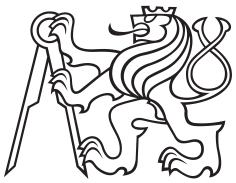
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Radioelectronics

Implementation of the JESD204B Standard on an FPGA Enabling the Interfacing of High-speed A/D Converters with a Sampling Rate Higher than 250 MSPS

František Boháček

Supervisor: Ing. Radek Sedláček, Ph.D.
Study program: Open Electronic Systems
May 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Boháček František** Personal ID number: **498966**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Radioelectronics**

Study program: **Open Electronic Systems**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Implementation of the JESD204B Standard on an FPGA Enabling the Interfacing of High-speed A/D Converters with a Sampling Rate Higher than 250 MSPS

Bachelor’s thesis title in Czech:

Implementace standardu JESD204B na FPGA umožňující připojení vysokorychlostních A/D převodníků se vzorkovací frekvencí vyšší než 250 MSPS

Guidelines:

1. Study the principle of the receiving part of the JESD204B standard, intended for connecting A/D converters to FPGA circuits. 2. In VHDL, implement the receiving part of the JESD204B standard for connecting A/D converters with this interface. 3. Verify the draft of the standard using a suitable simulation tool (e.g. in the ModelSim environment). 4. Choose an affordable A/D converter with JESD204B interface, design a simple development board for it to connect to the Intel Cyclone 10 GX development kit. 5. Verify the correct operation of the implementation of the JESD204B standard on real data obtained from the chosen A/D converter.

Bibliography / sources:

[1] JESD204B Survival Guide: Practical JESD204B Technical Information, Tips, and Advice from the World’s Data Converter Market Share Leader. In: Analog Devices [online]. [cit. 2023-01-30]. Available from: https://www.analog.com/media/en/technical-documentation/technical-articles/JESD204B-Survival-Guide.pdf
[2] Intel® Cyclone® 10 GX FPGA Development Kit. In: Terasic [online]. [cit. 2023-01-30]. Available from: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=253&No=1147
[3] PEDRONI, Volnei A. Digital electronics and design with VHDL. Amsterdam ; Boston, c2008. ISBN 978-0123742704.

Name and workplace of bachelor’s thesis supervisor:

Ing. Radek Sedláček, Ph.D. Department of Measurement FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **30.01.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Radek Sedláček, Ph.D.	doc. Ing. Stanislav Vítek, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my supervisor, Ing. Radek Sedláček, Ph. D., for helping me to choose a topic I would be happy with and for pointing me in the right directions during the work on the thesis.

I would also like to thank LVR (laboratoře pro vývoj a realizaci) for mounting the BGA connector (10 rows, 40 pins each) on my board and Ing. Stanislav Drozd for mounting most of the other components on the board.

Last but not least, I would like to thank my family for supporting me during my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

František Boháček,
Prague, May 26, 2023

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

František Boháček,
V Praze, 26. května 2023

Abstract

The aim of this thesis is to implement a receiver of the JESD204B protocol that is used for high-speed ADCs (250 MSPS and more). The receiver has been implemented in the language VHDL. Testing components, called testbenches, were made and simulated using ghdl and Modelsim. These testbenches verified that the components behave as expected. A testing board with two ADCs has been designed. This board is compatible with the Intel Cyclone 10 GX FPGA development kit. It was attempted to test the board connecting it to the development kit using the custom design. The attempt was not successful and it was discussed where the problem could be and how to find out for sure.

Keywords: JESD204B, ADC, ADC receiver, VHDL, FPGA

Supervisor: Ing. Radek Sedláček, Ph.D.

Abstrakt

Cílem této práce je vytvoření přijímače protokolu JESD204B, který se používá pro vysokorychlostní AČ převodníky (250 milionů vzorků za vteřinu nebo více). Přijímač byl implementován v jazyce VHDL. Byly vytvořeny komponenty pro testování, tzv. testbenche, které byly odsimulovány prostřednictvím programu ghdl a Modelsim. Tyto testy verifikovaly, že komponenty dělají to, co je očekáváno. Dále byla navržena testovací deska se dvěma AČ převodníky. Tato deska je kompatibilní s vývojovou sadou pro FPGA Intel Cyclone 10 GX. Proběhl pokus o otestování vlastní implementace přijímače připojením k této vývojové sadě. Tento pokus nebyl úspěšný a bylo diskutováno, kde by mohla být chyba, případně, jak ji v budoucnu nalézt.

Klíčová slova: JESD204B, AČP, AČP přijímač, VHDL, FPGA

Překlad názvu: Implementace standardu JESD204B na FPGA umožňující připojení vysokorychlostních A/D převodníků se vzorkovací frekvencí vyšší než 250 MSPS

Contents

Figures

Tables

Acronyms

ADC Analog to digital converter. vi, 3, 5, 6, 9, 10, 15, 20, 22, 23, 47–53, 60, 61, 63, 69

ASIC Application specific integrated circuit. 3, 5, 9

BGA Ball grid array. v

CDR Clock/Data Recovery. 7, 12

CGS Code group synchronization. 14

CML Current mode logic. viii, 5–7, 12, 47, 48, 52–56

CMOS Complementary metal oxide semiconductor. viii, 5–7, 50

CSV Comma-separated values. 59

DAC Digital to analog converter. 9, 10, 17, 22, 23

DDR Double data rate. 6

FIFO First in, first out. 11, 15, 33, 39

FMC FPGA mezzanine card. viii, 47–49, 52, 53, 58, 63

FPGA Field programmable gate array. i, vi, 3–5, 9, 25, 26, 47, 49, 50, 52, 57–59, 62, 63

ILAS Initial lane alignment sequence. 14, 15, 20, 22, 23, 39, 45, 61

IP core Intellectual property core. 57, 58, 61

I²C Inter-Integrated Circuit. 5, 6, 47, 49, 50, 59, 63

LMFC Local multiframe clock. viii, 12, 17, 20–23, 28, 30–32

LOL Loss of lock. 60

LVDS Low voltage differential signaling. viii, 5–7, 47, 50, 55

PCB Printed circuit board. 71

PCS Physical coding sublayer. 57, 58

PMA Physical media attachment. 57, 58

RBD Rx buffer delay. 15, 20

RMS Root mean square. 50

SDR Single data rate. 6

SNR Signal to noise ratio. 48, 51, 52

SPI Serial Peripheral Interface. 5, 6, 47, 48, 50, 52, 59, 63

VCO Voltage controlled oscillator. 58

VHDL VHSIC Hardware Description Language. vi, viii, 3, 4, 25, 26, 29–32, 34, 35, 37–41, 43, 44, 59, 62

Chapter 1

Introduction

There is still an upward trend in the needs for performance in data processing systems, including an upward trend in ADC sampling frequencies. Some of the applications that require higher sampling frequencies are traffic control systems or wireless communications.

Attempts have been made to make new standards for transmitting data from Analog to digital converter (ADC) to FPGAs/ASICs while having fast sampling rates (more than 250 MSPS) and while keeping the number of connections low. Having a small number of connections allows for easier PCB designs and keeps the costs lower. JESD204 is a standard from JEDEC that is made for interfacing high-speed ADCs. There are multiple revisions of JESD204, it started with JESD204 in 2006, in 2008, revision A was published (JESD204A). In 2011, JESD204B came out. This thesis is mainly about this revision. The last revision[1] is JESD204C, from 2017. The original standard supported only one signal to transfer data through, of frequency up to 3.125 Gbps. In rev. A, support for sending data over multiple lanes has been added. Revision B added the possibility of speeds of up to 12.5 Gbps over one lane. This revision also introduced a mechanism for ensuring determined latency. Determined latency may be used for synchronizing data from multiple ADCs. [1] JESD204C introduced usage of 64b/66b encoding instead of 8b/10b encoding that had been used in the previous revisions, as well as some other changes. [2]

The main goal of this thesis is to implement JESD204B receiver in VHDL. This receiver will be tested in a simulation as well as on an FPGA. To test the receiver on FPGA, a testing board with an ADC will be developed. Attempt to synchronize data from two ADC will be made by using subclass 1 with the support of deterministic latency.

In chapter 2, various digital output standards as well as standards used for interfacing ADCs will be presented. It will be discussed where the JESD204B standard comes in.

In chapter 3, the JESD204B standard will be described. It will be shown what it is used for, its terminology, and how the data being transmitted looks. This will be presented mainly from the viewpoint of a receiver as a receiver

[1] as of May 2023, revision D is expected to be published late Q3 2023, see `https://www.comcores.com/chip-to-chip-solutions/jesd204d/`

will be implemented as part of this thesis.

In chapter 4, an implementation of custom JESD204B receiver in VHDL will be described. This receiver should contain most of the features of the standard for subclass 0 and 1. Testbenches for testing the final design will be discussed here as well.

In chapter 5, the design of the PCB for testing of the receiver, will be presented. It will be discussed what components were chosen for the PCB and why, how the board was designed and some challenges that came with routing the high-speed signals.

And finally, in chapter 6, testing using Intel FPGA will be described and the results of the testing will be presented.

The appendices contain the main folder structure of the attachment, in Appendix A, and exported Gerber files, in Appendix B.

Chapter 2

Comparison of interfacing methods for A/D converters

There are several ways to interface data from ADCs to FPGAs/ASICs. What the right way for a specific application is, may depend on multiple parameters. Some of the most important parameters may be the sampling frequency or the number of I/O pins.

When it comes to the format of the data being transmitted, there are various standards for that. Some of these standards include: parallel CMOS, parallel LVDS, serial LVDS, I²C, SPI, and JESD204. [1] Some of these are named after a digital output standard, specifically Complementary metal oxide semiconductor (CMOS), Low voltage differential signaling (LVDS), and Current mode logic (CML). Digital output standards prescribe how the output is driven and thus also tell how to correctly receive the data from the signal. Power consumption of these digital output standards for dual channel ADC with 14 bits of resolution is illustrated in Figure 2.1. [1] CMOS is common among lower-speed (sub 200 MSPS) ADCs as it consumes less power. One of the motivations to use JESD204 (using CML) for high-speed ADCs is that it draws less current than both CMOS or LVDS for higher sampling rates.

CMOS digital output standard draws little current when the output is not changing. That is because there are transistors that draw little current when in static state. However, when the signal is changing, the transistors draw more current. For faster transition higher currents are needed. Another drawback of parallel CMOS is that N outputs are needed to transfer data from N-bit converter. That would lead to very complex board layouts as the number of converters grows. [1] If the data were transmitted serially, that would mean higher frequencies and consequently more current drawn.

LVDS is a differential standard and thus needs double the number of wires compared to CMOS for the same number of signals making the routing more difficult. There are a couple of advantages to LVDS compared to CMOS though. LVDS is differential and that offers the benefit mutual noise cancellation. Noise is usually common to both signal paths and is thus canceled out. LVDS uses lower voltage swings compared to CMOS and the driver draws a constant current. This offers benefits compared to CMOS. There could be a situation where all of the CMOS drivers are being

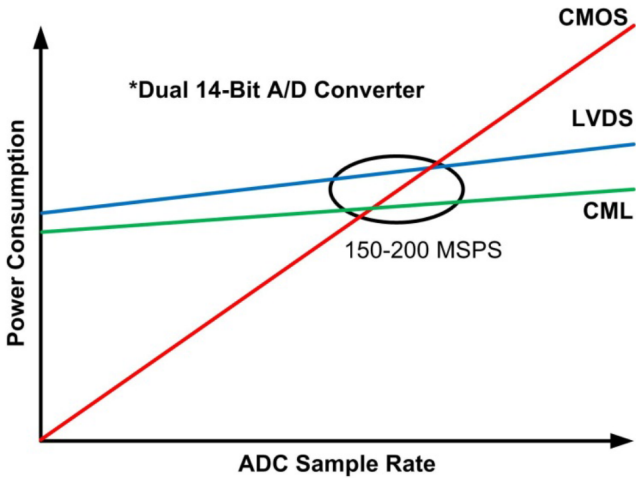


Figure 2.1: Comparison of CMOS, LVDS and CML drivers power consumption. [1]

simultaneously changed and the power supply voltage may get pulled down due to that.

Power consumption of LVDS is not that much higher than for CML, but all of the LVDS lanes must be aligned to a data clock. Because of that, there is an upper bound for LVDS, whereas CML may go to much higher speeds. [1] LVDS is limited to 1.9 Gbps for an ideal transmission medium, however for the real world that limit is usually about 1.0 Gbps. [1]

Serial LVDS usually uses data and frame clocks aligned with the data on the LVDS pairs. Serial LVDS may be employed instead of parallel LVDS where the number of pins is more important than interface speed. Parallel LVDS can be though of as being made from multiple serial LVDS lines. [1]

Both LVDS and CMOS may utilize Double data rate (DDR). In Single data rate (SDR), data are sent from the transmitter on one clock edge and sampled at the receiver on the next edge. For DDR the data are sent and sampled on both edges. [1]

Another possibility to interface ADCs, is to use Inter-Integrated Circuit (I^2C), it uses only two wires - clock an data. A large number of devices may be present on one I^2C bus without any more additional pins. Each of the devices has an address that is sent at the beginning of the communication. It is relatively slow, allowing for only up to 1 MHz. If size is an important parameter, I^2C may come in handy. It may also be used as a control interface. [1].

Serial Peripheral Interface (SPI) is an interface that uses 3 or 4 wires: clock, data in and data out (or bidirectional data) and a chip select. Multiple devices may be present on one bus, however, each slave device needs to have a separate "chip select" signal for selecting which device the master is talking to. SPI allows for up to 100 MHz speeds. It's commonly used as a control interface as well as a data interface. [1]

Converters with higher resolutions and higher speeds may utilize CML.

CML allows up to 12.5 Gbps speeds. That means it's possible to use fewer signals compared to both LVDS and CMOS that do not support such high speeds. This makes the board design simpler and less expensive. [1]

CML requires very few connections as the speeds may go higher than for the other standards, and the data are transferred serially. That also means a need for the introduction of a serialized data interface. One of these interfaces is JESD204. To correctly recover the clock from the data stream it's also needed to make as many changes in the data as possible. This may be done using encoding schemes such as 8b/10b (see subsection 3.3.1) or 64b/66b, leading to larger throughput for same frequency. The downside of using these encoding schemes is that they reduce the effective bandwidth. For 8b/10b encoding, which is used by JESD204B, the bandwidth is reduced to 80 % of the theoretical value. [1]

There are also greater requirements for LVDS compared to CML as it must be ensured that all of the lanes and data clock skews are not too large. JESD204B has means of aligning multiple lanes and does not require routing data clock synchronized to the data on CML lanes. The bit clocks are recovered on the receiver using Clock/Data Recovery (CDR). [1]

JESD204B includes more advanced features such as multidevice synchronization, deterministic latency, and harmonic clocking. Applications requiring these won't be able to use LVDS or CMOS standards. [1]

Both LVDS and CML need a controlled differential impedance of $100\,\Omega$ and a termination resistor of $100\,\Omega$ to remove any reflections. [1]

It's important to note that CMOS and LVDS are still being used. CML offers advantages for higher frequencies, but for lower sampling rates, CMOS and LVDS may still be employed.

Chapter 3

Description of JESD204B protocol specification

JESD204B is a protocol used for either transferring data from high-speed ADCs to FPGA/ASIC (logic device) or from FPGA/ASIC (logic device) to DACs. Its aim is to use as few connections as possible and allow for high sampling frequencies. Synchronizing multiple ADCs is possible using determined latency.

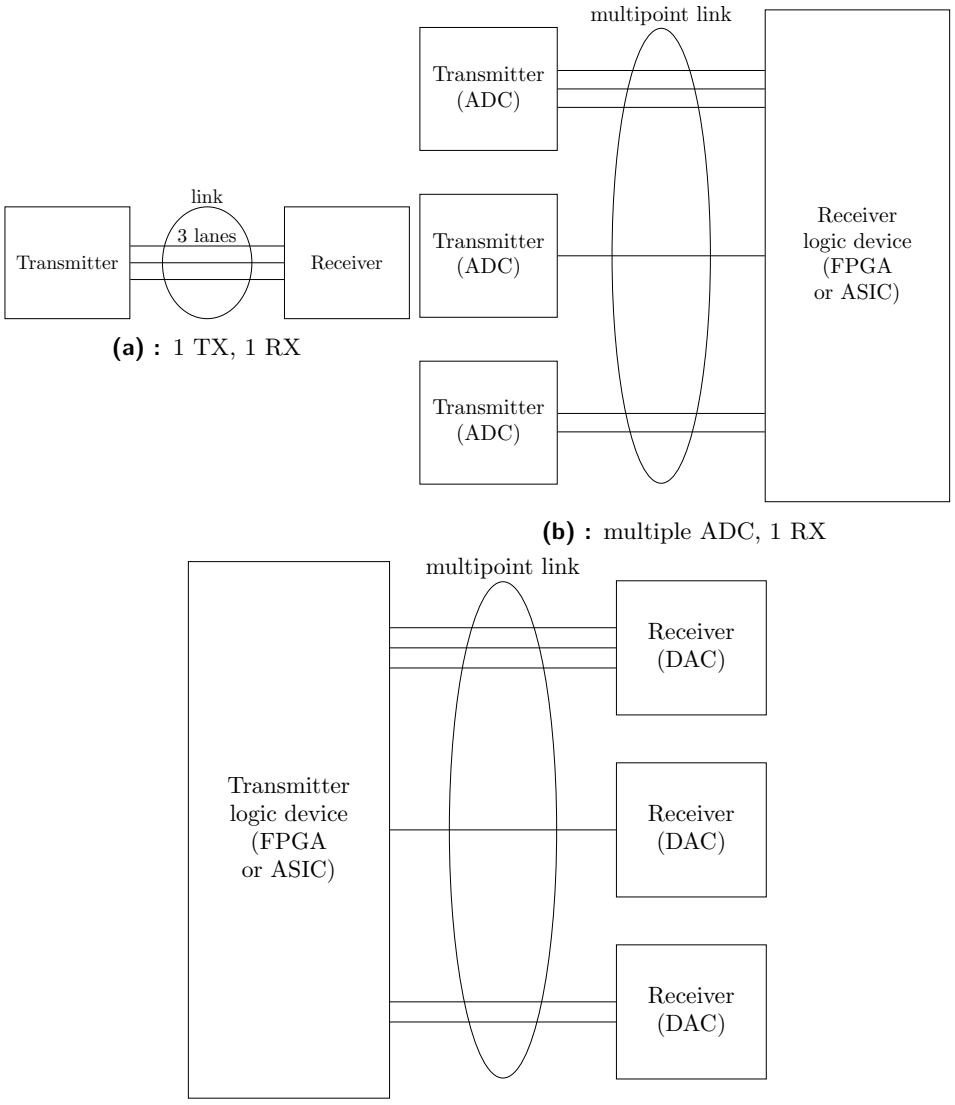
There is always only one logic device in the design. It may either be the transmitter, then it sends data to DACs. In the other case where the logic device is the receiver, ADCs are sending data to it. This is illustrated on Figure 3.1. The terminology for links, lanes, and multipoint links is demonstrated in these figures as well. ADCs or DACs may have different numbers of lanes. The channels of ADC devices are not displayed for simplicity, but it should be noted that each transmitter may contain multiple channels.

In case there are multiple ADCs, the structure is called a multipoint link. Between each ADC and FPGA, there is a link. One link may contain multiple lanes consisting of differential pairs. On these lanes, bits are transmitted using 8b/10b encoded characters.

Similar to protocols such as TCP/IP, JESD204B consists of multiple layers. The layer names and the data flow is illustrated on Figure 3.2. JESD204B has 3 subclasses that define how the data link layer behaves.

- Subclass 0 is for backward compatibility with an older version of JESD204, JESD204A. It does not support deterministic latency. [3]
- Subclass 1 supports deterministic latency by adding one more signal called SYSREF. [3]
- Subclass 2 may encode signal used for ensuring deterministic latency using SYNC$\sim$ signal that is already present in subclass 0 devices. Subclass 2 uses the same number of signals as subclass 0. [3]

Data are sent serially over one or more lanes. The number of lanes is denoted as L. Data characters are 8b/10b encoded to ensure DC balance. 8b/10b encoding ensures the same number of zeros and ones in longer periods. Data may be additionally scrambled as well to reduce EMI noise caused by



(a) : 1 TX, 1 RX

(b) : multiple ADC, 1 RX

(c) : 1 TX, multiple DAC

Figure 3.1: JESD204B link and multipoint link demonstrations.

repeating patterns. Scrambling may be disabled for applications where it does not introduce any benefit. [1]

A short introduction to what each layer on the receiver does, follows. A more thorough explanation is below.

Physical layer consists of transceivers that deserialize serial data from the transmitter. Bits are coming from differential lanes of up to 12.5 Gbps, as noted before. There is a separate transceiver for every physical lane. The output is one or more 10-bit characters going to the data link layer. [3]

JESD204B receiver has to first synchronize with data from transmitters, that is what the **Data link layer** is used for. The characters are enclosed inside frames and frames are enclosed inside multiframes. These are then used for ensuring and monitoring alignment. The data link layer also processes

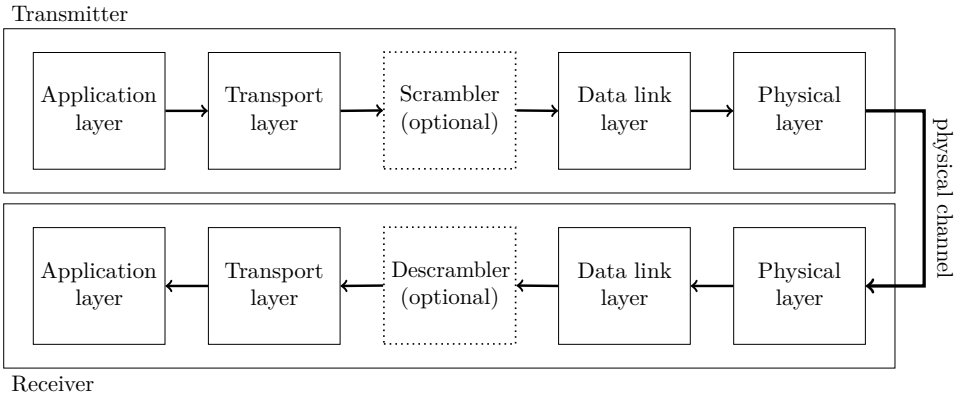


Figure 3.2: Illustration of JESD204B layers flow.

lane configuration (used to check whether the configuration of the receiver and transmitter is the same) that is transmitted during synchronization. For every lane, there is one data link, the data links need a connection between themselves to ensure alignment between the lanes. In case of multiple transmitters, another connection between links is needed for subclass 1 and 2. [4]

Lane alignment is done by waiting for a character at a given position on all the links. Until that character is received on every lane, oncoming data are stored in a FIFO starting from the same character on every lane. After every lane receives the given character, the data link is ready to start sending data. This is done to account for skews between the lanes. Subclass 0 starts sending data right after all lanes are aligned, subclass 1 and 2 work slightly differently, waiting for the right moment until releasing the data. The output of the data link layer is a frame of characters. The data goes to the transport layer. [5]

Between the data link and transport layer, there may be a (de)scrambler if the characters are scrambled. This descrambler takes and outputs a whole frame. Data during synchronization are never scrambled. [4]

Transport layer is used for converting received frames into samples from the converters. Samples consist of the sampled voltage itself, and may consist of one or more control bits. Control bits are application specific. Some common usages may be to use them for indicating overflow or underflow. The exact layout of the frame depends on the configuration of the link. [4]

Application layer is application specific. It should have control over lower layers, such as allowing to realign to the lost frame boundary or to request a synchronization if something goes wrong. The application layer on the transmitter should support generating test sequences. The receiver should be able to detect test sequences. [4]

3.1 Clocks

It's recommended that there is one source for the device clock going to each device on the multipoint link. Each device will need to generate its clocks from the main device clock. These generated clocks will be called local clocks as they are local for the given device and may have different phases in each of the devices. There are means of synchronization of some local clocks as described below. [3]

Data are transmitted over differential lanes on data/bit rate. As characters are sent as 10 bits, the character rate is the bit rate divided by 10. [3]

Each device needs to generate a frame clock, that is a clock with the frequency of a frame. In terms of character rate, it's character rate divided by F, the number of characters in a frame. [3]

Each device will need an LMFC as well, its frequency is frame clock divided by K, the number of frames in a multiframe. [3]

Devices that use subclass 1 and 2 have to have multiframe clocks aligned to each other. For subclass 1, that happens using a separate signal called SYSREF. It's recommended that SYSREF is generated from the same device that generates the device clock. For subclass 2, SYSREF is not needed and the data needed for synchronization are encoded inside of SYNC$\sim$ signal. [3]

SYSREF may be either periodic or gapped periodic. Its frequency should be an integer multiple of multiframe clock period. It's preferred to generate SYSREF using one device, but it's not required. Generating separate SYSREF for each device is possible. But it must be ensured that there is a deterministic relationship between each of these to allow for deterministic latency. [3]

3.2 Physical layer

The physical layer uses CML to transmit the data from the transmitter to the receiver across a transmission line. The physical layer on the transmitter consists of a parallel to serial converter and differential CML driver. [3]

On the receiver it is made of:

- Differential CML Receiver
- Optional Equalizer
- Clock/Data Recovery (CDR)
- Character alignment and Serial to Parallel Converter 1:10

The CDR block finds the bit boundary and aligns a bit clock to the bit boundary (recovers the clock). This bit clock may be used to generate a character clock that is utilized inside the data link layer. The character alignment should align to /K/ character boundary only upon link synchronization. The equalizer is not required but may be implemented to support sending the data across larger distances. [3]

3.3 Data link layer

In the following text, symbol aliases will be used for 8b/10b control characters. Below is a table explaining what characters these aliases map to.

Table 3.1 : Control characters from 8b/10b encoding with their aliases used in the text.

Character bits	Symbol	Alias
101 11100	/K28.5/	/K/
011 11100	/K28.3/	/A/
111 11100	/K28.7/	/F/
000 11100	/K28.0/	/R/
100 11100	/K28.4/	/Q/

The data link may receive one or more 10-bit characters at a time. Its responsibility is to decode 8b/10b characters, align all lanes, align with the frame, and output the frame. [3]

3.3.1 8b/10b encoding

As the name suggests, 8b/10b encoding encodes 8-bit data into 10-bit symbols. It's used mainly for high-speed systems that need to have the same number of 0's and 1's in the channels. That is to prevent a charge from being built up in the media and allows for easier clock data recovery as well. It's used in protocols such as PCI Express or HDMI as well. [6]

Every character is encoded either with five 1's and five 0's, or four 1's and six 0's or six 1's and four 0's. The encoder must keep track of the last symbol difference in 1's and 0's to make sure that in a long time, the data will be balanced. The difference between the number of 1's and 0's is called running disparity. It may be either +1 or -1. Every character that has an unbalanced number of 1's and 0's, has a counterpart representing the same character, but with swapped number of 1's and 0's. The transmitter will swap between the counterparts according to the current running disparity. It must keep the disparity either +1 or -1 at all times. The receiver may check whether the current character has the expected running disparity. That may be useful for detecting errors. [6]

The 8-bit data words are split into two smaller words, 3-bit and 5-bit words. 3-bit words are then encoded into 4 bits, and 5-bit words are encoded into 6 bits. This effectively means that 8b/10b encoding is decomposed into two simpler encodings, 5b/6b encoding, and 3b/4b encoding. Coded words are then joined. 5b/6b code at the top half and 3b/4b at the bottom half. [6] The decomposition of 8-bit characters and composition of 10-bit characters can be seen in Figure 3.3. The symbols are usually denoted as /D.x.y/, where x stands for a 5-bit number (0 - 31) whereas y stands for a 3-bit number (0 - 7).

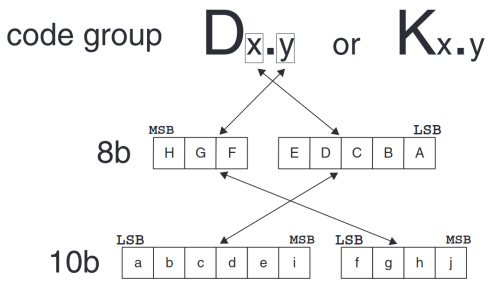


Figure 3.3: The 8b/10b coding scheme. [6]

Apart from standard data symbols, 8b/10b encoding contains control symbols. These may be used to notify the receiver about a special event. JESD204B uses control characters when synchronizing and later for alignment monitoring. Control symbols are denoted using K instead of D, ie. /K.28.y/. [6]

There are three special control symbols called comma symbols. These symbols contain a unique sequence that may allow recognizing the boundary of a character. Alignment with character boundary may take place when synchronizing the link. [6]

For JESD204B, this is done in the code group synchronization stage. The character boundary should not be changed after synchronization is established. That's to ensure that the link is not desynchronized because the data may contain an error that would be confused for a comma symbol. [3]

3.3.2 Synchronization

Synchronization of the link has two stages, code group synchronization and initial lane alignment synchronization. CGS and ILAS are never scrambled. [3] The whole process is illustrated on Figure 3.4.

3.3.3 Code group synchronization (CGS)

After losing synchronization, SYNC~ going to the transmitter should be set to notify the transmitter about lost synchronization. The synchronization may be requested from the transmitter as well, by sending /K/ symbols. The SYNC~ may be used for reporting errors after synchronization, too. That may be done by deasserting it for shorter periods of time. The transmitter will acknowledge code group synchronization on all lanes by sending /K/ characters. [3]

The receiver may get aligned with the start of a character when in CGS. That's done by special properties of /K/ comma character. After 4 consequent /K/ characters are successfully received, the receiver should deassign SYNC~. Full synchronization is assumed after receiving 4 more correct 8b/10b symbols (the character is in the encoding table and with correct disparity). Until then, any symbol error will result in the need for resynchronization. [3]

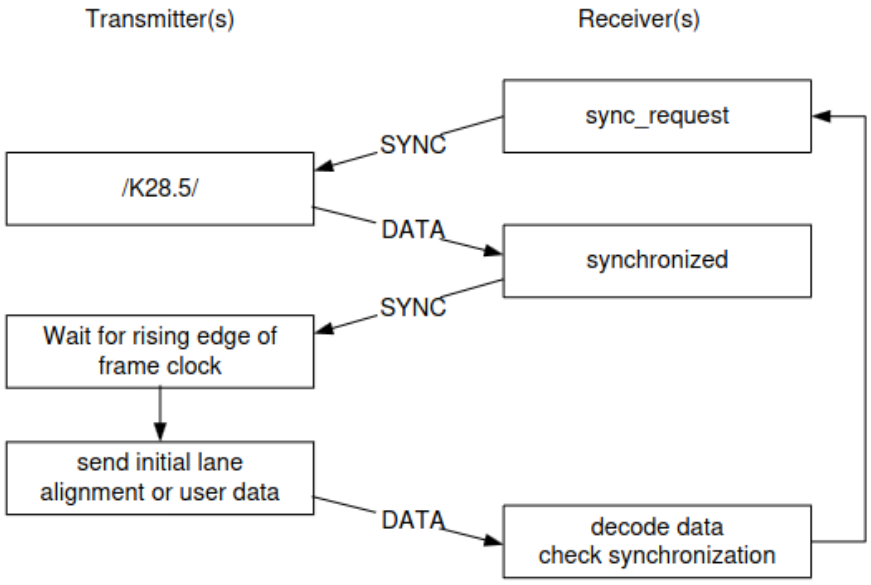


Figure 3.4: Link synchronization sequence (data link layer function chart), valid for subclass 0. [3]

Initial lane synchronization (ILS)

Initial lane synchronization will begin after the first non-/K/ character is received. That character will be at a start of a multiframe. [3]

ILAS consists of 4 multiframes. Each of the multiframes starts with /R/ and ends with /A/. The second multiframe contains link configuration data. See Table 3.3 for configuration options. The mapping of link configuration may be seen in Table 3.2. The configuration should be the same for the whole link, except for lane identification. It should match the configuration of the receiver. The configuration will follow right after /Q/ character which should be at the second position of the second multiframe. [3] From the mapping, it's obvious that there are 14 bytes needed for the configuration. Adding 1 for /Q/, 1 for /R/ (beginning of multiframe), and /A/ (end of multiframe), that makes 17 bytes the minimum number of bytes needed in one multiframe. That effectively means that $F \cdot K$ must be ≥ 17.

During ILAS the lanes in one link should get aligned to each other. That allows for the correct function of the transport layer. Right after the ILAS starts with /R/ character, data should get saved to an elastic FIFO buffer. The buffer may be released only after all lanes received the first /R/. It's possible that due to differences in the delay of the links, every lane will receive /R/ at a different time. [3] Lane alignment with FIFO buffers is demonstrated on Figure 3.5.

In subclass 0, the data will start going out of the buffers right after all lanes receive /R/. In subclass 1 or 2, data should be released 1 to K frames after multiframe clock pulse, this delay is called Rx buffer delay (RBD) (see section 3.6). [3]

Right after ILAS, the data from ADCs will follow. [3]

Table 3.2 : Mapping of configuration fields to octets in ILAS. [3]

Configuration octet no.	Bits							
	MSB	**6**	**5**	**4**	**3**	**2**	**1**	**LSB**
0	DID<7:0>							
1	ADJCNT<3:0>				BID<3:0>			
2	X	ADJDIR <0>	PHADJ <0>	LID<4:0>				
3	SCR<0>	X	X	L<4:0>				
4	F<7:0>							
5	X	X	X	K<4:0>				
6	M<7:0>							
7	CS<1:0>		X	N<4:0>				
8	SUBCLASSV<2:0>			N'<4:0>				
9	JESDV<2:0>			S<4:0>				
10	HD<0>	X	X	CF<4:0>				
11	RES1<7:0> - Set to all X							
12	RES2<7:0> - Set to all X							
13	FCHK<7:0>							

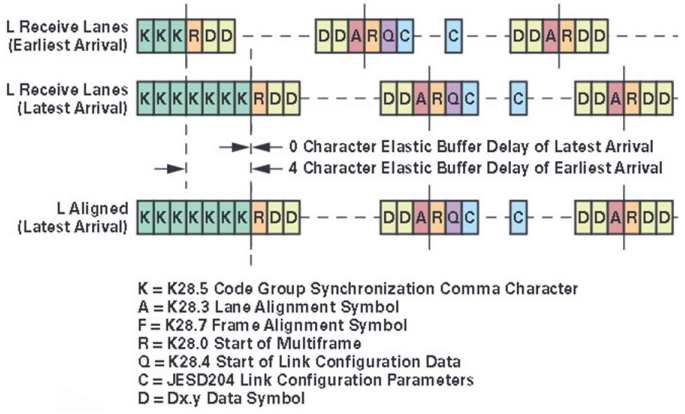


Figure 3.5: Initial lane alignment within a single link with elastic buffer demonstration. [5]

Alignment monitoring, realignment

During data transmission, there will be alignment characters inserted in special scenarios to verify the link is still aligned. If the alignment is lost, it may sometimes be restored. In cases where that's not possible, resynchronization must be requested. [3]

For non-scrambled data, if two consequent frames end with the same character, /F/ will be sent instead. The receiver has to replace that /F/ with the character from the last frame. /F/ may be only on the last position in a frame. Thus its position may be compared with the previous frame alignment character. [3]

In case two consequent multiframes end with the same character, it's similar to the case above. /A/ will be sent instead. That may be used to check lane alignment. [3]

For scrambled data, if /D28.7/ is at an end of a frame, it will get replaced by /F/ = /K28.7/. At the end of a multiframe the same is true for /D28.4/.

It will get replaced by /A/ = /K28.4/. The receiver should replace /F/ with /D28.7/ and /A/ with /D28.4/. [3]

If alignment characters (/A/, /F/) are at the wrong location, the receiver may realign. At least two alignment characters should be at the same position before realignment. If one alignment character right after realignment is detected at the previous location, the receiver should realign back to the previous position. Since data are stored in a buffer for ensuring lane alignment, the position in the buffer may be changed to restore alignment over the lanes. [3]

Table 3.3 : Fields of the link configuration for individual lanes. [3]

Parameter	**Description**
ADJCNT	Number of adjustment resolution steps to adjust DAC LMFC. Applies to subclass 2 operation only.
ADJDIR	Direction to adjust DAC LMFC, 0 - Advance, 1 - Delay. Applies to subclass 2 operation only.
BID	Bank ID - Extension to DID.
CF	Number of control words per frame clock period per link.
CS	Number of control bits per sample.
DID	Device (= link) identification number.
F	Number of octets per frame.
HD	High Density format.
JESDV	JESD204 version, 000 - JESD204A, 001 - JESD204B.
K	Number of frames per multiframe.
L	Number of lanes per converter device (link).
LID	Lane identification number (within link).
M	Number of converters per device.
N	Converter resolution.
N'	Total number of bits per sample.
PHADJ	phase adjustment request to DAC, subclass 2 only.
S	Number of samples per converter per frame cycle.
SCR	Scrambling enabled.
SUBCLASSV	Device subclass version.
RES1	Reserved field 1.
RES2	Reserved field 2.
CHKSUM	Checksum Σ(all above fields) mod 256.

3.4 Scrambling

Scrambling may prevent spectral peaks caused by repeating the same data. Every device should support scrambling. Scrambling must be enabled for the whole device, it may not be used only for some of the lanes. [3]

There is one scrambler per lane. The scrambler word size is equal to F. The bit order to a serial scrambler is illustrated in Figure 3.6. The polynomial of the self-synchronous scrambler is $1 + x^{14} + x^{15}$. The scrambler or descrambler should be situated between the data link layer and the transport layer. It's possible to implement the scrambler as either serial or parallel. [3]

Some of the scrambled data will be lost when the descrambler initially doesn't have state registers synchronized. This loss may be prevented by allowing unscrambled octets to flow through the state register as well. [3]

Initial lane alignment sequence as well as code group synchronization sequence are never scrambled. [3]

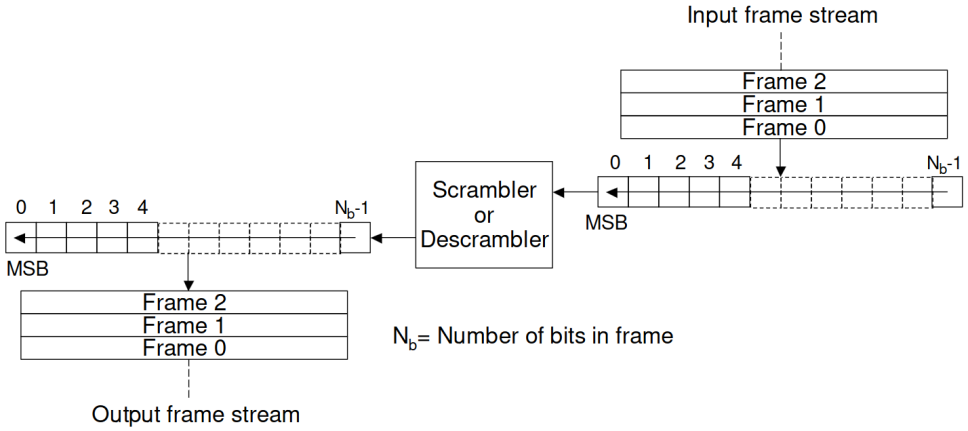


Figure 3.6: Serial scrambling bit order. [3]

3.5 Transport layer

The transport layer operates on frame clock frequency. Its responsibility is to map octets from all of the lanes to raw samples. Each sample may have associated control bits. [3]

Every frame contains samples from all of the converters. One frame may contain more than one sample for every converter, that is called oversampling. [3]

For one lane, samples are mapped to a linear axis starting with the first converter. If oversampling is enabled, all of the samples from the first converter are added as first. Then samples from the rest of the converters follow. Sample words are padded to multiples of 4 bits. 4 bits create a nibble group. There is a possibility to enable a high-density mode that will work without inserting tail bits. Samples may be directly followed by control bits.

Another possibility is to send control bits at the end of a frame as control words. The control bits in a control word are mapped the same way as samples. [3]

For multiple lanes, the process is the same. At the end, the data is split into multiple lanes. Every lane will contain F characters per frame. There will be a total of $L \cdot F$ characters in one frame. See Figure 3.7 for visualization of the mapping. [3]

To prevent the tail bits from reducing the generation of frame synchronization symbols, they should meet one of the requirements:

- The sequence is the same for all frames. [3]
- The sequence is generated pseudo-randomly based on a polynomial that has a degree of at least 9. [3]

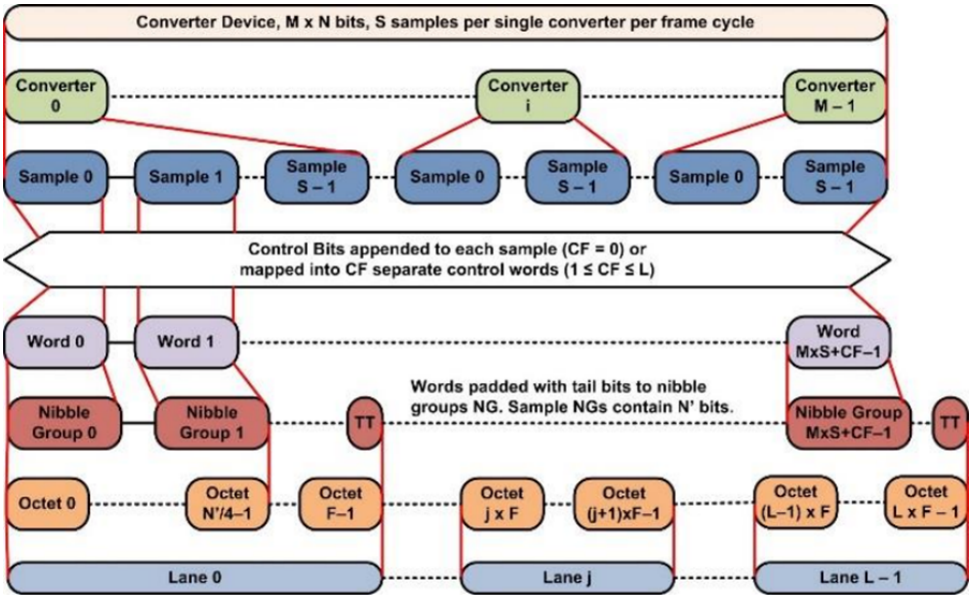


Figure 3.7: Transport layer samples to lane octets decomposition. [4]

3.6 Deterministic latency

There are means defined in the JESD204B standard to allow for synchronization of some of the local clocks between the devices and also to allow for ensuring a deterministic latency between sampling and receiving the data. Ultimately leading to synchronization between multiple transmitter device samples. [3]

According to JEDEC standard description, the definition of the deterministic latency is: "The deterministic latency across the link is defined from the parallel frame-based data input on the TX device to the parallel frame-based

data output on the RX device, all measured within the frame clock domain." [3] This definition is illustrated on Figure 3.8

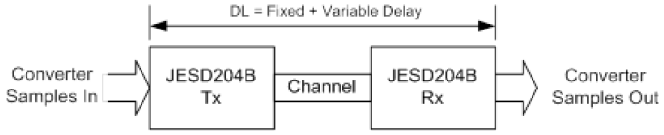


Figure 3.8: Deterministic latency definition.

The deterministic latency is a sum of fixed and variable delays. Fixed delays are a result of circuit design and are constant from one power cycle to another. Variable delays may be caused by various factors and are dependent on the power cycle. [5]

To achieve deterministic latency, there are two requirements:

- The transmitters should begin sending ILAS at a well-defined moment in time. [3]
- The receiver should buffer the incoming data to allow data from all lanes to be received. As each lane may have different skew, the data won't arrive simultaneously. The buffers should be released at a well-defined time. [3]

A well-defined moment in time at which transmitters should begin initial lane alignment, is the first LMFC tick after SYNC∼ is deassigned. A well-defined moment in time at which the receiver should release the buffers is a whole number of frame cycles after an LMFC tick. The number of frame cycles should be programmable and is called Rx buffer delay (RBD). [3] The timing diagram for achieving deterministic latency is illustrated on Figure 3.9.

For proper performance of the deterministic latency, some requirements must be followed:

- The period of a multiframe must be larger than the maximum delay between transmit and receive devices. [3]
- The value of RBD multiplied by the frame period must be larger than the maximum delay. [3]
- The RBD must fit into a multiframe. [3]

Subclass 0 does not support deterministic latency. The variable delays cannot be accounted for as there are no means for that present. [3] It still may be possible to synchronize multiple ADCs in the application layer though. JESD204B allows for control bits to be added to the sample bits. If there is a SYSREF-like signal distributed to all of the devices and the ADCs are sending a control bit along the samples that indicates the SYSREF edge has been encountered, it's possible to align multiple ADCs by aligning samples with the indication. [5]

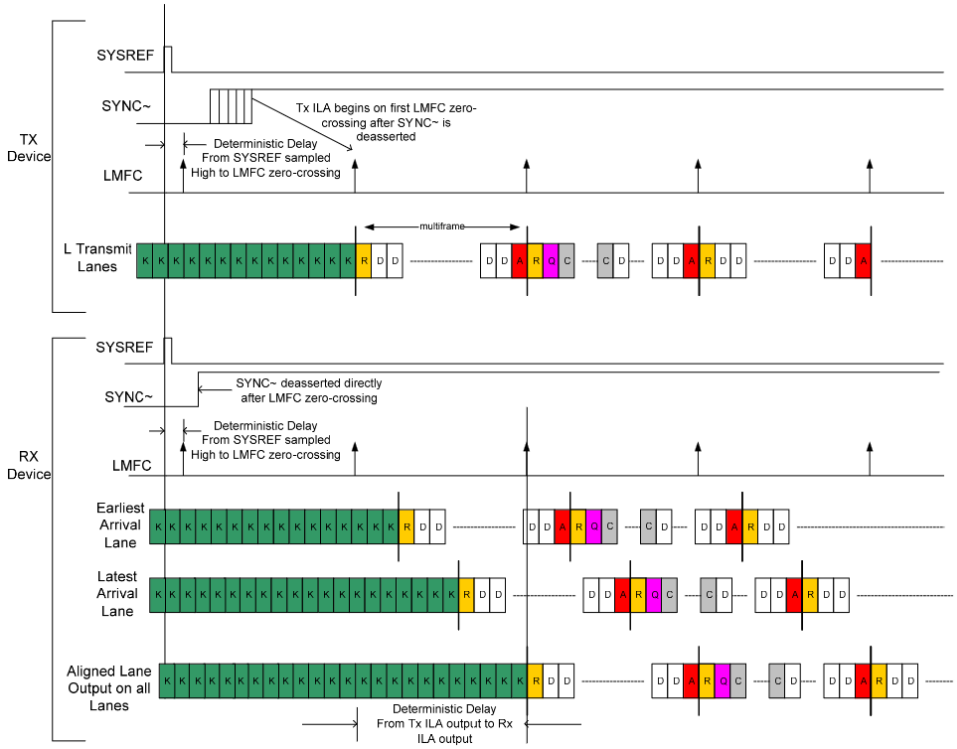


Figure 3.9: Timing diagram illustration for deterministic latency equal to multiple of multiframe period. [3]

3.6.1 Subclass 1

Subclass 1 allows for deterministic latency by utilizing the SYSREF signal. LMFC and frame clock realignment are necessary only when the link is being synchronized. [3]

All devices should be able to issue a request for SYSREF generation. That may be usable if the SYSREF is a gapped periodic or one-shot signal. The request should be issued upon link synchronization. The devices should adjust LMFC and frame clock boundary on some of the SYSREF rising edges. The exact alignment moment is left to the implementer. For example, the device may be instructed to use the next detected SYSREF pulse to force the alignments through a configuration interface. This idea is illustrated on Figure 3.10 [3]

The delay between SYSREF being sampled and LMFC rising edge should be specified for each device. [3]

3.6.2 Subclass 2

In subclass 1 the deterministic latency is achieved by aligning LMFC between the devices. It is achieved the same way in subclass 2, but the alignment part is different. Whereas in subclass 1 another external signal, SYSREF, is needed, in subclass 2 the data needed for LMFC alignment are inside of the

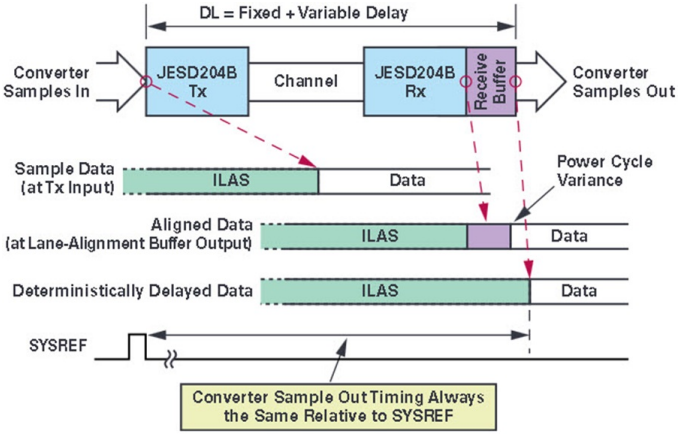


Figure 3.10: Data release timing using SYSREF in a subclass 1. [5]

SYNC~ signal. [5]

The SYNC~ signal generated by the receiver must be generated based on the receiver's LMFC. That will mean it will carry LMFC information to the transmitter. This information may then be used for aligning LMFC on all of the devices. [5]

This means that the master reference is generated from a different source, in subclass 1 the master is the clock source, whereas in subclass 2, it's the master logic device. The implementation is different for both ADCs and DACs. [5]

ADC subclass 2 deterministic latency

The SYNC~ is de-asserted by the logic device (receiver in this case) and sampled by the ADC. When the ADC detects the deassignment of SYNC~, it resets its LMFC. The transmitter will begin sending /K/ characters until the clock is settled. After the clocks have settled, the ADC will begin sending ILAS. It's possible to utilize a periodic SYNC~ to monitor the phase alignment of the ADC's LMFC. [5]

DAC subclass 2 deterministic latency

For DACs, the logic device is the transmitter. It's not possible to align the transmitter to all of the DACs, as the clock phase may be different for each of them. This presents a challenge as there is not a separate indication signal going from the transmitter to the DACs. The DACs will deassign the SYNC~ on their LMFC edge. The logic device may detect the phase difference between its LMFC and the DAC LMFC. It may issue a command to the DAC to adjust the phase during ILAS. The relevant parts of the link configuration are PHADJ (phase adjust), ADJCNT (adjustment count), and ADJDIR (adjustment direction). [5]

- PHADJ indicates whether an adjustment is needed.

- ADJCNT indicates the number of adjustment steps needed.
- ADJDIR indicates what direction to adjust the LMFC to.

It's possible that the adjustment will have to be repeated mutliple times. After each adjustment, the DAC issues an error report by deasserting the SYNC∼. (again aligned with LMFC) If misalignment is detected, the ILAS will be issued with PHADJ request again and the process will be repeated. In other case, ILAS will be issued without PHADJ. Data may be sent after the ILAS. [5]

The deterministic latency is achieved the same way as in subclass 1 after the LMFCs are aligned.

3.7 Test modes

To test the link, the JESD204B specification includes a predetermined sequences of characters that every transmitter should be able to send. Some ADCs allow for specifying a user test pattern. The sequences that every transmitter should be able to send are:

- Continuous sequence of /D21.5/ characters (1010101010). [3]
- Continuous sequence of /K/ characters. [3]
- Repeated transmission of lane alignment sequence. The transmitter should be able to initiate code group synchronization upon receiving a synchronization request. [3]
- Continuous sequence of a modified random pattern or scrambled jitter pattern. [3]

All receivers should be able to verify some of these sequences. Namely a continuous /K/ sequence and a repeated lane alignment sequence. [3]

Chapter 4

Implementation of the receiver

The implementation was done in VHDL. The code has been tested using testbenches for each component itself and for the whole receiver connecting all of the components. It's possible to start these testbenches using ghdl and view the result inside gtkwave using make, Makefile is located in the root folder. Another possibility is to use the Quartus project that is also in the root folder and all testbenches are present there, it's possible to run these in Modelsim.

A short introduction to each of the programs or tools mentioned, follows.

GHDL is an open-source compiler that may simulate VHDL code. It directly translates/compiles VHDL code to machine code, so an executable program is created. [7]

Gtkwave is an open-source wave viewer for Unix and Win32. GHDL program may produce a standard Verilog VCD file that may be explored using Gtkwave. [8]

Modelsim is a simulation tool from Siemens, it can simulate some of the hardware description languages. Among other languages, it supports VHDL or Verilog.

Quartus Prime is a software bundle developed by Intel. It supports everything for designing on Intel FPGAs, such as synthesizing. It is able to simulate VHDL code as well. [9]

The root folder of the custom implementation contains `src/` folder containing all the sources. The sources are grouped into folders by layer. Testbench folder `testbench/` has the same structure. Testbenches are named the same way as sources with added `_tb` at the end. When there is need to test multiple configurations of one entity, the file names may be different. The files are located on attachment for this thesis and the file tree structure of the attachment is located in Appendix A.

Care was taken to make all of the entities modular using generics. Every entity may be used with different configurations to accommodate for any possible usage of the JESD204B protocol. The design is split into smaller entities to make it possible to use only smaller parts of the design, if there was need to replace some of the entities.

4.1 VHDL introduction

VHDL is a hardware description language. VHDL code specifies a structure of a digital circuit. The code may be synthesized to a digital circuit and put on an Field programmable gate array (FPGA). The resulting circuits may as well be simulated using specialized software. There are some spin-offs of VHDL language by different vendors. [10] VHDL derives syntax from a language called Ada.

VHDL is similar to programming languages. Any algorithm may be written using VHDL, but its main purpose is to describe hardware. VHDL may be executed, similar to other programs. But it's common to call the process of execution simulation for VHDL. That is because VHDL is mainly for modeling designs. [7]

To use VHDL on real hardware, it must be synthesized. Synthesis tool transforms a program into a gate-level description. [7]

4.2 Design

The following section should act as a sort of a documentation of the VHDL implementation. It mainly shows the conception of the code and describes the inner workings of some of the more complex components. The design contains top level entities that may be used as a standalone entity representing the whole JESD204B receiver, even for a multipoint link consisting of multiple transmitters. The structure of these components is discussed to show how to compose an entity like that from the individual components.

4.2.1 Defined record types

VHDL allows to make user defined record types that store multiple fields of specified types. There are some records in the implementation to ease transferring data between entities.

One of these is a link config, fields of the link config are specified in the table below. Link config contains JESD204B configuration fields of a link that can be seen in Table 3.3.

Another custom record type is a link character containing 8-bit character information, the 8-bit character itself, whether it's a control character and whether there was a disparity or not in table error.

For the transport layer, there is a record called frame state, it contains information about errors in a frame, whether the frame contains user data (data from the converters), whether there were some errors for the 8-bit decoder or later. It also tells whether the last frame was repeated, the frame gets repeated in case there is an error.

Every testbench contains a custom record type as well, containing inputs to the unit under test and usually expected output as well.

Table 4.1 : Fields of defined link_character record type.

Name	Type	Description
kout	std_logic	Whether the character is a control character
disparity_error	std_logic	Disparity does not match
missing_error	std_logic	Not in 8b10b encoding table
d8b	std_logic	8-bit character
user_data	std_logic	Whether the character is from data state or is from synchronization

Table 4.2 : Fields of defined frame_state record type.

Name	Type	Description
user_data	std_logic	Whether the frame consists only of samples
invalid_characters	std_logic	Whether there are any characters that shouldn't be in the frame
not_enough_data	std_logic	An error stating there isn't enough data to output a whole frame
ring_buffer_overflow	std_logic	An error stating that there isn't enough data to output a whole frame
disparity_error	std_logic	An error stating that there was a disparity mismatch in the frame
not_in_table_error	std_logic	An error stating that some of the characters weren't present in 8b/10b encoding table
wrong_alignment	std_logic	An error stating alignment characters were found on unexpected position
last_frame_repeated	std_logic	Whether a frame has been repeated, due to errors

4.2.2 Top level entities

Top level entities are just wrappers of other entities without a lot of logic added. They allow for easier implementation of JESD204B standard to new design as well as show how to link the lower level components correctly.

JESD204B multipoint receive link

Multipoint receive link is a component that connects multiple JESD204B link receive components (see subsubsection 4.2.2). It may generate multiframe clock and align it with the SYSREF signal. Subclasses 0 and 1 are supported. It should be possible to use this component as a top level entity for data link and transport layers in any design with one or more JESD204B links. It expects 10-bit characters from the lanes directly on its input, as di_data input, shown on Figure 4.1. Samples from the converters are on the output of this component.

The inside of this component is quite simple, as it just connects jesd204b_link_rx, synced_combination and lmfc_generation.

Table 4.3 : Input and output ports of jesd204b_multipoint_link_rx VHDL entity.

Name	Description
ci_device_clk	Device clock
ci_char_clk	Character clock
ci_frame_clk	Frame clock
ci_sysref	Sysref signal
ci_reset	Asynchronous reset (active low)
ci_request_sync	Externally request synchronization
co_nsynced	Indicates correct synchronization (active low)
co_error	Indicates any kind of error
di_data	Input data from transceivers, 10-bits for every lane
do_samples	Output samples, valid if frame_state is okay
do_ctrl_bits	Output control bits, valid if frame_state is okay
co_frame_state	Combined state of all of the links and lanes
co_correct_data	Indication coming from co_frame_state, indicating whether the data are okay

Table 4.4 : Generic parameters of jesd204b_multipoint_link_rx VHDL entity.

Name	Description
K_CHAR	The 8-bit /K/ character
R_CHAR	The 8-bit /R/ character
A_CHAR	The 8-bit /A/ character
Q_CHAR	The 8-bit /Q/ character
DATA_RATE	Multiple of device clock to get data rate
MULTIFRAME_RATE	$F \cdot K$
ALIGN_BUFFER_SIZE	Size of the buffer for aligning lanes
RX_BUFFER_DELAY	Number of frames to wait before releasing buffer (for subclass 1)
LINKS	Number of links on the multipoint link
LANES	Total number of lanes
CONVERTERS	Total number of converters
CONFIG	The configuration for each link
ERROR_CONFIG	The configuration for error_handler

JESD204B link receive

This entity encapsulates multiple data_link_layer entities, a descrambler and a transport_layer for one JESD204B link. It receives a local multiframe clock as an input, seen on Figure 4.2, and thus may not generate multiframe clock by itself. For generating LMFC, lmfc_generation should be used. Or alternatively, the multipoint top level entity may be used instead.

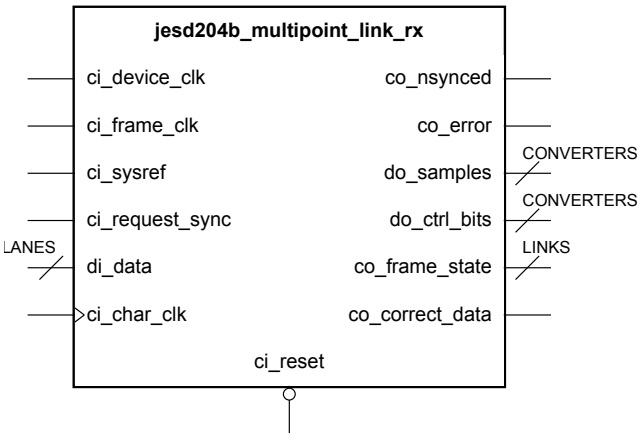


Figure 4.1: Diagram of JESD204B multipoint link receive VHDL entity.

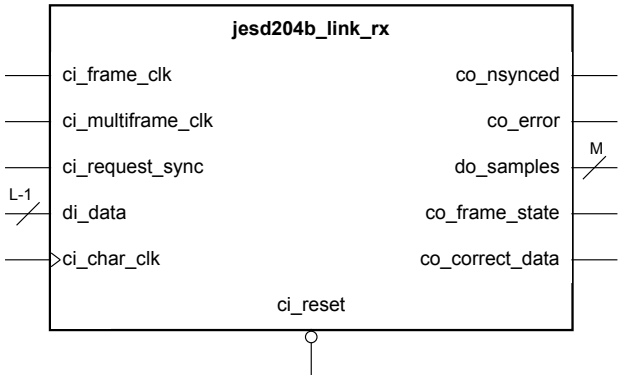


Figure 4.2: Diagram of JESD204B link receive VHDL entity.

Table 4.5 : Input and output ports of jesd204b_link_rx VHDL entity.

Name	Description
ci_char_clk	Character clock
ci_frame_clk	Frame clock
ci_multiframe_clk	Multiframe clock
ci_reset	Asynchronous reset (active low)
ci_request_sync	Externally request synchronization
co_nsynced	Whether the link is synchronized (active low)
co_error	Whether the output data are errorful
di_data	10 bit characters for each lane
do_samples	Samples in a frame
do_ctrl_bits	Samples control bits in a frame
co_frame_state	State of the frame
co_correct_data	Whether samples are without an error

Table 4.6 : Generic parameters of jesd204b_link_rx VHDL entity.

Name	Description
K_CHAR	The 8-bit /K/ character
R_CHAR	The 8-bit /R/ character
A_CHAR	The 8-bit /A/ character
Q_CHAR	The 8-bit /Q/ character
Link config	See Table 3.2
ALIGN_BUFFER_SIZE	lane alignment FIFO buffer size
RX_BUFFER_DELAY	Number of frames to wait until releasing lanes (used for subclass 1)
ERROR_CONFIG	Configuration for error handler
SCRAMBLING	Whether data are scrambled

4.2.3 Helpers

There are some entities used inside of top level entities to split logic that does not go into any of the layers directly.

Sync combination

Sync combination component is used in top level entities for the link and multipoint link. Its function is to take multiple sync signals, as may be observed on Figure 4.3, and combine them into one sync output signal.

It respects JESD204B specification timing requirements for different subclasses. For subclass 0, sync combination will set sync signal only on frame clock, for subclass 1, sync combination will set sync signal only on multiframe clock and only in case the LMFC is aligned.

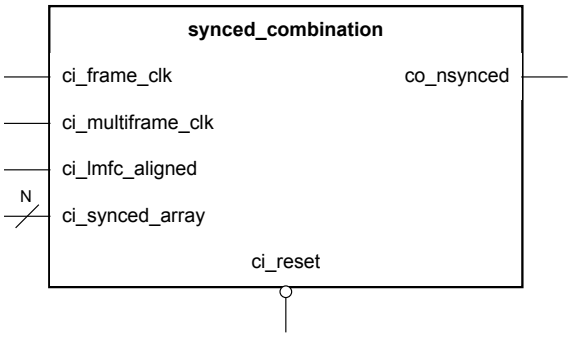


Figure 4.3: Diagram of synced combination VHDL entity.

Table 4.7 : Input and output ports of synced_combination VHDL entity.

Name	Description
ci_frame_clk	Frame clock
ci_multiframe_clk	Multiframe clock
ci_reset	Asynchonous reset (active low)
ci_lmfc_aligned	Whether multiframe clock is aligned

Table 4.8 : Generic parameters of synced_combination VHDL entity.

Name	Description
SUBCLASSV	What subclass version to use
N	Number of outputs (usually number of links)
INVERT	Whether synced input is inverted (1 = active low sync is on input)

LMFC generation

This component manages LMFC generation. It uses LMFC counter entity for generating the local clock itself. LMFC generation tells the counter to resynchronize in case the link is not synchronized. It outputs whether LMFC was aligned after synchronization was lost. Sync generation takes that input to tell the transmitter the link is synchronized only after LMFC is aligned. This component has a SYSREF signal as its input, see that on Figure 4.4, that works as a reset of the counter, to align the multiframe to the SYSREF.

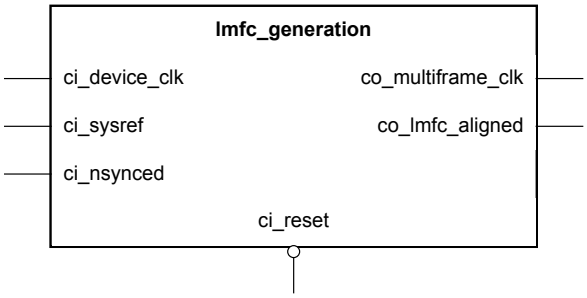


Figure 4.4: Diagram of LMFC generation VHDL entity.

Table 4.9 : Input and output ports of lmfc_generation VHDL entity.

Name	Description
ci_device_clk	Device clock
ci_reset	Asynchronous reset (active low)
ci_sysref	Sysref signal
ci_nsynced	Whether synchronization is established (active low)
co_multiframe_clk	Aligned mutliframe clock
co_lmfc_aligned	Whether clock is aligned

Table 4.10 : Generic parameters of lmfc_generation VHDL entity.

Name	Description
MULTIFRAME_RATE	Number of device clocks in a multiframe, $F \cdot K$
DATA_RATE	Multiplication factor of serial data speed relative to device clock

LMFC counter

LMFC counter entity is a simple counter with the possibility to reset its count so multiframe clock may be synchronized to SYSREF signal rising edge. See the input and output ports on Figure 4.5.

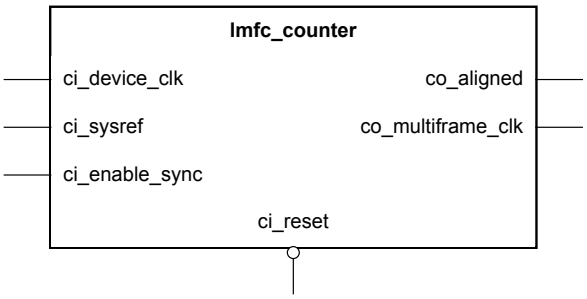


Figure 4.5: Diagram of LMFC counter VHDL entity.

Table 4.11 : Input and output ports of lmfc_counter VHDL entity.

Name	Description
ci_device_clk	Device clock
ci_reset	Asynchronous reset (active low)
ci_sysref	Sysref signal
ci_enable_sync	Whether to align clock to next sysref
co_aligned	Whether clock is aligned to sysref
co_multiframe_clk	Aligned multiframe clock

Table 4.12 : Generic parameters of lmfc_counter VHDL entity.

Name	Description
DATA_RATE	Multiple of device clock to get data rate
PHASE_ADJUST	How many device clock ticks to wait after SYSREF
MULTIFRAME_RATE	$F \cdot K$

4.2.4 Data link layer

A description of entities that compose the data link layer will be introduced now.

The data link layer receives 10-bit character for every lane on its input, as may be observed on Figure 4.7, and outputs a frame along with its state.

The Data link layer entity is a composite entity that contains entities that together form a data link. It receives 10-bit characters and outputs aligned frames of 8-bit characters combined with a frame state that may report any errors encountered. It deals with the synchronization of the link. It detects errors and acts according to configuration, requesting another synchronization in some cases.

Data flow inside of a data link layer is illustrated on the diagram on Figure 4.6. First, the data goes into a character alignment that finds the boundary of /K/ character upon synchronization. After that, the data are decoded using 8b/10b decoder. From that, the data go to the link controller and lane alignment. Lane alignment stores the data in FIFO buffer to align all of the lanes on the link. At last, after the data go through lane alignment, they will go to frame alignment. Frame alignment finds the frame boundary and outputs whole frames, aligned with that boundary.

Table 4.13 : Input and output ports of data_link_layer VHDL entity.

Name	Description
ci_char_clk	Character clock
ci_frame_clk	Frame clock
ci_reset	Asynchronous reset (active low)
do_lane_config	The configuration parsed from ILAS
co_lane_ready	Whether ILAS has started (/R/ received)
ci_lane_start	Set to release lane alignment FIFO buffers
ci_request_sync	Externally request synchronization of the link
co_synced	Whether synchronization is established
di_10b	10-bit character input from transceivers
do_aligned_chars	Aligned frame characters
co_frame_state	State of the frame being outputted (disparity errors, not in table errors, not aligned)

ILAS parser

ILAS parser detects the start of an ILAS sequence, reports errors in the sequence, and parses configuration from the sequence. The parser is used inside of the link controller entity. The parser must know that the current state is CGS in order to know when to look for /R/ character that denotes the start of the ILAS sequence. Skipping the ILAS sequence is currently not supported. See Figure 4.8 for input and output ports.

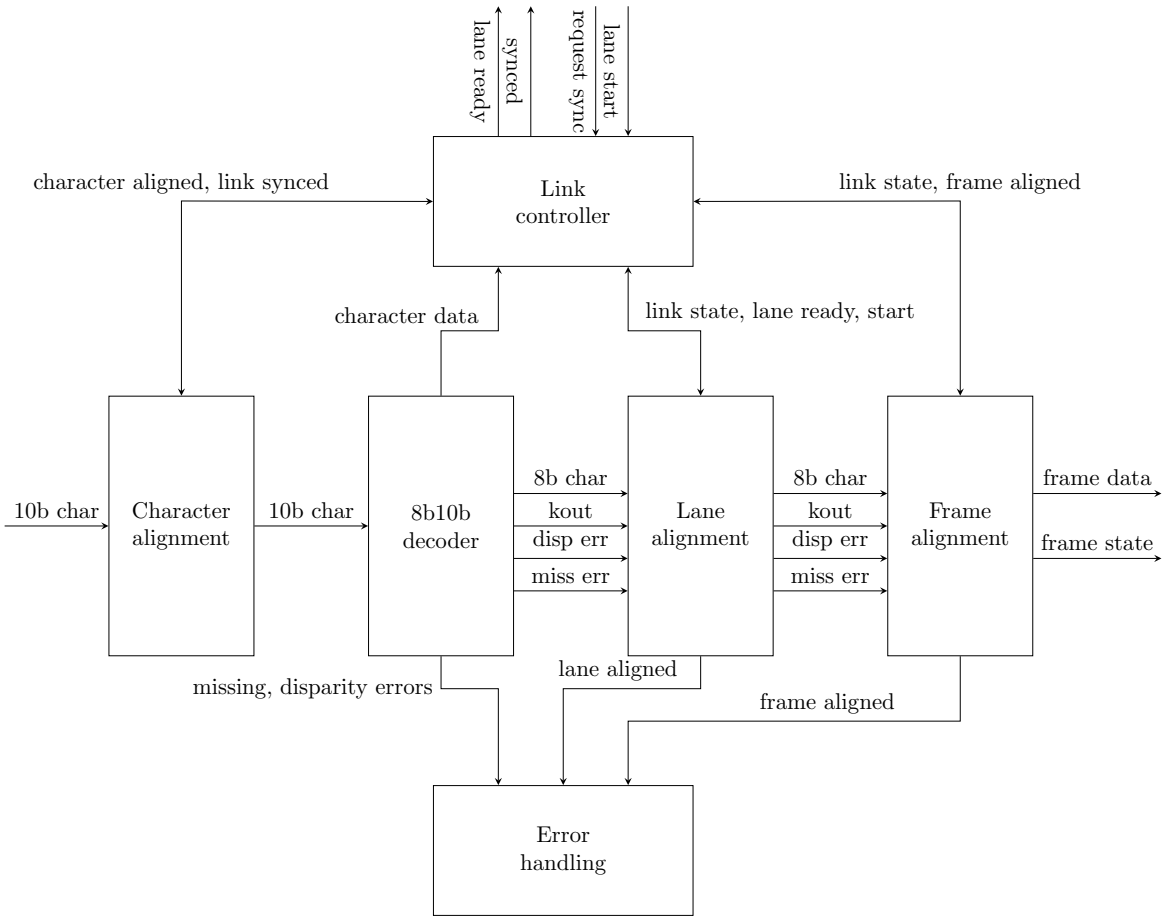


Figure 4.6: Block diagram of the data link layer VHDL entity.

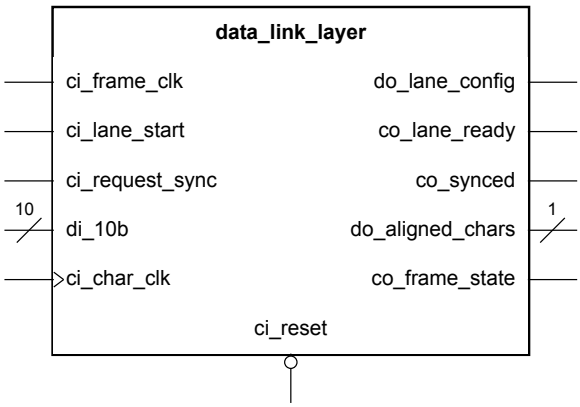


Figure 4.7: Diagram of data link layer VHDL entity.

Table 4.14 : Generic parameters of data_link_layer VHDL entity.

Name	Description
K_CHAR	The 8-bit /K/ character
R_CHAR	The 8-bit /R/ characterr
A_CHAR	The 8-bit /A/ character
Q_CHAR	The 8-bit /Q/ character
ALIGN_BUFFER_SIZE	Size of the FIFO lane alignment buffer
ERROR_CONFIG	Configuration for error handler
SCRAMBLING	Whether scrambling is enabled
SUBCLASSV	What subclass version to use
F	Number of characters in a frame
K	Number of frames in a multiframe

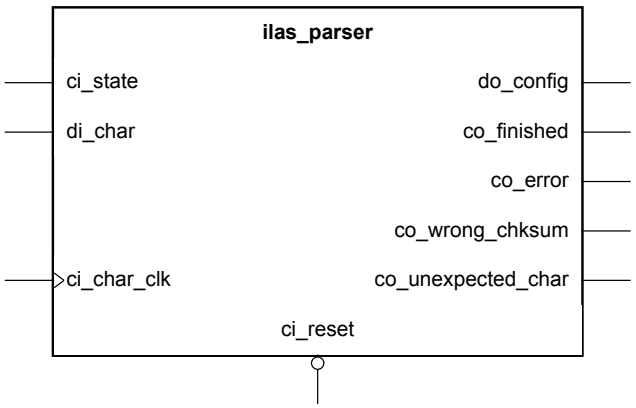


Figure 4.8: Diagram of ILAS parser VHDL entity.

Table 4.15 : Input and output ports of ilas_parser VHDL entity.

Name	Description
ci_char_clk	Character clock
ci_reset	Asynchronous reset (active low)
ci_state	State of the link
di_char	8-bit decoded character
do_config	Config that found in the ILAS sequence
co_finished	Whether sequence has been successfully finished
co_error	Whether there was an error in the sequence

Table 4.16 : Generic parameters of ilas_parser VHDL entity.

Name	Description
F	Number of characters in a frame
K	Number of frames in a multiframe
K_CHAR	The 8-bit /K/ character
R_CHAR	The 8-bit /R/ character
A_CHAR	The 8-bit /A/ character
Q_CHAR	The 8-bit /Q/ character

Link controller

This entity is responsible for controlling the synchronization process. It outputs SYNC~ signal according to JESD204B specification. The state of the link is outputted as well, as co_state (seen on Figure 4.10), so it may be used in other data link layer components as well. It parses ILAS with the help of the ILAS parser entity.

The link controller has multiple states that represent the state of the whole data link layer, changing behavior in most of the entities. The state diagram is illustrated on Figure 4.9 Upon initialization, the component is in INIT state and it will go to CGS state after at least one /K/ character is received. The state will be set to CGS upon receiving /K/ in any state. After receiving at least 4 /K/ characters correctly, SYNC~ will be assigned. The ILS state will be set according to the ilas_parser component. After the ilas_parser component outputs that the sequence was correct, the state will be set to DATA, indicating that the synchronization was successful and user data are being sent.

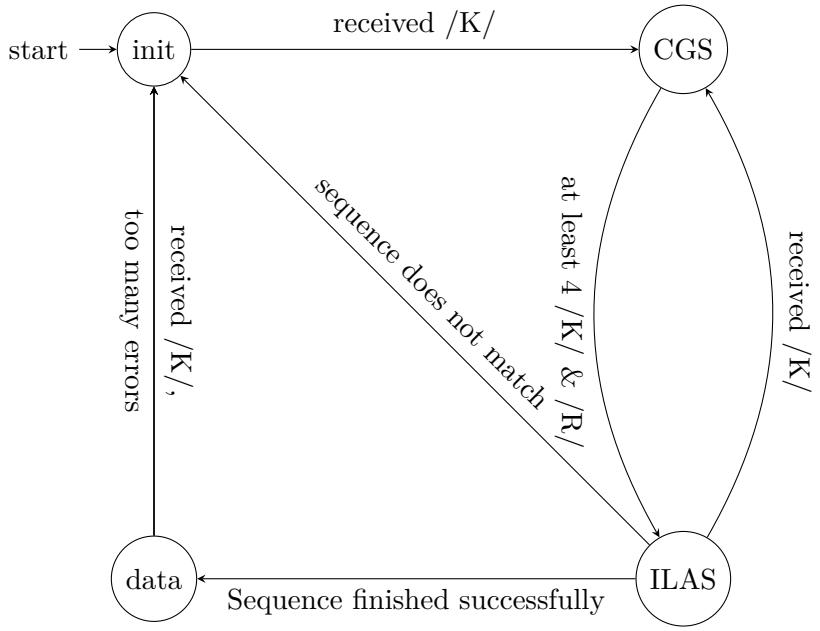


Figure 4.9: Link controller states.

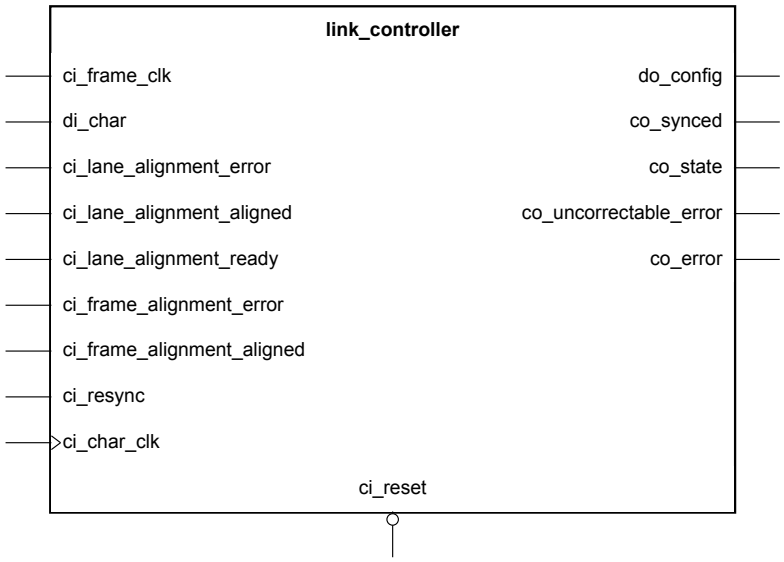


Figure 4.10: Diagram of link controller VHDL entity.

Table 4.17 : Input and output ports of link_controller VHDL entity.

Name	Description
ci_frame_clk	Frame clock
ci_char_clk	Character clock
ci_reset	Asynchronous reset (active low)
di_char	8-bit decoded character
do_config	Configuration parsed from ILAS sequence
ci_lane_alignment_error	Whether lane alignment detected /A/ character
	on wrong position
ci_lane_alignment_aligned	Whether alignment to multiframe is acquired
ci_lane_alignment_ready	
ci_frame_alignment_error	Whether frame alignment detected /F/ character
	on wrong position
ci_frame_alignment_aligned	Whether alignment to frame is acquired
ci_resync	Externally request synchronization
co_synced	Whether the link is synchronized
co_state	State of the link
co_uncorrectable_error	Indicates an uncorrectable error
co_error	Indicates any error

Table 4.18 : Generic parameters of link_controller VHDL entity.

Name	Description
SUBCLASSV	What subclass version to use
F	Number of characters in a frame
K	Number of frames in a multiframe
K_CHAR	The 8-bit /K/ character

Character alignment

Character alignment entity is especially important in the code group synchronization stage. It aligns with the beginning of /K/ character. Aligning with the start of a character is possible to be done in the physical layer instead. As /K/ is a comma character, it contains a unique sequence (1111101 or 0000010) that is not encountered anywhere else. This sequence may be used for quick detection of the boundary of a character. After the character is found, co_aligned, that may be seen on Figure 4.11, will indicate that. Characters going to 8b/10b decoder should be aligned correctly then.

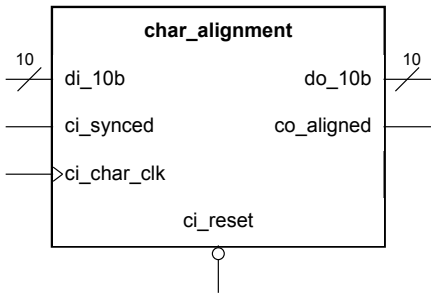


Figure 4.11: Diagram of character alignment VHDL entity.

Table 4.19 : Input and output ports of char_alignment VHDL entity.

Name	Description
ci_char_clk	Character clock
ci_reset	Reset (active low)
di_10b	10-bit data to align to a character
ci_synced	Indicates that the link is synchronized
do_10b	10-bit aligned character
co_aligned	Character alignment status, was /K/ found?

Table 4.20 : Generic parameters of char_alignment VHDL entity.

Name	Description
K_CHAR	The character to align to (/K/)

8b10b decoder

This entity takes 10-bit characters, as seen on Figure 4.12, and decodes them according to the 8b/10b decoding table. It checks whether the disparity of the current character is the expected one. In case it isn't, an error is outputted, using do_char (of link_character type), but the data are decoded correctly. In case the input character is not found in the decoding table, the error is outputted and the data are incorrect.

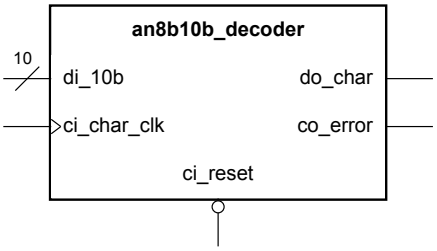


Figure 4.12: Diagram of 8b10b decoder VHDL entity.

Table 4.21 : Input and output ports of an8b10b_decoder VHDL entity.

Name	Description
ci_char_clk	Character clock
ci_reset	Reset (active low)
di_10b	10-bit character
do_char	8-bit character with errors (disparity, not in table)

Lane alignment

The lane alignment entity has a FIFO buffer inside of it. This buffer is used when the link is being synchronized. The entity waits for the first /R/ character and from that character starts storing oncoming characters until a release signal is set. Detection of the /R/ character is indicated by co_ready, an output seen on Figure 4.13. After that, buffers are released from the first /R/ character. That ensures all lanes may start at a well-defined position, the first /R/ character in ILAS sequence.

According to the JESD204B standard, the lane alignment should be monitored after synchronization is established using /A/ characters. Unfortunately, this behavior was not implemented completely as of the date of submission. Part of that behavior is replaced by frame_alignment entity.

Frame alignment

- Upon synchronization, it finds /A/ or /F/ in the data to align to a frame. It stores incoming characters in a ring buffer and releases them when the frame clock ticks. The component releases whole frames. It aligns to the correct beginning and end of the frame using /A/ or /F/ characters.

Table 4.22 : Input and output ports of lane_alignment VHDL entity.

Name	Description
ci_char_clk	Character clock
ci_reset	Asynchronous reset (active low)
ci_start	Tells to release the lane buffer
ci_state	State of the link
di_char	8-bit decoded character
co_ready	Whether /R/ (start of ILAS) was detected
do_char	8-bit character released from the buffer or DUMMY_CHAR until the buffer is released

Table 4.23 : Generic parameters of lane_alignment VHDL entity.

Name	Description
F	Number of characters in a frame
K	Number of frames in a multiframe
BUFFER_SIZE	Size of the FIFO buffer
R_CHAR	The 8-bit /R/ character
DUMMY_CHAR	Character to send until buffer is released

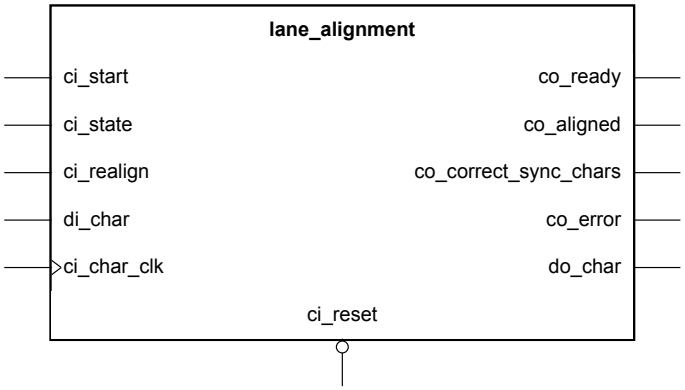


Figure 4.13: Diagram of lane alignment VHDL entity.

- After synchronization, it monitors whether /F/ and /A/ characters are at expected positions, that means the end of a frame (or multiframe).
- It outputs information about how many characters are at the wrong position and allows realigning to a new frame beginning.
- It replaces /F/ and /A/ characters according to the JESD204B standard rules. For non-scrambled data, /F/ is replaced with the last character from the last frame whereas /A/ is changed for the last character from the last multiframe. For scrambled data, /F/ is replaced with /D28.3/ and /A/ is replaced with /D28.0/.
- It combines errors of the characters into a frame state. If any of the

characters has an error, that error will be propagated into the frame state.

In case there is an error, it will be reported to the link controller as well, using co_error, that may be observed on Figure 4.14.

Inside, this entity uses another entity, called ring_buffer. There is no separate documentation for this entity, it's quite simple. It receives one character at a time, at a character rate. It outputs frames on frame rate. For aligning to frame correctly, it allows for adjusting position by a minimum of one character. The frame alignment will instruct this entity to adjust the position when needed.

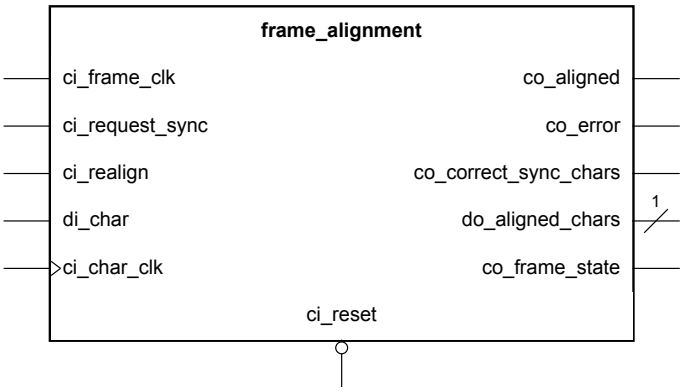


Figure 4.14: Diagram of frame alignment VHDL entity.

Table 4.24 : Input and output ports of frame_alignment VHDL entity.

Name	Description
ci_char_clk	Character clock
ci_frame_clk	Frame clock
ci_reset	Asynchronous reset (active low)
ci_request_sync	External synchronization request
ci_realign	Realign to newly found frame position
di_char	8-bit character data
co_aligned	Tells that the frame position was found and aligned to
co_error	Tells whether there was found an alignment character on wrong position
co_correct_sync_chars	Number of correct alignment characters found on same position
do_aligned_chars	Aligned characters in a frame
co_frame_state	State of the output frame

Table 4.25 : Generic parameters of frame_alignment VHDL entity.

Name	Description
SCRAMBLING	Whether data are scrambled
F	Number of characters in a frame
K	Number of frames in a multiframe
K_CHAR	The 8-bit /K/ character
A_CHAR	The 8-bit /A/ character
F_CHAR	The 8-bit /F/ character
F_REPLACE_CHAR	What to replace /F/ with for scrambled data
A_REPLACE_CHAR	What to replace /A/ with for scrambled data

States of the entity are illustrated on the Figure 4.15. After receiving the first /A/ or /F/ character, the entity will assume the end of a frame on that position. It will then be assumed that every /A/ or /F/ characters should be on the end of the frame and if that is not the case, the state will be set to MISALIGNED and the frame alignment may realign upon an external request. The realignment will set the state back to ALIGNED afterward.

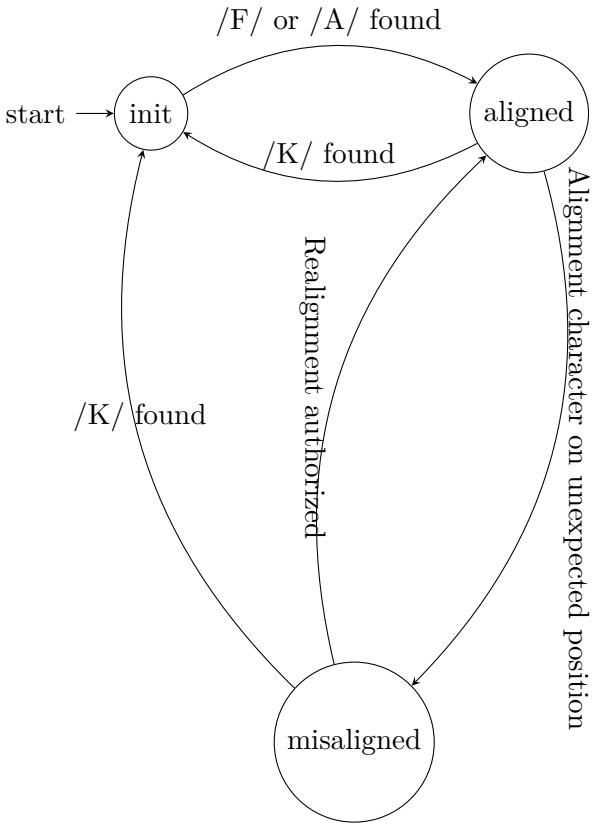


Figure 4.15: Frame alignment states.

Error handler

Error handler receives alignment information given from frame and lane alignment components, as well as errors from the 8b/10b decoder component.

It counts how many errors there are in a multiframe, if it reaches a certain number of errors (configured using generics), it may request realignment from the frame or lane alignment entities. That maximum number of errors has been reached is indicated by setting an output called co_request_sync, the rest of the outputs may be seen Figure 4.16. The other outputs are for requesting lane or frame realignment.

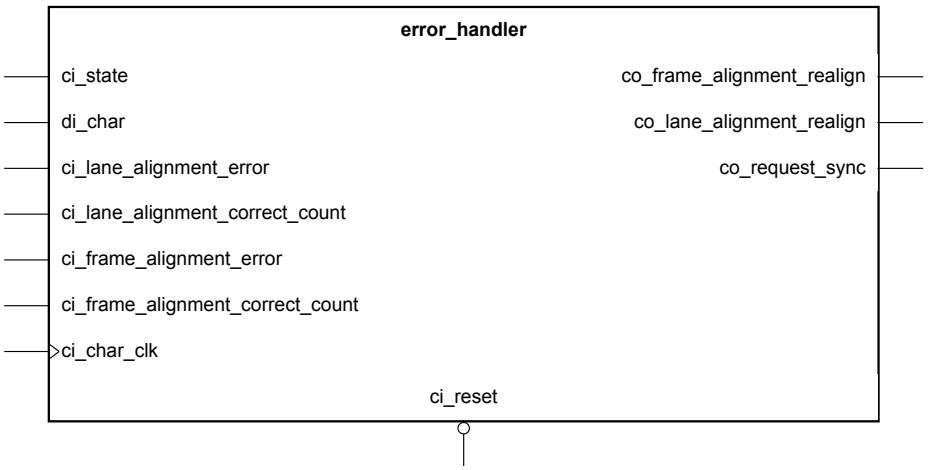


Figure 4.16: Diagram of error handler VHDL entity.

Table 4.26 : Input and output ports of error_handler VHDL entity.

Name	Description
ci_char_clk	Character clock
ci_reset	Asynchronous reset (active low)
ci_state	State of the link
di_char	8-bit character information
ci_error	Whether lane alignment detected /A/ character on wrong position
ci_correct_count	Number of /A/ or /F/ characters located on the same relative position
ci_alignment_error	Whether frame alignment detected /F/ character on wrong position
co_request_sync	Indicates a need to resynchronize the link

4.2.5 Transport layer

The transport layer entity is used for decomposing frames into samples. It receives a whole (aligned) frame of characters and outputs: frame state,

Table 4.27 : Generic parameters of error_handler VHDL entity.

Name	Description
CONFIG	Configuration to specify number of tolerated errors
F	Number of characters in a frame
K	Number of frames in a multiframe

samples, and control bits. As the data link layer outputs aligned frames, it is easy to connect a data link layer to the transport layer directly.

Any possible JESD204B configuration is supported by this entity, by utilizing generics and VHDL generate statements.

This entity combines frame states from each lane. If there is an error in any of these lanes, it will be reported in the frame state. The last frame will be repeated upon an error as well.

The inputs and outputs of this entity may be seen on Figure 4.17.

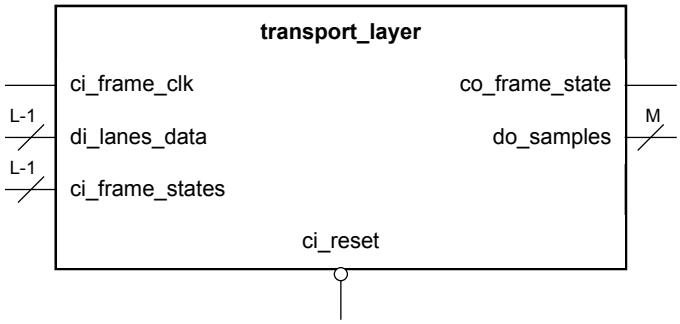


Figure 4.17: Diagram of transport layer VHDL entity.

Table 4.28 : Input and output ports of transport_layer VHDL entity.

Name	Description
ci_frame_clk	Frame clock
ci_reset	Asynchronous reset (active low)
di_lanes_data	(Descrambled) aligned frames from data link
ci_frame_states	States of the aligned frames
co_frame_state	Combined state of the frame
do_samples	Samples in the frame
do_ctrl_bits	Control bits of the samples in the frame

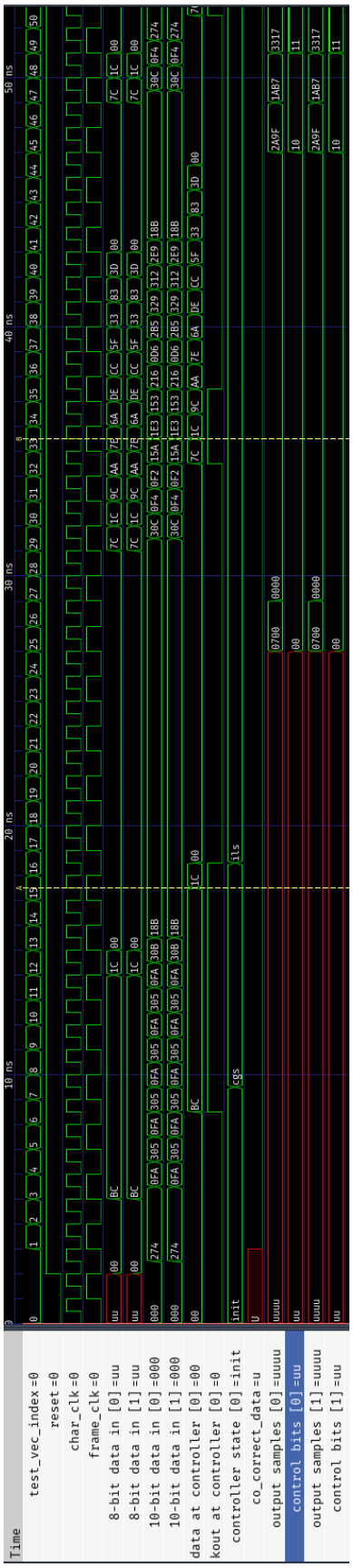
4.3 Testbenches

There is a standalone test for almost all of the entities to make sure each of the entities behaves as expected. There are also testbenches testing entities that connect other entities together.

The most important testbench is probably the one that tests the whole link, simulating a possible synchronization sequence and a short possible data sequence. This test allows us to check for the capability of the design to recognize the synchronization sequence, parse config out of the ILAS, to recognize the start of the frame and parse the samples that were sent as data. The result of the testbench may be seen on Figure 4.18. It is split into two subfigures. It may be observed that the state of the link is progressing from initialization to user data. The data are shown as hexadecimal.

First, /K/ characters are being sent and the link controller is waiting for detecting at least 4 consequent /K/ before deassigning co_nsynced (SYNC∼). After a short while, ILAS is being transmitted and detected by the receiver. The waveform shows a test configured to using two lanes. Data going to the first lane are marked with [0] at the end. The 8-bit data going in are shown in the waveform. These are encoded and sent as 10-bit data to the design. Notice that the 10-bit data are not always the same for the same 8-bit data, that is because running disparity must be kept +1 or -1 and that is done by switching some of the bits. On the figure, markers are present. These markers mark starts of multiframes in ILAS sequence, when at the input of the link controller (data at controller, not 8-bit data in), there is a delay between the data going in and the data at the link controller due to character alignment and 8b/10b decoder entities. A multiframe always starts with /R/ = 0x1C and ends with /A/ = 0x7C. The last marker marks start of the data, indicated by co_correct_data. These are the samples with the control bits that were sent.

Multiple configurations and sequences were tested, mainly configurations that are similar to the board that will be designed for this thesis. Other testbench results may be generated using Modelsim or ghdl with the help of the attached files. Some of the testbenches contain expected sequences that are validated programmatically and any mismatches are reported back.



(a) : First part of the waveform.

(b) : Second part of the waveform.

Figure 4.18: Signal timing diagram of JESD204B link rx test bench.

Chapter 5

Design of the testing (mezzanine) board

Apart from testing the implementation using testbenches, a test on hardware using FPGA and ADCs will be performed as well. An FPGA that is capable of high-speed CML signals will be needed for the test. To test the implementation thoroughly, ADCs should be chosen carefully. The ADC should allow using multiple lanes, it should have multiple analog input channels. Its speed should be as high as possible. But there are some cost constraints as well. ADCs capable of 12.5 Gbps data rate speeds are too expensive for these price constraints. Testing Subclass 1 synchronization between multiple ADCs would be appropriate, too. For that, at least two ADCs are needed, both with the support of subclass 1.

From all of the available FPGA development boards, Intel Cyclone 10 GX development kit was the most suitable carrier board. It has an FPGA mezzanine card (FMC), the PCB board that will be designed, will be compatible with that connector. This connector is highlighted on the development kit on Figure 5.1. The FMC contains 10 rows, 40 pins each. It is connected to a lot of LVDS pairs that may be used as single-ended if needed. There are also 5 transceiver channels available, with a receiver and a transmitter each. That means that the board may have ADCs with a total of 5 JESD204B lanes. These transceivers are capable of up to 12.5 Gbps speeds, the maximum rating of JESD204B. [11] It's not possible to drive 3.3 V logic from Cyclone 10 GX FPGA that is needed for I²C support. However, the development kit contains a buffer converter from 1.8 V logic to 3.3 V logic, connected to the FMC, making support of I²C through the FMC possible without any external components.

Various ADCs were considered, and LTC2123 and AD9683 were chosen in the end. LTC2123 supports 250 MSPS and may transfer data with up to 5 Gbps speeds. The chosen variant of AD9683 supports 170 MSPS. [12][13] Both of these are not so much expensive and offer a lot of features that may be tested. The most important parameters are listed in Table 5.1.

Both of the ADCs have a lot of parameters that may be configured using SPI. CML output current may be adjusted. Control bits may be set to include valid sample indications, over-range or under-range indications. It's possible to enable test patterns on both of the ADCs. AD9683 has a configurable user pattern whereas LTC2123 has pre-configured patterns that are compliant

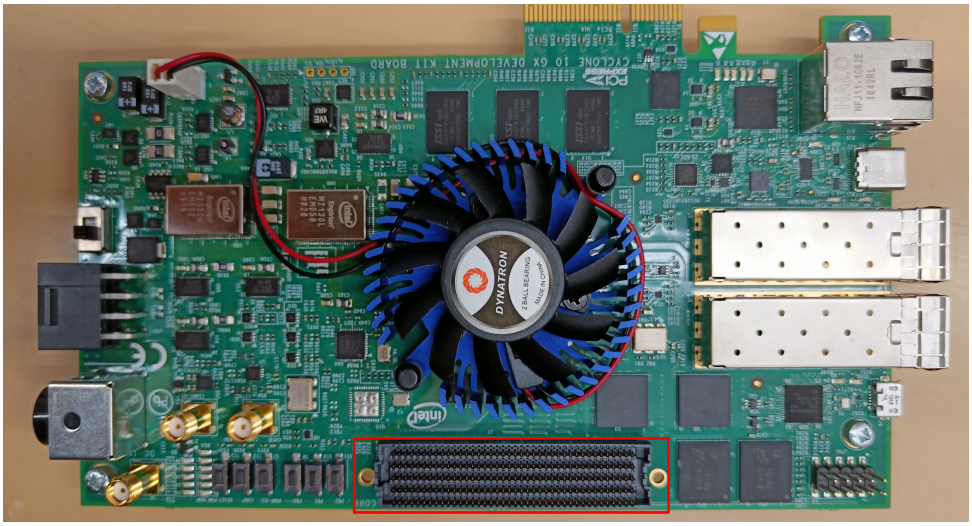


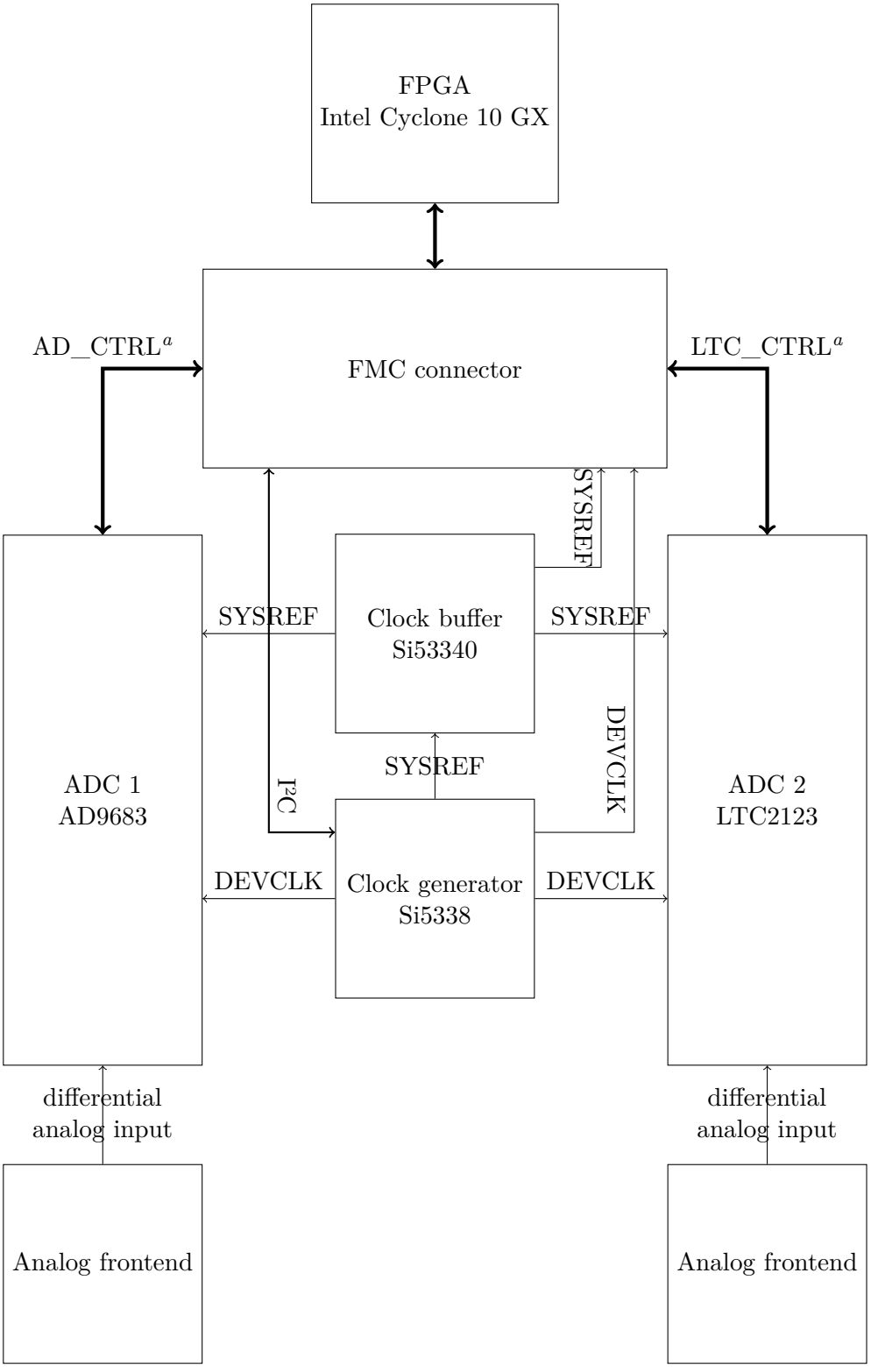
Figure 5.1: Top side of Intel Cyclone 10 GX development kit with highlighted FMC connector.

Table 5.1 : Listing of the most important parameters of LTC2123 and AD9683. [12][13]

Parameter	**LTC2123**	**AD9683**
Maximum sampling frequency	250 MSPS	170 MSPS
Supported JESD subclasses	0, 1, 2	0, 1
Configuration interface	SPI	SPI
No. of analog channels	2	1
SNR	70 dbFS	70.6 dBFS
CML lanes	2 or 4 (configurable)	1
Supply voltages	1.8 V	1.8 V
Power consumption	864 mW	434 mW

with the JESD204B standard. Some of the JESD204B parameters may be changed as well. Whether scrambling is enabled, device ID, number of frames in a multiframe, etc. Initial lane alignment sequence, frame, and lane monitoring may be disabled. [13][12]

It should be possible to test simple as well as more advanced features on the board to be designed. For testing the simple features such as initial synchronization and that correct data have been received, test modes may be used. To test more advanced features such as deterministic latency, at least two ADCs supporting subclass 1 or 2 are needed. The conception of the board to be designed is illustrated on Figure 5.2



[a] Represents a group of signals. See Table 5.2 for individual signals from the group.

Figure 5.2: Testing board high-level conception.

Table 5.2 : Individual signals of AD_CTRL and LTC_CTRL signal groups in Figure 5.2.

Signal / Group	Signal description
CML lanes	Lanes going to the transceivers, using JESD204B protocol
SPI	Interface for configuration (3 signals, clock, data i/o, chip select)
OF∼	Indicates overflow or underflow on input
SYNC∼	JESD204B SYNC∼ signal
PWDN	Powerdown ADC (applies only to AD_CTRL)
RESET∼	Reset of the ADC, active-low (applies only to AD_CTRL)

5.1 Clock generation

It's needed to choose a clock generator that will supply DEVCLK to all of the devices. This clock generator should generate SYSREF as well.

As both ADCs support various clock division factors, a device that allows for generating clock signals of multiple different frequencies would be appreciated. To test multiple frequencies, a configurable device would come in handy.

Si5338 clock generator suits all of these needs. It has 4 outputs, each individually configurable. It may be configured using I²C. Any frequency from 160 kHz to 350 MHz may be chosen and a selected frequencies up to 710 MHz may be used as well. Various output standards such as LVDS, LVPECL, HCSL, CMOS, SSTL, HSTL may be driven. [14]

Si530 has been chosen as an oscillator with differential output that will drive the Si5338 clock generator. Si530 comes in many variants with different frequencies and digital output standards. [15] The generator supports any differential input frequency from 5 MHz to 710 MHz. [14]

As there are only four outputs on the clock generator, three are needed for device clocks, only one may be used for SYSREF. Because SYSREF has to go to all three devices (FPGA, LTC2123, AD9683), a clock buffer must be used to distribute one SYSREF to three devices. Si53340 has been chosen. It has ultra-low additive jitter of 50 fs RMS and supports a wide range of frequencies from dc to 1250 MHz. [16]

Si5338 supports two clock signals and it's possible to configure what input clock signal to use. To make it possible to test with better or worse accuracy of the clock, an external clock signal will be added as well. This signal has to be differential on the input of the Si5338, but it's more convenient to use a single-ended external clock for the needs of this thesis as only single-ended clock generators are on hand. High-speed differential LVDS drivers may drive LVDS signals from TTL logic. One of those is DS90LV011A, it has a maximum of 700 ps skew and 1.5 ns maximum propagation delay. [17]

5.2 Analog front-end

Both of the ADCs have differential analog inputs. The differential input should get the common mode voltage from V_{CM} pin that is present on both ADCs. There are several ways to drive differential analog inputs from single-ended signals. The circuit driving the ADC may be called an analog front-end.

One way is using a differential transformer-coupled configuration, illustrated on Figure 5.3. This configuration may come in handy in applications that need to optimize for SNR. [12] From the nature of transformers, only baseband applications may be realized.

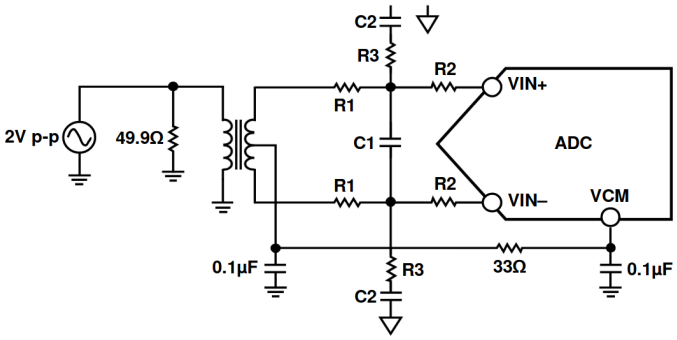


Figure 5.3: Differential transformer-coupled configuration. [12]

Another possibility is to use a differential amplifier, such as ADA4930 recommended by Analog Devices for AD9683. [12] Using an amplifier gives the benefit of allowing a dc signal to the ADC as well. To make the design easier, ADA4930 has been chosen to drive LTC2123, too. ADA4930 comes in two packages, one for a single channel and another for a dual channel. As LTC2123 has two analog inputs, it's convenient to use the variant for dual channels for the LTC2123.

One of the possible wirings is illustrated on Figure 5.4. The driver is an operational amplifier with differential output. The output has a common mode voltage that is specified using the V_{CM} input. This wiring is directly from the datasheet of AD9683 and has a gain of 1. Very similar wiring was used for LTC2123 driving as well.

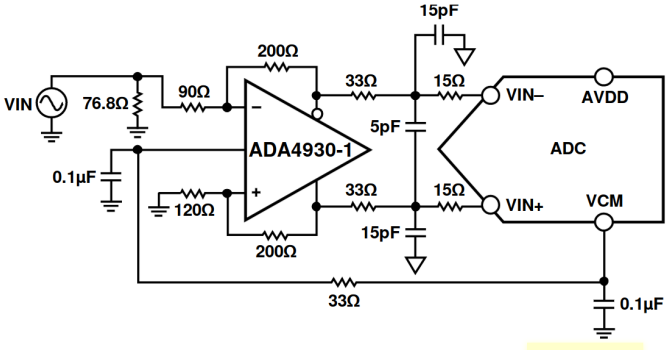


Figure 5.4: Differential input configuration using the ADA4930. [12]

5.3 Supply, voltage levels

The development kit outputs three distinct voltages through the FMC. These are: 1.8 V (adjustable, default is 1.8 V), 3.3 V and 12 V. 1.8 V may power up to 4 A, 3.3 V may power up to 12 A.

Both of the ADCs require 1.8 V voltage only, but it's recommended to isolate some of the voltages from each other. AD9683 datasheet recommends isolating digital and analog voltages. It also recommends adding an inductor between digital and CML drivers voltages. The supported SPI voltage level is 1.8 V as well. LTC2123 does not have separate voltages for digital and analog parts, only for digital and CML output drivers. These should have a ferrite bead between them.

As the main objective of this thesis is to receive data from the ADCs using JESD204B and not to achieve optimal values of some of the parameters (such as SNR or THD), isolating voltages is unnecessary. Only ferrite beads with a combination of capacitors will be used to get rid of some of the higher-frequency noise.

The ADC drivers need 3.3 V or 5 V supply voltage. As the required common mode voltage of analog input for LTC2123 is roughly 783 mV and the drivers may output minimum of 800 mV common mode for 3.3 V supply voltage, it will be necessary to provide 5 V to the drivers. The development kit does not provide 5 V directly, a step-down and a voltage regulator will be needed. A step-down from 12 V to 6.5 V, specifically TSR 1-2465[18], has been chosen. LD29150PT50R will be used as a voltage regulator, it outputs 5 V. [19]

Clock generation-related circuits all accept a 3.3 V voltage level that is provided directly from the development kit, making it the option that is easiest to provide.

Every component requires some decoupling capacitors that reduce the effect of noise caused by other circuit elements. Datasheets of the components provide information about the position of these capacitors as well as the recommended values.

It has to be made sure that the maximum current of the power supplies won't be reached. Information about maximum current ratings for each of the components is provided in Table 5.3, all of these were obtained from relevant datasheets. As can be observed, all of the values are well below the maximum possible current supplied, there should thus be no issues with power supply.

5.4 High-speed CML lanes

CML is used for transmitting samples from the ADCs to the FPGA. As the lanes may work on up to 5 Gbps speeds, that makes these signals high-speed. It's important to take some precautions when working with high-speed signals.

High-speed differential pairs should have the same distance between the traces. The area where the traces do not have the same distance should be minimal. There shouldn't be any components in between the traces. Coupling

Table 5.3 : Maximum supply currents needed from datasheets of the components. The total row contains the sum for the given voltage level and the row with the maximum shows the allowed limit for the given voltage level, according to the development kit user guide. For 5 V, specifications of the chosen step-down and LDO are shown.

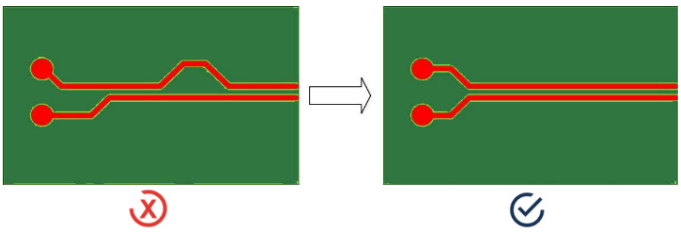
-	**Component**	**Max supply current (mA)**
5 V voltage level		
	ADA4930-2	76.8[a]
	ADA4930-1	38.4[a]
Total		**115.2**
Maximum		**1500**
3.3 V voltage level		
	Si5338	92
	Si53340	140[b]
	Si530	98
	DS90LV011A	10
Total		**340**
Maximum		**12000**
1.8 V voltage level		
	AD9683	260
	LTC2123	533.8
Total		**793.8**
Maximum		**4000**

[a] Quiescent current specified, current under operation not specified.
[b] Typical value, maximum is not specified.

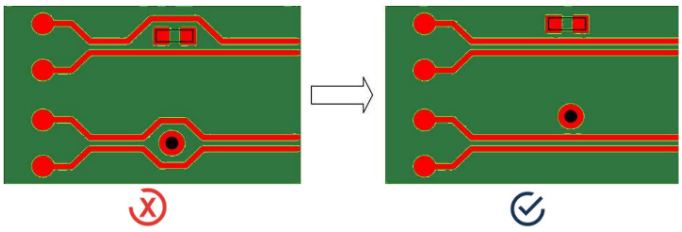
capacitors or vias should be placed symmetrically. The number of vias should be as low as possible as they introduce a discontinuity in impedance. All of these are demonstrated on Figure 5.5. [20]

In this design, it's possible to route the high-speed CML lanes on the same, top, layer. It was made sure that the ADCs are on the same side of the board as the FMC connector. That way, it's not necessary to route the CML signals into other layers of the board. The signals are routed to other layers on the development kit side, though, but that does not make any difference in designing the mezzanine board.

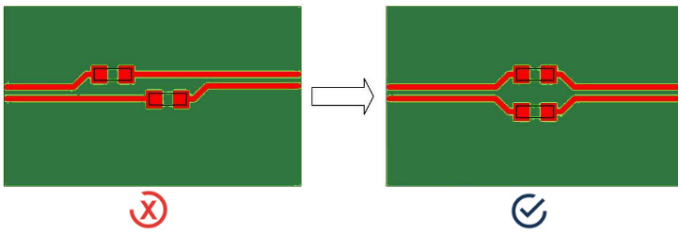
High-speed traces should not be routed over a split plane. It may lead to the trace acting as a loop antenna. That is because, for high-speed signals, the return path tends to follow the trace and by routing over a split plane, the return path won't be able to follow the signal trace. When it's needed to route the trace over a split plane, there should be a stitching capacitor placed over the split plane, as illustrated on Figure 5.5d. [20]



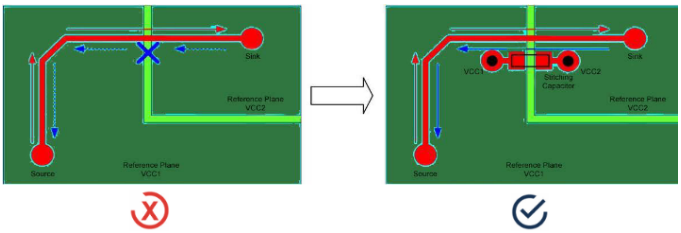
(a) : Constant distance between traces.



(b) : No components in between.



(c) : Components placed symmetrically.



(d) : Routing over a split plane.

Figure 5.5: Some of the guidelines for high-speed signal routing. [20]

5.5 Length matching

As pairs are bent one way more than the other, one of the traces becomes shorter than the other. It's important to make sure that both of the traces in one pair have the same length to minimize the skew. It's also important to match these lengths close to the bends. [20]

There may be additional requirements for skews between different signals, ie. to make sure that multiple signals arrive within one clock cycle and meet the setup and hold time requirements. [20]

JESD204B standard specifies maximum skews between some of the signals. SYSREF should arrive at each device within the same device clock cycle. SYNC~ signals should arrive at each device within the same device clock cycle (for subclass 2). CML lanes or device clocks have a maximum defined

skew between them. It was important to match all of these signals on the PCB. A maximum of 2 mils of difference between relevant trace lengths was achieved among all of the signals.

5.6 Controlled impedance

All of the differential pairs have to have a controlled impedance. Both LVDS and CML should have a differential impedance of 100 Ω. When designing the PCB, that must be taken into account. To make sure the impedance matches, the distance between the differential pairs and the width of one trace has to be carefully chosen.

One of the ways to compute the necessary widths and distances is to use Saturn PCB Design Toolkit, a screenshot from the PCB design toolkit may be seen on Figure 5.6. On this screenshot, the final spacing and width chosen, may be seen. From the screenshot, this leads to roughly 100 Ω differential impedance.

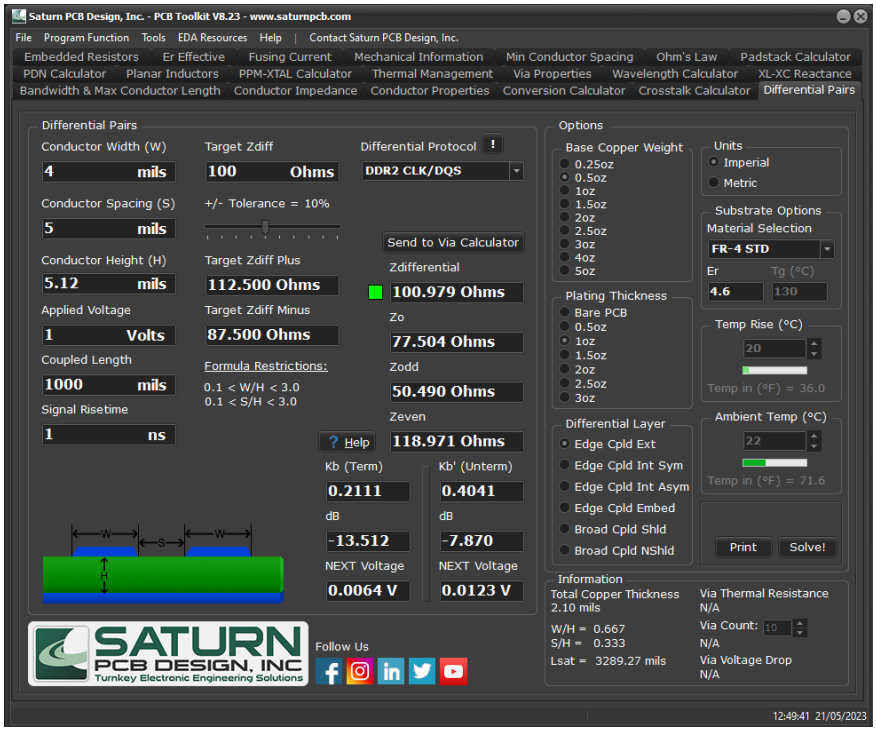


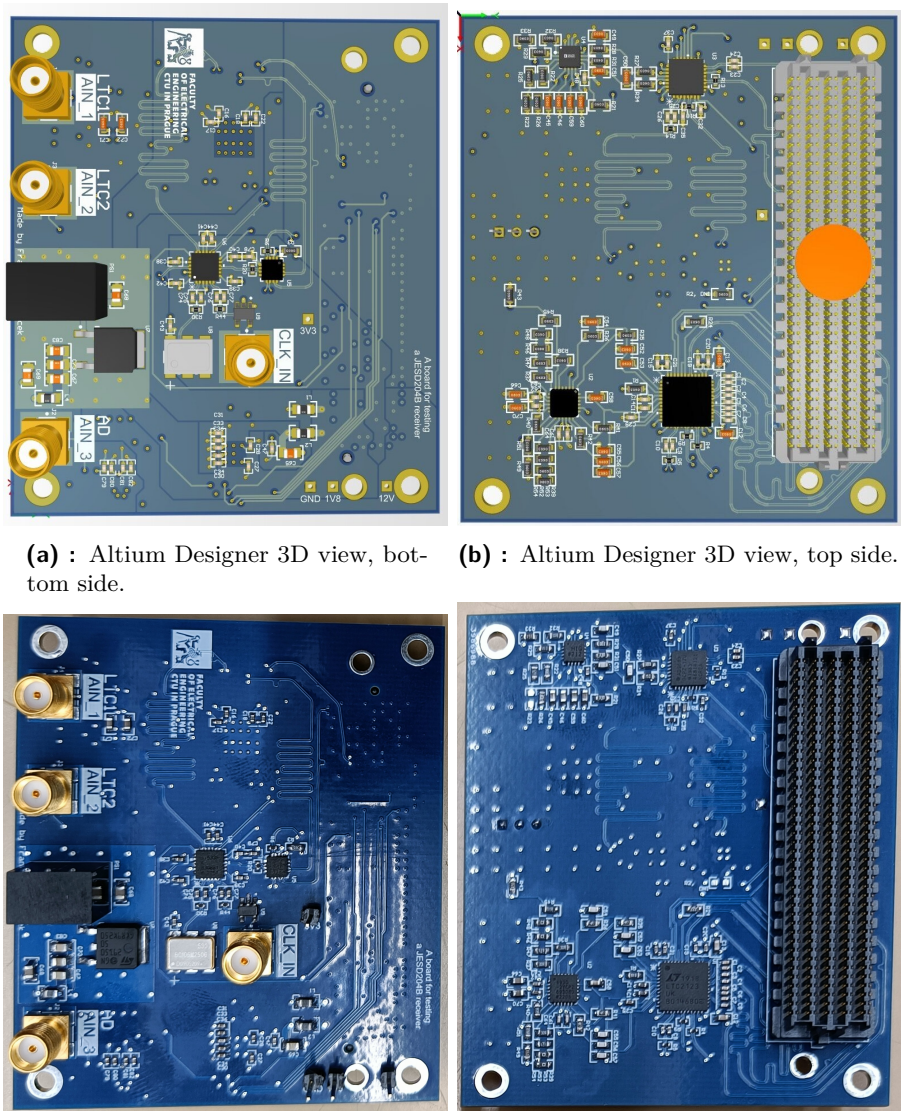
Figure 5.6: Configuration of differential pair controlled impedance from Saturn PCB Toolkit.

5.7 Final board

To make sure all requirements of differential pairs are met, at least 4 layer PCB is required.

The board has been designed in Altium Designer. Altium Designer is a software package for designing PCBs. It was developed by Altium Ltd. Gerber files of the board are attached to the thesis in Appendix B. The final board, compared with Altium Designer 3D view, is shown on Figure 5.7.

Unfortunately, not all of the needs for high-speed signals were met when designing the PCB. As the board was designed with only 4 layers and it was not discovered until too late that there should not be any split planes when routing high-speed signals. Some of the high-speed CML signals go through a split plane.



(a) : Altium Designer 3D view, bottom side.

(b) : Altium Designer 3D view, top side.

(c) : Photo of the board, bottom side.

(d) : Photo of the board, top side.

Figure 5.7: Comparison of the final printed board and Altium Designer 3D view.

Chapter 6

Setup and testing on FPGA development kit

Intel FPGAs may be accessed using Quartus Prime software. To work with Intel Cyclone 10 GX, the Quartus Prime Pro version is required. Although the Pro version normally requires a paid license, work with Intel Cyclone 10 GX is possible without any license, for free.

6.1 FPGA setup using Quartus

To make the test possible, a transceiver (SERDES) must be utilized to receive the data. Intel provides an IP core for accessing the transceivers directly. It's called the Transceiver PHY IP core. To use this IP core as a receiver, it's necessary to create and configure at least two blocks, one for the transceiver itself and the other as a reset controller. A custom reset controller may be designed with custom functionality, but a default reset controller meeting requirements to correctly reset the transceiver is provided in Quartus Prime. [21]

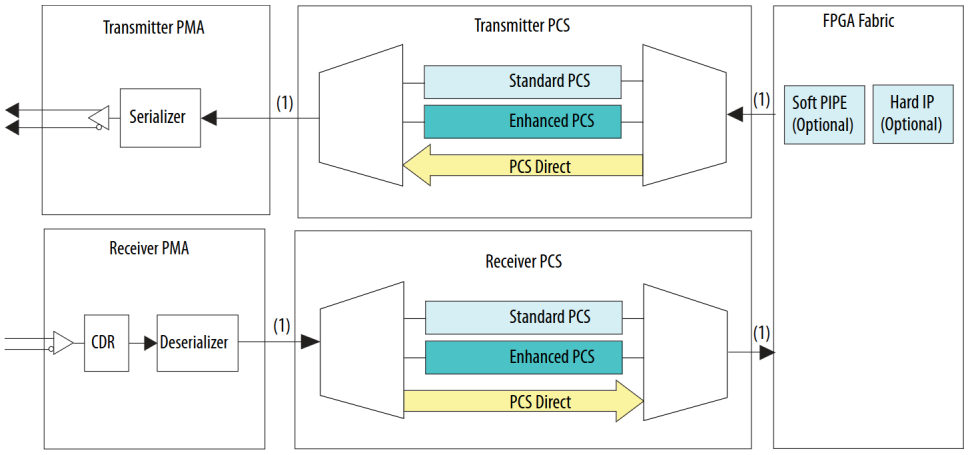


Figure 6.1: Transceiver channel in full duplex mode. [21]

It may be observed from Figure 6.1 that the transceiver consists of Physical media attachment (PMA) and Physical coding sublayer (PCS) parts. The

PMA is the interface to the physical medium. The interface width between PMA and PCS is configurable. It's possible to bypass the function of PCS by utilizing PCS Direct mode. By using the Standard or Extended PCS, it's possible to configure the transceiver to use features of a specific protocol. The Standard PCS supports features such as aligning to a word (denoted by a pattern) or decoding 8b/10b words. The IP core supports 1.0 Gbps to 12.5 Gbps speeds. For lower frequencies, oversampling is required. [21]

For a transmitter, an external PLL must be provided as well. Intel provides a block for creating a PLL as well. The transmitter supports various clock frequencies so PLL may not be needed. The receiver has a channel PLL directly inside of it. This PLL is responsible for clock and data recovery. [21]

For purposes of this thesis, it should be fine to use the PCS Direct mode. Although the character alignment of Standard PCS could be utilized, there is a character alignment already implemented in the VHDL design as well. Thus it's not a requirement to align on the physical layer. Decoding is done in the VHDL as well, so there is no need in performing it in the transceiver. [21]

As the design requires both a character clock and a frame clock, a PLL may be employed to obtain these from the device clock going in from the clock generator. Intel provides IOPLL IP core for implementing a PLL on the FPGA. It may be shown on the typical architecture, present on, Figure 6.2 that the PLL may produce multiple outputs by dividing the VCO output.

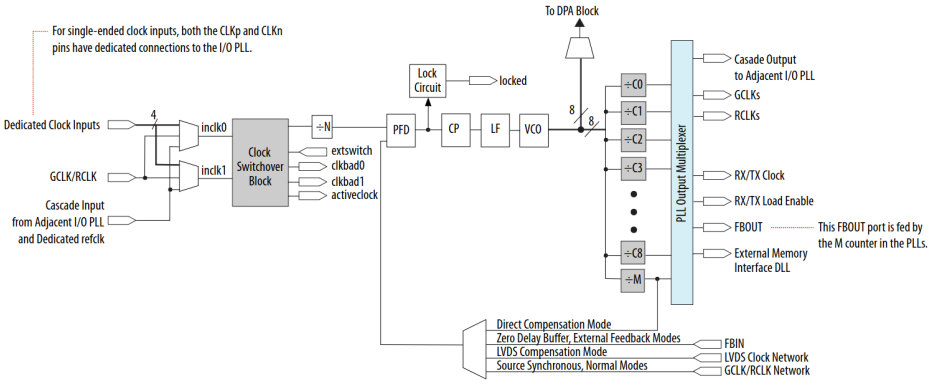


Figure 6.2: Typical I/O PLL architecture. [22]

If the tests of the implementation were done using the custom-designed board right away and there were problems, it would be hard to track down where the problem is exactly. It could be better to test in such a way as to rule out as many problems as possible. The development kit comes with an FMC loopback card that connects transceiver channel TX to RX. Intel Quartus includes JESD204B Intellectual property core (IP core) that includes physical and data link layers. One of the possible ways to test, would be to use this JESD204B IP core as a transmitter and connect it to the implemented receiver. After this works, it should be possible to rule out many of the problems coming from the FPGA design itself.

But there is a problem with this approach as well. The Intel implementation

of JESD204B is complex. The block does not contain the JESD204B interface only. It's also possible to access the JESD204B in Nios II using an Avalon memory-mapped interface. The user guide for JESD204B doesn't seem to contain all of the information needed for the correct operation, it has to be combined with other user guides such as the user guide for Transceiver PHY. It is thus possible that I have missed something in the user guides and that would mean the test would not work as expected. This still leaves a possibility for an error outside of the custom VHDL design.

6.2 Board configuration

Before the board may be tested, the components on it have to be properly configured. The configuration may be done from the FPGA directly. A convenient way to use the FPGA as a master of I²C or SPI bus is to utilize the Nios II along with peripherals.

Nios II is a soft processor designed specifically for FPGAs of the Altera family. Using a soft processor such as Nios provides cost flexibility as no external processor is needed and peripherals may be reconfigured at any time. It's possible to meet many different demands with soft processors.

For purposes of this thesis, a Nios processor with I²C and two SPI interfaces will be used.

The clock generator is configured using I²C protocol. To figure out the values of registers of the clock generator for specific input and output clock frequencies, Skyworks provides a ClockBuilder Pro software. This software may export the registers to be programmed to a CSV or a C header as an array. The datasheet also specifies the order in which to program the generator. A simplified version of the configuration is illustrated on Figure 6.3. All of the configurations may be uploaded through the use of I²C. The relevant alarms to check input clock validity and PLL locking, are stored inside some of the registers of the device. It's not necessary to use any other pins of the Si5338. [14]

AD9683 uses SPI for configuration. It has just one SDIO pin for both data in and data out. There is a pull-down on that signal inside of the AD9683. AD9683 uses 16-bit instructions. This instruction contains a read indication, the number of bytes to read or write and the address to write to. The bits labeled W1 and W0 represent the number of registers, to be read or written to, minus one. If W1W0 = 0, one register will be accessed. Instruction and data read or write is illustrated on Figure 6.4. AD9683 uses a flag in a register for committing changes. The changes to some of the registers are not written immediately but wait for a flag in a register to be set to one. This allows for multiple configuration fields to be reprogrammed at once.

Same as AD9683, LTC2123 uses SPI for configuration. LTC2123 requires an external 2 kΩ pull-up on SDO so it is able to send data back. Contrary to AD9683, LTC2123 exposes two wires for data, SDI (input), and SDO (output). However, it does not allow for a full-duplex operation. Only one direction at a time is supported. During the design of the board, it was

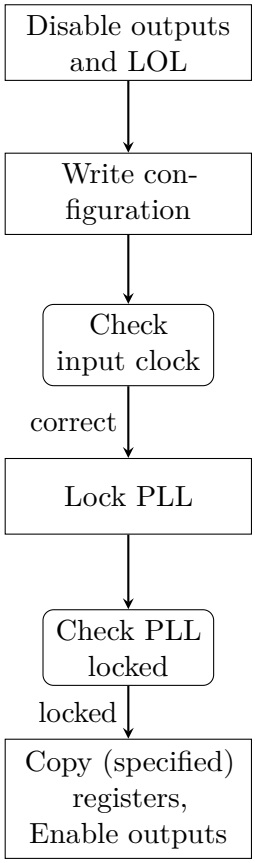


Figure 6.3: Simplified flowchart of Si5338 configuration. Full flowchart available in [14] on page 23.

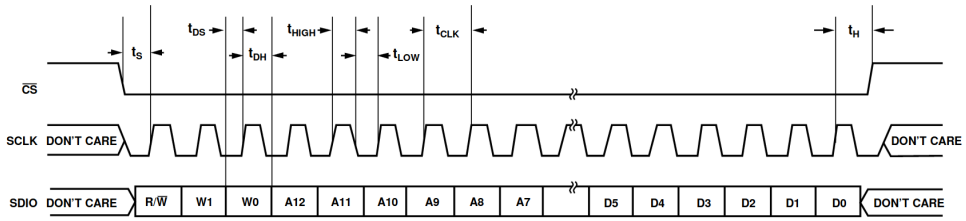


Figure 6.4: AD9683, SPI Timing, writing data to a register. [12]

decided to make a connection between SDI and SDO, making one signal, SDIO. This makes the interface of AD9683 and LTC2123 the same, except for one detail. LTC2123 has a pull-up whereas AD9683 has a pull-down. As there is enough signals on the FPGA available, the SPI channels of both ADCs were separated. The instruction for LTC2123 has 8 bits. It contains read flag and an address. Then the data follows. Reading or writing only one register at a time is supported. Writing data to a register is illustrated on Figure 6.5. [13]

For testing purposes, it may be useful to set up a test mode on both of the ADCs. Thanks to a test mode, it may be observed whether the high-speed

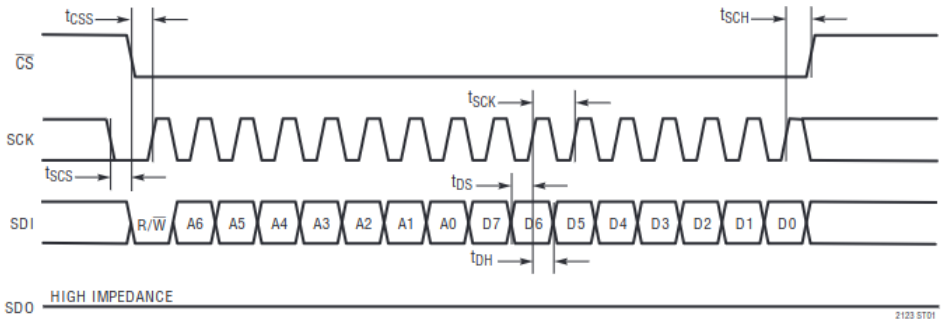


Figure 6.5: LTC2123, SPI Timing, writing a byte to a register. [13]

signal is received correctly. Both of the ADCs support test modes as specified by the JESD204B standard. Moreover, AD9683 supports specifying a custom user pattern. This user pattern may be used as either: input to scrambler, output of 8b/10b encoder or even replace the samples directly, while still keeping the JESD204B interface working regularly.

6.3 Results

A Quartus project with the Nios II platform was created. A C code for configuring the clock generator and the ADCs was written. Both of the ADCs as well as the clock generator were successfully configured using these programs.

Then, transceivers IP cores were added to the project as well and the output was observed directly using the Signal Tap logic analyzer. When configuring the ADCs to use test modes, to send specific characters, the received sequence was the expected one. For example, characters consisting of '1010101010' (D21.5) or '1111100000' (K28.7) bits, were received correctly.

It was attempted to connect the outputs of the transceivers to the custom implementation of JESD204B receiver. The first important thing is to check whether /K/ character is observed when not synchronized and whether the character boundary is found and correctly aligned to. This was exactly the behavior of the design.

However, after notifying the transmitter of the correct alignment, effectively requesting ILAS, problems arose. The receiver started reporting errors in character decoding, both disparity and not in table errors. The ILAS was thus not detected at all, not even the first character was correct.

This behavior was observed multiple times and on both of the ADCs. Unfortunately, the origin of the error, was not found exactly, in time. The possible causes of the problem may origin somewhere from all of there parts:

- ADCs are sending wrong data. (improbable)
- PCB routing does not allow for high-speed signals to propagate without errors.
- Development kit PCB or FPGA transceivers.
- Transceivers configuration.
- Custom VHDL JESD204B receiver design.

One part that is possible to be ruled out, is the transceivers of the FPGA. The development kit comes with a loopback board connecting transceiver channel transmitters to receivers. There is also a test program that tests whether a sequence that is being sent is also correctly received back. This program was not reporting any errors, thus effectively ruling out errors directly on the transceivers. But other than that, it's not possible to tell. There were not enough tests done due to time constraints. (more on this in chapter 7)

The tests were done using a data rate of 2 Gbps (100 MHz sampling frequency). It's possible that some of the parts did not work well with 200 MHz speeds. Test for lower frequencies could be done to check for that. LTC2123 supports a minimum of 50 MSPS and AD9683 supports a minimum of 40 MSPS.

The resulting testing Quartus project described here is attached to this thesis. For the list of some of the most important contents of this attachment, see Appendix A.

Chapter 7

Conclusion

The aim of this thesis was to first study the principles of the JESD204B receiver. As a result of this first point, a short description of the protocol has been made. This description aims to show most of the aspects of the protocol in a simplified way. The description of the JESD204B protocol thus does not contain everything there is to know about the protocol. It misses mainly the physical properties that are recommended or must be met, such as the maximum skews or interface standards in use.

Next, an implementation of the JESD204B receiver in VHDL should have been made. This implementation should have been verified in a simulation environment. The protocol's subclass 0 and subclass 1 were implemented except for some advanced features such as the detection of test modes or error reporting through SYNC~ signal. The implementation also misses the detection of test modes or a correct descrambler. These features are not critical for the design, they are needed only for some configurations and the design may correctly run without these. Testbenches were written for most of the entities and verified the expected behavior.

At last, a testing board should have been designed and a test of the implementation should have been performed using this board. The board has been designed using Altium Designer. It contains two ADCs that are highly configurable. Making it convenient to test as many aspects of the implementation including all of the subclasses and deterministic latency. Many different sampling frequencies could be tested as well thanks to a clock generator that is configurable through I²C. One of the ADCs may go up to 250 MSPS, leading up to 5 Gbps bit rates between the ADC and FPGA.

There were some time delays before the custom board was ready, leaving little time for testing. The board had to be made abroad and that meant more time for shipping and it got stuck at customs for quite some time as well. After the board got on our hands, it had been discovered that a stencil mask had to be made to mount the FMC, leading to another delay. These problems were not foreseen and led to a few weeks of delay, and because of that, the last point to test the implementation using the board was not fulfilled completely. A simple program for configuring the clock generator (using I²C), as well as both of the ADCs (using SPI), has been made. It was attempted to look at the data coming out of the ADCs and connect this

output to the custom implementation, but the output was not correct, the first synchronization character was found, but the rest of the sequence was not processed correctly. The possible causes of the problem were discussed in section 6.3. It should also be noted that it was one of the first times the author designed a board and the first time designing a board with high-speed signals. That also led to the board being designed for a longer period than it could have been, had the author had the necessary requirements.

7.1 What's next

It was noticed that Intel JESD204B IP Core has wider channel width going to the data link layer. It receives four characters at a time instead of just one. A similar feature could be implemented in the custom design in the future. The data link layer could receive an arbitrary number of symbols configured through generics. That would mean that the logic would have to be changed to ensure the correct processing of multiple characters at a time. Thanks to these changes, the clock going to the data link layer could be slowed down. Implementing the advanced features and support for subclass 2 could be the next step, as well.

More thorough testing using the board could be done to discover where an error is and how it could be fixed. Currently, there were not enough tests done to pinpoint where the problem is exactly.

Bibliography

[1] Analog Devices. *JESD204B Survival Guide*. [online]. 2014. Available at: https://www.analog.com/media/en/technical-documentation/technical-articles/JESD204B-Survival-Guide.pdf (visited on 21/05/2023).

[2] Del Jones. *JESD204C Primer: What's New and in It for You—Part 1*. [online]. Available at: https://www.analog.com/en/analog-dialogue/articles/jesd204c-primer-part1.html (visited on 22/05/2023).

[3] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. *JESD204B - Serial Interface for Data Converters*. [online]. 2011. Available at: https://www.jedec.org/sites/default/files/docs/JESD204B.pdf (visited on 21/05/2023).

[4] Jonathan Harris. *Understanding Layers in the JESD204B Specification - A High Speed ADC Perspective*. [online]. 2017. Available at: https://www.analog.com/en/technical-articles/understanding-layers-in-jesd204b-specification.html (visited on 21/05/2023).

[5] Del Jones. *JESD204B Subclasses—Part 1: An Introduction to JESD204B Subclasses and Deterministic Latency*. [online]. 2019. Available at: https://www.analog.com/en/technical-articles/jesd204b-subclasses-part1-an-introduction-to-jesd204b-subclasses-and-deterministic-latency.html (visited on 21/05/2023).

[6] Lattice Semiconductor. *8b/10b Encoder/Decoder*. [online]. 2015. Available at: https://www.latticesemi.com/-/media/LatticeSemi/Documents/ReferenceDesigns/1D/8b10bEncoderDecoder-Documentation.ashx?la=en (visited on 21/05/2023).

[7] Tristan Gingold. *GHDL guide*. [online]. Available at: http://ghdl.free.fr/ghdl/index.html (visited on 22/05/2023).

[8] *Gtkwave*. [online]. Available at: https://gtkwave.sourceforge.net/ (visited on 22/05/2023).

[9] Intel. *Intel® Quartus® Prime Software*. Available at: https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime/article.html (visited on 22/05/2023).

[10] Volnei A Pedroni. *Digital Electronics and Design with VHDL*. Oxford, England: Morgan Kaufmann, Jan. 2008. ISBN: 978-0123742704.

[11] Intel. *Intel® Cyclone® 10 GX Device Overview*. [online]. Available at: https://www.intel.com/content/www/us/en/docs/programmable/683485/current/device-overview.html (visited on 21/05/2023).

[12] Analog Devices. *AD9683 - 14-Bit, 170 MSPS/250 MSPS, JESD204B, Analog-to-Digital Converter*. [online]. Available at: https://www.analog.com/media/en/technical-documentation/data-sheets/AD9683.pdf (visited on 21/05/2023).

[13] Linear Technology. *LTC2123 - Dual 14-Bit 250 MSPS ADC with JESD204B Serial Outputs*. [online]. Available at: https://www.analog.com/media/en/technical-documentation/data-sheets/2123fc.pdf (visited on 21/05/2023).

[14] Skyworks. *Si5338 - I2C-Programmable Any-Frequency, Any-Output Quad Clock Generator*. [online]. Available at: https://www.skyworksinc.com/-/media/Skyworks/SL/documents/public/data-sheets/Si5338.pdf (visited on 21/05/2023).

[15] Skyworks. *Si530/531 - Crystal Oscillator (XO) (10 MHz to 1.4 GHz)*. [online]. Available at: https://www.skyworksinc.com/-/media/skyworks/sl/documents/public/data-sheets/si530-31.pdf (visited on 21/05/2023).

[16] Skyworks. *Si53340-45 - Data Sheet*. [online]. Available at: https://www.skyworksinc.com/-/media/SkyWorks/SL/documents/public/data-sheets/Si5334x-datasheet.pdf (visited on 21/05/2023).

[17] Texas Instruments. *DS90LV011A 3V LVDS Single High Speed Differential Driver*. [online]. Available at: https://www.ti.com/lit/ds/snls140c/snls140c.pdf?ts=1684658522480 (visited on 21/05/2023).

[18] Traco Power. *TSR 1 Series, Non-Isolated DC/DC Converter (POL) - Datasheet*. [online]. Available at: https://www.tracopower.com/sites/default/files/products/datasheets/tsr1_datasheet.pdf (visited on 21/05/2023).

[19] STMicroelectronics. *LD29150 - 1.5 A, very low drop voltage regulators - Datasheet*. [online]. Available at: https://www.st.com/resource/en/datasheet/ld29150.pdf (visited on 21/05/2023).

[20] Sierra Circuits. *High-Speed PCB Design Guide*.

[21] Intel. *Intel® Cyclone® 10 GX Transceiver PHY User Guide*. [online]. Available at: https://www.intel.com/content/www/us/en/docs/programmable/683054/20-1/transceiver-phy-overview.html.

[22] Intel. *IOPLL Intel® FPGA IP Core User Guide*. [online]. Available at: https://www.intel.com/content/www/us/en/docs/programmable/683285/18-1/core-user-guide.html (visited on 21/05/2023).

Appendix A

Contents of the attachment

The attachment contains:

1. VHDL code of the JESD204B receiver in VHDL.
2. Altium project containing the schematics and PCB design of the custom board.
3. Quartus project for testing the design using the board along with Nios software for configuring the clock generator and ADCs.

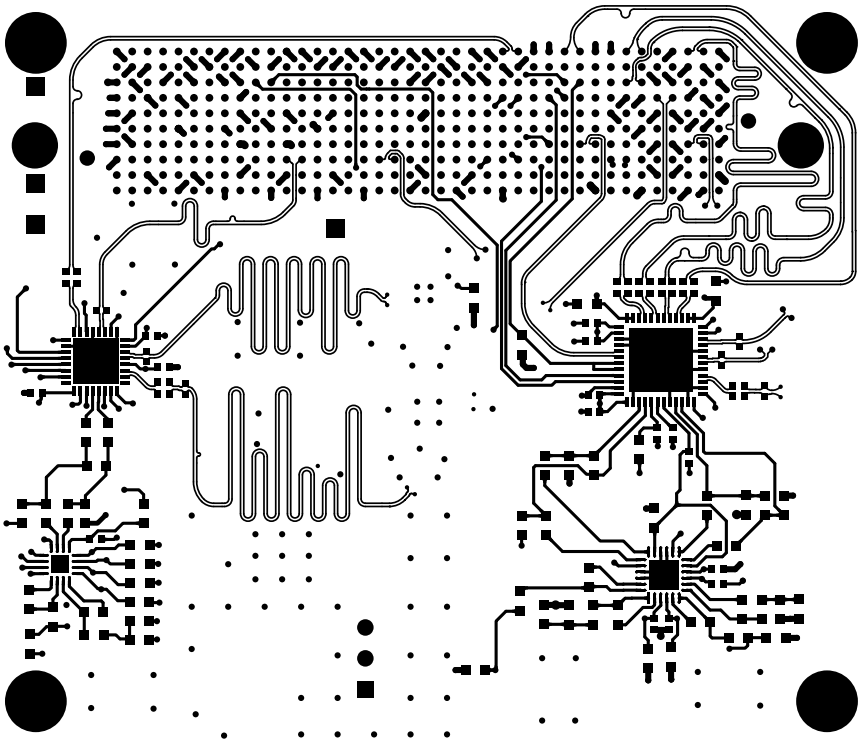
Some of the most important folders and files from the attachment (not everything is listed here):

```
bohacek_bp_sources
├── fpga_testing_quartus
│   ├── jesd_rx
│   ├── nios_platform
│   │   └── software
│   │       ├── ad9683_configuration
│   │       ├── ad9683_configuration_bsp
│   │       ├── clock_gen_configuration
│   │       ├── clock_gen_configuration_bsp
│   │       ├── ltc2123_configuration
│   │       └── ltc2123_configuration_bsp
│   └── bohacek_jesd_board.qpf
├── pcb_altium_project
│   ├── Project Outputs for BOHACEK_BP_ADC_board
│   └── BOHACEK_BP_ADC_board.PrjPcb
└── vhdl_jesd204b_rx
    ├── src
    │   ├── data_link
    │   ├── transport
    │   ├── jesd204b_link_rx.vhd
    │   └── jesd204b_multipoint_link_rx.vhd
    └── testbench
        ├── data_link
        └── transport
```

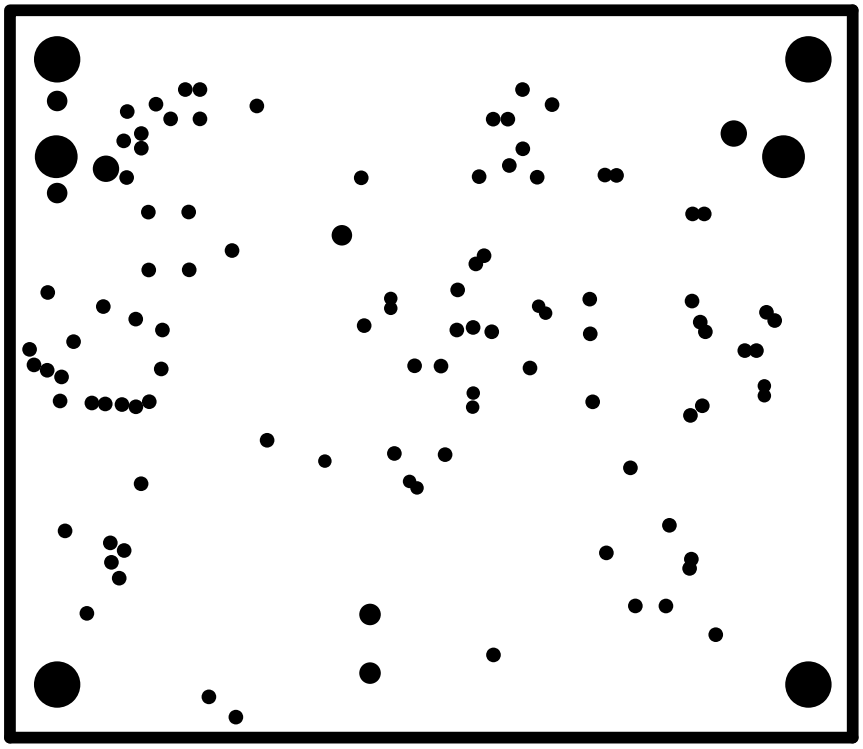
Appendix B

Exported Gerber files of the custom printed circuit board

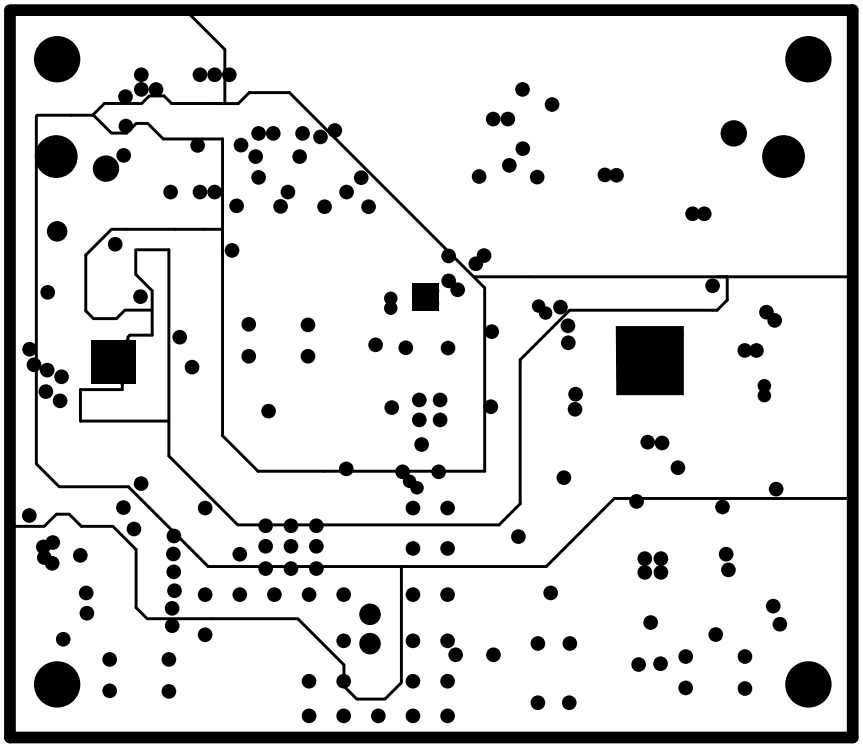
Exported gerber files of the custom 4-layer PCB follow.



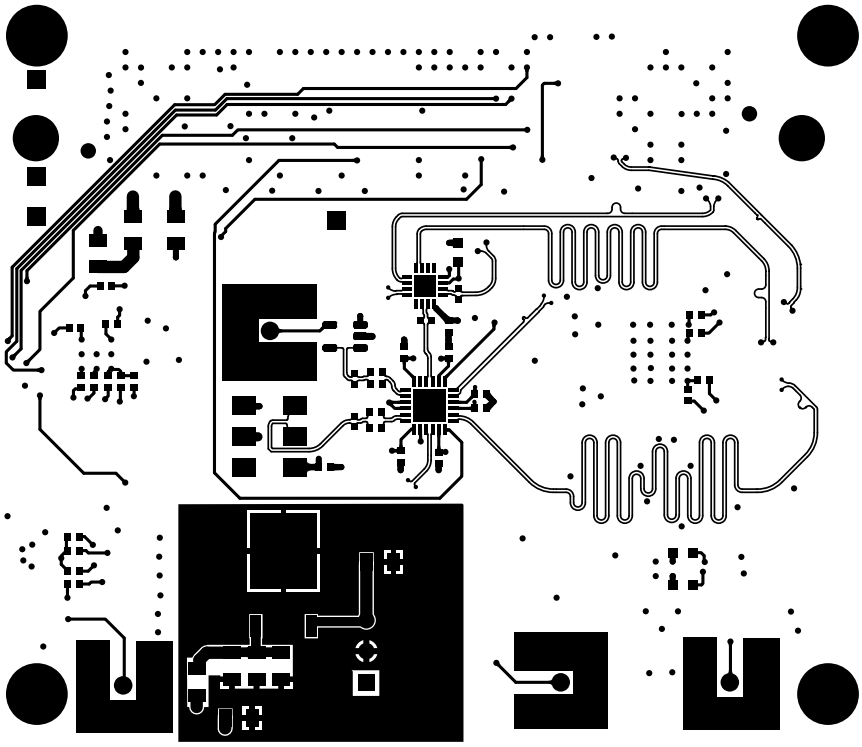
(a) : Top side.



(b) : Inner layer 1, plane, GND.



(c) : Inner layer 2, plane, power (12 V, 5 V, 3.3 V, 1.8 V voltage levels).



(d) : Bottom side.

Figure B.1: Exported Gerber files of each of the 4 layers of the custom PCB.