FPGA IMPLEMENTATION OF QUANTUM KEY DISTRIBUTION BASED COMMUNICATION

by

Yiğit Bilgin

Submitted to Graduate School of Natural and Applied Sciences

in Partial Fulfillment of the Requirements

for the Degree of Master of Science in

Computer Engineering

Yeditepe University

2023

FPGA IMPLEMENTATION OF QUANTUM KEY DISTRIBUTION BASED COMMUNICATION

APPROVED BY:

Prof. Dr. Sezer Gören Uğurdağ .....
(Thesis Supervisor)
(Yeditepe University)

Prof. Dr. H. Fatih Uğurdağ
(Thesis Co-Supervisor)
(Özyeğin University)

Assist. Prof. Dr. Onur Demir .....
(Yeditepe University)

Assist. Prof. Kadir Durak .....
(Özyeğin University)

DATE OF APPROVAL: ..... / ..... / 20.....

I hereby declare that this thesis is my own work and that all information in this thesis has been obtained and presented in accordance with academic rules and ethical conduct. I have fully cited and referenced all material and results as required by these rules and conduct, and this thesis study does not contain any plagiarism. If any material used in the thesis requires copyright, the necessary permissions have been obtained. No material from this thesis has been used for the award of another degree.

I accept all kinds of legal liability that may arise in case contrary to these situations.

Name, Last Name Yiğit Bilgin

Signature .....

ACKNOWLEDGEMENTS

It is with immense gratitude that I acknowledge the support and help of Professor Sezer Gören Uğurdağ and Professor H. Fatih Uğurdağ. Pursuing my thesis under their supervision has been an experience which broadens the mind and presents an unlimited source of learning.

Finally, I would like to thank my family and friends for their endless love and support, which makes everything more beautiful.

The work presented here is supported under TÜBİTAK ARDEB-1003 subproject numbered 118E993. Prof. Sezer Gören is the principle investigator. Asst. Prof. Kadir Durak is the principle investigator of the main TÜBİTAK ARDEB-1003 project (numbered 118E991).

ABSTRACT

FPGA IMPLEMENTATION OF QUANTUM KEY DISTRIBUTION BASED COMMUNICATION

Secure communication between devices becomes harder to achieve with classical methods due to increase in computational speed of computers. Goal of Quantum Key Distribution is sent entangled photon pairs with entangled photon source (EPS) to two or more terminals and create a secure key to use in classical public channels between terminals. EPS sends five million entangled photon pairs to each terminal to create a single key. For this complex and time critical processing operations, a central process unit (CPU) is not efficient enough. So every single operation that needs to be done in a pipeline architecture design with field programmable gate arrays (FPGAs) to obtain this inherently secure key in a reasonable time frame for use in public channels. Hardware implementation of quantum key distribution has multiple sub problems. Some of these problems are clock synchronization between terminals, calculation of time shift between terminal with entangled photon pairs, error correction in each terminal with received entangled photon pairs and generating a secure key with received entangled photon pairs from EPS. This thesis covers explanation of a quantum key distribution systems, an architectural design to create a quantum key distribution system with FPGAs, and a methodology for absolute verification of the final implemented system with FPGAs.

ÖZET

KUANTUM ANAHTAR DAĞITICILI İLETİŞİMİN FPGA'LE GERÇEKLENMESİ

Her geçen gün klasik methodlarla cihazlar arasında güvenli iletişim kurmak, bilgisayarların hesaplama hızındaki artıştan dolayı zorlaşmaktadır. Kuantum Anahtar Dağıtıcısı'nın amacı dolanık foton kaynağı (EPS) ile yaratılan dolanık foton çiftlerinin iki veya daha fazla terminale gönderilmesiyle terminaller arasında klasik iletişim kanallarında kullanılması için güvenli bir anahtar oluşturulmasıdır. EPS tek bir anahtar oluşturmak için her terminale beş milyon dolanık foton çifti göndermektedir. Bu kompleks ve zaman açısından kritik işlem için bir merkezi işlem birimi (CPU) yeterince verimli değildir. Özü itibariyle güvenli olan bu anahtarın klasik iletişim kanallarında kullanılması için makul bir zaman çerçevesinde elde edilebilmesi gerekmektedir. Bu sebebten yapılması gereken her işlemin alan programlanabilir kapı dizileriyle (FPGA'lerle) bir veri hattı mimarisi tasarımında yapılması gerekmektedir. Kuantum anahtar dağıtımının FPGA'lerle gerçeklenmesi içerisinde birçok alt problem barındırmaktadır. Bu problemler terminaller arası saat senkronizasyonu, terminaller arasındaki zaman kaymasının hesaplanması, foton çiftleriyle güvenli bir anahtar üretilmesi ve oluşturulmuş güvenli anahtar üzerinde hata düzeltmelerinin yapılmasıdır. Bu tez kuantum anahtar dağıtım sistemlerini, FPGA'lerle bir kuantum anahtar sistemi oluşturmak için bir mimari tasarımı, ve FPGA'lerle oluşturulmuş son sistemin tam anlamıyla doğrulanması için bir metodoloji sunmaktadır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv
ABSTRACT v
ÖZET vi
TABLE OF CONTENTS vii
LIST OF FIGURES ix
LIST OF TABLES xi
LIST OF SYMBOLS/ABBREVIATIONS xii
1. INTRODUCTION 1
1.1. IMPORTANCE OF SECURITY 2
1.1.1. History of Cryptography 3
1.2. QUANTUM CRYPTOGRAPHY 4
1.2.1. BBM92 Protocol 5
1.2.2. E91 Protocol 7
1.3. IMPLEMENTATION WITH FPGA 9
1.3.1. PYNQ 10
1.4. OBJECTIVE OF THE HARDWARE IMPLEMENTATION OF QKD 10
2. RELATED RESEARCH 12
3. ARCHITECTURE 13
3.1. IDEAL QKD ARCHITECTURE 13
3.2. FINAL QKD ARCHITECTURE 15
3.3. INNER MODULES OF QKD ARCHITECTURE 16
3.3.1. QCE Module 17
3.3.1.1. Communication Module 17
3.3.1.2. Quantum Calculation Module 19
4. METHODOLOGY. 21
4.1. TIME-STAMP SEQUENCE GENERATOR (TSSGEN) 22
4.2. PYTHON REFERENCE MODEL (PYREF) 25
4.2.1. Verification of Python reference model 28
4.3. TESTING PYREF MODEL 29
4.3.1. Testing PyRef on a single computer 29

4.3.2. Testing PyRef on FPGAs 30
4.3.3. Testing PyRef on FPGAs with Overlays 31
4.4. RTL IMPLEMENTATION OF QKD SYSTEM 31
5. RESULTS 33
5.1. RESULTS OF PYREF ON A SINGLE COMPUTER 34
5.2. RESULTS OF PYREF ON FPGAS 35
5.3. RESULTS OF PYREF ON FPGAS WITH OVERLAYS 37
5.4. RESULTS OF RTL IMPLEMENTATION OF QKD SYSTEM 39
6. CONCLUSIONS 44
REFERENCES 45

LIST OF FIGURES

Figure 1.1. QKD-based communication 1

Figure 1.2. QKD-based communication system for BBM92 protocol 6

Figure 1.3. QKD-based communication system for E91 protocol 8

Figure 3.1. Ideal QKD architecture 14

Figure 3.2. Final QKD architecture 15

Figure 3.3. Top-level of FPGA-1 16

Figure 3.4. Inner design of communication module of FPGA-1 18

Figure 3.5. Inner design of quantum calculation module of FPGA-1 19

Figure 4.1. Inner design of PyRef 26

Figure 4.2. Number of detected photons within time frames 26

Figure 4.3. Verification model of PyRef 28

Figure 4.4. Testing PyRef on FPGAs 30

Figure 4.5. Inner design of a FPGA with Overlays 31

Figure 4.6. Block design of RTL implementation of QKD system 32

Figure 5.1. Accessed PYNQ environment in a browser 36

Figure 5.2. Setup created with two FPGAs, a computer, and a network switch 36

Figure 5.3. A part from simulation of RTL designs 38

Figure 5.4. Comparison between key generated PyRef and RTL designs 39

Figure 5.5. Setup with two ATLYS boards, a computer connected to a network switch .. 42

Figure 5.6. Result for verification of received data 42

Figure 5.7. Extracted secure key from 500 entangled photon information..... 43

LIST OF TABLES

Table 4.1. A part from TSS1 generated by TSSgen 25

Table 5.1. Results of expected key difference rate between terminals and found key difference between terminals for 10 thousand entangled photon pairs 34

Table 5.2. Performance results of PyRef on PS side of the FPGA..... 37

Table 5.3. Test results for FPGA directly connected to a computer with an Ethernet cable40

Table 5.4. Test results for FPGA and computer connected to a network switch with Ethernet cables 41

Table 5.5. Test results for RTL implementation of QKD system..... 43

LIST OF SYMBOLS/ABBREVIATIONS

FPGA	Field programmable gate array
QKD	Quantum key distribution
EPS	Entangled photon source
TSU	Time-stamp unit
BBM92	Bennett, Brassard and Mermin in 1992 protocol
E91	Artur Ekert’s protocol
CHSH	Clauser-Horne-Shimony-Holt
ASIC	Application specific integrated circuit
VHDL	Very High Description Language
RTL	Register-transfer level
CPU	Central process unit
BB84	Bennett, Brassard in 1984 protocol
TSS	Time-stamp sequence
TSSgen	Time-stamp sequence generator
PyRef	Python reference model of design
RTL	Register-transfer level
PS	Processing system
PL	Programmable logic
UART	Universal asynchronous receiver-transmitter
UDP	User Datagram Protocol
TCP\IP	Transmission Control Protocol and Internet Protocol

1. INTRODUCTION

The classical methods of achieving a secure communication between devices intensely rely on the computation power and complexity of the mathematical problem but QKD is inherently secure due to the underlying fundamentals of quantum mechanic [1]. A QKD-based communication consists of an EPS and two or more terminals. EPS generates entangled photon pairs, and sends one of the generated entangled photon to each terminal through an optical channel. Each terminal has an optical telescope locked to the EPS to receive their entangled photons. After terminals receive their entangled photons, limited information about the entangled photons is shared between terminals through a public channel. Due to the entangled photon behaves the same while not observed by an eavesdropper, this limited information is enough to generate a secure key to use in a public channel between terminals. A representation of QKD-based communication can be seen in Figure 1.1.

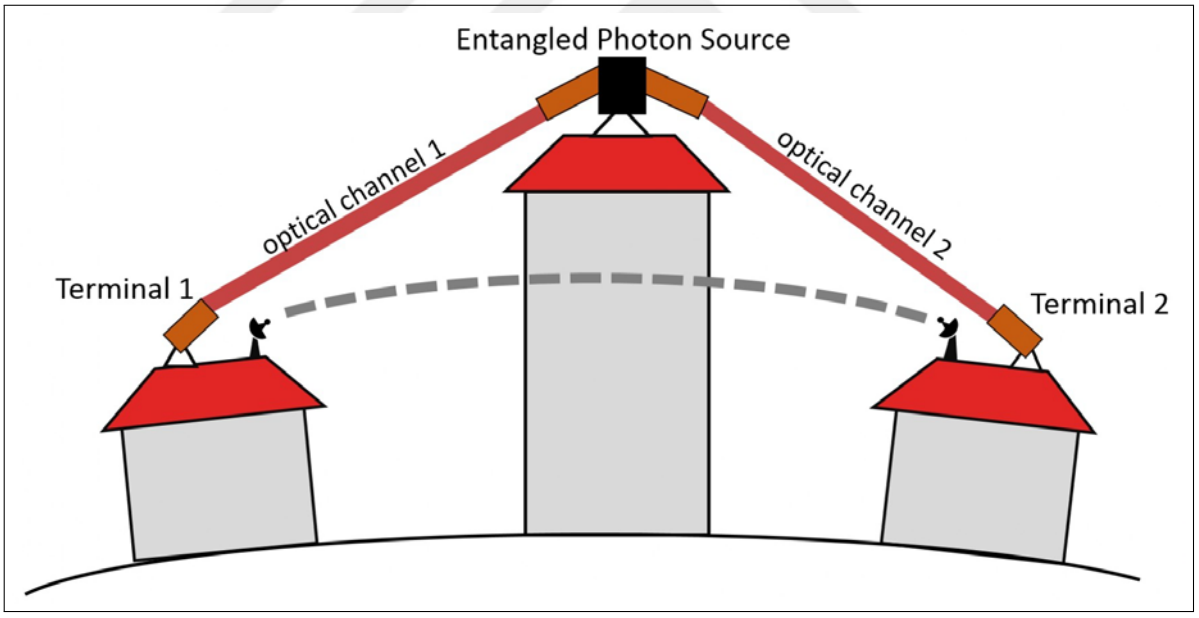

Figure 1.1. QKD-based communication

Before diving into how quantum cryptography works, there is a much more important question that needs to be answered. This question is, why quantum cryptography is needed. To better understand why quantum cryptography is needed, the importance of security has to be known. In the Section 1.1, the importance of security and history of cryptography, which created to achieve more secure communication are explained to better comprehend the need for quantum

cryptography in the near future.

After the importance of security becomes clear, it is time to move on to the underlying fundamentals of quantum cryptography. In Section 1.2, the idea and physics behind QKD-based communication and two potential QKD-based quantum cryptography protocols to use in this project are explained for a better understanding of QKD-based communication.

Later on, to achieve QKD-based communication in an optimal way, FPGA implementation of the system is a necessity due to time critical operations. With FPGAs, it is possible to create solutions which directly meet the needs of any system. In Section 1.3, ideas behind designing a system with FPGA and the usage of PYNQ to accelerate the process of this design are further explained.

Lastly, in Section 1.4, the advantages and disadvantages of chosen two QKD-based quantum cryptography protocols are compared and the more suitable protocol for this project is chosen. After the protocol to use in the project set, the final outcomes expected to achieve with FPGA implementation of QKD-based communication system are explained in detail.

1.1. IMPORTANCE OF SECURITY

Throughout history, all living creatures have chosen their habitats in geographies that they think are safe for them. While some established their habitats in the sea, others have preferred the land. The main common aspects of these preferences are to provide a safe continuity for shelter, nutrition, reproduction and to make new generations exist in similar environments. This need for safety has been studied throughout the years by humans. In Maslow's hierarchy of needs, human needs divided into a five-tier model and needs lower down in the hierarchy must be satisfied before individuals can attend to needs higher up [2]. From the bottom of the hierarchy upwards, the needs are: physiological, safety, love and belonging, esteem and self-actualization [2]. After the physiological needs of a human are met, without safety, a person can not meet higher needs. One of the major parts of safety is secure communication.

The need for secure communication has been one of the most important parts of safety. Since the Romans, the need for security does not only depend on a safe shelter, but also depends on a way to communicate safely. Security of communication is crucial for security of private

information, because this private information can reveal a party's weaknesses or plans. To provide secure communication between parties, people have developed cryptography methods. Briefly, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages [3]. In Section 1.1.1, these cryptography methods used by humans throughout history are explicated.

1.1.1. History of Cryptography

The art and science of creating no readable data or cipher so that only the intended person is only-able to read the data is called Cryptography [4]. The earliest forms of cryptography were found in the cradle of civilization, which comes as no surprise, including the regions currently encompassed by Egypt, Greece, and Rome [5].

A great example of the early cryptography method is Caesar cipher, which gets its name from Roman Emperor Gaius Julius Caesar. Caesar cipher is one of the most known encryption and decryption methods. Caesar cipher is a type of substitution type cipher. In this kind of cipher, each letter in the plain-text is replaced by a letter some fixed number of positions down the alphabet [6]. Caesar cipher is one of the simplest type of substitution method. In this letters of alphabets are replaced by letters three places further down the alphabet. But in general, this shift may be of any places [7]. For an example, the message "RETURN TO ROME" is encrypted as "UHWXUA WR URPH" with usage of the Caesar cipher. So any attacker is not able to read the message, if he intercepts the message [8]. This might have been a viable way to encrypt data before the rise of computer technology. But with increased computational speed, this encrypted message can easily be deciphered, even with brute force attacks.

With the advancement of computer technologies, the need for new cryptography methods has increased, and cryptography methods used in modern devices have also improved significantly over the last half century. Unlike Caesar cipher, modern cryptography methods use complex mathematical problems. There are many different methods of cryptography used for different applications. It is possible to divide cryptography used in modern systems into symmetric and asymmetric encryption.

A symmetric encryption considers two parties who share a key and will use this key to imbue communicated data with various security attributes [3]. A block of text and a specific key will always result in the same encrypted text when using symmetric encryption. The original text will always be produced by using the same key on that block of encrypted text. Some examples of symmetric encryption algorithms are Advanced Encryption Standard (AES), Data Encryption Standard (DES), Rivest Cipher 4 (RC4). An asymmetric encryption is just like a symmetric encryption except for an asymmetry in the key structure. The public key used to encrypt is different from the secret key used to decrypt [3]. Furthermore, the public key is public, known to the sender and also to the adversary [3]. So while only a receiver in possession of the secret key can decrypt, anyone in possession of the corresponding public key can encrypt data to send to this one receiver [3]. Some examples of asymmetric encryption algorithms are Rivest Shamir Adleman (RSA), Elliptical Curve Cryptography (ECC). The main difference between symmetric and asymmetric encryption is the priority of speed or safety. While asymmetric encryption prioritizes safety over speed, symmetric encryption prioritizes speed over safety.

1.2. QUANTUM CRYPTOGRAPHY

Today, with the rise in technological advancements and increase in computational speed, it is getting harder to provide secure communication with classical methods. With quantum computing, the need for secure communication is increasing rapidly. An important point to understand is that the fragility of current classical cryptosystems is not only a potential threat for the present, but a more serious and realistic threat for the future [9]. For the need for secure communication, scientists search for new methods to create secure communication systems. Under these circumstances, to provide fast and reliable secure communication between parties, we will explore two QKD-based quantum cryptography protocols. These protocols are Bennett, Brassard and Mermin in 1992 protocol (BBM92) and Artur Ekert's protocol (E91). Detailed information about these protocols can be found in Section 1.2.1, and Section 1.2.2.

Both of the protocols need the same setup in Figure 1.1. There will be an EPS, and two terminals. Terminals need to be locked at EPS through an optical channel to receive entangled

photons. In both terminals, there needs to be a time-stamp unit (TSU) to time-tag detected photons, and a device for computational purposes. Both setups use beam splitters, polarizers, and polarizing beam splitters in the system. There need to be classical communication channel between terminals to share information to extract a secure key. With this information in mind, we can analyze the fundamentals of both protocols and compare these protocols in Section 1.4 to select the more suitable one for the project.

1.2.1. BBM92 Protocol

Each terminal uses six photon detectors for BBM92 protocol. Inner design of terminals needed to implement BBM92 protocol can be seen in Figure 1.2. After the EPS generates the entangled photons and sends them to both terminals via an optical channel, each photon reaching the terminals is measured using a randomly chosen basis. This randomness in basis is achieved with beam splitters (B1, B2, B3, B4 in Figure 1.2). The beam splitter transforms the polarization of the photon's optical path by using a set of polarizers (H1, H2, H3, H4 in Figure 1.2). After the transformation of photons, the polarization of the photons is measured by polarizing beam splitters (PB1, PB2, PB3, PB4, PB5, PB6 in Figure 1.2). At the end of each polarizing beam splitter, a photon detector can be found. When a photon detector detects a photon, it sends a signal to the TSU and the TSU at the terminal creates a time tag for the detection time of the detected photon. This process takes place on both terminals and the TSUs need to be roughly synchronized. After TSU generates a time-tag for a detected photon, it sends the information of time-tag of the photon and detector number to a device. Both devices in the terminals share the measurement basis of detected photons through a public channel with each other to find coinciding photons. After the coinciding photons are found, both terminals can extract a secure key.

Each coinciding photon has to have a constant time shift between them and this time shift can be found by taking cross-correlation of the pulse trains of the detected photons at each detector. The secure key can be generated by using coinciding photons on a selected reference basis (Detector 5, 6 for Terminal-1, Detector 5', 6' for Terminal-2). In a perfect scenario, the detected coinciding photons in both terminals have the same polarization, which also indicates that both terminals extract the same key. Whereas, imperfections in the real world

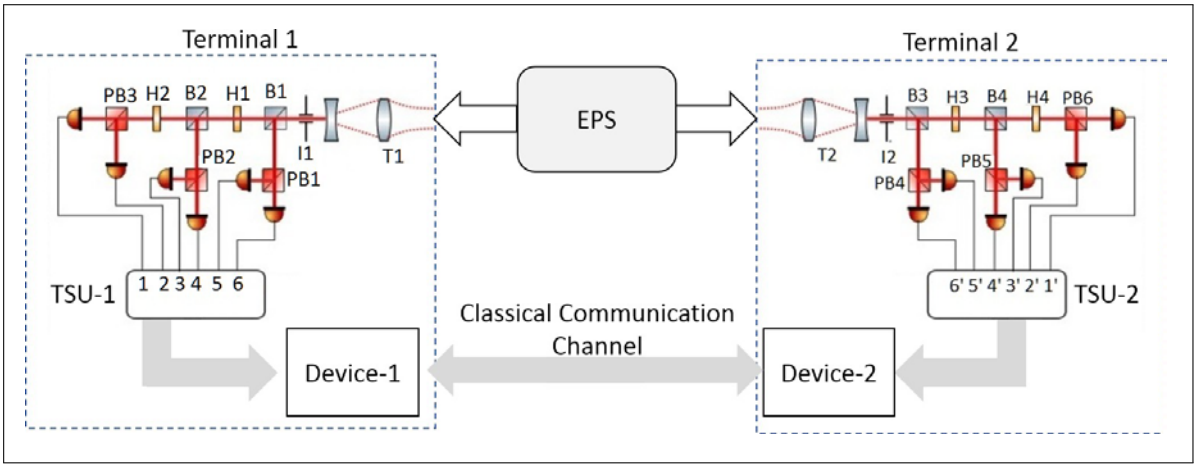

Figure 1.2. QKD-based communication system for BBM92 protocol

can cause errors in the final extracted key. Due to these imperfections, a key negotiation protocol needs to be implemented between terminals for error detection and correction.

Channel security is the most crucial priority of QKD-based communication. Before communication between terminals begins, checking whether or not an eavesdropper is listening is a must for QKD-based communication. To check for an eavesdropper, quantum based cryptography methods use the Bell's theorem, also known as Clauser-Horne-Shimony-Holt (CHSH) [10]. The CHSH value is commonly called S value, and S value is calculated using Equation (1.1).

$$S = E(a, b) - E(a, b') + E(a', b) + E(a', b') \tag{1.1}$$

Variables a and a' are detector basis for Terminal-1, and b and b' are detector basis for Terminal-2. Photons that are detected on a basis different from the chosen reference basis are used to check CHSH inequality violation with the below combinations.

- Terminal-1 Detector 1, 2 and Terminal-2 Detector 1', 2'
- Terminal-1 Detector 1, 2 and Terminal-2 Detector 3', 4'
- Terminal-1 Detector 3, 4 and Terminal-2 Detector 1', 2'
- Terminal-1 Detector 3, 4 and Terminal-2 Detector 3', 4'

Coincidences from each detector pair in both terminals are counted. The result of coincidences are categorized as ++, +−, −+ and −−. First + or − in a category represents the detector number of coinciding photons from Terminal-1 and second + or − represents the detector number of coinciding photons from Terminal-2. For each category, the number of coincidences found between photons are represented as N_{++}, N_{-+}, N_{+-}, and N_{--}. After all of the coincidences between detected photons are found, the value of E (a, b) is calculated with Equation (1.2).

$$E = \frac{(N_{++}) - (N_{+-}) - (N_{-+}) + (N_{--})}{(N_{++}) + (N_{+-}) + (N_{-+}) + (N_{--})} \tag{1.2}$$

After the calculation of each E value, the final result of S value can be obtained. The result of the S value points out whether or not an eavesdropper is listening to the optical channels. If the result of S value equals or greater than 2, there are no eavesdroppers between terminals and EPS. If S value equals or greater than 2, a secure key can be extracted with the chosen reference basis coinciding photons. After the key generation, extracted key from the chosen reference basis coinciding photons can be used on a public communication channel between terminals.

1.2.2. E91 Protocol

Each terminal uses four photon detectors for E91 protocol. Inner design of terminals needed to implement E91 protocol can be seen in Figure 1.3. After the EPS generates the entangled photons and sends them to both terminals via an optical channel, each photon reaching the terminals is measured using a randomly chosen basis. This randomness in basis is achieved with beam splitters (B1, B2 in Figure 1.3). The beam splitter transforms the polarization of the photon's optical path by using a set of polarizers (H1, H2 in Figure 1.3). After the transformation of photons, the polarization of the photons is measured by polarizing beam splitters (PB1, PB2, PB3, PB4 in Figure 1.3). At the end of each polarizing beam splitter, a photon detector can be found. When a photon detector detects a photon, it sends a signal to the TSU and the TSU at the terminal creates a time tag for the detection time of the detected photon. This process takes place on both terminals and the TSUs need to be synchronized.

After TSU generates a time-tag for a detected photon, it sends the information of time-tag of the photon and detector number to a device. Both devices in the terminals share the measurement basis of detected photons through a public channel with each other to find coinciding photons. After the coinciding photons are found, both terminals can extract a secure key.

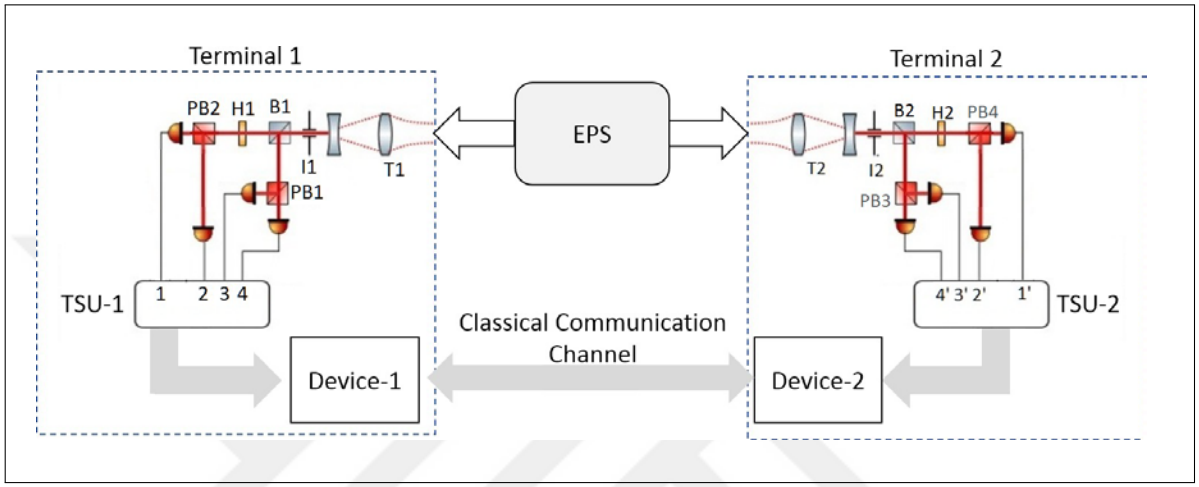

Figure 1.3. QKD-based communication system for E91 protocol

Each coinciding photon has to have a constant time shift between them and this time shift can be found by taking cross-correlation of the pulse trains of the detected photons at each detector. The two secure keys can be generated after coinciding photons are found. One with coinciding photons of Terminal-1 Detector 1, 2 and Terminal-2 Detector 1', 2' and the other with Terminal-1 Detector 3, 4 and Terminal-2 Detector 3', 4' at both terminals.

For choosing which key to use, Terminal-1 chooses one of the keys randomly and informs Terminal-2. A fixed threshold is chosen for the error rate of the key between terminals. This fixed error rate is for checking whether or not an eavesdropper listened to optical channels between EPS and terminals. If the difference between keys extracted from terminals before error correction is lower than this constant error rate, extracted keys can be used on a public communication channel.

1.3. IMPLEMENTATION WITH FPGA

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks connected via programmable interconnects [11]. Logic block of an FPGA can be configured in such a way that it can provide functionality as simple as that of transistor or as complex as that of a microprocessor [12]. FPGAs can be reprogrammed repeatedly to perform different operations, which gives FPGAs the flexibility of computer software.

FPGAs can be programmed using a language called hardware description language, which is used to express the structure and behavior of digital logic circuits. Very High Description Language (VHDL), and Verilog are the two most used hardware description languages in current day. VHDL is a strongly typed language with a non-C like syntax, and its highly verbose nature allows designers to describe designs with a comparatively lower ambiguity. Verilog is a more compact and less wordy language with a C-like syntax, making it easier to understand for designers with knowledge of C [13].

FPGAs have a remarkable role in embedded system development due to their capability to start system software development simultaneously with hardware, enable system performance simulations at a very early phase of the development, and allow various system partitioning trials and iterations before final freezing of the system architecture [14]. Due to high speed real time operation, and repeatedly programmable structure, FPGA are used in many different sectors, such as aerospace, defense, Application Specific Integrated Circuit (ASIC) prototyping, and automotive. FPGAs have made hardware-accelerated computing a reality for many application domains, including image processing, digital signal processing, data security and communications. Until recently, such platforms required detailed hardware design expertise on the part of the application developer [12].

The driving forces of designing a system with FPGA are performance and stability. While CPUs offer versatility for general purpose computing, FPGAs offer more performance for specific and specialized operations. The main reason FPGAs outperform CPUs in time-critical operations is that they are basically an under-performing ASIC for any design they are programmed for. Another important difference between FPGAs and CPUs is latency

for an operation. As mentioned before, FPGAs have lower latency than CPUs for a set of operations. Moreover, the latency of FPGAs is deterministic. Determinism is important for a system because it shows that no random operations are involved in the system. If given the same initial conditions, the output of the system will always be the same. High-end modern FPGAs such as Zynq UltraScale+ MPSoC ZCU102 which is used in this project, consist of a programmable logic (PL) and a processing system (PS). This PL and PS side of the modern FPGAs can be used together to accelerate the design process or system as a whole. This PS side of the FPGA can run C code with a bare-metal design or environments such as PetaLinux and PYNQ can run on PS side of the FPGA to accelerate design productivity. In Section 1.3.1, detailed information about PYNQ can be found.

1.3.1. PYNQ

PYNQ is an open source project by Xilinx that uses the Python language and its libraries to develop applications for Zynq all programmable devices. PYNQ offers a Jupyter Notebook browser based interactive computing environment. PYNQ enabled boards can be easily programmed in Jupyter Notebook using Python. Using Python, developers can use hardware libraries and Overlays on the programmable logic [15].

The extraordinary part of PYNQ is Overlays, which is provided by Xilinx. A hardware design can be packaged as an Overlay and can be called like a function in Python. The communication between PS and PL sides of the FPGA handled by high speed AXI-DMAs. Overlays not only save time but also simplify the development process of any system as a whole. With Overlays, any bottleneck in the system can be custom designed as a hardware module to use in Python.

1.4. OBJECTIVE OF THE HARDWARE IMPLEMENTATION OF QKD

The final goal of this project is the creation and verification of a QKD-based communication system with FPGAs. The purpose of FPGAs for a QKD-based system is a necessity due to time critical operations. A secure generated key without in a reasonable time frame is not an appropriate fit for communication systems. To achieve this goal, first we will talk about

which quantum cryptography protocol to use in the project and the final outcome expected from the system.

Two quantum cryptography protocols we will compare for this project are BBM92 protocol and E91 protocol and the chosen protocol for this project is E91 protocol. BBM92 protocol has to do more computational operations than E91 protocol for generating a secure key. To generate a secure key with BBM92 protocol, it not only needs to generate a key and make error corrections for the key, with limited information that terminals share with each other, but it also needs to calculate the S value. The calculated S value determines whether or not the optical channel between terminals and EPS are listened by an eavesdropper. To generate a secure key with E91 protocol, generation of the key and error correction of the key with limited information that terminals share with each other is enough for secure key generation. Because E91 protocol checks the difference between two generated keys to determine whether or not optical channels between terminals and EPS are listened by an eavesdropper. This is already a major difference between the two protocols because E91 needs less computational operation to generate a secure key, but another reason to use E91 protocol in this project is cost. Implementing a QKD-based communication system with BBM92 protocol requires six photon detectors for each terminal, while E91 protocol requires only 4 detectors for each terminal. This difference in the number of photon detectors is important due to the necessity of high-end photon detectors needed to implement these protocols. These are the main driving forces to implement QKD-based communication system with E91 protocol for this project.

The final target for this QKD-based communication system is, while EPS sends up to five million entangled photon pairs to two terminals, extracting a single secure key within the next second with FPGAs. In Chapter 3, we will explore a QKD-based communication architecture design for E91 protocol to implement this protocol with FPGAs. Then a methodology for step by step explanation of how to reach the final architecture in the Chapter 3, also verification and implementation steps of these designed models included in Chapter 4. In the Chapter 5, we will share final outcomes of the implemented configurations which are explained in Chapter 4.

2. RELATED RESEARCH

The modern computer field is looking for new methods to create secure communication between devices due to advances in quantum computing. Quantum computing is a new topic, but with increasing demand for faster computational systems, quantum computing is improving rapidly. This interest in quantum computing also affects the quantum cryptography field due to, quantum computing can easily collapse current computer security systems. The part of the quantum cryptography field we will focus on is QKD-based communication systems with hardware acceleration. QKD-based communication systems with hardware acceleration are still a fairly new area of research; thus there is limited research focusing on this area. We will examine a few results from QKD-based communication system with hardware acceleration with usage of different protocols.

Implementation of BB84 protocol with T12 protocol setup by Lucamarini [16] has a maximum 2.20 megabits per second secure key generation rate with a loss rate equivalent to 35 kilometers length of fiber cable. Implementation of BB84 protocol with HD-QKD protocol setup by Zhong [17] has a maximum 2.70 megabits per second secure key generation rate with a loss rate equivalent to 20 kilometers length fiber cable. Implementation of BB84 protocol with HD-QKD protocol setup by Lee [18] has a maximum 5.3 megabits per second secure key generation rate with a loss rate equivalent to 38 kilometers length fiber cable. While the protocols used in QKD-based communication systems are different, the aim of this project is similar to the above research, which is, create a QKD-based communication system with hardware acceleration for secure key generation. In this project we want to generate a secure key in a second from 5 million entangled photon pairs with a range of loss rate which differs from 0 to 20 kilometers of length fiber cable. With 5 million entangled photon pairs, we expect to get an average of 2.15 megabits per second secure key generation rate. The separation between this project and other research is QKD-based setup. Before the QKD-based setup step, which is needed in real life operation, we will generate entangled photon information to test our system, due to building a QKD-based setup is an expensive and difficult operation. Details of entangled photon generation operation can be found in Chapter 4. With this generated data, we verify our reference model, then move our model to FPGAs for acceleration of the system.

3. ARCHITECTURE

In this chapter, we dive into a detailed explanation of the components in the terminals of QKD and their purpose for the system. We will examine two main architectures for the QKD which are the ideal QKD architecture, and the final QKD architecture. After the explanation of the ideal and final QKD architecture, we will move on to the detailed explanation of the inner modules QKD architecture.

Firstly, in Section 3.1 ideal QKD architecture, we look into how the QKD system should work without any physical restriction from the components and proposal of an ideal QKD architecture.

In the following Section 3.2 final QKD architecture, we will clarify the physical restrictions of the components and how to overcome these physical restrictions for achieving an optimal architecture for QKD system.

In the last section, we will explain the inner modules of QKD architecture and the purpose of each module in an explanatory way.

3.1. IDEAL QKD ARCHITECTURE

In an ideal scenario, QKD architecture has an EPS and two identical terminals. EPS generates entangled photon pairs and sends these entangled photons to terminals, and each terminal has three components, which are a Timestamp Unit (TSU) for detection and time-stamping the photons; a FPGA board for computational purposes, and a host device (HOST) for communication. Connections between EPS, terminals and components in the terminals for the ideal QKD architecture can be seen in Figure 3.1. Connection between terminals and components of the terminals connected to each other with Ethernet connection to achieve high-speed communication between components.

The steps of generating a secure communication with ideal QKD architecture starts with EPS generating entangled photon pairs and sending these photon pairs to each terminal’s TSU, which are TSU-1 and TSU-2 through an optical channel. Apart from this process,

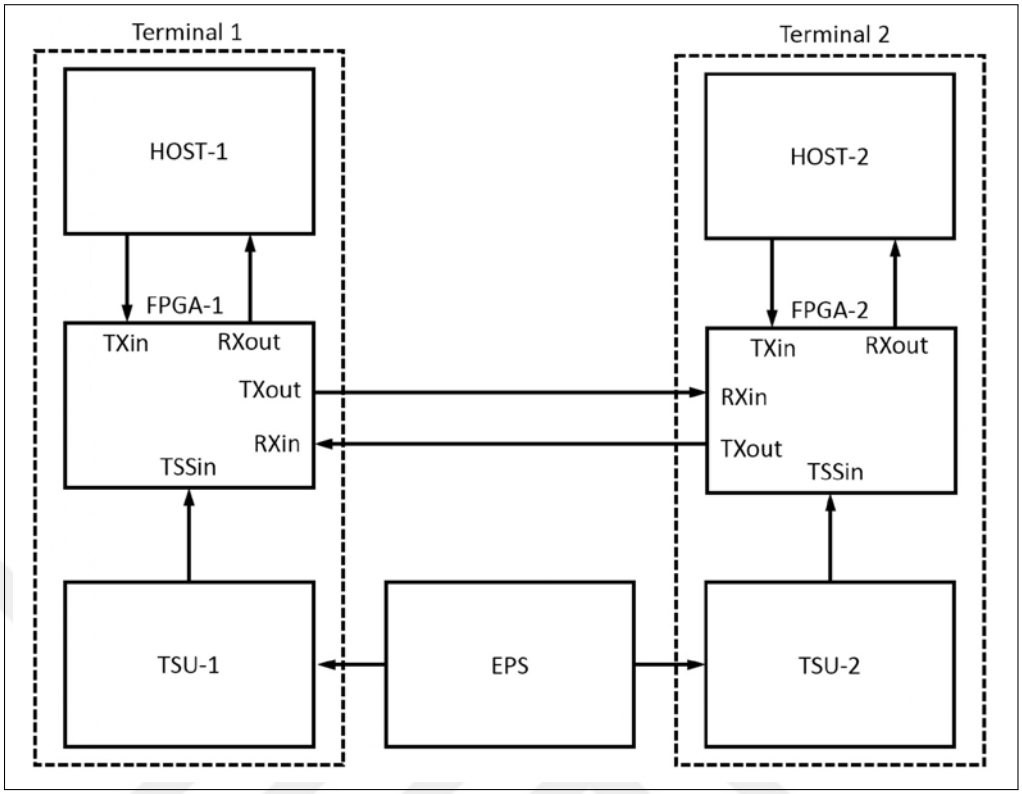

Figure 3.1. Ideal QKD architecture

all communication between terminals and components in the terminals is provided by each terminal's own FPGA. TSU-1 and TSU-2's purpose is to create a timestamp information whenever it detects a photon and send this detected photon's timestamp information to its own terminal's FPGA board through the TSSin port of FPGA. Whenever a timestamp information is received by a FPGA, FPGAs share necessary timestamp information through TXout with other terminal and receive shared timestamp information from other terminal via RXin. After both terminals' FPGA have their necessary timestamped photon pair information, they generate a secure key to use in public communication.

After the generation of a secure key, HOST-1 and HOST-2 are able to send encrypted data to each other. HOSTs can send data or messages to their own terminal's FPGA through TXin port. FPGA encrypts the received data or message with the generated secure key and sends it to other terminal's FPGA. FPGA that receives encrypted data, deciphers the received data or message with the generated secure key, and finally sends the deciphered data or message to it's own HOST via RXout.

As mentioned before, ideal QKD architecture assumes there are no physical restrictions on

components and is designed to be in the most optimal way possible for QKD system. In the Section 3.2, we will explain the physical restrictions of components and how we overcome these restrictions.

3.2. FINAL QKD ARCHITECTURE

The FPGA boards used in the project are Zynq+MPSoc ZCU102 and have a single Ethernet port. The main issue with the ideal QKD architecture is that each FPGA in the terminals needs at least two full-duplex and one half duplex physical Ethernet connection ports. Due to this scarcity of physical Ethernet port on the Zynq+MPSoc ZCU102, ideal QKD architecture evolved into the final QKD architecture design as shown in Figure 3.2.

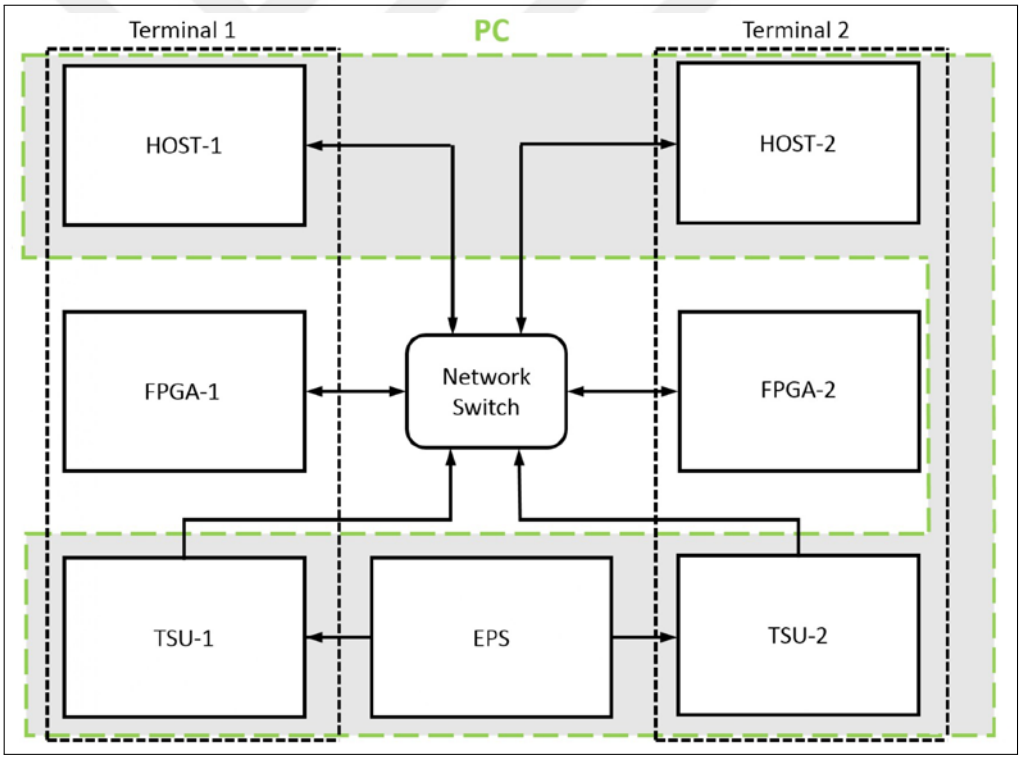

Figure 3.2. Final QKD architecture

While the idea is still the same as the ideal QKD architecture, the difference between ideal and final QKD architecture is that there are no direct Ethernet connections between terminals and components in the terminals. A network switch is used to connected all components together; thus communication between terminals and components in the terminals can be performed over single Ethernet port [19].

The part divided by dashed line (PC) in Figure 3.2 will be explained further in Section 4.3.1.

3.3. INNER MODULES OF QKD ARCHITECTURE

If we examine the components of a terminal in Figure 3.2, we can see that there is a HOST device which can be any kind of device with a Ethernet connection, a TSU which solely purpose is generating timestamp information whenever a photon received, and a FPGA to make computational operations. In this section, we will examine these computational operations which occur in the inner modules of the FPGAs in the terminals.

FPGAs at both terminals are programed with the same top-level design that consists of QCE and Ethernet physical instances. In Figure 3.3, top-level of FPGA-1 can be seen. The QCE module is responsible for all the operations that need to be done in the system, and Ethernet Physical is the physical Ethernet port of the FPGA which provides the connection between the network switch and QCE module to receive and send data.

Details of QCE module can be found in Section 3.3.1.

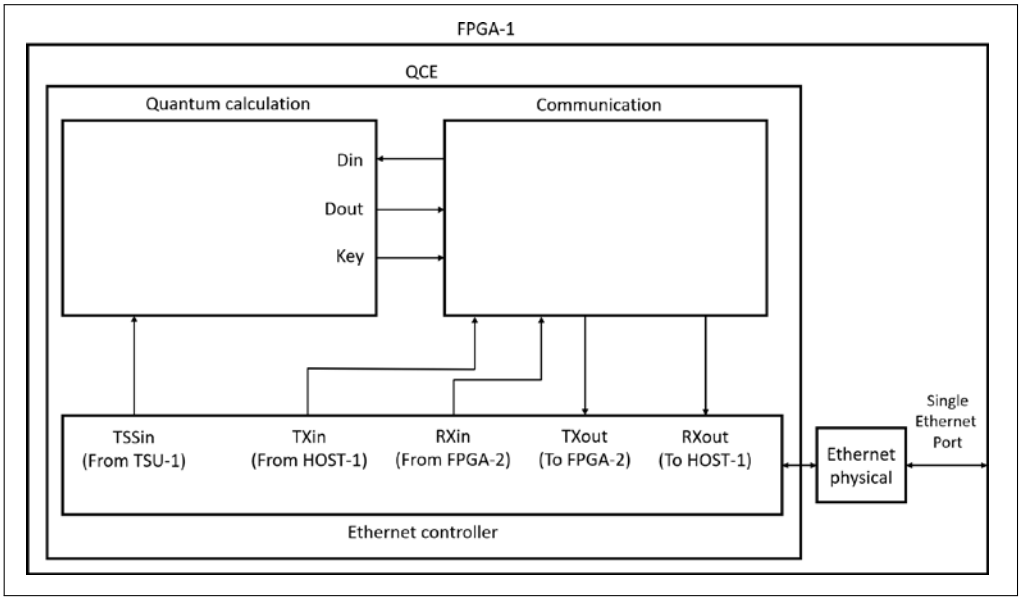

Figure 3.3. Top-level of FPGA-1

3.3.1. QCE Module

QCE module is the top module of the design which connects necessary connections between other modules in the design. QCE module basically a wrapper module for other modules in the design. QCE module gets its' name from main three modules in itself. Q from Quantum calculation module, C from Communication module, and E from Ethernet controller module.

Quantum calculation module responsible from all computational operations like correlation, time shift calculation, and key generation. Communication module handles the communication between Quantum calculation module and Ethernet controller module. Ethernet controller module is a demultiplexer for different data streams that arrive from Ethernet physical connection on the FPGA from a single Ethernet port. Ethernet controller module is the module that enables transformation of the ideal QKD architecture in Figure 3.1 to final QKD architecture in Figure 3.2.

Detailed explanations of Communication module, and Quantum calculation module can be found in Section 3.3.1.1, and Section 3.3.1.2.

3.3.1.1. Communication Module

The purpose of communication module is to manage the communication between FPGA, terminal's own host, corresponding terminal's host and corresponding terminal's FPGA.

Communication module manages the two inputs which data flows in, and two outputs which data flows out. These two inputs are TXin input, which contains data from corresponding terminal's host device or RXin input, which contains data from corresponding terminal's FPGA. Ethernet controller directs the incoming data from other devices to the necessary input points of the communication module. These two outputs are TXout which sends data to the corresponding terminal's FPGA, and RXout which sends data to terminal's own host device. Again, these outputs are connected to Ethernet controller module and Ethernet controller module handles the transfer of these data to the necessary devices.

Inner design of communication module of FPGA-1 can be seen in Figure 3.4. Communication module can be divided into two instances which a receive data modules and transmit data modules. Receive data modules consists RX, Parse, and Decipher modules. Transmit data

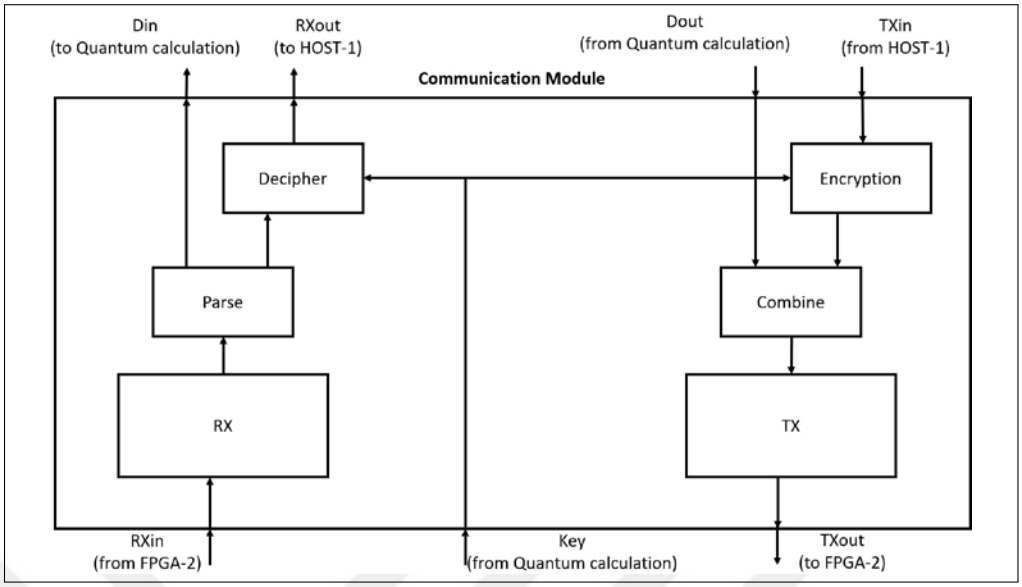

Figure 3.4. Inner design of communication module of FPGA-1

modules consists TX, Combine, and Encryption modules.

Purpose of receive data modules is to transmit received data from corresponding terminal to necessary modules. These received data can be corresponding terminal's TSS information, corresponding terminal's key negotiation information, or encrypted data from corresponding terminal. Any received data from RXin (from corresponding FPGA) input connected to RX module which is essentially a buffer. RX module connected to Parse module which determines whether data is TSS information, key negotiation information, or encrypted data. If received data TSS information or key negotiation information, Parse module directs the data to Din (to Quantum calculation module) output. Parse module directs the received data to Decipher module. If received data is encrypted data. Decipher module deciphers the received data coming from Parse module with Key (from Quantum calculation) and sends deciphered data RXout (to terminal's own HOST).

Purpose of transmit data modules is to transmit data to corresponding terminal. These transmitted data can be terminal's own TSS information, terminal's own key negotiation information, or data to encrypt and send corresponding terminal. Data received from Dout (from Quantum calculation) input can be terminal's own TSS information or terminal's own key negotiation information. Data received from TXin (from terminal's own HOST) input is data to encrypt and send corresponding terminal. Terminal' own TSS information and

key negotiation information is directly sent to Combine module and data received from TXin is sent to Encryption module which encrypts the received data with Key (from Quantum calculation) and sends encrypted data to Combine Module. Combine module marks data according to where it should be sent and sends marked data to TX module which is a buffer. Data inside of the TX module ready to send corresponding FPGA via TXout (to corresponding FPGA) output whenever Ethernet controller is available.

3.3.1.2. Quantum Calculation Module

Quantum calculation module is responsible for all logic operations in the system. These operations include finding the posedges of TSSs, correlation between TSSs to calculate time shift amount, secure key generation, and key negotiation between terminals. Finding the posedge of TSSs operation means finding the arrival point of entangled photon pairs by checking the density of detected photons in a time interval. After both posedge of TSSs are found, a correlation between found posedge is necessary to find time shift between terminals. After the time shift is calculated, a secure key can be generated. Lastly, a key negotiation operation between terminals is needed for error correction. Inner design of Quantum calculation module of FPGA-1 can be seen in Figure 3.4.

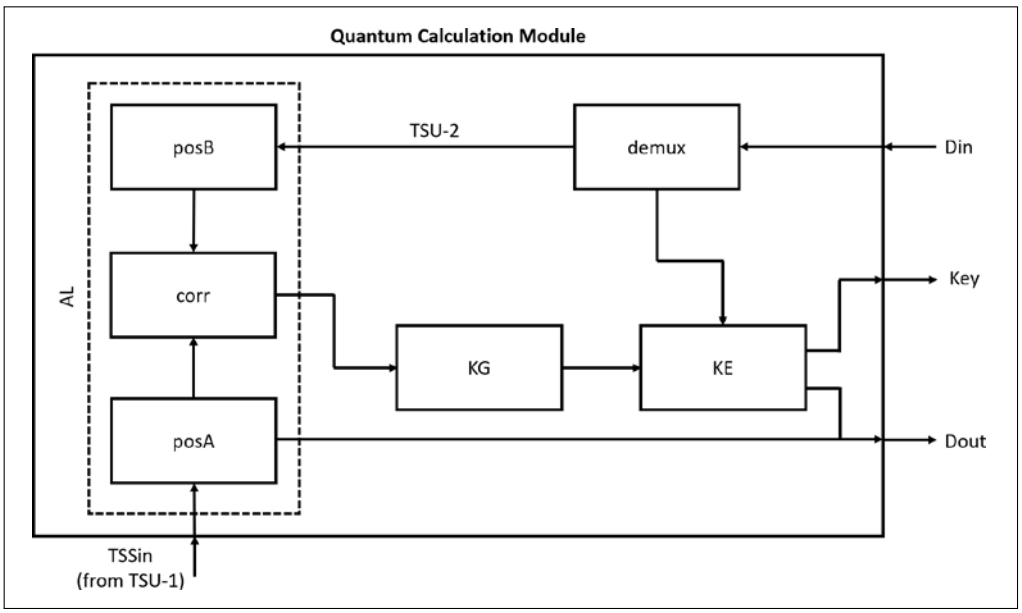

Figure 3.5. Inner design of quantum calculation module of FPGA-1

Quantum calculation module can be divided into four main modules which are a demultiplexer (demux), ALign (AL), Key Generation (KG), and Key Extraction (KE). Each module

responsible from one of the logic operations except module demux. Module demux is a demultiplexer for data coming from Din input. Data coming from Din input can be TSS information, or key negotiation information of corresponding terminal. If received data by demux module is TSS information of corresponding terminal, demux module sends these data to AL module. If received data by demux is key negotiation information, demux module sends this data to KE module.

The purpose of AL module is to find the posedges of TSSs and the time shift between TSSs. AL module consists of three sub-modules which are posA, posB, and corr modules. Design of modules posA and posB are the same and operation of this module is finding the posedge of the incoming TSS which facilitate the work load of the corr module. After finding the posedges of TSSs, posA and posB sends found posedge information to corr module which checks the correlation between two TSSs to find time shift amount between terminals, then send TSSs and time shift amount to KG module. With the known time shift amount between terminals, KG module generates a secure key and sends found secure to KE module. Lastly, KE starts a negotiation between corresponding terminal to generate identical key to use in a general communication channel.

4. METHODOLOGY

The objective of this project is to create a fully functional QKD system with FPGAs to accelerate time critical operations. To achieve this objective and verify the design, we divided problems into four phases. These phases are creating a time-stamp sequence generator (TSSgen), a Python model which replicates the behavior of the system (PyRef), testing this Python model with different configurations, and RTL implementation of QKD system.

The first phase is the creation of TSSgen. The necessity of TSSgen is for validation of the system. It is possible to get TSS information with TSUs, but there is no way to find out which detected photon pairs are surely entangled photon pairs. Therefore, to check if the system works as intended, it is obligatory to create a script which imitates the behavior of the EPS and TSU to generate TSSs. Another important reason to create TSSgen is, with TSSgen it is possible to create extreme case scenarios to verify the design for worst case scenarios. More comprehensive explanation of TSSgen can be found in Section 4.1.

The next phase is the creation of PyRef model which is a Python script that does the logic operations in the system. The reasons behind creating PyRef model are rapid implementation and verification of the design with high level synthesis of Python. The advantages of creating designs with Python are designs can be tested, and verified much faster than register-transfer level (RTL) design and with PYNQ framework PyRef can be easily imported to PS side of the FPGA. More detailed explanation of PyRef model can be found in Section 4.2.

The third phase is testing PyRef model with different configurations which are testing PyRef model on a single computer, testing PyRef model running on PS side of FPGA, and testing PyRef model running on PS side of FPGA while computational expensive parts moved to PL part of FPGA. With these three configuration steps, PyRef model can be imported to the intended environment without extra verification of the system. More details of testing PyRef model can be found in Section 4.3.

The last phase is RTL implementation of QKD system. The purpose of RTL implementation of QKD system is to achieve the best possible secure key generation speed for QKD system. Since the RTL implementation of the PyRef model is a must for completion of phase three, only the communication part of RTL implementation of QKD system is missing. More

details about needed communication module and elaborate explanation on how to achieve RTL implementation of QKD system can be found in Section 4.4.

4.1. TIME-STAMP SEQUENCE GENERATOR (TSSGEN)

The purpose of TSSgen is to create a script that emulates the behavior of EPS and TSU to generate TSS information to use in testing of the system. Normally, EPS generates entangled photon pairs to send these pairs to TSU through an optical channel, and TSU time-stamps the detected photons with photon detectors to create TSS. To create TSS information sets artificially, TSSgen has to imitate EPS and TSU behaviours. These behaviours of EPS and TSU are parameterized for creation of TSSgen thus TSSgen can create TSS information for different scenarios.

The algorithm of TSSgen use a seed to create a general randomness for all parameters which called seed_tss. With usage of a seed is identical TSS information can be generated. After the seed is set, TSSgen generates a TSS information for two terminals, which will be called TSS1 for Terminal-1, and TSS2 for Terminal-2. For generating TSS information, TSSgen follows a step-by step approach. Each step creates or adjusts a part of TSS information to achieve an emulation of EPS and TSU. Each step has its own parameters for generating TSS information for satisfying results. There are a total of eight steps in TSSgen which are entangled photon generation, splitting entangled photons, ambient photon generation, shifting TSS1 and TSS2, merging TSS1 and TSS2 with generated ambient photons, photon loss, jitter addition, and drift addition.

Entangled photon generation step generates a sequence of entangled photon pairs and uses below parameters.

- seed_e_min, seed_e_max: minimum and maximum range of randomness in entangled photons
- min_e, max_e: minimum and maximum time delay between two entangled photons
- size_e_min, size_e_max: minimum and maximum number of photon pairs in a sequence

- base1Rate, base2Rate: rate of the photons, arriving at base 1 and base 2 detectors
- start_e_t1_e_b2_e_min, start_e_t1_e_b2_e_max: minimum and maximum time delay difference of base 2 compared to base 1 in Terminal-1
- start_e_t2_e_b2_e_min, start_e_t2_e_b2_e_max: minimum and maximum time delay difference of base 2 compared to base 1 in Terminal-2
- key_err_rate: key error rate in key pairs

After the generation of entangled photons pairs in the entangled photon generation step, the splitting entangled photons step starts. In this step, generated entangled photon pairs split and two sequences are made from a single photon sequence which is called TSS1 and TSS2. After the splitting entangled photons step, each split sequence continuous next steps in parallel.

The third steps is ambient photon generation, which generates ambient photons for both sequences. Ambient photons have more time delay between each other than entangled photons. Ambient photons are generated by different seeds for TSS1 and TSS2 to generate different ambient photons for each sequence. Parameters of ambient photon generation can be seen below.

- min_1a, max_1a: minimum and maximum number of ambient photons for TSS1
- min_2a, max_2a: minimum and maximum number of ambient photons for TSS2

Next step is shifting TSS1 and TSS2, which shifts TSS1 and TSS2 by different amounts. Parameters of this step can be seen below.

- start_t1_b1_min, start_t1_b1_max: minimum and maximum time shift for TSS1
- start_t2_b1_min, start_t2_b1_max: minimum and maximum time shift for TSS2

The next step is merging TSS1 and TSS2 with ambient photons. In this step, generated ambient photons for TSS1 and TSS2 are merged with generated entangled photons of both sequences.

The next step is called photon loss. In this step to some random photons from TSS1 and TSS2 are deleted. This step is for replicating TSU due to, TSU can not detect every photon

sent from EPS. Parameters used in this step can be seen below.

- loss_1rate, loss_2rate: loss rate in TSS1 and TSS2
- loss_1seed_min, loss_1seed_max: minimum and maximum number of random deletion in TSS1
- loss_2seed_min, loss_2seed_max: minimum and maximum number of random deletion in TSS1

After photon loss, jitter addition step starts. In the actual system, photon pairs come with jitter, due to imperfections in the clock. In order to replicate this clock jitter, photons in TSS1 and TSS2 are shifted up and down in a range.

The final step of TSSgen is drift addition. The purpose of the drift addition step is to replicate clock drift in a real system. Only TSS2 scaled by a set amount in this step to replicate clock drift.

After the completion of every step, TSSgen generates two TSS information for Terminal-1 and Terminal-2, which are called TSS1 and TSS2. Both TSS sequences generated by considering behaviours of EPS and TSU. Due to detailed parameterization in TSSgen, different types of scenarios and extreme cases can be generated to use and test the system.

In Table 4.1, a small part of TSS1 generated by TSSgen can be seen. In Table 4.1 each row represent a single photon's information. There are three different information each photon have which are detection time, detector no, and ambient or entangled. Detection time represents the time that a photon is detected. Detector no represents which detector detected the photon. The last information is ambient or entangled indicates whether or not the detected photon is an ambient photon or an entangled photon. If ambient or entangled value is zero, it means the detected photon is an ambient. If ambient or entangled value is not zero, it means the detected photon is a entangled. Each entangled photon is enumerated in order to generate an expected key with TSS information. With this ambient or entangled information, it is possible to verify whether any key generated by the system is accurate or not. This information is not shared with any of the system, and is only used in verification.

Table 4.1. A part from TSS1 generated by TSSgen

Detection time	Detector no	Ambient or entangled
88265	4	0
90609	4	0
99434	1	0
100134	3	1
100450	4	2
100559	1	3
100797	1	4
101060	2	5
101303	2	6
101514	1	7
101582	1	8
101869	1	9
102332	3	11
102792	2	13

4.2. PYTHON REFERENCE MODEL (PYREF)

PyRef is Python implementation of the logic operations of the system. These logic operations PyRef does are ALign (AL), Key Generation (KG), and Key Extraction (KE) modules which shown in Figure 3.5 and inner design of PyRef can be seen in Figure 4.1. TSS1 represent TSS information of Terminal-1, and TSS2 represent TSS information of Terminal-2 in Figure 4.1. AL process finds the coincidences between entangled photon pairs between two TSS information; then with the coincidences between entangled photon pairs found, KG process creates a secure key with given data. Lastly, KE starts a key negotiation for error correction.

As in shown in Figure 3.5, AL process has three sub-processes which posA, posB, and corr. Each sub-process runs on a different thread to accelerate and imitate the behavior of a RTL design. Processes posA and posB do the same operation, which is to find arrival time of the first entangled pairs, then send the found entangled photon pairs to process corr. After process corr receives entangled photon pairs of that found in TSS from posA and posB, it

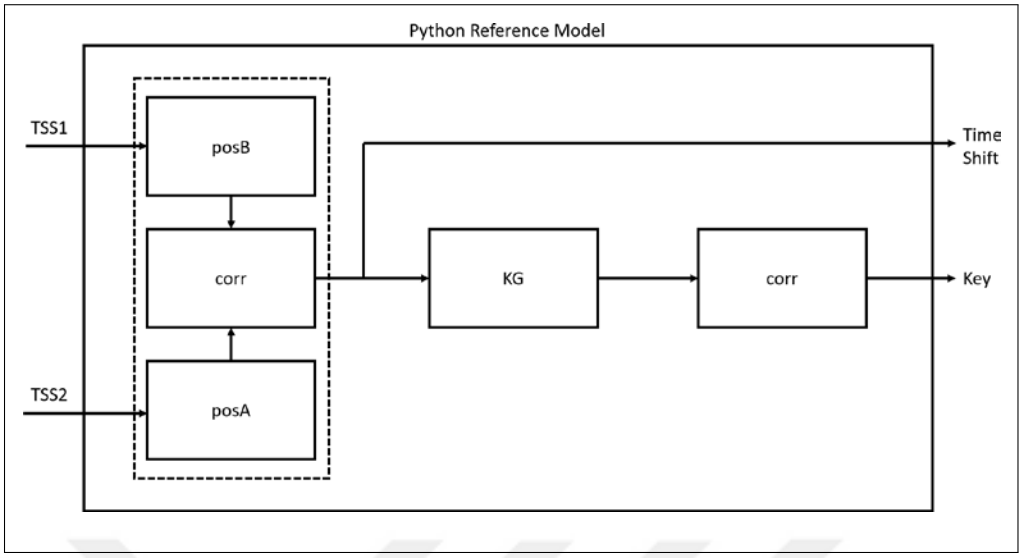

Figure 4.1. Inner design of PyRef

finds the correlation between two TSS information to find the time shift between. With the time-shift, coincidences between two TSSs can be found exactly to generate a secure key.

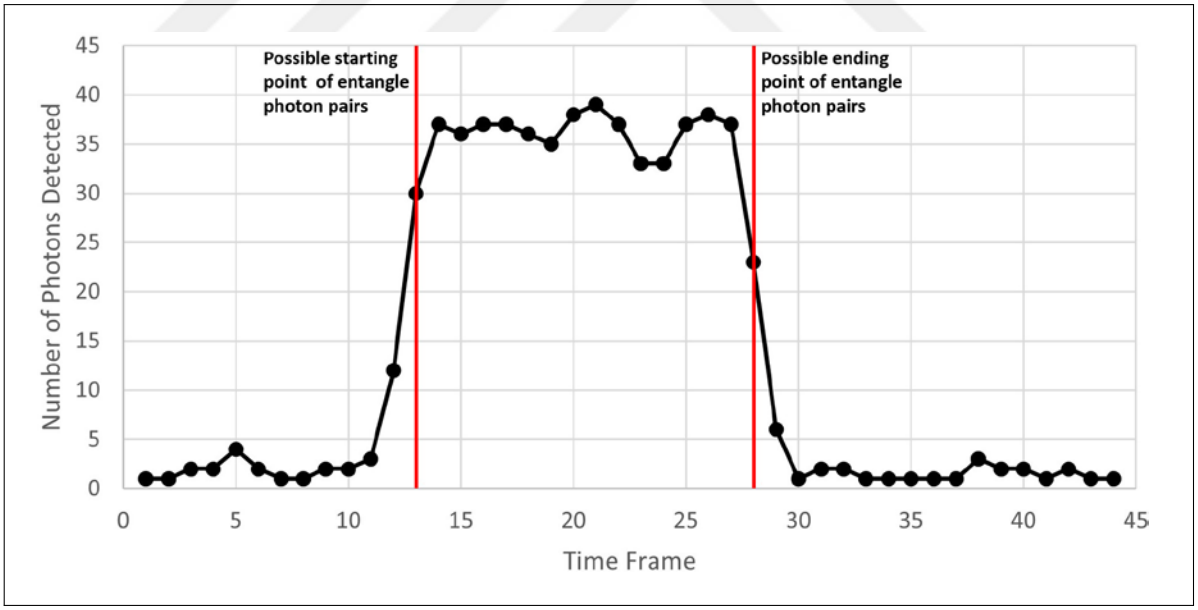

Figure 4.2. Number of detected photons within time frames

The purpose of posA and posB is to find the arrival time of the first entangled pairs, due to there is no way to know whether or not a detected photon is an ambient photon or an entangled photon. The idea of finding the arrival of the first entangled photon depends on the fact that whenever the frequency of detected photons increases drastically, it means entangled photons, which are sent from EPS are detected by photon detectors. In Figure 4.2 an example

of number of detected photons within time frames which generated by TSSgen can be seen. This example shows that whenever, entangled photons start arriving at photon detectors, a rapid raise in the number of detected photons occurs within a time frame. Process posA and posB look for a time frame with a high frequency compared to a set threshold. This point considered as starting point of the first entangled photons arrival. Process posA and posB start sending detected photon time-stamp information to corr starting from previous time frame when, this rapid raise occurs. When the frequency of number detected photons decreases under a threshold, process posA and posB consider this as the ending point of entangled photons and stop sending detected photon time-stamp information to corr after next time frame. These before and after time frames are needed to check for entangled photons due to, before a rapid raise or after a rapid drop in can happen, these time frames have a possibility of containing detected entangled photons. This starting and ending points found in TSS information called posedge.

After the posedges of both TSS information found, posA and posB send detected found information to corr for finding time shift between TSSs. The operation that corr does is, to find the correlation between two TSSs for finding time shift between them so that, coinciding entangled photon pairs between TSSs can be found. The time shift amount between two TSSs is calculated by Sparse Correlation method. The time shift between each possible detected photon pair is checked to find the highest correlation between them in small time frames. Number of detected entangled photons is correlated further than detected ambient photons over a looked time frame. This correlation operation needs to be done continuously over TSS information to find time shift between them because of, the clock jitter and drift in TSSgen and TSU. Even though the first founded time shift between TSSs are correct due to, clock jitter and drift in any electronic device, time shift between TSSs are continuously increasing or decreasing. Without checking time shift between TSSs continuously, number of founded coinciding photon pairs will be small part of the real coinciding photon pairs. After calculation of time shift in a time frame, coinciding photons in this time frame can be found and this operation is called matching. With this correlation method, detected photons are not aimed to be matched perfectly per photon pair. It is aimed to line detected photons close enough in an acceptable error amount, which will be referred to as blur amount. This correlation and matching operations are periodically repeated until corr stops getting data

from posA and posB. After matching of TSSs is finished, a secure key can be generated.

For key generation process, each terminal checks for their matching photons. Each terminal has two 2 bases. Each base consists of 2 photon detectors. Key is generated bit by bit with every matched photon for each base. Ideally, a pair of entangled photons should be detected by the same detectors on the same base, but there is a low error rate so terminals might have a different keys. For each base, key generation process starts with checking matched photons for detectors in the base. For each photon matched with detector-1's photons, a 1 added to the key. For each photon matched with detector-2's photons, a 0 is added to the key. This process happens for each base and two separate key generated for each terminal. After key generation, Terminal-1 starts a key negotiation and informs Terminal-2 with which key to use. With a key negotiation, errors in Terminal-2's key are corrected to Terminal-1's key. If there is a low amount of errors between two terminal's keys, the generated key is secure to use in general communication channels.

4.2.1. Verification of Python reference model

To verify PyRef, it is necessary to check the outputs of the model. There are two main outputs of the PyRef which are calculated time shift between TSSs, and generated key. With TSSs generated by TSSgen, the key and time shift between TSSs are already known.

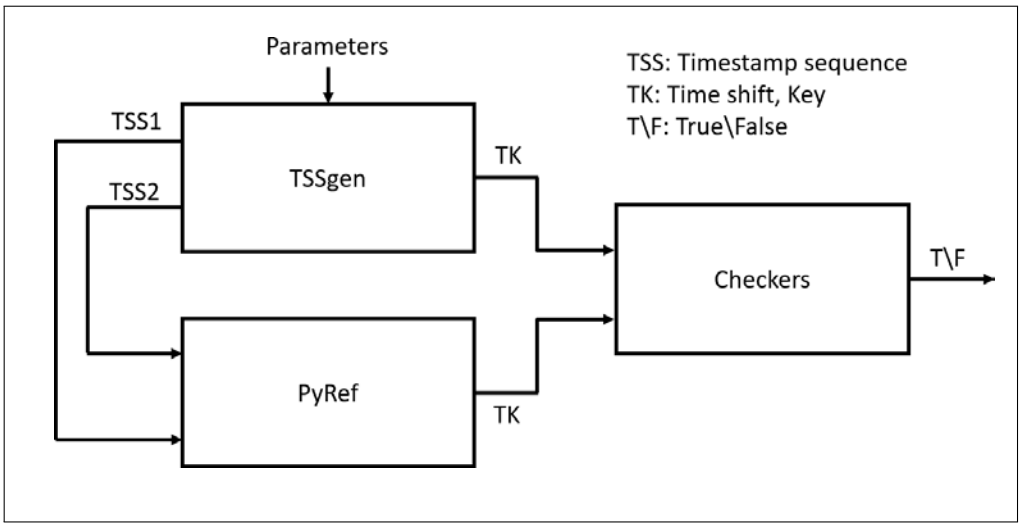

Figure 4.3. Verification model of PyRef

In Figure 4.3, verification model of PyRef can be seen. TSSgen generates two TSSs with given parameters, then sends the TSSs to PyRef. TSSgen also sends expected key and time shift between TSSs to a script called Checkers, which is used to compare the final outputs of PyRef. After PyRef finishes the operation to generate a key and find time shift between TSSs, sends found key and time shift between TSSs to Checkers. Outputs of two scripts are compared in Checkers. If the outputs are the same Checkers returns true; else returns false. With this model, verification of PyRef can be done for any scenario with different parameters. Another important part is with usage of seeds in TSSgen, scenario which returns false can generated again for debugging of PyRef.

4.3. TESTING PYREF MODEL

The proposed QKD system consists of an EPS, and two identical terminals which have a HOST, a FPGA, and a TSU. TSSgen makes it possible to emulate operations of EPS and TSU; thus with verified PyRef a complete QKD system can be created. Three configurations have been prepared to bring the intended QKD system on real hardware. Each configuration is a step towards achieving the intended QKD system on real hardware. These three configurations are testing PyRef on a single computer, testing PyRef model on PS side of FPGAs, and testing PyRef on FPGAs with Overlays.

4.3.1. Testing PyRef on a single computer

In testing PyRef model on a single computer purpose is creating a complete model of QKD system on a single computer. Aim of this configuration is verification of the created QKD system as a whole. To create this configuration on a single computer which simulates the whole QKD system, it is needed to use three terminal. Each of the terminals communicate to other terminal with TCP\IP or UDP protocol as in the real system. Two terminals run PyRef which makes the operations of FPGA, and the last terminal runs a script called feeder which sends the other two terminals TSS information generated by TSSgen. For TSS sharing and key negotiation, communication modules also need to be implemented and included in PyRef.

After testing PyRef is finished, we will move on to testing QKD system on real hardware.

4.3.2. Testing PyRef on FPGAs

In testing PyRef on FPGAs with help of PYNQ framework, we will move our design to a real hardware environment. With PYNQ framework, PS side of the FPGAs can run python scripts. Already tested and verified PyRef will be running on the FPGAs while, a computer simulates the behaviour of EPS, TSUs and HOSTs. The system in Figure 3.2, created with two FPGAs and a computer (PC in Figure 4.4) to test system on real hardware.

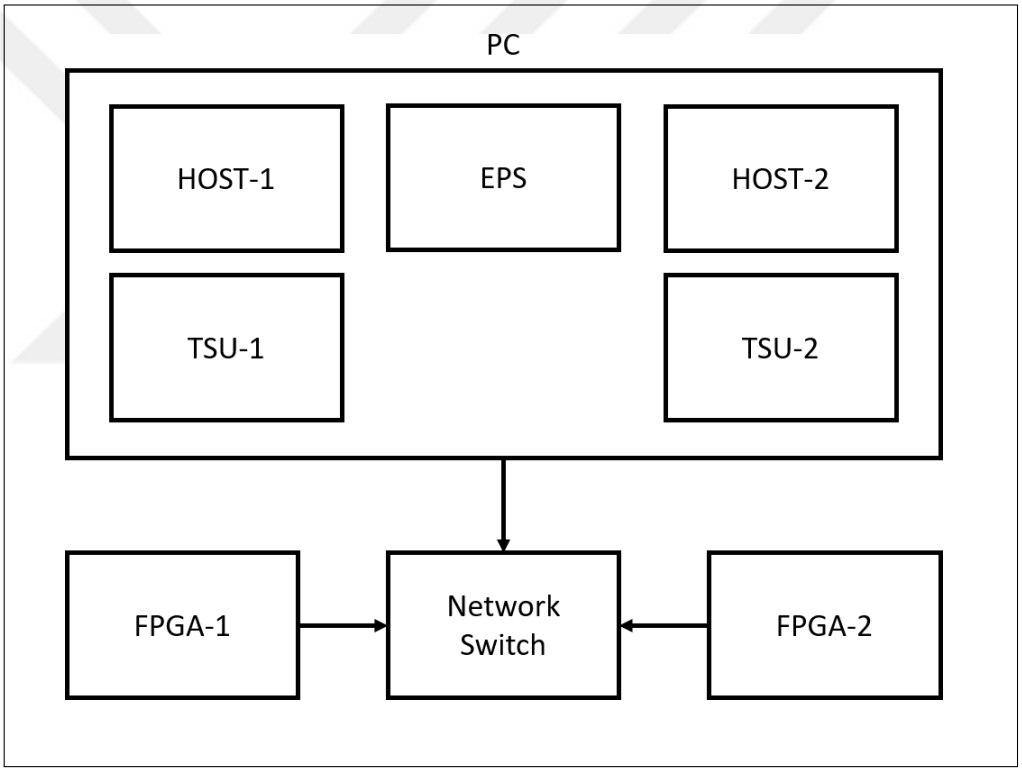

Figure 4.4. Testing PyRef on FPGAs

All three components to simulate real system connected to each other with a network switch and all communication between components are done over TCP\IP or UDP protocol.

After the testing PyRef on PS side of the FPGAs, we will move on to the acceleration of the model with Overlays provided by PYNQ in the Section 4.3.3.

4.3.3. Testing PyRef on FPGAs with Overlays

In the final part of testing PyRef model, the objective is to move computationally heavy parts of the PyRef to PL side of the FPGAs. These computationally heavy parts in the PyRef are finding the posedges, correlation between TSSs, and matching operations. These computationally heavy part referred as Quantum Calculation Critical in Figure 4.5.

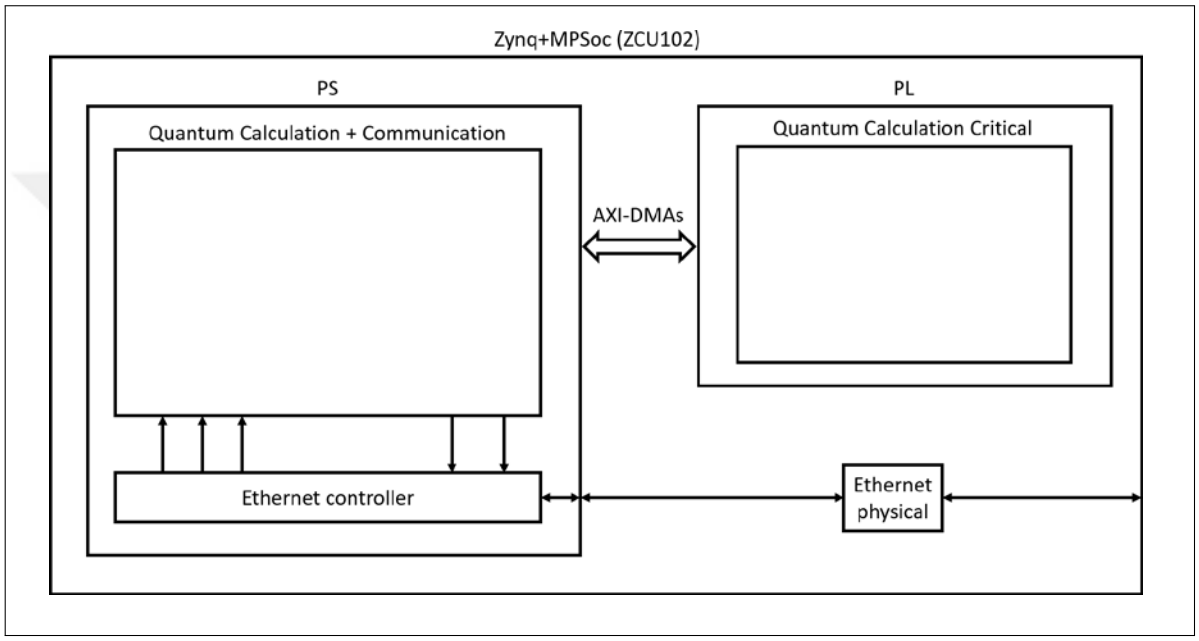

Figure 4.5. Inner design of a FPGA with Overlays

To use PL part of the FPGA with PYNQ framework, we need to design a custom Overlay for Quantum calculation critical parts in RTL. Each part of Quantum calculation design as an IP core and packed in Vivado. To sustain high data transfer between PS and PL sides of the FPGA, these IP cores are required to support AXI-DMA interface.

The design in the Figure 4.4 used again to verify this system. Only difference is PL side of the FPGA is used for acceleration the system.

4.4. RTL IMPLEMENTATION OF QKD SYSTEM

In the RTL implementation of QKD system objective is to implement and integrate already created RTL implementation of PyRef model with the necessary communication module of

QKD system. This necessary communication module is an Ethernet communication module. Block design of the QKD system with Ethernet communication module and other necessary modules can be seen in Figure 4.6.

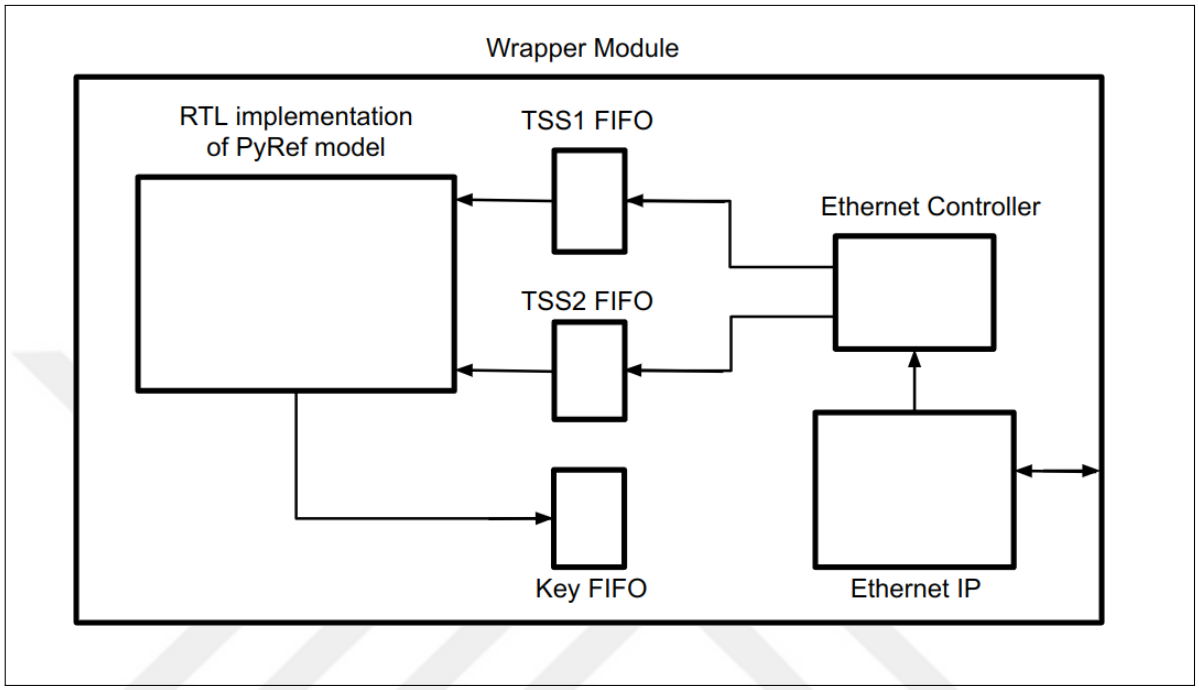

Figure 4.6. Block design of RTL implementation of QKD system

To accelerate the design process, we decided to use an open-source Ethernet IP which supports 1G Ethernet data transfer speed. With the Ethernet IP, the only needed module is an Ethernet controller between RTL implementation of PyRef model and Ethernet IP to control data stream coming from a physical Ethernet connection. To avoid clock domain crossing between Ethernet IP and RTL implementation of PyRef model, we decided to use double clock FIFOs for TSS1 and TSS2 information. These FIFOs store incoming TSS information from Ethernet control, which decides the data received from Ethernet IP whether TSS1 or TSS2 data information. Whenever both TSS FIFOs have data in them, RTL implementation of PyRef model starts reading from FIFOs to generate secure key information. RTL implementation of PyRef model finds the entangled photon pair from TSSs information and extracts a secure key. To store these key RTL implementation of PyRef model writes secure key bits to another FIFO called Key FIFO. After these steps, we can send extracted secure key to any device via Ethernet connection or an UART connection.

5. RESULTS

In this chapter, we will examine the results of three configuration of PyRef model, and RTL implementation of QKD system which explained in Section 4.3 and Section 4.4.

In Section 5.1, we will review the results of PyRef model working as a whole QKD-based communication system. Three terminals needed for this system are two PyRef models running in two different terminals as FPGAs, and a terminal for feeder module for sending TSS information as TSUs. All three terminals communicate each other with TCP\IP protocol to better simulate behaviour of the system. This configuration's main purpose is to verify PyRef model further and prepare necessary communication parts between these terminals which are needed in the next configuration.

In Section 5.2, we will examine the results of PyRef model running on the real hardware environment. In this configuration, two FPGAs and a computer connected to each other with a network switch. PyRef model runs on the PS side of two FPGAs with PYNQ framework, and a computer sends TSS information as TSUs with a feeder module. All three devices communicate each other with TCP\IP protocol. This configuration's main purpose is to move PyRef model to FPGA environment, to check transmission speed between devices and to examine run time of PyRef model for comparing results later with hardware accelerated final system.

Later in Section 5.3, we will examine finished steps for PyRef on FPGAs with Overlays. This configuration has two FPGAs and a computer connected to each other with a network switch which is same as setup of Section 5.2. Main purpose of this configuration is to accelerate the slow parts in PyRef with Overlays. To create Overlays for system, we need to design this slow parts of PyRef in RTL and verify this RTL designs. After verification of custom designed RTLs, we will proceed to Overlay production with these RTL modules to speed up the system.

Lastly in Section 5.4, we will test the performance of chosen Ethernet IP; then we test the whole system with two FPGA setup in Figure 4.4 with ATLYS model FPGAs. Reason for testing system with ATLYS FPGAs is physical Ethernet connection of ZCU102 is connected to PS side of the ZCU102. Since there is no direct Ethernet connection to the PL side of

the ZCU102 via the physical Ethernet connection, it is not possible to implement the RTL implementation of the QKD system. For this reason, we chose ATLYS board to implement and test.

5.1. RESULTS OF PYREF ON A SINGLE COMPUTER

The goals and results of this configuration are, after the creation of communication module between devices with TCP\IP protocol test the QKD-based communication system as a whole and check the system with under different cases which can be occur. Tests are done by two PyRef model runs in two different terminal as FPGAs, and a terminal for feeder module for sending TSS information as TSUs and three terminals communicate each other with TCP\IP protocol.

Table 5.1. Results of expected key difference rate between terminals and found key difference between terminals for 10 thousand entangled photon pairs

Expected Key Difference Rate	Found Key Difference Rate
0%	0.001%
1%	0.98%
2%	2.458%
3%	3.327%
4%	4.221%
5%	5.961%
6%	6.763%
7%	7.781%
8%	8.595%
9%	9.606%
10%	10.729%
11%	12.833%
12%	13.982%
13%	14.445%
14%	15.422%

The different cases of TSSs, we focus on is generating unique TSS with an expected key difference rate. It is possible to know expected key difference rate from keys generated by

two terminals because with TSSgen, TSS information can be generated for different occurring scenarios. To verify our system, we generated different TSSs with expected key difference rates to test our system. The test results for difference between expected key error rate and found key error rate can be seen in Table 5.1. Expected key difference rate is expected key difference between keys generated by terminals, since the TSS information is generated by TSSgen, it is possible to know exact keys terminal need to extract and difference between them. Found key difference rate is key difference between keys generated by terminals. For every expected key difference rate scenario, we generated 10 different TSS with this expected key difference rate and tested the whole system. Found key difference rates are the average key difference rates for these 10 tests.

From Table 5.1, it is possible to see the results of our system. Our extracted key from TSSs almost have the same difference rate as perfect scenario. The difference between expected key difference rate and found key difference rate occurs, due to terminals can match ambient photons with entangled photons by mistake. Moreover, we test our system with different numbers of entangled photons. Due to results of expected key difference rate and found key difference rate similar to each other with different number of entangled photons, we used 10 thousand entangled photons to verify our system.

With the results from Table 5.1, we can see that our system is working as intended and we can continue our project with moving our model to real hardware environment.

5.2. RESULTS OF PYREF ON FPGAS

The goals of this configuration are creating a PYNQ image for ZCU102 model FPGAs to move PyRef model to FPGAs, and test the system in real hardware environment to find slow parts of PyRef model.

To use PYNQ framework with ZCU102, it is need to create a PYNQ image for ZCU102. Due to Xilinx not providing a PYNQ image for ZCU102, we need to create a PYNQ image file for ZCU102 using a different board PYNQ image file. It is possible to create a PYNQ image for ZCU102 using PYNQ image files of ZCU104. After we generate a PYNQ image for ZCU102 with ZCU104’s PYNQ image files, we can access PYNQ environment on ZCU102.

To use PYNQ on ZCU102, we need to write ZCU102 PYNQ image file to a SD card. After SD card with PYNQ image is ready, we can insert the SD card to ZCU102 and set ZCU102 to SD card boot mode for accessing PYNQ environment. To access PYNQ environment for programming FPGA, we need an Ethernet connection between ZCU102 board and a computer. Once the Ethernet connection between the ZCU102 and a computer is provided, we can access the PYNQ environment with FPGA board IP in a browser. Accessed PYNQ environment in a browser can be seen in Figure 5.1. PYNQ environment uses Jupyter Notebook, which is an interactive computing platform.

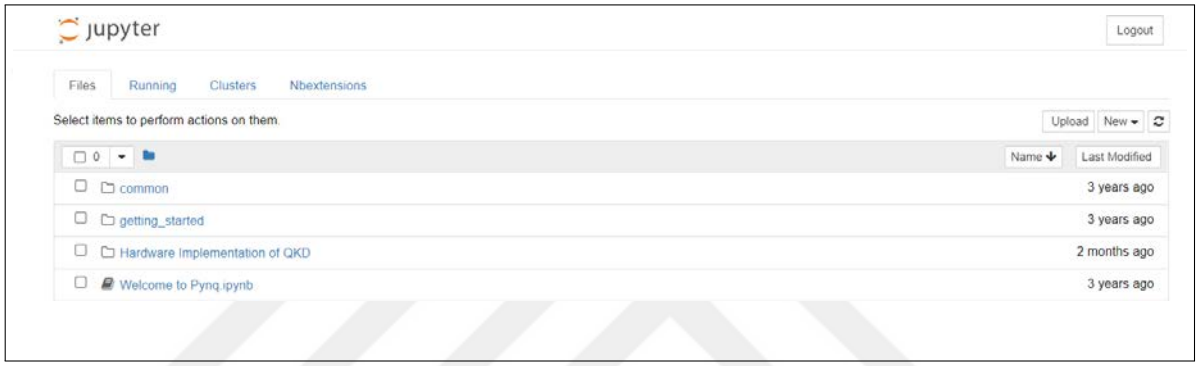

Figure 5.1. Accessed PYNQ environment in a browser

With the PYNQ environment, it is possible to run Python scripts on the PS side of the FPGA. To test PyRef model on ZCU102, we need to build a hardware setup with two ZCU102, a computer, and a network switch. Prepared setup can be seen in Figure 5.2.

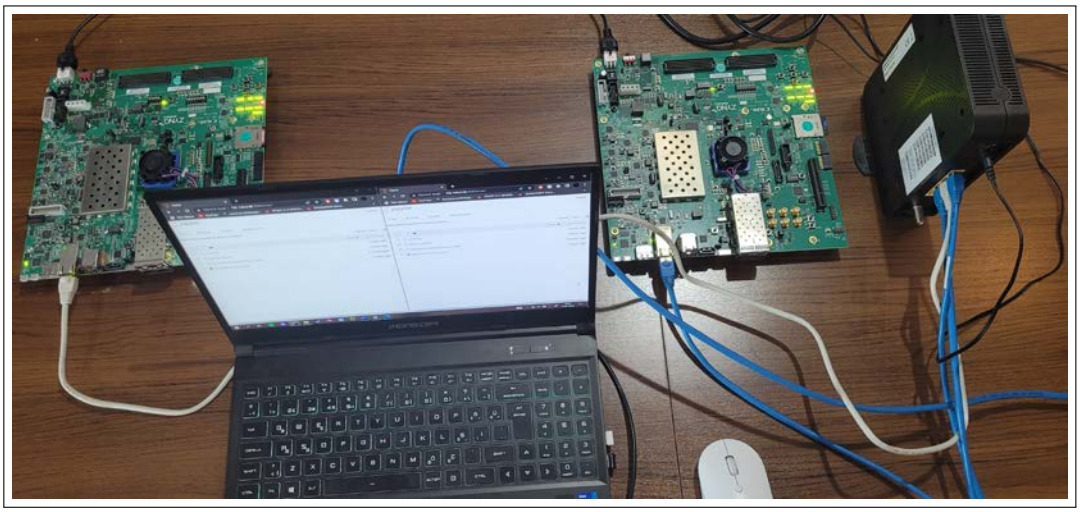

Figure 5.2. Setup created with two FPGAs, a computer, and a network switch

With the setup ready to use, we can upload PyRef to FPGAs via PYNQ environment and

test our system with real hardware. Transmission speed between FPGAs is a problem we need to admit, because sending 5 million entangled photons information takes 1.2 second on the average. After Overlays were created for acceleration of the system, we will look into this issue in more detail. For testing performance of the PS side of the FPGAs we created TSS with 1000, 10000, 30000, 100000, 200000, and 300000 entangled photons. While the FPGAs run PyRef model, computer runs a feeder to send TSS information to FPGAs via TCP\IP protocol. All communication between FPGAs again done via TCP\IP protocol.

Table 5.2. Performance results of PyRef on PS side of the FPGA

Number of Entangled Photons	Secure Key Generation Time (in seconds)
1000	0.122345018
10000	5.276523542
30000	45.00339947
100000	558.3742185
200000	2211.722558
300000	4861.560943

In Table 5.2, performance results of PyRef can be seen. Both FPGAs able to generate same secure key with given TSS information every time without failing, but the problem of PyRef model's performance on the PS side of FPGA. A higher number of entangled photons in a TSS dropped the performance of PyRef. The bottlenecks in the PyRef model are the computational parts of the model. These computationally heavy parts are finding posedges, correlation between TSSs, matching of entangled photons, and key generation in PyRef model.

With the PYNQ environment and PyRef model ready to use, we can continue our project with acceleration of the PyRef with creating Overlays for computationally heavy parts in the model.

5.3. RESULTS OF PYREF ON FPGAS WITH OVERLAYS

To accelerate PyRef model, we need to create Overlays for computationally heavy parts of the model. For the creation of Overlays, it is needed to implement and verify RTL design of these

computationally heavy parts. We completed the RTL design for posedge finding, correlation between TSSs, entangled photons matching and key generation. After the verification of all modules is completed, we will move on to creating custom Overlays to use these RTL modules in PyRef.

Figure 5.3. A part from simulation of RTL designs

Implemented RTL design of posedge finding, correlation between TSSs, entangled photons matching, and key generation makes the same operation as they do in PyRef model. The connections between all RTL designs are also the same as in PyRef model. We created a test-bench which contains all of these RTL modules. Before testing RTL modules we follow a few steps in order to control the final results of the RTL modules. These steps can be seen below.

- Generating a TSS information with TSSgen
- Extracting key and finding time-shift between TSSs using PyRef
- Controlling the results of PyRef model's key and time-shift between TSSs with TSSgen's key and time-shift between TSSs information
- If the found key and time-shift between TSSs by PyRef model correct, use this TSS information data in our test-bench
- Controlling results of PyRef and test-bench to verify the custom RTL designs

After the key extracted and time-shift between TSSs are found by PyRef, we can use this TSS information in our test-bench. We are giving same TSS information data to the test-bench,

in order to extract key and find time-shift between TSSs. After test-bench finishes its operation, we compare the extracted key and time-shift between TSSs found by test-bench and PyRef. A part from simulation with chosen signals of test-bench can be seen in Figure 5.3. In Figure 5.4, key_real1 signal represent the key generated by PyRef and key signal represent the key generated by RTL design. As it can be seen both generated key are exactly the same.

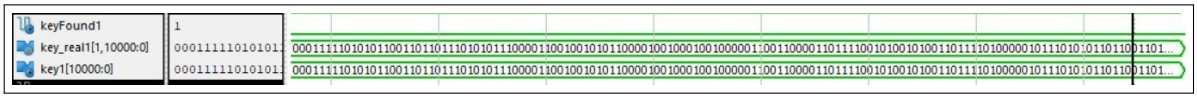

Figure 5.4. Comparison between key generated PyRef and RTL designs

If we check the performance of custom RTL design, we can see that with these custom RTL design it is possible to handle 50 million entangled photons within a second. Right now, RTL designs operate 10 times faster than needed. A performance decrease is expected due to the communication speed of the PL and PS sides of the FPGA, but the high performance of our RTL design is expected to compensate for this decline.

But there are some problems with Overlay configuration. The first problem is Ethernet data transmission speed. Because the physical Ethernet connection on ZCU102 is directly connected to PS side of the ZCU102, we need to make our communication between terminals and devices on PS side of the FPGAs. This occurs a problem in the system, due to TCP\IP protocol is a bottleneck for the system. It slows down the data transmission speed, and with UDP protocol PS side of the FPGAs have a loss rate of 15 percent. Another problem we encounter is, limited time frame we have for the project. After this configuration with Overlays, we will also create the same QKD system with only RTL. A dedicated RTL design for the system always performs better than this configuration with Overlays. After working on both parts at the same time for a while, we decided to cancel configurations with Overlays and solely focus on the RTL implementation of the QKD system.

5.4. RESULTS OF RTL IMPLEMENTATION OF QKD SYSTEM

In this final part, we will examine three different results. First, we will examine the Ethernet IP and performance of Ethernet IP. Then, integration between Ethernet IP and RTL implementation of PyRef model and verification of this integration. After the Ethernet

IP performance and integration of modules, we will explore the final results and performance of RTL implementation of QKD system.

To check the performance of Ethernet connection transmission speed we created a simple wrapper module for our Ethernet IP to use it. After setting wrapper module, our Ethernet IP was ready to use, and we examined the performance of Ethernet IP in three categories. These categories are data transmission speed, loss rate, and verification of received data.

Data transmission speed and loss rate are controlled together. We are using UDP protocol to send data to FPGAs because Ethernet IP we picked only supports UDP protocol. We tested data transmission speed and loss rate with two different setups. The first setup is ATLYS board directly connected to a computer via Ethernet cables. The second setup is ATLYS board and computer connected to a network switch via Ethernet cables. To check for loss rate, we added a register in wrapper module of Ethernet IP which counts the number of data received. We sent different-sized data to ATLYS board with a different number of packets, and packet sizes to find the best fit for our system. Every different number of packets and packet size was tested 10 times. Each data transmission speed and loss rate is an average of these 10 tests. We tested setups with gradually increasing sent data size. The result for ATLYS board directly connected to a computer can be seen in Table 5.3. Result for ATLYS board and computer connected to a network switch via Ethernet cable in Table 5.4.

Table 5.3. Test results for FPGA directly connected to a computer with an Ethernet cable

Number of Packets	Packet Size (Bits)	Transmission Speed (Gbps)	Loss Rate (%)
16384	6624	0.064731	0
16384	13248	0.123561	0
16384	26496	0.259232	0
32768	16032	0.543773	0
32768	24048	0.837697	0
32768	32096	0.957716	0

From Table 5.3, and Table 5.4 we can observe that there is not a significant difference in data transmission speed between setups. With Ethernet IP, we can get around 1G Ethernet speed, if needed without a problem. Another important result we got is loss rates. In both setups, we never observe any loss packets which is a very substantial outcome.

Table 5.4. Test results for FPGA and computer connected to a network switch with Ethernet cables

Number of Packets	Packet Size (Bits)	Transmission Speed (Gbps)	Loss Rate (%)
16384	6624	0.064663	0
16384	13248	0.122911	0
16384	26496	0.260448	0
32768	16032	0.541821	0
32768	24048	0.843941	0
32768	32096	0.943364	0

The next control we did on Ethernet IP is verifying the received data. To verify data received by ATLYS board, we created a simple UART transmitter module for sending data to a computer via serial port. We write received data from the Ethernet connection to a double clock FIFO in ATLYS board and send it back to a computer with UART protocol. To read data from a serial port in the computer, we used PuTTY. We tested the verification of received data with two different setups. The first setup was with ATLYS board and a computer connected to a network switch via Ethernet cable and the second setup was with two ATLYS boards and a computer connected to a network switch via Ethernet cables.

In the first setup, we send data to ATLYS board from a computer, and both of them are connected to a network switch. In the second setup, all devices are connected over a network switch. The computer sends data to the first ATLYS board, then the first ATLYS board sends received data to the second ATLYS board and we control the data received by the second ATLYS board over UART serial connection.

For testing the setups, we send "This is a test: CONTROL" to ATLYS boards. The second setup can be seen in Figure 5.5. The results of the first and second setups are the same can be seen in Figure 5.6.

For verification of the system, we created a simple UDP data sender script. In Figure 5.6, we executed our simple UDP data sender to send our message to ATLYS board over the network switch, and in PuTTY, we can see the same message sent back to the computer from ATLYS board in PuTTY terminal.

After testing and verification of Ethernet IP finished, we moved to integration between

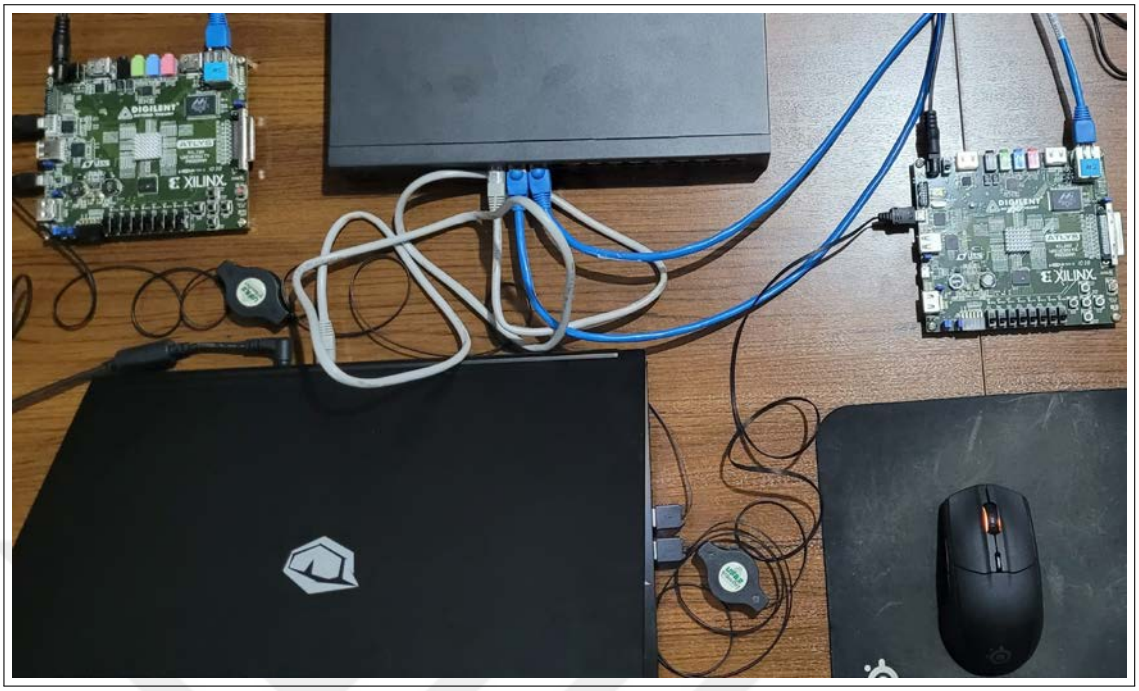

Figure 5.5. Setup with two ATLYS boards, a computer connected to a network switch

Ethernet IP and RTL implementation of PyRef model.

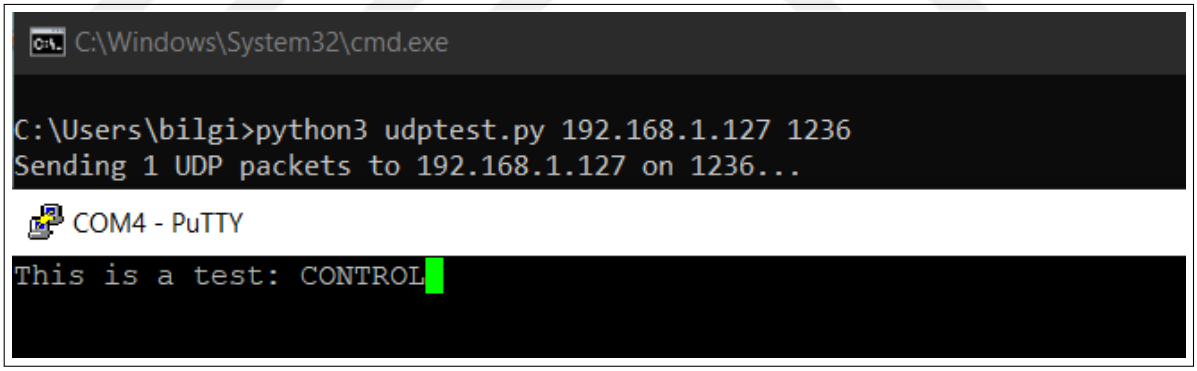

Figure 5.6. Result for verification of received data

For the integration between Ethernet IP and RTL implementation of PyRef model we need to implement two modules. The first module is for sending back TSS1 information to the corresponding ATLYS board, and the second module is an Ethernet controller to guide income UDP packets to necessary double clock FIFOs.

For sending TSS1 information to the corresponding ATLYS board, we used a module we already created in the verification of Ethernet IP to send data between FPGAs. For the Ethernet controller, we created a demultiplexer that inspects incoming UDP packet's IP and

port information and directs incoming data bytes to the necessary double clock FIFO.

After the integration between Ethernet IP and RTL implementation of PyRef is done, we started testing RTL implementation of QKD system with different numbers of entangled photon information. The main outputs we controlled are extracted secure key, and the time taken to extract a secure key. To control extracted secure key, we write it to a double clock FIFO and read it via our UART transmitter. For our final test, we used the setup in Figure 5.5. In this final test, we compared the extracted key from ATLYS boards with the key extracted by PyRef model and measured the time taken to extract a secure key. The results of this test with different numbers of entangled photon information can be seen in Table 5.5.

Table 5.5. Test results for RTL implementation of QKD system

Number of Entangled Photons	Key Extraction Time (ms)	Extracted Key Error Rate (%)
500	0.021096	0
2000	0.171816	0
5000	0.920648	0.0002
20000	5.079712	0.00036
50000	10.740464	0.00028

In Table 5.5, to calculate key extraction time we created a counter inside of RTL implementation of PyRef and to calculate key error rate we checked the key extracted by RTL implementation of PyRef with key generated by PyRef model with the same TSS information which generated by TSSgen. A secure key extracted from 500 entangled photon information by ATLYS boards can be seen in Figure 5.7.

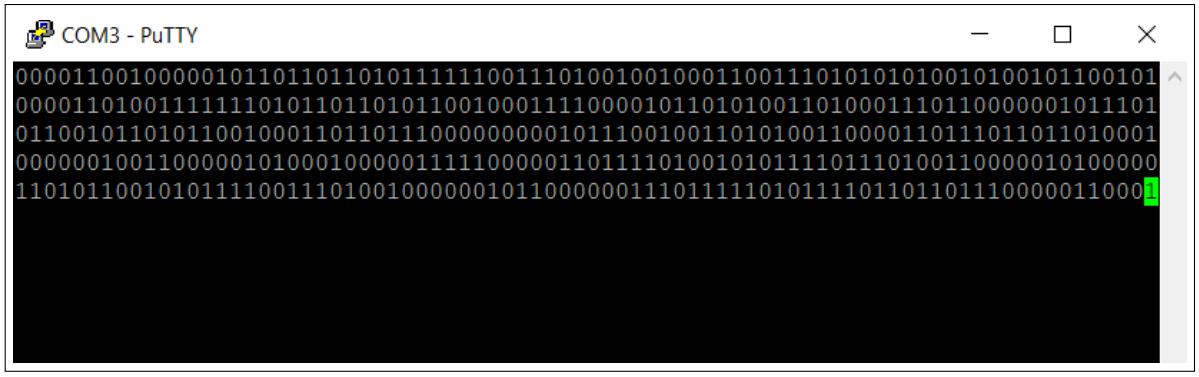

Figure 5.7. Extracted secure key from 500 entangled photon information

6. CONCLUSIONS

In this project, we proposed an architecture for QKD-based communication systems, create a reference module for the system, uniquely approached verification of the system and create a TSS information generation script, and created four different configurations for validating and accelerating the system.

We completed the reference module, TSS information generation script, and three configurations. We cancel PyRef on FPGAs with Overlays configuration due to the limited time frame. Even before starting on this configuration, we know that RTL implementation of QKD system will be faster than this configuration. However, we designed and verified RTL implementation of PyRef in this configuration which was a compulsory part of this project.

At the start of the project, our final target was extracting a secure from 5 million entangled photon pairs in a second. Our RTL implementation of QKD system can extract a secure key from 50 thousand entangled photon pairs in 10 milliseconds. Due to the deterministic linear time complexity of our RTL implementation of QKD system can handle 5 million entangled photon pairs in a second.

We accelerated the secure key extraction process with our RTL implementation. Before RTL implementation, our PyRef model has exponential time complexity and secure key extraction from 50000 entangled photons takes around 5 seconds. For 5 million entangled photons with exponential time complexity, it is not viable to extract a secure key within a reasonable time frame to use in real-time. With our RTL design, we accelerated the main bottleneck in the QKD system. The only missing part is a key error correction for the extracted secure key. The key error correction operation can be done with a CPU after the secure key is extracted in real-time.

REFERENCES

1. Sharma A, Ojha V, Lenka SK. Security of entanglement based version of BB84 protocol for Quantum Cryptography. *2010 3rd International Conference on Computer Science and Information Technology*. vol. 9. 2010:615-9.

2. McLeod S. Maslow's hierarchy of needs. *Simply psychology*. 2007;1(1-18).

3. Bellare M, Rogaway P. Introduction to modern cryptography. *Ucsd Cse*. 2005;207:16.

4. Kahate A. *Cryptography and network security*. Tata McGraw-Hill Education; 2013.

5. Damico TM. A brief history of cryptography. *Inquiries Journal*. 2009;1(11).

6. Goyal K, Kinger S. Modified caesar cipher for better security enhancement. *International Journal of Computer Applications*. 2013;73(3):0975-8887.

7. Stallings W. *Network security essentials: Applications and standards, 4/e*. Pearson Education India; 2003.

8. Stallings W. Cryptography and Network Security 2nd edition, p 23-24. Prentice-Hall, Inc. 1999.

9. Pirandola S, Andersen UL, Banchi L, Berta M, Bunandar D, Colbeck R, et al. Advances in quantum cryptography. *Advances in optics and photonics*. 2020;12(4):1012-236.

10. Ekert AK. Quantum cryptography based on Bell's theorem. *Physical review letters*. 1991;67(6):661.

11. Field Programmable Gate Array (FPGA) [cited 2022 10 June]. Available from: https: //www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html.

12. Nasri Sulaiman ZAO, Marhaban M, Hamidon M. Design and implementation of FPGA-based systems-a review. *Australian Journal of Basic and Applied Sciences*. 2009;3(4):3575-96.

13. Aafreen R, Abhishek R, Ajithkumar B, Vaidyanathan AM, Barve I, Bhattramakki S, et al. High-Performance Computing for SKA Transient Search: Use of FPGA based

Accelerators–a brief review. *arXiv preprint arXiv:220707054*. 2022.

14. Simpson P. *FPGA design*. Springer; 2010.

15. Xilinx Inc. pynq.io. Accessed: 2021-06-18. http://www.pynq.io/.

16. Lucamarini M, Patel K, Dynes J, Fröhlich B, Sharpe A, Dixon A, et al. Efficient decoy-state quantum key distribution with quantified security. *Optics express*. 2013;21(21):24550-65.

17. Zhong T, Zhou H, Horansky RD, Lee C, Verma VB, Lita AE, et al. Photon-efficient quantum key distribution using time–energy entanglement with high-dimensional encoding. *New Journal of Physics*. 2015;17(2):022002.

18. Lee C, Bunandar D, Zhang Z, Steinbrecher GR, Dixon PB, Wong FN, et al. High-rate field demonstration of large-alphabet quantum key distribution. *arXiv preprint arXiv:161101139*. 2016.

19. Bilgin Y, Tesfay SW, Ipek S, Uğurdağ HF, Durak K, Goren S. PYNQ-Based Rapid FPGA Implementation of Quantum Key Distribution. *2021 6th International Conference on Computer Science and Engineering (UBMK)*. IEEE. 2021:483-8.