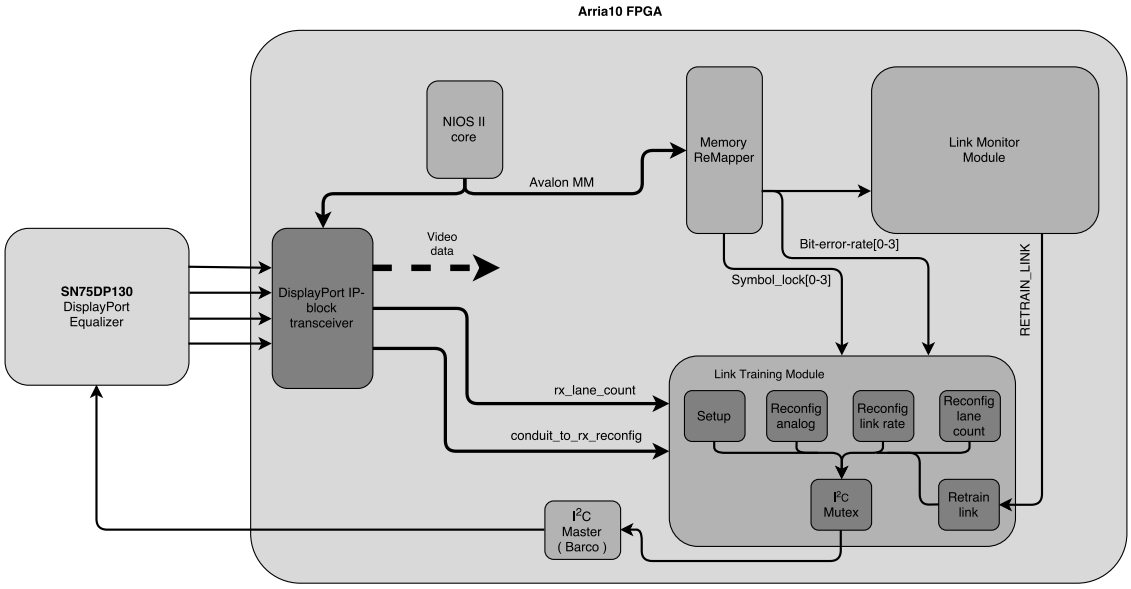
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



DisplayPort Link training optimization

Master's thesis in Embedded Electronic System Design

MATS ERIK SETTERBERG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

DisplayPort Link training optimization

MATS ERIK SETTERBERG

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

DisplayPort link training optimization
MATS ERIK SETTERBERG

Supervisors: Svein-Arne Jervell Hansen, Barco - Lars Svensson, Chalmers
Examiner: Per Larsson-Edefors

Master's Thesis 2017
Department of Computer Science and Eningeering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Block diagram of the final project implementation.

Typeset in LaTeX
Gothenburg, Sweden 2017

DisplayPort link training optimization
Mats Erik Setterberg
Department of Computer Science and Eningeering
Chalmers University of Technology

Abstract

As the requirement for bandwidth continues to increase in the video market, retaining the signal integrity becomes increasingly more difficult. For many of todays commonly used video interfaces, there are devices that can be used to assist in this matter. However, the use of such a device is only partially documented in the DisplayPort specification for the receiving image device, which means that the receiving side of the video link is free to choose its own implementation. This report presents, together with background research and design decisions, a suggestion for such an implementation. This implementation would need to be compatible towards a wide range of possible video Source devices and DisplayPort cables. This suggestion will be tested, implemented and verified using the Pulse platform developed by Barco.

Keywords: DisplayPort, link training, equalizer, equalization, redriver, Barco

Acknowledgements

First of all, I would like to thank my Barco supervisor Svein Arne Jervell Hansen for all help during this project. Your guidance has been very much appreciated. I would also like to thank my Chalmers supervisor Lars Svensson for all the help with the writing that has gone into this report.

I would also like to thank Barco Fredrikstad for giving me the opportunity to do this project with them. A special thanks to Tarjei Fjelgaard, Jorgen Krohn, Edvard Fosdahl, Jorgen Dahlback and the rest of the R&D group for all input and help along the way.

Mats Erik Setterberg, Gothenburg, June 2017

Contents

List of Figures **xi**

List of Tables **xiii**

List of Acronyms **xv**

1 Introduction **1**
1.1 Background . . . 1
1.2 Barco . . . 2
1.3 Goal . . . 2
1.4 Delimitations . . . 2
1.5 Ethical Aspects . . . 3
1.6 Report structure . . . 3

2 Theory and technical background **5**
2.1 High-speed signal propagation on slow mediums . . . 5
2.2 The DisplayPort protocol . . . 6
2.2.1 Overview . . . 6
2.2.2 The data channel . . . 7
2.2.3 The AUX channel . . . 8
2.2.4 DisplayPort Configuration Data (DPCD) . . . 8
2.2.5 Link training in general . . . 8
2.2.6 Signal propagation in DisplayPort . . . 9
2.3 The Barco Pulse platform . . . 10
2.3.1 DisplayPort Equalizer SN75DP130 . . . 10
2.3.2 The DisplayPort IP Core . . . 12
2.4 I^2C . . . 13
2.5 Avalon Communication Interfaces . . . 13

3 Method **17**
3.1 Equipment . . . 18

4 Implementation **19**
4.1 Link training in detail . . . 19
4.1.1 Clock Recovery . . . 20
4.1.2 Channel Equalization . . . 20
4.2 Link training module . . . 22

4.2.1 Before starting the design . . . 23
4.2.2 System overview . . . 24
4.2.3 I²C Master . . . 25
4.2.4 I²C bus handler . . . 26
4.2.5 Equalizer initiation . . . 30
4.2.6 Equalizer gain reconfiguration . . . 31
4.2.7 Equalizer link rate reconfiguration . . . 34
4.2.8 Equalizer lane count reconfiguration . . . 35
4.3 Nios II core . . . 38
4.4 MemoryReMapper . . . 38
4.4.1 Input and output signals . . . 39
4.4.2 Block functionality . . . 40
4.5 Link monitor module . . . 40
4.5.1 Before starting the design . . . 41

5 Results and discussion **43**
5.1 Design verification . . . 43
5.1.1 Timing analysis . . . 43
5.2 Project time plan . . . 45
5.3 Future work . . . 46
5.3.1 Avoiding DisplayPort deviations . . . 46
5.3.2 Compatibility towards other equalizers . . . 46
5.3.3 The link training module . . . 47

6 Conclusion **49**

A Appendix 1 - Project time plan **I**

List of Figures

2.1 Illustration of a DisplayPort Link 6
2.2 Impact of pre-emphasis on the electrical signal [1]. 7
2.3 Block diagram showing the contents of the Barco Pulse platform. . . 10
2.4 Core of the SN75DP130 DisplayPort Equalizer. 11
2.5 Frequency response for different gain levels of the equalizer stage inside the SN75DP130 [2]. 12
2.6 A typical data transfer over the I^2C bus [3]. 13
2.7 Waveforms of a standard Avalon MM read operation. 15
2.8 Waveforms of a standard Avalon MM write operation. 15

4.1 A block diagram of the entire system. 19
4.2 Flow chart for the clock recovery sequence during link training. . . . 21
4.3 Flowchart for the channel equalization sequence during link training. 22
4.4 Blockdiagram of the link training module 25
4.5 Inputs and outputs from the I^2C Master component 26
4.6 Inputs and outputs for the I2C handler block 27
4.7 Flowchart for the I^2C bus handler 29
4.8 Timing diagram for changing the equalizer link rate. 30
4.9 Flowchart of the equalizer initializer. 31
4.10 Illustration of the equalizer gain reconfiguration block. 31
4.11 Flowchart for the equalizer gain reconfiguration block 33
4.12 Inputs and outputs of the equalizer link rate reconfiguration block. . 34
4.13 Flowchart for the equalizer link rate reconfiguration block. 36
4.14 Flowchart for the lane count reconfiguration block. 37
4.15 Flowchart of the Nios II soft processor program. 39
4.16 Flowchart for the memory remapper. 40

5.1 Oscilloscope plot showing the I^2C bus activity during a single iteration of link training during clock recovery. Bottom signal is the clock signal, while the top signal is the data. 44
5.2 Oscilloscope plot showing the I^2C bus activity during link training. Bottom signal is the clock signal, while the top signal is the data. . . 44
5.3 Blockdiagram for the SN65DP141 [4]. 46

List of Tables

2.1 Valid combinations of pre-emphasis(PRE) and voltage swing(VOD)[5]. 7
2.2 Different lane speeds available for DisplayPort [5]. 8
2.3 Equalization levels of the SN75DP130, based on link speed [2]. 12

4.1 Data contents of the `rx_reconfig` vector, based on current lane count. 34

5.1 Timing slack analysis of the project VHDL-implementation. 45

List of Acronyms

AUX Auxiliary
dB Decibel
DP DisplayPort
DPCD Display Port Configuration Data
DVI Digital Visual Interface
FPGA Field Programmable Gate Array
Gbps Gigabit per second
GPU Graphics Processing Unit
HBR1 High Bit Rate 1
HBR2 High Bit Rate 2
HDMI High-Definition Multimedia Interface
HPD Hot-Plug Detect
Hz Hertz
I2C Inter-Integrated Circuit
IP Intellectual Property
Avalon MM Avalon Memory Mapped Interface
PCB Printed Circuit Board
PLL Phase-Locked Loop
PRE Pre-emphasis
RBR Reduced Bit Rate
SCL Serial Clock Line
SDA Serial Data Line
Sink The receiving side of a DisplayPort video link is referred to as the sink
Source The transmitting side of a DisplayPort video link is referred to as the sink
VESA Video Electronics Standards Association
VHDL Very High Speed Integrated Circuit Hardware Description Language
VOD Voltage swing

1
Introduction

The video display market today presents a wide range of available display devices in the form of projectors, TVs and computer monitors. A common factor for all of these is that they need some kind of interface between the display device and the video source. A wide range of interfaces are available for this purpose, but with the introduction of the 4K resolution displays, and with 8K displays on the horizon [6], the requirements for these interfaces have significantly increased.

1.1 Background

The DisplayPort protocol was first released by the Video Electronics Standards Association (VESA) in 2006 and has been upgraded to new versions in the following years with support for extra functionality and higher bandwidths. DisplayPort[7] is one of today's more commonly used interfaces, together with HDMI[8] and DVI[9].

After the introduction of version 1.2 in 2009, DisplayPort has supported data rates up to 5.4 Gbps. Transfers at this speed, especially across slow mediums like copper which is commonly used in DisplayPort cables, introduces signal attenuation and noise. This grows with the length of the cable. DisplayPort includes a feature to counteract some of these problems, called the link training procedure. The purpose of link training is to acquire the most optimal transmission settings possible for data transfer between the source and receiving DisplayPort device.

Even if the DisplayPort protocol includes some functionality to counteract these problems, it might not be able to compensate for all the signal loss in some cables. An equalizer or re-driver may be added to the video link to help counteract these problems even further. The use of such a component is partially documented for the source side of the link [5], but the DisplayPort standard does not include any implementation guidelines for the use of this component on the receiving side of the link. Even though there are some parts of the protocol that have taken the use of such a component into consideration, the receiver side is free to choose its own implementation. The DisplayPort equalizer and re-driver components are usually designed with a wide range of settings that are changeable to optimize the component towards each specific implementation.

1.2 Barco

Barco is a global technology company that develops and manufactures high end projectors [10]. Barco has a wide projector product range that covers everything from small venue set-ups all the way to large cinema projectors [10]. The new Pulse platform from Barco is fully compatible with DisplayPort and uses a DisplayPort equalizer and re-driver in the DisplayPort video link. As Barco has a main focus on high quality rather than low cost, their customers expect no problems when connecting their devices to any image source. Barco is therefore interested in an implementation that uses the flexibility of the equalizer or re-driver settings to improve the outcome of the link training procedure.

1.3 Goal

The goal of this thesis project is to create an implementation that utilizes the full flexibility of an equalizer during the DisplayPort link training procedure in order to improve the outcome of the link training procedure.

If possible, this implementation should also be extended to included real-time monitoring of the video link. If some unexpected event would cause an increase in signal attenuation or noise over the video link, the equalizer should be configured in real time to try to prevent the video link from shutting down.

1.4 Delimitations

Due to the limited time span, some delimitations have been set for the thesis project.

- Even though there are several DisplayPort equalizers available on the market, only the SN75DP130[2] from Texas Instruments will be tested during this project.

- The optimized DisplayPort sink equalizer needs to work across a wide range of sources and set-ups. Because of the limited time budget, the devices used for testing needs to be limited to one from Nvidia, one from AMD and one from Intel.

- No new hardware will be added to the Barco Pulse platform.

- The DisplayPort protocol supports custom test patterns to be used during link trainnig [5], in order to achieve optimal link quality. This feature will not be investigated.

1.5 Ethical Aspects

The ethical aspects of this project has been taken into consideration and concluded not to be highly relevant.

The final implementation could lead to more DisplayPort cables and sources being compatible towards more DisplayPort sink devices. This would reduce the total number of new cables and sources needed to be purchased, where the devices are not compatible. Over a longer time span, this could reduce the environmental impact of cables and electronics thrown into the trash. Electronics and cables often contain a lot of copper, which can then be used for other things.

1.6 Report structure

This thesis report starts with presenting theoretical aspects that is required in order to follow the design decisions during construction of the target implementation. Block diagrams and flow charts of the implementation will then be presented together with a walk through of the design decisions made for this implementation. Test result from the design verification stage will be presented at the end of the work. Future work will also be discussed.

2

Theory and technical background

This chapter provides some of the technical background knowledge needed to follow and understand some of the implementation- and design decisions made during this thesis project.

2.1 High-speed signal propagation on slow mediums

As signal frequencies move into the gigahertz domain, the use of slow mediums, such as copper wires or PCB-traces, become increasingly more difficult [11]. If not handled properly, the medium can distort the signal beyond recognition. [12, 13]. This happens especially if the impedance between two conductors, such as a PCB-trace and a cable, is mismatched [11]. Impedance is defined as the effective resistance of an electronic circuit during alternating current [14]. The impedance of a conductor is represented by the following formula:

$$Z_0 = \sqrt{\frac{R + j\omega L}{G + j\omega C}} \quad , \quad G = \frac{1}{R} \, , \quad \omega = 2\pi f \tag{2.1}$$

, where R is the resistance, L is the inductance, G is the conductance and C is the capacitance of the conductor per unit length [15]. f is the frequency of the signal traveling along the conductor. The resistance, inductance and capacitance depends on the length, width and height of the conductor [11]. Formulas for calculating these parameters are different for each signal medium.

As shown by Equation 2.1, the impedance of the conductor depends on the frequency of the signal. On lower frequencies, the impact of the L and C components of the equation is reduced. This removes a lot of the design challenges that occur on higher frequencies, where the impedance of the signal path must be taken into account during the design process. If this is not handled correctly, certain phenomena, such as oscillations or ringing, might start to appear on the signal [13]. The cause of these problems are often reflections. This may lead to signal distortion so severe that the receiver might not be able to decode the signal. The frequency of the signal then needs to be lowered in order to allow information to flow through the medium.

Reflections occur when a signal moves between mediums with different impedance. Reflections can be limited by having the same impedance for all used mediums. This is often referred to as impedance matching [12]. Impedance matching significantly reduces the amplitude of the reflection generated, and is achieved in different ways for different mediums. For common interfaces, such as DisplayPort and HDMI, the cables and PCB-traces are designed for a specific characteristic impedance [5].

Because the reflections and attenuation will distort the signal, DisplayPort has an assigned procedure called link training. This procedure is intended to compensate for these signal distortions, and thereby make error free communication possible.

2.2 The DisplayPort protocol

The DisplayPort protocol is published and developed by the Video Electronics Standards Association(VESA) [16]. The DisplayPort protocol was first introduced with version 1 in 2006, but new versions were presented by VESA in later years with version 1.2 in 2009, 1.3 in 2014 and 1.4 in 2016. This thesis project is based around the DisplayPort version 1.2, which is the only version that will be discussed further in this report, unless otherwise specified.

2.2.1 Overview

An illustration of a typical DisplayPort link between a source and a sink device is shown in Figure 2.1. This link consists of the main data link, the AUX channel and the Hot-Plug Detect(HPD) signal. The HPD signal is used by both the source and the sink to detect when two devices has been connected together.

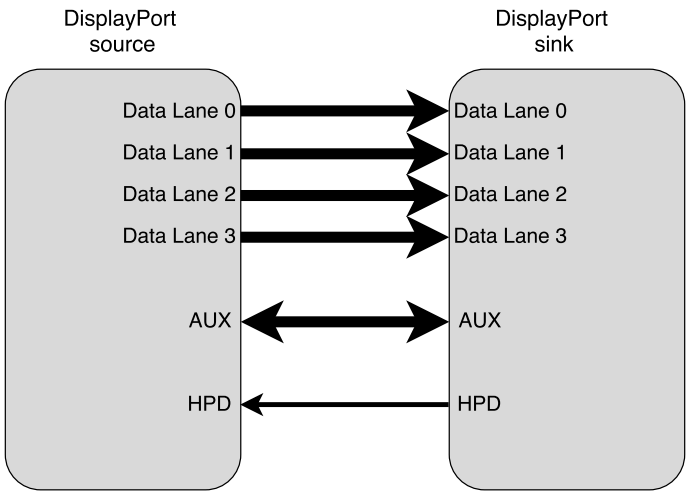


Figure 2.1: Illustration of a DisplayPort Link

2.2.2 The data channel

The data channel consists of a total of four high-speed parallel data lanes and handles all the video transfer over the DisplayPort link. The data channel mainly has four different settings:

- Voltage swing (VOD)
- Pre-emphasis (PRE)
- Lane speed
- Lane count

The voltage swing is the amplitude of the electrical signal. This varies between 340 mV and 1000 mV for the lowest and the highest setting [2]. The pre-emphasis parameter adds an overshoot to the signal as it leaves the transmitter. This overshoot is added to the signal to compensate for signal attenuation in the medium. An example of how the pre-emphasis impacts the electrical signal is shown in Figure 2.2.

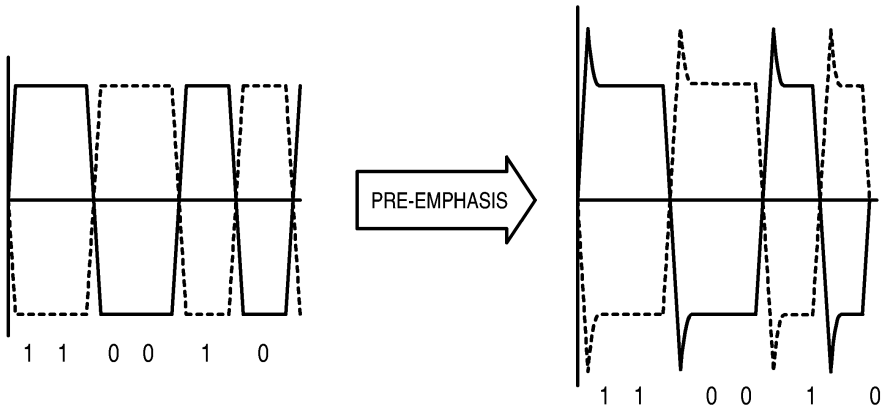


Figure 2.2: Impact of pre-emphasis on the electrical signal [1].

As both the voltage swing and pre-emphasis is used to control the electrical signal, there are some combinations of these settings that are invalid, which is represented in Table 2.1 below.

Table 2.1: Valid combinations of pre-emphasis(PRE) and voltage swing(VOD)[5].

	PRE0	PRE1	PRE2	PRE3
VOD0	**OK**	**OK**	**OK**	**OK**
VOD1	**OK**	**OK**	**OK**	*NOK*
VOD2	**OK**	**OK**	*NOK*	*NOK*
VOD3	**OK**	*NOK*	*NOK*	*NOK*

The lane speed represents the transfer speed of each respective lane. The different lane speeds defined in the protocol are Reduced Bit Rate(RBR), High Bit Rate 1(HBR1) and High Bit Rate 2(HBR2), presented in Table 2.2

Table 2.2: Different lane speeds available for DisplayPort [5].

Name	Abbreviation	Speed (Gbps)
High Bit Rate 2	HBR2	5.4
High Bit Rate 1	HBR1	2.7
Reduced Bit Rate	RBR	1.6

2.2.3 The AUX channel

The AUX channel is a separate communication channel between the source and the Sink device. This channel runs at a much lower speed compared to the main data channel, running at about 1 Mbps [5]. This channel is mainly used during the link training procedure, before communication over the main link has been established.

2.2.4 DisplayPort Configuration Data (DPCD)

DPCD is a register map used by both the source and receiver device in a DisplayPort video link. The register map holds information such as current lane count, current lane speed, pre-emphasis, voltage swing, max supported lane speed by the source and max supported lane speed by the receiver. There are a lot of other information stored in the DPCD register. The contents of these registers will be discussed in further detail when they become relevant later in this report.

2.2.5 Link training in general

When a DisplayPort source and receiving device is connected together using a DisplayPort cable, a procedure called Link Training is initiated. The purpose of link training is to acquire the most optimal transmission settings possible for data transfer between the source and receiving DisplayPort devices. With the increased requirement for bandwidth over the physical layer, the link training procedure becomes more important to enable good signal integrity for the high speed video signals. Cables built using slow mediums, such as copper, may introduce noise and signal attenuation [11]. The vast number of available DisplayPort sources and DisplayPort cables available on the market makes the number of possible source- and cable-combinations almost infinite. This means that the sink side implementation of the DisplayPort link needs to be quite flexible to enable linking with as many other devices as possible.

An equalizer or re-driver may be used in order to counteract some of these problems. The use of an equalizer or re-driver is partially documented in the DisplayPort standard for the source side of the link [5], but there are no implementation guidelines for the use of this component on the receiving side of the link. Even though there are some parts of the protocol that have taken the use of such a component into consideration, the sink side is free to choose its own implementation.

There are a number of different re-drivers or equalizers available on the market [17, 18, 19]. One of these is the SN75DP130 from Texas Instruments [2]. This components contains both a reconfigurable equalizer and re-driver. This device is very flexible due to all its configurable settings. If this flexibility is taken advantage of and the DisplayPort re-driver is dynamically configured for each setup, the link training should be able to succeed with a wider range of DisplayPort sources and cables.

2.2.6 Signal propagation in DisplayPort

The communication link can be set up either as an internal chip-to-chip configuration, or an external box-to-box configuration [5]. The chip-to-chip configuration is often referred to as embedded-DisplayPort and is used between chips on a PCB. Box-to-box configurations are often between PCs and monitors, TVs or projectors. The box-to-box configuration requires a cable to connect each device. Some cables, if not properly manufactured, can introduce a lot of noise into the video signal. This noise can originate from external sources, or crosstalk between the conductors in the cable itself. The PCB traces used between the two chips in a chip-to-chip configuration, or between the DisplayPort transmitter and the DisplayPort connector in a box-to-box configuration, can introduce signal attenuation. Both of these problems can be mitigated by the use of an equalizer and a re-driver.

Taking a standard box-to-box configuration as an example, the signal gets affected in different ways along the signal path. The path mainly consists of PCB traces and the cable. Starting off at the PCB level, the DisplayPort specification suggest that the two traces making up the differential pair for each lane should be as close to each other as possible [5]. This introduces a big capacitive load on the differential pair, which in its turn also increases the noise resilience of the conductor [20]. In order to match this differential pair to the correct impedance, the thickness of the trace needs to be reduced, as this increases the conductor inductance [20]. The downside of reducing the width of the trace is that the resistance increases significantly, which also increases the signal attenuation in the conductor [21]. This is one of the reasons that the re-driver on the DisplayPort link is placed close to the DisplayPort connector on the source side. Because the signal gets attenuated between the GPU transmitter and the connector, the re-driver is needed to make sure that the signal integrity are as good as possible before the signal starts traveling across the cable.

In the cable, the problem is a little bit different. Here, the distance between the positive and negative rail of the signal is larger, in addition to the dielectric properties of the cable materials are much different than that of the PCB. The thickness of the conductor in the cable is larger compared to the PCB. This reduces the resistance in the cable, which in turn reduces the signal attenuation. Now, one might think that this solves a lot of the problems that were present on the PCB, but there still is a small catch to this. Because the distance between the leads in the cable now has increased, the conductor capacitance is much lower. This means that the signal

is much more susceptible to external noise. The four lanes in the video link are now also connected very close together in parallel. This introduces a lot of cross-talk between the signals. This added noise gets increasingly worse with longer cables. This is why longer cables does not manage to run at the maximum transfer speeds. The DispalyPort protocol specifies a maximum cable length of 1.8 meters when running at 5.4 Gbps [5].

After the cable, the sink device may place an equalizer or re-driver right at the input connector. The equalizer will then filter out the noise introduced in the cable, while the re-driver amplifies the signal before being sent across the PCB leads to the receiver chip.

2.3 The Barco Pulse platform

The hardware platform used for this thesis is the new Pulse platform designed by Barco. This platform is designed to output video at 4K resolution with up to 120 frames per second. The platform is FPGA-based with an external equalizer from Texas Instruments. An overview of the platform is shown in Figure 2.3. The input from the DisplayPort cable is connected directly to the Texas Instruments SN75DP130 equalizer. This equalizer is controlled via a I^2C bus from the FPGA. The video signal output from the equalizer is connected to a transceiver input on the FPGA.

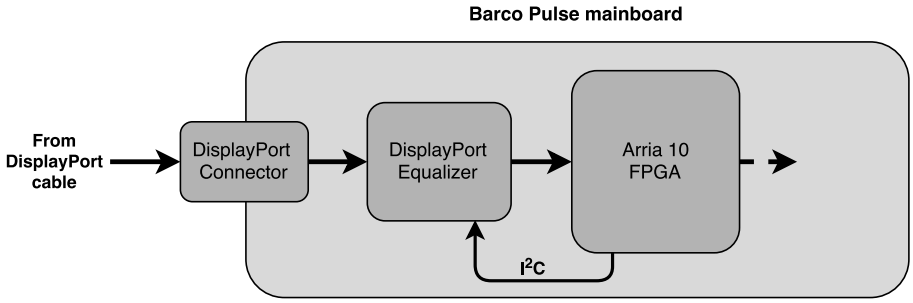


Figure 2.3: Block diagram showing the contents of the Barco Pulse platform.

2.3.1 DisplayPort Equalizer SN75DP130

The SN75DP130 contains both a re-driver and an equalizer, as well as some built-in link training functionality. These features makes the device suitable for both source and sink implementations. A simplified block diagram of the architecture of the equalizer is shown in Figure 2.4.

The SN75DP130 is divided into two parts; the equalizer stage and the re-driver stage. Settings for each stage can be set by changing registers within the device

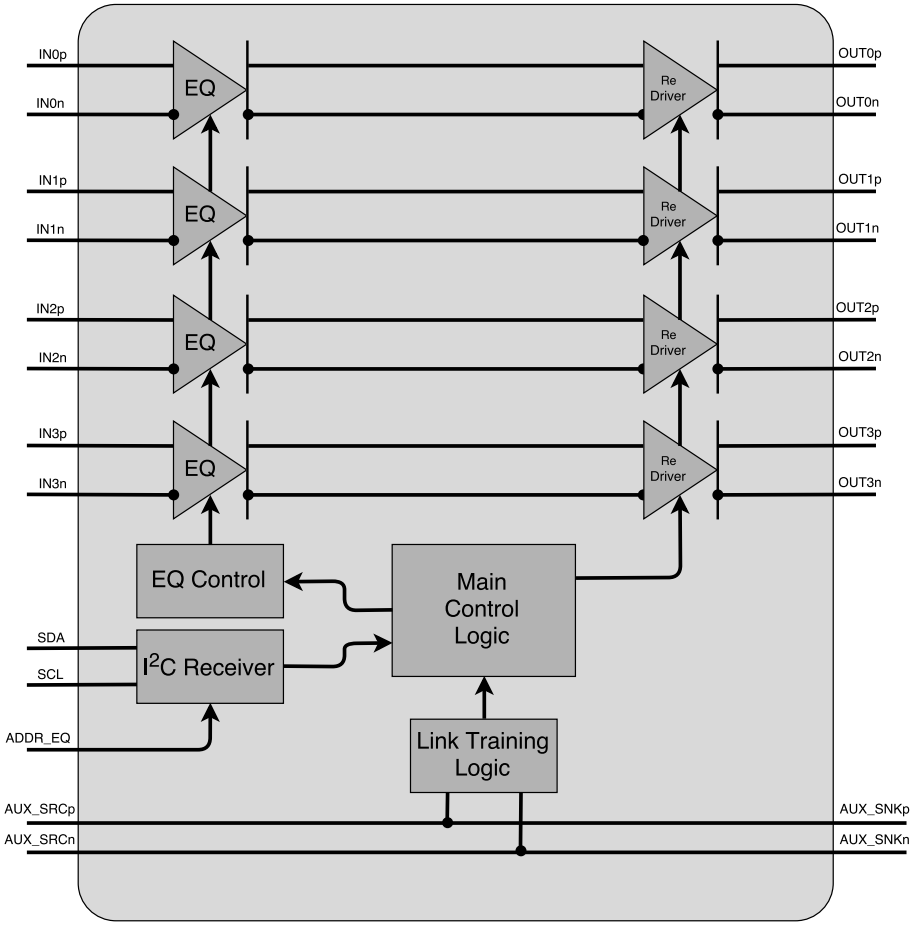


Figure 2.4: Core of the SN75DP130 DisplayPort Equalizer.

through the I^2C interface. The maximum speed for the I^2C interface is 100 kHz. The I^2C address of the equalizer is selectable by setting the voltage level on the `ADDR_EQ` pin.

The first stage is the equalizer stage, taking the four differential data lanes as inputs. The purpose of the equalizer is to reduce frequency response deviations and possible noise introduced to the signal. There are eight level of equalization settings available for the equalizer input, ranging from level 0 to level 7. Each level corresponds to an amount of equalization, that is dependent on the link speed. These equalization levels is listed in Table 2.3. Changing the level of equalization changes the impulse response of the equalizer stage. See Figure 2.5 for an overview of the frequency response based on equalization level.

Following the equalization stage is the re-driver part of the component. The re-driver will output the incoming video data with new pre-emphasis and voltage levels, specified in the component register map.

Table 2.3: Equalization levels of the SN75DP130, based on link speed [2].

Level	Equalization	
	HBR1(dB)	HBR2(dB)
0	0	0
1	1.5	3.5
2	3	6
3	4	8
4	5	10
5	6	13
6	7	15
8	9	18

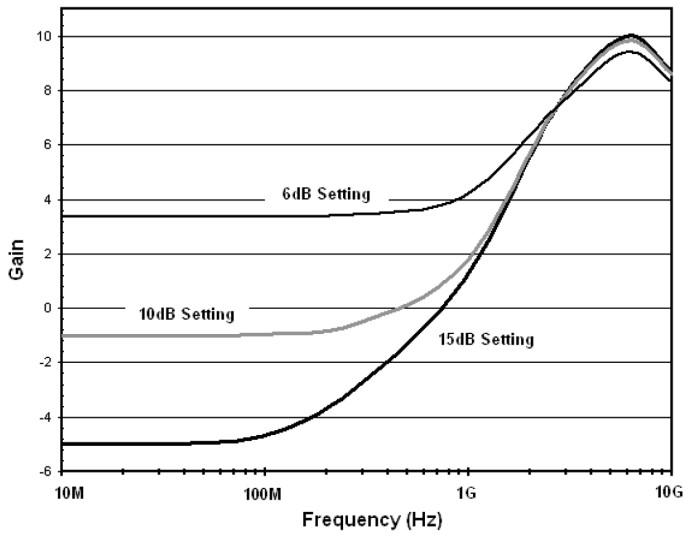


Figure 2.5: Frequency response for different gain levels of the equalizer stage inside the SN75DP130 [2].

2.3.2 The DisplayPort IP Core

The FPGA hosts a 3rd party DisplayPort IP block. This IP block acts as the DisplayPort receiver controller and manages everything related to the DisplayPort video link such as link training, decoding the video signals, bit error counting and more. The IP block offers a wide range of trigger signals, status bits and other information available as VHDL ports. These ports will be presented later in this report when relevant.

Some additional information might be needed that is not available on the VHDL port. The IP block includes an Avalon MM interface that can be used to read the internal registers of the IP block. More about the Avalon MM interface is presented in Section 2.5.

2.4 I^{2}C

I2C is a 2-wire bidirectional interface developed by Philips Semiconductor[3]. The interface allows communication between one or more master nodes and a set of slave nodes. The communication bus only consists of two wires, the SDA data line and the SCL clock line. These two wires are connected to all nodes on the I^{2}C bus. [3].

The bus uses an address priority system. The lower the address number, the higher priority of the message. Address zero gets the highest priority. A typical I^{2}C transfer is represented by Figure 2.6 below.

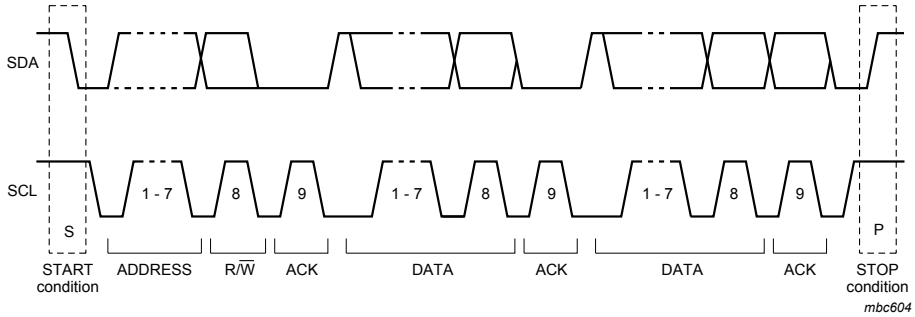


Figure 2.6: A typical data transfer over the I^{2}C bus [3].

The transfer is started by the SDA pin being pulled low. This is followed by the 8 address bits that represents the receiver address. The 8th bit in the address byte is used to differentiate between read and write commands. For every byte sent, there needs to be an acknowledgement signal from the receiver. This acknowledgement signal is used by the receiver to signal to the sender that the byte was successfully received [3].

Followed by the address byte and the acknowledgement bit, the first data byte is sent. This data byte is also acknowledged by the receiver. The sender can send more than one databyte to the receiver. This is signaled by a repeated start signal, followed by the second data byte. This will repeat untill the sender sends a stop signal. After the stop signal, other nodes can start using the bus, or the same master can start transmitting data to other nodes with different addresses.

2.5 Avalon Communication Interfaces

The Avalon communication interfaces is developed by Altera with a purpose to simplify communication between IP blocks within Altera FPGAs[22]. The Avalon interface can also be connected to external chips outside the FPGA, if the Avalon interface is supported by the chip.

The Avalon Communication interface consists of different sub-protocols, which is listed below:

- Avalon Clock and Reset Interfaces
- Avalon Memory-Mapped Interfaces (Avalon MM)
- Avalon Interrupt Interfaces
- Avalon Streaming Interfaces
- Avalon Conduit Interfaces
- Avalon Tristate Conduit Interfaces

All of these interfaces are designed for different purposes, but only the Avalon Memory-Mapped Interface is used for this thesis project. Therefore, only this sub-protocol is discussed in this section.

The Avalon Avalon MM interface is used in a master-slave configuration, where each Avalon MM interface contains one master node and several slave nodes. How many slave nodes that can be used depends on how big of an address area that is reserved for each slave node.

The mandatory signals in a Avalon MM interface is listed below:

- `clk` is the communication interface main clock.
- `read` is a single bit signal which is asserted by the master when it wants to read from the slave.
- `write` is a single bit signal which is asserted by the master when it wants to write to the slave.
- `chipselect` is a single bit signal that controlled via a separate memory conduit block.
- `address` is the address of the memory location the master wants to read from or write to. This vector can be of variable bit size.
- `readdata` contains the data read from the memory specified memory location. This vector can be of variable bit size.
- `writedata` contains the data that the master wants to write to the specific memory location. This vector can be of variable bit size.

A typical Avalon MM read operation is illustrated in Figure 2.7. The `read` and `chipselect` signals will be asserted at the same time as the address for the requested register is presented on the address bus. This will be picked up by the slave on the following avalon bus clock sycle. The requested data will be presented on the `readdata` bus on the next clock cycle.

A typical Avalon MM write operation is illustrated in Figure 2.8. On a rising edge

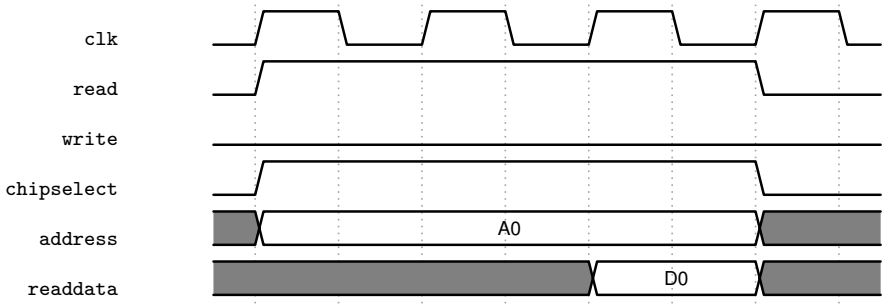


Figure 2.7: Waveforms of a standard Avalon MM read operation.

of the bus clock, the master will present the data and address on each dedicated vector. The `write` and `chipselect` signals will also be asserted. The data will be stored in the slave memory on the next rising edge of the bus clock.

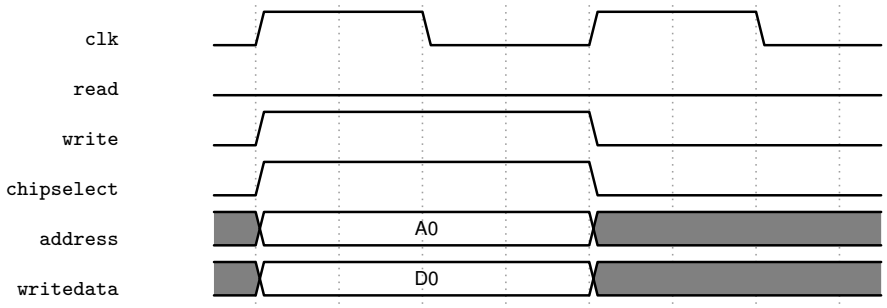


Figure 2.8: Waveforms of a standard Avalon MM write operation.

3
Method

A work plan for the entire project will be created, based on the project description written by Barco. The steps in this work plan, as well as the equipment needed to complete the project, is further explained in this chapter.

The first part of the thesis project will be to acquire all the background knowledge needed to design the implementation requested by Barco. This will be done by studying relevant datasheets and documentation for the related protocols and components. This includes:

- Technical information regarding the Barco Pulse platform
- The DisplayPort protocol
- The I^2C protocol
- Datasheet for the equalizer
- Documentation for the DisplayPort IP core
- Inspecting the already implemented VHDL code

This information is critical in order to create an implementation that fulfills all requirements set by Barco, but at the same time does not interfere with any of the surrounding components in the system.

During the second part of the thesis project, the focus will be on implementation of the DisplayPort link training module. The purpose of this module will be to set up the equalizer correctly during link training. Texas Instruments have published an application note on the implementation of an equalizer in a sink device[23] that will be evaluated before starting the design of the VHDL module. The design approach for this module will be to first draw block diagrams and flow charts for the entire link training module. The block diagrams should contain all signals in the implementation, and show where they are connected.

The block diagrams and flow charts should be used as a template when writing the VHDL code for the implementation. When the VHDL code is complete the design will be tested through simulations using the Mentor Graphics ModelSim package. All sub-blocks will first be tested individually before they are connected together and tested as a complete module.

After testing through simulations, the VHDL implementation will be synthesized using the Altera Quartus Prime package, before being transferred into the Arria 10 FPGA. Because the DisplayPort IP block will not be available during simulations

in ModelSim, the rest of the functionality will have to be tested inside the FPGA. Altera SignalTap II will be used to monitor and verify the correct behavior of the different signals when the functionality is tested within the FPGA.

When the correct behavior of the design have been fully verified, the quality of the implementation will be tested by trying to set up a video link against different video sources. This should be one source from Nvidia, one from Intel and one from AMD. These sources will also be tested with cables of lengths between 0.8 and 10 meter.

The third and final part of the thesis will be to implement the Link Monitoring module. The purpose of this module is to monitor the video link, and detect if the quality of the link starts to degrade. If this situation occurs, the link monitoring module should try to tune the equalizer to prevent the video link from shutting down. The design approach for this module will be the same as for the link training module.

3.1 Equipment

A list of the equipment that will be used for testing, simulation and measurements for this thesis project is listen below.

- Barco Pulse mainboard [24]
- ModelSim - Altera starter edition [25]
- Quartus Prime 16 [26]
- UNIGRAF DPA-400 DisplayPort AUX-channel monitor [27]
- Quantum 980 Video source [28]
- DisplayPort cables of variable types and lengths

4
Implementation

The following chapter contains a description of the VHDL implementation that has been created during this thesis project. Design decisions, background research, block diagrams and flow charts will be presented for every component.

Figure 4.1 below shows a block diagram that illustrate the entire system. Each component will be explained in further detail later in this chapter.

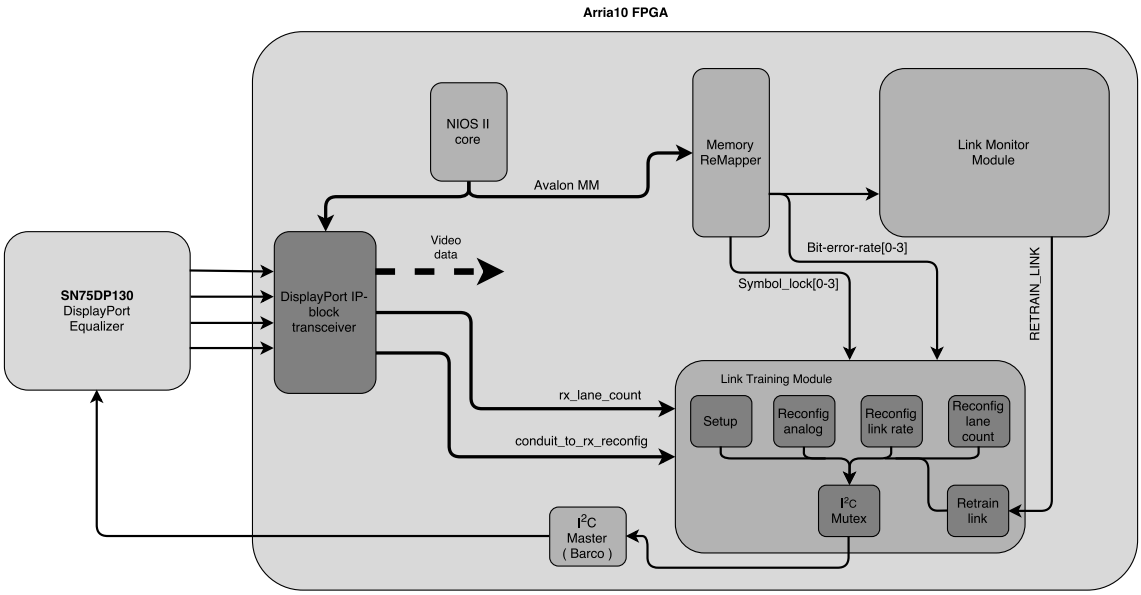


Figure 4.1: A block diagram of the entire system.

4.1 Link training in detail

When two devices are connected through a DisplayPort cable, the protocol runs a link training procedure. The purpose of the link training is to configure the link with optimal settings for voltage swing, pre-emphasis, lane speed and lane count for the coming session. The link training procedure is controlled from the source side of the link, in a master-slave configuration. The receiver, which is the slave, only responds to commands sent by the source [5].

Once a DisplayPort cable is connected, the source and sink device will communicate the supported link speed, lane count, maximum screen resolution as well as maximum frame rate using the AUX channel [5]. This information will be used to set the first test parameters for the first section of the link training procedure.

4.1.1 Clock Recovery

The first step in the link training procedure is clock recovery. The purpose of clock recovery is for the sink to recover the source clock from the data stream. Unlike HDMI, DisplayPort does not transfer any clock signal across the physical medium. The source and sink device each has a referance clock of 270 MHz, that gets upscaled using a PLL [5]. The phase and frequency of the sink PLL will be synchronized with the datastream during clock recovery. A flow chart of the clock recovery sequence during link training is presented in Figure 4.2.

The clock recovery procedure starts with the source transmitting a predetermined test pattern to the sink unit, with the lowest settings for voltage swing, pre-emphasis level and at the highest supported bit rate. This training pattern will be repeatedly transmitted for a delay set in a DPCD register. This time is between 100 μs and 16 ms, and is specified in the `TRAINING_AUX_RD_INTERVAL` register.

The source will then check via the AUX channel if the source and sink has managed to recover the clock. If the clock has not been recovered, the sink device can request new settings for voltage swing and pre-emphasis to try for the next iteration. The training pattern will then be re-transmitted for another 100 us and the source will again check if the sink has managed to synchronize the reference clocks.

This behavior will loop until the pre-emphasis and voltage swing have changed up to five times, or until the reference clocks have been synchronized. At the fifth iteration, if the clocks are still not synchronized, the source will reduce the lane speed, and restart from the lowest level of pre-emphasis and voltage swing, and repeat for five new iterations.

The source will keep trying to synchronize the clocks until it succeeds, or until it failed with max voltage swing and pre-emphasis settings on the slowest lane speed. As soon as the clocks get synchronized, the source will continue to the next step in the link training procedure.

4.1.2 Channel Equalization

Following clock recovery comes channel equalization. The behavior of this stage of the link training is very similar to that of the clock recovery stage. A flowchart of the channel equalization sequence during link training is presented in Figure 4.3.

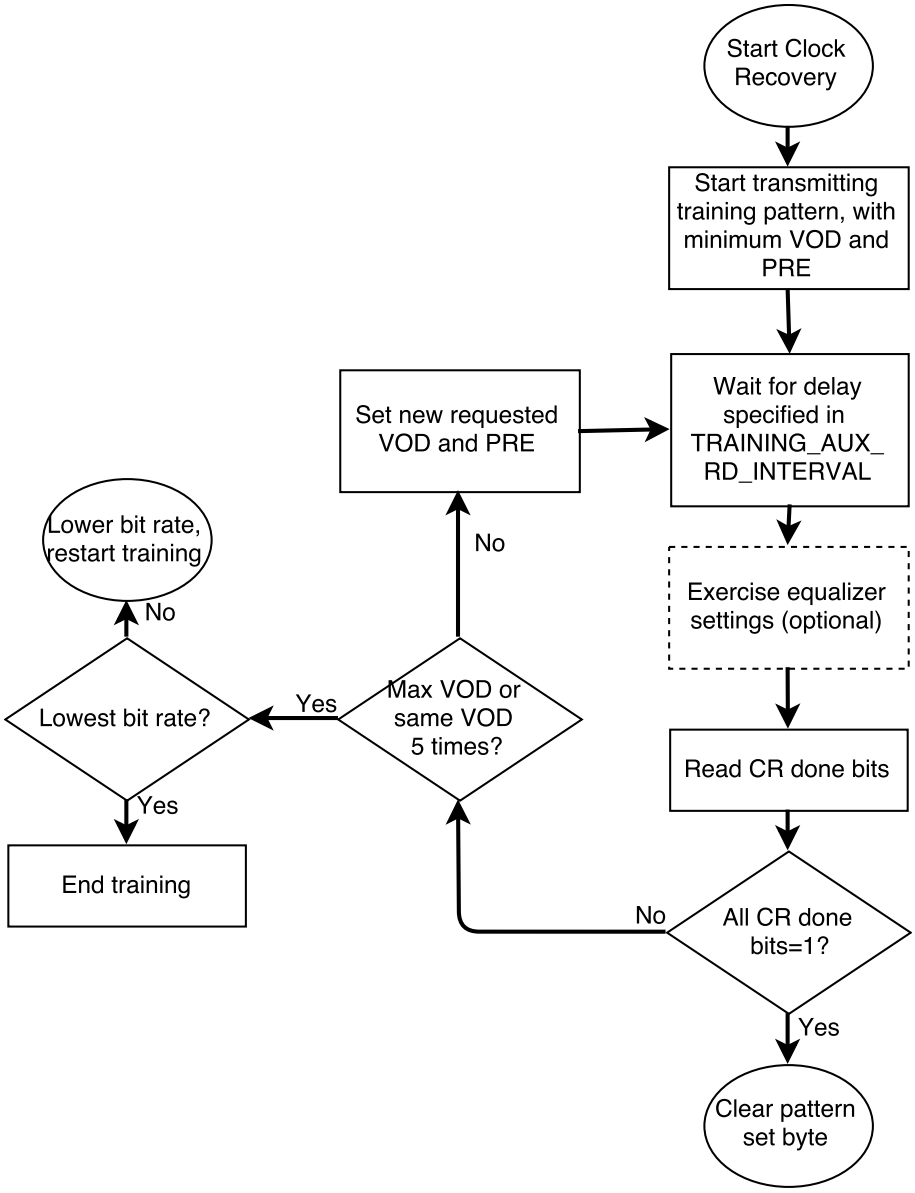


Figure 4.2: Flow chart for the clock recovery sequence during link training.

The channel equalization procedure starts out with the source transmitting one out of two pre-defined training patters, with the same settings used when clock recovery successfully managed to synchronize the clocks. This training pattern will be repeatedly transmitted for the time specified in the DPCD register `TRAINING_AUX_RD_INTERVAL`. After the specified delay, the source will check if the reference clocks are still synchronized as well as checking if the source managed to recognize the transmitted training pattern. If the pattern was not recognized, the source will read new suggested settings for pre-emphasis and voltage swing through the AUX channel and re-transmit the training pattern. This will loop until the source manages to recognize the training pattern, or it has reached five itterations.

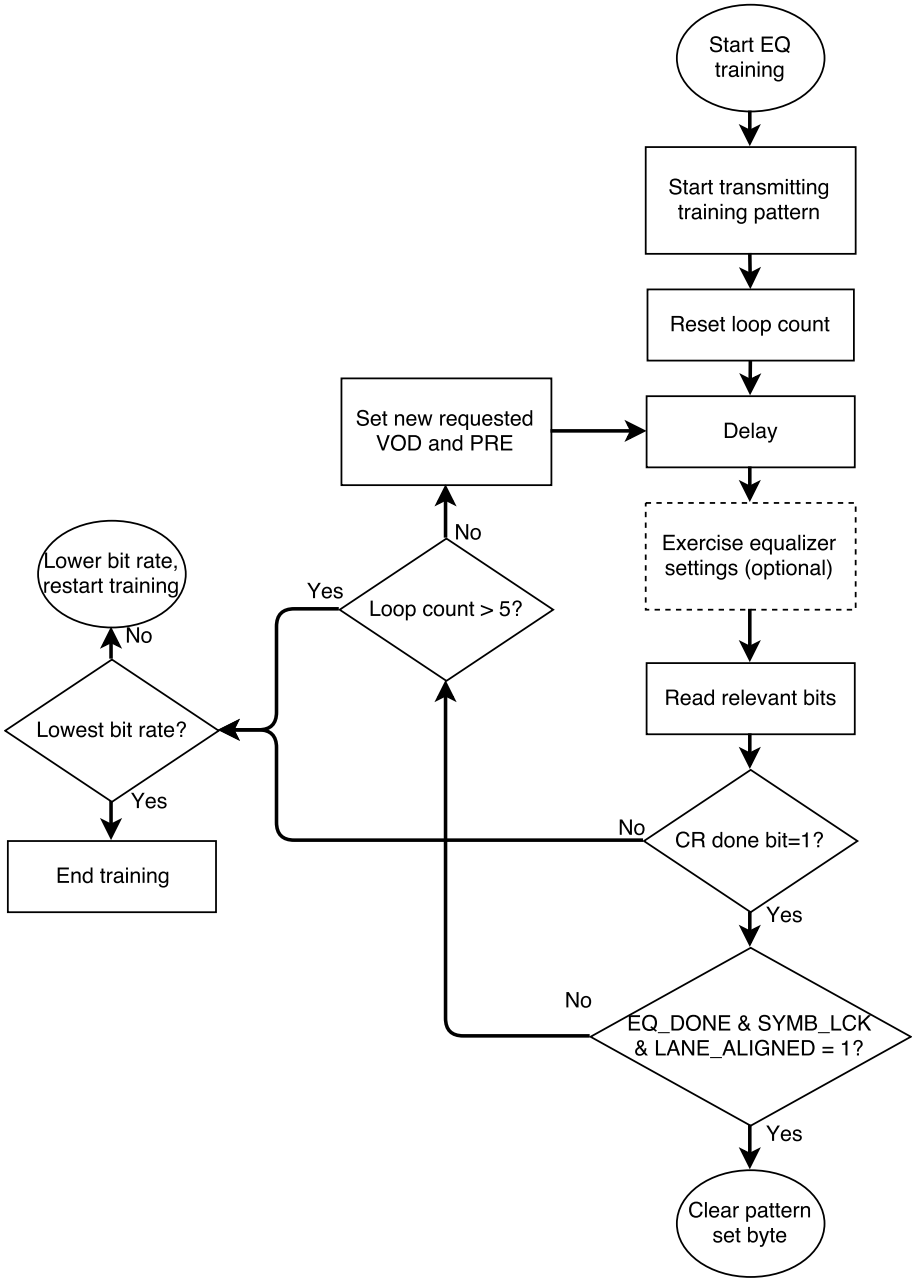


Figure 4.3: Flowchart for the channel equalization sequence during link training.

4.2 Link training module

The purpose of the link training module is to configure the equalizer during the link training procedure. This section covers the design process of this module. Some of the background research is first presented together with the general design decisions , before the design of each submodule is described in more detail.

4.2.1 Before starting the design

Some research went in prior to starting the design of the link training module. The first step was to evaluate the suggested application note from Texas Instruments on implementations of a DisplayPort equalizers in a sink unit.

4.2.1.1 Texas Instruments equalizer application note

Texas Instruments has published an application note on implementation of a DisplayPort equalizer in sink units, using the SN75DP130. This application note was evaluated in order to see if the suggested implementation was suitable for the Barco pulse platform.

The application note suggests that a constant higher level of equalization should be used. This way of implementing the equalizer is deemed not suitable, because in the case where a lower level of equalization is needed, the link training would fail.

For the parameters of the output re-driver stage, the application note suggests that these should be set to constant values. Because the signal path between the re-driver and the FPGA are non-changing, the re-driver settings can be tuned for this particular signal path. This also means that less parameters needs to be changed in real time during the link training procedure, making the implementation of the link training module less complicated. This suggestion is therefore deemed suitable.

4.2.1.2 Transfer times

Because the parameters of the SN75DP130 would have to be configured in real time during link training, the transfer times over the I^2C bus was further investigated.

The SN75DP130 supports a maximum I^2C bus speed of 100 kHz. To configure the equalization levels on all inputs, eight bytes would have to be written to the equalizer. Adding the address byte, start-, stop- and acknowledgment-bits, this means that a total of 83 bits would need to be written in order to configure the equalization levels. Write time for the equalizer is calculated by the following formula:

$$Writetime = \frac{83\ bits}{100000\ Hz} = 830\mu s \tag{4.1}$$

The datasheet for the equalizer does not specify any setup time for a new equalizer level, but a setup time of 170 μs was added to the calculation to introduce some margin. This means that at least 1 ms is required to change the input equalization levels.

4.2.1.3 DisplayPort protocols deviations

During the early research phases of this thesis project, some deviations on the DisplayPort protocol was noticed on some video sources. An AUX-channel analyzer

from UNIGRAPH[27] was used to inspect some of the traffic on the AUX-channel between the Barco Pulse platform and four different video sources. The tested sources was the Quantum 980 video source[28], Nvidia Quatro 2200, Nvidia Quatro 1000M and a Nvidia Quatro 5000 graphics cards. While monitoring this AUX traffic, a couple of deviations from the DisplayPort protocol was observed. Firstly, some of the video sources did not follow the settings for voltage swing and pre-emphasis suggested by the sink device, as specified by the DisplayPort protocol. See Figure 4.2 and Figure 4.3 for reference. This means that the link training module could not rely on affecting the output settings of the source device, as this part of the protocol was not supported by all image sources.

Secondly, some video sources did not follow the delay specified in the `TRAINING_AUX_RD_INTERVAL` register, which was set to 16 ms for these tests. The video source should start transmitting a training pattern, then wait for the delay specified in this register while the sink device has time to set up the correct settings for the link. See Figure 4.2 and Figure 4.3 for reference. Some of these sources only used the standard delay which is 100 μs for clock recovery, and 400 μs for channel equalization. As shown by the calculations in Section 4.2.1.2, more time than 100 μs or 400 μs will be needed in order to re-configure the equalization levels during runtime. This means that if the source unit does not follow the delay specified in the `TRAINING_AUX_RD_INTERVAL` register, the link training module would not be able to reconfigure the equalizer during runtime.

This issue was resolved by using a unit that was known to follow this delay during testing of the link training module, more specifically the Nvidia Quatro 5000 graphics card. If there was time left at the end of the project, an extra module should be added to the design that disabled the link training module if the video source did not follow this delay.

4.2.2 System overview

Initially, there were five parameters available to be changed in order to achieve optimal transmission settings for the link. These were the pre-emphasis and voltage swing at the output of the source transmitter, the equalization level as well as pre-emphasis and voltage swing at the SN75DP130. Based on the research of the parameters before starting the design, three of these parameters are no longer viable. Firstly, the implementation may not rely on affecting the output parameters of the source device, as this is not supported by all video sources. Secondly, the output parameters of the SN75DP130 does not have to be changed, as these may be set to constant values. This leaves only the input equalization levels on the SN75DP130 that needs to be changed in real time during the link training protocol. The lane count and link speed of the equalizer will also have to be configured. These settings are not very critical in form of timing, which means they have a lower priority compared to the equalization level.

The module will reconfigure different parts of the equalizer based on different inputs from other VHDL blocks in the system. The link training module consist of several smaller blocks that configures different parts of the equalizer. All modules, and how they are connected together, are shown in Figure 4.4.

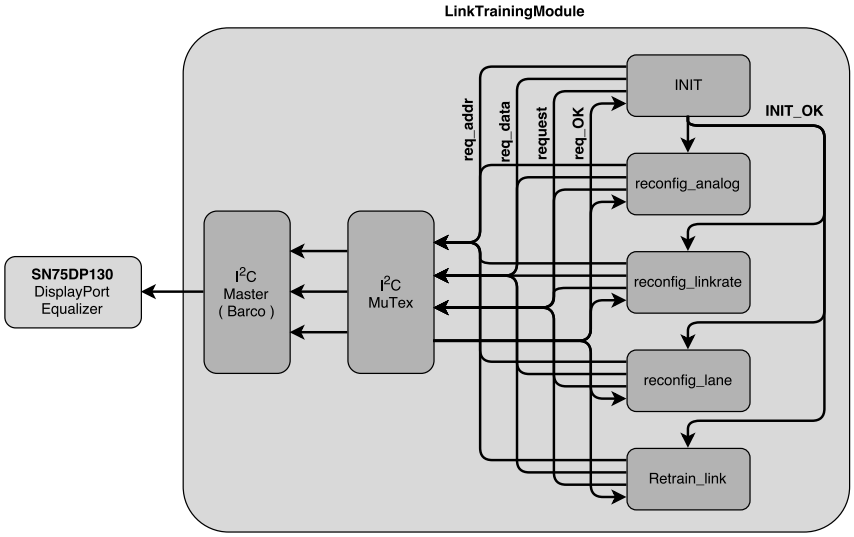


Figure 4.4: Blockdiagram of the link training module

4.2.3 I²C Master

The purpose of the I²C Master is to interface the I²C bus with the implemented logic inside the FPGA. The I²C master is an IP-block developed by Barco. How the input signals of the I²C master are used is described in more detail in Section 4.2.4. Figure 4.5 below illustrates the inputs and outputs of the I²C-master.

4.2.3.1 Input and output signals

The I²C master uses two inout VHDL type signals, `SCL` and `SDA` which is connected directly to two of the pins on the FPGA. These are the clock and data signals for the I²C interface, which is further explained in Section 2.4.

The input signals of the component is listed below.

- `clk` is the main clock input. This clock is used to drive the internal state machine as well as generate the clock for the I²C bus.

- `reset` is the block reset input. When this signal is asserted, any ongoing transfer will be stopped, and the component will return to an idle state.

- `I2C_Address` is a 7-bit vector containing the address of the target I²C component connected to the I²C bus.

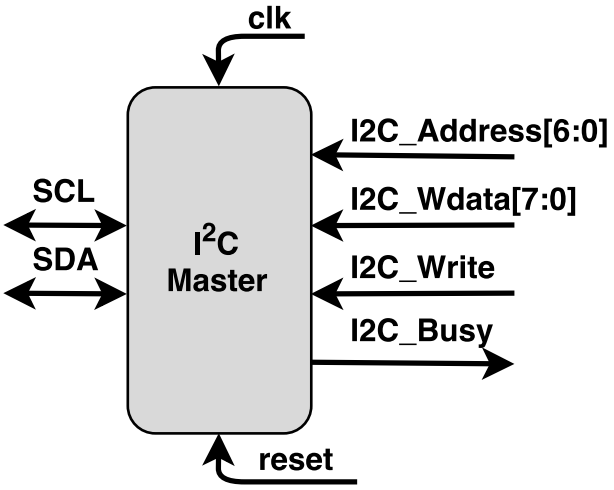


Figure 4.5: Inputs and outputs from the I²C Master component

- `I2C_WData` is an 8-bit vector that holds the data to be written to the target address.

- `I2C_Write` is a single bit signal that starts a write transfer from the I²C master to the target component connected to the bus once asserted. This signal can be kept asserted to enable burst writes to the I²C component.

- `I2C_RRequest` is used to request a read from the specified I²C address. This signal is not used for this implementation.

There are a number of output signals available from the I²C master block.

- `I2C_Busy` is a signal used by the I²C master to signal that a transfer is in progress. If `I2C_Write` is kept asserted, `I2C_Busy` will be deasserted for one clock cycle to signal that new data has been read by the I²C master for burst transfer to the I²C component.

- `I2C_RData` is an 8-bit output vector that contains the data read through the I²C bus after a read operation. This vector is not used in this implementation.

This component also has two generic input integers called `ClockFrequency` and `I2C_ClockFrequency`, that holds in input frequency of the input signal `clk`, as well as the desired clock frequency for the I²C bus.

4.2.4 I²C bus handler

The purpose of the I²C bus handler is to process all communication between the different blocks in the link training module and the external equalizer. The bus handler implements a priority system with mutual exclusion of the I²C bus among

all the different blocks in the link training module. This functionality is required because all the different blocks in the link training module trigger off different signals, and the I^{2}C master does not contain any mutual exclusion or support for receiving data from several blocks. Preemption is not supported by the I^{2}C bus handler. A block diagram of the component can be seen in Figure 4.6.

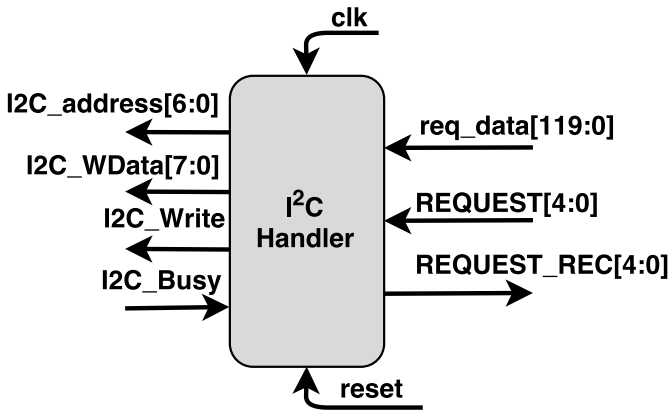


Figure 4.6: Inputs and outputs for the I2C handler block

4.2.4.1 Input and output signals

There are four output signals coming from the I^{2}C bus handler block.

- `I2C_address` is a 7-bit vector that contains the equalizer I^{2}C bus address. This address vector is set to a constant value of 0b0101100.
- `I2C_WData` is a 8-bit vector which is connected to the data input pins on the I^{2}C Master block. This vector holds the internal equalizer register address as well as the data to write to each respective register.
- `I2C_Write` is a single bit signal that is asserted to start a new I^{2}C write. This signal is kept asserted until all data has been transferred to the equalizer. This signal is explained in more detail in Section 4.2.3.
- `REQUEST_REC` is a 5-bit vector, where each bit is connected to different blocks in the link training module. This signal is asserted by the bus handler to signal the requesting block that all information needed to start communication with the equalizer has been received.

There are also five different inputs to the bus handler.

- `clk` is the 100 MHz clock input for the block.

- `reset` is the block synchronous reset signal. When reset is asserted, all internal signals are set to zero, and the state machine returns to its idle state.

- `REQ_DATA` is a 119-bit vector that holds the input data to the bus handler. The bus is set up to allow parallel transfers of bytes from other blocks to the bus handler.

- `REQUEST` is a 5-bit vector, where each bit is connected to different blocks in the link training module. When one of these bits are asserted, the bus handler will store the data available on `REQ_DATA` in an internal array, and start a transfer to the equalizer. This signal is used to set the priority of the different blocks, where bit 0 has the highest priority.

- `I2C_Busy` is an input signal coming from the I^2C master block. This signal is used to synchronize transfers of data between the bus handler and the bus master to enable burst transfers.

4.2.4.2 Block functionality

A flowchart of the I^2C bus handler can be seen in Figure 4.7.

The bus handler is implemented using a state machine, that will start when at least one of the bits in the `REQUEST` input vector is asserted. Each asserted bit in this vector indicates that one of the other blocks in the link training module wants to write to the equalizer. If two or more bits are asserted, the block will give priority to the block represented by the least significant bit that is asserted. As an example; if both bit 1 and bit 4 is asserted, priority will be given to the block represented by bit 1. A transfer example is presented in Figure 4.8.

Based on which `REQUEST`-bit that is asserted, the bus handler will store data from `REQ_DATA` into an array called `SENDARRAY`, capable of storing up to nine bytes (one equalizer register address byte and up to eight data bytes). `MessageCount` is also updated with the number of bytes that is stored in `SENDARRAY`. This variable is later used to determine if all bytes has been transferred to the equalizer.

As soon as all relevant information is stored in the internal registers of the bus handler, one of the bits in the `REQUEST_REC` vector will be asserted in order to signal to the requesting block that the data has been received by the bus handler. As an example; if bit 1 in the `REQUEST` vector triggered the initial transfer, bit 1 in the `REQUEST_REC` vector will be triggered to signal that the data has been received by the bus handler. This signal will stay asserted for two clock pulses, which is enough time for the requesting block to react to the signal, but not too long so that the block can trigger off the same `REQUEST_REC` signal twice.

When all relevant information is stored in the internal array and the correct `REQUEST_REC` bit has been asserted, the transfer to the equalizer will start. The transfer is

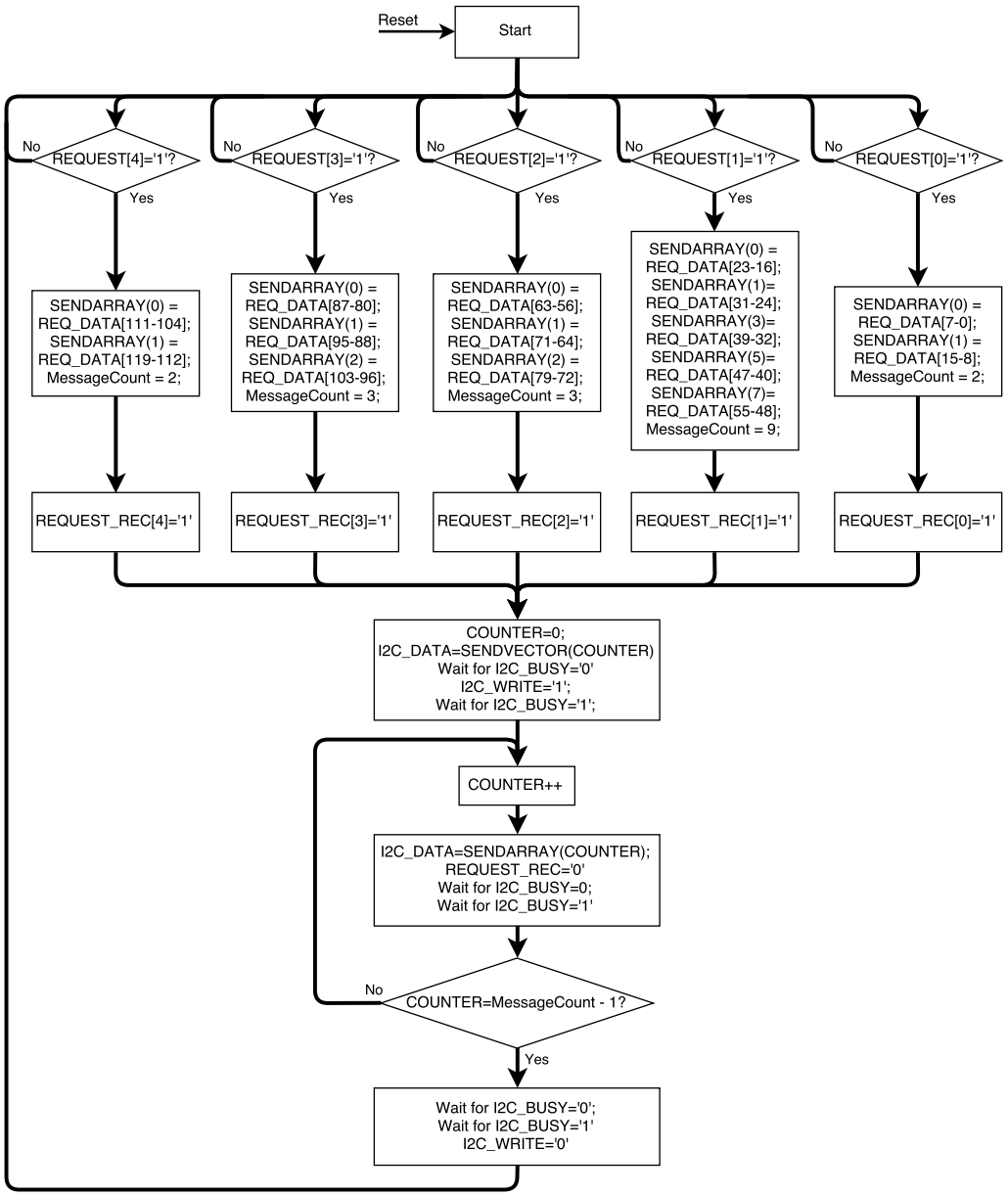


Figure 4.7: Flowchart for the I^{2}C bus handler

started by setting an internal counter to zero, as well as outputting the first byte in `SENDARRAY` to the I^{2}C master through the `I2C_Data` port. Before `I2C_Write` is asserted, the bus handler will wait for `I2C_Busy` to be deasserted, in case a previous transfer has not yet finished.

When `I2C_Write` has been asserted, the bus handler will wait for `I2C_Busy` to be asserted, signaling that the data has been received and a transfer has been started by the I^{2}C master. The next byte will then be output to the `I2C_Data` port. As long as `I2C_Write` is kept asserted, the I^{2}C master will continuously transmit the bytes available on the `I2C_Data` port. The I^{2}C master signals to the bus handler that the transfer of the next byte has started by deasserting `I2C_Busy` for one clock cycle.

These steps will keep looping until the internal counter reaches the value of `MessageCount` −1, in which case `I2C_Write` will be deasserted and the bus handler will prepare for receiving the next request from one of the blocks in the link training module.

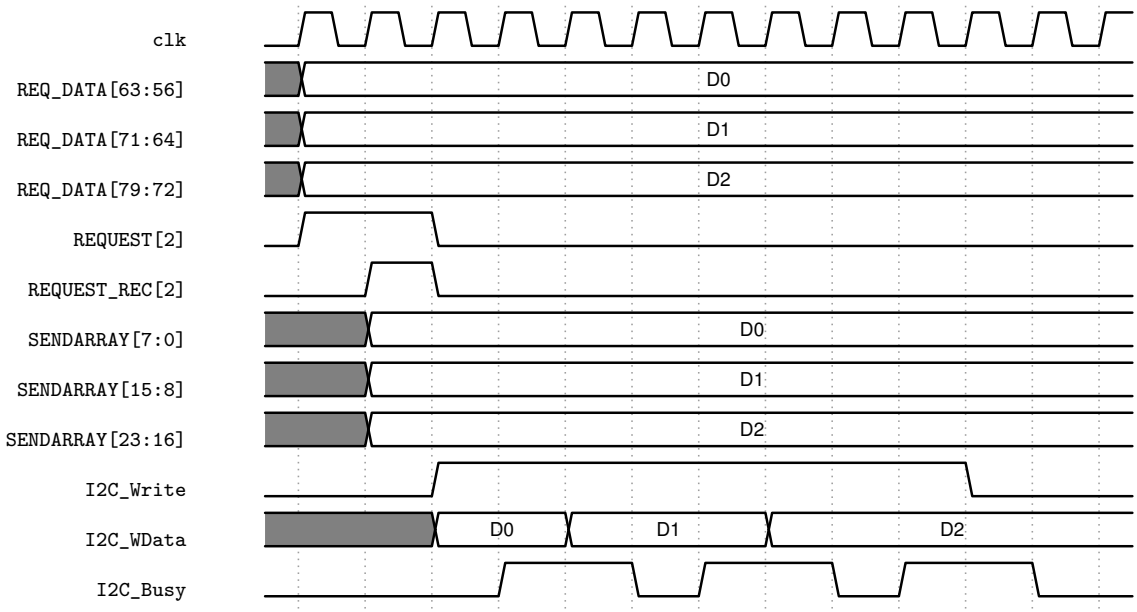


Figure 4.8: Timing diagram for changing the equalizer link rate.

4.2.5 Equalizer initiation

The purpose of the initialization block is to set up the equalizer with a set of predetermined settings as soon as a reset occurs. The settings that was set during initialization was maximum input equalization, maximum link speed and highest lane count.

A flowchart for the equalizer initiation module can be seen in Figure 4.9. The block runs when reset is asserted and will send a series of bytes to the equalizer via the I^2C interface. These bytes are stored in two separate 8-bit arrays with a generic amount of elements. One of the arrays holds the address for the different target registers, and the other vector holds the data to be written to each respective register.

The block will start with setting a counter to zero and loop until it reaches the total number of messages to be sent to the equalizer. This variable is called `I2C_Parameter_count` and is also used to set the number of elements in the I^2C data and address arrays mentioned above. When all data has been transferred to the equalizer, a flag called `INIT_OK` will be asserted, signaling for all the other blocks in the link training module that the equalizer setup is complete.

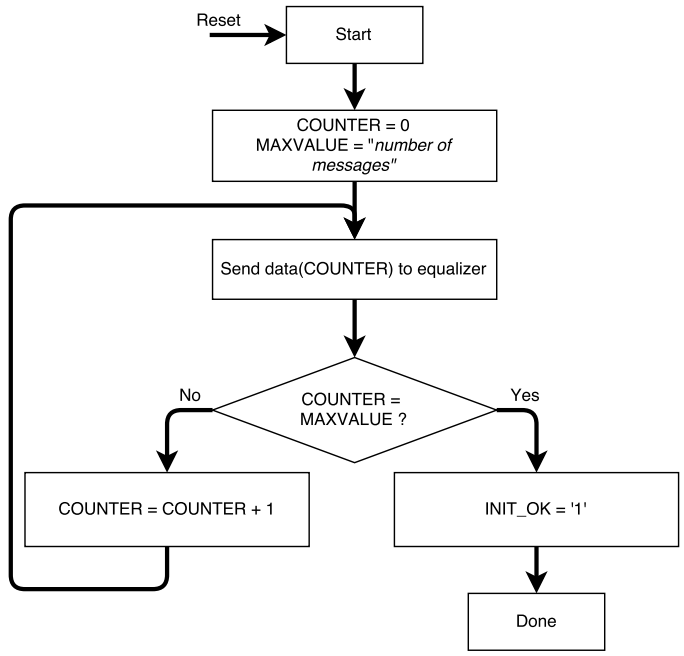


Figure 4.9: Flowchart of the equalizer initializer.

4.2.6 Equalizer gain reconfiguration

The equalizer gain reconfiguration block monitors the DisplayPort link training procedure and reconfigures the external equalizer with appropriate settings for the current link conditions. This is one of the primary components of the thesis project, so a lot of thought went in to the design process of this part.

The equalizer gain configuration block is illustrated in Figure 4.10.

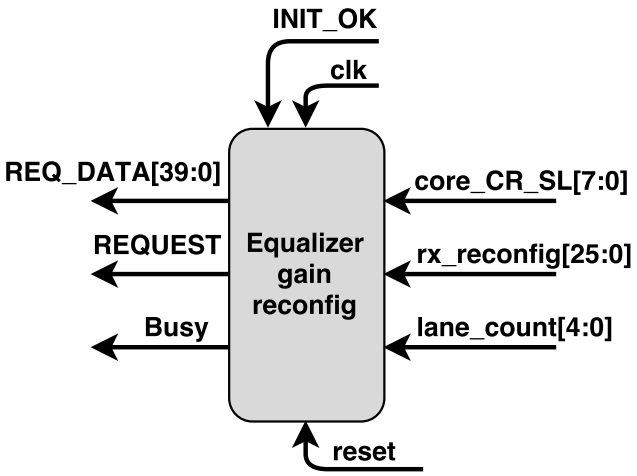


Figure 4.10: Illustration of the equalizer gain reconfiguration block.

4.2.6.1 Input and output signals

The equalizer gain reconfiguration block uses the following input signals.

- `clk` is the design 100 MHz reference clock.
- `reset` is the synchronous reset input. When a reset occurs, the design will return to its start state as well as return all internal variables to zero.
- `INIT_OK` is a signal coming from the equalizer initiation module. This signal will be asserted as soon as the equalizer has been programmed with a set of predetermined settings after a reset has occurred.
- `REQUEST_REC` is a signal coming from the I^2C bus handler. This signal is asserted once all data needed for a I^2C transfer to the equalizer has been received by the bus handler. See the I^2C bus handler section for more info regarding this signal.
- `core_CR_SL` is an 8-bit vector coming from the DisplayPort IP core. The four least significant bits in this vector holds the clock recovery status for each lane, while the four most significant bits holds the symbol lock status for each lane. This information is used to configure the link during link training.
- `rx_reconfig` is a 26-bit vector coming from the DisplayPort IP core. This input signal is explained in further detail later in this chapter.
- `lane_count` is a 5 bit vector coming from the DisplayPort IP core. This vector holds information regarding how many lanes that are currently active in the video link.

The block output signals are listed below.

- `REQ_DATA` is a 40-bit output vector that is connected to the I^2C bus handler. This vector holds the data bytes used to set gain for the different video lanes. One byte is used for addressing the first gain register in the equalizer, the remaining 4 bytes holds gain data for each lane.
- `REQUEST` is used to signal the I^2C bus handler that new data is available on the `REQ_DATA` vector that needs to be transferred to the equalizer.

4.2.6.2 Block functionality

A flowchart for the equalizer gain reconfiguration block can be seen in Figure 4.11.

The equalizer gain reconfiguration block is implemented using a state machine. Once a reset occurs the state machine will go to its start state, where it will wait for `INIT_OK` to be asserted. Once `INIT_OK` has been asserted, it will go to its idle state,

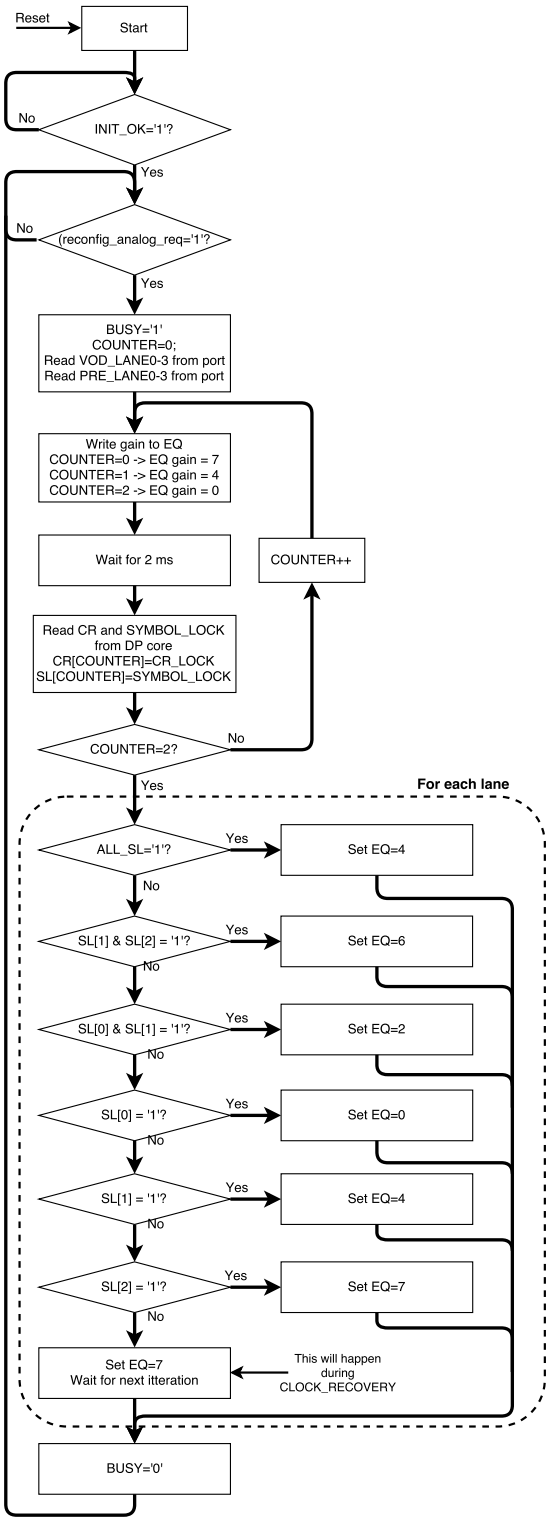


Figure 4.11: Flowchart for the equalizer gain reconfiguration block

where it will wait for a trigger signal from the DisplayPort IP core to signal that a new link setting has been set, and the equalizer needs to be reconfigured. This trigger is located within the `rx_reconfig` vector, called `reconfig_analog`. All contents of the `rx_reconfig` vector is presented in Table 4.1. When the DisplayPort

IP core asserts `reconfig_analog`, this indicates that the video source has changed its pre-emphasis and voltage-swing output settings. This signal will cause the state machine to start the gain reconfiguration sequence.

Table 4.1: Data contents of the `rx_reconfig` vector, based on current lane count.

rx__reconfig			
Signal	**4 lanes**	**2 lanes**	**1 lane**
`reconfig_linkrate`	`rx_reconfig`[0]	`rx_reconfig`[]	`rx_reconfig`[]
`link_rate`[7:0]	`rx_reconfig`[8:1]	`rx_reconfig`[8:1]	`rx_reconfig`[8:1]
`reconfig_analog`	`rx_reconfig`[9]	`rx_reconfig`[9]	`rx_reconfig`[9]
`vod_lane0`[1:0]	`rx_reconfig`[11:10]	`rx_reconfig`[11:10]	`rx_reconfig`[11:10]
`vod_lane1`[1:0]	`rx_reconfig`[13:12]	`rx_reconfig`[13:12]	Not used
`vod_lane2`[1:0]	`rx_reconfig`[15:14]	Not used	Not used
`vod_lane3`[1:0]	`rx_reconfig`[17:16]	Not used	Not used
`pre_lane0`[1:0]	`rx_reconfig`[19:18]	`rx_reconfig`[15:14]	`rx_reconfig`[13:12]
`pre_lane1`[1:0]	`rx_reconfig`[21:20]	`rx_reconfig`[17:16]	Not used
`pre_lane2`[1:0]	`rx_reconfig`[23:22]	Not used	Not used
`pre_lane3`[1:0]	`rx_reconfig`[25:24]	Not used	Not used

The gain reconfiguration sequence will have up to 16 ms, as set by the `TRAINING_AUX_RD_INTERVAL` DPCD register, to find a suitable equalizer setting after asserting the `reconfig_analog` signal.

4.2.7 Equalizer link rate reconfiguration

The purpose of the link reconfiguration block is to change the equalizer link rate, to follow the same speed of the video link. This is implemented by a state machine.

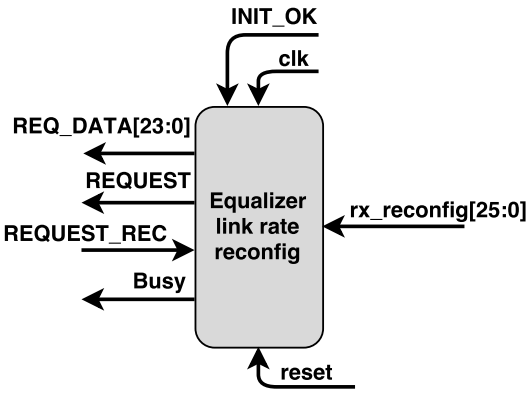


Figure 4.12: Inputs and outputs of the equalizer link rate reconfiguration block.

4.2.7.1 Input and output signals

The block uses the following input signals.

- `clk` is the 100 MHz input clock for this block.
- `reset` is the synchronous reset input.
- `INIT_OK` is the signal coming from the equalizer initiation block, signaling that initiation of the equalizer has finished.
- `REQUEST_REC` is a signal coming from the I^2C bus handler signaling that data has been received and transmission to the equalizer has been started.
- `rx_reconfig` is a 26-bit vector coming from the DisplayPort IP core. This vector is presented in Table 4.1.

The link rate reconfiguration block uses the following output signals.

- `REQ_DATA` is a 24-bit output vector that is connected to the I^2C bus handler. This vector holds the data bytes used to set the link rate for the equalizer.
- `REQUEST` is used to signal the I^2C bus handler that new data is available on the `REQ_DATA` vector that needs to be transferred to the equalizer.

4.2.7.2 Block functionality

A flowchart for the link rate reconfiguration block can be seen in Figure 4.13.

Once a reset occurs, the link rate reconfiguration block will return to its start state waiting for `INIT_OK` to be asserted. As soon as this happens, the state machine will jump to its idle state, waiting for a change in the video link rate.

A change in the video link rate is signaled by the DisplayPort IP core asserting a bit in the `rx_reconfig` vector, earlier presented in Table 4.1, called `reconfig_linkrate`. This will trigger the link rate reconfiguration module to send the new link rate to the equalizer through the I^2C bus handler, using the `REQ_DATA` and `REQUEST` signals.

4.2.8 Equalizer lane count reconfiguration

The purpose of the lane count reconfiguration block is to make sure that the amount of active lanes on the equalizer matches that of the video link.

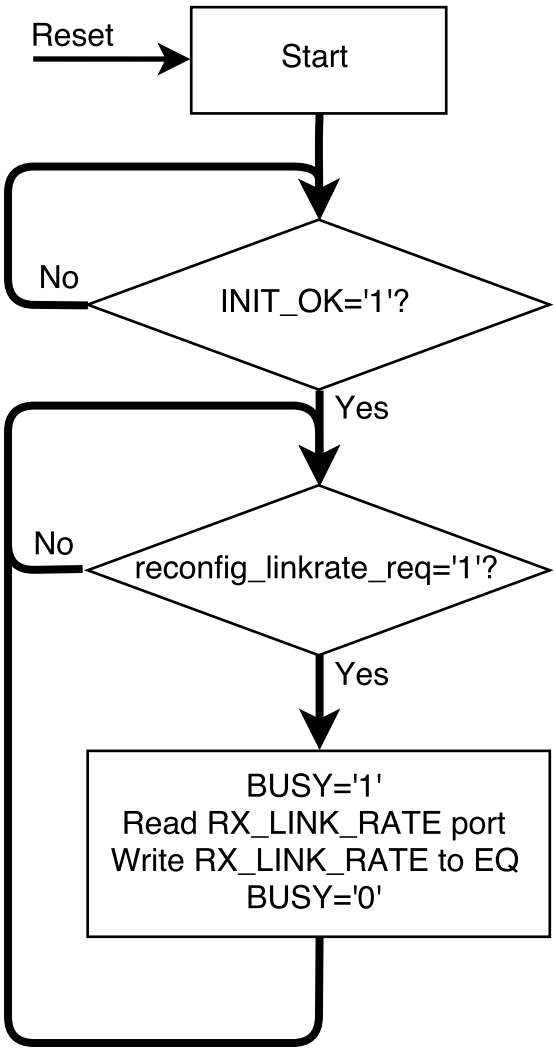


Figure 4.13: Flowchart for the equalizer link rate reconfiguration block.

4.2.8.1 Input and output signals

The input signals for the lane count reconfiguration block can be seen in the list below.

- `clk` is the 100 MHz system reference clock.
- `reset` is the block synchronous reset input.
- `INIT_OK` is a signal coming from the equalizer initiation module that is asserted when the equalizer is programmed with a set of predetermined settings after a reset occurs.
- `REQUEST_REC` is a signal used by the I^2C bus handler, signaling that data has been received and a transfer to the equalizer has been started.

- `rx_lane_count` is a 5-bit vector coming from the DisplayPort IP core. This signal holds information about how many lanes that is currently active over the link.

The lane count reconfiguration block only uses output signals that are connected to the I^2C bus handler.

- `REQ_DATA` holds data to be transferred to the equalizer over the I^2C bus.
- `REQUEST` is asserted to let the bus handler know that there are new data ready to be transferred to the equalizer via the I^2C bus.

4.2.8.2 Block functionality

A flowchart for the lane count reconfiguration state machine can be seen in Figure 4.14.

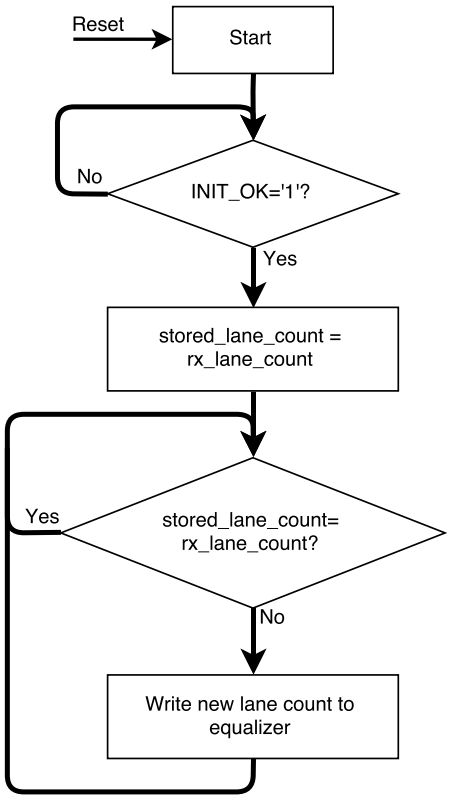


Figure 4.14: Flowchart for the lane count reconfiguration block.

Since there is no trigger signal dedicated for lane count reconfiguration, like the previously mentioned `reconfig_linkrate` and `reconfig_analog` used in the other modules, a slightly different approach was used for this block. This block monitors the output signal from the DisplayPort IP core called `rx_lane_count`. Once this

signal changes value, the new lane count will be transferred to the equalizer.

4.3 Nios II core

The Nios II core[29] is a soft processor provided by Altera as an IP block to be used within the FPGA. In this project, the Nios II core is used to acquire data from the DisplayPort IP core internal registers via the Avalon MM interface, which is not available on any of the IP core VHDL signal ports. For more information regarding the Avalon MM interface, see Section 2.5. The implementation in this core is written in the C programming language.

Prior to implementing the link training module, the Nios soft processor was already used to run a monitoring function for the DisplayPort IP core. The functionality of the monitoring function is unknown, but the documentation accompanying the DisplayPort IP core states that the monitoring function is run at least once every 50 ms. This requirement was taken into account when additional functionality was added for the link monitoring module.

A flowchart of the Nios II program can be seen in Figure 4.15.

After a reset, the Nios core runs a setup function for the DisplayPort IP core. The behavior of the setup function is unknown. When setup is complete, the program enters an infinite while-loop, where the main program code is placed. The DisplayPort IP core monitor is one of the functions that is called in this while-loop.

The additional code that is added is the reading of the internal DisplayPort IP core registers. This is done by using a function provided in one of Altera's libraries for the Nios core called `IORD`. The data exported from the DisplayPort IP core through the Avalon MM interface are clock recovery information, symbol lock information and bit error counters for all lanes.

To export the data to the VHDL domain, a separate VHDL component is written, called the MemoryReMapper. This component is further explained in Section 4.4. The data is written from the Nios processor to the `MemoryReMapper` using the `IOWR` function, defined in the same library as the previously mentioned `IORD` function.

4.4 MemoryReMapper

The purpose of the MemoryReMapper component is to act as a bridge between the Avalon MM communication bus and the rest of the VHDL-implementation.

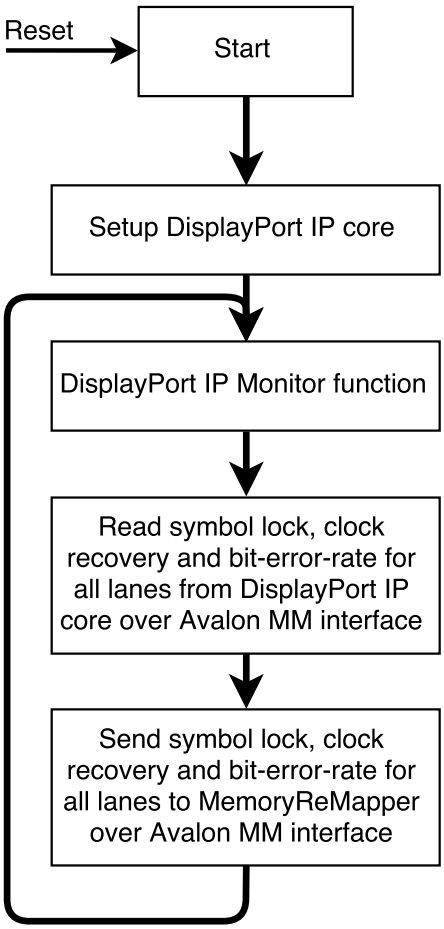


Figure 4.15: Flowchart of the Nios II soft processor program.

4.4.1 Input and output signals

The input signals for the MemoryReMapper can be seen below.

- `avalon_mm_clk` is the Avalon MM bus clock. This signals is used as the block main clock input.
- `avalon_mm_address` holds the address for the register where the Nios core wants to write.
- `avalon_mm_read` will be asserted if the Nios core wants to read data from the MemoryReMapper. This signal is used in this implementation.
- `avalon_mm_write` will be asserted if the Nios core want to write data to the specified address.
- `avalon_mm_writedata` holds that data that is to be written to the MemoryReMapper.

The remapper outputs the data received on the Avalon to the following output pins.

- `dp_sink_CR_SL` holds the clock recovery and symbol lock data for each lane.
- `dp_sink_BER_0_1` holds the bit error counters for lane 0 and lane 1.
- `dp_sink_BER_2_3` holds the bit error counters for lane 2 and lane 3.

4.4.2 Block functionality

The MemoryReMapper will monitor the input address from the Avalon MM bus. When the address is within the memory area dedicated to the MemoryReMapper and the Avalon MM write signal is asserted, data will be read from the Avalon data bus and output to the correct vector. This behavior is demonstrated in Figure 4.16.

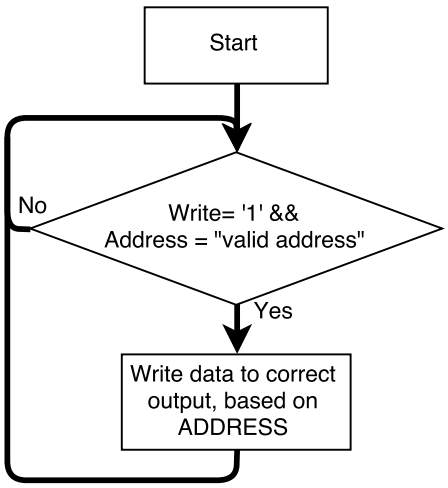


Figure 4.16: Flowchart for the memory remapper.

4.5 Link monitor module

The second implementation of this project is the link monitoring module. The purpose of this module is to monitor the quality of the video link during runtime and prevent the link from shutting down. The initial idea for this module was to monitor the bit error rate for all lanes on the DisplayPort communication link. If any of the error values started to increase in value due to some unknown reason, the goal was to try to reconfigure the equalizer to compensate for the added errors, before the link was shut down.

4.5.1 Before starting the design

A set of tests was set up before the design of this component was started. The goal was to see how the the link reacted to bit errors. It was quickly discovered that the DisplayPort IP core took action to correct for the bit errors before the bit-error registers was updated. This was often in the form of shutting down the link. There was no way to control this behavior of the DisplayPort IP core.

Because of the delayed update of the bit-error rate registers, combined with the slow bus speed of the I^2C interface, there was no time available to reconfigure the equalizer in time to compensate for the bit errors before the DisplayPort IP core shut down the video link. Because of this, the link monitor module was not implemented.

5
Results and discussion

This chapters covers some of the main results from this project, as well as some discussion and future work that can be based off this project.

5.1 Design verification

Every submodule of the link training module was first tested and verified individually. The submodules were then connected together and simulated as an entire unit. The link training module was finally tested within the FPGA together with the DisplayPort IP core as soon as the simulations showed correct behavior.
Figure 5.1 below shows the I^2C bus activity during a single iteration during link training. In this situation, the link training module first sets the input equalization on all lanes to 7. After a short delay, the equalization levels is set to 5. After another delay, the equalization level is set to 0. Data is collected by the link training module for these three scenarios and evaluated. When the optimal equalization level has been found, it is written to the equalizer.

Figure 5.2 shows the I^2C bus activity for the equalizer during the entire link training procedure. The two first data iterations, marked as 1, shows the clock recovery phase. Training at PRE and VOD level 0 is first unsuccessful, which means that the source has to increase VOD level to 1 in order to synchronize clocks. The link training procedure then continues with channel equalization, which finished after the first iteration. This is marked as 2 in Figure 5.2.

The design was meant to be tested with an Intel, AMD and Nvidia source. There were however no AMD or Intel sources available that followed the delay specified in the `TRAINING_AUX_RD_INTERVAL` register. Therefore, these sources has not been tested. Testing with a Nvidia source showed that the implementation was compatible. The implementation is expected to also be compatible with sources from AMD and Intel.

5.1.1 Timing analysis

When correct functionality had been verified for the created implementation, a timing analysis was run for the design. One of the requirements for the design was to

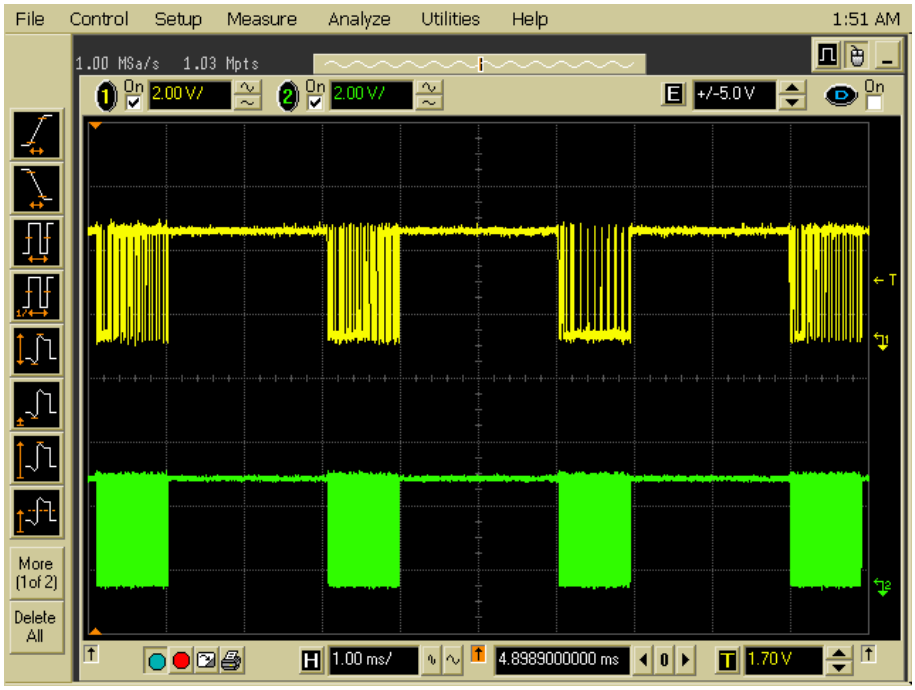


Figure 5.1: Oscilloscope plot showing the I^{2}C bus activity during a single iteration of link training during clock recovery. Bottom signal is the clock signal, while the top signal is the data.

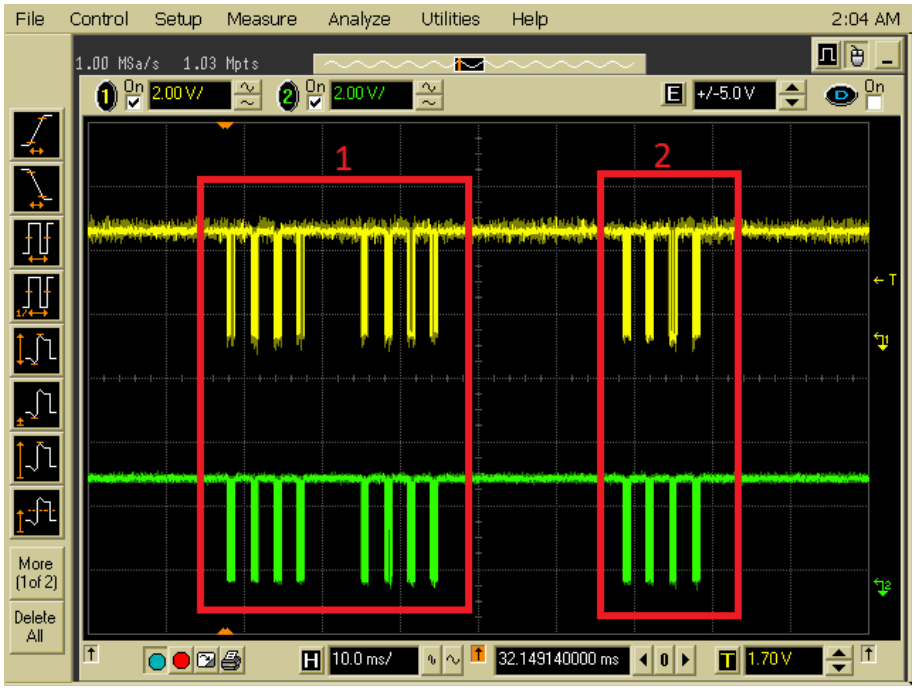


Figure 5.2: Oscilloscope plot showing the I^{2}C bus activity during link training. Bottom signal is the clock signal, while the top signal is the data.

not interfere with any of the other VHDL-blocks inside the FPGA. Timing analysis was run on the FPGA both with and without the VHDL-implementation created

during this project. The analysis showed that the critical path for the entire design was unchanged when the project implementation was present.

The maximum frequency of the project VHDL implementation was also investigated. Table 5.1 shows the timing slack of the design simulated with different clock frequencies. It can be seen that the design has a large timing slack at the operation frequency of this project, which is 100 MHz. It can also be seen that the design could be run at 500 MHz, but with a very small slack.

Table 5.1: Timing slack analysis of the project VHDL-implementation.

Frequency	Slack (ns)
100 MHz	7.784
200 MHz	3.291
250 MHz	2.259
400 MHz	1.291
450 MHz	0.926
500 MHz	0.01

5.2 Project time plan

The initial project time plan is presented in Appendix 1. The order in which the different tasks were performed was followed for the major part of the project, but the time spent on each task was changed significantly. As the Barco Pulse platform is a very complex system, much more time was spent on investigating it than initially planned. There were also detected some deviations in the behavior of some DisplayPort sources in the early stages of the project. Investigating these issues, as well as figuring out how to overcome them, required a lot of extra time.

There were also other, smaller problems that occurred during the implementation phase of the link training module. Some of the smaller problems faced was setting up communication with the NIOS soft processor, problems when setting up the `inout` VHDL signals for the I^2C bus as well as some registers in the DisplayPort IP core that did not update frequently enough. All these problems were fixed, but took more time than expected.

When it was discovered towards the end of the project that the implementation of the link monitoring module would be way more complex than initially planned, and would require way more time to implement, this component was excluded from the project. This made up for the extra time spent earlier in the project.

5.3 Future work

Some of the future work that can be based on this project is presented in this section.

5.3.1 Avoiding DisplayPort deviations

The next step in this project would be to add an extra block that accounts for the deviations from the DisplayPort protocol, especially sources that does not follow the delay specified in the `TRAINING_AUX_RD_INTERVAL` register, as discussed in Section 4.2.1.3. An example of this kind of implementation would be to monitor the AUX-channel during the delay period, and stop the link training module if any activity is detected. This way, the link training module designed during this project could still be present and work with sources that does not follow the specified delay.

5.3.2 Compatibility towards other equalizers

The equalizer used for this project only supports datarates up to 5.4 Gbps per lane, which is the maximum supported bandwidth for DisplayPort version 1.2. For the new versions 1.3 and 1.4 of the DisplayPort protocol, the maximum bandwidth has been increased to 8.1 Gbps per lane. Therefore, a different equalizer is needed. The SN65DP141 from Texas Instruments is designed to support the new versions of DisplayPort. This equalizer supports up to 12 Gbps on the data lanes, as well as a 400 kHz bus speed for the I^2C interface. The main signal path of this component has a different architecture compared to the one used in this thesis project. This architecture consists of an input gain stage, an equalization stage and an output gain stage as is illustrated in Figure 5.3.

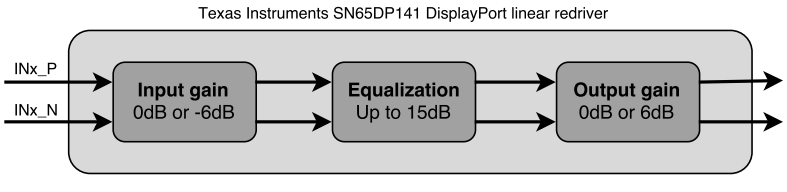


Figure 5.3: Blockdiagram for the SN65DP141 [4].

The changeable settings for the new architecture of the SN65DO141 is different compared to that of the SN75DO130 that was used during this project. These settings are also changeable through logic levels on the pins on the component package. This means that the slow I^2C interface can be avoided when changing the equalization levels of the video path. The link speed and lane count would still have to be changed through the I^2C interface. If this interface would also be used to change the video path parameters, the bus speed of the new equalizer is four times as fast, giving time to test more parameters during the 16 ms delay.

Support for the new SN65DP141 was not a requirement for the link training module during the design process, but the final implementation could be modified without much work in order to be compatible with this component as well.

5.3.3 The link training module

The link training module has not been implemented during this project due to timing issues. This is a potential topic for future work, especially with the next generation of equalizers that can be controlled using GPIO.

6
Conclusion

The main goal of this project was to create an implementation that configures a DisplayPort equalizer during runtime, with the most optimal settings on order to create a stable video link between the video source and the displaying unit. This implementation has been created using an FPGA with VHDL-code. The VHDL-code has been tested and verified against an compatible Nvidia video source, which means that the main goal of the project has been met.

The second goal of the project, which was designing a link monitoring, has not been completed due to time issues. These issues is because of the slow I^2 interface of the currently used equalizer. The next generation of equalizers can be controlled through a faster I^2C interface, but also through GPIO pins. These features may remove the timing issue that was faced in this project, and might make the link monitor possible to implement.

The main implementation of this project has taken such a link monitor into account, preparing some of the VHDL ports that might be needed for this module.

Even if the main goal of the project has been reached, there are still some work to be done in the form of compensating for the deviation found in some of the DisplayPort sources, especially for the missing `TRAINING_AUX_RD_INTERVALL` delay. The current architecture of the created implementation has taken this into account.

The final implementation has been designed in order to cover as my corner cases as possible. Even if the suggestion in the application note from the manufacturer of the equalizer specifies that the equalizer gain should be set to the maximum value at all times, the project implementation will end up using lower equalizer gain values for those setups that might require this setting. The project implementation will also set a gain level so that the threshold for a non-approved signal is as far away from the current driving settings as possible. Having such an implementation in a DisplayPort source is not required in order to pass DisplayPort certification, but it may lead to a wider range of possible setups available to the user.

The implemented VHDL code will be handed over to Barco for further development.

Bibliography

[1] M. Zhang and H. Ngai, "Pre-emphasis and de-emphasis emulation and wave shaping using a programmable delay without using a clock," Jun. 24 2008, uS Patent 7,391,251. [Online]. Available: https://www.google.com/patents/US7391251

[2] "Sn75dp130 datasheet," Texas Instruments, April 2011, Accessed: 16.01.2017. [Online]. Available: http://www.ti.com/lit/ds/symlink/sn75dp130.pdf

[3] "I^2c-bus specification and user manual," NXP Semiconductors, April 2014, Accessed: 07.02.2017. [Online]. Available: http://www.nxp.com/documents/user_manual/UM10204.pdf

[4] "Sn65dp141 datasheet," Texas Instruments, October 2016, Accessed: 29.05.2017. [Online]. Available: http://www.ti.com/lit/ds/symlink/sn65dp141.pdf

[5] "Displayport standard, version 1, revision 2a," Video Electronics Standards Association(VESA), May 2012.

[6] P. Balasubramanian, "Displayport 8k in olympics 2020," Cadence, 2016, Accessed: 25.01.2017. [Online]. Available: https://community.cadence.com/cadence_blogs_8/b/ip/archive/2016/08/31/displayport-8k-in-olympics-2020

[7] "Displayport: High performance digital technology," VESA. [Online]. Available: https://www.displayport.org/

[8] "Hdmi.org," HDMI. [Online]. Available: http://www.hdmi.org/

[9] "Digital visual interface," Wikipedi. [Online]. Available: https://en.wikipedia.org/wiki/Digital_Visual_Interface

[10] "About barco - company profile," Barco, 2017, Accessed: 26.01.2017. [Online]. Available: http://www.barco.com/en/aboutbarco

[11] B. C. Wadell, *Transmission Line Design Handbook*. Artech House, Inc, 1991, iSBN 0-89006-436-9.

[12] "High-speed dsp systems design," Texas Instruments, May 2005, Accessed: 24.04.2017. [Online]. Available: http://www.ti.com/lit/ug/spru889/spru889.pdf

[13] A. P. Alexander Weiler, "High-speed layout guidelines," Texas Instruments, November 2016, Accessed: 24.04.2017. [Online]. Available: http://www.ti.com/lit/an/scaa082/scaa082.pdf

[14] W. H. Paul Horowitz, *The Art of Electronics*, 3rd ed. Cambridge University Press, 2015, iSBN 978-0-521-80926-9.

[15] T. R. Kuphaldt, *Lessons in Electric Circuits*. All About Circuits, https://www.allaboutcircuits.com/textbook/alternating-current/chpt-14/characteristic-impedance/.

[16] "Displayport faq," Video Electronics Standards Association(VESA), 2017, Accessed: 23.01.2017. [Online]. Available: https://www.displayport.org/faq/

[17] "Ds32ev400 - displayport quad equalizer," Texas Instruments, August 20107. [Online]. Available: http://www.ti.com/lit/ds/symlink/ds32ev400.pdf

[18] "Sn65dp141 displayport linear redriver," Texas Instruments, February 2016, document number: SLLSES6A. [Online]. Available: http://www.ti.com/lit/ds/symlink/sn65dp141.pdf

[19] "Qlx4270-dp - displayport lane extender," Intersil, March 2010, document number: FN6972.2. [Online]. Available: https://www.intersil.com/content/dam/Intersil/documents/qlx4/qlx4270-dp.pdf

[20] H. Johnson, *High-Speed Signal Propagation: Advanced Black Magic*. Prentice Hall, 2003, iSBN-13: 978-0130844088.

[21] ——, *High Speed Digital Design: A Handbook of Black Magic*. Prentice Hall, 1993, iSBN-13: 978-0133957242.

[22] "Avalon interface specifications," Altera, December 2017, Accessed: 19.01.2017. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf

[23] "Implementation guide: Dp130 in a sink," Texas Instruments, March 2015, Accessed: 16.01.2017. [Online]. Available: www.ti.com/lit/an/slla349/slla349.pdf

[24] "About barco - company profile," https://www.barco.com/en/News/Press-releases/Barco-expands-its-F-series-range-with-a-unique-rugged-laser-phosphor-projector-that-brings-high-reli.aspx, Barco, 2017, Accessed: 10.04.2017.

[25] "Intel fpga development tools," Intel. [Online]. Available: https://www.altera.com/products/design-software/model---simulation/modelsim-altera-software.html

[26] "Intel quartus prime design software overview," Intel. [Online]. Available: https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html

[27] "Dpa-400," Unigraf, Accessed: 21.04.2017. [Online]. Available: https://www.unigraf.fi/products/dp-test-hardware/dpa-400

[28] "980 series test instruments," Quantum Data. [Online]. Available: https://www.quantumdata.com/980_series.html

[29] "Nios ii processor," Altera. [Online]. Available: https://www.altera.com/products/processors/overview.html

[30] N. Singh, "Link training: Establishing link communication between displayport source and sink devices," Cadence, 2015, Accessed: 23.01.2017. [Online]. Available: https://community.cadence.com/cadence_blogs_8/b/ip/archive/2015/03/23/link-training-_2d00_-establishing-communication-between-displayport-source-and-sink-devices

[31] "Introducing hdmi specification version 1.4a," HDMI, 2017, Accessed: 27.01.2017. [Online]. Available: http://www.hdmi.org/manufacturer/hdmi_1_4/

[32] "Hdmi," Wikipedia, 2017, Accessed: 27.01.2017. [Online]. Available: https://en.wikipedia.org/wiki/HDMI

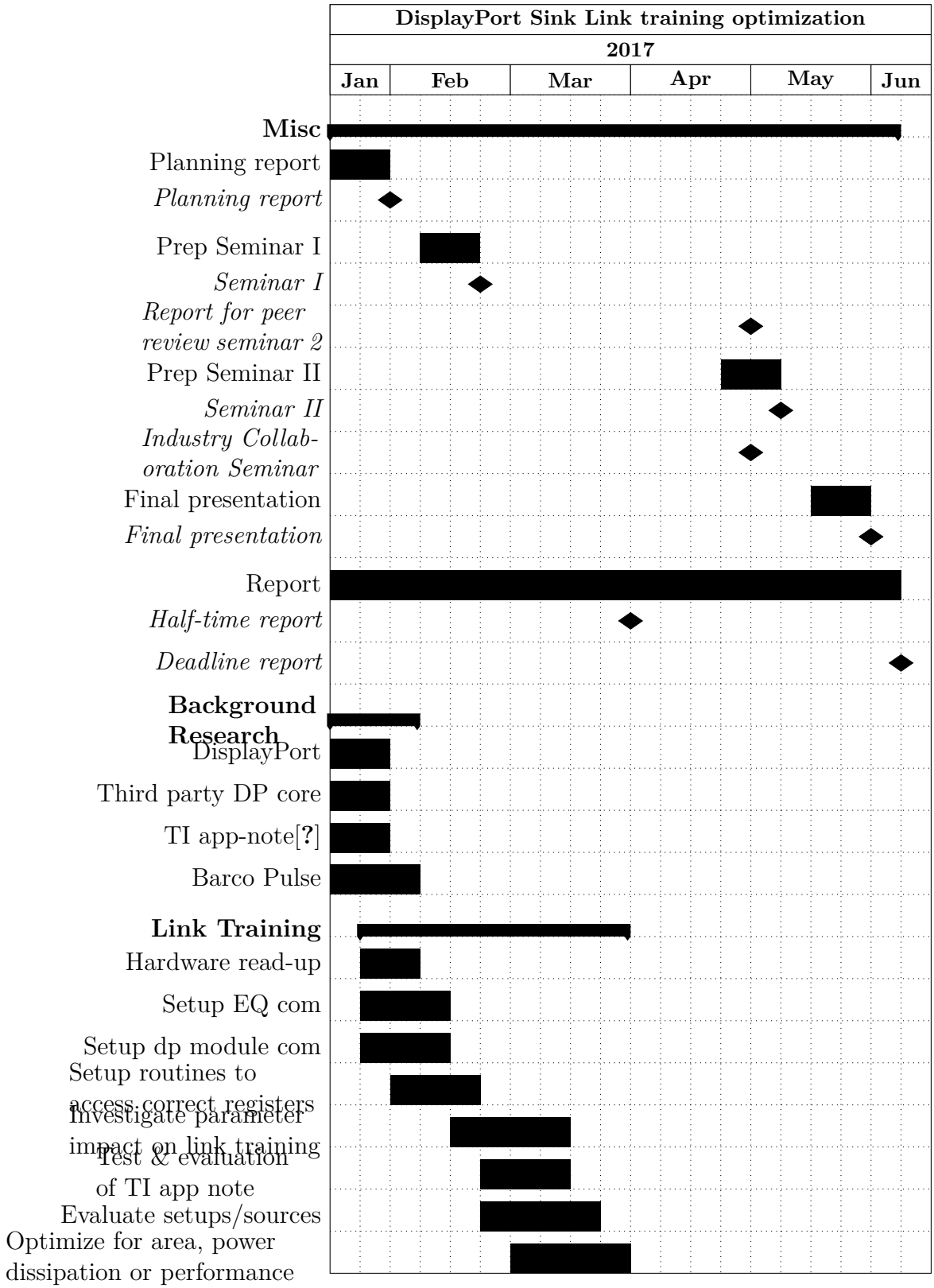
[33] "Arria 10 tramsceover phy user guide," Altera, October 2016, Accessed: 09.02.2017. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/hb/arria-10/ug_arria10_xcvr_phy.pdf

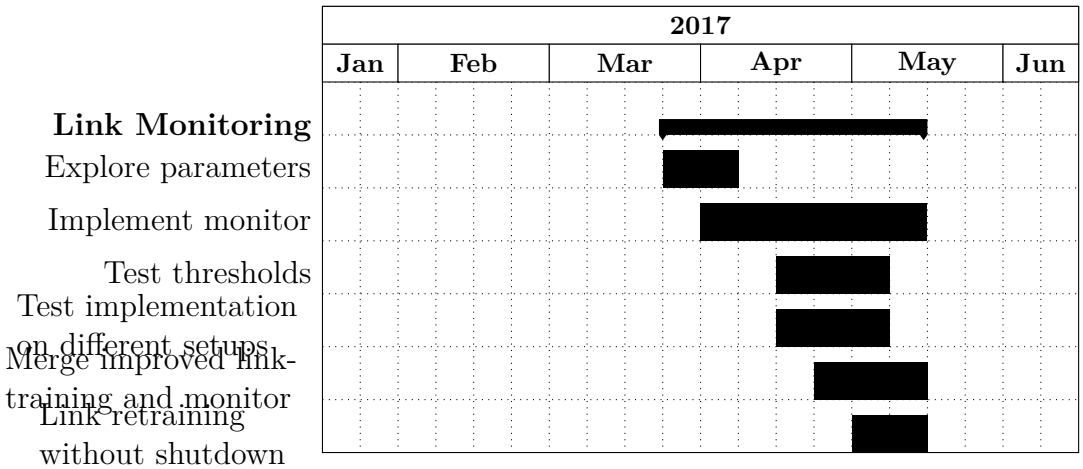
[34] E. Bogatin, "Signal speed on an interconnect," Accessed: 24.04.2017. [Online]. Available: http://www.edn.com/electronics-blogs/all-aboard-/4426188/Rule-of-Thumb--3-Signal-speed-on-an-interconnect

[35] C. R. Wiley, "Displayport 1.2, embedded displayport, and future trends," June 2011, iSSN: 0097-966X.

A

Appendix 1 - Project time plan


DisplayPort Sink Link training optimization
2017
Jan
Feb
Mar
Apr
May
Jun
Misc
Planning report
Planning report
Prep Seminar I
Seminar I
Report for peer review seminar 2
Prep Seminar II
Seminar II
Industry Collaboration Seminar
Final presentation
Final presentation
Report
Half-time report
Deadline report
Background Research
DisplayPort
Third party DP core
TI app-note[?]
Barco Pulse
Link Training
Hardware read-up
Setup EQ com
Setup dp module com
Setup routines to access correct registers
Investigate parameter impact on link training
Test & evaluation of TI app note
Evaluate setups/sources
Optimize for area, power dissipation or performance


2017
Jan
Feb
Mar
Apr
May
Jun
Link Monitoring
Explore parameters
Implement monitor
Test thresholds
Test implementation on different setups
Merge improved link-training and monitor
Link retraining without shutdown