UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Telecommunications Engineering

Design and Verification of a High-Speed Digitizer System for Beam Loss Detection at CERN.

End of Degree Project

Bachelor's Degree in Telecommunication Technologies and Services Engineering

AUTHOR: Presmanes Cardama, Javier

Tutor: Gadea Gironés, Rafael

External cotutor: CALVO GIRALDO, EVA

ACADEMIC YEAR: 2022/2023

Resumen

Con el objetivo de mejorar las funcionalidades del actual sistema de adquisición de datos, la sección de Beam Loss del CERN decidió implementar un novedoso sistema basado en FPGA. Este desarrollo ofrece una mayor flexibilidad en el firmware y facilita el procesamiento de datos internamente en el dispositivo, además de brindar la capacidad de almacenar estos datos. Esta innovación realza las capacidades existentes y proporciona al CERN un control total sobre el sistema.

La presente tesis se enfocará en el desarrollo y validación del mencionado sistema, buscando satisfacer los requisitos técnicos emitidos por el centro de control del CERN. A lo largo de este documento, detallaremos los diversos módulos desarrollados y explicaremos sus respectivas funciones y responsabilidades de cada uno de ellos. Además, profundizaremos en la verificación funcional del sistema y describiremos las herramientas empleadas para su validación. Finalmente, presentaremos los resultados obtenidos tras aplicar diferentes señales de entrada y realizaremos su correspondiente análisis.

Resum

Amb l'objectiu de millorar les funcionalitats del sistema d'adquisició de dades actual, la secció de Beam Loss del CERN va decidir implementar un nou sistema basat en FPGA. Aquest desenvolupament ofereix més flexibilitat al firmwaer i facilita el processament de dades internament al dispositiu, a més de brindar la capacitat d'emmagatzemar aquestes dades. Aquesta innovació realça les capacitats existents i proporciona al CERN un control total sobre el sistema.

Aquesta tesi s'enfocarà en el desenvolupament i validació del sistema esmentat, buscant satisfer els requisits tècnics emesos pel centre de control del CERN. Al llarg d'aquest document, detallarem els diversos mòduls desenvolupats i explicarem les seves funcions i responsabilitats respectives de cadascun. A més, aprofundirem en la verificació funcional del sistema i descriurem les eines emprades per a la seva validació. Finalment, presentarem els resultats obtinguts després d'aplicar diferents senyals d'entrada i farem la corresponent anàlisi.

Abstract

With the aim of improving the functionalities of the current data acquisition system, the Beam Loss section of CERN decided to implement a novel system based on FPGA. This development offers greater flexibility in the firmware and facilitates internal data processing in the device, in addition to providing the capacity to store these data. This innovation enhances the existing capabilities and gives CERN full control over the system.

The present thesis will focus on the development and validation of the aforementioned system, aiming to meet the technical requirements issued by CERN's control center. Throughout this document, we will detail the various developed modules and explain their respective functions and responsibilities. Furthermore, we will delve into the functional verification of the system and describe the tools used for its validation. Finally, we will present the results obtained after applying different input signals and perform their corresponding analysis.

ACKNOWLEDGEMENTS

First of all, I would like to thank my tutor, Rafael Gadea, for contributing with his knowledge and expertise to the preparation of this work. From the very beginning he made me feel completely free to conduct my research as I wanted, encouraging me to expand my horizons, develop a critic thinking and pursue excellence.

It goes without saying that this would not have been possible if it had not been for Eva, who gave me the opportunity to take part in this amazing project. I sincerely appreciate the countless hours you have dedicated to explain new concepts for me. Eva, your resilience and inspirational nature are truly appreciated. The wisdom I have garnered from your experiences has been enlightening. Your commitment to hard work, and your continuous efforts to foster an environment conducive to improvement, are genuinely commendable.

I would also like to express my appreciation to my colleagues from the BE-BI-BL section who made me feel so welcomed and transformed my tenure at CERN into one of the most enriching experiences of my life. A special mention must be made for Mathieu, who was always available to assist me whenever Eva was unavailable. I thank you for your patience with my queries related to implementation, testing, and timing. Furthermore, I want to acknowledge William for providing me with the opportunity to participate in a Fermilab interview. The faith that someone of your stature placed in me for this position is truly meaningful.

I wish to express my gratitude to Umberto Giovinazzo and Ángel Romero Llop, who instilled in me the values of hard work, humility, and personal growth. Every journey begins with a single step, and I am deeply thankful that you were a part of my inaugural chapter at La Salle during my vocational training. Your lessons will always remain with me.

My heartfelt thanks go to Don Vicente Botella and Mª José Asarta, who recognized and empathized with the challenges I faced, but never lost faith in me. It may have taken me some time to grasp your guidance, but I now carry those lessons with pride, as a shield for the future.

My deepest gratitude to my partner, Elena del Águila, who has steadfastly stood by my side over the past eight years. Through every stage of my journey, you've exemplified the very essence of loyalty. Additionally, I want to express my thanks to my younger sister, Ariadna, who uniquely understands the magnitude of sacrifices I've made, including the countless hours dedicated to Hyperloop. Thank you for being there with me through it all.

My heartfelt thanks also extend to my uncle, Nacho Cardama. It has been a truly remarkable journey to follow in your footsteps, studying at the same university and learning from the same teachers you did, albeit two decades later. You are the finest role model I could ask for, always guiding me to make best decisions, and encouraging me to step out of my comfort zone.

Gracias a mi abuelo Ignacio Cardama por demostrarme que no hace falta tener un título para ser un auténtico ingeniero. Siempre depositaste tu confianza en mí, me enseñaste que todo se consigue con esfuerzo y paciencia, que las cosas mal hechas se hacen dos veces y que siempre que ganas cinco tienes que guardar dos. Te lo debo todo.

Contents

List of Acronyms 9

1 Introduction 1

1.1 European Laboratory of Particle Physics (CERN)

1.1.1 Proton Synchrotron Booster (PSB) 2

1.1.2 Beam Loss Section 3

1.2 Project Overview

1.2.1 Requirements 4

1.2.2 Thesis Objectives 5

1.2.3 Project Set-up 5

1.2.3.1 VME Crate 6

1.2.3.1.1 VFC-HD 6

1.2.3.1.2 Mezzanine Board 7

1.2.3.1.3 CTRV (Central Time Receiver for VME) 8

1.2.3.2 System installation at PSB 9

2 Background 11

2.1 Field Programmable Gate Array (FPGA)

2.2 Beam Loss Diamond-Based Detector System

2.3 High-Speed Data Acquisition

2.3.1 Deterministic Systems 14

2.3.2 Clock Domain Crossing (CDC) 15

2.4 JESD204B Protocol

2.4.1 Protocol Overview 17

2.4.2 Physical Layer (PHY) 18

2.4.2.1 Receiver PMA 19

2.4.2.1.1 Clock Data Recovery (CDR) 19

2.4.2.2 8b/10b Encoding/Decoding 20

2.4.3 Link Layer 23

2.4.3.1 Synchronization 23

2.4.3.1.1 Code Group Synchronization (CGS) 23

2.4.3.1.2 Initial Lane Synchronization (ILAS) 24

2.4.3.1.3 SYSREF Synchronization 25

2.4.4 Transport Layer 26

2.5 ADS54J54 ADC

2.5.1 ADS54J54 JESD204B Capabilities 26

2.5.1.1 Frame Assembly 28

2.5.2 SPI Configuration 29

2.6 Communication Protocols
2.6.1 Wishbone 30
2.6.1.1 Interface 30
2.6.1.2 Data transmission 31
2.6.1.2.1 Handshaking 32
2.6.1.2.2 Single Read Cycle 32
2.6.1.2.3 Single Write Cycle 32
2.6.2 Serial Peripheral Interface (SPI) 33
2.6.2.1 Interface 33
2.6.2.2 Data transmission 34
2.6.3 VMEbus (Versa Module Eurocard bus) 34

3 Development & Results **37**
3.1 VFC
3.1.1 Architecture 37
3.1.1.1 Application 39
3.2 Design
3.2.1 Wishbone Address Decoder 40
3.2.2 Mezzanine Controller 41
3.2.2.1 Voltage Controller 44
3.2.2.2 ADS54J54 Controller 45
3.2.2.3 SPI Controller 46
3.2.2.3.1 ADS54J54 Master WB Configurator 47
3.2.2.3.2 SPI Through Wishbone 49
3.2.2.3.3 SPI Listener 49
3.2.2.4 JESD204B Controller 49
3.2.2.4.1 Altera IP Core 52
3.2.2.4.2 Finite State Machine 54
3.2.2.4.3 Transport Layer 55
3.2.3 RAM Controller 57
3.2.3.1 Channel RAM 59
3.2.3.1.1 Space Computation 59
3.2.3.1.2 Architecture 60
3.2.3.1.3 Addressing 62
3.2.3.2 Finite State Machine 63
3.2.4 Trigger Controller 64
3.3 Verification
3.3.1 Mezzanine Controller 66
3.3.1.1 SPI Controller 66
3.3.1.2 JESD204B Controller 69
3.3.1.2.1 Finite State Machine 69
3.3.1.2.2 Link Layer 69
3.3.1.2.3 Simulation 73
3.3.2 RAM Controller 75
3.3.3 Trigger Controller 77
3.4 Results

4 Conclusions **83**

Bibliography **85**

List of Figures

1.1 CERN Accelerator Complex. The PSB is the light pink ring in a complex chain of particle accelerators [1]. 1
1.2 PSB Ring diagram. 2
1.3 PSB Rings photography. 3
1.4 Lab crate for testing and debugging. 6
1.6 IAM500 mezzanine back view. 7
1.7 CTRV board. 8
1.8 PSB rings with the two stations equipped with detectors 9
1.9 PSB RACK set-up. 10

2.2 Analog front-end. 13
2.3 CDC Metastability and Synchronization process. 15
2.4 JESD204B Basic TX - RX Configuration. 17
2.5 Intel IP Core for JESD204B. 18
2.6 Intel Transceiver PMA Receiver. 19
2.7 Clock Data Recovery block diagram. 20
2.8 3b/4b encoding table. 21
2.9 5b/6b encoding table. 21
2.10 8b/10b encoding process. 22
2.11 8b/10b control symbols. 22
2.12 CGS phase. 24
2.13 ILAS phase. 25
2.14 SYSREF synced with rxlink_clk. 25
2.15 JESD204B Lane Assignment. 27
2.16 JESD204B Block detailed. 27
2.17 ADS54J54 JESD204B Frame Assebly. 28
2.18 ADS54J54 SPI frame. 30
2.19 Simple Master-Slave Wishbone communication interface. 31
2.20 Wishbone Handshake. 32
2.22 Simple Master-Slave SPI communication interface. 34
2.23 Simple full-duplex communication diagram. 34

3.1 VFC Top Level module architecture. 38
3.2 Firmware architecture. 39
3.3 Address decoder module. 40
3.4 Address decoder memory map. 41
3.5 Mezzanine board block diagram. 42
3.6 *MezzanineController* module. 44
3.7 *MezzanineVoltageController* module. 45

3.8 Mezzanine power-up LED. 45
3.9 SPI Controller module signals. 46
3.10 SPI Controller sub-modules details. 47
3.11 JESD204B Controller module signals. 50
3.12 JESD204B Controller sub-modules details. 51
3.13 Altera JESD204B wizard for PHY only instantiation. 52
3.16 JESD204B link FSM module. 54
3.17 JESD204B link FSM. 55
3.18 JESD204B ADC data formatting for lanes. 56
3.19 JESD204B data before transport layer decoding. 57
3.20 JESD204B data sorted by the transport layer. 57
3.21 RAM Controller module. 58
3.22 RAM Controller detailed view. 58
3.23 Channel RAM module. 59
3.24 Channel RAM architecture. 61
3.25 Channel RAM write address. 62
3.26 Channel RAM write select decoder. 62
3.27 Channel RAM read address. 63
3.29 RAM Controller finite state machine. 64
3.30 Trigger controller module. 65
3.31 Trigger controller module in detail. 65
3.32 Python script for writing and checking ADC register content. 67
3.33 Signal Tap configuration for SPI Controller. 68
3.34 Signal Tap output for SPI Controller. 68
3.35 Signal Tap configuration for JESD204B FSM. 69
3.36 Signal Tap output for JESD204B FSM. 69
3.37 Intel’s documentation for JESD204B Link process. 70
3.38 SignalTap configuration for JESD204B Link layer. 71
3.39 SignalTap output for JESD204B Link layer - Phases. 71
3.40 SignalTap output for JESD204B Link layer - Overview. 72
3.41 Python script for modifiying the ADS54J54 operational mode. 72
3.42 ADC triangle pattern bit-by-bit. 73
3.43 ADC Triangle pattern sample value increment. 73
3.44 ADC Chess pattern. 73
3.45 JESD204B Controller testbench. 74
3.46 JESD204B Controller testbench waveform. 75
3.47 JESD204B Controller testbench triangle output. 75
3.48 Big RAM / Samll RAM write enable selector. 75
3.49 RAM Controller testbench FSM waveform. 76
3.50 RAM Controller Signal Tap output for writing. 76
3.51 RAM Controller Signal Tap output for reading. 76
3.52 Trigger controller waveform capturing the configuration. 77
3.53 Trigger controller waveform start delayed output. 77
3.54 Trigger controller waveform FSM. 78
3.55 Sin wave plot (250kHz, 100μs). 79
3.56 Triangular wave plot (250kHz, 135μs). 80
3.57 Detailed triangular wave plot (250kHz, 135μs). 81

Tables

2.1 8b/10b control symbols relevant for JESD204B. 23
2.2 ILAS configuration octets. 25

3.1 Si5394 External PLL pinout configuration. 43
3.2 ADS54J54 commands Table. 48

List of Acronyms

AC Alternating Current.

ADC Analogue-to-Digital Converter.

ASIC Application-Specific Integrated Circuit.

ATLAS A Toroidal LHC ApparatuS.

BI Beam Instrumentation.

BL Beam Loss.

BLM Beam Loss Monitoring.

BST Beam Synchronous Timing.

CDC Clock Domain Crossing.

CDR Clock Data Recovery.

CERN Conseil Européen pour la Recherche Nucléaire.

CGS Code Group Synchronization.

CID Consecutive Identical Digits.

CML Current Mode Logic.

CMOS Complementary Metal-Oxide-Semiconductor.

CMS Compact Muon Solenoid.

CSV Comma Separated Values.

CTIM Central Time Events.

CTR Central Time Receiver.

CTRV Central Time Receiver for VME.

DAC Digital to Analog Converter.

DC Direct Current.

DPC	Digital Processing Card.
DSP	Digital Signal Processor.
EMI	Electromagnetic Interference.
FMC	FPGA Mezzanine Card.
FPGA	Field Programmable Gate Array.
FSM	Finite State Machine.
GMT	General Machine Timing.
HL	High Luminosity.
HPC	High-Pin Count.
IC	Integrated Circuit.
ILAS	Initial Lane Synchronization.
IP	Intellectual Property.
ISOLDE	Isotope Separator On-line Detector.
LED	Light Emitter Diode.
LFSR	Linear Feedback Shift Register.
LHC	Large Hadron Collider.
LINAC	Linear Accelerator.
LIU	LHC Injector Upgrade.
LS	Long Shutdown.
LSB	Less Significant Bit.
LVDS	Low Voltage Differential Signal.
MAC	Media Access Control.
MSB	Most Significant Bit.
NTP	Network Time Protocol.
OASIS	Open Analog Signals Information System.
PCB	Printed Circuit Board.
PCS	Physical Coding Sublayer.

PHY	Physical Layer.
PLL	Phase-Locked Loop.
PMA	Physical Media Attachment.
PSB	Proton Synchrotron Booster.
PTP	Precision Time Protocol.
RAM	Random Access Memory.
RD	Running Disparity.
RF	Radio Frequency.
RTOS	Real Time Operating System.
SFDR	Spurious Free Dynamic Range.
SPI	Serial Peripheral Interface.
SPS	Super Proton Synchrotron.
TAI	International Atomic Time.
TTC	Timing Trigger and Control.
UTC	Coordinated Universal Time.
VME	Versa Module Eurocard.

Chapter 1

Introduction

1.1 European Laboratory of Particle Physics (CERN)

The name CERN is derived from the acronym in French ”Conseil Européen pour la Recherche Nucléaire”. It is an international organization that holds the title of the world’s biggest laboratory for particle physics. Its establishment can be traced back to the aftermath of World War II when European scientific advancements were no longer at the forefront.

The CERN Accelerator Complex consists of a series of accelerators, experiments, and machines that accelerate particles to increasingly higher energies. Each machine boosts the energy of a beam of particles, before injecting the beam into the next machine in the sequence. Presently, the Complex comprises eight accelerators, two decelerators, transfer lines connecting them, and numerous facilities hosting experiments focused on various topics in Particle Physics (e.g., ATLAS, CMS), Nuclear Physics (ISOLDE), Antimatter (ALPHA, ASACUSA), and other fields. Although some of these facilities are above ground, the majority are situated underground. A schematic of the Accelerator Complex is displayed in Figure 1.1.

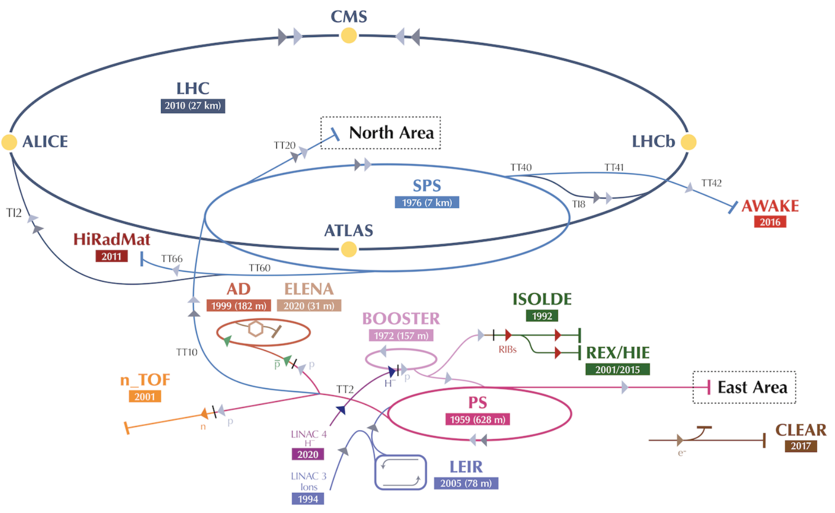


Figure 1.1: CERN Accelerator Complex. The PSB is the light pink ring in a complex chain of particle accelerators [1].

1.1.1 Proton Synchrotron Booster (PSB)

The CERN Proton Synchrotron Booster (PSB) is the first synchrotron in the CERN injector complex and is used for tailoring the wide range of transverse beam characteristics as requested by the various users at CERN, covering intensities from 10^{10} to 10^{13} protons and normalized transverse emittances from $\epsilon_{n,rms} < 0.7\mu m$ (LHC-like beams) to ≈ 9–$10\mu m$ (high intensity beams). As part of the LHC Injectors Upgrade (LIU) project, the PSB was upgraded during the Long Shut Down 2 (LS2) in 2019/2020. A major aspect of this upgrade was the connection of the newly built LINAC4 to the PSB, increasing the injection energy from 50 to $160MeV$ and hence the relativistic factor β, γ^2 by a factor of two. This increased injection energy allows the beam brightness to be doubled, as required for the High Luminosity era of LHC (HL-LHC), while maintaining the tune-spread induced by space charge as pre LS2.

A detailed view of the complex diagram can be seen at Figure 1.2 :

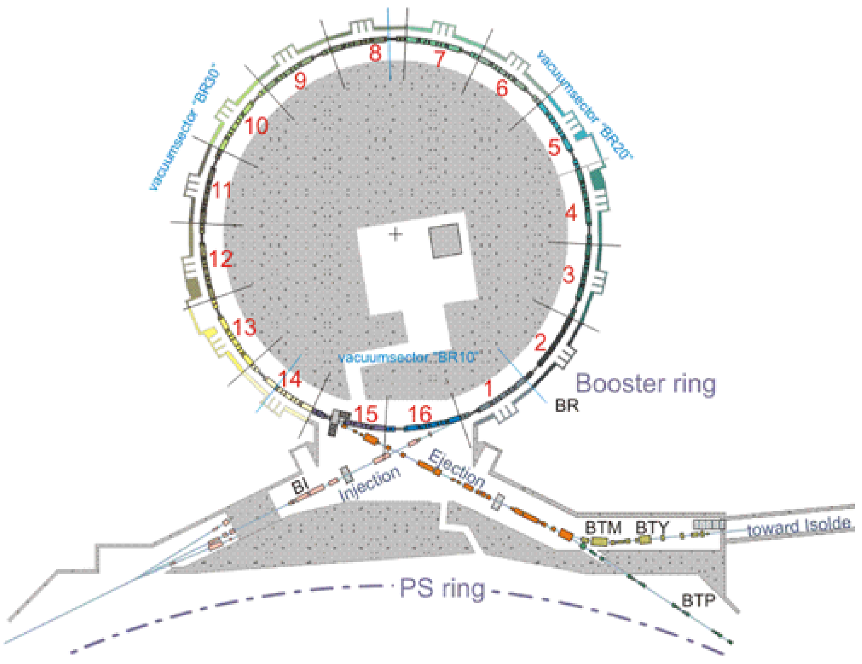


Figure 1.2: PSB Ring diagram.

Here, we can observe that the injection is provided by LINAC4, while the ejection system is split into two distinct paths: the primary path is connected to the Proton Synchrotron accelerator, and the secondary path leads to a different line connected to the ISOLDE experiment.

The PSB is comprised of four rings stacked above each other. LINAC4 is responsible for supplying these rings with bunches of particles. These particles are then synchronized and accelerated to a target energy within the PSB, following which the Proton Synchrotron accelerator is fed with the particles.

The subsequent figure offers a closer view of the accelerator rings from within the complex:

Figure 1.3: PSB Rings photography.

1.1.2 Beam Loss Section

CERN structure is divided into sectors (Director-General, Accelerators and Technology, Finance and Human Resources, International Relations, and Research and Computing). Each of these sectors is split into departments, which are composed by groups and finally by sections.

This thesis has been carried out within the ”Beam Loss” section (BL), which forms a part of the ”Beam Instrumentation” group (BI). The core objective of this section is the design, testing, installation, and maintenance of all systems pertaining to beam loss monitoring across CERN’s complex. This diverse team comprises technicians, engineers, physicists, and students.

Despite consistent monitoring and optimization of accelerator optics, various inevitable phenomena can cause particles to deviate from the intended trajectory and scatter outside the beam pipes. These errant particles are likely to interact with surrounding materials, leading to the production of secondary particle showers and electromagnetic radiation. This results in undesired energy deposition in different machine elements.

The principal role of the Beam Loss Monitoring systems is to control radiation levels in the regions surrounding the beam pipes, estimate the scale of energy depositions, and act promptly to protect the accelerator components. Furthermore, it provides an efficient diagnostic tool for machine operations by detecting local or widespread irregularities, thus empowering operators to optimize machine parameters to minimize them.

At present, the section works with several types of beam loss detectors:

- sep0em
- Ionization chambers
- Secondary emission monitors

- Solid state detectors based on diamond material
- Optical fibres based on Cherenkov detection

1.2 Project Overview

Currently, a system named OASIS (Open Analog Signals Information System) is employed to measure the signals of the BLM diamond detectors installed in the PSB injection region. This system uses Acquiris DC270 cPCI digitizers [2] (also referenced as Keysight U1063A [3]) and switching matrices for the trigger signal routing. These digitizer modules, currently deprecated, provided the following characteristics:

- 4-channels digitization at 1GS/s.
- 250MHz bandwidth.
- 8-bits resolution.
- 2M points memory

The OASIS system provides the control room operators with a graphical interface very similar to a virtual oscilloscope. However, it does not provides them with the capability of performing additional signal processing, such as integrating the losses per revolution turn, per bunch, calculating histograms, etc. The recording of the measured data was also impractical, particularly when the signals were multiplexed.

Due to these limitations, and others related to the homogenization between acquisition systems, the Beam Loss section has decided to develop its own system based on an FPGA, and their standard hardware platform based on VME64x crates. This system would not only function as a fast digitizer board but also offer the flexibility of implementing future additional embedded Digital Signal Processing (DSP) algorithms.

1.2.1 Requirements

To enhance the flexibility and performance of the existing system, we will implement a new system anchored on an Intel FPGA. This upgrade is intended to supplant the current OASIS-based setup. The proposed system will utilize a versatile electronic board, designed by the Beam Instrumentation group, which is intended to serve as a blueprint for various projects. This innovative approach will increase overall efficiency and adaptability, ensuring that our system can effectively cater to diverse requirements and evolving scenarios.

The requirements of the new data acquisition system are listed in the specifications [4] file. These requirements can be summarized into the following task:

- Capture the injection process of all rings of the PSB.
- Provide per turn losses.

- Start-Stop capabilities based on triggering signals.
- Data integration.
- Data saving for later processing.

The PSB beam injection timings events depend on various factors, such as the current machine configuration and status and the users requirements. Each injection into the accelerator inevitably results in some losses. The newly developed system will be capable of measuring up to $160\mu s$/channel of beam losses following a new injection cycle. Furthermore, it will provide the ”per turn losses” value, which will be stored in internal RAM memory. This value represents the discrete integral of the losses in a $1\mu s$ integral window.

Prior to each injection cycle, the CTRV board (Paragraph 1.2.3.1.3) generates timing signals for initiating and concluding data recording. These signals must be processed by the FPGA, which should be capable of incorporating an additional tunning delay if necessary.

After the data from the four ADC channels has been saved, the CPU software can read back all the data from the RAM memory through the VME bus, and publish it or log it in the appropriate control service. This ensures that the entire data set is readily available for further analysis and diagnostics, thereby improving the overall effectiveness and safety of the beam operation.

1.2.2 Thesis Objectives

This thesis will comprehensively summarize the critical steps involved in the development, testing, and verification of the aforementioned system.

The thesis is structured in two main sections. Firstly, there will be a ’Background’ chapter (Chapter 2) that will elucidate key concepts essential to comprehending the system’s implementation. This section will offer a concise explanation of the workings of High-Speed Data Acquisition Systems (Section 2.3), the JESD204B protocol (Section 2.4), and various communication protocols utilized for internal and external interaction with or within the system (Section 2.6).

Secondly, a ’Develop & Results’ chapter (Chapter 3) will concentrate on the design (Section 3.2) and verification (Section 3.3) of different system verilog modules engineered to carry out various tasks needed to meet the specifications.

Chapter 2 is grounded on a collection of papers, books, and online articles that have been consulted to design, implement, and verify the entire system from the ground up. Conversely, Chapter 3 is predicated on the schematics created to facilitate a better understanding of the modules and their collaborative functions.

1.2.3 Project Set-up

In this subsection, we will introduce the various boards that constitute the system. These boards, discussed below, have been designed and tested by various CERN teams. They serve as crucial components for seamlessly integrating our new system into the existing CERN ecosystem, which encompasses a wide range of technologies and equipment.

1.2.3.1 VME Crate

First part of the set-up is the VME Crate, a standardized modular enclosure used in computer systems. VME stands for Versa Module Eurocard and refers to an industry-standard bus system used in the field of embedded computing and industrial automation. It is specifically designed to fit into a 19” industrial rack. The one used on this work has a height of 3U. [1]

Figure 1.4 shows the current crate installed at the laboratory where all the testing and debugging of the project is done:

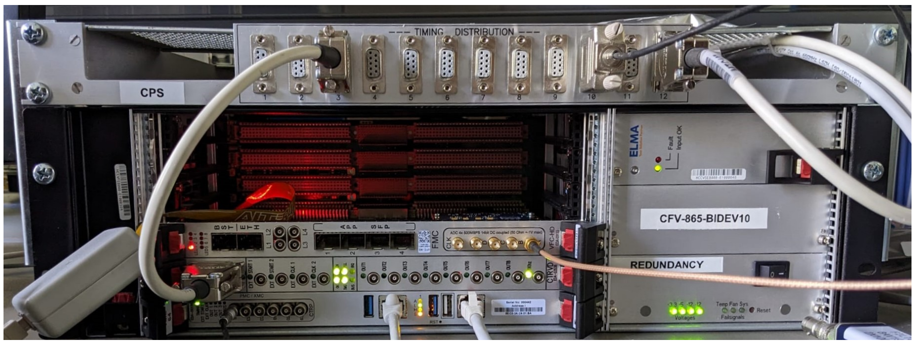


Figure 1.4: Lab crate for testing and debugging.

Within this crate, we have installed a single VFC board alongside a CTRV board (Paragraph 1.2.3.1.1 and Paragraph 1.2.3.1.3). The VME crate also houses a master CPU, consisting on a multi-core single-board computer based on Intel’s Xeon processor. This card, referred as MEN A25, is responsible for communicating with the rest of the boards via the VME bus interface.

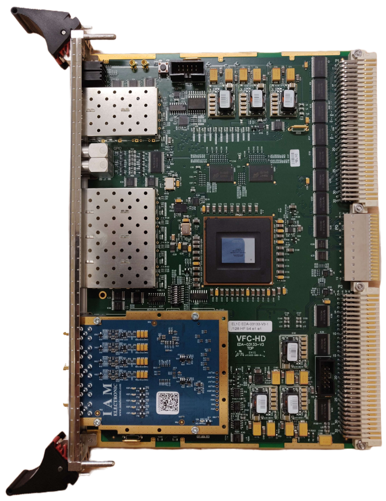
1.2.3.1.1 VFC-HD The Beam Instrumentation group at CERN has developed a common Digital Processing Card (DPC) named VFC, which stands for VME FMC Carrier, (Figure 1.5a). This DPC incorporates an Intel Arria V Field Programmable Gate Array (FPGA) and a variety of other inputs/outputs, which are frequently utilized for Ethernet communication, time management, GPIO, and more.

The motivation behind creating this unified board system from the diverse sections within the BI group, which cater to an extensive range of requirements for the machines. In an effort to enhance the efficiency of development, testing, and debugging for the various systems created within the group, the VFC board was designed.

The FMC (FPGA Mezzanine Card) [5] provides a convenient way to expand the functionality of an DPC by allowing the addition of different mezzanine boards. In this project, the IAM500 Mezzanine board is installed on the VFC board, as shown in Figure 1.5b.

[1]Rack slots follow a standard size measured in ”U,” with ”3U” denoting 3 units of rack space. A single rack unit is equivalent to 44.45mm.

(a) VFC DPC top view.


(b) VFC with a IAM500 mezzanine installed.

The firmware architecture is divided into two main parts (Section 3.1): the System and the Application (Subsubsection 3.1.1.1). The System encapsulates common features for all projects using the VFC, such as clock generators, voltage monitoring, and interruption management. The Application layer, on the other hand, is where users write their own code, typically dependent on the installed mezzanine. This layer is the most relevant for this thesis, as it is where all the logic has been implemented.

1.2.3.1.2 Mezzanine Board Figure 1.6 displays the IAM500 [6] mezzanine board selected for this project. CERN commissioned the design and manufacturing of this mezzanine to IAM Electronic company in 2019 [7]. The mezzanine features a High-Pin Count (HPC) connector with a standard pin-out to interface with the FPGA [5].

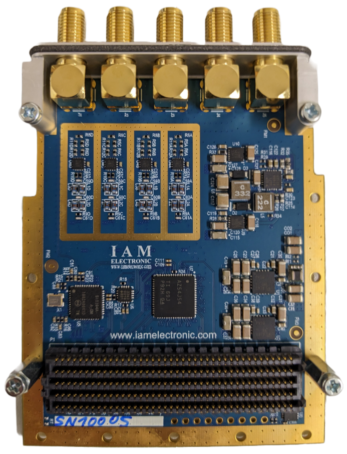


Figure 1.6: IAM500 mezzanine back view.

The mezzanine board features a Texas Instruments Analog-to-Digital Converter (ADC) model ADS54J54 [8] and an external Phase-Locked Loop (PLL) [2] model Si5395 [9] from Skyworks.

The Si5395 PLL's primary function is to generate stable frequencies for both the ADC and FPGA, providing signals of 500 MHz and 1.046 MHz. The 500 MHz signal serves as the standard sampling frequency for this application (as the ADC is not configured for down-sampling) and as a reference frequency for the JESD204B protocol. Meanwhile, the 1.046 MHz frequency is utilized by the JESD204B protocol for synchronization purposes.

The ADS54J54 ADC is a four-channel ADC with 14-bit resolution, capable of sampling at 500 MSPS for each channel. To manage the transmission of this large amount of data (28Gb/s), it implements the JESD204B protocol (Explained in Section 2.4), a high-speed data transfer protocol designed by JEDEC (Joint Electron Device Engineering Council) for transferring ADCs and DACs data to or from FPGAs.

1.2.3.1.3 CTRV (Central Time Receiver for VME) CERN relies on precise timing and synchronization for the success of its large-scale projects. To achieve this, deterministic latency (Subsection 2.3.1) timing signals are supplied from a centralized location to all experiments, allowing boards throughout CERN to synchronize their tasks.

The CTRV module, a VME format component, receives events through the General Machine Timing (GMT) link. This differential RS-485 network, driven by Timing Master (CTG) modules, is distributed across the accelerator complex. By using software libraries, users can command the modules to perform actions upon receiving an event, such as generating front panel pulses or bus interrupts.

Figure 1.7 illustrates a CTRV board, showcasing various connectors on its front panel for generating different signals. This setup plays a critical role in ensuring seamless synchronization and efficient operation at CERN.

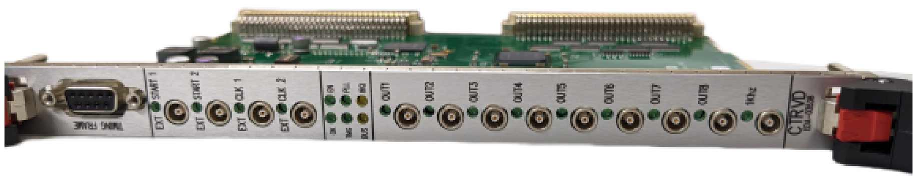


Figure 1.7: CTRV board.

[2]Phase-Locked Loop is a control system that synchronizes an output signal's frequency and phase with that of an input reference signal. It consists of a phase detector, a low-pass filter, and a voltage-controlled oscillator. PLLs are widely used in communication systems, frequency synthesis, and clock recovery applications

1.2.3.2 System installation at PSB

The PSB accelerator comprises four distinct rings. Numbered from bottom to top, we have Rings R1 through R4. All these rings are independently fed by the LINAC4.

To measure the system's losses, two different columns have been installed where the systems will be housed. These columns, named E1 and E2, are positioned just after and just before the beam injection system, as depicted in Figure 1.8:

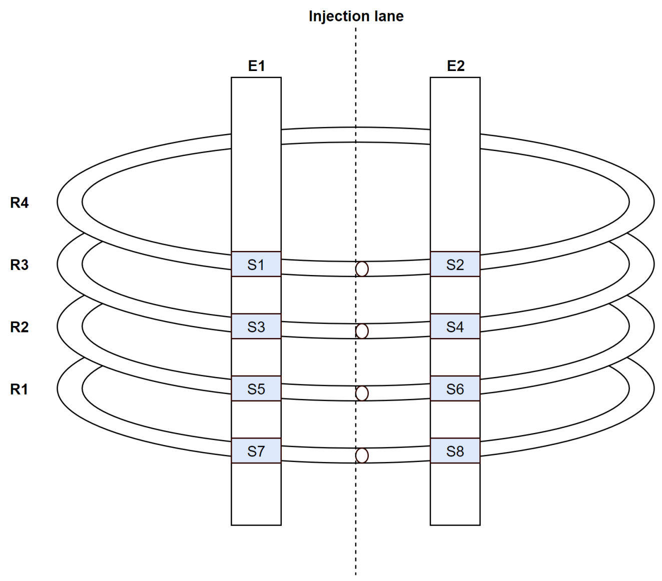


Figure 1.8: PSB rings with the two stations equipped with detectors .

Each of these columns will house a sensor set-up for each ring. This sensor set-up comprises the diamond sensor (Section 2.2) and an analog front end (Figure 2.2), which is responsible for splitting the sensor data into two paths and amplifying one of them in order to enhance the system's dynamic range.

To read these sensors, a rack with two independent VME crates (Subsection 1.2.3.1) will be installed. Inside these crates, two VFC boards and two CTRV boards will be set up.

Since each sensor set-up has two different outputs (amplified and non-amplified), and there are two sensor set-ups per ring, we will need one Mezzanine (and therefore a VFC) for each ring.

Furthermore, because each VFC has two input signals (start and stop) from a timing board, there will be an equal amount of CTRVs as VFCs.

A more illustrative depiction of the rack set-up can be found in Figure 1.9:

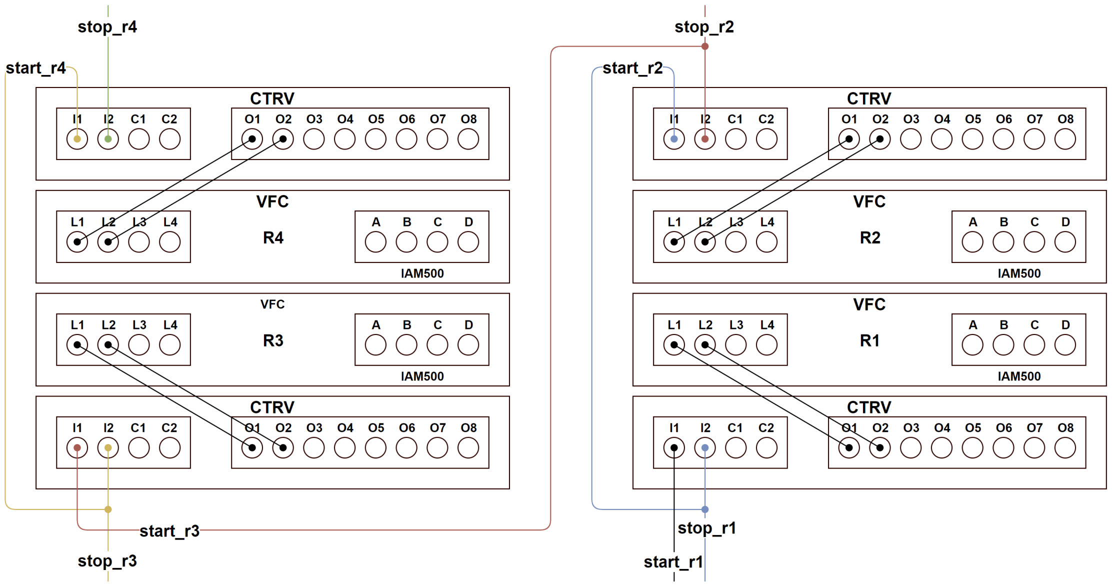


Figure 1.9: PSB RACK set-up.

The mezzanine channels (from A to D) are intended to be connected to the sensors, and LEMO connectors 1 and 2 of the VFC (L1 and L2) will be linked to the CTRV outputs.

Conversely, five signals are received from the central timing crate (start_r1, stop_r1, stop_r2, stop_r3, stop_r4) and are directly connected to the GMT. These signals are directly fed into the LEMO connector as depicted in the previous figure. To derive the subsequent start signals (start_r2, start_r3, etc.), the stop signals from the preceding CTRV are utilized. Therefore, stop_r1 serves as start_r2, stop_r2 becomes start_r3, and so forth.

Chapter 2

Background

As previously mentioned, the goal of this chapter is to introduce various technologies, protocols, and concepts that are crucial for understanding the system's implementation. This chapter will be based on concepts explained in various articles or books that have been consulted during this study.

The first part of this chapter will be dedicated to discussing what a Field-Programmable Gate Array (FPGA) is (Section 2.1) and why it's of paramount importance in certain projects where parallel processing is a requirement.

Secondly, we will delve into what a high-speed data acquisition system is (Section 2.3), along with some crucial concepts such as Determinism, Clock Domain Crossing, and Synchronization.

Subsequently, we will elucidate how the JESD204B protocol operates (Section 2.4), given it forms the backbone of the system. We will explain the various layers that constitute it and their respective responsibilities.

Finally, there will be a brief overview of the Analog-to-Digital Converter (ADC) used for this project (Section 2.5), and the communication protocols (Section 2.6) for both internal and external communication among modules or the VME Crate CPU.

2.1 Field Programmable Gate Array (FPGA)

A FPGA is a semiconductor device that can be programmed or reprogrammed to perform complex computational tasks after manufacturing. Unlike a traditional microprocessor that carries out instructions in a sequential manner, FPGAs use a large array of digital logic gates (hence the name ”gate array”) that can be configured to work together to perform complex operations in parallel.

One key aspect of FPGAs is their reprogrammability. The functionality of the FPGA is defined by the configuration data loaded into it, usually stored in an external memory device. This means that the functionality of an FPGA can be updated or changed after it has been deployed in the field, hence the term ”field-programmable”.

FPGAs offer digital designers the ability to create hardware-based devices capable of performing multiple tasks concurrently. This parallelism is one of the most powerful features and advantages of using FPGAs over other solutions such as microcontrollers.

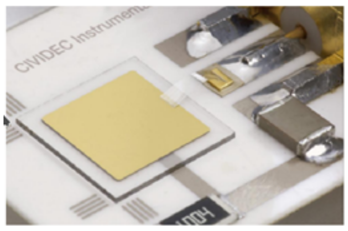
Additionally, because they are hardware-based, FPGAs can achieve extremely high speeds for computations, making them suitable for various tasks such as hardware acceleration, cryptography, or parallel data processing.

However, debugging and testing code on an FPGA can be more challenging than implementing it in software. This complexity arises because FPGA modules must be simulated by software capable of interpreting the code and generating the necessary signals to verify the system’s overall behaviour. This simulation must be meticulous to account for all the system’s possible states, adding to the complexity of this method. Moreover, some aspects, such as meta-stability, cannot be easily simulated.

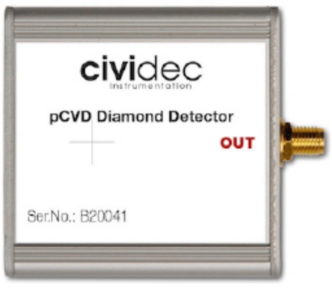
An alternative method of verifying system functionality, which doesn’t require functional verification, involves lab testing or using tools like Intel’s SignalTap to ensure all signals are working as expected. This method can be slow, and in some cases, it may not be possible to test all scenarios of your system.

2.2 Beam Loss Diamond-Based Detector System

In order to be able to measure the losses of the accelerator, there will be installed a front-end hardware system based on a CIVIDEC sensor [10]. This sensor consists of a 10 mm × 10 mm × 0.5 mm polycrystalline Chemical Vapor Deposition (pCVD) diamond substrate coated on each side with a 200 nm thick gold electrode with a size of 8 mm × 8 mm, as can be seen in Figure 2.1a. The detector capacitance is typically 8 pF. The bottom electrode is glued to a ceramic PCB and the top electrode is bonded with 10 um thick aluminium bonding wires. The detector is mounted in an aluminium box with extra RF shielding shown in Figure 2.1b. The detectors are operated with a bias voltage of 500 V, which corresponds to an electric field strength of 1 V/um.



(a) Photo of diamond sensor.



(b) Detector in its enclosure.

This sensor acts as a capacitor. It is connected to a 500V DC voltage generating an electric field in between its electrodes. When a particle (representing losses) traverses the sensor, it generates an electron-hole pair, culminating in a current flow. This current is subsequently converted into a voltage that can be measured by the ADC.

The detectors are connected to an AC/DC splitter as shown in Figure 2.2. The AC path is connected to a 2 GHz broadband current amplifier [11]. This has a lower cut-off frequency of 1 MHz and an effective upper -3 dB cut-off frequency of 2 GHz.

The detector signal is split into two paths with distinct attenuation and amplification configurations. As a result, information from one sensor is available with both high and low attenuation. This approach extends the dynamic range of the entire system, as there may be situations where the non-attenuated path is saturated, while the attenuated path remains unaffected.

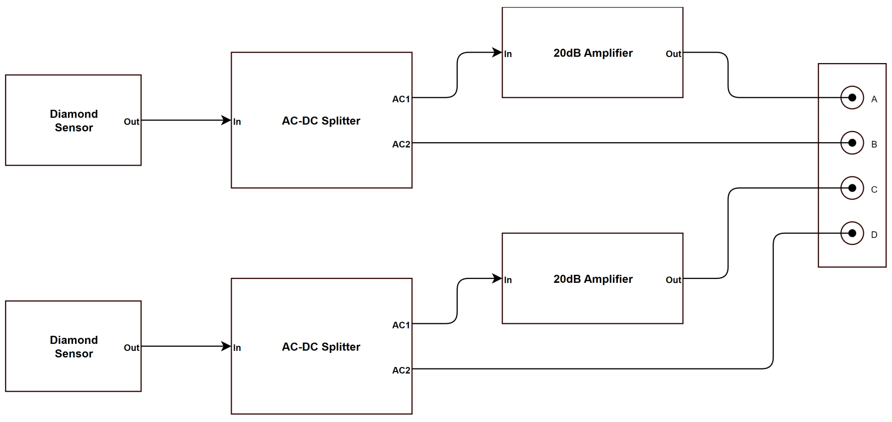


Figure 2.2: Analog front-end.

2.3 High-Speed Data Acquisition

High-speed data acquisition is a critical aspect of many scientific and engineering applications. From medical imaging to aerospace testing, the ability to capture vast amounts of data quickly and accurately is essential for gaining insights into complex systems and processes.

High-speed data acquisition systems represent the convergence of various technological developments, such as signal distribution (fibre optic, twisted pair, coaxial), hardware drivers (CMOS, LVDS, CML), and protocol advancements. These technologies enable the creation of more robust and accurate systems today.

In this section, our primary focus will be on understanding deterministic systems and clock domain crossing, and why these concepts are critically significant for synchronization purposes. This foundational knowledge will play an essential role in later discussions on the JESD204B protocol (Section 2.4).

2.3.1 Deterministic Systems

By definition, deterministic systems are the opposite of random events. A deterministic system will consistently behave in the same manner for identical input events. When applied to electronics and signaling, a deterministic system guarantees a specific latency [1] and jitter [2] for a communication process.

Nowadays, modern software and firmware implement various methods to achieve deterministic latency and jitter. For example, computer systems utilize protocols such as Precision Time Protocol (PTP) or Network Time Protocol (NTP) to accomplish this task. On the other hand, when dealing with micro-controllers, it is common to use a Real-Time Operating System (RTOS), like FreeRTOS [12], which is highly popular due to its open-source nature[3].

At CERN, deterministic behaviour is achieved through the use of various timing networks such as General Machine Timing (GMT), Beam Synchronous Timing (BST), and Timing Trigger and Control (TTC). The TTC is mostly used by the LHC experiments, while the BST is only available in the LHC and SPS. The rest of the accelerators rely on the GMT, which is specifically utilized by the PSB diamond system, and will be further elaborated on in this description.

GMT is UTC-synchronous, with a reference clock oscillating at a frequency of 40 MHz. Distributed across the accelerator complex via fibre optic and twisted pair (RS-422) cables, GMT operates at a bandwidth of 500 kbit/s and ensures a granularity of 1 ms with jitter lower than 1 ns [13]. It transmits eight 32-bit frames per millisecond, where the first slot is used for the Millisecond (MS) event, leaving 7 places for all other messages to be sent within a single millisecond, which can be central timing events (CTIM), telegrams, the UTC time or the GMT cable ID. The MS event encodes the millisecond number since the beginning of the cycle. The telegrams describe current beam parameters. And the central timing events describe key moments of the accelerator cycles such as ”start of cycle”, ”Beam injection”, ”Beam extraction”, etc.

On the reception side, CTIM events are decoded by Central Timing Receiver (CTR) modules (Paragraph 1.2.3.1.3) and used to either directly trigger software actions, or (more often) to generate derived local timing events to synchronise additional equipment (Section 1.2).

UTC, or Coordinated Universal Time, is a time standard that is used globally as a reference for timekeeping. It is based on the time standard known as International Atomic Time (TAI), which is generated by a network of atomic clocks maintained by national laboratories around the world.

UTC is kept in sync with TAI through the use of leap seconds, which are added or subtracted from

[1]Latency is the time taken for a signal to travel from the sender to the receiver.

[2]Jitter refers to the variation in time or latency of signal events, usually measured in reference to a clock source

[3]Open source refers to a type of software whose source code is made publicly available, allowing anyone to view, use, modify, and distribute it. This encourages collaboration and fosters innovation by enabling a community-driven approach to software development.

the UTC time scale as needed to keep it within $0.9s$ of TAI. Leap seconds are added to UTC on either June 30th or December 31st of each year, or on occasion, on other dates as needed.

2.3.2 Clock Domain Crossing (CDC)

In the design of FPGA-based digital systems, clock domain crossing (CDC) is a key concept. This refers to the process where signals pass from one clock domain to another. A clock domain can be defined as a part of a system where all elements are driven by the same clock signal.

When designing digital systems, sometimes it's necessary to have different parts of the system operating at different clock frequencies, or even different clock phases. These different parts are said to be in different clock domains. For example, as we will see later on in Figure 3.11, there is an application clock, two different high-speed clocks from an external PLL needed for the JESD204B PHY layers, a low-speed SYSREF for protocol synchronization, etc...

The process of transferring data across these clock domains is non-trivial and can potentially cause data integrity issues if not managed correctly. Some issues that can occur include metastability, data loss, and data coherency problems.

Metastability occurs when a signal is sampled too close to its transition, causing an ambiguous state that can propagate through the design and lead to unpredictable results as shown in Figure 2.3:

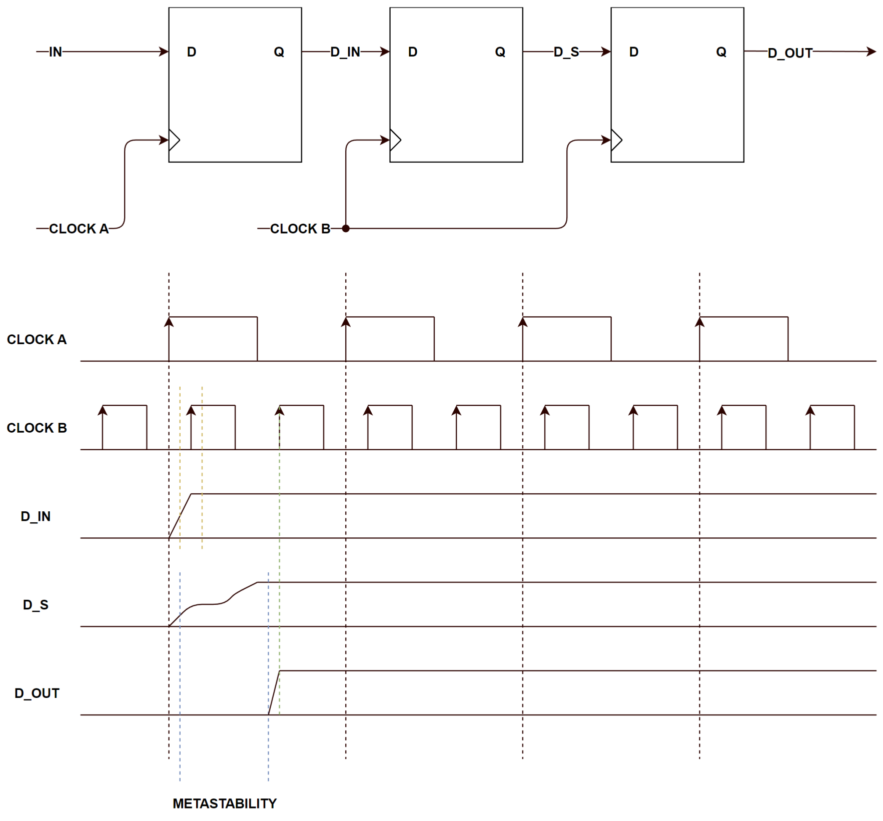


Figure 2.3: CDC Metastability and Synchronization process.

The image illustrates a system where an input signal exists within clock domain A. Conversely, we have a receiver system within clock domain B. If domain B samples the signal of domain A during its transition state (orange lane), the output D_S may not be guaranteed to be stable, leading to what is referred to as a ”metastable” state (blue lane). To circumvent this, a synchronization technique must be applied.

Throughout this thesis, the double register synchronization method is predominantly employed. As depicted in the preceding figure, this method involves using two cascading flip flops to synchronize the signal. This technique is commonly utilized for single signals. However, when there is a need to synchronize a bus, memory, or similar entities, different techniques should be employed.

Data loss can occur if data is not synchronized correctly across clock domains. For example, if data is transferred from a faster clock domain to a slower one without the proper synchronization mechanisms, some data could be lost.

2.4 JESD204B Protocol

The JESD204 protocol is an intricate mechanism crafted for high-speed communication, typically engaged between an Application-Specific Integrated Circuit (ASIC) [4] or FPGA and one or more ADCs or DACs. The complexity of this protocol is rooted in its versatility, as it is designed to accommodate varying factors such as the number of transmission lanes, devices, and converters per device. This versatility makes the JESD204B protocol highly adaptable across a wide range of projects, albeit it also amplifies its complexity. The protocol also plays a significant role in clock distribution as it incorporates a Clock Data Recovery (CDR) [5] system for deriving the clock from the data itself, employs 8b/10b data encoding and other strategies to optimize the efficiency of this technique.

Figure 2.4 presents a basic system based on JESD204B, featuring a generic TX and RX block where various JESD204B subclass 1 components can be identified. In the following sections of this thesis, we will focus on the different layers of the RX block, its behavior, and other aspects of the protocol, such as synchronization.

[4] An Application-Specific Integrated Circuit (ASIC) is a type of electronic chip custom-designed for a specific purpose or application. In contrast to general-purpose chips, ASICs are tailored to carry out a specific function, leading to enhanced performance, power efficiency, and size reduction for the targeted application.

[5] Clock Data Recovery (CDR) is a technique used in digital communication systems to extract and synchronize the clock signal from a data stream. This method enables accurate timing and reliable data recovery, particularly in high-speed applications where a separate clock signal may not be available or practical. CDR is essential for preserving data integrity and maintaining synchronization.

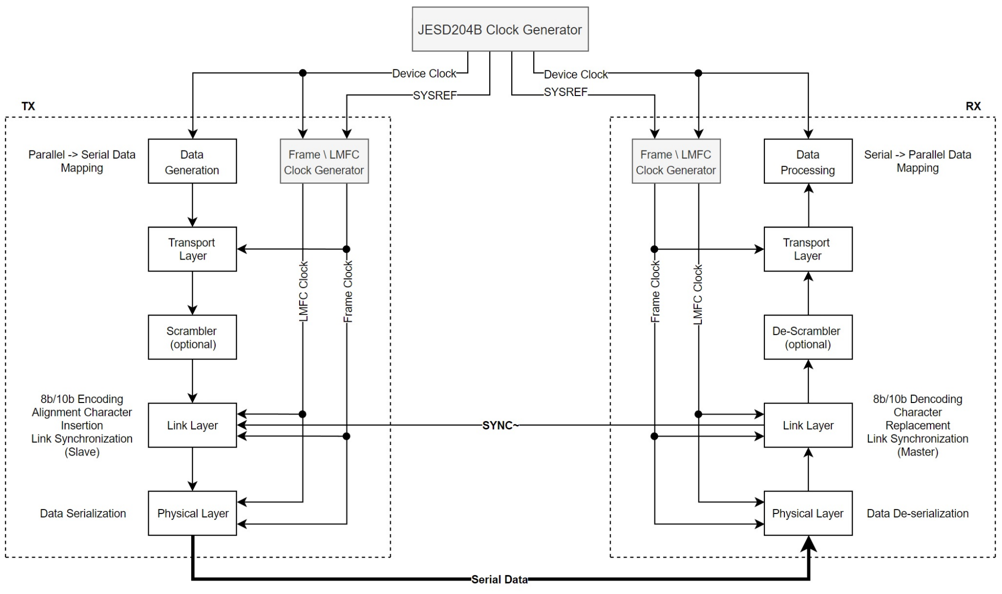


Figure 2.4: JESD204B Basic TX - RX Configuration.

2.4.1 Protocol Overview

The JESD204B standard offers a method to interface one or multiple data converters with a digital-signal processing device (typically an ADC or DAC connected to an FPGA) through a high-speed serial interface, as opposed to more traditional parallel data transfers. Capable of running at up to 12.5 Gbps/lane, this interface uses a framed serial data link with embedded clock and alignment characters. The interface simplifies the implementation of high-speed converters by reducing the number of traces between devices, minimizing trace-matching requirements, and eliminating setup- [6] and hold-timing [7] constraint issues. However, establishing a link prior to data transfer introduces new challenges and techniques for ensuring proper interface functionality.

The JESD204B interface undergoes three phases to establish a synchronized link: Code Group Synchronization (CGS), Initial Lane Synchronization (ILAS), and data transmission phase. The link requires a shared reference clock (device clock), at least one differential CML physical data electrical connection (called a lane), and at least one other synchronization signal (SYNC and possibly SYSREF).

The following list indicates what are the parameters that can be modified for each JESD204B application in order to fit the architecture of your system:

- **M**: number of converters.
- **L**: number of physical lanes.
- **F**: number of octets per frame.

[6]Setup time is the minimum interval before the clock edge during which the input data must be stable and valid.
[7]Hold time is the minimum interval after the clock edge during which the input data must remain stable and valid.

- **S**: number of samples per frame.
- **K**: number of frames per multiframe.
- **N and N'**: converter resolution and number of bits used per sample, respectively. N' value is N value, plus control and padding bits. The protocol specifies that N' should be a multiple of 4.

In this project, we are working with four converters (M=4) distributed across eight lanes (L=8). Additionally, our converters have a resolution of fourteen bits (N=14), so it is necessary to add control and tail bits until we reach sixteen bits (N'=16). For the frame format, we have established one octet per frame (F=1) and only one sample per frame (S=1), with a default number of thirty-two frames per multi-frame (K=32).

2.4.2 Physical Layer (PHY)

As in other protocols and based on the OSI Model [14], the physical layer defines the relationship between a device and a transmission medium. This layer includes the electrical and mechanical characteristics such as pin layouts, voltage levels, etc.

For this project, the IP core used for the JESD204B protocol is from Intel. The Physical Layer and Link Layer are somewhat different due to Intel's implementation. In this case, the Physical Layer is responsible for 8b/10b decoding and word alignment, while the descrambling is part of the Link Layer.

The Physical Layer is composed of the Physical Coding Sub-layer (PCS) and the Physical Media Attachment (PMA) [15].

Figure 2.5 illustrates Intel's IP core and how the Physical Layer and Link Layer are structured, along with their responsibilities:

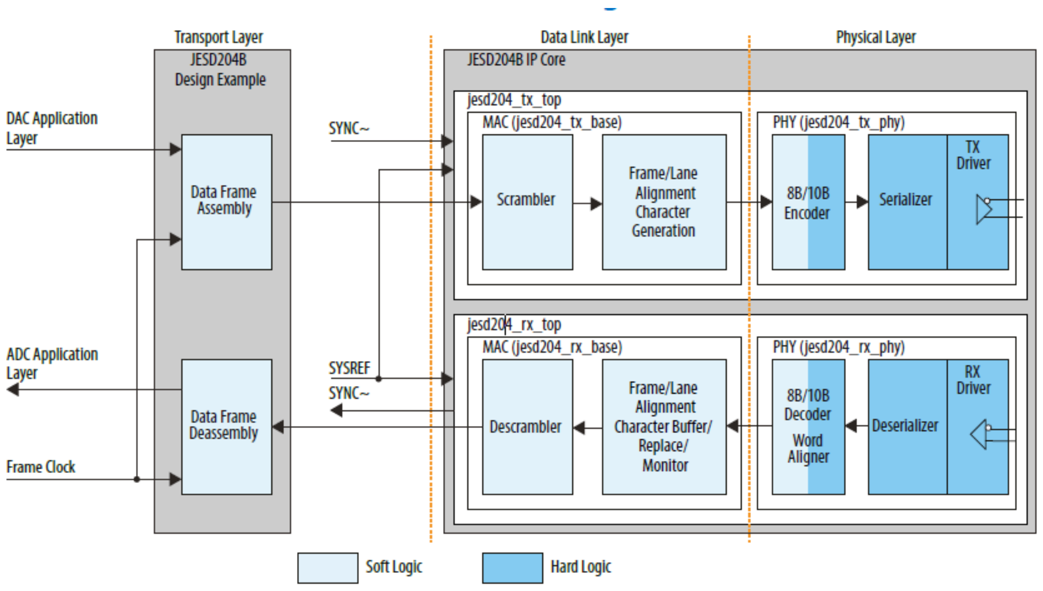


Figure 2.5: Intel IP Core for JESD204B.

The RX Driver is the hard-logic [8] component that adapts the lane's physical medium. It converts the data from differential to logical values.

On the other hand, to implement the 8b/10b encoding, a soft-logic [9] component is also required, which provides greater flexibility. This is crucial, as the JESD204B protocol is highly configurable, catering to various use cases.

2.4.2.1 Receiver PMA

The receiver recovers the clock information from the received data, deserializes the high-speed serial data and creates a parallel data stream for either the receiver PCS or the FPGA fabric. The receiver portion of the PMA is comprised of the receiver buffer, the clock data recovery (CDR) unit, and the deserializer as shown in Figure 2.6.

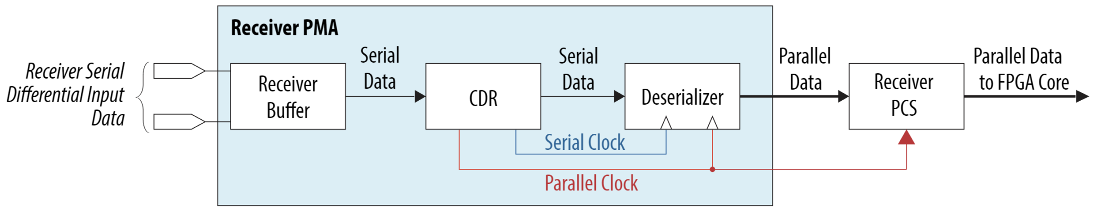


Figure 2.6: Intel Transceiver PMA Receiver.

The receiver input buffer receives serial data from input pin and feeds the serial data to the clock data recovery (CDR) unit and deserializer.

On the other hand, CDR is in charge of extracting the clock information from the data, which is one of the key features for being able to have high-speed systems nowadays.

Finally, the deserializer block clocks in serial input data from the receiver buffer using the high-speed serial recovered clock and deserializes the data using the low-speed parallel recovered clock.

2.4.2.1.1 Clock Data Recovery (CDR) The CDR block locks onto the received serial data stream and extracts the embedded clock information in the serial data. There are two operating modes:

- **CDR mode**: The CDR initially locks onto the reference clock, causing it to operate near the received data rate. After locking to the reference clock, the CDR transitions to lock-to-data mode where it adjusts the clock phase and frequency based on incoming data.
- **CMU mode**: The CDR locks onto the reference clock and acts as a TX PLL, generating the clock source for the TX. The CDR cannot capture any recovered data in this mode and can only drive x1 clock lines.

[8]Hard-logic refers to the actual hardware installed on the FPGA, not created by the FPGA at synthesis time. Hard-logic is designed around the FPGA to be highly efficient.

[9]Soft-logic refers to the programmable logic that can be configured and customized within an FPGA, allowing for greater flexibility and adaptability to various applications.

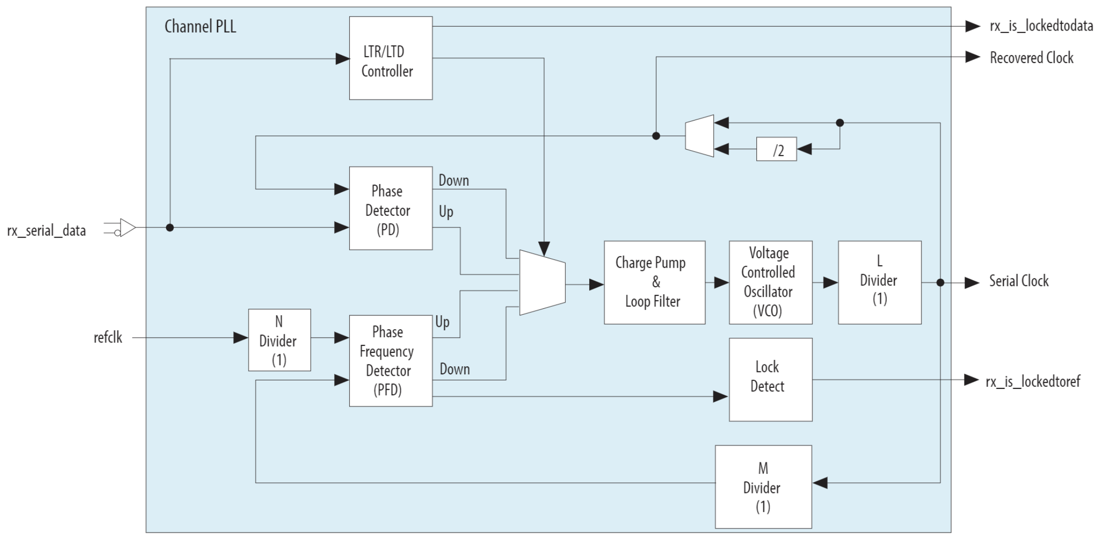


Figure 2.7: Clock Data Recovery block diagram.

The rx_serial_data is introduced to the module along with a refclk, which must have the same nominal frequency as the clock that we want to extract from the data. In this project, it will be 500 MHz, as we will see in the Methodology chapter.

While the system is not locked to the data, it will be locked to the ref, and vice versa. This is accomplished by a multiplexer that selects the clock source. Once the system is synced to the data, the rx_is_lockedtodata bit will be set, and the recovered clock will be available for the user.

When the system is locked, it can maintain the clock with a tolerance that depends on the FPGA specifications. This tolerance is also important, as the reference clock must meet this tolerance when feeding the system.

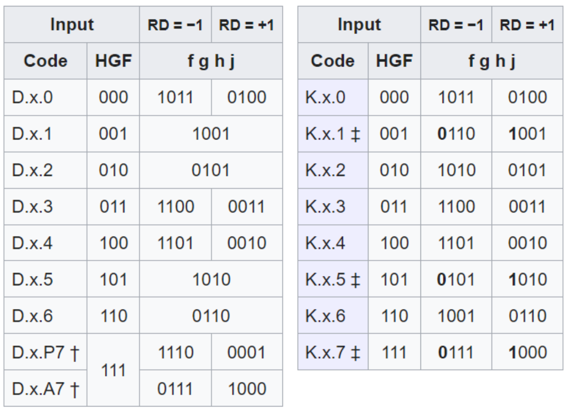
2.4.2.2 8b/10b Encoding/Decoding

The 8b/10b encoding system [16] is a widely used technique to improve data transmission reliability and maintain signal integrity. It works by transforming an 8-bit data word into a 10-bit encoded data word before transmission. The primary goals of the 8b/10b encoding system are:

- **DC balance**: By ensuring an equal distribution of 1's and 0's in the transmitted data, the system minimizes the DC offset, which helps maintain signal integrity and reduces the likelihood of signal errors.
- **Control characters and synchronization**: The 8b/10b encoding system provides a set of special control characters (known as K characters), which are used for synchronization, alignment, and delimiting frames. These control characters have specific properties that make them easily distinguishable from regular data.

To encode the data, the first step is to split the 8-bit block into two blocks. The first block consists of the three most significant bits (MSB of the byte, referred to as "HGF." The second block is formed by the five least significant bits (LSB) called "EDCBA."

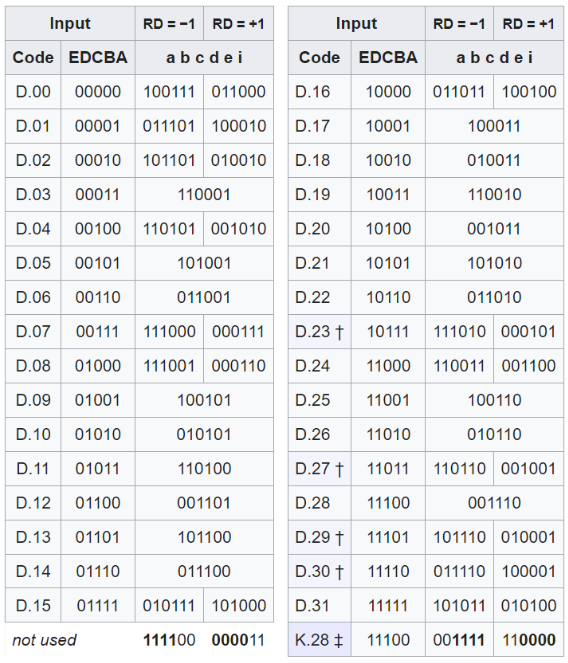
Once the blocks are created, two different conversions are needed. The first conversion is a 3b/4b data conversion using the table shown in Figure 2.8, and the second conversion is a 5b/6b data conversion using the table in Figure 2.9.



Input		RD = −1	RD = +1
Code	HGF	f g h j	
D.x.0	000	1011	0100
D.x.1	001	1001	
D.x.2	010	0101	
D.x.3	011	1100	0011
D.x.4	100	1101	0010
D.x.5	101	1010	
D.x.6	110	0110	
D.x.P7 †	111	1110	0001
D.x.A7 †		0111	1000

Input		RD = −1	RD = +1
Code	HGF	f g h j	
K.x.0	000	1011	0100
K.x.1 ‡	001	0110	1001
K.x.2	010	1010	0101
K.x.3	011	1100	0011
K.x.4	100	1101	0010
K.x.5 ‡	101	0101	1010
K.x.6	110	1001	0110
K.x.7 ‡	111	0111	1000

Figure 2.8: 3b/4b encoding table.



Input		RD = −1	RD = +1
Code	EDCBA	a b c d e i	
D.00	00000	100111	011000
D.01	00001	011101	100010
D.02	00010	101101	010010
D.03	00011	110001	
D.04	00100	110101	001010
D.05	00101	101001	
D.06	00110	011001	
D.07	00111	111000	000111
D.08	01000	111001	000110
D.09	01001	100101	
D.10	01010	010101	
D.11	01011	110100	
D.12	01100	001101	
D.13	01101	101100	
D.14	01110	011100	
D.15	01111	010111	101000
not used		111100	000011

Input		RD = −1	RD = +1
Code	EDCBA	a b c d e i	
D.16	10000	011011	100100
D.17	10001	100011	
D.18	10010	010011	
D.19	10011	110010	
D.20	10100	001011	
D.21	10101	101010	
D.22	10110	011010	
D.23 †	10111	111010	000101
D.24	11000	110011	001100
D.25	11001	100110	
D.26	11010	010110	
D.27 †	11011	110110	001001
D.28	11100	001110	
D.29 †	11101	101110	010001
D.30 †	11110	011110	100001
D.31	11111	101011	010100
K.28 ‡	11100	001111	110000

Figure 2.9: 5b/6b encoding table.

In these tables, we can see that depending on the current value of the 3b or 5b, there is an output called ”abcdei” or ”fghj,” which can have one or two different values. When two options are available, an additional parameter must be considered. This parameter is called Running Disparity (RD) and is responsible for maintaining the long-term ratio of ones and zeros transmitted to be almost 50%, with a maximum difference of ±2 symbols.

When the disparity of code words is ±2, RD changes from -1 to $+1$ and vice-versa. Conversely, when the disparity of the code word remains neutral, RD retains its previous value.

After encoding the two blocks, the data is concatenated again but with the order of the blocks reversed, as shown in Figure 2.10.

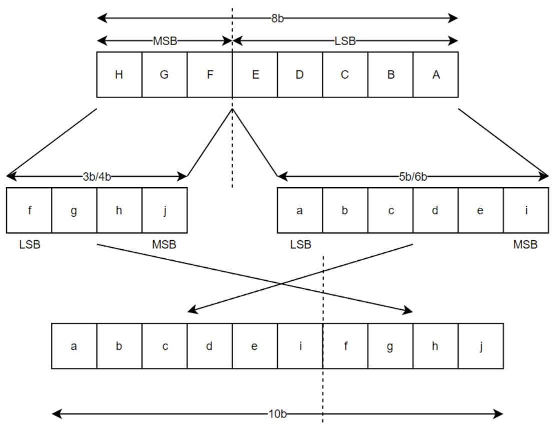
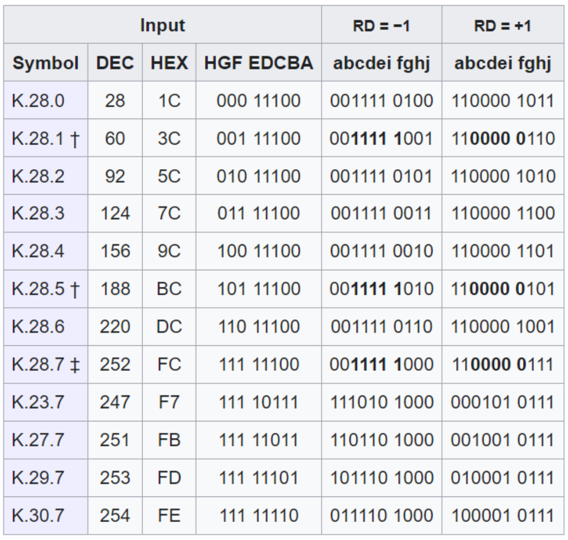


Figure 2.10: 8b/10b encoding process.

In addition to the data symbols that can be created, the 8b/10b encoding system includes 12 extra symbols called ”K” symbols or Control symbols. A list of these symbols can be found in Figure 2.11. Control symbols have the particular characteristic that they do not have a corresponding 8b data byte and they are normally used for synchronization and low-level control functions.



Input				RD = −1	RD = +1
Symbol	**DEC**	**HEX**	**HGF EDCBA**	**abcdei fghj**	**abcdei fghj**
K.28.0	28	1C	000 11100	001111 0100	110000 1011
K.28.1 †	60	3C	001 11100	00**1111 1**001	11**0000 0**110
K.28.2	92	5C	010 11100	001111 0101	110000 1010
K.28.3	124	7C	011 11100	001111 0011	110000 1100
K.28.4	156	9C	100 11100	001111 0010	110000 1101
K.28.5 †	188	BC	101 11100	00**1111 1**010	11**0000 0**101
K.28.6	220	DC	110 11100	001111 0110	110000 1001
K.28.7 ‡	252	FC	111 11100	00**1111 1**000	11**0000 0**111
K.23.7	247	F7	111 10111	111010 1000	000101 0111
K.27.7	251	FB	111 11011	110110 1000	001001 0111
K.29.7	253	FD	111 11101	101110 1000	010001 0111
K.30.7	254	FE	111 11110	011110 1000	100001 0111

Figure 2.11: 8b/10b control symbols.

From the above symbols, it is important to mention that there are some of them that are particularly important for JESD204B protocol. The following list shows these symbols and its meaning:

Control Character	Control Symbol	8b	10b, RD = -1	10b, RD = +1	Description
/R/	K28.0	000 11100	001111 0100	110000 1011	Start of multiframe
/A/	K28.3	011 11100	001111 0011	110000 1100	Lane alignment
/Q/	K28.4	100 11100	001111 0010	110000 1101	Start of link configuration data
/K/	K28.5	101 11100	001111 1010	110000 0101	Group synchronization
/F/	K28.7	111 11100	001111 1000	110000 0111	Frame alignment

Table 2.1: 8b/10b control symbols relevant for JESD204B.

2.4.3 Link Layer

In this subsection, our focus will be dedicated to the requisite phases of the JESD204B protocol necessary for establishing synchronization between TX and RX devices.

Aside from the synchronization process, it's important to note that this layer also takes responsibility for executing data scrambling, although this feature will be deactivated for the current project. Data scrambling is employed to randomize the transmitted data. The practice of scrambling is vital for mitigating electromagnetic interference (EMI), enhancing signal integrity, and reducing the effects of prolonged sequences of consecutive identical digits (CID), which may trigger timing recovery issues in the receiver.

2.4.3.1 Synchronization

The first stage is called Code Group Synchronization (CGS) described in Paragraph 2.4.3.1.1. In this case, the FPGA must locate K28.5 characters transmitted from the ADC. Once a certain number of consecutive characters have been detected on all link lanes, the receiver block deasserts the SYNC signal.

The second stage is called Initial Lane Synchronization (ILAS) described in Paragraph 2.4.3.1.2. Its main purpose is to align all the lanes of the link, verify the link parameters, and establish where the frame and multiframe boundaries are in the incoming data stream at the receiver. During this stage, the link parameters are sent to the FPGA to designate how data will be transmitted.

In the final stage, the transmitter (in this case, the ADS54J54 ADC described in Section 2.5) can begin sending out data, which should be pre-processed in the transport layer and then used in the application layer.

2.4.3.1.1 Code Group Synchronization (CGS) Figure 2.12 shows a time diagram of how CGS phase works. In this Figure we can see that $/K/$ characters are aligned and received at the same time, which in real life is not mandatory but the protocol implements techniques for correcting it. In addition to the Figure, a brief description of the steps is highlighted in the following list:

1. The Rx requests synchronization by driving the $\overline{SYNC}$ pin low.
2. The Tx transmits /K28.5/ symbols, unscrambled, starting with the next symbol.
3. The Rx synchronizes upon receiving at least four consecutive /K28.5/ symbols without error, after which it drives the $\overline{SYNC}$ pin high.

4. If the Rx fails to receive at least four 8B/10B characters without error, synchronization fails and the link remains in the CGS phase.
5. The CGS phase ends, and the ILAS phase begins.

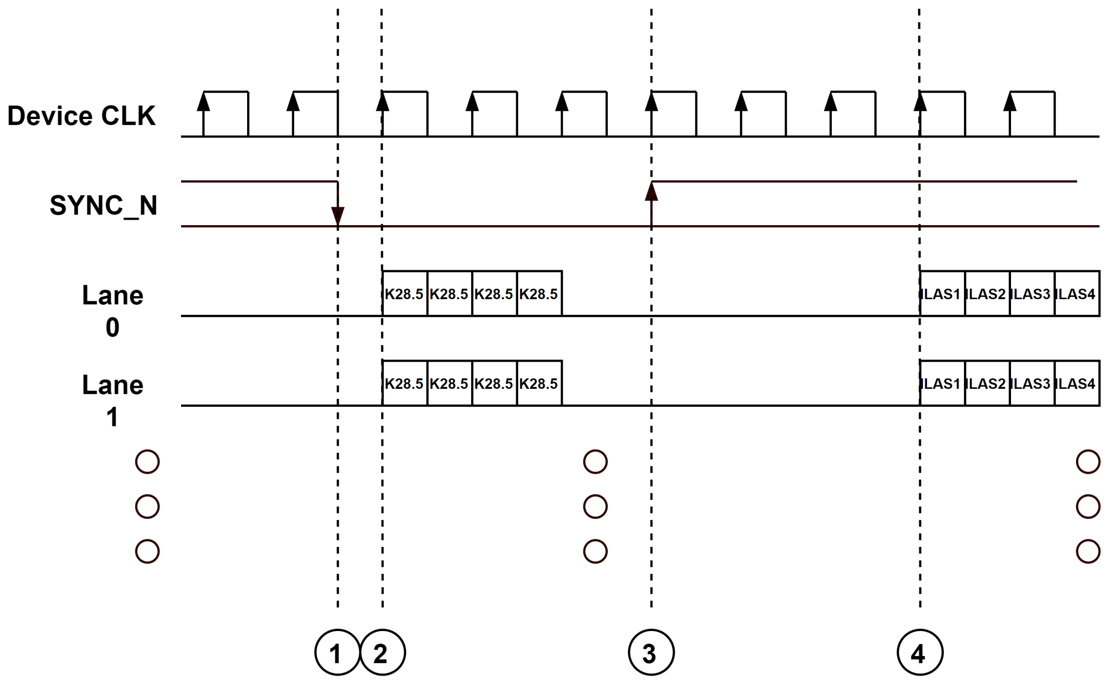


Figure 2.12: CGS phase.

Within the JESD204B standard, the /K28.5/ character, also known simply as /K/, is easy to recognize while debugging since the hexadecimal value is $0xBC$. This will be important in the Verification section (Section 3.3) where we will take a look at the signal tap and simulation testbenches in order to see how JESD204B synchronization is working in real life.

2.4.3.1.2 Initial Lane Synchronization (ILAS) The ILAS phase begins after $\overline{SYNC}$ has been deasserted (goes high). After the transmit block has internally tracked (within the ADC) a full multiframe, it will begin to transmit four multiframes. Dummy samples are inserted between the required characters so that full multiframes are transmitted (Figure 2.13). The four multiframes consist of the following:

- **Multiframe 1**: begins with an /R/ character [K28.0] and ends with an /A/ character [K28.3].
- **Multiframe 2**: begins with an /R/ character followed by a /Q/ [K28.4] character, followed by link configuration parameters over 14 configuration octets (Table 2.2), and ends with an /A/ character.
- **Multiframe 3**: the same as Multiframe 1.
- **Multiframe 4**: the same as Multiframe 1.

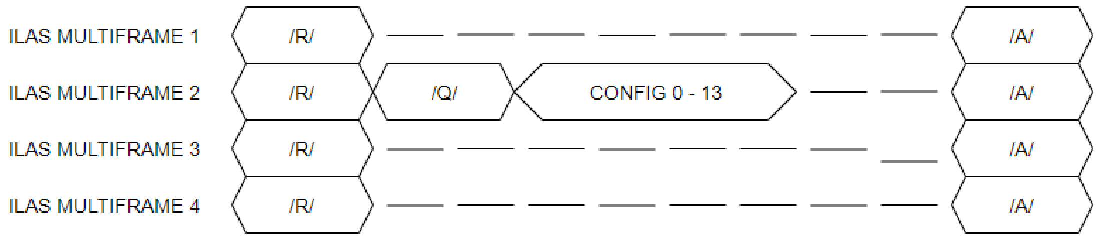


Figure 2.13: ILAS phase.

Octect Number	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Description
0	DID[7:0]								DID = Device ID
1	AJCNT[3:0]				BID[3:0]				ADJCNT = Number of adjustment resolution steps BID = Bank ID
2	0	ADJDIR	PHADJ	LID[4:0]					ADJDIR = Direction to adjust DAC LMFC PHADJ = Phase adjustment request LID = Lane ID
3	SCR	0		L[4:0]					SCR = Scrambling enabled/disabled L = Number of lanes per device(link)
4	F[7:0]								F = Number of octets per frame per lane
5	0			K[4:0]					K = Number of frames per multiframe
6	M[7:0]								M = Number of converters per device
7	0	CS[1:0]		N[4:0]					CS = Number of control bits per sample N = Converter resolution
8	SUBCLASS[2:0]			N_PRIME[4:0]					SUBCLASSV = Subclass version N_PRIME = Total bits per sample
9	JESDV[2:0]			S[4:0]					JESDV = JESD204 version S = Number of samples per converter per frame
10	HD	0		CF[4:0]					HD = High Density data format CF = Number of control words per frame clock per link
11	RESERVED 1								RES1 = Reserved. Set to 8'h00
12	RESERVED 2								RES2 = Reserved. Set to 8'h00
13	FCHK[7:0]								FCHK is the modulus 256 of the sum of the 13 configuration octets above.

Table 2.2: ILAS configuration octets.

2.4.3.1.3 SYSREF Synchronization It is important to mention that in order to execute the synchronization, it is important to synchronize the SYSREF signal to the Link clock. In order to correctly synchronize it, it is necessary to use two flip-flops in cascade mode as shown in Figure 2.14. This is an Intel recommendation.

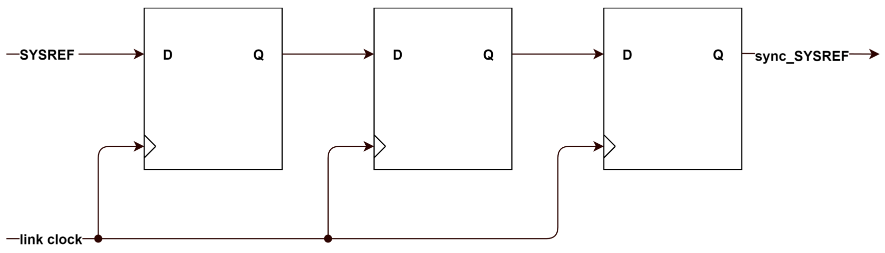


Figure 2.14: SYSREF synced with rxlink_clk.

2.4.4 Transport Layer

The transport layer is tasked with organizing the data in the proper sequence, enabling us to use the output of this layer as the raw data suitable for subsequent data processing, data storage, and so forth.

This layer is not incorporated within the Intel IP core, thereby necessitating its independent creation. The TX version of this layer will be elaborated in Subsubsection 2.5.1.1, where we will delve into the process by which the ADC constructs the frame to be received by the FPGA. Conversely, the RX version of this layer will be expounded upon in Paragraph 3.2.2.4.3, detailing the implementation of this layer for the FPGA.

2.5 ADS54J54 ADC

The ADS54J54 [8] is a 14-bit 500 MSPS low-power, wide-bandwidth quad-channel analog-to-digital converter (ADC) that supports the JESD204B serial interface with data rates up to 5 Gbps, enabling 1 or 2 lanes per ADC. Its buffered analog input maintains consistent input impedance across a broad frequency range while minimizing sample-and-hold glitch energy. The sampling clock divider allows for greater flexibility in system clock architecture design. The ADS54J54 delivers excellent spurious-free dynamic range (SFDR) over an extensive input frequency range with minimal power consumption. Additionally, an optional 2x Decimation Filter offers high-pass or low-pass filter modes.

In the upcoming subsections, we will concentrate on two primary functionalities of the ADC. The first, presented in Subsection 2.5.1, will examine the JESD204B from the perspective of a TX device, such as the ADS54J54. Here, we will elucidate how the TX frame is assembled and transmitted. Conversely, Subsection 2.5.2 will focus on configuring the ADS54J54 via the Serial Peripheral Interface (SPI), including the construction of register read and write requests, among other details.

2.5.1 ADS54J54 JESD204B Capabilities

The ADS54J54 is compatible with device subclass 1 and supports a maximum output data rate of 5 Gbps per serial transmitter. It allows individual JESD204B format configuration for channels A and B, as well as channels C and D. An external SYSREF signal aligns all internal clock phases and the local multi-frame clock with a specific sampling clock edge, enabling synchronization of multiple devices in a system and minimizing timing and alignment uncertainty. SYNCbAB input controls all the JESD204B SerDes blocks for channels A and B, while SYNCbCD controls channels C and D. If the same LMFS configuration is used for all four channels, the SYNCbAB and SYNCbCD signals can be externally connected and driven by a single source. Figure 2.15 depicts the internal schematic of Channel A and B, illustrating the path from the differential hardware lanes of various signals to the JESD204B output.

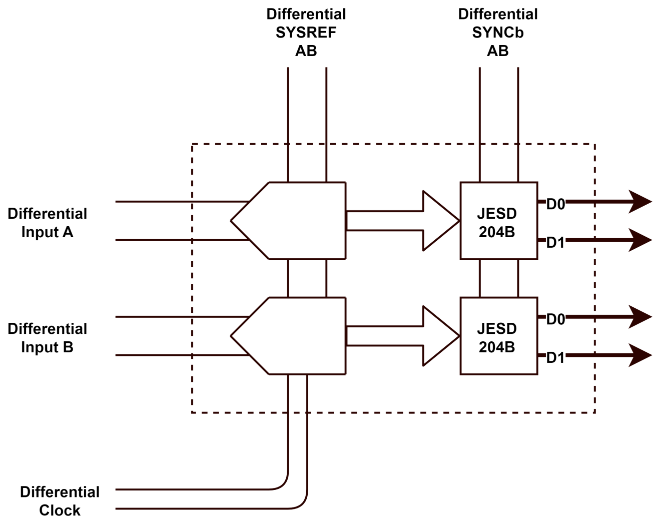


Figure 2.15: JESD204B Lane Assignment.

In addition, a more detailed view of the JESD204B block is shown in Figure 2.16 :

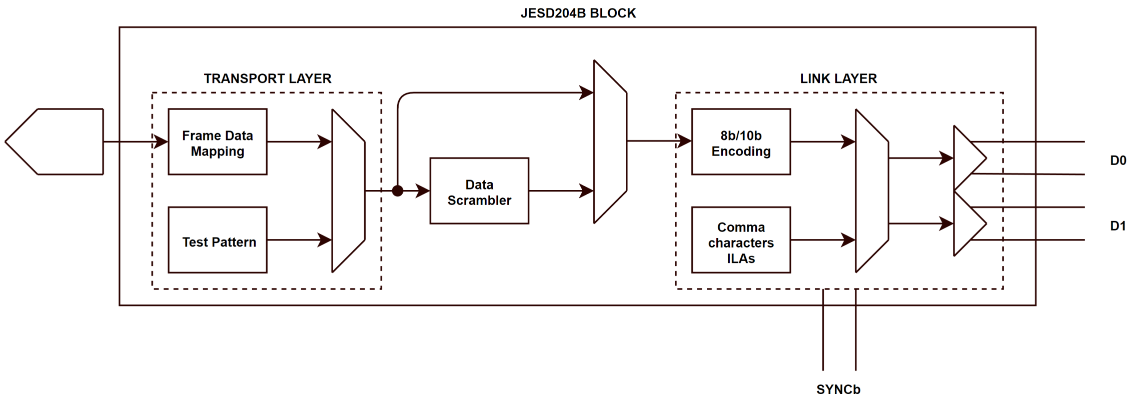


Figure 2.16: JESD204B Block detailed.

This figure provides a visualization of the different sub-blocks in the JESD204B implementation within the ADS54J54. On one side, we have the transport layer composed of the Frame data mapping (Subsubsection 2.5.1.1) and a test pattern module, which will be exceedingly helpful for debugging in the Verification Section (Section 3.3).

Conversely, there's a data scrambler that feeds a multiplexer but also allows for the retrieval of data without scrambling, an option that will be utilized for this thesis.

Lastly, a link layer is present, housing the 8b/10b encoding module as well as a module for transmitting the ILAS multiframes without encoding during the ILAS phase.

2.5.1.1 Frame Assembly

In this subsection, our focus will be on the functioning of the ADS54J54 as a JESD204B transmitter (TX). This is an integral part of the project as it aids in better understanding the JESD204B Transport Layer, detailed in Paragraph 3.2.2.4.3.

The frame assembly process is illustrated in Figure 2.17:

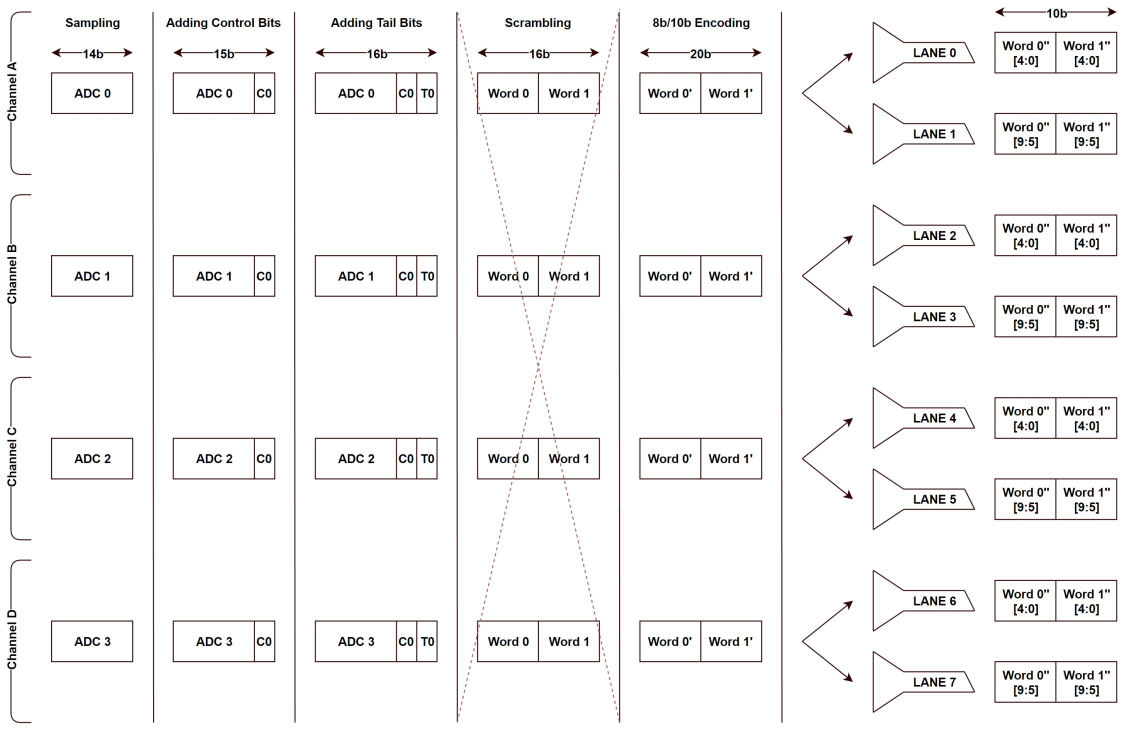


Figure 2.17: ADS54J54 JESD204B Frame Assebly.

As illustrated, the process is symmetrical for all channels, so we will concentrate on channel A to grasp the procedure.

1. The analog-to-digital converter (ADC) samples the input analog signal for each channel and converts it into a digital representation.
2. Control bits are added to the digital samples to indicate the start and end of a frame, as well as any other control information required by the JESD204B protocol. These control bits help the receiver identify and manage the data correctly.
3. Tail bits are added to the digital data to ensure that the total number of bits in a sample (including control bits) is a multiple of 4 . These tail bits are typically zeros and help maintain a consistent data structure throughout the frame.
4. The data stream, including control and tail bits, is scrambled using a pseudo-random bit sequence generated by a linear feedback shift register (LFSR).

5. The scrambled data stream is then encoded using the 8b/10b encoding scheme. This process maps each 8-bit data byte to a 10-bit transmission character, which helps maintain DC balance, improves signal integrity, and allows for better clock recovery at the receiver end.

6. The encoded data is distributed across the available high-speed serial lanes according to the lane assignments and configuration defined by the JESD204B protocol. This step involves interleaving the encoded data from multiple channels and distributing it over the lanes for transmission to the receiving device.

Moreover, it's crucial to consider the calculations of data transmitted through this system. Given that the ADC's sampling frequency is 500 MSPS, N' is 16 bits and there are four channels with 8b/10b encoding:

$$500 \cdot 10^6[MSPS] \cdot 16[bits/sample] \cdot 4[Channels] \cdot \frac{10}{8}[Encoding\ ratio] = 40\ Gbps. \tag{2.1}$$

Therefore, the volume of data transmitted through each lane is:

$$\frac{40 \cdot 10^9}{8} = 5\ Gbps. \tag{2.2}$$

2.5.2 SPI Configuration

Finally, the SPI protocol (more about how the SPI protocol works in Subsection 2.6.2) is employed to configure the ADC internal registers.

ADS54J54 is working in 4 pin configuration for this thesis. At this mode, there will be used the following pinout:

- **SDENb**: Active low enable.
- **SCLK**: System clock.
- **SDATA**: Input data lane.
- **SDOUT**: Output data lane.

Additionally, section 7.5 of the datasheet [8] provides instructions for programming the ADC through the protocol. The list of actions is as follows:

1. Driver SDENb pin low.
2. Set the R/W bit to ”0” for read or ”1” to write.
3. Initiate a serial interface cycle specifying the address of the register.
4. Write 16 bit data which is latched on the rising edge of SCLK.

Figure 2.18 illustrates the frame structure of the ADS54J54 SPI:

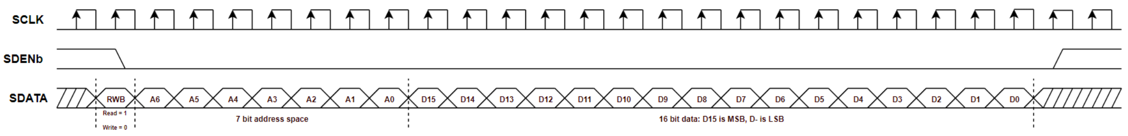


Figure 2.18: ADS54J54 SPI frame.

Here, we can observe that there is a single bit indicating whether the operation is read or write, denoted as R/W. Following that, there are 7 address bits utilized to select which ADC register is to be read or written. Ultimately, there are 16 data bits, employed for writing content into the register.

2.6 Communication Protocols

Another important section to be mentioned is the communication protocols that have been implemented and used for the different tasks: such as CPU communication, intra-module communication or ADC configuration.

2.6.1 Wishbone

The Wishbone Bus [17] is an open-source hardware computer bus designed to enable the components of an integrated circuit to communicate with one another. Its purpose is to facilitate the connection of diverse cores within a single chip.

Intended as a general-purpose interface, the Wishbone interconnect establishes the standard for data exchange between IP core modules. It does not, however, attempt to regulate the application-specific functions of the IP core.

Three factors significantly influenced the Wishbone architects during the protocol's development:

- First, there was a need for a robust, reliable System-on-Chip integration solution.
- Second, a common interface specification was necessary to promote structured design methodologies among large project teams.
- Third, they were inspired by the traditional system integration solutions provided by microcomputer buses, such as the PCI bus and VMEbus.

In this thesis, we will concentrate on the protocol's most fundamental behaviour, without delving into the intricacies of all signals, as the objective of this section is to grasp the basics.

2.6.1.1 Interface

The main wishbone signals can be described in the following list :

- **RST_I** : Receives the reset output from the system.

- **CLK_I**: Received the clock output from the system.
- **ADDR_I**: Slave address input from the Master.
- **ADDR_O**: Master address output to slave.
- **DAT_I**: Slave input data.
- **DAT_O**: Master output data.
- **WE_I / WE_O**: Write enable (active high).
- **STB_I / STB_O**: Strobe (kind of chip select)
- **ACK_I / ACK_O**: Acknowledge (active high).
- **CYC_I / CYC_O**: Bus Cycle (active high).

Figure 2.19 shows the most simple Wishbone interface between a master and a slave device.

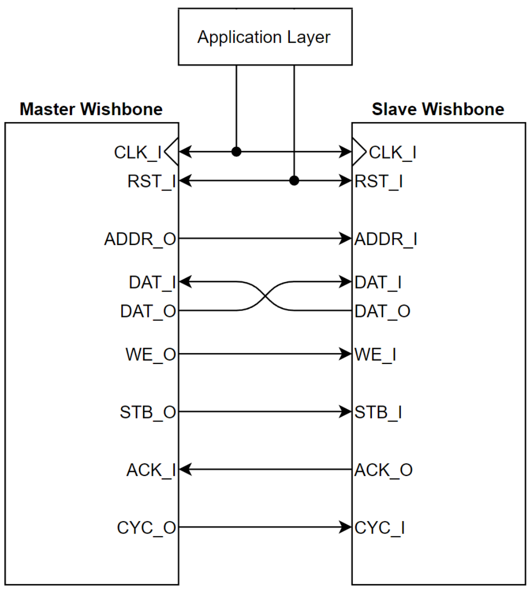


Figure 2.19: Simple Master-Slave Wishbone communication interface.

2.6.1.2 Data transmission

In order to understand how the protocol works, we are going to explain in a simple way what is the Handshaking and single Read/Write cycles. We will not take into account more advanced features as for example block Read/Write cycles or Read-Modify-Write (RMW) cycles.

2.6.1.2.1 Handshaking All bus cycles use a handshaking protocol between the MASTER and SLAVE interfaces. As shown in Figure 2.20, the MASTER asserts [STB_O] when it is ready to transfer data. [STB_O] remains asserted until the SLAVE asserts one of the cycle terminating signals [ACK_I], [ERR_I] or [RTY_I] [10] . At every rising edge of [CLK_I] the terminating signal is sampled. If it is asserted, then [STB_O] is negated. This gives both MASTER and SLAVE interfaces the possibility to control the rate at which data is transferred.

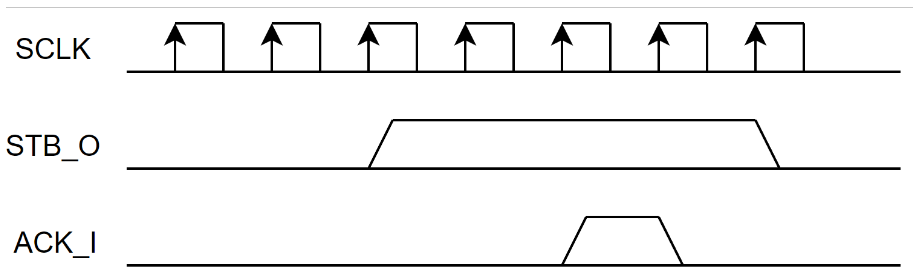


Figure 2.20: Wishbone Handshake.

2.6.1.2.2 Single Read Cycle The single read cycle can be divided into three steps, Figure 2.21b shows the whole process for the action.

At clock edge 0, MASTER presents address at [ADR_O] bus, sets [WE_O] to READ cycle [11] , and selects the bank with [SEL_O]. In addition, MASTER asserts [CYC_O] to initiate the cycle, then asserts [STB_O] to begin the phase.

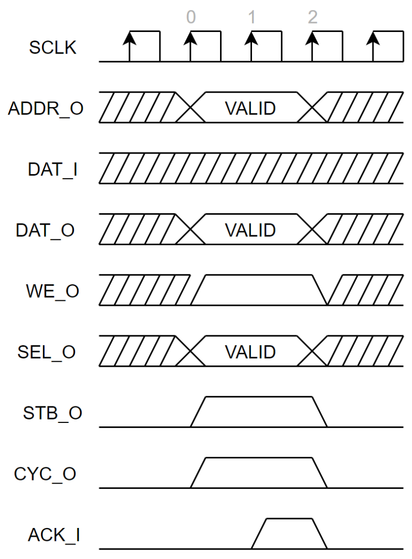
On the next setup before the clock cycle 1, SLAVE decodes inputs, asserts [ACK_I], and presents data on [DA_I]. MASTER monitors [ACK_I] and prepares to latch data.

Finally, on the clock edge 1, MASTER latches data from [DAT_I], then negates [STB_O] and [CYC_O] to terminate the cycle. At the same time, SLAVE negates [ACK_I] in response to the negated [STB_O].

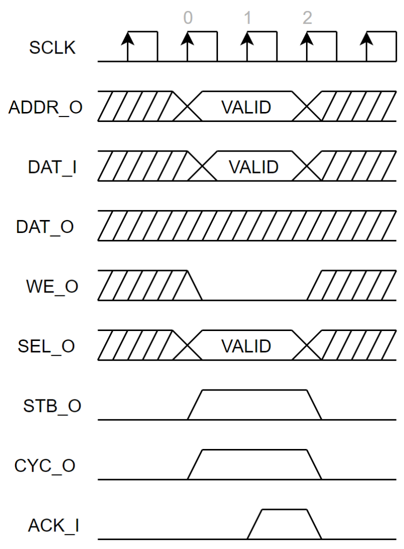
2.6.1.2.3 Single Write Cycle The single write cycle is quite similar to the read cycle, with the primary difference being that [WE_O] is asserted to ”1” this time, indicating a write cycle. In this case, the MASTER presents output data on the [DAT_O] bus.

[10][ERR_I] and [RTY_I] stand for error and retry signals. Both are used in order to extend basic wishbone functionalities but as mentioned before, we are not using them in this thesis.

[11][WE_O] is asserted 0 to indicate read and asserted 1 to indicate write.



(a) Wishbone single write cycle.



(b) Wishbone single read cycle.

2.6.2 Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication. The interface was developed by Motorola in the mid-1980s and has become a *de facto* standard.

SPI devices communicate in full duplex mode [12] using a master–slave architecture usually with a single master. The master device originates the frame for reading and writing. Multiple slave devices may be supported through selection with individual chip select (CS).

2.6.2.1 Interface

Interfacing the SPI protocol is quite simple since most of the time it only needs four wires (sometimes it uses only three). These wires are described as :

- **SCLK**: Serial Clock (output from master).
- **MOSI**: Master Output Slave Input (data output from master).
- **MISO**: Master Input Slave Output (data output from slave).
- **CS**: Chip Select (output from master to indicate that data is being sent).

Figure 2.22 shows the most simple SPI interface between a master and a slave device.

[12] In a full duplex communication system, data transmission is bidirectional and simultaneous. Both parties can send and receive information at the same time, allowing for more efficient communication.

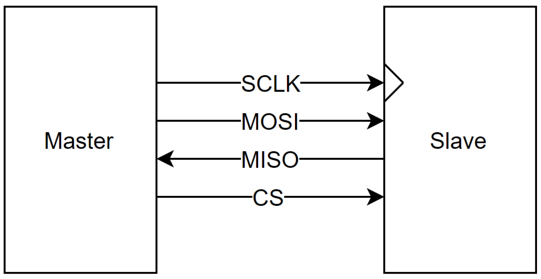


Figure 2.22: Simple Master-Slave SPI communication interface.

2.6.2.2 Data transmission

To initiate a data transmission, the master device applies a clock signal to the slave devices via the SCLK lane. This clock signal typically operates at a frequency below 10 MHz.

Once the clock is established, it is necessary to indicate the selected slave device for data transmission. This is commonly achieved by applying a 0 value to the $\overline{CS}$ of the target slave or slaves. In some cases, it may also be necessary to wait for a short period before initiating the transmission, as the slave might require time to make the data available (e.g., in certain ADCs).

During each SCLK cycle, full-duplex data transmission takes place. The master sends a bit through the MOSI line, while the slave utilizes the MISO line for the same purpose. The data is then saved in a shift register until the communication is complete, which is typically marked by asserting the CS to 1 once again.

Figure 2.23 shows a simple full-duplex communication between a master and a slave.

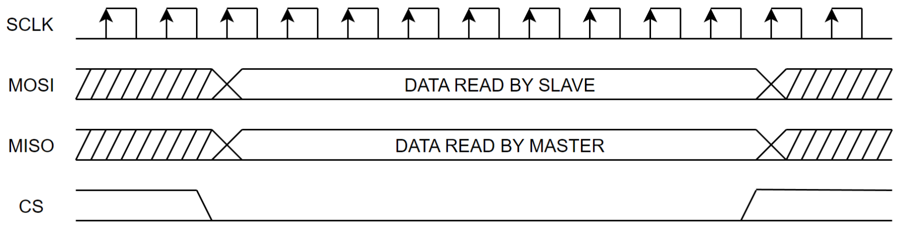


Figure 2.23: Simple full-duplex communication diagram.

2.6.3 VMEbus (Versa Module Eurocard bus)

VMEbus [18] is a computer bus standard, originally developed for the Motorola 68000 line of CPUs, but later widely used for many applications and standardized by the IEC as ANSI/IEEE 1014-1987. It is physically based on Eurocard sizes, mechanicals and connectors (DIN41612), but uses a different logical interface.

VMEbus has been traditionally used in HEP experiments [13] and accelerator control systems because it offers features such as:

[13]HEP stands for High Energy Physics, which is also known as particle physics.

- A well proven open standard that includes mechanical, electrical and protocol sections
- Suitable card sizes
- A data transfer protocol that is relatively easy to implement
- An “ecosystem” of third party products (crates, processors, I/O modules, etc.) which are supported by the manufacturers for about 10 years

The VMEbus uses a geographical addressing system. Geographical addressing is where the physical location of a board determines its address. The actual position of the board in the card cage (or rack) determines its base address. Each slot in the card cage has its own unique address range, which is determined by the backplane [14].

Apart from the address, an Address Modifier is also needed. This element is a part of the data transfer protocol and it stipulates the type of address space to be utilized during the transaction, as well as the nature of the data transfer itself. For instance, address modifier $0x09$ indicates a 32-bit non-privileged data access. This address modifier has been predominantly used in this project and is employed for single-cycle data transfers. If we wanted to read or write larger memory spaces, we could opt for block access by using the address modifier $0x0b$.

[14]The backplane is essentially the main circuit board that interconnects the different components or cards in the system.

Chapter 3

Development & Results

This chapter will concentrate on the practical design and verification process undertaken for this project, rather than the theoretical aspects.

We will initially review the architecture of different systems, such as the VFC (Section 3.1), and the Mezzanine connected to the FMC connector (Section 3.2.2).

We will present an overview of the application architecture implemented (Figure 3.2), before delving deeper into each module that constitutes it. We will describe the various functionalities and the methods used to test them (Section 3.3) to ensure they meet the specifications outlined in Subsection 1.2.1.

By the conclusion of this chapter, in Section 3.4, we will present various plots demonstrating the results of reading back data from the RAM memory upon the receipt of a trigger signal.

3.1 VFC

As outlined in Section 3.1, this board was designed by the BI CERN group to serve as a common starting point for various projects. To this end, they also developed a firmware template for this board, which facilitates the development of different applications.

This template allows for the configuration of various board features such as the GPIO pins, different connectors, clock sources, and so on.

3.1.1 Architecture

The firmware of this board has a Top Level module. This top level module encapsulates three different blocks as shown in Figure 3.1.

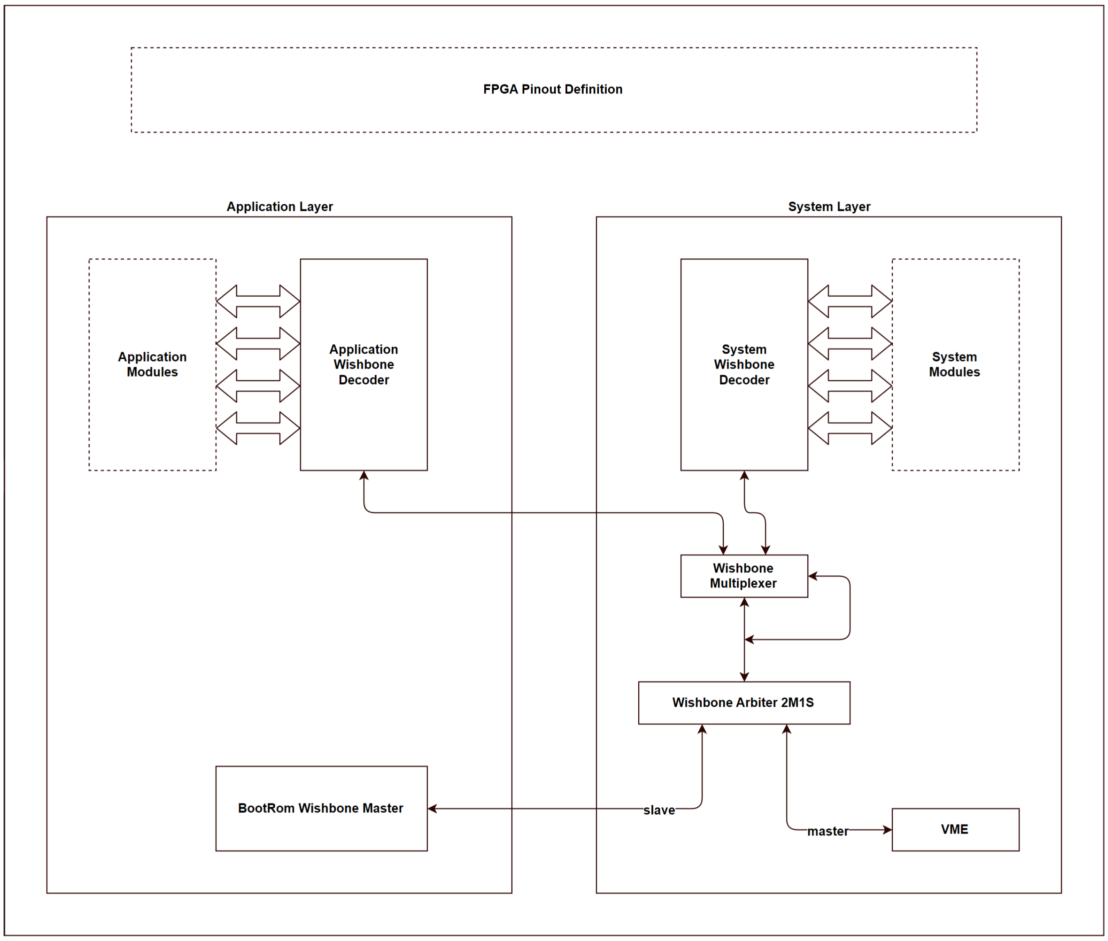


Figure 3.1: VFC Top Level module architecture.

The first block is the FPGA pinout configuration, where the definitions of the pins are displayed.

The second block is the System block, hosting common features utilized across all projects. In this layer, we encounter a VME-Wishbone bridge connected to a multiplexer. The purpose of this configuration is to convert VME data to Wishbone data and to determine if the command should be directed to the application layer or the system layer. Also included is a Wishbone decoder, which serves as a smart selector for modules. This module decides which Wishbone module should receive the Wishbone command from the VME component.

The third and final block is the Application block. This block is where all the modules listed later in this thesis are instantiated. The purpose of this block is to enable the developer to integrate their own logic into a larger framework. The block comprises a Wishbone arbiter, as in the System block, all the modules that we will examine in Section 3.2, and a BootRom module. This last module is a configuration file compiled into the FPGA for the purpose of issuing Wishbone commands at system startup. Thus, we can employ this module to automatically send Wishbone commands during boot up.

3.1.1.1 Application

For this thesis, we will focus on the Application layer since the System part is a common library that has not been modified. The Application overview can be found in Figure 3.2:

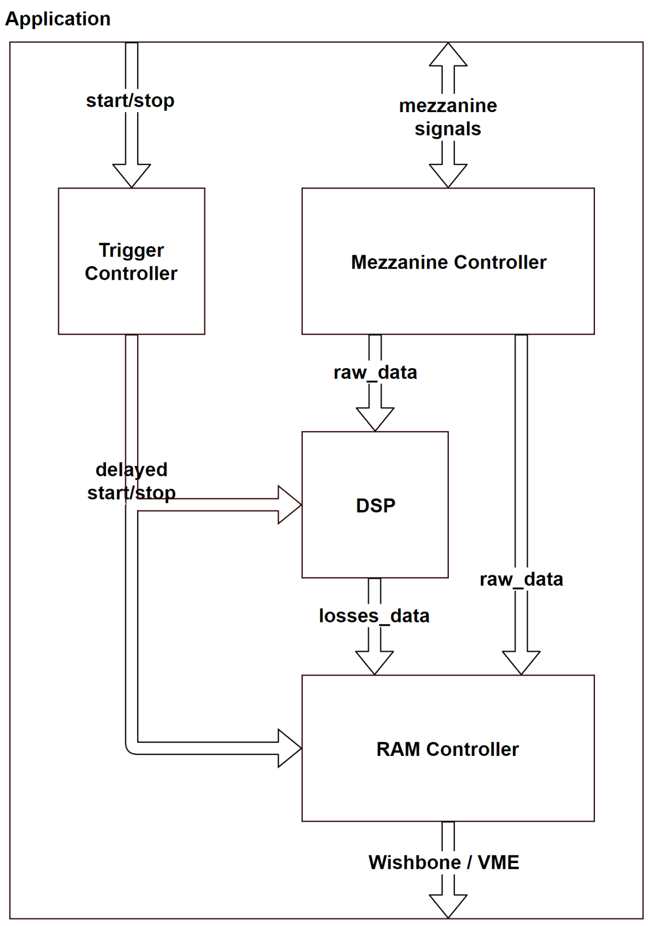


Figure 3.2: Firmware architecture.

Firstly, it's worth noting that the Wishbone address decoder, while not depicted in the previous figure, is connected to all the different modules, as we will see in Subsection 3.2.1.

Figure 3.1 illustrates the various blocks designed for this application. At the top, we can see the different inputs necessary for the applications, which include the start/stop signals originating from the CTRV (the timing board) and all the signals coming from the FMC connector (shown in Figure 3.5).

Timing signals are directly fed into the *TriggerController* (Subsection 3.2.4), where they are delayed by a configurable duration as outlined in the specifications of this thesis (Subsection 1.2.1).

Conversely, all the FMC signals are directed to the *MezzanineController* module (Subsection 3.2.2). This module houses all the logic for managing the mezzanine, from voltage control to the JESD204B protocol. This module is a complex entity divided into numerous other modules.

Once the raw data from the ADC is read by the FPGA through the *JESD204BController* and re-assembled by the transport layer, it is split into two different paths. The first path leads directly to the DSP module, where the ”per turn losses” will be calculated (for this thesis, this feature is not completed). The second path leads directly to the *RAM Controller*, where the data is stored for later read-back.

Lastly, there’s a *RAMController* module (Subsection 3.2.3), which is connected to the Wishbone, allowing the user to read back the data. This module stores the raw data from the ADC as well as the processed data from the DSP.

3.2 Design

The majority of this thesis revolves around the design and interconnection of all the modules specifically created for this application. Additionally, we utilized several modules from the Altera IP catalogue for implementing the RAM modules and the JESD204B protocol, as creating these from scratch would be an extensive task, warranting a separate thesis in its own right.

3.2.1 Wishbone Address Decoder

The Wishbone protocol has been utilized for module intercommunication, and is also connected to the VME bus, allowing the user to communicate (read and write) with various Wishbone registers created within the application or system layer.

As depicted in Figure 3.1, there is a Wishbone address decoder. This module is employed to determine which module should receive the information based on the address of the Wishbone command.

Figure 3.3 provides an overview of the address decoder module:

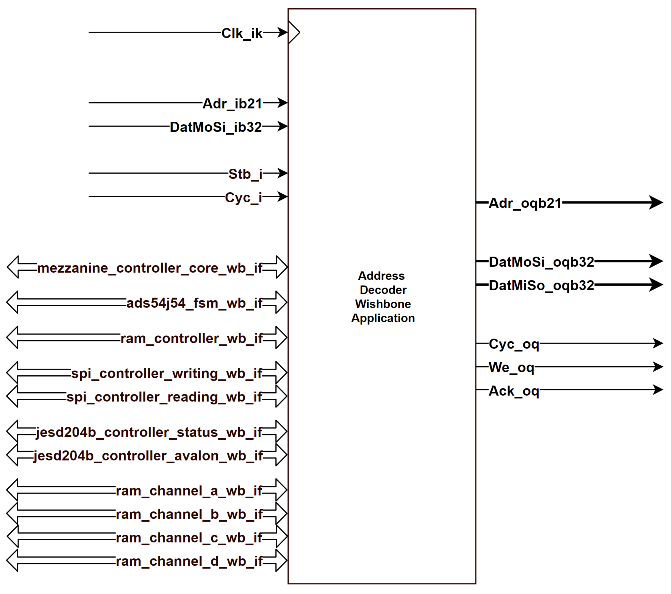


Figure 3.3: Address decoder module.

As can be seen, several Wishbone interfaces are connected to this module. Each of these interfaces is linked to different registers, which we will explore later in this section.

The module's operation is straightforward: it analyzes the address and determines which interface must be selected. Instead of comparing a unique address, it considers a range of addresses. This allows, for example, the reading of each of the RAM memories through the ram_channel_x_wb_if by addressing the range of addresses reserved for each RAM memory, as illustrated in Figure 3.4:

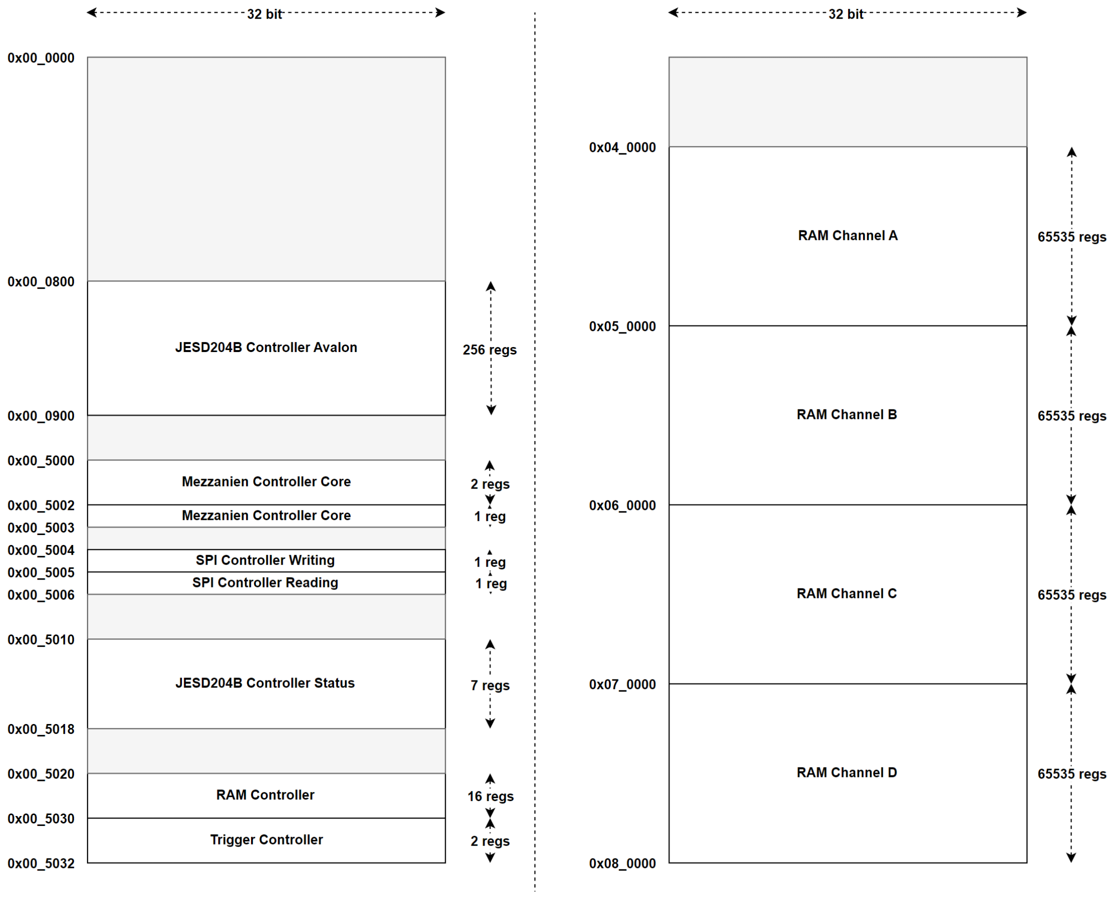


Figure 3.4: Address decoder memory map.

In this Figure we can realize how the wishbone memory is distributed for the different modules.

3.2.2 Mezzanine Controller

The mezzanine board is divided into different blocks (Figure 3.5), each with different responsibilities:

- **ADS54J54**: A 4-channel, 14-bit, 500 MSPS ADC from Analog Devices.
- **Si5394**: External PLL for generating various signals used for the correct operation and synchronization of the system with different input reference sources.

- **DC/DC Converter 1**: A DC voltage converter from $3.3V$ to $[-2.64V, 5V]$.
- **DC/DC Converter 2**: A DC voltage converter from $12.0V$ to $5V$.
- **EEPROM**: An I^2C accessible memory for data recording.
- **Analog front-end**: A front-end based on filters and a DC coupling system.

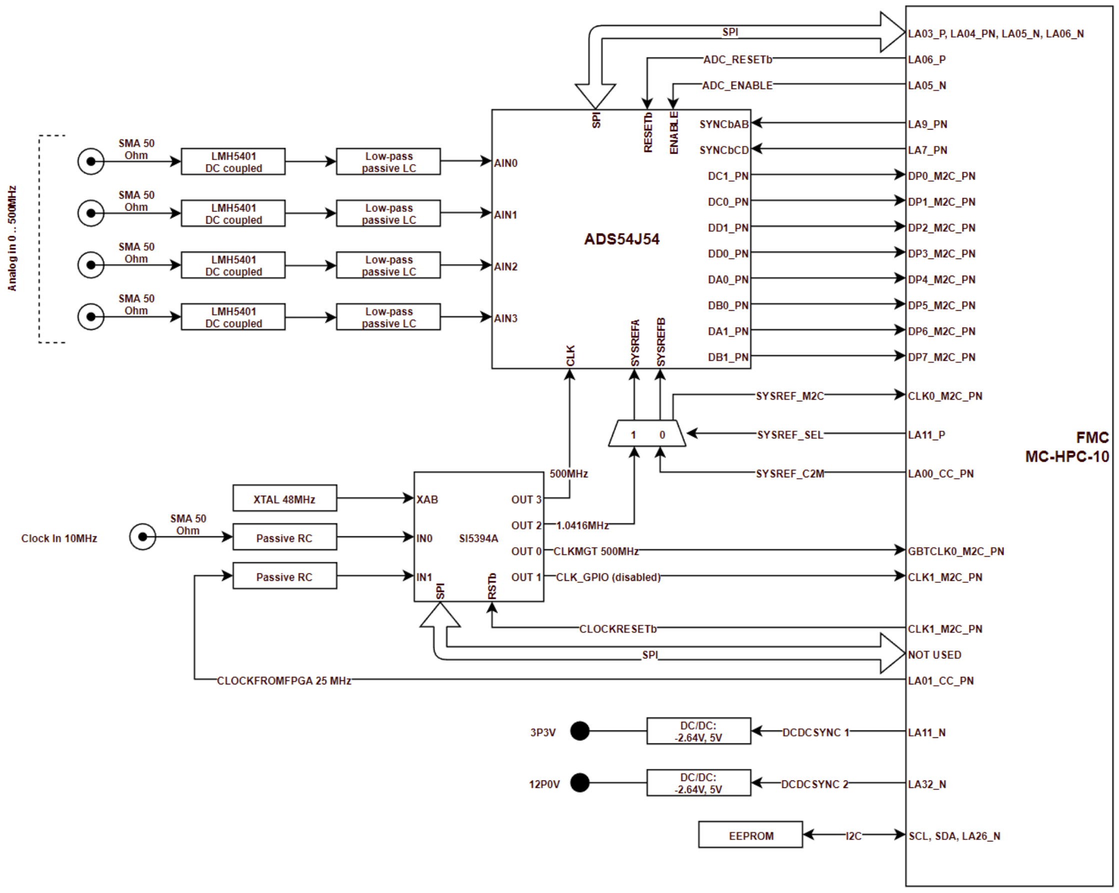


Figure 3.5: Mezzanine board block diagram.

As indicated above, there are several signals interconnected between the different blocks and the FMC connector. This allows the Application layer to read and drive the I/O, thereby controlling the different blocks.

The first module that one finds within the Application layer is the *MezzanineController*, which encapsulates all the logic that is specific to the IAM FMC card hardware, like all the configuration at boot time of the different ICs (ADS54J54, DC/DC converters, etc). This encapsulation aims to increase the code portability and re-use in case other instrumentation systems use a different FMC digitiser. It also allows to access all the internal registers of the ICs during normal operation through the Wishbone and VME bus for maintenance and diagnostic purposes.

Since IAM company has already programmed the Si5394 external PLL by saving some code into its internal non-volatile memory, there is no longer a need to configure it through SPI.

Si5394 PLL [9] has the following configuration:

Direction	Frequency	IO Standard	Description
IN0	10MHz	Single-ended 3.3V, AC-coupled	Clock input for external sync with SMA connector on front panel, highest priority (automatic selection)
IN1	25MHz	Single-ended 3.3V, AC-coupled	Clock input for external sync with clock generated by FPGA, lowest priority (automatic selection)
OUT0	500MHz	LVDS 1.8V, AC-coupled	GBT clock for the FPGA transceivers
OUT1	N.A	LVDS 1.8V, AC-coupled	General purpose clock output, disabled by default
OUT2	1.0467MHz	LVDS 1.8V, AC-coupled	SYSREF clock output for JESD204B interface, exact value: f = 500MHz/ 32 / 15
OUT3	500MHz	LVDS 1.8V, A-Ccoupled	Sampling clock for the ADC ADS54J54, output has lowest jitter

Table 3.1: Si5394 External PLL pinout configuration.

Within the *MezzanineController* module, we find the following sub-modules:

- **MezzanineVoltageController**: This module is responsible for the configuration at boot time and the continuous supervision of the voltage applied to the mezzanine card.
- **ADC_Controller**: This module is tasked with executing the ADS54J54 FSM for booting up. It also handles the configuration of ADS54J54 through SPI.
- **SPI_Controller**: This module is responsible for executing the automatic configuration of the ADC in sync with the ADC FSM, and also functions as a bridge between SPI and Wishbone.
- **JESD204B_Controller**: This module is in charge of executing the Link FSM and dividing the data from the JESD204B lanes into 14-bit wide channel data via the transport layer.
- **MezzanineController_reg**: This is a register bank for debugging and monitoring purposes.

Figure 3.6 shows a diagram of the module, along with all the wires and their respective directions.

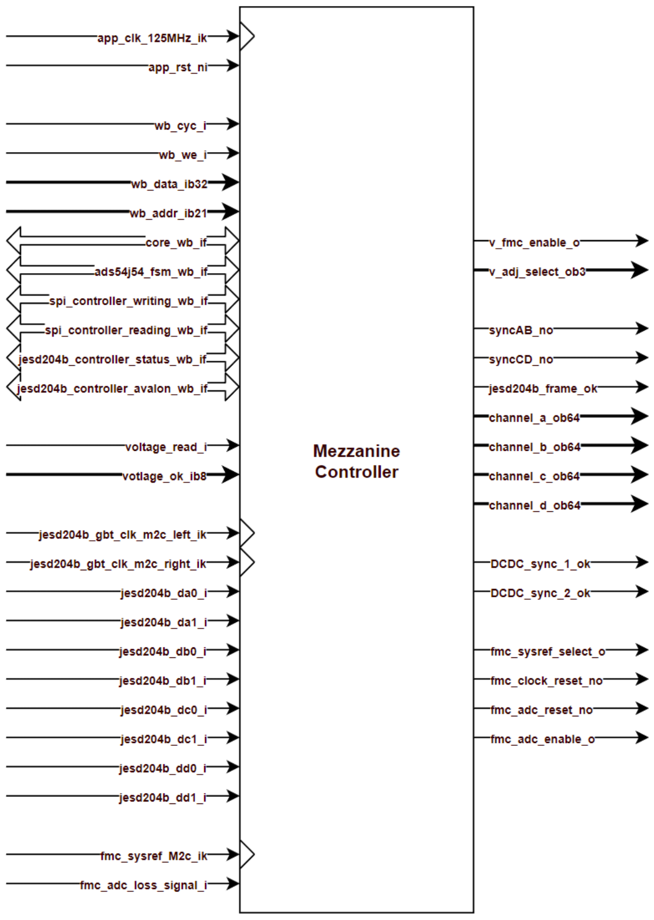


Figure 3.6: *MezzanineController* module.

In the aforementioned figure, we can observe that the module has a large number of inputs and outputs. This is attributable to the fact that the *MezzanineController* is used for encapsulation. Consequently, all signals that are meant to connect to each of the sub-modules must be available at the module definition.

3.2.2.1 Voltage Controller

The first module created for the project was the *MezzanineVoltageController* (Figure 3.7). The purpose of this module is to ensure that a specific voltage is applied to the mezzanine board through the FMC connector.

Given that the VFC board is a template for various projects, the mezzanine connected to the FMC connector may require a particular applied voltage. To accommodate this, the VFC card includes a voltage regulator which is controlled via the FPGA to select an output voltage within a range ($1.2V$ to $3.3V$). For the IAM500 mezzanine board, it is necessary to apply $1.8\ V$.

In addition to the output voltage controller, a local ADC allows to read back the applied output voltage to verify if it is correct. This is a double-check system designed to prevent damage to the mezzanine board, which could be quite costly. The firmware designed regularly surveys that voltage, and maintains a switch closed, powering the FMC card, while is kept stable and within an acceptance range, or opens it, protecting the FMC card, otherwise.

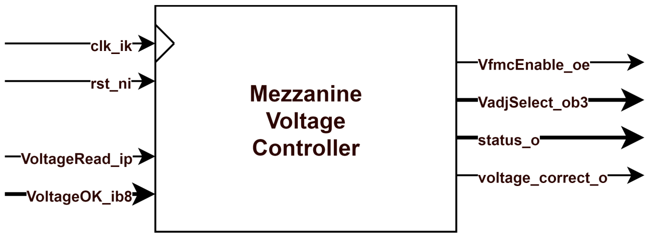


Figure 3.7: *MezzanineVoltageController* module.

In order to check that the voltage is correctly applied to the mezzanine board, there is a LED that brights up when $1.8\ V$ are applied as shown in Figure 3.8.

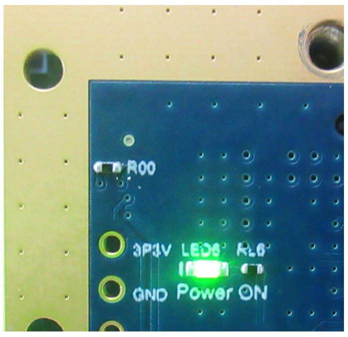


Figure 3.8: Mezzanine power-up LED.

3.2.2.2 ADS54J54 Controller

Current ADC ICs commonly contain a bank of internal registers that allow to use them with different configurations: enable/disable some of the converters, use different number of JESD lanes, activate their test modes, setup some over-range protection, to mention just a few. For this reason, once the voltage has been correctly established, it becomes necessary to follow a series of steps with the ADC signals originating from the FMC to boot up and configure the ADC [**technical_documentation_iam_cern**].

The following list delineates the procedure which must be followed by the finite state machine:

1. Set DCDCSYNC 1 and 2 to a LOW level.
2. Elevate SYSREF_SEL to a HIGH level.
3. Set both CLOCKRESETb and ADC_RESETb to a LOW level.
4. Set ADC_ENABLE to a LOW level.

5. Elevate CLOCKRESETb to a HIGH level.
6. Check if LossSignalb goes HIGH. If successful, proceed to the next step.
7. Verify if the frequency on SYSREF_M2C input is 1.046MHz. If successful, proceed to the next step.
8. Output the SYSREF frequency from the SYSREF_M2C input to the DCDCSYNC 1 and 2 output pins.
9. Set ADC_RESETb to a HIGH level.
10. Configure ADS54J54 over SPI (More details in Section 2.5.2).
11. Set ADC_ENABLE to a HIGH level.
12. Synchronize the JESD204B interface with the SYNCbAB and SYNCbCD signals (More information in Section 3.2.2.4).
13. Begin data capture.

3.2.2.3 SPI Controller

As previously stated, configuring the ADC necessitates the implementation of a module that utilizes the SPI protocol. To this end, a comprehensive module named *SPI_Controller* has been created (Figure 3.9), which encapsulates various features essential for SPI.

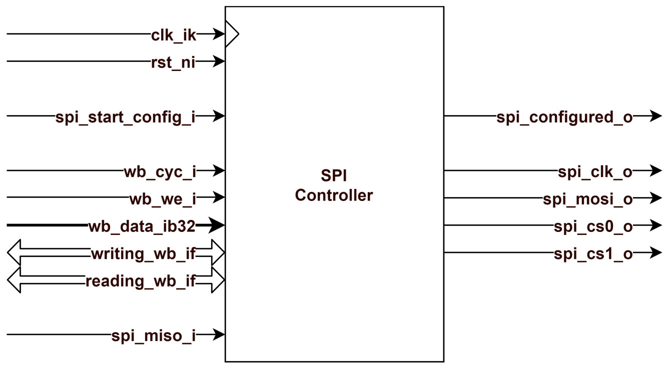


Figure 3.9: SPI Controller module signals.

The components that constitute the *SPI_Controller* module are listed below:

1. **ADS54J54 Master WB Configurator**: This module is responsible for automatically transmitting the configuration commands to the ADS54J54.
2. **WB to SPI Bridge**: This module serves as a conduit between Wishbone and SPI.
3. **Wishbone Multiplexer**: This module facilitates the connection of multiple Wishbone sources to a single output.

4. **SPI Listener**: This module is employed to capture SPI data when a read command is transmitted via Wishbone.

5. **SPI Controller Listener reg**: This is a Cheby register connected to Wishbone, designed to read back data captured by the SPI Listener module.

To better comprehend the dependencies of this module, let us refer to Figure 3.10, which illustrates the interconnections among the modules:

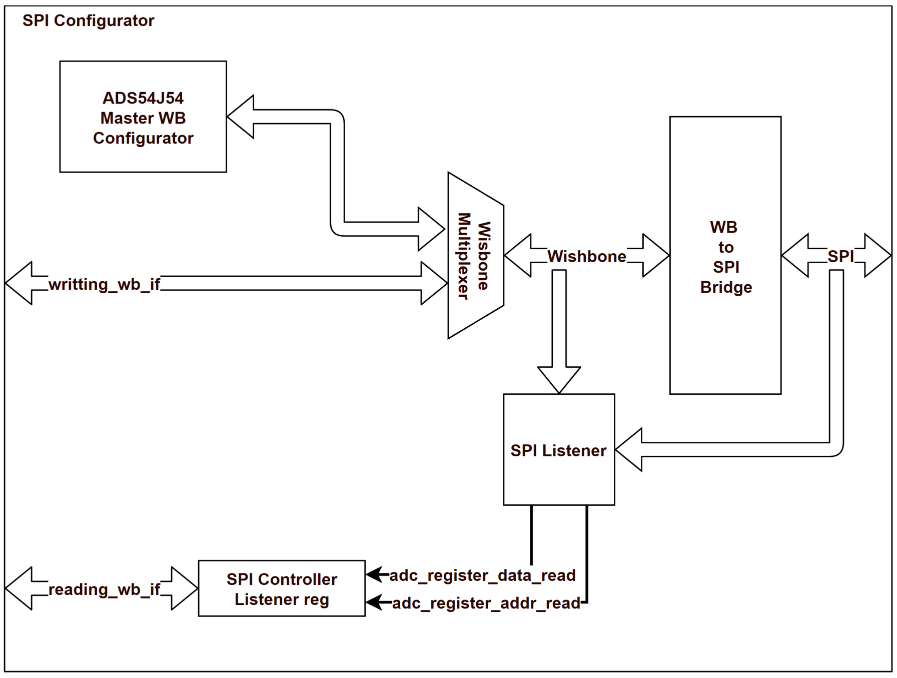


Figure 3.10: SPI Controller sub-modules details.

The intent of this figure is not to delve into the specifics of each signal, but rather to provide an understanding of the overall connections among the modules and the communication pathways.

3.2.2.3.1 ADS54J54 Master WB Configurator

This module responds to the *"spi_start_config_i"* signal, which is asserted by the main FSM as described in point 10 of the above list. When this signal is asserted, the ADS54J54 must receive a sequence of commands for configuration. After configuration is complete, this module will assert the signal *"spi_configured_o"* indicating the ADS54J54 main FSM that can move to next state.

These commands are employed to modify the default value of the internal ADC registers. As highlighted in Section 2.5.2, a "write" signal, along with the register's address and the new value, must be sent to the ADC.

This module incorporates a lookup table with pairs of address-value, and implements a counter system to send all the commands in the correct order. The counter increments a variable which serves as an index for the array in the lookup table.

The subsequent table is used as a reference for the order, value, and a brief description of each command sent to the ADC:

Order	Address	Value	Description
1	0x00	0x8000	Select 4 wires SPI configuration
2	0x0D	0x0000	Put JESD block in NON-INITIALIZATION state
3	0x0D	0x0202	Put JESD block in INITIALIZATION state
4	0x0D	0x0303	Deactivate reset bit for JESD block
5	0x01	0xAF7A	Fast OVR thresholds for CH A/B/C/D with read back
6	0x02	0x0000	
7	0x03	0x4040	Select Channel AB clock output divider for CH A/B/C/D
8	0x04	0x000F	SYNCbAB and SYNCbCD input buffers enabled
9	0x05	0x0000	Deactivate sleep mode controller by Enable Pin
10	0x06	0xFFFF	Deactivate sleep mode controller by SPI
11	0x07	0x0144	No clock phase of AB clock divider is changed
12	0x08	0x0144	No clock phase of CD clock divider is changed
13	0x0C	0x31E4	Skip one SYSREF pulse then use all pulses on JESD
14	0x0E	0x00FF	Enable all TX lanes for CH A/B/C/D
15	0x0F	0x0001	Select 2 converters per link for CH AB
16	0x10	0x03E3	Select 32 frames per multi-frame and 4 lanes for channel AB
17	0x13	0x0020	High density mode enabled for mode LMFS = 4221
18	0x16	0x0001	Select 2 converters per link for CH CD
19	0x17	0x03E3	Select 32 frames per multi-frame and 4 lanes for channel CD
20	0x1A	0x0020	High density mode enabled for mode LMFS = 4221
21	0x1D	0x0000	Deactivate test pattern in channel A/B/C/D
22	0x1E	0x0000	Deactivate sleep mode for JESD controlled by Enable pin
23	0x1F	0xFFFF	Deactivate sleep mode for JESD controller by SPI
24	0x20	0x0000	Outputs PRBS pattern selected in address 0x21 below
25	0x21	0x2000	Input full-sclare amplitude to 1.25Vpp and 16 bit PRBS output pattern
26	0x44	0x0074	trim value - required
27	0x47	0x0074	trim value - required
28	0x4C	0x4000	trim value - required
29	0x50	0x0800	trim value - required
30	0x51	0x0074	trim value - required
31	0x054	0x0074	trim value - required
32	0x59	0x4000	trim value - required
33	0x5D	0x0800	trim value - required
34	0x0D	0x0202	Put JESD block in INITIALIZATION state
35	0x0D	0x0303	Deactivate reset bit for JESD block
36	0x0D	0x0101	Put JESD in NON-INITIALIZATION state and deactivate reset bit for JESD

Table 3.2: ADS54J54 commands Table.

It is crucial to note that the register with address *0x1D* is utilized for configuring the operational mode of the ADC. Depending on the value of the register, there are three different modes, as indicated in the following list:

- **Normal operation**: When the register value is set to *0x00*, the ADC will commence sampling data from the channel inputs.
- **Chess pattern**: When the register value is set to *0x68*, the ADC will generate an alternating output pattern of *0x1555* and *0x2AAA*.

- **Triangle pattern**: When the register value is set to *0x60*, the ADC will generate a triangle pattern.

The implications of these variations in the register value will be discussed in the verification section of this thesis.

Additional configurable registers can be found in Section 7.6 of the ADS54J54 datasheet [8].

3.2.2.3.2 SPI Through Wishbone In addition to the automatic configuration of the ADC via the preceding module, there is also the option of accessing the SPI through the Wishbone bus.

To facilitate this, a Wishbone multiplexer has been instantiated. This multiplexer is directly fed from both the *ADS54J54_Master_Wb_Configurator* and the *SPI_writing_wb_if*.

This Wishbone interface is connected to the Wishbone Arbiter in the Application layer, allowing the user to manually write the ADC's registers individually, as demonstrated in Section 3.3. This feature is particularly important for testing and debugging because it enables the switching between ADC output modes, for example.

3.2.2.3.3 SPI Listener Finally, to read back the ADC registers, it's necessary to implement an *SPI_Listener* module, which is continuously monitoring signals from SPI and Wishbone for the ”read register” condition.

As discussed in Subsection 2.5.2, reading an ADC register requires specifying the action in a particular bit. The *SPI_Listener* module monitors this bit in order to start saving the read-back data into memory.

This method is necessary because the ADC begins transmitting the current value of the register as soon as it receives this bit set to ”1” via SPI.

This module outputs two separate variables:

- **adc_reg_addr_read_ob7**: 7-bit wide register address.
- **adc_reg_data_read_ob16**: 16-bit wide register read-back data.

These variables are stored in a register (*SPI_Controller_Listener_reg*), which is accessible through the Wishbone Arbiter in the Application layer.

3.2.2.4 JESD204B Controller

The *JESD204B_Controller* module (Figure 3.11) is responsible for executing the JESD204B Link FSM and for converting raw data arriving from the JESD204B lanes into distinct channel data, thereby generating a set of four samples per channel with each clock sample.

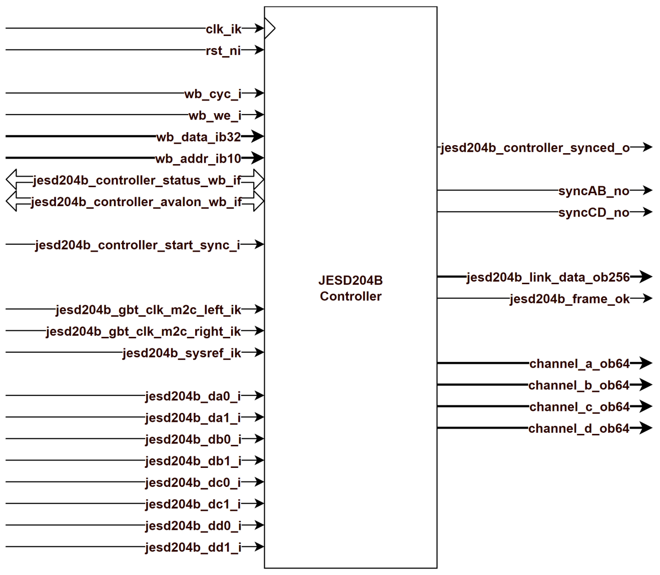


Figure 3.11: JESD204B Controller module signals.

This module is quite intricate in terms of signaling. The significance of Clock Domain Crossing (CDC) (Subsection 2.3.2) is paramount at this stage, as this module will operate with various clock sources such as:

- **clk_ik**: 125MHz clock from the Application layer.
- **jesd204b_gbt_clk_m2c_left_ik**: 500MHz clock from an external Phase-Locked Loop (PLL) feeding the left clock bank.
- **jesd204b_gbt_clk_m2c_right_ik**: 500MHz clock from an external PLL feeding the right clock bank.
- **jesd204b_sysref_ik**: 1.0467MHz clock from an external PLL for protocol synchronization and latency control.

A more detailed view of the clocking interconnection, as well as the module interconnection within the *JESD204B_Controller* module, can be found in Figure 3.12.

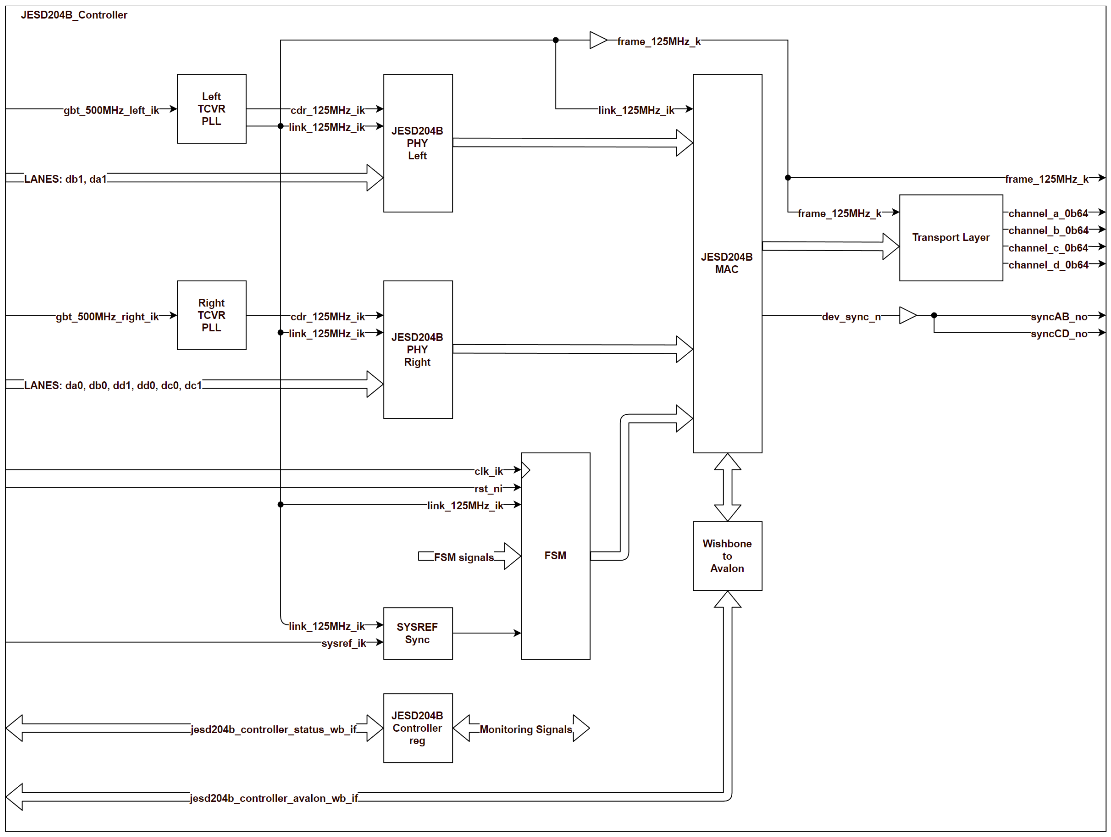


Figure 3.12: JESD204B Controller sub-modules details.

As demonstrated in the figure, both high-speed clocks feed a PLL. The purpose of this PLL is to create a 125MHz clock data recovery (CDR) clock (Paragraph 2.4.2.1.1) to feed the PHY layer. Additionally, in the case of the left PLL, it also generates a 125MHz link clock which is used for PHY, MAC, and even for synchronizing the SYSREF clock.

An additional clock, the frame clock, is necessary. The frequency of this clock depends on the JESD204B configuration (which can be found in Section 2.4), and for this project, it is also 125MHz. This clock must be synchronized to the link clock, hence we are assuming for this project that the link clock and frame clock are the same.

In addition to the PLL and the Altera IP cores, a few more modules are instantiated. The first one to highlight is the SYSREF synchronizer, the aim of which is to synchronize the SYSREF with the Link clock, as detailed in Paragraph 2.4.3.1.3. This synchronization is achieved using two series-connected flip-flops. Furthermore, it is essential to instantiate the *WishboneToAvalon* bridge module to enable reading the Avalon bus through Wishbone. This is mandated by Altera to ensure the proper functioning of the IP cores.

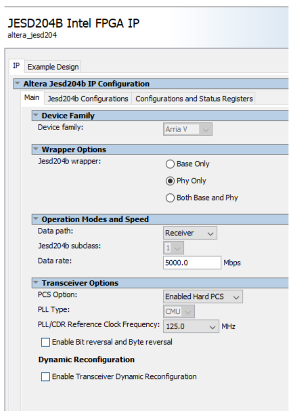
Lastly, it's worth noting that, as with the other modules of the *MezzanineController*, a custom Cheby register is instantiated for testing, debugging, and status monitoring of the signals.

3.2.2.4.1 Altera IP Core The JESD204B layer was not developed from scratch; instead, a mandatory Altera IP core [1] was used.

Additionally, due to the routing of the VFC (Section 3.1) board, it is not possible to create a single module. Instead, it is necessary to divide it into three separate modules:

- Left PHY
- Right PHY
- MAC

To instantiate the PHYs, it is necessary to indicate that the wrapper should be ”PHY only” in the wizard, as demonstrated in Figure 3.13:



JESD204B Intel FPGA IP
altera_jesd204

IP | Example Design

Altera Jesd204b IP Configuration

Main | Jesd204b Configurations | Configurations and Status Registers

Device Family
Device family: Arria V

Wrapper Options
Jesd204b wrapper: Base Only [✓ Phy Only] Both Base and Phy

Operation Modes and Speed
Data path: Receiver
Jesd204b subclass: 1
Data rate: 5000.0 Mbps

Transceiver Options
PCS Option: Enabled Hard PCS
PLL Type: CMU
PLL/CDR Reference Clock Frequency: 125.0 MHz
☐ Enable Bit reversal and Byte reversal

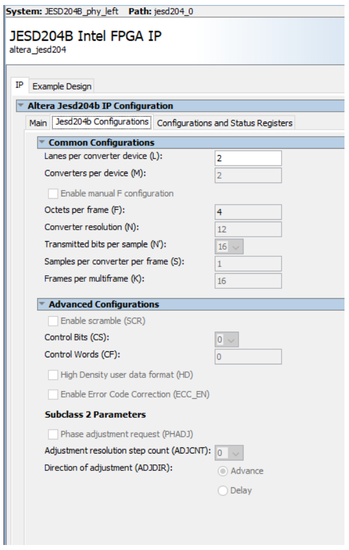
Dynamic Reconfiguration
☐ Enable Transceiver Dynamic Reconfiguration

Figure 3.13: Altera JESD204B wizard for PHY only instantiation.

Moreover, it is also crucial to specify other features such as whether it is a JESD204B receiver or transmitter, the Data rate for the lane (5Gbps), and the CDR frequency (125MHz).

Given that the two PHYs are not exactly identical (the left PHY receives only two lanes while the right PHY receives six lanes), it is necessary to specify the individual configuration in the ”JESD204B configuration” tab, as demonstrated in Figure 3.14a:

[1] An intellectual property core (IP core) is a functional block of logic or data used to make a field-programmable gate array (FPGA) or application-specific integrated circuit for a product. Normally this module is available through a company catalogue.



System: JESD204B_phy_left **Path:** jesd204_0

JESD204B Intel FPGA IP
altera_jesd204

IP | Example Design

Altera Jesd204b IP Configuration

Main | Jesd204b Configurations | Configurations and Status Registers

Common Configurations

Lanes per converter device (L): 2
Converters per device (M): 2
☐ Enable manual F configuration
Octets per frame (F): 4
Converter resolution (N): 12
Transmitted bits per sample (N'): 16
Samples per converter per frame (S): 1
Frames per multiframe (K): 16

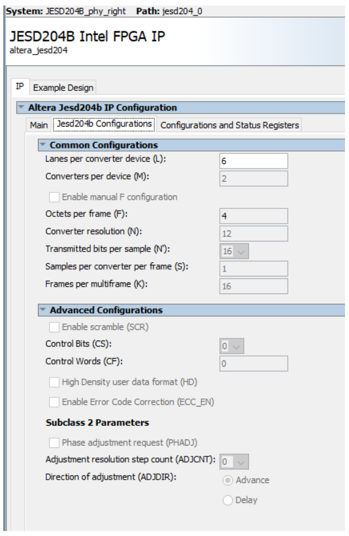
Advanced Configurations

☐ Enable scramble (SCR)
Control Bits (CS): 0
Control Words (CF): 0
☐ High Density user data format (HD)
☐ Enable Error Code Correction (ECC_EN)

Subclass 2 Parameters

☐ Phase adjustment request (PHADJ)
Adjustment resolution step count (ADJCNT): 0
Direction of adjustment (ADJDIR): [✓ Advance] Delay

(a) Altera JESD204B left PHY configuration.



System: JESD204B_phy_right **Path:** jesd204_0

JESD204B Intel FPGA IP
altera_jesd204

IP | Example Design

Altera Jesd204b IP Configuration

Main | Jesd204b Configurations | Configurations and Status Registers

Common Configurations

Lanes per converter device (L): 6
Converters per device (M): 2
☐ Enable manual F configuration
Octets per frame (F): 4
Converter resolution (N): 12
Transmitted bits per sample (N'): 16
Samples per converter per frame (S): 1
Frames per multiframe (K): 16

Advanced Configurations

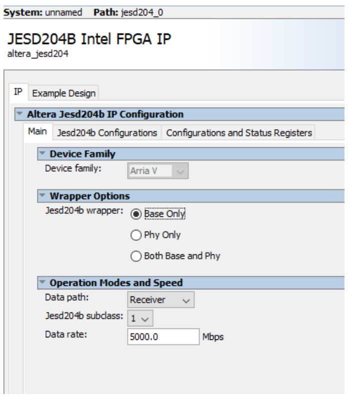
☐ Enable scramble (SCR)
Control Bits (CS): 0
Control Words (CF): 0
☐ High Density user data format (HD)
☐ Enable Error Code Correction (ECC_EN)

Subclass 2 Parameters

☐ Phase adjustment request (PHADJ)
Adjustment resolution step count (ADJCNT): 0
Direction of adjustment (ADJDIR): [✓ Advance] Delay

(b) Altera JESD204B right PHY configuration.

On the other hand, in order to instantiate the MAC layer, it is necessary to select the ”base only” option in the wizard as shown below:



System: unnamed **Path:** jesd204_0

JESD204B Intel FPGA IP
altera_jesd204

IP | Example Design

Altera Jesd204b IP Configuration

Main | Jesd204b Configurations | Configurations and Status Registers

Device Family

Device family: Arria V

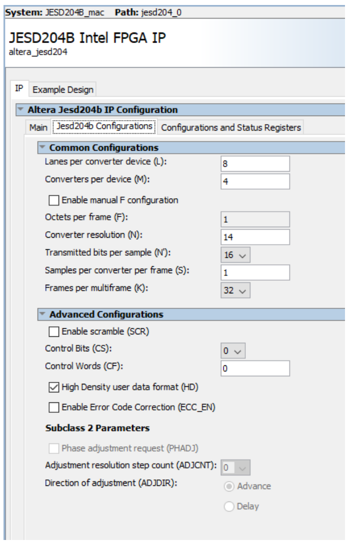
Wrapper Options

Jesd204b wrapper: [✓ Base Only] Phy Only Both Base and Phy

Operation Modes and Speed

Data path: Receiver
Jesd204b subclass: 1
Data rate: 5000.0 Mbps

(a) Altera JESD204B left PHY configuration.



System: JESD204B_mac **Path:** jesd204_0

JESD204B Intel FPGA IP
altera_jesd204

IP | Example Design

Altera Jesd204b IP Configuration

Main | Jesd204b Configurations | Configurations and Status Registers

Common Configurations

Lanes per converter device (L): 8
Converters per device (M): 4
☐ Enable manual F configuration
Octets per frame (F): 1
Converter resolution (N): 14
Transmitted bits per sample (N'): 16
Samples per converter per frame (S): 1
Frames per multiframe (K): 32

Advanced Configurations

☐ Enable scramble (SCR)
Control Bits (CS): 0
Control Words (CF): 0
☑ High Density user data format (HD)
☐ Enable Error Code Correction (ECC_EN)

Subclass 2 Parameters

☐ Phase adjustment request (PHADJ)
Adjustment resolution step count (ADJCNT): 0
Direction of adjustment (ADJDIR): [✓ Advance] Delay

(b) Altera JESD204B right PHY configuration.

By completing these steps, the wizard generates the necessary design and verification files for utilizing the JESD204B protocol. However, it is the responsibility of the designer to implement a finite state machine that controls the sequence of signals required to establish the JESD links as well as to implement the application specific transport layer. It’s also important to note that the verification files generated by Altera contain the MAC and PHY logical blocks combined in a single module, which does not correspond to the design implemented. Consequently, they cannot be used to validate the actual design, a topic which will be explored in Section 3.3.

3.2.2.4.2 Finite State Machine To ensure the proper functioning of the aforementioned modules, it is crucial to construct a Finite State Machine (FSM) that checks and drives various signals, thereby establishing the link of the JESD204B protocol with the ADC.

The inputs and outputs of this module are depicted in the subsequent figure:

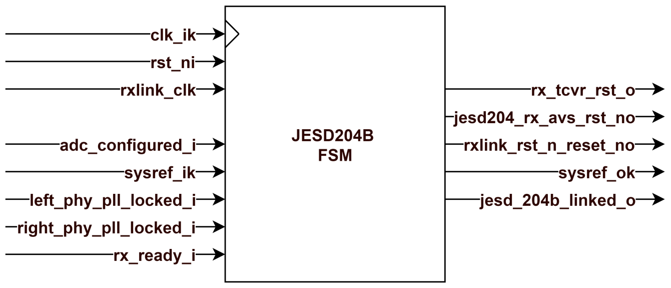


Figure 3.16: JESD204B link FSM module.

A concise description of the signals can be found in the list below:

- **clk_ik**: 125MHz application clock.
- **rst_ni**: Application reset.
- **rxlink_clk**: 125MHz link clock from the left PLL.
- **adc_configured_i**: Start signal for the FSM. Asserted after ADC SPI configuration.
- **sysref_ik**: Sysref signal synchronized with the link clock.
- **left_phy_pll_locked_i**: Asserted when the left PLL is locked to the reference 500MHz clock.
- **right_phy_pll_locked_i**: Asserted when the right PLL is locked to the reference 500MHz clock.
- **rx_ready_i**: Synchronized rx reset from the transceiver reset controller.
- **rx_tcvr_rst_o**: Transceiver reset signal.
- **jesd204_rx_avalon_rst_no**: Avalon reset.
- **rxlink_rst_n_reset_no**: Link reset.
- **sysref_ok**: Bypass for sysref_i controller by FSM logic.
- **jesd204b_linked_o**: Asserted when the FSM link has been established.

Figure 3.17 shows the diagram state that the FSM needs to guarantee to establish the serial communication link between the ADC and the FPGA at boot time. ”ADC-FPGA Subsystem Reset Sequence”, in the Intel JESD204B documentation [19].

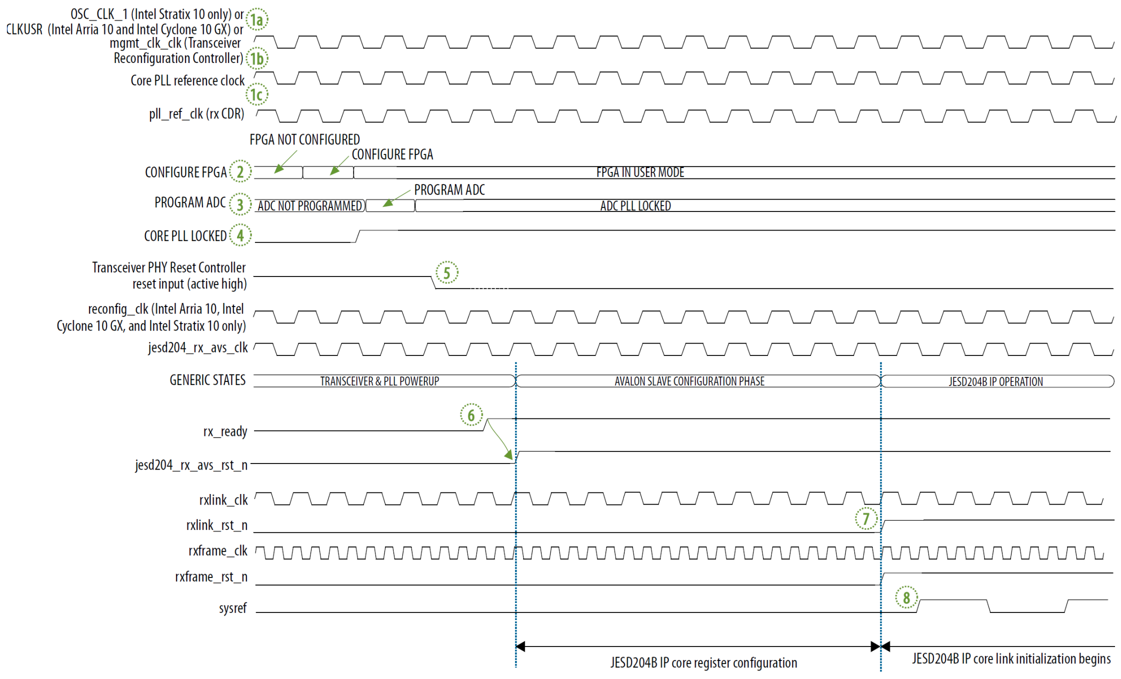
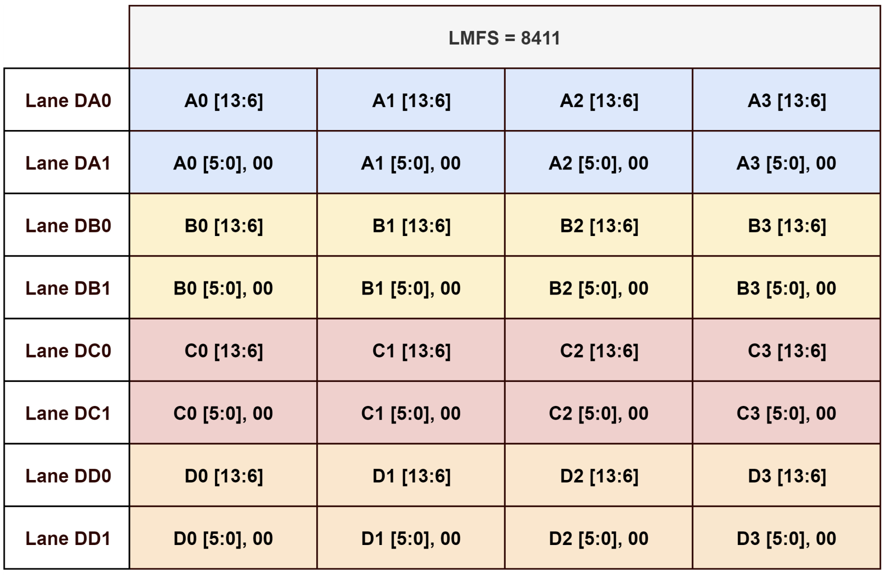

OSC_CLK_1 (Intel Stratix 10 only) or
CLKUSR (Intel Arria 10 and Intel Cyclone 10 GX) or
mgmt_clk_clk (Transceiver
Reconfiguration Controller)
1a
1b
Core PLL reference clock
1c
pll_ref_clk (rx CDR)
FPGA NOT CONFIGURED
CONFIGURE FPGA
CONFIGURE FPGA
2
FPGA IN USER MODE
PROGRAM ADC
PROGRAM ADC
3
ADC NOT PROGRAMMED
ADC PLL LOCKED
CORE PLL LOCKED
4
Transceiver PHY Reset Controller
reset input (active high)
5
reconfig_clk (Intel Arria 10, Intel
Cyclone 10 GX, and Intel Stratix 10 only)
jesd204_rx_avs_clk
GENERIC STATES
TRANSCEIVER & PLL POWERUP
AVALON SLAVE CONFIGURATION PHASE
JESD204B IP OPERATION
6
rx_ready
jesd204_rx_avs_rst_n
rxlink_clk
rxlink_rst_n
7
rxframe_clk
rxframe_rst_n
8
sysref
JESD204B IP core register configuration
JESD204B IP core link initialization begins

Figure 3.17: JESD204B link FSM.

1. Reference Clocks:
   (a) Calibration clock.
   (b) Management clock (500MHz).
   (c) Transceiver reference clock (125MHz).
   (d) Device clock (125MHz).
2. Configure the FPGA. Keep the RX transceiver channel in reset (*rx_tcvr_rst_o*).
3. Program the ADC via SPI. Wait for the *adc_configure_i* signal.
4. Ensure the PLLs (*left_phy_pll_locked_i* and *right_phy_pll_locked_i*) are locked.
5. Deassert the FPGA RX transceiver channel reset (*rx_tcvr_rst_o*).
6. Check if the transceiver is out of reset (*rx_ready* must be asserted), then deassert the Avalon reset (*jesd204_rx_avalon_rst_no*).
7. Deassert both the link reset (*rxlink_rst_n_reset_no*) and the frame reset.
8. Drive *sysref_ik* input to output (*sysref_ok*).

3.2.2.4.3 Transport Layer The JESD204B MAC module provides at its output a stream of 256 bits every 8s. This corresponds to four samples for each of the four converters present in the ADC IC. This data is intermixed, and might contain padding bits, therefore it must be reorganized to be able to post-process the samples of each channel independently. That is the function of the Transport Layer module.

Figure 3.18 provides a scheme of how the data is sent by the ADC when configured to work in the mode LMFS=8411. The letters A to D make reference to the ADC channels. Thus A0 is the most recent sample from channel 0, A1 is the previous one, and so on. The number within the brackets indicates the bits of each sample. Since we use 8 lanes for four converters, each converter uses two lanes, thus the lane names DA0-DA1, DB0-DB1, etc. As one can observe, the bits of one sample are sent simultaneously through different lanes.



	LMFS = 8411			
Lane DA0	A0 [13:6]	A1 [13:6]	A2 [13:6]	A3 [13:6]
Lane DA1	A0 [5:0], 00	A1 [5:0], 00	A2 [5:0], 00	A3 [5:0], 00
Lane DB0	B0 [13:6]	B1 [13:6]	B2 [13:6]	B3 [13:6]
Lane DB1	B0 [5:0], 00	B1 [5:0], 00	B2 [5:0], 00	B3 [5:0], 00
Lane DC0	C0 [13:6]	C1 [13:6]	C2 [13:6]	C3 [13:6]
Lane DC1	C0 [5:0], 00	C1 [5:0], 00	C2 [5:0], 00	C3 [5:0], 00
Lane DD0	D0 [13:6]	D1 [13:6]	D2 [13:6]	D3 [13:6]
Lane DD1	D0 [5:0], 00	D1 [5:0], 00	D2 [5:0], 00	D3 [5:0], 00

Figure 3.18: JESD204B ADC data formatting for lanes.

In addition, as observed earlier, some lanes are received through the left transceiver bank of the FPGA (left PHY) and others through the right bank (right PHY). After the PHY stage, data is concatenated and inserted into the MAC layer. The order of concatenation is first the data from the left lanes and then that of the right ones. Therefore, the order of the lanes is as follows:

1. Lane DB1.
2. Lane DA1.
3. Lane DA0.
4. Lane DB0.
5. Lane DD1.
6. Lane DD0.
7. Lane DC0.
8. Lane DC1.

Consequently, the 256-bit data array available at each link_clock cycle that needs to be processed by the transport layer adopts the following form:

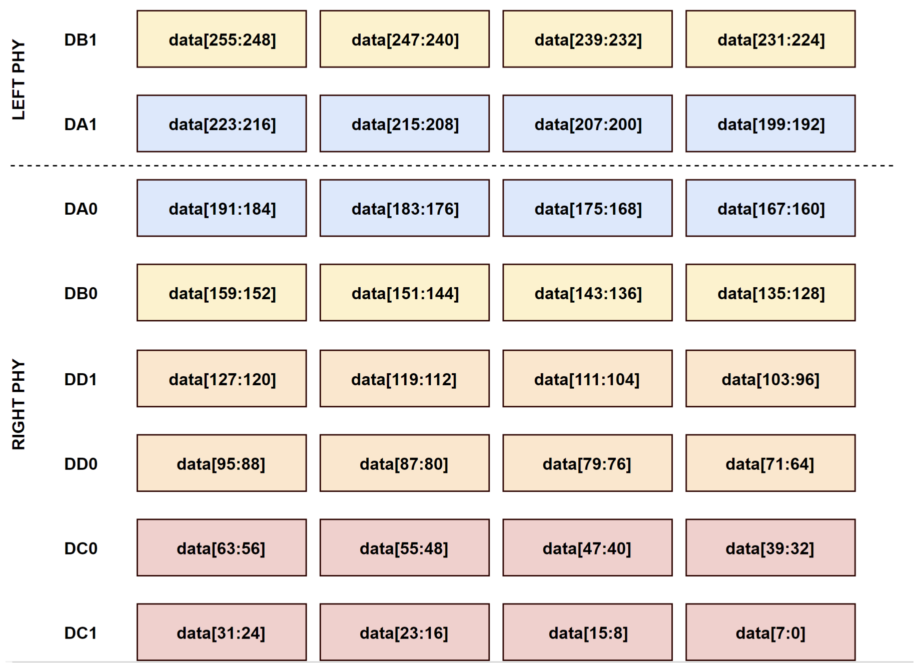


Figure 3.19: JESD204B data before transport layer decoding.

The *JESD204B_Transport* module takes this data at each link_clock cycle and, using combinational logic, transforms it into four arrays. Each array is associated with a channel and it contains 4 samples of 16 bits each. There must be taken into account that the last two LSB are the Control and Tail bits added during the transmission as shown in Figure 2.17.

Figure 3.20 illustrates how the data must be arranged to have the samples split by channels:

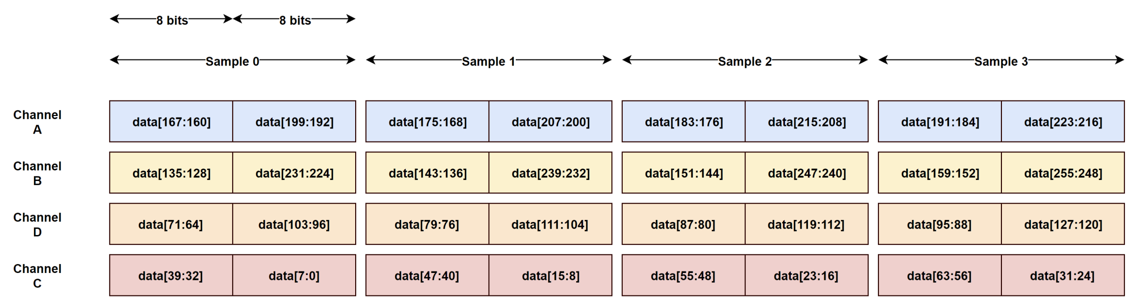


Figure 3.20: JESD204B data sorted by the transport layer.

3.2.3 RAM Controller

After data acquisition is accomplished, data must be saved into a RAM memory. In order to do this, a RAM_Controller module has been created (Figure 3.21). This module is at the same layer than *MezzanineController* module in the Application layer as mentioned before.

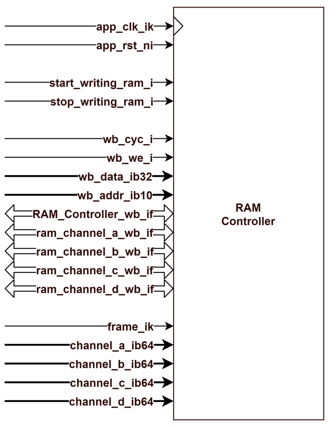


Figure 3.21: RAM Controller module.

Figure 3.22 shows the different submodules that are inside the *RAM_Controller* module:

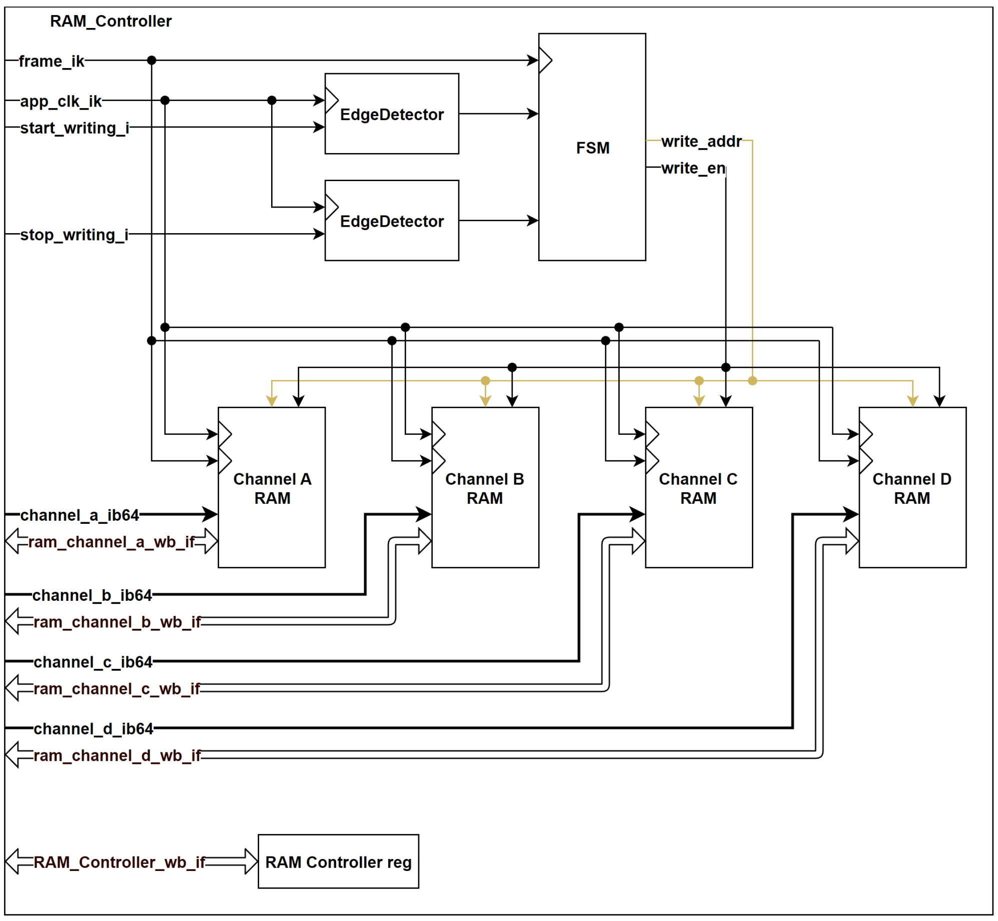


Figure 3.22: RAM Controller detailed view.

Module is divided into the finite state machine in charge of executing the circular buffer [2] and the *Channel_RAM* modules, which is the actual RAM that saves data from the channels.

In addition to the previous modules, it also instantiated a register bank called *RAM_Controller_reg* for monitoring and debugging purposes.

3.2.3.1 Channel RAM

The objective of this module (Figure 3.23) is to store data for a single channel of the ADC. As stated in Subsection 1.2.1, the goal of the project is to accumulate up to $160\mu s$ of data per channel (corresponding to the first 160 turns after the beam is injected into a particular ring).

This module receives a $64b$ input data, representing four samples of a channel, and stores them into a RAM IP core from Altera. Additionally, this module is directly linked to the Wishbone bus, enabling the user to read back the memory.

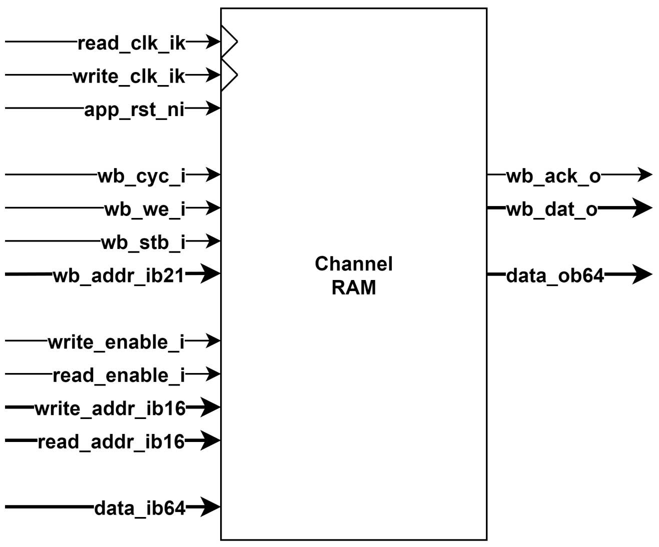


Figure 3.23: Channel RAM module.

The processes of writing and reading are two independent tasks, working on two different clock domains, thus the module necessitates two distinct clock signals. Data is written using the clock signal named, write_clk_ik, which is connected to the frame_clk from the JESD204B module (and thus recovered from the CDR circuit from the ADC data itself). And it is read, by means of the read_clk_ik, which is synchronized with the Wishbone bus clock signal. While both clock signals operate at 125MHz, they are not necessarily in phase, and special attention should be paid to ensure a synchronous design.

3.2.3.1.1 Space Computation Considering the sampling frequency ($500MSPS$), the total number of samples that need to be stored in each RAM can be computed as:

[2]Circular buffer is based on an array with a counter. Once the counter reaches the maximum index of the array it resets the current counter value to 0 and starts from the beginning.

$$500 \cdot 10^6 \times 160 \cdot 10^{-6} = 80000 \tag{3.1}$$

This establishes the RAM capacity for each channel as:

$$80000 \times 14b = 1120000b = 1.12\ Mb \tag{3.2}$$

Given that RAM sizes are typically a power of two and the nearest power of two is sixteen, the computation will be performed with this value:

$$80000 \times 16b = 1280000b = 1.28\ Mb \tag{3.3}$$

This will be the total amount of RAM necessary for each channel, so the total amount of RAM for the entire system will be:

$$4 \times 1280000b = 5120000 = 5.12\ Mb \tag{3.4}$$

On the other hand, each frame_clk cycle provides us with 4 samples at a time. To simplify the logic of saving, it would be preferable to save 4 samples at each time.

Since each frame_clk cycle provides a data stream with 4 samples, the most practical design strategy is to use a RAM that is $64b$ wide. This will make unnecessary the generation of a clock four times faster, or splitting the memory in additional blocks with the consequent logic addressing complication. With this width, the total number of addresses needed to reach $1.28Mb$ would be:

$$\frac{1.28Mb}{64b} = 20000 \tag{3.5}$$

And hence, the number of bits required for the address is:

$$\log_2 20000 = 14.287\ b \simeq 15\ b \tag{3.6}$$

3.2.3.1.2 Architecture Figure 3.24 illustrates how the RAM is instantiated within the *Channel_RAM* module:

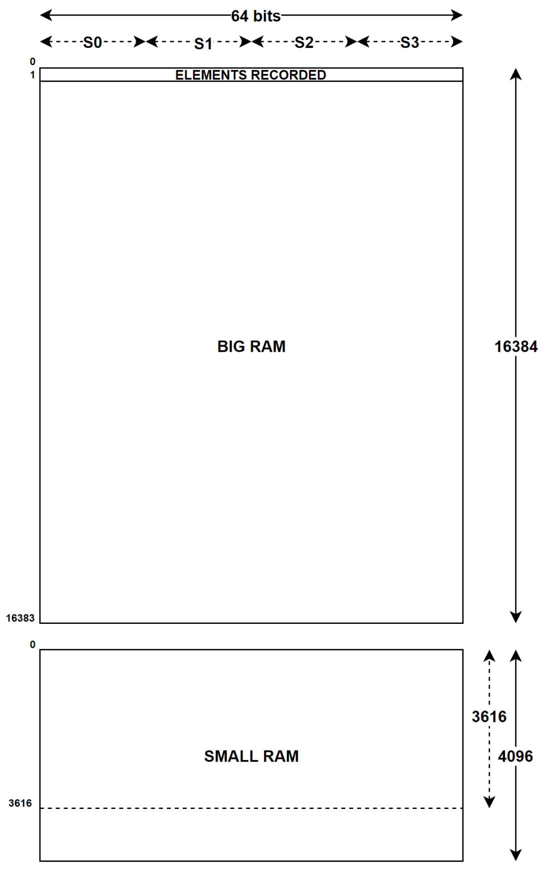


Figure 3.24: Channel RAM architecture.

As depicted in the previous figure, the RAM is internally partitioned into two distinct modules. This is due to the fact that the Altera RAM IP core requires the instantiation of a RAM of size that is a power of two. In our case, we need 20000 memory slots per channel, but this number is not a power of two (hence the need for $14.287b$ to address the entire system).

Instantiating a RAM memory that uses an address bus of $15b$ would represent 32768 slots of RAM, consuming unnecessary internal FPGA logic:

$$\frac{20000}{32768} = 0.61 \simeq 61\% \tag{3.7}$$

For this reason, it was decided to use a design strategy that optimises the resources used, based on instantiating two different RAMs. The first RAM, referred to as *"BIG_RAM"*, uses a $14b$ wide address bus and has 16384 available slots. Conversely, the *"SMALL_RAM"* is a $12b$ wide address bus RAM with 4096 available slots.

By implementing this system, we will have 20480 memory slots available, thereby reducing the inefficiency to:

$$\frac{20000}{20480} = 0.976 \simeq 97.6\% \tag{3.8}$$

This way, only 480 slots (7680 b) will not be used. This is represented by a pointed line in 3.24

3.2.3.1.3 Addressing

The heart of this module lies in the addressing logic. Given that the RAM is divided, it is necessary to implement some combinational logic to determine which RAM should be read or written to, based on the address.

Let's initially focus on the writing process. As previously mentioned, we need $15b$ to count from 0 to 200000. The following table represents the binary representation of each number:

Decimal	Binary (15b)
0	000 0000 0000 0000
16383	011 1111 1111 1111
16384	100 0000 0000 0000
20000	100 1110 0010 0000

As can be seen, we only need $14b$ to count from 0 to 16383. Consequently, we can use the first $14b$ for addressing the "BIG_RAM", and the 15th bit (MSB) as an enable signal for selecting the target RAM for writing. When this bit is set to 0, it indicates that we should write into the *"BIG_RAM"*, and when it is set to 1, it suggests that we should write into the *"SMALL_RAM"*. Figure 3.25 illustrates the address bits for a writing cycle:

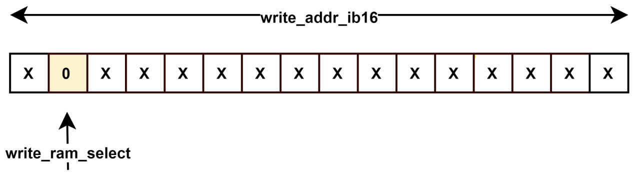


Figure 3.25: Channel RAM write address.

The write_ram_select is the MSB and serves as the control bit of a decoder, as shown in Figure 3.26.

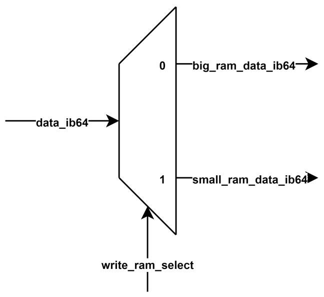


Figure 3.26: Channel RAM write select decoder.

Conversely, a read cycle is slightly more complex. This complexity arises because the wishbone bus, which is used for reading, is $32b$ wide, whereas the RAM output is $64b$ wide. This time, we need to use a bit to select the source RAM and also split the RAM output into two $32b$ slots, using a bit to select the desired slot for reading.

Figure 3.27 demonstrates how the bits of a reading address are employed for:

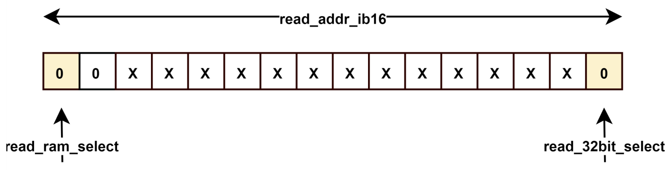
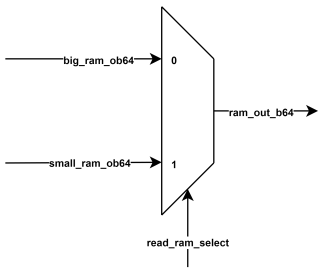
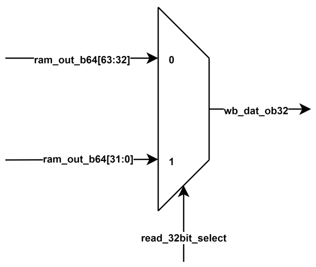


Figure 3.27: Channel RAM read address.

Using these two bits, we can refer to Figure 3.28a and Figure 3.28b to comprehend the operation of the selectors:



(a) Channel RAM read select multiplexer.



(b) Channel RAM wishbone read multiplexer.

When read_32bit_select is set to 0, the first $32b$ (Sample 0 and 1) of the selected RAM (depending on read_ram_select) will be available on the wishbone bus. When this bit is set to 1, the second $32b$ (Sample 2 and 3) of the RAM will be available on the Wishbone bus.

3.2.3.2 Finite State Machine

Lastly, to manage the start and stop conditions of the system, and therefore determine when the RAM should or should not save data, a Finite State Machine (FSM) is instantiated within the *RAM_Controller* module.

Simultaneously, this FSM is split into two distinct states. The first state is the *"FREEZED"* state, which is the base state the FSM maintains until an "unfreeze" request is received from the software layer. While in this state, the RAM will never capture the trigger and thus will not save data. After each write cycle, the FSM transitions back to this state.

Conversely, there is the *"UNFREEZED"* state, which is the standard operational state during a write cycle. While in this state, a single start/stop trigger can be handled in order to write data into the RAM. This state operates its own FSM as shown below.

The FSM diagram can be found in Figure 3.29. The finite state machine consists of three states.

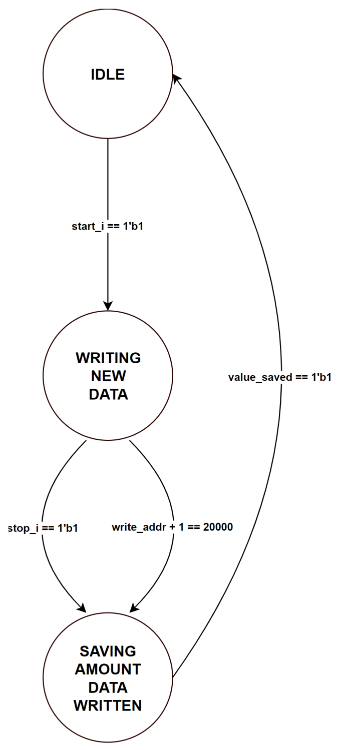


Figure 3.29: RAM Controller finite state machine.

The first state, *"IDLE"*, involves the FSM awaiting a "start_i" trigger from the timing board (CTRV) without performing any actions.

Upon receiving this signal, the FSM transitions to the next state, termed *"WRITING_NEW_DATA"*. In this state, the FSM begins to count each write_clk_ik cycle and increments a variable, which is fed to the RAM as the write address. To exit this state, one of two conditions needs to be met: either the assertion of the "stop_i" signal, or the counter reaching its maximum value (which corresponds to $160\mu s$ of written data).

The final state is the *"SAVING_AMOUNT_DATA_WRITTEN"*. This state is designed to provide system flexibility during data writing. Given that the system requirements dictate the ability to write up to $160\mu s$ of data, there may be instances where this time period is not fully utilized, resulting in the system saving a different time value, for example $120\mu s$. This state is used to store the counter value (which represents the saved time) into the RAM memory at address $0x0$. Consequently, the user can read the address $0x0$ of the RAM before reading the entire memory, or they can read all the memory for simplicity but only use the number of elements saved at this address. Figure 3.24 illustrates how this value is saved into the RAM.

3.2.4 Trigger Controller

The final feature that needs to be incorporated into the system is the capacity to introduce a configurable delay in the trigger system. This functionality has been realized within the *Trigger_Controller* module, as shown in Figure 3.30.

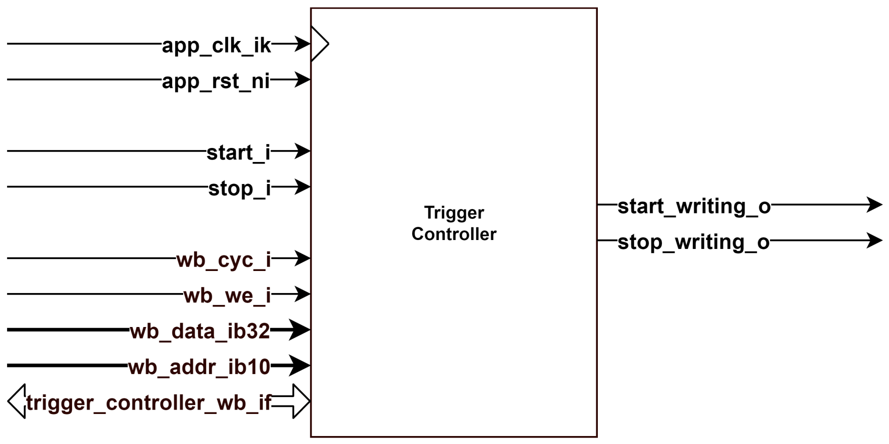


Figure 3.30: Trigger controller module.

The aim of this module is to take two inputs from the front part of the VFC board (LEMO connectors), apply a configurable delay and feed the RAM_Controller module in order to start saving data from the ADC.

A more detailed view of the module can be shown at Figure 3.31 :

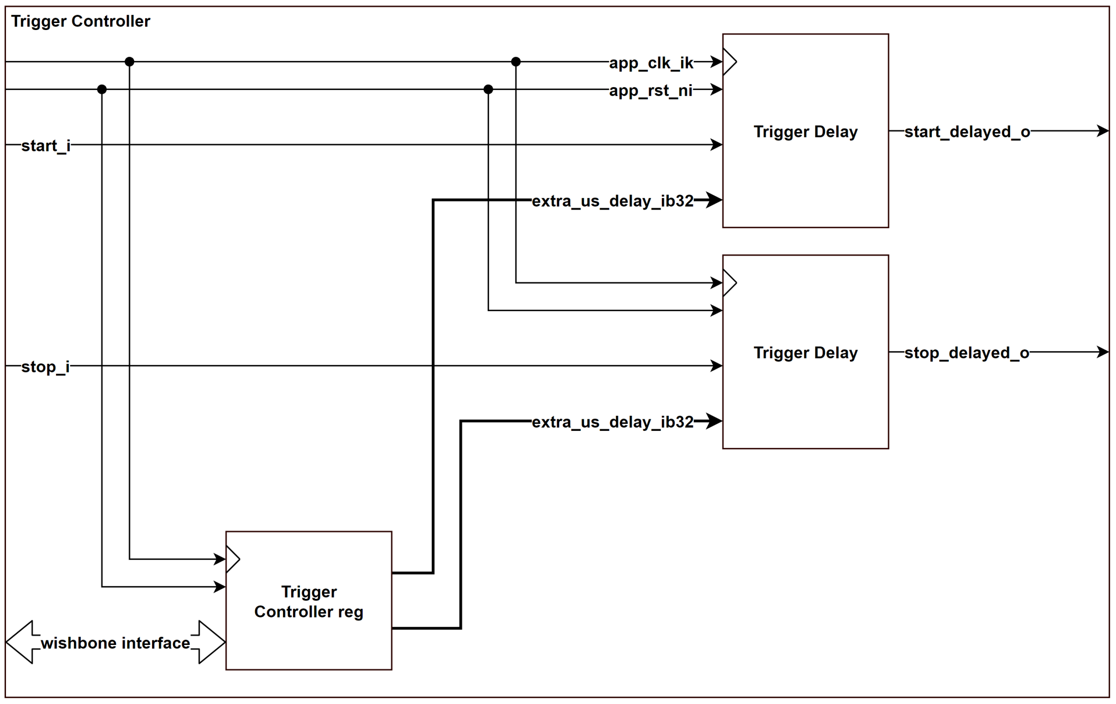


Figure 3.31: Trigger controller module in detail.

The architecture of this module is very simple. It uses the *Trigger_Delay* module in order to apply a delay to the input signal. This module embeds a counter that counts up to a given number which can be configured through the *Trigger_Controller_reg* register connected to the Wishbone bus.

3.3 Verification

The final segment of this thesis will center on verifying the design detailed previously. In the current landscape of engineering, verification stands as one of the most crucial facets of any project. It facilitates a thorough comprehension of the system's functionality and validates the system's performance under designated conditions.

In order to verify this data acquisition system, a strictly functional verification strategy has been utilized. This strategy incorporates the development of SignalTap modules (SignalTap is an Intel application that allows for real-time monitoring of various signals within the design), Python scripts, and simulations. These tools are employed either to cross-check the results or to accelerate the development of specific modules.

3.3.1 Mezzanine Controller

As previously mentioned, the *MezzanineController* module is a complex entity comprised of various submodules. To verify this module, the problem has been dissected into distinct tasks. Addressing these tasks individually ensures the correct functionality of the entire module.

The tasks are as follows:

1. Verify that the output voltage of the FMC is 1.8V.
2. Confirm that the ADS54J54 FSM generates the correct signals.
3. Ensure that the SPI Controller transmits the correct commands via SPI.
4. Validate the link process of the JESD204B protocol through the JESD204B Controller.
5. Verify the Transport layer.

In this thesis, we will not delve into all the verification steps as some of them are simpler than others. Therefore, we will focus on those tasks whose complexity warrants a detailed explanation.

3.3.1.1 SPI Controller

The first substantial module that required thorough debugging and verification was the *SPI_Controller* module. As delineated in Subsubsection 3.2.2.3, this module can send data through SPI using a custom module, which maintains the entire configuration for the ADC, and by utilizing a wishbone-to-SPI bridge.

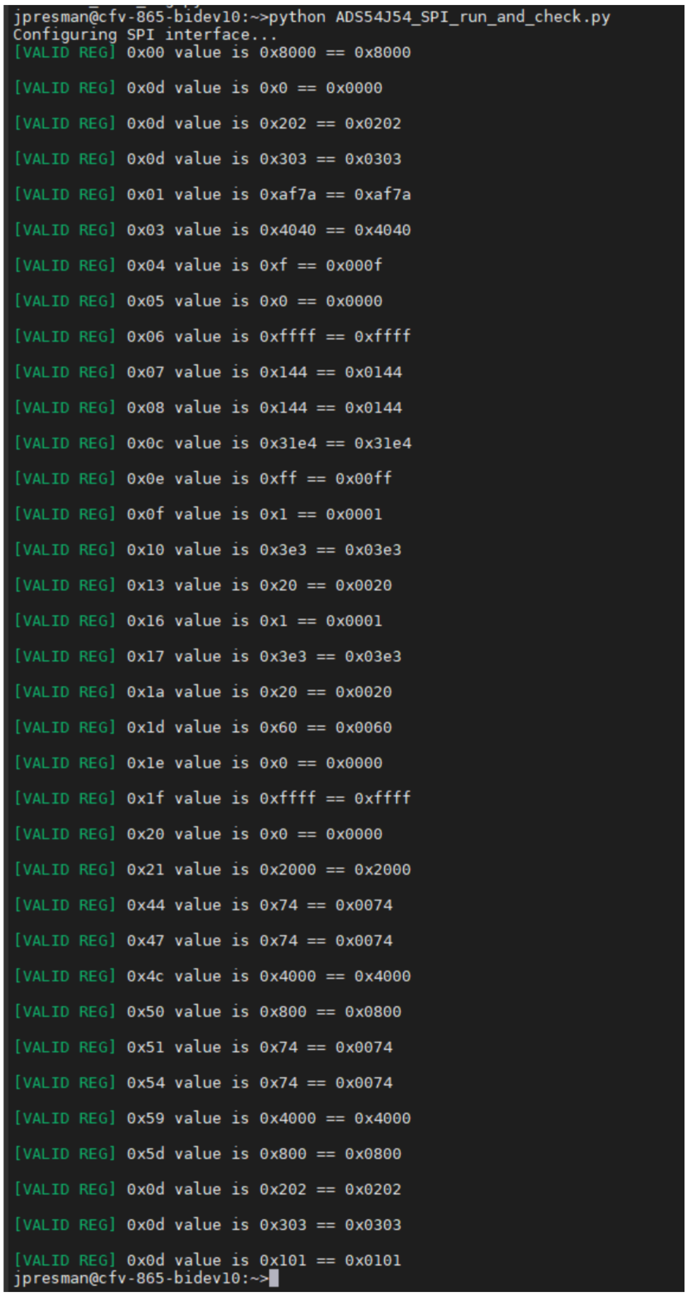
To verify the ADC's correct configuration, a list of commands was dispatched using a Python script.

This Python script accessed the VME bus, which is in turn connected to the Wishbone Address Decoder (Subsection 3.2.1), where the *SPI_Controller* is connected via an interface. By leveraging this VME bus, the Python script was capable of sending the correct commands to the ADC and then reading back data from it by interchanging writing commands and reading requests.

This approach tested both the wishbone connection and the *SPI_Listener* module, which is responsible for intercepting the data from the wishbone and storing it in a register connected to the wishbone.

Once the write command was dispatched and the read operation completed, the Python script could compare the actual value of the ADC register with the written value. If the values did not match, it signaled an error, indicating that the write operation was not successful.

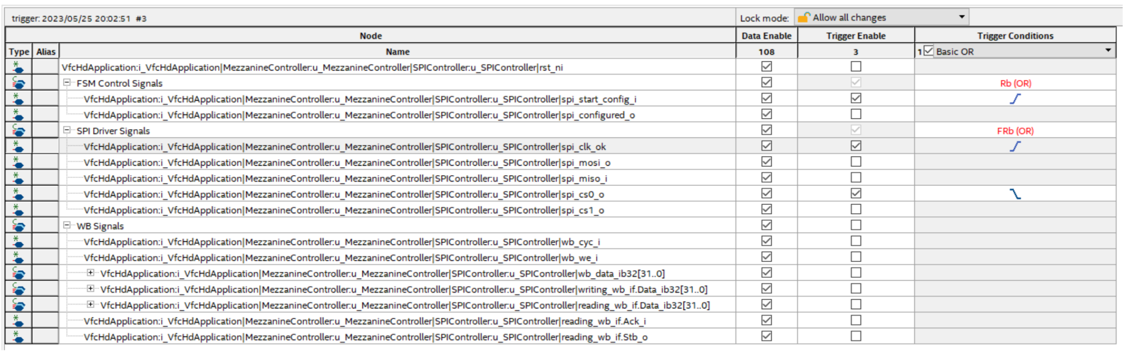
Figure 3.32 displays the output of the Python script when all the registers have been correctly written.



```
jpresman@cfv-865-bidev10:~>python ADS54J54_SPI_run_and_check.py
Configuring SPI interface...
[VALID REG] 0x00 value is 0x8000 == 0x8000

[VALID REG] 0x0d value is 0x0 == 0x0000

[VALID REG] 0x0d value is 0x202 == 0x0202

[VALID REG] 0x0d value is 0x303 == 0x0303

[VALID REG] 0x01 value is 0xaf7a == 0xaf7a

[VALID REG] 0x03 value is 0x4040 == 0x4040

[VALID REG] 0x04 value is 0xf == 0x000f

[VALID REG] 0x05 value is 0x0 == 0x0000

[VALID REG] 0x06 value is 0xffff == 0xffff

[VALID REG] 0x07 value is 0x144 == 0x0144

[VALID REG] 0x08 value is 0x144 == 0x0144

[VALID REG] 0x0c value is 0x31e4 == 0x31e4

[VALID REG] 0x0e value is 0xff == 0x00ff

[VALID REG] 0x0f value is 0x1 == 0x0001

[VALID REG] 0x10 value is 0x3e3 == 0x03e3

[VALID REG] 0x13 value is 0x20 == 0x0020

[VALID REG] 0x16 value is 0x1 == 0x0001

[VALID REG] 0x17 value is 0x3e3 == 0x03e3

[VALID REG] 0x1a value is 0x20 == 0x0020

[VALID REG] 0x1d value is 0x60 == 0x0060

[VALID REG] 0x1e value is 0x0 == 0x0000

[VALID REG] 0x1f value is 0xffff == 0xffff

[VALID REG] 0x20 value is 0x0 == 0x0000

[VALID REG] 0x21 value is 0x2000 == 0x2000

[VALID REG] 0x44 value is 0x74 == 0x0074

[VALID REG] 0x47 value is 0x74 == 0x0074

[VALID REG] 0x4c value is 0x4000 == 0x4000

[VALID REG] 0x50 value is 0x800 == 0x0800

[VALID REG] 0x51 value is 0x74 == 0x0074

[VALID REG] 0x54 value is 0x74 == 0x0074

[VALID REG] 0x59 value is 0x4000 == 0x4000

[VALID REG] 0x5d value is 0x800 == 0x0800

[VALID REG] 0x0d value is 0x202 == 0x0202

[VALID REG] 0x0d value is 0x303 == 0x0303

[VALID REG] 0x0d value is 0x101 == 0x0101
jpresman@cfv-865-bidev10:~>
```

Figure 3.32: Python script for writing and checking ADC register content.

Once checked that the configuration is correct, we used a SignalTap module (Figure 3.33) in order to verify that the custom module for writing these commands automatically was working correctly.



trigger: 2023/05/25 20:02:51 #3 — Lock mode: Allow all changes

Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
			108	3	1 ☑ Basic OR
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|rst_ni	☑	☐	
		⊟ FSM Control Signals	☑	☑	Rb (OR)
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|spi_start_config_i	☑	☑	↗
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|spi_configured_o	☑	☐	
		⊟ SPI Driver Signals	☑	☑	FRb (OR)
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|spi_clk_ok	☑	☑	↗
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|spi_mosi_o	☑	☐	
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|spi_miso_i	☑	☐	
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|spi_cs0_o	☑	☑	↘
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|spi_cs1_o	☑	☐	
		⊟ WB Signals	☑	☐	
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|wb_cyc_i	☑	☐	
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|wb_we_i	☑	☐	
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|wb_data_ib32[31..0]	☑	☐	
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|writing_wb_if.Data_ib32[31..0]	☑	☐	
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|reading_wb_if.Data_ib32[31..0]	☑	☐	
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|reading_wb_if.Ack_i	☑	☐	
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|SPIController:u_SPIController|reading_wb_if.Stb_o	☑	☐	

Figure 3.33: Signal Tap configuration for SPI Controller.

This SignalTap module is divided into three sections:

1. FSM Control Signals: start/stop signals originating from the FSM.
2. SPI Driver signals: signals from the SPI bus.
3. Wishbone Signals: signals for manual reading and writing via Wishbone.

To initiate data recording, a trigger is applied on the positive edge of the ”spi_start_config_i” signal. This signal, derived from the main FSM, indicates when the ADC can be configured.

The output of this module is illustrated in Figure 3.34:

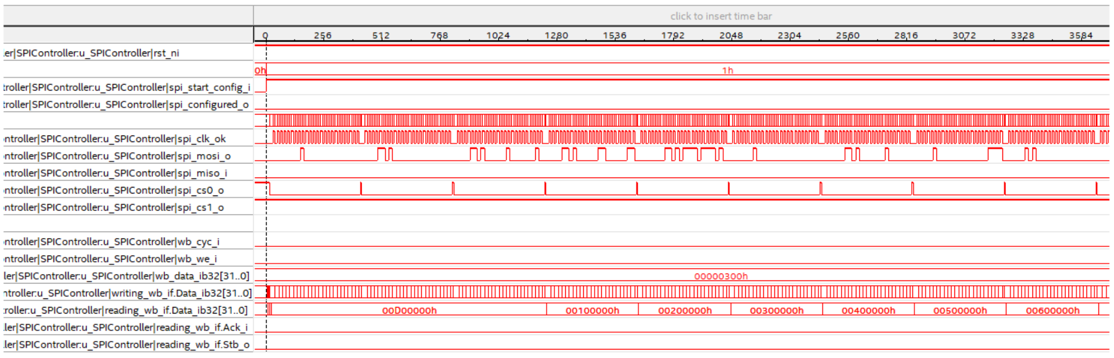

click to insert time bar
0
256
512
768
1024
1280
1536
1792
2048
2304
2560
2816
3072
3328
3584
er|SPIController:u_SPIController|rst_ni
0h
1h
troller|SPIController:u_SPIController|spi_start_config_i
troller|SPIController:u_SPIController|spi_configured_o
ntroller|SPIController:u_SPIController|spi_clk_ok
ntroller|SPIController:u_SPIController|spi_mosi_o
ntroller|SPIController:u_SPIController|spi_miso_i
ntroller|SPIController:u_SPIController|spi_cs0_o
ntroller|SPIController:u_SPIController|spi_cs1_o
ntroller|SPIController:u_SPIController|wb_cyc_i
ntroller|SPIController:u_SPIController|wb_we_i
ler|SPIController:u_SPIController|wb_data_ib32[31..0]
00000300h
ntroller:u_SPIController|writing_wb_if.Data_ib32[31..0]
roller:u_SPIController|reading_wb_if.Data_ib32[31..0]
00D00000h
00100000h
00200000h
00300000h
00400000h
00500000h
00600000h
ler|SPIController:u_SPIController|reading_wb_if.Ack_i
ler|SPIController:u_SPIController|reading_wb_if.Stb_o

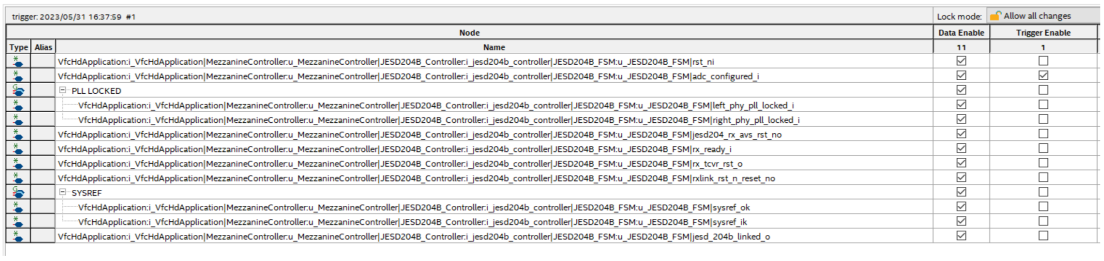
Figure 3.34: Signal Tap output for SPI Controller.

Here we can observe that when the trigger is asserted, the system commences data recording. The crucial part for this validation is the ”mosi_o” signal, where we can clearly see data at the SPI output. This signal is generated through the Wishbone-to-SPI module, so this means that the module which is in charge of automatically generating this wishbone sequence is working properly. Remember that this module is connected to a wishbone arbiter as shown in Figure 3.10

3.3.1.2 JESD204B Controller

The second and most crucial module to be scrutinized is the *JESD204B_Controller*. Validating this module has been challenging because of its enormous number of signals, multiple clock domains, and the intricate system represented by this protocol.

3.3.1.2.1 Finite State Machine On the one hand, two distinct SignalTap modules have been developed. The first one, as depicted in Figure 3.35, is utilized for verifying the JESD204B FSM.



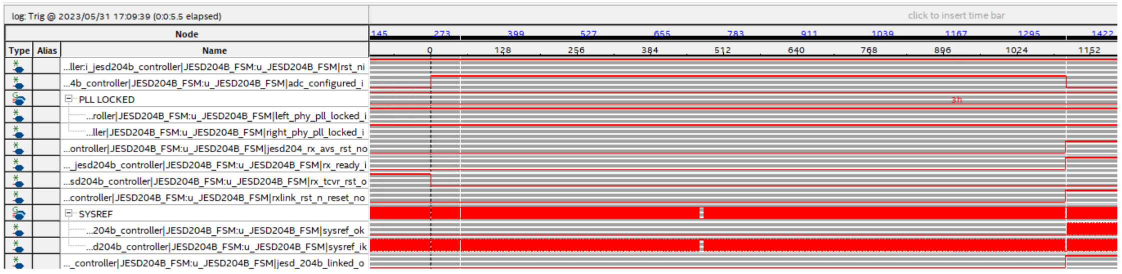
trigger: 2023/05/31 16:37:59 #1 | Lock mode: Allow all changes

Type	Alias	Name	Data Enable 11	Trigger Enable 1
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|rst_ni	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|adc_configured_i	☑	☑
		⊟ PLL LOCKED	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|left_phy_pll_locked_i	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|right_phy_pll_locked_i	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|jesd204_rx_avs_rst_no	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|rx_ready_i	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|rx_tcvr_rst_o	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|rxlink_rst_n_reset_no	☑	☐
		⊟ SYSREF	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|sysref_ok	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|sysref_ik	☑	☐
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|jesd_204b_linked_o	☑	☐

Figure 3.35: Signal Tap configuration for JESD204B FSM.

This SignalTap was established to facilitate inspection of the aforementioned FSM (Figure 3.17), thereby improving our understanding of which signals emanating from various Altera modules are malfunctioning and impeding the Link initialization.

The output from this SignalTap is demonstrated in Figure 3.36:



log: Trig @ 2023/05/31 17:09:39 (0:0:5.5 elapsed)

Type	Alias	Name
		...ller:i_jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|rst_ni
		...4b_controller|JESD204B_FSM:u_JESD204B_FSM|adc_configured_i
		⊟ PLL LOCKED
		...roller|JESD204B_FSM:u_JESD204B_FSM|left_phy_pll_locked_i
		...ller|JESD204B_FSM:u_JESD204B_FSM|right_phy_pll_locked_i
		...ontroller|JESD204B_FSM:u_JESD204B_FSM|jesd204_rx_avs_rst_no
		..._jesd204b_controller|JESD204B_FSM:u_JESD204B_FSM|rx_ready_i
		...sd204b_controller|JESD204B_FSM:u_JESD204B_FSM|rx_tcvr_rst_o
		...controller|JESD204B_FSM:u_JESD204B_FSM|rxlink_rst_n_reset_no
		⊟ SYSREF
		...204b_controller|JESD204B_FSM:u_JESD204B_FSM|sysref_ok
		...d204b_controller|JESD204B_FSM:u_JESD204B_FSM|sysref_ik
		..._controller|JESD204B_FSM:u_JESD204B_FSM|jesd_204b_linked_o

Figure 3.36: Signal Tap output for JESD204B FSM.

This allows for comparison between Intel's FSM documentation and the system's actual output, thereby enabling the verification of correct behavior.

3.3.1.2.2 Link Layer The subsequent step involves verifying the Link process. This process hinges on the execution of the preceding FSM, and the outcome is detailed in the Intel Documentation. Figure 3.37 illustrates the temporal progression of signals necessary to secure the link.

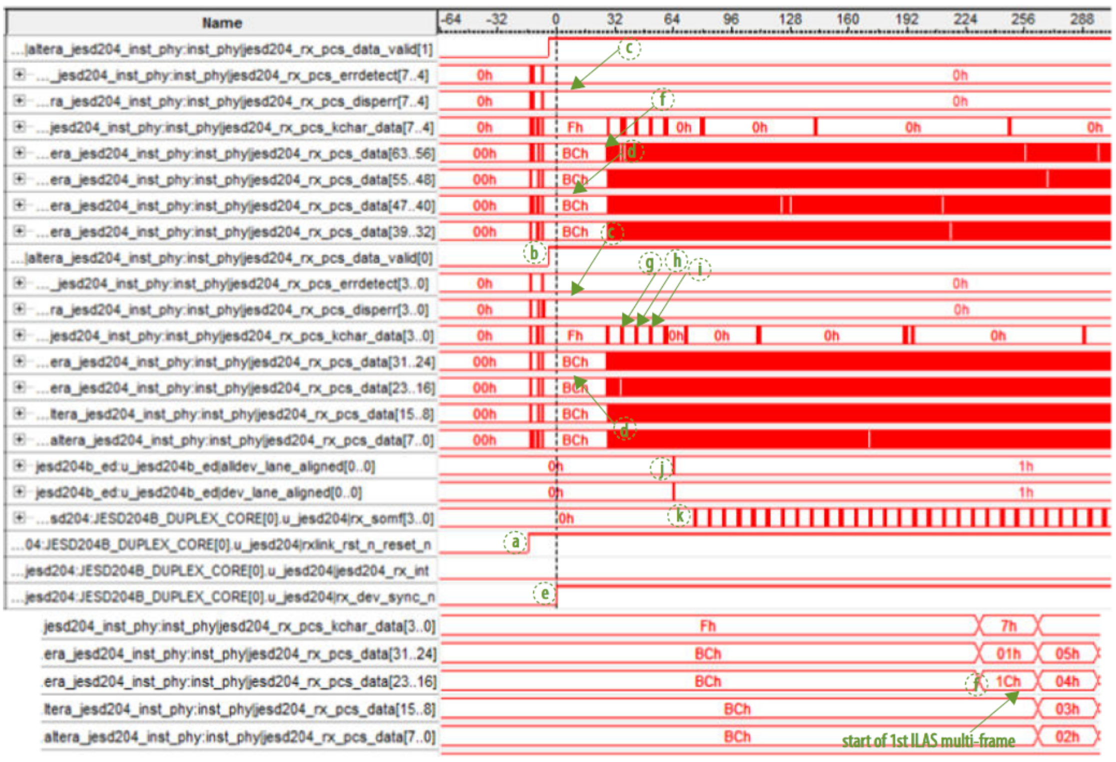

Name
...altera_jesd204_inst_phy:inst_phy|jesd204_rx_pcs_data_valid[1]
..._jesd204_inst_phy:inst_phy|jesd204_rx_pcs_errdetect[7..4]
...ra_jesd204_inst_phy:inst_phy|jesd204_rx_pcs_disperr[7..4]
...jesd204_inst_phy:inst_phy|jesd204_rx_pcs_kchar_data[7..4]
...era_jesd204_inst_phy:inst_phy|jesd204_rx_pcs_data[63..56]
...altera_jesd204_inst_phy:inst_phy|jesd204_rx_pcs_data_valid[0]
jesd204b_ed:u_jesd204b_ed|alldev_lane_aligned[0..0]
jesd204b_ed:u_jesd204b_ed|dev_lane_aligned[0..0]
...sd204:JESD204B_DUPLEX_CORE[0].u_jesd204|rx_somf[3..0]
...04:JESD204B_DUPLEX_CORE[0].u_jesd204|rxlink_rst_n_reset_n
...jesd204:JESD204B_DUPLEX_CORE[0].u_jesd204|jesd204_rx_int
...jesd204:JESD204B_DUPLEX_CORE[0].u_jesd204|rx_dev_sync_n
0h
Fh
BCh
00h
1h
7h
01h
05h
1Ch
04h
03h
02h
start of 1st ILAS multi-frame
a
b
c
d
e
f
g
h
i
j
k

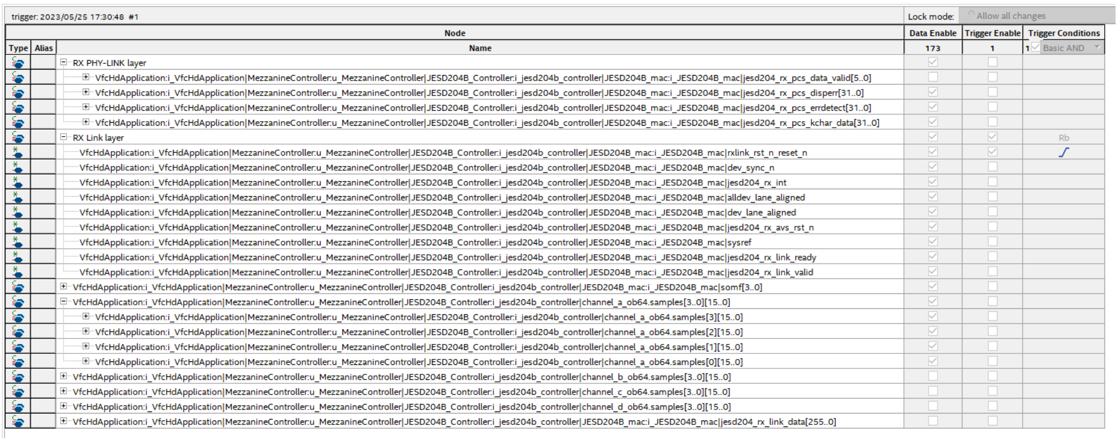
Figure 3.37: Intel's documentation for JESD204B Link process.

In this documentation, we can also find a small explanation of the process :

- a. The JESD204B link is out of reset.
- b. The RX CDR is locked and PCS outputs valid characters to link layer.
- c. No running disparity error and 8b/10b block within PCS successfully decodes the incoming characters.
- d. The ADC transmits /K/ character or BC hexadecimal number to the FPGA, which starts the CGS phase.
- e. Upon receiving 4 consecutive /K/ characters, the link layer deasserts the rx_dev_sync_n signal.
- f. The JESD204B link transition from CGS to ILAS phase when ADC transmit /R/ or 1C hexadecimal afteR /K/ character.
- g. Start of 2nd multi-frame in ILAS phase. 2nd multi-frame contains the JESD204B link configuration data.
- h. Start of 3rd multi-frame.
- i. Start of 4th multi-frame.

- j. Device lanes alignment is achieved. In this example, there is only one device, the dev_lane_aligned connects to alldev_lane_aligned and both signals are asserted together.
- k. Start of user data phase where user data is streamed through the JESD204B link

In this thesis, we have relied on this documentation as a reference for debugging custom modules. The following figure presents the SignalTap module crafted to verify the aforementioned signals:

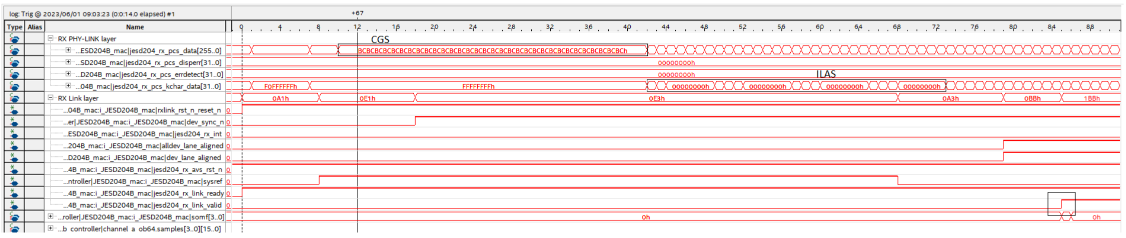


trigger: 2023/05/25 17:30:48 #1 | Lock mode: Allow all changes

Type	Alias	Node Name	Data Enable (173)	Trigger Enable (1)	Trigger Conditions (1 ✓ Basic AND)
		⊟ RX PHY-LINK layer	✓		
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|jesd204_rx_pcs_data_valid[5..0]			
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|jesd204_rx_pcs_disperr[31..0]	✓		
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|jesd204_rx_pcs_errdetect[31..0]	✓		
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|jesd204_rx_pcs_kchar_data[31..0]	✓		
		⊟ RX Link layer	✓	✓	Rb
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|rxlink_rst_n_reset_n	✓	✓	⎍
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|dev_sync_n	✓		
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|jesd204_rx_int	✓		
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|alldev_lane_aligned	✓		
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|dev_lane_aligned	✓		
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|jesd204_rx_avs_rst_n	✓		
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|sysref	✓		
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|jesd204_rx_link_ready	✓		
		VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|jesd204_rx_link_valid	✓		
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|somf[3..0]	✓		
		⊟ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|channel_a_ob64.samples[3..0][15..0]	✓		
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|channel_a_ob64.samples[3][15..0]	✓		
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|channel_a_ob64.samples[2][15..0]	✓		
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|channel_a_ob64.samples[1][15..0]	✓		
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|channel_a_ob64.samples[0][15..0]	✓		
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|channel_b_ob64.samples[3..0][15..0]			
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|channel_c_ob64.samples[3..0][15..0]			
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|channel_d_ob64.samples[3..0][15..0]			
		⊞ VfcHdApplication:i_VfcHdApplication|MezzanineController:u_MezzanineController|JESD204B_Controller:i_jesd204b_controller|JESD204B_mac:i_JESD204B_mac|jesd204_rx_link_data[255..0]			

Figure 3.38: SignalTap configuration for JESD204B Link layer.

The output for the previous SignalTap module is shown in Figure 3.39 :



log: Trig @ 2023/06/01 09:03:23 (0:0:14.0 elapsed) #1

Type	Alias	Name	
		⊟ RX PHY-LINK layer	
		⊞ ...ESD204B_mac|jesd204_rx_pcs_data[255..0]	
		⊞ ...SD204B_mac|jesd204_rx_pcs_disperr[31..0]	
		⊞ ...D204B_mac|jesd204_rx_pcs_errdetect[31..0]	
		⊞ ...04B_mac|jesd204_rx_pcs_kchar_data[31..0]	
		⊟ RX Link layer	
		...04B_mac:i_JESD204B_mac|rxlink_rst_n_reset_n	0
		...er|JESD204B_mac:i_JESD204B_mac|dev_sync_n	0
		...ESD204B_mac:i_JESD204B_mac|jesd204_rx_int	0
		...204B_mac:i_JESD204B_mac|alldev_lane_aligned	0
		...D204B_mac:i_JESD204B_mac|dev_lane_aligned	0
		...4B_mac:i_JESD204B_mac|jesd204_rx_avs_rst_n	0
		...ntroller|JESD204B_mac:i_JESD204B_mac|sysref	0
		...4B_mac:i_JESD204B_mac|jesd204_rx_link_ready	0
		...4B_mac:i_JESD204B_mac|jesd204_rx_link_valid	0
		⊞ ...roller|JESD204B_mac:i_JESD204B_mac|somf[3..0]	
		⊞ ...b controller|channel a ob64.samples[3..0][15..0]	

Figure 3.39: SignalTap output for JESD204B Link layer - Phases.

If we zoom out it is also possible to see a better overview of the process as shown in the next Figure:

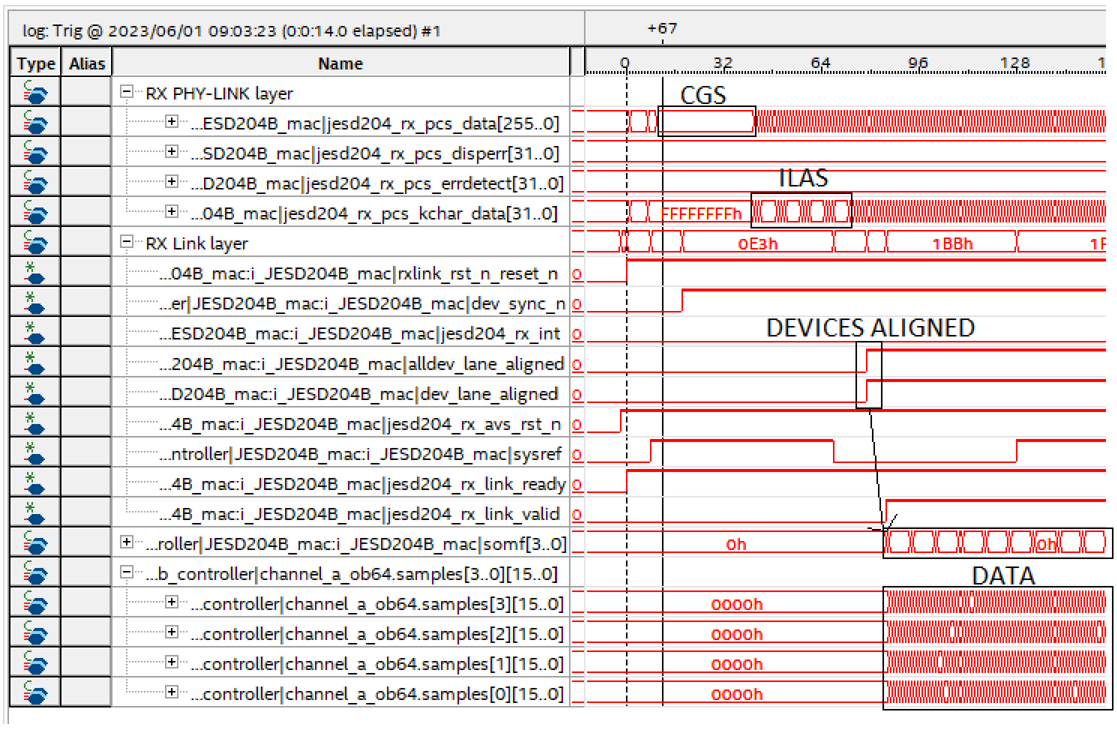
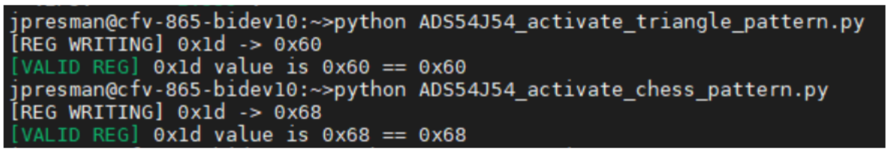

log: Trig @ 2023/06/01 09:03:23 (0:0:14.0 elapsed) #1
+67
Type
Alias
Name
0
32
64
96
128
RX PHY-LINK layer
...ESD204B_mac|jesd204_rx_pcs_data[255..0]
...SD204B_mac|jesd204_rx_pcs_disperr[31..0]
...D204B_mac|jesd204_rx_pcs_errdetect[31..0]
...04B_mac|jesd204_rx_pcs_kchar_data[31..0]
RX Link layer
...04B_mac:i_JESD204B_mac|rxlink_rst_n_reset_n
...er|JESD204B_mac:i_JESD204B_mac|dev_sync_n
...ESD204B_mac:i_JESD204B_mac|jesd204_rx_int
...204B_mac:i_JESD204B_mac|alldev_lane_aligned
...D204B_mac:i_JESD204B_mac|dev_lane_aligned
...4B_mac:i_JESD204B_mac|jesd204_rx_avs_rst_n
...ntroller|JESD204B_mac:i_JESD204B_mac|sysref
...4B_mac:i_JESD204B_mac|jesd204_rx_link_ready
...4B_mac:i_JESD204B_mac|jesd204_rx_link_valid
...roller|JESD204B_mac:i_JESD204B_mac|somf[3..0]
...b_controller|channel_a_ob64.samples[3..0][15..0]
...controller|channel_a_ob64.samples[3][15..0]
...controller|channel_a_ob64.samples[2][15..0]
...controller|channel_a_ob64.samples[1][15..0]
...controller|channel_a_ob64.samples[0][15..0]
CGS
ILAS
FFFFFFFFh
0E3h
1BBh
DEVICES ALIGNED
0h
DATA
0000h

Figure 3.40: SignalTap output for JESD204B Link layer - Overview.

Here, the various JESD204B phases (CGS, ILAS, data, etc.) can be easily identified, along with key signals such as dev_lane_aligned, jesd204_rx_link_valid, and the output from the transport layer, which, for the purposes of this thesis, is split into four distinct channel outputs.

Lastly, as outlined in Section 2.5.1, it is crucial to validate the correct operation of JESD204B by configuring the ADC in test mode and applying the "Chess pattern" and the "Triangle Pattern". In order to change the operational mode of the ADC there has been created a python script for modifying the necessary register:



```
jpresman@cfv-865-bidev10:~>python ADS54J54_activate_triangle_pattern.py
[REG WRITING] 0x1d -> 0x60
[VALID REG] 0x1d value is 0x60 == 0x60
jpresman@cfv-865-bidev10:~>python ADS54J54_activate_chess_pattern.py
[REG WRITING] 0x1d -> 0x68
[VALID REG] 0x1d value is 0x68 == 0x68
```

Figure 3.41: Python script for modifiying the ADS54J54 operational mode.

Figure 3.42 displays the output for the triangle pattern, wherein we can discern a "frequency divider" pattern for each bit of the sample. This pattern is also readily identifiable when we view the sample as a bit group rather than individual bits, as shown in Figure 3.43. Here, it can be seen that the samples increase in value by one with each clock cycle:

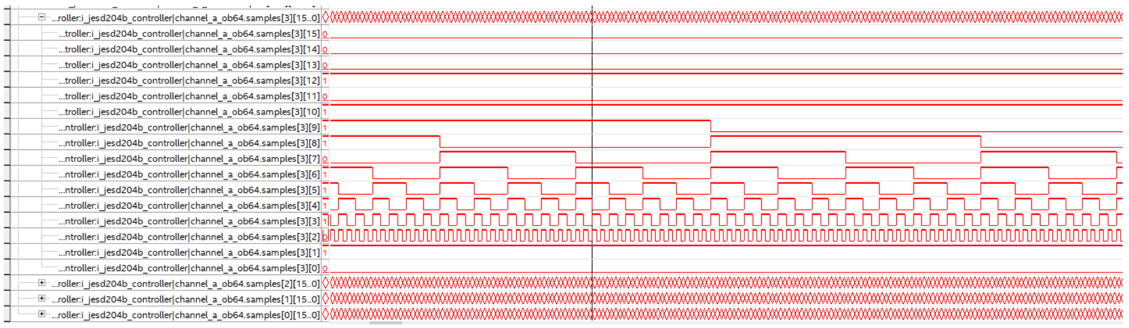

...roller:i_jesd204b_controller|channel_a_ob64.samples[3][15..0]
...troller:i_jesd204b_controller|channel_a_ob64.samples[3][15] 0
...troller:i_jesd204b_controller|channel_a_ob64.samples[3][14] 0
...troller:i_jesd204b_controller|channel_a_ob64.samples[3][13] 0
...troller:i_jesd204b_controller|channel_a_ob64.samples[3][12] 1
...troller:i_jesd204b_controller|channel_a_ob64.samples[3][11] 0
...troller:i_jesd204b_controller|channel_a_ob64.samples[3][10] 1
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][9] 1
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][8] 1
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][7] 0
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][6] 1
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][5] 1
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][4] 1
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][3] 1
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][2] 0
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][1] 1
...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3][0] 0
...roller:i_jesd204b_controller|channel_a_ob64.samples[2][15..0]
...roller:i_jesd204b_controller|channel_a_ob64.samples[1][15..0]
...roller:i_jesd204b_controller|channel_a_ob64.samples[0][15..0]

Figure 3.42: ADC triangle pattern bit-by-bit.

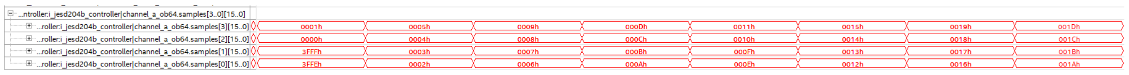

...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3..0][15..0]
...roller:i_jesd204b_controller|channel_a_ob64.samples[3][15..0] 0001h 0005h 0009h 000Dh 0011h 0015h 0019h 001Dh
...roller:i_jesd204b_controller|channel_a_ob64.samples[2][15..0] 0000h 0004h 0008h 000Ch 0010h 0014h 0018h 001Ch
...roller:i_jesd204b_controller|channel_a_ob64.samples[1][15..0] 3FFFh 0003h 0007h 000Bh 000Fh 0013h 0017h 001Bh
...roller:i_jesd204b_controller|channel_a_ob64.samples[0][15..0] 3FFEh 0002h 0006h 000Ah 000Eh 0012h 0016h 001Ah

Figure 3.43: ADC Triangle pattern sample value increment.

The same procedure can be implemented for the Chess pattern, wherein the hexadecimal values of $0x1555$ and $0x2AAA$ (as outlined in Subsection 2.5.1) can be alternately observed for each sample, as demonstrated in Figure 3.44:

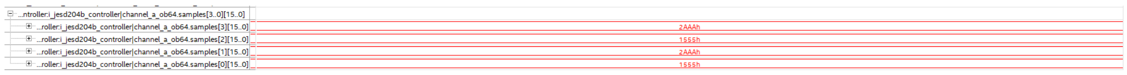

...ntroller:i_jesd204b_controller|channel_a_ob64.samples[3..0][15..0]
...roller:i_jesd204b_controller|channel_a_ob64.samples[3][15..0] 2AAAh
...roller:i_jesd204b_controller|channel_a_ob64.samples[2][15..0] 1555h
...roller:i_jesd204b_controller|channel_a_ob64.samples[1][15..0] 2AAAh
...roller:i_jesd204b_controller|channel_a_ob64.samples[0][15..0] 1555h

Figure 3.44: ADC Chess pattern.

3.3.1.2.3 Simulation Since the JESD204B protocol was so difficult to understand and the fitting time was quite long, we also decided to implement a simulation testbench in order to understand the different parts and interconnections of the modules.

Figure 3.45 shown the testbench architecture:

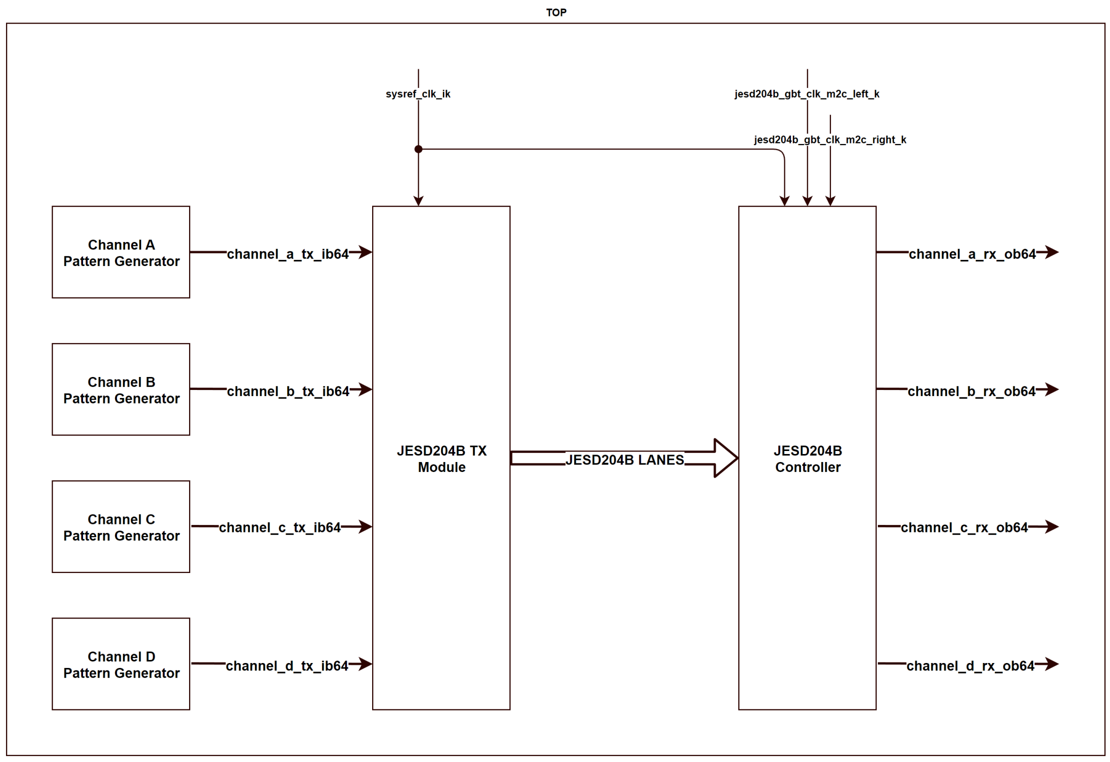


Figure 3.45: JESD204B Controller testbench.

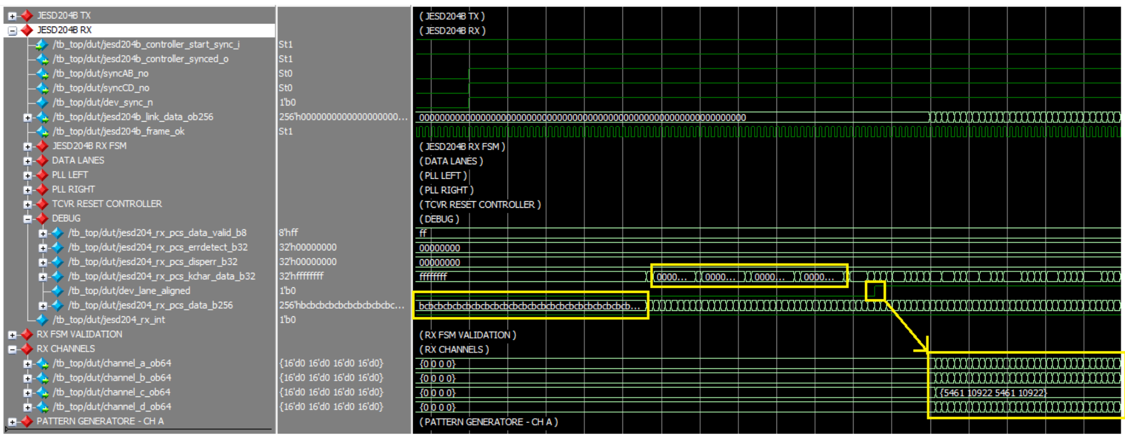
In this simulation testbench, we instantiated a TX module to emulate the ADC behaviour.

The input for this module is a pattern generator module for each channel, enabling independent examination of each channel. This pattern generator can produce a sinusoidal waveform, triangle waveform, chess pattern, and random pattern.

Contained within the TX module is a transport layer, a FSM, all necessary PLLs, and the Altera IP Cores.

The TX module's output are the ”physical lanes” of the JESD204B protocol, connected to the RX and subsequently to our custom *JESD204B_Controller* module.

After running the simulation, we obtain the following output:



```
JESD204B TX
JESD204B RX
  /tb_top/dut/jesd204b_controller_start_sync_i     St1
  /tb_top/dut/jesd204b_controller_synced_o         St1
  /tb_top/dut/syncAB_no                            St0
  /tb_top/dut/syncCD_no                            St0
  /tb_top/dut/dev_sync_n                           1'b0
  /tb_top/dut/jesd204b_link_data_ob256             256'h00000000000000000000...
  /tb_top/dut/jesd204b_frame_ok                    St1
  JESD204B RX FSM
  DATA LANES
  PLL LEFT
  PLL RIGHT
  TCVR RESET CONTROLLER
  DEBUG
    /tb_top/dut/jesd204_rx_pcs_data_valid_b8       8'hff
    /tb_top/dut/jesd204_rx_pcs_errdetect_b32       32'h00000000
    /tb_top/dut/jesd204_rx_pcs_disperr_b32         32'h00000000
    /tb_top/dut/jesd204_rx_pcs_kchar_data_b32      32'hffffffff
    /tb_top/dut/dev_lane_aligned                   1'b0
    /tb_top/dut/jesd204_rx_pcs_data_b256           256'hbcbcbcbcbcbcbcbcbcbc...
  /tb_top/dut/jesd204_rx_int                       1'b0
RX FSM VALIDATION
RX CHANNELS
  /tb_top/dut/channel_a_ob64                       {16'd0 16'd0 16'd0 16'd0}
  /tb_top/dut/channel_b_ob64                       {16'd0 16'd0 16'd0 16'd0}
  /tb_top/dut/channel_c_ob64                       {16'd0 16'd0 16'd0 16'd0}
  /tb_top/dut/channel_d_ob64                       {16'd0 16'd0 16'd0 16'd0}
PATTERN GENERATORE - CH A
```

Figure 3.46: JESD204B Controller testbench waveform.

As in the SignalTap we can check the different signals and phases of the JESD204B protocol. Here we can distinguish between CGS, ILAS and DATA phase. Also we can check that the received data from the TX module and after the transport layer is as we expected, for example by applying a triangle waveform to one channel:

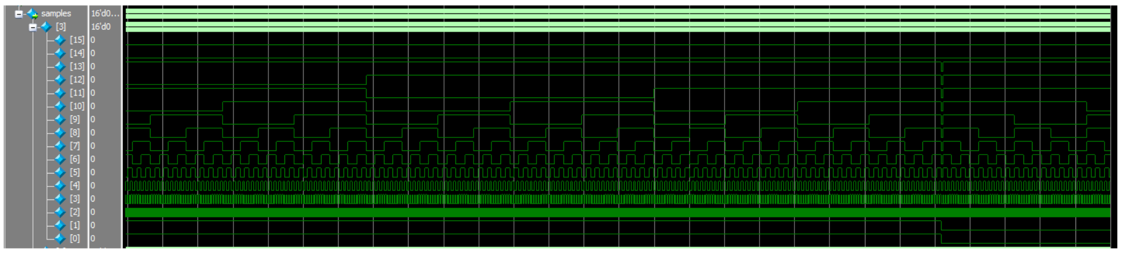
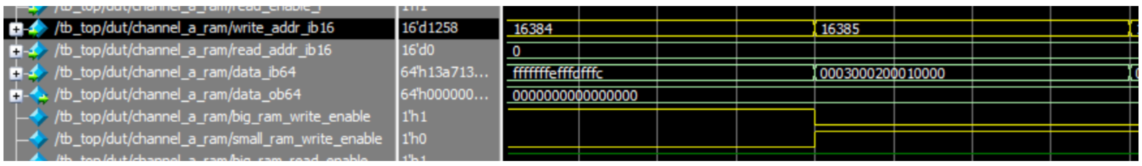


Figure 3.47: JESD204B Controller testbench triangle output.

3.3.2 RAM Controller

The same procedure was utilized to test the *RAM_Controller* module, but in this instance, we will concentrate on the simulation aspect, as Modelsim offers us tools to better debug the RAM memories.

The initial feature that the testbench seeks to test is the change of RAM selector. This implies that when the large RAM is full, the system must switch the RAM being used for data storage to the smaller RAM. The subsequent Figure illustrates this process:



```
/tb_top/dut/channel_a_ram/write_addr_ib16         16'd1258     16384              16385
/tb_top/dut/channel_a_ram/read_addr_ib16          16'd0        0
/tb_top/dut/channel_a_ram/data_ib64               64'h13a713...  fffffffefffdfffc  0003000200010000
/tb_top/dut/channel_a_ram/data_ob64               64'h000000...  0000000000000000
/tb_top/dut/channel_a_ram/big_ram_write_enable    1'h1
/tb_top/dut/channel_a_ram/small_ram_write_enable  1'h0
```

Figure 3.48: Big RAM / Samll RAM write enable selector.

The next feature to be tested is the finite state machine shown in Subsubsection 3.2.3.2 which is represented in Figure 3.49 :

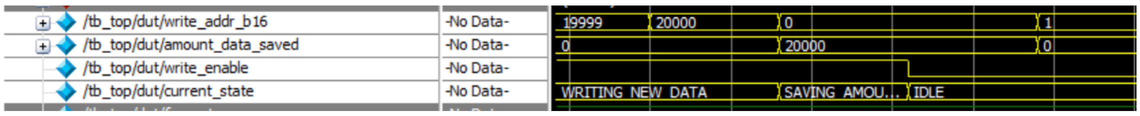


Figure 3.49: RAM Controller testbench FSM waveform.

In this figure, we can observe that when the maximum data storage capacity is reached (20,000 elements in this case), the current state of the FSM transitions from *"WRITING NEW DATA"* to *"SAVING AMOUNT DATA WRITTEN"*. In this state, the write address shifts to address $0x0$ and the write data changes to $20,000$. After one clock cycle, the state progresses to *"IDLE"*, where the variable for the amount of data saved is reset to zero.

Once we have checked the correct behaviour in the simulation we can move to the Signal Tap instance and check the writing and reading process as shown in Figure 3.50 and Figure 3.51:

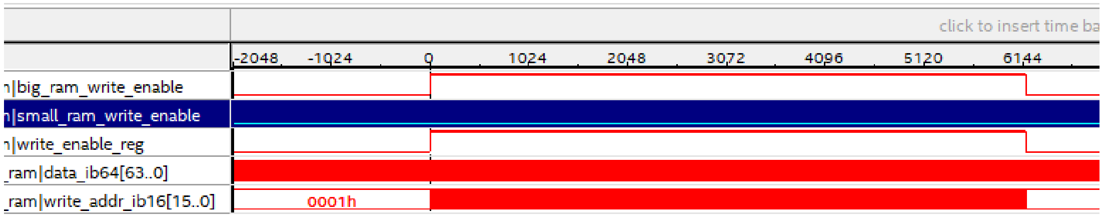


Figure 3.50: RAM Controller Signal Tap output for writing.

To achieve this waveform, a trigger start and stop signals of with a $50\mu s$ delay in between them has been injected. It is noticeable that the large RAM enable signal is activated while the write address signal undergoes changes during this period.

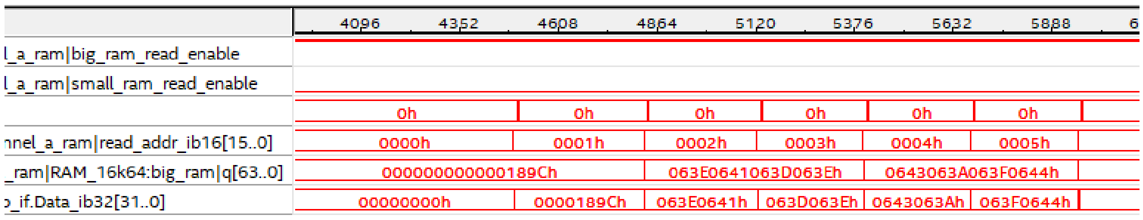


Figure 3.51: RAM Controller Signal Tap output for reading.

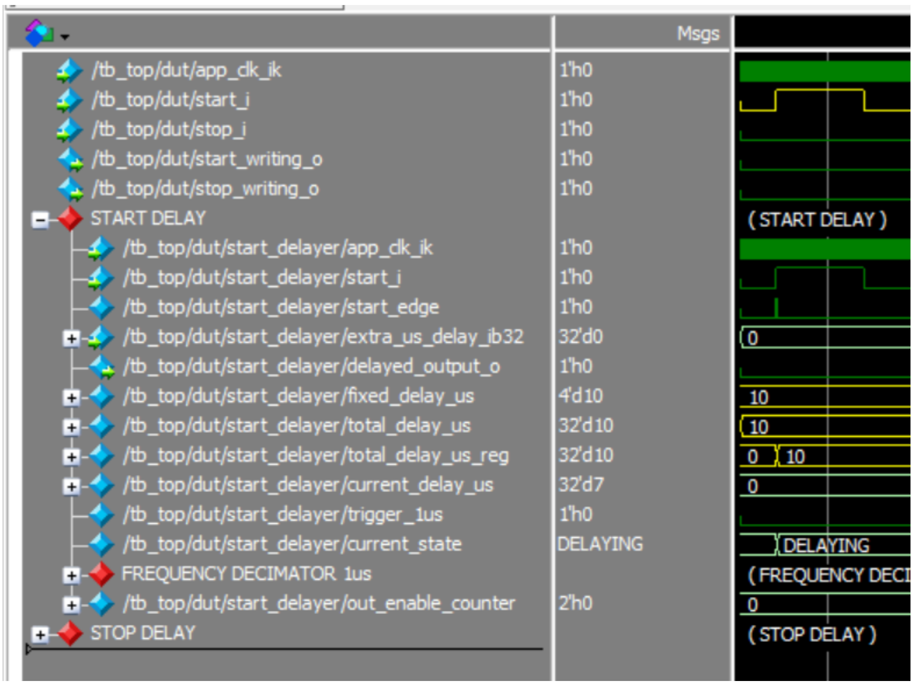
In this figure, we can observe that as the large RAM read enable is activated, the read address changes incrementally. This read-back operation is conducted by a Python script tasked with reading the entirety of the RAM.

We can also note that two different read addresses are required to change the output of the large RAM. This is attributed to the addressing theory expounded in Paragraph 3.2.3.1.3. It can be seen in the Wishbone data input bus (Data_ib32 signal) that when the read address changes, the data available at Wishbone also changes.

3.3.3 Trigger Controller

The final module to be verified for this thesis is the *Trigger_Controller* module. To verify this module, we used only a simulation process to ensure that once the start/stop signals are reached, the system commences the count of the necessary microseconds.

On one hand, the module captures the configuration in which it will operate. That is, the module will register the amount of time it will delay the input for this cycle. This process can be seen in the following figure:

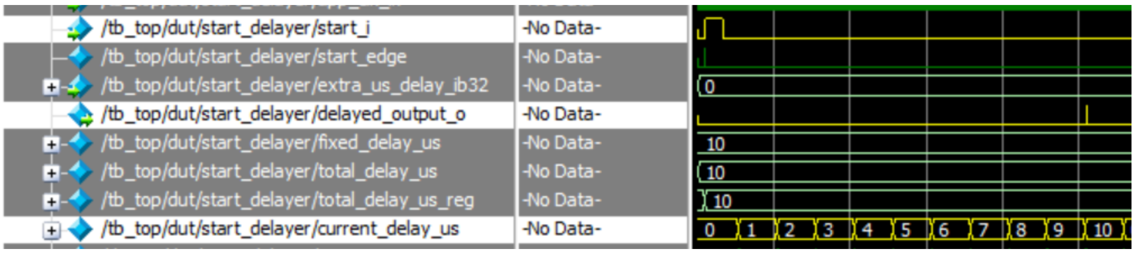


```
                                                        Msgs
/tb_top/dut/app_clk_ik                                  1'h0
/tb_top/dut/start_i                                     1'h0
/tb_top/dut/stop_i                                      1'h0
/tb_top/dut/start_writing_o                             1'h0
/tb_top/dut/stop_writing_o                              1'h0
START DELAY
    /tb_top/dut/start_delayer/app_clk_ik                1'h0
    /tb_top/dut/start_delayer/start_i                   1'h0
    /tb_top/dut/start_delayer/start_edge                1'h0
    /tb_top/dut/start_delayer/extra_us_delay_ib32       32'd0
    /tb_top/dut/start_delayer/delayed_output_o          1'h0
    /tb_top/dut/start_delayer/fixed_delay_us            4'd10
    /tb_top/dut/start_delayer/total_delay_us            32'd10
    /tb_top/dut/start_delayer/total_delay_us_reg        32'd10
    /tb_top/dut/start_delayer/current_delay_us          32'd7
    /tb_top/dut/start_delayer/trigger_1us               1'h0
    /tb_top/dut/start_delayer/current_state             DELAYING
    FREQUENCY DECIMATOR 1us
    /tb_top/dut/start_delayer/out_enable_counter        2'h0
STOP DELAY
```

Figure 3.52: Trigger controller waveform capturing the configuration.

In this instance, we can observe that once the start signal is received, the system adds the extra_us_delay_ib32 variable to the fixed_delay_us variable and saves this value to total_delay_us_reg.

The second feature to be examined is the delay itself, how the system captures the start input signal and applies the delay, as shown in Figure 3.53:

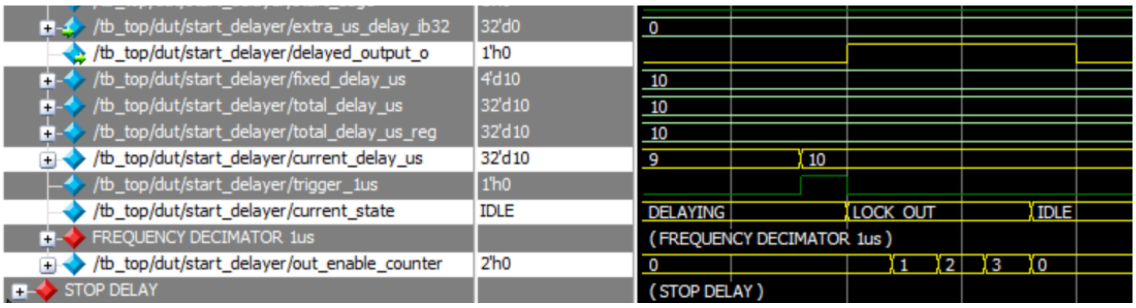


```
/tb_top/dut/start_delayer/start_i                -No Data-
/tb_top/dut/start_delayer/start_edge             -No Data-
/tb_top/dut/start_delayer/extra_us_delay_ib32    -No Data-
/tb_top/dut/start_delayer/delayed_output_o       -No Data-
/tb_top/dut/start_delayer/fixed_delay_us         -No Data-
/tb_top/dut/start_delayer/total_delay_us         -No Data-
/tb_top/dut/start_delayer/total_delay_us_reg     -No Data-
/tb_top/dut/start_delayer/current_delay_us       -No Data-
```

Figure 3.53: Trigger controller waveform start delayed output.

Once the system reaches the value of 10, we can observe that the output is asserted to a high level.

Lastly, the finite state machine should be examined, as shown in the following figure:



/tb_top/dut/start_delayer/extra_us_delay_ib32	32'd0
/tb_top/dut/start_delayer/delayed_output_o	1'h0
/tb_top/dut/start_delayer/fixed_delay_us	4'd10
/tb_top/dut/start_delayer/total_delay_us	32'd10
/tb_top/dut/start_delayer/total_delay_us_reg	32'd10
/tb_top/dut/start_delayer/current_delay_us	32'd10
/tb_top/dut/start_delayer/trigger_1us	1'h0
/tb_top/dut/start_delayer/current_state	IDLE
FREQUENCY DECIMATOR 1us	
/tb_top/dut/start_delayer/out_enable_counter	2'h0
STOP DELAY	

Figure 3.54: Trigger controller waveform FSM.

The system transitions from the *"DELAYING"* state to the *"LOCKED OUT"* state once it reaches the configured delay time. In this new state, we can observe that the output remains asserted for as long as the module remains in this state (five clock cycles), after which it transitions to the *"IDLE"* state, awaiting a new trigger.

3.4 Results

To conclude this chapter, and utilizing the set-up illustrated in Figure 1.4, we will analyze the results of injecting various trigger signals and inputs into Channel A of the Mezzanine board.

By using a Python script to read the entirety of the memory, saving it to a CSV file, and another Python script to analyze and plot the contents of this file, we can verify the correct functioning of the entire system working in tandem.

Initially, the first Figure to be presented will depict a sinusoidal signal of 250kHz injected into the ADC, with the trigger system generating start/stop signals spaced at $100\mu s$:

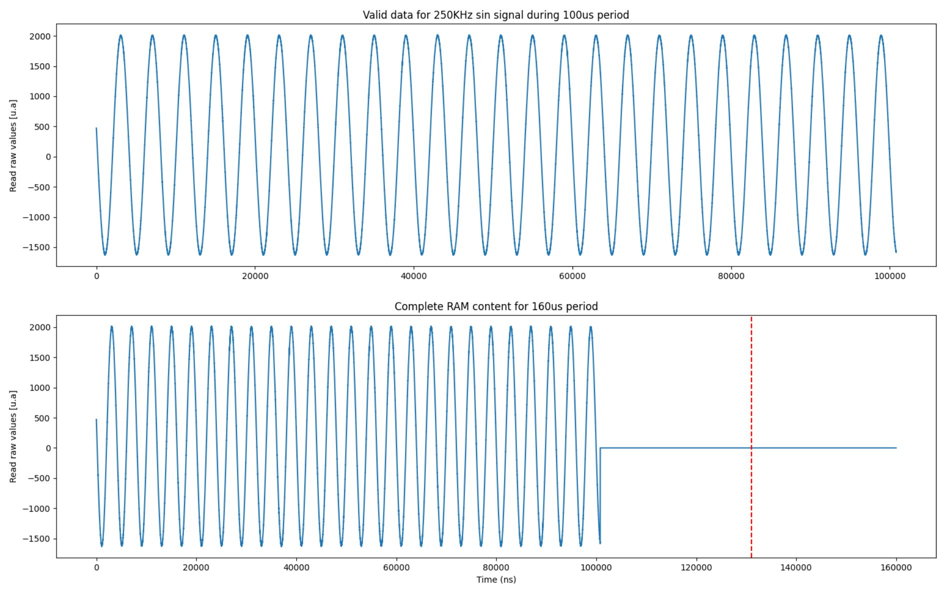


Figure 3.55: Sin wave plot (250kHz, 100μs).

In this figure, we examine two different plots. The first plot displays only the valid elements saved into the RAM memory. The quantity of these elements is stored in the first RAM address, and as depicted, it represents approximately 100μs.

The second plot depicts the entire RAM up to 160μs. This was done to show that the remaining values are not written, thereby confirming that the trigger system is correctly implemented.

Figure 3.56 represents a triangular wave of 250kHz with a 135μs delay between start and stop trigger signals:

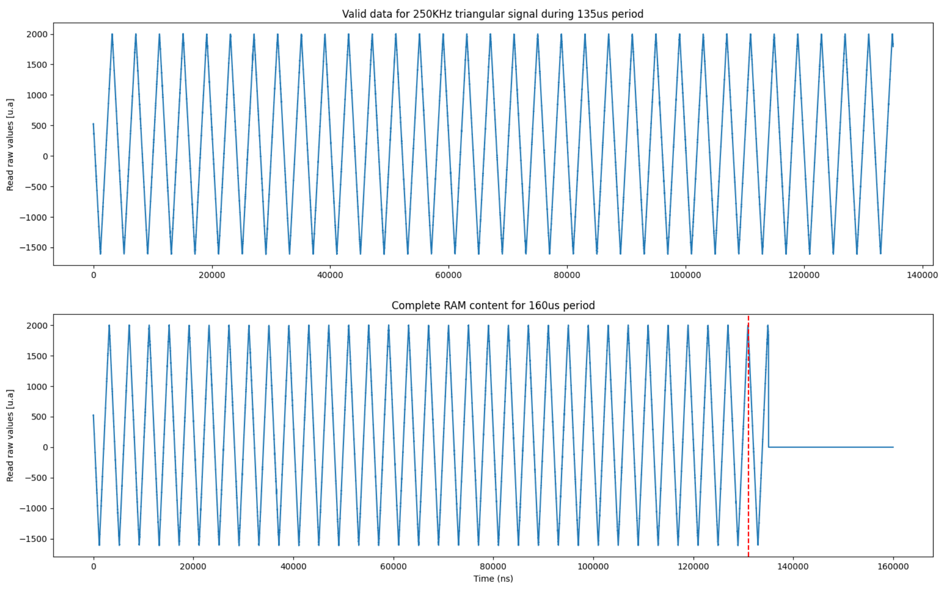


Figure 3.56: Triangular wave plot (250kHz, 135μs).

This plot also displays a different trigger period and waveform. However, notably, there is a vertical red line composed of dots. This line represents the physical point where the big and the small RAM are separated. This point serves to validate that the addressing system is functioning correctly.

To better appreciate this phenomenon, Figure 3.57 provides a more detailed view of the waveform at this moment:

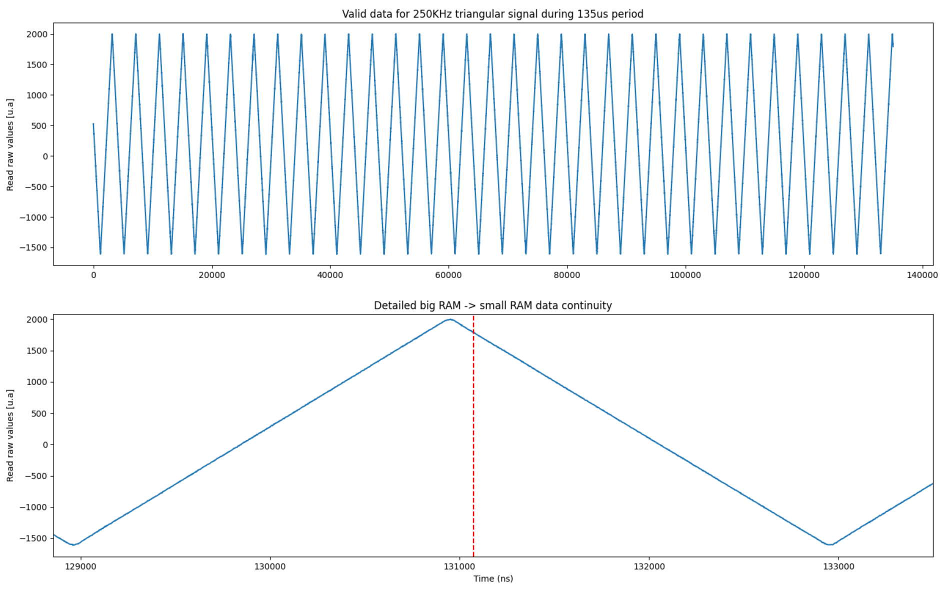


Figure 3.57: Detailed triangular wave plot (250kHz, 135μs).

Here, we can observe a continuity in the waveform when zoomed in, which affirms that the addressing system is functioning as intended.

Chapter 4

Conclusions

The obsolescence of the hardware currently used on the BLM diamond acquisition system at PSB, its homogenization with other equipment used by the CERN beam instrumentation group and the improvement of the system capabilities are some of the driving factors behind the design and implementation of the new acquisition platform described in this thesis.

The installation of the system (detectors, front-end and processing crates) in its final PSB location as well as the laboratory setup used during the development and debugging phases, including the distribution of the triggering and detector signals has been thoroughly detailed.

Before defining the various logic modules that have been developed and tested, we provided an overview of the main topics needed to understand the system. These range from an overview of various high-speed data acquisition concepts, such as clock domain crossing or synchronization, as well as a description of the hardware components that constitute the system.

Additionally, this work has made use of many different communication protocols. These were used either between components on the same card (e.g., SPI), between different cards (e.g., VME64x and JESD204B), or among modules within the FPGA logic (e.g., Wishbone or Avalon). Understanding their characteristics and use cases has been a crucial part of the project, since the FPGA firmware developed required either their implementation, either their configuration and integration and in all cases, the validation of their correct functioning. Therefore, the most important ones have been discussed.

Having grasped all the theoretical concepts, the firmware architecture was outlined, demonstrating how the VFC firmware is built upon a template which divides the board into a System and Application layer. The different modules implemented on the Application layer were elaborated upon, highlighting how these modules have been partitioned based on their functionality, and the rationale behind their encapsulation.

Lastly, we delved into the methods used for testing and verifying the different parts, always ensuring that the system fulfills the specifications. These included simulation, in-system probing with an embedded logic analyzer such as Signal Tap, Static Timing Analysis and Python scripts. Additionally, we compared the differences in how a system can be evaluated using some of these complementary methods, exemplified by the JESD204B link process.

On the other hand, given the breadth of this project, it's important to note that, while a prototype capable of capturing the diamond signals is currently available, it is not yet completed. Further testing for edge cases is foreseen, as well as the development of a graphical software interface application and the system installation and commissioning in a real-world environment. Moreover, a crucial aspect that needs to be implemented is the beam losses integration per turn basis.

Bibliography

[1] E. Mobs. *The CERN accelerator complex - August 2018. Complexe des accélérateurs du CERN - Août 2018*. OPEN-PHO-ACCEL-2018-005. General Photo. Aug. 2018. URL: https://cds.cern.ch/record/2636343.

[2] *Oasis Acquisition System*. URL: https://cds.cern.ch/record/693174/files/ab-2003-110.pdf.

[3] *U1063A 8-bit High-Speed cPCI Digitizers*. URL: https://www.keysight.com/us/en/product/U1063A/8bit-highspeed-cpci-digitizers.html.

[4] *PSB BLM Design Requirements*. URL: https://edms.cern.ch/ui/file/2054239/1.1/PSB-BLM-ES-0001-10-10.pdf.

[5] *FMC github*. URL: https://fmchub.github.io/appendix/VITA57_FMC_HPC_LPC_SIGNALS_AND_PINOUT.html.

[6] *IAM500 shopping webpage*. URL: http://www.iamelectronic.com/shop/produkt/fpga-mezzanine-card-fmc-adc-4x-500-msps-14-bit-dc-coupled/.

[7] Dr. Philipp Födisch. *IAM 500 Technical Documentation 1.0*. 2019FMC0020. Nov. 2019.

[8] Texas Instruments. *ADS54J54 Quad Channel 14-Bit 500 MSPS ADC*. Jan. 2015. URL: https://www.ti.com/lit/ds/symlink/ads54j54.pdf?ts=1680735767278.

[9] SkyWorks. *Si5395/94/92 Data Sheet*. June 2018. URL: https://www.skyworksinc.com/-/media/Skyworks/SL/documents/public/data-sheets/si5395-94-92-a-datasheet.pdf.

[10] *Diamond Sensor*. URL: https://cividec.at/index.php?module=public.product&idProduct=11&scr=0.

[11] *Boardband Diamong Amplifier 2GHz*. URL: https://cividec.at/index.php?module=public.product&idProduct=33&scr=0.

[12] *FreeRTOS*. URL: https://www.freertos.org/.

[13] *The evolution of the CERN SPS timing system for the LHC era*. URL: https://inspirehep.net/literature/636904.

[14] *OSI Model*. URL: https://en.wikipedia.org/wiki/OSI_model.

[15] *Intel PMA Doc*. URL: https://www.intel.com/content/www/us/en/docs/programmable/683621/current/pma-architecture.html.

[16] *8b/10b Encoding system*. URL: https://en.wikipedia.org/wiki/8b/10b_encoding.

[17] *Wishbone Specification*. URL: https://cdn.opencores.org/downloads/wbspec_b3.pdf.

[18] *American National Standard for VME64*. URL: `https://www.ge.infn.it/~musico/Vme/Vme64.pdf`.

[19] *JESD204B Intel documentation*. URL: `https://cdrdv2-public.intel.com/730782/ug_jesd204b-683442-730782.pdf`.