FPGA-based Real-time Charged Particle Trajectory Reconstruction at the Large Hadron Collider

Edward Bartz[†], Jorge Chaves[*], Yuri Gershtein[†], Eva Halkiadakis[†], Michael Hildreth[‡], Savvas Kyriacou[†], Kevin Lannon[‡], Anthony Lefeld[§], Anders Ryd[*], Louise Skinnari[*], Robert Stone[†], Charles Strohman[*], Zhengcheng Tao[*], Brian Winer[§], Peter Wittich[*], Zhiru Zhang[*], and Margaret Zientek[*]

[*]*Cornell University, Ithaca, NY*

[†]*Rutgers University, New Brunswick, NJ*

[‡]*University of Notre Dame, South Bend, IN*

[§] *The Ohio State University, Columbus, OH*

Abstract—**The upgrades of the Compact Muon Solenoid particle physics experiment at CERN's Large Hadron Collider provide a major challenge for the real-time collision data selection. This paper presents a novel approach to pattern recognition and charged particle trajectory reconstruction using an all-FPGA solution. The challenges include a large input data rate of about 20 to 40 Tbps, processing a new batch of input data every 25 ns, each consisting of about 10,000 precise position measurements of particles ('stubs'), perform the pattern recognition on these stubs to find the trajectories, and produce the list of parameters describing these trajectories within 4 μs. A proposed solution to this problem is described, in particular, the implementation of the pattern recognition and particle trajectory determination using an all-FPGA system. The results of an end-to-end demonstrator system based on Xilinx Virtex-7 FPGAs that meets timing and performance requirements are presented.**

1. Introduction

This paper describes results from a demonstration system for a novel implementation of a charge particle trajectory ('track') reconstruction approach based on FPGAs for proton-collider physics experiments. The implementation is intended to be used by the upgraded Compact Muon Solenoid (CMS) experiment [1] at CERN's Large Hadron Collider (LHC) [2]. The upgrade of the LHC [3] will produce more intense collisions, leading to a large increase in the input data rates that the experiments must process. These upgrades will enable searches for undiscovered rare particle physics processes as well as detailed measurements of the properties of the Higgs boson. The LHC collides proton bunches every 25 ns; once upgraded, each of these bunch collisions (an 'event') will consist of an average of 200 proton-proton collisions. Only a small fraction of these collisions are of interest for further study. A fast real-time selection, referred to as the 'trigger', is applied to decide whether a given collision should be saved for further analysis. 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.

To reconstruct the trajectories of charged particles, the CMS experiment includes a tracking detector. Charged particles leave energy deposits ('stubs') when crossing the detector material. These stubs can be linked together to reconstruct the trajectory of the charged particles. The proposed detector layout of the upgraded device is illustrated in Fig. 1.[1] Particles produced at the interaction point, (0,0) in the Figure, travel outwards in a uniform magnetic field parallel to the z axis with a strength of 3.8 T. A charged particle traversing this magnetic field is bent such that its trajectory forms a helix. In the r-ϕ plane, the helix forms a circle and the radius of this circle is proportional to the momentum in this plane, the transverse momentum, or p_{T}, of the particle. In the trigger, we are interested in particles with $p_{\mathrm{T}} > 2$ GeV/c; this corresponds to a radius of curvature greater than 1.75 m. Our challenge lies in linking these stubs to form the trajectories of the particles. In each collision of counter-rotating bunches (every 25 ns), about 10,000 stubs are formed. However, only about 10% of the stubs belong to trajectories of interest, so many stubs need 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 trigger; previously, the amount of data to be processed and the calculational complexity was out of reach of FPGAs.

To summarize, some of the challenges are:

- Absorb approximately 10,000 stubs arriving each 25 ns. The input bandwidth is about 20–40 Tbps.
- Perform pattern recognition to identify the stubs that belong to a given trajectory.
- Fit the stubs to extract optimal trajectory parameters.
- Complete all above steps within 4μs to feed into the decision of whether to retain the event before the on-detector hit buffers are exhausted.

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, and the z axis along the counterclockwise-beam. The azimuthal angle ϕ is measured in the x-y plane.

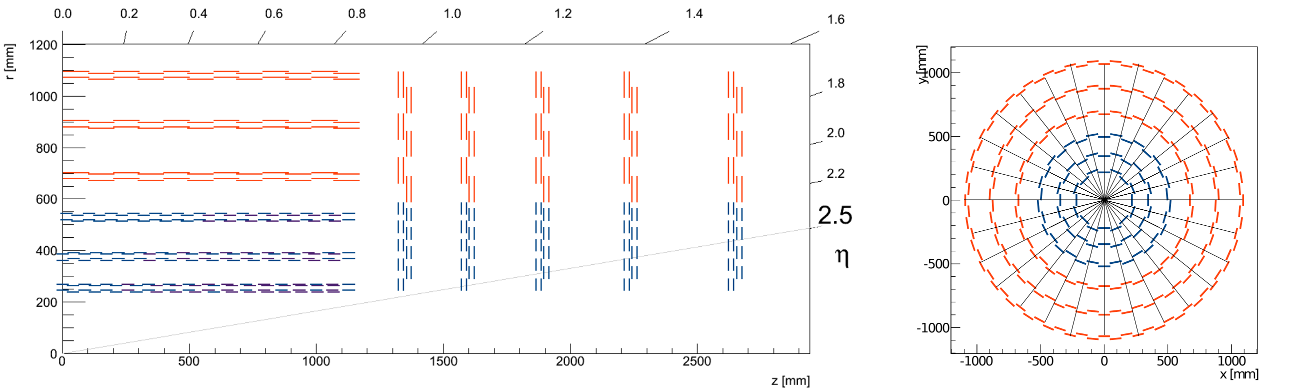

Figure 1. [Left] One quarter of the layout of the proposed CMS charged particle tracking detector upgrade. Collisions take place at (0,0). The layout shown represents one quarter of the full detector. In the central barrel there are six layers at radii from 23 cm to 110 cm. In the forward region, there are five disks at z positions from 130 cm to 270 cm. [Right] The x-y view of the barrel. The detector is divided into 28 regions for track finding.

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 being considered by the collaboration.[2] It is a 'road-search' algorithm, implemented using commercially available FPGA technology. Their 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 results from a demonstrator which implements end-to-end reconstruction, from input stubs to output trajectories, within the available trigger processing time ('latency') and with a reasonable system size.

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. The typical spatial position resolution of the stubs is about 30 μm in ϕ 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. Therefore, the search window (road) around the projected trajectory is small and the probability for finding false matches is low. However, with previous generations of FPGAs, the computational power for implementing this type of tracking algorithm in the trigger was not available. Today, the large number of DSPs and other resources available in FPGAs make such an approach feasible. Earlier real-time, hardware implementations of particle tracking made use of either dedicated ASICs [6], [7] to solve the combinatorics problem in pattern recognition or used binning of the data combined with memory lookup tables [8], [9].

Alternative commercial technologies to FPGAs were considered and rejected for this project. While the large numerical processing capability of GPGPUs suggests a good match at first glance, these technologies are optimized for high throughput, not low latency, and tests have shown they are incompatible with the 4 μs latency requirement [10]. Additionally, GPGPUs and technologies such as many-core processor architectures typically introduce variable latency associated with non-real-time operating systems; such variability is incompatible with the trigger system requirements.

2. The two others are a Hough-transform based approach using FPGAs [4] and an associative memory based approach using a custom ASIC [5].

2. Tracklet Algorithm

The goal of the real-time hardware based track finding is to reconstruct the trajectories of charged particles with $p_{\mathrm{T}} > 2$ GeV/c and to identify the track z_0 position (the z coordinate where the track intercepts the z axis) with about 1 mm precision, similar to the expected average separation of proton-proton collisions in the bunch collisions of the upgraded LHC. The proposed tracklet method forms track seeds, 'tracklets', from pairs of stubs in adjacent layers or disks. The tracklets provide roads where compatible stubs are included to form track candidates. A linearized χ^2 fit determines the final track parameters.

2.1. Algorithm overview

A diagram depicting the serial algorithm can be seen in Fig. 2. Pseudo-code follows.

```
    Form seeds – tracklets
 1: for all stubs in layers $n = 1, 3, 5$ do
 2:     for all stubs in layer $n + 1$ do
 3:         Consider pairs of stubs as tracklet candidate
 4:         if pairs meet $z_0$ and $p_T$ requirements then
 5:             add to list of tracklets
 6:             calculate initial trajectory parameters
 7:         else
 8:             discard tracklet candidate
 9:         end if
10:     end for
11: end for
    Look for matching stubs in other layers
12: for all tracklets do
```

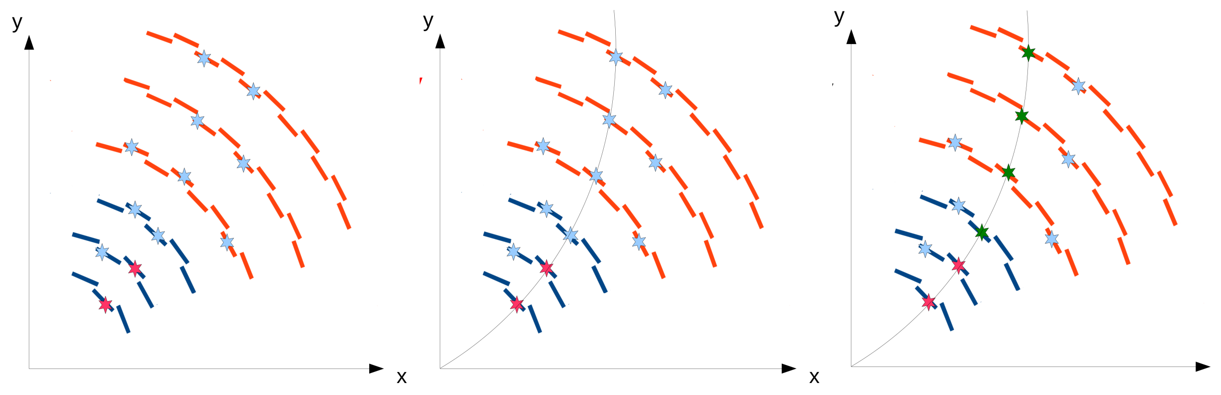

Figure 2. In the first step (left) pairs of stubs (red) are combined to form seeds, or tracklets, for the track finding. Combined with the interaction point (0,0) a helical trajectory for the particle is formed, assuming a uniform magnetic field. This trajectory is projected (middle) to the other layers.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

```
13:    Extrapolate track position to all other layers
14:    for all layers not part of tracklet do
15:       if stub is found close to extrapolated position then
16:          add stub to track candidate
17:       end if
18:    end for
19:    if two or more stubs added to tracklet then
20:       Create candidate track with seed and matches
21:    end if
22: end for
    Calculate final track parameters
23: for all candidate tracks do
24:    Linearized $\chi^2$ fit for final trajectory parameters
25:    if bad $\chi^2$ then
26:       Reject track candidate
27:    end if
28: end for
    Get rid of duplicates
29: for all final tracks do
30:    for all other tracks do
31:       Count number of shared stubs to find duplicates
32:       Remove duplicated tracks
33:    end for
34: end for
```

In the seeding step, seeds are rejected if they are inconsistent with a track with $p_\mathrm{T} > 2$ GeV/c and $|z_0| < 15$ cm. In addition to the example listed above, the seeding also includes pairs between disks 1+2 and 3+4, and between barrel layer 1+disk 1. When the tracklets are projected to other layers and disks to search for matching stubs, the 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. If a matching stub is found, the stub is included in the track candidate and the difference, 'residual', between the projected tracklet position and the stub position is stored. The track fit implementation uses pre-calculated derivatives and the tracklet-stub residuals from the projection step. The linearized χ^2 fit corrects the initial tracklet parameters to give the final track parameters p_T, azimuthal angle, polar angle ϕ_0, z_0 (and optionally d_0, the distance of closest approach to the origin in the x-y plane). Duplicate tracks are removed by comparing tracks in pairs, counting the number of independent and shared stubs.

2.2. Parallelization

The algorithm is parallelized in the following manner. First, the detector is split along azimuth into 28 sections, called "sectors". The number of sectors is chosen such that a track with largest acceptable curvature ($p_\mathrm{T} = 2$ GeV/c) is contained in at most two sectors. A sector processor is a dedicated processing unit and is assigned to each sector and tracks are found in parallel on each sector processor. The tracklet formation is performed within sectors and a small amount of data are duplicated in the even layers to allow tracklet formation locally on a sector processing board and to avoid gaps in any area of the detector coverage. The system of 28 sectors is replicated n times using a round-robin time multiplexing approach. Each system is entirely independent, and therefore, since new data is generated every 25 ns, each independent time multiplexed unit has to process a new event every $n \times 25$ ns. The choice of time multiplexing factor n is driven by a balance of cost, efficiency and needed processing power. 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 an algorithmic inefficiency. For the system in question, $n = 4$–8 have been considered to balance these three factors; the system currently uses $n = 6$; that is, each sector processor receives new data every 150 ns.

Additionally, the algorithm is 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 like N or N^2 if considering all possible combinations. The number of combinations, or combinatorics, is a challenge to the algorithm. The combinatorics in forming tracklets and matching projections to stubs is efficiently reduced by dividing sectors into smaller units in z and ϕ to allow additional parallel processing. These smaller units are referred to as "virtual modules" (VMs). Only a small fraction of virtual module 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.

When implementing this algorithm on an FPGA, we work with fixed-precision math and low-order Taylor expansions of trigonometric functions. We adjust the number of bits kept to ensure adequate precision.

3. Hardware Platform

For the hardware implementation, the design centers around the sector processors. In the final system, each sector processor is foreseen to be an ATCA blade with a Virtex Ultrascale+-class FPGA. The demonstrator system that is discussed in this paper instead uses Virtex-7 FPGAs, currently available on processing boards already developed for other CMS applications. Implementing a full sector in one FPGA on a processing board is out of reach for Virtex-7 class FPGAs, so for the demonstrator system we focus on the implementation of a half-sector.

For the final system, input data (stubs) will be received from upstream over 35 25-Gbps SERDES links. Twelve 25 Gbps links provide nearest-neighbor communication. Output data (reconstructed tracks) will be sent downstream over a single 25 Gbps link to a 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] The heart of the tracklet approach is the processing FPGA. It must have adequate DSP resources (about 2000 DSP48E2 equivalent units), I/O (about 50 SERDES links running at 25 Gbps), 2800 18 Kb block RAMs, approximately 2.5 Mb of distributed (LUT) RAM and adequate LUT resources. These requirements are met by the Xilinx Virtex UltraScale+ family of chips. With 28 sectors and a factor of six time multiplexing, we anticipate that the complete system will consist of 168 blades.

The demonstrator system consists of three ϕ sectors and one time-multiplexing slice. A total of four processing blades are used, one for the central ϕ sector, two for its

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.

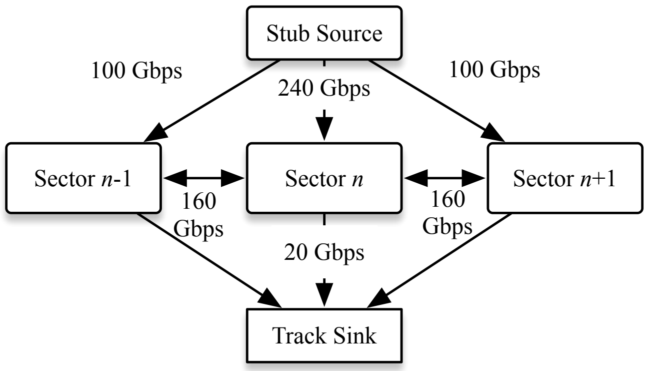

Figure 3. Schematic of the demonstrator system. Three μTCA blades implement three sectors and a fourth blade serves as the source and sink of data. The central sector processor is the actual system under test.

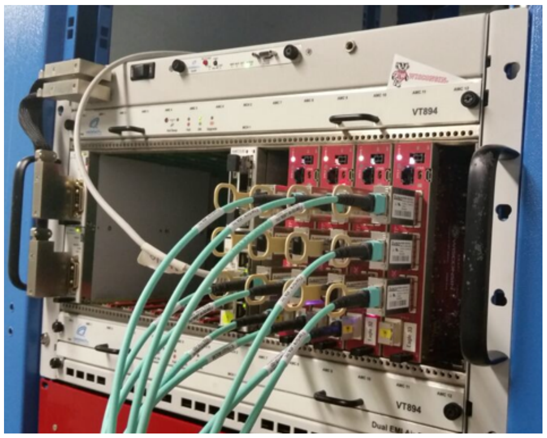

Figure 4. The demonstrator test system. The system is based on the CTP7 μTCA blade, used in the current CMS trigger. The system consists of four CTP7s. Each CTP7 has 63 input and 48 output 10 Gbps optical links.

nearest neighbor sectors, and one blade that acts as a data source (providing input stubs) and a data sink (accepting the final output tracks.) The system configuration is shown in Fig. 3. The demonstrator is fed with simulated data derived from a GEANT-based simulation of the CMS detector [11]. The boards used for the demonstrator system are μTCA boards with a Xilinx Virtex-7 (XC7VX690T-2) FPGA [12] and a Xilinx Zynq-7000 SoC for configuration and outside communication. These so-called CTP7 boards [13] were developed for the current CMS trigger [14]. An AMC13 [15] card provides the central clock distribution. The inter-board communication uses 8b/10b encoding with 10 Gbps link speed. The demonstrator system is shown in Fig. 4.

4. Implementation

The tracklet algorithm is implemented in VERILOG HDL as nine processing steps and two transmission steps [16]. These processing steps are illustrated in Fig. 5. The red boxes are processing modules and the data are stored

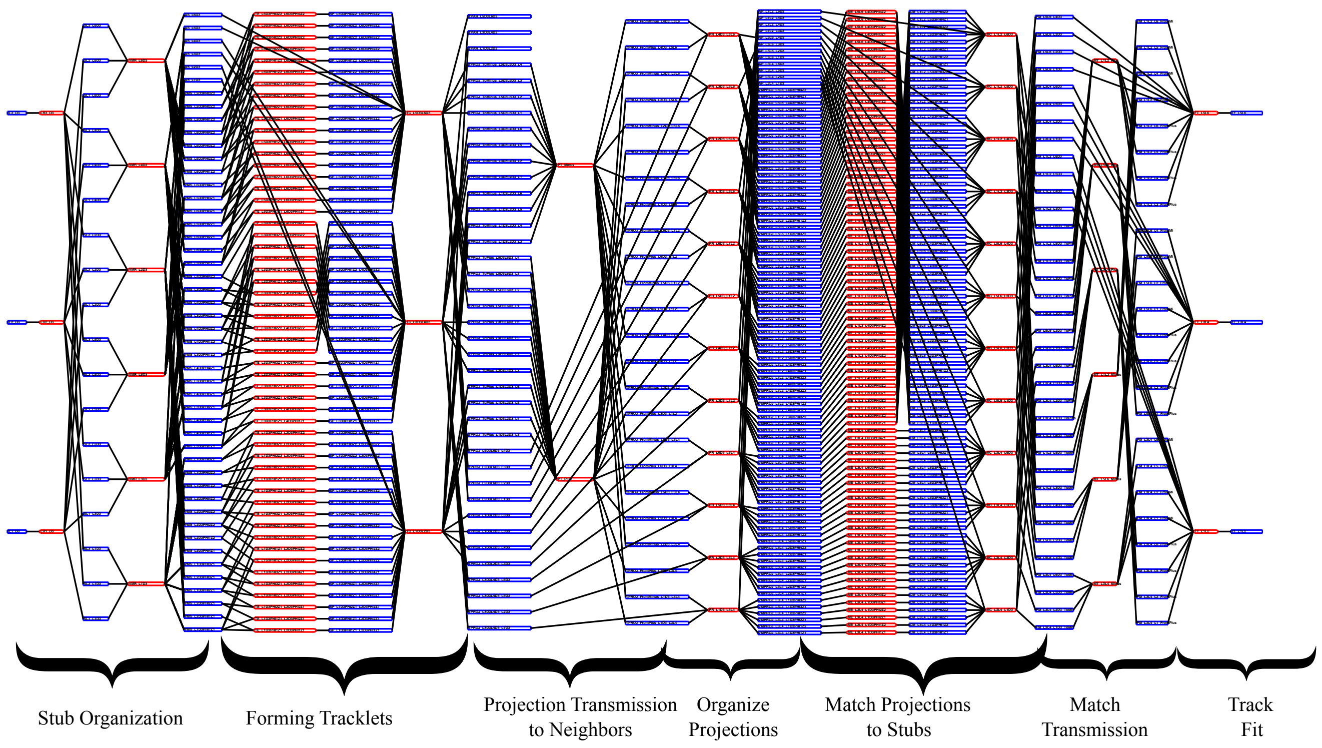

Figure 5. A high-level routing diagram for the tracklet project. As described in the text, the project consists of nine processing steps and two transmission steps indicated in red, and memories, shown in blue, in which the data are stored between the processing steps. The input stubs arrive from the left in the picture and the first two processing modules sort the stubs into the correct virtual modules 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 virtual modules and then the projections are matched to stubs and the matches are then used to perform the track fit. Duplicate removal is not shown.

in memories, blue boxes, between the different processing steps. The implementation of the algorithm in the FPGA takes place in the following processing steps.

- *Stub organization:* (1) Sort the input stubs by their corresponding layer (**LayerRouter**), and (2) into smaller units in z and ϕ, referred to as "virtual modules" (**VM-Router**).
- *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 (virtual modules) in z and ϕ (**ProjectionRouter**).
- *Stub matching:* (7) Match projected tracklets to stubs (**MatchEngine**), and (8) calculate the difference in position between the stubs and projected tracklet (**Match-Calculator**). (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**).

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 sorter 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 Fig. 5 corresponds to 1/4 of the barrel in one sector. A complete project would contain approximately eight times as many instantiations of the same 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. These PYTHON scripts also generate the module connection diagram shown in Fig. 5 and drives 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 are synchronized to a single common clock (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, more than ten events are in the processing pipeline.

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 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 it will only be used later in the chain. This data is 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 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 request 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 SERDES optical links. As discussed above, the charged particles bend in the strong magnetic field of the CMS detector. This bend can cause the tracks of low-momentum particles to curl into neighboring sectors. If a tracklet projects into a neighboring sector, 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 reduces the amount of data duplication globally at the cost of some increase in latency; preliminary studies have shown that at the cost of 40% increase in data duplication we could save approximately 1 μs of latency.

5. Module Examples

To illustrate the method in more detail, we present the functionality of two processing modules. Fig. 6 shows schematically how the virtual module router works. The module receives a start signal every 150 ns for every new event. This VMRouter module reads stubs from three input layer memories. The stub format is illustrated in the Figure and uses 36 bits per stub to encode its geometric position. All stubs are written to the 'AllStubs' memory in the full

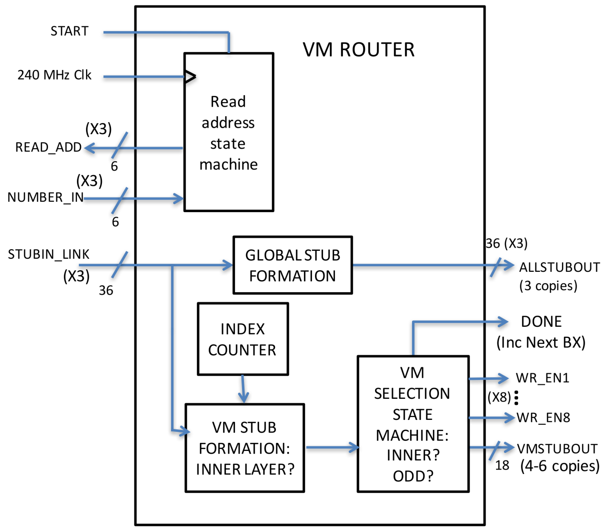

Figure 6. Schematic illustrating the connections of the Virtual Module (VM) Router processing modules. The module reads input stubs from the input memories and routes them to the correct virtual modules based on the stubs coordinates (z and ϕ).

36 bit format. In addition, based on their coordinates (ϕ 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 Fig. 7. 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. Fig. 8 shows the distributions of the number of stub pairs that each tracklet engine 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 150 ns per event and a clock speed of 240 MHz a maximum of 36 input stub pairs can be considered. As can be seen in the Figure, there are cases where there are more than 36 input stubs; the 37th stub 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

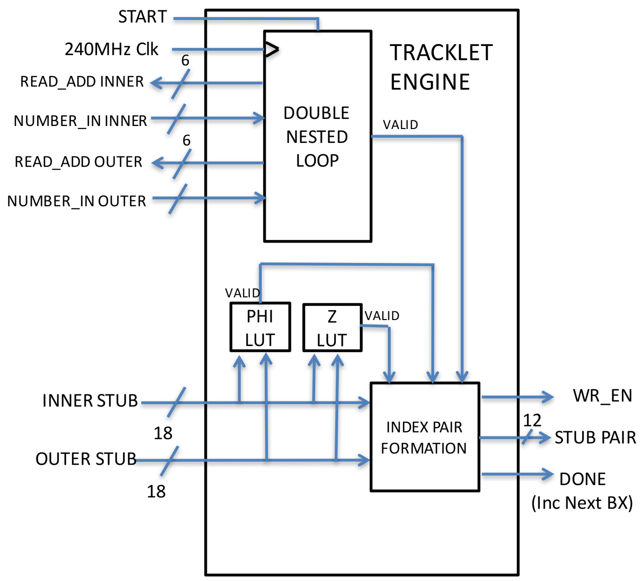

Figure 7. Schematic illustrating the connections of the TrackletEngine processing module. The module reads stubs from two virtual module memories. Two lookup tables are used to check consistency with the momentum and z vertex. If the pair of stubs pass the selection, a stub-pair tracklet candidate is written out.

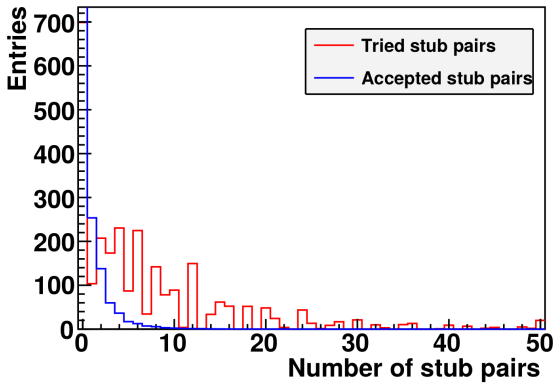

Figure 8. Simulation of the distribution of the number of stub pairs that TrackletEngines seeding in L1+L2 have to process. The red curve shows the number of stub pairs that the module has to consider, while the blue curve shows the number that pass.The non-smooth red curve is due to the fact that the number of tried combinations is a product of two integers. Only 36 stub pairs can be processed in the amount of time available.

observed to be small.

The half-sector project includes seeding in multiple layer and disk combinations (L1+L2, L3+L4, F1+F2, and F3+F4). This project consists of the following processing modules: 12 LayerRouters, 22 VMRouters, 126 TrackletEngines, 8 TrackletCalculators, 22 ProjectionRouters, 156 MatchEngines, 22 MatchCalculators, 4 TrackFits and one PurgeDuplicate. In Table 1, resource usage is summarized from the VERILOG synthesis. The most heavily used resource is BRAMs.

TABLE 1. FPGA RESOURCE UTILIZATION AS REPORTED BY VIVADO FOR THE SECTOR PROJECT. THE TOP LINE SHOWS THE NEEDS FOR THE FINAL PROJECT. THE LINES BELOW SHOW WHAT FRACTION OF THE RESOURCES THIS PROJECT WOULD FILL IN VIRTEX-7 690-T (TOP LINE) AND VIRTEX ULTRASCALE+ FPGAS (OTHERS).

Full sector	LUT Logic 279733	LUT Memory 151191	BRAM 2721.5	DSP 1818
V7 690T	65%	87%	185%	51%
VU3P	32%	81%	85%	80%
VU5P	21%	53%	58%	52%
VU7P	16%	40%	42%	40%
VU9P	11%	27%	28%	27%
VU11P	10%	27%	29%	20%
VU13P	7%	20%	22%	15%

6. Demonstrator Tests

Events are processed through the demonstrator as illustrated in Fig. 3. First, input stubs obtained from simulations are written to the data trigger and control emulator board. On a GO signal, stubs are sent to the three sector processor boards. A new event is sent to each sector board every 150 ns. The events are processed and projections and matches are sent to and received from neighboring boards as in the final system. The final output tracks are received by the track sink board. Systematic studies are 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 is observed in processing single-track events between the emulation, FPGA simulation, and board output. Better than 99.9% agreement is observed with many-track events with high pileup. The demonstrator has a 28-fold azimuthal symmetry, so we test the full $+z$ range by using different input data, corresponding to the different sectors, without any modifications of the demonstrator itself.

7. Demonstrator Tracking System Latency

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 write to the memories for 150 ns before switching to the next event. For some of the steps where data transmission between the neighboring sectors is necessary, latency due to inter-board links is also included. The measured transmission latency is 316.7 ns (76 clock cycles), which includes GTH transceiver TX and RX, data propagation in 15 m optical fibers, channel bonding, and time needed to prepare and pass data from processing modules to the transceivers. The total latency of the algorithm is 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 includes the data transmission latency for receiving stubs from and sending

TABLE 2. DEMONSTRATOR LATENCY MODEL. 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 (CLK)	Step latency (ns)	Link delay (ns)	Step total (ns)
Input link	0.0	1	4.2	316.7	320.8
Layer Router	150.0	1	4.2	-	154.2
VM Router	150.0	4	16.7	-	166.7
Tracklet Engine	150.0	5	20.8	-	170.8
Tracklet Calculation	150.0	43	179.2	-	329.2
Projection Trans.	150.0	13	54.2	316.7	520.8
Projection Router	150.0	5	20.8	-	170.8
Match Engine	150.0	6	25.0	-	175.0
Match Calculator	150.0	16	66.7	-	216.7
Match Trans.	150.0	12	50.0	316.7	516.7
Track Fit	150.0	26	108.3	-	258.3
Duplicate Removal	150.0	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

final tracks back to the data source/sink blade. A summary of the estimated latency is shown in Table 2. 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 has also been measured with a clock counter on the data source/sink blade. The measured latency is 3333 ns, which agrees within three clock cycles (0.4%) with the model.

8. Conclusions

For the upgraded LHC, 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 of the proposed methods for performing 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 Virtex-7 FPGA-based μTCA blade. The final system, which is to be deployed in 2023, 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. (The occupancy within the modules, and therefore the algorithmic inefficiency due to truncation, will therefore not change.) Since more data is coming into the sector processor, 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 UltraScale+ family of FPGAs. None of these changes represent more than an evolution of the demonstrator. The demonstrator system 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.

Acknowledgments

The authors would like to thank the Wisconsin CMS group for their support of the CTP7 platform. This work was supported by the US National Science Foundation through the grants NSF-PHY-1607096, NSF PHY-1312842, NSF-PHY-1307256, NSF-PHY-1120138 and the US Department of Energy Office of Science DE-SC0011726.

References

[1] The CMS Collaboration, “Technical proposal for the Phase-II upgrade of the CMS detector,” Tech. Rep. CERN-LHCC-2015-010, 2015.

[2] Two LHC experiments (CMS and ATLAS) jointly announced the discovery of the Higgs boson in 2012; see Science 338 (2012), 1569.

[3] F. Zimmermann, “CERN upgrade plans for the LHC and its injectors,” *PoS*, vol. EPS-HEP 2009, p. 140, 2009. [Online]. Available: https://cds.cern.ch/record/1211584

[4] G. Hall, “A time-multiplexed track-trigger for the CMS HL-LHC upgrade,” *Nucl. Instr. Meth. A*, vol. 824, p. 292, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900215011304

[5] F. Palla, M. Pesaresi, and A. Ryd, “Track finding in CMS for the level-1 trigger at the HL-LHC,” *J. Inst.*, vol. 11, no. 03, p. C03011, 2016. [Online]. Available: http://stacks.iop.org/1748-0221/11/i=03/a=C03011

[6] J. Adelman *et al.*, “The silicon vertex trigger upgrade at CDF,” *Nucl. Instr. Meth. A*, vol. 572, p. 361, 2007. [Online]. Available: http://dx.doi.org/10.1016/j.nima.2006.10.383

[7] M. Shochet *et al.*, “Fast TracKer (FTK) Technical Design Report,” Tech. Rep. CERN-LHCC-2013-007, 2013. [Online]. Available: https://cds.cern.ch/record/1552953

[8] E. J. Thomson *et al.*, “Online track processor for the CDF upgrade,” *IEEE Trans. Nucl. Sci.*, vol. 49, p. 1063, 2002. [Online]. Available: http://dx.doi.org/10.1109/TNS.2002.1039615

[9] M. Abolins *et al.*, “The Run IIB trigger upgrade for the D0 experiment,” *IEEE Trans. Nucl. Sci.*, vol. 51, p. 340, 2004. [Online]. Available: http://dx.doi.org/10.1109/TNS.2004.828811

[10] S. Amerio *et al.*, “Many-core applications to online track reconstruction,” *JPCS*, vol. 513, p. 012002, 2014. [Online]. Available: http://dx.doi.org/10.1088/1742-6596/513/1/012002

[11] A. Rybin *et al.*, “Geant 4 - a simulation toolkit,” *Nucl. Instr. Meth. A*, vol. 506, no. 3, p. 250, 2003. [Online]. Available: http://dx.doi.org/10.1016/S0168-9002(03)01368-8

[12] Xilinx Incorporated, “7 Series FPGAs Overview,” http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.

[13] A. Svetek *et al.*, “The Calorimeter Trigger Processor Card,” *JINST*, vol. 11, p. C02011, 2016. [Online]. Available: http://stacks.iop.org/1748-0221/11/i=02/a=C02011

[14] The CMS Collaboration, “CMS technical design report for the Level-1 trigger upgrade,” Tech. Rep. CERN-LHCC-2013-011, 2013.

[15] Boston University, “The AMC13 Project,” http://bucms.bu.edu/twiki/bin/view/BUCMSPublic/HcalDTC, last accessed on 2016-07-21.

[16] J. Chaves, “Implementation of FPGA-based level-1 tracking at CMS for the HL-LHC,” *JINST*, vol. 9, p. C10038, 2014. [Online]. Available: http://stacks.iop.org/1748-0221/9/i=10/a=C10038