Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Maestría en Electrónica

Design of Elastic Buffers for USB Transceivers in a 180-nm CMOS process

Master thesis presented in partial fulfillment of the requirements to obtain the degree of Master of Science in Electronics – Microelectronics

Javier Andrés Aparicio Morales

Cartago, Costa Rica

September, 2021

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Electrónica
Master Thesis
Evaluation Record

Master's thesis presented to the current Evaluation Committee as a requirement to qualify for the master's degree of the Instituto Tecnológico de Costa Rica

Evaluation Committee

Firmado por RONNY GIOVANNI GARCIA RAMIREZ (FIRMA)
PERSONA FISICA, CPF-01-1137-0229.
Fecha declarada: 11/02/2022 08:13 AM
Esta representación visual no es fuente
de confianza. Valide siempre la firma.

Dr.-Ing Ronny García-Ramírez
Reviewer

ALFONSO CHACON RODRIGUEZ (FIRMA)
PERSONA FISICA, CPF-01-0702-0796.
Fecha declarada: 11/02/2022 01:47:03 PM
Razón: Doy fe
Lugar: Cartago Contacto: alchacon@tec.ac.cr

Dr.-Ing Alfonso Chacón-Rodríguez
Reviewer

JUAN JOSE MONTERO RODRIGUEZ (FIRMA)
Digitally signed by JUAN JOSE MONTERO RODRIGUEZ (FIRMA)
Date: 2022.02.11 11:40:46 -06'00'

Dr.-Ing Juan Montero Rodríguez
External Reviewer

JOHAN CARVAJAL GODINEZ (FIRMA)
Firmado digitalmente por JOHAN CARVAJAL GODINEZ (FIRMA)
Fecha: 2022.02.11 16:18:58 -06'00'

Dr.-Ing Johan Carvajal-Godínez
Master's Coordinator

RENATO RIMOLO DONADIO (FIRMA)
Firmado digitalmente por RENATO RIMOLO DONADIO (FIRMA)
Fecha: 2022.02.10 21:01:06 -06'00'

Dr.-Ing Renato Rímolo-Donadio
Advisor

The members of this Evaluation Committee attest that the present Master's Thesis work has been approved and that fulfills the requirements set by Electronics Engineering School.

Cartago, September 1, 2021

Declaration of authenticity

I herewith declare that I wrote this thesis on my own and did not use any unnamed sources or aid. Thus, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made by correct citation. This includes any thoughts taken over directly or indirectly from printed books and articles as well as all kinds of online material.

Javier Andrés Aparicio Morales
Cartago, September 1, 2021
ID: 1-1575-0288

Abstract

In this work, the design and implementation of elastic buffers at pre-silicon level are addressed. The buffers are designed to be part of USB 2.0 and 3.0 transceivers. Based on a register-transfer level (RTL) description in Verilog and a digital design framework implemented in Synopsys, the design implementation and synthesis are performed using a XFAB 180-nm CMOS process design kit. A high-level simulation framework for USB transceivers was also developed, which allows the incorporation of the building blocks of transmitters and receivers at different abstraction levels, as well as different channel models in terms of S-Parameters.

Strategies for high-speed design, such as clock equalization, were applied to complete the design. Although the designs could be successfully characterized and simulated, only the 2.0 version could reach the specified speed. The 3.0 version can work up to a frequency of 3.8 GHz, but cannot reach the required speed due to process limitations.

Keywords: Digital Flow Synthesis, Elastic Buffer, Links, Timing, Transceiver, USB.

Resumen

En este trabajo se aborda el diseño y la implementación de buffers elásticos a nivel de presilicio. Los buffers se diseñan para formar parte de transceptores USB 2.0 y 3.0. A partir de una descripción a nivel de transferencia de registros (RTL) en Verilog y un entorno de diseño digital implementado en Synopsys, se realiza la implementación del diseño y la síntesis utilizando un kit de diseño en proceso XFAB de 180 nm CMOS. También se desarrolló un entorno de simulación de alto nivel para transceptores USB, que permite incorporar los bloques construidos de transmisores y receptores a diferentes niveles de abstracción, así como diferentes modelos de canal en términos de parámetros S.

Para completar el diseño se aplicaron estrategias de diseño de alta velocidad, como la ecualización del reloj. Aunque los diseños pudieron caracterizarse y simularse con éxito, sólo la versión 2.0 pudo alcanzar la velocidad especificada. La versión 3.0 puede trabajar hasta una frecuencia de 3.8 GHz, pero no puede alcanzar la velocidad requerida debido a las limitaciones del proceso.

Palabras clave: Buffer Elástico, Enlaces, Flujo de Diseño Digital, Temporizado, Transceptor, USB.

to my parents, sister and my dear Alexa

Acknowledgment

First of all I want to thank God for allowing me to be with health, wisdom and understanding during this program. Next, I want to thank my parents Eliecer and Patricia, as well as my sister for always supporting me during this stage. I appreciate all the support they gave me during the internship. I thank Alexa, for always being with me, encouraging me at every moment.

I appreciate the ITCR for providing me with a scholarship to continue my university studies. I want to give a special thanks to Dr.-Ing Renato Rimolo-Donadio who gave me his confidence to carry out this project, besides being a guide during the realization of the project, where he always had time to attend me and correct me when necessary. I also appreciate the fact that he supported me to participate in the internship, which was a great learning experience, and I will always thank him for everything I have learned. Special thanks to Dr.-Ing Alfonso Chacón and Dr.-Ing Ronny García for being part of the evaluation committee. My gratitude with M.Sc. Sergio Arriola-Valverde for all the help that gave me and for always support me with his advices.

Special thanks to Gabriel Rodíguez who was someone I lived for a long time with and was always there to help in whatever was needed. I want to thak to the VLSI group (Luis, Bernardo, Felipe and Daniel) for all the experiences lived in the specialization courses, as well as outside of them. Finally, my thanks to the communications lab team (Andres, Jose and Sofia) who collaborated in the development of this project until the end.

Javier Aparicio Morales

Cartago, September, 2021

Contents

List of Figures

List of Tables

1 Introduction

Nowadays, *Medical Devices* (MD) have gain attention in the last years due to its capabilities for disease detection, monitoring, and control. A medical device is a machine, gadget, or implantable to be used on humans for medical purposes [1, 2, 3]. Recently, the Integrated Circuit Design Laboratory (DCI Lab) of the Electronics Engineering Department at the Instituto Tecnologico de Costa Rica has been working on the development of integrated circuits (IC) for Implantable Medical Devices (IMD). Last year, a RISC-V based microprocessor tailored for IMD applications was tapped out and validated [4, 5], targeting its first application as part of a cardiac stimulation prototype [6].

Since the proper interfaces with external devices and compatibility with well established standards is mandatory for modern IMD solutions, next generations of this microcontroller IP require to extended the supported communications interfaces. An ubiquitous interface is the universal serial bus (USB), which reaches speeds in the multi-gigabit range for its latest specifications [7]. This work is a contribution to the initiative of creating a propietary USB transceiver IP block for its incorporation of custom IC designs developed by the DCI Lab.

In particular, this works deals with the design and implementation of an important building block of the receiver: the elastic buffer. It is in charge of handling slight timing differences between the input received stream clock domain and the one of the internal receiver circuitry.

The development of buffers that can work below the standards 2.0 and 3.0 is addressed, using a top-down digital IC design flow, depicted in figure 1.1. The first block is related to the definition of the specifications of the elastic buffer, such as the operating frequency, power, size, buffer depth. Then, the architecture that is used for the elastic buffer is chosen, and then a functional model at high-level description is proposed and it is implemented up

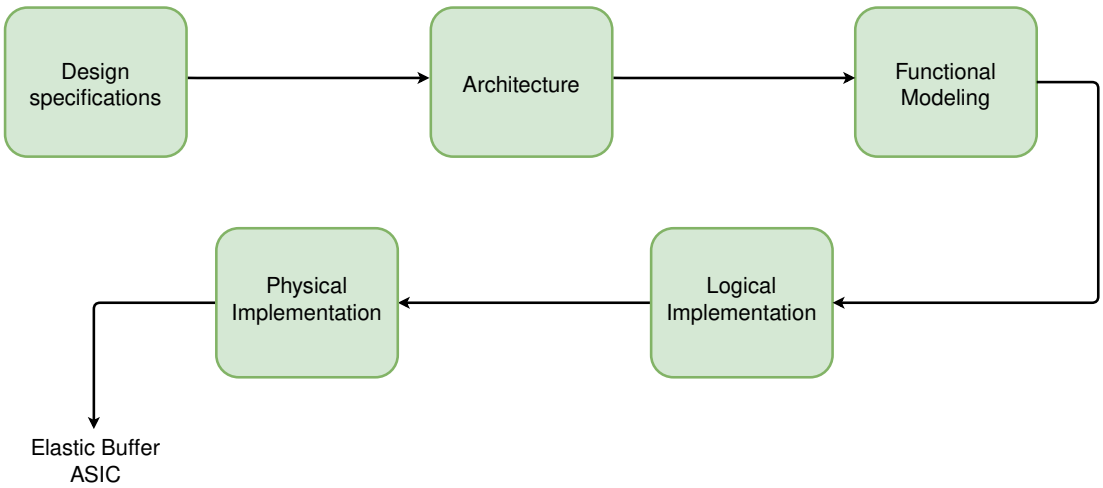


Figure 1.1: Simplified process steps for the design of a digital IC block using a top-down approach

to a pre-silicon level through the logical and physical synthesis design flows.

The design of custom elastic buffers were implemented on a 180-nm CMOS process; to achieve this, first, possible architectural solutions were explored in order to define the specifications and high-level design of the buffers. Then, the designs were implemented at RTL level, for the standard USB 2.0 and 3.0, following a top-down digital design flow with the Synopsys IC framework. The physical design was also proposed and validated with post-layout annotations and simulations.

A USB transceiver simulation framework is also proposed in this work, which allows to consider the other functional blocks and channel models for pre-silicon validation purposes. For ensuring the operation of the transceiver, a behavioral view was used for other blocks than the elastic buffer, for which a detailed implementation was not yet available.

This work is organized as follows: in chapter 2 the literature related to elastic buffer architectures is reviewed. In chapter 3, the high-level simulation framework for USB 3.0 and 2.0 transceivers is presented. The design and implementation of the elastic buffers, from their specifications up to the pre-silicon level, are gathered in chapter 4, where results merely for the elastic buffer implementations were considered. Then, serial transceivers simulations through mixed-signal with the design of elastic buffer are carried out in chapter 5, and, finally, conclusions and recommendations are presented in chapter 6.

2 Elastic Buffer Architectures

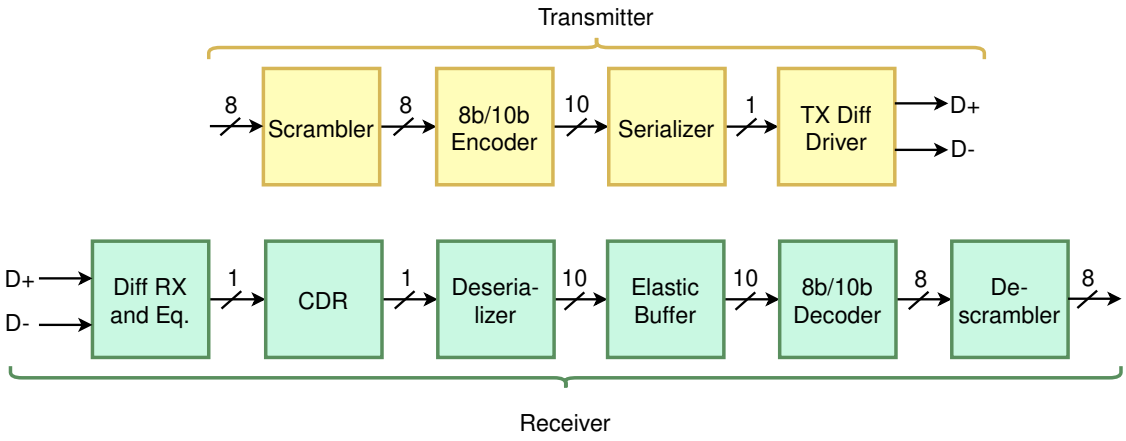
This chapter contains a literature review related to implementations of elastic buffers at integrated circuit level. Two types of elastic buffers are mentioned, namely synchronous and asynchronous, and their differences are discussed. Implementations based on the USB standard are also included. Finally, a comparison of different types of elastic buffers found in the literature is presented, which is used to select the most suitable architecture considering the constraints associated with the selected process and standard constraints.

2.1. Elastic Buffers

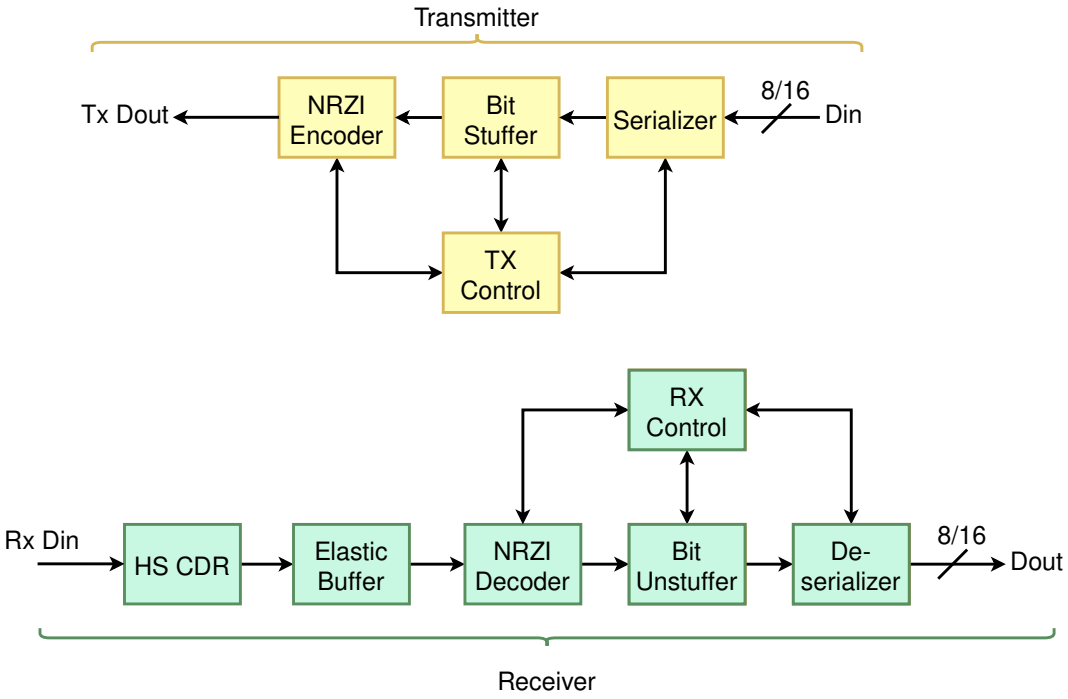
The scope of this work is focused on the implementation of elastic buffers, using a serial transceiver based on an industrial standard. Such standards are the USB with its 3.0 and 2.0 versions. The reasons for using a serial path instead of a parallel one are the complexity and cost of the IC and packaging. Additionally, signal and power integrity are compromised as data rates increase. Increasing the number of lines in the circuits, along with a shrinking transistor channel length, make the proper functioning of an electronic system a challenge.

Typical serial transceivers at a simplified block diagram level are shown in figure 2.1 [8, 9], where the yellow blocks correspond to the transmitter and the green blocks to the receiver. Both stages of the aforementioned figure are not directly connected, an interconnect is presented between the stages, which is responsible for carrying the signals from one place to another. Moreover, this interconnect adds losses such as reflections, crosstalk, attenuation, and dispersion to the transmitted signals that might be harmful.

Synchronization in digital systems is vital to their operation and for ensuring it, the clock signal must be the same in both stages of the transceiver. The synchronization of the serial link relies on the clock signal that is sent along with the information that flows through the



(a) Simplified block diagram of a serial transceiver for USB 3.0 standard. Adapted from [8]



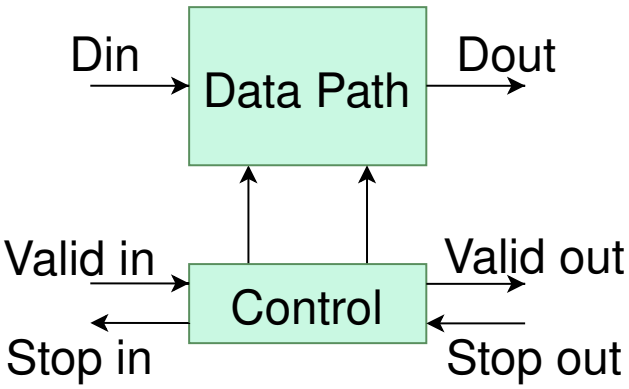
(b) Simplified block diagram of a serial transceiver for USB 2.0 standard. Adapted from [9]

Figure 2.1: Simplified block diagrams for USB transceivers

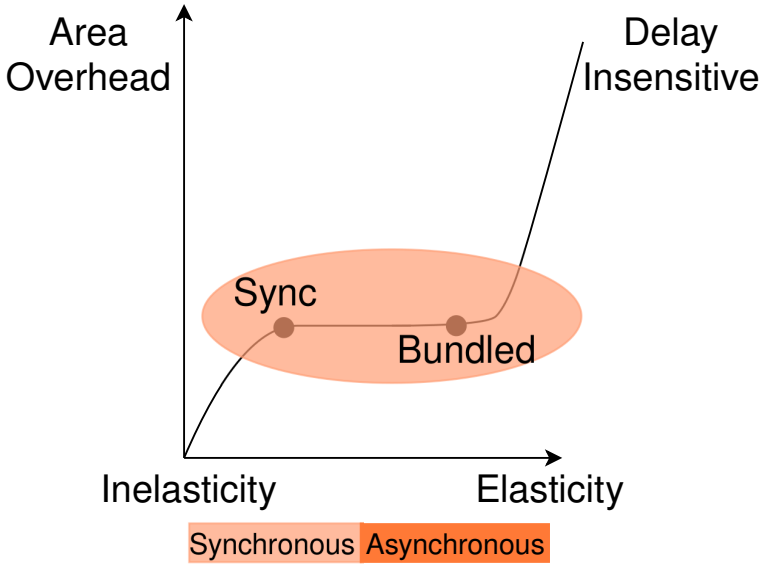
USB link and the stage in charge of this task is the *Clock and Data Recovery (CDR)* block at the receiver stage. Since the clock signal is recovered by the CDR, variations in the data might be introduced because the same clock of the transmitter stage is not being used and hence, the system must be capable of tolerating such variations. There will be occasions when the recovered clock will be faster or slower than the reference signal of the receiver stage, therefore, the block that can tolerate such variations is the elastic buffer. The EB may operate with handshaking protocols or the addition of special symbols, which add delays in the data flow without changing the original content sent from the transmitter [10, 11]. The goal of the elastic buffer is to provide synchronization in the serial link with a local clock domain at the receiver, along with the clock from the received data [12]. Additionally, the advantages offered by the elastic buffer are the reduction in area, power, and delays [13], whose variables are analyzed in this chapter for different architectures of elastic buffers.

The main idea of the elastic buffer is shown in figure 2.2 (a), which includes the control unit and data path blocks as well as the control signals. Either by handshaking protocol or by the use of special symbols, it is able to control the data flow that passes through the serial link [14, 15]. A trade-off is present in the use of elastic buffer, depicted in figure 2.2 (b), since there is a compromise between elasticity and area overhead for different types of circuits. While more elasticity is achieved, the area overhead at a certain moment will increase tremendously. Nevertheless, a certain degree of elasticity can be achieved without a high penalty cost in terms of area. The main difference between the different types of EBs is the control unit and the clock domain they use, because the topology used for the data

path, essentially is the same in both types of EBs [16].



(a) Elastic Buffer structure. Adapted from [14]



(b) Overhead associated for different types of elasticity. Adapted from [16]

Figure 2.2: Block diagram and trade-offs of elastic buffer architectures

2.1.1. Synchronous Elastic Buffers

A synchronous EB can be implemented with master-slave flip-flops for the data path block and a control unit. The two latches that conform the flip-flop, use different enable signals that are given by the control unit. Depending on the status of each latch detected by the control unit, a valid or invalid signal of whether or not to allow the flow of information will be sent. This type of EB uses the same clock signal in the system, where an approach in [14] is with AND and NOT gate arrays, which handle the control unit. These arrays are used to activate the invalid signal, indicating the stopping of the input data, because the current data that is being processed, have not passed completely through the master-slave

flip flop that models the data path block of the figure 2.2 (a). If the control signal is valid, the stream of information will be continued without any problem.

The control signals can be classified into three states for an EB. The first one is the *Transfer* state and this occurs when the receiver, which is the second latch of the flip-flop accepts data from the transmitter (the first latch of the flip-flop). The second one is the *Inactive* state and this occurs when the transmitter is not sending any data. The last state is *Retry* state, which happens when the receiver is not able to accept the data and the transmitter is trying to send new data [14].

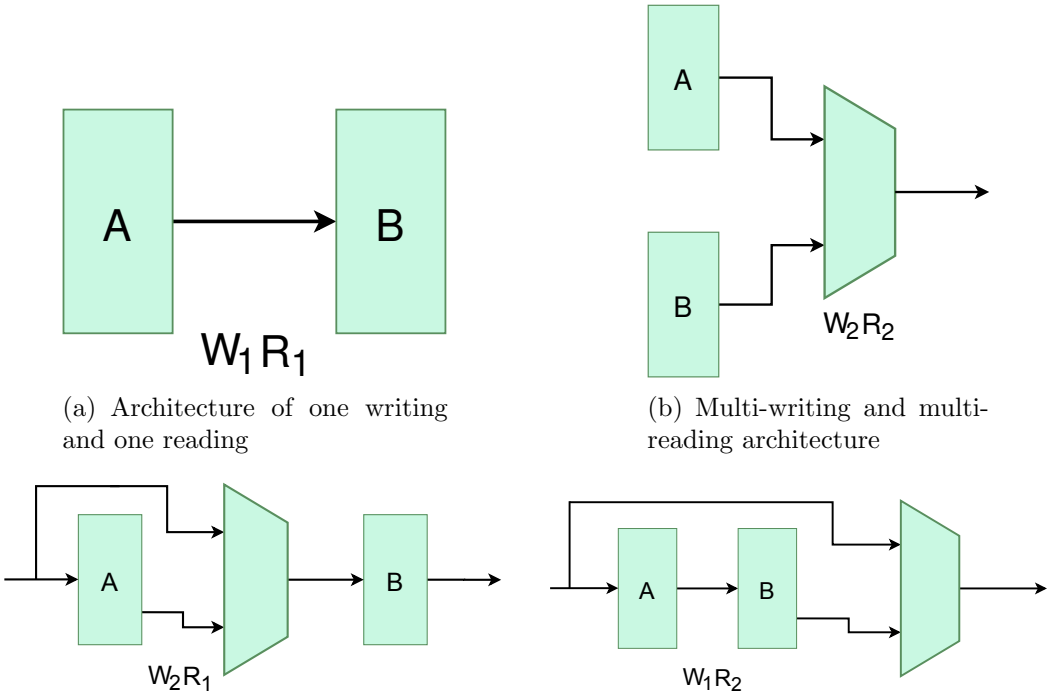
The elastic buffer seeks to reduce latency to a minimum and for ensuring the previous premise, the latency of the sending signal L_f, which is the valid signal, and the response signal L_b, which is the stop signal. Latency must have a value greater than zero; however, this value cannot be high since this would decrease the frequency response of the system. Therefore, the latency value of each signal mentioned must at least be one [14] and this is shown in the equation (1).

$$L_f = L_b = 1 \tag{1}$$

With the previously mentioned, the number of data that the elastic buffer can handle is the sum of both latencies. This entails that the minimum capacity of the EB to operate, based on the equation (1) is 2. Several implementations of elastic buffers are described in figure 2.3, where each one allows several writings and readings that can be performed in the EB [14]. Figure 2.3 shows the different possible combinations: (a), one write and one read, (b)two writes and two reads, (c) two writes and one read, and (d) one write and two reads. The writings and readings for the indicated architectures, are performed through muxes and buffers in first instance. The independent handling of each buffer or latch is achieved with the control signals and handshaking protocol. It is possible to obtain less power consumption, area and latency by using latches to capture each data instead of flip-flops [14].

The operation of the data path is addressed by a control unit where *Finite State Machine (FSM)* controls the information flow through the signals generated by handshaking. The control is composed by three states, the first one corresponds to an empty state, the second one corresponds to a medium state and the last one is the complete state. The empty state indicates that no valid data was captured in the data path. In the medium state, the output latch holds the valid data, while the full state indicates that both latches have valid data, which causes the control unit to stop sending data for processing the current data. The state diagram is shown in figure 2.4, where an equivalent circuit of this diagram can be obtained with OR, AND and NOT gates [17, 18]. The preceding description assumes the minimum latency case that is 2, therefore, this indicates that two latches are used to represent latency.

In [19, 20], the data path block is modeled by a master-slave flip-flop and this is depicted



(a) Architecture of one writing and one reading

(b) Multi-writing and multi-reading architecture

(c) Architecture of two writings and one reading

(d) Architecture of one writing and two reading

Figure 2.3: Elastic buffer data flow architectures. Adapted from [14]

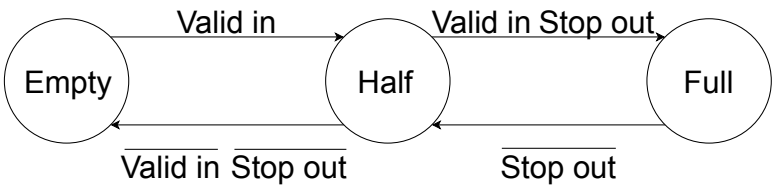


Figure 2.4: State diagram of the EB control unit. Adapted from [18]

in the figure 2.5, where each latch has different enablers coming from the control unit, to allow the transfer of information. The minimum implementation of the EB latency has been shown in this chapter, which corresponds to two and simultaneously corresponds to the same number of latches used for the data flow. Nevertheless, the number of latches can be increased to get a greater data width, which for the USB 3.0 standard would correspond to 10 bits and for the USB 2.0 standard to one bit. The reviewed authors mention that for the implementation at the physical level, the routing of the EB wires must be optimal, for reducing the overhead that the FSM and the latches could add to the system. Increasing the width of the wires and using repeaters to reduce delays are manners to reduce the overhead [20].

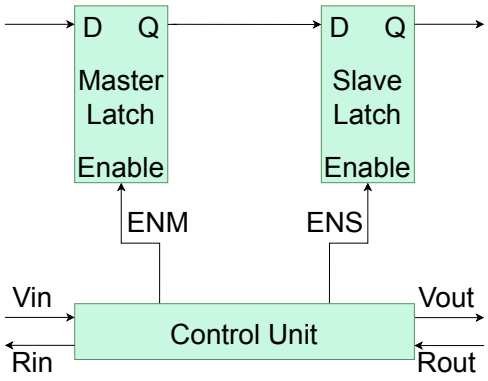


Figure 2.5: Topology of the EB master-slave flip-flop. Adapted from [19]

Among the most typical applications where the elastic buffers are used, there is the *Network on Chip* (NoC). The elasticity that the EB provides is very useful for the interconnection of different NoCs. Since there are many virtual channels in NoCs, a proposal of *Elastistore* is mentioned in [18], where the addition of latches and muxes for each virtual channel are proposed. These channels at the end are managed by an arbiter that is a larger control unit than the conventional EB control that is called *Elastistore Control*. For the generation of the final data output, a mux is used where all the outputs of the virtual channels are connected. In figure 2.6, the Elastistore architecture is described, where each virtual channel uses a latch and a mux. The state machine of this architecture is based on the same principle as the state machine previously described for the EB; therefore, the EB allows the integration of functions for many applications and thus reducing the complexity of the system.

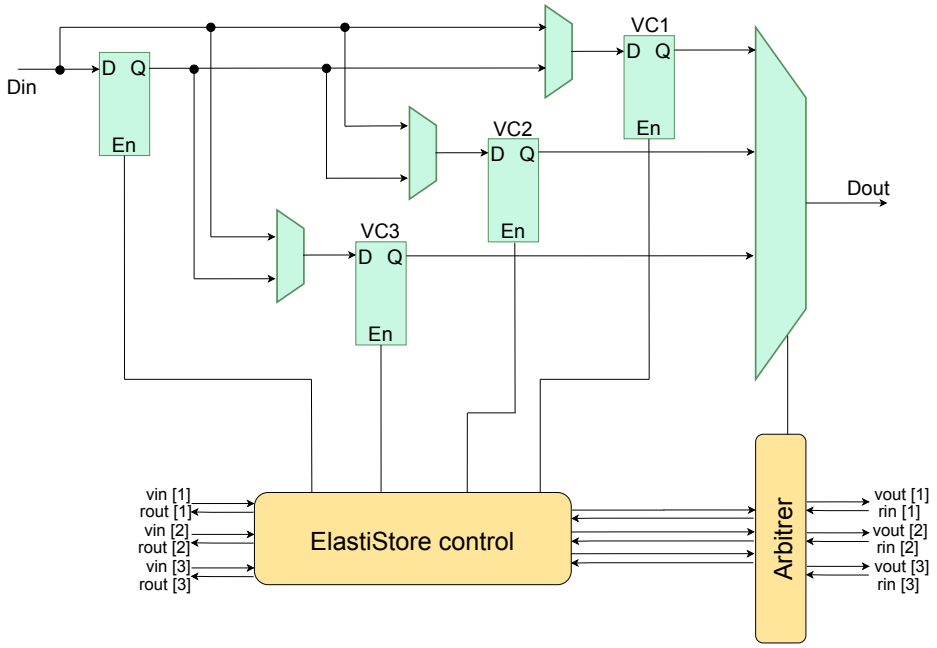


Figure 2.6: Elastistore architecture. Adapted from [18]

2.1.2. Asynchronous Elastic Buffers

Asynchronous elastic buffers can be classified in two groups, which are the four-phase and two-phase approaches. The four-phase protocol is based on handshaking signals, where positive and negative edges of the request and acknowledgment signals are expected. The four-phase implementation is also known as a return-to-zero event, due to the negative edges of the control signals. Its operation is based on the enable signal of the first latch coming from the control unit. The positive edge of the enable signal is presented with the data transfer completion with the next latch; in other words, the negative edge of acknowledge signal and the remaining latches will follow the behavior of the first handshaking when the word width is larger than two bits. Considerations such as the non-overlapping clock for transparent latches should be taken into account, which indicates that the transmission cannot occur simultaneously [16].

The two-phase family has no semantic difference between the positive or negative edges of the signal. A latch called *Capture-Pass* is used in this family. The events in the capture and pass of signals are alternated to allow the latch to capture and pass information alternatively [16]. The capture and pass signals are delays that indicate the capture and pass operations. There are compromises between the two families presented. The two-phase family might present a lower power consumption and achieve higher performance because it avoids a return to zero events. Nevertheless, the two-phase family is more complex to design in terms of logic than the four-phase family.

In [21], an elastic buffer architecture is proposed for the USB 3.0 serial standard. The use of special symbols that are 10-bit data is used for this standard. The *SKP* symbol allows it to be used in the information flow depending on the difference presented between the local domain clock and the clock recovered from the incoming data. If the recovered clock is faster than the local clock, redundant data should be removed or new data added to the buffer should be avoided because more data is being handled than allowed. The opposite happens when the local clock is faster than the retrieved clock, where the SKP signal must be added to the buffer to compensate for the lack of data. For USB, a based of 16 slots is used where each slot has a width of 10 bits and it is sought that half of these slots are always in use. This type of implementation reduces the complexity of the EB, because the handshaking protocol is not the main source of control of the EB, rather the difference between clocks and buffer occupation. The SKP symbol alleviates the EB design because when a difference between clocks exists, the symbol will act depending on the difference obtained. The block diagram that exemplifies the described above, is shown in figure 2.7.

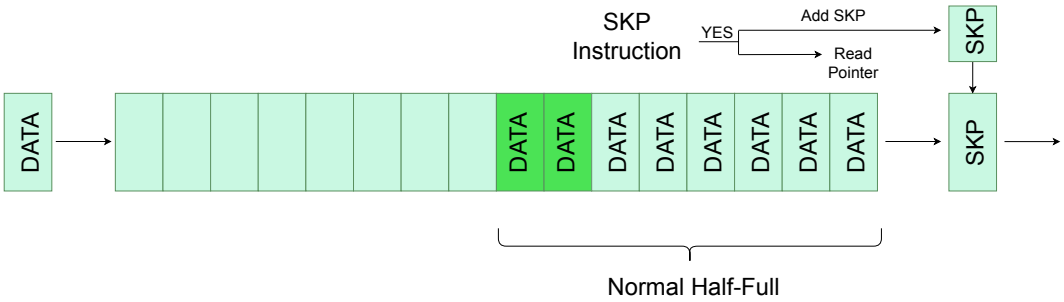


Figure 2.7: EB block diagram for a USB 3.0 implementation. Adapted from [21]

2.2. Comparison of distinct elastic buffers

Table 2.1 shows a comparison of different implementations found in the literature, based on most relevant metrics that characterize the performance of elastic buffers. Only one work is implemented in FPGA, while the remaining works are implemented at CMOS level with technology-node variations from 32 nm up to 180 nm. The clock that the elastic buffer uses to operate varies from 10 MHz up to 7.1 GHz; it is observed that as the process decreases, a higher operating frequency can be achieved.

Table 2.1: Variable comparison between different implementations of elastic buffers

Work	Architecture	Tech	Clock	Cells	Area	Delay	Power	Width
[20]	Synchronous EB for routing NoCs	65nm	2 GHz	4143	14730 μm^2	2.7 ns	0.12 mW	64 bits
[22]	Synchronous EB for routing NoCs	45nm	2 GHz	5691	15080 μm^2	1.8 ns	NA	64 bits
[18]	Synchronous EB for routing NoCs	40nm	NA	NA	62000 μm^2	2 ns	NA	64 bits
[19]	Synchronous EB for routing NoCs	45 nm	NA	NA	17000 μm^2	3.6 ns	2.6 mW	64 bits
[23]	Synchronous EB for routing NoCs	45 nm	NA	NA	52000 μm^2	1.8 ns	38 mW	64 bits
[24]	Synchronous EB for routing NoCs	45 nm	1.1 GHz	NA	3070 μm^2	0.91 ns	330 mW	64 bits
[25]	Synchronous EB for routing NoCs	90 nm	10 MHz	NA	33.8636 μm^2	NA	653.14 μW	24
[26]	Synchronous EB	45nm	500 MHz	NA	1099 μm^2	NA	452.13 nW/MHz	64 bits
[17]	Synchronous EB for FIR filter	180 nm	216 MHz	7821	NA	NA	78 mW-ns	16 bits
[27]	Synchronous EB in FPGA implementation	NA	890 MHz	52	NA	1.123 ns	3473.83 mW	8 bits
[28]	Synchronous EB under process variation	32 nm	7.1 GHz	NA	960 μm^2	0.538	542 μW	NA
[21]	Asynchronous EB for USB 3.0 application	65 nm	500 MHz	NA	11290 μm^2	0.218 ns	4.164 mW	10 bits
[29]	Asynchronous EB for USB 3.1 application	180 nm	318 MHz	1279	33027 μm^2	NA	22.102 mW	32 bits

It should be noted that the evaluation of the elastic buffer topology by itself is not performed in all the mentioned works in table 2.1. For instance, the elastic buffer allows the handling of many routers in a network-on-chip, therefore the evaluation of the NoC system is reported instead of the performance of the elastic buffer only. The delay shown in the above table is quite similar for all works, regardless of whether they operate at high or low frequency. In terms of power, there are variations from 542 μW up to the order of mW. The word width is an important parameter because if it is larger, this implies a more complex logic in the data path to pass the data to the output.

2.3. Summary

Overall, the main advantage that the elastic buffer offers is the is vaster control over the information flow, which allows avoiding losing or corrupting information. The implementation of the elastic buffer can be performed with two approaches, which are asynchronous and synchronous. The synchronous one uses the same clock for the whole architecture, while the asynchronous one uses different clocks; both approaches can use the handshaking method, whereas half-full approach is only used on the asynchronous one.

The half-full method is the best option to implement the EB in both USB standards. The control for the 3.0 version is based on the use of SKP symbols, while for the 2.0 a data threshold must be passed. Thus, further stages are independent of the EB performance. Considerations as the word width to be processed must be considered for the USB 3.0 standard. It could result in a large area requirements or impose tighter timing requirements.

The operating frequency of the EB for the USB 3.0 standard is 10 times less than data rate and hence, this frequency would be around 500 MHz. Since the USB 2.0 standard considers a single serial data path through the whole transceiver, the operating frequency is the same for all the blocks. The aim of this work is focused on high-speed serial links, which causes only the implementation of the high-speed link that works at a frequency of 480 MHz for the 2.0 version.

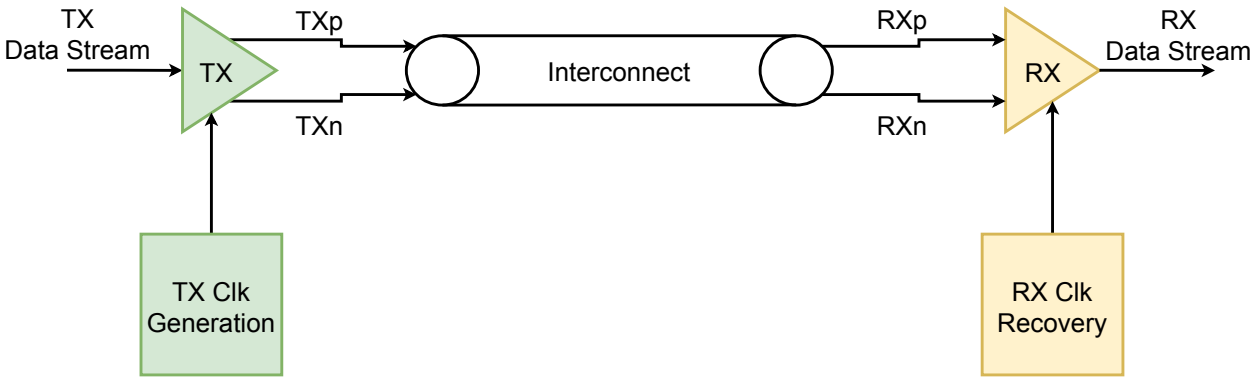
3 Framework Simulation of USB Transceivers

The description of the main USB 3.0 and 2.0 transceiver building blocks is presented in figure 2.1. The overall framework and mixed-signal simulation environment that can incorporate channel models is explained in this chapter. The transmitter, receiver, and the channel composition are explained as part of the simulation framework used to evaluate the design of the different building blocks. Different levels of abstraction can be incorporated to the simulation framework for each building block, from its behavioral Verilog-AMS version up to its post-layout implementation, as it is detailed next.

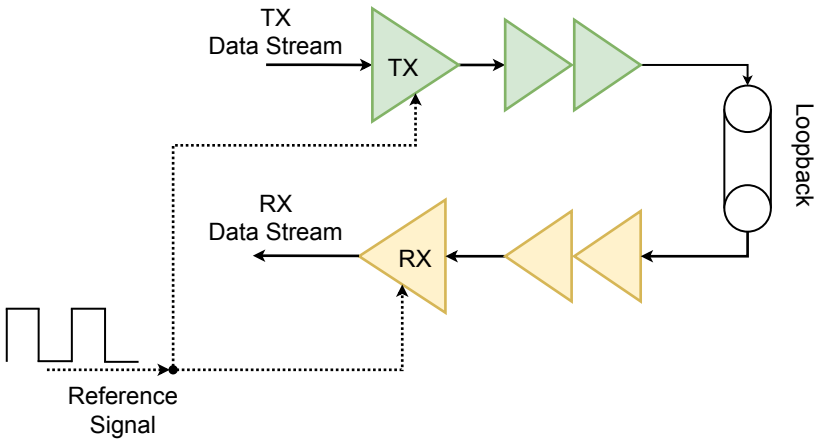
3.1. Framework Simulation for USB Transceivers

In figure 3.1 (a), it can be observed how a full USB link is simulated. This includes the main blocks which are the transmitter (Tx), interconnect channel, receiver (Rx) and the clock generator at the Tx and the recovery timing unit at the Rx. The simulation configuration used in this work is a loopback, as illustrated in figure 3.1 (b), where the interconnect channel is incorporated in the Tx/Rx connection. It is observed how the reference signal feeds the transmitter and receiver. The objective of the framework is to evaluate that the data sent through the transmitter are the same at the output of the receiver stage. Through a mixed-signal simulation environment in Verilog-AMS, analog and digital models are included, as well as the models generated for the EBs through the digital design flow that is detailed in chapter 4. The simulation and validation results using this framework are shown in chapter 5.

The representation of the interconnect is based on S-parameters, where for simulating such channels, a netlist is created and extracted in *Custom Compiler* of Synopsys. The representation is generated including the instantiation of a block that includes the S-parameters from a characterization presented in [30]. The coupling of this block is represented with a differential pair connection. The termination resistance at the Tx and Rx sides are considered



(a) Simplified diagram of the simulation setup for a single differential lane



(b) Loopback configuration used for USB transceiver simulations

Figure 3.1: Framework for simulation of USB transceivers

in the simulation, as it will be discussed later. The simulator used for mixed-signal analysis is the *Custom Waveview* of Synopsys. The framework simulations are performed through scripts, which are documented in [31]. This means that the description of each of the blocks that integrate the transceiver is included through coding lines in Verilog-AMS to represent either their analog or digital behavior. The interconnect and the elastic buffer are modeled from a derived netlist. The channel mode is imported from a touchstone S-Parameter file, and the EB is a generated from the IC post place and route netlist obtained by the Synopsys digital synthesis framework, using tools such as *Design Compiler* and *Integrated Circuit Compiler*.

Since both the digital and analog domains are required in the simulations, analog to digital (AD) and digital to analog (DA) converters must be used in the AMS simulation framework. Digital signals such as clock, reset, data and control signals are injected at the input of the transceiver before being connected to the channel, and are transformed to the analog domain at the equalizer and Tx driver. The conversion is based on a description in Verilog-AMS, where characteristics such as the number of bits to be converted (1 bit for the USB link), and the maximum voltage level (3.3 V for both links) can be adjusted. The signals should remain analog up to the first digital block of each USB transceiver at the receiver (deserializer for 3.0 and EB for 2.0). Before converting the signals back to the digital domain, comparators are used in order to determine if the receiver input differential voltage represents a higher or lower value. Once the comparator generates the value, analog-digital converters (ADCs) are used. Settings as the number of bits to represent and midpoint level can be adjust. More details about digital-analog converters (DAC) and ADC can be found in [31].

The descriptions of the USB 3.0 and 2.0 transceivers are detailed next. Each transceiver contains transmitter and receiver blocks, in which the analog and digital models are detailed. The interconnect that is used in the loopback is included, with its characterization and model

extraction from S-parameters.

3.2. Transmitters for USB 3.0

The first part of the transceiver belongs to the transmitter, which is depicted in figure 3.2. It has four blocks, which are the scrambler, encoder, serializer, and differential driver. Each block is explained below and modeled at a behavioral level. It is used the environment of mixed-signal that combines digital and analog behaviors. The general specifications based on USB 3.0 standard are the data rate of 5 Gbps with a bit-time of 200 ps for serial data, whereas for parallel data, 10 bits are used with a period of 2 ns.

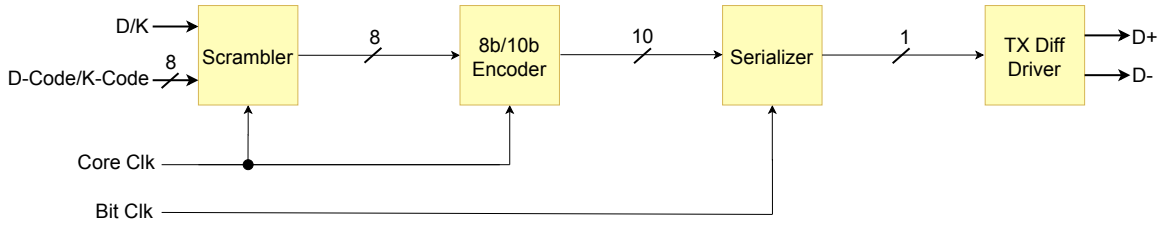


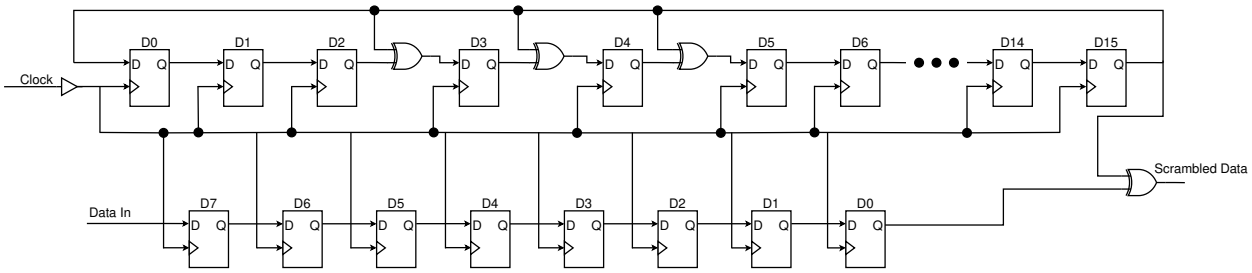
Figure 3.2: USB 3.0 transmitter simplified block diagram

3.2.1. Scramblers

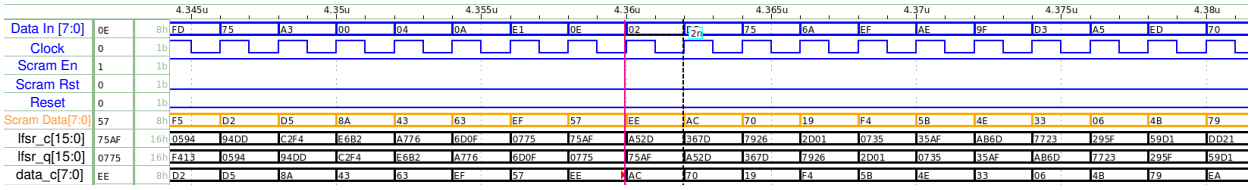
Data scrambling is necessary for serial links as a result of imperfections in the transmission lines. Poor line integrity leads to consecutive bits in error and these errors cannot be corrected or even detected. Scrambling data solves this issue, by adding controlled randomness and spreading error bits in close proximity [32]. The implementation used for the scrambler consists of a linear feedback shift register with the polynomial expression in equation (2) [8], using the open-source code implementation in [33]. The circuit representation of equation (2) is shown in figure 3.3 (a).

$$G(X) = X^{16} + X^5 + X^4 + X^3 + 1 \tag{2}$$

This block is implemented at a behavioral level with the use of a hardware description language as Verilog. A mixed-signal simulation is carried out for modeling the USB transceiver. This block is one of digital type and its simulation results are observed in figure 3.3 (b). The blue signals belong to its inputs, the orange one is the output and the black ones are internal registers that perform some preliminary results. The reference signal has a period of 2 ns since the data is driven in a parallel form and it is also observed with the markers. The input data is performed with the use of several *Pseudo-Aleatory Random Bit Sequence* (PRBS) of different seeds to introduce to the system some randomness in terms of evaluation and not expected predictable patterns. Among the purposes of scrambling, the balance in the number of logic levels is performed for avoiding issues at the receiver stage with the recovery clock [34].



(a) LFSR with scrambling polynomial. Adapted from [8]



(b) Scrambler simulation results at 500 MHz

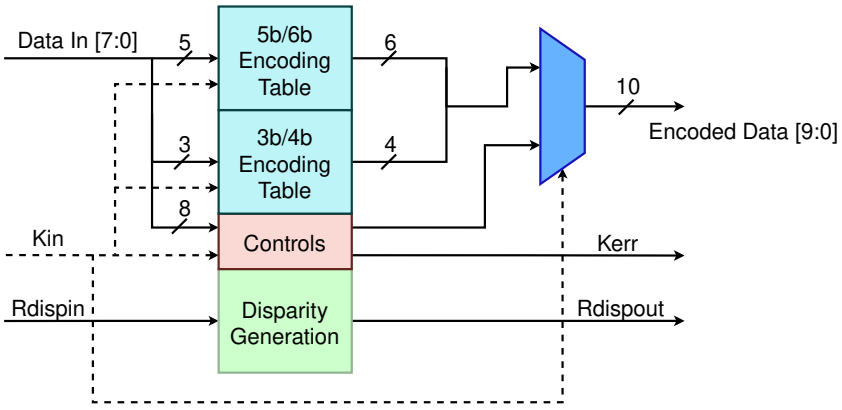
Figure 3.3: Scrambler block diagram description event-driven simulation of its behavioral AMS model

3.2.2. 8b/10b Encoders

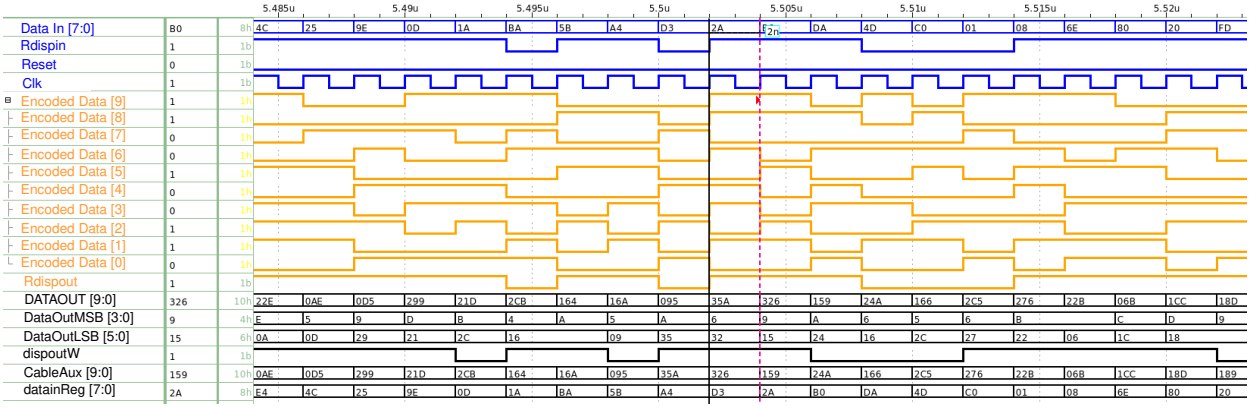
The next block is an encoder 8b/10b that converts an 8-bit input data into 10-bit data. In figure 3.4 (a), it is depicted a general description on how the input data is separated into 3 MSB and 5 LSB. The 3 bits transform into 4 bits, and the remaining 5 bits transform into 6 bits. Two conversion tables are used for making the mapping depending on disparity generation. The disparity generation is implemented with a state machine as follows: if the incoming data show even parity, the next input data maps to a different disparity with respect to the actual data, but if the incoming data have odd parity, the next data block will use the same disparity [35].

There are special characters that are used for control. The first input is called *Kin* to let the encoder knows how to map the 8-bit input. In case the input is an invalid special character, the second signal *Kerr* toggles and the output will be the corresponding information block [36]. Three encoder implementations are explored in [35]; for this work is implemented the decoders and multiplexers approach, which is depicted in figure 3.4 (a). The 8-bit data is converted properly into the 10-bit representation. The decoder transforms the input data to a specific value and the multiplexer determines, which data go out for each decoder since the disparity signal acts as a control line.

The simulation of this block belongs to a digital type, and the procedure as scrambler is applied for the encoder. In figure 3.4 (b) is observed the response of the encoder; the blue, orange and black signals correspond to the inputs, outputs, and internal registers, respectively. Balance in the number of ones and zeros is exploded by this block, in order to facilitate the recovery reference signal in the receiver stage by the CDR. This balance allows a



(a) General diagram of 8b/10b encoder. Adapted from [35]



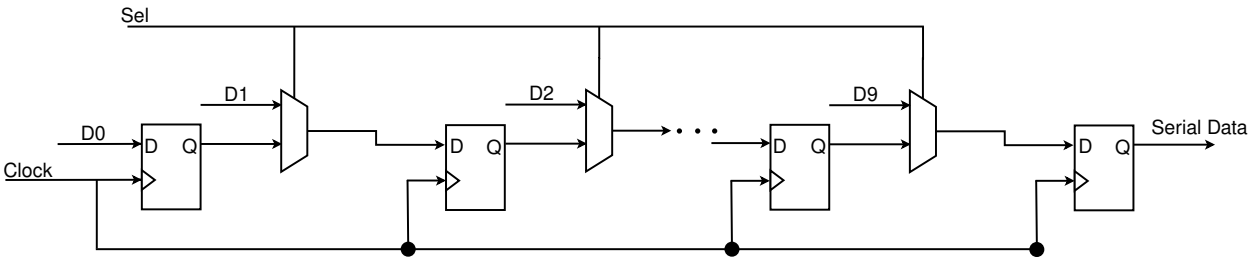
(b) 8b/10b encoder simulation results at 500 MHz

Figure 3.4: 8b/10b encoder block diagram description and event-driven simulation of its behavioral AMS model

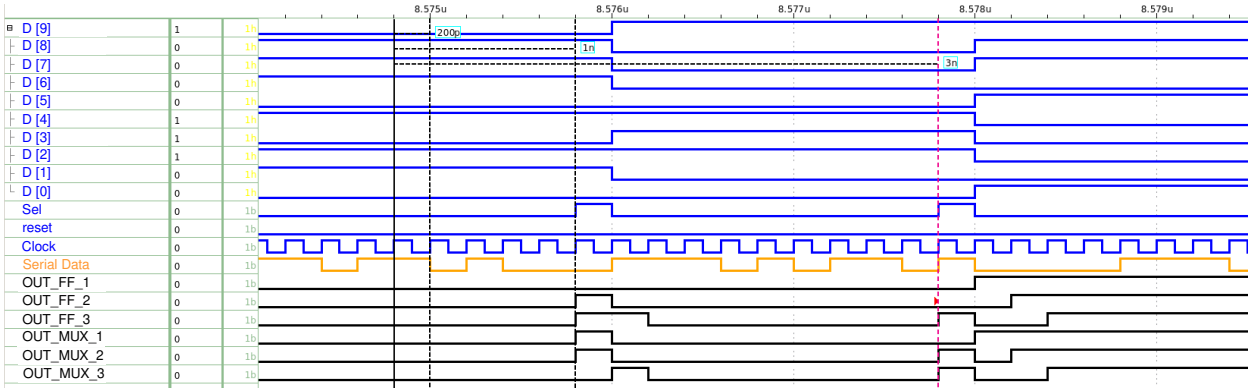
maximum of 5 consecutive bits with the same logic level, otherwise, the codification is wrong. The balance can be observed and counted with the markers of the aforementioned figure. This block is the last one with a slower reference signal, because this block is connected to the serializer. The action of disparity is observed, where the parity signal is changed with respect to its output and input data.

3.2.3. Serializers

The next block is the serializer, which converts data from a parallel arrangement to a stream of bits that can be converged through a single lane. This allows reaching transmission rates at gigabits per second with a dedicated serial channel that is properly designed. The implementation for this work is based on shift registers and multiplexers [37]. For transferring the 10-bit data without losing any bit after the 8b/10b encoder, the serializer must work ten times faster than the parallel blocks such as scrambler and encoder. The faster clock of 5 GHz and the slower one of 500 MHz feed the serial and parallel blocks,



(a) Serializer's simplified schematic. Adapted from [37]



(b) Serializer simulation results at 5 GHz

Figure 3.5: Serializer block diagram and event-driven simulation of its behavioral AMS model

respectively. In figure 3.5 (a), it is observed a simplified circuit for serializing the parallel data. For each 9 clock cycle of the serial reference signal, the mux selector is switched for the propagation of data through the circuit, making the serialization possible.

In figure 3.5 (b), the serializer is modeled at a behavioral level with the same scenario of previous blocks. The difference is the frequency because this block must be 10 times faster. The blue signals are their inputs and these are the clock, reset, and selector. The stream of data is observed with the orange signal and it has period of 200 ps that is the bit-time given by USB 3.0 standard. The black signals perform the shift of parallel data, which are internal signals of the logic. It is observed that the selector is asserted only 1 of 10 possible clock periods. The remaining 9 periods are low and it is for propagating the data through the registers.

3.2.4. Transmitter Differential Drivers

The signals from previous blocks might degrade their levels, which cause *Intersymbol Interference* (ISI) and inadequate interpretations at the receiver caused by the losses that interconnect adds. Therefore, the signals are enhanced before the interconnect to avoid such problems. They are driven by a buffer that allows them to transmit the signal either by their current or voltage levels. *Low-Voltage Differential Signaling* (LVDS) is used for this

implementation, since the facility to transmit voltage levels instead of current, along with a decrease in power consumption in an eventual IC implementation. The block diagram of the transmitter driver is observed in figure 3.6. The enhancement of the signal is addressed by the *Feed-Forward Equalizer* (FFE), followed by the LVDS. The output of this driver goes to the physical channel.

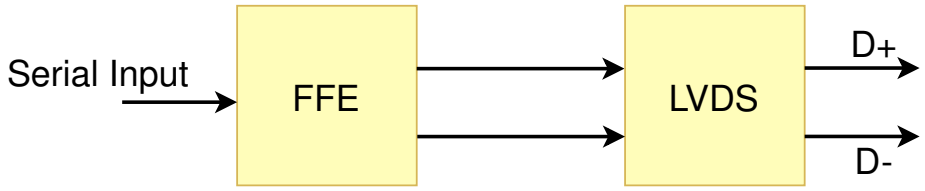
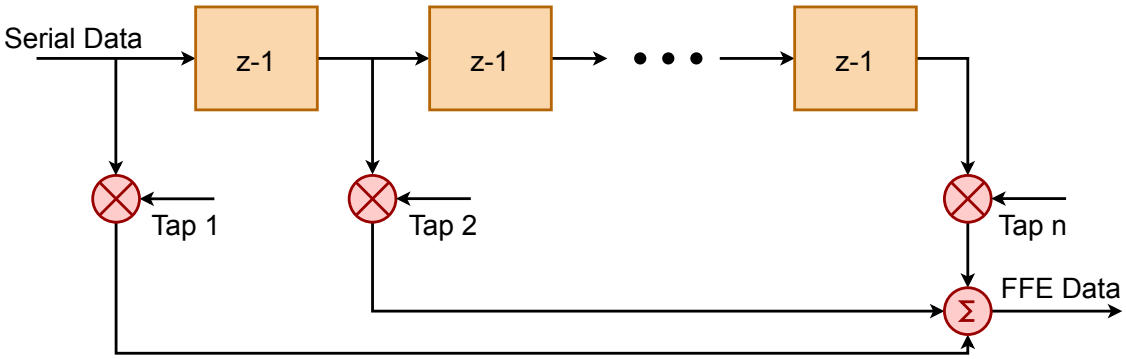


Figure 3.6: Transmitter differential driver simplified block diagram

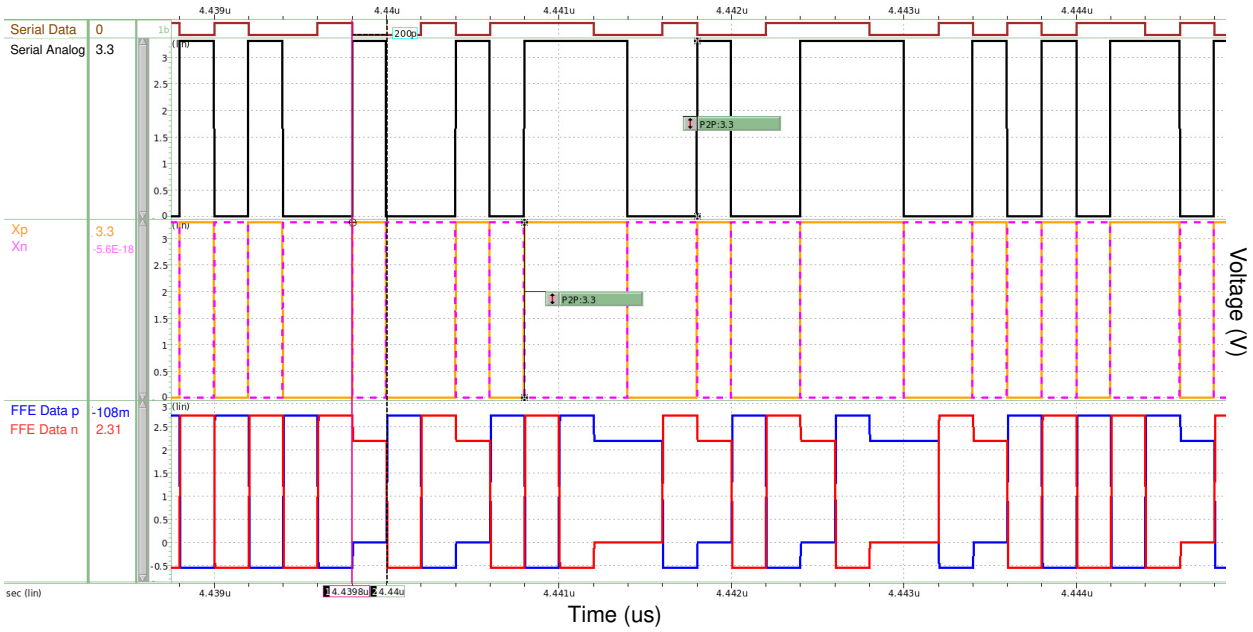
The FFE improves the signals through the use of a discrete filter. The filter is a *Finite Impulse Response* (FIR) topology that means no feedback on the system. The representation of this filter is by a difference equation and the operating principle is observed in figure 3.7 (a). The sum is performed by the input sample and its delayed samples multiplied by a factor called tap. The goal of the filter is to eliminate the post-cursor or pre-cursor of the signal to increase the signal levels [38]. The transfer function of the FFE is observed in equation (3), where the N factor belongs to the number of taps. The minimum number of taps is 2, otherwise, it does not behave as a filter. The number of taps used for this filter is 2 and they are 0.834 and -0.166 as first and second tap, respectively [39].

$$y(n) = \sum_{k=0}^{N} b_k x(n-k) \tag{3}$$

Since this driver is connected to a physical channel, this must be modeled in an analog manner. The first step is to transform the digital serial data that comes from the serializer. Once this is done, the signal is passed to analog. The generation of the negative signal is necessary in order to accomplish the principle operation of the equalizer, where the positive



(a) FFE equalizer's simplified block diagram. Adapted from [38]



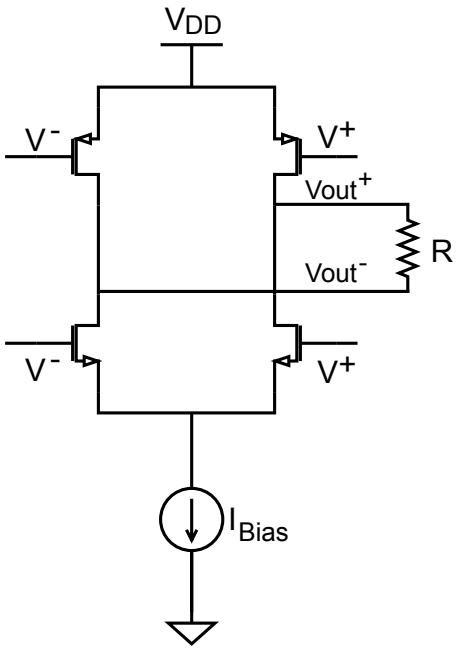
(b) Feed-forward equalizer simulation results at 5 Gbps

Figure 3.7: FFE block diagram description and simulation of its behavioral AMS model

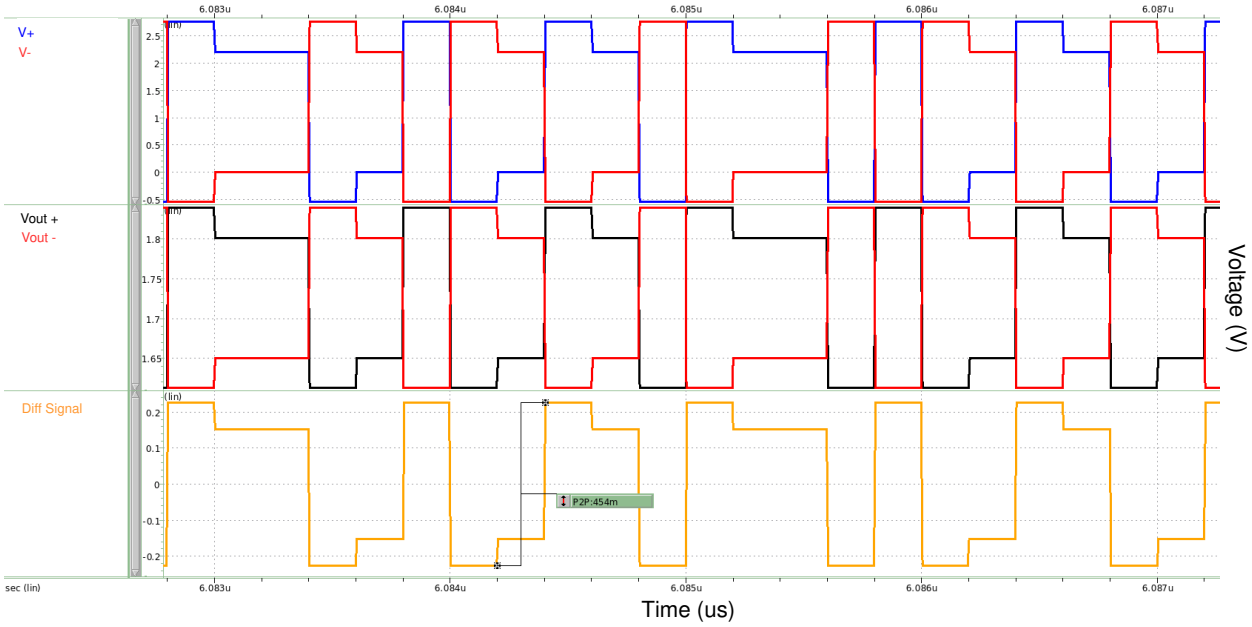
and negative signals are required. In figure 3.7 (b), it is depicted all the previous description, where the input digital data is the brown signal. The black signal shows the transformation of digital to analog, and a variation voltage from 0 to 3.3 V is observed. The generation of the positive and negative signals are observed in the ,solid-orange and dash-pink signals. respectively; these signals are opposite to each other. Finally, the equalization of the signals is performed for providing to the interconnect lower values of noise, distortion, and losses. In the blue and red signals are described the behavior of the equalizer. A decrease in each signal is observed when consecutive bits with the same level voltage is presented; otherwise the high-level is preserved. The main idea of this response is to eliminate either the pre-cursors or post-cursors of the main signal. It is observed that the outputs have a period of 200 ps dictated by the USB 3.0 version.

The equalizer is driven by a differential buffer that can be voltage or current. In this work is used voltage signaling because of the simplicity of interpretation [40]. The buffer allows transmitting a lower peak to peak voltage (Vpp). This alleviation enables the transmission of a signal range of 150 mVpp up to 400 mVpp [41, 42]. Several configurations are found in [40, 41, 42], however, the scope of this work is not the implementation of this block. In figure 3.8 (a), it is depicted a configuration for LVDS to give some insight at circuit level; nevertheless, the use of other blocks as *Common-Mode Feedback* (CMFB) is necessary to improve the differential signals. The outputs are connected to a resistance (R) that models the differential impedance of the interconnection that is 90 Ω.

The output of the equalizer is buffered to an LVDS for matching the interconnect and the transmitter. The block is modeled at analog-behavioral level, where the positive and



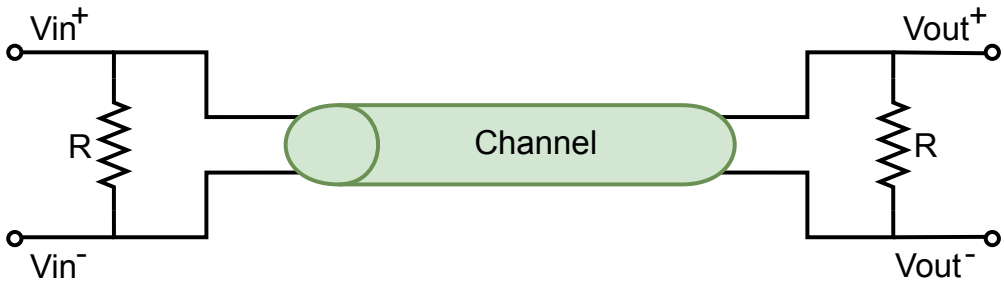
(a) LVDS's simplified circuit diagram. Adapted from [40]



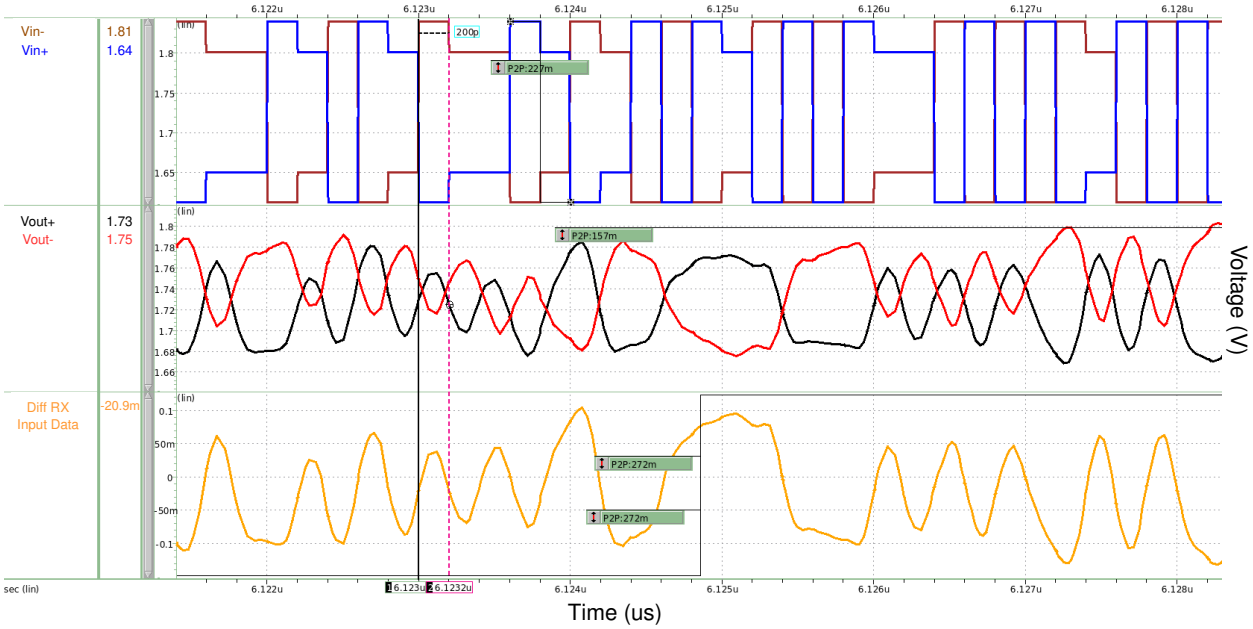
(b) Transmitter LVDS simulation results at 5 Gbps

Figure 3.8: LVDS circuit description and event-driven simulation of its behavioral AMS model

negative are received and these are conditioned to represent a lower Vpp. In figure 3.8 (b), is observed the simulation results of LVDS, where the red and blue signals (first row) are the inputs. The red and black signals (second row) are the outputs of the LVDS. It is observed the offset of the signals as well the decrease in the Vpp voltage around 225 mV for each



(a) Channel simplified schematic that connects transmitter and receiver



(b) Channel simulation results of 2 meter length at 5 Gbps

Figure 3.9: Interconnect description and simulation at mixed-signal level

one. The difference of such signals is used to evaluate if the incoming signal at the receiver is represented with a high or low value. The difference is represented in the orange signal (third row), where the Vpp injected to the interconnect is near 450 mV. Inevitably, losses caused by interconnect are presented and the receiver will drive a lower Vpp value. It is vital the implementation of a transmitter driver to avoid further loss of signals at higher data rates.

3.3. Interconnect for USB 3.0 and 2.0 Transceivers

The physical channel is responsible for carrying the signals from transmitter to receiver. This channel can be made of cables, connector or PCB segments. Characterization of different physical channels is discussed in [30]. At the same time the specifications for USB 2.0 transceiver are covered with this interconnect. Essentially, the channel models the differential pair, where its differential impedance is 90 Ω. The purpose of the interconnect for

this work is to model a complete transceiver, where a channel is included in order to add losses to the signals for getting a realistic response. The used channel for this work is from [30], where the S-parameters are used for the simulation in the mixed-signal environment. In figure 3.9 (a), it is depicted the functioning of the interconnect, where the resistances describe the matching that should exist between both inputs and outputs. The matching along with both LVDS at transmitter and receiver must avoid the arise of losses as well other effects that appear on high-speed interfaces.

The results of the measurement of the interconnect are observed in figure 3.9 (b). The brown and blue signals represent the output of the transmitter LVDS buffer. The outputs are represented with the red and black signals, where losses and distortion are observed in such signals. The differential voltage at the output is observed in the yellow signal. There is a decrease in the magnitude of Vpp since the value passes from 450 mV around 225 mV.

3.4. Receiver of USB 3.0 Transceivers

The third part of the transceiver belongs to the receiver, which is depicted in figure 3.10. It has seven blocks, which are the differential receiver driver, CDR, deserializer, elastic buffer, k28.5 detector, decoder, and descrambler. Each block is explained below and modeled at a behavioral level and is used a mixed-signal environment that combines digital and analog behaviors. The same specifications are applied for the receiver based on USB 3.0 standard. The data rate is 5 Gbps with a bit-time of 200 ps for serial data, whereas for parallel data, 10 bits are used with a period of 2 ns. Most of the blocks work in an inverse manner of transmitter because the data must be recovered into an 8-bit data.

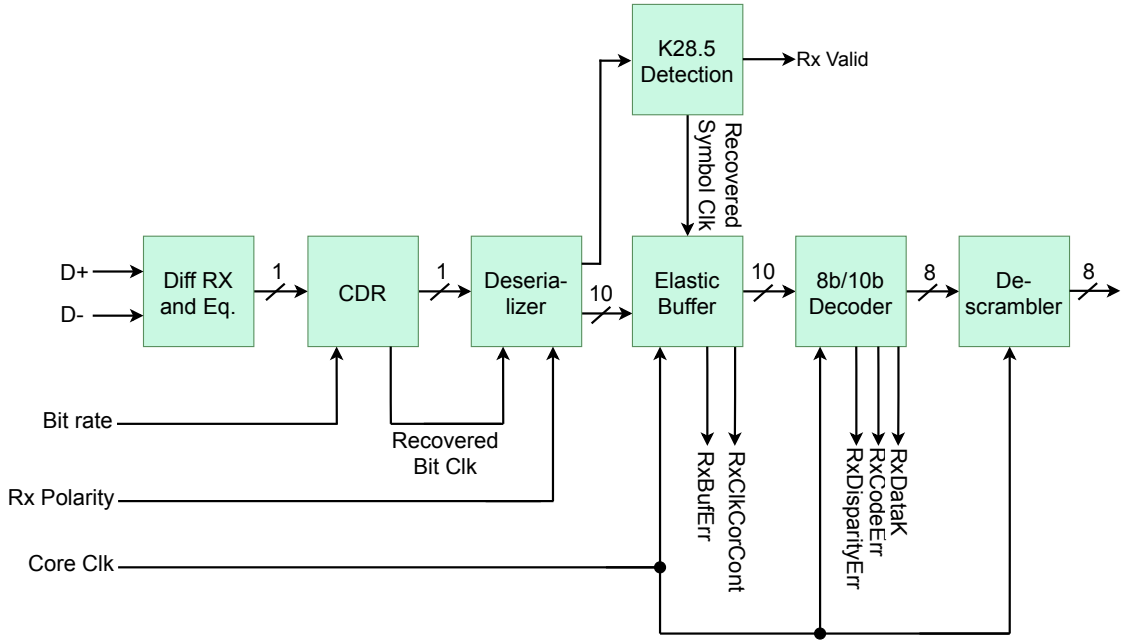


Figure 3.10: USB 3.0 receiver simplified block diagram

3.4.1. Receiver Differential Drivers

The signals at the edge of the interconnect are received by a differential driver, which operates oppositely to the transmitter driver. The signals are treated by the receiver LVDS and then, they are equalized because of the losses that the interconnect adds. These losses are the reflection, crosstalk, dispersion, attenuation, among others [43]. The aforementioned description is depicted in the block diagram of figure 3.11, where the LVDS is the same as the transmitter in terms of operation principle, and the equalizer is a *Continuous Time Linear Equalizer* (CTLE). The aim of the LVDS is to match the signals of the interconnect with the receiver. Such signals are degraded resulting in a lower Vpp than when they were transmitted. It is sought that their differential Vpp is above 200 mVpp [44].

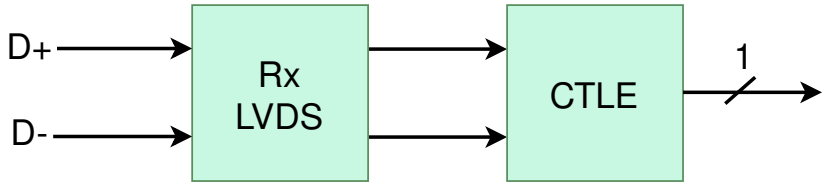


Figure 3.11: Receiver differential driver block diagram

The signals at the output of the receiver LVDS are enhanced by the CTLE. Such equalizers is a continuous equalizer, where the circuit that represents the equalizer is observed in figure 3.12. The differential configuration, along with the array of the passive components as resistances and capacitors, generate the existence of two poles and a zero in the frequency response of the equalizer [45]. The transfer function of the equalizer is observed in equation (4).

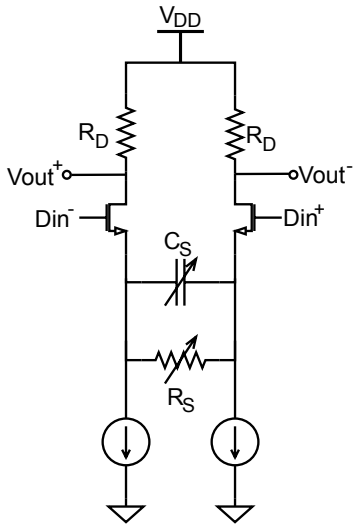


Figure 3.12: CTLE circuit representation. Adapted from [45]

$$H(s) = A_{DC}\frac{s-\omega_z}{(s-\omega_{p1})(s-\omega_{p2})} \tag{4}$$

Where A_{DC}, ω_z, ω_{p1} and ω_{p2} are described in equations (5), (6), (7), and (8) respectively. The parameters shown in these equation belong to the components of figure 3.12; furthermore, the transconductance corresponds to the transistor pair-differential and C_p is the load that CTLE drives at the output.

$$A_{DC} = \frac{g_m}{C_p} \tag{5}$$

$$\omega_z = \frac{1}{R_s C_s} \tag{6}$$

$$\omega_{p1} = \frac{1 + g_m R_s/2}{R_s C_s} \tag{7}$$

$$\omega_{p2} = \frac{1}{R_D C_p} \tag{8}$$

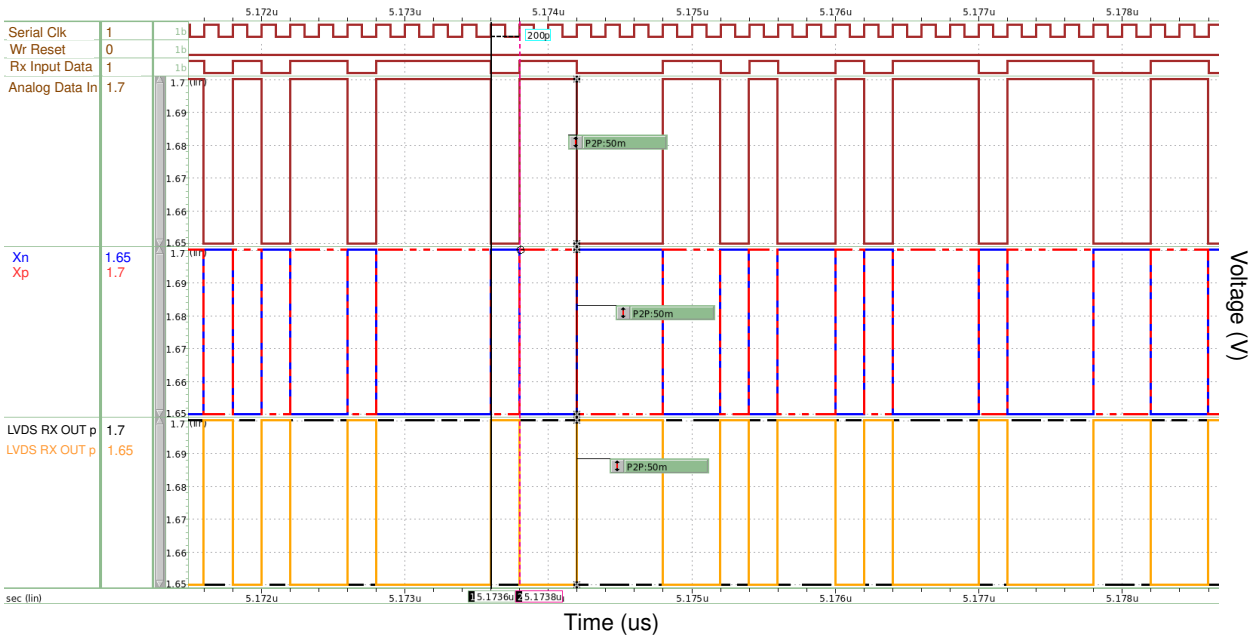
In [8], it is mentioned the values of each parameter of the transfer function of equation (4). The values are the gain, zero, and poles, which are observed in table 3.1 with their frequency expressed in Hz and the angular frequency value. The scope of this work is to model the CTLE at a behavioral level, which means using the values of the below table and not making a circuit derivation.

Table 3.1: CTLE values of their transfer function

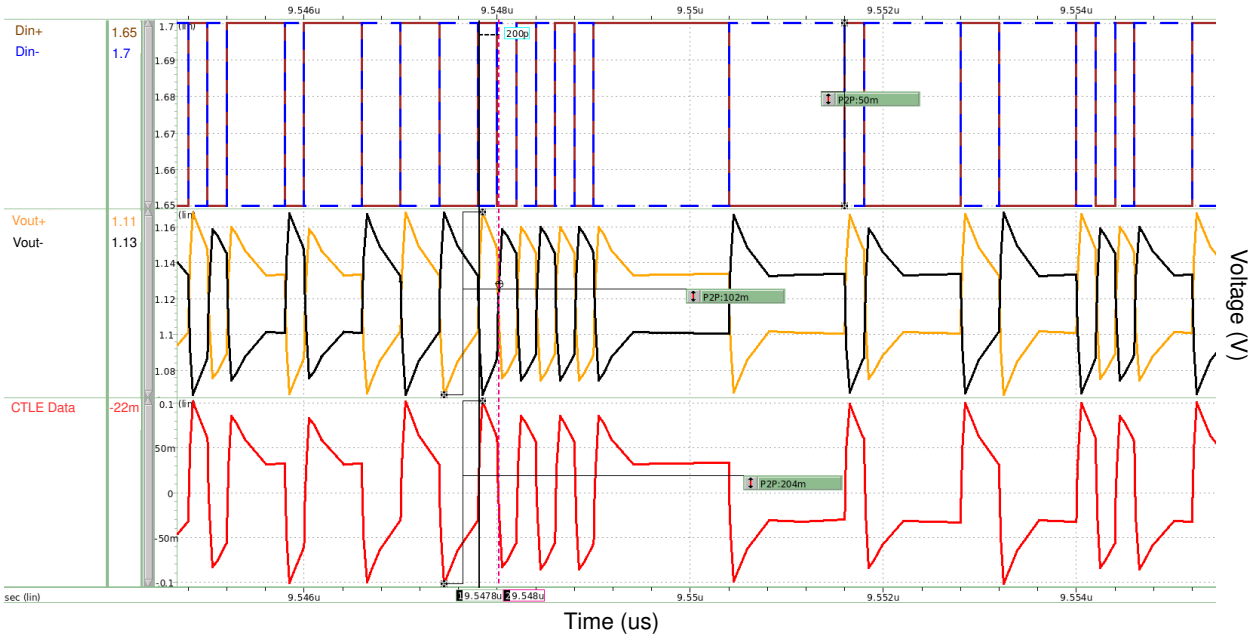
A_{DC}	$f_z\ (GHz)$	$\omega_z\ (Grad/s)$	$f_{p1}\ (GHz)$	$\omega_{p1}\ (Grad/s)$	$f_{p2}\ (GHz)$	$\omega_{p2}\ (Grad/s)$
$110 x 10^9$	0.65	4.0841	1.95	12.2522	5	31.4159

Simulation results of receiver LVDS and CTLE are observed in figure 3.13 (a) and (b), respectively. For demonstration, digital signals are injected into receiver module LVDS and these are in brown, and correspond to the reference signal, reset, and serial data. Then, serial data is transformed and modeled with the values of LVDS standard mentioned in [41, 42] to an analog signal of brown color. Then, the negative and positive signals represented in solid blue and dashed red colors are generated. As the transformation of digital to analog has implicit the signaling value to pass through the interconnect, the receiver LVDS works as a coupler. The outputs are highlighted in solid yellow and dash black.

The outputs of the receiver buffer are connected to the CTLE, where the signals of figure 3.13 (b) in solid brown and dash blue colors of the first row, represent such signals. Positive and negative signals are observed in the second row with black and yellow colors. It is



(a) Receiver LVDS simulation results at 5 Gbps



(b) CTLE simulation results at 5 Gbps

Figure 3.13: Receiver driver implementation at behavioral AMS level

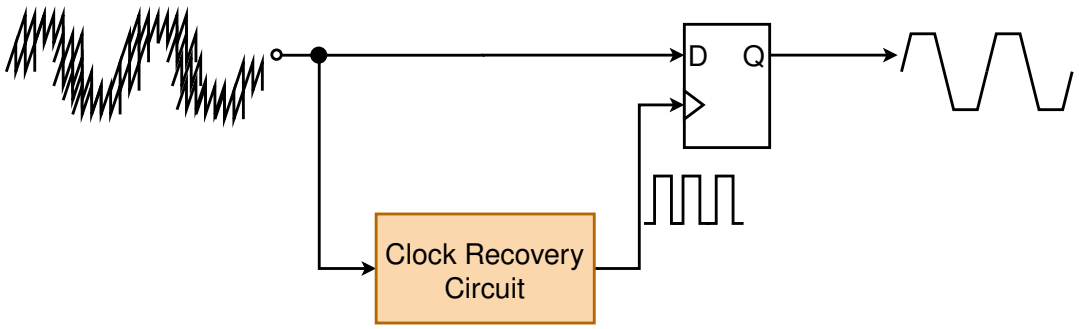
observed a comparable response as FFE when consecutive bits have the same level voltage. Only the first bit has the maximum voltage peak and the remaining bits decrease their values until a voltage change is presented. The function of the equalizer is accomplished because the differential voltage value at the input is around 100 mV, whereas for the output

observed in the red signal, the difference increases up to 200 mV. A comparator block must be performed to decide with its difference, if the next bit will be represented with a lower or higher value.

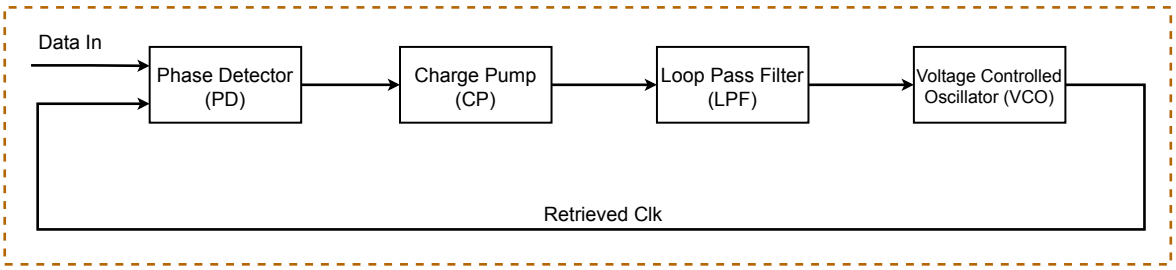
3.4.2. Clock and Data Recoverys (CDR)

The recovery of the transmitter's clock signal is required and is performed by the Clock and Data Recovery (CDR). The two main functions of the CDR are to recover the clock signal from the data stream and retimed the data with the recovery clock in order to reduce jitter [46]. The representation of CDR's purpose is observed in figure 3.14 (a); however, in this work, the implementation of CDR is avoided by the fact that a loopback is performed for the evaluation of the transceiver. Thus, it is used the same clock of the transmitter in the receiver; only the data is retimed at the output of CTLE to synchronize the stream at the receiver stage.

CDRs can be implemented by phase or delay detection of the input data regarding the reference signal generated by the CDR. An implementation with delay detection leads to jitter propagation and thereby the retimed data is not a cleaner signal. Meanwhile, phase detection enables the jitter to be significantly reduced and consequently, clearer retimed data. In figure 3.14 (b), a simplified block diagram of the CDR using phase detection is depicted.



(a) CDR simplified circuit representation. Adapted from [46]



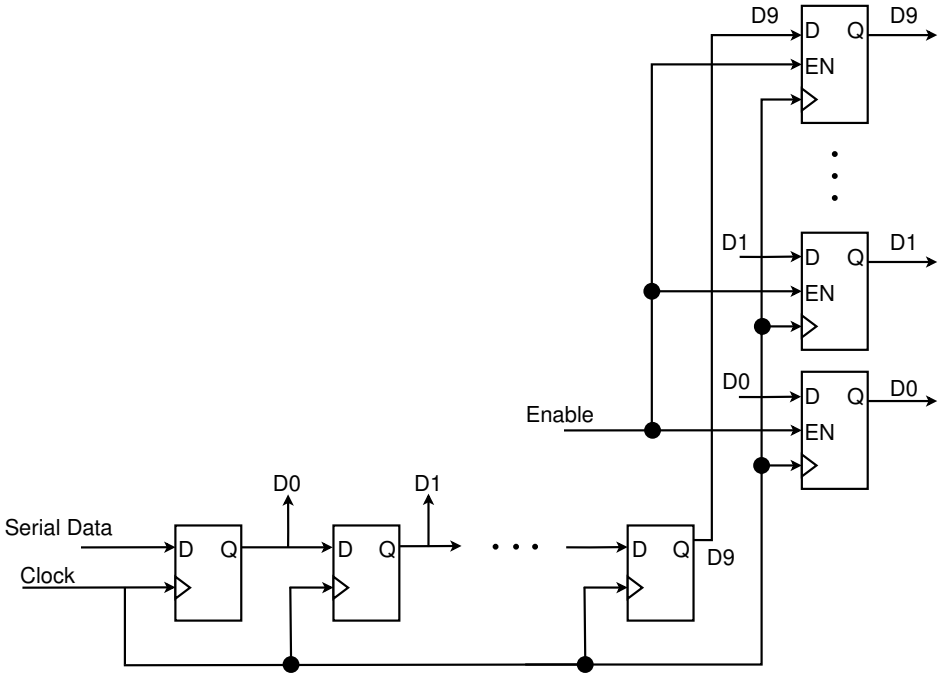
(b) Simplified block diagram of the CDR with phase detection approach

Figure 3.14: Simplified operation and block diagram of the CDR

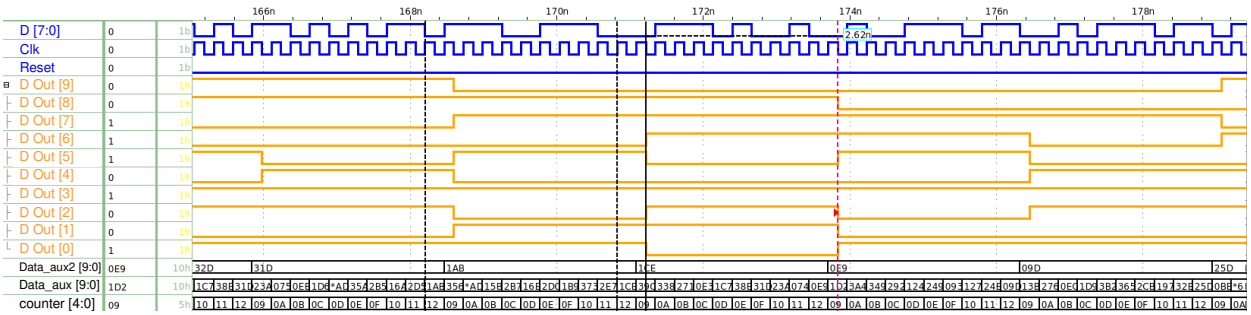
The first block is the phase detector (PD), which compares the phase between the input data and the CDR reference. This provides information to adjust the phase of the CDR clock. The charge pump (CP) is used to store energy to increase or decrease the voltage for a certain amount of time. The low pass filter (LPF) suppresses ripples that can affect the VCO. The last block is the voltage-controlled oscillator (VCO), which generates a periodic signal at the output according to the input voltage applied to it. The goal of the CDR is that the phases of the input data and the reference signal of the CDR are as similar as possible such that the receiving stage is locked [46].

3.4.3. Deserializers

Once the data is retimed, the deserialization of the data stream is performed. Therefore, the serial data changes to a parallel one of 10-bits. The behavior of the deserializer is based on figure 3.15 (a), where the data is injected in the first register and each clock cycle is propagated through the next registers [37]. When the data is on the 10 registers, these are connected to the vertical array of enable-registers. This array facilitates the holding of the 10-bit data for a whole clock cycle of the parallel reference signal. In figure 3.15 (b), is observed the response at the behavioral level. The blue signals belong to the inputs, the black ones are internal signals for producing the output and the orange one is the deserialized data. The deserialization is produced with the use of a counter that behaves as the enable signal to propagate a new deserialized data.



(a) Deserializer simplified circuit representation. Adapted from [37]



(b) Deserializer simulation result at 5 Gbps

Figure 3.15: Deserializer circuit description and event-driven simulation of its behavioral AMS model

3.4.4. K28.5 Detectors

This block performs the data alignment according to the special K-symbol. This K-symbol is the K28.5 that has a special value that the block detects and allows the data alignment. The special feature of K-symbols is that they have 5 bits in a row, either of the two logical levels available. This facilitates the detection and data alignment [47]. Nevertheless, the goal of this block is when a loopback is not performed because it must ensure the data synchronization. The implementation of this block is out of the scope of this work.

3.4.5. Elastic Buffers

The purpose of a USB transceiver is the information flow between two modules (host and hub). Each one can assume the function of transmitter or receiver by the half-duplex configuration. In figure 3.16, it is depicted a description of host and hub with their respective clock domains, where the host performs the function of transmitter and the hub the function of the receiver. The same reference signal is not the given for both devices. Hence, the transmitted reference signal is retrieved in the hub from the data stream, where the reference transmitter signal is inserted. The recovered reference signal is used to synchronize the incoming data at the receiver stage. However, this signal might have deviations by the nature of a communication system; therefore, an elastic buffer is used. The elastic buffer tolerates these deviations with data insertion in a memory block or controlling the data path of further stages to avoid data losses. The implementation of this block is detailed later.

3.4.6. 8b/10b Decoders

The reverse operation to coding is implemented with the 8b/10b decoder. In essence, the idea is the same as encoder 8b/10b. Instead of just mapping data of 8 bits to 10 bits, it is mapped a data of 10 bits to 8 bits. The same control signals as disparity and special character are used to produce the 8-bit output. The data is split into 6 and 4 bits, which represent the LSB and MSB, respectively. The previous description is observed in figure

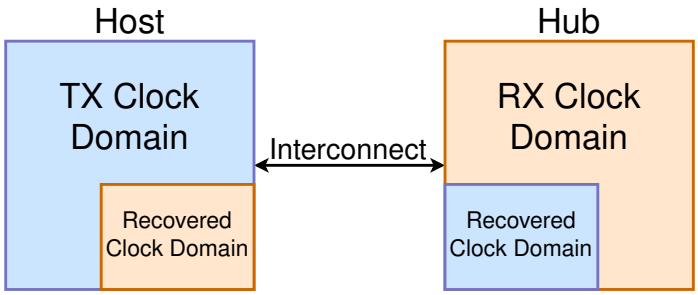
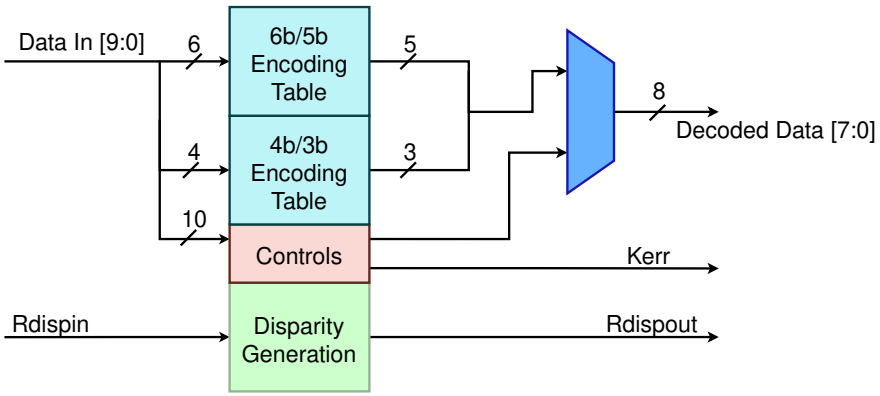


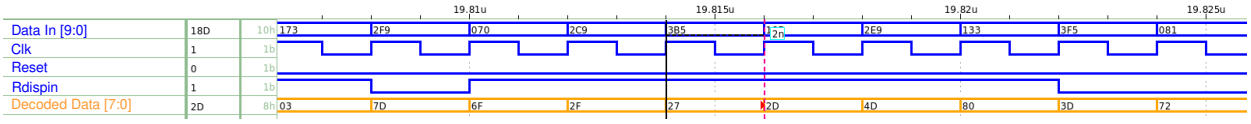
Figure 3.16: Clock domains representation of USB transceiver

3.17 (a). Each data is transformed into 5-bit and 3-bit data and then they are concatenated to generate the 8-bit data.

Simulation results are observed in figure 3.17 (b), where inputs are represented with blue signals and the output is represented with the orange signal. It can be observed the variation in parity signal is based on input data variations. The period of this block is 2 ns and transformation of 10-bit data to 8-bit data is performed. This output data is almost the original, because the remaining block that is the descrambler, converts the data into the original one.



(a) General diagram of 8b/10b decoder



(b) Simulation results of 8b/10b decoder at 500 MHz

Figure 3.17: 8b/10b decoder description and event-driven simulation of its behavioral AMS model

3.4.7. Descramblers

This block restores the originally sent data. The implementation is absolutely the same as the scrambler. This implies that the same polynomial showed in equation (2), circuits of figure 3.3 and behavioral description of scrambler are used. In figure 3.18, it is observed the

simulation response. Blue signals belong to the inputs, the black one to internal signals for producing the orange signal output. For performing this simulation, the data is scrambled and then descrambled. Once the data is scrambled, it takes two cycles to return its original value.

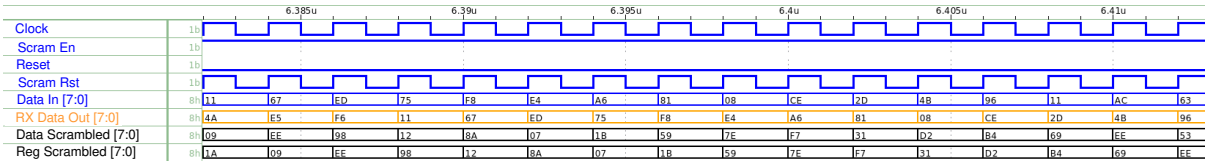


Figure 3.18: Descrambler event-driven simulation of its behavioral AMS model at 500 MHz

3.5. Transmitter of USB 2.0 Transceivers

The first part of the transceiver belongs to the transmitter, which is depicted in figure 3.19. The simplified block diagram shown in chapter 2 has 4 blocks. However, by the fact that the data is transmitted serially instead of parallel, several patterns must be generated to provide synchronization into the transceiver [48]. Therefore, this block diagram has 7 blocks and these are the serializer, Sync/EOP generators, a multiplexer that connects the previously mentioned blocks, a bit stuffer, an encoder, and a control FSM. These blocks are modeled digitally, and mixed-signal simulation is performed with the interconnect and LVDS described for USB 3.0 transceiver. Equalizers are not used because the data rate is less, and losses are not harmful. The general specifications based on USB 2.0 standard are the data rate of 480 Mbps with a bit-time of 2.08 ns for serial data, whereas for parallel data, 8 bits are used with a period of 16.64 ns.

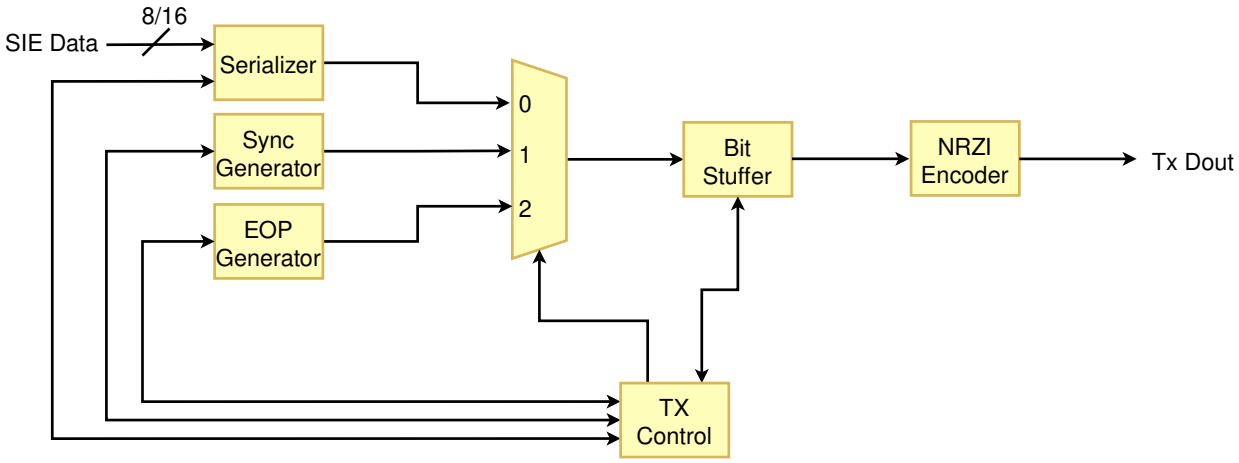


Figure 3.19: USB 2.0 transmitter simplified block diagram

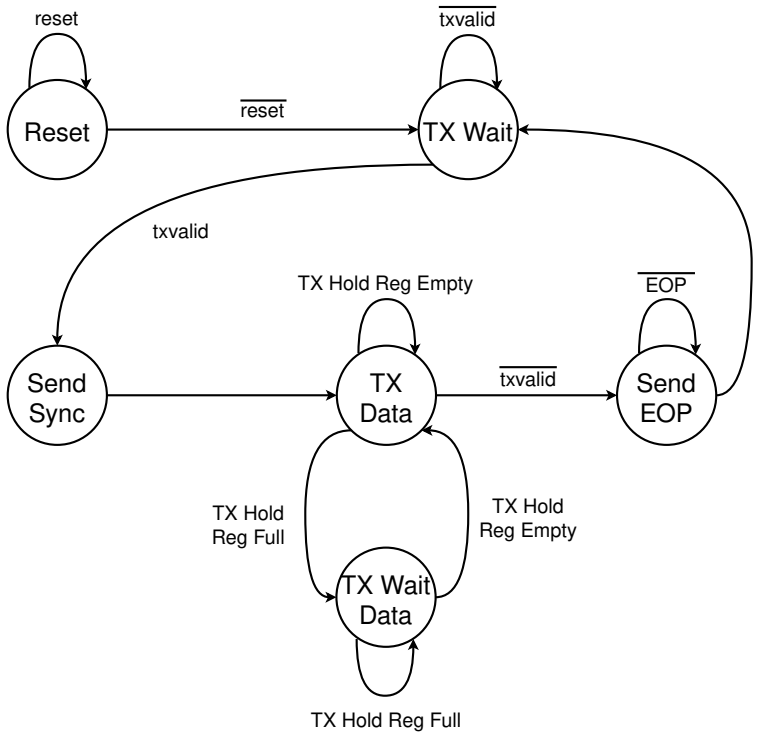
3.5.1. Transmitter Control FSM

Unlike the USB 3.0 standard, this transceiver uses a state machine to control the digital shared path and to communicate with the Serial Engine Interface (SIE) that is responsible

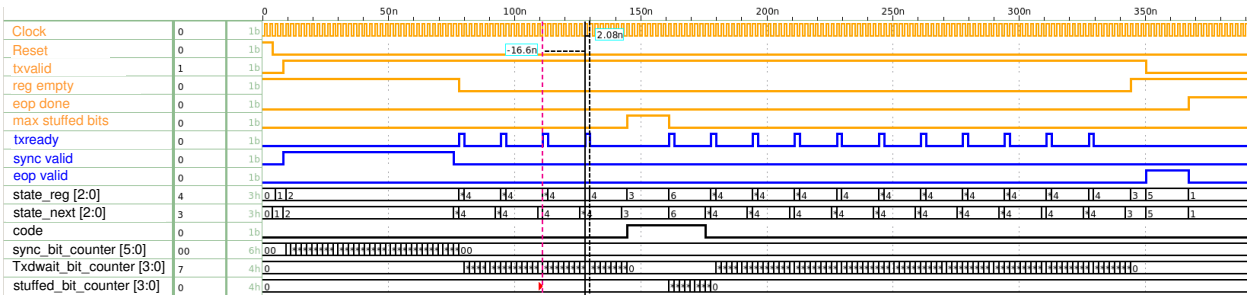
for generating the appropriate packets for the USB link. This interface is out of the scope of this work.

The state machine is composed by six states and is depicted in the figure 3.20 (a). The first state is the *Reset* state and is entered when the reset signal is asserted. Once this signal is not asserted, it goes to the *TX Wait* state, where this machine waits for the *txvalid* signal sent by the SIE. This indicates that data transmission is about to take place. When this signal is asserted, it goes to the *Send Sync* state, and is sent the Sync pattern through the digital shared path. When the sending of this pattern is finished, it switches to the *TX Data* state, where it is analyzed if new data is coming from the SIE. If the hold register is no longer empty then goes to the *TX Wait Data* state; it is waited for a byte time to return to the previous state and analyze if there is new data. During the waiting state, the present data is serialized and sent through the digital shared path [49].

The maximum number of stuffed bits is also detected, which it is 8. It occurs when the transmitter waits for a byte time for reading new data. The control machine will be transitioning in the last two mentioned states until the txvalid signal is no longer asserted and that is when it is moved to the *Send EOP* state. This state sends the end of packet (EOP) pattern and it waits for an EOP flag indicating that the pattern has been inserted into the digital shared path. Thus, it is returned to the initial wait state waiting for the assertion of the txvalid signal by the SIE again.



(a) State diagram of the transmitter FSM. Adapted from [49]



(b) Transmitter control FSM simulation results at 480 MHz

Figure 3.20: Transmitter control FSM state diagram description and implementation at behavioral level

The simulation of the digital path control of the transmitter is shown in figure 3.20 (b). The orange signals correspond to the inputs, the blue ones to the outputs, and the black ones to internal signals that allow the monitoring of the FSM. With the assertion of the txvalid signal, it is passed to the sending state of the Sync pattern. It will be a time of 32 bits that is the length of the high-speed link, and the same time is the *sync valid* signal asserted. Once the sending of the Sync pattern is finished, it will be switching between the data and waiting states that indicates the reading and conversion of the parallel to serial data sent from the SIE. The txready signal is asserted every byte time, indicating to the SIE that the data was consumed by the transmitter. The scenario where the maximum number of stuffed bits is reached is also observed, which implies that a byte time must be waited to be able to continue reading. If the txvalid signal is no longer asserted it goes to the EOP pattern sending state. This indicates the end of data transmission, wherein this state it remains for a byte time and then returns to the waiting state, in expectation of txvalid assertion by the SIE.

3.5.2. Sync Generators

Channel synchronization is achieved with different sent patterns through the channel. The first sent pattern is the *Sync*, which indicates the start of data transmission. For the high-speed link the pattern consists of 32 bits, where the first and last bits have a logic level of 0 and the remaining bits have a logic level of 1. The bit stuffer is disabled when this pattern is sent. The simulation response of this block is shown in figure 3.21, where the blue colored signal corresponds to the output pattern that is injected into the digital shared path. The first and last bit have a logic value of 0 and the remaining ones a logic level of 1, this when the sync valid orange color signal has been asserted.

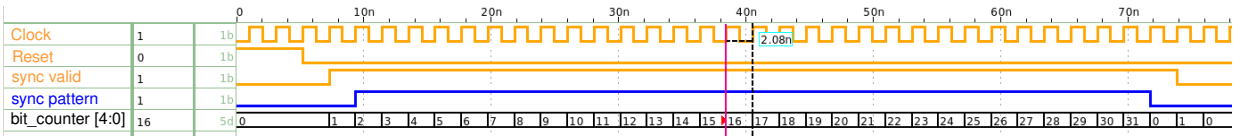


Figure 3.21: Sync generator simulation results at 480 MHz

3.5.3. EOP Generators

The completion of the data transmission is done with the *EOP* pattern. For the high-speed link, the error detected by the bit unstuffer module at the receiver is used. The error is presented when there are more than 6 consecutive bits with a logical value of 1; hence, it must be reported as an error, but for this link, it means that the data reception has been completed. The first bit of the pattern has a logic level of 0 and then 7 consecutive bits with a logic level of 1, in which the bit stuffer is disabled in the same way as for the Sync generator. The simulation response of this block is depicted in figure 3.22, where the blue colored signal corresponds to the output pattern that is injected into the digital shared path. The first bit has a logic value of 0 and the remaining ones a logic level of 1, this when the EOP valid orange color signal has been asserted. Once the pattern is completely sent, the assertion of *eop done* occurs. Thus, indicating to the FSM that this pattern has been already injected.

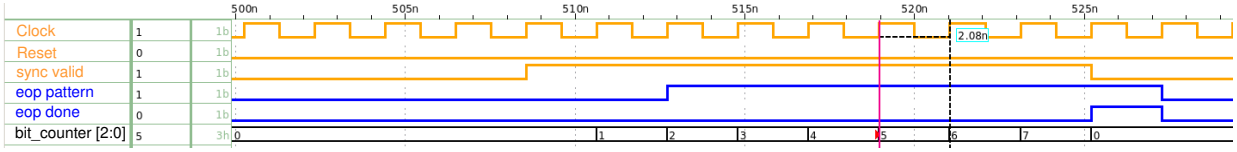
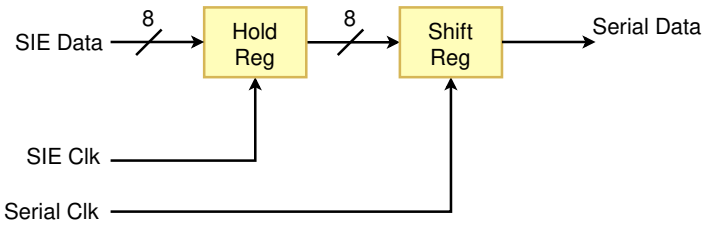


Figure 3.22: EOP generator simulation results at 480 MHz

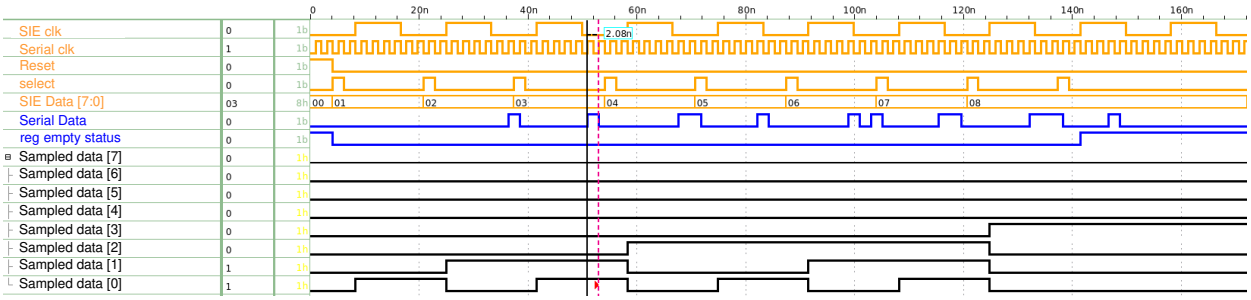
3.5.4. Serializers

The serialization block is integrated by two elements shown in figure 3.23 (a). The first one consists of the *Hold Register*, which is responsible for sampling the data coming from the SIE, as well as determining whether or not there is a data update. The second block is the *Shift Register*, which is responsible for serializing the sampled data [50]. The implementation of this serialization is performed in the same way as the serializer block of the previous USB transceiver. The selector signal that is the one that pushes the data, comes from the control machine.

The serializer simulation is observed in figure 3.23 (b). The signals in orange correspond to the inputs, which include the SIE data, parallel clock, serial clock, reset, and selector that allows the shifting of the loaded data in the Hold register. The blue signals correspond to the serial output and the data comparator to indicate to the FSM whether the Hold register



(a) Serializer simplified block diagram. Adapted from [49]



(b) Serializer simulation results at 480 MHz

Figure 3.23: Serializer block diagram description and event-driven simulation of its behavioral AMS model

is empty or not. For simplicity of explanation, the serialization of an ascending numeration starting at 0 as the first value and 8 as the last value is performed, where the serialized value of these numbers is observed. The data in black color corresponds to the register that samples the SIE data.

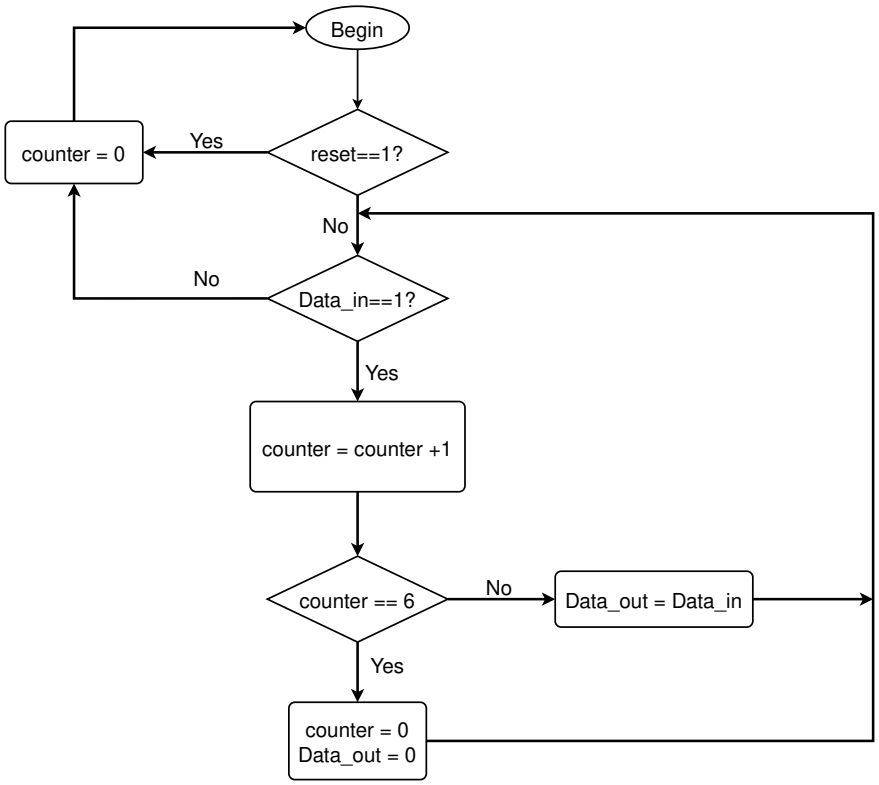
3.5.5. Bit Stuffers

In order to achieve an adequate number of transitions, to ensure that the CDR can be synchronized, the bit stuffer is used. This block inserts a logic value of 0 at the output when 6 consecutive bits with a logic value of 1 have been read. A total of 8 inserted bits are allowed. Once the limit is reached, a byte time must be waited to read data again. In figure 3.24 (a), it is depicted the flow diagram for determining whether a bit with a value of 0 should be inserted after 6 consecutive bits with high-level values have been read. For special patterns as Sync and EOP, this block is disabled [51].

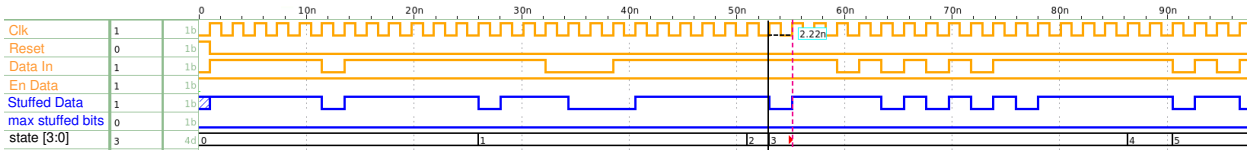
The simulation result of this module is observed in figure 3.24 (b), where the signals in orange are the inputs that correspond to the clock, reset, serial input data, and the enable that is given by the control FSM. The outputs correspond to the serial data stream, where the additional bits are inserted, as well as the signal indicating that the maximum number of bits that can be inserted has been reached, these are in blue. It is observed at the output response that for every 6 consecutive bits with a logic level of 1, a bit with a logic level of 0 is inserted. The number of inserted bits is monitored by the signal state in black color.

3.5.6. NRZI Encoders

The last digital block of the transmitter corresponds to the encoding. An NRZI type is used, where for each high value detected at the input, the state line is inverted. Otherwise, the state line remains unchanged [52]. In figure 3.25 (a), the operating principle of this encoder is shown; it can be observed that the black color output is preserved when a zero is detected on the blue color pattern. The output of this block is connected to an analog output buffer that is the LVDS. The differential output is generated and connected to the



(a) Bit stuffer flow diagram. Adapted from [51]

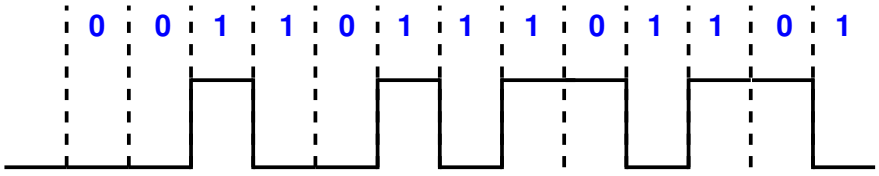


(b) Bit stuffer simulation results at 480 MHz

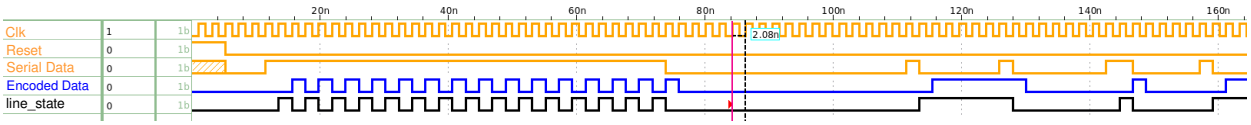
Figure 3.24: Bit stuffer flow diagram description and event-driven simulation of its behavioral AMS model

PCB link by the LVDS, that goes to the receiver.

The encoding is observed in the figure 3.25 (b). The orange signals correspond to the inputs, which are the reset, clock, and serial input data; the blue signal corresponds to the encoded output, whereas the black signal is the state line that is used internally to determine whether or not to change the state based on the input. At the beginning of the simulation,



(a) Functioning principle of the NRZI encoder



(b) NRZI Encoder simulation results at 480 MHz

Figure 3.25: NRZI encoder description and event-driven simulation of its behavioral AMS model

the input with a logic value of 1 is prolonged for several clock cycles, and this at the output is a changing transition since the level is toggled. Once a logic level of 0 is detected, the output remains at the same level as in the previous cycle.

3.6. Receiver of USB 2.0 Transceivers

A simplified block diagram of the receiver is depicted in figure 3.26, with the addition of a detector for a specific pattern. It has seven blocks, which are the CDR, elastic buffer, Sync detector, NRZI encoder, bit unstuffer, deserializer, and the control FSM [53]. Each block is explained below and modeled at a behavioral level. The same scenario of the transmitter is presented in the receiver in terms of a digital simulation is performed on the whole path. Nevertheless, this stage is connected along with the transmitter and interconnect, to perform a mixed-signal simulation. The same specifications are applied for the receiver with a data rate of 480 Mbps and a bit-time of 2.08 ns for serial data, whereas for parallel data, 8 bits are used with a period of 16.64 ns. Most of the block works in an inverse manner of transmitter since the data must be recovered to an 8-bit data.

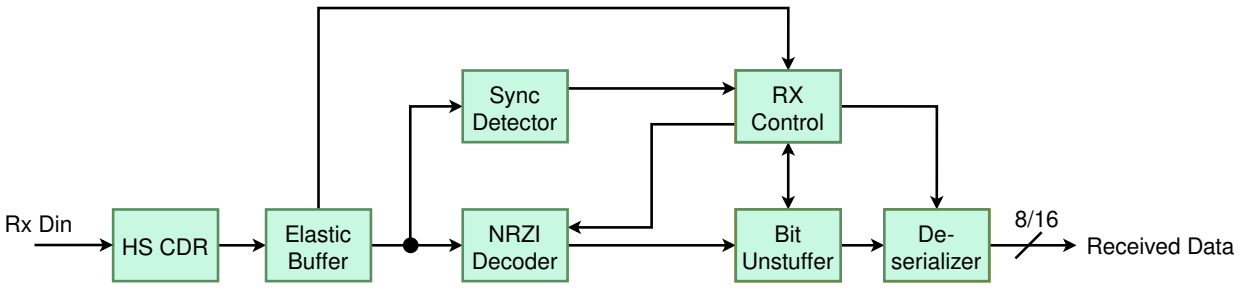


Figure 3.26: USB 2.0 receiver simplified block diagram

3.6.1. High-Speed Clock and Data Recoverys (HS CDR)

The recovery of the transmitter's clock is required, and it is performed by the Clock and Data Recovery (CDR) for the high-speed link. In the previous USB transceiver is explained the function of this block as well as the block diagram. In this work, the implementation of this CDR is avoided by the loopback implementation, and the same reference signal is used for the transceiver.

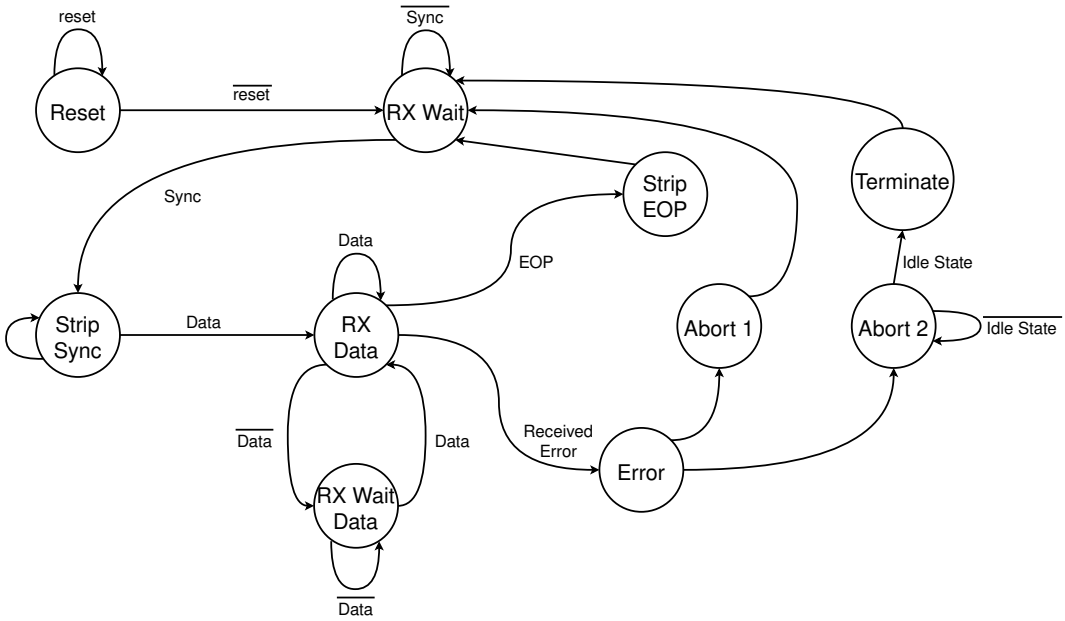
3.6.2. Elastic Buffers

In the previous USB transceiver, the functionality of the elastic buffer is detailed, which for this transceiver can be applied the same operating principle. The implementation of this block is detailed later.

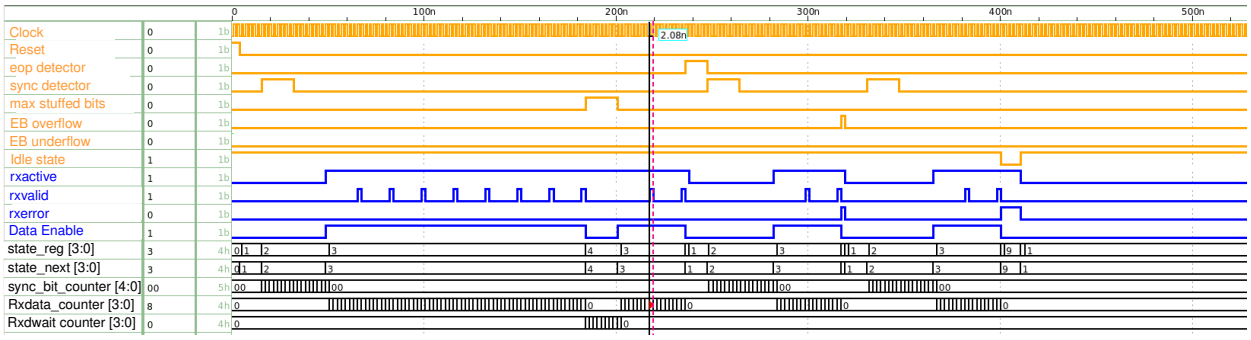
3.6.3. Receiver Control FSMs

The receiver state machine is responsible for controlling the digital shared path. The state diagram is shown in figure 3.27 (a), which has 10 states. The first state is *Reset*, that is when this signal is asserted. Once it is no longer asserted, it goes to the *RX Wait* state, which waits for the assertion of the *Sync* signal coming from the Sync detector. If the received pattern is the Sync one, then data reception is going to occur. As soon as this happens, it goes to the *Strip Sync* state, which indicates to the SIE of the receiver that data will start to be received with the assertion of the *rxactive* signal. Then it goes to the *RX Data* state.

The RX Data state performs the reading of the serial data as well as the conversion to the original data that was sent from the transmitter. Since these are byte-sized conversions, it implies that every 8 cycles the *rxvalid* signal is asserted to tell the SIE that the converted data is valid. It will move to the *RX Wait Data* state when the maximum number of stuffed bits has been reached, which is eight. As the same procedure as in the transmitter, the machine must wait for a byte time to be able to continue reading data [54]. The next state corresponds to the *Strip EOP* and this occurs with the error detection (data completion for



(a) State diagram of the receiver FSM. Adapted from [49]



(b) Receiver control FSM simulation results at 480 MHz

Figure 3.27: Receiver control FSM state diagram description and implementation at behavioral level

high-speed link) of the bit unstuffer module, which is when there are more than 6 consecutive bits with a value of 1. After data completion has been detected, it is indicated to the SIE that the data reception has been terminated with the non-assertion of the rxactive and rxvalid signals [49, 55].

The following states correspond when there is an error in the data reception and this can occur firstly by an underflow/overflow condition of the elastic buffer, wherein the state diagram would have to go through the states *Error* and *Abort 1*. The second scenario is presented when there is an error in the analog front-end module that is out of the scope of this work. This error is presented with the data reception and the voltage levels that should be presented depending on the speed at, which the data is being received. If this is the case, it goes through the states *Error*, *Abort 2* and *Terminate*.

The simulation of the digital path control of the receiver is depicted in figure 3.27 (b). The orange signals correspond to the inputs, the blue ones to the outputs, and the black ones to internal signals that allow the monitoring of the FSM. It is observed that with the assertion of the Sync detection signal, it is passed to the strip sync state, where it will be held until is finished the detection of the Sync pattern. Once the pattern is already detected, it is passed into the data state, and the rxactive is asserted for the whole reception. The rxvalid is asserted every 8 cycles, indicating the reception of new data. The data reception is paused when the maximum number of stuffed bits is reached, and at this moment the FSM goes into the data wait state. The data reception is finished when the EOP signal is detected, and it is moved into strip eop state to indicate to the SIE that data reception is over. Furthermore, it is observed the scenarios when an error is presented either by an elastic buffer error or an idle state error. In both errors, the rxerror is asserted and the data enable is not asserted.

3.6.4. Sync Detectors

This block detects the Sync pattern from the output data of the elastic buffer. At each positive clock edge, the data is read, where a different logic level must be detected at each

edge, in other words, toggling between 0 and 1. A state machine is used to read the data, which merely detects that the sequence is oscillating between the two logic levels. If the same logic level is detected on two positive clock edges, the state machine is restarted to detect the complete 32-bit pattern. The simulation of this block is observed in figure 3.28, where the input pattern is the output decoded from the transmitter. When this pattern is detected, the sync detection signal is asserted and control FSM is moved into strip sync state, where pattern detection is completed at this state.

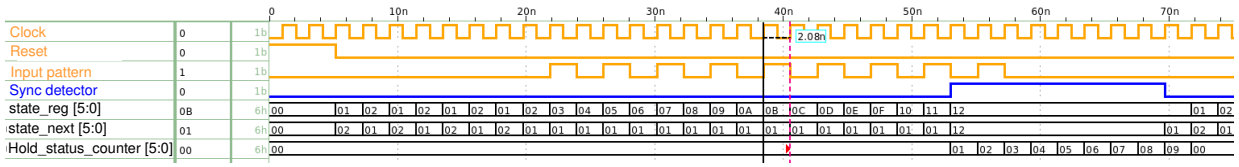


Figure 3.28: Sync detector simulation results at 480 MHz

3.6.5. NRZI Decoders

The decoding block is performed by the NRZI decoder. The function is based on the implementation of an XOR of its present value with the previous value [56]. With this, the original input pattern is recovered. Its simulation result is observed in figure 3.29, where the orange serial input data is encoded and the encoding is observed in the black color signal. The decoding is observed in the blue signal, where this signal is the same presented to the original.

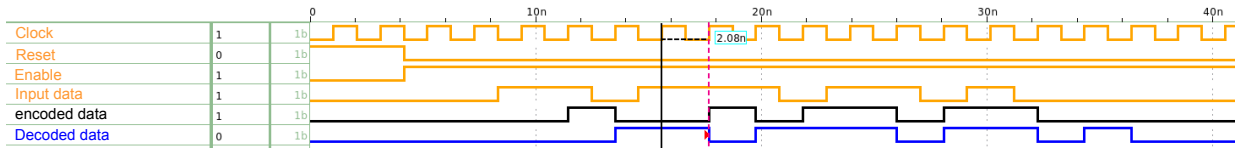
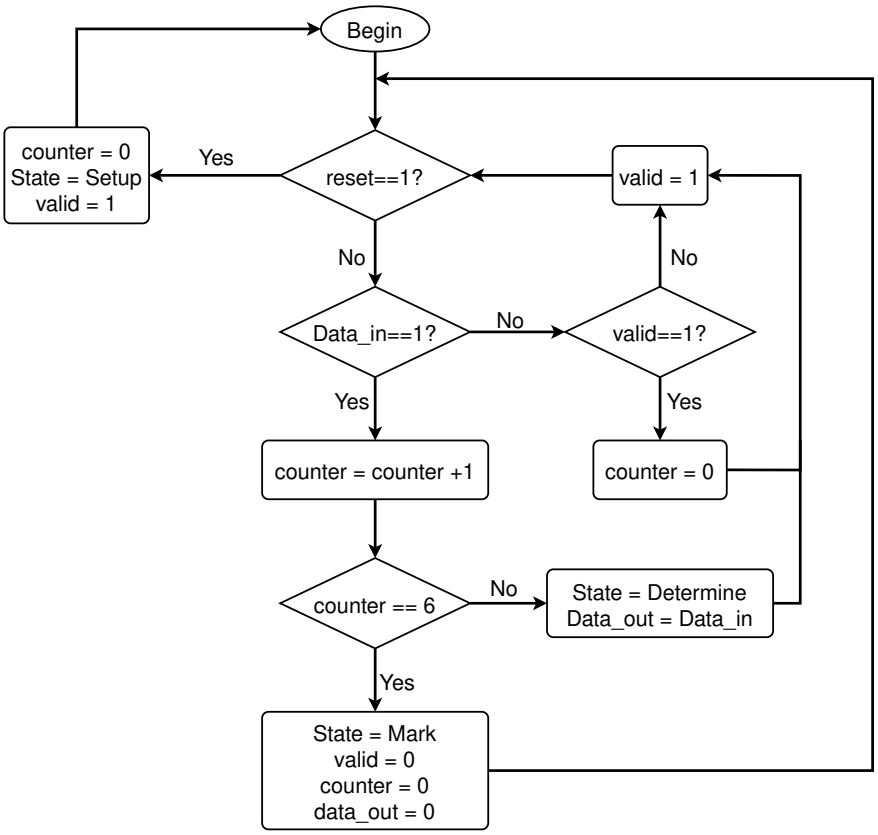


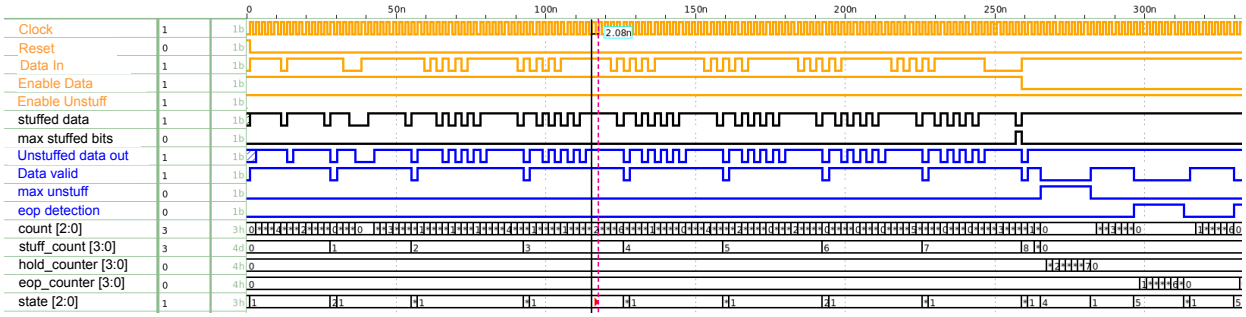
Figure 3.29: NRZI encoder simulation results at 480 MHz

3.6.6. Bit Unstuffers

The detection of the inserted bits on the stream of data for recovering the clock at the receiver is carried out by the bit unstuffer. The function of this block is to indicate, which bits should not be read by the deserializer. The flow diagram is observed in figure 3.30 (a). It shows how this block determines whether a bit should be discarded [51]. Initially, in the diagram it is detected whether the reset signal is high, if so the counter is assigned with a value of 0, the status is set to the *Setup* state, and the signal called *valid* with a value of 1. When the reset is at a low level, it is passed to the detection of the input data, in which if it is a high level, a value of 1 is added to the counter and compare if the counter is at a value of 6. If the counter reaches 6, it means that the next bit to be read is not valid for the deserializer, which makes the valid signal to be set to a low value, that is when it is in the *Mark* state. If the input data has a low level, then it goes to the *Determine* state where



(a) Bit unstuffer flow diagram. Adapted from [51]



(b) Bit unstuffer simulation results at 480 MHz

Figure 3.30: Bit unstuffer flow diagram description and implementation at behavioral level

it is until the counter reaches a value of 6 that implies that six consecutive bits have a high logic level.

The simulation result is observed in figure 3.30 (b). The original pattern is stuffed in order to detect, which bits are stuffed. The serial input data is presented in orange, the stuffed pattern in black color and the blue signals are the outputs. These are the serial data, the valid signal that indicates, which bits are stuffed, the assertion of the maximum number of stuffed bits signal, and the signal that indicates the finalization of data reception.

It is observed in the simulation that the valid signal is not asserted when stuffed bits are inserted, this implies that these bits will be discarded by the deserializer.

3.6.7. Deserializers

The last block before retrieving the original data sent corresponds to the deserializer. This is the same as the one implemented for the USB 3.0 standard; the main difference is that instead of deserializing the data in 10-bit words, 8 bits are used. The valid signal coming from the bit unstuffer is used that allows, which data must be shifted to generate the 8-bit word. This means discarding the inserted bits that are not from the original frame of the transmitter. The simulation result is presented in figure 3.31, where the only output is the deserialized data in blue color. It is shown that when the no assertion of the valid signal, the current data is held one more clock cycle and it takes an additional bit-time to convert new data from serial to parallel. The input data that is marked between the two black vertical lines, it is the data that is deserialized and load at the output when 8 bits are shifted, and the red vertical line shows the content of the new output data.

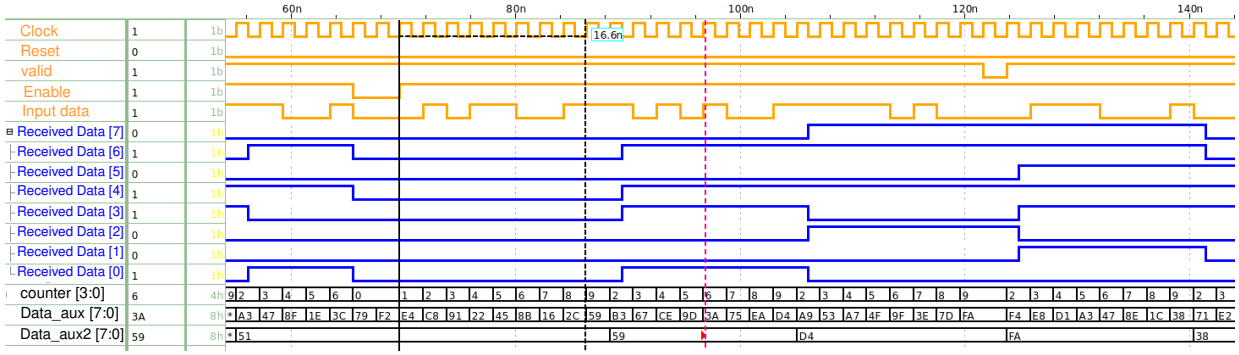


Figure 3.31: Deserializer simulation results at 60 MHz

4 Elastic Buffer Design

This chapter deals with the implementation of Elastic Buffers for the USB 3.0 and 2.0 transceivers. The modeling at RTL and requirements for each implementation are shown. Furthermore, the digital flow synthesis that is used for both implementations is detailed. Simulation results are shown where metrics as power, area, read-time are discussed.

In chapter 2 the literature related to the elastic buffer was presented , where two families are mentioned, synchronous and asynchronous. For a USB link, an asynchronous topology is used to pass the data. The half-full method is chosen over the handshaking methodology caused by further stages are independent of the EB. It must be recalled that half-full topology aims to hold the buffer depth at the half with the use of special symbols for USB 3.0 standard. In the USB 2.0 standard, the buffer must reach a threshold to allow the data transfer.

4.1. Modeling and Requirements for the Elastic Buffer

The implementation of both elastic buffers is presented in this section. In the manner that is implemented each EB is discussed as well, it is mentioned the minimum requirements that USB standards demand. The implementation of both buffers is at high-level description on Verilog language and they are synthesized following a framework on Synopsys.

4.1.1. USB 3.0 Standard

The main requirements for the design of an elastic buffer are the deviation in parts per million (ppm) and the max payload of the transfer. The USB 3.0 standard indicates that the max payload is 1024 bytes with a deviation of ±5600 ppm. The depth buffer calculation is performed by the aforementioned requirements, and the buffer depth is calculated as follows [21, 29]:

$$Frequency\ difference = \frac{|\pm 5600|}{1x10^6} = 0.0056\, clock\ cycles \tag{9}$$

A shift in the buffer position occurs each:

$$1\ shift = \frac{1}{0.0056}\, clock\ cycles = 179\, clock\ cycles \tag{10}$$

The maximum shift amount when the max payload is transferred is:

$$Shift\ amount = \frac{1024}{179} = 6\, bytes \tag{11}$$

Hence, the elastic buffer must tolerate shifts of ±6 positions, which implies a minimum depth of 12 positions. Although it is chosen a depth of 16 by the fact that it is used a Gray-code scheme for the address generator, which allows 1-bit transition per clock cycle. Thus, if it is used the minimum depth of 12 and it is restarted the address to the beginning in the circular memory, then more than one bit would change in a clock cycle. In figure 4.1, it is depicted a simplified block diagram of the EB for USB 3.0 standard. It is constituted by seven main blocks. Their main functions are the address generator for write and read operations, as well as the comparison between pointers to compute if the buffer is at half of its depth, or if it is below or above of the half state. It is observed how the system is operated by two clock domains, where the blocks with light gray color use the recovered clock from the CDR, whereas the blocks with dark gray color use the local clock domain of the receiver. The functionality of each block is detailed below.

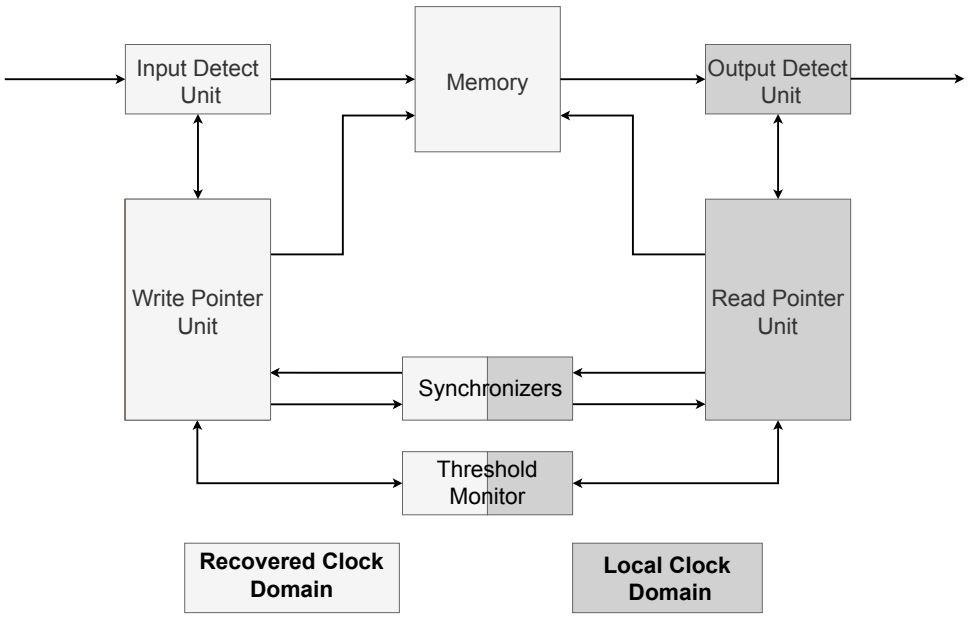


Figure 4.1: Simplified block diagram of Elastic Buffer for USB 3.0

The input data is received by the *Input Detect Unit* whose main function is the detection of SKP symbols to indicate to the *Write Pointer Unit* that an SKP symbol is asserted. If an END symbol is asserted, the enable signal is deactivated in the Write Pointer Unit for avoiding the continuous writing into the *Memory* block. In figure 4.2, is observed the block diagram of the aforementioned block where three signals are received, which are input data, clock, and reset signals. The outputs are the sampled data that goes into the memory, SKP detection, and enable signal that they go to the Write Pointer Unit.

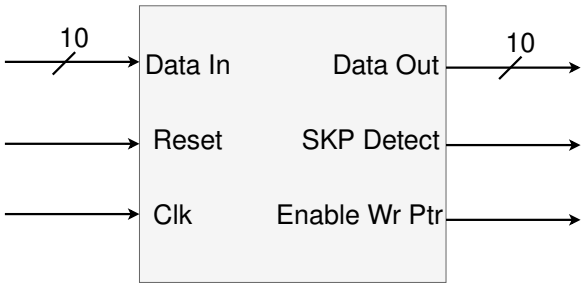
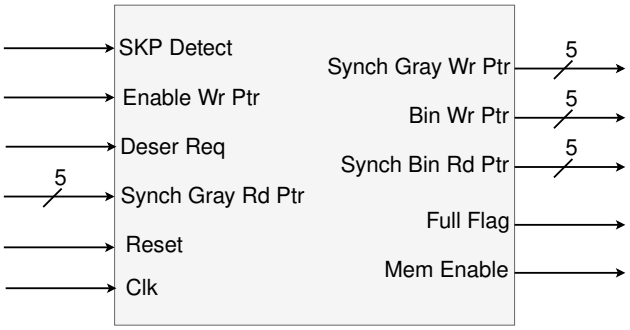


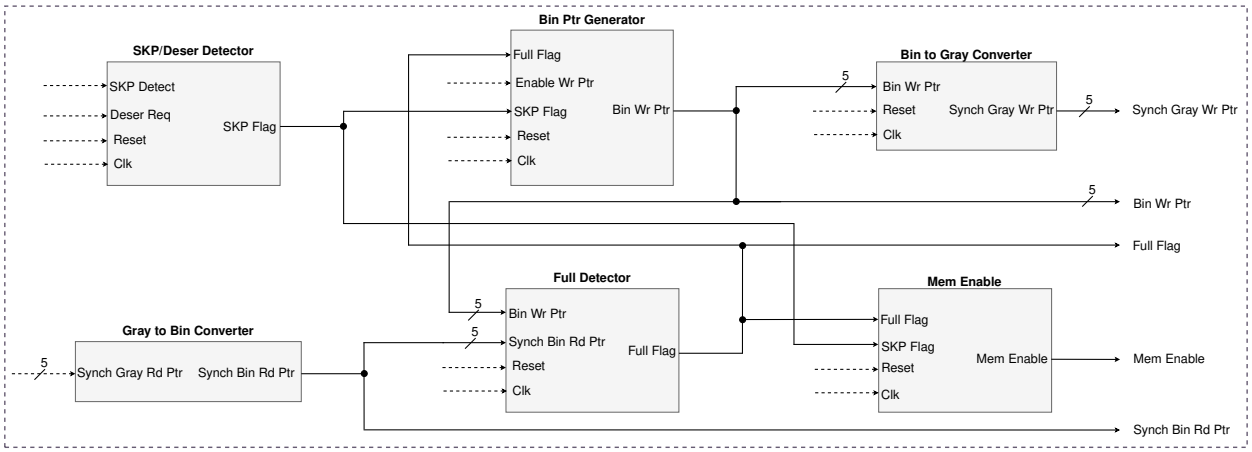
Figure 4.2: Block diagram of the Input Detect Unit of the Elastic Buffer for USB 3.0

The next block is the Write Pointer Unit that it is depicted in figure 4.3 (a) with its simplified block diagram. Their inputs are the SKP detection and enable signals, both provided by the Input Detect Unit. The input that is called desertion request is provided by the *Threshold Monitor*, the 5-bit read address with Gray-code scheme comes from the *Synchronizers Unit*; finally clock and reset signals are used by this block. The outputs of this unit are the three 5-bit data, which are the binary and gray representation of the write address that go to the Memory and Synchronizers respectively, as well as the synchronized binary read address, which goes to the Threshold Monitor. The full flag signal is used to indicate to the system if memory is full or not. The writings into the memory are permitted by the memory enable signal.

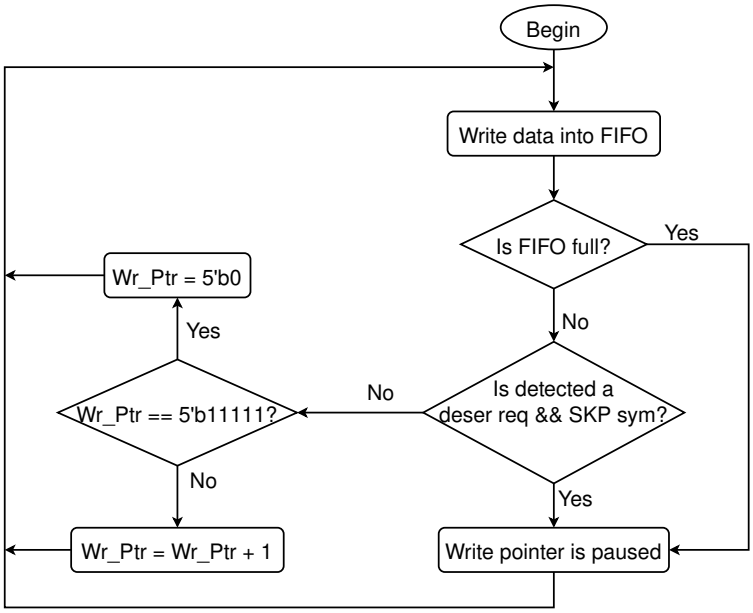
Figure 4.3 (b) details the simplified block diagram of the Write Pointer Unit, which is constituted by six blocks. The *SKP/Deser Detector* indicates to the *Binary Pointer Generator* if the current address is held one more clock cycle. This occurs when an SKP symbol is asserted and the memory has more data than its half, which implies having more than 8 data on the memory block. The conversion from gray representation to binary is performed by the *Gray to Bin Converter*. The binary address is generated by the Binary Pointer Generator that uses conditions as if the FIFO is full, and if the enable and SKP flag signals are asserted to generate a new address or to preserve the current one. The *Full Detector* is used for comparing the read and write addresses to determine if the memory is full. The *Memory Enable* is performed for allowing new writing into memory. The last block is the *Binary to Gray Converter* whose function is to convert the binary address to a



(a) Top block diagram of Write Pointer Generator



(b) Detailed block diagram of Write Pointer Generator



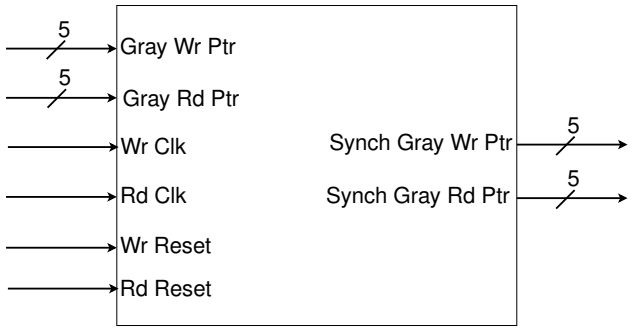
(c) Flow diagram for the desertion operation

Figure 4.3: Write Pointer Generator of the Elastic Buffer for USB 3.0

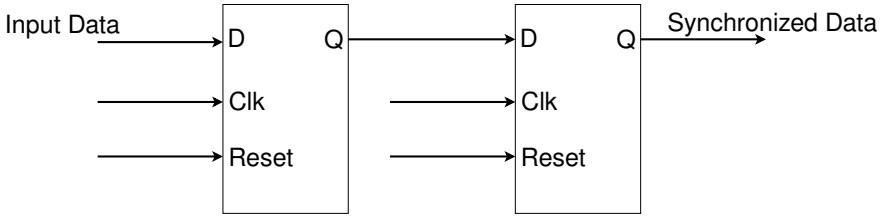
Gray-code representation, which will be used to compare along with the read address of the *Read Pointer Unit* if the memory is empty.

The flow diagram for the desertion operation is observed in figure 4.3 (c); it is tracked if after the writing into memory, it is already full or not. If the memory is not full, then it is detected if the desertion signal by the Threshold Monitor and SKP symbol by the Input Detect Unit is asserted. If both signals are asserted the address pointer is held by one more cycle, otherwise the address is changed either to add 1 bit to its current address or if it must return to zero.

Synchronizers are used to synchronize data with its opposite clock domain to compare in both address generator units, either the memory is full or empty. In figure 4.4 (a), a simplified block of this unit is depicted. Two 5-bit addresses from both address generator units are received. Both clock domains as well both reset signals are used. The outputs of this block are the synchronized address in a Gray-code scheme for both addresses. The schematic implementation of the synchronizer is shown in figure 4.4 (b); the first register might enter to metastability state, and thus, a second register is used to mitigate such problems [57, 58]. The schematic view is referred to only one bit, but it is applied to all the bits for each address. The less significant bit is the one that toggles more times by the principle operation of the Gray-code scheme.



(a) Simplified block diagram of Synchronizers



(b) Schematic implementation of Synchronizers

Figure 4.4: Synchronizers of the Elastic Buffer for USB 3.0

The simplified block diagram of the Threshold Monitor is observed in figure 4.5. The input signals are the synchronized binary addresses and binary addresses of both pointer units; both clock and reset signals are used in this block. The outputs are the desertion and addition requests for the write and read pointer units, respectively. The generation of each request is performed from the difference between their respective binary and synchronized binary addresses. The desertion request is asserted when the difference is higher than 8 and the addition request is asserted when the difference is less than 8.

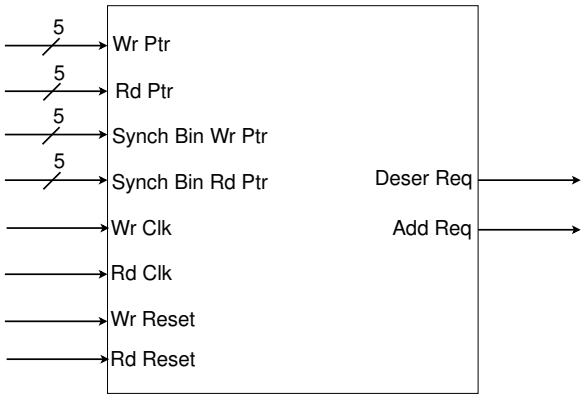
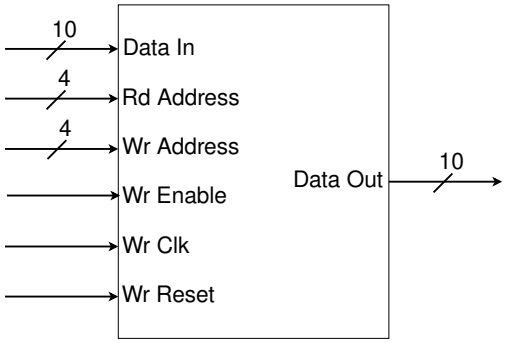


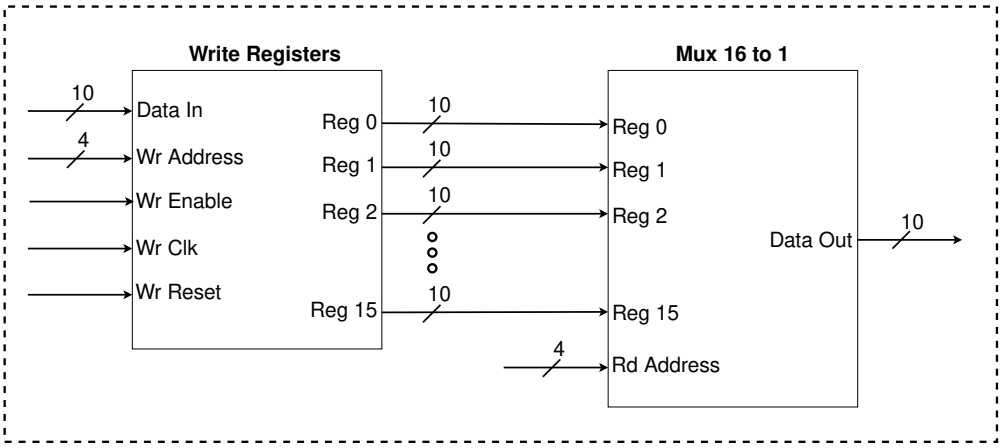
Figure 4.5: Block diagram of the Threshold Monitor Unit of the Elastic Buffer for USB 3.0

The input data is stored and loaded by the *Memory Unit*. The simplified and detailed block diagram is observed in figure 4.6 (a) and 4.6 (b), respectively. Their inputs are the data from the output of the Input Detect Unit, both binary addresses from write and read pointer units, the enable that allows to write into memory and clock and reset signals from write domain. For writing into the memory, the write address is used along with the enable the signal to select one register of 16 possible. The output is performed when a read address is given. For making the lecture of a specific register, the read address is used as a mux selector. The 16 registers are connected into the mux, and the output is connected to the *Output Detect Unit*.

The next block is the Read Pointer Unit that it is depicted in figure 4.7 (a) with its simplified block diagram. Their inputs are the SKP detection provided by the Output Detect Unit, the addition request that is provided by the Threshold Monitor, the 5-bit write address with Gray-code scheme comes from the Synchronizers Unit; lastly clock and reset signals are used by this block. The outputs of this unit are the three 5-bit data, which are the binary and Gray representation of the read address that go to the memory and Synchronizers respectively. The synchronized binary write address goes to the Threshold Monitor; also, empty flag signal is used to indicate to the system that memory is empty. The insert flag is used at Output Detect Unit when the buffer is below of its half for balancing.

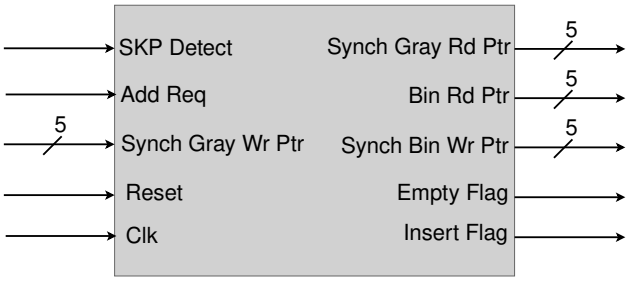


(a) Top block diagram of Memory Unit

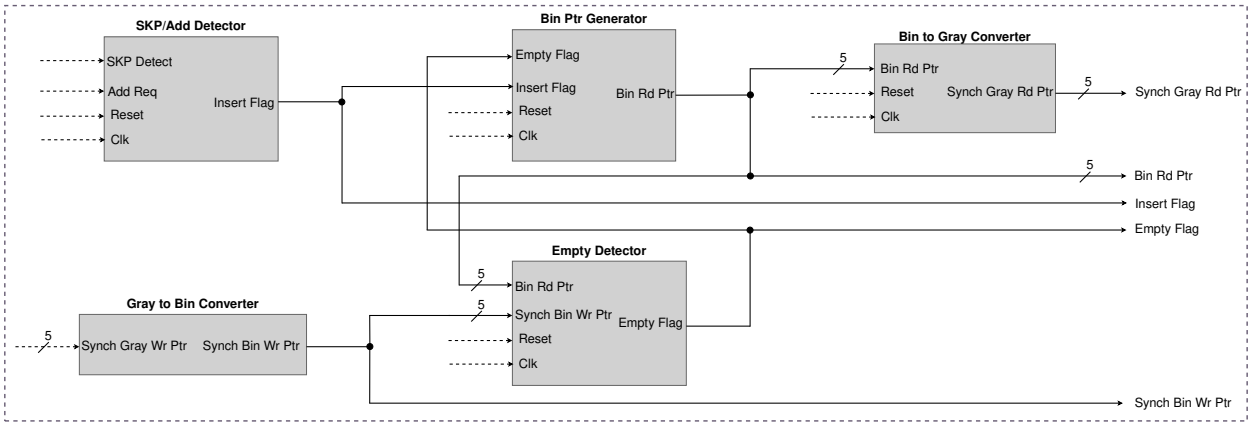


(b) Detailed block diagram of Memory Unit

Figure 4.6: Memory Unit of the Elastic Buffer for USB 3.0



(a) Top block diagram of Read Pointer Generator



(b) Detailed block diagram of Read Pointer Generator

Figure 4.7: Read Pointer Generator of the Elastic Buffer for USB 3.0

In figure 4.7 (b), it is detailed the simplified block diagram of the Read Pointer Unit, which is constituted by five blocks. The *SKP/Add Detector* indicates to the Binary Pointer Generator if the current address is held one more clock cycle. This occurs when an SKP symbol is asserted and the memory has less data than its half. The conversion from Gray

representation to binary is performed by the Gray to Binary Converter. The binary address is generated by the Binary Pointer Generator that uses conditions as if the memory is empty, and if the insert flag signal is asserted to generate a new address or to preserve the current one. The *Empty Detector* is used for comparing the read and write address to determine if the memory is empty. The last block is the Binary to Gray Converter, which its function is to convert the binary address to a Gray-code address that will be used to compare at the Write Pointer Unit if the memory is full.

The block that puts the data read from memory at the output is the Output Detect Unit, which its block diagram is depicted in figure 4.8. The SKP detection is performed by this block and is obtained when the data read from memory is an SKP symbol. The local clock domain and read reset are used by this block, as well as the empty and insert flags. When the empty signal is asserted, the output data is the END character given by the USB standard. The insert flag is used when this one has been asserted and the memory is not empty; hence, at the output data is put an SKP symbol while the memory is filled with more data in the current clock cycle, in order to reach its half-full state.

Figure 4.8: Block diagram of the Output Detect Unit of the Elastic Buffer for USB 3.0

The flow diagram for applying an addition operation is observed in figure 4.9. At the output is put the END symbol if memory is empty, otherwise is analyzed if an insert flag is asserted. If this occurs, an SKP symbol is put at the output. If the previous condition does not occur, then a reading from memory is performed. The next condition to analyze is if the addition request and SKP symbol are asserted or not. When both signals are not asserted, the read address pointer is changed either to add 1 bit in its current position or if the address is returned to zero. Otherwise, the address is held one more clock cycle.

4.1.2. USB 2.0 Standard

The main requirements of elastic buffer for USB 2.0 standard are quite similar in the manner that frequency deviation and max payload must be taken into consideration for the buffer depth estimation. For this standard, the deviation and max payload are ±1000 ppm and 9644 bits, respectively. The buffer depth is calculated as follows [21, 29]:

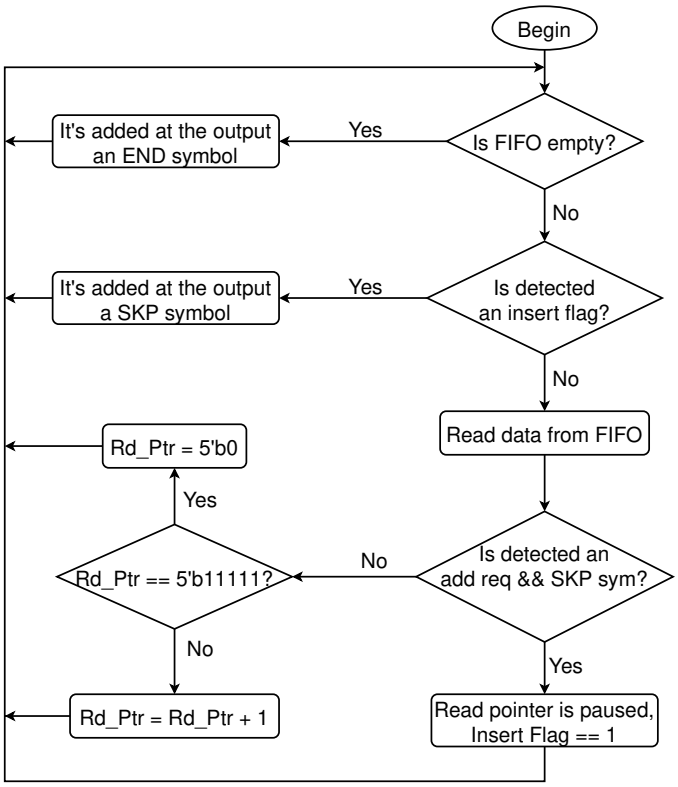


Figure 4.9: Flow diagram for the addition operation

$$Frequency\ difference = \frac{|\pm 1000|}{1x10^6} = 0.001\, clock\, cycles \tag{12}$$

A shift in the buffer position occurs each:

$$1\ shift = \frac{1}{0.001}\, clock\, cycles = 1000\, clock\, cycles \tag{13}$$

The maximum shift amount when the max payload is transferred is:

$$Shift\ amount = \frac{9644}{1000} = 9.644\, bits \tag{14}$$

Hence, the elastic buffer must tolerate shifts of ±10 positions, which implies a minimum depth of 20 positions; however, for ensuring timing margin since special symbols are not used for this standard, a minimum depth of 24 must be used [9]. Nevertheless, it is chosen a depth of 32 because the same drawback in terms of bit transitions of the USB 3.0 version is presented on this standard. In figure 4.10, is depicted a simplified block diagram of the EB for USB 2.0 standard; it is constituted by four main blocks, where their main functions are

the address generator for write and read operations. The system is operated by two clock domains. The blocks with light gray color use the recovered clock from the CDR, whereas the blocks with dark gray color use the local clock domain of the receiver. The functionality of each block is detailed further.

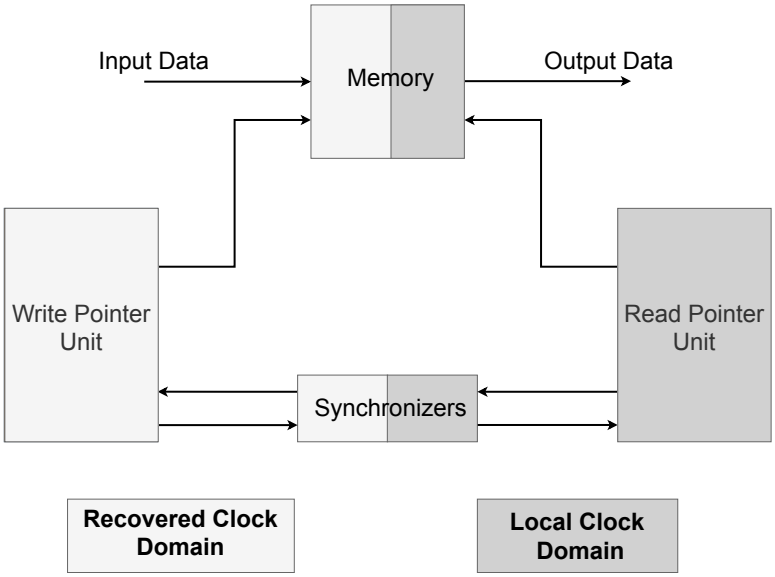
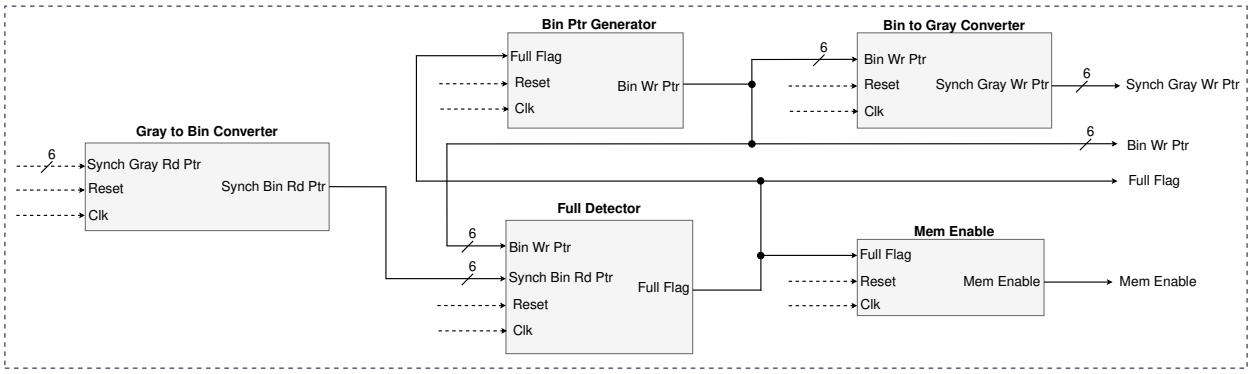


Figure 4.10: Simplified block diagram of Elastic Buffer for USB 2.0

The *Write Pointer Unit*, depicted in figure 4.11 (a) with its simplified block diagram, is used for address generation of the memory as well as to monitor whether the memory is full or not. Their inputs are the 6-bit read address with Gray-code scheme from the *Synchronizers Unit*, and clock and reset signals. The outputs of this unit are the two 6-bit data, which are the binary and gray representation of the write address that go to the Memory and Synchronizers, respectively. Full flag signal is used to indicate to the system if memory is full or not. The writings into the memory are generated by the memory enable signal.

In figure 4.11 (b), it is detailed the simplified block diagram of the Write Pointer Unit. The main difference respect the 3.0 version is the absence of the block, which works with SKP symbols. Overall, the unit works at the same manner in terms of address generation, and calculation of the memory occupancy.

(a) Top block diagram of Write Pointer Generator



(b) Detailed block diagram of Write Pointer Generator

Figure 4.11: Write Pointer Generator of the Elastic Buffer for USB 2.0

The synchronizers used by this standard are the same as the 3.0 version in terms of inputs/outputs and operation. The difference is that the address width is one bit wider to cover all memory locations. In figure 4.12, the block diagram is observed in which are also considered the metastability problems that are solved in the same way as for the 3.0 version.

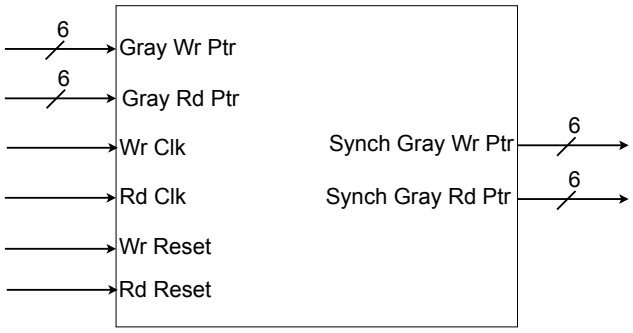
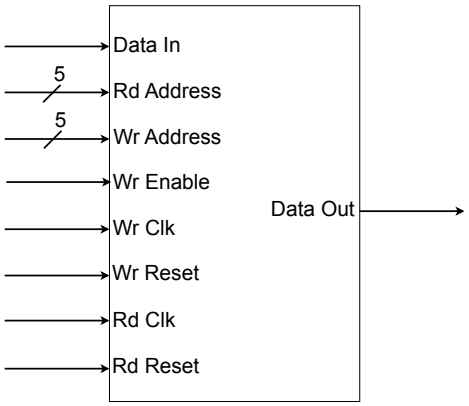


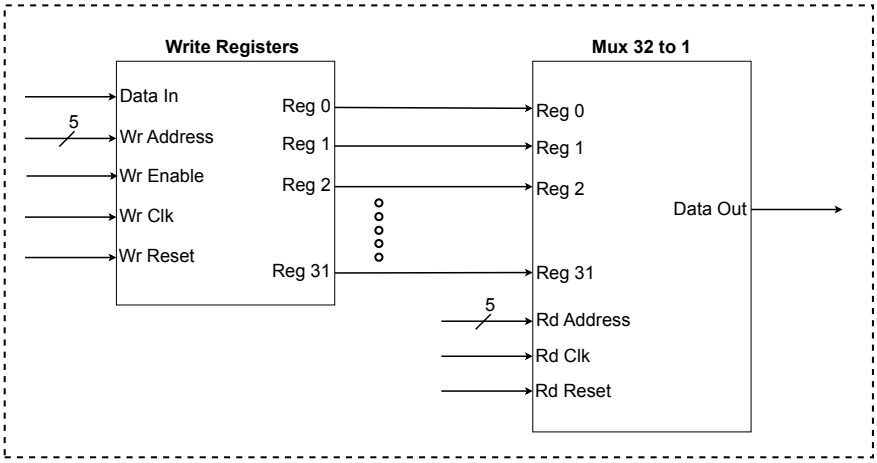
Figure 4.12: Simplified block diagram of Synchronizers for USB 2.0

The input data is stored and loaded by the *Memory Unit*. The simplified and detailed block diagram is observed in figure 4.13 (a) and 4.13 (b), respectively. Their inputs are the input data, binary addresses from write and read pointer units, the enable signal that allows writing into memory and clock and reset of both domains. For writing into the memory, the write address is used along with the enable the signal to select one register of 32 possible. For making the lecture of a specific register, the read address is used as a mux selector, and the 32 registers are connected into the mux. The read data is connected directly to the output, in comparison to the USB 3.0 version that it is detected if the read data is a special symbol.

The last block is the Read Pointer Unit that it is depicted in figure 4.14 (a) with its simplified block diagram. Their inputs are the 6-bit write address with a Gray-code scheme that comes from the Synchronizers Unit and clock and reset signals. The outputs of this unit are two 6-bit data, which are the binary and Gray representation of the read address that go to the memory and Synchronizers respectively. An empty flag signal is used to indicate



(a) Top block diagram of Memory Unit

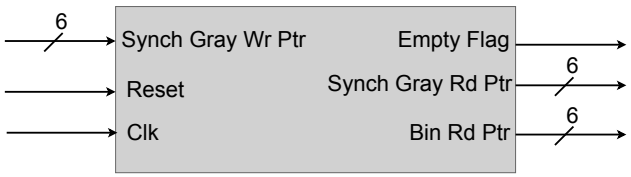


(b) Detailed block diagram of Memory Unit

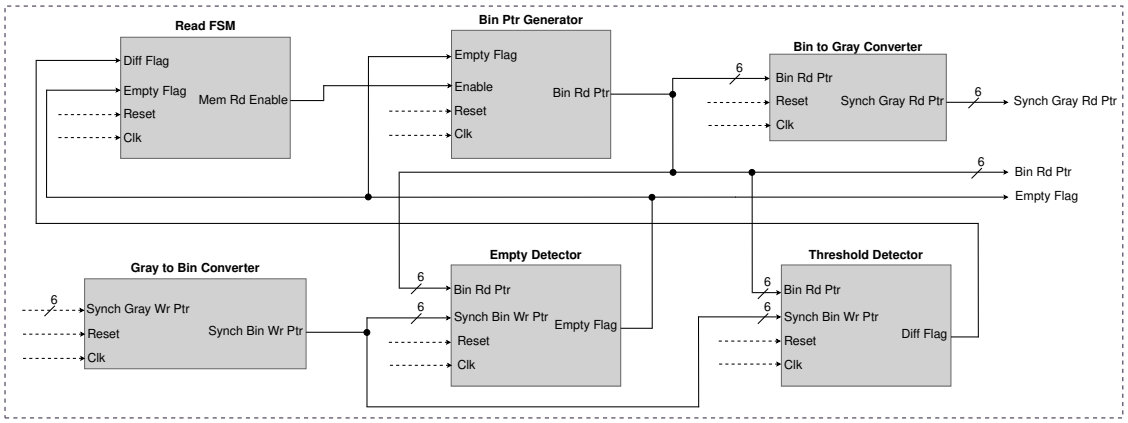
Figure 4.13: Memory Unit of the Elastic Buffer for USB 2.0

to the system that memory is empty.

The simplified block diagram of the Read Pointer Unit that is constituted by six blocks is depicted in figure 4.14 (b). The *Read FSM* is used to enable readings from the memory, which occurs when a threshold is reached. It is produced when the difference between read and write pointers is larger than the half-depth buffer. The conversion from Gray to binary representation is performed by the Gray to Binary Converter. The binary address is generated by the Binary Pointer Generator that uses conditions as if the memory is empty,



(a) Top block diagram of Read Pointer Generator



(b) Top block diagram of Read Pointer Generator

Figure 4.14: Read Pointer Generator of the Elastic Buffer for USB 2.0

and if the threshold difference has been reached. The *Empty Detector* is used for comparing the read and write address to determine if the memory is empty. The *Threshold Detector* is used to indicate whether there is a 16-position difference between the read and write addresses. The last block is the Binary to Gray Converter, which its function is to convert the binary address to a Gray-code address that will be used at the Write Pointer Unit.

The state diagram of the FSM that is used in the read pointer unit, it is depicted in figure 4.15. Three states are performed by this FSM. The first state is the reset state and it is passed to the next state represented as number 1 when the reset signal is not asserted. The difference of 16 positions between the read and write addresses is detected by the second state. When the difference is asserted, it is passed to the third state represented as number 2. In this state, readings are performed, and only an underflow condition will return to the second state for waiting by a 16 position difference again. It must be recalled that only one time is detected the difference between addresses.

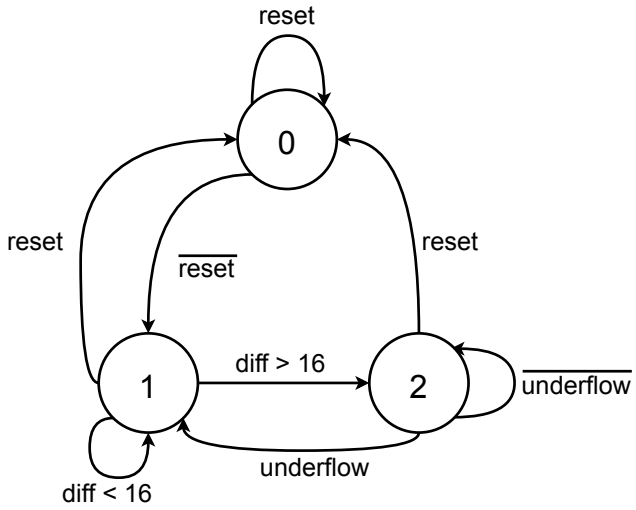


Figure 4.15: FSM state diagram for reading operation for USB 2.0

4.2. Digital Flow Synthesis

Nowadays, the generation of a digital circuit up to a physical level is a complex task caused by the number of cells that an implementation uses. An alternative for this task is the use of automated processes for digital designs. This section deals with the digital flow synthesis that is used in this work. The front-end and back-end are explained, where essentially both elastic buffers are implemented by the same flow with different considerations as would be timing. The implementation of each EB was explained in the previous section, where it is included the explanation of each block that composes each buffer. This section is focused on the use of a digital design flow to consolidate the generation of an elastic buffer at integrated circuit level.

In figure 4.16, is observed a simplified digital flow diagram for the synthesis of an integrated circuit. It is included two main stages, which are the front-end and back-end. In the front-end is included the generation of the RTL of the system to implement and the gate-level synthesis. The back-end is focused on the generation of a physical view to fabricate an integrated circuit. The framework that is used in this work is based on [59, 60].

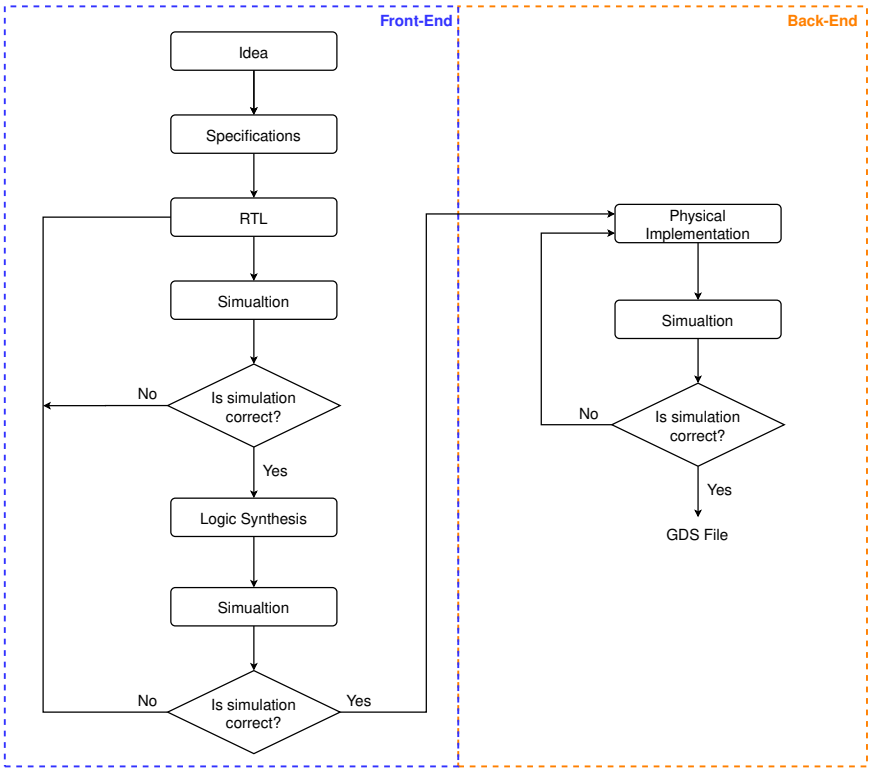


Figure 4.16: Flow diagram for a digital flow synthesis

4.2.1. Front-End

Based on figure 4.16, the front-end has several stages of the design. The first one is the idea to implement that for this work is the elastic buffer. Once the idea has been consolidated, should include the requirements that must have the circuit to implement, which for this work are the buffer depth, as well as the operating frequency that is detailed later. One important stage is the high-level description by RTL using Verilog language. The functionality and description of the system are developed in this stage. The goal of a high-level description is to model a circuit without the use of specific cells of a given library.

For verifying if the RTL description is correct, the system is simulated. It is checked if the system performs the description given by the designer. For this work, it is analyzed if all data is at the output, which means that the data is stored and read from the memory. Furthermore, the control signals are checked, which are the addition and desertion signals for the EB for USB 3.0 standard that indicates to the system that is not at its half-full state. It is monitored the empty and full signals that indicate to the system that an overflow/underflow condition is presented. If the previous description is not fulfilled, then the RTL should be redesigned to converge the simulation at RTL level as it is observed in the decision chart in the front-end section in figure 4.16.

The next step on the front-end stage is the gate-level synthesis that maps the described circuit into a specific library. For performing such synthesis, constraints by the designer must be settled. These constraints are related to delay, period, and loads. In table 4.1 is observed different values related to different parameters of timing parameters. These are the clock period, the uncertainty for setup and hold times, latency, and transition. All of them are defined from the USB standard. The activity factor is configured with a value of 25% for the activity ratio and a 50% for the static probability [59].

Table 4.1: Timing constraints for the Elastic Buffer

Standard	**Clock Period (ns)**	**Uncertainty Setup (ns)**	**Uncertainty Hold (ns)**	**Transition (ns)**	**Latency (ns)**
USB 3.0	2.615	0.075	0.075	0.1	0.075
USB 2.0	2.0812	0.104	0.104	0.15	0.104

Other constraints that must be defined by the designer are the delay at the inputs and outputs of the cells. These definitions allow the tool to achieve a realistic analysis. The input and output delay for the USB 3.0 standard have values of 0.496 ns and 0.075 ns, for max and min values respectively. For the USB 2.0 standard, the input and output delays have values of 0.104 ns and 0.285 ns, for min and max values respectively. For modeling the load that each cell can drive, it must be defined, which cell will be used as a reference for the tool's calculations. Therefore, the technology that is used for this work is the *DCELLS HD* of XFAB for a 180-nm process. This selection is based on the fact that this family is faster than others as *DCELLS HDLL* that have lower leakage, and they are slower. Since

the implementation is for high-speed links, faster cells must be used. Finally, a fanout of 10 is set for each cell.

Once the constraints are defined, it is verified that on the logical synthesis there are no time, load, or design rule violations that the tool internally uses as verification parameters. If the synthesis is successful, the tool provides messages that there are no violations. Since it is a high-speed design, the time edge becomes a major issue. The most critical paths have a high load of combinational logic, which causes these signals to arrive after the time estimated by the tool. If the previous assumption happens, a time violation arises.

Critical paths are improved by using optimization techniques through commands such as *retime*, which allows the use of a retiming algorithm to reduce the delay on a given path. There are other techniques such as *timing effort* and *area effort* that perform a more exhaustive optimization intending to reduce the most critical paths to achieve design convergence in terms of timing [61, 62, 63, 64]. Within the logical synthesis, it is checked if the response of the system is the same as the high-level simulation. In case this is fulfilled, the flow continues with the back-end stage.

4.2.2. Back-End

In this section, the physical implementation of the elastic buffer is detailed. This stage is based on figure 4.16 on the back-end section. The floorplanning, placement, routing, and clock tree synthesis are included in the back-end. It is taken the model that was synthesized in the front-end stage, that is a description at gate-level with the mapped library technology. It is included the clock tree synthesis at this stage. This means that the clock is modeled with realistic delays and it is not assumed an ideal propagation.

It must be ensured that the power domains do not have large drops and that the area is as compact as possible to avoid unwanted delay increases in the nets. In the same manner as in the front-end, timing convergence is the most difficult edge to achieve with no violation. The delays in critical paths are increased by the clock tree synthesis. Optimization techniques are used to reduce the delay and to achieve a positive slack timing path. Such techniques can be performed from different edges as placement, routing, and clock tree optimization. Changes in size cells and proximity between interconnect cells are performed by placement. The goal of routing is to avoid the delay increase in the nets, hence it is sought that it should be as less loaded as possible with parasitics. The clock optimization seeks to balance the clocking paths for avoiding mismatches through such paths [65].

Another tool that is used in the back-end for verification is *Primetime*. The delay with all placed cells that includes front-end and clock-tree cells are modeled by Primetime. It is analyzed all critical paths with the contribution of all net and cell delays. Such analysis allows to get the violation time and to improve in several ways such as size changes on specific cells for reducing the cell delay. The buffer insertion permits to decrease the load of

a specific net. All these changes that are performed in Primetime must be included in the physical synthesis, since Primetime is an auxiliary tool that does not belong to the used one for making the physical synthesis. With all these techniques, timing convergence is met.

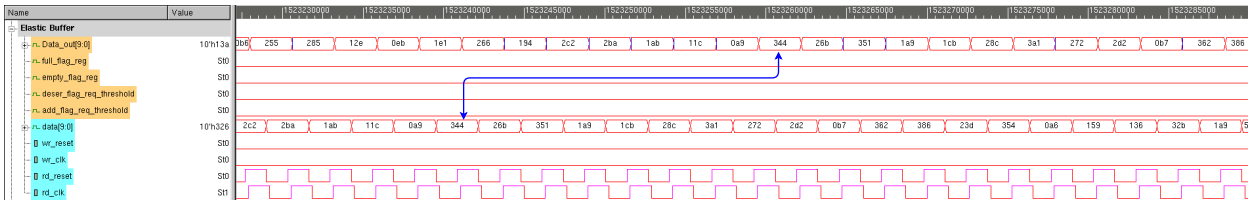
Once, there is no violation in timing, DRC, and LVS, it can be extracted a final model, which its response is simulated and checked. If the simulation is the desired one, then it can be said that implementation is completed, which means that it can send it to fabricate, otherwise it must be iterated again since the beginning of the back-end or beyond as the worst scenario.

4.3. Simulation Results

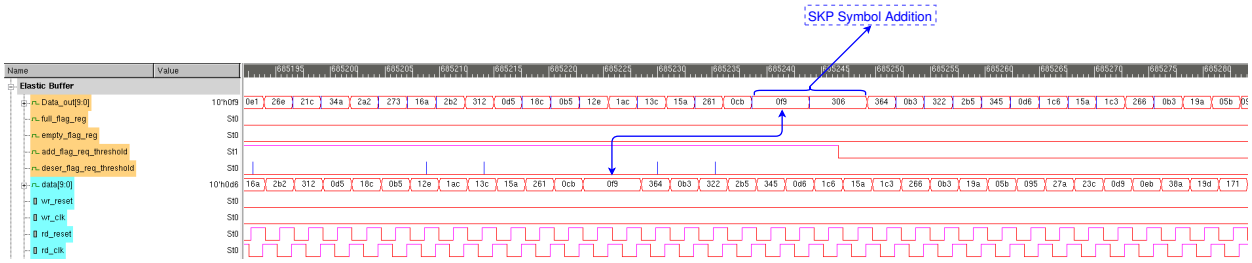
The results related to the EB are shown in this section. There are three main operating states for the EB for USB 3.0 standard. The first one consists of its normal operation that occurs when the buffer is at its half depth. The second and third operating states are when the buffer is either below or above its half memory capacity. Thus, the use of SKP symbols are necessary when they are detected, to balance the memory at its half-full state. The use of two clock domains are presented in the EB, and deviations in the clocks are inserted. This allows to detect the functionality of the buffer in terms of its occupancy. The clocks are deviated more than the mentioned cycles by the standard, for generating more shifts than usual in the memory occupancy. In figure 4.17, it can be observed the aforementioned operating states. The signals that are highlighted with the orange and blue color belong to the output and input, respectively. In figure 4.17 (a), the EB is at its middle depth, because the difference in terms of data positions between the data that are signaled with blue arrows is eight positions. The add/deser flag requests are not asserted, which means the half-full state of the buffer.

In figure 4.17 (b), the addition state is depicted. The data position difference between the marked data is below eight positions, then a deviation between clock signals arises. In order to compensate the buffer, SKP symbols are added. The SKP symbol with a value in hexadecimal representation could be either $0F9_h$ or 206_h, and with this symbol at the output, more SKP symbols can be added. It is observed on the marked blue chart, that is indicated the addition of data that does not belong to the original data stream. Furthermore, when the SKP symbol is added it can be observed that after some clock cycles, the addition request is not asserted, which means that the EB is returned to its half-full state.

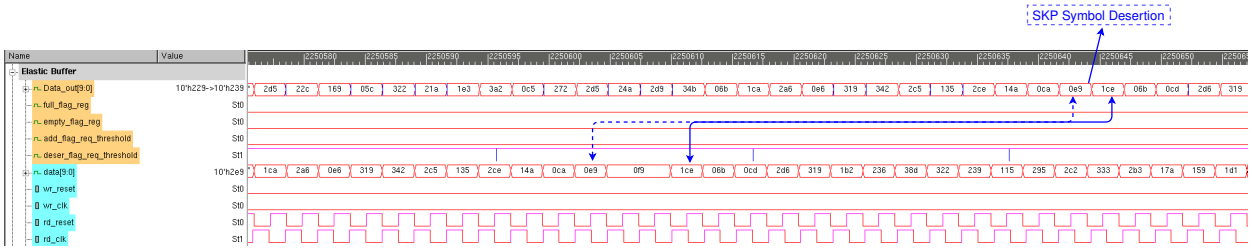
The last operating state is when the buffer is above its half depth wherein figure 4.17 (c), this situation is observed. The data to analyze are indicated with blue arrows. The difference in terms of data positions is above eight, which is not the normal state of the buffer. When the desertion request is asserted, it means that at the output SKP symbols are deserted. SKP symbols are not written into memory to attempt the reduction of the difference in data positions. With the desertion of such symbols and if both clock signals are the same, then the balance in the buffer can be reached.



(a) Response of Elastic Buffer at its half-full depth



(b) Response of Elastic Buffer when its below at its half depth



(c) Response of Elastic Buffer when its above at its half depth

Figure 4.17: Simulation results of Elastic Buffer for USB 3.0 standard

Applying the digital flow synthesis, a physical layout design will be achieved. In figure 4.18, it is depicted the layout of the Elastic Buffer for the USB 3.0 standard. Including the net routing and power rails, the layout has a dimension of 239.2 μm x 239.2 μm. If the total space used is less, then DRC violations would occur. If space is higher, then the net delay might increase because the nets are larger, increasing the parasitics that are harmful for timing reasons.

The results and comparison with the literature reviewed are observed in table 4.2. Metrics as power, operation frequency, technology-node, area, and timing are analyzed. The goal of this implementation is to operate with a frequency of 500 MHz. However, the elastic buffer for USB 3.0 standard is not achieved at this frequency and it can be viewed with the comparison table. In [21], a technology-node of 65-nm is used, allowing that delay cells to be lower in comparison with the used node in this work. An increase of the data width up to 32 bits is performed in [29], and a reduction in the operation frequency is available. For this work, the data width is 10 bits, implying an increase in terms of speed for achieving the data rate given by the standard. In terms of area, the 180-nm works are quite similar, whereas the 65-nm work is much less. The number of cells is increased for this work in comparison to

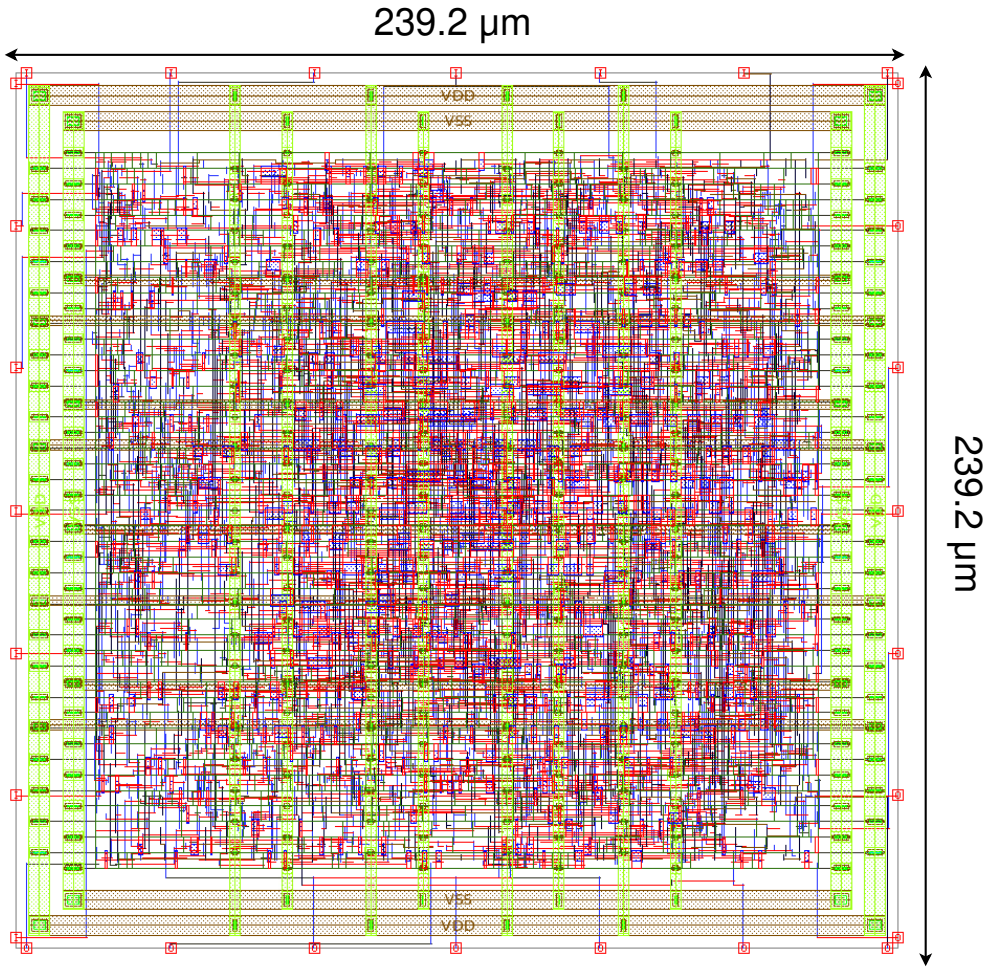


Figure 4.18: Layout of the Elastic Buffer for the USB 3.0 standard

[29], because of the effort to achieve a higher frequency operation. Related to timing metrics, the read time is faster for [21] in comparison to this work. Furthermore, the time margin for the worst path for this work is of 0.02 ns whereas in [29] is 0.06 ns. A relaxation in terms of frequency, generates a larger positive slack. It is observed that the cells delay for the technology-node used is higher since it represents 80% more with respect [21]. Limitations by the used technology are generated in terms of speed. An implementation with a larger word width can alleviate the operation frequency, and make convergence in terms of time. The power consumption varies for each implementation; the lowest is [21] and the highest is [29]. Even though the frequency operation is lower for [29], the number of nets is larger, resulting an increase in the switching activity.

The analysis of process variation, temperature, and voltage are analyzed as follows. In table 4.3 is observed the results for the typical corner, wherein scenarios with a lower voltage of 1.62 V, the implementation is not met. For the corner of 1.8 V and a temperature of 70°C, a violation is presented. For the rest of the corners, the implementation is met. Therefore, larger voltage drops or high temperatures must be avoided in order to achieve an adequate

Table 4.2: Comparison results of Elastic Buffer for USB 3.0 standard

	[21]	[29]	This Work
Technology-node (nm)	65	180	180
Power (mW)	4.164	22.102	20.5567
Frequency Operation (MHz)	500	318	380
Cells	NA	1279	1559
Area (μm^2)	11290	33027	38689
Read Time (ns)	0.218	NA	1.03
Max Critical Path (ns)	NA	0.06	0.02
Min Critical Path (ns)	NA	NA	0.09

Table 4.3: PVT typical scenario of Elastic Buffer for USB 3.0 standard

Condition	Power (mW)	Read Time (ns)	Min Path (ns)	Max Path (ns)
@ 1.62V and 25°C	16.3866	1.26	0.14	-0.55
@ 1.62V and 70°C	17.1077	1.38	0.15	-0.89
@ 1.8V and 0°C	20.0645	0.97	0.09	0.16
@ 1.8V and 25°C	20.5567	1.03	0.09	0.02
@ 1.8V and 70°C	21.4457	1.13	0.02	-0.24
@ 1.98V and 0°C	24.3344	0.76	0.05	0.66
@ 1.98V and 25°C	24.9204	0.81	0.06	0.56

performance of the system.

In table 4.4, it is observed the results for fast corners. All corners are met with variations in the temperature or voltage. However, the increase in terms of power can not be neglected because corners with larger voltage and temperature, the power is above the typical scenario. The last scenario is when the cells are slow and it is observed in table 4.5. In this scenario, neither of the corners are met where several reasons for these results can be mentioned. The first one is the use of a lower power voltage than nominal, which makes the cells slower. The second reason is the temperature because it is observed that as the temperature increases the delay does it in the same way. An alternative to improve this scenario is the frequency reduction, this can be achieved with an increase in the data width. Drawbacks for this technology-node are observed to operate at higher frequencies when the system is complex or larger in terms of the number of cells and nets.

In figure 4.19, a timing report is observed from the synthesis of the elastic buffer for the USB 3.0 standard. An analysis is performed for the worst critical path along with the subsequent critical paths, and a dependency between the memory block and the output detector block is observed. This dependency is because the memory block is divided into two blocks that correspond to the writing of the input data to the various registers (named in the figure as *xwriteregs*), and the block that reads the data from these registers and places it at the output from a given address (named in the figure as *xmux16to1*). The dependence is that the critical path must pass from the data writing up to the output reading. Hence,

```
Startpoint: xwr_ptr/xmem_enable/R_720
            (rising edge-triggered flip-flop clocked by CLK)
Endpoint: xmemory/xmux16to1/R_589
            (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

Point                                                    Incr       Path
-----------------------------------------------------------------------------
clock CLK (rise edge)                                    0.00       0.00
clock network delay (propagated)                         0.66       0.66
xwr_ptr/xmem_enable/R_720/C (DFRRQHDX1)                  0.00       0.66 r
xwr_ptr/xmem_enable/R_720/Q (DFRRQHDX1)                  0.51 +     1.17 r
xwr_ptr/xmem_enable/place2/Q (NO2HDX2)                   0.11 +     1.28 f
xmemory/xwriteregs/place53/Q (BUHDX6)                    0.18 +     1.46 f
xmemory/xwriteregs/place46/Q (BUHDX8)                    0.21 +     1.67 f
xmemory/xwriteregs/U71/Q (NA2HDX2)                       0.31 +     1.98 r
xmemory/xwriteregs/U72/Q (MU2IHDX1)                      0.37 +     2.35 f
xmemory/xwriteregs/place146/Q (INHDX1)                   0.15 +     2.49 r
xmemory/xwriteregs/place79/Q (INHDX2)                    0.08 +     2.57 f
xmemory/xmux16to1/U112/Q (AN22HDX0)                      0.40 +     2.97 r
xmemory/xmux16to1/place20/Q (NA3HDX1)                    0.15 +     3.13 f
xmemory/xmux16to1/R_589/D (DFRRQHDX1)                    0.00 +     3.13 f
data arrival time                                                   3.13

clock CLK (rise edge)                                    2.00       2.00
clock network delay (propagated)                         0.98       2.98
clock reconvergence pessimism                            0.00       2.98
clock uncertainty                                       -0.08       2.90
xmemory/xmux16to1/R_589/C (DFRRQHDX1)                               2.90 r
library setup time                                      -0.37       2.53
data required time                                                  2.53
-----------------------------------------------------------------------------
data required time                                                  2.53
data arrival time                                                  -3.13
-----------------------------------------------------------------------------
slack (VIOLATED)                                                   -0.60
```

Figure 4.19: Timing report of critical path of the Elastic Buffer for the USB 3.0 standard

between the data writing and reading block no register allows to split the combinational logic path, thus implies the use of only one clock, which is the writing clock.

The use of the read clock and the inclusion of registers between the write and read stages in the memory block is detrimental to the design; there is also a dependency between the detected data at the output detector block and the read address generator. They must be

Table 4.4: PVT fast scenario of Elastic Buffer for USB 3.0 standard

Condition	Power (mW)	Read Time (ns)	Min Path (ns)	Max Path (ns)
@ 1.8V and -40°C	19.2797	0.88	0.08	0.38
@ 1.8V and 0°C	20.0645	0.97	0.09	0.17
@ 1.8V and 25°C	20.5567	1.03	0.1	0.03
@ 1.98V and -40°C	23.4217	0.7	0.04	0.83
@ 1.98V and 0°C	24.334	0.76	0.06	0.68
@ 1.98V and 100°C	26.7008	0.93	0.08	0.25
@ 1.98V and 125°C	27.2959	0.98	0.09	0.13

Table 4.5: PVT slow scenario of Elastic Buffer for USB 3.0 standard

Condition	Power (mW)	Read Time (ns)	Min Path (ns)	Max Path (ns)
@ 1.62V and -40°C	15.3628	1.22	-0.85	-0.93
@ 1.62V and 70°C	17.1077	1.34	-0.86	-1.96
@ 1.62V and 125°C	18.0053	1.49	-0.86	-2.4
@ 1.8V and 70°C	21.4457	1.11	-0.92	-1.36
@ 1.8V and 85°C	21.7444	1.12	-0.92	-1.45
@ 1.8V and 125°C	22.5446	1.22	-0.93	-1.7

in phase to determine if the read data is a special symbol and if an additional symbol must be added by the unbalance that occurs in the memory occupation in the next cycle. When registering data coming from memory, it implies that an offset is generated. It means that when the special symbol is detected at the output, the insertion functionality of the special symbols cannot be executed. Therefore, an undesired performance of the elastic buffer is presented.

Unlike the 2.0 version, this dependency does not exist. This means that the data is written and read without any dependency on other modules, allowing the writing and reading stages of the memory block to be registered. Thus providing a reduction in the combinational logic path.

The simulation results of the EB for the USB 2.0 standard are depicted in figure 4.20. The outputs are highlight in orange color and the inputs with blue color. The operation principle of this buffer is to be a repeater for the incoming data but sampled at the output with the local clock domain instead of the recovery clock domain. It is observed that incoming data is not at the output immediately, and this is because by the minimum difference between the read and write addresses that must exist in order to get timing margins. This difference acts as the inclusion and desertion of the special symbols of the USB 3.0 standard. Clock domains deviations are also applied in this buffer for its validation. Similarly, the clocks are deviated more cycles than the standard indicates.

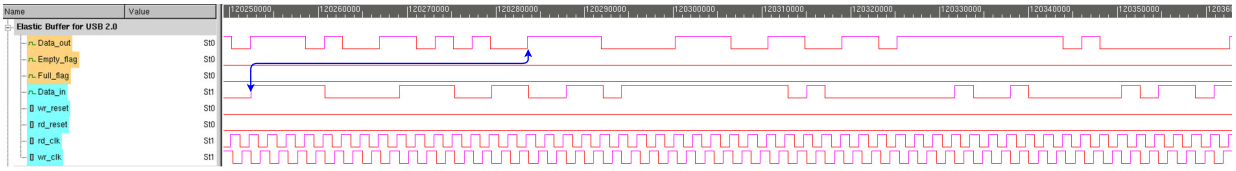


Figure 4.20: Simulation results of Elastic Buffer for USB 2.0 standard

In the same manner, as of the EB for USB 3.0 standard, the digital flow synthesis is applied, that is described in the previous section. The physical view of the EB for USB 2.0 standard is observed in figure 4.21. Including the net routing and power rails, the layout has a dimension of 173.8 μm x 173.8 μm.

The results of the elastic buffer for USB 2.0 standard are observed in table 4.6. It is included the read time, area, and critical timing paths. Since the data in the USB 2.0

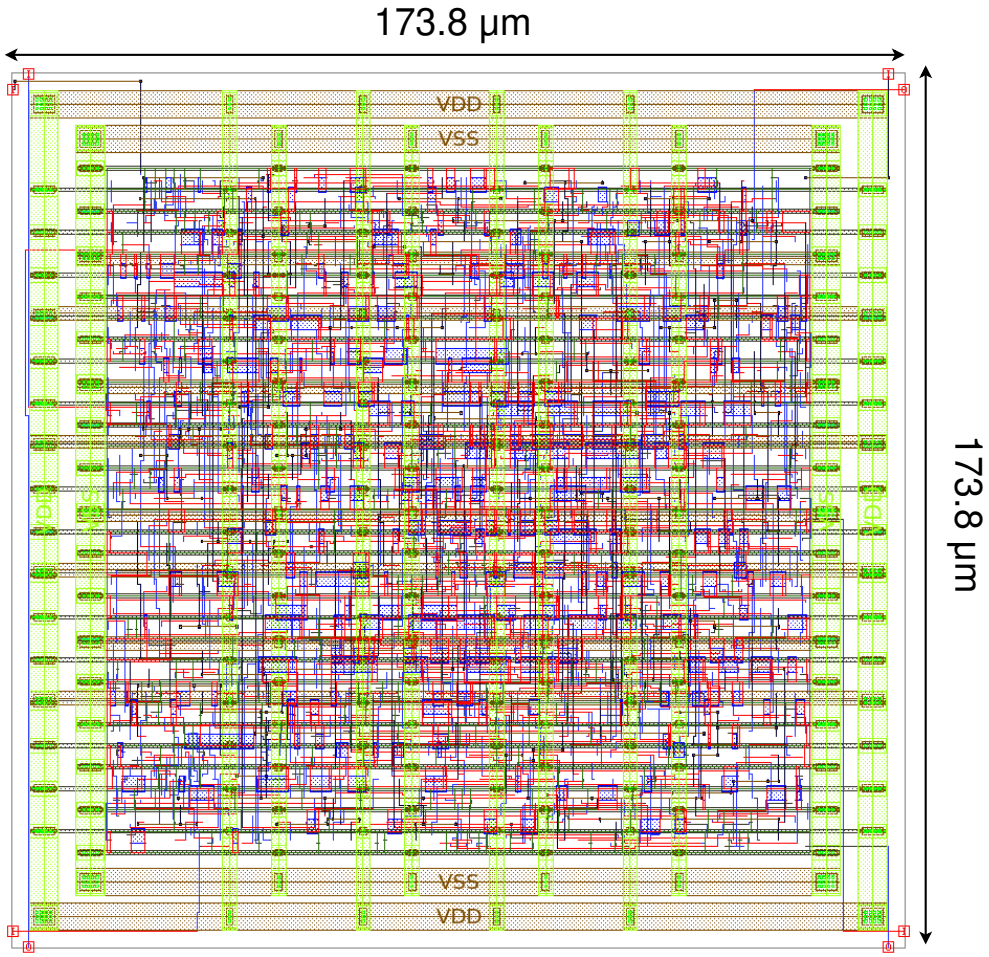


Figure 4.21: Layout of the Elastic Buffer for the USB 2.0 standard

transceiver is transmitted serially, the amount of area and cells are less in comparison with the USB 3.0 link. Despite the buffer depth for the USB 2.0 link is larger than the USB 3.0 link, the combinational logic is lower because fewer blocks are implemented by the simplicity of the design. The detection of special symbols at the input and output, as well as to compute the difference between addresses in both unit generators, are not presented in the EB for the USB 2.0 version. This allows an increase in terms of timing margin paths despite the operation frequency is higher. The number of cells implemented on this buffer and power consumption are less. The read time is quite similar with respect to the USB 3.0 implementation, because the read operation that is used in both buffers is the same. However, the use of the read clock in the memory block for the EB produces deviations in the logic that is used for the reading operation.

The analysis of process variation, temperature, and voltage are analyzed as follows. In table 4.7 is observed the results for the typical corner, wherein scenarios with a lower voltage as 1.62 V, the implementation is not met. For the corner of 1.8 V up to a temperature of 70°C, no violations are presented. For the remaining corners, the implementation is met.

Table 4.6: Results of Elastic Buffer for USB 2.0 standard

Technology-node (nm)	180
Power (mW)	16.6719
Frequency Operation (MHz)	480
Cells	660
Area (μm^2)	17594
Read Time (ns)	1.26
Max Critical Path (ns)	0.22
Min Critical Path (ns)	0.14

Table 4.7: PVT typical scenario of Elastic Buffer for USB 2.0 standard

Condition	**Power (mW)**	**Read Time (ns)**	**Min Path (ns)**	**Max Path (ns)**
@ 1.62V and 25°C	13.2602	1.54	0.17	-0.15
@ 1.62V and 70°C	13.8006	1.71	0.17	-0.34
@ 1.8V and 0°C	16.3038	1.18	0.14	0.31
@ 1.8V and 25°C	16.6719	1.26	0.14	0.22
@ 1.8V and 70°C	17.3437	1.38	0.14	0.07
@ 1.98V and 0°C	19.8489	0.92	0.10	0.65
@ 1.98V and 25°C	20.2970	0.97	0.10	0.58

Results for fast corners are observed in table 4.8. All corners are met even though with variations in the temperature or voltage. However, the increase in terms of power can not be neglected by the the fact that corners with larger voltage and temperature, the power is above the typical scenario.

Table 4.8: PVT fast scenario of Elastic Buffer for USB 2.0 standard

Condition	**Power (mW)**	**Read Time (ns)**	**Min Path (ns)**	**Max Path (ns)**
@ 1.8V and -40°C	15.7129	0.87	0.14	0.44
@ 1.8V and 0°C	16.3038	1.18	0.15	0.31
@ 1.8V and 25°C	16.6719	1.26	0.15	0.23
@ 1.98V and -40°C	19.1294	0.68	0.10	0.75
@ 1.98V and 0°C	19.8489	0.92	0.11	0.65
@ 1.98V and 100°C	21.6346	1.15	0.12	0.39
@ 1.98V and 125°C	22.0819	1.21	0.12	0.32

The last scenario is when the cells are slow. In this scenario, some of the corners are met and it is depicted in table 4.9. The corners that met with the implementation are the scenario at a temperature of 70°C and a nominal power supply of 1.8 V, and the scenario at -40°C; for the remaining corners, several reasons for these results can be mentioned. The first one is the use of a lower power voltage than nominal, which makes the cells slower. The second reason is the temperature because it is observed that as the temperature increases the delay does it in the same way. Frequency reduction cannot be applied because the data is received serially. A change in the node-technology that is used, could reduce the cells

delay, and improve the critical timing paths.

Table 4.9: PVT slow scenario of Elastic Buffer for USB 2.0 standard

Condition	Power (mW)	Read Time (ns)	Min Path (ns)	Max Path (ns)
@ 1.62V and -40°C	12.4878	1.31	0.11	0.08
@ 1.62V and 70°C	13.8006	1.71	0.11	-0.39
@ 1.62V and 125°C	14.4714	1.93	0.10	-0.67
@ 1.8V and 70°C	17.3437	1.38	0.08	0.02
@ 1.8V and 85°C	17.5691	1.44	0.08	-0.03
@ 1.8V and 125°C	18.1719	1.55	0.07	-0.19

A solvent and convincing design when corner analyses are performed, are necessary to assure the performance of the integrated circuit once it is fabricated. As it was observed for the implementation of the USB 3.0 version, it fails to converge even in the typical case involving a transistor scenario at 25°C and operating frequency of 500 MHz. On the other hand, the design of the USB 2.0 version is a more simplified one in terms of logic. Therefore, convergence is achieved for all corners.

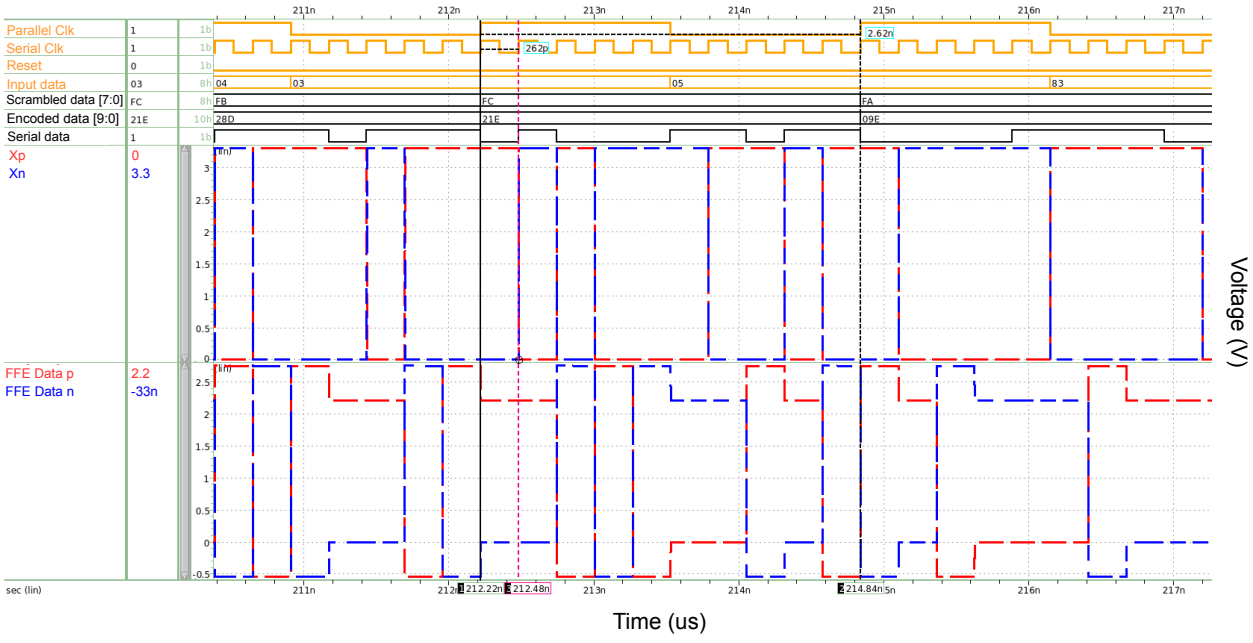
For the consolidation of this convergence, the techniques mentioned in the back-end section and additional ones are used for this design. The additional techniques included the use of a more pessimistic library than the typical scenario, causing the tool to make a greater effort when logic cells are synthesized to achieve convergence in terms of timing. Higher granularity clock tree synthesis is performed that permits a clock cell having to drive fewer clock nets that feed the several registers on its clock pins. Inserting buffers into the clock net for a given path is used, and this allows delays to be decreased because a particular register drives a smaller fan-in. Changing sizes is another technique used in combinational logic paths between registers, such variations allow the delays of a given path to be decreased and thus allow the slack to be positive. All these changes come with the cost of an increase in terms of power and area.

5 Transceiver Link Simulation with Design of Elastic Buffers

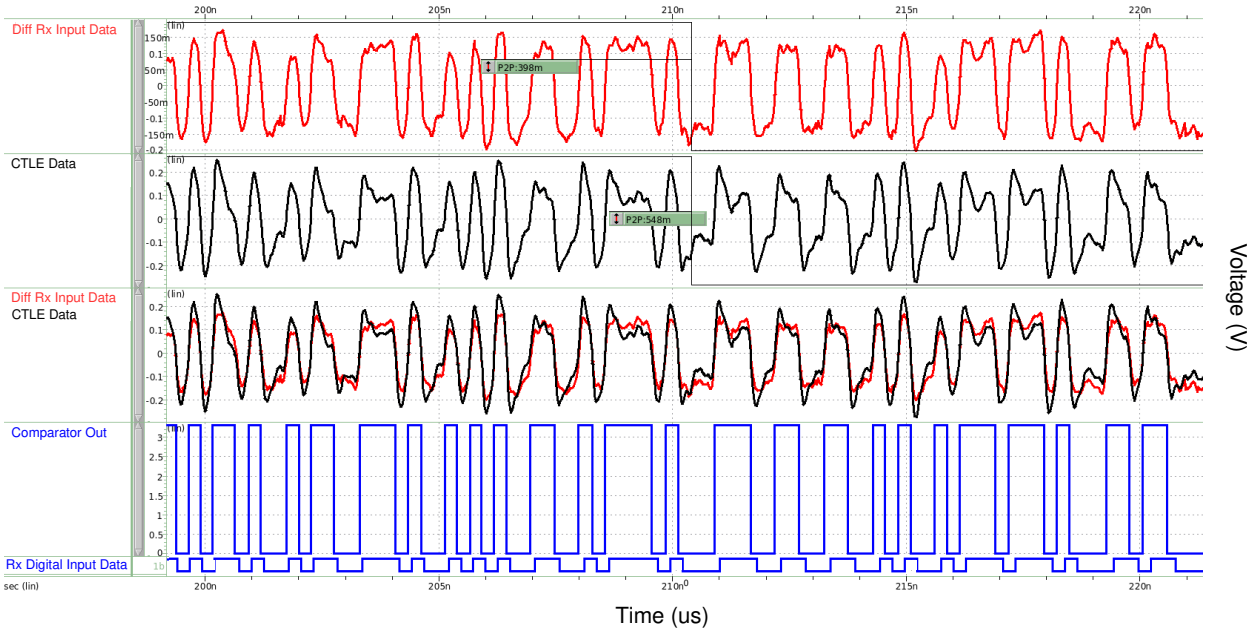
The connection of the three main modules, which are the transmitter, interconnect and receiver is performed in this chapter for both USB transceivers. The framework for USB simulation that was detailed in chapter 3 is used. In addition, eye-diagrams are included for the analysis of the interconnect, as well as for the data enhancement demonstration with the use of equalization for the USB 3.0 link. This is a functional validation chapter that includes the high-level description of the transceivers on Verilog-AMS along the design of EBs for corroborating the transmission and reception of data. These data go through the transmission lines that were discussed as the interconnect in chapter 3. It should be recalled that all the blocks used for these validations are modeled at high-level description, except for the interconnect and the EBs, in which netlists are extracted.

5.1. USB 3.0 Transceivers

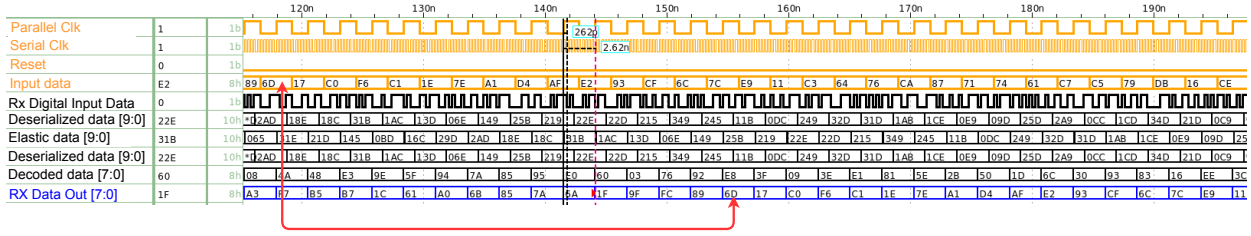
The connection of the whole modules is performed in this section. The description of each module was described previously in *Chapter 3*. A loopback connection of the transmitter, interconnect and receiver for the USB 3.0 transceiver is dealt with in this section. Such connections allow the validation of the transceiver with the inclusion of the elastic buffer that was presented in *Chapter 4*. The signal labels used in this validation are identified according to the USB 3.0 simulation framework in *Chapter 3*. The conversion from the



(a) Transmitter stage of USB 3.0 transceiver



(b) Interconnect stage of USB 3.0 transceiver



(c) Receiver stage of USB 3.0 transceiver

Figure 5.1: USB transceiver validation framework for the 3.0 standard

digital to analog signals is carried out at the output of the serializer in the transmitter stage. The analog to digital conversion is performed at the output of the comparator that has the CTLE equalized data in the receiver stage.

The transmission stage for this link is observed in figure 5.1 (a), which digital and analog signals are included. The orange digital inputs are injected into the link, which are the serial clock, parallel clock, reset, and the data. It must be recalled that the difference between the serial and parallel clock is 10 times. For the data generation, several PRBSs with different seeds are used for introducing randomness to the validation testbench. Internal black signals are observed such as the scrambled, encoded, and serialized data. The serialized data is the last digital signal and then a transformation into analog domain of this signal is performed. The positive and negative signals transformation of the serialized and equalized data are observed in figure 5.1 (a) in the dashed lines.

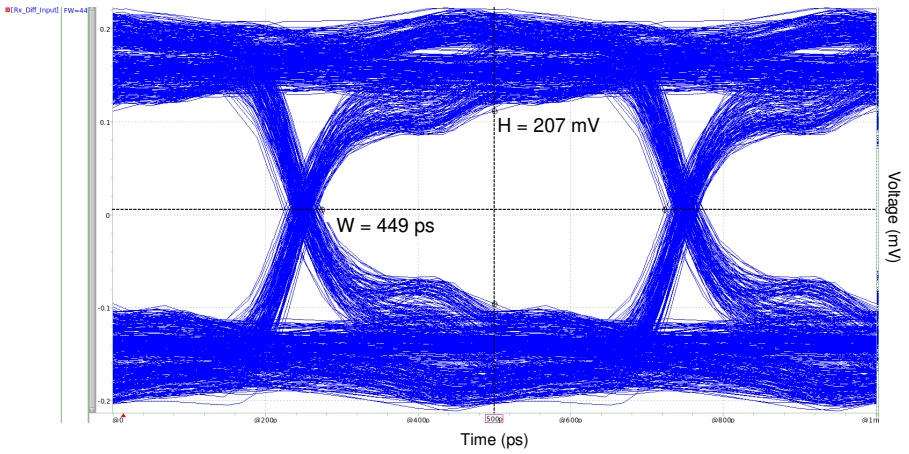
The interconnect effect of the USB link 3.0 is observed in figure 5.1 (b). In this figure,

the reception of the signal is highlighted in red color that is established after the receiver LVDS. The black signal is the equalization with the CTLE technique and a superposition of the last signals is observed in the third row. An enhancement is presented with the equalizer and it is observed an increase in the amplitude of the equalized data in comparison to the received data. A comparator is used to determine if the differential data represents a high or low signal value, which represents the serial value that is transmitted. The transformation of the analog signal into digital is carried-out for feeding the further digital blocks of the receiver stage; these signals are represented in blue.

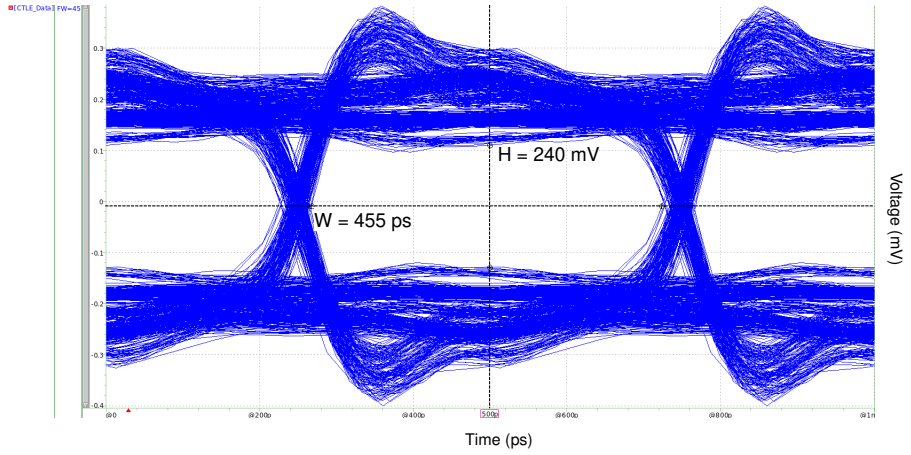
In figure 5.1 (c), it is injected the converted signal into the receiver. The orange signals are the inputs and these are serial clock, parallel clock, reset, and the transmitted data. Internal signals are represented in black colors. It is observed several signals such as the received serial, deserialized, elastic, and decoded data. The blue signal is the descrambled data and this one represents the original data that is sent from the transmitter. Two data are indicated with the red line, and these are the original data and the received data. It is observed that the transmitted data is received at the receiver stage with the same value, and beyond this indicator, the pattern is the same for the transmitter and receiver.

Eye-diagrams are used for analyzing signal integrity, which must have a considerable opening in its height and width. If the previous characteristics are met, there will be no loss or corruption of data that can occur by high-frequency effects. For the evaluation of the interconnect of the USB transceiver, eye-diagrams are used at the input of the receiver stage. The variation of the data rate as well as variations in the cable length that was used for the characterization of the interconnect in terms of S-parameters is included for the evaluation. The maximum data rate that is used, is the one obtained by the elastic buffer that is 3.81 Gbps.

The first analysis is with a data rate of 1 Gbps and a cable length of 1 meter. This description involves two analyses and these are the differential data with and without equalization



(a) Differential data without equalization

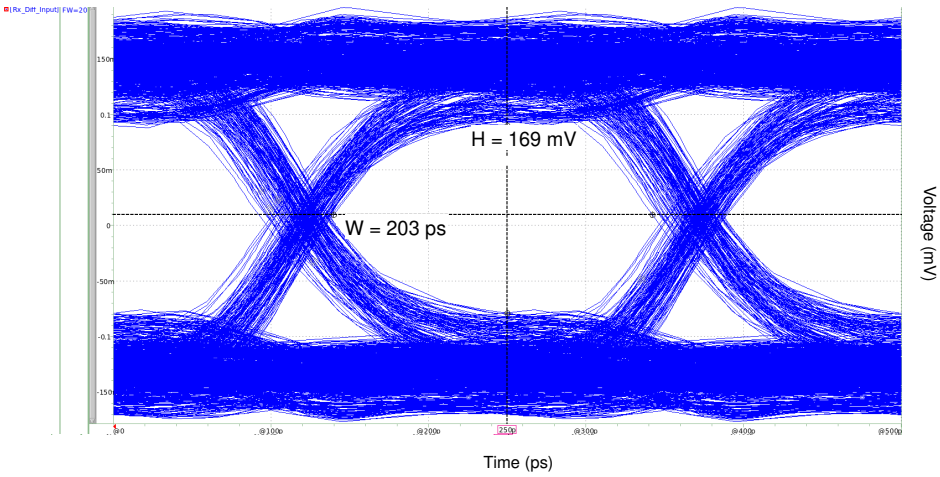


(b) Differential data with CTLE equalization

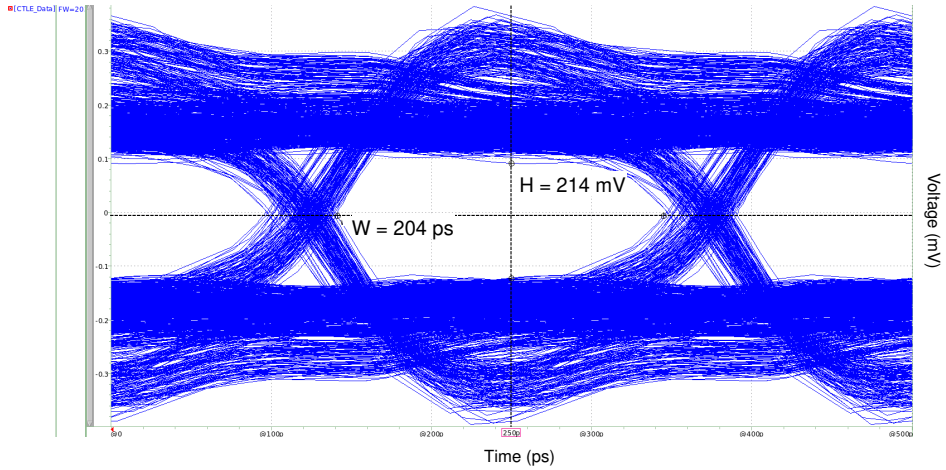
Figure 5.2: Eye-Diagrams comparison at the input receiver stage at 1 Gbps and 1 meter of length

at the receiver with the CTLE enhancement. The differential data without equalization is observed in figure 5.2 (a) with a *Unit Interval* (UI) of 500 ps. It is observed a clear opening of the eye in both terms of interest, which are the width that is measured in *ps* and the height that is measured in *mV*. The equalized data is observed in figure 5.2 (b). An improvement of 30 mV and 5 ps is achieved. The equalization exhibits less inference of the noise and jitter by the ISI reduction.

Increasing the data rate and holding the cable length are presented in the next analysis. In figure 5.3 (a), it is presented the differential data without equalization at a data rate of 2 Gbps with a UI of 250 ps, whilst in figure 5.3 (b) is presented the equalized data at the same data rate. A similar response is observed, in which a clear opening can be observed. The equalization exhibits an increase in the height around 45 mV and the jitter is reduced 1 ps.



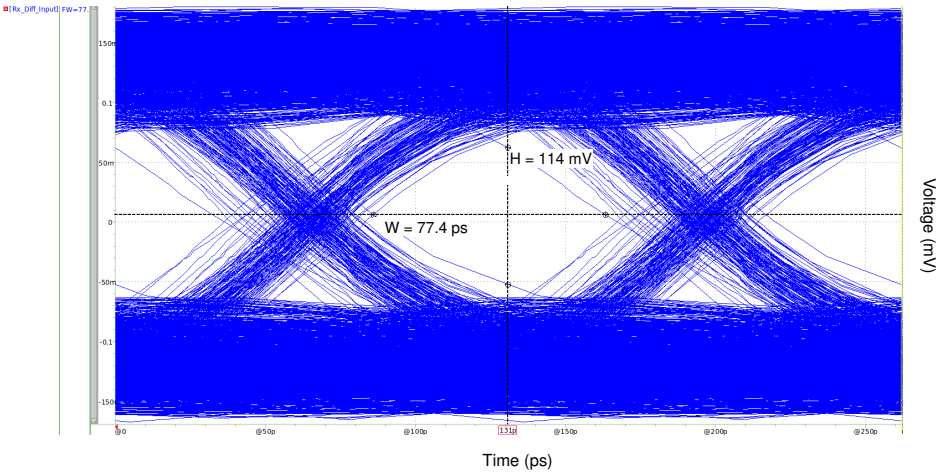
(a) Differential data without equalization



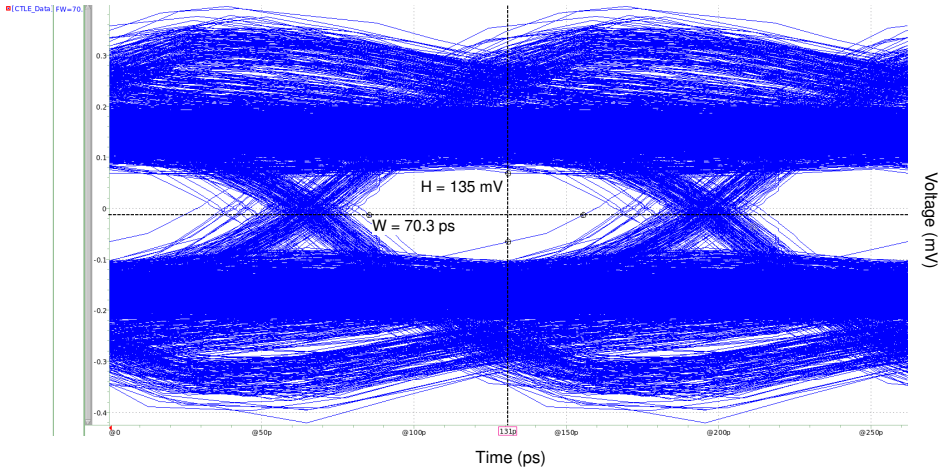
(b) Differential data with CTLE equalization

Figure 5.3: Eye-Diagrams comparison at the input receiver stage at 2 Gbps and 1 meter of length

The last analysis with a cable length of 1 meter is increasing up to 3.81 Gbps the data rate with a UI of 131 ps. The results without and with equalization are observed in figure 5.4 (a) and 5.4 (b), respectively. As the data rate increases, the same is seen with the ISI. This implies that the openings are not clear and enhancement with equalization is required. Although in the equalized data there is a reduction of 7 ps, meaning that the jitter increases, an improvement in noise reduction is achieved because the unwanted effect that disturbs the amplitude of the signal, is improved around 20 mV.



(a) Differential data without equalization

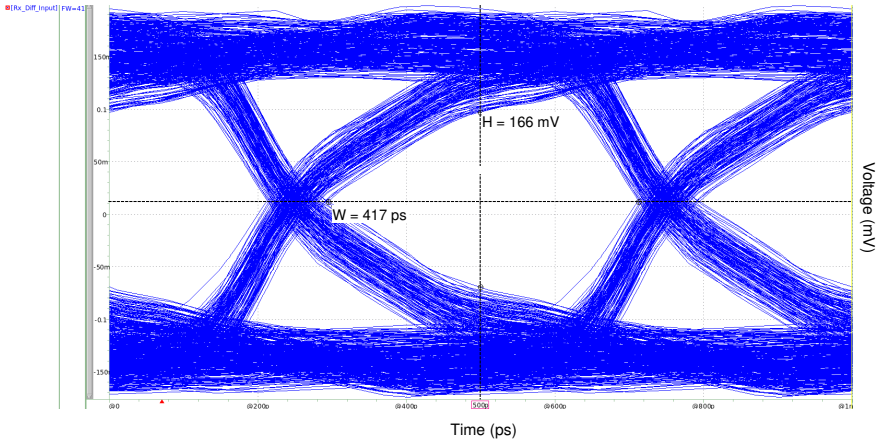


(b) Differential data with CTLE equalization

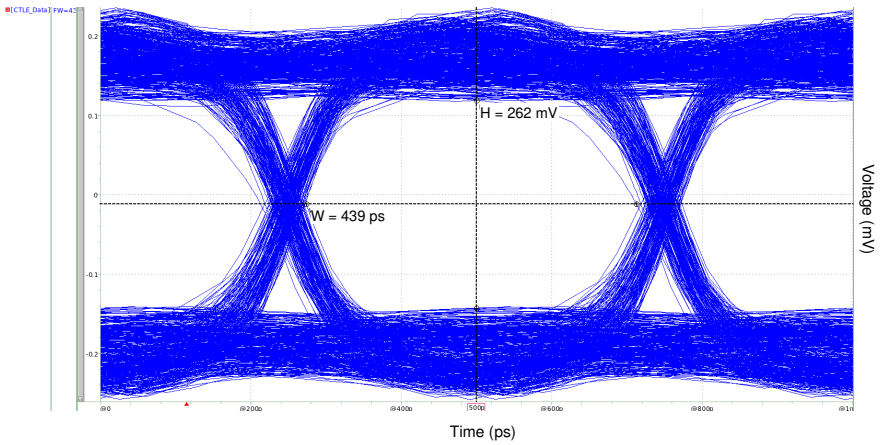
Figure 5.4: Eye-Diagrams comparison at the input receiver stage at 3.81 Gbps and 1 meter of length

The next analysis includes the same data rates variations with its respective UIs, and an increase in the cable length up to 2 meters. The first result with the new cable length is with a data rate of 1 Gbps. The data without and with equalization are observed in figures 5.5 (a) and 5.5 (b), respectively. The increase in cable length introduces higher losses. However, an enhancement is performed with the use of equalization. The height is increase around 100 mV and the width is increased by 20 ps, resulting in a wider eye-opening.

Increasing the data rate and holding the cable length is presented in the next analysis. In figure 5.6 (a), it is presented the differential data without equalization at a data rate of 2 Gbps, whilst in figure 5.6 (b) is presented the equalized data at the same data rate. A similar response is observed in the data without equalization, in which harmful effects are increased. The eye-opening is less in comparison with a cable length of 1 meter. The equalization increases the height around 65 mV and the width in 25 ps. The effect of the

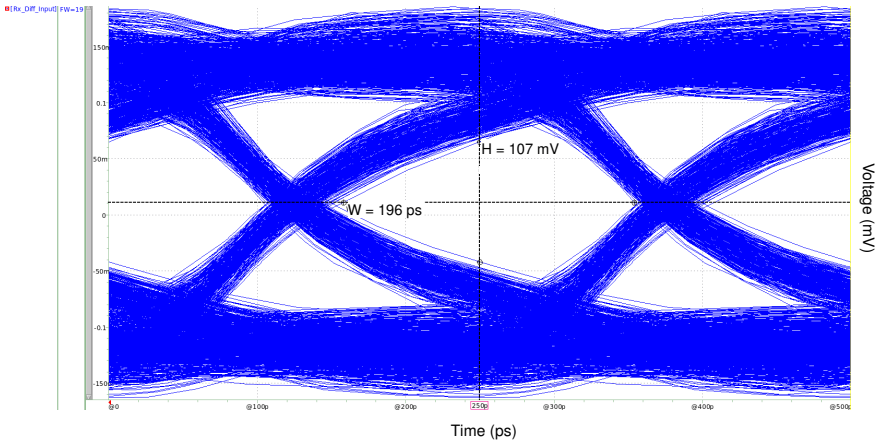


(a) Differential data without equalization

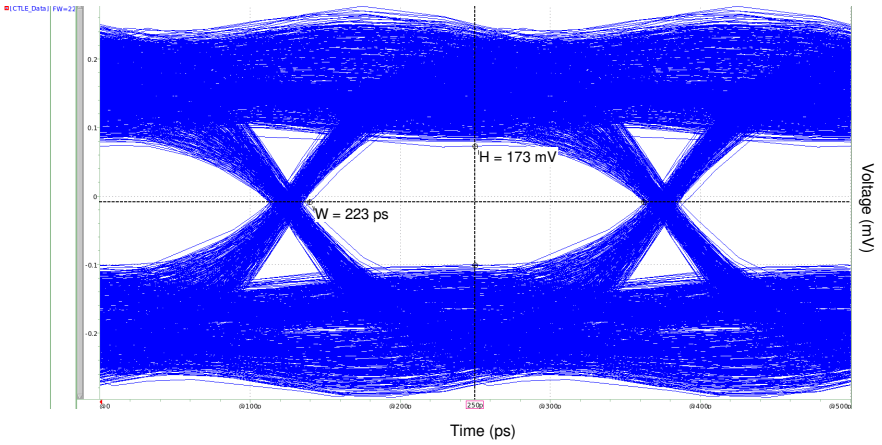


(b) Differential data with CTLE equalization

Figure 5.5: Eye-Diagrams comparison at the input receiver stage at 1 Gbps and 2 meters of length

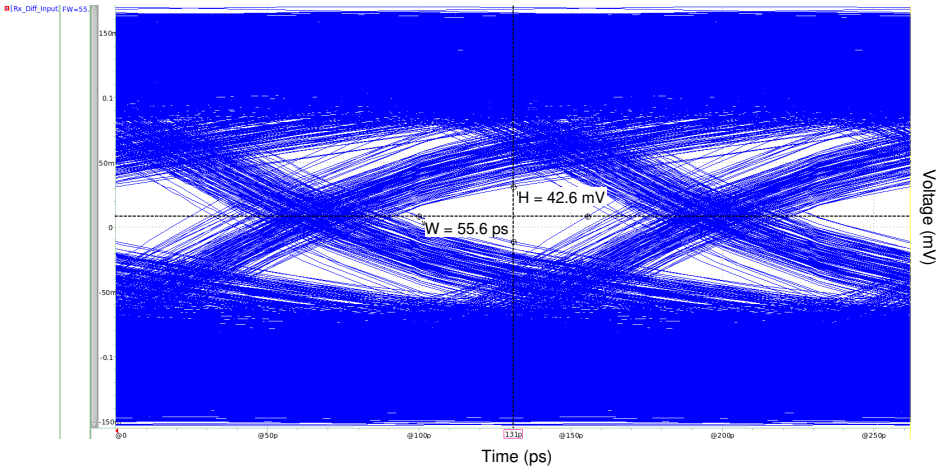


(a) Differential data without equalization

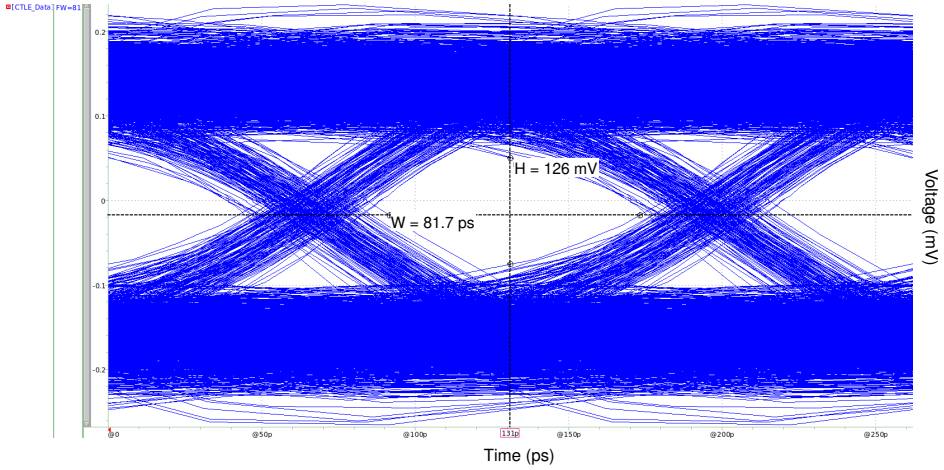


(b) Differential data with CTLE equalization

Figure 5.6: Eye-Diagrams comparison at the input receiver stage at 2 Gbps and 2 meters of length



(a) Differential data without equalization



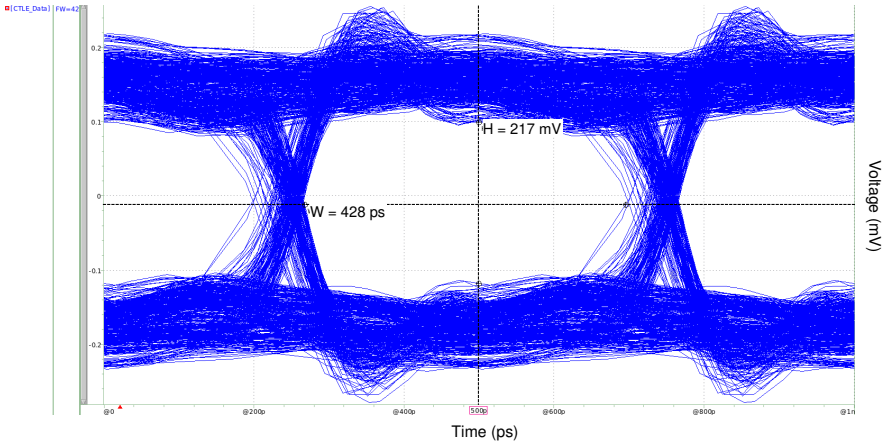
(b) Differential data with CTLE equalization

Figure 5.7: Eye-Diagrams comparison at the input receiver stage at 3.81 Gbps and 2 meters of length

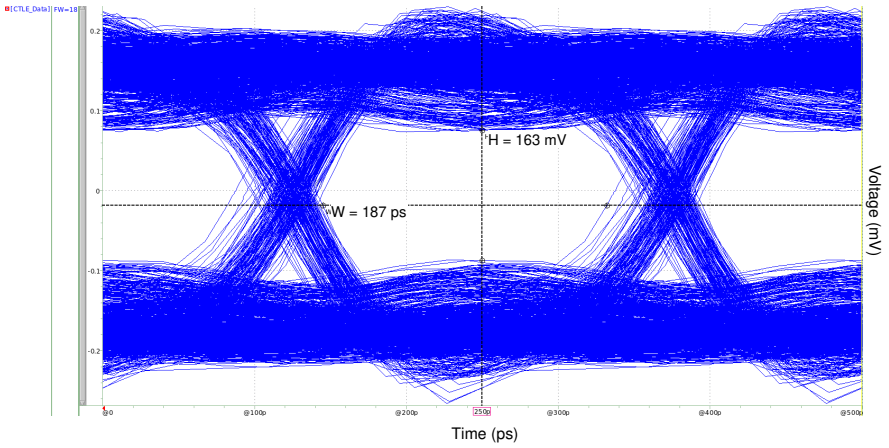
equalizer is more visible in interconnects with higher losses since the improvements are more significant.

The last analysis with a cable length of 2 meters is increasing up to 3.81 Gbps the data rate. The results without and with equalization are observed in figure 5.7 (a) and 5.7 (b), respectively. As the data rate increases, the same is presented with the ISI. This implies that the openings are not clear and enhancement with equalization is required. An increase of 80 mV and 30 ps in terms of height and width is presented with the use of equalization. It is visible the effect of equalization since harmful effects are reduced.

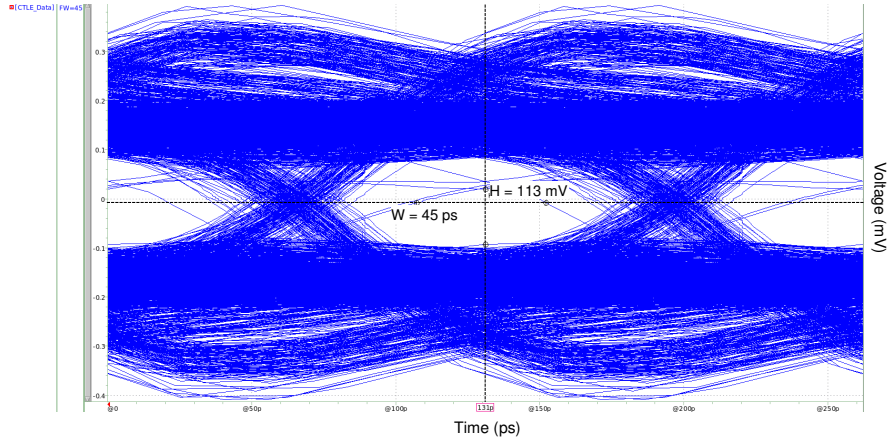
Finally, it should be taken into consideration that not having control over jitter and skew throughout the system would cause the eye-diagrams to be affected. Such effects imply that the eye-aperture is reduced in terms of height and width. In figures 5.8 (a), 5.8 (b), and 5.8



(a) Equalized differential data at 1 Gbps



(b) Equalized differential data at 2 Gbps



(c) Equalized differential data at 3.81 Gbps

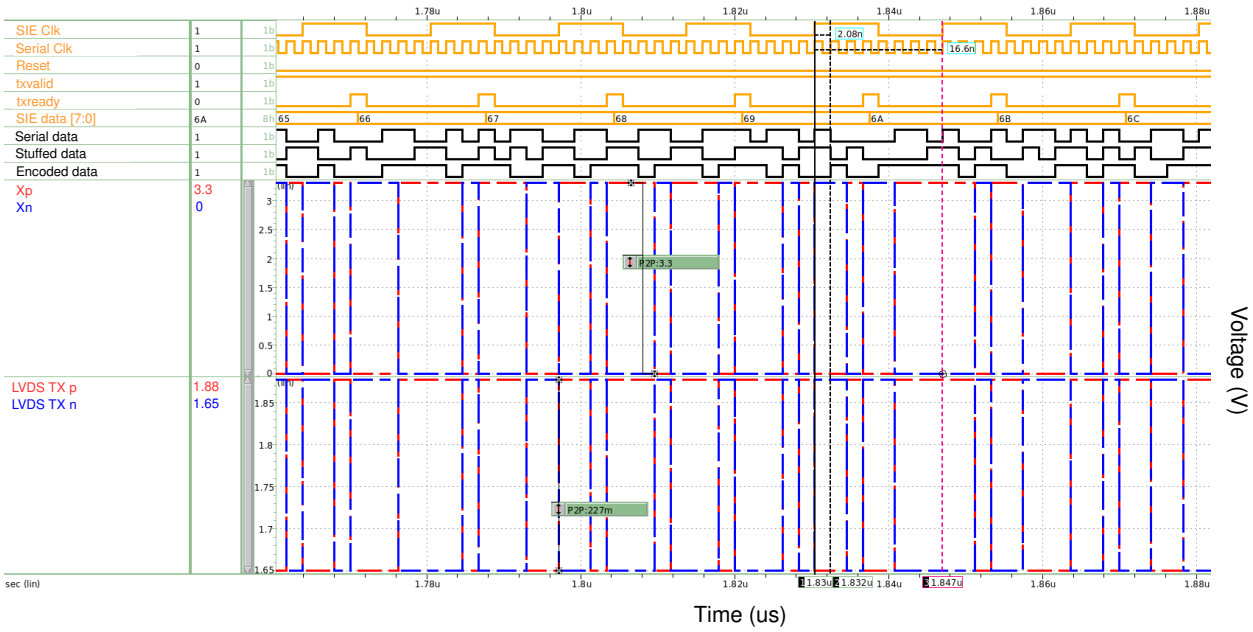
Figure 5.8: Eye-Diagrams comparison at the input receiver stage at different data rates and 1 meter of length with CTLE enhancement

(c) are observed the different data rates, which are 1, 2 and 3.81 Gbps, respectively with an increase in the aforementioned harmful effects. The cable length that is used is 1 meter and the equalized data are shown in the figure. It is observed that as the data rate increases, the eye-openings are reduced despite the use of an equalizer at the receiver stage.

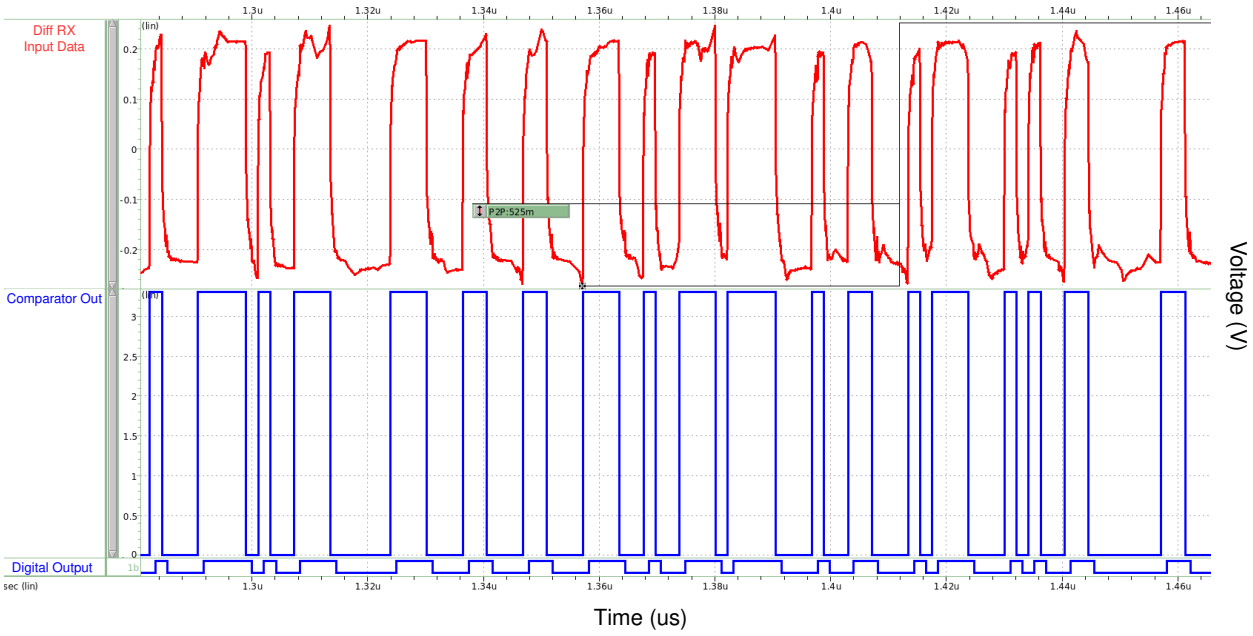
5.2. USB 2.0 Transceivers

A loopback connection of the transmitter, interconnect and receiver for the USB 2.0 transceiver is dealt with in this section. The validation of the link is addressed along the inclusion of the elastic buffer that was presented in *Chapter 4*. The signal labels used in this validation are identified according to the USB 2.0 simulation framework in *Chapter 3*. The conversion from the digital to analog signals is carried out at the output of the encoder in the transmitter stage. The analog to digital conversion is performed at the output of the comparator that has the received data by the LVDS in the receiver stage.

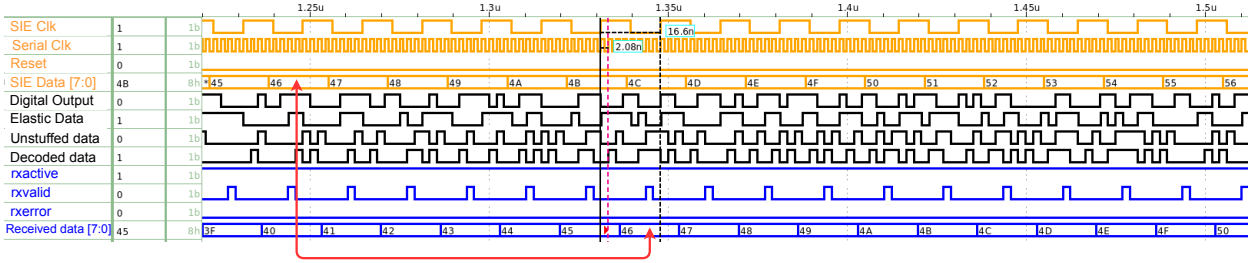
The transmission stage for this link is observed in figure 5.9 (a), where digital and analog signals are observed. The orange digital inputs are injected into the link. Such inputs are the serial clock, SIE clock, reset, txvalid, and the data. The txready signal also represented in orange, is an output of the transmitter digital shared path; however, this signal is used for the communication between this path and the SIE of the transmitter. It must be recalled that the difference between the serial and SIE clock is 8 times. Internal signals are observed in black such as the shifted, stuffed, and encoded data. The serialized data goes into a transformation from digital to the analog domain. Therefore, the positive and negative



(a) Transmitter stage of USB 2.0 transceiver



(b) Interconnect stage of USB 2.0 transceiver



(c) Receiver stage of USB 2.0 transceiver

Figure 5.9: USB transceiver validation framework for the 2.0 standard

signals the serialized data and the LVDS signals are observed in figure 5.9 (a) in the dashed lines.

The interconnect effect of the USB link 2.0 is observed in figure 5.9 (b). In this figure is included the reception of the signal in red color that is established after the receiver LVDS. The simplicity of this link in comparison with the 3.0 version, alleviates the data rate that it is 480 Mbps. A comparator is used to determine if the differential data represents a high or low signal value, which represents the serial value that is transmitted. Blue signals represents the transformation of the analog to digital domain that feeds the blocks at the receiver.

In figure 5.9 (c), it is observed the receiver stage. The orange signals are the inputs and these are serial clock, SIE clock, reset, and the transmitted data (SIE data). Internal signals are represented in black, which the received serial (named in the figure as *Digital output*),

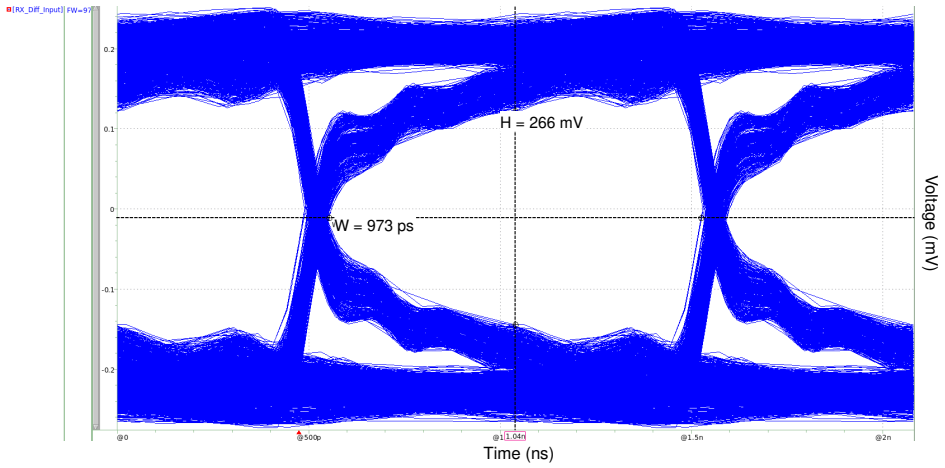
elastic, unstuffed, and decoded data are observed. There are several signals in blue such as rxactive, rxvalid, and rxerror that are used by the SIE of the receiver. In addition, the received data is observed in blue and represents the original data sent from the transmitter. Two data are indicated with the red line, and these are the original data and the received data. It is observed that both data are the same in their respective stages, thus the data compliance.

For the evaluation of the interconnect of the USB link, eye-diagrams are used at the input of the receiver stage. Since the interconnect that is used in this link is the same as the 3.0 version, variation of the data rate is not shown since the maximum rate is 480 Mbps with a UI of 1.04 ns. Analyses beyond this rate were mentioned in the USB 3.0 link validation. However, cable length variations are presented for these analyses.

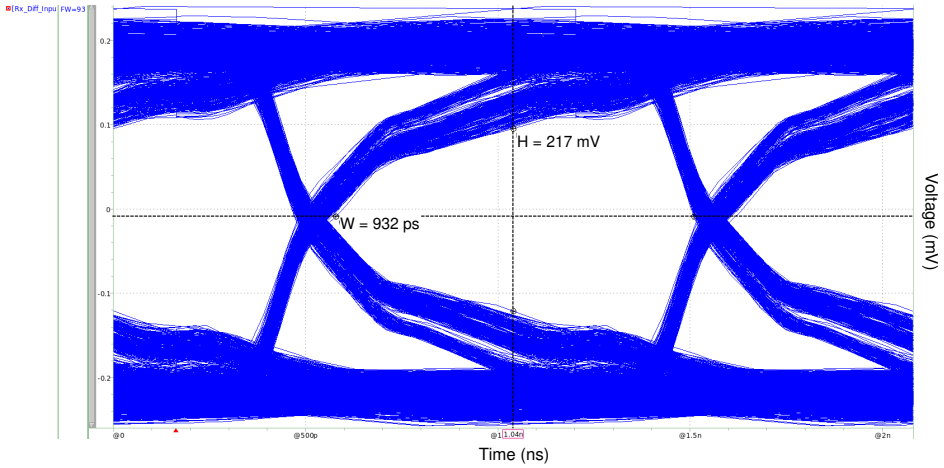
The first analysis is with a cable length of 1 meter. The simplicity of the link in terms of the number of blocks that compose it and the data rate enables the no utilization of equalizers. The differential data is observed in figure 5.10 (a). It is observed a clear opening of the eye in both terms of interest, which are the width that is measured in ps and the height that is measured in mV. Effects such as jitter and noise might be negligible at these rates. It is achieved a height and a width of 266 mV and 973 ps, respectively.

The next analysis includes the same data rate and an increase in the cable length up to 2 meters. In figure 5.10 (b), it is observed the eye-diagram with this new cable length. The increase in cable length causes higher losses, and the jitter is increased by 30 ps. In addition, the width is decreased around 50 mV, which is harmful and this implies that the eye-opening is being reduced.

Finally, it should be taken into consideration that not having control over jitter and skew throughout the system would cause the eye-diagrams to be degraded. In figure 5.11, it is



(a) Differential data with a length of 1 meter



(b) Differential data with a length of 2 meter

Figure 5.10: Eye-Diagrams comparison at the input receiver stage at 480 Mbps with several lengths

depicted an increase in the aforementioned effects applied into this link. The cable length that is used is 1 meter. The height and width of this eye-diagram are less in comparison to the eye-diagram of figure 5.10 (a), Thus, the correct timing of the signals is harmful and the data loss is presented by these unwanted effects.

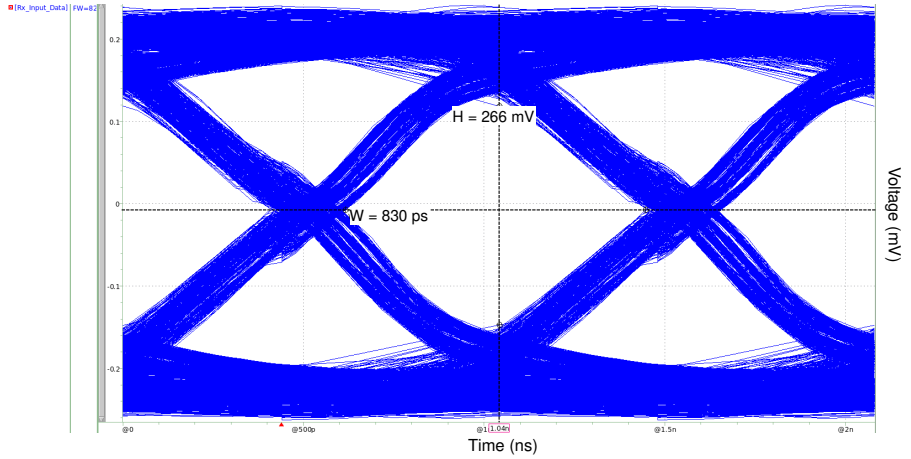


Figure 5.11: Eye-Diagram comparison at the input receiver stage 480 Mbps and 1 meter of length with error insertion

6 Conclusions and Recommendations

Elastic buffer designs considering the USB 2.0 and 3.0 standards were proposed and validated in this work. Half-full approach was implemented in both EBs. For the USB 3.0 version, the word size is of 10 bits, and 1 bit for the USB 2.0 case. The estimated size for 3.0 and 2.0 versions are of 239.2 x 239. μm and 173.8 x 173.8 μm, respectively. It was necessary to apply clock tree synthesis suitable for high-speed designs, in order to decrease the fanout in the clock tree branches, an enabling timing convergence on the critical paths.

A high-level simulation framework for USB transceivers was developed, where each block of the transmitter, receiver and interconnect channel can be incorporated and evaluated. This framework was used to validate the designs using an AMS simulation approach, where the elastic buffers were incorporated with other blocks available, some of them described at behavioral level with Verilog-AMS.

Speeds of 380 MHz (3.8 Gbps) and 480 MHz (480 Mbps) for the proposed elastic buffers were achieved for USB 3.0 and 2.0 standards, respectively. The USB 3.0 implementation is slower than 2.0 version because of the number of logic levels, that cannot be neglected or reduced, due to the presence of special symbols (SKP). The USB 2.0 implementation converges in all corners at the target speed, which was possible due to a smaller number of logic levels.

Serial blocks are the bottleneck for pre-silicon level implementations because they must operate at a higher frequency to meet the data rate. Special attention should be at the receiver clock and data recovery stage, since the timing margins become narrow at higher speeds and the used CMOS process performance might be insufficient.

Improvements in timing paths for the USB 3.0 design could be performed, as future work, with the use of a lower CMOS process node; nevertheless, the migration to another process must be accompanied by optimizations in the digital design flow synthesis. In addition, an increase in the word size used by the elastic buffer for the 3.0 version is an alternative that might facilitate the design convergence at 500 MHz operation, required to reach the data rate of 5 Gbps.

References

[1] Y. C. Chu, L. R. Chang-Chien, N. S. Artan, D. Czarkowski, C. H. Chang, J. Zou, and H. J. Chao, "On-Chip AC-DC Multiple-Power-Supplies Module for Transcutaneously Powered Wearable Medical Devices," *IEEE Transactions on Industry Applications*, vol. 54, no. 2, pp. 1724–1736, 2018.

[2] M. Augustýnek, D. Laryš, J. Kubíček, P. Marešová, and K. Kuča, "Use Effectiveness of Medical Devices: A Case Study on the Deployment of Ultrasonographic Devices," *Therapeutic Innovation and Regulatory Science*, vol. 52, no. 4, pp. 499–506, 2018.

[3] J. M. Lee, J. H. Lee, and T. M. Chung, "Experimental QoS test of medical data over wired and wireless networks," *International Conference on Advanced Communication Technology, ICACT*, vol. 1, pp. 142–146, 2008.

[4] R. García-Ramírez, A. Chacon-Rodriguez, R. Molina Robles, R. Castro-Gonzalez, E. Solera-Bolanos, G. Madrigal-Boza, M. Oviedo, D. Salazar-Sibaja, D. Sanchez-Jimenez, M. Fonseca-Rodriguez, J. Arrieta-Solorzano, R. Rimolo-Donadio, A. Arnaud, M. Miguez, and J. Gak, "Siwa: A custom risc-v based system on chip (soc) for low power medical applications," *Microelectronics Journal*, p. 104753, 03 2020.

[5] Tecnológico de Costa Rica, "Costa Rica creó el primer microcontrolador para microcircuito de aplicaciones médicas totalmente diseñado y desarrollado en el país," https://www.tec.ac.cr/hoyeneltec/2019/05/08/costa-rica-creo-primer-microcontrolador-microcircuito-aplicaciones-medicas-totalmente, May 2019.

[6] R. Garcia-Ramirez, A. Chacon-Rodriguez, R. Castro-Gonzalez, A. Arnaud, M. Miguez, J. Gak, R. Molina-Robles, G. Madrigal-Boza, M. Oviedo-Hernandez, E. Solera-Bolanos, D. Salazar-Sibaja, D. Sanchez-Jimenez, M. Fonseca-Rodriguez, J. Arrieta-Solorzano, and R. Rimolo-Donadio, "Siwa: a risc-v rv32i based micro-controller for implantable medical applications," in *2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS)*, 2020, pp. 1–4.

[7] S. Govindaraman, Ashwini ; Gandhi, "USB 3.0—The Next-Generation Interconnect," vol. 58, no. 2, pp. 67–71, 2010. [Online]. Available: https://web.b.ebscohost.com/ehost/detail/detail?vid=0{&}sid=ede20de7-3739-4368-a362-ed2e02a173cf{%}40sessionmgr102{&}bdata=Jmxhbmc9ZGUmc2l0ZT1laG9zdC1saXZl{#}db=buh{&}AN=48079673

[8] "Universal Serial Bus 3.0 Specification," Sep. 2013. [Online]. Available: https://www.usb3.com/whitepapers/USB%203%200%20(11132008)-final.pdf

[9] Intel-Corporation, "USB 2.0 Transceiver Macrocell Interface (UTMI) Specification," 2001.

[10] D. Gebhardt and K. S. Stevens, "Elastic Flow in an Application Specific Network-on-Chip," *Electronic Notes in Theoretical Computer Science*, vol. 200, no. 1, pp. 3–15, 2008.

[11] M. Ehlert, "Different approaches of high speed data transmission standards," *Advances in Radio Science*, vol. 2, pp. 187–193, 2004.

[12] P. L. Yang, C. C. Lin, M. Z. Kuo, S. H. Dhong, C. M. Lin, K. Huang, C. N. Peng, and M. J. Wang, "A 4-GHz universal high-frequency on-chip testing platform for IP validation," *Proceedings of the IEEE VLSI Test Symposium*, no. 2, pp. 1–6, 2014.

[13] C. Effiong, G. Sassatelli, and A. Gamatie, "Distributed and dynamic shared-buffer router for high-performance interconnect," *2017 11th IEEE/ACM International Symposium on Networks-on-Chip, NOCS 2017*, 2017.

[14] J. Cortadella, M. Kishinevsky, and B. Grundmann, "SELF: Specification and design of a synchronous elastic architecture for DSM systems," no. January, 2005. [Online]. Available: http://www.cs.upc.edu/{~}jordicf/gavina/BIB/reports/self{_}tr.pdf

[15] J. Cortadella, M. Galceran-Oms, and M. Kishinevsky, "Elastic systems," *8th ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2010*, no. May, pp. 149–158, 2010.

[16] J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin, "Elastic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 1437–1455, 2009.

[17] K. Ren-Der, Chen; Sheng-Yu, "A Study on the Performance Characteristics of a Synchronous Elastic FIR Filter," *Lecture Notes in Electrical Engineering*, vol. 345, 2016.
[18] I. Seitanidis, A. Psarras, G. Dimitrakopoulos, and C. Nicopoulos, "ElastiStore: An elastic buffer architecture for Network-on-Chip routers," *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, vol. 1, pp. 1–6, 2014.
[19] G. Michelogiannakis and W. J. Dally, "Router designs for elastic buffer on-chip networks," *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09*, 2009.
[20] G. Michelogiannakis, J. Balfour, and W. J. Dally, "Elastic-buffer flow control for on-chip networks," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, Feb 2009, pp. 151–162.
[21] M. Hong, Chang; Te, Peng; Daoming, Ke; Tailong, Xu; Jian, "Design and implementation of elastic buffer for Universal Serial Bus 3.0," *IEICE Electronics Express*, vol. 13, no. 5, pp. 1–10, 2016.
[22] G. Michelogiannakis and W. J. Dally, "Elastic Buffer Flow Control for On-Chip Networks," *IEEE Transactions On Computers*, vol. 62, no. 2, pp. 295–309, 2013.
[23] I. Seitanidis, A. Psarras, K. Chrysanthou, C. Nicopoulos, and G. Dimitrakopoulos, "ElastiStore: Flexible Elastic Buffering for Virtual-Channel-Based Networks on Chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 12, pp. 3015–3028, 2015.
[24] A. Psarras, S. Moisidis, C. Nicopoulos, and G. Dimitrakopoulos, "Networks-on-Chip With Double-Data-Rate Links," *IEEE Transactions On Circuits And System*, vol. 64, no. 12, pp. 3103–3114, 2017.
[25] R. Louis, M. Vinodhini, and N. S. Murty, "Reliable router architecture with elastic buffer for NoC architecture," *2015 International Conference on VLSI Systems, Architecture, Technology and Applications, VLSI-SATA 2015*, pp. 1–4, 2015.
[26] M. R. Casu, "Improving synchronous elastic circuits: Token cages and half-buffer retiming," *Proceedings - International Symposium on Asynchronous Circuits and Systems*, pp. 128–137, 2010.
[27] L. Nazir and R. N. Mir, "Evaluation of efficient elastic buffers for network on chip router," *IEEE International Conference on Power, Control, Signals and Instrumentation Engineering, ICPCSI 2017*, pp. 2176–2180, 2018.
[28] S. M. T. Adl, M. Mirzaei, and S. Mohammadi, "Elastic buffer evaluation for link pipelining under process variation," *IET Circuits, Devices and Systems*, vol. 12, no. 5, pp. 645–654, 2018.
[29] S. Tovar, *Decodificador 132b/128b Compatible con USB 3.1 en Tecnología CMOS 180nm.* Universidad Industrial de Santander, 2017.
[30] D. León, *Diseño y Caracterización de Canales de Interconexión de Alta Velocidad para Estándar USB 3.0.* Instituto Tecnológico de Costa Rica, 2019.
[31] J. Aparicio, A. Quesada, J. Campos, B. Urrea, and D. Leon, *Documentación para la generación de un transceiver USB 3.0 y 2.0 para un proceso CMOS 180-nm.* Tecnológico de Costa Rica, 2021.
[32] IEEE, "Standard for information technology. part 3: 8b/10b transmission code," NA 2012.
[33] OutputLogic.com, "Scrambler Generator," http://outputlogic.com/?page_id=205, NA 2019.
[34] B. Sundeep, A. Thibbaiah, and S. Yellampalli, "Vlsi implementation of stm-1 framer and de-framer," 08 2012, pp. 239–243.
[35] A. Quesada-Martínez, J. Aparicio-Morales, J. Campos-Araya, A. Chacón-Rodríguez, R. Garcia-Ramirez, and R. Rimolo-Donadio, "Evaluation of 8b/10b fpga encoder implementations for serdes links," in *2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS)*, 2020, pp. 1–4.
[36] L. S. Corporation, *8b/10b Encoder/Decoder.* Lattice Semiconductor Corporation, 2012.
[37] B. C. Hien, S. M. Kim, and K. Cho, "Design of a wave-pipelined serializer-deserializer with an asynchronous protocol for high speed interfaces," *Proceedings of the 4th Asia Symposium on Quality Electronic Design, ASQED 2012*, pp. 265–268, 2012.
[38] S. Palermo, "Lecture 7: Equalization Introduction & TX FIR Eq," Spring 2019.
[39] A. Q. Martinez, *Design and Development of an FFE Equalizer at Integrated Circuit Level for High-Speed USB Links.* Costa Rica: Instituto Tecnologico de Costa Rica, 2020.
[40] M. Chen, J. Silva-Martinez, M. Nix, and M. E. Robinson, "Low-voltage low-power LVDS drivers," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 2, pp. 472–479, 2005.

[41] X. Bai, J. Zhao, S. Zuo, and Y. Zhou, "A 2.5 Gbps, 10-lane, low-power, LVDS transceiver in 28 nm CMOS technology," *Electronics (Switzerland)*, vol. 8, no. 3, 2019.
[42] H. C. Chow and W. W. Sheen, "Low power LVDS circuit for serial data communications," *Proceedings of 2005 International Symposium on Intelligent Signal Processing and Communication Systems, ISPACS 2005*, vol. 2005, pp. 293–296, 2005.
[43] B. Wei, Wang; James, "A 10-Gb/s, 107-mW Double-Edge Pulsewidth Modulation Transceiver," *IEEE Transactions On Circuits And System*, vol. 61, no. 4, pp. 1068–1080, 2014.
[44] D. Andreson, *Universal Serial Bus System Architecture.* Addison Wesley, 2001.
[45] S. Palermo, "Lecture 8: RX FIR, CTLE, & DFE Equalization," Spring 2014.
[46] B. Razavi, "Challenges in the design of high-speed clock and data recovery circuits," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 94–101, 2002.
[47] A. Technologies, *Agilent Signal Integrity Seminar 2013.* Keysight, 2013.
[48] L. Xu, L. Zheng, Y. Chen, and Z. Li, "Design and implementation of universal serial bus transceiver with verilog," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, 06 2014.
[49] S. M. Signal, *USB 2.0 Device Transceiver.* Soft Mixed Signal, 2003.
[50] R. Dwivedi and M. S. Narula, "Usb 2.0 transceiver macrocell interface implementation on xilinx vivado," in *2016 International Conference on Signal Processing and Communication (ICSC)*, 2016, pp. 419–424.
[51] G. Sung, L. Tung, H. Wang, and J. Lin, "Usb transceiver with a serial interface engine and fifo queue for efficient fpga-to-fpga communication," *IEEE Access*, vol. 8, pp. 69 788–69 799, 2020.
[52] G. W. den Besten, *The USB 2.0 Physical Layer: Standard and Implementation.* Boston, MA: Springer US, 2003, pp. 359–378. [Online]. Available: https://doi.org/10.1007/0-306-48707-1_17
[53] J. . Nam, Y. . Kim, K. . Choi, and H. . Park, "A utmi-compatible physical-layer usb2.0 transceiver chip," in *IEEE International [Systems-on-Chip] SOC Conference, 2003. Proceedings.*, 2003, pp. 309–312.
[54] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips, "Universal Serial Bus Specification," 2000.
[55] K. Babulu and K. S. Rajan, "Fpga implementation of usb transceiver macrocell interface with usb2.0 specifications," in *2008 First International Conference on Emerging Trends in Engineering and Technology*, 2008, pp. 966–970.
[56] S. K. Mahapatra, *FPGA Implementation of UTMI and Protocol Layer for USB 2.0.* India: National Institute of Technology Rourkela, 2013.
[57] C. Cummings, "Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog," in *SNUG Boston 2008*, 2008.
[58] C. Cummings and P. Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," Jan 2002.
[59] R. C. Gonzalez, *Integracion de un microprocesador de aplicacion especıfica en un flujo de diseno de circuitos integrados digitales.* Costa Rica: Instituto Tecnologico de Costa Rica, 2017.
[60] R. C. Gonzalez, G. M. Boza, M. O. Hernandez, and E. S. Bolanos, *Tutorial sobre el flujo automatizado de diseno de circuitos integrados digitales mediante Herramientas EDA de Synopsys con una tecnologıa X-FAB CMOS 180nm.* Costa Rica: Instituto Tecnologico de Costa Rica, 2019.
[61] W. F. Lee, *VHDL Coding and Logic Synthesis with SYNOPSYS.* USA: Academic Press, Inc., 2000.
[62] B. Hima and M. Hamid, *ASIC Design Flow Tutorial Using Synopsys Tools.* USA: San Francisco State University, 2009.
[63] H. Bhatnagar, *Advanced ASIC Chip Synthesis: Using Synopsys, Design Compiler and PrimeTime.* USA: Kluwer Academic Publishers, 1999.
[64] Synopsys, *Design Compiler, User Guide.* USA: Synopsys, 2016.
[65] Synopsys., *IC Compiler, Implementation User Guide.* USA: Synopsys, 2016.