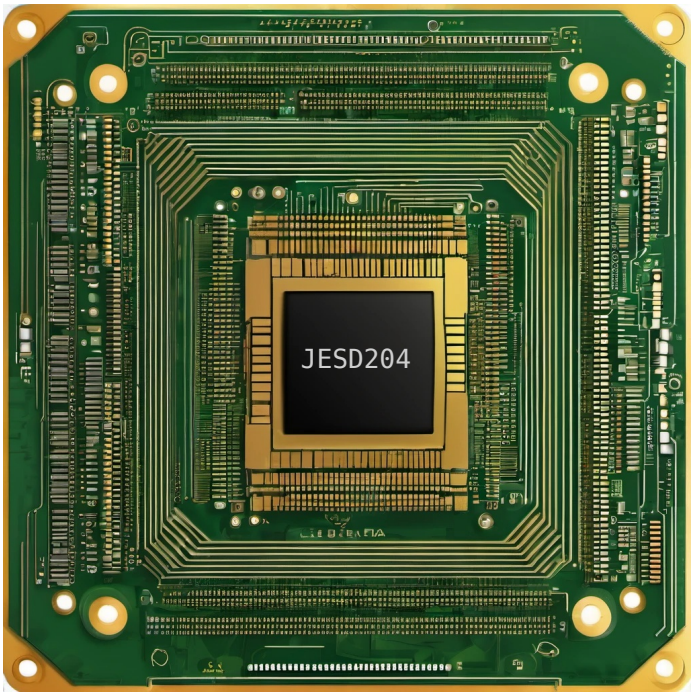
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



JESD204 Receiver and Data Reduction Implementation for an SoC Platform

Master's thesis in Embedded Electronic System Design

Ammar Shihabi
Lucas Johansson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

JESD204 Receiver and Data Reduction Implementation for an SoC Platform

Ammar Shihabi
Lucas Johansson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

JESD204 Receiver and Data Reduction Implementation for SoC Platform

Ammar Shihabi
Lucas Johansson

Supervisor: Lars Svensson, Department of computer science and engineering
Advisors: Jan Andersson, Joaquín España Navarro, Frontgrade Gaisler
Examiner: Per Larsson-Edefors, Department of computer science and engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: AI generated picture of "System on Chip" from Pixlr

Typeset in LaTeX
Gothenburg, Sweden 2024

JESD204 Receiver and Data Reduction Implementation for SoC Platform
Ammar Shihabi, Lucas Johansson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Field programmable gate arrays (FPGAs) are currently used within the signal processing chain of space systems to transfer data from high-speed sensors. Such systems employ a number of FPGAs, that are primarily used for tasks such as data reception and data reduction, which is a critical step as microprocessors often struggle to handle data at such high rates. The FPGAs cause an overall increase in resource and power usage for the critical space computer systems with limited resources. This thesis presents a prototype implementation of a JESD204B high-speed serial receiver, together with an investigation of some existing data reduction algorithms that are suitable for hardware implementation.

The methodology used in this project involved constructing an SoC subsystem on an FPGA to create the signal processing chain needed to obtain high-speed communication. Formal verification methods were used extensively to verify the functionality of the receiver. Python was used to explore two different implementations for data reduction. The receiver RTL demonstrated correct behavioral functionality against a transmitter in simulation. Although the receiver was successfully implemented on the FPGA, actual data reception on the hardware was not achieved due to time limitations. The study of the algorithms showed valuable results, making them practical both for research and in terms of hardware implementation. Future work includes establishing receiver and transmitter communication to read actual data on hardware, further developing the receiver, and finally implementing the data reduction algorithms in hardware.

Keywords: JESD204 receiver, data reduction algorithms, FPGA, System On Chip, ADC, SerDes, Register Transfer Level, Advanced Peripheral Bus, Advanced High-performance Bus, GRLIB.

Acknowledgements

First and foremost, we would like to express our deepest gratitude to our families. Their unwavering support, infinite love, and endless encouragement have been the reason for us continuing our journey and this education. Through every challenge and achievement, they have stood by us, offering strength and inspiration. This thesis is as much a testament to their dedication and sacrifices as it is to our hard work. We are extremely grateful for their presence in our lives and for believing in us every step of the way.

We also extend our sincere thanks to Jan Andersson for giving us this opportunity to do this project. Special thanks to our technical advisor Joaquín España Navarro and the others at Gaisler for giving us technical guidelines, as well as inspiring ideas to enhance the outcome of this project. Finally, a big thanks to our academic supervisor from Chalmers, Prof. Lars Svensson, as well as our examiner Prof. Per-Larsson Edefors, who not only gave invaluable counseling when planning this project but also provided careful reviews and insightful feedback.

Ammar Shihabi, Lucas Johansson, Gothenburg, 2024-09-18

Contents

Abbreviations and definitions

General
FPGA : Field-Programmable Gate Array
ADC : Analog to Digital Converter
SoC : System on Chip
SerDes : Serializer/De-serializer
RD : Running Disparity
RLE : Run-Length Encoding
SMA : Sliding moving average
BMA : Block moving average
Octet : A group of eight bits
VHDL : Very High-Speed Integrated Circuit Hardware Description Language
RTL : Register-Transfer Level
IP : Intellectual property
SPI : Serial Peripheral Interface
AMBA : Advanced Microcontroller Bus Architecture
APB : Advanced Peripheral Bus
AHB : Advanced High-performance Bus
GRLIB : Gaisler Research IP Library
GRMON : Gaisler Research Monitor
LVDS : Low-Voltage Differential Signaling
PLL : Phase-Locked Loop

JESD204 specific
F : Number of octets per frame on a single lane
K : Number of frames per multiframe
N : Converter resolution
N' : Bits per sample
CS : Control Bits
T : Tail Bits
LMFC : Local Multi-Frame Clock
Lane : A high-speed serial data channel in the JESD204B interface
Link : A high-speed connection between transmitter and receiver devices
Device clock : Clock signal from which a Tx or Rx must generate its local clocks
Core clock : A clock inside a transmitter or receiver device, used in the implementation of the JESD204 link
CGS : Code Group Synchronization
ILA : Initial Lane Alignment
Subclass : JESD204B supports three subclasses (0, 1, 2), each providing different synchronization
SYSREF : Signal for achieving deterministic latency in subclass 1 systems
SYNC$\sim$: Signal for establishing synchronization between transmitter and receiver, and deterministic latency in subclass 2 systems

1

Introduction

Contemporary space systems employ field programmable gate arrays (FPGAs) within the signal processing chain, transferring data from high-speed sensors using analog to digital converters (ADCs) to the FPGAs, before reaching the microprocessor system on chip (SoC) [1]. The FPGA's primary role in these systems would be to act as a high-speed serial link to receive sensor data, which is a critical step as the SoC processor often struggles to handle data at the requisite high rates [2]. The corresponding large amount of sensor data needs to be handled in the critical space computer systems with limited resources.

Often, a series of FPGAs are used to perform different tasks, such as data reception and data reduction. However, the FPGAs used for data reduction come with a significant overhead in terms of power dissipation and resource usage. In this report, we explore the feasibility of implementing a high-speed serial link by the means of a JESD204 receiver as an IP-core which will be directly integrated into the GR765 SoC from Frontgrade Gaisler [3]. Additionally, our work will attempt to investigate data reduction algorithms on the incoming sensor data on the embedded FPGA fabric of the SoC. This would eliminate the need for an external FPGA.

1.1 Related work

One relevant example of an existing architecture that could be improved through the proposed thesis work, is the Laser-Ablation Time-of-Flight Mass Spectrometer instrument (LMS) designed at the Physikalisches Institut of the University of Bern [4][5]. This application produces 4 GBps (32 Gbps) in an architecture that is in the end controlled by a GR712RC 100 MHz dual-core 32-bit processor. The GR712RC SoC lacks a high-speed serial interface to acquire the data and does not have the computational capabilities to process a high-bandwidth data stream [6]. To solve the problem, the GR712RC would today be combined with several FPGAs to do acquisition and data decimation, where relatively simple operations such as binning can reduce the data stream from Gbps to kbps.

1.2 Purpose and aim

The goal of this project is to investigate the feasibility of implementing a JESD204 receiver into Gaisler's SoC. For that, we will study and specify the requirements of a JESD204 receiver, to establish communication with data converters, with the goal of developing a receiver prototype. Furthermore, we aim to study different data reduction algorithms, and perform data reduction on the received data.

The motivation behind this integration is to achieve an overall reduction in resource and power usage for SoC processors. To interface a JESD204 component, which is gaining popularity amongst data converter manufacturers [7], with any of the Gaisler LEON/NOEL processors [8], customers would currently need to implement their own JESD204 receiver on an external FPGA, increasing resource usage and power in resource-constrained space applications.

1.3 Scope and limitations

We develop this system for usage within Gaisler's own SoC and GRLIB, which is a library containing different SoC IP cores [9]. Verification of the receiver needs to be done, ideally using a third party transmitter to make sure the standard in the receiver is as specified by the JESD204 specification. The RTL of the receiver IP core will be verified using EDA tools, then implemented on a hardware platform by the means of an FPGA.

The primary focus of this work will be on the implementation of the JESD204 receiver, with the development and complexity of the data reduction algorithms being considered a secondary priority, since the reduction rate and accuracy of the data reduction algorithm are not useful without a functional receiver.

An important part of the complete IP core would be the interface between the JESD204 receiver and the direct memory access (DMA) engine. Writing a new DMA engine can be beneficial for the new JESD204 receiver. However, this would introduce extra time and complexity to this project, which is why it will be omitted from the scope of this thesis.

1.4 Report organization

In the theory chapter, a technical background of the JESD204 standard is first established. Communication protocols and other modules related to the system are introduced. Then, we will investigate some existing data reduction algorithms, with the intent to perform data reduction on received data. In the design and implementation chapter, the system is described from top to bottom in detail. In chapter five, the performance of the system is evaluated by means of behavioral simulation of RTL and hardware implementation. Chapter six concludes our work with some reflections, and presents ideas for future work surrounding this project.

2

Technical background

In this part, necessary technical background for this thesis project is presented. We will review aspects such as the JESD204 standard with its different layers, data reduction, serial communication protocol for ADC programming, GRLIB and the tools used in this project.

For the hardware part, we focus on the establishment of the sensor data reception from an ADC. An FPGA is used for the receiver implementation and to support an investigation of algorithmic data reduction, leveraging plenty of available programmable logic units and memory resources.

Figure 2.1 presents the intended JESD204 receiver IP-core on a Gaisler SoC architecture.

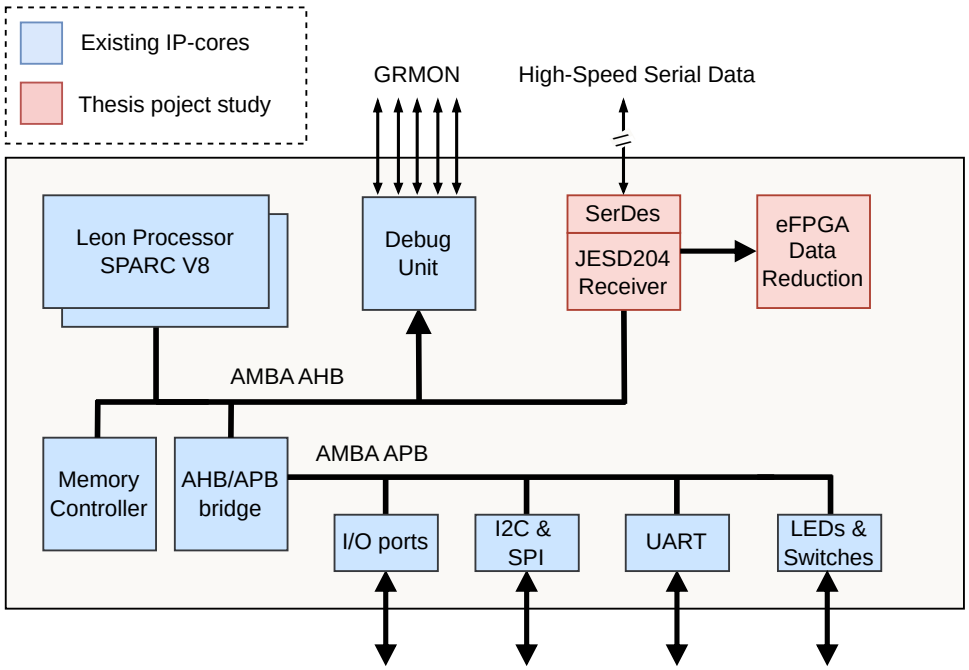


Figure 2.1: System overview of the desired JESD204 receiver IP, with data reduction implementation on FPGA fabric, inside an SoC.

2.1 Field programmable gate array (FPGA)

An FPGA is a reconfigurable integrated circuit that can be configured to mimic the behavior of any logic circuit. FPGAs are configured via a hardware description language (HDL), such as Verilog or VHDL. FPGAs are made up of interconnects and configurable logic blocks (CLBs), which use lookup-tables (LUTs) to implement desired logic gates. Due to the unique design of FPGAs, the CLBs can be configured and operate in parallel, making FPGAs a very suitable platform for high-performance applications [10].

Compared with a design implemented in an application specific integrated circuit (ASIC), the same design implemented in an FPGA will consume more power [10]. FPGAs carry additional resources and routing to support reconfigurability, which in turn results in higher power consumption compared to the optimized, fixed paths in ASICs. However, this trade-off allows designs implemented in FPGAs to be more versatile and quicker to deploy, as they do not require the extensive fabrication process associated with ASICs.

2.2 Data converters

Data converters serve as a link between the analog world and the digital world. They are responsible for the transformation of data between the discrete and continuous domains [11].

2.2.1 Analog-to-digital conversion

An analog-to-digital converter (ADC) converts continuous-time and continuous-amplitude signals into discrete-time and discrete-amplitude. The main blocks of an ADC are the sampler and quantizer [11]. The purpose of a sampler is to sample the continuous signal at certain time intervals, and construct a sampled duplicate of the original waveform. The distance between two samples is determined by the ADC sampling frequency. One should opt for a sampling frequency operating at no less than twice the rate of the frequency present in the input signal, in order to reduce aliasing. To further avoid aliasing and interference, an anti-aliasing filter is commonly implemented before the sampler.

The role of the quantizer is to map the captured samples to a fixed set of discrete digital values, determined by the ADC resolution. Greater resolution leads to a small quantization error, which is the difference between the original sample and the output.

2.2.2 Digital-to-analog conversion

A digital-to-analog converter (DAC) consists of a transcoder stage and a reconstruction stage. The transcoder converts digital samples into analog pulses. The reconstruction stage merges the analog pulses into a continuous signal, using a sample-

and-hold (S&H) circuit, and a reconstruction filter. The filter will remove high frequency components, resulting in a more polished output [11].

2.3 Serial peripheral interface (SPI)

Serial communication protocols are needed to configure many ADCs and DACs. One common protocol that is extensively used in data converter devices is the serial peripheral interface (SPI).

SPI is a synchronous fully duplex interface, which means that both nodes can send data at the same time. SPI is one of the most popular and used communication protocols between controller devices (FPGAs, microcontrollers, etc) and peripheral circuits such as sensors and actuators but also ADCs and DACs [12]. Since there is no standard way of specifying an SPI bus, different vendors use different approaches to accomplish SPI communication.

There are the possibilities of using three or four-wire SPI. Four-wire SPI, which is the most common, uses `CS, SCLK, MOSI` and `MISO` signals. In three-wire protocol, `MISO` and `MOSI` are combined into one bidirectional wire [12] that sends and receives between the nodes. The idle state of chip select signal is high. To activate communication with a peripheral, the controller unit pulls `CS` signal to low. After that, data is read and sent at serial clock rate.

The clock signal can be generated in several ways. Clock polarity (`CPOL`) and clock phase (`CPHA`) are two parameters to consider when working with SPI protocol, the choice of which will affect the order of how data is sampled and shifted out. Table 2.1 describes the different clock polarity and phase combinations.

Table 2.1: SPI modes with serial clock.

SPI mode	CPOL	CPHA	Clock idle state	Data
0	0	0	Low	Data sampled on rising edge. Shifted out on falling edge
1	0	1	Low	Data sampled on falling edge. Shifted out on rising edge
2	1	0	High	Data sampled on rising edge. Shifted out on falling edge
3	1	1	High	Data sampled on falling edge. Shifted out on rising edge

As previously mentioned, there is no standard way of using SPI. However, most of the datasheets for sensors and ADCs use mode 0 with `CPOL` 0 and `CPHA` 0. A possible timing diagram can be found in Figure 2.2. Notice that the first bit in `MOSI` specifies if the controller unit wishes to write or read data to a certain register in the peripheral. Next bit is responsible for indicating data size (few/many bits). The rest of the data stream contains the address and eventually the data that will be sent to the peripheral. If a ”read” signal is read from the peripheral, then the current data

of the specified register will be sent back through `MISO` after successfully reading the address.

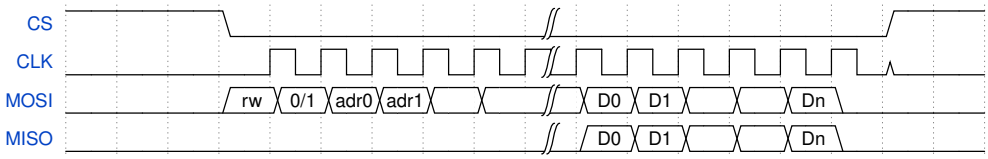


Figure 2.2: SPI timing diagram. CPOL = 0, CPHA = 0.

2.4 AMBA shared bus

Advanced microcontroller bus architecture (AMBA) is an on-chip interconnect specification developed by ARM, used for the management of SoC functional block such as controllers and peripherals [13].

2.4.1 AMBA AHB

Advanced high-performance bus (AHB) is essential in SoC systems. It is designed for high performance synthesizable designs. Designs such as internal memory devices, external memory interfaces and high-bandwidth peripherals are common examples of designs that utilize the AHB protocol [14].

2.4.2 AMBA APB

Advanced peripheral bus (APB), is another protocol which as the name suggest, is a standard used to handle lower-bandwidth peripheral devices on an SoC. Furthermore, this standard is less complex, also, optimized for minimal power consumption compared to AMBA AHB [15].

2.5 GRLIB

GRLIB is a complete IP library containing different reusable VHDL design environments targeting ASICs and different FPGA platforms [9]. Below is a list of the four categories inside GRLIB.

- **Processors**: LEON SPARC, NOEL-V RISC-V
- **Memory Controllers**: DDR2, DDR3, SDRAM, SRAM, NANDflash, QSPI
- **Interfaces**: SpaceWire, SpaceFibre and WizardLink controller, 32-bit PCI bridge, 10/100/1000 Mbit Ethernet MAC, USB 2.0 host and device controllers, CAN, SPI, I2C, UART
- **SoC infrastructure**: AMBA AHB and APB controllers, AMBA bridges

The idea of having an IP core library with several smaller blocks would be to flexibly edit and create different SoCs for different applications as can be seen in Figure 2.3.

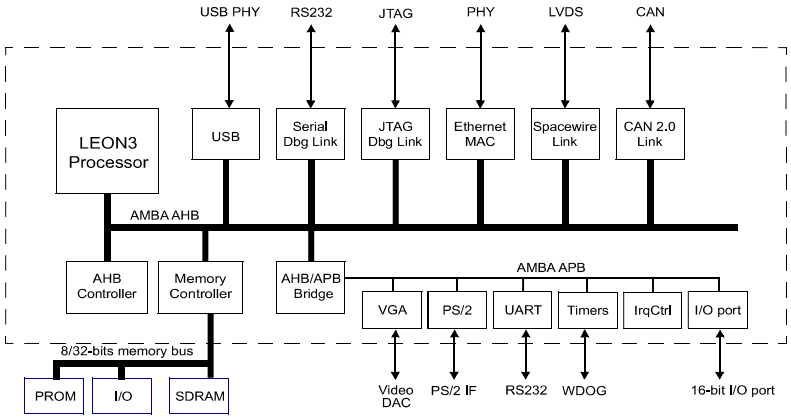


Figure 2.3: Block diagram of the GR-XC3S-1500 FPGA development board [9].

2.6 JESD204 high-speed interface

To keep up with the newest applications, manufacturers of data converters have been forced to continuously increase the sampling rates of their products. Additionally, multiple converter channels are typically used in parallel in order to further increase bandwidth. As a result, existing converter interface methods, such as the Low Voltage Differential Signaling (LVDS) were under considerable pressure to deal with the high bandwidth demands. The transmission speed of LVDS is limited to around 1 Gbps per serial lane, necessitating the use of multiple lanes to achieve high throughput. Multiple converter channels quickly add up the number of required pins on the transmitter and receiver devices, increasing the overall area requirement and complicating PCB routing. With these things in mind, the JEDEC Solid State Technology Association decided 2006 to create the JESD204 standard, which is a a high-speed serial interface standard. Nowadays, JESD204 is a well-known standard, that is widely used across telecommunications, aerospace, and industrial automation when interfacing with converters [7]. Since its launch, the JESD204 standard has seen multiple revisions, shown in Figure 2.4.

Revision	JESD204 (2006)	JESD204A (2008)	JESD204B (2011)	JESD204C (2017)
Max. serial bit rate	3.125 Gbps	3.125 Gbps	12.5 Gbps	32 Gbps
Lanes / links	Single	Multiple	Multiple	Multiple
Deterministic latency	No	No	YES	YES
Encoding scheme	8B/10B	8B/10B	8B/10B	8B/10B, 64b/66B, 64B/80B
Subclasses	No	No	Subclass 0, 1, 2	Subclass 0, 1, 2

Figure 2.4: JESD204 revisions.

Revision A and B

The initial version of JESD204 supports speeds up to 3.125 Gbps [7]. In 2008, revision A of the standard was done in order to support interface with multiple converter channels. Four years later, the standard was upgraded again, to version JESD204B. This revision is divided into subclass 0, 1 and 2, each of which supports an upgraded transfer rate of 12.5 Gbps [16].

Revision C

Another modification to the standard, JESD204C, was launched in 2017. JESD204C offers an even greater bandwidth, up to 32 Gbps. Another focus point of revision C is the improved link resilience, which is made possible by more advanced error detection mechanisms. Also, revision C introduces support for a new and more efficient 64b/66b encoding scheme, compared to the 8b/10b encoding that is used in JESD204B [17].

2.6.1 JESD204B overview

Device clock

In a JESD204B system, the `device clock` is the main timing reference for each layer within a JESD204B transmitter or receiver. Device clocks of different converters and logic devices do not have to be identical, however, they need to be phase aligned [18].

Local multiframe clock (LMFC)

Packets are transmitted over a JESD204B link in multiframes. Multiframes consist of frames, which carries sample data, link configuration data and alignment characters. To synchronize the transmitter and receiver in relation to a multiframe, the JESD204B uses a `LMFC`. The `LMFC` acts as a low frequency cross-device timing reference, allowing for the link to have a deterministic latency (discussed in subsection 2.6.2) [18].

SYSREF

`SYSREF` is an externally applied signal that is active high, and is used in subclass 1 (discussed in subsection 2.6.3) to align the `LMFC` of all transmitter and receiver devices. `SYSREF` could be a one-shot, periodic, or gapped periodic signal [18].

SYNC~

`SYNC`~ is an active low signal used to establish the initial synchronization between a transmitter and receiver. Moreover, it is used in subclass 2 (discussed in subsection 2.6.3) to fulfil deterministic latency. The clocks that generate `SYNC`~ should be phase-locked to the `LMFC` of the receiver device [18].

2.6.2 Deterministic latency

Link latency is typically measured in **`frame clock`** or **`device clock`** periods, and defined in a JESD204 link as the time difference between when a converter sample enters the serial transmitter and when the same sample is output from the serial receiver. When setting up a JESD204 link, it may be required that the link latency is deterministic, meaning that one can reliably determine when the next sample will arrive. The JESD204 standard allows for various ways to achieve deterministic link latency. Other than the link latency, the overall latency of the system would also be dependent on the core latency of the converter device [18].

Elastic buffer

A key module for achieving deterministic latency across all lanes in JESD204 systems is the elastic buffer. Factors such as unequal lengths in the transmission medium or cross-talk, could cause skew in the arrival of the lanes at the receiver. The purpose of an elastic buffer is to align all lanes, so that they can be output from the receiver at the same point in time. Instead of releasing all lanes at the next boundary of **`LMFC`**, the release of the lanes would be delayed by the release buffer delay **`RBD`**, specified by the implementer, to allow for the slower lanes to catch up [18]. An illustration of the elastic buffer is depicted in Figure 2.5.

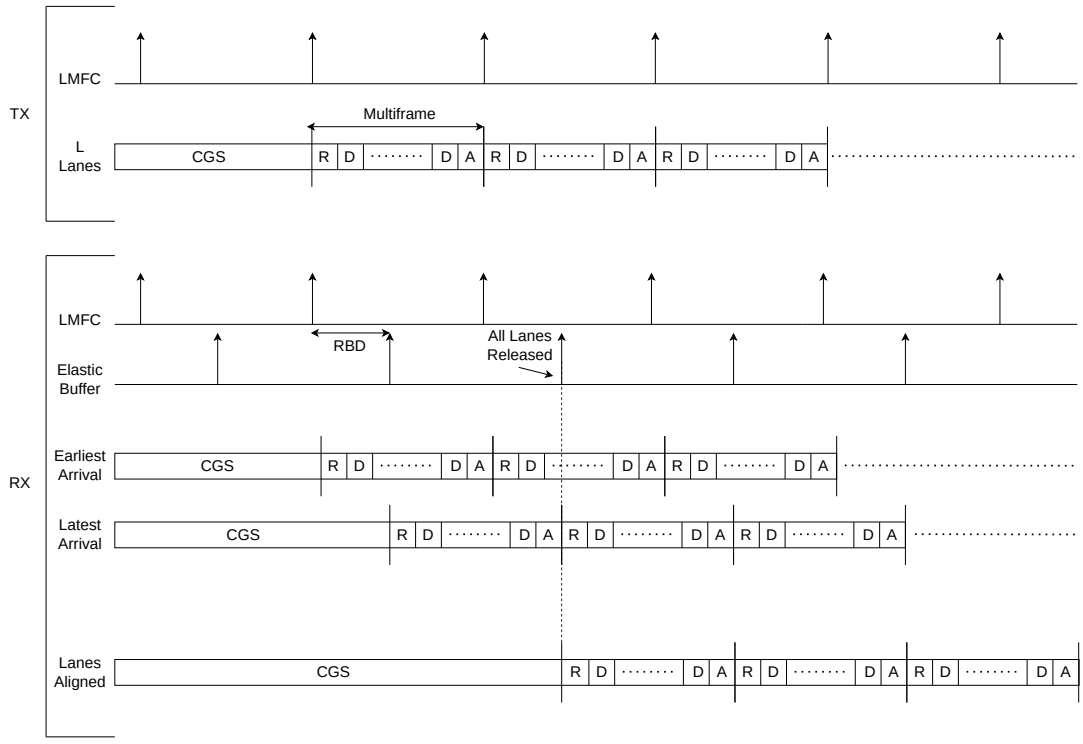


Figure 2.5: Timing diagram illustrating the elastic buffer.

2.6.3 JESD204B subclasses

Subclass 0

Subclass 0 supports backwards compatibility with JESD204A. No additional signals are introduced in subclass 0, which makes it the relatively simple to implement, compared to the other subclasses [7].

Subclass 1

Out of the three classes, subclass 1 offers the most stable performance at very high device clock rates. Subclass 1 is not backwards compatible with JESD204A. It does however offer support for deterministic latency. To achieve deterministic latency, the subclass uses an external **SYSREF** signal. The purpose of **SYSREF** is to align the **LMFC** of the transmitter devices and the receiver device [18].

Subclass 2

Subclass 2 works well with device clock rates up to 500 MHz. The class does not support compatibility with JESD204A. Similar to subclass 1, subclass 2 also allows for deterministic latency. Subclass 2 however, uses the **SYNC**$\sim$ signal for **LMFC** alignment. In contrast to the external **SYSREF** signal, **SYNC**$\sim$ is already used for code group synchronization (discussed in subsection 2.7.4) and frame synchronization, meaning that fewer pins are needed on the transmitter and receiver [18].

2.7 JESD204B layers

The JESD204B is a standard that is built using several layers, each designed with distinct functionalities. The layers remain the same whether implemented in a receiver or in a transmitter, differing only in executing the opposite operations. The layers used in a JESD204 are as follows: Physical layer, transport layer, data link layer and an application layer that is unique for every implementation. The layers within the JESD204 standard follow a similar layout to the OSI-model, described in [19].

2.7.1 Physical layer

The physical layer's primary role is to interact with the other components outside the JESD204 standard. It is where the data is serialized/deserialized using (SerDes) at line rate speed. For a transmitter, the physical layer takes the parallel data frame and sends it as a serial stream using a serializer. The serial output is sent at Gbps rates. For the receiver part, the physical layer captures the data and uses it to recover the serial clock, to later deserialize data from Gbps to Mbps for it to be handled by an FPGA. The deserialized data in the receiver becomes parallel and spread across a number of output pins [20]. Additionally, a SerDes contains advanced yet crucial phased locked loop (PLL) and clock data recovery (CDR) circuitry. CDR enables clock recovery from the incoming data, ensuring proper synchronization and jitter

reduction. Meanwhile, the PLL is used for high-frequency clock used for serialization, ensuring phase alignment. In Figure 2.6, a block diagram of the physical layer of the JESD204 standard is presented.

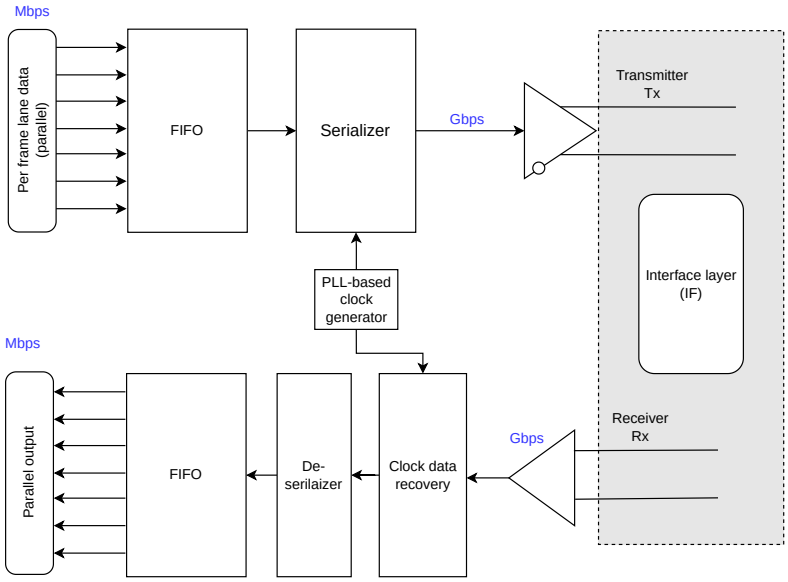


Figure 2.6: Block diagram of a serializer and deserializer for the physical layer.

2.7.2 Transport layer

The transport layer is the layer responsible for data mapping in the JESD204 standard. Data gets reframed and prepared before being sent from transmitter to receiver. All control bits and tail bits are added to the data stream. They are later split into octets and lanes. It is important to note that a transport layer delivers the samples to the link layer in a JESD204 transmitter. The transport layer in the receiver does the exact opposite. It cleans the received data from all control bits and tail bits to get the original content of the data stream [17]. Below is a list of parameters needed for a reader to understand the transport layer functionality.

- M = Number of converters
- N = Converter resolution
- S = Sample per frame
- F_C = Sampling frequency
- C_S = Control bits per sample per lane
- N' = Number of bits per sample in user data format
- L = Number of Lanes
- F = Number of octets "8 bits" per frame
- K = Frames per multiframe

Designers may want to configure the transmitter and receiver setup to match their implementation requirements. Figure 2.7 shows what parameters need to be defined for a transport layer to reframe data.

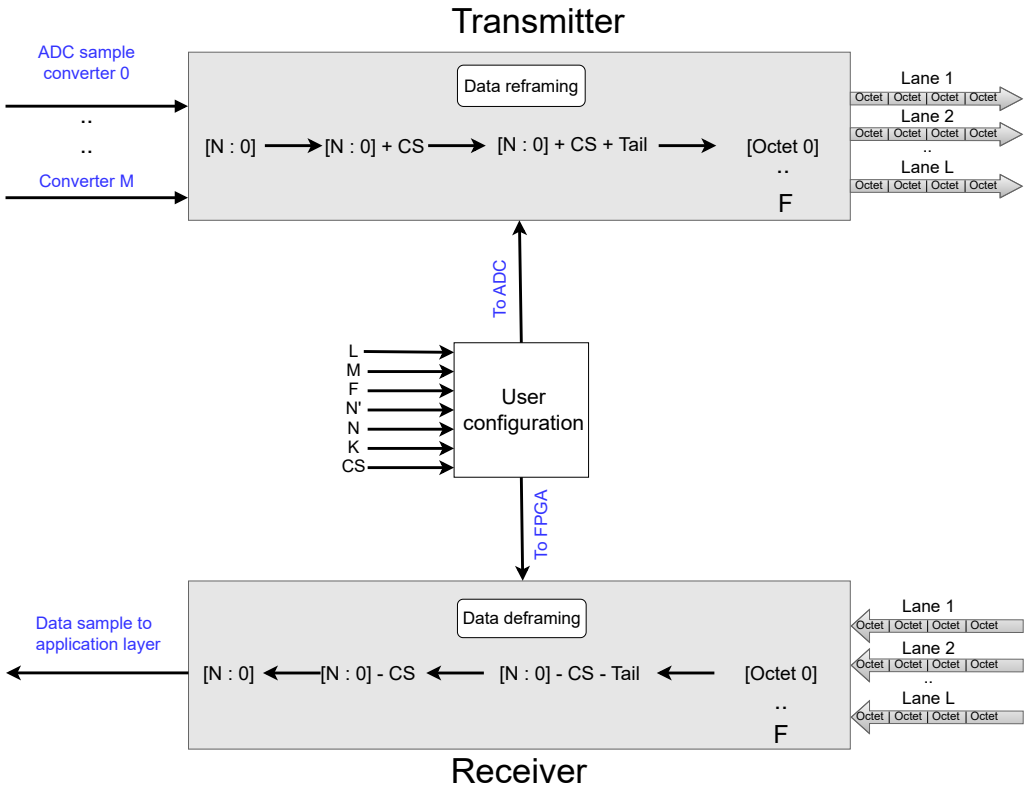


Figure 2.7: Design of a transport layer in Rx (FPGA) and Tx (ADC).

Some of the parameters are hardware dependent, for instance FPGA maximum clock, or number of converters in the ADC as well as the number of bits in the ADC. However, by using a high-speed serial link with SerDes, data can be sent split into multilane links. This is specially handy for GSPS ADCs. Therefore, designers can increase the data rate and use multiple lanes to catch up with the high speed of the link. Hence, the number of lanes and octets needs to be calculated for each JESD204 implementation. Equations 2.1 and 2.2 which can be found in [21] show how to calculate the number of needed lanes L.

$$\delta = M \cdot N' \cdot S \cdot 1.25 \cdot F_C \tag{2.1}$$

$$L = \frac{\delta}{\text{LaneRate}} \tag{2.2}$$

From this point the number of octets per frame F can be calculated. To determine this parameter, equation 2.3 can be used.

$$F = \frac{M \cdot S \cdot N'}{8 \cdot L} \tag{2.3}$$

Taking all these parameters into account, transmitter user data can be formatted in different ways. Multiple lanes, line, oversampling or the number of octets can affect how data formation looks like. Generally, one device with M converters produces N bits per sample. A set of samples contains F octets. Each octet is transmitted as a group of N' bits, where N' is N bits plus any additional tail bits T and control

bits C_S. The numbering starts in order with converter 0, followed by converter 1 up to converter M-1 as can be seen in Figure 2.8. A new word is created for each converter, containing data identical to the corresponding. Control bits and tail bits are yet to be added. At the next stage, additional bits are added to fill up empty slots when forming the octets.

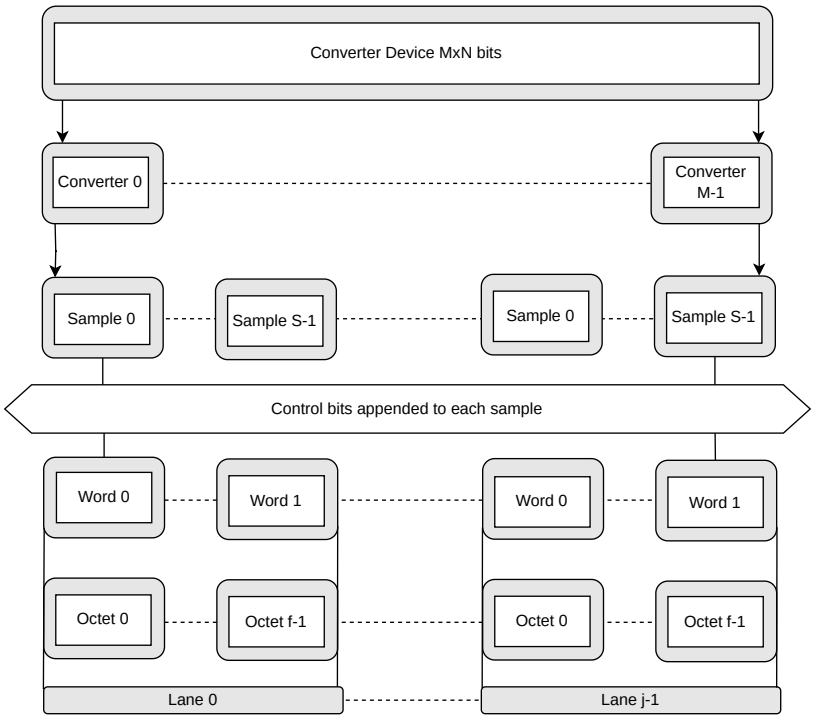


Figure 2.8: User data format for general use. This can be used with values as low as $L = 1$, $S = 1$ and $M = 1$, inspired from the JESD204C standard [17].

The figure shown above describes the structure of the transport layer, which is applicable for any type of JESD204 implementation. For example, a device with $M = 4$, $N = 12$, $S = 1$, $L = 1$ can be used, as illustrated in Figure 2.9. Data coming from the four converters is mapped into one single lane, following the approach previously seen in Figure 2.8. Note that the upper 8 bits of the data are getting mapped to the first octet, and the last 4 bits are mapped to the second octet. The second octet of each converter is eventually filled up with one control bit and three tail bits.

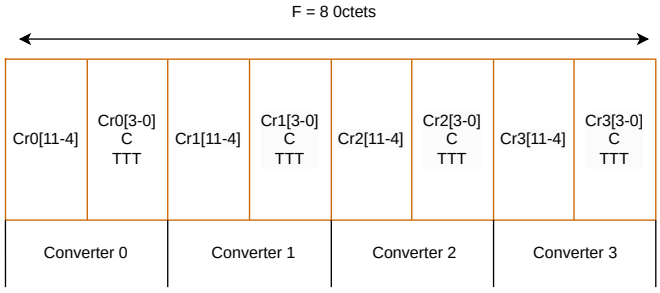


Figure 2.9: Grouping control bits and tail bits to octets and converters.

2.7.3 Scrambling

Scrambling is an optional block in the JESD204 standard. It is mainly used to avoid spectral peaks that would occur by specific data patterns that may affect the performance of the serial link [18]. Spectral peaks may affect the link by means of EMI in sensitive applications, DC balance and signal integrity.

Scrambling is issued deterministically. The standard implies that the scrambling polynomial shall be:

$$1 + x^{14} + x^{15}$$

Introducing scrambling means that Hex values for alignment characters will be different depending on if scrambling is used or not.

2.7.4 Data link layer

Data link layer is a crucial layer where synchronization, data encoding, error detection and general flow control happens in the standard. It can easiest be described as the controller due to its variety of tasks.

8b/10b encoding

Dealing with serial high-speed signals introduce some noteworthy challenges comparably to traditional serial interfaces such as SPI, UART or I2C. One of these challenges is the DC biasing, which leads to signal integrity and metastability issues [22]. This occurs due to the count imbalance between 1's and 0's of the data sent across the link. By having this imbalance, the cable charges up if there are many consecutive ones in the data stream, leading to uncertainty in reading an incoming zero signal.

Consequently, the goal of ensuring a relatively constant average voltage is required for signal integrity. Encoders and decoders can be used to accomplish a balanced data stream. By following a look up table (LUT), the encoder maps all possible outcomes of the data stream into corresponding encoded values. Some input values have two valid codes. To determine which one to use, a running disparity (`RD`) checker between ones and zeroes is used as a signal in the algorithm. This signal acts as an input to the next code encoding to reverse disparity and maintain an overall DC balance if needed [22].

8b/10b encoding is used in the JESD204 standard to encode data before being transmitted. As the name suggests, it encodes an 8-bit value into 10-bits. In Figure 2.10, we illustrate a typical implementation of the 8b/10b encoding algorithm.

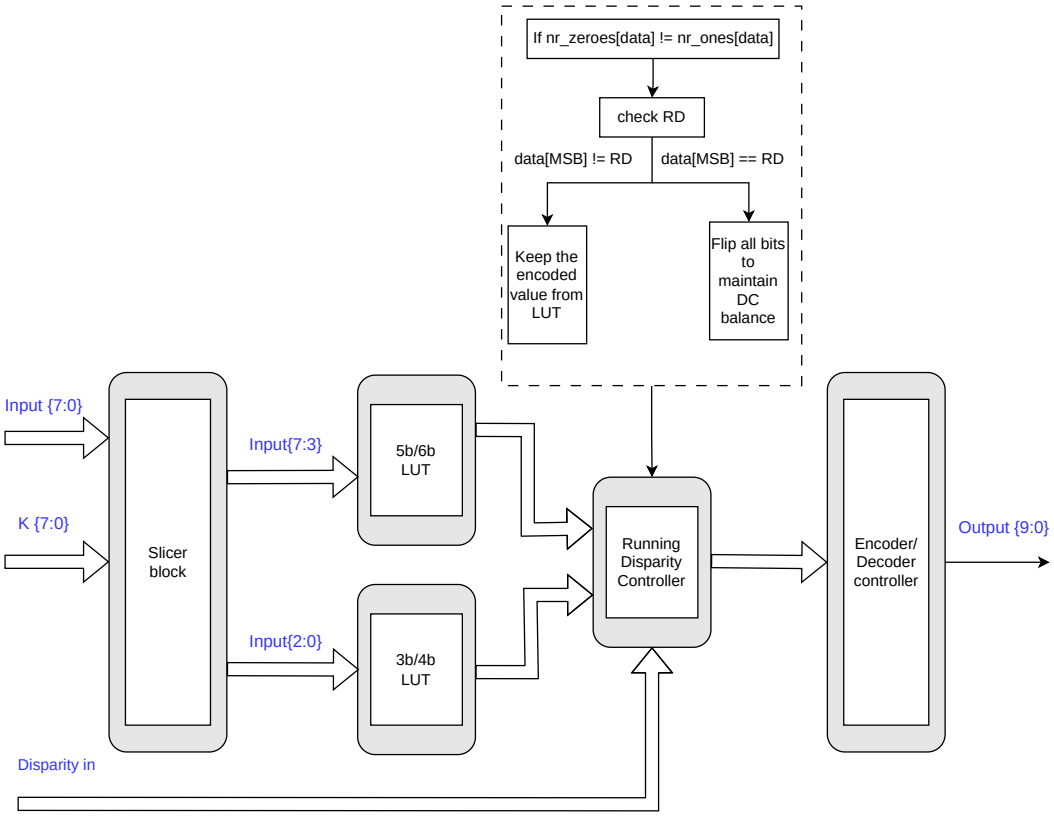


Figure 2.10: Block design of the 8b/10b encoding.

While 8b/10b encoding introduces 25% overhead in terms of additional bits, the benefits still outweigh the drawback. An achieved DC balance, and therefore reduced electromagnetic interference (EMI), as well as introduced error detection and correction mechanisms for increased reliability and integrity, are the benefits of this encoding algorithm [22]. The 8-bit input is spitted into two encoders, a 3b/4b encoder as well as a 5b/6b encoder, that gets merged back again at the end with LSB first, see Figure. 2.11 for clarification.

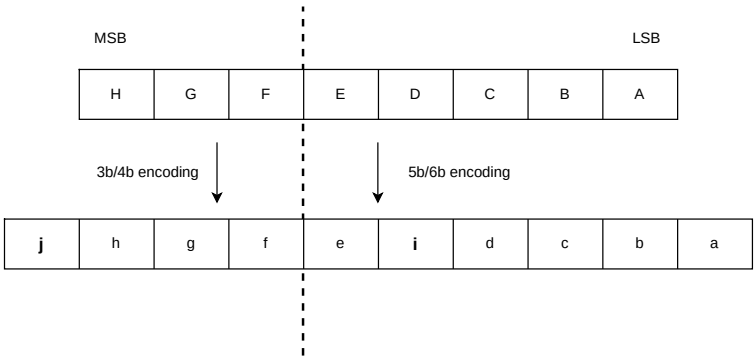


Figure 2.11: 8b/10b conversion.

The encoded output has three states. It can either have disparity zero, meaning

equal number of 1's and 0's. A positive disparity indicates more ones than zeroes, and vice versa to obtain negative disparity. In Table 2.2, a few examples of each encoding LUT is found. Note that the `RD` signal is the determining factor for inputs with two valid codes.

Table 2.2: Encoding table examples for 5b/6b on the left, and 3b/4b on the right.

input		RD= -1	RD = +1
code	EDCBA	abcdei	
D.01	00001	011101	100010
D.03	00011	110001	
D.28	11100	001110	

Input		RD = -1	RD = +1
code	HGF	fghj	
D.01	001	1001	
D.03	011	1100	0011
D.06	110	0110	

As part of the 256 possible data characters, the 8b/10b encoding reserve some characters as so called special comma characters, that are used for control ($\mathsf{K}x.y$) . Here the x represents the 5-bit group, and the y value corresponds to the 3-bit group. Table 2.3 describes the important comma characters used in the JESD204 standard.

Table 2.3: Comma characters for special control.

/K/	k28.5	code group synchronization
/R/	K28.0	Start of multiframe
/Q/	K28.4	Start of configuration data
/A/	K28.3	End of multiframe
/F/	K28.7	Frame alignment synchronization

As described before, error detection and correction comes automatically when using the 8b/10b encoding algorithm. Using the control characters and having the ability to check the LUTs for an expected encoded value, designers have the ability for increased reliability and check for

- Disparity errors.
- Not in table code errors.
- Wrong control character/control character in wrong position.
- Code group synchronization error.

Code group synchronization (CGS)

Code Group Synchronization (CGS) is a crucial operation in the JESD204 standard. When a JESD204 receiver is ready to collect data, it issues a synchronization request by asserting the `SYNC`$\sim$ signal to the transmitter. Immediately upon reading the `SYNC`$\sim$ signal from the receiver, the transmitter transmits a number of /K/ characters through the data stream. When the receiver has read the required amount of /K/ characters, `SYNC`$\sim$ gets deasserted again, indicating a synchronized transmitter and receiver. Figure 2.12 visualizes the CGS process written in the JESD204B standard. [18]

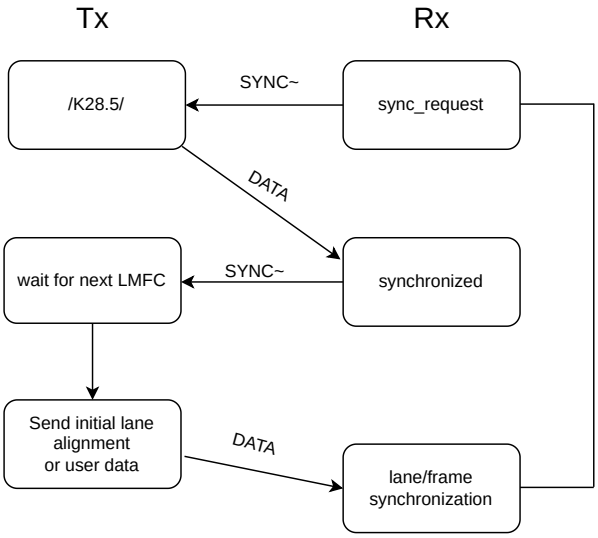


Figure 2.12: State machine of CGS.

Initial lane alignment (ILA)

The CGS and the initial frame synchronization (discussed in section 4.2) are succeeded by the ILA. The purpose of ILA is to synchronize the lanes of a multi-lane link. The format of the ILA is shown in Figure 2.13. The length of the ILA is always four multiframes long, where the second multiframe will include link configuration data [18].

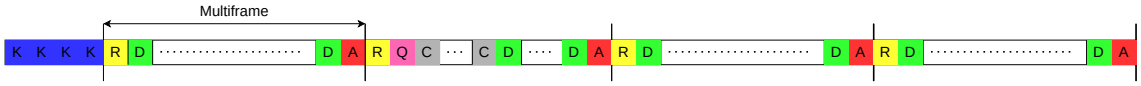


Figure 2.13: Initial Lane Alignment.

Frame alignment and correction

Apart from the previously discussed /K/ characters, there are also various alignment characters that are inserted into the serial data stream by the transmitter under certain conditions [18]. The /R/ character is used to mark the beginning of a multiframe, and /A/ is used to mark the end of a multiframe. /A/ and /R/ characters are always embedded in the ILA, to align the multiframes. A new frame starts once every F octets. The /A/ character is also inserted into the serial data stream by the transmitter, if the last octet of the last frame in a data multiframe, is equal to the last octet of the previous frame. In this case, the /A/ character would replace the octet in the last frame of the multiframe. This would happen even if the last octet of the previous frame was an /A/ character as specified in the standard [18].

According to the same standard, the /Q/ character is used exclusively during the ILA, and will mark the beginning of the receiver parameter data. The receiver parameter

data is always placed in the second multiframe of the ILA.

Additionally, there is an /F/ character. This character is used similarly to the /A/ character, however, only on the last octet of the current frame, which is not the last frame of the multiframe. Another key difference from the /A/ character is that for the /F/ character, if the last octet from the previous frame was an /F/ character, the octet replacement would not be performed.

2.7.5 Application layer

The application layer is the highest level of abstraction in the JESD204 protocol. It is the layer responsible for further processing the received data stream from previous JESD204 layers [18]. In the case of this master thesis, the application layer consists of the data reduction algorithm, where reduced data size is the goal before further processing or storage.

2.8 Data Reduction

Data reduction is the process of reducing the amount of information to make it easier to store in terms of physical space. Technology in the 1800s had yet to be developed to transmit human voice over long distances. Engineers could nevertheless transmit short electric signals through wires. Morse code was one of first methods used for long-distance communication, using an encoding scheme to encode letters into dashes and dots. Morse code is viewed as one of the earliest communication methods, which uses a form of data compression [23]. This allowed telegraph operators to compress human language to binary format and restore them to full format at the other station.

In the current age of computers, data compression is used increasingly, and can be found on most of the files that can be found on modern computers [23]. JPEG, PNG, GIF, MP3 and MP4 are examples of file formats commonly used on a daily basis, that apply various data compression algorithms.

Data compression algorithms like the above mentioned are divided into two primary categories: lossy and lossless. While some experts in the computer science filed indicate that it is mathematically impossible to have a totally lossless algorithm, lossless algorithms show no noticeable change in the data over time [23]. Lossless algorithms transform the input data into a more compact representation in a different format, ensuring efficient storage. PDF, SVG, ZIP and GZIP are examples of lossless compression algorithms using techniques from Lempel-Ziv and Huffman to primary reduce redundancy on the input data [23] [24].

Lossy compression is the process of reducing input size by permanently removing some information. The size of the output file produced by such algorithms is significantly compact. However, data quality and purity may be affected and downgraded over time. MP3, MP4 and JPEG are examples of lossy algorithms. Such algorithms can be used on the hardware side if area and power dissipation are two major considerations.

2.8.1 Run-length encoding

Run-length encoding (RLE) is a type of lossless data compression in which data with redundant occurrences are stored as a single data value and count, rather than its original sequence. This algorithm is most useful when data is repeated inside the data stream, hence producing high compression ratio [24]. The compression ratio for varying input data should therefore be minimal.

RLE can be used in various applications, like image and video compression. It can be used to minimize the size of the incoming image matrix. The algorithm would read the binary pixels of each row and group them if a sequence of redundant values is located. Therefore, RLE is a well suited for hardware implementation due to the fact that it doesn't require complex logic or large storage. Only a number of logic gates and flip flops to perform the counting sequence are required. Figure 2.14 shows how RLE can be implemented to compress binary images.

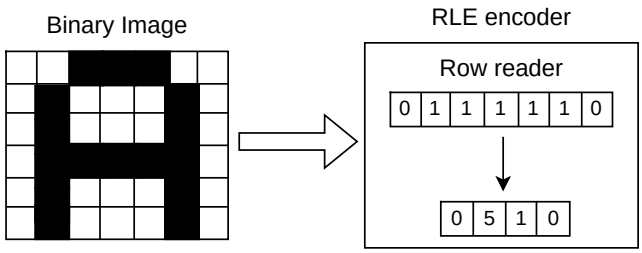


Figure 2.14: Block diagram of RLE algorithm used on binary image compression.

2.8.2 Moving average

While not traditionally known to be a data compression algorithm, moving average is a statistical model that is used to calculate the average of a fixed-sized window of samples. Therefore, moving average can be seen as a lossy data compression algorithm. In digital signal processing, moving average is often used to smoothen out noise or fluctuations in time series data [25].

The implementation of moving average on hardware can be very cheap depending on the desired accuracy. For low accuracy applications, a shift register with a small window size and the simple arithmetic operation as seen in equation 2.4 are required. If higher accuracy is needed then a larger shift register is required, which is costly on hardware in terms of area, due to the linear scaling.

$$y[n] = \frac{1}{N} \sum_{i=0}^{N-1} X[n-i] \tag{2.4}$$

In the moving average formula, $y[n]$ represents the moving average at n-th point in time. The input sample is symbolized by $x[n]$, and N represents the sample window.

3

Methods

The method to be used in this project are divided into the following six sections: literature study, define data acquisition use case, designing the receiver, creation of a subsystem, verification through simulation, and, finally hardware verification.

3.1 Literature study

In order to set the specification requirements of a JESD204 receiver, literature study of the different versions provided by JEDEC is required. Such study looks into the evolution and the differences between the various JESD204 versions. The main goal behind this study is for us to gain knowledge about the pros and cons of each version implementation, to later be developed on a Gaisler's platform.

Another source of valuable information can be found by studying academic papers, industry publications and official documentations of hardware supporting the JESD204 standard.

3.2 Define data acquisition use case

To be able to understand the needs of a JESD204 receiver, we need to engage with the company to get insight of their specific requirements and use cases for a JESD204 implementation is needed. This can also include discussions with customers and the need to get external data for simulation of the receiver, as well as obtaining a sensor for reading real input data to test on hardware implementation.

3.3 Designing the receiver

Designing the receiver will start by evaluating the above mentioned JESD204 versions, and selecting the most suitable version based on the company's use case. Thereafter, we should define the specification requirements of the selected standard, by the means of data rate, number of converters, lane configurations, clocking alternatives and protocol features.

RTL coding guidelines

All RTL modules designed within this project follow the ”two-process method”, advocated by Jiri Gaisler [26]. The two-process method is appropriate for any single-clock design, where the aim of the method is to improve readability of complex designs. Additionally, simulation time is improved by optimizing the use of variables over signals within VHDL processes, as variable assignments are significantly faster than signal assignments [26].

Designs following the two-process method use only two processes per entity. The purpose of using two processes is to separate the logic from the registers. One process is fully combinational, and contains all the non-synchronous logic, and the other process is clocked, and contains all registers. Signals and ports are declared using record types, which simplifies maintainability. A block schematic of the two-process method is shown is Figure 3.1.

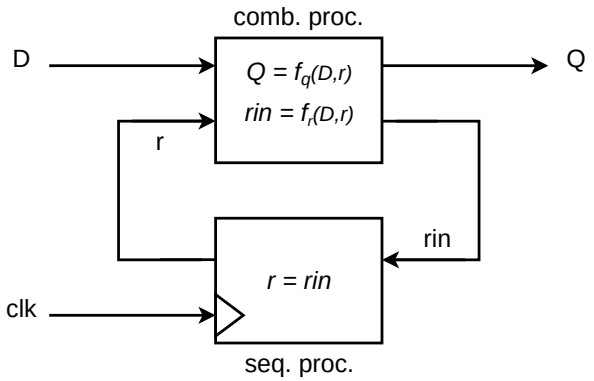


Figure 3.1: Jiri Gaisler’s two process method.

3.4 Creation of a subsystem

GRLIB is a large set of IP-cores. Some IP-cores require larger hardware utilization, and therefore, make the EDA tools run significantly slower. This means that developing the receiver on a complete SoC system that contains blocks such as LEON5 processor, Ethernet and the Xilinx memory interface, might not be an efficient approach. Instead, this project might be its own minimal SoC that runs on the same AMBA shared bus, but with a reduced number of IP blocks.

3.5 Verification through simulation

The company uses VHDL exclusively for RTL writing. Simulation of the RTL code is done using Mentor modelsim 10.6a, making it easier to spot typing and behavioral errors at early stages of the design. Due to the large folder hierarchy of a SoC, makefiles are used to compile and verify different parts of the system.

3.6 Hardware verification

The designed subsystem is to be synthesized and implemented on an FPGA using Xilinx Vivado 2019.2, to later be connected to an ADC to read incoming data. A receiver's foundational part is the physical layer, which serves as a port for incoming data. It is worth planning a robust infrastructure on how verification of the hardware needs to be carried out. Oscilloscope and logic analyzers can be used to measure the incoming clocks. Additionally, a signal generator might come in handy to give stimuli inputs to the ADC.

GRMON

Finally, there is Gaisler Research monitor (GRMON), which is a Gaisler-made debug environment used for SoC designed based on the GRLIB IP library [27]. GRMON facilitates read and write accesses to internal registers and memory blocks of IP-cores, which are interconnected via the APB and AHB bus architectures. GRMON can be used via a tcl environment or using a GUI, with supported debug interfaces via JTAG, UART and Ethernet.

4

Design and Implementation

In this chapter, the system is introduced in detail. We will discuss the design decisions made for the receiver and the data reduction algorithm. Furthermore, we describe the hardware components used in the project. Finally, specifications of the key parameters in the system are presented. Figure 4.1 shows a simple overview of the design. Employing a separate subsystem for the receiver IP development offers many advantages over working from a complete SoC. As stated previously in the method chapter, some IP-cores are large, and extend the synthesis time of the EDA tools. Additionally, creating the receiver inside a subsystem minimizes the likelihood of introducing unexpected errors associated with integrating several blocks into one design.

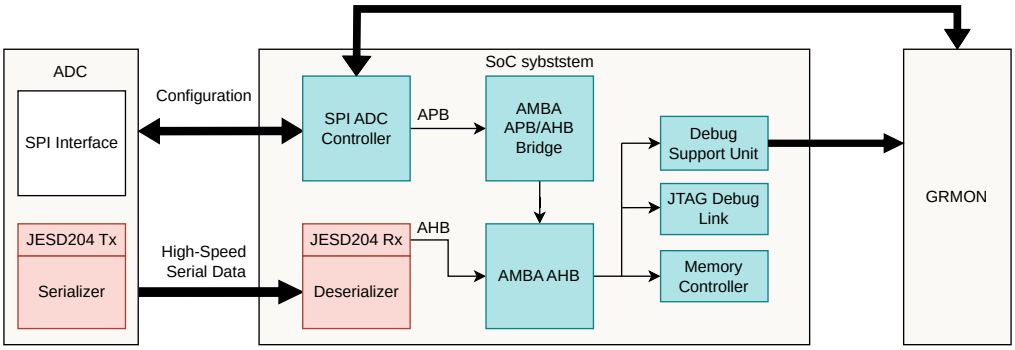


Figure 4.1: Simple overview of the designed subsystem on FPGA.

4.1 Design overview

In this section, a more sophisticated overview of the JESD204 standard and hardware selection is mentioned. We will examine each component of the minimally designed SoC, referred to as the subsystem, in greater detail.

4.1.1 Choice of the JESD204 version

After studying the different revisions of the JESD204 protocol and the infrastructure of GRLIB, we chose revision JESD204B. One reason for that is the similarity with the existing high-speed serial link IP provided by Gaisler, called GRHSSL [28]. Additionally, the selected version had to be compatible with the JESD204 transmitter of the ADC, so the receiver IP can be verified together with a third party transmitter. Revision C supports faster lane rates and uses the more effective 64b/66b encoding compared to the 8b/10b encoding in revision B, which introduces 25% overhead. However, SpaceFibre inside GRHSSL uses 8b/10b encoding to maintain running disparity, and since it is the default decoding scheme for revision B, an encoding scheme does not have to be implemented from scratch. Additionally, revision B has been around for longer than revision C, meaning that documentation and previous implementations are more accessible for B, compared to C. This is also visible in the market for data converters, where a clear majority applies JESD204B [29]. Advantages for B over A includes higher data rates and the support for deterministic latency.

4.1.2 Hardware selection and setup

A very important factor when selecting the hardware, was to acquire compatible hardware with reasonable delivery time. The FPGA used within this project is the Kintex UltraScale XCKU040 on the KCU105 board, from Xilinx [30]. This was a straightforward choice for us because of two reasons: the availability of that card at the company's inventory, but most essentially the GRHSSL IP was developed and implemented on that same FPGA, which has been previously verified to achieve a successful SerDes communication up to 6.25 Gbps.

The ADC evaluation board selected is the ADC32J44EVM from Texas Instruments [31]. The ADC sample rate needed to be sufficient for the onboard serializer to be able to generate a serial lane rate of 3.2 Gbps. The system designed includes two converters, indicating the use of a dual port configuration. This setup allows for a larger set of debugging opportunities, but also enables further development using multi-lanes and oversampling. A critical factor is the generation of the data. The evaluation board has test patterns stored on the memory chip, which can be routed to the JESD204B transmitter on the chip, eliminating the need to insert data through the analog channels.

In Figure 4.2, an overview of the hardware setup is presented. Note that the data stream is moving at speeds of Gbps, meaning that some verification infrastructure needs to be developed, in order to verify that ADC data is being fetched on the

FPGA's physical layer (SerDes). The design methodology seen in the figure below, shows a FIFO module, underneath the SerDes. This FIFO stores the incoming high speed data and provides an AMBA APB interface, which allows us to monitor the data via the GRMON. The main purpose of the so called **APBFIFO** is to act as a buffer between the slower read-rate of the FPGA compared to the faster write-rate of the SerDes data. Being able to read actual ADC data via the **APBFIFO** , indicates a functional JESD204B physical layer.

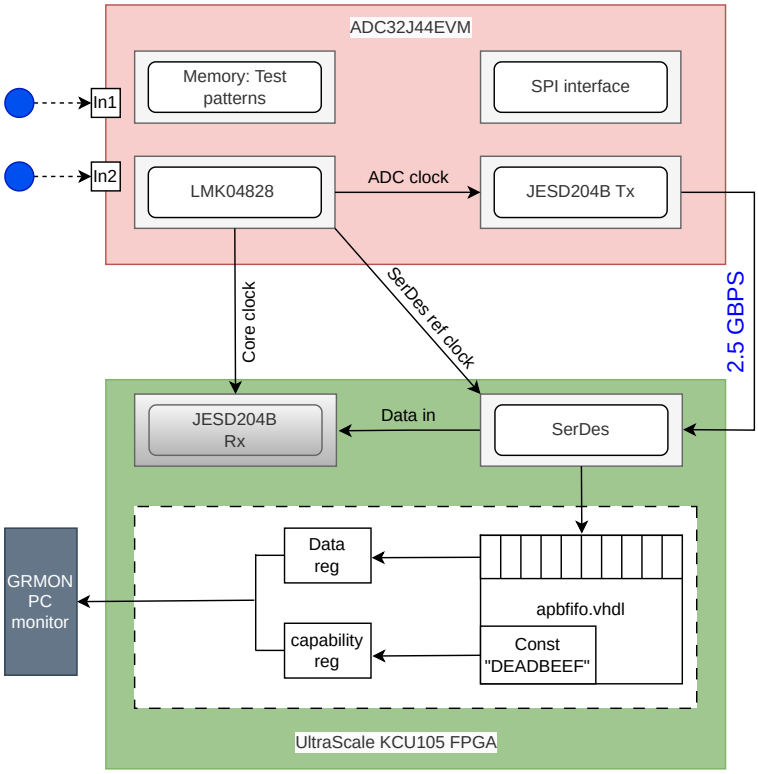


Figure 4.2: Overview of hardware setup.

4.2 JESD204B receiver operation

As previously discussed, the trigger for link synchronization is the **SYNC**~ assertion. According to the standard [18], the CGS on the receiver side is divided into three states, shown in Figure 4.3. In the initialization state, the receiver checks for the recognition of four /K/ characters. The receiver would then enter the check state, where it has to detect four new 8b/10b characters, before it enters normal operation mode. If the receiver detects an illegal character, it will remain in the check state until four valid characters are detected. In the case of identifying three invalid characters, the receiver has to re-assert the **SYNC**~ signal to re-initialize synchronization.

Once **SYNC**~ is deasserted, the initial frame synchronization will align the edge of the frame with the next non /K/ character. The process of the initial frame synchronization is presented in Figure 4.4. Inside the FS_DATA state, after it detects the ILA sequence, the receiver performs alignment checking on the incoming

multiframes, as explained in subsection 2.7.4. In the event of a synchronization error, a **SYNC**∼ assertion or the reception of four consecutive /K/ characters, the receiver will return to CS_INIT and FS_INIT, and launch a new CGS.

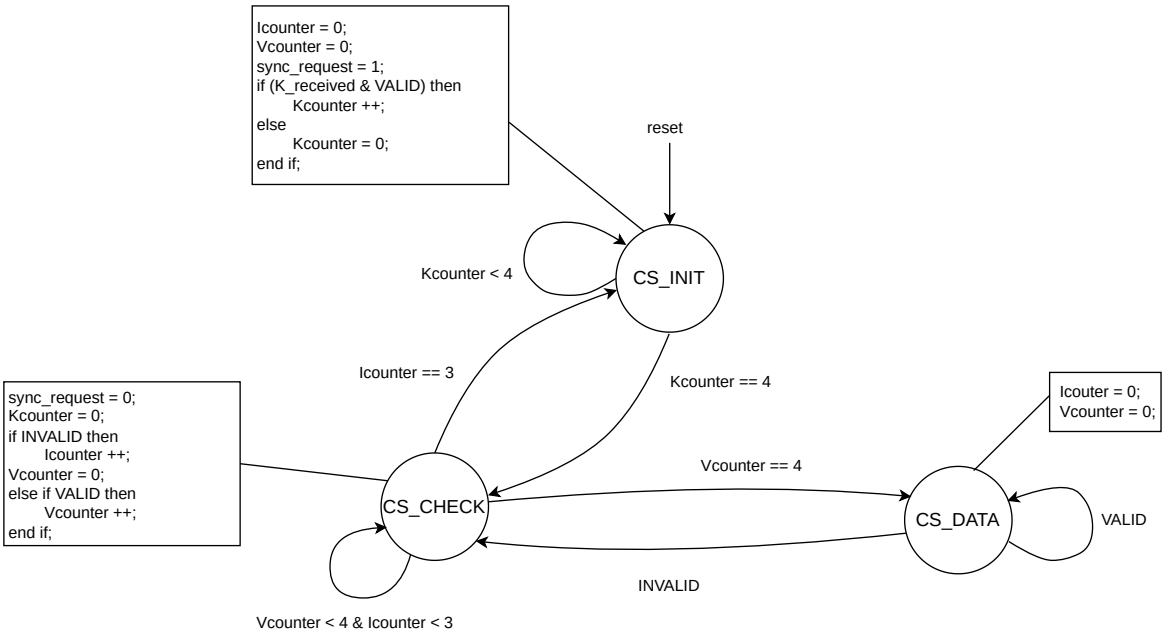


Figure 4.3: Code Group Synchronization FSM.

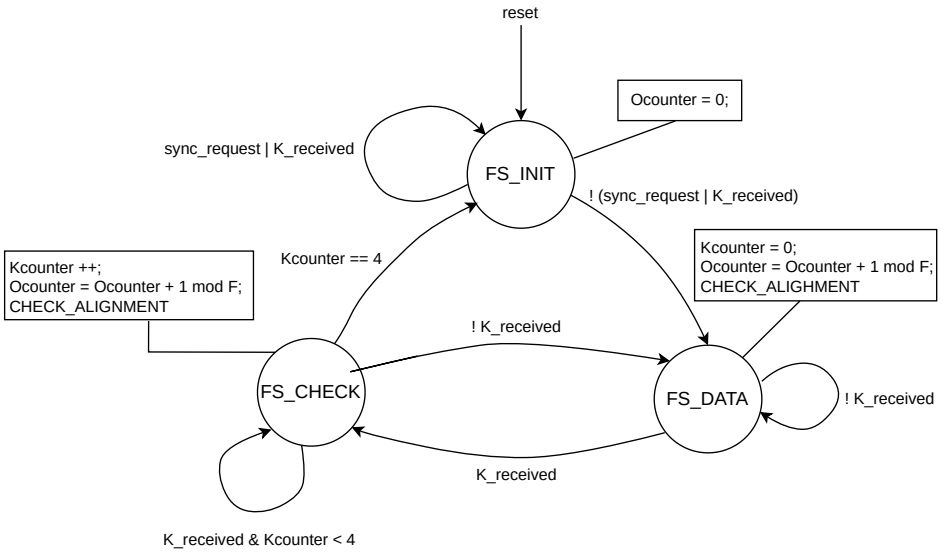


Figure 4.4: Initial frame synchronization FSM.

4.3 JESD204B receiver design overview

Figure 4.5 shows an overview of the JESD204B receiver design. As previously explained, the receiver is divided into three layers. The first layer deserializes the high-speed data, and routes the data in parallel at a lower rate to the link layer. The link layer checks for alignment characters, and reports back to the transmitter via the **SYNC**~ interface, if any errors are detected. The **SYNC**~ interface is also used to synchronize the transmitter with the receiver, and allows for deterministic latency. Valid data is then mapped from the link layer to the transport layer, where control bits and tail bits are extracted from the octets, and the original sample data is then ready for use in an application. Scrambler is not enabled in this project. The reason for this is that the selected ADC board operates with a JESD204B throughput at lower bit rates compared to the maximum bit rate specified in the standard. This means that spectral peaks occurring due to higher frequency are not an issue, that otherwise must be handled by a scrambler.

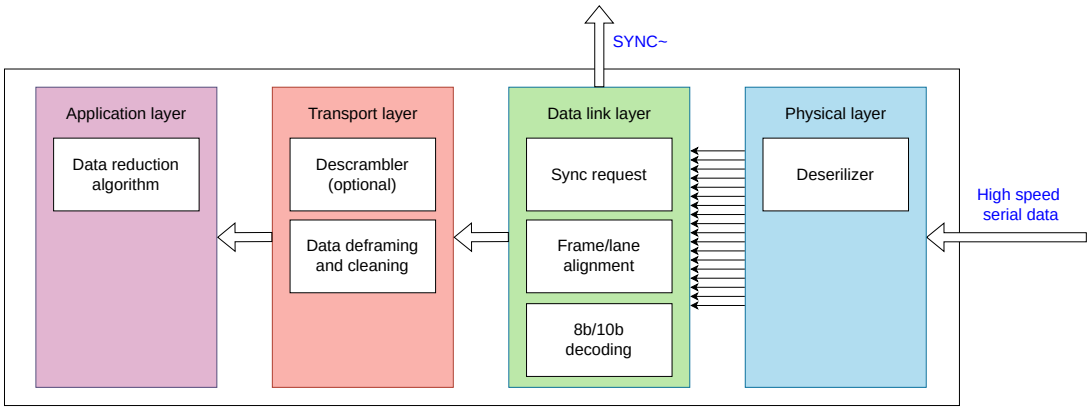


Figure 4.5: Layers of the JESD204 receiver.

4.3.1 Link parameter configuration

The standard supports multi-lane and multi-sample setup. For simplicity, we will proceed with one lane and one sample per frame cycle. The selected JESD204B ADC family supports two link configuration setups, presented in Table 4.1.

Table 4.1: Link parameters and interface rates.

L	M	F	S	Min. ADC sampling rate (MSPS)	Min. SerDes frequency (Mbps)	Max. ADC sampling rate (MSPS)	Max. SerDes frequency (Gbps)	Serialization factor
2	2	2	1	16	300	160	3.2	20X
1	2	4	1	10	400	80	3.2	40X

Given that the model chosen for the above mentioned ADC family provides a maximum sampling frequency of 125 MSPS, this results in a maximum sampling rate of 125 and 62.5 respectively for the two different configurations. The second setup

from Table 4.1 was selected, yielding a serial lane rate of 2.5 Gbps, according to equations 2.1 and 2.2. Below, a calculation of the expected lane rate is presented.

$$\frac{2 \cdot 16 \cdot 1 \cdot 1.25 \cdot 62.5 \cdot 10^6}{1} = 2.5 \text{ Gbps} \tag{4.1}$$

Octets per frame (F) is calculated using Equation 2.3, which resulted in 4 octets per frame, to be able to uphold a lane rate of 2.5 Gbps, for our link. Calculation is shown below.

$$\frac{2 \cdot 1 \cdot 16}{8 \cdot 1} = 4 \text{ octets/frame} \tag{4.2}$$

A lane rate of 2.5 Gbps is below the limit of 6.25, which is the maximum frequency that the SpaceFibre has been tested with on the UltraScale FPGA. Deterministic latency between transmitter and receiver is to be established via the `SYNC~` signal, following subclass 2 of the standard.

4.3.2 High-speed SerDes IP

A proper physical layer is essential to be able to establish efficient data transmission between the FPGA and the ADC. Dealing with high-speed data introduces jitter, synchronization and phase alignment issues, which affects signal integrity. Therefore, PLL and CDR circuits are used in the SerDes architecture as previously mentioned. The UltraScale FPGAs Transceiver IP from Xilinx, is an IP that utilizes the on-chip transceiver on the intended FPGA used in this project [32]. The IP is quite configurable to support various industry standards that require a physical layer. However, having this large set of variation means that configuring and establishing the SerDes involves some trial and error.

4.3.3 Design of link layer

Our implemented link layer module follows a state transition flow, similar to Figure 4.6. The purpose of the link layer is to pass the incoming octets to the transport layer, and maintain link synchronization between the receiver device and the transmitter device. The link layer consists of four states; idle, cgs, ila and data. In the cgs state, the receiver will look for four consecutive /K/ characters before it enters ila. In ila, the receiver will scan every multiframe for start and end characters, fetch possible configuration data in the second multiframe, and proceed to the data state if all multiframes have valid start and end characters. If the receiver at any point fails to detect valid start or end characters, we issue a re-synchronization request, and re-enter the idle state. Otherwise, after a successful reception of four multiframes at correct positions, the receiver enters the data state, where it will remain for as long as there are no alignment errors, and no re-synchronization requests.

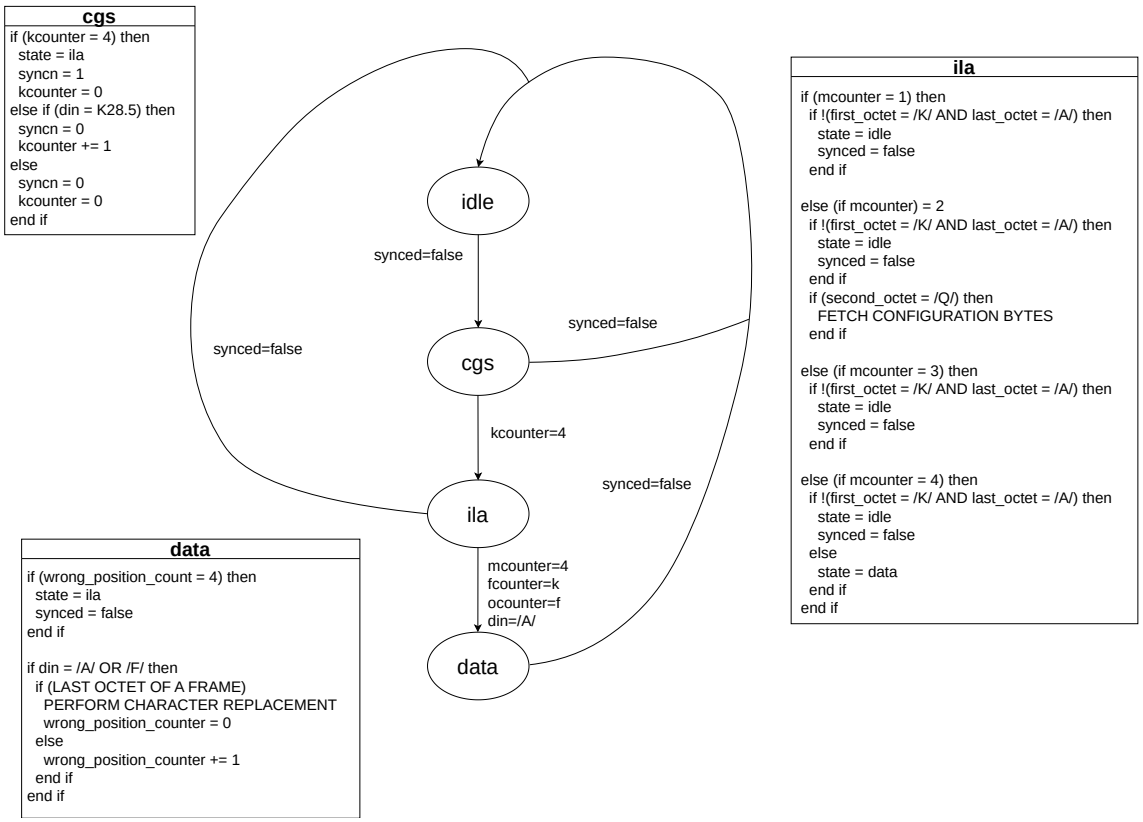


Figure 4.6: Link layer FSM.

The 8b/10b encoding scheme is a module that is already implemented in the existing GRHSSL. Therefore, we excluded the development of such decoder inside the link layer of the JESD204B receiver. Instead, the link layer will only consist of the controller that controls the flow of the receiver, as previously explained with the state machines above.

4.3.4 Design of transport layer

Valid data is transmitted at octet rate from the link layer to the transport layer. The role of the transport layer is to filter out control bits and tail bits from each octet, and merge the data from F octets, in order to reconstruct the original 14-bit samples, that were sampled by the data converter. Refer to Figure 2.9, for an illustration of the octet structure.

The transport layer should route the valid data out to the application layer when two conditions are fulfilled: data is valid from the link layer, and the data is de-framed and merged successfully in the transport layer.

4.4 Testbench setup

Behavioral verification methods are required to be able to verify that the desired output is obtained. It is a very important step when dealing with complex and large systems that are integrated into other major blocks such as SoC systems. In order to verify the designed JESD204B receiver, a VHDL testbench was implemented. Since the receiver dictates the transmitter state transitions, a dummy ADC, which is sensitive to the **SYNC**~ signal was needed. This dummy ADC acts similarly to the transmitter in the actual hardware.

The intended input to the testbench were 14-bits signals generated via python and saved into a file. This input is then automatically fed into the testbench from an external file. The purpose of this setup is to mimic a real world scenario with a large set of data coming from a data converter. Following this approach, the data could also come from a data dump from a physical sensor, for a more authentic simulation.

The receiver implementation is able to assert synchronization requests, but also catch data from the dummy ADC. The outputted results of the JESD204B receiver is saved into a log file that can be used for verification, comparing it to a golden reference file. The log file can also be used into a high level application layer. A clock generation module was provided into the testbench, to generate a **SYNC**~ detection clock, as well as a sample clock used for octet sampling. An overview of the designed testbench environment is shown in Figure 4.7.

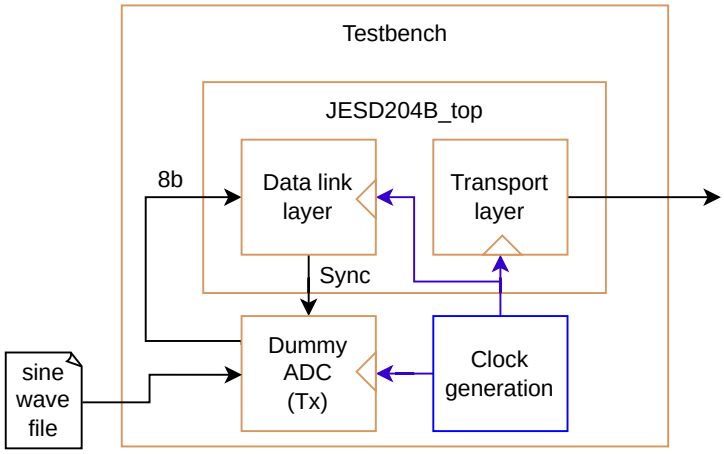


Figure 4.7: The testbench used to verify the behavioral RTL of the receiver.

4.5 Application layer

The intention of the data reduction algorithm is to be implemented on the FPGA fabric of the upcoming SoCs. We have chosen two data reduction algorithms to be investigated, keeping in mind how well they translate into hardware implementation, especially in terms of area. RLE and moving average are two applications that can be used as the receiver application layer, which are well-suited for hardware implementation. They can be implemented using flip flops, shift registers and basic

arithmetic operations. The algorithms read the output from the JESD204B as input, to later perform either RLE or moving average. In this investigation, the algorithms in the application layer were implemented and studied in software only, due to time limitation.

4.5.1 Run-length encoding

The key here is to find consecutive 1's or 0's then group them together. For that a counter is used to keep track of the current and previous index in a given word. Below is a pseudo code that describes the implemented RLE algorithm.

Algorithm 1 Run-Length Encoding (RLE) Algorithm

```
1:  count = 1
2:  currentChar = data[0]
3:  for i from 1 to len(data) - 1 do
4:      if data[i] = currentChar then
5:          count = count + 1
6:      else
7:          if count > 1 then
8:              append (count, currentChar) to compressedData
9:          else
10:             append currentChar to compressedData
11:         end if
12:         count++
13:         currentChar = data[i]
14:     end if
15: end for
16: if count > 1 then
17:     append (count, currentChar) to compressedData
18: else
19:     append currentChar to compressedData
20: end if
```

4.5.2 Moving average

The algorithm is built upon reading a sampling window, where the samples are summed together. The average is then computed through the sample window. The sampling window can be built in hardware through shift registers, meanwhile the other operations can be accomplished by simple arithmetic in hardware. Figure 4.8 shows the implementation of moving average.

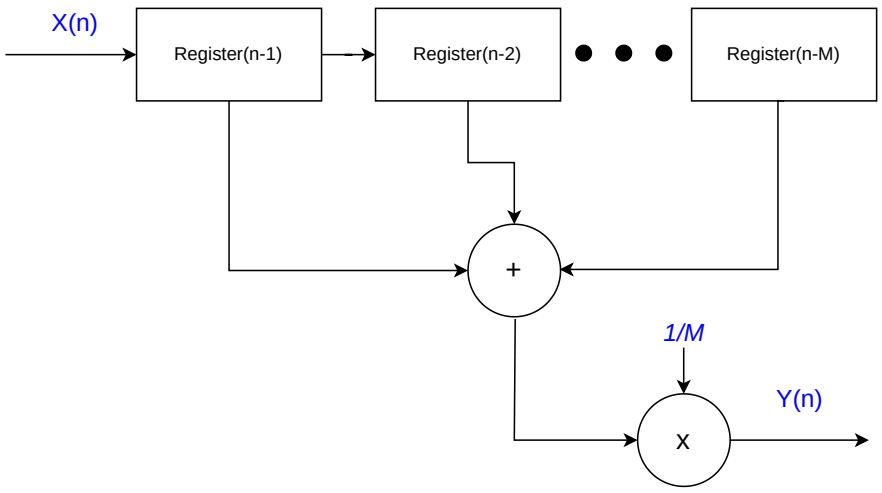


Figure 4.8: Block diagram of moving average algorithm.

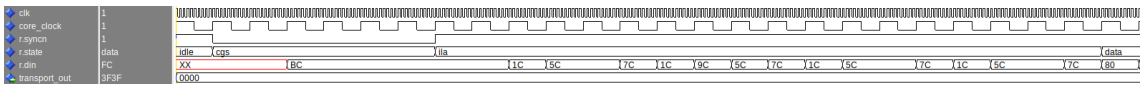
5

Results

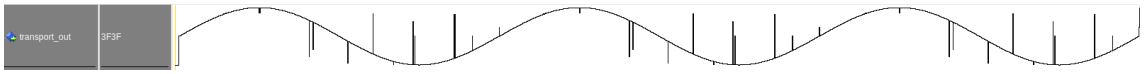
The following chapter presents the results obtained throughout the project. Initially, we discuss the outcomes from the behavioral simulation of the receiver, highlighting both the output and the synchronization request initialization. Subsequently, we present the results from the FPGA implementation, including the ADC configuration and SerDes setup, as well as the outcomes from the JSED204B implementation. Finally, we present the results from our investigation of the selected data reduction algorithms.

5.1 Behavioral simulation of receiver RTL

Simulation of our IP, containing the link layer and transport layer was performed using a sine wave of 14-bits resolution as stimuli, to represent ADC samples. The samples were packaged into octets and fed into the IP. A successful synchronization sequence of CGS and ILA can be seen in Figure 5.1a. One can clearly see the sequence of receiving K28.5 characters during CGS (0xBC), followed by four multiframes during the ILA state, where each multiframe starts with /R/ and ends with /A/ characters (0x1C and 0x7C). The output from the transport layer is shown in Figure. 5.1b. `SYSREF`



(a) CGS and ILA.



(b) Output from transport layer.

Figure 5.1: Behavioral simulation results.

As seen in Figure 5.1b, noticeable spikes are present on the sine wave output, further shown in Figure 5.2. The spikes occur inside the link layer, and come as a consequence of receiver character replacement.

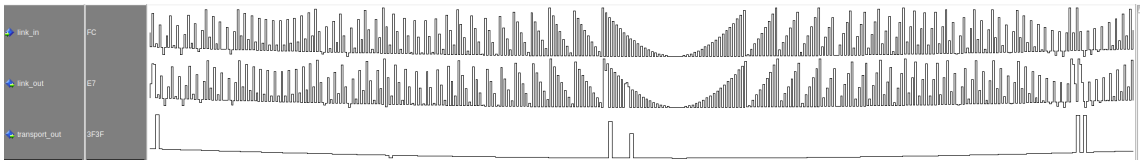


Figure 5.2: Sine wave spikes from the receiver output.

As stated earlier, 8b/10b encoding and decoding were excluded from our testbench environment, resulting in all inputs being 8 bits long. This meant that our receiver misinterpreted data values to be control characters. An actual transmitter however, would encode each 8-bit octet into a 10 bit word. The way 8b/10b encoding encodes the data octets, makes it impossible for a 10-bit data word to be identical to any of the 10-bit control characters. If an 8b/10b encoder and decoder would have been implemented on the transmitter and the receiver side, we would not expect to see any spikes on the output.

A working re-synchronization mechanism inside the receiver was verified, by inserting /A/ and /F/ characters at faulty positions into the data stream, to replicate a scenario where the transmitter and the receiver are out of phase. Figure 5.3 illustrates a re-synchronization request from the receiver, after it has received a certain amount consecutively misplaced alignment characters.

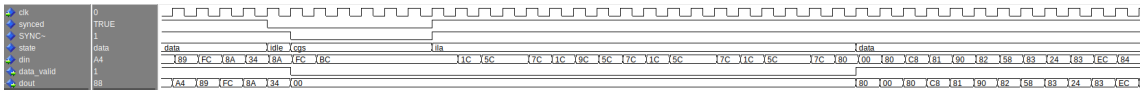


Figure 5.3: Re-synchronization process. The receiver suspects misalignment after detecting faulty placed comma characters. The receiver issues a synchronization request and goes through ILA state.

5.2 FPGA implementation

The FPGA was connected to the ADC through the FPGA Mezzanine Card (FMC) connector. Signals such as: SPI pins for ADC configuration, core and SerDes clocks, **SYSREF** , **SYNC**~ and most importantly the data pins are routed through the FMC connector. Figure 5.4 shows the hardware parts used in the projects.

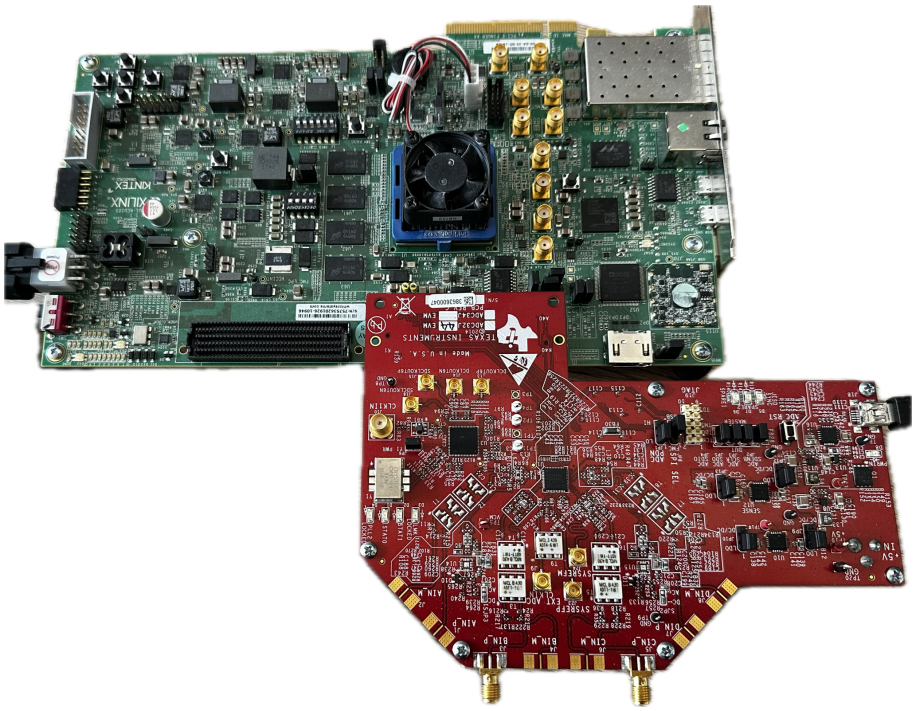


Figure 5.4: FPGA and ADC connected together on FMC connector.

5.2.1 Apbfifo and GRMON

The **APBFIFO**, used for storing the SerDes samples was successfully synthesized and implemented on the FPGA. Inside the JESD204B wrapper module, two **APBFIFO** registers are found. One of them is reporting a constant value to simplify the verification of AMBA communication with our JESD204B receiver. The second register reports values coming from the data stream of the SerDes through the same AMBA interface. Being able to read the registers indicates a working connection between the receiver and the AMBA bus. As expected, when doing a read request on GRMON at startup, the real data register only shows zeroes, while the capability

register reports the given constant value. Figure 5.5 shows the GRMON debug monitor output, verifying that it was able to detect the receiver module on the AMBA APB bus.

```
GRMON debug monitor v3.3.10 64-bit pro version

  Copyright (C) 2024 Frontgrade Gaisler - All rights reserved.
  For latest updates, go to https://www.gaisler.com/
  Comments or bug-reports to support@gaisler.com

JTAG chain (1): xcku040
  GRLIB build version: 4292
  Detected frequency:  100.0 MHz

  Component                              Vendor
  AHB Debug UART                         Frontgrade Gaisler
  JTAG Debug Link                        Frontgrade Gaisler
  AHB/APB Bridge                         Frontgrade Gaisler
  Single-port AHB SRAM module            Frontgrade Gaisler
  Modular Timer Unit                     Frontgrade Gaisler
  General Purpose Register               Frontgrade Gaisler

  Use command 'info sys' to print a detailed report of attached cores

grmon3> info sys
  ahbuart0   Frontgrade Gaisler  AHB Debug UART
             AHB Master 0
             APB: 80000000 - 80000100
             Baudrate 115200, AHB frequency 100.00 MHz
  ahbjtag0   Frontgrade Gaisler  JTAG Debug Link
             AHB Master 1
  apbmst0    Frontgrade Gaisler  AHB/APB Bridge
             AHB: 80000000 - 80100000
  ahbram0    Frontgrade Gaisler  Single-port AHB SRAM module
             AHB: a0000000 - a0100000
             32-bit SRAM: 4 kB @ 0xa0000000
  gptimer0   Frontgrade Gaisler  Modular Timer Unit
             APB: 80000100 - 80000200
             IRQ: 1
             16-bit scalar, 1 * 32-bit timers, divisor 100
  jesd0      Frontgrade Gaisler  JESD204B Receiver
             APB: 80000200 - 80000300

grmon3> info reg jesd0
  General Purpose Register
  → 0x80000200  General purpose register 0              0xdeadbeef
    0x80000204  General purpose register 1              0x00000000
```

Figure 5.5: GRMON read access result. IP-cores shown are detected on the AMBA bus.

5.2.2 ADC configuration and SerDes

Regarding ADC configuration, the registers of the ADC chip were successfully configured. We could verify the configuration by reading the registers before and after configuration. The evaluation board has both an ADC chip, and an LMK04828 clock generation chip. The LMK04828 chip is responsible for generating the SerDes clock, core clock to FPGA, core clock to the ADC chip and an external clock to an output pin for debugging. The chip is configured using the same SPI interface on the evaluation board. The LMK04828 chip can be configured using a larger set of parameters compared to the ADC. Many of them are used for PLL and VCO configuration. This led to an iterative workflow to set up the clocks and getting the board in operation.

The ADC contains a set of stored patterns in the memory that can be sent to the receiver via the SerDes. Therefore, the first registers to be configured on the LMK04828 chip were the registers controlling the ADC clock as well as the external pinout clock. The external debug clock was successfully generated and captured via an oscilloscope using the external pinout on the evaluation board. Afterwards, the registers controlling the ADC clock, receiver core clock and receiver SerDes clock needed to be configured and generated. There has to be a specific relationship between the clocks to obtain a correct SerDes communication. Since the clocks are transmitted via the FMC connector, they are complicated to measure and debug.

Our strategy was to, through trial and error, configure the registers of the evaluation board, and inspect the incoming data stream after the FPGA SerDes using `APBFIFO` module. However, due to limitation in time, and the large set of configurations for the LMK chip, we were unable to verify the correct clocks and capture any SerDes data on the FPGA board. Looking back at our work methodology, we would have divided the tasks so more time could be spent on configuring the hardware. According to our technical supervisor, a physical layer typically requires notable time and extensive experimentation to achieve proper functionality.

5.2.3 JESD204B receiver performance evaluation

The simulated JESD204B receiver was synthesized and implemented in Vivado 2019.2. In this section we present the obtained implementation results. Area, timing and power are presented for the created SoC subsystem, as shown in 5.6. The results are presented for the subsystem with and without the receiver initialization.

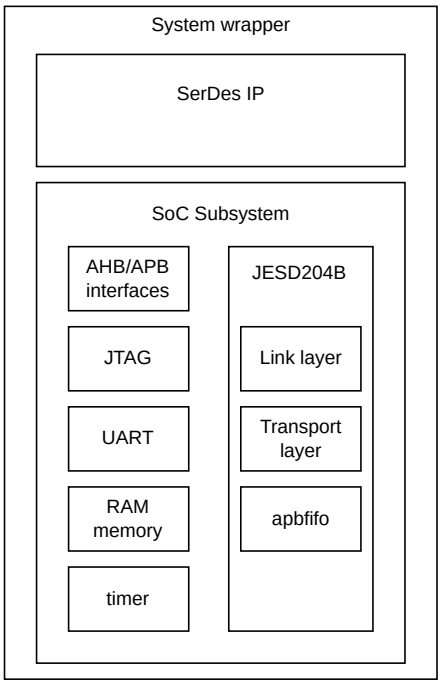


Figure 5.6: Design hierarchy of the implemented subsystem with JESD204B.

Utilization

From the Vivado resource usage report, presented in Table 5.1 and Table 5.2, it was seen that the addition of the SerDes, `APBFIFO` and the JESD204B receiver, resulted in a total increase of resource-utilization of around 50%, for the whole design.

Table 5.1: Subsystem without JESD204B.

Name	CLB LUTs	CLB Registers	CARRY8	F7 Muxes	BRAM
system_wrapper	680	609	12	2	2
subsystem	679	603	12	2	2
ahbctrl	57	20	0	0	0
apbctrl	86	88	0	2	0
jtag	123	194	0	0	0
uart	272	179	8	0	0
gptimer	121	106	4	0	0
ahbram	20	16	0	0	2
total	2038	1815	36	6	6

Table 5.2: Subsystem including JESD204B receiver and **APBFIFO**.

Name	CLB LUTs	CLB Registers	CARRY8	F7 Muxes	BRAM
system_werapper	956	993	18	13	3
serdes_ip	99	212	4	0	0
subsystem	848	763	14	13	3
ahbctrl	58	20	0	0	0
apbctrl	135	89	0	13	0
jtag	122	194	0	0	0
uart	277	179	8	0	0
jesd_wrapper	119	159	2	0	1
apbfifo	54	87	2	0	1
jesd_top	65	72	0	0	0
gptimer	118	106	4	0	0
ahbram	20	16	0	0	2
total	2871	2890	52	39	10

From the above tables, it is visible that **APBCTRL** is significantly bigger when the receiver is connected to the subsystem. The reason is that the JESD204B contains the **APBFIFO** module, which has an APB interface that is connected to the APB controller. This addition causes a significant resource increase in CLB LUTs.

Timing

The main clock in the subsystem is the AMBA clock at 100 MHz. From the timing report provided by Vivado, no timing violations are seen. Tables 5.3 and 5.4 show the timing report for the SoC subsystem with and without the receiver.

Table 5.3: SoC subsystem No JESD204B is connected.

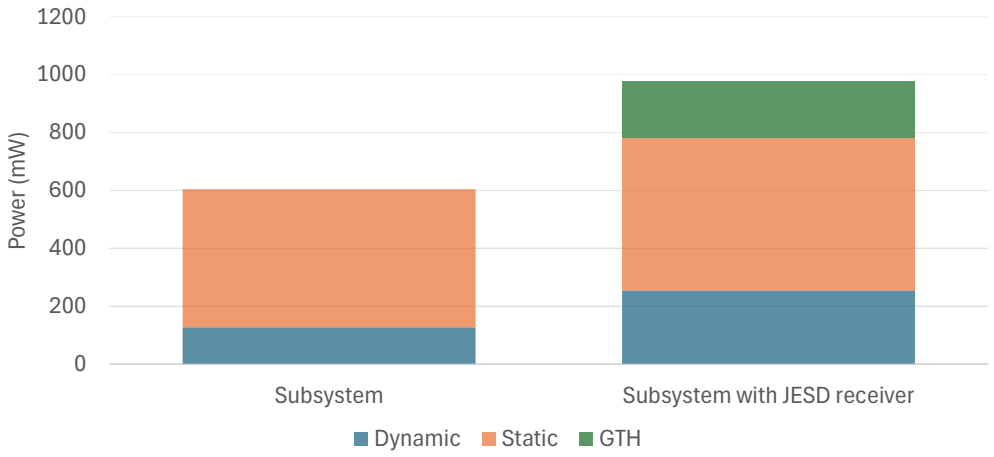
Setup	Hold	Pulse Width
Worst Negative Slack: 6.283 ns	Worst Hold Slack: 0.016 ns	Worst Pulse Width Slack: 0.500 ns
Total Negative Slack: 0.000 ns	Total Hold Slack: 0.000 ns	Total Pulse Width Negative Slack: 0.000 ns

Table 5.4: SoC subsystem, including JESD204B receiver and **APBFIFO**.

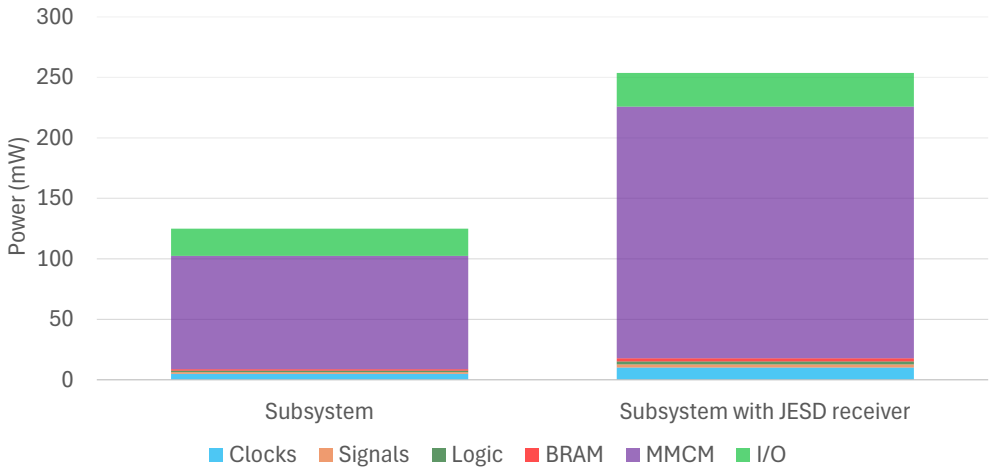
Setup	Hold	Pulse Width
Worst Negative Slack: 5.375 ns	Worst Hold Slack: 0.024 ns	Worst Pulse Width Slack: 0.486 ns
Total Negative Slack: 0.000 ns	Total Hold Slack: 0.000 ns	Total Pulse Width Negative Slack: 0.000 ns

Power

Power dissipation of the design is evaluated using the Vivado tool. Again, power is presented for the subsystem with and without the receiver design. The subsystem without a receiver and **APBFIFO**, would have an estimated power consumption of 604 mW. With the inclusion of the receiver and **APBFIFO**, the total on-chip power consumption reached 977 mW. See Figure 5.7 for a detailed power breakdown.



(a) On-chip power (W).



(b) Dynamic power breakdown (W).

Figure 5.7: Subsystem power consumption, including JESD204B receiver.

As seen in Figure 5.7, there was not a significant change in static power after the addition of our receiver prototype to the subsystem. A notable increase in dynamic power was seen after the receiver was added, but the majority of the power increase was caused by the on-board transceiver, housing the SerDes.

5.3 Application layer

It is crucial to note that the application layer of the JESD204B receiver varies for each implementation. Specifically, this master thesis focuses on studying two distinct data reduction algorithms as the application layer. The results of these algorithms are presented here. Various stimuli inputs were utilized to evaluate the suitability of each algorithm for different applications. It is important to highlight that all tests were conducted using Python, and no hardware implementation was carried through.

5.3.1 RLE results

In theory, RLE is expected to achieve the best data reduction when the data demonstrates repetitive patterns. By contrast , RLE is likely to produce minimal reduction for inputs characterized by randomness and infrequent occurrences. To thoroughly evaluate the RLE algorithm across different contexts, it is important to utilize a diverse dataset that encapsulates various characteristics of input data.

The intuition behind this approach was to evaluate the algorithms performance across different scenarios. Therefore, the test data was categorized into four categories: Inputs with repeating data, inputs with less repeating data, random or complex inputs to ensure real-world data evaluation, and finally, inputs with different sizes to understand how RLE scales with image size. A set of binary images were initially selected to stimulate the RLE algorithm, which can be shown as results in Figure 5.8.

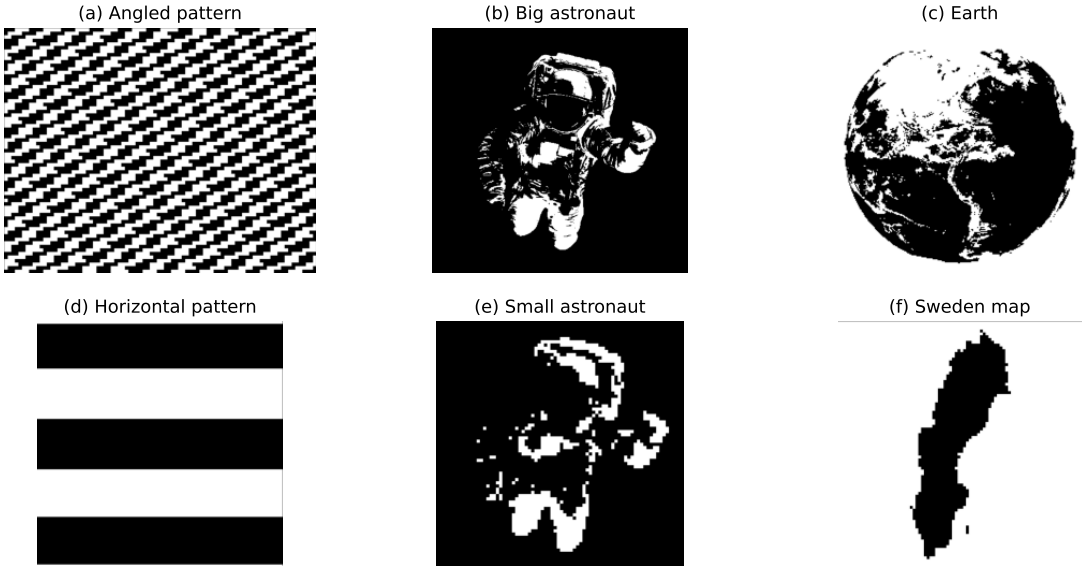


Figure 5.8: Binary image inputs to the RLE algorithm.

The resulting output after running the RLE algorithm was saved into a log file that was then compared to the reference input. To determine the effectiveness of data reduction, we selected two benchmarks that quantified the performance of RLE.

The first benchmark; average word size after reduction, provides insight into the efficiency of the reduction process. The second benchmark, average reduction rate, is calculated by dividing the original file size by the compressed file size, offering a clear ratio of size reduction. Table 5.5 shows the corresponding information and benchmark values for the different inputs.

Table 5.5: RLE reduction results for different input types and sizes.

Input description	Input type	Input size (pixels)	Average word size after reduction	Average reduction rate
Sine wave	Signal	14x3072	10.33	1.48
Triangle wave	Signal	16x100,000	12.47	1.32
Map of Sweden	Image	88x88	8.61	11.18
Angled pattern	Image	88x88	34	2.6
Horizontal pattern	Image	88x88	3	29.3
Earth	Image	250x250	31	16.54
Big astronaut	Image	488x467	31.4	31.4
Small astronaut	Image	88x88	12.23	11.48
Audio recording	Signal	16x191,000	9.77	1.75

One can clearly see that RLE gave the highest reduction rate when the rows of the inputs were following a repeated pattern. One obvious example is Figure 5.8 (d) which features horizontal lines, with every line containing identical binary values. Additionally, inputs with larger pixels that are similar in nature yielded a higher reduction rate. This is expected, because longer rows of binary values have a statistically higher probability of containing repeated values. Example of that are the two inputs for big astronaut and small astronaut. From the table, one can see that the input sizes are 88 compared to 488.

RLE was also evaluated using signals, specifically a sine wave and an audio recording. However, the reduction rate for these signals was noticeably low. Signals tend to have a high information content with fewer repeating patterns, which is what RLE relies on for reduction. Another challenge with signals is their resolution, as they typically have a wordlength between 8-32 bits, which can further limit the effectiveness of RLE. All things considered, RLE provided a solid lossless reduction for binary input images. The expected reduction rate is estimated to be between 1 and 8.

5.3.2 Moving average results

The moving average algorithm was implemented in two ways: The first one is called sliding moving average (SMA), which calculates the average of a sampling window, and increments one sample at a time. The second approach is called block moving average (BMA). This second approach runs the same algorithm, with the only difference being that it calculates the average of a sample window, then jumps k number of samples and calculates the moving average of that block, providing a downsampled result. This makes it an excellent method to accomplish frequency reduction for signals.

Measuring reduction rate is not as applicable for SMA in the same way as it is for RLE. SMA relies on frequently calculating averages of a window sample, and fading away the odd values in that window sample to become as close as possible to other neighbouring signals. The results of the SMA showed a significant granular and detailed smoothing for noisy inputs. The noisy sine wave output obtained from the

receiver was tested with SMA and a sample window of 100 was selected. Figure 5.9 shows the results. As expected, a nice and clean output was obtained after running SMA algorithm on the noisy input.

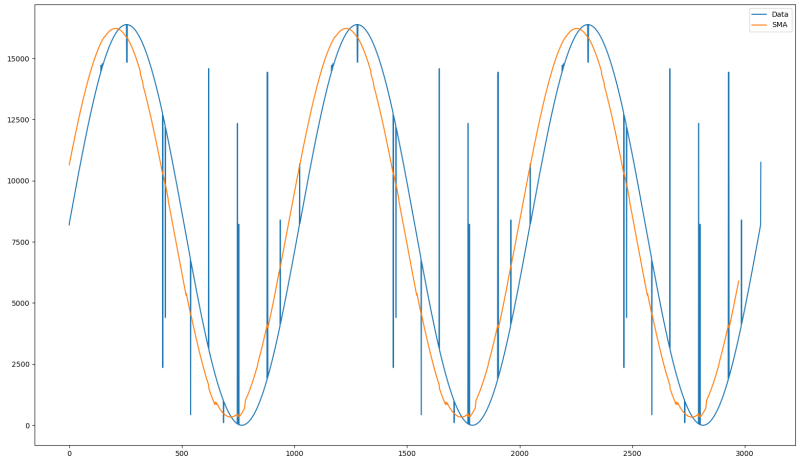
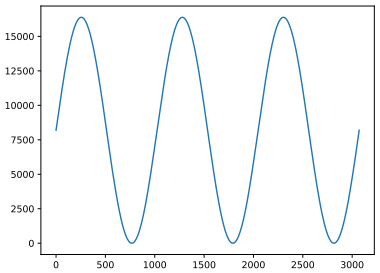
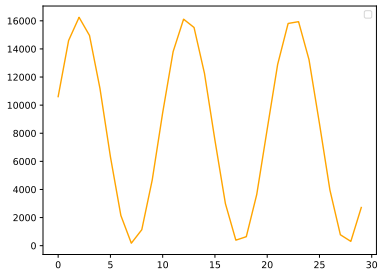


Figure 5.9: A filtered sine wave using sliding moving average.

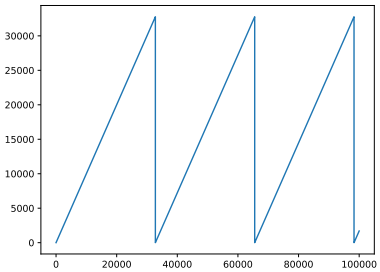
The goal of the BMA is to downsample a signal, which is particularly useful when an application runs at a slower rate compared to the input rate. An average is calculated once every k samples, resulting in a lossy type of downsampling and data reduction. Figure 5.10 shows the reconstruction of a sine and a triangle wave using the BMA algorithm.



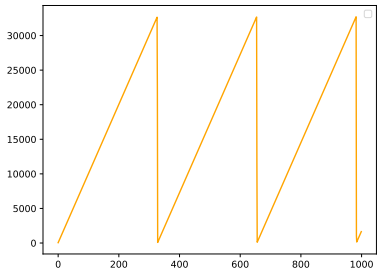
(a) Sine wave input.



(b) Sine wave output.



(c) Triangle wave input.



(d) Triangle wave output.

Figure 5.10: BMA algorithm performed on a sine wave and triangle wave signals.

By its nature, BMA is a lossy algorithm, which would not be optimal for systems where data loss is intolerable. Running BMA on such systems corresponds in loss of details and reduced quality. The audio recording was evaluated using BMA; however, SMA proved to be the better choice for the audio file, due to the complex and non-periodic characteristic of speech and music.

5.3.3 Algorithm-FPGA interaction

The theoretically studied BMA, SMA, and RLE algorithms offer distinct advantages and trade-offs in performance. They will however be implemented on FPGA fabric, resulting in different area affects on the hardware.

BMA on FPGA

BMA is particularly effective for downsampling signals, with a significant data reduction capabilities. However, its primary drawback is its lossy nature when applied to complex signals that contain critical information. This makes BMA less suitable for applications where signal integrity is of great importance. The simplicity of BMA's arithmetic operations translates well to FPGA implementation using flip-flops and a small number of shift registers depending on the downsampling resolution.

SMA on FPGA

On the other hand, SMA offers precise and detailed granular smoothing for input signals, making it more appropriate for complex and non-periodic signals that carry essential information. Despite its advantages in maintaining signal integrity, the increased complexity of SMA means it will occupy more FPGA resources in terms of shift registers compared to BMA. SMA achieves a noticeably less data reduction compared to BMA.

RLE on FPGA

RLE provides a robust lossless reduction primary for input images, with an expected reduction rate typically ranging between 1 and 8 for larger sets of inputs. The algorithm achieves the highest reduction rate with inputs that have repeated patterns. However, it performs poorly with signals like sine waves and audio recordings due to their pseudo random information, resulting in fewer repeating patterns. Implementing RLE on FPGA requires control logic gates to detect repeated values, with flip-flops to manage the counts.

In conclusion, the selection between BMA, SMA, RLE or any other algorithm for the receiver application layer should be application specific to each space mission. This means balancing trade-offs between performance, signal integrity, data reduction, and hardware area. Integrating these algorithms into the FPGA fabric allows the application layer of the JESD204B receiver to dynamically adapt to changes without making modifications to the JESD204 receiver IP-core that can be part of any SoC.

6

Conclusion

In this project, we aimed to investigate the attainability of integrating a JESD204B receiver into an SoC architecture, by the means of a minimal SoC referred to as a subsystem on an FPGA. The investigation comprised of a comprehensive study of the JESD204 standard, as well as preparatory calculations of the parameters for a link between a JESD204B receiver and transmitter.

A prototyped JESD204B receiver containing a link layer and a transport layer was developed and verified in simulation. According to our initial time plan, we estimated that the receiver IP would be quick to implement. However, we later realized that the scope was much larger than anticipated, leading to adjustments in our time planning. Despite that, the development of the subsystem prototype went according to redefined plan. It is however worth noting that the prototype and the testbench environment were relatively simple, lacking certain functionalities that need to be addressed in future development. Implementing a fully-featured JESD204 receiver IP, which supports the wide range of configuration options would take a significant amount of time.

A third party transmitter was part of an ADC evaluation board, which was configured via an SPI interface. An UltraScale XCKU040 FPGA was the platform for our JESD204B receiver implementation, and the link was to be established through an FMC connector. The foundation for a physical layer in the receiver was created, where much time was spent on trying to set up communication over the JESD204B link between the transmitter and the receiver. This phase required extensive debugging, which involved the GRMON debugger, and the created `APBFIFO` to capture SerDes data.

Ultimately, after several iterations of trial and error, communication over the configured JESD204B link could not be established. In hindsight, the project timelines should have been more conservatively estimated to make room for potentially unforeseen technical challenges. It was anticipated that the setup of the ADC could be accomplished within a relatively short time-frame, which in reality proved to be overly optimistic, and very time-consuming. Even if a working JESD204B link was not achieved, a solid infrastructure for further development of the JESD204B receiver was created, which will potentially simplify future work surrounding this project.

Two data reduction algorithms were studied and developed in software with the aim of using them inside the application layer of the JESD204B: RLE; and Moving Average, which includes both SMA and BMA. Both algorithms are well-suited for hardware implementation. The designer should be aware of the trade-offs between the algorithms performance and area utilization when implemented in hardware. In terms of performance, RLE is recommended when the inputs are known to contain repeated patterns. For the moving average: SMA is best when a smoother output is desired, meanwhile BMA is best used for downsampling.

During the planning phase, we allocated the same amount of time for the data reduction algorithms and the receiver IP. Upon reflection, it would have been more advantageous to spend more time on the receiver due its complexity, but also because the algorithms were only studied in software, not implemented in hardware, and required less time that planned.

In future work, the system can be improved in several ways: first and foremost is to read ADC data and to establish JESD204 communication on hardware. Regarding the RTL and the receiver itself, the prototype is a simplified version of what to expect from a complete JESD204B receiver IP. Therefore, one essential part of the future work is to add several features to the receiver and include it on GRLIB. The limitations are the currently designed transport layer, which only works for the exact same setup proposed in the results section. For the link layer, the 8b/10b decoder from GRHSSL needs to be integrated together with the controller. This will eliminate the misinterpretation of actual data and control characters. The current testbench can be improved by making it self checking, and not being dependent on the wave window for verification.

Bibliography

[1] *Using FPGA in space applications everything you need to know*, vorago. [Online]. Available: https://www.voragotech.com/blog/fpga-in-space-applications.

[2] T. Nguyen, C. MacLean, M. Siracusa, D. Doerfler, N. J. Wright, and S. Williams, "FPGA-based HPC accelerators: An evaluation on performance and energy efficiency," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 20, e6570, Sep. 10, 2022, ISSN: 1532-0626, 1532-0634. DOI: 10.1002/cpe.6570. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/cpe.6570 (visited on 04/11/2024).

[3] *Gr765 octa-core processor*, Gaisler. [Online]. Available: https://www.gaisler.com/index.php/products/components/gr765.

[4] *Lms instrument*, Universität Bern. [Online]. Available: https://www.space.unibe.ch/research/research_groups/space_science_group/science/lms_instrument/index_eng.html.

[5] A. Riedo, S. Meyer, B. Heredia, *et al.*, "Highly accurate isotope composition measurements by a miniature laser ablation mass spectrometer designed for in situ investigations on planetary surfaces," *Planetary and Space Science*, vol. 87, pp. 1–13, 2013, ISSN: 0032-0633. DOI: https://doi.org/10.1016/j.pss.2013.09.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0032063313002341.

[6] *Gr712rc dual-core leon3ft sparc v8 processor*, Gaisler. [Online]. Available: https://www.gaisler.com/index.php/products/components/gr712rc.

[7] H. Saheb and S. Haider, "Scalable high speed serial interface for data converters: Using the JESD204B industry standard," in *2014 9th International Design and Test Symposium (IDT)*, Algeries, Algeria: IEEE, Dec. 2014, pp. 6–11, ISBN: 978-1-4799-8200-4. DOI: 10.1109/IDT.2014.7038577. [Online]. Available: http://ieeexplore.ieee.org/document/7038577/ (visited on 03/20/2024).

[8] Gaisler, "Gaisler processors," 2023. [Online]. Available: https://www.gaisler.com/index.php/products/processors.

[9] *Grlib vhdl ip library*, Gaisler. [Online]. Available: https://www.gaisler.com/index.php/products/ipcores/soclibrary.

[10] A. Amara, F. Amiel, and T. Ea, "Fpga vs. asic for low power applications," *Microelectronics Journal*, vol. 37, no. 8, pp. 669–677, 2006, ISSN: 1879-2391. DOI: https://doi.org/10.1016/j.mejo.2005.11.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026269205003927.

[11] F. Maloberti, *Data Converters*. Dordrecht: Springer, 2008, 440 pp., ISBN: 978-0-387-32485-2.
[12] P. Dhaker, "Introduction to SPI Interface," *Analog Dialogue*, vol. 52, 2018.
[13] *AMBA*, ARM. [Online]. Available: https://developer.arm.com/Architectures/AMBA.
[14] *AMBA AHB protocol specification*, ARM. [Online]. Available: https://developer.arm.com/documentation/ihi0033/latest/.
[15] *AMBA APB protocol specification*, ARM. [Online]. Available: https://developer.arm.com/documentation/ihi0024/latest/.
[16] F. Zhang, *High-Speed Serial Buses in Embedded Systems*. Singapore: Springer Singapore, 2020, ISBN: 9789811518676 9789811518683. DOI: 10.1007/978-981-15-1868-3. [Online]. Available: http://link.springer.com/10.1007/978-981-15-1868-3 (visited on 03/20/2024).
[17] *JESD204C Standard*, 2021. [Online]. Available: https://www.jedec.org/.
[18] *JESD204B Standard*, 2011. [Online]. Available: https://www.jedec.org/.
[19] C. Panek, *Networking Fundamentals*. Indianapolis: John Wiley & Sons, Inc, 2019, ISBN: 978-1-119-65074-4.
[20] A. Athavale, *High-Speed Serial I/O Made Simple*. Xilinx, Inc, 2005, ch. 3, 20-25.
[21] J. Harris, "Understanding JESD204B link parameters," *Planet analog*, 2013. [Online]. Available: https://www.planetanalog.com/understanding-jesd204b-link-parameters/.
[22] "8b10b Encoder/Decoder MegaCore Function (ED8B10B) Data Sheet," Altera Corporation, 2001.
[23] T. Biscontini, "Data compression.," *Salem Press Encyclopedia of Science*, 2023. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=ers&AN=87321445&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.
[24] A. Birajdar, H. Agarwal, M. Bolia, and V. Gupte, "Image compression using run length encoding and Lempel Ziev Welch method," in *2019 Global Conference for Advancement in Technology (GCAT)*, 2019, pp. 1–6. DOI: 10.1109/GCAT47503.2019.8978408.
[25] S. Fairooz, S. Balaji, R. Ramya, M. S. Prakash Balaji, P. Thanapal, and V. Elamaran, "A case study using simple moving average filters to accomplish ECG denoising on an FPGA," in *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, 2023, pp. 244–248. DOI: 10.1109/ICACCS57279.2023.10112847.
[26] J. Gaisler, "5 A structured VHDL design method," [Online]. Available: https://www.gaisler.com/doc/vhdl2proc.pdf.
[27] Gaisler, "GRMON debug monitor," 2024. [Online]. Available: https://www.gaisler.com/index.php/products/debug-tools/grmon3.
[28] *Grhssl*, Gaisler. [Online]. Available: https://www.gaisler.com/index.php/products/ipcores/high-speed-serial-links.
[29] R. Zarr, "JESD204B vs. JESD204C: What designers need to know," *All About Circuits*, 2020. [Online]. Available: https://www.allaboutcircuits.com/

industry-articles/jesd204b-vs-jesd204c-what-designers-need-to-know/.

[30] "KCU105 Board User Guide," 2019. [Online]. Available: https://www.xilinx.com/support/documents/boards_and_kits/kcu105/ug917-kcu105-eval-bd.pdf, Xilinx, Inc.

[31] *ADC32J44 dual-channel, 14-bit, 125-MSPS analog-to-digital converter evaluation module*, Texas Instruments. [Online]. Available: https://www.ti.com/tool/ADC32J44EVM.

[32] "Ultrascale FPGAs transceivers wizard v1.6," *AMD*, 2017. [Online]. Available: https://docs.amd.com/r/en-US/pg182-gtwizard-ultrascale.