. This means that the multiplication is no longer trivially implemented by using the inbuilt $*$-operator anymore, but relocated to this generated component.
The Stratix 10 FPGA that will be used for this project has a special DSP architecture, where each DSP contains two multiplication units. The best way to use them here to process two pairs of input values together. The results of both multiplications will then be added up automatically if the correct operating mode is used. Furthermore, DSPs can be chained up for a multiply-add cascade over multiple clock cycles as described in section 3.3. This is an inherent feature of these DSPs and further motivates the use of this pipelining structure. For this, the IP-core and wrapper have a *chainin* and *chainout* port, which can be used for this feature. This means that the results of the necessary multiplications are accumulated automatically without further resource usage or latency over what the multiplications itself already require.
The idea here is to pair two multiplications together and assign them to one DSP. Here only multiplications should be combined, whose output values need to be summed up afterwards. Therefore, a DSP can be used only within one feature map of one layer. One neuron has a number of input values equal to the kernel size multiplied by the number of input channels. To pair up the multiplications, the values of two input channels at the same time step can be grouped. This means that the number of DSPs in one chain will be equal to the kernel size, but two input channels will be processed together.
This approach can minimize the number of necessary DSPs for an even number of input channels. For an odd number, the last chain of DSPs will only be utilized by 50 %. To optimise this further, a separate calculation path is introduced for this case. On this path, two input samples of consecutive time steps, but from the same input channel, are assigned to one DSP in the chain. For an odd kernel size, this will leave one multiplication unit unused. Since all multiplications within that one neuron are assigned, this last unit cannot be used for any other purpose. This means, that without changing the mode of the DSPs, the number of necessary DSPs cannot be optimised further. The expected loss here is also rather small

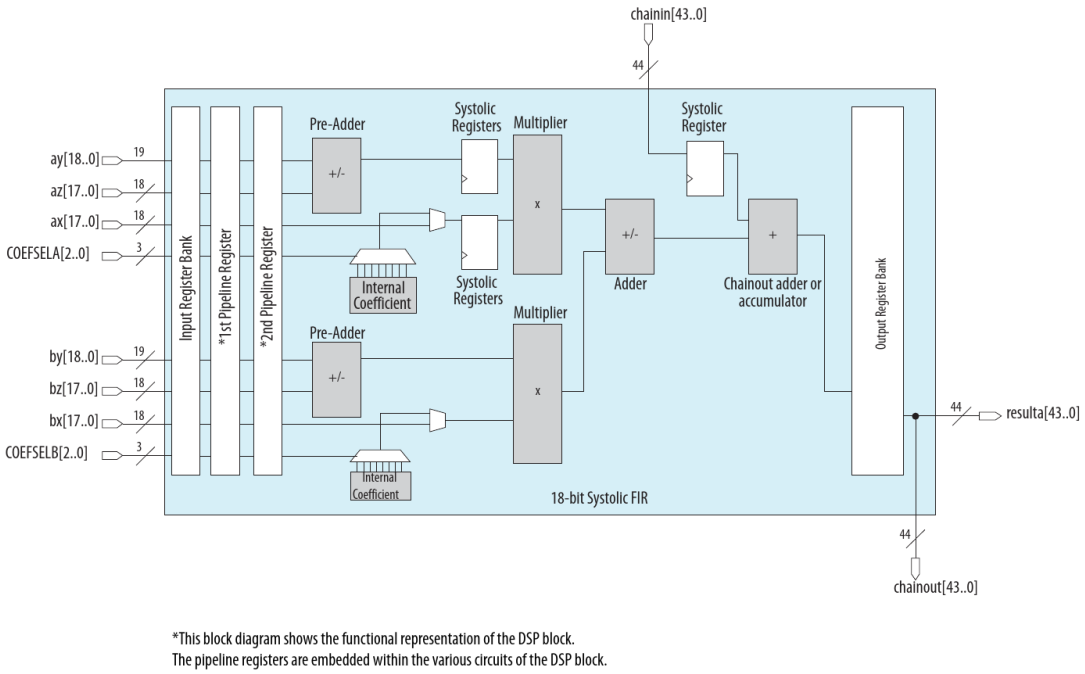


Figure 3.1: Structure of a DSP on the Stratix 10 FPGA in systolic FIR mode [32, p. 28 (Fig. 15)]

and an acceptable trade-off for the performance benefits of using the systolic finite impulse response (FIR) mode with the automatic chain-adding feature. The dedicated calculation path for the last input channel is only necessary for an odd number of input channels to a neuron.

A sketch of the DSP in this mode can be seen in figure 3.1. The input values are the two numbers to be multiplied as well as a third unused value to be added for both multipliers. It can be seen, that the results of this operation are added up before being output. The results are also accumulated onto the value input on the *chainin* port and *chainout* is an alternative output port. Those ports are handled differently from normal input and output ports and can be used to chain the DSPs with less input latency than usual and make use of the free summation step in the DSP.

The results of those two calculation paths need to be properly synchronized before being summed up and the activation function is applied. For this, each path's results are channelled through a delay chain, that is configured with the expected difference in latency of the two paths from the layer architecture. The summation of the results of all the DSP chains is first done over the values from the first calculation path including the bias. In the next step, this intermediate result is accumulated with the result from the separate last calculation path.

Table 3.5 shows the data flow and exact timing through one feature map of a layer

Table 3.5: Data flow inside a layer through DSPs and internal registers

BC	Input from FM			Operation	
n	$x_{0,1}$	$x_{0,2}$	$x_{0,3}$	Output y_{11} usable in next layer	
$n-1$	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	Output y_{11}	
$n-2$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	Activation function	
$n-3$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$		
$n-4$	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	Sum over paths	
$n-5$	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	Sum + Bias	
$n-6$	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	DSP 2	
$n-7$	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$	DSP 2	DSP 3
$n-8$	$x_{8,1}$	$x_{8,2}$	$x_{8,3}$	DSP 1	DSP 3
$n-9$	$x_{9,1}$	$x_{9,2}$	$x_{9,3}$	DSP 1	DSP 3 inp. reg.
$n-10$	$x_{10,1}$	$x_{10,2}$	$x_{10,3}$	DSP 1 inp. reg.	DSP 3 inp. reg.
$n-11$	$x_{11,1}$	$x_{11,2}$	$x_{11,3}$	DSP 1 inp. reg.	
$n-12$	$x_{12,1}$	$x_{12,2}$	$x_{12,3}$		

of the CNN. This layer has a kernel size of two and three input channels, which means that the previous layer had three feature maps. It can be seen how the DSPs are chained up. Notable is the delay of two clock cycles per chained DSP with two initial cycles additional input delay in the first DSP. This initial delay exists as well for the following DSPs, but not for the *chainin* port. Therefore, it does only influence total latency once for the first DSP in chain.

Table 3.6 shows how the assignment of multiplications changes for two more input channels compared to the precious example case.

The total latency L of this implementation measured in clock cycles can be calculated according to

$$L = 2 + \sum_{\text{layer}} \left(11 + (k_i \mod 2) + \begin{cases} k_i - 1 & \text{for } d_i = 1 \\ 0 & \text{for } d_i > 1 \end{cases} \right) \tag{3.1}$$

from the kernel size k_i and the dilation d_i of the respective layer. The different sources that make up the constant delay of 11 clock cycles per layer can be seen in table 3.5. For a dilation of 1, the normal calculation path has an additional delay of kernel size minus one. This results from the fact, that each DSP has an internal delay of two clock cycles, but a new input value is available at each clock cycle. This means that, while the first input value can be sent to the first DSP in the chain immediately after being available, each consecutive sample needs to be delayed by one more cycle. Here the delay of $k-1$ comes from. This phenomenon only occurs in the normal calculation path, since the dedicated last

Table 3.6: Extension of the DSP layout for more input channels (feature maps in previous layer)

BC	Input	Operation		
n	x_0	Output y_{11} usable in next layer		
		...		
$n-4$	x_4	Sum over paths		
$n-5$	x_5	Sum + Bias		
$n-6$	x_6	DSP 2	DSP 4	
$n-7$	x_7	DSP 2	DSP 4	DSP 5
$n-8$	x_8	DSP 1	DSP 3	DSP 5
$n-9$	x_9	DSP 1	DSP 3	
$n-10$	x_{10}			
$n-11$	$x_{11,\mathrm{FM}12345}$			
$n-12$	$x_{12,\mathrm{FM}12345}$			

path can process one input value per clock cycle, because the two multiplication units are used for two consecutive samples instead of ones from the same time step there. The modifier for the case of odd-numbered kernel sizes comes from the last calculation path. Since two consecutive samples are processed in the same DSP, the last DSP in the chain processes only one sample and gets zero as input for the second multiplication unit in this case. The DSP still needs two clock cycles to process the single sample. Therefore, this calculation path has an additional delay of one clock cycle for odd-numbered kernel sizes. The global base latency of two comes from data input and output delay in the simulation.

With the approach outlined here, a maximum clock frequency of 381.68 MHz is achieved, which is significantly higher than the previous performance. This uses the PLAN approximation for the activation function. When using the simpler LUT, the performance increases to 425.35 MHz and the critical path reported by Quartus is actually not in the network calculation itself anymore, but the logic governing the loading process for the weights. Here it turns out, that PLAN needs to be further pipelined to compete with the LUT in terms of maximum clock frequency. The different range checks and shift operations followed by the subtraction in PLAN are actually slower than just using a LUT for the full bit width. Therefore, the LUT is used for the performance and latency optimisations here. At a later time, PLAN can be revisited to reduce the number of necessary ALMs in the design.

With some simplifications in the state machine that determines whether the network is in coefficient reading or calculation mode, and removing some unnecessary reset logic for the weights buffer within a neuron, the maximum clock frequency

reaches 439.56 MHz.
After porting this to the LASP project, the frequency increases to 511.25 MHz. This is most likely due to port virtualisation. The LASP project uses a script to virtualise all input and output ports that are not actually mapped to hardware pins. Previously, the synthesis tool mapped paths to such pins for all the input and output signals for the network. But since this induces some delays which are not coming from the network calculation in itself, it is a good idea to virtualise those and not include them into the performance considerations. Later, the CNN module will be embedded into a larger design on the FPGA and thus not be directly connected to any pins anyway. This situation is therefore more representative of the intended use case than the previous approach. This is also the first time, that the network actually reaches the performance target of 480 MHz, at least for this example network and without significantly lowering the bit widths. This milestone brings this implementation significantly closer to be usable for the real LAr calorimeter readout in the future.

4 Energy Reconstruction Networks

4.1 Network Architecture for Energy Reconstruction

The previous networks were only able to estimate the probability for each BC that there was a signal hit over the 3σ threshold of 240 MeV. This is already a very useful information for the energy reconstruction, but further processing is necessary. The networks under investigation are composed of two subnetworks [7, p. 6]. The architecture can be seen in figure 4.1. The bottom part is a trigger network very similar to the networks used until now. It receives the ADC samples from the detector as an input and outputs an estimate how likely it is that there was a hit over the noise threshold in the current BC. The trigger part consists of two convolutional layers and uses the sigmoid activation function. The first layer has five feature maps and a kernel size of three. The second layer has a kernel size of six and only one feature map.
The second part is an energy reconstruction subnetwork. It makes use of the *concatenate* feature of Keras to combine the ADC samples with the output of the trigger subnetwork. The energy reconstruction consists of one or two convolutional layers with rectified linear units (ReLUs) as activation functions. The sigmoid function was very suitable for the trigger part, since it can only output values in the range of $(0, 1)$, which is very convenient to estimate probabilities and is therefore often used for decision-making networks. However, this is not appropriate for the energy reconstruction. Here a more linear and unconstrained function is more promising.
The energy reconstruction subnetwork in figure 4.1 has two layers, where the first has three feature maps and a kernel size of four. The second layer has only one feature map and a kernel of three. This architecture is referred to as *4-Conv* and used as one of the example architectures in the following performance considerations. The alternative architecture has only one energy reconstruction layer with a kernel size of 21 and one feature map. The trigger subnetwork is still the same as before. This architecture will be referred to as *3-Conv*.

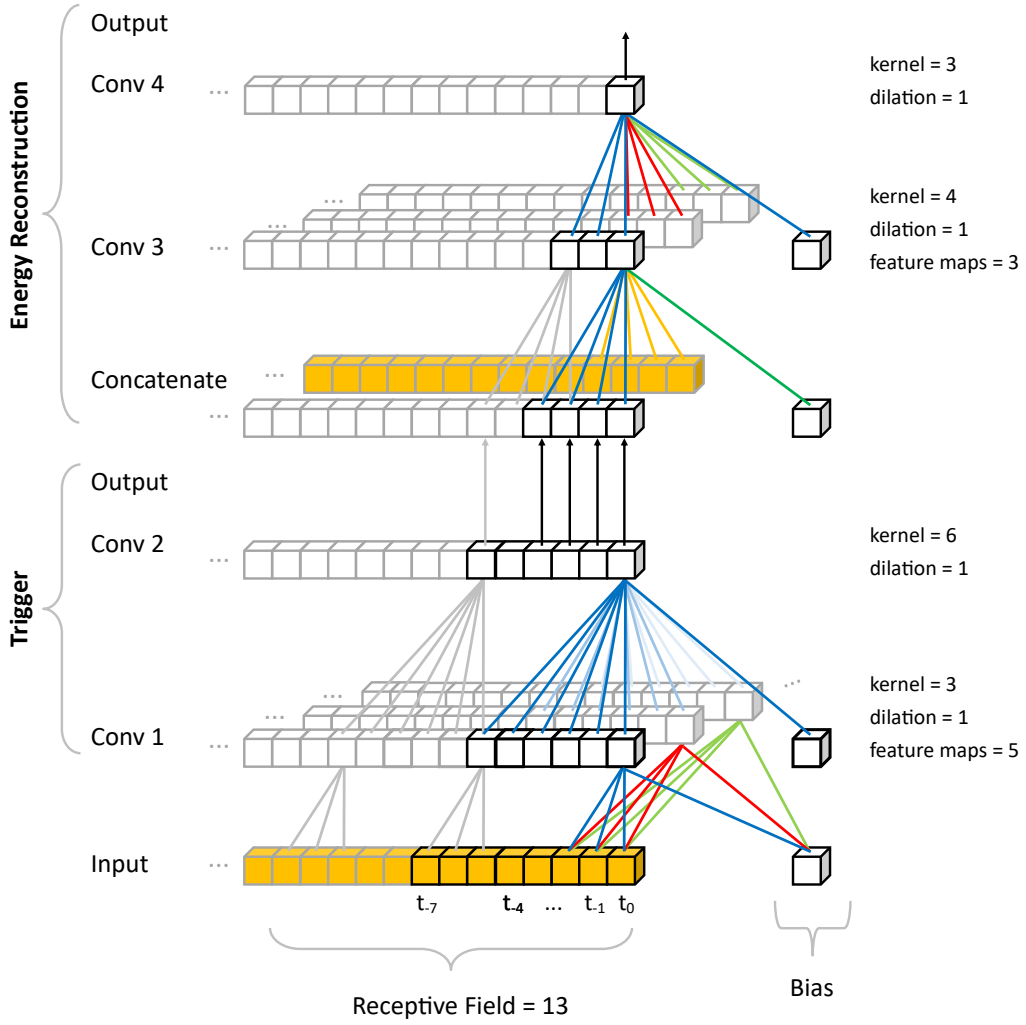


Figure 4.1: Architecture of the combined energy reconstruction network with trigger part as subnetwork in the four layer version *4-Conv* [7, p. 6]

4.2 Performance and Resource Usage of the Energy Reconstruction Networks

The performance estimates of the energy reconstruction networks can be seen in table 4.1. Both network architecture reach the desired frequency of 480 MHz, albeit barely. While the performance degraded somewhat compared to the isolated trigger network, it is still in the same range. On the FPGA, the energy reconstruction network is no different from the trigger one. It is only larger and uses a delay chain to provide the ADC samples together with the trigger network output to the first energy reconstruction layer. The energy reconstruction layers themselves use the ReLU activation function, which is significantly easier to implement on the FPGA, since it is constant for negative input values and linear for positive ones.

The number of DSPs matches the expectations and none are used other than the ones manually assigned for the multiplications. It can be seen in the table, that some multiplication units remain unused due to the reasons presented in section 3.4. For the network architectures used here, this is a loss of about 5 %.

The networks both reach a latency of approximately 60 clock cycles. This translates to a time of 125 ns. The target value here is 125 ns to 150 ns. The networks are therefore fast enough with the energy reconstruction to meet the timing requirements of the trigger system. That is a key result and means that the network implementation is really suitable for use in the real time energy reconstruction. Other solutions using more complex networks have a much worse latency, that is not suitable for the energy reconstruction for the trigger system. The CNNs under evaluation here are smaller and use a less sophisticated architecture, but they fit within the latency constraints of the trigger and therefore enable the use of the improved energy reconstruction by this system. Since a better energy information for the trigger has been a major motivation for the development of these networks, this is a very important milestone.

However, the tests here were done with a single network instance on the FPGA. The results will be negatively impacted when the target number of 43 networks is implemented, since a fuller FPGA is significantly more challenging for the routing

Table 4.1: Performance, resource usage and latency of the energy reconstruction networks

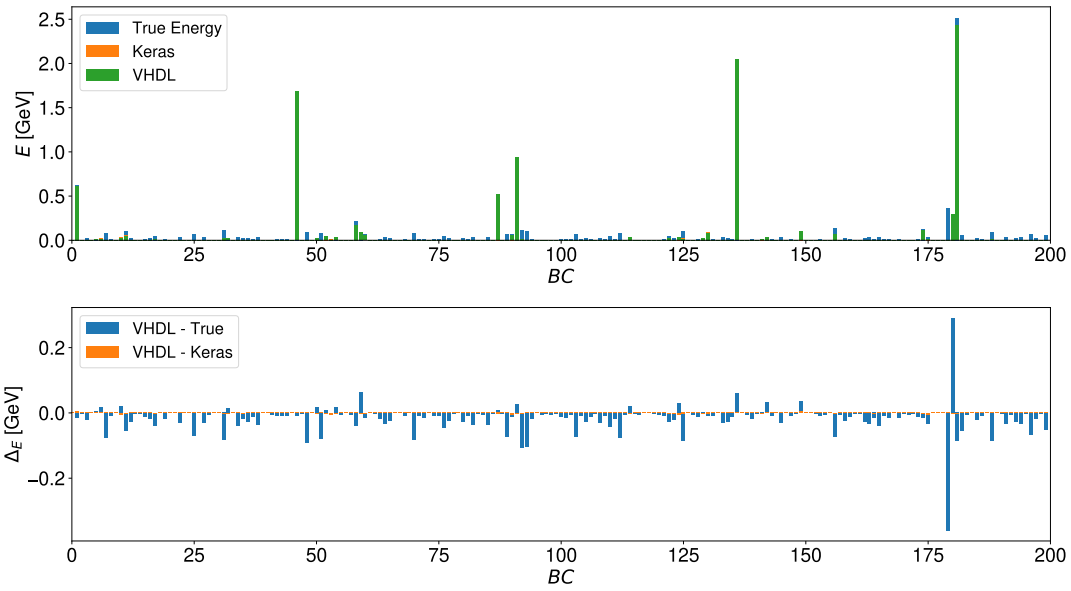
Network	Performance	Resource usage		Multipli-	Latency
	$F_{\max}$ in MHz	ALM	DSP	cations	in clk cycles
3-Conv CNN	493 MHz	5684	46	87	62
4-Conv CNN	480 MHz	5702	42	78	58

tool. This number is the result of the planned 512 input channels per FPGA combined with the 12 times multiplexing. Also, the multiplexing itself is not implemented yet. It is projected, that this will have only a minor influence on the latency, since it can largely be done with the existing signal buffers without large control logic inside the layers. The resource consumption on the FPGA in ALMs translates to 0.6 % of the total available units per instance. The DSP usage is 0.8 % and 0.7 % of the total number for the two network architectures respectively. All these numbers should allow fitting the 43 instances on one FPGA without problems. The hardware usage constraints are therefore met at this stage. Similarly to the latency, the maximum clock frequency may degrade when more network instances are programmed onto one device. More testing and more optimisation will be necessary at that point to conserve the frequency of 480 MHz.

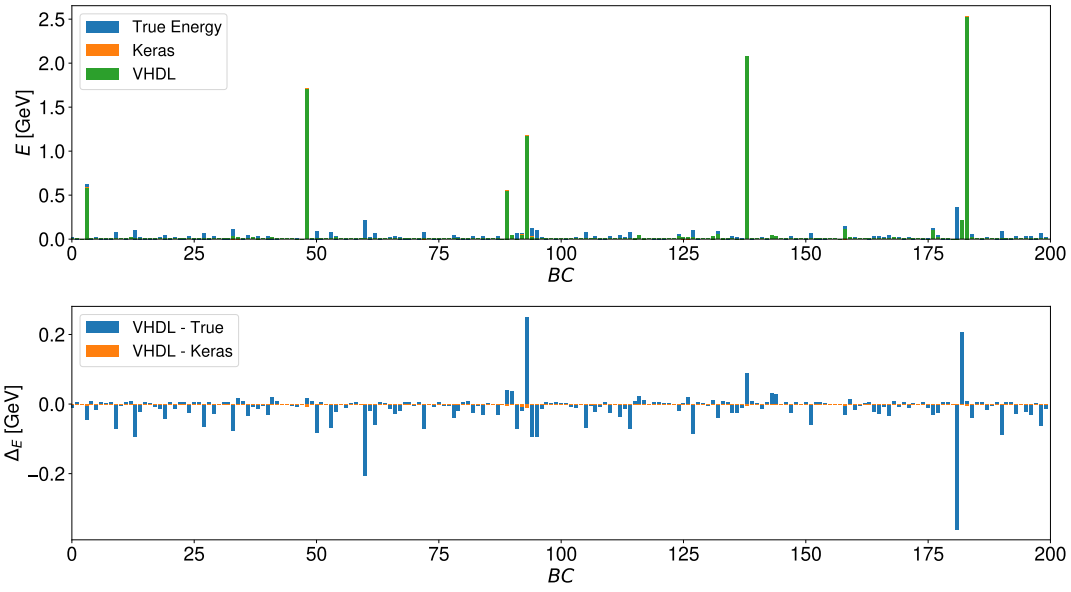
4.3 Comparison of VHDL Implementation Results to Keras Reference

Figure 4.2 shows the energy reconstruction of the combined network for an example sequence with continuous pile-up and an additional signal hit of up to 5 GeV every 45 BCs. This is done for the two energy reconstruction architectures with either three or four layers total as introduced in the previous chapter. The upper plots for each architecture show the true hit energy together with the output from the Keras and VHDL networks. The lower plots show the difference between the results of the VHDL version and the Keras implementation or the true hit energy. In general, a good agreement between the two CNN implementations can be observed. The absolute difference between the two implementations is consistently very low. The agreement shows no obvious variations between pile-up and signal hits and is similar for both network architectures.

To put the deviations into perspective, the plots also show the true hit energy and difference to the true hit energy respectively. There is still a general agreement between the values, but the deviations are significantly larger. It should be noted that the networks are trained to only reconstruct energies above the threshold of 240 MeV. Therefore, the deviations seen between the high energy hits are for the most part actually desirable and show the expected reconstruction behaviour of the trained networks, independent of implementation. This leads to spikes in the deviation up to the energy threshold. While the differences for the higher energy hits are generally lower, they are still visible and at least an order of magnitude over the differences between the network implementations. After BC 175 the highest differences of the reconstructed to the true hit energy can be observed. This is likely the result of the several energy deposits at consecutive BCs. While the neural networks are designed to handle those cases, they still cannot reach the



(a) 3-Conv energy reconstruction network



(b) 4-Conv energy reconstruction network

Figure 4.2: Example output sequence of the energy reconstruction networks compared to the true hit energy, as well as the deviation of VHDL implementation results to Keras and true hit energy respectively.

same accuracy here compared to isolated large energy deposits. Compared to the OF, this is however already an advantage. It was not able to distinguish energy deposits this close together at all.
The comparison between reconstructed and true hit energy shows that the energy reconstruction itself is rather challenging. The reproduction of the suggested network architectures in VHDL instead of Keras will therefore not be the main error source. This is an important finding regarding the network training, since it means that the performance of the Keras networks is a good estimate of the expected performance of the firmware implementation.
To analyse the differences between the implementations further, a statistical evaluation over a longer sequence is necessary. This is done over a sequence with the same mentioned characteristics, but a length of two million input samples. Figure 4.3 shows the distribution of the relative deviation between the results from the simulation of the VHDL model and the Keras reference. Only BCs with a true deposited energy above 240 MeV are considered. BCs where the Keras model predicts an energy of zero are also excluded since this leads to a division by zero in the calculation of the relative deviation.
The distribution peaks at zero for both example networks. This means that the VHDL and Keras models generally agree with each other. There are some deviations, but the curves fall off fast to both sides and do not show significant tails. There is only one notable side-peak for the 4-Conv model between $-3\,\%$ and $-4\,\%$ which hints towards a bias value in that region. The general asymmetry suggests, that the VHDL output tends to be slightly smaller than the Keras results. This can be due to rounding near the zero energy mark or a general slight bias. But the deviations are within the expected range considering the fixed point accuracy.
After the networks were retrained with a modified LHC bunch train structure, larger differences between VHDL and Keras are observed. The respective plot is shown in 4.4. The 3-Conv network develops a kind of plateau towards negative deviations. Where these deviations originate from, could not be determined with certainty. It is possible, that some part of the network now depends on the exact value of a very small intermediate result. In that case, the fixed point nature of the calculation in the VHDL implementation leads to inaccurate results, since those numbers may be rounded to zero or reproduced with very low accuracy, if they are in the range of the least significant bit (LSB). The peak is also shifted slightly towards negative values. A small bias on the results may lead to the observed behaviour, since this would lead to a certain deviation for all energies.
The firmware simulation in Questasim does not offer an easily accessible solution to export intermediate results. A Python implementation that reproduces the fixed point precision could therefore help identify the origin of the differences. However, this is not yet done for the full network. The other advantage of such an independent implementation is, that it can be used to verify the results, especially after changes to the code or to verify some edge cases, where one is not sure

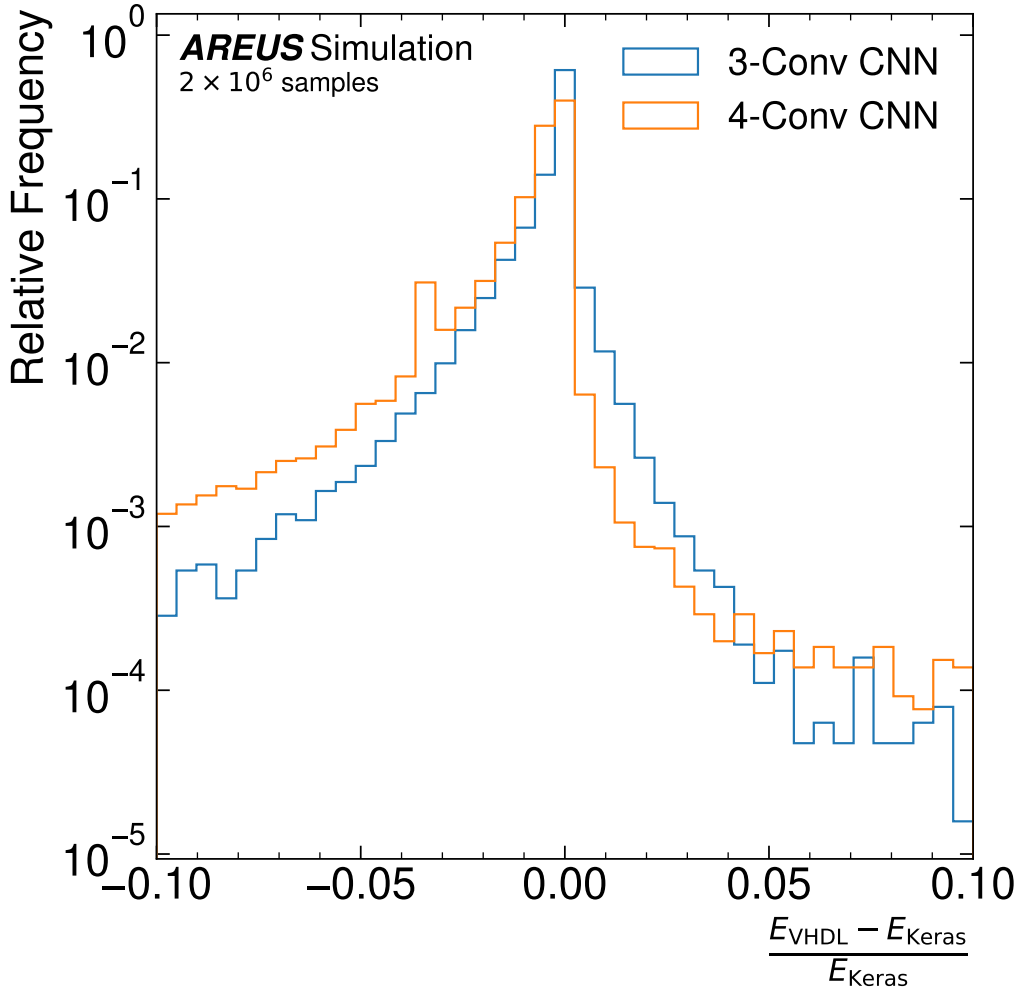


Figure 4.3: Distribution of relative deviation of VHDL implementation simulation results compared to Keras reference for events above the 240 MeV threshold.

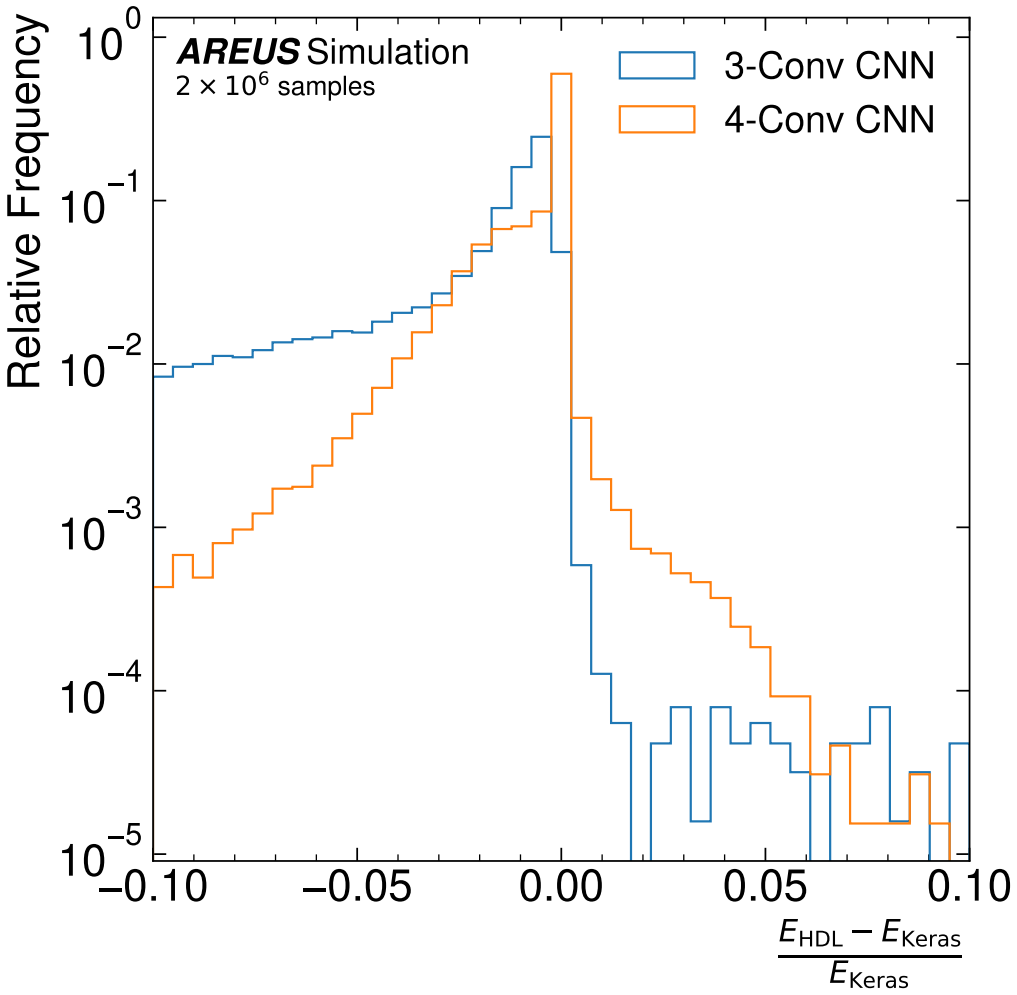
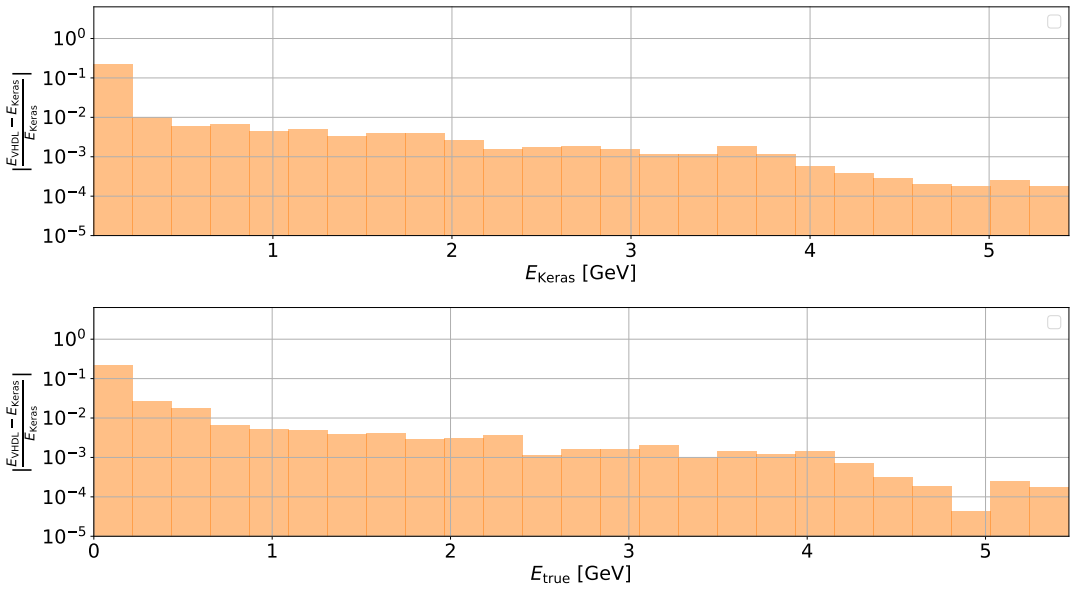
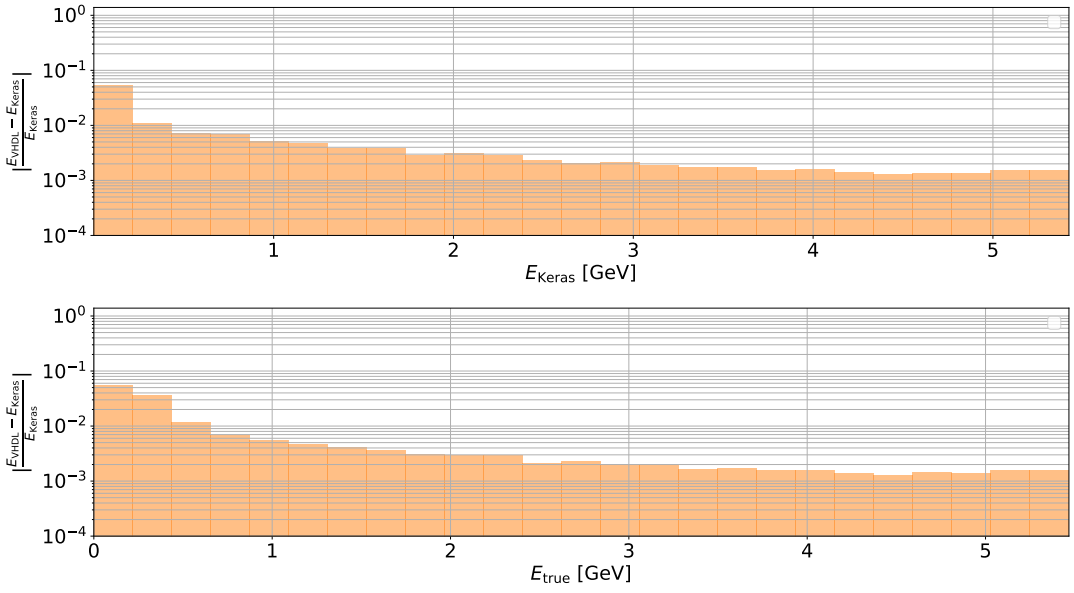


Figure 4.4: Distribution of relative deviation for retrained networks. The much larger deviations hint at numerical stability problems.

whether the deviation is due to a bug or just an artefact of the simplifications in the VHDL model. Later it may be used for automatic tests inside the build pipeline of the continuous integration (CI) system.
Figure 4.5 shows the absolute value of the relative deviations between the VHDL implementation and Keras as a function of the Keras results and the true hit energy for the two example networks with three or four convolutional layers. For this plot, there was no energy cut applied. As expected, it can be seen that the highest deviations appear at low energies, where even small absolute differences are boosted, since relative deviations are being plotted. With higher energies, the deviation generally decreases. The simulations for this were run with the usual 10 bit precision for the decimal part, while the numbers are expressed in GeV. The used events have a maximum energy of 5 GeV. From the numeric inaccuracies alone one can therefore expect differences in the order of magnitude of 1 MeV, which translates to a relative deviation of 2×10^{-4}. The plotted average deviation does not reach that value, but comes relatively close. For the 3-Conv network there is one outlying bin at around 5 GeV, which does not appear in the plot as a function of Keras results. This is due to low statistics in this energy region, as not many events with such a high energy appear in the input sequence. The results for the 4-Conv network look similar, but the plotted average does not pass the 10^{-3} threshold. No special cause for this could be determined. It is assumed that this is just due to the stacking numerical inaccuracies due to the lower bit width and the fixed point notation. For this network, the additional layer may also enhance this effect. This can however not be proven, since the two networks have other differences in the architecture that may lead to different behaviour in the VHDL version. For both networks, the deviations are higher in the low energy region when plotting over the true hit energy than compared to the Keras results. This is an artefact of the low resolution of the energy reconstruction below the 240 MeV threshold of the networks in general.



(a) 3-Conv energy reconstruction network

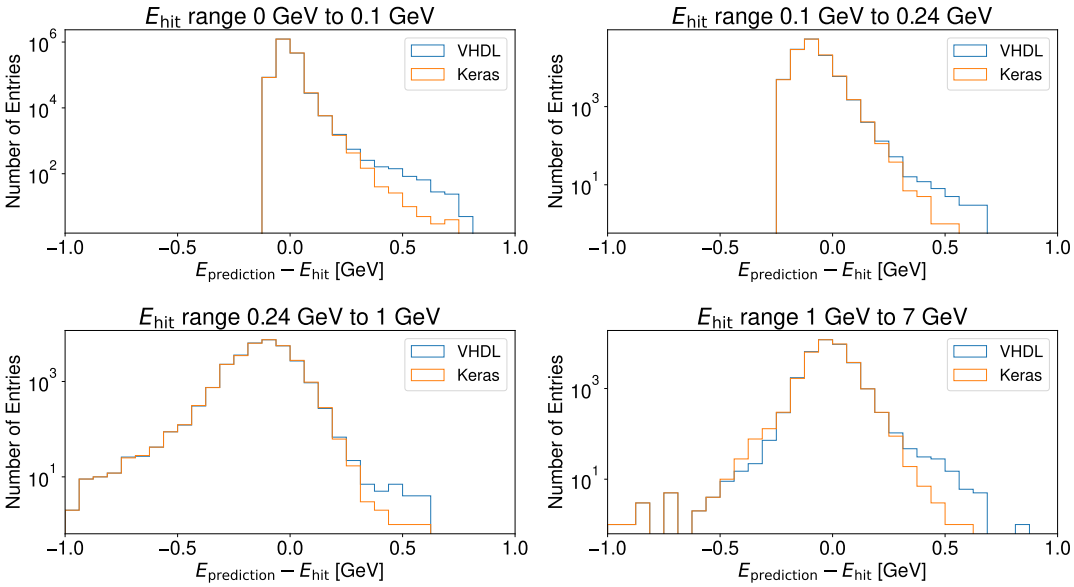


(b) 4-Conv energy reconstruction network

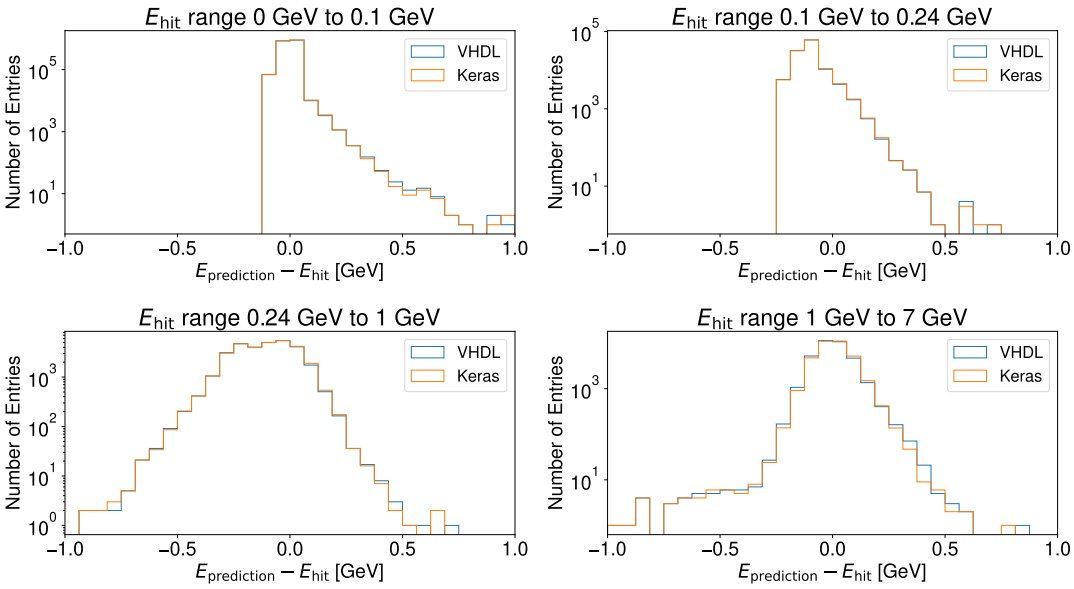
Figure 4.5: Absolute of the relative deviation between VHDL and Keras implementation results as a function of Keras results and true deposited energy in the detector cell

4.4 Quality of the Energy Reconstruction

In the previous section the focus was on the comparison between the VHDL and Keras results with the goal of judging how well one implementation reproduces the results of the other. While this is one way of approaching this comparison, one can also look at the overall quality of the energy reconstruction and see how the differences influence those results.
Figure 4.6 shows the distribution of the difference between the predicted energy and the true hit energy. This is shown for the same two example networks as the previous comparisons. For each network this is further split into four energy regions to see the energy dependence of the reconstruction quality. The plots are split according to the true hit energy. This also explains the cut-off on the left side of the curve, since no lower values can be reached, than a predicted energy of 0 and a true energy of 0.1 GeV. A similar effect can be seen in the second and third subplots, although with decreasing severity.
The first two subplots for the 3-Conv network reveal a slightly stronger positive bias of the prediction in VHDL compared to the Keras version. This can be seen in the region where the blue curve separates from the orange one and stays at higher values for longer before falling below the displayed range as well. This means that more events with a prediction greater than the true hit energy occur in this energy region for the VHDL version of the network. A similar effect can be observed in the other energy regions as well, where there is an increased number of events where the predicted energy is about 0.5 GeV above the true hit energy. The frequency of those events is about two to three orders of magnitude lower than the peak. This means that less than one in 100 events is affected by this additional bias compared to the Keras version.
For the 4-Conv network, no such effect could be observed. There are no obvious regions of larger deviations between the two implementations here and a good reproduction of the results of the Keras network can be expected. Interestingly enough, this network showed a slightly higher deviation between the two implementations in the previous section. So, while the average deviation seems to be higher here, the bias behaviour is better and leads to a more consistent reproduction.
Both networks show a definitive shift towards too small predictions in the second and third energy range, as the peaks are clearly shifted relative to the 0 GeV mark. But since this effect is observed for both implementations, this is a result of the training and not the VHDL model.
In general, the energy reconstruction networks can be seamlessly transferred onto the VHDL platform. The firmware implementation reproduces the results of the software reference model in Keras very well. However, there seem to be some numerical instabilities that degrade the performance for some parameter sets. This remains to be investigated further. With the performance and latency results



(a) 3-Conv energy reconstruction network



(b) 4-Conv energy reconstruction network

Figure 4.6: Performance of the CNNs for energy reconstruction in different energy ranges

shown here, the suggested CNNs become an attractive alternative to the current OF approach. The energy reconstruction using CNNs itself outperforms the OF. It could be shown here, that this still holds true for the firmware implementation with its inherent numerical inaccuracies.

5 Summary

Due to the increase in pile-up on the High Luminosity LHC, new machine learning solutions for the energy reconstruction of the LAr calorimeter are being considered. They show a significant improvement in trigger efficiency and energy resolution over the current OF. Consequently, these algorithms needed to be transferred onto the FPGA platform and the resulting quality evaluated.
During this master's thesis, the pre-existing implementation of CNNs has been extended by the necessary features to represent the new networks under investigation for energy reconstruction. This includes the ability to combine the trigger and energy reconstruction subnetworks through the use of the concatenation feature, as well as specific network properties like dilation. With the help of the created toolbox, the networks can be configured directly and automatically from the network files after the training and the results can be evaluated and verified after the simulation.
The maximum clock frequency of the networks on the FPGA has been increased greatly by transforming the structure of the calculations inside the network from the trivial but functional previous version into a model that properly uses the available resources. The subsequent performance results are promising and meet the requirement of a maximum clock frequency of at least 480 MHz. The use of DSPs has been optimised to utilize all multiplication units as much as possible. By including the DSPs directly instead of letting the synthesis tool do the assignments, a much better integration into the structure of the calculation inside the CNN layers has been achieved. This can be seen in significantly lower numbers of necessary DSPs, as well as greatly improved performance estimates. The latency and performance are also less dependent on the specific network architecture in the new design.
The latency itself is on track to meet the requirements of the trigger as well, since the current energy reconstruction networks have a latency of approximately 125 ns. The architectures of the CNNs themselves were already chosen to be small enough for the required number of network instances to fit onto one FPGA. The synthesis reports show that this can indeed be achieved.
This will need to be proven again in a synthesis of the entire structure to rule out any timing or mapping problems, which cannot be easily anticipated beforehand. This includes the planned multiplexing feature as well as the implementation of 43 parallel network instances on one chip. It is assumed that this will not significantly decrease the performance or increase the latency, but the proof for this can only

be provided by a full implementation.
The accuracy of the VHDL implementation in the energy reconstruction could be shown to be very close to the expected results based on the reference implementation in Keras on the PC. The necessary reductions in accuracy because of the used fixed-point numbers result in a slight decrease in energy resolution, but within the expected magnitude. The simplifications are therefore valid and viable for the use in the detector readout. While some problems with numerical stability remain for specific sets of trained networks, the deviations due to the VHDL implementation are in general much smaller than the inherent inaccuracies of the energy reconstruction networks themselves. The significantly better quality of the neural network solutions compared to the currently used OF for the LAr calorimeter readout are therefore retained in the actual firmware implementation.

Bibliography

[1] G. Aad et al. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Physics Letters B* 716.1 (Sept. 2012), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020.

[2] Georges Aad et al. "Artificial Neural Networks on FPGAs for Real-Time Energy Reconstruction of the ATLAS LAr Calorimeters". 25th International Conference on Computing in High-Energy and Nuclear Physics. Geneva, Feb. 2021. URL: https://cds.cern.ch/record/2752649. Accepted for publication 2021.

[3] H. Amin, K.M. Curtis, and B.R. Hayes-Gill. "Piecewise linear approximation applied to nonlinear function of a neural network". In: *IEE Proceedings - Circuits, Devices and Systems* 144.6 (1997), pp. 313–317. DOI: 10.1049/ip-cds:19971587. URL: https://ieeexplore.ieee.org/document/646812 (visited on 04/20/2020).

[4] ATLAS Collaboration. "Monitoring and data quality assessment of the ATLAS liquid argon calorimeter". In: *JINST* 9.arXiv:1405.3768. CERN-PH-EP-2014-045 (May 2014). Plot available separately: http://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PAPERS/LARG-2013-01/, P07024. 39 p. URL: http://cds.cern.ch/record/1701107 (visited on 05/28/2017).

[5] A. Bazan et al. "The ATLAS liquid argon calorimeter read-out system". In: *IEEE Transactions on Nuclear Science* 53.3 (June 2006), pp. 735–740. DOI: 10.1109/tns.2006.873312. (Visited on 04/01/2021).

[6] I Béjar Alonso et al. *High-Luminosity Large Hadron Collider (HL-LHC): Technical design report*. Ed. by I Béjar Alonso et al. CERN Yellow Reports: Monographs. Geneva: CERN, 2020. DOI: 10.23731/CYRM-2020-0010. URL: https://cds.cern.ch/record/2749422 (visited on 03/19/2021).

[7] Anne-Sophie Berthold et al. "Artificial Neural Networks for the Energy Reconstruction of ATLAS Liquid-Argon Calorimeter Signals". In: DPG Frühjahrstagung. Dortmund, Mar. 17, 2021. URL: https://iktp.tu-dresden.de/IKTP/pub/21/DPG_2021_A_Berthold.pdf (visited on 04/19/2021).

[8] Maximilien Brice. *Aerial View of the CERN taken in 2008*. License: CC-BY-SA-4.0. CERN. July 15, 2008. URL: https://cds.cern.ch/record/1295244 (visited on 08/17/2020).

[9] CERN. *The HL-LHC project*. CERN. Jan. 2021. URL: https://hilumilhc.web.cern.ch/content/hl-lhc-project (visited on 03/19/2021).

[10] S. Chatrchyan et al. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Physics Letters B* 716.1 (Sept. 2012), pp. 30–61. DOI: 10.1016/j.physletb.2012.08.021.

[11] François Chollet et al. *Keras*. 2015. URL: https://keras.io.

[12] W.E. Cleland and E.G. Stern. "Signal processing considerations for liquid ionization calorimeters in a high rate environment". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 338.2 (1994), pp. 467–497. ISSN: 0168-9002. DOI: https://doi.org/10.1016/0168-9002(94)91332-3. URL: https://www.sciencedirect.com/science/article/pii/0168900294913323 (visited on 03/10/2021).

[13] Abdelhak Djouadi. "The anatomy of electroweak symmetry breaking Tome II: The Higgs bosons in the Minimal Supersymmetric Model". In: *Physics Reports* 459.1-6 (May 3, 2005), pp. 1–241. DOI: 10.1016/j.physrep.2007.10.005. URL: https://arxiv.org/abs/hep-ph/0503173v2 (visited on 03/31/2021).

[14] J. Duarte et al. "Fast inference of deep neural networks in FPGAs for particle physics". In: *Journal of Instrumentation* 13.07 (July 2018), P07027–P07027. DOI: 10.1088/1748-0221/13/07/p07027. (Visited on 03/29/2021).

[15] F. Englert and R. Brout. "Broken Symmetry and the Mass of Gauge Vector Mesons". In: *Physical Review Letters* 13.9 (Aug. 1964), pp. 321–323. DOI: 10.1103/physrevlett.13.321.

[16] Lyndon Evans and Philip Bryant. "LHC Machine". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08001. DOI: 10.1088/1748-0221/3/08/s08001.

[17] H. Fritzsch, M. Gell-Mann, and H. Leutwyler. "Advantages of the color octet gluon picture". In: *Physics Letters B* 47.4 (Nov. 1973), pp. 365–368. DOI: 10.1016/0370-2693(73)90625-4.

[18] Nick Fritzsche. "Development of Digital Signal Processing for the ATLAS LAr Calorimeters with Artificial Neural Networks using FPGAs". MA thesis. Institut für Kern- und Teilchenphysik, TU Dresden, Mar. 31, 2020.

[19] Sheldon L. Glashow. "Partial-symmetries of weak interactions". In: *Nuclear Physics* 22.4 (Feb. 1961), pp. 579–588. DOI: 10.1016/0029-5582(61)90469-2.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (visited on 02/02/2020).

[21] David J. Gross and Frank Wilczek. "Ultraviolet Behavior of Non-Abelian Gauge Theories". In: *Physical Review Letters* 30.26 (June 1973), pp. 1343–1346. DOI: 10.1103/physrevlett.30.1343.

[22] G. S. Guralnik, C. R. Hagen, and T. W. B. Kibble. "Global Conservation Laws and Massless Particles". In: *Physical Review Letters* 13.20 (Nov. 1964), pp. 585–587. DOI: 10.1103/physrevlett.13.585.

[23] P. W. Higgs. "Broken symmetries, massless particles and gauge fields". In: *Physics Letters* 12.2 (Sept. 1964), pp. 132–133. DOI: 10.1016/0031-9163(64)91136-9.

[24] Peter W. Higgs. "Broken Symmetries and the Masses of Gauge Bosons". In: *Physical Review Letters* 13.16 (Oct. 1964), pp. 508–509. DOI: 10.1103/physrevlett.13.508.

[25] Peter W. Higgs. "Spontaneous Symmetry Breakdown without Massless Bosons". In: *Physical Review* 145.4 (May 1966), pp. 1156–1163. DOI: 10.1103/physrev.145.1156.

[26] Philipp Horn. "Simulation and Hardware Development of the Liquid-Argon Calorimeters Phase-II Read-out Path of the ATLAS Detector". PhD thesis. TU Dresden, IKTP, Apr. 10, 2020. URL: https://iktp.tu-dresden.de/IKTP/pub/20/Dissertation_Philipp_Horn.pdf (visited on 03/18/2021).

[27] Intel. *Intel Quartus Prime Pro Edition User Guide: Timing Analyzer, Updated for Intel Quartus Prime Design Suite: 19.3*. Sept. 30, 2019. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/archives/ug-qpp-timing-analyzer-18-1.pdf (visited on 04/18/2021).

[28] Intel. *Intel Quartus Prime Software Suite*. URL: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html (visited on 04/18/2021).

[29] Intel. *Intel Stratix 10 Device Datasheet*. May 22, 2020. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10_datasheet.pdf (visited on 05/27/2020).

[30] Intel. *Intel Stratix 10 GX FPGA Development Kit User Guide*. Apr. 2, 2020. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-s10-fpga-devl-kit.pdf (visited on 04/21/2021).

[31] Intel. *Intel Stratix 10 GX/SX Product Table*. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/stratix-10-product-table.pdf (visited on 04/21/2021).

[32] Intel. *Intel Stratix 10 Variable Precision DSP Blocks User Guide*. Apr. 26, 2020. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/ug-s10-dsp.pdf (visited on 07/09/2020).

[33] Intel. *ModelSim-Intel FPGA Edition Software*. URL: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/model-sim.html (visited on 04/21/2021).

[34] K. Kajantie et al. "Is There a Hot Electroweak Phase Transition atmH$\gtrsim$mW?" In: *Physical Review Letters* 77.14 (Sept. 1996), pp. 2887–2890. DOI: 10.1103/physrevlett.77.2887. URL: https://arxiv.org/abs/hep-ph/9605288 (visited on 04/14/2021).

[35] Steve Kilts. *Advanced FPGA design : architecture, implementation, and optimization*. Hoboken, N.J: Wiley IEEE, 2007. ISBN: 9780470127896. URL: https://ieeexplore.ieee.org/servlet/opac?bknumber=5201491 (visited on 03/09/2021).

[36] LHC Experiments Committee. *ATLAS Liquid Argon Calorimeter Phase-II Upgrade: Technical Design Report*. Tech. rep. CERN-LHCC-2017-018. ATLAS-TDR-027. Geneva: CERN, Sept. 2017. URL: https://cds.cern.ch/record/2285582.

[37] LHC Experiments Committee, LHCC. *ATLAS liquid-argon calorimeter: Technical Design Report*. Technical design report. ATLAS. Geneva: CERN, 1996. URL: https://cds.cern.ch/record/331061 (visited on 04/06/2021).

[38] Nico Madysa. "AREUS: A Software Framework for ATLAS Readout Electronics Upgrade Simulation". In: *EPJ Web of Conferences* 214 (2019). Ed. by A. Forti et al., p. 02006. DOI: 10.1051/epjconf/201921402006.

[39] Stephen P. Martin. *A Supersymmetry Primer*. Jan. 27, 2016. URL: https://arxiv.org/abs/hep-ph/9709356v7 (visited on 04/12/2021).

[40] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[41] MissMJ and Cush. *Standard Model of Elementary Particles*. https://creativecommons.org/licenses/by/3.0/deed.en. PBS NOVA, Fermilab, Office of Science, United States Department of Energy, Particle Data Group. Sept. 17, 2019. URL: https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg (visited on 03/29/2021).

[42] Joao Pequenao. *Computer generated image of the ATLAS Liquid Argon*. CERN. Mar. 27, 2008. URL: https://cds.cern.ch/record/1095928 (visited on 03/29/2021).

[43] Joao Pequenao. *Computer generated image of the whole ATLAS detector*. CERN, ATLAS. Feb. 27, 2015. URL: https://cds.cern.ch/images/CERN-GE-0803012-01 (visited on 08/06/2020).

[44] H. David Politzer. "Reliable Perturbative Results for Strong Interactions?" In: *Physical Review Letters* 30.26 (June 1973), pp. 1346–1349. DOI: 10.1103/physrevlett.30.1346.

[45] Abdus Salam. "Weak and electromagnetic interactions". In: *Selected Papers of Abdus Salam*. WORLD SCIENTIFIC, May 1994, pp. 244–254. DOI: 10.1142/9789812795915_0034.

[46] Siemens. *Questa Verification and Simulation*. URL: https://eda.sw.siemens.com/en-US/ic/questa/simulation/ (visited on 04/21/2021).

[47] Steffen Stärz. "Energy Reconstruction and high-speed Data Transmission with FPGAs for the Upgrade of the ATLAS Liquid Argon Calorimeter at LHC". Presented 19 May 2015. Feb. 2015. URL: https://cds.cern.ch/record/2030122.

[48] G. t'Hooft and M. Veltman. "Regularization and renormalization of gauge fields". In: *Nuclear Physics B* 44.1 (July 1972), pp. 189–213. DOI: 10.1016/0550-3213(72)90279-9.

[49] The ALICE Collaboration. "The ALICE experiment at the CERN LHC. A Large Ion Collider Experiment". In: *JINST* 3 (2008). Also published by CERN Geneva in 2010, S08002. 259 p. DOI: 10.1088/1748-0221/3/08/S08002. URL: https://cds.cern.ch/record/1129812.

[50] The ATLAS Collaboration. *Luminosity determination in pp collisions at $\sqrt{s} = 13$ TeV using the ATLAS detector at the LHC*. Tech. rep. ATLAS-CONF-2019-021. Geneva: CERN, June 2019. URL: https://cds.cern.ch/record/2677054.

[51] The ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (Aug. 14, 2008), S08003. DOI: 10.1088/1748-0221/3/08/s08003.

[52] The CMS Collaboration. "The CMS experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08004. DOI: 10.1088/1748-0221/3/08/s08004.

[53] The LHCb Collaboration. "The LHCb Detector at the LHC". In: *JINST* 3.LHCb-DP-2008-001 (2008). Also published by CERN Geneva in 2010, S08005. DOI: 10.1088/1748-0221/3/08/S08005. URL: https://cds.cern.ch/record/1129809.

[54] Steven Weinberg. “A Model of Leptons”. In: *Phys. Rev. Lett.* 19 (21 Nov. 1967), pp. 1264–1266. DOI: 10.1103/PhysRevLett.19.1264. URL: https://link.aps.org/doi/10.1103/PhysRevLett.19.1264.

[55] Steven Weinberg. “Non-Abelian Gauge Theories of the Strong Interactions”. In: *Physical Review Letters* 31.7 (Aug. 1973), pp. 494–497. DOI: 10.1103/physrevlett.31.494.

List of Figures

Glossary

ADC analog-to-digital converter

ALICE A Large Ion Collider Experiment

ALM adaptive logic modules

ANN artificial neural network

API application programming interface

AREUS ATLAS Readout Electronics Upgrade Simulation.

ASIC application-specific integrated circuit

ATLAS proper name (formerly: A Toroidal LHC Apparatus)

BC bunch crossing

CERN European Organization for Nuclear Research

CI continuous integration

CMS Compact Muon Solenoid

CNN convolutional neural network

CPU central processing unit

DAQ data acquisition

DSP digital signal processor

FEB2 Front-End Board 2

FEX Feature Extractor

FIR finite impulse response

FPGA field-programmable gate array

GPU graphics processing unit

HDF5 Hierarchical Data Format 5

HDL hardware description language

HLS4ML High-Level Synthesis for Machine Learning

IP intellectual property

JSON JavaScript Object Notation

LAr liquid argon

LASP Liquid Argon Signal Processor

LHC Large Hadron Collider

LHCb Large Hadron Collider beauty

LSB least significant bit

LUT look-up table

OF Optimal Filter

PLAN piecewise linear approximation of a nonlinear function

PLL phase-locked loop

RAM random-access memory

ReLU rectified linear unit

RMS root mean square

ROC receiver operating characteristic

RTL register-transfer level

SUSY Supersymmetry

VHDL Very High Speed Integrated Circuit Hardware Description Language

Acknowledgements

First I want to thank my supervisor Arno Straessner for enabling this master's thesis and his continued support throughout.
My direct colleagues on this project Nick Fritzsche and Anne-Sophie Berthold were always very supportive and easy to work with. So my special thanks goes to them for the good cooperation that made this work possible. Rainer Hentges was also always ready to give support and help in any situation. Similarly, the detector working group was always very supportive and ensured a nice work environment, both at the university and in remote. Therefore, my thanks of course also goes to Wolfgang Mader, Andreas Glatte, Markus Helbig, Martin Serfling, Tobias Teichmann and Philipp Horn.

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit im Rahmen der Betreuung am Institut für Kern- und Teilchenphysik ohne unzulässige Hilfe Dritter verfasst und alle Quellen als solche gekennzeichnet habe.

Johann Christoph Voigt
Dresden, 28. April 2021