A More User-Friendly Global Pattern Format for the LTI

Stepan Mikoyan

Summer 2025

Abstract

The High-Luminosity LHC (HL-LHC) will increase ATLAS trigger and data-acquisition rates to 1 MHz, requiring an updated Local Trigger Interface (LTI) for distributing TTC signals and BUSY feedback. This report presents a Global Pattern Format and associated parser that simplify the creation of LTI patterns while preserving full flexibility. The format integrates TTC, GLOBAL, SYNCUSER, and UPSTREAM controls into a row-based table with explicit timing, trigger, and feedback fields, along with multiplicity operators and automatic gap handling. The parser is validated against firmware specifications and by reproducing a complex Liquid Argon calorimeter test sequence. To support realistic studies, two legacy ALTI pattern generators were adapted: a Python tool for simple dead-time models and a C++ tool implementing the Run-4 sliding-window scheme. Tests show accurate reproduction of expected dead-time behaviour, confirming the framework's suitability for Phase-II LTI development and future inclusion within the low-level menu.

1 Introduction

1.1 HL-LHC upgrade

Starting in 2026, the LHC will undergo its third long shutdown to implement the first steps of the High-Luminocity Large Hadron Collider (HL-LHC) upgrade. Following this major upgrade, the LHC will operate with increased instantaneous luminosity during the LHC Phase 2. For the next Run 4 (2030-2033), the peak instantaneous luminosity will increase to $\mathcal{L} = 5 \times 10^{34}$ cm^{-2} s^{-1}, which is 2.5 times higher than currently in LHC Run 3 (2022-2026). This corresponds to approximately 200 inelastic p-p collisions per bunch crossing, and more than ten times greater integrated luminosity than all three of the previous runs combined (up to 4000 fb^{-1} [1]). In order to utilise the full potential of the improved collider, the ATLAS trigger and data acquisition (TDAQ) system has to process data at a detector readout rate of 1 MHz rather than the current 100 kHz. This increase in rate would cause a substantial increase in dead-time, up to unacceptable levels. For these reasons, together with the limitations present when the hardware was conceived, an update to the TDAQ system is necessary.

1.2 TDAQ System [1]

The overview of the Phase-II TDAQ system is shown in Figure 1. Data from detectors (blue boxes) flows into the upgraded Level-1 (L1) system. In Phase-2 operation, the ATLAS Level-1 system will comprise the L0Calo and L0Muon systems, the Global Trigger, the Central Trigger Processor (CTP), and the Muon to CTP Interface (MUCTPI). The L0Calo and L0Muon modules will collect the information from the on-detector electronics from the ATLAS calorimeters and muon detectors, respectively,

to perform initial event selection and flag certain details that are to be processed more thoroughly by the rest of the system. To complement the L0Muon, the MUCTPI module identifies overlaps in muon candidates, calculates multiplicities, and provides an interface between the various components of the L0Muon system and the rest of the downstream modules.

The next processing step comes in the form of the new Global Trigger module which performs additional offline-like algorithms to allow for better rejections in various scenarios and help identify topological signatures with a variety of four-vector combinations. The Global Trigger utilises the data sent from the L0 subsystems and combines with those objects to refine selections.

After these selection steps, the Central Trigger Processor (CTP) outputs the final Level-0 decision on the event. Figure 2 shows the steps taken to form the L0A signal. Synchronised, aligned, preprocessed, and mapped, in total 1536 trigger input bits are reduced to 1024 bits by the switch matrix, which performs selecting and routing of trigger inputs. Further in the CTP signal pipeline, the Trigger Logic block implementing the Look-up Tables (LUT) and Content-Addressable Memories (CAM) forms 1024 trigger items (which is four times more than the maximum supported number in the current CTP). The formed trigger times then undergo LHC bunch pattern matching and pre-scaling. They are then subject to a veto decision, which represents the BUSY status of ATLAS sub-detectors. Namely, a signal from a given event can be vetoed or masked if any sub-systems send a BUSY signal due to either filled TDAQ buffers or a global veto, and therefore not propagated downstream to the TDAQ chain. Finally, the L0 Accept (L0A) signal is formed through the use of a logical OR of the resulting trigger objects. This signal is transmitted via the Local Trigger Interface (LTI) modules which link the CTP to the FELIX readout boards [2].

The FELIX (FrontEnd LInk eXchange) readout boards are the first step within the Readout part of the DAQ system. Together with the Data Handlers, the dataflow system prepares the data for processing by the Event Filter system by aggregating and compressing it. Finally, if an event has made it all the way down the system and is accepted by the Event Filter, it is saved in permanent storage. The expected final output rate is 10 kHz.

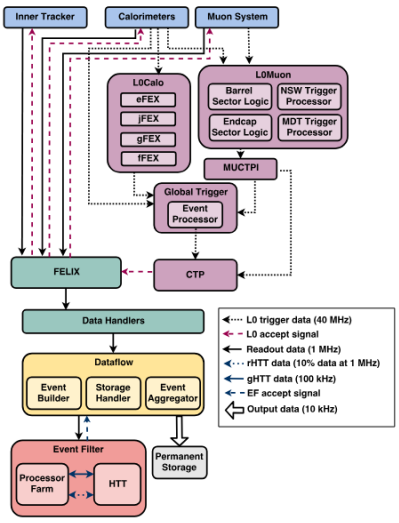


Figure 1: High-level diagram of the proposed TDAQ system for the Phase-II upgrade [1].

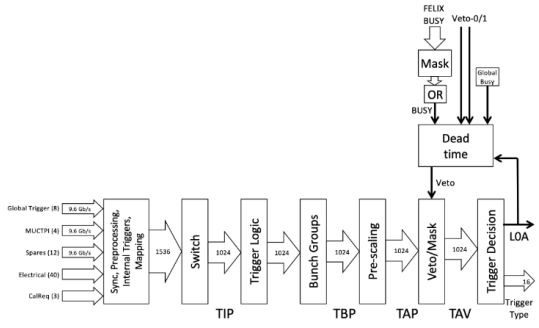


Figure 2: Pipeline for the formation of an L0A signal [3].

1.2.1 The LTI Module

The focus of my summer project is the LTI module to whose development I have contributed in the ATLAS L1 Central Trigger group. The LTI module is replacing the current ALTI (ATLAS Local Trigger Interface) boards, which were introduced for Run 3 during the Phase-1 upgrade. The main functionality of the LTI is to provide an interface between the CTP and the Timing Trigger and Control (TTC) broadcasting network to the front-end readout electronics ofof ATLAS sub-detectors. As well as the distribution of TTC signals downstream, the LTI also sends BUSY signals, which are received from the FELIX systems that collect the BUSY status from sub-systems [2]. To facilitate various situations during both development and physics run conditions, the LTI modules are able to take inputs not only from the CTP (during data-taking), but also from other LTI modules, on-board pattern generators, or detector-specific signals. The LTI modules can be chained together with one acting as a master to the rest. During routine data-taking runs the CTP acts as the master; however, for testing and calibration purposes, the LTI modules can utilise their MiniCTP functionality and command their own slave LTI modules.

The MiniCTP consists of all the trigger evolution steps with the synchronisation, alignment, and selection stages that end with the formation of L0A identical to the CTP [2]. The BUSY signals can be generated internally from the dead-time module. The conceptual layout of the MiniCTPs can be seen in Figure 3. To ensure that the readout modules are not overloaded, a certain dead-time must be applied to the rate of L0A signals. In Run 3 a simple dead-time algorithm, four leaky-bucket algorithms, and one sliding-window algorithm are used. For Run 4, leaky-bucket algorithms are no longer present [4], and the frequency of L0A signals is limited to four acceptances within five bunch crossings (BC).

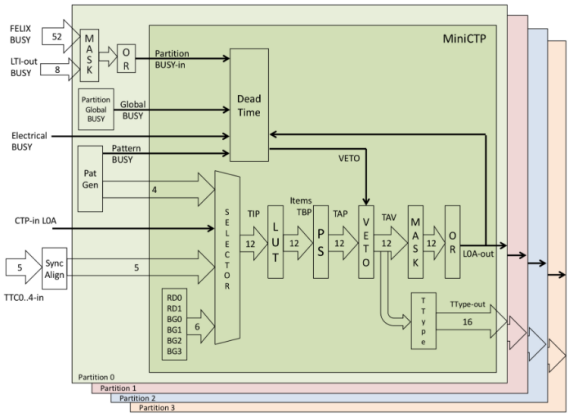


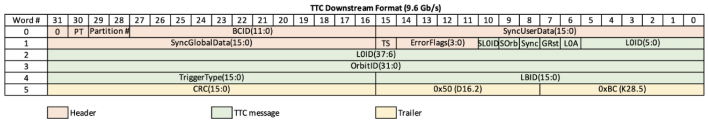
Figure 3: Conceptual diagram of the LTI MiniCTP Trigger Flow [2].

2 LTI Pattern Generators

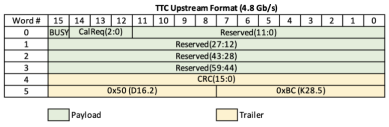
For the MiniCTP to function, the LTI uses a number of pattern generators (PGs) to allow the generation of signals that would normally come from the CTP or one of the LTI inputs. These pattern generators can create pulses for the L0A signal, five TTC-in signals, upstream data such as BUSY and calibration requests, as well as sending synchronous user commands, and other information such as generating a local bunch crossing ID (BCID). The instructions for generating given signals can be written by the user through a pattern loaded on and processed by the pattern generator [2].

2.1 Patterns

At the lowest writable level, these patterns are hex strings associated with various pulses and signals. The formats of the downstream and upstream messages can be seen in Figure 4. However, designing, writing, and implementing patterns written at such a low level is not the most practical and straightforward for non-experts in the LTIs; the L1CT team has previously implemented a method for writing patterns in human-readable words. The process is described in the following.



(a) Downlink message format. In this, the user input pattern controls the boolean values of SL0ID, SOrb, Sync, GRst, and L0A, the TriggerType field, and the synchronous data. The rest of the fields, such as the OrbitID, are filled in automatically by the LTI or external inputs.



(b) Upstream message format. Here the user pattern dictates the contents of the BUSY fields and the CalReq fields.

Figure 4: Downstream (a) and Upstream (b) TTC message formats

- First, the user needs to launch the low-level menu and navigate to the PG section (Figure 5).
- Then select various options, requiring separate instances of the writing menu to edit each of the individual sections of the pattern generator (Figure 6).
- Finally, the user has to manually input each of the command words along with the associated data, as well as having to manually calculate the necessary BCID gaps between signals. For example, the pattern in Figure 7 is designed to send an L0A pulse on even BCID values (Result shown in Figure 8). This method works well if the user has a good understanding of the inner workings of the system and the low-level firmware. However, this cannot be expected from other departments, which are likely to use the PG functionality of the LTI. As mentioned by the Liquid Argon Calorimeter department, their test pattern was difficult to sync between the different PG sections and caused some erroneous results due to the tedious writing process.

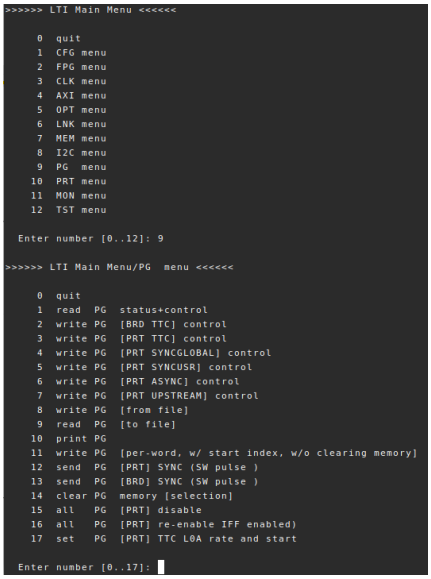


Figure 5: Low Level LTI PG menu.

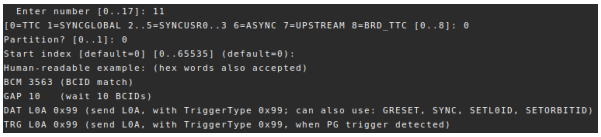


Figure 6: LTI Write Options.

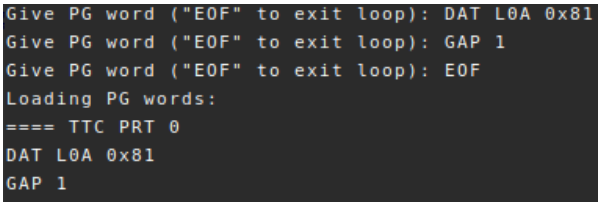
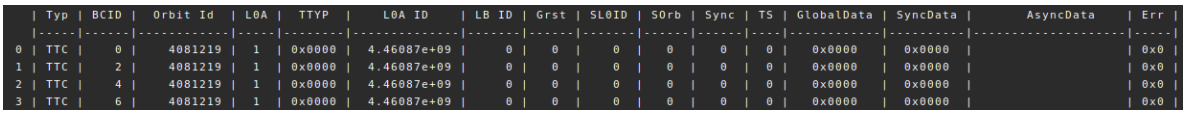


Figure 7: Short Pattern Example



	Typ	BCID	Orbit Id	L0A	TTYP	L0A ID	LB ID	Grst	SL0ID	SOrb	Sync	TS	GlobalData	SyncData	AsyncData	Err
0	TTC	0	4081219	1	0x0000	4.46087e+09	0	0	0	0	0	0	0x0000	0x0000		0x0
1	TTC	2	4081219	1	0x0000	4.46087e+09	0	0	0	0	0	0	0x0000	0x0000		0x0
2	TTC	4	4081219	1	0x0000	4.46087e+09	0	0	0	0	0	0	0x0000	0x0000		0x0
3	TTC	6	4081219	1	0x0000	4.46087e+09	0	0	0	0	0	0	0x0000	0x0000		0x0

Figure 8: Spy Memory of the simple example pattern

3 Global Pattern Format

In order to simplify the process of writing patterns, I was tasked with developing and implementing a separate, more user-friendly, global pattern format. The goal of this was to provide a more intuitive and high-level front end for the end user, which is then converted into a format accepted by the LTI module.

3.1 Requirements

The most crucial requirement for this program was to maintain as close to the same flexibility as the lower-level patterns. This would ensure that most of the possible scope of functionality that could be required during the testing and physics run is achievable through the global pattern. Furthermore, the parsing program must have the built-in functionality to calculate the required amount of GAP to ensure that the signals are sent on the desired BCID and with the correct spacing. In addition to this, the global format must support writing patterns to all of the sections and partitions available on the LTI. Namely, the TTC section, the GLOBAL section, the four SYNCUSER sections, and the UPSTREAM section. The asynchronous data section is omitted by request due to its nature of not being synchronous with the BC. Also, the program must support various program starting modes, outlined in the LTI firmware specifications [4]. These are:

- 'Immediate' start where pulses are sent on the next available BCID.

- 'GAP N' start where the generators commence on the next available BCID after waiting for N (on the N+1 th BCID).
- 'BCM N', where the pattern starts on the next BCID that is equal to N+1.
- 'TRG' start in which the pattern generator waits for a, pre-defined, external (or generated on-board) trigger signal before starting the pattern.

3.2 The Format

After considering the requirements outlined above and going through multiple iterations, the following is the final Global Pattern Format (Table 1). The pattern is presented as a table where each row represents one data word. Each row can contain instructions for all 4 supported sections (TTC, GLOBAL, SYNC, USTREAM) at once if necessary.

Using the BCID and ORBITID columns, the user defines when a given word is sent, with the parsing program calculating the required GAP to match the instruction. Both of the columns accept only integer values with BCID in the range 0–3563 and ORBITID ≥ 0. The BCID column can also accept an empty value, which represents the next BCID*. The ORBITID value is relative to the start of the program; i.e., within the global pattern an ORBITID of 1 does not equate to the ORBITID of 1 within the LTI.

The L0A column allows the user to select whether a Level-0 Accept signal is sent on their chosen BCID. This field accepts values of 0, 1 or empty (equivalent to 0). Most often, this is also accompanied with the TTYP (Trigger TYPe) column, which accepts hexadecimal data in the format 0x.. signifying the type of trigger the L0A represents. During physics running this is calculated as part of the process shown in Figure 2. To complete the TTC DATA word, the TTCIN column accepts a binary string that represents boolean flags for the TTCIN keywords outlined in the LTI specification [2]. These are in the order of SYNC, SETL0ID, SETORBITID, GRESET. If the field is empty, it is taken as "0000".

BCID	ORBITID	L0A	TTYP	GLOBALDATA	SYNCDATA	BUSY	CRQ	TTCIN	MULTIPLICITY

Table 1: Global Pattern Format

The two DATA columns, GLOBALDATA and SYNCDATA, allow the user to send data on the GLOBAL and SYNC sections respectively. Like all the other sections, the sending of the data can be controlled using the first two columns.

The UPSTREAM data is controlled through the BUSY and CRQ columns. The BUSY column, just like L0A, accepts a numerically represented boolean input, which signifies whether to send a busy signal or not. Whereas the CRQ column takes a binary input which each bit representing the status of the three CalReqs. If the CRQ field is not empty and non-zero, the BUSY is automatically returned as 1.

The MULTIPLICITY column repeats the row n times on consecutive BCIDs when n ¿ 0† allowing for a less cluttered global pattern. For example, the input in Table 2 will result in the human-readable output in Table 3.

*This is contingent on the value of ORBITID. If this value does not match the previous (or the next BCID would go over the orbit) then the empty BCID counts as BCID = 0 in the desired orbit.

†The default multiplicity is 1. It will display the word on the BCID given by the BCID column and no more.

BCID	ORBITID	L0A	TTYP	GLOBALDATA	SYNCDATA	BUSY	CRQ	TTCIN	MULTIPLICITY
10	0	1	0x81						3

Table 2: Simple Multiplicity Example

```
==== TTC PRT 0
GAP 9
DAT L0A 0x81
DAT L0A 0x81
DAT L0A 0x81
```

Table 3: Simple Multiplicity Result

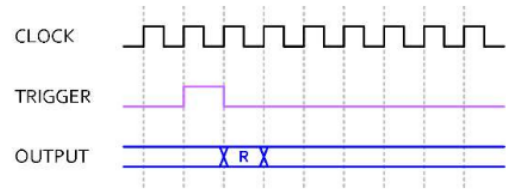
In order to satisfy the first requirement, extra functionality is added to the TTYP, GLOBALDATA, SYNCDATA, and CRQ fields. If any of these fields contain the string "GAP" and the rest of the word is empty then this word will be treated as an extra GAP for the corresponding section. The value for this GAP is still controlled through BCID and ORBITID as before. This addition allows, among other things, to have a custom length skip at the very end of the pattern, this was deemed necessary since the Liquid Argon group utilised this in their testing pattern (written using the low level menu) in order to synchronise pattern outputs.

3.3 Comparisons and Tests

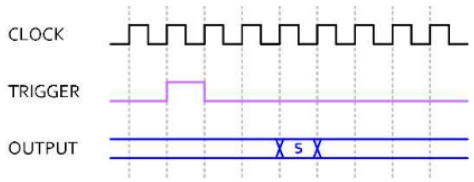
Multiple existing patterns were converted into a global format in order to test its functionality. First, the group of examples from the LTI Firmware Specification [4]. The behaviour is outlined in the specification and shown in Figure 9. The global patterns for these examples were created through the input structure defined for this project, designed to follow an intuitive representation of the original examples. These patterns were then put through the parsing program and executed on the LTI module. Because only hexadecimal data is accepted by the LTI, the values of R, S, W, and Z are set to 0x10. All the examples below illustrate PGs that start to be executed when the TRIGGER signal arrives.

For example 1, Figure 9a, the global pattern and the resulting output (Tables 4 and 5 respectively) produced the output in Figure 10. This matches the expected result, as the TRIGGER signal comes from BRDORBIT and occurs on BCID = 3563*.

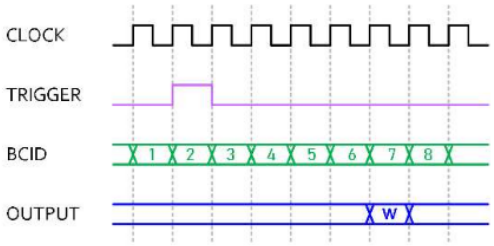
*Since the value of BCID is cyclic, this is directly before BCID = 0.



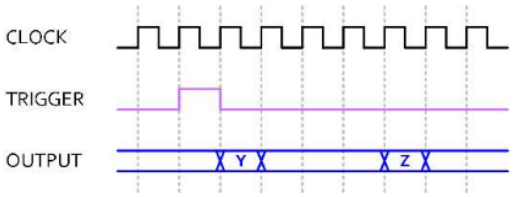
(a) Write a value R immediately after TRIGGER. Requires a single TRG entry.



(b) Write a value S, N ticks after TRIGGER. Requires three entries: TRG, GAP N-1, and DAT.



(c) Write a value W the first time after TRIGGER that the BCID takes the value T. Requires three entries: TRG, BCM T-1, and DAT.



(d) Write value Y immediately after TRIGGER and value Z M ticks after the same TRIGGER. Requires three entries: TRG, GAP M-2, and DAT.

Figure 9: Clock timings for the (a) first, (b) second, (c) third, and (d) fourth example.

BCID	ORBITID	L0A	TTYP	GLOBALDATA	SYNCDATA	BUSY	CRQ	TTCIN	MULTIPLICITY
	0				0x10				

Table 4: Example 1 Global Pattern (Figure 9a).

```
==== SYNC 0 PRT 0
TRG 0x10
```

Table 5: Example 1 Human-Readable Pattern.

	Typ	BCID	Orbit Id	L0A	TTYP	L0A ID	LB ID	Grst	SL0ID	SOrb	Sync	TS	GlobalData	SyncData	AsyncData	Flg
0	TTC	0	937031073				0							0x0010		

Figure 10: Example 1 Spy Memory Result.

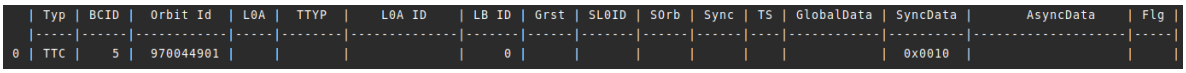
Example 2, Figure 9b, was performed with $N = 4$ and treating the BCID column as a GAP (more on this in Section 5.1). The global pattern (Table 6) results in the human-readable pattern in Table 7. This produces the expected result with the data being sent in BCID = 5 (Figure 11).

BCID	ORBITID	L0A	TTYP	GLOBALDATA	SYNCDATA	BUSY	CRQ	TTCIN	MULTIPLICITY
5	0				0x10				

Table 6: Example 2 Global Pattern (Figure 9b).

```
==== SYNC 0 PRT 0
TRG 0x0
GAP 4
DAT 0x10
```

Table 7: Example 2 Human-Readable Pattern.



	Typ	BCID	Orbit Id	L0A	TTYP	L0A ID	LB ID	Grst	SL0ID	SOrb	Sync	TS	GlobalData	SyncData	AsyncData	Flg
0	TTC	5	970044901				0							0x0010		

Figure 11: Example 2 Spy Memory Result.

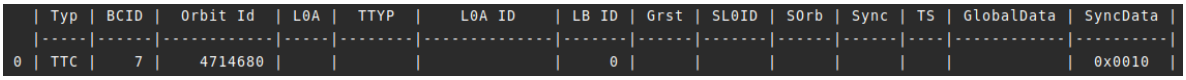
The global pattern for example 3 (Table 8) with T = 7 (set by the value of the BCID column) is parsed to the human-readable input shown in Table 9 as outlined by the specification. When executed on the LTI the result matches the specified timing (Figure 12).

BCID	ORBITID	L0A	TTYP	GLOBALDATA	SYNCDATA	BUSY	CRQ	TTCIN	MULTIPLICITY
7	0				0x10				

Table 8: Example 3 Global Pattern (Figure 9c).

```
==== SYNC 0 PRT 0
TRG 0x0
BCM 6
DAT 0x10
```

Table 9: Example 3 Human-Readable Pattern.



	Typ	BCID	Orbit Id	L0A	TTYP	L0A ID	LB ID	Grst	SL0ID	SOrb	Sync	TS	GlobalData	SyncData
0	TTC	7	4714680				0							0x0010

Figure 12: Example 3 Spy Memory Result.

Finally, the fourth example, Figure 9d, can be recreated with the following global pattern (Table 10). When using the local trigger generator of the LTI, the trigger signal is sent on BCID = 3563 of the previous orbit. When the trigger signal is received from an external source, the BCID of the signal is not preset. Thus having GAP 4 is expected in this case since BCID = 5 is a total of six bunch crossings away from the original trigger signal.

BCID	ORBITID	L0A	TTYP	GLOBALDATA	SYNCDATA	BUSY	CRQ	TTCIN	MULTIPLICITY
	0				0x9				
5	0				0x10				

Table 10: Example 4 Global Pattern (Figure 9d).

```
==== SYNC 0 PRT 0
```
TRG 0x9
GAP 4
DAT 0x10

Table 11: Example 4 Human-Readable Pattern.

	Typ	BCID	Orbit Id	L0A	TTYP	L0A ID	LB ID	Grst	SL0ID	SOrb	Sync	TS	GlobalData	SyncData	AsyncData	Flg
0	TTC	0	982537347				0							0x0009		
1	TTC	5	982537347				0							0x0010		

Figure 13: Example 4 Spy Memory Result.

After confirming the behaviour of simple patterns defined in the specification, the next large test was to convert the existing pattern used for testing the LTI with the Liquid Argon Calorimeter, specifically to "test the different pulsing conditions that are available to [the Liquid Argon electronics]". This pattern was originally designed and written using the human-readable format and, as mentioned above, caused some issues due to the complex nature of it. The original pattern was used to establish its desired properties. The choice of the gap in between and the BCID on which to send the signals were arbitrary as long as the command and the L0A signal were on the same orbits. For this example they were chosen to be: Sending SyncUserData on BCID = 97 every 41 orbits, followed by an L0A signal on BCID = 100 on the same orbits. Since the original pattern sent one fewer instance of SyncUserData than L0A signals, an extra GAP is required at the end to ensure the TTC and SYNC sections are properly synchronised. After these considerations, the global pattern in Table 12 was created and the custom GAP can be added through setting the contents of the data field (TTYP, GLOBALDATA, SYNCDATA, CRQ) of the desired section to 'GAP'. This will make the program use the BCID and ORBITID columns to calculate the size of the desired GAP. Within the pattern file itself, lines starting with `'#'` can be used to annotate the pattern. The resulting functionality of the pattern generator matches the behaviour of the existing pattern. The SPY memory for two cycles of this pattern can be seen in Figure 15.

BCID	ORBITID	L0A	TTYP	GLOBALDATA	SYNCDATA	BUSY	CRQ	TTCIN	MULTIPLICITY
	0		0x0		0x0				
100	0	1	0x80						
100	41	1	0x81						
100	82	1	0x82						
100	123	1	0x83						
100	164	1	0x84						
100	205	1	0x85						
100	246	1	0x86						
100	287	1	0x87						
100	328	1	0x88						
97	0				0x10				
97	41				0x11				
97	82				0x12				
97	123				0x13				
97	164				0x14				
97	205				0x15				
97	246				0x16				
97	287				0x17				
	329				GAP				

Table 12: Liquid Argon Calorimeter Testing Global Pattern.

```
==== TTC PRT 0
TRG 0x0
BCM 99
DAT L0A 0x80
GAP 3463
GAP 35640
GAP 35640
GAP 35640
GAP 35640
GAP 99
DAT L0A 0x81
GAP 3463
GAP 35640
GAP 35640
GAP 35640
GAP 35640
GAP 99
DAT L0A 0x82
GAP 3463
```

(a) Truncated LAr pattern on the TTC section in Human-Readable format.

```
==== SYNC 0 PRT 0
TRG 0x0
BCM 96
DAT 0x10
GAP 3466
GAP 35640
GAP 35640
GAP 35640
GAP 35640
GAP 96
DAT 0x11
GAP 3466
GAP 35640
GAP 35640
GAP 35640
GAP 35640
GAP 96
DAT 0x12
GAP 3466
```

(b) Truncated LAr pattern on the SYNC 0 section in Human-Readable format.

Figure 14: Truncated LAr human-readable pattern.

	Typ	BCID	Orbit Id	L0A	TTYP	L0A ID	LB ID	Grst	SL0ID	SOrb	Sync	TS	GlobalData	SyncData	AsyncData	Flg
0	TTC	97	4100443		0x0080		0							0x0010		
1	TTC	100	4100443	1	0x0080	30895	0									
2	TTC	97	4100484		0x0080		0							0x0011		
3	TTC	100	4100484	1	0x0081	30896	0									
4	TTC	97	4100525		0x0081		0							0x0012		
5	TTC	100	4100525	1	0x0082	30897	0									
6	TTC	97	4100566		0x0082		0							0x0013		
7	TTC	100	4100566	1	0x0083	30898	0									
8	TTC	97	4100607		0x0083		0							0x0014		
9	TTC	100	4100607	1	0x0084	30899	0									
10	TTC	97	4100648		0x0084		0							0x0015		
11	TTC	100	4100648	1	0x0085	30900	0									
12	TTC	97	4100689		0x0085		0							0x0016		
13	TTC	100	4100689	1	0x0086	30901	0									
14	TTC	97	4100730		0x0086		0							0x0017		
15	TTC	100	4100730	1	0x0087	30902	0									
16	TTC	100	4100771	1	0x0088	30903	0									
17	TTC	97	4100773		0x0088		0							0x0010		
18	TTC	100	4100773	1	0x0080	30904	0									
19	TTC	97	4100814		0x0080		0							0x0011		
20	TTC	100	4100814	1	0x0081	30905	0									
21	TTC	97	4100855		0x0081		0							0x0012		
22	TTC	100	4100855	1	0x0082	30906	0									
23	TTC	97	4100896		0x0082		0							0x0013		
24	TTC	100	4100896	1	0x0083	30907	0									
25	TTC	97	4100937		0x0083		0							0x0014		
26	TTC	100	4100937	1	0x0084	30908	0									
27	TTC	97	4100978		0x0084		0							0x0015		
28	TTC	100	4100978	1	0x0085	30909	0									
29	TTC	97	4101019		0x0085		0							0x0016		
30	TTC	100	4101019	1	0x0086	30910	0									
31	TTC	97	4101060		0x0086		0							0x0017		
32	TTC	100	4101060	1	0x0087	30911	0									
33	TTC	100	4101101	1	0x0088	30912	0									

Figure 15: LAr Test Pattern Output.

4 Global Pattern Generators

The first full use of the global pattern format was adapting the ALTI pattern generating files. These programs generate Level-1 Accept signals with random spacing but at a given rate and are designed for testing the module. There are two separate scripts, one with only simple dead-time and no L1A veto (only moving the BCID on which it occurs); the other included the same simple dead-time, as well as the sliding window and leaky bucket complex dead-time algorithms. The latter program also vetoes the L1A signal if it was set to occur during dead-time, rather than just postponing it.

These files were adapted to fit the Phase-II parameters [2] and to output a pattern in the global format.

4.1 GeneratePGRandom.py

The first generator is a Python file which generates a set amount of L0A signals, the amount set by the rate and number of orbits. For example, at 1kHz within 100 orbits, the program generated 9 L0A signals, Table 13. The multiplicity of 4 represents the simple dead-time, mimicking the ALTI.

BCID	ORBITID	L0A	TTYP	GLOBALDATA	SYNCDATA	BUSY	CRQ	TTCIN	MULTIPLICITY
249	6	1	0x81						
	6		0x81						4
1673	8	1	0x81						
	8		0x81						4
1649	14	1	0x81						
	14		0x81						4
3306	19	1	0x81						
	19		0x81						4
962	24	1	0x81						
	24		0x81						4
2563	60	1	0x81						
	60		0x81						4
2289	66	1	0x81						
	66		0x81						4
1007	73	1	0x81						
	73		0x81						4
1982	79	1	0x81						
	79		0x81						4

Table 13: Random L0A distribution at 1 kHz with simple dead-time.

4.2 testGlobalPatternGenerator.cpp

This generator has slightly more variation than the former. Written in C++ it also accounts for complex dead-time which in Phase-II is solely a sliding window algorithm and other simple dead-times such as calibration requests and BCR* veto. Just as its little brother, this program also outputs a selection of L0A signals; however, in this case the program is not concerned with keeping all L0A signals – differing input rate vs. actual rate. Additionally, this program uses a given probability of generating an L0A rather than generating a given amount of randomised BCID as in the first. In the same 1 kHz example, this program generated 7 L0A signals with none being vetoed by the dead time, Table 14.

*BCR is a Bunch Clock Reset – short, asynchronous command which was sent in each orbit in the Phase-1 operation, in order to adjust the phase of the bunch counters in all the receivers.

BCID	ORBITID	L0A	TTYP	GLOBALDATA	SYNCDATA	BUSY	CRQ	TTCIN	MULTIPLICITY
1588	8	1	0x81						
268	27	1	0x82						
2109	27	1	0x83						
3350	27	1	0x84						
2316	43	1	0x85						
1627	71	1	0x86						
960	74	1	0x87						

Table 14: Random L0A distribution at 1 kHz with complex dead-time. Sliding window of four L0A within five BCID [5].

It is instructive to look at unrealistically high rates. For example, setting a 20 kHz rate (greater than the LHC rate of 11.2455 kHz) and sliding window of 2 L0A signals within one orbit (3564 BCID) results in a physics dead-time fraction of 34.9%, Figure 16.

```
*****Configuration object print (detailed)*****
***Complex deadtime***

***Sliding window***
size [NL0A, NWindow]       : 2,3564

*****PatternGenerator configuration****
Output file:              pg_lti_cdt_1kHz.dat
Random pattern rate:      20 kHz

*****PatternGenerator parameters****
Nb of orbits in pattern (set by config):     100
Nb of BCID in pattern (set by orbits):       356397
Nb of L0A expected in pattern (set by rate): 178

*****PatternGenerator: generation****
generation completed, 110 total L0A recorded.

*****PatternGenerator: stat****
Nb of L0A generated in pattern (after deadtime): 110
Nb of L0A generated (before deadtime): 169
simulated L0 trigger rate (aft. deadtime) [kHz]: 12.37 +/- 1.17944
**
CalibReq fraction [%BC]:                         1.96
**
simple deadtime fraction [%BC]:                  0
sliding window deadtime fraction [%BC]:          35.7
**
total   deadtime fraction  [%BC]:                35.7
physics deadtime fraction [%L0A]:                34.9
```

Figure 16: Dead-time fraction at 20 kHz with a two signals in one orbit sliding window limit.

5 Using the Programs

5.1 Global Pattern Parser

To use the parsing program, the user must have a file containing the correct format for a global pattern. Usually this is a .dat type file; however, .txt is also supported. If the program is launched with no extra arguments, the user is prompted to enter three file paths which correspond to the files with global patterns which correspond to the files with global patterns for the supported partitions, namely: BRD*, 0, and 1. All three of these prompts also accept the input of 'None' which signifies to the program that nothing needs to be placed on those partitions. The user is then asked to provide an output path; this does not need to be an existing file (Figure 17).

*Since the BRD partition only contains a TTC section, the allowed patterns for this are more limited.

Once these are entered, the user can select a number of options. First, the option to *Fill Last Orbit*. This boolean option determines whether the remaining BCIDs in the final orbit of the pattern will be explicitly skipped by assigning the 'END' keywords. This feature is most useful when running in cyclic mode, since having available BCIDs may cause synchronisation issues.

```
> ./build/GlobalPatternParser
Input file path for BRD (or None): filepath1.dat
Input file path for 0   (or None): filepath2.dat
Input file path for 1   (or None): filepath3.dat
Output file path: filepath4.dat
```

Figure 17: File Paths Prompt.

Following that is the option to use **TRG** as the starting signal; this is asked separately per partition, and the section is selected through the data fields in the first row (see Table 12). Finally, the user can pick to treat the BCID column of the first word of each section as an IdToMatch, resulting in the **BCM** word type being used. If any of the patterns submitted by the user are designed to parse onto a SYNC section, the user is also prompted for which SYNC section to use (Figure 18).

```
Fill Last Orbit? [0/1]: 1
Wait for TRG on partition 0 [0/1]? 1
Treat BCID as BCM? [0/1]: 1
Which SYNC entry should the 0 partition pattern be on? (0 - 3) 0
```

Figure 18: Various formatting options.

For example, to recreate the Liquid Argon Pattern (Figure 14), the user would select the options shown in Figure 19.

```
> ./build/GlobalPatternParser
Input file path for BRD (or None): None
Input file path for 0   (or None): data/LAr.dat
Input file path for 1   (or None): none
Output file path: data/output.dat
Fill Last Orbit? [0/1]: 1
Wait for TRG on partition 0 [0/1]? 1
Treat BCID as BCM? [0/1]: 1
Which SYNC entry should the 0 partition pattern be on? (0 - 3) 0
COMPLETED PARSING
```

Figure 19: Full LAr User Inputs. Global pattern stored locally in LAr.dat.

The program also supports a number of options. The first one is '-i'. This option (individual mode) omits the headers for any sections that have no output to them thus decluttering the final output. No extra inputs are required when this is used.

The second flag is '-g'. This signals to the program to output long GAPs, which are due to multiple orbits being skipped, as a single number rather than splitting them into 'GAP 35640' and repeated 'GAP 3564' lines. This is purely a personal preference for the user, since both have their advantages and disadvantages when it comes to readability. Once again this requires no extra user inputs.

Finally, the '-e' flag allows the user to edit an existing human-readable file by inputting a global pattern into the desired section. When this mode is in use the user is instead prompted to enter a *path to edit* and the file path to the new global pattern input (or None to empty the selected section). The user also has a choice whether the edited contents will replace the original file, or be placed into a new one.

Then the user has to choose the partition and the section they wish to edit. Once this is selected, the same options as above are presented to the user once again. Once the edit has been made, the user is able to add as many edits to their originally chosen file as they wish (Figure 20).

All three flags can be used together, however during the edit mode, only the changed section will be affected by the user's selections.

```
> ./build/GlobalPatternParser -e
>>> EDIT MODE ACTIVE <<<
Input file path to edit: data/output.dat
Input file path with new pattern (or None): data/pattern_input.dat
Output to same file? [y/n] y
Partition to edit (PRT 0 = 0, PRT 1 = 1, PRT BRD = 2): 0
Section to edit (TTC, GLOBAL, SYNC, USTREAM): ttc
Fill Last Orbit? [0/1]: 1
Wait for TRG [0/1]? 1
Treat BCID as BCM? [0/1]: 1
Make another edit? (y/n) n
```

Figure 20: User Inputs for Edit Mode.

5.2 Global Pattern Generators

The global pattern files generated by the following scripts can be directly inserted into the parsing program discussed above.

5.2.1 GeneratePGRandom.py

This Python file accepts three arguments. A custom output file is set with '-o `file_path`', a custom rate is set with '-r rate', and the number of orbits is selected with '-n `n_orbits`'. These are selected when the program is executed and have default values of `global_gp_lti.dat`, 1kHz, and 100 respectively. The main addition to this file is the use of `n_orbits`; since the pattern generators on the LTI are able to run in cyclic mode, there is no maximum memory size (maximum number of lines) as there was on the ALTI, so this restriction was dropped.

5.2.2 testGlobalPatternGenerator.cpp

The more complex Global Format Pattern Generator supports a configuration file as its execution argument. The format of this file can be seen in Figure 21. Clarifying some of the configuration options: 'seed' is usually set automatically using the time of the executing device, applyBCRVeto and applyCalibReq are boolean flags which veto out certain reserved BCIDs, and loopTTYP increments the trigger type associated with each L0A signal and loops back to a starting value once the maximum, 0xff, is reached. In addition to swapping over to `n_orbits` as with the former file, the leaky bucket algorithms were also removed since they are not present during Phase-II.

```
#*******************************************************************************
# file: testPatternGenerator.cfg
# desc: config file to test PatternGenerator class
# auth: 28-NOV-2019 M. Saimpert
# modified:25-AUG-2025 S. Mikoyan
#*******************************************************************************

# removing an argument set it to its default value
# for info about default values and units see
# PatternGenerator.h
# rate is in kHz
output_file     pg_lti_cdt_1kHz.dat
rate            1
# According to the specifications, L0A signals can occur in consequtive BCIDs with a limit of 4 L0A in 5 BC.
smpl_deadtime       0
seed            -99
orbits          100
applyBCRVeto        1
applyCalibReq       1
loopTTYP        1
#                N_L0A | N_BCID
sliding_window      4     5
```

Figure 21: Configuration File for Complex Dead-Time PG.

6 Final Remarks

In the future, the new format should hopefully simplify the creation and administering of many tests which will be performed as part of the preparation and implementation for the Phase-II upgrade of the ALTAS experiment. The program was designed to be implemented and executed from within the low-level menu present on the LTI. Currently, its functionality has not yet been integrated with the complete firmware. In addition, due to the limited availability of the LTI modules themselves, the integration tests performed by the departments which could utilise my program, lined up on either side of the project time line. Thus there has been a lack on unbiased user feedback, however, many iterations of the format were created with the goal of eliminating any complaints before they arose.

7 Acknowledgements

I would like to sincerely thank Kristina Mihule, Emil Koulouris, and Lorenzo Sanfilippo along with other members of the ATLAS-L1CT group for their continuous help throughout my summer studentship. Without their support this project would not have come to fruition.

References

[1] Imma Riu, Stephanie Majewski, and Francesco Lanni. ATLAS Trigger and Data Acquisition Phase-II Upgrade Technical Design Report, 2018. This version represents the book-like version of the TDR to be circulated to the CB.

[2] N. Ellis, S. Haas, T. Pauly, R. Spiwoks, and P. Vichoudis. Phase-II Local Trigger Interface (LTI) Specification. https://edms.cern.ch/document/2379978. AT2-DA-ES-0005.

[3] N. Ellis, S. Haas, T. Pauly, R. Spiwoks, and P. Vichoudis. Phase-II Central Trigger Processor (CTP) Requirements and Specification. https://edms.cern.ch/document/2787779/1. AT2-DA-ES-0012.

[4] Level-1 Central Trigger Group Internal Document. Phase-II Local Trigger Interface (LTI) Firmware Specification. ATLAS internal communication, 19-Mar-2025. Unpublished.

[5] H. Chen, W. Panduro Vazquez, and T. Pauly. ATLAS Trigger DAQ Interfaces with Detector Front-End Systems: Requirement Document for HL-LHC. https://edms.cern.ch/document/1563801/2. AT2-DI-ES-0002.