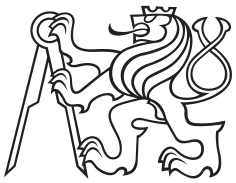
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Dapartment of Microelectronics

JESD 204B Rx Controller Design

Bohdan Jůza

Supervisor: prof. Ing. Pavel Hazdra, CSc.
Supervisor–specialist: Ing. Martin Hujer, Ph.D.
Field of study: Electronics and Communications
Subfield: Electronics
May 2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jůza** Jméno: **Bohdan** Osobní číslo: **466308**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra mikroelektroniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Elektronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Návrh Rx řadiče ve standardu JESD 204B

Název diplomové práce anglicky:

JESD 204B Rx Controller Design

Pokyny pro vypracování:

1. Seznamte se s protokolem JESD204 ve verzi B a C a multigigabitovými transceivery v Xilinx FPGA.
2. Porovnejte verze A, B a C protokolu JESD204, jejich výhody, nevýhody a jejich použití.
3. V jazyce Verilog2001 implementujte přijímací linkovou vrstvu protokolu JESD204B/C (kontrolér) s kódováním 8B/10B.
4. Ověřte funkci řadiče simulací na RTL úrovni pomocí referenční implementace s Xilinx JESD204C Tx IP a popřípadě validací na desce s FPGA.

Seznam doporučené literatury:

[1] JESD204 standard, https://www.jedec.org/document_search?search_api_views_fulltext=jesd204
[2] Úvod do přehledu JESD204 od Analog Devices,
https://www.analog.com/media/en/technical-documentation/technical-articles/JESD2
04B-Survival-Guide.pd

Jméno a pracoviště vedoucí(ho) diplomové práce:

prof. Ing. Pavel Hazdra, CSc., katedra mikroelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Ing. Martin Hujer, Ph.D., Dialog Semiconductor CZECH, s.r.o.,Klicperova3208/12, Praha 5

Datum zadání diplomové práce: **26.01.2021** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2022**

prof. Ing. Pavel Hazdra, CSc. podpis vedoucí(ho) práce	prof. Ing. Pavel Hazdra, CSc. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Acknowledgements

Firstly, I would like to thank the supervisor specialist of my master thesis Mr. Ing. Martin Hujer, Ph.D. from Dialog Semicondutor CZECH company for consultation and factual comments. Secondly, my thanks go to prof. Ing. Pavel Hazdra, CSc. from CTU in Prague for enabling and supervising of my thesis. An lastly, I would like to thank my family for their support during the whole study.

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.

In Prague, 21 May 2021

.....
Bohdan Jůza

Abstract

The master thesis deals with the design of the receiving link layer according to the JESD204B standard. It introduces the history of the standard from its first version to the current revision C. It delves deeper into the theoretical foundations of the revision B, from the perspective of the link layer. Further, the designed modules themselves are described. First the main one and then the ones of which it consists. Finally, the verification process is also shown, most notably the simulation of the designed block. The possibility of implementing the device in FPGA for a proper verification is also outlined.

Keywords: Verilog, Xilinx, FPGA, JESD204, JESD204B

Supervisor: prof. Ing. Pavel Hazdra, CSc.
CTU Faculty of Electrical Engineering,
Technická 1902/2,
16027 Prague, Czech Republic

Abstrakt

Diplomová práce se věnuje návrhu linkové vrstvy přijímače dle standardu JESD204B. Seznamuje s historií standardu od své první verze až po současnou revizí C. Hlouběji se věnuje představení teoretických základů revize B, a to z pohledu linkové vrstvy. Dále jsou popsány samotné navržené moduly. Nejprve ten hlavní a pak také ty, z kterých se skládá. Na závěr je rovněž ukázán proces verifikace, nejvíce pak simulace navrženého bloku. Nastíněna je i možnost implementace zařízení v FPGA pro plnohodnotnou verifikaci.

Klíčová slova: Verilog, Xilinx, FPGA, JESD204, JESD204B

Překlad názvu: Návrh Rx řadiče ve standardu JESD 204B

Contents

1 Introduction 1

2 Theoretical Background 3

2.1 JESD204 in General 3
2.2 Motivation to Use JESD204 3
2.3 Development of Revisions 4
2.3.1 Original JESD204 4
2.3.2 JESD204A 5
2.3.3 JESD204B 6
2.3.4 JESD204C 7
2.4 LVDS compared to JESD204 7
2.5 Layer Architecture in JESD204B 8
2.6 JESD204B in Detail 10
2.6.1 Deterministic Latency 10
2.6.2 Device clock 13
2.6.3 Frame and Multiframe Clock 14
2.6.4 Scrambling 14
2.6.5 8B/10B Coding 15
2.7 Link Layer Operation of Receiver 19
2.7.1 Code Group Synchronization 19
2.7.2 SYNC~ Signal Combining . . . 21
2.7.3 Frame Synchronization 21
2.7.4 Lane Synchronization 23
2.8 Implementation of JESD204 protocol in ASIC 24
2.9 Xilinx Multigigabit Transceiver . 25
2.10 Receiving Side of JESD204 Protocol on FPGA 27
2.11 Verilog 28

3 Development of Receiving Link Layer of JESD204B Protocol 31

3.1 Development Tools 32
3.1.1 Vivado Design Suite 2020.1 . . 32
3.1.2 ModelSim PE Student Edition 10.4a 32
3.1.3 Cadence Xcelium 32
3.2 Design 33
3.2.1 Top Module of Receiving Link Layer of JESD204B 33
3.2.2 Designed Submodules 38
3.3 Verification 48
3.3.1 Simulation of Xilinx JESD204 TX IP in Vivado 48
3.3.2 Simulation of DUT 50
3.3.3 Simulation of DUT with Xilinx IPs 53
3.4 Synthesis 59
3.4.1 FPGA 59
3.4.2 ASIC 60
3.5 Validation on FPGA 61

4 Future Updates 63

5 Conclusion 65

Bibliography 67

A Contents on the enclosed CD 69

Figures

2.1 CMOS, LVDS and CML power consumption dependency on sample rate [6] 4
2.2 JESD204 original standard illustration 5
2.3 JESD204A standard illustration . 6
2.4 JESD204B standard illustration . 6
2.5 System design using LVDS 8
2.6 System design using JESD204 . . . 8
2.7 Layer architecture of JESD204B 10
2.8 Illustration of deterministic latency 11
2.9 Timing diagram illustration for deterministic latency by SYSREF [1] 12
2.10 Serial implementation of scrambler 15
2.11 Code group synchronization for subclass 0 20
2.12 Code group synchronization for subclasses 1 and 2 20
2.13 Illustration of SYNC~ signal combining 21
2.14 Initial lane alignment sequence with four multiframes [1] 24
2.15 Illustration of example implementation of JESD204 in ASIC 25
2.16 GTP RX Transceiver Block Diagram [11] 26
2.17 Illustration of receiving side of JESD204 protocol implemented on FPGA 27

3.1 Graphical representation of designed JESD204B receiver for $NR\_OF\_LANES = 1$ 34
3.2 Simplified graphical representation of internal structure 36
3.3 Graphical representation of the module *cgs* 38
3.4 State machine implemented in module *cgs* 39
3.5 Graphical representation of the module *s3_8b10b_decoder* 40
3.6 Graphical representation of the module *cg_check* 40
3.7 State machine implemented in module *cg_chceck* 41
3.8 Graphical representation of the module *clock_gen* 42
3.9 Graphical representation of the module *frame_lane_align* 43
3.10 Illustration of data flow in the *frame_lane_align* module 43
3.11 Graphical representation of the module *descrambler* 46
3.12 Graphical representation of the module *controller* 47
3.13 Idea scheme of simulation 49
3.14 Transmitter core overview [12] . 50
3.15 Idea scheme of simulation 51
3.16 Code coverage of particular submodules 53
3.17 Graphical representation of *gearbox* 54
3.18 Idea scheme of simulation 55
3.19 Idea scheme of simulation 56
3.20 Screenshot of messages in simulation console 57
3.21 Screenshot of waveform 58
3.22 Data path illustration for validation on FPGA 61

Tables

2.1 Comparison between the CMOS, LVDS and JESD204B in the pin count for a 200 MSPS ADC [4] 5
2.2 Comparison between LVDS and JESD204 revisions 9
2.3 Electrical specifications for JESD204B [5] [2] 10
2.4 Rules for 5B/6B encoding..... 17
2.5 Rules for 3B/4B encoding..... 17
2.6 Control characters used in JESD204 18
2.7 Rules for running disparity..... 18
2.8 Link configuration parameters [1] 29

3.1 Description of sharing submodules in multiple lane configuration 33
3.2 Ports description for the JESD204B receiver..... 35
3.3 Parameters of designed block... 37
3.4 Ports description for the *cgs* module 38
3.5 Ports description for the *s3_8b10b_decoder* module 40
3.6 Ports description for the *cg_check* module 41
3.7 Ports description for the *frame_lane_align* module 44
3.8 Ports description for the *descrambler* module 47
3.9 Ports description for the *controller* module 48
3.10 Ports description for the *gearbox* module 54
3.11 Parameters of the example simulation 57
3.12 Parameters for synthesis 59
3.13 Post-Synthesis utilization of designed JESD204B receiver for FPGA..... 59
3.14 Post-Synthesis area report for ASIC..... 60
3.15 Post-Synthesis power report for ASIC..... 60

Chapter 1

Introduction

The master thesis deals with the design of the link layer of the receiver in the JESD204 standard, specifically in revision B. Four main goals were set for the work. The first one was an introduction to the JESD204 protocol in versions B and C, as well as to multigigabit transceivers within the Xilinx FPGA. The second task was to compare the individual versions A, B and C of the protocol with each other and to explain the advantages and disadvantages of their use in practice. The third and main goal was to design the link layer of the receiver itself in the specified standard with the 8B/10B coding using the Verilog2001 language. The fourth and last task was to verify the designed device by means of a RTL simulation using the reference Xilinx JESD204 TX IP core and possibly verify the system on a board with a FPGA. One side goal could be mentioned and it was designing in the Verilog language, which was a new experience.

The work itself is divided into five chapters including this introduction and the final conclusion. The first part after the introduction presents the theoretical basis of the JESD204 protocol. It deals with the reason for its implementation, the motivation why to use it, the comparison of individual versions with each other, which are also juxtaposed with the LVDS technology for a parallel signalling. Additionally, it focuses on the revision B itself, listing its main benefits and improvements over previous versions. The section focused on the link layer follows, its implementation was the main goal of this thesis. At the end of the chapter, there are also two sections focusing on Verilog and multigigabit transceivers within the Xilinx FPGA. The third chapter outlines the design of the receiver itself. At the beginning, the tools that were used for the development are listed. It is followed by presenting the designed block. The attention is focused on the design of the developed block from the outside. Then it forms a description of its internal connection with the designed submodules and their internal arrangement is described. The next section is dedicated to the verification of the developed device. It deals mainly with simulations in various simulation tools, the simulation procedure and lists other circuits that had to be designed with regard to the fact, that the resulting receiver was supposed to be validated on the board with a FPGA. The final information within this chapter is about the synthesized design of

the developed receiving link layer of the JESD204B protocol. The last but one chapter then outlines the direction in which the proven work could be followed up in the future and finally there is a conclusion at the end of this thesis.

Chapter 2

Theoretical Background

2.1 JESD204 in General

Analog to digital converters (ADC) and digital to analog converters (DAC) are becoming steadily more and more accurate and faster. It means that their interface is getting larger in the count of pins needed to transport the data and the output or input frequency has also grown. This is the reason, why the interface JESD204 was developed several years ago and has undergone revisions to make it even more suitable for this case of use. The JESD204 interface comes out with some crucial advantages over the conventional CMOS (Complementary Metal–Oxide–Semiconductor) and LVDS (Low-Voltage Differential Signaling) interfaces. The implementation of the JESD204 on the FPGA (Field Programmable Gate Array) or ASIC (Application Specific Integrated Circuit) yields to the ability to keep up with faster sample rates of converters. It reduces the pin count, so the resulting device has a smaller package size. The reduction of the pin count also leads to a decrease of trace routes, which makes the board design much easier and offers a lower overall system cost. The scalability is another benefit. The JESD204 standard was introduced in 2006 and since then it has been updated in three revisions, which have improved its efficiency. The latest one is the JESD204C revision. [4]

2.2 Motivation to Use JESD204

As requirements of converters were by time incessantly increasing, the development of their interface was performed too. For converters with a high data rate, wide resolution and demand for a low power consumption is currently JESD204, which uses the current mode logic (CML), concerned as the best way, how to transport the data for the further processing in a FPGA or ASIC. The CMOS technology lacks in the power consumption in higher speeds. The LVDS allows higher data rates, but it is also limited, due to the driver architecture and the necessity of the synchronization of all lanes, where the number of lanes could be significantly great. These statements are summarized in the figure 2.1 for the use with a dual 14-bit ADC. [4]

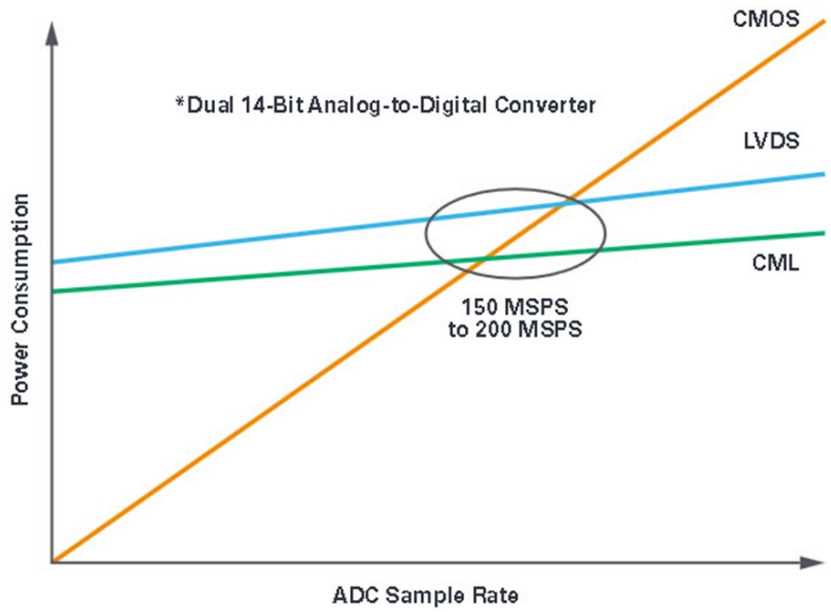


Figure 2.1: CMOS, LVDS and CML power consumption dependency on sample rate [6]

As stated above, the CML technology requires a fewer portion of pins at the same speed and resolution, as it is a serialized communication, when not talking about low data rates. Differences between technologies in this aspect are illustrated in the table 2.1. There is a comparison for a 200 MSPS ADC with a various number of channels and resolution. From the number of pins for each interface it is evident the advantage of using the JESD204B. It can be stated that the use of the JESD204 is not appropriate in cases of lower sample rates, approximately 150 MSPS, and if a minimal latency across the communication chain is needed. [3] [4]

2.3 Development of Revisions

2.3.1 Original JESD204

The original version of the JESD204 was launched in 2006. The standard was designed for a multigigabit serial data link between a converter, or multiple converters, and a receiver or transceiver, most often a device such as a FPGA or ASIC. This original version was limited for only one link connecting converter, or converters, and a receiver. The link could only consist of one lane. The illustration of this version is shown in the figure 2.2. The lane in the figure is a physical medium transporting a serialized data between M converters and a receiver. It is achieved by means of a differential pair working with the CML technology. As could be seen, the exactly same frame clock is provided for both sides, for converters and for a receiver. The source

Number of Channels	Resolution	CMOS Pin Count	LVDS Pin Count	JESD204B Pin Count
1	12	13	14	2
2	12	26	28	4
4	12	52	56	8
8	12	104	112	16
1	14	15	16	2
2	14	30	32	4
4	14	60	64	8
8	14	120	128	16
1	16	17	18	2
2	16	34	36	4
4	16	68	72	8
8	16	136	144	16

Table 2.1: Comparison between the CMOS, LVDS and JESD204B in the pin count for a 200 MSPS ADC [4]

and load impedances were defined as $100\,\Omega \pm 20\,\%$ and the differential voltage at the nominal value of 800 mV. The original version also supported the 8B/10B encoding. All above aspects led to the defined lane data rate within limits 312.5 Mb/s and 3.125 Gb/s. The standard became very popular and the first revision was needed. The goals were mainly about dealing with the increasing speed and resolution of converters. [1] [4]

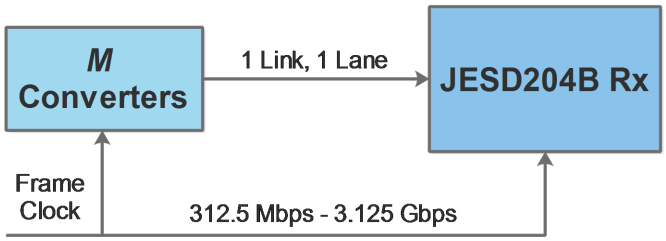


Figure 2.2: JESD204 original standard illustration

2.3.2 JESD204A

In 2008 the new revision of the standard called JESD204A came into the market. The main improvement was a support of multiple aligned serial lanes with multiple converters, as could be seen in the figure 2.3. Other specifications as the lane data rate, electrical specifications and the frame clock distribution remained unchanged. Now, it was possible to meet the maximum data rate 3.125 Gb/s for converters with a high sample rate and high resolution due to multiple aligned serial lanes. However, in the revised standard one important feature was missing. It was a deterministic latency, which is crucial for a correct interpretation of the received digital data in

some applications. This was the reason for developing a new revision of the standard. [4]

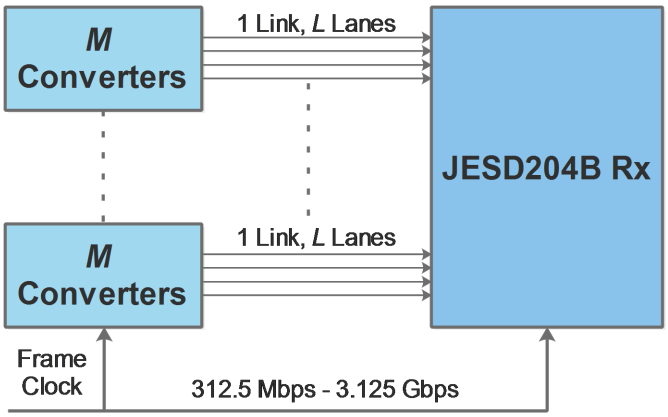


Figure 2.3: JESD204A standard illustration

2.3.3 JESD204B

The improved revision JESD204B was released in 2011. The key element missing in previous ones, deterministic latency, was since then included. Furthermore, several other features were introduced. Firstly, the supported data rate was raised up to 12.5 Gb/s. Three grades of devices differing in the data rate were distinguished. Speed grade 1 supports up to 3.125 Gb/s, the second one supports up to 6.375 Gb/s and the last one supports up to 12.5 Gb/s. They have also different electric specifications. Secondly, the common source of the frame clock was omitted and the main clock source specific for each device became a device clock. The figure 2.4 shows the illustration of the JESD204B revision. The JESD204B will be described in more detail further in the thesis. [1]

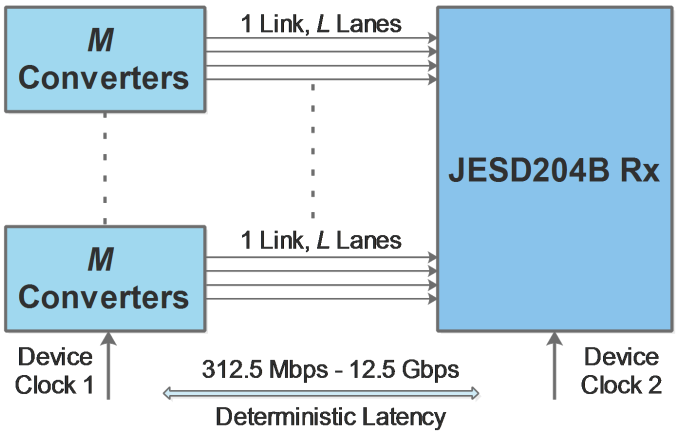


Figure 2.4: JESD204B standard illustration

2.3.4 JESD204C

The latest revision, JESD204C, was launched in 2017. It brings some further improvements of the revision B to achieve a greater data rate and more efficient transport of the payload data. The revision's C definition of the physical layer allows to greatly increase the data rate up to 32 Gb/s, while also providing a backward compatibility to the revision B. The efficiency improvement is mainly caused by changing the encoding scheme. Newly, it supports the 64B/66B and 64B/80B encoding beyond the 8B/10B encoding. They have a shorter coding overhead, so there is more place for data bits. Devices are similarly divided into device classes as in the previous revision in order to provide driver/receiver pairs that have varying amounts of a signal integrity processing to reduce the power in shorter channels. The downside of this revision can be a not full backward compatibility with the JESD204A revision. [2]

2.4 LVDS compared to JESD204

The parallel low voltage differential signalling (LVDS) is an older method to interface converters on a FPGA or ASIC. It was revised lastly in 2001 and it was a replacement of the RS-422 and RS-485 protocols, which had a higher power consumption and lower bandwidth. The LVDS uses differential signals with low voltage swings for a high speed data transmission.

The first problem of this technology lies in the low bandwidth for a use case with modern converters. The bandwidth of a differential LVDS wire is theoretically limited to 1.9 Gb/s, but in the real world it is maximally about 1.0 Gb/s. The second problem is a need of a great number of interconnects, as converters have a wider resolution. Therefore the JESD204 interface was developed. [5]

Figures 2.5 and 2.6 are illustrating differences in a system design using the LVDS for a parallel data transportation, respectively the JESD204 interface. It could be clearly seen that the JESD204 reduces the number of traces and the complexity of routing, simplifies the synchronization and the design is easily scalable. [5]

The table 2.2 summarises differences between the LVDS and all current revisions of the JESD204 standard. [1] [2] [5]

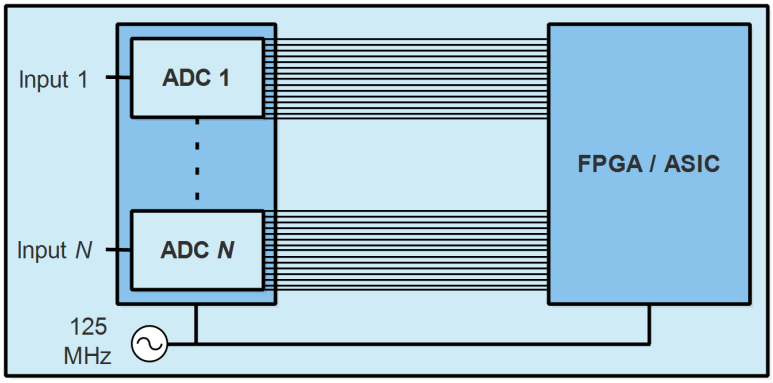


Figure 2.5: System design using LVDS

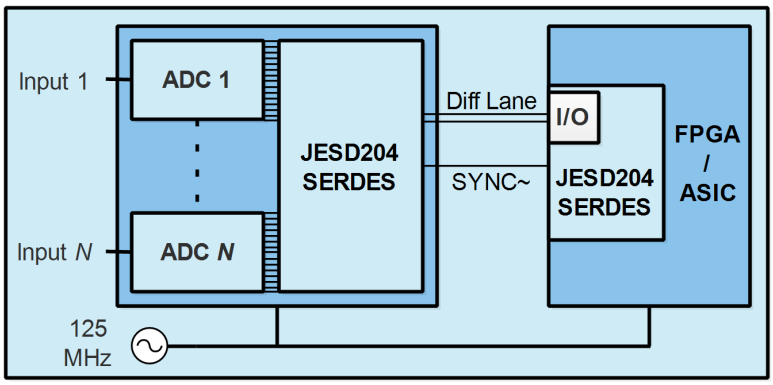


Figure 2.6: System design using JESD204

2.5 Layer Architecture in JESD204B

Similarly like the TCP/IP protocol, the JESD204 is divided into four individual layers, each with its own function. The layers are named as application layer, transport layer, data link layer and physical layer. The illustration of the layer composition is in the figure 2.7. [1]

Application Layer

The application layer is used for a special configuration and data mapping. Thanks to the specific framing of bits and other options, it is possible to achieve a reduction in the power consumption or to ensure a better data transfer from a converter with an atypical bit width. In that case, it is crucial to have configured the transceiver and receiver in the exactly same way. [8]

Transport Layer

The main task of the transport layer is to map samples from converter or converters to non-scrambled octets. There are several possibilities, how the mapping is done [1]:

Function	LVDS	JESD204	rev.A	rev.B	rev.C
Specification Realease	2001	2006	2008	2011	2017
Max Lane Rate [Gb/s]	1.0	3.125	3.125	12.5	32.0
Multiple Lanes	No	No	Yes	Yes	Yes
Lane Sync	No	No	Yes	Yes	Yes
Multidevice Sync	No	Yes	Yes	Yes	Yes
Deterministic Latency	No	No	No	Yes	Yes
Harmonic Clocking	No	No	No	Yes	Yes
Coding Scheme	No	8B/10B	8B/10B	8B/10B	8B/10B 64B/66B 64B/80B

Table 2.2: Comparison between LVDS and JESD204 revisions

- Single converter - single-lane link
- Multiple converters within the same device - single-lane link
- Single converter - multi-lane link
- Multiple converters within the same device - multi-lane link

The frame is then formed by a concatenation of F octets. There is also an option of sending more then one sample from a single converter in a single frame. This must be defined by the number S. [1]

Data Link Layer

Implementing the data link layer is the main purpose of this thesis and will be discussed in more detail later.

Physical Layer

The data serialization is realized in the physical layer. The physical layer contains serializer/deserializer (SerDes) blocks, drivers, receivers, and CDR. Also, the physical layer indicates the data rate speeds. As mentioned above, the JESD204B is divided into three grades. The maximum supported speed in JESD204B is 12.5 Gbps. The table 2.3 summarizes the electrical specifications for each grade. [8]

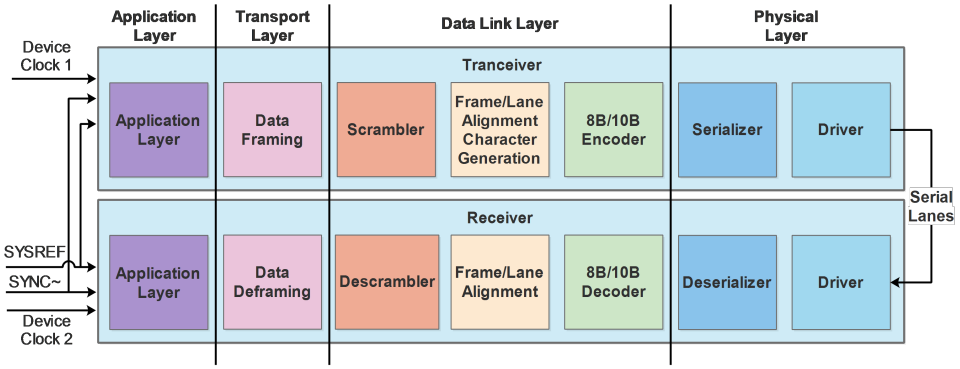


Figure 2.7: Layer architecture of JESD204B

Parameter	Grade 1	Grade 2	Grade 3
Line Rate [Gbps]	3.125	6.375	12.5
Out Differential Voltage [mVppd]	500 - 1000	400 - 750	360 - 770
Out Rise/Fall Time [ps]	50	30	24
Out Total Jitter (p-p UI)	0.35	0.30	0.30
p-p UI = peak-to-peak Unit Interval			

Table 2.3: Electrical specifications for JESD204B [5] [2]

2.6 JESD204B in Detail

This section will cover a closer look to the JESD204B interface, its features and standardizations. Some basic information were already provided in the section 2.3.3. According to the assignment, the main focus will be devoted to the link layer.

2.6.1 Deterministic Latency

In the engineering, a latency is a designation for the time elapsed between an action and reaction, or also the time required for the signal to pass between a point A and point B. For a JESD204 link is the point A the parallel input to the JESD204B transmitter and the point B corresponds to the parallel output of the JESD204B receiver's buffer. If so, it can be talked about the deterministic latency. It is important to not get confused with the term link delay. The link delay has an endpoint as an input to the receiver's buffer. The figure 2.8 illustrates the meaning of the deterministic latency. [1] [4]

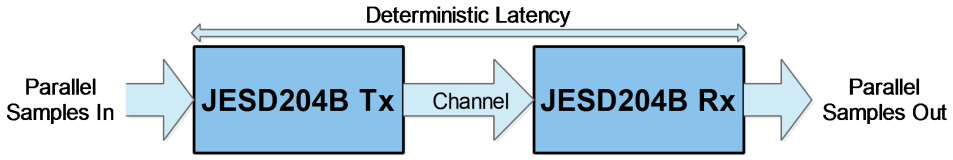


Figure 2.8: Illustration of deterministic latency

The JESD204B standard is divided into three subclasses according to how they achieve the deterministic latency:

- Subclass 0 - deterministic latency is not supported
- Subclass 1 - deterministic latency by means of signal SYSREF
- Subclass 2 - deterministic latency by means of signal SYNC~

Subclass 0

Due to a backward compatibility to the JESD204A revision is the JESD204B containing the subclass 0. The deterministic latency is not supported in this case. This could be useful for implementing the revision B with some older devices designed for the revision A. The subclass 0 has also different requirements for the SYNC~ signal from the subclass 1, as listed below. [4]

SYNC~ signal requirements (corresponding with the subclass 2) [1] [4]:

- The SYNC~ signal and receiver's and transmitter's frame clock must be synchronous.
- The SYNC~ can not be AC coupled.
- The device clock to the SYNC~ delay (t_{DS_R}) at the receiver device pins must be specified.
- The setup and hold time for the SYNC~ to the device clock at the transmitter must be specified.

The lane alignment in subclass 0 is achieved by means of an elastic buffer. The buffer is applied on each lane in the link. All incoming characters are stored in these buffers during the initial lane alignment sequence (ILAS) and after the arrival of the last lane's first start of multiframe control character (/R/ = K28.0), all buffers are released at the exactly same time. The same mechanism is applied also for the subclass 1 and 2, however the release time differs. [4]

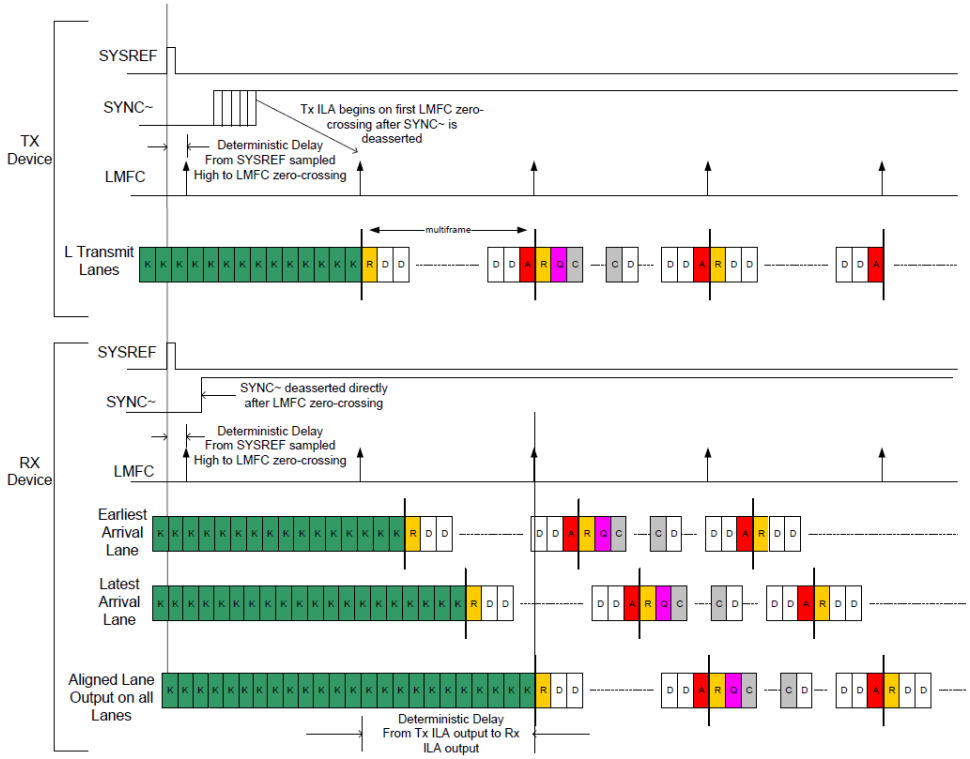


Figure 2.9: Timing diagram illustration for deterministic latency by SYSREF [1]

Subclass 1

In a technical practice, there are many applications where we need not only the synchronization across one link or multiple devices, but also we need to know the exact latency of the information transfer from the converter to the logical device.It means the deterministic latency in this case. For instance, some ADCs realize their calibration by means of a feedback loop and for this is a knowledge of the latency a crucial requirement. In other words, the arriving of the data must be stable after every power cycle. [4]

In the subclass 0, the receiver was monitoring all lanes until the first /R/ control character on latest lane arrived. Then the sample data were released. This solution is not counting with the time variation between each power cycle. The subclass 1 is synchronizing the release to the external SYSREF signal. This is done by the phase aligning of the local multiframe clock (LMFC) to the SYSREF signal. The illustration of the timing in the subclass 1 could be seen in the figure 2.9. The application of the subclass 1 is suitable for the data rate above 500 MSPS. [4]

Requirements for subclass 1 [1] [4]:

- The delay between the leading edge of the SYSREF signal and the frame and multiframe boundary must be specified for all devices in the JESD204B system.
- The buffer depth is defined by the receive buffer delay (RBD) and is ranging from 1 to K frame cycles. Mostly is the RBD set to 32.
- If the system needs a multichip synchronization between JESD204 devices, the use of the exactly same model of the converters is required. Otherwise, there could be problems with the deterministic latency.
- The inter-device lane skewness must be minimized.
- The device clock and SYSREF generation from the same clock source is required. The inter-device skewness must be also minimized.

Subclass 2

If a particular application requires a reduction of the pin and net count, it is appropriate to use the realization of the deterministic latency by means of SYNC~ signal. This is called subclass 2. The subclass 2 is suitable for the data rate below approximately 500 MSPS. The provision of the deterministic latency this way is possible because the SYNC~ is derived from the receiver's LMFC. Using this knowledge, the synchronization between the receiver and transmitter can be ensured. [1] [4]

2.6.2 Device clock

One improvement, which the JESD204B revision came up with, was the absence of the clock interconnection between the transmitter and receiver. Each device has its own device clock. The device clock is a timing reference specific for each device, but it must be derived from the common source, the source clock. The frequency of the device clock and frame or multiframe clock may vary. This means that each device has to generate its own local frame and multiframce clock with exactly the same frequency in a receiver and transmitter. [1]

Relationships between device clocks, frame or multiframe clocks differs across particular subclasses as follows:

- Subclass 0: Specified by the device implementer [1]
- Subclass 1: The multiframe period shall be a whole number of device clock periods [1]
- Subclass 2: The multiframe period shall be a whole number of device clock periods. Additionally, the TX device clock period shall be a whole number of RX device clock periods, or the RX device clock period shall be a whole number of TX device clocks periods. [1]

2.6.3 Frame and Multiframe Clock

The importance of the frame clock is in the establishing of the interface from the link layer to the application layer and conversely. The data are composed into multiframes, which are aligned with the local multiframe clock (LMFC). The clock reference is called local, if it is derived from the device clock. In other words, when the clock is not directly supplied on the input to the device. The phase alignment of the LMFC is intended by the SYSREF signal for the subclass 1 or by the SYNC~ signal for the subclass 2, not supported in the subclass 0. [1]

Requirements for frame and multiframe clocks are as follows:

- Identical frame frequency across all devices [1]
- Identical multiframe frequency across all devices [1]
- Both derived from the device clock [1]
- Phase aligning of the frame clock and LMFC within each device [1]
- The SYSREF for the subclass 1 and the SYNC~ for the subclass 2 intends the phase of the frame clock and LMFC [1]
- For multiple links, each above requirement is applied separately on each link [1]

2.6.4 Scrambling

The scrambling is an optional function in the JESD204, however, all devices shall support this technique. If the octets transmitting in the JESD204 system are repetitive frame to frame, there is a possibility of an occurrence of the spectral peaks. This can lead to problems with the electromagnetic compatibility (EMC), DC offsets or interferences. The application of the scrambling also makes the spectrum data-independent. On the other hand, this operation can have some negative effects. One disadvantage could be a switching noise, whose amount may be so great, that the disabling of the scrambling would be advantageous. [1]

The scrambling operation is located between the transport and link layer. There is one scrambler per a particular lane. The enabling of the scrambling means that all scramblers and descramblers on all links and all lanes are enabled. There is no support for a mixed mode. [1]

The scrambler shall be of the self-synchronous type with the polynomial $1 + x^{14} + x^{15}$. [1] The serial implementation is illustrated in the figure 2.10

One important thing is the initial state of the scrambler. In order to not send repetitive octets, the scrambler must have initiated internal registers.

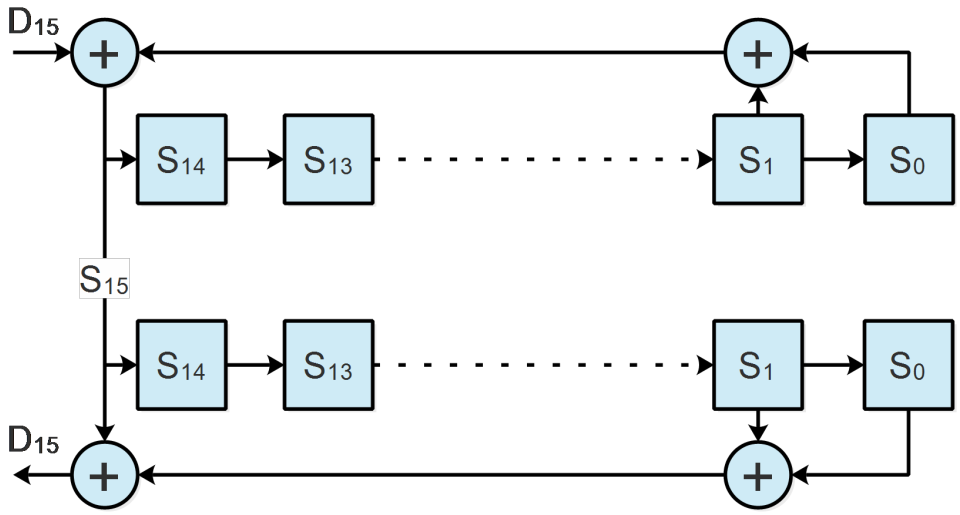


Figure 2.10: Serial implementation of scrambler

It is recommended to preset the upper eight registers to logical 1 and lower seven registers to logical 0. For descrambler, there is no need for presetting registers as the descrambler is self-synchronized. [1]

2.6.5 8B/10B Coding

Before transmitting to the receiver, the data are encoded by the 8B/10B coding. An 8-bit octet is encoded to a 10-bit character, which is then transmitted. This coding scheme is used to achieve a DC balance on lanes and a bounded disparity. The coding ensures, that the count of ones and zeros in the 20-bit long sequence does not differ more than by two and it is impossible to have more than five ones or zeros in a row. Such changes in states are useful for the clock recovery. [7]

The 8B/10B coding is separated in two processes. An input octet is divided in two groups, the lower 5 bits portion and upper 3 bits portion. The first group is encoded to a 6-bit sequence and second group to a 4-bit sequence. These sequences are concatenated together and this is how the 10-bit character is made. The code group (CG) characters are called as D.x.y, where x ranges from 0 to 31 and y ranges from 0 to 7. In addition, 12 special characters (control characters) are defined. They are called as K.x.y and indicates some special events. [7]

The rules for the 5B/6B encoding are summarized in the table 2.4 and the rules for the 3B/4B encoding are summarized in the table 2.5. Control characters, which are used in the JESD204, are summarized in the table 2.6.

Running Disparity

As mentioned above, 8B/10B coding is DC balanced. It means, that in long period the counts of ones and zeros are equal. In other words, the ratio between ones and zeros is exactly 50 %. This is achieved by the fact, that the difference between ones and zeros in each coding group (5B/6B and 3B/4B) is in range from -2 to +2. At the end of the 8B/10B coding is the difference either +1 or -1. This is called as the running disparity (RD). [7] The rules for calculating the running disparity for each sub-block are as follows.

The running disparity at the end of any sub-block is positive, if

- the count of ones is greater than count of zeros in a sub-block [7],
- the six-bit sub-block is 000111 [7],
- the four-bit sub-block is 0011. [7]

The running disparity at the end of any sub-block is negative, if

- the count of ones is lower than count of zeros in a sub-block [7],
- the six-bit sub-block is 111000 [7],
- the four-bit sub-block is 1100. [7]

If none of the condition above occurred, the running disparity at the end of the sub-block is the same as the running disparity at the beginning of the sub-block. [7]

The rules for the running disparity are summarized in the table 2.7.

Input		RD-	RD+	Input		RD-	RD+
CG	Octet Bits EDCBA	Encoded Bits abcdei		CG	Octet Bits EDCBA	Encoded Bits abcdei	
D.00	00000	100111	011000	D.16	10000	011011	100100
D.01	00001	011101	100010	D.17	10001	100011	
D.02	00010	101101	010010	D.18	10010	010011	
D.03	00011	110001		D.19	10011	110010	
D.04	00100	110101	1010	D.20	10100	001011	
D.05	00101	101001		D.21	10101	101010	
D.06	00110	011001		D.22	10110	011010	
D.07	00111	111000	000111	D.23	10111	111010	000101
D.08	01000	111001	000110	D.24	11000	110011	001100
D.09	01001	100101		D.25	11001	100110	
D.10	01010	010101		D.26	11010	010110	
D.11	01011	110100		D.27	11011	110110	001001
D.12	01100	001101		D.28	11100	001110	
D.13	01101	101100		D.29	11101	101110	010001
D.14	01110	011100		D.30	11110	011110	100001
D.15	01111	010111	101000	D.31	11111	101011	010100
not used		111100	000011	K.28	11100	001111	110000

Table 2.4: Rules for 5B/6B encoding

Input		RD-	RD+	Input		RD-	RD+
CG	Octet Bits HGF	Encoded Bits fghj		CG	Octet Bits HGF	Encoded Bits fghj	
D.x.0	000	1011	0100	K.x.0	000	1011	0100
D.x.1	001	1001		K.x.1	001	0110	
D.x.2	010	0101		K.x.2	010	1010	
D.x.3	011	1100		K.x.3	011	1100	
D.x.4	100	1101	0010	K.x.4	100	1101	
D.x.5	101	1010		K.x.5	101	0101	
D.x.6	110	0110		K.x.6	110	1001	
D.x.P7	111	1110	0001	K.x.7	111	0111	1000
D.x.A7	111	0111	1000	P7 or A7 selected to avoid 5 times 0 or 1 in a row			

Table 2.5: Rules for 3B/4B encoding

Char	CG	Octet Bits HGFEDCBA	Encoded abcdeifghj RD-	Encoded abcdeifghj RD+	Note
/R/	K.28.0	00011100	0011110100	1100001011	Start of multiframe
/A/	K.28.3	01111100	0011110011	1100001100	Lane alignment
/Q/	K.28.4	10011100	0011110010	1100001101	Start of link configuration data
/K/	K.28.5	10111100	0011111010	1100000101	Code group synchronization
/F/	K.28.7	11111100	0011111000	1100000111	Frame alignment

Table 2.6: Control characters used in JESD204

Previous RD	Disparity of Code Word	Disparity Chosen	Next RD
-1	0	0	-1
-1	±2	+2	+1
+1	0	0	+1
+1	±2	-2	-1

Table 2.7: Rules for running disparity

2.7 Link Layer Operation of Receiver

The main goal of this thesis was the implementation of the link layer of the JESD204B interface on the receiver's side. Link layer processes will be described in following subsections.

2.7.1 Code Group Synchronization

The code group synchronization is a process used to detect the start and the end of the character transmitted via the serial interface after the deserialization. This is provided by the receiving of /K/ control characters. When the receiver issues the synchronization request via the SYNC~ interface, the transmitter begins to emit the K symbols. The receiver detects the K symbols and wait until the reception of four K symbols in a row. After this, the receiver is synchronized and deasserts the SYNC~ interface. [1]

At this point, the subsequent procedures of the JESD204B are divided according to the subclasses.

For the subclass 0 transmitters are the rules as follows:

- After the detection of the deactivation of the synchronization request by all receivers, the K symbols continue to being transmitted by transceivers until the start of the next frame. [1]
- With the start of the next frame, transmitters transmit the initial lane alignment sequence (ILAS). [1]

For subclasses 1 and 2 transmitters are the rules as follows:

- After the detection of the deactivation of the synchronization request by all receivers, the K symbols continue to being transmitted by transceivers until the next local multiframe clock boundary. (There is an option to use also some later LMFC boundary by programming it so) [1]
- With the start of the next frame following the chosen LMFC boundary, transmitters begin to transmit the ILAS. [1]

Code Group Synchronization Check

After the achievement of the synchronicity, characters are further transmitted to the following design. Although, there is a possibility, that invalid octet is received. The invalidity can be caused by the running disparity error or code error. These errors are detected by the 8B/10B decoder. Whenever the invalid character is received, the code group synchronization enters to the check phase. During this phase, the design starts counting valid and invalid characters. There are two options, which may occur. The first option is, that three invalid characters are detected and the synchronization request is issued.

The second variant is the detection of four consecutive valid characters and entering to the normal, synchronized, phase. It must be stated, that during the checking phase characters are still proceeding to the further logic, even if the character is invalid. [1]

The code group synchronization process is illustrated in the figure 2.11 for the subclass 0 and in the figure 2.12 for subclasses 1 and 2 supporting the deterministic latency. [1]

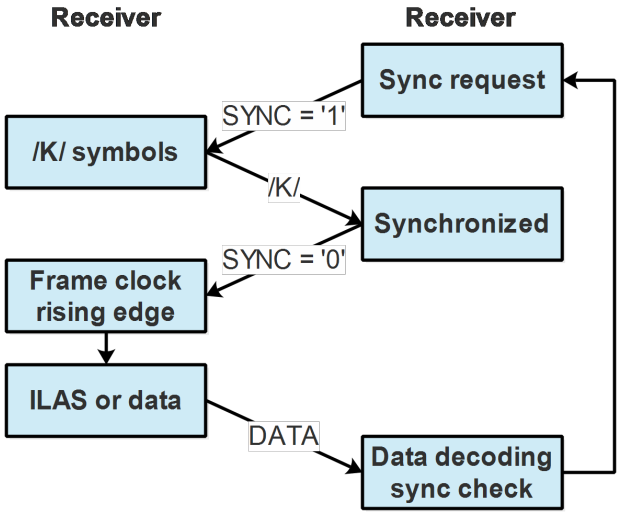


Figure 2.11: Code group synchronization for subclass 0

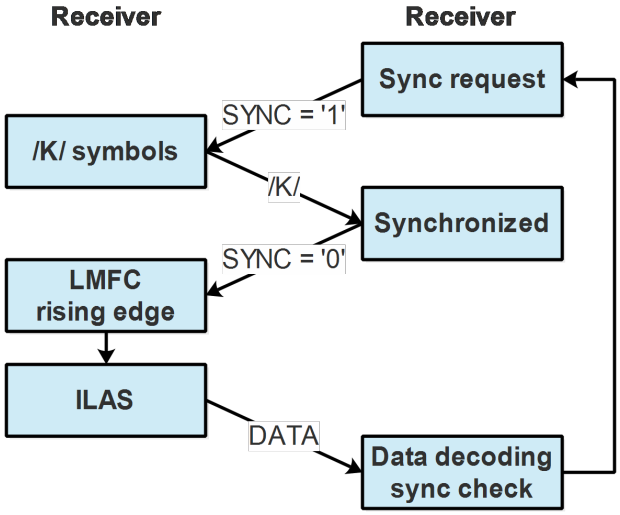


Figure 2.12: Code group synchronization for subclasses 1 and 2

2.7.2 SYNC~ Signal Combining

Synchronization requests from single links can be combined into one signal when using multipoint links. Using this technique can one single receiver affect all other receivers combined by the SYNC~ interface. If one of the receivers issues a synchronization request, all transmitters begin to emit /K/ symbols. This is mandatory for subclass 0 devices, because of aligning the ILAS generation across all links. [1] The combination can be done in two possible ways:

- inside the receiver device, [1]
- inside the transceiver device. [1]

The illustration of both implementations is in the figure 2.13.

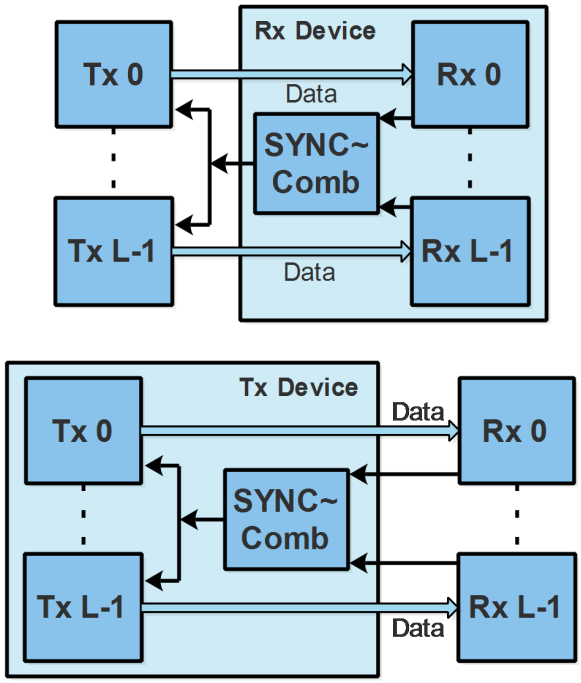


Figure 2.13: Illustration of SYNC~ signal combining

2.7.3 Frame Synchronization

Initial Frame Synchronization

At the beginning of the link establishment, the first non /K/ symbol (also called comma) is the indication of the start of the frame. The length of the frame is defined by the number F, number of octets per frame. This means that the next rising edge of the frame clock is associated with the resetting of the octet counter. [1]

Alignment Characters

There are two types of alignment characters in the JESD204B. They are represented as special characters /F/ = K28.7 and /A/ = K28.3. The use of the /A/ character is possible only if both, transceiver and receiver, supports lane synchronization. [1]

Character Replacement without Scrambling

In this part, the thesis will focus on that cases, when both sides support the lane synchronization.

Rules for character replacement are as follows:

- If the octet on the last position of the frame, which is about to be transmitted, is the same octet as in the previous frame, the octet shall be replaced by the /F/ character. However, this position must not be the last octet of the multiframe. Another situation, when the /F/ character is not inserted, is, if the /F/ character was replaced in the previous frame. [1]
- Analogously, if the same octet is at the end of the multiframe, the octet is replaced by the /A/ character. This is also the case, if the /F/ was sent in the previous frame. [1]
- Receiver shall store the data from the previous frame and if the /F/ or /A/ is received, the receiver shall replace the control character with the octet stored on the same position as in the previous frame. [1]

Character Replacement with Scrambling

Similarly like the previous subsection, this part of the thesis will focus on that cases, when both sides support the lane synchronization.

Rules for character replacement are as follows

- If the scrambled octet on the last position of the frame, which is about to be transmitted, is equal to 0xFC, the octet shall be replaced by the /F/ character. However, this position must not be the last octet of the multiframe. [1]
- Analogously, if the scrambled octet of the value 0x7C is at the end of the multiframe, the octet is replaced by the /A/ character. [1]
- Receiver shall replace /F/ or /A/ characters with 0xFC, respectively 0x7C, on the input of the descrambler. [1]

Frame Alignment Correction

The frame realignment is realized as follows:

- If two alignment characters are received at the same position, which is not as expected for the current frame align, without an interruption of the alignment character at the correct position, the realignment is done in the accordance to detected two alignment characters. [1]
- If the lane alignment is issued, it indicates a "cross coupling" and there is no need for waiting for two alignment characters as in the previous point. The frame realignment is realized after detecting one alignment character. [1]
- There shall be an option to disable the process described in the previous points on the receiver's side. Realignment could be problematic, if not enough alignment characters are emitted or if the realignment could cause the lane alignment and latency error. [1]

2.7.4 Lane Synchronization

Initial Lane Synchronization

Before the reception of the user data, the initial lane alignment must be achieved. The initial lane alignment sequence (ILAS) usually consists of 4 multiframes (in some cases it can be up to 256 multiframes, this must me programmable). Each of the multiframe is started with the /R/ character and ended with the /A/ control character. Because of the possible difference in the lane delays, this character can be received at a different time. After the reception of the first /R/ character, each receiver start to store the incoming data in the buffer and they raise the "ready" flag. When all receivers have detected the first /R/ character and raised their flags, they start to propagate the stored data from the buffer for a further logic processing. In addition, the multiframe consists of K frames. The K is ranging form 1 to 32 frames and this value shall be also programmable. [1]

During the ILAS, each multiframe is introduced by the /R/ control character and ended with the /A/ control character. The second multiframe is specific. It transmits configuration information about the JESD204B link. These information are introduced with the /Q/ control character. The figure 2.14 is showing the ILAS with four multiframes.

Meanings of parameters transmitted during the ILAS are summarized in the table 2.8.

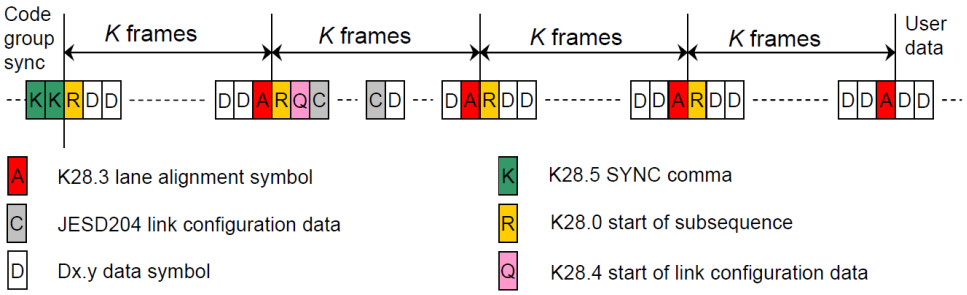


Figure 2.14: Initial lane alignment sequence with four multiframes [1]

Lane Alignment Monitoring and Correction

After ILAS, the payload of user data begins to transmit over the JESD204B link. At this time, the lane alignment is monitored and eventually corrected. It is made by means of monitoring the arrivals of /A/ characters. [1]

The rules for the dynamic realignment are analogous to those ones presented for the frame realignment:

- If two /A/ characters are received at the same position, which is not as expected for the current lane align, without an interruption of the /A/ character at the correct position, the realignment is done in the accordance to detected /A/ characters. [1]
- If the frame alignment is issued and it indicates a "cross coupling", there is no need for waiting for two /A/ characters as in te previous point. The lane realignment is realized after detecting one /A/ character. [1]

2.8 Implementation of JESD204 protocol in ASIC

The main goal of this thesis was to develop a receiving link layer of the JESD204B protocol suitable for use in an ASIC. The implementation in an ASIC of the whole data path from the JESD204 transmitter to the receiver is shown in the figure 2.15. The configuration with one lane and 100 MHz frequency is presented. Input data are mapped into octets in the transport layer of the JESD204 protocol. Octets are proceeded to the link layer, where data are coded by 8B/10B coding scheme and optionally scrambled. Data are subsequentially written to the asynchronous FIFO from where are read by a serializer, which is an analog circuit used for serializing the data. An asynchronous FIFO is a device used to cross clock domains. Serial bits are then differentially transmitted from the transmitting ASIC to the receiving ASIC. Data are there deserialized and via another asynchronous FIFO provided to the receiving JESD204 link layer. Data are there descrambled and decoded and in the transport layer demapped from octets. The clocking source is common for both ASICs and each ASIC is provided with its own device clock for JESD204 devices.

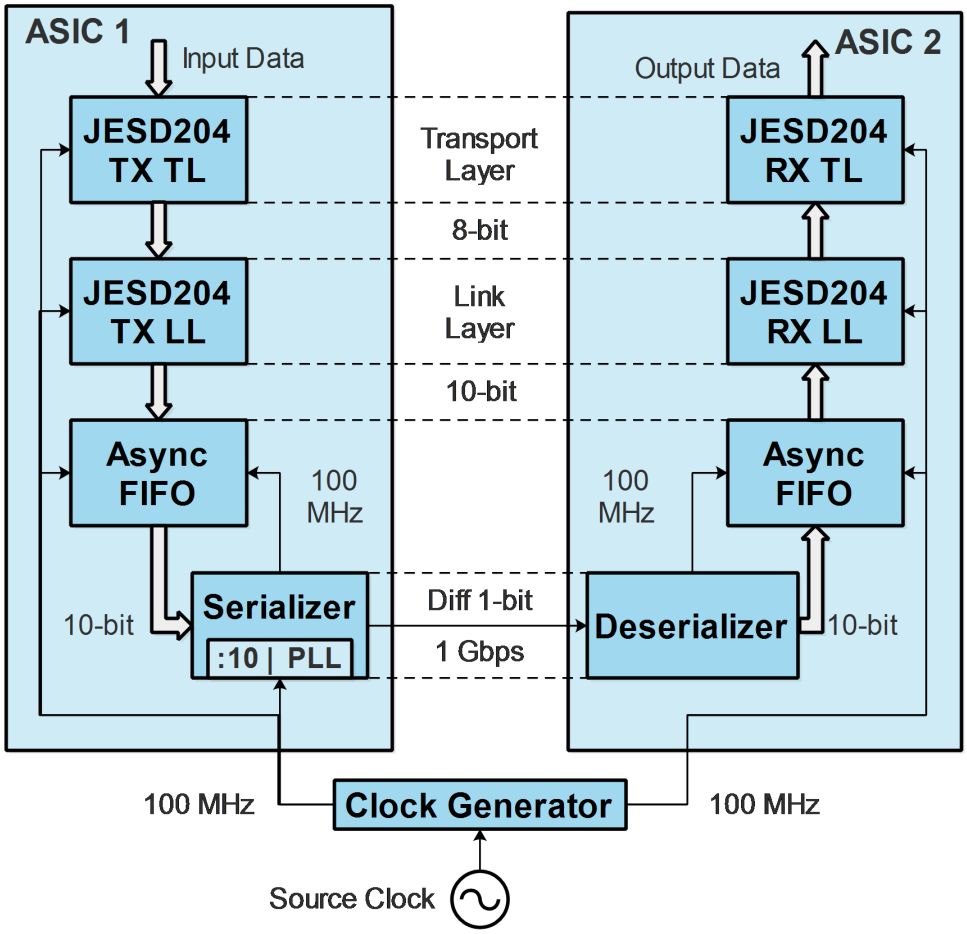


Figure 2.15: Illustration of example implementation of JESD204 in ASIC

2.9 Xilinx Multigigabit Transceiver

The pressure at increasing transfer speeds over long distances using fewer wires has led to the use of multi-gigabit transceivers (MGT). Multi-gigabit transceiver is a digital SerDes device, which can operate at serial rates higher than 1 Gb/s. In the case of vision of a deployment of the JESD204B receiver developed in this thesis in the Nexys Video FPGA with built in *Artix®-7 XC7A200T* chip made by Xilinx, we talk about the 7 series FPGAs GTP transceiver. [11] There is no possibility of implementing an analog SerDes in FPGA.

As Xilinx declare, the 7 series FPGAs GTP transceiver is a power-efficient transceiver, supporting line rates between 500 Mb/s and 6.6 Gb/s. The maximum rate is limited by a particular device. [11]

The key elements within the GTP RX transceiver are: [11]

- Analog Front End
- Out-of-Band Signalling
- Equalizer
- Clock Data Recovery
- Fabric Clock Output Control
- Margin Analysis
- Polarity Control
- Pattern Checker
- Byte and Word Alignment
- 8B/10B Decoder
- Elastic Buffer
- Gearbox

Many of them are irrelevant for the use with the designed JESD204B receiver and are bypassed. The block diagram of the GTP transceiver is displayed in the figure 2.16. The blue line illustrates the use case for this thesis.

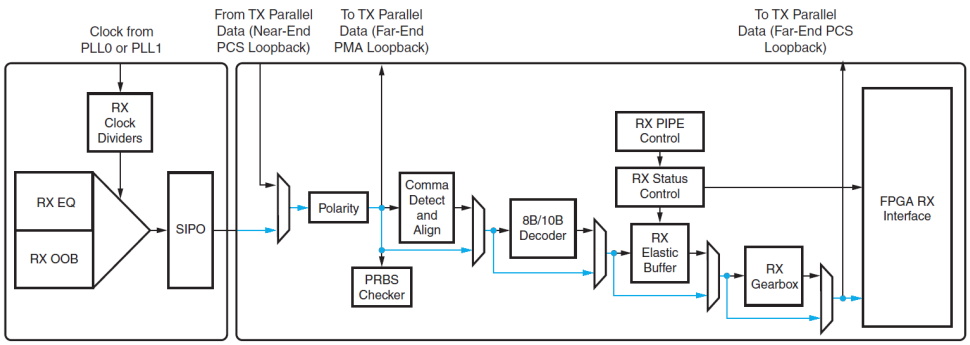


Figure 2.16: GTP RX Transceiver Block Diagram [11]

2.10 Receiving Side of JESD204 Protocol on FPGA

In the figure 2.17 there is a visualisation of the receiving side of the JESD204 system on a FPGA. Bit widths displayed are corresponding to the configuration with one lane per link. The first block in the chain is a multigigabit transceiver, because on a FPGA there can not be any analog SerDes implemented. It is supplied with a reference clock of the declared frequency. In case of this thesis, it just deserializes the data stream into 20-bit wide words. These words with a corresponding clock signal are then transmitted to the gearbox. This block is used for a conversion of the 20-bit data into 10-bit words synchronized to the second clock domain provided, because the receiver in this thesis was designed for a 10-bit input. The gearbox block is described further in the subsection 3.3.3. The 10-bit wide data with the corresponding clock domain are then transmitted to the designed block, the receiving link layer of the JESD204B protocol. Operations presented in sections above are performed there. The module is outputting the data octets. In the JESD204 protocol, the transport layer is responsible for demapping the data from octets and the data are then provided for a particular application.

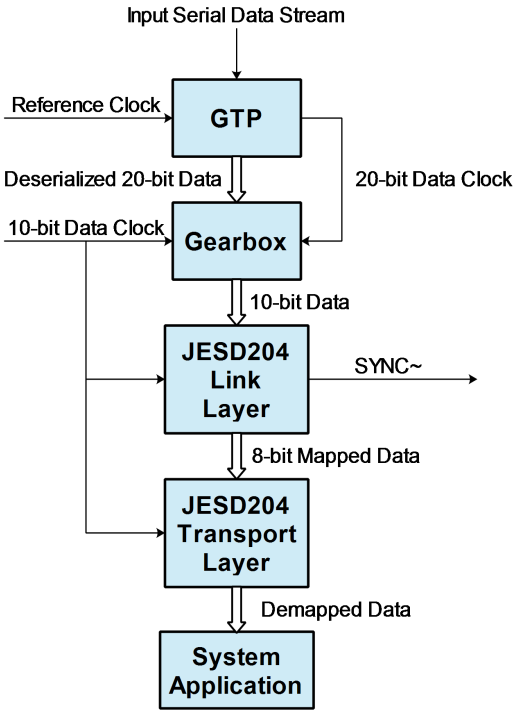


Figure 2.17: Illustration of receiving side of JESD204 protocol implemented on FPGA

2.11 Verilog

A few words about Verilog language will follow. Verilog belongs to a group of HDL (*Hardware Description Language*) languages used to design PLD (Programmable Logic Device), FPGA applications and also ASICs (*Application Specific Integrated Circuits*). Another language of this kind is VHDL or SystemVerilog. Both Verilog and VHDL were originally developed for simulation and documentation purposes, not for synthesis. Many of their constructs have signs of it and are not synthesizable, while are dedicated for modelling. [10]

The basic characteristics of the Verilog language are listed below [10]:

- Open standard - There is no requirement for a license from the owner to build a design, unlike some other HDL languages.
- Device independent design - They allow the designer to work without selecting the target circuit first.
- Portability - It is possible to simulate the designed circuit based on the same source text, which will then be used for the synthesis and implementation in the target circuit. The source code can be processed in various simulators and in synthesizers from various manufacturers. The simulated text can then be used in various target circuits in the future, which is made possible by the fact that Verilog supports the hierarchical structure of projects, where the created system consists of subblocks.
- ASIC compatibility - In case of successful launch on the market, the description of the design in HDL languages can be used as a basis for its implementation in the ASIC circuit suitable for large series.

The basic version of Verilog was adopted as IEEE (*Institute of Electrical and Electronics Engineers*) Standard No. 1364 in 1995. Constructs conforming to this standard are referred to as Verilog-95 constructs. Based on the experience with this version, a new version was adopted in 2001, Verilog-2001. A number of modifications and improvements have been made to this version and are supported by most current design systems. And in this version is also written this diploma thesis. [10]

Parameter	Description	Range	Value
ADJCNT	Number of adjustment resolution steps to adjust DAC LMFC Subclass 2 only	0–15	Binary
ADJDIR	Direction to adjust DAC LMFC 0 – Advance 1 – Delay Subclass 2 only	0–1	Binary
BID	Bank ID - Extension to DID	0–15	Binary
CF	Number of control words per frame clock per link	0–32	Binary
CS	Number of control bits per sample	0–3	Binary
DID	Device (= link) ID number	0–255	Binary
F	Number of octets per frame	1–256	Binary – 1
HD	High Density format	0–1	Binary
JESDV	JESD204 version 000 – JESD204A 001 – JESD204B	0–7	Binary
K	Number of frames per multiframe	1–32	Binary – 1
L	Number of lanes per converter device (link)	1–32	Binary – 1
LID	Lane ID number (within link)	0–31	Binary
M	Number of converters per device	1–256	Binary – 1
N	Converter resolution	1–32	Binary – 1
N'	Total number of bits per sample	1–32	Binary – 1
PHADJ	Phase adjustment request to DAC Subclass 2 only	0–1	Binary
S	Number of samples per converter per frame cycle	1–32	Binary – 1
SCR	Scrambling enabled	0–1	Binary
SUBCLASSV	Device Subclass Version 000 – Subclass 0 001 – Subclass 1 010 – Subclass 2	0–7	Binary
RES1	Reserved field 1	0–255	Binary
RES2	Reserved field 2	0–255	Binary
CHKSUM	Checksum $\sum$(all above fields) mod 256	0–255	Binary

Table 2.8: Link configuration parameters [1]

Chapter 3

Development of Receiving Link Layer of JESD204B Protocol

At the beginning of the chapter, there will be presented tools, which were used for the development of the design, for the behavioral simulation and also for the verification.

This chapter will further cover information about designed receiver itself. Then subblocks resulting in the desired receiver will be described. To make the explanation easier, the use with only one serial lane will be covered (unless otherwise stated), but the design is supporting the cases with more serial lanes between the transmitter and receiver.

A section about the verification process will follow. There will be presented simulations, which were done during the development. Subsections will gradually guide through particular steps of the simulation process. Several modules, which were designed for the simulation purposes, will be described.

The fourth section is dealing with the physical representation of the designed block representing the link layer of the JESD204B receiver. The utilization inside the Xilinx FPGA *xc7a200tsbv484-1* and for ASIC realized in the TSMC 28HPC+ technology will be outlined.

The last section is in short describing the validation in the chosen FPGA. Mainly the idea of this operation is discussed.

It must be stated that after a discussion with the supervisor of the thesis it was decided to implement only the subclass 0 of the JESD204B receiver and it was done due to the time pressure before the submitting of the thesis. For this reason, only subclass 0 will continue to be discussed in the work, and all blocks are fully compatible only with this subclass.

3.1 Development Tools

During the design flow of the JESD204B several tools were used. The first one was Vivado Design Suite 2020.1, see subsection 3.1.1, the second on was ModelSim PE Student Edition 10.4a, see subsection 3.1.2 and the last one was Cadence Xcelium, see subsection 3.1.3.

3.1.1 Vivado Design Suite 2020.1

The first development tool was XIlinx Vivado Design Suite in version 2020.1. Vivado Design Suite is a development environment from the Xilinx company for a synthesis and analysis that supports both VHDL and Verilog languages. It includes many tools that allow developers to properly debug Xilinx's FPGA or CPLD circuitry. This software is free to download in its HL WebPACK™ Edition. [9] At first, the premise was to utilize this tool for everything, because it is suitable for the code development, for the simulations and also for the verification with Xilinx IP *intellectual property* cores. However, it was found that simulations are too much time consuming even with just the original design and for simulations with Xilinx IP cores it was only hardly usable. So it was decided to write the design and do the basic simulations in another software described below in subsection 3.1.2. The chosen tool was ModelSim PE in its Student Edition 10.4a.

Another problem related with Vivado and the migration to ModelSim was the import of Xilinx IP cores to ModelSim. After some research was found, that the current version of the Vivado, which had to be used in case of generating the JESD204B Tx IP core, is not compatible with the most recent student version of ModelSim in order to transfer the IP core to the ModelSim. This fact led to the simultaneous design and simulation in both programs.

3.1.2 ModelSim PE Student Edition 10.4a

ModelSim delivered by Mentor Graphics is an alternative to Vivado in means of the developing and simulating the digital design, but for the needs of this thesis its performance was in simulations much more effective. But of course it has its limitations explained above. So, for the simulations purposes it was necessary to generate the data stream from the Xilinx IP core, save it to the text file and and then work with this file within the ModelSim environment.

3.1.3 Cadence Xcelium

In the end of the development of the receiver, another tool was used and it was Cadence Xcelium. Access to this software was provided by Dialog Semiconductor company. The use of this tool was necessary especially when simulating the system with all Xilinx IP cores. This means when using reference Xilinx JESD204 IP core and Xilinx GTP IP core. In that case, it offers a much better performance than other tools used.

3.2 Design

The section will describe the designed top module and also several submodules, of which the receiver consists. The beginning will be devoted to the top module and the rest of the section will be about submodules. Blocks will be presented both from the outside and their internal arrangement and behaviour.

3.2.1 Top Module of Receiving Link Layer of JESD204B

This subsection is presenting the top module, which is the desired designed JESD204B receiver, respectively its link layer part. The illustration for one lane configuration of it is in the figure 3.1. The table 3.2 is summarizing the meanings of input and output ports of the block. It should be stated, that it was decided for an implementation of 10-bit wide input data for a particular lane. This has been done due to the fact, that it is the simplest variant, however other possibilities are also suitable, for instance parallel processing of multiple data words. Finally, the figure 3.2 is displaying simplified internal structure of the design block. Input ports *clk_i* and *rst_n_i* are not routed in the picture, but in fact they are delivered to all of the submodules. Input ports *nr_f_ostets_i* and *nr_k_frames_i* are not routed too, however they are delivered to submodules *frame_lane_align*, *clock_gen* and only *nr_f_ostets_i* to *controller*. Also output ports of ILA configuration data are simplified. They are packed into one port called *ILA_data*. For detailed information about output ports see the table 3.2. For a configuration with multiple lanes, some of the submodules are shared for all lanes a some of them are instantiated for separate lanes. The table 3.1 is summarizing the information about this use.

Submodule	Shared / Separate
controller	Shared
cgs	Separate
s3_8b10b_decoder	Separate
cg_check	Separate
frame_lane_align	Separate
ila_combiner	Shared
descrambler	Separate
clock_gen	Shared
delay	Separate

Table 3.1: Description of sharing submodules in multiple lane configuration

The module must be instantiated with parameters to ensure the correct functioning. Meanings and ranges of the parameters are as in the table 3.3.

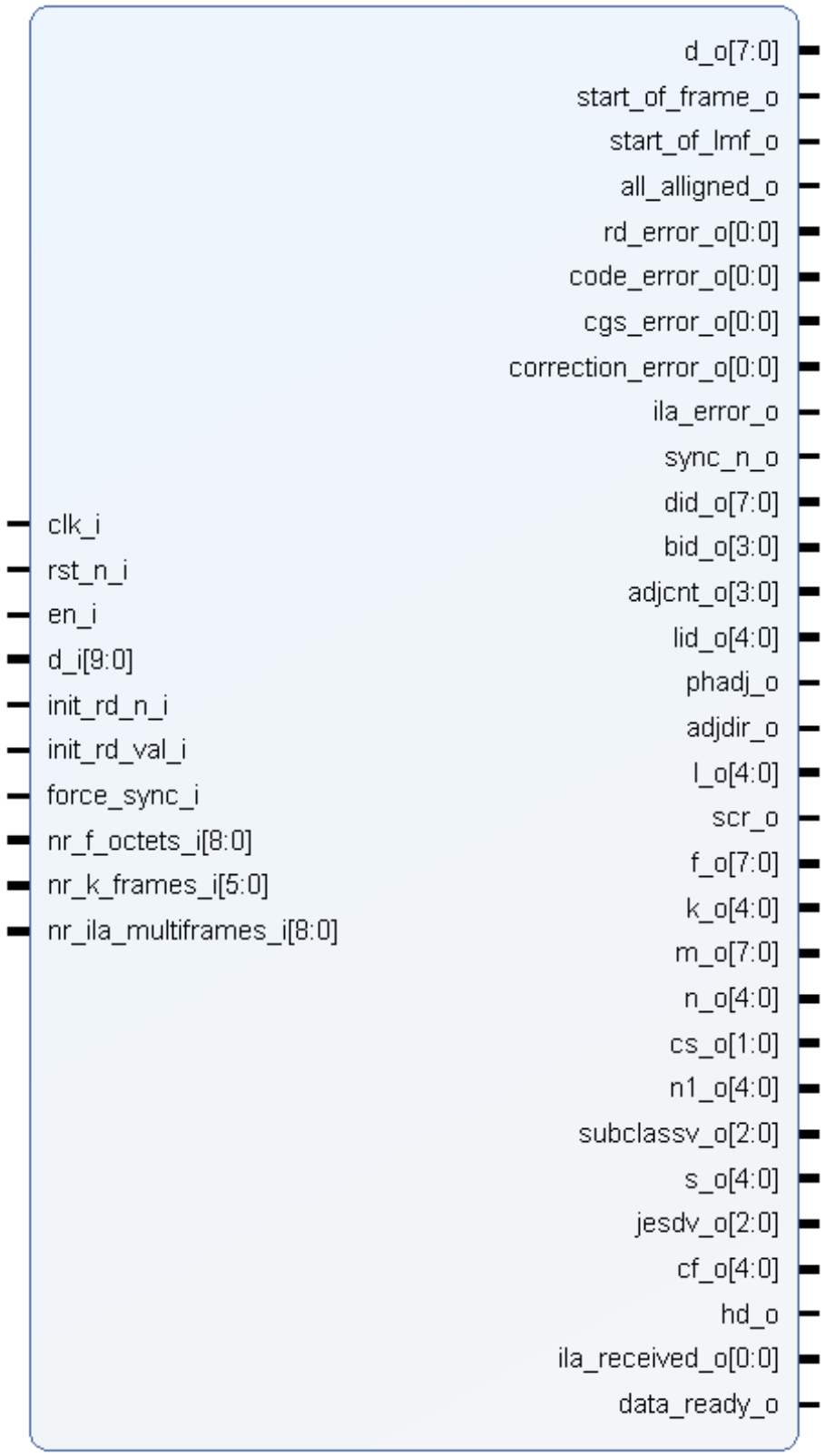


Figure 3.1: Graphical representation of designed JESD204B receiver for $NR\_OF\_LANES = 1$

Ports	I / O	Description
clk_i	Input	Clock input
rst_n_i	Input	Reset signal, active low, asynchronously asserted, synchronously deasserted
en_i	Input	Enable signal
*d_i[(L*10)-1:0]*	Input	Input serialized 10-bit data
init_rd_n_i	Input	Running disparity initialization request, active low
init_rd_val_i	Input	Initial value of running disparity
force_sync_i	Input	Input for issuing force resynchronization
nr_f_octets_i	Input	No. of octets per frame
nr_k_frames_i	Input	No. of frames per multiframe
nr_ila_multiframes_i	Input	No. of ILAS multiframes
*d_o[(L*8)-1:0]*	Output	Output 8-bit data
start_of_frame_o	Output	Start of frame indicator, active high
start_of_lmf_o	Output	Start of multiframe indicator, active high
all_alligned_o	Output	Indicator that all lanes are alligned, active high
rd_error_o[L-1:0]	Output	Running disparity error output, specific for each lane, active high
code_error_o[L-1:0]	Output	Code error output, specific for each lane, active high
cgs_error_o[L-1:0]	Output	Indicator of non-synchronized code group synchronization, active high
ila_error_o	Output	Parameters detected during ILAS are not equal on all lanes, active high
correction_error_o[L-1:0]	Output	Indicator of overflowed correction buffer, active high
sync_n_o	Output	SYNC~interface to transceiver, active low
did_o - hd_o	Output	Link configuration data from ILAS, see 2.8
ila_received_o[L-1:0]	Output	Indicator of received ILAS on each lane, active high
data_ready_o	Output	Indicator of valid data on output
L = NUMBER_OF_LANES		

Table 3.2: Ports description for the JESD204B receiver

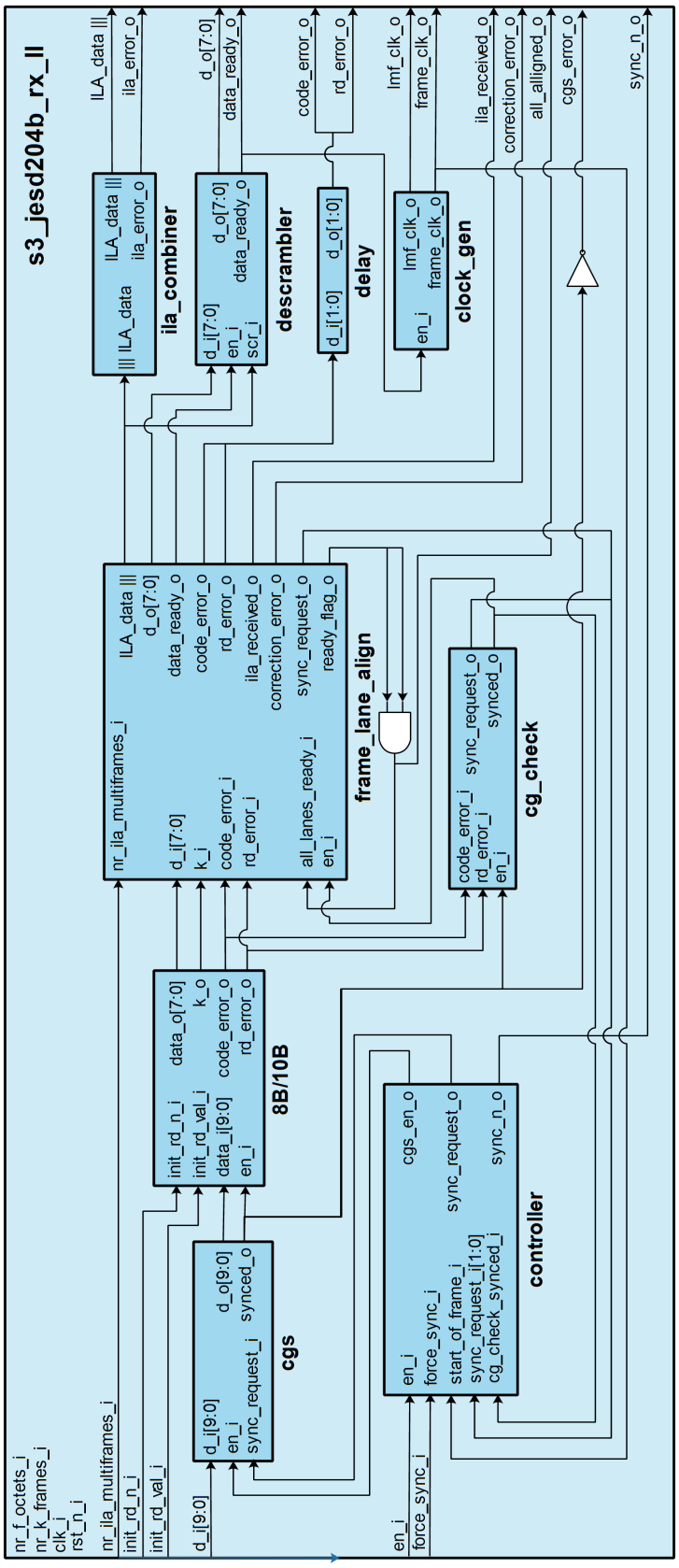


Figure 3.2: Simplified graphical representation of internal structure

Table 3.3: Parameters of designed block

Parameter	Value	Description
NR_OF_LANES	1 – 32	No. of lanes per converter device (link)
SYNC_DURATION_BUFFER_WIDTH	ceil(log2((*nr_f_octets_i* * 5) + 10))	Width of buffer determining SYNC∼ signal duration
INITIAL_BUFFER_WIDTH	(*nr_ila_multiframes_i* - 1) * *nr_k_frmaes_i* * *nr_f_octets_i* or lower	Width of buffer for ILAS
INITIAL_POINTER_WIDTH	ceil(log2(*INITIAL_BUFFER_WIDTH*))	Pointer for buffer above
MONITORING_BUFFER_WIDTH	5 * *nr_f_octets_i*	Width of buffer used for monitoring and correction
MONITORING_POINTER_WIDTH	ceil(log2(*MONITORING_BUFFER_WIDTH* - *nr_f_octets_i*))	Pointer for buffer above

3.2.2 Designed Submodules

The aim of this subsection is to introduce the designed modules inside the whole receiver device. Not all of them are covered, only the most essential ones.

Module *cgs*

The module called *cgs* is responsible for the synchronization described in the section 2.7.1, except the check phase, for which is responsible the module described in the following section 3.2.2. The *cgs* module is displayed in the figure 3.3. Input and output ports are discussed in the table 3.4.

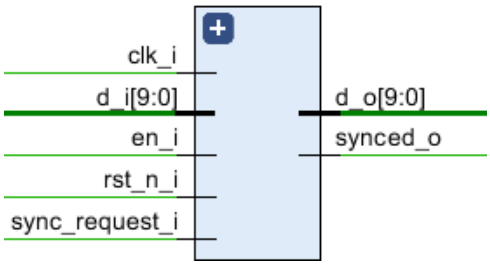


Figure 3.3: Graphical representation of the module *cgs*

Ports	I / O	Description
clk_i	Input	Clock input
d_i[9:0]	Input	Input serialized 10-bit data
en_i	Input	Enable signal
rst_n_i	Input	Reset signal, active low
sync_request_i	Input	Synchronization request input
d_o[9:0]	Output	Synchronized 10-bit data
synced_o	Output	Indicator of valid data on the output

Table 3.4: Ports description for the *cgs* module

It is basically a Mealy four-state machine, to which a few other circuits are connected. Transition conditions are illustrated in the figure 3.4. Initially, incoming 10-bit data are read into a 20-bit shift register. The processing of this data then takes place within the state machine itself. The first state is IDLE, when all values are reset and the state FIRTS_K is entered. It remains in this state until a control character /K/ is detected in the 20-bit word, with either a positive or negative running disparity. /K/ characters are detected in 10 parallel comparator pairs. The number 10 is used because of 10 possible positions of /K/ within the 20-bit word. Comparators are in pairs because there are two options, how the /K/ can be represented, according to the running disparity. If the /K/ is detected, its position is memorized, its representation is memorized, the counter is incremented and the OTHER_K

state is entered. Here, the arrival of other /K/ characters at the same position in the respective running disparity is assumed. If four /K/ characters come in a row in this way, ie the counter acquires this value, it moves to the next state, the SYNCED state. However, if the /K/ arrives at another position or does not arrive at all, the machine returns to the IDLE state. In the SYNCED state, a 10-bit portion of data is connected to the detected position from the 20-bit input register with data to output, and the *synced_o* output is also activated. From all the named states it is also possible to switch to the IDLE state if the *sync_request_i* input is activated.

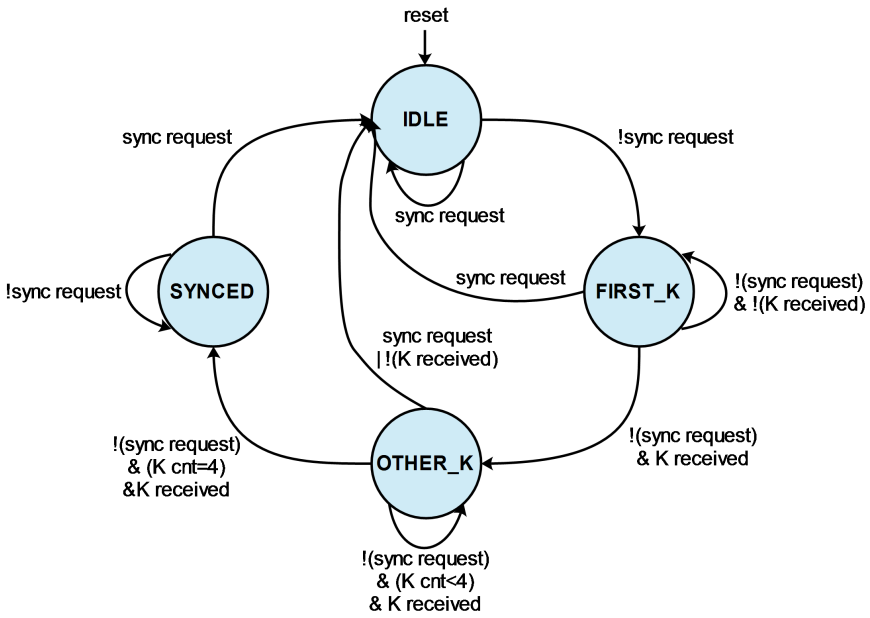


Figure 3.4: State machine implemented in module *cgs*

Module *s3_8b10b_decoder*

The next module connected to the *cgs* is the *s3_8b10b_decoder*, see the figure 3.5, and as the name suggest, it decodes the 10-bit encoded and synchronized data (characters) from the transmitter to the 8-bit octets. The process is described in further details in the section 2.6.5. Ports of the module are listed in the following table 3.5. The module was provided by the Dialog Semicondutor company.

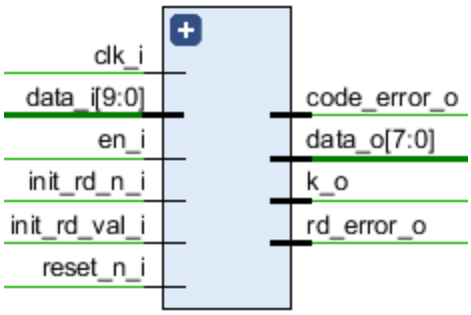


Figure 3.5: Graphical representation of the module *s3_8b10b_decoder*

Ports	I / O	Description
clk_i	Input	Clock input
data_i[9:0]	Input	Input 10-bit data
en_i	Input	Enable signal
init_rd_n_i	Input	Running disparity init. request, active low
init_rd_val_i	Input	Initial value of running disparity
reset_n_i	Input	Reset signal, active low
code_error_o	Output	Code error detected
data_o	Output	Output decoded 8-bit data
k_o	Output	Control symbol detected
rd_error_o	Output	Running disparity error detected

Table 3.5: Ports description for the *s3_8b10b_decoder* module

Module *cg_check*

The module *cg_check* is superstructure of the above mentioned *cgs* and uses error reports from the decoder. This module is constantly checking whether the transmitting octets are valid or not. The validity is recognized by means of values of the error signals from the decoder. The whole behaviour is explained above in the section 2.7.1. Illustration of the module is in the figure 3.6 and meanings of the ports are described in the table 3.6.

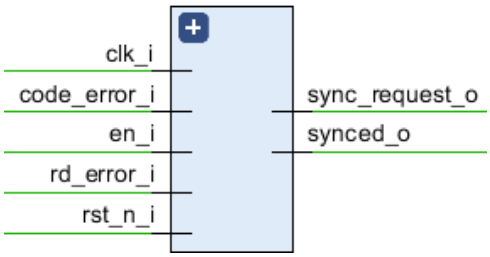


Figure 3.6: Graphical representation of the module *cg_check*

This module is another Mealy state machine, this time it is a five-state concept, which is supplemented by two more counters for counting valid and invalid symbols. Transition conditions are illustrated in the figure 3.7. From the de-

Ports	I / O	Description
clk_i	Input	Clock input
code_error_i	Input	Code error detected
en_i	Input	Enable signal
rd_error_i	Input	Running disparity error detected
rst_n_i	Input	Reset signal, active low
sync_request_o	Output	Request from frame synchronizaton
synced_o	Output	Indicator of synchronization

Table 3.6: Ports description for the *cg_check* module

fault IDLE state and after resetting the values, the machine switches to the SYNCING state. Here, based on the inputs *code_error_i* and *rd_error_i* signalling the validity of the data, the counters are incremented or the counter of valid symbols is reset. It happens whenever there is an interruption detected in a stream of valid symbols. If the invalid symbol counter reaches three, the machine enters the SYNC_REQUEST state. If the valid symbol counter reaches four, the machine enters the SYNCED state. Otherwise, the machine remains in the SYNCING state. The SYNC_REQUEST state resets the counters, activates the *sync_request_o* output, and then the machine enters the IDLE state. The SYNCED state also resets the counters, but activates the *synced_o* output. If an invalid symbol is detected, it is switched to the SYNCING2 state. Its behaviour is analogous to the SYNCING state, but maintains the *synced_o* output in logical one.

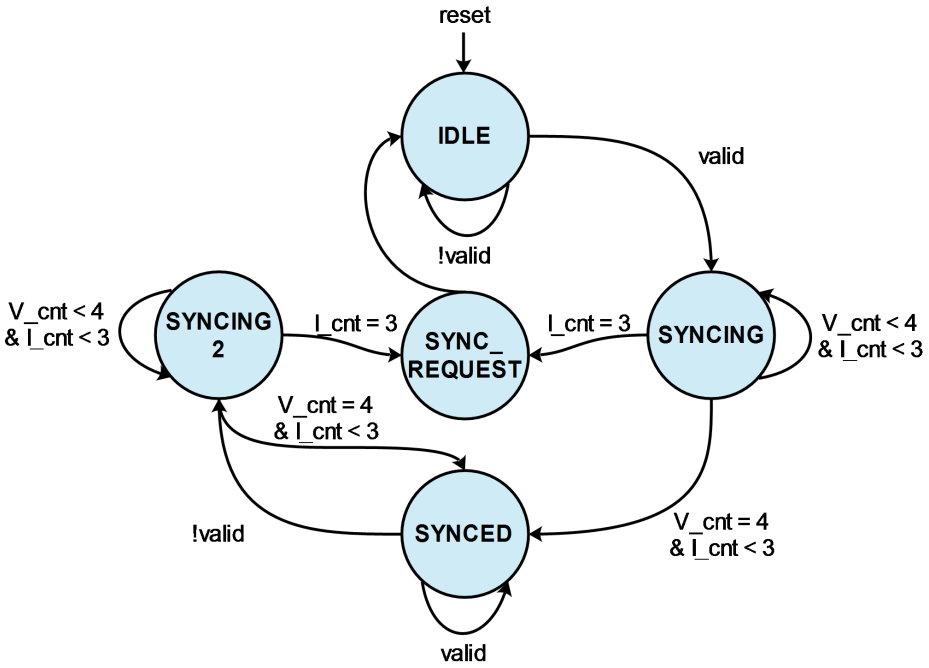


Figure 3.7: State machine implemented in module *cg_chceck*

Module *clock_gen*

The module in the figure 3.8 is responsible for the generation of the frame and local multiframe clock. The input ports are *clk_i* as an input clock, *en_i* as an enable signal and *rst_n_i* as a reset signal. Also inputs indicating number of octets per frame and frames per multiframe are connected. Outputs are *frame_clk_o* and *lmf_clk_o* for frame a and local multiframe clock. The expression "clock" is used in accordance to the standard, but in this design it is meant as a start of frame or local multiframe.

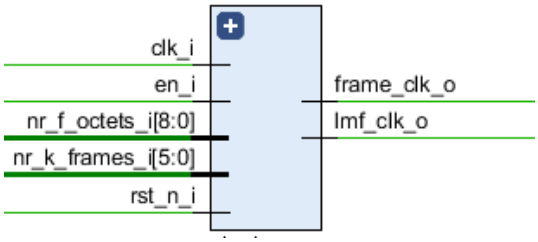


Figure 3.8: Graphical representation of the module *clock_gen*

The module is further divided into two more submodules. One is responsible for generating the frame clock and the other one for the local multiframe clock. Both are clock frequency dividers. The division ratio for the frame clock is determined by the input *nr_f_octets_i*. The input *nr_k_frames_i* is then added for the local multiframe clock. The outputs of these blocks are then connected to the respective outputs of the entire module.

Module *frame_lane_align*

The largest module in the design is named *frame_lane_align* and its representation is shown in the figure 3.9. This module covers all the functionalities presented in the sections 2.7.3 and 2.7.4. It is also recognizing the link configuration data from the ILAS and checking, if there are four consecutive /K/ control characters in data passing through the block, which indicates a lost of the synchronization on other links. If so, the synchronization request is asserted. The details about each port of the module is containing the table 3.7.

The main path, which the data passes through this block, is shown in the figure 3.10. The input data are at first stored in the initial buffer. Its size is set by means of the *INITIAL_BUFFER_WIDTH* parameter. The start of buffering occurs when the block receives the first /R/ symbol. At that point, it will also let other lanes know about this situation via the *ready_flag_o* output port. As the data octets are gradually loaded into the registers, the counter of the current buffer depth is incremented, whose width is set by means of the *INITIAL_POINTER_WIDTH* parameter. This process continues until the symbol /R/ is present on all lanes. Each block is informed about this via the input signal *all_lanes_ready_i*. Here, the count-

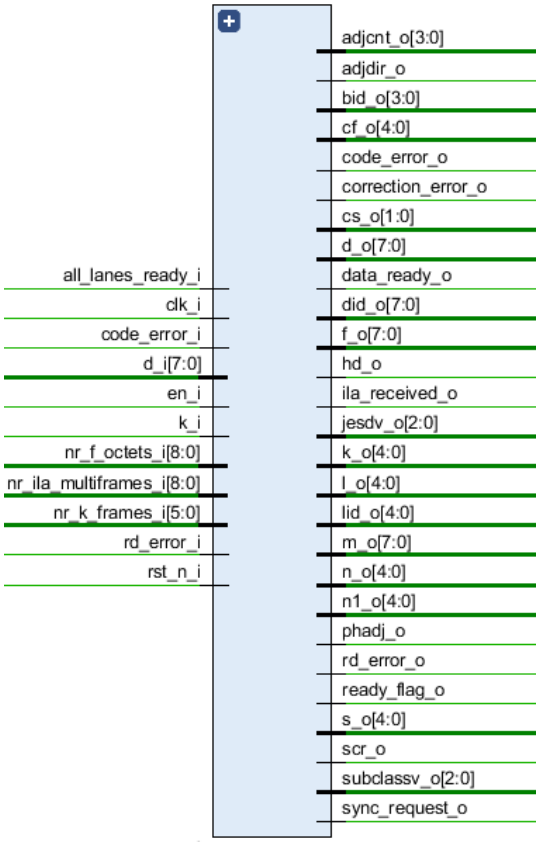


Figure 3.9: Graphical representation of the module *frame_lane_align*

ing is stopped, and the data are then released from the appropriate buffer location. In the same way as data octets, information from the inputs *k_i*, *code_error_i* and *rd_error_i* are also buffered, in other words information whether the given octet is a control character and whether it is valid according to the 8B/10B decoder.

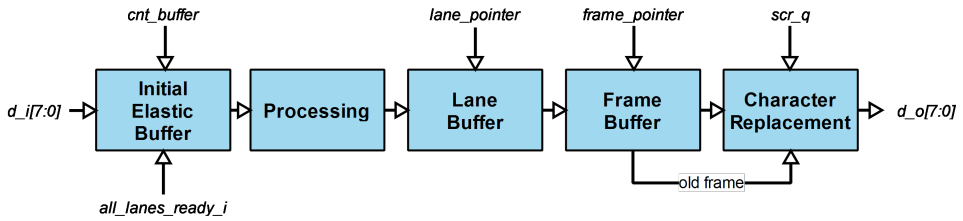


Figure 3.10: Illustration of data flow in the *frame_lane_align* module

This is followed by the main part of the processing of the received data. Several parallel processes run here simultaneously. First of them is a process for detecting control characters. A number of functions are implemented for comparing data with the values of control characters. On the basis of these functions, indications about the present characters are written to the

Ports	I / O	Description
all_lanes_ready_i	Input	Signalling of receiveing the first /R/ symbol on all lanes
clk_i	Input	Clock input
code_error_i	Input	Code error detected
d_i[7:0]	Input	Input 8-bit data
en_i	Input	Enable signal
k_i	Input	Control symbol detected
rd_error_i	Input	Running disparity error detected
rst_n_i	Input	Reset signal, active low
nr_f_octets_i	Input	No. of octets per frame
nr_k_frames_i	Input	No. of frames per multiframe
nr_ila_multiframes_i	Input	No. of ILAS multiframes
adjcnt_o - s_o	Output	Link configuration data from ILAS, see 2.8
code_error_o	Output	Code error detected aligned with data output
d_o[7:0]	Output	Output 8-bit data
data_ready_o	Output	Indicator of valid data output
ila_received_o	Output	Indicator of valid values on link configuration data outputs
rd_error_o	Output	Running disparity error detected aligned with data output
ready_flag_o	Output	Indicator that the current link received the firt /R/ symbol
sync_request_o	Output	Synchronization request asserted after three invalid characters are detected

Table 3.7: Ports description for the *frame_lane_align* module

appropriate registers named *X_spotted*, where *X* represents the letter *A*, *F*, *Q* or *R* according to the names of the control characters. The values of these registers then used in further processing.

These are exclusively within ILAS two processes. The first one is the /A/ characters counter, which is incremented up to the value corresponding to the *NR_ILA_MULTIFRAMES* parameter. Its value is used to determine the end of the ILAS. Then there is a process for obtaining ILA configuration data. This is started by accepting the /Q/ character, which also starts a counter that identifies the current position within the sequence of configuration data and is incremented with the reception of a new octet. Configuration data are stored and outputted outside the block. After the process is completed, the reading is stopped and by passing the value on the output port *ila_received_o* to the logical one, it is confirmed that the configuration data on the output are valid and the sequence has terminated.

Other processes take place continuously until a possible resynchronization between the receiver and the transmitter happens. It means mainly those that lead to the dynamic data alignment correction. The previous position of the /A/ and /F/ characters, cross coupling between frame and multiframe alignment and also incrementing and aligning octet counters and frames with data are issued there. All these linked processes are described in the sections 2.7.3 and 2.7.4. Then they lead to the setting of values of frame and lane pointers important for the processes described below in this section.

The last process that runs at this level is the detection of /K/ characters in the data stream. This is one way for the receiver to recognize that another receiver has requested a resynchronization with the transmitter. If the block detects, by means of counter, that it has received four consecutive /K/ characters, then it decides that synchronization has been lost and the synchronization request is issued using the *sync_request_o* output.

In the next phase, the data are buffered into two buffers. One is used for the lane synchronization monitoring and correction and the other one for the frame synchronization monitoring and correction. Their widths are set according to the value of the *MONITORING_BUFFER_WIDTH* parameter. The width of the respective pointers used for reading the correct data from buffers is then set based on the value of the *MONITORING_POINTER_WIDTH* parameter. Firstly, the data are stored in *lane_buffer*, from where are read from the position specified by the value of the *lane_pointer* to the *frame_buffer*. Secondly, the data are similarly stored in the *frame_buffer* and read from here according to the value of the *frame_pointer* for a further processing. If the maximum or minimum value of one of the pointers is reached, this fact is reported via the *correction_error_o* output and the synchronization request is issued via *sync_request_o*. In the same way as data octets, information, whether the given octet is a control character and whether it is valid according to the 8B/10B decoder, are also buffered here.

Finally, there is another process running and it is responsible for the character replacement of the outputting data. Data from the *frame_buffer* are at this point sent to the output *d_o* of the block. However, if there is a control character /A/ or /F/ present in the data, it is replaced on the output by a specific value or an octet value that appeared at the same place in the last frame. Switching between the replacement with the value or with the old octet is done on the basis of enabling or disabling the scrambling. The values from the *frame_buffer* at the position given by the *frame_pointer* are always read to the outputs *code_error_o* and *rd_error_o*. More about this operation is described in sections 2.7.3 and 2.7.3.

Module *ila_combiner*

The module *ila_combiner* is a logic device used for comparing the link configuration data received from the previous module described above in the section 3.2.2 from each link and combining them together. If it detects any differences in the input data, it asserts the *ila_error_o* output. The clock input *clk_i* is used for clocking flip-flops on the output.

It is based on particular smaller combinational circuits dedicated for each parameter. Each of them can be parametrized by the number of lanes and the width of the input data. In the combination logic, it compares the input data with each other to see if they differ from each other. The assumption is that the configuration parameters came the same from all lanes during ILAS. If a difference is detected, it is signalled on the output. The outputs of the individual blocks are combined by the OR function into one output. At the output of the entire block, the ILA configuration parameters are already in their basic bit width.

Module *descrambler*

Descrambler is an optional module, whose functionality depends on the link configuration. In further details, the principle of scrambling is presented in the section 2.6.4. The visualisation of the module is shown in the figure 3.11 and the ports description is in the table 3.8.

The module consists of 16 XOR gates, 24 D flip-flops and 4 multiplexers. There are two registers for storing data older by one and two time cycles. The oldest data is then combined (descrambled) with the current data using XOR gates to create the correct output data. Of course, this only happens when scrambling is enabled. If scrambling is not enabled in the configuration data, then the input data passes through the register directly to the output. Furthermore, the data validity signals also passes through the flip-flop to ensure its synchronicity with the data.

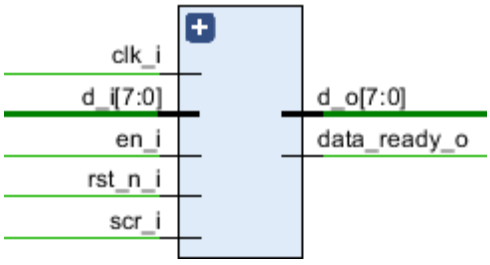


Figure 3.11: Graphical representation of the module *descrambler*

Ports	I / O	Description
clk_i	Input	Clock input
d_i[7:0]	Input	Input 8-bit data
en_i	Input	Enable signal
rst_n_i	Input	Reset signal, active low
scr_i	Input	Scrambling enable
d_o[7:0]	Output	Output 8-bit data
data_ready_o	Output	Indicator of valid data output

Table 3.8: Ports description for the *descrambler* module

Module *controller*

The last important module, displayed in the figure 3.12, used inside the receiver device is called *controller*. Mainly the module takes care of the SYNC~ interface between the JESD204 transmitter and receiver. Other functions are distributing the synchronization requests and enabling the code group synchronization. The design uses the SYNC~ signal combining inside the receiver device, for more information see the section 2.7.2. The description of the ports is shown in the table 3.9.

There are several things going on inside this module, but all are related with the SYNC~ interface. The synchronization requests are here combined and distributed further via an output port to the *cgs* module. The duration of the synchronization request for the transmitter device is ((*NR_F_OCTETS**5) + 10) octets and the width of the counter used for measuring is defined by the parameter *SYNC_DURATION_BUFFER_WIDTH*. The duration is also decisive for the enable signal of the *cgs* module and the enable signal is distributed via the output port *cgs_en_o*.

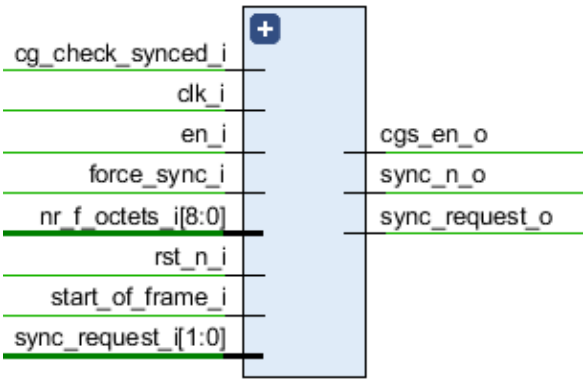


Figure 3.12: Graphical representation of the module *controller*

Ports	I / O	Description
cg_check_synced_i	Input	Indication of *cg_check* synchronization
clk_i	Input	Clock input
en_i	Input	Enable signal
force_sync_i	Input	Input for issuing force resynchronization
rst_n_i	Input	Reset signal, active low
start_of_frame_i	Input	Indicator of start of the frame
sync_request_i[1:0]	Input	Requests for resynchronization
nr_f_octets_i	Input	No. of octets per frame
cgs_en_o	Output	Enable signal for *cgs* module
sync_n_o	Output	SYNC~ interface, active low
sync_request_o	Output	Combined synchronization request signal

Table 3.9: Ports description for the *controller* module

3.3 Verification

An integral part of every digital design is also its verification. It was no different in the case of this thesis. The verification also takes place during development, when the individual components are subjected to simulations and tests. This is followed by a more complex simulation of the whole designed circuit. Finally, it is a good practise to verify the design in a hardware, in this case in a FPGA, before it will be fully deployed in a production of an ASIC. In the case of this thesis, the designed receiver was tested only by simulations and the validation on the FPGA did not occur due to a time pressure, although the most of the circuits needed for this were prepared.

Simulations were run in multiple tools. The first tool was Vivado, where only the reference Xilinx JESD204 IP core was instantiated and was used to generate a serial data stream, which was captured into a text file. This file was then used for simulations in the second tool, ModelSim. There the designed submodules and in the end the whole block were simulated. The cases that could be simulated in ModelSim were limited and an additional use of Vivado was necessary for more complex testing, when there were, for example, repeated synchronizations of the designed receiver. Lastly, Cadence Xcelium came into an account, because of its performance in simulating larger designs with Xilinx IP cores. As above, the presented results are for a one lane configuration in order to make the description clearer.

3.3.1 Simulation of Xilinx JESD204 TX IP in Vivado

The first simulation, realized in Vivado, was just about simulating the Xilinx JESD204 IP core as a reference transmitter device and recording the transmitted data bits into a text file. The simplified idea scheme of the simulation is in the figure 3.13. The Xilinx JESD204 IP core is in more detail described further in this subsection. Most important in that case was to supply the IP

core with clocks, user data, reset and configure it via the AXI4-Lite interface. There are several link configuration registers needed to be set to the desired values. It was done by tasks included in the testbench. Once the link configuration was defined, the core needed to be reset via the AXI4-Lite interface. For each lane there is a 32-bit wide space for user data. After the propagation of /K/ control characters started, the data stream began to be recorded and the simulation waited for a while until it released the synchronization request, so the ILAS and then user data started to transmit. After a moment, the simulation finished and the text file was closed and saved. In that way, several scenarios were saved based on the configuration of the IP core and user data. Also only a stream of /K/ characters was recorded, which was essential for the first simulations in ModelSim.

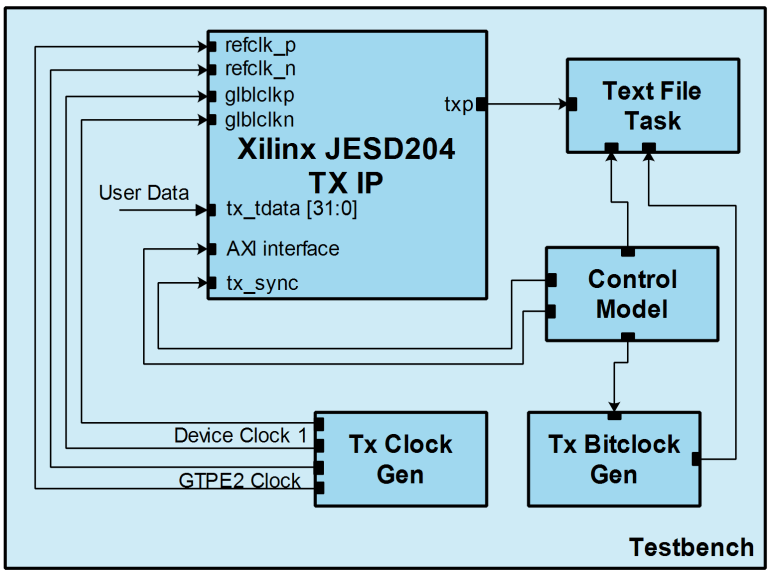


Figure 3.13: Idea scheme of simulation

Reference Xilinx JESD204 IP Core

The Xilinx JESD204 IP core implements a JESD204B interface supporting line rates from 1 Gb/s to 12.5 Gb/s. The maximum data rate can be limited by a particular FPGA device. Inside the core, there is also included a multigigabit transceiver of type GTX, GTH, GTP or GTY depending on a FPGA. The number of lanes per core is ranging between 1 to 8. The number can be also further limited by the type of the FPGA. The user interface is guaranteed by the AXI4-Lite bus. The JESD204 core can be configured as a transmitter or receiver.

For FPGA *Artix®-7 XC7A200T* the maximum data rate is limited to 3.75 Gb/s. Supported transceiver is GTP and the maximum number of lanes per core is 4. [12] The figure 3.14 shows an overview block diagram for the transmitter of the JESD204 core.

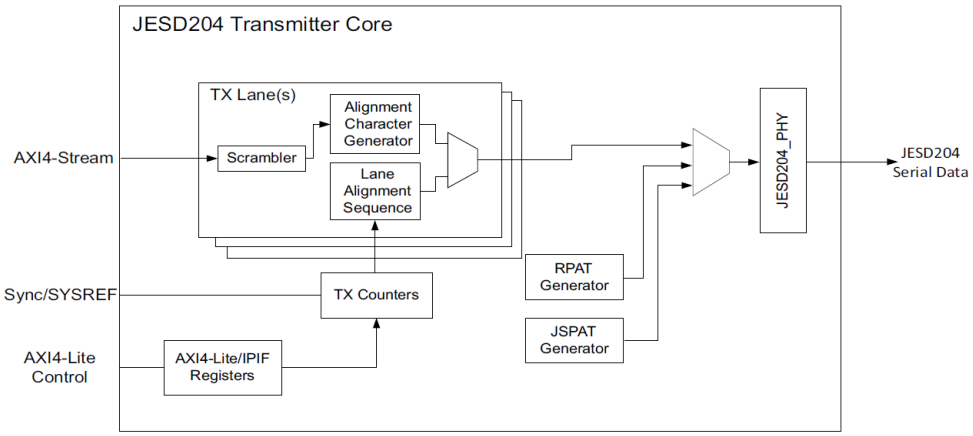


Figure 3.14: Transmitter core overview [12]

3.3.2 Simulation of DUT

Data streams saved in text files from the previous simulations were used during this step. Serial bits from files were read by means of a task and distributed to the first block. It was a model, which was simulating a delay. This model could delay lanes by a number of bits between each or the delay could get longer during the simulation. The delay can occur on a lane for instance because of a heating of the chip during an operation or a different length of a routing on PCB. The delay amount was defined by means of the input signal *stream_delay*. Data were then propagating to the second block, which was a deserializer. This model represents a shift register with its own clock reference inside. Every ten clock cycles it outputs a 10-bit deserialized data and also the clock. The last model before the DUT (Device Under Test) was another delay, this time a lane delay. This model automatically delays lanes between each other by a multiple of ten bits. After that, data finally reached the receiver (DUT). The clock for DUT was supplied by the deserializer. The whole testbench is illustrated in the figure 3.15. Not all ports of the receiver and other blocks are listed in case of more readable picture.

Test Cases

At the beginning, the whole receiver was not designed, so the blocks inside it were simulated gradually. Until the code for ILAS was developed, the task was reading the stream of /K/ control characters and the behaviour of previously designed submodules was visually observed. After the implementation of a code for the ILA configuration data detection, some other files could be included. These files have included a synchronization part consisting of /K/ characters, ILAS and also a user data portion. A big milestone was the implementation of the *frame_lane_align* submodule, when the outputting data were these user data, which were on the input of the Xilinx JESD204 IP core in the previous subsection 3.3.1, if the data were not scrambled. From

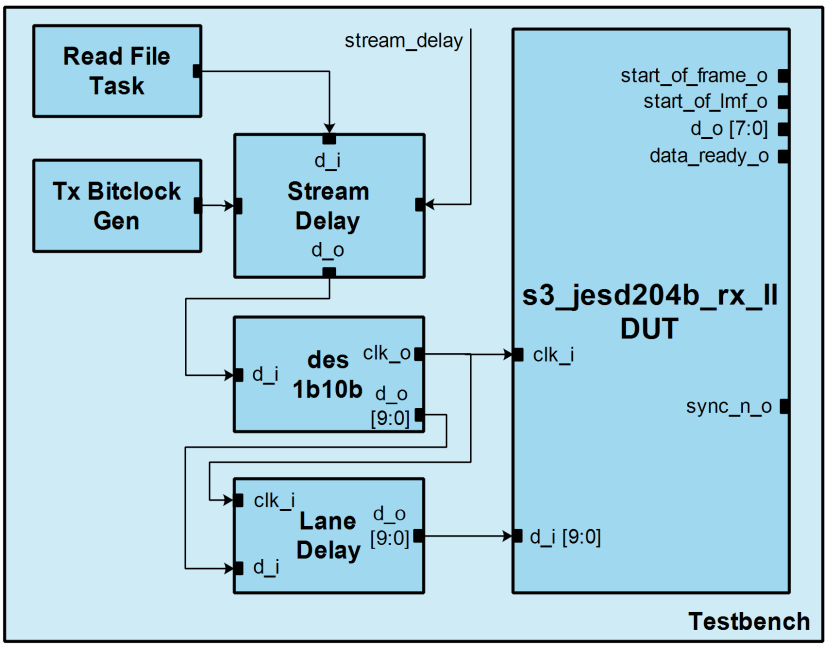


Figure 3.15: Idea scheme of simulation

that point, the outputting data were controlled automatically by means of a model of a comparator. After the rest of submodules were added to the design, some more simple and advanced test cases could be used. The tests were run in ModelSim during the development process and in the end, when the whole module was designed, also in Cadence Xcelium, where a code coverage was examined.

Simple Test Cases. Several simple test cases were issued. Basically they were all the same, recorded data into text files previously from the Xilinx JESD204 IP core. Differences were in the user data, number of octets per frame, number of frames per multiframe, scrambling and number of lanes. The simulation monitored these most crucial outputs:

- *cgs_error_o*,
- *sync_n_o*,
- *all_alligned_o*,
- *ila_received_o*,
- *data_ready_o*.

Without their activity, the simulation will end because of a timeout. After a defined time of the simulation, where the presence of correct user data were expected, the simulation automatically compared their correctness to the preset data sequence and decided, whether the test was passed successfully or not.

Advanced Test Cases. After the DUT have passed simple test cases successfully, some advanced options were tested. It started with delaying the data stream by means of *stream_delay* block while running the simulation. It led to the fact, that the *cgs* block started to send the wrong 10-bit word from this point of time and so the running disparity or code errors were detected by the 8B/10B decoder. As a consequence, the lost of the synchronization was detected and the synchronization request to the transmitter was triggered. In that case, the *sync_n_o* interface was monitored. Then, the text files were manually modified to simulate some non-standard situations. Some symbols were added or deleted from the data stream and the frame and lane monitoring and correction features were monitored. Whether the receiver corrects frame and lane pointers or whether they are not within their ranges and *correction_error_o* is asserted was monitored. Outputting data were similarly compared like previously at a well defined time. And still, the configuration of user data, F, K and scrambling was changing.

Cadence Xcelium Code Coverage

Test cases were also run in Cadence Xcelium and the code coverage of DUT was measured. The code coverage is a method of assessing how well the test cases have exercised the design. The results for each submodule are summarized in the figure 3.16. Meanings of tests are explained further in this subsection. Reasons of a low coverage in some cases are listed below:

- Only one transition for the reset signal was issued.
- The tests were not run with the reference Xilinx JESD204 IP core.
  - It indicates one transition for the SYNC~ and ILA parameters.
  - Alternatively some more edited test vectors would eliminate this.
- Large buffers for initial synchronization and monitoring buffers were used. (Applies to the *frame_lane_align* submodule)

Block Coverage. The block coverage identifies the lines of code that are executed during a simulation run. It helps to determine what areas of the DUT design are not exercised by the testbenches. A block is a statement or sequence of statements that executes with no branches or delays. [13]

Branch Coverage. The branch coverage provides more precise details than the block coverage by obtaining coverage results for various branches individually. With the branch coverage, a piece of code is considered 100% covered when each branch of a conditional statement has been executed at least once. [13]

Expression Coverage. The expression coverage measures how thoroughly the testbench exercises expressions in assignments and procedural control constructs (if/case conditions). It identifies each input condition that makes the expression true or false and whether that condition happened in the simulation. [13]

Toggle Coverage. The toggle coverage collects and reports the design signal toggle activity. Signal toggle is a binary transition (and return after a finite delay) of a DUT signal. [13]

FSM Coverage. The FSM (Finite State Machine) coverage measures how well FSMs are exercised. It reports, if all possible transitions are issued. [13]

Block Average	Block Covered	Branch Average	Branch Covered	Expression Average	name
88.89%	88.89% (40/45)	88.89%	88.89% (32/36)	66.67%	controller
94.37%	94.37% (67/71)	93.10%	93.10% (54/58)	83.33%	cgs[1]
97.17%	94.87% (111/117)	96.88%	94.00% (94/100)	n/a	s3_8b10b_decoder[1]
89.47%	89.47% (51/57)	88.00%	88.00% (44/50)	66.67%	cg_check[1]
95.53%	95.53% (299/313)	93.94%	93.94% (217/231)	83.61%	frame_lane_align[1]
84.17%	83.90% (99/118)	52.50%	52.50% (21/40)	n/a	ila_combiner
100.00%	100.00% (13/13)	100.00%	100.00% (10/10)	n/a	descrambler[1]
100.00%	100.00% (28/28)	100.00%	100.00% (24/24)	n/a	clock_gen
100.00%	100.00% (4/4)	100.00%	100.00% (2/2)	n/a	error_delay[1]

Expression Covered	Toggle Average	Toggle Covered	Fsm Average	Fsm Covered	name
66.67% (4/6)	52.63%	52.63% (10/19/9)	n/a	n/a	controller
83.33% (10/12)	88.75%	88.75% (71/80)	91.67%	90.00% (9/10)	cgs[1]
n/a	92.59%	92.59% (25/27/1)	n/a	n/a	s3_8b10b_decoder[1]
66.67% (20/30)	96.15%	96.15% (25/26)	92.86%	91.67% (11/12)	cg_check[1]
83.61% (51/61)	60.24%	60.24% (594/986/24)	n/a	n/a	frame_lane_align[1]
n/a	15.64%	18.86% (86/456)	n/a	n/a	ila_combiner
n/a	89.19%	89.19% (33/37)	n/a	n/a	descrambler[1]
n/a	66.48%	62.86% (22/35/39)	n/a	n/a	clock_gen
n/a	50.00%	50.00% (3/6)	n/a	n/a	error_delay[1]

Figure 3.16: Code coverage of particular submodules

3.3.3 Simulation of DUT with Xilinx IPs

The next step, which opened some new possibilities in testing the receiver, was a simulation with the usage of Xilinx IP cores. At first, only the Xilinx JESD204 IP core was used during simulations in Vivado. For simulations with Xilinx JESD204 IP core and GTP transceiver the tool with a greater performance was needed, so it was done in Cadence Xcelium. For both cases, the gearbox described below was used.

Gearbox

The gearbox in general is a module, which changes a bit width and a frequency of data words. This particular one consists of 3 submodules. The first one is a clock generator. It generates a gated clock with half of the input frequency of the receiver's device clock with the 25% duty cycle. This clock is important for reading from the second submodule, which is an asynchronous FIFO. The FIFO is used because of the crossing between two clock domains. The first domain is given by the deserializer and received data are written into the FIFO on its positive edge. The second domain is supplied by the gated clock described previously. However, on the output of the FIFO, there are still 20-bit wide data, which needs to be splitted into two 10-bit wide words with

Ports	I/O	Description
clear_i	Input	FIFO clear signal
d_i[19:0]	Input	Input data
write_en_i	Input	FIFO write enable
serdes_clk_i	Input	Clock signal from deserializer
read_en_i	Input	FIFO read enable
device_clk_i	Input	Clock for output data
rst_n_i	Input	Reste signal, active low
empty_o	Output	FIFO empty signal
full_o	Output	FIFO full signal
d_o[9:0]	Output	Output data

Table 3.10: Ports description for the *gearbox* module

a two times faster frequency. So the last submodule is a multiplexer (MUX), which is responsible for this operation. It reads data from the FIFO, split them into two words and outputs them one by one on the positive edge of receiver's device clock. The whole system of the *gearbox* is in the figure 3.17. The description of input and output ports of the *gearbox* is in the table 3.10.

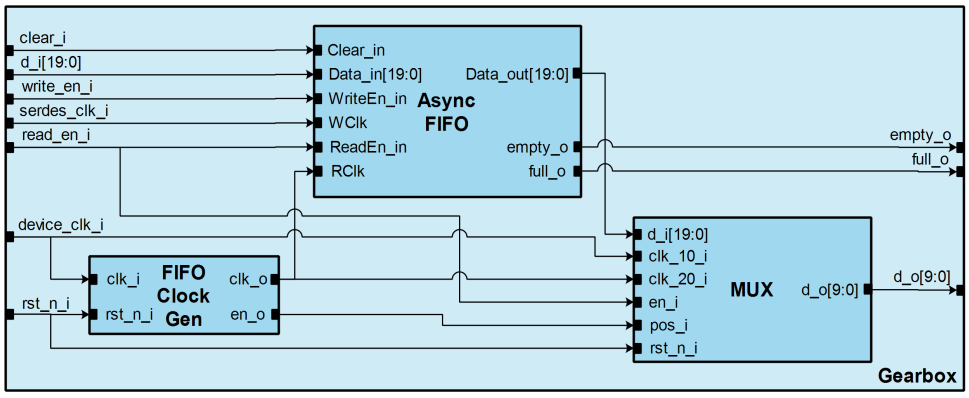


Figure 3.17: Graphical representation of *gearbox*

Simulation with Model of Deserializer

The simulation scheme, see the figure 3.18, was adjusted a little, so it was closer to a future implementation on a FPGA. This mainly means adding of the gearbox and updated deserializer. The bit width of the deserializer was updated in order to match the data width of the GTP transceiver. The gearbox had to be used, because the use of a multigigabit transceiver on a FPGA was intended and the transceiver had the 20-bit output data interface, while the receiver was designed for the 10-bit wide data input. The scheme is again only a simplified version of the real implementation and there are just the most crucial ports displayed. This kind of the simulation was run in Vivado.

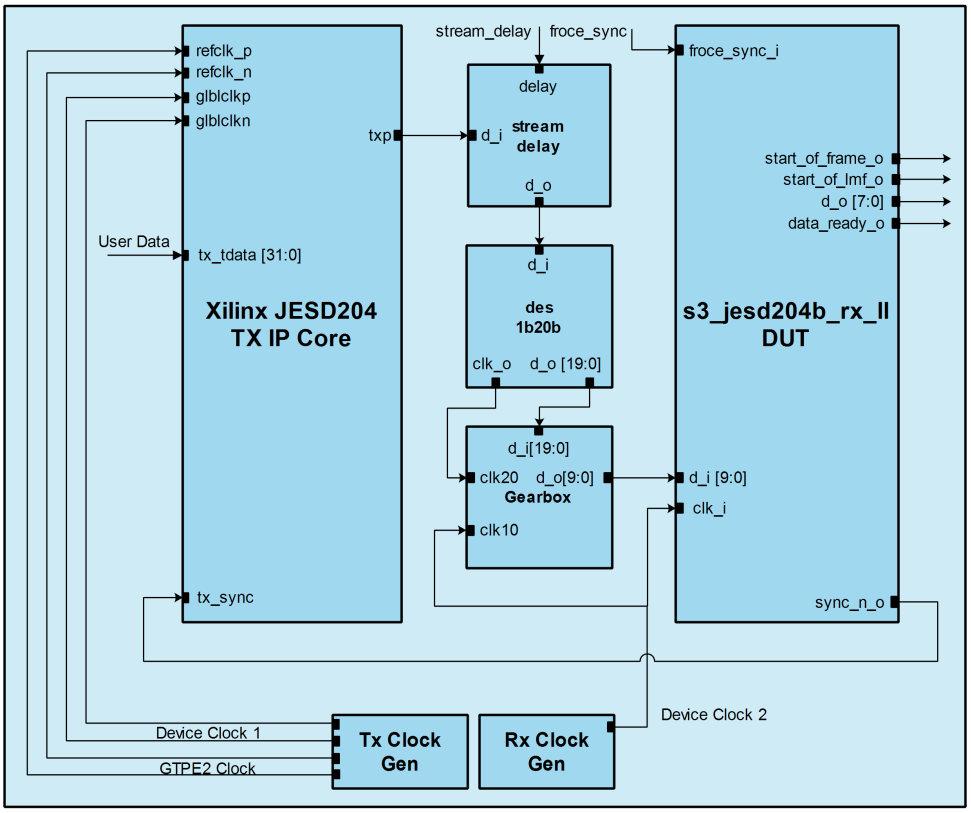


Figure 3.18: Idea scheme of simulation

It all starts with the reference JESD204 IP core delivered by Xilinx via Vivado IP catalogue. The output data are connected to the block named *stream_lane_delay*, described previously. The next part is a deserializer, in this case it is making a 20-bit wide output data bus from a continuous input data stream. Also similarly to the previous one, it is supplying the clock related with the output data. The subsequent block is a gearbox, whose purpose is to split the 20-bit wide data in two 10-bit words sent one by one via the output interface. The gearbox, which is in more details described below, ensures, that the output data are synchronized with the supplied device clock, which is also delivered to the last module, the designed JESD204B receiver. The receiver is there the DUT.

The main goal of this kind of a simulation was to test the functionality of the SYNC~ interface and issue multiple resynchronizations between devices. Synchronization requests were triggered either by delay inserted in the serial stream or by asserting a force synchronization via the *force_sync_i* input port. Like previously, the behaviour of output ports and the correctness of data were monitored at well defined moments.

Simulation with GTP Transceiver

The simulation taking place in Cadence Xcelium was supposed to be the last one before the implementation on the FPGA. The deserializer was replaced by the Xilinx GTP IP core. Because of it, the further simulations in Vivado were ineffective due to time demands. The transceiver, besides other connections, had to be supplied with a reference differential clock. Like it was said in the section 2.9, the transceiver had just the deserializing function. All of the others were bypassed. The output data and clock were then connected to the designed gearbox. The whole simplified simulation scheme is in the figure 3.19.

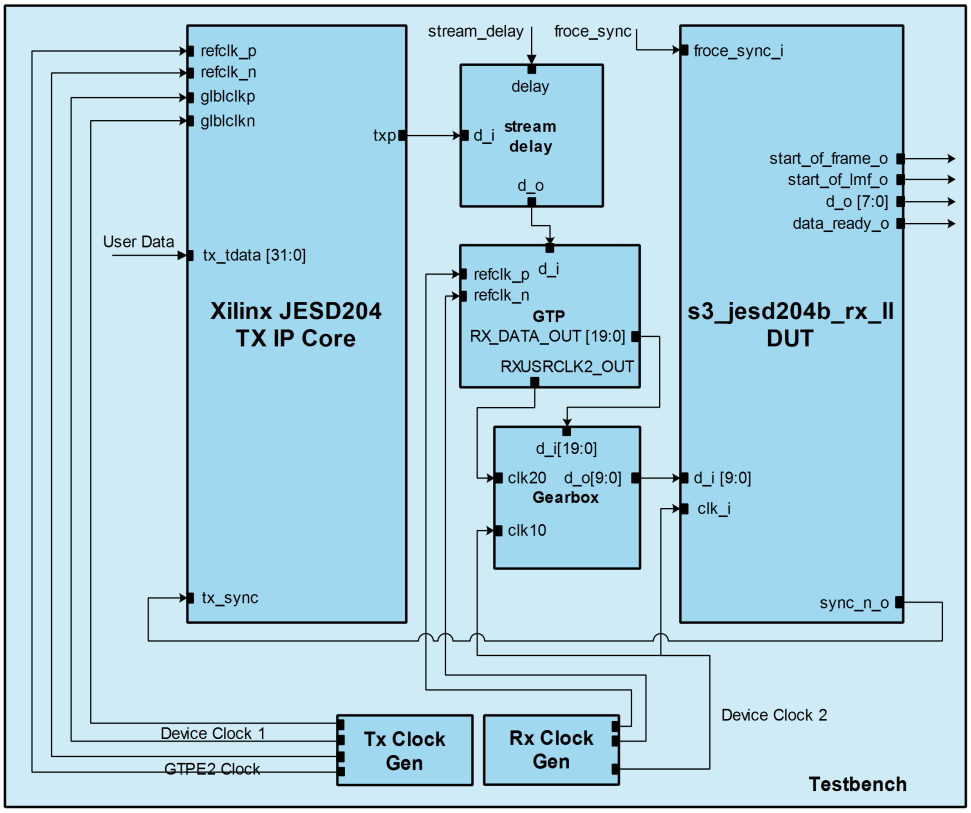


Figure 3.19: Idea scheme of simulation

The proper functionality of the GTP block was verified by means of the same tests as in the previous simulation. However for a physical implementation on a FPGA was no time left. So, this kind of simulation was the last step of the verification process done during the elaboration of this master thesis.

Example of Simulation

In the endof this subsection, there will be one test case of the simulation in Cadence Xcelium presented. In this case, the parameters of the simulation were as follows in the table 3.11.

Parameter	Value
No. of lanes L	1
No. of octets per frame F	4
No. of frames per multiframe K	16
Scrambling enabled SCR	0
No. of ILAS multiframes	4
Delay stream after 1st sync	1

Table 3.11: Parameters of the example simulation

In the figure 3.21 there is a screenshot of the waveform from the simulation. On the first row, there are user data on the input of the reference Xilinx JESD204 IP core. It can be seen that there are static data for the whole time of the simulation. Next two rows are differential serial outputs from the transmitter. Following three rows are inputs of the DUT and it can be seen that there are the same values as stated in the table 3.11. Parameters from the table can be compared with outputs of the block *l_o*, *f_o*, *k_o* and *scr_o*. The signal from the testbench *en_rx* is enabling the designed module to start working. Then it can be seen, when the code group synchronization was achieved, when the synchronization request to the transmitter was deasserted, when data were aligned in the initial buffer and when ILA configuration data were received. Coloured waves are indicating the reception of control characters. Finally, in the bottom it can be seen, when data are proceeded on the output of the block with corresponding indication of the start of frame and local multiframe.

After 1.067 ms of the simulation, the *stream_delay* signal raised up from 0 to 1. It led to a shift in the data, detection of running disparity or code errors and lost of the synchronization. Because of it, the synchronization request was asserted and the DUT was synchronized again and worked correctly until the simulation ended. Messages reported in the console are shown in the figure 3.20.

```
xcelium> run
** Timing checks are not valid (GTP) **
** Resetting the TX core... **
** Timing checks are valid (GTP) **
** Version = Major   7 Minor   2 Rev   8 **
** AXI Configuration and Reset complete.... (TX) **
** GT TX Reset Done **
** Enabled RX **
** CGS Synced **
** All Lanes Alligned **
** ILA Configuration Data Received **
** Start Sending User Data **
** Data present on Output **
** Test before resync completed successfully **
** Test after resync completed successfully **
** Test completed successfully **
Simulation stopped via $stop(1) at time 1072084800 PS + 1
xcelium>
```

Figure 3.20: Screenshot of messages in simulation console

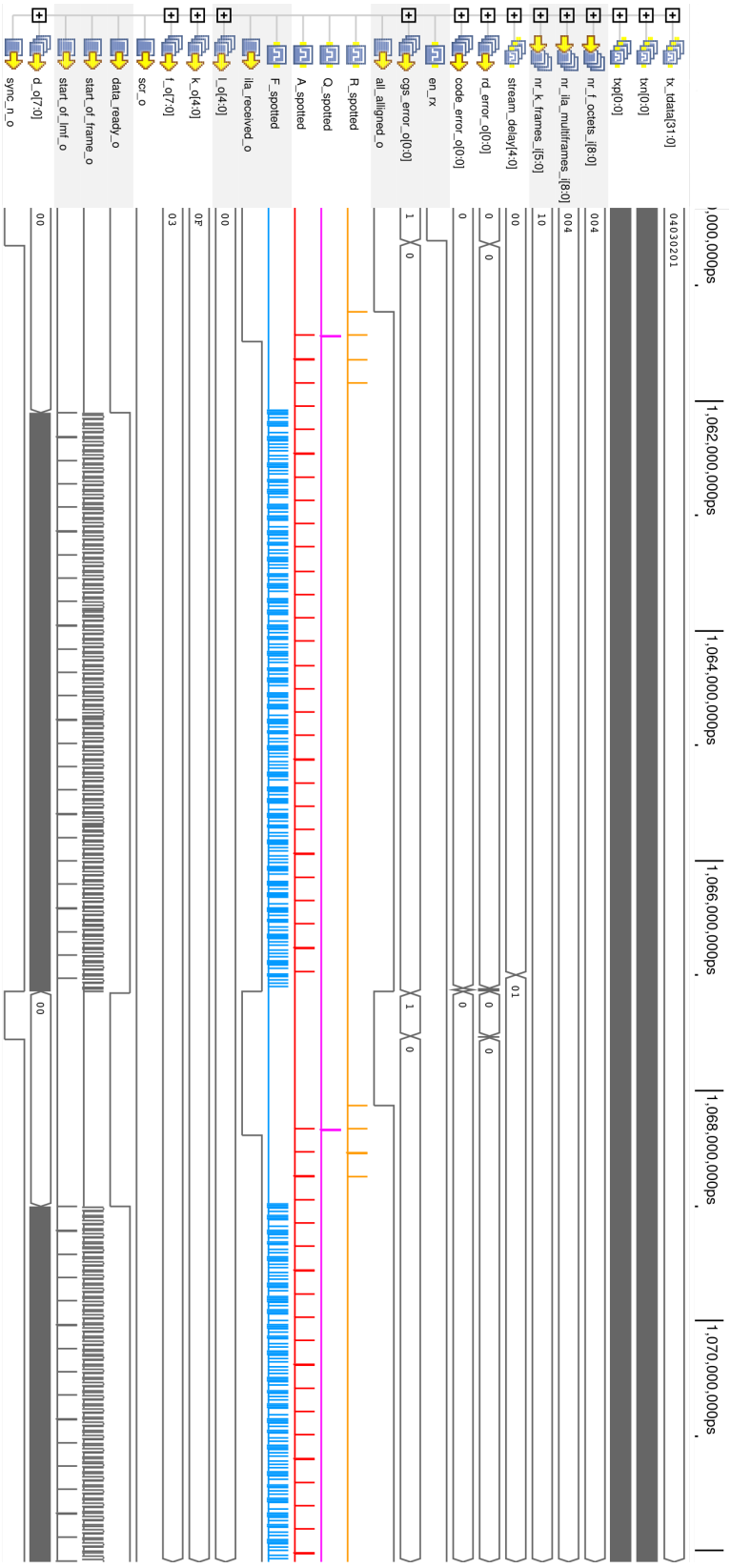


Figure 3.21: Screenshot of waveform

3.4 Synthesis

The designed module was synthesized for FPGA and ASIC use cases. Reports from these cases are presented in two subsections below. Parameters for both synthesis were as in the table 3.12.

Parameter	Value
Clock frequency [MHz]	156.25
NR_OF_LANES	1
SYNC_DURATION_BUFFER_WIDTH	5
INITIAL_BUFFER_WIDTH	192
INITIAL_POINTER_WIDTH	8
MONITORING_BUFFER_WIDTH	20
MONITORING_POINTER_WIDTH	3

Table 3.12: Parameters for synthesis

3.4.1 FPGA

Whether the designed receiver block is without any syntax mistakes and is synthesizable was verified by the synthesis tool in Vivado. The synthesis did not report any warnings and the post-synthesis utilization for the module supplied with the device clock equal to 156.25 MHz on the *xc7a200tsbv484-1* FPGA is shown in the table 3.13.

Resource	Utilization	Available	Utilization [%]
LUT	876	133800	0.65
FF	684	267600	0.25
IO	138	285	48.42
BUFG	1	32	3.13
LUT...Lookup Table *FF...Flifp-Flop* *IO...Input / Output Pins, only if whole FPGA is designed receiver* *BUFG...Global Buffer*			

Table 3.13: Post-Synthesis utilization of designed JESD204B receiver for FPGA

3.4.2 ASIC

The design with one lane configuration was also synthesized by the Synopsys Design Compiler. This software was provided by the Dialog Semiconductor company. The synthesis was made for TSMC 28HPC+ technology. It means it uses 28 nm CMOS logic for high performance compact mobile computing. The *tcbn28hpcplusbwp7t35p140ssg0p81v125c* library was used. Because of using this library it uses the global operating voltage equal to 0.81 V instead of typical 0.9 V.

The synthesis tool reported that it was used 2777 registers and found no timing violators. Information about the area of the module on a chip are summarized in the table 3.14. Results about the power consumption are summarized in the table 3.15. The dynamic consumption reported by the synthesis is for a default signal activity. In reality, it can be different.

Area Parameter	Result
Number of ports	1315
Number of nets	9774
Number of cells	8689
Number of combinational cells	5818
Number of sequential cells	2803
Number of macros/black boxes	0
Number of buf/inv	719
Number of references	12
Combinational area [μm^2]	2515.855975
Buf/Inv area [μm^2]	162.875997
Noncombinational area [μm^2]	6732.698074
Total cell area [μm^2]	9248.554049

Table 3.14: Post-Synthesis area report for ASIC

Power Parameter	Result
Cell Internal Power [mW]	1.4011
Net Switching Power [μW]	35.9431
Total Dynamic Power [mW]	1.4370
Cell Leakage Power [μW]	190.3822
Total Power [mW]	1.6274

Table 3.15: Post-Synthesis power report for ASIC

3.5 Validation on FPGA

As it was said above, the verification on a FPGA itself did not happen during the elaboration of this thesis. The most significant modules were prepared. This means the receiver, GTP transceiver and gearbox. It remained, to modify the Xilinx JESD204 IP core, so it met requirements of the desired FPGA board. Also several circuits for timing references were needed to be developed. But the most crucial was to implement a controller for the communication with Xilinx JESD204 IP cores via the AXI4 interface, so it could be configured. Most likely it should have been the MicroBlaze processor implemented on the FPGA.

The data path for a validation process on a FPGA is displayed in the figure 3.22. It starts with the MicroBlaze processor implemented on a FPGA. It is used as a control device for AXI4 interface. By means of this interface, the transmitter, RAM and FIFO are configured. The RAM is supplying data for the transmission. Data from the TX core are then connected to the GTP transceiver outside the FPGA, because of the requirements of the FPGA. Data are then transmitted via the gearbox to the designed receiver. The processed data are saved in the FIFO memory and compared with those ones, which were transmitted at the beginning of the chain.

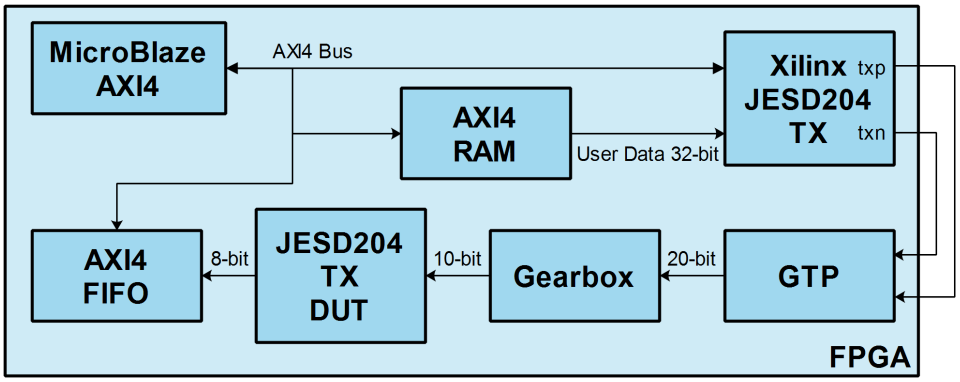


Figure 3.22: Data path illustration for validation on FPGA

Chapter 4

Future Updates

The future work subsequent to this thesis would probably be a replenishment of two other subclasses of JESD204B standard. In the current state, it is actually a part of JESD204B and it does not support all features, especially the deterministic latency. For further development it will be suitable to use some more advanced simulation tool, for example Xcelium by Cadence, in the case of more efficient work. The reason for this is that the work with ModelSim in student version and then the verification in Vivado was quite much time consuming and demanding. ModelSim is not compatible with Xilinx IP cores and the simulation in Vivado took too long. The use of current paid version of ModelSim would bring the benefit of the ability of simulations with Xilinx IP cores too. Reserves are also likely to occur in the efficiency and readability of the code, not to say the size of the resulting design and its power consumption, where it would certainly be possible to make further adjustments to improve the properties of the device. In addition, there are some functionalities missing in the resulting device, such as reporting of an unexpected control character, which means that a control character received is not expected at the given character position, or reporting of the realignments in the frame and lane synchronization. Furthermore, the device does not support test sequences of continuous stream of /K28.5/ characters for code group synchronization (it just synchronizes and will be waiting for ILAS) and it does not have the capability to suppress error reports due to missing ILA configuration data for bit error measurements.

Chapter 5

Conclusion

Theoretical part of the thesis deals mainly with the specifications of the JESD204 protocol and also with the description of Verilog and Xilinx multi-gigabit transceivers. The practical part was about designing the receiving link layer of JESD204B and also about its verification in multiple tools in several situations.

The result of the thesis is the receiving link layer of JESD204B receiver for the subclass 0. Only some minor functionalities listed in the standard are missing. The main goal of the thesis was to develop the link layer of the JESD204 receiver and it can be stated, that it has been done with exceptions of subclasses 1 and 2. Also, the simulation scheme with the reference Xilinx JESD204 IP core, GTP transceiver and gearbox was prepared for a further verification and it can be used as the basis for a validation on a FPGA. For the implementation of it, more time would be needed. The rest of the goals were met during the elaboration of the thesis. In addition, there has been a great development in skills within the Verilog language.

During this work, several difficulties were faced. The first one, right at the beginning, was simulating of the Xilinx IP cores and the rest of the design. The official tool within Xilinx environment, Vivado, was found not suitable for it due to its time consuming processes. So another tool, ModelSim, was chosen with the limitation of no possibility of including Xilinx IP cores. As the designed module was getting more complex and the presence of IP cores in the simulation was needed, another tool had to be found for issuing simulations and it was Cadence Xcelium. Other difficulties lied in an unfamiliarity with the Verilog language, which also took some time during the work.

Bibliography

[1] JESD204B.01. : ©JEDEC Solid State Technology Association, 2012.

[2] JESD204C: Revision of JESD204B.01 January 2012. : ©JEDEC Solid State Technology Association, 2017.

[3] JESD204A (Revision of JESD204, April 2006). : ©JEDEC Solid State Technology Association, 2008.

[4] JESD204B Survival Guide: Practical JESD204B Technical Information, Tips, and Advice from the World's Data Converter Market Share Leader. Analog Devices [online]. 2011, 2011, , 2-76 [cit. 2020-12-26]. Dostupné z: https://www.analog.com/media/en/technical-documentation/technical-articles/JESD204B-Survival-Guide.pdf

[5] JESD204B vs. Serial LVDS Interface Considerations for Wideband Data Converter Applications. Analog Devices [online]. 2019, , 1-3 [cit. 2020-12-26]. Dostupné z: https://www.analog.com/en/technical-articles/jesd204b-vs-serial-lvds-interface-considerations-for-wideband-data-converter-applications.html

[6] CMOS, LVDS, and CML Driver Power Comparison. In: Analog Devices [online]. Wilmington: Analog Devices, 2019 [cit. 2021-01-09]. Dostupné z: https://www.analog.com/-/media/analog/en/landing-pages/technical-articles/what-is-jesd204-and-why-should-we-pay-attention-to-it/figure-4.jpg?w=900&la=en

[7] IEEE Std 802.3-2008®, Part 3, Section Three, Local and metropolitan area networks - CSMA/CD access methods and Physical Layer specifications, 2008. http://standards.ieee.org/getieee802/

[8] HARRIS, Jonathan. Understanding Layers in the JESD204B Specification—A High Speed ADC Perspective. Analog Devices [online]. 2017, 2017(MS-2714), 1-6 [cit. 2021-01-23]. Dostupné z: https://www.analog.com/en/technical-articles/understanding-layers-in-jesd204b-specification.html#author

[9] Vivado Design Suite User Guide: Getting Started. Xilinx Inc. [online]. 2014 [cit. 201821-04-15]. Dostupné z:

https://www.xilinx.com/support/documentation/sw_manuals
/xilinx2014_1/ug910-vivado-getting-started.pdf

[10] KOLOUCH, Jaromír. Jazyk Verilog a jeho užití při modelování a syntéze číslicových systémů: příručka. Brno: VUTIUM, 2012. ISBN 978-802-1445-161.

[11] 7 Series FPGAs GTP Transceivers: User Guide. V1.9. Xilinx, 2016. Dostupné z: https://www.xilinx.com/support/documentation/user_guides/ug482_7Series_GTP_Transceivers.pdf

[12] JESD204 v7.2: User GuideLogiCORE IP Product Guide. Xilinx, 2017. Dostupné z: https://www.xilinx.com/support/documentation/ip_documentation/jesd204/v7_2/pg066-jesd204.pdf

[13] Incisive Coverage Introduction and RAK Overview. Cadence [online]. San Jose: Cadence, 2013 [cit. 2021-5-10]. Dostupné z: http://docshare01.docshare.tips/files/31703/317032353.pdf

Appendix A

Contents on the enclosed CD