Journal of Instrumentation

FPGA-based tracking for the CMS Level-1 trigger using the tracklet algorithm

To cite this article: E. Bartz *et al* 2020 *JINST* **15** P06024

View the article online for updates and enhancements.

IOP ebooks™

Bringing together innovative digital publishing with leading authors from the global scientific community.

Start exploring the collection–download the first chapter of every title for free.

This content was downloaded from IP address 132.174.252.179 on 10/08/2020 at 16:50

Published by IOP Publishing for Sissa Medialab

Received: *October 23, 2019*
Revised: *April 9, 2020*
Accepted: *May 21, 2020*
Published: *June 23, 2020*

FPGA-based tracking for the CMS Level-1 trigger using the tracklet algorithm

E. Bartz,[a] G. Boudoul,[b] R. Bucci,[c] J. Chaves,[d,1] E. Clement,[e] D. Cranshaw,[d] S. Dutta,[f] Y. Gershtein,[a] R. Glein,[g] K. Hahn,[h] E. Halkiadakis,[a] M. Hildreth,[c] S. Kyriacou,[a] K. Lannon,[c] A. Lefeld,[i] Y. Liu,[h] E. MacDonald,[g] N. Pozzobon,[j] A. Ryd,[d] K. Salyer,[c] P. Shields,[c] L. Skinnari,[k] K. Stenson,[g] R. Stone,[a] C. Strohman,[d] K. Sung,[d] Z. Tao,[d,2] M. Trovato,[h,3] K. Ulmer,[g] S. Viret,[b] B. Winer,[i] P. Wittich,[d,4] B. Yates[i] and M. Zientek[d]

[a]*Rutgers University, New Brunswick, NJ, U.S.A.*
[b]*Université de Lyon, Université Claude Bernard Lyon 1, CNRS/IN2P3, IP2I Lyon, UMR 5822, Villeurbanne, France*
[c]*University of Notre Dame, South Bend, IN, U.S.A.*
[d]*Cornell University, Ithaca, NY, U.S.A.*
[e]*University of Bristol, Bristol, U.K.*
[f]*Saha Institute of Nuclear Physics, Kolkata, India*
[g]*University of Colorado, Boulder, CO, U.S.A.*
[h]*Northwestern University, Evanston, IL, U.S.A.*
[i]*The Ohio State University, Columbus, OH, U.S.A.*
[j]*Università degli Studi di Padova and INFN Sezione di Padova, Padova, Italy*
[k]*Northeastern University, Boston, MA, U.S.A.*

E-mail: wittich@cornell.edu

Abstract: The high instantaneous luminosities expected following the upgrade of the Large Hadron Collider (LHC) to the High-Luminosity LHC (HL-LHC) pose major experimental challenges for the CMS experiment. A central component to allow efficient operation under these conditions is the reconstruction of charged particle trajectories and their inclusion in the hardware-based trigger system. There are many challenges involved in achieving this: a large input data rate of about 20–40 Tb/s; processing a new batch of input data every 25 ns, each consisting of about 15,000 precise position measurements and rough transverse momentum measurements of particles ("stubs"); performing the pattern recognition on these stubs to find the trajectories; and producing

[1]Currently at Johnson and Johnson, New Brunswick, NJ, U.S.A. .
[2]Currently at University of British Columbia, Vancouver BC, Canada.
[3]Currently at Argonne National Laboratory, Argonne, IL, U.S.A. .
[4]Corresponding author.

2020 JINST 15 P06024

the list of trajectory parameters within 4 µs. This paper describes a proposed solution to this problem, specifically, it presents a novel approach to pattern recognition and charged particle trajectory reconstruction using an all-FPGA solution. The results of an end-to-end demonstrator system, based on Xilinx Virtex-7 FPGAs, that meets timing and performance requirements are presented along with a further improved, optimized version of the algorithm together with its corresponding expected performance.

Keywords: Particle tracking detectors (Solid-state detectors); Pattern recognition, cluster finding, calibration and fitting methods; Trigger algorithms; Trigger concepts and systems (hardware and software)

ArXiv ePrint: 1910.09970

2020 JINST 15 P06024

Contents

1 Introduction

This paper describes a novel implementation of a charged particle trajectory reconstruction approach based on Field-Programmable Gate Arrays (FPGAs) for the CMS experiment [1] at the CERN LHC. The LHC accelerator complex will undergo major upgrades, to be completed in 2026, to increase the instantaneous luminosity to approximately $7\ 5\quad 10^{34}\,\mathrm{cm}^{\ 2}\mathrm{s}^{\ 1}$ [2]. The "High-Luminosity LHC" (HL-LHC) upgrades will enable searches for undiscovered rare particle physics processes as well as detailed measurements of the properties of the Higgs boson. The HL-LHC will collide proton bunches every 25 ns, and each of these bunch collisions (an "event") will consist of an average of 200 proton-proton (pp) collisions. Only a small fraction of these collisions are of interest for further studies. A fast real-time selection, referred to as the Level-1 (L1) "trigger", is applied to decide whether a given collision should be considered for further analysis. The L1 trigger is implemented in custom hardware. The number of overlapping pp collisions per event, referred to as pileup (PU), in the HL-LHC era represents a large increase over previous data-taking eras (200 at the HL-LHC versus 30 during LHC Run-2), resulting in a significant challenge to the CMS trigger system — new handles are required. One such handle is the inclusion of information from the charged particle

2020 JINST 15 P06024

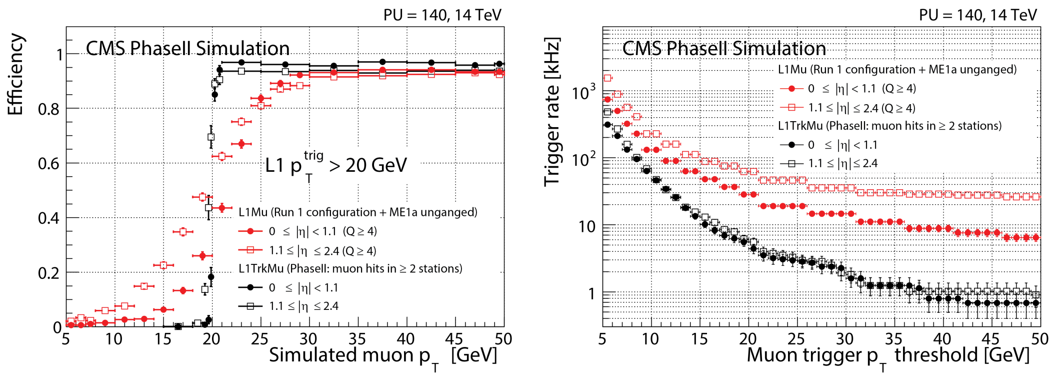

Figure 1. The efficiency of a single muon trigger with a L1 threshold of p_T 20 GeV as a function of muon p_T (left) and the trigger rate as a function of the muon trigger threshold (right), shown for the stand-alone muon trigger (red) and when including L1 tracking (black) for various pseudorapidity ( ) ranges [3].

tracking system. This will be the first time that information from a solid-state tracking detector has been included in the hardware trigger at the high collision rates of the LHC.

Integrating charged particle tracking in the L1 trigger will improve lepton identification and momentum measurements as well as provide track isolation and vertex reconstruction. These additional handles have the potential to reduce the L1 trigger rates while maintaining trigger thresholds acceptable for the CMS physics program. One example of this is shown in figure 1, where the efficiency of a single muon trigger as a function of the muon generated p_T (left) and the corresponding L1 trigger rate (right) are shown [3]. The red curves show the behavior of a stand-alone L1 muon trigger system, while the black curves show the performance of the same triggers when including L1 tracking. In particular, L1 tracking improves the momentum measurement, which translates to a sharper turn-on curve at the trigger threshold and hence a reduced trigger rate. The tracks must be delivered within 4 µs in order to be used in the trigger decision.

To reconstruct the trajectories of charged particles, the CMS experiment includes a tracking detector, which will be replaced for the HL-LHC operation. This upgraded tracking detector will consist of an inner tracker based on silicon pixel modules (not available in the L1 trigger), and an outer tracker based on silicon modules with strips and macro-pixel sensors [4]. The layout of the upgraded outer tracker, as proposed in ref. [4], is illustrated in figure 2.[1] Particles are produced at the interaction point and travel outwards in a 3.8 T uniform magnetic field, which is parallel to the z axis. The trajectory of a charged particle traversing this magnetic field is bent such that it forms a helix. In the r-   plane, the helix forms a circle. The radius of this circle is proportional to the momentum in this plane, the transverse momentum, or p_T, of the particle. The tracking detector is based on the concept that charged particles leave energy deposits ("hits") when crossing the sensitive detector material. In the upgraded outer tracker, closely spaced pairs of such hits will be linked on the detector front-end to form "stubs". With a sensor spacing of 1–4 mm, the relative position of the pairs of hits can be used to read out only those stubs that are likely to come from a particle with p_T 2 GeV (corresponding to a radius of curvature greater than 1.75 m). This momentum

[1]We use a right-handed coordinate system, with the origin at the nominal interaction point, the x axis pointing to the center of the LHC, the y axis pointing up (perpendicular to the LHC plane), and the z axis along the counterclockwise-beam. The azimuthal angle   is measured in the x-y plane.

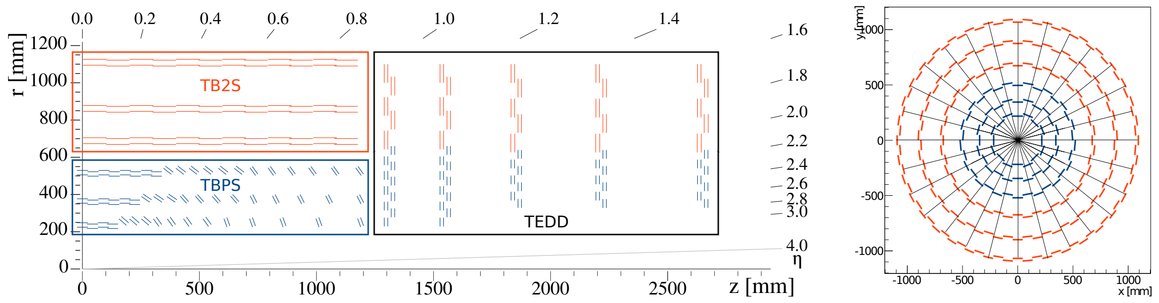

Figure 2. One quarter of the layout of the upgraded CMS charged particle tracking detector (left), as proposed in ref. [4], showing the Tracker Barrel with Pixel Strip modules (TBPS) or Strip-Strip modules (TB2S) and the Tracker Endcap Double-Disks (TEDD). Not shown is the inner pixel part of the CMS charged particle tracking detector, as it is not available to use for the L1 tracking. The LHC beams cross at (0,0). In the central barrel region (extending to $z \lesssim 125$ cm), there are six layers at radii from 23 cm to 110 cm. In the forward region (covering $z \gtrsim 125$ cm), there are five disks at z positions from 130 cm to 270 cm. The detector is divided into *sectors*, slices in that cover the entire length of the detector along z, for L1 track finding. In the x-y view of the barrel (right), 28 sectors are shown.

selection on the stubs reduces the readout bandwidth requirement by a factor of 10 [3]. In the L1 trigger, the stubs can be linked together to reconstruct the trajectories of the charged particles.

The upgraded outer tracker consists of a central portion with six detector layers parallel to the beam line, called the barrel, and five layers perpendicular to the beam line at large z, called the disks. The inner three layers in the barrel consist of so-called Pixel-Strip (PS) modules. The modules have a top sensor with 2.5 cm long silicon strips with 100 µm "pitch" (segmentation in the bending plane), and a bottom sensor consisting of 1.5 mm wide macro-pixels, again with 100 µm pitch. The inner three layers of the barrel are further divided into two regions: the "flat" region, near $z = 0$, where the modules are parallel to the beamline, and the "tilted" region, at higher z, where the modules are tilted toward the interaction point. The outer three barrel layers instead use "Strip-Strip" (2S) modules, where both sensors have 5 cm long strips with 90 µm pitch. The modules on each of the five disks in each half of the detector are arranged in concentric circles, or "rings", around the beamline. The two disks closest to the interaction point in each half of the detector have 15 rings (ten consisting of PS modules and five of 2S modules), and the remaining three have twelve rings (seven consisting of PS modules and five of 2S modules). Both PS and 2S modules provide a precise position and momentum coordinate for stubs in the transverse plane, while only the PS modules give a precise measurement of the z coordinate. Further details on the upgraded CMS tracker detector are available in ref. [4].

This paper discusses the linking of the stubs to form trajectories of charged particles, known as L1 tracking, and its associated challenges. In each collision of counter-rotating proton bunches (every 25 ns), about 15,000 stubs are formed. However, only about 10% of the stubs belong to trajectories of interest, so the majority of stubs needs to be filtered. The remaining stubs are combined to form, on average, 180 trajectories every 25 ns. This is the first time that data from the tracking detector is included in the CMS L1 trigger; previously, the amount of data to be processed and the computational complexity required, placed this task out of reach of FPGAs.

2020 JINST 15 P06024

To summarize, some of the challenges involved in reconstructing trajectories of charged particles for the CMS L1 trigger are:

- Absorb approximately 15,000 stubs arriving each 25 ns. The total estimated input bandwidth is in the range of 20–40 Tb/s, depending on the beam conditions and trigger conditions.
- Perform pattern recognition to identify the stubs that belong to a given trajectory, rejecting stubs from low-momentum particles.
- Fit the stubs to extract optimal trajectory parameters.
- Complete all above steps within the available processing time ("latency") of 4 µs, in order to feed into the decision of whether to retain the event, or discard it, before the on-detector data buffers are exhausted.

The "tracklet" approach for real-time track reconstruction in the hardware-based trigger system of CMS, presented in this paper, is one of three possible implementations that were considered by the collaboration. The other two approaches that were studied were a Hough-transform based approach using FPGAs [5] and an associative memory based approach using a custom ASIC [6]. The tracklet approach is a "road-search" algorithm, implemented using commercially available FPGA technology. Ever-increasing capability and programming flexibility make FPGAs ideal for performing fast track finding. The tracklet approach allows a naturally pipelined implementation with a modest overall system size. It also allows for simple software emulation of the algorithm. We present here the design of such a system and results from a hardware demonstrator system that implements end-to-end reconstruction, from input stubs to output trajectories, within the available latency and with a reasonable system size, for a slice of the detector. In addition to the results from the hardware demonstrator, some further improvements to the algorithm are presented.

Many software-based particle tracking algorithms use a road-search technique where track seeds are found and the trajectories extrapolated to look for matching stubs. This technique works well with the high-precision hits in particle detectors such as the CMS tracker. In the barrel part of the detector, the typical spatial position resolution of the stubs is about 30 µm in r– and either 0.5 mm (inner layers) or 1.5 cm (outer layers) in z in a cylindrical detector volume of about 2 m in diameter and 5 m in length [4]. Therefore, the search window (road) around the projected trajectory is small and the probability of finding false matches is low. However, with previous generations of FPGAs, the required computational power for implementing this type of tracking algorithm in the trigger was not available. Today, the large number of Digital Signal Processing (DSP) blocks and Random Access Memory (RAM) resources available in FPGAs make such an approach feasible. The use of FPGAs for the implementation of this algorithm provides a good match to the requirements. FPGAs provide high-speed serial, low-latency links that are well suited to bring the data into the FPGA for processing. The resources of FPGAs allow the implementation of a highly parallelized architecture, and their reconfigurability allows flexibility to changing needs. The precise space points from the stubs are used to determine trajectories of the particles using the DSP blocks in the FPGA. The DSP blocks are also used to calculate the final track parameters. The RAMs in the FPGA are used to implement the data movement and distributed storage required for this highly pipelined algorithm.

This paper is organized as follows. First, the tracklet algorithm is discussed in section 2. Two configurations of the system are discussed in this paper. The first configuration (referred to as "tracklet 1.0") corresponds to that employed for a test that implemented a vertical slice of the system as a hardware demonstrator. Since the hardware demonstrator, extensions to the tracklet approach have been implemented that further improve the load balancing and resulting physics performance. This corresponds to the second configuration (referred to as "tracklet 2.0"). Except as indicated, this section describes both tracklet 1.0 and tracklet 2.0. Section 3 explains the structure of the firmware, which is common for both configurations. In section 4 we detail the results of the hardware demonstrator test, using the tracklet 1.0 configuration. Section 5 describes the foreseen overall system architecture. Section 6 reports on the physics performance of the current system (tracklet 2.0). Further developments are foreseen. These are discussed in section 7, along with conclusions.

2 Tracklet algorithm

The goal of the real-time hardware-based track finding is to reconstruct the charged particle trajectories as an estimator of their momenta for particles with p_T 2 GeV, and to identify the track z_0 position with about 1 mm precision. The z_0 resolution is similar to the expected average separation of proton-proton collisions in the bunch collisions of the upgraded LHC, and thus allows precise determination of the collision vertices. The proposed tracklet method forms track seeds, "tracklets", from pairs of stubs in adjacent layers or disks. The tracklets provide roads to search for compatible stubs that are attached to form track candidates. A linearized ^{2}-fit determines the final track parameters.

The tracklet algorithm has been optimized to provide full coverage of the tracker with a small amount of redundancy in data duplication.

2.1 Algorithm overview

The tracklet algorithm proceeds in multiple steps, illustrated in figure 3. The algorithm begins with a seeding step, where tracklets are formed from pairs of stubs in adjacent layers or disks. An initial estimate of the tracklet parameters is calculated from the two stubs and using the detector origin as a constraint in the r– plane. Seeds are rejected if they are inconsistent with a track with p_T 2 GeV and z_0 15 cm. The seeding is performed in multiple pairs of layers (or disks) to ensure good efficiency for full azimuthal coverage and for redundancy in the system. Nominally, seeding between layers 1+2, 3+4, 5+6, between disks 1+2, 3+4, and between layers 1 or 2 and disk 1, are considered. Different seeding layers can be used in the event of malfunction of a particular layer or disk.

The tracklets are projected to other layers and disks to search for matching stubs. These projections use predetermined search windows, derived from measured residuals between projected tracklets and stubs in simulated data. The tracklets are projected both inside-out and outside-in, i.e., towards and away from the collision point, as needed for a given seeding combination. If a matching stub is found, the stub is included in the track candidate and the difference between the projected tracklet position and the stub position, "residual", is stored. If there are multiple stubs matched in a given layer or disk, the stub with the smallest residual is retained for the track fit.

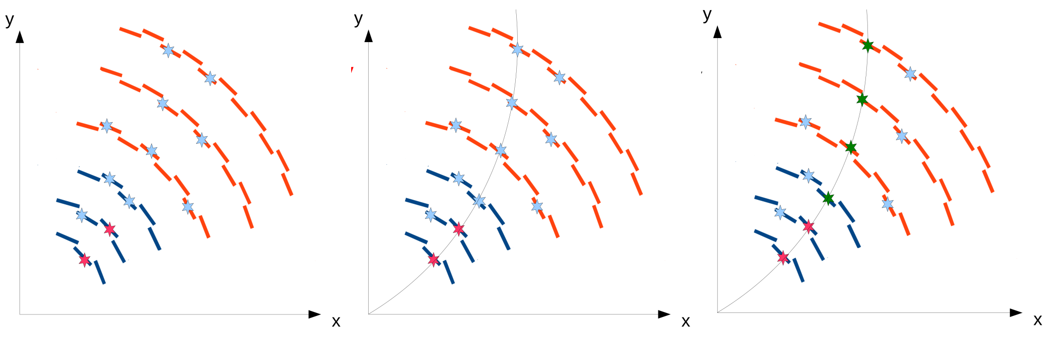

Figure 3. Illustration of the various steps of the tracklet algorithm. In the first step (left) pairs of stubs (red stars) are combined to form seeds, or tracklets, for the track finding. Combined with the interaction point (0,0) a helical trajectory for the particle is formed, assuming a uniform magnetic field. This trajectory is projected (middle) to the other layers. Stubs in the other layers that are close to the projection (green stars) are selected as matches (right) to the tracklet to form a track. Final trajectory parameters are calculated using a linearized ^{2} fit.

A linearized ^{2} fit is performed for all stubs matched to the trajectory. The track fit implementation uses pre-calculated derivatives and the tracklet-stub residuals from the projection step. The linearized ^{2} fit corrects the initial tracklet parameters to give the final track parameters: inverse radius of curvature (^{1}), azimuthal angle ($_0$), polar angle (tan), z intercept (z_0), and optionally the transverse impact parameter (d_0). The main source of duplicates is due to the same track, with the same hits, being found multiple times using different seeding layers. The initial duplicate removal we studied wanted to identify these duplicates and the use of the ^{2} from the fit was not initially considered. However, with further studies we have, in tracklet 2.0, now also considered the use of the ^{2} in the duplicate removal.

When implementing the algorithm on an FPGA, we work with fixed-precision math and low-order Taylor expansions of trigonometric functions. The number of bits kept are adjusted to ensure adequate precision. The loss of precision from using fixed-point calculations is negligible.

2.2 Parallelization

The algorithm is parallelized in the following manner. First, the detector is split along azimuth into sections, called “sectors”. The number of sectors is a tunable parameter and is chosen based on an optimization of the cost of FPGAs, the required per-sector input bandwidth, constraints due to the cabling interface of the on-detector electronics, the choice of time-multiplexing factor and the overall algorithm latency, as discussed in the following paragraphs. The nominal choice for the number of sectors is nine, while other values were also considered. For a given bunch crossing, each sector is assigned a dedicated hardware called the sector processor, and tracklet formation and track reconstruction are done in parallel in all of the sector processors. A small amount of data is duplicated near the boundaries of sectors to allow track formation to take place entirely within one sector processor and to avoid gaps in detector coverage. The data duplication scheme will be described in section 5.1.

The upper limit on the number of sectors is chosen such that a track with largest acceptable curvature ($p_T = 2\,\mathrm{GeV}$) is contained in at most two sectors. This corresponds to 28 sectors, and was the configuration used for the tracklet 1.0 hardware demonstrator. A minimal overlap for stubs in the even layers and disks were used, such that we can form the tracklets within a sector and maintain full seeding coverage. However, for matching stubs to projections in tracklet 1.0 we used a nearest neighbor sector communication for the projections that extended outside the sector. In addition to the projections that were sent to the nearest neighbor, matches that were found also needed to be sent back. This means that we had two steps in the algorithm where communication with neighboring sectors were needed. For the tracklet 2.0 implementation, the sector definition was revised to instead use 9 -sectors. These sectors are defined such that they include all stubs required to reconstruct tracks that pass through the sector at $r_{\text{crit}} = 55$ cm. This avoids the need for the nearest neighbor communication and, as a consequence, saves approximately 1 µs of latency.

The system of sector processors is replicated n times using a round-robin time multiplexing approach. Each system is entirely independent, and therefore, since new data are generated every 25 ns, each independent time multiplexed unit has to process a new event every n 25 ns. As with the number of sectors, the choice of time multiplexing factor n is driven by a balance of cost, efficiency, and needed processing power. For the current system, $n = 18$ is considered to balance these three factors; that is, each sector processor receives new data every 450 ns. For the 28 sector configuration, $n = 6$ was chosen, leading to each sector receiving data every 150 ns.

By construction, the system operates with a fixed latency. Each processing step proceeds for a fixed amount of time. If we have too many objects, some will not be processed, leading to truncation of processing and an algorithmic inefficiency.

The algorithm is further parallelized within sectors. In the serial algorithm, there are several places where loops over stubs or double loops over pairs of stubs are required. In a naive implementation, the time to process these parts of the algorithm scales as N or N^2, where N is the number of stubs in the sector, if considering all possible combinations. The number of combinations, or combinatorics, is a challenge to the algorithm. The combinatorics in forming tracklets and matching tracklet projections to stubs is efficiently reduced by dividing sectors into smaller units in z and to allow additional parallel processing. These smaller units are referred to as "Virtual Modules" (VMs). Only a small fraction of VM pairs can form a valid tracklet — the majority would be inconsistent with a track originating at the point of collision and with high enough transverse momentum. Data are distributed into those VMs satisfying these requirements in an early stage of the algorithm. This subdivision efficiently reduces the number of combinations that need to be considered by the algorithm from the start. Additionally, each VM is processed in parallel. At the next stage of the algorithm, the amount of parallelism is reduced when the accepted VM pairs' (the tracklets) initial track parameters are calculated.

Two different configurations of VMs are considered. In the first, used in tracklet 1.0 for the demonstrator, a sector is divided into $8z$ 3 ($8z$ 4 VMs for the odd (even) layers, resulting in 24 (32) VMs per layer. In forming the tracklet seeds, only 96 of the 768 (= 24 32) possible combinations are consistent with a possible tracklet that has the appropriate z_0 and maximum curvature (minimum momentum); the others do not need to be considered. It is by this method of data binning that the VMs eliminate the need to expend cycles considering combinations that are known a priori not to lead to a viable tracklet candidate.

In the second configuration (long VMs) used in tracklet 2.0, a finer segmentation is used. The sectors are here subdivided into 24 (32) VMs for the odd (even) layers. These VMs cover the

2020 JINST 15 P06024

full length of the sector but, internally to the VM, the data are collected into eight z bins. With this configuration, 120 of the same 768 combinations lead to viable tracklet seeds. The advantage of this configuration is that it leads to more even resource usage (better load balancing) and the binning in z allows a significant reduction in the combinatorics that has to be tried when forming tracklets and matching projections to stub, as will be demonstrated in section 4. In the firmware implementation, the difference between the two versions of the algorithm is restricted to a few places in the design.

3 Firmware implementation

The tracklet algorithm is implemented in the Verilog Hardware Description Language (HDL) [7] as nine processing steps and two transmission steps [8]. These processing steps are illustrated in figure 4. The red boxes are processing modules and the data are stored in memories (blue boxes) between the different processing steps. The black lines indicate which processing modules read and write data from which memory. The implementation of the algorithm in the FPGA takes place in the following steps.

2020 JINST 15 P06024

- *Stub organization*: (1) Sort the input stubs by their corresponding layer (**LayerRouter**), and (2) into smaller units in z and , the VMs (**VMRouter**).
- *Tracklet formation*: (3) Select candidate stub pairs for the formation of tracklets (**TrackletEngine**), and (4) calculate the tracklet parameters and projections to other layers (**TrackletCalculator** module).
- *Projections*: (5) Transmission of projections pointing to neighboring sectors (**ProjectionTranceiver**). (6) Route the projections based on smaller units (VMs) in z and (**ProjectionRouter**).
- *Stub matching*: (7) Match projected tracklets to stubs (**MatchEngine**), and (8) calculate the difference in position between the stubs and projected tracklet (**MatchCalculator**). (9) Transmission of matches between sectors (**MatchTransceiver**).
- *Track fit*: (10) Perform track fit; update the initial tracklet parameter estimate (**TrackFit**).
- *Duplicate Removal*: (11) Remove tracks found multiple times (**PurgeDuplicate**).

Both the tracklet 1.0 and tracklet 2.0 projects follow the same structure; they differ in the internals of some of the modules and how the modules are inter-connected.

Each of the steps outlined above corresponds to HDL modules (named in bold). These modules are hand-optimized. They can be customized with Verilog parameter statements on instantiation to account for differences between use cases. For example, in the second step of stub organization, six router modules are needed to process the stubs in each layer. The bit assignment in the data differs between the inner and outer three layers of the barrel. On instantiation, a parameter is used to select the appropriate version. The project illustrated in figure 4 corresponds to ¼ of the barrel for one sector for the tracklet 1.0 configuration. A complete project would contain approximately eight times as many instantiations of the same set of modules. The wiring between modules is specified in a master project configuration file. This configuration file is processed with python scripts to generate the top-level Verilog, which is then synthesized using Xilinx Vivado 2016.1.

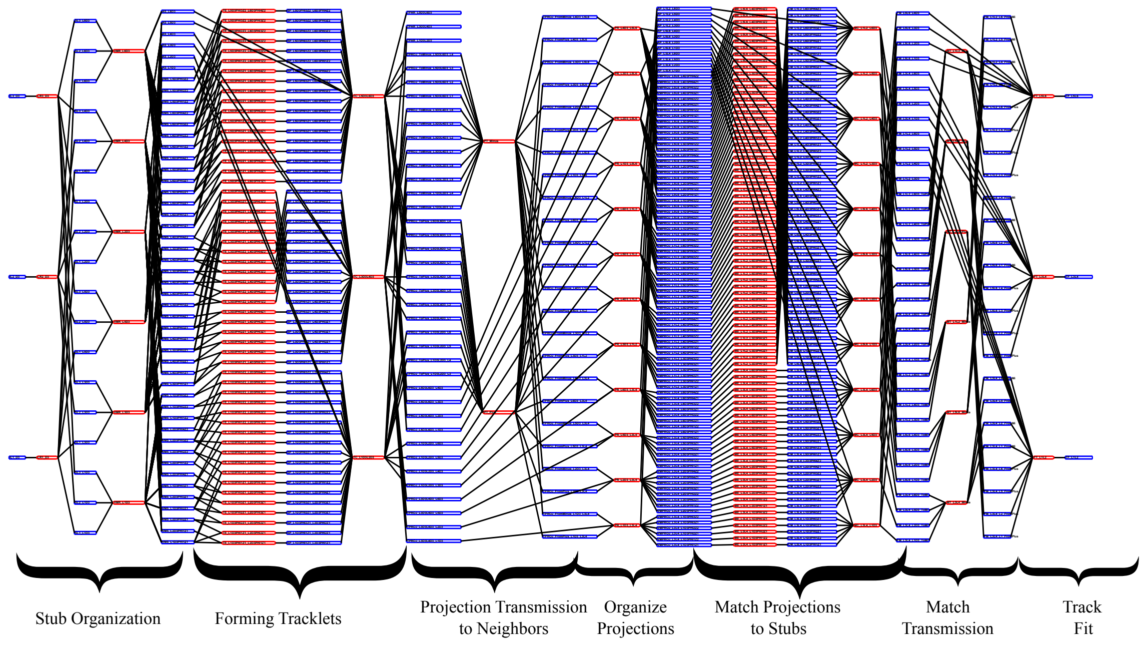

Figure 4. An illustrative high-level routing diagram for the tracklet project. As described in the text, the project consists of nine processing steps and two transmission steps indicated in red, and memories, shown in blue, in which the data are stored between the processing steps. The black lines show which processing block reads (and writes) from which memory. The input stubs arrive from the left in the picture and the first two processing modules sort the stubs into the correct VMs based on the physical location of the stubs. The next two stages involve the tracklet finding and projection calculation. Next the projections are routed to the correct VMs and then the projections are matched to stubs and the matches are then used to perform the track fit. The removal of duplicates is performed after the track fit (not shown for reasons of clarity).

These PYTHON scripts also generate the module connection diagram shown in figure 4 and drive a bit-level C++ emulation of the system.

All processing modules follow a similar format where the input is read from memories filled by the previous step and the output is written to another set of memories. All processing modules across all sector processors use a single common clock (currently, 240 MHz). As soon as a new event arrives, the next step in the chain will start processing the previous event. This implies that at any given time, several events are in the processing pipeline depending on the number of processing steps. Though the project as implemented in the demonstrator used the same clock for all processing steps, the fact that we use memories as buffers between the different steps allows the use of different clock speeds for different processing modules.

An event identifier propagates with the data and is used by the processing steps to access the appropriate data. We use the event identifier in the top bits of the memory address. This assumes a fixed maximum number of entries per event in the memory buffer. The fixed latency design implies that the maximum number of entries that can be processed is known and as such the limitation due to the fixed number can be understood and tuned. Most of the data from a processing step is only used in the next step and thus we can make very shallow buffers that will hold only two events at the same time (writing one and reading the other). These small buffers are implemented as distributed

2020 JINST 15 P06024

RAM in order to minimize the Block RAM (BRAM) resource usage in the FPGA. On the other hand, some data need to be stored for up to eight events since they will only be used later in the chain. These data are stored in BRAMs, but we try to minimize the usage of this resource as we have observed correlation of routing difficulties with the number of BRAMs used.

Since the calculations needed for routing the data are simple and using Lookup Tables (LUTs) is quick, most of the processing modules take only a few clock cycles to complete. We do not send the data to the next step immediately, but buffer it in memories until the allocated time is finished for the processing step. At this time, the module corresponding to the next step in the processing will start reading the data for the previous event and new data will be written for the current event. We use the true dual-port memories available in the Xilinx FPGAs for our buffers such that we can write the data from one event while simultaneously reading from the previous one. These dual-port memories also allow different modules to exist in separate clock domains.

In addition to the nine processing modules, we also implement two steps of neighbor communication using optical high-speed serial links. The bending of charged particles in the magnetic field can cause trajectories to curl into neighboring sectors. In this instance, the projected position of the track is sent across fiber links to the neighbor sector processor to look for matching stubs. Simultaneously as each sector processor is sending data to its left and right neighbors, it is also receiving from them as well for the same purpose. This system configuration was chosen to reduce the amount of data duplication globally at the cost of some increase in latency. As discussed in section 2, in the tracklet 2.0 implementation nearest neighbor communication is not needed.

3.1 Module examples

To illustrate the method in more detail, we present the functionality of two processing modules. Figure 5 shows schematically how the VMRouter works. The module receives a start signal every 150 ns for every new event. This VMRouter module reads stubs from three input layer memories. All stubs are written to the "AllStubs" memory in the full 36-bit format. In addition, based on their coordinates (and z), the stubs are routed to a specific output memory (VM stub memory) corresponding to a specific small area of the detector. Here, only coarse position information is retained and a six bit index into the AllStubs memory is saved such that we can later retrieve the precise stub position. The process loops over the input stubs and writes them out to different memories based on their position information.

A more complex example is the TrackletEngine processing module illustrated in figure 6. This module forms pairs of stubs as seed candidates. As such, this module reads input stubs from two VM stub memories filled by the VMRouter module described previously, but since we are interested in forming pairs of stubs, this module implements a double nested loop over all pairs. For each pair the coarse position information is used in two LUTs to check that the seed candidate is consistent with a trajectory with the p_T and z_0 requirements described above. If the stub pair passes this check, the indices of the stubs in the AllStub memories are saved in the output memory of candidate stub pairs. These indices are used in the next step, the TrackletCalculator, to retrieve the stubs and calculate the precise trajectory.

Figure 7 shows the distributions of the number of stub pairs that each TrackletEngine has to process. Since each step operates with a fixed latency, we have a maximum number of stub pairs that can be processed per event. With 450 ns per event and a clock speed of 240 MHz, a maximum

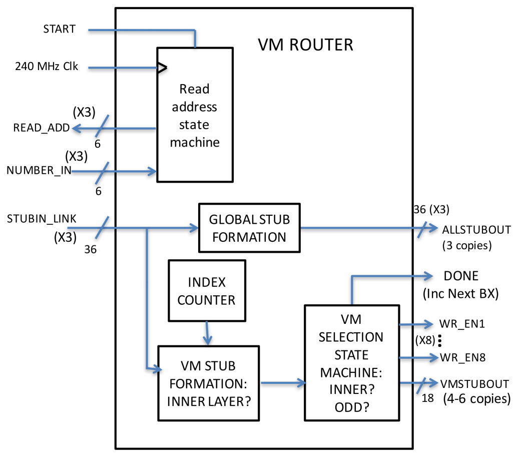

Figure 5. Schematic illustrating the connections of the VMRouter processing module. The module reads input stubs from the input memories and routes them to the correct VMs based on the stubs' coordinates (z and). A stub per clock cycle is read from the input stub memories and based on the most significant bits of the z and coordinates the stub information is written to the relevant VM memory.

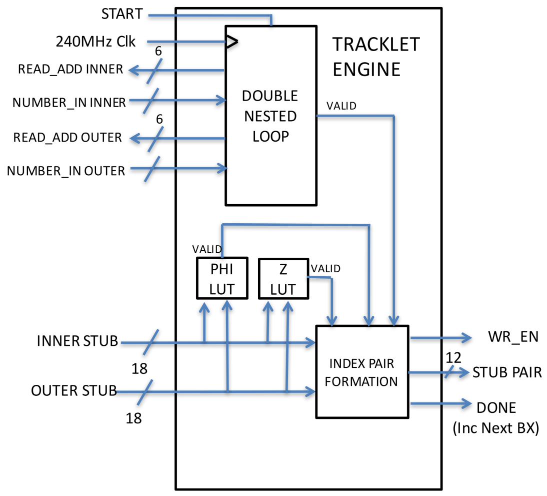

Figure 6. Schematic illustrating the connections of the TrackletEngine processing module. The module reads stubs from two VM memories. Two lookup tables are used to check consistency with the momentum and z vertex. If the pair of stubs pass the selection, a stub-pair tracklet candidate is written out.

of 108 input stub pairs can be considered. As can be seen in the figure, there are cases where there are more than 108 input stubs; the 109th stub-pair and later will not be processed and could lead to an inefficiency of the tracking algorithm. However, due to the built-in redundancy of seeding in multiple layers, the ultimate effect of this truncation on the final efficiency is observed to be small, as discussed in section 6.

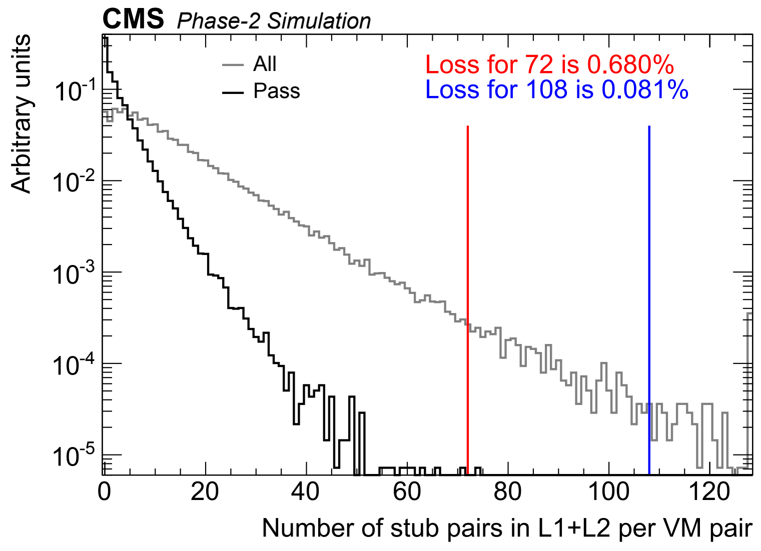

Figure 7. Simulation of the distribution of the number of stub pairs that TrackletEngines seeding in the two innermost barrel layers (L1+L2) have to process for $t\bar{t}$ events with an average pileup of 200. The grey curve shows the number of stub pairs that the module has to consider, while the black curve the number of stub pairs that pass. The blue line corresponds to 108 processing steps per bunch crossing and the red line to 72 processing steps. With a cut-off at 108 processing steps, we drop less than 0.1% of the stub pairs.

4 Demonstrator system

To explore the feasibility of the tracklet system, a demonstrator was completed in 2016. The goal of the test was to implement an end-to-end working system, using simulated data, to validate the design methodology and system modeling. The hardware used for the demonstrator system were MicroTCA boards with a Xilinx Virtex-7 (XC7VX690T-2) FPGA [9] and a Xilinx Zynq-7000 SoC for configuration and outside communication. These so-called Calorimeter Trigger Processor (CTP7) boards [10] were developed for the current CMS trigger [11, 12]. Implementing a full sector in one FPGA on a processing board was out of reach for the older Virtex-7 class FPGAs. For this reason, the demonstrator system focused on the implementation of a half-sector. The simulated data were derived from a Geant-based simulation of the CMS detector [13]. Data sets used included single particle (electron, muon, and pion) events, as well as fully simulated top quark-antiquark ($t\bar{t}$) events. To accurately simulate HL-LHC conditions, up to 200 extra pp collisions were included in addition to the event under study.

The demonstrator system consisted of three sectors and one time-multiplexing slice. A total of four MicroTCA processing blades were used, one for the central sector, two for its nearest neighbor sectors, and one blade that acted as a data source (providing input stubs) and a data sink (accepting the final output tracks). The system configuration is shown in figure 8. The demonstrator was fed with simulated data derived from the simulation of the CMS detector mentioned above. An Advanced Mezzanine Card (AMC13) [14] provided the central clock distribution. The inter-board communication used 8b/10b encoding with 10 Gb/s link speed. The demonstrator system is shown in figure 9. The demonstrator system assumed 28 sector processors and a time multiplexing factor of six, leading to new data arriving every 150 ns.

2020 JINST 15 P06024

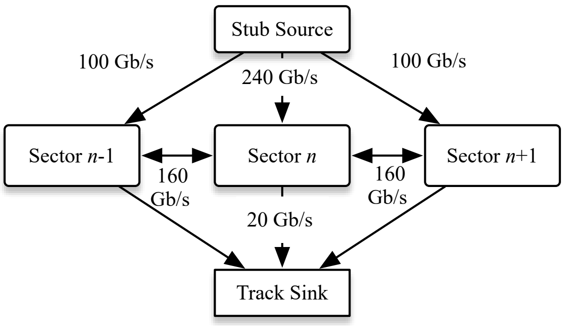

Figure 8. Schematic of the demonstrator system. Three MicroTCA blades implemented three sectors and a fourth blade served as the source and sink of data. The central sector processor was the actual system under test.

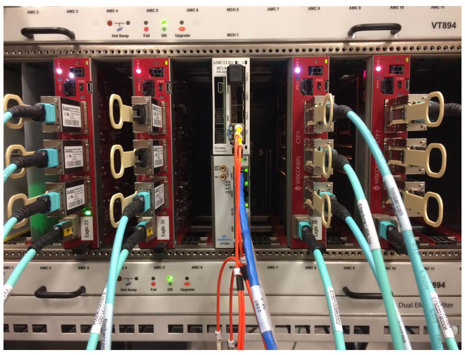

Figure 9. The demonstrator test system. The system was based on the CTP7 MicroTCA blade, used in the current CMS trigger. The system consisted of four CTP7s. Each CTP7 has 63 input and 48 output 10 Gb/s optical links.

Two complete implementations of the firmware project — one for half the barrel ($+z$) and one for a quarter of the barrel plus the forward ($+z$) endcap — were used to demonstrate the feasibility of this approach for the full range of the detector. These two projects covered two regions of the detector:

1. The barrel-only region. These tracks only traverse the cylindrical part of the detectors.
2. The hybrid region. Tracks that traverse both a part of the barrel region as well as the disk region of the detector.

By dividing the projects into these two regions, we exercised all regions of the detector while still working within the constraints of the Virtex-7 FPGAs.

A sketch of the tracker regions covered by each of the implementations is shown in figure 10. As discussed in section 3, data formats and calculations are slightly different in the two regions of the detector. The differences were absorbed using parameter statements in the Verilog code and determined at module instantiation time.

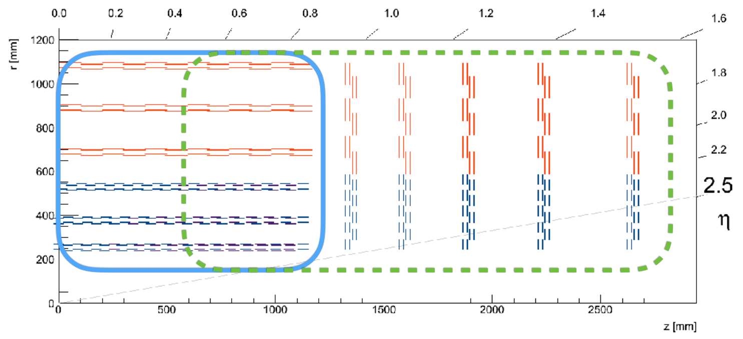

Figure 10. Sketch of the $+z$ portion of the detector in the r-z plane. A box in blue shows the detector region covered by the half-barrel implementation. A box in green shows the detector region covered by the other project that spans a quarter of the barrel, the transition region between the barrel and endcaps, and the endcaps.

2020 JINST 15 P06024

4.1 Validation and testing

Events were processed through the demonstrator as illustrated in figure 8. First, input stubs obtained from simulations were written to a piece of hardware that emulates the data delivery in the final system. On a GO signal, stubs were sent to the three sector processor boards. A new event was sent to each sector board every 150 ns. The events were processed and projections and matches were sent to and received from neighboring boards. The final output tracks were received by the track sink board. Systematic studies were performed to compare the integer-based emulation of the tracklet algorithm with a HDL simulation of the FPGA using Xilinx Vivado, as well as with the output tracks from the demonstrator system. Full agreement was observed in processing single-track events between the emulation, FPGA simulation, and board output. Better than 99.9% agreement was observed with many-track events with high pileup (figure 11). The demonstrator had a 28-fold azimuthal symmetry (i.e., 28 sectors), so we tested the full $+z$ range by using different input data, corresponding to the different sectors, without any modifications of the demonstrator itself.

4.2 System latency

For both the demonstrator system as well as the current configuration, each processing step of the tracklet algorithm takes a fixed number of clock cycles to process its input data. The processing modules' latency from receiving upstream data to producing the first result varies between 1–50 cycles depending on the module. Each module then continues to handle the data of the same event and writes to the memories for 150 ns (for a time multiplexing factor of six) before switching to the next event. For some of the steps where data transmission between the neighboring sectors was necessary, latency due to inter-board links was also included. The measured transmission latency was 316.7 ns (76 clock cycles), which included serialization and de-serialization via the Xilinx GTH transceivers, data propagation in 15 m long optical fibers, channel bonding, and time needed to prepare and pass data from processing modules to the transceivers. The total latency of the algorithm was therefore the sum of the processing module latencies and processing time, as well as inter-board data transmission latency, of all the processing steps. The latency of the hardware demonstrator also included the data transmission latency for receiving stubs from and sending

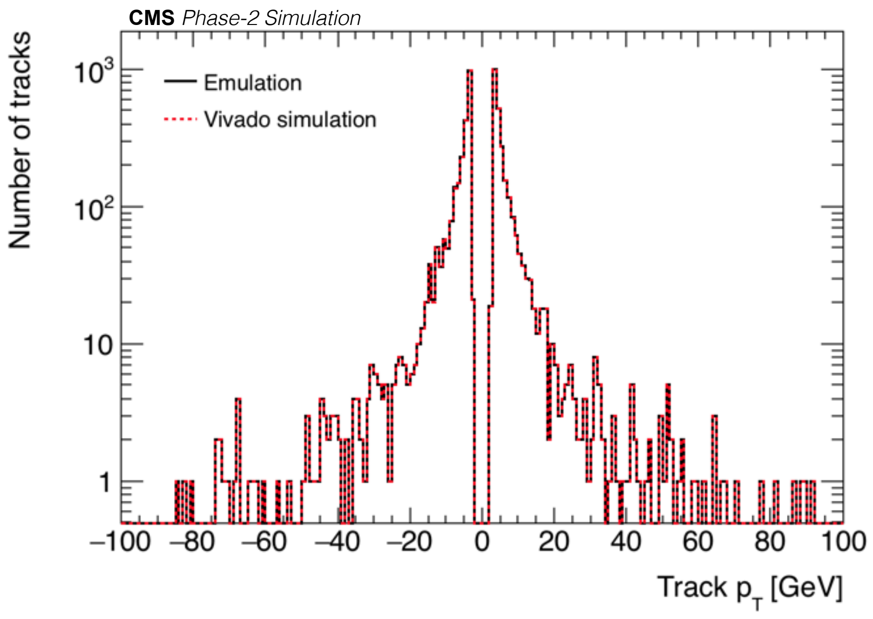

Figure 11. Comparison of the final track p_T between C++ emulation and HDL simulation, for simulated $t\bar{t}$ events with 200 pileup events. Better than 99% agreement is observed.

final tracks back to the data source/sink blade. A summary of the estimated latency is shown in table 1. With a 240 MHz clock and a time-multiplex factor of six, the total estimated latency is 3345.8 ns. The total latency of the demonstrator was also measured with a clock counter on the data source/sink blade. We started the counter when sending out stubs, and record the counter outputs when receiving valid tracks. The actual measurement was done with 240 MHz processing clock. The measured latency was 3333 ns, which agrees within three clock cycles (0.4%) with the model.[2]

For the tracklet 2.0 release, we are implementing the algorithm in C/C++ using the Xilinx Vivado HLS tools [15]. We believe HLS will produce code that will be easier to maintain and provide a lower barrier for entry of new people to contribute to the development of the project. Tracklet 2.0 provides several improvements over the algorithm as implemented for the demonstrator. The algorithmic improvements from using the "long virtual modules" reduce the effects of truncation and the new sector definitions remove the need for nearest neighbor communication. One significant change is that the TMUX factor is now 18, which results in 450 ns of processing time. This time increase is offset by the reduction in the number of processing steps achieved removing the steps that require communication with the nearest neighbors. This allows us to keep within the 4 µs latency target. Further reductions in latency are possible by combining processing steps into fewer modules. The estimated latency with this new configuration is shown in table 2.

The half-sector project included seeding in multiple layer and disk combinations (L1+L2, L3+L4, D1+D2, and D3+D4). This project consisted of the following processing modules: 12 LayerRouters, 22 VMRouters, 126 TrackletEngines, 8 TrackletCalculators, 22 ProjectionRouters, 156 MatchEngines, 22 MatchCalculators, 4 TrackFits, and one PurgeDuplicate. The resources used

[2]We did not try to understand the source of the final missing three clock cycles, as an example of the law of diminishing return.

2020 JINST 15 P06024

Table 1. Demonstrator latency model using the tracklet 1.0 configuration. For each step, the processing time and latency is given. For steps involving data transfers, the link latency is given. The model and measured latency agree within 0.4% (three clock cycles).

Step	Proc. time (ns)	Step latency (clock cycles)	Step latency (ns)	Link delay (ns)	Step total (ns)
Input link	0.0	1	4.2	316.7	320.8
LayerRouter	150.0	1	4.2	-	154.2
VMRouter	150.0	4	16.7	-	166.7
TrackletEngine	150.0	5	20.8	-	170.8
TrackletCalculator	150.0	43	179.2	-	329.2
ProjectionTransceiver	150.0	13	54.2	316.7	520.8
ProjectionRouter	150.0	5	20.8	-	170.8
MatchEngine	150.0	6	25.0	-	175.0
MatchCalculator	150.0	16	66.7	-	216.7
MatchTransceiver	150.0	12	50.0	316.7	516.7
TrackFit	150.0	26	108.3	-	258.3
PurgeDuplicate	0.0	6	25.0	-	25.0
Output link	0.0	1	4.2	316.7	320.8
Total	1500.0	139	579.2	1266.7	3345.8

Table 2. The estimated latency without sector-to-sector communication and with a time-multiplex factor of 18. The same clock speed (240 MHz) as in the demonstrator is assumed. We assign conservatively 150 ns for the link latency, as studies suggest it could be reduced by 200 ns compared to the demonstrator version with improved link speed and protocol.

Step	Proc. time (ns)	Step latency (clock cycles)	Step latency (ns)	Link delay (ns)	Step total (ns)
Input link	0.0	1	4.2	150.0	154.2
VMRouter	450.0	4	16.7	-	466.7
TrackletEngine	450.0	5	20.8	-	470.8
TrackletCalculator	450.0	43	179.2	-	529.2
ProjectionRouter	450.0	5	20.8	-	470.8
MatchEngine	450.0	6	25.0	-	475.0
MatchCalculator	450.0	16	66.7	-	516.7
TrackFit	450.0	26	108.3	-	558.3
PurgeDuplicate	0.0	6	25.0	-	25.0
Output link	0.0	1	4.2	150.0	154.2
Total	3150.0	113	470.8	300.0	3920.8

2020 JINST 15 P06024

Table 3. Summary of post-synthesis resource utilization on the Virtex-7 FPGA for the half-sector project.

Resource type	Estimated utilization (%)
Lookup tables	46
LUT memory	29
Flip-flops	39
Block RAMs	61
Digital signal processing blocks	21
Input and outputs	8
Gigabit transceivers	80
Global clock buffers	38
Mixed-mode clock manager modules	15

Table 4. Estimated FPGA resource utilization based on simulation of the Tracklet 2.0 full sector project, compared to the resources available in the Xilinx Ultrascale+ VU7P device.

Resource	Unit	VU7P	Tracklet 2.0	Fraction used (%)
18kbit BRAMs	count	2810	1108	39
DRAM	Mb	24	4.55	18
288kbit UltraRAMs	count	625	200	32
DSP	count	4560	1696	37

in the demonstrator project are shown in table 3. The resource usage from the VERILOG synthesis for the full sector project is summarized in table 4. The most heavily used resource was BRAMs. We ran the project at 240 MHz, limited by external constraints. First, we operated the links at 10 Gb/s with a 8b/10b encoding. This meant that we transferred data packages of 32 bits at 250 MHz. We also wanted the clock frequency to be a multiple of the 40 MHz bunch collision rate in CMS. This meant that if we operated at 240 MHz and produced 32 bit data packages at this clock speed the links could transport the data. Increases in clock speed are being considered as a future improvement.

5 Hardware platform and system architecture

A high-level overview of the tracker data flow is shown in figure 12. The data from the tracker are sent off the detector to the Data, Trigger, and Control (DTC) cards [4]. About 90% of the bandwidth is dedicated to a reduced-precision data stream that is sent to the L1 tracking system, or the "track trigger" (labeled TT in the figure and described in this paper). Data from the track trigger are sent downstream to a part of the L1 Trigger system (L1T) called the correlator system, where the information from the tracking detector is combined with information from other subdetectors to identify electrons, muons, and other physics quantities.[3] Based on these data, a L1 Accept (L1A) is issued or not. If the L1A is issued, a signal is sent to the DTC, which then initiates the readout

[3]These quantities are used to decide if the event should be dropped or stored for further analysis. In total, CMS can store about 0 0025% of the collisions for offline study.

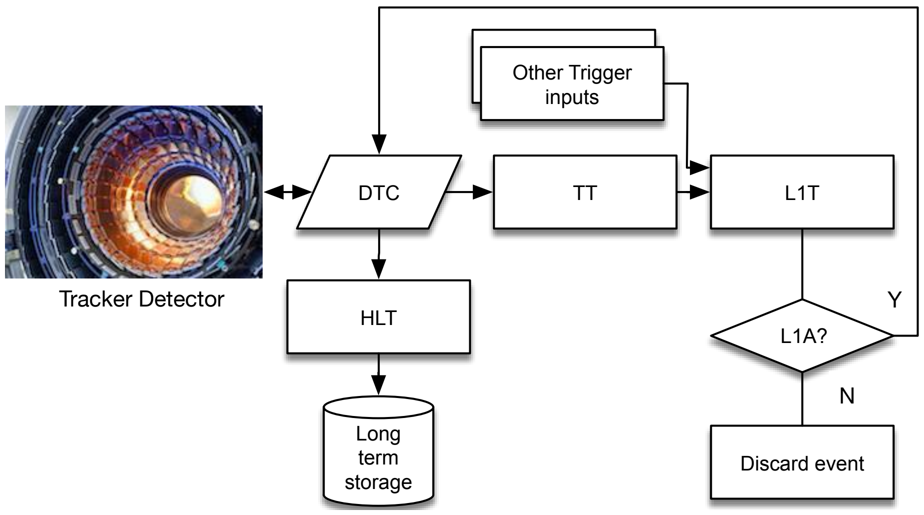

Figure 12. High-level overview of the data flow for the tracker data, showing how the track trigger is integrated into the detector readout and the trigger flow.

of the full-precision data from the detector for that bunch crossing. These data are then sent to secondary software trigger processors (the High-Level Trigger, HLT) for potential storage to disk. If the L1A is not issued, the data are discarded by letting the on-detector data buffers expire. As can be seen in the diagram, the track trigger receives its data upstream from the DTC cards and sends the results of its processing downstream to the L1 trigger.

As was mentioned in section 2.2, the processing of the L1 tracking occurs in sectors that span the entire polar angle and a slice in . Each sector is assigned a dedicated hardware unit, the sector processor. The system design centers around these sector processors. Each sector processor is foreseen to be a single ATCA blade with a Xilinx Virtex Ultrascale+ FPGA. The heart of the tracklet approach is this FPGA. It must have adequate DSP resources (about 2000 DSP48E2 equivalent units), I/O (about 50 high-speed serial transceivers running at 28 Gb/s), 2800 18 kb BRAMs, approximately 2.5 Mb of distributed LUT RAM, and adequate LUT resources. These requirements are met by the Xilinx Virtex UltraScale+ family. The input data (stubs) will be received from the upstream DTCs via 24–36 high-speed optical serial links running at 28 Gb/s. Output data (reconstructed tracks) will be sent downstream over a single 25 Gb/s link. The details of the foreseen cabling for the system is discussed in the section below.

The complete set of sector processors is duplicated a small number of times (time multiplexing of 18 is currently under consideration) and data are distributed to these identical copies in a round-robin scheme. For a system with 9 sectors and a factor of 18 time-multiplexing, the complete system will consist of 162 blades.

The trigger is implemented in custom hardware and sits in a well-shielded cavern away from the detector; as such, radiation and single-event upsets are not a concern. Initial estimates suggest that the per-slot power requirements will be well below the ATCA specification of 400 W per slot.

5.1 Cabling

The baseline strategy for the project assumes a tilted barrel geometry [4], 9 -sectors for track finding and a time-multiplex factor of 18 for the tracklet 2.0 configuration. The DTC layout assumed here is as follows: a total of 108 DTCs for a half-detector (+ and z); 54 for the PS modules, connected to the inner three layers of the half-barrel and the inner seven rings of the endcap, and 54 for the 2S modules, connected to the outer three layers of the half-barrel and the outer five rings of the endcap. The DTCs are arranged such that they cover "nonants" in (nonants is referring to the currently proposed division of the readout of the outer tracker in a nine-fold symmetry) over a half-length of the tracker, and so each track finder nonant and half-length connects to 12 DTCs (that is, 108 DTCs per half-length in z split into nine in). Within each nonant and half-length, the modules connected to each DTC are distributed in both the barrel and disk regions in order to better balance the stub load in each DTC, and to minimize or avoid multiple hits along a track being routed to a single DTC. A summary of the DTC layout within a nonant and half-length is shown in table 5. We also assume a maximum of 72 modules connected per DTC, and that each DTC handles a maximum output of 672 Gb/s. This system also requires that the sector processor boards are arranged within two ATCA shelves for each n^{th} time-multiplexed slice.

As mentioned above, the DTCs will process the data from each of the modules in the tracker. Each DTC will connect to one front-end cable, and each cable will handle 144 fibers; 72 front-end fibers running at 10 Gb/s (inner detector layers) or 5 Gb/s (outer detector layers) coming in, and 72 front-end fibers running at 2.5 Gb/s going out. The DTC sends data to the track trigger using 28 Gb/s cables, with a maximum capacity of 672 Gb/s for each DTC (i.e., 24 28 Gb/s links). The global coordinate information is sent to the track finder and uses 36 bits per stub, and with 64/66 bit encoding we assume a total of 45 bits per stub.

The layout of the DTCs' connections to the detector modules divides the detector into -nonants, and into two halves along z. Twelve DTCs are connected to each -nonant and z half, for a total of 12 18 = 216 DTCs assuming a 9-sector processor configuration; this corresponds to one sector processor per nonant. However, due to the curvature of tracks within the detector, a small amount of data duplication is needed near the boundaries of sectors to ensure that track formation can take place entirely within one sector processor. The tracker is divided into 9 sectors at a critical radius r_c. Tracklets that are generated within a sector and, during the projection step, produce trajectories that cross r_c within a particular sector, are kept in that sector. Tracklets that during the projection step instead cross r_c in an adjacent sector are discarded (they will be rediscovered by the adjacent sector). At radii above and below r_c, curves corresponding to the lowest acceptable p_{T} are drawn from each edge of a sector, and all stubs that lie within these curves have the potential to form tracks within the sector, and so must be sent to the corresponding sector processor. The curves in adjacent sectors will overlap at radii above and below r_c, and it is in these small regions where data duplication is necessary. The parameter r_c is tunable and can be chosen based on an optimization of the algorithmic precision, algorithmic efficiency, load balancing in the DTCs, and constraints on the number of links available to the DTCs. If the nine sector processors were aligned with the DTC -nonants, the overlap regions on either side of a sector would require each DTC to communicate with three sector processors. To reduce the number of required output links from the DTCs, the 9 sectors are offset from the nonants such that each of the DTC nonants only needs to communicate with at most two sector processors.

2020 JINST 15 P06024

Table 5. DTC layout in each nonant and half-length in z. This layout yields a total of 216 DTCs. Each line corresponds to one DTC; there are 12 DTCs for each side of the detector (+z and z).

Module type	Connected detector regions			
PS	Layer 1 Flat Region	Disk 1, Rings 1–7	Disk 3, Rings 1–3	Disk 5, Rings 1–3
	Layer 1 Tilted Region	Disk 2, Rings 5–7	Disk 4, Rings 1–3	
	Layer 2	Disk 2, Rings 1–4		
	Layer 3 Flat Region	Disk 2, Rings 8–10		
	Layer 3 Tilted Region	Disk 4, Rings 4–7		
	Disk 1, Rings 8–10	Disk 3, Rings 4–7	Disk 5, Rings 4–7	
2S	Layer 4			
	Layer 5			
	Layer 6			
	Layer 6	Disk 3, Rings 8–12		
	Disk 1, Rings 11–15	Disk 4, Rings 8–12		
	Disk 2, Rings 11–15	Disk 5, Rings 8–12		

The cabling requirements are studied assuming 9 sectors, 18 time-multiplexed slices, and 28 Gb/s links from the DTCs to the sector processors. These cabling studies have been performed using simulated $t\bar{t}$ events, with an average of 200 overlaid pileup interactions, to determine whether, assuming the cabling configuration described above, 28 Gb/s cables have sufficient bandwidth to deliver the stubs from the DTCs to the sector processors. For a single 28 Gb/s link, one can transfer information for a maximum of 210 stubs, and the proposed cabling configuration was found to be sufficient to transfer the stub information from the DTCs to the appropriate sector processors with zero stub loss.

The described cabling scheme requires 36 output links from the DTCs for a time-multiplex factor of 18. This corresponds to 48 input links to each sector processor (24 DTCs/nonant 2 nonants) resulting in a total of 7776 total links between the DTCs and sector processors. We conclude from these studies that the I/O requirements of the algorithm are met by the system, and that 28 Gb/s links have enough bandwidth for transferring the stub information.

6 Physics performance

In this section the expected performance of the tracklet algorithm is discussed. The studies use simulated data derived from a Geant4-based simulation of the CMS detector [13]. To evaluate the performance, a C++ implementation of the algorithm is used. It models the limitations in the hardware implementation resulting from fixed-point arithmetic, finite buffer lengths, and similar constraints. This “emulation” algorithm, or emulator, truncates all calculations and bit widths as is done in the hardware, and implements all internal buffers and memories as they exist in the firmware. The firmware implementation, which runs on the hardware, is made to match the emulator bit-by-bit in our qualification process. The precision of the calculations used in the emulator is tuned to ensure sufficient accuracy to meet the required physics performance goals in the trigger.

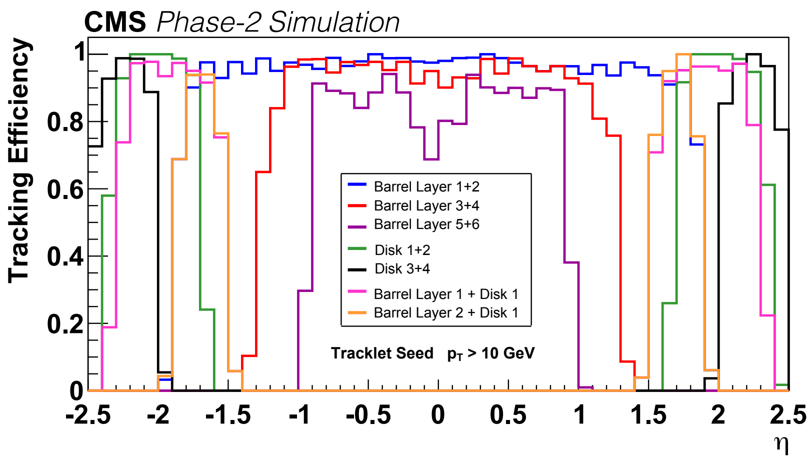

Figure 13. Final tracking efficiency for single muons with p_{T} 10 GeV in events without pileup, shown as a function of muon   for different layer and disk combinations in forming the seed tracklets. The overlapping curves demonstrate the redundancy in seeding as a function of pseudorapidity.

The demonstrator setup for tracklet 1.0 shows excellent agreement between the actual firmware results and the integer-based C++ emulation of the system. For single object events, the output tracks from the firmware have 100% bitwise compatibility with the integer-based emulation. For busier events, for example $\mathrm{t\bar{t}}$ with an average pileup of 200, the emulation and firmware tracks agree to better than 99%, as was shown in figure 11. These tests validate the use of the integer-based emulation code for extrapolating to future performance improvements. This section shows the performance of the tracklet 2.0 configuration.

6.1 Performance of different steps of the tracklet algorithm

The efficiency for identifying charged particle trajectories, using different seeding combinations, is shown in figure 13 for a simulated sample of events. Each event contains a single muon without any additional pileup interactions, using the integer-based C++ emulation of the algorithm. With the multiple seeding combinations, the same particle trajectory is often found multiple times, which ensures redundancy and a complete detector coverage. As illustrated in figure 14, already the resolution of the tracking seeds (tracklets) is good — within a factor of two or less of the resolution of the fitted track, motivating the feasibility of the road search algorithm. Given the resolution of the seeds, we can use narrow windows in projecting the tracklet seeds to other layers and disks to search for matching stubs. This is a key element in limiting the number of combinations that must be tried by the algorithm.

The track parameters found by the TrackletCalculator are used to project the tracklet to other layers and disks. This projection is performed in the TrackletCalculator to a nominal position of the layer or disk. In the MatchCalculator, once a candidate stub is found to which the projection is matched, an exact projection is calculated using the derivatives $_{\mathrm{proj}}$ r and z_{proj} r. A projection-stub match is accepted if the r– and z (or r in disks) residuals pass a selection criterium that depends on (i) in which layer or disk the match is found and (ii) which combination of layers or disks were used to form the tracklet. The residuals are tuned to be nearly 100% efficient for trajectories from prompt particles originating from the primary interaction vertex.

2020 JINST 15 P06024

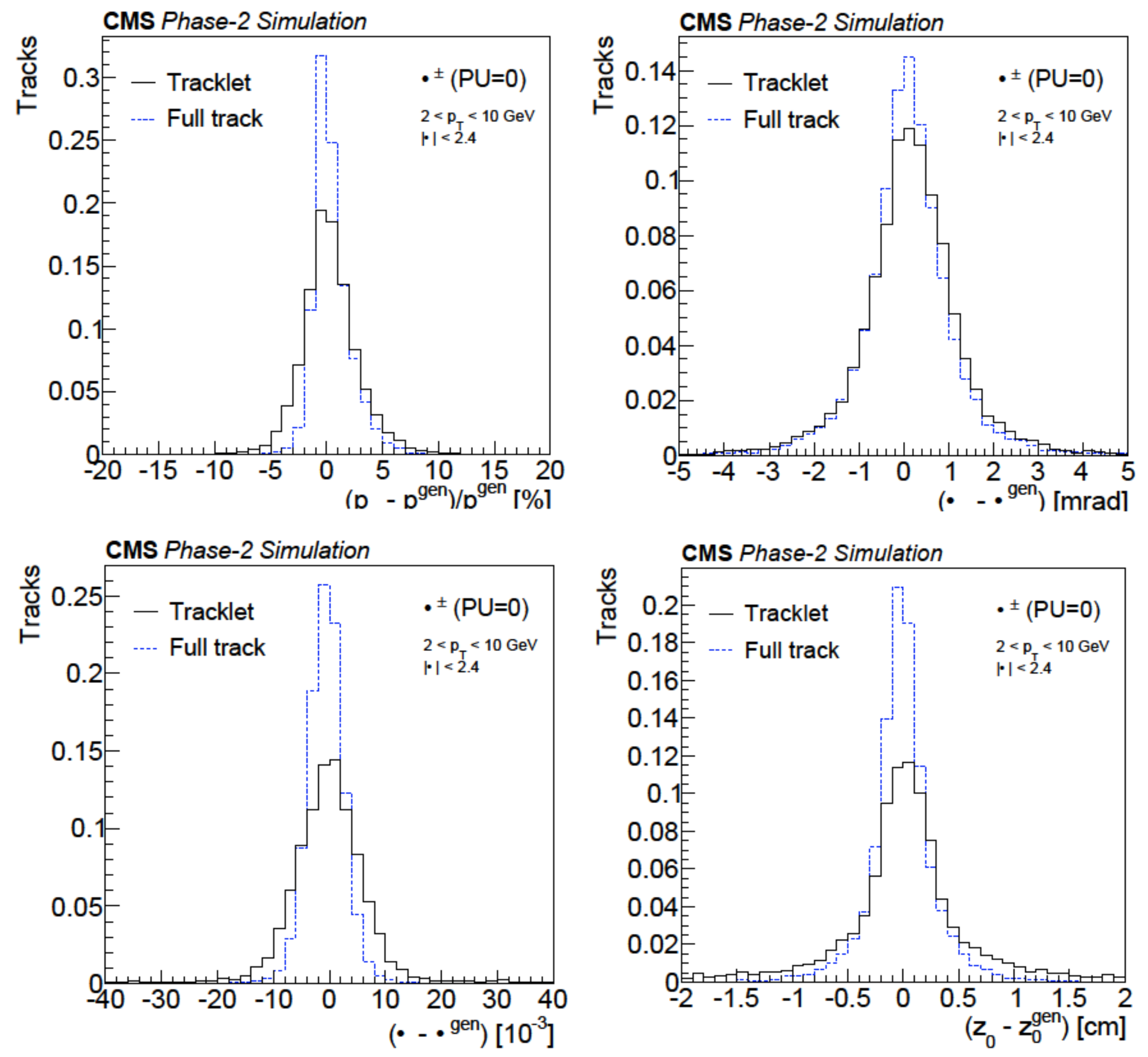

Figure 14. Residuals for relative p_T (top left), ϕ_0 (top right), η (bottom left), and z_0 (bottom right) for the tracking seeds, tracklets, as solid black curves as compared to residuals for the full, final track fit in dashed blue curves. The track parameters are calculated using fixed-point calculations and shown for single muons ($\mu^{\pm}$) with $2 < p_T < 10\,\mathrm{GeV}$ and $|\eta| < 2.4$ in events without pileup.

Next, the performance of the duplicate removal is studied. Figure 15 shows for single-muon events the number of tracks found per event prior to any duplicate removal and after the duplicate removal between tracks within a ϕ sector. For events containing a single muon, one track per event is expected, which is what is observed for most events after the duplicate removal step.

6.2 Final tracking performance

The estimated performance of the tracklet algorithm is studied with the integer-based emulation of the algorithm. The L1 tracking efficiency as a function of p_T and η for single muons or electrons (generated with a uniform p_T distribution) in events without pileup is shown in figure 16. Electrons have a lower efficiency compared to muons due to their higher interaction rate with the detector material at the momenta in question.

2020 JINST 15 P06024

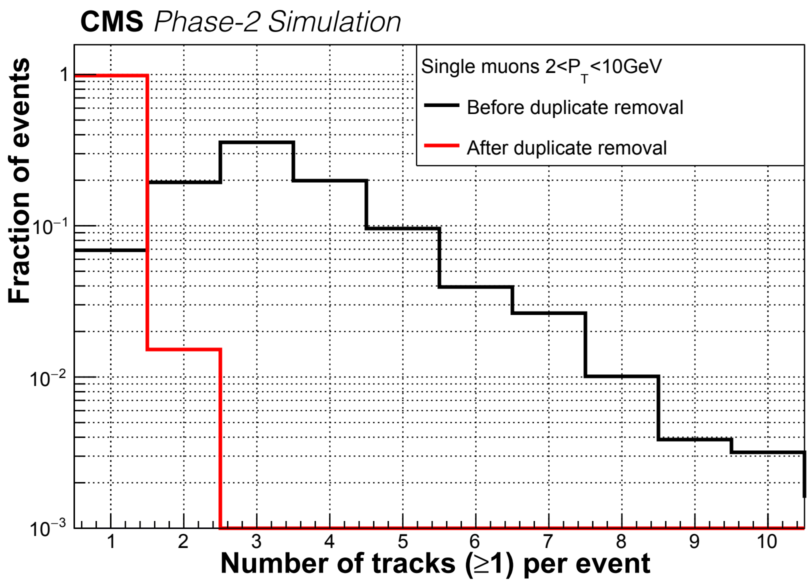

Figure 15. Number of tracks per event for single muons with 2 p_{T} 10 GeV in events without pileup. The number of tracks is shown before any duplicate removal (black) and after duplicate removal within sectors (red). As expected, after duplicate removal, most events have exactly one track. Only events with at least one track prior to duplicate removal are shown.

Similarly in figure 17, the L1 tracking efficiency as a function of p_{T} and is shown for charged particles from $\mathrm{t\bar{t}}$ production, in events with an average pileup of 0, 140, or 200. The efficiency is computed as implemented in the demonstrator, i.e., it includes the effects of truncating data. These effects are minimal, for two main reasons, (i) because of the large parallelization of the system, most of the modules are sparsely populated, (ii) the different seeding combinations provide additional redundancy that can recover tracks that may otherwise be lost.

The L1 track parameter resolutions for , , z_0, and the relative resolution in p_{T} are shown in figure 18 for charged particles from $\mathrm{t\bar{t}}$ production, in events with an average pileup of 200. The resolutions are shown as a function of for two different ranges in p_{T} (2 p_{T} 8 GeV and p_{T} 8 GeV). The z_0 resolution is about 1 mm in the central barrel region, similar to the average separation of pileup vertices. The z_0 resolution slightly worsens with increasing due to (i) the module tilts, and (ii), for 2 2, the fact that a charged particle in this region does not cross a barrel layer with PS modules but only disk modules, which due to their orientation do not provide a precise z measurement. The momentum resolution is about 1% for central and increases to about 4% for the outermost region. The momentum resolution is increased for higher values of due to the increased extrapolation distance to the beam axis, where the track parameters are calculated. The precise z_0 resolution allows the selection of tracks originating from a common vertex for use in L1 trigger algorithms and the accurate p_{T} resolution results in sharp muon trigger thresholds.

The track rates are shown in figure 19. It shows the truth track rate, namely the rate of reconstructable particles that produce stubs in at least four layers or disks in the outer tracker (corresponding to the "denominator" in the efficiency plots). Also shown is the total rate of reconstructed L1 tracks above 2 GeV, as well as the rate of tracks when requiring quality criteria

2020 JINST 15 P06024

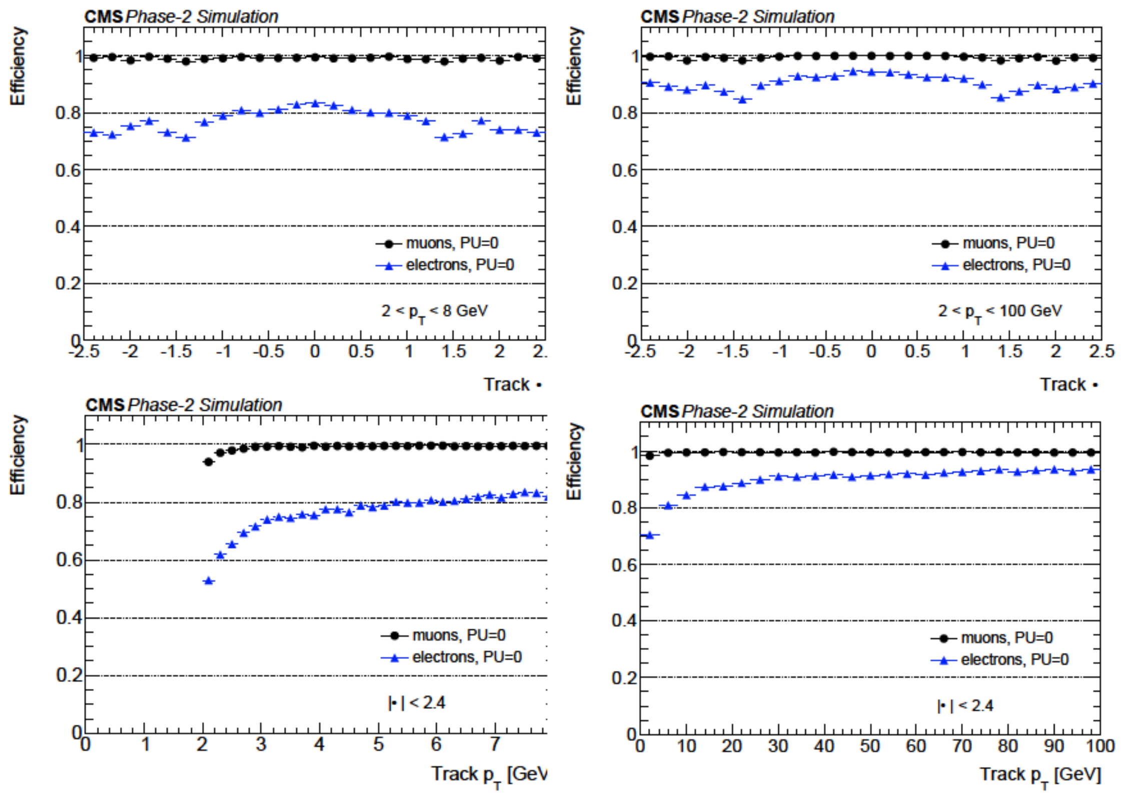

Figure 16. Efficiency as a function of η (top row) and p_T (bottom row) for muons and electrons with $2 < p_T < 8$ GeV (left column) or $2 < p_T < 100$ GeV (right column) in single-particle events without pileup. The propensity for electrons to radiate in the detector leads to a slower turn-on and lower plateau in the efficiency.

based on the stub bend information and the track fit χ^2. The quality criteria reduces the contribution from misreconstructed ("fake") tracks. These have, fractionally, a small contribution to the overall track rate, which is completely dominated by low-p_T tracks where the fake rate is small. Minimizing the rate of fake tracks is regardless important for some use-cases of the L1 tracks, e.g. track-based missing transverse energy determination or hadronic tau lepton reconstruction.

In summary, the track performance shown in this section achieves high track finding efficiency across the full p_T and η range covered. Additionally, track parameter resolutions are sufficiently precise for the downstream L1 trigger, where the tracks will either be correlated with information from other CMS detector systems, or used standalone to form tracker-only L1 trigger signatures.

7 Conclusions

For the High-Luminosity LHC upgrade, the CMS experiment will require a new tracking system that enables the identification of charged particle trajectories in real-time to maintain high efficiencies for identifying physics objects at manageable rates. The tracklet approach is one proposed implementation of the real-time track finding. The method is based on a road-search algorithm and uses commercially available FPGA technology for maximum flexibility. An end-to-end system demonstrator consisting of a slice of the detector in azimuth has been implemented using a

2020 JINST 15 P06024

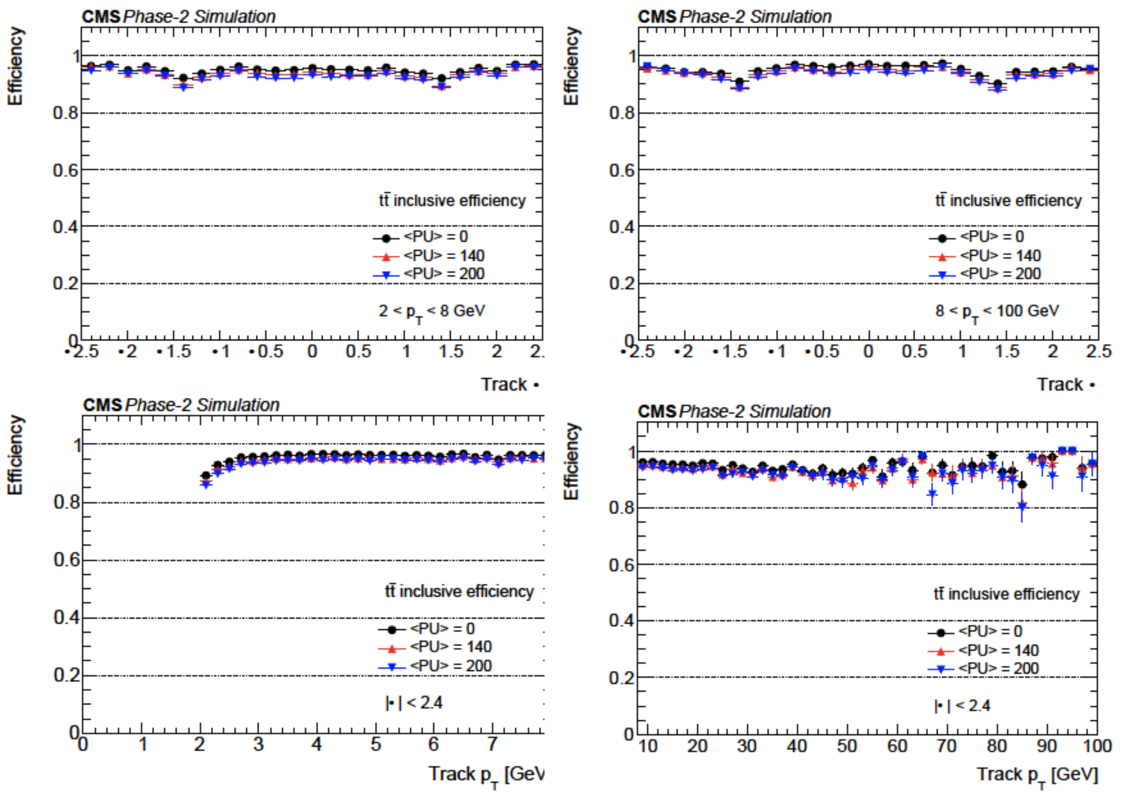

Figure 17. Efficiency as a function of η (top row) and p_{T} (bottom row) for charged particles in $t\bar{t}$ events, with an average pileup of 0, 140, or 200. Possible effects from truncation or the usage of fixed-point calculations are included.

Virtex-7 FPGA-based MicroTCA blade. The final system, which is to be deployed in 2025, will use future-generation FPGAs. To scale the demonstrator to the final system, only a small extrapolation is required. Currently, the demonstrator only covers the $+z$ side of the detector; in the full system, both sides will be covered. The detector is largely symmetric in $\pm z$, so the addition of the $-z$ side only results in increased occupancy, which is handled by more instances of already-existing HDL modules. Since more data are coming into the sector processor, about a factor of two, the total I/O requirements will increase by roughly a factor of three, taking into account both the increase of the total data rate and the cabling scheme of the new detector. These I/O requirements are within the capabilities of the specifications of the Xilinx Virtex UltraScale+ family of FPGAs. None of these changes represent more than an evolution of the demonstrator. The demonstrator has been used to validate the algorithm and board-to-board communication, to measure timing and latency, and to establish the algorithm performance. Studies from the demonstrator, processing events from the input stubs to the final output tracks, show that the tracklet algorithm meets timing and efficiency requirements for the final system.

The tracklet 1.0 demonstrator showed the feasibility of the tracklet approach for L1 tracking. Following this demonstration a number of algorithmic improvements were developed and described in this paper for tracklet 2.0. The L1 tracking algorithm was shown to achieve high efficiencies, e.g. an efficiency of 90% or higher for charged particles in $t\bar{t}$ events, as well as precise track parameter

2020 JINST 15 P06024

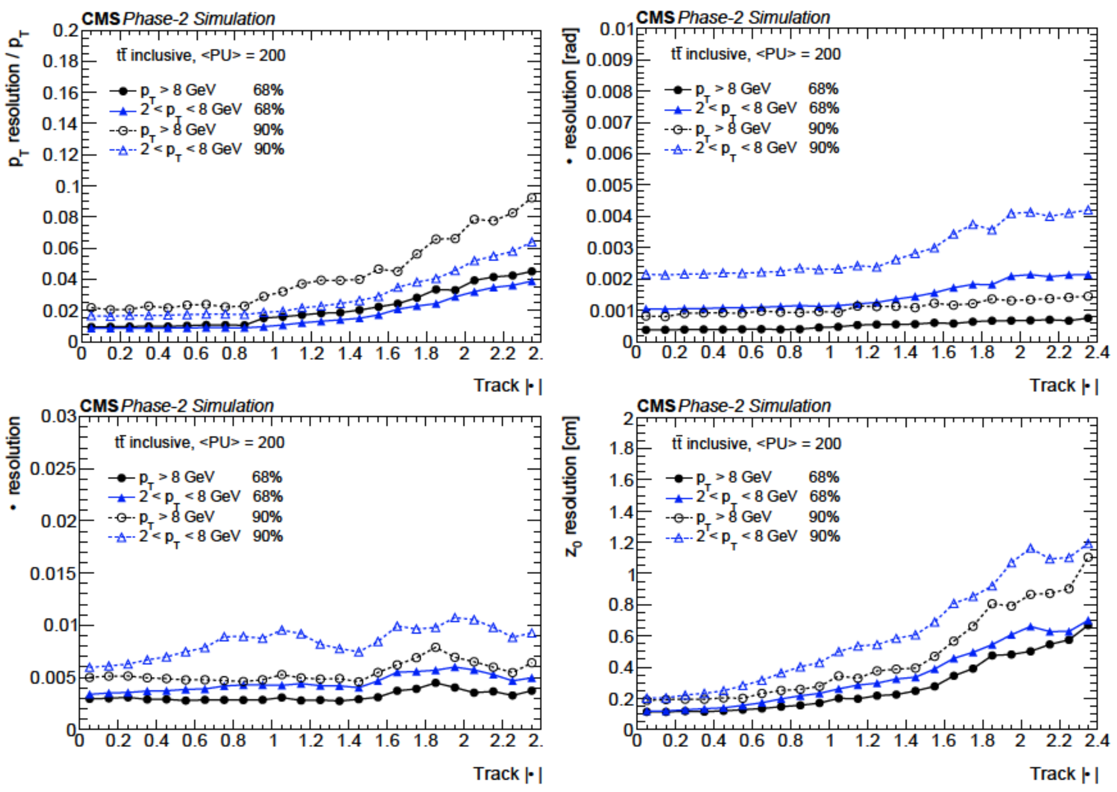

Figure 18. Relative p_T resolution ($\sigma(p_T)/p_T$) (top left) and resolution in ϕ_0 (top right), η (bottom left), and z_0 (bottom right) for charged particles in $t\bar{t}$ events with an average pileup of 200. The filled circles / solid lines correspond to the 68% confidence intervals, while the open circles / dashed lines correspond to the 90% confidence intervals. The resolutions are shown separately for tracks with $2 < p_T < 8$ GeV and $p_T > 8$ GeV. Possible effects from truncation or the usage of fixed-point calculations are included.

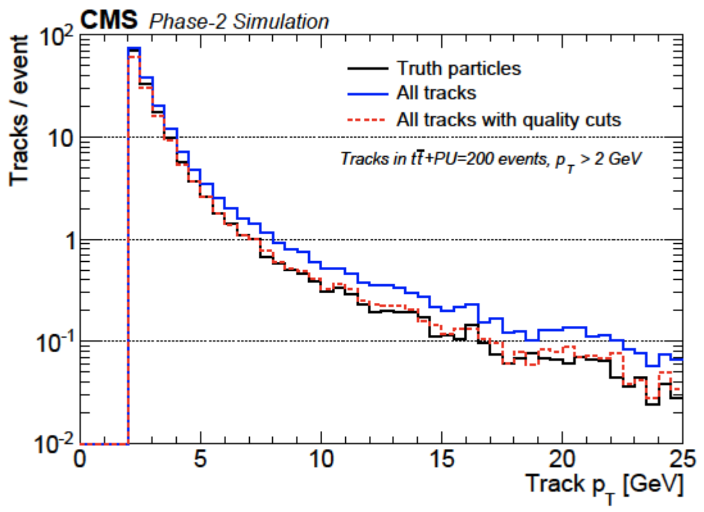

Figure 19. Track rate for $t\bar{t}$ events with an average of 200 pileup interactions. The lines show the truth track rate (solid black line), the rate of all reconstructed tracks above 2 GeV (solid blue line), as well as the rate of tracks after applying a set of quality criteria (dashed red line).

2020 JINST 15 P06024

resolutions, such as a 1 mm (5 mm) z_0 resolution at central (forward) , and a relative p_T resolution ranging between 1–4%. This performance is more than sufficient for how the tracks will be used in the downstream L1 trigger.

Beyond tracklet 2.0, improvements are considered that will enhance the efficiency for displaced tracking as well as reduce the resource needs and latency by combining different processing steps. In particular, we can combine the ProjectionRouter, MatchEngine, and MatchCalculator modules into one combined module and the TrackletEngine and the TrackletCalculator into another module. A combined ProjectionRouter, MatchEngine, and MatchCalculator module, 'MatchProcessor', would reduce the need for storing the projections and candidate matches in memories between the processing steps. At the same, it would reduce the latency by only incurring the processing time of one processing step, but with a slightly longer step latency.

Acknowledgments

The authors would like to thank the Wisconsin CMS group for their support of the CTP7 platform. We are grateful to the CMS Collaboration for use of their detector simulation software. This work was supported by the U.S. National Science Foundation through the grants NSF-PHY-1607096, NSF PHY-1312842, NSF-PHY-1307256, NSF-PHY-1120138 and the U.S. Department of Energy Office of Science DE-SC0011726 and DE-SC0015973.

References

[1] CMS collaboration, *The CMS Experiment at the CERN LHC*, 2008 *JINST* **3** S08004.

[2] F. Zimmermann, *CERN Upgrade Plans for the LHC and its Injectors*, *PoS* **EPS-HEP 2009** (2009) 140.

[3] CMS collaboration, *Technical Proposal for the Phase-II Upgrade of the CMS Detector*, CERN-LHCC-2015-010 (2015).

[4] CMS collaboration, *The Phase-2 Upgrade of the CMS Tracker*, CERN-LHCC-2017-009 (2017).

[5] I. Tomalin et al., *An FPGA based track finder for the L1 trigger of the CMS experiment at the High Luminosity LHC*, 2017 *JINST* **12** P12019.

[6] G. Fedi, *Associative Memory Pattern Matching for the L1 Track Trigger of CMS at the HL-LHC*, *Eur. Phys. J. Conf.* **127** (2016) 7.

[7] IEEE Computer Society, *IEEE 1364-2001 — IEEE standard hardware description language based on the Verilog(R) hardware description language.*, https://standards.ieee.org/standard/1364-2001.html, last accessed on 2019-7-31 (2001).

[8] J. Chaves, *Implementation of FPGA-based level-1 tracking at CMS for the HL-LHC*, 2014 *JINST* **9** C10038.

[9] Xilinx Incorporated, *7 Series FPGAs Overview*, https://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html, last accessed on 2017-03-28.

[10] A. Svetek et al., *The Calorimeter Trigger ProceSSOR Card: the next generation of high speed algorithmic data processing at CMS*, 2016 *JINST* **11** C02011.

[11] CMS collaboration, *The CMS trigger system*, 2017 *JINST* **12** P01020 [arXiv:1609.02366].

2020 JINST 15 P06024

[12] CMS collaboration, *CMS Technical Design Report for the Level-1 Trigger Upgrade*, CERN-LHCC-2013-011 (2013).

[13] GEANT4 collaboration, *Geant4 — a simulation toolkit*, *Nucl. Instrum. Meth. A* **506** (2003) 250.

[14] Boston University, *The AMC13 Project*, http://bucms.bu.edu/twiki/bin/view/BUCMSPublic/HcalDTC, last accessed on 2017-03-28.

[15] Xilinx Incorporated, *Vivado HLS*, https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html, last accessed on 2019-7-31.

2020 JINST 15 P06024