MEETING 5G RELIABILITY DEMANDS WITH HARDWARE ACCELERATION IN CELL OUTAGE COMPENSATION FOR SELF-ORGANIZING NETWORK

By

Al-Shaimaa Yusuf Diab
Abdallah Nagy Brisha
Eman Ahmed Zyada
Hatem Medhat Elshorbagy
Hossam Hassanin Abdelhay
Hussein Sobhy El-Shamy
Mohamed Saeed Helal
Mohamed Shaban Abdelhadi
Mohammed Mahmoud Shalaby

A Thesis Submitted to the
Faculty of Engineering at Tanta University
in Partial Fulfillment of the Requirements for the Degree of
BACHELOR OF SCIENCE
in
Electronics and Communications Engineering

Under the Supervision of

Dr.Hussein El-Taibee Sleem
Assistant Professor
Electronics and Communications Engineering
Faculty of Engineering , Tanta University

Technically Sponsored By
Valeo

FACULTY OF ENGINEERING ,TANTA UNIVERSITY
TANTA,EGYPT
2023

MEETING 5G RELIABILITY DEMANDS WITH HARDWARE ACCELERATION IN CELL OUTAGE COMPENSATION FOR SELF-ORGANIZING NETWORK

By

Al-Shaimaa Yusuf Diab
Abdallah Nagy Brisha
Eman Ahmed Zyada
Hatem Medhat Elshorbagy
Hossam Hassanin Abdelhay
Hussein Sobhy El-Shamy
Mohamed Saeed Helal
Mohamed Shaban Abdelhadi
Mohammed Mahmoud Shalaby

A Thesis Submitted to the
Faculty of Engineering at Tanta University
in Partial Fulfillment of the Requirements for the Degree of
BACHELOR OF SCIENCE
in
Electronics and Communications Engineering

Under the Supervision of

Dr.Hussein El-Taibee Sleem
Assistant Professor
Electronics and Communications Engineering
Faculty of Engineering , Tanta University

Technically Sponsored By
Valeo

FACULTY OF ENGINEERING ,TANTA UNIVERSITY
TANTA,EGYPT
2023

Student's Name: Al-Shaimaa Yusuf Teleb Diab
Date of Birth: 01/04/2000
Nationality: Egyptian
E-mail: elshaimaa30981255@f-eng.tanta.edu.eg
Phone: 01126045918
Address: Zefta, Gharbia Governorate, 31641
Registration Date: September 2018
Awarding Date: September 2023
Degree: Bachelor of Science
Department: Electronics and Communications Engineering

Student's Name: Abdallah Nagy Mohamed Brisha
Date of Birth: 10/12/2000
Nationality: Egyptian
E-mail: abdallah30981622@f-eng.tanta.edu.eg
Phone: 01063735748
Address: Tanta, Gharbia Governorate, 31733
Registration Date: September 2018
Awarding Date: July 2024
Degree: Bachelor of Science
Department: Electronics and Communications Engineering

Student's Name: Eman Ahmed Zyada Al-sherbiny
Date of Birth: 14/09/2001
Nationality: Egyptian
E-mail: eman30981302@f-eng.tanta.edu eg
Phone: 01068737955
Address: El-Santa, Gharbia Governorate, 31631
Registration Date: September 2018
Awarding Date: September 2023
Degree: Bachelor of Science
Department: Electronics and Communications Engineering

Student's Name: Hatem Medhat Mohamed Elshorbagy
Date of Birth: 20/09/2000
Nationality: Egyptian
E-mail: hatem30981565@f-eng.tanta.edu.eg
Phone: 01558090667
Address: El-Santa, Gharbia Governorate, 31667
Registration Date: September 2018
Awarding Date: September 2023
Degree: Bachelor of Science
Department: Electronics and Communications Engineering

Student's Name:	Hossam Hassanin Abdelshahid Abdelhay
Date of Birth:	22/05/2000
Nationality:	Egyptian
E-mail:	hossam30998689@f-eng.tanta.edu.eg
Phone:	01554574031
Address:	Kafr Elzayat, Gharbia Governorate, 31611
Registration Date:	September 2018
Awarding Date:	September 2023
Degree:	Bachelor of Science
Department:	Electronics and Communications Engineering

Student's Name:	Hussein Sobhy Sobhy El-Sebaey Hassanin El-Shamy
Date of Birth:	05/12/2000
Nationality:	Egyptian
E-mail:	hussein30981883@f-eng.tanta.edu.eg
Phone:	01064490820
Address:	Basioun, Gharbia Governorate, 31714
Registration Date:	September 2018
Awarding Date:	September 2023
Degree:	Bachelor of Science
Department:	Electronics and Communications Engineering

Student's Name:	Mohamed Saeed Abdsalam Helal
Date of Birth:	16/09/2000
Nationality:	Egyptian
E-mail:	mohammed30982076@f-eng.tanta.edu.eg
Phone:	01003670535
Address:	Basioun, Gharbia Governorate, 31778
Registration Date:	September 2018
Awarding Date:	September 2023
Degree:	Bachelor of Science
Department:	Electronics and Communications Engineering

Student's Name:	Mohamed Shaaban Ahmed Abdelhady
Date of Birth:	20/08/2000
Nationality:	Egyptian
E-mail:	mohammed31038894@f-eng.tanta.edu.eg
Phone:	01117508013
Address:	Ihnasia, Benisuef Governorate, 62631
Registration Date:	September 2018
Awarding Date:	September 2023
Degree:	Bachelor of Science
Department:	Electronics and Communications Engineering

Student's Name:	Mohammed Mahmoud Mohammed El-Tohamy
Date of Birth:	08/03/2000
Nationality:	Egyptian
E-mail:	mohammed30981976@feng.tanta.edu.eg
Phone:	01128088651
Address:	El-Santa, Gharbia Governorate, 31631
Registration Date:	September 2018
Awarding Date:	September 2023
Degree:	Bachelor of Science
Department:	Electronics and Communications Engineering

Title of Thesis:

Meeting 5G Reliability Demands with Hardware Acceleration in Cell Outage Compensation for Self-Organizing Network

Supervisor:
Dr.Hussein El-Taibee Sleem

Key Words:
Self-Organzing Networks; Self-Healing; Cell Outage Compensation; Q-Learning; Hardware Acceleration; FPGA

Summary:
Accelerating the Cell Outage Compensation (COC) algorithm in Self-Organizing Networks(SONs) through AI-Hardware Acceleration to enhance 5G network reliability and support emerging technologies such as autonomous driving

Disclaimer

I hereby declare that this thesis is my own original work and that no part of it has been submitted for a degree qualification at any other university or institute.

I further declare that I have appropriately acknowledged all sources used and have cited them in the references section.

Name: Al-Shaimaa Yusuf Diab Date:
Signature:

Name: Abdallah Nagy Brisha Date:
Signature:

Name: Eman Ahmed Zyada Date:
Signature:

Name: Hatem Medhat Elshorbagy Date:
Signature:

Name: Hossam Hassanin Abdelhay Date:
Signature:

Name: Hussein Sobhy El-Shamy Date:
Signature:

Name: Mohamed Saeed Helal Date:
Signature:

Name: Mohamed Shaban Abdelhadi Date:
Signature:

Name: Mohammed Mahmoud Shalaby Date:
Signature:

Acknowledgements

We would like to express our sincere gratitude to our supervisor, **Dr. Hussein El-Taeibe Sleem**, for their invaluable guidance, support, and expertise throughout the entire process of conducting this research. Their insightful feedback and encouragement greatly contributed to the success of this thesis.

We also extend our heartfelt appreciation to **families and friends** for their unwavering support and encouragement during this academic journey. Their constant belief in our abilities and their understanding of the demands of this endeavor have been instrumental in our perseverance and accomplishment.

Furthermore, we would like to express our sincere appreciation for the resources and facilities provided by **Valeo Egypt**. We are grateful to our mentor, **Eng. Enas Atef**, for her invaluable guidance and support throughout the entire research process. We also extend our thanks to the Graduation Project Support Program Coordinator, **Dr. Neveen Abdelkader**, and Valeo Academy Program Manager, **Eng. Osama Shaaban**, for their support.The access to technical materials, support training sessions with experts, and provision of electronic equipment greatly enhanced the quality of our study. We are grateful for the opportunities provided through the Techie Fellowship Program, which added significant value to our team's skills in Embedded Software Development.

Lastly, we are thankful to all the individuals who have contributed in various ways, be it through discussions, suggestions, or constructive criticism, that have shaped the development of this thesis. Your contributions have been invaluable in refining the ideas and arguments presented here.

Table of Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

Abbreviation	Description
GSM	Global System for Mobile Communications
3GPP	3rd Generation Partnership Project
Rel	RELease
GPRS	General Packet Radio Services
EDGE	Enhanced Data rates for GSM Evolution
UMTS	Universal Mobile Communications System
HSDPA	High Speed Downlink Packet Access
LTE	Long Term Evolution
LTE-A	LTE-Advanced
IMT	International Mobile Telecommunications
ITU-R	International Telecommunication Union - Radiocommunication Sector
ITU-T	International Telecommunication Union - Telecommunication Sector
HDTV	High-Definition TeleVision
IMS	IP Multimedia Subsystem
SIP	Session Initiation Protocol
SBI	Service-Based Interface
NFV	Network Function Virtualization
SDN	Software-Defined Networking
TCO	Total Cost of Ownership
TMN	Telecommunication Management Network
FCAPS	Fault, Configuration, Accounting, Performance, and Security
CAN	Cognitive Autonomous Network
CNM	Cognitive Network Management
CM	Configuration Management

FM	Fault Management
NE	Network Element
NGMN	Next Generation Mobile Networks
OAM	Operations, Administration, and Maintenance
OMC	Operational and Maintenance Centre
PM	Performance Management
PnP	Plug-and-Play
RAT	Radio Access Technologie
SON	Self-Organizing Network
RL	Reinforcement Learning
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
GPU	Graphical Processing Unit
FPGA	Field-Programmable Gate Array
ASIC	Application-Specific Integrated Circuit
EDIF	Electronic Design Interchange Format
DEF	Design Exchange Format
STA	Static Timing Analysis
LUTs	LookUp Tables
ATs	Arrival Times
RATs	Required Arrival Times
PRNG	Pseudo Random Number Generator
LFSR	Linear Feedback Shift Register
RTL	Register Transfere Level
RAM	Random Access Memory

Abstract

This thesis presents a comprehensive study on Q-Learning Hardware Acceleration for Cell Outage Compensation (COC) scenarios in the context of network management automation. The increasing complexity and dynamic nature of modern communication networks, particularly in the advent of 5G and beyond, necessitate efficient and real-time decision-making algorithms. Reinforcement learning, specifically Q-Learning, has shown promise in optimizing network performance and reliability. However, the speed limitations of traditional software-based implementations hinder their effectiveness in real-time scenarios. To address this challenge, this research focuses on the design and implementation of FPGA-based Q-Learning hardware, aiming to accelerate the decision-making process for COC scenarios. The thesis begins with an introduction to the research problem, followed by a thorough exploration of the theoretical framework, including reinforcement learning concepts and network management automation. The subsequent chapters discuss the modelling of COC scenarios using MATLAB and the design and implementation of FPGA-based Q-Learning hardware. Through experimental evaluations, the proposed hardware solution demonstrates its ability to enhance real-time decision-making in COC scenarios. The thesis concludes with a summary of the key findings, their implications, and potential future research directions in the field of Q-Learning Hardware Acceleration for network management automation.

Chapter 1

Introduction

1.1 Background and Context

The evolution of mobile networks generation has had a significant impact on the complexity of these networks. Starting from the first generation (1G) networks, subsequent generations such as 2G, 3G, and 4G brought advancements in digital communication, higher data rates, and the introduction of services like internet access and multimedia. With the advent of 5G and beyond, mobile networks have undergone a paradigm shift, offering ultra-fast speeds, low latency, massive connectivity, and support for emerging technologies like Internet of Things (IoT) and autonomous vehicles. However, this evolution has also resulted in increased complexity in terms of network architecture, protocols, spectrum management, security, and the need for advanced technologies like beamforming, MIMO, and network slicing. Managing and optimizing these complex mobile networks has become a critical challenge for network operators, requiring advanced techniques and tools for network planning, optimization, and troubleshooting.

Table 1.1: Mobile subscriptions by technology (billion) - Ericsson Mobility Report June 2023

Technology	**Number of Subscribers (in millions)**
LTE/TD-LTE	5.168
5G	1.481
GSM	0.94
WCDMA/HSPA	0.81
CDMA	0.0157

Network management plays a crucial role in addressing the complexity of modern mobile networks resulting from the evolution of mobile systems across generations. With each new generation, mobile networks have become more advanced and feature-rich, offering enhanced services and capabilities. However, these advancements have also introduced challenges in terms of network design, deployment, operation, and maintenance. Network management encompasses a range of activities and technologies aimed at effectively managing and optimizing mobile networks to ensure reliable and efficient operation. It involves tasks such as network planning, configuration management, performance monitoring, fault detection and resolution, security

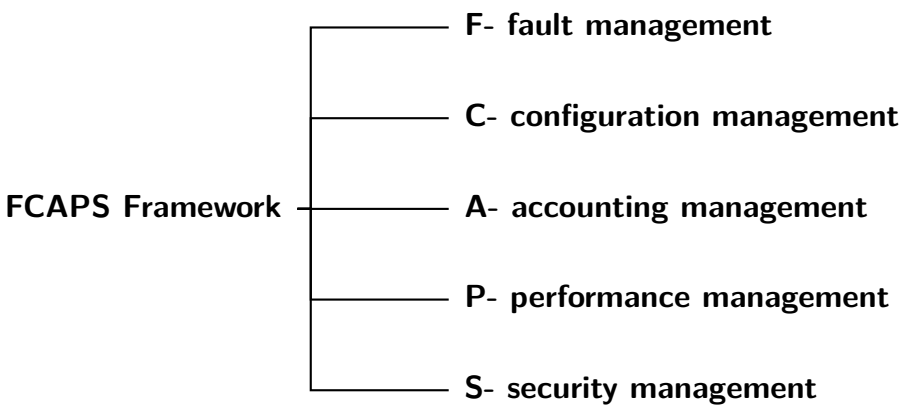


Figure 1.1: FCAPS Framework

management, and capacity optimization. Through effective network management practices and the use of intelligent automation, network operators can streamline operations, proactively address issues, and enhance the overall performance, reliability, and user experience of mobile networks. By leveraging network management solutions, operators can efficiently adapt to the evolving requirements of mobile systems, mitigate complexities, and optimize network performance to meet the growing demands of mobile users and emerging technologies.

Network automation has become increasingly important in next-generation networks due to the growing complexity and scale of network infrastructure. Automation can be categorized into different levels, each with varying degrees of intelligence and autonomy. At the lowest level, basic automation involves manual configuration and provisioning tasks, which can be time-consuming and prone to human errors. As we move up the automation ladder, we encounter higher levels of automation, such as policy-driven automation, where predefined rules and policies guide network operations. Further up, we find cognitive automation, which leverages machine learning and AI techniques to enable networks to learn and adapt based on real-time data and feedback. Finally, at the highest level, autonomous networks have the ability to make decisions and take actions independently, leveraging advanced algorithms and analytics. The need for network management automation arises from the inherent complexity of next-generation networks, which encompass a diverse range of devices, technologies, and services. By automating network management tasks, operators can improve operational efficiency, reduce human errors, enhance network reliability and performance, and enable rapid service deployment and innovation. Automation also enables networks to dynamically adapt to changing conditions, optimize resource allocation, and proactively respond to security threats. Overall, network management automation is essential to unlock the full potential of next-generation networks, enabling operators to effectively manage and scale their networks to meet the increasing demands of modern connectivity.

Self-Organizing Networks (SONs) represent a paradigm shift in network management and optimization, aiming to automate and simplify the operation of modern telecommunications networks. SONs employ intelligent algorithms and techniques to autonomously configure, optimize, and heal network elements without human

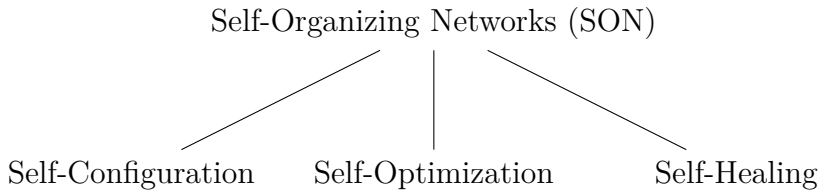


Figure 1.2: Self-Organizing Networks Sub-Functions

intervention. There are three main types of SON architectures: centralized SON (C-SON), distributed SON (D-SON), and hybrid SON (H-SON), each offering different degrees of control and flexibility. C-SON centralizes decision-making and management functions, allowing for global optimization but may suffer from scalability issues. D-SON distributes intelligence to individual network elements, providing local optimization and scalability but lacking global coordination. H-SON combines the advantages of both C-SON and D-SON, striking a balance between centralized and distributed control. SONs consist of various sub-functions such as self-configuration, self-optimization, and self-healing. Self-configuration enables plug-and-play deployment of network elements, self-optimization dynamically adjusts network parameters to optimize performance, and self-healing ensures fault detection and recovery. By employing SON architectures and sub-functions, network operators can improve network efficiency, reduce operational costs, enhance user experience, and enable rapid deployment and scalability of next-generation networks.

The self-healing function is an integral part of Self-Organizing Networks (SON) and plays a crucial role in ensuring the robustness and reliability of modern telecommunications networks. Self-healing aims to automatically detect, diagnose, and resolve network faults or failures without manual intervention. By continuously monitoring the network, the self-healing function can identify any anomalies, such as cell outages, link failures, or congestion, and take appropriate corrective actions in real-time. These actions may include reconfiguring network parameters, rerouting traffic, or activating backup resources to restore service and maintain optimal network performance. The self-healing function minimizes service disruptions, improves network availability, and enhances the overall quality of service for end-users. It enables networks to recover quickly from failures, adapt to changing conditions, and maintain seamless connectivity, ensuring a reliable and robust communication infrastructure.

Reinforcement Learning (RL) algorithms have gained significant attention in the context of Self-Organizing Networks (SON) due to their ability to learn and adapt in dynamic and complex environments. RL enables SON to make intelligent and autonomous decisions to optimize network performance and efficiency. However, there are challenges that need to be addressed for RL algorithms to be effective in SON. One of the key challenges is the speed of the algorithms, especially in the context of real-time decision-making required in 5G networks. RL algorithms need to quickly process and analyze large amounts of data to make timely decisions, which can be computationally intensive. To overcome this challenge, hardware acceleration techniques such as GPUs, FPGAs, or ASICs can be employed to enhance the speed and efficiency of RL algorithms, enabling them to meet the real-time requirements

of next-generation networks. By leveraging hardware acceleration, RL algorithms can achieve the required processing power and speed to effectively optimize network operations and adapt to changing network conditions in a timely manner.

1.2 Problem Statement and Research Objective

1.2.1 Problem Statement

The speed limitations of Reinforcement Learning (RL) algorithms used in self-healing networks, specifically in the context of cell outage compensation (COC) problem, pose a significant challenge in achieving the required level of reliability for 5G communication systems. In Ultra-Reliable Low Latency Communication (URLLC) use cases like autonomous vehicles, where real-time decision-making is crucial, the slow processing speed of RL algorithms hinders their ability to provide timely and efficient solutions. This can lead to delays in network restoration and negatively impact the reliability and performance of critical communication services. Addressing the speed limitations of RL algorithms is essential to ensure the effective and rapid restoration of network connections, particularly in URLLC scenarios, where any disruption can have severe consequences. By overcoming these speed limitations, the self-healing capability of networks can be enhanced, enabling faster response times and improved reliability for 5G communication systems, thereby meeting the stringent requirements of URLLC use cases such as autonomous vehicles.

1.2.2 Research Objective

The objective of this research is to develop an efficient and accelerated approach for connection restoration in the case of cell outages, leveraging the power of reinforcement learning algorithms in self-healing networks. The research aims to address the speed limitations of existing RL algorithms by exploring hardware acceleration techniques and optimizing their implementation. By improving the speed and efficiency of RL algorithms, the research seeks to enhance the real-time decision-making capabilities of self-healing networks, particularly in the context of cell outage compensation (COC) problem in 5G communication systems. The research aims to investigate the use of hardware accelerators such as GPUs, FPGAs, or ASICs to expedite the processing and analysis of data required for RL-based decision-making. Additionally, the research intends to optimize the implementation of RL algorithms by exploring algorithmic optimizations, parallel processing, and efficient data handling techniques. By achieving accelerated connection restoration, the research aims to enhance the reliability, performance, and responsiveness of self-healing networks, enabling faster recovery from cell outages and improving the overall quality of service for critical communication applications.

1.3 Approaches and Methodology

In order to accelerate AI algorithms, several approaches can be employed. These approaches can be categorized into hardware acceleration and software acceleration techniques.

Table 1.2: Approaches to Accelerate AI Algorithms

Type	Approach
Hardware Acceleration	Processor selection Parallelization Use of external GPUs Additional AI accelerator ASIC FPGA implementation
Software Acceleration	Pruning Remove sparsity Quantization Pre-processing Workflow optimization Parallelization

1.3.1 Hardware Acceleration:

One approach is to leverage hardware acceleration techniques to improve the performance of AI algorithms. This includes selecting the appropriate processors that are optimized for AI computations, such as graphics processing units (GPUs) or tensor processing units (TPUs). Parallelization can also be utilized by distributing the workload across multiple cores or processors to increase processing speed. Additionally, external GPUs or dedicated AI accelerator ASICs can be utilized for even faster computations. Another hardware acceleration approach involves implementing the AI algorithms on field-programmable gate arrays (FPGAs), which provide flexibility and high-performance computing capabilities.

1.3.2 Software Acceleration:

Software-based approaches can also be used to accelerate AI algorithms. One method is pruning, which involves removing unnecessary connections or parameters in the neural network to reduce computational requirements. Sparsity removal can also be applied to optimize the network by filling in the gaps created by pruning. Quantization is another technique that reduces the precision of weights and activations, resulting in smaller memory requirements and faster computations. Pre-processing the data by applying data transformations or feature extraction techniques can also improve the efficiency of AI algorithms. Workflow optimization,

such as streamlining the data flow and minimizing redundant computations, can further enhance the performance. Lastly, parallelization can be utilized by dividing the workload into smaller tasks that can be executed concurrently.

1.4 Proposed Solution

1.4.1 Develop a MATLAB-Based Model:

The first part of the proposed solution involves developing a MATLAB-based model to simplify the understanding of the cell outage compensation problem and the behavior of the Q-Learning algorithm. This model will provide a visual representation of the problem and the learning process, allowing for better insights and analysis. By implementing the model in MATLAB, it will be easier to experiment with different scenarios, parameters, and algorithms, enabling researchers to gain a deeper understanding of the problem and evaluate the performance of different approaches. The MATLAB-based model will serve as a valuable tool for studying and optimizing the cell outage compensation problem in self-healing networks.

1.4.2 Implement Reinforcement Learning Q-Learning Technique on FPGA:

The second part of the proposed solution focuses on the implementation of the reinforcement learning Q-Learning technique on a field-programmable gate array (FPGA) for enhanced real-time decision-making. FPGA offers high-performance computing capabilities and parallel processing, making it well-suited for accelerating computations and meeting the real-time requirements of next-generation networks. By implementing the Q-Learning algorithm on FPGA, the speed limitations of the algorithm can be overcome, enabling faster and more efficient decision-making. The FPGA implementation will be optimized to leverage the hardware capabilities and ensure reliable and robust performance. The enhanced real-time decision-making provided by the FPGA implementation will contribute to improving the reliability and responsiveness of self-healing networks, specifically in the context of cell outage compensation.

By combining the MATLAB-based model for analysis and understanding with the FPGA implementation for accelerated real-time decision-making, this research aims to provide a comprehensive solution to the challenges faced in cell outage compensation and the need for fast and reliable decision-making in next-generation networks. The proposed solution will contribute to advancing the field of self-healing networks and enabling the development of more efficient and resilient communication systems.

1.5 Related Works and Literature Review

S. Adel et al., "Cell Outage Compensation Using Q-learning for Self-Organizing Networks," 2021 International Conference on Microelectronics (ICM), New Cairo City, Egypt, 2021, pp. 102-105, doi: 10.1109/ICM52667.2021.9664907.

Abstract: In this paper, we introduce a Q-learning-based algorithm for Cell Outage Compensation (COC) in Self Organizing Networks (SONs). The algorithm compensates the coverage in the outage area by modifying the power and antenna tilt angle parameters of the neighboring cells. The proposed Q-learning algorithm adapts the reward via learning the consequences of the taken actions to compensate the coverage gap, which guarantees a fully autonomous and accurate COC as we do not assume the knowledge of the propagation model or other models of the environment. This contrasts with existing COC approaches which are inaccurate as they assume the knowledge of the mathematical models of the system and solve the COC problem given such mathematical models. Simulation results show a 92

L. M. D. Da Silva, M. F. Torquato and M. A. C. Fernandes, "Parallel Implementation of Reinforcement Learning Q-Learning Technique for FPGA," in IEEE Access, vol. 7, pp. 2782-2798, 2019, doi: 10.1109/ACCESS.2018.2885950.

Abstract: Q-learning is an off-policy reinforcement learning technique, which has the main advantage of obtaining an optimal policy interacting with an unknown model environment. This paper proposes a parallel fixed-point Q-learning algorithm architecture implemented on field programmable gate arrays (FPGA) focusing on optimizing the system processing time. The convergence results are presented, and the processing time and occupied area were analyzed for different states and actions sizes scenarios and various fixed-point formats. The studies concerning the accuracy of the Q-learning technique response and resolution error associated with a decrease in the number of bits were also carried out for hardware implementation. The architecture implementation details were featured. The entire project was developed using the system generator platform (Xilinx), with a Virtex-6 xc6vcx240t-1ff1156 as the target FPGA.

1.6 Thesis Organization and Structure

The thesis is organized into seven chapters that provide a comprehensive exploration of the topic. Chapter 1 serves as the introduction, providing an overview of the research problem, objectives, and significance. Chapter 2 delves into the evolution of network management and the challenges posed by next-generation networks, setting the context for the study. In Chapter 3, the theoretical framework underlying the research is presented, including key concepts and theories related to reinforcement learning and network management automation. Chapter 4 explores the role of reinforcement learning as a tool for end-to-end cognition in network management

automation. Chapter 5 focuses on the modelling of the Cell Outage Compensation (COC) scenario using MATLAB, discussing the process, implementation, and results. In Chapter 6, the design and implementation of an FPGA-based Q-Learning hardware for COC is presented. Finally, Chapter 7 concludes the thesis by summarizing the key findings, discussing their implications, and suggesting avenues for future research.

By following this organizational structure, the thesis aims to provide a comprehensive analysis of Q-Learning Hardware Acceleration in the context of Cell Outage Compensation (COC) scenarios.

Chapter 2

A New Era for Network Management

This chapter provides a great overview of the development of mobile network generations by illuminating the complexities of network management procedures and delving into the idea of automation in network management along with its taxonomy.

2.1 Evolution of Mobile Networks

In this section, we will discuss the significant and transformative changes in mobile networks, extending from the earliest generations to the most recent ones.

2.1.1 Voice and Low-Volume Data Communications

Different generations of mobile communication systems are distinguished by technological and functional developments. The first generation (1G), which was developed in the 1980s and was focused on providing voice services, was only widely adopted to a limited extent. The Global System for Mobile Communications (GSM), which spearheaded the development of the second generation (2G), was created by the 3GPP and deployed in Europe in the early 1990s, providing voice and low-volume data services. The Universal Mobile Communications System (UMTS), which served as a representative of the third generation (3G), was developed in the late 1990s and early 2000s with the goal of enhancing data services.
The evolution of mobile networks can be summarized as follows: In contrast to 1G analogue systems, 2G GSM offered improved capacity and enabled extensive voice communication through a circuit-switched core network, supporting roaming and providing more capacity compared to 1G analog systems. From 2000 onwards, GPRS, EDGE, and 3GPP Rel. 99 (3G) introduced both circuit-switched voice services and packet-switched data services. A fully IP-based system with a flat, scalable architecture designed for high-throughput mobile broadband services was introduced with 3GPP Rel. 8 and 3GPP Rel. 10 (4G), which represented a significant design

shift. The IP Multimedia Subsystem (IMS), which provided a control architecture based on Session Initiation Protocol (SIP), enabled the packet switching of all 4G services.

Table 2.1: Overview of mobile network generations and 3GPP releases.

Network Generation	Year	First 3GPP release
2G GSM	1992	GSM Phase 1
'2.5G' GPRS	1997	GSM Phase 2+/Rel.97
3G UMTS	2000	Rel.99
'3.5G' HSDPA	2002	Rel.5
LTE	2008	Rel.8
4G LTE-A	2011	Rel.10
'4.5G' LTE-A Pre	2016	Rel.13
5G new radio	2018	Rel.15

2.1.2 Mobile Broadband Communications

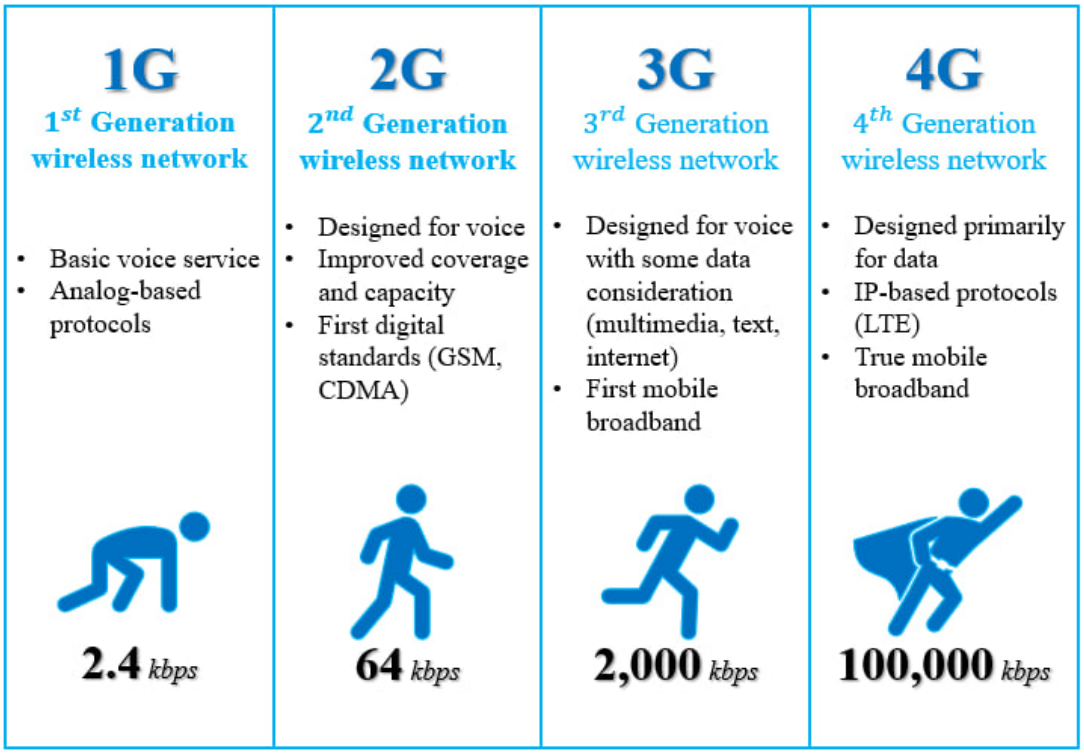


Figure 2.1: The evolution of telecommunication technologies.

The increasing demand for faster data and throughput was the primary driver behind the development of 4G mobile networks. For 4G networks, the International Mobile Telecommunications-Advanced (IMT-Advanced) report, issued by the ITU-R in 2008, established specifications such as a nominal downlink throughput of 100 Mbps for mobile users, improved spectral efficiency, and scalable channel bandwidth.

A high-quality service for a variety of applications, including mobile broadband access, video chat, mobile TV, and new services like high-definition television (HDTV), was the goal of 4G systems. Delivering faster data rates and meeting the requirements of new and advanced services were the main priorities.

2.1.3 Cloud-Native Networks

With a focus on cloud services, network architecture underwent a major review prompted by the introduction of 5G. Traditional network function reference points were supplemented with a service-based interface (SBI) design in the 5G architecture, which broke down the 4G core network functionality into smaller functions.

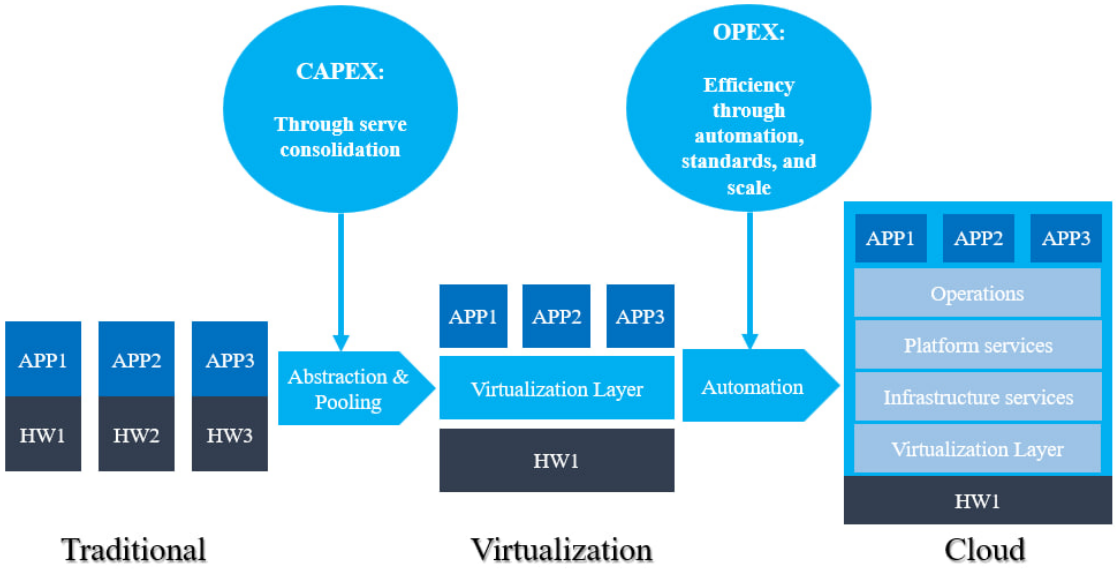


Figure 2.2: Telecommunications networks from traditional to cloud-native.

IP Multimedia Subsystem (IMS): The IP Multimedia Subsystem (IMS) is a standardized architecture for delivering multimedia services over IP networks. It is a framework that enables the integration of various communication services, such as voice, video, messaging, and data, into a single IP-based network. IMS provides the necessary infrastructure and protocols to support these services, ensuring interoperability between different networks and devices.
Session Initiation Protocol (SIP): a signaling protocol used for establishing, modifying, and terminating communication sessions in IP-based networks. It is an application-layer protocol that is widely used for initiating voice and video calls, multimedia conferences, instant messaging, and other real-time communication services over the Internet Protocol (IP) network.
Service-based Interface (SBI): refers to an architectural concept in the context of network function virtualization (NFV) and software-defined networking (SDN). It is a standardized interface that enables communication and interaction between different components or modules within an NFV/SDN framework.

2.1.4 Multi-Service Mobile Communications

The fifth generation of mobile networks plays a crucial role in the digital transformation of both business and daily life, fostering connectivity and enabling a connected society. Key sectors such as education, healthcare, and government services, as well as industries like smart grids, intelligent transport systems, and industrial automation, significantly rely on the capabilities of a flexible mobile network infrastructure.

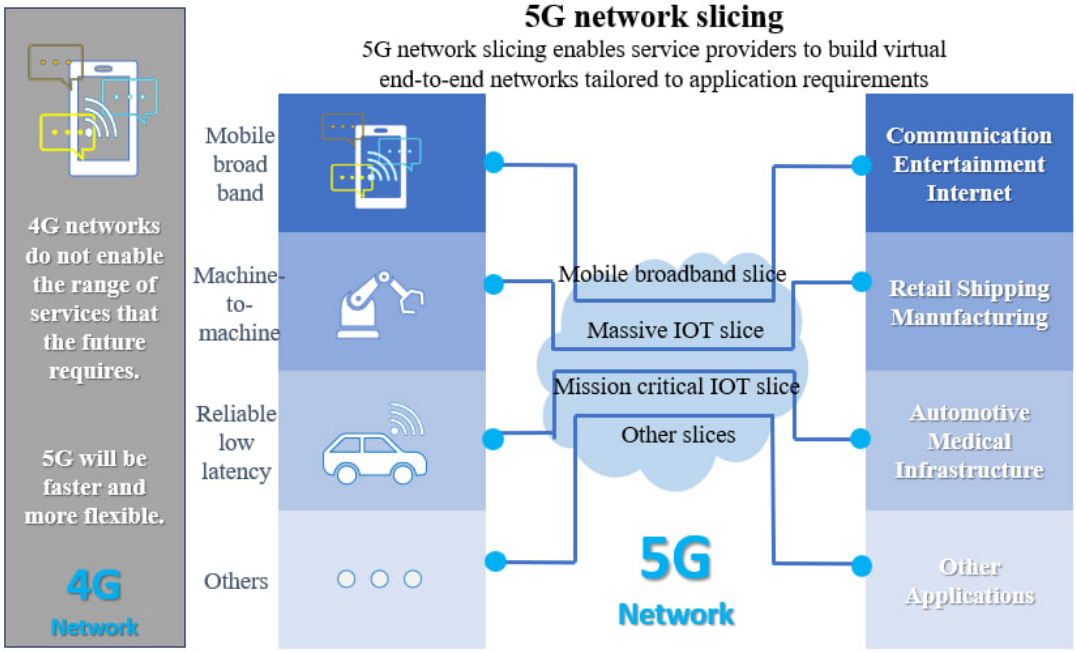


Figure 2.3: 5G Network Slicing.

The characteristics and requirements of mobile traffic become highly diverse and distinct from traditional human-centric traffic as these regions incorporate machine-type devices into the network. In order to provide service-specific functionality, 5G networks must deliver high performance. A platform with a shared infrastructure that can support multiple communication services is required to achieve cost-effectiveness. This is where the concept of "network slicing" becomes crucial since it enables 5G networks to provide multi-service and multi-tenant capable systems.

2.2 Understanding Communication Network Complexity

In a communication network, data is transmitted as packet data from one device (A) to another device (B), which can be mobile or fixed. The network consists of various subparts with devices and network elements (NEs) of different sizes

and capacities. The access part has many small-sized network elements serving a limited number of user terminals, while the core part has a few large-sized network elements supporting thousands of users or sessions. The network's complexity arises from the multitude of network elements and their interdependencies. Managing this complexity falls under network management, requiring innovative approaches to ensure operational functionality. The following sections explore the sources of complexity and their implications for network management.
In general, there are three main sources of complexity in communication networks:

1. **Network protocol stack in each node**: Each node in the network has its own complex interactions and functions. These interactions connect network routing, higher-layer applications, transport dimensioning, network partitioning, and network caching.
2. **Scale**: Scale refers to the number of nodes in the network and the level of interconnections between them. It is a measure of the network's size and density, indicating the complexity of its structure.
3. **Connectivity**: It measures the distribution, centralization, and homogeneity of connections within the network. The connectivity among nodes is essential for maximizing the network's value. Metrics such as node and edge centrality, including edge betweenness, radius, and closeness, are used to describe the complexity of connectivity [1].

These sources of complexity impact network management and require careful consideration and analysis to ensure efficient and effective operation of communication networks.

2.2.1 Enabling Flexibility in 5G Network Management

In order to meet the dynamic requirements of 5G networks, the deployment and configuration of network functions and services need to be flexible. This necessitates effective communication between orchestrating entities and managed entities through multiple interfaces. Efficient network management algorithms, particularly for self-organization and self-optimization, rely on seamless communication between various management functions and network elements. However, the integration of these components is often challenging due to incompatible interfaces.

2.3 Basic Principles of Network Management

Network management plays a crucial role in the overall operations, administration, and maintenance (OAM) of a network. It encompasses a range of functions and

procedures necessary for establishing and maintaining connections within predefined limits. The International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) defines a Telecommunication Management Network (TMN) as the set of functions and procedures used to manage a telecommunication network. ITU-T distinguishes between Telecommunications Managed Areas, which encompass resources involved in one or more telecommunications services and are managed as a whole, and TMN Management Services, which refer to the integrated set of processes within a company to achieve business and management objectives. In order to support these objectives, network management is categorized into five main functional areas known as FCAPS, which stand for Fault, Configuration, Accounting, Performance, and Security. These areas encompass various management services aimed at ensuring quality for telecommunications service customers and operational productivity.

2.3.1 The FCAPS Framework

- **Fault Management**: Includes functions for detecting, isolating, and correcting abnormal operations in the telecommunication network and its environment.
- **Configuration Management**: Involves controlling, identifying, and collecting data from network elements (NEs) and configuring aspects such as configuration files, inventory, and software management.
- **Accounting Management**: Enables the measurement of network service and resource usage, determining costs for the service provider and charges to customers, and supporting pricing decisions for services.
- **Performance Management**: Comprises functions for gathering and analyzing statistical data, evaluating, and reporting on the behavior of telecommunication equipment and network elements, and maintaining overall performance at a defined level.
- **Security Management**: Encompasses functions related to authentication, access control, data confidentiality, data integrity, and non-repudiation to ensure secure communications between systems, customers, and internal users.

2.4 Taxonomy for Cognitive Autonomous Networks

Categorizing cognitive autonomous networks as following:

2.4.1 Automation, Autonomy, Self-Organization, and Cognition

Automation means an entity, E, is configured and triggered to accomplish a task without external intervention while using some functionality for which the mechanism of accomplishing the task is not of interest. Correspondingly, even when the mechanism is created by the external entity, entity E is still considered automated even if the external entity intervenes during execution of that mechanism.
Autonomy implies that the entity acts on its own over multiple related or unrelated tasks and where the mechanisms used to accomplish the tasks are of no interest.
Self-Organization implies that an entity is capable of selecting actions without external control as triggered by a signal which may be external or otherwise. The internal mechanisms of such an entity are of no interest to the owner or user of such an entity.
Cognitive Entity is one capable of perceiving a signal and processing it into a data element over which the entity reasons to select an action. It conceptualizes and contextualizes the data element and, logically or arithmetically, relates the data element to other data elements to make inferences about the elements, their relations, and subsequently selects the appropriate action.

In the context of networks, cognition refers to the network's ability to reason using various data sources to learn context and make optimal decisions. Cognitive autonomy, on the other hand, implies that the network utilizes its cognition to determine appropriate behavior in different contexts and independently acts accordingly.

2.4.2 Levels of Network Automation

Describing the degree of automation in a network is complex due to its multi-dimensional nature.

- **Automation** is typically evaluated relative to a manually controlled network, where operations personnel make decisions based on network state.
- An automated network may have predefined actions set by the operator that are executed without explicit triggering.
- **Self-Organization Network (SON)** automates actions and interprets events in different contexts to determine cause-effect relationships.
- An autonomous network can act independently but may not reason based on its environment or make optimal decisions.
- **Cognitive Network** can reason and make recommendations but may require human approval before executing them.

- SON is closer to an autonomous network, while Cognitive Network Management (CNM) adds cognitive capabilities to SON.
- **Cognitive Autonomous Network (CAN)** combines cognitive and autonomous abilities to reason, recommend, and independently execute actions.
- CAN requires a broader view of the network and a deeper understanding of operator expectations.
- Cognitive functions in CAN need to perceive insights from a wider range of data, learn larger context spaces, and build comprehensive cause-effect models.
- The CAN framework adds a dynamic brain to Self-x functions in the network.

Note: "Self-x" refers to self-organizing, self-optimizing, self-configuring, etc., functions within the network.

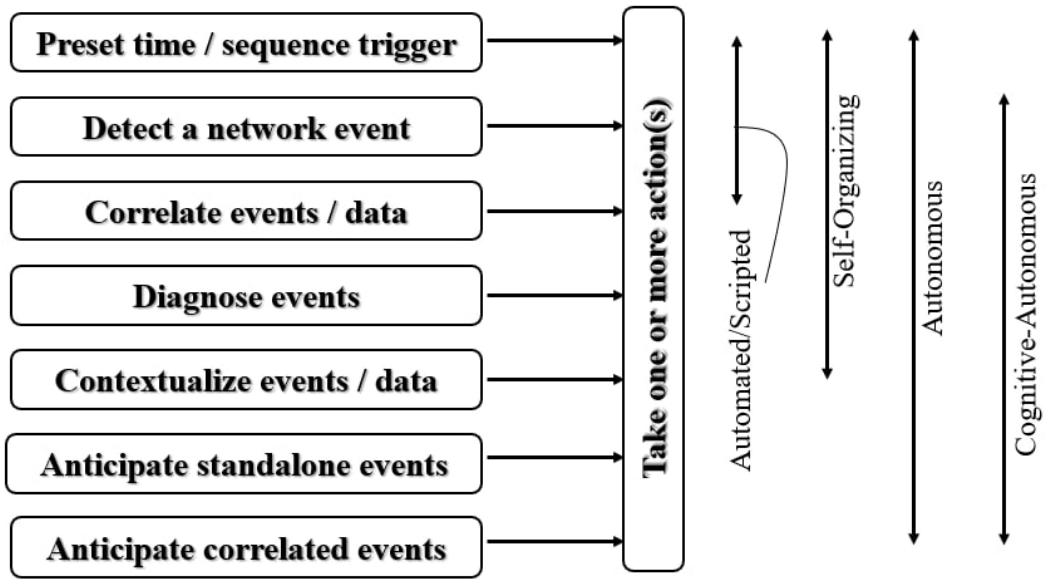


Figure 2.4: Levels of automy in networks.

Chapter 3

Understanding Cell Outage Compensation Technique

To address the rising network management challenges generated by numerous cellular mobile technologies and their varied layers of cells, self-organizing networks (SONs) for mobile networks were proposed. SON strives to reduce this complexity by automating the majority of the time-consuming, repetitive network management operations such as network element (NE) configuration, optimization, and troubleshooting, minimizing manual effort and human error. As a result, SON is the first generation of network management automation, making mobile networks more sensitive to changes in their operational environment, such as new buildings, weather, and mobility, by reducing long human network problem analysis.

In this chapter, we will delve into the concept of self-healing in mobile communication networks, starting with an overview of self-organizing networks. The main objective of this chapter is to gain a comprehensive understanding of our specific use case, which is coverage area optimization. Additionally, we will explore how we can enhance the efficiency of this use case by leveraging hardware acceleration techniques.

3.1 Automating Network Management

The introduction of network automation features is altering network operations. Initially, certain tasks could be automated by scripting and conducted at specific times or when specific events were registered in the network. Then, with the advent of self-organizing capabilities, more complex modules became available, significantly transforming network operating operations.

3.1.1 Traditional Network Operations

Traditional network operation relies on centralized Operations, Administration, and Maintenance (OAM) architecture, with major network management tasks called

FCAPS (Fault/ Configuration/ Accounting/ Performance/ Security management) performed from a central Operational and Maintenance Centre (OMC). These tasks are human-driven and heavily supervised by the human operator. The process of correlating information from various sources, such as Performance Management (PM)/Fault Management (FM)/Configuration Management (CM), consumes significant network operator resources, making it a time-consuming, expensive, error-prone task. The efficiency of these manual processes still depends on the human operator's expertise. [2]

The resource-intensive nature of network management processes, requiring specialized human resources and software tools, and their manual execution, make network planning and optimization difficult to respond to short-term variations in the network environment. To compensate, network operators often over-provide network resources to minimize user experience degradation.

3.1.2 SON Architecture Types

SON (Self-Organizing Networks) was introduced as a solution to manage the increasing complexity of network management and improve the efficiency of traditional manual processes. With closed-loop algorithms, SON automates various network management tasks throughout the lifecycle of a mobile network. By implementing SON-based network operation, resource-intensive and time-consuming tasks can be automated, allowing mobile networks to quickly adapt to changes in network elements and their environment. This agility enables the implementation of new optimization and healing use cases that were not feasible with manual operations, such as real-time coverage outage detection and compensation. SON functions, responsible for automation, are introduced at different network locations to address specific use cases. 3GPP has defined three architectural options for implementing SON functions within the general 3GPP network management architecture.

3.1.2.1 The Key Points of Centralized SON

- Centralized SON involves implementing SON functions in a central entity like a Network Manager or Domain Manager as shown in Figure 3.1.
- It is suitable for SON use cases with a wider network scope, allowing access to information from multiple network elements.
- Centralized SON functions can analyze the performance of multiple NEs, coordinate their requirements, and calculate network parameter re-configurations.
- Standardized KPIs and measurements via the north-bound interface enable the design of centralized SON functions for multi-vendor environments.
- Centralized SON requires transporting performance data from individual NEs to the central entity, usually through aggregated intervals.

- Aggregation sacrifices the short-term dynamics of performance data and increases the response time of SON functions.
- Centralized SON is more suitable for long-term optimizations that don't require immediate response to changes in NE performance and can tolerate data transfer delays.
- Centralized SON benefits from greater compute and storage capacity but introduces a single point of failure.
- Compromised execution of SON functions can impact the performance of a significant portion of the network.
- Careful consideration is needed to balance computational complexity and potential risks associated with centralized SON implementations.

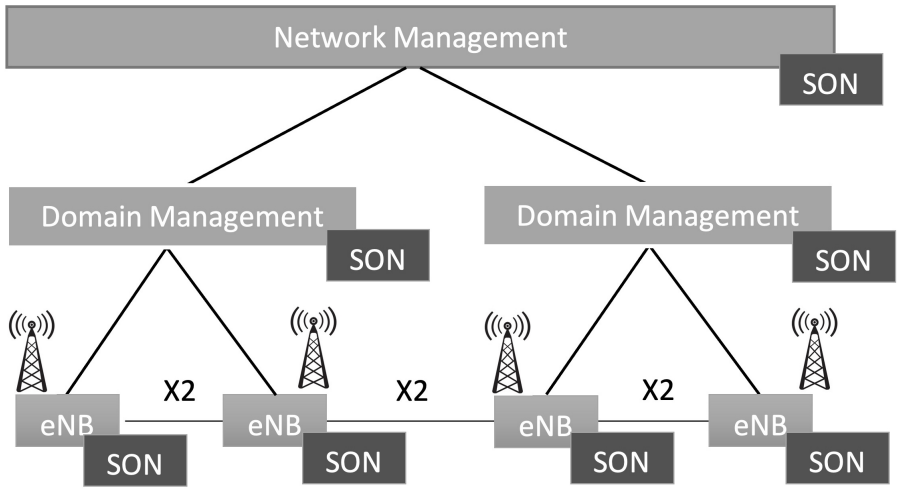


Figure 3.1: Location of SON functions in the 3GPP OAM architecture.

3.1.2.2 The Key Points of Distributed SON

- Distributed SON functions are implemented at the NE level, specifically at the eNB.
- They can utilize information from neighboring NEs through interfaces like X2.
- This implementation option is suitable for SON use cases focusing on a limited scope, such as a single cell or a cluster of neighboring cells.
- By implementing SON function logic close to the NE performance data source, distributed functions can quickly respond to changes in performance indicators and counters.
- Distributed implementations are preferred for SON use cases requiring real-time responsiveness and unable to tolerate data transfer delays or aggregated performance data.

- While most NE parameters are vendor-specific, multi-vendor SON solutions defined by 3GPP can be implemented in a distributed manner.
- 3GPP has standardized data elements and interfaces for exchanging information among neighboring NEs.
- Consideration should be given to the limited compute and storage resources of NEs when implementing distributed SON functions.
- Careful design of SON functions is necessary to manage actions among distributed functions to avoid issues like oscillation, race conditions, and deadlocks.
- Distributed SON functions offer inherent redundancy, preventing a single point of failure in the network.

3.1.2.3 The Key Points of Hybrid SON

- Hybrid SON combines the benefits of centralized and distributed SON implementations.
- Certain SON functionalities are implemented at central entities like NM or DM.
- Other functionalities are implemented at the NE level.
- Centralized functionality primarily handles long-term aspects of network management.
- Distributed functionality focuses on instantaneous and short-term optimization goals.
- Central functions establish policies and boundaries for parameter modifications.
- Distributed functions execute optimization and reconfiguration tasks within those policies and boundaries.
- Hybrid SON achieves a balanced network management strategy by leveraging the strengths of both centralized and distributed approaches.

3.2 Self-Configuration

The deployment and initial configuration of network elements (NEs) require significant resources, including network planning, physical site visits, and configuration efforts. However, with the increasing complexity of HetNet deployments, involving multiple radio access technologies (RATs) and layers of cells, the traditional approach of physical site visits for configuration becomes prohibitively expensive. Automation is crucial during this phase to streamline the deployment process, as recognized by

organizations like NGMN. Self-configuration use cases focus on the pre-operational phase, which spans from the NE's physical deployment to its readiness for live network traffic handling.

Tasks in this phase, also known as 'Plug-and-Play (PnP)', include establishing basic connectivity between the NE and its Operations, Administration, and Maintenance (OAM) system, NE commissioning, and initial radio parameter configuration.

3.3 Self-Optimization

Self-optimization involves optimizing network parameters during the operational stage of a mobile network, while network elements (NEs) are actively handling live network traffic. It is necessary to adjust network configuration parameters to accommodate changes in the network and its environment. Factors such as new constructions, seasonal effects, changes in vegetation, evolving user traffic patterns, and modifications in the network itself can impact the network environment. Additionally, optimization in the operational phase helps overcome limitations in the planning and initial configuration phases, which are based on assumptions about the deployment environment. By analyzing data from various network sources like performance management, fault management, and configuration management, self-optimization algorithms continuously assess the current network performance and propose configuration changes to improve performance in line with network operator objectives. The subsequent sections provide a summary of the essential self-optimization use cases standardized within the LTE-SON framework.

3.4 Self-Healing

Self-Healing also deals with the operational state but focuses on failure detection and recovery, for example, outage detection and NE failure recovery. Due to the distributed nature and large size of mobile networks, FM is a challenging task, which is further accentuated by the fact that in many parts of the network, especially in the radio access network (RAN), there is often little redundancy. Therefore, special SON functions have been created for automating the detection, diagnosis, and recovery of network performance degradation, which are collectively called self-healing SON functions.[2]

Self-Healing is an autonomous property that aims to maintain the system in a healthy state, as opposed to states that are considered faulty or degraded [3]. This is achieved through constant system monitoring, detection of unexpected, degraded states, diagnosis of potential root causes, identification of corrective actions, planning their deployment, and monitoring the outcome of the recovery process. Unlike self-optimization, self-healing functions primarily focus on monitoring network performance degradation symptoms, without prior knowledge of the root causes.

The self-healing concepts and requirements are defined in 3GPP's technical specification (3GPP TS 32.541) [4]. In addition, there are four documents specifying the Information Service (IS) and Solution Sets (SSs) for the SON Policy NRM IRP. 3GPP identifies three main use cases: software faults, hardware faults, and cell outage detection and compensation.

For software faults, the corrective actions include software initialization and restarts on different levels, and if this is not able to solve the problem, re-installing and possibly reverting either to a backup of an earlier software version or an activation of a fall-back software load. Alternatively, software issues can be attempted to be resolved with re-configurations, but this requires a reliable diagnosis of the problem [12]. In the case of faulty hardware, the recovery depends on whether there is a redundant backup unit for the failing resource. If there is no redundancy, the corrective actions can include isolating the faulty resource and trying to circumvent or work around the failure by re-configuring the other working resources. If there is redundancy, then the recovery would be executed with a switchover using the necessary re-configurations.

3.4.1 Next-Generation Self-Healing Networks: Cognitive Autonomy

While self-optimization functions optimize a set of configuration parameters to improve the network performance from a given network state, self-healing functions focus on ensuring that the network can fulfil its purpose and serve its customers even in the case of failures,unexpected changes, or events that risk a degradation in the network performance[5]. In other words, their objective is to make the network more resilient[6, 7]. This section discusses the methods for improving the resiliency of future cognitive autonomous networks. In particular, concepts for cognitive self-healing functions.

3.4.2 Resilience and Self-Healing

Resiliency is the capability of a system to recover to a stable, functioning state after failure or adverse events[6]. By definition, it is not the same as robustness. A robust system is strongly designed to withstand any foreseen problems or failures, but it may be too rigid and fail to survive and adapt in the case of unforeseen circumstances, which are inevitably bound to happen in complex systems. For example, a farmer may prepare his crop against fire, flooding, and local pests, but the crop can be destroyed by a foreign plant virus introduced into the environment. Paradoxically, a very robust system can sometimes be more susceptible to failure due to its increased rigidity and complexity[6]. Modern mobile telephone networks are often said to be among the largest and most complex human-created systems, and their distributed nature makes them even more complex to manage and predict. So, simple robust design principles (redundancy, etc.) are not sufficient to ensure the

ultra-reliable network performance required for many 5G use cases, such as remote surgery.

Making networks more robust and redundant makes them more complex, which, in turn, may create new possibilities for failure. It can also make the problems harder to diagnose when they occur. This also applies to the self-healing functions. The functions may not always be able to diagnose the complex causes and effects that lead to degradation, and the applied corrections can sometimes be more workarounds than really addressing the root cause. Such workarounds remedy the immediate symptoms, but since the root cause has not been resolved, the problem may still appear later. A risk is that at a later, more escalated stage, the problem may be more severe and harder to troubleshoot and correct.

Monitoring the corrective actions performed by self-healing functions is crucial. However, it is important to avoid triggering erroneous corrective actions, which can result in system faults. This scenario is similar to an allergic reaction caused by the human immune system. To prevent false positive triggers, additional verification layers can be implemented, but this adds complexity to the system. Striking the right balance between robustness and complexity is essential when implementing self-healing mechanisms.

3.4.3 Overview on Cognitive Self-Healing

Self-healing represents the ideal state of automation, where a system can recover from any failure. However, achieving such a perfect self-healing system is an immensely complex task. To tackle this complexity, it is necessary to break down the system structure into smaller, simpler components that focus on limited functionality. Gradually, as these simpler components solidify, complexity can be added to enhance their capabilities. This incremental approach allows for the more manageable development of a self-healing system.

3.4.4 The Basic Building Blocks of Self-Healing

The simplest self-healing solutions are rule-based systems, that trigger a specified automated corrective action when a given condition is fulfilled, e.g., by a specified alarm. Such systems only work reliably on anticipated problems and typically underperform in unforeseen circumstances or changing environments. Furthermore, the creation and maintenance of the rule base are expensive and laborious. Rule-based self-healing functions are very rigid, and because of this, rigidity can even make the system less resilient against unexpected changes[8].

The rules governing which corrective actions to trigger and when can be formulated as a classification problem and learned through machine learning. In this context, the network or network element can reside in one of many states, where

each state describes either normal or degraded operation. Degraded states can be connected to one or more corrective workflows. Since anomalous or degraded states are, by definition, rare, the rules are learned on a skewed training dataset. If the system learns to detect degraded states from examples, it may also fail to recognize new, unforeseen problematic states. An additional complication can be the scarcity of such labeled training datasets, a topic that will be discussed later[8].

The implementation of self-healing functions typically involves a four-step process: profiling the normal states of the system, detecting anomalies or deviations from the normal, diagnosing the detected anomalies, and performing corrective actions. This process is illustrated in Figure 3.2. By learning and profiling the normal behavior of the system, any deviations from it, even unexpected ones, can be identified. However, not all deviations indicate degradation, so a diagnosis function is necessary to connect the detected anomalies with possible corrective actions. Moreover, to adapt to changes in network behavior over time, such as trends or seasonal variations in network traffic characteristics, the profiles of normal states should be able to adjust accordingly.

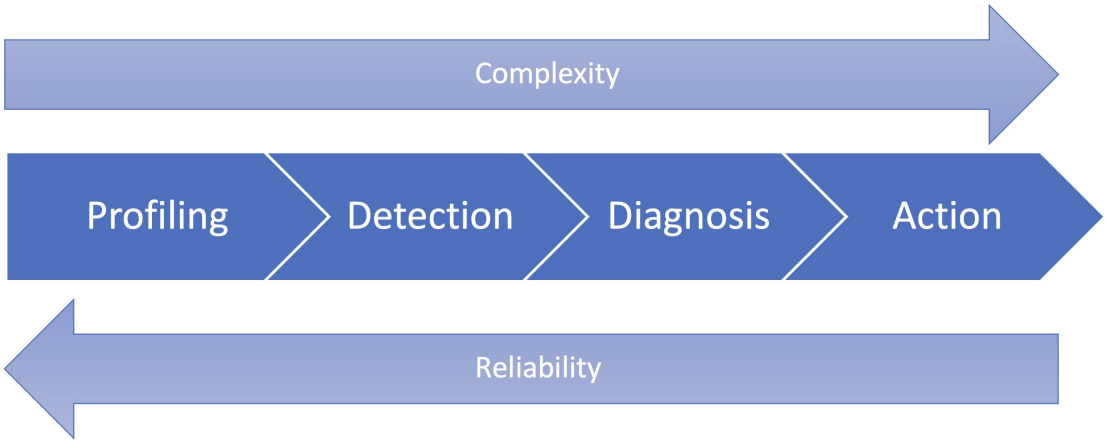


Figure 3.2: Four-step self-healing process.

The four steps in the self-healing process are interdependent, with each stage building upon the results of the previous one. The reliability of the earlier stages becomes crucial for the overall performance of the self-healing process. The initial stages of profiling and anomaly detection are comparatively simpler, while the complexity increases for both training and runtime decision-making. The most complex step is determining the appropriate corrective action to take. This complexity arises from the large problem space and the need for generalization and application of knowledge across similar but different domains or tasks. Achieving this ability is challenging in machine learning and artificial intelligence. As we move further to the right of Figure 3.2, there is likely to be a greater need for human operator involvement.

3.4.5 Self-healing Framework for Mobile Networks

As the number of physical entities in a network increases, the likelihood of network outages, both full and partial, also increases. To address these outages, self-healing solutions typically employ a three-stage framework.

The first stage involves detecting network outages using outage detection algorithms. Effective self-healing requires the ability to detect both full and partial outages. When a network outage is detected, the affected network node is flagged for further action based on the type of outage. For example, if a cell experiences a hardware failure, it will be flagged for self-healing.

Once an outage is detected, diagnostic algorithms are executed to identify the exact cause of the network outage. For instance, in the case of hardware failure, the detection algorithm examines alarms and fault codes to determine the failed hardware component. This information is then communicated to the Network Controller, which can command field teams to replace the failed component or activate redundancy elements as necessary.

After completing the outage diagnosis, the information is passed to the final stage of the self-healing function, which is outage compensation. In this stage, the self-healing function assesses the impact of the outage on neighboring entities and subscribers. This information is used to implement changes that mitigate the outage. For example, if there is a hardware failure, the outage compensation solution identifies coverage gaps and makes adjustments in neighboring cells to provide temporary coverage for affected subscribers. In the case of a partial outage, emergency parameter changes may be executed at the affected cell or its neighbors to recover degraded key performance indicators (KPIs).

The complete self-healing framework, along with relevant studies, is depicted in Figure 3.3. Additionally, Figure 3.4 presents a taxonomy of studies based on these framework components.

3.4.6 Key Components of Self-Healing Techniques

3.4.6.1 Methodology

Studies presenting solutions for the detection, diagnosis, and compensation of outages can be categorized into three main methodologies: heuristic, analytical, and learning based.

- Heuristic solutions rely on pre-defined rules and incorporate intuition or prior knowledge derived from existing literature or experience.

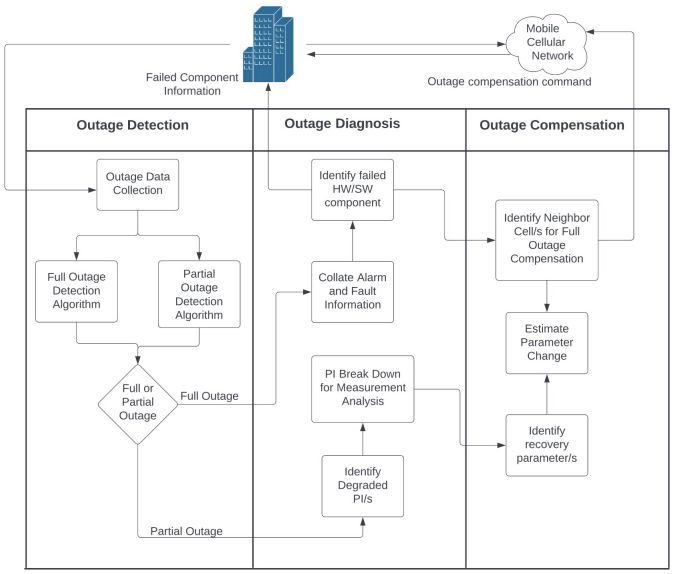


Figure 3.3: Self-Healing Framework.

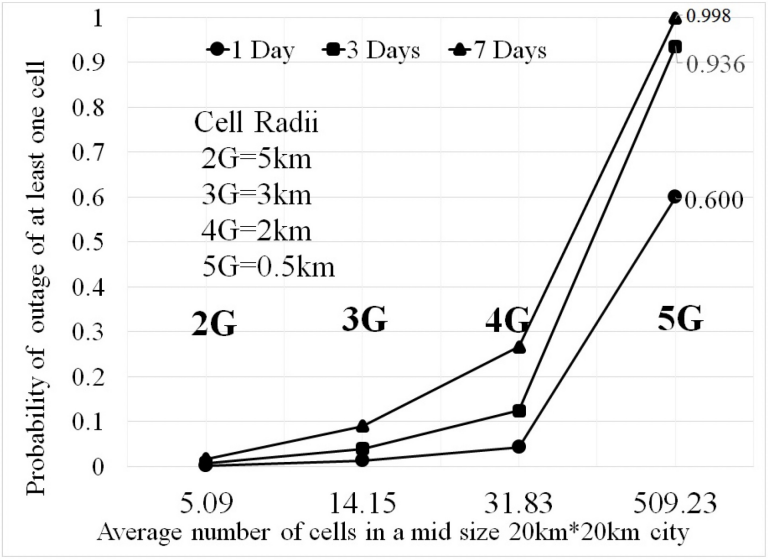


Figure 3.4: Outage Probability of One Cell with Increase in Cell Density

- Analytical solutions involve breaking down a problem into its mathematical components and solving them to obtain optimal or near-optimal solutions.

- Learning-based solutions utilize machine learning techniques, commonly used in computer science, to develop solutions by learning patterns and making predictions based on data.

These three methodologies provide different approaches for addressing outages and offer a range of techniques to improve the self-healing capabilities of networks.

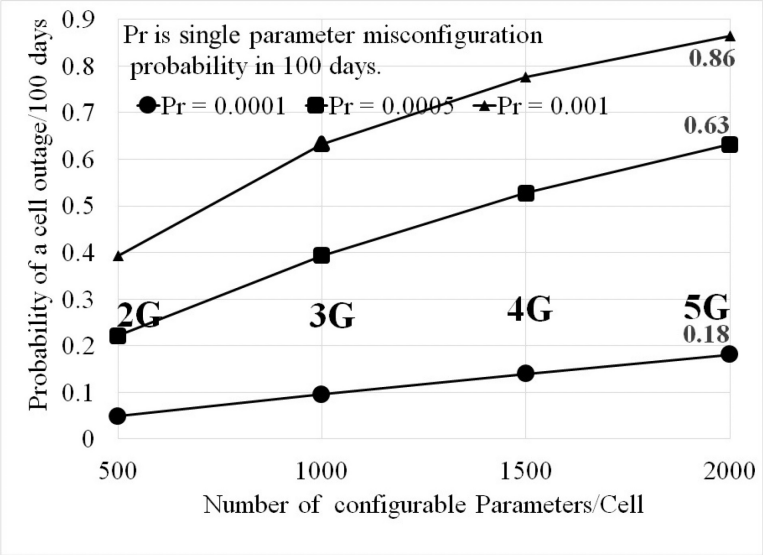


Figure 3.5: Probability of Single Parameter Misconfiguration with Increase in Configurable Parameters

3.4.6.2 Network Topology

Network topology refers to the arrangement or structure of a network, particularly in terms of cell deployments. It describes the hierarchical organization of the network. There are two primary types of network topologies: homogeneous and heterogeneous.

Homogeneous networks consist of a single tier of cells. These cells can either be macrocells, which have broad coverage areas, or small cells with lower power and coverage capabilities. On the other hand, heterogeneous networks, also known as HetNets, comprise a combination of macro and small cells, forming a multi-tier cellular network. This mix of cell types enables enhanced coverage, capacity, and network performance.

While most studies on legacy mobile cellular networks employ homogeneous network topology as the baseline, HetNets are quickly gaining popularity due to their flexibility and their potential to achieve the goals set out for 5G cellular networks [9].

3.4.6.3 Performance Metrics

Network performance metrics are essential for evaluating the performance of a network, and they can be derived from various sources such as network entities and user reports. The selection of performance metrics plays a crucial role in the development and assessment of solutions and algorithms in any study.

In the context of Self-healing, performance metrics related to network health are particularly relevant. These metrics provide insights into the overall state and

well-being of the network, allowing for the identification of issues and the effectiveness of Self-healing mechanisms. By monitoring and analyzing network health metrics, researchers and practitioners can better understand the impact of outages, faults, and other factors on the network's performance and take appropriate actions to improve its resilience and stability.

Network health is a broad term used to describe the performance of the network in terms of universally accepted KPIs such as Accessibility, sustainability, and Mobility [9].

Accessibility refers to the capability of subscribers to connect to the network and access its resources for data transmission. It is assessed through various key performance indicators (KPIs), such as attach success rate, radio resource control setup success rate, connection setup success rate, and random access success rate. These metrics measure the effectiveness of the network in enabling subscribers to establish and maintain connections.

Retainability measures the network's ability to sustain a data session without any disruptions or drops. The session drop rate KPI is used to evaluate retained ability, indicating the rate at which data sessions are prematurely terminated. A lower session drop rate signifies better retention ability and a more reliable network.

Mobility evaluates the network's capability to support seamless subscriber transitions between different cells without significant impact on services. It is typically represented by the handover attempt rate, handover success rate, and handover failure rate KPIs. These metrics indicate how effectively the network manages handovers, ensuring smooth transitions for subscribers as they move between cells while maintaining uninterrupted service quality.

Moreover, network coverage and quality are crucial aspects considered in the design and analysis of Self-healing solutions. Various measurements are utilized to assess these factors. Network coverage is evaluated using metrics such as reference signal received power (RSRP), which indicates the strength of the received signal. Network quality, on the other hand, involves assessing parameters such as spectral efficiency, signal-to-interference, noise ratio (SINR), reference signal received quality (RSRQ), network and user data throughputs, channel quality indicators, and data latency. These measurements provide valuable insights into the performance and efficiency of the network, enabling effective Self-healing strategies to be developed and implemented.

3.4.6.4 Control Mechanism

The control mechanism of a SON solution determines how the functionality of the solution is managed. It can be categorized into three methods: centralized, distributed, and hybrid. In centralized control, the control mechanism of a SON solution determines how the functionality of the solution is managed. It can be

categorized into three methods: centralized, distributed, and hybrid. In centralized control, the SON functions are controlled by a single central controller that is connected to every node in the network. In distributed control, the control of SON functions is decentralized, and each network node can manage its SON functions. Hybrid control is a combination of centralized and distributed control. In this approach, some SON functions are located within a centralized controller, while other less computationally intensive functions that do not directly impact neighboring nodes are distributed among the network nodes.

3.4.6.5 Direction of Control

The direction of control in a SON function determines whether it is focused on optimizing the node-to-user link, the user-to-node link, or both. SON functions that prioritize optimizing the node-to-user link are known as downlink controlled, as they aim to enhance the performance of data transmission from the network node to the user. Conversely, functions that prioritize optimizing the user-to-node link are referred to as uplink controlled, as they aim to improve the performance of data transmission from the user to the network node. There are also SON functions that optimize both the downlink and uplink, providing bidirectional control for enhancing network performance.

3.4.6.6 Taxonomy for Self-Healing Concept

The proposed taxonomy for self-healing in mobile networks is structured as follows:

1. Outage Detection:
    - Full Outage Detection
    - Partial Outage Detection
2. Outage Diagnosis:
    - Full Outage Detection
    - Partial Outage Detection
3. Outage Compensation:
    - Coverage Area Optimizationb.
    - SINR Optimizationc.
    - Cell Capacity Optimization.
    - Spectral Efficiency Optimization

This taxonomy categorizes self-healing functionalities into three main stages: detection, diagnosis, and compensation. The detection stage focuses on identifying network outages and distinguishing between full and partial outages. The diagnosis stage involves determining the specific causes of the outages and differentiating between full and partial cases. Finally, the compensation stage addresses the optimization of various aspects, such as coverage area, SINR, cell capacity, and spectral efficiency, to mitigate the impact of the outages and enhance network performance.

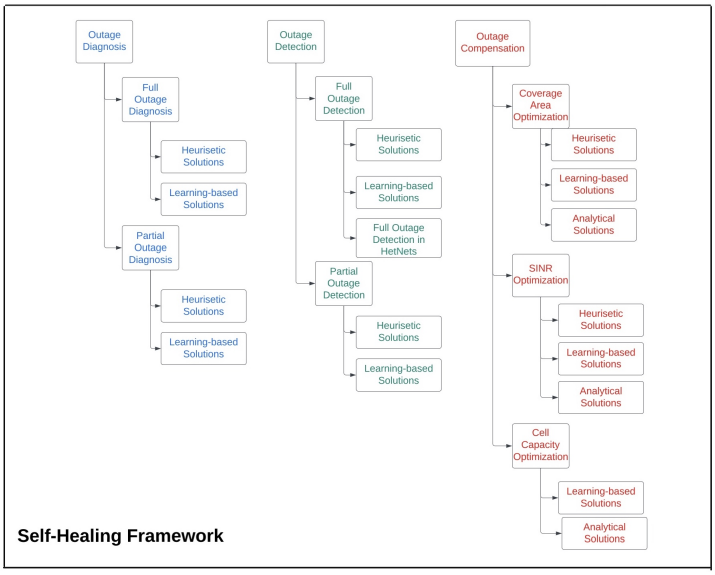


Figure 3.6: Proposed Taxonomy.

3.5 Coverage Area Optimization for Outage Compensation

The component of outage compensation holds a central role in the Self-healing framework, and as a result, it has garnered significant interest and research focus within the academic community. Compensation actions and algorithms are specifically designed to restore service temporarily in the event of a full or partial outage, as both scenarios cannot be immediately resolved. While the methodologies for detecting and diagnosing full outages and partial outages may differ, the compensatory techniques employed for both events exhibit similarities. Although many studies primarily address compensation algorithms for full outages, these solutions can also be applied effectively to address partial outages.

The key principle of outage compensation is to leverage resources from neighboring cells of outage-affected cells to provide temporary services in the affected area. These resources include cell bandwidth and user associations, which can be modified using primary parameters such as cell/user equipment transmit powers and antenna parameters, as well as secondary parameters such as neighbor lists and cell selection parameters [10].

This section focuses on the presentation of coverage area optimization algorithms, which play a crucial role in our case study. The algorithms are described in detail, highlighting the methodology employed. Coverage area optimization is an essential aspect of our work, making this section particularly significant.

Table 3.1 presents a compilation of studies [11–17] that have addressed the issue of network outages and KPI degradations by proposing outage compensation algorithms with a specific focus on coverage optimization. The table provides an overview of these studies and the techniques they have proposed.

Table 3.1: Qualitative Comparison of Cell Outage Detection Algorithms.

Solution	Reference	Methodology	Sub-Method	Network Topology	Performance Metrics	Control Mechanism	Direction of Control
Coverage Optimization	[13]	Heuristic	Framework	Homogeneous	Retainability, Coverage, Quality	Centralized	DL
	[14]				Coverage, Quality		UL/ DL
	[15]				Coverage, Quality		DL
	[16]	Analytical	Non-convex Optimization				UL/ DL
	[17]						
	[11]	Learning Based	Reinforcement Learning		Coverage		DL
	[12]			HetNet			

3.5.1 Choosing the right neighboring cells, optimization parameters, and recovery action:

In the context of outage compensation, three important aspects are discussed: the choice of neighboring cells, optimization parameters, and recovery actions. Several studies have investigated these areas, namely [13–15].

In the study by Asghar et al. [13], a Self-healing framework is proposed, that utilizes received power measurements from users in outage-affected cells to create coverage polygons for neighboring cells. The algorithm iteratively adjusts the antenna configurations of key neighboring cells to ensure coverage constraints are met for all users. By monitoring downlink throughputs and radio link failures, the algorithm benchmarks network recovery. Real network outage demonstrations show that this algorithm effectively compensates for outages within 2 hours.

Amirijoo et al. [14] present an outage compensation framework that compares the potential of different control parameters (such as reference signal power, uplink target received power level P0, and antenna tilt) in mitigating outage-induced performance degradation. Using an iterative algorithm, the parameters of neighboring cells are

updated and their results evaluated. The findings indicate that P0 and antenna tilt are the most effective parameters for improving coverage, while P0 is the most effective for enhancing throughput.

In the work by Frenzel et al. [15], the authors discuss the selection of an optimal recovery action based on three inputs: the probability of effectiveness of a solution (dependent on the outage cause), the network operator's preference for a recovery action, and the operator's preference for a degradation resolution. They propose a weighted-sum function that calculates the cost of selecting a solution, action, and resolution tuple. The framework is adaptable to different network technologies, allowing for the inclusion of additional tuples in future networks. However, determining probabilities and preferences requires manual input from experts. These studies shed light on the significance of choosing the right neighboring cells, optimization parameters, and recovery actions in developing effective outage compensation solutions within the Self-healing framework.

3.5.2 Non-convex Coverage Optimization Techniques for Outage Compensation:

Jiang et al. [16] and Wenjing et al. [17] address the challenge of outage compensation using non-convex optimization methods. They recognize that in large networks with diverse optimization parameters, outage compensation can be a non-convex problem that is computationally complex and difficult to solve optimally. Converting the problem into a convex one would require numerous generalizations and assumptions, which may not be practical for real-world implementation.

Jiang et al. [16] propose a cost function minimization approach for coverage optimization. They use a weighted sum of downlink channel quality and received signal strength as the cost function. The authors apply an immune algorithm, a non-convex optimization technique, to maximize the cost function and determine the optimal uplink target received power (P0) for outage compensation. The immune algorithm demonstrates improved coverage and channel quality after optimization, converging quickly. Compared to other techniques, their methodology significantly enhances coverage by 10% without sacrificing cell edge throughput. However, the immune algorithm's performance is sensitive to the selection of initial parameters, as incorrect settings may result in an infeasible solution.

Similarly, Wenjing et al. [17] centralized the non-convex problem of minimizing coverage holes and pilot pollution by using the downlink pilot powers of neighboring cells for outage compensation. They employ a particle swarm algorithm, another non-convex optimization technique, to tackle this problem. The analysis of the algorithm shows that it is highly efficient in terms of execution time while recovering over 98% of the coverage area in terms of signal strength without significant degradation in link quality. Similar to the immune algorithm, the particle swarm algorithm's convergence is highly dependent on the initialization parameters.

Both studies highlight the effectiveness of non-convex optimization methods for outage compensation, overcoming the limitations of converting the problem into a convex one. The immune algorithm and particle swarm algorithm offers promising results in improving coverage and addressing specific challenges related to outage compensation. However, careful consideration of initialization parameters is necessary to ensure optimal convergence.

3.5.3 Learning-based Coverage Optimization Solutions for Outage Compensation:

Outage Compensation: Learning-based algorithms have shown promise in outage detection and diagnosis, primarily through classification and clustering techniques. However, reinforcement learning emerges as the most effective approach for outage compensation due to its ability to identify optimal strategies based on maximum rewards over a learning period.

Zoha et al. [11] propose a reinforcement learning-based solution for outage compensation within a comprehensive Self-healing framework. Their algorithm utilizes fuzzy logic-based reinforcement learning to adjust antenna tilts and cell transmits powers, aiming to achieve desirable compensated performance in terms of cell coverage. The compensation algorithm iteratively modifies optimization parameters through the exploration of new rewards or exploitation of past rewards. The reinforcement learning database interprets the resulting network state through a fuzzy-logic regulator, determining whether it is better or worse than the previous state and guiding the next step of the algorithm. The authors demonstrate that their solution can enhance post-outage cell edge coverage by 5 dB while restoring the mean data rate to pre-outage levels.

Onireti et al. [12] presents a similar approach to Zoha et al. [11], but tailored for heterogeneous networks. In their solution, they replace the fuzzy logic component with an actor-critic module to facilitate reinforcement learning. The actor-critic module performs exploratory or exploitative actions, such as adjusting antenna tilt or transmitting power of neighboring cells, based on the learned probability of rewards over time. The critic evaluates the reward associated with the action taken and updates past rewards and probabilities accordingly. Comparative analysis with Zoha et al.'s solution shows that this approach improves cell coverage and channel quality, particularly for cell edge users, bringing them closer to pre-outage levels.

Both studies demonstrate the effectiveness of reinforcement learning in outage compensation algorithms. By leveraging rewards and probabilities learned over time, these learning-based solutions adapt optimization parameters to mitigate the impact of outages and enhance network performance, ultimately improving cell coverage and channel quality.

Chapter 4

Reinforcement Learning: A Tool for End-to-End Cognition in Network Management Automation

Reinforcement Learning: A Tool for End-to-End Cognition in Network Management Automation
In this chapter, we will delve into the fundamental concepts of reinforcement learning, providing an overview of its basic definitions. The Q-learning algorithm, a popular approach in reinforcement learning, will be explored. We'll also address the challenges associated with reinforcement learning and highlight the complexities of enhancing its effectiveness. The motivation for the need to accelerate reinforcement learning algorithms will be explored as we go through the chapter. We will discuss the limitations and time-consuming nature of traditional reinforcement learning techniques, highlighting the importance of improving their effectiveness and speed. The training and decision-making capabilities of reinforcement learning agents can be greatly enhanced by utilizing hardware acceleration.

4.1 Reinforcement Learning

Reinforcement learning is the body of theory and techniques for optimal sequential decision-making [18, 19]. It encompasses the challenge of an agent learning by interacting with its environment through trial and error, and evaluative, sequentially delayed feedback [20]. Figure demonstrates how the agent adapts its actions in response to rewards and penalties associated with specific input states of the environment.

4.1.1 Learning Through Exploration

In reinforcement learning, agents explore the environment to acquire knowledge about optimal behavior. For instance, suppose the agent wants to determine the optimal time to deactivate a capacity-enhancing radio cell to minimize energy con-

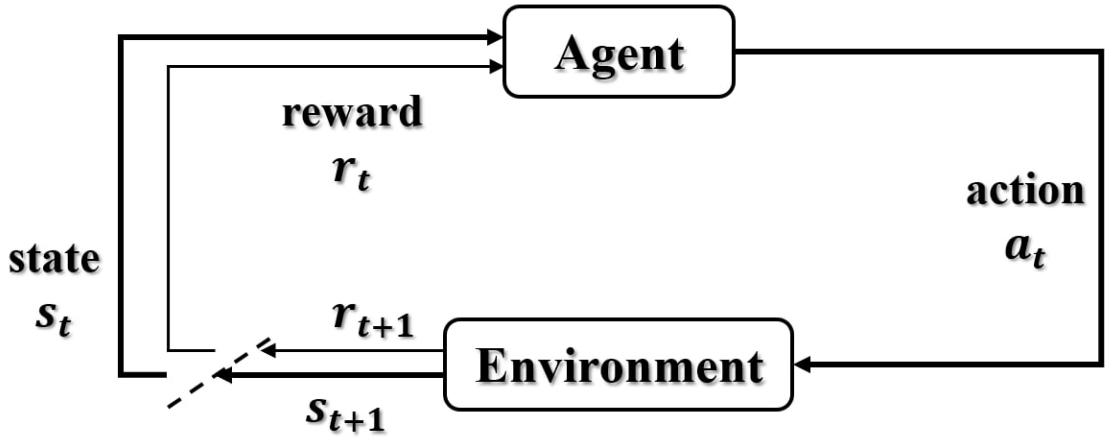


Figure 4.1: Reinforcement Learning Process.

sumption during periods of low traffic. Through exploration, the agent can learn the optimal timing, which can be defined as follows:

Agent: The entity hosting the learning algorithm, responsible for taking actions and learning from them. In the energy-saving example, it is the energy-saving algorithm learning the optimal cell switch-off time.

Environment: The unknown and possibly complex world in which the agent operates. It takes the agent's actions as input, transitions to a new state, and provides rewards. In the cell switch problem, the environment includes the network, cells, active users, etc.

State (s): The concrete situation in which the agent finds itself. It represents a specific instance within the agent's space of interest. For the cell switch problem, states can be the active or inactive states of the cell at a given time.

Actions (a): The set of possible moves the agent can make. Actions can range from simple parameter configurations to complex combinations of changes. In the cell switch example, there are two actions: switching off the cell or keeping it active.

Reward (r): The feedback that measures the quality of the agent's actions. It quantifies the outcome of an action in a given state. Rewards can be discrete (e.g., binary) or continuous, indicating the acceptability of an action. In the cell switch-off example, rewards can indicate the occurrence of unwanted events or their absence.

Policy (π): The strategy used by the agent to determine the next action based on the current state. It maps states to actions, aiming to maximize the agent's cumulative rewards in the long run. Policies can be deterministic or stochastic, with distributions over actions. For example, a policy can predict the mean and deviation of a normal distribution for action selection. RL aims to enable the agent to learn a policy, which is a control strategy for taking actions in the environment. The agent receives feedback about the quality of the different states visited or the quality of the actions taken. The optimal policy, which maximizes the expected return or cumulative, discounted reward, is sought in order to guide the agent's behaviour in the environment.

4.1.2 Cognitive Capabilities and Application of Reinforcement Learning

Reinforcement learning (RL) has the potential for general-purpose artificial intelligence by combining perception and reasoning. It has the ability to recognize different types of data and, depending on the context, make informed decisions. RL can address core challenges in network management automation, such as diagnosing network faults and optimizing network processes. The implementation of RL in communication networks and other critical systems where failure is unacceptable can be challenging and takes time to learn. To mitigate this, RL can be used for optimization solutions within controlled spaces or trained in a simulation environment before applying it to real-world scenarios.

4.1.3 Q-Learning Algorithm

Q-learning is a popular algorithm in reinforcement learning that uses the Bellman equation to find an optimal policy in a Markov decision process (MDP). The Q-learning algorithm iteratively updates a value function called the Q-function. The Q-function estimates the expected cumulative reward for taking a specific action in a given state. It is typically represented as a table, where each entry represents the Q-value for a state-action pair.

The learning process begins with initializing the Q-values for all state-action pairs. Then, the agent interacts with the environment by taking actions and observing the next state and the immediate reward. Based on these observations, the agent updates the Q-values using the following update rule:

$$Q(s,a) = Q(s,a) + \alpha * (r + \gamma * max(Q(s',a')) - Q(s,a))$$

In this equation:

- Q(s, a) is the Q-value for state s and action a.

- α (alpha) is the learning rate, determining the impact of new information on the Q-value update.
- r is the immediate reward received after taking action a in state s.
- γ (gamma) is the discount factor, determining the importance of future rewards.
- max(Q(s', a')) represents the maximum Q-value among all possible actions a'in the next state s'.

The update rule is derived from the Bellman equation, which states that the optimal Q-value for a state-action pair is equal to the immediate reward plus the maximum Q-value of the next state-action pair, discounted by the discount factor. The Q-learning algorithm continues to interact with the environment, updating Q-values based on observed rewards and updating the policy accordingly, until it converges to the optimal policy. Exploration is typically encouraged by using an exploration policy, such as epsilon-greedy, to balance between exploring new actions and exploiting current knowledge. By iteratively updating Q-values based on the Bellman equation, Q-learning enables the agent to learn the optimal policy that maximizes cumulative rewards over time in a given MDP.

4.2 Reinforcement Learning Challenges

1. **Exploration and Exploitation**: Agents face the challenge of balancing exploration of the environment to gather new information and exploitation of learned knowledge to maximize rewards. Finding the right balance between exploration and exploitation is crucial for efficient learning.
2. **Credit Assignment**: In reinforcement learning, the consequences of an action may not be immediately observable. The credit assignment problem refers to the challenge of attributing credit or blame to actions taken in the past that may have influenced the current rewards. This problem becomes more complex when there are long time delays or indirect causal relationships between actions and rewards.
3. **Partial Observability**: In many real-world scenarios, agents have limited or incomplete information about the environment. This partial observability makes it challenging to accurately estimate the state of the environment and can lead to suboptimal decision-making. Techniques such as Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) are used to handle such situations.
4. **High-Dimensional State and Action Spaces**: In complex environments, the number of possible states and actions can be extremely large, resulting in a high-dimensional state and action space. This poses challenges for exploration, policy representation, and computation. Dimensionality reduction techniques and function approximation methods are often used to address this challenge.

5. **Sample Efficiency**: Reinforcement learning algorithms typically require a large number of interactions with the environment to learn optimal policies. This sample inefficiency can be a limiting factor in real-world applications where interactions are costly or time-consuming. Developing algorithms that can learn efficiently from limited samples is an ongoing challenge.

6. **Transfer Learning**: Reinforcement learning algorithms often struggle to generalize knowledge learned in one environment to new, unseen environments. Transfer learning aims to leverage knowledge from previously learned tasks to improve learning efficiency and performance in new tasks. Developing effective transfer learning techniques in reinforcement learning is an active area of research.

7. **Safety and Ethical Considerations**: In certain domains, reinforcement learning agents may have real-world impacts and consequences. Ensuring safety and addressing ethical concerns such as fairness, accountability, and transparency in decision-making are important challenges that need to be addressed to deploy reinforcement learning agents responsibly. Overall, addressing these challenges is crucial for advancing the field of reinforcement learning and enabling its successful application in a wide range of domains.

4.3 Accelerating Reinforcement Learning: The Need for Hardware Acceleration

Reinforcement learning (RL) hardware acceleration refers to the use of specialized hardware devices or techniques to enhance the computational efficiency and performance of RL algorithms. There are several reasons why RL hardware acceleration is beneficial:

1. **Improved Computational Efficiency**: RL algorithms can be computationally demanding, especially when dealing with large state and action spaces or complex environments. Hardware acceleration techniques such as graphical processing units (GPUs), field-programmable gate arrays (FPGAs), or application-specific integrated circuits (ASICs) can significantly speed up the execution of RL algorithms, allowing for faster learning and decision-making.

2. **Real-Time Decision-Making**: In certain applications, RL agents need to make decisions in real-time or near real-time. Hardware acceleration enables faster processing and inference, reducing the latency between perceiving the environment, selecting actions, and receiving feedback. This is crucial in time-sensitive domains such as robotics, autonomous vehicles, or online recommender systems.

3. **Scalability**: RL algorithms often involve training and evaluating models on large datasets. Hardware acceleration techniques can handle larger datasets

and enable parallel processing, making it easier to scale RL algorithms to handle complex tasks and large-scale environments.

4. **Energy Efficiency**: RL algorithms can consume significant computational resources, leading to high energy consumption. Hardware acceleration techniques can optimize the energy efficiency of RL computations by utilizing specialized hardware architectures that are designed for efficient parallel processing or by offloading computations to dedicated hardware accelerators.

5. **Customization and Flexibility**: Hardware acceleration allows for the design and implementation of custom architectures tailored to specific RL algorithms or problem domains. This flexibility enables researchers and practitioners to explore novel algorithmic techniques and optimize hardware resources for better performance.

6. **Deployment in Edge Devices**: With the growing demand for edge computing and the deployment of intelligent agents on resource-constrained devices, RL hardware acceleration becomes crucial. By offloading computationally intensive RL tasks to dedicated hardware accelerators embedded in edge devices, RL agents can operate efficiently and autonomously in edge computing scenarios.

Overall, RL hardware acceleration offers significant advantages in terms of computational efficiency, real-time decision-making, scalability, energy efficiency, customization, and edge device deployment. By leveraging specialized hardware, RL algorithms can be accelerated, enabling faster learning, more efficient inference, and the deployment of intelligent agents in various real-world applications.

Chapter 5

Modelling Cell Outage Compensation (COC) Scenario Using MATLAB

5.1 Need for Modelling

5.1.1 Modelling Definition

Modeling definition:

System Modeling involves creating representations of real-world systems using mathematical equations, computer simulations, physical prototypes, or conceptual frameworks. These models capture the essential components, relationships, and behavior of the system being analyzed. The goal of engineering system modeling is to enhance understanding, predict system performance, identify potential issues, and optimize system design and operation.

Purpose and importance of modeling:

1. Understanding system behavior: Modeling provides powerful tools to understand the behavior of complex systems. By developing mathematical or computational models, we can capture the fundamental principles and interactions governing the system's operation. Models help identify critical variables, quantify system responses, and reveal the underlying dynamics, facilitating a comprehensive understanding of the system under investigation.

2. Prediction and Optimization: One of the primary objectives of system modeling is to predict system performance and optimize design parameters. Models allow us to simulate various scenarios, analyze the impact of different inputs, and forecast system behavior under different operating conditions. This predictive capability aids in making informed decisions, optimizing system parameters, and minimizing risks during the design, development, and operation phases.

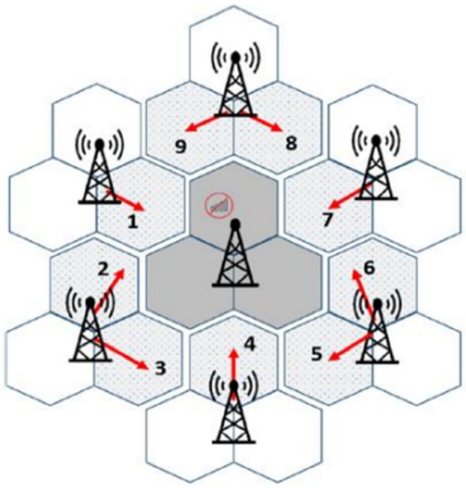


Figure 5.1: Caption

3. System Integrations and interactions: Systems are often composed of multiple interconnected components. Modeling enables us to understand and optimize the interactions between these components, ensuring effective system integration. By developing models that capture the coupling and interdependencies, engineers can identify potential bottlenecks, optimize component interfaces, and enhance overall system performance.

4. Cost and Resource Optimization: System modeling is invaluable in minimizing costs and optimizing resource utilization. Models facilitate the evaluation of different design alternatives, enabling engineers to identify the most efficient and cost-effective solutions. By simulating and analyzing the behavior of the system under various conditions, we can identify areas of improvement, optimize material usage, reduce energy consumption, and streamline manufacturing processes.

5.1.2 Why we need to model COC algorithm?

1. Complexity and system understanding: Engineering systems, such as power grids, transportation networks, manufacturing processes, and cellular networks often exhibit high levels of complexity. Understanding and predicting the behavior of such systems without modeling would be challenging and error-prone. By developing accurate and validated models, engineers can gain insights into the complex interactions, identify system vulnerabilities, and design effective control strategies.

2. Risk Mitigation and Safety Assessment: Engineering system modeling is crucial for assessing risks and ensuring safety in various industries. By simulating system responses to potential failures or disturbances, engineers can identify critical points of failure, evaluate safety measures, and develop contingency

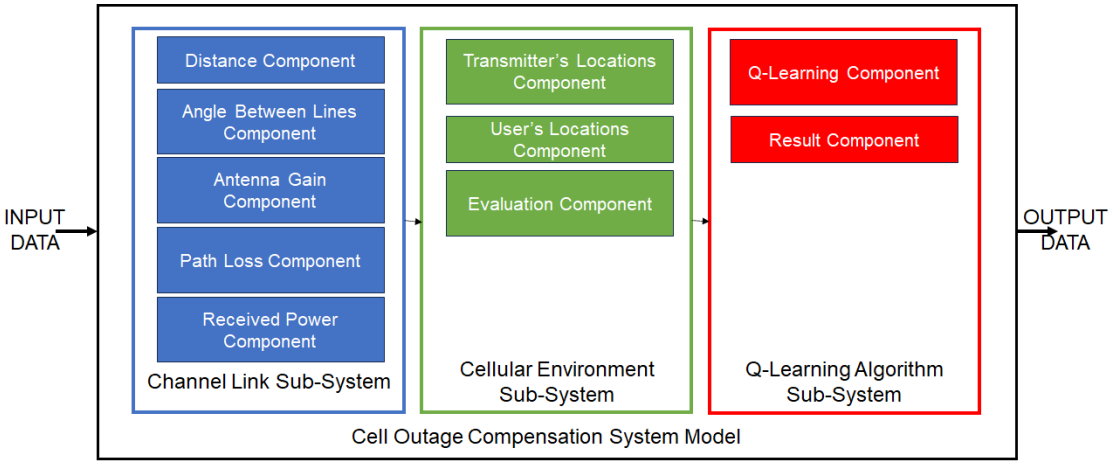


Figure 5.2: Cell Outage Compensation System Model

plans. Modeling allows for virtual testing and validation, minimizing the need for costly physical prototypes and ensuring that safety standards are met.

3. Design Optimization and Iterative Improvement: Engineering system modeling enables engineers to iteratively refine system designs, optimize performance, and address design limitations. By simulating different design configurations, engineers can identify design flaws, test modifications, and optimize system parameters. Modeling facilitates a cost-effective and efficient design process, reducing the need for expensive trial-and-error approaches.

5.2 Proposed Model Architecture

5.2.1 Overview on Proposed System-Level of The Model

The model we used to solve this problem is a simulation of a cluster that consist of seven base stations. The center site failed and other base stations are trying to compensate for this failure and restore coverage to the users in the coverage area of the failed base stations. To achieve this we used a model that is based on 3GPP model. This model consist of three subsystems. The subsystems are RF subsystem, Cellular subsystem, RL subsystem. This model simulate the cellular environment and whether there is a coverage or a connection to the uncovered users. This model calculate the received power for each user based on the 3GPP path loss model and check whether the received is high enough to make a connection with any of the base stations. This environment is used by a Q-Learning Agent to change the transmitted power and the tilt of each base stations to recover the coverage area of the base station.

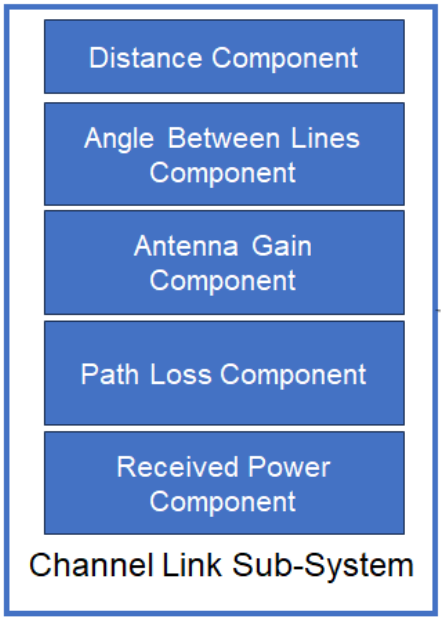


Figure 5.3: Channel Link Sub-System

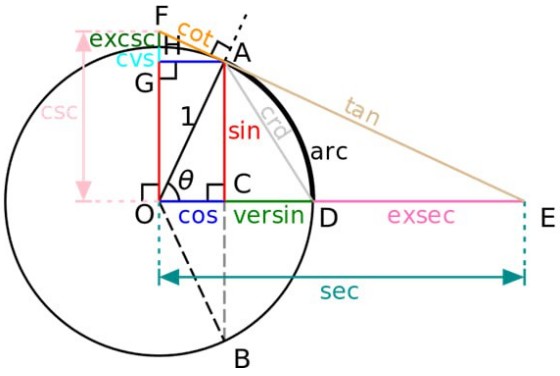


Figure 5.4: haversine formula

5.2.2 Channel Link Sub-System

This subsystem is used calculate the antenna gain and the path loss and thus calculate the received power at any point given their latitude and longitude. This subsystem consist of components that are going to be explained. First we had to determine the antenna parameter that are going to be used to calculate the antenna gain. The antenna's horizontal half power beam width (HPBW) is 120 degrees, and the vertical HPBW 10 degrees. The horizontal Side Lobe Level (SLL) is -20 dB and the vertical SLL is -18 dB. The transmitter's antenna height is 40 meters. The receiver's antenna height is 2 meters. The receiver's antenna gain is 5 dBi. The user equipment (UE) sensitivity is -70 dB.

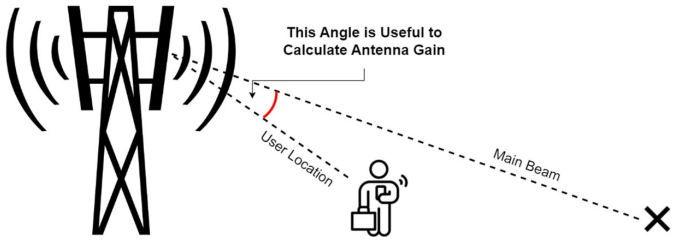


Figure 5.5: Angle Between Lines

5.2.2.1 Distance Component

This is used to calculate the distance between two points on a sphere given their latitudes and longitudes using the Haversine formula. The Haversine formula works by calculating the haversine of theta, which is defined as:

$$a = \text{hav}(\Theta) = \sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) - \cos(\phi_2)\cos(\phi_1)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)$$

where: - Θ is the central angle between two points in radians, - ϕ represents latitude in radians, and - λ represents longitude in radians.

By calculating this, we can obtain the angle Θ using the formula:

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

Then, we can find the distance by multiplying c with the Earth's radius:

$$D = c \cdot R$$

where R is the Earth's radius.

5.2.2.2 Angle Between Lines Component

This is used to determine the angle between two lines given their latitudes and longitudes. First we convert all latitudes and longitudes to radians and then calculate the bearings from point b to point 1 and from point b to point 2. Then we calculate the angle between the two lines and the result will be in radians and convert it to degrees.

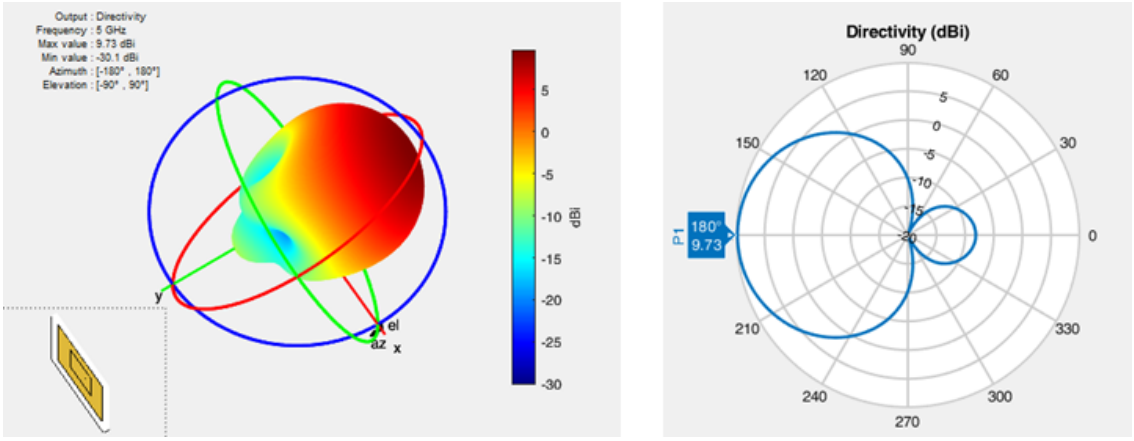


Figure 5.6: Antenna Radiation Pattern in 3D

5.2.2.3 Antenna Gain Component

This is used to calculate the total gain from the transmitter's antenna to the any point. First we calculate the horizontal gain by finding the angle between the receiver's location and the direction for maximum gain. From this angle we can calculate the horizontal gain component at the receiver. After that we can calculate the vertical gain component by calculating theta which the angle between the line that connects the receiver and the base station and the height difference between transmitter's and receiver's antenna. From those components we can calculate the total gain in .

$$G_{H,ij} = \max\left(-12\left(\frac{\phi_{ij}}{HPBW_H}\right)^2, SLL_H\right)$$

$$G_{V,ij} = \max\left(-12\left(\frac{\theta_{ij} - \theta_{tilt,i}}{HPBW_v}\right)^2, SLL_v\right)$$

$$\theta_{ij} = \arctan\left(\frac{h_{te} - h_{re}}{d_{ij}}\right)$$

$$G_{t,ij} = \max(G_{H,ij} + G_{V,ij}, SLL_0) + G_0$$

5.2.2.4 Path Loss Component

In this component we calculate the path loss based on 3GPP 3D channel model for 5G. This is calculated based on a propagation model which takes the transmitter's and receiver's antenna height (meters), the operating frequency (GHz), and the

distance between transmitter and receiver (meters) and the scenario of operation (urban, suburban, rural, or open area). This component return the path loss in dB.

$$PL\_urban = 28 + 22\log(d) + 20\log(f) + \frac{10\log(h_t \cdot h_r)}{1.5^2} - 3.2(\log(11.75h_r))^2 - 0.5$$

$$PL_{suburban} = 28 + 40\log(d) + 20\log(f) + \frac{10\log(h_t \cdot h_r)}{1.5^2} - 9(\log(h_r^2/3.5))^2 - 0.5$$

$$PL_{rural} = 28 + 20\log(d) + 20\log(f) + \frac{10\log(h_t \cdot h_r)}{1.5^2} - 2(\log(h_r/3))^2 - 0.5$$

$$PL_{open} = 32.4 + 20\log(d) + 20\log(f) + \frac{10\log(h_t \cdot h_r)}{1.5^2}$$

5.2.2.5 Received Power Component

This component estimates the received power at the receiver using previous components. The received power is given by the formula:

$$P_r = P_t + G_t + G_r - \mathrm{PL}$$

where: - P_r represents the received power, - P_t is the transmitter's power, - G_t is the total gain of the transmitter's antenna, - G_r is the gain of the receiver's antenna, and - PL represents the path loss.

5.2.3 Cellular Environment Sub-System

This subsystem is responsible for assigning the location of the base stations and the direction of each antenna in the sectors. It is also responsible for the locations of the users on the coverage area of the central site.

First we determine the central site location and the angle of each sector. After this the subsystem consist of the following components

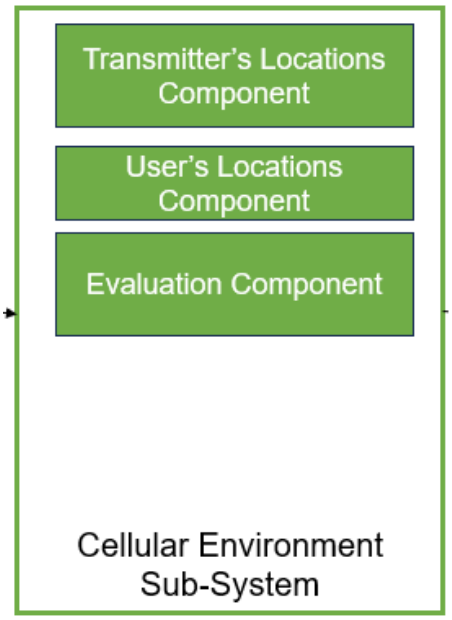


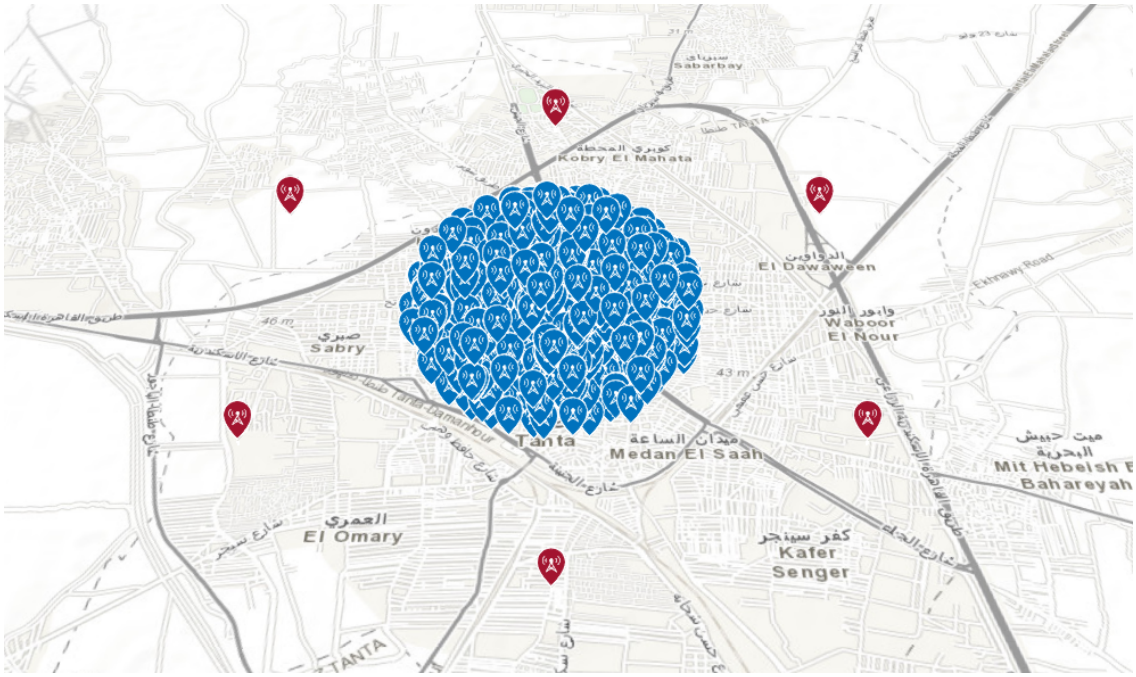
Figure 5.7: Cellular Environment Sub-System

5.2.3.1 Base stations Location Component

This function is used to assign the location of the base stations based on the inter-site distance and the angle that makes with the central site. Each base stations is 60 degrees apart from each other and all base stations are separated by the inter-site distance.

5.2.3.2 Users Locations Component

This function works by determining the location of the users based on the location of the central site. First we generate random angles and distances for the users and use these values to get the latitudes and longitude of each user.



5.2.3.3 Evaluation Component

This function is used to evaluate the performance of the system. This evaluation is using two parameter Agap, Aoverlap. Agap is a parameter that is used to describe the percentage of uncovered users to the total number of users. Aoverlap is a parameter that is used to describe if the user is covered by more than 2 base stations. In the compensation process we choose only three base stations. These base stations has the main lobe of sector's antenna is directed toward the center site. To determine the evaluation parameters we measure the received power form each base station to each user if the received power is less than the receiver's sensitivity this means the user isn't covered by any base stations and it is counted in the Agap and if the measured power from more than 2 base stations is more than the receiver's sensitivity it is counted in the Aoverlap.

5.2.4 Q-Learning Sub-System

This subsystem is responsible for the Q-learning process. Firstly, we select several essential parameters for the learning process. One of these parameters is the learning rate, which determines the rate at which the algorithm updates the values and thus affects the pace of the learning process. Another parameter is epsilon, which controls the balance between exploitation and exploration. Additionally, we have the discount factor, which is used in the Bellman equation to determine how much the agent cares about future rewards.

Once the parameters are chosen, we define the actions and states. In our problem, the agent controls the transmitted power and the tilt of the antennas for

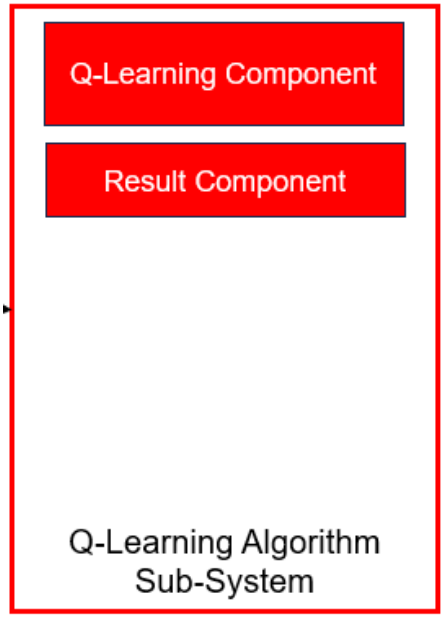


Figure 5.8: Q-Learning Sub-System

Before Learning

0.9913	0.5010	0.1669	0.6838	0.9052	0.6727	0.7413	0.3999	0.4853
0.8105	0.1069	0.5316	0.5458	0.7826	0.9190	0.9997	0.5136	0.0685
0.2480	0.3279	0.2013	0.7491	0.3269	0.5617	0.5490	0.7933	0.5179
0.2663	0.6567	0.1572	0.6434	0.2017	0.2014	0.5931	0.3954	0.4062
0.1055	0.8768	0.7569	0.1459	0.6550	0.2551	0.1544	0.1607	0.5817
0.2897	0.7621	0.9147	0.8380	0.1089	0.6222	0.6367	0.5695	0.9664
0.1613	0.8119	0.2198	0.1301	0.8860	0.3760	0.5461	0.5972	0.9626
0.7347	0.4783	0.1687	0.6866	0.5873	0.3590	0.2844	0.8209	0.9984
0.4427	0.2848	0.7992	0.9403	0.7813	0.0287	0.6577	0.9934	0.5135
0.7415	0.4535	0.6114	0.4554	0.5966	0.7674	0.9942	0.3426	0.7220
0.9992	0.2873	0.7735	0.1114	0.1745	0.2374	0.5099	0.0006	0.1458
0.8247	0.3574	0.5539	0.0616	0.0805	0.6659	0.2052	0.0630	0.1868
0.5769	0.6131	0.2936	0.7762	0.8288	0.7485	0.6324	0.4660	0.2377
0.4926	0.4145	0.5906	0.0027	0.1535	0.7284	0.6831	0.9872	0.9196
0.2337	0.9214	0.1637	0.6068	0.0615	0.8143	0.4551	0.0395	0.0817
0.1537	0.3211	0.3714	0.3569	0.5899	0.7401	0.0844	0.3666	0.6959
0.7182	0.7510	0.9020	0.3946	0.4656	0.6445	0.9905	0.1700	0.1887
0.6006	0.2054	0.4105	0.3966	0.4624	0.0990	0.0393	0.8667	0.8171
0.5142	0.3037	0.8301	0.4193	0.9093	0.9994	0.7413	0.6382	0.3244
0.8416	0.2687	0.9734	0.8270	0.1594	0.4181	0.5869	0.8871	0.8004
0.2837	0.6931	0.5610	0.7261	0.7940	0.0760	0.9566	0.7649	0.8159

Figure 5.9: Q-table Before Learning

compensation. The action space consists of combinations of increasing, decreasing, or maintaining constant power and tilt. This gives us a total of nine actions. For the states, we choose seven states based on the Agap parameter, which represents the gap between the desired and actual performance. These states include Agap = 0%, Agap < 20%, Agap < 40%, Agap < 60%, Agap < 80%, Agap < 100%, and Agap = 100%.

This subsystem consists of the following two components:

5.2.4.1 Q-Learning Component

First, we evaluate the system to determine its initial state. Then, we initiate the learning process by initializing the Q-table with zeros. Next, we employ an ϵ-greedy policy for action selection. Under this policy, the agent generates a random value between 0 and 1. If this value is smaller than ϵ, the agent explores by randomly

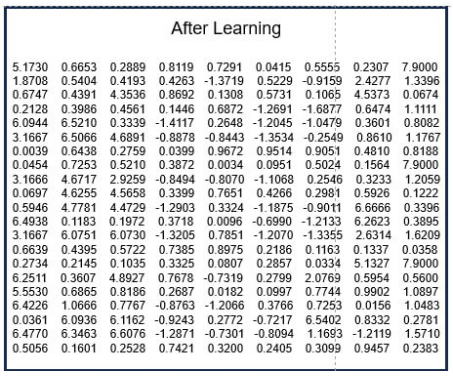


Figure 5.10: Q-table After Learning

choosing an action from the action space. Conversely, if the value is larger than ϵ, the agent exploits by selecting the best action that maximizes the reward. After each action, the agent evaluates the system and updates the state. Based on the new state, the environment provides a reward. Using this reward, the agent updates the Q-table using the Bellman function. At the end of each episode, the agent evaluates the system to check if a termination state has been reached (Agap = 100%). If the agent reaches a termination state, the ϵ value is set to 1, causing the system to stop exploring. If a termination state is not reached, the agent repeats the process.

5.2.4.2 Result Component

The result component is responsible for testing the final policy of the agent, and displaying the results. It display how the system parameters changes with the learning process. First it display how Agap is changing with each step and how the power and the tilt of each base station is changing until it reaches the final value.

5.3 Model Implementation

The model implementation involved designing and developing a system to address the cell outage compensation (COC) problem in a wireless communication network. The model aimed to improve the reliability and performance of the network by dynamically adjusting the transmit power and tilt of antennas to compensate for cell outages.

The implementation process consisted of several key steps. Firstly, a comprehensive analysis of the network architecture and COC requirements was conducted to understand the system's behavior and identify the necessary functionalities. Next, a mathematical or computational model was developed to capture the fundamental principles and interactions of the system.

The model implementation involved writing code in a programming language

such as MATLAB or Python to simulate and test the behavior of the network under various conditions. This included calculating the received power at the receiver, evaluating performance metrics, and making decisions based on the Q-learning algorithm.

During the implementation, various parameters were chosen and optimized to ensure effective learning and decision-making. These parameters included the learning rate, exploration-exploitation trade-off, and discount factor, which controlled the pace of the learning process and the agent's consideration of future rewards.

The model implementation also involved integrating the COC functionality into the existing network management system. This required developing interfaces and communication protocols to enable seamless interaction between the COC module and other network components.

Throughout the implementation, extensive testing and validation were performed to ensure the accuracy, efficiency, and reliability of the model. This involved running simulations, analyzing results, and comparing them with expected outcomes or real-world data.

In conclusion, the model implementation successfully addressed the cell outage compensation problem by incorporating the Q-learning algorithm and dynamic adjustment of transmit power and tilt. The implemented model provided a reliable and efficient solution for optimizing network performance and ensuring uninterrupted service in the face of cell outages.

5.3.1 Programming Language and Environment:

5.3.1.1 Overview on MATLAB

MATLAB is a high-level programming language and environment designed for numerical computation, data analysis, visualization, and algorithm development. It stands for "MATrix LABoratory" and was developed by MathWorks. With MATLAB, users can perform a wide range of tasks, from basic calculations and data manipulation to advanced algorithm development and simulation. It provides a rich set of built-in functions and toolboxes, making it a powerful tool for scientific and engineering applications. One of the key features of MATLAB is its extensive support for matrix operations. MATLAB treats many data types, including scalars, vectors, matrices, and even higher-dimensional arrays, as matrices by default. This makes it convenient for performing mathematical operations and solving systems of equations.

MATLAB's syntax is relatively easy to understand and read, resembling traditional mathematical notation. It offers a command-line interface, known as the MATLAB Command Window, where users can enter commands and receive immedi-

Figure 5.11: MATLAB Logo

ate results. Additionally, MATLAB provides a graphical user interface (GUI) known as the MATLAB Desktop, which offers an integrated development environment (IDE) for writing, debugging, and running MATLAB code. MATLAB's functionality can be extended through the use of toolboxes. MathWorks offers a variety of toolboxes that cover diverse areas such as signal processing, image processing, control systems, optimization, machine learning, and more. These toolboxes provide additional functions, algorithms, and specialized features to enhance MATLAB's capabilities in specific domains. Furthermore, MATLAB supports the creation of graphical visualizations, enabling users to generate plots, charts, and 3D visualizations to explore and present their data. It also supports the creation of user interfaces, allowing users to build interactive applications and GUIs for their MATLAB programs.

MATLAB is widely used in academia, research institutions, and industries such as engineering, physics, finance, data analysis, and machine learning. Its versatility, extensive functionality, and user-friendly nature make it a popular choice for professionals and researchers who require a powerful computational tool for their numerical and scientific computing needs.

In summary, MATLAB is a comprehensive programming language and environment that provides a wide range of tools and functions for numerical computation, data analysis, and algorithm development. It enables users to perform complex mathematical operations, visualize data, and build sophisticated applications efficiently.

5.3.1.2 Libraries: Antenna Toolbox

The Antenna Toolbox is a specialized toolbox in MATLAB developed by MathWorks that focuses on antenna design, analysis, and visualization. It provides a comprehensive set of functions, algorithms, and tools for engineers and researchers working in the field of electromagnetic wave propagation and antenna engineering. The Antenna Toolbox within MATLAB offers a range of capabilities for designing and analyzing various types of antennas, including wire antennas, patch antennas, reflector antennas, array antennas, and more. It provides an intuitive graphical user interface (GUI) for creating, modifying, and visualizing antenna geometries. Key features of the Antenna Toolbox include:

1. Antenna Modeling: The toolbox offers functions for modeling and creating antenna structures with customizable parameters such as dimensions, materials, and operating frequencies. Users can define antennas using basic shapes, import existing CAD files, or create custom antenna geometries.

2. Radiation Patterns and Gain: The Antenna Toolbox allows users to calculate and visualize radiation patterns, gain, and directivity of antennas. These features enable engineers to analyze and optimize antenna performance in terms of coverage, beamwidth, and power radiated in different directions.

3. Array Analysis: The toolbox includes tools for designing and analyzing antenna arrays, including linear arrays, planar arrays, and conformal arrays. Users can explore array properties, such as beamforming, side lobe suppression, and array factor calculations.

4. Antenna Impedance and Matching: MATLAB's Antenna Toolbox facilitates the calculation and optimization of antenna impedance and matching networks. It provides tools to analyze impedance matching techniques such as baluns, matching stubs, and transmission line transformers.

5. Performance Metrics: The toolbox supports the evaluation of various performance metrics for antennas, including return loss, bandwidth, efficiency, and cross-polarization. These metrics help engineers assess the overall performance and characteristics of their antenna designs.

6. Visualization and Reporting: MATLAB's graphical capabilities enable users to visualize antenna structures, radiation patterns, and other antenna characteristics through 2D and 3D plots. Users can also generate customizable reports and export simulation results for documentation and sharing purposes.

7. The Antenna Toolbox in MATLAB offers an extensive collection of functions and features that enable engineers and researchers to design, simulate, and analyze antennas efficiently. It provides a flexible and intuitive platform for exploring antenna properties, optimizing designs, and assessing performance. By leveraging the power of MATLAB, users can seamlessly integrate antenna design and analysis with other computational and numerical tasks, making it a valuable tool in the field of electromagnetic wave propagation and antenna engineering.

5.3.1.3 OOP: Code Structure and Organization

Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around objects that encapsulate data and behavior. It provides a structured approach to programming by modeling real-world entities as objects and defining their properties (attributes) and actions (methods). Here's an overview of key concepts and principles in Object-Oriented Programming:

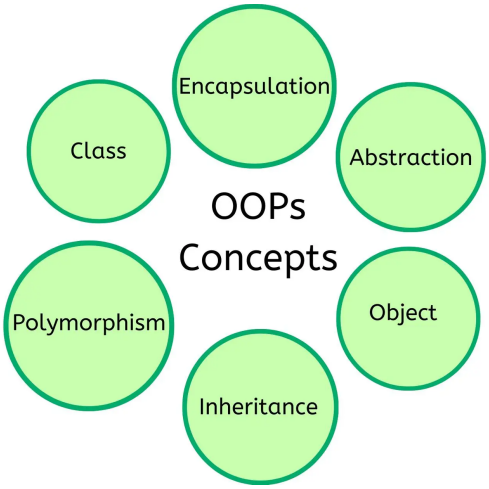


Figure 5.12: Object-Oriented Programming

1. Objects: Objects are instances of classes and represent individual entities with a distinct state and behavior. They encapsulate data (attributes) and operations (methods) that manipulate and interact with that data.

2. Classes: A class is a blueprint or template that defines the structure and behavior of objects. It serves as a blueprint for creating multiple instances of objects with shared characteristics. Classes specify the attributes and methods that objects of that class possess.

3. Encapsulation: Encapsulation is the principle of bundling data and related methods within an object, hiding internal details from the outside world. It ensures data integrity and provides a clear interface for interacting with the object, promoting modular and reusable code.

4. Inheritance: Inheritance is a mechanism that allows classes to inherit properties and behavior from other classes. It enables the creation of hierarchies and the reuse of code. A derived class (subclass) inherits characteristics from a base class (superclass), and it can add or override existing behavior.

5. Polymorphism: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables the use of the same interface to perform different actions based on the actual type of the object. Polymorphism promotes flexibility and extensibility in code.

6. Abstraction: Abstraction focuses on defining essential features of an object while hiding unnecessary implementation details. It allows programmers to create abstract classes or interfaces that specify the common behavior and characteristics shared by a group of related objects.

7. Modularity: OOP promotes modularity by breaking down complex systems into smaller, manageable units (objects and classes). Each object or class has a specific responsibility and can be developed, tested, and modified independently, enhancing code maintainability and reusability.

8. Message Passing: In OOP, objects communicate with each other by sending messages. Messages trigger the execution of methods on the receiving object, allowing objects to collaborate and exchange information.

Object-Oriented Programming provides several benefits, including code reusability, modularity, encapsulation, and the ability to model complex systems. It promotes a structured and organized approach to software development and helps manage complexity in large-scale projects. Popular programming languages that support OOP include Java, C++, Python, and C#. These languages provide features and syntax specifically designed for creating and working with objects and classes. By adopting OOP principles, developers can write cleaner, more maintainable code, leverage code reuse, and design software systems that closely align with real-world concepts and entities.

5.3.1.4 UML: Class Diagram of Model

In this section, we present the UML class diagram of the Model, which illustrates the structure and relationships between the classes involved in our system. The diagram includes three classes: **Channel Link**, **Cellular Environment**, and **Q-Learning**.

The **Channel Link** class represents the link between two communication points and has attributes such as *link_distance*, *received_power*, and *transmit_power*. It encapsulates information about the distance between the communication points and the power levels involved in the link.

The **Cellular Environment** class represents the environment in which the communication system operates. It has attributes such as *latitude*, *longitude*, *transmit_power*, and *antenna_gain*, which describe the geographical location and characteristics of the cellular environment.

The **Q-Learning** class represents the Q-Learning algorithm used in our system for decision-making and optimization. It has attributes such as *learning_rate*, *epsilon*, and *discount_factor*, which control the learning process and exploration-exploitation trade-off in the algorithm.

The UML class diagram provides a visual representation of the classes and their relationships in the Model. It serves as a blueprint for understanding the structure and behavior of the system, facilitating the implementation and maintenance of the software components.

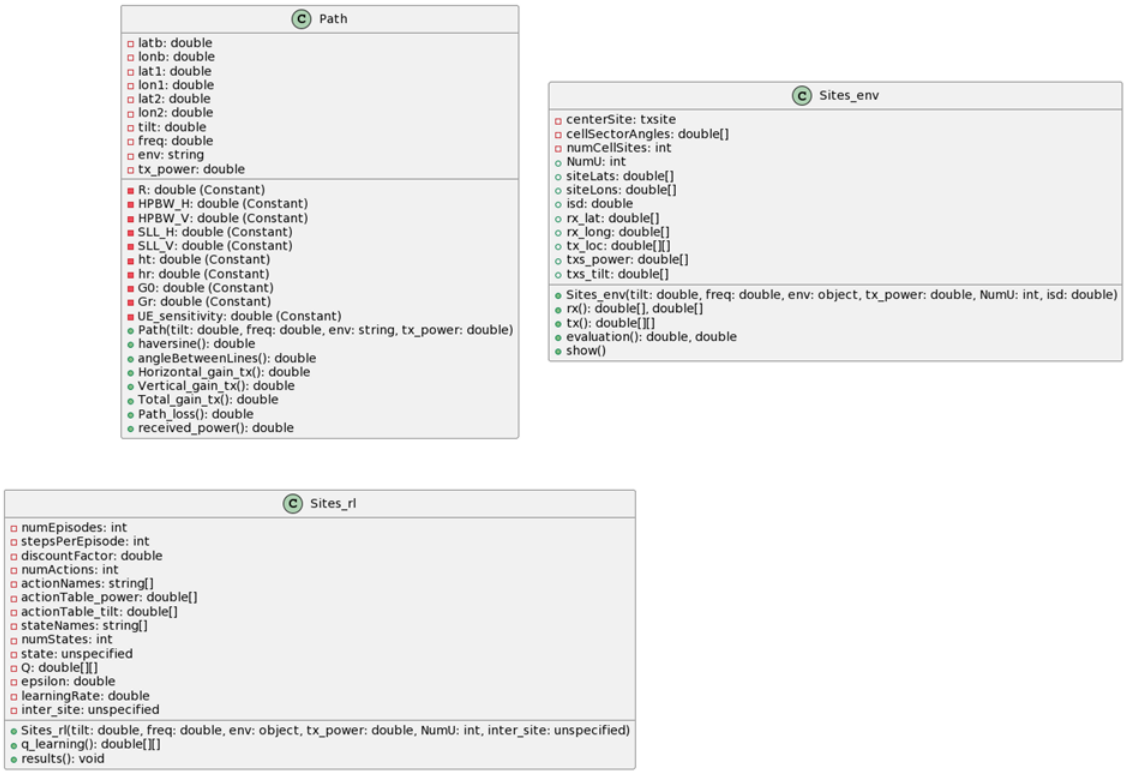


Figure 5.13: UML: Class Diagram of Model

5.3.2 Algorithms and Techniques

5.3.2.1 Evaluation Implementation

The following MATLAB function, named "evaluation", is used to evaluate the performance of the system based on the metrics AGap (Number of unconnected users) and AOverlap (Number of connected users).

```
function [AGap,AOverlap] = evaluation(obj)
    AGap = 0;
    AOverlap = 0;
    main_beam_loc = zeros(obj.numCellSites/2,2);
    txSite = txsite('Name','txsite','Latitude',obj.tx_loc(2,1),'
   Longitude',obj.tx_loc(2,2));
    [main_beam_loc(1,1),main_beam_loc(1,2)] = location(txSite,
   1000, 270);
    txSite = txsite('Name','txsite','Latitude',obj.tx_loc(4,1),'
   Longitude',obj.tx_loc(4,2));
    [main_beam_loc(2,1),main_beam_loc(2,2)] = location(txSite,
   1000, 30);
    txSite = txsite('Name','txsite','Latitude',obj.tx_loc(6,1),'
   Longitude',obj.tx_loc(6,2));
    [main_beam_loc(3,1),main_beam_loc(3,2)] = location(txSite,
   1000, 150);
    for j = 1:obj.NumU
        count_g = 0;
        count_o = 0;
```

```
        for i = [2,4,6]
            obj.tx_power = obj.txs_power(i,1);
            obj.tilt = obj.txs_tilt(i,1);
            obj.latb = obj.tx_loc(i,1);
            obj.lonb = obj.tx_loc(i,2);
            obj.lat1 = main_beam_loc(i/2,1);
            obj.lon1 = main_beam_loc(i/2,2);
            obj.lat2 = obj.rx_lat(j);
            obj.lon2 = obj.rx_long(j);
            power = obj.GetReceivedPower;
            if (power<obj.UE_sensitivity)
                count_g = count_g + 1;
            else
                count_o = count_o + 1;
            end
        end
        if (count_g == 3)
            AGap = AGap + 1;
        end
        if (count_o == 3)
            AOverlap = AOverlap + 1;
        end
    end
    AGap = AGap/obj.NumU;
    AOverlap = AOverlap/obj.NumU;
end
```

Listing 5.1: Evaluation Function Implementation

The "evaluation" function takes an input object, "obj", and returns the values of AGap and AOverlap.

The function first initializes the variables AGap and AOverlap to zero. It then calculates the latitude and longitude of the main beam locations using the "location" function based on the given transmitter locations.

Next, the function iterates through each user (j) and performs a nested loop for each transmitter (i). It sets the transmitter power, tilt, and location parameters, as well as the receiver latitude and longitude. The received power is calculated using the "GetReceivedPower" method of the object. Depending on the received power, the function increments the corresponding counters (count_g and count_o).

After the loops, the function calculates the ratios of AGap and AOverlap by dividing the respective counters by the total number of users (obj.NumU).

This "evaluation" function provides a quantitative assessment of the system's performance based on the number of unconnected and connected users, allowing for evaluation and optimization of the system parameters.

5.3.2.2 Q-Learning Implementation

The q_learning function begins by evaluating the system's performance using the evaluation method, which calculates the number of unconnected users (AGap) and the number of connected users (AOverlap). Based on the AGap value, the current state of the system is determined.

The function then enters a loop where it iterates over multiple episodes and steps within each episode. Within each step, the function chooses an action based on an epsilon-greedy policy. With a certain probability (epsilon), a random action is selected; otherwise, the action with the maximum Q-value for the current state is chosen.

The chosen action is applied to the system by adjusting the transmit power and tilt of the antennas. The evaluation is performed again, and the new AGap and AOverlap values are obtained. The previous state, AGap, and AOverlap values are stored for reward calculation.

The reward is assigned based on the change in AGap and AOverlap values compared to their previous values. If AGap decreases and AOverlap is within 1.04 times the previous AOverlap, a positive reward of 6 is given. If AGap decreases but AOverlap increases beyond 1.04 times the previous AOverlap, a smaller positive reward of 3 is given. If AGap increases, a negative reward of -2 is given. If none of these conditions are met, and the current state is the best state (AGap equals zero), a large positive reward of 10 is given, and the epsilon value is set to zero to exploit the best action.

Finally, the Q-value for the previous state-action pair is updated using the Q-learning update equation. The learning rate and discount factor are used to control the impact of immediate rewards and future rewards, respectively.

After completing all the episodes and steps, the function returns the updated Q-values.

```
function q = q_learning(obj)
          [AGap,AOverlap] = obj.evaluation;
          if AGap == 0
              obj.state = 1;
          elseif AGap*100 < 20 && AGap*100 > 0
              obj.state = 2;
          elseif AGap*100 < 40 && AGap*100 >= 20
              obj.state = 3;
          elseif AGap*100 < 60 && AGap*100 >= 40
              obj.state = 4;
          elseif AGap*100 < 80 && AGap*100 >= 60
              obj.state = 5;
          elseif AGap*100 < 100 && AGap*100 >= 80
              obj.state = 6;
          else
              obj.state = 7;
          end
```

```
            for episode = 1:obj.numEpisodes
                for step = 1:obj.stepsPerEpisode
                    for i = 1:3
                        % Choose action based on epsilon-greedy
    policy
                        if rand < obj.epsilon
                            action = randi(obj.numActions);
                        else
                            [~, action] = max(obj.Q(obj.state + 7*(
    i-1), :));
                        end
                        % Take action and observe new state and
    reward
                        obj.txs_power(2*i) = obj.txs_power(2*i) +
    obj.actionTable_power(action);
                        obj.txs_tilt(2*i) = obj.txs_tilt(2*i) + obj
    .actionTable_tilt(action);
                        AGap_old = AGap;
                        AOverlap_old = AOverlap;
                        obj.tx_power = obj.txs_power(2*i);
                        obj.tilt = obj.txs_tilt(2*i);
                        [AGap,AOverlap] = obj.evaluation;
                        state_old = obj.state;
                        if AGap == 0
                            obj.state = 1;
                        elseif AGap*100 < 20 && AGap*100 > 0
                            obj.state = 2;
                        elseif AGap*100 < 40 && AGap*100 >= 20
                            obj.state = 3;
                        elseif AGap*100 < 60 && AGap*100 >= 40
                            obj.state = 4;
                        elseif AGap*100 < 80 && AGap*100 >= 60
                            obj.state = 5;
                        elseif AGap*100 < 100 && AGap*100 >= 80
                            obj.state = 6;
                        else
                            obj.state = 7;
                        end
                        if AGap<AGap_old && AOverlap <= 1.04*
    AOverlap_old
                            reward = 6;
                        elseif AGap<AGap_old && AOverlap > 1.04*
    AOverlap_old
                            reward = 3;
                        elseif AGap>AGap_old
                            reward = -2;
                        else
                            if obj.state == 1
                                obj.Q(1 + 7*(i-1),find(strcmp(obj.
    actionNames, 'const_power & const_tilt'))) = 10;
                                obj.epsilon = 0;
                            else
                                reward = 1;
                            end
                        end
                        obj.Q(state_old + 7*(i-1), action) = obj.Q(
    state_old + 7*(i-1), action) +...
                            obj.learningRate * (reward + obj.
```

```
    discountFactor * max(obj.Q(obj.state + 7*(i-1), :))...
67                              - obj.Q(state_old + 7*(i-1), action));
68                      end
69                  end
70              end
71              q = obj.Q;
72          end
73          %{
```

Listing 5.2: Q-learning Function Implementation

5.3.3 Input/Output Handling

5.3.3.1 Input Data to Model

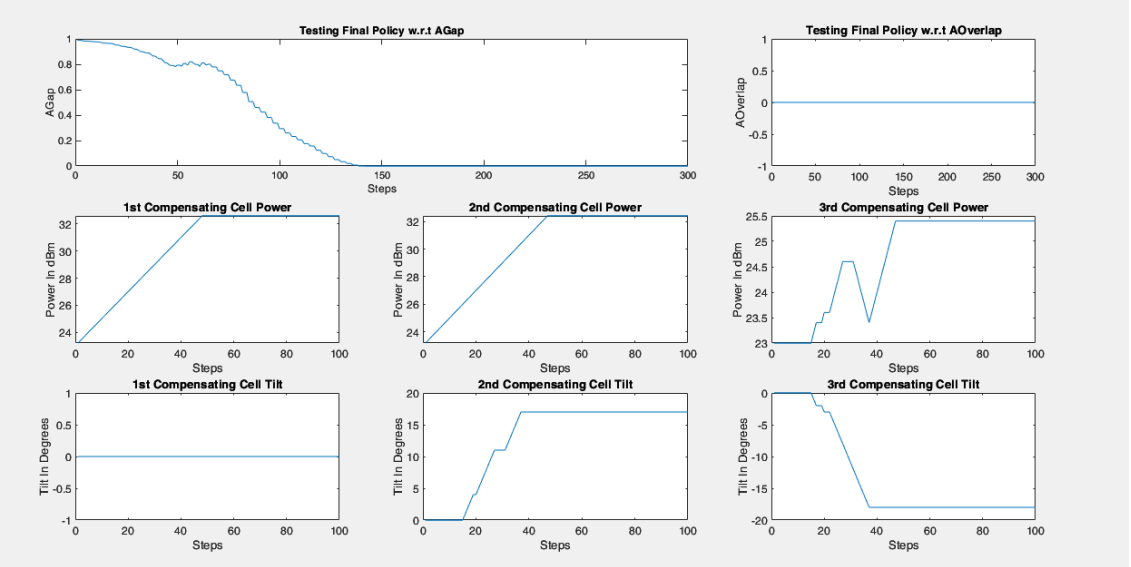
in this table the input parameters are explained

Tilt	The initial tilt state of each sector antenna
Frequency	The frequency that the cells operate at
Environment	The type of the environment
Transmitted power	The initial transmitted power of each sector antenna
No. of users	The number of unconnected users in the center site
Inter-site distance	The distance between the base stations

Table 5.1: Network Parameters

5.3.3.2 Output Data and Graphs

The graphs represent the change of the system with no. of steps. In these graphs we see how the Agap is changed with each step and how it changes with the change of power and tilt. Also we can see how the transmitted power and tilt is changing to compensate the coverage area of the failed center site.



5.3.4 GUI Design of Model

To design the GUI we used MATLAB App Designer. MATLAB App Designer is a visual development environment provided by MathWorks that allows users to create interactive applications with a graphical user interface (GUI) in MATLAB. It simplifies the process of building and deploying MATLAB apps by providing a drag-and-drop interface for designing the user interface and a code editor for implementing the functionality. There are several benefits of the MATLAB App Designer:

1. Visual App Design: App Designer provides a visual interface for designing the layout and components of the app's GUI. Users can simply drag and drop components like buttons, sliders, tables, plots, and more onto the design canvas. Components can be resized, aligned, and customized using properties to achieve the desired look and behavior.

2. Interactive Controls: MATLAB App Designer offers a wide range of interactive controls that can be added to the GUI, enabling users to interact with the app. These controls include buttons, checkboxes, radio buttons, sliders, drop-down menus, text boxes, and more. Users can define the actions associated with these controls using callback functions.

3. Data Visualization: App Designer allows users to incorporate data visualization into their apps. It supports plotting and displaying data using built-in functions like plot, scatter, bar, and surface plots. Users can update plots dynamically based on user input or data processing results.

4. Integration with MATLAB: MATLAB App Designer seamlessly integrates with MATLAB's extensive library of functions, toolboxes, and capabilities. Users can leverage MATLAB's powerful computational and data analysis functions within their app. MATLAB code written outside App Designer can be easily incorporated into the app's callbacks and functions.

5.4 Model Validation

In our Cell Outage Compensation (COC) model, we aim to validate the effectiveness and reliability of our model through random/monkey testing. Random/monkey testing is a type of testing approach where inputs are generated randomly or in a non-deterministic manner to simulate real-world scenarios and assess the behavior and performance of the system.

By performing random/monkey testing in our COC model, we introduce various random inputs and simulate different scenarios to validate the robustness and stability of our model. This testing approach helps us identify potential issues,

uncover hidden bugs or vulnerabilities, and ensure that our COC model can handle unexpected situations and provide accurate and consistent results.

During random/monkey testing, we generate random test cases by varying parameters such as transmit power, tilt angles, link distances, or other relevant variables in our COC model. These random inputs challenge the model with different combinations and configurations, allowing us to observe the system's response and evaluate its performance under diverse conditions.

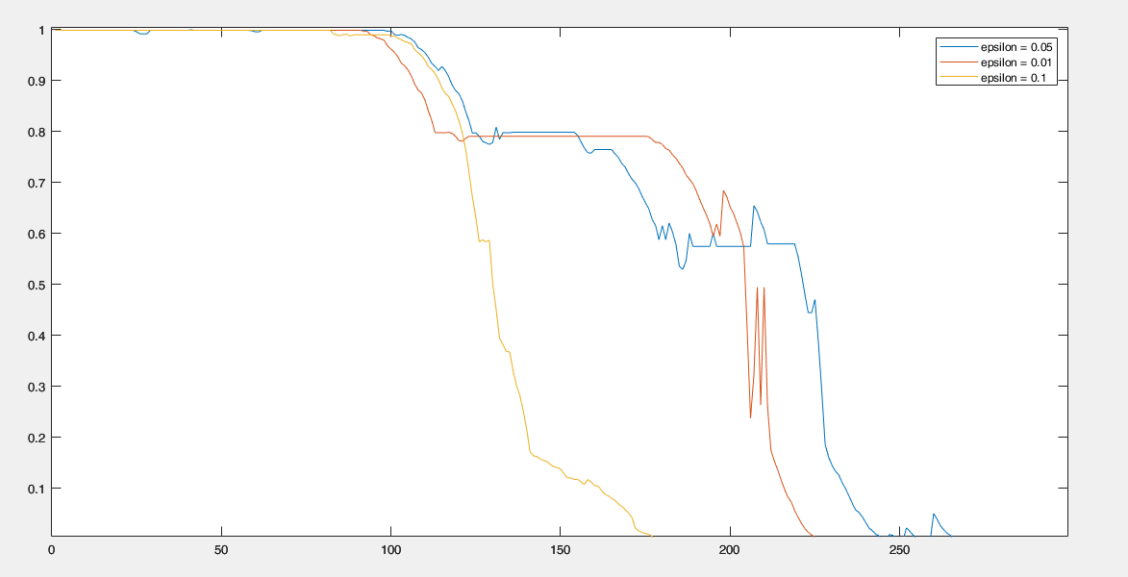
The purpose of random/monkey testing is to uncover any unexpected behaviors, errors, or inconsistencies that may arise in our COC model. By subjecting the model to a wide range of random inputs, we can validate its functionality, identify edge cases, and ensure that it operates correctly and reliably in different scenarios.

Through random/monkey testing, we aim to gain confidence in the accuracy and robustness of our COC model. By validating the model against a diverse set of random inputs, we can increase its reliability and ensure that it can effectively compensate for cell outages in real-world communication systems.

5.5 Model Optimization

5.5.1 Epsilon

Epsilon also called the exploration factor it is responsible for the balance between exploration and exploitation. The agent choose its action based on the ϵ-greedy policy. This policy works by generating a random value between 0 and 1. If this value is smaller than ϵ, the agent chooses a random action from the action space and if the random value is larger the agent choose the action that will lead to the maximum cumulative reward. This allows the agent to explore all the possible and also allows the agent to exploit the knowledge that was gathered. There are several strategies for choosing Epsilon value such as: Constant Epsilon, and Decaying Epsilon. The Epsilon strategy and the epsilon value is dependent on the problem.



5.5.2 Discount Factor

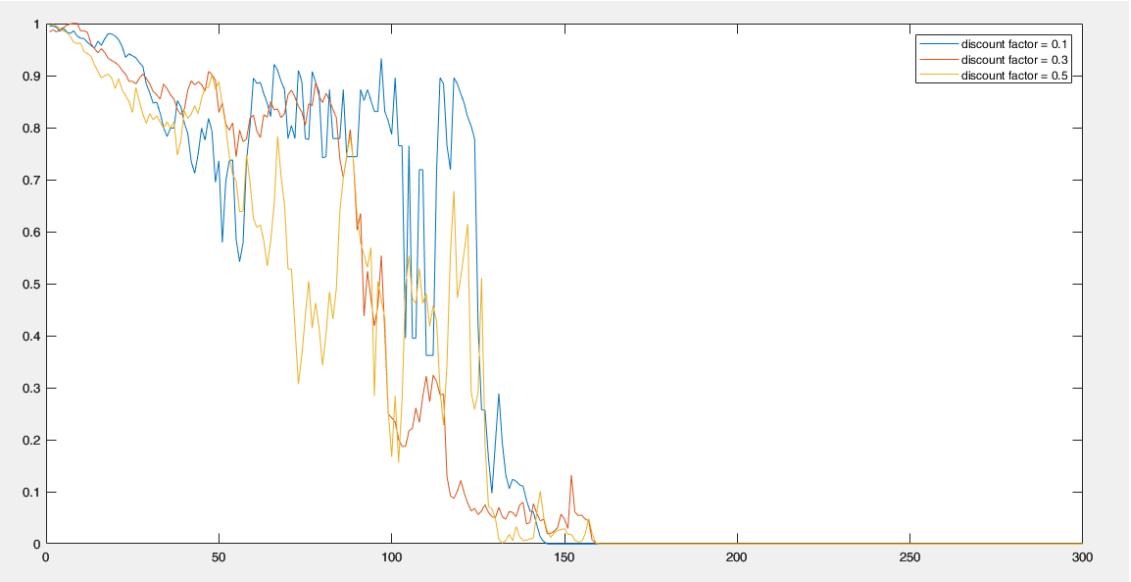
The discount factor, often denoted as γ, is a parameter used in Q-learning and other reinforcement learning algorithms to determine the importance of future rewards compared to immediate rewards. It controls the extent to which the agent values long-term rewards when making decisions.
The discount factor affects the Q-learning process in two main ways:

1. Future Reward Consideration: The discount factor determines how much the agent values future rewards compared to immediate rewards. A value of gamma close to 1 indicates that the agent heavily weighs future rewards. In this case, the agent considers long-term consequences and prefers actions that may lead to higher cumulative rewards in the future. On the other hand, a value of gamma close to 0 means the agent primarily focuses on immediate rewards and pays less attention to future rewards. It tends to prioritize short-term gains rather than long-term planning.

2. Convergence and Stability: The discount factor affects the convergence and stability of the Q-learning algorithm. A higher gamma value encourages the agent to consider future rewards, which can lead to more stable and well-considered policies. However, it can also increase the time required for the algorithm to converge. Conversely, a lower gamma value promotes faster convergence but may result in policies that are more sensitive to changes in the environment.

Choosing an appropriate discount factor depends on the specific problem and environment. If the agent's actions have long-term consequences and future rewards are significant, a higher gamma value is preferable. If immediate rewards are more

critical or the environment is dynamic, a lower gamma value might be more suitable.



5.5.3 Learning Rate

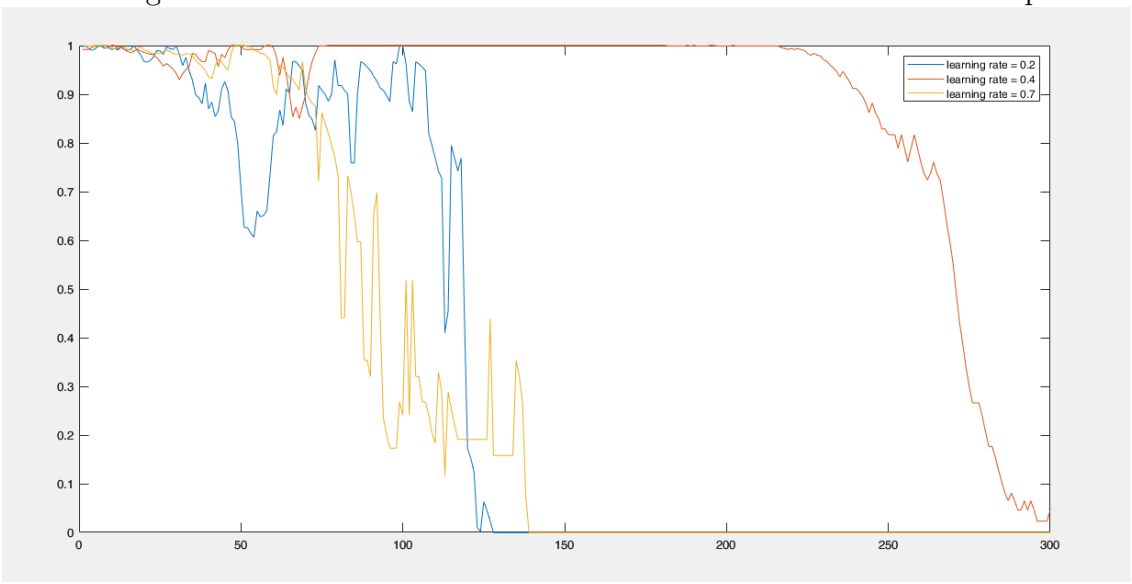
The learning rate, often denoted as α, is a parameter used in Q-learning and other reinforcement learning algorithms to control the rate at which the agent updates its Q-values based on new information. It determines the weight given to new experiences compared to previously learned values.
The learning rate parameter affects the Q-learning process in the following ways:

1. Rate of Learning: The learning rate determines how quickly the agent updates its Q-values. A higher learning rate means that the agent places more emphasis on the most recent information it encounters. Consequently, the agent adapts its Q-values more rapidly and is more responsive to changes in the environment. On the other hand, a lower learning rate results in slower updates and a more stable learning process. The agent relies more on accumulated experience, which can be beneficial in situations where the environment is noisy or subject to frequent changes.

2. Exploration vs. Exploitation: The learning rate interacts with the exploration and exploitation trade-off in Q-learning.A higher learning rate can encourage more exploration as the agent assigns higher weight to new experiences and adjusts its Q-values more significantly. Conversely, a lower learning rate might promote exploitation, as the agent relies more on the existing Q-values and is less likely to deviate from its learned policy.

Choosing an appropriate learning rate is crucial for successful Q-learning. If the learning rate is too high, the agent may overreact to noisy or irrelevant information, leading to unstable or suboptimal policies. On the other hand, if the learning rate is too low, the agent might take longer to converge or may struggle to adapt to changes

in the environment. Finding the optimal learning rate often requires experimentation and fine-tuning. Some common approaches include using a decaying learning rate schedule that gradually reduces the learning rate over time or dynamically adjusting the learning rate based on the observed rewards or state-action visitation frequencies.



Chapter 6

Design and Implementation of FPGA-Based Q-Learning Hardware for COC

6.1 Q-Learning Implementation

The behavioral task of the reinforcement learning system is to find an optimal policy after trying out various possible sequences of actions. Q-learning defines a procedure for taking samples of the environment and associating actions with observed rewards as explained in detail in Chapter 4. Environment is needed with the goal of improving and accelerating existing software-based implementation methodologies and sequential computational frameworks.For more feasibility, preliminary hardware designing, prototyping and design for manufacturing & assembly The following steps shown in figure 6.1 are followed.

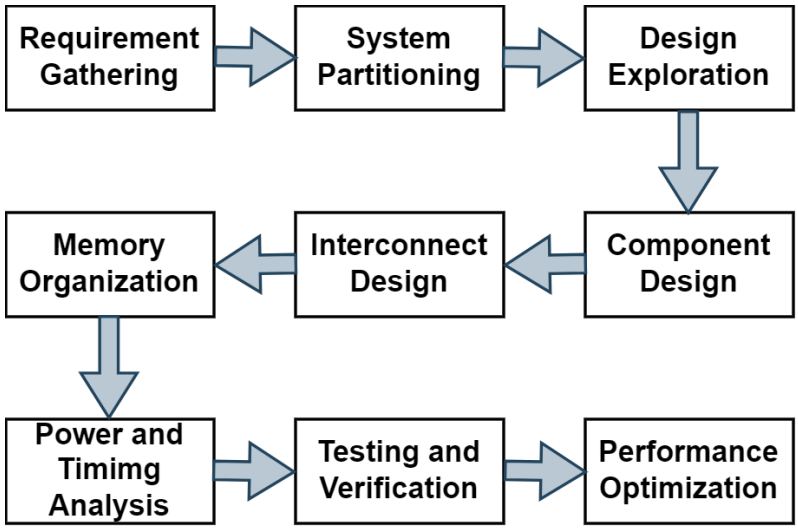


Figure 6.1: Design Procedure

6.2 Requirement Gathering

Requirement gathering in hardware design involves collecting and understanding the specifications, constraints, and expectations for the desired hardware system. It is a crucial initial step that helps define the design goals and guides the entire hardware design process. The requirements should be clear, concise, and unambiguous so the system should achieve functional requirements, performance requirements and Input/Output specifications, data formats, and processing requirements.

6.2.1 Functional Requirements

Efficient Q-Learning Algorithm Calculation
The system should efficiently support the calculations involved in the Q-learning RL algorithm. This includes performing the necessary computations for updating and accessing the Q-table, which represents the expected rewards for each action in each state. The system should optimize the algorithms and data structures used in these calculations to minimize computational overhead and ensure efficient processing.

Q-Table Generation Component
include a component responsible for generating the Q-table. This component initializes and sets up the Q-table based on the environment's state and action space. It ensures that the Q-table is properly initialized with default values or any domain-specific requirements. The component should be able to handle various types of environments and their corresponding state and action spaces.

Action Discovery Component
It is responsible for enabling the agent to choose actions based on the Q-table values or exploration strategies. It should provide mechanisms for the agent to select actions based on the highest Q-value (exploitation) or to explore new actions based on a certain exploration strategy, such as the aforementioned epsilon-greedy policy. The component should facilitate efficient and effective action selection based on the current state of the environment.

Reward Updates Component
A component responsible for updating the rewards in the Q-table is incorporated. This component receives feedback in the form of rewards from the environment and updates the corresponding Q-values accordingly. It should handle the proper reward update mechanism, such as applying the Bellman equation or other reward aggregation techniques. The component should ensure that the rewards are accurately incorporated into the Q-value calculation process to reflect the agent's experiences and improve the decision-making process.

Real-Time Learning and Decision-Making Capabilities
Real-time learning and decision-making capabilities are enabled. This includes supporting continuous learning and decision-making processes that can adapt and

respond in real-time to changing conditions. The system should facilitate online learning, where the agent can update the Q-table incrementally as new data is received from the environment. It should also provide mechanisms for dynamic decision-making, allowing the agent to make informed decisions based on the current state and the information stored in the Q-table. The system should balance exploration and exploitation to ensure effective learning and decision-making. Additionally, the system should optimize computations to minimize delays and efficiently handle real-time interactions with the environment.

6.2.2 Performance requirements

The system aims to achieve a significant speedup compared to software-based implementations of the Q-learning RL algorithm. This means that the system should utilize hardware acceleration or optimized algorithms to perform the necessary computations faster. By leveraging hardware resources efficiently or employing specialized techniques, the system should strive to minimize the time required for Q-table updates, action selection, and reward updates. The speedup should be noticeable and result in improved overall performance compared to traditional software-based approaches.

The system should be capable of processing a large number of states and actions efficiently. Q-learning typically involves dealing with a state space and an action space, both of which can be extensive in complex environments. The system should optimize the data structures and algorithms used for storing and accessing states and actions to handle large volumes of data effectively. This includes efficient lookup and retrieval of states and actions from the Q-table and other relevant data structures.

Minimizing the latency between receiving inputs (such as observations or requests) and producing outputs (such as action decisions or Q-table updates). It should aim to minimize any delays or processing time between these two steps to ensure real-time or near-real-time performance. The latency reduction should be achieved through efficient algorithms, optimized computations, and streamlined processing steps. Minimizing latency is crucial for enabling responsive decision-making and providing a seamless user experience.

Determine the format of the input states that the Q-learning RL algorithm will operate on. This can involve specifying the structure and representation of the states, such as network states or default assumptions. For example, in a game environment, the input state format might include pixel-level information from a screen or higher-level game state variables. By determining the input state format, the system can appropriately handle and process the inputs, ensuring compatibility with the Q-learning algorithm and providing accurate representations of the environment to the agent.

6.2.3 Interfaces and Data Formats

System Inputs:

- **State**
  The current state of the system is provided as an input. It could be represented using binary signals, numerical values, or any other suitable format that captures the relevant information about the system's state.

- **Reward**
  After the system performs the action, a reward signal is received as feedback. The reward represents the immediate feedback or reinforcement signal that indicates how well the chosen action performed in the current state. The reward can be a numerical value or a predefined set of values indicating success or failure.

System Outputs:

- **Action**
  The chosen action to be taken in the current state is provided as an input.

- **Updated Q-Table**
  The updated Q-values are computed based on the observed reward and are stored in the Q-table or Q-value function. The Q-values represent the expected future rewards for different state-action pairs.

- **Policy**
  The learned policy is an output that guides the selection of actions based on the current state. It represents the learned behavior or strategy for the agent to choose the most optimal action in different states.

6.2.3.1 Q-Table Storage and Result Export

The Q-table is typically stored in a data structure suitable for efficient retrieval and update operations. After training the reinforcement learning algorithm, we may want to export the learned Q-table or policy for evaluation, visualization, or deployment purposes.

6.2.3.2 Float to Fixed-Point Data Processing and Storage

Conversion and handling of floating-point data to fixed-point representation for processing and storage in the system. Utilization of fixed-point arithmetic for numerical computations to minimize resource usage on the hardware platform.

6.3 System Partitioning and Design Exploration

In both parallel and series implementations, system partitioning involves breaking down the system into smaller components or modules, and design exploration involves evaluating and comparing different design alternatives to optimize system performance and meet the desired objectives. The choice between parallel and series implementation depends on factors such as the system's requirements, computational resources, scalability, and available parallelization techniques.

6.3.1 System Partitioning

To partition the system for parallel implementation in Q-learning, the major functions or capabilities of the system are identified and divided into independent subsystems. As shown in figure 6.2, Each subsystem should have well-defined inputs, outputs, and interactions with other subsystems. It is crucial to consider factors like load balancing and data dependencies to ensure effective parallel execution.

6.3.2 Design Exploration

During the design exploration phase, different approaches for parallelization are considered. Task parallelism involves dividing the workload into distinct tasks that can be executed concurrently, while data parallelism focuses on dividing the data into segments that can be processed simultaneously. Various parallelization techniques, algorithms, and hardware architectures are evaluated to determine the most suitable design alternative for achieving the desired system performance.

6.3.2.1 Parallel Implementation

Parallel implementation involves dividing a system into multiple subsystems that can operate simultaneously and independently, each performing specific tasks or sets of tasks concurrently. This approach is commonly used to improve system performance and efficiency.

In the context of Q-learning, parallel implementation can significantly speed up the learning process by taking advantage of concurrent processing[21].

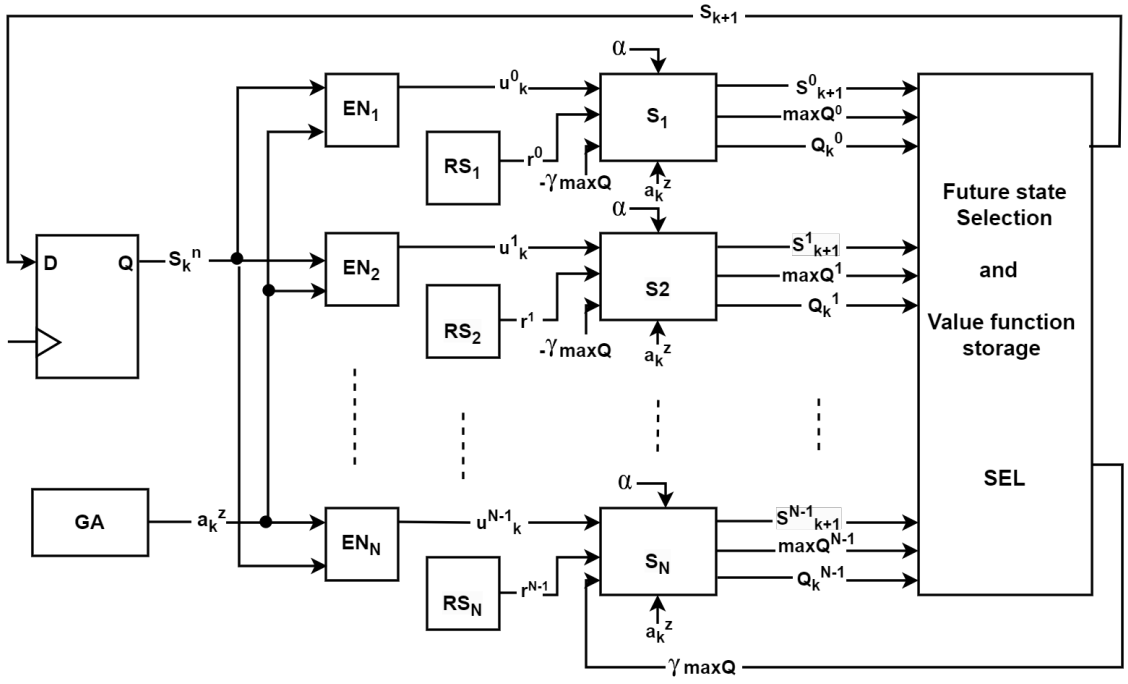


Figure 6.2: Overview Architecture of The Parallel Q-Learning Accelerator

6.3.2.2 Advantages of Q-learning Parallel Implementation

Increased Speed and Throughput
Parallelism enables concurrent processing, resulting in faster execution of Q-learning algorithms. Multiple agents operating in parallel allow for increased exploration and exploitation, leading to faster convergence and improved learning efficiency. Customized hardware accelerators on FPGA can further boost the performance of computationally intensive operations, such as Q-value updates.

Resource Utilization
FPGAs offer abundant hardware resources that can be efficiently utilized for parallel Q-learning implementations. Replicating Q-tables and associated logic in parallel can fully leverage the available FPGA resources, maximizing the utilization of the hardware[22].

Real-Time Responsiveness
Parallel implementation allows for real-time decision-making and quick response to changing environments. The parallel processing capabilities of FPGA enable rapid exploration and update of Q-values, facilitating timely decision-making by the Q-learning agent.

6.3.2.3 Disadvantages of Q-learning Parallel Implementation

Design Complexity
Parallel implementation on FPGA introduces additional complexity in terms of design, verification, and debugging[22]. Managing the synchronization and coordination of multiple agents and pipeline stages requires careful consideration to ensure correct operation.

Development Effort
Implementing Q-learning on FPGA requires expertise in both Q-learning algorithms and FPGA development. Designing and optimizing hardware accelerators specifically for Q-learning may demand significant development effort.

Resource Limitations Despite FPGA's abundant resources, limitations may arise when dealing with large-scale Q-learning problems. The number of parallel agents and the size of Q-tables may be constrained by the available FPGA resources[22].

6.3.2.4 Series Implementation

In a series implementation of Q-learning according to 6.3, a single agent performs the learning process sequentially. It explores the environment, updates Q-values, and refines its policy step by step.

Key aspects of series implementation in Q-learning include:

Sequential Exploration
The agent explores the environment sequentially, taking actions and updating Q-values based on the observed rewards and experiences. It continues this process iteratively until it converges to an optimal policy[23].

Policy Improvement
In series implementation, the agent continually improves its policy by adjusting its actions based on the learned Q-values. It follows a feedback loop of exploration, Q-value update, and action selection.

Deterministic Execution
The execution order of the learning process is well-defined and deterministic[23]. This deterministic nature allows for easier control, tracking, and reproduction of the learning process.

Lower Communication Overhead
Unlike parallel implementations, series implementations do not require communication or synchronization between agents, reducing communication overhead.

6.3.2.5 Advantages of Series Implementation

Simplicity
Serial implementation is generally easier to design and implement, requiring fewer hardware resources compared to parallel architectures.

Reduced Resource Usage
Since only one state-action pair is processed at a time, the memory and processing requirements are typically lower compared to parallel implementations.

Determinism
In a series implementation, the execution order of the Q-learning steps is fixed and deterministic. This deterministic nature can be useful when debugging or reproducing results, as the behavior of the algorithm remains consistent across multiple runs.

Resource utilization
Series implementation can be advantageous when the computational resources are limited or the environment is complex. By executing one step at a time, it allows the agent to focus its resources on a single step, potentially leading to more efficient resource utilization.

Inter-step dependencies
Q-learning often involves dependencies between consecutive steps, especially when updating the Q-values. In a series implementation, these dependencies can be easily managed and maintained, as each step is performed in a sequential manner. This can simplify the synchronization and communication between steps.

Debugging and error handling
Series implementation allows for easier debugging and error handling. Since each step is executed one after another, it becomes simpler to identify and isolate issues that may occur during the execution. Errors or inconsistencies can be traced back to specific steps, making it easier to pinpoint and resolve them.

6.3.2.6 Disadvantages of Series Implementation

Slower convergence
Series implementation may suffer from slower convergence compared to parallel implementation. In a series implementation, each step is executed sequentially, which can limit the agent's ability to explore and learn from different parts of the environment simultaneously. This can result in slower convergence towards an optimal policy or value function.

Resource inefficiency
In series implementation, the computational resources are not fully utilized. While one step is being executed, the other steps are idle, leading to potential resource wastage. This inefficiency becomes more pronounced when dealing with large state

or action spaces, where parallel execution could significantly speed up the learning process.

Limited exploitation-exploration balance
Q-learning requires a balance between exploitation (taking actions based on the current knowledge) and exploration (exploring new actions to gather more information). In a series implementation, the agent explores and exploits sequentially, which may limit its ability to strike an optimal balance[23]. For example, the agent may exploit too early without fully exploring the environment, or it may spend excessive time exploring in a region it has already learned extensively.

Increased sensitivity to initial conditions
In series implementation, the order of executing steps can have an impact on the learning process. If the initial conditions or the order of steps are not favorable, it can lead to sub-optimal learning or convergence. This sensitivity to initial conditions can make series implementation less robust compared to parallel implementation, where the order of execution is less critical.

In our implementation we highlight the series implementation for its features

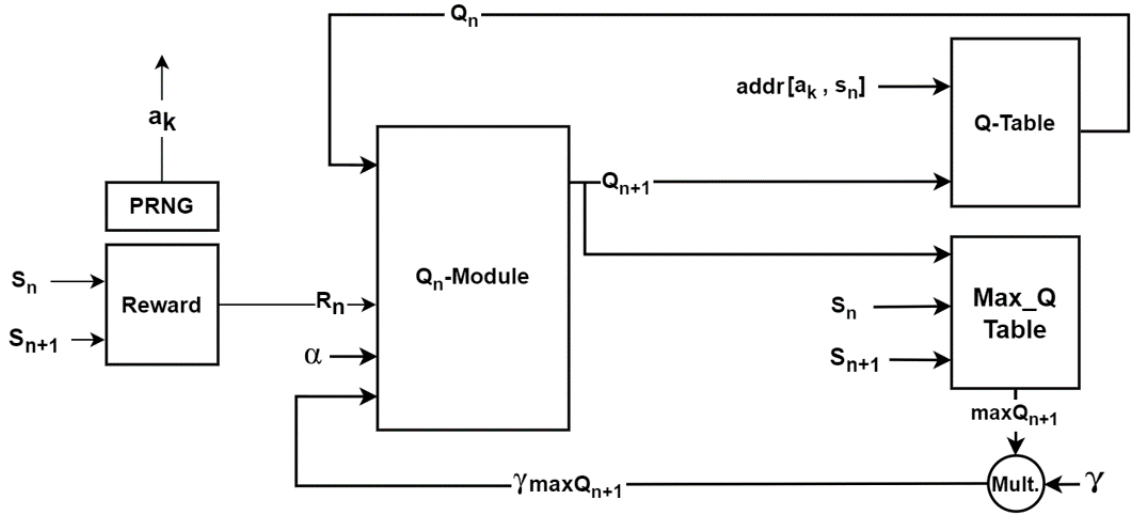


Figure 6.3: Overview Architecture of the proposed Q-Learning Accelerator

that allows for precise control, tracking, and reproduction of the learning process. Furthermore, it requires fewer computational resources, making it a preferred choice in scenarios with limited hardware capabilities.

6.4 Module Function Definition

6.4.1 PRNG Module

PRNG (pseudo random number generator) is the starting point for executing the algorithm. PRNG is a deterministic algorithm that generates a sequence of numbers that approximates random numbers. PRNGs are widely used in various applications such as simulations, cryptography, gaming, and scientific research. it is necessary to draw the a_z actions for each s_n for this purpose a random action a_z is generated using a random value from a Pseudo-Random Number Generator (PRNG) module[24]. The output of the PRNG is represented by the 4-bit signal Action, which represents the four most significant bits of the $PRNG_{out}$ register bits is illustrated in Figure 6.4.

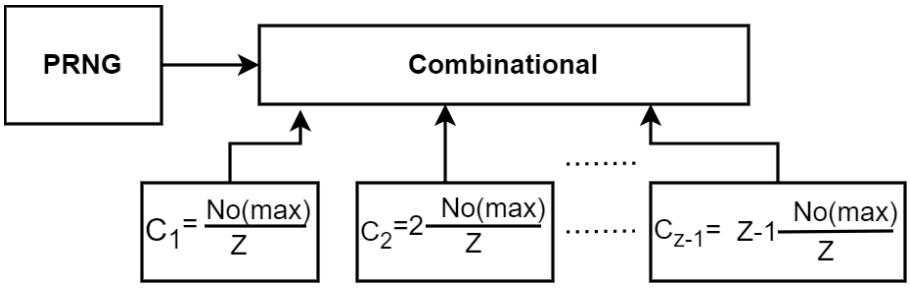


Figure 6.4: PRNG Module

6.4.2 Unit Delay

A unit delay can refer to the use of registers to introduce a time delay in the computation. This delay is often necessary to properly synchronize and pipeline the operations in the FPGA design to ensure that data propagates through each stage and is available for the subsequent stage at the correct time. Accurate prediction of future states requires precise timing coordination and accounting for system delays to capture the time-dependent nature of the environment[25].

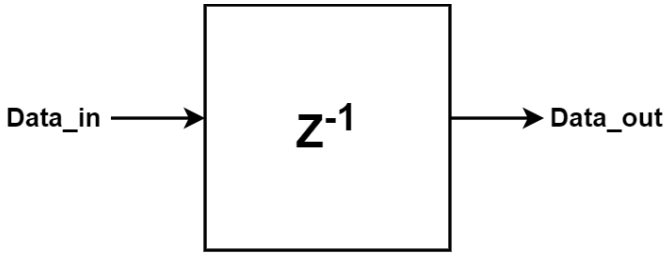


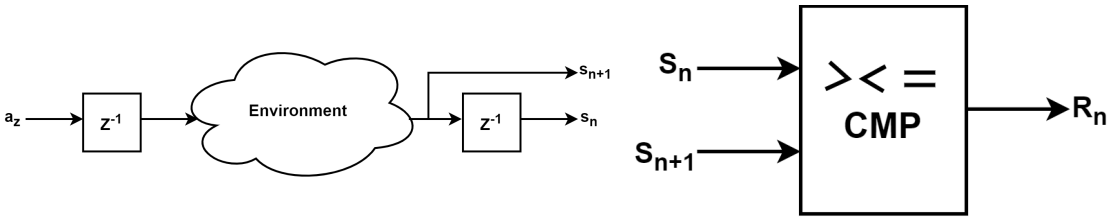
Figure 6.5: System Delay

6.4.3 Reward Function Module

The reward function r_n is a vector of Z elements, represents as

$$r^n = \begin{bmatrix} r^{n,0} \\ r^{n,1} \\ \vdots \\ r^{n,z-1} \end{bmatrix} \tag{6.1}$$

Compares the future state of the environment S_{n+1} with the current system state S_n registered in the memory then, The feedback can be in the form of a signal or data that is sent to the Q-learning module or directly used in the Q-value update process based on the comparison. Actions leading to the target state have a positive



numerical $r_{n,z}$ reinforcement value. Undesired actions in the state receive a negative numeric boost r_z^n. The actions that lead to other states receive $r_z^n = 0$.

6.4.4 Qn Module

In the Qn module, the calculation of the value function vector elements Q_n^z is performed according to equation 6.2 . It is a key component in implementing the Q-learning algorithm[25]. It performs calculations to update the Q-value for the current state-action pair based on various input parameters.

$$Q(s,a) = Q(s,a) + \alpha \cdot (r + \gamma \cdot \max Q(s',a') - Q(s,a)) \tag{6.2}$$

The updated Q-value $(Q_n + 1)$ is calculated using account the maximum discounted future reward (γmaxQ), the previous Q-value (Q_n), the learning coefficient (α) and the current reward (R_n). Each n-th module Qn computes the vector

$$Q_z^n = \begin{bmatrix} Q_0^n \\ Q_1^n \\ \vdots \\ Q_{z-1}^n \end{bmatrix} \tag{6.3}$$

The set of N vectors, Q_z^n forms the matrix value function $\mathrm{Q}(s_n, a_z)$ that can be expressed as

$$Q_z^n = \begin{bmatrix} Q_0^0 & Q_0^1 & \cdots & Q_0^{N-1} \\ Q_1^0 & Q_1^1 & \cdots & Q_1^{N-1} \\ \vdots & \vdots & \vdots & \vdots \\ Q_{z-1}^0 & Q_{z-1}^1 & \cdots & Q_{z-1}^{N-1} \end{bmatrix} \tag{6.4}$$

6.4.5 Q-Table

the Q-values represent the expected utility of taking a particular action in a specific state. These values are stored in a data structure called the Q-table. The Q-table is a two-dimensional table where the rows represent the states of the environment, and the columns represent the available actions. Each entry in the table corresponds to the Q-value of a particular state-action pair. The Q-values are updated during the learning process based on the rewards received and the transition to the next state. The size of the Q-table depends on the complexity of the environment. If the state and action spaces are discrete and relatively small, the Q-table can be represented as a simple matrix. However, in more complex environments with large or continuous state and action spaces, it becomes impractical to store all Q-values explicitly.

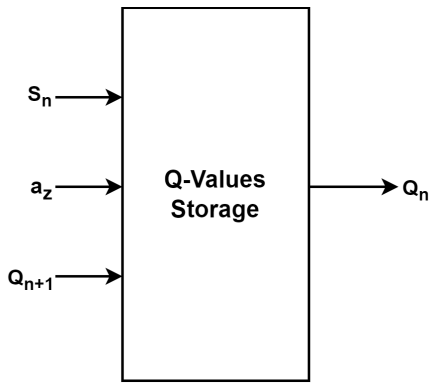


Figure 6.6: Synchronous RAM

6.4.6 MAXQ-Table

The maxQ storage module is crucial for our design, enabling efficient learning and optimal decision-making based on the highest Q-values observed for each state.

It acts as a memory element in Q-learning algorithms, storing and retrieving

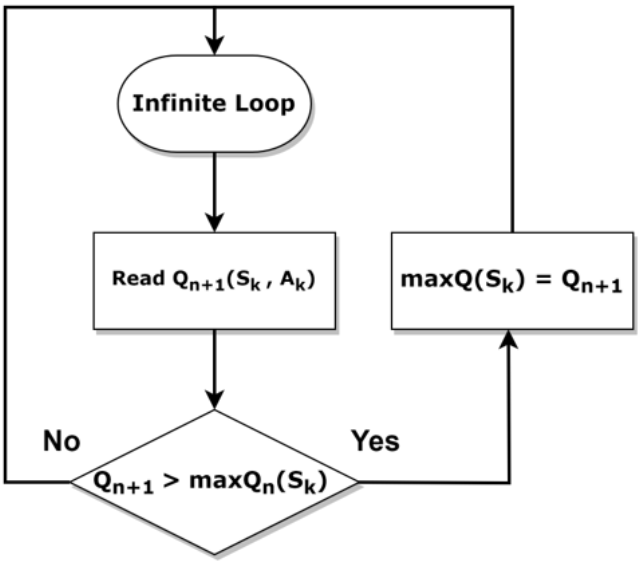


Figure 6.7: Flow chart of comparison and the storage process

the maximum Q-values associated with each state. This design ensures accurate storage and retrieval of the maxQ values, providing essential information for decision-making. By using a memory array, it organizes and facilitates easy access to the maximum Q-values for different states. According to figure 6.7, The module compares current Q-values with the stored maximum, updating it if the current value is greater. It then outputs the maximum Q-value for the next state, allowing for seamless tracking of the maximum Q-value as state transitions occur.

6.5 RTL Reconnaissance

6.5.1 PRNG Architecture

Inputs:

- CLK: The global system clock.
- RST: the global reset signal.

Outputs:

- Q_{n+1}: 4 Bit action word are generated from LFSR.

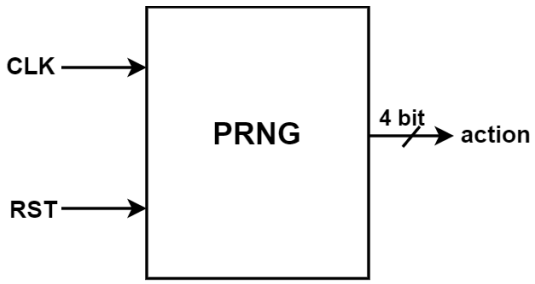


Figure 6.8: PRNG Ports

Designing a PRNG that produces high-quality random numbers requires careful consideration of the algorithm and the register size, in our case the use of a 16-bit register proved sufficient. For a 16-bit LFSR, to achieve maximum randomization,the random numbers calculated according to the equation $x^{16}+x^{14}+x^{13}+x^{11}+1$. The

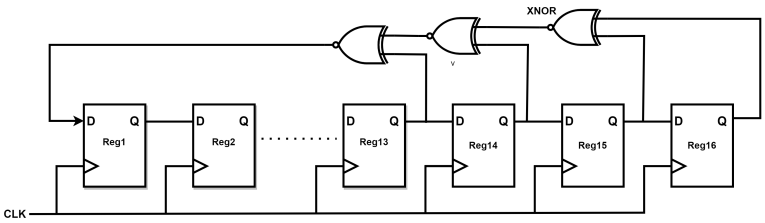


Figure 6.9: 16-bit Linear Feedback shift Register Internal Architecture

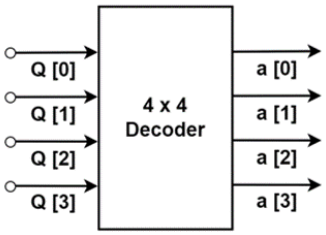


Figure 6.10: Decoder

output is generated by applying an XNOR operation to two or more previous values in the sequence. The XNOR operation is used to combine the bits of the previous values in a way that produces a new value that appears to be random. A decoder is used to convert the registers output into a set of randomly chosen action for each state.

6.5.2 Unit Delay

Inputs:

- Data_in: 8 Bits Data_in by default, we want to delay them by one-clock cycle.
- CLK: The global system clock.
- RST: the global reset signal.

outputs:

- Data_out: 8 Bits Data_out are delayed by one-clock cycle.

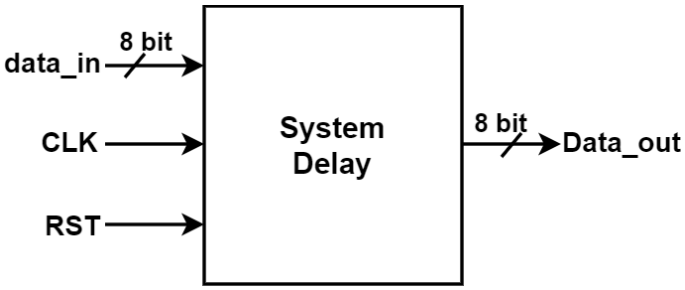


Figure 6.11: Unit Delay ports

The unit delay circuit can be implemented using flip-flops or registers, which are common building blocks in digital systems. Each flip-flop or register stores a single bit of information, and a chain of these elements can be used to store multiple bits simultaneously. 8-bit inputs, means that the input signal consists of 8 binary digits (bits), and the delay circuit preserves the values of these 8 bits for one clock cycle before propagating them as the output.

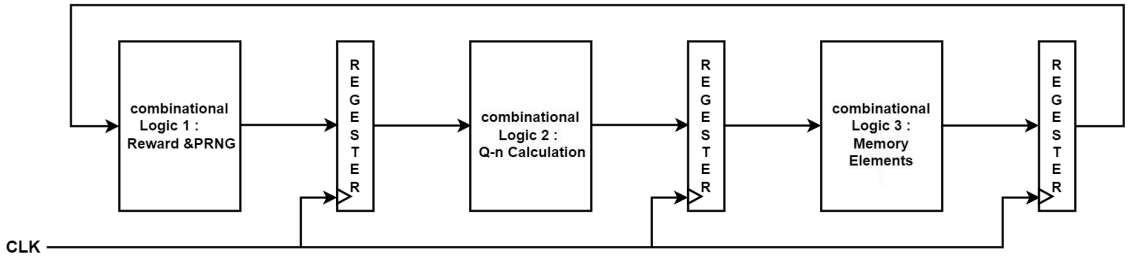


Figure 6.12: Unit Delay

6.5.3 Qn-Module

Inputs:

- γmaxQ: 14 Bit γmaxQ which it was generated from previous state to update Q-values during the learning process.
- R_n: 14 bit Reward which we entered to provide higher rewards for actions that bring the agent closer to achieving its goal or completing the task successfully.
- Q_n: 14 Bit Q_n which it was generated from $Q_nModule$ to update Q-values based on action applied on one state.
- α: 14 Bit learning rate we entered to determines the weight given to new information during the Q-value update.

outputs:

- Q_{n+1}: 14 Bit updated Q_n generated from Q_n module

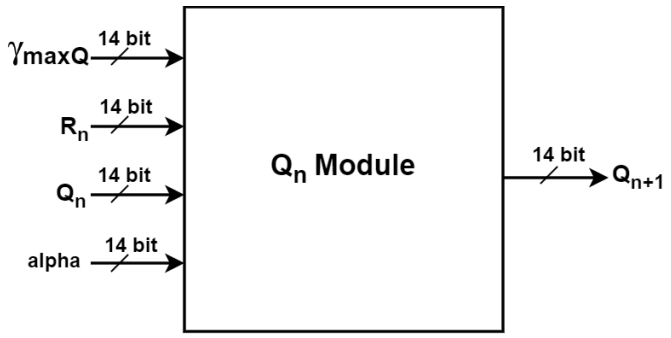


Figure 6.13: Q_n-Module Ports

Figure 6.14 show how to calculate the updated Q-value, the module utilizes Fixed Point arithmetic operations. With the current Q_n value, the immediate reward R_n, the maximum Q_n value from the next state, and the learning rate α and γmaxQ as inputs, the Q_n module performs the update calculation using arithmetic and logic operations. This calculation involves subtraction, multiplication, and addition operations to adjust the Q-value based on the observed reward and the estimated maximum future reward. After the update calculation, the modified Q-value is written back to the memory at the appropriate address. This involves activating the memory cell corresponding to the address and storing the updated Q-value.The Q_n module have output connections to provide the updated Q-value as output to other parts of the Q-learning system or to external components for further processing.

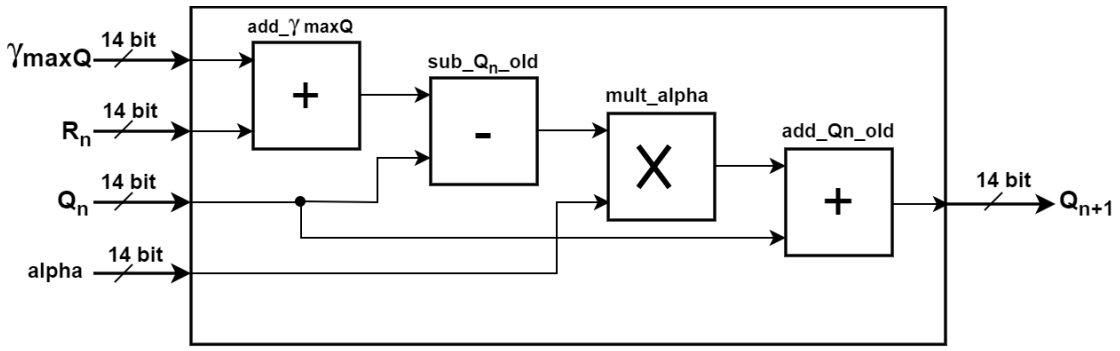


Figure 6.14: Q_n-Module Internal Architecture

6.5.4 Q-Table Module

Inputs:

- Q_{n+1}: 14 Bit Q_{n+1} which it was generated from Q_n module.
- addr: 5 bit addr depends on state and action.
- CLK: The global system clock.
- RST: The global reset signal.
- Read_en: Input determines whether the module is performing a read operation.
- Write_en: Input determines whether the module is performing a write operation.

outputs:

- Q_n: 14 Bit Q_n were stored in Q-table.

synchronous RAM (Random Access Memory) is commonly used to store and access Q-values efficiently. Synchronous RAM, often referred to as synchronous dual-port RAM (SDP RAM), allows simultaneous read and write operations based on the clock signal (CLK), reset signal (RST), Write_en and Read_en making it suitable for real-time Q-learning updates.

6.5.5 MAXQ_n-module Storage

Inputs:

- Q_{n+1}: 14 Bit updated Q_n generated from Q_n module.
- S_n: 4 Bit Current state.
- S_{n+1}: 4 Bit Future state.
- CLK: The global system clock.

outputs:

- maxQ_{n+1}: 14 Bit maxQ_n generated from maxQ_n storage module

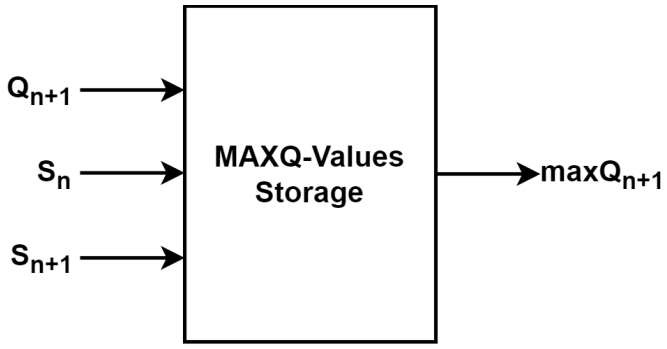


Figure 6.15: MAXQ_n Storage Ports

Figure 6.16 describes a behavioral module that manages a memory called $maxQ_n$ S_n stored. This memory consists of seven registers, each capable of storing a 14-bit value. Its purpose is to store the maximum Q-values for each state. During initialization, the code sets all elements in $maxQ_n$, S_n stored to zero in order to clear the memory. This initialization process only occurs once during simulation. Within one cycle, the code is checked. If the current Q_n value is greater than the maxQ$_n$ value stored for the current state (maxQ_n, S_n storedS_n, the code enters an if condition. This condition compares the current Q-value to the previously stored maximum for the current state. If the condition evaluates to true, the code updates the maxQ_n value for the current state by assigning Qn to $maxQ_n$, S_n stored S_n. This update occurs on the positive edge of the clock.Finally, the code assigns the maxQ_n value stored for the next state maxQ_n, S_n storedS_{n+1} to the output register maxQ_n, S_n. This enables the module to output the maxQ_n value for the future state. Improper Initialization, Incorrect Comparisons or Conditions and Timing Violations or Synchronization Issues may have Potential Issues.

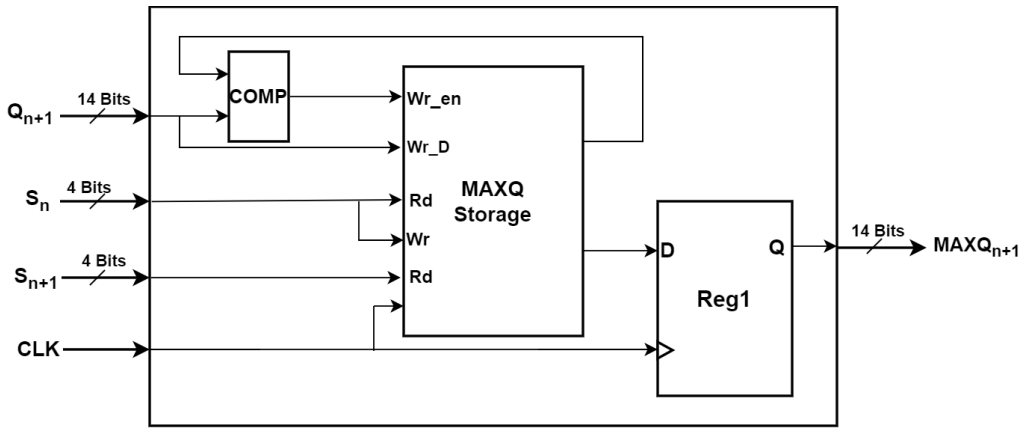


Figure 6.16: MAXQ_n Internal Architecture

6.6 Design Flow and Implementation

Design synthesis is an important step in FPGA implementation. Synthesis optimization plays a vital role in modern FPGAs in order to achieve high performance, in terms of resource utilization and reducing time-consuming test processes. The synthesis tools optimize HDL code for both logic utilization and performance of an intended design. In FPGA each slices, LUT and register utilization are very important in order to accommodate larger design unit. The synthesis step creates netlist files from the various source files. The netlist files can serve as input to the implementation module. After generating the netlist file (synthesis step), the implementation will convert the logic design into a physical file that can be downloaded on the target device (e.g. Altera FPGA). This step involves three sub-steps: Translating the netlist, Mapping and Placement.

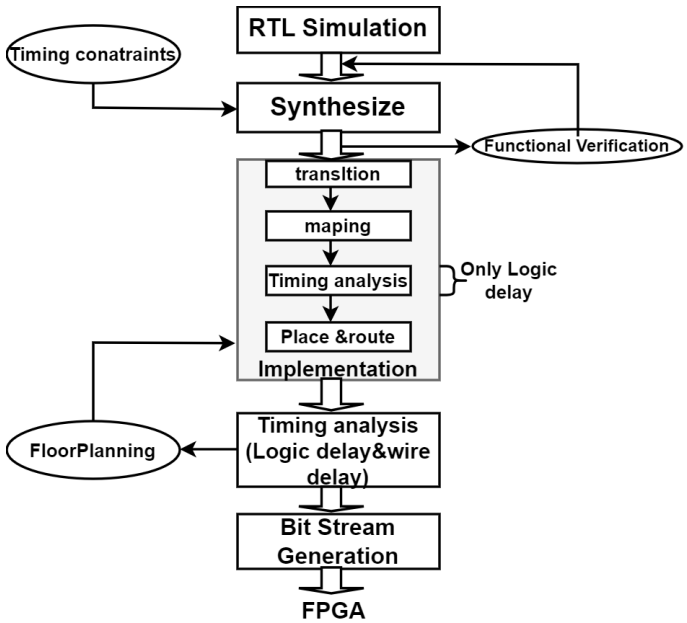


Figure 6.17: FPGA Synthesis Flow

6.6.1 Synthesis

In the synthesis step, a high-level hardware description language (HDL) code, such as VHDL or Verilog, is transformed into a logical gate-level representation. This step involves converting the behavioral description of the design into a structural representation of the underlying hardware components.

6.6.2 Translation

Once the synthesis step is complete, the gate-level representation is translated into a format that can be understood by the FPGA toolchain. This format is typically a standard netlist format like Electronic Design Interchange Format (EDIF) or Design Exchange Format (DEF).

6.6.3 Mapping

The mapping step involves mapping the logical gates from the gate-level representation onto the available resources of the target FPGA device. This process determines how the design will be implemented in terms of the specific configurable elements, such as lookup tables (LUTs), flip-flops, and interconnect resources, present in the FPGA.

6.6.4 Timing Analysis

Static timing analysis refers to using delays extracted from physical implementation to analyze timing directly rather than through simulation. There are three important timing parameters associated with a register. **The set-up time (t_{su})** is the time that the data inputs (D input) must be valid before the clock transition (this is, the 0 to 1 transition for a positive edge-triggered register). **The hold time (t_{hold})** is the time the data input must remain valid after the clock edge.the data at the D input is copied to the Q output after a worst-case propagation delay (with reference to the clock edge) denoted by t_{c-q}.

In FPGA After Place and route, we have a fully routed physical design and a timing analysis that can extract timing and check for any timing violations (setup, hold,etc...) associated with any of the internal registers. Typically pessimistic delay assumptions are made to arrive at a worst-case model – a data-driven simulation may reveal what delays are actually relevant in a design.

6.6.4.1 Static Timing Analysis (STA) Objectives

Static Timing Analysis (STA) is a crucial step in digital design verification that aims to determine the worst-case arrival time of signals at all pins of design elements.

This reduction in complexity can be achieved by making certain assumptions that trade off some accuracy.

One common pitfall in STA is the assumption of implied synchronicity of clock domain paths. It is essential to be cautious when dealing with multiple clock domains and accurately account for the timing relationships between them to avoid incorrect results.

STA relies on the concept of arrival times (ATs) and required arrival times (RATs) to determine if any path in the design violates the specified timing constraintsBy constructing a timing graph that represents the design using delay arcs and performing checks based on ATs and RATs, STA can identify any timing violations that may exist.

To validate the STA results, a sanity check is necessary, comparing the expected results with the actual ones. This verification step helps identify any discrepancies and ensures the correctness of the timing analysis.

Uses timing graph of delay arcs and checks to represent the design.

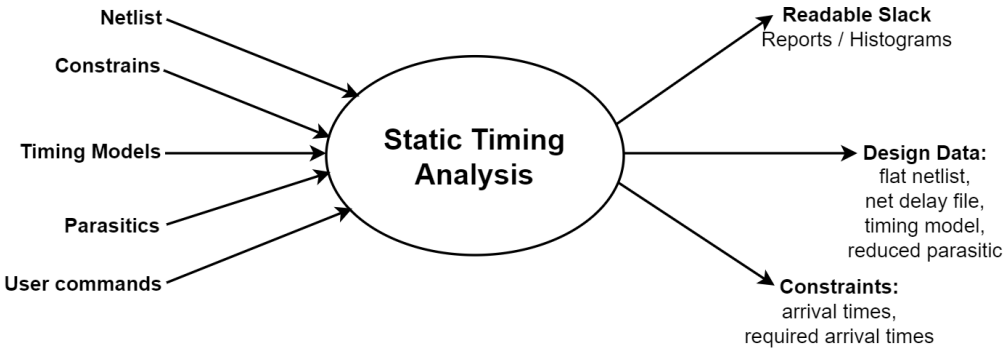


Figure 6.18: STA Objectives

6.6.4.2 Timing Analysis (Pre-Place & Route)

Before the actual placement and routing process, a timing analysis is performed to estimate the delays and ensure that the design meets the required timing constraints.

This analysis helps identify potential timing violations, such as setup and hold time violations, and guides subsequent steps to optimize the design for timing performance.

6.6.5 Placement & Route

In this step, The placement stage assigns individual logic components (gates, flip-flops, etc.) to specific locations on the FPGA chip. The placement process aims to minimize the interconnect delays and optimize for various factors such as timing, power, and area. During placement, the FPGA tools determine the optimal placement of the design components based on the available resources and constraints.

After placement, the routing stage connects the placed components by config-

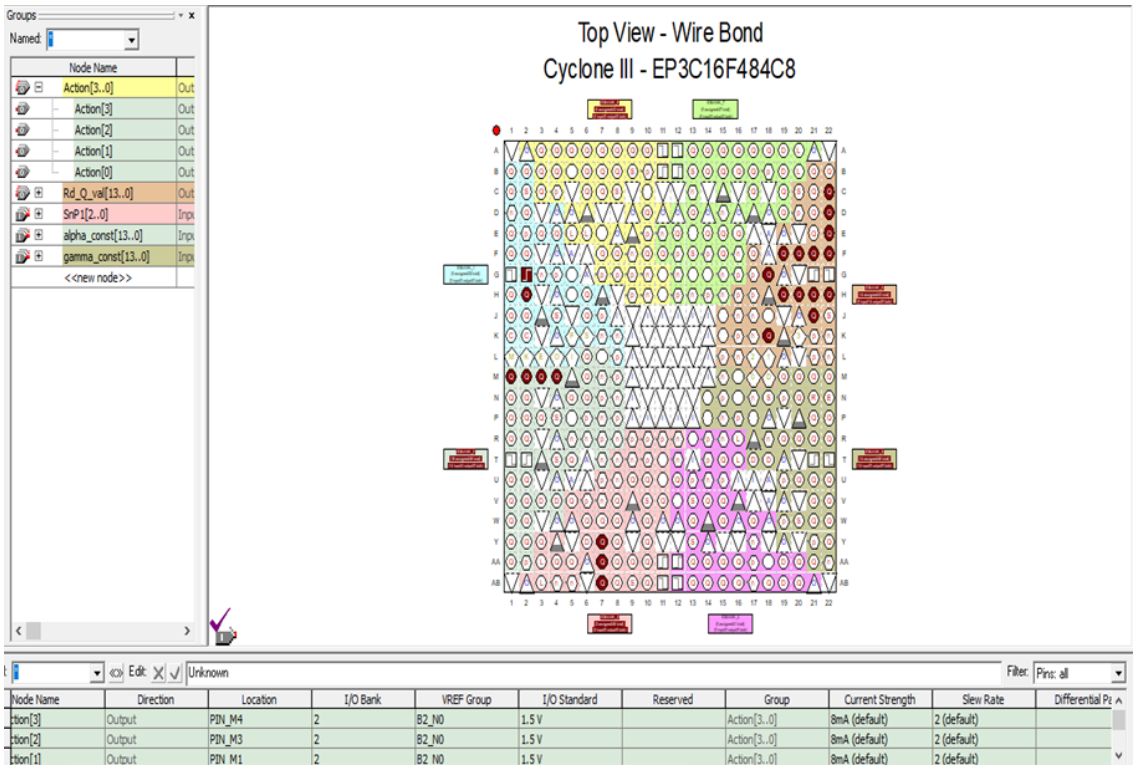


Figure 6.19: Location Assignments

uring the programmable routing resources on the FPGA. The routing process determines the paths for interconnecting the logic components while considering constraints such as timing, signal integrity, and resource availability. It involves determining the appropriate routing resources, setting up the routing connections, and configuring the interconnects on the FPGA.

6.6.6 Timing Analysis (Post-Place & Route)

After the placement and routing process, a final timing analysis is performed to evaluate the actual delays in the design and verify if the timing constraints are still met. This analysis takes into account the physical placement and routing details, and any necessary optimizations or iterations are made to meet the desired timing goals.

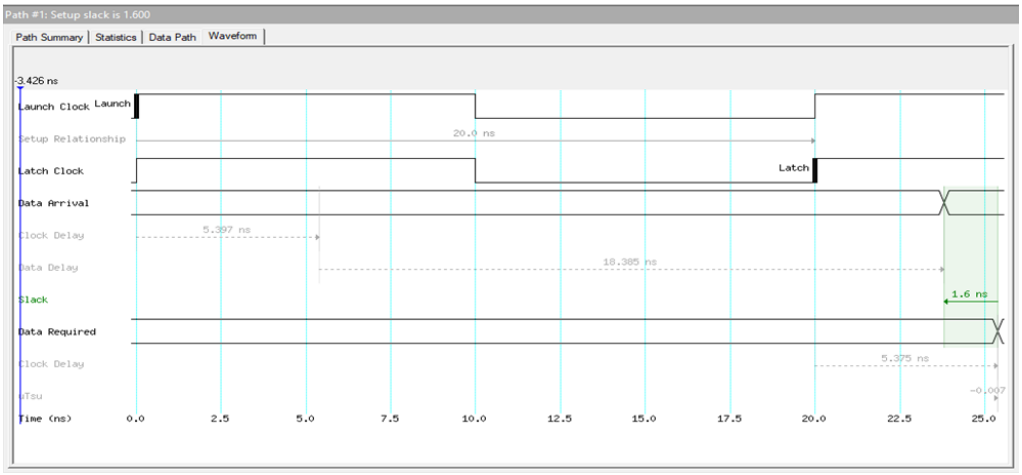


Figure 6.20: setup timing analysis

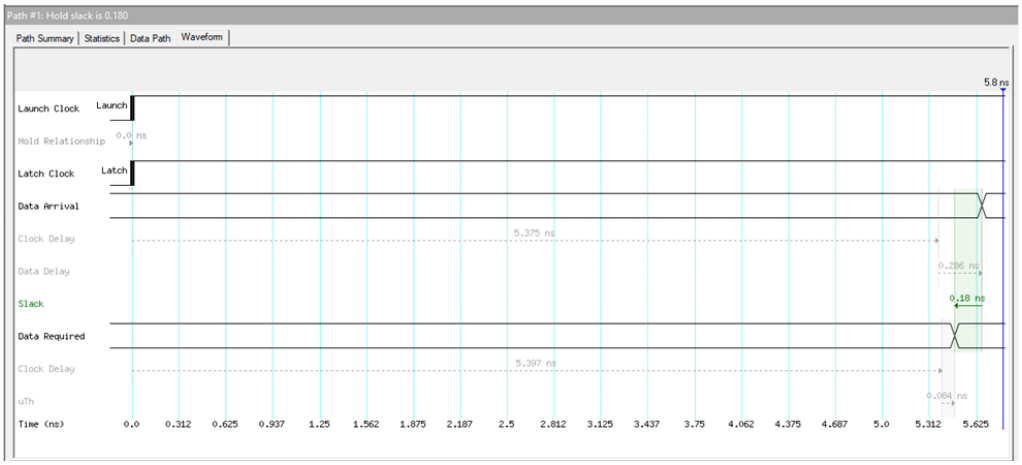


Figure 6.21: hold timing analysis

In our design according to figures 6.20 and 6.21,the design has a positive slack of 1.6ns for setup time and 0.18ns for hold time. Positive slack indicates that the design has more setup time available than required for the critical paths and meet the timing requirements with 1.6ns of margin and more hold time available than required for the critical paths and meet the timing requirements with 0.18ns of margin.

By analysing the set-up time, hold time, and maximum frequency reports, we can assess the timing behaviour of the circuit and ensure that it meets the required timing constraints for accurate and reliable data capture.

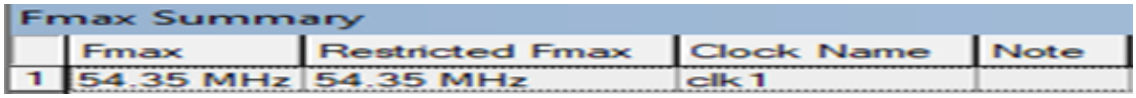


Figure 6.22: Report of Maximum Frequency

Device	EP3C16F484C8
Timing Models	Final
Met Timing Requirements	Yes
Total Logic Elements	410/15,408(<3%)
Total Combinational Function	404/15,408(<3%)
Dedicated Logic Registers	124/15,408(<1%)
Total Registers	124
Total Pins	51/347(15%)
Total Virtual Pins	0
Total Memory Bits	1,792/516.096(<1%)
Embedded Multiplier 9-bits Elements	4/112(4%)
Total PLLs	0/4(0%)

6.6.7 Bitstream Generation

The last step involves generating the configuration bitstream file, which contains the information required to program the FPGA device. The bitstream file contains the settings for each configurable element on the FPGA, such as LUT configurations, interconnect routing, and I/O pin assignments. The generated bitstream can then be loaded onto the FPGA to configure it with the desired design functionality. These steps collectively form the FPGA design flow, enabling the transformation of a high-level hardware description into a bitstream that configures the FPGA to implement the desired digital circuit.

6.6.8 Power Estimation and Analysis

The "toggle rate" and "static power" properties in PPA (Power, Performance, and Area) analysis for Cyclone 3 FPGAs in Quartus 2 refer to specific characteristics of the device.

Toggle Rate (also known as Toggle Power or Dynamic Power) represents the average rate at which the signals in the design change their state. It measures the frequency at which the logic elements (LEs) or flip-flops within the FPGA toggle between '0' and '1'. Higher toggle rates result in increased power consumption due to the switching activity. The value of 12.5 in this context likely indicates that, on average, each LE or flip-flop in the Cyclone 3 FPGA toggles 12.5 times per clock cycle, according to figure 6.23.

Static Power refers to the power consumed by the FPGA when there is no switching activity or toggling of signals. It represents the power required to maintain the state of the device even when there are no inputs changing. Static power consumption is constant and independent of the activity in the design.

The values of 12.5 mentioned for both toggle rate and static power properties are not commonly associated with a specific unit of measurement. Therefore, it is important to refer to the specific documentation or user guide provided by the FPGA manufacturer (Intel/Altera) or Quartus 2 software for a more accurate understanding of the toggle rate and static power characteristics of the Cyclone 3 FPGA.

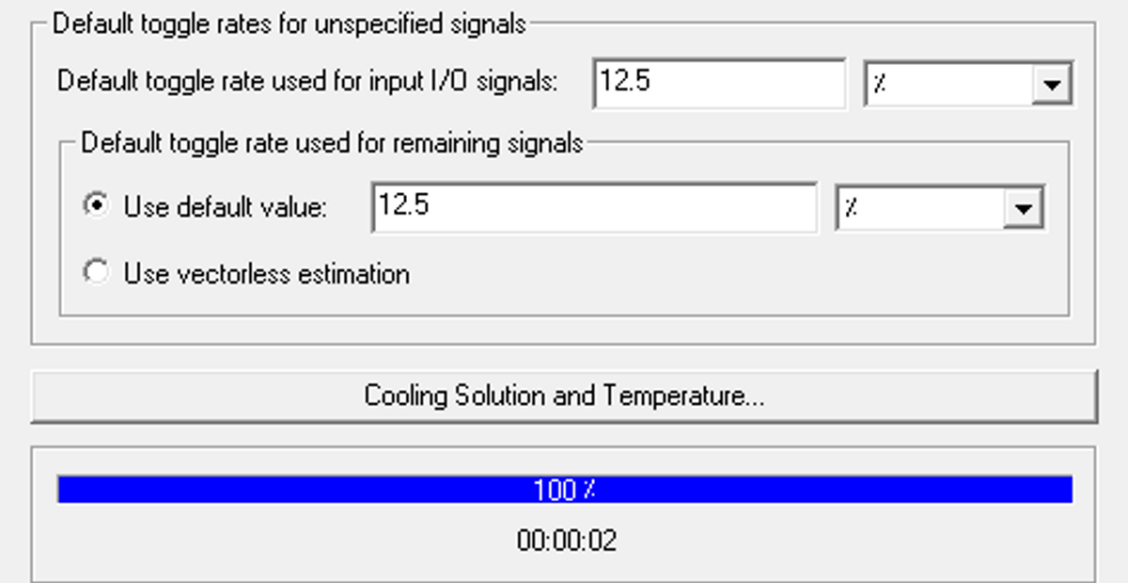


Figure 6.23: Toggle Rate

Power estimation is performed to estimate the power consumption of the Q-learning implementation on the FPGA. The FPGA tools analyze the design's characteristics, including the number of resources used, switching activities, and other factors affecting power consumption. The FPGA tools generate power analysis reports that provide detailed information about the power consumption of the design. These reports include metrics such as dynamic power, static power, and total power consumed by the FPGA during operation. According to report shown in figure **??**, The power analysis reports indicate that the Q-learning implementation on the FPGA has a dynamic power consumption of 117.48mW, static power consumption of 52.03W, input/output thermal power consumption 41.67 and a total power consumption of 211.19mW. That is mean, The design meets the power estimates and stays within the specified power constraints.

PowerPlay Power Analyzer Status	Successful - Wed Jul 12 18:53:08 2023
Quartus II Version	9.0 Build 132 02/25/2009 SJ Web Edition
Revision Name	Qn_TOP
Top-level Entity Name	Qn_TOP
Family	Cyclone III
Device	EP3C16F484C8
Power Models	Final
Total Thermal Power Dissipation	211.19 mW
Core Dynamic Thermal Power Dissipation	117.48 mW
Core Static Thermal Power Dissipation	52.03 mW
I/O Thermal Power Dissipation	41.67 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Figure 6.24: Power Estimation and Analysis

6.7 Testing and Verification

Testing is often confused with verification. The purpose of testing is to verify that the design was manufactured correctly. The purpose of the verification is a process used to demonstrate the correctness of a design and ensure that a design meets its functionality.

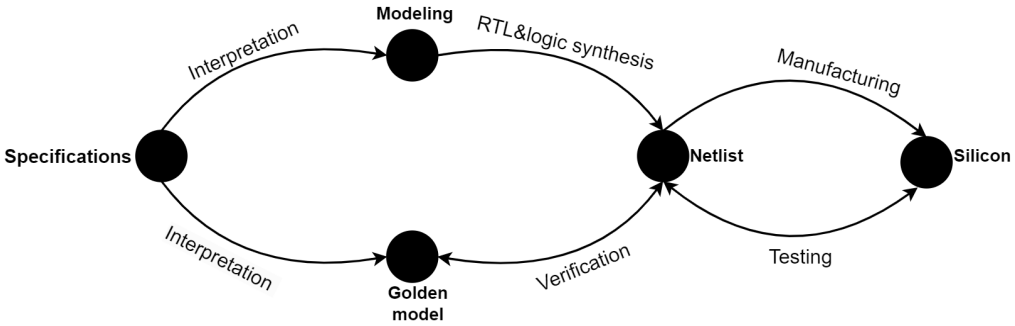


Figure 6.25: Testing versus Verification

6.7.1 Verification aspects

6.7.1.1 Functional verification

Functional verification is a crucial step in ensuring the correct functionality of the designed circuit according to its intended specifications. This verification process encompasses several important steps:

Testbench Development
The first step involves developing a comprehensive testbench that provides inputs to the digital circuit and verifies the outputs against the expected results. The testbench acts as a stimulus to thoroughly test the functionality of the circuit.

Test Case Generation
To ensure thorough testing, a set of test cases is generated to cover various input combinations and scenarios. These test cases aim to exercise different aspects of the circuit's functionality and assess its behavior under diverse conditions.

Simulation
The testbench is used to simulate the behavior of the digital circuit using a hardware description language (HDL) simulator. The simulated results are compared against the expected outputs to verify the correctness of the circuit's functionality.

Code Coverage Analysis To ensure that all parts of the digital circuit design are exercised by the test cases, code coverage analysis is performed. This analysis examines the extent to which the test cases exercise the circuit's code, identifying any areas that may require additional testing.

6.7.1.2 Timing verification

Timing verification focuses on ensuring that the designed circuit meets its timing requirements and operates within the specified timing constraints. The timing verification process involves the following steps:

Static Timing Analysis (STA)
STA is performed to analyze the timing paths within the circuit. It estimates signal arrival times, setup/hold times, and clock requirements to ensure that the circuit's timing specifications are met.

Timing Constraints Specification
The required timing constraints, such as input/output delays, clock frequencies, and maximum propagation delays, are defined and specified for the circuit. These constraints serve as guidelines for proper operation within the desired timing parameters.

Timing Simulation
To verify compliance with the specified timing constraints, the circuit's behavior is simulated while considering these constraints. This simulation ensures that all signals and operations meet the required setup/hold times and operate within the desired clock frequencies.

Clock Domain Crossing Verification
Signals crossing different clock domains are verified for correct functionality. This verification step ensures that proper synchronization techniques are employed to handle signals transitioning between different clock domains.

6.7.1.3 Performance verification

Performance verification involves evaluating the performance characteristics of the designed circuit, such as throughput, latency, power consumption, and resource utilization. The performance verification process encompasses the following steps:

Performance Metrics Definition
Performance metrics that need to be evaluated, such as maximum achievable throughput, minimum latency, or power consumption targets, are defined to assess the circuit's performance.

Performance Analysis
The circuit's behavior and performance are analyzed using techniques like profiling, simulation, or emulation. This analysis measures the defined performance metrics of interest, providing insights into the circuit's efficiency and effectiveness.

Resource Utilization Analysis
The utilization of various resources, such as flip-flops, lookup tables, or memory blocks, is assessed to ensure optimal resource allocation and identify potential bottlenecks or inefficiencies. This analysis helps identify areas for improvement and optimization.

Optimization Techniques
Based on the identified bottlenecks or resource utilization analysis, optimization techniques such as pipelining, parallelism, or resource sharing can be applied to improve the circuit's performance. These techniques aim to enhance throughput, reduce latency, optimize power consumption, and improve overall resource utilization.

By rigorously performing functional verification, timing verification, and performance verification, the designed digital circuit can be ensured to function correctly, meet timing requirements, and exhibit efficient performance characteristics such as throughput, latency, power consumption, and resource utilization. These verification processes contribute to the overall reliability and effectiveness of the circuit in the specific field or domain.

6.7.2 Verification Approaches

Hardware verification can be done using three different approaches: black-box, white-box, and grey-box.

Blackbox Verification
Blackbox verification is performed without any knowledge of the actual implementation of a design. All verification must be accomplished through the available interfaces, without direct access to the internal state of the design, without knowledge of its structure and implementation. This method suffers from an obvious lack of visibility and controllability. However, the advantage of black-box verification is that it does not depend on any specific implementation. Whether the design is implemented in a single ASIC, multiple FPGAs, or a circuit board.

White-box Verification
White-box verification has full visibility and controlability of the internal structure and implementation of the design being verified. This method has the advantage of being able to quickly set up an interesting combination of states and inputs or isolate a particular function. It can then easily observe the results as the verification progresses and immediately report any discrepancies from the expected behavior.

Grey-box Verification
Grey-box verification is a compromise between black-box and white-box verification. It has some visibility into the internal structure and implementation of the design being verified but not full visibility. Test cases may not be relevant on another implementation.

6.7.3 Model Verification (White box)

Model verification involves testing and verifying the functionality of each individual module within a system. It focuses on the internal structure and behavior of the modules and ensure that each module functions correctly.

6.7.3.1 PRNG verifying

The white-box verification of PRNG module involves designing and executing test cases that cover various input scenarios and expected outputs.

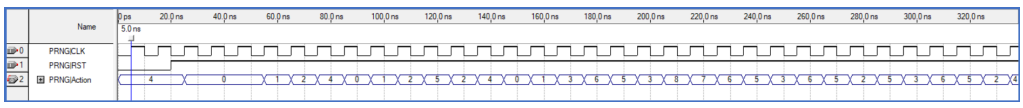


Figure 6.26: PRNG Testbench

6.7.3.2 Reward Verifying

Reward module interacts with PRNG module and performs additional processing based on the input received from PRNG module. The white-box verification of reward module focuses on testing its internal logic, interfaces with other modules, and any specific functionality it provides.

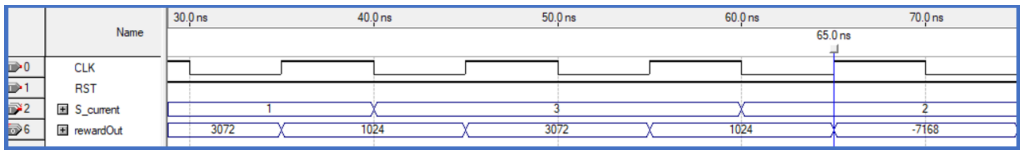


Figure 6.27: Reward Testbench

6.7.3.3 Q-table Verifying

Qn module is responsible for write updated Q-value and Read previous value based on the inputs received from Qn Module. The white-box verification of Q-table module ensures that the read and write process behave correctly and produces the desired output.

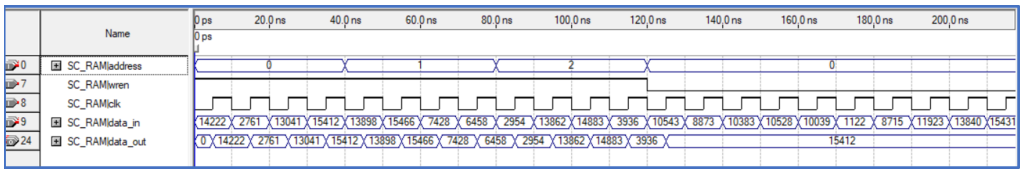


Figure 6.28: Q-table Testbench

6.7.3.4 maxQ-table Verifying

maxQ-table module is responsible for generating the maxQ based on the inputs received from Modules Qn and Reward. The white-box verification of maxQ-table module ensures that it correctly combines the outputs from Modules Qn and Reward and produces the desired final output.

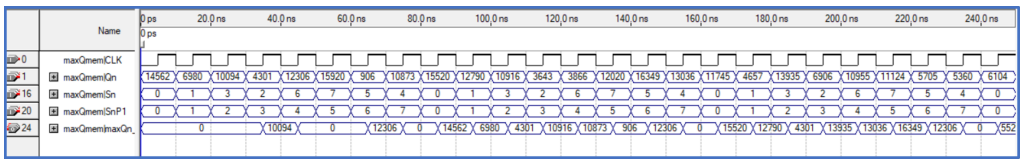


Figure 6.29: maxQ-table Testbench

6.7.4 System Verification

System verification(Black-box verification) refers to the process of verifying and validating a complete system to ensure that it meets the specified requirements and functions as intended. Black-box verification typically involves test case generation, testing the Device Under Test (DUT), comparing the actual outputs with expected outputs, and logging values in case of mismatches.

6.7.4.1 Test Case Generation

Test case generation involves creating a set of test scenarios or inputs that cover various aspects of the system's functionality. Test cases should be designed to exercise different features, boundary conditions, and error cases of the system.

6.7.4.2 Device Under Test (DUT)

The Device Under Test (DUT) is the system or component being tested, treated as a black box. The DUT receives inputs and produces outputs based on its internal processing.

6.7.4.3 Comparisons

The actual outputs generated by the DUT are compared with the expected outputs. Comparisons are performed to check if the DUT's behavior matches the desired functionality.

6.7.4.4 Logging in Case of Mismatch

When a mismatch occurs between the actual and expected outputs, it is logged for further investigation and analysis. Logging the mismatch helps identify the specific test case and provides information for debugging and resolving issues.

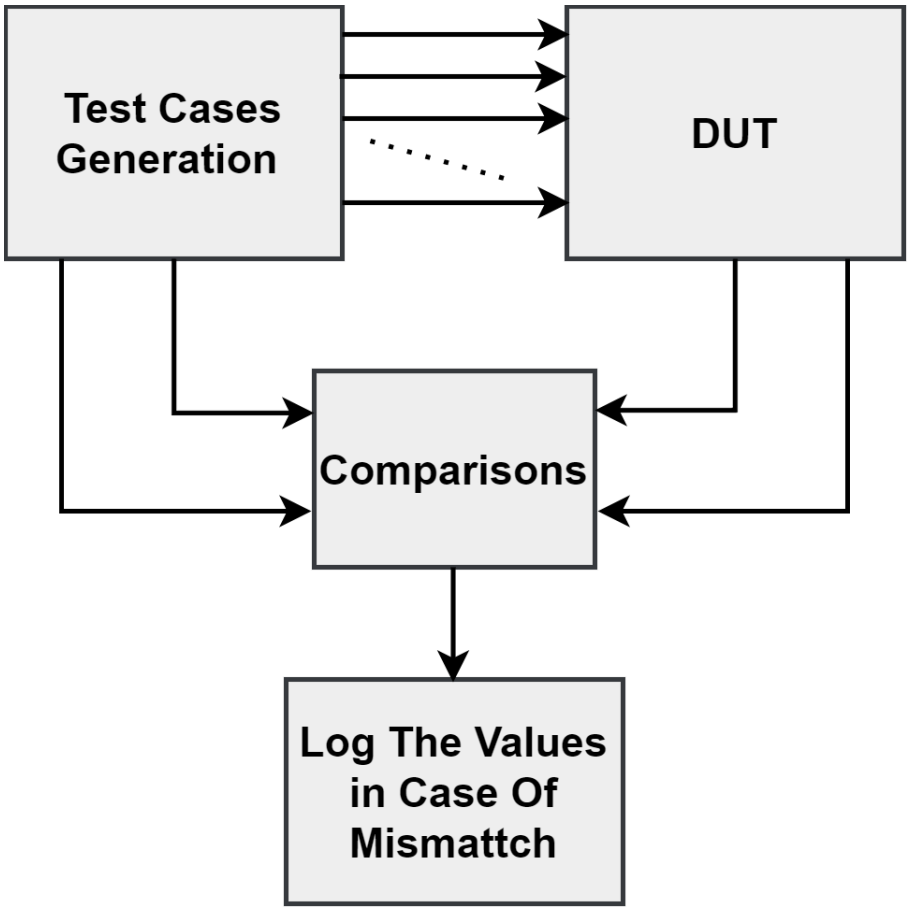


Figure 6.30: The Black-box Technique for Design Testing

Chapter 7

Discussion and Conclusion

7.1 Summary

Chapter one of this book provides an introduction to the research topic, which focuses on addressing the speed limitations of reinforcement learning (RL) algorithms in self-healing networks, specifically in the context of cell outage compensation (COC) in 5G communication systems. The chapter highlights the background and context of mobile network evolution, the challenges of network management in complex networks, and the importance of network management automation. It also discusses the concept of self-organizing networks (SONs) and their sub-functions, including self-healing.

The problem statement is identified as the speed limitations of RL algorithms, which hinder real-time decision-making in self-healing networks, particularly in URLLC use cases like autonomous vehicles. The research objective is to develop an efficient and accelerated approach for connection restoration in the case of cell outages by leveraging RL algorithms and hardware acceleration techniques.

The proposed solution consists of two main parts. The first part involves developing a MATLAB-based model to understand the COC problem and the behavior of the Q-Learning algorithm. The model will provide visual representations and allow for experimentation with different scenarios and parameters. The second part focuses on implementing the Q-Learning algorithm on a field-programmable gate array (FPGA) to overcome speed limitations. FPGA offers high-performance computing capabilities and parallel processing, making it suitable for real-time decision-making.

The chapter also discusses different approaches to accelerate AI algorithms, including hardware acceleration (such as selecting appropriate processors, parallelization, and FPGA implementation) and software acceleration (such as pruning, quantization, and workflow optimization).

Related works and literature reviews are presented, highlighting research on cell outage compensation using Q-learning and parallel implementation of Q-learning on FPGAs.

The chapter concludes by outlining the organization and structure of the thesis, which includes an exploration of network management challenges, RL in network management automation, COC modeling in MATLAB, FPGA implementation of Q-learning, and a summary of key findings.

In summary, this chapter provides an overview of the research topic, including the background, problem statement, research objective, proposed solution, and literature review. It sets the stage for the subsequent chapters that delve deeper into the research aspects of accelerating RL algorithms for cell outage compensation in self-healing networks.

Chapter 2 provides an overview of the evolution of mobile networks and the complexities associated with network management. It begins by discussing the development of mobile network generations, starting from 1G to the most recent 5G. Each generation brought advancements in voice and data communication, with 4G focusing on mobile broadband services. The chapter emphasizes the shift towards cloud-native networks in the 5G era, where network architecture is designed to support cloud services and enable flexibility.

The chapter explores the understanding of communication network complexity, highlighting the sources of complexity such as the network protocol stack, scale, and connectivity. It emphasizes the need for effective network management algorithms to handle the complexity and ensure efficient network operations.

The principles of network management are introduced, with a focus on the FCAPS framework (Fault, Configuration, Accounting, Performance, and Security). These functional areas encompass various management services aimed at maintaining quality for telecommunications service customers.

The chapter further presents a taxonomy for cognitive autonomous networks, distinguishing between automation, autonomy, self-organization, and cognition. It discusses the levels of network automation, ranging from manual control to cognitive autonomous networks (CAN) that combine cognitive and autonomous abilities to reason, recommend, and execute actions.

Overall, Chapter 2 provides a comprehensive understanding of the development of mobile networks, the complexities of network management, and the concept of automation and autonomy in network operations. This foundational knowledge sets the stage for further exploration of self-healing networks and the application of reinforcement learning algorithms in Chapter 3 and beyond.

Chapter three discusses the concept of self-healing in mobile communication networks, specifically focusing on coverage area optimization. It introduces the idea of self-organizing networks (SONs) as a solution to manage the complexity of network management and automate various network operations.

The chapter begins by explaining the need for network management automation and the limitations of traditional manual processes. It then introduces SON architecture types, including centralized SON, distributed SON, and hybrid SON, and discusses their key points and benefits.

The chapter further explores the self-configuration phase, which involves the deployment and initial configuration of network elements. It emphasizes the importance of automation in streamlining the deployment process, particularly in heterogeneous network deployments.

Self-optimization is another aspect covered in the chapter, focusing on optimizing network parameters during the operational stage of a mobile network. It discusses the need for continuous adjustment of network configuration parameters to accommodate changes in the network and its environment.

The main focus of the chapter is self-healing, which deals with failure detection and recovery in mobile networks. It explains that self-healing functions aim to maintain the network in a healthy state by constantly monitoring system performance, detecting anomalies, diagnosing root causes, and taking corrective actions. It also introduces the concept of cognitive autonomy and resilience in self-healing networks.

The chapter highlights the basic building blocks of self-healing, including rule-based systems, machine learning techniques, and the four-step self-healing process: profiling, anomaly detection, diagnosis, and corrective actions.

A self-healing framework for mobile networks is presented, consisting of three stages: outage detection, outage diagnosis, and outage compensation. It explains the importance of detecting and diagnosing network outages accurately and proposes various compensation techniques such as coverage area optimization, SINR optimization, cell capacity optimization, and spectral efficiency optimization.

The chapter concludes by discussing the key components of self-healing techniques, including the methodology, network topology, performance metrics, control mechanisms, and the direction of control. It also presents a taxonomy for self-healing concepts and provides a qualitative comparison of cell outage detection algorithms with a focus on coverage optimization.

chapter four introduces the concept of reinforcement learning (RL) as a tool for end-to-end cognition in network management automation. It begins by providing an overview of RL and its basic definitions. The Q-learning algorithm, a popular approach in RL, is discussed, along with its iterative process of updating the Q-values based on observed rewards.

The chapter highlights the challenges associated with RL, such as balancing exploration and exploitation, credit assignment, partial observability, high-dimensional state and action spaces, sample efficiency, transfer learning, and safety and ethical considerations. These challenges need to be addressed to advance the field of RL and enable its successful application in various domains.

The need for hardware acceleration in RL is also emphasized. Hardware acceleration techniques, such as GPUs, FPGAs, and ASICs, offer improved computational efficiency, real-time decision-making, scalability, energy efficiency, customization, and deployment in edge devices. These advantages enable faster learning, efficient inference, and the deployment of intelligent agents in real-world applications.

Chapter 5 focuses on the implementation, validation, and optimization of the Cell Outage Compensation (COC) model. The chapter begins by presenting the MATLAB function "evaluation," which evaluates the performance of the system based on two metrics: AGap (Number of unconnected users) and AOverlap (Number of connected users). The function iterates through each user and transmitter, calculates the received power, and updates the counters based on the power level. The ratios of AGap and AOverlap are then calculated.

Next, the chapter introduces the "q_learning" function, which implements Q-learning in the COC model. The function starts by evaluating the system's performance using the "evaluation" method and determining the current state based on the AGap value. It then enters a loop that iterates over episodes and steps. In each step, an action is chosen based on an epsilon-greedy policy, applied to the system, and the evaluation is performed again to obtain new AGap and AOverlap values. The reward is assigned based on the change in AGap and AOverlap, and the Q-value for the previous state-action pair is updated using the Q-learning update equation.

The chapter also discusses the input and output handling in the model. It presents a table of input parameters, such as tilt, frequency, environment, transmitted power, number of users, and inter-site distance. The output data includes graphs that show the changes in AGap, power, and tilt with each step, allowing for visualization and analysis of the system's behavior.

Furthermore, the chapter briefly mentions the GUI design of the model using MATLAB App Designer. App Designer provides a visual development environment for creating interactive applications with a GUI, simplifying the design process and integrating MATLAB's capabilities.

The chapter concludes with sections on model validation and optimization. Validation is performed through random/monkey testing, where random inputs are generated to simulate real-world scenarios and assess the model's behavior. The goal is to uncover any unexpected behaviors or issues and ensure the model's accuracy and reliability. Model optimization is discussed in terms of the epsilon (exploration factor), discount factor, and learning rate. These parameters affect the exploration-exploitation balance, consideration of future rewards, and the rate of learning in the Q-learning process.

Chapter 6 of the document presents a comprehensive overview of the design and implementation process of an FPGA-based Q-learning hardware system for COC. The chapter covers various crucial aspects, including requirement gathering, system partitioning, module function definitions, RTL (Register Transfer Level) implementation, and verification.

Requirement gathering involves the collection and understanding of specifications and constraints for the hardware system. Functional requirements, such as efficient Q-learning algorithm calculation, action discovery, reward updates, and real-time decision-making capabilities, are defined. Performance requirements include achieving speedup compared to software-based implementations, efficient processing of large states and actions, and minimizing latency.

System partitioning involves breaking down the system into smaller components or modules. Design exploration explores different design alternatives, including parallel and series implementations. Parallel implementation allows for concurrent processing, increased speed, and higher throughput, while series implementation offers simplicity and reduced resource usage. The chapter specifically focuses on the series implementation approach, which enables precise control and reproduction of the learning process.

Module function definitions are provided for several modules utilized in the Q-learning implementation. These modules include the PRNG (Pseudorandom Number Generator) module for generating random actions, the unit delay module for introducing time delays, the reward function module for calculating rewards, the Qn module responsible for updating Q-values, the Q-table for storing Q-values, and the MAXQ-table for tracking maximum Q-values.

The RTL implementation is discussed in detail, highlighting the architecture and design of specific modules. For example, the PRNG module employs a 16-bit LFSR (Linear Feedback Shift Register) algorithm for generating random actions. The unit delay circuit utilizes flip-flops or registers to introduce time delays. The Qn module updates Q-values based on inputs such as previous Qn values, immediate rewards, maximum Qn values from the next state, and the learning rate. The Q-table module employs synchronous RAM for efficient storage and access of Q-values. The MAXQ-table module stores the maximum Q-values for each state.

The chapter emphasizes the critical role of verification in the design process. Functional verification involves the development of a testbench, generation of test cases, simulation, and code coverage analysis. Timing verification ensures that the design meets timing requirements through activities such as static timing analysis, specification of timing constraints, timing simulation, and verification of clock domain crossings. Performance verification focuses on evaluating metrics such as throughput, latency, power consumption, and resource utilization.

7.2 Conclusion

We have successfully completed the hardware implementation for Q-learning, a widely used reinforcement learning algorithm. This accomplishment signifies a significant advancement in the field of artificial intelligence and holds promising potential for real-world applications.

Throughout the implementation process, we capitalized on our understanding of Q-learning and hardware design principles to create a specialized system capable of efficiently executing Q-learning algorithms. Our series implementation offers accelerated processing capabilities specifically tailored for reinforcement learning tasks, resulting in faster and more efficient training of intelligent agents. Additionally, we successfully executed the design flow and implementation process, including synthesis, mapping, timing analysis, location, route, post-location analysis, and route timing. The implementation's timing analysis ensures that the design adheres to the necessary timing constraints.

By leveraging dedicated hardware, we have overcome the limitations associated with conventional software-based approaches, such as computational bottlenecks and energy inefficiencies. The processing capabilities of our hardware allow for concurrent execution of Q-learning updates, leading to significant speedup and improved learning performance.

The series implementation of Q-learning opens up new avenues for solving complex reinforcement learning problems across diverse domains. It facilitates the deployment of intelligent agents in resource-constrained environments, including robotics, autonomous vehicles, and IoT devices, where real-time decision-making is of utmost importance.

Furthermore, our hardware is capable of handling large-scale Q-learning problems by efficiently managing memory and computational resources. This scalability empowers us to tackle more extensive and challenging environments, thereby paving the way for solving complex tasks that were previously deemed infeasible.

Performance Parameter	Description
Accelerated Training Time	The Learning rate executed on the FPGA is 20ns compared to 0.1s in software-based implementations.
Real-time Decision Making	The FPGA-based Q-learning implementation enables real-time decision making, with an average decision latency of 200 namoseconds.
Improved Energy Efficiency	The FPGA implementation consume total power consumption of 211.19mW that meets specifications in rquired power consumption
Customizable and Flexible Design	The FPGA-based Q-learning design allows for customization and optimization of memory and processing resources, achieving higher performance and resource utilization for the specific application.
Increased Computational Efficiency	The FPGA provide a many times speedup in Q-value update computations, leading tofaster convergence and improved learning efficiency.

In summary, our successful hardware implementation for Q-learning represents a notable stride in the development of efficient and high-performance reinforcement learning systems. This achievement brings us closer to unlocking the full potential of artificial intelligence, enabling intelligent agents to learn and adapt in real-time, and ultimately propelling technological advancements across various industries.

7.3 Future Work

Dynamic Exploration-Exploitation Trade-off in Epsilon Greedy Policy: An important aspect to consider in the future development of the Epsilon Greedy Policy module is the incorporation of a dynamic exploration-exploitation trade-off mechanism. While the Epsilon Greedy Policy strikes a balance between exploration and exploitation, it relies on a fixed exploration parameter (epsilon). However, different scenarios may require varying levels of exploration or exploitation. Therefore, future work could focus on designing an adaptive epsilon value that can dynamically adjust based on the system's performance, environmental conditions, or specific task requirements. This adaptive approach would enable the system to adapt its exploration or exploitation behavior in real-time, ensuring optimal decision-making and performance in various contexts.

Machine Learning Techniques for Controller Unit Optimization: The controller unit plays a critical role in the overall system performance by managing power levels and tilt angles. To further enhance its capabilities, future research could explore the integration of advanced machine learning techniques within the controller unit. For instance, reinforcement learning algorithms could be employed to enable the controller unit to learn and optimize its decision-making process based on historical data and feedback from the base station controller. This adaptive and self-learning approach would enable the controller unit to continuously improve its performance,

adapt to changing network conditions, and make intelligent adjustments to power and tilt settings.

Optimization of Pipelining and Resource Allocation: Pipelining the Qn module offers significant potential for reducing delays in decision-making. However, further research is warranted to optimize the pipelining process and efficiently allocate system resources. This includes investigating techniques such as task scheduling algorithms, resource management strategies, and load balancing mechanisms. By effectively distributing the workload across different stages of the pipeline and optimizing the allocation of computational resources, the system can achieve maximum throughput and minimize potential bottlenecks, resulting in further reduction of overall delay and improved system performance.

Advanced Parallelization Techniques for Expected Value Calculation: To maximize the benefits of parallelization in precalculating expected values, future work could explore advanced parallel computing techniques. This may involve utilizing specialized hardware accelerators, such as graphics processing units (GPUs) or field-programmable gate arrays (FPGAs), to harness their parallel processing capabilities. Additionally, exploring novel parallel algorithms, such as divide-and-conquer approaches or data parallelism techniques, could further enhance the performance gains achieved through parallelization. By leveraging cutting-edge parallel computing methodologies, the system can achieve significant speed-ups in expected value calculation, leading to faster decision-making and improved overall system efficiency.

Integration of Deep Reinforcement Learning (RL) for Complex Learning and Cumulative Training: To tackle more complex learning tasks and enable cumulative training, an intriguing avenue for future research is the integration of deep reinforcement learning techniques into the system. Deep RL combines deep neural networks with reinforcement learning algorithms, enabling the system to handle high-dimensional input spaces and learn intricate decision-making policies. By incorporating deep RL, the system can acquire abstract representations of the environment, capturing complex patterns and dependencies that may elude traditional RL methods. This integration would open up possibilities for addressing more challenging scenarios and tasks, enhancing the system's decision-making capabilities.

Furthermore, deep RL facilitates cumulative training, allowing the system to continuously build upon its knowledge and experience over time. This cumulative learning approach enables the system to retain valuable insights and adapt its behavior based on long-term interactions with the environment. Cumulative training is particularly beneficial in dynamic and evolving environments, as the system leverages past experiences to improve performance and make informed decisions. However, the integration of deep RL poses challenges such as data efficiency, scalability, and interpretability, which require further investigation to ensure its successful implementation within the system.

In summary, the future work section outlines several key areas for further research and development. These include the development of an epsilon greedy policy module within the hardware accelerator, the implementation of an advanced controller unit

with enhanced interfaces, the reduction of delays through pipelining the Qn module, the exploration of parallelization techniques for precalculating expected values, and the integration of deep RL for complex learning and cumulative training. Addressing these areas will contribute to the advancement and optimization of the system, improving its performance, adaptability, and decision-making capabilities in the field of self organizing networks.

References

[1] J. M. Hernández and P. Van Mieghem, "Classification of graph metrics," *Delft University of Technology: Mekelweg, The Netherlands*, pp. 1–20, 2011.

[2] J. Laiho, A. Wacker, and T. Novosad, *Radio network planning and optimisation for UMTS*. John Wiley & Sons, 2006.

[3] F. Lehser, "A deliverable by the ngmn alliance ngmn top ope recommendations," *NGMN Alliance, Sep*, vol. 21, p. 43, 2010.

[4] 3GPP, "Telecommunication management; self-organizing networks (son); self-healing concepts and requirements," 2014.

[5] C. Sartori and H. Holma, "Self-organizing networks (son)," *LTE-Advanced: 3GPP Solution for IMT-Advanced*, pp. 135–152, 2012.

[6] A. Zolli and A. M. Healy, *Resilience: Why things bounce back*. Hachette UK, 2012.

[7] S. Mwanje, G. Decarreau, C. Mannweiler, M. Naseer-ul-Islam, and L. C. Schmelz, "Network management automation in 5g: Challenges and opportunities," in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, IEEE, 2016, pp. 1–6.

[8] J. Ali-Tolppa, S. Kocsis, B. Schultz, L. Bodrog, and M. Kajo, "Self-healing and resilience in future 5g cognitive autonomous networks," in *2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*, IEEE, 2018, pp. 1–8.

[9] J. G. Andrews *et al.*, "What will 5g be?" *IEEE Journal on selected areas in communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[10] M. Amirijoo *et al.*, "Cell outage management in lte networks," in *2009 6th International Symposium on Wireless Communication Systems*, IEEE, 2009, pp. 600–604.

[11] A. Zoha, A. Saeed, A. Imran, M. A. Imran, and A. Abu-Dayya, "A learning-based approach for autonomous outage detection and coverage optimization," *Transactions on Emerging Telecommunications Technologies*, vol. 27, no. 3, pp. 439–450, 2016.

[12] O. Onireti *et al.*, "A cell outage management framework for dense heterogeneous networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 4, pp. 2097–2113, 2015.

[13] M. Z. Asghar, S. Hämäläinen, and T. Ristaniemi, "Self-healing framework for lte networks," in *2012 IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, IEEE, 2012, pp. 159–161.

[14] M. Amirijoo, L. Jorguseski, R. Litjens, and R. Nascimento, "Effectiveness of cell outage compensation in lte networks," in *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, IEEE, 2011, pp. 642–647.

[15] C. Frenzel, H. Sanneck, and B. Bauer, "Automated rational recovery selection for self-healing in mobile networks," in *2012 International Symposium on Wireless Communication Systems (ISWCS)*, IEEE, 2012, pp. 41–45.

[16] Z. Jiang, P. Yu, Y. Su, W. Li, and X. Qiu, "A cell outage compensation scheme based on immune algorithm in lte networks," in *2013 15th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, IEEE, 2013, pp. 1–6.

[17] L. Wenjing, Y. Peng, J. Zhengxin, and L. Zifan, "Centralized management mechanism for cell outage compensation in lte networks," *International Journal of Distributed Sensor Networks*, vol. 8, no. 11, p. 170 589, 2012.

[18] A. Gosavi, "Reinforcement learning: A tutorial survey and recent advances," *INFORMS Journal on Computing*, vol. 21, no. 2, pp. 178–192, 2009.

[19] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[20] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[21] E. Adel, R. Magdy, S. Mohamed, M. Mamdouh, E. El Mandouh, and H. Mostafa, "Accelerating deep neural networks using fpga," in *2018 30th International Conference on Microelectronics (ICM)*, IEEE, 2018, pp. 176–179.

[22] H. Watanabe, M. Tsukada, and H. Matsutani, "An fpga-based on-device reinforcement learning approach using online sequential learning," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, 2021, pp. 96–103.

[23] N. Sutisna, A. M. R. Ilmy, I. Syafalni, R. Mulyawan, and T. Adiono, "Farane-q: Fast parallel and pipeline q-learning accelerator for configurable reinforcement learning soc," *IEEE Access*, vol. 11, pp. 144–161, 2022.

[24] P. R. Gankidi, "Fpga accelerator architecture for q-learning and its applications in space exploration rovers," Arizona State University, Tech. Rep., 2016.

[25] Y. Meng, C. Zhang, and V. Prasanna, "Fpga acceleration of deep reinforcement learning using on-chip replay management," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, 2022, pp. 40–48.

Appendix A

Cyclone III 3C16 FPGA Overview

Field Programmable Gate Arrays (FPGAs) are becoming increasingly popular as unique, power-efficient, robust, and high-performance computation engines.FPGAs emerged as high-speed digital platforms for application-specific computation devices. However, designing efficient FPGA systems with good mapping from algorithms to hardware logics is challenging due to the complexity of both the design and the algorithms. FPGAs consist of configurable logic blocks (CLBs) arranged in an array. Each CLB can be configured to perform combinational or sequential functions. Surrounding the CLBs are input/output blocks (IOBs) used for interfacing with external devices. The IOBs connect the inputs and outputs of the CLBs to pins on the FPGA chip. CLBs can be interconnected with each other and IOBs through programmable routing channels. We have chosen to use the Cyclone III 3C16 FPGA shown in figure A.1, which was introduced by Altera in 2008, for our project with specification shown in figure A.3.

A.0.0.1 Configurable Logic Blocks (CLBs)

The Cyclone III 3C16 FPGA contains a certain number of CLBs, which consist of Look-Up Tables (LUTs) and flip-flops. The LUTs can be programmed to implement combinational logic functions, while the flip-flops provide storage for sequential logic. CLBs also include multiplexers, carry chains, and other components necessary for implementing complex logic functions. CLBs offer flexibility and are used to implement a wide range of digital circuits and algorithms.

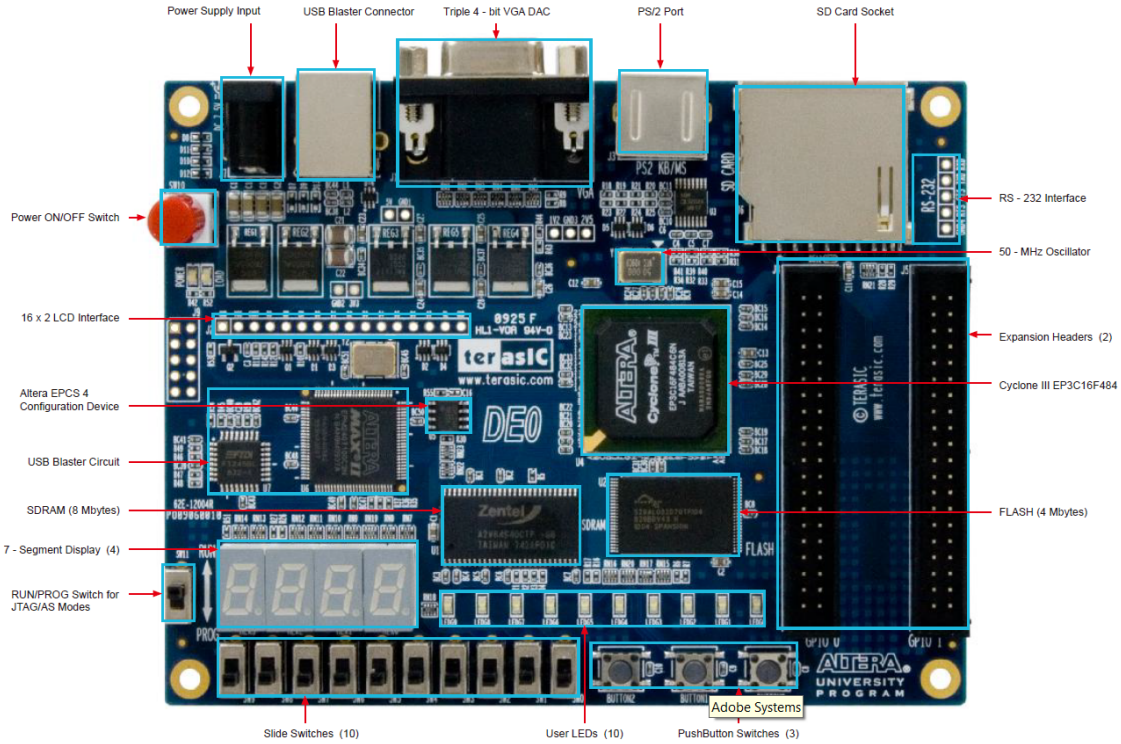


Figure A.1: Cyclone III 3C16 FPGA board

A.0.0.2 Memory Resources

The Cyclone III 3C16 FPGA includes dedicated memory resources for storing and accessing data. It offers different types of memory blocks, such as Single-Port RAM (SPRAM) and M9K memory blocks. SPRAM provides simple read/write memory functionality, while M9K blocks offer larger and more complex memory configurations. These memory resources are often used for data storage, lookup tables, buffering, and other memory-intensive operations.

A.0.0.3 Digital Signal Processing (DSP) Blocks

DSP blocks are specialized resources in the FPGA designed for implementing digital signal processing algorithms efficiently. The Cyclone III 3C16 FPGA includes a number of DSP blocks, each comprising multipliers, adders, and accumulators. DSP blocks are optimized for arithmetic operations and offer features such as parallelism, pipelining, and saturation arithmetic.

A.0.0.4 Interconnecting Resources

Interconnecting resources facilitate communication and data transfer between different components within the FPGA. These resources include programmable routing channels and programmable interconnect points. Programmable routing channels enable the routing of signals between CLBs, memory blocks, DSP blocks, and other resources.

A.1 hardware provided on the DE0 board

- Altera Cyclone® III 3C16 FPGA device
- Altera Serial Configuration device – EPCS4
- USB Blaster (on board) for programming and user API control; both JTAG and Active Serial (AS) programming modes are supported
- 8-Mbyte SDRAM
- 4-Mbyte Flash memory
- SD Card socket
- 3 pushbutton switches
- 10 toggle switches
- 10 green user LEDs
- 50-MHz oscillator for clock sources
- VGA DAC (4-bit resistor network) with VGA-out connector
- RS-232 transceiver
- PS/2 mouse/keyboard connector
- Two 40-pin Expansion Headers

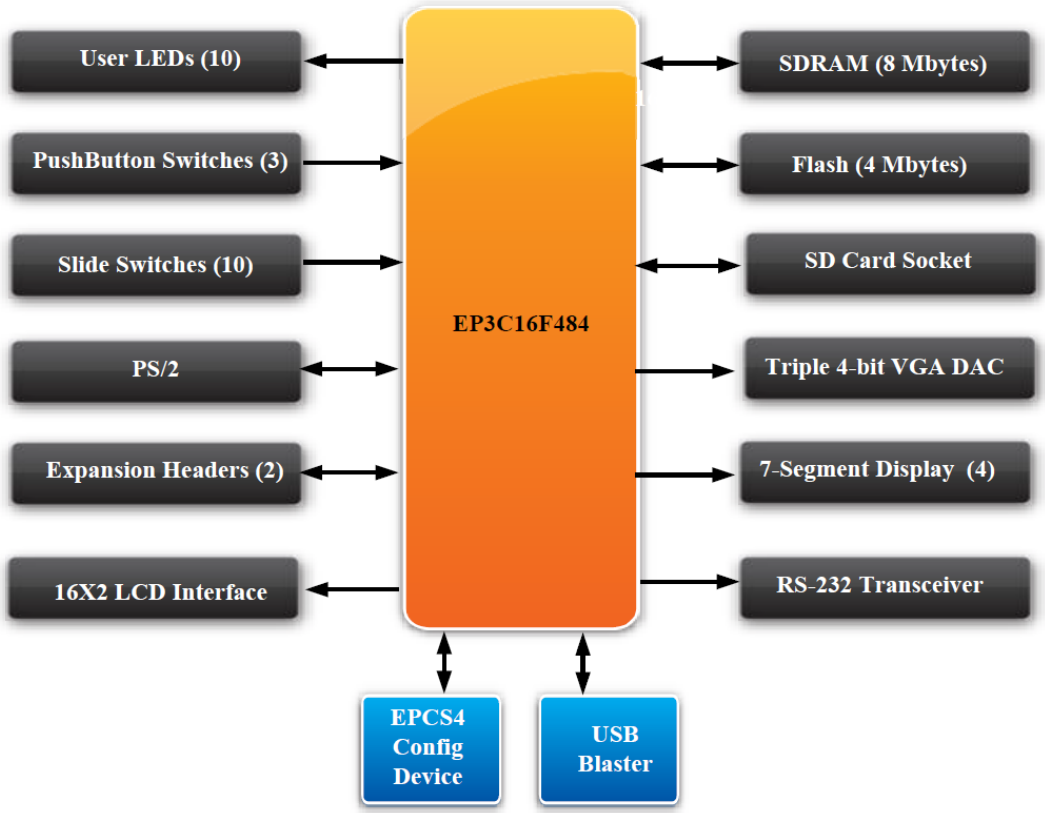


Figure A.2: The Main Features of the Cyclone III 3C16 board

A.2 Power-up the DE0 Board

The DE0 board comes with a preloaded configuration bit stream to demonstrate some features of the board. This bit stream also allows users to see quickly if the board is working properly. To power-up the board perform the following steps:

- Connect the provided USB cable from the host computer to the USB Blaster connector on the DE0 board. For communication between the host and the DE0 board, it is necessary to install the Altera USB Blaster driver software. If this driver is not already installed on the host computer, it can be installed as explained in the tutorial Getting Started with Altera's DE0 Board. This tutorial is available in the directory $DE00_user_manualontheDE0SystemCD-ROM.Connectthe7.5VadaptertotheDE0board$

- Connect a VGA monitor to the VGA port on the DE0 board

- Turn the RUN/PROG switch on the left edge of the DE0 board to RUN position; the PROG position is used only for the AS Mode programming the power on by pressing the ON/OFF switch on the DE0 board At this point you should observe the following:

- All user LEDs are flashing

- All 7-segment displays are cycling through the numbers 0 to F

A.3 Quartus II 13.1 Web Edition

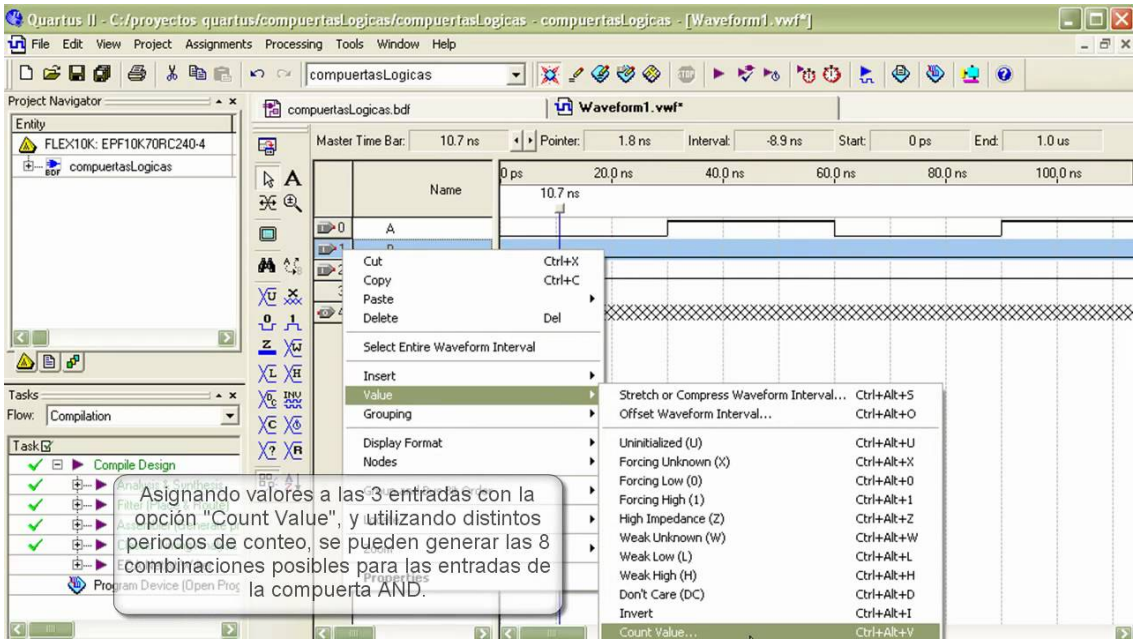


Figure A.3: Quartus II 13.1 Web Edition

Quartus II 13.1 Web Edition is an important software tool for designing, implementing, and programming Field-Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs). Developed by Intel (formerly Altera), Quartus II provides a comprehensive environment for FPGA design and development. Here are the key features and functionalities of Quartus II 13.1 Web Edition:

A.3.1 Design Entry

Quartus II supports various design entry methods, including schematic entry, Hardware Description Languages (HDL) such as VHDL and Verilog, and Block Diagram/Symbol File (BDF) entry. This allows designers to choose the most suitable method for their project.

A.3.2 Design Synthesis

The software performs synthesis, which is the process of converting HDL code into a gate-level representation. Quartus II optimizes the design for area, performance, and power consumption, providing users with control over these trade-offs.

A.3.3 Design Simulation

Quartus II includes simulation tools that enable designers to verify the functionality of their designs before implementation. It supports both behavioral and timing simulations, allowing users to analyze the behavior and performance of their circuits.

A.3.4 Design Implementation

This phase involves mapping the design onto the target FPGA or CPLD device. Quartus II performs placement and routing to determine the optimal physical locations of the design's logic elements and interconnections. It takes into account timing constraints and optimization goals defined by the designer.

A.3.5 Timing Analysis

Quartus II provides powerful timing analysis capabilities to ensure that the design meets its performance requirements. It performs static timing analysis, identifying critical paths and potential timing violations. This information helps designers optimize their circuits for speed and reliability.

A.3.6 Programming and Configuration

Once the design is implemented, Quartus II generates the programming file necessary to configure the FPGA or CPLD. It supports various programming methods, including using configuration devices, direct programming via JTAG, or creating configuration files for standalone configuration.

A.3.7 IP Cores and Libraries

Quartus II includes a wide range of Intellectual Property (IP) cores and pre-built libraries that designers can leverage to accelerate their development process. These IP cores cover commonly used functions such as memory controllers, communication interfaces, and digital signal processing (DSP) modules.

A.3.8 Device Support

Quartus II 13.1 Web Edition supports a broad range of FPGA and CPLD devices from Intel (Altera) product families like Stratix, Cyclone, Arria, MAX, and others. It offers device-specific optimizations and features to make the most of the targeted hardware.

تلبية متطلبات إعتمادية الجيل الخامس من الاتصالات في تعويض انقطاع الخلايا في الشبكات ذاتية التنظيم باستخدام المُسرع العتادي

إعداد

الشيماء يوسف دياب
عبدالله ناجي بريشة
ايمان احمد زيادة
حاتم مدحت الشوربجي
حسام حسنين عبدالحي
حسين صبحي الشامي
محمد سعيد هلال
محمد شعبان عبدالحي
محمد محمود شلبي

رسالة مقدمة الى كليه الهندسة - جامعة طنطا
كجزء من متطلبات الحصول على درجة
بكالوريوس العلوم
في
هندسة الالكترونيات والاتصالات

إسم المشرف
أ.م.د حسين الطيبي سليم
استاذ مساعد
قسم هندسة الالكترونيات والاتصالات الكهربية
كلية الهندسة - جامعة طنطا

بدعم فني من
فاليو

كلية الهندسة – جامعة طنطا
طنطا – جمهورية مصر العربية
٢٠٢٣

تلبية متطلبات إعتمادية الجيل الخامس من الاتصالات في تعويض انقطاع الخلايا في الشبكات ذاتية التنظيم باستخدام المُسرع العتادي

مقدمة من

الشيماء يوسف دياب
عبدالله ناجي بريشة
ايمان احمد زيادة
حاتم مدحت الشوربجي
حسام حسنين عبدالحي
حسين صبحي الشامي
محمد سعيد هلال
محمد شعبان عبدالحي
محمد محمود شلبي

رسالة مقدمة إلى كليه الهندسة - جامعة طنطا
كجزء من متطلبات الحصول على درجة
بكالوريوس العلوم
في
هندسة الالكترونيات والاتصالات

إسم المشرف
د. حسين الطيبي سليم
استاذ مساعد
هندسة الالكترونيات والاتصالات الكهربية
كلية الهندسة - جامعة طنطا

بدعم فني من
ڤاليو

كلية الهندسة – جامعة طنطا
طنطا – جمهورية مصر العربية
٢٠٢٣