

# Application of FPGAs to Triggering in High Energy Physics

Sioni Paris Summers  
Imperial College London  
Department of Physics

CERN-THESIS-2018-248



A dissertation submitted to Imperial College London  
for the degree of Doctor of Philosophy

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.

# Abstract

The High Luminosity upgrade of the LHC will increase the instantaneous luminosity to  $5 \times 10^{34} \text{ cm}^{-2} \text{ s}^{-1}$ , resulting in an increase in the number of simultaneous proton-proton collisions per event (pileup) to the range 140-200. The CMS Level 1 Trigger system will be upgraded, and will reduce the 40 MHz event rate to 750 kHz. The system will perform a fast event reconstruction on FPGA devices, and select events for read out within a latency of 12.5  $\mu\text{s}$ .

The high level FPGA programming tool MaxCompiler is investigated for use in Level 1 Trigger applications. An existing trigger algorithm, originally developed with a Hardware Description Language, is reimplemented using MaxCompiler and compared to the original. Bitwise agreement between the outputs is observed, with half as many lines of code, at the expense of some extra FPGA resources.

A hardware demonstration of a proposed Level 1 track reconstruction is presented, with a Kalman Filter track fit developed with MaxCompiler. The performance of the tracking is investigated, as well as the potential for developing advanced algorithms with low latency using the tool. A high tracking efficiency, and precise parameter resolutions, are achieved with a 3.7  $\mu\text{s}$  latency in high pileup events. A boosted decision tree classifier, implemented with inference latency of a few clock cycles, is presented as a means to reject fake tracks.

After the Level 1 Trigger, events are further processed on commodity PCs in the High Level Trigger (HLT). The High Luminosity LHC will also challenge the HLT, which is projected to require twenty times the processing power used during LHC Run II. Part of the HLT tracking is ported to Maxeler Dataflow Engines (DFEs), a hardware acceleration technology. A faster rate of processing is achieved, but with an initial latency of the host-DFE communication that limits the performance. Steps which might yield acceleration are identified.

# Declaration

This thesis is the result of my own work, with contributions from others in collaboration. Where an argument, result or figure is not my own, the original work is acknowledged and indicated by a reference.

In Chapter 3, the algorithm implementation using MaxCompiler, and subsequent integration with an MP7 and testing, was performed by me, and led to the publication [1]. The algorithm was originally developed and implemented by others.

The work presented in Chapter 4 led to the publication [2], and was performed in collaboration with the other listed authors. My main contribution was the development of the Kalman Filter described in the chapter. I developed the implementation for the collaboration software, and studied the performance. The matrix mathematics FPGA implementation was developed by myself, while the control logic was developed by a collaborator. I carried out the investigation into removing fake tracks, including the training and testing of classifiers, and the implementation of the Boosted Decision Tree for the FPGA. I also implemented the calculations performed by the Geometric Processor for the FPGA.

I performed the development of the Kalman Filter presented in Chapter 5. This includes the implementation and testing of the FPGA, and the modification of the original software with an interface to the FPGA.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	LHC and HL-LHC . . . . .	1
1.2	Trigger and FPGAs . . . . .	4
1.2.1	Trigger . . . . .	4
1.3	Introduction to FPGA computing . . . . .	7
1.3.1	FPGA Programming . . . . .	8
1.3.2	Data Flow . . . . .	9
<b>2</b>	<b>The Large Hadron Collider and Compact Muon Solenoid Experiment</b>	<b>11</b>
2.1	Large Hadron Collider . . . . .	11
2.2	Compact Muon Solenoid Experiment . . . . .	13
2.2.1	CMS Phase II Upgrade . . . . .	14
2.3	Tracker . . . . .	16
2.4	Phase II Tracker . . . . .	16
2.4.1	Inner Tracker . . . . .	17
2.4.2	Outer Tracker . . . . .	18
2.4.3	Performance . . . . .	21
2.5	Electromagnetic Calorimeter . . . . .	22
2.5.1	Input to the Trigger . . . . .	23
2.6	Hadronic Calorimeter . . . . .	23
2.7	Muon System . . . . .	25
2.8	Level 1 Trigger . . . . .	27
2.8.1	Differences for Phase II . . . . .	27
2.8.2	Phase I Calorimeter Trigger . . . . .	30
2.9	High Level Trigger . . . . .	31
2.10	Track Reconstruction . . . . .	33

---

2.10.1	Seeding . . . . .	34
2.10.2	Track Building . . . . .	35
2.10.3	Track Fitting . . . . .	37
2.10.4	Track Selection . . . . .	38
<b>3</b>	<b>MaxCompiler for Level 1 Trigger Applications</b>	<b>39</b>
3.1	High Level FPGA Programming . . . . .	39
3.1.1	MaxCompiler . . . . .	41
3.2	Jets and Energy Sums . . . . .	42
3.3	Algorithm . . . . .	44
3.3.1	Jets . . . . .	44
3.3.2	Energy sum . . . . .	47
3.4	MaxJ Implementation . . . . .	47
3.4.1	Interface with MP7 . . . . .	48
3.5	Comparison . . . . .	49
3.5.1	Functional Correctness . . . . .	49
3.5.2	FPGA Resources . . . . .	49
3.6	Summary . . . . .	51
<b>4</b>	<b>Track Reconstruction for the Level 1 Trigger</b>	<b>52</b>
4.1	Track Trigger Demonstrator . . . . .	53
4.1.1	Time Multiplexed Track Trigger . . . . .	53
4.1.2	Geometric Processor . . . . .	54
4.1.3	Hough Transform . . . . .	56
4.1.4	Kalman Filter . . . . .	62
4.1.5	Duplicate Removal . . . . .	75
4.1.6	Demonstrator System . . . . .	77
4.1.7	System Performance . . . . .	78
4.1.8	Latency . . . . .	84
4.1.9	System capacity . . . . .	85
4.1.10	Towards a Final System . . . . .	86
4.1.11	Future Developments . . . . .	89
4.2	Identifying Fake Tracks with a Boosted Decision Tree . . . . .	91
4.2.1	FPGA implementation . . . . .	95
4.3	Conclusions . . . . .	99

---

<b>5</b>	<b>Hardware Acceleration of Track Reconstruction in the High Level</b>	
	<b>Trigger</b>	<b>101</b>
5.1	Kalman Filter on a DFE . . . . .	101
5.1.1	MaxJ Implementation . . . . .	103
5.1.2	Data Types . . . . .	103
5.1.3	Data Flow . . . . .	105
5.1.4	Interface with CMSSW . . . . .	107
5.1.5	Performance . . . . .	109
5.2	Future Developments . . . . .	111
5.2.1	Design Improvements . . . . .	111
5.2.2	Technological Improvements . . . . .	113
5.3	Conclusion . . . . .	116
<b>6</b>	<b>Conclusion</b>	<b>118</b>
	<b>Appendices</b>	<b>128</b>
<b>A</b>	<b>Integer Division</b>	<b>129</b>
A.0.1	Floating Point . . . . .	131

# List of Figures

1.1	Timeline of the LHC project. . . . .	2
1.2	The cross section for several standard model processes as a function of centre-of-mass energy. . . . .	4
2.1	The CERN accelerator complex. . . . .	12
2.2	Cutaway of the CMS detector. . . . .	15
2.3	One quarter view of the proposed layout of the CMS Phase II tracker in the $r$ - $z$ plane. . . . .	17
2.4	The pixel-strip (PS) and strip-strip (2S) implementations of the $p_T$ module concept. . . . .	19
2.5	Layout of the $p_T$ modules in the outer tracker used for the track trigger demonstrator. . . . .	21
2.6	One quarter view of the ECAL in the $r$ - $z$ plane. . . . .	22
2.7	View of the grouping of ECAL crystals into trigger towers. . . . .	24
2.8	A schematic of the HCAL in the $r$ - $z$ plane. . . . .	25
2.9	Schematic of a one quarter slice of the CMS detector, highlighting the layout of the muon system. . . . .	26
2.10	Diagram of components of the Phase I Upgrade of the CMS Level 1 Trigger. . . . .	28
2.11	Overview of the components of the Phase II Level 1 Trigger. . . . .	29
2.12	System architecture of the Level-1 Calorimeter Trigger. . . . .	30
2.13	The timing of track reconstruction at CMS. . . . .	34
3.1	Dataflow graph for $z = x^2 + y$ . . . . .	42
3.2	Templates describing the definition of a Level 1 Calorimeter Trigger jet object. . . . .	43
3.3	The scheme for reuse of partial sums of jet energy to save FPGA resources. . . . .	45

3.4	Dataflow graph of a bitonic sorting network with four parallel inputs.	46
3.5	Accumulation stage of the jet sorter. . . . .	47
3.6	Output distributions of reconstructed jet and $E_T$ parameters from the VHDL and MaxJ implementations of the L1 Calorimeter Trigger algorithms. . . . .	50
4.1	Segmentation of the tracker into processing regions. . . . .	55
4.2	The Geometric Processor (GP) routing network. . . . .	56
4.3	Examples of Hough Transform evaluation for straight lines. . . . .	58
4.4	Schematic of the implementation of one Hough Transform column processor. . . . .	60
4.5	Schematic of one Hough Transform array as implemented in FPGA firmware. . . . .	61
4.6	An example track candidate found by the HT with intermediate KF states shown. . . . .	66
4.7	Histograms of numerical differences between the firmware and software for each state update. . . . .	69
4.8	A maximally parallel matrix multiplication dataflow graph. . . . .	70
4.9	Dataflow graph of the Kalman Filter state update. . . . .	71
4.10	A schematic of the main components of the Kalman Filter processing node. . . . .	73
4.11	Fraction of track candidates which pass through four iterations of the KF state updater as a function of the processing timeout. . . . .	74
4.12	An illustration of one track producing three candidates in the Hough Transform. . . . .	76
4.13	Duplicate Removal FPGA implementation architecture. . . . .	77
4.14	Photograph and diagram of the demonstrator crate. . . . .	79
4.15	Track reconstruction efficiency of the ‘full-chain’ algorithm in $t\bar{t}$ events with 200 PU. . . . .	82
4.16	Resolutions of the track parameters as fitted by the Kalman Filter. . . . .	83
4.17	Resolutions of the track parameters as fitted by the Kalman Filter for muons. . . . .	84
4.18	Number of stubs output per GP sub-sector per event and number of track candidates found by the HT per sub-sector per event. . . . .	86
4.19	One quarter of the $r - z$ plane of the CMS outer tracker with tilted modules in the barrel. . . . .	90

4.20	Receiver Operator Characteristic (ROC) curve for the five classifiers under investigation and histogram of separation between genuine and fake tracks as a function of BDT score. . . . .	92
4.21	Distributions of fake tracks before and after BDT class prediction. . .	94
4.22	Schematic for the FPGA implementation of a decision tree with a depth of 3. . . . .	96
4.23	Area under the ROC curve for a scan over BDT hyper-parameters, and the impact on latency and resources. . . . .	97
4.24	Decision contour for the FPGA and CPU implementations of a BDT.	98
4.25	Number of classified data points vs. execution time on the FPGA and CPU. . . . .	99
5.1	Pseudo-code of the track building procedure of the Combinatorial Track Finder. . . . .	102
5.2	Kernel Graph representation of the HLT 5 parameter, floating point, Kalman Filter. . . . .	104
5.3	Diagram of execution placement of logical components of the track building, and the flow of data between the devices. . . . .	106
5.4	Pseudo-code of the track building procedure modified to coalesce multiple states into a stream. . . . .	107
5.5	Histogram of the number of measurements found for all states. . . .	108
5.6	C++ functions and data structures for the operation of the Kalman Filter DFE from the host application. . . . .	109
5.7	Measured time to perform a number of iterations of Kalman Filter state update on CPU and DFE. . . . .	110
5.8	DFE configuration possibilities with different PCIe capabilities. . . .	114

# List of Tables

2.1	Beam parameters of the LHC design and HL-LHC. . . . .	14
3.1	Number of each type of resource used by each implementation of the jet and energy sum algorithm. . . . .	49
4.1	Utilisation of FPGA resources for the stub unpacking and pre-processing block. . . . .	56
4.2	FPGA resource utilisation for one HT column processor, and one HT accumulator array. . . . .	61
4.3	Kalman Filter resource utilisation. . . . .	75
4.4	Resource usage of one instance of the Duplicate Removal block. . . .	77
4.5	Track reconstruction quality at different stages of the demonstrator chain, for $t\bar{t}$ events with 200 PU interactions. . . . .	80
4.6	Measured latency of the demonstrated components of the track re- construction chain . . . . .	85
4.7	Resource availability of the FPGA on an MP7, and of two current generation Xilinx chips which are suitable for HL-LHC triggers. . . .	88
4.8	Tracking performance for both flat and tilted barrel tracker geometries.	89
4.9	Post-synthesis resource usage of the trained BDT with 4 features, 100 trees with a maximum depth of 3, for 3 different FPGA parts. . . . .	97
5.1	The range of the exponent of floating point variables in the Kalman Filter state update. . . . .	105
5.2	Resource usage of the Kalman Filter state updater and CPU IO on a Maia DFE. . . . .	109

# Chapter 1

## Introduction

### 1.1 LHC and HL-LHC

The Large Hadron Collider (LHC), situated 100 m underground near Geneva, Switzerland, is the largest and most powerful synchrotron ever constructed. Particles, most often protons, are accelerated up to 6.5 TeV by superconducting magnets in two counter-rotating beams in a 27 km circumference ring. At four sites around the ring the beams are brought into head on collision with a centre of mass energy of 13 TeV. At each of these sites a detector measures the new particles created by the collision. These detectors are the experiments: CMS, ATLAS, LHCb, and ALICE.

The LHC was constructed to further the understanding of the fundamental forces of nature: for the discovery of new particles and new physics. To date the Standard Model (SM) of particle physics is the theory that best describes the fundamental forces, excluding gravity: electromagnetic; weak; and strong nuclear force. The electromagnetic and weak forces are unified to a single force, the electroweak force [3–5]. A breaking of the electroweak symmetry is necessary to provide the  $W^\pm$  and Z vector bosons with mass. The Brout-Englert-Higgs mechanism [6–9] provides this symmetry breaking, with the addition of a scalar boson, the Higgs. At the time of the construction of the LHC, the Higgs boson was the only SM particle not yet directly observed. The potential for its discovery was a particular motivation for the LHC.

Despite its successes, there are many phenomena that the SM cannot account for. There is no SM particle that could be a candidate for Dark Matter, well motivated by cosmological observations. Neutrinos in the SM are massless, which cannot be reconciled with the observation of neutrino oscillation, which requires that they have



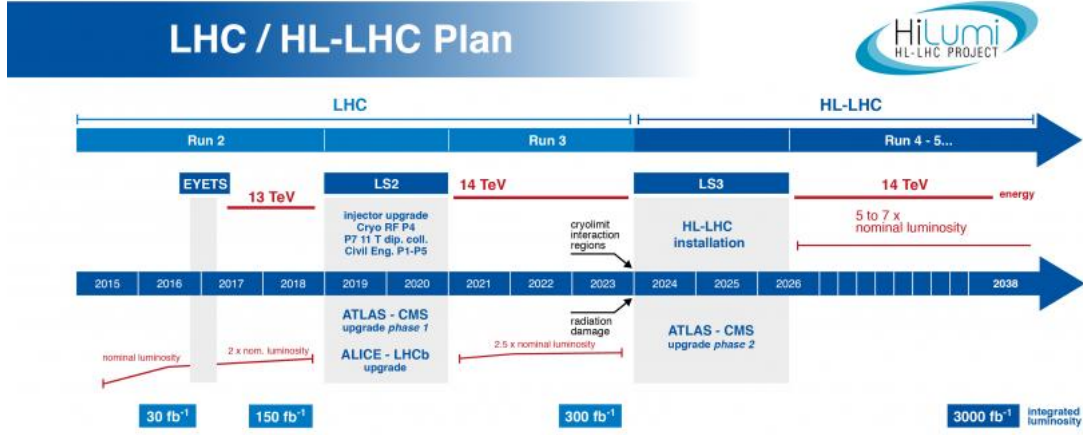


Figure 1.1: Timeline of the LHC project.

mass. There is no explanation for matter-antimatter asymmetry, with every SM interaction conserving baryon number. Theories such as Supersymmetry propose extensions to the SM which might resolve some of these outstanding issues, and predict new particles with masses accessible at the LHC. Searches for signs of physics beyond the Standard Model was another major motivation for the LHC.

Despite the success of the LHC in discovering the Higgs boson [10, 11], no new physics has been discovered. To manifestly increase the potential for discovery of any new phenomenon, that has so far proved elusive, the LHC will be upgraded. This upgrade will take place after LHC Run III, planning to restart according to the schedule shown in Figure 1.1, around mid-2026. The upgraded machine will be called the High Luminosity LHC (HL-LHC), a reflection of the beam luminosity increase by a factor five over the LHC design luminosity. This will facilitate the collection of  $3000 \text{ fb}^{-1}$  over a ten year period, compared to the  $300 \text{ fb}^{-1}$  collected by the end of Run III.

For the CMS experiment at the HL-LHC [12], the Higgs boson will remain a major subject of interest. With the  $3000 \text{ fb}^{-1}$  dataset, many measurements of Higgs properties will be made possible with high precision. The couplings of the Higgs boson to Standard Model fermions are currently measured with around 20% uncertainty, which will decrease to around 1% with the full HL-LHC data. The Higgs self coupling will be measurable, with the 40 fb cross section of di-Higgs production.

Vector boson scattering processes, which can be sensitive to new physics through anomalous triple- and quartic-gauge couplings, will be another class of processes to benefit from the additional integrated luminosity [13]. These processes probe

the nature of the electroweak symmetry breaking and the role of the Higgs boson. Tagging of forward jets is essential to reduce background contributions from pileup, and a greater tracker acceptance will increase sensitivity.

CMS will also continue to carry out direct searches for new particles. New particles with masses of up to a few TeV, accessible at the LHC, but with small cross sections, may become detectable at the HL-LHC where they were not at the LHC. The upper limits on the accessible mass of new particles will be extended somewhat by the extra integrated luminosity. In the absence of any new discoveries at the LHC, more exotic event topologies, for example with semi-stable particles that decay inside the detector, but some distance from the beam line, become topics of interest. The CMS detector must maintain sensitivity to these channels in the more challenging environment.

The Phase II upgrade of the CMS detector will prepare the experiment for HL-LHC conditions [12]. Much of the detector will be replaced, although the overall concept will remain. A new tracker, with greater acceptance and finer granularity, will also provide input to the Level 1 Trigger for the first time. A new endcap calorimeter, a highly segmented sampling calorimeter, will provide precise energy resolution, particle identification capability, and pileup mitigation. A correlator trigger will utilise these detectors with full particle-flow [14] reconstruction. These upgrades will increase the power of the trigger, while presenting a significant challenge due to the harsh latency and throughput constraints.

The Level 1 Trigger will use the latest generation high performance Field Programmable Gate Array (FPGA) devices for computing. These latest devices are increasingly commonly programmed using high level programming languages and compilers. These approaches promise to improve the development time and effort of complicated algorithms: an appealing prospect for the trigger processing at CMS at the HL-LHC. In this thesis, the high level tools of Maxeler Technologies were turned to the development of Level 1 Trigger algorithms. In Chapter 3 the tool is benchmarked against the low-level handwritten approach to test its suitability for applications with microsecond latency and tight resource constraints. It is then used in Chapter 4 in the development of track reconstruction for the Level 1 Trigger. Finally in Chapter 5, the high level tracking code is turned to an application targeting the High Level Trigger, using a Maxeler Technologies Dataflow Engine FPGA accelerator card.

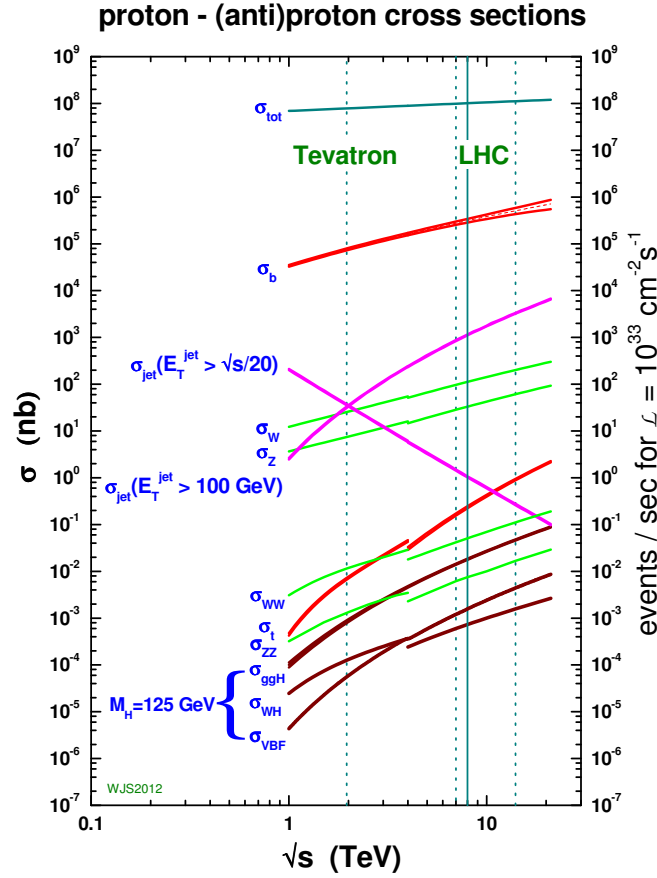


Figure 1.2: The cross section for several standard model processes as a function of centre-of-mass energy [15].

## 1.2 Trigger and FPGAs

### 1.2.1 Trigger

The trigger system of a HEP experiment determines when to read signals from the detector and write them to storage for later analysis. The first trigger, for example, activated a cloud chamber when a high energy particle caused a coincidence of signals in Geiger counters surrounding the chamber [16]. Triggers at the LHC experiment perform the same operation: determine when ‘something of interest’ occurs, and trigger the reading out of the detector for later analysis. These systems can perform vastly more sophisticated analysis to reach their decision, but the principle remains.

Unlike the cloud chamber trigger, required to capture an event occurring at an unknown time, the LHC collisions occur at a metronomic 40 MHz. The need for

a trigger arises because the vast majority of collisions do not produce any results of interest. This may be because no hard proton scatter took place, or because the products are of some well understood process. The cross sections of several electroweak processes in proton-proton collisions are shown in Figure 1.2, and can be seen to be several orders of magnitude less than the total cross section.

The ability to sift out such uninteresting events at the earliest opportunity allows for a saving in infrastructure: a lower rate of event storage, requires lower bandwidth to transport data from detector to storage, and less total storage capacity. The workload to reconstruct and analyse event data is also reduced, which is already a major globally distributed computing effort [17]. The existence of the trigger system is also in some sense ‘built in’ to the rest of the CMS detector, as components responsible for the readout of data would be unable to read out at the full 40 MHz rate, so the trigger *must* perform this reduction.

Without yet describing the trigger as a system, the concept of a trigger ‘menu’, as used at CMS, can be understood. The trigger menu defines what is an ‘interesting’ event worthy of sending to permanent storage. This takes the form of a list of particle types and kinematic requirements. The items on the menu are a balance between the interests of different analysis groups and the available bandwidth for reading data off the detector and sending it to disk. Processing of the measurements made by the CMS detector is required to construct the information used in the menu to make the ultimate trigger decision. This processing is executed in the trigger system, as well. Since, necessarily, all detector data must be stored somewhere until the trigger decision is made, a latency restriction applies in order to avoid loss of events. At CMS the data is kept in buffers on or near the detector until a trigger signal is received.

The CMS trigger, which will be described in more detail in Chapter 2, is a two stage processing system. The first stage – Level 1 – is situated underground in the cavern adjacent to the detector. The Level 1 Trigger (L1T) processes events at the full 40 MHz and triggers readout of around 100 kHz: a factor of 400 reduction. Latency is constrained to 4  $\mu$ s by the depth of buffers on tracker front end chips. Only a subset of the full detector data is sent to the L1T, to limit the bandwidth required for transmission of data: the calorimeter (at a reduced granularity), and muon detectors are used while the tracker is not. Processing is carried out on FPGA devices with large data bandwidth and significant parallel computation.

Events which pass L1T are sent to a computing facility directly above the detector on the surface, where space presents less of a restriction. At this High Level

Trigger (HLT) commodity PCs are used to reconstruct events using all detectors at their full granularity. The processing at HLT is also latency constrained, with a limitation from the incoming event rate and the number of processors. With around 1,000 CPU nodes and the aforementioned 100 kHz rate, the limit is around 200 ms.

By contrast the ATLAS experiment operates a three level trigger [18]. The first level (L1) is similar to the CMS L1T, with a latency of  $2.5\,\mu\text{s}$  and a maximum output rate of 75 kHz. The Level 2 Trigger (L2) is comprised of conventional processors, reducing the rate to 3 kHz within 40 ms per event. The processing at this second level performs similar algorithms to the full reconstruction, but only within regions of interest (RoI) identified by L1. This RoI seeded reconstruction uses around 2–6% of the total event data. Events passing L2 are then fully reconstructed, also using conventional processors in the Event Filter (EF). The EF reduces the event rate to disk to 200 Hz within 4 s. The HLT system consists of around 1300 compute nodes split between L2 and EF. ATLAS triggers are also produced after comparing reconstructed particles with a menu of events of interest.

The LHCb experiment uses a two level trigger: one level of hardware and one of software, with the majority of the processing performed by the high level (software) trigger [19, 20]. The hardware level (L0) reduces the rate from around 40 MHz to 1 MHz in  $4\,\mu\text{s}$ . Only the calorimeter and muon system are input to L0, and events with high transverse momentum ( $p_T$ ) muons or large transverse energy deposited in the calorimeter are read out. A CPU farm of 27000 cores reduces the rate to 12.5 kHz. The software running at the HLT is structured into two subcomponents – HLT1 and HLT2. HLT1 performs partial reconstruction, and reduces the rate to around 70 kHz. HLT2 then reconstructs all tracks above a  $p_T$  threshold for the remaining events. LHCb also operates a ‘deferred trigger’ which stores 20% of all L0 accepted events onto disks local to the HLT, and processes them during gaps in LHC operation, which effectively increases the processing power of the farm.

For Run III LHCb will upgrade to a trigger-less readout [21]. This upgrade will see the entire detector read out to a farm of conventional processors at the full 40 MHz event rate. With the current trigger system, a significant loss of efficiency is incurred by L0. The LHCb detector bandwidth is significantly less than the CMS bandwidth, at  $4\,\text{TB s}^{-1}$ . This upgrade might also see the HLT architecture change from CPU-only, to a configuration with FPGA coprocessors [22]. Certain algorithms used at the HLT show an improvement in processing speed using FPGA accelerators [23].

This thesis is primarily concerned with the preparation of the CMS trigger for

the HL-LHC, and the role of FPGAs within this development. Where it relates to processing performed on CPUs, it looks to augment the performance with FPGAs. A short primer on computing with FPGAs is provided, for the uninitiated.

### 1.3 Introduction to FPGA computing

FPGAs are the computing platform of choice for triggers in HEP experiments. They are characterised by their huge number of configurable blocks for logical operations – generically grouped with the term ‘resources’ – including specialised components for multiplication, memory, and input/output (IO). Compared to CPUs, the clock frequencies achieved on FPGAs are a factor ten lower. However, the FPGA’s power comes from the massive parallelism which they enable. Modern devices contain around 10,000 multipliers, 10 MB of internal memory with  $10 \text{ TBs}^{-1}$  peak bandwidth, millions of logic cells performing arbitrary six bit functions, and transceivers capable of delivering data at  $1 \text{ TBs}^{-1}$ . Registers are placed at the output of each component, holding the value of their data until the next clock cycle. Components are distributed across the chip, like machines in a production line, and linked with a ‘routing fabric’.

In Chapters 3 and 4 the Xilinx Virtex 7 device is used. The basic computation unit is the Look-Up Table (LUT) [24]. In the Virtex 7 these are 6-bit input function generators. LUTs are arranged in Configurable Logic Blocks (CLBs), each containing eight LUTs, flip-flops on the LUT outputs, high-speed carry logic (for chaining LUTs to perform arithmetic) and multiplexers. Arithmetic such as multiplication and accumulation is better executed using dedicated digital signal processing (DSP) blocks [25]. In the Virtex 7 these contain a  $25 \times 18$  bit multiplier for two’s complement data. Around the multiplier each DSP also contains an accumulator, pre-adder, logic unit, and multiple registers. While LUTs can be used as memories, a dedicated Block RAM (BRAM) [26] component allows memories with more than a small number of addresses to be constructed efficiently. The basic memory unit in the Virtex 7 is the BRAM36, a 36 Kb RAM. The BRAM36 is a dual-port RAM with two independent sets of ports, and the memory is partitioned into two 18 Kb regions which can be utilised independently, or as one. The aspect ratio of the data and address width is configurable, supporting from  $32 \text{ K} \times 1$  to  $512 \text{ K} \times 72$  modes. With many of each of these components across the chip, an FPGA can perform massively parallel computation.

Use of the flip-flops in each component causes the data to propagate at each

tick of a clock signal, with the data at the input of the flip-flop only transferring to its output (and the next computation) at the edge of the clock. The clock frequency at which an FPGA runs depends on the program being executed. The clock period is limited by the longest separation between any two components used in the design: try to use a clock period shorter than this, and the design will malfunction. Achieving high clock frequencies is part of the FPGA program design process. Registers must be used to segment the processing into small chunks, and processing should ideally be ‘local’ – using only data from nearby computations.

### 1.3.1 FPGA Programming

The role of the FPGA programmer is to specify which functions should be implemented with logic cells, and how the blocks should be connected for their application. This is aided by the use of ‘synthesis tools’, which map the program description to the available FPGA resources. The ‘place and route’ process specifies precisely which particular resource should be used in order to meet the designer’s placement and timing constraints. Hardware Description Languages (HDLs), such as VHDL and Verilog, are typically used to write applications for FPGAs. Some level of abstraction from the base components is provided, allowing, for example, the programmer to instantiate a multiplication and allowing the synthesis tool to infer whether specialised multipliers or general logic should be used. These languages are nonetheless typically ‘close to the metal’, and far removed from the domain of the typical C++ developer. They might seem extreme even to the experienced Assembly programmer, who is never concerned with the placement of clock edges in their program.

Efforts to create higher level programming languages for FPGAs have existed for almost as long as FPGAs. The principle is exactly the same as for higher level languages for conventional CPU programming: by adding layers of abstraction between the language and the implementation on silicon, programmers are able to realise more complex computations on their data. By removing the requirement that the developer have expert knowledge of the architecture, the platform is opened up to new people. When the developer is able to spend less time on the minutiae, they are freed to optimise their design at the algorithmic level. However, as also with high level CPU programming, there are concerns as to their efficacy. Is the compiler able to map concepts to compute components as efficiently as the programmer? Is it possible for the developer to know how a line of code will be implemented by the

compiler?

For CPU programming, for a long time, the favour has fallen on the side of the high level paradigm. The ease of creating programs in C++ compared to Assembly; in Python compared to C++, vastly outweigh any reservations on the effectiveness of the compiler. For FPGA programming, and in particular for FPGA applications intended for Level 1 Triggers of LHC experiments, however, the conversation is ongoing.

### 1.3.2 Data Flow

In parallel and dataflow computing it is informative to think of the computational loops involved in an algorithm. The existence or absence of data dependencies between loop iterations has a large impact on how an algorithm may be implemented with low latency in a parallel processor, such as an FPGA. In the following code listing the operation at iteration  $i$  does not depend on the result of any other iteration. This loop could be ‘fully unrolled’ in a sufficiently large FPGA (that is,

```
1  int x[];  
2  for(i = 0; i < iMax; i++){  
3      x[i] = f(i);  
4  }
```

each iteration could execute on different resources) and each iteration could execute simultaneously. In the second code, however, the result at iteration  $i$  depends on the result at iteration  $i - 1$ . The execution of iteration  $i$  must now follow that of  $i - 1$ . It

```
1  int x[];  
2  x[0] = 12;  
3  for(i = 1; i < iMax; i++){  
4      x[i] = f(x[i-1]);  
5  }
```

may still be possible to fully unroll this loop, but the latency will be longer than for a loop performing a similar function with no dependency. Still more complexities arise in scenarios where, for example, the iteration itself is data dependent. This may arise when there is a termination condition, or the iteration limit is a variable derived from data.



In general, computation performed in the trigger has favoured ‘fully unrolled’ loops. In Chapter 3, the input data from the calorimeter can map neatly to data arrays within the FPGA with no data dependence, and processing executes in a rigidly deterministic manner. This contrasts with the offline jet reconstruction algorithm anti- $k_T$  [27], which is completely iterative with loop dependencies, and disregarded for use in the trigger.

In the Phase II detector, the extra granularity provided to the trigger will make ‘zero suppressed’ detector readout (whereby channels with no signal above a threshold do not send data) essential. This will prevent convenient correspondence between detector elements and data locations in the FPGAs. The types of algorithm used to reconstruct particles in the new detectors tend also to be highly iterative: track reconstruction offline favours the Combinatorial Kalman Filter, while the particle-flow algorithm loops over lists of particles to match them. To fully utilise the tracker and endcap-calorimeter in the Level 1 Trigger, algorithms with complicated loop dependencies will be required, and imagination must be used to fit them within the harsh constraints of latency, throughput and resources. For these developments, a high level language might enable the realisation of more sophisticated algorithms, with more time spent optimising the data flow when the low level details are handled by a compiler.

## Chapter 2

# The Large Hadron Collider and Compact Muon Solenoid Experiment

### 2.1 Large Hadron Collider

A significant accelerator complex, depicted in Figure 2.1 is required to accelerate protons up to 6.5 TeV. Initially accelerated with a linear accelerator, the protons are then injected into a series of synchrotrons of increasing circumference. The series is as follows [28]. Protons from hydrogen gas are first accelerated by LINAC2 (Linear Accelerator 2) to 50 MeV, into the PSB (Proton Synchrotron Booster). The PSB accelerates the protons to 1.4 GeV, before injecting them into the Proton Synchrotron (PS), which accelerates the beam to 25 GeV. The beam is next injected into the Super Proton Synchrotron (SPS), which is the final booster before the LHC, and accelerated up to 450 GeV. Finally the beams are injected into the LHC which accelerates them to their ultimate energy of 6.5 TeV.

The beams circulate in 2808 distinct bunches of protons, each containing around  $10^{11}$  protons, and separated by 25 ns (around 7.5 m). Each collision between protons in two bunches is termed a ‘bunch crossing’, and the collision and its products are referred to as the ‘event’. With bunch crossings spaced by 25 ns, the LHC event rate is 40 MHz.

For any specific particle interaction the rate of event occurrence  $N$  is:

$$N = L\sigma, \tag{2.1}$$



$300 \text{ fb}^{-1}$  will have been collected by CMS. During the Long Shutdown that follows, LS3, the accelerator and experiments will prepare for high luminosity operation, dubbed the High Luminosity Large Hadron Collider, or HL-LHC. An instantaneous luminosity of  $5 \times 10^{34} \text{ cm}^{-2} \text{ s}^{-1}$  is planned for this phase. Over the projected 10 year running time of the HL-LHC,  $3000 \text{ fb}^{-1}$  of integrated luminosity will be collected by CMS and ATLAS.

By equation 2.1 the number of simultaneous pp collisions per bunch crossing (pileup) is proportional to the instantaneous luminosity, and so will increase during the HL-LHC compared to the LHC conditions. At the instantaneous luminosity expected for the HL-LHC the mean pileup will be 200, where the original LHC design value was 27.

Beam parameters of the LHC and HL-LHC accelerator are presented in table 2.1. The most significant change arises from the use of new large aperture inner triplet quadrupole magnets of  $\text{Nb}_3\text{Sn}$  with a 12 T peak magnetic field. These can yield a much shorter  $\beta^*$  in the collision region. However the crossing angle is increased in the process, which reduces the geometric reduction factor and hence limits the luminosity increase. This will be mitigated by the addition of superconducting RF crab cavities which rotate each bunch to collide head on. These can also provide a mechanism for luminosity levelling.

Without levelling, the luminosity profile decreases throughout a fill from an initial peak value as  $N_b$  decreases during proton collisions. Levelling will maintain a constant luminosity which is lower than the maximum, but yielding the same integrated luminosity over a fill. This provides more stable pileup conditions for the experiments, and deposits less energy into the interaction region magnets from debris [30].

## 2.2 Compact Muon Solenoid Experiment

The Compact Muon Solenoid (CMS) experiment is situated at one of the collision points on the LHC ring. A cutaway diagram, showing the various subsystems is shown in Figure 2.2. The detector comprises a cylindrical barrel section with planar endcaps, and measures 28.7 m in length with a diameter of 17 m. Propagating radially outwards from the centre, particles first pass through the tracker, which measures points along the trajectories of charged particles. Radially outwards of the tracker is the electromagnetic calorimeter (ECAL), followed by the hadronic calorimeter (HCAL), each measuring particle energy. These detectors are contained

Table 2.1: Beam parameters of the LHC design and HL-LHC [30].

Parameter	LHC (design)	HL-LHC
Proton energy [TeV]	7	7
$n_b$	2808	2748
$N_b$	$1.15 \times 10^{11}$	$2.2 \times 10^{11}$
$f_{rev}$ [kHz]	11.2	11.2
$\beta^*$ [m]	0.55	0.15
$\epsilon_n$ [ $\mu\text{m rad}$ ]	3.75	2.50
$\theta_c$ [ $\mu\text{rad}$ ]	285	590
$\sigma_z$ [cm]	7.55	7.55
$\sigma$ [ $\mu\text{m}$ ]	16.7	7.13
$F$	0.84	0.305
Peak luminosity [ $\text{cm}^{-2}\text{s}^{-1}$ ]	$1.0 \times 10^{34}$	$5.0 \times 10^{34}$
Mean events per crossing	27	198

within the eponymous superconducting solenoid, and the 3.8 T magnetic field it produces. Muon chambers are positioned outwards of the solenoid. The return yoke of the magnet is interleaved with the muon chambers.

A right handed Cartesian coordinate system is defined with the origin at the nominal interaction point. The  $x$  axis points towards the centre of the LHC, the  $y$  axis points vertically upwards, and the  $z$  axis along the anticlockwise rotating beam. The  $z$  axis is sometimes referred to as the longitudinal direction, and the  $x$ – $y$  plane is also referred to as the transverse plane. Transverse energy and momentum,  $E_T$  and  $p_T$  respectively, are therefore the magnitude of energy and momentum in the  $x$ – $y$  plane. From this coordinate system, other frequently used quantities are derived. The azimuthal angle  $\phi$  is measured in the  $x$ – $y$  plane from the  $x$  axis, and the radius  $r$  in the same plane completes a cylindrical coordinate system along with the  $z$  axis. The angle  $\theta$  is measured from the  $z$  axis in the  $r$ – $z$  plane, and the dip angle  $\lambda = \pi - \theta$ . Pseudorapidity is then defined to be  $\eta = -\ln \tan \theta/2$ .

### 2.2.1 CMS Phase II Upgrade

The luminosity of the HL-LHC will necessitate significant changes to the LHC experiments. In part this will be due to damage to the detectors from radiation already deposited over the lifetime of the LHC. Detectors will also be improved, generally with increased granularity, to further enable the mitigation of pileup in the reconstruction. In both the trigger and offline analysis the objects of interest are the particles produced from the primary vertex – the hard scatter surrounded

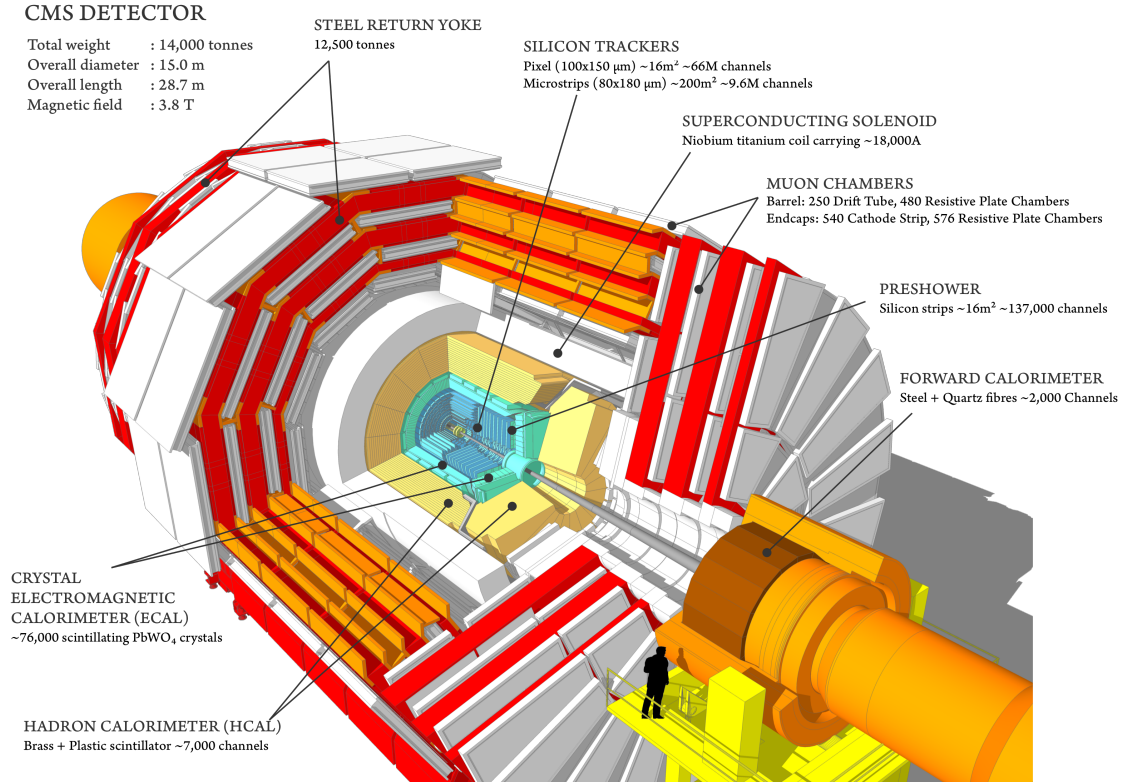


Figure 2.2: Cutaway of the CMS detector [31].

by the soft scatters of pileup. With more pileup interactions it becomes increasingly difficult to correctly attribute particles to the primary vertex, and to accurately determine the particle energy. The CMS experiment upgrades are presented in detail in [12].

Damage is dealt to the detector in the form of radiation from the particles produced by particle collisions. Charged particles ionise the detectors, and nuclear interactions produce particle cascades. Electromagnetic cascades are produced by the interaction of  $e^+e^-$  pairs, themselves the product of photon interactions in material, in the tracker. Calorimeters, based on scintillating materials, generally lose transparency when subjected to high radiation doses. This effect reduces the amplitude of signals, which, after calibration, effects the energy resolution. Detectors with better resilience to radiation damage, not just in a material sense, but in terms of detector performance, will be used.

The particle-flow technique [14] is the primary reconstruction technique at CMS currently, with ‘pileup per particle identification’ (PUPPI) [32] for the mitigation of pileup. These rely on the ability to separate energy deposits of particles in the

calorimeter, therefore requiring adequate energy resolution and granularity; efficient reconstruction of charged particles in the tracker; muon identification and momentum measurement from the muon systems. Maintaining the performance of the detector in the face of extreme pileup generally requires better resolution of reconstructed objects: in  $p_T$  and spatially.

In order to make use of the increased data rate in analysis, the trigger and readout must also be upgraded. It is desirable to maintain trigger energy and momentum thresholds as low as possible, within the limitations of the readout system, to have sensitivity to low mass particles. The energy and momentum resolution of the trigger must be improved to achieve this without losing efficiency. More advanced mitigation of pileup will be required for the trigger, to reduce the impact of combinatorial background. For this purpose, the tracker will provide input to the Level 1 trigger, enabling identification of pileup particles, and allowing the L1T to perform particle-flow-like reconstruction.

## 2.3 Tracker

The innermost sub-detector is the silicon tracker [33]. Planes of silicon sensors measure the position of charged particles along their trajectory. Particle momentum can be measured by determining the curvature of the particle trajectory in the 3.8 T magnetic field. An inner pixel detector with high spatial resolution allows a precise measurement of the vertex. Tracks which, when followed, lead to measurements in other sub-detectors provides a means of particle identification:  $e/\gamma$  like deposits in the calorimeter can be distinguished, and tracks created by muons can be identified.

## 2.4 Phase II Tracker

The CMS tracker will be completely replaced for the HL-LHC. Damage done to the existing detector due to radiation at the end of Run III will necessitate the change. Radiation damages the lattice in silicon detectors, such as the tracker, and alters the electrical behaviour. This leads to an increased leakage current, reduces the charge collection efficiency, and increases the depletion voltage, which all result in lower amplitude signals from charged particles. A reduction in charge sharing between sensors in the pixel detector worsens the spatial resolution of hits, directly impacting the tracking performance. The tracker, situated closest to the beam pipe,

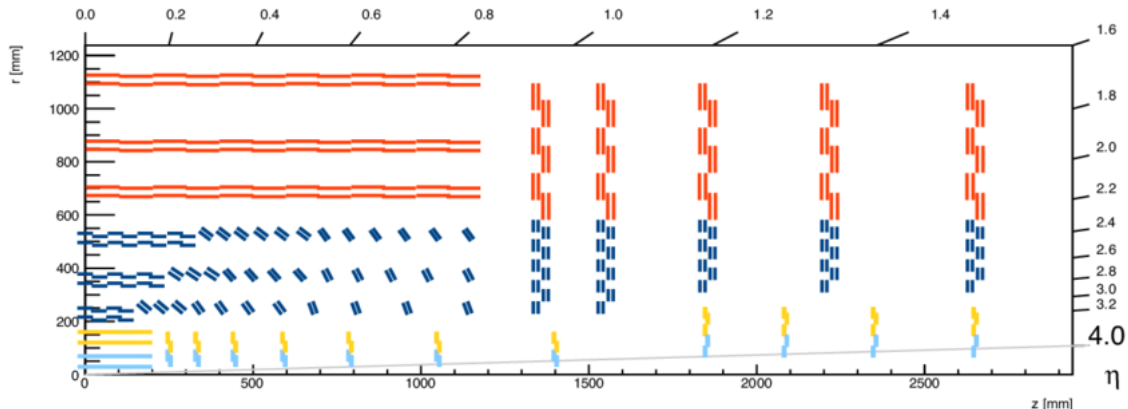


Figure 2.3: One quarter view of the proposed layout of the CMS Phase II tracker in the  $r$ - $z$  plane. The inner tracker is shown in light blue and yellow (modules with two or four readout chips respectively), and the outer tracker is shown in dark blue and red (PS and 2S modules respectively) [34].

is the sub-detector with the highest radiation fluence.

The new tracker will be more radiation tolerant, to survive the  $3000 \text{ fb}^{-1}$  integrated luminosity. In addition, its performance must improve to maintain tracking and vertexing efficiency and resolution in the 200 PU conditions. The granularity will be increased to keep channel occupancy around the per mille level, and hit merging in the pixels will be corrected to improve the distinguishability of two closely separated tracks. Less material will be used by the tracker, to reduce the effect of particle interactions with the tracker material. Finally, the outer tracker will send tracking information to the Level 1 Trigger, the subject of Chapter 4.

The proposed tracker layout is shown in Figure 2.3. Compared to the Phase 1 tracker, the Phase 2 tracker will extend further in  $\eta$ , to  $|\eta| \approx 4$  up from  $|\eta| \approx 2.4$ . This extra coverage will improve the pileup mitigation capabilities, and facilitate object tagging, at higher pseudo-rapidity. Particles will cross at least 9 layers of modules across the whole pseudo-rapidity range, and up to 12 at the highest pseudorapidity.

### 2.4.1 Inner Tracker

The Phase II pixel detector will be required to withstand a radiation dose of 1.2 Grad, and up to  $3 \text{ GHz cm}^{-2}$  hit rate in the innermost layer. At the same time the new detector performance must improve relative to the LHC in order to maintain efficient track reconstruction in the higher pileup environment. The pixel surface area will be a factor 6 smaller than the Phase I detector, with sensors of  $25 \times 100 \mu\text{m}^2$  or



$50 \times 50 \mu\text{m}^2$ . These smaller pixels will improve the resolution on the longitudinal and transverse impact parameters compared to Phase I, improve the separation between track in dense jets, and maintain an occupancy around 0.1%.

Development of the pixel sensor is ongoing. The sensors will be thinner than the Phase I pixels: in the range  $100 \mu\text{m}$  to  $150 \mu\text{m}$  compared to  $270 \mu\text{m}$  to  $285 \mu\text{m}$ . The thinner pixel is protective against radiation damage, with less charge carrier trapping after irradiation compared to a thick sensor. For the innermost layers, which will receive the highest fluence, a 3D sensor with electrodes embedded in the silicon is under investigation. These are potentially less susceptible to charge trapping than planar sensors, although more difficult to fabricate.

A radiation hard pixel readout chip (PROC) is under development, which will use a 65 nm CMOS technology [35]. These will digitise the detected sensor current, store hits for the  $12.5 \mu\text{s}$  trigger latency, and send out hits for triggered events (the inner tracker does not contribute to the trigger). Data transfer will be along electrical connections to Low-power Gigabit Transceivers (LpGBT) on the inner tracker service cylinder, where the signals are then sent optically to the back-end electronics. Modules will host either two or four PROCs, with the two PROC modules occupying the two innermost layers throughout.

The inner tracker DAQ will comprise modular Data, Trigger and Control (DTC) boards, similar to those used for the outer tracker. The DTC boards will interface to the LpGBT optical links, and house an FPGA with large logic capacity for processing and buffering. Optical links operating at  $10 \text{ Gb s}^{-1}$  will send data to the DAQ.

## 2.4.2 Outer Tracker

The outer tracker will have six barrel layers and five endcap disks on each side. Particles will traverse six detector layers up to  $|\eta| \approx 2.4$ , apart from a narrow region around  $|\eta| \approx 1$ , which is the transition between barrel and endcap, where five layers will be crossed. The number of layers is the minimum possible for efficient tracking using only the outer tracker (for the Level 1 Trigger). Track reconstruction can be performed with a high efficiency and low fake rate with only five layers, while an additional layer allows for detector inefficiencies.

The outer tracker will send hits from tracks above a  $p_T$  threshold (called ‘stubs’) to the trigger. The  $p_T$  threshold is applied on the detector, enabled by a  $p_T$  module.

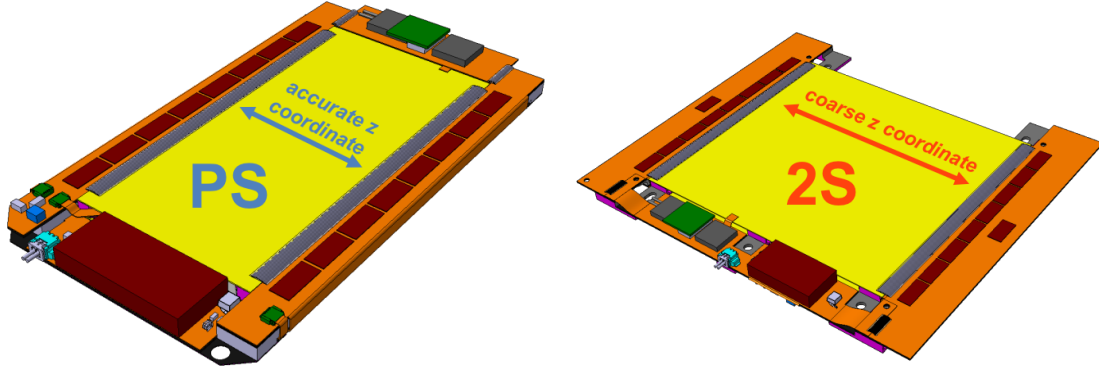


Figure 2.4: The pixel-strip (PS) and strip-strip (2S) implementations (left, right image respectively) of the  $p_T$  module concept [40].

### $p_T$ Module

A key technology enabling the development of a track trigger is the so called ‘ $p_T$  module’ [36, 37]. This module will enable the readout of tracker data at a reduced bandwidth, and reconstruction within the trigger latency budget. The concept begins with the reality that the readout of all tracker hits is not possible, or necessary at the LHC collision rate of 40 MHz. In LHC pp collisions, especially with 200 pileup, most particles in an event are associated with pileup collisions rather than the primary vertex. These pileup particles also tend to have lower  $p_T$  than those particles which are useful for physics analyses. Avoiding the readout and reconstruction of hits from low  $p_T$  tracks greatly reduces the bandwidth required from the tracker to the Level 1 Trigger, and results in fewer hits for the track trigger to process.

The  $p_T$  module exploits the bending of charged particles in the large magnetic field of CMS in order to detect hits associated with high  $p_T$  tracks. Two silicon sensors, segmented in  $\phi$ , the track bending direction, and separated by a few millimetres, make a  $p_T$  module. This separation is wide enough to have a coarse resolution on the track  $p_T$ , while being small enough that the sensors can be read out by the same electronics. Logic on the module clusters pairs of hits on the two layers. The  $\phi$  direction separation of the hits gives a coarse measurement of the track  $p_T$ . A threshold on the hit separation is then used to apply a  $p_T$  cut to tracker hits. Pairs of hits with a  $p_T$  over the threshold are called ‘stubs’. Only these stubs are read out to the Level 1 Trigger. A  $p_T$  threshold of 2-3 GeV is typically considered, providing an order of magnitude rate reduction compared to reading out all hits [38, 39].

Two types of  $p_T$  modules have been developed, shown in Figure 2.4. The first, for use at radii in the range  $200 < r < 600$  mm, is the ‘pixel-strip’ (PS) module. These consist of a silicon pixel sensor beneath a silicon strip sensor. Both layers have a strip pitch of  $100\text{ }\mu\text{m}$  in the  $x - y$  plane. The pixel sensor has a fine granularity of  $1.47$  mm in the  $z$  direction, while the upper layer has a  $z$  granularity of  $23.5$  mm. The fine  $z$  granularity of the lower sensor is necessary for precision vertexing in the Level 1 track reconstruction.

The second  $p_T$  module, for use at radii  $> 600$  mm where the hit occupancy is lower, is the ‘strip-strip’ (2S) module. These have two layers of silicon strip sensors, both with a pitch of  $90\text{ }\mu\text{m}$  in  $x - y$ , and a strip length of  $50.3$  mm in the  $z$  direction. The coarse  $z$  resolution of these sensors adds little to the vertex resolution, but the fine pitch, together with the long lever arm of a tracker extending to around  $1$  m in a  $3.8$  T magnetic field, provides the best  $p_T$  resolution.

Signals are processed by Front End (FE) chips, which find hits in the sensors and correlate measurements on the two layers to find stubs. In the 2S module, the CMS binary chip (CBC) correlates hits from the two silicon layers. In the PS modules, the macro-pixel ASIC (MPA) processes hits in the lower layer, and the strip-sensor ASIC (SSA) processes hits in the upper layer. Signals are sent from the SSA to MPA which performs the hit correlation to form stubs.

These FE chips communicate with a Back End (BE) system, the main component of which is the Data, Trigger and Control board (DTC). This board will send and receive data from up to  $72$  modules, utilising FPGAs with optical communications. The DTC will aggregate stubs from its front end modules, process them, and forward them to the L1 track finder. The processing performed by the DTC will be to convert the stub data received from the FE chip to a global format useful for track finding, and also to carry out any time multiplexing required by the track finder. A separate communication stream will handle DAQ functionality: sending trigger accept signals to the FE chips, and forwarding data to the DAQ system.

Figure 2.5 shows the layout of the outer tracker used for the developments in Chapter 4, called the ‘flat-barrel’ layout. This differs from the layout shown in Figure 2.3, in which some modules in the barrel are tilted towards the luminous region, called the ‘tilted-barrel’ layout. The tilted-barrel layout is the variation preferred by CMS, however was proposed later than the flat-barrel, for which the Monte Carlo event samples used in Chapter 4 were produced. The tilted-barrel results in a much greater stub reconstruction efficiency and lower module occupancy than the flat-barrel, while also requiring fewer modules [40]. The flat-barrel layout

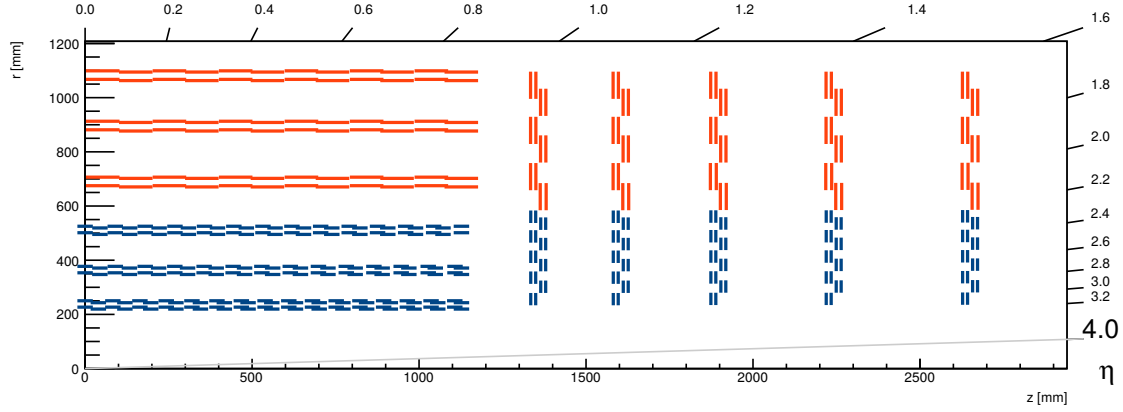


Figure 2.5: Layout of the  $p_T$  modules in the outer tracker used for the track trigger demonstrator, shown for one quarter of the  $r - z$  plane [12]. PS modules are shown in blue, 2S modules are shown in red.

creates an inefficiency at the  $p_T$  module edges, whereby a track crossing the inner layer of one half module, and the upper layer of the other half module cannot form a stub, since the signals are processed in different chips. This issue is eliminated for the tilted modules, which are oriented facing the interaction region.

### 2.4.3 Performance

The expected performance of the Phase II tracker is presented in [34]. The metrics of interest are the efficiency, fake rate and resolution. A track is deemed correctly reconstructed if 75% of its hits are associated with the same simulated charged particle, otherwise it is ‘fake’. Efficiency is defined as the fraction of charged particles in the sample which are correctly reconstructed, while the fake rate is the fraction of reconstructed tracks which are fake. Resolution is the RMS of the residuals of reconstructed parameters and simulated parameter. The efficiency of finding charged particles in  $t\bar{t}$  samples with 140 or 200 PU is around 90% between 1 GeV and 100 GeV. The fake rate is typically below 5% for 140 PU and below 10% for 200 PU, but is best (around 1%) at 2 GeV and worsens above and below this. The resolution of all track parameters improves compared to the Phase I tracker, most notably for the transverse impact parameter, which has a resolution approximately twice as good.

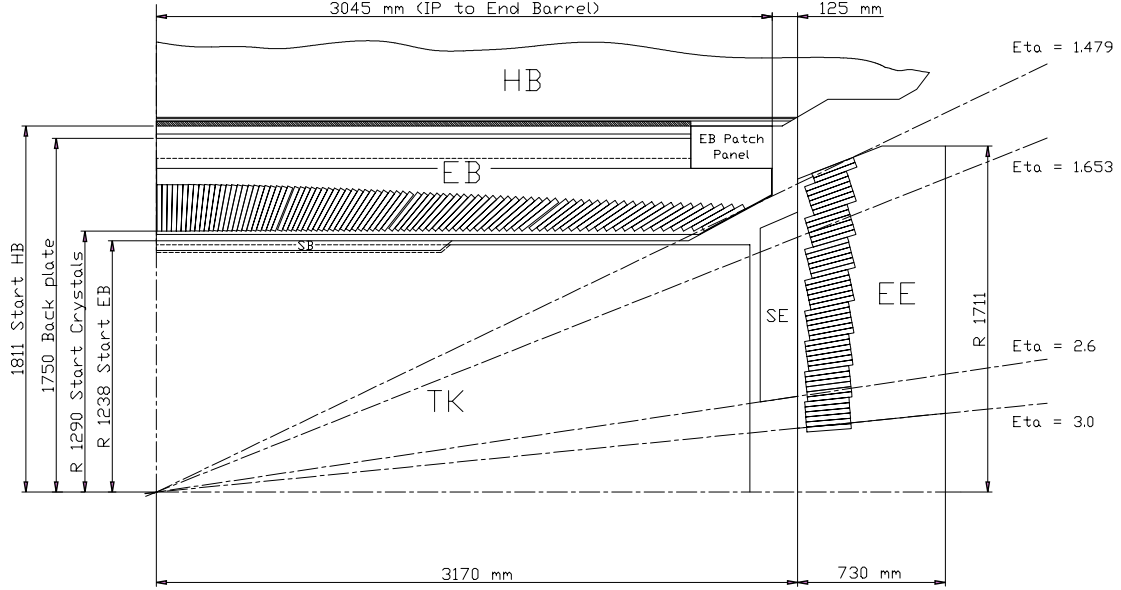


Figure 2.6: One quarter view of the ECAL in the  $r$ - $z$  plane. Depicted are the ECAL barrel (EB), ECAL endcap (EE), preshower endcap (SE). The diagram also shows the HCAL barrel (HB) and tracker (TK) volume [41].

## 2.5 Electromagnetic Calorimeter

The electromagnetic calorimeter (ECAL) measures energy deposited by charged particles producing electromagnetic showers within the material, with photodetectors to collect the energy [41]. The design considerations of the ECAL are largely motivated to provide the best energy and angular resolution for the decay of the Higgs boson to two photons.

The ECAL consists of  $\text{PbWO}_4$  scintillating crystals, arranged as a barrel covering  $|\eta| < 1.479$  with 61200 crystals (in rings of 360 in  $\phi$ ), and two endcaps, extending in the range  $1.479 < |\eta| < 3.0$ , with 10764 crystals each [42]. A schematic of the layout is shown in Figure 2.6. The choice of  $\text{PbWO}_4$  crystals was motivated by the tolerance to radiation, short radiation length of 0.89 cm, small Molière radius of 2.19 cm, and fast response, with 100 ns duration to collect 99% of the light.

Tapered crystals, with a  $3^\circ$  offset, in both  $\eta$  and  $\phi$ , to a straight line from the nominal interaction vertex to each crystal, ensure the ECAL barrel is hermetic. The crystal length of 23 cm in the barrel is 26 radiation lengths, to capture as much of the energy from electromagnetic showers as possible in the ECAL. In the endcap, which is covered by the preshower, the length is 22 cm.

Photodetection is carried out with two types of devices. A high gain must be

used due to the low light yield of the  $\text{PbWO}_4$ , operating in a high magnetic field, and within a high radiation environment. In the barrel, avalanche photodiodes (APDs) are used. In the endcap the much higher radiation dose makes APDs unsuitable, due to the excessive electronic noise this would create. Vacuum phototriodes (VPTs) are used in the endcap instead. From measurements in an electron test beam [43], the relative energy resolution has been determined to be

$$\left(\frac{\sigma}{E}\right)^2 = \left(\frac{2.8\%}{\sqrt{E}}\right)^2 + \left(\frac{12\%}{E}\right)^2 + (0.3\%)^2.$$

### 2.5.1 Input to the Trigger

A 12-bit digitisation is performed on the measured signal, with an upper value of around 2 TeV to accommodate an extreme energy deposition in one crystal, and with the least-significant-bit around the magnitude of the single channel noise. Data are transferred off-detector along high speed optical links individually from each crystal. At the receiving end of the links, in the counting room, the data are summed into trigger towers (TTs) for input into the trigger. Data with crystal level granularity is buffered awaiting the trigger decision.

A view of the grouping of crystals into TTs is shown in Figure 2.7. In the barrel a trigger tower has a size of  $0.087 \times 0.087$  in  $\Delta\eta \times \Delta\phi$  ( $5 \times 5$  crystals), which matches the HCAL granularity as well as the structure of the muon system. Up to  $|\eta| < 2.1$  the endcap TT size matches this, while for  $|\eta| > 2.6$  the size in  $\phi$  is the same, but the size in  $\eta$  is  $\Delta\eta = 2 \times 0.087 = 0.174$ . TTs therefore form rings of 72 in  $\phi$ , with 32 such rings in  $\eta$  in the barrel. For the endcap the mapping to TTs is more complicated. For consistency with the barrel TT angular size of  $5^\circ$  the endcap crystals, which are packed onto a rectilinear grid in  $x$ - $y$ , are assigned to  $5^\circ$  slices which align with the barrel  $\phi$  slices.

## 2.6 Hadronic Calorimeter

The hadronic calorimeter (HCAL) [44] detects particles through their interaction with a dense absorber material via the strong force. This is the only part of the detector which measures neutral hadrons, and is therefore essential for their identification and in determining their contribution to the energy of jets and other quantities.

The HCAL consists of four sections: the HCAL barrel (HB), HCAL endcap (HE),

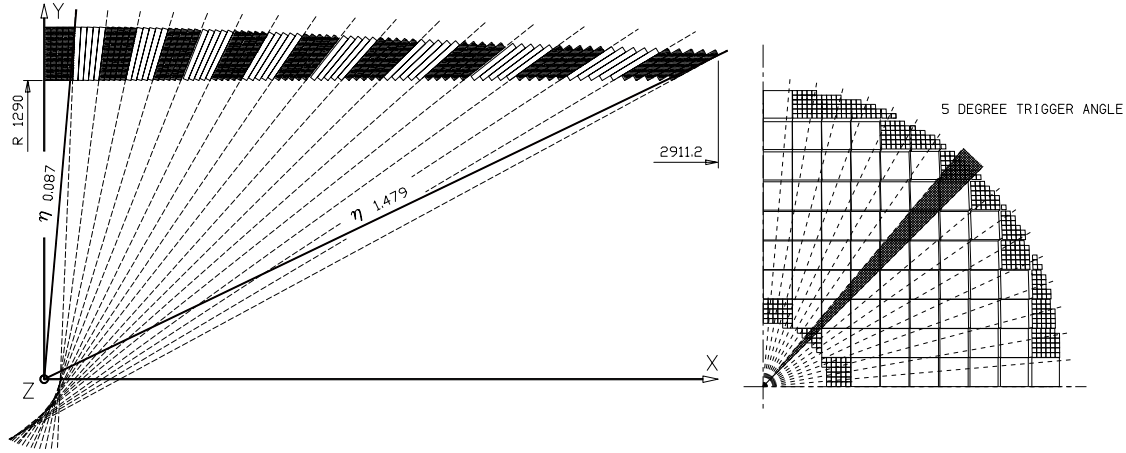


Figure 2.7: View of the grouping of ECAL crystals into trigger towers [41]. On the left is the view of the barrel in one quarter of the  $r$ - $z$  plane. On the right is the view of the endcap in one quarter of the  $x$ - $y$  plane, the small boxes show individual crystals, and the large boxes show modules. Dashed lines from the origin depict the sections of  $5^\circ$  onto which crystals are assigned to trigger towers.

HCAL Outer (HO) and HCAL Forward (HF), a cross section of which is shown in Figure 2.8. The HB and HF are sampling calorimeters with interleaved layers of brass absorber and plastic scintillator active material. Wavelength-shifting fibres transport light from the scintillator into hybrid photodiode (HPD) phototransducers to measure the light amplitude. The HO is located outside of the solenoid, which constitutes the active material along with the steel magnet return yoke. Plastic scintillators, wavelength shifting fibres and HPDs are again used for the detection of light. The HF extends the reach of the HCAL to  $|\eta| = 5$ , and consists of a steel absorber and quartz fibres permeating the steel. Cherenkov light produced in the steel is transported along the fibres into photomultiplier tubes for detection.

Each phototransducer signal is integrated over several 25 ns bunch crossings, and subsequently filtered and digitised, in a front-end ASIC. Optical links carry digitised data off detector into HCAL Trigger and Readout (HTR) cards. The HTR forms the trigger tower data for the HCAL and forwards this to the trigger. This requires summing the measurements along the depth-segmented sections, providing one measurement at each trigger tower location. Data is buffered in the HTR awaiting the Level 1 Accept decision. Accepted events are transmitted into the DAQ system.

The granularity of one HCAL cell matches that of the  $5 \times 5$  grouping of ECAL crystals which form one trigger tower. The lines of constant  $\eta$ , numbered 1 to 29, in Figure 2.8 depict the segmentation of the HCAL into trigger towers. This

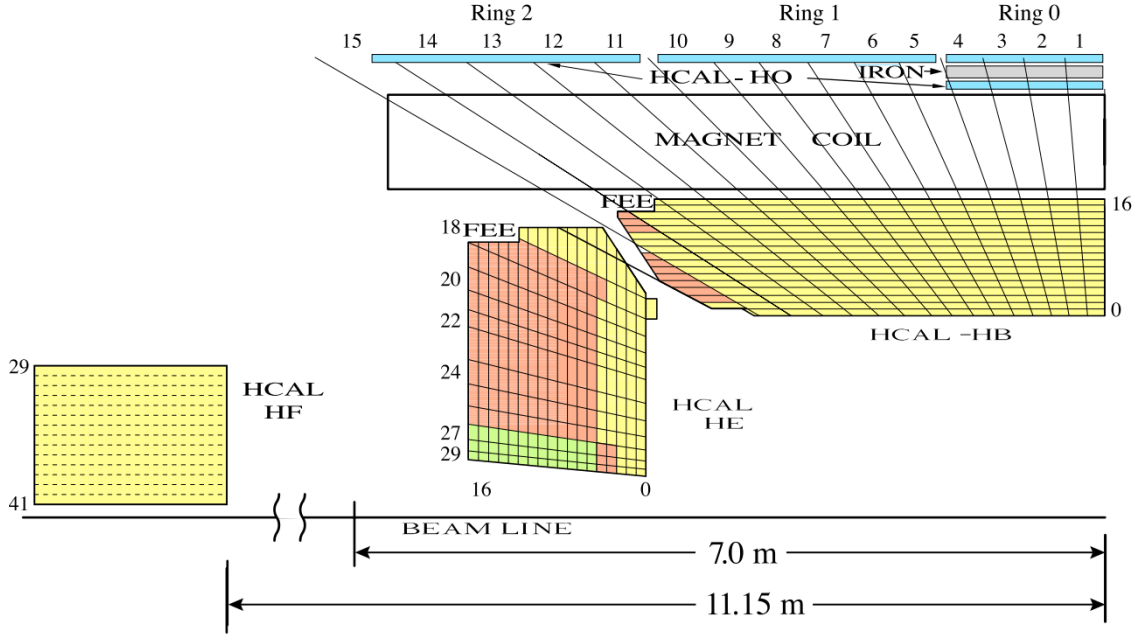


Figure 2.8: A schematic of the HCAL in the  $r$ - $z$  plane [44]. The depth segmentation is pictured, and the segmentation into towers.

segmentation also aligns with the ECAL trigger tower segmentation.

The energy resolution of the HCAL, measured in a test beam [45], was determined to be

$$\left(\frac{\sigma}{E}\right)^2 = \left(\frac{94.3\%}{\sqrt{E}}\right)^2 + (8.4\%)^2.$$

## 2.7 Muon System

The muon system is the outermost part of the CMS detector, designed to detect muons and measure their transverse momentum [46]. All other detectable particles should be stopped in the calorimeters, such that only muons leave signals in the system. Some level of background charged particles do nonetheless enter the muon system, predominating at higher pseudorapidity.

Three different technologies are used for the detection of muons. All of the devices contain a gas which is ionised when traversed by a muon, with an electric field produced by plates or wires to collect the charges and detect the produced current. The gas is segmented into chambers, which facilitates the measurement of muon position. The detectors are arranged such that any single muon will cross several chambers, thus allowing a trajectory to be reconstructed.



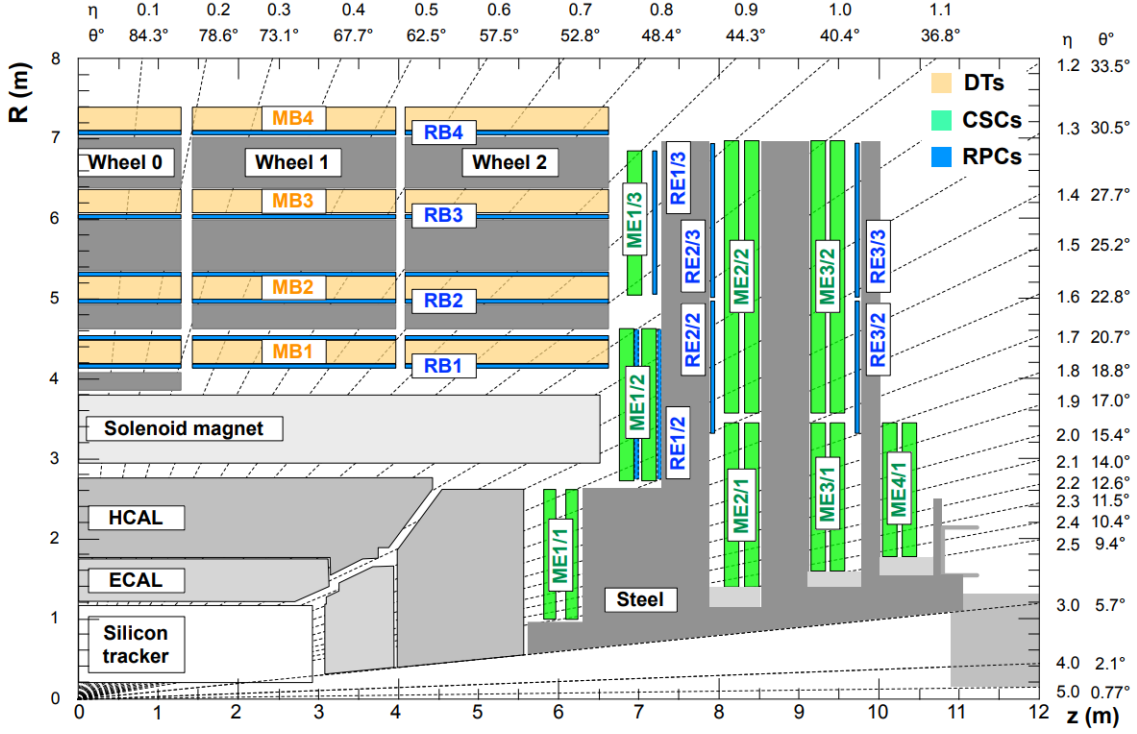


Figure 2.9: Schematic of a one quarter slice of the CMS detector, highlighting the layout of the muon system [47].

The layout of the muon system, with the different detector types highlighted, is displayed in Figure 2.9. In the barrel part of the detector, covering  $|\eta| < 1.3$ , drift tubes (DTs) containing a mix of Ar and CO<sub>2</sub> at atmospheric pressure are used. Cathode strip chambers (CSCs) are used in the endcaps, from  $0.9 < |\eta| < 2.4$ . The technology is more capable than the DTs at providing high spatial and temporal resolution in the presence of higher magnetic field and muon rate in the endcap. Resistive plate chambers (RPCs) are used throughout  $|\eta| < 2.1$  for their fast response, to provide a trigger signal.

Muon identification efficiency is better than 95% for muons with momenta above a few GeV [48]. For muons with  $p_T < 200$  GeV, the silicon tracker provides the better  $p_T$  measurement, which is between 1.3% and 6% from the barrel to endcap. Above  $p_T = 200$  GeV, up to around 1 TeV, the muon and tracker measurements combined provide resolution of around 10% in the barrel region.

## 2.8 Level 1 Trigger

The CMS Level 1 Trigger (L1T) receives detector data at the full 40 MHz collision rate, performs a fast reconstruction of each event, and determines whether the event should be read out from the detector, sending a signal to the front-end buffers when this occurs. L1T is split into separate systems which reconstruct particles within one sub-detector, and feed into a final system which combines that information and makes the trigger decision. Work in this thesis relates to the Phase I calorimeter trigger (Chapter 3) and the Phase II L1T track reconstruction (Chapter 4).

The Phase I Upgrade of the L1T, which fully began operations in March 2016, during LHC Run II, processes measurements from the muon detector and calorimeters to trigger readout of the full detector [49]. A maximum latency limit of  $4\mu\text{s}$  is imposed by the depth of front end buffers, and the readout has a maximum rate of 100 kHz. The design of the upgraded system sought to maintain trigger performance after the increase of LHC collision centre of mass energy from 8 TeV to 13 TeV and instantaneous luminosities yielding a mean pileup of 50, up from the 20 experienced during Run I. In particular, pileup mitigation was required to avoid saturating the maximum 100 kHz trigger rate with acceptably low thresholds. For example, a 20 GeV single electron trigger would consume half of the available bandwidth using the Run I system [50].

The system architecture is shown in Figure 2.10. FPGAs are used throughout for processing. For the calorimeter, trigger tower data is sent into a two layer FPGA processing system, described in more detail in section 2.8.2. Energy deposits consistent with jets,  $e/\gamma$ , or  $\tau$  particles are clustered. The total event energy and transverse energy is also found. The muon trigger combines hits from the three different types of muon detector to find muon tracks and measure the  $p_T$ . The muon and calorimeter trigger systems send their reconstructed objects to the Global Trigger (GT). The GT performs the event selection, based on comparing the input objects to a ‘menu’ of criteria on which to select an event. Each item on the menu will specify one or more object type (jet,  $e/\gamma$ ,  $\tau$ ,  $\mu$ ),  $E_T$  thresholds, and quantities such as angular separation.

### 2.8.1 Differences for Phase II

The Phase II L1T upgrade will take place ahead of the HL-LHC [51]. The entire trigger and DAQ system will be replaced and upgraded. A longer latency of  $12.5\mu\text{s}$  (including contingency) will be allowed, and a maximum L1T accept rate of 750 kHz

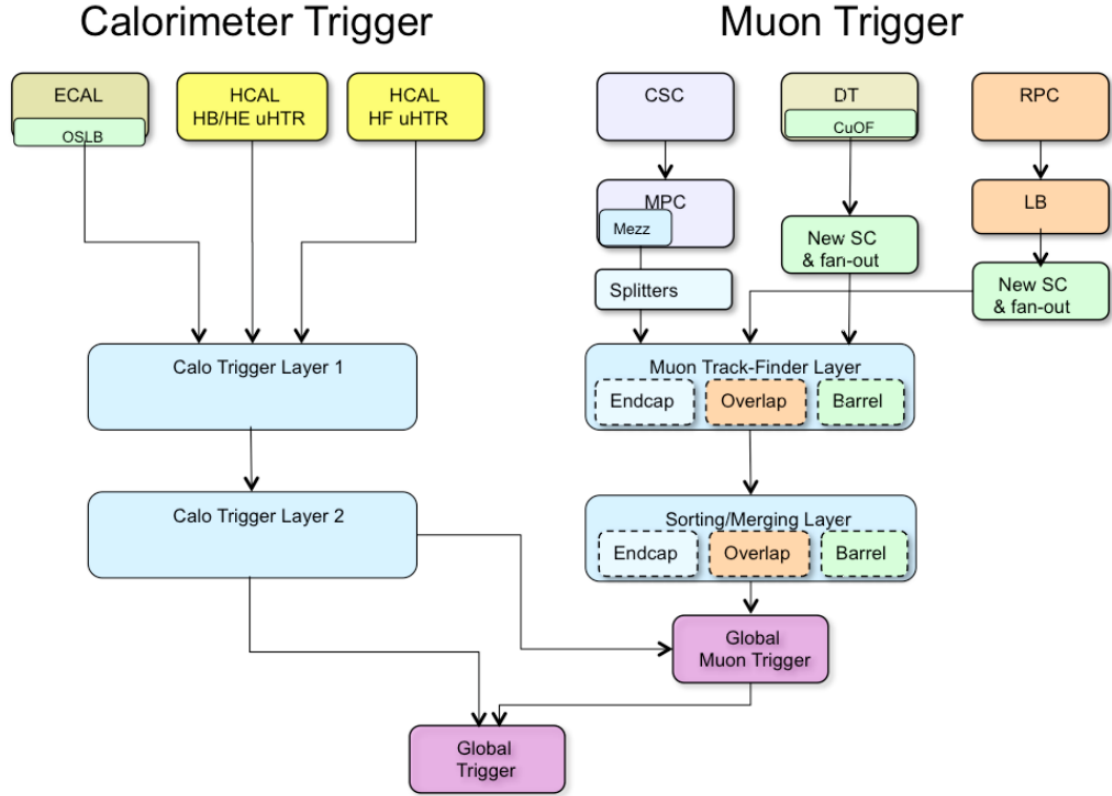


Figure 2.10: Diagram of components of the Phase I Upgrade of the CMS Level 1 Trigger [49].

will be permitted. The functionality of the calorimeter and muon processing systems will change to accommodate changes to the detectors. In particular, the processing of the new endcap calorimeter [52] will necessarily be very different from the processing for Phase I, due to the significantly increased detector granularity, and the addition of depth information. For the first time CMS will also reconstruct tracks at the L1T. Chapter 4 describes a demonstrator system for the track reconstruction.

The planned overall structure of the Phase II system is shown in Figure 2.11, and is conceptually similar to the Phase I system. Each sub-detector will be served by a processing system which reconstructs the raw signals into composite objects: clustered energy deposits, tracks, and muons. These will forward those objects to a system which combines them and makes the trigger decision. The task of combining trigger primitive objects will become more difficult than for Phase I, because the addition of tracks will facilitate discrimination of neutral from charged hadrons, electrons from photons, and provide a more precise measurement of muon  $p_T$  than the muon detectors provide. In addition, the primary vertex can be identified from the tracks. The processing to combine the objects from the various detector systems

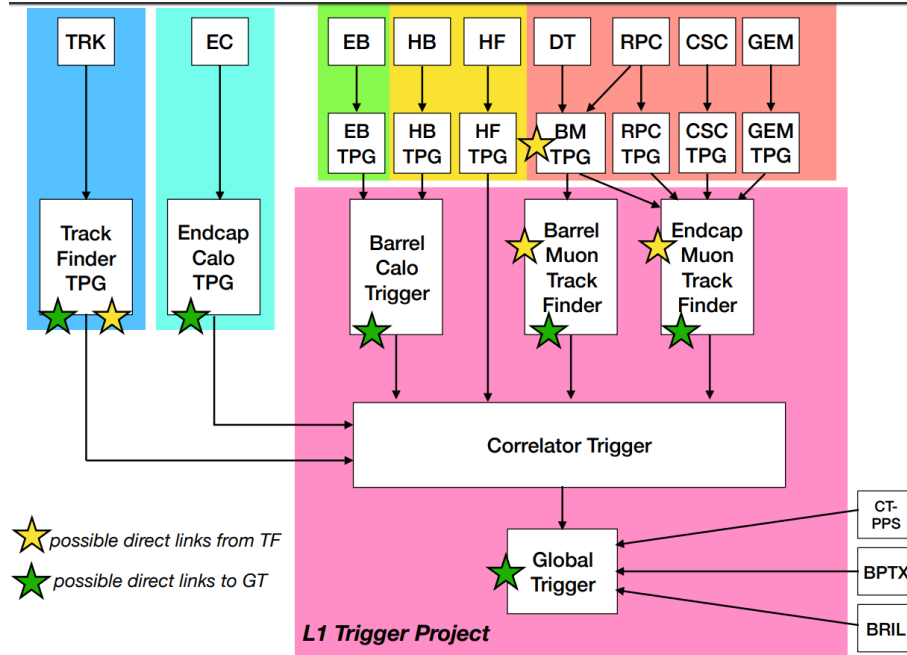


Figure 2.11: Overview of the components of the Phase II Level 1 Trigger [51].

will be a modified version of the Particle-Flow algorithm [14], used at the HLT and offline. Pileup subtraction will also be carried out using the PUPPI algorithm [32]. This system will be called the Correlator Trigger. Particle candidate objects will be sent from the Correlator to the Global Trigger, which will perform the same role as it currently does.

### Impact of tracks

The tracks reconstructed in the L1T will enable triggers with sharper turn on curves and lower rate for the same efficiency [12]. A single muon trigger with a 20 GeV threshold on the reconstructed  $p_T$  displays a turn on curve which ramps from around 10 GeV to 30 GeV on the simulated muon  $p_T$ , when using only the muon system. With the addition of tracking information, assigning the muon  $p_T$  from a matched Level 1 track, the width of the ramp up is reduced to around 2 GeV, due to the superior resolution of the tracks at this momentum. As a result of the much sharper turn on curve, the rate of the trigger is reduced by a factor of 10. Similarly, the rate of a single electron trigger with a 20 GeV threshold is reduced by a factor 5 when tracks are matched to  $e/\gamma$  deposits in the calorimeter, compared to using the calorimeter alone.

The rate of multi-object triggers, for example dimuon or dijet, can be reduced

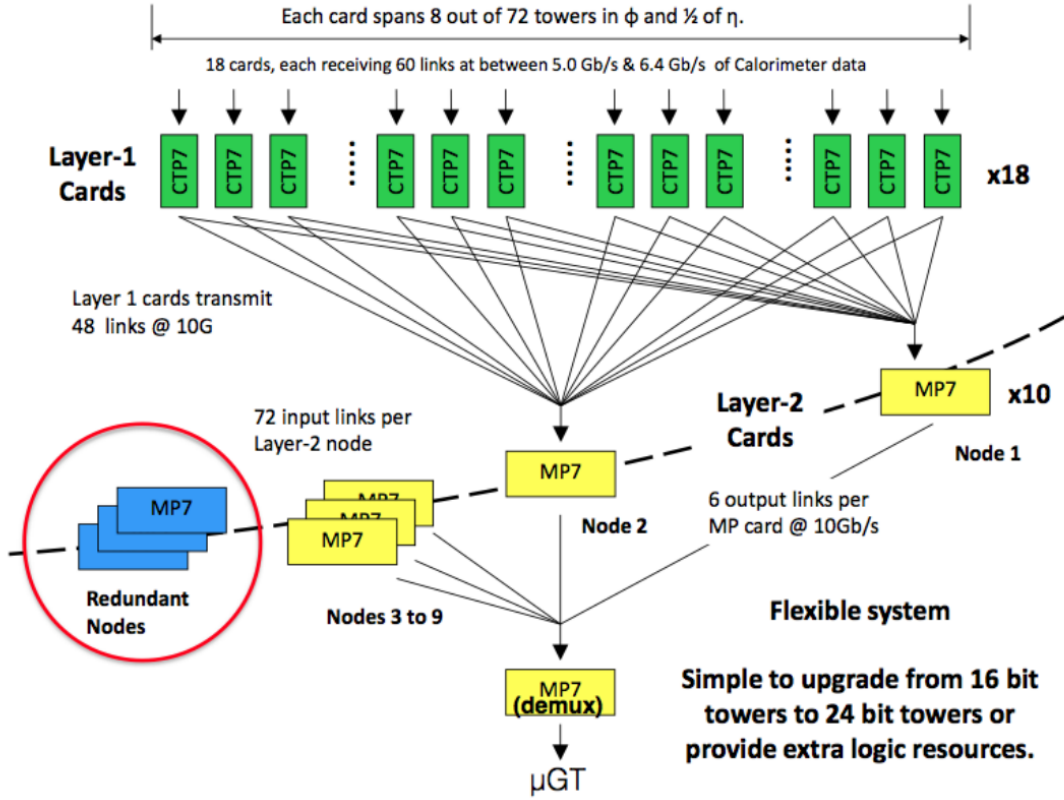


Figure 2.12: System architecture of the Level-1 Calorimeter Trigger [50].

by reconstructing the primary event vertex and imposing that both objects are associated with that vertex. A dimuon trigger rate can be reduced by a factor three by requiring that the muons are consistent with the vertex to within 1 cm. The efficiency to reconstruct the vertex within 5 mm of the generated vertex is 97% in  $t\bar{t}$  events, or 90% to reconstruct it within 1 mm.

### 2.8.2 Phase I Calorimeter Trigger

The system architecture of the Level-1 Calorimeter Trigger is shown in Figure 2.12. A time multiplexed architecture is used. Time multiplexing is a technique which allows data for the entire detector for one event to be processed in a single processing node (in this case, one board). Since no board possesses the bandwidth to receive all of this data in one bunch crossing (around  $2.5 \text{ Tb s}^{-1}$ ), the data must be spread out over a longer period of time<sup>†</sup>. A two layer architecture is required to perform time

<sup>†</sup>An alternative architecture would split the processing geometrically, such that each processing board would receive data at the full collision rate. Drawbacks of such an approach are that data

multiplexing. In the first layer, which consists of 18 CTP7 boards [53], each board must receive data at the full detector rate, and forward it at the same rate to a node in the second layer. Time multiplexing is carried out by switching the destination node at each bunch crossing. A node in the second layer receives data from each board in the first layer in turn. Each node sends and receives data continuously to keep up with the collision rate. An optical patch panel facilitates the routing between nodes in a compact form factor. A time multiplexing factor of nine is used, meaning that the data from one bunch crossing are spread out over nine bunch crossings to the second layer.

The second layer in the system executes the algorithms to construct particle-like objects from the trigger tower input, and is implemented with nine MP7 boards [54, 55]. The firmware running on each node in the second layer is identical. On each 240 MHz clock cycle, the algorithm receives one complete  $2\pi$  ring in  $\phi$  (72 trigger towers) from each half of the detector, beginning from the middle and progressing outwards on subsequent cycles. From the trigger towers, the processors reconstruct jets,  $e/\gamma$  and  $\tau$  candidates, in addition to finding the total event energy, and transverse missing energy. The performance of these algorithms can be found in [56].

## 2.9 High Level Trigger

The HLT is responsible for the final trigger decision before an event is stored to disk. During Run I and II the HLT was a farm of CPUs situated on the surface directly above the CMS detector, consisting of approximately 1,000 compute nodes. Software running on the nodes execute several ‘trigger paths’, which require reconstruction of subdetector signals into particle hypotheses, and kinematic criteria on the particles which determine whether a path succeeds or fails. The software running at the HLT farm is a modified version of the CMS software, CMSSW [57], optimised to achieve a low latency for creating a trigger decision. Events are dynamically allocated to nodes in the farm by a high speed network.

Processing is latency constrained, with a mean time per event limited by the input rate from the L1T and the number of nodes. The full detector granularity is available, and the reconstruction is a lightweight version of the offline software. The reconstruction is seeded by the Level 1 objects, which saves some processing effort.

The time taken to process an event at the HLT varies depending on the event

---

sharing becomes necessary to cover boundaries between regions, and that the loss of a processing node effects one part of the detector for every event.

content. The software is structured in such a way that events are rejected using the fastest to execute paths first, using the calorimeters and muon chambers, before running track reconstruction. Each trigger path comprises a series of sequential reconstruction modules. Each module is followed by a filter step, which checks the output of the module. In the case that any filter fails to meet its criteria, the entire paths fails and its processing is terminated. Given the 100 kHz L1T accept rate, and the number of nodes available, a latency budget can be derived. In Run I and II this was around 200 ms. This budget is the maximum average processing time per event before the farm would become overloaded.

Detailed timing measurements of the HLT reconstruction are presented in [58]. With Run II conditions of  $\sqrt{s} = 13$  TeV and 40 PU collisions, the mean HLT processing time per event was predicted to be 162 ms. A much lower mean processing time of 66.5 ms was predicted for 20 PU conditions, despite the most probable time remaining similar at around 40 ms. The difference arises from the much longer tail of event time with 40 PU, due to the extra combinations encountered during tracking that arise with higher pileup.

At the HL-LHC, the CMS HLT will face the challenge of maintaining low latency trigger decisions in the high pileup environment. The impact of high pileup on track reconstruction is discussed in section 2.10. At the same time, CPU power will continue to improve in the time before the HL-LHC, although the benefit from clock frequency scaling is slowing, with most improvements coming from higher core count and additional vector units. Efforts to parallelise CMSSW beyond the event level show reduced event reconstruction times [59].

In addition to the direct impact on computation time from higher pileup, the L1T event accept rate will increase from 100 kHz to 750 kHz at the HL-LHC. Combining an anticipated scaling of event reconstruction time due to the increased number and occupancy of detector channels; the increased L1T accept rate; and an anticipated saving from utilisation of information from L1 tracking, it is projected that the HLT for 200 PU events will need to be 22 times more powerful than the Run II HLT (11.0 MHS06<sup>†</sup> up from 0.5 MHS06) [60]. The Run II HLT comprises 940 nodes, and under two different CPU performance scaling scenarios the HL-LHC HLT would require between 1400 and 7800 nodes.

---

<sup>†</sup>HepSpec06 (HS06) is a HEP specific CPU benchmark.

## 2.10 Track Reconstruction

Track reconstruction, which is the focus of Chapters 4 and 5, is the procedure by which the trajectory parameters of charged particles are determined. In the solenoidal magnetic field, charged particles follow helical trajectories. The tracking detector measures hits, that is position measurements, at each intersection of a charged particle with a detector plane. The detector, depicted in Figure 2.3, samples the trajectory on around 10 different surfaces. Reconstructing tracks requires the identification of hits which lie on the same trajectory, referred to as track building or pattern recognition, followed by a fit to the hits to obtain the parameters of the trajectory.

The tracking performed at CMS, both offline and at the HLT, consists of four steps, repeated iteratively to find all charged particle trajectories, and is termed the Combinatorial Track Finder [61]. These steps are:

- Seeding: Initial estimates of track trajectories are obtained from combinations of hits in the pixel detector.
- Building: Track seeds are propagated, searching for compatible hits and updating the trajectory estimate using a Kalman Filter.
- Fitting: Found tracks are fitted with a Kalman Filter and smoother.
- Selection: Tracks are checked against quality criteria to identify fake tracks.

The procedure is repeated iteratively, each iteration targetting a specific class of tracks by the adjustment of cut parameters. Following each iteration, any hits contributing to finished tracks are ignored in subsequent iterations. Earlier iterations find the easiest to reconstruct tracks, such as those with a high  $p_T$ , many hits, and originating from near the interaction region. Later iterations find more difficult classes of tracks, such as those with low  $p_T$  and displaced from the beam line, the task made easier following the removal of already utilised hits.

One of the optimisations of the tracking at HLT is to reduce the number of iterations of tracking compared to the offline reconstruction [62]. The first iteration reconstructs prompt tracks from the highest quality seeds (with hits on 3 pixel layers), and finds approximately 80% of tracks. The second iteration finds low  $p_T$  prompt tracks, again from pixel triplet seeds. The third iteration finds prompt tracks from pixel seeds with hits on only 2 layers. The final iteration finds displaced tracks.



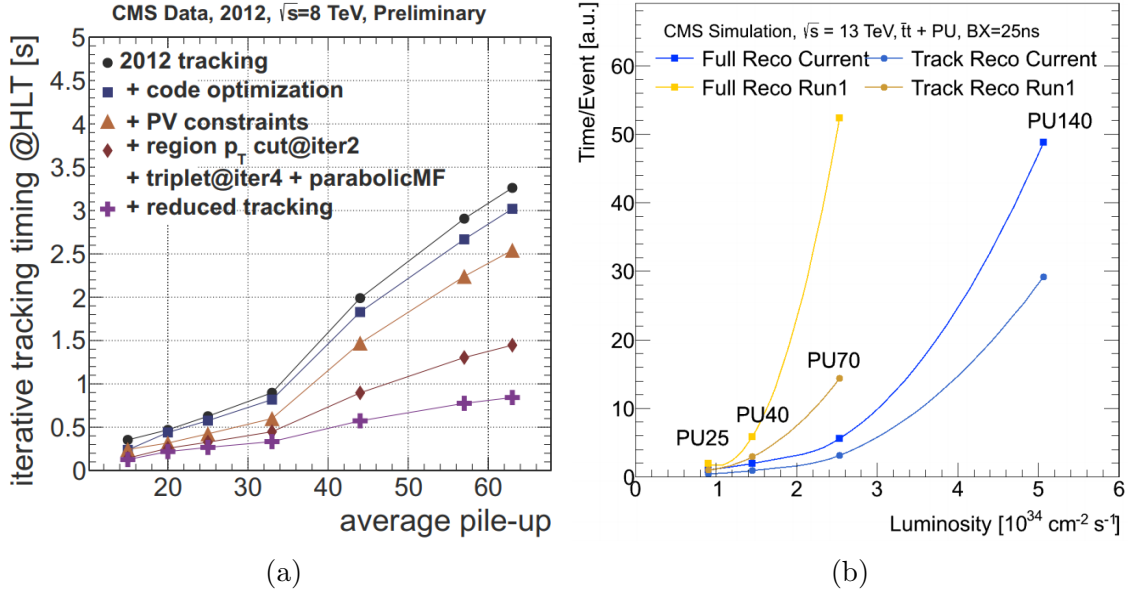


Figure 2.13: The timing of track reconstruction at CMS, with several improvements for speed enhancement at the HLT (left) [62], and for the full reconstruction as a function of instantaneous luminosity (right) for the Run I software in yellow and Run II software in blue [63].

Figure 2.13a shows the timing of track reconstruction at the HLT during Run I, and for a series of improvements made for Run II. Even with the fastest configuration, the tracking reaches the total HLT latency budget at a PU of 20, and exceeds it by a factor 2.5 at 40 PU [62]. This necessitates the spared usage of the track reconstruction, only following calorimeter and muon algorithms. Figure 2.13b shows the scaling of the timing of reconstruction with instantaneous luminosity. For the Run II software the tracking takes more than half of the total event reconstruction time at all pileup scenarios [63]. The time to reconstruct tracks at 140 PU is approximately ten times greater than the time taken with 70 PU, with the same software. For events of  $t\bar{t}$ , and  $t\bar{t}$  with 8 PU, 64% of the CPU time of offline tracking is spent on track building, with approximately 15% on each of seeding and fitting [61].

### 2.10.1 Seeding

The seeding step finds initial estimates of the track parameters and the associated uncertainties from just a few hits. Usually this uses only hits from the pixel detector, which has the best position resolution and lowest hit occupancy. Seeds must be formed either from three pixel hits, or two hits and an assumption that the track passes through the beamline. Seeds with three hits have a lower fake rate than those

with only two hits. At each iteration of the combinatorial track finder, a tracking region is defined in which seeds are constrained, based on limits on the distance of closest approach to the primary vertex and minimum  $p_T$ .

### 2.10.2 Track Building

A Kalman Filter [64, 65] is used for both the track building and fitting. The filter is a ‘local’ fitter: an estimate of the parameters of interest is updated with each measurement in turn. This attribute makes the technique suitable for track building, since multiple measurements on the same detector layer (of which only one can lie on any track) can be considered independently, without affecting each other. Compared to a ‘global’ fit, such as a linearised  $\chi^2$  technique, this ensures that measurements not belonging to a track cannot pull the fit parameters. A second advantage is that effects such as multiple scattering can be included in the fit more easily.

The Kalman Filter operates on a ‘state’, which consists of the vector of track parameters  $x$ , and their covariance matrix  $\mathbf{C}$ . The equations of the Kalman Filter state update are given by Equations 2.4 to 2.12. The index  $k$  refers to the iteration. Since the measurements are included in order, this also corresponds to the detector layers.

$$x_k^{k-1} = \mathbf{F}_{k-1} x_{k-1} \quad (2.4)$$

$$\mathbf{C}_k^{k-1} = \mathbf{F}_{k-1} \mathbf{C}_{k-1} \mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \quad (2.5)$$

$$r_k^{k-1} = m_k - \mathbf{H}_k x_k^{k-1} \quad (2.6)$$

$$\mathbf{R}_k^{k-1} = \mathbf{V}_k + \mathbf{H}_k \mathbf{C}_k^{k-1} \mathbf{H}_k^T \quad (2.7)$$

$$\mathbf{K}_k = \mathbf{C}_k^{k-1} \mathbf{H}_k^T (\mathbf{R}_k^{k-1})^{-1} \quad (2.8)$$

$$x_k = x_k^{k-1} + \mathbf{K}_k r_k^{k-1} \quad (2.9)$$

$$\mathbf{C}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{C}_k^{k-1} \quad (2.10)$$

$$\chi_+^2 = r_k^{k-1T} (\mathbf{R}_k^{k-1})^{-1} r_k^{k-1} \quad (2.11)$$

$$\chi_k^2 = \chi_{k-1}^2 + \chi_+^2 \quad (2.12)$$

These can be considered in two steps: the projection of the state to the next layer, and the update of the state with a measurement on that layer. Projection is carried out by Equations 2.4 to 2.5. The state is propagated from the previous layer,  $k-1$ , to the current layer  $k$ . Multiple scattering, which alters the path of the

track, is included in the matrix  $\mathbf{Q}$ .

The measurement  $m$ , with covariance matrix  $\mathbf{V}$  is used to adjust the parameters. This predicted state is updated in Equation 2.9 by the residual between the projection and the measurement, weighted by the Kalman gain  $\mathbf{K}$ . The Kalman gain weights the adjustment of the state according to the uncertainties and correlations in the state and measurement. In simplistic terms, a small uncertainty in the state compared to a large uncertainty in the measurement will change the state only a little, whereas a large uncertainty in the state with a small measurement uncertainty will result in the state pulling towards the measurement.

The track reconstruction performed at CMS, both offline and at the HLT, uses a 5 parameter state in the curvilinear frame to describe a track at any point along its trajectory:

$$x = (q/p, \lambda, \phi, x_{\perp}, y_{\perp}), \quad (2.13)$$

where  $q$  is the sign of the particle charge,  $p$  the magnitude of the momentum,  $\lambda$  the dip angle and  $\phi$  the azimuthal angle, both in the global reference frame [66]. The coordinates  $x_{\perp}$  and  $y_{\perp}$  are defined in a local coordinate system to the track given by  $\mathbf{T}$ , a unit vector pointing parallel to the track, and the orthogonal vectors  $\mathbf{U}$  and  $\mathbf{V}$  defined by:

$$\mathbf{U} = \frac{\mathbf{Z} \times \mathbf{T}}{|\mathbf{Z} \times \mathbf{T}|}, \quad (2.14)$$

$$\mathbf{V} = \mathbf{T} \times \mathbf{U}, \quad (2.15)$$

where  $\mathbf{Z}$  is a unit vector pointing along the global  $z$ -axis. The vector  $\mathbf{T}$  points along  $z_{\perp}$  by definition,  $\mathbf{U}$  points along  $x_{\perp}$  – lying in the global  $xy$ -plane – and  $\mathbf{V}$  points along  $y_{\perp}$  at an orthogonal to the above such that a right-handed Cartesian coordinate system is formed.

Since the propagation of the track parameters is non-linear in the parameters, the state propagation is modified slightly from Equations 2.4 to 2.5. The state vector is simply propagated according to the track helix equations

$$x_k^{k-1} = f_{k-1}(x_{k-1}).$$

The covariance matrix propagation is performed with the Jacobian, the matrix of

first order partial derivatives of  $f$  with respect to the track parameters:

$$\mathbf{F}_k = \left. \frac{\partial f}{\partial x} \right|_{k-1},$$

$$\mathbf{C}_k^{k-1} = \mathbf{F}_k \mathbf{C}_k \mathbf{F}_k^T + \mathbf{Q}_k.$$

The state is constructed such that the measurement equation, Equation 2.7 is linear in the state.

In the track building stage, the hits which belong to a track are not known. Beginning with a seed track, the state is propagated to the next detector layer according to the track helix equation. A search is then performed in all of the hits measured on the detector layer, and any within a suitable  $\chi^2$  are kept. Each of these hits (there may be none, one, or more than one) is filtered with the state update equations separately. These states are now independent track candidates, and the procedure continues: propagation, hit searching, updating. States which have been filtered with a random collection of hits will eventually not find any hits within the  $\chi^2$  window, and can be rejected. Similarly, tracks which fall outside of the  $p_T$  or vertex constraints of the tracking iteration are rejected. For the purposes of the HLT, the track parameter uncertainty is good enough for a track with 8 hits that the building can be stopped in order to save processing time, rather than continuing to the outermost detector layer. This track building procedure, while powerful for finding all tracks, leads to the combinatorial problem that slows the tracking execution time with increasing pileup. With more tracks, and hence more hits, there are more combinations of hits that need to be tried in order to find all of the tracks.

A cleaning step is performed to remove tracks which are reconstructed more than once, which may occur when the same hits are added to different, similar, seed tracks, or when a single seed develops into multiple viable tracks. The hits on each pair of tracks are compared, and where the fraction of shared hits exceeds 19%, the track with the most hits, then the smallest  $\chi^2$  is kept.

### 2.10.3 Track Fitting

After the track building step a collection of trajectories corresponding to the Kalman Filter states at the outermost detector layer is obtained. In order to obtain the best estimate of the track parameters at the location of every hit, the hits are refit and smoothed. The Kalman Filter state is initialised at the innermost hit, with

inflated uncertainties in the covariance matrix to remove bias towards the original result from the building step. The procedure of propagation and update is repeated, simplified now that at most only one hit on each layer remains in the track candidate. A Runge-Kutta method is used to propagate the state between layers, in order to take into account material between layers, and magnetic field inhomogeneities. At the outermost hit, the uncertainties are again increased by a large factor and the propagate-update procedure is carried out in the reverse direction. At each measurement the weighted average of the inside-out and outside-in state is taken as the best measurement. Now with the smallest uncertainty state estimate at every layer, outlier hits can be identified and rejected with better accuracy. A  $\chi^2$  cut is used to identify outlier hits, with a tighter cut than used during building. If any hits were removed from a track, the fit is repeated until no more hits are removed, or the track has hits removed from two consecutive layers, in which case it is discarded. The state at the outermost hit can then be propagated to the calorimeter or muon subsystems, while the state at the innermost hit can be propagated towards the beamline to find the distance of closest approach.

#### 2.10.4 Track Selection

The final step of an iteration of the track reconstruction is to select good quality tracks, with the aim of rejecting fake tracks. Selection is made based on the number of hits on the track, the  $\chi^2$  per degree of freedom of the fit result, the compatibility with a primary vertex, and the  $\eta$  and  $p_T$  of the track. Different selection criteria are applied for each tracking iteration, as well as ‘loose’, ‘tight’, and ‘high-purity’ working points [61].

# Chapter 3

## MaxCompiler for Level 1 Trigger Applications

High-level programming languages for FPGAs enable the development of more advanced algorithms for the Phase II upgrades of the CMS trigger, and other HL-LHC experiment triggers. These languages abstract some of the implementation details from the developer, allowing more effort to be spent on optimising algorithms. Whether high level languages can match the performance (in FPGA resources and latency) of expert, hand-written HDL has been an issue within the community preventing more widespread adoption. Inefficiencies with a compiler that create overly large and slow implementations would negate the benefits of being able to realise complex trigger algorithms. In this chapter the MaxCompiler tool is benchmarked against VHDL, using an algorithm originally developed in VHDL for the CMS Phase I Calorimeter Trigger upgrade, and subsequently reimplemented using MaxCompiler.

### 3.1 High Level FPGA Programming

Each of the two major FPGA vendors, Xilinx and Altera, offer their own proprietary high level tools. Xilinx supports Vivado High-Level Synthesis (HLS) [67], which allows FPGA designs to be implemented with C, C++ and System C. The tool significantly abstracts implementation details from the developer. Much of the mapping of code to hardware relies on analysis of loop dependencies and algorithm memory usage within the code. The compiler can be ‘guided’ with directives and pragma statements to achieve a desired implementation.

Vivado HLS has been investigated for use in triggers for LHC experiments, and used in the development of new systems for the Run 3 and HL-LHC upgrades. For example the tool was used to redevelop some parts of the CMS Endcap Muon Track Finder (EMTF) firmware by way of comparison with handwritten Verilog [68]. Some parts of the design were implemented with lower resource usage with Vivado HLS than the original. Achieving low latencies with the design required careful guiding of the compiler with specific code constructs and compiler directives. The system has a relatively low clock frequency of 40 MHz. A similar exercise in translating an existing design to Vivado HLS, implementing a Finite Impulse Response filter for the CMS ECAL Data Concentrator Card resulted in a design with 60% longer latency than the expectation [69].

An advantage of Vivado HLS is the support for integration of C-based code within other software. Generally performance studies of trigger algorithms are carried out using a custom made software emulator which can run on conventional processors such as in [70]. This requires a significant development effort to achieve results which accurately match the HDL implementation used in the trigger. The ability to utilise the same code in both FPGA and CPU platform implementations can lift this development burden. It was found in [68] that the execution time on a CPU of the HLS algorithm was a factor two slower than a hand made C++ emulation, but with none of the development overhead.

Other attempts to move away from HDL programming have involved *ad hoc* developments of tools. For example the Tracklet group of the CMS Level 1 Track Trigger developed with a mix of Verilog and Python [71]. Expert developers created core Verilog modules by hand, and described their connectivity with a Python model of the design. This approach can save the work of specifying component connections in HDL, for a design with high reuse of the handwritten modules, but without the portability and flexibility of a fully fledged HLS tool.

Menus for the CMS Global Trigger in Run I were created from a set of VHDL templates [72], a concept that was extended and refined to a custom software tool for the 2016 Global Trigger upgrade ( $\mu$ GT) [73]. Compared to the preceding Global Trigger,  $\mu$ GT allows more sophisticated, analysis-like, combinations of trigger primitives to be used to make the trigger decision, such as the mass of a pair of objects. A grammar is defined for describing combinations of trigger objects, functions to apply on the objects, and cuts which together comprise a trigger algorithm. The grammar is parsed by the Boost.Spirit C++ library, producing a VHDL implementation of the specific menu which can be synthesised by FPGA vendor tools. This approach

totally decouples trigger menu development from hardware implementation, which is useful for allowing physicists with no FPGA expertise to develop such menus. A limited number of object combining functions are made available, which restricts the algorithm developer to operations known to be practical in the resource and latency constraints.

### 3.1.1 MaxCompiler

MaxCompiler [74] and MaxJ are the compiler and language produced by Maxeler Technologies for the development of algorithms for FPGAs, with a focus on dataflow. The MaxJ language is an extension of Java, variables are references to locations in a data stream. Execution of the code (on a CPU) generates a dataflow graph, an example of which is shown in Figure 3.1. Each node of the graph represents a computation, with inputs and outputs corresponding to data. An implicit ‘for loop’ surrounds the dataflow graph, since presenting new data at the input to the graph generates new results. Simply by ‘pushing’ data through the graph its function is executed multiple times.

MaxCompiler optimises the design graph, for example propagating operations on constants to reduce unnecessary computation in the FPGA. Each node, or cluster of nodes, is then represented with a Register Transfer Level (RTL) HDL description or vendor IP core (such as multipliers and memories), and connected according to the graph. The design can then be synthesised to a bitstream for the target FPGA. In this sense MaxJ is more like a ‘High Level HDL’ than HLS, since each operation maps to a logical component in the FPGA.

Normally, MaxCompiler designs target ‘Dataflow Engine’ (DFE) boards produced by Maxeler. These are typically PCIe form factor boards with an FPGA and DDR memory for performing compute acceleration. Boards with both Altera and Xilinx components are available. In this scenario, MaxCompiler runs the vendor synthesis tools ‘under the hood’, generating a ‘.max’ file that contains the compiled design (bitstream) and information used by other pieces of Maxeler software to run the design in their hardware.

The compute acceleration boards manufactured by Maxeler do not provide the optical IO bandwidth required by LHC Level 1 Trigger systems – very few other setups could produce data at multiple Tb/s. Working with Maxeler, provision was made to halt the MaxCompiler process at an intermediate stage, providing access to the HDL of the design. With this access it was possible to utilise designs written



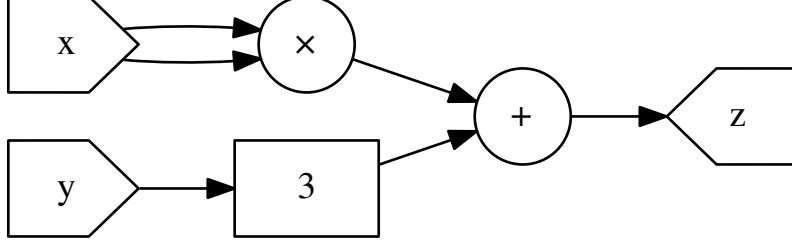


Figure 3.1: Dataflow graph for  $z = x^2 + y$ . Variables  $x$  and  $y$  are inputs,  $z$  is an output. The box labelled ‘3’ represents a FIFO of depth 3, which is the latency of the multiplication in clock cycles.

using MaxJ in other hardware, with the optical bandwidth needed, and therefore to explore the use of the language for trigger applications.

## 3.2 Jets and Energy Sums

Of the reconstruction performed by the Layer 2 processor, the jet and energy sum algorithms were chosen to be reimplemented with MaxCompiler. The energy sum is the most simple quantity reconstructed, while the jet algorithm builds on some elements of the energy sum, with additional processing. The hardware description language VHDL was used for the implementation deployed in the trigger at CMS.

The energy sum is simply the sum of  $E_T$  of all trigger towers:

$$E_{T,Event} = \sum_{i_\phi, j_\eta} E_{T,i_\phi j_\eta}, \quad (3.1)$$

where  $E_{T,i_\phi j_\eta}$  is the transverse energy of the  $i^{\text{th}}$  tower in  $\phi$  and  $j^{\text{th}}$  tower in  $\eta$ . The missing energy is the vector sum

$$\vec{E}_T = \left( \sum_{i_\phi, j_\eta} (E_{T,i_\phi j_\eta} \cos(\phi)) \hat{x}, \sum_{i_\phi, j_\eta} (E_{T,i_\phi j_\eta} \sin(\phi)) \hat{y} \right). \quad (3.2)$$

Equation 3.3 defines a jet in the Level 1 calorimeter trigger, and a diagram is shown in Figure 3.2. A jet candidate is defined as the sum of  $E_T$  of a  $9 \times 9$  window

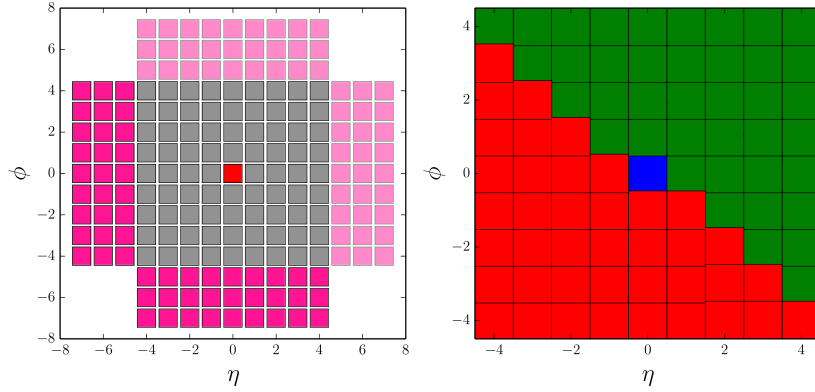


Figure 3.2: Templates describing the definition of a Level 1 Calorimeter Trigger jet object. On the left is the template for summation of trigger tower energies to obtain the jet energy. The pink outer regions are the bands used to obtain pileup energy. On the right is the veto template defining whether a jet centred on the central tower is a valid jet. If the energy of any of the red shaded towers is *greater than* the central tower energy, or if the energy of any of the green shaded towers is *greater than or equal to* the central tower energy, then the jet candidate is invalid.

around the central tower, which must be the highest energy tower in the window. In order to avoid the mutual veto by two equal energy towers in the grid, while also preventing double counting, a jet candidate may have a tower of equal energy within its window to one side, but not to the other side. Energy from pileup is estimated from a region neighbouring the jet. From the four strips of  $9 \times 3$  neighbouring the jet candidate, the lowest three are used as the pileup estimate, and their energy is subtracted from the jet  $E_T$ . The twelve highest energy jets are passed to the output, and sent to the  $\mu$ GT.

$$E_{T,\text{Jet}}(i_\phi, j_\eta) = \sum_{i_\phi-4, j_\eta-4}^{i_\phi+4, j_\eta+4} E_{T, i_\phi j_\eta} - E_{T,\text{Pileup}}, \quad (3.3)$$

where

$$E_{T,\text{Neighbours}} = \left( \sum_{i_\phi-4, j_\eta\pm 5}^{i_\phi+4, j_\eta\pm 7} E_{i_\phi, j_\eta}, \sum_{i_\phi\pm 5, j_\eta-4}^{i_\phi\pm 7, j_\eta+4} E_{i_\phi, j_\eta} \right), \quad (3.4)$$

$$E_{T,\text{Pileup}} = E_{T,\text{Neighbours}} - \max(E_{T,\text{Neighbours}}). \quad (3.5)$$

The  $9 \times 9$  area was chosen to be close to the size of the  $\Delta R = 0.4$  parameter

frequently used with the anti- $k_T$  algorithm to compute jets offline, while having a fixed size greatly simplifies the implementation. The energy of the leading jet computed by the L1T very closely matches that computed by anti- $k_T$  with  $\Delta R = 0.4$  using trigger tower inputs, with a slight bias towards higher energy [56].

### 3.3 Algorithm

#### 3.3.1 Jets

The time multiplexing scheme introduces a mapping of trigger tower to input link and time slice. The link on which a trigger tower arrives corresponds to the  $\phi$  position, while the time of arrival relates to the  $\eta$  position. Summing tower energies in a window with extent in  $\eta$  therefore requires significant pipelining to access data from multiple clock cycles after it was received, while the  $\phi$  extent requires a combination of signals from multiple locations across the chip. All of the algorithm is fully pipelined, that is, new data arrives on every clock cycle and the algorithm execution never stalls. A clock frequency of 240 MHz is used throughout.

In order to keep up with the incoming data rate, and to achieve the smallest possible latency, the algorithm is highly parallelised. For the jets, this means that each trigger tower is initially considered a jet candidate, and processed as such: its  $E_{T,\text{Jet}}$  is computed. The  $E_{T,\text{Jet}}$  centred around every tower in two  $\phi$  rings on opposite sides of the interaction point is computed in parallel. Simultaneously, a map of the highest energy towers is constructed to ultimately select the jet candidates which are centred on local maxima of energy. All of this is performed as soon as data begins to arrive from the detector, before the whole event has been seen. Because the span of a jet object is still much less than the total extent of the calorimeter, and only twelve are selected at the end, the incoming trigger towers can be reduced to a handful of possible jet candidates of a smaller data size, with a final accumulation step to pick out only the highest energy.

The formation of jet sums and the map of local maxima are a fully unrolled, static configuration of the algorithm within the FPGA. Both the jet energy and validity are calculated in a piecewise fashion, illustrated in Figure 3.3 for a section nine towers wide in  $\phi$ . As every tower is initially considered to be a jet candidate, until vetoed, there is a large overlap between the computations of the energy of nearby jet candidates. In order to save FPGA resources, and since addition components only take two inputs, rather than performing multiple summations with the same data,

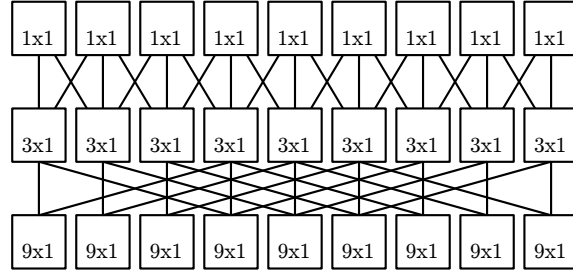


Figure 3.3: The scheme for reuse of partial sums of jet energy to save FPGA resources. Each cell represents an object with the summed energy of a number of trigger towers in  $\phi \times \eta$ . Towers (with  $1 \times 1$  area in  $\phi \times \eta$ ) enter at the top row of the diagram. Three neighbouring towers are grouped, and their energy is summed making a  $3 \times 1$  partial jet sum. Finally three non-overlapping  $3 \times 1$  sums are grouped and summed.

intermediate sums are reused multiple times. At each stage three objects (input towers, or partial sums) are grouped and summed into a composite object. The inputs are then discarded (again saving resources), and the result propagates to the next calculation.

When both the  $9 \times 9$  sums and local maxima are available for one particular ring in  $\phi$ , the number of jet candidates can immediately be reduced from 72 to 18 with a multiplexer, since no strip of four neighbouring towers can contain more than one maximum according to the template of Figure 3.2. The pileup estimate around each tower is also multiplexed with the map of maxima, and then subtracted from the surviving 18.

To obtain the top six from each half barrel, the 18 are sorted in order of  $E_T$  and the lowest twelve are discarded. A bitonic sorting network, a static configuration of pairwise comparisons and swaps with no data dependence, is used. The dataflow graph of a bitonic sort network for four parallel inputs is shown in Figure 3.4. The basic unit is a comparator controlling the output of a pair of multiplexers, which swaps the order of the inputs depending on the comparison. At the output of this sort, the six highest  $E_T$  jets with the same  $\eta$  remain, ordered by  $E_T$ .

In order to accumulate over the highest  $E_T$  from the whole  $\eta$  range (which means allowing all data to arrive and propagate through the preceding steps), a pipelined accumulation step is used, an illustration of which is shown in Figure 3.5. The pipeline has six stages, one for each of the six jets to be selected from this half barrel (and duplicated for the other half barrel). The first stage holds the maximum  $E_T$  jet seen so far in a register. Each cycle, when a new ring of jet candidates is produced, a comparison is made between the current maximum and each of the new

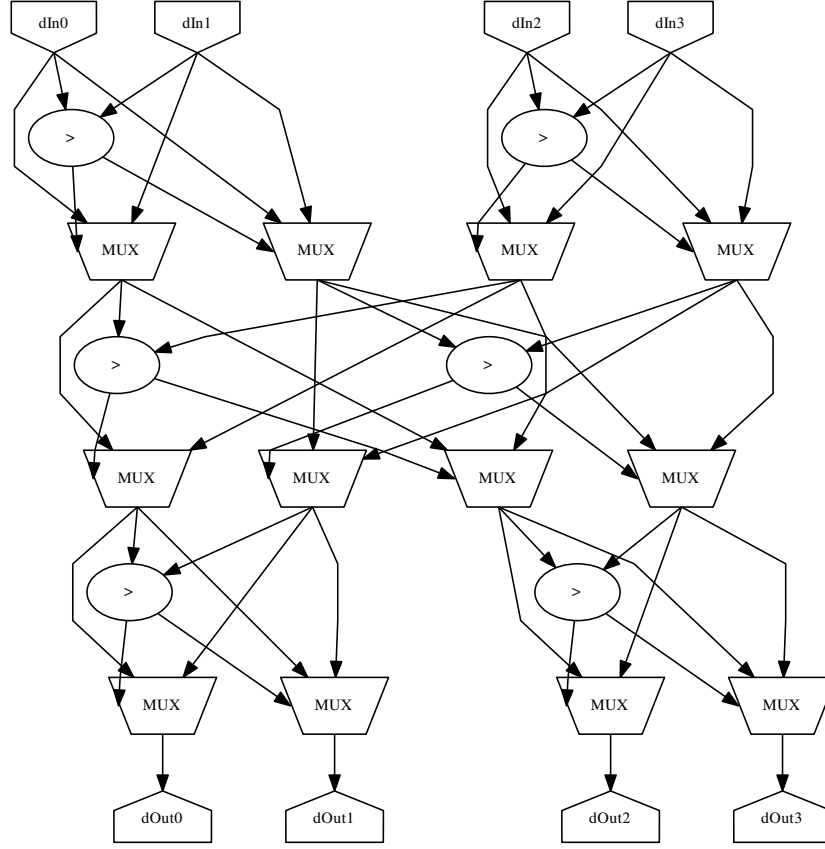


Figure 3.4: Dataflow graph of a bitonic sorting network with four parallel inputs. The basic unit consisting of a comparison and two multiplexers (labelled MUX) is repeated, with the data routing between units achieving the desired sorting.

six. If one or more of the new six candidates has a higher  $E_T$ , the highest of these becomes the new maximum, and the old maximum shuffles into the remaining five, maintaining the  $E_T$  ordering. If none of the six have a higher  $E_T$  than the current maximum, they all propagate to the next stage. The second stage carries out the same procedure, but selects the second highest  $E_T$  jet candidate because the highest  $E_T$  candidate never propagates past the first stage. The procedure continues for four more stages, such that the register at each stage holds one of the six highest  $E_T$  jets in one half barrel. When all of the calorimeter trigger towers have been received and propagated through the pipeline the six jets from each of the two accumulation stages are sent to the output.

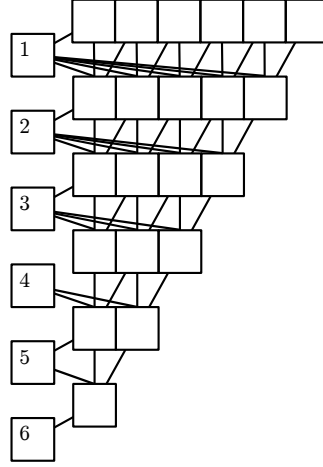


Figure 3.5: Accumulation stage of the jet sorter. Six  $E_T$  ordered jets from one  $\eta$  ring enter the accumulation at the top of the schematic, and propagate towards the bottom. At each stage the highest  $E_T$  jet object remains in the register on the left (numbered 1-6 in the diagram), while the remaining objects continue to the next stage. Each stage outputs one fewer jet than it received to keep the resource usage only as large as necessary.

### 3.3.2 Energy sum

The scalar and vector energy sums are executed in parallel with the jet algorithm. Since the scalar sum involves the same tower  $E_T$  quantities as the jet algorithm, the partial sums are reused for the event energy sum. The  $E_T$  sum for one ring in  $\phi$  is carried out with a balanced adder tree, and an adder with the output redirected to the input accumulates these sums over  $\eta$ . For the vector sums, the tower  $E_T$ s are first split into  $\hat{x}$  and  $\hat{y}$  components, then summed with a balanced adder tree and accumulated.

## 3.4 MaxJ Implementation

Similar design patterns to the VHDL were used when implementing the design using MaxJ. In both cases, sharing partial sums of tower energy between jets with different tower centres required explicit control by the developer, by constructing arrays of partial sums and using indexing to select neighbouring sums.

An advantage of MaxCompiler over VHDL is the functionality of the compiler to automatically set data types. When summing two integer values of  $n$  bits,  $n + 1$  bits are required to store the result at the same precision as the summands, and for all possible outcomes. In VHDL this must be coded by hand, whereas MaxJ will

guarantee the result has the correct type, unless overridden. Several strategies are provided for the compiler to set the type for the output of all operations, these are used extensively in Chapter 4.

The bitonic sorting network used to select the highest  $E_T$  jets can be easily implemented using recursion, as was done in the MaxJ implementation. VHDL, however, does not support recursion, and here the two code bodies differ greatly in achieving the same desired functionality. For a sorting network which discards some of the sorted data, some comparisons do not lead to the output. MaxCompiler analyses the path of the dataflow graph, and trims the network to only include operations which lead to outputs.

Latency is also controlled differently in the two languages. In VHDL, registers are added to create the pipeline with the construct `'if rising_edge(clk):'`. Conversely, in MaxJ, every operation is followed by a register unless otherwise forced with `'optimization.pushPipeliningFactor(0)'`. A VHDL design therefore has no pipelining by default, while a MaxJ design is maximally pipelined. The VHDL developer must add registers to achieve a reasonable clock frequency, while a MaxJ developer must remove registers to reduce the latency.

### 3.4.1 Interface with MP7

An interface was required to use the MaxJ design within an MP7 board. For all MP7 designs, a 'core firmware' package exists, which largely separates the algorithm and IO functionality of a design. The package provides firmware for all necessary external communications, with connections to a 'payload' defined by the user.

MaxCompiler produces VHDL output from the MaxJ Kernel, which can be interfaced with other VHDL. This corresponds to the top level in the algorithm hierarchy. MaxCompiler Kernels support asynchronous communication with each other, and so contain control ports for stalling and terminating execution. These are not required for the L1 Calorimeter Trigger, however, since it runs continuously. A wrapper module was programmatically generated to connect the data ports of the Kernel to the core firmware as the payload, and tie other control ports to constant values.

Table 3.1: Number of each type of resource used by each implementation of the jet and energy sum algorithm. Built for a Virtex-7 690T using Vivado 2015.4 with MP7 core infrastructure (numbers for algorithm only).

Resource	VHDL	MaxJ
Slice LUTs	95235	102508
Slice registers	153198	130072
DSPs	288	288
BRAM tiles	0	0
Lines of source code	3000	1500

## 3.5 Comparison

### 3.5.1 Functional Correctness

Monte Carlo events were passed through both the VHDL and MaxJ implementations of the algorithm. Figure 3.6 shows the functional correctness of the MaxJ implementation. Bit-identical results were obtained for the jet energies, and  $\eta$ , and for the scalar and vector energy sums. A small discrepancy is observed in the  $\phi$  distribution, which arises from a difference in the bitonic sorting network. Two jets with the same  $E_T$  and  $\eta$  but different  $\phi$  can emerge in a different order from the two implementations. When selecting the top six jets in the event one of these may be removed in the case that one has the sixth highest  $E_T$  and the other has the seventh. In other cases, they will simply emerge in a different order. The trigger performance would not be affected by this difference.

### 3.5.2 FPGA Resources

Resource consumption of each implementation is shown in Table 3.1. Approximately 8% more slice LUTs are used by the MaxJ implementation, with slightly fewer registers than the VHDL. The DSP usage is the same between the two. Multiplication is only performed for the projection of 144 towers onto the  $x$  and  $y$  axis, so the tool has correctly mapped these onto DSPs. No BRAMs are used by either implementation. The MaxJ implementation latency matches that of the VHDL. One noteworthy distinction is that the MaxJ code body is half the length of the VHDL for the same functionality. This suggests that the code may be easier to maintain.



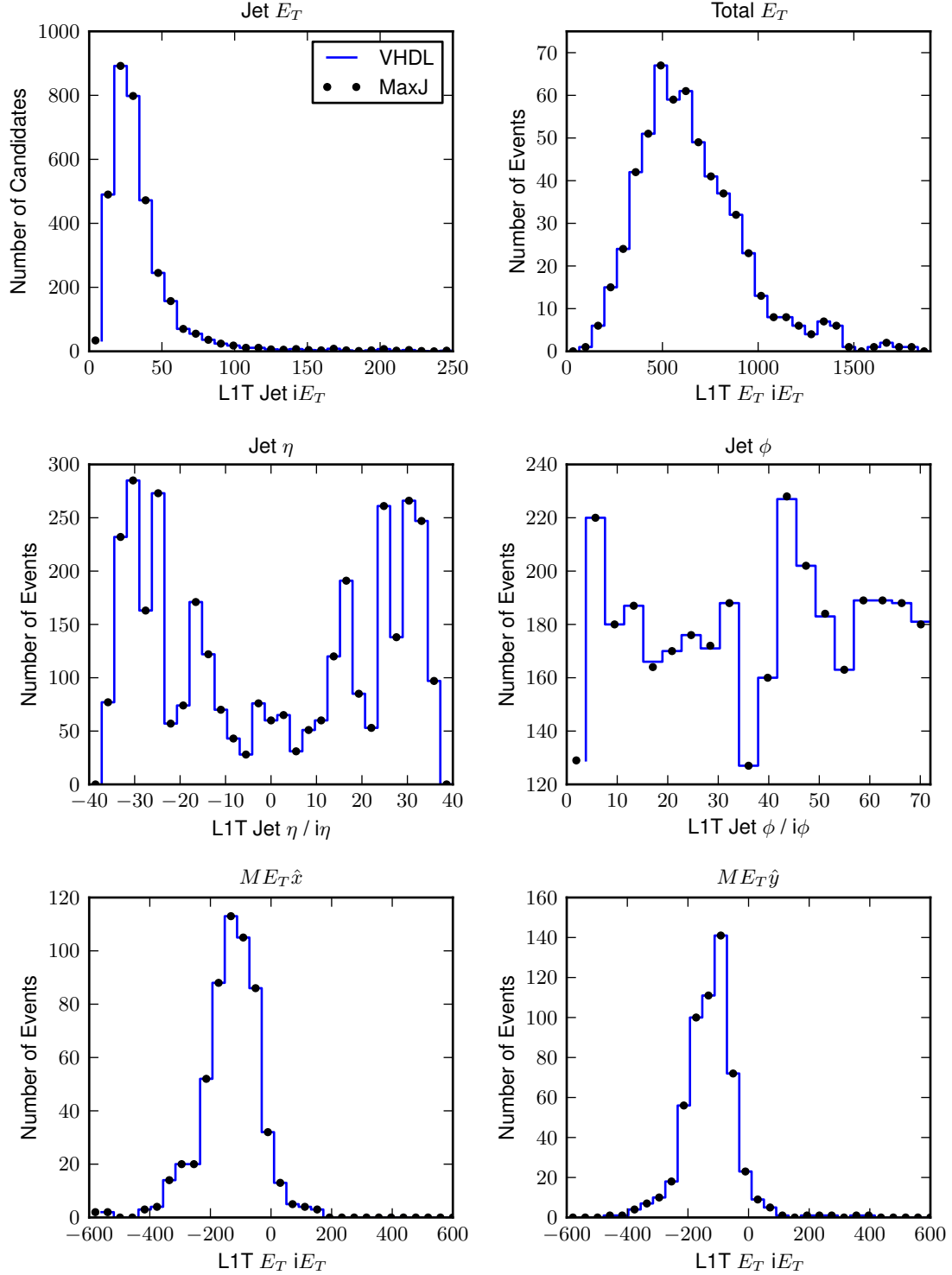


Figure 3.6: Output distributions of reconstructed jet and  $E_T$  parameters from the VHDL and MaxJ implementations of the L1 Calorimeter Trigger algorithms, in the integer units of the calorimeter trigger. A bit-exact matching is observed for all parameters apart from the jet  $\phi$ , due to an internal difference in the sorting of jets.

## 3.6 Summary

This chapter presents the first use of MaxCompiler in a low latency, high throughput, scenario interfacing to hardware not produced by Maxeler Technologies. By comparing the performance of a MaxCompiler implementation of an algorithm originally implemented using VHDL, it has been shown that the tool is viable for producing low latency algorithms with an efficient use of FPGA resources. Due to the fine level of control provided by the tool, bit-wise matching to the handwritten VHDL was achieved. The MaxCompiler implementation resulted in an 8% increase in LUT usage, while requiring half the number of lines of code. This inefficiency might be acceptable, if the ease of algorithm design with the higher level language enables realisation of designs that would be extremely difficult with the conventional HDL approach to writing trigger algorithms, as shall be explored in the next chapter.

## Chapter 4

# Track Reconstruction for the Level 1 Trigger

During Long Shutdown 3, operation of the LHC will cease, and the machine will be upgraded to deliver an instantaneous luminosity of  $L = 5 \times 10^{34} \text{ cm s}^{-1}$ , five times its current value, restarting as the High Luminosity LHC (HL-LHC). The CMS experiment will be significantly upgraded in order to maintain performance in the high radiation, high pileup conditions [12]. The tracking detector will be completely replaced, and for the first time will send data to the Level 1 Trigger (L1T).

The new detector will consist of an inner and outer part: the inner containing silicon pixel sensors, and the outer part containing silicon strip, and macro-pixel detectors. These sensors will have a higher granularity, better radiation tolerance, and extend further in pseudo-rapidity than the current tracker. They will also allow for the higher trigger acceptance rate of 750 kHz, and longer Level 1 latency of 12.5  $\mu\text{s}$  of the Phase II detector. Furthermore the outer tracker modules will send information to the Level 1 trigger, for particles with transverse momentum above a threshold. This information, which is first made available for reconstruction in the High Level Trigger at CMS currently, will be used to maintain trigger thresholds for particle energy and momentum as low as possible within the maximum trigger accept rate. It is the reconstruction of tracks, that is determining the kinematic properties of the charged particles, from the outer tracker in the Level 1 trigger that is the main topic of this chapter.

Reconstruction of charged particle trajectories is an essential part of CMS event reconstruction. The procedure is, however, computationally expensive, as discussed in Section 2.10. During LHC Run I the CPU time required for track reconstruction

was almost as much as the time for all other reconstruction [61]. Tracking execution time on a CPU scales badly with increasing pileup as the number of possible combinations of hits increases. The track reconstruction time with 140 PU is a factor ten slower than with 70 PU. At the HLT tracking is executed only sparingly, after attempting to reject events using faster to reconstruct requirements such as those based on calorimetry. Track reconstruction is not currently performed in the Level 1 Trigger at CMS, in part as the tracker cannot be read out at the full LHC event rate, but also because the reconstruction is extremely difficult in the microsecond timescale available at L1.

## 4.1 Track Trigger Demonstrator

A demonstration of a concept for a Level 1 Track Trigger was constructed to investigate the feasibility and performance of track reconstruction within the restrictions of the Level 1 Trigger in a 200 pileup regime, and also to determine the scale of the final system in hardware terms.

For the Phase II upgrade, the total L1 latency budget is at most  $12.5\,\mu\text{s}$  [12]. Of this,  $3.5\,\mu\text{s}$  are allocated to utilise the reconstructed tracks, calorimeter and muon primitives to make the final trigger decision.  $1\,\mu\text{s}$  is required for the propagation of the L1 accept decision to the front end chips, and  $3\,\mu\text{s}$  is reserved for a safety margin. The total time to receive stubs and completely reconstruct them into tracks is therefore  $5\,\mu\text{s}$ . Of this,  $1\,\mu\text{s}$  is required for the on detector processing, and transmission to the Data, Trigger and Control (DTC) boards, so  $4\,\mu\text{s}$  remains for reconstruction [40].

### 4.1.1 Time Multiplexed Track Trigger

The time-multiplexed track trigger demonstrator hinges on the platform of a time-multiplexed architecture using FPGAs for processing that is feed-forward only (that is, there are no bidirectional links between boards). This concept decouples the algorithm design from the architecture as much as possible, and maintains a high degree of flexibility for algorithm changes. By segmenting the processors only in  $\phi$  and by bunch crossing (by time-multiplexing), the algorithm is the same on every processor in the system. This scalable design allowed the demonstration of the entire track reconstruction system with just one instance of a Track Finder Processor (TFP). The reconstruction algorithm centres on the use of a Hough Transform for

track finding, with pre and post-processing steps to parallelise the processing and perform track fitting. The reconstruction steps proceed as follows:

- Geometric Processor (GP) - Performs preprocessing of stub data to a format designed for the Hough Transform, and routes the stubs to Hough Transform instances compatible with the narrower internal sectors.
- Hough Transform (HT) - Groups stubs consistent with tracks in the  $r - \phi$  plane. Multiple instances are created, covering different detector regions, and are evaluated in parallel.
- Kalman Filter (KF) - Fits track parameters to the stubs in a candidate, simultaneously rejecting inconsistent candidates and stubs.
- Duplicate Removal (DR) - Utilises the precise fit information to remove tracks found in multiple HT cells.

The demonstrator TFP, described in more detail in Section 4.1.6, covers one eighth of the detector – the full range in  $\eta$  and  $\pi/4$  in  $\phi$  – and one event in every 36 with time multiplexing. Each element of the reconstruction procedure will now be discussed in detail.

### 4.1.2 Geometric Processor

The geometric processor (GP) firstly unpacks 48-bit stubs (containing the  $(r, \phi, z)$  coordinates, bend, and a validity bit) from the DTC into an extended 64-bit format which removes some processing work from the HT. The first extra bits are an ID field for the detector layer, since the HT track candidate definition requires a number of stubs on unique layers. The second extra bits correspond to the track  $p_T$  range with which a stub is compatible, derived from the bend, in the units used internally by the HT.

The detector segmentation performed by the GP is constrained by the availability of optical links on the MP7, which is 72 in each direction, since the GP alone occupies one board in the demonstrator system. Since the stubs are 64-bits wide, they must be carried across two links, yielding a maximum of 36 sectors. Two divisions are made in  $\phi$  (within the  $\pi/4$  covered by the TFP), and 18 in  $\eta$ , shown in Figure 4.1. Each geometrical sector is served by a separate instance of an HT algorithm, thus parallelising the track finding. These sectors overlap, allowing for the curvature of tracks in  $r - \phi$  and the 150 mm interaction region length in  $z$ .

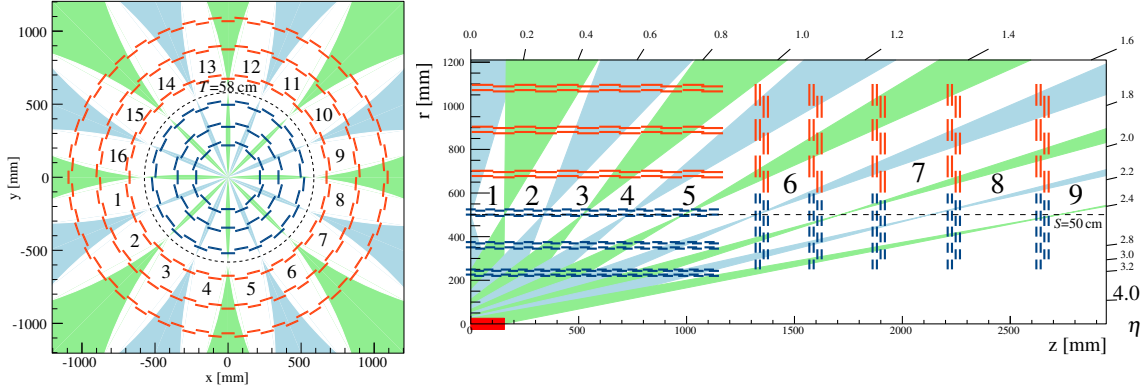


Figure 4.1: Segmentation of the tracker into processing regions. Shaded sections show overlaps between regions. Each demonstrator TFP serves one eighth of the detector in  $\phi$ , for the full range in  $\eta$ . Internal segmentation creates two additional  $\phi$  regions and 18  $\eta$  regions within each TFP.

The HT implementation, which is described in section 4.1.3, is capable of handling additional sectors within one physical processing unit, for a small cost in additional resources. The 18  $\eta$  sectors are further split into two halves with this mechanism. The GP attaches a bit for each of these sectors to the stub data. Since the HT finds tracks only in the  $r - \phi$  plane, it is prone to producing track candidates whose stubs do not form a physical trajectory in the  $r - z$  plane. The 36  $\eta$  segments limit the production of these fake tracks, ensuring that the candidates are somewhat consistent with a straight line in  $r - z$ .

An ‘any-to-any’ routing mesh transmits formatted stubs from any input link to any HT segment, with compatible segments calculated in the GP processing step. The routing is performed in stages, with one input leading to at most either 3 or 4 outputs depending on the layer, shown in Figure 4.2. This way, the movement of data at each layer is small, enabling a high clock frequency when placed in the FPGA compared to a network which performs ‘any-to-any’ routing in one step. Each node in the routing network receives data from 3 to 4 nodes in the preceding layer, but processes at most one stub per clock cycle. This requires buffering and arbitration between inputs, which ultimately leads to gaps in the data packet at the output of the GP. A small loss of stubs is introduced, as the packet length may exceed the time multiplexing period.

The resource usage of the components is shown in table 4.1. Since each of the 72 input links is deserialised into 32 bit data, 48 of the 48 bit stubs arrive in parallel each cycle. The pre-processing block is duplicated once per input stub, therefore 48 times in total. One instance of the 48 to 36 routing network is used, with each

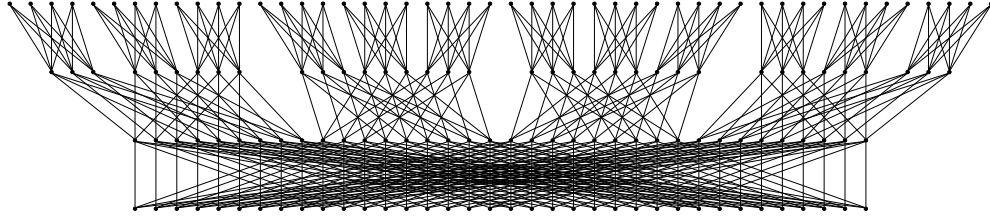


Figure 4.2: The Geometric Processor (GP) routing network. 64 bit stubs, and their sector addresses, arrive at the nodes at the top of the diagram. They move along lines to nodes in the next layer according to the address. At the output layer, at the bottom of the diagram, each node corresponds to one geometric processing sector.

Table 4.1: Utilisation of FPGA resources for the stub unpacking and pre-processing block, of which 48 instances are needed to cover one TFP, and the routing network. Percentages are reported as the fraction used of the total available in the Xilinx Virtex-7 XC7VX690T FPGA used for the demonstrator.

Component	LUTs	DSPs	FFs	BRAM (36 Kb)
Pre-processing Block	1942 (0.4%)	22 (0.6%)	2416 (0.3%)	1 (0.0%)
Routing Network	27700 (6.4%)	0 (0.0%)	89531 (10.3%)	174 (11.8%)

output node containing stubs from only one of the geometric regions within the TFP. The GP output is sent to the Hough Transform.

### 4.1.3 Hough Transform

The Hough Transform is a technique used for detecting features in image pixel data [75], and used early on for detecting charged particle tracks in bubble chamber pictures [76]. Initially developed for finding straight lines, the method was first extended to finding more general curves by Duda and Hart [77], and subsequently for arbitrary objects [78]. All types of Hough Transform have been utilised within HEP, from the aforementioned finding of straight lines in 2-dimensional bubble chamber photographs, to a 4-dimensional accumulator space, using templates accounting for the fact that physical effects (multiple Coulomb scattering and energy loss) alter the particle trajectory in a cylindrical tracker such as that used at CMS [79].

### Algorithm

The basic algorithm, for finding straight lines, maps the space of spatial coordinates to the space of line parameters. The straight line may be described by

$$y = mx + c, \quad (4.1)$$

which can be rearranged to

$$c = y - xm. \quad (4.2)$$

A single pair of coordinates  $(x, y)$  lie on infinite straight lines parametrised by  $(m, c)$  constrained by equation 4.2. With multiple coordinates,  $(x, y)$ , along a line, the lines of possible parameters,  $(m, c)$ , for each point intersect at the true  $(m, c)$  value of the line. In order to find the  $(m, c)$  values of interest, a two-dimensional histogram of the  $(m, c)$  space is made, with one count for each bin that a line intersects. Local maxima in this accumulator space correspond to the  $(m, c)$  parameters of the straight lines. Examples of the  $(x, y)$ , continuous  $(m, c)$ , and binned  $(m, c)$  accumulator spaces are shown for different scenarios in Figure 4.3.

For particles of charge  $q$ , in a magnetic field  $B$  with momentum in the plane perpendicular to the magnetic field  $p_T$  (in CMS this is the  $x - y$  plane), the radius of curvature is

$$R = \frac{p_T}{0.3qB}, \quad (4.3)$$

for  $R$  in metres,  $p_T$  in GeV and  $B$  in Tesla. The trajectory of particles originating at the interaction point in the  $r - \phi$  plane is

$$\frac{r}{2R} = \sin(\phi - \phi_0), \quad (4.4)$$

where  $\phi_0$  is the initial angle in the  $x - y$  plane from the positive  $x$  axis towards the positive  $y$  axis. With a sufficiently large  $R$ , and therefore  $p_T$  by equation 4.3, the small angle approximation can be made, yielding

$$\frac{r}{2R} \approx \phi - \phi_0. \quad (4.5)$$

Given the on detector  $p_T$  cut minimum of 2 GeV, and the maximum tracker radius of 1 m, the worst case difference of the approximation is  $0.23^\circ$ , or around 1.4%.



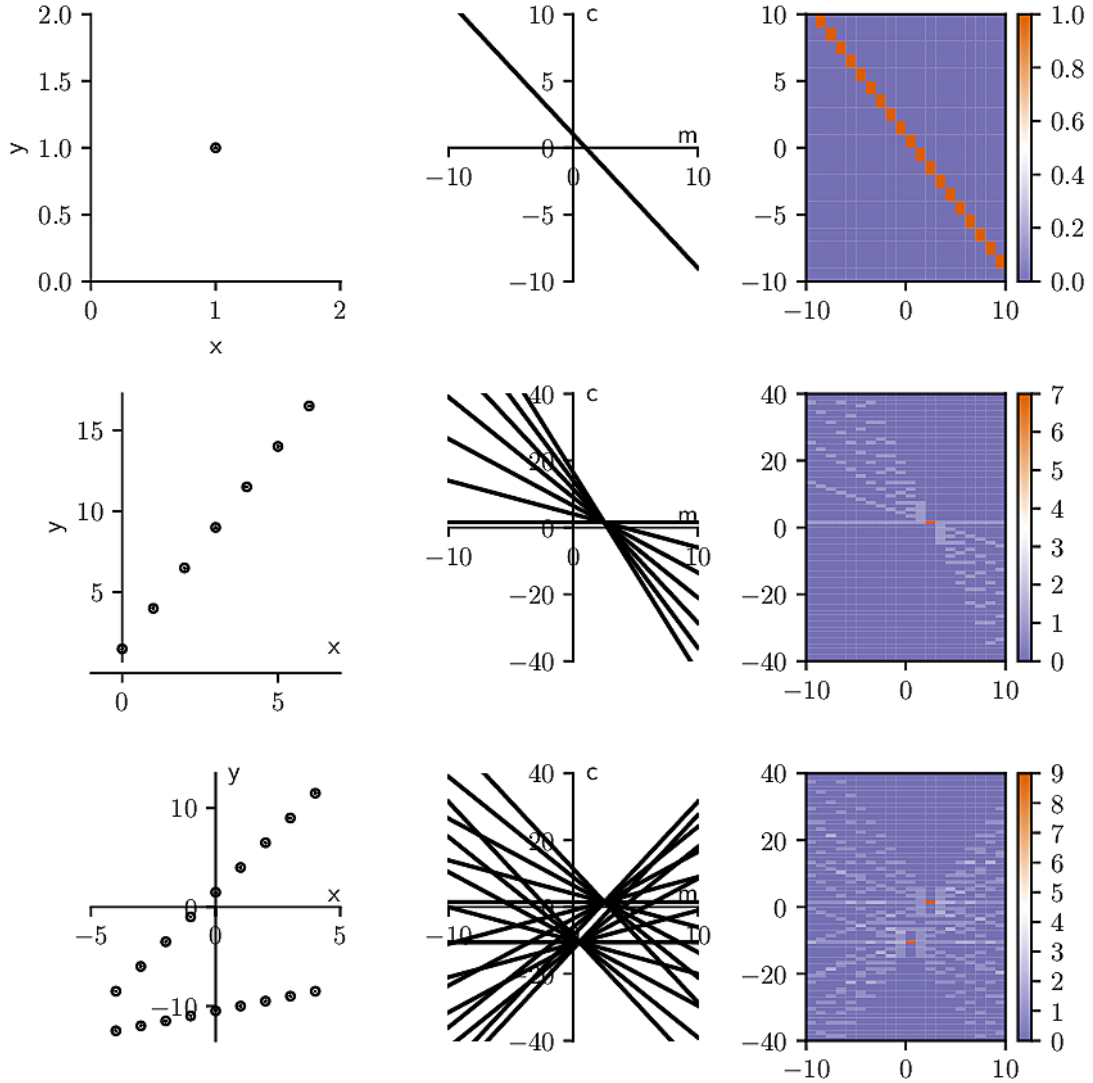


Figure 4.3: Examples of Hough Transform evaluation for straight lines. Each row displays the  $(x, y)$  space (left), the continuous  $(m, c)$  space (middle), and discrete  $(m, c)$  accumulator space (right). The top row shows how a single point in  $(x, y)$  maps to a line in  $(m, c)$  space. On the middle row, each point in  $(x, y)$  corresponds to one of the lines in  $(m, c)$  space. The lines intersect at the actual straight line parameters. In the accumulator space, one can see that only the bin containing the true parameters has a ‘vote’ from every  $(x, y)$  point, all other bins have one vote. On the bottom row, there are  $(x, y)$  points from two different straight lines. The distinct parameters of the two lines create local maxima in the accumulator space. Some bins accrue more than one vote where the many lines in  $(m, c)$  space overlap. A further difference between the middle and bottom row is that the  $x$  axis has been transformed so that there are equal numbers of measurements on either side of the  $y$  axis. This improves the separation between lines in the accumulator space, and is used in the track finding HT.

Equation 4.5 is the basis of the straight line relationship used here to find tracks in the  $r - \phi$  plane by means of a Hough Transform.

It can be seen in equation 4.2 that the gradient of lines in accumulator space is given by the  $x$  (here  $r$ ) ordinate of the measurement. In the conventional cylindrical coordinate system,  $r$  is only positive, so only negative line gradients are possible in the Hough Transform. When creating the histogram of  $(m, c)$  parameter space with a rectilinear grid, the separation between lines can be improved by allowing positive gradients. This is achieved by transforming the radius  $r$  with an offset

$$r_T = r - T. \quad (4.6)$$

$T$  is chosen to be 580 mm, a radius which transforms approximately equal number of stubs to produce positive and negative line gradients in the transformed space. The transformed track parameter space of equation 4.5 then becomes

$$\phi = \frac{1}{2R}r_T + \phi_T, \quad (4.7)$$

where  $\phi_T$  is the angle of the track at radius  $T$ . The Hough Transform algorithm is a histogram over all stubs within a GP sector of the  $((2R)^{-1}, \phi_T)$  space. The  $\phi_T$  range of the array must cover the whole sector, and the  $(2R)^{-1}$  range must cover  $|p_T| > 3 \text{ GeV}$ . A bin containing stubs from five unique detector layers in the same  $\eta$  subsector is classed as a track candidate.

## Implementation

The HT algorithm described in section 4.1.3 is implemented in FPGA firmware in two pipelined stages. Firstly the accumulator array is filled with stubs, followed by the readout of track candidates. The array implementation can input and output one stub on each clock cycle.

Within the array, the  $32 \times 64$  bins are split into columns spanning the length of the  $\phi_T$  axis, for one bin in  $(2R)^{-1}$ , each served by an instance of processing logic, shown in Figure 4.4. Several columns are daisy-chained together, with stubs passed along the column in a pipeline. Propagating the stubs this way means that the HT straight line equation can be expressed

$$\phi_T(n) = \phi_T(0) + n \cdot \Delta_{(2R)^{-1}} \cdot r_T, \quad (4.8)$$

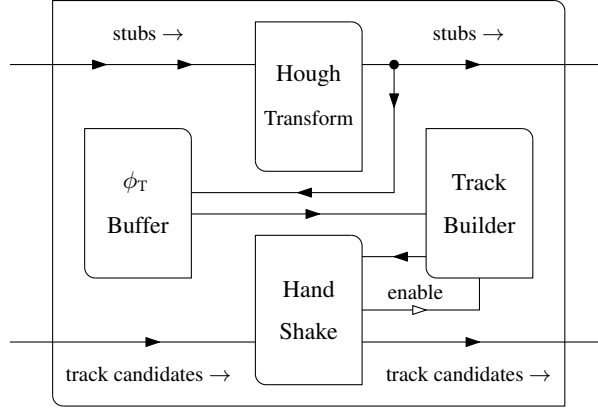


Figure 4.4: Schematic of the implementation of one Hough Transform column processor. Stubs arrive from, and are sent to, neighbouring column processors, forming a pipelined chain. The ‘Hough Transform’ block evaluates the  $\phi_T$  of the current stub at the  $(2R)^{-1}$  edges of the column. The appropriate  $\phi_T$  bins are sent through the ‘ $\phi_T$  Buffer’, which handles cases where more than one bin in the column must be incremented. The ‘Track Builder’ writes the stub pointer to the column memory and determines which cells are track candidates. Finally, the ‘Hand Shake’ block outputs the track candidate stubs in a contiguous stream during event readout.

for column  $n$ .  $\phi_T(0)$  is simply the  $\phi$  coordinate of the stub. Equation 4.8 is implemented using two DSPs per column: one for each bin edge. The chain of column processors need not span the full  $(2R)^{-1}$  axis, and splitting the axis into chunks, each two or three columns long, reduces the latency of the design, and the memory size requirement for the processor.

Since the finding of track candidates is not the end of the procedure – they must be fit and filtered – access to the stubs which contributed to the bin reaching the threshold must be provided. This is achieved using a segmented memory, with capacity for pointers to 16 stubs in the column manager’s memory reserved for each of the 64 cells in the column. The template is duplicated, one half for alternate LHC collision events, allowing for simultaneous readout of the first event and filling of the second event. This requires one of the 18 Kb block RAMs available in the FPGA.

The same logical element maintains a count of which detector layers have stubs in each bin. A separate count is maintained for each of the two  $\eta$  subsectors. The cell is only classed as a track candidate if the threshold is reached in one, or both, of the subsectors (stubs from both subsectors are read out if the threshold is reached in either).

It can be possible for a stub to lie in two  $\phi_T$  bins within the  $(2R)^{-1}$  column, in which case two addresses must be written to. This writing must take place on

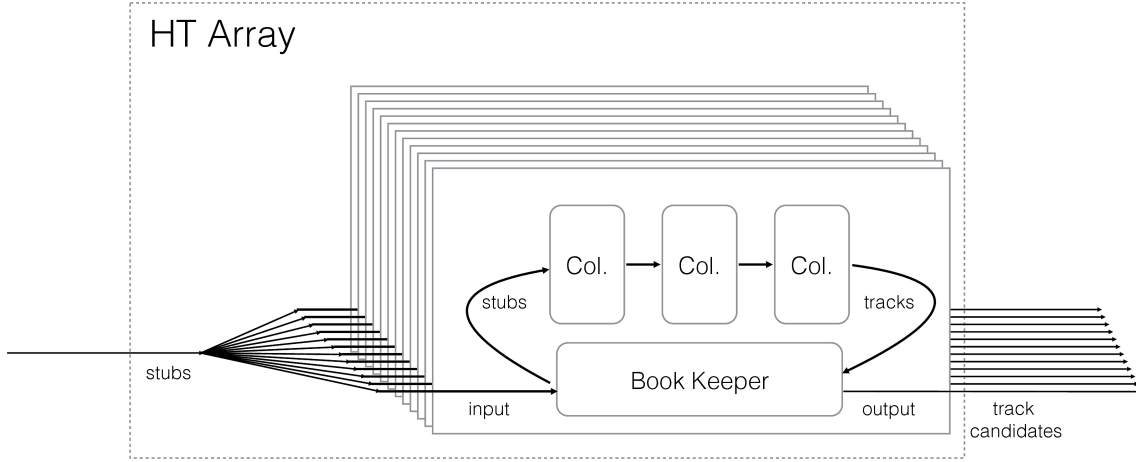


Figure 4.5: Schematic of one Hough Transform array as implemented in FPGA firmware. Each column (‘Col.’) processor spans all 64 bins in the  $\phi_T$  variable, and one bin in  $(2R)^{-1}$ . A book keeper stores input stubs and passes them through the daisy-chained columns. The 32 bins of the  $(2R)^{-1}$  axis are split into twelve chunks managed by independent book keepers: 8 of which manage 3 columns; the remaining 4 of which manage 2.

Table 4.2: FPGA resource utilisation for one HT column processor, and one HT accumulator array. Percentages are of the resources available in the Xilinx Virtex-7 XCVX690T FPGA used. The 36 geometric sectors are served by separate array instances, so resources for 36 HT arrays are required.

	LUTs	DSPs	FFs	BRAM (36 Kb)
One Column	188 (0.0%)	2 (0.1%)	204 (0.0%)	1 (0.1%)
One HT array	6014 (1.4%)	64 (1.8%)	6718 (0.8%)	33 (2.2%)

separate clock cycles, so a buffer is introduced to interleave the storage of the second column stub when there is a gap in the datastream.

Chains of HT column processors are managed by a ‘book keeper’, an array of which is shown in Figure 4.5. This block receives the stubs from the GP, stores them in a memory, and passes them through the HT array columns. One stub is received per clock cycle, and written into a 36 Kb memory. The minimum data needed for the HT calculations, and stub retrieval, are sent to the first column in the chain: the  $r_T$ ,  $\phi_T$  at a reduced resolution; the ID of the layer on which the stub was found; the range of  $(2R)^{-1}$  bins compatible with the stub bend, and the pointer to the full stub data in the book keeper memory. The length of the  $(2R)^{-1}$  axis is split into twelve chains of neighbouring columns, each arranged into a pipeline and managed by one book keeper instance. FPGA resource usage for these components is shown in table 4.2.

At the end of the time-multiplexed period, after all stubs have been written into the HT array, the cells meeting the threshold requirement are read out. A ‘Hand Shake’ component reads the stubs, one per clock cycle, and presents them on the track candidate bus, attaching the cell array indices to the data. All the stubs belonging to the candidate are read out consecutively into the stream, before beginning the next candidate. The stream of stub pointers and  $(m, c)$  indices passes through the book keeper, which retrieves the original stub data to pass to the track fit.

At the output of the book keepers, a multiplexer consolidates six streams into a single output, so tracks from each HT are read out on two streams, and therefore 72 per TFP. A load balancing unit then moves track candidates between output streams initially paired with different geometrical sectors. Dense jets tend to create many track candidates in a single sector, which could overload the fitting stage. Lifting the geometrical segmentation of the data streams better utilises the track fitting resources later. At the output of this load balancing, the candidates are streamed into the track fitter.

#### 4.1.4 Kalman Filter

A Kalman Filter [64] was chosen to fit and simultaneously clean the track candidates after the finding performed by the Hough Transform. The algorithm, which is the *de facto* choice for offline and HLT tracking in CMS, was chosen in this case for a number of reasons. Firstly, the algorithm is a local one, meaning hits within the collection are sequentially added to the fit, contrasting with a global method which considers all hits at once, such as a linear regression. The resolution, and also efficiency, of a global track fit is adversely affected by the presence of the many outliers in the track candidates found by the HT, since all measurements pull the fit to the parameters simultaneously. These outliers can lie several standard deviations of detector resolution away from the true track due to the coarseness of the longitudinal segmentation. Since the Kalman Filter introduces measurements sequentially, a stub can be left out of the fit if deemed to be too far from the track hypothesis at that point. The local nature of the Kalman Filter also allows for small matrices in the algorithm. The matrices have dimensions of the number of track parameters, and the dimensionality of the measurement, in some combination. By contrast, a linear regression requires matrices with dimensions up to the product of the number of measurements with the measurement dimensionality. When prioritising a low

latency implementation, small matrices should enable a faster algorithm. A linear regression also has the complication that the number of measurements in a track candidate is not constant, which makes a highly parallel FPGA implementation more challenging.

The track fit seeks to obtain the track parameters from the simplified track equations 4.9 and 4.10.

$$\phi(r) = \frac{1}{2R}r + \phi_0, \quad (4.9)$$

$$z(r) = \tan(\lambda) \cdot r + z_0, \quad (4.10)$$

where  $\phi$  is the azimuthal angle of the trajectory at radius  $r$  from the beam line,  $R$  is the radius of curvature of the trajectory,  $\phi_0$  is the initial azimuthal angle,  $z$  is the longitudinal position of the trajectory at  $r$ ,  $\lambda$  is the angle from the  $r$  axis in the  $r - z$  plane (the dip angle), and  $z_0$  is the initial longitudinal position of the track at the beam line. Using the Kalman Filter nomenclature introduced in Equations 2.4 to 2.12, these relations define the state and measurement vectors to be those given by Equations 4.11 and 4.12.

$$x = \begin{pmatrix} (2R)^{-1} \\ \phi_0 \\ \tan \lambda \\ z_0 \end{pmatrix}, \quad (4.11)$$

$$m = \begin{pmatrix} \phi \\ z \end{pmatrix}. \quad (4.12)$$

The matrix  $\mathbf{F}$  must transport the state from layer  $k - 1$  to layer  $k$  according to equation 2.5, and since we do not expect the track parameters to change we have that  $\mathbf{F} = \mathbf{I}_4$ . In order to obtain the residual of equation 2.7,  $\mathbf{H}$  is:

$$H = \begin{pmatrix} r & 1 & 0 & 0 \\ 0 & 0 & r & 1 \end{pmatrix}.$$

Without including the effects of multiple scattering, the matrix  $\mathbf{Q} = \mathbf{0}_4$ . Uncertainties in the stub measurements are included into the matrix  $\mathbf{V}$  as in equation

4.13.

$$\mathbf{V} = \begin{pmatrix} \sigma_\phi^2 & 0 \\ 0 & \sigma_z^2 \end{pmatrix}, \quad (4.13)$$

where

$$\sigma_\phi^2 = \left( \frac{1}{\sqrt{12}} \frac{p}{r} \right)^2 + \begin{cases} 0 & \text{if stub in barrel} \\ \left( \frac{1.05}{2R} \right)^2 & \text{if stub in endcap} \end{cases} \quad (4.14)$$

where  $p$  is the pitch of a module located at radius  $r$ . An endcap correction is required since a track originating at the nominal interaction point does not intersect the strip at a tangent for modules in the endcap. The factor 1.05 differs from 1 as it was found to increase efficiency during testing.

$$\sigma_z^2 = \left( \frac{l}{\sqrt{12}} \right)^2 \times \begin{cases} 1 & \text{if stub in barrel} \\ 0.9 \tan^2 \lambda & \text{if stub in endcap,} \end{cases} \quad (4.15)$$

where  $l$  is the strip length of the module, and the factor 0.9 was found to increase the efficiency compared to 1.

In order to begin the Kalman Filter, an estimate of the state and its covariance is required. The track finding information is used. For the  $r - \phi$  part the central value of the  $m$  and  $c$  bin of the candidate are used, transformed to  $(2R)^{-1}$  and  $\phi_0$ . In the  $r - z$  plane, the initial vertex is unknown so  $z_0 = 0$  in the initial state. The central value of the  $\eta$  sector is used to estimate the initial  $\tan \lambda$ . Similarly, for the covariance matrix, the width of the HT cells,  $\eta$  sector and beam spot uncertainty are used.

The Equations 2.4 to 2.12 describe the update of a state,  $x$ , with a measurement  $m$  on detector layer  $k$ . A track candidate found by the HT contains multiple measurements, some of which may be on the same layer. In this instance, the measurements are considered in turn, yielding a new state for each hit on the same layer, each of which is independent of the other hits. Further measurements on subsequent layers are then filtered with each of these new states independently. The HT may also find candidates with no stub on a given layer, or with only erroneous stubs in a particular layer on an otherwise real track. To account for this, as well as filtering all measurements on a layer into new states, each state is propagated to

the next layer unadjusted. This is only done for states whereby skipping the layer would not make the state invalid. A valid candidate is one with no more than two missing layers, which may not be consecutive layers, and may not both be PS layers. This also allows the fitter to mitigate against the presence of broken modules. The requirement for a valid candidate to contain at least two stubs from PS modules is motivated by their superior  $z$  resolution, giving a better vertex resolution at the end of fitting.

The creation of multiple new track states at every layer creates a kind of ‘combinatorial explosion’, and slows down the processing as each is processed in turn. To reduce the impact of the extra combinations in the hardware, two procedures are used. The first is limiting propagation of states where one or more parameter fails a specified cut. The cuts applied are as follows:

- $z_0$ : the nominal interaction region has an extent of 150 mm either side of  $z = 0$  with a flat distribution of interactions. States with a  $z_0$  outside this range are cut
- $p_T$ : since only tracks with  $p_T > 3$  GeV should be found, any with lower  $p_T$  are deemed spurious, and cut.
- $\phi_0$ : duplication of stubs across  $\phi$  processors can lead to the same track being reconstructed in each processor. In one processor the  $\phi_0$  will be outside of its bounds, so these states are cut.
- $\eta$ : the same duplication occurs across  $\eta$  sectors, so states with trajectories with an out of bounds  $\tan \lambda$  are cut.
- $\chi^2$ : candidates with stubs far from the state trajectory obtain a large  $\chi^2$  value, and these are cut to reduce the number of fake stubs, and tracks.

In addition to the cuts, an accumulation step is introduced with only the ‘best’ states surviving. At each iteration, states are ordered first by their number of skipped layers, then by the  $\chi^2$ . At the first iteration, the best four are propagated, and at subsequent iterations only the single best state continues. Figure 4.6 illustrates the procedure for an example candidate which has one erroneous stub on the second layer. States filtered with the erroneous stub can be rejected when the  $\chi^2$  of the residual with later stubs is large.

The FPGA implementation of the algorithm can be considered in two parts: the filtering of a state and measurement according to equations 2.4 to 2.12; and the iteration over layers, and over multiple stubs in a layer.



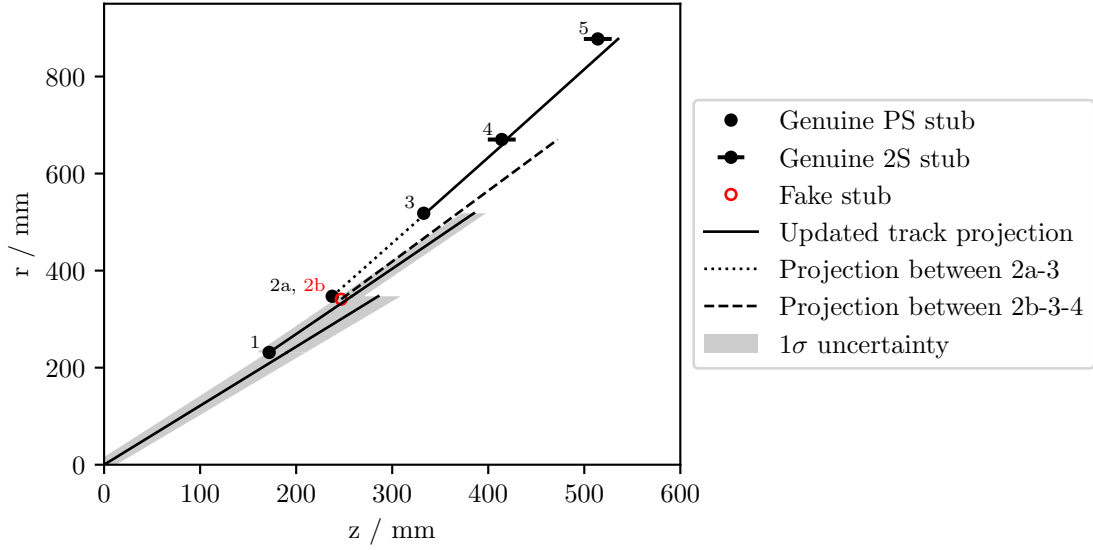


Figure 4.6: An example track candidate found by the HT with intermediate KF states shown. The candidate has a single stub, on the second layer, not associated with the same MC particle as the rest (labelled 2b). The state filtered with the fake stub does not match either the stub on layer 3 or 4, and is terminated due to having too high a  $\chi^2$ . The state filtered with the genuine stubs matches to each stub and is output.

## Matrix Mathematics

Equations 2.4 to 2.12 were implemented using MaxCompiler. Fixed-point arithmetic is used throughout, the tuning of which contributed significantly to the time for design completion. Floating-point arithmetic, as commonly used on CPUs, requires more FPGA resources and results in a longer latency than fixed-point. This is due, mostly, to the denormalisation and renormalisation steps required before and after an equivalent fixed-point step. In the FPGA, arbitrary bit widths are possible, and the radix point can be placed wherever desired. However, constraints arise from the DSP and BRAM port widths, which must be respected for a result with efficient resource uses and high clock frequency.

The DSP units of the Virtex 7 FPGA have one port of 18 bits and another of 25 bits<sup>†</sup> [25]. This allows for an  $18 \times 25$  bit multiplication of two's complement, fixed-point quantities. It is possible to tile multiple DSPs together to multiply wider quantities. For example, the product of a 25 bit ' $a$ ' with a 35 bit ' $b$ ' can be achieved with two DSPs [81, p. 4]. The first DSP receives the lower 18 bits of  $b$ , and the

<sup>†</sup>The Ultrascale generation devices, which are likely to be used in the final system, have DSPs with one 18 bit input and one 27 bit input [80].

full 25 bits of  $a$ . The second then receives the upper 17 bits of  $b$ , the 25 bits of  $a$  and a carry bit from the first DSP. The need for a carry signal to propagate when chaining DSPs this way results in longer latencies for multiplication wider than the single DSP port widths.

The Xilinx BRAM component is a memory with a configurable data depth and address width. The basic unit is an 18 Kb memory, with data depth  $d$ , and address width  $a$  such that  $d \times a = 18432$ , with  $d \leq 36$  [26]. Multiple BRAMs may be combined to make a larger capacity memory. Given these constraints, data representations of 18, 25, and 35 bits are commonly used in the KF implementation.

When using fixed-point representation, the dynamic range made possible by floating-point is lost. Care is therefore required when assigning data types that the full range of a parameter's possible values can be represented with sufficient precision. When constructing the KF implementation, two techniques were used to choose data types: known data ranges, and numerical profiling. During construction of the data types, a distinction was also made between quantities internal to the state update, and those which are propagated to the next iteration, including the state itself. Data propagated to the next iteration requires queueing in a memory, as shall be discussed further. To reduce the memory requirements of the queue, smaller widths are desirable. At a minimum the four state parameters, six unique covariance matrix elements, and the  $\chi^2$  must be stored, which easily overflows the 36 bit data depth maximum of a single BRAM when encoded with any reasonable precision. Several ID fields are also required to associate states with stored stubs.

All of the state parameter types could be tuned by physical constraints. For  $(2R)^{-1}$ , the range is bounded by the on detector  $p_T$  cut:  $|p_T| > 3 \text{ GeV}$ , or equivalently  $|(2R)^{-1}| > 189.9 \text{ mm}^{-1}$ . The initial angle of the track in the  $r - \phi$  plane,  $\phi_0$ , is measured relative to the processing sector boundary, of which there are eight, and therefore bounded by  $-\pi/16 \leq \phi_0 \leq \pi/16$ . The Phase II Upgrade outer tracker extends up to  $|\eta| < 3$ , constraining  $\tan \lambda$  to  $-10 \leq \tan \lambda \leq 10$ . Finally, the beam spot is constrained to  $-150 \text{ mm} \leq z_0 \leq 150 \text{ mm}$ .

Since, when using the DSP resources, there is no resource advantage to using bit widths of less than 18 bits, this width was chosen as the default, with more used where necessary. It was found, for example, that 18 bits was insufficient for the diagonal covariance matrix elements. Compared to its initial value, the element  $C_{22}$ , which is  $(\sigma_{\phi_0})^2$ , decreases by a factor approximately  $10^{6.5}$  when the fit converges. Since  $10^{6.5} \approx 2^{22}$ , 18 bits cannot represent the full range without saturation at one end. Instability in the fit was observed when encoding with 18 bits, manifesting

as covariance matrices which were not positive semi-definite: a required feature for a covariance matrix. For the diagonal elements, 25 bits were therefore used. Off-diagonal elements, which are initialised to zero, were found to be adequately represented with 18 bits.

In order to accurately compute the 25 bit diagonal, 18 bit off-diagonal covariance matrix, 35 bits were used for matrices in the update path of the covariance matrix:  $(\mathbf{R})^{-1}$ ,  $\mathbf{K}$ , and  $(\mathbf{I} - \mathbf{KH})$ . A copy of these matrices truncated to 25 bits was used for the state and  $\chi^2$  update of equations 2.9 and 2.11, in order to save resources, since the extra bits were not needed for these calculations.

Matrix multiplications are performed in the shortest possible latency, at the expense of resources. This requires simultaneous execution of all multiplications, with an adder tree for the result, shown schematically in Figure 4.8. MaxCompiler optimises out operations such as multiplying by a constant ‘0’ or ‘1’, and adding ‘0’. This saves many DSPs, in particular, owing to the form of the  $\mathbf{F}$  and  $\mathbf{H}$  matrices. The many constant valued elements of these matrices can be optimised out of operations, and propagated to subsequent calculations.

A custom division algorithm was used for the matrix inversion, which requires 1 BRAM, 2 DSPs with a latency of 20 clock cycles, and has a worst case accuracy of 16 bits. The algorithm is described in detail in Appendix A. After updating the state and covariance matrix according to equations 2.4 to 2.12, the MaxCompiler block additionally computes the validity of the state according to the previously stated parameter cuts. This set of calculations to update the state contributes the majority of the latency of any single component of the Kalman Filter in the FPGA.

The numerical deviation of the state parameters computed by the FPGA firmware with respect to the simulation software is shown in Figure 4.7. The firmware output is compared to both a double precision floating point software implementation, and a handmade emulation of the fixed point operations. The fixed point implementation can be seen to introduce differences compared to the double precision – which encodes numbers with greater precision and dynamic range than any of the fixed point types used – however the tracking performance is not significantly affected, as will be seen in Section 4.1.7. The fixed point software emulation better matches the output of the FPGA, and can be used to reproduce the output on standard CPU processors.

The use of MaxCompiler was advantageous for the development of the matrix maths firmware. The automated pipelining and scheduling provided by the tool completely removes a large overhead in the development phase compared to a VHDL

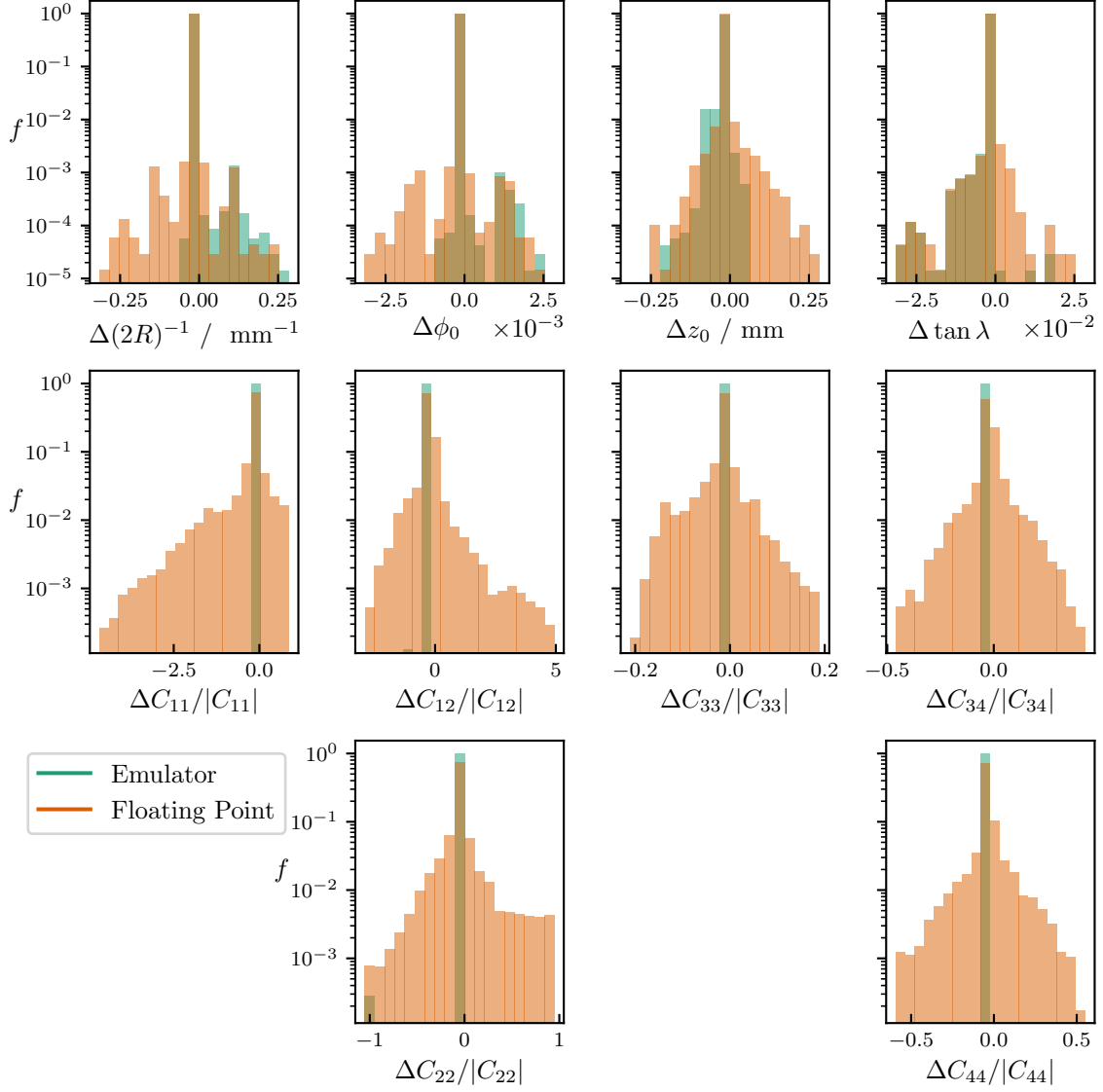


Figure 4.7: Histograms of numerical differences between the firmware and both of the fixed-point emulator, and floating point software, for each state update. On the top row are the track parameters of the state  $x$ , where the difference shown is  $\Delta x_i = x_{\text{SW}} - x_{\text{FW}}$ . The remaining plots are the relative differences of the non-zero covariance matrix elements:  $\Delta C_{ij}/|C_{ij}| = (C_{ij,\text{SW}} - C_{ij,\text{FW}})/|C_{ij,\text{SW}}|$ .

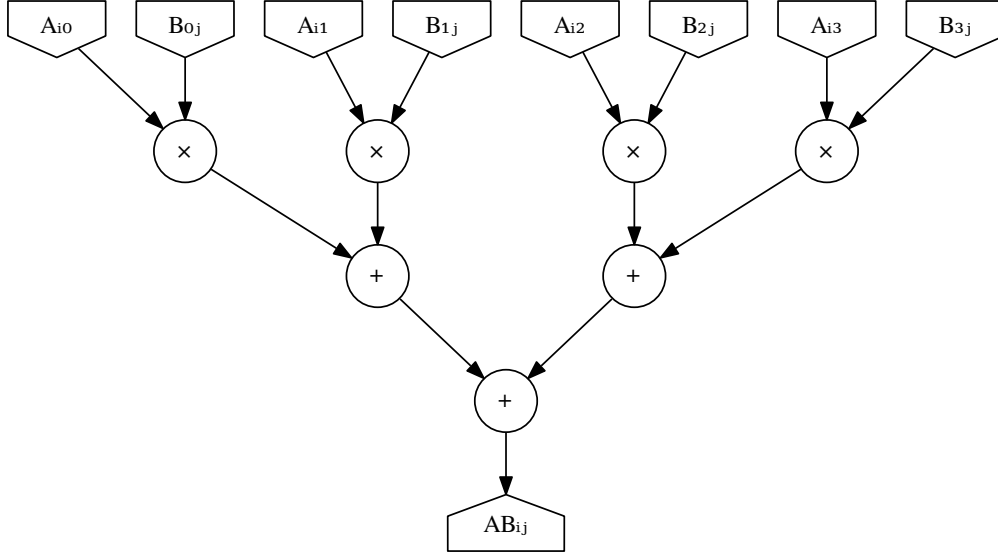


Figure 4.8: A maximally parallel matrix multiplication dataflow graph for the  $ij^{\text{th}}$  element of the product  $\mathbf{A} \times \mathbf{B}$  where  $\mathbf{A}$  and  $\mathbf{B}$  have dimensions  $n \times 4$  and  $4 \times m$  respectively.

design. Certain equations of the update 2.4 to 2.12 can be computed in parallel, while others must be calculated sequentially using intermediate results. In a hardware description language this requires the construction of the pipeline, and the precise delay to each variable must be input by hand. This can make such a design significantly more time consuming to implement.

MaxJ benefits from being an object oriented language, which permits the use of classes and overloaded methods to implement a set of linear algebra equations such as the Kalman Filter. Conversely, VHDL does not support user defined objects, so constructs such as matrices must be represented and manipulated using built-in functionality such as arrays. Some encapsulation is possible with VHDL, however properly developed object oriented code is more readable and maintainable.

The dataflow graph of the implementation of the matrix maths is shown in Figure 4.9. The division operation required for the inversion of the matrix  $\mathbf{R}$  require a significant number of sequential steps, shown in the two long sequences of operations towards the beginning of the graph. Once  $\mathbf{R}^{-1}$  is obtained the calculation of  $\mathbf{K}$ , followed by  $x_k$  and  $\mathbf{C}_k$ , proceed with fine grained parallelism with each vector or matrix element computed simultaneously. This is visualised by the horizontal extent

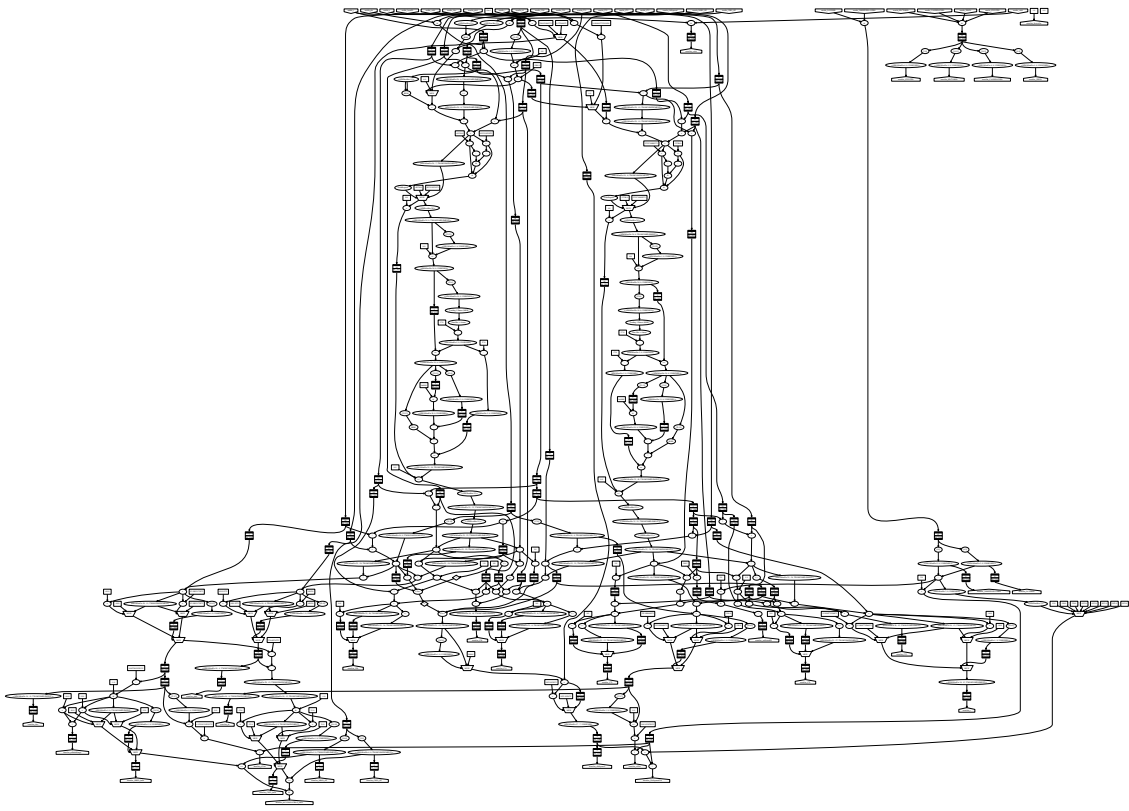


Figure 4.9: Dataflow graph of the implementation of the Kalman Filter state update Equations 2.4-2.12 with inputs at the top and outputs at the bottom.

of the dataflow graph after the division operations.

### Control Logic

The previous section described the update of a state given a stub. The control logic orchestrates the movement of states and stubs to the updater, handling the updated state, and eventual output of completed tracks. Figure 4.10 is a schematic of the connectivity of the control logic elements within the KF worker node. Their operation is summarised as follows:

- Stub packets stream from the HT on the input. They are written into a memory for later retrieval, since a single iteration of the updater takes many clock cycles.
- HT stubs carry the  $(m, c)$  indices of the cell in the array, which the Seed Creator converts to the state parameters given by Equation 4.11. A unique ID for the candidate is generated by the Seed Creator, and used to reference the stored stubs later.
- The State Control arbitrates between the states promoted by the Seed Creator, and queued working states at its other input. This allows new candidates to be started at the same time as others are processed. The State Updater accepts one state-stub pair per clock cycle, so the arbitration is necessary. Priority is given to new candidates.
- The unique IDs stored on the state are used by the State-Stub Associator to retrieve the stubs from memory required for updating the current state in question. A field of the state data specifies how many iterations it has been through, and the block reads stubs belonging to the candidate from the next layer (or on the subsequent layer if applicable), one per clock cycle. The number of layers skipped for this state is contained in another field, which determines whether stubs from one, or two layers, are read. The State-Stub Associator determines when all stubs on the layer have been read using the ‘last in layer’ flag set on the appropriate stub.
- The associated state-stub pair are input to the KF State Update block, which performs the matrix maths operations previously described, outputting the updated state.

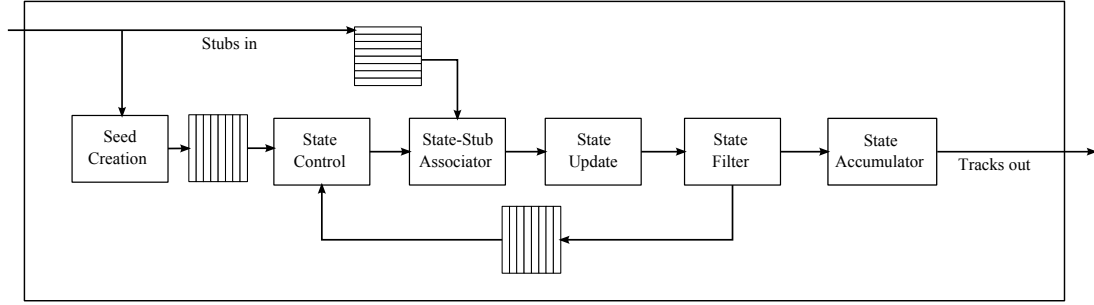


Figure 4.10: A schematic of the main components of the Kalman Filter processing node. Stubs arrive from the HT at the input on the left. Tracks are output from the State Accumulator on the right.

- The validity of the updated state is determined by the State Filter. Requirements on the  $p_T$ ,  $\chi^2$ ,  $z_0$ , compatibility with sub-sector, and PS-module layer are applied. A state failing any cut is not continued, and its processing finishes. Before writing surviving states into the state queue, the State Filter accumulates the  $N$  states for that candidate at the current iteration with the lowest  $\chi^2$  (separately for states in which a layer has been skipped, and those in which one has not). On the first iteration,  $N = 4$ , after which  $N = 1$ . Any additional states with greater  $\chi^2$  are discarded. Limiting the number of circulating states helps avoid the possibility of a processing timeout, while ordering by  $\chi^2$  ensures the most compatible candidates are kept.
- After the State Filter, the surviving states are written into a queue for further iteration. Tracks are completed after four iterations of the filter, after which they are no longer circulated in the queue. The State Accumulator maintains a copy of the best current state for each candidate in the node. Preference is given to the state with the most iterations, then with the fewest skipped layers, and then with the smallest  $\chi^2$ . If processing for a candidate is completed before the timeout (described below), this state in the accumulator is therefore the best possible for the candidate. In the event of a timeout, a partially completed track can be read out, which may occur for candidates found in dense jets with many stubs.

All of the algorithm components are implemented in pipelined, fixed latency blocks in the FPGA, and operate at 240 MHz. This gives the worker node a significant amount of pipeline parallelism, such that multiple track candidates, and all of the combinations of stubs and states, can be processed simultaneously, in a data stream. The matrix maths involved in the state update contributes the most to the



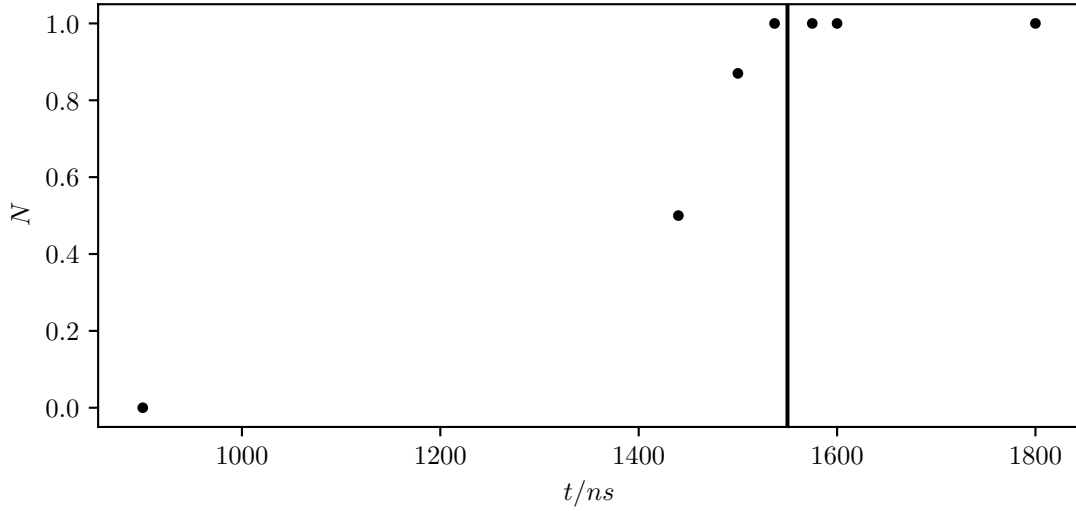


Figure 4.11: Points show the fraction,  $N$ , of track candidates which pass through four iterations of the KF state updater as a function of the processing timeout,  $t$ . The line is the chosen timeout used in the demonstrator system, 1550 ns.

latency, with 55 clock cycles (230 ns) taken to output the updated state. Since the state update accepts one stub-state combination per clock cycle, and subsequent combinations are streamed consecutively, the total time for reconstruction with four iterations is a little more than four multiples of the single iteration latency. The total time depends on the exact configuration of stubs on layers in the candidate, and on whether other candidates are being processed in the same node. The processing of some tracks also begins later than others, since the packet of track candidates from the HT contains some gaps. Figure 4.11 shows the fraction of tracks which complete four iterations as a function of time. In order to keep the track finder total processing within the latency constraint of 4  $\mu$ s, a timeout of 1550 ns is imposed on the KF. After this time all fitted tracks in the accumulator are read out. More than 99.9% of tracks are completely reconstructed in this time, in events of  $t\bar{t}$  with 200 PU.

Table 4.3 shows the FPGA resource consumption of one worker node. Since a single node uses at most 2% of each type of resource, it was possible to create 36 instances within one FPGA, operable at 240 MHz. This amounts to one worker per input from the HT, which avoids any multiplexing or arbitration of candidates to nodes. Since two MP7 boards are used for the HT in the TFP, each outputting track candidates on all links, two MP7 boards are required for the KF (since the number of inputs on an MP7 is the same as the number of outputs).

Table 4.3: Kalman Filter resource utilisation, shown for the matrix maths (and supplementary calculations) of the State Update, and for the whole worker node which includes the logic to complete the algorithm. Percentages are the fractional utilisation of a Xilinx Virtex-7 XC7VX690T FPGA. One TFP is served by 72 KF instances connected to 36 HT arrays.

	<b>LUTs</b>	<b>DSPs</b>	<b>FFs</b>	<b>BRAM (36 Kb)</b>
State Update Block	4014 (0.9%)	70 (1.9%)	3094 (0.4%)	6 (0.4%)
One Kalman Worker	5520 (1.3%)	71 (2.0%)	4370 (0.5%)	24.5 (1.7%)

As shall be discussed in Section 4.1.7, the KF greatly reduces the number of tracks compared to the number received from the HT due to its ability to identify and reject fake tracks. The data size of a fitted track candidate is also smaller than the size of the HT candidate, since only the track parameters and quality information is kept, and the stubs are discarded. The tracks remaining after the Kalman Filter still may contain some duplication of tracks, which are finally removed in the subsequent Duplicate Removal step.

#### 4.1.5 Duplicate Removal

After the KF, over half of the fitted tracks are duplicates of other tracks. These are created by the HT, and their elimination is based on the mechanism by which the HT creates them. In Figure 4.12, a track with five stubs creates three track candidates due to the overlap of the stub  $(m, c)$  lines with three HT cells. Since the stubs are the same for each candidate, the parameters fit by the KF will be the same. The fitted parameters, which have a higher precision than the HT cells, will therefore all lie in the same HT cell range. Tracks whose post-KF  $(m, c)$  bin index do not match the  $(m, c)$  bin in which the HT found them are likely to be duplicates, and can be removed. Due to resolution effects, it is possible that the KF fits a track candidate, which is not duplicated, with parameters falling in a different HT bin to the one in which it was originally found. To avoid losing these tracks the DR proceeds in a two stage algorithm.

In the first step, all tracks whose KF fitted parameters are different from the original HT cell are marked. All tracks which are fit to the same bin as the one in which they were originally found are kept at this stage, and the cells in which these tracks exist are marked. The second pass compares the HT cells of those tracks with consistent KF-HT parameters, with the remaining tracks. Tracks in HT bins not marked in the first phase are also read out. After this two classes of tracks

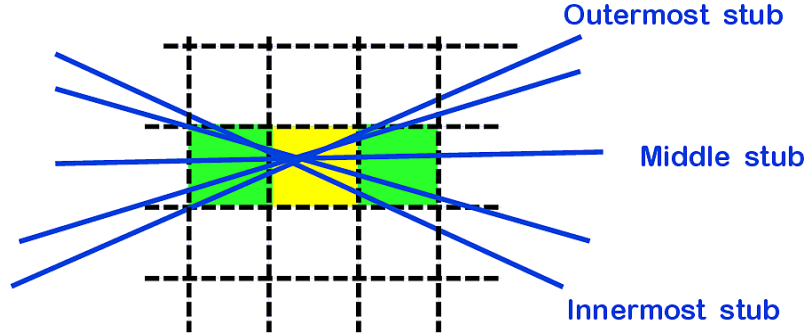


Figure 4.12: An illustration of one track producing three candidates in the Hough Transform. The central, yellow, cell is considered the correct cell, and the Kalman Filter parameters will most likely agree with this cell for all three track candidates.

remain: those for which the track fit yielded parameters consistent with the HT cell in which the track was initially found; and those for which the fitted parameters are inconsistent with the original HT cell, but where no other track consistent with that fitted HT cell was found.

Figure 4.13 illustrates the FPGA implementation of the algorithm described. Track candidates, already marked by the KF with the HT cell index of the fitted parameters, and the consistency with the original HT cell, are streamed into the DR. Tracks which are consistent are forwarded to the output stream, and the HT cell index is marked as containing a track in the memory ('MatrixA' or 'MatrixB' in the Figure). Inconsistent tracks are instead buffered in a FIFO for the second stage.

After all tracks from the KF have been streamed through the first phase, the tracks queued in the FIFO have their HT bin indices compared against the marked cells in the memory. Tracks with bin indices that were not marked in the first phase are appended to the readout stream.

Since the memories used for the HT cell mask and inconsistent track FIFO must be reset after each event, two separate memory instances are used. Each memory alternates between filling while tracks are streamed, and resetting while the other memory fills for the next event. The HT array memory is implemented with one 36 Kb block RAM: one half for the HT matrix for 6 geometric sectors, and one half containing the list of addresses which were marked in the first phase and hence must be cleared in the reset phase. The FIFO for tracks with fitted track parameters inconsistent with their original HT cell is implemented with two 36 Kb block RAMs. Table 4.4 shows the resource usage of the DR block for one instance covering six geometric subsectors. Since one TFP finds tracks in 36 subsectors, six instances of the block are required. The algorithm has a latency of only 4 clock cycles.

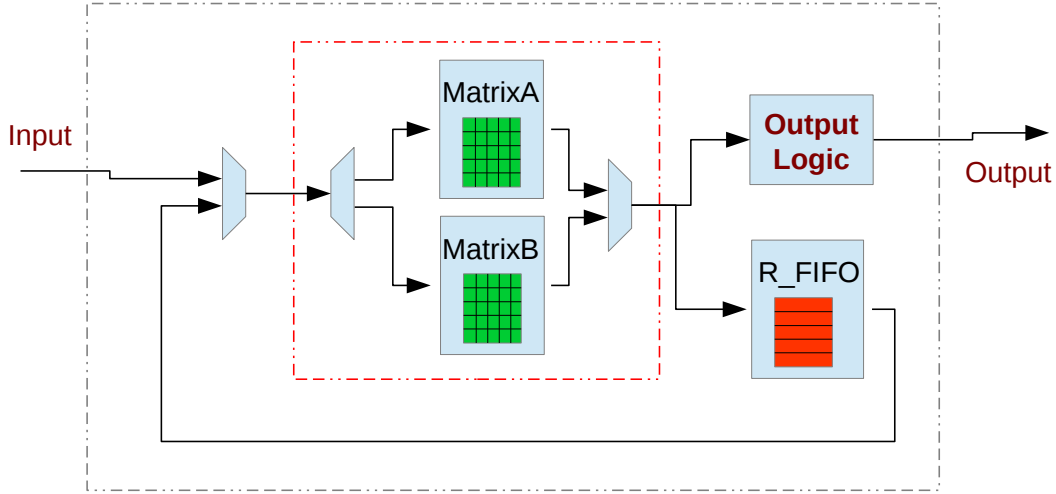


Figure 4.13: Duplicate Removal FPGA implementation architecture.

Table 4.4: Resource usage of one instance of the Duplicate Removal block, covering six subsectors, in a Xilinx Virtex-7 XC7VX690T. Six instances are required to cover the 36 subsectors used by one TFP to find tracks.

	LUTs	DSPs	FFs	BRAM (36 Kb)
One DR Block	291 (0.1%)	0 (0.0%)	496 (0.1%)	4 (0.3%)

The DR is the last step in the proposed L1 track reconstruction procedure. At the end of the processing chain in the full trigger system, the collection of tracks (represented by their four helix parameters,  $\chi^2$ , and supplementary data such as the number of layers that were skipped during the track fit) would be forwarded to the Correlator Trigger and used to perform particle level reconstruction by matching with the other subdetector trigger primitives; to find the primary event vertex; and ultimately to trigger the readout of the CMS detector.

#### 4.1.6 Demonstrator System

A system was constructed to demonstrate the operability and performance of the algorithms for reconstructing tracks of charged particles with  $p_T$  above 3 GeV in 4  $\mu$ s described in sections 4.1.2-4.1.5 in realistic hardware, pictured in Figure 4.14. Stubs were generated using the CMS experiment software (CMSSW) from Monte Carlo generated physics events with HL-LHC pileup conditions of 200 PU, including modelling of the detector response and particle-material interactions. The system, one TFP, demonstrates reconstruction of tracks for one detector octant, for one event in 36. Since the algorithms for each octant and each event are identical, the

system is able to perform the full track reconstruction by running over each octant sequentially.

All of the algorithms were running on eleven MP7 cards [54], housed in a MicroTCA crate. A commercial NAT MicroTCA Carrier Hub enables communication via Gigabit Ethernet over the crate backplane. Synchronisation, timing and control are provided by a CMS auxiliary AMC13 card [82]. The system utilises the core firmware of the MP7, with components for handling all IO requirements: serialisation/deserialisation of optical data; data buffering; formatting; board and clock configuration; and external communication via the Gigabit Ethernet interface.

In the so called ‘full chain’ configuration, high speed optical links connect several of the MP7s. The role of the DTCs – packaging stubs into a 48 bit format and performing time multiplexing – is performed on a PC. The stubs are sent, using IPBus [83], to two MP7s which act as large buffers, called the source cards. Each card emulates 36 DTCs, and can store stubs for 30 simulated collisions. The source cards stream data optically to the TFP, which is implemented on five of the MP7s.

The TFP cards are segmented as follows: one for the GP, two for the HT and two for the KF and DR. The GP board provides 36 optical links to each HT board, and each HT board sends stubs over 72 links to one KF board. The final board in this setup, the sink, receives input from each KF board and represents the end of the track reconstruction chain. Tracks sent to the sink are sent to the system PC for analysis. These connections are displayed in the lower part of Figure 4.14. The remaining boards are used for performing standalone tests of firmware components.

A C++ software emulation of the algorithms has also been developed, which is able to perform the same reconstruction as the FPGA system. The software was used for validation of the hardware system, and for carrying out performance studies. Fixed-point operations were used where possible, although in some places floating-point is used. The FPGA logic was approximated as closely as possible to emulate effects due to ordering of data, truncations and timeout. However, the emulation is not clock-cycle accurate, and so small differences between the software and hardware implementations arise.

## 4.1.7 System Performance

### Track Finding

The track reconstruction efficiency of the system is of critical importance. All simulated charged particles originating from the primary interaction which can pro-

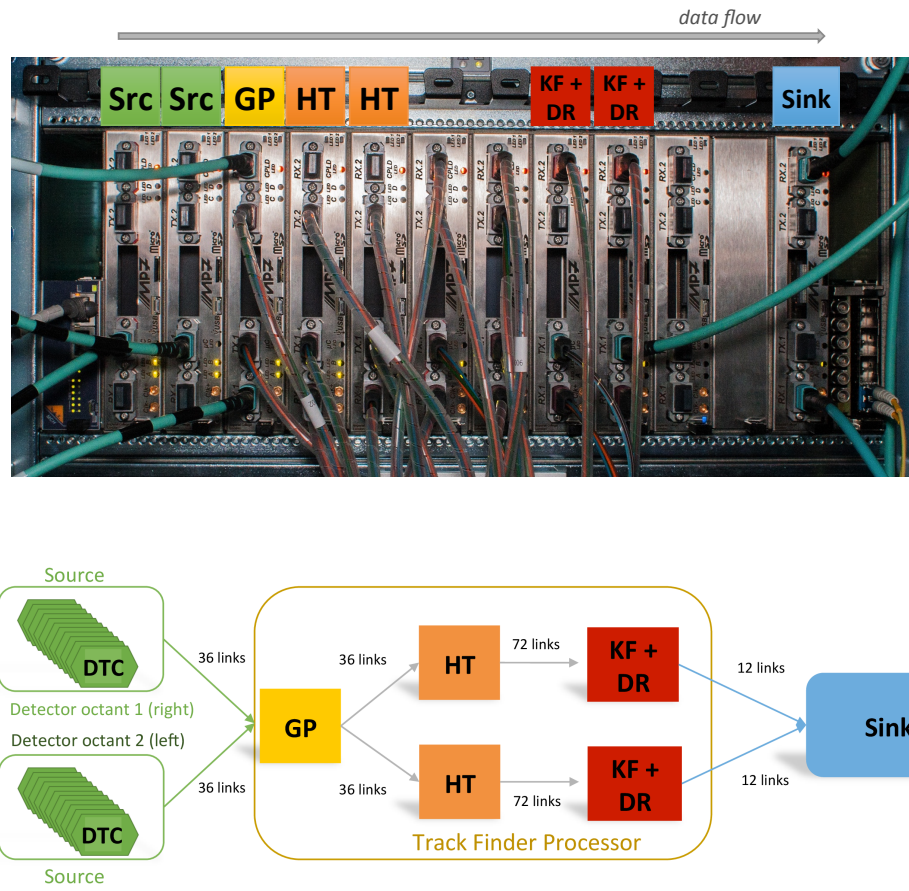


Figure 4.14: Top: photograph of the MP7s in the demonstrator crate, labelled by function. Bottom: the segmentation of the TFP algorithm across the eight MP7 boards, and the connections between boards.

duce stubs are considered for the reconstruction efficiency. That is, those with  $p_T > 3 \text{ GeV}$ ,  $|\eta| < 2.4$ ,  $|z_0| < 150 \text{ mm}$ , and  $L_{xy} < 10 \text{ mm}$ , where  $L_{xy}$  is the distance from the beamline to the particle production vertex in the  $x - y$  plane. The particle must also produce stubs in at least four different layers of the tracker to be included in the efficiency measurement. A charged particle is considered to be reconstructed if the found track has stubs associated with the particle from at least four different tracker layers, and secondly that there are no stubs associated with other charged particles. The second requirement is only imposed when considering tracks which are processed by the ‘full-chain’. Tracks found by the HT are likely to contain erroneous stubs, so only the first requirement is imposed when analysing the collection of tracks at the HT output, with the expectation that the KF may be able to remove the erroneous stubs. Tracks which are found by the system, but with stubs produced by multiple generated charged particles, are termed ‘fake’ tracks. For a charged particle which is found more than once, the extra tracks are termed ‘duplicates’. Fake tracks can also be duplicated.

Table 4.5: Track reconstruction quality at different stages of the demonstrator chain, for  $t\bar{t}$  events with 200 PU interactions. The efficiencies follow the definitions in the text. The total number of found tracks, fakes and duplicates are also reported. The fraction of the total tracks which are fakes and duplicates are given in parentheses.

Stage	Efficiency [%]	Total tracks	Fakes	Duplicates
<b>HT</b>	97.1	331	139 (42%)	126 (38%)
<b>KF</b>	95.1	190	27 (14%)	103 (54%)
<b>DR</b>	94.4	79	16 (20%)	3 (4%)
<b>Full chain</b>	94.4	79	16 (20%)	3 (4%)

Table 4.5 presents the tracking performance of the demonstrator components and full procedure when reconstructing tracks in events of  $t\bar{t}$  with 200 PU with the algorithm emulation. The HT successfully finds almost all tracks, but with a large number of fake and duplicate tracks. The Kalman Filter then removes 80% of fake tracks, at the expense of two percentage points of efficiency, equivalent to 1 to 2 genuine particles per event on average. Some duplicates are also removed by the KF, because fake tracks can be duplicated, and both identified separately as fake. All of the tracks which satisfy the first of the criteria outlined above also meet the second after the KF: that is, the KF removes all erroneous stubs from the HT candidates which it keeps. Finally the Duplicate Removal greatly reduces the number of duplicate tracks, again with a small penalty of efficiency. The mean tracking efficiency of the full chain in hardware is measured to be 94.5%: in good

agreement with the emulation.

Figure 4.15 shows the efficiency of the track reconstruction as a function of  $p_T$  and  $\eta$  for  $t\bar{t}$  events with 200 PU. The emulation and hardware are in close agreement across both variables. For all tracks there is a slight turn on at low  $p_T$ , up to around 95% efficiency, with some drop off for higher  $p_T$  tracks. Reconstruction efficiency is also best in the barrel part of the detector, with some degradation at the most forward part of the detector. Muons are reconstructed with higher efficiency than the average across all  $p_T$  and  $\eta$ , with no loss in efficiency at high  $p_T$ . Some efficiency degradation is observed in the overlap region from barrel to endcap (around  $|\eta| = 1$ ), and at the most forward part of the detector. The efficiency for reconstructing electrons is somewhat lower. Electrons lose energy via bremsstrahlung, which deviate the particle from the helix trajectory assumed by the tracking algorithm. Tracks within high  $p_T$  jets ( $p_T > 100$  GeV) are reconstructed slightly less efficiently than the average, especially for  $|\eta| > 1$ , which is the detector endcap. The many overlapping tracks increase the confusion of assigning hits to the correct track. Most of the ‘lost’ tracks are incorrect by only one stub (the other three all matched to the same generated charged particle), as seen by adjusting the efficiency criteria to allow for a single erroneous stub. These tracks are classed as fakes, but may still be of use to the trigger, albeit with a worse track parameter resolution. An improved rejection of stubs in the Kalman Filter might also recover this efficiency loss.

## Track Fitting

Figure 4.16 shows the residuals of the four track parameters, measured for both the FPGA demonstrator, and the emulation, for tracks originating from the primary vertex. The resolution is defined to be the RMS of the residual: the difference between the reconstructed and simulated parameter. Resolution worsens for all parameters at greater  $|\eta|$ . The shorter lever arm for tracks with  $|\eta| > 1$  accounts for the degradation in  $p_T$  and  $\phi$ . Emulation and hardware resolutions are in good agreement.

Figure 4.17 shows the resolution for single muons without pileup in different  $p_T$  ranges. This provides a handle on the ultimate limitation of the reconstruction, as no track trajectory is more ideal than that of a muon, and the clean environment of 0 PU guarantees there will be no contamination by other tracks. Compared to Figure 4.16 it can be seen that the resolution for single muons is relatively improved across the whole detector, with the exception of the relative  $p_T$  resolution of muons



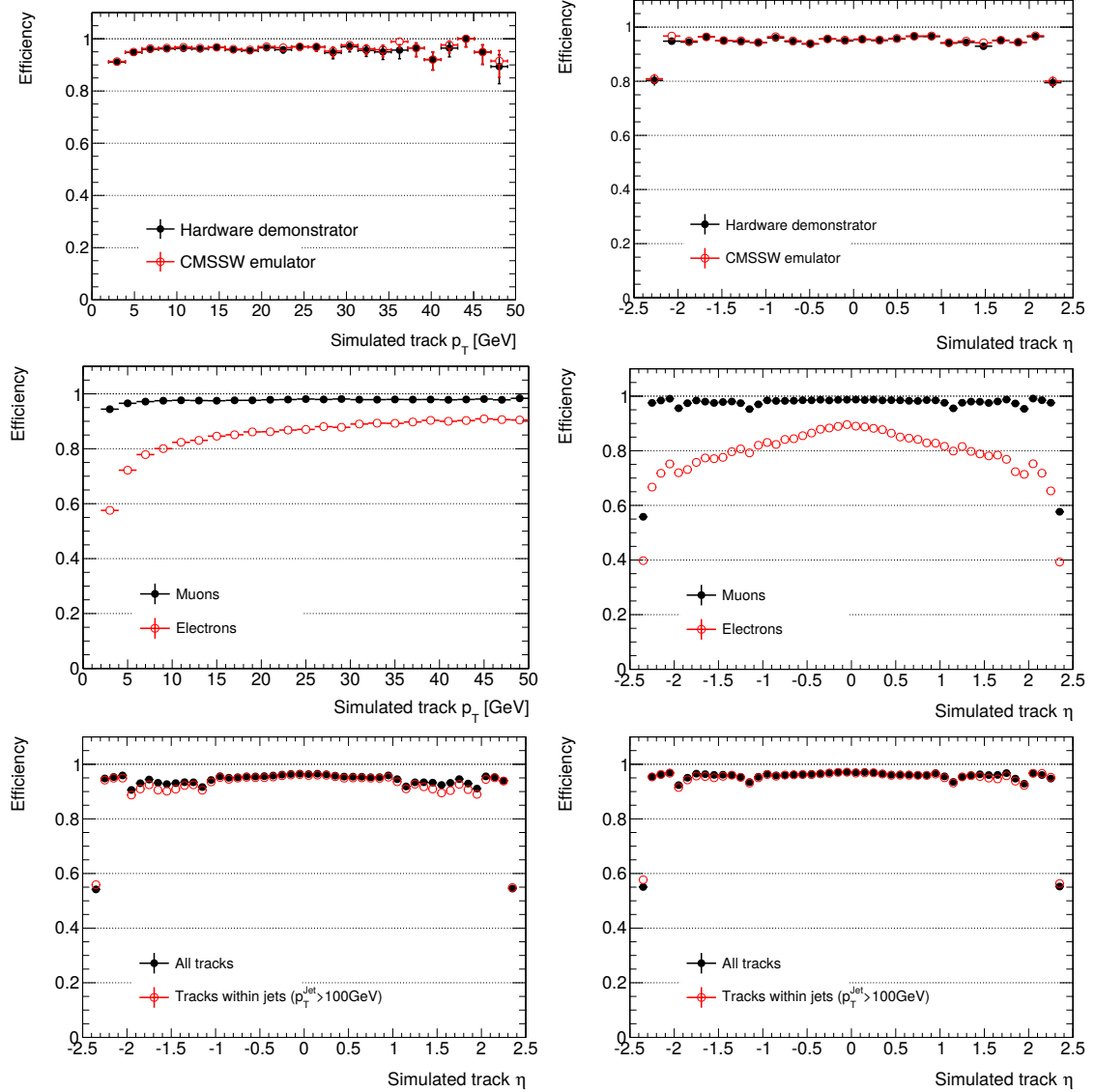


Figure 4.15: Track reconstruction efficiency of the ‘full-chain’ algorithm in  $t\bar{t}$  events with 200 PU. Top row: for all tracks originating from the primary interaction, for both the hardware demonstrator system and the software emulation, as a function of  $p_T$  (left) and  $\eta$  (right). Middle row: for electrons and muons, using the emulation, as a function of  $p_T$  (left) and  $\eta$  (right). Bottom row: for all tracks originating from the primary vertex, or only tracks contained within a primary jet with  $p_T > 100 \text{ GeV}$ . On the right plot, the efficiency definition is altered such that at most one erroneous stub is permitted on the track. The original efficiency definition is used for the left plot.

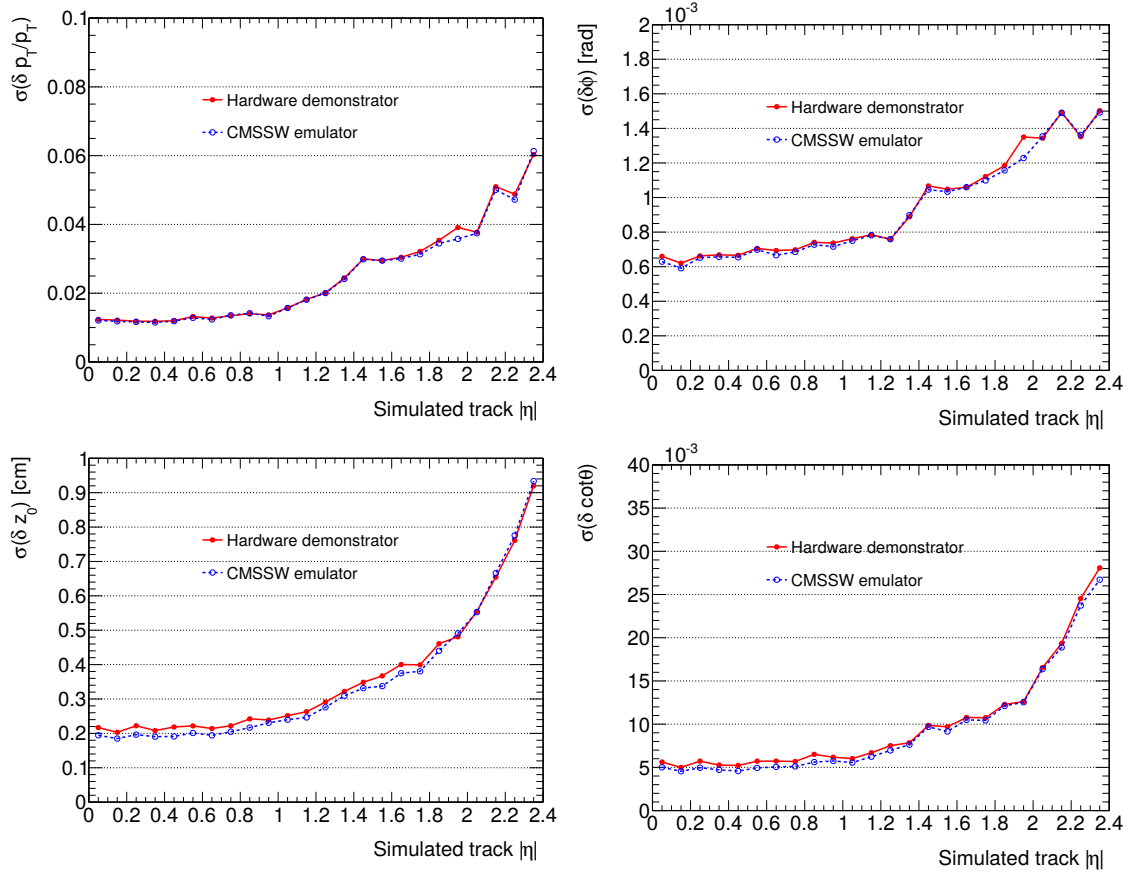


Figure 4.16: Resolutions of the track parameters as fitted by the Kalman Filter. Relative  $p_T$  (top left),  $\phi_0$  (top right),  $\cot\theta$  (bottom left) and  $z_0$  (bottom right) resolutions are shown as a function of pseudorapidity for tracks originating from the primary interaction, in events of  $t\bar{t}$  with 200 PU.

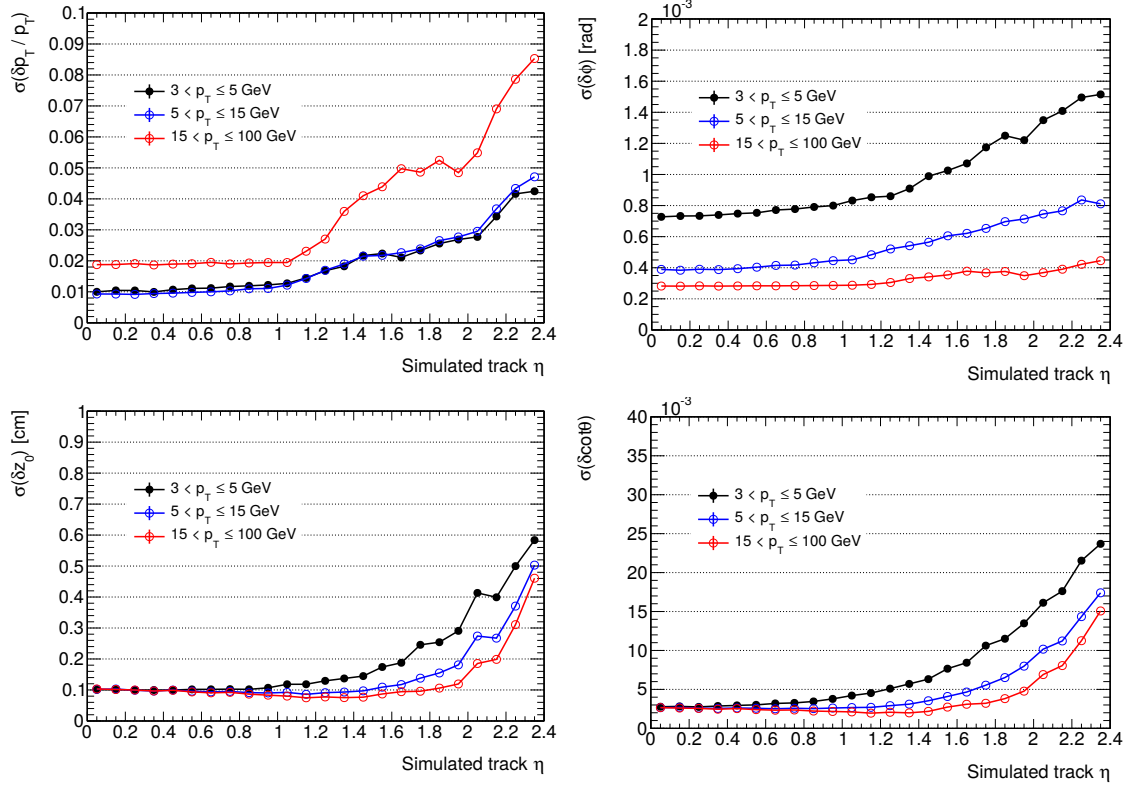


Figure 4.17: Resolutions of the track parameters as fitted by the Kalman Filter. Relative  $p_T$  (top left),  $\phi_0$  (top right),  $\cot\theta$  (bottom left) and  $z_0$  (bottom right) resolutions are shown as a function of pseudorapidity for tracks originating from the primary interaction, in events of single muons without pileup, for muons in the ranges  $3 < p_T^\mu < 5$  GeV,  $5 < p_T^\mu < 15$  GeV, and  $15 < p_T^\mu < 100$  GeV.

with  $15 < p_T^\mu < 100$  GeV. Compared to the full offline track reconstruction with the full Phase-2 tracker in [40, p. 289], for 10 GeV isolated muons, the relative  $p_T$ , and  $\phi$  resolution of the Level 1 tracking is worse by around a factor of two, while  $z_0$  and  $\cot\theta$  resolution is worse by more than a factor of ten. The PS stubs have a length of 1.5 mm in the  $z$  direction, while the pixels which are available for the full tracking, have a length of 0.1 mm, which accounts for the difference in  $r - z$  plane parameter resolutions.

#### 4.1.8 Latency

The latency of the track reconstruction is presented in Table 4.6, broken down for each part of the processing and connections between boards. As can be seen, the KF and DR are the components with the highest latency, followed by the HT. The latency of the system is measured to be less than the imposed  $4 \mu\text{s}$  limit, both for the

first and last track output by the system. This latency is constant by construction, and independent of the pileup scenario, and the specific event topology.

Table 4.6: Measured latency of the demonstrated components of the track reconstruction chain, including the serialisation/de-serialisation (SERDES) and optical transmission delays between each board.

<b>System Latency</b>	<b>Latency [ns]</b>
SERDES + optical length 1	143
Geometric Processor	251
SERDES + optical length 2	144
Hough Transform	1025
SERDES + optical length 3	129
Kalman Filter & Duplicate Removal	1658
SERDES + optical length 4	129
Total: First out - First in	3479
Last out - First out	225
Total: Last out - First in	3704

#### 4.1.9 System capacity

Since the track trigger processes zero-suppressed data (that is, only tracker channels with stubs are read out), the rate of data processed by the track finder will vary event by event. It is instructive, therefore, to explore how the number of objects varies through the processor, and where the limitations lie. Throughout the demonstrator system, several sources of data loss exist. Movement of data packets can never exceed the time multiplexing period, since data from the next event must transmit at this time. The GP incurs some loss by this mechanism: stubs enter the GP within the time multiplexing period, and pass through the formatter with no change in distribution. The routing network may spread the stub packet however. When a ‘collision’ occurs, with two stubs attempting to pass through a network node simultaneously, one must wait. With a time multiplexing period of 900 ns and a 240 MHz clock frequency, 216 stubs can transmit per link. The gaps introduced reduce this limit on average to 175 stubs per HT sector per event. As can be seen in Figure 4.18, this is approximately double the average seen in  $t\bar{t}$  with 200 PU events. The truncation loss is therefore quite small: 0.3% of stubs, which results in the loss of 0.5% of tracks.

The HT output is also sensitive to losing stubs due to transmission exceeding the time multiplexing period. The right side of Figure 4.18 shows the uneven distribution

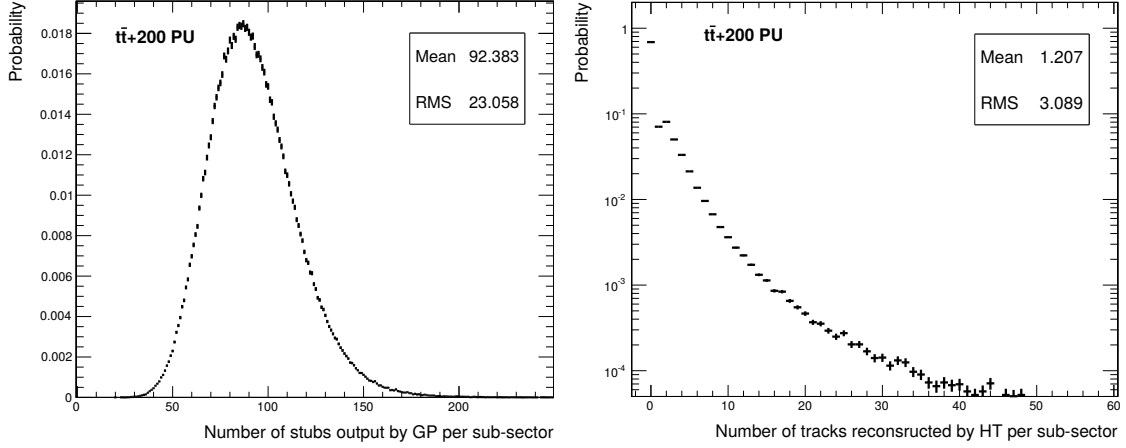


Figure 4.18: Number of stubs output per GP sub-sector per event (left) and number of track candidates found by the HT per sub-sector per event (right), both in events of  $t\bar{t}$  with 200 PU.

of found tracks across HT sectors. 70% of sectors find no tracks, and 97.5% find fewer than 10 tracks. The average number of stubs per track candidate is 7, and the 216 stubs per event restriction due to time multiplexing applies, so generally the truncation at the HT output is very small. Collimated, high- $p_T$  jets, however, may produce many tracks and stubs within a narrow region, which must all be found in one, or a few HTs. The tracks in these types of ‘busy’ regions would also tend to contain more than the average number of stubs. The load-balancing at the HT output helps to mitigate against this effect, by decoupling the output streams from geometric regions. The remaining effect is a 0.1% loss in tracking efficiency in events of  $t\bar{t}$  with 200 PU.

The final loss of efficiency due to ‘overloading’ the system capacity arises from the Kalman Filter. The mechanism by which tracks are lost is the timeout described in section 4.1.4 whereby a track may not have been completely fitted before the latency budget expires. Again, tracks in high  $p_T$  jets with many stubs are the most susceptible to this effect due to the greater number of trajectory combinations arising from the many stubs. Loss by this mechanism is less than 0.1% in  $t\bar{t}$  with 200 PU events. The net loss of tracks in the full systems is around 0.6% in events with  $t\bar{t}$  and 200 PU, and is dominated by the truncation in the GP.

#### 4.1.10 Towards a Final System

The MP7 platform was used throughout the demonstrator system, and a number of optimisations were made to fully utilise the architecture. Any production track

trigger system will certainly not be constructed with MP7s, which were first released around the year 2012, with an FPGA which first became available a year or so before. The track trigger at the HL-LHC will comprise boards using latest generation components at the time of construction, preceding 2025. Many system parameters were chosen with a view to balance tracking performance with FPGA resource usage and communication bandwidth. While some parameters have values motivated by ultimate tracking performance, the final choice takes into account the mapping onto the particular FPGA platform. Furthermore, some parameters are highly interdependent, such that a change to one requires an adjustment elsewhere. This includes for example communication bandwidth and rate, clock frequency, resource utilisation and latency. A reoptimisation of the whole track finder algorithm will likely take place to better utilise the upgraded architecture.

In 2017 Xilinx Ultrascale and Ultrascale+ parts became available. These are constructed with a 20 nm and 16 nm process respectively, compared to the 28 nm Virtex-7 chips, so are typically more cost and power efficient than their predecessor. This is likely to be the iteration used for the HL-LHC triggers, since specifications and parts for the next generation will become available only very close to the time of construction of the trigger. In particular the VU9P and KU115 FPGAs have been identified as being of particular interest for HL-LHC era triggers. The former represents a part with some of the most resources available, while the latter has the best ‘resources per \$’. The cost of an FPGA board is dominated by the FPGA itself, so to some extent a component with good cost efficiency is desirable.

Data transferred optically from detector front-ends to the trigger are received at dedicated transceivers on FPGAs. The Virtex-7 component on an MP7 is capable of  $12.5 \text{ Gb s}^{-1}$  per transceiver, while the newer Xilinx FPGAs support up to  $16 \text{ Gb s}^{-1}$  or  $32.75 \text{ Gb s}^{-1}$ , although only optical components of up to  $25 \text{ Gb s}^{-1}$  are available. Data encoding also moves from 8b/10b to 64b/66b encoding which improves the encoding efficiency from 25% to 3.125%, better utilising the available bandwidth. A  $16 \text{ Gb s}^{-1}$ , or  $25 \text{ Gb s}^{-1}$  link with the updated encoding has a 1.65, or 2.58 times bandwidth increase over one MP7 link, respectively. For the 2025 track trigger, this means that fewer links, shorter (in time) data packets, or a combination will be sufficient for data transfer compared to the demonstrator. It is anticipated that the time multiplexing factor will be halved from 36 to 18.

In the demonstrator system, the exact number of instances of certain processing steps is tightly linked to the number of optical links. There are 36 Kalman Filter worker instances, for example, since this allows a neat correspondence of links to

Table 4.7: Resource availability of the FPGA on an MP7, and of two current generation Xilinx chips which are suitable for HL-LHC triggers.

Component	Logic Cells (K)	DSPs	Memory (Mb)
XC7VX690T	693	3600	51.7
KU115	1451	5520	75.9
VU9P	2586	6840	345.9
<b>One TFP</b>	1629	16944	231

workers with 72 links and 64 bit stubs. The same preference towards multiples or integer divisors of 72 is manifest in the choice of 36 geometric processing sectors. Since no Ultrascale(+) FPGAs host exactly 72 transceivers, the number of links between boards is likely to differ for the final system. The number of instances in certain steps may therefore be slightly different for the final system.

The increase in optical communications bandwidth has a subsequent impact on the algorithm clock frequency, and to some extent the number of resources allocated. The first processing step to receive external data, the GP, must be able to absorb stubs at the same rate they are sent. The rate of arrival of stubs increases with the same factor as the updated optical connection, so the capacity of the GP algorithm must increase. Since the processing rate with  $n$  algorithm instances, and frequency  $f$  is  $r = nf$ , this can be achieved by increasing: the clock frequency of the algorithm; the algorithm parallelisation; or both. In practice  $n$  is bounded by the FPGA resources available and  $f$  has an upper limit of around 500 MHz, although the limit of  $f$  will be dependent on the algorithm, the implementation, and the total chip utilisation. The maximum frequency will likely differ for each algorithm component. Practically, a design optimised for latency, working at 240 MHz, will require additional registers to operate at 500 MHz. Experience from reoptimising the router used in the GP to 480 MHz suggests that a 30% latency reduction is achievable compared to the 240 MHz design, rather than the 50% that might be assumed without the additional registers. Assuming that the same scaling applies to other algorithm components, a system with three FPGAs, connected sequentially, operating at 480 MHz with a time multiplexing factor of 18 would achieve a latency of around  $2.5\text{ }\mu\text{s}$  from the first stub arriving at the DTC to the last track exiting the track finder.

The FPGA resources of selected Xilinx parts are shown in table 4.7, with those of the MP7 FPGA for reference. The relative increase in size is not uniform for the different resource types, but is at least 1.5 for the KU115, and 2 times for

Table 4.8: Tracking performance for both flat and tilted barrel tracker geometries using events containing a single muon with a  $p_T$  of 10 GeV and 200 pileup collisions, reconstructed using the unmodified demonstrator algorithm.

	Flat geometry	Tilted geometry
<b>Tracks after HT</b>	229	161
<b>Fakes after HT</b>	92	35
<b>Tracks after full chain</b>	55	48
<b>Fakes after full chain</b>	9	4
<b>Efficiency after full chain</b>	97.3%	97.3%

the VU9P. Resource consumption of one TFP is presented assuming the algorithm is ported to the Ultrascale architecture (necessarily including adopting  $16 \text{ Gbs}^{-1}$  optical connections) by simply doubling the resource usage to accommodate the doubling of stub input rate. This corresponds to three KU115 parts, mostly due to the DSPs and memory. As discussed, however, the rate increase can be met by increasing the algorithm clock frequency as well as the resources. A frequency of 350 MHz would sufficient to bring the resource usage of one TFP down to use only two KU115 parts. A natural splitting of the algorithm components would then be to place the GP and HT in a single FPGA, and the KF and DR into the second chip. Dividing the algorithm this way would also remove the preference towards using a number of geometric sectors which is an integer divisor of the number of links, since the distribution of stubs from GP to HT would take place within the FPGA.

#### 4.1.11 Future Developments

The adoption of next generation FPGAs described does not assume any changes to the algorithm, other than scaling the number of instances. Of course, the algorithm can be improved further. Increasing the efficiency, reducing the fake rate, improving the parameter resolution and introducing the transverse impact parameter to the fit would all improve the tracking performance. Separately, improvements to the load balancing of work within the system would allow the trigger to become less susceptible to particularly populated events, and would also potentially allow FPGA resources to be saved.

One development which is necessary is to adopt the updated tracker design which uses tilted modules at the ends of the barrel to reduce the number of modules required for continuous coverage. This layout is depicted in Figure 4.19. A test on a preliminary Monte Carlo sample using this geometry, and without modification of



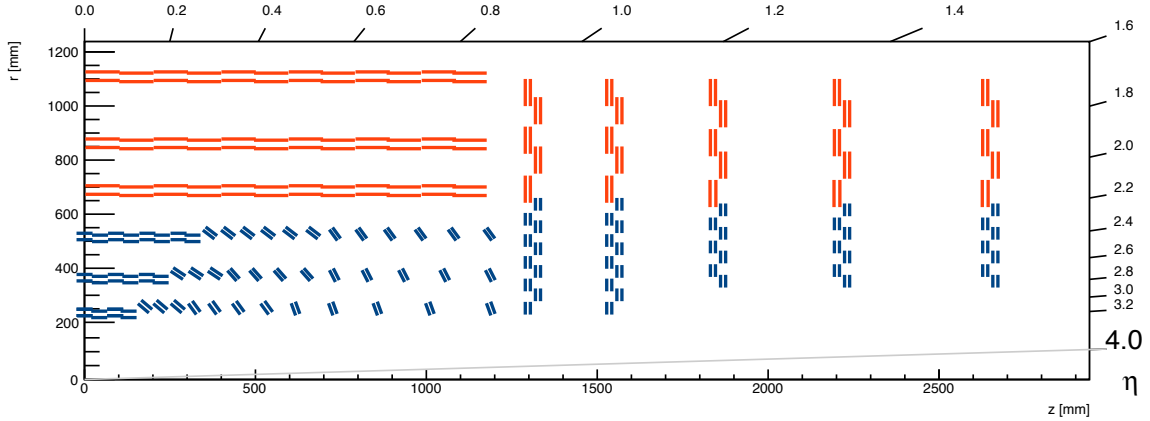


Figure 4.19: One quarter of the  $r - z$  plane of the CMS outer tracker with tilted modules in the barrel [12].

the track reconstruction yields the results in table 4.8. The change makes an immediate reduction to the number of fake tracks found by the HT, without affecting the efficiency, due simply to a reduction in the number of stubs from tilted modules. With fewer tracks found by the HT, there will be additional spare capacity in the track finding system, which should allow for a resource saving, for example by reducing the number of KF nodes for track fitting.

In the demonstrator system the GP is responsible for the loss of 0.5% of genuine tracks (although this might improve with the aforementioned tilted geometry). As discussed, this arises from gaps introduced in the data packet as stubs are routed from input to geometric sector through a routing network. A change to the design of this network, with more parallel links between routing layer, might be less susceptible to this loss. This would come at the cost of more FPGA resources.

The  $T$  parameter was introduced in Section 4.1.3 to transform the stub radius  $r$  to  $r_T = r - T$ , such that stubs with  $r < T$  produce lines in track parameter space with a negative gradient, which improves the separation of tracks in the rectilinearly sampled space, and yields a lower fake rate without losing efficiency. The chosen value of  $T = 580$  mm allows approximately equal numbers of stubs with positive and negative line gradients in the barrel. In the endcap, however, tracks do not reach the outermost radii of the tracker, so this choice of  $T$  biases towards negative gradient stubs. A better separation can be achieved for the endcap by choosing a smaller value of  $T$  for this region to again improve the separation between tracks. Setting  $T = 470$  mm at the highest  $|\eta|$  sectors reduces the number of tracks found in those sectors by a factor of two. The Hough Transform might be further improved by altering the shape of the HT cells, since fakes occur due to the accidental intersection

of several stub lines with a bin. Using hexagonal bins has been seen to reduce the number of tracks found by 20% without adversely affecting the efficiency [84].

The Kalman Filter implemented for this development is significantly simplified compared to the algorithm used in the reconstruction at HLT and offline. Steps to introduce missing features ought to improve the tracking performance further, without requiring a complete redesign, but at the expense of some resources and possibly latency. Adding the transverse impact parameter to the track fit would improve the performance for tracks not originating from the beam line, notably those from B hadron decays. The state would become a five parameter vector, and certain matrices would increase in dimension, thus requiring some extra FPGA resources for matrix operations. The treatment of multiple scattering within the Kalman Filter ought to improve the efficiency at a given fake rate. Scattering can be included within the Kalman Filter formalism, and would require some additional computation of the most probable scattering angle. A slight modification to the parametrisation would also be beneficial: from including the initial track position  $(\phi_0, z_0)$  in the state, to the current position  $(\phi, z)$ . An additional step to recover the track parameters at the vertex would then also be needed. The linear approximation to tracks is a good one across most of the parameter space of the track trigger, but introduces a 1.5% error in  $\phi$  for the lowest  $p_T$  tracks at the outer barrel layer. Evaluating the arcsin function exactly would require some extra FPGA resources, and would potentially provide better discrimination for fake tracks at low  $p_T$ .

Section 4.2 describes a development to reduce the number of fake tracks at the output of the track reconstruction.

## 4.2 Identifying Fake Tracks with a Boosted Decision Tree

A means to identify fake tracks from the Kalman Filter output, shown in section 4.1.7, is desirable. The presence of fake tracks in the reconstruction is unavoidable in a high pileup environment, but can only negatively effect the trigger performance, so any mitigation may be worthwhile. Fakes, particularly those reconstructed with high  $p_T$ , would increase the trigger rate, and potentially mimic a physics signal. As with all trigger algorithms, a low latency, low resource usage in the FPGA is also desirable. Further, ideally only quantities which are output by the track trigger, or which require minimal calculation from those outputs, should be used to identify fake

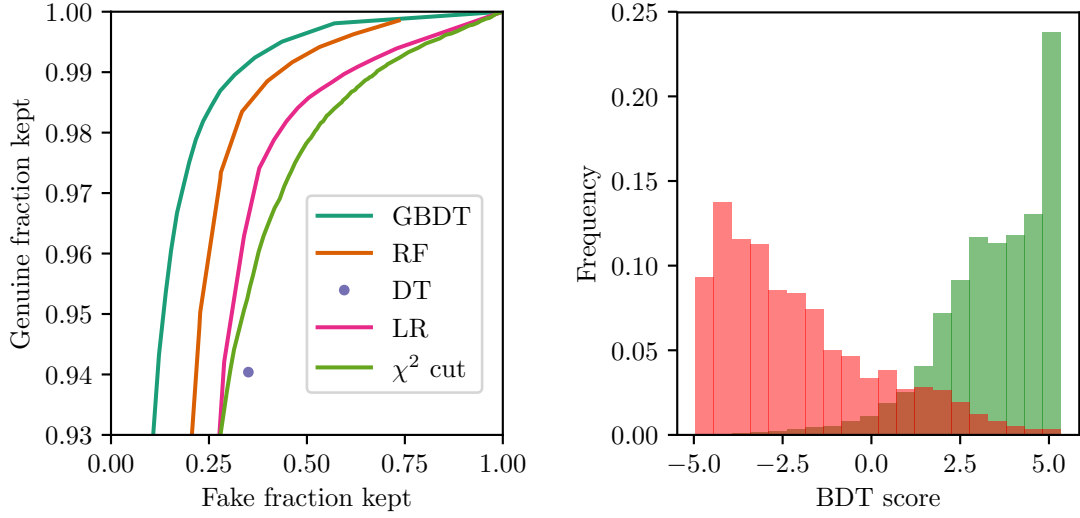


Figure 4.20: Left: Receiver Operator Characteristic (ROC) curve for the five classifiers under investigation: Gradient Boosted Decision Tree (GBDT); Random Forest (RF); Decision Tree (DT); Logistic Regression (LR);  $\chi^2$  cut. The desired working point is at the top left: all genuine tracks kept, and no fake tracks kept. By tuning a cut on the classifier score above which all tracks are labelled ‘fake’ a working point can be selected. Right: histogram of separation between genuine (green) and fake (red) tracks as a function of BDT score.

tracks. The identification of fake tracks is now treated as a classification problem.

A selection of multivariate classifiers were trained on Kalman Filter output tracks using the SCIKIT-LEARN package [85]. The  $\chi^2$ , number of skipped layers,  $\tan \lambda$ , and  $(2R)^{-1}$  of the track were used as features, with ‘fake’ and ‘genuine’ classes labelled from the Monte Carlo truth information. Receiver Operator Characteristic (ROC) curves for the classifiers are shown in Figure 4.20, by varying the classifier output threshold at which to label a track as fake. All of the multivariate classifiers perform better than the  $\chi^2$  cut only baseline. A Gradient Boosted Decision Tree (GBDT) is the most performant of those investigated, with an area under the ROC curve of 0.966. A loss of genuine tracks is undesirable, so a conservative cut is required. The GBDT can remove around 75% of the fake tracks while retaining 99% of the genuine tracks. The GBDT ROC curve never crosses that of another classifier, i.e. it yields the best (lowest) fake fraction at every efficiency working point. Furthermore, the GBDT was deemed to be among the most suitable to be implemented in an FPGA for a number of reasons, which will be outlined.

The distribution of fake tracks in  $(2R)^{-1}$  and  $\tan \lambda$  (the proxies for  $(p_T)^{-1}$  and  $\eta$  which are available in the demonstrator hardware) can be seen in Figure 4.21,

alongside the ratio of fakes which are identified as genuine to all fakes. The BDT is most effective at identifying fake tracks in the tracker barrel-endcap transition (around  $|\tan \lambda| = 2$ ), and those with high  $p_T$  (low  $(2R)^{-1}$ ).

The GBDT inference algorithm has aspects which suggest it should be implementable within an FPGA. The output of the GBDT, the score, is given by equation 4.16.

$$F = \sum_i \beta_i h_i(x) \quad (4.16)$$

where  $x$  are the input features,  $h_i$  are the decision trees and  $\beta_i$  are the tree weights. The inverse of the loss function used in training can be used to obtain the probability for each class from the score. In a scenario where the BDT output is used to keep or discard tracks, this function need not be computed, as the score can be used to cut directly. The decision tree score,  $h_i$ , depends on which leaf node the feature vector ends in. The value of the score for each leaf is set during training.

Each  $h_i$  in equation 4.16 is independent of any other. This is the first opportunity for parallelism:  $h_i$  can be computed simultaneously for all  $i$ . Within the decision trees, if the node is considered as a processing unit, each node consists of some processing which is independent of any other – the comparison of a feature with a threshold – and one part which depends upon the result of other nodes – whether or not the decision path passed through the node’s parent.

Other works have implemented Decision Tree ensembles for FPGAs previously [86–90], although since generally these target ‘Big Data’ applications, they tend to be optimised for throughput or power consumption over latency. Some implementations [87, 89] target ensembles larger than the capacity of the target FPGA, developing a base decision tree unit onto which the model is loaded from a memory. Such an approach adds flexibility, and classification performance by supporting large ensembles, at the expense of latency and resource overheads. BDTs have been developed with low latency for the L1 muon trigger of CMS [91]. An external memory of 1.2 GB is used as a look up table to assign the muon  $p_T$ . This approach requires few FPGA resources, but does require an external memory, with the associated latency to retrieve data. To maintain a reasonable memory size, the features are significantly truncated: 30 bits (which forms the address) are used to encode five features. The approach used here is to fully map all of the operations onto discrete FPGA components, to achieve the shortest latency without restricting the number or width of features used.

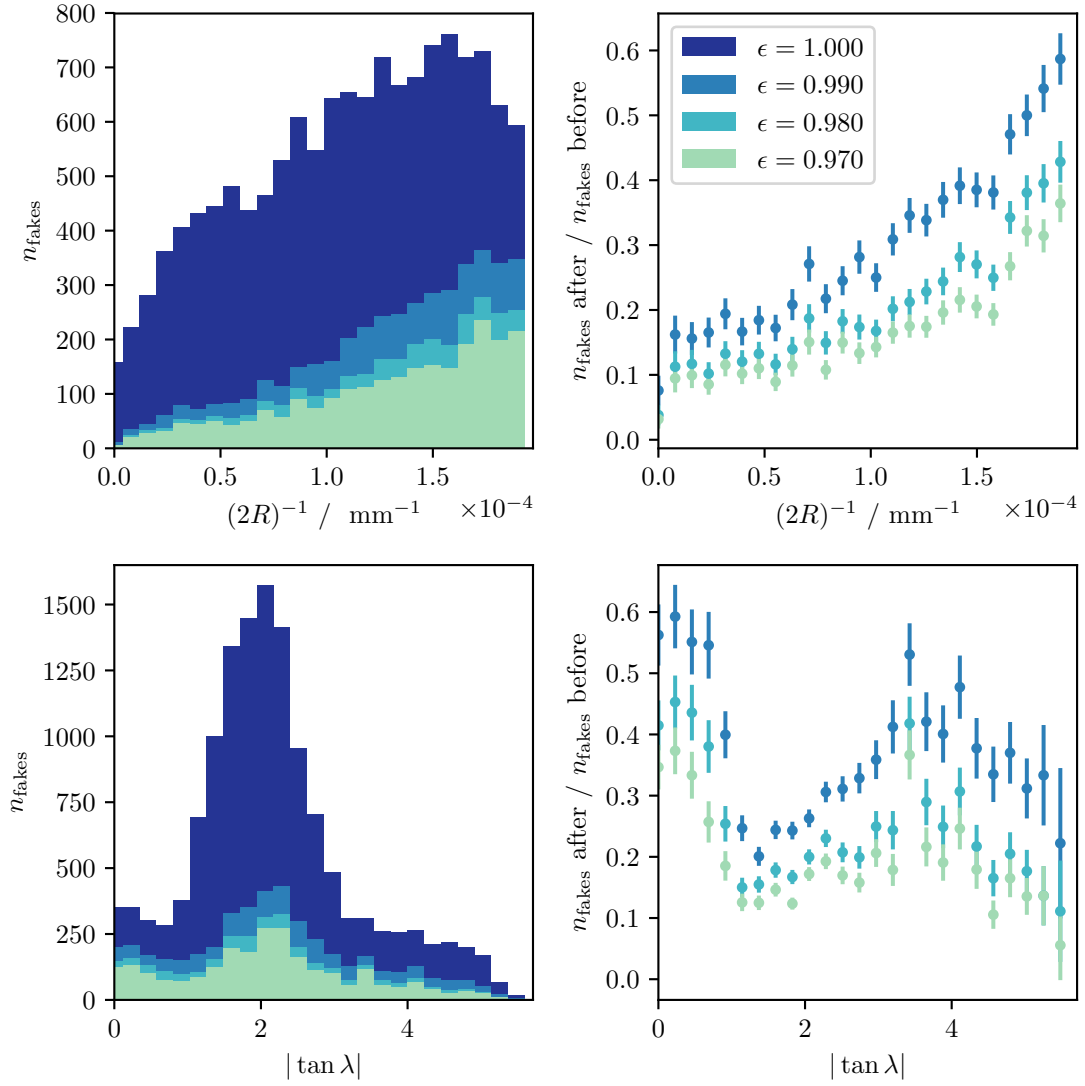


Figure 4.21: Distributions of fake tracks before and after BDT class prediction, and the fraction which remain, as a function of  $(2R)^{-1}$  (top left, top right respectively) and  $\tan \lambda$  (bottom left, bottom right respectively) at different BDT score cuts, selected to yield one percentage point decreases in efficiency between cut values.

### 4.2.1 FPGA implementation

In order to evaluate the suitability of the BDT for a trigger application, a MaxJ firmware was developed. The design maximised the trivial parallelisability of BDTs, discussed in section 4.2, by performing all comparisons, and evaluating all trees in parallel. Figure 4.22 shows schematically how a single decision tree is implemented in the FPGA. All of the comparisons execute in parallel, and a chain of parallel boolean operations is constructed for each leaf node such that the decision path is encoded in an array of boolean values. The leaf boolean activation values are used to select the appropriate leaf score to output for the tree from the small table of possible values. Within the tree, each operation is pipelined, allowing a new feature vector to be classified on every clock cycle. The sum of tree scores, as required by Equation 4.16, is performed with a balanced adder tree. Features are distributed to the trees through a register tree, avoiding a significant fan out of signals which would degrade the maximum clock frequency. This comes at the expense of some extra latency.

From the FPGA implementation details, it is possible to construct a model for the resource and latency usage of a BDT dependent on the hyper-parameters. The parameters which impact the FPGA consumption are the number of trees,  $n$ , and the maximum depth  $d$ . The resource usage scales linearly with the number of trees - both for the decision tree inference and adder tree size - and exponentially (base 2) with the tree depth. A constant five clock cycles are used to fan out the features and execute all comparisons. The boolean logic to determine the active leaf is pipelined, scaling linearly with the tree depth. Finally the adder tree latency scales logarithmically with the number of trees. The model to predict resource usage,  $R$ , and latency,  $T$ , relative to a reference ensemble, with  $n_{\text{ref}} = 100$  and  $d_{\text{ref}} = 3$  (with 12 clock cycles latency), is given by Equations 4.17 and 4.18.

$$R = \frac{n}{n_{\text{ref}}} \times 2^{d-d_{\text{ref}}} \quad (4.17)$$

$$T = \frac{5}{12} + \frac{7}{24} \left( \log_2 \left( \frac{n}{n_{\text{ref}}} \right) + \frac{d}{d_{\text{ref}}} \right). \quad (4.18)$$

This model was used to explore the resource usage and latency of ensembles when scanning across a range of hyper-parameters  $n$  and  $d$ , as shown in Figure 4.23. The area under the ROC curve (AUC) is used as the metric of performance of the BDT. Some ensembles were synthesised to obtain true resource and latency figures, and

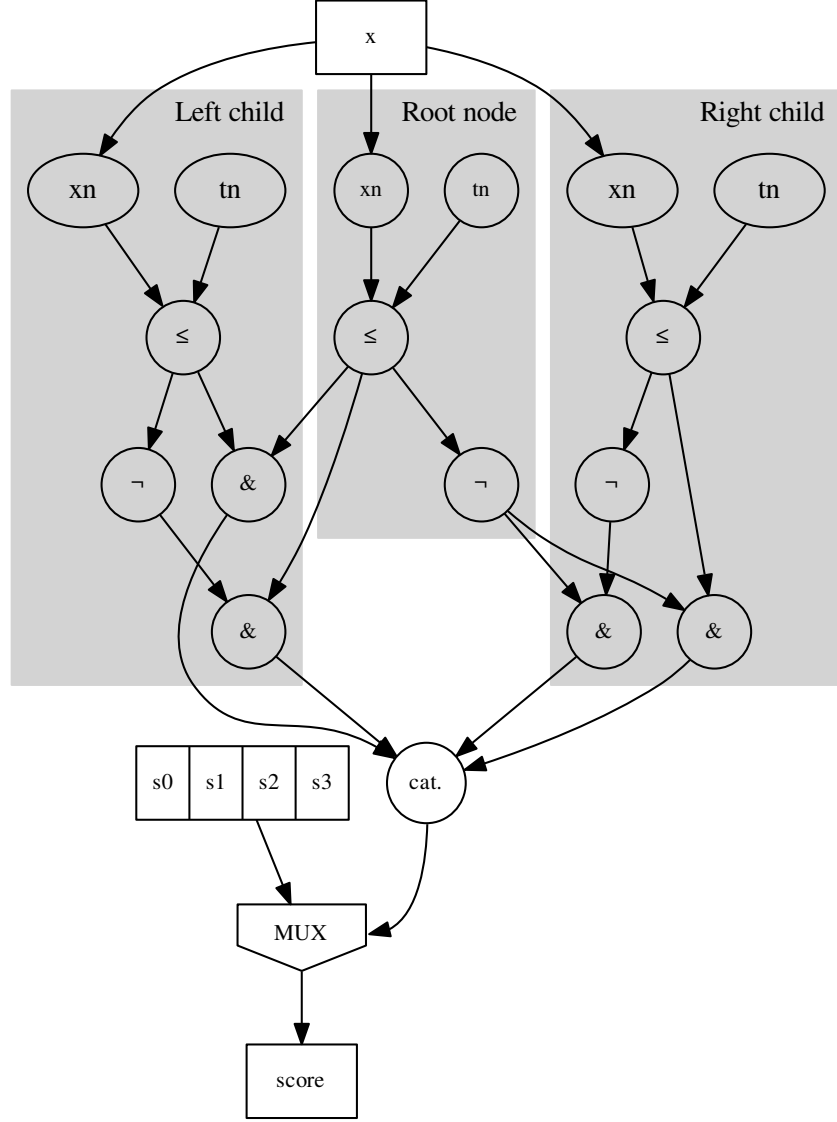


Figure 4.22: Schematic for the FPGA implementation of a decision tree with a depth of 3. A node stores its feature threshold  $t_n$  as a constant, and selects its required feature from the input vector  $x$ . A branch of the decision tree is activated depending on the comparison between the branch's node's feature and threshold, and the node's parents' comparisons. Each node in the FPGA implementation outputs two boolean signals corresponding to the branch's activation. At the leaf nodes, the leaf scores are presented as the input to a multiplexer (the node labelled 'MUX'). The branch bits are concatenated (the node labelled 'cat.') and used as the multiplexer address to select the decision tree score to output.

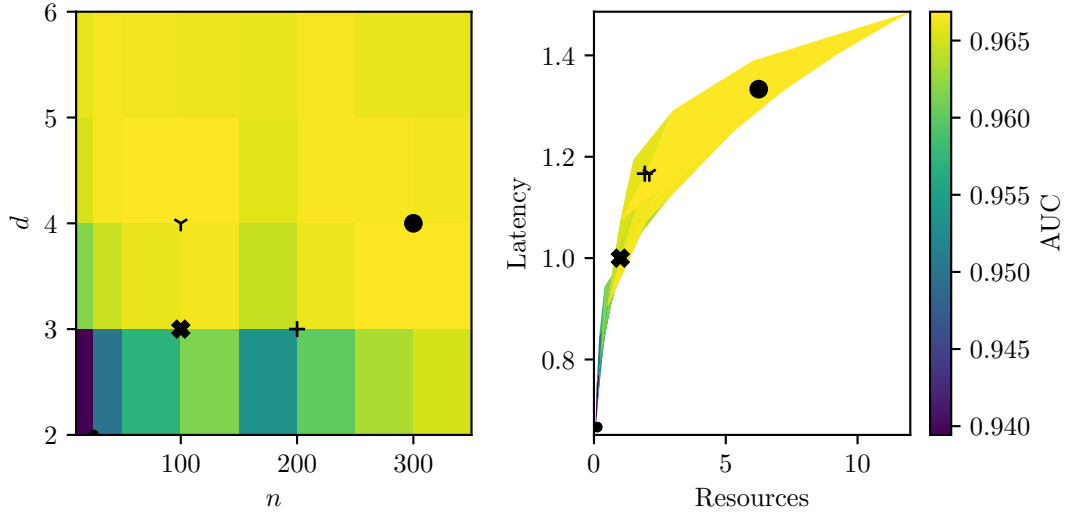


Figure 4.23: Area under the ROC curve (AUC), in colour, for a scan over BDT hyper-parameters  $n$  and  $d$ . The left plot shows how AUC varies with  $n$  and  $d$ , and the right plot uses the model of equations 4.17 and 4.18 to show how AUC varies with FPGA resources and latency. Resources and latency values are with respect to an ensemble with  $n = 100$  and  $d = 3$ . Points on the right plot are resource and latency measurements obtained after synthesising the ensemble.

Table 4.9: Post-synthesis resource usage of the trained BDT with 4 features, 100 trees with a maximum depth of 3, for 3 different FPGA parts: Virtex 7 690 (V7 690), Kintex Ultrascale 115 (KU 115), and Virtex Ultrascale 9+ (VU 9+).

Device	LUTs (%)	Registers (%)
V7 690	2.24	1.15
KU 115	1.46	0.75
VU 9+	0.82	0.42

are displayed in the Figure, and are well described by the model. It can be seen that the achievable AUC reaches a plateau for ensembles with  $n > 100$  and  $d > 3$ , which are correspondingly more expensive in resource consumption and latency. The BDT with  $n = 100$  and  $d = 3$  is therefore selected as the appropriate ensemble for the L1 trigger constraints.

The resource usage of the previously described BDT for several FPGA parts is shown in Table 4.9. The usage is around the 1% level, and approximately twice as many LUTs are used over registers, as a percentage of the total available.

The functional correctness of the FPGA implementation was tested against SCIKIT-LEARN running on a CPU. The FPGA used for this test was an Altera



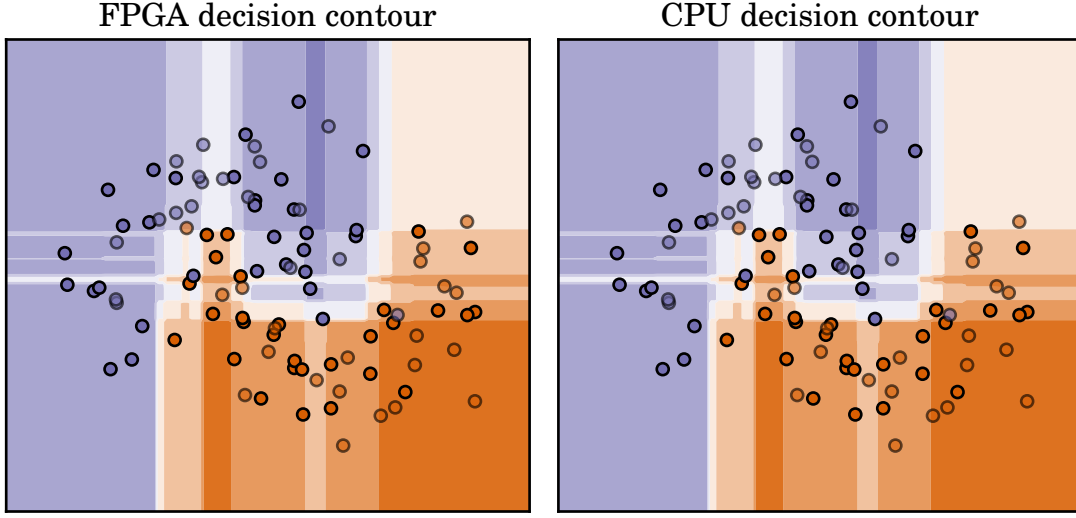


Figure 4.24: Decision contour for the FPGA and CPU implementations of a BDT. A BDT was trained on a randomly generated, separable 2D dataset with 2 classes (‘purple’ and ‘orange’). Points show the training data. Shading shows the prediction for a uniform sampling of the feature space.

Stratix V [92] on a Maxeler Maia DFE, in an MPC-X [93] machine housed at the DeLorean facility at the STFC Daresbury computing centre. The Maxeler ‘SLiC Interface’ and MAXELEROS software was used for CPU-FPGA communication, which is across an Infiniband network to a PCIe switch in the MPC-X. A BDT was trained to classify a randomly generated, separable, 2D dataset with 2 classes. A static FPGA implementation was then generated, as previously described. Next, 2D features were created, uniformly sampling the feature space of the generated dataset. These features were classified on the CPU and FPGA separately, and the output compared. The result is shown in Figure 4.24, where it can be seen that the architectures are in perfect agreement.

The DeLorean machine also facilitated a measurement of the speedup factor of the FPGA compared to a CPU, using the same trained BDT as used for Figure 4.24. While class prediction on the CPU may be relatively fast for a single datum, latencies of  $1.7\mu\text{s}$  were typical, the latency for classifying a large dataset may be long. The time to classify  $n$  data items,  $t_n$  is  $t_n = nt_1$ , for single item classification latency  $t_1$ , on a CPU. On the FPGA, on the other hand,  $t_n = t_{\text{setup}} + t_l + n/f$ , where  $t_{\text{setup}}$  is the FPGA setup time including communication latency,  $t_l$  is the algorithm latency, and  $f$  is the clock frequency of the algorithm. The setup time may be relatively long, typically around 1 s, but  $t_l$  is small for this implementation, and

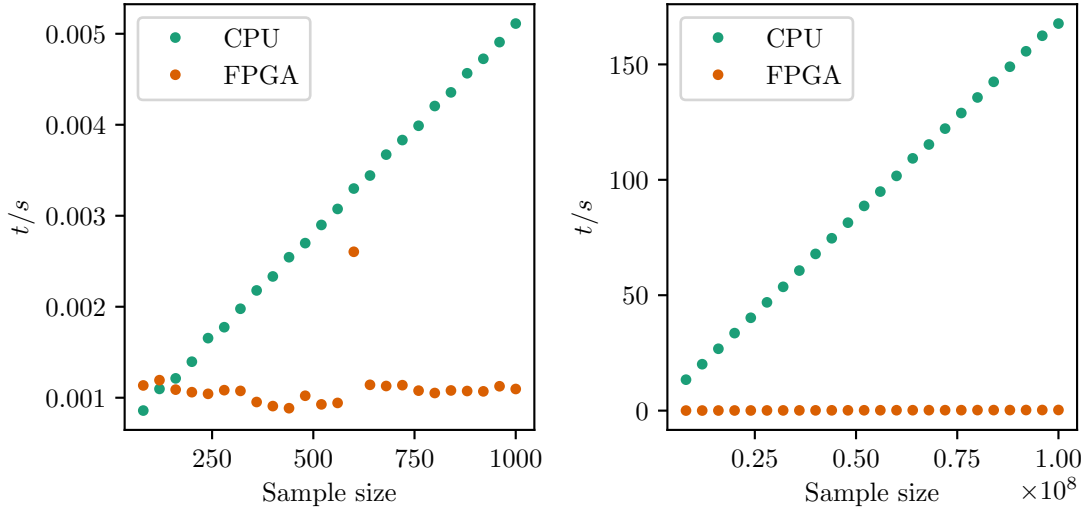


Figure 4.25: Number of classified data points,  $N$ , vs. execution time on the FPGA and CPU.

$f = 400$  MHz was achieved. With a large enough  $n$ , therefore, the FPGA should begin to classify faster than the CPU. As can be seen in Figure 4.25, this is indeed the case. With the BDT image preloaded onto the FPGA, and in CPU memory, the FPGA classifies a dataset fastest when there are more than 100 samples in the dataset. With 1000 samples the FPGA is five times faster. With a large number of samples, such that  $t \gg 1$  ms, the FPGA is up to 670 times faster.

### 4.3 Conclusions

The time multiplexed track trigger demonstrator shows the feasibility of performing track reconstruction on charged particles with  $p_T$  above 3 GeV in less than 4  $\mu$ s at the full 40 MHz LHC collision rate. A hardware system was constructed, using five MP7 boards carrying out track reconstruction for one eighth of the detector in azimuth, and one in thirty six LHC collisions. Since the processing for every detector octant and event is identical, the full system could be constructed by duplicating the demonstrator platform, and utilising an optical routing construct of the type used currently in the CMS calorimeter trigger.

Within the Track Finder Processor the reconstruction is first divided geometrically, to introduce additional parallelisation. Track candidates are found by thirty six instances of a Hough Transform algorithm, which groups stubs with a consistent

trajectory in the  $r$ - $\phi$  plane. These candidates are then cleaned, and a fit to their track parameters is obtained by a Kalman Filter. Finally, duplicate tracks are removed. A tracking efficiency of 95% is obtained for tracks from the primary vertex of  $t\bar{t}$  events with 200 PU, reproduced in both a software emulation and the FPGA platform. Track parameter resolutions adequate for use in the later L1 trigger stages are also achieved.

The system has been tested with events of  $t\bar{t}$  with 200 PU to find the limitations, and losses. Some 0.6% of tracks are lost due to the latency and throughput restrictions imposed. This occurs particularly at the very first processing step, where stubs are distributed geometrically. Collimated high- $p_T$  jets also create ‘busy’ regions in the processor which strain the readout of the HT.

The development towards a production trigger system for the HL-LHC would see the algorithm operating on an updated FPGA, with significantly increased resources and faster IO connections. This development also presents an opportunity to further improve the tracking algorithm. Regarding tracking performance, the rate of fake tracks can be reduced while maintaining the efficiency. For the benefit of the trigger project, reducing the FPGA resources can be pursued in order to save on system cost, and the latency can be pushed lower in order to add slack or allow other trigger systems more time.

One particular development for rejecting fake tracks uses a boosted decision tree. This machine learning technique achieves a significant improvement in fake rejection over a simple  $\chi^2$  cut, which was the method used for the demonstrator project. An FPGA implementation, developed using MaxCompiler, performs the class inference step for a trained BDT within tens of nanoseconds. Optimisation of the ensemble hyperparameters, with awareness of the impact on resources and latency, led to an ensemble which uses around 1% of the resources of an Ultrascale FPGA. The flexibility of the implementation is such that it could be adopted in other areas of the Level 1 Trigger.

## Chapter 5

# Hardware Acceleration of Track Reconstruction in the High Level Trigger

High Luminosity LHC (HL-LHC) conditions of 200 pileup (PU) collisions per event will increase the computation required to make a trigger decision at the High Level Trigger (HLT). In particular the latency of track reconstruction, which is computationally expensive, scales exponentially with pileup due to the ‘combinatorial explosion’ from the vastly increased number of hits in the tracker. As described in Section 2.10, the tracking executed at the HLT is modified from the offline tracking to improve the timing, and used sparingly, yet still contributes significantly to the processing time. Expected increases in computing power by 2025 are insufficient to prevent an increase in the time taken to reconstruct tracks at the HLT at the HL-LHC compared to the LHC. In this chapter, the use of FPGA hardware acceleration of tracking with a Maxeler DFE is investigated as a means of reducing the impact of the pileup increase.

### 5.1 Kalman Filter on a DFE

The Combinatorial Track Finder uses a Kalman Filter for both track building and fitting. A pseudo code representation of the track building procedure is shown in Figure 5.1. For every candidate compatible measurements in the next detector layers along the trajectory are found. Then for each found measurement, the candidate is updated by the Kalman Filter, branching each updated state to a new candidate.

```

1 while candidate pool is not empty do:
2   initialise empty temporary candidate pool;
3   for all candidates in candidate pool do:
4     find hits compatible with candidate;
5     for all compatible hits do:
6       update candidate state with hit;
7       if updated state is finished then:
8         add candidate to result tracks;
9       else:
10        add candidate to temporary candidate pool;
11    sort temporary candidate pool by quality;
12    discard all but best N candidates;
13    replace candidate pool with temporary candidate pool;
14 return result tracks;

```

Figure 5.1: Pseudo-code of the track building procedure of the Combinatorial Track Finder.

The finding of compatible measurements and state update contain all of the mathematical computations of the Kalman Filter. The project and update parts of the filter are split between the functions. In order to find compatible measurements, the algorithm first determines which detector layers are next intersected by the state trajectory, then propagates the state to the surfaces (there may only be one) in turn. Once the coordinates of the track intersection with a detector element are calculated, the hits lying in that detector element are found. Hits which are compatible (the  $\chi^2$  of the residual is below a threshold) are returned.

A Kalman Filter state update is performed for each of the found measurements. If no measurements were found, the state may be permitted to skip this iteration if it has not already skipped too many layers previously. Updated states which have reached a completion condition – the trajectory has been updated by a sufficient number of hits – are added to a container of results and no longer circulate the track builder loop. States requiring further iterations are stored in a separate container. After all states known at the start of the current iteration have been updated with measurements, the states are sorted by  $\chi^2$ , and only the best  $N$  are retained, with  $N$  depending on the iteration. Limiting the number of states kept at each iteration prevents an excessive number of combinations from being considered. At the end of the iteration the surviving, unfinished, states are recirculated.

Both finding compatible measurements and the state update perform matrix

operations, which can readily be parallelised and pipelined on a DFE, as already demonstrated in Chapter 4. The additional work performed to find measurements, by returning detector hits stored in memory, is less well suited to a parallel and pipelined implementation. Searching for compatible hits in memory requires significantly random access memory operations and highly data dependent processing, neither of which are desirable on the DFE. The Kalman Filter update, conversely, has no data dependent operation. Only the state update part of the track building was therefore ported to the DFE. A parallelised state update, with pipeline parallelism of the loop over measurements was developed.

### 5.1.1 MaxJ Implementation

The Kalman Filter state update is similar to that implemented in Section 4.1.4, using Equations 2.4 to 2.12. Since the HLT tracking fits 5 parameters, as introduced in Section 2.10, the state vector is 5 elements and matrices in the fit have dimensions  $5 \times 5$ ,  $5 \times 2$  or  $2 \times 2$ . All matrix multiplications are fully unrolled, as in Chapter 4. The data flow graph of this state update is shown in Figure 5.2. The extra parameter, giving wider matrices, results in more operations executing in parallel compared to Figure 4.9, for which the fit used only 4 parameters.

### 5.1.2 Data Types

Track reconstruction in CMSSW makes use of IEEE 754 standard [94] double precision floating point throughout. Floating point operations (whether double, single, or custom) always takes more time and uses more area than fixed point arithmetic tuned to the DSP port sizes (18 and 27 bits for an Altera Stratix V, as used here). Nonetheless, a floating point representation is the most appropriate choice where variables have a wide dynamic range, and need to be precisely represented across that range.

Table 5.1 shows the dynamic range of the state vector and matrices, as well as matrices calculated as part of the update of the state, obtained from numerical profiling. For each variable a histogram of the exponent (base 2) is filled during the execution of the unmodified Kalman Filter software, for every state update performed. This procedure informs the viability of using a fixed point representation when ported to the FPGA. In order to fully represent the range of a variable  $x$ , a fixed point format would require at least as many bits as the range  $\max(\log_2 |x|) -$

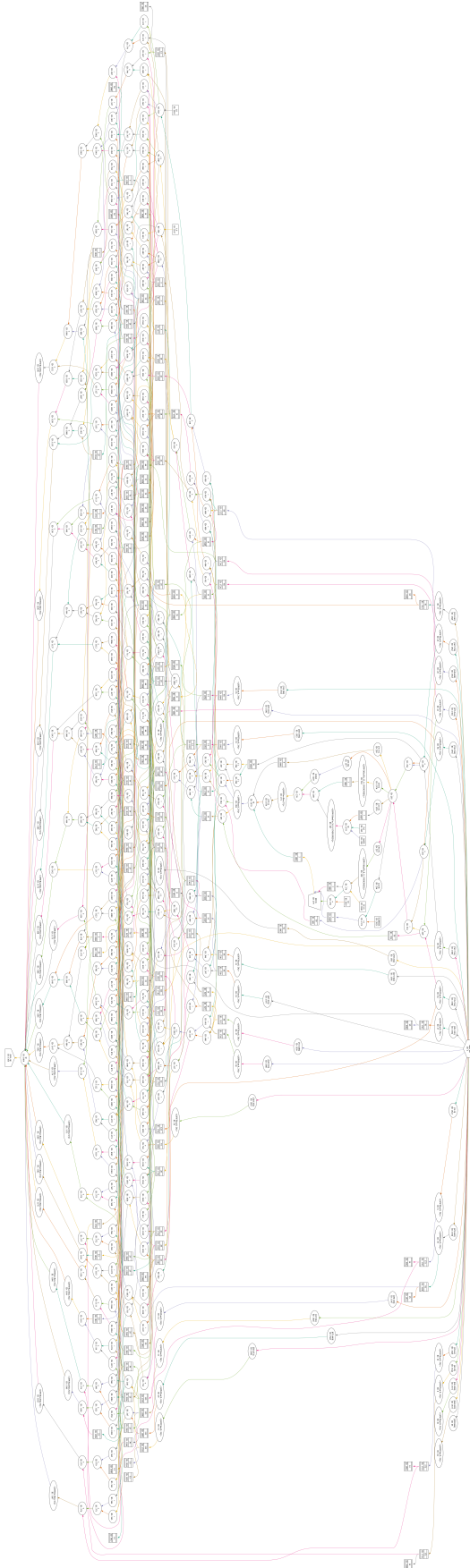


Figure 5.2: Kernel Graph representation of the HLT 5 parameter, floating point, Kalman Filter. The width of the graph illustrates the operation parallelism achieved for the state update.

Table 5.1: The range of the exponent of floating point variables in the Kalman Filter state update.

Matrix	column		1	2	3	4	5
	row						
<b>C</b>	1		18	27	34	31	31
	2		-	37	45	34	40
	3		-	-	33	44	29
	4		-	-	-	34	37
	5		-	-	-	-	26
<b>x<sup>T</sup></b>	-		19	25	23	21	20
<b>K<sup>T</sup></b>	1		23	20	29	18	50
	2		23	30	27	41	23

$\min(\log_2 |x|)$ . Extra bits would be needed to retain a reasonable precision on values close to  $\min(|x|)$ .

Only a very small number of variables have exponent distributions with a width of a few bits. The state vector elements, for example, cover a reasonably narrow range of around 20 bits each. Conversely the covariance matrix elements occupy a range much larger than the DSP port width, of up to 45 bits. Elements of the gain matrix **K** have an exponent range up to 50. To avoid the introduction of instability by saturating or overflowing fixed-point arithmetic, floating point representation was used throughout the DFE implementation. As a concession to the architecture, single precision was used rather than double as in the C++. No variables come near to the minimum or maximum values of single precision floating point of  $2^{-126} < x < 2^{+127}$ . For the matrix inversion the floating point configuration of the division algorithm presented in Appendix A was used.

### 5.1.3 Data Flow

Figure 5.3 shows schematically the flow of data between the CPU and DFE, and the execution of the main parts of the track building procedure. Measurements are paired with states on the CPU, and streamed to the DFE. The DFE updates the state and streams these back to the host.

On each platform, CPU and DFE, the time to process  $n$  state updates,  $T$ , is:

$$T_{\text{CPU}} = n \cdot t_{\text{CPU}}, \quad (5.1)$$

$$T_{\text{DFE}} = t_1 + n \cdot t_{\text{DFE}}, \quad (5.2)$$



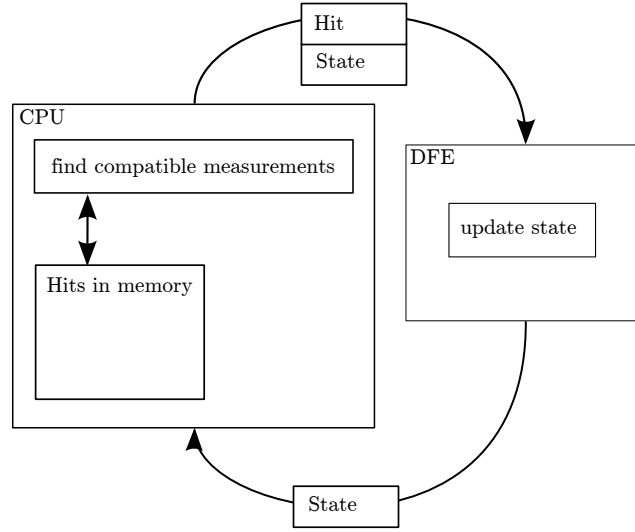


Figure 5.3: Diagram of execution placement of logical components of the track building, and the flow of data between the devices.

where  $t_{\text{CPU}}$  is the time for a single state update on the CPU;  $t_l$  is the latency incurred each time the DFE is called due to algorithm and data transfer latency, and initialisation; and  $t_{\text{DFE}}$  is the time between completion of each update. With a fully pipelined algorithm and data sent between devices for each iteration the time between updated states is bound by the slower of the algorithm clock frequency and the data transfer rate:  $t_{\text{DFE}} = \min(1/f_{\text{clk}}, 1/f_{\text{data}})$ .

In order to achieve speedup, that is  $T_{\text{DFE}} < T_{\text{CPU}}$ , it is necessary that  $t_{\text{DFE}} < t_{\text{CPU}}$ . Assuming that this is the case, the speedup factor  $T_{\text{DFE}}/T_{\text{CPU}}$  will be greater for larger values of  $n$ . Once the initial data transfer latency has been paid, and the pipeline is full, the additional latency for processing new state updates ought to be very small. From the algorithm presented in Figure 5.1, such as it is implemented in CMSSW, a restructuring was made in order to increase  $n$  by sending more states to the DFE in each transaction. This restructuring is shown in the pseudo-code listing of Figure 5.4. With this modification, the maximum possible number of states and measurements – before their updates are required for the next iteration – are sent to the DFE.

A histogram of the number of measurements that can be sent to the DFE in one transaction with the modified track building is shown in Figure 5.5. The restructured code was sampled when executing the HLT software on simulated events of  $t\bar{t}$  with 200 PU. Small numbers of updates occur much more frequently than large ones: 75% of iterations find fewer than 50 measurements across all states, and the most

```
1 while candidate pool is not empty do:
2   for all candidates in candidate pool do:
3     find hits compatible with candidate;
4     queue candidate and hits for update;
5   update all candidates with hits on DFE;
6   for all updated candidates do:
7     if updated state is finished then:
8       add candidate to result tracks;
9     else:
10      add candidate to pool of candidates;
11   sort candidate pool by quality;
12   discard all but best N candidates;
13 return result tracks;
```

Figure 5.4: Pseudo-code of the track building procedure modified to coalesce multiple states into a stream.

probable number of measurements is 1. The weighted histogram shows the raw count,  $n$ , weighted by the time taken to update  $n$  states on the CPU according to Equation 5.2. While occurring much less frequently, iterations with larger numbers of state updates contribute significantly to the execution time on the CPU. As discussed, the DFE speedup ought to become more significant with larger  $n$ .

On the DFE, a further optimisation was made for the benefit of speedup. The state is a data structure with 20 fields – 15 for the unique covariance matrix elements and 5 for the vector, while a hit is 5 fields – 3 unique covariance matrix elements and 2 coordinates. Each state may be updated with multiple hits, so rather than sending a stream of states with the same length as the stream of hits (the easier to implement variant), each state is sent only once and a separate stream specifying how many hits are to be updated with each state is used. Logic in the DFE reads a new state from the input after the current state has been updated with the appropriate number of hits. This greatly reduces the amount of data sent across the PCIe bus.

#### 5.1.4 Interface with CMSSW

Some steps were needed in order to utilise the DFE from the CPU application. The ‘Simple Live CPU Interface’ (SLiC) [95] is used to provide handles to the DFE application to embed in the software. A SLiC interface for the Kalman Filter update was defined, specifying the type and dimensions of the input and output streams. During the compile process a header file and shared object library are

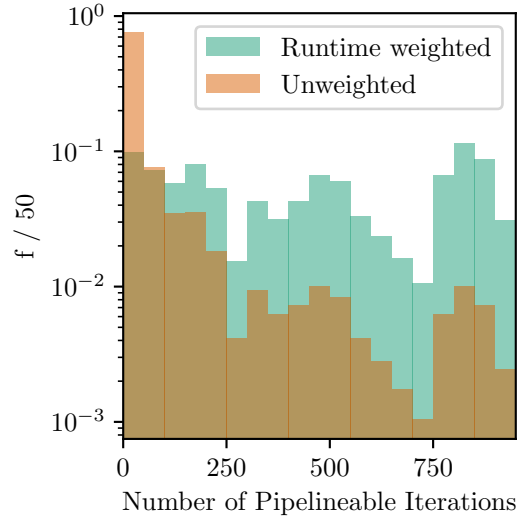


Figure 5.5: Histogram of the number of measurements found for all states, equivalent to the number of state updates that can be performed in a single DFE transaction. The unweighted histogram is the frequency of each bin, while the weighted histogram scales each entry by the contribution to the run time on a CPU.

generated containing functions for loading and executing the DFE from the CMSSW host code at a high level of abstraction from the underlying PCIe transfer and device configuration.

The main functions are displayed in the listing of Figure 5.6. A call to the functions on lines 1 and 2 load the Kalman Filter updater image onto a DFE available on the system (on a system with many DFEs this can be specified by the argument or left to be determined by the host). The `struct` on line 3 contains pointers to data corresponding to the data streams expected by the DFE: the hits, the states, the control stream specifying the number of hits per state, fields for the size of each stream, and a pointer to memory allocated for the returned updated states. The dataset to send to the DFE is termed an ‘action’. Finally the function on line 4 executes the action on the loaded DFE. A DFE can remain loaded and reserved by an application for its duration, or until explicitly unloaded.

Each of the application specific functions call further operations within the MAXELEROS software. The MAXELEROS package comprises software components which run on the host machine and corresponding hardware on the DFE which handle all low level communication between the devices. As well as providing functionality for inter-device communication and application control, the software includes a daemon process which monitors DFEs on the system.

```

1 max_file_t* dfeKFUpdater_init();
2 max_engine_t* max_load(max_file_t*, const char*);
3 struct dfeKFUpdater_actions_t;
4 void dfeKFUpdater_run(max_engine_t*, dfeKFUpdater_actions_t);

```

Figure 5.6: C++ functions and data structures for the operation of the Kalman Filter DFE from the host application.

Table 5.2: Resource usage of the Kalman Filter state updater and CPU IO on a Maia DFE. Resources are also expressed as a percentage of the Altera Stratix V 5SGSD8 device used.

Resource	Number	Percentage of available
ALMs	76054	28.98%
DSP Blocks	149	7.59%
Block Memory	1027	40.01%
Latency	181	-
$f_{\text{clk}}$	250 MHz	-

The CMSSW tracking code was modified according to the pseudo-code of Figure 5.4, with the functions of Figure 5.6. When testing with simulated events, a DFE was loaded at the beginning of the test and retained for the duration. The hits and states were repackaged into the appropriate `struct`, and the DFE was run with the required method. The CMSSW compile step, which uses the SCRAM build tool [96], was modified, including the relevant DFE application and Maxeler software header files, and linking to the libraries.

### 5.1.5 Performance

The performance of the DFE augmented CMSSW was measured on the STFC DeLorean MPC-X machine. An Intel Xeon Sandy Bridge E5-2650v2 processor (as also used for some HLT timing tests in [58]) running at 2.6 GHz executed the CMSSW application. The Kalman Filter state update was executed on a Maxeler Maia DFE, which hosts an Altera Stratix V 5SGSD8 FPGA and 48 GB of DRAM. The DFE card, housed in an MPC-X, interfaced to the CPU over a PCIe switch inside the MPC-X chassis, and an Infiniband network between the machines.

The resource usage, and other performance metrics, of the Kalman Filter state updater is listed in Table 5.2. Execution time of a single state update on the CPU was measured to be  $(255 \pm 34)$  ns, and the time for  $n$  state updates increased linearly

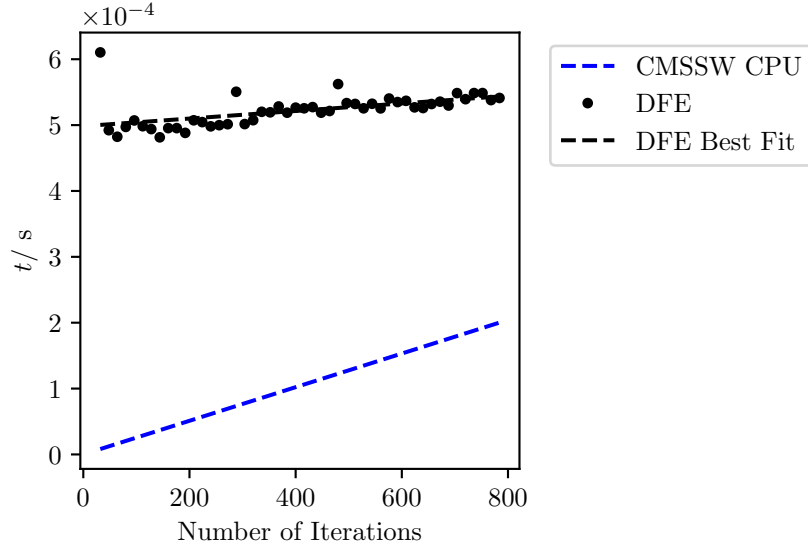


Figure 5.7: Measured time to perform a number of iterations of Kalman Filter state update on CPU and DFE.

as in Equation 5.2. The latency of the algorithm on the DFE is 181 clock cycles, which is 724 ns at the 250 MHz algorithm clock frequency.

Figure 5.7 shows the measured time of state update execution as a function of the number of updates, across a range compatible with the observed numbers of Figure 5.5. In all cases the CPU performs faster than the DFE. The gradient of the DFE execution time best fit is shallower than the CPU line, in other words  $t_{\text{DFE}} < t_{\text{CPU}}$ . A large initial latency,  $t_1$ , however, means that the CPU has completed execution before the DFE has begun for realistic numbers of updates. Here  $t_1$  is observed to be around 500  $\mu\text{s}$ , which is as long as 2000 state updates on the CPU. The DFE would eventually become faster than the CPU above 2519 updates.

The rate of iteration (from the gradient of the best fit) is  $17.4 \times 10^6 \text{ s}^{-1}$  on the DFE compared to  $3.92 \times 10^6 \text{ s}^{-1}$  on the CPU, a factor 4.4 faster on the DFE. This increased rate is the result of the pipelining of the states and hits into a continuous stream. The  $17.4 \times 10^6 \text{ s}^{-1}$  iteration rate achieved on the DFE is significantly below the 250 MHz clock frequency – the upper limit imposed by the algorithm. With one 80 B state produced for each iteration, a data rate of  $1.30 \text{ GB s}^{-1}$  is achieved, approaching the  $2 \text{ GB s}^{-1}$  limit of the card, suggesting the implementation is IO limited. Section 5.2 will discuss how this rate might be further improved, along with possible strategies to mitigate against the long initial latency of utilising the DFE.

## 5.2 Future Developments

While a speedup was not achieved from the development presented, the potential has been demonstrated. The DFE was able to perform state updates at a faster rate than the CPU, but with a large overhead, and at a rate limited by the link bandwidth. Several design and technology changes might help realise a definitive speedup for the tracking on a DFE. All of the improvements would be mutually beneficial.

### 5.2.1 Design Improvements

The bottleneck in the presented design is in the communication between the host and accelerator devices. No design improvement can increase the available bandwidth, but it may be possible to better utilise the available link. The ultimate performance would be realised by using the PCIe bus as little as possible.

#### Closing the Loop

The diagram of Figure 5.3 showed how state and hit data loops between the DFE and CPU. This is required, as part of the application – the finding of measurement compatible with a state – was not ported to the DFE. Executing this task on the DFE would remove the need for constant communication between the devices. Instead, all hits and seed states would be sent to the DFE once at the beginning of reconstruction, and the resulting states would be returned to the CPU at completion. Either the  $\mathcal{O}(10)$  MB of high throughput internal memory or GBs of DRAM could be used for this data. It was stated in Section 5.1 that the finding of measurements was less suited to the parallel, pipelined approach. Nonetheless, removing the communication overhead completely, even if one part of the application is slower on the DFE than the CPU, may enable the application to become faster as a whole. Given that the state update has been shown to offer a potentially significant speedup, some time penalty in finding measurements might be acceptable.

The state projection mathematics can be highly parallelised and pipelined as with the Kalman Filter. Mapping projections onto the detector geometry might also be parallelised by considering multiple detector elements simultaneously. Full or partial unrolling of any loop over detector layers will benefit from the predetermined detector layout. The final task, finding hits within compatible detectors, might be parallelised with a memory partitioning which makes use of the helical trajectories

of tracks. Such a scheme was used to parallelise the Hough Transform presented in Chapter 4, and also to speed up the hit searching on parallel processors in [97]. The internal FPGA memory can be partitioned, with each partition supporting independent read and write. A hit searching block could be developed to iterate over stored hits in a geometric partition and forward to them to a state updater.

### Link Usage

Even in a design for which the hit searching step remains on the CPU, a more efficient usage of the PCIe link could be made. All data pertaining to hits is known at the start of the event. The hits could be sent to the DFE at the start of processing, and retrieved from the on chip BRAM or on board DRAM. Instead of sending hits across the PCIe link, the host would send pointers to the hits in the DFE memory. Sending 32 bit pointers would address all of the on-chip memory and use  $\frac{1}{5}^{\text{th}}$  the bandwidth. The  $\mathcal{O}(10)$  MB of on chip memory would be sufficient to store  $\mathcal{O}(10^6)$  hits, although it cannot necessarily be used efficiently as one coalesced RAM, and some is used for the creation of the algorithm pipeline, as seen in Table 5.2.

The presented design returns the updated state from the DFE to the CPU, which is then propagated to a detector surface, followed by the search for hits. A state is represented by 20 fields of 32 bit data. Performing the state propagation on the DFE, and returning only enough information to identify the detector element and decide hit compatibility – the propagated coordinates and uncertainty should be sufficient – would then reduce the size of data sent between the devices.

These improvements would improve the rate of iteration on the DFE by reducing the volume of data transferred per iteration. The overhead paid each time data is transferred between devices would remain, however, and this can only be overcome with some changes to the accelerator setup.

### Data Types

While single precision floating point was used, due to the wide range of some variables, alternative data representations more suited to the FPGA architecture might be adopted. A floating point type with 17 mantissa bits and 7 exponent bits is suited to the DSP block on both Altera and Xilinx FPGAs. Adopting this representation would halve the DSP usage of multiplications, allowing more parallel instances of the state update. The smaller exponent would still be sufficient to represent the full range of the Kalman Filter variables. A 17 bit mantissa, however, would reduce the

precision by a factor  $2^7$  compared to single precision, which may lead to instability. This custom data type may also introduce complications for studying the performance of the HLT tracking, for which the ability to run on ‘standard’ processors is beneficial.

### 5.2.2 Technological Improvements

In addition to performance changes to the DFE algorithm, advances in technology will improve the performance and potential of a track reconstruction accelerator ahead of the HL-LHC upgrade.

#### FPGA

Next generation Xilinx FPGAs have been described in Section 4.1.10. An FPGA accelerator card based on a Xilinx FPGA, such as the Maxeler Max5, can leverage the increased resource availability to fit more parallel computation on a single device. For the Kalman Filter this could enable more instances of updater units to execute in parallel, as long as an improved IO bandwidth is also achieved.

Altera/Intel devices have improved significantly since the Stratix V generation used for the presented design and testing. The current generation, Arria and Stratix 10, feature hardened floating point units [98]. These move parts of the floating point operation that previously used general FPGA logic into the multiplier unit. The barrel shifting operation required to perform exponent normalisation and denormalisation is carried out inside the DSP. This change reduces the amount of logic resource used, and also allows floating point designs to achieve higher clock frequencies, since the barrel shift generally requires significant routing. The hardened floating point DSP blocks are limited to single precision.

In addition, the Intel synthesis tools support ‘fused’ floating point operations. Rather than treating sequential floating point operations as independent IEEE 754 operations, the synthesis tool analyses the datapath and allows the number of mantissa bits to grow in order to reduce the impact of successive normalisation/denormalisation steps that otherwise precede and follow each IEEE 754 operation. The wider mantissa also increases the accuracy compared to IEEE 754.

For the Kalman Filter state update, which uses a much higher percentage of the available logic than DSPs on the Stratix V (as seen in Table 5.2), these high floating point performance devices would increase capacity for more parallel updater instances, and have potential for higher clock frequencies.



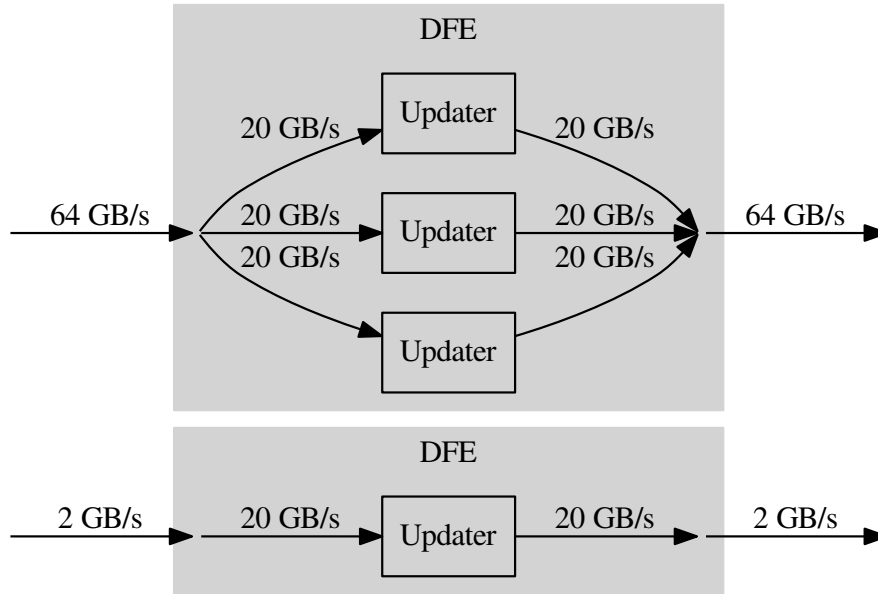


Figure 5.8: DFE configuration possibilities with different PCIe capabilities.

## PCIe

Figure 5.8 shows how a faster PCIe link speed might be utilised with an unchanged state updater. The Kalman Filter state updater kernel outputs one state per clock cycle. At 80 B per state, and a 250 MHz clock frequency, this is  $20 \text{ GB s}^{-1}$ . The PCIe communication rate accessible on a Maia card peaks at  $2 \text{ GB s}^{-1}$ , so the kernel must periodically stall to allow the communication to keep up. PCIe v.5, which is expected to be available around 2019, specifies  $64 \text{ GB s}^{-1}$ , which exceeds the rate at which a single updater can produce states. Given the FPGA resource utilisation of the single updater shown in Table 5.2, it is reasonable to expect to fit 3 parallel instances within a chip, especially with a next-generation device with higher resources. Such a configuration would then use almost the full available PCIe link rate (notwithstanding the better link utilisation described). This would yield a data rate improvement of around 30 times, alongside any aforementioned improvements to the algorithm data transfer.

## Alternative Accelerators

Integrated CPU-FPGA sockets, of the kind under development by Intel/Altera may provide the ultimate performance for a tightly coupled tracking accelerator. Placing the two processors on the same die will enable very low latency and high bandwidth

communications between the devices. While specifications for these future devices are not readily available, one would expect to be able to process data at the full rate that the algorithm can accommodate. Such a processor architecture would allow the FPGA to be used for only those tasks most suited to it, such as arithmetic on long data streams, while the CPU can be used for all control aspects such as branching and data dependent looping. This might allow a design similar to the one presented to achieve a speedup over a CPU only. CPU-FPGA sockets could likely fit into a heterogeneous HLT-like data-centre in a similar way to an architecture augmented only with FPGA accelerators.

A precursor to these tightly coupled devices has been used to demonstrate the acceleration of the particle identification algorithm used by LHCb [23]. The device is a dual socket server with a Xeon CPU and Stratix V connected over Intel's QPI, rather than a single die containing each device. This application is also limited by the connection between the host and accelerator, with an initial latency close to the 0.5 ms observed for the state update in this chapter. The connection bandwidth also limits the algorithm pipeline to be 50% full.

Graphics Processing Units (GPUs) are one of the most commonly used accelerators. The GPU architecture is very different to FPGAs, but they are also highly parallel processors. Where an FPGA is a 'hardware accelerator', a GPU is a 'software accelerator', that is, a GPU executes an instruction set on many cores in parallel.

GPUs have been applied to tracking problems, as in [97] for track building and fitting, and [99] for seeding. The authors of [97] note the difficulty of achieving an efficient use of the GPU when dealing with the branching that occurs from pursuing multiple combinations of tracks during track building, since the GPU threads within a warp must branch identically in order to execute in parallel. A modest speedup over a single threaded CPU is obtained, with the performance limited by data transfer to the GPU.

GPUs generally offer a lower entry point than FPGAs in terms of both cost and effort. FPGAs, however, do have significant benefits after these overheads. While the performance on any computation platform is highly algorithm dependent, FPGAs compare favourably to GPUs for many problems. In [100] the FPGA outperforms a GPU and CPU both in speed and operations per Watt for a recurrent neural network. Performance per Watt is an important consideration for the operational cost of data centres, such as the HLT, which will operate over a period of many years.

## 5.3 Conclusion

Track reconstruction is one of the most time consuming parts of the CMS experiment reconstruction. At the HLT currently, the timing of tracking is such that it must be used on only a subset of events, and within limited regions of phase-space. The track reconstruction time scales exponentially with pileup, which will pose problems at the HL-LHC where pileup of 200 is expected. FPGA accelerators may be able to speed-up the track reconstruction, by leveraging their significant operational and pipeline parallelism.

The Kalman Filter state update of the CMS HLT track reconstruction has been ported to a Maxeler DFE and integrated with the CMSSW tracking. The CMSSW application executes on a CPU host, and buffers hits and state combinations during the track building. These data are streamed through a DFE in an MPC-X, connected to the host over an Infiniband network. State updates are executed on the DFE and the data is streamed back to the host, where they are used to continue building tracks. Control flow for track branching – the exploration of the many possible combinations of hits which could belong to a track – is performed on the CPU.

Timing measurements of the execution of state updates on the DFE show that the initial latency of utilising the DFE is too slow to gain over using the host alone for realistic numbers of state updates encountered during track reconstruction in events of  $t\bar{t}$  with 200 PU. A state update rate enhancement of 4.4 times is obtained, limited by the bandwidth of the host-DFE connection, which shows the potential for a speedup from the DFE with further development.

Possible developments towards achieving a true speed enhancement were presented, including algorithm and technological developments anticipated before the HL-LHC. For the algorithm, the main improvement will be gained by ‘closing the loop’ on the DFE, only streaming data between host and DFE at the event start, and streaming tracks back after performing all of the track reconstruction on the DFE. This will necessitate the implementation of the more complicated control and branching, required to search for hits and combine multiple of them with tracks, on the DFE. Next generation FPGA devices, particularly those from Intel, may significantly benefit acceleration of the tracking. Dedicated floating point resources in the Intel Arria and Stratix 10 families will enable more parallel floating point computations, with higher clock frequencies than older Altera and current Xilinx devices. The prospect of Intel Xeon processors integrated with high capacity FPGA devices might also permit a use case like the one presented in this chapter, where both

the FPGA and host processor execute part of the application, with data streaming between the devices.

# Chapter 6

## Conclusion

The High Luminosity upgrade of the Large Hadron Collider will present a significant challenge for the trigger system. The CMS Level 1 Trigger will be upgraded, taking input from the calorimeters at higher granularity, receiving data from the tracker for the first time, and using particle flow reconstruction to achieve the best precision on particle momentum and identity, with mitigation for pileup. Latest generation, high performance FPGAs will carry out the processing for reconstruction and ultimately make the trigger decision. Implementing these algorithms on FPGAs, with microsecond latency, will be a demanding task. The latest design techniques, utilising high level languages, may enable more complicated algorithms to be realised.

The toolset of Maxeler Technologies, the language MaxJ and compiler MaxCompiler, was investigated for use in Level 1 Trigger applications. The algorithm for finding jets in the calorimeter, and the total event energy and transverse energy were reimplemented using MaxCompiler. The implementation currently deployed during Run II at CMS was developed with VHDL. The two designs produce bit-wise identical results, with a small discrepancy in jet  $\phi$  position. The MaxCompiler implementation used 7% more LUT resources than the VHDL, and matched the latency, while requiring only half as many lines of code to implement. The Maxeler tools were therefore demonstrated to be capable of producing high performance trigger applications with low latency.

In order to investigate the advantages of using the tool in the development of a sophisticated algorithm, MaxCompiler was then used in the development of a demonstrator system for the track reconstruction in the Level 1 Trigger at the HL-LHC. A Kalman Filter (KF) track fit was developed to fit and filter track candidates

found by a Hough Transform (HT). The algorithm is used in the offline track reconstruction at CMS, and can reject outlier stubs while simultaneously refining the calculated track parameters. The KF rejects 80% of the fake tracks produced by the HT in events of  $t\bar{t}$  with 200 PU, at the expense of 1 or 2 genuine particles per event. For muons without pileup the KF fits the  $p_T$  with 1% resolution in the barrel, up to 9% for high  $p_T$  muons at the most forward part of the detector, with a  $z_0$  resolution of 1 mm, up to 6 mm.

A boosted decision tree (BDT) classifier was developed to further reject fake tracks, using information from the track fit. The BDT is able to reject 75% of the remaining fakes at the cost of 1% more of genuine tracks. An FPGA implementation of BDT inference was developed, using MaxCompiler. The configuration with 100 trees with a depth of 3 uses around 1% of the resources of a KU115 or VU9P chip. For the same ensemble, an inference latency of 12 clock cycles was obtained, tested up to 400 MHz on a Stratix V, with full pipelining to enable a new classification on every clock cycle.

With the development of the Kalman Filter and BDT algorithms, both with low latency and reasonable resource usage, MaxCompiler is seen to be beneficial to the design of the kinds of sophisticated processing required for the CMS Phase II Upgrade. The automatic scheduling and pipelining, and the extensive support for custom fixed point arithmetic make the tool advantageous for the design of high throughput, mathematically intensive algorithms.

Finally, Maxeler Dataflow Engine (DFE) devices were targeted to investigate acceleration of the track reconstruction of the High Level Trigger. The Kalman Filter state update part of the tracking procedure was ported to the DFE. This implementation fits the five track parameters, and uses floating point arithmetic compared to the four parameter, fixed point fit developed for the Level 1 tracking. The connection between DFE and host proved to be the bottleneck: both the bandwidth and the latency. A 500  $\mu$ s overhead is spent each time the DFE is used, in which time around 2000 state updates can be executed on the CPU. The potential for speedup is seen by the factor 4.4 faster rate of iteration on the DFE compared to the CPU. Developments to overcome this limitation were presented. These include moving the entire computation to the DFE, and reducing the data size transferred between the devices, making a more efficient use of the link. Technological advances ahead of the HL-LHC will also benefit FPGA accelerated computation, with higher bandwidth host-FPGA connections, and tightly coupled CPU-FPGA processors.

# Bibliography

- [1] S. Summers, A. Rose, and P. Sanders, “Using maxcompiler for the high level synthesis of trigger algorithms,” *Journal of Instrumentation*, vol. 12, no. 02, p. C02015, 2017.
- [2] R. Aggleton *et al.*, “An FPGA based track finder for the L1 trigger of the CMS experiment at the High Luminosity LHC,” *Journal of Instrumentation*, vol. 12, no. 12, P12019, 2017.
- [3] S. L. Glashow, “Partial-symmetries of weak interactions,” *Nuclear Physics*, vol. 22, no. 4, pp. 579–588, 1961. DOI: 10.1016/0029-5582(61)90469-2.
- [4] S. Weinberg, “A Model of Leptons,” *Phys. Rev. Lett.*, vol. 19, pp. 1264–1266, 21 Nov. 1967. DOI: 10.1103/PhysRevLett.19.1264.
- [5] A. Salam and J. Ward, “Electromagnetic and weak interactions,” *Physics Letters*, vol. 13, no. 2, pp. 168–171, 1964. DOI: 10.1016/0031-9163(64)90711-5.
- [6] F. Englert and R. Brout, “Broken Symmetry and the Mass of Gauge Vector Mesons,” *Phys. Rev. Lett.*, vol. 13, pp. 321–323, 1964. DOI: 10.1103/PhysRevLett.13.321.
- [7] P. W. Higgs, “Broken symmetries, massless particles and gauge fields,” *Phys. Lett.*, vol. 12, pp. 132–133, 1964. DOI: 10.1016/0031-9163(64)91136-9.
- [8] P. W. Higgs, “Spontaneous Symmetry Breakdown without Massless Bosons,” *Phys. Rev.*, vol. 145, pp. 1156–1163, 1966. DOI: 10.1103/PhysRev.145.1156.
- [9] T. W. B. Kibble, “Symmetry breaking in Non-Abelian gauge theories,” *Phys. Rev.*, vol. 155, pp. 1554–1561, 1967. DOI: 10.1103/PhysRev.155.1554.
- [10] The CMS Collaboration, “Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC,” *Physics Letters B*, vol. 716, no. 1, pp. 30–61, 2012. DOI: 10.1016/j.physletb.2012.08.021.

- [11] ATLAS Collaboration, “Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC,” *Physics Letters B*, vol. 716, no. 1, pp. 1–29, 2012. DOI: 10.1016/j.physletb.2012.08.020.
- [12] D. Contardo *et al.*, “Technical Proposal for the Phase-II Upgrade of the CMS Detector,” Geneva, Tech. Rep. CERN-LHCC-2015-010. LHCC-P-008. CMS-TDR-15-02, Jun. 2015.
- [13] The CMS Collaboration, “Vector Boson Scattering and Quartic Gauge Coupling Studies in WZ Production at 14 TeV,” CERN, Geneva, Tech. Rep. CMS-PAS-FTR-13-006, 2013.
- [14] The CMS Collaboration, “Particle-Flow Event Reconstruction in CMS and Performance for Jets, Taus, and MET,” CERN, Geneva, Tech. Rep. CMS-PAS-PFT-09-001, Apr. 2009.
- [15] W. J. Stirling, *Private communication*.
- [16] P. M. S. Blackett, “On the Technique of the Counter Controlled Cloud Chamber,” *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 146, no. 857, pp. 281–299, 1934. DOI: 10.1098/rspa.1934.0154.
- [17] G. L. Bayatyan *et al.*, *CMS computing: Technical Design Report*, ser. Technical Design Report CMS. Geneva: CERN, 2005, Submitted on 31 May 2005.
- [18] G. Aad *et al.*, “Performance of the ATLAS Trigger System in 2010,” *Eur. Phys. J.*, vol. C72, p. 1849, 2012. DOI: 10.1140/epjc/s10052-011-1849-1.
- [19] R. Aaij *et al.*, “The LHCb Trigger and its Performance in 2011,” *JINST*, vol. 8, P04022, 2013. DOI: 10.1088/1748-0221/8/04/P04022.
- [20] E. Michielin, “The LHCb trigger in Run II,” *PoS*, vol. ICHEP2016, 996. 4 p, 2016.
- [21] LHCb Collaboration, “LHCb Trigger and Online Upgrade Technical Design Report,” Tech. Rep. CERN-LHCC-2014-016. LHCb-TDR-016, May 2014.
- [22] A. Abba *et al.*, “A specialized track processor for the LHCb upgrade,” CERN, Geneva, Tech. Rep. LHCb-PUB-2014-026. CERN-LHCb-PUB-2014-026, Mar. 2014.



- [23] C. Färber *et al.*, “Particle identification on an FPGA accelerated compute platform for the LHCb upgrade,” in *2016 IEEE-NPSS Real Time Conference (RT)*, Jun. 2016, pp. 1–2.
- [24] Xilinx Inc., *7 Series FPGAs Configurable Logic Block*, vol. UG474 (v1.8).
- [25] Xilinx Inc., *7 Series DSP48E1 Slice*, vol. UG479 (v1.9).
- [26] Xilinx Inc., *7 Series FPGAs Memory Resources*, vol. UG473 (v1.12).
- [27] M. Cacciari, G. P. Salam, and G. Soyez, “The anti-kt jet clustering algorithm,” *Journal of High Energy Physics*, vol. 2008, no. 04, p. 063, 2008.
- [28] M. Benedikt *et al.*, *LHC Design Report Volume III*, ser. CERN Yellow Reports: Monographs. Geneva: CERN, 2004.
- [29] O. S. Brüning *et al.*, *LHC Design Report Volume I*, ser. CERN Yellow Reports: Monographs. Geneva: CERN, 2004.
- [30] G. Apollinari *et al.*, *High-Luminosity Large Hadron Collider (HL-LHC): Preliminary Design Report*, ser. CERN Yellow Reports: Monographs. Geneva: CERN, 2015.
- [31] T. Sakuma and T. McCauley, “Detector and Event Visualization with SketchUp at the CMS Experiment,” *Journal of Physics: Conference Series*, vol. 513, no. 2, p. 022 032, 2014.
- [32] D. Bertolini *et al.*, “Pileup per particle identification,” *Journal of High Energy Physics*, vol. 2014, no. 10, p. 59, Oct. 2014. DOI: 10.1007/JHEP10(2014)059.
- [33] V. Karimäki *et al.*, *The CMS tracker system project: Technical Design Report*, ser. Technical Design Report CMS. Geneva: CERN, 1997.
- [34] Brondolin, Erica, “Expected Performance of Tracking in CMS at the HL-LHC,” *EPJ Web Conf.*, vol. 150, 2017. DOI: 10.1051/epjconf/201715000001.
- [35] J. Chistiansen and M. Garcia-Sciveres, “RD Collaboration Proposal: Development of pixel readout integrated circuits for extreme rate and radiation,” CERN, Geneva, Tech. Rep. CERN-LHCC-2013-008. LHCC-P-006, Jun. 2013.
- [36] J. Jones *et al.*, “A Pixel Detector for Level-1 Triggering at SLHC,” Oct. 2005.
- [37] M. Pesaresi, “Development of a new Silicon Tracker for CMS at Super-LHC,” CERN-THESIS-2010-083, PhD thesis, Imperial College, London, 2010.

- [38] G. Hall, M. Raymond, and A. Rose, “2-D PT module concept for the SLHC CMS tracker,” *Journal of Instrumentation*, vol. 5, p. C07012, 2010. DOI: 10.1088/1748-0221/5/07/C07012.
- [39] M. Pesaresi and G. Hall, “Simulating the performance of a pT tracking trigger for CMS,” *Journal of Instrumentation*, vol. 5, p. C08003, 2010. DOI: 10.1088/1748-0221/5/08/C08003.
- [40] K. Klein, “The Phase-2 Upgrade of the CMS Tracker,” CERN, Geneva, Tech. Rep. CERN-LHCC-2017-009. CMS-TDR-014, Jun. 2017.
- [41] The CMS Collaboration, *The CMS electromagnetic calorimeter project: Technical Design Report*, ser. Technical Design Report CMS. Geneva: CERN, 1997.
- [42] The CMS Collaboration, “The CMS Experiment at the CERN LHC,” *Journal of Instrumentation*, vol. 3, no. 08, S08004, 2008.
- [43] P. Adzic, “Energy resolution of the barrel of the CMS Electromagnetic Calorimeter,” *Journal of Instrumentation*, vol. 2, no. 04, 2007.
- [44] J. Mans *et al.*, “CMS Technical Design Report for the Phase 1 Upgrade of the Hadron Calorimeter,” Tech. Rep. CERN-LHCC-2012-015. CMS-TDR-10, Sep. 2012.
- [45] S. Abdullin *et al.*, “The CMS barrel calorimeter response to particle beams from 2 to 350 GeV/c,” *The European Physical Journal C*, vol. 60, no. 3, pp. 359–373, Apr. 2009. DOI: 10.1140/epjc/s10052-009-0959-5.
- [46] The CMS Collaboration, *The CMS muon project: Technical Design Report*, ser. Technical Design Report CMS. Geneva: CERN, 1997.
- [47] S. Chatrchyan *et al.*, “The performance of the CMS muon detector in proton-proton collisions at  $\sqrt{s} = 7$  TeV at the LHC,” *JINST*, vol. 8, P11002, 2013. DOI: 10.1088/1748-0221/8/11/P11002.
- [48] S. Chatrchyan *et al.*, “Performance of CMS muon reconstruction in  $pp$  collision events at  $\sqrt{s} = 7$  TeV,” *JINST*, vol. 7, P10002, 2012. DOI: 10.1088/1748-0221/7/10/P10002.
- [49] A. Tapper and D. Acosta, “CMS Technical Design Report for the Level-1 Trigger Upgrade,” CERN-LHCC-2013-011. CMS-TDR-12, Jun. 2013.
- [50] A. Zabi *et al.*, “The CMS Level-1 Calorimeter Trigger for the LHC Run II,” *Journal of Instrumentation*, vol. 12, no. 01, p. C01065, 2017.

- [51] The CMS Collaboration, “The Phase-2 Upgrade of the CMS L1 Trigger Interim Technical Design Report,” CERN, Geneva, Tech. Rep. CERN-LHCC-2017-013. CMS-TDR-017, Sep. 2017.
- [52] A.-M. Magnan, “HGCAL: a High-Granularity Calorimeter for the endcaps of CMS at HL-LHC,” *Journal of Instrumentation*, vol. 12, no. 01, p. C01042, 2017.
- [53] A. Svetek, “Construction, Testing, Installation, Commissioning and Operation of the CMS Calorimeter Trigger Layer-1 CTP7 Cards,” Poster at TWEPP 2015.
- [54] K. Compton *et al.*, “The MP7 and CTP-6: multi-hundred Gbps processing boards for calorimeter trigger upgrades at CMS,” *Journal of Instrumentation*, vol. 7, no. 12, p. C12024, 2012.
- [55] Imperial College London, “MP7 website <http://www.hep.ph.ic.ac.uk/mp7>.”
- [56] A. Zabi *et al.*, “Triggering on electrons, jets and tau leptons with the CMS upgraded calorimeter trigger for the LHC Run II,” *Journal of Instrumentation*, vol. 11, no. 02, 2016.
- [57] D. J. Lange and the CMS Collaboration, “The CMS Reconstruction Software,” *Journal of Physics: Conference Series*, vol. 331, no. 3, p. 032 020, 2011.
- [58] C. Richardson, “CMS High Level Trigger Timing Measurements,” *Journal of Physics: Conference Series*, vol. 664, no. 8, p. 082 045, 2015.
- [59] T. Hauth, V. Innocente, and D. Piparo, “Development and Evaluation of Vectorised and Multi-Core Event Reconstruction Algorithms within the CMS Software Framework,” *Journal of Physics: Conference Series*, vol. 396, no. 5, p. 052 065, 2012.
- [60] J.-M. Andre *et al.*, “The CMS Data Acquisition - Architectures for the Phase-2 Upgrade,” *Journal of Physics: Conference Series*, vol. 898, no. 3, p. 032 019, 2017.
- [61] The CMS Collaboration, “Description and performance of track and primary-vertex reconstruction with the CMS tracker,” *Journal of Instrumentation*, vol. 9, no. 10, P10009, 2014.

- [62] M. Tosi, “Tracking at High Level Trigger in CMS,” *Nuclear and Particle Physics Proceedings*, vol. 273-275, pp. 2494–2496, 2016, 37th International Conference on High Energy Physics (ICHEP). DOI: 10.1016/j.nuclphysbps.2015.09.436.
- [63] M. Rovere, “CMS reconstruction improvements for the tracking in large pileup events,” *J. Phys.: Conf. Ser.*, vol. 664, no. 7, 072040. 8 p, 2015.
- [64] R. E. Kálmán, “A New Approach to Linear Filtering and Prediction Problems,” *Transactions of the ASME—Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
- [65] R. Fruhwirth, “Application of Kalman filtering to track and vertex fitting,” *Nucl. Instrum. Meth.*, vol. A262, pp. 444–450, 1987. DOI: 10.1016/0168-9002(87)90887-4.
- [66] A. Strandlie and W. Wittek, “Propagation of covariance matrices of track parameters in homogeneous magnetic fields in CMS,” 2006.
- [67] Xilinx Inc., *High-Level Synthesis*, vol. UG902 (v2014.1).
- [68] N. Ghanathe *et al.*, “Software and firmware co-development using high-level synthesis,” *Journal of Instrumentation*, vol. 12, no. 01, p. C01083, 2017.
- [69] M. Husejko, J. Evans, and J. C. R. da Silva, “Investigation of High-Level Synthesis tools’ applicability to data acquisition systems design based on the CMS ECAL Data Concentrator Card example,” *Journal of Physics: Conference Series*, vol. 664, no. 8, p. 082019, 2015.
- [70] V. M. Ghete and CMS Collaboration, “The CMS L1 Trigger emulation software,” *Journal of Physics: Conference Series*, vol. 219, no. 3, p. 032009, 2010.
- [71] J. Chaves, “Implementation of FPGA-based level-1 tracking at CMS for the HL-LHC,” *Journal of Instrumentation*, vol. 9, no. 10, p. C10038, 2014.
- [72] M. Jeitler *et al.*, “The level-1 global trigger for the CMS experiment at LHC,” *Journal of Instrumentation*, vol. 2, no. 01, 2007.
- [73] T. Matsushita and CMS Collaboration, “Flexible trigger menu implementation on the Global Trigger for the CMS Level-1 trigger upgrade,” *Journal of Physics: Conference Series*, vol. 898, no. 3, p. 032033, 2017.
- [74] Maxeler Technologies, “MaxCompiler (White Paper),”

- 
- [75] P. V. C. Hough, *Method and means for recognizing complex patterns*, US Patent 3,069,654, Dec. 1962.
- [76] P. V. C. Hough, “Machine Analysis of Bubble Chamber Pictures,” *Conf. Proc.*, vol. C590914, pp. 554–558, 1959.
- [77] R. O. Duda and P. E. Hart, “Use of the Hough Transformation to Detect Lines and Curves in Pictures,” *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972. DOI: 10.1145/361237.361242.
- [78] D. Ballard, “Generalizing the Hough transform to detect arbitrary shapes,” *Pattern Recognition*, vol. 13, no. 2, pp. 111–122, 1981. DOI: 10.1016/0031-3203(81)90009-1.
- [79] Siklér, Ferenc, “Combination of various data analysis techniques for efficient track reconstruction in very high multiplicity events,” *EPJ Web Conf.*, vol. 150, p. 00 011, 2017. DOI: 10.1051/epjconf/201715000011.
- [80] Xilinx Inc., *UltraScale Architecture DSP Slice*, vol. UG579 (v1.4).
- [81] Xilinx Inc., *Accelerating Design Productivity with 7 Series FPGAs and DSP Platforms*. Feb. 2013, vol. WP406 (v1.1).
- [82] E. Hazen *et al.*, “The AMC13XG: a new generation clock/timing/DAQ module for CMS MicroTCA,” *Journal of Instrumentation*, vol. 8, no. 12, p. C12036, 2013.
- [83] C. G. Larrea *et al.*, “IPbus: a flexible Ethernet-based control system for xTCA hardware,” *Journal of Instrumentation*, vol. 10, no. 02, p. C02019, 2015.
- [84] H. Mohr *et al.*, “Evaluation of GPUs as a level-1 track trigger for the High-Luminosity LHC,” *Journal of Instrumentation*, vol. 12, no. 04, p. C04019, 2017.
- [85] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [86] M. Barbareschi *et al.*, “Decision Tree-Based Multiple Classifier Systems: An FPGA Perspective,” in *Multiple Classifier Systems*, F. Schwenker, F. Roli, and J. Kittler, Eds., Springer International Publishing, 2015, pp. 194–205.
- [87] M. Owaida *et al.*, “Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms,” in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–8. DOI: 10.23919/FPL.2017.8056784.

- [88] B. V. Essen *et al.*, “Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA?” In *IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, Apr. 2012, pp. 232–239. DOI: 10.1109/FCCM.2012.47.
- [89] R. Kulaga and M. Gorgon, “FPGA Implementation of Decision Trees and Tree Ensembles for Character Recognition in Vivado HLS,” *Image Processing & Communications*, vol. 19, Jan. 2014.
- [90] J. Oberg *et al.*, “Random decision tree body part recognition using FPGAs,” in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012, pp. 330–337. DOI: 10.1109/FPL.2012.6339226.
- [91] D. E. Acosta *et al.*, “Boosted Decision Trees in the Level-1 Muon Endcap Trigger at CMS,” CERN, Geneva, Tech. Rep. CMS-CR-2017-357, Oct. 2017.
- [92] Altera, *Stratix V Device Overview*, SV51001, Oct. 2015.
- [93] Maxeler Technologies, *New Maxeler MPC-X series: Maximum Performance Computing for Big Data Applications*, Mar. 2012.
- [94] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*, 2008.
- [95] Maxeler Technologies, *Multiscale Dataflow Programming*. 2014.
- [96] J. Wellisch, C. Williams, and S. Ashby, “SCRAM: Software configuration and management for the LHC Computing Grid project,” Jun. 2003.
- [97] G. Cerati *et al.*, “Parallelized Kalman-Filter-Based Reconstruction of Particle Tracks on Many-Core Processors and GPUs,” *EPJ Web Conf.*, vol. 150, p. 00 006, 2017. DOI: 10.1051/epjconf/201715000006.
- [98] Intel Corporation, “Enabling Impactful DSP Designs on FPGAs with Hardened Floating-Point Implementation (White Paper),”
- [99] F. Pantaleo *et al.*, “New Track Seeding Techniques for the CMS Experiment,” 2017.
- [100] E. Nurvitadhi *et al.*, “Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC,” in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2016, pp. 1–4. DOI: 10.1109/FPL.2016.7577314.

# Appendices

# Appendix A

## Integer Division

Consider the inversion of the  $2 \times 2$ , diagonal matrix  $\mathbf{X}$ . This matrix is simple enough to invert using the analytic solution.

$$X^{-1} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}^{-1} = \frac{1}{ab} \begin{bmatrix} b & 0 \\ 0 & a \end{bmatrix} = \begin{bmatrix} 1/a & 0 \\ 0 & 1/b \end{bmatrix} \quad (\text{A.1})$$

The final expression requires fewer processing steps than the intermediate solution, and allows for finer control over the precision of the two non-zero elements. An implementation of  $1/x$  is therefore required, which is usually an expensive operation in the FPGA. The algorithm must also be fast, in order to meet the latency requirement. A lookup would be the fastest possible algorithm, but since the divisor is a 25 bit quantity, the cost in BRAMs would be too great. An algorithm using a single BRAM for a lookup was developed.

The fixed-point divisor  $x$  can be expressed as the sum of individual powers of two as:  $x = \sum_n x_n 2^n$  where  $x_n$  can be 0 or 1. This sum can in turn be expressed as the sum of two smaller sums:

$$x = \sum_{n=m}^{\infty} x_n 2^n + \sum_{n=0}^{m-1} x_n 2^n = x_H + x_L \quad (\text{A.2})$$

where  $m$  bits are used to encode  $x_L$ . Then:

$$\frac{1}{x} = \frac{1}{x_H + x_L} = \frac{1}{x_H \left(1 + \frac{x_L}{x_H}\right)} = \frac{1}{x_H} \left(1 + \frac{x_L}{x_H}\right)^{-1} \approx \frac{x_H - x_L}{x_H^2} \quad (\text{A.3})$$

where a binomial series, truncated after the second term was used for the last step.



The value of  $m$ , that is the number of bits used for  $x_L$  is chosen such that  $x_H$  uses 11 bits, and therefore one BRAM is used to lookup  $1/x_H^2$  which are stored as 35 bit quantities. In the implementation a shift is performed such that the most significant bit of  $x$  has value 1. Doing so gives the best precision of  $x_H$ , and reduces the number of bits needed to address the lookup by 1 as an msb value of 1 can be assumed. A corresponding restoring shift is performed on the result. The  $x_H - x_L$  calculation is performed in LUTs, and  $(1/x_H^2) \times (x_H - x_L)$  uses DSPs. The algorithm steps are:

- Take the absolute value of  $x$
- Identify position of leading 1 in  $|x|$
- Shift  $|x|$  to the left such that the msb has value 1
- Slice  $|x|$  to obtain  $x_H$  and  $x_L$ , then in parallel:
  - Compute  $x_H - x_L$
  - Lookup  $1/x_H^2$
- Multiply  $1/x_H^2$  by  $x_H - x_L$
- Shift the result to the right by the same amount as the first shift
- Restore the sign of  $1/x$

The data type required for the output is defined by  $\max(1/x) = 1/\min(x)$ . Since the greatest precision in  $1/x$  is desired, in the case that  $\min(x) > \text{lsb}(x)$  some extra precision can be gained using ‘ignore bits’. The number of ignore bits should be defined as  $n_I = \log_2 \text{lsb}(x) - \text{ceil} \log_2 \min(x)$ , in which case the  $n_I$  lsbs cannot contribute to the initial shift, and the  $\max(1/x)$  value can be set to  $1/\min(x)$  rather than  $1/\text{lsb}(x)$ .

The configuration used for the Level 1 track finding, with  $\min(x) = 2^6 \times \text{lsb}(x)$  the division algorithm has a worst case accuracy, defined in equation A.4, of 16 bits. The worst case accuracy is observed for the largest divisors, and with small divisors the accuracy is as good as 29 bits (not counting instances of power of two divisors, where the result is exact).

$$\text{accuracy} = -\log_2 \left( 1 - x \frac{1}{x} \right) \quad (\text{A.4})$$

### A.0.1 Floating Point

The algorithm can also be modified for floating point quantities. A floating point quantity  $x$  is represented as  $x = m \cdot 2^e$  for a mantissa  $m$  such that  $1 \leq m < 2$ , and exponent  $e$ . The inverse of  $x$  is therefore:

$$\frac{1}{x} = \frac{1}{m} \cdot 2^{-e}. \quad (\text{A.5})$$

Using IEEE 754 standard floating point formats, mantissas are normalised such that a leading value of ‘1’ can be assumed. Denormalized values are not supported by this implementation. The above algorithm is therefore modified to skip the bit shifting steps, since the mantissa is already appropriately aligned. The new exponent must be calculated separately, and is simply the negative of the original exponent. Unusually, this algorithm is faster for floating point variables than for fixed point.