Deadtime Simulation for the ATLAS Level 1 Central Trigger

Sarah MacHarg

Supervisors: Aimilianos Koulouris, Lorenzo Sanfilippo

CERN Summer Student Program

August 18, 2023

Contents

Background & Motivation 2

Triggering and Bunch Groups 4

Deadtime Logic 5

Deadtime Simulation 6

Project Scope and Objectives 8

Webpage for the CTP Web Monitoring Platform 9

Technical Specifications 9

Frontend Functionality 11

File Storage 11

Deadtime Simulation Functionality Upgrades 14

Multi-group Triggering 14

LHC Filling Information Sources 16

Conclusions and Future Work 17

Acknowledgements 18

Abstract

Total deadtime is an important performance indicator for ATLAS, and an expert on call may find themselves having to make live changes to deadtime parameters; in such a situation, it is important to be able to simulate how a change in parameters would impact system performance before making the decision. Prior to this summer, the L1CT team developed a Python simulation to predict the performance of CTP deadtime configurations under different incoming trigger rates (specifically, Trigger After Prescale - TAP - rates). The primary objectives of this project were: first, to create a new page for the CTP Web Monitoring Platform (and potentially, for use at P1 and the ATLAS Control Room) that would run the simulation and display results; and second, to add additional functionality to the original simulation that more accurately reflects actual LHC behavior. Future paths for improvement include introducing multithreading to improve simulation speed and adding additional functionality beyond the features introduced this summer.

Background & Motivation

The ATLAS Level 1 Central Trigger (L1CT) is an essential part of the ATLAS Trigger and Data Acquisition (TDAQ) system. With 1 bunch crossing every 25 nanoseconds, the LHC produces data at a rate of 40 MHz. Due to time and money limitations, it is not feasible to store all of this data, or even to process it with slower, software-level logic. This problem is addressed in part by the Level 1 trigger, a hardware trigger that slows data influx rates from 40MHz to 100 kHz. [6] The Central Trigger Processor (CTP) limits the number of events that move past initial detection to the next phase of filtration/recording by using:

- Triggering: The CTP only flags particular events to continue to the next steps of data processing. It does so by performing calculations on event measurements to look for "interesting" features, such as muon production. To ensure no events are overlooked, this calculation must necessarily be as fast as the bunch crossing rate.[1] The CTP then issues a "Level 1 Accept" signal (L1A) if trigger conditions are satisfied.

- Deadtime: The CTP has so-called "deadtime" logic that limits the frequency with which L1As can be issued. Depending on the frequency and timing of incoming triggers, an L1A can be "vetoed" by any one of several deadtime algorithms, meaning that the trigger will be "blocked," an L1A will not be issued, and data from the event will not move forward to higher level triggers and long-term storage. Deadtime prevents back-end detector buffers – which store information from events that triggered an L1A – from overflowing, and decreases the rate of data being sent to the next step of the TDAQ pipeline (see Figure 1 on the following page).[3],[6]

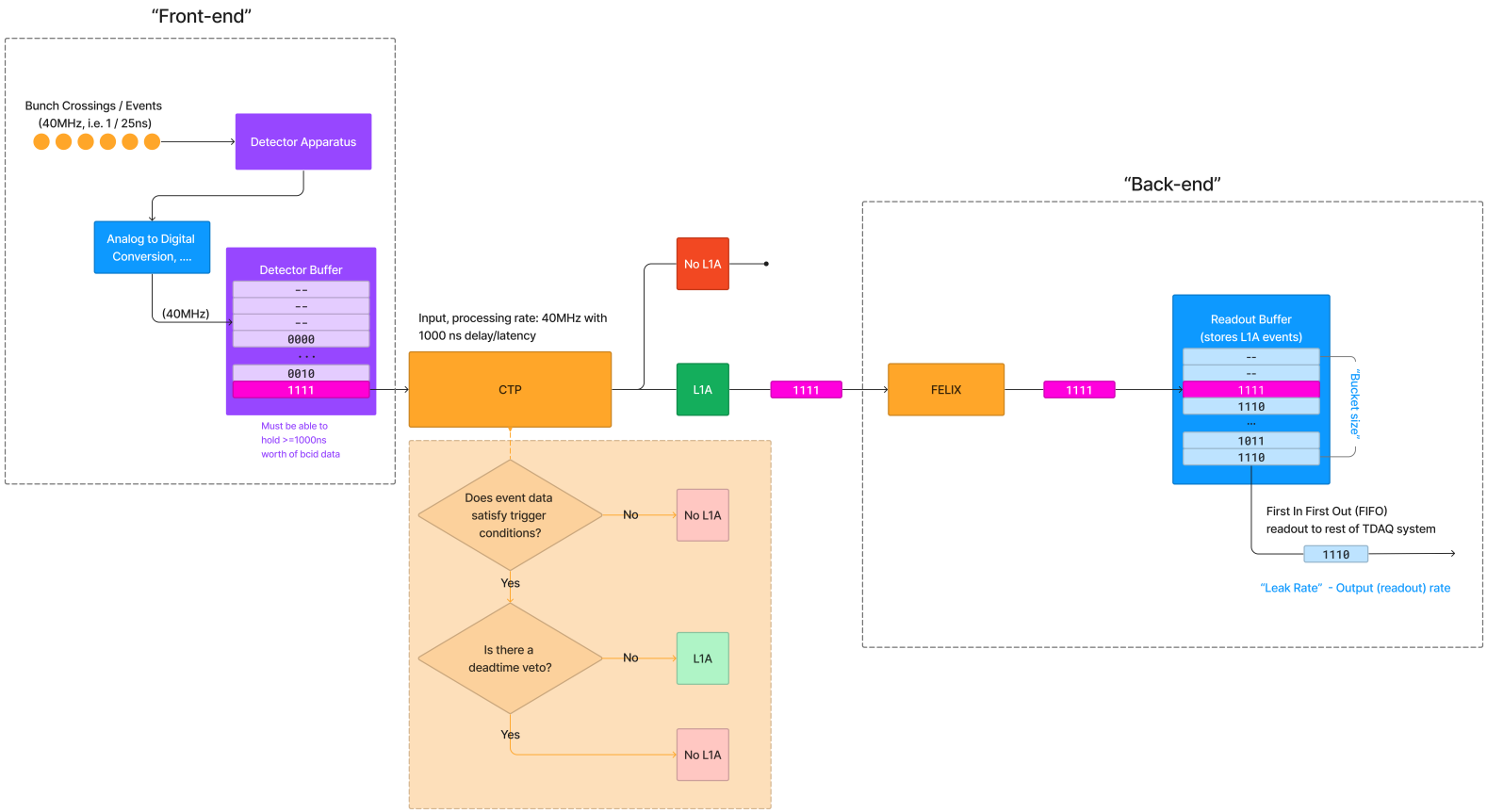

Figure 1: This flowchart provides a high-level schematic of how raw data from the LHC detectors moves from initial storage in front-end detector buffers to back-end buffers and eventually further processing. Incoming data is passed to the CTP. If the CTP issues an L1A, data is then sent to FELIX and written to back-end readout buffers, which take more time to empty as they pass on data from L1A events to later portions of the TDAQ pipeline.

Triggering and Bunch Groups

The issuance of L1As by the CTP depends not only on the signals being sent by the detectors, but also on the configuration of the LHC bunch crossings. Each beam of the LHC contains 3564 bunches of protons. When the beams cross, each "bunch crossing" is assigned an ID number, known as the BCID (bunch crossing ID).[7]

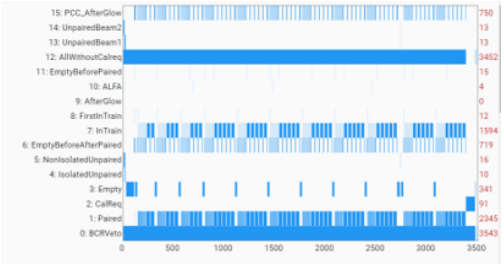

Figure 2: A screenshot of information accessible by Bunch Group Key on the TriggerTool database

When the LHC is in operation, BCIDs are organized into different "bunch groups." A given BCID can belong to multiple bunch groups; in total, there are 16 bunch groups (BG0 - BG15), each with a different name and properties. BG1, for example, represents "paired" bcids – i.e. crossings where both beams are filled with a proton bunch, and so there is the possibility for a collision. [1]

When the CTP decides whether to issue an L1A for a given BCID, it does so by determining whether the crossing event satisfies any of the conditions outlined in the "Trigger Menu."[6] Each item on the trigger menu provides a condition that, if satisfied, triggers an L1A. For example:

```
"L1_EM8VH": {
        "name": "L1_EM8VH",
        "legacy": true,
        "ctpid": 38,
        "definition": "EM8VH[x1]",
        "bunchgroups": [
            "BGRP0",
            "BGRP1"
        ],
        "triggerType": "10000100",
        "partition": 1,
        "monitor": "LF:000|HF:000"
    }
```

Per this menu item, the CTP only issues an L1A if the signal received from the detectors matches certain

[1] Bunch group specifications are briefly described in the bunch grouping TWiki

criteria, *and* the bcid belongs to particular bunch groups.[6] Given this setup, events that are interesting for physics (paired crossings with certain characteristics) may trigger an L1A, but BCIDs from other groups – even those corresponding to non-collision events – can also trigger an L1A depending on the current trigger menu items. As such, they can also contribute to deadtime, and an effective deadtime simulation should properly account for this possibility.

Deadtime Logic

The CTP uses several algorithms concurrently to determine whether to veto an L1A (a "deadtime veto"). If any of these separate processes results in a veto, this is enough for the L1A to be suppressed (OR, not AND).[5],[6] The different forms of deadtime logic can be understood as follows:

- **Simple Deadtime** - after an L1A is issued, suppress additional L1As for a set number of bunch crossings. For example: after issuing an L1A, suppress L1As for the next 4 events.[4]
- **Complex Deadtime** - there are multiple complex deadtime algorithms, which approximate the behavior of the detectors' back-end buffers (helping to prevent overflow):
  - *Leaky Bucket* - a given detector's back-end buffer can hold a certain number of events' worth of data, and reads out those events to the rest of the TDAQ system at a particular rate. Model this as a bucket with a fixed number of spaces (B) and a leak rate (reads out one event's worth of data every R bcids; R is the inverse leak rate). The spaces are filled as L1As are issued, and freed as data is read out.[4][2]
  - *Sliding Window* - take a fixed "window" of a certain number (SW_size) of consecutive bunch crossings (referred to as bcids from here on). Within any window, only allow a certain number of L1As (SW_nb). For example, take a sliding window with SW_size = 3000 and SW_nb = 15. If a

[2]Each of these "leaky buckets" corresponds to an actual detector system within ATLAS – bucket size values, and inverse leak rates, are selected to approximate the behavior of the actual buffers; this way, the system can avoid overloading said buffers by issuing too many L1As.[6] As currently configured, Bucket 0 corresponds to the L1 Calorimeter (L1Calo), Bucket 1 to the Transition Radiation Tracker (TRT), Bucket 2 to the Liquid-Argon Calorimeter (LAr), and Bucket 3 to the Level-1 Topological Processor (L1Topo). There are also default configuration parameters for different LHC run scenarios, such as measuring cosmics or conducting heavy ion experiments.[2]

given bcid has the right conditions to trigger an L1A, the sliding window algorithm checks to see how many L1As were issued across the last 3000 bcids. If this number is ≥ 15, it will suppress the L1A for this event.[4] This also serves to protect buffers from having "trigger bursts" and overflowing. [5]

- **Subdetectors Busy** - separate from the above algorithms, which are calculated within the CTP, the back-end of a detector can issue a "DAQ busy" signal, which will also prompt a deadtime veto.[6]

The amount of deadtime generated by each of these algorithms depends not only on the frequency and spacing of incoming triggers, but also on the configuration parameters – the number S of bcids to wait after an L1A for Simple Deadtime, for example, or the bucket size and inverse leaking rate for one of the buckets. Since the parameters for deadtime logic on the can be reconfigured on the CTP hardware, when trying to optimize ATLAS' performance, it is useful for the L1 expert on call to be able to predict the performance of different parameters given a) a certain frequency of incoming triggers, and b) a particular filling scheme or bunch grouping for the LHC. This was the motivation for the creation of a deadtime simulator.

Deadtime Simulation

There are two primary values of interest when examining deadtime in ATLAS:

- **Total deadtime** - $\frac{\text{total \# of bcids ignored}}{\text{total \# of bcids}}$ - What percentage of all the bcids are ignored due to deadtime? More accurately, for how many bcids (both trigger and non-trigger events) is a deadtime veto issued by the CTP? [3]

- **Physics deadtime** - $\frac{\text{total \# of paired L1As ignored due to deadtime}}{\text{total \# of paired L1As}}$ - What percentage of interesting physics events are ignored due to deadtime? More accurately, what percentage of BG1 paired *collisions* that satisfy a trigger condition and would otherwise result in an L1A happen when the CTP also issues a deadtime veto, and are therefore not sent to the readout system?

[3] It is also useful and interesting to break down deadtime by type: for each deadtime algorithm, and for the "simple" and "complex" categories more broadly, for what percentage of bcids does the algorithm issue a deadtime veto? Since multiple algorithms can issue a veto concurrently, examining only total deadtime loses important information about individual detector performance that can be regained by also tracking deadtime by individual detector/deadtime mode.

The original program[4] simulated deadtime metrics over a given number of LHC orbits using the following approach:

1. Given a bunch grouping,[5] determine which bcids are "paired" (belong to BG1), and thus could produce interesting physics and trigger an L1A signal.

2. For a given frequency for incoming L1A triggers (e.g. 30 kHz), randomly select (according to a Poisson distribution) bcids from BG1 that will trigger an L1A

3. Take what is now a sequence of triggers and non-triggers and run the simple and complex deadtime algorithms to calculate:

   - Total deadtime (% bcids for which `deadtime_veto` was True for one, some, or all of the deadtime algorithms)
   - Simple deadtime (% bcids for which `deadtime_veto` was True for the simple deadtime algorithm)
   - Complex deadtime (% bcids for which `deadtime_veto` was `True` for one, some or all of: four leaky bucket algorithms and sliding window algorithm
   - Deadtime by bucket, or for sliding window individually
   - Physics deadtime (% of bcids that were a trigger/L1A event but were ignored due to deadtime)

4. Repeat for different trigger rates and plot deadtimes as a function of Trigger-After-Veto (TAV) Rate, i.e. frequency of L1A signals that are not vetoed by deadtime.

The original simulation could be run by copying the source files to one's local repository, then running the bash script from the terminal. Any adjustments to parameters (input BGK, trigger frequency, or deadtime logic parameters) had to be done by directly editing the Python script.

[4]Information on the original simulation, created by Antoine Marzin, can be found on the ATLAS TDAQ TWiki

[5]Accessed via Bunch Group Key (BGK), bunch group information is stored in the TriggerTool database, and is one of several means of getting bunch group information for the LHC. For more information, see LHC Filling Information Access

Project Scope and Objectives

Deadtime is an important performance indicator for ATLAS, and an expert on-call may find themselves having to recommend live changes to these parameters; in such a situation, it is important to be able to simulate how a change in parameters would impact system performance before making the change. The existing simulation presented several limitations:

- Ease of access: Operating exclusively from a terminal meant that an on-call expert would need to have access to a computer at all times during their shift in order to be able to run the simulation
- Ease of use/setup: To get started, a user would have to download the content, then fix any package dependency issues on their own, before being able to run the simulation. Changing parameters would also require familiarizing oneself with the code and actually opening up the "black box" of the program.
- TriggerDB BGKs are not the only way in which LHC bunch filling patterns are recorded – the LHC also publishes "fill schemes" that are simpler, but also contain information about which bcids are paired and which are not. Not all "fill schemes" have published BGKs in the TriggerTool database. Prior to this summer, there was no way to run the simulation on those fill schemes, limiting simulations only to published BGKs.
- If a user wanted to use the current LHC fill pattern to run the simulation, the user would need to manually find and retrieve the information from WebIS, ATLAS' internal information system for up-to-date information on LHC and experiment parameters, then manually input the BGK into the Python program.
- The program could only simulate L1A triggers for BCIDs in BG1 (or, with some manual adjustment to the code, for another individual bunch group). However, this is not an accurate representation of the actual behavior of ATLAS, where the trigger menu can be configured to issue L1As for bcids from other bunch groups as well.

These issues motivated the two phases of this summer project.

1. Webpage for the CTP Web Monitoring Platform

Moving the simulation to an online platform allows for centralized maintenance of package dependencies, and for the program to consistently be run by the same back-end user account (simplifying access limitations). At the same time, having an online interface for simulation allows on-call experts to access the simulation from their smartphone or any PC (no special environment configuration or Linux installation required), resulting in added convenience and quicker decision turnaround for the L1 control team.

Technical Specifications

The pre-existing CTP Web Monitoring platform runs `MainPage.php` as an outer frame, then calls other PHP files to fill the content of the page, depending on which page is selected from the menu. This project needed to fit into that existing framework, meaning that it would use HTML, CSS, JavaScript, and PHP. Furthermore, the intent was to run the simulation on a remote machine, not the user's local computer. As such, the project needed to use PHP, not JavaScript, to accept user input, pass it to Python, and retrieve and display the Python output.

This framework introduced certain non-trivial limitations. First, there are limited pipelines of communication between PHP and Python: the command line, and terminal output streams. Second, when PHP runs a terminal command, it does so as an `apache` user. However, because access to the TriggerTool database (and other resources) is restricted, a simple PHP call of a bash script would not suffice, as it would fail to retrieve the information necessary to log into TriggerTool. Furthermore, with the migration of the `pc-adt-04` to the ALMA09 version of Linux, certain dependencies needed to be revised. Though this was fixed with the help of the L1CT, dependencies will likely need to be maintained and perhaps reworked in the future as development continues on `pc-adt-04`, or if the project is used on other servers.

To accommodate these limitations, the project was updated as follows:

The webpage was configured to gather simulation parameters from the user via an HTML form. Upon form submission, the parameters are sent via the POST method to PHP. Communication between PHP and Python is done entirely via command line arguments; as such, the Python script was updated to

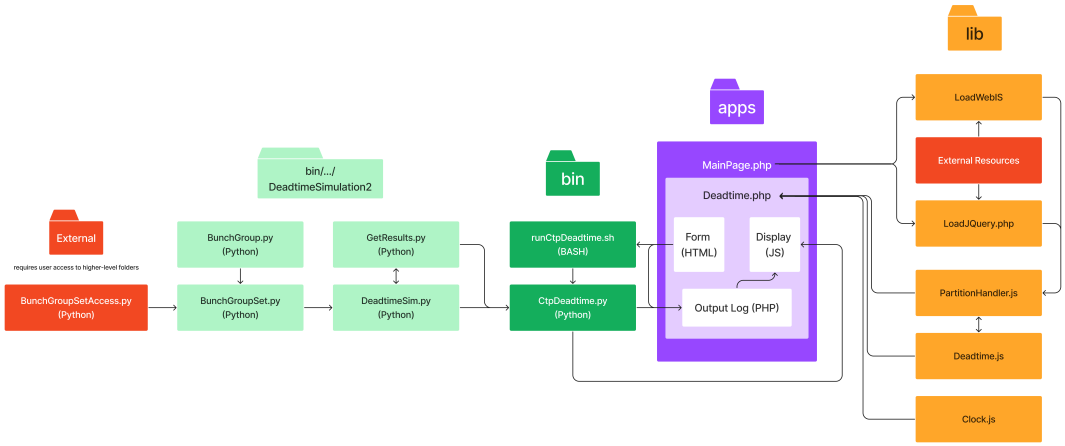

Figure 3: A graphical overview of the final project file structure, code types, and communication pipelines between the front and back end.

accept and parse command line arguments that communicate deadtime parameters, simulation configuration information, bunch group and database names, and file pathways for any user uploads (see LHC Filling Information Access). The PHP then echoes any printed output from Python to the webpage and uses those strings to determine the progress of the simulation. Throughout this process, PHP calls JavaScript to make live updates to the page display, showing a progress bar and displaying JSRoot and PNG versions of the plots once they have been generated. While significant changes were made to the Python program to allow it to interface with the website, changes were designed to also maintain the program's original terminal functionality. A user can still run the program directly from their terminal with both simple and complex commands; these are outlined in the project repository's README.

Certain measures were adopted to ensure continuity in project development after this summer. First, webpage style was, as much as possible, kept consistent with the rest of the CTP Web Monitoring Platform pages for ease of use. Second, and more importantly, documentation for both the original program and the new additions was fleshed out to ensure that an unfamiliar developer could fully understand both communication lines between the front end and back end of the project, as well as the actual functioning of the simulation program.

Finally, security was a nontrivial concern due to the use of a service account, and the use of user input to

generate terminal commands. To help improve security, service account information was stored in a separate, higher-level folder to prevent access. User input was also limited by implementing basic HTML validation rules to limit opportunities for malicious code injection.

Frontend Functionality

All the functionality of the original simulation – in addition to the newly added features(see Upgrades) – has been preserved and made available via a user-friendly interface (partially pictured in Figure 4). Deadtime logic parameters can be changed, and complex deadtime buckets can be enabled or disabled simply by selecting or deselecting checkboxes. The user can adjust the minimum and maximum TAP rates, as well as the step size, allowing them to adjust how many iterations of the simulation are run, and at what level of incoming trigger rate.

As was the case with the original terminal program, there is also a live output log that displays information about the simulation, with some added printouts for newly added features (discussed in Functionality Upgrades). As the simulation runs, a progress bar allows the user to track progress without having to continuously scroll through the output log.

After the simulation is complete, the page displays both an interactive JSRoot module and a saveable and copyable PNG file (as shown in Figure 4). These plots were generated by the original simulation, but have now been made much easier to access and view, enabling a quick analysis of parameters on the go.

File Storage

Because the new webpage allows users to upload JSON files, and makes it much easier for users to generate plots which are then saved to the `pc-adt-04` server (rather than the user's local machine, as in the original simulation program), long-term file storage becomes an issue. At the same time, because multiple users could be using the page at once to run different simulations, files must be stored in a way that prevents "cross-talk" – if a user uploads a JSON fill scheme file, their simulation should run based on that file, not another user's upload. Likewise, the plots that are displayed should correspond to their simulation, not

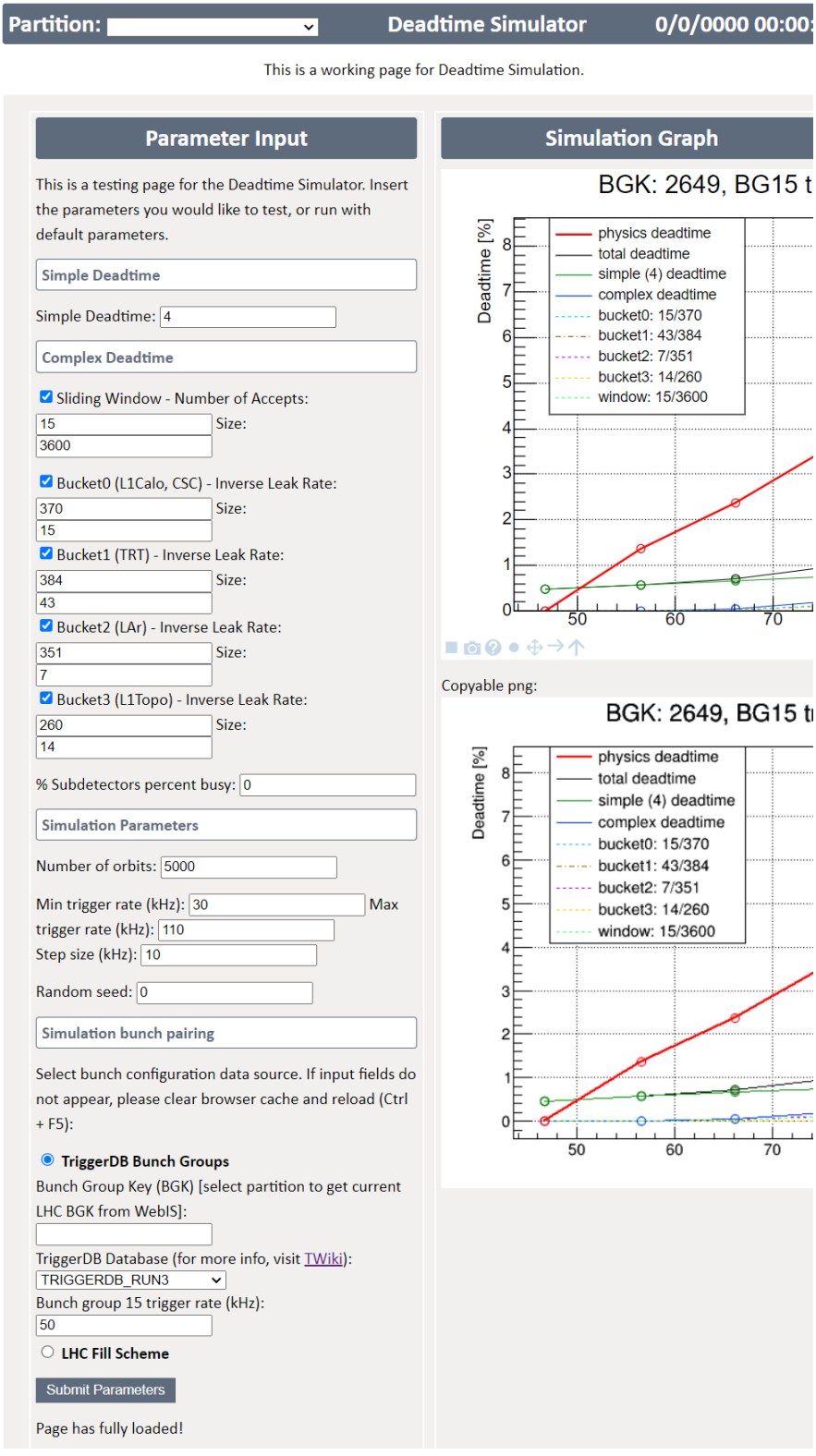

Figure 4: A partial screencapture of the Deadtime webpage, showing the HTML form and part of the plot display

someone else's.

This project addresses these problems by assigning unique filenames to plots, usings not only simulation parameters (as in the original simulation) but also timestamps, giving each filename a unique signature that can be passed between PHP, Python, and JavaScript.

To prevent file buildup, JSON uploads are deleted immediately after being parsed for bcid information. As for plots, any time the webpage is reloaded, old root and png files created by the webpage are deleted. Files created from running the program in the terminal are left untouched by running the webpage.

2. Deadtime Simulation Functionality Upgrades

Multi-group Triggering

The first version of the simulation left room for additional functionality to be added. In the real LHC, "interesting physics" events are not the only events that can trigger an L1A; BG15 events may trigger an L1A and initiate deadtime, but only lost/covered BG1 triggers – not BG15 triggers – should count towards "physics deadtime." Figure 3 demonstrates how these values would be calculated in a simple example where only Sliding Window deadtime logic is in use. After initial development of the CTP Web Monitoring page was complete, this BG15 triggering functionality was implemented by making alterations to the initial program: allowing the user to specify a BG15 triggering rate (which contributes towards the total TAP rate, and eventually TAV rate, specified by the user for the simulation), selecting triggers from the bcids corresponding to BG15, then using those to trigger deadtime, but not counting them towards physics deadtime. Such functionality creates a better simulation of what is actually taking place in the LHC, and allows us to examine the effects of changing BG15 rates on total deadtime.

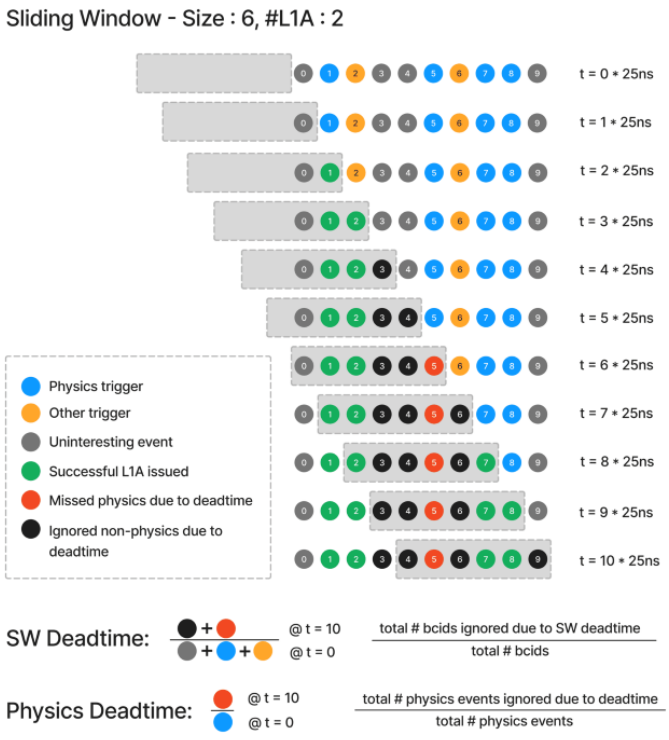

Figure 5: A screenshot of information accessible by BGK on the TriggerTool database

The plots to the right show a simulation for BGK 2649 (which has both BG1 and BG15 bcids), with triggering rates of 0, 10, and 50 kHz for BG15. Examining data points with comparable rates of BG1 triggering, one can see an significant change in behavior due to BG15 triggering. For example, compare:

- The first data point from the first graph (30kHz of BG1 triggering + 0kHz of BG15 triggering = 30kHz TAP rate, ≈30kHz TAV rate)
- The second data point from the second graph (30kHz of BG1 triggering + 10 kHz of BG15 triggering = 40kHz TAP rate, ≈40kHz TAV rate)
- The fourth data point on the third graph (30kHz of BG1 triggering + 50kHz of BG15 triggering = 80kHz TAP rate, ≈75kHz TAV rate)

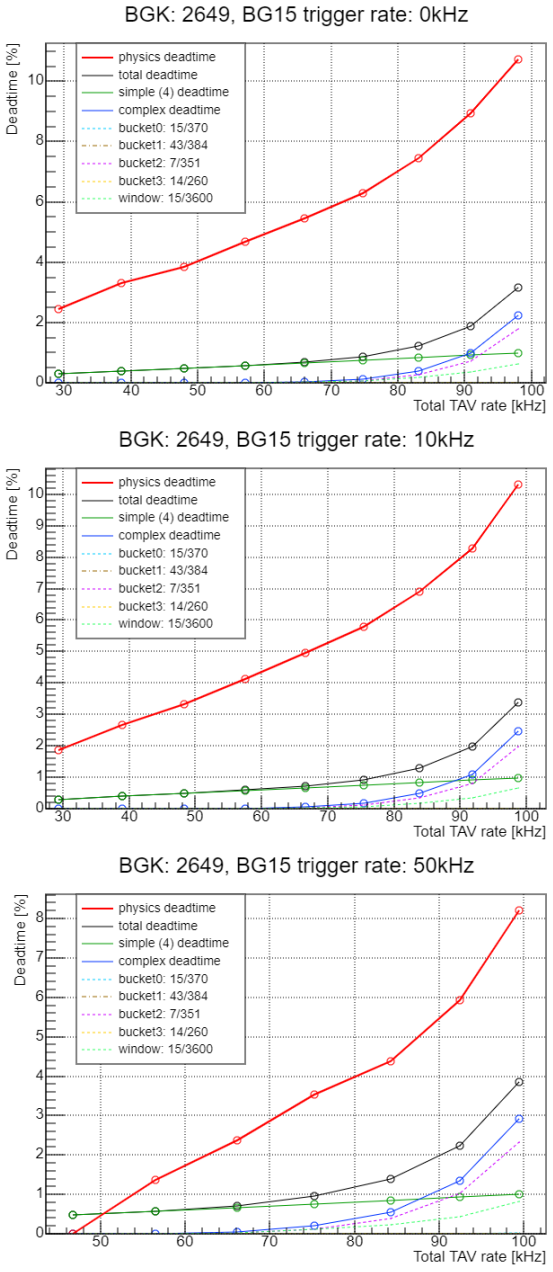

Figure 6: Three plots of deadtime vs. TAV rate produced by the simulation, with BG15 triggering rates of 0, 10, and 50 kHz from top to bottom.

We see that the physics deadtime for the higher BG15 triggering is much higher than that without BG15: 3.5% as compared to ≈2% at lower levels of BG15 triggering. It is therefore valuable to be able to simulate the effects of BG15 to predict *actual* deadtime behavior in ATLAS. Though BG15 triggering was the primary feature requested by the L1CT

team, future work on this tool would do well to allow the user to add a trigger rate for any of the 16 bunch groups.

LHC Filling Information Sources

The original simulation was designed to pull Bunch Group information from the TriggerTool database using a numeric bunch group key ad the alias of one of the databases.[6] As the primary use case for the Deadtime Simulation is to make real-time decisions about deadtime parameters for LHC operation, it makes sense to make it as easy as possible for the user to run the simulation based on current LHC fill status. WebIS is ATLAS' internal information system where current information on the LHC, subdetectors, etc. is published and made remotely accessible. The Deadtime Simulation page would be most useful if, when requested by the user, it could pull the BGK for the current filling of the LHC from WebIS and populate the HTML form with that key.

For project development on the `pc-adt-04` server, the webpage was configured to pull information from WebIS. However, because the partitions available on the `pc-adt-04` WebIS did not have a "bunch group key" feature, and cross-origin limitations built into the CERN system do not allow a `pc-adt-04` application to pull information from the `atlasop` WebIS, the website was developed to pull a different, unrelated integer to demonstrate the viability of live information access. The project README, as well as documentation within the JavaScript, provide sample lines and instructions for how to pull the actual BGK from the ATLAS operations WebIS if/once the project is moved to that server.

Finally, a user may want to test deadtime parameters on LHC fill schemes that have not been published under BGKs – many such schemes are available through the LHC Programme Coordination website as Fill Schemes (see the LPC website visualization of Fill Schemes for more information). On the Fill Scheme Editor Page, users can create fill schemes and download a JSON file that contains simple information about bcid filling in each beam, formatted as follows:

[6] More information available on the TriggerDB GitHub Page

```
{"beam1": [0,0,0,0,1,1,1,1,....,1,1],
 "beam2": [0,0,0,1,1,1,1,0,....,0,0]}
```

The Deadtime Simulation program and webpage were expanded to allow the user to upload a JSON file containing such a fill scheme to the CTP Web Monitoring Platform. The JSON is then read by the Python program, which uses the arrays to determine which bcids are paired and runs the simulation based on this information (treating paired bunch crossings as "BG1"). This provides a substitute for using BGKs to get fill data, which can be useful when a particular fillscheme has not been associated with a BGK or uploaded to the TriggerTool database. However, because of the lack of bunch group information in Fill Scheme JSONs, BG15 triggering capabilities are not available for this mode.

Conclusions and Future Work

Though the webpage is a working application, there are many additional features that could be added to enhance its performance.

First, as mentioned above, it would be helpful to alter the simulation program so that it can accept non-BG1 triggers from any of BG3-15, rather than only BG15 (quick fix), or to be able to take triggers from multiple bunch groups concurrently. This would allow for maximal flexibility and, ideally, allow users to simulate for any of the real-life LHC filling setups encountered.

Second, simulation performance could be improved by introducing multithreading: running each frequency/iteration of the simulation in parallel to reduce total computation time.

Finally, there are many opportunities to use this simulation (or slight adjustments to it) to explore the impact of deadtime on detector performance. For example, by configuring an additional "bucket" to represent an actual backend storage buffer, then running the simulation as normal and sending final L1As to said buffer, one could track buffer overflow to determine the effects of individual deadtime algorithms or adjustments to parameters on buffer performance, creating an additional metric for optimizing deadtime settings beyond simply minimizing physics deadtime. [7]

[7] Credit to Aimilianos Koulouris for this idea and the majority of the proposals for future functionality upgrades

Simulating deadtime performance across different BGKs and BG trigger rates could also provide useful insight for optimizing trigger menu items or bunch group organization to improve data acquisition at the LHC.

Acknowledgements

There are many parties without whom this project would not have been possible.

First, I would like to thank my supervisors, Emil and Lorenzo, for their help in introducing me to CERN and the ATLAS L1CT team, for guiding me through this project, and for answering many, many questions. Additionally, many thanks to Antoine Marzin and Patrick Czodrowski for the many hours they spent troubleshooting access privileges with me.

Second, I would like to thank Theo Alexopoulis, Nikolaos Kanellos, Foteini Kolitsi, and Valerio D'Amico for allowing me to shadow their work at the MicroMegas tests at the GIF facility.

Third, I would like to thank Myron Campbell, Junjie Zhu, Steven Goldfarb, Magdalen Norland, and the University of Michigan REU team for coordinating the US CERN Summer Students and making this experience possible.

I would also like to thank Patricia Burchat, my physics advisor at Stanford, and her colleague Lauren Tompkins for their guidance in making the most of this opportunity. Additionally, I am grateful to my first physics teacher, Brett Guisti, for putting me on the path that has led here.

Finally, I would like to thank my family and the friends I've met at CERN this summer for all of their support. It has been an excellent summer.

Bibliography

[1] ATLAS Level-1 Trigger Group. *Level-1 Trigger Technical Design Report*. Tech. rep. ATLAS TDR-12. CERN, 1998.

[2] Antoine Marzin. *Deadtime Simulation Tool*. URL: `https://twiki.cern.ch/twiki/bin/viewauth/Atlas/LevelOneCentralTriggerDeadtimeSettings#Deadtime_simulation_tool`.

[3] Thilo Pauly. "The ATLAS Level-1 Central Trigger System in operation". In: *Journal of Physics: Conference Series* 219 (2010), pp. 1–8. DOI: `10.1088/1742-6596/219/2/022017`.

[4] R. Spiwoks. "Dead-time Generation in the Level-1 Central Trigger Processor". ATLAS Internal Note.

[5] Mark Stockton. "Error detection, handling and recovery at the High Level Trigger of the ATLAS experiment at the LHC". In: 2016.

[6] Mark Stockton. "The ATLAS Level-1 Central Trigger". In: *Journal of Physics: Conference Series* 331 (2011), pp. 1–6. DOI: `10.1088/1742-6596/331/2/022041`.

[7] B.G. Taylor. "Timing Distribution at the LHC". In: 2002.