Firmware track search for the P2 experiment

Raoul Jaime Janske-Drost

August 22, 2024
Version: 1.0

Johannes Gutenberg University Mainz
FB08
Institute of Nuclear Physics
Research Group Berger

Bachelor thesis

Firmware track search for the P2 experiment

Raoul Jaime Janske-Drost

1. Reviewer Prof. Dr. Niklaus Berger
Institute of Nuclear Physics
Johannes Gutenberg University Mainz

2. Reviewer Dr. Stefan Endler
Institute of Computer Science - Sports Computer Science
Johannes Gutenberg University Mainz

Supervisors Prof. Dr. Niklaus Berger and Dr.-Ing. Lars Weinstock

August 22, 2024

Raoul Jaime Janske-Drost
Firmware track search for the P2 experiment
Bachelor thesis, August 22, 2024
Reviewers: Prof. Dr. Niklaus Berger and Dr. Stefan Endler
Supervisors: Prof. Dr. Niklaus Berger and Dr.-Ing. Lars Weinstock

Johannes Gutenberg University Mainz
Research Group Berger
Institute of Nuclear Physics
FB08
Staudingerweg 7
55128 Mainz

Abstract

The P2 experiment aims to provide the most precise measurement of the weak mixing angle to date by measuring the parity-violating asymmetry in electron-proton scattering. The objective of this work is to identify the electron tracks from hits detected by tracking detectors. Given the high frequency of electron hits in the order of 0.1 THz, a fast real-time, online filter is implemented prior to feeding the captured data into a computer. FPGA boards filter out other particles with the goal of maximizing the amount of captured electron tracks. This work provides a time and space difference filter for hits in one double layer with a low-level FPGA firmware implementation for the P2 experiment achieving a high throughput rate thanks to an effective buffer mechanism. The algorithm was developed and tested with data from a simulation of the experiment.

Abstract (german)

Das Ziel vom P2 Experiment ist es die bisher präziseste Messung des Weinberg-Winkels zu erreichen durch die messen der paritätsverletzenden Asymmetrie in der Elektron-Proton Streuung. Diese Arbeit identifiziert dabei die Elektronenspuren aus Treffern in den Teilchendetektoren. Da die Frequenz der Elektronentreffer sich in der in der Größenordnung von 0.1THz befindet, wird ein schneller echtzeit, online Filter vor dem Einlesen in einen Computer implementiert. FPGA-Boards sortieren andere Teilchen aus mit dem Ziel die Anzahl der gespeicherten Elektronenspuren zu maximieren. Diese Arbeit präsentiert einen Zeit- und Raumdifferenzfilter in einer low-level FPGA Firmware für das P2 Experiment mit einer hohen Durchsatzrate dank eines effektiven Speicherpuffermechanismus. Der Algorithmus wurde entwickelt und getestet mit Daten aus einer Simulation des Experiments.

Contents

1 Introduction

To understand the theory and implementation we first need to discuss the relevant details of the P2 experiment. We will continue with an explanation of an Field Programmable Gate Array (FPGA) and finish this chapter with an introduction to the problem this work aims to solve.

1.1 P2 Experiment

The objective of the P2 experiment is to enhance the precision of the known weak mixing angle $\sin^2 \theta_W$ to 0.15% by measuring the parity-violating cross-section asymmetry in electron-proton scattering [6][26]. The experiment is planned to collect data for 10,000 h.
The Mainz Energy Recovery Superconducting Accelerator (MESA) will focus a beam of electrons with an energy of 155 MeV on a 60 cm long liquid hydrogen target in a vacuum. The electrons will scatter off the target in a process called elastic electron-proton scattering. It is elastic because no kinetic energy is lost in the system, and the internal states of the involved particles remain unaltered. Some of the scattered particles' energy is converted into bremsstrahlung photons [13]. The lead shielding in the centre will block the created photons from hitting the integrating Cherenkov detector directly [16]. The photons still outnumber the electrons at the initial detector plane by a factor of six orders of magnitude, necessitating the use of thin detector modules. This will reduce the probability of a photon being registered as a hit. Subsequently, the scattered particles are guided by a magnetic field through two double layers of tracking detectors before hitting the integrating Cherenkov detectors at the back [6] whose data is not used in this work.
All described features are shown in figure 1.1. The rotation between the front and back double layers of tracking detectors is visible. This rotational offset of 17° along the beam axis is necessary to account for the helix-shaped particle trajectories in the solenoid's magnetic field. This ensures that the hit coordinates axes from the front and back double layers are aligned. The

four tracking detector rotations are all multiples of 90° to better detect spacial asymmetries and each detector orientation covers about 15° of a full circle.

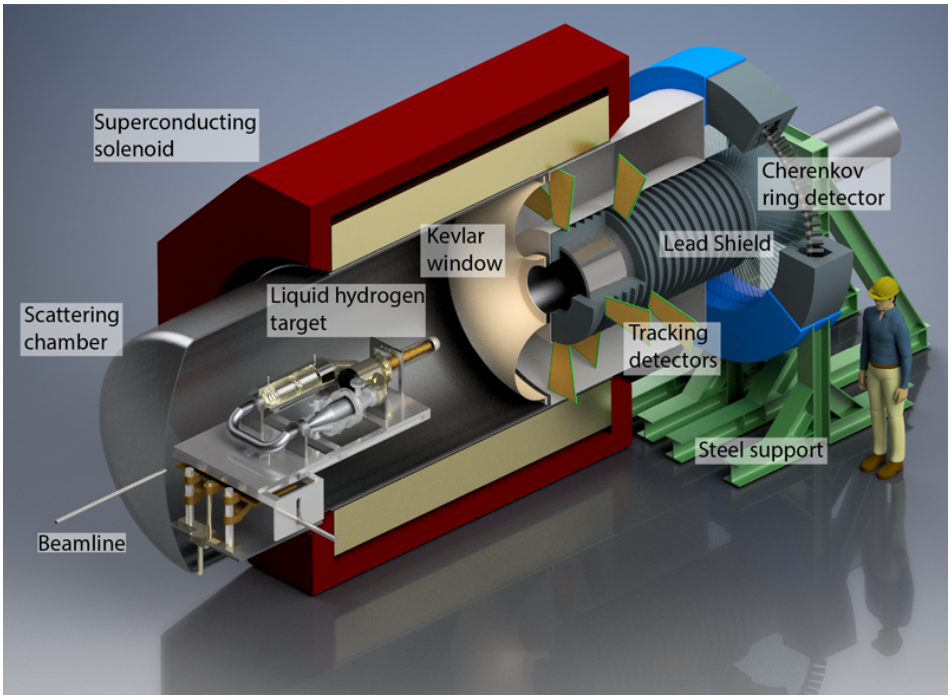


Fig. 1.1.: An overview of the P2 experiment setup.

The two tracking detectors within one double layer are positioned 2 cm apart, with the two double layers 54 cm apart from one another. The electron trajectories are later reconstructed in order to compute the electron momentum transfer in the liquid hydrogen target [26]. In figure 1.2 a two-dimensional projection of the geometry is shown. It should be noted that the figure is not to scale and that certain elements have been omitted for the sake of simplicity. Furthermore, the particle trajectories have been simplified, as they would originate from the entire target length. The diagram provides a good overview of the problem this work seeks to solve. The trajectories follow a complex three-dimensional curve, which is challenging to reconstruct in real time. Working within one double layer reduces the importance of a precise full reconstruction and allows for a simpler space filter.

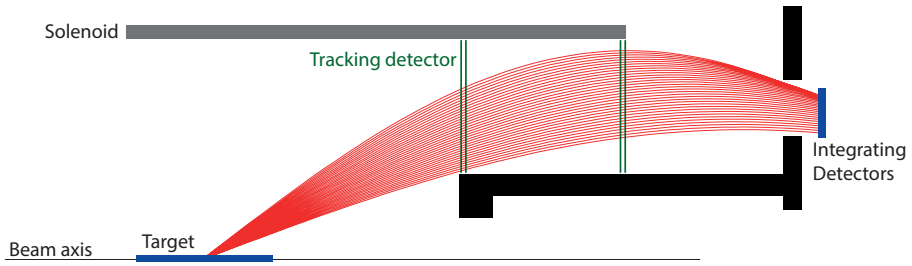


Fig. 1.2.: A schematic view of the particle trajectories and tracking detector geometry of one side.

This works role The objective is to implement an electron reduction algorithm on a FPGA in order to capture the relevant data. The FPGAs are planned to be located between the pixel detectors and computers. The pixel detector chips provide an 11-bit timestamp and 8-bit x and y coordinates per hit via an optical fibre connection shared by seven chips with a multiplexer to the FPGA. The multiplexer selects which chip will be read out. A total of 538 MuPix chips will be connected to each FPGA from both layers. The decoding and full normalisation of the signal into the coordinate space used in this implementation is not within the scope of this work. Similarly, the implementation of the interface to stream the identified electron tracks to the subsequent stage of the processing pipeline is also not a part of it. The hits in the trajectory of an electron are defined as a track and will be detected using a fast filter. The filter needs to work in real-time with an online algorithm. This means the filter can work with an input stream and can process that with sufficient speed to start processing each hit immediately while the experiment runs.

1.2 FPGA

This work heavily relies on the benefits of an FPGA. The abbreviation stands for field programmable gate array. Its abstraction level sits between low-level languages such as C, C++, and ASICs such as a machine learning accelerator [11] or an encoding chip [10]. They are programmed similarly to an Application Specific Integrated Circuit (ASIC) with the main differentiator being the ability to reprogram an FPGA. The cost of an ASIC is what makes FPGAs better in use cases where only a small volume of the firmware is needed. Designing an ASIC takes much more time and the manufacturing process is expensive to build. The FPGA used for this work costs $3,882.00. ASICs are very fast because

of the hardware-level implementation of the used algorithms. An FPGA can achieve similar speeds by implementing logic mostly through Lookup table (LUT) between registers [14].
FPGAs feature many different parts. In this work, Block RAM [25] and Digital Signal Processing (DSP) slices [18] will be used. Block RAM is a physical fast memory with a simple interface directly on the FPGA chip. DSP Slices are special hardware parts on the FPGA optimized for specific purposes like multiplication. They can be inferred by a synthesis tool or explicitly created. Arithmetics can greatly benefit from the speed gained by the specialized hardware found in the DSP Slices [15].
System Logic Cells are a unified measurement of the amount of information that can be stored in the LUTs of the FPGA. LUTs on an FPGA typically have four inputs. They store a truth table mapping each combination of inputs to a desired output. High-end FPGAs can also have six input LUTs. Each LUT can be filled with any function using the inputs given. If they exceed the amount of inputs of one LUT, the intermediate result will get stored in a register or piped into another LUT after the first one. With these methods many LUTs can be combined to calculate one logic function. Flip-flops are used to store signals. A Flip-flop can change its stored value on a clock signal change. This makes them a perfect fit to store a signal between clock cycles and synchronize the whole design. On the FPGA used, eight 6-input LUTs and sixteen Flip-flop registers along with individual carry-ins and multiplexers are combined into a Configurable Logic Block (CLB) [17]. A LUT will compute the given part of a function which then can either be stored in a Flip-flop, a latch or directly connected to a signal outside the CLB. All logic not executed on specialized hardware like DSP slices will be put into CLBs. This is why an FPGA is mostly comprised of CLBs.
To program an FPGA a hardware description language such as Verilog or in this work, VHDL, can be used. They allow the user to write complex logic and some arithmetic without specifying the logical operations on each bit. However, features like overflow avoidance or invalid assignment detection are not covered and need to be accounted for by the user. Special synthesis software such as Vivado from AMD [21] can map, place, route and then optimize the logic on a specific FPGA model. Every step is computationally very complex, and heuristics are used for such large problems. The program first is transformed into a truth table for all registers. Then if needed and available on the selected FPGA some operations such as multiplications are mapped to DSP slices [12], Block RAM memory gets integrated into the circuit [9] and the truth table gets mapped into the specific LUTs while optimizing for space and

speed [8]. After that, all the elements used are connected with pre-defined data buses on the chip.
Block RAM has a pre-defined interface which can be generated using Vivado's IP Generator [21][25]. An address in the unsigned format needs to be given to access the address of the Block RAM port. In order to read or write with the Block RAM an enable signal needs activation. Then a signal given by the enabled Block RAM port will contain the content of the memory address with a delay of less than a clock cycle [25]. If the write signal is also set, the memory address selected will get overwritten with the content of a signal designed for inputting data into the Block RAM. Another comparable available storage option is Double Data Rate 4 Synchronous Dynamic Random-Access Memory (DDR4 RAM). At the cost of a complicated interface [19], it features speeds up to 1200 MHz [2] while Block RAM can only work up to 450 MHz [25]. In this project, only setup parameters and new hits are loaded and statistics are written to Block RAM so speeds faster than the firmware clock are not needed. The reading speed needs to be only as fast as the module 'Compare Hits' to keep up.
The DSP Slices are an important part of this work. They are partially used explicitly like the Block RAM but minor arithmetic operations are left to be inferred by the synthesis tool. They speed up the arithmetic calculations needed for the filters. Their usage lowers the amount of LUTs used and the latency between hit reception and track detection is reduced. The Ultra Scale architecture on which the firmware will be run features many different DSPs including a 27 x 18 multiplier, a multiply-adder and a 27-bit pre-adder. They can be used in a cascading manner [18]. For addition and subtraction, no special DSP Slice is needed since it is computationally much less complex and CLBs already are designed for this.

1.3 Motivation and Problem Statement

It is not feasible to reconstruct all tracks in the P2 experiment. Nevertheless, the rapid processing speed and high throughput of a modern FPGA board permit for the deployment of an online filter to reduce the amount of data to a size that can be processed with a CPU or GPU. The pixel detectors [4] provide a timestamp and an x and y coordinate for every hit detected. The objective is to establish a connection between each hit and an earlier hit in the preceding layer. As electrons emitted from the hydrogen target will exhibit a distinctive

pattern caused by the solenoid magnet field, it is possible to filter out other particles that are not of interest to this experiment. The firmware will be deployed to reduce the rate of data acquisition from the tracking detectors. The false positive rate is allowed to be high, provided that the data output rate does not exceed the reading speed of the next part of the pipeline, which is planned to be a computer. It is of the utmost importance to minimise the occurrence of false negatives in order to avoid the potential for missing true electron hits that could be crucial for later reconstruction. This filter must satisfy a high throughput rate to be able to run when the experiment is in progress. It is clear that computers are unable to meet these requirements. Therefore, it is necessary to consider implementing this at a lower abstraction level. FPGAs offer good processing speed and high parallelisation with time guarantees at a much lower price than an ASIC. With fixed processing times we can always keep up with new inputs no matter the situation. The high parallelization factor enables high throughput. Programming firmware is more difficult and testing is much more limited than with software. This combination calls for a simple yet effective algorithm utilizing the benefits of hardware-level implementation.

2 Related Work

This chapter will discuss similar works and their approaches. Each brings forward ideas that were considered during the development or are an integral part of this work.

2.1 Parameterization-based tracking for the P2 experiment

The paper "Parameterization-based tracking for the P2 experiment" [16] approaches a similar problem for the same experiment with tracking over all four planes with an online algorithm. The concept is track following over all planes with track seeds constructed in the last plane. For each seed a rectangular Region of Interest (ROI) on the detector before it is defined. When a track candidate has been extrapolated through all four planes its quality is compared to others with a χ^2 fit. The ROI is defined by a 3rd order polynomial function consisting of the coordinates x_{pos}, y_{pos}, the size x_{size}, y_{size} and the rotation θ_{rot}. The rotation was fixed at zero for technical reasons and splines were planned to replace the polynomial. Due to the symmetry of the detector placement, all functions depend only on the absolute hit in the last layer and the relative position of the second layer hit to the last layer hit. The coefficients of the polynomials are derived from fitting the function to reference tracks from the simulation. The momentum track parameter was also parametrized with the same two hit positions needed. This method was found to provide a good estimation of the momentum. The number of track candidates with a parameterized ROI scaled much better with the relative beam intensity than a fixed-size ROI leading to less χ^2 fit rejections. The chosen functions also show that the parametrization works with an efficiency between 91% and 97% [16]. Unfortunately, their work was only tested with simulations and no implementation was provided. It is at the writing of this work planned to be implemented after this firmware on a computer running on a GPU or CPU. A differentiating factor to this thesis' algorithm is the reconstruction from all

four tracking detector layers. This allows for more accurate filtering at the cost of more computation. The concept of track following with a ROI is a concept this work also implements.

2.2 Tracklet Approach

Bartz et. al. [5] are able to filter down proton collisions at CERN's LHC accelerator by a factor of 1 million to the interesting examples based on their tracklet approach. They use two transverse momentum correlated hits with a similar origin from the first two layers as seeds to then extrapolate a curve. The trajectory is calculated with estimated parameters. This curve is then matched to new correlated pairs from the other layers. To save time the derivatives are stored in a lookup table. The tracks are evaluated with a χ^2 fit to find more accurate parameters. Multiple permutations are allowed as seeds which warrants a duplicate track removal at the end. The processing is split over 28 sectors with 4 - 8 boards per sector working in parallel. The incoming hits are distributed to the multiplexer of the sector which then uses a round-robin approach for load balancing. The algorithm achieves an efficiency of around 80% with a low false-positive rate [5]. The throughput rate is about 240 MHz. The latency is rather high with 3345.8 ns, but it is within their 4 μs target. It is largely due to data transfers between chips. The data transfer over Block RAM in the processing chain causes too high Block RAM usage on the Virtex-7 FPGA used for testing for a full sector [5]. But more modern FPGAs solve this issue.
This work shows online algorithms requiring a high throughput are possible with track search algorithms. The approach is similar to the other presented work from section 2.1 as it constructs the trajectory with estimated parameters from a starting point with a search window and then selects the best fits with a χ^2 fit.

2.3 MuPix sensor

The source of the data is also an important part of this work. The MuPix is being developed for the Mu3e experiment [7], but will also be used in the P2 experiment [6]. It uses high-voltage monolithic active pixel sensors (HV-MAPS)

to detect hits. The chip provides an 11-bit timestamp alongside an 8-bit x and y coordinate per hit. But the y coordinate only ranges from 0 to 250 since the readout logic blocks part of the chip. All hits are sent to a low-voltage differential signalling link with 1.25Gbit/s. The time resolution sits below 10ns [3] and the efficiency is above 99% with low noise [4]. To achieve these requirements set in the Mu3e experiment [7], the chip went through over 10 generations.
Each FPGA will be linked to about 500 chips and transform the coordinates into a 12-bit representation each corresponding to a range of 0 to 4096.

2.4 Conclusion

Track search on FPGAs is established to be an efficient way of filtering high-frequency data in particle physics experiments. Parametrization and a thorough analysis of patterns found in simulated data sets can lead to high accuracy. Special resources on the FPGA and an efficient yet simple algorithm help maintain very high throughput. The data from the tracking detector chips has sufficient accuracy for time and space parameters.

3 Implementation

This chapter will discuss the more technical details of the implementation as well as the theory behind the filter. The full VHDL code of the implementation can be found in the linked GitLab repository in the appendix A.2. All modules share a common clock down from the 'State Machine' module. From there the current state gets distributed down to the two main modules below it. 'Hit Receiver' and 'Track Sender' both have an enable signal controlled by the 'Check in Region' module. Both are set to active in the 'Detect' state. The 'Hit Receiver' module additionally has a reset signal. The 'Shift Register' module is controlled by 'Compare Hits'. It is in a constant reset state in the 'Idle' and 'Setup' states and active in all other states. The main modules all reset their signals in the 'Idle' state. The other modules get reset by the main module above them shown in figure 3.9.
A hit is stored in a uniform format over all modules handling hits. The format is shown in figure 3.1. The x coordinate is stored in the lower 12 bits, the y coordinate in the next 12 bits, and the timestamp in the following 11 bits. Between 'Hit Receiver' and 'Check in Region' the layer is stored in bit 35. After that point, the layer information is implicitly known through its origin from the stack or shift register. More on that in section 3.3.

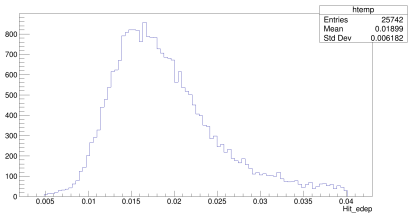
1-bit	11-bit	12-bit	12-bit
layer	timestamp	y	x

Fig. 3.1.: Data format for hits in this firmware.

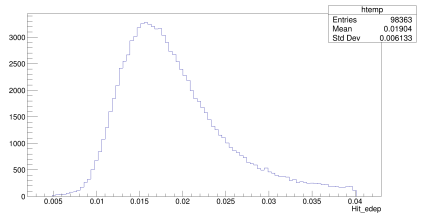
3.1 Methodology

Before diving into the implementation the filter needs to be defined. To find distinguishing features between electrons and other particles a data set from a simulation of the P2 experiment was used. The analyzed data is part of a data set generated by a GEANT4 [1] simulation of the full experimental setup. The tracking detector chips measure the hit coordinates and the energy deposited

by each hit and combine them with a timestamp. We will analyze the latter first. In figure 3.2b we can see a rise in hits between 0.01 MeV and 0.03 MeV which indicates that electrons deposit more energy than other particles in more cases shown in figure 3.2a. Unfortunately, this feature is not significant enough because this difference only exists for about a quarter of all electrons and this ratio is similar to the ratio of electrons to non-electrons in this data set.



(a) Energy deposited at the chip from other particles.



(b) Energy deposited at the chip from eletron.

Fig. 3.2.: Comparison of the energy deposited between electrons and other particles.

For filtering out tracks a time filter is a fast and simple method. As each FPGA is responsible for one double layer the constraints shown in figure 3.3 do apply. In practice, we need to account for some inaccuracies in the sensor.

1. $h1 < h2$
2. $h1 + \Delta_{t_{max}} > h2$
3. $h2 - h1 > \Delta_{t_{min}}$

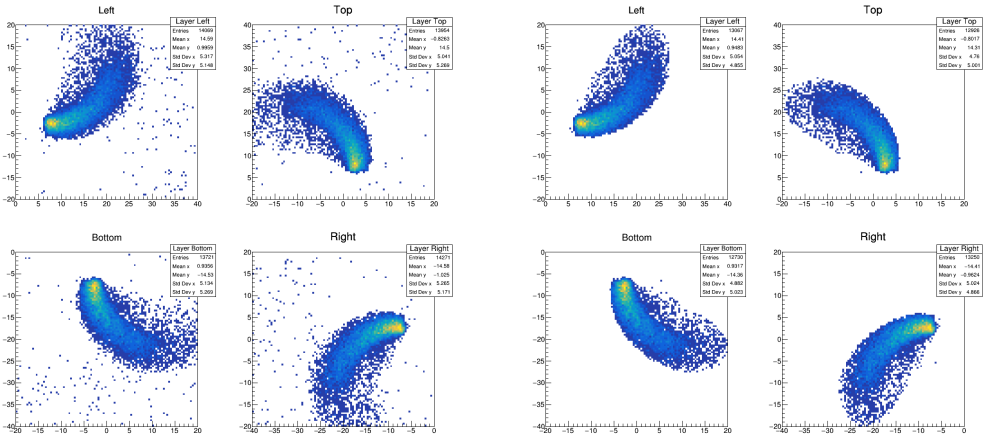
Fig. 3.3.: $h1$: timestamp from front layer, $h2$: timestamp from back layer.

$\Delta_{t_{min}}$ is the minimum timestamp difference between the hit in the first layer and the hit in the second layer. $\Delta_{t_{max}}$ correspondingly the maximum timestamp difference. These limits need to be adjusted depending on the inaccuracies of the system. Some tracks will be missed since the time filter does not account for the rolling timestamps. This is done since the 11-bit range is not big enough to represent all possible timestamps for the entire experiments runtime. The chosen solution is to count up to $2^{11} - 1$ and then start back at 0. These results in some timestamp differences being very far away from zero with one near 2^{11} and one near 0. Implementing extra logic to catch these tracks was chosen not be worth the complexity since very few potential tracks will be affected.

The other main feature is the space filter. Here the hits are being filtered by the difference between the two sets of coordinates. The data shows that over 90% of the electrons points lay inside a scaled bean curve [24]. The points get rotated and offset to avoid heavier computation on the quadrics side. The equation [24] plotted in 3.4b and modified for this purpose is the following:

$$x' = x \cdot \cos(\theta_{orientation}) - y \cdot \sin(\theta_{orientation}) + b_{offset-x}$$
$$y' = x \cdot \sin(\theta_{orientation}) + y \cdot \cos(\theta_{orientation}) + b_{offset-y}$$
$$(x'^2 + y'^2)^2 = (x'^3 - y'^3) \cdot b_{size}$$

The points rotation and offset are added according to the FPGAs tracking detector orientation. Then the new transformed coordinates get put in the bean equation to check if they are inside the shape. In the firmware the parameter b_{size} is set to 20. The rotations are multiples of 90 ° starting at the top and proceeding in a clockwise direction. The $b_{offset-x}$ is 0 for the bottom tracking detector and 22 for the other detectors. The bottom is also the only one with $b_{offset-y}$ set to -22, while the remaining rotations are set to 0.



(a) unfiltered hits from simulation data. **(b)** space filtered hits from simulation data.

Fig. 3.4.: Plots of spacial difference between both hits.

We can see that the electron hits all focus on one point. For the left side that is at (8, -3). From there, a bean-shaped tail develops. On the concave side, there are many single scattered hits. On the convex side, the edge of the shape is clearly delineated. Hits further away from the bean are also present, but their amount is insignificant. The other rotations offer the same features

with different focus points and rotations. This leads to the need for different offsets and rotations in the implementation depending on the orientation of the tracking detectors.
The space filter covers 92.78% of all electron tracks averaged out overall four plots. The maximum deviation between orientations is minimal with 0.15%. Over 99% of the background noise is filtered out by the space filter in the simulation data.

3.2 Compare Hits

The ’Compare Hits’ module takes two hits as inputs and checks if their timestamp difference and coordinates fulfil the time and space restrictions listed in section 3.1.

```
bool compareHits(Hit h1, Hit h2) {
    int timeDelta = h2.timestamp - h1.timestamp;
    double dx = h2.x - h1.x;
    double dy = h2.y - h1.y;
    switch (rotation) {
        case 0:
            dx = -dy;
            dy = dx;
        case 1:
            dx = -dx;
            dy = -dy;
        case 2:
            dx = dx;
            dy = -dy;
        case 3:
            dx = dx;
            dy = dy;
    dx += beanOffsetX;
    dy += beanOffsetY;
    if( timeDeltaMin <= timeDelta && timeDelta <= timeDeltaMax ) {
        if ( pow( pow(dx, 2) + pow(dy, 2), 2) <=
        ( pow(dx, 3) - pow(dy, 3) ) * beanSize )
            return true;
    }
    return false;
}
```

Listing 3.2.1.: A simplified C++ implementation of the 'Compare Hits' module.

Listing 3.2.1 shows the control flow of the module. The time filter is calculated as a signed signal to account for possible negative Δ_t due to the limited time resolution of the MuPix chip [3]. To convert the incoming unsigned timestamps, a zero is prepended before storing them as signed signals. The rotation and offset depends on the orientation of the tracking detectors connected to the FPGA. The if-statements in the firmware get evaluated on already computed signals which are computed before them. The calculations for the space filter are pipelined for the firmware implementation. The individual steps are shown in figure 3.5.

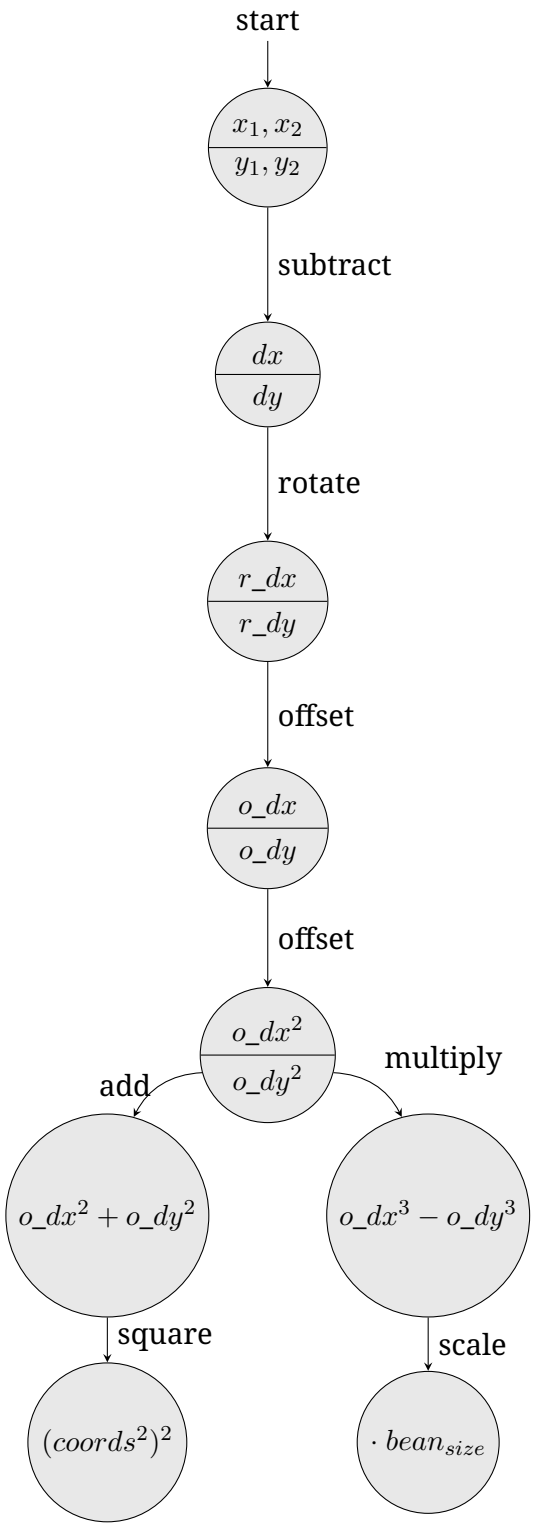


Fig. 3.5.: Visualization of the pipelining in 'Compare Hits'.

The difference for each coordinate first gets computed from the input signals of the module as the other signals depend on it. The following rotation options dictated by the detector plane rotations are all multiples of 90°. This way the result of the multiplication with a rotation matrix is a simple reflection or in the case of the right tracking detectors an identity function. When the rotated dx and dy signals are set, we use the case expression on the orientation of the FPGA's tracking detectors to assign a helper variable to the correct offset of the space filter and add the x and y offset to the corresponding signal. Here they also get converted to a signed data type. This is needed for the cubed difference calculated later which can be negative. A comparison between signed and unsigned is not possible safely with the standard libraries and thus the left side also needs to have a signed datatype. From this point onward all multiplications are done with explicit multipliers. All use DSP multipliers except for scaling of the cubed difference. This is done to balance out the resource usage on the FPGA to maximize the buffer sizes. The squared coordinates are used on both sides and thus the result is stored in a signal used by both sides. Now the two sides of the bean quartic are calculated with a continued split-up pipeline. Between each step shown in figure 3.5 the intermediate result is stored for at least one clock cycle. Some are stored for longer to synchronize both sides at the end. The multiplication DSPs also all have at least the optimum amount of pipeline register stages.
The time filter only needs the difference between the back and front hit. That is stored as a signed signal due to negative results being possible given that the resolution of the tracking detector chip is more than a nanosecond [3]. Theoretically, the timestamp difference should be between 0 ns and 128 ns. In that scenario, an unsigned would suffice. With up to 10 ns error for the timestamp that would open up the correct timestamp window from -10 ns up to 138 ns. These values are used in the implementation.
The pipelining of the calculations is necessary to reduce the number of logic operations between registers. This enables the firmware to operate at higher clock speeds, thereby increasing the throughput. However, this may result in higher latency, which is not a significant concern in the intended use case.
After all calculations are done for the time and space filter the condition is checked. The two time filter parameters $\Delta_{t_{min}}$ and $\Delta_{t_{max}}$ are constants known at compile time and compared to the Δ_t. For this, the timestamp difference signal is not used, because the computation of the space filter has a higher delay. To compensate, the 'Shift Register' module delays the result. Then the space filter compares the two sides of the synced bean quadric. The track gets rejected if the scaled cubed difference is bigger than the left side. If not the

output signal for a found track is set.
The module needs to stay in the 'Detect' state for 17 clock cycles after the last new hit, plus the length of the stack size clock cycles (see following section 3.3), to ensure that the ultimate hit got compared to all other stored hits and all results finished calculating.

3.3 Check in Region

The 'Check in Region' module manages the stack and the shift register used for comparing the hits. The stack is a round-robin buffer. The shift register shifts each entry to the next slot with every cycle. Both buffers are Serial In Parallel Out (SIPO). The module stores the new hits from the front tracking detector in the stack and the new hits from the back tracking detector in the shift register. It then stores hits from each layer long enough for the 'Compare Hits' module to finish computing the result of the comparison. When a pair has been determined to be an electron track the two corresponding entries in the stack and shift register are sent to the 'Track Sender' module, which receives its enable signal.

```
1  void checkInRegion( Hit &newHit ) {
2      if( newHit.layer == 0)
3          stack[stack_pointer] = newHit;
4      else
5          shift_register[0] = newHit;
6      for ( int i = 0; i < stack_size; i++ ) {
7          if ( compareHits( stack[i], shift_register[i]) )
8              trackSender( stack[i], shift_register[i] );
9      }
10 }
```

Listing 3.3.1.: A simplified C++ implementation of the 'Check in Region' module.

In Listing 3.3.1 we can see how the stack and shift register are used to compare each new hit to every item stored. The items in the stack are from the front tracking detector and the items in the shift register are from the back tracking detector in the double layer. This ensures that only tracks with two hits from both planes are considered. An electron can not hit the same plane multiple

times.
The shift register has 17 more slots than the stack to compensate for the delay in the 'Compare Hits' module. The stack does not need that extension due to its circular behaviour. A minimum length of 18 is required to avoid overriding a value that produced a track. This length corresponds to the latency of the 'Compare Hits' module in clock cycles plus one.
As the stack gets filled with values from the front layer, it gets initialized with all bits set to '1'. That prevents any false tracks as the timestamps at the start will be much too small to result in a Δ_t between $\Delta_{t_{min}}$ and $\Delta_{t_{max}}$. 2047 is the unsigned integer limit of the 11-bit timestamp 3.2.

$$h2 - 2047 \ll \Delta_{t_{min}} \tag{3.1}$$

An empty shift register entry is filled with only zeros. If no new hit gets inserted into the shift register an empty entry will be shifted into the first slot. As the front hit's timestamp will be larger than the absolute of the $\Delta_{t_{min}}$ after around 10 ns depending on the $\Delta_{t_{min}}$ value used this will only lead to false positives with a timestamp smaller or equal to the absolute value of $\Delta_{t_{min}}$.

$$0 - h1 < \Delta_{t_{min}} \tag{3.2}$$

The exact behaviour of the stack and shift register is visualized in 3.6. The stored hits are represented by their timestamp which in this visualization, increases by one for every new hit. A newly inserted hit is highlighted in white and a hyphen symbolizes an empty entry. The extension of the shift register to account for the delay in 'Compare Hits' is hidden for simplicity.

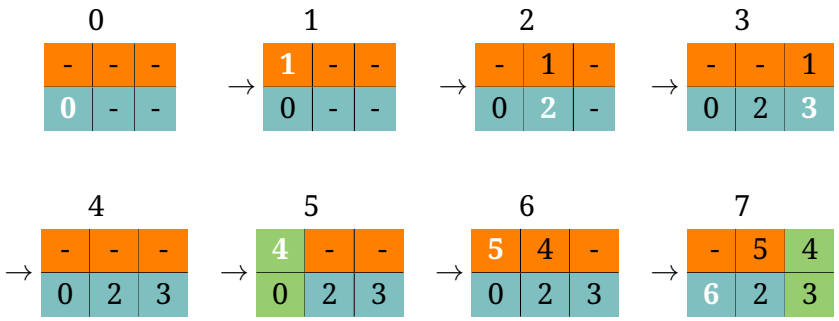


Fig. 3.6.: A simplified example for the shift register (top) and stack (bottom).

In figure 3.6 we start with the stack pointer initialized to 0. The first hit is from the front layer so the hit gets stored in the stack at location 0. The stack pointer is increased by one every time a new hit is stored in the stack. The next hit is from the back layer, it gets stored in the shift register at location 0. In the next clock cycle a front layer hit gets stored into the stack at the next stack location. The hit with timestamp one in the shift register gets shifted by one to the right. The next step demonstrates that two consecutive front-layer hits can be inserted. Another hit from the front layer gets stored in the last stack entry while the shift register once again shifts all entries one to the right. In the next cycle, no new hit arrived and thus the stack remains unchanged. The shift register shifts out the hit number one. After this two consecutive hits from the back layer get inserted into the shift register. The hit pair (4, 0) is detected as a track and will be sent to the 'Track Sender' module. In step 7 a new stack entry is inserted overwriting the first entry. The stack pointer addition features a modulo operation after the increase to ensure it stays within the stack length. Another hit pair (4, 3) is signalled to be a track. One hit can be part of multiple track candidates as seen in the last step.
Fig. 3.6 shows that every hit gets compared to a maximum of others from the other layer. One hit can be part of multiple tracks, but no duplicate tracks will be created. To avoid duplicates the shift register shifts all entries with every clock cycle so that at no point 'Compare Hits' will get the same combination of hits again barring two hits in the same layer with identical coordinates and timestamps. The hits can be inserted into the stack or shift register in every cycle. Repeatedly inserting into one of them is also possible. The example shows this for the stack in cycles 2 to 3 and for the shift register in cycles 5 to 6. The shift register shifts each entry one to the right in each cycle.
When the 'Hit Receiver' module reads a new hit from Random Access Memory (RAM) a signal is set to notify the 'Check in Region' module to update its buffers. If no new hits are read, the shift register insert an empty hit in its first slot and the stack remains unchanged. This is to cover edge cases at the start and end of the detection phase. Other modules do not need a signal for this. The 'Track Sender' module gets activated by a found track in 'Compare Hits' which compares all entries with the same index from two the buffers controlled by 'Check in Region'.
When this module changes into 'Detect' the 'Hit Receiver' module gets enabled. Every pair of hits between the stack and shift register has a 'Compare Hits' and 'Track Sender' module to operate in parallel. This way we can utilize the massive parallelism of the FPGA chip. To account for the delay of 'Compare Hits' the 'Track Sender' module takes the shift register entries 17 positions

after the index pointed to by the hit vector. This necessitates that the shift register be 17 entries longer. The stack entry will still be at the same location. The stack pointer increases by one every time a new entry is stored in the stack. If it exceeds the length it starts at 0 again thanks to a modulo operation. In the 'Collect' state the processed hits counter is written to Block RAM. The counter is 64-bit wide to prevent overflow and split into a lower and upper half to be able to fit it into the 36-bit Block RAM.

3.4 State Machine

The 'State Machine' top module manages the current state of the firmware. In the 'Idle' state all signals get reset. In the 'Setup' state the 'Compare Hits' module loads the orientation of the FPGA from Block RAM. In the 'Detect' state the other two main modules 'Check in Region' and 'Compare Hits' get activated. In the 'Collect' state the processed hits counter gets written to Block RAM by the 'Check in Region' module. As illustrated in figure 3.7, the transition to the next state is entirely sequential and initiated by an input of '1'. This signal will be controlled from outside this firmware. The signal is evaluated at each cycle cycle and input '0' will result in no state change. It is designed to be controlled by a signal from an outside source with more knowledge of the current experiment's state.

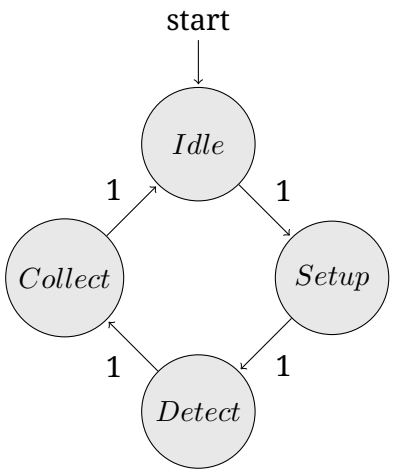


Fig. 3.7.: A diagram of the state machine. With input '0' the state will remain the same.

3.5 Other Modules

The implementation also uses some helper modules for smaller tasks. The ’Hit Receiver’ module reads the hits over Block RAM and projects the coordinates to the positive 12-bit representation. The ’Track Sender’ module is a placeholder module that sends the detected tracks to the next element of the processing chain. The ’Shift Register’ module is a simple buffer needed for the ’Compare Hit’ module.

3.5.1 Hit Receiver

This module sits between the tracking chips and the ’Check in Region’ module. It reads the new hits from Block RAM with their 11-bit timestamp and 8-bit x, y coordinate. The layer information is derived from the Block RAM address. Hits from addresses one to fifteen are from the front layer and hits from addresses 16 to 31 are from the back layer within the double layer. The x, y coordinate is then normalized to be within 48 and 4048. The 48 margin left at the bottom and top of the 12-bit representation is for the bean offset added in the ’Compare Hits’ module. The timestamp information does not get altered. Depending on the address of the hit the layer information is inferred. The hits get read from Block RAMaddresses 1 to 32. This improves the throughput and opens up the option of parallelizing the process if more throughput is needed.

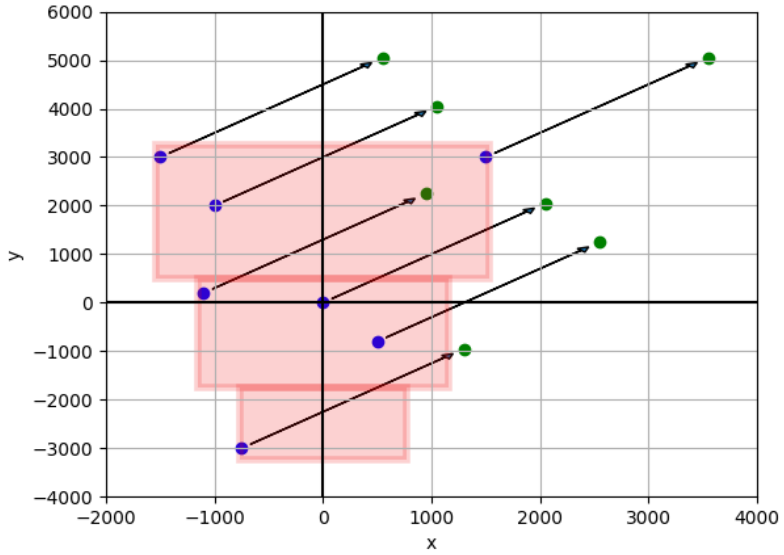


Fig. 3.8.: Coordinate normalization from MuPix coordinate space to positive 12-bit integers as implemented without scaling.

In 3.8 the coordinate space from the tracking detector is at the time of writing planned to be a grid of 6500 by 3072. The tracking detector is comprised of three rectangular parts. The top part is planned to be 3072 by 2750, the middle part 2304 by 2250 and the bottom part 1536 by 1500. This arrangement is chosen to simulate a trapezoidal shape the electron trajectories follow in this intersection. This shape is highlighted in the graph in red. From there the points get shifted to the positive coordinate space by adding 2048 on both axes with enough margin for a maximum offset of 48 in each direction. This information is not yet final and thus the scaling of the coordinates to 12-bit coordinate space needs to be added later.

3.5.2 Track Sender

The 'Track Sender' module is a placeholder module which will not be part of this work. The interface to the next element of the processing chain is unknown at the time of writing. Only the inputs of the module are set. In order to obtain a minimum amount of logic for benchmarking purposes, a track signal containing all incoming information has been placed in this module. The module receives the timestamp and the x, y coordinate from 'Check in Region'. The layer information can be inferred from the signals. The

'h1' signals are from the front layer and the 'h2' signals from the back layer. It has an enable signal controlled by the 'Check in Region' module. Every hit pair between the stack and shift register from 'Check in Region' corresponds to one 'Track Sender' module. Because of this multiple tracks can be found in the same clock cycle. To prevent sending two tracks at the same time we will advise using a link faster than the clock speed of this implementation. That way all track candidates get sent and only a small buffer is needed to momentarily store a track candidate for a few cycles.

3.5.3 Shift Register

The 'Shift Register' facilitates the 'Compare Hits' module to synchronize the time and space filter. It is a synchronous Serial In Serial Out (SISO) First In First Out (FIFO) memory buffer. It is used to store the timestamp difference from the incoming hit pairs and stores them for 14 clock cycles to offset the delay in the calculation of the space filter. This way the space and time filter can be evaluated at the same time ensuring correctness. It is initialized with all zeros except for the highest bit to ensure it does not trigger any false positives. This bit is the highest bit of the timestamp of the hit. This way it will be outside the time filters $\Delta_{t_{min}}$ and $\Delta_{t_{max}}$ when the firmware starts operating and the initial values are used.

3.6 Conclusion

The full FPGA implementation uses the state machine as the top module with the 'Check in Region' module below it. The buffers in 'Check in Region' are only active in the 'Detect' state. Below that module the 'Compare Hits' module checks if the two hits given by 'Check in Region' form a potential track. The 'Check in Region' module also manages the 'Hit Receiver' module to store new hits in the stack or shift register. If the module 'Compare Hits' detects a track it signals the 'Check in Region' module to send that hit pair. The 'Track Sender' module gets enabled and the track is sent to the next part in the processing chain. The hierarchy of the modules is depicted in figure 3.9.

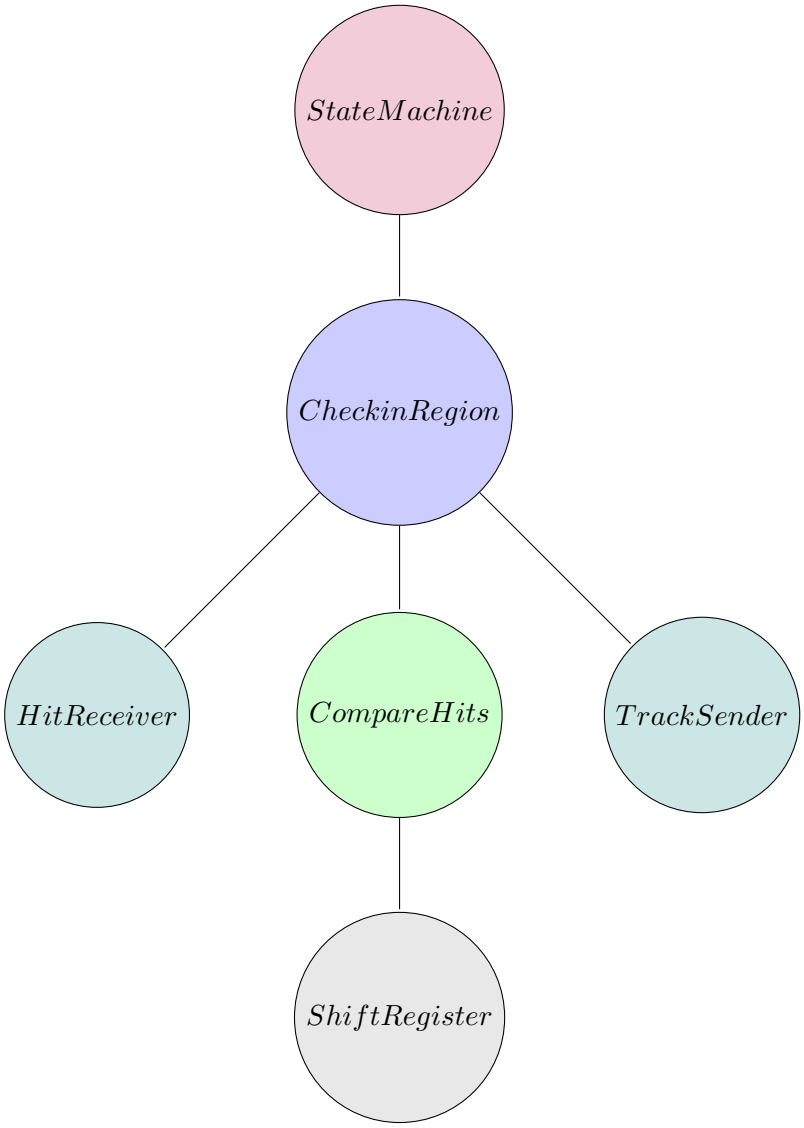


Fig. 3.9.: A diagram of the module hierarchy.

4 Benchmarks

The reason for choosing an FPGA implementation was the speed and throughput. This chapter will dive into all the performance benchmarks.
All simulations, synthetizations and implementations were executed for the AMD Kintex UltraScale FPGA KCU105 [2] in Vivado 2023.2 [21]. The specifications of the board and chip are shown in table 4.1.

	FPGA
Name	XCKU040-2FFVA1156E
System Logic Cell	530,250
DSP Slices	1,920
Block RAM	21.1 Mb
I/O Pins	520

(a) Specifications of the FPGA chip used.

	Board
Name	AMD Kintex UltraScale FPGA KCU105 Evaluation Kit
Memory	2GB 1200MHz DDR4 64MB (512Mb) Quad SPI Flash 8Kb IIC EEPROM Micro SD Card Slot
Communication	Gigabit Ethernet GMII, RGMII, SGMII 2x SFP / SFP+ cage GTX port (TX, RX) with four SMA connectors UART To USB Bridge PCI Express x8 edge connector
Power	12V wall adapter or ATX

(b) Specifications of the Evaluation Board used.

Fig. 4.1.: Overview of FPGA and Evaluation Board used.

4.1 Correctness

Every module has undergone extensive testing through the utilization of the Vivado [21] simulation tool. Edge cases were included in all tests. Only valid inputs from the tracking detector chips [4] were considered.

4.1.1 Check in Region

All states and many hit combinations are tested to ensure the correctness of this module. The state machine's state signal controls the state of this module. It is ensured by checking the signal states to be reset after the 'Idle' state is set. It is analysed for the 'Setup' state if the 'Compare Hits' module reads the orientation parameter from Block RAM correctly. To initialize the Block RAM in the simulation a coefficient (COE) file is used [25]. In this, the orientation specified is right. This is done for ease of testing as other orientations would lead to a reflection of the coordinate differences on the x and or y-axis. Then in the 'Detect' state, the new hit signal is set. A base track candidate with a coordinate offset of (-20, -10) and a timestamp difference of one is correctly identified as well as one other track. Six more space filter rejections are tested. Among them are edge cases with extreme x and y coordinate differences to ensure no overflow occurs. The 'Compare Hit' module is tested further with more edge cases in its testbench. For most of the 'Detect' state, a new hit is fed into the module in every clock cycle to test if the module can handle the frequency. At the end, the new hit signal is disabled to ensure new track candidates already stored in the buffers are detected without it. The processed hit count is also verified to be written to Block RAM. At the end of the simulation, the module is reset through a transition back to the 'Idle' state. Another test with two hits stored in Block RAM initialized with a coefficient file is conducted. The track is successfully read, detected and sent.

4.1.2 Compare Hits

Like in the 'Check in Region' testbench, this module is tested in all states and with many hits. The reset through the 'Idle' state is tested by probing the track output signal. This signals that the hit pair combination is a track candidate. It is not triggered in or immediately after the 'Idle' state. The output has a

delay of 17 clock cycles. Because of this, after each hit pair the testbench waits for more than 17 clock cycles to ensure the result has been computed. Then the 'Setup' state is entered to let the module read its orientation. Like in the 'Check in Region' testbench this is read from a coefficient file and set to right. This is to simplify the writing of the tests and mitigate human errors. Other orientations are tested as well. Different Δ_t are tested. The minimum is tested by providing hits with a Δ_t of the exact $\Delta_{t_{min}}$ and just under it. The same procedure is done for $\Delta_{t_{max}}$. The maximum coordinate offset is tested for x and y to ensure no overflow errors occur. Valid track candidates are found with negative and positive Δ_t. The track candidates at the edge of the space filter as well as in the middle are included. All tracks are correctly identified.

4.1.3 Other Modules

The testbenches of the other modules are much smaller as their functionality is much simpler.
The 'State Machine' modules reset and state transition is tested. The transition signal is activated for one clock cycle, then the state is tested. The states are tested in order from 'Idle' back to 'Idle'. All tests pass for this module.
The reset behaviour of the 'Hit Receiver' module is verified to be correct. Since it reads new hits from Block RAM the coefficient file [25] is expanded to include a hit from layer one with x, y and timestamp set to 1. The receiver module correctly adds the normalization value of 2048 in both coordinate axes to the value read from Block RAM. It also correctly infers the layer from the Block RAM address. At the end the module is verified to reset when the read enable signal is disabled.
The 'Shift Register' testbench validates correct reset and shifting behaviour. A testbench for the 'Track Sender' module is present, but like the module itself, it is a placeholder.

4.1.4 Timing

A timing analysis was conducted to validate the algorithm's functionality on the FPGA. Timing constraints were constructed for this with the help of the Vivado timing constraint wizard [20] and the constraint guide [22]. The top module's input and output signals were constrained with a setup delay of up to

3 ns and a hold delay of up to 1 ns. This delay is deliberately chosen to be large as the signals from the 'State Machine' module will not change frequently apart from the clock which does not need a setup or hold constraint. Timing violations on that control level would not lead to any incorrect results from the other modules. A missed or delayed state transition can be detected from outside as the current state is communicated and this operation is not time-critical. Nonetheless, the signal for the current state leads to a negative slack of up to -6.7 ns for the lower state output pin of the 'State Machine' module. All paths apart from the output pins with the current state stay within the 4 ns clock period. The worst hold slack is 0.029 ns and well within the timing constraints.

4.1.5 Accuracy

The priority of this firmware was a low false negative rate to avoid losing true electron tracks. This led to a coarse filter approach for both the time and space filters. The actual firmware was not tested for its accuracy due to the difficulties connecting the simulation or hardware with simulated input data. As mentioned in the introduction 92.78% got found with the space filter. The false positive rate is 4.23% and the false negative rate is just 0.91%. These numbers are derived from data created in the GEANT4 [1] simulation of the complete P2 experiment. The signal and background rates have been combined for this. The time filter could not be simulated with the data set and is thus derived from a theoretical point of view. These accuracy numbers achieve the targets for this work.

4.2 Throughput

The throughput is the most important benchmark of this work owing to the very high rate of particle hits detected by the tracking detectors. This constant high data rate prohibits us from buffering the hits due to no storage fast enough being able to store that amount of data. The firmware runs with a clock rate of 250 MHz. In every clock cycle the firmware can read in one new hit and process it. A pipelining delay from the different modules introduces some latency until the first output is observed. From then on the throughput is one hit per clock cycle or a throughput rate of 250MHz. This

enables the firmware to accomplish its real-time online filtering target set out at the start. Faster clock speeds could be possible, but with this version of the implementation, the timing violations in the top module and in the 'Check in Region' module are already close to the limit.

4.3 Other Metrics

This firmware is placed between the tracking detectors and most likely a computer. We process each hit, so no buffering of hits before processing is needed. Since the time the firmware processes a hit does not affect any other parts of the experiment the importance of latency is low. Yet an analysis of the latencies introduced by each module gives us an important insight into the computational complexity of each part. The most important latency is between a hit read from Block RAM in the 'Hit Receiver' module and the 'Track Sender' module. This path has a variable latency due to the shift register in 'Check in Region'. The 'Compare Hits' module takes up 17 clock cycles or 68 ns. This is the largest static latency introduced since the module is responsible for the computation of the filters. The 'Hit Receiver' module takes three clock cycles or 12 ns from reading the signal to sending it to the 'Check in Region' module. This is due to the coordinate normalization. The module 'Check in Region' introduces a latency between zero and the stack and shift register length. This can lead to a very noticeable delay, but it is necessary. The variable nature is not a concern for this application. The 'State Machine' module is not a part of the critical path for track detection. Nonetheless, the latency is important for controlling the firmware. The latency here is just one clock cycle or 4 ns to transition to the next state. The latency of the 'Track Sender' module is irrelevant as it is just a placeholder.
For the following metrics, a stack size of 300 was chosen. This is near the maximum possible while using DSP slices and LUTs. The remaining space is left for the full implementation with a complete 'Track Sender', 'Hit Receiver' scaling as well as miscellaneous logic that could be needed. The data was gathered from the reports of a complete implementation in Vivado [21]. The firmware uses 73.33% (177,748) of the LUTs of the FPGA. This allows for other firmware or more logic in this firmware to run concurrently on the same FPGA. 25.08% (150.5) of the available Block RAM tiles are used. Nearly all are used in the 'Compare Hit' modules since that is implemented with a high parallelization factor. 78.13% (1500) of the DSP slices are used. Only 0.96% (5) of the input and

output (IO) pins are used. The used pins stem from the 'State Machine' module.

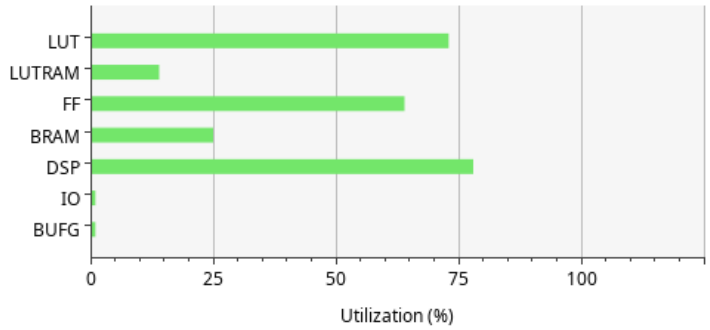


Fig. 4.2.: FPGA resource utilization

All following power-related benchmarks are estimations with a medium confidence level from the Vivado power report [21]. The default settings with an ambient temperature of 50 °C are used [23]. The firmware static power draw is 1.101 W with a dynamic or design power draw of 13.614 W. Most of the dynamic power draw is from signals with 42% followed by logic with 31%. This adds up to a total on-chip power of 14.715 W. The thermal margin is 27.3 °C with 16.9 W.
The exact amounts of these resources may differ in the final implementation of the P2 experiment due to changes in requirements or environment at that time.

4.4 Conclusion

The required throughput has been met with resources of the chips still available for other firmware. The accuracy of the filter is found to be sufficiently accurate, particularly given that the low false negative rate is a crucial target that has been met. The correctness of all modules has been extensively tested and can be guaranteed within the test suite's limits.

5 Conclusion

This work provides an electron track filter on FPGAs for the P2 experiment with high throughput and low false negatives. The firmware combines restrictions on the time and coordinate differences of two hits from one double layer to find possible electron tracks. The entire firmware has been thoroughly tested and the timing has been verified. The time filter accounts for the timestamp resolution of the MuPix tracking detector chip. A quartic curve defines a region of acceptable diffusion between the layer of a tracking detector double plane. A novel buffer mechanism enables the firmware to process a new hit in every clock cycle. These attributes lead to a simple filter allowing a high throughput of about 250MHz. These benchmarks hit the targets this work was set to achieve.

5.1 Future Work

The entire firmware could be expanded to work over multiple FPGAs. This would require an interface and the extra computation power is not needed for this application. If in the future more computing power is necessary, the detector grid could be split into blocks with one FPGA for each block. The maximum space filter width and height are then set as a margin at the edge of the block. If a hit would occur there the FPGA of that block and the neighbours of that margin would process that hit.
If the latency is required the calculation of the bean curve can be assisted by a lookup table. Latency was not a concern for this work so this approach was not implemented. The amount of possible inputs of the function is limited so storing the square for the left side of the equation and as part of the cube can improve performance. The same strategy could also be applied to the results on the left and right side. This would be at the expense of more storage, but as storage is barely used in the rest of the design, a lot of it is available for lookup tables.
There is also room for improvement on the filter side. It could be expanded to

include more parameters like the momentum. The space filter could also be designed to be more precise at the cost of speed. Depending on the requirements this could be integrated into the 'Compare Hits' module. To achieve this an in-depth analysis of the data from the simulation or by then particle accelerator experiment is needed.
If the throughput is found to be insufficient, reading in two hits from different layers per cycle would be possible. The 'Check in Region' module can store both in the different respective buffers in one cycle. A second 'Hit Receiver' module operating in parallel would be necessary. Each 'Hit Receiver' would be responsible for reading in hits from one layer.

Glossary

ASIC Application Specific Integrated Circuit. 3, 6

Block RAM Dual Port On-Chip Memory Random Access Memory. 4, 5, 8, 21, 22, 27–29, 31

CLB Configurable Logic Block. 4, 5

CPU Central Processing Unit. 5, 7

DDR4 RAM Double Data Rate 4 Synchronous Dynamic Random-Access Memory. 5

DSP Digital Signal Processing. 4, 5, 17, 27, 31

FIFO First In First Out. 24

FPGA Field Programmable Gate Array. 1, 3–6, 8, 9, 12, 13, 15, 17, 20, 21, 24, 27, 29, 31–33, 41

GPU Graphical Processing Unit. 5, 7

LUT Lookup table. 4, 5, 31

multiplexer device for selection of signals. 8

RAM Random Access Memory. 20

ROI Region of Interest. 7, 8

SIPO Serial In Parallel Out. 18

SISO Serial In Serial Out. 24

System Logic Cell Unified measurement for the number of Lookup tables on an FPGA. 4, 27

track Set of hits from the trajectory of one specific particle. 3, 5, 7, 8, 12, 14, 17–20, 22–24, 28–31, 33

Verilog IEEE Standardized Hardware Description Language.. 4

VHDL VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. 4, 11

Bibliography

[1]S. Agostinelli, J. Allison, K. Amako, et al. “Geant4—a simulation toolkit”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (2003), pp. 250–303 (cit. on pp. 11, 30).

[2]*AMD Kintex UltraScale FPGA KCU105 Evaluation Kit*. AMD. 2024. URL: `https://www.xilinx.com/products/boards-and-kits/kcu105.html` (visited on July 31, 2024) (cit. on pp. 5, 27).

[3]H. Augustin, N. Berger, C. Blattgerste, et al. “Performance of the large scale HV-CMOS pixel sensor MuPix8”. In: *Journal of Instrumentation* 14.10 (Oct. 2019), p. C10011 (cit. on pp. 9, 15, 17).

[4]Heiko Augustin, Ivan Perić, Andre Schöning, and Alena Weber. “The MuPix sensor for the Mu3e experiment”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 979 (2020), p. 164441 (cit. on pp. 5, 9, 28).

[5]Edward Bartz, Jorge Chaves, Yuri Gershtein, et al. “FPGA-Based Tracklet Approach to Level-1 Track Finding at CMS for the HL-LHC”. In: *EPJ Web of Conferences* 150 (2017). Ed. by C. Germain, H. Grasland, A. Lounis, D. Rousseau, and D. Varouchas, p. 00016 (cit. on p. 8).

[6]Dominik Becker, Razvan Bucoveanu, Carsten Grzesik, et al. “The P2 experiment: A future high-precision measurement of the weak mixing angle at low momentum transfer”. In: *The European Physical Journal A* 54.11 (Nov. 2018) (cit. on pp. 1, 8).

[7]Niklaus Berger. “The Mu3e Experiment”. In: *Nuclear Physics B - Proceedings Supplements* 248-250 (2014). 1st Conference on Charged Lepton Flavor Violation, pp. 35–40 (cit. on pp. 8, 9).

[8]Jason Cong and Kirill Minkovich. “Optimality study of logic synthesis for LUT-based FPGAs”. In: *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*. FPGA ’06. Monterey, California, USA: Association for Computing Machinery, 2006, pp. 33–40 (cit. on p. 5).

[9]Jason Cong and Kenneth Yan. “Synthesis for FPGAs with embedded memory blocks”. In: *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays*. FPGA ’00. Monterey, California, USA: Association for Computing Machinery, 2000, pp. 75–82 (cit. on p. 4).

[10]Marcel Corrêa, Luiz Neto, Daniel Palomino, Guilherme Corrêa, and Luciano Agostini. “ASIC Solution for the Directional Intra Prediction of the AV1 Encoder Targeting UHD 4K Videos”. In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2020, pp. 1–5 (cit. on p. 3).

[11]Neha Gupta. “Chapter One - Introduction to hardware accelerator systems for artificial intelligence and machine learning”. In: *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*. Ed. by Shiho Kim and Ganesh Chandra Deka. Vol. 122. Advances in Computers. Elsevier, 2021, pp. 1–21 (cit. on p. 3).

[12]P. Jamieson and J. Rose. “A Verilog RTL synthesis tool for heterogeneous FPGAs”. In: *International Conference on Field Programmable Logic and Applications, 2005.* 2005, pp. 305–310 (cit. on p. 4).

[13]Teimuraz Kopaleishvili. *Collision Theory: a Short Course*. 1995 (cit. on p. 1).

[14]Ian Kuon and Jonathan Rose. “Measuring the gap between FPGAs and ASICs”. In: *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*. FPGA ’06. Monterey, California, USA: Association for Computing Machinery, 2006, pp. 21–30 (cit. on p. 4).

[15]Pedro Paz and Mario Garrido. “Efficient Implementation of Complex Multipliers on FPGAs Using DSP Slices”. In: *Journal of Signal Processing Systems* 95 (Apr. 2023), pp. 1–8 (cit. on p. 4).

[16]Iurii Sorokin. “Parameterization-based tracking for the P2 experiment”. In: *EPJ Web Conf.* 150 (2017). Ed. by C. Germain, H. Grasland, A. Lounis, D. Rousseau, and D. Varouchas, p. 00012 (cit. on pp. 1, 7).

[17]*UltraScale Architecture Configurable Logic Block User Guide*. AMD. 2017 (cit. on p. 4).

[18]*UltraScale Architecture DSP Slice User Guide (UG579)*. Version 1.7. AMD. 2018 (cit. on pp. 4, 5).

[19]*UltraScale Architecture-Based FPGAs Memory IP Product Guide (PG150)*. Version 1.4. AMD. Apr. 2022 (cit. on p. 5).

[20]*Using the Vivado Timing Constraint Wizard*. AMD (cit. on p. 29).

[21]*Vivado*. Version 2024.1. AMD, July 30, 2024 (cit. on pp. 4, 5, 27, 28, 31, 32).

[22]*Vivado Design Suite User Guide Using Constraints*. AMD. 2022 (cit. on p. 29).

[23]*Vivado Design Suite User Guide: Power Analysis and Optimization*. AMD. 2021 (cit. on p. 32).

[24]Eric W. Weisstein. *Wolfram MathWorld Bean Curve*. 2024. URL: `https://mathworld.wolfram.com/BeanCurve.html` (visited on July 3, 2024) (cit. on p. 13).

[25]*Xilinx DS512 LogiCORE IP Block Memory Generator v7.1*. Version 7.1. AMD. Apr. 2012 (cit. on pp. 4, 5, 28, 29).

[26] Marco Zimmermann. “Particle Rate Studies and Technical Design Development for the P2 Silicon Pixel Tracking Detector”. PhD thesis. Mainz, 2019 (cit. on pp. 1, 2).

List of Figures

List of Listings

Appendix

A

A.1 Appendix: Usage of AI Tools

All conversations with an AI like ChatGPT can be found in the following repository:
https://gitlab.rlp.net/rjansked/bachelorthesisaiconversations

AI Tool	Used for	Why	When
ChatGPT	error debugging & search	time saving	during Firmware Development
gemma2 2b	search	time saving	thesis writing in overleaf
elicit	academic search	time saving	thesis research
Grammarly	grammar & spell checking	better readability	thesis writing in Overleaf
deepL Write	writing style	better readability	thesis writing in overleaf

Fig. A.1.: Overview of AI Tool used.

A.2 Appendix: Firmware Code

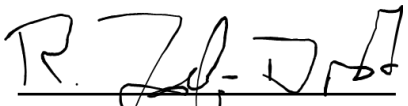
The entire firmware code used as well as the project directory can be found here:

https://gitlab.rlp.net/rjansked/electronDetectorFirmware

Declaration

I hereby declare that I have written the present thesis independently and without use of other than the indicated means. I also declare that to the best of my knowledge all passages taken from published and unpublished sources have been referenced. The thesis has not been submitted for evaluation to any other examining authority, nor has it been published in any form whatsoever. I duly noted the Regulations for Good Scientific Practice and Dealing with Scientific Misconduct.

Mainz, August 22, 2024



Raoul Jaime Janske-Drost