Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Diseño Electrónico

TÍTULO DEL TRABAJO

Compute Express Link Optimization for Low Latency

TESIS que para obtener el **GRADO** de
MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: **ÁNGEL ADÁN BALTAZAR ORTIZ**

Director **FRANCISCO ELÍAS RANGEL PATIÑO**

Tlaquepaque, Jalisco. 10 de mayo de 2024.

Acknowledgment

I am deeply grateful to my supervisor, Dr. Francisco Elias Rangel Patino. He has been not merely a guide but also a beacon illuminating the path through all those experiments and failed theories. His sagacious counsel and expertise have been instrumental in shaping this thesis.

I would like to thank Intel Corporation for the financial support and for allowing me to pursue this study in parallel with my daily work within the Xeon Data Center CXL team. I extend my sincere gratitude to my colleagues for their invaluable support.

I want to dedicate this work to my parents, Angel and Imelda, who instilled in me the values of humility and perseverance. I also thank my family, friends, and beloved girlfriend for their understanding and support, even when I could not be present on special occasions.

Summary

This thesis delves into the architecture of the Compute Express Link® (CXL®) 1.1 specification, which is built on top of the physical and electrical interface of the peripheral component interconnect express® (PCIe®) Gen 5. CXL is a technology focused on efficiently handling generative artificial intelligence (AI) and its enormous memory performance demands. CXL enables coherency between the central processing unit (CPU) memory space and memory connected under devices for memory pooling.

As CXL technology gains traction, optimizing latency has become a significant area for improvement. Fortunately, new configurations are available that can improve the user experience for advanced workloads. One way to reduce latency is to use the equalization process in fixed mode during CXL link bring-up and avoid recoveries or retries under the link training state machine (LTSSM) setup that are continuously triggered due to the bit error. Some equalizers are subject to operate in dynamic mode and require real-time latency optimization.

The purpose of this thesis is to demonstrate the advantages and disadvantages of different optimization methods using latency measurement tools that are applicable only on the CXL validation stage. The study also utilizes optimization techniques implemented in MATLAB to determine the minimum global solution of an objective function.

This research suggests that creating tools that can monitor, analyze, and control the optimal CXL link latency can contribute and improve significantly to future CXL technology implementations.

Resumen

Esta tesis ahonda en la arquitectura de la especificación de Compute Express Link® (CXL®) 1.1, la cual está construida sobre la interface física y eléctrica de la interface PCI Express® (PCIe®, por sus siglas en inglés) de quinta generación. CXL es una tecnología enfocada en el manejo eficiente de inteligencia artificial (IA) generativa y su enorme demanda de rendimiento de memoria. CXL habilita coherencia entre el espacio de memoria de la unidad central de procesamiento (CPU, por sus siglas en inglés) y la memoria conectada sobre los dispositivos para la característica de gestión compartida de memoria.

A medida que la tecnología CXL gana terreno, la optimización de latencia se ha convertido en una gran área de oportunidad. Afortunadamente, nuevas configuraciones pueden mejorar la experiencia de uso de cargas de trabajo avanzados. Una forma de reducir la latencia es utilizando el proceso de ecualización en modo fijo duran el levantamiento del enlace de comunicación CXL, así como evitar recuperaciones o reintentos dentro del ajuste de la máquina de estados de entrenamiento del enlace (LTSSM, por sus siglas en inglés) que continuamente están siendo puestos en marcha debido a los errores de bit (BER, por sus siglas en inglés). Algunos ecualizadores están sujetos a operar en modo dinámico y requieren una optimización de latencia en tiempo real.

El propósito de esta tesis es demonstrar las ventajas y desventajas de los diferentes métodos de optimización usando herramientas de medición de latencia que solo aplican en la etapa de validación de CXL. El estudio también utiliza técnicas de optimización implementadas en MATLAB para determinar una solución mínima global de una función objetivo.

Esta investigación sugiere que la creación de herramientas que puedan monitorear, analizar y controlar la latencia óptima del enlace de comunicación CXL puede contribuir y mejorar significativamente las futuras implementaciones de tecnología CXL.

Content

Introduction

Some hardware components and applications, such as artificial intelligence (AI), advancing memory, cloud servers, and machine learning (ML), can yield significant performance benefits if a low-latency computer interface exists. The most widely used computer interface for these types of applications is the peripheral component interconnect express (PCIe). However, lower latencies are often needed to optimize such applications. In response to room for improvement, the Compute Express Link (CXL) standard was developed as a low-latency offshoot of PCIe and it is rapidly gaining acceptance in the chip architecture and system designs.

CXL is a standard interconnection developed by Intel to boost performance in data centers, allowing a high bandwidth between central processing unit (CPU)-to-memory and CPU-to-device. Current market devices that support CXL rely on the PCIe generation 5 (Gen5) physical layer, which covers three main areas: memory, cache coherence, and input/output (I/O) protocols.

According to the PCIe Gen5 specification, a natural latency scenario may occur during CXL transactions. This scenario requires each bit error (BER) of lane transceiver ratio to be $1x10^{-12}$ or there should be less than one error for every $1x10^{-12}$ bit. Therefore, one bit-flip error per Terabit of data could happen within the link every one second. In the best-case scenario and with ideal conditions, a PCIe link may still encounter a bit-flip error every 2 seconds despite being designed to meet the exact BER specification. A BER is under the "correctable error" category. While correctable errors affect performance (latency and bandwidth), no data or information is lost, and the PCIe fabric remains reliable for different types of transactions.

Besides the natural bit-flip error impact on latency in any high-performance center (HPC) using CXL communication faces compute latency, storage latency, and network latency; so knowing on detail the computer interface mechanism to modulate or fine-tune the equalization process on the channel during transmission/reception can bring huge benefits and tradeoffs to minimize latency.

Equalization is part of the device operating at 32 giga transfers per second (GT/s) (PCIe Gen5) to adjust the transmitter (Tx) equalization (EQ) and receiver (Rx) EQ setup to improve the signal quality. The EQ phases for the Gen5 device are consistent with the previous generations, comprising phases 0, 1, 2, and 3. Initially, the link should be trained to L0 at 2.5 GT/s, followed

by equalization at 8.0 GT/s, 16 GT/s, and 32 GT/s consecutively. This sequence is referred to as the conventional 'Full Equalization' mode. Nevertheless, the bandwidth limitation poses a significant challenge to high-speed wireline transceivers as the data rate increases, mainly due to channel and package loss. An increased data rate also means signal attenuation increases, impacting higher-frequency components and leading to distortion, which requires equalization.

This thesis hypothesizes that it is possible to significantly reduce CXL latency through a series of optimizations and procedural adjustments. Specifically, the research will investigate the following sub-hypotheses:

- Latency Reduction through Link Optimization: By optimizing the transmission (Tx Cm/Cp) and reception (Rx CTLE) parameters, it is hypothesized that the link equalization procedure can be bypassed entirely, particularly at lower data rates, thereby reducing overall latency.
- Firmware-Based Equalization Settings: It is hypothesized that storing optimal equalization settings directly in the firmware can eliminate the need for analog equalization (EQ), thus streamlining the data transmission process and reducing latency.
- Error Elimination and Latency Optimization: Finally, the thesis will test the hypothesis that optimizing the link can eliminate any correctable errors, maintaining low latency and improving overall system efficiency.

These hypotheses will guide the research and experimentation process, aiming to validate the potential for these optimizations to enhance CXL performance.

The main goal of this thesis is to reduce latency on a CXL link by modifying different parameters involved in the different components of the EQ process, such TxEQ, and continuous time linear equalization (CTLE) in the Rx. The objective is to maintain the observed latency modulation at a minimum level. This thesis is organized as follows:

Chapter 1 covers fundamental knowledge about the physical layer (PHY) of PCIe, its evolution at each generation, how data is packetized on the link, and the different elements capable of interacting within a PCIe network topology.

Chapter 2 presents a detailed comparison between a CXL and a PCIe device. It highlights the similarities and differences in their physical layer, data link layer, and elements in a CXL network topology. The chapter also emphasizes the significance of having three distinct subprotocols, each serving a specific purpose: IO, Cache, and Memory.

Chapter 3 illustrates how CXL memory access affects the latency of HPC as compared to other computer technologies. The chapter also introduces equalization modes that help reduce link bring-up time. Additionally, hardware components that help reduce latency in physical CXL channels are highlighted.

Chapter 4 explains the purpose of various circuits involved in the equalization process and provides insights into optimizing equalization. This chapter includes a description of the simulation setting along with the corresponding outcomes.

Chapter 5 describes the mechanism used for optimizing latency based on CTLE parameters. It describes optimization techniques such as Pattern-Search/Nelder-Mead, Genetic Algorithm, and Branch and Bound Discrete Optimization.

The conclusions presented in the last section provide a comprehensive overview of the key findings in this study regarding the optimization of CXL latency. Appendix A to F includes the MATLAB code used to simulate and optimize the CXL link.

1. PCI Express

1.1. What is PCIe?

The PCIe local bus is an interface that is fully serial and highly scalable. It offers important features such as a load-store architecture, software interfaces, and a usage model. By utilizing advancements in point-to-point interconnects, packetized protocol, and switch-based technology, this protocol can provide enhanced performance levels. The advanced capabilities of PCIe include quality of service (QoS), power management, data integrity, hot-plug/hot-swap support, and error handling [*1*]. *PCIe provides a high-speed, high-performance, point-to-point, dual simplex, differential signaling link for interconnecting devices. All data transmitted and received between two PCIe devices implies the usage of a separate pair set of signals for each type of communication with respect to the other* [*2*].

1.1.1 PCI Express Evolution

For over three decades, the PCIe architecture has been working in I/O connectivity, facilitating efficient, high-speed, and low-latency communication between components, as shown in [*3*].

TABLE 1

PCI EXPRESS SPECIFICATION EVOLUTION TAKEN FROM [3]

Base PCIe Specification							Data Rate (Gb/s)	Modulation	Encoding	x16 bandwidth after encoding overhead	Year (First Spec Revision)
1.x	2.x	3.x	4.x	5.x	6.x	7.x					
✓							2.5	NRZ	8b/10b	32 Gb/s	2003
✓	✓						5.0	NRZ	8b/10b	64 Gb/s	2007
✓	✓	✓					8.0	NRZ	128b/130b	126 Gb/s	2010
✓	✓	✓	✓				16.0	NRZ	128b/130b	252 Gb/s	2017
✓	✓	✓	✓	✓			32.0	NRZ	128b/130b	504 Gb/s	2019
✓	✓	✓	✓	✓	✓		64.0	PAM4	1b/1b	1024 Gb/s	2021
✓	✓	✓	✓	✓	✓	✓	128.0	PAM4	1b/1b	242 GB/s	2022

1. PCI EXPRESS

PCIe seventh generation technology will debut in 2025, doubling the bandwidth to 128 Giga-transfers per second (GT/s) as expected after PCIe sixth generation and maintaining complete backward compatibility, as illustrated in Figure 1.

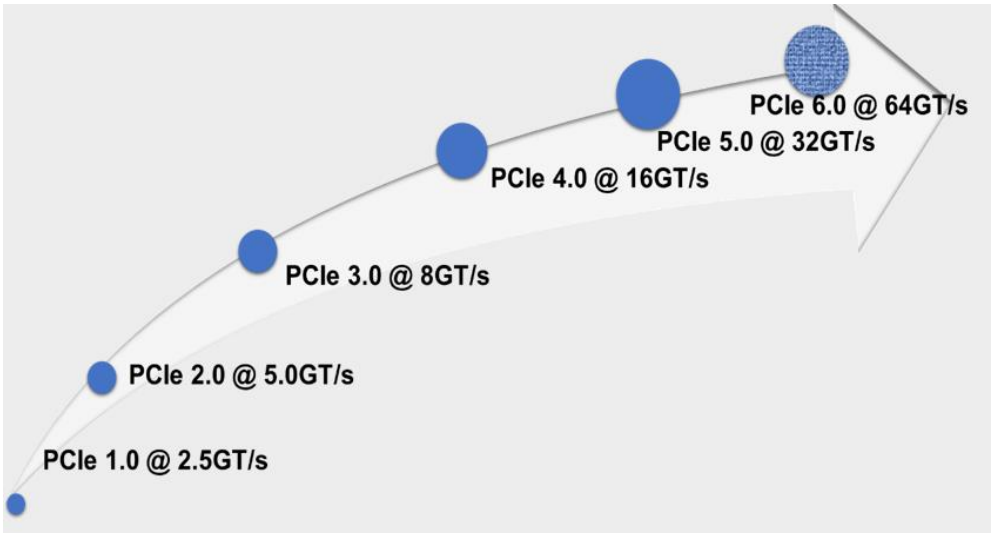


Figure 1 PCI Express Bandwidth Evolution. Figure taken from [*3*]

1.1.2 PCI Express Link

A link represents a dual-simplex communication channel between two components, as shown in Figure 2. The essential PCIe connection consists of a double pair of differential low-voltage signals, one for transmission and the other for reception [*1*].

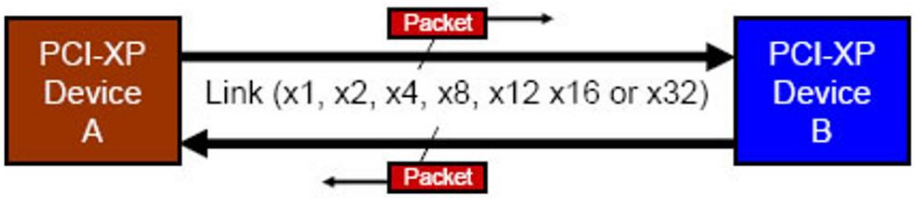


Figure 2 PCI Express Link. Figure taken from [*4*]

PCIe uses a packet-based protocol based on encoding transactions. Data packets are sent and received in serial mode one at a time, byte-striped across all link lanes [*2*]. Increasing the

number of lanes implemented on a link will increase the packet transmission speed and the link bandwidth.

The primary link attributes of PCIe link are:

- *Basic link*: Dual unidirectional differential links are employed by PCIe links, with one link serving as the transmit pair and the other link as the receive pair. A data clock that uses an encoding scheme is embedded to achieve high data rates [*1*].
- *Signaling method*: Every new generation of PCIe signaling method has matured in more than one characteristic, always looking for increased bandwidth. GT/s term stands for the number of encoded bits transferred in a second in the direction of a lane. A combination of modulation method, encoding method, and data rate are strong dependencies for producing an effective data rate [*1*].
- *Lanes*: Most basic link configurations should support one lane. Each lane constitutes a set of differential signal pairs: one for transmission and one for reception. A link with multiple lanes will act as a scaled bandwidth represented by width factors of x1, x2, x4, x8, and x16 [*1*].
- *Initialization*: Each PCIe link undergoes two negotiation processes during hardware initialization: lane widths and frequency of operation are negotiated by the two agents at each end of the link. There is no firmware or operating system software involved at this point [*1*].
- *Symmetry*: Depending on the link configuration, each direction should have a symmetric number of lanes, i.e., a maximum bandwidth scale factor (x16), indicating 16 differential signal pairs in each direction [*1*].

Figure 3 shows a graphical representation of the most common terminology used in the PCIe link concept:

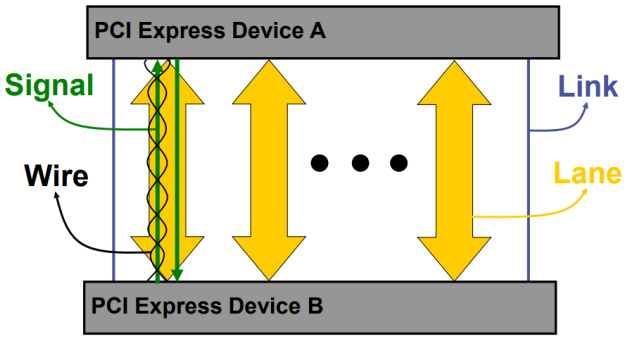


Figure 3 PCIe Terminology. Figure taken from [*4*]

1.1.3 PCIe Fabric Topology

A PCIe fabric comprises point-to-point links that interconnect a set of components and Figure 4 represents an example of such fabric topology:

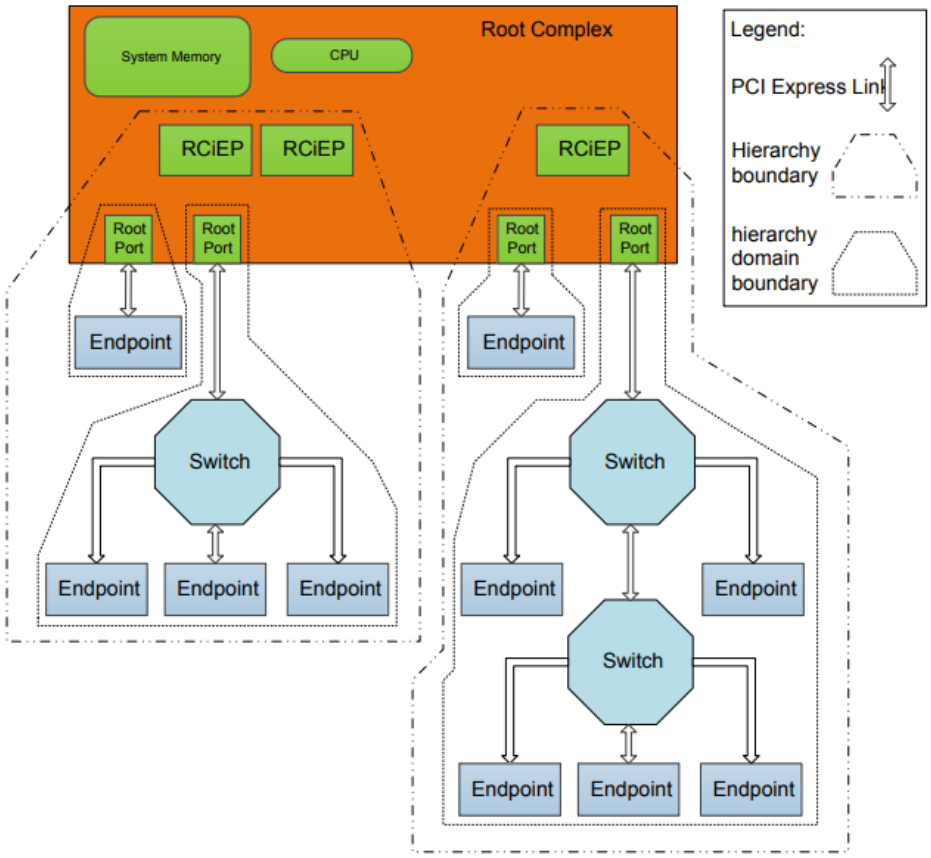


Figure 4 Example PCIe Topology. Figure taken from [*1*]

Figure 4 illustrates an example of a single fabric instance with two hierarchies composed of a root complex, multiple endpoints, and multiple switches, all interconnected via PCIe links.

The following bullet section describes in high detail the most representative elements within a PCIe topology:

- *Root complex*: It is the mechanism linking the CPU and the memory subsystem to the PCIe fabric. It can contain one or various PCIe ports [*2*].
- *Hierarchy boundary*: The limits between all the devices and links associated with a root complex. These devices and links may connect directly to the root complex through its port(s) or indirectly through switches and bridges [*2*].
- *Hierarchy domain boundary:* It is the complex that contains devices and links that correspond with one port of the root complex [*2*].
- *Endpoint*: It is the function that can request or complete a PCIe transaction, each of two ways: on its behalf or on behalf of a distinct non-PCIe device (other than a PCIe device or host CPU) [*1*].
- *Switch*: comprises the logical construction of multiple virtual PCIe-to-PCIe Bridge devices [*1*].

2. Compute Express Link

2.1. What is CXL?

The next worldwide data center capacity expansion is merging different technologies to become a reliable mechanism. Due to this capacity expansion, a significant update impacts data center computing, storage, and network capabilities infrastructures. Compute Express Link® (CXL®) successfully solves this problem as a solution that will rapidly grow in most data centers in coming years and perform better than PCIe. Besides, it adds lower high bandwidth and low latency protocols on top of PCIe for applications such as accelerator and memory buffers [*5*].

The core of CXL resides in a multiprotocol technology intended to handle devices categorized as accelerator and memory devices. Analogous to PCIe, CXL offers support with a bast set of protocols that covers I/O semantics (CXL.io), caching semantics (CXL.cache), and memory semantics (CXL.mem) through a discrete link. Discovery and enumeration, error report, and host physical address (HPA) lookup are mandatory tasks realized through the CXL.io protocol. Alternatively, depending on the proper application, CXL devices can support CXL.mem and CXL.cache.

The biggest advantage of using CXL against PCIe is that it operates in a low-latency, high-bandwidth path for an accelerator to access the system; in other words, it accesses memory attached

to the CXL device. Figure 5 illustrates a conceptual representation of a device that is connected to a Host processor using CXL [*6*].

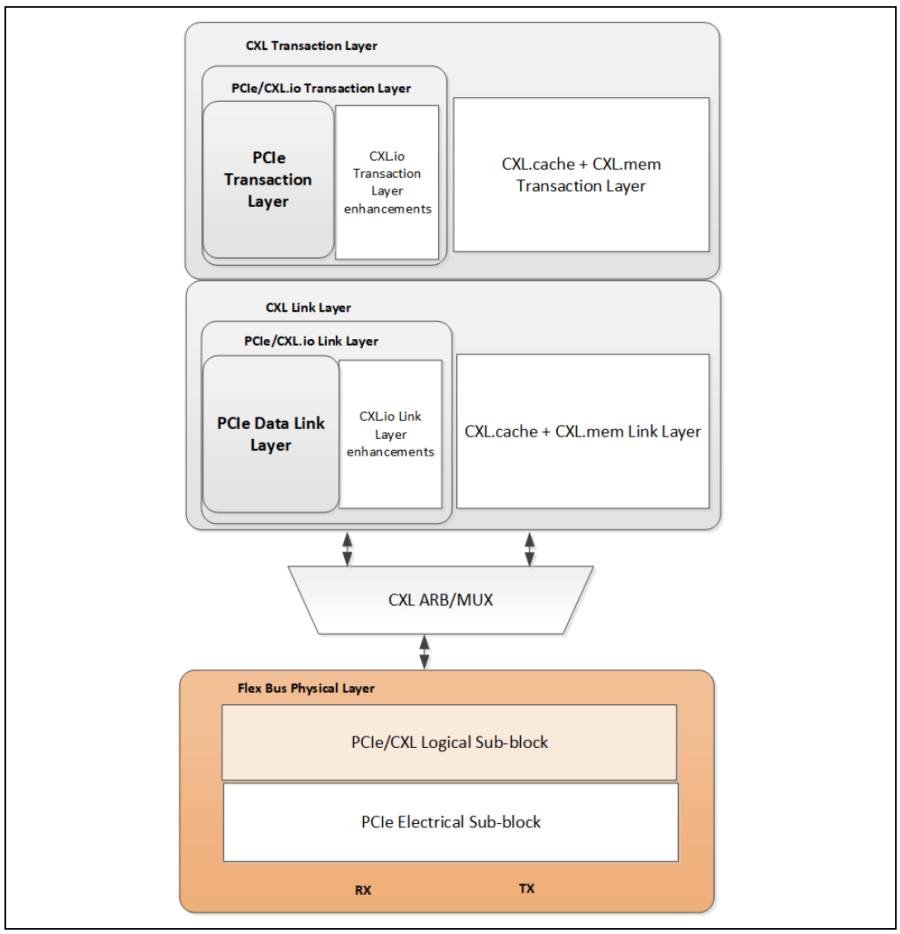


Figure 5 Flex Bus Layer Physical. Figure taken from [*6*]

2.2. CXL Physical Layer

Thanks to the Flex Bus unit that implements a high bandwidth and off-package link, it is possible by design to select which protocol to operate, PCIe or CXL. This election occurs during link training via alternate protocol negotiation and depends on the device connected to the slot. The Flex Bus operates under the same PCIe physical lines, ensuring the reuse of PCIe retimers with the same form factors available for PCIe. Figure 6 provides a high-level diagram of a Flex Bus port, illustrating the usage of a slot and custom accelerator soldered down on the main board

[7]. A slot implementation can strategically place a CXL or PCIe card and one or two optional retimers in the middle of the trajectory to increase the distance between them.

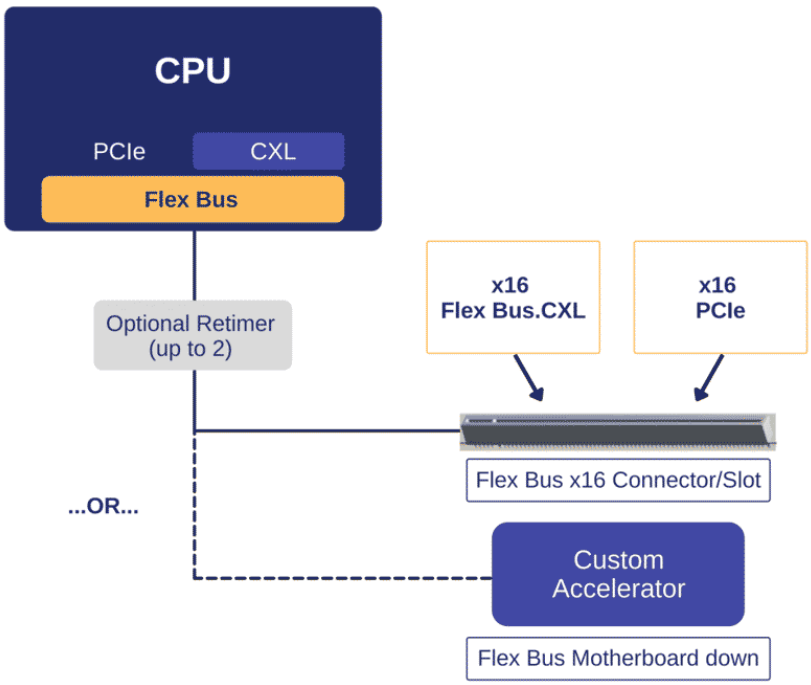


Figure 6 Flex Bus Diagram. Figure taken from [7]

The device slot operation with the host system, whether PCIe or CXL, uses the Flex Bus mechanism. The first point during link training is to ensure the CXL device can initialize negotiation into 2.5GT/s by utilizing 8b/10b encoding.

After the first point, at the time of the Link Training State Machine (LTSSM) setup, the CXL device can notify that it can handle CXL during alternate mode negotiation. If the alternate mode is available on both sides of the link, the training will enter the L0 state at 2.5 GT/s [5]. This state is part of the LTSMM rules that are part of the PCIe specification. It is important to note that if CXL operation is required instead of PCIe on a physical channel, the link training must reach at least 8 GT/s, also known as the degraded mode of operation in CXL. However, according to the CXL specification, if CXL mode is preferred during link training, both sides will require at least 32 GT/s.

The system has a built-in mechanism that automatically selects the non-operational behavior if the link cannot negotiate a speed higher than 8 GT/s. When the applications demand performance, the CXL channel must operate at a normal high speed of at least 32 GT/s.

For validation engineers, the relative operation between CXL and PCIe can add some difficulties to their validations. Also, assuming the PCIe LTSSM subject is dominant and has no relation with the Alternate Mode Negotiation mechanism needed to handle CXL operation, this could represent a problem. CXL operation is a more dynamic mechanism than PCIe operation, where it is optional to anticipate the exact number of CXL lanes against PCI lanes required at the time of system design [5].

Based on system needs, it is required to designate the number of lanes to be used for each slot, which means being able to support just PCIe or CXL devices. In comparison to PCIe, the electrical signaling remains equal, and the Link/Transaction layer is reported differently.
Regardless of low-latency and overhead optimizations introduced to CXL set side by side with PCIe operation, these can be represented as overhead. Therefore, critical CXL optimizations are implemented for low-latency transactions to cache and memory. Flow Control UnIT (FLIT) encoding is the most critical optimization [5].

2.3. CXL Protocol

2.3.1 FLIT Mode

With the arrival of PCIe 6.0, a new feature called FLIT mode was introduced. In this mode, packets are encapsulated into flow control units of fixed sizes that differ from previous PCIe generations.

The original purpose of launching FLIT mode is error correction, which implies working with discrete fixed-size packets. Nevertheless, FLIT mode signifies data administration at the controller side, which brings better bandwidth efficiency, lower latency, and a smaller controller footprint. In relation to the usage of fixed-size packets at the Physical Layer, the FLIT implementation algorithm allows reducing 4 bytes for every packet. The algorithm removes the

128/130B encoding mechanism and DLLP overhead from previous PCIe specifications [*8*]. Implementing smaller packets increases efficiency at TLP.

2.3.1 Subprotocols

The implementation of CXL devices involves using partial or full CXL protocol features. The three subprotocols, CXL.io, CXL.memory, and CXL.cache, were specifically designed to simplify design and address unique use cases within CXL scenarios. Each of these subprotocols plays a crucial role in the overall implementation, depending on the specific requirements and complexities of the design.

CXL.io is equivalent to the PCIe transaction layer and provides basic actions such as discovery, configuration, and interrupts. Therefore, all CXL devices are required to support CXL.io. According to the CXL specification, CXL Type 1 can use the CXL.cache subprotocol optionally. The processor inside of the accelerator is a specific-purpose device optimized for a singular computation task. Using the CXL.cache protocol, as shown in Figure 7, will facilitate the accelerator device to access the CPU-attached memory, allowing the cache under the device to be coherent with the memory system and keeping a minimal overhead [*5*].

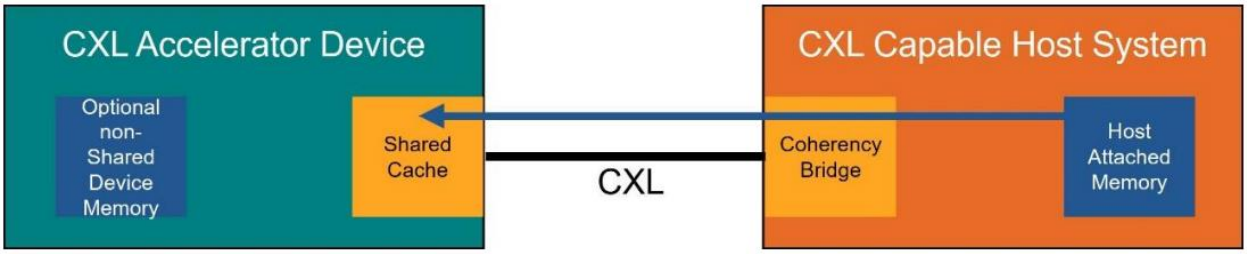


Figure 7 CXL.cache allows sharing cache between a host and acceleration device. Figure taken from [*5*]

As a result, computation will be executed more rapidly and decrease the interaction with power-hungry or busy CPUs.

As an option subprotocol, CXL.memory permits maintaining a CPU accessing memory from an expansion device or buffer, also known as a CXL Type 3 device within the CXL specification context represented by Figure 8.

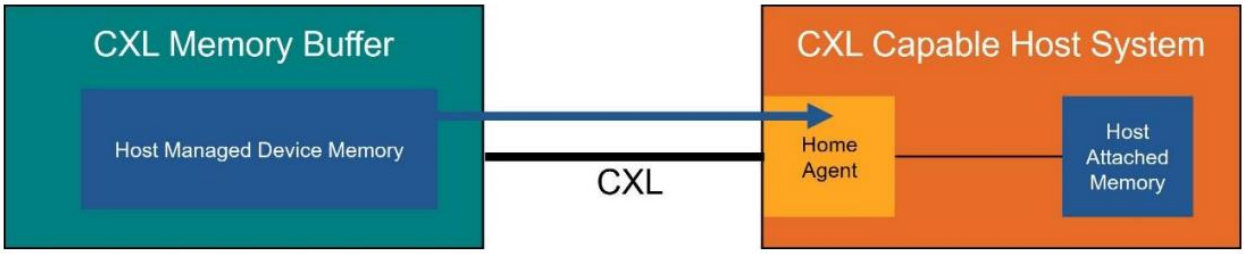


Figure 8 CXL.mem allows a host to access memory on an attached Memory Buffer device. Figure taken from [*5*]

Regarding access latency, it is crucial to utilize DRAM and NAND connected to CXL slots for persistent memory activities. Within a PCIe bus system, all CXL Type 3 devices will permit memory expansion. Optional CXL.cache/CXL.mem subprotocols can be part of trade-offs depending on performance system demands and system cost.

As per the CXL specification, a CXL Type 2 device is not just an accelerator but a versatile one with memory within the same board. This device implements both CXL subprotocol options, CXL.cache and CXL.memory. The CXL.cache feature enables the device to access the CPU memory, while the CXL.memory feature allows the CPU to access the memory attached to the endpoint.

Capability complexity can be increased or reduced depending on the sub-protocols incorporated into the device. Isolating CXL capabilities will help increase analysis and validation [*5*].

3. Latency

3.1. What is Latency?

In a high-speed IO channel communication, the time it takes for a data packet to travel from its source to its destination is known as latency, and it is caused by physics, processing, and queuing. [*9*]. PCIe stands out as one of the most latency-sensitive models among serial communication protocols. Its address-based semantics often lead to processor threads waiting for the results of a data negotiation. The arrival of PCIe Gen5 and PCIe Gen6 has required the implementation of retimers for more extended PCIe signal designs [*10*] to maintain consistent latency.

3.2. Latency in CXL

Application latency emerges from multiple sources. The next table [*11*] illustrates six different layers, the typical latency extra, the origin of such latency, and methods that can appear to reduce the latency at specific layers. Latency must be improved from top to bottom in Table 2:

TABLE 2

TYPICAL LATENCY FOR DIFFERENT SW/HW LAYERS

Layer	Typical Latency	Latency Source	Minimization Methods
PCIe Driver & OS Kernel	~ms to seconds	OS scheduler and PCIe driver	Use kernel bypass (e.g., DPDK), HW assist acceleration
Transaction layer	~500ns	Protocol-related credit & transactions	Use alternative protocol (e.g. CXL)
Data Link Layer	No errors: few ns Retry: ~us to ms	Error triggered retransmit	Improve the system or link Bit Error Ratio (BER) by using retimers
Physical Layer – Logical Portion	Few ns	Encode/Decode serial to parallel	Optimize the silicon architecture
Physical Layer – Electrical Portion	No errors: few ns Link recovery: ~ms	Poor Signal Integrity (SI) and Noise Can Trigger Link Recovery	Ensure that there is enough SI margin
Retimers	64ns or less	Elastic buffer, lane to lane skew compensation	Use common clocks and control skew on a system

basis allowing the use of
a short bypass buffer

From the top to the middle to the bottom of the table, saving seconds to milliseconds to microseconds to nanoseconds is possible. It is important to note that each element of latency contributes to global system performance, and reducing latency can be implemented at each level where feasible and economic [*10*].

The main memory latency adopted by the CXL consortium is 80-140ns, while the CXL memory latency ranges from 170-250ns as stated in section 3.1 before [*11*], where the CXL consortium conducted detailed technical analyses and experiments to measure the additional latency introduced by CXL memory compared to traditional memory access methods. These measurements would have involved benchmarking various memory operations and configurations to determine the precise latency overhead introduced by CXL. The latency range of 170-250ns reflects the additional time it takes for data to travel through the CXL interconnect and be processed by the memory subsystem. This range provides a realistic estimate of the performance impact that users can expect when utilizing CXL memory in their systems. Figure 9 shows the location of CXL memory latency-access in relation to other types of memory technologies.

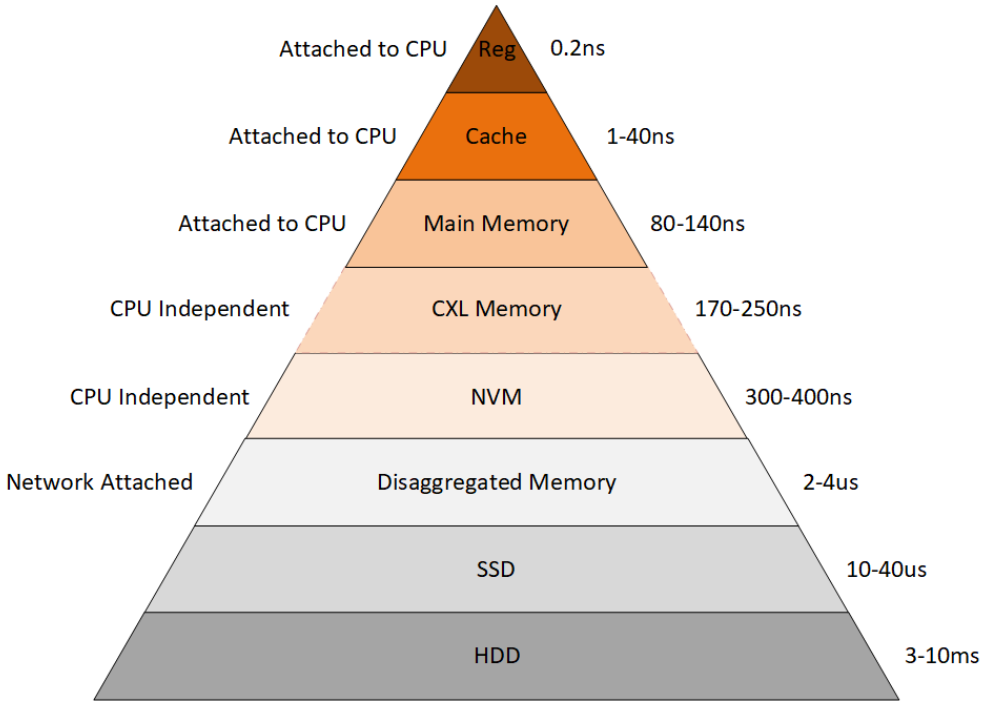


Figure 9 Pyramidal latency from different memory technologies. Figure taken from [*11*].

For CXL users in the era of data bandwidth more significant than 1 Terabits per second (Tbps), this can be typically handled in a 16-lane (x16) link between two CXL devices, with each lane transporting 32 Gigabits per second (Gbps). Since CXL relies on PCIe protocol and PCIe mandates a BER of $1x10^{-12}$ (or better) for each transceiver lane, this corresponds to at least one error or less for every $1x10^{12}$ bits transferred between two CXL devices. In other words, a bit-flip error could be present on the CXL link every 1 second. These bit-flip errors, which are part of the correctable errors category based on PCIe specification, can increase latency and bandwidth. It is important to understand that depending on the position where the bit-flip error occurs within the data stream and the state of the link training and LTSSM, it is possible to determine the type of latency error introduced to the system. This understanding is crucial for managing the performance of your CXL devices, as illustrated in the following examples:

- Data packets are transmitted and received during the L0 LTSSM state. Suppose an invalid bit happens at Data Link Layer Packets (DLLP). In that case, it can create a failure in the data integrity check and reflect such behavior at the Cyclic Redundancy Check (CRC) mechanism, which is part of each transmitted packet. When the Data Link Layer (DLL) discovers a packet with a CRC error, the receiver CXL device will report a Negative Acknowledgement (NACK) to the original transmitter CXL device. Once the link partner has received NACK, the transmission from that packet will restart. If the BER is within the specified, the link will experience an increase in latency.
- Using an encoding scheme of 128b/130b for 8.0Gbps and above rates means a 2-bit sync header and 128-bit data payload form a 130-bit block. There are two types of sync header encodings: 2b’10 related to data blocks and 2b’01 for ordered set blocks. Considering these sync header bits exist within a chunk of 130bits, a straightforward probability analysis and considering all lanes are operating in the worst-case scenario of BER ($1x10^{-12}$), there is an opportunity of an error appearing on the sync header in one of the x16 lanes at every minute.

If an invalid sync header is received during the L0 state, the PCIe controller stops data traffic and switches the link to the recovery LTSSM state. This transition enables the PCIe controller to regain control of the physical layer (PHY) transceivers. It allows them to retrain and regain block lock, among other tasks, to re-establish a stable link connection [*12*]. One

microsecond is the maximum trip time into the Recovery LTSSM state, adding a considerable delay to the original NACK packet and the subsequent until the link returns to the L0 state. In addition, other situations related to the CXL protocol can send the link to the Recovery state, such as overhead tokens and corrupted data sets [*12*]. Table 3 shows the relative probabilities and consequences of different error events:

TABLE 3

RELATIVE PROBABILITIES AND CONSEQUENCES OF DIFFERENT ERROR EVENTS. TAKEN FROM [*12*]

Data type	Percentage of link bandwidth	Error probability per link	Error consequences
Sync Headers	~1.5%	~1.5×10^{-14}	Link to Recovery LTSSM
Skip Ordered Set	~0.26% (separate reference no spread spectrum SNRS)	2.6×10^{-15}	Link to Recovery LTSSM
	~2.6% (separate reference independent spread spectrum SRIS)	2.6×10^{-14}	
Framing Tokens, dominated by idle tokens (IDLs)	1%	1×10^{-14}	Link to Recovery LTSSM
Data payload	Remainder	~1×10^{-12}	Data replay by the Data Link Layer

3.3. Reducing latency in CXL devices

Previous research and developments in CXL latency optimization have resulted in significant advancements, particularly at the data link layer. For example, Synopsys, Inc. announced the availability of its optimized DesignWare physical and controller intellectual property (IP) solution for PCIe 4.0 architecture, which has been shown to reduce latency by up to 20 percent and area by 15 percent when compared to previous implementations [*13*].

Additional research to reduce latency suggests using re-timers and re-drivers [*14*]. These components are important for improving eye height (EH) and width (EW) in different scenarios. They help to increase EW by reducing various sources of jitter, such as data-dependent jitter (DDJ) caused by inter-symbol interference (ISI), random jitter (RJ) due to thermal noise, clock jitter, and

bounded uncorrected jitter (BUJ) resulting from crosstalk. By addressing these sources of jitter, re-timers significantly improve signal integrity and reduce the likelihood of the LTSSM entering the recovery state, directly impacting the latency.

Another study [*15*] has demonstrated significant latency issues when the link training and status state machine (LTSSM) is forced into the recovery state due to poor signal quality. This situation can be exacerbated by factors such as high or fluctuating temperatures. It can take the recovery state machine hundreds of milliseconds or even longer to establish a new configuration for the transmit equalizers.

However, these options rely on a hardware component that is more difficult for customers to access and modify. This type of research focuses on equalization parameters and offers a software-oriented solution, which can be beneficial for optimizing latency in CXL at the customer level.

3.3.1 Equalization Modes: Reducing Link Bring-Up Time

Besides increasing bandwidth by two times in PCIe Gen5 from previous generations, it delivers other new features such as [*15*]:

- Modes that bypass equalization can achieve quicker link initialization.
- While conducting link training, to permit alternative protocols to engage in negotiation.
- Assistance for precoding to prevent burst errors.
- Enhancements in loopback facilitate the crosstalk behavior.

Increasing the data rate in PCIe Gen5 also brings a significant challenge-signal attenuation. When data rates go up, this becomes more time-consuming and increases the latency of link bring-up. Between different CXL vendors, it takes approximately ~100ms for a single speed transition with equalization, and it will take ~300ms to bring up a link to 32GT/s. These times are derived from in-house simulations [*15*] of a general CXL system showing the duration required for link training at various speeds (2.5 GT/s, 8.0 GT/s, 16 GT/s, and 32 GT/s) as shown below:

4. CXL PHY EQUALIZATION MODES

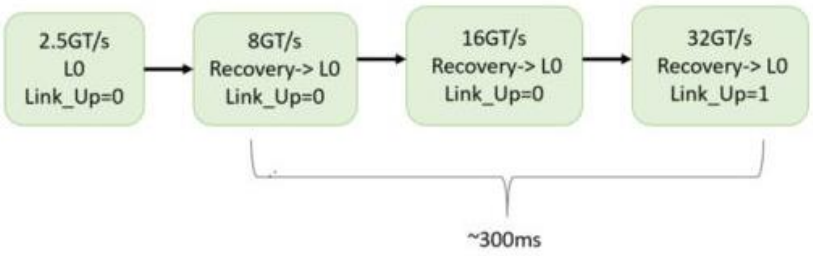


Figure 10 Time involved for bring-up CXL system. Figure taken from [*15*]

Table 4 shows the two new optional modes that can optimize link bring-up:

TABLE 4

BRING-UP MODES. TAKEN FROM [*15*]

Equalization Bypass to Highest Data Rate	No Equalization Needed
• Optimizing the link to skip equalization at lower data rates supporting 32Gtps. • What if Endpoint is never expected to operate at 8Gtps and 16GTps. • Link up to 32Gtps can be much faster by skipping intermediate equalization at 8.0Gtps and 16Gtps speeds and directly performing 32Gtps equalization. • Only operable speeds are 2.5Gtps, 5Gtps, and 32Gtps. • Both the link partners should support if or bypassing link equalization directly to the highest data rate.	• It is the fastest way to get to 32Gtps by skipping the Link equalization procedure entirely. • For this, all devices must know the optimal link equalization settings beforehand. • For skipping the equalization procedure entirely, all the devices in the link chain must support it.

The CXL Rambus [*16*] controller, a powerhouse capable of CXL 2.0 and PCIe Gen5, utilizes a robust 256-bit advanced encryption standard, Galois/Counter mode (AES-GCM) symmetric-key cryptographic block cipher. This cutting-edge encryption ensures that designers and security architects can trust in the integrity, confidentiality, and replay protection of traffic running within CXL and PCIe links. Here are the standout features [*17*]:

- Zero latency on integrity and data encryption (IDE) Security for CXL.mem and CXL.cache.
- Minimize the security attacks, financial, and brand reputation risks.
- Rambus CXL 2.0 and PCIe Gen5 controllers decrease implementation risks and speed time-to-market thanks to IDE modules pre-integrated.

– Integrating Rambus CXL 2.0 and PCIe Gen5 PHYs in controllers will complete CXL 2.0 and PCIe Gen5 interconnect subsystems.

3.3.2 Re-driver

A re-driver is a signal conditioning component used to improve the quality of high-speed data transmission. Specifically, it boosts and cleans up the signal as it travels along the transmission path.

This device does not use a clock and is completely constituted of pure analog devices, excluding the presence of a sideband low-frequency bus to program its analog features.

It is possible to configure the most basic program options using strap pins. Re-drivers do not store data digitally but rather compensate for inter-symbol interference, causing a delay of around 100 picoseconds and usually adding some jitter. Nevertheless, a re-driver is not essential to the High-Speed serial bus [*14*].

3.3.3 Re-timer

In contrast to a redriver, a re-timer uses a clock and data recovery circuit, which is the main component of a Serializer/Deserializer (SerDes) PHY. Besides, a re-timer uses a phase-locked loop (PLL), which requires an input reference clock, converting this component into a mixed analog-digital device. It transforms an incoming analog bit stream into digital bits that are part of the internal memory. Analog information from the original bit stream is not part of the internal digital data.

As the name suggests, a re-timer re-transmits data again by using new equalization and a new jitter packet that is not related analog-wise to the input bit stream. Hence, a re-timer interrupts activity links into two different sub-links independent from the Signal Integrity perspective (analog amplitude and timing). Some re-timers can debug capabilities, including margining, link status, and link health indicators. On the other hand, sophisticated retimers that can handle protocol

awareness of digital logic to manage link initialization, training, data encoding and decoding, and clock domain frequency differences [*14*].

In conclusion, a re-timer is a more complex device than a re-driver, bigger in die and package size, higher in cost, and fewer vendors offer this type of solution. There are two types of Re-timers, simple “Bit Re-timers” and Intelligent Re-timers. A bit re-timer is protocol-independent and commonly TxEQ training-incapable, although an intelligent re-timer is protocol-aware and TxEQ training capable.

4. CXL PHY Equalization Modes

4.1. Tx and Rx Equalizer

The PCIe specification outlines the requirements for implementing EQ at the Tx and Rx to reduce unwanted effects and lower the BER. The signal on Tx EQ can be reshaped prior to transmission, effectively surpassing all distortion the channel adds. On the other hand, at Rx EQ, signal quality can improve through reconditioning the original signal received [*18*].

For PCIe Gen3 to Gen5, a simple 3-tap FIR filter has been implemented into the equalizer at the transmitter side (TxEQ), as shown in *Figure 11*.

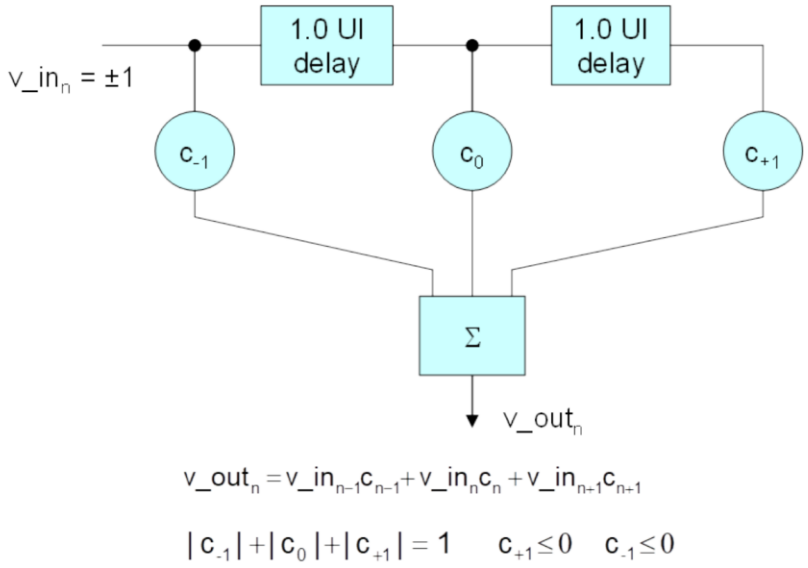


Figure 11 Tx EQ FIR representation. Figure taken from [*19*]

Within PCIe [*1*] and high-speed serial communication, "1.0 UI" refers to 1.0 unit interval (UI). A UI measures the time duration representing one bit in a serial data stream. This concept is essential in analyzing and designing high-speed digital communication systems.

Three consecutive received pulses (v_in_{n-1}, v_in_n, and v_in_{n+1}) multiply three different filter coefficients (C_{-1}, C_0, and C_{+1}) , and then multiplier outputs are added together to produce the final

filter output. C_{-1}, C_0, and C_{+1} are the pre-cursor, cursor, and post-cursor coefficients, respectively [*18*].

In some of the protocols of LTSSM, the Rx can communicate coefficient values in TS1/TS2 symbols so that Tx can use them or load them from Tx registers [*20*].

In contrast to the 3-tap FIR filter implemented in PCIe Gen3 through PCIe Gen5, as shown in Figure 5, and the PCIe Gen6 uses a 4-tap FIR filter, as shown in Figure 12.

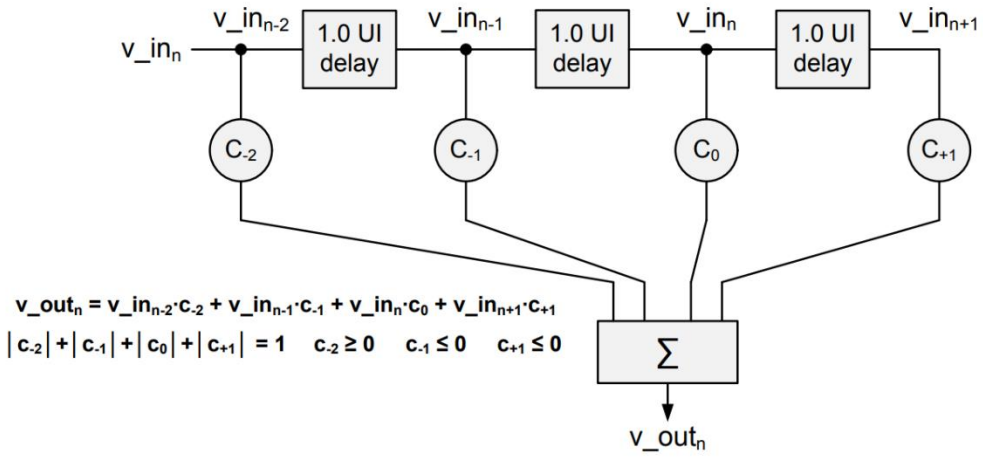


Figure 12 Tx EQ FIR representation for 64.0 GT/s. Figure taken from [*1*]

The coefficients in a 4-tap FIR filter are precursor, precursor, cursor, and post cursor. The only difference with a 3-tap FIR filter is that the second precursor is not used.

PCIe defines ten presets available for Tx/Rx EQ to reach from 8 GT/s/32 GT/s (PCIe Gen3-Gen5) to 64 GT/s (PCIe Gen6) with some differences in coefficient values and voltage levels between PCIe Gen3 - Gen5 (Table 5) and PCIe Gen6 (Table 6).

TABLE 5

TX PRESETS VALUES FOR AT 8/16/32 GT/S [*19*]

Preset #	Preshoot (dB)	De-emphasis (dB)	C_{-1}	C_{+1}	Va/Vd	Vb/Vd	Vc/Vd
P4	0.000	0.000	0.000	0.000	1.000	1.000	1.000
P1	0.000	-3.5±1dB	0.000	-0.167	1.000	0.668	0.668
P0	0.000	-6.0±1.5dB	0.000	-0.250	1.000	0.500	0.500
P9	3.5±1dB	0.000	-0.166	0.000	0.668	0.668	1.000
P8	3.5±1dB	-3.5±1dB	-0.125	-0.125	0.750	0.500	0.750
P7	3.5±1dB	-6.0±1.5dB	-0.100	-0.200	0.800	0.400	0.600
P5	1.9±1dB	0.000	-0.100	0.000	0.800	0.800	1.000

P6	2.5±1dB	0.000	-0.125	0.000	0.750	0.750	1.000
P3	0.000	-2.5±1dB	0.000	-0.125	1.000	0.750	0.750
P2	0.000	-4.4±1.5dB	0.000	-0.200	1.000	0.600	0.600
P10	0.000	Note 2	0.000	Note 2	1.000	Note 2	Note 2

TABLE 6

TX PRESET RATIOS AND CORRESPONDING COEFFICIENT VALUES FOR 64.0 GT/S [*1*]

Preset #	Preshoot 2 (dB)	Preshoot 1 (dB)	De-emphasis (dB)	C_{-2}	C_{-1}	C_{+1}	Va/Vd	Vb/Vd	Vc1/Vd	Vc2/Vd
Q0	0.0±0.5dB	0.0±0.5dB	0.0±0.5dB	0.000	0.000	0.000	1.000	1.000	1.000	1.000
Q1	0.0±0.5dB	1.6±0.5dB	0.0±0.5dB	0.000	-0.083	0.000	0.834	0.834	1.000	0.834
Q2	0.0±0.5dB	3.5±0.5dB	0.0±0.5dB	0.000	-0.167	0.000	0.666	0.666	1.000	0.666
Q3	0.0±0.5dB	0.0±0.5dB	-1.6±0.5dB	0.000	0.000	-0.083	1.000	0.834	0.834	0.834
Q4	0.0±0.5dB	0.0±0.5dB	-3.5±0.5dB	0.000	0.000	-0.167	1.000	0.666	0.666	0.666
Q5	-1.3±0.5dB	4.7±1.0dB	0.0±0.5dB	0.042	-0.208	0.000	0.584	0.584	1.000	0.500
Q6	-1.6±0.5dB	3.5±0.5dB	-3.5±0.5dB	0.042	-0.125	-0.125	0.750	0.500	0.750	0.416
Q7	-2.9±0.5dB	4.7±1.0dB	0.0±0.5dB	0.083	-0.208	0.000	0.584	0.584	1.000	0.418
Q8	-3.5±0.5dB	6.0±1.0dB	0.0±0.5dB	0.083	-0.250	0.000	0.500	0.500	1.000	0.418
Q9	-4.4±1.0dB	6.9±1.0dB	-1.6±0.5dB	0.083	-0.250	-0.042	0.500	0.416	0.916	0.250
Q10	0.0±0.5dB	0.0±0.5dB	Note 2	0.000	0.000	Note 2	1.000	Note 2	Note 2	Note 2

From the Rx equalization perspective, a continuous-time linear equalizer (CTLE) delivers a linear amplifier to boost the signal amplitude and model the waveform by offering a peaking at high frequencies. After the CTLE stage, an automatic gain control (AGC) tailors the data amplitude to a helpful level ahead of the decision feedback equalizer (DFE) stage, which implements additional equalization and sampling [*21*].

4.2. Equalization Modes

The high demand for real-time applications from devices connected to the internet will require 100 Gigabit Ethernet (GE) speeds (the standard speed used in current data centers) and even higher speeds, such as 400 GE, to fully utilize the available bandwidth. This requires faster memory and serial bus communication in this data center sector.

PCIe is the preferred choice for standard serial point-to-point high-speed transactions. Besides all the pros and cons mentioned in PCIe chapter two, a new set of features for faster links is part of PCIe Gen5 as follows:

- Equalization Bypass Modes.
- Alternate protocols to negotiate.

- Pre-coding.
- Loopback enhancements.

As the data rate continues increasing, the signal attenuation goes in the same direction, causing a direct impact on all components designed for high-frequency operation. This can lead to various types of distortion, such as inter-symbol interference (ISI) and jitter. All accumulated distortion must be contra-rested via the equalization procedure when this occurs. In a PCIe device operating at 32GT/s, for example, in PCIe Gen3 or Gen4, the equalization setup is needed on both sides (Tx and Rx) to gain signal quality. All these equalization stages remain equal for the newest generation, such as PCIe Gen5 and PCIe Gen6.

Following the steps for bringing up the link to 32GT/s, the first thing needed is to train to L0 at 2.5GT/s and keep going sequentially from 8.0GT/s to 16GT/s and finally to 32GT/s. This method is also known as “Full Equalization.”

Every data rate stage setup is directly associated with latency time consumption, which can increase or reduce the link bring-up time. One equalization stage can consume about ~100ms from vendor to vendor, so transitioning from 2.5GT/s to 32GT/s will take approximately ~300ms.

Two different optional flows during the equalization to decrease latency are:

- Equalization bypass to highest data rates.
- No equalization needed.

A modification on Symbol5 (training control) must be present to achieve one of these two modes. Importantly, both options are mutually supported in EQ and reported to configuration states, ensuring a flexible and compatible system.

Figure 13 shows how depending on the equalization model selected can significantly reduce the bring-up time to 32GT/s.

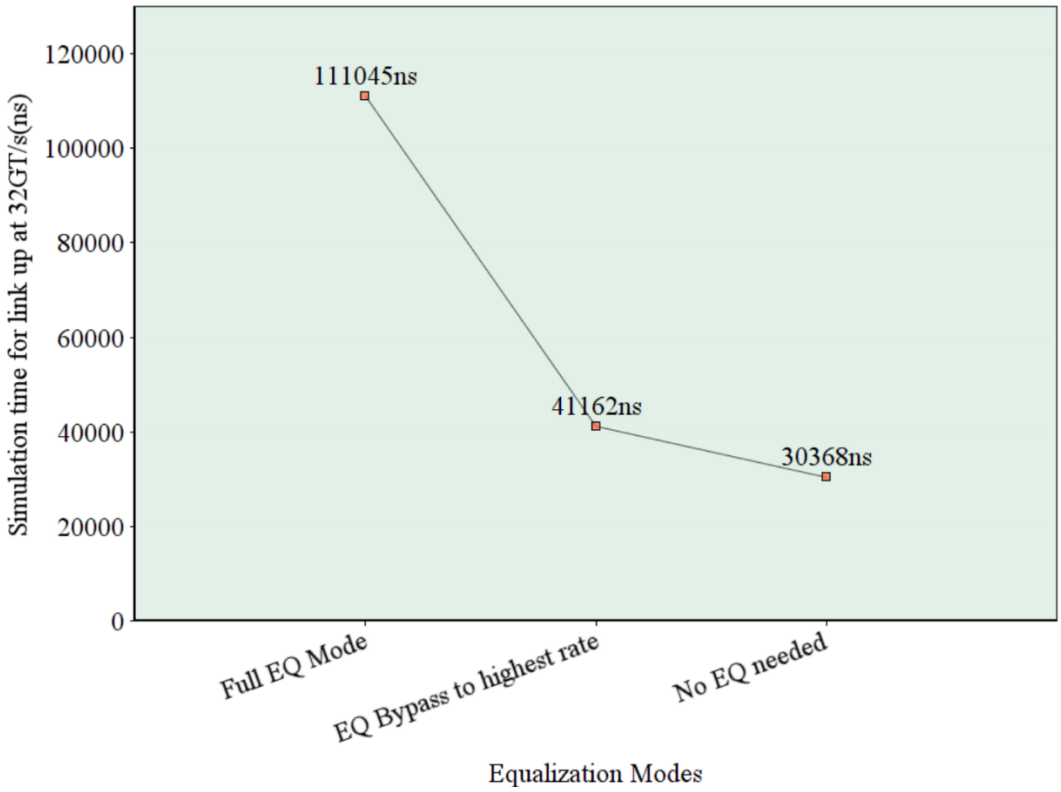


Figure 13 Equalization modes. Figure taken from [*22*]

Also, during link-up, the LTSSM transitions between different equalization modes. By making a good selection of equalization features, the full range of LTSSM and speed transition during link-up are highly configurable.

4.3. Equalization

Compensating distortion within the channel is a process called equalization (EQ). This process is often measured by an eye diagram, a visual representation of the signal quality that ideally resembles an open eye. In this diagram, the high-frequency signal components are substantially attenuated when passing across a band-limited channel, which deforms the original signal and extends to subsequent symbol periods. This effect can reflect a closed eye in the eye diagram.

4. CXL PHY EQUALIZATION MODES

The EQ process aims to generate an adequate opening on the eye diagram and decrease ISI. This allows for easier recovery of transmitted symbols, reducing the BER [*23*].

It is important to note that both the Tx and Rx sides can perform EQ. The Rx equalizer controls noise mixed into the channel, which can derive a lousy signal-to-noise ratio (SNR). This way, the signal tries to be free from noise during EQ on the Tx side [*23*].

4.3.1 Equalization Procedure

The equalization process in PCIe Gen6 stays identical to PCIe Gen3 to PCIe Gen5, with the difference of ordered sets back and forth in each phase. With the introduction of PCIe Gen6, a new ordered set, TS0, works similarly to TS1 and modifies the definition of TS1 and TS2 ordered sets to integrate data replication. Representing TS0 with non-return-to-zero (NRZ) mode uses alternate bits set to 0, which enables its representation using only two voltage levels (0V and 3V) in the four-level pulse amplitude modulation (PAM4) [*23*].

This permits easy detection when the link is not completely trained during the initial stages of EQ. Table 7 shows the different EQ phase usage from TS0 instead of TS1.

TABLE 7

USE OF TS0 VS TS1 IN DIFFERENT PHASES [*23*]

Current Data Rate / Port	Phase 0 / Phase 1	Phase 2	Phase 3
8.0 GT/s, 16.0 GT/s, or 32 GT/s; Downstream Lanes	TS1	TS1	TS1
8.0 GT/s, 16.0 GT/s, or 32 GT/s; Upstream Lanes	TS1	TS1	TS1
64.0 GT/s Downstream Lanes	TS0	TS1	TS1
64.0 GT/s Upstream Lanes	TS0	TS0	TS1

In PCIe Gen 6, the LTSSM has a vital role in the equalization process to guarantee that the link attains the best signal integrity.

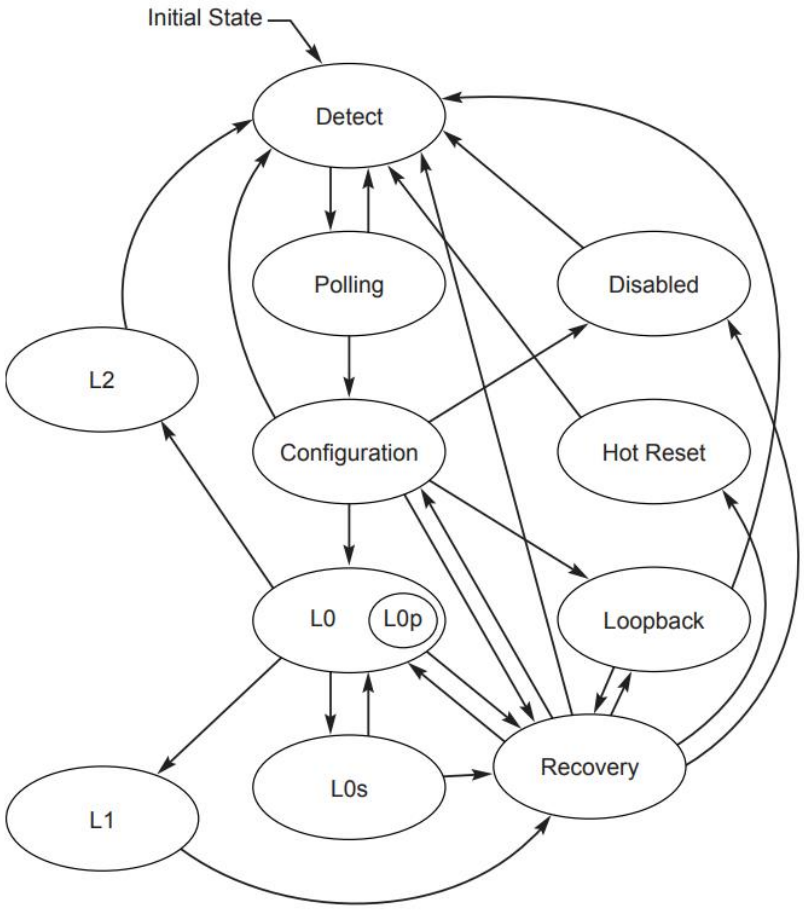


Figure 14 Main State Diagram for Link Training and Status State Machine, Figure taken from [*1*]

The LTSSM moves through various states, such as Polling, Configuration, Recovery, and L0 (Active), each containing sub-states responsible for different aspects of link training and equalization as shown in Figure 14. A brief overview of the main states of LTSSM:

- Polling: The link initializes and detects the presence of a partner device. Basic link parameters, such as speed and lane width, are established.
- Configuration: The link negotiates the data rate, lane count, and other link parameters. During this state, equalization settings may begin to be established to prepare the link for higher data rates.
- Recovery: The link adjusts equalization settings based on real-time signal measurements. This is where most of the equalization tuning occurs, particularly in the "Equalization" sub-state. The link undergoes several phases (e.g., Phase 1, Phase 2, Phase 3) to optimize the

signal. In each phase, different transmitter and receiver settings are adjusted to achieve the best possible signal quality.

- L0 (Active): The link is fully operational, and the equalization settings have been optimized. However, if any issues are detected, the link may transition back to the Recovery state to re-optimize.

The equalization process in PCIe Gen 6 is more complex than in previous generations due to the higher data rates (64 GT/s). The LTSSM ensures that the link maintains integrity by dynamically adjusting the equalization settings as the link conditions change. This process is critical to achieving the high data rates required by PCIe Gen 6 while maintaining signal integrity across various link conditions.

During the EQ procedure, all the lanes related to the LTSSM are active. The EQ process does not need to run between components at any data rate for trusty assignment, but it could run again at certain conditions. For operations at 8.0 GT/s and higher data rates, components are requested to save Tx setups during the EQ procedure. Even when the EQ procedure is complete, components can fine-tune their receiver setup, considering that when applied, that does not cause the link to be unreliable or go to Recovery [*23*].

The path to achieving 64 GT/s of bandwidth is a sequential one, starting with training from 32 GT/s with the link fully functional. This process ensures a smooth transition and optimal performance. Data link-level packets (DLLPs) and transaction-level packets (TLPs) are efficiently commuted in the normal operating L0 state of the LTSSM [*23*].

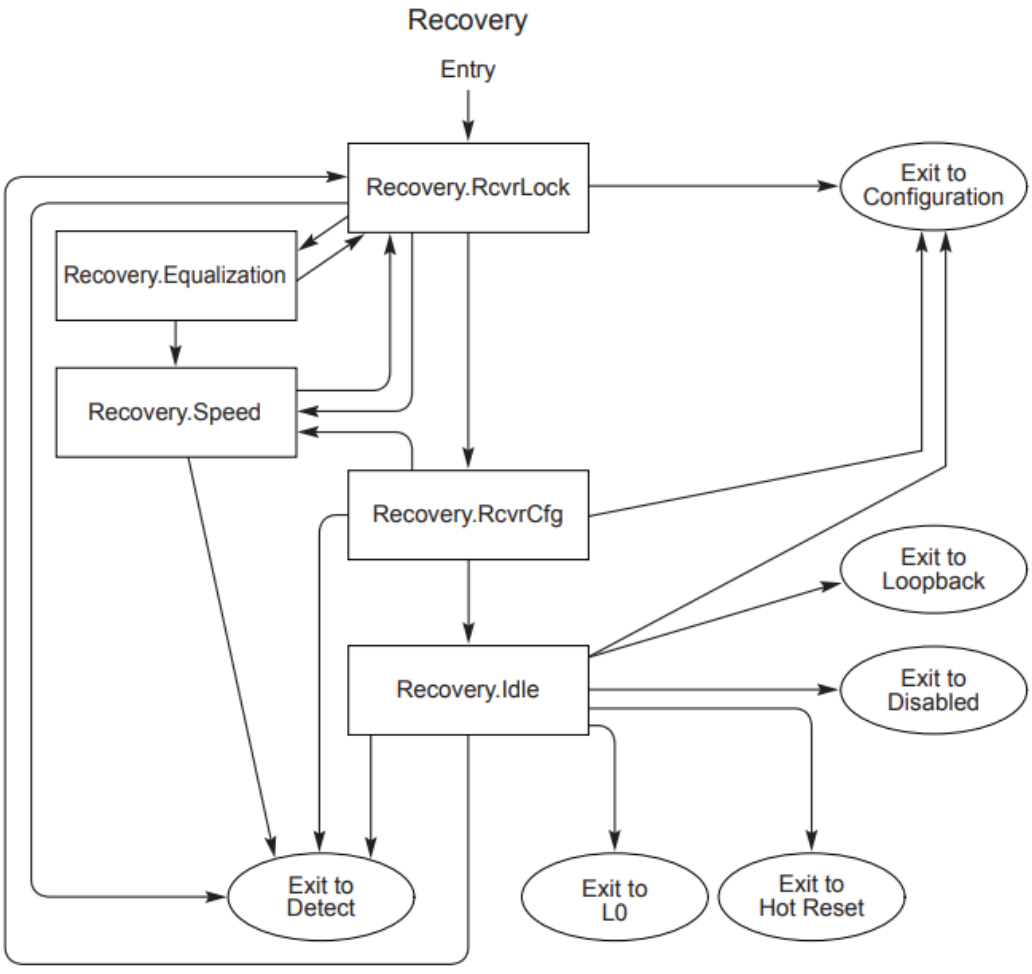


Figure 15 Recovery substate machine. Figure taken from [*1*]

When a device requests equalization, it changes from the Recover.RcvrLock sub-state to the L0 state at 8 GT/s, according to Figure 15. Each device communicates TS1 ordered sets with a speed-change bit set in the Recovery.RcvrLock device [*20*].

– *Step-1*: For upstream and downstream components, the following state is Recovery.RcvrCfg once received eight TS1 OS. The downstream port transmits EQ TS2 with a valid Tx preset and Rx preset hint fields at Recovery.RcvrCfg substate. All preset values come from the upstream port's EQ control register (available in secondary PCIe extended capability). The upstream port transmits regular TS2 OS. Similarly, it saves the preset values admitted during EQ TS2 OS from the downstream port to employ them in EQ Phase 0.

- *Step-2*: The following state reached is Recovery. Speed once has received eight TS2 OS.
- *Step-3*: In the Recovery sub-state, the upstream and the downstream components send EIOS OS and go into electrical idle where both change to new speed (8 GT/s) and then shift to Recovery.RcvrLock sub-state one more time. Recovery.RcvrLock acts as a bypass state for the downstream port, and it jumps directly into Recovery.Equalization Phase 1.
- *Step-4*: The upstream ports send TS1 OS in Recovery.RcvrLock state, and it changes to Recovery. Equalization Phase 0 once has accepted TS1 OS with EQ command bit (Symbol 6, bit 7). When at Recovery.Equalization sub-state, the downstream port moves directly from Phase 1, and the upstream port moves directly from Phase 0.
- *Step-5*: Next in order is that both devices continue to Phase 3 and transition to Recovery.RcvrLock sub-state.
- *Step-6/7/8*: RcvrLock, Recovery.RcvrCfg, Recovery.Idle sub-states jump to L0 in an expected form.

For PCIe Gen6, the Tx coefficient values are ruled by next protocol restrictions,

$$|C_{-2}| + |C_{-1}| + |C_0| + |C_{+1}| = 1 \tag{4-1}$$

where:

$$C_{-2} \geq 0, C_{-1} \leq 0, C_0 \geq 0, C_{+1} \leq 0 \tag{4-2}$$

In Figure 12, only C_{-2}, C_{-1}, and C_{+1} are in charge to define V_{out}, with C_0 implied by (4-2). Subsequent conditions also constrain coefficient range and tolerance. Different presets incorporate the use of coefficients with their respective tolerances defined in the Tx preset ratios table in the PCIe specification. Full swing (4-3) announces the maximum differential voltage that the Tx can produce, which helps maintain the output-transmitted power constant regarding coefficients.

$$FS = |C_{-2}| + |C_{-1}| + |C_0| + |C_{+1}| \tag{4-3}$$

$$|C_0| - |C_{-1}| + |C_{-2}| - |C_{+1}| \geq LF \tag{4-4}$$

The low-frequency parameter (LF) (4-4) indicates that the flat-level voltage must be greater than the minimum differential voltage produced by Tx.

When those constraints are applied, a triangular matrix results as shown in the following trapezoidal tables:

2nd Pre-Cursor C_{-2} = 0/24

PS2 PS1 DE PRESET / BOOST	C_{+1} 0/24	1/24	2/24	3/24	4/24	5/24	6/24	7/24	8/24
0/24	0.0 0.0 0.0 **Q0** 0.0	0.0 0.0 -0.8 0.8	0.0 0.0 -1.6 **Q3** 1.6	0.0 0.0 -2.5 2.5	0.0 0.0 -3.5 **Q4** 3.5	0.0 0.0 -4.7 4.7	0.0 0.0 -6.0 6.0	0.0 0.0 -7.6 7.6	0.0 0.0 -9.5 9.5
1/24	0.0 0.8 0.0 0.8	0.0 0.8 -0.8 1.6	0.0 0.9 -1.7 2.5	0.0 1.0 -2.8 3.5	0.0 1.2 -3.9 4.7	0.0 1.3 -5.3 6.0	0.0 1.6 -6.8 7.6	0.0 1.9 -8.8 9.5	
2/24	0.0 1.6 0.0 **Q1** 1.6	0.0 1.7 -0.9 2.5	0.0 1.9 -1.9 3.5	0.0 2.2 -3.1 4.7	0.0 2.5 -4.4 6.0	0.0 2.9 -6.0 7.6	0.0 3.5 -8.0 9.5		
3/24	0.0 2.5 0.0 2.5	0.0 2.8 -1.0 3.5	0.0 3.1 -2.2 4.7	0.0 3.5 -3.5 6.0	0.0 4.1 -5.1 7.6	0.0 4.9 -7.0 9.5			
4/24	0.0 3.5 0.0 **Q2** 3.5	0.0 3.9 -1.2 4.7	0.0 4.4 -2.5 6.0	0.0 5.1 -4.1 7.6	0.0 6.0 -6.0 9.5				
5/24	0.0 4.7 0.0 4.7	0.0 5.3 -1.3 6.0	0.0 6.0 -2.9 7.6	0.0 7.0 -4.9 9.5					
6/24	0.0 6.0 0.0 6.0	0.0 6.9 -1.6 7.6	0.0 8.0 -3.5 9.5						

C_{-1}

Full Swing Limit or
Max Reduced Swing Limit

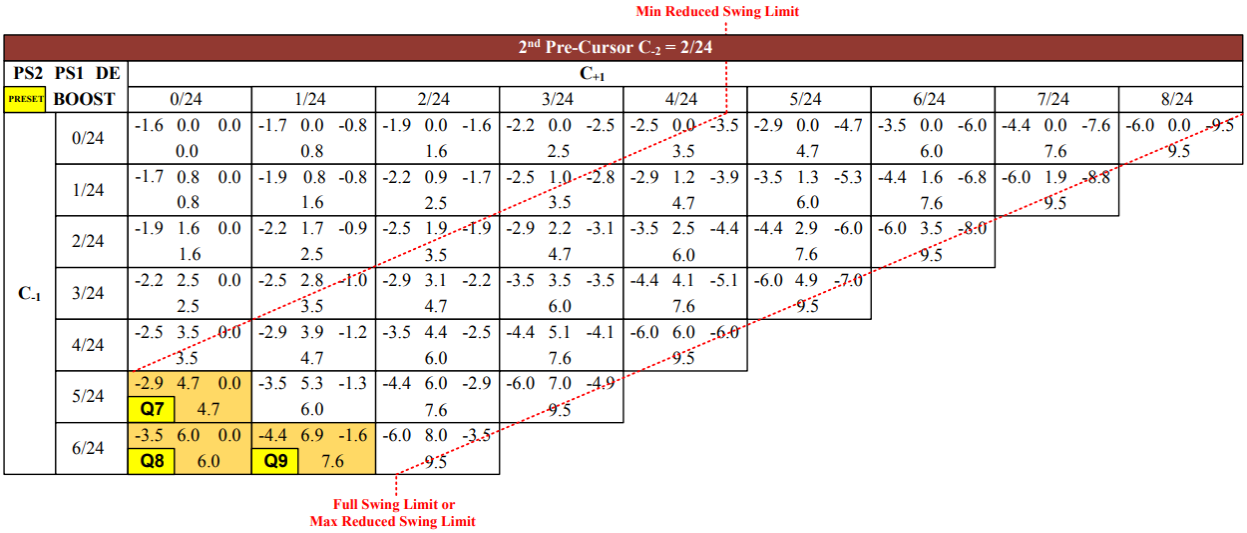
Min Reduced Swing Limit

2nd Pre-Cursor C_{-2} = 1/24

PS2 PS1 DE PRESET / BOOST	C_{+1} 0/24	1/24	2/24	3/24	4/24	5/24	6/24	7/24	8/24
0/24	-0.8 0.0 0.0 0.0	-0.8 0.0 -0.8 0.8	-0.9 0.0 -1.6 1.6	-1.0 0.0 -2.5 2.5	-1.2 0.0 -3.5 3.5	-1.3 0.0 -4.7 4.7	-1.6 0.0 -6.0 6.0	-1.9 0.0 -7.6 7.6	-2.5 0.0 -9.5 9.5
1/24	-0.8 0.8 0.0 0.8	-0.9 0.8 -0.8 1.6	-1.0 0.9 -1.7 2.5	-1.2 1.0 -2.8 3.5	-1.3 1.2 -3.9 4.7	-1.6 1.3 -5.3 6.0	-1.9 1.6 -6.8 7.6	-2.5 1.9 -8.8 9.5	
2/24	-0.9 1.6 0.0 1.6	-1.0 1.7 -0.9 2.5	-1.2 1.9 -1.9 3.5	-1.3 2.2 -3.1 4.7	-1.6 2.5 -4.4 6.0	-1.9 2.9 -6.0 7.6	-2.5 3.5 -8.0 9.5		
3/24	-1.0 2.5 0.0 2.5	-1.2 2.8 -1.0 3.5	-1.3 3.1 -2.2 4.7	-1.6 3.5 -3.5 **Q6** 6.0	-1.9 4.1 -5.1 7.6	-2.5 4.9 -7.0 9.5			
4/24	-1.2 3.5 0.0 3.5	-1.3 3.9 -1.2 4.7	-1.6 4.4 -2.5 6.0	-1.9 5.1 -4.1 7.6	-2.5 6.0 -6.0 9.5				
5/24	-1.3 4.7 0.0 **Q5** 4.7	-1.6 5.3 -1.3 6.0	-1.9 6.0 -2.9 7.6	-2.5 7.0 -4.9 9.5					
6/24	-1.6 6.0 0.0 6.0	-1.9 6.9 -1.6 7.6	-2.5 8.0 -3.5 9.5						

C_{-1}

Full Swing Limit or
Max Reduced Swing Limit



2nd Pre-Cursor C_{-2} = 2/24																										
PS2 PS1 DE		C_{+1}																								
PRESET	BOOST	0/24			1/24			2/24			3/24			4/24			5/24			6/24			7/24			8/24	
C_{-1}	0/24	-1.6	0.0	0.0	-1.7	0.0	-0.8	-1.9	0.0	-1.6	-2.2	0.0	-2.5	-2.5	0.0	-3.5	-2.9	0.0	-4.7	-3.5	0.0	-6.0	-4.4	0.0	-7.6	-6.0	0.0	-9.5
				0.0			0.8			1.6			2.5			3.5			4.7			6.0			7.6			9.5
	1/24	-1.7	0.8	0.0	-1.9	0.8	-0.8	-2.2	0.9	-1.7	-2.5	1.0	-2.8	-2.9	1.2	-3.9	-3.5	1.3	-5.3	-4.4	1.6	-6.8	-6.0	1.9	-8.8			
				0.8			1.6			2.5			3.5			4.7			6.0			7.6			9.5			
	2/24	-1.9	1.6	0.0	-2.2	1.7	-0.9	-2.5	1.9	-1.9	-2.9	2.2	-3.1	-3.5	2.5	-4.4	-4.4	2.9	-6.0	-6.0	3.5	-8.0						
				1.6			2.5			3.5			4.7			6.0			7.6			9.5						
	3/24	-2.2	2.5	0.0	-2.5	2.8	-1.0	-2.9	3.1	-2.2	-3.5	3.5	-3.5	-4.4	4.1	-5.1	-6.0	4.9	-7.0									
				2.5			3.5			4.7			6.0			7.6			9.5									
	4/24	-2.5	3.5	0.0	-2.9	3.9	-1.2	-3.5	4.4	-2.5	-4.4	5.1	-4.1	-6.0	6.0	-6.0												
				3.5			4.7			6.0			7.6			9.5												
	5/24	-2.9	4.7	0.0	-3.5	5.3	-1.3	-4.4	6.0	-2.9	-6.0	7.0	-4.9															
		Q7		4.7			6.0			7.6			9.5															
	6/24	-3.5	6.0	0.0	-4.4	6.9	-1.6	-6.0	8.0	-3.5																		
		Q8		6.0	Q9		7.6			9.5																		

Figure 16 Transmit EQ coefficient space triangular matrix example for 64.0 GT/s. Figures taken from [*1*]

At the post-silicon stage, to obtain the optimum subset of coefficients for Tx and Rx, rest practically by employing EQ maps, as shown in Figure 1. These maps are collected by measuring eye diagram key aspects such as height, width, and asymmetries of the received signal for each of the C_{-2}, C_{-1}, and C_{+1} sequences [*18*].

4.4. CXL Vs PCIe

A major difference between CXL and PCIe is that CXL uses something called an alternate protocol just right after initial link training. This mechanism specified within PCIe permits the link to be trained at a distinct protocol outside the PCIe field. Once an alternate protocol connects between the CPU host and the device, the PCIe specification no longer has jurisdiction.

CXL is even a different protocol than nonvolatile memory express (NVMe) technology over PCIe. NVMe uses PCIe memory to read and write transactions over a standard PCIe link.

Even when the data link is absent, the CXL mode operates thanks to the early negotiation of the CXL at the lower down layer. After the link comes up and the speed changes, CXL.io (the

same as the PCIe data link layer) will be initialized at the first instance, followed by CXL.mem and CXL.cache.

From a data link layer perspective, using the same PCIe physical layer under its protocol avoids some inefficient practices performed by memory and accelerated applications. CXL does not work on PCIe Gen3 nor Gen4 systems since it requires a feature of PCIe Gen5 called the alternate protocol.

Also, the alternate protocol negotiation at the initial link training is not part of the PCIe Gen4 or previous specifications. Nevertheless, CXL can work through PCIe Gen5 from 8GT/s to 16GT/s, or it is possible to go lower speeds thanks to the degraded mode concept, which allows the links to operate at lower data rates and widths. Also, it only supports the links encoding at 128/130b. Whenever possible, CXL tries to run at the higher data bit transfer of 16GT/s or 32GT/s supported by PCIe Gen5 [*24*].

4.5. Equalization for Tx FFE, CTLE, DFE

To reach more than 28GT/s in a CXL link, both parts of a high-speed serial data channel must be equalized. EQ is critical from baseband, two-level NRZ (non-return to zero) to PAM4 modulations when running lanes exceed 50GT/s [*25*].

The aim of the EQ method at the Tx, Rx, or both is to invert a channel's frequency response and use it to remove a big part of the ISI and leave random noise, jitter, DCD (duty-cycle distortion), crosstalk, and EMI [*25*].

The expected process from an ideal equalizer is to invert the channel response and convert the pulse response back into a square pulse.

There are three modes of equalization available for high-speed serial designs [*25*]:

1. Tx FFE (transmitter feed-forward).
2. Rx CTLE (continuous time linear equalization).
3. Rx DFE (decision feedback equalization).

Looking for the transmitted power constant, the amplitude of symbols surrounding transitions into the Tx FFE is modified. In other words, invert ISI when several symbols are changing.

4. CXL PHY EQUALIZATION MODES

Three taps of Tx FFE increase the ratio from high to low-frequency signal components to counter the channel's low-pass nature that, at some point, cannot ultimately contribute to fixing the channel frequency response.

CTLE is situated at the Rx side, which oversees low-frequency signal component attenuation, followed by amplification components all over Nyquist frequency, and filters off higher frequencies, as shown in Figure 17.

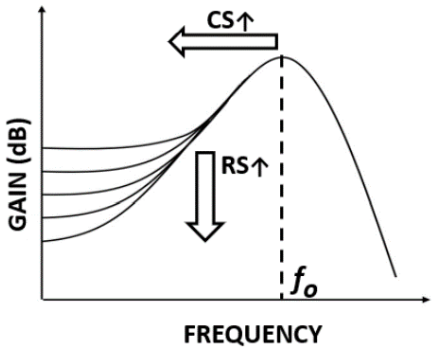


Figure 17 Tunability of CTLE. Figure taken from [*26*]

The main purpose of CTLE is to fine-tune the ratio of low-frequency attenuation to high-frequency amplification, but this does not work when attenuation pushes the low-frequency signal component into the noise domain. CTLE only affects the channel's gross low-pass filtering effect [*25*].

Tx FFE and CTLE can be useless when attacking this issue by inverting the channel's low-pass nature. It is not usual to include both mechanisms for high-speed designs, particularly those using PAM4 signaling.

For Tx FFE, there are tree taps for configuration. On the other hand, ISI-inverting capabilities usually use six-pointers. PAM4 signaling is not related to CTLE technology. Tx FFE does not change any principle; it only has four different symbols. DFE is entirely different for PAM4 trying to select one of the four possible "decisions' into a decision circuit. The high or low frequency can be amplified or attenuated because of the natural behavior of either Tx FFE or CTLE with the number of DFE taps, which is an optimal equalization process over the serial differential signal [*25*].

DFE is assembled into the receiver side as shown in Figure 18:

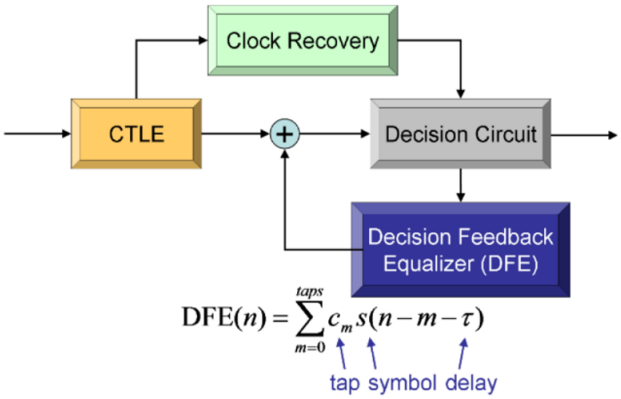


Figure 18 DFE uses a decision circuit as part of its feedback loop. Figure taken from [*25*]

DFE provides a symbol decision back to the symbol decoder implicitly nonlinearly. DFE comes together with Reed-Solomon FEC to create a better resolution.

5. CXL Optimization Low Latency

Finding affordable latency measurement tools that could be embedded in the CXL test card or CXL host was challenging. The CXL logical analyzer was not part of the scope due to its unavailability. Therefore, we created our custom tool as an alternative to overcome this challenge.

Designing scenarios that can add latency to the CXL channel was also challenging. Even after experimenting with adding a PCIe length extensor card (PLEC) between the CXL host and CXL device or error injection, the variation in modulation was less than $160x10^{-9}$ seconds (±10 ticks) and lacked reproducibility.

5.1. Improving Latency by Reducing Link Bring-up Time

The CXL technology used in recent Intel processors loads a PHY recipe containing optimal values for different PCIe EQs based on the platform's requirements at the boot process. Those values are obtained in a controlled environment by analyzing the aperture of the eye diagram using various combinations of parameter values, as discussed in Chapter 4.

The DFE equalizer parameters at the Rx are fixed and cannot be changed dynamically after the boot process. However, the CTLE equalizer is an exception as it can modify its parameters, such as the CTLE index (CTLE gain), capacitance, and resistance, even after the boot process, thanks to firmware that operates in a dedicated microcontroller inside the processor. This microcontroller embedded within the processor manages the CTLE parameters. This integration enables efficient real-time adjustments and management of the CTLE equalizer parameters. The process of updating the CTLE firmware is confidential and is not covered in this thesis. However, some memory-mapped registers for status and control interact with certain CTLE features through firmware.

This thesis proposes reducing latency by developing a mechanism that measures the latency of a non-posted memory read transaction between two CXL devices; the system will dynamically control CTLE parameters under a CXL link by using an optimization technique such as Pattern

Search [*27*], Nelder Mead [*28*], Genetic Algorithm [*29*], or Discrete Optimization (Branch and Bound) [*30*].

5.2. Surrogate Modeling - Gaussian Process Regression

The proposal uses surrogate-base optimization by using the Gaussian process regression (GPR) as the modeling technique. To better understand what GPR means, it can be broken down into some fundamental topics such as Gaussian distribution, multivariate normal distribution (MVN), kernels, non-parametric models, and finally regression.

A) Gaussian distribution: When having a mean μ and variance σ^2 of its probability density function (PDF) of an aleatory variable X, it is said that it is Gaussian distributed. This can be represented by the next formula,

$$P_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\boldsymbol{x}-\mu)^2}{2\sigma^2}\right) \tag{5-1}$$

where, $\boldsymbol{x}$ is the actual argument, and X represents random variables. The normal distribution of X is represented by,

$$P_{\boldsymbol{X}}(\boldsymbol{x}) \sim \mathcal{N}(\mu, \sigma^2) \tag{5-2}$$

B) Multivariate normal distribution (MVN): It is common to implement multiple feature variables (x_1, x_2, ..., x_D) that are correlated between them to describe a system.

For that reason, the multivariate Gaussian distribution model (MVN) is best suited for modeling all variables as a single Gaussian model.

The PDF of a D-dimensional MVN is defined as:

$$\mathcal{N}(\boldsymbol{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)\right) \tag{5-3}$$

where:

- D represents the number of the dimensionality
- x denotes the variable
- $\mu = \mathbb{E}[x] \in \mathbb{R}^{\mathrm{D}}$ is the mean vector

- $\Sigma = \text{cov}[x]$ is the $D \times D$ covariance matrix.

The matrix Σ is symmetric and contains the pairwise covariance of all jointly modeled random variables, with $\Sigma_{ij} = \text{cov}(y_i, y_j)$ as its (i, j) element.

The smoothing process relies on the kernel or covariance function, which is in charge to encapsulate our prior knowledge about the functions that we want to model.

When using regression, the idea of having predictions is that they can be smooth and logical, or similar inputs return similar results.

C) The "kernel function," which can be denoted as *k(x, x')*, helps us to achieve this lift and enhanced compression in the feature space. There is a famous "kernel trick" that facilitates the comparison of data obtained by using the dot product. The kernel function *k(x, x')* works on the similarity between data points in a high-dimensional feature space without explicitly consuming the inputs [*31*].

The kernel function is an alternative and efficient process due to its ability to obtain similar results by computing the dot product of transformed inputs, $\langle \phi(x) | \phi(x') \rangle$ with ϕ being the feature mapping function.

The most common kernel method used for doing Gaussian process is the squared exponential (SE) kernel, also known as the Gaussian or radial basis function (RBF) kernel.

The SE kernel function is defined as:

$$\text{cov}(x_i, x_j) = \exp\left(-\frac{(x_i - x_j)^2}{2}\right) \tag{5-4}$$

– *Non-parametric Model*: Parametric models assume that the data distribution will use a set of finite parameters. The basic idea of using regression is that we can predict the function value y = *f(x)* for a new specific ***x*** based on existing data points. When using parametric models for regressions, the number of parameters inherently constrains the complexity or flexibility of the models. On the other hand, if the model parameter counts increases with the size of the observed dataset, it is classified as a non-parametric model.

– *Gaussian process*: A set of points can be fitted by a probability distribution over possible functions known as a Gaussian process. A model based on Gaussian process regression offers both predictions as well as estimates of uncertainty.

The function of regression that is modeled by a multivariate Gaussian can be expressed as follows:

$$P(\mathbf{f} \mid \mathbf{X}) = \mathcal{N}(\mathbf{f} \mid \mu, \mathbf{K}) \tag{5-5}$$

- where $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n]$ represents the observed data
- points, $\mathbf{f} = [f(x_1), ..., f(x_n)]$ the function values
- $\mu = [m(\mathbf{x}_1), ..., m(\mathbf{x}_n)]$ the mean function
- $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ the kernel function, which is a positive definite.

With no observation, the default mean function is $m(\mathbf{X}) = 0$, assuming the data is zero mean. The Gaussian process model is essentially a distribution over functions. The variable **K** determines the smoothness and shape of these functions.

Kernel functions are used in the model to incorporate prior knowledge about the function nature. The process of regression using Gaussian processes is demonstrated in the Figure 19. In such image, we can see observed data points represented by red dots and a mean function represented by a *f* or blue line, which is estimated from these observed data points [*31*].

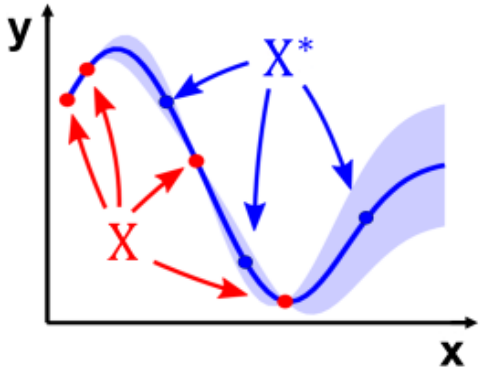


Figure 19 Representation of conducting regressions by Gaussian processes. Figure taken from [*31*]

5.2.1 GPR Modeling CXL Latency Optimization

Figure 20 displays the relationship between GPR simulated values of CTLE combinations (x-axis) and latency measurements (y-axis) in ticks (1 tick equals 16ns). The red line represents the simulated values, and the blue circles represent the latency measurements.

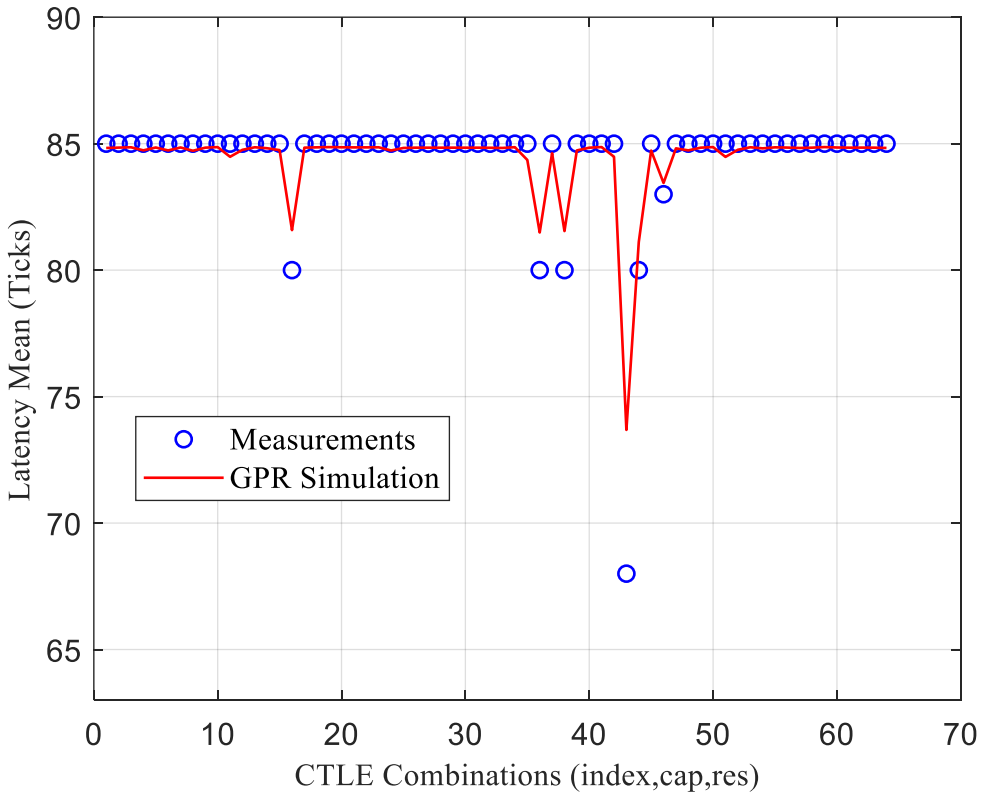


Figure 20 GPR Simulation of CTLE Combinations Vs Latency Measurements

The GPR model shows a strong performance in predicting the outcomes. The low Mean Squared Error (MSE) of 0.68637 using all data contained in Figure 20 indicates that the average difference between the predicted and actual values is small, suggesting high accuracy in the predictions. Additionally, the high Determination Coefficient (R^2) of 0.88103 means that the model explains about 88.1% of the variance in the data, signifying a robust fit. Overall, the model effectively captures the underlying patterns in the data with only a 6.68% percent error.

Analyzing the relationship between latency results using different parameter equalizer configurations can be quite complex, and a surrogate model approach will be used to simplify it. Unlike traditional methods, it offers a cost-effective solution for system response approximation. While an accurate model is desired for surrogate-based optimization (SBO), it often necessitates

numerous lab measurements. This process is time-consuming and financially burdensome, especially under the current time-to-market schedules. Once a surrogate model is available, the user has the power to optimize it according to their specific figure of merit (FOM) using an optimization algorithm such as Nelder-Mead-Pattern-Search (PSNM), Genetic Algorithm (GA) or Branch and Bound (BB).

5.3. CXL Equalization by Surrogate-Based Optimization

5.3.1 Objective Function Formulation and Optimization

The objective is to find the optimal set of CTLE coefficients to minimize the latency. Then, the objective function to be minimized is defined as,

$$u(\boldsymbol{x}) = latency\ measurement\ (\boldsymbol{x}) \tag{5-6}$$

The latency is function of the CTLE coefficient values (index, capacitance, resistance) contained in vector ***x.*** The optimization problem is then defined as [*32*],

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} u(\boldsymbol{x}) \tag{5-7}$$

with $u(\boldsymbol{x})$ defined by (5-7).

5.3.2 Pattern Search

One of the fundamentals of the pattern search method is implementing a set of search directions in an iterative form. The implemented search directions should cover the entire search space.

In different ways, setting two different initial points within the search space at any place can be reached by traversing these search directions only. If an issue is categorized as *N*-dimensional it will require *N* linearly independent search directions at a minimum.

For instance, when moving from any given point to another point within a two-variable function it is crucial to employ two search directions at least. It is possible to have many different

combinations of *N* search directions. Depending on the high or low iteration number needed, some combinations may reach the destination slower or faster, respectively.

An iterative sequence of exploratory moves and heuristic pattern moves is required as part of the Hooke-Jeeves method [*27*]. To identify the most optimal point within the search area an exploratory change is applied in the vicinity of the current point recursively.

Pros and Cons of pattern search technique for CXL latency optimization:

- Pros
  - *Derivative-free*: Suitable for non-smooth, noisy, or discontinuous objective functions, common in complex latency optimization problems.
  - *Flexibility*: Can handle constraints and discrete variables, making it versatile in different scenarios.
  - *Global Search Capability*: Avoids getting trapped in local minima due to its exploratory nature.
- Cons:
  - *Slow Convergence*: May require many iterations, leading to longer optimization times, particularly in high-dimensional spaces.
  - *Inefficiency*: May be inefficient for smooth functions where gradient-based methods would be faster.

5.3.3 Nelder-Mead

John Nelder and his colleague Roger Mead, two English statisticians, invented the Nelder-Mead "simplex" direct search method [*28*] at the National Vegetable Research Station looking to optimize complex nonlinear real-world problems. This method offers a more practical solution that only requires values of function *f* instead of getting the first derivatives.

Within the framework of the Computer Journal, Nelder-Mead presented their paper in 1965 as part of the British Computer Society. At that time, the existing implementations and numerical testing algorithms were equally compared against the Nelder-Mead method, which demonstrated high efficiency and was easy for professionals to understand.

5. CXL Optimization Low Latency

Spendley, Hext, and Himsworth first introduced the "simplex" term for response surface exploration, a method that describes patterns of *n*+1 points in a dimension that move across the exploration surface, often changing in size but always keeping the same shape; the Nelder-Mead simplex can be represented in two dimensions as shown in Figure 21 by only using five different moves within the iteration process.

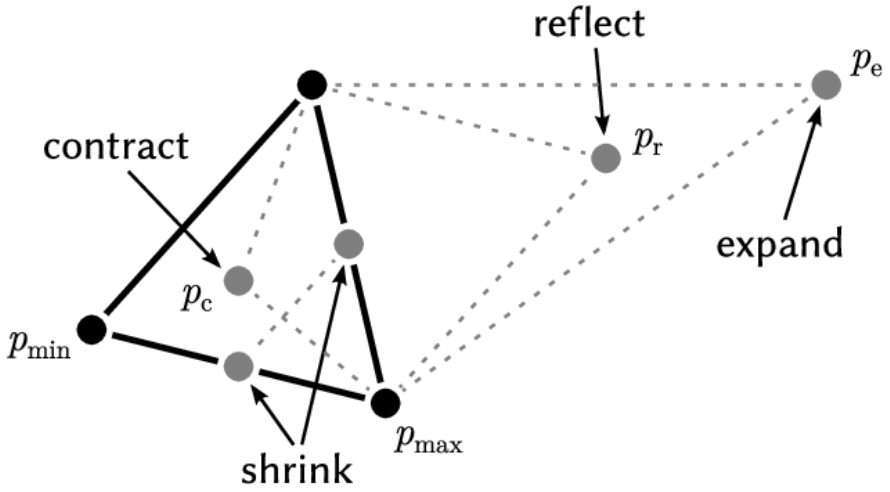


Figure 21 Nelder-Mead Moves. Figure taken from [*33*]

The simplicity of the shrink differs from the other moves in that it replaces the worst vertex of the simplex at iteration *z* with one of the reflection, expansion, or contraction points at iteration z + 1. Therefore, the Nelder-Mead method has been adopted and has become more widely used than other simplex-based methods in the context of unconstrained optimization since the 1970s.

Even when the Nelder-Mead method is more heuristically based and is well known to experience practical difficulties ranging from stagnation to failure, it has proved solid and reliable in practice than other direct search methods.

Pros and Cons of Nelder-Mead technique for CXL latency optimization:

- Pros:
  - *Simple Implementation*: Easy to understand and implement, requiring only function evaluations without derivatives.
  - *Effective for Small Problems*: Works well for lower-dimensional problems where simplicity and robustness are more critical than speed.

- Cons:
  - *Scalability Issues*: Can struggle with high-dimensional problems and may converge slowly.
  - *Local Convergence*: Susceptible to getting trapped in local minima, especially in complex landscapes like CXL latency optimization.

5.3.4 Genetic Algorithm

A search algorithm that utilizes natural selection and genetics is known as a Genetic Algorithm (GA). Another more precise definition of GA from MathWorks: *a method for solving both constrained and unconstrained optimization problems based on a natural selection process that mimics biological evolution. The algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm randomly selects individuals from the current population and uses them as parents to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution* [*34*].

This algorithm connects the survival of the fittest among string structures with a structured but random exchange of information to create a search algorithm that has some unique human-like search capabilities. Bits and pieces of the fittest old generation are utilized to create a new set of artificial creatures or strings, occasionally incorporating a new part to improve the outcome.

John Holland, along with his colleagues and students, developed GA at the University of Michigan looking two main objectives as part of this research: 1) to abstract and explain with so many details the adaptive processes of natural systems and 2) to design artificial systems software that keeps the important mechanisms of natural systems. By this way, important discoveries have been done in both systems: natural and artificial [*29*].

The GA research has focused on robustness across different survival environments by balancing efficiency and efficacy. Nevertheless, achieving higher robustness for artificial systems requires a significant redesign, which may involve reducing or eliminating specific components. When adaptation can achieve high levels of robustness, current systems can execute their functions better and more quickly. For both types of artificial system designers, hardware and software, robustness or efficiency can only be achieved by incorporating the flexibility of biological systems.

5. CXL Optimization Low Latency

The most critical features of biological systems are self-repair, self-guidance, and reproduction, but they do not exist in the most sophisticated artificial systems.

Which brings an interesting conclusion, nature does it better than artificial systems when robust performance is needed where secret is adaptation and survival. GA can theoretically provide a robust search in complex spaces and has been established as a valid approach for business, scientific, and engineering applications; many papers leverage this technique in function optimization and control applications since these algorithms does not represent a high cost in terms computationally speaking.

The process of crossover combines genes from both parents during reproduction, resulting in a mutation that is not properly inherited. In the next generation the offspring reproduce, and the cycle continues. The evolutionary algorithm takes inspiration from this theory, using crossover to generate new solutions based on the population's genetic information. It also integrates mutation to introduce new information or maintain diversity within the population, which helps to prevent premature convergence and create a more generic solution.

Figure 22 shows a flow diagram with the most representative sections of the GA process that are described next [*35*]:

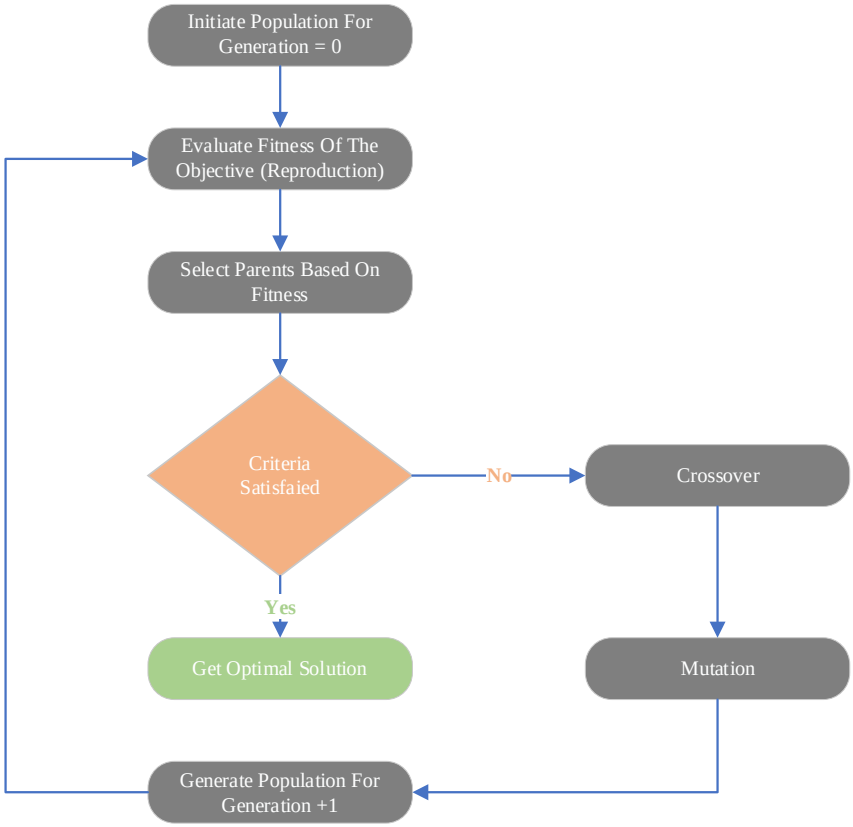


Figure 22 Diagram Flow Genetic Algorithm Process. Figure taken from [*35*]

- *Initiate population:* The first-generation is randomly created and selected by the algorithm (Random Initialization); the size of the population relies on the nature of the issue where decision variable is considering for selecting the size which remains equal across new generations.
- *Evaluate the fitness:* The fitness function or objective function scores how fit each individual candidate solution is for the objective that represents their probability to be selected for reproduction.
- *Select parents:* Identifying the fittest solution within a population so they can reproduce the next level of good solutions capable of maintaining good diversity after each generation is called Parent Selection. Since generation 0 is absent of any offspring, the parent selection is made from the population initiated.
- *Crossover:* This is the step where GA can develop new offspring using genes coming from each parent in a random percentual way. Depending on the weights utilized, parent

chromosomes overlap or mix to produce new generations, which will be reflected in different characteristics.

- *Mutation:* Mutation plays a significant role in GA process as it brings about new genetic traits that keep the population diverse and hinder early convergence. Mutation is associated with exploring the search space, and it is vital for achieving convergence, unlike crossover.
- *Criteria satisfied:* The algorithm stops running when one of these conditions is met:
  - If there is no improvement in the fitness of the population for *X* number of iterations.
  - If the objective function reaches its optimal or predetermined value.
  - If the highest possible number of generations has been completed.

Pros and Cons of Nelder-Mead technique for CXL latency optimization:

- Pros:
  - *Global Optimization*: Efficient in exploring large, complex search spaces, making it suitable for global optimization.
  - *Parallelism*: Easily parallelizable, speeding up the optimization process.
  - *Versatile*: Can handle discrete and continuous variables, constraints, and multiple objectives, fitting diverse CXL latency scenarios.
- Cons:
  - *Computationally Expensive*: Requires a large number of function evaluations, leading to high computational costs.
  - *Premature Convergence*: May converge to suboptimal solutions if not properly managed, especially in difficult optimization landscapes.

5.3.5 Discrete Optimization

The discrete optimization technique is characterized by its complexity, diversity, and ongoing evolution. These problems commonly involve combinatorial calculations and are more computationally expensive than continuous optimization problems.

The research community specializing in discrete optimization primarily focuses on operations research, which typically involves linear models. On the other hand, engineering optimization deals mostly with nonlinear relations, and design variables are essential in engineering optimization from a practical standpoint.

The selection of aspects favoring optimization (diameters, lengths, washers, valves, men, components, stock, sizes, etc.) is usually based on minimizing procurement costs for non-standard components unless the price can be recovered due to large volumes. In cases where a one-of-a-kind design is being created or if the item is to be manufactured entirely in-house where no off-the-shelf component is required, continuous optimization can be justified.

Designing practical engineering solutions involves working with a set of ordered values known as discrete variables, which makes this a discrete optimization problem. Nevertheless, engineering problems currently solved as discrete optimization problems are complex to apply and time-consuming, and most discrete algorithms and code address linear mathematical models. Adaptations of current decision-making community techniques are necessary in engineering for discrete optimization, yet such adaptations are not expected. Despite its vast extent, the subject of nonlinear discrete optimization is rarely addressed or included in any design optimization book, where most of the literature attempts to familiarize the reader with this subject area. Similar to rounding a number to get an integer, a commonly used approach to incorporate discrete design variables in engineering is to solve the corresponding continuous optimization problem and then adjust the optimal design to match the closest discrete values [*36*].

The rounding process may frequently produce impractical designs where a solution is only sometimes possible. A methodical approach to this rounding procedure has become more widely accepted when needed.

When the design variables are not continuous, discrete optimization problems (DP) are involved. If the side constraints concept is extended, these design variables must be chosen from an ordered set of values. Each design variable has its own unique set with no constraints on the spacing of values within the set. If the set is composed of integers only, the problem is referred to as an integer optimization problem. Such a problem is identified as integer programming (IP) if the mathematical model is linear.

5. CXL Optimization Low Latency

IP problems require design variables that can handle values of either "0" or "1" (also known as Zero-One IP problems), but they need to be transformed into Zero-One (0-1) first. This transformation can be done by representing each integer in binary form and assigning a design variable to each binary location.

For example, the eight integer decimal value equals 1000 in binary. As a result, if the design variable $\boldsymbol{x}$ can be any integer between 0 and 16 inclusive, it can be represented using five design variables (y_1, y_2, y_3, y_4, and y_5) that can only take the values 0 or 1. By using the following equation, we can calculate the value of x from the values of the design variables,

$$x = y_1(2^4) + y_2(2^3) + y_3(2^2) + y_4(2^1) + y_5(2^0) \tag{5-8}$$

It is not recommended to perform such a conversion when dealing with many integer variables. Engineering design issues typically involve continuous and discrete variables, and when the mathematical model is linear, they are known as mixed-integer programming (MIP) problems. Next, three different discrete optimization techniques will be described:

- *Exhaustive Enumeration*: This technique involves identifying solutions for all possible combinations of discrete variables. Previously, the methods used needed to be more sophisticated and involved evaluating functions at many points. However, modern computer resources can considerably improve these methods in a peer-to-peer computing environment.
- *Branch and Bound*: Also known as partial or selective enumeration. The fundamental approach utilizes partial enumeration, restricting the exploration to only a portion of the combinations on which the remaining combinations do not contribute to finding the solution. This technique is currently present in most engineering designs for solving discrete optimization problems.
- *Dynamic Programming*: This method is an elegant approach to optimization problems. However, it was less popular because it required significantly more computational resources than other methods, even for reasonably sized problems. Dynamic Programming is particularly well-suited to problems requiring a sequential selection of design variables. However, given the powerful PCs available today, the resource limitations that previously hindered the method are disappearing.

  Pros and Cons of discrete optimization technique for CXL latency optimization:

- Pros:
  - *Exact Solutions*: Provides precise solutions for problems with discrete variables, which are common in CXL latency scenarios.
  - *Efficient for Structured Problems*: Particularly effective when the problem has a clear, discrete structure.
- Cons:
  - *Complexity*: Can be computationally challenging for large, complex problems due to the combinatorial explosion of possibilities.
  - *Limited to Discrete Variables*: Not applicable to continuous optimization problems without discretization, which may reduce accuracy.

The following section will describe in more detail the Branch and Bound discrete optimization technique that is part of the analysis of this research.

5.3.6 Branch and Bound Discrete Optimization

It is important to note that complete enumeration is only feasible in a restricted set of variables because the significant amount of work required to scrutinize all potential solutions rises exponentially.

Solving discrete optimization problems can be made by using partial, incomplete, or selective enumeration. This approach involves making decisions about group solutions rather than examining every single one in handling significant problems.

The Branch and Bound (BB) algorithm is truly unique in its approach. It eschews the original complete discrete mathematical model in favor of relaxation models. Additionally, the BB algorithm employs a tree structure to represent the completion of partial solutions, with nodes and edge links serving as a trail of these solutions [*36*].

The BB algorithm steps, and terminology are described in next.

- *Partial solution*: The first step of the BB process is to identify the nodes of the tree, where some specific design variables are set while others are not yet determined. To illustrate, consider the optimization problem expressed in the following equation,

$$\boldsymbol{x} = [f, 0.5, f] \tag{5-9}$$

where x_2 is fixed at 0.5, and x_1 and x_3 are yet to be determined and represented by the symbol '*f*.' A practical and proven approach to solving these undetermined variables is employing a continuous relaxation of the model, an essential step in the optimization process.

- *Completions (of partial solution)*: The second step of the BB process is the branching or expansion of a node, which occurs when each partial solution or node can produce additional partial solutions/nodes directly under it. In this new set of nodes, one of the free discrete variables is in fix mode while the rest remain free. The fixed node plays a crucial role in maintaining the stability and direction of the solution space. For instance, if we consider the partial solution in Equation (5-9), where the second element remains as the fixed node, a new set of five additional solutions varying the third element in the array are present in Equation (5-10).

$$\begin{aligned} \boldsymbol{x} &= [f, 0.5, 0.22] \\ \boldsymbol{x} &= [f, 0.5, 0.75] \\ \boldsymbol{x} &= [f, 0.5, 1.73] \\ \boldsymbol{x} &= [f, 0.5, 2.24] \\ \boldsymbol{x} &= [f, 0.5, 2.78] \end{aligned} \tag{5-10}$$

- These completions or branches can create a treelike structure with hierarchically structured nodes. Each tier in the structure holds a variable at its prescribed discrete value, and each branch leads to one of these tiers.
- *Fathoming the partial solution*: This step is recurrent within the BB process. Fathoming a node involves expanding it until it reaches its end. This process allows for identifying nodes that are not good candidates and selecting the good ones that can be completed or branched. The tree is pruned or terminated to eliminate any partial solutions/nodes that will not be active in the future during the fathoming operation.
- *Active partial solution*: This is not a step, but a concept involved in the BB process. A node identified as not a good candidate in the solution space acts as a pruned node in the hierarchical tree.

– *Edges/Links*: This is not a step, but a concept involved in the BB process. Node-to-node connections in a tree are known as edges. These edges establish the value of the fixed variable.
– *Root*: This is not a step, but a concept involved in the BB process. The BB algorithm explores a tree-like structure. The exploration begins at the root of the tree, where all the design variables are free, and terminates when all possible sub-solutions within the tree have been explored or stopped. The search is first completed within the same node level before proceeding to other deeper level nodes.
– *Incumbent solution*: Ideally, the BB algorithm should terminate its search for optimum values at the incumbent solution. This means it will find the best solution when all discrete variables use discrete values, making it globally optimum.

Figure 23 and pseudo-code at (*1*) illustrate how the BB algorithm flow works:

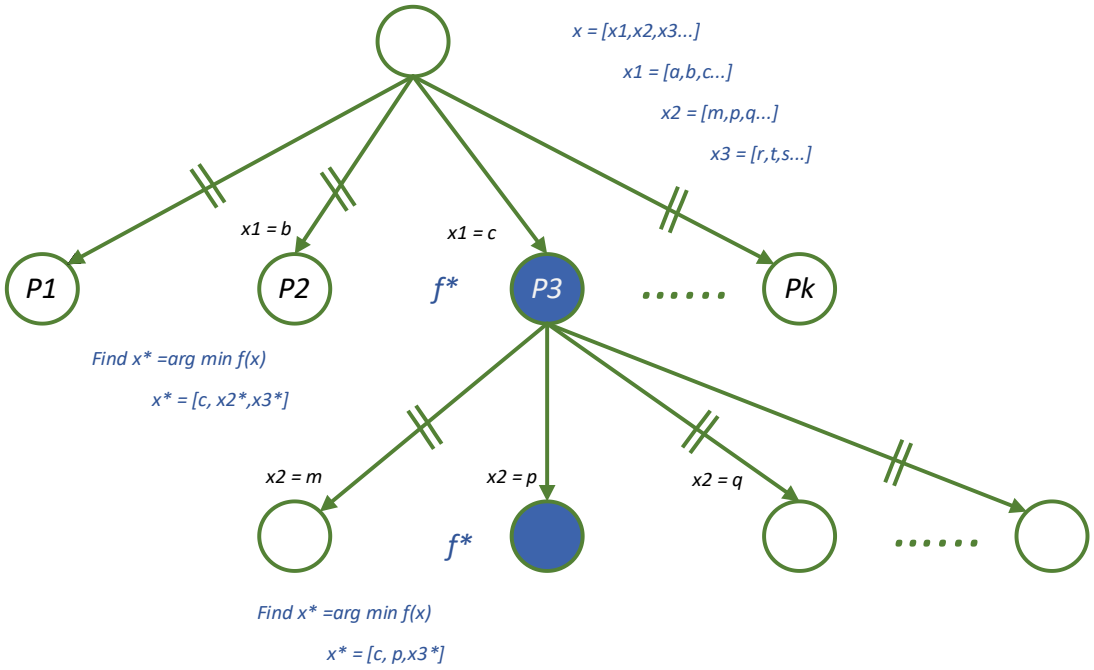


Figure 23 BB Algorithm

Pros and Cons of branch and bound discrete optimization technique for CXL latency optimization:

– Pros:
  - Guaranteed Optimality: Provides exact solutions by systematically exploring all

possibilities while pruning suboptimal branches.

- Effective for Structured Problems: Suitable for problems with a clear structure, allowing for efficient pruning.

– Cons:

- Computational Intensity: Can become computationally expensive, particularly for large-scale problems with many variables.
- Not Suitable for Continuous Problems: Primarily designed for discrete optimization, limiting its applicability in mixed-variable scenarios.

5.4. Optimization Results

In this section, the results of the three different optimization methods are presented individually, along with a comparison of their execution time.

It is important to mention that the optimization space results have limitations due to a restricted range of CTLE index parameters. The anticipated range of operation for the CTLE index parameter values fall between 0 and 15. However, due to system limitations in the Lab, the experiments were conducted using a subset range of CTLE index parameters which includes only 0 and 1. The CTLE capacitance and CTLE resistance parameters have no restrictions and use values between 0 to 3 and 0 to 7, respectively.

The sensitivity of the optimization results tightly depends on the TxEQ and RxEQ parameter values that are automatically adjusted at the system boot process and are expected to remain unchanged while the system is operational. However, different types of CXL configurations can directly impact the optimization results, including:

– The bandwidth and performance characteristics of the memory attached directly to the CXL device.
– The CXL interleave memory configuration port where the device is connected.
– The number of CXL devices connected to the system and the CXL QoS policies.
– The CXL protocol implemented by CXL devices, such as CXL 1.1, CXL 2.0, or CXL 3.0, that offers potential for latency optimization feature in newer versions.

5.4.1 Pattern-Search & Nelder-Mead

As a first optimization approach, a combination of pattern search method [*27*] and the Nelder-Mead method [*28*] was used. *The optimization starts with pattern search, which serves for exploring the design space until finding a potential region where the global minimum is located. Then, the solution found by pattern search is used as seed for the Nelder-Mead method, which further minimizes the objective function for a more precise solution* [*18*]. The pattern search algorithm was implemented through MATLAB using the functions patternsearch [*37*] for pattern search and fminsearch [*38*] for Nelder-Mead.

The Pattern Search / Nealder-Mead (PSNM) combination algorithms in MATLAB produced an output console (5-11) showing a global minimum of 73.6 ticks using a starting seed of CTLE index set to zero, capacitance set to zero, and resistance set to six. The optimal values for CTLE index, capacitance, and resistance are all shown in the console (5-11). The CTLE index is equal to one, the CTLE capacitance is also equal to one, and the CTLE resistance is equal to two.

```
opt_pattern_search >>> Best Objective Function :   73.6575
opt_pattern_search >>> Design Optimal Variables :    1    1    2
```
(5-11)

In Figure 24, the resolution for the optimal CTLE parameters stabilizes after 465 evaluations of the objective function. A median execution time of 100 iterations of PSNM optimization results in 0.096 seconds.

5. CXL Optimization Low Latency

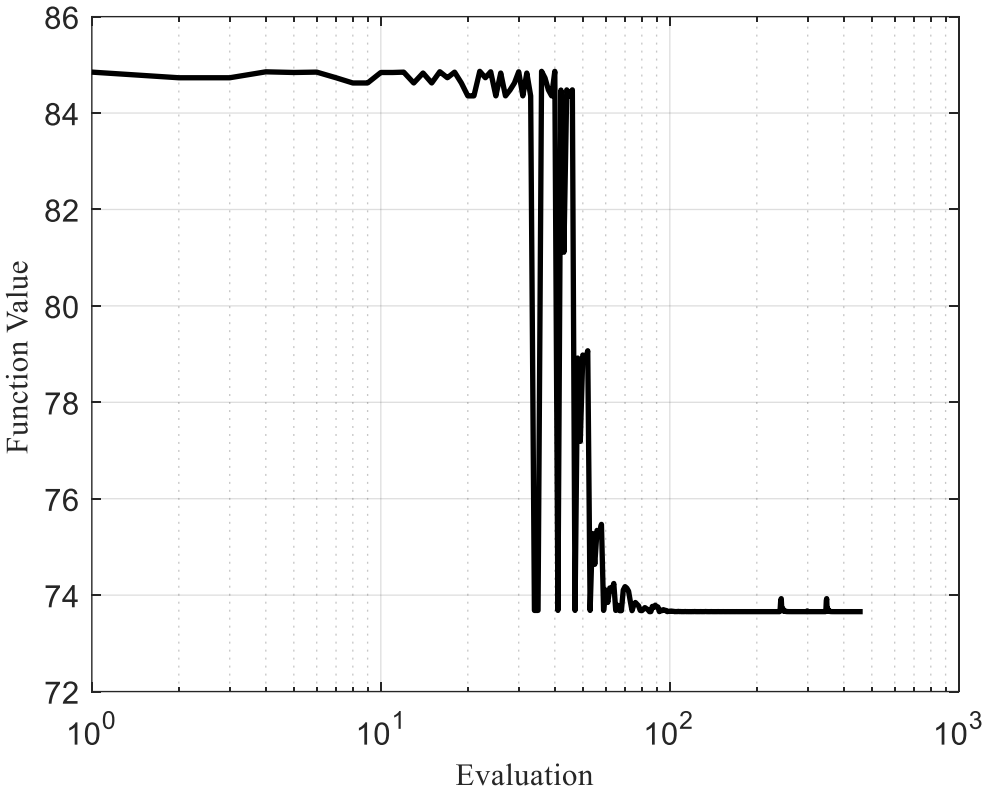


Figure 24 PSNM: CXL latency objective function values across iterations.

Figure 25 displays the process of the PSNM algorithm, which seeks optimal latency optimization values. The three CTLE parameters become stable after about 100 evaluation iterations.

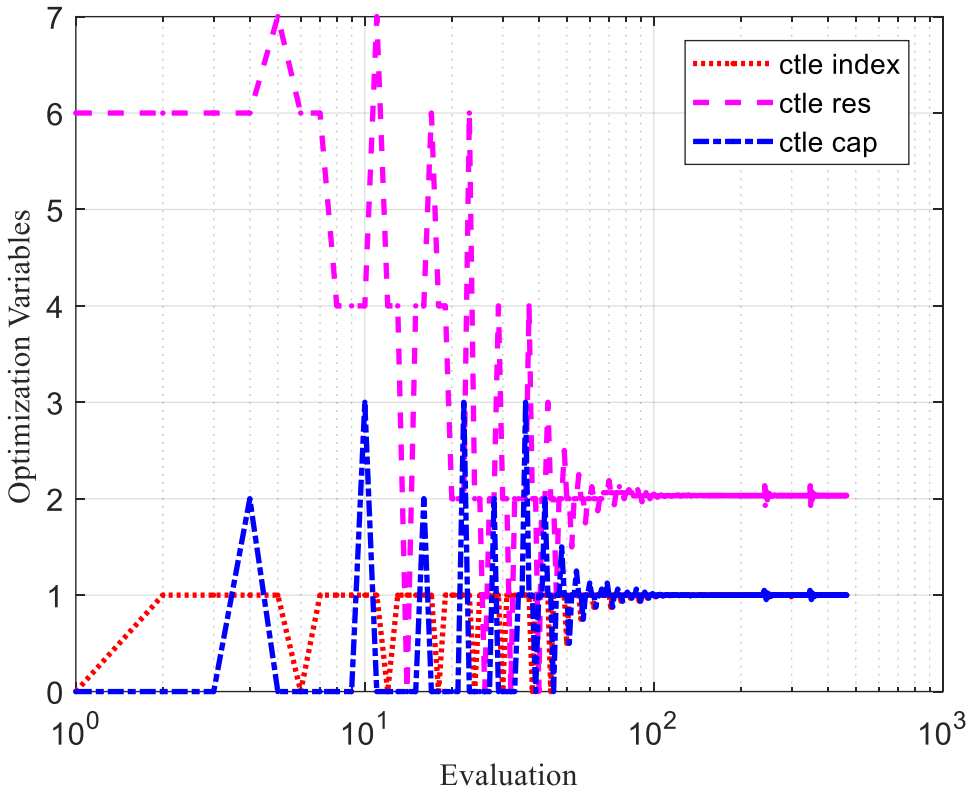


Figure 25 CTLE parameters optimization using PSNM Algorithm

5.4.2 Genetic Algorithm

The GA was implemented through MATLAB using the function ga [*39*]. The GA algorithm in MATLAB produced an output console (5-12) showing a global minimum of 73.6 ticks. The optimal values for CTLE index, capacitance, and resistance are all shown in the console (5-12). The CTLE index is equal to one, the CTLE capacitance is also equal to one, and the CTLE resistance is equal to two.

```
opt_genetic_algorithm >>> Best Objective Function :   73.6575
opt_genetic_algorithm >>> Design Optimal Variables :    1    1    2
```
(5-12)

In Figure 26, the resolution for the optimal CTLE parameters stabilizes after 4007 evaluations of the objective function. A median execution time of 100 iterations of genetic algorithm optimization results in 0.668 seconds.

5. CXL Optimization Low Latency

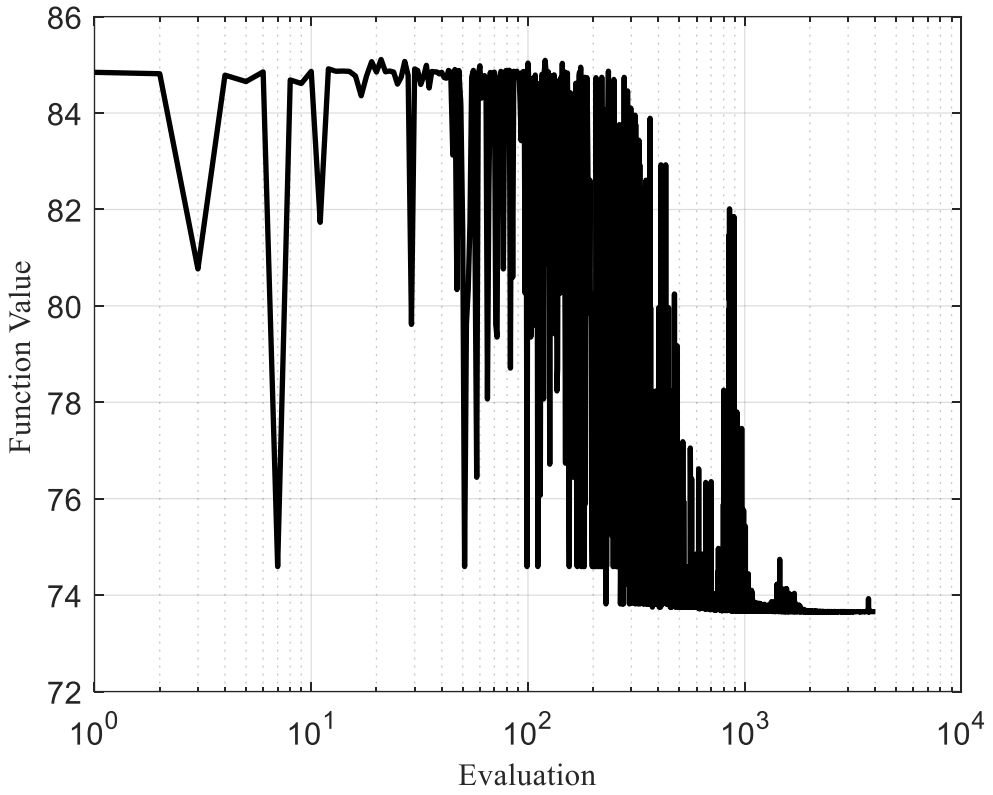


Figure 26 Genetic Algorithm: CXL latency objective function values across iterations.

Figure 27 displays the process of the GA, which seeks optimal latency optimization values. The three CTLE parameters become stable after about 1000 evaluation iterations.

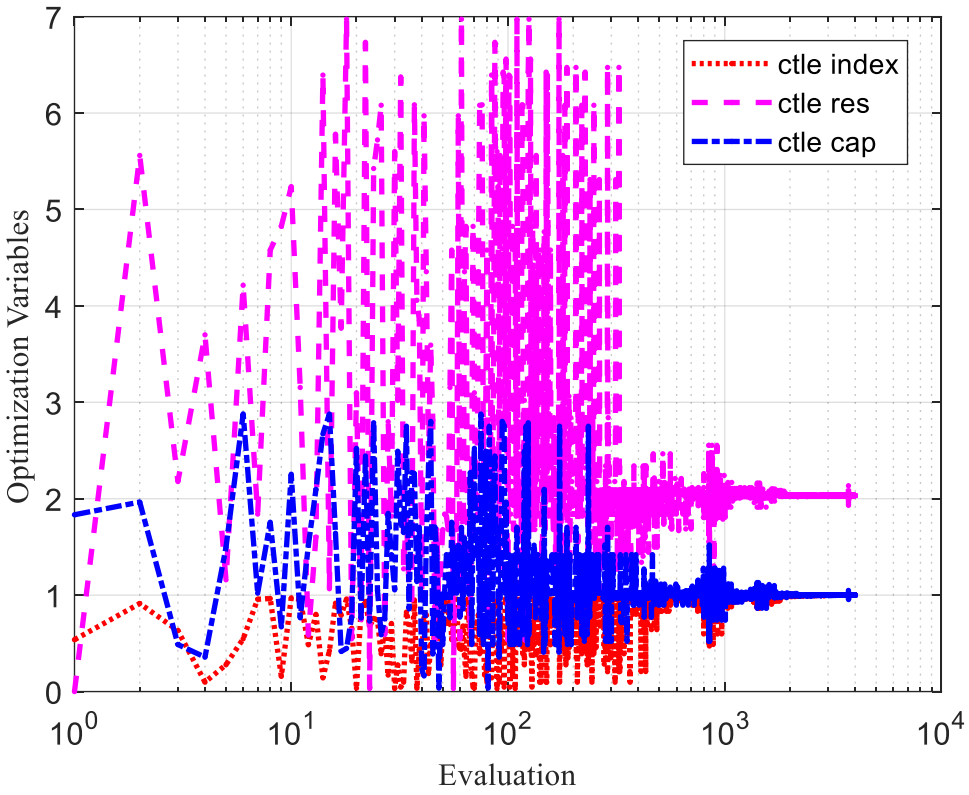


Figure 27 CTLE parameters optimization using GA Algorithm

5.4.3 Branch and Bound

The original BB algorithm [*36*] was modified to avoid using gradients, and replaced with Nelder-Mead. BB algorithm was implemented through MATLAB using the function predict [*40*] and fminsearch [36].

The BB algorithm in MATLAB produced an output console (5-13) showing a global minimum of 73.6 ticks. The optimal values for the CTLE index, capacitance, and resistance are all shown in the console (5-13). The CTLE index is equal to one, the CTLE capacitance is equal to one, and the CTLE resistance is equal to two.

```
bb opt_discrete >>> Best Objective Function :   73.6830
bb opt_discrete >>> Design Optimal Variables :    1    1    2
```
(5-13)

In Figure 28, the resolution for the optimal CTLE parameters stabilizes after 9 evaluations of the objective function. A median execution time of 100 iterations of BB algorithm optimization results in 0.073 seconds.

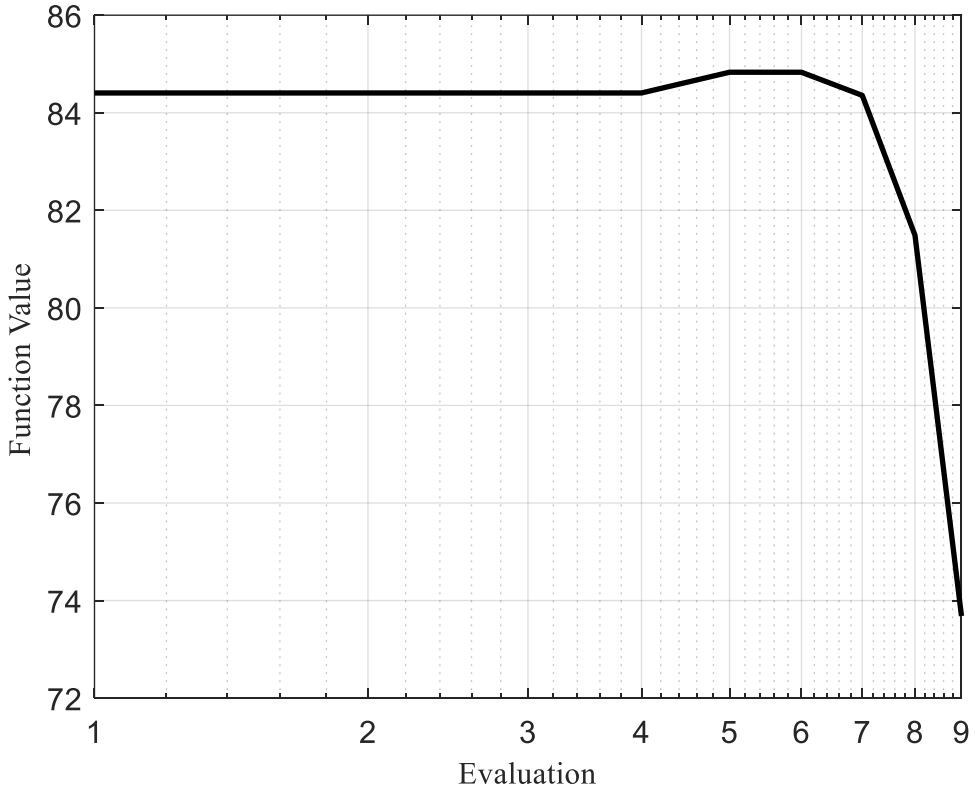


Figure 28 BB Discrete Optimization: CXL latency objective function values across iterations.

5.4.4 Conclusions

The optimization techniques presented in this thesis aim to achieve the same optimization results. However, the primary difference lies in their respective execution times. This factor was crucial in determining the most suitable modeling option, especially for use in the post-silicon stage. It is important to note that the focus of this work was not to gain a deep understanding of each optimization algorithm modeling, but instead to rely on the functions provided by the MATLAB tool when faced with challenges in implementing these methods.

The implementation of the BB algorithm is faster than PSNM and Genetic Algorithm, as demonstrated in the output of the MATLAB script. This is based on the CTLE parameters represented at (5-14):

```
ctle_idx_vec = [0 1];              % CTLE index coefficient
cap_vec = [0 1 2 3];               % CTLE coefficient parameter Cap
res_vec = [0 1 2 3 4 5 6 7];       % CTLE coefficient parameter Res
Execution Time delta_time_opt_pattern_search_nelder_mead 0.096 seconds
Execution Time delta_time_opt_genetic_algorithm 0.668 seconds
Execution Time delta_time_opt_discrete_bb 0.073 seconds
```
(5-14)

When adding interval values, the output of the Matlab script in (5-15) showed a slowness of 2.87 times compared to (5-14) when increasing the dataset size of the CTLE capacitance and CTLE resistance as shown below:

```
ctle_idx_vec = [0 1];                                  % CTLE index coefficient
cap_vec = [0 0.5 1 1.5 2 2.5 3];                       % CTLE coefficient parameter Cap
res_vec = [0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7];   % CTLE coefficient parameter Res
Execution Time delta_time_opt_pattern_search_nelder_mead 0.127 seconds
Execution Time delta_time_opt_genetic_algorithm 0.980 seconds
Execution Time delta_time_opt_discrete_bb 0.210 seconds
```
(5-15)

In TABLE 8 shows an execution time comparison between three different optimization algorithms using different input vector sizes. The increment for PSNM and GA is similar, around 1.4 times, but BB is augmented 2.8 times.

TABLE 8

COMPARISON EXECUTION TIME - OPTIMIZATION METHODS

	Execution Time Normal Cap & Res Vectors (sec)	Execution Time Extended Cap & Res Vectors (sec)	Gain Between Normal & Extended Vectors
Pattern-Search-Nelder Mead	0.096	0.127	1.32
Genetic Algorithm	0.668	0.980	1.46
BB Optimization Discrete	0.073	0.210	2.87

6. General Conclusions

CXL was developed to meet the changing requirements of modern computing systems, offering a high-performance, scalable, and efficient interconnect solution for data-centric workloads across various application domains. One of the primary motivations for creating CXL is its higher bandwidth and lower latency compared to traditional PCIe interfaces. This is critical for managing large volumes of data and complex computational tasks with minimal delays.

This thesis discusses optimizing CXL latency across five chapters.

Chapter 1 provided a brief overview of PCIe. Upcoming PCIe implementations, such as PCIe Gen6 is now a reality, doubling speed and reducing latency. This improvement is valuable for new AI and ML computations.

Chapter 2 described the importance of the LTSSM initializing a CXL link. It plays a crucial role in determining the latency of the link. The chapter also introduced a new feature called flit mode, which is included in PCIe Gen6 and inherited by CXL 3.0. The flit mode feature is focused on reducing latency.

Chapter 3 explored the concept of latency and its impact on system performance, both at the hardware and software levels. It also discussed the pros and cons of bypassing the EQ process and not equalizing. Additionally, a subchapter delved into re-timers, which are more complex devices than re-drivers since they come at a higher cost and have fewer vendors offering this latency solution.

Chapter 4 described the parameters of the three main equalizers used in the EQ process: FIR filter in the Tx, and CTLE and DFE in the Rx. The bypassing or excluding EQ can speed up CXL link bring-up by up to 300ms, but this requires good pre-characterization and optimization of Tx and Rx EQ. However, it is worth noting that CTLE EQ is firmware dynamic.

In *Chapter 5*, the results of CXL latency optimization through EQ obtained from three optimization algorithms were compared, PSNM, GA, and BB Discrete Optimization. Our analysis found that all three optimization methods could find and resolve the global minimum, but each had varying levels of reliability. For example, the BB algorithm outperformed the other two

algorithms when dealing with small discrete vectors. However, GA was more efficient for larger vectors in terms of execution speed and overall performance. It is worth noting that since the PSNM algorithm requires a starting seed, it does not guarantee a global minimum solution within the search space. Therefore, it should be used carefully based on user judgment.

It was challenging to determine the global minimum when evaluating the objective function using the PSNM algorithm. This was due to the nature of starting seed, which made it difficult to identify. The chance of finding a global minimum decreased to 55% when using a non-random starting seed. To address this issue, a 3D plot representation of the CTLE parameter combinations and latency results was created. This allowed us to clearly distinguish between the local and global minimum.

Future work includes exploring possible combinations of TxEQ, RxEQ, and CTLE EQ parameters to minimize latency and create simulation models for different equalizers. Also, a new set of CXL configurations offers new latency optimization features, which are available on the latest versions of the CXL protocol.

Appendix

A. MATLAB Code: Optimization Comparison

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Student: Adan Baltazar
% Date: April 23, 2024
% Department of Electronics, Systems, and Informatics
% University: ITESO
% Thesis: CXL Latency Optimization
% Description:
%               This script compares Pattern Search Nelder Mead, Genetic Algorithm, and
%               Branch & Bound latency optimization algorithm.
%               Comment Test1 or Test2 for desired analysis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   MANAGEMENT FUNCTIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear  % clear all variable/information in the workspace
clear global  %  again use caution - clears global information
clc    % position the cursor at the top of the screen
format compact  % avoid skipping a line when writing to the command window
warning off  % don't report any warnings like divide by zero etc.

global x1v; %#ok<NUSED,GVMIS>
global x2v;  %#ok<NUSED,GVMIS>
global x3v; %#ok<NUSED,GVMIS>
global iterator_j;  %#ok<NUSED,GVMIS>
global fsol;  %#ok<NUSED,GVMIS>
global f2sol;  %#ok<NUSED,GVMIS>
global f3sol;  %#ok<NUSED,GVMIS>
global noprune;  %#ok<NUSED,GVMIS>
global noprune3;  %#ok<NUSED,GVMIS>
global x1sol;  %#ok<NUSED,GVMIS>
global x2sol;  %#ok<NUSED,GVMIS>
global x3sol;  %#ok<NUSED,GVMIS>
global U_1;  %#ok<NUSED,GVMIS>
global U_2;  %#ok<NUSED,GVMIS>
global iteration_cluster_count;  %#ok<NUSED,GVMIS>
global function_cluster; %#ok<NUSED,GVMIS>
global function_cluster_without_zeros; %#ok<NUSED,GVMIS>
global iterator_cluster_without_zeros; %#ok<GVMIS,NUSED>

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   READING RAW DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reading DoE data CTLE parameters for modeling:
ctle_parameters_csv = csvread('ctle_parameters.csv',1,0); %#ok<CSVRD>

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   ASSIGNING CTLE PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ctle_index_modeling_data =  ctle_parameters_csv(:,1);
ctle_cap_modeling_data = ctle_parameters_csv(:,2);
ctle_res_modeling_data = ctle_parameters_csv(:,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   MEDIAN LATENCY MEASUREMENTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
median_latency_s1p3_csv = csvread('median_latency_s1p3.csv',1,0); %#ok<CSVRD>
median_latency_ticks_s1p3 =  median_latency_s1p3_csv(:,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   MODELING
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
latency_cluster_inputs = [ctle_index_modeling_data,ctle_cap_modeling_data,ctle_res_modeling_data];
latency_cluster_outputs = median_latency_ticks_s1p3;  % CXL Mean Latency

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    GPR MODELING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gpr_model_latency_cluster = fitrgp(latency_cluster_inputs,latency_cluster_outputs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    SIMULATE THE GPR MODELS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
latency_sim_cluster = predict(gpr_model_latency_cluster,latency_cluster_inputs);
latency_measurements_cluster = latency_cluster_outputs;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    COMPUTE TRAINING ERROR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
latency_dif_cluster = gsubtract(latency_sim_cluster,latency_measurements_cluster);
latency_error_cluster = norm(latency_dif_cluster,inf)/norm(latency_measurements_cluster,inf)*100;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    PRINT & PLOT RESULTS (Training)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp(['Modeling Error Latency_Cluster1 = ' num2str(latency_error_cluster) ' %']);
%
figure
size_inputs=length(latency_cluster_inputs(:,1));
test_case=1:size_inputs;
% plot(test_case,latency_sim_cluster,'k-o',test_case,latency_measurements_cluster,'b--*');
plot(test_case,latency_measurements_cluster,'B o',test_case,latency_sim_cluster,'r','LineWidth',1); %BEST
FOR PRESENTATION
ylim([min(latency_measurements_cluster)-5 max(latency_measurements_cluster)+5])
set(gca,'fontsize',12)
xlabel('CTLE Combinations (index,cap,res)','FontName','Times','FontSize',12);
ylabel('Latency Mean (Ticks)','FontName','Times','FontSize',12);
legend('Measurements','GPR Simulation','location','best');
set(legend,'FontSize',12,'FontName','Times');
% title('CXL MemRd s1p3<->s0p4 - Modeling')
grid on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    Test1 - Getting Optimal CTLE Coefficients
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ctle_idx_vec = [0 1];                % CTLE index coefficient
cap_vec = [0 1 2 3];                 % CTLE coefficient parameter Cap
res_vec = [0 1 2 3 4 5 6 7];         % CTLE coefficient parameter Res
% cap_vec = [0 0.5 1 1.5 2 2.5 3];                 % CTLE coefficient parameter Cap
% res_vec = [0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7];        % CTLE coefficient parameter Res

delta_time_opt_pattern_search_nelder_mead =
opt_pattern_search_and_nelder_mead(ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
delta_time_opt_genetic_algorithm =
opt_genetic_algorithm(ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
delta_time_opt_discrete =
opt_discrete_branch_and_bound(ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
fprintf('\nperformance_comparison >>> Execution Time delta_time_opt_pattern_search_nelder_mead %s',
datestr(delta_time_opt_pattern_search_nelder_mead,'HH:MM:SS.FFF')); %#ok<DATST>
fprintf('\nperformance_comparison >>> Execution Time delta_time_opt_genetic_algorithm %s',
datestr(delta_time_opt_genetic_algorithm,'HH:MM:SS.FFF')); %#ok<DATST>
fprintf('\nperformance_comparison >>> Execution Time delta_time_opt_discrete %s',
datestr(delta_time_opt_discrete,'HH:MM:SS.FFF')); %#ok<DATST>

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    Test2 - Performance Execution
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% vector_delta_time_opt_pattern_search_nelder_mead = 1:100;
% vector_delta_time_opt_genetic_algorithm = 1:100;
% vector_delta_time_opt_discrete = 1:1;
%
% for i = 1:100
%     delta_time_opt_pattern_search_nelder_mead =
opt_pattern_search_and_nelder_mead(ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
%     vector_delta_time_opt_pattern_search_nelder_mead(i) = delta_time_opt_pattern_search_nelder_mead;
% end
%
% for i = 1:100
%     delta_time_opt_genetic_algorithm =
opt_genetic_algorithm(ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
%     vector_delta_time_opt_genetic_algorithm(i) = delta_time_opt_genetic_algorithm;
% end
%
% for i = 1:1
%     delta_time_opt_discrete =
opt_discrete_branch_and_bound(ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
%     vector_delta_time_opt_discrete(i) = delta_time_opt_discrete;
% end
% median_vector_delta_time_opt_pattern_search_nelder_mead =
median(vector_delta_time_opt_pattern_search_nelder_mead);
% median_vector_delta_time_opt_genetic_algorithm = median(vector_delta_time_opt_genetic_algorithm);
% median_vector_delta_time_opt_discrete = median(vector_delta_time_opt_discrete);
%
% fprintf('median_vector_delta_time_opt_pattern_search_nelder_mead >>> Execution Time %s\n',
datestr(median_vector_delta_time_opt_pattern_search_nelder_mead,'HH:MM:SS.FFF')); %#ok<DATST>
% fprintf('median_vector_delta_time_opt_genetic_algorithm >>> Execution Time %s\n',
datestr(median_vector_delta_time_opt_genetic_algorithm,'HH:MM:SS.FFF')); %#ok<DATST>
% fprintf('median_vector_delta_time_opt_discrete >>> Execution Time %s\n',
datestr(median_vector_delta_time_opt_genetic_algorithm,'HH:MM:SS.FFF')); %#ok<DATST>
```

B. MATLAB Code: Objective Function Latency

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Student: Adan Baltazar
% Date: April 23, 2024
% Department of Electronics, Systems, and Informatics
% University: ITESO
% Thesis: CXL Latency Optimization
% Description:
%               This script computes a Gaussian Process Regression (GPR) model of mean
%               latency of a CXL memory read transaction when overwriting CTLE
%               parameters: CTLE_index, Cap, Res.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function U = objective_function_latency(Xo,ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster)

    global fevaluations u_i ctle_index_i cap_i res_i %#ok<GVMIS>

    fevaluations = fevaluations + 1;

    % Defining limits for Xo values based on CTLE_index_Cap, and Res vectors
    if Xo(1) < ctle_idx_vec(1)
            Xo(1) = ctle_idx_vec(1);
        elseif Xo(1) > ctle_idx_vec(2)
            Xo(1) = ctle_idx_vec(2);
    end
    if Xo(2) < cap_vec(1)
            Xo(2) = cap_vec(1);
        elseif Xo(2) > cap_vec(4)
            Xo(2) = cap_vec(4);
    end
    if Xo(3) < res_vec(1)
            Xo(3) = res_vec(1);
        elseif Xo(3) > res_vec(8)
            Xo(3) = res_vec(8);
    end

    ctle_index = Xo(1);
    cap = Xo(2);
    res = Xo(3);

    % Using (-) Find Maximum Global Points
    % Using (+) Find Minimum Global Points
    U = predict(gpr_model_latency_cluster,Xo);

    u_i(fevaluations) = U;
    ctle_index_i(fevaluations) = ctle_index;
    cap_i(fevaluations) = cap;
    res_i(fevaluations) =  res;

end
```

C. MATLAB Code: Optimizer Pattern Search Nelder Mead

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Student: Adan Baltazar
% Date: April 23, 2024
% Department of Electronics, Systems, and Informatics
% University: ITESO
% Thesis: CXL Latency Optimization
% Description:
%               This script computes Pattern-Search and Nelder-Mead optimization algorithms.
%               It looks for the optimal combination of CTLE_index, Cap, and Res
%               that creates the minimum CXL latency.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function t_delta =
opt_pattern_search_and_nelder_mead(ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Options Pattern Search
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    MaxIter = 1e12;
    MaxFunEvals = 1e12;
    tolx=1e-3;
    tolfun=1e-3;
    options = optimset('MaxFunEvals',MaxFunEvals,'MaxIter',MaxIter,'TolX',tolx,'TolFun',tolfun);

    %==================================================
    % We will use the fminsearch matlab function for Nelder-Mead optimization.
    % CTLE_index = [0-14], Cap = [0-3], Res = [0-7]
    Xo = [0,0,6];% Seed: Xo = [CTLE_index, Cap, Res]

    % Computing objective functin "u0(Xo)" (with initial values)
    ctle_index = Xo(1); %#ok<NASGU>
    cap = Xo(2); %#ok<NASGU>
    res = Xo(3); %#ok<NASGU>

    %   Global variables definition to register numer of models evaluations
    global fevaluations u_i res_i cap_i ctle_index_i; %#ok<GVMIS>
    fevaluations = 0;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Pattern_Search Options
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    opts = optimoptions('patternsearch');
    opts = optimoptions(opts,"Algorithm","classic");
    opts = optimoptions(opts,"TolX",1e-7,"TolFun",1e-7);
    opts = optimoptions(opts,"MeshContraction",0.5);
    opts = optimoptions(opts,"MeshExpansion",2.0);
    opts = optimoptions(opts,'AccelerateMesh',true);
    opts = optimoptions(opts,'ScaleMesh',false);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Define Algorithm
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    t1 = now; %#ok<TNOW1>

    objective_function = @(x)
objective_function_latency(x,ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
    [x,fval] = patternsearch(objective_function,Xo,[],[],[],[],[],[],[],opts); %#ok<ASGLU>
    Xo(1) = x(1);
    Xo(2) = x(2);
```

```
    Xo(3) = x(3);
    %-----------------------------------------------------
    [Xopt,fval] =
fminsearch(@objective_function_latency,Xo,options,ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
%#ok<ASGLU>
    Xo(1) = Xopt(1);
    Xo(2) = Xopt(2);
    Xo(3) = Xopt(3);

    [Xopt,fval] =
fminsearch(@objective_function_latency,Xo,options,ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
    [knob] = knobfilter(ctle_idx_vec',Xopt(1)); % Rounding Xo(1) to a valid value per Res vector
    Xopt(1) = knob;
    [knob] = knobfilter(cap_vec',Xopt(2));      % Rounding Xo(2) to a valid value per Cap vector
    Xopt(2) = knob;
    [knob] = knobfilter(res_vec',Xopt(3));      % Rounding Xo(3) to a valid value per Res vector
    Xopt(3) = knob;

    t2 = now; %#ok<TNOW1>

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Data For Plot
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ctle_index_opt = Xopt(1);    % Optimal CTLE_index parameter value
    cap_opt = Xopt(2);           % Optimal CTLE Cap parameter value
    res_opt = Xopt(3);           % Optimal CTLE Res parameter value
    fsol_cluster = fval;
    t_delta = t2 - t1;
    iteration_cluster = 1:fevaluations;
    function_cluster = u_i;
    resistance_cluster = res_i;
    capacitance_cluster = cap_i;
    ctle_index_cluster = ctle_index_i;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Graph
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    figure
    p1 = semilogx(iteration_cluster,function_cluster,'k');
    p1(1).LineWidth = 2;
    set(gca,'fontsize',12)
    xlabel('Evaluation','FontName','Times','FontSize',12);
    ylabel('Function Value','FontName','Times','FontSize',12);
    % title('Cluster Objective Function - Pattern Seach & Nealder Mead');
    grid on;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Graph ctle_index_i, res_i, cap_i
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    figure
    p2 = semilogx(iteration_cluster,ctle_index_cluster); hold on
    p3 = semilogx(iteration_cluster,resistance_cluster); hold on
    p4 = semilogx(iteration_cluster,capacitance_cluster);
    p2(1).LineWidth = 2;
    p3(1).LineWidth = 2;
    p4(1).LineWidth = 2;
    p2(1).LineStyle = ":";
    p2(1).Color = "red";
    p2(1).Marker = ".";
    p3(1).LineStyle = "--";
    p3(1).Color = "magenta";
    p3(1).Marker = ".";
    p4(1).LineStyle = "-.";
    p4(1).Color = "blue";
    p4(1).Marker = ".";
```

```
    set(gca,'fontsize',12)
    xlabel('Evaluation','FontName','Times','FontSize',12);
    ylabel('Optimization Variables','FontName','Times','FontSize',12);
    legend('ctle index','ctle res','ctle cap','location','best');
    % title('Cluster Objective Function - Pattern Seach & Nealder Mead');
    grid on;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                   TBD
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % P1 = semilogx(iteration_cluster3,Cm_Cluster3,'b-o'); hold on
    % p2 = semilogx(iteration_cluster3,Cp_Cluster3,'k-d');
    % p1(1).LineWidth = 1;
    % p2(1).LineWidth = 1;
    % hold off
    % set(gca,'fontsize',12)
    % xlabel('evaluation ','FontName','Times','FontSize',16);
    % ylabel('coefficients ','FontName','Times','FontSize',16);
    % legend('\it\rmCm','\it\rmCp','location','best');
    % set(legend,'FontSize',12,'FontName','Times');
    % title('Cluster3 Normalized Coefficients Responses');
    % grid on;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                   Print Results
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fprintf('\nopt_pattern_search >>> Best Objective Function :'),disp(fsol_cluster);
    fprintf('\nopt_pattern_search >>> Design Optimal Variables :'),disp(double([ctle_index_opt cap_opt
res_opt]));

end
```

D. MATLAB Code: Optimizer Genetic Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Student: Adan Baltazar
% Date: April 23, 2024
% Department of Electronics, Systems, and Informatics
% University: ITESO
% Thesis: CXL Latency Optimization
% Description:
%               This script computes Genetic Algorithm.
%               It looks for the optimal combination of CTLE_index, Cap, and Res
%               that creates the minimum CXL latency.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function t_delta = opt_genetic_algorithm(ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Options Pattern Search
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    global fevaluations u_i res_i cap_i ctle_index_i; %#ok<GVMIS>
    fevaluations = 0;
    MaxIter = 1e12;
    MaxFunEvals = 1e12;
    tolx=1e-12;
    tolfun=1e-12;
    options = optimset('MaxFunEvals',MaxFunEvals,'MaxIter',MaxIter,'TolX',tolx,'TolFun',tolfun);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Define Algorithm
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    t1 = now;
    % Genetic Algoritghm
    rng default         % For reproducibility
    objective_function = @(x)
objective_function_latency(x,ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
    lb = [0 0 0];       % lower bound
    ub = [1 3 7];       % upper bound
    [x,fval] = ga(objective_function,3,[], [], [], [], lb, ub); %#ok<ASGLU>

    Xo(1) = x(1);
    Xo(2) = x(2);
    Xo(3) = x(3);

    [Xopt,fval] =
fminsearch(@objective_function_latency,Xo,options,ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster);
    Xo(1) = Xopt(1);
    Xo(2) = Xopt(2);
    Xo(3) = Xopt(3); %#ok<NASGU>

    [knob] = knobfilter(ctle_idx_vec',Xopt(1));     % Rounding Xo(1) to a valid value per Res vector
    Xopt(1) = knob;
    [knob] = knobfilter(cap_vec',Xopt(2));          % Rounding Xo(2) to a valid value per Cap vector
    Xopt(2) = knob;
    [knob] = knobfilter(res_vec',Xopt(3));          % Rounding Xo(3) to a valid value per Res vector
    Xopt(3) = knob;

    t2 = now; %#ok<TNOW1>

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    %                                  Data For Plot
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ctle_index_opt = Xopt(1);                   % Optimal CTLE_index parameter value
    cap_opt = Xopt(2);                          % Optimal CTLE Cap parameter value
    res_opt = Xopt(3);                        % Optimal CTLE Res parameter value
    fsol_cluster = fval;
    iteration_cluster = 1:fevaluations;
    function_cluster = u_i;
    resistance_cluster = res_i;
    capacitance_cluster = cap_i;
    ctle_index_cluster = ctle_index_i;
    t_delta = t2 - t1;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Data For Plot
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    figure
    p1 = semilogx(iteration_cluster,function_cluster,'k');
    p1(1).LineWidth = 2;
    set(gca,'fontsize',12)
    xlabel('Evaluation','FontName','Times','FontSize',12);
    ylabel('Function Value','FontName','Times','FontSize',12);
    % title('Cluster Objective Function - Genetic Algorithm');
    grid on;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Graph ctle_index_i, res_i, cap_i
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    figure
    p2 = semilogx(iteration_cluster,ctle_index_cluster); hold on
    p3 = semilogx(iteration_cluster,resistance_cluster); hold on
    p4 = semilogx(iteration_cluster,capacitance_cluster);
    p2(1).LineWidth = 2;
    p3(1).LineWidth = 2;
    p4(1).LineWidth = 2;
    p2(1).LineStyle = ":";
    p2(1).Color = "red";
    p2(1).Marker = ".";
    p3(1).LineStyle = "--";
    p3(1).Color = "magenta";
    p3(1).Marker = ".";
    p4(1).LineStyle = "-.";
    p4(1).Color = "blue";
    p4(1).Marker = ".";
    set(gca,'fontsize',12)
    xlabel('Evaluation','FontName','Times','FontSize',12);
    ylabel('Optimization Variables','FontName','Times','FontSize',12);
    legend('ctle index','ctle res','ctle cap','location','best');
    % title('Cluster Objective Function - Pattern Seach & Nealder Mead');
    grid on;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Print Results
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fprintf('\nopt_genetic_algorithm >>> Best Objective Function :'),disp(fsol_cluster);
    fprintf('\nopt_genetic_algorithm >>> Design Optimal Variables :'),disp(double([ctle_index_opt cap_opt
res_opt]));
end
```

E. MATLAB Code: Optimizer Branch and Bound

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Student: Adan Baltazar
% Date: April 23, 2024
% Department of Electronics, Systems, and Informatics
% University: ITESO
% Thesis: CXL Latency Optimization
% Description:
%               This script computes Branch and Bound Discrete Optimization.
%               It looks for the optimal combination of CTLE_index, Cap, and Res
%               that creates the minimum CXL latency.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function t_delta = opt_discrete_branch_and_bound(ctle_idx_vec,cap_vec,res_vec,gpr_model_latency_cluster)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Variable Definition
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    global x1v; %#ok<GVMIS>
    global x2v; %#ok<GVMIS>
    global x3v;%#ok<GVMIS>
    global iterator_j; %#ok<GVMIS>
    global f2sol; %#ok<GVMIS>
    global noprune; %#ok<GVMIS>
    global noprune3; %#ok<GVMIS>
    global x1sol; %#ok<GVMIS>
    global x2sol; %#ok<GVMIS>
    global x3sol; %#ok<GVMIS>
    global u_1; %#ok<GVMIS>
    global u_2; %#ok<GVMIS>
    global iteration_cluster_count; %#ok<GVMIS>
    global function_cluster; %#ok<GVMIS>
    global function_cluster_without_zeros; %#ok<GVMIS>
    global iterator_cluster_without_zeros; %#ok<GVMIS>

    x1v  = ctle_idx_vec;    % CTLE index coefficient -> x1v
    x2v  = cap_vec;         % CTLE coefficient parameter Cap -> x2v
    x3v  = res_vec;         % CTLE coefficient parameter Res -> x3v
    iterator_j = 1;
    fsol = inf; %#ok<NASGU>
    f2sol = inf;
    f3sol = inf;
    noprune = [];
    noprune3 = [];
    x1sol = [];
    x2sol = [];
    x3sol = [];
    u_1 = [];
    u_2 = [];
    iteration_cluster_count = 1;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                   Define Algorithm
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    t1 = now; %#ok<TNOW1>
```

```
for iterator_1st_level = 1:length(x2v)
    x2b = x2v(iterator_1st_level);
    %---------------------------------------------------
    funf2 = @(x)(predict(gpr_model_latency_cluster,[x(1) x2b x(2)]));
    % Starting Points
    x0 = [0 0];
    % ++++++++ fminsearch solution ++++++++
    x = fminsearch(funf2,x0); % Call Matlab command fminsearch
    %---------------------------------------------------
    x1b = x(1);
    x3b = x(2);
    y = [x1b x2b x3b];
    fsol = feval(funf2,y); %#ok<FVAL>
    %---------------------------------------------------
    if fsol < f2sol
        f2sol(iterator_j) = fsol;
        x2sol(iterator_j) = x2b; %#ok<AGROW>
        noprune = [iterator_1st_level]; %#ok<NBRAK2>
        u_1(iteration_cluster_count) = fsol; %#ok<AGROW>
        iteration_cluster_count = iteration_cluster_count + 1;
    elseif fsol == f2sol
        iterator_j = iterator_j+1;
        f2sol(iterator_j) = fsol;
        x2sol(iterator_j) = x2b; %#ok<AGROW>
        noprune = [noprune iterator_1st_level]; %#ok<AGROW>
        u_1(iteration_cluster_count) = fsol; %#ok<AGROW>
        iteration_cluster_count = iteration_cluster_count + 1;
    end
end
%---------------------------------------------------
clear x1b x2b x3b

iterator_j=1;
for iterator_1st_level = 1:length(noprune)
    x2b = x2v(noprune(iterator_1st_level));
    for iterator_2nd_level = 1:length(x3v)
        x3b = x3v(iterator_2nd_level);
        %---------------------------------------------------
        for iterator_3rd_level = 1:length(x1v)
            x1b = x1v(iterator_3rd_level);
            %---------------------------------------------------
            Xo = [x1b x2b x3b];
            funf3 = @(x)(predict(gpr_model_latency_cluster,Xo));
            % ++++++++ fminsearch solution ++++++++
            z = fminsearch(funf3,Xo); % Call Matlab command fminsearch
            x1b = z(1);
            x2b = z(2);
            x3b = z(3);
            %---------------------------------------------------
            y = [x1b x2b x3b];
            fsol = feval(funf3,y); %#ok<FVAL>
            %---------------------------------------------------
            if fsol < f3sol
                f3sol(iterator_j) = fsol;
                x3sol(iterator_j) = x3b; %#ok<AGROW>
                x2sol(iterator_j) = x2b; %#ok<AGROW>
                x1sol(iterator_j) = x1b; %#ok<AGROW>
                noprune3 = [iterator_3rd_level]; %#ok<NBRAK2>
                u_1(iteration_cluster_count) = fsol; %#ok<AGROW>
                iteration_cluster_count = iteration_cluster_count + 1;
            elseif fsol == f3sol
                iterator_j = iterator_j+1;
                f3sol(iterator_j) = fsol;
                x3sol(iterator_j) = x3b; %#ok<AGROW>
                x2sol(iterator_j) = x2b; %#ok<AGROW>
                x1sol(iterator_j) = x1b; %#ok<AGROW>
                noprune3 = [noprune3 iterator_3rd_level]; %#ok<AGROW>
```

```
                    u_1(iteration_cluster_count) = fsol; %#ok<AGROW>
                    iteration_cluster_count = iteration_cluster_count + 1;
                end
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                 Results
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    t2 = now; %#ok<TNOW1>
    t_delta = t2 - t1;
    x1d = min(x1sol);
    x2d = min(x2sol);
    x3d = min(x3sol);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Data For Plot
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function_cluster = u_1;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Graph
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    figure
    function_cluster_without_zeros = nonzeros(function_cluster');
    iterator_cluster_without_zeros = 1:iteration_cluster_count-1;
    p1 = semilogx(iterator_cluster_without_zeros,function_cluster_without_zeros,'k');
    p1(1).LineWidth = 2;
    set(gca,'fontsize',12);
    xlabel('Evaluation','FontName','Times','FontSize',12);
    ylabel('Function Value','FontName','Times','FontSize',12);
    % title('Cluster Objective Function - Branch & Bound - DO');
    grid on;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                  Print Results
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fprintf('\nopt_discrete >>> Best Objective Function :'),disp(f3sol(1));
    fprintf('\nopt_discrete >>> Design Optimal Variables :'),disp(double([x1d x2d x3d]));

end
```

F. Pseudo Code: Branch and Bound

```
Initialize:
    Incumbent := ∞;
    LB(P0) ≔ g(P0);
    Live ≔ {(P0,LB(P0))};

Repeat until Live= Ø
    Select the node P from Live to be processed;
    Live := Live \ {P};
    Branch on P generating P1,...Pk;
    For 1 ≤ i ≤ k do
        Bound Pi:
                            LB(Pi:=g(Pi));
            If LB(Pi) = f(X) for a feasible solution X and f(X) < Incumbent
                then
                    Incumbent := f(X);
                    Solution := X;
                goto EndBound;
            If LB(Pi) ≥ Incumbent
                then fathom Pi
            else
                Live := Live ∪ {(Pi,LB(Pi))}
        EndBound;

OptimalSolution :=  Solution;
OptimalValue := Incumbent; [1]
```

(1)

7. References

[1] PCI-SIG, *PCI Express Base Specification Revision 6.0, Version 1.0,* August. 25, 2022, https://members.pcisig.com/wg/PCI-SIG/document/16609.

[2] rson, and Ravi Budruk, *PCI Express System Architecture*, 1st ed. United States of America: Addison-Wesley Professional, 2003

[3] Das Sharma, *The Evolution of the PCI Express Specification: On its Sixth Generation, Third Decade and Still Going Strong*, January. 11, 2022, https://pcisig.com/blog/evolution-pci-express-specification-its-sixth-generation-third-decade-and-still-going-strong.

[4] Solomon, *PCI Express Basics* , 2022, https://members.pcisig.com/wg/PCI-SIG/document/17304.

[5] Keysight Technologies, *Compute Express Link (CXL): A New Coherency Protocol Over PCIe*, December. 23, 2022, https://www.keysight.com/us/en/assets/3122-1353/application-notes/Compute-Express-Link-CXL-A-New-Coherency-Protocol-Over-PCIe.pdf.

[6] Compute Express Link, *CXL Specification_rev2p0_ver1p0_2020Oct26_clean_Approved*, November. 9, 2020, https://members.computeexpresslink.org/wg/CXL/document/1172.

[7] Rambus, *CXL Glossary*, https://www.rambus.com/interface-ip/cxl-glossary/.

[8] Rambus, *Data Center Evolution: The Leap to 64 GT/s Signaling with PCI Express 6.1,* January. 2024, https://go.rambus.com/l/803123/2022-01-25/3b1yyf/803123/1643137617AZZzGMjZ/Data_Center_Evolution_The_Leap_to_64GTs_Signaling_PCI_Express_6.0._wp.pdf.

[9] Domos, *Latency Explained*, 2024, https://www.domos.no/latency-explained/#what-is-latency.

[10] JAYBRIAN, *Solving PCIe Latency with Retimers and Elastic Buffers,* July. 22, 2022, https://www.edn.com/a-short-primer-on-pcie-latency-and-its-optimization-with-retimers/.

[11] Patrick, *Compute Express Link CXL Latency How Much is Added at HC34*, August. 21, 2022, https://www.servethehome.com/compute-express-link-cxl-latency-how-much-is-added-at-hc34/#:~:text=The%20CXL%20Consortium%20is%20using,170%2D250ns%20for%20CXL%20memory.&text=If%20CXL%20seems%20to%20be,with%20Q2%202022%20Wind%2DDown.

[12] Astera Labs, *The Impact of Bit Errors in PCI Express® Links: The Painful Realities of Low-Probability Events*, https://www.asteralabs.com/smart-retimers/impact-of-bit-errors-in-pci-express-links/.

[13] Synopsis, *Synopsys Optimizes DesignWare IP for PCI Express 4.0 Architecture to Reduce Latency by up to 20 Percent*, June. 2016, https://news.synopsys.com/2016-06-21-Synopsys-Optimizes-DesignWare-IP-for-PCI-Express-4-0-Architecture-to-Reduce-Latency-by-up-to-20-Percent.

[14] Samie B. Samaan, Dan Froelich, and Samuel Johnson. (2015, Oct) High-Speed Serial Bus Repeater Primer for Properties and Usage. [Online]. https://www.intel.es/content/dam/www/public/us/en/documents/white-papers/serial-bus-white-paper.pdf

[15] Synopsys, *Reduce Link Bring-Up Time with PCIe 5.0 Modes*, October. 2019, https://www.synopsys.com/blogs/chip-design/reduce-pcie-5-0-link-time.html.

[16] Rambus, *PCI Express Controller IP*, https://www.rambus.com/interface-ip/pci-express/.

[17] EE Journal, *Rambus Delivers CXL 2.0 Controller with Industry-leading Zero-Latency IDE*, October. 6, 2021, https://www.eejournal.com/industry_news/rambus-delivers-cxl-2-0-controller-with-industry-leading-zero-latency-ide/?gclid=Cj0KCQjwz7uRBhDRARIsAFqjulm9lErkws_q-ofdhrfJtC-4snaQSSf0m3dzwCApp5YgXEYqzO59BWIaAhWaEALw_wcB.

[18] Francisco E. Rangel-Patiño, José E. Rayas-Sánchez, Edgar A. Vega-Ochoa, and Nagib Hakim, "Direct Optimization of a PCI Express Link Equalization in Industrial Post-Silicon Validation," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, Sao Paulo, Brazil, April 2018.

[19] PCI-SIG, *PCI Express Base Specification Revision 5.0, Version 1.0*, May. 2019, https://members.pcisig.com/wg/PCI-SIG/document/previewpdf/13005.

[20] SanjeevAgarwal, *PCI Express 3.0 Equalization: The Mystery Unsolved*, https://www.logic-fruit.com/whitepaper/pci-express-3-0-equalization/.

[21] Zhang Hanqiao, Krooswyk Steve , and Ou Jeff, *High Speed Digital Design: Design of High Speed Interconnects and Signaling*, 1st ed. United States of America: Morgan Kaufmann, 2015

[22] *PCIe 5.0 Equalization Modes: Reducing Link Bring-Up Time*, October. 24, 2019, https://blogs.synopsys.com/vip-central/2019/10/24/pcie-5-0-equalization-modes-reducing-link-bring-up-time/#:~:text=Like%20PCIe%203.0%20and%204.0,same%20as%20the%20previous%20generations.

[23] P Shanmuga, *PCIe Gen 6 Equalization Unraveled*, https://www.logic-fruit.com/wp-content/uploads/2021/09/PCIe-Gen-6-Equalization-Unraveled.pdf.

[24] Teledyne LeCroy, *Get Ready for Compute Express Link (CXL) Protocol Compliance Testing*, August. 2021, https://go.teledynelecroy.com/l/48392/2021-08-20/88zj51.

[25] Stephens , *Equalization Techniques: CTLE, DFE, FFE, and Crosstalk*, October. 21, 2015, https://www.edn.com/optimize-equalization-for-ffe-ctle-dfe-and-crosstalk/.

[26] Thulasiraman Deepanraj , G N Chiranjeevi , S Gaggatur Javed, and Sankara Reddy K S, "A 18.6 fJ/bit/dB Power Efficient Active Inductor-based CTLE for 20 Gb/s High Speed Serial Link," in *2019 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, Bangalore, India, February 2020.

[27] KIRGAT G.S and Surde A.N, "Review of Hooke and Jeeves Direct Search Solution Method Analysis," *INTERNATIONAL JOURNAL OF INNOVATIONS IN ENGINEERING RESEARCH AND TECHNOLOGY [IJIERT]*, vol. Volume 1, no. ISSUE 2 DEC-2014, p. 14, December 2014.

[28] Margaret H. Wright, "Nelder, Mead, and the Other Simplex Method," *Documenta Mathematica Extra Volumes 2012*, pp. 271-276, 2012.

[29] Goldberg David E, *Genetic Algorithms in Search, Optimization & Machine Learning*, 1st ed. United States of America: Addison-Wesley Publishing Company, Reading, Mass., 1989

[30] Venkataraman P. , *Branch and Bound*. Rochester Institute of Technology: A WILEY-INTERSCIENCE PUBLICATION JOHN WiLEY & SONS, INC, 2002

[31] Jie Wang, "An Intuitive Tutorial to Gaussian Process Regression," *Computing in Science & Engineering*, vol. 25, pp. 4-11, Jul-Aug 2023.

[32] Roberto Jorge Ruiz Urbina, *PAM4 Transmitter and Receiver Equalizers*, October. 2021, https://rei.iteso.mx/server/api/core/bitstreams/13269b57-648f-440d-b34c-f42e9278f53e/content.

[33] Jade Yu ChengThomas Mailund, *https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method#/media/File:An-iteration-of-the-Nelder-Mead-method-over-two-dimensional-space-showing-point-p-min.png*, March. 16, 2015, https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method#/media/File:An-iteration-of-the-Nelder-Mead-method-over-two-dimensional-space-showing-point-p-min.png.

[34] MathWorks, *What Is a Genetic Algorithm?*, https://www.mathworks.com/discovery/genetic-algorithm.html.

[35] Prasun, *Genetic Algorithm - An Optimization Approach*, November. 15, 2022, https://towardsdatascience.com/genetic-algorithm-an-optimization-approach-dd60e23261e6#:~:text=Genetic%20Algorithm%20(GA)%20is%20a,a%20significant%20amount%20to%20solve.

[36] P Venkataram, *Applied Optimization With Matlab*, 2nd ed. Hoboken, New Jersey: John Wiley & Sons Inc, 2009

[37] MathWorks, *patternsearch*, 2006, https://www.mathworks.com/help/gads/patternsearch.html?s_tid=srchtitle_site_search_1_patternsearch.

[38] MathWorks, *fminsearch*, https://www.mathworks.com/help/matlab/ref/fminsearch.html?searchHighlight=fminsearch&s_tid=srchtitle_support_results_1_fminsearch.

[39] MathWorks, *GA*, https://www.mathworks.com/help/gads/ga.html?s_tid=doc_ta.

[40] MathWorks, *predict*, 2019, https://www.mathworks.com/help/deeplearning/ref/dlnetwork.predict.html?searchHighlight=predict&s_tid=srchtitle_support_results_1_predict.