

УНИВЕРЗИТЕТ У БЕОГРАДУ  
ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ



**РАЗВОЈ И ТЕСТИРАЊЕ ALTI МОДУЛА  
ЗА ВРЕМЕНСКУ СИНХРОНИЗАЦИЈУ  
У ОКВИРУ ATLAS ЕКСПЕРИМЕНТА У CERN-У**

Мастер рад

Ментор:  
доц. др Јелена Поповић-Божовић

Кандидат:  
Предраг Кузмановић  
3209/2016

Београд, Август 2018.

CERN-THESIS-2018-335  
21/09/2018



UNIVERSITY OF BELGRADE  
SCHOOL OF ELECTRICAL ENGINEERING



**DEVELOPMENT AND TESTING OF THE ALTI MODULE  
FOR TIMING AND SYNCHRONIZATION  
IN THE ATLAS EXPERIMENT AT CERN**

Master thesis

Mentor:  
asst. prof. dr Jelena Popović-Božović

Candidate:  
Predrag Kuzmanović  
3209/2016

Belgrade, August 2018.

## САЖЕТАК

Овај документ описује допринос аутора у развоју и тестирању *ATLAS Local Trigger Interface (ALTI)* модула. ALTI је нови модул дизајниран за ATLAS експеримент у CERN-у, и део је система за временску синхронизацију, такозваног *Timing, Trigger and Control (TTC)* система. Централна функционалност ALTI модула је серијски трансмисиони протокол којим се дистрибуирају тригери и поруке ка сваком од субдетектора у оквиру ATLAS експеримента путем оптичких влакана. Временска синхронизација тригера и порука је од кључног значаја за исправну аквизицију података о честицама добијеним након судара протонских снопова. ALTI је 6U VME64x модул који интегрише функционалности четири постојећа модула која се тренутно користе у експерименту: LTP, LTPI, TTCv1 и TTCex. ALTI модул обједињује функционалности ова четири модула у један, али их и унапређује, што је последица већег логичког капацитета. Модул ће бити постављен у експеримент током дуготрајног искључења Великог Хадронског Судараца честица (*Large Hadron Collider, LHC*) у 2019. години.

ALTI је систем реализован на две штампане плоче, матичној и мезанин плочи. Сва контролна логика је имплементирана је у оквиру фирмвера на *Xilinx*-овом *Artix-7* FPGA чипу, који се налази на матичној плочи. Одређене делове фирмвера имплементирао је и аутор, и за то је коришћен језик за опис хардвера *Verilog* и алат *Xilinx Vivado*.

Контролни *low-level* софтвер за ALTI модул на језику C++ извршава се на "рачунару на једној плочи" базираном на *Intel* процесору. Овај рачунар покреће *Scientific Linux* оперативни систем. Развијена је софтверска библиотека која омогућава приступ до свих делова хардвера и фирмвера, тј. омогућава потпуну конфигурацију ALTI модула. Поред *low-level* софтвера, развијено је неколико тест програма и скрипти за наменско тестирање појединачних функционалности. Такође, у програмском језику *Python* развијен је програм за свеобухватно тестирање прототипа ALTI модула. Овај програм се користи за тестирање свих могућих путања различитих сигнала кроз модул. Помоћу овог програма пронађене су грешке на неколико модула, повезане са монтажом штампаних плоча и лоше залемљеним компонентама.

Читав хардвер на прототипима ALTI модула је тестиран уз помоћ контролног софтвера, као и мерних инструмената попут осцилоскопа и анализатора спектра. То се такође односи и на тестирање функционалности имплементираних у оквиру фирмвера FPGA. Ова тестирања хардвера и функционалности помогла су да се пронађу у грешке у дизајну, као што су: обрнути поларитет једног диференцијалног пара, погрешан напон напајања за чипове компаратора, итд. Све ове грешке су исправљене у другој верзији штампаних плоча које ће се произвести за следећу верзију ALTI прототипа.

Поред тестирања харвера и функционалности, урађена су и мерења перформанси ALTI модула. Као најважнији параметар, мерено је кашњење тј. латенца тригер сигнала од електричног сигнала на улазу до појаве тригера у оптичком сигналу на излазу. Резултати показују да систем базиран на ALTI модулу може да постигне једнаку латенцу као и систем базиран на постојећим модулима.

Други битан параметар перформанси представља количина цитера у излазном сигналу који се даље оптички преноси до субдетектора, јер велика количина цитера може нарушити исправно декодовање порука и тригера од стране пријемника. Тестови су показали виши ниво цитера у систему базираном на ALTI модулу у односу на постојећи систем. Међутим, тестирања на пријемницима су показала да је дековање успешно у оба случаја.

## **ABSTRACT**

This paper describes the author's contribution in the development and testing of the ATLAS Local Trigger Interface (ALTI) module. The ALTI is a new module designed for the ATLAS experiment at CERN, a part of the Timing, Trigger and Control (TTC) system. It is a 6U VME64x module which integrates the functionalities of four existing modules currently used in the experiment: LTP, LTPI, TTCvi and TTCex. The module will be deployed during the long shutdown LS2 of the Large Hadron Collider (LHC) in 2019.

## ACKNOWLEDGEMENTS

I would like to express my gratitude towards my all of my colleagues at CERN that I have collaborated with during my stay as a technical student in this organization. Without them, it would not have been possible to contribute to the ALTI project and deliver this thesis.

First of all, I would like to thank my supervisor, Ralf Spiwoks. He got me introduced to the ALTI project and helped me understand the basic principles behind the ALTAS experiment and the LHC in general. With his invaluable knowledge and years of experience as a physicist and software developer, he has helped me get acquainted with the ATLAS TDAQ and L1CT software developing process. His suggestions were always very helpful and led to continual improvement of ALTI low-level software. A huge number of software packages that he has contributed to in CERN have served as a basis and a model for the ALTI low-level software that I have been developing. Ralf gave me the support for writing this thesis and has also helped me in reviewing it and gave me a lot of helpful advices.

I would also like to thank Vladimir Ryzhov, who is probably the colleague I have collaborated with the most. He is the designer of the ALTI module and has also written the FPGA firmware for the module. I have helped him evaluate the ALTI firmware and he has used my feedback in order to constantly improve it. I have learned a lot from him about FPGA design and electronics in general.

Stefan Haas is another colleague of mine that I am thankful to. As our coordinator, he was always able to clearly put the most important goals in front of us and help us in the organization. He has helped me immensely in the numerous laboratory tests that have been performed on ALTI modules. He has also taught me to use Xilinx Vivado and some other EDA tools.

My thanks also go to Thilo Pauly, the leader of the ATLAS Level-1 Central Trigger team. He always gave us the feedback from the sub-detector people on the ALTI features they would like us to implement. He also helped spread the word about the ALTI among them and introduced them to the possibility of transitioning to the new system. Therefore, if and when the ALTI suppresses and replaces the legacy TTC system, it will be primarily thanks to him.

Many thanks also go the other L1CT colleagues, especially Antoine Marzin, who has been developing the run control application for the ALTI based on the low-level API. His feedback was invaluable in order to improve the low-level ALTI software. By running various laboratory tests on ALTI on his own, he helped us improve both software and firmware of the ALTI module.

Finally, I would like to thank the colleagues from the EP-ESE-BE section that I was associated with during my stay at CERN. They helped us by borrowing the laboratory equipment in order to do the ALTI performance measurements. Besides that, their easy-going and friendly attitude certainly helped me feel comfortably in my working environment.

# TABLE OF CONTENTS

|   |           |
|---|-----------|
| <b>TABLE OF CONTENTS .....</b>                                  | <b>I</b>  |
| <b>1. INTRODUCTION.....</b>                                     | <b>1</b>  |
| <b>2. OVERVIEW OF THE CURRENT SYSTEM .....</b>                  | <b>3</b>  |
| 2.1. ATLAS EXPERIMENT .....                                     | 3         |
| 2.2. TTC SYSTEM .....   | 3         |
| 2.2.1. <i>TTC signals</i> .....                                 | 6         |
| 2.3. LEGACY TTC MODULES .....                                   | 6         |
| 2.3.1. <i>Local Trigger Processor (LTP)</i> .....               | 7         |
| 2.3.2. <i>Local Trigger Processor Interface (LTPI)</i> .....    | 8         |
| 2.3.3. <i>TTC VMEbus Interface (TTCvi)</i> .....                | 8         |
| 2.3.4. <i>TTC Encoder/Transmitter (TTCex)</i> .....             | 10        |
| 2.4. VMEBUS .....   | 10        |
| <b>3. ALTI HARDWARE SPECIFICATION .....</b>                     | <b>11</b> |
| 3.1. INTERFACES.....  | 11        |
| 3.2. ARCHITECTURE .....   | 13        |
| 3.2.1. <i>Cross-point switches</i> .....                        | 15        |
| 3.2.2. <i>Clock distribution</i> .....                          | 16        |
| 3.2.3. <i>TTC signals multiplexing</i> .....                    | 17        |
| 3.2.4. <i>Cable equalizers</i> .....                            | 18        |
| 3.2.5. <i>Memories</i> .....                                    | 18        |
| 3.2.6. <i>Optical transmitter and receiver modules</i> .....    | 19        |
| 3.2.7. <i>Clock and data recovery from the TTC stream</i> ..... | 19        |
| 3.2.8. <i>Power supply</i> .....                                | 19        |
| 3.2.9. <i>Hardware monitoring</i> .....                         | 19        |
| 3.2.10. <i>I2C network</i> .....                                | 19        |
| <b>4. ALTI FUNCTIONALITY AND FIRMWARE.....</b>                  | <b>21</b> |
| 4.1. CLOCKING .....   | 23        |
| 4.2. INPUT SIGNAL SYNCHRONIZATION .....                         | 23        |
| 4.3. PATTERN GENERATION .....                                   | 23        |
| 4.4. SNAPSHOT TAKING .....                                      | 24        |
| 4.5. TTC ENCODER.....   | 24        |
| 4.6. TTC DECODER.....   | 25        |
| 4.7. I2C MASTER CORE .....                                      | 25        |
| 4.8. 1-WIRE MASTER CORE.....                                    | 25        |
| 4.9. BUSY AND CALIBRATION REQUEST ROUTING .....                 | 25        |
| <b>5. ALTI SOFTWARE .....</b>                                   | <b>28</b> |
| 5.1. ATLAS TDAQ .....   | 28        |
| 5.2. LOW-LEVEL API.....   | 29        |
| 5.3. MENU PROGRAM .....   | 32        |
| 5.4. CONFIGURATION OBJECT .....                                 | 32        |
| 5.5. TEST PROGRAMS.....   | 34        |
| 5.5.1. <i>testAltiVME</i> .....                                 | 34        |
| 5.5.2. <i>testAltiInitial</i> .....                             | 35        |
| 5.5.3. <i>testAltiQuickBoot</i> .....                           | 37        |
| 5.5.4. <i>testAltiCapture</i> .....                             | 38        |

|           |  |           |
|-----------|--|-----------|
| 5.5.5.    | <i>testAltiTtc</i> .....                       | 40        |
| 5.5.6.    | <i>testAltiSync</i> .....                      | 43        |
| <b>6.</b> | <b>MODULE TESTING</b> .....                    | <b>46</b> |
| 6.1.      | LABORATORY TESTS.....                          | 46        |
| 6.2.      | AUTOMATED CONNECTION TEST.....                 | 47        |
| <b>7.</b> | <b>PERFORMANCE MEASUREMENTS</b> .....          | <b>51</b> |
| 7.1.      | LATENCY OF ELECTRICAL TTC SIGNALS.....         | 51        |
| 7.2.      | LEVEL-1 ACCEPT LATENCY: LAR DAISY CHAIN.....   | 54        |
| 7.3.      | TTC STREAM AND RECOVERED CLOCK JITTER.....     | 57        |
| 7.3.1.    | <i>Oscilloscope measurements</i> .....         | 57        |
| 7.3.2.    | <i>Phase noise analyzer measurements</i> ..... | 59        |
| <b>8.</b> | <b>CONCLUSION</b> .....                        | <b>63</b> |
|           | <b>BIBLIOGRAPHY</b> .....                      | <b>64</b> |
|           | <b>LIST OF ABBREVIATIONS</b> .....             | <b>67</b> |
|           | <b>LIST OF FIGURES</b> .....                   | <b>68</b> |
|           | <b>LIST OF TABLES</b> .....                    | <b>70</b> |

# 1. INTRODUCTION

In the ATLAS high energy physics experiment at CERN, a new module called ATLAS Local Trigger Interface (ALTI) is being developed. This module provides the interface between the Level-1 Central Trigger Processor (CTP) and the timing, trigger and control (TTC) optical broadcasting network to the front-end electronics of each of the ATLAS sub-detectors. ALTI is a replacement for four existing modules currently being used in the experiment: Local Trigger Processor (LTP), Local Trigger Processor Interface (LTPI), TTC VMEbus Interface (TTCvi) and TTC Encoder/Transmitter (TTCex). It has become increasingly difficult to produce spares for these four modules, and the current spare modules have obsolete and ageing components. In that sense, the ALTI combines and upgrades the functionalities of these modules while preserving backward compatibility. It also extends them and adds new features due to increased amount of programmable logic resources.

ALTI is a custom-made 6U VME64x module made out of two PCBs (motherboard and mezzanine) and it takes up two slots in the VME64x crate. It is an FPGA-based system and uses Xilinx's 7-Series FPGA chip from the Artix family (Artix-7). The module is connected with other modules in the same crate through a common VME backplane. Control software for the ALTI is being executed on a single-board computer (SBC) with Intel's CPU, located in the first slot in the same crate. The SBC runs Scientific Linux operating system and has an on-board interface chip which acts as a PCI-to-VME bus bridge.

As of late 2017, four fully assembled ALTI prototype modules have been available. There are several aspects of the ALTI development and testing that the author has contributed to since: some parts of the FPGA firmware, software for configuration, control and testing of the module, as well as module testing and various performance measurements. For the firmware development, Verilog hardware description language and Xilinx Vivado tool have been used. Low-level software has been written in C++, and it allows access to all the functionalities available in the hardware and firmware. For thorough and systematic testing of the module, a higher level software has been written in the Python programming language.

The author's main contribution to the ALTI project is a software suite for testing and validation of the ALTI prototype modules. Automatization of the testing will allow quick evaluation and qualification of the mass-produced modules which will be necessary for the experiment. Low-level software library will be used further for the run control application development, a control system used to operate the whole experiment.

In Chapter 2 of this document, a brief overview of the current TTC system in the ATLAS experiment will be given. In this way, the reader will be introduced to the specific nomenclature of modules, signals and interfaces being used, so it lays the foundation necessary for the later chapters. This chapter will also emphasize the flaws of the current system the and further explain the motivation to migrate to the new, ALTI-based system.



The ALTI hardware architecture will be presented in the Chapter 3. Then, in Chapter 4, all the functionalities of the module will be explained in detail. These functionalities are reflected in the FPGA firmware, which is also described in Chapter 4. Relevant parts of the firmware (the one the author has contributed to) will be presented in detail, while the others will be described briefly. The software that has been developed is the main topic of the Chapter 5. This includes both the low-level software for configuration, control and testing of the module, as well as the software of higher level used for (semi-)automatized tests. Numerous tests that have been used to verify the proper functioning of the modules are described in Chapter 6. Performance of the ALTI module was determined with various measurements, all of which are described in detail in Chapter 7. The same tests have been done for the modules in the existing TTC system, in order to compare them to the new ALTI module. This comparison is also a subject of Chapter 7. Finally, Chapter 8 summarizes all the work that was done and the results that were obtained, and gives a conclusion to the thesis.

## 2. OVERVIEW OF THE CURRENT SYSTEM

A brief introduction to the relevant parts of the ATLAS experiment is given in Section 2.1. Then, in Section 2.2, the TTC system that the ALTI is made for is described. This description includes the current distribution of modules in the system, as well as the main signals being used in the TTC. Section 2.3 gives an overview of the four modules that are currently being used in the TTC system (so called "legacy" modules). Finally, Section 2.4 describes the VMEbus that these legacy modules are based on, as is the ALTI module.

### 2.1. ATLAS experiment

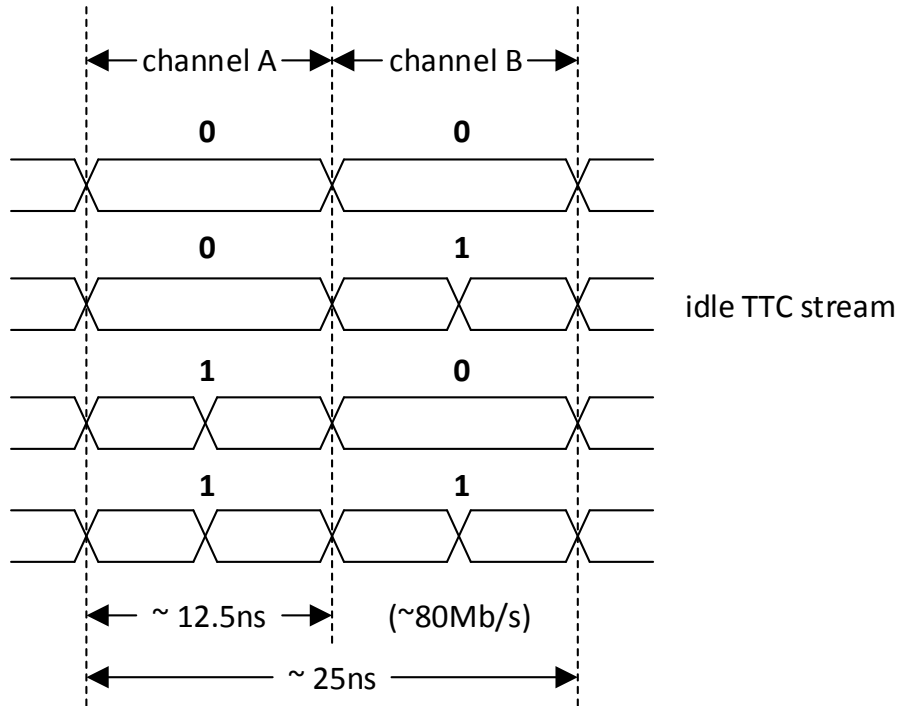
The ATLAS experiment is a general-purpose particle physics experiment operating at the Large Hadron Collider (LHC) at CERN [1]. The full LHC turn consists of 3564 bunch crossings (BC). The bunch clock is the main timing signal produced by the LHC and has the frequency of 40.079MHz. The second timing signal is the orbit (ORB) signal, which indicates the start of a new LHC turn and allows one to identify the bunch crossings. The LHC orbit period is about 90 $\mu$ s, while the orbit pulse width is 40BCs, or about 1 $\mu$ s.

Several tens of proton-proton collisions that happen each bunch crossing yield about a billion collisions each second [1]. Particles created by these collisions are then captured by various types of particle detectors. The Level-1 calorimeter and Level-1 muon trigger systems identify interesting particle candidates. The Central Trigger Processor (CTP) makes combinations of these and takes the final decision, reducing the event rate to a maximum of 100kHz [2]. This is called the Level-1 trigger system and the corresponding event signal produced by the CTP is called Level-1 Accept (L1A). The High Level Trigger (HLT) system of ATLAS operates at lower event accept and readout frequencies than the Level-1 trigger system. High level trigger systems are based on commercial computers and networks, unlike the Level-1 trigger system which is based on custom electronics.

### 2.2. TTC system

Level-1 central trigger system is followed by the Timing, Trigger and Control (TTC) system, whose backbone is the optical transmission network used for communication with the sub-detector front-end electronics. The TTC system is also based on custom electronics, and is composed of several VME modules or boards. This system is responsible for the distribution and fan-out of the timing signals (BC, ORB), the trigger signal (L1A, together with an 8-bit trigger type word) and the control commands like Bunch Counter Reset (BCR) and Event Counter Reset (ECR). Proper timing and control provided by the TTC system is essential for making sure that the right data ("interesting" physics) are read out from the sub-detector buffers in due time. A detailed overview of the TTC system can be found on one of the websites listed in the bibliography [3].

The triggers (channel A) and commands (channel B) are time-division multiplexed and biphasemark encoded into an optical signal called the TTC stream, and then sent to the front-end electronics of each sub-detector system via optical fibre networks. Multiplexing of channels A and B and their encoding into the TTC stream is shown on Figure 2.2.1.



**Figure 2.2.1. Multiplexing and encoding of the TTC channels A and B.**

Two bits are being transmitted on every bunch crossing, one for each channel. Channels A and B are thus interleaved and the carrier frequency is two times the BC frequency, which gives the rate of about 80M bits per second. A transition on the TTC stream indicates a logic "1", while logic "0" is assumed if no transition occurs, as indicated on the Figure 2.2.1. When no triggers are being accepted and no B-channel commands are being transmitted, the TTC stream is idle. In that case, channel A is a logic "0" (no transitions occurring) and channel B is a logic "1" (transitions occurring).

The front-end electronics of the sub-detectors use a TTC Receiver (TTCrx) ASIC module to receive and decode the TTC stream [4]. From the TTC stream only, the receiver is able to decode and de-multiplex the channels A and B. First, the receiver makes an initial guess on which bit corresponds to which channel, since the channels are interleaved. Because of the constraint made on the maximum L1A rate and the fact that the idle bits are different for channels A and B, the receiver is able to switch the phase of the stream if the initial guess turns out to be wrong. The BC clock is also recovered in the process.

TTC system is partitioned in order to be able to run sub-detectors (or parts of sub-detectors) independently and in parallel. Associated with each sub-detector is a link from the CTP to one or more TTC partitions. Currently, there are 21 connections from the CTP to 35 different TTC partitions in the ATLAS experiment, some of which are daisy-chained. Currently, the maximum daisy chain length is three partitions. Each partition is typically composed of the following modules: LTPI (optional), LTP, TTCvi and TTCex. A detailed sketch of the current TTC system distribution is shown on Figure 2.2.2.

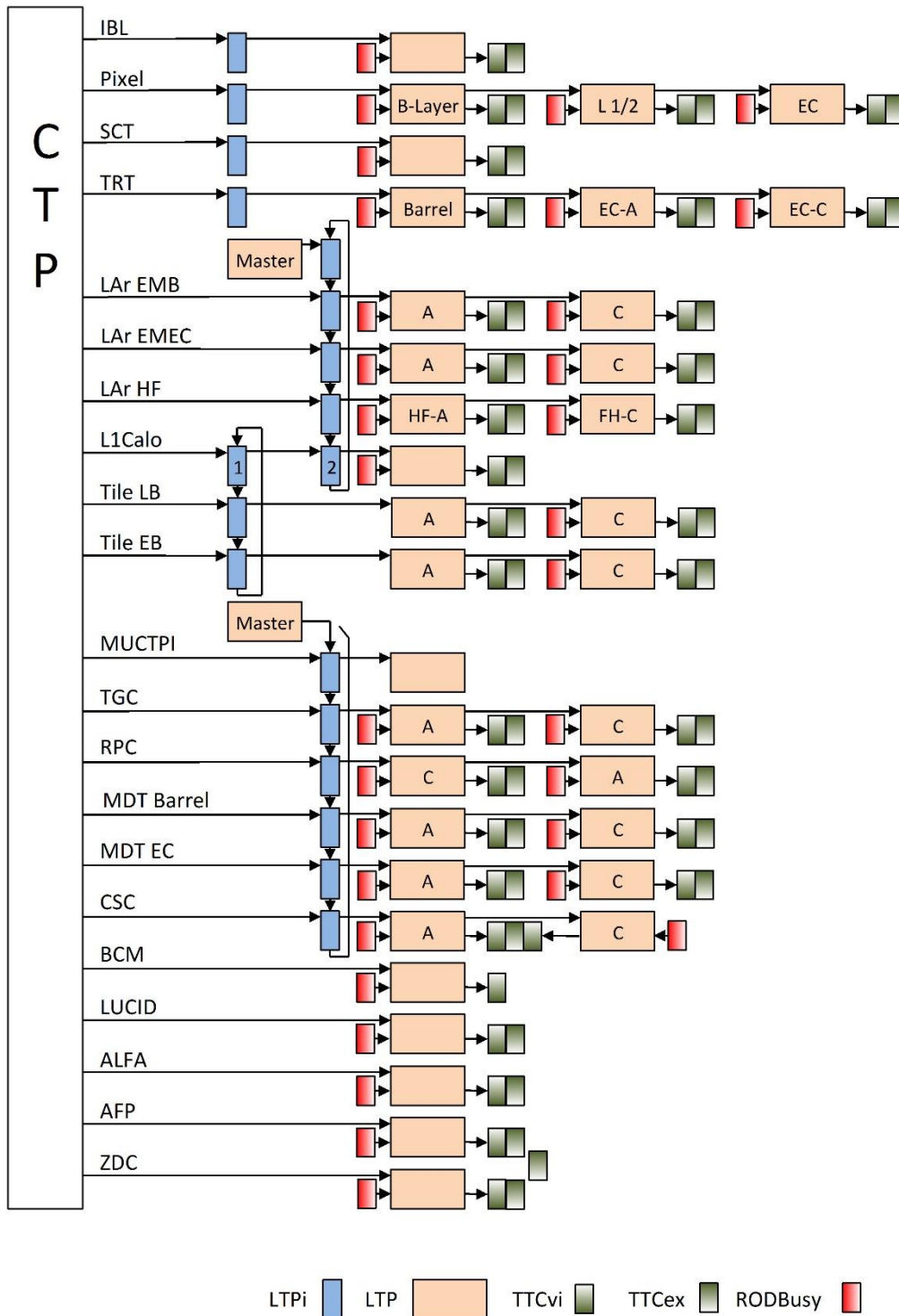


Figure 2.2.2. Current TTC distribution network in the ATLAS experiment [12].

However, the next ATLAS upgrade will include new sub-detectors which will require TTC modules. Unfortunately, CERN is low on spare TTC modules. Also, some of the TTC modules are now more than 15 years old and use components that are now obsolete. Therefore, it is not possible to reproduce modules for the new sub-detectors and to replenish the stock of available spare modules. Other issues of the legacy modules include: aging effects, no firmware replacement flexibility and very limited monitoring capabilities.

This is why the new ALTI module was created. It is designed to replace a combination of LTPI, LTP, TTCvi and TTCex with a single module. The replacement also creates a benefit of getting more free space in TTC VME crates. ALTI provides almost full backwards compatibility with the hardware of other modules. Full compatibility is not provided from the interfaces point of view, though, since space on the front panel is lost in the transition from four separate VME boards to a single, 2-slot VME board. More details on the compromises made because of this will follow in the next chapter.

From the point of view of functionality, the ALTI keeps all the functions of the previous modules. Some of them are extended and optimized, though, since more powerful logic resources are available. Additional useful functionalities are available, too.

### 2.2.1. TTC signals

In order to understand the functionality of the TTC system, it is necessary to get familiar with the interface signals being used. There are 22 digital TTC signals in total (some of them logically grouped together), and they are listed in the Table 2.2.1. For each of the TTC signals, one can see a description of a typical use in the experiment in the same table. Direction column in this table serves to make a distinction between signals going downstream (from CTP, forward) and upstream (to CTP, backward).

**Table 2.2.1. List of TTC signals.**

| TTC SIGNAL   | DIRECTION | DESCRIPTION   |
|--------------|-----------|---|
| BC           | forward   | Bunch crossing clock: 40.079MHz, 50% duty ratio.  |
| ORB          | forward   | Periodic signal representing one LHC turn. Period is 3564 bunch crossings, pulse width is 40BC.                           |
| L1A          | forward   | Level-1 trigger accept signal of 1BC pulse width.   |
| TTR[3..1]    | forward   | Auxiliary triggers generated locally by the partition.  |
| BGO[3..0]    | forward   | Signals for sending B-channel TTC commands.   |
| TTYP[7..0]   | forward   | 8-bit trigger type identification word associated with each L1A.  |
| BUSY         | backward  | Used to inform the CTP to introduce L1A dead-time, i.e. throttle L1A generation when the readout buffers are overwhelmed. |
| CALREQ[2..0] | backward  | 3-bit word issued by the sub-detector and used by the CTP to generate calibration triggers.                               |

## 2.3. Legacy TTC modules

All of the legacy TTC modules use the TTC signals mentioned in the previous section. Each of these modules has a particular set of functionalities which will be described in this section. The modules differ in the interfaces for the TTC signals on their front panels. This can be clearly seen on Figure 2.3.1, where the legacy TTC modules are shown from the front panel view.

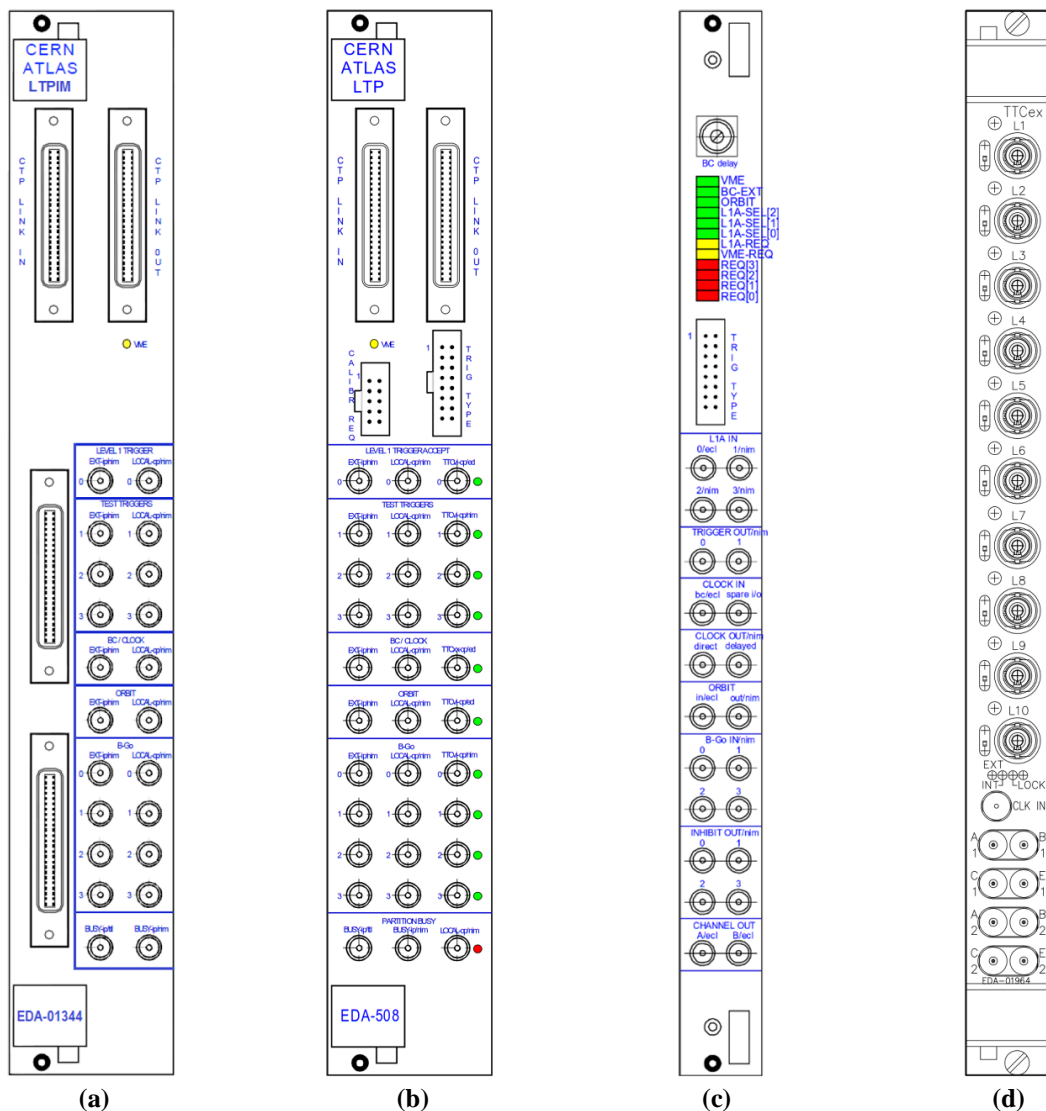


Figure 2.3.1. Legacy TTC modules, front panel view: (a) LTPI [7], (b) LTP [5], (c) TTCvi [8], (d) TTCex [9].

### 2.3.1. Local Trigger Processor (LTP)

The main purpose of the LTP is to receive the timing, trigger and control signals from the CTP and inject them into the TTC distribution system through TTCvi. More details can be found in the LTP technical description and user manual [5].

Connection with the CTP is done with LVDS-LINK cables. The LTP has two female 50-pin 3M Mini Delta Ribbon (MDR) connectors, the one being an input (CTP\_IN link) and the other an output (CTP\_OUT link), as can be seen on Figure 2.3.1. (b). In case of daisy-chaining the TTC partitions, the output is used to connect the module with the downstream LTP.

Electrical signals can also be injected in the LTP using coaxial cables and LEMO connectors on the front panel. There are also LEMO connectors for output signals, which are very useful for looking at the signal waveforms on the oscilloscope. Input LEMO connectors are suited for standard Nuclear Instrumentation Module (NIM) logic levels. This is a standard that uses negative logic, with 0V low voltage, and -0.8V high voltage on a 50Ω termination [6]. There are two sets of output LEMO connectors: one set for local use and the other for connecting to the TTCvi downstream. All local LEMO outputs are NIM-level. The ones used for the connection with the TTCvi are a bit different: latency critical signals (BC, ORB and L1A) are routed in ECL logic, which is faster.

For trigger type and calibration request, there are custom made input/output ports on the front panel. Trigger type connector is usually used as an output and is connected to the TTCvi connector of the same type with a flat cable. This connector can also be used as an input (programmable), as well as the calibration request connector.

In addition to being run by the CTP (so called "CTP slave" mode), the LTP can also run in standalone mode (so called "Master" mode). That means that the TTC signals can be generated locally by the LTP. Master mode is typically used in laboratory testing, when the full CTP system is not available.

For generating the BC clock internally, there is an on-board 40.079MHz quartz oscillator. The internal ORB signal is derived directly from this clock. Orbit signal and the other TTC signals can also be generated from the on-board memory called pattern generation memory. Pattern generation memory can be run in continuous mode (pattern gets repeated periodically) and in single-shot mode (triggered by the VME access or the ORB signal). Entries of this memory contain the desired values of TTC signals, and one entry correspond to a single BC period.

To sum up, the LTP is used for TTC signal propagation and generation. Switching of the input signals to the outputs is very flexible: signals on the CTP\_OUT link and LEMO outputs can be sourced from the CTP\_IN link, front panel connectors, or generated internally in the LTP. However, the latency for different TTC signals through the LTP is not equal because of the different circuitry.

### **2.3.2. Local Trigger Processor Interface (LTPI)**

The main purpose of the LTPI is to help run TTC partitions in parallel. More details can be found in the LTPI functional description [7].

Unlike the LTP, the LTPI has two pairs of LVDS-LINK connectors called CTP\_IN/CTP\_OUT link and LTP\_IN/LTP\_OUT link. This allows the reception of TTC signals from an upstream CTP, as well as from another parallel LTPI. Both LVDS-LINK inputs have a separate equalizer in the LTPI to allow the undistortion of the TTC signals when long LVDS cables are used.

As for the LEMO connectors, there are NIM-level input and output connectors for BC, ORB, L1A, TTR and BGO signals. Separate connectors for NIM and TTL-logic levels are available for the BUSY input signal. There are no input/output ports for trigger type and calibration request, so these signals can be propagated only through the LVDS connectors.

Preceding the CTP\_OUT link is the delay chip, which allows fine shifting (with 0.5ns step) of the TTC signals (all except the BC).

To summarize, the main function of the LTPI is to switch TTC signals between partitions.

### **2.3.3. TTC VMEbus Interface (TTCvi)**

The main purpose of the TTCvi is to encode the signals for TTC channels A and B. Channel A is carrying the L1A triggers, while the channel B is carrying TTC commands, as described in Section 2.2. Commands on the B channel are framed, formatted and protected with Hamming code for error detection and correction. More details can be found in the TTCvi functional description and user manual [8].

One can select between four external triggers: L1A and three test triggers. There is also an ability to generate random triggers with a few predefined average frequencies, ranging from 1Hz to 100kHz.

There are two types of TTC command formats: short and long. Start bit is indicated by a logic "0" on the B-channel portion of the TTC stream. The second bit indicates the type of the frame: logic "0" is used for short, and logic "1" is used for long commands. Then, the remainder of the command depends on the frame type, as shown on figures 2.3.2 and 2.3.3 for short and long commands, respectively.

| start    | frame type | data [7..0]    | checksum [4..0] | stop     |
|----------|------------|----------------|-----------------|----------|
| <b>0</b> | <b>0</b>   | <b>ddddddd</b> | <b>cccc</b>     | <b>1</b> |

**Figure 2.3.2. Frame format for short TTC commands.**

| start    | frame type | address [13..0]     | ext/int  | 1        | sub-address [7..0] | data [7..0]    | checksum [6..0] | stop     |
|----------|------------|---------------------|----------|----------|--------------------|----------------|-----------------|----------|
| <b>0</b> | <b>1</b>   | <b>aaaaaaaaaaaa</b> | <b>e</b> | <b>1</b> | <b>sssssss</b>     | <b>ddddddd</b> | <b>cccccc</b>   | <b>1</b> |

**Figure 2.3.3. Frame format for long TTC commands.**

Short commands just carry an 8-bit data field. On the other hand, the long commands carry a few additional fields: 14-bit address field for addressing a specific TTCrx receiver, 8-bit sub-address field for addressing a specific register in the addressed receiver. When the address field is 0, it means that the command is being broadcasted to all of the TTCrx receivers. Otherwise, a receiver with a given address is being individually addressed. External/internal bit field indicates whether the data is being written to the addressed register of the receiver, or made available externally and transmitted to the front-end electronics. Every command is terminated with a stop bit that returns the B-channel to the idle state.

Hamming code is used to protect the command contents. Start, stop and frame type bits are not included in the Hamming code scheme. For short commands, Hamming code with Hamming distance 5 is used to protect 8 information bits. On the other hand, Hamming code with Hamming distance 7 is used to protect 32 information bits for the long commands. Single-bit error correction and double-bit error detection is possible using this code.

Besides the short/long command distinction, there is another major distinction between the TTC commands: synchronous and asynchronous commands.

Synchronous commands have a precise timing with respect to the LHC orbit. These are time-critical commands like the Bunch Counter Reset, which is sent at a fixed point on each orbit in order to adjust the phase of the bunch counters in the receivers. BCR is a short, synchronous command that is broadcasted to all of the receivers. Its data field is equal to 1.

Asynchronous commands are those that are not time-critical, like commands for calibration of the front-end electronics. Their timing is not fixed to the orbit and they have lower priority than synchronous commands. Event Counter Reset (ECR) is a short, asynchronous command that is sent to reset the 24-bit L1A (event) counters in the receivers. Data field of the ECR is equal to 2. Another example for asynchronous commands are the trigger type cycle commands. These are the four long commands that are being transmitted following each L1A. Trigger type word is the data content of the first of these commands, while the other three commands are used to transmit the current value of a 24-bit event or orbit counter in the TTCvi (programmable).

There are four independent channels for sending commands, called BGO0, BGO1, BGO2 and BGO3. Associated with each of the four channels is a dedicated FIFO memory for storing the commands to be transmitted. For sending the commands in a loop after the FIFO gets empty, a so called retransmit FIFO mode is used. Choosing the types of commands associated with each BGO channel is completely flexible: all of them support synchronous (repetitive and single-shot) and asynchronous commands (triggered by external BGO signal, a VME access or by a write access to the FIFO). However, by convention in the ATLAS experiment, the BGO0 channel is associated with the BCR, and the BGO1 is associated with the ECR.



The timing of synchronous commands is governed by internally generated inhibit signals. Each BGO channel has a separate inhibit signal associated with it. The inhibit signal is a pulse train with the period of one orbit, with fully programmable pulse width and delay with respect to the orbit. The transmission of a synchronous command commences at the end of the inhibit pulse. Thus, the width of the inhibit pulse must be chosen properly, such that any ongoing command gets transmitted during that interval of time. The empirically determined value of the minimum inhibit width for the TTCvi is 51BCs. This allows for the full transmission of a long command over this interval of time.

Since the synchronous commands must have precise timing, they have the highest priority. Moreover, channels associated with signals BGO0 through BGO3 have a descending priority. The full list of commands ordered by priority, from highest to lowest, is the following:

- 1) Synchronous commands BGO0 to BGO3
- 2) Asynchronous commands BGO0 to BGO3
- 3) Trigger type commands
- 4) VME-mapped commands

#### **2.3.4. TTC Encoder/Transmitter (TTCex)**

The main purpose of the TTCex is the conversion the electrical A and B channels to the optical TTC signal. More details can be found in the TTCex user manual [9].

Lasers for the TTCex transmitters operate at the wavelength 1280-1330nm [9]. There are 10 optical outputs available on the front panel, each providing the optical power of about 0dBm. Standard ST-type optical connectors are used.

## **2.4. VMEbus**

VMEbus has been the technology of choice in CERN for many years. It originated in 1982 and it provides an open mechanical, electrical and protocol standard [10].

VMEbus crates provide a common backplane and include a mounted power supply. Typical VME crate consists of 21 slots for inserting the modules, with the SBC installed in the first slot as a master and arbiter on the bus. In particular, the Concurrent Technologies VP-E24 single-board computer [11] has been used, which has an on-board Tundra Universe II interface chip that acts as a bridge from the Intel's PCI to the VMEbus on the backplane. VME boards come in three different standards based on their size and the connectors they utilize: 3U, 6U and 9U (1U = 1.75 inches). All of the previously mentioned TTC modules are 6U VME64x boards, as is the ALTI module.

All VME lines use TTL levels with a voltage swing of 0V to 5V. Data and address lines are active high, while the protocol lines (data and address strobes, data acknowledge, etc.) are active low. VMEbus is big-endian, so it stores the most significant byte a 32-bit word at the lowest address. It supports both single cycles and block transfers. Legacy TTC modules use 24-bit addressing and the ALTI uses 32-bit addressing, while they all use 32-bit data on the VMEbus for communication.

By the VME64x standard, each VME slave has an address space of 512kB reserved Configuration ROM (CR) and Control and Status Register (CSR) sections. This is called CR/CSR space, and is used in 24-bit addressing mode. Included in this section are identification registers for identifying the board and the manufacturer. However, the most important registers in the CR/CSR space are the BAR and the ADER0. The BAR is a read-only register containing the slot number of the given module. The ADER0 is a read/write register used to dynamically change the VME base address of the module. By default, ADER0 is preloaded upon startup with a value depending on the BAR. This is called geographical addressing.

## 3. ALTI HARDWARE SPECIFICATION

In this chapter, the hardware specification of the ALTI module will be briefly described. A more detailed description of the ALTI hardware is given in the ALTI specification document [12]. First, the interfaces available on the front panel of the ALTI module are described in Section 3.1. Then, the module architecture will be presented in Section 3.2, with each relevant part of the hardware described in a separate subsection.

### 3.1. Interfaces

The front panel view of the ALTI module is shown on Figure 3.1.1.

On the ALTI front panel, there are four 50-pin 3M MDR female connectors for LVDS-LINK cables. Two of them, called CTP\_IN and ALTI\_IN, are used as inputs. Corresponding output ports are called CTP\_OUT and ALTI\_OUT. These are analogous to the LTPI parallel cable connectors. The connectors are fully compatible with the ones used in LTP and LTPI modules, with the pin-out shown in Table 3.1.1. Also, the same LVDS-LINK cables with point-to-point signalling are used interchangeably between the three modules. For the LVDS receivers, a 100 $\Omega$  termination is used.

Several pairs of coaxial LEMO connectors are available for local injection and monitoring of the TTC signals. Because of the limited space on a front panel of a 2-slot VME module, not all the TTC signals have an independent input and output LEMO connector. The list of available LEMO connectors is shown in Table 3.1.2. Since the BGO0 and BGO1 connectors are not available, multiplexing with the inputs for TTR2 and TTR3 and outputs for BGO2 and BGO3 has been introduced. This will be further discussed in the Subsection 3.2.3. All input and output connectors are compatible with NIM logic levels. In addition, the input connector for the BUSY signal can be programmed to accept both NIM and TTL logic levels. NIM inputs are terminated with 50 $\Omega$  resistors internally and the outputs should be terminated with 50 $\Omega$  resistors to obtain the necessary logic levels at the destination.

The cages for the Small Form-factor Pluggable (SFP) transceiver and the dual transmitters are also available on the front panel. Five of those are for the dual transmitters, while the sixth cage is for a transceiver. This gives a total of 11 optical outputs and a single optical input per module.

The calibration request input is in the form of an RJ45 connector, compatible with a standard Ethernet UTP cable differential pair wiring. Hence a standard Ethernet cable can be used as an input. In order to drive the calibration request input from an LTP, a custom patch cable has been made, with a ribbon connector on one side, and a standard RJ45 connector on the other.

Some LED indicators for monitoring and diagnostics purposes are also available on the front panel. These include the LEDs which indicate: power supply status, VME access, detection of L1A, ORB, BUSY and CALREQ, PLL lock, optical link statuses. Bi-colour red/green LEDs are used for some of these to distinguish between correct and faulty functioning.

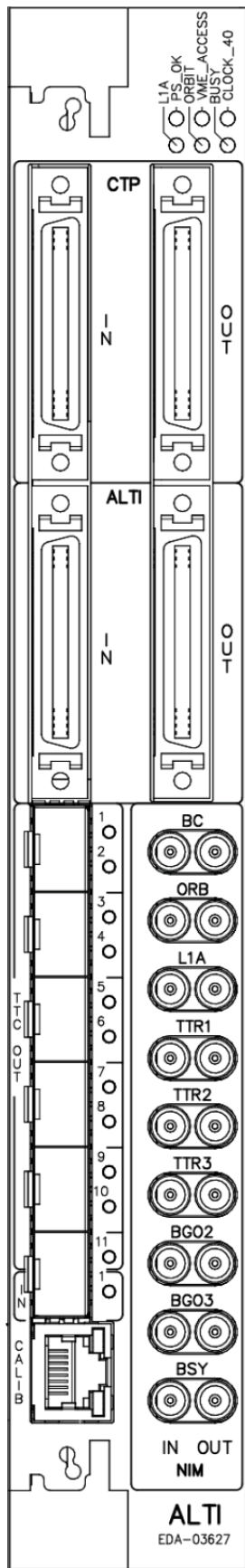


Figure 3.1.1. The ALTI module, front panel view [12].

**Table 3.1.1. LVDS-LINK connector pin-out [5].**

| SIGNAL  | PAIR # | PIN # LEFT | PIN # RIGHT |
|---------|--------|------------|-------------|
| TTR1    | 25     | 50         | 25          |
| TTR2    | 24     | 49         | 24          |
| TTR3    | 23     | 48         | 23          |
| TTYP0   | 22     | 47         | 22          |
| TTYP1   | 21     | 46         | 21          |
| TTYP2   | 20     | 45         | 20          |
| TTYP3   | 19     | 44         | 19          |
| TTYP4   | 18     | 43         | 18          |
| TTYP5   | 17     | 42         | 17          |
| TTYP6   | 16     | 41         | 16          |
| TTYP7   | 15     | 40         | 15          |
| BGO0    | 14     | 39         | 14          |
| BGO1    | 13     | 38         | 13          |
| BGO2    | 12     | 37         | 12          |
| BGO3    | 11     | 36         | 11          |
| CALREQ0 | 10     | 35         | 10          |
| CALREQ1 | 9      | 34         | 9           |
| CALREQ2 | 8      | 33         | 8           |
| BUSY    | 7      | 32         | 7           |
| GND     | 6      | 31         | 6           |
| L1A     | 5      | 30         | 5           |
| ORB     | 4      | 29         | 4           |
| GND     | 3      | 28         | 3           |
| BC      | 2      | 27         | 2           |
| GND     | 1      | 26         | 1           |

**Table 3.1.2. Front panel coaxial LEMO connectors.**

| NAME     | DIRECTION | LOGIC LEVEL |
|----------|-----------|-------------|
| BC IN    | Input     | NIM         |
| BC OUT   | Output    | NIM         |
| ORB IN   | Input     | NIM         |
| ORB OUT  | Output    | NIM         |
| L1A IN   | Input     | NIM         |
| L1A OUT  | Output    | NIM         |
| TTR1 IN  | Input     | NIM         |
| TTR1 OUT | Output    | NIM         |
| TTR2 IN  | Input     | NIM         |
| TTR2 OUT | Output    | NIM         |
| TTR3 IN  | Input     | NIM         |
| TTR3 OUT | Output    | NIM         |
| BGO2 IN  | Input     | NIM         |
| BGO2 OUT | Output    | NIM         |
| BGO3 IN  | Input     | NIM         |
| BGO3 OUT | Output    | NIM         |
| BUSY IN  | Input     | NIM/TTL     |
| BUSY OUT | Output    | NIM         |

### 3.2. Architecture

The ALTI module is a 6U VME64x module that takes two slots in a VME crate. It consists of two PCBs: a motherboard and a mezzanine. Project documents for the motherboard and the mezzanine, including the schematics, the PCB layout and mechanical descriptions are publically available on the CERN Engineering and Equipment Data Management System (EDMS) [13] [14].

All of the logic is located on the motherboard. This includes the Xilinx Artix-7 XCA200T FPGA [15], the power supply network, the I2C network, RAM memories and other discrete logic and integrated circuits. The motherboard houses the VMEbus connectors, two LVDS-LINK input connectors, six SFP modules and the calibration request RJ45 connector.

The mezzanine plugs into the motherboard via a Samtec high-speed connector [16] carrying 180 signals, three power supplies and a ground. It houses all of the coaxial input and output connectors as well as the two LVDS-LINK output connectors.

A fully assembled ALTI prototype module is shown on Figure 3.2.1, while the functional block diagram of the ALTI hardware is shown on Figure 3.2.2.



Figure 3.2.1. Fully-assembled prototype ALTI module.

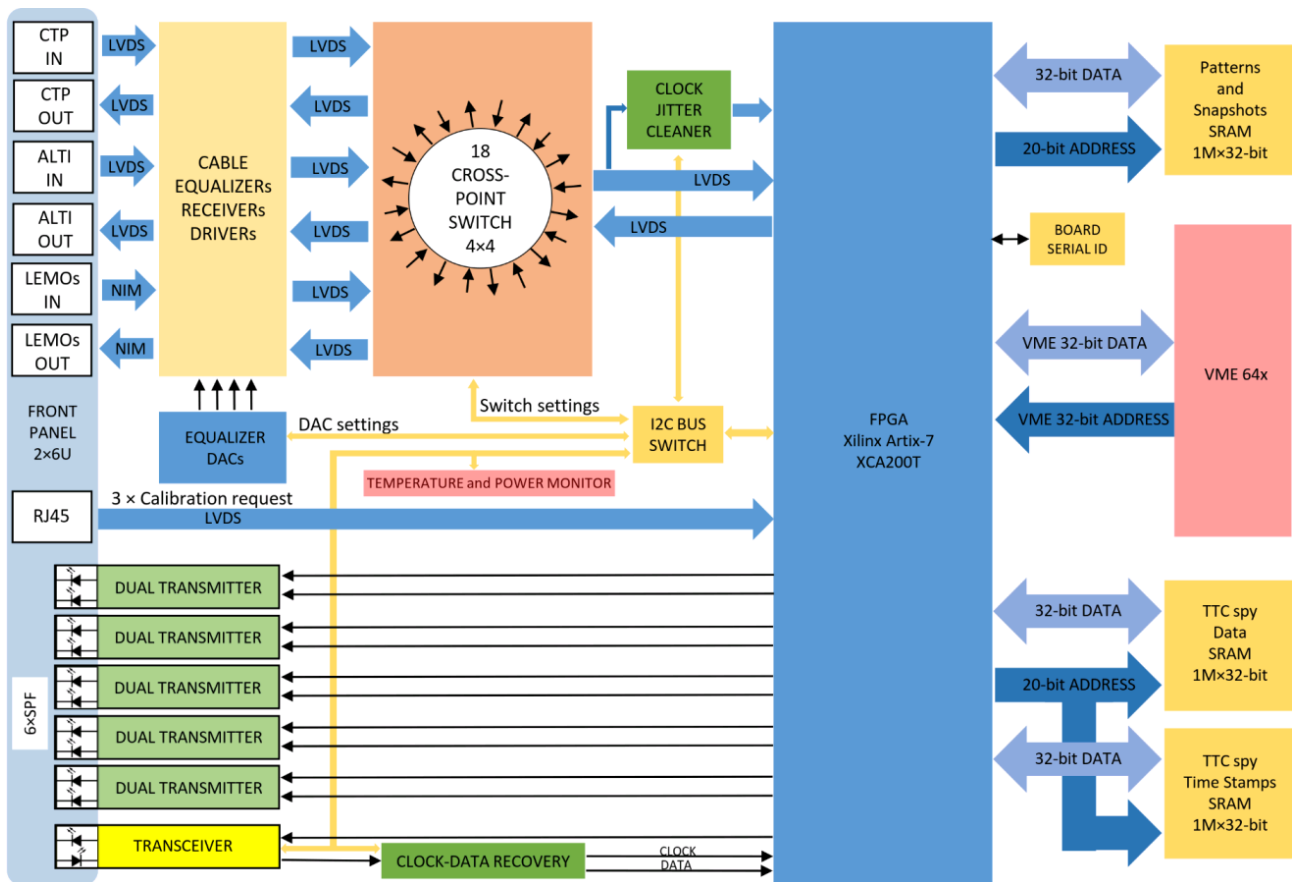


Figure 3.2.2. ALTI functional block diagram [12].

### 3.2.1. Cross-point switches

All forward-going TTC signals, 18 of them in total (see Table 2.2.1), are routed through the Texas Instruments DS10CP154A 4x4 LVDS cross-point switches [17]. This device has an I2C interface and allows independent routing of all four inputs to any of the four outputs, with high speed and low channel-to-channel skew.

Inputs of the cross-point switch are sourcing the corresponding TTC signal from: CTP input LVDS-LINK connector, ALTI input LVDS-LINK connector, LEMO input connector and the FPGA output. For convenience, we call these sources CTP\_IN, ALTI\_IN, LEMO\_IN and FROM\_FPGA, respectively.

Outputs of the cross-point switch are driving the corresponding TTC signal on: CTP output LVDS-LINK connector, ALTI output LVDS-LINK connector, LEMO output connector and the FPGA input. For convenience, we call these destinations CTP\_OUT, ALTI\_OUT, LEMO\_OUT and TO\_FPGA, respectively.

The routing of the Level-1 Accept signal through the cross-point switch is shown on Figure 3.2.3. This generic routing applies to all of the TTC signals, except the following: BC, TTR2, TTR3, BGO2 and BGO3. The slight exceptions to this generic routing are also the TTYP[7..0] cross-point switches, which do not have the front panel LEMO input and output connectors.

As the signal which is clocking the whole module, the BC has a special routing path. Because of the multiplexing on the LEMO coaxial inputs and outputs for TTRs and BGOs, these routings are special, too. These non-generic routings the TTC signals will be described in the following sections.

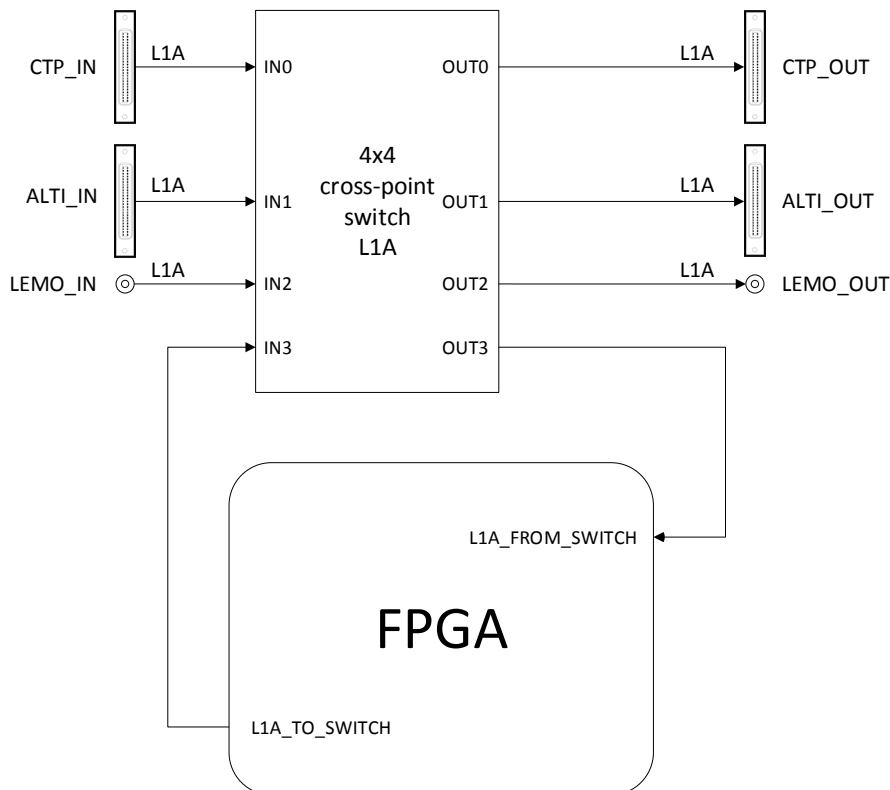


Figure 3.2.3. Example of the generic routing of a TTC signal through the cross-point switch: Level-1 Accept.

### 3.2.2. Clock distribution

The ALTI module has the following clock signal distribution, as shown on the diagram on Figure 3.2.4.

The cross-point switch paths to and from the FPGA differ with respect to the generic TTC signal routing. Before going to the FPGA, the clock output coming from the switch is also forwarded to a 4-channel jitter attenuator and clock multiplier circuit Silicon Labs SI5344 [18]. The role of this circuit is to provide a clean clock to the FPGA, with decreased jitter. From here on, we will call this circuit the "jitter cleaner". The jitter cleaner has an I2C interface and is fully programmable. It features the holdover mode, which is automatically activated once the selected input clock becomes invalid. In holdover mode, the jitter cleaner continues providing the output clock based on the sampled input clock prior to the failure. This minimizes the disturbance of the phase and frequency of the TTC clock, but can also be dangerous because of long-term drifts. It is therefore advised to acknowledge the jitter cleaner interrupt (asserted upon entering the holdover mode) by switching to a stable clock source.

In standalone tests when ALTI acts as a master, the clock can be provided to the FPGA logic with an on-board fixed 40.079MHz fixed-frequency crystal oscillator (FXO). With a help of a 2/1 multiplexer, it is also possible to use the clean clock coming from the FPGA as an input to the cross-point switch.

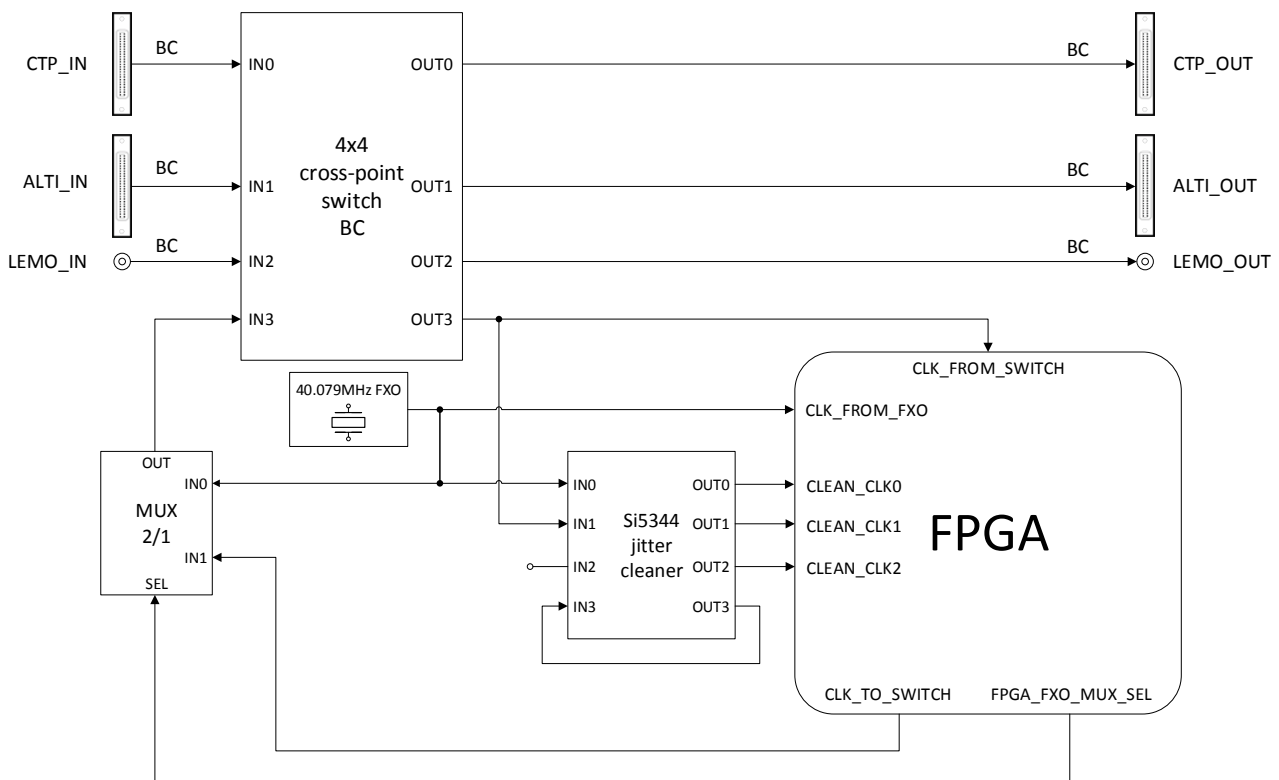


Figure 3.2.4. ALTI clock distribution diagram.

### 3.2.3. TTC signals multiplexing

The lack of space on the ALTI front panel has resulted in the fact that there are no dedicated LEMO inputs and outputs for the BGO0 and BGO1 signals. In order to still be able to inject and monitor these signals using the front panel connector, a multiplexing has been introduced in the following way.

Front panel LEMO input for the TTR2 is fanned out to two cross-point switches: one corresponding to the TTR2 itself, and the other corresponding to the BGO0. The same routing applies to TTR3 LEMO input and TTR3/BGO1 cross-point switches.

Outputs of the BGO0 and BGO2 cross-point switches corresponding to a LEMO\_OUT are connected to the inputs a 2/1 multiplexer. Selection of the desired signal on the LEMO output is done with a single programmable pin from the FPGA. The same routing applies to the cross-point LEMO outputs of BGO1 and BGO3.

The multiplexing of BGO0, BGO2 and TTR2 is shown on the diagram on Figure 3.2.5. The multiplexing of BGO1, BGO3 and TTR3 is shown on the diagram on Figure 3.2.6.

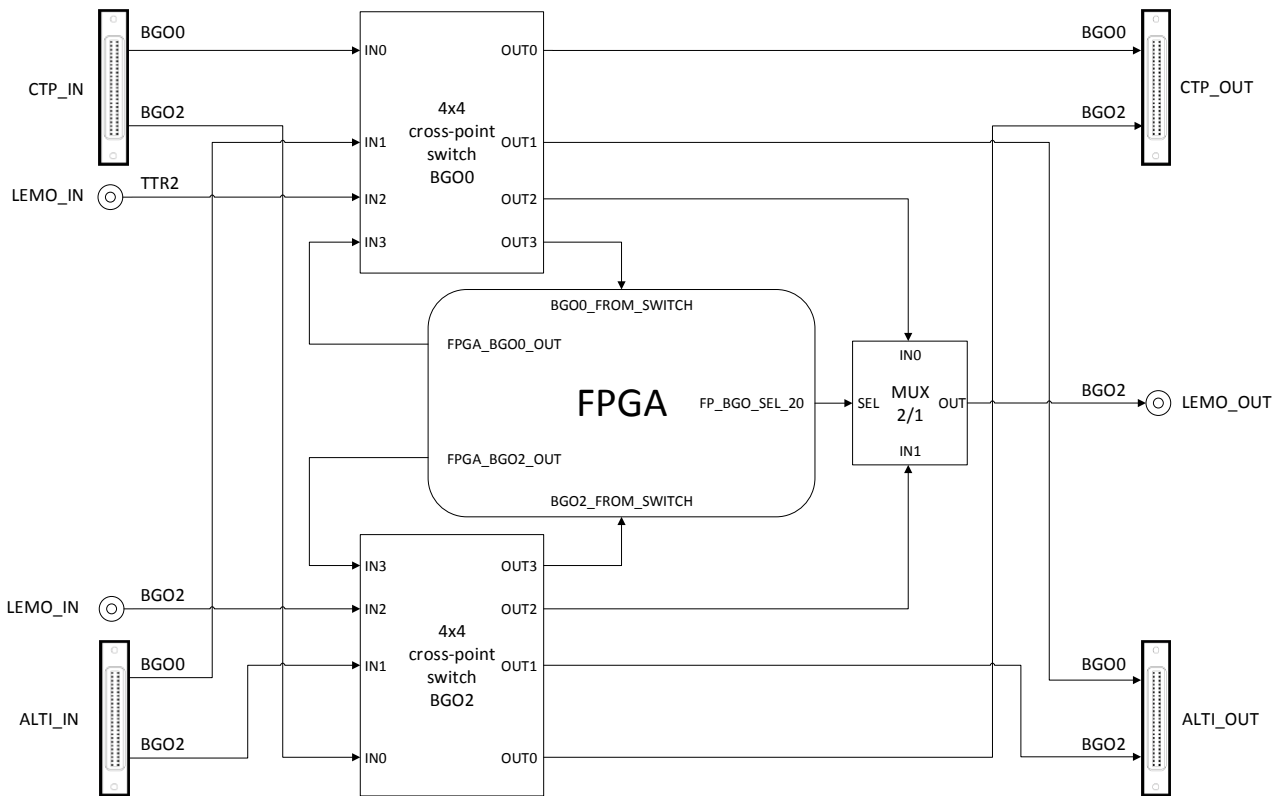


Figure 3.2.5. BGO0/TTR2 multiplexing on the input and BGO0/BGO2 multiplexing on the output.



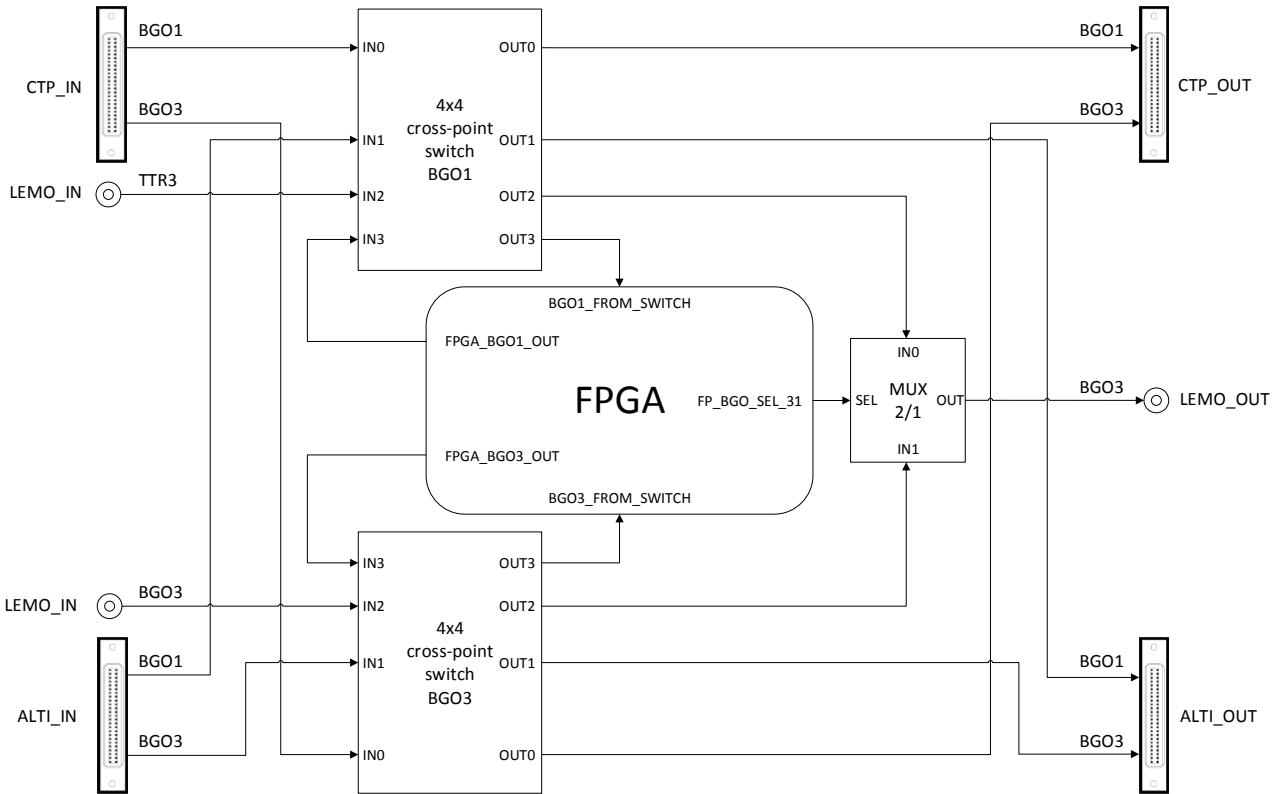


Figure 3.2.6. BGO1/TTR3 multiplexing on the input and BGO1/BGO3 multiplexing on the output.

### 3.2.4. Cable equalizers

High-frequency transmission losses can occur when long LVDS-LINK cables are used. This is common in the experiment, where cables up to 40 meters long are used. In order to compensate for these losses, two cable equalizers are used in the ALTI, for both cable inputs (CTP\_IN and ALTI\_IN).

The circuit that is used for equalization in the ALTI is the Analog Devices AD8123 triple differential receiver with adjustable line equalization [19]. A total of six of these devices are necessary to equalize all 18 forward-going TTC signals coming from a single cable connector. A total of four different analog inputs are used to adjust the equalization: V\_PEAK, V\_POLE, V\_OFFSET and V\_GAIN. Two Analog Devices AD5305 8-bit D/A converters [20] are controlled via I2C and used to drive these equalizer inputs. Offset DC voltage on the output is controlled with V\_OFFSET. The other three inputs are used to adjust the cable frequency response of the equalizer. Single-ended outputs of the equalizers are converted back to differential LVDS signals using Analog Devices ADCMP604 comparators [21].

### 3.2.5. Memories

The ALTI uses Cypress CY7C10612G 1Mx16 SRAM chips [22]. Six of these chips are organized in three memory banks which are accessed as 1Mx32 memories through the VME interface. The first memory can be used either as a pattern generation memory or a snapshot memory. The other two memories are used for storing the TTC triggers and commands received over the optical receiver (spy memory).

### **3.2.6. Optical transmitter and receiver modules**

Five dual transmitter cages are populated with W-Optics SAA-xAF1-111 SFP modules [23], while the Finisar FTLF1323P1xTL SFP transceiver module [24] is used. Both SFP modules use single-mode fibre transmission and reception at 1310nm wavelength. Input and output connectors are of type LC, so LC-ST single-mode patch cords are necessary to connect with the optical splitters currently used in the experiment. The optical power that these SFP modules provide is smaller than the one that the TTCex lasers provide: -9dBm to 0dBm for the dual transmitter, and -5dBm to 0dBm for the transceiver. Whether or not this optical power is sufficient is something that needs to be checked in the experiment.

### **3.2.7. Clock and data recovery from the TTC stream**

The Analog Devices ADN2814 [25] is used for clock and data recovery from the signal received TTC stream. It extracts the clock (carrier, ~160MHz) and the data (multiplexed A and B channels) and provides them to the FPGA for further decoding.

### **3.2.8. Power supply**

From the VME64x backplane, the ALTI module receives the following supply voltages: +3.3V, +5V, +12V and -12V. Four Maxim Integrated MAXM17515 DC/DC converters [26] are used to generate four more supply voltages needed to properly power the FPGA: +1.0V, +1.8V, +2.5V and +3.3V. An additional Traco Power THN 15-1211 DC/DC converter [27] is used to generate a -5V supply voltage, necessary for the NIM-level inputs and output drivers.

### **3.2.9. Hardware monitoring**

All of the supply voltages can be monitored by a Linear Technology LTC2991 voltage monitor [28]. This sensor allows the monitoring of eight input voltages, plus the supply voltage of the chip itself. It also measures the internal temperature and has an I2C interface. Another temperature sensor, Maxim Integrated MAX1617A [29], is used to measure both local and remote diode temperature. Remote diode port is connected to the XADC block of the FPGA in order to measure the temperature of the FPGA die.

### **3.2.10. I2C network**

Numerous devices on the ALTI board mentioned so far have an I2C interface. An I2C switch device Texas Instruments TCA9546A [30] is used to split the devices into four bus sections, in order to avoid conflicting I2C addresses. For example, the cross-point switch devices have input pins for two LSBs of the I2C address. This allows for four different devices addressable on the same I2C bus section. Without the bus switch, it would be impossible to put all 18 cross-point switches on the same I2C bus.

The full list of I2C-addressable slave devices and their I2C sections and addresses is shown in Table 3.2.1. The I2C master which reads from and writes to these slave devices is implemented in the FPGA firmware.

**Table 3.2.1. The list of slave devices on the ALTI I2C network.**

| <b>BUS SECTION #</b> | <b>I2C SLAVE ADDRESS</b> | <b>DEVICE</b>   |
|----------------------|--------------------------|---|
| N/A                  | 111 0000 = 0x70          | I2C bus switch (TI TCA9546A)                                    |
| 0                    | 000 1100 = 0x0c          | DAC for CTP_IN equalizer voltages (AD 5305)                     |
|                      | 000 1101 = 0x0d          | DAC for ALTI_IN equalizer voltages (AD 5305)                    |
|                      | 110 1000 = 0x68          | Jitter cleaner (SI 5344)  |
| 1                    | 101 1000 = 0x58          | BC cross-point switch (TI DS10CP154A)                           |
|                      | 101 01XX = 0x54..0x57    | TTYYP[4..7] cross-point switches (4 x TI DS10CP154A)            |
|                      | 101 00XX = 0x50..0x53    | TTYYP[0..3] cross-point switches (4 x TI DS10CP154A)            |
| 2                    | 101 00XX = 0x50..0x53    | BGO[0..3] cross-point switches (4 x TI DS10CP154A)              |
|                      | 101 1000 = 0x58          | ORB cross-point switch (TI DS10CP154A)                          |
|                      | 101 01XX = 0x54..0x57    | L1A and TTR[1..3] cross-point switches (4 x TI DS10CP154A)      |
| 3                    | 100 0000 = 0x40          | Clock and data recovery circuit (ADN2814)                       |
|                      | 101 000X = 0x50..0x51    | SFP transceiver, extended address space (Finisar FTLF1323P1xTL) |
|                      | 001 1000 = 0x18          | Local and remote temperature sensor (MAX1617A)                  |
|                      | 100 1000 = 0x48          | Voltage and temperature monitor (LTC2991)                       |

## 4. ALTI FUNCTIONALITY AND FIRMWARE

This chapter briefly describes the ALTI functionalities implemented in the FPGA firmware. The emphasis is put on describing the configuration and control of various firmware blocks, which is then used in the low-level ALTI software. A more detailed description of the ALTI firmware implementation is given in the ALTI specification document [12].

The ALTI FPGA firmware is organized as shown on the high-level functional block diagram on Figure 4.1. Some of the blocks shown will be briefly discussed in the following sections. For a more detailed description of the other blocks is given in the ALTI specification document [12].

Each of the firmware blocks has a number of VME-mapped control and status registers. Some of them are also associated with on-chip FIFO buffers and external RAM memories, which are also VME-mapped. The full ALTI VME address space of 16MB is divided in blocks, as shown in Table 4.1.

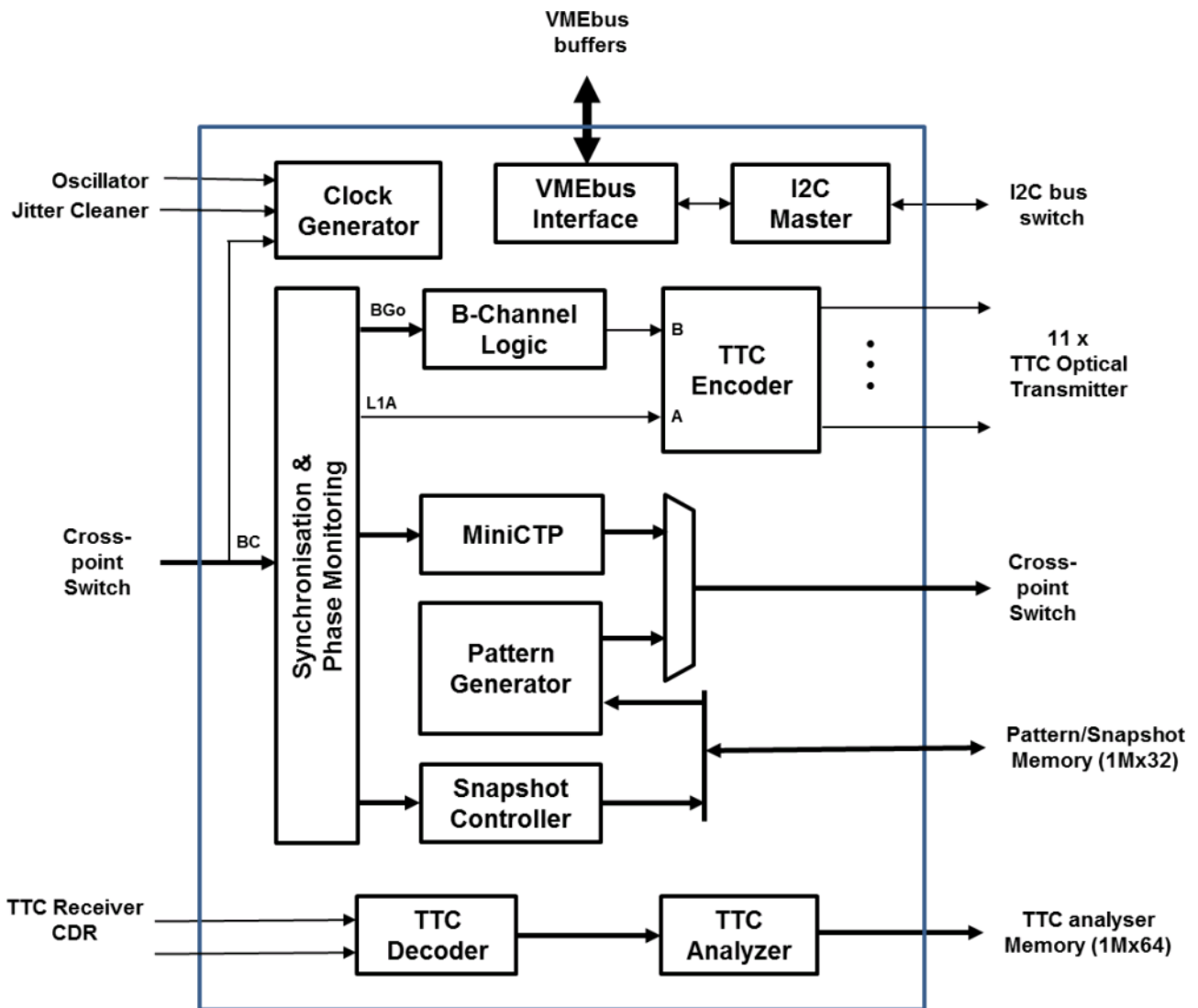


Figure 4.1. High-level functional block diagram of the ALTI firmware.

**Table 4.1. ALTI VMEbus address space map.**

| ADDRESS RANGE                    | DESCRIPTION                                    |
|----------------------------------|--|
| 0x00000000<br>..<br>0x0000FFFF   | Registers, 64kB = 16k words                    |
| 0x00010000<br>..<br>0x000107FF   | BGO0 FIFO, 2kB = 512 words                     |
| 0x00020000<br>..<br>0x000207FF   | BGO1 FIFO, 2kB = 512 words                     |
| 0x00030000<br>..<br>0x000307FF   | BGO2 FIFO, 2kB = 512 words                     |
| 0x00040000<br>..<br>0x000407FF   | BGO3 FIFO, 2kB = 512 words                     |
| 0x00050000<br>..<br>0x000507FF   | TTYF FIFO, 2kB = 512 words                     |
| 0x00080000<br>..<br>0x00080FFF   | QuickBoot FIFO, 4kB = 1k words                 |
| 0x00400000<br>..<br>0x007FFFFFFF | TTC spy "command" RAM memory, 4MB = 1M words   |
| 0x00800000<br>..<br>0x00BFFFFFFF | TTC spy "timestamp" RAM memory, 4MB = 1M words |
| 0x00C00000<br>..<br>0x00FFFFFFF  | Pattern/snapshot RAM memory, 4MB = 1M words    |

As written before, the ALTI preserves the functionalities of the TTC legacy modules: LTPI, LTP, TTCvi and TTCex. However, it is important to emphasize here what novelties and improvements the ALTI provides in terms of functionalities.

The biggest improvements are made in the monitoring capabilities. Phases of the input signals can be monitored and it is possible to take snapshots of incoming TTC signals and store them in the memory. Another new monitoring capability is the optical TTC stream analyzer, which allows to decode and store the triggers and commands that are being sent on the TTC stream.

Improvements have also been made in the pattern generation functionality. As will be explained, the pattern compression allows more efficient memory management and provides longer effective patterns than the ones provided by the LTP.

It is also envisioned to implement a "miniCTP" block with some CTP-like functionalities like: simple and complex deadtime generation for L1A throttling, random triggers with pseudo-random prescaling, etc.

## 4.1. Clocking

A Mixed-Mode Clock Manager (MMCM) IP provided by Xilinx is used to generate all clocks required in the FPGA, including: ~160MHz (four times the BC frequency) clock needed for the TTC encoding and 90 degree phase-shifted 40.079MHz clock needed for the input synchronization. The core has two clock inputs and the user can select which one is used, i.e. multiplied and phase-lock looped:

- 1) 40.079MHz on-board crystal quartz oscillator ("PRIMARY\_FXO")
- 2) BC input coming from the cross-point switch directly, or from the jitter cleaner ("SECONDARY\_EXT")

Until it was decided whether to use the jitter cleaner or not, two separate versions of the firmware were kept, differing only in what clock was connected to the "SECONDARY\_EXT" input.

In addition to the generator functionality, the MMCM also provides monitoring for four user clocks. With three status bits for each input, it can be checked if the input clocks are present, if any glitches were caught or if the frequency is outside some specified range. Four clocks that are monitored are coming from the four cross-point BC sources: CTP\_IN, ALTI\_IN, LEMO\_IN and TO\_FPGA.

Another functionality of the MMCM is the fine-tuned phase shift of the output clock. One step is about 15ps for a VCO frequency of 40.079MHz used in this particular case. This feature is extremely useful for the front-end electronics for shifting the TTC stream.

Most of the internal FPGA logic is running on the 40.079MHz clock provided by the MMCM. However, some parts of the logic like the MMCM control register and the I2C master core are always running on the FXO. In this way, it is always possible to recover from the unexpected losses of the external TTC clocks. In such cases, the user can change the cross-point switch settings and reset the MMCM PLL to change the external clock being used.

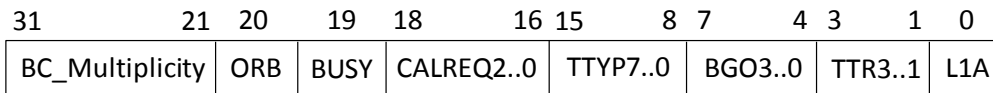
## 4.2. Input signal synchronization

The input TTC signals arriving at the FPGA are sampled by four 90 degree phase-shifted clocks. These four clocks (shifted 0, 90, 180 and 270 degrees from the main clock) create four bins that the input signal edges can fall into. For each of the input signals, the positive and negative edges are counted in this way an accumulated in 3-bit histograms. These histograms allow one to monitor the phase of the input signals.

When the main clock and the input signal are synchronized, all the edges fall into a single histogram bin, or two bins at most (when the input signal edges are basically aligned with the clock edges). Otherwise, the clock and the signals are not synchronized, or there is a hardware problem with a particular signal line. Based on the histogram, the user can select which of the four phases should be used to latch the input signal safely.

## 4.3. Pattern generation

The pattern memory has a capacity of 4MB or 1M 32-bit words, whose format is shown on Figure 4.3.1. Least significant bits of each entry represent a pattern of TTC signals to be generated. The 11 most significant bits are reserved for the multiplicity, which represents the duration of the given pattern in BCs, up to a maximum of 2048 BCs per entry. This compression mechanism allows for longer patterns to be stored in the memory, compared to the LTP which stores a separate entry for each BC.



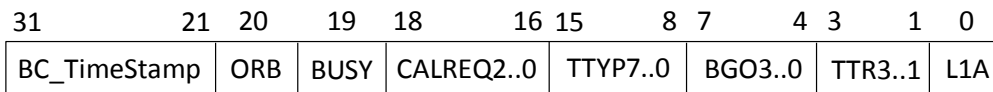
**Figure 4.3.1. Pattern generation memory format.**

Each TTC signal can be independently enabled and disabled from the pattern. Two registers control the addresses of the start and stop entries of the pattern to be generated. The pattern can be generated in two different modes: one-shot and repeated. In case of a repeated generation, the address pointer returns the start address after reaching the stop address, so the pattern is generated in a loop.

#### 4.4. Snapshot taking

The snapshot memory has a capacity of 4MB or 1M 32-bit words and is shared with the pattern generation memory. The format is shown on Figure 4.4.1 and is basically identical to the pattern format. The only difference is that the 11 most significant bits are here interpreted as a timestamp.

Whenever there is a change in any of the incoming 21 TTC signals (all except the BC), one such word gets written to the snapshot memory. Each entry has an 11-bit timestamp associated to it. Timestamps are relative, i.e. represent the number of BCs elapsed since the last change. If no changes occur in 2k BCs, an overflow entry (with all bits equal to "1" in the timestamp) is stored in the snapshot memory.



**Figure 4.4.1. Snapshot memory format.**

As for the control, the taking of a snapshot can be enabled and disabled. When it is enabled, the snapshot memory starts being filled entries, and the current address pointer is kept in a separate register. There is also a mask register for the snapshot memory, in which each TTC signal can be masked. Masking the TTC signal means that the changes of that signal are not stored in the snapshot memory.

#### 4.5. TTC encoder

The generation and encoding of TTC B-channel commands in the ALTI module has been implemented in the same way as in the TTCvi, which was discussed in Chapter 2. Also, the format of the B-channel commands is the same and has already been discussed. The same is true for the priority scheme of B-channel commands.

There are five different modes of sending commands from each of the BGO channel FIFOs:

- 1) "SYNCHRONOUS\_SINGLE\_BGO\_SIGNAL" - send once at the end of the inhibit if the corresponding BGO signal occurred
- 2) "SYNCHRONOUS\_REPETITIVE" - send once each orbit at the end of the inhibit, regardless of the corresponding BGO signal
- 3) "ASYNCHRONOUS\_BGO\_SIGNAL" - send when the BGO signal is received
- 4) "ASYNCHRONOUS\_VME\_ON\_TRIGGER" - send when the corresponding VME register is written
- 5) "ASYNCHRONOUS\_VME\_WHEN\_NOT\_EMPTY" - send whenever the corresponding FIFO is not empty

## 4.6. TTC decoder

The TTC decoder memory consists of two 1Mx32 memory blocks, which can logically be viewed as a single 1Mx64 block. We call these blocks "command" and "timestamp", because of their content. The format of the decoder memory is shown on Figure 4.6.1. Whenever a command or a trigger gets decoded from the TTC stream, a single 64-bit entry with such format gets written into the TTC decoder memory.

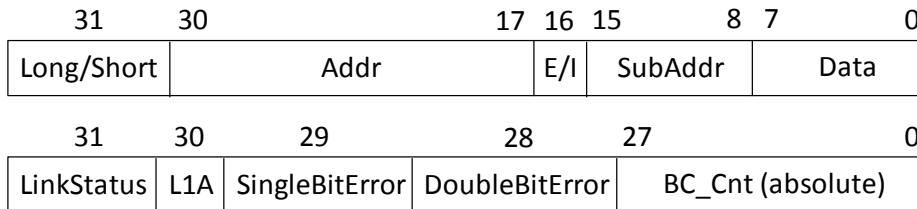


Figure 4.6.1. TTC spy memory format: "command" and "timestamp".

The control of the TTC decoder is very similar to the control of the snapshot memory. It is possible to enable and disable the decoding, and an address pointer is kept to indicate the range of valid entries that were written.

## 4.7. I2C master core

A simple, wishbone-compatible I2C master core available on OpenCores [31] has been used in order to communicate with the devices on the ALTI I2C network.

## 4.8. 1-Wire master core

A 1-Wire master core provided by Dallas Semiconductor (acquired by Maxim Integrated) [32] has been used to implement the 1-Wire protocol. This protocol is used to communicate with the 1-Wire chip, which gives each ALTI module a unique identifier.

## 4.9. Busy and calibration request routing

Unlike the forward-going TTC signals, BUSY and CALREQ[2..0] are not routed through the cross-point switches. That is why the routing logic for these signals was done in the firmware.

Functional block diagram of this routing logic for the BUSY signal is shown on Figure 4.9.1.

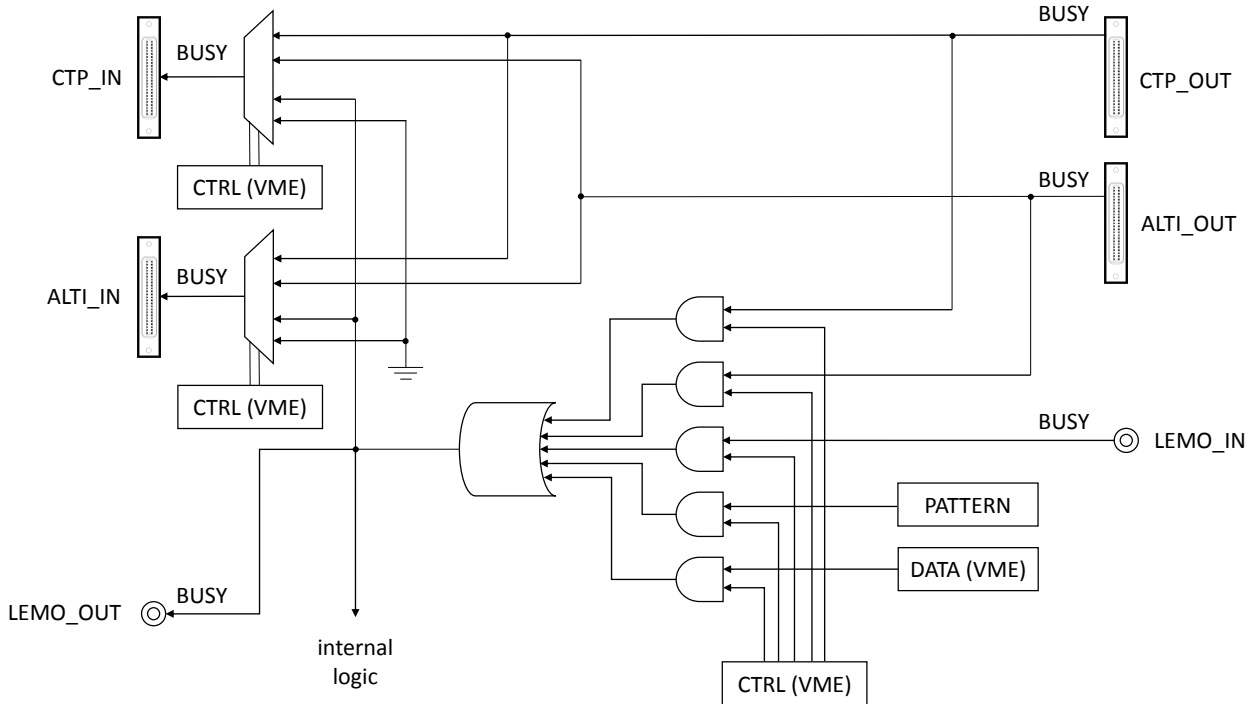
The local BUSY signal, used in the ALTI internal logic, is a logical OR of five possible BUSY input sources. All of the busy sources can be independently masked. This flexible masking allows the ALTI to logically add together BUSY inputs from multiple sources, which is used when ALTI plays a role of a master in two parallel TTC daisy chains. Sources for the BUSY signal are the following:

- 1) BUSY input from the CTP\_OUT connector
- 2) BUSY input from the ALTI\_OUT connector
- 3) BUSY input from the LEMO\_IN connector
- 4) BUSY from the pattern generator
- 5) BUSY from the internal register



The local BUSY signal is routed to the BUSY LEMO output for monitoring purposes. For sending the BUSY signal upstream to the CTP\_IN and ALTI\_IN LVDS-LINK connectors, one of the four sources can be chosen with a VME-controllable multiplexor:

- 1) BUSY input from the CTP\_OUT connector
- 2) BUSY input from the ALTI\_OUT connector
- 3) Local BUSY signal
- 4) Inactive BUSY signal



**Figure 4.9.1. Functional block diagram of the BUSY signal routing in the FPGA logic.**

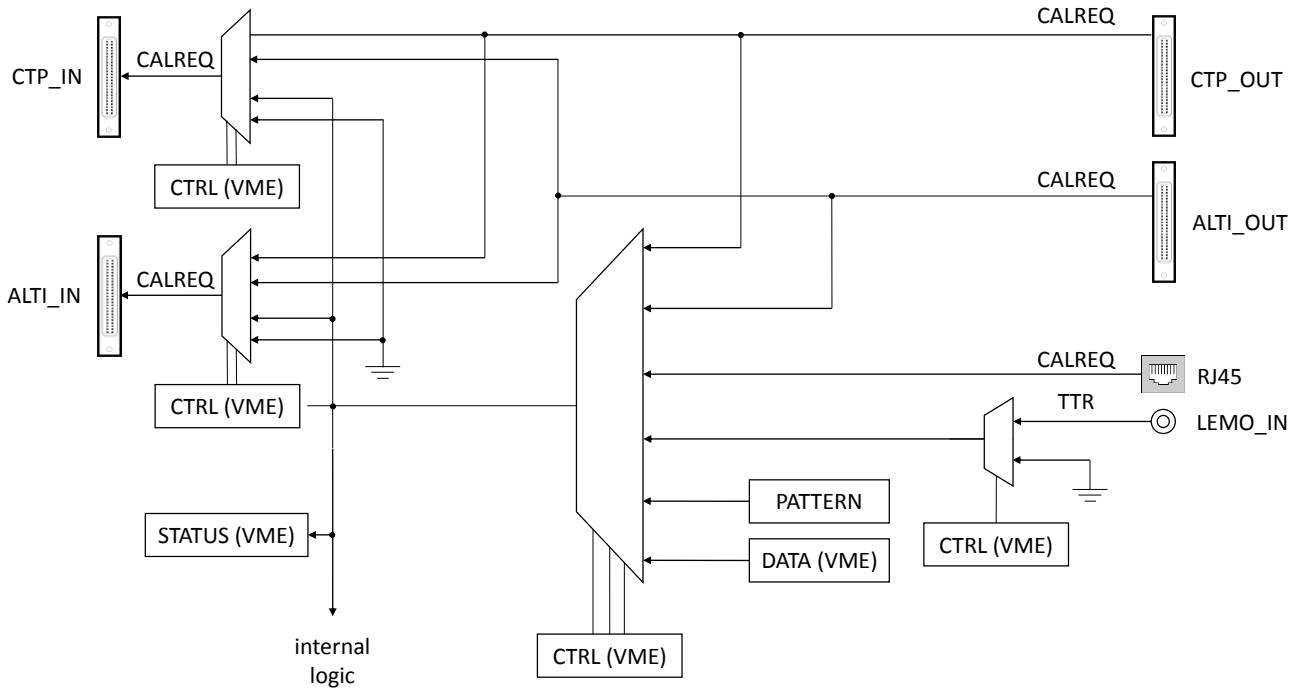
The functional block diagram of the routing logic for the CALREQ[2..0] signals is shown on Figure 4.9.2. Each of the three CALREQ signals can be routed independently from the other two.

Local CALREQ signal, used in the ALTI internal logic, can be sourced from any of the six possible CALREQ input sources. Compared to the BUSY signal sources, there is one additional source because the TTR[1..3] LEMO inputs are multiplexed with the CALREQ[0..2] in order to allow more input flexibility. Thus, the sources for the CALREQ[2..0] signals are the following:

- 1) CALREQ input from the CTP\_OUT connector
- 2) CALREQ input from the ALTI\_OUT connector
- 3) CALREQ input from the RJ45 connector
- 4) CALREQ input from the corresponding TTR LEMO\_IN connector
- 5) CALREQ from the pattern generator
- 6) CALREQ from the internal register

For sending the CALREQ signal upstream to the CTP\_IN and ALTI\_IN LVDS-LINK connectors, one of the four sources can be chosen with a VME-controllable multiplexor:

- 1) CALREQ input from the CTP\_OUT connector
- 2) CALREQ input from the ALTI\_OUT connector
- 3) Local CALREQ signal
- 4) Inactive CALREQ signal



**Figure 4.9.2. Functional block diagram of the CALREQ signal routing in the FPGA logic.**

# 5. ALTI SOFTWARE

This chapter is about the ALTI software, which is an essential part of the ALTI development and is the one that the author has contributed to most. Low-level software that has been developed for the ALTI modules makes the board "alive" by allowing an easy access to all of the hardware and firmware that is available. Without previously developing this software, it would be impossible to test the module and measure its performances in an efficient and reproducible way.

First, in Section 5.1, the ATLAS Trigger and Data Acquisition (TDAQ) infrastructure on which the ALTI software relies on is briefly described. Then, the low-level API for the ALTI is discussed in Section 5.2. In Section 5.3, the menu program which exercises the low-level API is discussed. This is followed by the discussion of the ALTI configuration object in Section 5.4. Test programs are independently discussed in subsections of Section 5.5.

## 5.1. ATLAS TDAQ

The ATLAS TDAQ system provides the software infrastructure for Level-1 Trigger, Data Acquisition (DAQ) and HLT systems. Software packages for TDAQ are maintained on a private GitLab server hosted in CERN. The TDAQ team provides the necessary build tools, which are based on CMake. They also take care of tagging the software packages and making sure the software packages with dependencies are compatible in every new TDAQ release.

Software packages for the Level-1 Trigger and the TTC modules are also a part of ATLAS TDAQ. These packages include the low-level software for control, configuration and monitoring of the modules. High-level run control application software that is built on top of the low-level APIs is also included in the ATLAS TDAQ.

VME-addressable TTC legacy modules (LTPI, LTP and TTCvi) and the other modules all have a similar low-level software organization. They provide an API for accessing all the functionalities of a given module in the form of public methods of the module base class. All of these methods are then exercised in a menu program, which guides the user to interactively access the module. Besides the menu program, there are usually various test programs accompanying each module. The ALTI module is no exception and its low-level software is organized in a similar way.

There are several software packages that the ALTI package depends on. Some of them are a part of the TDAQ ReadOut Driver Crate DAQ (RCD) and some of them are specific to the Level-1 Central Trigger (L1CT):

### 1) ATLAS TDAQ RCD

- `vme_rcc`: VMEbus driver
- `RCDVme`: wrapper for the `vme_rcc` driver
- `RCDBitString`: manipulation of bit arrays of arbitrary size
- `RCDMenu`: package for interactive menus that support nesting
- `RCDUtilities`: common utilities like print out and error-handling

### 2) ATLAS L1CT

- `L1CTHardwareCompiler`: translation of an `.xml` description of a module to low-level VME read/write methods
- `I2C`: driver for the I2C master core
- `DS1WM`: driver for the 1-Wire master core

## 5.2. Low-level API

Low-level software has been developed in order to provide the access to the ALTI module and all of the functionalities of hardware and firmware. It provides programmable support for control, configuration and monitoring of the ALTI module in terms of its registers, memories, and FIFO buffers. The software was developed for the single-board computer of the ATLAS readout driver crate, which uses a library and a driver to communicate with the ALTI module using the VMEbus.

The first thing that was done on the ALTI software is the description of the module with an xml file called *alti.xml*. This file gives a basic description of the module in terms of specific bit-assignments and addresses of various registers, FIFOs and RAM memories within the ALTI module. In this sense, the *alti.xml* closely resembles the firmware. The file format allows for dividing the module address space into blocks with logically grouped functionalities. A glimpse of the *alti.xml* file is given on figures 5.2.1 and 5.2.2. These figures show how the snapshot memory bitstring and the I2C core block were defined, respectively.

A software package called the L1CT hardware compiler then generates low-level VME read/write functions, such that the bit fields are addressed by their name and the specific bit-assignments are abstracted. This allows for better maintainability of the code, because this is the only place where changes need to be made if the register addresses or their bit-assignments are changed as the firmware development progresses. As a result of running the L1CT hardware compiler, a class called *ALTI* is automatically generated, providing the low-level VME read/write functions of the bit fields.

Built on top of that is the *AltiModule* base class, which provides more user-friendly API for control, configuration and monitoring of the ALTI module. This class incorporates an object of low-level class *ALTI* as its private member and uses its automatically generated public methods. However, *AltiModule* methods are not just a wrapper around the *ALTI* methods, since the API also includes methods that must perform a certain sequence of operations in order to exercise some functionality implemented in the firmware. These sequences or sets of operations are put together for convenience. The API of the base class *AltiModule* is also the place where the meaning of variables and functions are changed from a hardware point of view to a user point of view.

```
<bitstring name="SNAPSHOT">
  <field name="BC_TimeStamp"          mask="0xffe00000" form="NUMBER"/>
  <field name="ORB"                   mask="0x00100000"/>
  <field name="BUSY"                   mask="0x00080000"/>
  <field name="CalibrationRequest"     mask="0x00070000" form="NUMBER"/>
  <field name="TriggerTypeWord"       mask="0x0000ff00" form="NUMBER"/>
  <field name="BGo3"                   mask="0x00000080"/>
  <field name="BGo2"                   mask="0x00000040"/>
  <field name="BGo1"                   mask="0x00000020"/>
  <field name="BGo0"                   mask="0x00000010"/>
  <field name="TestTrigger3"           mask="0x00000008"/>
  <field name="TestTrigger2"           mask="0x00000004"/>
  <field name="TestTrigger1"           mask="0x00000002"/>
  <field name="L1A"                    mask="0x00000001"/>
</bitstring>
```

Figure 5.2.1. A section of the *alti.xml* file containing a bitstring description for the snapshot memory entry.

```

<block name="I2C" addr="0x00009080">

  <register name="PrescaleLow"          addr="0x00000000"/>
  <register name="PrescaleHigh"        addr="0x00000004"/>
  <register name="Control"              addr="0x00000008">
    <field name="Enable"               mask="0x00000080">
      <value name="Enabled"            data="0x00000080"/>
      <value name="Disabled"          data="0x00000000"/>
    </field>
    <field name="InterruptEnable"      mask="0x00000040">
      <value name="Enabled"            data="0x00000040"/>
      <value name="Disabled"          data="0x00000000"/>
    </field>
  </register>
  <register name="Transmit"            addr="0x0000000c" modf="W"/>
  <register name="Receive"            addr="0x0000000c" modf="R"/>
  <register name="Command"            addr="0x00000010" modf="W">
    <field name="Start"                mask="0x00000080"/>
    <field name="Stop"                 mask="0x00000040"/>
    <field name="Read"                 mask="0x00000020"/>
    <field name="Write"                mask="0x00000010"/>
    <field name="Acknowledge"          mask="0x00000008"/>
    <field name="IntAcknowledge"       mask="0x00000001"/>
  </register>
  <register name="Status"             addr="0x00000010" modf="R">
    <field name="RxAck"                mask="0x00000080"/>
    <field name="Busy"                 mask="0x00000040"/>
    <field name="ArbLost"              mask="0x00000020"/>
    <field name="TIP"                  mask="0x00000002"/>
    <field name="IntFlag"              mask="0x00000001"/>
  </register>
</block>

```

Figure 5.2.2. A section of the *alti.xml* file containing the I2C block.

The API is logically grouped into blocks of the module functionalities. Each block is distinguished by a 3-letter abbreviation, which each also serve as prefixes for the names of method names. Base class methods are thus grouped in the following blocks:

- **CSR**: VME64x CR/CSR space
- **CFG**: AltiConfiguration object, read and write, default setup and check of setup
- **CLK**: PLL and jitter cleaner configuration
- **SIG**: signal settings - configuration of cross-point switches, equalizers and input synchronization/shaping
- **BSY**: selection and routing of BUSY signal
- **CRQ**: selection and routing of Calibration Request signals
- **PAT**: pattern generation memory
- **SNP**: snapshot memory
- **ENC**: TTC encoder control
- **DEC**: TTC decoder control
- **CNT**: ALTI counters - BC, ORB, L1A, Test Triggers and BGOs
- **MON**: hardware monitoring - voltages/temperatures readout
- **I2C**: access each individual device in the I2C network

To give an example of the API methods, the list of methods associated with the functionality blocks "SIG", "PAT" and "SNP" are shown on Figure 5.2.3, Figure 5.2.4 and Figure 5.2.5, respectively.

```

// public SIG methods
// Cross-point switches (SWX)
int SIGSwitchConfigRead(const SIGNAL, const SIGNAL_DESTINATION, SIGNAL_SOURCE &); // 1 signal, 1 output
int SIGSwitchConfigRead(const SIGNAL, std::vector<SIGNAL_SOURCE> &); // 1 signal, all outputs
int SIGSwitchConfigWrite(const SIGNAL, const SIGNAL_DESTINATION, const SIGNAL_SOURCE); // 1 signal, 1 output
int SIGSwitchConfigWrite(const SIGNAL_DESTINATION, const SIGNAL_SOURCE); // all signals, 1 output
int SIGSwitchConfigWrite(const SIGNAL, const SIGNAL_SOURCE); // 1 signal, all outputs
int SIGSwitchConfigWrite(const SIGNAL_SOURCE); // all signals, all outputs
int SIGSwitchConfigWrite(const ALTI_MODE);
int SIGSwitchPredefinedModeWrite(const PREDEFINED_SWITCH_MODE);
void SIGSwitchPredefinedModeLegendPrint(std::ostream & = std::cout);
// Equalizers (EQZ)
int SIGEqualizersConfigRead(EQUALIZER_CONFIG &, EQUALIZER_CONFIG &);
int SIGEqualizersConfigWrite(const EQUALIZER_CONFIG, const EQUALIZER_CONFIG);
// Input synchronization & shaping
// sync
int SIGInputSyncEnableRead(const ASYNC_INPUT_SIGNAL, bool &, unsigned int &);
int SIGInputSyncEnableWrite(const ASYNC_INPUT_SIGNAL, const bool, const unsigned int);
int SIGInputSyncEnableRead(std::vector<bool> &, std::vector<unsigned int> &); // all
int SIGInputSyncEnableWrite(const std::vector<bool> &, const std::vector<unsigned int> &); // all
int SIGInputSyncEnableWrite(const bool, const unsigned int); // all
int SIGInputSyncHistogramsRead(const ASYNC_INPUT_SIGNAL, std::vector<unsigned int> &, std::vector<unsigned int> &);
int SIGInputSyncHistogramsReset();
// shape
int SIGInputShapeEnableRead(const ASYNC_INPUT_SIGNAL, bool &);
int SIGInputShapeEnableWrite(const ASYNC_INPUT_SIGNAL, const bool);
int SIGInputShapeEnableRead(std::vector<bool> &); // all
int SIGInputShapeEnableWrite(const std::vector<bool> &); // all
int SIGInputShapeEnableWrite(const bool); // all
// BGO LEMO outputs mux
int SIGMuxBGOOutputLEMORead(bool &, bool &); // both
int SIGMuxBGOOutputLEMOWrite(const bool, const bool); // both

```

Figure 5.2.3. The list of low-level API methods of block "SIG".

```

// public PAT methods
int PATGenerationEnableRead(const SIGNAL_PG, bool &);
int PATGenerationEnableWrite(const SIGNAL_PG, const bool);
int PATGenerationEnableRead(std::vector<bool> &); // all
int PATGenerationEnableWrite(const std::vector<bool> &); // all
int PATGenerationEnableWrite(const bool); // all
int PATGenerationRepeatRead(bool &);
int PATGenerationRepeatWrite(const bool);
int PATGenerationStartAddressRead(unsigned int &);
int PATGenerationStartAddressWrite(const unsigned int);
int PATGenerationStopAddressRead(unsigned int &);
int PATGenerationStopAddressWrite(const unsigned int);
int PATRead(RCD::CMEMSegment *); // read pattern generation memory
int PATReadFile(RCD::CMEMSegment *, std::ifstream &); // read pattern generation memory from file

```

Figure 5.2.4. The list of low-level API methods of block "PAT".

```

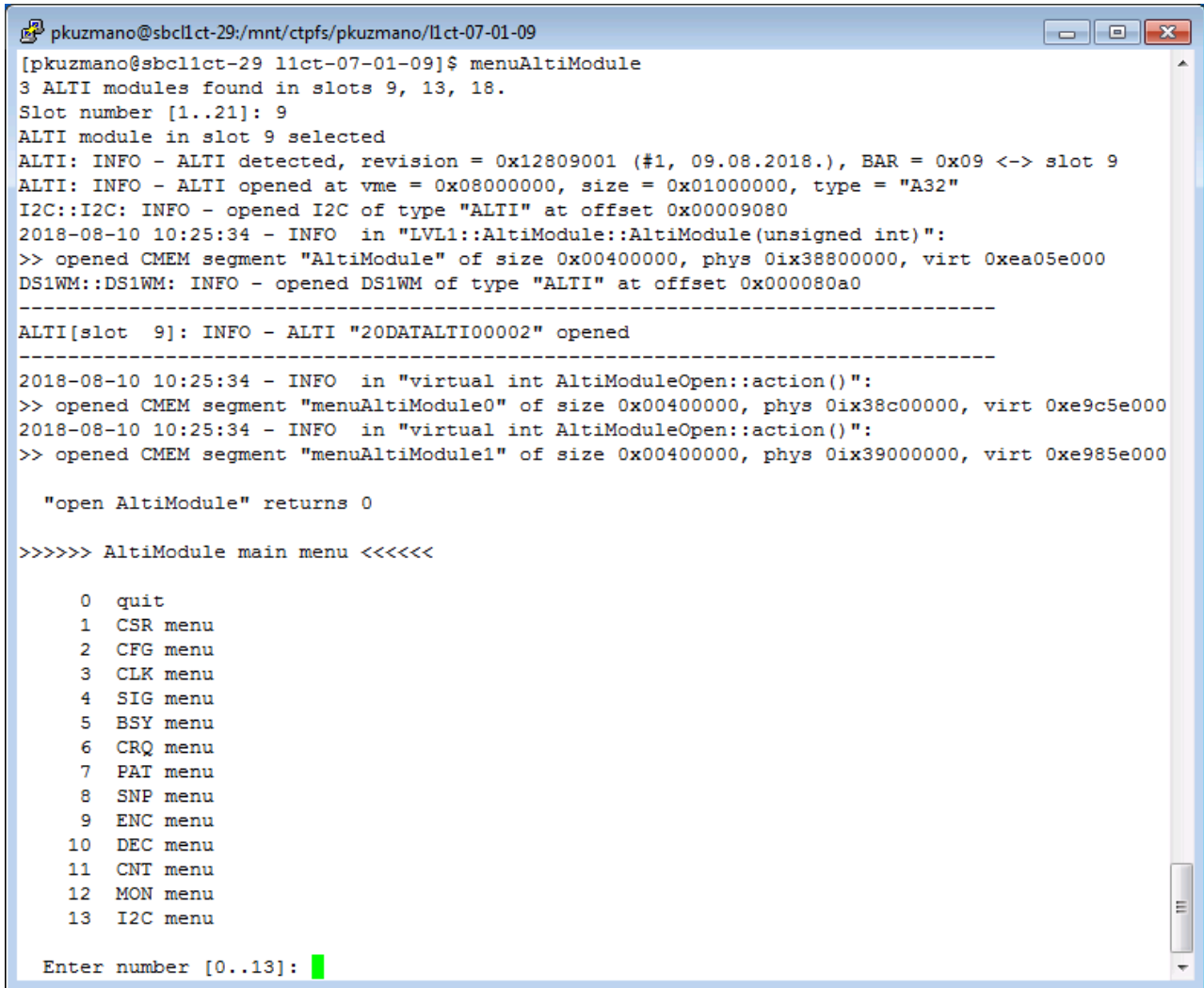
// public SNP methods
int SNPEnableRead(bool &);
int SNPEnableWrite(const bool);
int SNPCurrentAddressRead(unsigned int &);
int SNPMaskRead(unsigned int &);
int SNPMaskWrite(const unsigned int);
int SNPMaskRead(const SIGNAL, bool &); // 1 signal
int SNPMaskWrite(const SIGNAL, const bool); // 1 signal
int SNPMaskRead(std::vector<bool> &); // all signals
int SNPMaskWrite(const bool); // all signals
int SNPRead(RCD::CMEMSegment *); // read snapshot memory
int SNPWriteFile(RCD::CMEMSegment *, int, std::ofstream &); // write snapshot memory into file

```

Figure 5.2.5. The list of low-level API methods of block "SNP".

### 5.3. Menu program

This is an interactive program, which allows the user to access the module and configure it at will. On Figure 5.3.1, the main menu of the *menuAltiModule* program is shown. With the use of the *menuAltiModule* program, the user can execute all of the methods from the base class API, in order to set the module up to a desired state. In this way, low-level API and the menu program proved quite useful for hardware and firmware evaluation. Functionalities of the module are logically grouped into the sub-menus of the menu program, in the same way as the API is partitioned.



```
pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/11ct-07-01-09
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$ menuAltiModule
3 ALTI modules found in slots 9, 13, 18.
Slot number [1..21]: 9
ALTI module in slot 9 selected
ALTI: INFO - ALTI detected, revision = 0x12809001 (#1, 09.08.2018.), BAR = 0x09 <-> slot 9
ALTI: INFO - ALTI opened at vme = 0x08000000, size = 0x01000000, type = "A32"
I2C::I2C: INFO - opened I2C of type "ALTI" at offset 0x00009080
2018-08-10 10:25:34 - INFO in "I2C::I2C::I2C(I2CDevice *, unsigned int)":
>> opened CMEM segment "AltiModule" of size 0x00400000, phys 0ix38800000, virt 0xea05e000
DS1WM::DS1WM: INFO - opened DS1WM of type "ALTI" at offset 0x000080a0
-----
ALTI[slot 9]: INFO - ALTI "20DATAALTI00002" opened
-----
2018-08-10 10:25:34 - INFO in "virtual int AltiModuleOpen::action()":
>> opened CMEM segment "menuAltiModule0" of size 0x00400000, phys 0ix38c00000, virt 0xe9c5e000
2018-08-10 10:25:34 - INFO in "virtual int AltiModuleOpen::action()":
>> opened CMEM segment "menuAltiModule1" of size 0x00400000, phys 0ix39000000, virt 0xe985e000

"open AltiModule" returns 0

>>>>> AltiModule main menu <<<<<<

    0  quit
    1  CSR menu
    2  CFG menu
    3  CLK menu
    4  SIG menu
    5  BSY menu
    6  CRQ menu
    7  PAT menu
    8  SNP menu
    9  ENC menu
   10  DEC menu
   11  CNT menu
   12  MON menu
   13  I2C menu

Enter number [0..13]: █
```

Figure 5.3.1. Main menu of the ALTI menu program.

### 5.4. Configuration object

The ALTI module has many registers and memories. In fact, its current address space takes up 16MB. Thus, the number of parameters that define a state of the module is also quite large. It was therefore decided to design a convenient way of configuring the whole module at once, without having to manually set each parameter using the menu program.

A class for the configuration of the ALTI module, called *AltiConfiguration*, allows the user to fully configure the ALTI module to a known state. The *AltiConfiguration* object is associated with a specific file format which contains a list of key/value pairs for all the parameters, and provides a complete configuration of the ALTI module. Both configuring the module and reading its current state is possible using this configuration class. Input and output file formats are the same, so the configuration of some module with a file that was read back as the state of some other module is also possible. The list of parameters in the configuration file is partitioned into blocks of functionalities, in the same way as the API. Each block can be independently included and excluded from being configured using its "CONFIG" parameter in the configuration file. Thus, a partial configuration of the module is also possible. There is a dedicated test program which allows the user to write or read the configuration from the command line, which will be discussed in Subsection 5.5.2. A glimpse of a particular ALTI configuration file is shown on Figure 5.4.1.

```
#####
# CONFIG|NOCONFIG
CLK_CONFIG                = CONFIG
# PRIMARY_FXO|SECONDARY_EXT
CLK_SOURCE                 = PRIMARY_FXO
# PRIMARY_FXO|SECONDARY_EXT
CLK_JC_SOURCE             = SECONDARY_EXT
#####
# CONFIG|NOCONFIG
SIG_CONFIG                = CONFIG
# BGO0|BGO2, BGO1|BGO3
SIG_BGO_FP                = BGO0, BGO1
# <CTP_OUT>, <ALTI_OUT>, <LEMO_OUT>, <TO_FPGA>
# <CTP_OUT> = CTP_IN|ALTI_IN|LEMO_IN|FROM_FPGA
# <ALTI_OUT> = CTP_IN|ALTI_IN|LEMO_IN|FROM_FPGA
# <LEMO_OUT> = CTP_IN|ALTI_IN|LEMO_IN|FROM_FPGA
# <TO_FPGA> = CTP_IN|ALTI_IN|LEMO_IN|FROM_FPGA
SIG_SWX_BC                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_ORB                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_L1A                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTR1                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTR2                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTR3                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTYP0               = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTYP1               = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTYP2               = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTYP3               = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTYP4               = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTYP5               = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTYP6               = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_TTYP7               = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_BGO0                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_BGO1                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_BGO2                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
SIG_SWX_BGO3                = FROM_FPGA, CTP_IN, FROM_FPGA, CTP_IN
# SHORT_CABLE|LONG_CABLE
SIG_EQZ_CTP_IN              = SHORT_CABLE
SIG_EQZ_ALTI_IN             = LONG_CABLE
# SYNC: ENABLED|DISABLED, 000DEG|090DEG|180DEG|270DEG
# SHAPE: ENABLED|DISABLED
SIG_IO_SYNC_SWX_ORB         = ENABLED, 270DEG
SIG_IO_SHAPE_SWX_ORB        = ENABLED
SIG_IO_SYNC_SWX_L1A         = ENABLED, 270DEG
SIG_IO_SHAPE_SWX_L1A        = ENABLED
# BGO0..3
SIG_IO_SYNC_SWX_BGO         = ENABLED, 000DEG, ENABLED, 000DEG, ENABLED, 270DEG, ENABLED, 180DEG
SIG_IO_SHAPE_SWX_BGO        = ENABLED, ENABLED, ENABLED, DISABLED
```

Figure 5.4.1. A section of the ALTI configuration file containing parameters in a form of key/value pairs.



## 5.5. Test programs

Aside from the menu program, various test programs have been developed for testing and diagnostics of the ALTI module. They are used for: automated specific tests (snapshot memory, and the TTC decoder), initializing the module with a configuration given in a form of a file, remote firmware update, etc. Each of them will be described in the following subsections.

### 5.5.1. *testAltVME*

This test program is used to test VMEbus read/write transfers on RAM memories and FIFOs, as well as the internal FPGA registers that are safe to write to (not used for some critical control).

Testing the VME interface is done by writing some data to a particular chunk of memory, and making sure that the same data is then properly read back. The user specifies the base address of the test area, a number of 32-bit words to be tested and a comparison mask. Also, both single and block cycles are supported. Types of tests that are included are the following: simple fixed value writes/readouts, writing incrementing/decrementing data, writing walking "1" bits to successive addresses. The latter two types of tests are useful for checking the address and data lines, since the value being written differs according to the address.

On Figure 5.5.1 one can see the full list of the program parameters. A common use of the *testAltVME* program is shown on Figure 5.5.2.

```
pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltVME -h
-----
testAltVME <OPTIONS>:
-----
-b <hex> => ALTI VME base address                (def = 0x08000000)
-o <hex> => offset within ALTI VME space          (def = 0x00c00000)
-s <hex> => size of the window to be tested (in 32b words) (def = 0x00000001)
-m <hex> => bitmask used for checking each word    (def = 0xffffffff)
-n <hex> => number of checks                       (def = 0x00000001)
-t <hex> => type of test                          (def = 0x00000000)
    0 => simple 0, 5, a, f test
    1 => counter test
    2 => walking ones test
-B      => use block transfer                      (def = NO)
-h      => Print this help
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$
```

Figure 5.5.1. *testAltVME* program help with the full list of parameters.

```
pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltVME -b 0x08000000 -o 0x00c00000 -s 0x00100000 -m 0xffffffff -n 5 -B -t 0
2018-08-10 10:37:26 - INFO in "int main(int, char**)":
>> opened CMEM segment "testAltVME" of size 0x00400000, phys 0ix38800000, virt 0xc0205000
Memory 0x08c00000..0x08bffffc test 0 OK
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltVME -b 0x08000000 -o 0x00800000 -s 0x00100000 -m 0xffffffff -n 5 -B -t 1
2018-08-10 10:37:34 - INFO in "int main(int, char**)":
>> opened CMEM segment "testAltVME" of size 0x00400000, phys 0ix38800000, virt 0x1f2ce000
Memory 0x08800000..0x087ffffc test 1 OK
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltVME -b 0x08000000 -o 0x00400000 -s 0x00100000 -m 0xffffffff -n 5 -B -t 2
2018-08-10 10:37:42 - INFO in "int main(int, char**)":
>> opened CMEM segment "testAltVME" of size 0x00400000, phys 0ix38800000, virt 0x50e16000
Memory 0x08400000..0x083ffffc test 2 OK
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$
```

Figure 5.5.2. Typical use of the *testAltVME* program.

### 5.5.2. *testAltiInitial*

This test program is used for module initialization and configuration.

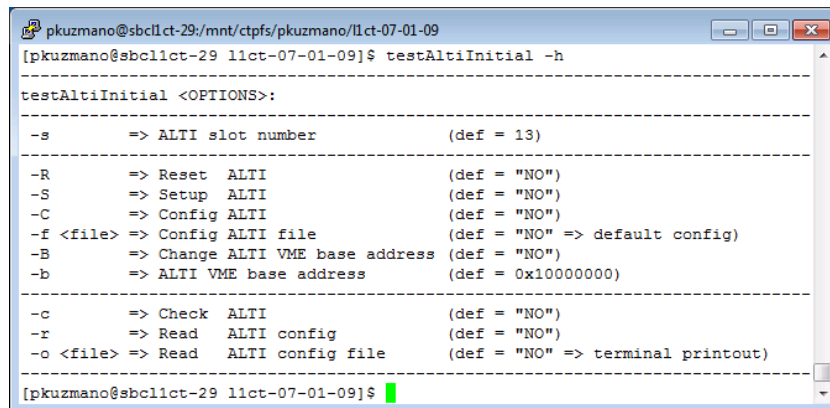
When the ALTI module is powered on, this program is used for basic initialization and setup. That means the user can change the VME base address of the module, if the default geographical addressing (according to the slot number) is not desired. For example, it could be that the base address given by the geographical addressing is not the range of any of the static master mappings defined by the SBC. On the other hand, basic setup includes the following: setting the jitter cleaner up with a default configuration, setting the recommended clock prescaler value of the I2C master core, etc. It is recommended to perform the initialization and setup tasks once upon power up before using the module.

Configuring the module is done with an *AltiConfiguration* object, corresponding to an input file format discussed in Section 5.4. This file contains all the configurable parameters of the module and it grew as each new feature got added to the firmware, and afterwards the low-level software. The user also has an ability to read current configuration into a file, or dump it onto the standard output screen.

Another feature of this test program is the ability to perform a basic check of the setup of the ALTI. In this way, the user can check if the module is operational, i.e. if the PLL is locked, the I2C prescaler and the jitter cleaner are properly set up, and so on.

By using the *testAltiInitial* program, it is easy to quickly configure multiple ALTI modules for automated tests, not having to use the menu program. Various configuration files for common ALTI modes of operation are available: "Master" (Pattern generator), "CTP slave", "ALTI slave", "LEMO slave", TTC encoder/decoder, etc.

On Figure 5.5.3 one can see the full list of the program parameters. Two common uses of the *testAltiInitial* program are shown on Figure 5.5.4.



```
pkuzmano@sbc11ct-29/mnt/ctpfs/pkuzmano/l1ct-07-01-09
[pkuzmano@sbc11ct-29 11ct-07-01-09]$ testAltiInitial -h
-----
testAltiInitial <OPTIONS>:
-----
-s      => ALTI slot number          (def = 13)
-----
-R      => Reset  ALTI                (def = "NO")
-S      => Setup  ALTI                (def = "NO")
-C      => Config ALTI                (def = "NO")
-f <file> => Config ALTI file         (def = "NO" => default config)
-B      => Change ALTI VME base address (def = "NO")
-b      => ALTI VME base address      (def = 0x10000000)
-----
-c      => Check  ALTI                (def = "NO")
-r      => Read   ALTI config          (def = "NO")
-o <file> => Read   ALTI config file    (def = "NO" => terminal printout)
-----
[pkuzmano@sbc11ct-29 11ct-07-01-09]$
```

Figure 5.5.3. *testAltiInitial* program help with the full list of parameters

```

pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltiInitial -s 13 -B -b 0x10000000 -S -c
2018-08-10 11:59:38 - INFO in "daq::tmgr::TestResult LVL1::TestAltiInitial::init()"
-----
testAltiInitial:
-----
ALTI slot number          = 13
ALTI VME base address     = 0x10000000
-----
Reset ALTI                = "NO"
Setup ALTI                = "YES"
Config ALTI               = "NO"
Change ALTI VME base address = "YES", 0x10000000
-----
Check ALTI                = "YES"
Read ALTI config          = "NO"
-----
ALTI: INFO - ALTI detected, BAR = 0x0d <-> slot 13
ALTI: INFO - ALTI detected, revision = 0x12809001 (#1, 09.08.2018.), BAR = 0x0d <-> slot 13
ALTI: INFO - ALTI opened at vme = 0x10000000, size = 0x01000000, type = "A32"
I2C::I2C: INFO - opened I2C of type "ALTI" at offset 0x00009080
2018-08-10 11:59:38 - INFO in "LVL1::AltiModule::AltiModule(unsigned int)":
>> opened CMEM segment "AltiModule" of size 0x00400000, phys 0ix37c00000, virt 0x7fbc000
DS1WM::DS1WM: INFO - opened DS1WM of type "ALTI" at offset 0x000080a0
-----
ALTI[slot 13]: INFO - ALTI "20DATAALTI00005" opened
-----
>> TestAltiInitial::init ..... [PASS]
-----
wrote 326 (addr,data) lines to CLK jitter cleaner
2018-08-10 11:59:40 - INFO in "int LVL1::AltiModule::AltiCheck()":
>> ALTI: INFO - CLK mux = "SETUP"
2018-08-10 11:59:40 - INFO in "int LVL1::AltiModule::AltiCheck()":
>> ALTI: INFO - I2C core = "SETUP"
2018-08-10 11:59:40 - INFO in "int LVL1::AltiModule::AltiCheck()":
>> ALTI: INFO - PLL = "LOCKED"
2018-08-10 11:59:40 - INFO in "int LVL1::AltiModule::AltiCheck()":
>> ALTI: INFO - jitter cleaner design = "ALTI_001" (DEFAULT)
>> TestAltiInitial::exec ..... [PASS]
-----
>> TestAltiInitial::test ..... [PASS]
-----
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$

```

(a)

```

pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltiInitial -s 13 -C -f ALTI/data/AltiModule_CTP_Slave_cfg.dat
2018-08-10 12:00:53 - INFO in "daq::tmgr::TestResult LVL1::TestAltiInitial::init()"
-----
testAltiInitial:
-----
ALTI slot number          = 13
ALTI VME base address     = 0x10000000
-----
Reset ALTI                = "NO"
Setup ALTI                = "NO"
Config ALTI               = "YES", "ALTI/data/AltiModule_CTP_Slave_cfg.dat"
Change ALTI VME base address = "NO"
-----
Check ALTI                = "NO"
Read ALTI config          = "NO"
-----
ALTI: INFO - ALTI detected, revision = 0x12809001 (#1, 09.08.2018.), BAR = 0x0d <-> slot 13
ALTI: INFO - ALTI opened at vme = 0x10000000, size = 0x01000000, type = "A32"
I2C::I2C: INFO - opened I2C of type "ALTI" at offset 0x00009080
2018-08-10 12:00:53 - INFO in "LVL1::AltiModule::AltiModule(unsigned int)":
>> opened CMEM segment "AltiModule" of size 0x00400000, phys 0ix37c00000, virt 0xcee60000
DS1WM::DS1WM: INFO - opened DS1WM of type "ALTI" at offset 0x000080a0
-----
ALTI[slot 13]: INFO - ALTI "20DATAALTI00005" opened
-----
>> TestAltiInitial::init ..... [PASS]
-----
2018-08-10 12:00:53 - INFO in "int LVL1::AltiConfiguration::read(const string&)":
>> read AltiConfiguration from file "ALTI/data/AltiModule_CTP_Slave_cfg.dat"
>> TestAltiInitial::exec ..... [PASS]
-----
>> TestAltiInitial::test ..... [PASS]
-----
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$

```

(b)

Figure 5.5.4. Typical use of the *testAltiInitial* program: (a) base address initialization, basic setup and check, (b) module configuration as a "CTP slave".

### 5.5.3. testAltiQuickBoot

This test program is used to remotely update the FPGA firmware through VME, without using Xilinx tools. Not having to use the USB programmer makes it more convenient to update the firmware of modules installed in the VME crate.

The remote firmware update is based on the QuickBoot mechanism, described in detail in one of the application notes provided by Xilinx [33]. Basically, the initial bitstream stored in the configuration memory contains "golden" and "update" images, which are identical at the beginning. This initial bitstream contains these two exact copies of a stable firmware release, and is flashed to the configuration memory once using the Xilinx tools. Afterwards, remote firmware update is done only by overwriting the "update" image area, using the previously mentioned QuickBoot mechanism.

Xilinx provides a QuickBoot FlashProgrammer core in VHDL. They also provide a Perl script for converting a standard mcs image file (result of firmware compilation in Xilinx Vivado) to two of them: initial and update mcs files. Update mcs file is then converted to a binary file using open-source solution SRecord [34], which handles manipulations of different EPROM load file standards. The resulting binary file contains the sequence of 32-bit data of the update image which needs to be sent to the flash programmer. Interfacing to the flash programmer is done via single FIFO memory in the FPGA.

The software takes care of enabling the flash programmer and filling this FIFO with the update image data, one 32-bit word at the time. Successful termination or error in the update process is also reported by the flash programmer and caught by the *testAltiQuickBoot* program. In addition to the update feature, this test program allows verifying the existing update in the image by calculating its CRC32 checksum, which is also written in the last entry in the update image area of the configuration memory.

QuickBoot mechanism is summarized on Figure 5.5.5. The full list of *testAltiQuickBoot* program parameters can be seen on Figure 5.5.6, while the common use of this program is shown on Figure 5.5.7.

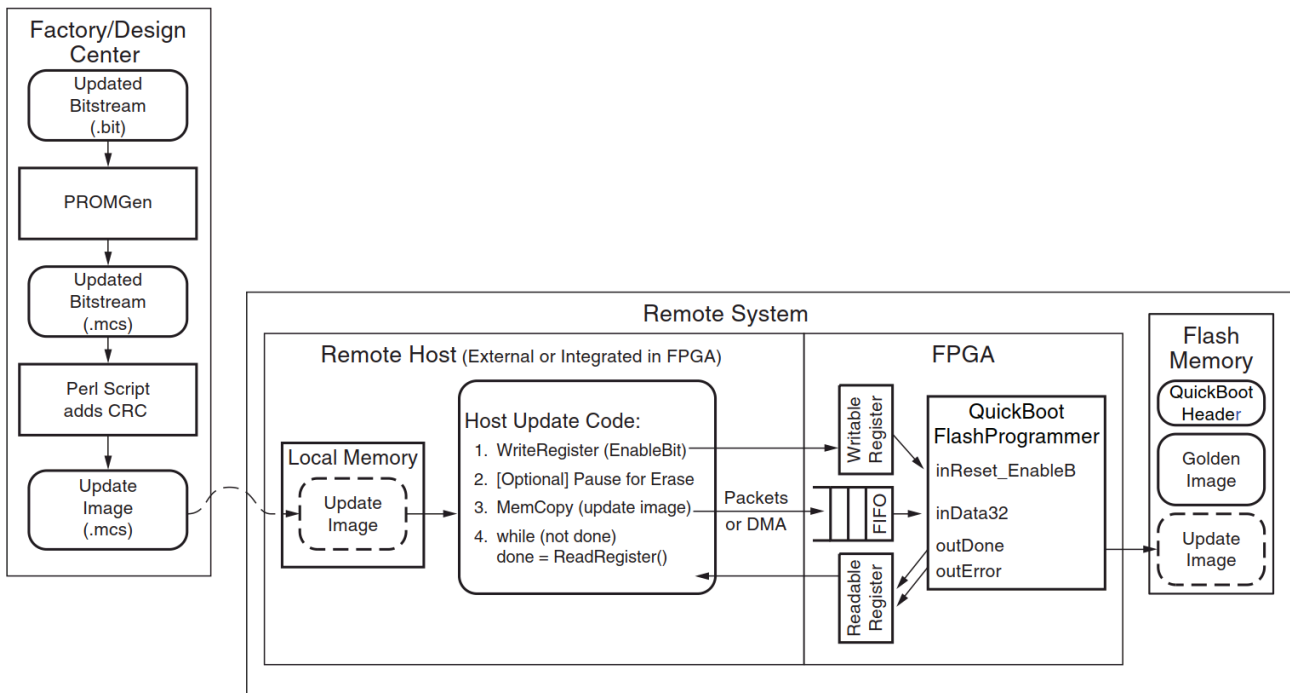


Figure 5.5.5. The QuickBoot mechanism [33].

```

pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$ testAltiQuickBoot -h
-----
testAltiQuickBoot <OPTIONS>:
-----
-s <int> => ALTI slot number          (def = 13)
-f <file> => Update image binary file (def = "ALTI/data/top_alti_update.bin")
-C        => Check ID only            (def = "NO")
-V        => Verify only              (def = "NO")
-h        => Print this help
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$ █

```

Figure 5.5.6. *testAltiQuickBoot* program help with the full list of parameters.

```

pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$ testAltiQuickBoot -s 18 -f TOP_ALTI_update_12809001.bin ^
ALTI: INFO - ALTI detected, revision = 0x12809001 (#1, 09.08.2018.), BAR = 0x12 <-> slot 18
ALTI: INFO - ALTI opened at vme = 0x20000000, size = 0x01000000, type = "A32"
I2C::I2C: INFO - opened I2C of type "ALTI" at offset 0x00009080
2018-08-10 12:21:49 - INFO in "LVL1::AltiModule::AltiModule(unsigned int)":
>> opened CMEM segment "AltiModule" of size 0x00400000, phys 0ix37c00000, virt 0x8904a000
DS1WM::DS1WM: INFO - opened DS1WM of type "ALTI" at offset 0x000080a0
-----
ALTI[slot 18]: INFO - ALTI "20DATAALTI00004" opened
-----
FIFO pointers and state machine reset
file length (bytes): 0x00800000
update_img_len (32b words): 0x00100000
...
Update done: OK
FIFO pointers and state machine reset
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$ █

```

Figure 5.5.7. Typical use of the *testAltiQuickBoot* program

Using the *testAltiQuickBoot* program proved beneficial in laboratory testing, when multiple ALTI modules are inserted the same VME crate, and the JTAG programming port is inaccessible. It also allowed the firmware update for the remotely installed modules, used and tested by other colleagues. So, this feature is rather useful for release updates where bug fixes make it mandatory to update the firmware.

#### 5.5.4. *testAltiCapture*

This test program is used for reading the snapshot memory and for comparison with expected patterns of TTC signals. Comparing the snapshot with the predefined pattern of input signals is very useful for testing all signal paths and connections through the module.

Before using the *testAltiCapture* program, the user first has to send a known TTC signal pattern to the FPGA. This is most easily accomplished by setting up one ALTI/LTP module in master mode, as a pattern generator. So, the pattern input file of an LTP or ALTI can be used as a reference for comparison with a snapshot memory of the ALTI module under test. Then, the ALTI module being tested has to be set up in a slave mode, such that it receives those TTC signals.

The test program repeatedly triggers the snapshot memory, reads its content and compares it to a pattern input file. Relative timestamps in the snapshot memory make the comparison with an LTP/ALTI pattern generation input file easier. Before actually comparing the data, the program finds the alignment between the data read from the ALTI and the comparison data. This is necessary since the data obtained from the snapshot memory depends on when the snapshot is actually enabled.

There are numerous parameters for the *testAltiCapture* program, including a comparison mask, the amount of data to be read, the comparison file type (LTP/ALTI), etc. They are shown on Figure 5.5.8. An example of the ALTI pattern file is shown on Figure 5.5.9, while a common use of the *testAltiCapture* program to compare snapshots with the given pattern is shown on Figure 5.5.10.

```

pkuzmano@sbcl1ct-29:/mnt/ctf/fs/pkuzmano/11ct-07-01-09
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$ testAltiCapture -h
-----
testAltiCapture <OPTIONS>:
-----
TEST MODE:
-s <int>          => ALTI slot number                (def = 11)
-S <file>         => Snapshot file                    (def = "")
-I <file>         => Input data file                  (def = "")
-J <uint>         => Input data file type (0=ALTI,1=LTP) (def = ALTI)
-----
TRIGGER DATA COMPARISON:
-i <file>         => Input data size                (def = 1048576)
-f <file>         => Comparison file                    (def = "")
-F <uint>         => Comparison file type (0=ALTI,1=LTP) (def = ALTI)
-t <uint8>        => TYP words (file type LTP)          (def = 0x00,0x00,0x00,0x00)
-x <int>          => Comparison size                (def = 3564)
-y <int>          => Comparison minimum matches          (def = 3564)
-m <uint(,uint)>  => Comparison mask                    (def = 0xffffffff)
-n <int>          => Maximum #data      (-1=INF)            (def = 1048575)
-e <int>          => Maximum #errors   (-1=INF)            (def = 0)
-----
PRINT & DEBUG LEVEL:
-P <int>          => Print number                    (def = 100)
-d <int>          => Print level                      (def = 1)
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$

```

Figure 5.5.8. *testAltiCapture* program help with the full list of parameters.

```

#-----
#                               M
#                               u
#                               l
#                               t
#                               i
#                               p
#                               l
#   CCC                          i
# BRRR   T BBBB TTT   c
#O UEEE   T GGGG TTTL i
#R SQQQ   Y OOOO RRR1 t
#B Y210   P 3210 321A y
-----
0 0000 0x00 0000 0000 1
1 0000 0x00 0000 0000 40
0 0000 0x12 0000 0001 1
0 0000 0x12 0000 0000 4
0 0000 0x12 0001 0000 1
0 0000 0x12 0000 0000 5
0 0000 0xba 0000 0001 1
0 0000 0xba 0000 0000 6
0 0000 0xba 0010 0000 1
0 0000 0xba 0000 0000 7
0 0000 0x00 0000 0000 3493

```

Figure 5.5.9. An example of the ALTI pattern input file used to run the pattern generator.

```

pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/11ct-07-01-09
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$ testAltiCapture -s 13 -i 3564 -f pg_alti.dat -m 0xfffffff ^
2018-08-20 10:18:12 - INFO in "daq::tmgr::TestResult LVL1::TestAltiCapture::init()"
-----
testAltiCapture:
-----
ALTI slot number          =          13
Snapshot file             = NO SNAPSHOT
Input data size           =          3564
Comparison file           = "pg_alti.dat"
Comparison file type      = ALTI
Comparison size           =          3564
Comparison mask           = 0xfffffff
Comparison matches       =          3564
Maximum number of data   =       1048575
Maximum number of errors =           0
-----
Print number              =          100
Print level                =           1
-----
2018-08-20 10:18:12 - INFO in "daq::tmgr::TestResult LVL1::TestAltiCapture::readCompFile()":
>> limiting number of data read from comparison file to file size
Comparison file status    = OPENED
Comparison file size      =          11
Comparison minimum matches =          11
ALTI: INFO - ALTI detected, revision = 0x1280f001 (#1, 15.08.2018.), BAR = 0x0d <-> slot 13
ALTI: INFO - ALTI opened at vme = 0x10000000, size = 0x01000000, type = "A32"
I2C::I2C: INFO - opened I2C of type "ALTI" at offset 0x00009080
2018-08-20 10:18:12 - INFO in "LVL1::AltiModule::AltiModule(unsigned int)":
>> opened CMEM segment "AltiModule" of size 0x00400000, phys 0ix37c00000, virt 0xaff39000
DS1WM::DS1WM: INFO - opened DS1WM of type "ALTI" at offset 0x000080a0
-----
ALTI[slot 13]: INFO - ALTI "20DATAALTI00007" opened
-----
TestAltiCapture::TestAltiCapture0: INFO - opened CMEM buffer of size 0x000037b0
2018-08-20 10:18:12 - INFO in "daq::tmgr::TestResult LVL1::TestAltiCapture::init()":
>> signal handler installed for SIGINT
2018-08-20 10:18:16 - INFO in "daq::tmgr::TestResult LVL1::TestAltiCapture::stat()"
  number of loop          :           100 => 30.377 Hz
  number of data read     :       355361 => 421.669 kByte/s
  number of data errors   :           0 => 0% and 0 bits per word
2018-08-20 10:18:19 - INFO in "daq::tmgr::TestResult LVL1::TestAltiCapture::stat()"
  number of loop          :           200 => 30.358 Hz
  number of data read     :       710727 => 421.410 kByte/s
  number of data errors   :           0 => 0% and 0 bits per word
2018-08-20 10:18:22 - INFO in "daq::tmgr::TestResult LVL1::TestAltiCapture::stat()"
  number of loop          :           296 => 30.350 Hz
  number of data read     :       1048575 => 419.982 kByte/s
  number of data errors   :           0 => 0% and 0 bits per word
2018-08-20 10:18:22 - INFO in "daq::tmgr::TestResult LVL1::TestAltiCapture::exit()":
>> previous signal handler re-installed for SIGINT
>> TestAltiCapture::test ..... [PASS]
-----
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$

```

Figure 5.5.10. Typical use of the *testAltiCapture* program.

### 5.5.5. *testAltiTtc*

This test program is used for reading the TTC decoder memory and for comparing the data with the expected TTC stream commands and triggers. Both the TTC decoder and encoder are being tested using this program.

Before using the *testAltiTtc* program, the user has to set up one ALTI module to send TTC commands and triggers. It is possible to use the same ALTI module for testing by sending its optical output to the optical receiver of the same module, too. Alternatively, one can set up a legacy TTC partition using TTCvi and TTCex and use that optical stream, too.

In order to run an automatic test in a loop, we need something as a reference, similar to a pattern input file in the capture program. For that purpose, a software package called ttcscope was used. This is TTC decoder software which gets the TTC stream data samples from the LeCroy oscilloscope [35]. The processing PC is connected with an Ethernet cable to the oscilloscope and the data is being transmitted over the TCP/IP protocol. Optical/electrical converter probe OE455 [36] is used to feed the electrical TTC stream signal to the oscilloscope. By running the ttcscope, the printout of decoded commands and triggers is obtained, as shown on Figure 5.5.11. Such printout gives a neat description of the TTC commands and triggers that can be used as a reference for comparison.

```
[pkuzmano@pcphese03 ~]$ ttcWaveformAnalyser -ip 192.168.0.2 -ch 2 -p > ttcscope.dat
```

List of TTC messages:

| Msg   | TTCrx E SUB                | DATA CHCK   | Pseudo-L1ID | CHCK VAL | BCID | BC Id:fied | BC Count |
|-------|----------------------------|-------------|-------------|----------|------|------------|----------|
|       | [Type CNT-HEX              | (CNT-DEC )] |             |          |      |            |          |
| Short |                            | D 0x01 0x13 |             | OK       | 0000 | NO         | 00000448 |
| Long  | A 0x1111 1 S 0x00          | D 0x11 0x3a |             | OK       | 0987 | YES        | 00001436 |
| Long  | A 0x2222 1 S 0x00          | D 0x22 0x42 |             | OK       | 1491 | YES        | 00001940 |
| Long  | A 0x3333 1 S 0x00          | D 0x33 0x18 |             | OK       | 1996 | YES        | 00002445 |
| L1A   |                            |             | 0x00000000  |          | 2145 | YES        | 00002594 |
| L1A   |                            |             | 0x00000001  |          | 2158 | YES        | 00002607 |
| L1A   |                            |             | 0x00000002  |          | 2172 | YES        | 00002621 |
| L1A   |                            |             | 0x00000003  |          | 2187 | YES        | 00002636 |
| Long  | A 0x1234 0 S 0x00          | D 0x55 0x4b |             | OK       | 2200 | YES        | 00002649 |
| L1A   |                            |             | 0x00000004  |          | 2203 | YES        | 00002652 |
| L1A   |                            |             | 0x00000005  |          | 2226 | YES        | 00002675 |
| Long  | A 0x1234 0 S 0x01          | D 0xdf 0x63 |             | OK       | 2247 | YES        | 00002696 |
| L1A   |                            |             | 0x00000006  |          | 2259 | YES        | 00002708 |
| L1A   |                            |             | 0x00000007  |          | 2293 | YES        | 00002742 |
| Long  | A 0x1234 0 S 0x02          | D 0x3f 0x02 |             | OK       | 2294 | YES        | 00002743 |
| L1A   |                            |             | 0x00000008  |          | 2336 | YES        | 00002785 |
| Long  | A 0x1234 0 S 0x03          | D 0x7b 0x1b |             | OK       | 2341 | YES        | 00002790 |
| L1A   |                            |             | 0x00000009  |          | 2380 | YES        | 00002829 |
| Long  | A 0x1234 0 S 0x00          | D 0xaa 0x53 |             | OK       | 2388 | YES        | 00002837 |
| Long  | A 0x1234 0 S 0x01          | D 0xdf 0x63 |             | OK       | 2435 | YES        | 00002884 |
| Long  | A 0x1234 0 S 0x02          | D 0x3f 0x02 |             | OK       | 2482 | YES        | 00002931 |
| Long  | A 0x1234 0 S 0x03          | D 0x7c 0x62 |             | OK       | 2529 | YES        | 00002978 |
| TType | [0xaa 0xdf3f7c (14630780)] |             |             | OK       | 2529 | YES        | 00002978 |
| Long  | A 0x1234 0 S 0x00          | D 0xff 0x6c |             | OK       | 2576 | YES        | 00003025 |
| Long  | A 0x1234 0 S 0x01          | D 0xdf 0x63 |             | OK       | 2623 | YES        | 00003072 |
| Long  | A 0x1234 0 S 0x02          | D 0x3f 0x02 |             | OK       | 2670 | YES        | 00003119 |
| Long  | A 0x1234 0 S 0x03          | D 0x7d 0x7b |             | OK       | 2717 | YES        | 00003166 |
| TType | [0xff 0xdf3f7d (14630781)] |             |             | OK       | 2717 | YES        | 00003166 |
| Long  | A 0x1234 0 S 0x00          | D 0xde 0x14 |             | OK       | 2764 | YES        | 00003213 |
| Long  | A 0x1234 0 S 0x01          | D 0xdf 0x63 |             | OK       | 2811 | YES        | 00003260 |
| Long  | A 0x1234 0 S 0x02          | D 0x3f 0x02 |             | OK       | 2858 | YES        | 00003307 |
| Long  | A 0x1234 0 S 0x03          | D 0x7e 0x4b |             | OK       | 2905 | YES        | 00003354 |
| TType | [0xde 0xdf3f7e (14630782)] |             |             | OK       | 2905 | YES        | 00003354 |

Statistics:

```
Data acquisition time      : Mon Aug 20 11:46:42.914837000 2018 CEST
Scope trigger time       : Mon Aug 20 12:01:41.374810706 2018 CEST

Scope sampling interval  : 0.2000 ns
BC clock interval (mean) : 24.9502 ns
BC clock interval (RMS)  : 24.9510 ns
Sequence duration        : 4008 BC counts (1 orbits + 444 BC counts)

L1As                     : 10
Commands (consistent)    : 20 ( 20)
Long commands (consistent) : 19 ( 19)
Short commands (consistent) : 1 ( 1)

BCRs                     : 1
ECRs                     : 0
Joint BCRs and ECRs     : 0
Unrecognised short commands : 0
```

Figure 5.5.11. Result of running the ttcscope program.



The test program then repeatedly triggers the two TTC decoder memories (called "command" and "timestamp"), reads their content, interprets the data and compares it to the ttcscope reference file. Alignment and comparison is analogous to the ones used in *testAltiCapture* program, with a few subtle differences. Timestamps in ttcscope and the ALTI decoder have a relative offset, so they are masked in the alignment part of the algorithm. However, this offset is calculated after the alignment and the timestamps are adjusted so that they can be compared. Comparing the timestamps makes sure that the timing of periodic triggers and commands stays stable.

The testAltiTtc program was created by applying slight modifications to the testAltiCapture program, so the list of program parameters is also very similar. The full list of parameters is shown on Figure 5.5.12, while the typical use of the program is shown on Figure 5.5.13.

```

pkuzmano@sbc11ct-29:/mnt/ctfps/pkuzmano/11ct-07-01-09
[pkuzmano@sbc11ct-29 11ct-07-01-09]$ testAltiTtc -h
-----
testAltiTtc <OPTIONS>:
-----
TEST MODE:
-s <int>          => ALTI slot number                (def = 11)
-S <file>         => Snapshot file                    (def = "")
-I <file>         => Input data file                        (def = "")
-----
TRIGGER DATA COMPARISON:
-i <file>         => Input data size                            (def = 1048576)
-f <file>         => Comparison file                                (def = "")
-t <uint16,uint8> => TYP address,subaddress                       (def = 0x0000,0x00)
-p <int>          => TYP address space (0-INTERNAL/1-EXTERNAL) (def = EXTERNAL)
-x <int>          => Comparison size                                (def = 3564)
-y <int>          => Comparison minimum matches                 (def = 3564)
-m <uint(,uint)>  => Comparison mask                               (def = 0xffffffff,0xffffffff)
-n <int>          => Maximum #data      (-1=INF)                  (def = 1048575)
-e <int>          => Maximum #errors   (-1=INF)                  (def = 0)
-----
PRINT & DEBUG LEVEL:
-P <int>          => Print number                                    (def = 100)
-d <int>          => Print level                                       (def = 1)
[pkuzmano@sbc11ct-29 11ct-07-01-09]$ █

```

Figure 5.5.12. *testAltiTtc* program help with the full list of parameters.

```

pkuzmano@sbcl1ct-29/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltiTtc -s 9 -f ttcscope.dat -t 0x1234,0x00 -m 0xffffffff,0x5fffffff -p 0 -n 1000 -i 128 ^
2018-08-21 12:07:45 - INFO in "daq::tmgr::TestResult LVL1::TestAltiTtc::init()"
-----
testAltiTtc:
-----
ALTI slot number          =          9
Snapshot file             = NO SNAPSHOT
Input data size           =         128
Comparison file           = "ttcscope.dat"
TTY address, subaddress   = 0x1234,0x00
TTY address space         = INTERNAL
Comparison size           =        3564
Comparison mask           = 0xffffffff,0x5fffffff
Comparison matches        =        3564
Maximum number of data    =        1000
Maximum number of errors  =          0
-----
Print number              =         100
Print level                =          1
-----
2018-08-21 12:07:45 - INFO in "daq::tmgr::TestResult LVL1::TestAltiTtc::readCompFile()":
>> limiting number of data read from comparison file to file size
Comparison file status    = OPENED
Comparison file size      =         30
Comparison minimum matches =         30
ALTI: INFO - ALTI detected, revision = 0x1280f001 (#1, 15.08.2018.), BAR = 0x09 <-> slot 9
ALTI: INFO - ALTI opened at vme = 0x08000000, size = 0x01000000, type = "A32"
I2C::I2C: INFO - opened I2C of type "ALTI" at offset 0x00009080
2018-08-21 12:07:45 - INFO in "LVL1::AltiModule::AltiModule(unsigned int)":
>> opened CMEM segment "AltiModule" of size 0x00400000, phys 0ix37c00000, virt 0xa02dc000
DS1WM::DS1WM: INFO - opened DS1WM of type "ALTI" at offset 0x000080a0
-----
ALTI[slot 9]: INFO - ALTI "20DATAALTI00002" opened
-----
TestAltiTtc::TestAltiTtc0: INFO - opened CMEM buffer of size 0x00000200
TestAltiTtc::TestAltiTtc1: INFO - opened CMEM buffer of size 0x00000200
2018-08-21 12:07:45 - INFO in "daq::tmgr::TestResult LVL1::TestAltiTtc::init()":
>> signal handler installed for SIGINT
2018-08-21 12:07:46 - INFO in "daq::tmgr::TestResult LVL1::TestAltiTtc::stat()":
  number of loop          :          33 => 295.895 Hz
  number of data read      :        1000 => 35.025 kByte/s
  number of data errors    :          0 => 0% and 0 bits per word
2018-08-21 12:07:46 - INFO in "daq::tmgr::TestResult LVL1::TestAltiTtc::exit()":
>> previous signal handler re-installed for SIGINT
>> TestAltiTtc::test ..... [PASS]
-----
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$

```

Figure 5.5.13. Typical use of the *testAltiTtc* program.

### 5.5.6. *testAltiSync*

This test program was developed in order to evaluate the input synchronization and monitoring firmware. The idea was to make use of the LTPI shifting functionality and observe what happens to the ALTI histograms for input synchronization and monitoring. Since this test program operates on both modules, the LTPI and the ALTI, the low-level LTPI software library is used to access that module and shift the TTC signals.

Before running the test program, a pattern of TTC signals is generated (with an LTP or an ALTI) and propagated through the LTPI module, to its CTP\_OUT connector. Then, the connection with the downstream ALTI module under test is done using a single LVDS-LINK cable to either of the two ALTI LVDS-LINK input connectors.

When running the *testAltiSync* program, the user can choose the source connector of the signals coming from an upstream LTPI (CTP\_OUT or ALTI\_OUT) and the desired delay (in nanoseconds) to be introduced by the LTPI for all the signals. The result of running the program is the printout of ALTI histograms.

The full list of available program parameters is shown on Figure 5.5.14, while the common use of the *testAltiSync* is shown on figures 5.5.15 and 5.5.16. Note the shift of the histogram content to the next bin as a result of a different phase of the TTC signals on Figure 5.5.15 and Figure 5.5.16.

```

pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltiSync -h
-----
testAltiSync <OPTIONS>:
-----
-a <uint>      => LTPI VME base address                (def = 0x00ee2000)
-d <int>      => LTPI CTP link delay, 0.5ns step [0-60] (def = 0)
-s <int>      => ALTI slot number                    (def = 13)
-c <int>      => ALTI input cable (0-CTP_IN/1-ALTI_IN) (def = CTP_IN)
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$

```

Figure 5.5.14. *testAltiSync* program help with the full list of parameters.

```

pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltiSync -s 13 -c 0 -d 0
2018-08-21 11:48:00 - INFO in "daq:tmgr::TestResult LVL1::TestAltiSync::init()"
-----
testAltiSync:
-----
LTPI VME base address      = 0x00ee2000
LTPI CTP link delay       = 0 (0.0ns)
ALTI slot number          = 13
ALTI input cable          = CTP_IN
-----
ALTI: INFO - ALTI detected, revision = 0x1280f001 (#1, 15.08.2018.), BAR = 0x0d <-> slot 13
ALTI: INFO - ALTI opened at vme = 0x10000000, size = 0x01000000, type = "A32"
I2C::I2C: INFO - opened I2C of type "ALTI" at offset 0x00009080
2018-08-21 11:48:00 - INFO in "LVL1::AltiModule::AltiModule(unsigned int)":
>> opened CMEM segment "AltiModule" of size 0x00400000, phys 0ix38800000, virt 0x9f4cb000
DS1WM::DS1WM: INFO - opened DS1WM of type "ALTI" at offset 0x000080a0
-----
ALTI[slot 13]: INFO - ALTI "20DATAALTI00007" opened
-----
2018-08-21 11:48:00 - INFO in "daq:tmgr::TestResult LVL1::TestAltiSync::init()":
>> opened module "LTPI" at vme = 0x00ee2000, size = 0x00000100, type = "A24"
-----
*****
Manufacturer ID = 0x080030
Board ID        = 0x01081034
Revision number = 0x01022008
*****
LTPI CTP link delay set to 0 (0.0ns)

2018-08-21 11:48:01 - INFO in "daq:tmgr::TestResult LVL1::TestAltiSync::init()":
>> signal handler installed for SIGINT

Histogram statistics (delay = 0.0ns):
-----+-----
| Asynchronous input signal | Histogram count by phase [pos/neg] |
-----+-----
|                               | 0 deg | 90 deg | 180 deg | 270 deg |
-----+-----
| [00]SWX_ORB                 | 7/7   | 0/0   | 0/0   | 0/0   |
| [01]SWX_L1A                 | 7/7   | 0/0   | 0/0   | 0/0   |
| [02]SWX_BGO0                | 7/7   | 0/0   | 0/0   | 0/0   |
| [03]SWX_BGO1                | 7/7   | 0/0   | 0/0   | 0/0   |
| [04]SWX_BGO2                | 7/7   | 0/0   | 0/0   | 0/0   |
| [05]SWX_BGO3                | 7/7   | 0/0   | 0/0   | 0/0   |
| [06]SWX_TTYP0               | 7/7   | 0/0   | 0/0   | 0/0   |
| [07]SWX_TTYP1               | 7/7   | 0/0   | 0/0   | 0/0   |
| [08]SWX_TTYP2               | 7/7   | 0/0   | 0/0   | 0/0   |
| [09]SWX_TTYP3               | 7/7   | 0/0   | 0/0   | 0/0   |
| [10]SWX_TTYP4               | 7/7   | 0/0   | 0/0   | 0/0   |
| [11]SWX_TTYP5               | 7/7   | 0/0   | 0/0   | 0/0   |
| [12]SWX_TTYP6               | 7/7   | 0/0   | 0/0   | 0/0   |
| [13]SWX_TTYP7               | 7/7   | 0/0   | 0/0   | 0/0   |
| [14]SWX_TTR1                | 7/7   | 0/0   | 0/0   | 0/0   |
| [15]SWX_TTR2                | 7/7   | 0/0   | 0/0   | 0/0   |
-----+-----
2018-08-21 11:48:01 - INFO in "daq:tmgr::TestResult LVL1::TestAltiSync::exit()":
>> previous signal handler re-installed for SIGINT
>> TestAltiSync::test ..... [PASS]
-----
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$

```

Figure 5.5.15. Typical use of the *testAltiSync* program: without the LTPI delay.

```

pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/l1ct-07-01-09
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$ testAltiSync -s 13 -c 0 -d 8
2018-08-21 11:48:34 - INFO in "daq:tmgr::TestResult LVL1::TestAltiSync::init()"
-----
testAltiSync:
-----
LTPI VME base address      = 0x00ee2000
LTPI CTP link delay       = 8 (4.0ns)
ALTI slot number          = 13
ALTI input cable          = CTP_IN
-----
ALTI: INFO - ALTI detected, revision = 0x1280f001 (#1, 15.08.2018.), BAR = 0x0d <-> slot 13
ALTI: INFO - ALTI opened at vme = 0x10000000, size = 0x01000000, type = "A32"
I2C::I2C: INFO - opened I2C of type "ALTI" at offset 0x00009080
2018-08-21 11:48:34 - INFO in "LVL1::AltiModule::AltiModule(unsigned int)":
>> opened CMEM segment "AltiModule" of size 0x00400000, phys 0ix38800000, virt 0xb3c31000
DS1WM::DS1WM: INFO - opened DS1WM of type "ALTI" at offset 0x000080a0
-----
ALTI[slot 13]: INFO - ALTI "20DATAALTI00007" opened
-----
2018-08-21 11:48:34 - INFO in "daq:tmgr::TestResult LVL1::TestAltiSync::init()":
>> opened module "LTPI" at vme = 0x00ee2000, size = 0x00000100, type = "A24"
-----
*****
Manufacturer ID = 0x080030
Board ID        = 0x01081034
Revision number = 0x01022008
*****
LTPI CTP link delay set to 8 (4.0ns)

2018-08-21 11:48:35 - INFO in "daq:tmgr::TestResult LVL1::TestAltiSync::init()":
>> signal handler installed for SIGINT

Histogram statistics (delay = 4.0ns):
-----+-----
| Asynchronous input signal | Histogram count by phase [pos/neg] |
-----+-----
|                               | 0 deg | 90 deg | 180 deg | 270 deg |
-----+-----
| [00]SWX_ORB                 | 0/7   | 7/7   | 0/0   | 0/0   |
| [01]SWX_L1A                 | 0/0   | 7/7   | 0/0   | 0/0   |
| [02]SWX_BGO0                | 0/0   | 7/7   | 0/0   | 0/0   |
| [03]SWX_BGO1                | 0/0   | 7/7   | 0/0   | 0/0   |
| [04]SWX_BGO2                | 7/7   | 7/7   | 0/0   | 0/0   |
| [05]SWX_BGO3                | 7/7   | 7/7   | 0/0   | 0/0   |
| [06]SWX_TTYP0               | 0/0   | 7/7   | 0/0   | 0/0   |
| [07]SWX_TTYP1               | 0/0   | 7/7   | 0/0   | 0/0   |
| [08]SWX_TTYP2               | 7/7   | 7/7   | 0/0   | 0/0   |
| [09]SWX_TTYP3               | 0/0   | 7/7   | 0/0   | 0/0   |
| [10]SWX_TTYP4               | 0/0   | 7/7   | 0/0   | 0/0   |
| [11]SWX_TTYP5               | 7/0   | 7/7   | 7/0   | 0/0   |
| [12]SWX_TTYP6               | 0/0   | 7/7   | 0/0   | 0/0   |
| [13]SWX_TTYP7               | 0/0   | 7/7   | 0/0   | 0/0   |
| [14]SWX_TTR1                | 7/7   | 0/0   | 0/0   | 0/0   |
| [15]SWX_TTR2                | 7/7   | 0/0   | 0/0   | 0/0   |
-----+-----
2018-08-21 11:48:35 - INFO in "daq:tmgr::TestResult LVL1::TestAltiSync::exit()":
>> previous signal handler re-installed for SIGINT
>> TestAltiSync::test ..... [PASS]
-----
[pkuzmano@sbcl1ct-29 l1ct-07-01-09]$

```

Figure 5.5.16. Typical use of the *testAltiSync* program: with the LTPI delay of 4ns, resulting in the histogram shift.

## 6. MODULE TESTING

This chapter describes the process of testing the ALTI prototype modules and lists the most important test results. First, the laboratory tests on individual modules are discussed in Section 6.1. An automated connection test that was designed for systematic testing and evaluation of the large number of prototype modules is discussed in Section 6.2.

### 6.1. Laboratory tests

With the help of the low-level software, the menu programs (for both the ALTI and the other legacy TTC modules) and the oscilloscope, the prototype modules have been tested. A typical laboratory setup used in such module tests is shown on Figure 6.1.1

That laboratory tests include the checking of all the hardware on the module:

- 1) Complete I2C network
- 2) RAM memories external to the FPGA
- 3) LVDS-LINK cable inputs/outputs
- 4) LEMO NIM-level inputs/outputs
- 5) LEMO TTL-level BUSY input
- 6) All routing paths through the cross-point switches
- 7) Equalizers: configurations for short/long cables found
- 8) RJ45 calibration request inputs
- 9) 1-Wire ID chip for labeling the modules



Figure 6.1.1. Typical laboratory test setup with multiple ALTI and legacy TTC modules in the same VME crate.

Also, laboratory tests were performed in order to check all of the ALTI functionalities and make sure that they are properly implemented in the firmware.

In the module testing process, several PCB design issues have been found, all of which shall be fixed in the next pre-production series:

- 1) ADCMP564 comparators on the NIM outputs should be supplied by +5V, instead of +3.3.V (prototype modules have been fixed by rewiring)
- 2) MAX1617A temperature sensor remote diode pins polarity is swapped
- 3) Power-down mode pins for the AD8123 equalizers should be driven from the FPGA to reduce the heat dissipation
- 4) The termination in the ALTI BUSY input needs to be slightly modified in order to allow TTL-level inputs

Some issues related to the bad assembly and soldering have been identified on the particular prototype modules. These were not related to any of the design issues, and were easily fixed on each of these particular modules.

## 6.2. Automated connection test

The *ConnectionTestAlti* program was designed in order to systematically check all forward-going signal paths through the ALTI, for varying configurations of cross-point switches. It was written in Python and it makes extensive calls to the *testAltiInitial* and *testAltiCapture* programs.

The ALTI module under test is put in between two "golden" ALTI modules, fully connected with LVDS cables and LEMO-connector coaxial cables. This setup is shown on Figure 6.2.1.

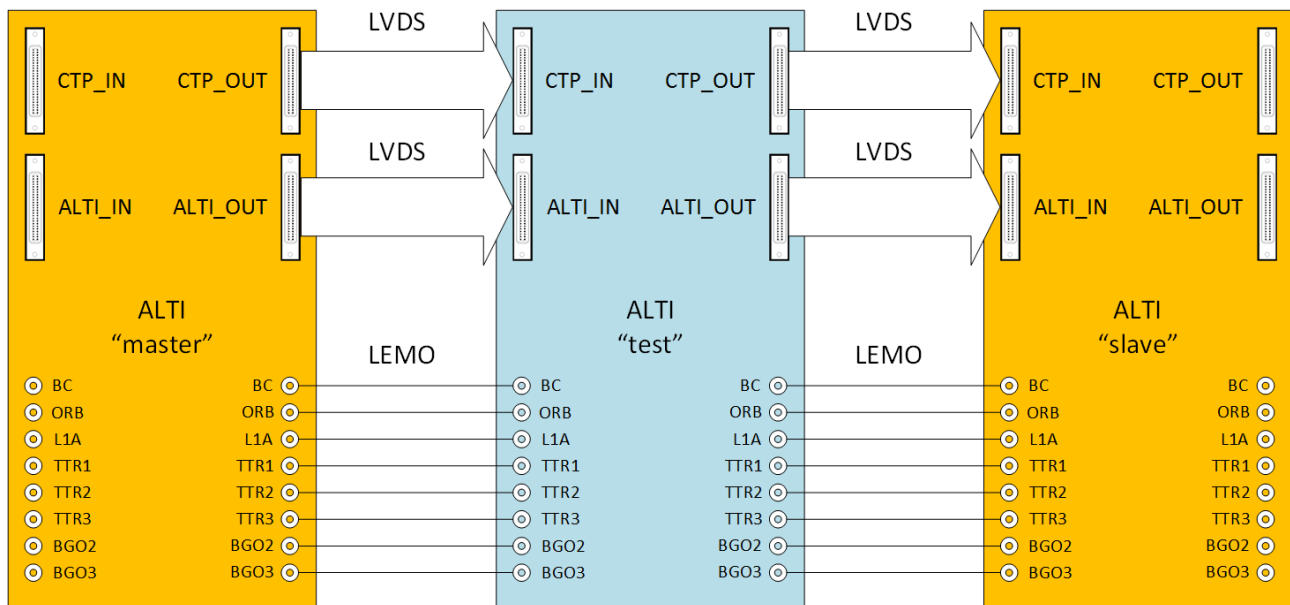


Figure 6.2.1. Setup for testing of all input/output paths of TTC signals.

Typically, the "Master" ALTI module sends a pattern of TTC signals given by a randomly generated input file. The lengths of the random patterns and the probabilities for "0" or "1" occurrence are programmable. The signals are then propagated through the "Test" ALTI module and are captured in the "Slave" ALTI module and compared with the given pattern. Exceptions to this rule are the tests of FROM\_FPGA input paths (where the "Test" ALTI acts as a pattern generator) and TO\_FPGA output paths (where the "Test" ALTI takes snapshots). The test of a single input/output combination consists of initializing the ALTI modules with *testAltiInital* program, and then taking the snapshot with the *testAltiCapture* program. Predefined ALTI configurations are used for the initialization: Pattern generator, "CTP slave", "ALTI slave" and "LEMO slave".

All input/output paths are being tested, from the front panel inputs to the front panel outputs, as well as to and from the FPGA. Thus, there are 15 different input/output combinations that are being tested. Paths with FROM\_FPGA input and TO\_FPGA output are not being tested, since it is not possible to use both the pattern generation and the snapshot features of the same module simultaneously.

Table 6.2.1 shows all input/output paths being tested and the corresponding configurations of the "Master", "Test" and "Slave" ALTI modules.

**Table 6.2.1. Standard configurations of ALTI modules in the connection test for various test paths.**

| PATH                  | "MASTER" ALTI CONFIGURATION | "TEST" ALTI CONFIGURATION | "SLAVE" ALTI CONFIGURATION |
|-----------------------|-----------------------------|---------------------------|----------------------------|
| CTP_IN -> CTP_OUT     | Pattern generator           | CTP slave                 | CTP slave                  |
| CTP_IN -> ALTI_OUT    | Pattern generator           | CTP slave                 | ALTI slave                 |
| CTP_IN -> LEMO_OUT    | Pattern generator           | CTP slave                 | LEMO slave                 |
| CTP_IN -> TO_FPGA     | Pattern generator           | CTP slave                 | N/A                        |
| ALTI_IN -> CTP_OUT    | Pattern generator           | ALTI slave                | CTP slave                  |
| ALTI_IN -> ALTI_OUT   | Pattern generator           | ALTI slave                | ALTI slave                 |
| ALTI_IN -> LEMO_OUT   | Pattern generator           | ALTI slave                | LEMO slave                 |
| ALTI_IN -> TO_FPGA    | Pattern generator           | ALTI slave                | N/A                        |
| LEMO_IN -> CTP_OUT    | Pattern generator           | LEMO slave                | CTP slave                  |
| LEMO_IN -> ALTI_OUT   | Pattern generator           | LEMO slave                | ALTI slave                 |
| LEMO_IN -> LEMO_OUT   | Pattern generator           | LEMO slave                | LEMO slave                 |
| LEMO_IN -> TO_FPGA    | Pattern generator           | LEMO slave                | N/A                        |
| FROM_FPGA -> CTP_OUT  | N/A                         | Pattern generator         | CTP slave                  |
| FROM_FPGA -> ALTI_OUT | N/A                         | Pattern generator         | ALTI slave                 |
| FROM_FPGA -> LEMO_OUT | N/A                         | Pattern generator         | LEMO slave                 |

If the *testAltiCapture* returns an error, the corresponding input/output path failure is reported. There can be multiple reasons for this failure: wrong equalizer settings, wrong input signal synchronization or simply a bad soldering or some assembly issue. In order to report what TTC signal lines are problematic, the snapshot memory masking feature is used. After the failure, TTC signals are independently checked by masking all other signals from the snapshot, and the *testAltiCapture* is ran again. This means that the changes of all the masked signals are not stored in the snapshot memory, so only one TTC signal is checked at a time. If this call of the *testAltiCapture* reports failure, that means there is a problem for this particular TTC signal for a given input/output combination of the cross-point switch.

On Figure 6.2.2, a typical use of the *ConnectionTestAlti* is shown. The full report gives the user the information on which paths failed the test, and which TTC signals are in fact problematic. As shown on Figure 6.2.3, by using the proper parameter the user can also check the BGO multiplexed paths discussed in Subsection 3.2.3. The user is also guided on how to do the cabling between the modules, as can be seen on figures 6.2.2 and 6.2.3.

```
pkuzmano@sbcl1ct-29:/mnt/ctpfs/pkuzmano/11ct-07-01-09
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$ python ALTI/src/test/ConnectionTestAlti.py -n 100 ^
"Master" ALTI: slot 9
"Test"  ALTI: slot 13
"Slave" ALTI: slot 18

Pattern file:
ALTI/src/test/../../data/alti_pg_random_forward.dat

-----

Connection instructions:
(BGO2/3 LEMO inputs/outputs tested)

-----

| ALTI      | CTP_OUT  CTP_IN | ALTI      | CTP_OUT  CTP_IN | ALTI      |
| master    |             | test      |             |             | slave     |
| slot 9    | ALTI_OUT  ALTI_IN | slot 13   | ALTI_OUT  ALTI_IN | slot 18   |
|           | ----->|           | ----->|           |
|           |             |           |             |             |
|           | NIM_OUT   NIM_IN |           | NIM_OUT   NIM_IN |           |
|           | ----->|           | ----->|           |
| BC        |             | BC        | BC        |             | BC        |
| ORB       |             | ORB       | ORB       |             | ORB       |
| L1A       |             | L1A       | L1A       |             | L1A       |
| TTR1      |             | TTR1      | TTR1      |             | TTR1      |
| TTR2      |             | TTR2      | TTR2      |             | TTR2      |
| TTR3      |             | TTR3      | TTR3      |             | TTR3      |
| BGO2      |             | BGO2      | BGO2      |             | BGO2      |
| BGO3      |             | BGO3      | BGO3      |             | BGO3      |
|           | ----->|           | ----->|           |

After recabling, press any key to continue...

-----

CTP_IN  -> CTP_OUT connection OK
CTP_IN  -> ALTI_OUT connection failed
"TTYPO " connection failed
CTP_IN  -> LEMO_OUT connection OK
CTP_IN  -> TO_FPGA connection OK
ALTI_IN -> CTP_OUT connection OK
ALTI_IN -> ALTI_OUT connection failed
"TTYPO " connection failed
ALTI_IN -> LEMO_OUT connection OK
ALTI_IN -> TO_FPGA connection OK
LEMO_IN -> CTP_OUT connection failed
"ORB   " connection failed
"L1A   " connection failed
LEMO_IN -> ALTI_OUT connection failed
"ORB   " connection failed
"L1A   " connection failed
LEMO_IN -> LEMO_OUT connection failed
"ORB   " connection failed
"L1A   " connection failed
LEMO_IN -> TO_FPGA connection failed
"ORB   " connection failed
"L1A   " connection failed
FROM_FPGA -> CTP_OUT connection OK
FROM_FPGA -> ALTI_OUT connection failed
"TTYPO " connection failed
FROM_FPGA -> LEMO_OUT connection OK
[pkuzmano@sbcl1ct-29 11ct-07-01-09]$
```

Figure 6.2.2. Connection test results, LEMO BGO2 and BGO3 cabling.



```

pkuzmano@sbcl1ct-29:/mnt/ctfps/pkuzmano/llct-07-01-09
[pkuzmano@sbcl1ct-29 llct-07-01-09]$ python ALTI/src/test/ConnectionTestAlti.py -n 100 -M ^
"Master" ALTI: slot 9
"Test" ALTI: slot 13
"Slave" ALTI: slot 18

Pattern file:
ALTI/src/test/../../data/alti_pg_random_forward.dat

-----
Connection instructions:
(BGO0/1 LEMO inputs/outputs tested)

|-----|
| ALTI   | CTP_OUT  CTP_IN | ALTI   | CTP_OUT  CTP_IN | ALTI   |
| master | ----->| test   | ----->| slave  |
| slot 9 | ALTI_OUT ALTI_IN | slot 13| ALTI_OUT ALTI_IN | slot 18|
|-----|
|         | NIM_OUT   NIM_IN |         | NIM_OUT   NIM_IN |         |
| BC     | ----->| BC     BC | ----->| BC     |
| ORB    | ----->| ORB    ORB | ----->| ORB    |
| L1A    | ----->| L1A   L1A | ----->| L1A    |
| TTR1   | ----->| TTR1  TTR1 | ----->| TTR1   |
| TTR2   | /----->| TTR2  TTR2 | /----->| TTR2   |
| TTR3   | / /----->| TTR3  TTR3 | / /----->| TTR3   |
| BGO2   | -/ /     | BGO2  BGO2 | -/ /     | BGO2   |
| BGO3   | --/      | BGO3  BGO3 | --/      | BGO3   |
|-----|

After recabling, press any key to continue...

-----
CTP_IN   -> CTP_OUT connection OK
CTP_IN   -> ALTI_OUT connection failed
"TTYPO " connection failed
CTP_IN   -> LEMO_OUT connection OK
CTP_IN   -> TO_FPGA connection OK
ALTI_IN  -> CTP_OUT connection OK
ALTI_IN  -> ALTI_OUT connection failed
"TTYPO " connection failed
ALTI_IN  -> LEMO_OUT connection OK
ALTI_IN  -> TO_FPGA connection OK
LEMO_IN  -> CTP_OUT connection failed
"ORB " connection failed
"L1A " connection failed
LEMO_IN  -> ALTI_OUT connection failed
"ORB " connection failed
"L1A " connection failed
LEMO_IN  -> LEMO_OUT connection failed
"ORB " connection failed
"L1A " connection failed
LEMO_IN  -> TO_FPGA connection failed
"ORB " connection failed
"L1A " connection failed
FROM_FPGA -> CTP_OUT connection OK
FROM_FPGA -> ALTI_OUT connection failed
"TTYPO " connection failed
FROM_FPGA -> LEMO_OUT connection OK
[pkuzmano@sbcl1ct-29 llct-07-01-09]$

```

Figure 6.2.3. Connection test results, LEMO BGO0 and BGO1 cabling (multiplexed).

The reports shown on figures 6.2.2 and 6.2.3 suggest that there is a problem with L1A and ORB LEMO inputs for this particular ALTI module, as well as TTYPO line on the ALTI\_OUT path. Such an automated test allows for quick check and evaluation of the upcoming ALTI modules, something that would be practically impossible to do manually using the oscilloscope.

# 7. PERFORMANCE MEASUREMENTS

This chapter describes the performance measurements that were done on the ALTI module, and shows the obtained results. Those tests have been done in order to qualify some critical performance parameters of the ALTI module and compare them the legacy TTC modules. Low-level software for the ALTI and the other legacy TTC modules (in the form of menu programs) has been used in order to set the modules up appropriately for each measurement.

The parameters that were measured include:

- 1) Cable-to-cable latency of electrical TTC signals, compared to LTPI and LTP modules
- 2) Level-1 Accept latency in the TTC encoded optical stream, compared to the current daisy chain setup of LAr in the experiment
- 3) Jitter in the TTC stream and on-board jitter cleaner chip performance, compared to the current system based on TTCvi and TTCex

Measurements of each of these parameters are discussed in separate sections.

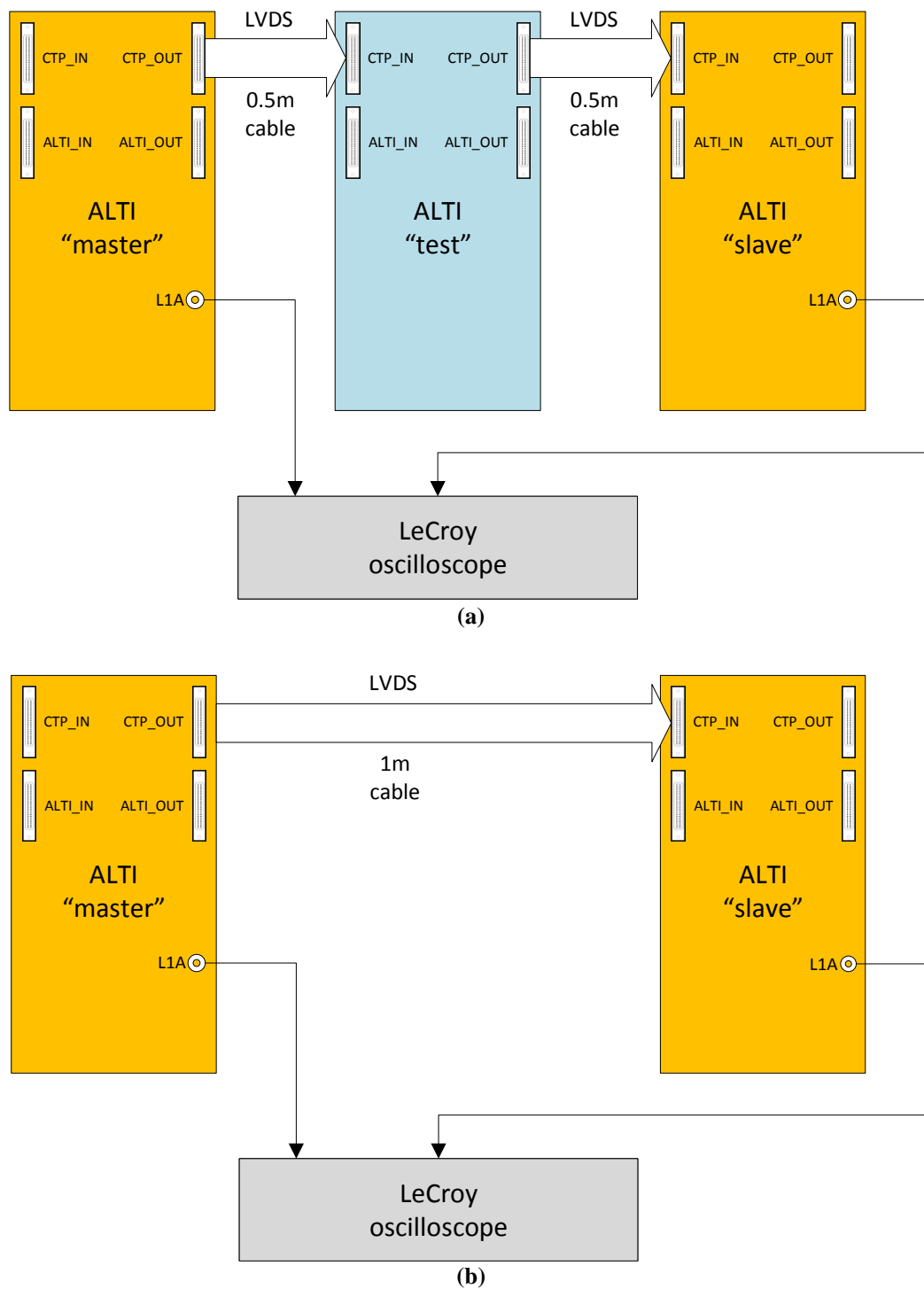
## 7.1. Latency of electrical TTC signals

Latency is very important in the TTC system and the ATLAS experiment. In particular, the latency of the L1A is very critical, because the front-end electronics buffers can only hold event data for a given time, before they lose or overrun the existing data.

Latencies of most of the forward-going TTC signals from the cable inputs (CTP\_IN/ALTI\_IN) to the cable outputs (CTP\_OUT/ALTI\_OUT) have been measured. Each of these delay paths consists of: an LVDS receiver, an equalizer, a cross-point switch and an LVDS driver.

A setup that was used for these measurements is shown on Figure 7.1.1. The delay of a particular signal is measured between the two ALTI modules called "Master" and "Slave". For this, output LEMO connectors for that particular signals are used. Two coaxial cables are used to feed the signals to the LeCroy oscilloscope and the delay between the two waveforms is then measured. On Figure 7.1.1, one such measurement has been shown, namely the latency of L1A through the CTP\_IN to CTP\_OUT path.

This is an indirect measurement, and the latency is obtained by subtracting the results of two measurements shown on Figure 7.1.1. The first measurement includes the propagation delay through the ALTI module under test, while the second measurement bypasses this ALTI module. In the first measurement two LVDS-LINK cables length 0.5m are used, and in the second measurement one LVDS-LINK cable of length 1m is used. Thus, the propagation delays through the cables are matched in both measurements and they cancel out in the subtraction of the results. The same holds for the propagation delays through the coaxial cables, which are the same in both measurements.



**Figure 7.1.1. CTP\_IN->CTP\_OUT path latency measurement for L1A: (a) delay of ALTI under test included, (b) ALTI module under test bypassed.**

The results of measuring the propagation delays through the ALTI module are shown in Table 7.1.1. What we can see from the table is that the latency is about 12ns, from any LVDS cable input to any LVDS cable output of the ALTI module. So, the delays for all the signals and for all four different input/output configurations are roughly the same. This is expected, since this circuitry in the cross-point switch path is the same for all the TTC signals.

**Table 7.1.1. Cable-to-cable latencies of TTC signals for the ALTI module.**

| SIGNAL | CTP/CTP<br>DELAY [NS] | CTP/ALTI<br>DELAY [NS] | ALTI/CTP<br>DELAY [NS] | ALTI/ALTI<br>DELAY [NS] |
|--------|-----------------------|------------------------|------------------------|-------------------------|
| BC     | 11.76                 | 11.56                  | 11.47                  | 11.18                   |
| ORB    | 12.04                 | 11.99                  | 11.57                  | 11.49                   |
| L1A    | 11.92                 | 11.69                  | 11.46                  | 11.21                   |
| TTR1   | 11.69                 | 11.58                  | 11.31                  | 11.17                   |
| TTR2   | 11.72                 | 11.91                  | 11.44                  | 11.58                   |
| TTR3   | 11.63                 | 11.61                  | 11.26                  | 11.19                   |
| BGO2   | 11.88                 | 11.81                  | 11.58                  | 11.46                   |
| BGO3   | 11.99                 | 11.99                  | 11.59                  | 11.54                   |

With a similar setup, latencies of other modules with LVDS-LINK connectors (LTPI and LTP) have also been measured. The only difference in the setup is that the module under test is LTP/LTPI, instead of the ALTI. Results are shown in Table 7.1.2 and Table 7.1.3 for the LTPI and the LTP, respectively.

**Table 7.1.2. Cable-to-cable latencies of TTC signals for the LTPI module.**

| SIGNAL | CTP/CTP<br>DELAY [NS] | CTP/LTP<br>DELAY [NS] | LTP/CTP<br>DELAY [NS] | LTP/LTP<br>DELAY [NS] |
|--------|-----------------------|-----------------------|-----------------------|-----------------------|
| BC     | 10.89                 | 13.74                 | 13.29                 | 8.76                  |
| ORB    | 13.99                 | 12.43                 | 17.41                 | 9.16                  |
| L1A    | 14.66                 | 13.28                 | 17.87                 | 8.83                  |
| TTR1   | 14.48                 | 11.27                 | 16.56                 | 9.49                  |
| TTR2   | 14.39                 | 11.27                 | 16.42                 | 9.39                  |
| TTR3   | 14.67                 | 11.51                 | 16.69                 | 9.26                  |
| BGO2   | 14.61                 | 12.72                 | 18.76                 | 9.13                  |
| BGO3   | 14.88                 | 13.09                 | 18.51                 | 9.02                  |

**Table 7.1.3. Cable-to-cable latencies of TTC signals for the LTP module.**

| SIGNAL | CTP/CTP<br>DELAY [NS] |
|--------|-----------------------|
| BC     | 4.63                  |
| ORB    | 4.83                  |
| L1A    | 4.48                  |
| TTR1   | 11.60                 |
| TTR2   | 11.18                 |
| TTR3   | 11.94                 |
| BGO2   | 12.31                 |
| BGO3   | 12.59                 |

The results of these measurements are also expected. In the LTPI, CTP output link is preceded by a fine-tune delay chip (0.5 ns step), which can be used to phase shift all TTC signals [7], except the BC. Also, in the LTP, propagation delays are lower for BC, ORB and L1A because of PECL circuitry instead of TTL (as for the rest of the TTC signals) [5].

Based on the results that were shown, one can compare latencies between systems composed of legacy TTC modules and corresponding replacement systems based on the ALTI module. Here are some comparisons of L1A latency in some common configurations:

- **ALTI** is about **7.5ns slower** than **LTP**
- **ALTI** is about **10ns faster** than **LTPI + LTP**
- **ALTI + ALTI** is about **2ns faster** than **LTPI + LTP + LTP**

In these calculations, short 0.5m LVDS-LINK cables are assumed between the modules. Based on the measurements, LVDS-LINK cables introduce the propagation delay of about 5ns per meter. That corresponds to about 2.5ns delay for each 0.5m cable.

The comparisons show that ALTI-based system is faster in a typical TTC partition daisy chain of length two, due to the smaller number of interconnecting LVDS-LINK cables being used in the setup.

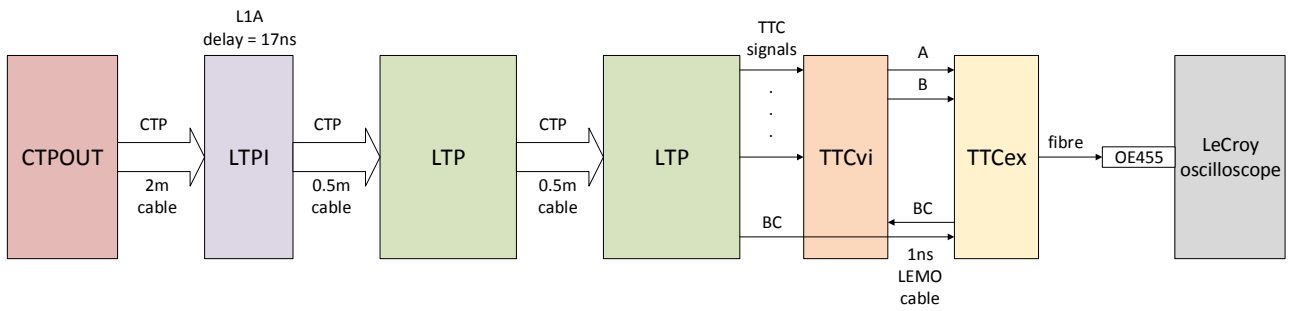
## 7.2. Level-1 Accept latency: LAr daisy chain

Level-1 Accept latency from the CTP to the TTC stream is one of the most important performance parameters in the ATLAS experiment. As we discussed earlier, propagating the trigger to the front-end electronics as early as possible is of the utmost importance. Actually, the liquid Argon calorimeter sub-detector (LAr) is the sub-detector that is the most latency-critical. That is why the Level-1 Accept latency for this particular TTC configuration was measured and compared to the possible ALTI-based replacement setup. The LAr sub-detector uses the daisy chain of two TTC partitions, as shown on Figure 2.2.2.

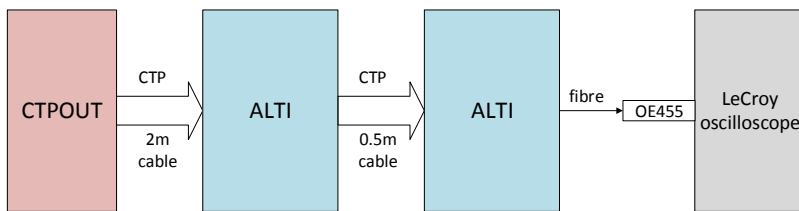
Shown on Figure 7.2.1 (a) and (b) are the legacy LAr setup and a corresponding ALTI-based setup, respectively. The latency of L1A in the TTC stream is measured on the second stage of the daisy chain, since only this stage is latency critical.

Legacy setup mimics the actual current configuration in the experiment. Shortest available LVDS-LINK cables were used, the ones of length 0.5m. The same is true for coaxial cables between the second LTP, TTCvi and TTCex: cables of 1ns and 2ns delay were used. In the LAr setup, LTPI is used for delaying the L1A exactly 17ns (34 steps of 0.5ns each), in addition to the equalization function. This is also shown on Figure 7.2.1 (a). The only major difference in the actual experiment is the length of the LVDS-LINK cable from the CTP: they are longer than the 2m cables that were used in this measurement, and the use of the LTPI for equalization is crucial.

In the replacement setup, two legacy partitions are substituted with two cascaded ALTI modules. The interface to the CTP and the oscilloscope is the same as in the legacy setup. Fibre lengths and propagation delays are matched in both setups, though different patch cords were used because of different transmitter connectors in the TTCex and ALTI.



(a)



(b)

Figure 7.2.1. Daisy-chained TTC partitions of LAR: (a) legacy setup, (b) proposed replacement setup based on two ALTI modules.

Bunch clock phases at the optical TTC output of both systems were aligned by configuring the PLL of the ALTI jitter cleaner. Then, relative comparison of two setups is possible, with the ORB signal from CTPOUT as a reference. Propagation delays from this reference signal to the Level-1 Accept in the TTC stream waveform were measured.

The waveforms from which the delays were measured are shown on Figure 7.2.2, Figure 7.2.3 and Figure 7.2.4. Figure 7.2.2 and Figure 7.2.3 refer to the legacy system, without and with the 17ns L1A delay induced by the LTPI, respectively. For the ALTI-based system with the optimal clock phase chosen for L1A input synchronization, the measurement is shown on Figure 7.2.4.

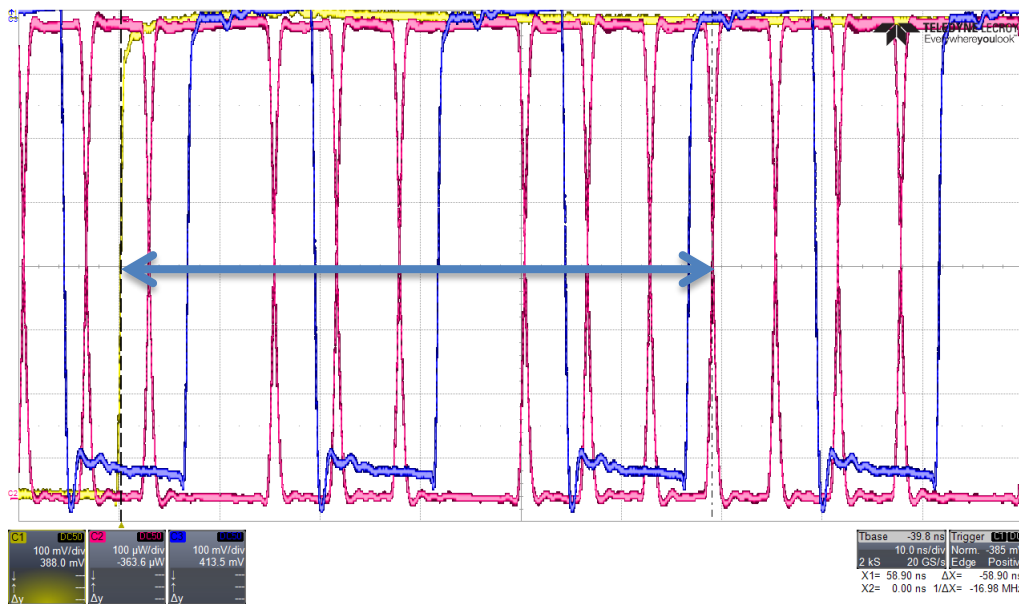


Figure 7.2.2. Level-1 Accept latency for the legacy LAR setup and LTPI delay = 0ns: 59ns.

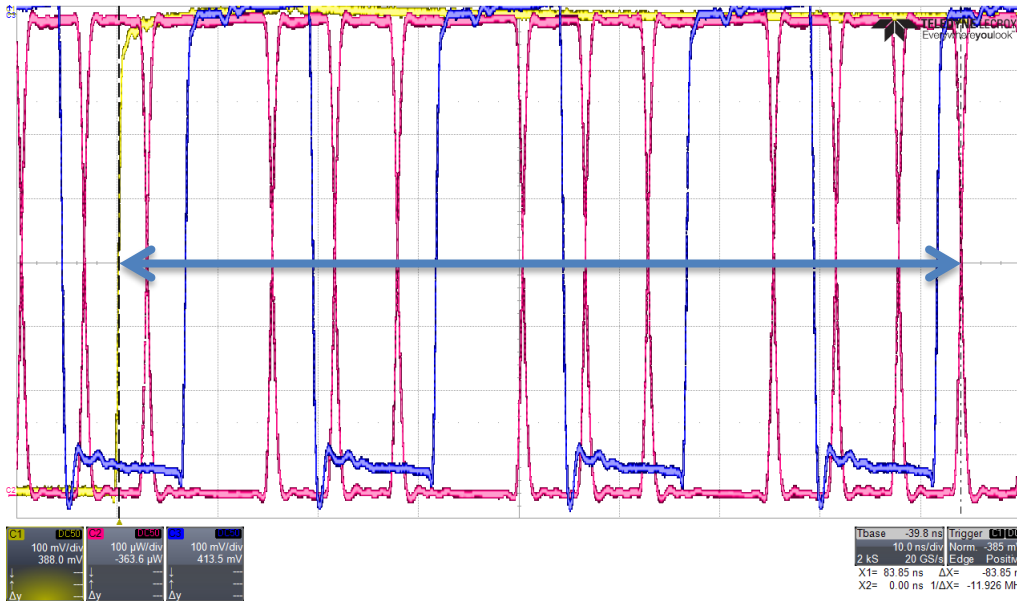


Figure 7.2.3. Level-1 Accept latency for the legacy LAr setup and LTPI delay = 17ns: 84ns.

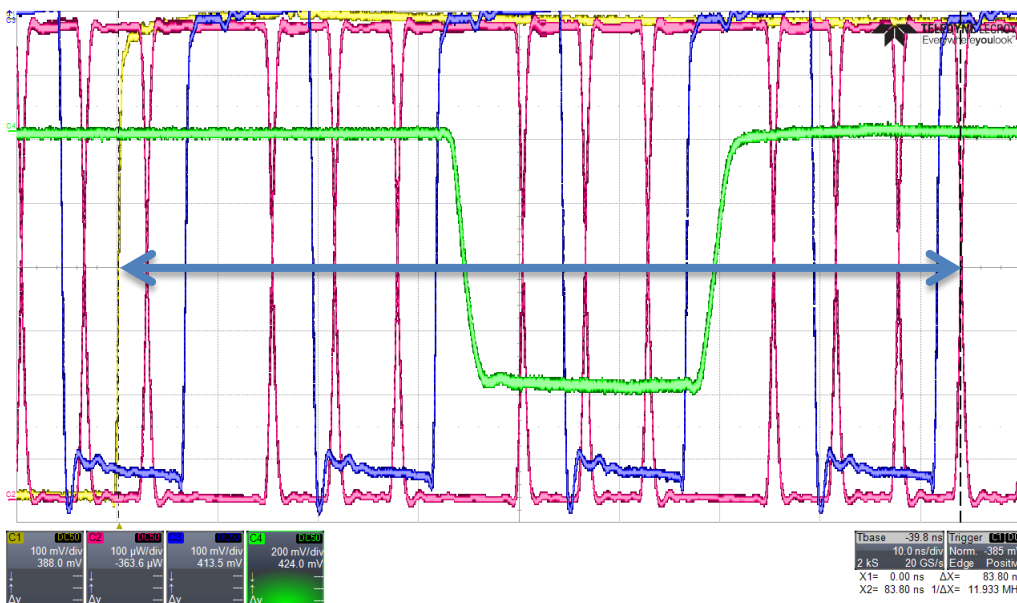


Figure 7.2.4. Level-1 Accept latency for the ALTI-based LAr setup and optimal input synchronization: 84ns.

For the legacy TTC setup, the latency is either 59ns or 84ns (L1A jumps to the next bunch crossing), depending on the L1A delay that is used in the LTPI (0ns or 17ns, respectively). Latency of L1A for this setup could also be up to 1BC longer than this if a bigger L1A delay was used, since 30ns is the maximum possible delay that the LTPI supports.

For the ALTI-based setup, the latency is either 84ns or 109ns (L1A jumps to the next bunch crossing), depending on which clock phase is chosen to synchronize the incoming L1A signal.

To conclude, the L1A latency can be the same (84ns) as for the legacy TTC modules using the settings in the experiment. Depending on the L1A signal phase, the delay could also be 1BC longer for the new system.

### 7.3. TTC stream and recovered clock jitter

The level of jitter in the optical TTC stream that the ALTI introduces had to be addressed, in order to see if the receiver modules of the sub-detectors can cope with it. Comparison with the legacy TTCex-based setup was also made. On Figure 7.3.1, the two setups are shown. The first setup, shown on Figure 7.3.1 (a), measures the TTC stream jitter in a typical partition composed of LTP, TTCvi and TTCex. In the second setup, shown on Figure 7.3.1 (b), this legacy TTC partition is replaced by a single ALTI module. In fact, three different setups were compared, since ALTI-based setups with and without using the on-board jitter cleaner were tested.

Both setups use a common bunch clock, one that has been injected through the BC LEMO input of an additional LTP module. The clock comes from the clock generator with a modulation input for adding jitter. The nominal value of the clock signal frequency is 40.079MHz, which is the standard LHC clock frequency. The modulation signal in the form of white noise is provided with a signal generator. Increasing the peak-to-peak voltage of the white noise generator has an effect of adding jitter to the bunch clock.

For both setups, the optical TTC stream is sent to the TTCrq receiver mezzanine board [37]. This is a combination of TTCrx receiver and a Quartz-crystal based PLL (QPLL) [38] ASICs. The QPLL was designed for jitter cleaning applications in the LHC to accompany the TTCrx.

The levels of jitter in the TTC stream and the bunch clock recovered from it by the TTCrq are then measured by the LeCroy oscilloscope and the Agilent Phase Noise Analyzer (PNA) [39].

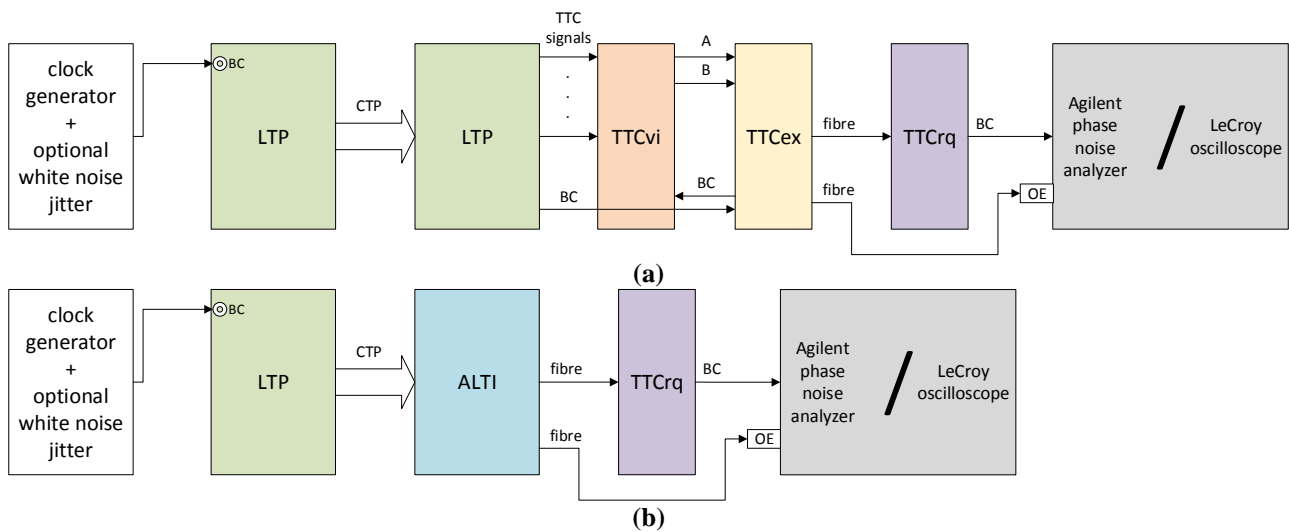
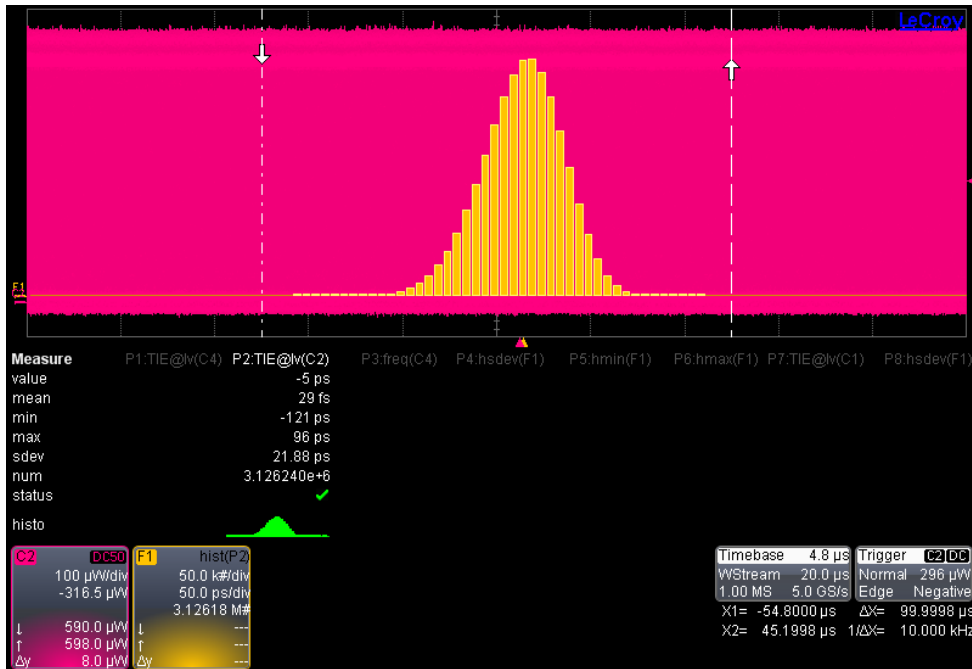


Figure 7.3.1. Setups for the TTC stream jitter measurement where the optical transmitter module is: (a) TTCex, (b) ALTI.

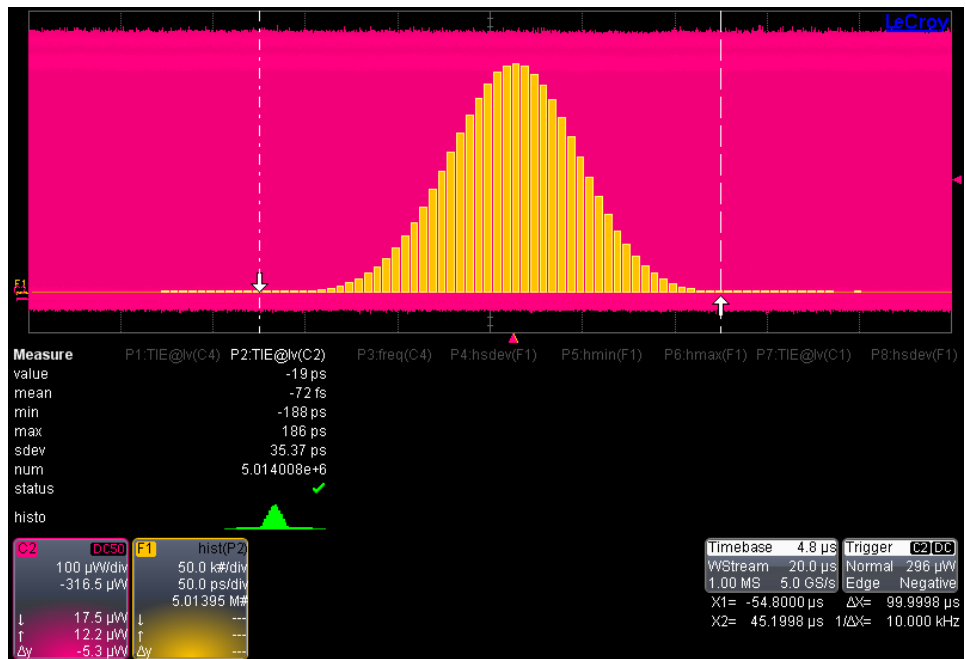
#### 7.3.1. Oscilloscope measurements

For measuring the jitter, the Time Interval Error (TIE) method of the digital LeCroy oscilloscope has been used. The TIE of a particular rising or falling edge is the deviation of that edge from its ideal position. The ideal signal is created based on the average estimate of the signal period. The measurements are accumulated by the oscilloscope and shown on its display overlaid on top of the signal waveform. This gives a Gaussian-shaped distribution histogram, with the mean value ideally equal to zero. Standard deviation of this distribution is a good measure of jitter and is expressed in pico-seconds. Usually, this metric is called the RMS jitter. The effect of adding white-noise jitter is clearly shown on Figure 7.3.2: the histogram is more spread when there is more jitter, i.e. it has larger standard deviation or RMS jitter.





(a)



(b)

**Figure 7.3.2. TTC stream RMS jitter oscilloscope measurements for ALTI setup without the jitter cleaner: (a) without added jitter, RMS jitter equals 21.9ps, (b) with added jitter, RMS jitter equals 35.4ps.**

The results of the oscilloscope measurements of the TTC stream jitter for various setups are shown in Table 7.3.1. Similarly, results for the jitter of the recovered clock are shown in Table 7.3.2. Please note that the TTC stream was idle (no triggers and commands were sent) for these measurements.

**Table 7.3.1. TTC stream jitter for different setups (oscilloscope measurements).**

| SETUP          | TTC STREAM RMS JITTER [PS]<br>(WITHOUT ADDED JITTER) | TTC STREAM RMS JITTER [PS]<br>(WITH ADDED JITTER) |
|----------------|--|---|
| TTCex          | 7.9  | 8   |
| ALTI, JC = OFF | 21.9   | 35.4  |
| ALTI, JC = ON  | 16.7   | 16.7  |

**Table 7.3.2. Recovered clock jitter for different setups (oscilloscope measurements).**

| SETUP          | RECOVERED CLOCK RMS JITTER [PS]<br>(WITHOUT ADDED JITTER) | RECOVERED CLOCK RMS JITTER [PS]<br>(WITH ADDED JITTER) |
|----------------|---|--|
| TTCex          | 8.1   | 8.1  |
| ALTI, JC = OFF | 8.0   | 8.1  |
| ALTI, JC = ON  | 8.0   | 8.1  |

Based on the measurements that were shown, we can draw several conclusions. First of all, the ALTI jitter cleaner removes all the added jitter. However, it cannot remove intrinsic jitter in the ALTI. On the other hand, TTCex also removes all the added jitter with its PLL, and this setup has a lower overall intrinsic jitter compared to ALTI. But, although the overall jitter level is higher in the ALTI setup compared to the TTCex setup, the TTCrq which contains the QPLL can handle these levels of jitter for both systems, as can be seen in the recovered clock jitter. From this point on, it was decided to keep using the ALTI jitter cleaner.

The TIE jitter measurements were also repeated on a non-idle TTC stream. Random patterns with the 100kHz L1-Accept rate and heavy B-channel activity (four asynchronous long commands following each L1-Accept) were used. However, it was observed that the A and B-channel activity does not affect the jitter of the TTC stream in the ALTI, nor in the TTCex.

Another measurement was done for the ALTI TTC encoder driven from the same ALTI board using its pattern generation memory. The idea was to investigate if this increases the jitter. However, this does not have an effect on the jitter, either.

### **7.3.2. Phase noise analyzer measurements**

Measuring the jitter with the oscilloscope can be very sensitive to the measurement settings. For example, if the time division is too small (waveform zoomed in too much), low-frequency jitter will not be included in the measurement, thus giving lower RMS jitter. A more accurate measurement of jitter was done with the Agilent PNA, which measures the frequency spectrum of the jitter. In addition, this tool also measures the RMS jitter by integrating the spectrum.

The phase noise analyzer measures the "cleanliness" of the periodic, 50% duty cycle clock signal. In order to use the phase noise analyzer on the TTC stream, the L1A signal was set to be always active, which results in a TTC output bit pattern that corresponds to an 80MHz clock signal, as can be seen on Figure 2.2.1.

From the oscilloscope measurements we have understood that the added jitter is successfully removed by TTCex PLL and the ALTI jitter cleaner. This was also confirmed by the PNA measurements. The results shown below were obtained without adding the white-noise jitter. Common input clock was the internal oscillator in the LTP with the RMS jitter of 6.6ps. The results of the PNA measurements of TTC stream jitter for various setups are shown in Table 7.3.3. Similarly, results for the jitter of the recovered clock are shown in Table 7.3.4.

**Table 7.3.3. TTC stream jitter for different setups (PNA measurements).**

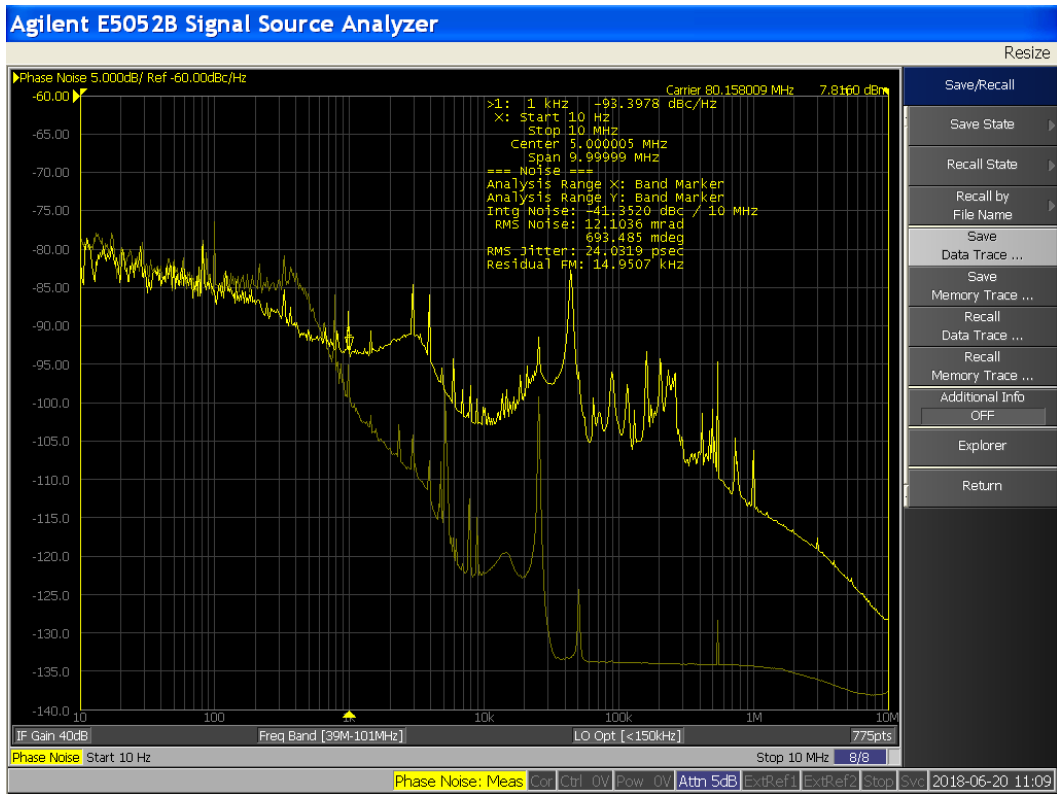
| SETUP          | TTC STREAM RMS JITTER [PS]<br>(WITHOUT ADDED JITTER) |
|----------------|--|
| TTCex          | 5.1  |
| ALTI, JC = OFF | 24.0   |
| ALTI, JC = ON  | 19.6   |

**Table 7.3.4. Recovered clock jitter for different setups (PNA measurements).**

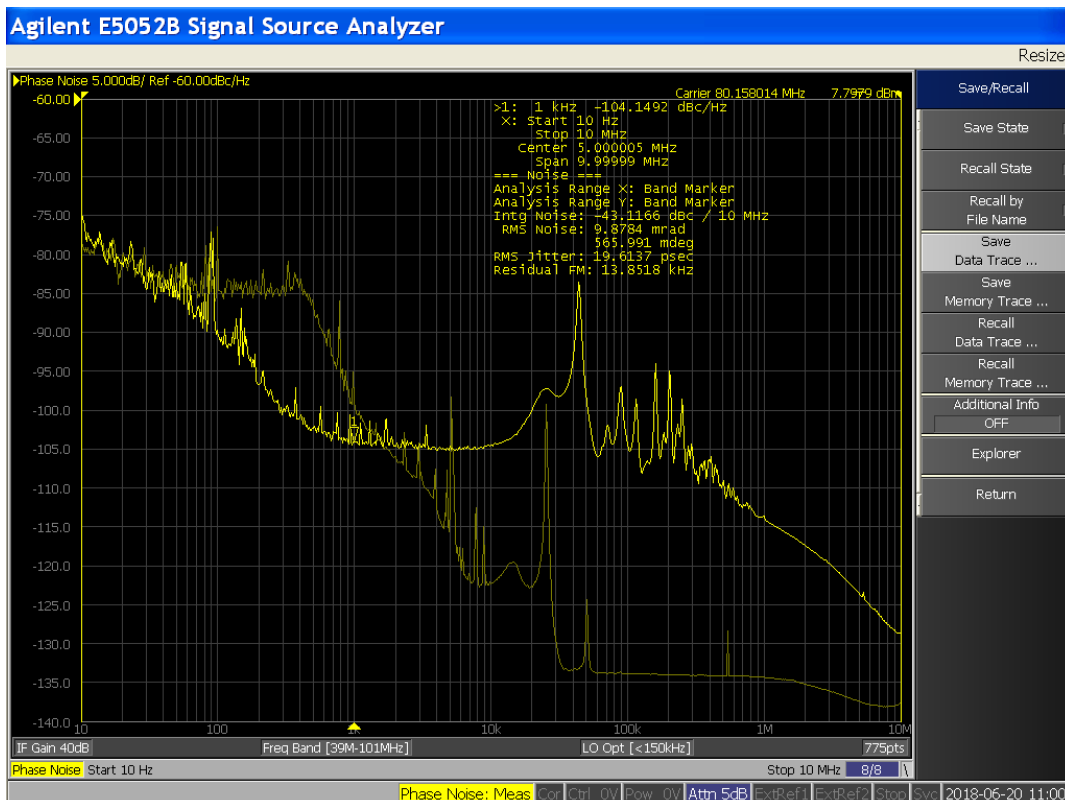
| SETUP          | RECOVERED CLOCK RMS JITTER [PS]<br>(WITHOUT ADDED JITTER) |
|----------------|---|
| TTCex          | 5.7   |
| ALTI, JC = OFF | 7.6   |
| ALTI, JC = ON  | 6.0   |

The same conclusions follow as the ones obtained after the oscilloscope measurements, although the numbers are slightly different. To compare the TTC stream jitter spectrums of the three setups, please refer to Figure 7.3.3. On Figure 7.3.3 (a), ALTI without using the jitter cleaner is compared against TTCex. On Figure 7.3.3 (b), ALTI with the jitter cleaner is compared against TTCex. Similarly, jitter spectrums of the recovered clock are compared on Figure 7.3.4 (a) and Figure 7.3.4 (b).

By looking more closely on the ALTI jitter spectrums, one can clearly see a few spikes in the ~30kHz to ~300kHz range. This could be related to the power supply noise on the ALTI, but certainly requires further investigation. The on-chip PLL settings might also provide a way to reduce the intrinsic ALTI jitter. So, there might be room for improvement on the ALTI intrinsic jitter by some modifications to the PCB, or the FPGA firmware.

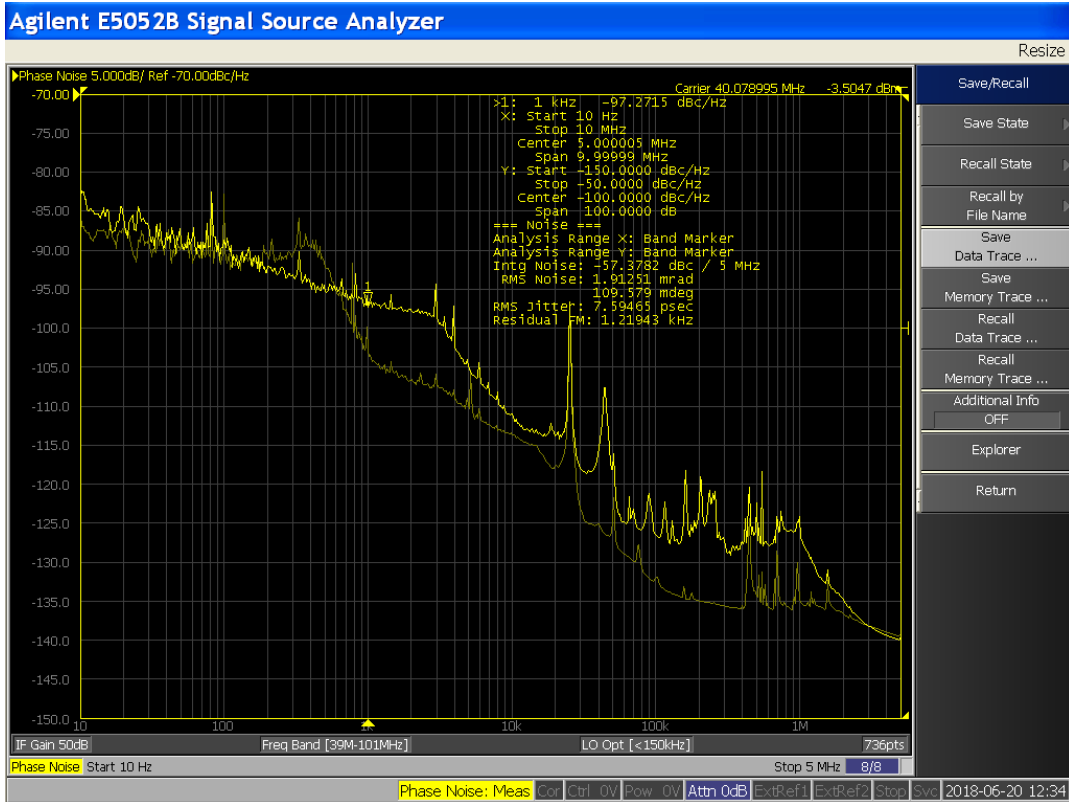


(a)

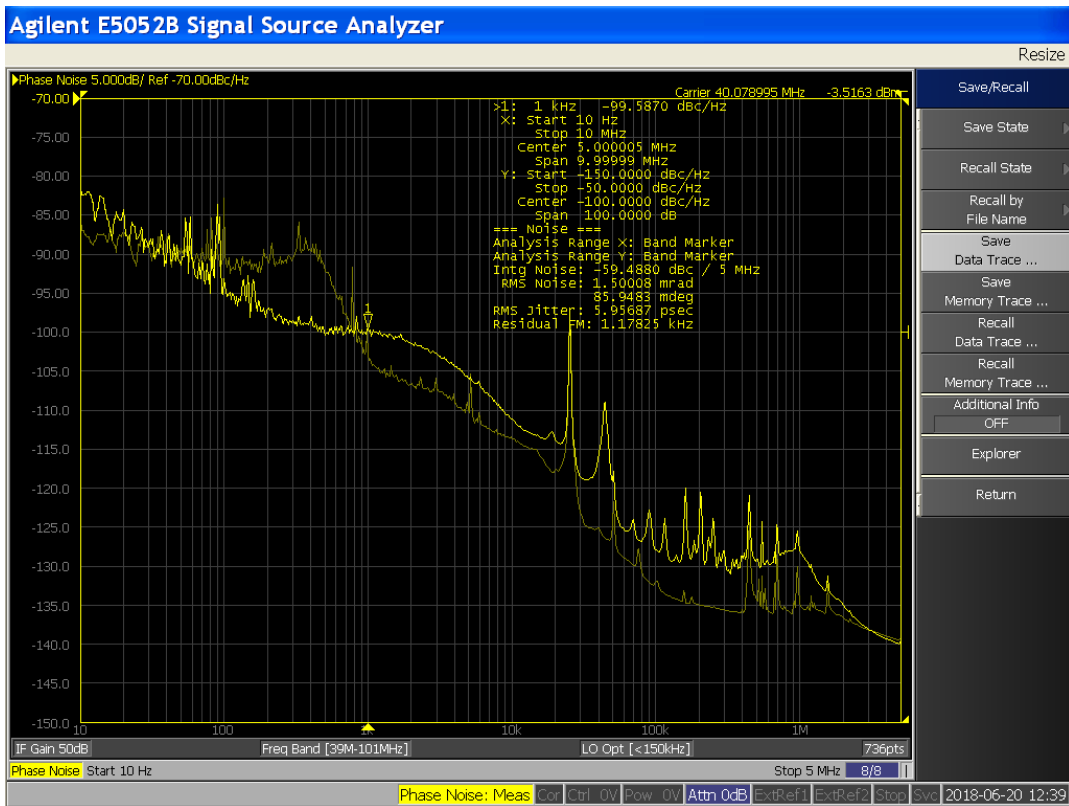


(b)

Figure 7.3.3. TTC stream jitter spectrum comparisons (a) ALTI without the jitter cleaner (bold) against TTCex (pale), (b) ALTI with the jitter cleaner (bold) against TTCex (pale).



(a)



(b)

Figure 7.3.4. Recovered clock jitter spectrum comparisons (a) ALTI without the jitter cleaner (bold) against TTCex (pale), (b) ALTI with the jitter cleaner (bold) against TTCex (pale).

## 8. CONCLUSION

The low-level software for the ALTI module has been developed. All its features are fully available to the users in the interactive menu program. The software is also available to the other L1CT colleagues who are using it in order to develop the run control applications for integration of the ALTI into the experiment. Besides the low level API, utilities in the form of test programs and scripts have been developed. These have been used for testing the ALTI functionalities like pattern generation, snapshot memory, TTC decoder, etc. An automated production testbench for the upcoming ALTI modules has also been developed. This testbench is used to test all input/output paths of the signals through the module, and thus can be used to identify soldering assembly issues.

All of the devices and interfaces on the available prototype boards have been fully tested. That is, the tests have been performed on a total of seven ALTI prototype modules that have been fully-assembled until August 2018. These module tests helped to find several design issues. The list of modifications to be done in the next series of pre-production ALTI modules has been noted and a new version of the schematics and the layout has been published for both the motherboard and the mezzanine board.

Five out of seven modules have passed all the automated tests. On the other two modules, the automated tests have shown problems with particular TTC signal paths. These modules are currently being investigated, and the fact that the problematic lines have been pinpointed will certainly help to understand what the problems are. Some working modules were made available to colleagues in the L1CT team in order to perform additional tests and write the high-level run control software. The other modules that have passed the tests will be lent to the sub-detectors to perform the initial tests with the ALTI module in their setup. In particular, it is planned to have tests with the LAr test stand in September 2018.

The performance measurements have shown how the ALTI module compares against the legacy TTC system. In particular, the Level-1 Accept latency in the LAr setup, using a daisy chain of two modules, is the same for the ALTI and the TTC legacy modules, if the L1A input synchronization has been chosen to be optimal.

The jitter measurements have shown that the jitter cleaner in the ALTI can effectively remove the low-frequency jitter. This encouraged the use of the jitter cleaner in the firmware, which was initially put in the design as a safety measure. Also, from the jitter measurements it has been concluded that the TTC stream jitter is higher in the ALTI-based system compared to the legacy TTC-ex based system. To be precise, the RMS jitter in the ALTI TTC stream is 19.6ps, compared to 5.1ps in the legacy system, using the same input clock with RMS jitter of 6.6ps. However, the tests with TTC receiver modules have shown that they can effectively remove the intrinsic ALTI jitter. The clock recovered from the TTC stream basically has the same RMS jitter in both setups: 6.0ps for the ALTI-based system, compared to 5.7ps for the legacy system. Nevertheless, the jitter measurements have shown the need to further investigate the ALTI design to see if the intrinsic jitter can be reduced.

Another possible issue to be addressed in the future is the optical power requirements needed by the sub-detectors, since the ALTI SFP modules provide lower power than the TTCex lasers. Several SFP modules with the same pin-out and higher optical power have already been ordered and tested. Going forward, they could be used if the sub-detector tests show a need to increase the optical power.

# BIBLIOGRAPHY

- [1] ATLAS Fact Sheet (<https://cds.cern.ch/record/1457044/files/ATLAS%20fact%20sheet.pdf>)
- [2] S. Ask, D. Berge, P. Borrego-Amaral, D. Caracinha, N. Ellis, P. Farthouat, P. Gällnö, S. Haas, J. Haller, P. Klofver, A. Krasznahorkay, A. Messina, C. Ohm, T. Pauly, M. Perantoni, H. Pessoa Lima Junior, G. Schuler, D. Sherman, R. Spiwoks, T. Wengler, J. M. de Seixas and R. Torga Teixeira, "ATLAS central level-1 trigger logic and TTC system", *Journal of Instrumentation (JINST)*, 2008. (<http://iopscience.iop.org/article/10.1088/1748-0221/3/08/P08002/pdf>)
- [3] S. Baron, "Timing, Trigger and Control (TTC) Systems for the LHC" (<http://ttc.web.cern.ch/TTC/intro.html>, 27.07.2018.)
- [4] J. Christiansen, A. Marchioro, P. Moreira and T. Toifl, "TTCrx Reference Manual, A Timing Trigger and Control Receiver ASIC for LHC Detectors" ([http://ttc.web.cern.ch/TTC/TTCrx\\_manual3.9.pdf](http://ttc.web.cern.ch/TTC/TTCrx_manual3.9.pdf))
- [5] P. Gällnö, "ATLAS Local Trigger Processor - LTP, Technical description and users manual" ([https://edms.cern.ch/ui/file/551992/2/LTP\\_manual\\_051.pdf](https://edms.cern.ch/ui/file/551992/2/LTP_manual_051.pdf))
- [6] U.S. Department of Energy, Office of Energy Research, Office of Health and Environmental Research, "Standard NIM Instrumentation System", *U.S. NIM Committee*, May 1990. (<https://cds.cern.ch/record/2026631/files/nim-standard.pdf>)
- [7] P. Farthouat, "LTP Interface" ([https://twiki.cern.ch/twiki/bin/viewfile/Main/MyATLASDocumentation?rev=1.1;filename=Interface\\_spec-v4.3.pdf](https://twiki.cern.ch/twiki/bin/viewfile/Main/MyATLASDocumentation?rev=1.1;filename=Interface_spec-v4.3.pdf))
- [8] P. Farthouat, P. Gällnö, "TTC-VMEbus Interface" (<http://ttc.web.cern.ch/TTC/TTCviSpec.pdf>)
- [9] B. G. Taylor, "TTC laser transmitter (TTCex, TTCtx, TTCmx) User Manual" (<http://ttc.web.cern.ch/TTC/TTCtxManual.pdf>)
- [10] M. Joos, "An introduction to VMEbus" (<https://indico.cern.ch/event/68278/contributions/1234555/attachments/1024465/1458672/VMEbus.pdf>)
- [11] Concurrent Technologies VP-E24 VME single-board computer based on Intel Atom E3800 Processor ([http://www.gocct.com/wp-content/uploads/2017/09/vpe2xmsd\\_0617.pdf](http://www.gocct.com/wp-content/uploads/2017/09/vpe2xmsd_0617.pdf))
- [12] S. Haas, P. Kuzmanovic, T. Pauly, V. Ryzhov, R. Spiwoks, "ALTI Specification", *ALTI specification review and preliminary design review*, June 2008. ([https://indico.cern.ch/event/735376/attachments/1673160/2684944/ALTI-Specification\\_Rev\\_1\\_2.pdf](https://indico.cern.ch/event/735376/attachments/1673160/2684944/ALTI-Specification_Rev_1_2.pdf))
- [13] "EDA-03627 ALTI - Atlas LTrg Interface", *CERN EDMS* (<https://edms.cern.ch/ui/#!/master/navigator/project?P:1844187571:1844187571:subDocs>, 18.08.2018.)
- [14] "EDA-03628 ALTI - Atlas LTrg Interface - Mezzanine", *CERN EDMS* (<https://edms.cern.ch/ui/#!/master/navigator/project?P:1059917782:1059917782:subDocs>, 18.08.2018.)
- [15] Xilinx 7 Series FPGAs Data Sheet: Overview ([https://www.xilinx.com/support/documentation/data\\_sheets/ds180\\_7Series\\_Overview.pdf](https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf))
- [16] Samtec QSH/QTH high-speed ground plane socket connectors ([http://suddendocs.samtec.com/catalog\\_english/qsh.pdf](http://suddendocs.samtec.com/catalog_english/qsh.pdf))
- [17] Texas Instruments DS10CP154A 1.5Gbps 4x4 Crosspoint Switch (<http://www.ti.com/lit/ds/symlink/ds10cp154a.pdf>)

- [18] Silicon Labs SI5344 4-channel jitter attenuating clock multiplier (<https://www.silabs.com/documents/public/data-sheets/Si5345-44-42-D-DataSheet.pdf>)
- [19] Analog Devices AD8123 Triple Differential Receiver with Adjustable Line Equalization (<http://www.analog.com/media/en/technical-documentation/data-sheets/AD8123.pdf>)
- [20] Analog Devices AD5305 Quad Voltage Output 8-bit DAC ([http://www.analog.com/media/en/technical-documentation/data-sheets/AD5305\\_5315\\_5325.pdf](http://www.analog.com/media/en/technical-documentation/data-sheets/AD5305_5315_5325.pdf))
- [21] Analog Devices ADCMP604 Rail-to-Rail Single-Supply LVDS Comparator ([http://www.analog.com/media/en/technical-documentation/data-sheets/ADCMP604\\_605.pdf](http://www.analog.com/media/en/technical-documentation/data-sheets/ADCMP604_605.pdf))
- [22] Cypress CY7C10612G 16Mbit (1Mx16) Static RAM (<http://www.cypress.com/file/46696/download>)
- [23] W-Optics SAA-xAF1-111 1250Mb/s Dual Optical Duplex LC SFP Transmitter ([https://indico.cern.ch/event/735376/attachments/1671862/2682296/W-Optics\\_SAA-1AF1-111SFP\\_2T\\_GbE\\_SM\\_10km\\_V04.pdf](https://indico.cern.ch/event/735376/attachments/1671862/2682296/W-Optics_SAA-1AF1-111SFP_2T_GbE_SM_10km_V04.pdf))
- [24] Finisar FTLF1323P1xTL 155Mb/s Duplex LC SFP Transceiver ([https://www.finisar.com/sites/default/files/downloads/finisar\\_ftlf1323p1xtl\\_oc-3\\_lr-1\\_stm\\_l-1.1\\_rohs\\_compliant\\_pluggable\\_sfp\\_transceiver\\_product\\_specification\\_0.pdf](https://www.finisar.com/sites/default/files/downloads/finisar_ftlf1323p1xtl_oc-3_lr-1_stm_l-1.1_rohs_compliant_pluggable_sfp_transceiver_product_specification_0.pdf))
- [25] Analog Devices ADN2814 Clock and Data Recovery IC with Integrated Limiting Amp (<http://www.analog.com/media/en/technical-documentation/data-sheets/ADN2814.pdf>)
- [26] Maxim Integrated MAXM17515 5A High-Efficiency Power Module (<https://datasheets.maximintegrated.com/en/ds/MAXM17515.pdf>)
- [27] Traco Power THN 15-1211 15W DC/DC converter (<https://www.tracopower.com/products/thn15.pdf>)
- [28] Linear Technology LTC2991 Octal Voltage, Current and Temperature Monitor (<http://www.analog.com/media/en/technical-documentation/data-sheets/2991ff.pdf>)
- [29] Maxim Integrated MAX1617A Remote/Local Temperature Sensor with SMBus Serial Interface (<https://datasheets.maximintegrated.com/en/ds/MAX1617A.pdf>)
- [30] Texas Instruments TCA9546A Low Voltage 4-Channel I2C Switch (<http://www.ti.com/lit/ds/symlink/tca9546a.pdf>)
- [31] Richard Herveille, "I2C-Master Core Specification", *OpenCores* ([https://opencores.org/websvn/filedetails?repname=i2c&path=%2Fi2c%2Ftrunk%2Fdoc%2Fi2c\\_specs.pdf](https://opencores.org/websvn/filedetails?repname=i2c&path=%2Fi2c%2Ftrunk%2Fdoc%2Fi2c_specs.pdf))
- [32] Maxim Integrated, "Embedding the 1-Wire Master in FPGAs or ASICs", *Application Note* (<https://pdfserv.maximintegrated.com/en/an/AN119.pdf>)
- [33] Randal Kuramoto, "QuickBoot Method for FPGA Design Remote Update", *Application Note* ([https://www.xilinx.com/support/documentation/application\\_notes/xapp1081-quickboot-remote-update.pdf](https://www.xilinx.com/support/documentation/application_notes/xapp1081-quickboot-remote-update.pdf))
- [34] SRecord 1.64 tool for manipulating EPROM load files (<http://srecord.sourceforge.net/>, 19.08.2018.)
- [35] Teledyne LeCroy WaveRunner 104MXi-A 1GHz 10GS/s oscilloscope ([http://teledynelecroy.com/japan/pdf/cata/waverunner\\_xi-a\\_spec.pdf](http://teledynelecroy.com/japan/pdf/cata/waverunner_xi-a_spec.pdf))
- [36] Teledyne LeCroy OE455 Optical-to-Electrical Converter ([http://cdn.teledynelecroy.com/files/manuals/o\\_e\\_maunal\\_insert.pdf](http://cdn.teledynelecroy.com/files/manuals/o_e_maunal_insert.pdf))
- [37] P. Moreira, "TTCrq Manual" (<https://proj-qpll.web.cern.ch/proj-qpll/images/manualTTCrq.pdf>)
- [38] P. Moreira, "QPLL Manual – Quartz Crystal Based Phase-Locked Loop for Jitter Filtering Application in LHC" (<https://proj-qpll.web.cern.ch/proj-qpll/images/qpllManual.pdf>)



[39] Agilent E505 2B Signal Source Analyzer  
(<http://literature.cdn.keysight.com/litweb/pdf/5989-7273EN.pdf>)

## LIST OF ABBREVIATIONS

|        |  |
|--------|--|
| ALTI   | <i>ATLAS Local Trigger Interface</i>   |
| ATLAS  | <i>A Toroidal LHC ApparatuS</i>  |
| BCR    | <i>Bunch Counter Reset</i>   |
| CERN   | <i>Conseil Européen pour la Recherche Nucléaire (European Organization for Nuclear Research)</i> |
| CTP    | <i>Central Trigger Processor</i>   |
| ECR    | <i>Event Counter Reset</i>   |
| HLT    | <i>High Level Trigger</i>  |
| L1CT   | <i>Level-1 Central Trigger</i>   |
| LAr    | <i>Liquid Argon calorimeter sub-detector</i>   |
| LHC    | <i>Large Hadron Collider</i>   |
| LTP    | <i>Local Trigger Processor</i>   |
| LTPI   | <i>Local Trigger Processor Interface</i>   |
| MDR    | <i>Mini Delta Ribbon</i>   |
| MMCM   | <i>Mixed-Mode Clock Manager</i>  |
| NIM    | <i>Nuclear Instrumentation Module</i>  |
| PNA    | <i>Phase Noise Analyzer</i>  |
| QPLL   | <i>Quartz-crystal based PLL</i>  |
| RCD    | <i>ReadOut Driver Crate DAQ</i>  |
| SBC    | <i>Single-Board Computer</i>   |
| SFP    | <i>Small Form-factor Pluggable</i>   |
| TDAQ   | <i>Trigger and Data Acquisition</i>  |
| TTC    | <i>Timing, Trigger and Control</i>   |
| TTCex  | <i>TTC Encoder/Transmitter</i>   |
| TTCrq  | <i>TTCrx and QPLL</i>  |
| TTCrx  | <i>TTC Receiver</i>  |
| TTCvi  | <i>TTC VMEbus Interface (TTCvi)</i>  |
| VMEbus | <i>Versa Module Europa bus</i>   |

# LIST OF FIGURES

|   |    |
|---|----|
| Figure 2.2.1. Multiplexing and encoding of the TTC channels A and B.....  | 4  |
| Figure 2.2.2. Current TTC distribution network in the ATLAS experiment [12]. .....  | 5  |
| Figure 2.3.1. Legacy TTC modules, front panel view: (a) LTPI [7], (b) LTP [5], (c) TTCvi [8], (d) TTCex [9].....  | 7  |
| Figure 2.3.2. Frame format for short TTC commands.....  | 9  |
| Figure 2.3.3. Frame format for long TTC commands.....   | 9  |
| Figure 3.1.1. The ALTI module, front panel view [12]. .....   | 12 |
| Figure 3.2.1. Fully-assembled prototype ALTI module. ....   | 14 |
| Figure 3.2.2. ALTI functional block diagram [12]. .....   | 14 |
| Figure 3.2.3. Example of the generic routing of a TTC signal through the cross-point switch: Level-1 Accept. ....   | 15 |
| Figure 3.2.4. ALTI clock distribution diagram. ....   | 16 |
| Figure 3.2.5. BGO0/TTR2 multiplexing on the input and BGO0/BGO2 multiplexing on the output. ....  | 17 |
| Figure 3.2.6. BGO1/TTR3 multiplexing on the input and BGO1/BGO3 multiplexing on the output. ....  | 18 |
| Figure 4.1. High-level functional block diagram of the ALTI firmware.....   | 21 |
| Figure 4.3.1. Pattern generation memory format. ....  | 24 |
| Figure 4.4.1. Snapshot memory format.....   | 24 |
| Figure 4.6.1. TTC spy memory format: "command" and "timestamp". ....  | 25 |
| Figure 4.9.1. Functional block diagram of the BUSY signal routing in the FPGA logic. ....   | 26 |
| Figure 4.9.2. Functional block diagram of the CALREQ signal routing in the FPGA logic.....  | 27 |
| Figure 5.2.1. A section of the <i>alti.xml</i> file containing a bitstring description for the snapshot memory entry. ....  | 29 |
| Figure 5.2.2. A section of the <i>alti.xml</i> file containing the I2C block.....   | 30 |
| Figure 5.2.3. The list of low-level API methods of block "SIG".....   | 31 |
| Figure 5.2.4. The list of low-level API methods of block "PAT".....   | 31 |
| Figure 5.2.5. The list of low-level API methods of block "SNP". ....  | 31 |
| Figure 5.3.1. Main menu of the ALTI menu program. ....  | 32 |
| Figure 5.4.1. A section of the ALTI configuration file containing parameters in a form of key/value pairs.....  | 33 |
| Figure 5.5.1. <i>testAltivME</i> program help with the full list of parameters. ....  | 34 |
| Figure 5.5.2. Typical use of the <i>testAltivME</i> program. ....   | 34 |
| Figure 5.5.3. <i>testAltInitial</i> program help with the full list of parameters .....   | 35 |
| Figure 5.5.4. Typical use of the <i>testAltInitial</i> program: (a) base address initialization, basic setup and check, (b) module configuration as a "CTP slave". .... | 36 |
| Figure 5.5.5. The QuickBoot mechanism [33].....   | 37 |
| Figure 5.5.6. <i>testAltQuickBoot</i> program help with the full list of parameters. ....   | 38 |
| Figure 5.5.7. Typical use of the <i>testAltQuickBoot</i> program .....  | 38 |
| Figure 5.5.8. <i>testAltCapture</i> program help with the full list of parameters. ....   | 39 |
| Figure 5.5.9. An example of the ALTI pattern input file used to run the pattern generator. ....   | 39 |
| Figure 5.5.10. Typical use of the <i>testAltCapture</i> program. ....   | 40 |
| Figure 5.5.11. Result of running the <i>ttcscope</i> program.....   | 41 |
| Figure 5.5.12. <i>testAltTtc</i> program help with the full list of parameters. ....  | 42 |
| Figure 5.5.13. Typical use of the <i>testAltTtc</i> program. ....   | 43 |

|  |    |
|--|----|
| Figure 5.5.14. <i>testAltiSync</i> program help with the full list of parameters.....  | 44 |
| Figure 5.5.15. Typical use of the <i>testAltiSync</i> program: without the LTPI delay.....   | 44 |
| Figure 5.5.16. Typical use of the <i>testAltiSync</i> program: with the LTPI delay of 4ns, resulting in the histogram shift.....   | 45 |
| Figure 6.1.1. Typical laboratory test setup with multiple ALTI and legacy TTC modules in the same VME crate.....   | 46 |
| Figure 6.2.1. Setup for testing of all input/output paths of TTC signals. ....   | 47 |
| Figure 6.2.2. Connection test results, LEMO BGO2 and BGO3 cabling. ....  | 49 |
| Figure 6.2.3. Connection test results, LEMO BGO0 and BGO1 cabling (multiplexed). ....  | 50 |
| Figure 7.1.1. CTP_IN->CTP_OUT path latency measurement for L1A: (a) delay of ALTI under test included, (b) ALTI module under test bypassed.....  | 52 |
| Figure 7.2.1. Daisy-chained TTC partitions of LAr: (a) legacy setup, (b) proposed replacement setup based on two ALTI modules. ....  | 55 |
| Figure 7.2.2. Level-1 Accept latency for the legacy LAr setup and LTPI delay = 0ns: 59ns. ....   | 55 |
| Figure 7.2.3. Level-1 Accept latency for the legacy LAr setup and LTPI delay = 17ns: 84ns. ....  | 56 |
| Figure 7.2.4. Level-1 Accept latency for the ALTI-based LAr setup and optimal input synchronization: 84ns. ....  | 56 |
| Figure 7.3.1. Setups for the TTC stream jitter measurement where the optical transmitter module is: (a) TTCex, (b) ALTI. ....  | 57 |
| Figure 7.3.2. TTC stream RMS jitter oscilloscope measurements for ALTI setup without the jitter cleaner: (a) without added jitter, RMS jitter equals 21.9ps, (b) with added jitter, RMS jitter equals 35.4ps. .... | 58 |
| Figure 7.3.3. TTC stream jitter spectrum comparisons (a) ALTI without the jitter cleaner (bold) against TTCex (pale), (b) ALTI with the jitter cleaner (bold) against TTCex (pale). ....                           | 61 |
| Figure 7.3.4. Recovered clock jitter spectrum comparisons (a) ALTI without the jitter cleaner (bold) against TTCex (pale), (b) ALTI with the jitter cleaner (bold) against TTCex (pale). ....                      | 62 |

## LIST OF TABLES

|  |    |
|--|----|
| Table 2.2.1. List of TTC signals. ....   | 6  |
| Table 3.1.1. LVDS-LINK connector pin-out [5]. ....   | 13 |
| Table 3.1.2. Front panel coaxial LEMO connectors. ....   | 13 |
| Table 3.2.1. The list of slave devices on the ALTI I2C network. ....   | 20 |
| Table 4.1. ALTI VMEbus address space map. ....   | 22 |
| Table 6.2.1. Standard configurations of ALTI modules in the connection test for various test paths.<br>..... | 48 |
| Table 7.1.1. Cable-to-cable latencies of TTC signals for the ALTI module. ....                               | 53 |
| Table 7.1.2. Cable-to-cable latencies of TTC signals for the LTPI module. ....                               | 53 |
| Table 7.1.3. Cable-to-cable latencies of TTC signals for the LTP module. ....                                | 53 |
| Table 7.3.1. TTC stream jitter for different setups (oscilloscope measurements). ....                        | 59 |
| Table 7.3.2. Recovered clock jitter for different setups (oscilloscope measurements). ....                   | 59 |
| Table 7.3.3. TTC stream jitter for different setups (PNA measurements). ....                                 | 60 |
| Table 7.3.4. Recovered clock jitter for different setups (PNA measurements). ....                            | 60 |