

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Masterarbeit

zur Erlangung des akademischen Grades
Master of Engineering

Entwicklung einer FPGA-basierten Steuereinheit für Quantenkryptographieexperimente

vorgelegt von:

Georg Kewitsch

Matrikelnummer: s770490

In Zusammenarbeit mit:
Humboldt Universität zu Berlin
Institut für Physik / Nano Optics

Betreuer: Prof. Dr. S.H. Voß

Gutachter: Prof. Dr. R. Weis

Abgabetermin: 16.09.2013

Abstrakt

Das Bedürfnis des Menschen nach abhörsicherer Kommunikation hat im Laufe der Zeit zur Entwicklung vielfältigster kryptographischer Verfahren geführt. Beruhte deren Sicherheit anfangs noch auf der Geheimhaltung ihres Verschlüsselungsprinzips, so ist sie bei heutigen Verfahren nicht selten abhängig von der Computerrechenleistung. Ange-sichts der rasanten Entwicklung im Bereich der Computertechnik, ist es eine Frage der Zeit, bis aktuell als sicher geltende Kryptographieverfahren unbrauchbar werden. Die Quantenkryptographie schafft hier Abhilfe. Sie ermöglicht, durch Nutzung quantenmechanischer Gesetze, die Realisierung eines nachweislich sicheren Kryptographieverfahrens. Die praktische Umsetzung dieser Verfahren befindet sich allerdings noch in den Anfängen. Ein Forscherteam am Institut für Physik der Berliner Humboldt-Universität hat es sich deshalb zur Aufgabe gemacht, neue quantenkryptographische Protokolle zu entwickeln und bereits bestehende zu verbessern. Für deren Umsetzung in einem Laborversuch, wurde eine Steuereinheit benötigt, die die einzelnen Versuchskomponen-ten gemäß den Protokollen ansprechen und die benötigte Datenverarbeitung realisieren kann. Im Speziellen sollte dabei das BB84-Protokoll umgesetzt werden.

Im Rahmen der vorliegenden Masterarbeit wurde vor diesem Hintergrund eine Steuereinheit entwickelt, die eine solche Versuchssteuerung realisiert. Sie umfasst zwei voneinan-der unabhängige Module als Sender- und Empfangseinheit. Beide Teilsysteme bestehen aus einem FPGA-basierten Steuermodul, zum Ansprechen der Versuchskomponenten, sowie einer PC-Anwendung zur Gesamtkoordination. Zudem ist die Datennachverar-beitung gemäß BB84 realisiert. Beschrieben sind die einzelnen Schritte der System-entwicklung, von der Konzepterstellung über die elektronische Realisierung, die Schal-tungsentwicklung im FPGA, die Softwarerealisierung bis hin zur Inbetriebnahme und der Funktionsüberprüfung. Es werden weiterhin die Hintergründe zum physikalischen Versuchsaufbau und den verwendeten Basistechnologien beschrieben, die der Nachvoll-ziehbarkeit der Arbeitsschritte dient. Abschließend werden die Ergebnisse diskutiert und ein Ausblick auf mögliche Projekterweiterungen gegeben.

Abstract

The human need for tap-proof communications has led to the development of a wide range of cryptographic methods. For a long time the security of these schemes was based on the secrecy of their encryption principle only. In the methods used today, security is frequently also related to the computer processing power. Considering the rapid development of computer technology, it is only a matter of time until currently safe cryptographic methods become inadequate. Quantum cryptography, in contrast, allows to implement a verifiably secure cryptography method by using the laws of quantum mechanics. Practical implementations of these methods are still in their early stages. A team of researchers at the Institute of Physics at the Humboldt University in Berlin is currently working on a project to develop new quantum cryptographic protocols and to improve existing ones. To use such protocols in a laboratory experiment, a control unit is needed, which can control the setup components in accordance with the protocols and implement the required data processing. Specifically, the intention is to implement the BB84 protocol.

Within the present master thesis, a control unit has been developed, which realizes this kind of setup control. It comprises two independent modules: a transmitter and a receiver unit. Each sub-system consists of an FPGA-based module to control the setup components, as well as a PC application for overall coordination. In addition, the data post-processing is implemented in accordance with BB84. The thesis describes the steps of system development, from the first concepts, the electronic implementation, the circuit design for the FPGA and the software implementation up to commissioning and functional testing. For a better traceability, the background for the physical setup and the used base technologies are furthermore described. Finally, the results of the project are discussed and an outlook for possible project extensions is given.

Inhaltsverzeichnis

Abbildungsverzeichnis	V
Tabellenverzeichnis	VII
Listings	VIII
Abkürzungsverzeichnis	IX
1 Einleitung	1
1.1 Motivation	1
1.2 Hintergrund Kryptographie	2
1.2.1 Klassische Kryptographie	2
1.2.2 Quantenkryptographie	4
1.2.3 BB84	5
1.3 Physikalische Versuchsrealisierung	8
1.4 Systemanforderungen	9
2 Stand der Technik	11
2.1 Einzelheiten zum Versuchsaufbau	11
2.1.1 Einzelphotonenquelle	11
2.1.2 Elektrooptischer Modulator	13
2.1.3 Avalanche Photodiode	14
2.1.4 Zufallszahlengenerator QRNG	14
2.2 Hardwaredesign	15
2.2.1 FPGA	15
2.2.2 VHDL	19
2.2.3 Entwicklungsumgebung	20
2.2.4 Entwicklungsablauf	21
2.3 Softwaredesign	27

2.3.1	C++ und Qt	27
2.3.2	Qt-Creator	28
3	Systemrealisierung	29
3.1	Systementwurf	29
3.2	Elektronische Realisierung	33
3.2.1	FPGA Evaluationboard SP 605	33
3.2.2	Spartan 6	35
3.2.3	Zedboard	36
3.2.4	Zynq	37
3.2.5	UM232H	39
3.2.6	Adapterplatine	40
3.3	Systemdesign FPGA	41
3.3.1	Allgemein	41
3.3.2	Zwischenspeicherung der Daten	43
3.3.3	USB-Protokoll	44
3.3.4	Steuerung des Versuchsaufbaus	49
3.4	Softwarerealisierung	50
3.4.1	Überblick	50
3.4.2	GUI	52
3.4.3	Netzwerkkommunikation	54
3.4.4	USB-Kommunikation	56
3.4.5	Beschaffung der Zufallszahlen	57
3.4.6	Postprocessing BB84	57
3.5	Test und Inbetriebnahme	64
4	Zusammenfassung und Ausblick	70
Literatur- und Quellenverzeichnis		72

Abbildungsverzeichnis

1.1	Historische Kryptographieverfahren	3
1.2	Übersicht über die verwendeten Polarisationen	6
1.3	Schematische Darstellung des Versuchsaufbaus	8
1.4	Versuchsaufbau im Labor	9
2.1	Physikalischer Hintergrund zur Einzelphotonenquelle	12
2.2	Schematischer Aufbau der Einzelphotonenquelle	13
2.3	APD und EOM	14
2.4	Quantenbasierter Zufallszahlengenerator PQRNG 150	15
2.5	Vereinfachte Darstellung vom Aufbau eines FPGA	16
2.6	Beispielbausteine verschiedener Speichertechnologien	17
2.7	Konfigurierbarer Logikblock des XC2000 FPGA von Xilinx	18
2.8	Schaltbild eines I/O-Blocks des XC4000-FPGA von Xilinx	19
2.9	Programmoberfläche Xilinx ISE Design Suite	21
2.10	Beispielschaltung mit Xilinx Schaltplaneditor	22
2.11	Ergebnis einer Schaltungssimulation mit ISim	23
2.12	Darstellung des entwickelten Schaltungsdesigns	24
2.13	Programmoberfläche Qt Creator	28
3.1	Übersicht der aufkommenden Daten	30
3.2	Schematischer Gesamtaufbau	32
3.3	Xilinx SP 605	33
3.4	Blockschaltbild SP605	35
3.5	ZedBoard	37
3.6	Schematischer Aufbau des Zynq	38
3.7	USB-Modul UM232H	39
3.8	Platinenanschlüsse	40
3.9	Adapterplatine	41
3.10	Top-Level-Design	42

3.11 Schematische Darstellung eines FIFO-Elements	43
3.12 USB-Modul-Design	44
3.13 Lese- und Schreibzugriffe auf das USB-Modul	46
3.14 Datenpaket mit angehängten Zusatzinformationen	47
3.15 Zeitliche Darstellung der FIFO-Füllstände	48
3.16 Konfigurierbares Zeitfenster	50
3.17 Klassendiagramme	51
3.18 Grafische Benutzeroberfläche	53
3.19 Beispielhafte Darstellung des CASCADE Algorithmus	59
3.20 Misch-Algorithmus	63
3.21 V-Modell	65
3.22 Konfigurationsdarstellung zum Testen der USB-Verbindungen	68
3.23 Gesamter Systemaufbau mit Simulations-FPGA	69

Tabellenverzeichnis

1.1	Beispielhafte Verschlüsselung von Binärdaten nach dem One-Time-Pad	5
1.2	Tabellarische Darstellung von zehn Photonentransmissionen	7
3.1	Darstellung der Befehlsarten anhand von Beispielbefehlen	47

Listings

3.1	Schaltungsbeispiel für einen bidirektionalen Bustreiber	45
3.2	Realisierungsprinzip der serverseitigen Socket-Verbindung	54
3.3	Realisierungsprinzip der clientseitigen Socket-Verbindung	55
3.4	Verwendung der GMP-Bibliothek bei der Multiplikation großer Zahlen	64

Abkürzungsverzeichnis

Abb.	Abbildung
APD	Avalanche Photo Diode
API	Application Programming Interface
ARM	Acorn Risc Machine
AXI	Advanced eXtensible Interface Bus
BB84	Bennett Brassard 1984
BRAM	Block Random Access Memory
BNC	Bayonet Neill Concelman
CAN	Controller Area Network
CLB	Configurable Logic Block
CLK	Clock
CMD	Command
CMOS	Complementary Metal Oxide Semiconductor
COM	Communication Equipment
DAC	Digital Analog Converter
DDR-RAM	Double Data Rate Random Access Memory
DMA	Direct Memory Access
DSP	Digitaler Signal-Prozessor
DVI	Digital Visual Interface
EOM	Elektro-optischer Modulator
EPQ	Einzelphotonenquelle
FIFO	First In First Out
FMC	FPGA Mezzanine Card
FPGA	Field programmable Gate Array
FTDI	Future Technology Devices International
Gb	Gigabit
GHz	Gigahertz

GLP	GNU General Public License
GmbH	Gesellschaft mit beschränkter Haftung
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
h	horizontal
HDL	Hardware Description Language
HDMI	High Definition Multimedia Interface
I2C	Inter Integrated Circuit
ID	Identifikator
IEEE	Institute of Electrical and Electronics Engineers
IOB	Input/Output-Block
IP	Intellectual Property
ISE	Integrated Software Environment
I/O	Input/Output
JTAG	Joint Test Action Group
Kb	Kilobit
l	linkszirkular
LAN	Local Area Network
LED	Light Emitting Diode
LGPL	GNU Lesser Gerneral Public License
LPC	Low Pin Count
LUT	Look Up Table
LVDS	Low Voltage Differential Signaling
LVTTL	Low Voltage Transistor Transistor Logik
MB	Megabyte
Mbit	Megabit
MHz	Megahertz
MUX	Multiplexer
nm	nanometer
NGD	Xilinx Native Generic Database
OS	Operating System
PC	Personal Computer
PCIe	Peripheral Component Interconnect Express
PHY	Physical Layer (Treiberbaustein)
PIN-Diode	Positive Intrinsic Negative Diode
PLL	Phase Locked Loop

PMOD	Peripheral Modules
QKD	Quantum Key Distribution
QRNG	Quantum Random Number Generator
Qt	Quasar Toolkit
r	rechtszirkular
RAM	Random Access Memory
RSA	Rivest Shamir Adleman
RST	Reset
RTL	Register Transfer Level
s	Sekunde
SD	Secure Digital
SDIO	Secure Digital Input/Output
SFP	Small Form-Factor Pluggable
SMA	Sub Miniature A(kleine koaxiale Steckverbindung)
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SYNC	Synchronisationssignal
TCP	Transmission Control Protokol
TTL	Transistor-Transistor Logik
UART	Universal Asynchronous Receiver Transmitter
UCF	User Constraints File
USB	Universal Serial Bus
USB-OTG	Universal Serial Bus - On The Go
v	vertikal
V	Volt
VGA	Video Graphics Array
VHDCI	Very High Density Cable Interconnect
VHDL	Very High Speed Integrated Circuit Hardware Description Language
XCF	XST Constraint File
XOR	Exclusive Or
XST	Xilinx Synthesis Technology

1 Einleitung

1.1 Motivation

Seit jeher ist es ein Bedürfnis der Menschen, Nachrichten auszutauschen, ohne dass diese von anderen als den angedachten Empfängern gelesen werden können. Im Laufe der Zeit sind daher eine Vielzahl von Verfahren entwickelt worden, die diese geheime Übertragung ermöglichen sollten. Allerdings waren bisherige Verfahren nur zu einem bestimmten Maße und nur für einen beschränkten Zeitraum sicher. Immer wieder ist es Außenstehenden gelungen, hinter die Geheimnisse der einzelnen Verschlüsselungs-techniken zu kommen und diese somit unsicher und unbrauchbar zu machen. Durch den schnellen technischen Fortschritt ist es absehbar, dass in geraumer Zeit auch die aktuell als sicher geltenden Verfahren unbrauchbar werden. Basierend auf einem beweisbar sicheren Verschlüsselungsverfahren bietet die Quantenkryptographie durch die Gesetze der Quantenmechanik eine Methode, nachweislich abhörsicher zu kommunizieren. Die technische Umsetzung entsprechender Verfahren befindet sich allerdings noch im Anfangsstadium. In einem Forschungsprojekt der Arbeitsgruppe Nano Optics am Physikalischen Institut der Humboldt-Universität Berlin werden bestehende quantenkryptographische Protokolle weiterentwickelt und Verfahren für die Umsetzung neuer Protokolle erprobt. Um die dabei entwickelten theoretischen Entwürfe auf ihre praktische Umsetzbarkeit zu überprüfen, ist es erforderlich, diese in einem physikalischen Versuchsaufbau zu testen. Für die Umsetzung dieser Versuche bedarf es einer Steuereinheit, die ein zeitlich und logisch richtiges Ansprechen der einzelnen Komponenten im Versuchsaufbau sowie eine Verarbeitung der entstehenden Daten realisiert. Im Rahmen dieser Masterarbeit ist eine solche Steuereinheit für die Quantenkryptographieexperimente entwickelt worden. Die folgende schriftliche Ausarbeitung beschreibt die Entwicklung dieser Steuereinheit. Zu Beginn wird dabei eine Einführung in die Hintergründe der Kryptographie bis hin zur Quantenkryptographie gegeben. Zudem erfolgt eine Beschrei-

bung der gestellten Systemanforderungen. In Kapitel 2 sind allgemeine Hintergründe zum Versuchsaufbau und zu den Techniken der Umsetzung gegeben. Kapitel 3 erläutert die eigentliche Entwicklungsarbeit und beschreibt zudem die Inbetriebnahme und die Funktionsüberprüfung des Systems. Abschließend werden in Kapitel 4 eine Zusammenfassung des Projektes und ein Ausblick auf mögliche Projekterweiterungen gegeben.

1.2 Hintergrund Kryptographie

1.2.1 Klassische Kryptographie

Die Kryptographie beschäftigt sich mit Verfahren zur Ver- und Entschlüsselung von Nachrichten, um diese bei der Übertragung für Unbefugte unlesbar zu machen. Der Begriff stammt aus dem Griechischen und leitet sich ab von den Wörtern *kryptos* (geheim) und *graphein* (schreiben). Die Kryptographie ist ein Teilgebiet der Kryptologie, die sich zudem mit der Kryptoanalyse, den Methoden und Techniken zur Informationsrückgewinnung aus verschlüsselten Nachrichten, beschäftigt. Erste Formen der geheimen Nachrichtenübertragung sind von den frühen Hochkulturen aus Mesopotamien und Ägypten bekannt. Sklaven wurde dabei das Haupthaar rasiert und die Nachricht auf deren Kopfhaut tätowiert. Sobald das Haar ausreichend nachgewachsen war, konnte die Nachricht unbemerkt zum Empfänger übertragen werden. Andere Überlieferungen weisen auf die Informationsübertragung mit Holztafeln hin, die beschrieben und zum Schutz der Information mit Wachs überzogen wurden. Aus der Antike sind erste Verschlüsselungsverfahren für militärische Zwecke bekannt. Dabei wurde ein Streifen Leder oder Pergamentpapier um einen Stab eines bestimmten Durchmessers, der sogenannten Skytale, gewickelt und entlang des Stabes beschrieben. Abgewickelt ergaben die Buchstaben für Außenstehende keinen Sinn. Erst mit einem Stab des gleichen Durchmessers, um den man den Streifen erneut wickelte, war der Inhalt der Botschaft zu deuten. Besonders im militärischen Bereich ist die geheime Nachrichtenübermittlung von großer Bedeutung und bringt strategische Vorteile, die über Sieg und Niederlage entscheiden können. So wurden während der beiden Weltkriege kryptographische Verfahren weiterentwickelt und automatisiert. Im Vordergrund stand dabei die Entwicklung und Nutzung mechanischer und elektromechanischer Maschinen zur Ver- und Entschlüsselung. Bekanntestes Beispiel ist dafür die Chiffriermaschine Enigma, die von dem deutschen Elektroingenieur Arthur Scherbius entwickelt wurde.[2] Diese und die

beschriebene Skytale sind in Abb. 1.1 dargestellt.

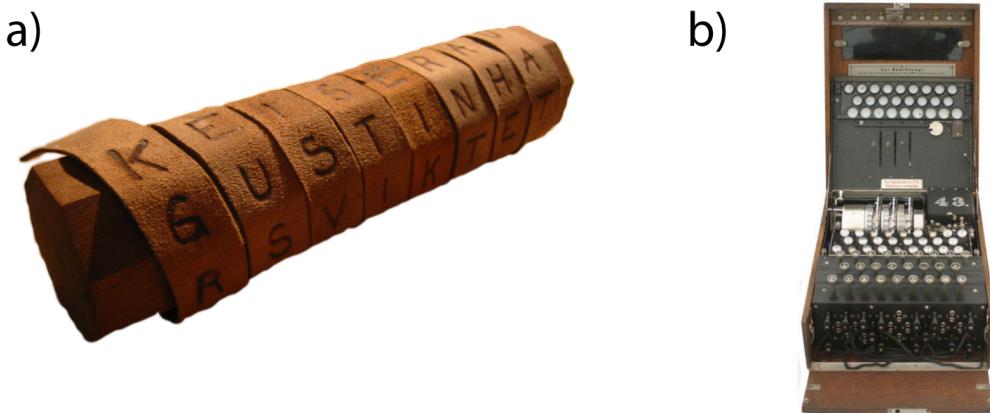


Abbildung 1.1: Historische Kryptographieverfahren: a) Skytale[12] und b) Chiffriermaschine Enigma[13]

Heutige Verfahren beruhen auf digitalen Methoden, da auch der Kommunikations- und Datenverkehr weitestgehend digital abläuft. Bei den ersten digitalen Verfahren wurden Algorithmen verwendet, deren Sicherheit von der Geheimhaltung ihrer Arbeitsweise abhing. Neuere Verfahren beruhen auf dem Einsatz eines digitalen Schlüssels. Die Sicherheit eines Kryptographiesystems hängt somit nicht mehr von der Geheimhaltung des Ver- und Entschlüsselungsalgorithmus sondern von der Geheimhaltung des Kryptographieschlüssels ab. Die Arbeitsweise des Algorithmus ist dabei zumeist bekannt. Grundsätzlich werden symmetrische und asymmetrische Verfahren unterschieden. Bei symmetrischen Verfahren wird zum Ver- und Entschlüsseln einer Nachricht der gleiche Schlüssel verwendet. Bei asymmetrischen Verfahren hingegen werden jeweils unterschiedliche Schlüssel verwendet. Wollen zwei Parteien über einen offenen Kommunikationskanal mit einem symmetrischen Kryptographieverfahren geheime Daten austauschen, besteht das Problem der sicheren Übermittlung des Schlüssels. Eine Lösung bieten hier bedingt sichere Schlüsselaustauschprotokolle wie beispielsweise der Diffie-Hellman-Schlüsselaustausch oder die Übergabe findet bei einem Treffen beider Parteien oder mit Hilfe eines vertrauten Boten statt. Mit Erfindung der Public-Key-Verfahren, Mitte der 1970er Jahren, wurde dieses Problem gelöst. Bei diesen nun asymmetrischen Verfahren erzeugt der Empfänger einer geheimen Nachricht einen öffentlichen Schlüssel, den er zum Sender über einen öffentlichen ungesicherten Kanal sendet, und einen privaten Schlüssel, den er für sich behält. Der Versender der Botschaft verschlüsselt die Nachricht mit dem öffentlichen Schlüssel und sendet sie über den öffentlichen Kanal. Nur der rechtmäßige Empfänger ist nun in der Lage die geheime

Nachricht mit seinem privaten Schlüssel zu rekonstruieren. Für Abhörer der Nachricht ist es nur mit extremen Zeit- und Rechenaufwand möglich, den privaten Schlüssel für die Entschlüsselung zu erraten oder zu berechnen. Dabei wird davon ausgegangen, dass die Zeit, die der Abhörer für die Beschaffung des privaten Schlüssels benötigt, länger ist, als ihm der Informationsgehalt der geheimen Nachricht von Nutzen wäre. Die Sicherheit derartiger Verfahren beruht auf der Lösung komplexer mathematischer Aufgaben, wie beispielsweise die Primfaktorzerlegung großer Zahlen beim Public Key Verfahren RSA. Mit der rasant steigenden Rechengeschwindigkeit von Computern und der bevorstehenden Entwicklung nutzbarer Quantencomputer, ist es absehbar, dass derartige Public-Key-Verfahren zukünftig keinen ausreichenden Schutz mehr bieten können.

1.2.2 Quantenkryptographie

Als einziges Kryptographieverfahren bietet das One Time Pad mathematisch beweisbare Sicherheit. Dieses symmetrische Verfahren wurde von Joseph Mauborgne entwickelt und ermöglicht vollständig geheime Datenübertragung insofern der Kryptographieschlüssel die folgenden drei Eigenschaften aufweist:

1. Die Schlüsseldaten sind rein zufällig.
2. Der Schlüssel ist genauso lang wie die Nachricht selbst.
3. Der Schlüssel wird nur einmal verwendet.

[3] Die zu verschlüsselnde Nachricht und die Schlüsseldaten werden dabei zeichenweise addiert. Das Ergebnis jeder einzelnen Addition wird durch die Gesamtanzahl der verwendeten Zeichen dividiert. Der Divisionsrest, üblicherweise durch die Modulooperation bestimmt, ergibt das entsprechende Zeichen der verschlüsselten Nachricht. Bei digitalen Nachrichten erfolgt die Verschlüsselung bitweise. Die Gesamtanzahl der verwendeten Zeichen ist bei einzelnen Bits 2. Dabei entsprechen Addition und Modulooperation einer Exklusiv-Oder-Verknüpfung ($\text{XOR } \oplus$) von Nachrichtenbits und Schlüsselbits. Eine beispielhafte digitale Verschlüsselung nach dem One Time Pad ist in Tabelle 1.1. dargestellt.

geheime Nachricht (A)	01010010 01001111 00101001 00110101 00101100
Schlüsseldaten (B)	10010110 10100110 10100111 11110010 10011000
verschlüsselte Nachricht (A \oplus B)	11000100 11101001 10001110 11000111 10110100

Tabelle 1.1: Beispielhafte Verschlüsselung von Binärdaten nach dem One-Time-Pad.

Die verschlüsselte Nachricht entspricht der XOR-Verknüpfung(\oplus) von geheimer Nachricht und Schlüsseldaten.

Trotz des Vorteils der beweisbaren Sicherheit bleibt dabei das Problem des sicheren Schlüsselaustauschs bestehen. Hier bietet die Quantenkryptographie eine Lösung. Sie ist kein Kryptographieverfahren im klassischen Sinn, sondern beschreibt Verfahren, die solch eine sichere Schlüsselverteilung ermöglichen. Im Verteilungsprozess Quantenkryptographie, oder auch Quantum Key Distribution (QKD) genannt, wird die Informationsübertragung mit Hilfe von photonischen Quantenzuständen, sogenannten Quantenbits oder Qubits, realisiert. Im wohl bekanntesten Protokoll der Quantenkryptographie, dem BB84, werden die einzelnen Informationszustände der Qubits anhand der Polarisation der Photonen unterschieden. Die Sicherheit solcher quantenbasierten Verfahren wird durch die Gesetze der Quantenmechanik gewährleistet, die für die einzelnen Photonen gelten. Diese besagen, dass es weder möglich ist, den Zustand eines einzelnen Photons zu bestimmen ohne diesen zu verändern, noch kann ein Photon in einem unbestimmten Zustand kopiert werden. Versucht also ein Unbefugter die Qubit-Übertragung abzuhören, indem er die einzelnen Zustände der Photonen misst, verursacht er zwangsläufig Fehler. Diese Fehler sind mit Hilfe von Stichproben statistisch erkennbar und ermöglichen es, einen stattgefundenen Abhörversuch zu erkennen.

1.2.3 BB84

Allein die Informationsübertragung mit Qubits bietet keine Abhörsicherheit. Erst ein entsprechendes Protokoll, welches die geltenden quantenmechanischen Gesetzmäßigkeiten ausschöpft, kann Sicherheit gewährleisten. Der erste Entwurf eines solchen Protokolls wurde 1984 von Charles Bennett und Gilles Brassard unter dem Namen BB84 vorgestellt.[4] Für die Modulation der Photonen werden dabei zwei verschiedene Basen verwendet, horizontal/vertikal (h/v) und diagonal beziehungsweise rechtszirkulär/linkszirkulär (r/l). Die Bitwerte 0 und 1 können innerhalb einer Basis beliebig zugeordnet werden, müssen aber bei Sender und Empfänger übereinstimmen. In Abb. 1.2 sind die

verwendeten Modulationen und ein jeweils beispielhaft zugeordneter Bitwert grafisch dargestellt.

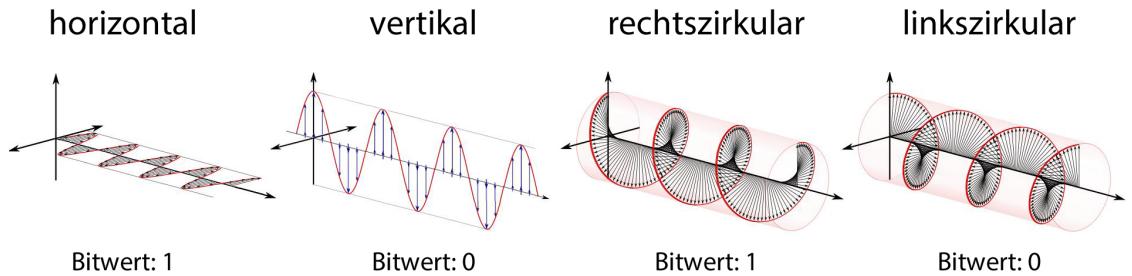


Abbildung 1.2: Übersicht über die verwendeten Polarisationen mit beispielhaft zugeordneten Bitwerten. Statt rechts- und linkszirkularen Polarisationen können auch diagonale Polarisationen verwendet werden.[14]

Der Sender, in der Kryptologie auch Alice genannt, sendet rein zufällig gewählte Bitwerte, die jeweils rein zufällig in einer der beiden Basen moduliert werden. Der Empfänger, Bob genannt, kann bedingt durch die Gesetze der Quantenmechanik in jeweils nur einer der beiden möglichen Basen ein Photon messen. Misst er es in der gleichen Basis, mit der es Alice ausgesandt hat, so kann er den Bitwert korrekt feststellen. Misst er in der falschen Basis, kommt es zu einem zufälligen Ergebnis. Eine Messung erfolgt dabei, indem das Photon einen Strahlteiler oder einen Filter mit der angenommenen Polarisation durchläuft und anschließend von einem Detektor erfasst wird. Nach der Übertragung der Qubits können Alice und Bob über einen offenen Kommunikationskanal die Wahl ihrer Basen austauschen, ohne dabei das Messergebnis zu verraten. Zudem teilt Bob mit, an welchen Transmissionen er kein Photon detektiert hat. Nachdem nun Sender und Empfänger die Wahl der Basis des jeweils anderen kennen und zudem wissen, an welchen Transmissionen Bob kein Photon detektiert hat, können diese Messergebnisse auf beiden Seiten aussortiert werden. Tabelle 1.2 zeigt eine beispielhafte Übertragung von zehn Einzelphotonen. Grau unterlegt sind Messergebnisse, die aussortiert werden.

Alice					Bob			
Nr.	Basis	Photon	Bit		Basis	Filter	Detektion	Bit
1	(h/v)	h	1	(h/v)	h	ja	1
2	(h/v)	v	0	(r/l)	l	nein	-
3	(r/l)	l	0	(r/l)	l	ja	0
4	(h/v)	v	0	(r/l)	r	ja	1
5	(r/l)	r	1	(r/l)	r	ja	1
6	(h/v)	h	1	(h/v)	v	nein	-
7	(h/v)	v	0	(r/l)	l	nein	-
8	(r/l)	l	0	(h/v)	h	ja	1
9	(r/l)	l	0	(r/l)	l	ja	0
10	(r/l)	r	1	(h/v)	v	nein	-

Tabelle 1.2: Tabellarische Darstellung von zehn Photonentransmissionen. Dargestellt sind auf der Seite von Alice die Basis, in der das Photon moduliert ist, die tatsächliche Polarisation des Photons und der zugeordnete Bitwert. Auf der Seite von Bob ist dargestellt, in welcher Basis eine Messung erfolgte, welcher Filter für die Detektion verwendet wurde, ob die Detektion erfolgreich war und welcher Bitwert aus der Messung resultiert. Grau unterlegte Transmissionen werden aussortiert, da sie entweder ungültig sind oder weil bei Alice und Bob verschiedene Basen gewählt wurden.

In einem idealen Experiment würden nun beide Parteien über identischen Schlüssel verfügen. Real kommt es allerdings aufgrund von Streulichteinfluss und Fehlausschlägen der Detektoren zu Messfehlern und somit zu Unterschieden in den Schlüsseldaten von Alice und Bob. Mit einer Fehlerkorrektur, die in Kapitel 3.4.6 Postprocessing BB84 näher beschrieben wird, können diese Fehler beseitigt werden.

Versucht ein Abhörer, Eve genannt, in den Quantenkanal einzugreifen, indem er beispielsweise die Photonen von Alice abfängt, misst und erneut an Bob aussendet, ist es auch ihm nur möglich, in einer von zwei möglichen Basen zu messen. Demzufolge kann Eve auch nur mit einer Wahrscheinlichkeit von 50% ein richtiges Messergebnis erzielen. Sendet Eve die Photonen in Basis und Bitwert entsprechend der fehlerhaften Messungen an Bob aus, verursacht sie eine erhöhte Bitfehlerrate bei Bob und kann so erkannt werden.

1.3 Physikalische Versuchsrealisierung

Im Rahmen des in Kapitel 1.1 beschriebenen Forschungsprojekts zur Verbesserung und Neuentwicklung quantenkryptographischer Protokolle ist am Institut ein Versuchsaufbau für die Umsetzung des BB84-Protokolls realisiert worden. Dieser ist weniger für real angewandte Schlüsselverteilung und die geheime Datenübertragung ausgelegt, als viel mehr für Testzwecke, um verschiedene Varianten der physischen Protokollrealisierung zu testen. Sender und Empfänger sind dementsprechend in geringer Entfernung voneinander platziert. Abb. 1.3 zeigt eine schematische Darstellung des Versuchsaufbaus.

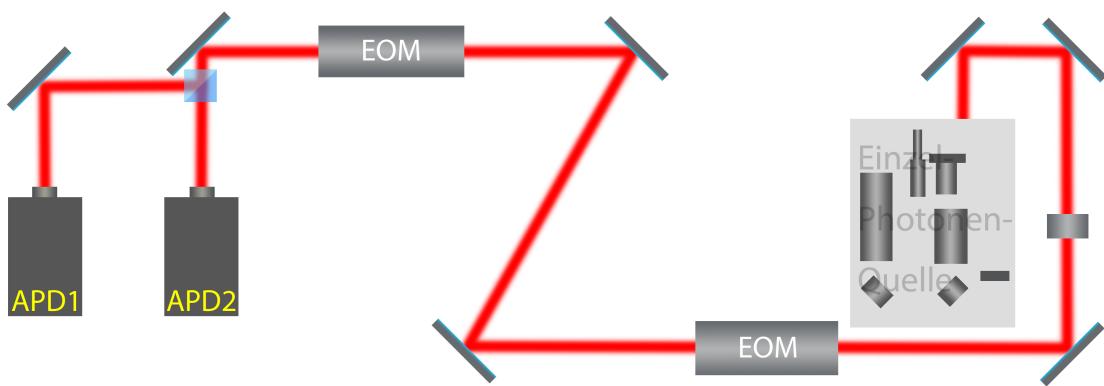


Abbildung 1.3: Schematische Darstellung des Versuchsaufbaus. Einzelne Photonen werden von der triggerbaren Einzelphotonenquelle erzeugt und mit Hilfe von Umlenkspiegeln durch die elektrooptischen Modulatoren(EOM) und einen polarisierten Strahlteiler geleitet. Die Avalanche Photodioden(APD) ermöglichen die Detektion der einzelnen Photonen beim Empfänger.[34]

Die Senderseite verfügt über eine triggerbare Einzelphotonenquelle. Die Empfängerseite beinhaltet Avalanche Photonen Dioden (APD) als Detektoren um das Auftreffen einzelner Photonen zu signalisieren. Um für alle Photonen eine einheitliche Ausgangspolarisation zu erreichen, durchlaufen sie nach dem Austritt aus der Photonenquelle ein vertikales Polarisationsfilter. Da die Photonenquelle hochpolarisierte Photonen aussendet, sind am Filter Verluste kleiner 10% zu verzeichnen. Die für die Protokollumsetzung benötigten vier Polarisationszustände werden senderseitig mit Hilfe eines elektrooptischen Modulators (EOM) erzeugt. Dieser kann in Abhängigkeit einer angelegten Spannung die Polarisation eines Photons verändern. Die Übertragung der einzelnen Lichtteilchen erfolgt per Freistrahl, ohne optischen Leiter. Treffen sie beim Empfänger

ein, kann ihre Polarisation erneut durch einen EOM verändert werden. Mit Hilfe dieses EOM kann die Polarisation der Photonen von zirkulär zu horizontal/vertikal verändert werden. Ein darauffolgender polarisierter Strahlteiler separiert die horizontal und vertikal polarisierten Photonen. Hinter jedem der Ausgangsrichtungen des Strahlteilers befindet sich ein Photonendetektor, der das Auftreffen eines Photons durch einen kurzen Spannungspuls signalisiert. Der gesamte Versuchsaufbau ist optisch abgeschirmt um das Eintreffen anderer Lichtteilchen zu verhindern. In Abb. 1.4 ist der reale Versuchsaufbau dargestellt.

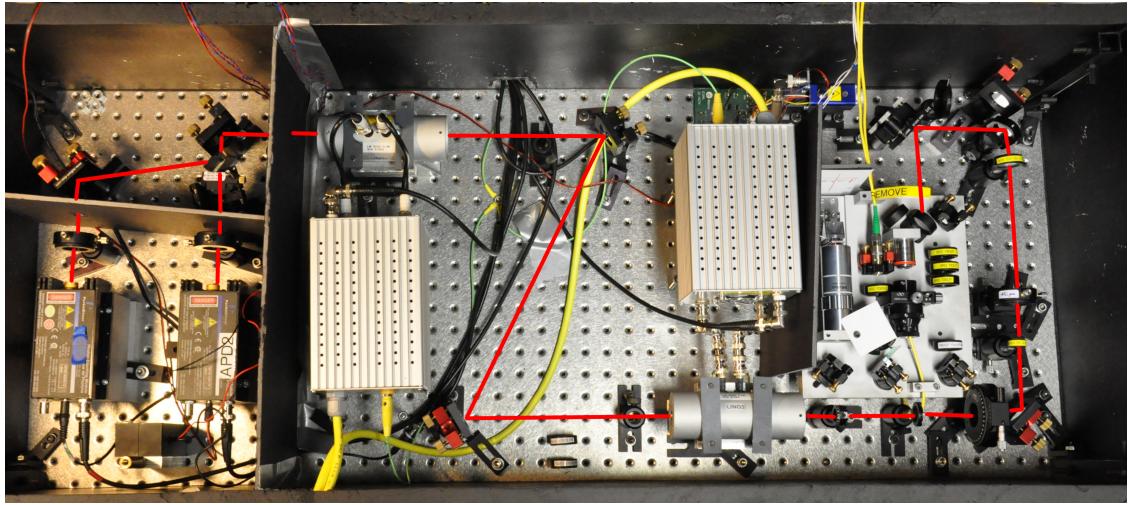


Abbildung 1.4: Versuchsaufbau im Labor. Dieser realisiert die Umsetzung des BB84-Protokolls.[15]

Im Rahmen von Erweiterungen des Versuchsaufbaus wird versucht die Qubitübertragung mediengebunden über eine Glasfaserleitung zu realisieren. Zudem soll die Übertragungsfrequenz der Einzelphotonen von bisher 1 MHz auf 8 MHz gesteigert werden. Die einzelnen Qubitzustände werden dann nicht mehr über die Polarisation der Photonen, sondern anhand deren Phasenwinkel unterschieden.

1.4 Systemanforderungen

Im Projektrahmen dieser Masterarbeit soll ein System entwickelt werden, das eine Implementierung des BB84 Protokolls zum bestehenden Versuchsaufbau realisiert. Dies umfasst die Steuerung der einzelnen Hardware-Komponenten und die Nachverarbeitung

der gewonnenen Daten für die Schlüsselerzeugung gemäß BB84. Sende- und Empfangseinheit sollen hinsichtlich der Versuchssteuerung und im Postprocessing eigenständig und voneinander getrennt sein. Schnittstellen mit dem Versuchsaufbau bestehen mit der Triggerung der Einzelphotonenquelle, der Steuerung der elektrooptischen Modulatoren und dem Auslesen der Photodetektoren. Die Kommunikation zwischen Sender und Empfänger für das Postprocessing soll über eine Standard-Netzwerkverbindung realisiert werden. Da sich der physische Versuchsaufbau verändern kann, muss das System hinsichtlich zeitlicher Abläufe und Art der Modulatorsteuerung konfigurierbar sein. Die Systemkonfiguration soll dabei über eine grafische Benutzeroberfläche erfolgen. Es wird eine Photonübertragungsfrequenz von 8MHz angestrebt. Neben der Bereitstellung des eigentlichen Kryptographieschlüssels sollen dem Anwender auch Statusinformationen zum aktuellen Arbeitsschritt sichtbar gemacht werden.

2 Stand der Technik

2.1 Einzelheiten zum Versuchsaufbau

2.1.1 Einzelphotonenquelle

Photonen können auf verschiedene Weisen entstehen. Beispielsweise werden sie freigesetzt, wenn Elektronen innerhalb eines Atoms von einem höheren auf niedrigeres Energieniveau übergehen. Dies geschieht entweder zufällig, man spricht von spontaner Emission, oder indem es von außen angeregt wird. Bei der Anregung geht ein Elektron, durch Zuführung von elektrischer oder Lichtenergie, auf ein höheres Energieniveau über. Nach einer gewissen Zeit begibt sich das Elektron auf das Ausgangsniveau zurück und emittiert ein Photon. Sogenannte Zweiniveausysteme, in denen sich ein Elektron in nur einem von zwei möglichen Zuständen befinden kann, können nie mehr als ein Lichtteilchen zur selben Zeit aussenden. Daher werden sie als Einzelphotonenquellen bezeichnet. Praktisch realisiert wird dies im Versuchsaufbau mit Hilfe von Farbzentren in Diamanten. Diese Farbzentren entstehen durch Verunreinigung mit Fremdatomen im Diamanten und verursachen deren Farbwirkung. Bei den im Versuch Verwendung findenden Farbzentren befindet sich neben einem Stickstoff-Fremdatom eine Lücke in der kristallinen Anordnung der Kohlenstoffatome im Diamanten. Regt man diese durch einen Laserimpuls geeigneter Frequenz an, emittiert sie nach einer bestimmten Zeit im angeregten Zustand ein einzelnes Photon. In Abb. 2.1 sind ein Zweiergenniveausystem und ein Stickstoff-Farbzentrums im Diamantgitter dargestellt.

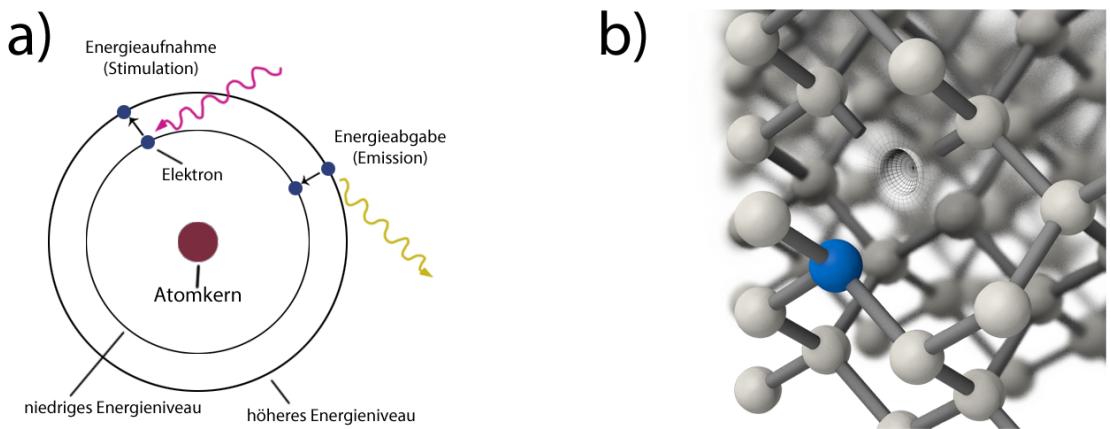


Abbildung 2.1: Physikalischer Hintergrund zur Einzelphotonenquelle. a) modellhafte Darstellung der Freisetzung eines Photons[34] b) modellhafte Darstellung eines Farbzentrums im Kohlenstoffgitter eines Diamanten.[16]

Vorteilhaft an dieser Methode im Vergleich zu anderen ist die Langlebigkeit der Quelle und, dass sie bei Zimmertemperatur funktioniert und nicht gekühlt werden muss. Eine solche Einzelphotonenquelle wurde am Physikalischen Institut der Humboldt Universität im Rahmen einer Doktorarbeit entwickelt.[5] Ihr Aufbau ist für Einzelphotonen mit einer Wellenlänge von 600 bis 800 nm ausgelegt. Die Proben mit den Defektzentren sowie der dazugehörige Erregungslaser können einfach ausgetauscht werden. Der Probenhalter, auf dem sich die Diamantstrukturen befinden, ist auf einer Piezobühne angebracht und kann in allen drei Dimensionen bewegt werden. Hochauflösende Sensoren ermöglichen bei der Fokussierung des Erregungslasers auf die Defektzentren im Diamanten Positionsbestimmungen im Nanometerbereich. Durch einfaches Hinzufügen oder Entfernen eines Spiegels können die emittierten Photonen in Glasfaserleitungen eingekoppelt werden oder im Freiraum übertragen werden. Mit Hilfe eines dichroischen Strahlteilers und zusätzlichen Filtern kann gestreutes Anregungslicht des Lasers entfernt werden. Der schematische Aufbau ist in Abb. 2.2 dargestellt.

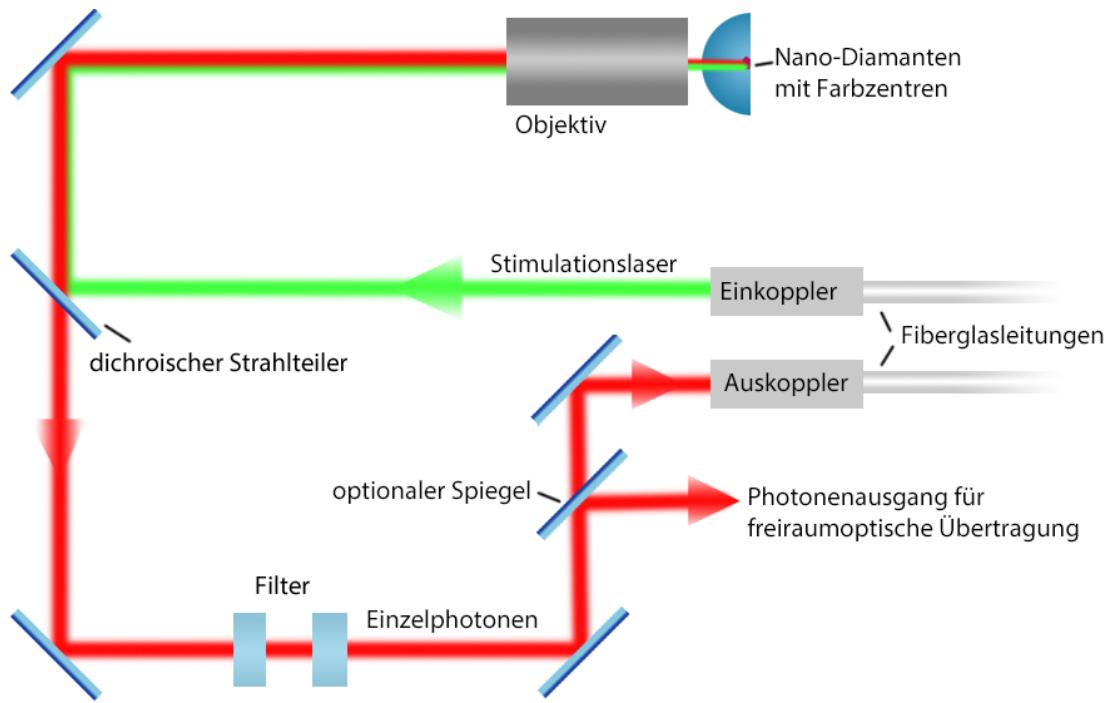


Abbildung 2.2: Schematischer Aufbau der Einzelphotonenquelle. Der Stimulationslaser wird von der Fiberglasleitung in das System eingekoppelt und durch das Objektiv auf das Farbzentrums einer Diamantenprobe fokussiert. Um Streulichteneinfluss vom Stimulationslaser zu verhindern, durchlaufen die Einzelphotonen einen dichroischen Strahlteiler und diverse Filter. Mit einem optionalen Spiegel können die Photonen zu einem Ausgang zur Freiraumübertragung oder zu einem Auskoppler für die Fiberglas-basierte Übertragung geleitet werden.[34]

2.1.2 Elektrooptischer Modulator

Elektrooptische Modulatoren können die Ausbreitungseigenschaften des Lichts verändern. Im Versuchsaufbau werden sie verwendet, um die Polarisationszustände der einzelnen Photonen zu modulieren. Die Funktionsweise basiert auf dem elektro-optischen Effekt. Dieser beschreibt die Veränderung des Brechungsindexes eines Materials beim Anlegen eines elektrischen oder magnetischen Feldes. Häufig finden dabei Kristalle Verwendung, aber auch bestimmte Flüssigkeiten weisen ähnliche nutzbare Eigenschaften auf. Die im Versuchsaufbau zum Einsatz kommenden EOMs basieren auf Kristallen aus Kalium-Dideuterium-Phosphat. Sie ermöglichen die Polarisationsmodulation von Licht in Wellenlängen von 400 bis 850nm.[6] Das magnetische Feld wird mit einer den Kristall umschließenden Spule erzeugt, die Spannungen bis zu +250V benötigt.

2.1.3 Avalanche Photodiode

Wenn kleinste Lichtmengen bis hin zu einzelnen Photonen gemessen und nachgewiesen werden sollen, kommen Avalanche Photodioden zum Einsatz. Diese funktionieren ähnlich wie herkömmliche Photodioden, indem sie Licht in elektrische Energie umwandeln. Anders als die üblichen PIN-Photodioden verfügen Avalanche Photodioden über eine zusätzliche hochdotierte Schicht, die beim Eintreffen kleinsten Lichtmengen einen kontrollierten Lawinendurchbruch hervorruft. Dadurch wird der kleinste Photostrom so verstärkt, dass er in einer elektronischen Schaltung genutzt werden kann. Die im Versuchsaufbau verwendeten APD können Lichtpulse und einzelne Photonen in den Wellenlängen von 400 bis 1060nm detektieren.[18] Die zeitliche Auflösung verspricht Zählraten von 10 Millionen Pulsen pro Sekunde.

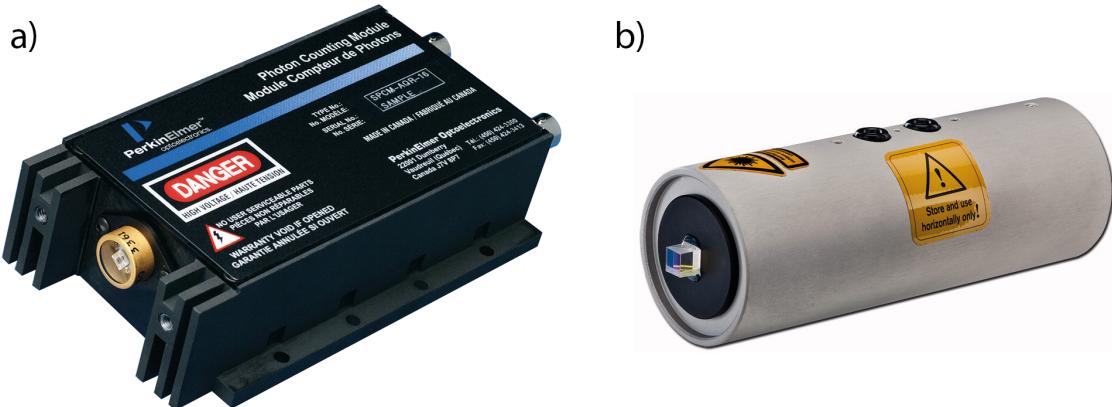


Abbildung 2.3: APD und EOM: a) Avalanche Photodiode (APD)[18] und b) Elektro-optischer Modulator (EOM)[17]

2.1.4 Zufallszahlengenerator QRNG

In vielen Anwendungen, insbesondere der Kryptographie, aber auch in statistischen Verfahren oder dem Glücksspiel sind Zufallszahlen nötig. Die Qualität der Zufallszahlen hängt dabei vom Verfahren ab, mit dem sie generiert wurden. Viele Verfahren basieren auf mathematischen Algorithmen und sind stets in irgendeiner Form nachvollziehbar. Selbst wenn physikalische Messungen in die Berechnung einbezogen werden, so sind diese stets -zumindest theoretisch- nachvollziehbar, da man sie ebenfalls messen könnte. Man spricht daher auch von Pseudozufallszahlen. Ein Zufallszahlengenerator bei dem diese physikalischen Messungen durch Naturgesetze geschützt sind, wurde in der Ar-

beitsgruppe um Prof. Oliver Benson am Institut für Physik der Humboldt Universität Berlin in Zusammenarbeit mit der Firma PicoQuant GmbH entwickelt.[7] Basis für die Bestimmung der Zufallszahlen ist die Zufälligkeit der Ankunftszeiten einzelner, von einer schwachen Lichtquelle emittierter, Photonen. Durch eine ultraschnelle Verarbeitung ist eine Bereitstellung der Zufallszahlen mit Datenraten bis zu 150 Mbit/s möglich.



Abbildung 2.4: Quantenbasiertes Zufallszahlengenerator PQRNG 150[19]

Am Institut sind Zufallszahlen aus diesem Generator frei über einen Webservice beziehbar. Ebenfalls kostenlos zur Verfügung gestellt wird ein Programmierinterface, mit der sich die Zufallszahlenbeschaffung leicht in eine Softwareapplikation einbinden lässt.

2.2 Hardwaredesign

2.2.1 FPGA

Ein Field Programmable Gate Array (FPGA) ist ein programmierbarer Logikbaustein, der nach seiner Herstellung vom Anwender konfiguriert und so mit dem Verhalten einer digitalen Schaltung versehen werden kann. Komplexe Digitalschaltungen können so innerhalb eines Chips implementiert und bei Bedarf wieder verändert werden. Im Wesentlichen bestehen FPGAs aus einer Vielzahl von konfigurierbaren Basiszellen, die matrixförmig auf dem Chip angeordnet sind. Die Basiszellen verschiedener Hersteller unterscheiden sich im Aufbau und werden zudem anders benannt. Beim Marktführer

für Programmierbare Logikbausteine Xilinx werden die Basiszellen als Configurable Logic Block (CLB) und beim größten Konkurrenten Altera als Logic Array Block (LAB) bezeichnet. Ein- und Ausgangsblöcke ermöglichen den Zugriff auf die physischen Anschlüsse des Bausteins. Über ein umfangreiches Leitungsnetz können die einzelnen Logikblöcke miteinander verbunden werden. Spezielle Module auf dem Chip ermöglichen Zusatzfunktionen. So können beispielsweise PLL-Blöcke zur Frequenzanpassung von Taktsignalen, Block-RAM-Einheiten (BRAM) zur Datenspeicherung oder fest verdrahtete Schaltungsmodule zur schnellen Umsetzung mathematischer Funktionen verwendet werden. Das Verhalten der Logikblöcke, ihre Verbindung untereinander, die Einstellungen der Ein- und Ausgänge und die Arbeitsweise der Zusatzmodule sind konfigurierbar. Abb. 2.5 zeigt den Aufbau eines FPGAs in stark vereinfachter Form.

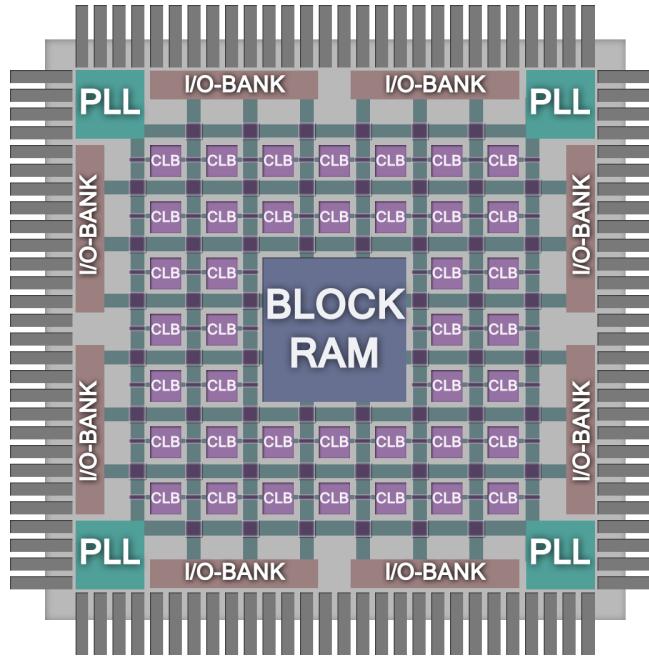


Abbildung 2.5: Vereinfachte Darstellung vom Aufbau eines FPGA. Zu erkennen sind die wesentlichen Bestandteile wie die einzelnen Logikblöcke(CLB), der Block-RAM, die PLL-Module, die I/O-Bänke sowie die Leitungsmatrix zur Verbindung der einzelnen Elemente.[34]

Die Konfigurationsdaten werden auf verschiedene Arten im FPGA gespeichert. Grundlegend wird dabei zwischen drei Technologien unterschieden. Diese sind Speicherung durch SRAM, durch Flash-Speicher und durch sogenannte Antifuses. Die geläufigste Variante ist die der Speicherung durch SRAM. Zu beachten ist hier, dass es sich bei SRAM um einen flüchtigen Speicher handelt. Dies bedeutet, dass er nur bei angelegter Versorgungsspannung die Fähigkeit besitzt, Informationen zu speichern. Somit ist es bei

jeder Inbetriebnahme des FPGAs notwendig, die Konfigurationsdaten in dessen SRAM zu laden. Dies geschieht indem das FPGA die Daten aus einem externen Speicher einliest oder von einem externen Controller beschrieben wird. Einzelne SRAM-basierte FPGA besitzen selbst internen Flash-Speicher, der die Konfigurationsdaten auch nach dem Abschalten der Versorgungsspannung speichert. Bei ausschließlich Flash-basierten FPGAs ist kein Einlesen der Konfigurationsdaten zu jedem Systemstart notwendig. Da Flash-Speicher ein nichtflüchtiger Speicher ist, wird er nur einmal zur Konfiguration beschrieben und behält diese Daten bis zur erneuten Überschreibung. Die Technologie der Antifuses basiert auf der gezielten Schaffung von Leitungsbrücken. Dabei wird durch Anlegen einer erhöhten Programmierspannung die Isolationsschicht zwischen zwei Kontakten irreversibel entfernt (durchgebrannt). Da diese Isolationsschicht nach der Entfernung nicht wiederhergestellt werden kann, ist diese Art von FPGAs nur einmal konfigurierbar. Besonderer Vorteil der Antifuse-Technologie ist ihre Unempfindlichkeit gegen Störeinflüsse wie Höhen- und Röntgenstrahlung. Sie kommen daher vermehrt im Satellitenbau und in der Luft- und Raumfahrt zum Einsatz. Abb. 2.6 zeigt jeweils einen Beispielbaustein zu jeder der drei Speichertechnologien.



Abbildung 2.6: Beispielbausteine verschiedener Speichertechnologien[20][21][22]

Der Aufbau der konfigurierbaren Logikblöcke ist von Hersteller zu Hersteller und von Bausteinfamily zu Bausteinfamily eines Herstellers unterschiedlich. Im einfachsten Fall bestehen sie aus einem Funktionsgenerator, der die kombinatorische Schaltungsfunktion realisiert, und einem Flip-Flop zur Zustandsspeicherung. Die Funktionsgeneratoren funktionieren nach verschiedenen Grundprinzipien. Grundsätzlich wird zwischen Multiplexer-basierten (MUX) und Look-Up-Table-basierten (LUT) Generatoren unterschieden. Die Konfiguration der MUX-basierten Logikzellen erfolgt ausschließlich über die programmierbare Verdrahtung der Multiplexer in der Logikzelle. Die Realisierung erfolgt dabei zumeist mit Flash- oder Antifuse Speichertechnologie. Als Hersteller sind

hier Quicklogic und Actel zu nennen. In SRAM-basierten FPGAs kommen zur Implementierung der Gatterfunktionen Look Up Tables zum Einsatz. Vertreter dieses Aufbauprinzips sind unter Anderem FPGAs der Marktführer Xilinx und Altera. Mit der Zeit wurden die Basiszellen weiterentwickelt und sind weitaus komplexer geworden. Heute enthalten sie mehrere Funktionsgeneratoren und mehrere Flip Flops. Zudem verfügen sie über Zusatzlogik, mit der sie sich zur Bildung komplexer Logikstrukturen zusammenschalten lassen. In Abb. 2.7 ist der Aufbau eines CLB des XC2000 FPGA von Xilinx dargestellt.

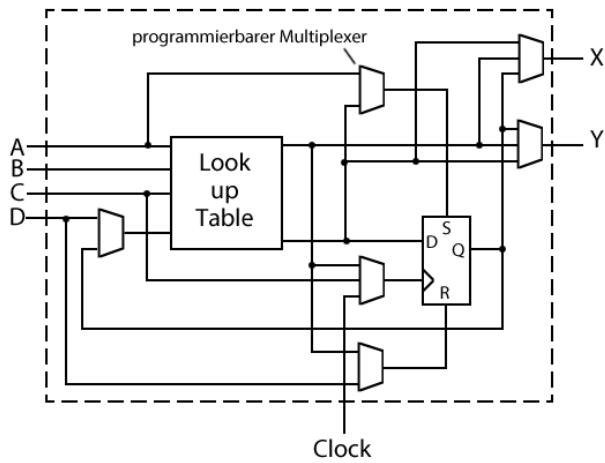


Abbildung 2.7: Konfigurierbarer Logikblock des XC2000 FPGA von Xilinx. Als Funktionsgenerator findet hier eine Look-Up-Table(LUT) Verwendung. Ein Flip-Flop dient zur Zustandsspeicherung[23]

Die Input-/Output-Blöcke (I/O-Blöcke) bilden die Verbindung zwischen der internen Logik des Chips und seinen physischen Anschlüssen, den Pins. Sie ermöglichen die Signalanpassung an eine Vielzahl von Pegel-Standards wie beispielsweise CMOS, TTL, LVTTL und LVDS. Die Blöcke besitzen Tri-State-Treiber und können so als Ein- oder Ausgang konfiguriert werden. Zudem enthalten sie einen Überspannungsschutz gegen hohe Ableitspannungen und bieten die Möglichkeit, den Pin intern mit einem Pull-Up oder Pull-Down-Widerstand zu beschalten. In manchen FPGAs können die Eingänge mit einer internen programmierbaren Signalverzögerung versehen werden. Mehrere I/O-Blöcke werden zu Bänken zusammengefasst. Jede dieser Bänke kann wahlweise Ein- und Ausgänge enthalten, jedoch nur mit einem Spannungspiegel arbeiten.[8] In Abb. 2.8 ist das Schaltbild eines I/O-Block von einem XC4000 FPGA von Xilinx dargestellt.

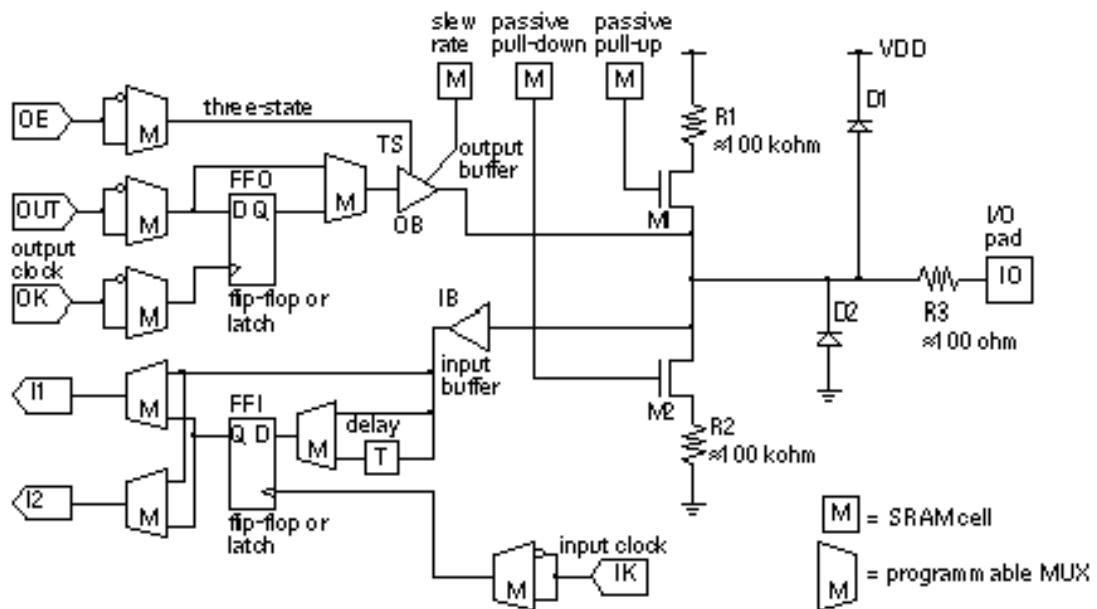


Abbildung 2.8: Schaltbild eines I/O-Blocks des XC4000-FPGA von Xilinx. Zu erkennen sind ein Überspannungsschutz durch die Dioden D1 und D2, die zuschaltbaren Pull-Up- und Pull-Down-Widerstände, die Flip-Flops FF0 und FF1 für die Ein- und Ausgangssignale sowie das eingangsseitige Modul zur Signalverzögerung.[24]

2.2.2 VHDL

Die ständig wachsenden Anforderungen des Marktes und stetig steigende Packungsdichten verfügbarer Logikbausteine lassen elektronische Schaltungen immer komplexer werden. Erfolgt deren Entwurf auf unterster Logikebene, werden sie schnell unüberschaubar und für andere als den eigentlichen Entwickler kaum nachvollziehbar. Zudem bedarf es eines sehr hohen Rechenaufwandes, die Schaltung auf niederer Logikebene zu simulieren. Motivation war es daher, eine Beschreibungssprache zu entwickeln, die den Entwurf, die Dokumentation und die Simulation komplexer elektronischer Schaltungen vereinfacht. Neben diesen Vereinfachungen setzte man sich zudem das Ziel, einen einheitlichen Sprachstandard für alle Entwickler und Hersteller elektronischer Logikbausteine zu etablieren. So begannen Anfang der 80er Jahre, ausgehend vom amerikanischen Verteidigungsministerium, die entsprechenden Entwicklungen, deren Ergebnisse im August 1985 als VHDL veröffentlicht wurden. VHDL steht für Very High Speed Integrated Circuit Hardware Description Language. 1987 wurde die an die Program-

miersprache Ada angelehnte Beschreibungssprache von der IEEE erstmals standardisiert. Die Entwicklung ist dank VHDL nun in strukturierter Form, durch die textuelle Zusammenschaltung einzelner Logikelemente aus Bibliotheken, und zudem auf höherer Abstraktionsebene in Verhaltensbeschreibung möglich. Die Übersetzung auf niedrigere Abstraktionsebenen erfolgt weitestgehend automatisiert durch ein Syntheseprogramm. VHDL als universeller Standard macht die Sprache unabhängig von speziellen Bauteileigenschaften einzelner Hersteller. Diese müssen selbst dafür Sorge tragen, dass die Sprachkonstrukte von VHDL richtig in die vom Bauteil gegebenen Strukturelemente übertragen werden. Dank VHDL kann die Schaltung auf hohem Abstraktionslevel mit geringerem Rechenaufwand simuliert werden. Zudem wird die Wiederverwendbarkeit einzelner Schaltungsmodule erleichtert.[9]

Auch wenn die Hardware-Beschreibungssprache(HDL) in ihrem Stil einer Programmiersprache ähnelt, ist doch das zugrundeliegende Prinzip ein anderes. Wo beim Softwareentwurf mit Programmiersprachen stets Anweisungen für einen Prozessor definiert werden, die von diesem nacheinander abarbeitet werden, wird bei der Verwendung von HDLs, die Beschaffenheit einer elektronischen Schaltung definiert. Die Signalverarbeitung in diesen Modulen erfolgt grundlegend nebenläufig. Eine sequenzielle Abarbeitung von VHDL-Statements kann mit Hilfe von Prozessen realisiert werden.

2.2.3 Entwicklungsumgebung

Bei der Erstellung eines Schaltungsdesigns für komplexe programmierbare Logikbausteine sind eine Vielzahl von Arbeitsschritten notwendig. Die ISE Design Suite (Integrated Software Environment) ist die Entwicklungsumgebung von Xilinx, die eine große Anzahl von Tools für die Bearbeitung der einzelnen Arbeitsschritte vereint. Verfügbar ist die Software in verschiedenen lizenzierbaren Versionen: Logic-, Embedded-, DSP- und System-Edition. Diese unterscheiden sich in der Verfügbarkeit einzelner Programmtools sowie der Unterstützung einzelner Bauteile und Bauteifamilien. Frei verfügbar ist die ISE als Webpack Edition. Dabei werden nur die einfachsten Bauteile unterstützt und die wichtigsten Tools sind verfügbar. Die Beschreibung des Entwicklungsprozesses mit der Xilinx ISE erfolgt ausführlich in Kapitel 2.2.4. Die grafische Programmoberfläche ist in Abb. 2.9 dargestellt.

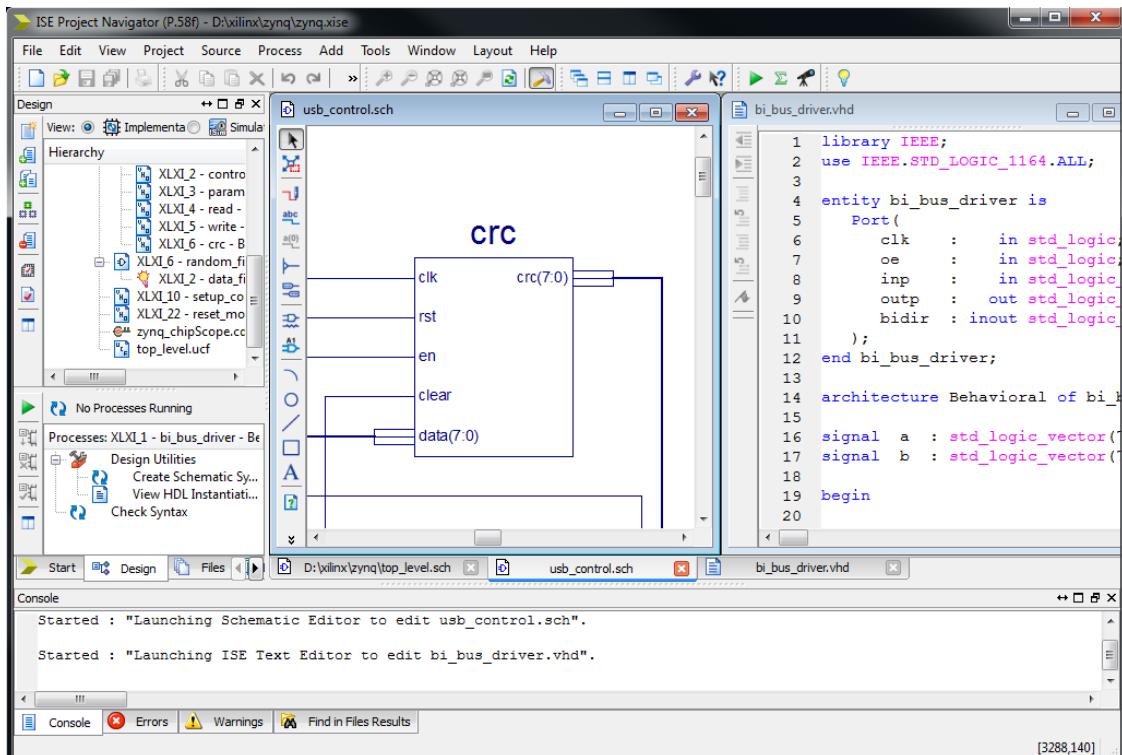


Abbildung 2.9: Programmoberfläche Xilinx ISE Design Suite[34]

2.2.4 Entwicklungsablauf

Eingabe des Schaltungsdesigns

Der Entwicklungsablauf mit der ISE Design Suite beginnt mit dem Entwurf und der Eingabe des Schaltungsdesigns. Dieses kann in Form von Schaltplänen mit Hilfe eines Schaltplaneditors und vorgefertigten Logikelementen oder mittels Hardwarebeschreibungssprachen (HDL) in Textform erstellt werden. Unterstützt werden die Sprachen VHDL und Verilog. Auch die Projekterstellung in Mischformen aus Schaltplänen und Hardwarebeschreibungssprachen ist möglich. Bereits bestehende Hardwarekomponenten in Form von IPs (Intellectual Property) können in das Projekt eingebunden werden. Eine Vielzahl solcher IP-Blöcke sind in Bibliotheken der Entwicklungsumgebung enthalten oder können mit diversen Tools erstellt und konfiguriert werden. Diese Funktionskomponenten reichen von einfachen Logikgattern bis hin zu komplexen Mikroprozessoren und DSP-Einheiten und sind in ihrer Architektur an den jeweils verwendeten programmierbaren Logikbaustein angepasst. Abb. 2.10 zeigt eine mit dem Schaltplane-

ditor entwickelte Beispielschaltung.

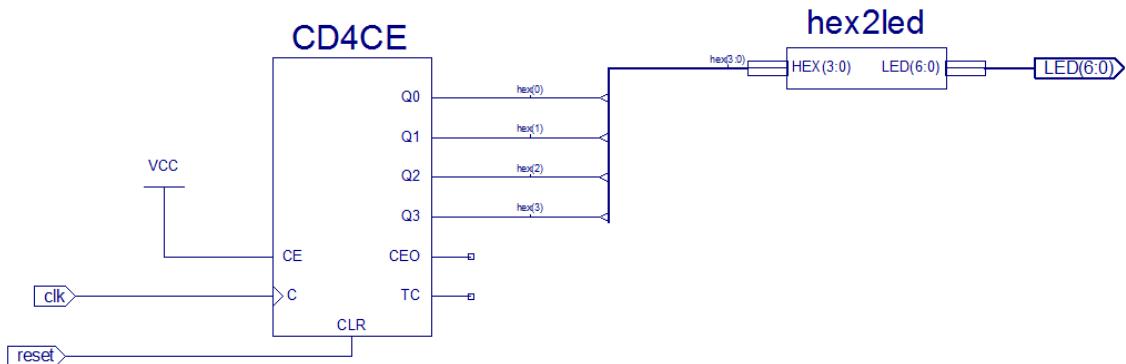


Abbildung 2.10: Beispielschaltung mit Xilinx Schaltplaneditor[34]

Simulation

Die Simulation ist wesentlicher Bestandteil des Entwicklungsprozesses. Bei ihr werden die Funktionalität und das zeitliche Verhalten der entworfenen Schaltung überprüft. Der Simulator interpretiert die Schaltungsbeschreibungen anhand des HDL-Codes und stellt das Schaltungsverhalten durch den Zeitverlauf ausgewählter Signale dar. Da ausschließlich der HDL-Code Basis für die Simulation ist und das Syntheseergebnis nicht in die Simulation einfließt, kann nur das angedachte, nicht aber das tatsächliche Verhalten der Schaltung überprüft werden. Vorteilhaft ist dabei die Zeittersparnis, da für eine Schaltungssimulation nicht die zeitaufwendige Synthese durchlaufen werden muss. Um das Verhalten eines Moduls im Zusammenspiel mit anderen Komponenten beziehungsweise mit dem Gesamtdesign zu testen, ist es oft erforderlich, das Verhalten des umgebenden Systems zu beschreiben. Dieses umgebende System wird als Testbench bezeichnet und wird in der Regel auch mit einer HDL beschrieben. Da der Beschreibungscode der Testbench nicht synthetisiert werden muss, können sogenannte nicht synthetisierbare Befehle verwendet werden, um das Zeitverhalten der Testbench zu beschreiben. Als Simulationstool steht ISim® als direkter Bestandteil der Entwicklungsumgebung zur Verfügung. Auch entsprechende Programme anderer Anbieter können verwendet werden. Als die am weitesten verbreitete Simulationsumgebung ist hier ModelSim von Mentor Graphics zu nennen. In Abb. 2.11 ist eine Beispielsimulation mit dem ISim-Tool dargestellt.

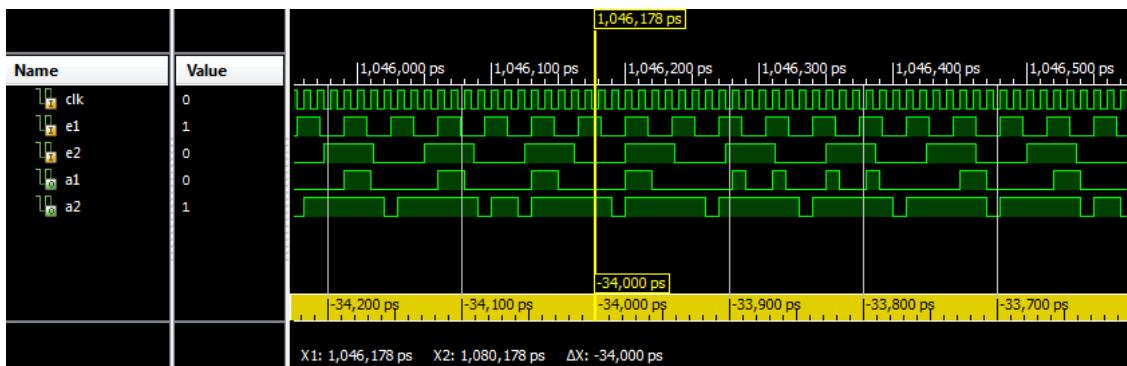


Abbildung 2.11: Ergebnis einer Schaltungssimulation mit ISim[34]

Synthese

Beim Vorgang der Synthese werden im Anschluss die bestehenden Schaltungsbeschreibungen in eine, ebenso speziell an den Baustein angepasste, Netzliste in Form einer NGC-Datei überführt. Zu Beginn dieses Vorgangs werden beim sogenannten HDL-Parsing die bestehenden VHDL- oder Verilog-Daten auf korrekte Syntax geprüft. Anschließend werden die Codesegmente in einzelne Makroblöcke überführt, deren Funktionen durch die auf dem Chip befindlichen Schaltungselemente umgesetzt werden können. Der Synthesizer prüft dabei auch eine mögliche gemeinsame Ressourcennutzung für verschiedene Makroblöcke. Entscheidend für das Syntheseergebnis ist zum einen der HDL-Code zum anderen sind es definierte Rahmenbedingungen, die in Form von Constraints der Syntheseeinheit übergeben werden. Die Vorgabe der Rahmenbedingung hat Einfluss auf die beim Synthesevorgang ablaufende Designoptimierung. Optimierungen können bezüglich des Platzverbrauchs, des Stromverbrauchs oder definierten Timingvorgaben durchgeführt werden. Ergebnis des Synthesierungsvorgangs sind zwei Netzlisten und eine Log-Datei. Die Netzlisten stellen das entworfene Schaltungsdesign vor und nach der Optimierung dar. Dabei zeigt die NGR-Netzliste die Schaltung auf Register-Transfer-Ebene vor der Optimierung an. Diese kann mit dem RTL-Viewer dargestellt werden. Die NGC-Netzliste stellt die durch den Synthesizer optimierte Schaltung aus bausteinspezifischen Schaltungskomponenten dar. Sie kann mit Hilfe des Technology-Viewer angezeigt werden. Die Log-Datei enthält Meldungen des Synthesizers, die bei der Schaltungsübersetzung und Optimierung entstanden sind. Sie liefert außerdem nützliche Hinweise, wie die Schaltung weiter optimiert werden kann.

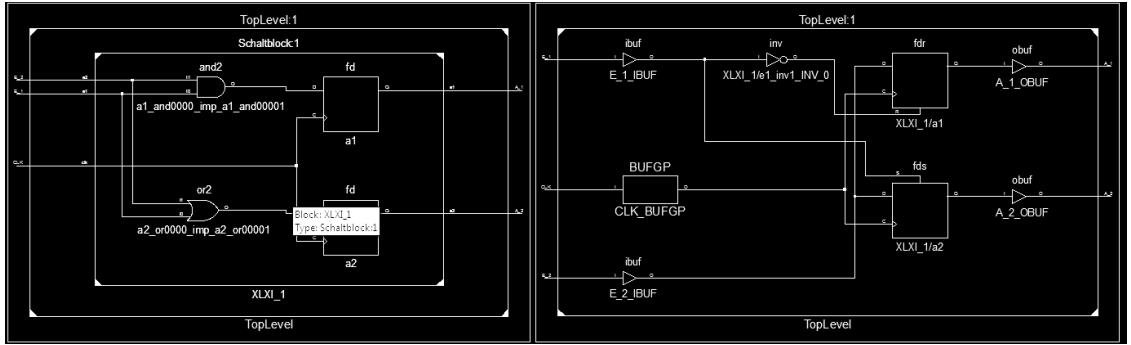


Abbildung 2.12: Darstellung des entwickelten Schaltungsdesigns. Synthesesegebnis RTL-Viewer (links) und Technology-Viewer (rechts)[34]

Definition von Constraints

Unter Constraints sind Festlegungen beziehungsweise Vorgaben für die Designumsetzung zu verstehen. Mit der Definition von Constraints kann das Schaltungsdesign genauer an die bestehenden Systemanforderungen angepasst werden. Die am häufigsten verwendeten Constraints lassen sich in Timing Constraints, Placement Constraints und Synthesis Constraints einteilen. Timing Constraints sind Vorgaben, die zeitliche Eigenschaften von Signalen, wie beispielsweise Taktfrequenzen, Tastverhältnisse oder Eingangs- und Ausgangsoffsets, beschreiben. Sie können sowohl global für das Gesamt-design als auch lokal für einzelne Signalwege definiert werden. Nach weiteren Schritten im Entwicklungsablauf können anhand dieser Vorgaben Aussagen getroffen werden, ob das entstandene Design den Timing Constraints-Vorgaben gerecht wird. Mit Placement Constraints werden Festlegungen über die räumliche Anordnung von Logikelementen getroffen. Dies beinhaltet beispielsweise die Zuordnung von logischen Ports zu physischen Pins oder die genaue Anordnung von Schaltungsmakros auf dem physischen Chip, wie dies bei der Floorplanung geschieht. Schaltungselemente können zudem zu Gruppen zusammengefasst werden, denen bestimmte Eigenschaften zugewiesen werden können. Synthesis Constraints geben der Syntheseeinheit (XST) Anweisungen, nach welchen Vorgaben die Synthese ablaufen soll beziehungsweise was dabei beachtet werden soll. Diverse Tools ermöglichen die anwenderfreundliche Eingabe der Constraints durch grafische Benutzeroberflächen. Da Constraints in Textform gespeichert werden, können sie auch mit dem Texteditor erstellt werden. Alle Constraints werden in sogenannten Constraints-Files gespeichert. Zu unterscheiden sind User Constraints Files (UCF) und XST Constraints Files (XCF). Diese unterscheiden sich im Zeitpunkt ihrer Anwendung.

XCF kommen schon bei der Synthese zum Einsatz. UCF-Daten werden erst später bei der Implementierung verwendet.

Implementierung

Die Implementierung umfasst vier Schritte: Translate, Map, Place and Route und Generating Programming File. Der Arbeitsschritt Translate erfolgt mit dem NGDBuild-Tool. Es bringt sämtliche im Projekt enthaltenen Netzlisten zusammen und erstellt unter Beachtung der definierten Constraints eine Native Generic Database (NGD-File). Diese beschreibt das logische Gesamtdesign, auf Basis von bauteilspezifischen Grundschaltungselementen. Im Arbeitsschritt Map wird die zuvor erstellte Datenbank auf die im FPGA existierenden Elemente wie CLB (Configurable Logic Block) und IOB (Input Output Block) übertragen. Diese übertragene Schaltung wird im Arbeitsschritt Place and Route entsprechend den Vorgaben der Constraints logisch auf dem Baustein verteilt. Einzelne Elemente werden miteinander verbunden. Im letzten Arbeitsschritt Generating Programming File wird der sogenannte Bitstream erzeugt. Diese Daten werden in den programmierbaren Logikbaustein geladen und dienen dessen realer Konfiguration.

Analyse der Implementierungsergebnisse

Mit einer Vielzahl von Methoden kann das Ergebnis der Implementierung analysiert werden. Informationen zum Ablauf der Implementierungsschritte sind den Reportmeldungen zu entnehmen. Diese enthalten Informationen zu einzelnen Arbeitsschritten der Implementierung, Fehlermeldungen, Warnungen, Hinweise und Tipps, mit denen bessere Ergebnisse erzielt werden können. Auf all diese Meldungen kann über die Design Summary zugegriffen werden. Diese stellt neben den Reportmeldungen auch Informationen zum aktuellen Projektstatus dar. Dazu gehören unter anderem aktuelle Projekteinstellungen, die gewählte Designstrategie und Informationen über die Bauteilausnutzung. Mit Hilfe der Timing Constraints kann mit der Timing Analyse geprüft werden, ob das Schaltungsdesign den Geschwindigkeitsanforderungen gerecht wird. Anhand von Verzögerungszeiten in Logikelementen einzelner Signalpfade kann für jede definierte Taktleitung die Maximalfrequenz errechnet werden, mit der die Schaltung fehlerfrei arbeitet. Ist die errechnete Maximalfrequenz geringer als die der gegebenen

Systemanforderungen, sollte das Design überarbeitet und optimiert werden. Die Power Analysis gibt anhand des Bauteiltyps, den definierten Constraints und dem Schaltungsdesign Auskunft über den Leistungsverbrauch des Bauteils. Diese Angaben lassen sich bis auf einzelne Schaltungselemente vertiefen. Mit Nutzung der genannten Analysethoden lassen sich mögliche Schwachstellen im Schaltungsdesign ausfindig machen. Die bestehende Implementierung kann nun gezielt optimiert werden.

Konfiguration des Zielbausteins

Sind Synthese und Implementierung fehlerfrei abgeschlossen, können die erstellten Konfigurationsdaten in den Zielbaustein geladen werden. Verwendung findet dabei das Tool iMPACT.

In-System Debugging

Das In-System Debugging ermöglicht es, das Systemverhalten einzelner Signalfäde auf dem Chip, auf Basis eines internen Logikanalysators zu überprüfen. Das Tool ChipScope Pro findet dabei Anwendung. Um entsprechende Funktionen eines solchen Analysators auf dem Chip zu realisieren, wird ein ChipScope-Core in das bestehende Design implementiert. Je nach Art der Anwendung stehen verschiedene Versionen dieses Moduls zur Verfügung. Vor der Implementierung wird der Analyzer-Core konfiguriert. Dabei werden Signale ausgewählt, die analysiert werden sollen und Trigger definiert, an denen die Aufnahme gestartet wird. Tritt die Triggerbedingung auf dem Chip ein, werden die angegebenen Signale in das Block-RAM (BRAM) geschrieben. Über das Programmierinterface lassen sich die so gespeicherten Daten auslesen und auf der grafischen Oberfläche des ChipScope Pro Tools darstellen.

2.3 Softwaredesign

2.3.1 C++ und Qt

Die Programmiersprache C++ wurde 1979 von Bjarne Stroustrup entwickelt. Als Basis dazu diente die Programmiersprache C, die im Sinne der Objektorientierung erweitert wurde. Der Funktionsumfang von C bleibt als Teilmenge erhalten. Der Haupteinfluss bei der Entwicklung von C++ war durch die Programmiersprachen SIMULA67 und ALGOL68 gegeben, von denen unter Anderem das Klassenkonzept und das Prinzip der Operatorenüberladung stammt. Ziel bei der Entwicklung war es, eine schnelle Sprache zu entwickeln, die die objektorientierte Programmierweise unterstützt. Durch die Erweiterung von C sollte ein fließender Übergang zur objektorientierten Programmierung geschaffen werden, ohne dass sich die Programmierer, von denen bereits viele mit C vertraut waren, in eine komplett neue Sprachsyntax einarbeiten mussten. Zuerst wurde die Sprache 'C mit Klassen' genannt. Ihr Name wurde später in C++ geändert, der durch den Inkrementieroperator ++ die Erweiterung beziehungsweise die Weiterentwicklung von C symbolisiert. Erst im Jahr 1998 wurde die Sprache von der ISO genormt. Inzwischen hat sich C++ fest etabliert und zählt zu den am meistverwendeten Programmiersprachen überhaupt. Aus diesem Grund sind im Laufe der Zeit zahlreiche Programmzbibliotheken für C++ entstanden.[10] Eine umfangreiche Bibliothekensammlung für C++ ist Qt. Ursprünglich wurde Qt entwickelt, um grafische Benutzeroberflächen plattformübergreifend erstellen zu können. Ziel war es dabei, dass ein Programm auf einem Betriebssystem programmiert und für jedes unterstützte Betriebssystem kompiliert werden kann. Unterstützung finden dabei Betriebssysteme wie Windows, Mac OS, Linux, Solaris und embedded Linuxsysteme. Mittlerweile ist das Bibliothekenframework um weitere Bibliotheksmodule erweitert worden. Somit unterstützt Qt beispielsweise Netzwerkanwendungen, die Integration von Datenbanken, OpenGL-Anwendungen und eine Vielzahl anderer Anwendungen. Die Ideen zu Qt stammten von Haavard Nord und Eirik Chambee-Eng, die 1991 mit den ersten Entwicklungen zu Qt begannen. Im Mai 1995 stand das Qt Framework erstmals zur Verfügung. Mittlerweile sind verschiedene andere Programmierer an der Weiterentwicklung von Qt beteiligt. Die Firma, unter der Qt entwickelt und vertrieben werden sollte trug den Namen Quasar Technologies. Daher stammt auch der Name von Qt, der sich von Quasar Toolkit ableitet. Später wurde die Firma in Trolltech umbenannt. Das Qt-Framework steht unter den Lizenzen für Freie Software GPL und LGPL sowie einer kommerziellen Lizenz für die Entwicklung proprie-

tärer Software zur Verfügung. Großer Beliebtheit erfreut sich das für Qt-Anwendungen entwickelte Signal/Slot-Konzept, welches die Kommunikation zwischen einzelnen Objekten ermöglicht. Es wurde auch in andere Anwendungen übertragen.[11]

2.3.2 Qt-Creator

Der Qt-Creator ist die Entwicklungsumgebung für die C++ Programmierung unter Verwendung der Qt Bibliotheken. Die Software ist für die geläufigsten Betriebssysteme Windows, Mac OS und Linux erhältlich. Enthalten ist ein Texteditor mit Syntaxhervorhebung und Autovervollständigung, eine umfangreiche Klassendokumentation und eine übersichtliche Projektverwaltung. Für die Gestaltung von Programmoberflächen bietet der Qt-Creator einen grafischen GUI-Designer. Dieser ermöglicht die schnelle Erzeugung einer grafischen Benutzeroberfläche indem einzelne Container- und Bedienelemente per Drag and Drop zusammengefügt und konfiguriert werden können. Ein Debugger erleichtert die Fehlersuche im Programm. Die Programmoberfläche des Qt-Creator ist in Abb. 2.13 dargestellt.

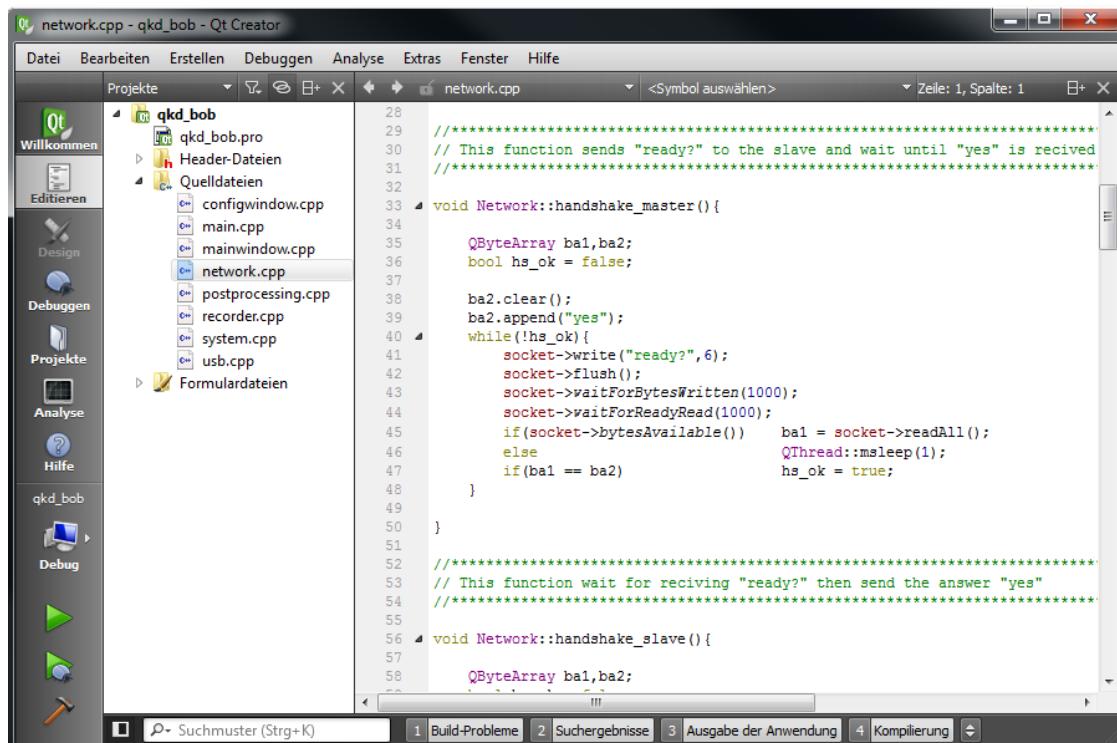


Abbildung 2.13: Programmoberfläche Qt Creator[34]

3 Systemrealisierung

3.1 Systementwurf

Grob lässt sich das zu erstellende Steuersystem in eine Sende- und eine Empfangseinheit unterteilen. Jedes dieser Teilsysteme benötigt eine Vorrichtung, um die einzelnen Gerätschaften des Versuchsaufbau zu steuern, und ein zentrales Rechnersystem zur Koordination der Gesamtanwendung. Die Vorrichtung zur Versuchsaufbausteuerung muss jeweils über die erforderlichen Schnittstellen verfügen und diese im gewünschten Zeitverhalten ansprechen können. Zudem ist eine Kommunikation mit dem zentralen Rechnersystem notwendig, um Steuerbefehle und anfallende Daten zeitnah auszutauschen. Die Aufgabe des zentralen Rechnersystems ist es, eine Kommunikation zwischen Sende- und Empfangseinheit zu realisieren, Zufallszahlen für die zufällige Modulation und Demodulation zu beschaffen, die Vorrichtung für den Versuchsaufbau zu steuern und einen Datenaustausch mit ihr zu betreiben. Weiterhin soll es die Nachverarbeitung der Daten, das Postprocessing, realisieren. Auf mindestens einer Seite, ob Sender oder Empfänger, muss eine Konfiguration des Systems möglich sein.

Am einfachsten lässt sich das zentrale Rechnersystem mit jeweils einem PC realisieren. Diese verfügen über die vielseitigsten Schnittstellen, die über das vorhandene Betriebssystem für eine Anwendung meist unkompliziert anzusprechen sind. So lässt sich einfach eine Sender-/ Empfängerkommunikation mit Hilfe vom TCP-Sockets realisieren. Die Konfiguration des Systems kann über eine grafische Benutzeroberfläche, mit Maus, Tastatur und Bildschirm erfolgen. Schneller Speicher ist in ausreichendem Maße vorhanden. Zudem verfügen PCs über Schnittstellen zu ausreichend schnellen Bussystemen, mit denen eine Datenverbindung zur Gerätesteuervorrichtung möglich wäre. Die Alternative wäre es, die zentrale Instanz auf einem eingebetteten Prozessorsystem zu realisieren. Diese verfügen zumeist auch über die erforderlichen Schnittstellen, die mit Hilfe eines speziellen Betriebssystems einfacher anzusprechen wären. Die Konfiguration

des Zielsystems könnte mit einer Benutzeroberfläche auf einem Bildschirm oder Grafikdisplay möglich sein, welche mit Maus, Tastatur oder Touchpad bedient wird. Der Entwicklungsaufwand und somit auch die Entwicklungszeit wären allerdings erheblich größer. Ein weiterer Nachteil für die Systemrealisierung auf einem eingebetteten System gegenüber einem PC zeigt ich beim nichtflüchtigen Speicher. Eingebettete Systeme verfügen dabei meist über Flashspeicher auf Speicherkarten. Dieser ist langsamer und meist nicht in so großem Maße vorhanden, wie dies bei Festplatten im PC der Fall ist. Auch für die Steuerung des Versuchsaufbaus wäre der Einsatz eines schnellen Prozessorsystems prinzipiell denkbar. Allerdings bietet hier die programmierbare Logik eines FPGA durch seine Parallelität in der Systemrealisierung erhebliche Vorteile.

Um dabei nicht die komplette Elektronik der Steuervorrichtung selbst zu entwickeln, wird auf verfügbare Entwicklungsboards zurückgegriffen. Diese stellen ein in sich funktionierendes elektronisches System dar und bieten mit zahlreichen Anschlussmöglichkeiten die Anpassung an die gewünschte Anwendung. Bei der Frage nach der Realisierung des Datenaustauschs zwischen PC und FPGA-Board muss zunächst bestimmt werden, wie viele Daten in einer bestimmten Zeiteinheit übertragen werden müssen. Die größte Datenmenge fällt auf der Seite des Empfängers an. Pro einzelner Photonentransmission muss ein Zufallsbit für die Modulatorsteuerung vom PC zum FPGA und zwei Bits entsprechend der Ausschläge an den beiden Photonendetektoren vom FPGA zum PC übertragen werden.

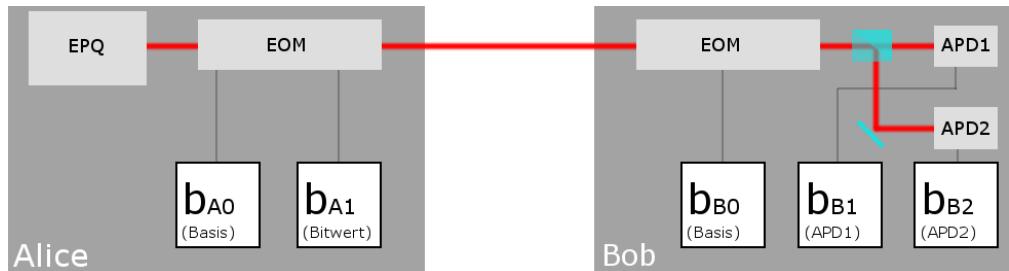


Abbildung 3.1: Übersicht der aufkommenden Daten. Bei Alice werden pro einzelner Photonentransmission zwei Zufallsbits (b_{A0} und b_{A1}) für die Modulatorsteuerung benötigt. Bob hingegen benötigt dazu nur ein Zufallsbit (b_{B0}). Bei jeder Transmission entstehen bei Bob zwei Bits (b_{B1} und b_{B2}), die jeweils den Ausschlag eines Photonendetektors signalisieren.[34]

Die maximale Transmissionsfrequenz wurde in den Systemanforderungen mit 8 MHz angegeben. Demzufolge müssen in einer Sekunde 8 Millionen Bits zum FPGA und 16 Millionen Bits zum PC gesendet werden. Nach der Addition der beiden Werte und der Umrechnung in eine geeignete Maßeinheit, ergibt sich eine benötigte Mindestda-

tenrate von 2,87 MB/s. Da bei jeder Übertragung systembedingt mit Verzögerungen in der Datenübertragung zu rechnen ist und eventuell während der Datenübertragung noch Steuerbefehle gesendet werden müssen, sollte die tatsächliche Kapazität der Datenübertragung um ein Vielfaches höher sein als diese errechnete Mindestdatenrate.

Eine Vielzahl von FPGA-Evaluationsboards verfügen über einen USB/seriell Wandler. Damit können Daten vom FPGA im Protokollrahmen einer seriellen Schnittstelle übertragen werden. Am PC wird diese Verbindung über einen virtuellen COM Port angeprochen. Die physische Datenübertragung findet über USB statt. Diese Verbindungen lassen sich einfach realisieren. Die mögliche Datenrate ist allerdings selten größer als 1 MB/s. Somit sind diese Verbindungen zwar für die Übertragung von Steuerbefehlen und kleinsten Datenmengen geeignet, für die geforderte Anwendung mit der Übertragung größerer Datenmengen ist die Übertragungsrate jedoch zu gering.

Weiterhin denkbar wäre eine Datenübertragung über den Computerbus PCIe. Eine solche Verbindung wäre bereits im einfachsten Fall -mit einer Lane in Version 1- bei Übertragungsraten von 250MB/s fast schon überdimensioniert. Für derartige PCIe-Anwendungen sind spezielle FPGA-Boards für den Einbau in den PC ausgelegt. Hersteller wie Xilinx bieten für ihre PCIe-fähigen Boards kostenlose IP-Cores, mit denen sich der Buszugriff erleichtern lässt. Nachteilig an dieser Variante ist zum Einen die enge mechanische Bindung an den PC. Zum Anderen ist es mit recht hohem Aufwand verbunden, einen funktionsfähigen Treiber für PCIe am PC zu erstellen. Die sehr hohen Taktraten des Bussystems im GHz-Bereich machen zudem die Signalanalyse bei der Fehlersuche und Fehlerbehebung schwierig.

Variante drei ist der Datenaustausch über eine Netzwerkverbindung. So könnte ein schneller Datenaustausch über Ethernet oder höhere Netzwerkprotokolle erfolgen. Diese bieten ausreichend hohe Datenraten. Allerdings wäre eine Netzwerkkommunikation innerhalb des Sender- oder Empfangssystems ein potenzieller Angriffspunkt für mögliche Abhörer. Es müsste zusätzlich Aufwand betrieben werden, um die Kommunikation autorisiert durchzuführen. Bei der Verwendung einer Direktverbindung zwischen PC und FPGA Board, wäre der oft einzige Netzwerkanschluss des PCs für andere Aufgaben, wie die Sender- / Empfängerkommunikation blockiert.

Letzte und realistischste Möglichkeit der schnellen Datenübertragung ist die über USB. Da das Protokollhandling von USB recht umfangreich auf einem FPGA zu implementieren ist und entsprechende IPs für die Einzelanwendung recht kostspielig sind, bietet

sich die Verwendung eines schnellen USB-Wandlermoduls an. Diese Module übernehmen das USB-Protokollhandling und stellen für den Datenaustausch zwischen Wandlermodul und elektronischem Zielsystem eine Reihe von einfacheren Datenaustauschprotokollen zur Verfügung. Ein solches Modul wird von der Firma FTDI angeboten. In seiner schnellsten Variante des Datenaustauschprotokolls, dem seriellen synchronen FIFO, sind Datenraten von bis zu 40 MB/s angegeben. Dies ist für die Anwendung ausreichend. Das Modul wird in Abschnitt 3.2.5 näher beschrieben.

Um die Photonen richtig messen und die einzelnen Transmissionen später fehlerfrei zuordnen zu können, ist es entscheidend, dass sowohl der Sender als auch der Empfänger in einem festen zeitlichen Raster arbeiten. Oszillatoren bieten zwar hinsichtlich ihrer Frequenz hohe Güten, jedoch führen schon kleinste Frequenzabweichungen auf Dauer zu einem Auseinanderdriften der Taktsignale und so zu einer Verschiebung des zeitlichen Rasters bei Sender und Empfänger. Aus diesem Grund ist es notwendig eine unabhängige Synchronisation zu schaffen, um die Stabilität des zeitlichen Rasters zu gewährleisten.

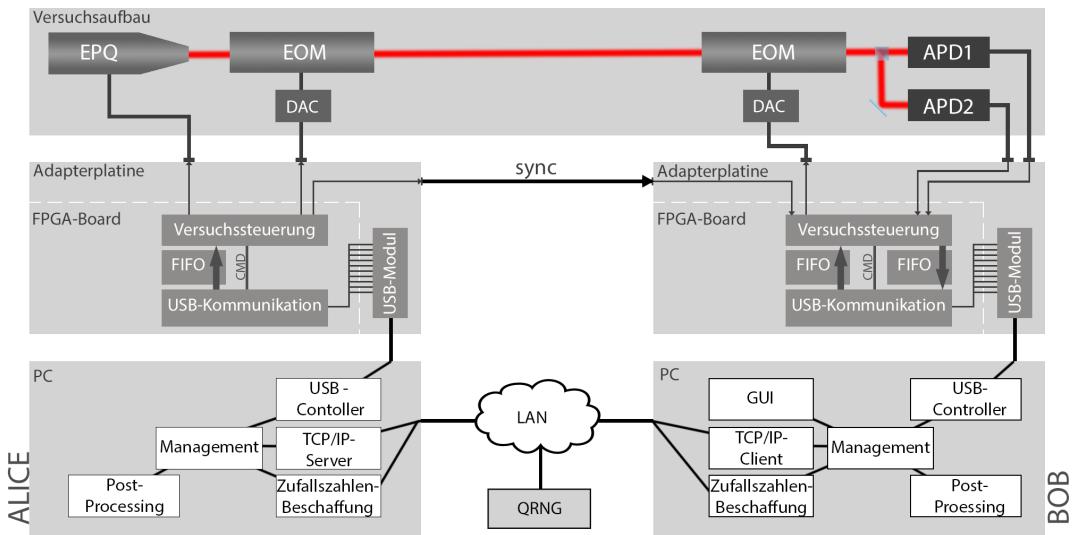


Abbildung 3.2: Schematischer Gesamtaufbau. Die Komponenten des Versuchsaufbaus werden von zwei eigenständigen Systemen gesteuert. Das Sendesystem(Alice) und das Empfangs-System(Bob) bestehen jeweils aus einem PC und einem FPGA-Board. Die Kommunikation zwischen den Systemen erfolgt über eine Netzwerkverbindung(LAN). Ebenfalls über das LAN werden die Zufallsdaten vom Zufallszahlengenerator(QRNG) bezogen. Die Anschlussmöglichkeiten der FPGA-Boards, für die Versuchsaufbausteuerung und die Kommunikation zum PC, wird über die Adapterplatten realisiert.[34]

3.2 Elektronische Realisierung

3.2.1 FPGA Evaluationboard SP 605

Evaluationsboards bieten dem Entwickler ein stabiles und getestetes elektronisches Ausgangssystem für die Entwicklung einer bestimmten Anwendung oder eines Funktionsprototypen. Je nach Art des Boards können ein oder mehrere Kernbausteine, wie FPGAs, Mikrocontroller oder DSPs, darauf enthalten sein. Zudem verfügen Evaluationboards zumeist über eine Vielzahl von zusätzlichen Bausteinen und universellen Anschlussmöglichkeiten, die eine Anpassung des Systems an die gewünschte Anwendung ermöglichen. Somit ersparen sie dem Entwickler viel Zeit und Arbeit, die für die Entwicklung eines solchen Ausgangssystems nötig wäre.

Das verwendete Xilinx Evaluationboard SP605 basiert auf einem Spartan 6 FPGA und bietet eine Vielzahl von Erweiterungsmöglichkeiten und Zusatzfunktionen. Bezogen werden kann es direkt beim Hersteller Xilinx oder bei verschiedenen Vertriebspartnern. In seiner einfachsten Form als Evaluation Kit ist es für 495\$ erhältlich. Das gleiche Board ist mit Programmerweiterungen für die Unterstützung des Softcore-Prozessors Microblaze als Embedded Kit für 695\$ und zudem mit umfangreicher IP-Bibliothek für die Peripherieprotokolle als Connectivity Kit für 1995\$ erhältlich. (Preisinformation Mai 2013). Im Projekt findet es als Evaluation Kit Verwendung.

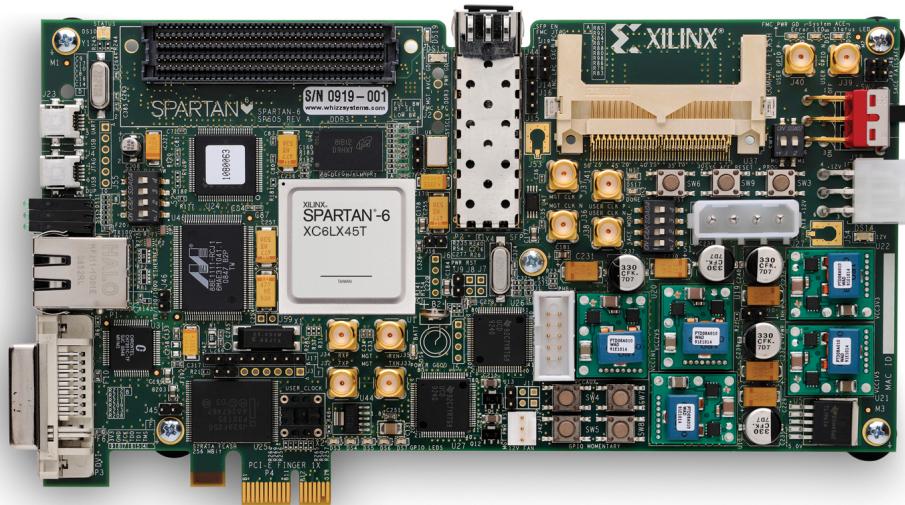


Abbildung 3.3: Xilinx SP 605[25]

Versorgt wird das Board über eine 12V Spannungsversorgung. Eine umfangreiche Schaltung zur Spannungsumsetzung auf dem Board generiert daraus eine Vielzahl von Spannungspegeln, die auf dem Board Verwendung finden. Diese sind 1.2V, 1.5V, 1.8V, 2.5V, 3V, 3.3V, 5V und 10V. Für die Kommunikation zwischen PC und FPGA-Board während der FPGA-Konfiguration und der Systemanalyse mit dem Logikanalysator ChipScope findet JTAG Verwendung. Ein auf dem Board befindlicher Konverter realisiert dessen Umsetzung zu USB. Ein weiterer Umsetzer von USB zu UART ermöglicht die serielle Kommunikation mit Übertragungsraten bis zu 921600 Baud zwischen PC und FPGA für universelle Anwendungen. Für Netzwerkanwendungen verfügt das Board über einen Netzwerkanschluss und einen PHY*-Baustein, welcher für Ethernetkommunikation mit Übertragungsraten von 10, 100 und 1000 Mbit/s verwendet werden kann. Für Grafikanwendungen ist das Board mit einem DVI-Anschluss ausgestattet. Der dazugehörige Treiberchip realisiert Auflösungen von maximal 1600 x 1200 Bildpunkten bei 24 bit Farbauflösung. Die Maße und Anschlussverbindungen des Boards sind so ausgelegt, dass es auch innerhalb eines Computers zum Einsatz kommen kann. Zudem ermöglicht die PCIe Gen1 Schnittstelle eine Anschlussmöglichkeit an das Bussystem des Rechners. Für speicherintensive Anwendungen stehen 128MB DDR3 und 32MB Flash Speicher zur Verfügung. Ein entsprechender Anschluss ermöglicht die Verwendung einer CompactFlash Speicherkarte (CF), auf der bis zu acht verschiedene Datenabbilder für die FPGA-Konfiguration abgelegt werden können. Für die Taktversorgung stehen zwei Oszillatoren mit 200MHz und 27MHz zur Verfügung. Externe Takteinspeisung kann über SMA-Buchsen realisiert werden. Zudem verfügt das Board über einen Quarzsockel, in den ohne Lötaufwand ein Quarz mit der gewünschten Frequenz eingesteckt werden kann. Ein SFP (small form-factor pluggable) bietet die Anschlussmöglichkeit an extrem schnelle Netzwerke. Eine Anschlussmöglichkeit für Erweiterungsboards bietet der FMC (FPGA Mezzanine Card) Connector. Auf diesen sind Leitungen der Spannungsversorgung, Taktleitungen und I/O-Leitungen als Differential Pair oder Einzelleitungen abgreifbar. Neben all diesen Peripheriemodulen sind auch Einheiten für die einfache Ein- und Ausgabe wie LEDs, Taster und Schalterleisten vorhanden. Nachteilig am SP605 ist, dass die Bänke des FPGA maximal mit 2.5V versorgt werden. Somit entfällt eine Vielzahl an Pegelstandards bei der Konfiguration der I/O-Blöcke, für die eine Spannungsversorgung von 3.3V notwendig wäre.

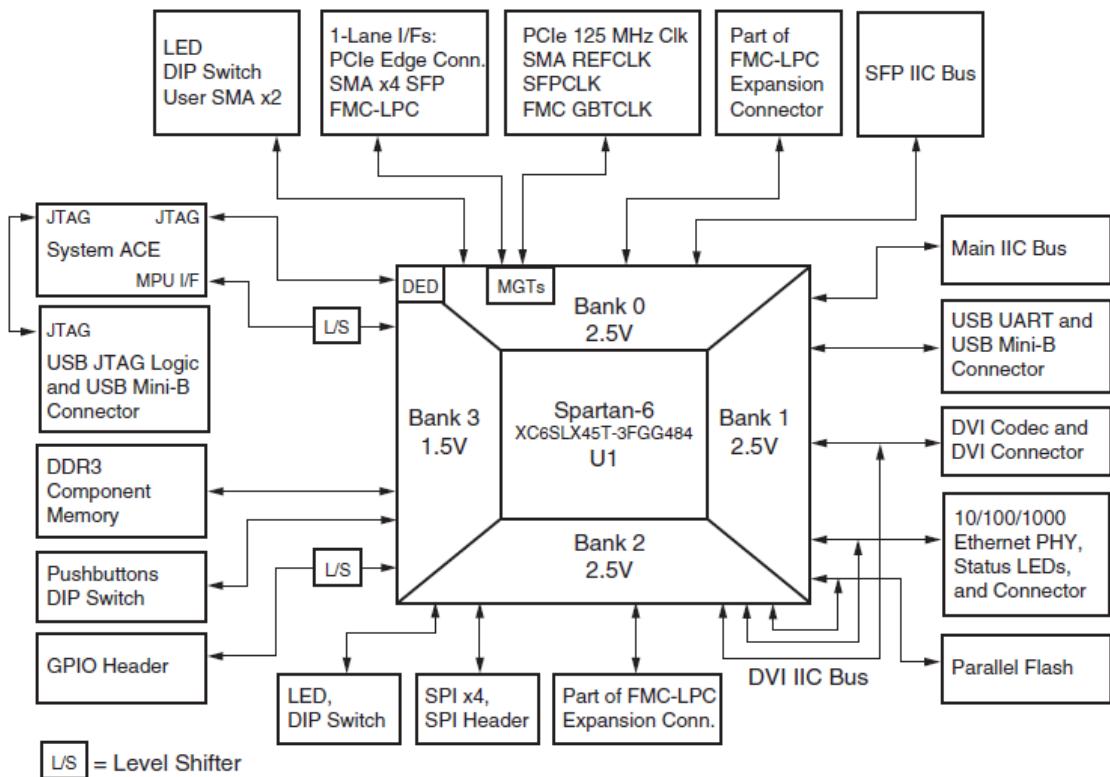


Abbildung 3.4: Blockschaltbild SP605. Zu erkennen sind das Spartan-6-FPGA in der Mitte, umgeben von den auf dem Board vorhandenen Schnittstellen und Peripheriekomponenten. Die vier Bänke des Spartan-6 sind fest mit 1,5V beziehungsweise 2.5V verbunden.[26]

3.2.2 Spartan 6

Spartan ist die Low-Level Variante der FPGAs von Xilinx. Auf dem SP605 kommt die Version XC6SLX45T zum Einsatz. Sie ist eine von insgesamt 13 Bauteilvarianten, die in der Spartan 6 Familie enthalten sind. Der Baustein verfügt über 43661 konfigurierbare Logikzellen. Als Funktionsgeneratoren in den SRAM-basierten Logikzellen kommen Look up Tables mit sechs Eingängen zum Einsatz. Zusätzliche 2088 Kb Block-RAM steht für die Datenspeicherung innerhalb einer Anwendung zur Verfügung. Der Anwender kann auf 296 I/O-Pins verteilt auf 4 Bänken zugreifen. Besonders an der T - Variante des Bausteins ist das Vorhandensein von High-Speed-Transceivern. Die ermöglichen eine serielle Datenübertragung mit bis zu 3.2 Gb/s. Insgesamt vier dieser Transceivereinheiten sind auf dem Chip enthalten. Sie werden beispielsweise für PCIe - und 1G Ethernetanwendungen benötigt. Für verschiedene DDR Speichervarianten sind

festverdrahtete Speichercontroller vorhanden, die das Ansprechen der externen Speicherchips vereinfacht.

3.2.3 Zedboard

Besonders am ZedBoard ist der Zynq-Kernbaustein. Dieser enthält, ähnlich wie ein FPGA, programmierbare Logik, die vom Anwender für das Erstellen einer digitalen Schaltung genutzt werden kann. Zudem befindet sich ein ARM Prozessor auf dem Chip, der unabhängig oder in Verbindung mit der programmierbaren Logik betrieben werden kann. Mit diesen Voraussetzungen wäre es im Rahmen einer späteren Projekterweiterung denkbar, eine der in Abschnitt 3.1 beschriebenen zentralen Rechnereinheiten auf dem Prozessor des Zynqs zu implementieren. Der Umfang der programmierbaren Logik des Zynq ist für die Anwendung ausreichend. Weiteres ausschlaggebendes Argument für das Zedboard ist der Preis. Dieser ist mit 285,60€(als akademische Version) für die gebotenen Funktionen sehr gering. Weiterhin ausschlaggebend für die Verwendung des ZedBoards ist das Vorhandensein einer FMC-Schnittstelle, die auch auf dem SP605 Verwendung findet. Da für die Ansteuerung der Gerätschaften des Versuchsaufbaus jeweils eine Adapterplatine mit den entsprechenden Schnittstellen für jedes FPGA Board erstellt werden muss, vereinfacht eine einheitliche Schnittstelle deren Entwicklung. Im Vergleich zum SP605 ist das Zedboard mit weniger Zusatzmodulen ausgestattet. Für Speicheranwendungen befinden sich 512MB DDR3 RAM auf dem Board, ein 256 Mb großes SPI Flash Modul ist vorhanden und eine SD-Karte kann angeschlossen werden. Auf ihr könnte das Abbild eines Betriebssystems abgelegt und beim Starten der Anwendung vom ARM Prozessor geladen werden. Eine HDMI-Schnittstelle mit entsprechendem Treiberbaustein ermöglicht, wie schon beim SP605 die Anschlussmöglichkeit für einen Bildschirm. Audiodaten könnten in den HDMI - Datenstrom eingebunden werden. Als analoge Alternative verfügt das ZedBoard über eine VGA-Schnittstelle. Der USB-Anschluss ist in drei Varianten möglich. Mit entsprechenden Umsetzern für JTAG und die serielle Kommunikation oder zum Anschluss universeller USB-Geräte auf der Basis von USB-OTG (on the go). Als Standardmodule für die anwenderseitige Ein- und Ausgabe sind Druckknöpfe, Schalter und LED vorhanden. Ein kleines Grafikdisplay mit einer Auflösung von 128x32 Bildpunkten ermöglicht die Darstellung kleiner einfarbiger Grafiken. Für zusätzliche Funktionen bietet das ZedBoard Anschlussmöglichkeiten für sogenannte PMODS. Dabei handelt es sich um kleine universelle Zusatzmodule die über einen standardisierten Steckverbinder an das Board angeschlossen werden können.

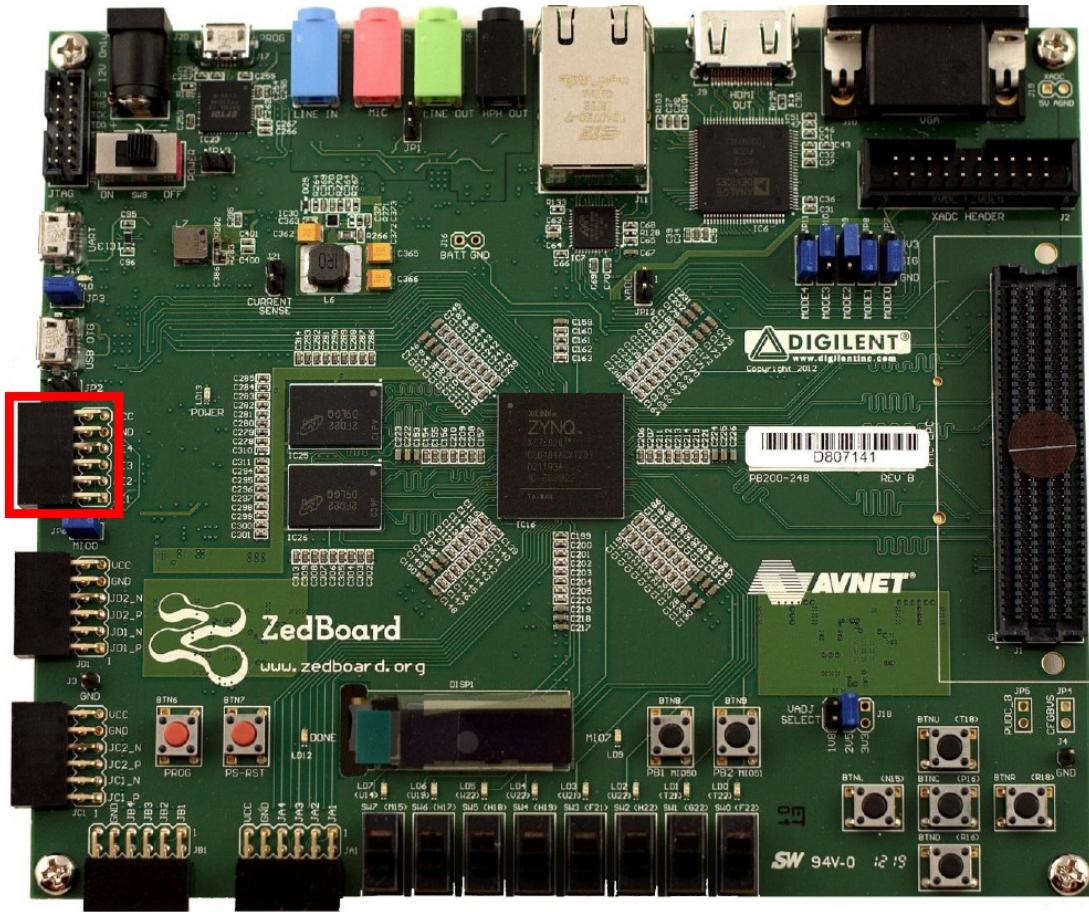


Abbildung 3.5: ZedBoard[27]

3.2.4 Zynq

Beim Zynq handelt es sich um ein sogenanntes System on Chip(SoC), welches von Xilinx vertrieben wird. Wie schon im Abschnitt 3.2.3 beschrieben, befindet sich neben der programmierbaren Logik, ein ARM-Prozessor mit zwei Prozessorkernen auf dem Chip. Die programmierbare Logik des Chips basiert auf der Technologie der Artix 7 FPGA - Familie des gleichen Herstellers. Diese basiert ebenso wie die des Spartan 6 auf SRAM-Technologie und realisiert die Funktionsgeneratoren der konfigurierbaren Logikzellen durch Look up Tables mit sechs Eingängen. Im Vergleich zum Spartan 6 ist hier jedoch die Strukturdichte enger. Dadurch ist zwar die Herstellung aufwendiger und teurer, die Leistungsfähigkeit des Chips hinsichtlich Stromverbrauch, Verarbeitungsgeschwindigkeit und Packungsdichte wird jedoch gesteigert. Auf dem ZedBoard kommt der Baustein in der Variante xc7z020 zum Einsatz. In dieser stehen dem Anwender

85000 Logikzellen für die Umsetzung des Schaltungsdesigns zur Verfügung. Somit bietet er sogar mehr Platz als das verwendete Spartan 6 FPGA. Für die Signalspeicherung stehen 560 kB Block-RAM zur Verfügung. Beim verwendeten Prozessor handelt es sich um einen Dual ARM Cortex A9. Dieser kann mit bis zu 866 MHz betrieben werden. Er bietet Unterstützung für zahlreiche Standardschnittstellen wie UART, CAN, I2C oder SPI. Zudem bietet er direkten Speicherzugriff (DMA) für USB, Gigabit-Ethernet und SD/SDIO. Dies sind ideale Voraussetzungen um ihn mit einem Betriebssystem zu betreiben.

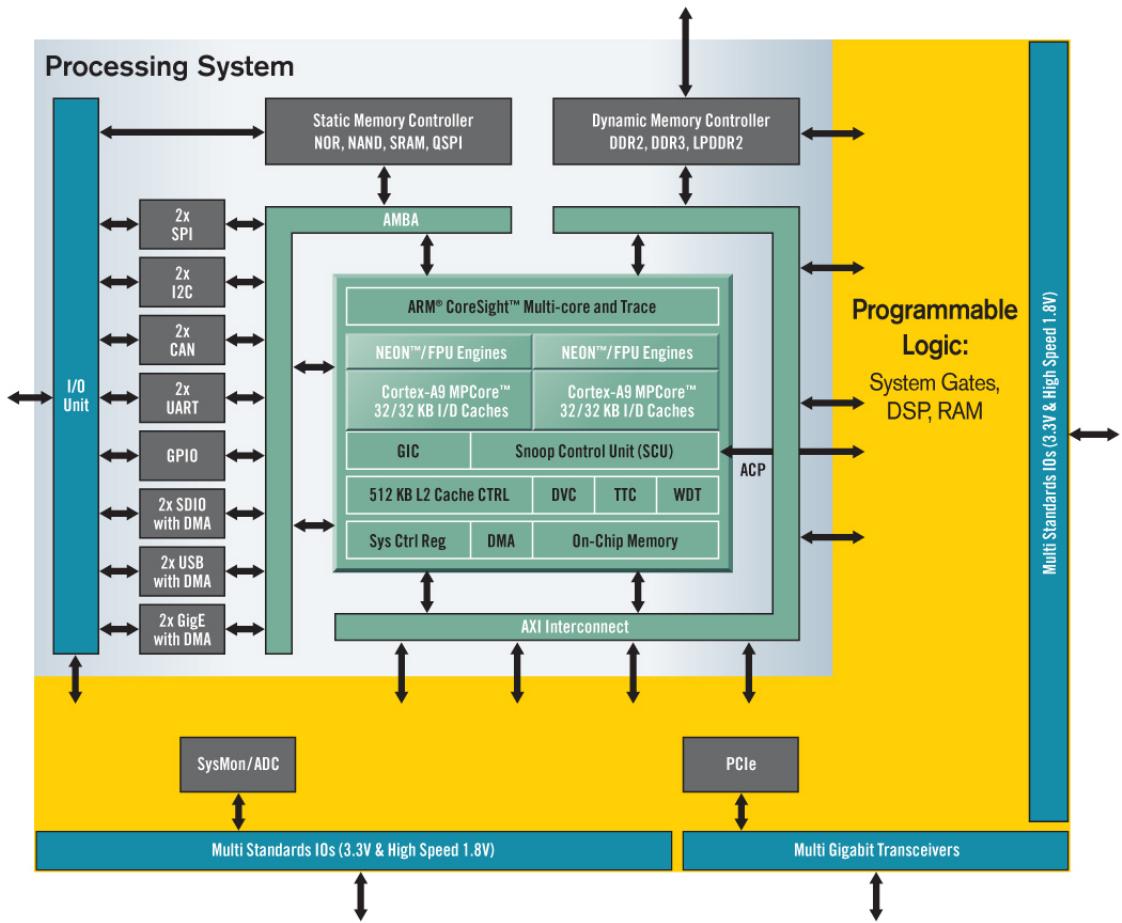


Abbildung 3.6: Schematischer Aufbau des Zynq. Dieser besteht aus einem Dual-Core-ARM-Prozessor und programmierbarer Logik.[28]

3.2.5 UM232H

Das UM232H von FTDI ist ein Kommunikationsmodul für den USB-basierten Datenaustausch zwischen PC und einem externen elektronischen System. Das Protokollhandling der USB 2.0-Verbindung wird rechnerseitig von einem Treiber und auf der Seite des elektronischen Systems vom Modul selbst übernommen. Zum PC-Treiber wird dem Anwender ein Programmierinterface zur Verfügung gestellt, mit dessen Funktionen das Modul zur Konfiguration und zum Datenaustausch vom PC aus angesprochen werden kann. Das Modul selbst bietet eine Reihe von Protokollen an, mit denen die Kommunikation mit dem elektronischen System realisiert werden kann. Dazu gehören serielle Protokolle wie JTAG, SPI und I2C sowie parallele Protokolle wie asynchroner und synchroner FIFO. Im Projekt findet das Modul unter Nutzung des synchronen FIFO-Protokolls Verwendung. Dieses Protokoll bietet mit bis zu 40MB/s die höchste Datenübertragungsrate. Das UM232H gibt dabei einen 60 MHz Takt vor, zu dem die Daten synchron übertragen werden. Neben den 8 bidirektionalen Datenleitungen erfolgt die Flusskontrolle über weitere sechs Leitungen. Eine genauere Beschreibung des Protokolls wird in Abschnitt 3.3.3 gegeben.



Abbildung 3.7: USB-Modul UM232H[29]

3.2.6 Adapterplatine

Um die verschiedenen Gerätschaften des Versuchsaufbaus anzusteuern und Daten von ihnen einzulesen, werden an den FPGA-Boards entsprechende Anschlüsse benötigt. Zudem ist es erforderlich, dass für die Kommunikation zwischen PC und FPGA vorgesehene USB-Wandlermodul in das System zu integrieren. Da die Boards selbst nicht die entsprechenden Anschlussmöglichkeiten besitzen, ist die Erstellung einer Adapterplatine notwendig. Speziell für solche Erweiterungsplatinen verfügen sowohl das SP605 als auch das ZedBoard über eine standardisierte FMC-Schnittstelle. Diese stellt als Low Pin Count (LPC) Variante mit insgesamt 160 Pins ausreichende Anschlussmöglichkeiten bereit. Da im Rahmen dieser 160 Pins auch Masseverbindungen, Leitungen zur Spannungsversorgung, Takteitungen und verschiedene andere Leitungen enthalten sind, bleiben 68 nutzbare Pins als universelle Ein- und Ausgänge. Um den Entwurfs- und Fertigungsaufwand gering zu halten, sind beide Adapterplatten weitestgehend identisch aufgebaut und jeweils mit allen, für den gesamten Versuchsaufbau benötigten Anschlässen versehen. Lediglich kleine Veränderungen wurden bei der Wahl von Ein- und Ausgängen vorgenommen. Dieses universelle Entwurfsprinzip hat zudem den Vorteil, dass es so möglich ist, den kompletten Versuchsaufbau auch mit einem FPGA zu steuern. Als Anschlüsse werden im Einzelnen drei BNC-Buchsen benötigt, zwei als Eingänge für die beiden Photonendetektoren und eine um Anregungslaser für die Einzelphotonenquelle zu triggern. Die optischen Modulatoren werden über externe 10 bit Digital/Analog-Wandler gesteuert. Da neben den zehn Datenleitungen pro Kanaleingang ein Taktsignal erforderlich ist und zwei Modulatoren im Versuchsaufbau Verwendung finden, werden insgesamt 22 Signalleitungen für die Modulatorsteuerung benötigt. Als Anschlussverbindung dient hierbei eine doppelte VHDCI-Buchse.

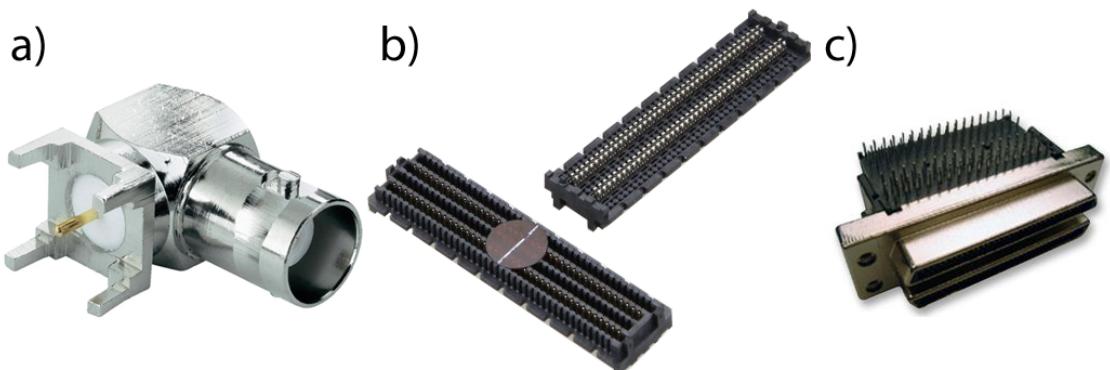


Abbildung 3.8: Platinenanschlüsse: a) BNC-Buchse[30] b) FMC-Connector[31] c) VHDCI-Connector[32]

Das USB-Wandlermodul wird über zwei Steckerleisten an die Platine angeschlossen. Neben der Spannungsversorgung und Masseverbindung sind darüber auch die 60MHz Taktverbindung sowie sämtliche Daten- und Steuerkanäle mit dem FPGA-Board verbunden. Da sämtliche Gerätschaften des Versuchsaufbaus mit einem Signalpegel von 3,3 V arbeiten, das SP605 aber maximal einen Signalspannungspegel von 2,5V ausgeben kann, ist es erforderlich eine Pegelanpassung mit Levelshiftern vorzunehmen. Die Platine wurde zweilagig realisiert. Um lange Signalleitungen und somit unnötig lange Signallaufzeiten zu verhindern, wurde die Platine so kompakt wie möglich gehalten.

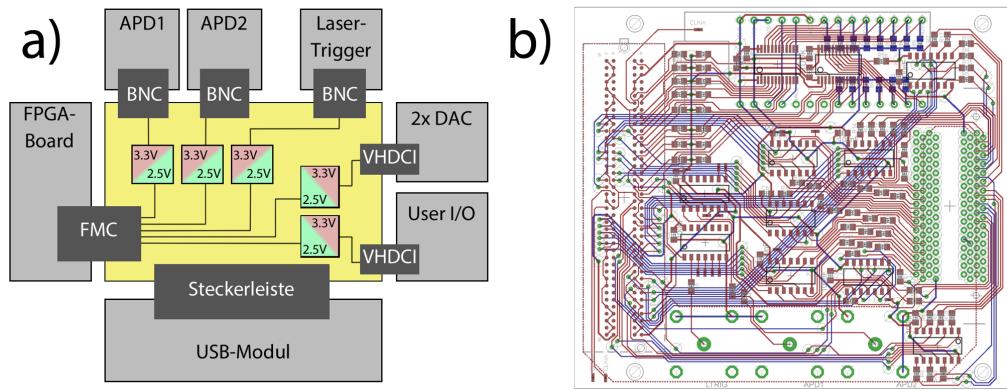


Abbildung 3.9: Adapterplatine: a) schematischer Aufbau[34] b)Platinenlayout[34]

3.3 Systemdesign FPGA

3.3.1 Allgemein

Das Schaltungsdesign lässt sich sowohl für die Sende- als auch für die Empfängerseite in drei Hauptmodule unterteilen. So muss beidseitig die Kommunikation mit dem USB-Modul stattfinden und der Versuchsaufbau gesteuert werden. Das dritte Hauptmodul übernimmt die Zwischenspeicherung anfallender Daten mit FIFO-Elementen. Dies ist erforderlich, da die bei der Versuchssteuerung benötigten oder anfallenden Daten nicht unmittelbar über die USB-Verbindung bezogen oder versendet werden können. Die benötigten Daten entsprechen dabei den Zufallszahlen zur Ansteuerung der elektrooptischen Modulatoren. Die anfallenden Daten entstehen bei der Detektion der einzelnen Photonen mit Hilfe der APDs. Da nur auf der Empfängerseite eine Photonendetektion erfolgt, entfällt senderseitig der Abtransport und die Zwischenspeicherung anfallender Daten.

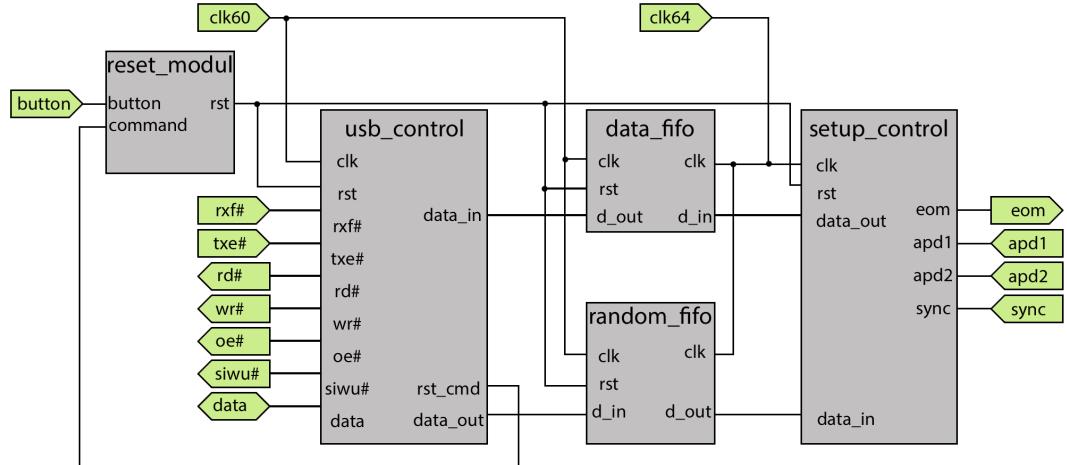


Abbildung 3.10: Top-Level-Design. Vereinfachte schematische Darstellung des Top-Levels auf der Empfängerseite.[34]

Zur besseren Überschaubarkeit der Schaltung wird der Gesamtentwurf auf höchster Hierarchieebene, dem Top Level, in diese Module unterteilt. Siehe dazu Abb. 3.10. Die Unterteilung erfolgt grafisch, indem die Module als Schaltplanelemente dargestellt und durch Signalleitungen miteinander verbunden werden. Die einzelnen Elemente enthalten die eigentliche Schaltungslogik in Form von VHDL-Code oder werden erneut schematisch in Untermodulen unterteilt. Ausschließlich auf dem Top Level erfolgt die Zuordnung einzelner Signale zu den Pins des FPGAs. Innerhalb der Schaltungslogik auf den FPGAs finden zwei verschiedene Taktsignale Verwendung. Der Datenaustausch mit dem USB-Modul erfolgt mit dem 60 MHz Taktsignal, welches vom USB-Modul ausgesandt wird. Dadurch wird die taktsynchrone Datenübermittlung zum USB-Modul sichergestellt. Die Versuchsaufbausteuerung hingegen wird aus Gründen der einfacheren Umsetzung einer Transmissionsfrequenz von 8 MHz mit einem 64MHz-Taktsignal versorgt. Dieses Taktignal wird mit Hilfe der PLLs aus den, auf den FPGA-Boards enthaltenen, Quarzsignalen generiert. Die Anpassung der beiden Frequenzen bei der Datenübertragung zwischen USB-Modullogik und Versuchsaufbausteuerlogik erfolgt durch die FIFO-Elemente. Diese ermöglichen die Verwendung unterschiedlicher Taktfrequenzen für Lese- und Schreibzugriffe. Sämtliche Schaltungslogik ist mit Reseteingängen ausgestattet und kann durch aktiven Resetpegel zurückgesetzt werden. Der Reset selbst kann auf verschiedene Weisen ausgelöst werden. Zum einen über Taster auf dem Board selbst oder über einen entsprechenden Resetbefehl vom PC. Zur Realisierung dieses Resetverhaltens befindet sich ein zusätzliches Schaltungsmodul auf dem Top Level. Dieses reagiert auf eingehende Signalflanken vom Resettaster oder auf einen entsprechenden Befehl vom USB-Modul.

Einmal aktiviert hält es den Resetpegel mit Hilfe einer Zählerschaltung für eine kurze aber definierte Zeit auf High und sorgt so für eine zuverlässige Rücksetzung der gesamten Schaltungslogik.

Bis auf Ausnahmen bei der Umsetzung der Busanbindung zum USB-Modul und der Resetverarbeitung der einzelnen Module wurde die gesamte Schaltungslogik taktsynchron realisiert. Im Vergleich zu asynchronen Schaltungen, bei denen das Systemverhalten direkt von den Verzögerungszeiten der kombinatorischen Logikelemente abhängig ist, kann so ein stabiles und berechenbares Schaltungsverhalten erreicht werden.

3.3.2 Zwischenspeicherung der Daten

Die verwendeten FIFO-Elemente sind als IPs in die Schaltung eingebunden. Mit einer Speicherkapazität von 64kB sind sie möglichst groß gewählt. Somit bleibt im Falle von Verzögerungen der Versuchsaufbausteuerung oder dem USB-Modul ausreichend Zeit, um ein Überlaufen oder ein Leerlaufen des Speichers zu verhindern. Bei der Konfiguration der FIFOs können bestimmte frei wählbare Schwellwertsignale aktiviert werden. Diese signalisieren das Über- beziehungsweise Unterschreiten eines bestimmten Füllwertes. Mit Hilfe dieser Schwellwertsignale können die FIFO-Elemente selbständig ein Auffüllen oder Datenabtransport veranlassen. Der Abtransport der Daten und das Auffüllen der FIFOs erfolgt mit 32kB Datenpaketen. Die Wortbreite, mit dem der FIFO beschrieben und ausgelesen wird, beträgt 8 Bit. Neben den konfigurierbaren Schwellwertsignalen verfügen die FIFO-Module über Full- und Emptysignale, die signalisieren, wenn der Speicher komplett gefüllt oder komplett leer ist. Im laufenden Betrieb werden diese Signale genutzt um Fehler im Füllstandsmanagement zu detektieren.

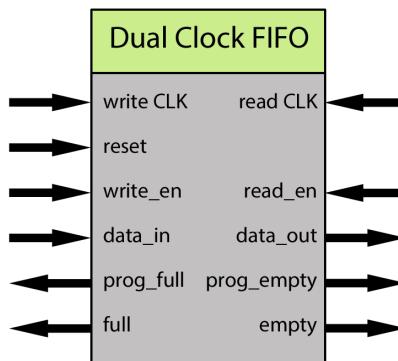


Abbildung 3.11: Schematische Darstellung eines FIFO-Elements[34]

3.3.3 USB-Protokoll

Das Hauptmodul der Protokollsteuerung für den USB-Wandler wird wiederum in folgende Module unterteilt: Ein Bustreiber steuert die Flussrichtung des bidirektionalen Datenports zum USB-Modul, von dem Daten eingelesen oder geschrieben werden. Eine Interfacesteuerung verarbeitet beidseitig einkommende Transmissionsanfragen vom FPGA und vom USB-Modul und entscheidet über deren Priorität bei der Abarbeitung. Für Lese- und Schreibvorgänge gibt es jeweils eigenständige Funktionsmodule. Diese können über ein erstelltes Protokoll Parameter setzen, abfragen und die FIFOs auslesen beziehungsweise auffüllen. In einem Parameterspeicher können einer bestimmten Adresse zugeordnete Parameter abgefragt oder gesetzt werden. Die Unterteilung der Schaltung für die USB-Kommunikation ist in Abb. 3.12 dargestellt.

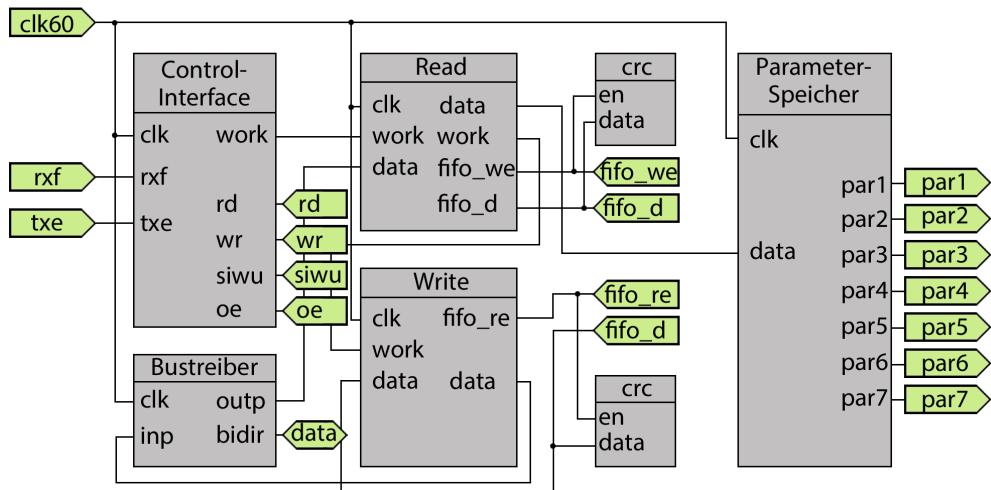


Abbildung 3.12: USB-Modul-Design. Unterteilung der Steuerlogik für das USB-Modul.[34]

Mit einer bidirektionalen Bussteuerung können Lese- und Schreibzugriffe auf einem Datenbus stattfinden. Elektronisch wird dies durch Tristate-Treiber in den I/O-Blöcken der FPGAs realisiert. Diese ermöglichen es, den Datenkanal hochohmig zu schalten. Für Schreibzugriffe wird wie gewöhnlich ein High- oder Lowpegel auf die Datenleitung geschrieben. Für Lesezugriffe ist es erforderlich, den Bus hochohmig zu schalten. Dies geschieht im VHDL-Code durch das Beschreiben des Datenports mit einem 'Z'. Die Hochohmigschaltung des Datenports erfolgt in einem ungetakteten Prozess. Ein weiterer Prozess sorgt für die Ein- und Austaktung eingehender und ausgehender Daten. Ein beispielhafter VHDL-Code für einen solchen Treiber ist im folgenden Codelisting

dargestellt.

Listing 3.1: Schaltungsbeispiel für einen bidirektionalen Bustreiber

```

1
2      process(clk)
3      begin
4          if(rising_edge(clk)) then
5              a <= inp;
6              outp <= b;
7          end if;
8      end process;
9
10     process(oe, bidir,a)
11     begin
12         if( oe = '0') then
13             bidir <= "ZZZZZZZZ";
14         else
15             bidir <= a;
16         end if;
17         b <= bidir;
18     end process;

```

Die Flusssteuerung der Kommunikation mit dem USB-Modul erfolgt mit Hilfe von sechs Steuerleitungen. Die Kanäle RXF# und TXE# ausgehend vom USB-Modul signalisieren mit einem Low-Pegel, dass Daten zum Einlesen bereit liegen beziehungsweise, dass das Modul bereit ist, Daten aufzunehmen. Gehen diese Signale während einer laufenden Datenübertragung auf High, so muss die Datenübertragung unterbrochen werden, bis der entsprechende Signalpegel wieder auf Low geht. Die tatsächliche Datenübertragung wird vom FPGA ausgehend durch einen Low-Pegel auf den Kanälen RD# und WR# signalisiert. Der bidirektionale Bustreiber des USB-Moduls wird, vom FPGA ausgehend, über den OE# Kanal gesteuert. Sollen Daten vom Modul gelesen werden, muss zuvor der OE# Kanal auf Low gesetzt werden. Somit ist der Signalausgang des Datenkanals vom USB-Modul nicht länger hochohmig und die Daten können auf den Datenbus geschrieben werden. Der Sechste der Steuerleitungen ist SIWU#. Ein kurzer Low Puls nach der Datenübertragung zum USB-Modul sorgt für die unverzügliche Übertragung der Daten über die USB-Leitung zum PC. Die Interfacesteuerung übernimmt die Kontrolle und Steuerung dieser Kanäle. Das genaue Timing der Steuerkanäle für Schreib- und Leseoperationen ist in Abb. 3.13 dargestellt.

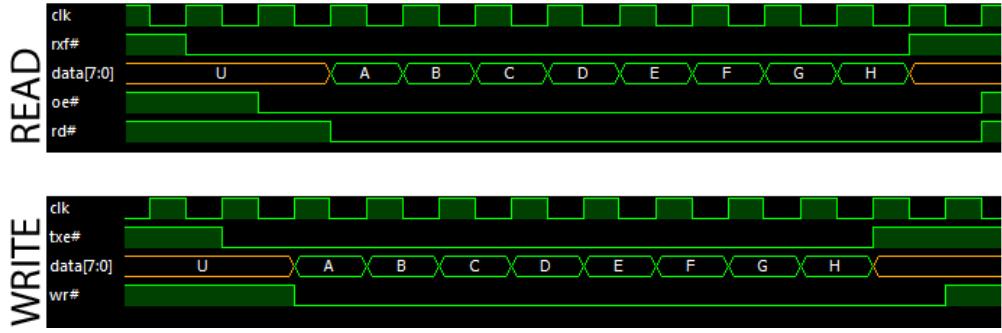


Abbildung 3.13: Lese- und Schreibzugriffe auf das USB-Modul. **READ:** Geht `rxif#` auf low, liegen Daten im USB-Modul zum Auslesen bereit. Nachdem `oe#` auf low gezogen wurde, können durch einen low-Pegel an `rd#` die Daten ins FPGA eingelesen werden. **WRITE:** Das USB-Modul signalisiert mit einem Low-Pegel an `txe#`, dass es zu Einlesen von Daten bereit ist. Mit einem low-Pegel an `wr#` erkennt das USB-Modul, dass Daten eingelesen werden.[34]

Bis auf die bei der Photonendetektion entstehenden Daten, die nach dem Überschreiten des eingestellten Schwellwert im FIFO automatisch zum PC versandt werden, geht sämtliche Kommunikation vom PC aus. Diese Kommunikation beinhaltet das Setzen von Parametern, das Abfragen von Parametern und das Übertragen von Datenpaketen für die Modulatorsteuerung. Um diese drei Varianten unterscheiden zu können, ist ein Protokollrahmen für die Kommunikation über USB eingerichtet. Die Verarbeitung der Daten erfolgt dabei byteweise. Unterschieden werden die einzelnen Befehlsarten anhand des ersten Bytes. Ein '#' - Zeichen gibt an, dass es sich um einen Befehl zum Setzen eines Parameters handelt. Daraufhin folgen vier weitere Bytes. Das erste gibt die Adresse des Parameters an. Die folgenden drei Byte repräsentieren den Parameterwert. Dazu werden die drei Bytes aneinandergereiht. Ein Parameter besteht demzufolge aus 24 Bits und kann somit einen Wertebereich von 0 bis 16777215 abdecken. Wurde der Parameter erfolgreich gesetzt, wird an den PC zur Bestätigung ein '!' - Zeichen gesendet. Beginnt der eingehende Befehl mit einem '?', so handelt es sich um eine Parameterabfrage. Das Zeichen nach diesem Anfragesymbol gibt die Adresse des entsprechenden Parameters an. Dieser wird daraufhin im Anschluss an ein '%' - Zeichen und zerlegt in drei aufeinanderfolgende Bytes an den PC übertragen. Auf eine Bestätigung von Seite des PCs wird hier verzichtet, da dieser bei einer Fehlübertragung erneut eine Parameteranfrage senden kann. Die dritte Art von Befehlen beginnt mit einem '@' - Zeichen und signalisiert die Übertragung eines Datenpaketes vom PC an das FPGA. Dabei wird ein Datenpaket fester Größe erwartet und dieses an den FIFO-Speicher weitergeleitet.

Die unterschiedlichen Befehlsarten sind mit Hilfe von frei gewählten Beispielbefehlen in Tab. 3.1 dargestellt.

Befehlsart	Anfrage vom PC	Antwort vom FPGA
Parameter setzen	#a007	!
Parameter abfragen	?a	%a007
Datentransfer	@abcde...xyz	

Tabelle 3.1: Darstellung der Befehlsarten anhand von Beispielbefehlen

Übersteigt das Fülllevel des Daten-FIFOs einen Wert von 32kB, wird, ausgelöst vom Schwellwertsignal, ein Datenpaket in der festen Größe von 32kB aus dem FIFO ausgelesen und an den PC übertragen. Ist zu diesem Zeitpunkt ein andere Übertragung in Bearbeitung, wird gewartet, bis die laufende Übertragung beendet ist. Bei der Übertragung der aufgenommenen Daten ist es entscheidend, dass der PC die Daten schnell genug aus dem Empfangspuffer ausliest und damit ein Datenverlust durch Überlaufen des Eingangspuffers im USB-Modul verhindert wird. Zur Kontrolle, dass ein Datenpaket fehlerfrei übertragen wurde, erfolgt während des Auslesens aus dem FIFO eine Prüfsummenberechnung, indem die einzelnen Datenworte addiert werden. Auch beim Empfang eines Datenpaketes vom PC erfolgt eine Prüfsummenberechnung. Damit beim Transmissionsvorgang und der dabei auftretenden hohen Anzahl von Datenübertragungen nicht auch noch parallel Parameterabfragen bezüglich der Prüfsummen getätigt werden müssen, werden die Prüfsummen an das zum PC gesendete Datenpaket angehängt. Zudem wird dem Datenpaket ein Byte angehängt, das ein Unterschreiten des Schwellwertes des FIFOs für die Zufallszahlen (Random-FIFO) signalisiert. Wurde dies vom PC erkannt, so erfolgt die Aussendung eines Datenpaketes mit weiteren Zufallszahlen. Eine schematische Darstellung des Datenpaketes mit angehängten Zusatzbytes ist in Abb. 3.14 dargestellt.



Abbildung 3.14: Datenpaket mit angehängten Zusatzinformationen. Ein Datenpaket beinhaltet 32768 Byte Nutzdaten. Angehängt sind ein Byte, welches signalisiert, dass neue Zufallszahlen im FPGA benötigt werden und zwei Prüfsummen. Die erste Prüfsumme ist die des zuletzt empfangenen Datenpakets. Die Zweite gilt für die im aktuellen Datenpaket enthaltenen Nutzdaten.[34]

Da auf dem FPGA der Senderseite keine Daten entstehen, entfällt die Notwendigkeit diese zum PC zu übertragen. Hier ist es ausschließlich notwendig, die für die Photonenmodulation benötigten Zufallszahlen zum FPGA zu übertragen. Die Signalisierung der Füllstandsunterschreitung als Zeichen zum Senden eines neuen Datenpaketes zum FPGA wird durch einen kurzen Befehl vom FPGA realisiert. Eine maßstabsgerechte Darstellung der zeitlichen Abläufe beim Beschreiben und Auslesen der FIFOs wird mit Abb. 3.15 gegeben. Da es bei der USB-Übertragung systembedingt immer wieder zu zeitlichen Verzögerungen kommt, wurde für die zeitliche Darstellung die dreifache Zeit einer unterbrechungsfreien Übertragung gewählt.

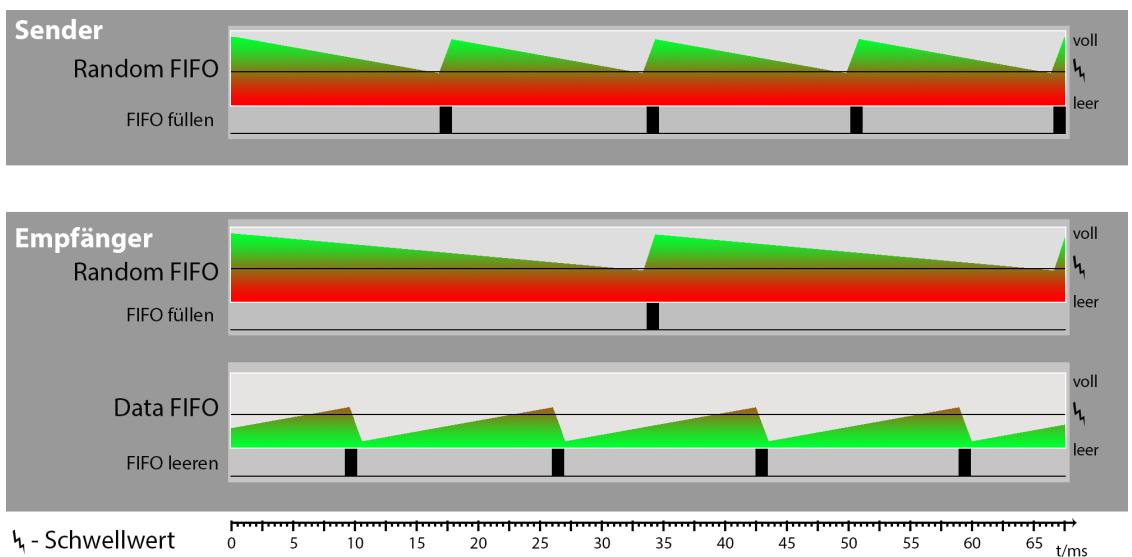


Abbildung 3.15: Zeitliche Darstellung der FIFO-Füllstände bei einer Transmissionsfrequenz von 8 MHz. Die Daten für die Modulatorsteuerung werden kontinuierlich aus den Random-FIFOs gelesen. Wird der Füllstandsschwellwert unterschritten, erfolgt die Wiederauffüllung der FIFOs. Auf der Empfängerseite werden zudem kontinuierlich Daten der detektierten Photonen in den Daten-FIFO eingelesen. Erreicht der Füllstand dieses FIFOs den konfigurierten Schwellwert, werden seine Daten zum PC übertragen.[34]

3.3.4 Steuerung des Versuchsaufbaus

Die Schaltungslogik für die Versuchsaufbausteuerung ist in größtem Maße konfigurierbar gestaltet. Somit kann das System an Veränderungen im physischen Versuchsaufbau angepasst und weiterverwendet werden. Um die geforderte maximale Übertragungsrate von 8MHz gut mit Frequenzteilern realisieren zu können, wird dieses Schaltungsmo-
dul mit einer Taktfrequenz von 64MHz versorgt. Neben den geforderten 8 MHz ist die Frequenz der Photonübertragung weiter teilbar. So sind auch Übertragungsfre-
quenzen von 4, 2, 1 und 0.5 MHz möglich. Einstellbar ist die Frequenz über einen Systemparameter, der über die USB-Verbindung verändert werden kann. Die Schal-
tung ist so ausgelegt, dass die Qubit-Übertragung kontinuierlich und unterbrechungsfrei stattfindet. Dementsprechend muss die Ausfallsicherheit der Datenversorgung sowie des Datenabtransportes gewährleistet sein. Wie schon beim Systementwurf in Kapitel 3.1 beschrieben, ist zur Einhaltung eines festen Zeitrasters für Sender und Empfänger eine Synchronisation notwendig. Dies wird über ein elektronisches Synchronisationssignal, welches über eine BNC-Verbindung vom Sender zum Empfänger übertragen wird, rea-
lisiert. Auf jeder steigenden Flanke synchronisiert sich der Empfänger und passt sich dem Zeitraster des Senders an. Je weiter die Synchronisationsflanken auseinander liegen, desto größer wird der Zeitversatz der durch die Synchronisation ausgeglichen werden muss. Dementsprechend groß wird der Fehler bei der Frequenzanpassung des Emp-
fängermoduls. Aus diesem Grund erfolgt nach jeweils 8 Photonentransmissionen eine Synchronisation. Für den Ablauf jeder Transmission ist bestimmtes Zeitfenster einge-
richtet, in dem der Sender seine Modulatoreinstellung vornimmt, ein Photon aussendet und der Empfänger ebenfalls seinen Modulator einstellt und bereit ist, ein Photon zu detektieren. Fest zu Beginn jedes Zeitfensters werden der Modulator des Senders und der Demodulator des Empfängers eingestellt. Da nun je nach Art des Modulators un-
terschiedlich viel Zeit vergehen kann, bis der Modulator tatsächlich die eingestellten Eigenschaften aufweist, sind Startzeitpunkt und Impulslänge des Stimulationslasertrig-
gers sowie Beginn und Länge des Zeitraums, an dem die Photonendetektoren aktiv sind, konfigurierbar. Dieses Zeitverhalten ist in Abbildung 3.16 näher dargestellt.

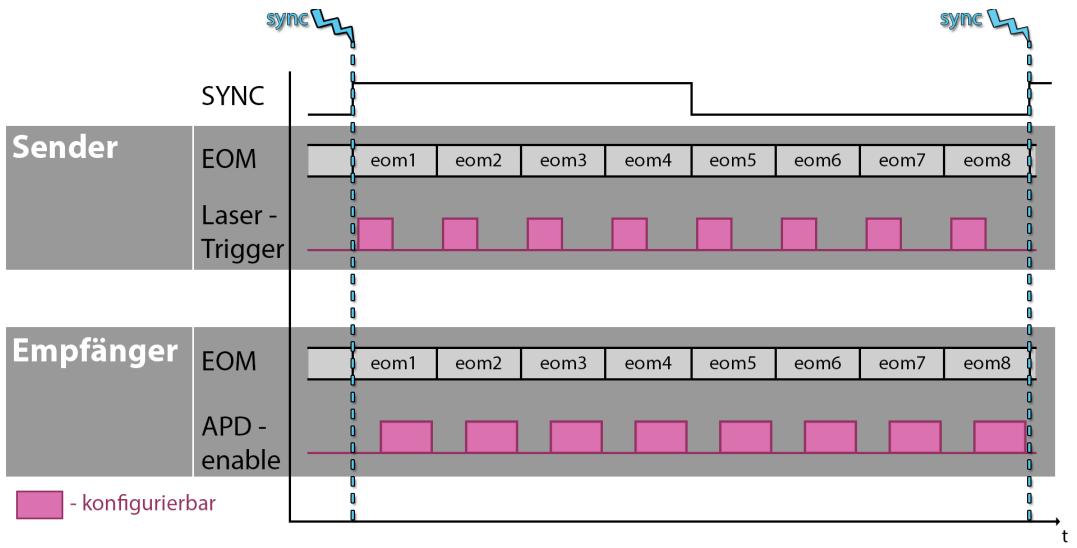


Abbildung 3.16: Konfigurierbares Zeitfenster der Photonentransmissionen. Zwischen zwei SYNC-Impulsen werden acht Photonen übertragen. Im festen Zeitraster erfolgt die Einstellung der EOMs. Start und Ende des Trigerpulses für die Einzelphotonenquelle und der Zeitraum, an dem die APDs aktiviert sind, ist konfigurierbar.[34]

Je nachdem wie die Zufallsdaten zur Modulator- und Demodulatorsteuerung benötigt werden, werden die entsprechenden FIFO-Speicher ausgelesen. Ebenso erfolgt das Einlesen der durch die Photonendetektion entstandenen Daten fest nach dem Zeitrahmen. Das Leeren und Befüllen der FIFOs, erfolgt selbstständig durch die Schaltungslogik der USB-Kommunikation. Die Konfiguration des Zeitfensters sowie die Auswahl der Anzahl der stattfindenden Transmissionen erfolgt mithilfe von Systemparametern.

3.4 Softwarerealisierung

3.4.1 Überblick

Wie schon beim Schaltungsentwurf für die FPGAs besteht auch im Softwareentwurf für Sende- und Empfangseinheit eine gewisse strukturelle Ähnlichkeit. So besitzen beide Programme ein Modul für die Netzwerkkommunikation, zum gegenseitigen Datenaustausch, und ein Modul für die USB-Kommunikation, zum Datenaustausch mit dem FPGA-Board. Beide Systeme müssen Zufallszahlen beziehen, die Photonentransmissio-

nen steuern und die Datennachverarbeitung realisieren. Ausgenommen von der USB-Kommunikation und der Beschaffung der Zufallszahlen unterscheiden sich die Module jedoch voneinander. Hinzu kommt eine grafische Benutzeroberfläche zur Systemkonfiguration. Diese ist auf dem Empfängersystem realisiert. Die strukturelle Ähnlichkeit ist auch in den Klassendiagrammen zu erkennen, die in Abb. 3.17 dargestellt sind.

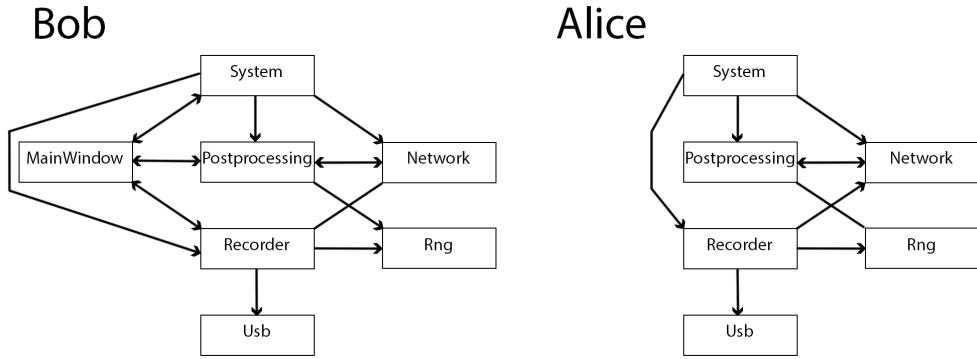


Abbildung 3.17: Klassendiagramme im Vergleich.[34]

Zentrale Klasse ist jeweils die Klasse **System**. Diese wird in der `main()`-Funktion instanziiert und erzeugt die Objekte der benötigten Klassen. Zudem erfolgt in der Klasse **System** in beiden Programmen die Verbindung einzelner Signale zu den entsprechenden Slots, was die Kommunikation der erzeugten Objekte untereinander ermöglicht. Nach dem Start der Programme beginnt die Initialisierungsphase. Dabei wird eine Netzwerkverbindung zwischen den Systemen hergestellt. Das System des Senders übernimmt dabei die Rolle des Servers, indem es auf eingehende Verbindungsanfragen wartet. Startet das Empfängersystem, der Client, sucht er nach dem Server mit der angegebenen IP-Adresse. Stimmen beidseitig die Portnummern überein, so wird eine Verbindung zwischen beiden Systemen aufgebaut. Weiterhin stellen sowohl Sender als auch Empfänger eine Verbindung zum Zufallszahlenserver her. Es erfolgt dabei eine Anmeldung mit Benutzername und Passwort. Neben den Netzwerkverbindungen nehmen beide PCs Verbindung zum USB-Modul auf dem FPGA-Board auf und konfigurieren dieses. Sind alle Objekte erzeugt und wurden alle Verbindungen erfolgreich hergestellt, befindet sich das Programm auf der Empfängerseite, welches die GUI enthält, in der sogenannten Event-Loop. Es wartet darauf, dass eine Eingabe durch den Benutzer erfolgt. Das Programm auf der Senderseite wartet auf eingehende Befehle vom Netzwerk. In diesem Zustand kann die Systemkonfiguration erfolgen. Parameteränderungen werden über die grafische Benutzeroberfläche aufgenommen und an die entsprechenden Instanzen weitergegeben. Betreffen die Einstellungen das senderseitige System, so werden die Parameterände-

rungen über einen entsprechenden Protokollrahmen an dieses weitergegeben und dort verarbeitet. Wird über die grafische Benutzeroberfläche der Vorgang der Photonentransmissionen gestartet, erfolgt die entsprechende Signalisierung an die Senderseite. Anhand der eingestellten Anzahl an Gesamttransmissionen beschaffen sich beide Systeme eine ausreichende Menge an Zufallszahlen und speichern diese auf der Festplatte ab. Diese Vorabbeziehung der Zufallszahlen vom Zufallszahlenserver ist notwendig, weil die Übertragungsrate, mit der die Daten bezogen werden, abhängig vom Netzwerkverkehr und der Auslastung des Zufallszahlenservers ist. Eine schlechte Übertragungsrate würde im Falle einer Direktbeziehung zum Zeitpunkt der Übertragung zum FPGA-System möglicherweise eine Unterbrechung der Datenversorgung verursachen und somit gravierende Fehler verursachen. Liegen die Daten bei beiden Parteien bereit, erfolgt eine Synchronisation über das Netzwerk und den FPGA-Systemen werden ein Startsignal übermittelt. Der Vorgang der Übertragung wird selbstständig von den FPGA-Systemen durchgeführt. Die PCs sorgen dafür, dass alle FIFOs über ausreichend Daten verfügen oder dass entsprechend Daten vom System ausgelesen und auf der Festplatte gespeichert werden. Alle Datenübertragungen werden mit Hilfe von Prüfsummen auf ihre Richtigkeit überprüft. Sollten fehlerhafte Datenpakete übertragen werden, werden diese vorgemerkt und finden beim späteren Postprocessing keine Beachtung. Wurde der Transmissionsvorgang beendet, befinden sich beide Programme wieder in Warteposition und warten auf neue Anweisungen. Nach einer möglichen erneuten Konfigurationsphase kann nun das Postprocessing gestartet werden. Aus den entstandenen Schlüsseldaten wird im Anschluss an das Postprocessing eine binäre Bilddatei erstellt. Dabei werden Nullen als schwarze und Einsen als weiße Bildpunkte dargestellt. So kann dem Systembenutzer Einblick in die Güte des Kryptographieschlüssels gewährt werden.

3.4.2 GUI

Die grafische Benutzeroberfläche wurde mit dem GUI-Designer des Qt-Creator erstellt. Aus Gründen der besseren Übersicht wurden die Bedienelemente in die Kategorien Systembedienung und Systemkonfiguration unterteilt und auf verschiedenen Tabs angeordnet. Der Tab zur Systembedienung ist durch graphische Rahmen in die drei Module Transmission, Postprocessing und Key Analysis unterteilt. In den Modulen Transmission und Postprocessing sind Elemente zum Starten der jeweiligen Vorgänge enthalten. Statusleisten geben dem Anwender einen Überblick über den zeitlichen Verlauf des Vorgangs. Da die Anzahl der Photonentransmissionen und die Transmissionsfrequenz

immer angegeben werden müssen, befinden sich Elemente für die entsprechenden Einstellungen auf dem Bedienungs-Tab. Im Modul Key Analysis ist die Größenveränderung der Schlüsseldaten im Postprocessing dargestellt. Zudem wird eine skalierte Version des, beim Postprocessing erstellten, Binärbildes dargestellt. Durch Anklicken des Bildes öffnet sich ein neues Fenster, in dem das Bild größer dargestellt wird. Im Konfigurationstab sind sämtliche tiefergehenden Konfigurationsmöglichkeiten enthalten. Diese sind die Auswahl der IP-Adresse für die Netzwerkkommunikation, die Anmeldedaten für den Zufallszahlenserver und die Konfigurationsdaten des Zeitrahmens für die Photonentransmissionen. Die grafische Benutzeroberfläche ist in Abb. 3.18 dargestellt.

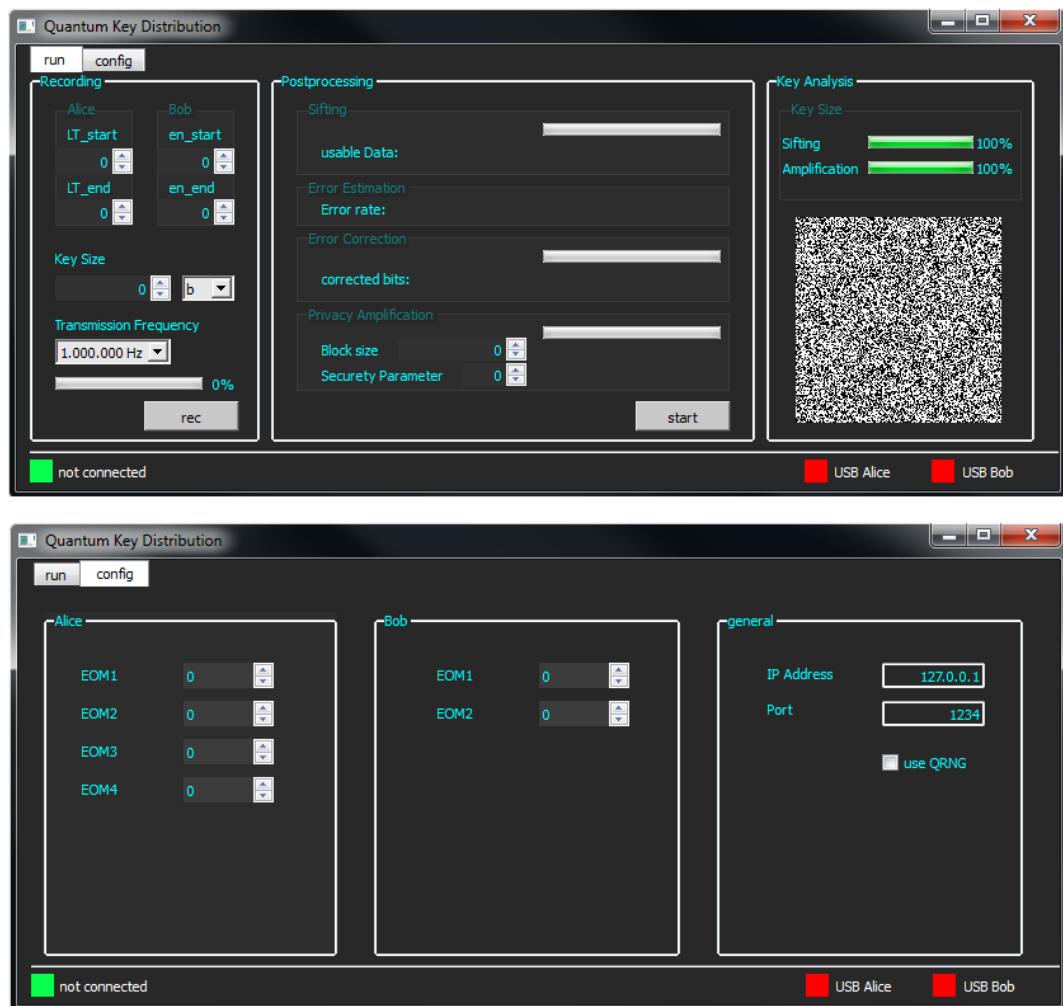


Abbildung 3.18: Grafische Benutzeroberfläche.[34]

3.4.3 Netzwerkkommunikation

Die Netzwerkkommunikation zwischen Sender und Empfänger wird über TCP-Sockets realisiert. Die Verbindung ist nach dem Client-Server-Prinzip aufgebaut. Die Sendeeinheit übernimmt die Rolle des Servers und antwortet auf die Anfragen des Empfängermoduls, der dabei die Rolle des Clients übernimmt. Die Umsetzung im Programm erfolgt mit Hilfe der Qt Netzwerkbibliothek. Sämtliche Programmlogik für die Netzwerkkommunikation zwischen den beiden Systemen wird in der Klasse Network zusammengefasst. Um in dieser die Signal/Slot-Methodik anwenden zu können, erbt die Klasse von QObject. Der Verbindungsauflaufbau des Servers wird mit Objekten der Klassen QTcpServer und QTcpSocket realisiert. Im Code-Listing 3.2 ist stark vereinfachter Programmcode der Klasse Network enthalten, der das Realisierungsprinzip der serverseitigen Socket-Verbindung darstellt.

Listing 3.2: Realisierungsprinzip der serverseitigen Socket-Verbindung

```

1  class Network : public QObject
2  {
3      Q_OBJECT
4  private:
5      QTcpServer *server;
6      QTcpSocket *socket;
7  public:
8      explicit Network(QObject *parent = 0);
9  public slots:
10     void newConnection();
11 };
12
13 Network::Network(QObject *parent) : QObject(parent) {
14     server = new QTcpServer(this);
15     connect(server, SIGNAL(newConnection()), this, SLOT(newConnection()));
16     server->listen(QHostAddress::Any, 1234)
17 }
18
19 void Network::newConnection() {
20     socket = server->nextPendingConnection();
21 }
```

Im Konstruktor der Klasse **Network** wird ein Objekt vom Typ **QTcpServer** erzeugt. Im Anschluss erfolgt die Verbindung des Signals **newConnection()** der **QTcpServer**-Klasse mit dem erzeugten Slot der **Network**-Klasse, der ebenfalls den Namen **newConnection()** trägt. In Zeile 16 wird der Server durch den Methodenaufruf **listen(QHostAddress::Any,1234)** dazu veranlasst, auf eingehende Verbindungsanfragen von jeder beliebigen IP-Adresse unter der Portnummer 1234 zu warten. Im Falle einer neuen Verbindung wird der Slot **newConnection()** aufgerufen. In diesem erfolgt der Methodenaufruf **server->nextPendingConnection()**. Diese gibt die nächste ausstehende Verbindung als **QTcpSocket**-Objekt zurück. Über dieses Objekt kann nun mit Hilfe der Methoden **read()** und **write()** der Datenaustausch realisiert werden. Die Verbindungsrealisierung auf der Seite des Clients erfolgt ausschließlich mit einem Objekt der Klasse **QTcpSocket**. Der beispielhafte Programmcode ist im Code-Listing 3.3 dargestellt.

Listing 3.3: Realisierungsprinzip der clientseitigen Socket-Verbindung

```
1     socket = new QTcpSocket(this);
2     socket->connectToHost("127.0.0.1",1234);
```

Das **QTcpSocket**-Objekt wird in der **Network**-Klasse auf der Empfängerseite erzeugt und stellt mit der Methode **connectToHost(...)** eine Verbindung zum Server her. Durch die angegebenen Parameter dieser Methode im beispielhaften Quellcode wird dabei eine Verbindung zum Localhost hergestellt. Als Portnummer findet, wie auch im Beispielcode des Servers, die 1234 Verwendung. Der eigentliche Datenaustausch erfolgt auf der Clientseite ebenso mit den Socketmethoden **read()** und **write()**.

Für die Übertragung von Parametern während der Konfigurationsphase ist ein Protokoll eingerichtet, dass eine Adressierung einzelner Parameter ermöglicht. Übertragen wird dazu eine Adresse in Form einer char-Variablen, gefolgt von einem Integer als Parameterwert. Da es ausschließlich Befehle zum Setzen von Parametern gibt, ist keine Unterscheidung von verschiedenen Befehlsarten notwendig. Das Setzen von Parametern geschieht ausschließlich von der Empfängerseite aus, da sich dort die Bedienoberfläche befindet. Die Kommunikation zwischen Sender und Empfänger während des Postprocessings erfolgt ohne Protokollrahmen. Dieser ist überflüssig, da beide Seiten stets prozessbedingt wissen, wie viele Daten sie zu übertragen haben beziehungsweise wie viele Daten einzulesen sind. Da beide Seiten unterschiedlich umfangreiche Berechnungen während

der einzelnen Arbeitsschritte des Postprocessings vornehmen, ist es notwendig, für eine zeitliche Synchronisation beider Systeme zu sorgen. Damit ist gemeint, dass das System, welches Daten einlesen will solange warten muss, bis das entsprechend andere System Daten versendet. Andersherum muss das System, welches Daten versendet, solange warten, bis die Gegenseite zum Empfang der Daten bereit ist. Dazu ist ein sogenanntes Handshake eingerichtet, welches vor der Übertragung eines größeren Datenblocks vollzogen wird. Dabei versendet derjenige, der ein großes Datenpaket versenden will, die kurze Anfrage 'ready?' und wartet solange, bis er ein 'yes' empfängt. Somit signalisieren beide Parteien, dass sie für die Übermittlung des großen Datenpaketes bereit sind.

Da ein möglicher Abhörer auch Informationen über den Schlüssel erhalten kann, indem er sich an der Kommunikation während des Postprocessings beteiligt, ist es erforderlich, dass die Kommunikation autorisiert abläuft. Dazu wird für jedes Datenpaket ein sogenannter Token erstellt. Dieser wird nach einer für beide Seiten festgelegten Berechnungsvorschrift aus den Daten der Nachricht und aus, auf beiden Seiten identisch vorliegenden, Zufallsdaten errechnet. Beim Empfang des Datenpaketes wird entsprechend der Token berechnet. Stimmt dieser mit dem, an die Nachricht angehängten Token überein, kann davon ausgegangen werden, dass die Nachricht vom autorisierten Kommunikationsteilnehmer stammt. Die richtige Berechnung des Tokens ist nur möglich, wenn dem Kommunikationsteilnehmer ein identisches Datenpaket vorliegt. Dieses kann vorher, im Rahmen eines Treffens, oder in einem vorausgegangenen Schlüsselverteilungsprozess übermittelt worden sein.

3.4.4 USB-Kommunikation

Das USB-Modul wird über einen Treiber angesprochen, der auf dem PC-System installiert wird. Zu diesem Treiber ist eine Programmbibliothek erhältlich, die die dazu benötigten Funktionen bereitstellt. Diese Bibliothek ist in die Klasse **Usb** eingebunden. Vor dem Verbindungsauflauf zu USB-Modul, wird eine sogenannte Device-Liste angefordert. Diese enthält sämtliche an den Computer angeschlossenen FTDI-Module. Das gewünschte Modul kann anhand des Namens oder seiner ID identifiziert und gezielt eine Verbindung zu ihm hergestellt werden. Somit wird verhindert, dass versehentlich ein anders Modul angesprochen und konfiguriert wird. Wurde das Modul erkannt und erfolgreich eine Verbindung mit ihm hergestellt, erfolgt die Konfiguration auf das gewünschte elektronische Ausgabeprotokoll. Der eigentliche Datenaustausch erfolgt mit

den Funktionen **readData()** und **writeData()**. Daten die so vom PC gesendet wurden, werden umgehend zum USB-Modul übertragen und können dort abgerufen werden. Sollen Daten eingelesen werden, kann zuvor eine Statusanfrage gestellt werden und damit ermittelt werden, ob sich Daten im Empfangsbuffer befinden. Um den Prozessor zu entlasten wird ein Eventhandling unterstützt. Damit ist es möglich, den aktuellen Thread in den Wait Mode zu überführen. Erst bei der Ankunft neuer Daten im Empfangsbuffer wird der Thread wieder aktiv. Aufgrund des zeitkritischen Verhaltens, wurde die Datenübertragung mit dem USB-Modul in einen eigenen Thread ausgelagert. Dieser erhielt die höchstmögliche Priorität. Damit soll verhindert werden, dass der zeitkritische Datenaustausch durch einen Prozess höherer Priorität blockiert wird.

3.4.5 Beschaffung der Zufallszahlen

Auch der Zugriff auf die Quantenzufallszahlen wird über ein sogenanntes Application Programming Interface (API) realisiert. Dieses ist für verschiedene Programmiersprachen verfügbar. Realisiert wird der Zugriff über eine dynamische Bibliothek. Aus dieser werden Funktionen verwendet, die das Anmelden am Server mit Benutzername und Passwortabfrage ermöglichen. Weiterhin sind Funktionen zum Download der Zufallsdaten in verschiedenen Zielformaten verfügbar. Im Programm kann entschieden werden, ob die Zufallszahlen vom Server geladen werden oder ob die Zufallszahlen offline bezogen werden. In letzterem Fall werden sie mit der Funktion **qrand()** erzeugt.

3.4.6 Postprocessing BB84

3.4.6.1 Beschreibung

Nachdem die Qubitübertragungen stattgefunden haben, liegen Alice pro einzelner Photonentransmission zwei Bit an Daten vor. Ein Bit entspricht der Wahl der Modulationsbasis, das andere gibt den eigentlichen Bitwert an. Diese wurden vor der Übertragung rein zufällig gewählt, entsprechen aber den tatsächlichen Einstellungen bei der senderseitigen Modulation. Bob hingegen liegen pro Einzeltransmission drei Bit vor. Ein ebenfalls zufällig gewähltes Bit entspricht der Basis, bei der das Photon detektiert worden ist. Die übrigen zwei Bits geben jeweils den Ausschlag eines Detektors an, der

somit die Ankunft der einzelnen Photonen signalisiert. Im ersten Arbeitsschritt werden auf beiden Seiten die Messwerte der Übertragungen entfernt, bei der es entweder zum Ausschlag beider Detektoren gekommen ist oder bei dem kein Photon erkannt wurde. Die Ursache von Doppelausschlägen liegen darin begründet, dass entweder ein Fremdphoton den Ausschlag verursacht hat oder es zu einem Fehlausschlag eines Detektors kam. Wurde kein Photon gemessen, kann es bei der Übertragung absorbiert worden sein oder es wies beim Austritt aus der Photonenquelle eine abweichende Polarisation auf und konnte so den Polarisationsfilter nicht passieren. Die Größenordnung dieser ungültigen Übertragungen liegen im Rahmen einiger Zehntel Promille. Da sie für die weitere Verarbeitung keinen Informationsgehalt besitzen, müssen sie dennoch entfernt werden. Im Anschluss werden die Informationen über die Wahl der Modulationsbasen für die Übertragungen über den offenen Kommunikationskanal, das Netzwerk, ausgetauscht. Bei diesem als Sifting (englisch für Sieben) bezeichneten Prozess, werden nach dem Basenaustausch die Bitwerte der Transmissionen entfernt, bei denen Sender und Empfänger eine unterschiedliche Basenwahl getroffen haben. Bei ausreichender Übertragungsmenge und hoher Qualität der Zufallszahlen werden dabei etwa 50% der Daten verworfen.

Im folgenden Schritt erfolgt eine Fehlerabschätzung der Rohschlüsseldaten. Dazu werden zufällig ausgewählte zwei Prozent der nun ausgesiebten Daten von Bob an Alice gesendet und miteinander verglichen. Durch einen erhöhten Fehler kann man hier einen Abhörer erkennen. Wurde die Übertragung aufgrund der Fehlerabschätzung für gültig erklärt, erfolgt die Fehlerkorrektur. Dabei ist es möglich, einzelne Bitfehler zu erkennen und zu beheben ohne die eigentlichen Bitinformationen des Schlüssels preiszugeben. Angewendet wird dabei der CASCADE - Algorithmus, welcher 1994 von G. Brassard und L. Salvail vorgestellt wurde. Dabei werden die gesamten Schlüsseldaten in einzelne Blöcke unterteilt, deren Größe anhand der vorangegangenen Fehlerabschätzung bestimmt wird. Die Größe der Blöcke wird dabei so gewählt, dass, bei Annahme einer statistischen Gleichverteilung, ein Block einen einzelnen Fehler enthält. Für jeden einzelnen Block wird senderseitig und beim Empfänger die Parität bestimmt und beide über den offenen Kommunikationskanal miteinander verglichen. Ist die Parität für den Block auf beiden Seiten unterschiedlich, so ist davon auszugehen, dass sich mindestens ein Bitfehler in diesem Block befindet. Daraufhin wird die Größe des Blocks halbiert und die Parität der ersten Hälfte bestimmt und erneut miteinander verglichen. Ist ein Paritätsunterschied in der ersten Hälfte festzustellen, so wird der entsprechende Block erneut halbiert. Wird für die erste Blockhälfte kein Paritätsunterschied festgestellt, so muss sich der Bitfehler

in der zweiten Blockhälfte befinden, die daraufhin unterteilt wird. Dieser Prozess wird solange fortgesetzt, bis der Fehler auf ein Bit begrenzt wurde. Das entsprechend fehlerhafte Bit wird auf einer Seite invertiert. Somit wurde der Fehler beseitigt. Dies erfolgt nun für alle weiteren Blocks, bis alle Fehler behoben wurden. In Abb. 3.19 ist der Vorgang eines Korrekturprozesses für einen Block mit einer Größe von 32 Bits dargestellt. Wie farblich zu erkennen ist, befinden sich drei fehlerhafte Bits im Block. Aufgrund der somit ungeraden Anzahl von Fehlern wurde ein Paritätsunterschied festgestellt und der Block für den Korrekturprozess ausgewählt. Nach dem Paritätsvergleich für die erste Blockhälfte (Bits 1..16) wurden erneut unterschiedliche Paritäten festgestellt. Die erste Blockhälfte wird dementsprechend erneut halbiert. Für diese erste dieser Blockhälften (Bits 1..8) ist wieder ein Paritätsunterschied feststellbar. Was auf einen Fehler in diesem Block schließen lässt. Nach erneuter Halbierung besteht der Block nun aus 4 Bit (Bits 1..4). Der Paritätsvergleich zeigt nun keinen Unterschied. Daraufhin wird die zweite Hälfte des Blocks halbiert. Da dieser Block (Bits 5..6) nun erneut keinen Paritätsunterschied aufweist, muss sich der Fehler in der zweiten Blockhälfte befinden. Diese zweite Blockhälfte wird erneut halbiert. Übrig bleibt ein einzelnes Bit (Bit 7). Da die Parität eines einzelnen Bit dem Bitwert selbst entspricht, macht es keinen Unterschied, ob die Parität oder der Bitwert selbst miteinander verglichen wird. Der Vergleich zeigt, dass beide Bits identisch sind. Somit kann sich der Fehler nur in Bit 8 befinden.

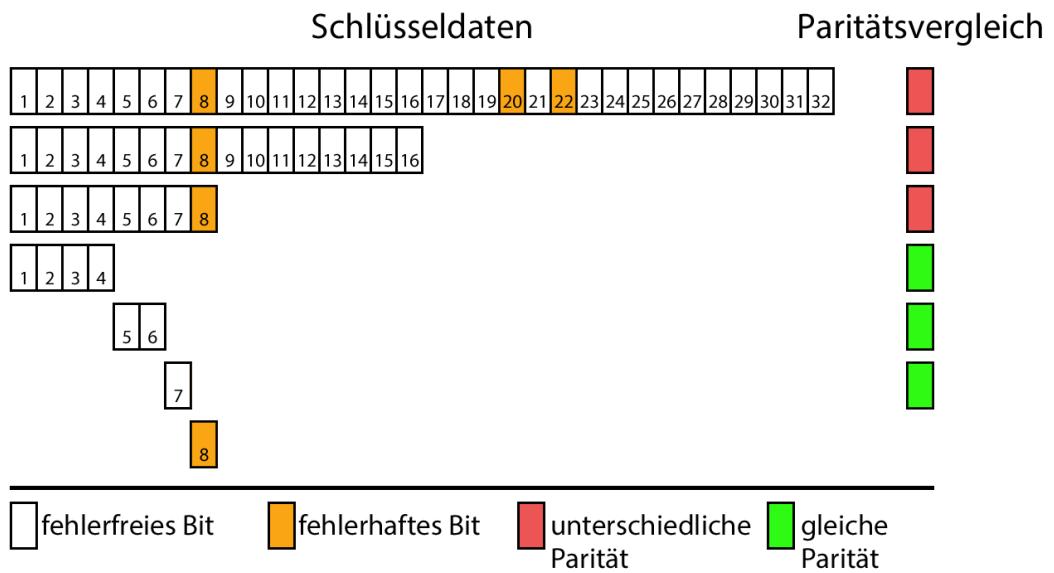


Abbildung 3.19: Beispielhafte Darstellung des CASCADE Algorithmus. Durch Paritätsvergleiche lässt sich eine ungerade Anzahl an Fehlern in einem Block feststellen. Durch die Unterteilung des Blockes und weitere Paritätsvergleiche lässt sich der Fehler auf ein Bit eingrenzen. [1]

Da bei diesem Verfahren jeweils nur eine ungerade Anzahl von Fehlern in einem Block erkannt und behoben werden kann, ist es erforderlich, die kompletten Schlüsseldaten zu durchmischen und den Fehlerkorrekturprozess erneut zu durchlaufen. Da schon eine bestimmte Anzahl von Fehlern korrigiert wurde, wird die anfängliche Blockgröße vergrößert. Dieses Vorgehen wird solange fortgeführt, bis sich die Blockgröße aufgrund der vorliegenden Datenmenge nicht weiter vergrößern lässt. Ein Abhörer könnte, durch das Belauschen der Kommunikation während des Korrekturvorganges viele Informationen über den Schlüssel, bis hin zu einzelnen Bitwerten erfahren. Daher werden in einem weiteren Prozess, Privacy Amplification genannt, die Schlüsseldaten so verändert, dass die Abhörinformationen ihre Gültigkeit verlieren. Dies wird erreicht, indem die Daten als Ganzes oder in großen Blöcken mit, auf beiden Seiten identisch vorliegenden, Zufallszahlen multipliziert werden.

3.4.6.2 Umsetzung

Die Ausgangsdaten für das Postprocessing liegen in Form von drei Binärdateien vor. Senderseitig, bei Alice, liegt eine Datei mit den bezogenen Zufallszahlen vor. Diese wurden während des Transmissionsvorgangs in das FPGA geladen und dort für die Polarisationsmodulation der Einzelphotonen verwendet. Zwei aufeinanderfolgende Bits entsprechen demzufolge der Modulationsbasis und dem Bitwert einer einzelnen Photonenübertragung. Auf der Seite des Empfängers, bei Bob, liegen zwei Binärdateien vor. Eine enthält ebenfalls Zufallszahlen die für die empfängerseitige Modulatorsteuerung verwendet wurden. Jedes einzelne Bit dieser Datei entspricht der Wahl der Basis einer einzelnen Transmission. Weiterhin liegt eine Binärdatei vor, die die Detektionsdaten der beiden APDs enthält. Zwei Bits entsprechen dabei einer Photonentransmission. Die Daten aus den Binärdateien werden in Form eines Character-Arrays eingelesen und in dieser Form auch weiterverarbeitet. Der Zugriff auf die einzelnen Bits erfolgt durch Maskierung der entsprechenden Arrayelemente.

Sifting

Ursprünglich ist das Vorgehen im Postprocessing so beschrieben, dass Bob beim Basenvergleich Alice nur die Basen der gültigen Transmissionen nennt. Dies setzt eine Indizierung der einzelnen Transmissionen voraus. Als Index ist dabei eine Zahl zu ver-

stehen, die durch die Anzahl der Bitstellen auf einen festen Wertebereich festgelegt ist. Es könnten demnach nur so viele Transmissionen stattfinden, wie durch den Indizes unterschieden werden können. Um eine große Anzahl von Transmissionen zu ermöglichen, müsste die Anzahl der Bitstellen des Indizes recht groß gewählt werden. Dies führt zu einem sehr hohen Datenaufkommen beim Basenvergleich. Um dieses Problem zu umgehen und bei der Transmissionenanzahl zudem unabhängig von der Anzahl der Bitstellen des Indizes zu sein, wird für die Informationsübermittlung ein Bitarray genutzt. Jeder Transmission wird dabei ein Bit in diesem Array zugeordnet. Die Indizierung erfolgt durch die Reihenfolge der einzelnen Bits im Array und bedarf somit keiner anderweitigen Indizierung. Da in einem Bit immer nur eine ja/nein-Information gespeichert werden kann, müssen die Informationen über die Gültigkeit einer Transmission und die Wahl der Modulationsbasis separat übertragen werden. Zusammenfassend lässt sich so der Informationsaustausch auf zwei Bit pro stattgefunder Transmission reduzieren. Dank der Indizierungsmethode können so beliebig viele Transmissionsergebnisse verarbeitet werden. Der Informationsgehalt des Datenaustausches bleibt bei dieser Übertragungsform identisch. Somit erhält ein möglicher Abhörer nicht mehr Informationen, als dies in der üblichen Übertragungsform der Fall wäre.

Fehlerbestimmung (Error Estimation)

Bei der Fehlerbestimmung werden üblicherweise 2% der Transmissionsdaten beider Parteien miteinander verglichen. Die Auswahl, welche Transmissionswerte veröffentlicht werden, geschieht zufällig. Dazu besorgt Alice Zufallszahlen und sendet diese an Bob. Auf beiden Seiten werden daraus die Indizes der zu vergleichenden Daten gebildet. Beide Parteien sortieren sich die Daten mit den entsprechenden Indizes aus. Alice schickt diese an Bob, der den eigentlichen Vergleich vornimmt. Die dabei entstehende Fehlerrate sendet er an Alice zurück. Anschließend ist es erforderlich die veröffentlichten Daten aus den Ursprungsdaten zu entfernen.

Fehlerkorrektur (Error Correction)

Wie in der Beschreibung des Postprocessings erläutert, erfolgt die Fehlerkorrektur nach dem CASCADE Algorithmus. Dabei werden die Gesamtdaten in Blöcke unterteilt und die Parität derer bestimmt und ausgetauscht. Je nach dem Ergebnis des Paritätsver-

gleichs werden die Blöcke weiter unterteilt bis das Fehlerbit identifiziert ist. Am zeit-aufwändigsten an diesem Vorgang ist der Netzwerkverkehr, der zum Paritätsvergleich stattfindet. Zwar lassen sich durch diesen schnell auch große Datenmengen übertragen, doch vergeht wertvolle Zeit, bis eine solche Transmission gestartet wird. Realisiert man die Unterteilung und den Paritätsvergleich der einzelnen Blöcke hintereinander, müssen sehr viele einzelne Transmissionen stattfinden. Dies führt durch die Verzögerungen im Netzwerkverkehr zu einer langen Gesamtbearbeitungszeit. Zur Effizienzsteigerung ist daher der CASCADE-Algorithmus so realisiert, dass er mit möglichst wenigen Netzwerktransmissionen auskommt. Dies wird erreicht indem die Blockunterteilung nicht hintereinander sondern parallel abläuft. Dazu werden die gesamten Daten in Blöcke unterteilt und hintereinander die Parität der einzelnen Blöcke bestimmt. Sind die Paritäten aller Blöcke bestimmt, erfolgt deren Übertragung in einem Datenpaket. Gespeichert und an die Gegenseite übertragen werden die Paritätsbits wieder in der effektivsten Form, als Bitarray. Nachdem die Paritätsdaten ausgetauscht wurden, wissen beide Seiten, Alice und Bob, welche Blöcke Paritätsunterschiede aufweisen und können die entsprechenden Blöcke unterteilen. Anschließend wird erneut die Parität für alle nun unterteilten Blöcke bestimmt und als Ganzes übertragen. Dieser Vorgang wiederholt sich solange, bis die einzelnen Fehlerbits erkannt und korrigiert sind. Da durch die Paritätsvergleiche nur eine ungerade Anzahl an Fehlern in einem Block erkannt werden kann, ist es erforderlich, die Daten zu durchmischen und den Korrekturvorgang erneut zu durchlaufen. Da schon eine Reihe von Fehlern korrigiert wurden, kann die Blockgröße im nächsten Durchlauf vergrößert werden. Im einfachsten Fall wird sie verdoppelt. Das Durchmischen der Schlüsseldaten wird durch einen Algorithmus realisiert, dessen Arbeitsweise in Abb. 3.20 dargestellt ist.

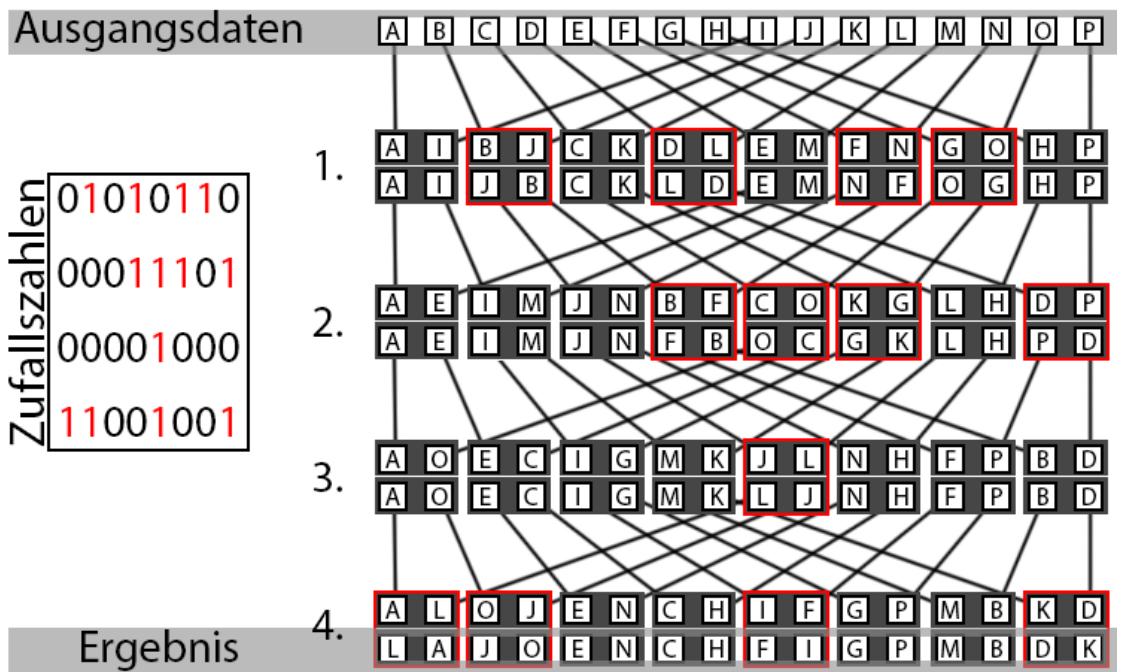


Abbildung 3.20: Misch-Algorithmus. Die Ausgangsdaten werden in den einzelnen Schritten, wie dargestellt, umsortiert. Anhand von Zufallszahlen wird nach jeder Umsortierung entschieden, ob zwei aufeinanderfolgende Bits vertauscht werden. Bei einer Gesamtzahl von N Elementen werden die Umsortierung und die Bitvertauschung $\log N$ mal ausgeführt.[33]

Die Daten werden zunächst logisch in zwei Hälften unterteilt. Anschließend werden die einzelnen Elemente abwechselnd aus der ersten und der zweiten Hälfte neu zusammengesetzt. Danach werden zwei nebeneinander liegende Bits in Abhängigkeit von Zufallsbits vertauscht. Ist das Zufallsbit 0, wird nicht getauscht. Ist es 1, wird getauscht. Das gesamte Vorgehen wird bei einer Gesamtzahl von N Datenbits $\log N$ mal wiederholt. Somit hat jedes Bit eine gleichverteilte Chance an jede beliebige Stelle der durchmischten Daten zu gelangen. Der gesamte CASCADE-Algorithmus wird unter Verdopplung der Blockgröße solange durchgeführt, bis die gesamten Schlüsseldaten innerhalb eines Blockes liegen.

Privacy Amplification

Wie in der Beschreibung des Postprocessings erläutert, werden die nun fehlerkorrigierten Schlüsseldaten mit Zufallszahlen multipliziert, um so die Informationen, die ein

Abhörer über einzelne Bits hätte erhalten können, unbrauchbar zu machen. Üblicherweise werden dabei als Zufallszahlen die Daten der Modulationsbasen verwendet, da diese schon im Sifting ausgetauscht und verglichen wurden. Somit liegen sie Alice und Bob identisch vor und müssen nicht erneut bezogen und ausgetauscht werden. Um die Multiplikation von so großen Zahlen zu realisieren, wurde die GMP Bibliothek verwendet. Diese ermöglicht die Anwendung mathematischer Operationen auf beliebig große Zahlen. Verwendung finden dabei die momentan effektivsten Algorithmen. Die Verwendung von Funktionen dieser Bibliothek sind im folgenden Codelisting dargestellt.

Listing 3.4: Verwendung der GMP-Bibliothek bei der Multiplikation großer Zahlen

```
1  mpz_t a,b,c;                      // gmp integer
2  mpz_init2(a, number*8);            // initialization of a
3  mpz_init2(b, number*8);            // initialization of b
4  mpz_init2(c, number*8*2);          // initialization of c
5
6  mpz_import(a,number,2,1,0,0,key);   // import key data
7
8  mpz_import(b,number,2,1,0,0,random); // import random data
9
10 mpz_setbit(a,0);                  // make key block odd
11 mpz_mul(c,a,b);                  // multiply the blocks
12
13 mpz_export(result,&num,-1,1,0,0,c); // export result
```

3.5 Test und Inbetriebnahme

Auch durch Systemtests ist es nur selten möglich, die absolute Fehlerfreiheit eines komplexen Systems festzustellen. Mit ihnen kann jedoch die Arbeitsweise des Systems in definierten Normal- und Extremsituationen überprüft werden. Orientierung bei der Planung der einzelnen Tests bietet das V-Modell. Dieses beschreibt die Vorgehensweise bei der Systementwicklung, bei der der Systementwurf in verschiedene Phasen unterteilt und jede dieser Entwurfsphasen durch entsprechende Tests überprüft wird. Das V-Modell ist in Abb. 3.21 dargestellt.

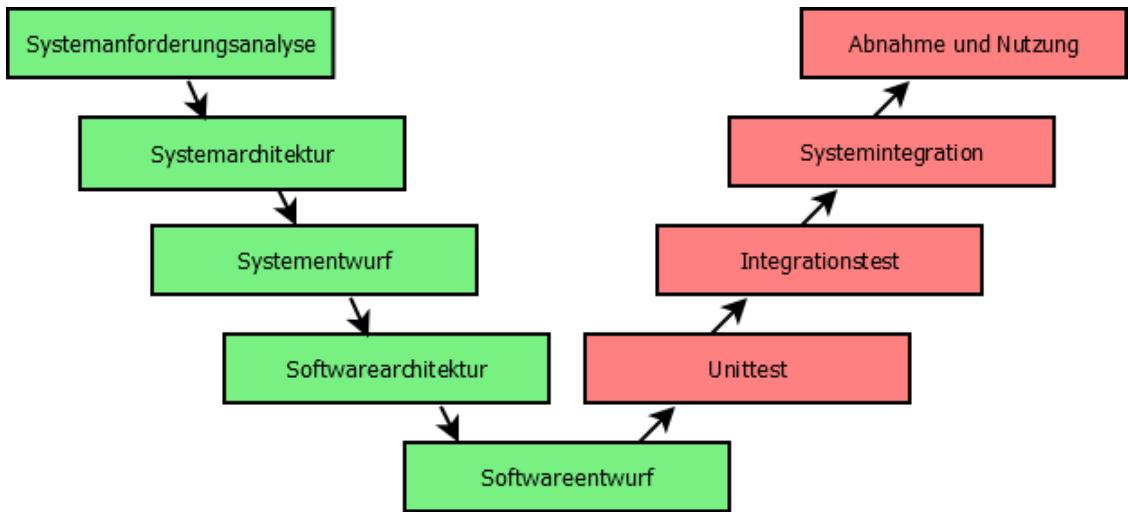


Abbildung 3.21: V-Modell. Jedem der dargestellten Entwicklungsprozesse(grün) wird in diesem Modell ein Testverfahren(rot) zugeordnet.[34]

Die Tests erfolgen demnach in vier Ebenen. Auf der niedrigsten Ebene finden die Unit-Tests statt. Dabei werden die Funktionalität und das Verhalten einzelner voneinander abgegrenzter Komponenten und Module geprüft. Beim Integrationstest auf der nächst höheren Ebene wird das Zusammenspiel voneinander abhängiger Komponenten geprüft. Schließlich wird beim Systemtest das komplette System auf die anfangs gestellten Anforderungen getestet. Zum Schluss erfolgt der Abnahmetest durch den Auftraggeber. Die einzelnen Tests auf den entsprechenden Ebenen werden im Folgenden näher erläutert.

Elektronisches System

Bevor überhaupt mit der Systementwicklung begonnen werden kann, ist es erforderlich, das elektronische System zu testen. Dies beginnt mit der Inbetriebnahme der FPGA-Boards. Dazu werden die Boards mit Spannung versorgt und ein erstes Beispieldesign in das FPGA geladen. Um anschließend die Fehlerfreiheit aller verwendeter Anschlusspins der Boards zu überprüfen, wird ein im FPGA erzeugtes Rechtecksignal auf sämtliche Ausgangspins geleitet. Mit einem Oszilloskop kann die Funktionsfähigkeit der einzelnen Pins überprüft werden. Zum Überprüfen der Eingänge wurde ein Schaltungsdesign erstellt, dass die als Eingänge vorgesehenen Pins auf die nun verifizierten Ausgangspins leitet. Mit Hilfe eines Funktionsgenerators, der ein definiertes Rechtecksignal in die Eingänge leitet, und einem Oszilloskop am entsprechenden Ausgangspin, zum Überprüfen

des ausgegebenen Signals, kann die Funktionsfähigkeit der Eingänge sichergestellt werden. Nachdem die Boards getestet sind, erfolgt die Überprüfung der Adapterplatinen. Zwar wurden diese nach der Fertigung vom Hersteller elektronisch getestet, dennoch können sich Fehler im Platinenentwurf und bei der Bestückung zeigen. Die korrekte Bestückung von Anschlusssteckern kann mit einem Durchgangsprüfer getestet werden. Nachdem die Adapterplatinen an die FPGA-Board angeschlossen sind, können die Ein- und Ausgänge der Platinen nach dem gleichen Prinzip, wie beim Test der FPGA-Boards, mit Frequenzgenerator und Oszilloskop überprüft werden.

Unit-Test

Softwareseitig sind unter Unit-Tests in erster Linie Funktionstests von Methoden der einzelnen Klassen zu verstehen. Diese entsprechen abgegrenzten Komponenten und können mit vorgegebenen Eingangsparametern, durch die Tests, auf ihr richtiges Verhalten beziehungsweise die Richtigkeit der Ausgangsparameter überprüft werden. Im Rahmen der Unit-Tests wurden dabei die Methoden getestet, die in jeglicher Form Berechnungen durchführen. Diese sind vorwiegend im Postprocessing und in der Tokenberechnung bei der Authentifizierung der Netzwerkkommunikation zu finden. Dabei wurden Testdaten im überschaubaren Umfang von Hand nach den definierten Rechenvorschriften erstellt und mit den Ergebnissen der entsprechenden Methoden verglichen. Einfachere Methoden, die keine Berechnungen ausführen, können mit Konsolenausgaben auf ihr richtiges Verhalten geprüft werden. Die Verifizierung der Einzelkomponenten im Schaltungsdesign der FPGAs erfolgt mit dem internen Logikanalysator. Mit ihm ist es möglich, das Zeitverhalten einzelner Signale grafisch darzustellen und so mit den Verhaltensvorhersagen der einzelnen Module zu vergleichen. Sind die Module direkt an die Ausgangspins des FPGA angeschlossen, ist auch eine Überprüfung per Oszilloskop oder externem Logikanalysator möglich. Überprüft wurden somit alle einzelnen Funktionsmodule des Schaltungsdesigns.

Integrationstest

Im Rahmen der Tests voneinander abhängiger Module galt es, das Postprocessing als Ganzes, die einzelnen Kommunikationsverbindungen samt Protokollfunktionen sowie die Versuchsaufbausteuerung mit Datenzu- und abtransport zu überprüfen. Die Kom-

plexität der Postprocessing Algorithmen liegt darin begründet, dass bei der Kommunikation zwischen Sender und Empfänger keine direkten Informationen über den Schlüssel preisgegeben werden dürfen. Liegen die Daten beider Systeme auf einem Rechner vor, lässt sich daraus mit überschaubarem Aufwand die aus der Postprocessing-Prozedur resultierenden Daten berechnen. So werden die Daten der ungültigen Transmissionen und die Transmissionsdaten bei unterschiedlicher Basenwahl verworfen und aussortiert. Die Datenbits jeder einzelnen Transmission können direkt miteinander verglichen und gegebenenfalls korrigiert werden. Zur Überprüfung der richtigen Arbeitsweise des Postprocessings können die so entstehenden Daten mit den Ergebnissen der Postprocessing-Prozedur verglichen werden. Weiterhin wird überprüft, dass die entstehenden Schlüsseldaten beider Systeme identisch sind.

Die Kommunikationsverbindungen, ob über das Netzwerk oder bei der USB-Verbindung, können mit aufeinanderfolgenden Zählerwerten und Schleifentests überprüft werden. Bei den Tests mit aufeinanderfolgenden Zählerwerten werden sich erhöhende Zahlen vom Sender erzeugt, zum Empfänger übertragen und von diesem auf die richtige Reihenfolge geprüft. Somit lässt sich feststellen, ob Daten bei der Übertragung verloren gegangen sind. Bei den Schleifentests hingegen werden definierte Daten zum Empfänger und von diesem wieder zurückgesendet. Erhält der Sender die identischen Daten, die er ausgesandt hat, zurück, war die Übertragung sowohl auf dem Hin-, als auch auf dem Rückweg erfolgreich. Wird dabei eine Menge an Daten geschickt, die mehrere Datenpakete des eigentlichen Übertragungsprotokolls umfasst, kann zudem die Nahtlosigkeit der Übertragung getestet werden. Diese Tests werden sowohl für die Netzwerkverbindung als auch für die USB-Verbindung durchgeführt. Diese Übertragungsschleifen wurden zudem bei Dauertests der USB-Verbindung verwendet, um deren Langzeitstabilität zu überprüfen. Dabei wurden dauerhaft Datenpakete versendet und bei der Rückankunft auf ihre Richtigkeit geprüft. Im gleichen Test wurde auch die dauerhafte Berechnung der Prüfsummen getestet. Bei der Überprüfung der USB-Verbindung konnte mit den Schleifentests zugleich die Funktionsweise der Versuchsaufbausteuerung überprüft werden. Dabei wurden auf der Empfängerseite die eingehenden Zufallszahlen, wie sie für die Demodulatorsteuerung benötigt werden, innerhalb des FPGA-Designs an die APD Eingänge gekoppelt. Dort wurden sie im vorgegebenen Zeitrahmen von der Versuchsaufbausteuerung eingelesen, im FIFO zwischengespeichert und anschließend zum PC übertragen. Nachdem dies auf Korrektheit geprüft war, konnte der Schleifentest auf die Sendeeinheit ausgeweitet werden. Da diese nur eine Datenzuleitung besitzt, mussten die Daten über Zusatzleitungen an die APD Eingänge der Empfangseinheit angeschlossen werden.

sen werden. Im entsprechenden Zeitrahmen der Photonentransmissionen konnten so die Daten von der Sendeeinheit zur Empfangseinheit übertragen, dort aufgenommen und zum PC übertragen werden. Die einzelnen Schritte der USB-Schleifentests sind in Abb. 3.22 dargestellt.

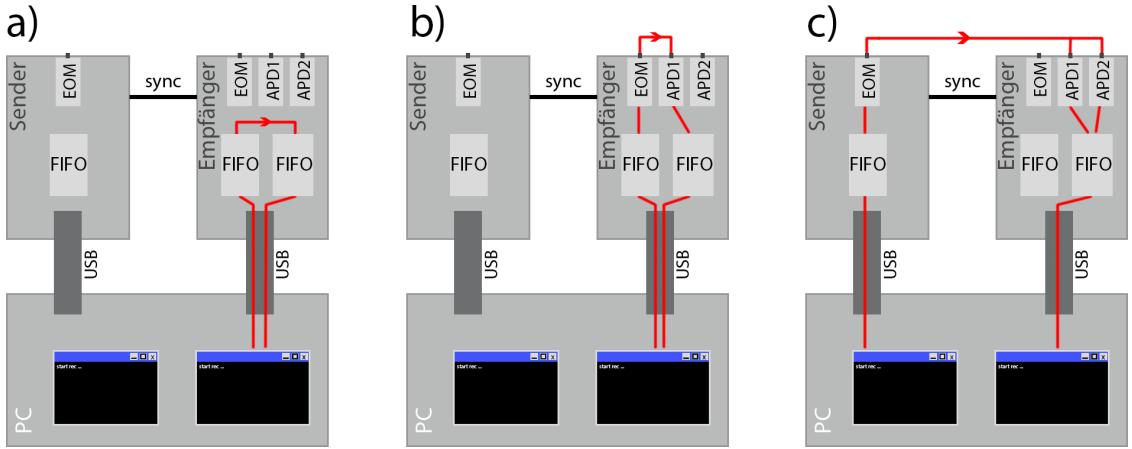


Abbildung 3.22: Konfigurationsdarstellung zum Testen der USB-Verbindungen. In a) werden die zum FPGA gesendeten Datenpakete direkt wieder zurück gesendet. Bei b) durchlaufen die Datenpakete die Versuchssteuerungslogik innerhalb des Empfänger-FPGAs. In c) werden Daten von der Sendeeinheit zur Versuchssteuerung des Empfänger-FPGAs übertragen und von diesem an den Empfänger-PC gesendet.[34]

Systemtest

Da die einzelnen Komponenten des Versuchsaufbaus durch deren Verwendung in anderen Experimenten zeitlich nur begrenzt zu Verfügung stehen und mit ihnen zudem nur eine Übertragungsfrequenz unterhalb von 1 MHz hätte realisiert werden können, ist der komplette Versuchsaufbau durch ein weiteres FPGA-System simuliert. Bei diesem FPGA-System handelt es sich um ein Entwicklungsboard von National Instruments. Es ist für den Einsatz innerhalb des PCs ausgelegt. Der Schaltungsentwurf für das FPGA erfolgt mit LabView. Da in den Laboren des Instituts die einzelnen Versuchsaufbauten überwiegend mit diesen FPGA-Systemen gesteuert werden und dementsprechend sämtliche Gerätschaften auf deren Anschlüsse ausgelegt sind, kann das Simulationsmodul leicht in das Gesamtsystem integriert werden. Intern kann für den Schaltungsentwurf eine Taktfrequenz von maximal 40 MHz verwendet werden. Das Ansprechen der Ausgänge ist weitaus langsamer. Das System kann daher maximal mit einer Transmissionsfrequenz von 2 MHz betrieben werden. Die Aufgabe des Simulationssystems ist es, auf

jeden eingehenden Impuls am Triggereingang für die Einzelphotonenquelle einen Impuls an einem der APD-Ausgänge auszugeben. An welchem der beiden APD-Ausgänge das Impulssignal ausgegeben wird, entscheidet sich anhand der Modulatoreinstellungen. Diese wird durch die jeweils 10Bit und den Triggerkanal für die DAC-Ansteuerung definiert. Da senderseitig vier und empfängerseitig zwei diskrete Modulatoreinstellungen unterschieden werden müssen, reicht es aus, senderseitig zwei Bit und empfängerseitig ein Bit statt der möglichen 10 Bit einzulesen und zu verarbeiten. Der Gesamtaufbau mit dem Simulationssystem ist in Abb. 3.23 schematisch dargestellt.

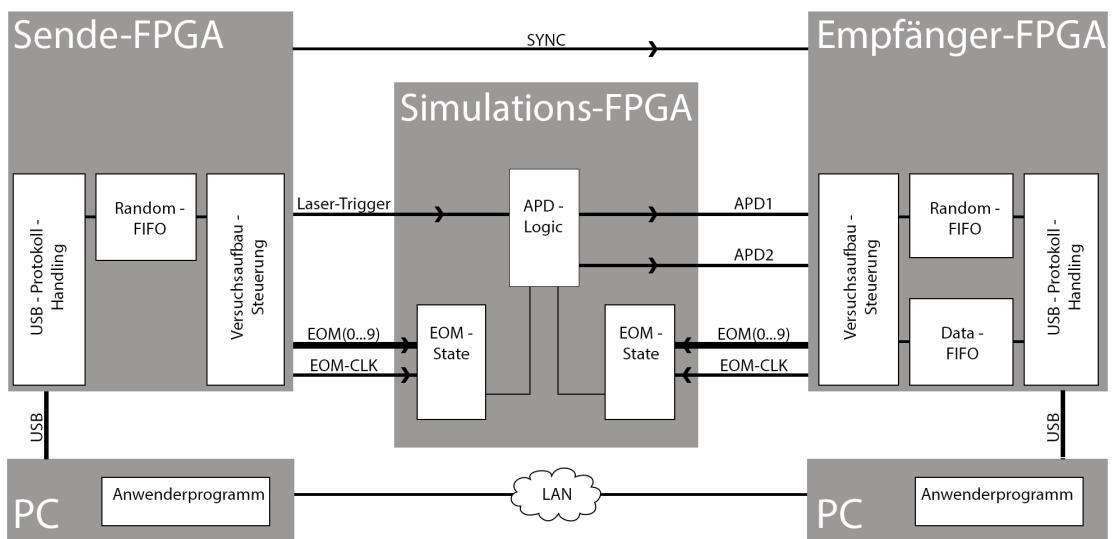


Abbildung 3.23: Gesamter Systemaufbau mit Simulations-FPGA. Das Verhalten des Versuchsaufbaus wird durch ein FPGA-System simuliert.[34]

Bis auf die Einschränkungen bei der Transmissionsfrequenz kann das entwickelte System somit in vollem Umfang getestet werden. Im Rahmen der Systemtests wurden dabei die Ergebnisse der Verteilungsprozesse beider Systeme bei verschiedenen Schlüsselgrößen und zudem bei unterschiedlichen Transmissionsfrequenzen auf Gleichheit geprüft.

4 Zusammenfassung und Ausblick

Im Projektrahmen dieser Masterarbeit ist ein System entstanden, das, wie in der Aufgabenstellung gefordert, die Versuchsaufbausteuerung der quantenkryptographischen Versuche gemäß dem BB84-Protokoll realisiert. Die der schriftlichen Ausarbeitung zugrundeliegende Entwicklungsarbeit begann mit der Zusammenstellung und Analyse der einzelnen Systemanforderungen. Auf deren Basis wurde ein Konzept für die Projektrealisierung erstellt. Nach diesem setzt sich das Gesamtsystem aus zwei getrennten und voneinander unabhängigen Teilsystemen zusammen - einer Sende- und einer Empfangseinheit. Um mit diesen Teilsystemen sowohl die Steuerung der einzelnen Hardwarekomponenten des Versuchsaufbaus, als auch die Konfiguration und Koordination des Gesamtsystems zu realisieren, war es erforderlich, Sende- und Empfangseinheit weiter zu unterteilen. Die Steuerung der Hardwarekomponenten wurde dabei durch FPGA-basierte elektronische Systeme umgesetzt. Um dabei eine Verbindung zwischen den einzelnen Hardwarekomponenten des Versuchsaufbaus und den FPGA-Boards zu ermöglichen, wurden Adapterplatinen erstellt. Für die Konfiguration und Steuerung der Gesamtanwendung kommen PCs zum Einsatz. Die Hauptaufgaben der darauf erstellten Softwareanwendung sind die Beschaffung der benötigten Zufallsdaten, die Realisierung der erforderlichen netzwerkbasierten Kommunikation zwischen Sende- und Empfangseinheit sowie die Nachverarbeitung der entstehenden Daten gemäß BB84. Zusätzlich wurde eine grafische Benutzeroberfläche erstellt, die die Konfiguration des Systems, das Starten der Transmissionsvorgänge und des Postprocessings sowie eine Analyse der resultierenden Schlüsseldaten ermöglicht. Für die Realisierung der Kommunikationsverbindung zwischen den PCs und den FPGA-Boards wurden verschiedene Möglichkeiten der Umsetzung analysiert und diskutiert. Nach der Gegenüberstellung von Vor- und Nachteilen der einzelnen Varianten, fiel die Wahl auf die Realisierung mit Hilfe von USB-Wandlermodulen. Diese ermöglichen die erforderliche Übertragungsrate und lassen sich, durch Platzierung auf den erstellten Adapterplatinen, in das System integrieren. Nach der Fertigstellung des elektronischen Systems, wurde, parallel zur Softwareentwicklung

auf den PCs, das Schaltungsdesign für die beiden FPGAs erstellt. Diese enthalten im wesentlichen Steuerlogik für das Ansprechen des Versuchsaufbaus und der Kommunikationsrealisierung mit dem USB-Modul. Sowohl für die Kommunikation über USB als auch für die Kommunikation über das Netzwerk, wurden Protokolle eingerichtet, die das Setzen von Parametern, Parameterabfragen und den Austausch von Datenpaketen ermöglichen. In den letzten Schritten des Entwicklungsprozesses wurde das erstellte System getestet. Im Rahmen von Unit-Tests wurden zu Beginn schrittweise die einzelnen voneinander unabhängigen Module getestet. Anschließend wurden die Tests auf Gruppen, voneinander abhängiger Module ausgeweitet. In einem abschließenden Testverfahren wurde die Arbeitsweise des Gesamtsystems getestet.

In einer möglichen Fortführung des Projekts, wäre es denkbar, die Programmlogik der Sendeeinheit vom PC auf den ARM-Prozessor des Zynq zu übertragen. Somit könnte auf einen Zweitrechner verzichtet werden. Sinnvoll wäre es dabei ein embedded Linux-System auf dem Zynq zu implementieren. Da die Programmbibliotheken von Qt auch auf embedded Linux-Systemen unterstützt werden und zudem der Qt-Creator ein Cross-Compiling ermöglicht, ist der Aufwand für diese Änderung überschaubar. Hinzukommen müsste allerdings Programmlogik, die die Übertragung der Schlüsseldaten auf ein anderes Speichermedium, wie beispielsweise eine externe Festplatte oder einen USB-Stick, ermöglicht. Sollte der Versuchsaufbau für die tatsächliche Schlüsselverteilung verwendet werden, so ist es erforderlich, das Sender- und das Empfängersystem jeweils mit einem eigenen Zufallszahlengenerator auszustatten. Da die Zufallsdaten momentan über einen Netzwerk-Service bezogen werden, sind sie vor möglichen Abhörern nicht geschützt.

Literatur- und Quellenverzeichnis

- [1] Kollmitzer C., Pivk M. 2010. *Applied Quantum Cryptography, Lect. Notes Phys.* 797 Seite 37 Berlin,Heidelberg: Springer Verlag.
- [2] Schumacher R. 2007. *Geschichte der Kryptographie* URL: http://www.walser-hm.ch/hans/Stud_Arbeiten/Kryptografie/Schumacher_Roger_Kryptographie.pdf Zugriff: 17.07.2013
- [3] Zeppmeisel M. 2006. *Lehrtext zur Klassischen Kryptographie* URL: <http://homepages.physik.uni-muenchen.de/~milq/quantenkryp/Kryptographie-Lehrtext.pdf> Seiten: 15..18
Zugriff: 19.07.2013
- [4] Bennett C. H., Brassard G. 1984. *Quantum cryptography: Public key distribution and coin tossing.*
- [5] Schröder T. 2013. *Integrated photonic systems for single photon generation and quantum applications-assembly of fluorescent diamond nanocrystals by novel nano-manipulation techniques* URL: <http://edoc.hu-berlin.de/dissertationen/schroeder-tim-2012-08-30/PDF/schroeder.pdf>
Zugriff: 02.09.2013
- [6] Datenblatt Laser Modulator LM 0202 Datei: LM0202_P_VIS_KDP_0,1_W3x3_V01_11.pdf (Auf beigelegter CD enthalten)
- [7] Wahl M., Leifgen M., Berlin M., Röhlicke T., Rahn H-J., and Benson O. 2011 *An ultrafast quantum random number generator with provably bounded output bias based on photon arrival time measurements.* Applied Physics Letters, 98(17):171105.

- [8] Kesel F., Bartholomä R. 2006 *Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs* Oldenbourg Wissenschaftsverlag
- [9] Lehmann G., Wunder B., Selz M. 1998 *Schaltungsdesign mit VHDL* Franzis Verlag
- [10] Stroustrup B. 1997 *The C++ Programming Language* Addison-Wesley Longman, Amsterdam Verlag
- [11] Summerfield M., Blanchette J. 2007 *C++ GUI-Programmierung Mit Qt 4: Die Offizielle Einführung* Pearson Deutschland
- [12] <http://www.mathe.tu-freiberg.de/~hebisch/cafe/kryptographie/Skytale.png>,
Zugriff: 09.05.2013
- [13] <http://www.theinquirer.net/IMG/233/86233/standard-enigma-540-540x874.jpg>,
Zugriff: 09.05.2013
- [14] <http://arnoldsat.com/polarizations.jpg>,
Zugriff: 14.08.2013
- [15] Foto: Robert Riemann, Institut für Physik, Humboldt-Universität zu Berlin / nanooptics
- [16] <http://www.jameshedberg.com/img/samples/nv-center-1.png>,
Zugriff: 13.08.2013
- [17] http://www.qioptiq-shop.com/out/Graphics/de/00111671_0.jpg,
Zugriff: 15.05.2013
- [18] Datenblatt: SPCMAQR.pdf (Auf beigelegter CD enthalten)
- [19] Datenblatt: <http://picoquant.com/images/uploads/downloads/pqrng150.pdf>,
Zugriff: 14.09.2013
- [20] <http://media.digikey.com/photos/AlteraPhotos/EP3C5E144I7N.JPG>,
Zugriff: 11.06.2013

- [21] <http://tautec-electronics.de/catalog/images/A54SX08ATQ144.jpg>,
Zugriff: 11.06.2013
- [22] <http://images.vogel.de/vogelonline/bdb/107400/107456/35.jpg>,
Zugriff: 11.06.2013
- [23] <http://book.huihoo.com/design-of-vlsi-systems/ch01/Figure-1.14.gif>,
Zugriff: 15.08.2013
- [24] <http://wwwdsa.uqac.ca/~daudet/Cours/Vlsi/DOCUMENTS/repertoire435/Livre-MJS-Smith/Book/CH06/CH06-21.gif>,
Zugriff: 15.08.2013
- [25] <http://press.xilinx.com/download/SP605-board.jpg>,
Zugriff: 27.04.2013
- [26] SP605 Hardware User Guide: http://www.xilinx.com/support/documentation/boards_and_kits/ug526.pdf,
Zugriff: 14.09.2013
- [27] http://maxim.eefocus.com/data/myspace/64/321951/bbs/2012-12-07/1354885545_182feb25.jpg,
Zugriff: 27.04.2013
- [28] http://www.bdti.com/sites/default/files/insidedsp/articlepix/201103/zync_blockdiagram.jpg,
Zugriff: 27.04.2013
- [29] [http://media.digikey.com/Photos/FTDI\(Future Tech Devices\)/MFG_UM232H.jpg](http://media.digikey.com/Photos/FTDI(Future Tech Devices)/MFG_UM232H.jpg),
Zugriff: 27.04.2013
- [30] http://www.conrad.de/medias/global/ce/7000_7999/7300/7300/7303/730387_BB_00_FB.EPS_1000.jpg,
Zugriff: 24.08.2013
- [31] <http://www.fpgadeveloper.com/en/general/fmc-connector.jpg>,
Zugriff: 24.08.2013

- [32] <http://de.farnell.com/productimages/farnell/standard/142825507-40.jpg>,
Zugriff: 24.08.2013
- [33] http://www.imn.htwk-leipzig.de/~medocpro/buecher/sedge1/pic/abb40_4.gif,
Zugriff: 14.09.2013
- [34] Eigene Darstellung

Anhang

Die angehängten Dokumente befinden sich auf der beigelegten CD.