LEOPOLD-FRANZENS-UNIVERSITÄT INNSBRUCK

MASTERARBEIT

Dynamische Kontrolle von Laserimpulsen zur Quanteninformationsverarbeitung

---

ein Teil der Voraussetzung
zur Erlangung des akademischen Grades

MASTER OF SCIENCE

durchgeführt am Institut für Experimentalphysik unter Betreuung
von Universitätsprofessor Dr. Rainer Blatt

präsentiert von

MICHAEL METH

2. Oktober 2017

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche gekennzeichnet.

Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit oder Dissertation eingereicht.

Unterschrift:

Datum:

Kurzfassung

In einem optischen Ionenfallen-Quantencomputer werden Gatteroperationen mit Laserlichtimpulsen erzeugt. Diese Masterarbeit stellt ein Pulsgeneratorsystem vor, welches Radiofrequenzsignale erzeugt, die in Frequenz, Phase und Amplitude kontrolliert werden. Diese Signale treiben einen akusto-optischen Modulator, der die Lichtpulse erzeugt. Die Dynamik der Modulatoren wird im Rahmen dieser Masterarbeit experimentell untersucht. Zur Bestimmung der Entwicklung der Pulsflächen und Phasen wird ein lineares Modell als erste Näherung verwendet. Anhand einer numerischen Simulation werden die Auswirkungen der Fluktuationen der Pulsflächen und Phasen auf die Güte der Gatteroperationen untersucht.

Abstract

In an optical ion trap quantum computer, quantum gate operations are realised using laser pulses. In the scope of this thesis a new pulse generator system is presented, generating radiofrequency signals which can be controlled in frequency, phase and amplitude. These signals are used to drive acousto-optic modulators, which create the actual light pulses. This thesis presents the experimental investigation of the dynamic effects of those modulators. A linear model is used to approximate the drifts of pulse area and phase up to first order. The influence of the fluctuations of pulse areas and phases on gate operations is estimated by means of numerical simulations.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Seit Anfang des 20. Jahrhunderts ist die Quantenphysik ein breites Forschungsfeld. Durch die Beschreibung der Wärmestrahlung eines schwarzen Körpers von Planck um 1900, der Erklärung des photoelektrischen Effekts durch Einstein und der Entwicklung des Bohrschen Atommodells konnten einige bis dato unerforschte Phänomene erklärt werden. Manche Modelle erwiesen sich als unvollständig, sodass diese stetig durch neue Theorien erweitert wurden. Dies mündete schließlich um 1925 in der modernen Quantenmechanik, welche unter Anderem die Grundlagen für Halbleiter- und Lasertechnologien und damit der modernen Informatik liefert. Während erste einfache Berechnungen durch Systeme aus elektromechanischen Schaltern aufgebaut wurden, ermöglichte erst die Entwicklung des Transistors der breiten Masse den Zugang zu Computern, Kommunikations- und Unterhaltungselektronik. Quantenmechanische Eigenschaften wie Kohärenz, Superposition und Verschränkung werden in Halbleitern jedoch nicht bewusst kontrolliert. Eine Kontrolle über diese Effekte eröffnet ein neues Forschungsfeld, das den technischen Fortschritt in Zukunft weiter vorantreiben wird. Ein vielversprechendes Gebiet ist die Quanteninformationsverarbeitung, in der ein im Vergleich zu einem klassischen Computer leistungsfähigerer Quantencomputer erforscht wird [1]. Dieser nützt quantenmechanische Effekte, um durch geeignete Algorithmen komplexe Berechnungen effizienter und in kürzerer Zeit auszuführen [2, 3]. Die Informationsträger des Quantencomputers werden als Quantenbits, kurz Qubits, bezeichnet, die das Analogon zum klassischen Bit als kleinste logische Einheit darstellen. Diese werden von Quantengattern manipuliert, welche auf ein, zwei oder mehrere Qubits wirken und so Rechenoperationen ermöglichen. Die Implementierung von Qubits und Gattern ist schwierig und beschränkt die Umsetzung eines Quantencomputers auf einige wenige Systeme, welche gewissen Kriterien genügen müssen [4]. Gespeicherte Ionen in einer Ionenfalle stellen einen geeigneten Kandidaten dar. In Innsbruck werden Kalzium-Ionen in einer linearen Paul-Falle gefangen und gespeichert; die Zustandsmanipulation erfolgt mittels gepulstem kohärentem Licht, welches von Lasern produziert wird. Die Erzeugung dieser Lichtimpulse und die Kontrolle ihrer Kenngrößen Frequenz, Phase und Amplitude stellt einen limitierenden Faktor bei der Realisierung qualitativ hochwertiger Quantengatter dar. Neben dem Lasersystem [5] sind dynamische Effekte der an der Impulserzeugung beteiligten akusto-optischen Modulatoren (AOM) zu berücksichtigen. Ein AOM ermöglicht es, die Frequenz des Lichts zu verschieben, die Phasenlage zu beeinflussen und die Amplitude eines Lichtimpulses zu kontrollieren.

Diese Arbeit befasst sich mit der Charakterisierung der dynamischen Effekte, die in akusto-optischen Modulatoren auftreten. Zur Ansteuerung der AOMs werden Radiofrequenzsignale erzeugt, welche in Frequenz, Phase und Amplitude genau kontrollierbar sind. Als Signalquelle wird ein neuer Impulsgenerator M-ActION verwendet, welcher das bisherige System PulseBox ersetzen soll [6, 7]. M-ActION erlaubt das Erzeugen von Impulssequenzen, die an den AOM angelegt werden und sich auf einen Laserstrahl auswirken. Amplitude, Frequenz und Phase der erzeugten Lichtimpulse werden gemessen und die Dynamik der Modulatoren anhand dieser Größen abgeschätzt.

Die theoretischen Grundlagen der Wechselwirkung eines Ions mit dem Lichtfeld werden in Kapitel 2 beschrieben und die Auswirkungen der Impulsfläche und Phase eines Lichtimpulses auf das Ion diskutiert. Hier wird deutlich, warum die Kontrolle von Frequenz, Phase und Amplitude eine enorme Rolle bei der Implementierung qualitativ hochwertiger Quantengatter spielt. Die Auswirkungen von Fluktuationen dieser Größen auf Gatteroperationen werden gezeigt. In Kapitel 3 wird die Erzeugung von Laserimpulsen erklärt. Die Bedeutung einer gemeinsamen Zeitbasis der an der Impulserzeugung beteiligten Elektronik wird dargelegt. Da in dieser Arbeit das neue Impulsgeneratorsystem M-ActION erstmalig eingesetzt wird, ist die System- und Programmstruktur von M-ActION kurz beschrieben und in Anhang A genauer diskutiert. Die Programmierung von Radiofrequenzpulsen und -sequenzen wird mit kompakten Beispielen präsentiert. Kapitel 4 beschreibt den experimentellen Aufbau und die Vorgehensweise bei der Datenaufzeichnung. Da große Datenmengen aufgezeichnet werden, liegt besonderes Augenmerk auf der Datenanalyse. In Kapitel 5 wird die Dynamik dreier verschiedener Modulatoren anhand der gewonnenen Daten diskutiert. Die Abweichung von Impulsfläche und -phase von einem idealen Impuls wird für Einzelimpulse sowie entlang einer Sequenz aus vielen Impulsen bestimmt und linear approximiert. Eine Abhängigkeit zur Impulsdauer sowie zum Abstand zweier aufeinanderfolgender Lichtimpulse wird näher untersucht. Die Modulatoren werden untereinander verglichen, um einen geeigneten Typ für den Einsatz in zukünftigen Experimenten auszuwählen. Die Auswirkung jedes Modulators auf die Zustandsmanipulation eines Qubits wird in Kapitel 6 anhand einer numerischen Simulation des Quantencomputers diskutiert. Es wird eine Gatterstichprobe berechnet, bei der die Ergebnisse aus Kapitel 5 verwendet werden, um ein Fehlermodell zu implementieren, welches das Verhalten der AOM möglichst akkurat wiedergibt. Ein Vergleich mit experimentellen Daten aus [5] wird vorgenommen. Die Arbeit schließt mit einer Zusammenfassung in Kapitel 7, welche auch die nächsten Schritte zur Implementierung von M-ActION im Laborbetrieb und die Überwindung der aktuellen Limitierungen des Systems beinhaltet.

Kapitel 2

Grundlagen der Quanteninformationsverarbeitung

2.1 Das Qubit

Die Grundlage eines klassischen Computers bildet das Bit als kleinste logische Einheit. Es kennt zwei Zustände, die durch 0 und 1 beschrieben werden; das Bit ist entweder gesetzt – logisch 1 – oder nicht – logisch 0. Ein Bit lässt sich durch elektronische Bauelemente aus einem Transistor und einem Kondensator realisieren. Die Ladung des Kondensators bestimmt den Zustand, wenn das Bit abgefragt wird. Ein Quantencomputer dagegen arbeitet mit einem Quantensystem, das neben den klassischen Zuständen 0 und 1 auch Superpositionen annehmen kann. In diesem Abschnitt wird ein solches Quantenbit (*Qubit*) auf Basis eines in einer Ionenfalle gefangenen Kalzium-Ions beschrieben.

Ein Qubit ist ein zweidimensionaler Hilbertraum[1] $\mathcal{H}^2$, dessen orthogonale Basiszustände mit $|0\rangle$ und $|1\rangle$ bezeichnet werden. Ein allgemeiner Zustand $|\psi\rangle$, in dem sich das Qubit befindet, wird für $\alpha,\ \beta \in \mathbb{C}$ durch

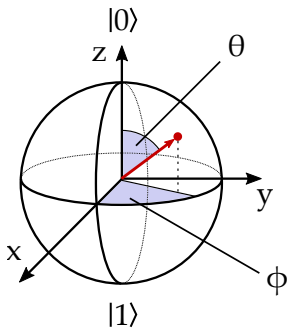
$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \qquad \text{mit} \quad |\alpha|^2 + |\beta|^2 = 1 \tag{2.1}$$

beschrieben. Mit der Bedingung $|\alpha|^2 + |\beta|^2 = 1$ ist die Norm von $|\psi\rangle$ auf Eins beschränkt – dies ist eine Konsequenz der bornschen Interpretation der Quantenmechanik [8]. Die Frage, ob sich das Qubit im einem der Basiszustände $|0\rangle$ oder $|1\rangle$ befindet, stellt eine Messung $\hat{M}$ an einem Quantensystem dar. Das Ergebnis ist durch Gl. 2.2 gegeben und beschreibt die Wahrscheinlichkeit, den zu messenden Zustand im System vorzufinden. Nach der Ausführung der Messung kollabiert das Qubit in den gemessenen Zustand $|\phi\rangle$.

$$p_M = \langle\psi|\hat{M}^\dagger\hat{M}|\psi\rangle \longrightarrow |\phi\rangle = \frac{1}{\sqrt{p_M}}\hat{M}\,|\psi\rangle \tag{2.2}$$

Durch die Normierung von $|\psi\rangle$ lässt sich der Zustand als Punkt auf der Oberfläche der Einheitskugel oder Blochkugel darstellen. Die Basiszustände werden als Nord- und Südpol festgelegt und bilden die z-Achse des in der Kugel eingeschriebenen karthesischen Koordinatensystems, das in Abb. 2.1 dargestellt ist. Die Koordinaten von $|\psi\rangle$ sind bezüglich des Polarwinkels θ und dem Azimut ϕ durch Gl. 2.3 gegeben.

1 Vollständiger normierter Vektorraum über einem Körper, hier $\mathbb{C}^2$.



$$|\psi\rangle = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix} = \\ = \cos\frac{\theta}{2}e^{-i\Phi/2}|0\rangle + \sin\frac{\theta}{2}e^{i\Phi/2}|1\rangle \tag{2.3}$$

ABBILDUNG 2.1: Darstellung eines Zustand $|\psi\rangle$ eines Qubits auf der Blochkugel. Polarwinkel θ und Azimut ϕ beschreiben $|\psi\rangle$ bis auf eine globale, nicht messbare Phase vollständig. Die Pole werden durch die klassischen Zustände $|0\rangle$ und $|1\rangle$ gebildet.

2.1.1 Gatteroperationen auf Qubits

Jede Berechnung in einem klassischen Computer wird durch eine Verkettung von binären Logikoperationen ausgeführt. Das quantenmechanische Analogon hierzu sind Quantengatter, die auf ein Qubit angewandt werden und eine Manipulation des Zustands bewirken. Für in einer Ionenfalle gefangene Ionen werden Gatter durch Wechselwirkung mit einem externen Strahlungsfeld in Form von gepulstem Laserlicht realisiert. Man unterscheidet zwischen lokalen, globalen und verschränkenden Operationen. Lokale Gatter werden auf einem einzelnen Qubit ausgeführt (*Ein-Qubit-Gatter*), während globale auf alle gefangenen Ionen wirken; Verschränkungen verknüpfen mindestens zwei Qubits und liefern eine besondere Art von Zuständen, die nicht aus dem Produkt der Qubitzustände erzeugbar sind.

Die Beschreibung der Qubit-Gatter erfolgt durch unitäre Abbildungen $\hat{U}$, welche die in Gl. 2.4 angeführten Bedingungen erfüllen; die zu $\hat{U}$ Adjungierte wird durch $\hat{U}^\dagger$ beschrieben. Eine unitäre Transformation ist nach $\hat{U}\hat{U}^\dagger = \hat{U}^\dagger\hat{U} = \mathbb{1}$ umkehrbar und verändert die Norm eines Zustands $|\psi\rangle$ nicht.

$$\hat{U}\hat{U}^\dagger = \hat{U}^\dagger\hat{U} = \mathbb{1} \quad \text{und} \quad |\,|\psi\rangle\,|^2 = |\hat{U}\,|\psi\rangle\,|^2 \tag{2.4}$$

Eine Abbildung $\hat{X}$, die diese Eigenschaften erfüllt, ist in Gl. 2.5 angeführt. Die Umkehrbarkeit ist direkt ersichtlich, da eine Hintereinanderausführung die Wirkung von $\hat{X}$ aufhebt, sodass $\hat{X}\hat{X}\,|\psi\rangle = |\psi\rangle$. Außerdem ist $\langle\psi|\psi\rangle = \langle\psi|\hat{X}^\dagger\hat{X}|\psi\rangle = \langle\psi|\hat{X}\hat{X}|\psi\rangle$, womit auch die Norm erhalten bleibt.

$$\hat{X} : \mathcal{H}_2 \to \mathcal{H}_2,\ |0\rangle \mapsto |1\rangle \quad \text{und} \quad |1\rangle \mapsto |0\rangle \tag{2.5}$$

Nach Wahl einer Basis von $\mathcal{H}_2$ können die Abbildungen durch Matrizen repräsentiert werden. Üblich ist die Darstellung in der z-Basis, für die $|0\rangle = (1,0)$ und $|1\rangle = (0,1)$ gilt. Mit dieser Wahl wird $\hat{X}$ durch die in Gl. 2.6 angegebene unitäre 2×2-Matrix beschrieben.

$$\hat{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.6}$$

Die Matrix zur Abbildung $\hat{X}$ ist eine der in Tab. 2.1 dargestellten Pauli-Matrizen, die zusammen mit der Identität $\mathbb{1}$ die Basis der komplexen unitären 2×2-Matrizen aufspannt. Sie werden als σ_j mit $j \in \{x, y, z\}$ bezeichnet und erlauben durch Bildung des Matrixexponentials $e^{-i\theta\sigma_j/2}$ die Beschreibung von Drehungen der Blochkugel um einen Winkel θ um die von j definierte Achse. In den Grafiken sind diese Rotationen für $\theta = \pi/2$ gezeigt – der rot angedeutete Ausgangszustand und wird in den blauen Endzustand überführt.

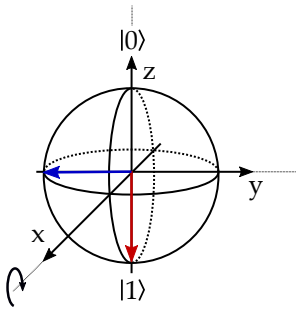
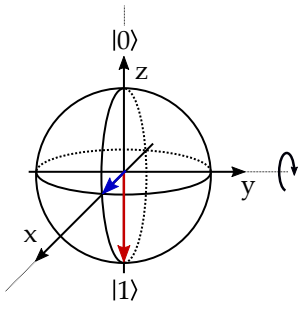
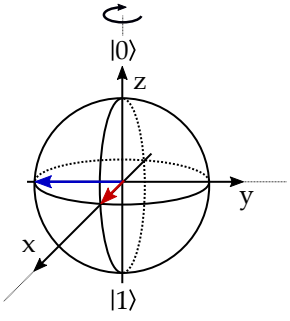
$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
		

TABELLE 2.1: Darstellung der Pauli-Matrizen und ihre Wirkung auf einen Zustand. Startzustand in rot wird in durch die Drehung $\exp(i\sigma_j\theta)$ um die Achse i in den blauen Zustand überführt. Der Drehwinkel beträgt jeweils $\theta = \pi/2$.

Eine Drehung R um eine von einem Vektor $\vec{n} = (n_x, n_y, n_z)$ mit $|\vec{n}| = 1$ definierte Achse lässt sich mit $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$ durch

$$R_{\vec{n}}(\theta) = e^{-i\theta\vec{n}\vec{\sigma}/2} = \cos\frac{\theta}{2}\mathbb{1} - i\sin\frac{\theta}{2}\left(n_x\sigma_x + n_y\sigma_y + n_z\sigma_z\right) \tag{2.7}$$

beschreiben. Ein beliebiger Zustand $|\phi\rangle$ lässt sich durch $R_{\vec{n}}(\theta)$ in jeden anderen Punkt auf der Blochkugel überführen. Bis auf einen globalen Phasenfaktor $e^{i\gamma}$ deckt $R_{\vec{n}}(\theta)$ den vollständigen Hilbertraum $\mathcal{H}_2$ ab. Daher lässt sich jedes Ein-Qubit-Gatter durch

$$\hat{U} = e^{i\gamma}R_{\vec{n}}(\theta) \tag{2.8}$$

ausdrücken. Mit $\vec{n} = (\cos(\phi), \sin(\phi), 0)$ wird $R_{\vec{n}}(\theta)$ auf die x/y-Ebene eingeschränkt:

$$R_{\vec{n}}(\theta) \longrightarrow R(\theta, \phi) = \cos\frac{\theta}{2}\mathbb{1} + i\sin\frac{\theta}{2}\left(\cos\phi\sigma_x + \sin\phi\sigma_y\right). \tag{2.9}$$

Für die Pauli-Matrizen gilt die Kommutatorrelation Gl. 2.10 mit dem vollständig antisymmetrischen Tensor ϵ_{jkl} (*Levi-Civita*-Symbol) für $j, k, l \in \{1, 2, 3\}$. Damit lässt sich je eine der in Tab. 2.1 angeführten Matrizen durch die zwei anderen ausdrücken; so ist beispielsweise $\sigma_z = -i\sigma_x\sigma_y$.

$$[\sigma_j, \sigma_k] = \sigma_j\sigma_k - \sigma_k\sigma_j = 2i\epsilon_{jkl}\sigma_l \tag{2.10}$$

Die Hintereinanderausführung zweier Rotationen $R_{\vec{n}}(\theta)$ und $R_{\vec{n}'}(\theta')$ mit den Drehachsen $\vec{n}$, $\vec{n}'$ und $|\vec{n}| = |\vec{n}'| = 1$ ist ebenfalls eine Rotation [9]. Gatter, welche den Zustand des Qubits um die z-Achse drehen, lassen sich mit Gl. 2.10 als Hintereinanderausführung von Rotationen in der x/y-Ebene beschreiben.

2.1.2 Das Kalzium-Ion als Qubit

Kalzium $^{40}\mathrm{Ca}$ ist ein chemisches Element der zweiten Hauptgruppe des Periodensystems und besitzt zwei Elektronen in der äußersten Schale. Durch Ionisation wird dem Atom eines dieser Valenzelektronen entrissen, sodass das wasserstoffähnliche $^{40}\mathrm{Ca}^{+}$ entsteht. Es erfüllt die folgenden zwei Kriterien und eignet sich daher gut, um darauf in einem Ionenfallen-Quantencomputer ein Qubit zu implementieren [10].

(1) Die zum Kühlen, zur Zustandsmanipulation und -detektion notwendigen Übergänge sind technisch erreichbar.

(2) Mit einer Zustandsmanipulation auf einer Zeitskala von $\approx 100\,\mu\mathrm{s}$ ist die Lebensdauer des Qubits mit $> 1\,\mathrm{s}$ ausreichend lang.

Ein vereinfachtes Termschema des Ions ist in Abb. 2.2 dargestellt. Das Qubit wird durch die Niveaus $|1\rangle = 4^2\mathrm{S}_{1/2}(m = -1/2)$ und dem metastabilen $|0\rangle = 3^2\mathrm{D}_{5/2}$ gebildet, dessen Lebensdauer mit $\tau = 1,168(7)\ \mathrm{s}$ [11, 12] gegeben ist. Diese sind über eine Quadrupol-Übergang gekoppelt, der mit einem Laser bei einer Wellenlänge von $729\,\mathrm{nm}$ getrieben wird.

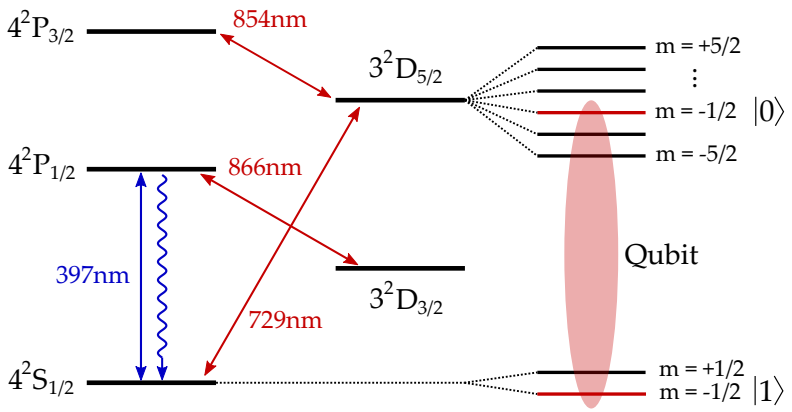


ABBILDUNG 2.2: Reduziertes Termschema von $^{40}\mathrm{Ca}^{+}$ – das Qubit ist im farblich hinterlegten Quadrupol-Übergang $4^2S_{1/2}(m = -1/2) \longleftrightarrow |0\rangle = 3^2D_{5/2}(m = -1/2)$ codiert. Die Zustandsdetektion erfolgt über das Niveau kurzlebige $4^2P_{1/2}$, welches nach $4^2S_{1/2}$ und $3^2D_{3/2}$ spontan zerfällt; ein zusätzlicher Laser mit $866\,\mathrm{nm}$ Wellenlänge erhöht die Detektionseffizienz. Das Qubit wird über $4^2P_{3/2}$ entleert.

Sowohl der Kühlprozess als auch die Zustandsdetektion erfolgen über den Übergang $4^2\mathrm{S}_{1/2} \longleftrightarrow 4^2\mathrm{P}_{1/2}$. Die Population in $4^2\mathrm{S}$ wird mit einem Laser der Wellenlänge $397\,\mathrm{nm}$ in den kurzlebigen Zustand $4^2\mathrm{P}_{1/2}$ (Lebensdauer $\tau \approx 7,1\ \mathrm{ns}$ [13]) überführt und das emittierte Fluoreszenzlicht detektiert. Da das Niveau $4^2\mathrm{P}_{1/2}$ schwach mit dem metastabilen $3^2D_{3/2}$ koppelt, wird ein zusätzlicher Laser bei $866\,\mathrm{nm}$ eingestrahlt und so die Detektionseffizienz erhöht.

Für das Rückpumpen der Population aus $|0\rangle$ wird der Übergang $3^2\mathrm{D}_{5/2} \longleftrightarrow 4^2\mathrm{P}_{3/2}$ getrieben. Nun zerfällt $4^2P_{3/2}$ nach $\approx 6.9\ \mathrm{ns}$ in $4^2S_{1/2}$, $3^2D_{5/2}$ und $3^2D_{3/2}$ [14], sodass auch hier neben dem Rückpump-Laser bei $854\,\mathrm{nm}$ Wellenlänge zusätzliches $866\,\mathrm{nm}$-Licht eingestrahlt wird, um $|0\rangle$ zuverlässig nach $|1\rangle$ zu überführen.

2.1.3 Gatterstichproben

Wie auch in einem klassischen Computer treten bei der Ausführung von Quantengattern Fehler auf, die es zu bestimmen und zu minimieren gilt. Eine Methode, die Fehlerrate einer Sequenz aus Gattern zu bestimmen, ist die Charakterisierung des vollständigen Prozesses durch Erzeugung $n \in \mathbb{N}$ bekannter Eingangszustände $|\psi\rangle_{\text{in}}$ und Messung der Eigenzustände verschiedener Basen [15] – das Verfahren ist in Abb. 2.3 schematisch dargestellt. Die Messung der Basiszustände ist für alle $|\psi\rangle_{\text{in}}$ vorzunehmen, sodass der Eingangszustand oft erzeugt werden muss. Diese Zustandspräparation ist fehlerbehaftet und führt für jeden $|\psi\rangle_{\text{in}}$ zu einem konstanten Fehler. Nach Ausführung der Sequenz liegt ein neuer Zustand vor, der neben dem Sequenzfehler jenen der Zustandspräparation enthält. Auch die Messung der Eigenzustände ist fehlerbehaftet – das Resultat beinhaltet Fehler der Präparation, der Messung und des eigentlichen Prozesses. Es ist nicht klar, welchen Anteil die Initialisierung und die Zustandsdetektion zum Gesamtfehler leisten – man spricht von SPAM (state preparation and measurement error). Rückschlüsse auf die Fehlerrate eines einzelnen an der Sequenz beteiligten Gatters sind auf diese Weise nicht exakt möglich.

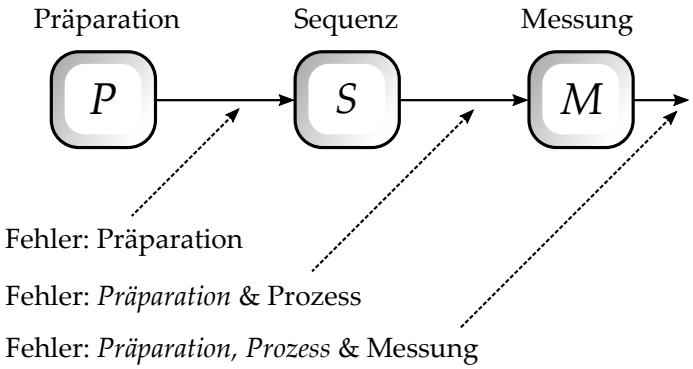


ABBILDUNG 2.3: Schematische Darstellung einer Prozesstomographie. Sowohl die Präparation als auch die Detektion sind fehlerbehaftet, sodass diese einen zu charakterisierenden Prozess verfälschen. Die Präparations- und Messfehler (SPAM, state preparation and measurement error) können nicht vom Fehler der Sequenz unterschieden werden.

Das Verfahren der Gatterstichproben (*RB*, *randomized benchmarking*) umgeht dieses Problem durch Randomisierung des Prozesses [16] [17]. Es werden zufällige Sequenzen unterschiedlicher Längen l aus Clifford-Gattern Cl erzeugt. Diese sind aus Kombinationen der physikalischen Gatter $\pm\alpha \cdot \sigma_{x,y,z}$ und $\pm\alpha \cdot \mathbb{1}$ mit $\alpha = \{1, i\}$ aufgebaut – pro Cl werden im Mittel $2,25$ Gatter implementiert [5]. Ausgehend von einem bekannten Startzustand $|\psi\rangle_{\text{in}}$ wird unter Annahme einer nicht-fehlerbehafteten Sequenz der resultierende Zustand $|\phi\rangle$ berechnet. Daraus wird ein Umkehrgatter Cl^{-1} berechnet, das $|\phi\rangle$ in einen gewählten Ausgangszustand $|\psi\rangle_{\text{out}}$ überführt. Mit der Hintereinanderausführung $\circ$ lässt sich der Prozess aus Sequenz und Umkehrung durch Gl. 2.11 beschreiben.

$$Cl^{-1} \circ \left(\prod_{i=1}^{l} Cl_i \right) = Cl^{-1} \circ (Cl_l \circ \ldots \circ Cl_1) \tag{2.11}$$

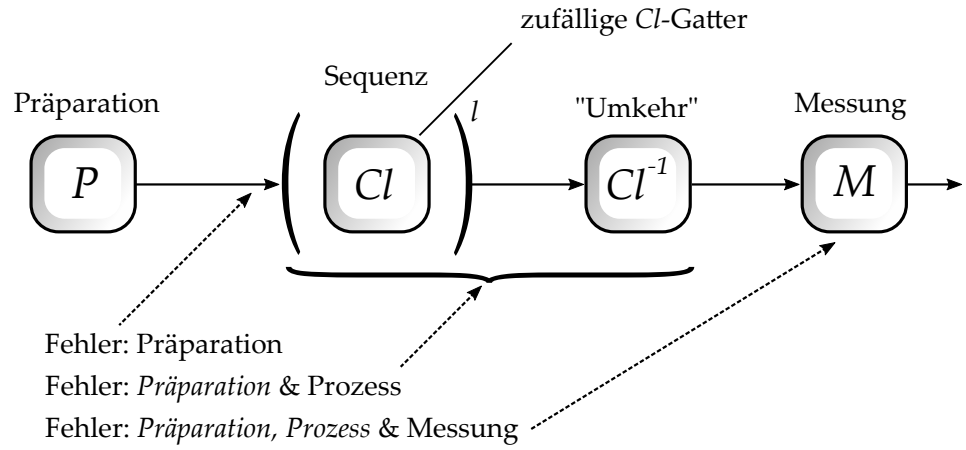


ABBILDUNG 2.4: Schematische Darstellung einer Gatterstichprobe – alle Komponenten des Experiments sind fehlerbehaftet. Nach der Präparation P eines bekannten Eingangszustands wird eine Sequenz aus l zufälligen Clifford-Gattern Cl und einem Umkehrgatter Cl^{-1} angewandt. Cl^{-1} wird aus der Annahme einer idealen Sequenz berechnet und so gewählt, dass ein gewünschter Ausgangszustand vorliegt. Dieser wird als Resultat der fehlerbehafteten Sequenz durch M gemessen.

Im Experiment sind Präparation P, Sequenz, Umkehrung und Messung M fehlerbehaftet – M bildet die Projektion des resultierenden Zustands auf das Ideal $|\psi\rangle_{\text{out}}$. Für jedes l wird eine Statistik durch Ausführung vieler randomisierter Sequenzen gebildet und daraus die mittlere Anregung P gebildet. Diese Methode wird für verschiedene l wiederholt und P als Funktion von l aufgetragen. Unter Annahme von Depolarisationsrauschen [18] wird der Verlauf $P(l)$ durch das Potenzgesetz

$$P(l) = A + B \cdot p^l \tag{2.12}$$

mit den Konstanten A, B und der Anregung p beschrieben[2]. A und B enthalten die gesamten Fehler der Zustandspräparation, der Messung und des Umkehrgatters [20] – diese ergeben sich unter Annahme eines SPAM-freien Systems und eines perfekten Cl^{-1} zu $A = B = 0,5$[3]. Der Gatterfehler (Infidelity) $\mathcal{I}$[4] ist durch Gl. 2.13 gegeben [19].

$$\mathcal{I} = \frac{1-p}{2} \tag{2.13}$$

Die Infidelity wird für die Clifford-Gatter als $\mathcal{I}(Cl)$ aus den Ergebnissen der randomisierten Sequenzen bestimmt. Für die an den Cl beteiligten physikalischen Gatter Ph gilt mit $2,25$ Gattern pro Cl die Beziehung $\mathcal{I}(Ph) = I(Cl)^{1/2,25}$ [5].

2 Es wird angenommen, dass die Fehler der einzelnen Gatter voneinander unabhängig sind [19].

3 Da der Zustand perfekt präpariert ist und die Drehung in den gewählten Ausgangszustand durch Cl^{-1} ohne Fehler erfolgt, wird die Messung von P für $l = 0$ immer 1 ergeben $\longrightarrow A = B = 1$ erfüllt diese Bedingung.

4 Die Gattergüte $\mathcal{F}$ (Fidelity) ist durch $\mathcal{F} = 1 - \mathcal{I}$ gegeben.

2.2 Licht-Materie Wechselwirkung

In diesem Abschnitt wird ein Zwei-Niveau-System beschrieben, das sich in einem externen, klassischen Lichtfeld befindet. Die Betrachtung eines freien Teilchens erlaubt eine einfache Ableitung der Wechselwirkung und stellt eine Verbindung zwischen dem in Abschnitt 2.1 diskutierten Qubit und den Größen des Laserlichts her. Die Dynamik eines gespeicherten Ions in einer Ionenfalle wird durch diese Beschreibung jedoch nur unvollständig wiedergegeben. Für eine vollständige Diskussion wird auf [21] verwiesen.

Ein freies Teilchen der Masse m befindet sich in einem zeitlich oszillierenden Feld der in Gl. 2.14 gegebenen Form, mit dem es in Wechselwirkung tritt.

$$\vec{E}(t) = E_0 \vec{\epsilon} e^{-i\omega_L t} + c.c. \tag{2.14}$$

Hier ist $\vec{\epsilon}$ der komplexwertige Polarisationsvektor und ω_L die Kreisfrequenz des eingestrahlten Lichts. Die räumliche Ausbreitung wird vorerst nicht betrachtet.

Anmerkung: die folgende Diskussion beschränkt sich auf Dipolübergänge, da dies eine einfache Beschreibung erlaubt. Für das in einem Kalzium-Ion auf dem Übergang $4^2\mathrm{S}_{1/2} \longleftrightarrow 3^2\mathrm{D}_{5/2}$ *kodierte Qubit aus Abschnitt 2.1.2 handelt es sich um einem Quadrupol-Übergang, der den elektrischen Quadrupoltensor*[5] *enthält und zu einer Änderung eines Parameter führt. Das in diesem Abschnitt abgeleitete Resultat ist jedoch ähnlich und lässt sich auf diese Übergänge übertragen.*

Die Dynamik des Teilchens in diesem Feld wird durch den Hamilton-Operator $\hat{H}$

$$\hat{H} = \hat{H}_0 + \hat{H}_1(t) = \frac{\hat{P}^2}{2m} - \vec{d}\vec{E}(t) \tag{2.15}$$

in Gl. 2.15 beschrieben. Er besteht aus zwei Komponenten – der ungestörte, zeitunabhängige Term $\hat{H}_0$ liefert die kinetische Energie, während $\hat{H}_1$ mit der Kopplungsstärke $\vec{d}$ die Wechselwirkung mit dem Lichtfeld beschreibt.

Das in Abschnitt 2.1 beschriebene Qubit wird aus den zwei Energieniveaus $|0\rangle$ und $|1\rangle$ gebildet. Ein allgemeiner Zustand $|\psi(t)\rangle$ aus diesem zweidimensionalen Hilbertraum $\mathcal{H}^2$ wird durch Gl. 2.16 dargestellt[6]. Da der Hamilton-Operator zeitabhängig ist, sind auch die Koeffizienten $c_0(t)$ und $c_1(t)$ zeitlich veränderliche Größen, die jedoch zu jedem Zeitpunkt die Normierungsbedingung $|c_0(t)|^2 + |c_1(t)|^2 = 1$ erfüllen müssen.

$$|\psi(t)\rangle = c_0(t)\,|0\rangle + c_1(t)\,|1\rangle \tag{2.16}$$

5 Der elektrische Quadrupoltensor ist durch $Q_{\alpha,\beta} = -e/2\left(r_\alpha r_\beta - r^2/3\delta_{\alpha,\beta}\right)$ gegeben. Hier sind r_α und r_β die Koordinaten des Elektrons mit Ladung e.

6 Soll ein n-dimensionaler Hilbertraum $\mathcal{H}^n$ betrachtet werden, so ist $|\psi(t)\rangle = \sum_{j=1}^{n} c_j(t)\,|j\rangle$ mit den Basiszuständen $|j\rangle$ und den Zustandsamplituden $c_j(t)$, für die $\sum_{j=1}^{n} |c_j(t)|^2 = 1\ \forall t$ gilt.

Die Dynamik von $|\psi(t)\rangle$ unter $\hat{H}$ wird durch die Schrödinger-Gleichung beschrieben. Diese liefert

$$\partial_t |\psi(t)\rangle = -\frac{i}{\hbar}\hat{H}|\psi(t)\rangle = -\frac{i}{\hbar}\hat{H}_0|\psi(t)\rangle + \frac{i}{\hbar}\vec{dE}(t)|\psi(t)\rangle \tag{2.17}$$

und enthält die Zeitableitungen der Koeffizienten $c_0(t)$ und $c_1(t)$. Mit der Frequenz $\omega = {}^1\!/\!_\hbar(E_{|1\rangle} - E_{|0\rangle})$, welche die Energiedifferenz zwischen $|0\rangle$ und $|1\rangle$ beschreibt ist

$$\hat{H}_0|0\rangle = 0 \qquad \text{und} \qquad \hat{H}_0|1\rangle = \hbar\omega|1\rangle\,.$$

Der zeitabhängige $\hat{H}_1(t)$ koppelt durch $\vec{d}$ die Zustände $|0\rangle \longleftrightarrow |1\rangle$. Mit der Projektion von Gl. 2.17 auf $|0\rangle$ und $|1\rangle$ ergibt sich das in Gl. 2.18 angeführte Differentialgleichungssystem.

$$\begin{aligned}\partial_t c_1(t) &= -i\omega c_1(t) + \frac{i}{\hbar}\left(\langle 1|\vec{d}\vec{\epsilon}|0\rangle\, E_0 e^{-i\omega_L t} + c.c.\right) c_0(t)\\ \partial_t c_0(t) &= \frac{i}{\hbar}\left(\langle 0|\vec{d}\vec{\epsilon}|1\rangle\, E_0 e^{-i\omega_L t} + c.c\right) c_1(t)\end{aligned} \tag{2.18}$$

Die Kopplungen $\langle 1|\vec{d}\vec{\epsilon}|0\rangle$ werden durch die komplexwertige Rabi-Frequenz

$$\Omega = \frac{2\,\langle 1|\vec{d}\vec{\epsilon}|0\rangle\, E_0}{\hbar} \propto \sqrt{I_0} \tag{2.19}$$

in Gl. 2.19 ausgedrückt, deren konjugierte Frequenz mit $\Omega^\star$ notiert wird. Diese ist proportional zur elektrischen Feldstärke E_0 und steht mit der Intensität I_0 durch $E_0 = \sqrt{I_0}$ in Beziehung.

Durch Transformation in ein mit der optischen Frequenz ω_L rotierendes System mit $c_1(t) = \bar{c}_1(t)\cdot e^{-i\omega_L t}$ folgt

$$\partial_t c_1(t) = (\partial_t \bar{c}_1(t))e^{-i\omega_L t} - i\omega_L \bar{c}_1(t)e^{-i\omega_L t}.$$

In diesem rotierenden System nimmt Gl. 2.18 die in Gl. 2.20 angeführte Form an.

$$\begin{aligned}\partial_t \bar{c}_1(t) &= i(\omega_L - \omega)\bar{c}_1(t) + \frac{i}{2}\left(\Omega + \Omega e^{2i\omega_L t}\right) c_0(t)\\ &= -i\Delta\bar{c}_1(t) + \frac{i}{2}\left(\Omega + \Omega e^{2i\omega_L t}\right) c_0(t)\\ \partial_t c_0(t) &= \frac{i}{2}\left(\Omega^\star + \Omega^\star e^{2i\omega_L t}\right) c_1(t)\end{aligned} \tag{2.20}$$

Es wird die Verstimmung $\Delta = \omega - \omega_L$ eingeführt, welche den Frequenzunterschied zwischen dem Lichtfeld und dem Energieunterschied der Niveaus $|0\rangle$ und $|1\rangle$ angibt.

In 2.20 treten Terme auf, die mit der doppelten optischen Frequenz $\pm 2\omega_L$ rotieren. Diese sind von der Anregung mit der Frequenz ω weit verstimmt und werden in der Drehwellennäherung[7] vernachlässigt. Mit dieser Näherung wird Gl. 2.20 zu

$$\begin{aligned} \partial_t \bar{c}_1(t) &= -i\Delta c_0(t) + \frac{i}{2}\Omega \bar{c}_1(t) \\ \partial_t \bar{c}_0(t) &= \frac{i}{2}\Omega^\star c_0(t). \end{aligned} \tag{2.21}$$

Im resonanten $\Delta = 0$ Fall wird Gl. 2.21 einfach entkoppelt – die Lösung ist durch das Matrixexponential

$$\partial_t |\psi(t)\rangle = \partial_t \begin{pmatrix} c_0(t) \\ \bar{c}_1(t) \end{pmatrix} = \frac{i}{2}\begin{pmatrix} 0 & \Omega \\ \Omega & 0 \end{pmatrix}\begin{pmatrix} c_0(t) \\ \bar{c}_1(t) \end{pmatrix} = M|\psi(t)\rangle \longrightarrow |\psi(t)\rangle = e^{iMt/2}|\psi(0)\rangle$$

gegeben. Für ein System, das sich bei $t = 0$ in $|0\rangle$ befindet, $c_0(0) = 1$ und so $|\psi)(0)\rangle = (1, 0)$; die Lösung des Differentialgleichungssystems ist

$$c_0(t) = \cos\frac{\Omega t}{2}, \quad \bar{c}_1(t) = i\sin\frac{\Omega t}{2} \longrightarrow |\psi\rangle(t) = \cos\frac{\Omega t}{2}|0\rangle + i\sin\frac{\Omega t}{2}|1\rangle. \tag{2.22}$$

Ein Vergleich von Gl. 2.22 mit Gl. 2.3 für den Fall $\phi = 0$ zeigt, dass der Polarwinkel θ durch

$$\theta = \Omega t \propto E_0 \propto \sqrt{I_0} \tag{2.23}$$

gegeben ist[8]. Die Wahrscheinlichkeit $p_1(t)$, das Qubit zum Zeitpunkt t im Zustand $|1\rangle$ zu messen, ist durch

$$p_1(t) = \sin^2\frac{\Omega t}{2} \tag{2.24}$$

für alle t bestimmt. Es findet eine kontinuierliche Oszillation zwischen $|0\rangle$ und $|1\rangle$ statt, solange das Lichfeld eingeschaltet ist; diese werden als Rabi-Oszillationen bezeichnet.

Für den allgemeinen Fall $\Delta \neq 0$ führt die Lösung von Gl. 2.21 auf eine andere Dynamik der Zustandsamplituden. Die Verstimmung wirkt sich auf die Rabi-Frequenz als $\Omega' = \sqrt{\Omega^2 + \Delta^2}$ aus. Mit der Anfangsbedingung, dass sich das System bei $t = 0$ im Zustand $|0\rangle$ befindet, ist

$$c_0(t) = \cos\frac{\Omega t}{2} \quad \text{und} \quad \bar{c}_1(t) = -i\Omega\sin\frac{\Omega t}{2}.$$

Die Wahrscheinlichkeit $p_1(t)$ ist für die verstimmte Anregung durch Gl. 2.25 gegeben und geht für $\Delta = 0$ in Gl. 2.24 über. Die Fälle für $\Delta = 0$ und $\Delta = \Omega$ sind in Abb. 2.5 dargestellt.

$$p_1(t) = \left(\frac{\Omega}{\Omega'}\right)^2 \sin^2\frac{\Omega' t}{2} \tag{2.25}$$

7 Die Argumentation lässt sich durch Betrachtung einer ebenen Welle als Superposition einer links- und einer rechtszirkular polarisierten Welle erklären. Nach Transformation in rotierendes System bewegt sich das Qubit mit einer der Polarisationen, während die andere Komponente mit der doppelten Frequenz oszilliert [22].

8 Die Berücksichtigung eines Phasenwinkels ϕ im Lichtfeld $\vec{E}(t)$ führt auf den exponentiellen Faktor in Gl. 2.3, der die Bewegung eines Zustands in der Äquatorialebene der Blochkugel beschreibt.

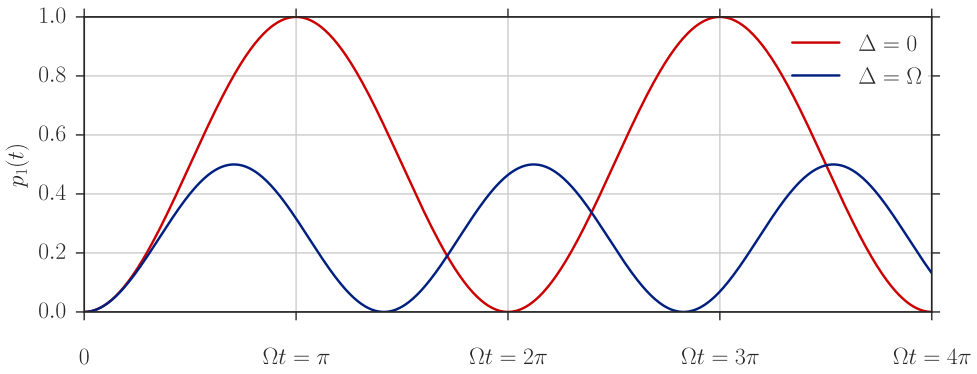


ABBILDUNG 2.5: Darstellung der resonanten und verstimmten Oszillation der Population $p_1(t)$ in $|1\rangle$. Für $\Delta = 0$ wird ein vollständiger Besetzungstransfer von $|0\rangle$ nach $|1\rangle$ in der Zeit $t = \pi/\Omega$ erreicht. Für eine Verstimmung $\Delta = \Omega$ wird eine maximale Besetzung von $0,5$ erreicht.

Für einen Quadrupol-Übergang zeigt sich ein ähnliches Verhalten – die Wechselwirkung wird durch den Quadrupoltensor $\hat{Q}$ und dem Gradienten des elektrischen Feldes $\nabla\vec{E}$ beschrieben. Die Rabi-Frequenz aus Gl. 2.19 ist nicht mehr gültig, sondern muss durch

$$\Omega = \frac{eE_0}{2\hbar}\,\langle 0|(\vec{\epsilon}\vec{r})(\vec{k}\vec{r})|1\rangle$$

ersetzt werden. Der Ortsvektor $\vec{r}$ bezeichnet die Position des Elektrons mit Ladung e; das Lichtfeld ist durch die Polarisation $\vec{\epsilon}$ und den Wellenvektor $\vec{k}$ gegeben [21].

2.3 Zustandskontrolle und -manipulation mit Radiofrequenzimpulsen

In einem Ionenfallen-Quantencomputer wird der Zustand von Quantenbits durch das Einstrahlen von kohärenten Lichtimpulsen beeinflusst. Frequenz, Phase, Amplitude und Dauer des Impulses sind Größen, die sich direkt auf die Zustandsmanipulation auswirken. Es ist daher von entscheidender Bedeutung, eine gute Kontrolle über diese Parameter zu gewinnen. Dies lässt sich mit einem akusto-optischen Modulator (AOM) bewerkstelligen, dessen Funktionsprinzip in Abschnitt 2.3.1 beschrieben wird; die Diskussion ist an [23, 24] angelehnt. Mit dem AOM werden Lichtimpulse erzeugt, deren Kenngrößen in Abschnitt 2.3.2 angeführt sind und in Frequenz und Phase einstellbar sind. Jeder Impuls ist dabei Fluktuationen ausgesetzt, die von der Lichtquelle selbst und von den Modulatoren bedingt werden. Diese beeinflussen die Quantengatter, sodass kein Gatter perfekt ausgeführt werden kann. Der Einfluss der Instabilitäten auf die Gatteroperationen wird in Abschnitt 2.3.3 diskutiert.

Im Folgenden wird die Wellenlänge des Lichts mit λ und jene des Schalls mit Λ bezeichnet. Die dazu assoziierten Wellenvektoren sind $\vec{k}$ und $\vec{K}$, die Frequenzen werden durch ω und Ω beschrieben.

2.3.1 Akusto-optische Modulatoren

Der akusto-optische Modulator ist ein optoelektronisches Element zur Modulation und Ablenkung von Licht. Er besteht aus einem Kristall, an dessen gegenüberliegenden Flächen ein Signalumformer (Transducer) und ein Schallabsorber angebracht sind. Der Umformer setzt ein angelegtes Radiofrequenzsignal mit einer Frequenz von mehreren $100\,\mathrm{MHz}$ in eine hochfrequente Schallwelle um. Diese propagiert im Kristall und führt zur Modulation des Brechungsindex n, bevor sie vom Absorber idealerweise vollständig aufgenommen wird. Für einen unter dem Winkel α zur Schallwelle eingestrahlten Laserstrahl stellt dies ein Gitter dar, welches sich mit der Schallgeschwindigkeit im Medium bewegt. Wie bei der Streuung an Kristallebenen eines Festkörpers gilt für die Ablenkung des Strahls

$$2\Lambda \sin(\alpha) = m\lambda \tag{2.26}$$

mit $m \in \mathbb{Z}$ der jeweiligen Beugungsordnung [25]. Es handelt sich um Bragg-Streuung, die sich im AOM quantenmechanisch durch die Wechselwirkung der Photonen des Lichts mit den Gitterschwingungen (Phononen) des Kristalls beschreiben lässt. Wie auch für Photonen können Phononen absorbiert und erzeugt werden; es gilt Energie- und Impulserhaltung. In Abb. 2.6 ist das Prinzip grafisch dargestellt. Der Wellenvektor des Lichts wird mit $\vec{k}$ bezeichnet, während die Phononen durch $\vec{K}$ beschrieben werden – der gebeugte Strahl ist $\vec{k}'$. Anhand der Richtung der z-Komponente der Vektoren $\vec{k}$ und $\vec{K}$ werden zwei Situationen unterschieden:

(1) Die z-Komponente von $\vec{k}$ und $\vec{K}$ sind antiparallel $\longrightarrow$ Phonon wird von Photon absorbiert.

(2) Die z-Komponente von $\vec{k}$ und $\vec{K}$ sind parallel $\longrightarrow$ Phonon wird erzeugt, die Energie des Photons nimmt ab.

Im ersten Fall absorbiert das Photon die Energie eines Phonons vollständig, sodass das Licht um die Frequenz der Schallwelle nach oben verschoben wird (Upshifting). Analog beschreibt der zweite Fall die Erzeugung einer Gitterschwingung im Kristall, wodurch die Frequenz des gestreuten Lichts abnimmt (Downshifting). Mit den Frequenzen ω, ω des Lichts und Ω des Schalls gilt

$$k' = k + K,\ \ \omega' = \omega + \Omega\ (\text{Upshifting}) \qquad k' = k - K,\ \ \omega' = \omega - \Omega\ (\text{Downshifting}).$$

Die auftretenden Beugungsordnungen werden nummeriert – das Upshifting wird mit positiven und das Downshifting mit negativen Zahlen indiziert. Als Beugungseffizienz wird das Verhältnis der Intensitäten der ersten Ordnung und des Strahls vor dem Passieren des AOMs angibt.

2.3.2 Kenngrößen eines Laserimpulses

Ein durch einen AOM erzeugter Lichtimpulse wird durch die Radiofrequenzquelle, der Elektronik und dem Modulator selbst beeinflusst. Der Impuls wird durch wenige Größen charakterisiert:

(1) Intensität I (Amplitude)

(2) Frequenz ν

(3) Phase ϕ

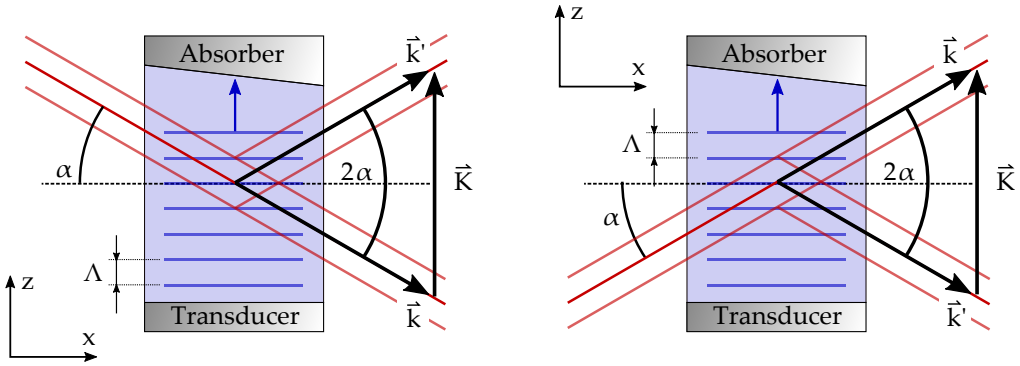


ABBILDUNG 2.6: Bragg-Streuung in akusto-optischen Modulatoren – die Schallwelle mit Wellenvektor Λ breitet sich parallel zur z-Achse aus. Das Licht mit Vektor $\vec{k}$ fällt unter dem Winkel α ein und wird an der Welle gestreut. Der gebeugte Strahl wird durch $\vec{k}'$ beschrieben und ist um die Frequenz des Schalls verschoben. Die Wechselwirkung aus Photon und Phonon beschreibt diese Verschiebung – wird ein Phonon absorbiert (linke Grafik), so wird die Frequenz erhöht. Für die Erzeugung eines Phonons verliert das Photon an Energie – die Frequenz nimmt ab.

(4) Impulsdauer t_{Puls}

(5) Anstiegszeit t_{r}

Von entscheidender Bedeutung in der Quanteninformationsverarbeitung ist die Impulsfläche $\mathcal{A}$, welche als das Produkt aus $\sqrt{I}$ und der Impulsdauer t_{Puls} definiert ist. Diese Größe koppelt mit dem Polarwinkel θ des Qubit – nach Gl. 2.19 ist die Rabi-Frequenz Ω im resonanten Fall proportional zur elektrischen Feldstärke E_0 und steht über $E_0 = \sqrt{I}$ mit der Intensität in Beziehung. Da sich I zeitlich ändert, ist $\mathcal{A}$ durch Gl. 2.27 gegeben.

$$\mathcal{A} = \int_0^{t_{\mathrm{Puls}}} \sqrt{I(\tau)}\, d\tau \tag{2.27}$$

Der Wechsel zwischen zwei Signalpegeln eines Impulses kann nicht sprunghaft erfolgen. Ein endliches Zeitintervall t_r gibt an ab welchem Zeitpunkt das Licht nach dem Schaltvorgang keine deutliche Intensitätsänderung mehr aufweist – t_r wird daher als Anstiegszeit bezeichnet. Die Schaltflanken treten jeweils zu Beginn und am Ende jedes Impulses auf und verzerren die Form, sodass sich ein Rechteckpuls in guter Näherung durch ein Trapez beschreiben lässt. Mit der mittleren Intensität I_0 eines Lichtimpulses lässt sich Gl. 2.27 durch

$$\mathcal{A} \approx \sqrt{I_0} \cdot (t_{\mathrm{Puls}} - t_{\mathrm{r}}) \tag{2.28}$$

approximieren – ist zudem die Anstiegszeit deutlich geringer als die Impulsdauer, so gilt

$$\mathcal{A} \approx \sqrt{I_0} t_{\mathrm{Puls}}. \tag{2.29}$$

Für Rechteckpulse mit $t_{\mathrm{r}} \ll t_{\mathrm{Puls}}$ ist diese Näherung zulässig; die Abweichung zwischen Gl. 2.28 und Gl. 2.29 ist $\propto {}^{t_r}/_{t_{\mathrm{Puls}}} \ll 1$. Mit einer typischen $t_r \approx 20\,\mathrm{ns}$ ist für eine Impulsdauer $t_{\mathrm{Puls}} = 1\,\mu\mathrm{s}$ ein Unterschied von 2% feststellbar. Rechteck-

und trapezförmige Impulse weisen jedoch ein Sinc-förmiges Spektrum[9] auf, dessen Fourier-Komponenten zu nicht-resonanten Anregungen des Qubits führen können. Eine Kontrolle der Impulsform durch Amplitudenmodulation der Intensität erlaubt es, unerwünschte Frequenzanteile zu unterdrücken [26]; für derartige Impulse ist nur Gl. 2.27 anwendbar.

Die Phase ϕ des Radiofrequenzsignals führt nach Gl. 2.9 direkt auf eine Bewegung des Zustands in der Äquatorialebene der Blochkugel. Idealerweise überträgt sich ein am Impulsgenerator fest eingestellter Winkel ϕ direkt auf die Schallwelle im AOM und wird nicht durch den Verlauf der Radiofrequenzamplitude beeinflusst. Auch kann für ein perfektes System während des Einschaltvorgangs ein fester Phasenbezug aufrechterhalten werden. In der Praxis treffen diese Annahmen nicht zu – der Phasenverlauf beim sprunghaften Einschalten ist chaotisch und kann nicht bestimmt werden. Zudem tritt ein Übersprechen zwischen Amplitude und Phase auf, das nicht von der Signalquelle selbst, sondern von den an der Führung des Lichts beteiligten Glasfasern, den verwendeten Detektoren und den akusto-optischen Modulatoren bestimmt wird. Diese Unsicherheiten addieren sich zum Phasenwinkel ϕ und führen so zu unerwünschten Drehungen des Zustands auf der Blochkugel.

2.3.3 Auswirkung von Instabilitäten auf Gatteroperationen

Fluktuationen der charakteristischen Größen eines Laserimpulses wirken sich direkt auf die Güte der Quantengatter aus, die den Zustand eines Qubits verändern. Die Impulsfläche $\mathcal{A}$ steht mit dem Polarwinkel θ in Beziehung, sodass Änderungen der Intensität I und der Impulslänge t_{Puls} die Fläche $\mathcal{A}$ und damit θ beeinflussen. Eine abweichende Phase ϕ führt zu einer zusätzlichen Drehung in der Äquatorialebene der Blochkugel. Die Fluktuationen sind in der Beschreibung eines Gatters wie folgt zu berücksichtigen:

- **Fluktuationen der Impulsfläche**

  Es soll ein Impuls der Fläche $\mathcal{A}_0$ erzeugt werden. Durch eine instabile Intensität des Laserlichts des akusto-optischen Modulators weicht die tatsächliche Fläche $\mathcal{A}$ vom Ideal ab. Nach Gl. 2.23 ist $\theta \propto \sqrt{I_0} t_{\mathrm{Puls}} = \mathcal{A}$ mit I_0 der Lichtintensität der Impulsdauer t_{Puls}. Mit θ_0 dem Drehwinkel des idealen Gatters gilt

  $$\theta = \theta_0 \frac{\mathcal{A}}{\mathcal{A}_0}. \tag{2.30}$$

- **Fluktuationen der Impulsphase**

  Eine instabile Impulsphase, die um $\Delta\phi$ vom Soll ϕ_0 abweicht, können direkt aufaddiert werden. Es ist

  $$\phi = \phi_0 + \Delta\phi \tag{2.31}$$

In der Beschreibung eines Ein-Qubit-Gatters nach Gl. 2.9 sind diese Fluktuationen zu berücksichtigen, um die tatsächliche Drehung eines Zustands wiedergeben zu

9 Die Abweichung des Spektrum eines Impulses in Form eines Trapezes von einem perfekten Sinc hängt von der Anstiegszeit t_r ab.

können. Ein fehlerbehaftetes Gatter lässt sich durch

$$\begin{aligned} R(\theta_0, \phi_0, \mathcal{A}, \Delta\phi) = \cos\left(\frac{\theta_0}{2}\frac{\mathcal{A}}{\mathcal{A}_0}\right)\mathbb{1} + \\ + i\sin\left(\frac{\theta_0}{2}\frac{\mathcal{A}}{\mathcal{A}_0}\right)(\cos(\phi_0+\Delta\phi)\sigma_x + \sin(\phi_0+\Delta\phi)\sigma_y) \end{aligned} \tag{2.32}$$

beschreiben. Dieses wird in Kapitel 6 im Rahmen einer Simulation implementiert, um die Auswirkungen der Modulatoren auf eine Gatterstichprobe abschätzen zu können.

Kapitel 3

Dynamische Erzeugung von Laserimpulsen

In einem Quantencomputer werden Operationen in Form von Quantengatter auf einem Qubit durchgeführt. Diese Gatter werden durch Laserimpulse beschrieben, die mit akusto-optischen Modulatoren (AOM) erzeugt werden. Die Ansteuerung der AOMs erfolgt mit Radiofrequenzsignalen im Bereich von $\nu = 80\,\mathrm{MHz} - 300\,\mathrm{MHz}$, deren Frequenz, Phase und Amplitude sich direkt auf das Licht auswirkt. In Abschnitt 3.1 wird die Erzeugung dieser Signale beschrieben. Die Bedeutung einer stabilen Zeitbasis wird in Abschnitt 3.1.1 erklärt, bevor in Abschnitt 3.1.2 ein Prinzip zur Erzeugung von durchstimmbaren Signalen beschrieben wird, die in Frequenz, Phase und Amplitude kontrolliert werden können. Zur Impulserzeugung wird ein neu entwickeltes Impulsgeneratorsystem eingesetz, das im Abschnitt 3.2 vorgestellt wird. Ein Programmcode, dessen Aufbau in Abschnitt 3.2.1 bildet die Grundlage zur Erzeugung von Radiofrequenzimpulsen – das Schreiben dieser Programme wird in Abschnitt 3.2.2 anhand von Beispielen beschrieben. Abschließend werden technische Details zur Kommunikation, die Behandlung von Fehlern im Programmcode und die aktuellen Limitierungen des Impulsgenerators in den Abschnitten 3.2.3, 3.2.4 und 3.2.5 diskutiert.

3.1 Erzeugung und Verarbeitung elektrischer Signale

3.1.1 Umsetzung stabiler Zeitbasen und Laufzeitkontrolle

Die Ausführung von Gatteroperationen hoher Güte auf einem Quantensystem setzt eine exakte Kontrolle über die Zeitparameter, Amplitude und Phase eines Laserimpulses voraus. Es ist daher wichtig, die Laufzeiten der Signale möglichst kurz zu halten, sodass zeitkritische Prozesse möglichst schnell und nahe am Experiment abgearbeitet werden können. Eine in Datendurchsatz und Verarbeitungsgeschwindigkeit hochperformante Signalverarbeitung ist unerlässlich, sodass auf digitale Systeme zurückgegriffen wird. Da Entscheidungen über den weiteren Verlauf eines Experiments in Zeiträumen unter einer Sekunde aus dem Ergebnis einer Messung abgeleitet werden müssen, ist eine genaue Kenntnis über die Verarbeitungszeit von entscheidender Bedeutung. Folgende Anforderungen können von handelsüblichen Mikroprozessoren nur bedingt erfüllt werden:

(M1) Laufzeitkontrolle – gewöhnliche Prozessoren verfügen über priorisierte Routinen (Interrupt-Routinen), welche den Programmcode unterbrechen und die Ausführung verzögern können.

(M2) Top-Down Verarbeitung des Programmcodes – der im Prozessor hinterlegten Funktionsanweisungen werden „von oben nach unten“ abgearbeitet. Die

gleichzeitige Ausführung zweier Prozesse ist nur bedingt möglich[1]. Befehle, die später im Code aufgerufen werden, werden auch zeitlich später ausgeführt.

Diese Eigenschaften schließen den Einsatz gewöhnlicher Prozessoren für zeitkritische Anwendungen auf Zeitskalen im Nanosekundenbereich aus. Durch (M1) kann einem Prozess keine exakte Bearbeitungsdauer zugeordnet werden; die Zeitspanne zwischen zwei Aufgaben kann ebenfalls variieren. Ein fester Zeitbezug dieser Art ist nur mit Echtzeitsystemen realisierbar. Ausgehend von einer globalen Zeitbasis (Systemtakt) kann genau bestimmt werden, mit welcher Verzögerung zwei Prozesse zueinander ausgeführt werden und wie lange jeder Prozess bis zum Abschluss seiner Aufgabe benötigt. Diese Systeme lassen sich in programmierbaren Gatter-Arrays (FPGA's) realisieren. Ein FPGA ist aus einer Vielzahl an Logikzellen aufgebaut, denen einfache, auf boolscher Logik basierende Aufgaben zugewiesen werden können. Durch Hintereinanderausführung dieser Zellen sind komplexe Schaltungen realisierbar; die Kenntnis über die Laufzeit eines Prozesses bleibt durch die Kontrolle des Signalflusses erhalten. Folgende Eigenschaften sind damit erfüllt:

(F1) feste Zeitbasis – der Ausgang jeder Logikzelle ist auf den globalen Systemtakt referenziert. Jede Zelle steht mit seinen Nachbarn in einer festen zeitlichen Beziehung $\longrightarrow$ *Laufzeitkontrolle*

(F2) Parallelisierung – sobald am Eingang einer Logikzelle ein Signal anliegt, wird dieses abhängig von der programmierten Funktion der Zelle an den Ausgang weitergegeben. Alle Logikzellen arbeiten parallel zueinander $\longrightarrow$ *Top-Down-Verarbeitung entfällt*.

Die Implementierung eines Systems in einem FPGA setzt ein gutes Verständnis für digitale Schaltungen und den Umgang mit Hardwarebeschreibungssprachen (Verilog oder VHDL) voraus. Das Einbringen neuer Komponenten in den Maschinencode ist für den Endbenutzer ungleich schwieriger als in mit Hochsprachen (*C*, *C++* oder *Pascal*) arbeitenden Mikroprozessoren. Komplexe Komponenten wie Netzwerkschnittstellen sind in VHDL sehr aufwändig umzusetzen. Etwaige Fehler können nur durch Analyse der Ein- und Ausgangssignale der Logikzellen gefunden werden, wodurch die Test- und Entwicklungszeit steigt. Seit einigen Jahren sind Verbundsysteme (PSoC, *Programmable System on Chip*) aus Prozessor (*processing system*, *CPU*) und FPGA (*programmable logic*, *FPGA*) verfügbar, die die Vorteile der Gatter-Arrays mit jenen eines Prozessors vereinen [27]. Die CPU ermöglicht die Umsetzung codebasierter Strukturen, während das FPGA zeitkritische Anwendungen ausführt. Beide Komponenten sind eigenständig und kommunizeren über einen gemeinsamen Datenbus miteinander. Durch die Unterbringung beider Systeme in einem einzelnen Chip und die optimierten Signallaufzeiten sind Verbindungen zwischen FPGA und CPU in kurzer Zeit möglich. Es ist keine zusätzliche Hardware notwendig, um Daten von der FPGA zu empfangen, sie in der CPU auf Basis eines nicht-zeitkritischen Codes zu analysieren und auf Basis des Ergebnisses neue Befehle an die Logik zu übergeben. Wartezeiten zwischen Entscheidungen werden so auf einige Mikrosekunden reduziert.

1 Multithreading umgeht dieses Problem: Prozesse werden „parallel" abgearbeitet, indem ihnen eine gewisse Bearbeitungszeit zugewiesen wird. Dieses Modell ist auf Hardwareebene schwer umzusetzen

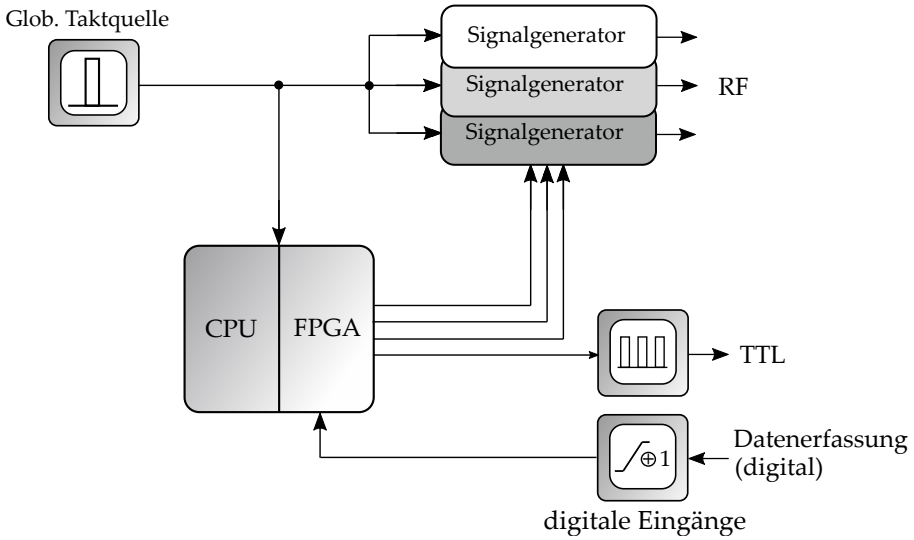


ABBILDUNG 3.1: Schematische Darstellung der Umsetzung einer gemeinsamen Zeitbasis für ein echtzeitfähiges System. Das Prozessorsystem (CPU) stellt Kommandos bereit, die von der programmierbaren Logik (FPGA) in Echtzeit an die Peripherie weitergegeben wird. Zwischen zwei Operationen des FPGA liegt ein fester Zeitbezug vor – dieser überträgt sich durch den globalen Takt auf die Radiofrequenzerzeugung, die digitalen Ausgänge (TTL) und die digitale Datenerfassung zur Zählerschaltungen.

3.1.2 Erzeugung durchstimmbarer Radiofrequenzsignale

Analoge Systeme eignen sich nicht zur Erzeugung von durchstimmbaren Radiofrequenzsignalen. Parameteränderungen sind in diesen Schaltungen oft mit einer Anpassung von Komponenten wie Widerständen und Kapazitäten verbunden. Eine gute Kontrolle über Frequenz, Phase und Amplitude ist mit digitalen Schaltkreisen nach dem Funktionsprinzip der direkten digitalen Synthese (DDS) realisierbar. Die DDS basiert auf einer in einem Halbleiterspeicher (LUT, *look-up table*) binär abgelegten Signalform. Adresse und Inhalt jeder Speicherzelle stellt einen eindeutigen Bezug zwischen dem Phasenwinkel ϕ und der Signalamplitude her; der Abstand zwischen zwei Adressen lässt sich in einen Winkel $\Delta\phi$ übersetzen. In Abb. 3.2 ist diese Zuordnung am Einheitskreis veranschaulicht. Die auszulesenden Speicheradressen werden durch einen Zähler (*Phasenakkumulator*) erzeugt, dessen Schrittweite N eingestellt werden kann. Der Akkumulator wird bei jedem Auftreten eines globalen Takts mit Periodendauer t_{CLK} um den Wert $N\Delta\phi$ erhöht. Mit einem konfigurierbaren Versatz ϕ_0 beschreibt der Zählerstand zu jedem Zeitpunkt den Winkel $\phi = \phi_0 + N\Delta\phi$. Durch Anpassung von N wird die Zeit für einen vollen Durchlauf des Speichers verändert; dies überträgt sich auf die Signalfrequenz ν.

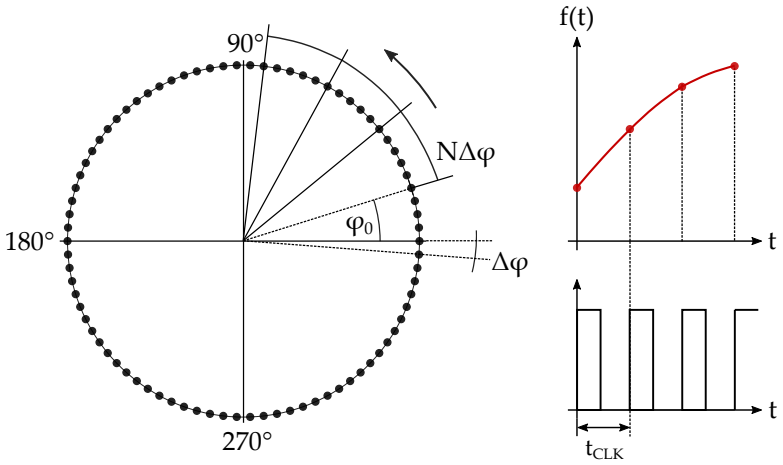


ABBILDUNG 3.2: Veranschaulichung des Funktionsprinzips der direkten digitalen Synthese am Einheitskreis. Die Amplitudenwerte $a(\Phi)$ liegen in Binärform als Funktion des Phasenwinkels Φ in einem Speicher vor. Die Speicheradresse stellt den Bezug zwischen a und Φ her. Das Auslesen des Speichers an einer beliebigen Adresse liefert die Signalamplitude an zu einem gewissen Winkel ϕ_0. Durch Addition einer Ganzzahl N zur dieser Adresse führt auf $a(\phi_0 + N\Delta\phi)$ – das Abarbeiten des gesamten Speichers rekonstruiert das ursprüngliche Sinus-Signal.

Der Inhalt einer Speicherzelle der LUT wird durch einen Digital-Analog-Umsetzer (D/A-Wandler) in eine Spannung konvertiert. Die Spannungswerte, welche am Ausgang des D/A ausgegeben werden können, sind quantisiert. Bei einer Bitbreite r und einer maximalen Ausgangsspannung $V_{\max}$ gilt Gl. 3.1.

$$\Delta V = \frac{V_{\max}}{2^r} \tag{3.1}$$

Der Frequenzgang beschreibt den Betrag einer Sinc-Funktion [28, 29], deren Minima bei ganzzahligen Vielfachen der Taktfrequenz $\nu_{\mathrm{CLK}} = {}^1\!/_{t_{\mathrm{CLK}}}$ liegen. Dies ist eine Folge der Quantisierung – das in eine analoge Größe konvertierte Bitmuster beschreibt eine „Treppenform“. Mit der Frequenz ν des Ausgangssignals treten Spiegelfrequenzen S_n mit $\nu_{\mathrm{I}} = n \cdot \nu_{\mathrm{CLK}} \pm \nu$ für $n \in \mathbb{N}$ auf. Diese sind im Spektrum in Abb. 3.3 dargestellt. Neben den S_n treten Obertöne der Grundschwingung als ganzzahlige Vielfache von ν auf und erscheinen erneut als Spiegelungen um die Taktfrequenz. Es entsteht ein für jedes ν konstantes Rauschen, das durch geeignete Filterschaltungen[2] unterdrückt wird.

Besonderes Augenmerk liegt auf der Synchronisierung zweier Radiofrequenzssignale, damit ein kohärentes Umschalten der relativen Phasenlage möglich ist. Diese Phasenkohärenz ermöglicht erst die in Abschnitt 2.1.1 diskutierten Gatteroperationen. Ein kohärenter Schaltprozess zwischen zwei Signalen A und B ist in Abb. 3.4 dargestellt. Zum Zeitpunkt des Umschaltens $A \longrightarrow B$ wird die Frequenz instantan

2 Der Frequenzgang des Wandlers kann mit Invers-Sinc-Filtern ausgeglichen werden. Diese heben den Sinc-Verlauf teilweise auf, führen aber dennoch nicht zu einem konstanten Spektrum, siehe [29] Abb. 29.

an B angepasst; es tritt ein Phasensprung auf. Das phasenkontinuierliche Schalten bewirkt eine gleichmäßige Frequenzänderung vom Startwert A zum Endwert B.

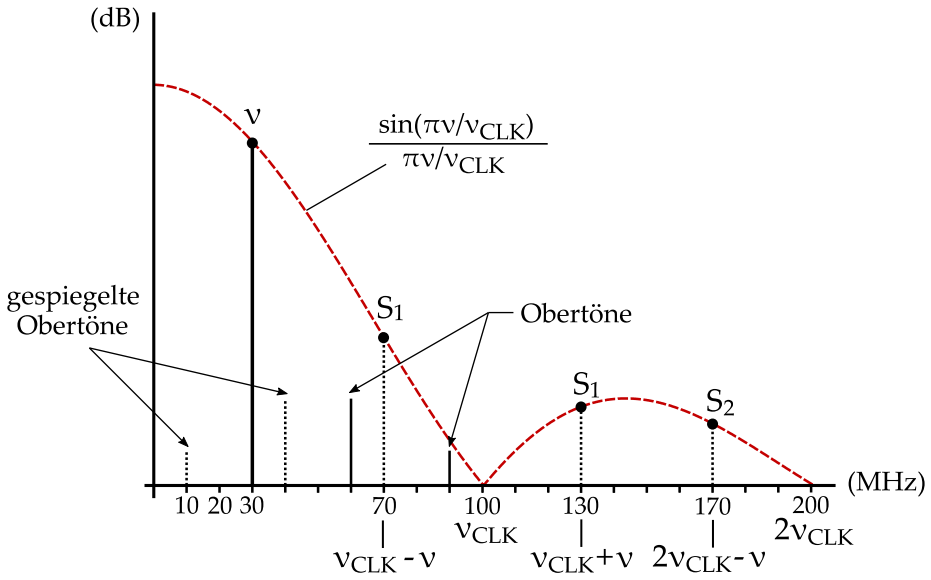


ABBILDUNG 3.3: Frequenzgang und Spektrum des Digital/Analog- Wandlers – der Frequenzgang beschreibt den Betrag einer Sinc-Funktion. Ausgehend von einem Takt ν_{CLK} wird das Signal der Frequenz ν erzeugt. Die ersten Spiegelfrequenzen sind mit S_1 bezeichnet und treten jeweils bei $\nu_{\mathrm{CLK}} \pm \nu$ auf; für S_2 folgt analog $2\nu_{\mathrm{CLK}} \pm \nu$. Es treten Obertöne der Grundschwingung als ganzzahlige Vielfache von ν auf – diese erscheinen ebenfalls gespiegelt um ν_{CLK}.

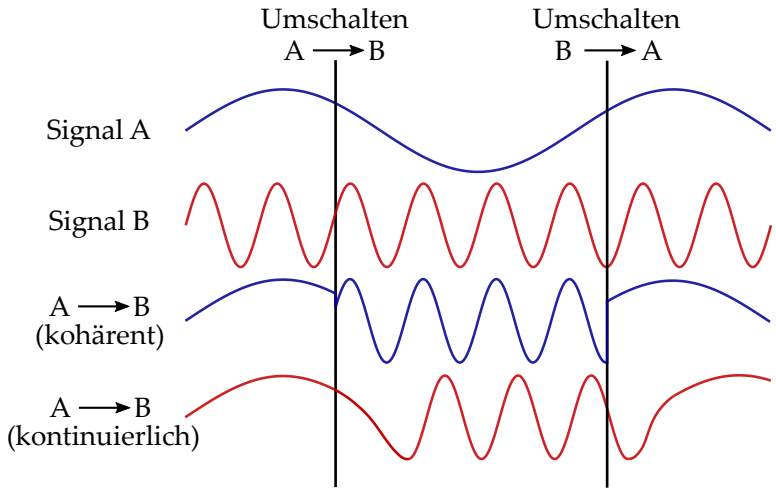


ABBILDUNG 3.4: Kohärentes und kontinuierliches Umschalten zwischen zwei Radiofrequenzssignalen A und B. Im kontinuierlichen Fall wird die Frequenz langsam von A nach B angepasst. Das kohärente Schalten bewirkt einen sofortigen Wechsel von A nach B.

3.2 Programmierbarer Impulsgenerator – M-ActION

Die Echtzeitkontrolle des Experiments wird von einem System auf FPGA-Basis vorgenommen. Der bisher verwendete Impulsgenerator *PulseBox/Sequencer* baut auf einem Gatter-Array[3] auf, welcher auf einer eigens entwickelten Trägerplatine aufgebracht ist [7]. Der Kern besteht aus einem simplen, aber echtzeitfähigen Mikroprozessor, der die Kommunikation und die Verarbeitung der Benutzereingaben vornimmt und empfangene Kommandos als Sequenzen aus digitalen und analogen Steuersignalen (*Impulse*) in Echtzeit an angeschlossene Peripheriegeräte übergibt. Die PulseBox soll aus folgenden Gründen durch ein neueres System ersetzt werden:

(1) Verfügbarkeit der Systemkomponenten – die in der PulseBox verwendete Hardware ist nicht mehr verfügbar. Neue Geräte können nicht bei Defekt einer Box nicht produziert werden.

(2) Begrenzte Länge der Impulssequenz – die Speichertiefe begrenzt die Anzahl der ausgeführten Sequenz auf ungefähr 700 Impulse. In einigen Experimenten wird dieses Limit bereits erreicht [5] und soll daher auf ≥ 10000 Impulse erweitert werden.

(3) Schnelles Umschalten zwischen Sequenzen – es soll zwischen zwei oder mehreren aufeinanderfolgenden Sequenzen gewechselt werden, deren Parameter vom Resultat der vorangehenden Messung abhängen. Die Erfassung der Daten, deren Analyse und das Generieren des Maschinencodes der PulseBox beträgt $\approx 100\,\mathrm{ms}$ und verzögert das Umschalten; in dieser Zeit ist keine Zustandsmanipulation des Quantensystems möglich.

Die Ereigniskette, welche das Ausführen zweier aufeinanderfolgender Impulssequenzen beschreibt, ist im Flussdiagramm Abb. 3.5 dargestellt. Nach Beenden der ersten Sequenz wird das Resultat der Messung analysiert und daraus eine Entscheidung über die Parameter zweiten Sequenz getroffen, bevor ein neuer Maschinencode erzeugt und an die PulseBox gesendet wird. Dieser Prozess beträgt nach (3) $\approx 100\,\mathrm{ms}$ und stellt ein deutliches Laufzeithindernis dar.
Das an der technische Hochschule Zürich entwickelte M-ActION System (*Modular Advanced Control for Trapped IONs*)[6] soll als Nachfolger der PulseBox eingesetzt werden. Das Blockschaltbild ist in Abb. 3.6 dargestellt und besteht aus folgenden Komponenten:

(1) Programmierbares Chip-System (*PSoC*, *programmable system on chip*) – dieser integrierte Schaltkreis koppelt einen FPGA und einen leistungsfähigen Mikroprozessor. Verwendet wird ein in Abb. 3.6 rot hinterlegtes PSoC ZynQ 7020 des Herstellers Xilinx, welches über einen ARM Cortex A9 Prozessor[4] und 85.000 Logikzellen bei 53.200 Konfigurationstabellen bietet. Die Kommunikation zwischen Gatterlogik und Prozessor erfolgt über einen optimierten Datenbus[5], sodass geringe Latenzzeiten erreicht werden. In Tab. 3.1 sind das PSoC und der Altera Cyclone EP1C12-FPGA der PulseBox gegenübergestellt [27, 30]. Die größere Speicherkapazität und der ARM-Prozessor des ZynQ 7020 ermöglichen im Vergleich zum Cyclone EP1C12 eine effiziente Programmierung. Verschiedene Schnittstellenstandards wie eine serielle Schnittstelle RS232[6] und

3 Altera Cyclone EP1C12Q240C6
4 ARM *Acorn RISC Machines* Prozessor in RISC-Architektur (Rechnen mit reduziertem Befehlssatz)
5 Xilinx AMBA AXI4 (*Advanced eXtensible Interface bus 4*
6 hier: Baud-Rate 115.200, effektive Datenübertragungsrate 11.520 Byte pro Sekunde

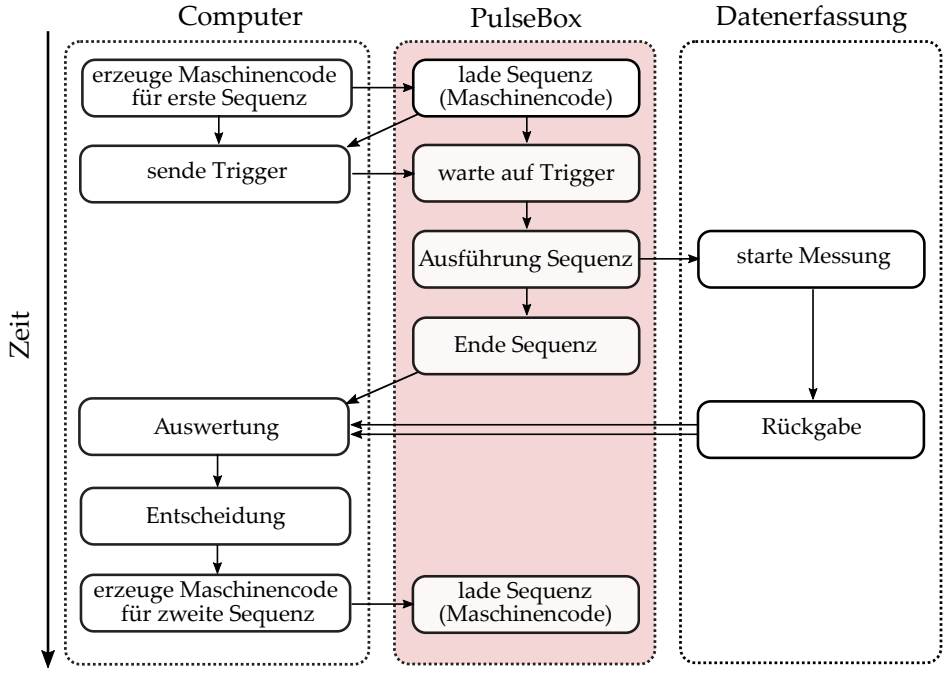


ABBILDUNG 3.5: Ereigniskette bei der Ausführung eines Experiments auf der PulseBox. Nach Ablauf der ersten Sequenz ist werden aus den gewonnenen Daten die Parameter der Folgesequenz bestimmt und in einem für das System verständlichen Maschinencode übersetzt. Diese Prozesse verzögern die Ausführung der zweiten Sequenz um $\approx 100\,\mathrm{ms}$

.

	ZynQ 7020	Cyclone EP1C12
Prozessor	2x ARM Cortex A9	nicht vorhanden
Logikzellen	85.000	12.060
Speicher (FGPA)	4,9 Mb	0,3 Mb

TABELLE 3.1: Gegenüberstellung des ZynQ 7020 PSoC von M-ActION mit dem Cyclone EP1C12 der PulseBox. Neben einem leistungsfähigen Prozessor bietet das PSoC die 7-fache Anzahl an Logikzellen und einen 16-fach größeren Speicher.

Gigabit-Ethernet werden vom PSoC auf der Prozessorseite bereits und müssen nicht eigens programmiert werden. Komplexe Schaltungen sind durch die höhere Anzahl an Logikzellen realisierbar.

(2) DDS-Signalgeneratoren – die Erzeugung der Radiofrequenzsignale wird von eigens entwickelten Schaltungen vorgenommen. Diese werden von einem Spartan 6E LX150T FPGA angesteuert und sind über das in Abb. 3.6 blau angedeutete Bus-System[7] mit dem PSoC verbunden, das bis zu 16 dieser Schaltungen unterstützt. Die Frequenzerzeugung erfolgt nach dem in Abschnitt 3.1.2 beschriebenen Prinzip der direkten digitalen Synthese durch die integrierten Schaltkreise Analog Devices AD9910 [29], deren Eigenschaften in Tab. 3.2 angeführt sind. Alle Signalgeneratoren werden mit einer Taktfrequenz von $\nu_{\mathrm{CLK}} =$

7 Das Bus-System verwendet die Hardware des PCI-Express/MicroTCA (*PCIe*, *µTCA*); das Protokoll ähnelt SPI (*serial peripherial bus*)

1 GHz betrieben, welche von einer gemeinsamen Quelle[8] abgeleitet wird und eine feste Zeitbasis realisiert. Das Taktsignal darf einen Signalpegel von $-10\mathrm{dBm}$ nicht überschreiten, um die Eingangsstufen der Schaltungen nicht zu beschädigen.

Technische Daten AD9910 bei $\nu_{\mathrm{CLK}} = 1\,\mathrm{GHz}$	
Frequenzauflösung	$\Delta\nu = 0,23$ Hz, 32 Bit
Phasenauflösung	$\Delta\phi = 1,5 \cdot 10^{-5}\pi$, 16 Bit
Amplitudenauflösung	$\Delta V = V_{\mathrm{max}}/2^{14}$, 14 Bit

TABELLE 3.2: Auflösung von Frequenz, Phase und Amplitude des AD9910 Signalgenerators bei einer Taktfrequenz von $\nu_{\mathrm{CLK}} = 1\,\mathrm{GHz}$.

(3) Digitale Ausgänge – M-ActIon stellt 32 Digitalausgänge bereit, die über eine Verbindungsplatine ausgeführt und galvanisch getrennt sind. Die Signale werden vom FPGA des PSoC erzeugt und anschließend durch Spannungskonverter auf TTL-Standard[9] umgesetzt.

(4) Zählerschaltungen – im FPGA der PSoC sind 8 Zählerschaltungen integriert, welche zur Erfassung gepulster Signale verwendet werden. Das Auslesen der Zählerstände erfolgt über den ARM Prozessor.

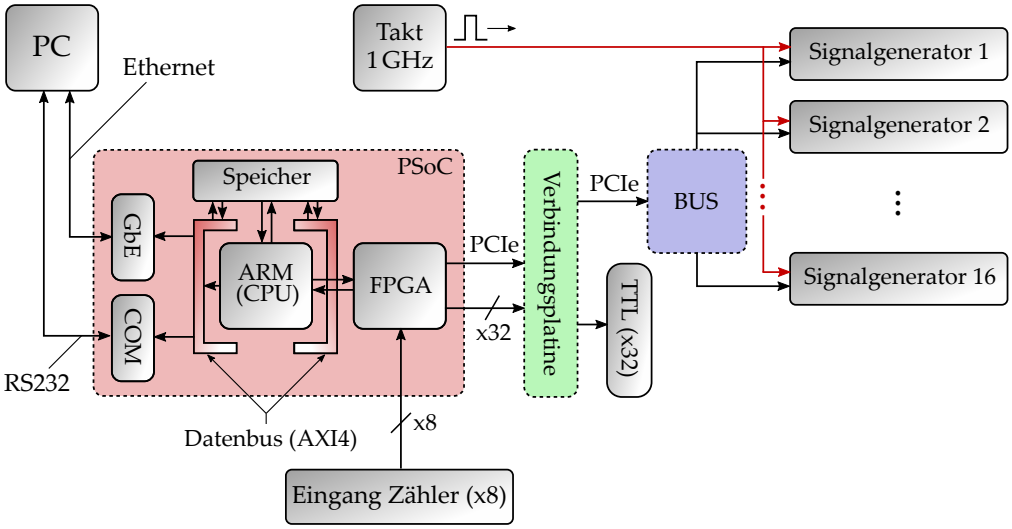


ABBILDUNG 3.6: Schematische Darstellung des M- ActION Impulsgenerators. Das PSoC (blau) koppelt einen leistungsfähigen ARM Mikroprozessor mit einem FPGA. Die Peripheriegeräte sind rot hinterlegt und über eine Anschlussplatine mit dem PSoC verbunden.

8 Marconi 2025 Signal Generator, $9\,\mathrm{kHz} - -2,51\mathrm{GHz}$

9 *Transistor- Transistor-Logik*, Spannungspegel Eingang: logisch 1 - $\geq 2\,\mathrm{V}$, logisch 0 - $\leq 0,8\,\mathrm{V}$, Spannungspegel Ausgang: 1 - $\geq 2,4\,\mathrm{V}$, 0 - $\leq 0,4\,\mathrm{V}$

3.2.1 Aufbau eines Experiments in M-ActION

Der Umsetzung eines Experiments am Impulsgenerator M-ActION wird in diesem Abschnitt diskutiert. Ein Experiment besteht aus einer Abfolge von Radiofrequenz- und Digitalpulsen, welche zeitlich aufeinander abgestimmt an den Signalgeneratoren und den TTL-Ausgänge anliegen. Jede dieser Impulssequenzen wird im Speicher von M-ActION abgelegt und vom ARM-Prozessor des PSoC ausgeführt. Wird ein Experiment vom Benutzer gestartet, so sind zu jedem Zeitpunkt andere Systemkomponenten an der Ausführung beteiligt. Die Ereigniskette wird grob durch die folgenden Schritte beschrieben:

(1) Kommando des Benutzers – der ARM-Prozessor empfängt ein Datenpaket über die Ethernet- Schnittstelle und verarbeitet den Inhalt. Wird der Befehl[10] zum Start einer Impulssequenz empfangen, so wird der nächste Schritt im Ablauf ausgelöst.

(2) Zurücksetzen der Hardware – die Signalgeneratoren und das FPGA werden zurückgesetzt. Impulse, welche von einem vorangehenden Experiment erzeugt wurden, werden gelöscht und die Zählerschaltungen neu initialisiert.

(3) Verarbeitung der Impulssequenz – das vordefinierte Experiment/die Impulssequenz wird vom ARM aufgerufen und der Programmcode von oben nach unten abgearbeitet. Die Impulse werden aus dem Code erzeugt und an das FPGA und die Signalgeneratoren weitergegeben.

(4) Ausführung – über das Bussystem werden Signalgeneratoren, Digitalausgänge und Zählerschaltungen einmalig synchronisiert und der Start der Sequenz ausgelöst. Während dem Abarbeiten der Impulse kommunizieren Digital- und Radiofrequenzerzeugung nicht miteinander, sondern bleiben durch Kenntnis der Taktsignale im Code synchron [6].

(5) Rückgabe der Zählerstände – am Ende der Sequenz werden die Zählerstände des FPGA ausgelesen und an den ARM-Prozessor zurückgegeben. Dieser verarbeitet die Daten nach Benutzerkriterien[11], trifft auf Basis des Resultats eine Entscheidung über das weitere Vorgehen[12] und gibt das Ergebnis an den Benutzer weiter.

M-ActION arbeitet auf Basis der objektorientierten Programmiersprache C++. Es wird mit Klassen gearbeitet, die eine Vererbung von Eigenschaften und Codestrukturen erlauben[13]. Dieses Merkmal ist für das Beschreiben eines Experiments vorteilhaft, da sich auch hier bestimmte Abläufe wiederholen:

- Zurücksetzen der Hardware
- Dopplerkühlen
- Seitenbandkühlen/optisches Pumpen

10 Der Aufbau dieses Befehls wird zu einem späteren Zeitpunkt erklärt und ist für die grobe Skizze der Ereignisse nicht von Bedeutung.

11 Die Analyse der Daten wird vom Benutzer im Code des jeweiligen Experiments festgelegt.

12 Dieser Schritt ist ebenfalls Abhängig vom Programmcode des Benutzers. Ein zustandsabhängiges Umschalten zwischen zwei oder mehreren Sequenzen kann für viele Anwendungen nicht notwendig sein.

13 Vererbungen ermöglichen das Weitergeben von Eigenschaften der übergeordneten Klasse, *parent*, an die abgeleiteten Klassen *child*. Das *child* kann einzelne Codefragmente überschreiben, aber nicht den vollständigen Code ändern

- Detektion
- Verarbeitung der Sequenz und Konfiguration der Hardware
- Ausführung der Sequenz
- Auslesen der Zählerstände
- Berechnung des Resultats und Rückgabe der Daten an den Anwender

Diese Schritte müssen durch die einmalige Definition in einer Klasse nicht für jedes Experiment erneut definiert werden, sondern lassen sich durch Vererbung übernehmen. M-ActION behandelt jedes Experiment als Klasse welche von einer gemeinsamen Basis `experiment`[14] abgeleitet wird. Dabei legt `experiment` die in Abb. 3.8 dargestellte Grundstruktur fest und gibt die Reihenfolge der einzelnen Routinen vor. Das Zurücksetzen und die Neukonfiguration der Hardware müssen so nicht für jedes Experiment separat definiert werden. Die blau hinterlegten Funktionen werden von den abgeleiteten Experimenten überschrieben und übernehmen folgende Aufgaben:

- `init_vars()` – Definition von globalen Variablen durch den Benutzer, welche im ganzen Experiment geteilt werden.
- `init_pulse_sequence()` – die eigentliche Impulssequenz wird durch diese Routine beschrieben. Es wird empfohlen, diese in weitere Funktionen zu unterteilen, um die Lesbarkeit des Codes zu gewährleisten. Das Doppler- und Seitenbandkühlen und die Ereignisdetektion werden hier definiert. Jedes Experiment muss eine nicht-leere `init_pulse_sequence()` aufweisen, um korrekt ausgeführt werden zu können.
- `read_out_pmts()` – Auslesen der Zählerstände, sofern notwendig. In Systemen, die auf die Verwendung der PulseBox optimiert sind, wird die Datenerfassung durch externe Hardeware vorgenommen. Für diese ist das Auslesen von M-ActION's Zählern nicht relevant und kann als Funktion ohne Inhalt beschrieben werden.
- `calc_results()` – Berechnung des Resultats des Experiments aus den erfassten Zählerständen. Diese Funktion ist hinfällig, falls `read_out_pmts()` keine Daten aufzeichnet.

Besonderes Augenmerk wird auf `init_pulse_sequence()` gelegt, da diese Routine die eigentliche Impulssequenz enthält. Die Sequenz ist von Experiment zu Experiment unterschiedlich, teilt jedoch wieder den Kühl- und Detektionsprozess – diese werden vom verwendeten Quantensystem vorgegeben. Es ist daher ratsam, erneut eine Klasse zu definieren, die auf den verwendeten Messaufbau im Labor zugeschnitten ist. Eine in Abb. 3.7 gezeigte Struktur wird empfohlen und soll eingehalten werden. Für jedes Labor wird eine spezifische Klasse als Vorlage erstellt, welche auf `experiment` aufbaut. In `init_pulse_sequence()` werden das Dopplerkühlen, Seitenbandkühlen und die Detektion definiert und zugleich eine neue Routine, im Folgenden `lab_sequence()` bezeichnet, angelegt, welche die jeweilige Impulssequenz für eine bestimmte Messung enthält.

14 Dateiname: `experiment.c`, Kopf in `experiment.h`

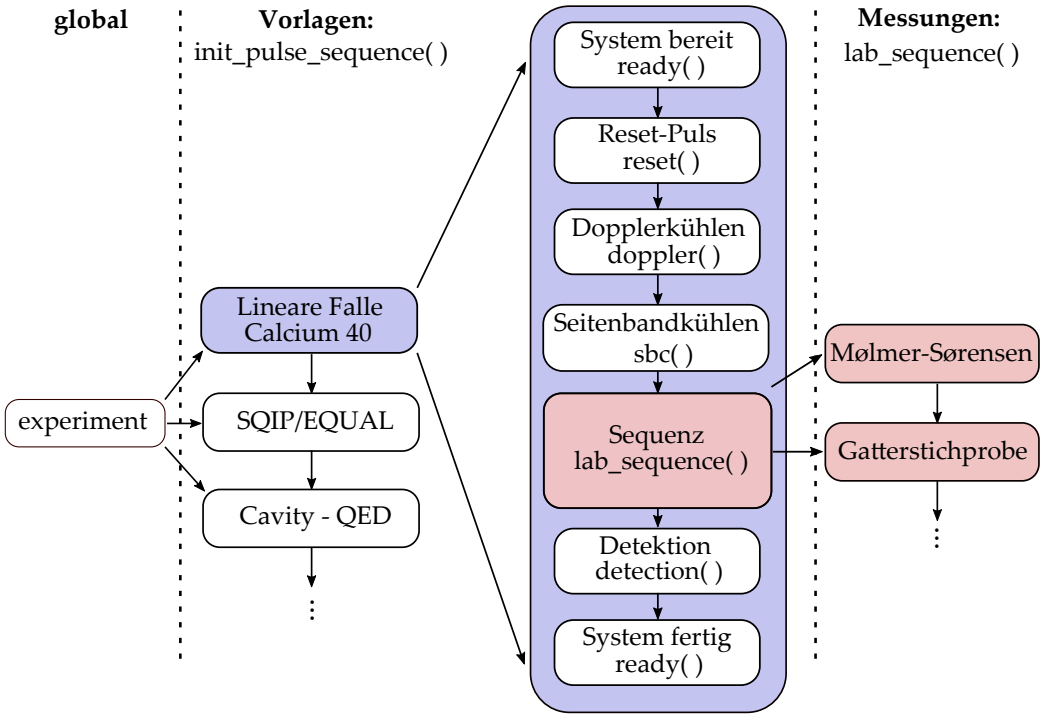


ABBILDUNG 3.7: Klassenstruktur für unterschiedliche Labors – jedes Labor (Lineare Falle, SQIP, Cavity-QED, . . .) definiert eine Vorlage, welche Kühl- und Detektionsprozesse beinhaltet. Die Sequenz wird in `lab_sequence()` für die jeweilige Messung eingebracht.

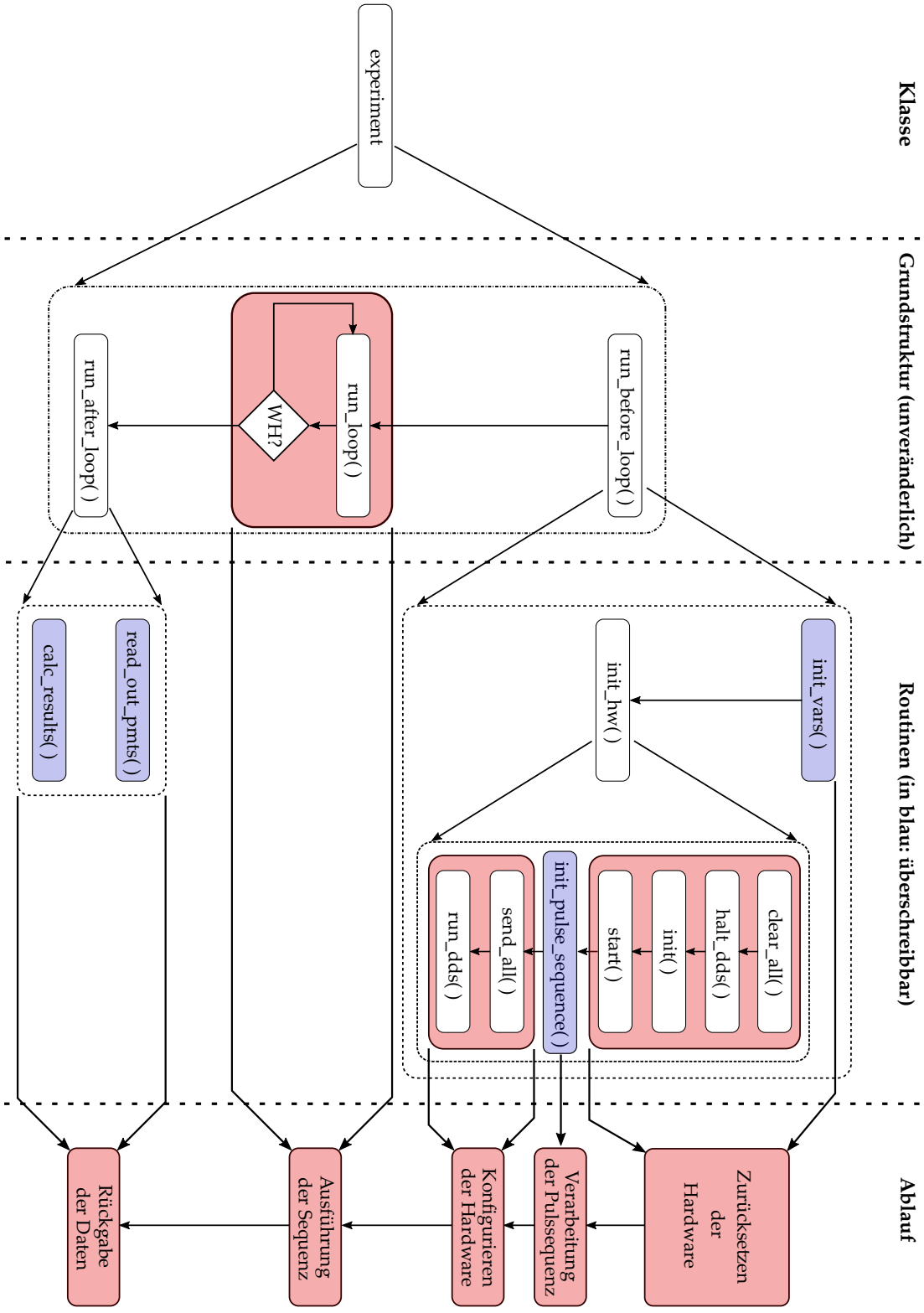


ABBILDUNG 3.8: Aufbau der Basisklasse `experiment` – die Grundstruktur ist unveränderlich und stellt sicher, dass die jeweiligen Routinen zum richtigen Zeitpunkt ausgeführt werden. Routinen, die von abgeleiteten Klassen überschrieben werden, sind blau hinterlegt.

Es sei an dieser Stelle nochmals angemerkt, dass jedes Experiment als Programmcode in M-ActION's Speicher hinterlegt ist. Die Kommunikation zwischen Anwender und Impulsgenerator beschränkt sich auf das Übergeben von Parametern, bevor eine Impulssequenz ausgeführt wird. M-ActION speichert den zuletzt verwendeten Parametersatz, sodass ein mehrmaliges Senden derselben Daten entfällt. Zusammen mit der Datenerfassung durch das FPGA erlauben diese Merkmale einen Wechsel zwischen einzelnen Impulssequenzen innerhalb von $26\,\mu s$ [31] – im Vergleich zur PulseBox stellt dies eine Laufzeitverbesserung um den Faktor ≈ 400 dar.

3.2.2 Programmierung von Radiofrequenzimpulsen

In diesem Abschnitt wird die Programmierung von einzelnen Radiofrequenz- und Digitalpulsen beschrieben. Diese Impulse werden in den in Abschnitt 3.2.1 diskutierten Routinen `init_pulse_sequence()` und `lab_sequence()` definiert und von M-ActION's Prozessor abgearbeitet. Die Impulserzeugung wird mit Codebeispielen und Darstellungen des jeweiligen Impulses behandelt – eine einfache Sequenz aus drei unterschiedlichen Impulsen wird am Ende dieses Abschnitts diskutiert. Zum besseren Verständnis der Beispiele sind zwei Konzepte der Programmiersprache C++ zu erklären:

(1) Namensräume (*namespaces*) – je größer ein Projekt wird, desto häufiger kommt es zu Überschneidungen bei der Namensgebung. Eine Variable `foo` kann von mehreren Programmierern in unterschiedlichem Kontext definiert werden; es kommt zu Kollisionen bei der Ausführung des Codes. Ein Namensraum ermöglicht die eindeutige Zuordnung der Bezeichner von Variablen oder Klassen. Mit diesem Konzept lässt sich eine Kollision durch die Definition der Räume `space_1` und `space_2` vermeiden – die Variablen sind `space_1::foo` und `space_2::foo` und werden als eigenständige Größen erkannt.

(2) Zeiger (*pointer*) – das Anlegen einer Variable `foo` führt dazu, dass eine Speicherzelle reserviert wird. Es existiert eine Zuordnung zwischen dem Namen `foo` und der Adresse der Speicherzelle. Wird `foo` mit ihrem Namen an eine Funktion im Programm übergeben, so wird der Inhalt der Speicherzelle gelesen und eine Kopie angelegt. Für diese Kopie wird erneut eine Speicherzelle belegt, welche von der Funktion manipuliert wird – das Original bleibt unangetastet[15]. Diese Methode kostet wertvollen Speicher, weshalb Zeiger bevorzugt werden. Ein Zeiger enthält die Adresse einer Speicherzelle; der Inhalt einer Zelle wird durch `*` zugänglich. Wird ein Zeiger an eine Funktion übergeben, so wird keine Kopie erstellt sondern das ursprüngliche `foo` bearbeitet. Zeigt ein Zeiger auf ein Objekt wie eine Klasse, so wird auf Variablen (*member*) durch `->` zugegriffen.

Im Folgenden wird der Namensraum `dds` und das Objekt `bp` verwendet. Sämtliche Routinen, welche die Impulse erzeugen, stammen aus `bp`, während Variablen aus `dds` abgeleitet werden. M-ActION arbeitet ausschließlich mit Zeigern, welche zu Beginn vom Benutzer auf den Nullzeiger `NULL`/**`nullptr`** zu initialisieren sind – dies wird anhand der Codebeispiele diskutiert.

15 Diese Art der Parameterübergabe wird als *call-by-value* bezeichnet. Der Wert wird aufgerufen, geklont und anschließend verarbeitet.

M-ActION unterscheidet zwischen verschiedenen Typen Radiofrequenzimpulsen, welche durch eigene Routinen generiert werden. Ein Impuls besteht aus den charakteristischen Größen Frequenz ν, Phase ϕ, Amplitude a und Impulsdauer t. Diese sind in einem Impulsobjekt `dds::ttl_dds_pulse` zusammenfasst. Frequenz, Phase und Amplitude stellen ein eigenes `dds::FPA`-Objekt (*Frequency/Phase/Amplitude*) dar; Zeitparameter werden mit `dds::Time` identifiziert. Die in Tab. 3.3 angeführten Grenzwerte für die einzelnen Parameter dürfen nicht überschritten werden. Es ist hier besonders auf das Minimum der Zeitparameter zu achten. Die Hardware benötigt eine Zeit von $1,4\mu$s, um einen Impuls vorzubereiten und auszuführen; diese Grenze kann nicht unterschritten werden. Frequenz und Phase werden durch die Signalgeneratoren festgelegt.

Größe	Symbol	Wertbereich	Einheit
Frequenz	ν	0 - 400	MHz
Phase	ϕ	0 - $360/2\pi$.	deg/rad (einstellbar)
Amplitude	a	0 - 100	% von $0\,\mathrm{dBm}$ (exakt 0,03 dBm)
Zeitparameter	t	1,4 - $2^{32}-1$.	µs, Schrittweite $8\,\mathrm{ns}$
Digitalpulse	TTL	0 - $2^{32}-1$	–
Zählereingänge	PMT	0 - $2^{8}-1$	–

TABELLE 3.3: Wertbereich der Kenngrößen eines Impulses in M-ActION – das System arbeitet mit Zahlen im Bereich von 0 bis $2^{32}-1$ (32-Bit Registerbreite), was die theoretische Obergrenze für die meisten Parameter darstellt. Für Amplitude, Frequenz und Phase sind nur physikalisch relevante Werte (z.B. strikt positiv) sinnvoll.

Die Zuweisung von Werten eines Objekts erfolgt durch eigene Funktionen, die in der Klasse `bp` definiert und durch `bp->use` im Code identifiziert werden. Jede `bp->use`-Routine enthält den Namen des jeweiligen Objekts, das sie generiert[16]. Ihr erstes Argument ist der Zeiger auf das zu bearbeitende Objekt `dds::FPA` oder `dds::Time`, dem der Operator `&` vorangestellt ist[17]. Dabei überprüft `bp->use`, ob der Zeiger auf `NULL`/**`nullptr`** zeigt – dies kennzeichnet einen leeren Parameter. Der zuzuweisende Wert stellt das zweite Argument der Routine dar. Hier ist der Wertebereich in Tab. 3.3 zu beachten – nicht unterstützte Werte führen zu einem nicht identifizierbaren Fehler im Programm. Nach Initialisierung der Parameter wird ein Impuls durch `bp->make` erzeugt. Wie auch für `bp->use` identifiziert der Name der jeweiligen `bp->make`-Routine den zu erzeugenden Impuls. Im Folgenden werden die Impulstypen mit ihren Parametern grafisch dargestellt und der Programmcode zur Erzeugung des Impulses beschrieben – Bitmuster, welche am Digitalbus ausgegeben werden, sind in den Codebeispielen nicht enthalten und werden anschließend diskutiert.

- **Edge**-Impulse sind RF-Signale, welche über Frequenz ν, Amplitude a und Phase ϕ definiert sind. Sie benötigen allgemein keine Zeitinformationen und eignen sich zur Erzeugung von kontinuierlichen Signalen (*CW*-Signale). Ein Zeitparameter t_{on}, welcher die Verzögerung bis zur Ausführung des Impulses angibt, kann zusätzlich gesetzt werden.

16 `bp->use_fpa` erzeugt ein FPA-Objekt

17 Für einen Zeiger **`int`** `*p` liefert `&p` die Adresse von `p` und `*p` den Inhalt der Speicherzelle, auf die `p` zeigt

```
// Erzeugen der Zeiger und Initialisierung auf NULL

 dds::FPA *edge_fpa = nullptr;     // der *-Operator
 dds::Time *t_on = nullptr; // identifiziert Zeiger
 dds::ttl_dds_pulse *edge = nullptr; // Impulsobjekt

/* Initialisiere Parameter mit bp->use
 nu, phi und a sind ganze Zahlen im gültigen
 Wertebereich für Frequenz, Phase und Amplitude */

 bp->use_fpa(&edge_fpa,[nu,phi,a]); // FPA-Objekt
 bp->use_time(&t_on, 10); // Zeitobjekt mit 10µs

// Erzeuge Impulsobjekt mit bp->make

 bp->make_edge(&edge,edge_fpa,t_on);

/* "edge" ist jetzt ein ausführbarer Impuls. */
```

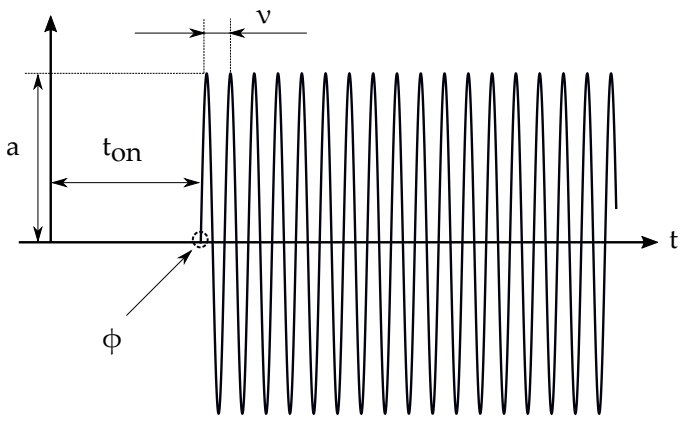


ABBILDUNG 3.9: Darstellung eines Edge-Impulses – Amplitude a, Phase ϕ und Frequenz ν können angepasst werden. Der Zeitparameter t_{on} wird lediglich bei Bedarf gesetzt; sein Standardwert ist Null.

- **Cap**-Impulse stellen einen Impuls im eigentlichen Sinn dar – sie haben definierte Start- und Endzustände, die mit *On* und *Off* bezeichnet werden. Die Umsetzung erfolgt durch die Hintereinanderausführung zweier Edge-Impulse. Der On-Zustand beim Impulsstart wird durch die Amplitude a_{on}, die Frequenz ν_{on} und den Phasenwinkel ϕ_{on} wie in Abb. 3.10 dargestellt definiert, während der Off-Zustand durch einen zweiten Parametersatz (a_{off}, ν_{off}, ϕ_{off}) beschrieben wird. Die Phase zwischen den beiden definierten Zuständen wird kohärent geschaltet. Als Zeitparameter muss mindestens die Impulslänge t_{off} angegeben werden, um den Impuls erzeugen zu können.

```
/* Für Cap-Impulse werden zwei FPAs und zwei Zeitparameter
 "_on" und "_off" benötigt -- Initialisierung der
 Parameter analog zu Edge-Impuls */

 dds::FPA *cap_fpa_on = nullptr;
 dds::FPA *cap_fpa_off = nullptr;
 dds::Time *t_on = nullptr;
 dds::Time *t_off = nullptr;
 dds::ttl_dds_pulse *cap = nullptr;

/* Initialisiere Parameter mit bp->use
 Erzeuge "ON"-Zustand zuerst: */

 bp->use_fpa(&cap_fpa_on,[nu_on,phi_on,a_on]);
 bp->use_time(&t_on, 10); // Zeitobjekt mit 10µs

// Erzeuge "OFF"-Zustand:

 bp->use_fpa(&cap_fpa_off,[nu_off,phi_off,a_off]);
 bp->use_time(&t_off, 20); // Zeitobjekt mit 20µs

/* Erzeuge Impulsobjekt mit bp->make

 Zuerst sind die FPA- und anschließend die
 Zeitobjekte anzugeben. */

 bp->make_cap(&cap,cap_fpa_on,cap_fpa_off,t_on,t_off) ;
```

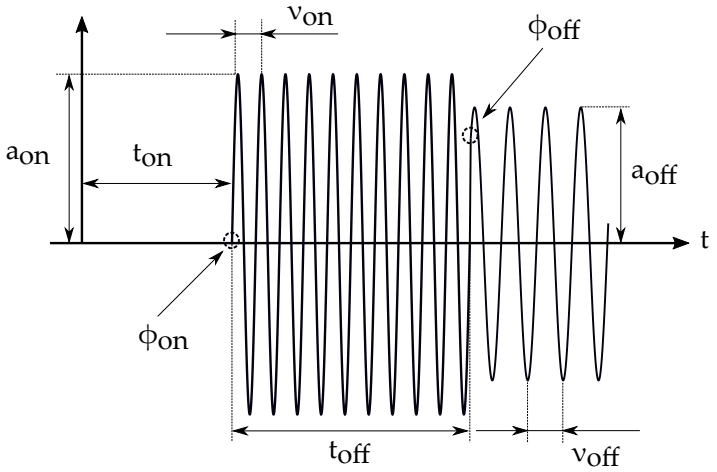


ABBILDUNG 3.10: Darstellung eines Cap-Impulses und seiner Parameter. Neben den Kenngrößen eines *Edge*-Impulses (a, ν, ϕ und t_{on}) wird ein weiterer Parametersatz für den Endzustand übergeben.

- **Shaped**-Impulse erlauben eine vollständige Kontrolle der Parameter a_{on}, ν_{on}, ϕ_{on}, t_{on} und t_{off} des *Cap*-Impulses einschließlich der Impulsform und der Anstiegszeit t_{r}. Anstelle eines sprunghaften Schaltvorgangs wird die Amplitude des Impulses nach einer vorgegebenen Funktion aufgebaut. Unerwünschte Frequenzkomponenten im Spektrum des Impulses werden so unterdrückt. Die Form wird durch eine Tabelle beschrieben, die vom Benutzer abgeändert werden kann. Zwei Varianten dieser Impulse (*Slow-Shaped* und *Med-Shaped*) sind bereits vordefiniert und benötigen keine zusätzlichen Forminformationen – diese werden im Code-Beispiel erzeugt. Eine genaue Beschreibung eines allgemeinen Shaped-Impulses ist in Abschnitt B.4 enthalten.

```
/* Für Shaped-Impuls werden zwei FPAs "_on" und "_off"
 und ein Zeitparameter t_on initialisiert.
 Zwei Impulse "sh_slow" und "sh_medium" werden erzeugt. */

 dds::FPA *sh_fpa_on = nullptr;
 dds::FPA *sh_fpa_off = nullptr;
 dds::Time *t_on = nullptr;
 dds::ttl_dds_pulse *sh_slow = nullptr;
 dds::ttl_dds_pulse *sh_medium = nullptr;

/* Initialisiere Parameter mit bp->use
 Erzeuge "ON"/"OFF" und t_on zuerst: */

 bp->use_fpa(&sh_fpa_on,[nu_on,phi_on,a_on]);
 bp->use_fpa(&sh_fpa_off,[nu_off,phi_off,a_off]);
 bp->use_time(&t_on, 10);

// Erzeuge Impulse mit bp->make

 bp->make_med_shaped(&sh_medium,sh_fpa_on,sh_fpa_off,t_on);
 bp->make_slow_shaped(&sh_slow,sh_fpa_on,sh_fpa_off,t_on);
```

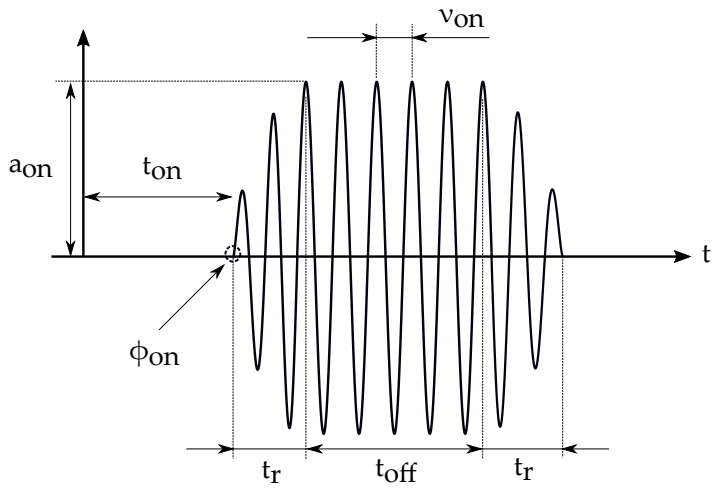


ABBILDUNG 3.11: Darstellung eines Shaped-Impulses. Die Impulsgrößen sind analog zu Abb. 3.10 definiert – der *Off*-Zustand ist bei diesem Impuls durch die Amplitude $a_{\mathrm{off}} = 0$ gegeben.

- **Wait**-Impulse beschreiben Verzögerungen und keinen Impuls im eigentlichen Sinn – sie benötigen lediglich einen Zeitparameter um erzeugt werden zu können. Dieser Impulstyp eignet sich zur Ausgabe reiner Logiksignale.

```
/* Ausgabe des Bitmusters 0x01010101 am Digitalbus
 Initialisierung eines Impulsobjekts und eines
 Zeitparameters, mehr wird nicht benötigt. */

 dds::ttl_dds_pulse *wait = nullptr;
 dds::Time *t_wait = nullptr;

// Initialisierung von t_wait

 bp->use_time(&t_wait,20) // 20µs

// Erzeugung des Impulses

 bp->make_wait(&wait, t_wait, 0x01010101)
```

Die Ausgabe von Bitmustern am Digitalbus ist für Wait-Impulse im Codebeispiel enthalten, ohne dass diese bisher diskutiert wurde. Jedes Digitalsignal besteht aus zwei Parametern `ttl_word` (*Wort*) und `ttl_mask` (*Maske*), die jeweils 32 Bit entsprechen. Der Ausgang wird „maskiert“, sodass das Wort nur jene Bits ändert, welche von der Maske nicht durch eine Null geschützt sind; `ttl_mask = 0xFFFFFFFF` erlaubt Änderungen auf allen 32 Bits. In Abb. 3.12 ist dieses Prinzip veranschaulicht. Nur die blau hinterlegten Bits dürfen verändert werden.

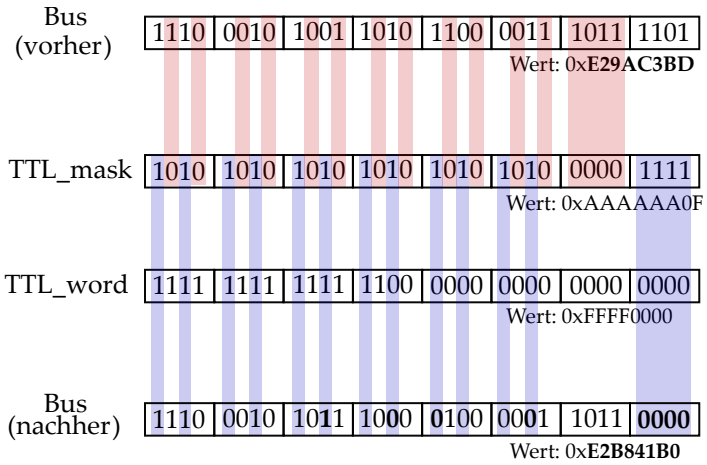


ABBILDUNG 3.12: Bit-Maskierung des Digitalbus – nur jene Bits am Bus, die in `TTL_mask` als 1 gesetzt sind, dürfen durch `TTL_word` verändert werden. Die übrigen Bits bleiben durch die Maske geschützt. Alle Werte sind binär (in den Registern) und hexadezimal (unterhalb der Register) angegeben.

Bitmaske und -muster sind optionale Parameter eines jeden Impulses; sie müssen nicht zwingend angegeben werden, sondern werden mit vordefinierten Standardwerten[18] automatisch gesetzt[19]. Zusammen mit dem zu aktivierenden Zähler (PMT) des FPGA bilden sie die letzten drei Parameter am Ende jeder `bp->make`-Routine. Jeder Zählereingang entspricht dabei einem Bit einer 8-Bit Zahl; der erste Eingang wird mit `0x01 = 0b0000001`[20] geöffnet und mit `0x00` geschlossen. Die Bitmuster liegen für die gesamte Impulsdauer am Ausgang an und werden mit dem nächsten Impuls zurückgesetzt; der Zähler wird mit Impulsende automatisch gestoppt.

In den Codebeispielen werden die einzelnen Impulse durch `bp->make` erzeugt, jedoch noch nicht ausgeführt. Das Kommando `bp->run` startet den Impuls; der erste Parameter ist das vollständig definierte Impulsobjekt[21], der zweite gibt den Kanal des Signalgenerators von 1 bis 16 an – jedes Bit entspricht einem Kanal. Die Ausgabe kann auf auch mehreren Kanälen erfolgen, indem ein logisches Oder der einzelnen Bits gebildet wird:

```
/* Parameterdefinition hier ...

 erzeuge Impuls dds::ttl_dds_pulse *puls
 mit Bitmuster 0x00000001, Maske 0xFFFFFFFF
 und starte Zähler Nr. 1 */

 int bits = 0x00000001;
 int mask = 0xFFFFFFFF;
 bp->make_cap(&pulse, ... ,bits,mask,0x01);

/* Ausführung des Impulses auf Kanal 1 und
 Kanal 9: 0x0001 | 0x0100 = 0x0101 */

 bp->run(*pulse,0x0101);
```

M-ActION erwartet auf jedes `bp->run` eine Synchronisation aller Signalgeneratoren durch `bp->sync()`. Dieser Befehl sichert, dass die einzelnen Radiofrequenz- und Digitalpulse zueinander den richtigen Zeitbezug aufweisen. Ein fehlendes Kommando würde zum Auftreten von Artefakten vorangehender Impulse führen – diese würden einander überlappen oder sich vollständig überdecken.

Zum Abschluss dieses Abschnitts wird ein kurzes Codebeispiel präsentiert. Es wird eine einfache Impulssequenz erzeugt, die auf drei Kanälen ausgeführt wird und Digital- sowie Radiofrequenzimpulse enthält. Der Signalverlauf in Abb. 3.13 soll nachgebildet werden. Ein Edge- und ein Cap-Impuls werden erzeugt, die Bitmuster `0x01010101` und `0x00000001` ausgegeben und die Zähler PMT 1 und PMT 2 gestartet; die Kenngrößen der Impulse sind in der Tabelle in Abb. 3.13 angeführt.

```
// Initialisierung der Parameter- und Impulsobjekte
```

18 Die Standardwerte sind `ttl_word = 0x00000000` und `ttl_mask = 0xFFFFFFFF`

19 Es handelt sich um „überladene Routinen“. Anhand der vom Benutzer übergebenen Parameter entscheidet das Programm, welche Standardwerte in die Funktion einzubringen sind. Fehlt beispielsweise für einen Cap-Impuls das zweite FPA, so wird dieses automatisch mit Amplitude, Frequenz und Phase Null gesetzt.

20 Der Marker `0b` identifiziert eine Zahl in Binärschreibweise.

21 Aus den Beispielen sind dies `edge`, `cap`, `sh_medium`, `sh_slow` und `wait`.

Edge-Impuls	
Frequenz	100 MHz
Phase	0 °
Amplitude	80%

Cap-Impuls	
Frequenz	200 MHz
Phase	45 °
Amplitude	100%
Dauer	10 μs

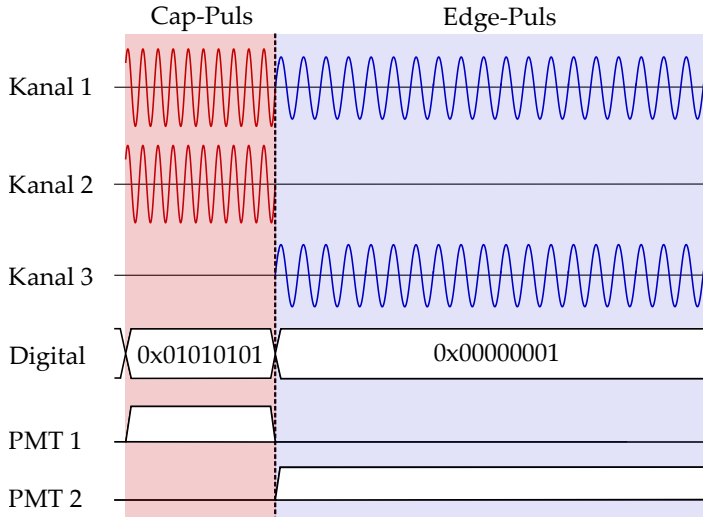


ABBILDUNG 3.13: Durch Programmcode nachzubildender Signalverlauf – die Kenngrößen der zu erzeugenden Cap- und Edge-Impulse sind in der Tabelle angeführt. Der Cap-Impuls ist rot hinterlegt, während der Edge-Impuls blau dargestellt ist.

```
dds::FPA *cap_fpa = NULL;  // FPA's für Cap und Edge
dds::FPA *edge_fpa = NULL;
dds::Time *cap_time_off = NULL;  // Impulsdauer für Cap
dds::dds_ttl_pul *cap_puls = NULL;  // Objekte Cap und Edge
dds::dds_ttl_pul *edge_puls = NULL;

// Erzeugung der Parameter nach Tabelle

  bp->use_time(&cap_time_off,10); // 10 µs Impulsdauer
  bp->use_fpa(&cap_fpa,[200,45,100]); // FPA für Cap-Impuls

  bp->use_fpa(&edge_fpa,80,0,100); // FPA für Edge-Impuls

/* Erzeugung der Impulsobjekte
 ttl_mask = 0xFFFFFFFF für beide Impulse */

  bp->make_cap(&cap_puls,cap_fpa,cap_time_off,
                                  0x01010101,0xFFFFFFFF,0x01);
  bp->make_edge(&edge_pulse,edge_fpa,1,0xFFFFFFFF,0x02);

// Ausgabe der Impulse

  bp->run(&cap_pulse,0x0003) // Cap auf Kanal 1 und 2
  bp->sync()
  bp->run(&edge_pulse,0x0005) // Edge auf Kanal 1 und 3
  bp->sync()
```

3.2.3 Mögliche Fehlerquellen bei der Programmierung von Impulsen

Da das Schreiben von Programmen zwangsläufig mit Fehlern verbunden ist, ist deren Behandlung ein wichtiger Bestandteil eines jeden Systems. M-ActION gibt einige Fehlerinformationen über eine serielle Schnittstelle nach dem RS232-Standard aus, welche direkt mit dem Computer verbunden wird – die Konfiguration der Schnittstelle ist in Abschnitt A.1.3 beschrieben. Nicht alle Fehler führen zur Ausgabe einer Warnung sondern bringen M-ActION direkt zum Absturz[22], sodass keine weitere Sequenz ausgeführt werden kann und jede Kommunikation mit dem Kontrollprogramm fehlschlägt. Das System läuft in eine Endlosschleife, welche es nicht selbst verlassen kann, sodass ein Neustart notwendig ist, um die Funktionalität wieder herzustellen. Folgende Fehlerquellen wurden im Laufe dieser Arbeit ausfindig gemacht:

- Der häufigste Fehler ist das Wiederbeschreiben eines Impuls-, Zeit oder *FPA*-Objektes durch erneuten Aufruf der in Abschnitt 3.2.2 beschriebenen `use`- und `make`-Routinen. Bevor die eigentliche Konfiguration des Objekts vorgenommen wird, überprüft M-ActION, ob die Instanzierung auf `NULL` durchgeführt wurde. Ist die Konfiguration bereits erfolgt, so ist die Zeiger-Adresse von `NULL` verschieden – eine Rekonfiguration wird verhindert und eine Fehlermeldung ausgegeben. Die Aktualisierung eines Objekts ist indirekt über eine Zuweisung aus einem Vektor möglich und wird in Abschnitt B.4 diskutiert[23].

- Bei der Anpassung der Impulsdauer können Fehler während der Initialisierung der Zeitparameter auftreten. Dieser ist für jeden Impuls nach Tab. 3.3 durch $1.4\,\mu$s nach unten beschränkt. Die `use`-Routine für Zeitparameter überprüft diesen Wert und gibt eine Fehlermeldung aus, wenn diese Bedingung verletzt wird.

- bei Shaped-Impulsen ist die Anstiegszeit der Signalflanken zu beachten – die Flanken dürfen einander nicht überlappen. Es wird empfohlen, eine Funktion zu implementieren, welche diese Zeitparameter berücksichtigt und sämtliche Änderungen überwacht. Für die vordefinierten `med_shaped` und `slow_shaped` wird eine Fehlermeldung gesendet.

- fehlende Synchronisierungen `bp->sync()` können zu einer Hintereinanderausführung der `bp->run()`-Befehle führen. Ein fehlendes Kommando würde dazu führen, dass die einzelnen DDS-Karten unsynchronisiert zueinander sowie auch zu den Logikausgängen ausgeführt werden. Dies wird bereits durch die Software verhindert; tritt kein `bp->sync()` nach jedem `bp->run()` auf, so wird die Sequenz nicht ausgeführt. Eine Fehlermeldung wird jedoch nicht ausgegeben, sodass Fehler dieser Art schwer zu identifizieren sind.

22 Ein Absturz des Systems kann durch einen *Ping* der Netzwerkschnittstelle detektiert werden.

23 Es wird mit einem neu definierten Vektortyp `dds::looped_setting` gearbeitet, der Daten schrittweise abarbeitet und dabei die Objekte `dds::FPA` und `dds::Time` laufen aktualisiert

3.2.4 Kommunikation zwischen Steuerung und M-ActION

Zwischen dem Impulsgenerator und dem Kontrollcomputer findet ein ständiger Datenaustausch statt. Für jeden an M-ActION gesendeten Befehl wird eine entsprechende Antwort zurückgegeben. Die Kommunikation erfolgt über eine Ethernet-Schnittstelle auf der Netzwerkadresse 192.168.2.5, Port 6007[24] erfolgt via Ethernet. Die übertragenen Daten sind nach dem *MessagePack*-Standard [32] formatiert, was einen sprach- und plattformunabhängigen Austausch ermöglicht. Damit ist der Entwurf neuer Software nicht an die Programmiersprache des Systems gebunden und kann stattdessen frei gewählt werden.

Jeder an M-ActION gesendete Befehl muss jedoch eine gewisse Struktur aufweisen, um verarbeitet werden zu können. Diese lässt sich grundlegend durch Abb. 3.14 darstellen. Der Befehl besteht aus einem Kommando, das die vom Impulsgenerator auszuführende Aktion beschreibt und über die Belegung der Felder „Daten“ und „Zusatz“ entscheidet. Eine Übersicht über die zum Betrieb des von M-ActION notwendigen Kommandos ist in Abb. 3.14 angeführt.

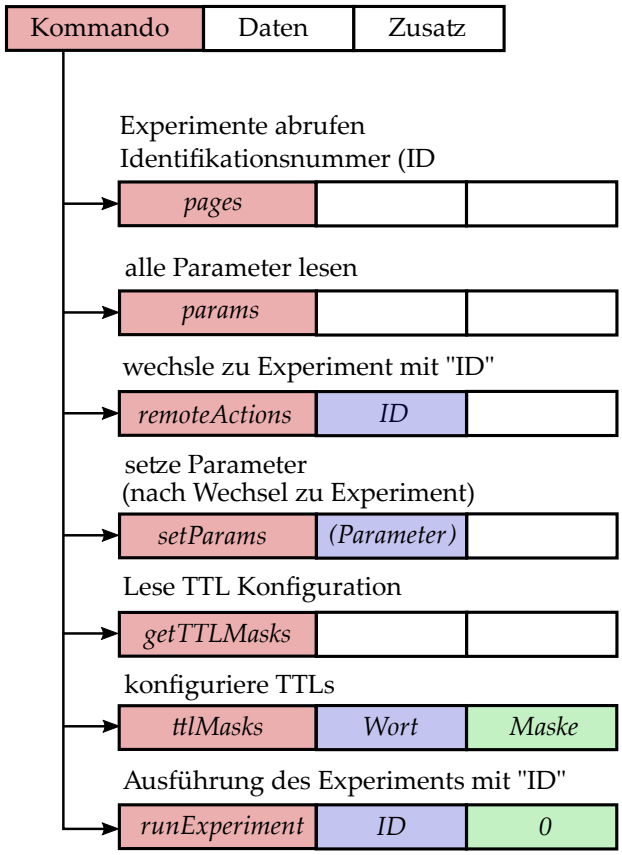


ABBILDUNG 3.14: Aufbau eines Befehls an M-ActION – der Befehl besteht aus dem Kommando, den zugehörigen Daten und einem Zusatzfeld. Abhängig vom Kommando werden die übrigen Felder belegt.

24 Die Adresse 192.168.2.5 ist im Programmcode von M-ActION konfiguriert und kann jederzeit angepasst werden – siehe hierzu Abschnitt B.3.

Die Rückgabe der einzelnen Befehle wird an dieser Stelle kurz diskutiert. Zurückgegebene Daten, die keine relevante Information über experimentspezifische Konfigurationen liefern, werden ausgespart. Anhand eines Pseudocodes wird das Datenformat beschrieben.

- *pages* – es wird eine Liste ausgegeben, die den Namen des Experiments, dessen eindeutig zugeordnete Identifikationsnummer (ID) und die ID's der verwendeten Parameter enthält.

```
Experimente = pages() # Abruf der Experimente, ist Liste
for ID in Experimente: # Exp.t-ID ist Position in Liste
  Name = Experimente[ID][0] # Name des Experiments
  ParameterIDs = Experimente[ID][2] # IDs der Parameter
```

- *params* – der Befehl liefert alle von M-ActION unterstützten Parameter für alle Experimente. Ein einzelner Parameter wird durch eine eindeutige ID, seinen Namen (kann mehrfach identisch auftreten) und seinen aktuellen Wert beschrieben. Ohne den Aufruf von *pages* ist die Parameterliste jedoch wertlos, da keine Zuordnung zu einem Experiment vorgenommen werden kann.

```
ParameterListe = params() # Abruf aller Parameter (Liste)
for ID in ParameterListe:
  Name = ParameterListe[ID][1][0] # Name des Parameters
  Wert = Experimente[ID][0][1] # Wert des Parameters
```

- *getTTLMasks* – der Rückgabewert enthält die aktuell gesetzte TTL-Maske und das TTL- Wort als Ganzzahl.

```
TTLs = getTTLMasks() # Abruf der TTLs (Maske und Wort)
Wort = TTLs[0] # TTL-Wort
Maske = TTLs[1] # TTL-Maske
```

- *runExperiment* – die Zählerstände von M-ActION werden zurückgegeben. Da dieser Befehl ein Experiment ausführt erfolgt die Rückgabe erst nach dem Auslesen der Zähler am Ende einer Sequenz.

```
Ergebnis = runExperiment() # Ausführung des Experiments
Mittelwert = Ergebnis[0][0] # Mittelwert der Zählerstände
# ist bei Mehrfachausführung eines Exp. relevant
Zählerstände = Ergebnis[1][0] # Rohdaten der Zähler
```

3.2.5 Aktuelle Grenzen des Systems

Das M-ActION System wird an der ETH Zürich bereits erfolgreich verwendet, jedoch befindet es sich in laufender Entwicklung. Einige Änderungen sind jedoch notwendig, bevor ein Einsatz in Innsbruck möglich ist. Diese sind hier kurz zusammengefasst.

- zur Zeit ist die Erzeugung von ungefähr 80 Impulsen in einer Sequenz als Schleife (siehe Abschnitt B.4.2) möglich, ohne dass ein Fehler auftritt. Durch Angabe von Einzelimpulsen lässt sich dieses Limit auf 700 erweitern, doch auch diese Zahl ist bei Weitem nicht zufriedenstellend. Dieser Fehler ist dem Programmcode der DDS-Schaltungen zuzuschreiben; an diesem wird laufend gearbeitet. Die neueste Version ermöglicht bereits die Ausgabe von 2000 Impulsen – dieser ist in Innsbruck zur Zeit nicht verfügbar und wird an der ETH Zürich getestet. Außerdem wurde auf das Ansprechen eines integrierten Speichers verzichtet, der die Umsetzung noch längerer Impulssequenzen ermöglicht. Um beispielsweise in Gatterstichproben-Experimenten einen Abfall in der Gattergüte erkennen zu können, ist ein Minimum von 10000 Impulsen veranschlagt [5].

- Verbesserungen an den DDS-Schaltungen werden laufend vorgenommen. Da die Entwicklung an eine speziell auf FPGA-Entwicklung spezialisierte Firma übergeben wurde, sind bereits einige Veränderungen in der jüngsten Version enthalten. Diese Schaltung befindet sich aktuell in der Testphase an der ETH Zürich und steht uns in Innsbruck noch nicht zur Verfügung.

- Während eines Funktionstests wurde ein Fehler in der Verarbeitungsroutine der PMT- Zählerschaltungen offensichtlich – ein zusätzliches Detektionsereignis mit Zählerstand 0 wird automatisch hinzugefügt. Dies beeinträchtigt eine laufende Messung nicht, da dieser Fehler durch Anpassung der Kommunikationssoftware eliminiert werden kann. Dennoch wird an der Behebung dieses Problems gearbeitet.

Kapitel 4

Aufbau zur Charakterisierung akusto-optischer Modulatoren

Alle Übergänge des in Kapitel 2 beschriebenen, auf einem Kalzium-Ion realisierten Qubits sind durch Einstrahlung von kohärentem Licht ansprechbar. Zur Zustandsmanipulation werden LaserimImpulse benötigt, die mit akusto-optischen Modulatoren (AOM) erzeugt werden. Die Impulse müssen analysiert werden, um etwaige systematische Fehler in der Impulserzeugung zu erfassen und die Auswirkung dieser Effekte auf das Qubit zu charakterisieren. Eine Analyse der Dynamik ist nur durch Aufzeichnung einer Schwebung zwischen einer Referenz und dem LichtImpuls auf einer Photodiode möglich. Dieses Interferenzsignal enthält die notwendigen Informationen über Frequenz, Phase und das Schaltverhalten des AOM. In diesem Abschnitt werden der optische Aufbau und die Vorgehensweise bei der Erfassung und Auswertung der Daten diskutiert. Zu Beginn wird in Abschnitt 4.1 der Aufbau gezeigt, der die Detektion des Interferenzsignals mit einer Photodiode ermöglicht. Anschließend wird die Vorgehensweise bei der Aufzeichnung der Daten in Abschnitt 4.2 beschrieben. Die Umsetzung der Impulssequenzen am Impulsgenerator M-ActION wird diskutiert und anhand eines Struktogramms veranschaulicht. Das Kapitel schließt mit der Weiterverarbeitung und Analyse der Daten in Abschnitt 4.3.

4.1 Optischer Aufbau

Der Aufbau zur Detektion der Schwebung wird schematisch in Abb. 4.1 dargestellt. Das Licht wird mit einer polarisationserhaltenden Glasfaser zu einem Tisch geführt. Die Polarisation wird bereits vor der Faser durch zwei Verzögerungsplatten ($\lambda/4$ und $\lambda/2$) angepasst. Am Tisch wird das Licht mit einem Proportional-Integral-Regler[1] (*PI*-Regler) stabilisiert, um Intensitätsfluktuationen zu vermeiden. Ein Teil der Intensität der ersten Beugungsordnung eines akusto-optischen Modulators[2] wird abgegriffen und mittels einer Photodiode[3] gemessen. Die Ausgangsspannung der Diode dient als Regelgröße und wird an den Eingang des PI-Regler angelegt. Ein spannungsgesteuerter Abschwächer moduliert das Radiofrequenzsignal des AOM und schließt so den Regelkreis.

1 SRS Stanford Research Systems
2 Gooch & Housego 3080
3 Thorlabs PDA36A

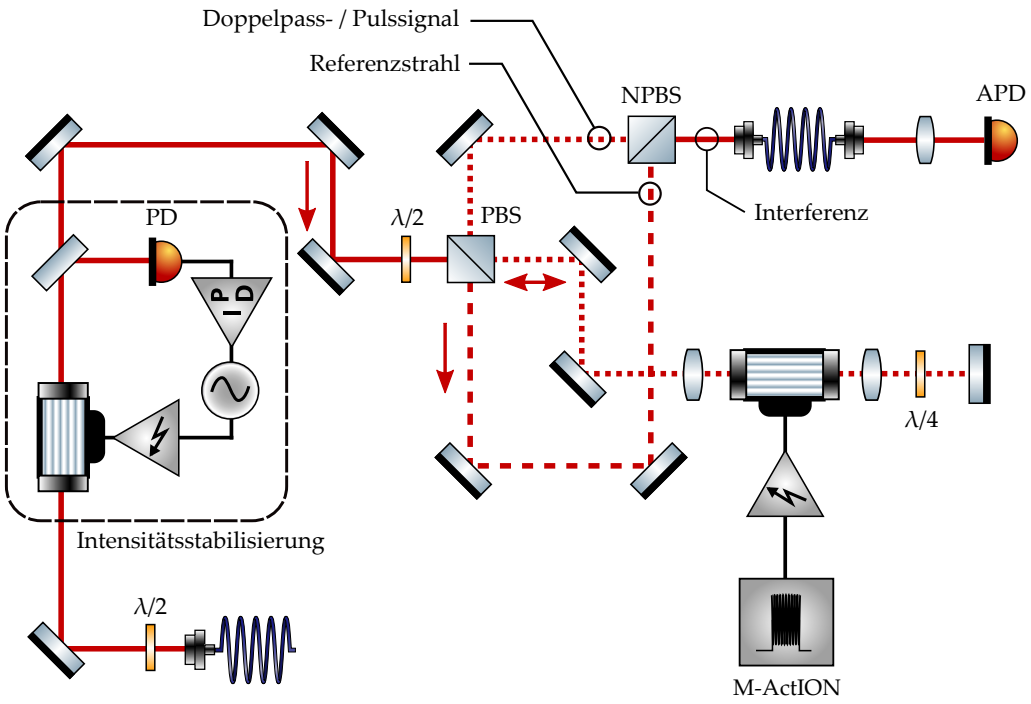


ABBILDUNG 4.1: Schematische Darstellung des optischen Aufbaus. Das Licht wird über eine Faser zum Aufbau geführt und in seiner Intensität stabilisiert. Der Regelkreis wird durch einen akusto-optischen Modulator (AOM), eine Photodiode, einem Proportional-Integralregler und einen Verstärker gebildet. Die erste Beugungsordnung wird an einem polarisierenden Strahlteiler (PBS) aufgeteilt – die Polarisationskomponenten sind punktiert (Transmission) und strichliert (Reflexion) angedeutet. Das transmittierte Licht wird dem zu charakterisierenden Modulator zugeführt, während die Reflexion den Referenzstrahl zur Erzeugung einer Interferenz bildet. Beide Strahlen werden an einem gewöhnlichen Strahlteiler (NPBS) überlagert und über eine Faser zu einer Photodiode mit integriertem Verstärker geführt (APD), die die Schwebung detektiert.

Das stabilisierte Licht der ersten Beugungsordnung wird über einen polarisierenden Strahlteiler in zwei orthogonale Polarisationskomponenten zerlegt. Die Intensität in den beiden Zweigen kann durch eine $\lambda/2$-Wellenplatte angepasst werden. Der transmittierte Anteil wird wird durch den zu charakterisierenden AOM geführt, während der reflektierte Strahl als Referenz für die interferometrische Messung verwendet wird. Der Modulator befindet sich im Fokus eines Teleskops, welches durch zwei Linsen gleicher Brennweite $f = f' = F = 200\,\mathrm{mm}$ realisiert wird. [33]. In dieser Konfiguration durchdringt das Licht den Modulator zweimal. Die in Abb. 4.2 dargestellte Linsenanordnung ermöglicht eine vom Beugungswinkel α unabhängige Rückführung der ersten Beugungsordnung in den Modulator. Der ungebeugte Strahl wird durch eine Apertur blockiert, während das gebeugte Licht ein $\lambda/4$-Verzögerungsplättchen passiert und an einem Spiegel reflektiert wird. Der Strahl durchdringt Plättchen und Modulator erneut – das Licht ist nun in seiner Frequenz

um die doppelte Radiofrequenz des AOM verschoben und orthogonal zum Referenzstrahl polarisiert. Durch Überlagerung mit dem Referenzstrahl an einem nichtpolarisierenden Strahlteiler wird das Interferenzsignal gebildet, welches zur Analyse der Impulsphasenentwicklung verwendet wird.

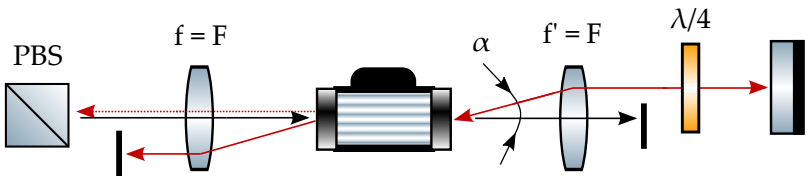


ABBILDUNG 4.2: Doppelpass-Konfiguration eines akusto-optischen Modulators. Der Aufbau erlaubt eine vom Beugungswinkel α unabhängige Rückführung des Strahls in den AOM. Ein aus zwei Linsen gleicher Brennweite $f = f' = F$ realisiertes Teleskop gleicht α unabhängig von der gewählten Radiofrequenz aus. Die erste Beugungsordnung wird an einem Spiegel reflektiert und durch den Modulator zurückgeführt; der ungebrochene Strahl wird von einer Apertur geblockt. Eine $\lambda/4$- Verzögerungsplatte dreht die Polarisation um $\pi/2$; die Strahlen vor und nach dem Durchgang durch den Doppelpass sind damit orthogonal polarisiert und können an einem polarisierenden Strahlteiler aufgetrennt werden.

Das überlagerte Licht beider Strahlen wird über eine Einmoden-Glasfaser zu einer schnellen Photodiode geführt – die Faser bewirkt neben der Strahlführung eine Maximierung des Überlapps der räumlichen Lichtmode. Zur Aufzeichnung der Schwebung wird eine Photodiode[4] verwendet, die über einen integrierten Verstärker verfügt. Die Bandbreite des Detektors muss ausreichend groß sein, um das Interferenzsignal des Doppelpass-AOM messen zu können. Der Frequenzgang der Diode ist in Abb. 4.3 gezeigt und beschreibt ein Tiefpassverhalten mit einer Grenzfrequenz von $\nu_\mathrm{G} \approx 400\,\mathrm{MHz}$ [34]. Das Ausgangssignal wird bei ν_G auf $-3\,\mathrm{dB}$ abgeschwächt; größere Frequenzen $\nu \gg \nu_\mathrm{G}$ werden stark gedämpft. Das zu messende Interferenzsignal oszilliert bei der doppelten Arbeitsfrequenz des zu untersuchenden Modulators – diese liegt im in Abb. 4.3 farblich hinterlegten Bereich. Das Signal ist für diese Frequenzen mit $\approx -20\,\mathrm{dB}$ stark abgeschwächt, jedoch mit einem Oszilloskop bei einer Amplitude von wenigen Millivolt messbar.

Im Rahmen dieser Arbeit werden drei unterschiedliche Modulatoren charakterisiert, welche in Tab. 4.1 zusammen mit den dem jeweiligen Datenblatt entnommenen Eigenschaften aufgeführt sind [35–37]. Die Arbeitsfrequenz, bei der der AOM betrieben wird, wird mit ν_C bezeichnet; die endliche Anstiegszeit, bis die Intensität der ersten Beugungsordnung eine stabile Amplitude erreicht, wird als t_r angegeben. Für das Modell Brimrose EF-270-100 sind keine Daten zur Anstiegszeit verfügbar – es handelt sich laut Hersteller hier um einen Frequenzschieber, der nicht für einen schnellen Schaltbetrieb ausgelegt ist. Vor der Charakterisierung jedes Modulators wird die Beugungseffizienz durch Anpassung des Strahlengangs maximiert; für jeden Typ werden $\approx 60\%$ Effizienz erreicht.

4 Thorlabs APD-430AM

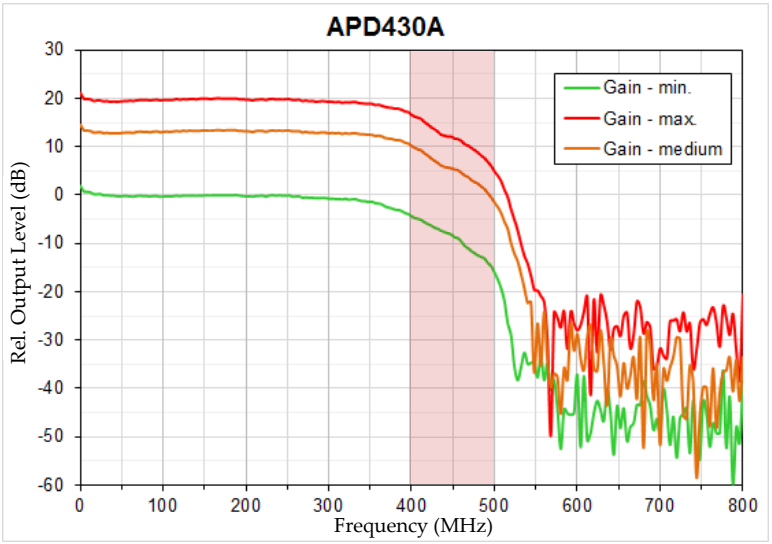


ABBILDUNG 4.3: Frequenzgang der zur Aufzeichnung des Interferenzsignals verwendeten Photodiode (*Thorlabs APD-430AM*, Grafik bearbeitet). Sämtliche Messungen werden im farblich hinterlegten Frequenzbereich durchgeführt. Das zu messende Interferenzssignal wird in diesem Bereich bereits gedämpft, ist aber im Millivolt-Bereich mit einem Oszilloskop darstellbar.

Modulator		ν_C (MHz)	Beugungseffizienz (%)	t_r (ns)
Hersteller Typ S/N	Brimrose EF270-100-729 0609 10 1299	270	60	N.A.
Hersteller Typ S/N	IntraAction ATM2351A2.14 446-357	235	$70 - 85$	40[5]
Hersteller Typ S/N	Gooch & Housego 3200-124 145-033	200	70	29

TABELLE 4.1: Spezifizierte Eigenschaften der verwendeten Modulatoren. Die Arbeitsfrequenz ν_C, die Beugungseffizienz und die Anstiegszeit t_r sind sofern verfügbar den Datenblättern entnommen.

5 Der Hersteller gibt $t_r = 151\,\mathrm{ns/mm}$ Strahldurchmesser an – es wird ein Strahldurchmesser von $\approx 0,25\,\mathrm{mm}$ angenommen.

4.2 Methodik

Zur digitalen Messung der Modulatoren ist eine ausreichend hohe Abtastrate der Signale – insbesondere des Interferenzsignals – notwendig. Die Frequenz $\nu_{\text{sig.}}$ dieses Signals entspricht der doppelten Arbeitsfrequenz und liegt im Bereich von $400\,\text{MHz}-500\,\text{MHz}$. Zur Rekonstruktion eines Signals sind nach dem Abtasttheorem von Nyquist und Shannon mindestens zwei Datenpunkte pro Periode notwendig. Diese Beziehung legt die untere Grenze der Abtastfrequenz mit $\nu_{\text{tast}} = 1\,\text{GHz}$ fest:

$$\nu_{\text{tast}} = 2\nu_{\text{sig.}}$$

Es wird $\nu_{\text{tast}} \geq 2.5\,\text{GHz}$ als Minimum gewählt – es werden so pro Signalperiode mindestens 5 Datenpunkte aufgezeichnet, womit eine Bestimmung der Impulsphasenentwicklung gewährleistet ist. Für einen Datensatz, welcher eine Sequenz aus 60 Impulsen mit einer Impuls- und Pausenzeit von $t_{\text{Puls}} = t_{\text{Pause}} = 100\,\mu\text{s}$ beschreibt, folgt

$$n_{\text{min}} = \nu_{\text{tast}} \cdot (t_{\text{Puls}} + t_{\text{Pause}}) \cdot 60 = 30 \cdot 10^6 \approx 450\,\text{MB (Megabyte)} \tag{4.1}$$

mit n_{min} der minimalen Anzahl an erfassten Datenpunkten. Die Impulspausen enthalten dabei keine relevanten Informationen und müssen entfernt werden. Die Vorgehensweise bei der Datenerfassung wird in diesem Abschnitt diskutiert.

4.2.1 Messung der Intensität des Interferenz- und Impulssignals

Die zur Charakterisierung eines akusto-optischen Modulators notwendigen Größen werden zwei Signalen entnommen. Aus der Intensität der Interferenz zwischen Referenzstrahl und dem modulierten Lichtimpuls wird die Dynamik der Phase des Impulses bestimmt. Ein zusätzliches kontinuierliches Signal ist nötig, um wie in Abschnitt 4.3.2 beschrieben eine Mischung mit der Schwebung zu bilden und eine intensitätsunabhängige Analyse des Phasenwinkels zu ermöglichen. Die Entwicklung der Impulsfläche wird aus dem Lichtimpuls selbst bestimmt; die Überlagerung mit dem Referenzstrahl liefert hier keine Information. Für unterschiedliche Ausgangsleistungen (in Prozent des Maximalwerts von $0,03\,\text{dBm}$) der Radiofrequenzimpulse werden Sequenzen mit jeweils 60 Impulsen aufgezeichnet. Es werden jeweils drei Werte für Impulsdauer und -pause gewählt und paarweise verändert, sodass pro Modulator und Leistung neun Datensätze aufgezeichnet werden. Die Dynamik der Impulsflächen und -phasen wird für die gesamte Sequenz sowie in höherer Auflösung für den ersten, zweiten und letzten Impuls der Sequenz bestimmt. Es werden die Impulsflächen- und Phasenentwicklung der gesamten Sequenz sowie in höherer Auflösung für den ersten, zweiten und letzten Impuls der Sequenz bestimmt. Das Phasensignal lässt sich für Leistungen von 70% bis 100% rekonstruieren, während bei 40% das Hintergrundrauschen einen deutlichen Einfluss auf die Phasenentwicklung zeigt – die Amplitude der aufgezeichneten Schwebung entspricht etwa dem 1,5-fachen der Rauschamplitude. Der reine Lichtimpuls dagegen ist bis 40% Radiofrequenzleistung zur Ermittlung der Impulsfläche eignet. Als Zeitparameter werden die Punkte $1.5\,\mu\text{s}$, $10\,\mu\text{s}$ und $100\,\mu\text{s}$ gewählt und alle möglichen Kombinationen gebildet.

Es sind keine Änderungen im optischen Aufbau notwendig, um zwischen der Aufzeichnung der Interferenzintensität und der des reinen Impulses zu wechseln – es ist lediglich der Referenzstrahl zu blockieren. Die Datenerfassung erfolgt mit einem Oszilloskop[6], dessen Abtastrate für jede Messung zwischen $\nu_{\mathrm{tast}} = 10\,\mathrm{GHz} - 2.5\,\mathrm{GHz}$ angepasst wird, um eine volle Sequenz aus 60 Pulsen im Speicher des Messgeräts aufzeichnen zu können.

4.2.2 Bestimmung des Frequenzgangs des akusto-optischen Modulators

Zur Bestimmung des Frequenzgangs eines Modulators bieten sich zwei Möglichkeiten an. Zum einen kann die Intensität der ersten Beugungsordnung in Einzelpass- oder Doppelpass-Konfiguration als Funktion der Frequenz gemessen werden, zum anderen ist eine elektronische Analyse mit einem Netzwerkanalysator[7] möglich. Da die Beugung des Lichts vom Einfallswinkel des Strahls zum Modulator abhängt, liefern beide Methoden abweichende Ergebnisse. Zudem deckt der Netzwerkanalysator lediglich einen Frequenzbereich von $0\,\mathrm{MHz}$ bis $200\,\mathrm{MHz}$ ab – für den Modulator Gooch & Housego 3200-124 wird die vom Hersteller spezifizierte Arbeitsfrequenz vom Analysator erreicht. Die Frequenz der übrigen Typen übersteigt die Bandbreite des Geräts $\nu > 200\,\mathrm{MHz}$, sodass eine elektronische Bestimmung des Frequenzgangs nur für den Modulator Gooch & Housego möglich ist.

Zur optischen Bestimmung wird ein Impuls mittels Photodiode aufgezeichnet und die Frequenz wird mit einer Schrittweite von $1\,\mathrm{MHz}$ in einem symmetrischen Bereich von $\approx 60\,\mathrm{MHz}$ um die spezifizierte Arbeitsfrequenz ν_{C} des Modulators durchgestimmt. Die Intensität in der ersten Beugungsordnung ist frequenzabhängig – die gegen die Frequenz des Impulses aufgetragene mittlere Impulsamplitude liefert den Frequenzgang. Für eine elektronische Analyse wird der Netzwerkanalysator mit dem Radiofrequenz-Eingang des AOM verbunden. Gemessen werden jeweils Eingangs- und Ausgangsreflexionsfaktor S_{11} und S_{22} – die beste Anpassung wird für die spezifizierte Mittenfrequenz des Modulators erwartet.

Der Frequenzgang wird durch die in Gl. 4.2 gegebene Lorentz-Funktion beschrieben. In diese Funktion gehen die Amplitude C, die Arbeitsfrequenz des Modulators ν_{C} und die Breite σ ein. Als Bandbreite $\Delta\nu$ wird die volle Breite bei halber Höhe (FWHM) von $L(x, C, \nu_{\mathrm{C}}, \sigma)$ bezeichnet – diese ist durch $\Delta\nu = 2\sigma$ gegeben.

$$L(x, C, \nu_{\mathrm{C}}, \sigma) = \frac{C}{\pi} \cdot \frac{\sigma}{(x - \nu_{\mathrm{C}})^2 + \sigma^2} \tag{4.2}$$

6 LeCroy WaveRunner

7 Hewlett-Packard Modell Nr. 8753B

4.2.3 Erzeugung der Impulssequenzen

Die Charakterisierung der dynamischen Eigenschaften eines Modulators setzt die Erzeugung von Sequenzen aus Radiofrequenzimpulsen voraus. Zudem müssen die AOMs auch kontinuierlich betrieben werden, um den Strahlengang anzupassen und in die Lichtwellenleiter einzukoppeln zu können. M-ActION stellt bereits eine Reihe von Experimenten bereit, die zum Funktionstest des Impulsgenerators dienen[8]. Folgende Experimente werden verwendet:

- `dds fixed` – vordefinierter Programmcode zur Erzeugung von kontinuierlichen Signalen variabler Frequenz, Phase und Amplitude.
- `pulses looped` – neu entwickelter Code zur Erzeugung von Impulssequenzen und Einzelimpulsen, deren Amplitude, Frequenz, Impuls- und Pausezeit eingestellt werden kann. Der vollständige Code ist in Anhang B angehängt. Anhand dieses Experiments soll auch die Handhabbarkeit von M-ActION und das Arbeiten mit neu erstellten Programmcodes getestet werden.

Während einer Messung wird zwischen `pulses looped` und `DDS fixed` gewechselt, um die Interferenzintensität anzupassen, die Beugungseffizienz zu verbessern oder Änderungen an den Parametern der Intensitätsstabilisierung vorzunehmen. Neben von M-ActION verwalteten Experimenten sind zwei Programme[9] an der Erzeugung der Sequenzen und am Ablauf der Messung beteiligt und arbeiten nach dem in Abb. 4.4 dargestellten Schema zusammen.

- Zedserver – dieses Skript regelt die Kommunikation zwischen Kontrollcomputer und dem Impulsgenerator und übersetzt Benutzereingaben in ein für M-ActION verständliches Format[10].
- Grafische Benutzeroberfläche (GUI) – jede Benutzereingabe wird von diesem Programm aus vorgenommen. Es ermöglicht das Setzen von Zeitparametern, der Amplitude und der Frequenz einer Impulssequenz, die Ausgabe von Einzelimpulsen und das Wechseln zwischen Impuls- und kontinuierlichem Betrieb. Im Hintergrund übernimmt das Programm die Erfassung und Verarbeitung der Daten des Oszilloskops[11]. Neben der Interaktion mit dem Benutzer nimmt die GUI komplette Messungen automatisch vor; Zeitparameter und Radiofrequenzleistung werden selbstständig eingestellt.

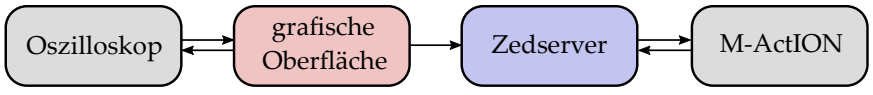


ABBILDUNG 4.4: Darstellung der Hardware-/Programmkette aus Oszilloskop (Datenerfassung), grafischer Oberfläche (Benutzereingaben), Zedserver (Kommunikation) und M-ActION (Impulserzeugung).

8 Als Experiment wird nach Kapitel 3 ein im Speicher von M-ActION hinterlegter Code zur Erzeugung einer spezifischen Impulssequenz bezeichnet.

9 Beide Programme basieren auf der Programmiersprache Python.

10 Es handelt sich um eine Liste aus Tupeln der Form `(ID,Wert)` mit der Nummer `ID`, welche den Parameter, dem ein Wert zugewiesen werden soll, eindeutig identifiziert.

11 LeCroy Waverunner, Datenerfassung via Ethernet auf Adresse 192.168.2.6.

Die Aufzeichnung der zur Charakterisierung der Modulatoren notwendigen Daten lässt sich durch das Flussdiagramm in Abb. 4.5 beschreiben. Die GUI führt den vollständigen Ablauf automatisch aus; eine Interaktion mit dem Benutzer findet nur zum Start der Messung statt. Zuerst werden die Interferenzsignale aufgezeichnet:

(1) das erste Paar aus Zeitparametern (Impulsdauer/Pause) wird gesetzt.

(2) ein Leistungspegel (40%, 70% und 100% des maximalen Ausgangspegels von M-ActION) wird eingestellt.

(3) M-ActION wird aktualisiert und das Experiment `pulses looped` ausgeführt.

(4) die Daten werden vom Oszilloskop gelesen.

(5) es wird überprüft, ob alle Leistungspegel abgearbeitet worden sind. Ist dies der Fall, so wird mit (6) fortgefahren; ist ein Pegel ausständig, so wird erneut (2) bearbeitet.

(6) es wird geprüft, ob alle Zeitparameter-Kombinationen abgearbeitet wurden. Falls nicht, wird die nächste Parameterkombination gesetzt, das Oszilloskop neu konfiguriert [12] und (2) erneut für 40%, 70% und 100% Signalpegel wiederholt. Die Datenaufzeichnung der Interferenzsignale ist abgeschlossen, sobald alle Kombinationen abgearbeitet sind; der Benutzer wird über die GUI benachrichtigt.

Nach Aufzeichnung der Interferenzsignale werden die Impulse des AOM aufgezeichnet – der Referenzstrahl wird blockiert und der eben beschriebene Ablauf wiederholt. Nach Abschluss der Datenerfassung wird die Messung des Frequenzgangs vorbereitet. Das Experiment `pulses_looped` wechselt in den Einzelimpuls-Modus und eine vordefinierte Frequenz wird an M-ActION übergeben. Der Impuls wird ausgegeben und die Daten des Oszilloskops gelesen. Es wird ein Frequenzintervall von $60\,\mathrm{MHz}$ um die Arbeitsfrequenz des AOM mit einer Schrittweite von $1\,\mathrm{MHz}$ durchgetastet. Nach Abarbeitung aller Frequenzwerte wird der Benutzer über die GUI erneut benachrichtigt.

12 Zeitbasis und Abtastrate werden angepasst, sodass die Impulssequenz vollständig aufgezeichnet wird.

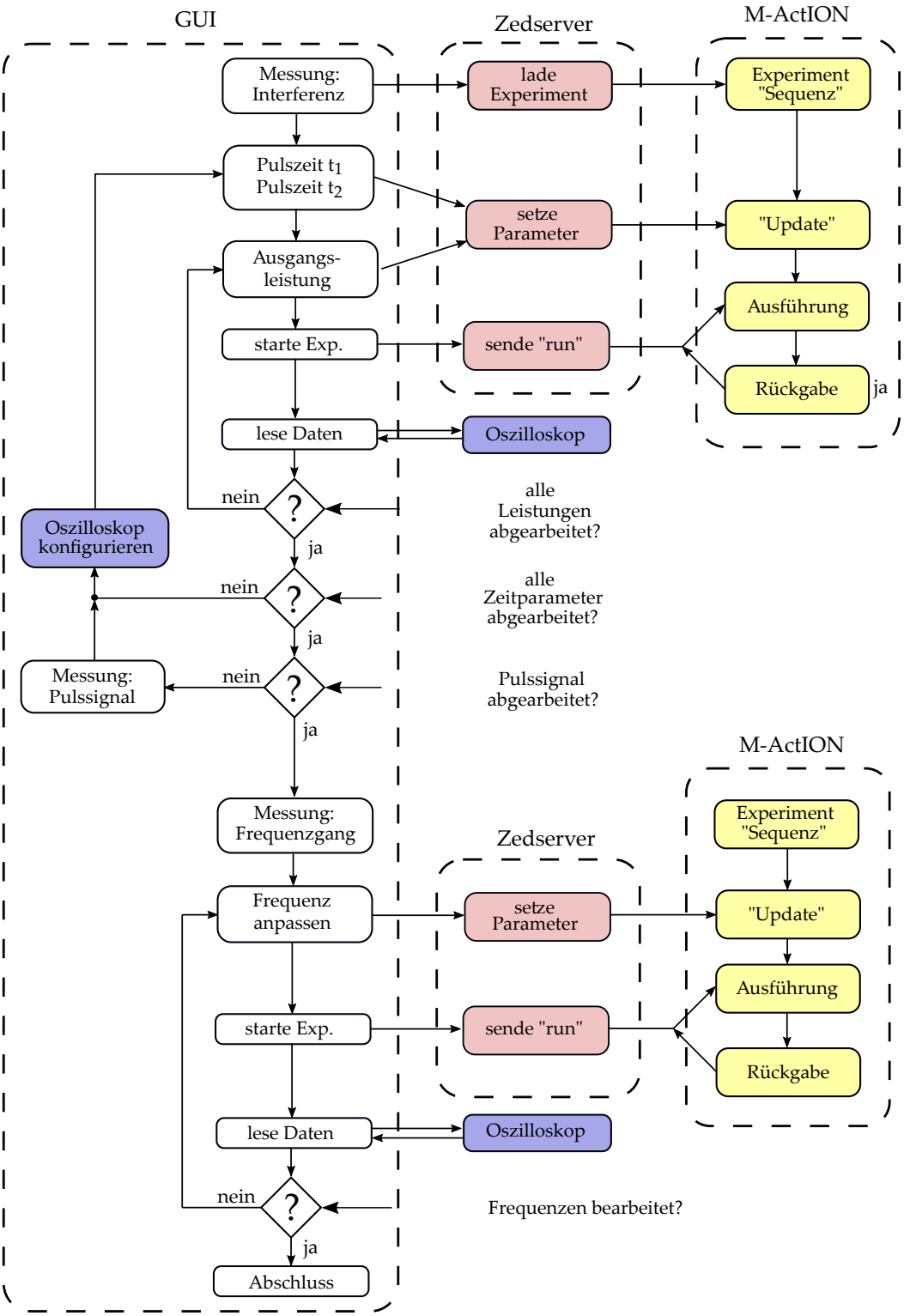


ABBILDUNG 4.5: Ablaufdiagramm der Messung zur Charakterisierung eines Modulators – die grafische Oberfläche (GUI) führt den Ablauf (weiß) im Hintergrund automatisch aus. Der Zedserver übernimmt die Kommunikation zwischen M-ActION und dem Computer. Die Datenerfassung erfolgt mit einem Oszilloskop, das via Ethernet von der GUI ausgelesen wird.

4.3 Analyse der Daten

Die Charakterisierung eines einzelnen Modulators liefert durch die hohe Abtastfrequenz des Oszilloskops eine große Menge an Daten. Der Datensatz einer Sequenz aus 60 Impulsen umfasst nach Gl. 4.1 30 Millionen Punkte – dies entspricht einer Dateigröße von ungefähr $450\,\mathrm{MB}$. Eine manuelle Auswertung des Datensatzes ist zeitlich aufwändig, weshalb eine automatisierte Routine entwickelt wurde. Diese identifiziert die Impulsflanken und isoliert die einzelnen Impulse aus der Sequenz. Die Analyse der Daten fokussiert sich auf die Bestimmung der Impulsflächen und Impulsphasen, deren dynamisches Verhalten während eines einzelnen Impulses sowie über die gesamte Sequenz.

4.3.1 Bestimmung der Impulsfläche

Zur Bestimmung der Impulsflächen wird die Intensität für eine vollständige Sequenz aufgezeichnet und die Daten werden in einzelne Impulse zerlegt. Zur Detektion der Impulsflanken wird das globale Maximum $U_{\max}$ der gemessenen Spannung bestimmt. Die Detektionsschwelle wird als $U_{\max}/2$ gewählt und die Daten werden punktweise abgearbeitet. Sobald ein Datenpunkt einen Wert innerhalb eines definierten Bereichs ΔU um $U_{\max}/2$ annimmt, wird der Index im Datensatz notiert – dieser dient als grobes Maß für die Position einer Impulsflanke. Durch paarweise Gruppierung der Indizes lassen sich so die einzelnen Impulse identifizieren. Da der Index einen Punkt auf der Flanke identifiziert, ist zur Darstellung des vollständigen Impulses ein konstanter Versatz zu berücksichtigen, um die Impulsflanken vollständig darstellen zu können. Die Flanken werden durch eine lineare Funktion als $g(t) = k \cdot t + d$ wie in Abb. 4.6 dargestellt approximiert. Folgende Größen werden daraus bestimmt:

(1) Impulsdauer t_{Puls} – durch Schnittpunkt der Geraden $g(t)$ mit der Spannung des Hintergrundrauschens U_{R} wird die Position der steigenden und fallenden Flanke bestimmt; diese werden mit τ_{r} und τ_{f} bezeichnet und legen den Start- und Endpunkt eines Impulses fest. Die Impulsdauer ist damit als $t_{\mathrm{Puls}} = \tau_{\mathrm{f}} - \tau_{\mathrm{r}}$ bestimmt.

(2) Anstiegszeit t_{r} – die Impulsamplitude U_{P} wird durch einen konstanten Fit bestimmt und der Schnittpunkt τ_{P} mit der steigenden Impulsflanke berechnet. Mit der Flankenposition τ_{r} aus (1) ist $t_r = \tau_P - \tau_r$.

(3) Impulsflächen A (Impulsausschnitt) und B (vollständiger Impuls) – mit U_{P} aus (2) und t_{Puls} aus (1) wird die Fläche A eines Ausschnitts eines Impulses durch $A = U_{\mathrm{P}} \cdot t_{\mathrm{Puls}}$ bestimmt; diese wird zur Beschreibung der Dynamik eines Einzelimpulses verwendet. Die endliche Anstiegszeit des Signals liefert nur einen geringen Beitrag zur Fläche und wird nicht berücksichtigt[13]. Dagegen entspricht B der Fläche des vollständigen Impulses – diese wird analog zu A berechnet. B dient zur Bestimmung der Flächenentwicklung entlang einer Sequenz.

Anmerkung: die hier bestimmte Fläche ergibt sich als Produkt aus Impulsdauer und der Ausgangsspannung U der Photodiode. Damit stimmen A und B nicht mit der Definition der Fläche $\mathcal{A}$ aus Gl. 2.27 überein. Die relativen Größen A/A_0 und B/B_0 mit

13 Für den kürzesten aufgezeichneten Impuls mit $t_{\mathrm{Puls}} = 1,5\ \mu$s trägt die Flanke $\approx 1\%$ zur Fläche bei.

den Mittelwerten A_0 und B_0 lassen sich jedoch direkt auf den Polarwinkel des Qubits übertragen. Im weiteren Dokument werden A und B als Impulsflächen bezeichnet.

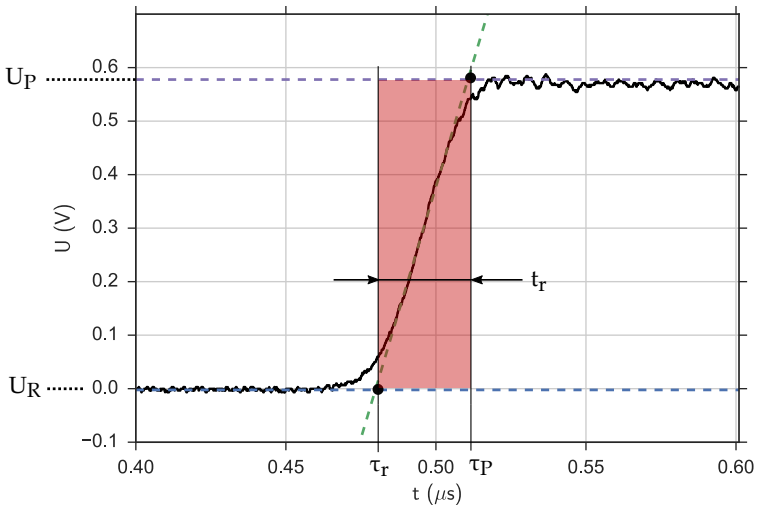


ABBILDUNG 4.6: Vorgehensweise zur Bestimmung der Impulsfläche und Signal-Anstiegszeit t_r des akusto-optischen Modulators. Die Impulsdauer t_Puls ergibt sich durch Differenzbildung der Flankenpositionen, die Impulsfläche B als Produkt aus der Signalamplitude U und der Impulslänge t_Puls.

Zunächst wird die Bestimmung der Dynamik der Impulsfläche B entlang einer Impulssequenz beschrieben. Die Fläche B wird für jeden Impuls der Sequenz bestimmt und gegenüber der Impulsposition x aufgetragen. Als relative Größe bezogen auf das Flächenmittel B_0 beschreibt ${}^{B}/_{B_0}$ den in Abb. 4.7 dargestellten Verlauf. Ein lineares Modell $B(x) = \Delta B x + B_C$ stellt eine erste Näherung dar – die Steigung ΔB wird als lineare Flächenänderung oder Flächendrift bezeichnet.

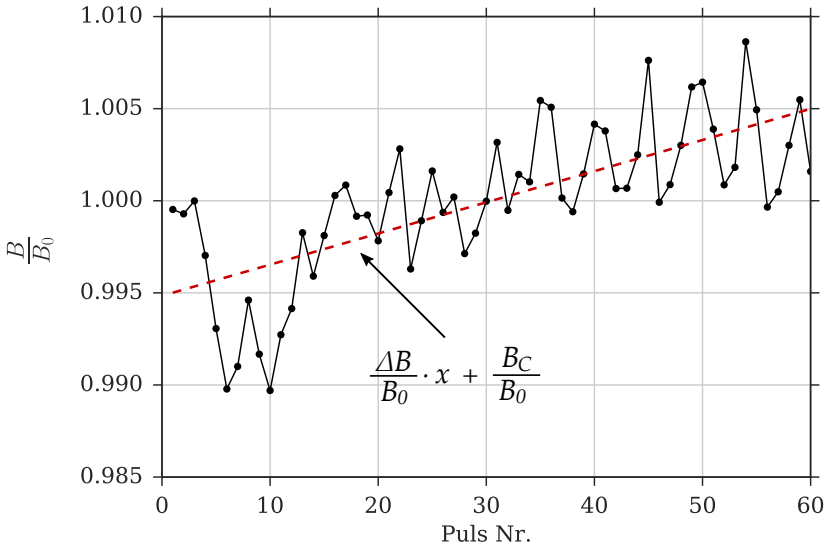


ABBILDUNG 4.7: Vorgehensweise zur Bestimmung der Flächenentwicklung in der Sequenz. Aufgetragen ist die Impulsfläche B relativ zum Mittelwert B_0. Der Verlauf der B wird linear genähert und die relative Drift ${}^{\Delta B}/_{B_0}$ bestimmt.

Diese Größen sind auch für die Laserintensität innerhalb eines Impulses zu bestimmen. Jeder Impuls wird in 10 Segmente gleicher Länge t_B zerlegt und deren mittlere Amplitude a_B ermittelt. Es wird die Fläche $A = a_\mathrm{B} t_\mathrm{B}$ des jedes Blocks berechnet – t_B ergibt sich aus der Zerlegung der Impulsdauer in $t_\mathrm{Puls} = 10 t_\mathrm{B}$. Die Impulsflanken werden detektiert und aus dem Datensatz entfernt. Abb. 4.8 veranschaulicht die Vorgehensweise der Zerlegung und Analyse für einen Impuls. Dargestellt sind die 10 Segmente, für die die Amplitude a_B bestimmt wird. Die Flächen der Abschnitte werden relativ zum Mittel A_0 als Funktion der Blocknummer x aufgetragen. Der Verlauf der A wird linear durch $A(x) = \Delta A \cdot x + A_C$ approximiert. Die ermittelten Größen werden erneut relativ zu A_0 angegeben. Die relative Drift pro Impuls und pro Segment sind identisch – es bedarf keiner weiteren Umrechnung.

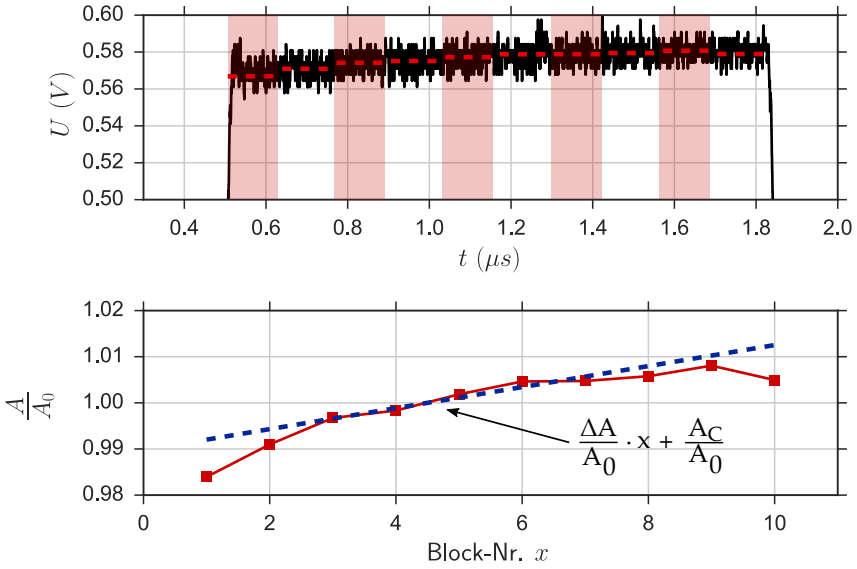


ABBILDUNG 4.8: Veranschaulichung der Vorgehensweise zur Bestimmung der Impulsfläche und Drift für einen Einzelimpuls. Die Zerlegung in Blöcke gleicher Länge ist durch die farblich hinterlegten Zonen angedeutet. Aus den Impulsamplituden und der Länge jedes Blocks wird die Fläche A berechnet und das Flächenmittel bestimmt. Durch lineare Approximation wird die Drift ΔA bestimmt.

4.3.2 Bestimmung der Impulsphase

Zur Bestimmung der Impulsphase Φ wird zusätzlich zur Interferenz f_I der Kreisfrequenz ω_I ein kontinuierliches Signal g_C mit ω_C als Referenz verwendet. Diese sind als

$$f_I = a \cdot \sin(\omega_I t + \Phi) + b \qquad g_C = c \sin(\omega_C t)$$

definiert – die Signalamplituden sind mit a und c bezeichnet; die Größe b stellt einen konstanten Hintergrund dar. Der Impulsgenerator erzeugt eine Impulssequenz und ein Referenzsignal mit der gleichen Frequenz ω_C. In Doppelpass-Konfiguration bewirkt das zweimalige Durchqueren des Modulators zu einer Frequenzverdopplung, sodass die Interferenz mit $\omega_I = 2\omega_C$ oszilliert. Durch numerisches Quadrieren von g_C wird das Referenzsignal auf die Frequenz ω_I gebracht.

Die Signale f_I und g_C werden aus den aufgezeichneten Daten gewonnen. Das Interferenzsignal f_I entspricht der Impulssequenz, die analog zu Abschnitt 4.3.1 zerlegt wird und Start- sowie Endpunkt jedes Impulses vermerkt werden. Es ist wichtig, dass das gemessene Referenzsignal g_C ebenfalls an diesen Punkten aufgeteilt wird, um den relativen Phasenbezug zwischen beiden Signalen zu messen. Diese Zerlegung ist in Abb. 4.9 dargestellt – die farblich hinterlegten Blöcke stellen zusammengehörende Impulse dar. Das quadrierte Referenzsignal g_C^2 wird wie in Gl. 4.3 numerisch mit dem Interferenzsignal gemischt.

$$\begin{aligned} f_I \cdot g_C^2 &= (a\sin(\omega_\mathrm{I} t + \Phi) + b) \cdot c^2 \sin^2(\omega_C t) = \\ &= (a\sin(\omega_\mathrm{I} t + \Phi) + b) \cdot c^2 (1 - \cos(\omega_I t)) = \\ &= b \cdot c^2 (1 - \cos(\omega_\mathrm{I} t)) + ac^2 \sin(\omega_\mathrm{I} t + \Phi) + \frac{ac^2}{2} \left(\sin(\Phi) + \sin(2\omega_\mathrm{I} t + \Phi)\right) \end{aligned} \tag{4.3}$$

In Gl. 4.3 treten Terme auf, die mit ω_I und $2\omega_I$ oszillieren. Aus diesen schnellen Schwingungen lässt sich die Information über den Phasenwinkel nur schwer gewinnen. Interessant ist der konstante Faktor $^{ac^2}/_2 \sin(\Phi)$, der eine direkte Bestimmung von Φ ermöglicht. Die hochfrequenten Oszillationen mit ω_I und $2\omega_i$ werden durch ein digitales Tiefpass-Filter F_Sinc mit einer Grenzfrequenz von $\nu_c = 100\,\mathrm{MHz}$ eliminiert. Das Filter wird durch eine Sinc-Funktion beschrieben, die mit einer Blackman-Funktion[14] gefaltet wird [38].

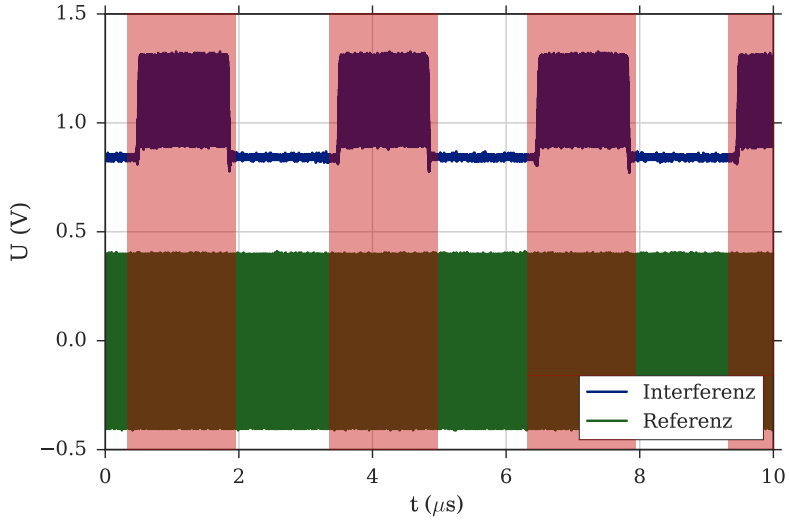


ABBILDUNG 4.9: Zerlegung einer Sequenz für das Interferenzsignal – das CW-Signal muss ebenfalls aufgeteilt werden, um die Phasenbestimmung zu ermöglichen.

Die Anwendung des Filters auf das Mischsignal Gl. 4.3 führt mit $C = {}^{ac^2}/_2$ auf Gl. 4.4.

$$F_\mathrm{Sinc}(f_I \cdot g_C^2) \approx b + \frac{ac^2}{2} \sin(\Phi) = b + C\sin(\Phi). \tag{4.4}$$

Dieser Ausdruck hängt lediglich von der Amplitude C und dem Phasenwinkel Φ zwischen Referenz- und Interferenzsignal ab. Der Hintergrund b enthält keinerlei

14 Das Blackman-Fenster ist numerisch durch $w(n, M) = 0,42 + 0,5\cos\left({}^{2\pi n}/_M\right) + 0,08\cos\left({}^{4\pi n}/_M\right)$ gegeben. M ist die Anzahl der Datenpunkte; $w(n, 100)$ erzeugt eine Fensterfunktion mit 100 Datenpunkten.

Information und kann ignoriert werden. Da C der Amplitude der Interferenz entspricht wird das Ergebnis von während der Sequenz auftretenden Schwankungen der Lichtintensität beeinflusst. Aus diesem Grund wird Gl. 4.4 für jeden Impuls selbst bei konstantem Φ einen unterschiedlichen Wert liefern. Eine Aussage über die tatsächliche Entwicklung von Φ ist nicht unabhängig von C möglich. Mit folgenden Überlegungen kann dieses Problem umgangen werden:

- jedes Verschieben des quadrierten Referenzsignals um einen Zeitschritt t_i entspricht einem zusätzlichen Phasenwinkel θ_i.
- die Mischung von $f_I(t)$ mit $g_C^2(t + \theta_i)$ liefert einen Ausdruck proportional zu $\sin(\Phi + \theta_i)$ – dieser ist periodisch in 2π.
- durchläuft θ_i den Bereich $[0, 2\pi]$ und notiert man für jedes i die Amplitude von $f_I \cdot g_C^2$, so beschreibt diese erneut ein sinusförmiges Signal – dieses ist unabhängig von der Intensität, wenn diese über einen Impuls hinweg als konstant angenommen werden kann.

Ein Verschieben des Referenzsignals um den Winkel θ führt demnach zu einer messbaren Amplitudenänderung von Gl. 4.4 – dieses ist periodisch in θ. In Abb. 4.10 ist das Signal für den ersten Impuls einer Sequenz dargestellt. Für jeden Schritt wird g_C^2 um den Winkel $\theta = 0.25\,\mathrm{rad}$ verschoben. Diese Methode erlaubt eine intensitätsunabhängige Bestimmung des Phasenwinkels Φ zwischen Interferenz- und Referenzsignal. Dieser lässt sich nun wahlweise durch die Lage der Knoten, der Maxima oder direkt durch eine Kurvenapproximation ermitteln.

Dieses Verfahren wird für jeden Einzelimpuls einer Sequenz sowie innerhalb eines Impulses angewandt und die Phasenentwicklung bestimmt.

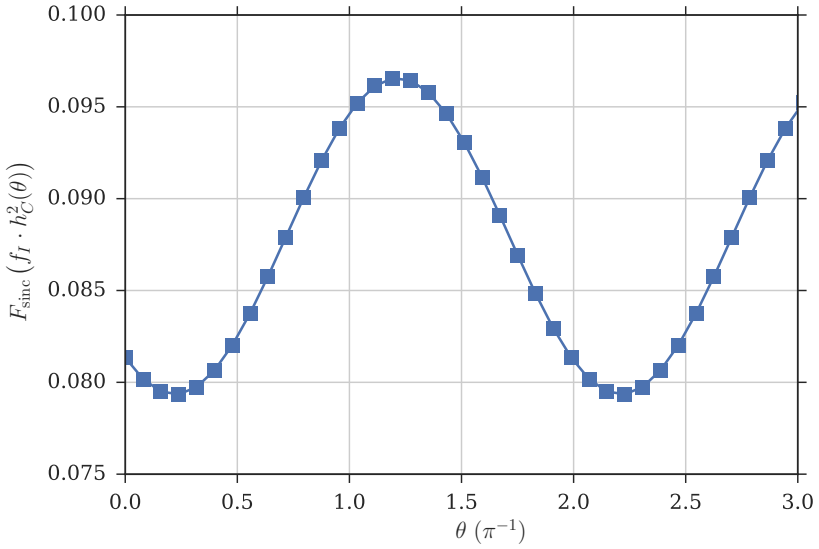


ABBILDUNG 4.10: Veranschaulichung der Vorgehensweise zur intensitätsunabhängigen Bestimmung des Phasenwinkels. Die Datenpunkte entsprechen den mittleren Signalamplituden, die für eine Verschiebung der Referenz nach Mischung und Filterung ermittelt werden.

Kapitel 5

Experimentelle Resultate

In diesem Abschnitt werden die erfassten Daten der einzelnen aktusto-optischen Modulatoren präsentiert. Für die drei in Tab. 4.1 angeführten AOMs werden Sequenzen aus 60 Radiofrequenzimpulsen erzeugt, die nach Abschnitt 4.3 ausgewertet werden. Die ermittelte Signalanstiegszeit wird in Abschnitt 5.1 beschrieben, bevor in Abschnitt 5.2 die Entwicklung von Impulsfläche und -phase innerhalb eines einzelnen Impulses diskutiert wird. Die Betrachtung der vollständigen Sequenz in Abschnitt 5.3.1 und Abschnitt 5.2.1 ermöglicht eine Abschätzung der Dynamik zwischen zwei Impulsen. Bevor in Abschnitt 5.5 Impuls- und Sequenzdaten abschließend diskutiert und zusammengefasst werden, wird der Frequenzgang in Abschnitt 5.4 dargestellt.

5.1 Anstiegszeiten der Modulatoren

Die Anstiegszeit t_r eines an einen Modulator angelegten Impulses wird wie in Abb. 4.6 dargestellt bestimmt. Hierzu werden die Position der steigenden Impulsflanke detektiert und linear durch $g(t) = kt + d$ approximiert sowie die Impulsamplitude a_0 und das Hintergrundrauschen b_0 des Signals bestimmt. Daraus lässt sich t_r nach Gl. 5.1 ermitteln:

$$t_\mathrm{r} = \frac{a_0 - b_0}{k}. \tag{5.1}$$

Für eine aus 60 Einzelimpulsen zusammengesetzte Sequenz wird für jeden Impuls die Anstiegszeit bestimmt wird und der Mittelwert t_r berechnet. Die Impulssequenzen werden für alle möglichen Kombinationen der Impulsdauer t_Puls und der Impulspause[1] t_Pause mit $t_\mathrm{Puls}, t_\mathrm{Pause} \in \{1, 5\mu\mathrm{s}, 10\mu\mathrm{s}, 100\mu\mathrm{s}\}$ aufgezeichnet; als Leistungspegel werden 70% und 100% der maximalen Ausgangsleistung gewählt. Abbildung 5.1 zeigt den Mittelwert über alle Zeitparametertupel für die drei untersuchten Modulatoren. Beide Leistungspegel sind separat dargestellt, um die Unabhängigkeit von t_r und der Radiofrequenzleistung aufzeigen. Die numerischen Werte sind in Tab. 5.1 angeführt und werden mit den Angaben der Hersteller verglichen. t_r hängt von der Ausdehnung des Laserstrahls ab – da die Strahltaille nicht bestimmt wurde, ist es fraglich, inwieweit eine Übereinstimmung mit den Herstellerdaten erzielt werden kann.

1 Die Impulspause ist der Zeitabstand zwischen zwei aufeinanderfolgende Impulse.

2 Der Strahldurchmesser ist mit $\approx 0,25\ \mathrm{mm}$ angenommen – die Taille wurde nicht bestimmt.

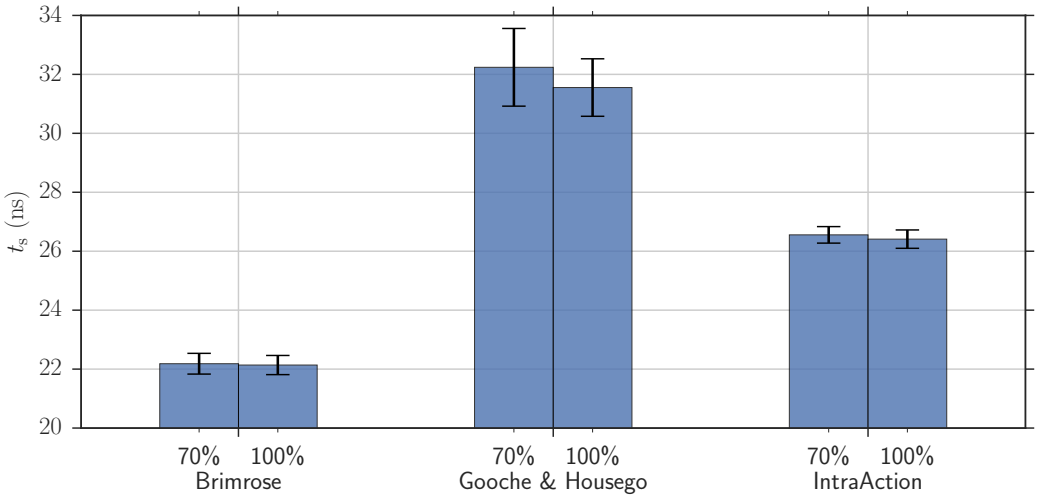


ABBILDUNG 5.1: Darstellung des Mittelwerts der Signalanstiegszeit t_r für die drei untersuchten Modulatoren. t_s wird als Mittelwert aller Impulse einer Sequenz bestimmt und das Resultat erneut über alle Kombinationen aus Impulslänge und -pause gemittelt. Die Unsicherheiten stellen die Standardabweichung vom Mittel dar.

	Ausgangsleistung		
Modell	100%	70%	Referenz
Brimrose	22,1(3) ns	22,1(3) ns	n.a. [35]
Gooch & Housego	32(1) ns	32(1) ns	29 ns [37]
IntraAction	26,4(3) ns	26,5(3) ns	40 ns [36],[2]

TABELLE 5.1: Auflistung der ermittelten Anstiegszeit t_r. Radiofrequenzleistung ermittelt. Es zeigt sich eine leistungsunabhängige Übereinstimmung der errechneten Größen.

5.2 Entwicklung der Impulsfläche und -phase eines Impulses

Thermische und elektronische Effekte eines akusto-optischen Modulators wirken sich auf die Intensität I und Phase ϕ eines Impulses aus. Es ist daher wichtig, die Entwicklung dieser Größen während eines Radiofrequenzimpulses zu charakterisieren. In Abschnitt 5.2.1 wird die Impulsfläche A betrachtet. Der erste, zweite und letzte Impuls einer Sequenz aus 60 Einzelimpulsen wird untersucht und die lineare Flächenänderung ΔA relativ zum Mittel A_0 innerhalb eines Impulses bestimmt – diese wird im folgenden Abschnitt als Drift bezeichnet.

Analog verhält es sich für die Phasenentwicklung welche in Abschnitt 5.2.2 diskutiert wird. Zusätzlich zur Sequenz ist auch das Referenzssignal zu berücksichtigen, um eine Bestimmung der Impulsphase ϕ zu ermöglichen. Die lineare Drift $\Delta\phi$ ist als absolute Größe in Einheiten von π für den ersten, zweiten und letzten Impuls der Sequenz bestimmt.

5.2.1 Entwicklung der Impulsfläche innerhalb des Impulses

Zur Bestimmung der relativen Flächendrift $^{\Delta A}/_{A_0}$ wird die aufgezeichnete Impulssequenz wie in Abschnitt 4.3.1 beschrieben analysiert. Der erste, zweite und letzte Impuls der Sequenz wird betrachtet und in jeweils 10 Teilstücke zerlegt und deren Fläche A bestimmt. Die Änderung dieser Flächenstücke wird linear durch $A(x) = \Delta A x + A_C$ approximiert und die Steigung ΔA bestimmt; der Achsenabschnitt A_C ist für die weitere Analyse nicht von Bedeutung. Durch Mittelung über alle Teilstücke wird die Fläche A_0 berechnet.

In Abb. 5.2 ist die relative Änderung der Impulsfläche für den ersten Impuls der Sequenz dargestellt. Aufgetragen ist die auf das Flächenmittel normierte lineare Drift $^{\Delta A}/_{A_0}$ in Abhängigkeit der Impulsdauer; die Impulspause hat keinen Einfluss auf den ersten Impuls.

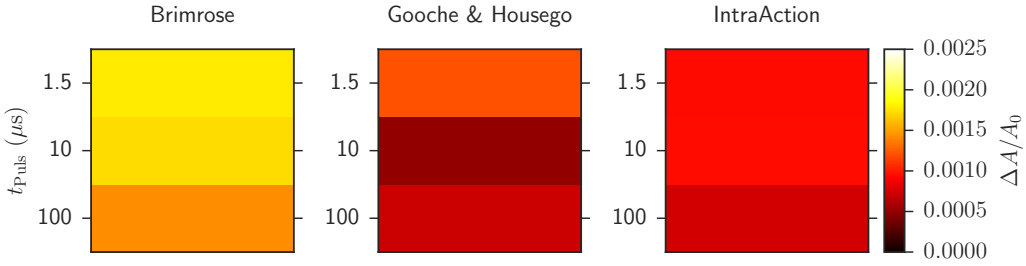


ABBILDUNG 5.2: Relative Änderung der Impulsfläche des ersten Impulses einer Sequenz für die untersuchten Modulatoren. Dargestellt ist $^{A}/_{A_0}$ in Abhängigkeit der Impulslänge t_{Puls}.

Das Modell des Herstellers Brimrose zeigt eine Drift im Bereich von $^{\Delta A}/_{A_0} \approx 0,002$, die mit zunehmender Impulsdauer geringfügig abnimmt. Für die Modulatoren Gooch & Housego 3200 and IntraAction ATM235 ist $^{\Delta A}/_{A_0} \approx 0,001$ und damit um den Faktor 2 geringer als für den Typ Brimrose EF-270. Der Einfluss der Pause zwischen zwei Impulsen wird in Abb. 5.3 und Abb. 5.4 anhand des zweiten und letzten Impulses der Sequenz deutlich. Folgende Zusammenhänge sind erkennbar:

(1) Kurze Impulse zeigen eine mit zunehmender Pause abnehmende Drift – eine Betrachtung der ersten Zeile zu $t_{\mathrm{Puls}} = 1,5\ \mu$s ist erkennbar, dass $^{\Delta A}/_{A_0}$ für die Modelle der Hersteller Brimrose und Gooch & Housego mit größerer t_{Pause} kleiner wird. Das Modell IntraAction ATM-235 folgt diesem Trend im zweiten Impuls nicht, zeigt jedoch für die unterschiedlichen Parameter nur geringfügige Änderungen der Drift.

(2) Für kurze Impulspausen nimmt die Drift mit größerer Impulsdauer t_{Puls} für alle Modulatoren ab – dies geht aus der ersten Spalte der in Abb. 5.3 und Abb. 5.4 dargestellten *Heat Maps* hervor.

(3) Bei langen Impulsen mit $t_{\mathrm{Puls}} = 100\,\mu$s steigt die Drift mit t_{Pause} an – die dritte Zeile der Grafiken veranschaulicht diesen Zusammenhang. Ein ähnliches Verhalten zeigen auch Impulse mit $t_{\mathrm{Puls}} = 10\,\mu$s.

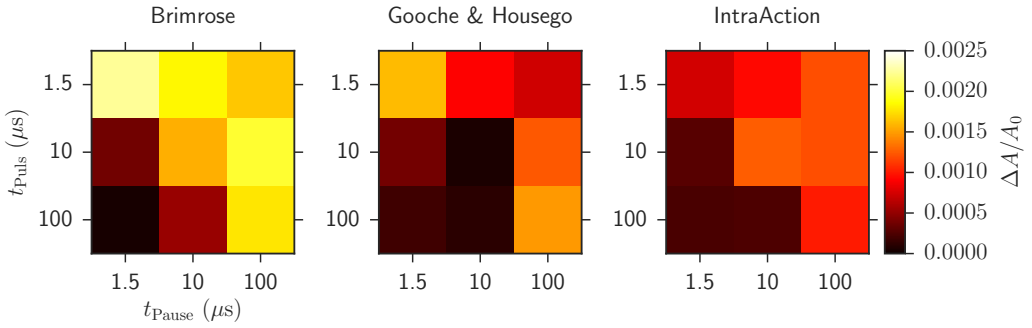


ABBILDUNG 5.3: Relative Änderung der Impulsfläche der untersuchten Modulatoren für den zweiten Impuls der Sequenz. Die Drift $^{\Delta A}/_{A_0}$ ist als Funktion der Impulsdauer t_{Puls} und Pause zwischen zwei Impulsen t_{Pause} dargestellt.

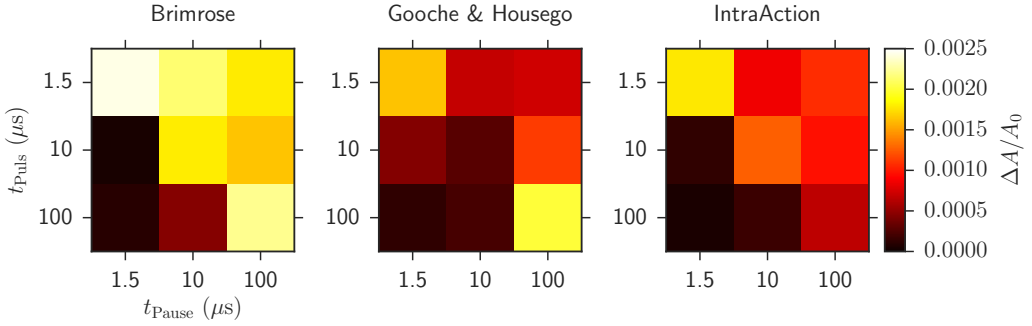


ABBILDUNG 5.4: Drift der relativen Impulsfläche des letzten Impulses für die untersuchten Modelle. Dargestellt ist $^{\Delta A}/_{A_0}$ in Abhängigkeit der Zeitparameter t_{Puls} und t_{Pause}.

Aus Abb. 5.3 und Abb. 5.4 wird deutlich, dass zwischen dem zweiten und dem letzten Impuls nahezu keine Änderung der Drift erkennbar ist. Besonders die Quadranten zu den Parametern $t_{\mathrm{Puls}} \geq 10\,\mu$s und $t_{\mathrm{Pause}} \leq 10\,\mu$s sind für alle Modulatoren nahezu identisch. Eine größere Drift ist für den letzen Impuls mit $t_{\mathrm{Puls}} = t_{\mathrm{Pause}} = 1,5\,\mu$s des Modells IntraAction ATM-235 erkennbar. Es wird vermutet, dass jeder Impuls der Sequenz nach Ausführung des ersten Impulses eine gleichbleibende Drift zeigt.

Die in der Auflistung angeführten Beobachtungen werden nun für die Extremfälle kurzer Impulse mit $t_{\mathrm{Puls}} = 1,5\,\mu$s und langer Impulse mit $t_{\mathrm{Puls}} = 100\,\mu$s genauer untersucht, indem die Abhängigkeit der Impulsfläche von der Impulspause näher betrachtet wird. In jeder der Abbildungen 5.6 bis 5.8 werden die relativen Flächen $^{A}/_{A_0}$ alle Impulse einer Sequenz mit gewählter Impulslänge und -pause als Funktion der Segmente aufgetragen. Mittelwert und Standardabweichung werden für jedes Segment berechnet und jeweils blau eingefärbt.

Zunächst wird Fall $t_{\mathrm{Puls}} = 100\,\mu$s für die Pausen $t_{\mathrm{Pause}} = 1,5\,\mu$s und $t_{\mathrm{Pause}} = 100\,\mu$s diskutiert. Die relativen Flächen der Impulse sind in Abb. 5.5 und Abb. 5.6 für alle AOMs dargestellt und zeigen deutlich die größere Drift für große t_{Pause} – dieses Verhalten wird unabhängig des untersuchten Modulators beobachtet. Es ist anzunehmen, dass es sich hier um einen thermischen Effekt handelt:

(1) der Modulator heizt sich während der Ausführung eines Impulses auf. In den Pausen zwischen den Impulsen kühlt der AOM wieder ab. Je größer t_{Puls} im Vergleich zu t_{Pause}, desto eher erreicht der Modulator ein thermisches Gleichgewicht.

(2) kurze Pausen erlauben es dem AOM nicht, sich ausreichend abzukühlen, um die Situation vor der Ausführung der Sequenz wiederherzustellen („kalter" Modulator). Der Modulator bleibt während der Sequenz im thermischen Gleichgewicht. In Abb. 5.5 ist erkennbar, dass eine signifikante Änderung der relativen Fläche nur für das erste Segment auftritt; anschließend bleibt A/A_0 nahezu konstant.

(3) während langer Impulspausen kühlt der Modulator – wenn auch nicht vollständig – ab. Jeder Impuls findet eine Situation vor, die der des ersten Impulses entspricht; der AOM ist „kalt" und muss sich während dem Impuls erneut auf Betriebstemperatur aufheizen. Die über alle Segmente kontinuierliche Änderung von A/A_0 in Abb. 5.6 stützt diese Vermutung.

Außerdem wird für Kurzimpulse mit größerer Impulspause eine abnehmende Drift bestimmt. In Abb. 5.7 und Abb. 5.8 sind erneut die relativen Flächen für die Extremfälle kurzer und langer Pausen dargestellt; die Zeitparameter sind $t_{\mathrm{Pause}} = 1,5\ \mu\mathrm{s}$ und $t_{\mathrm{Pause}} = 100\,\mu\mathrm{s}$ zu $t_{\mathrm{Puls}} = 1,5\ \mu\mathrm{s}$. Während kurzer Impulse ist es dem Modulator nicht möglich, das thermische Gleichgewicht zu erreichen, was sich in einem ähnlichen Verlauf von A/A_0 äußert. Das Einschwingverhalten ist besonders für das Modell Brimrose EF-270 ausgeprägt und kann auf die Ausbreitung der Schallwelle im Kristall zurückzuführen sein – die maximale Beugungseffizienz wird nur erreicht, wenn die Welle vollständig ausgebildet ist. Für diese Vermutung spricht, dass die Drift etwa ab der Mitte der Impulse abnimmt. Auffallend ist das Überschwingen zu Beginn und Ende jedes Impulses für den AOM IntraAction ATM-235 – diese sind in Zukunft genauer zu untersuchen.

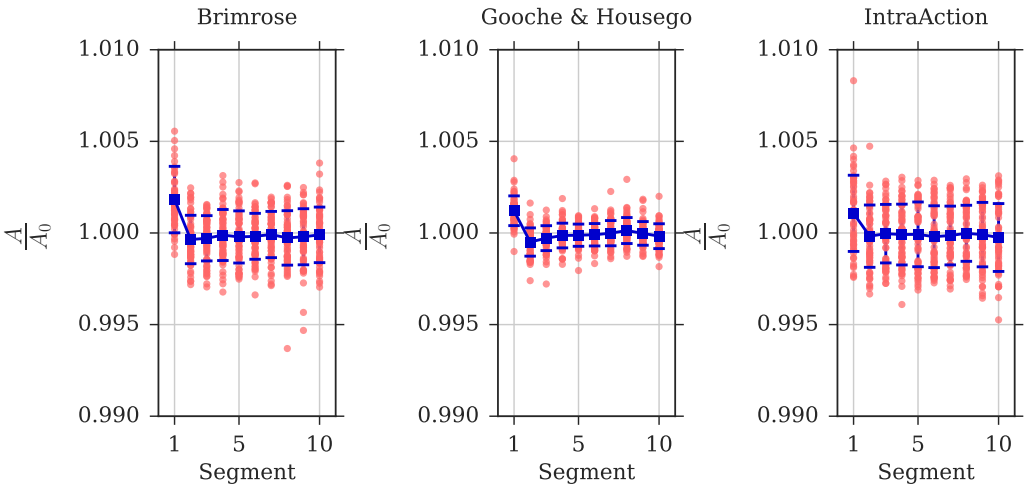


ABBILDUNG 5.5: Entwicklung der relativen Fläche A/A_0 für alle Impulse einer Sequenz mit den Parametern $t_{\mathrm{Puls}} = 100\,\mu\mathrm{s}$ und $t_{\mathrm{Pause}} = 1,5\ \mu\mathrm{s}$. Das Mittel der Daten jedes Segments wird über alle Impulse bestimmt und die Standardabweichung berechnet; Mittelwert und Unsicherheit sind in blau dargestellt.

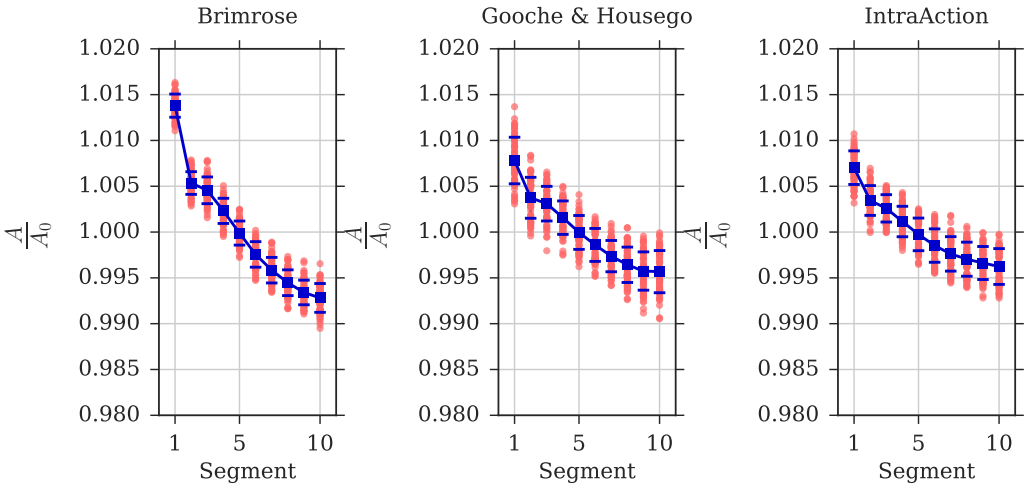


ABBILDUNG 5.6: Entwicklung der relativen Fläche für $t_{\text{Puls}} = t_{\text{Pause}} = 100\,\mu\text{s}$ für alle Impulse der Sequenz. Der Mittelwert und die Standardabweichung jedes Segments werden berechnet und sind in blau dargestellt.

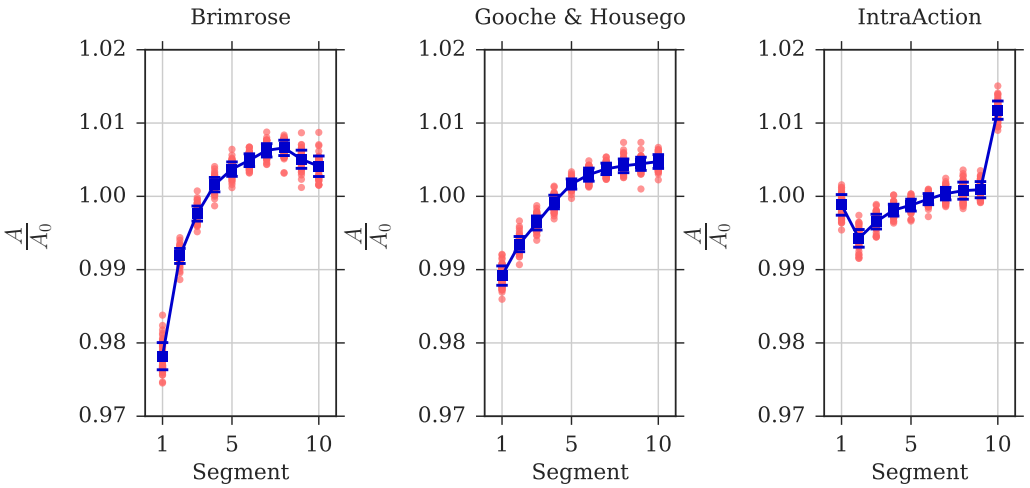


ABBILDUNG 5.7: Entwicklung der relativen Fläche für $t_{\text{Puls}} = t_{\text{Pause}} = 1,5\,\mu\text{s}$ für alle Impulse der Sequenz. Der Mittelwert und die Standardabweichung jedes Segments sind blau dargestellt.

Eine mögliche Quelle der thermischen Effekte sind die Signalumformer (Transducer) der Modulatoren, die aus dem Radiofrequenzsignal eine Schallwelle erzeugen. Ein typisches Material zur Herstellung der Transducer ist Lithiumniobat LiNbO_3, dessen akustische Eigenschaften stark temperaturabhängig sind [39]. Inwieweit sich der Kristall auf Zeitskalen von wenigen Mikrosekunden durch die angelegte Radiofrequenz erwärmt, ist jedoch fraglich und muss in Zukunft genauer untersucht werden. Auch das Kontaktmaterial, welches den Umformer mit dem Medium (Telluriumdioxid, TeO_2) jedes Modulators verbindet, kann thermischen Einflüssen unterliegen, welche die Ausbreitung der Schallwelle im AOM beeinflussen [40]. Anhand der Kurzimpulse sowie für Impulse mit großer Impulsdauer t_{Puls} und -pause t_{Pause} ist erkennbar, dass die Annahme eines linearen Modells zur Bestimmung der Drift nicht ausreichend ist und nur eine erste Näherung zur Beschreibung der Dynamik

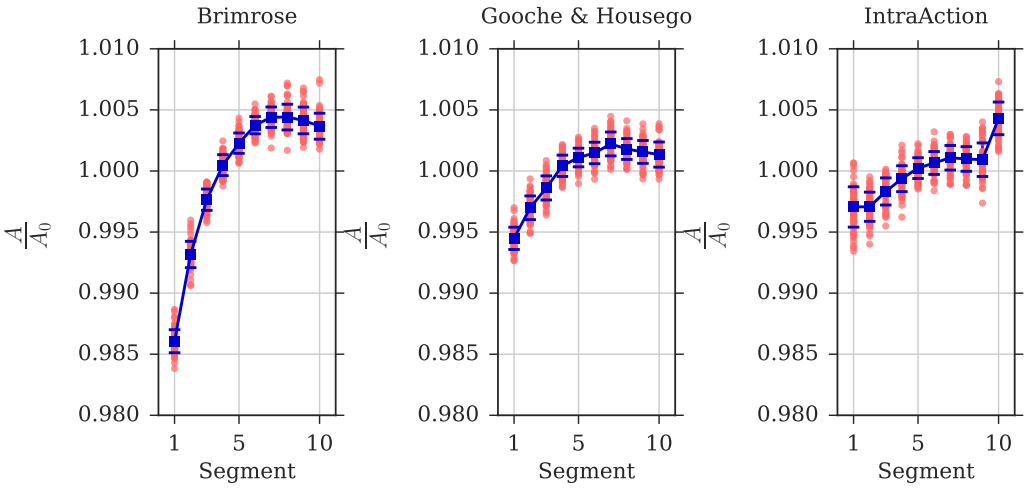


ABBILDUNG 5.8: Entwicklung der relativen Fläche für $t_{\mathrm{Puls}} = 1,5\,\mu$s und $t_{\mathrm{Pause}} = 100\,\mu$s für alle Impulse der Sequenz; Mittelwert und Standardabweichung sind blau dargestellt.

darstellt.

Während der Analyse der Daten zeigte sich, dass für zeitlich lange Sequenzen einige Impulse am Sequenzende nicht erfasst werden können. Dies ist auf ein fehlerhaftes Setzen der Trigger-Position des Oszilloskops zurückzuführen – der Speicher des Messgeräts wird vollständig gefüllt, bevor der letzte Impuls der Sequenz auftritt. Zur Auswertung wurde daher der letzte vollständig erfasste Impuls herangezogen.

5.2.2 Entwicklung der Impulsphase innerhalb eines Impuls

Die absolute Phasendrift $\Delta\phi$ wird nach Abschnitt 4.3.2 bestimmt und in Einheiten von π angegeben. Die aufgezeichneten Interferenz- und Referenzsignale werden zerlegt und gemischt. Betrachtet werden der erste, zweite und letzte Impuls der Sequenz – diese werden in 10 Teilstücke gleicher Länge zerlegt und die Phasen ϕ dieser Segmente bestimmt. Der resultierende Verlauf wird linear durch $\phi(x) = \Delta\phi \cdot x + \phi_0$ mit $x \in \{1, \dots 10\}$ approximiert – durch Verschieben des Signals in den Nullpunkt wird $\phi_0 = 0$ zu Beginn des Impulses immer erreicht. Die Phasendrift $\Delta\phi$ wird bestimmt und auf den gesamten Impuls umgerechnet[3]. In diesem Abschnitt werden die Daten werden als *Heat Map* für 100% der Radiofrequenzleistung grafisch dargestellt.

In Abb. 5.9 ist die Dynamik des ersten Impulses abgebildet – dargestellt ist $\Delta\phi$ als Mittel für jede Impulslänge t_{Puls}. Es wird für das Modell Brimrose EF-270 eine stärkere Drift von $\approx 0,02\pi$ beobachtet. Für die beiden anderen Modulatoren ist $\Delta\phi \approx 0,01\pi$ und damit um den Faktor 2 geringer. Während für den Typ Brimrose die Drift für $t_{\mathrm{Puls}} = 10\,\mu$s dominant ist, hebt sich $\Delta\phi$ für die anderen Modulatoren bei $t_{\mathrm{Puls}} = 100\,\mu$s ab.

Mit der Betrachtung des zweiten und letzten Impulses der Sequenz lässt sich eine

3 Die gesamte Phasenänderung vom Beginn bis zum Ende des Impulses entspricht dem 10-fachen des durch $\phi(x) = \Delta\phi \cdot x + \phi_0$ mit $x \in \{1, \dots 10\}$ ermittelten $\Delta\phi$ (in $\phi(x)$ ist die Drift pro Impulssegment enthalten.)

konkretere Aussage treffen. Die ermittelte Drift ist in Abb. 5.10 und Abb. 5.11 für alle Kombinationen der Zeitparameter dargestellt. Es lassen sich folgende Beziehungen feststellen:

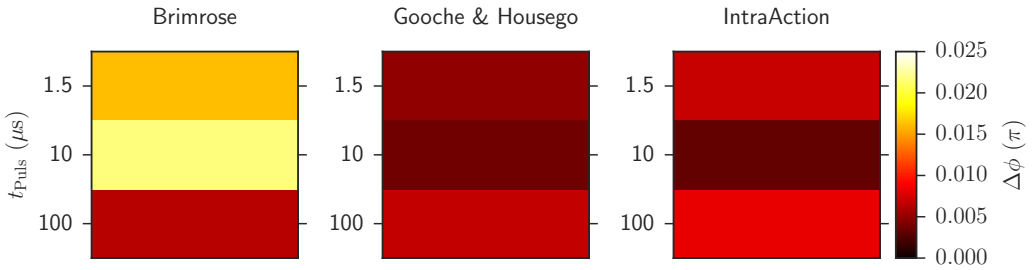


ABBILDUNG 5.9: Änderung der Impulsphase des ersten Impulses für die angegebenen Modulatoren. Dargestellt ist die auf einen Einzelimpuls umgerechnete Phasendrift $\Delta\phi$ in Abhängigkeit der Impulslänge t_{Puls} – $\Delta\phi$ ist in Einheiten von π angegeben.

(1) für kurze Impulspausen nimmt die Drift mit zunehmender Impulsdauer ab. Eine Abweichung zeigt sich hier für den Modulator Gooch & Housego zu $t_{\mathrm{Puls}} = t_{\mathrm{Pause}} = 1,5\ \mu$s.

(2) mit längerer Pause nimmt die Drift für $t_{\mathrm{Puls}} = 100\,\mu$s zu. Dieser Effekt tritt unabhängig des eingesetzten AOMs auf.

(3) für das Modell Brimrose EF-270 nimmt die Drift für $t_{\mathrm{Puls}} = 10\,\mu$s mit steigender t_{Pause} zu. Für die übrigen Modulatoren bleibt diese für diese Impulsdauer nahezu konstant.

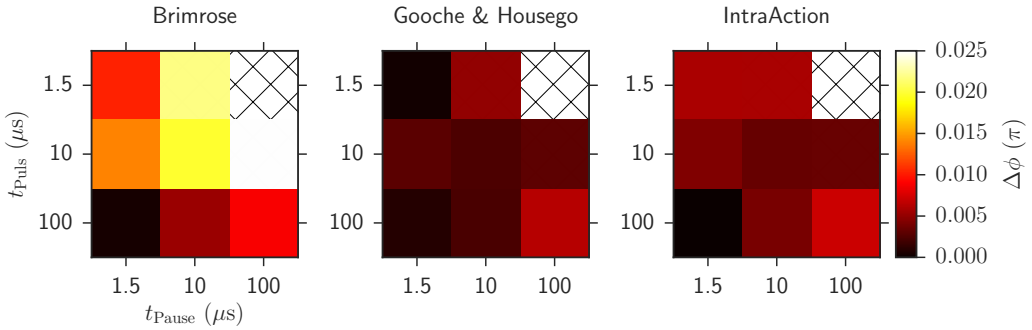


ABBILDUNG 5.10: Entwicklung der Impulsphase des zweiten Impulses einer Sequenz für die untersuchten Modulatoren. Die Phasendrift $\Delta\phi$ eines Impulses ist in Abhängigkeit der Impulslänge t_{Puls} und der Pause t_{Pause} in Einheiten von π dargestellt.

Für den Datensatz zu $t_{\mathrm{Puls}} = 1,5\ \mu$s und $t_{\mathrm{Pause}} = 100\,\mu$s ist die Phasenentwicklung nicht bestimmbar. Dies ist ein Resultat der Datenanalyse – nach Abschnitt 4.3.2 wird das Mischprodukt aus Interferenz- und Referenzsignal gefiltert, um die Summenfrequenz der Signale zu eliminieren. Das Filter reagiert auf ein Signal mit einer endlichen Impulsantwort (FIR, finite impulse response) – diese äußert sich durch

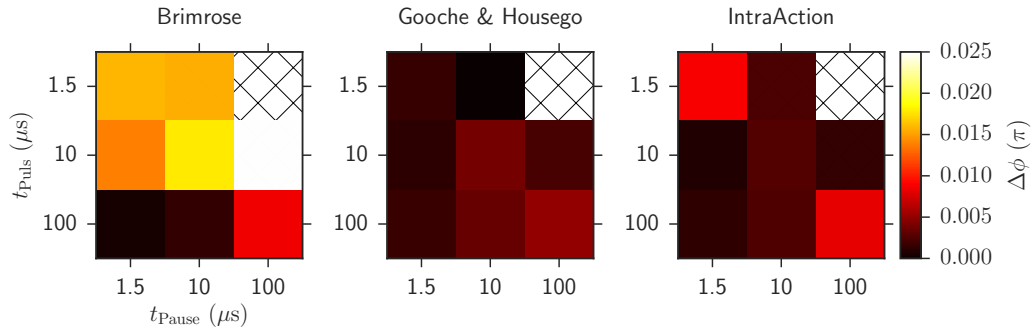


ABBILDUNG 5.11: Impulsphasenentwicklung $\Delta\phi$ des letzten Impulses einer Sequenz in Abhängigkeit der Impulsdauer t_{Puls} und Pause t_{Pause} zwischen zwei Impulsen.

ein Überschwingen im Signal und hängt von der Abtastfrequenz ab [41]. Diese wird während der Datenaufzeichnung laufend verändert, um die Aufzeichnung der vollen Impulssequenz zu gewährleisten. Für diesen speziellen Datensatz beträgt $\nu_{\mathrm{Tast}} = 2,5\,\mathrm{GHz}$, woraus sich für die Impulsantwort eine Zeit von $t_{\mathrm{FIR}} = 0,12\,\mu\mathrm{s}$ ergibt. Diese ist abzuwarten, bevor das Filter gültige Daten liefert. Für einen in 10 Segmente zerlegten Impuls ist die Länge eines Teilstücks mit $\approx 30\,\mathrm{ns}$ gegeben. Sind diese Daten noch dazu stark verrauscht, so ist eine Bestimmung der Signalamplitude nicht möglich – ein linearer Fit konvergiert nicht.

Analog zu Abschnitt 5.2.1 werden die Extremfälle kurzer und langer Impulse betrachtet. Für $t_{\mathrm{Puls}} = 1,5\,\mu\mathrm{s}$ und $t_{\mathrm{Puls}} = 100\,\mu\mathrm{s}$ wird die Abhängigkeit der Phasenentwicklung von der Impulsendauer untersucht. Für jeden Impuls einer Sequenz werden die $\Delta\phi$ bestimmt und in Abhängigkeit der Impulssegmente aufgetragen. Der Mittelwert und die Standardabweichung werden für jedes Segment berechnet – in den nachfolgenden Grafiken ist diese in blau dargestellt.

Die Abbildungen 5.12 und 5.13 zeigen die Phasenentwicklung ϕ für $t_{\mathrm{Puls}} = 100\,\mu\mathrm{s}$ zu den Pausen $t_{\mathrm{Pause}} = 1,5\,\mu\mathrm{s}$ und $t_{\mathrm{Pause}} = 100\,\mu\mathrm{s}$. Während für kleine t_{Pause} eine deutliche Phasenänderung nur vom ersten auf das zweite Segment auftritt, ist für $t_{\mathrm{Pause}} = 100\,\mu\mathrm{s}$ eine kontinuierliche Änderung entlang des gesamten Impulses feststellbar. Für $t_{\mathrm{Pause}} = 1,5\,\mu\mathrm{s}$ ist eine geringe Drift ab dem zweiten Teilstück des Impulses erkennbar, die für alle Modelle auftritt. Für beide Pausendauern wird die Streuung der Phasenwerte größer, je näher das Segment am Impulsende liegt. Wie für die Flächenentwicklung wird die Ursache für diese Beobachtungen in einem thermischen Effekt vermutet.

Die Kurzimpulse $t_{\mathrm{Puls}} = 1,5\,\mu\mathrm{s}$ werden zu den Pause $t_{\mathrm{Pause}} = 1,5\,\mu\mathrm{s}$ und $t_{\mathrm{Pause}} = 10\,\mu\mathrm{s}$ in Abb. 5.14 und Abb. 5.15 dargestellt. Unabhängig von t_{Pause} zeigen sich vergleichbare Ergebnisse. Der Phasenverlauf des Modells Brimrose EF-270 hebt sich deutlich von den übrigen Modulatoren ab. Hier ist ein Übersprechen zwischen Amplitude und Phase denkbar – der in den Abbildungen 5.7 und 5.8 für Kurzimpulse gezeigte Amplitudenverlauf belegt, dass der Einschwingvorgang nach $t_{\mathrm{Puls}} = 1,5\,\mu\mathrm{s}$ für diesen AOM nicht abgeschlossen ist. Es ist denkbar, dass dieser Effekt sich auf das Phasensignal niederschlägt.

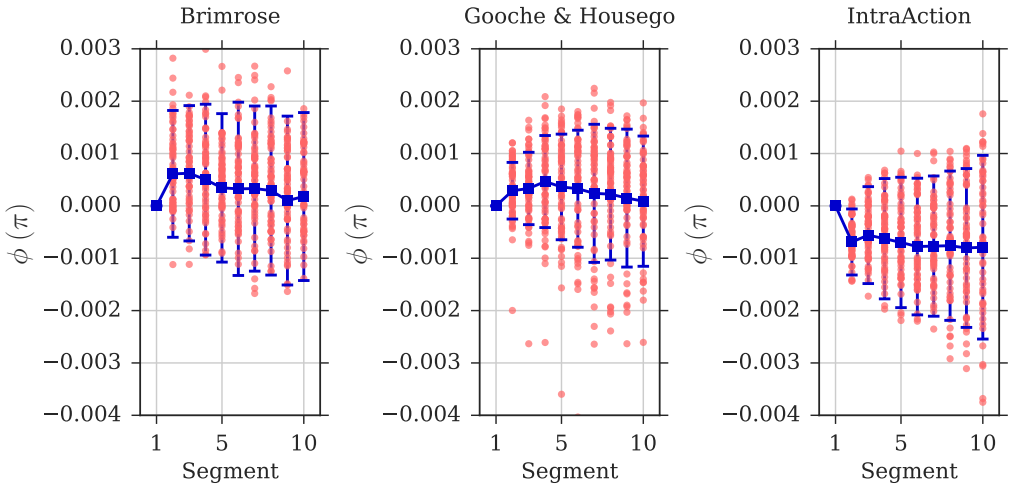


ABBILDUNG 5.12: Darstellung der Impulsphase ϕ für alle Impulse einer Sequenz mit den Parametern $t_{\mathrm{Puls}} = 100\,\mu\mathrm{s}$ und $t_{\mathrm{Pause}} = 1,5\,\mu\mathrm{s}$. Das Mittel der Phasenwerte jedes Segments wird über alle Impulse bestimmt und die Standardabweichung berechnet; Mittelwert und Unsicherheit sind in blau dargestellt.

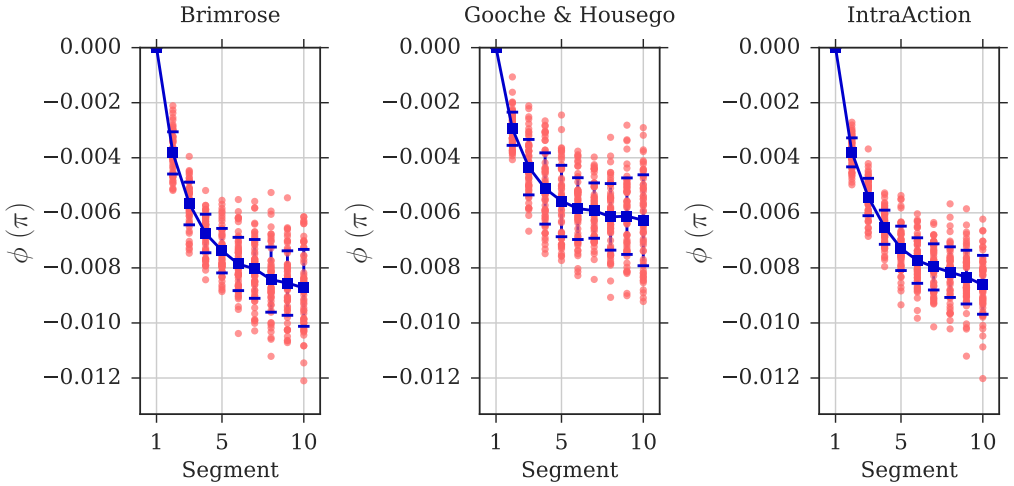


ABBILDUNG 5.13: Darstellung der Impulsphase ϕ für alle Impulse einer Sequenz mit den Parametern $t_{\mathrm{Puls}} = t_{\mathrm{Pause}} = 100\,\mu\mathrm{s}$. Mittel und Standardabweichung der Phasenwerte zu jedem Segment sind in blau dargestellt.

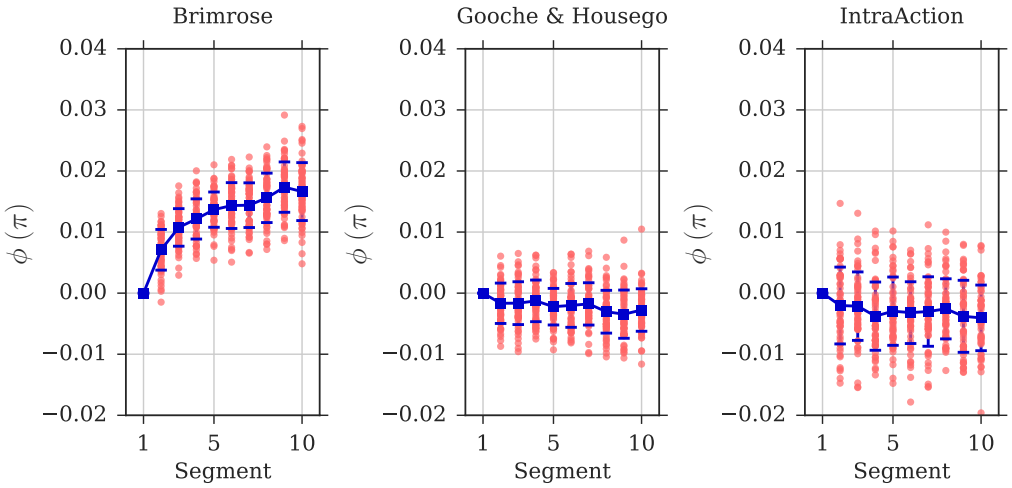


ABBILDUNG 5.14: Darstellung der Impulsphase ϕ für alle Impulse einer Sequenz mit den Parametern $t_{\mathrm{Puls}} = t_{\mathrm{Pause}} = 1,5\ \mu$s. Mittelwert und Standardabweichung jedes Segments sind in blau dargestellt.

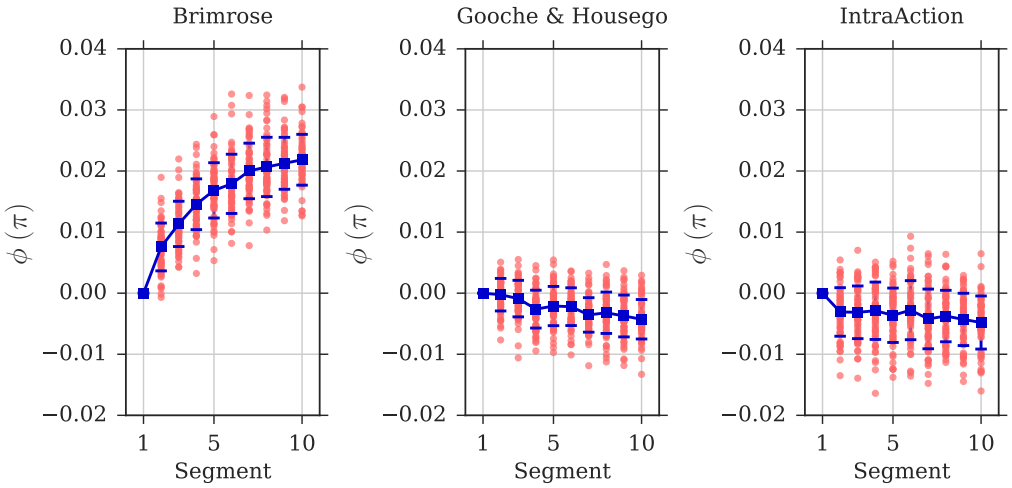


ABBILDUNG 5.15: Darstellung der Impulsphase ϕ für Impulse mit $t_{\mathrm{Pause}} = 1,5\ \mu$s und $t_{\mathrm{Pause}} = 10\ \mu$s. Aufgetragen ist ϕ für alle Einzelimpulse der Sequenz. In blau sind der Mittelwert und die Standardabweichung für jedes Segment dargestellt.

5.3 Entwicklung der Impulsfläche und -phase in der Sequenz

Während in Abschnitt 5.2 die Dynamik innerhalb eines einzelnen Impulses selbst betrachtet wurde, handelt es sich hier um die Änderung zwischen zwei aufeinanderfolgenden Impulsen. In diesem Abschnitt wird die Entwicklung der Impulsfläche V und Phase Φ entlang einer vollen Sequenz aus 60 Impulsen betrachtet. Bestimmt wird die lineare Flächenänderung $^{\Delta B}/_{B_0}$ bezogen auf das Mittel B_0 sowie die absolute Phasenänderung $\Delta\Phi$ – beide Größen werden als Drift bezeichnet.

5.3.1 Entwicklung der Impulsfläche in einer Sequenz

Die Analyse der Rohdaten einer vollständigen Sequenz wird nach Abschnitt 4.3.1 vorgenommen und die Fläche jedes Impulses bestimmt. Unabhängig des Modulators ist ein linearer Anstieg der Signalamplitude vom ersten bis zum letzten Impuls der Sequenz messbar – dies überträgt sich direkt auf die Fläche B. Abbildung 5.16 zeigt die relative Flächendrift $^{\Delta B}/_{B_0}$ für die untersuchten AOMs bei einer Radiofrequenzleistung von 100%. Ein Vergleich mit der Betrachtung der Einzelimpulse in Abschnitt 5.2.1 zeigt, dass $^{\Delta}/_{B_0}$ um einen Faktor 10 geringer ist als die Drift $^{\Delta A}/_{A_0}$ der Impulsflächen. Es werden folgende Effekte beobachtet:

(1) die Modulatoren der Hersteller Brimrose und Gooch & Housego zeigen ein ähnliches Verhalten:

- die Drift nimmt für alle Pausen mit wachsender Impulsdauer zu.
- für Impulse mit $t_{\mathrm{Puls}} = 10\,\mu\mathrm{s}$ und $t_{\mathrm{Puls}} = 100\,\mu\mathrm{s}$ nimmt $^{\Delta B}/_{B_0}$ mit zunehmender t_{Pause} ab. Eine Ausnahme bildet $^{\Delta B}/_{B_0}$ für $t_{\mathrm{Puls}} = 100\,\mu\mathrm{s}$ zu $t_{\mathrm{Pause}} = 10\,\mu\mathrm{s}$ – für diesen Datenpunkt wird bei Betrachtung der Sequenz ein außergewöhnlich starker Anstieg der Impulsintensität beobachtet. Die Ursache dieses Effekts kann nicht bestimmt werden; das Verhalten ist auf diesen Datensatz beschränkt.
- für $t_{\mathrm{Puls}} = 1,5\,\mu\mathrm{s}$ lässt sich keine Beziehung zwischen den Zeitparametern und der Drift herstellen.

(2) für das Modell IntraAction ATM-235 wird ein gänzlich anderer Zusammenhang beobachtet.

- für Kurzimpulse mit $t_{\mathrm{Puls}} = 1,5\,\mu\mathrm{s}$ ist keine Beziehung zwischen t_{Pause} und der Drift erkennbar.
- für $t_{\mathrm{Puls}} = 10\,\mu\mathrm{s}$ wächst $^{\Delta B}/_{B_0}$ mit t_{Pause} an. Dieses Verhalten ist auch für Impulse, deren Pause $t_{\mathrm{Pause}} = 10\,\mu\mathrm{s}$ für zunehmende Impulsdauer erkennbar.
- mit $t_{\mathrm{Puls}} = 100\,\mu\mathrm{s}$ nimmt die Drift mit größerer t_{Pause} geringfügig ab.

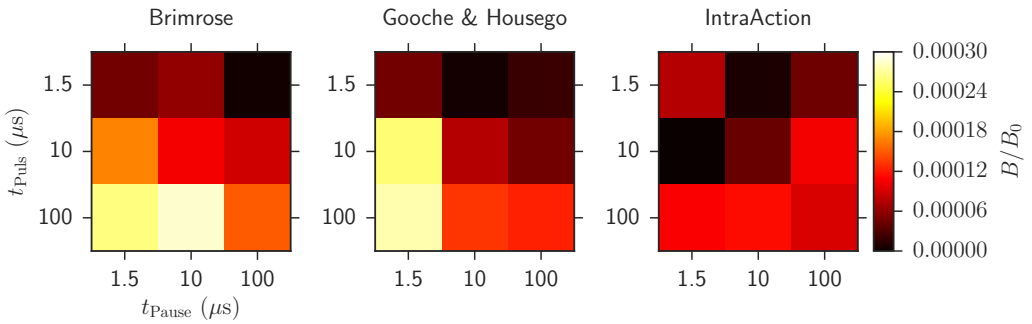


ABBILDUNG 5.16: Darstellung der relativen Flächenänderung für die angeführten Modulatoren – aufgetragen ist $^{\Delta B}/_{B_0}$ in Abhängigkeit der Zeitparameter t_{Puls} und t_{Pause}.

Als Quelle des beobachteten Verhaltens werden thermische Einflüsse vermutet. Die Abnahme der Drift für $t_{\mathrm{Puls}} = 100\,\mu\mathrm{s}$ mit zunehmender Pause spricht dafür, dass der AOM für lange Pausen vollständig abkühlt. Damit wird ein Zustand wiederhergestellt, der ähnlich dem des ersten Impulses der Sequenz entspricht – der Modulator

ist „kalt“. Ein kalter AOM ist demnach für die Drift entlang der Sequenz vorteilhaft, wirkt sich nach Abschnitt 5.2.1 jedoch negativ auf die Entwicklung der Impulsfläche aus. Die Zeitskalen, auf denen sich die Änderungen bemerkbar machen, sind hier jedoch deutlich größer als für die Betrachtung der Einzelimpulse. Es wird vermutet, dass Effekte des Signalumformers und die thermischen Eigenschaften des Telluriumdioxid-Kristalls zu berücksichtigen sind. Es wird vermutet, dass für den AOM IntraAction ATM-235 die thermische Kopplung zwischen Kristall, dem Umformer und dem Gehäuse im Vergleich zu den anderen Modellen vom Hersteller besser umgesetzt ist.

5.3.2 Entwicklung der Impulsphase entlang einer Sequenz

Das aufgezeichnete Interferenzsignal wird zusammen mit der Referenz wie in Abschnitt 4.3.2 zerlegt und die Phasen Φ bestimmt. Durch lineare Approximation $\Phi(t) = \Delta\Phi t$ wird die Phasenänderung $\Delta\Phi$ in Einheiten von π bestimmt. Die ermittelte Drift ist in Abb. 5.17 für den Leistungspegel von 100% dargestellt. Die Phasendrift scheint zufällig und nicht mit dem Zeitparametern korreliert zu sein. Unabhängig des Modells ist $\Delta\Phi \approx 10^{-4}\pi$ – zur Auflösung eine Phasenänderung in dieser Größenordnung muss das Messgerät in der Lage sein, einen Amplitudenunterschied in dieser Größenordnung zu messen. Unter Annahme eines verrauschten Spannungssignals mit einer Rauschamplitude von $\approx 0,1\%$ der mittleren Impulsamplitude ist es nicht möglich, $\Delta\Phi$ zuverlässig zu ermitteln.

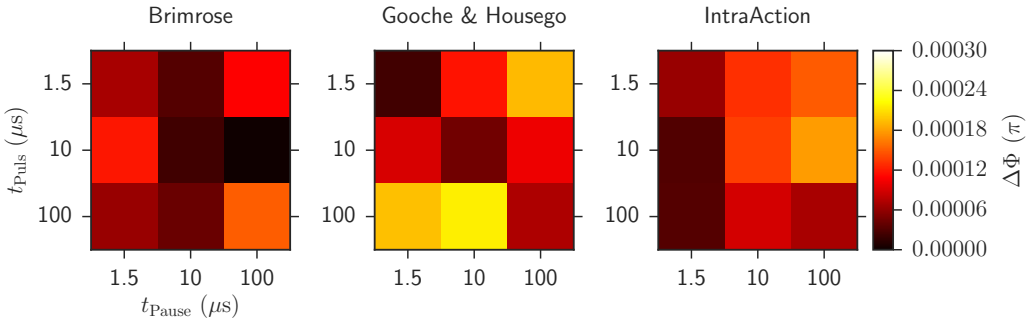


ABBILDUNG 5.17: Darstellung der Phasenänderung $\Delta\Phi$ in Einheiten von π für die angeführten Modulatoren. Die Phasendrift zwischen zwei Einzelimpulsen ist in Abhängigkeit der Zeitparameter t_{Puls} und t_{Pause} abgebildet.

Bei der Aufzeichnung des Interferenzsignals werden für zeitlich lang andauernde Sequenzen mit Zeitparametern $t_{\mathrm{Puls}} = t_{\mathrm{Pause}} = 100\,\mu$s Oszillationen im Phasensignal beobachtet. Abbildung 5.18 zeigt, dass diese für alle Modulatoren nahezu identisch mit eine Frequenz von $\approx 500\,$Hz auftreten. Da in diesem Aufbau keine aktive Rauschunterdrückung verwendet wird, kann dieses Verhalten auf das Phasenrauschen der verwendeten Glasfasern zurückzuführen sein, welche das Licht zum optischen Aufbau führen.

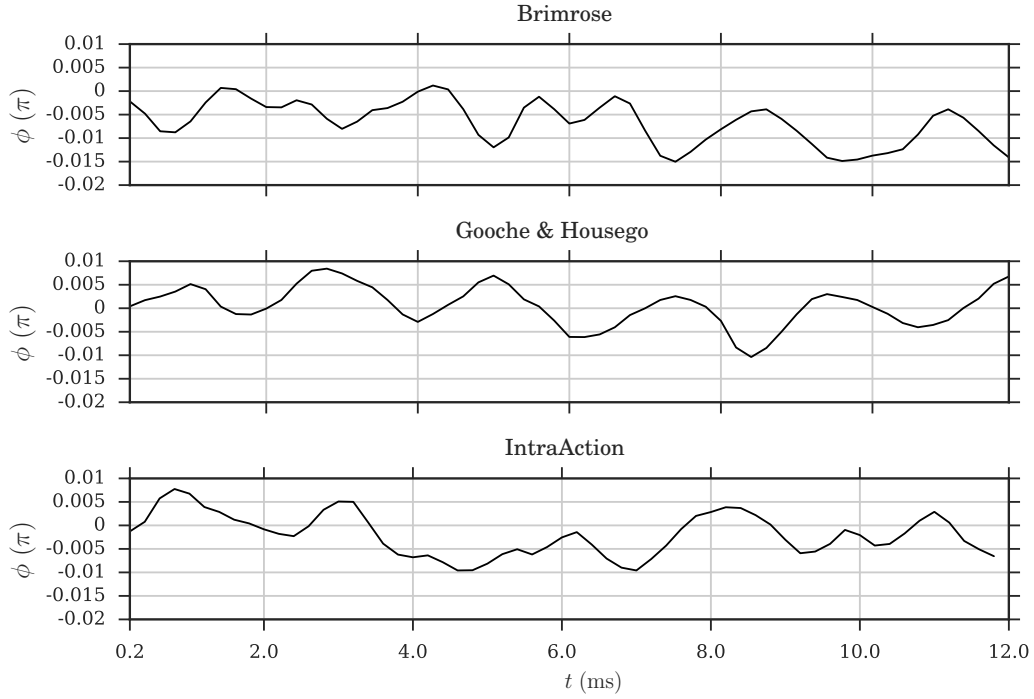


ABBILDUNG 5.18: Oszillation des Phasensignals für eine Sequenz mit $t_{\mathrm{Puls}} = t_{\mathrm{Pause}} = 100\,\mu$s für die verwendeten Modulatoren. Die Phase ϕ ist als Funktion der Zeit aufgetragen.

5.4 Frequenzgang der Modulatoren

In diesem Abschnitt wird die Arbeitsfrequenz ν_C und die Bandbreite $\Delta\nu$ der Modulatoren bestimmt. Diese stellt eine weitere charakteristische Größe dar und beschreibt, für welche Frequenz die Intensität in der ersten Beugungsordnung maximal wird. Neben ν_C und der Leistung des Radiofrequenzsignals wirkt sich auch der Winkel, unter dem das Licht auf den Kristall des AOM trifft, auf die Beugungseffizienz aus. Der Frequenzgang wird optisch nach Abschnitt 4.2.2 ermittelt. Die Intensität der ersten Ordnung wird in Abhängigkeit der Radiofrequenz auf einer Photodiode[4] aufgezeichnet und die Impulsamplitude bestimmt – das Licht trifft direkt nach dem AOM auf die Diode und passiert keine Glasfaser. Die Daten werden mit der in Gl. 4.2 definierten Lorentz-Funktion approximiert. In Tab. 5.2 sind ν_{C} und $\Delta\nu$ zusammen mit den Spezifikationen der Hersteller angeführt.

Modulator	Mittenfrequenz ν_{C} (MHz)		Bandbreite $\Delta\nu$ (MHz	
	Messung	Hersteller	Messung	Hersteller
Brimrose EF-270-100	258,6(4)	270	71,6(7)	–
Gooch & Housego 3200-124	203,1(4)	200	30,8(5)	50
IntraAction ATM2351A2.14	234,9(3)	235	18,6(2)	117[5]

TABELLE 5.2: Mittenfrequenzen ν_{C} der untersuchten Modulatoren. Angegeben sind die durch Approximation der Daten mit einer Lorentz-Funktion ermittelten ν_{C} sowie der Bandbreite $\Delta\nu$; die Unsicherheiten sind die errechneten Standardabweichungen.

4 Thorlabs PD-036A

Für das Modell IntraAction ATM-235 wird das beste Resultat erzielt; die Arbeitsfrequenz ν_C wird mit $\nu_C = 234,9(3)\ \mathrm{MHz}$ bestimmt und stimmt mit der Herstellerangabe überein. Die Bandbreite liegt mit $\Delta\nu = 18,6(2)\ \mathrm{MHz}$ deutlich unter dem vom Hersteller spezifizierten Wert. Der Modulator Gooch & Housego zeigt eine leicht größere ν_C als im Datenblatt angegeben, während der AOM Brimrose EF-270 deutlich von der Referenz abweicht. Ein Grund hierfür ist die Winkelabhängigkeit der Intensität in der ersten Beugungsordnung; der Strahlengang muss weiter optimiert werden.

5.5 Zusammenfassung der Ergebnisse

Anhand der ermittelten Daten soll nun ein Vergleich zwischen den einzelnen Modulatoren gezogen werden. Dies soll die Auswahl eines geeigneten Typs für Anwendungen in Experimenten erleichtern sowie gemeinsame Eigenschaften aufzeigen. In Abschnitt 5.5.1 werden die relative Drift der Impulsfläche $^{\Delta A}/_{A_0}$ und die absolute Phasendrift $\Delta\phi$ des ersten, zweiten und letzten Impulses betrachtet. Der Mittelwert wird über alle Parameterkombinationen aus t_{Puls} und t_{Pause} gebildet und für alle aufgezeichneten Leistungspegel angegeben; als Unsicherheit wird die Standardabweichung vom Mittel berechnet. Die Dynamik auf größeren Zeitskalen wird in Abschnitt 5.5.2 diskutiert und Änderungen zwischen zwei aufeinanderfolgenden Impulsen gegenübergestellt. Erneut werden für jeden AOM die Flächen- und Phasendrift $^{\Delta B}/_{B_0}$ und $\Delta\Phi$ über alle Zeitparameter gemittelt. Die Daten werden in Form von Balkendiagrammen für jeden Impuls dargestellt, um einen einfachen Vergleich zu ermöglichen.

5.5.1 Verhalten innerhalb eines Impulses

Die Dynamik innerhalb eines Impulses wird für die in den Abschnitten 5.2.1 und 5.2.2 diskutierten Daten verglichen. Als Vergleichsgröße dienen die Mittel aller Drifts über alle Kombinationen der Zeitparameter t_{Puls} und t_{Pause} für den ersten, zweiten und letzten Impuls der Sequenz – das Mittel der relativen Flächendrift wird mit $\langle {}^{\Delta A}/_{A_0} \rangle$ und das der Phasendrift mit $\langle \Delta\phi \rangle$ bezeichnet.

In Tab. 5.3 sind die ermittelten Flächenänderungen für alle aufgezeichneten Leistungspegel angegeben und in Abb. 5.19 für 70% und 100% Leistung als Balkendiagramme dargestellt. Gezeigt ist das Mittel und die Standardabweichung des ersten, zweiten und letzten Impulses der Sequenz.

5 Die Bandbreite ist für den Modulator IntraAction ATM-2351-A2 laut Hersteller als 50% von ν_C spezifiziert [36].

		relative Flächenänderung $\langle \Delta A/A_0 \rangle$		
		Ausgangsleistung		
Impuls	Modulator	100%	70%	40%
erster Impuls	Brimrose	$1,7(1) \cdot 10^{-3}$	$1,2(2) \cdot 10^{-3}$	$2,4(6) \cdot 10^{-3}$
	Gooch & Housego	$0,8(2) \cdot 10^{-3}$	$0,7(3) \cdot 10^{-3}$	$2,3(6) \cdot 10^{-3}$
	IntraAction	$0,9(1) \cdot 10^{-3}$	$0,5(1) \cdot 10^{-3}$	$1,5(4) \cdot 10^{-3}$
zweiter Impuls	Brimrose	$1,3(3) \cdot 10^{-3}$	$1,2(2) \cdot 10^{-3}$	$2,7(5) \cdot 10^{-3}$
	Gooch & Housego	$0,8(2) \cdot 10^{-3}$	$0,7(2) \cdot 10^{-3}$	$2,6(6) \cdot 10^{-3}$
	IntraAction	$0,8(1) \cdot 10^{-3}$	$0,6(1) \cdot 10^{-3}$	$2,0(5) \cdot 10^{-3}$
letzter Impuls	Brimrose	$1,4(3) \cdot 10^{-3}$	$1,3(3) \cdot 10^{-3}$	$2,2(6) \cdot 10^{-3}$
	Gooch & Housego	$0,8(2) \cdot 10^{-3}$	$0,9(2) \cdot 10^{-3}$	$2,4(5) \cdot 10^{-3}$
	IntraAction	$0,8(2) \cdot 10^{-3}$	$0,6(2) \cdot 10^{-3}$	$1,9(7) \cdot 10^{-3}$

TABELLE 5.3: Mittelwerte der Flächendrift $\Delta A/A_0$ des ersten, zweiten und letzten Impulses einer Sequenz. Für 100%, 70% und 40% der maximalen Radiofrequenzleistung ist das Mittel für jeden Modulator angeführt.

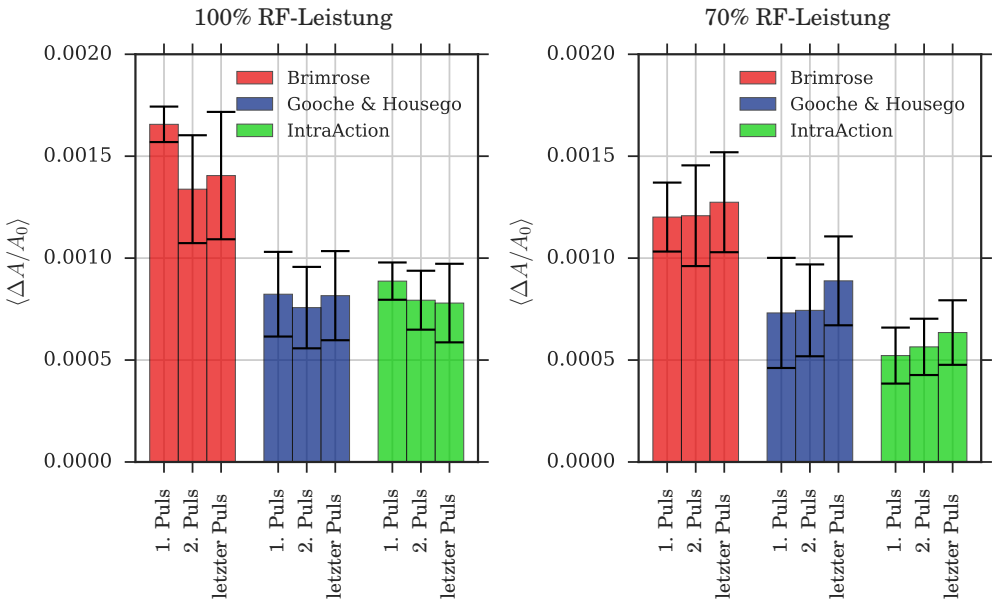


ABBILDUNG 5.19: Grafischer Vergleich der Impulsflächendrift für den ersten, zweiten und letzten Impuls der Impulssequenzen. Dargestellt sind der Mittelwert und die errechnete Standardabweichung der Drift.

Die Impulsflächenentwicklung zeigt bei 100% des Signalpegels für den Modulator Brimrose EF-270 eine um den Faktor 2 größere Drift. Dieser Faktor zeigt sich auch bei 70% der Leistung, kann jedoch bei 40% nicht nachgewiesen werden. Bei geringen Signalleistungen ist die Fläche aus dem gemessenen Signal durch einen kleinen Signal/Rauschabstand nicht verlässlich rekonstruierbar. Bereits die Detektion der Flanken zur Zerlegung der Sequenz in Einzelimpulse gestaltet sich in einigen Fällen als schwierig, was sich auf die Impulslänge auswirkt und damit wiederum auf die Fläche auswirkt – dies erklärt die größeren Resultate für die Drift für kleine Signalleistungen.

Die Dynamik der Impulsphasen liefert für 100% der maximalen Radiofrequenzleistung ein ähnliches Resultat. In Tab. 5.4 sind die ermittelten Werte aufgeführt. Das Modell Brimrose EF-270 zeigt eine um den Faktor 3 größere Drift im Vergleich zu den anderen Modulatoren. Aus Abb. 5.20 ist erkennbar, dass für diese Leistung ein Zusammenhang zwischen der Drift und der Impulsposition besteht. Je später der Impuls in der Sequenz auftritt, desto geringer ist seine $\langle\Delta\phi\rangle$. Eine eindeutige Aussage lässt sich jedoch nicht treffen – hier wäre die Analyse jedes Impulses der Sequenz notwendig. Bei 70% der Ausgangsleistung nimmt die Drift für das Modell Brimrose EF-270 ab und wird mit den anderen Modulatoren vergleichbar. Eine Abhängigkeit von der Impulsposition lässt sich für diesen Leistungspegel nicht feststellen. Unabhängig des Modells beträgt die Drift hier $\approx 3 \cdot 10^{-3}\pi$ pro Impuls.

absolute Phasenänderung $\langle\Delta\phi\rangle$ (π) über einen Impuls

Impuls	Modulator	Ausgangsleistung 100%	Ausgangsleistung 70%
erster Impuls	Brimrose	$1,5(3) \cdot 10^{-2}$	$0,4(1) \cdot 10^{-2}$
	Gooch & Housego	$0,5(1) \cdot 10^{-2}$	$0,4(1) \cdot 10^{-2}$
	IntraAction	$0,6(1) \cdot 10^{-2}$	$0,3(1) \cdot 10^{-2}$
zweiter Impuls	Brimrose	$1,3(3) \cdot 10^{-2}$	$0,34(7) \cdot 10^{-2}$
	Gooch & Housego	$0,3(1) \cdot 10^{-2}$	$0,22(4) \cdot 10^{-2}$
	IntraAction	$0,4(1) \cdot 10^{-2}$	$0,4(1) \cdot 10^{-2}$
letzter Impuls	Brimrose	$1,3(3) \cdot 10^{-2}$	$0,4(1) \cdot 10^{-2}$
	Gooch & Housego	$0,3(1) \cdot 10^{-2}$	$0,21(4) \cdot 10^{-2}$
	IntraAction	$0,4(1) \cdot 10^{-2}$	$0,23(7) \cdot 10^{-2}$

TABELLE 5.4: Mittelwertbildung der Impulsphasendrift über alle Zeitparameter für den ersten, zweiten und letzten Impuls der Sequenz. Eine größere Drift wird für bei 100% der RF-Leistung für das Modell Brimrose EF-270 beobachtet. Bei 70% liefert $\Delta\phi$ für die untersuchten Modulatoren vergleichbare Werte.

Als Ursache der Drift der Flächen- und Phasenentwicklung werden thermische Ursachen vermutet. In Abschnitt 5.2.1 und Abschnitt 5.2.2 wurde für verschiedene Sequenzen gezeigt, dass ΔA und $\Delta\phi$ mit der gewählten Impulsdauer t_{Puls} und der Impulspause t_{Pause} in Beziehung steht. In Abb. 5.20 zeigt sich eine Abnahme der Phasendrift für 70% der Radiofrequenzleistung. Es wird vermutet, dass es sich auch hier um einen thermischen Effekt handelt – ein geringerer Pegel erwärmt den AOM weniger stark. Gleichzeitig widerspricht diese Theorie der in Abschnitt 5.2 beobachteten Abnahme der Drift für kurze Impulspausen. Die genaue Ursache des Phänomens muss in Zukunft genauer untersucht werden.

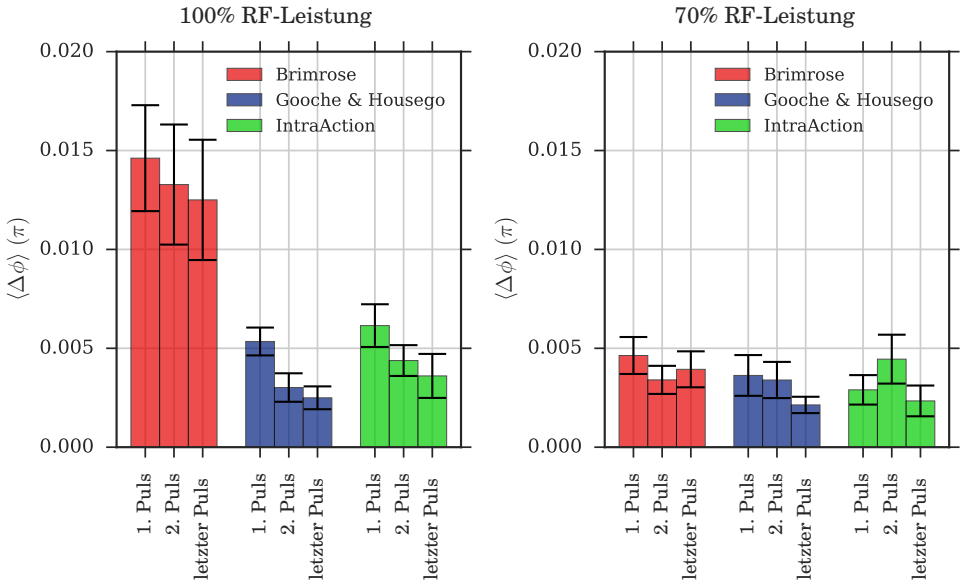


ABBILDUNG 5.20: Grafischer Vergleich der Modulatoren für die mittlere Phasendrift $\langle \Delta\phi \rangle$. Für den ersten, zweiten und letzten Impuls sind der Mittelwert und die errechnete Standardabweichung aufgetragen.

5.5.2 Verhalten in einer Sequenz

Die Dynamik der Impulsfläche B und der Phase Φ zwischen zwei Impulsen wird analog zu Abschnitt 5.5.1 behandelt. Für jedes Modell werden die Mittel der relativen Flächendrift $\langle {}^{\Delta B}/_{B_0} \rangle$ und der Phasendrift $\langle \Delta\Phi \rangle$ über alle Zeitparameter gebildet. Die errechneten Größen sind in Tab. 5.5 für die 70% und 100% der maximalen Ausgangsleistung von M-ActION angegeben.

Die Flächendrift zwischen zwei Impulsen ist in Abb. 5.21 als Balkendiagramm dargestellt. Die Drift stimmt für jedes Modell für beide Leistungen innerhalb der Unsicherheiten überein. Für die Modelle Brimrose EF-270 und Gooch & Housego 3200 zeigt sich eine Abnahme von $\langle {}^{\Delta B}/_{B_0} \rangle$ für 70% des Maximalpegels. Dies deutet wieder auf einen thermischen Effekt hin – die geringere Radiofrequenzleistung führt dazu, dass der TeO_2-Kristall des Modulators besonders für kurze Impulse schneller das thermische Gleichgewicht erreicht. Es wird vermutet, dass der AOM IntraAction ATM2351A2.14 den besten thermischen Kontakt zwischen Elektronik und Kristall oder Kristall und Gehäuse bietet, da das Gleichgewicht schneller erreicht wird (keine/geringe Drift). Unabhängig des Modells lässt sich abschätzen, dass sich zwei aufeinanderfolgende Impulse unter Annahme einer linearen Drift um $\approx 10^{-4}$ in Flächen und Phasen unterscheiden. Für die Phasendrift wird eine größere Drift für 70% der Signalleistung beobachtet. Im Mittel ist $\langle \Delta\Phi \rangle < 1,5 \cdot 10^{-4}\pi$ für alle Modulatoren. Ob die Phase aus den gewonnenen Daten auf diesem Niveau zuverlässig bestimmt werden kann ist jedoch fraglich – eine klare Aussage ist derzeit nicht möglich.

	Modulator	Ausgangsleistung 100%	70%
$\langle {}^{\Delta B}/_{B_0} \rangle$	Brimrose	$1,3(3) \cdot 10^{-4}$	$0,9(2) \cdot 10^{-4}$
	Gooch & Housego	$1,1(3) \cdot 10^{-4}$	$0,8(2) \cdot 10^{-4}$
	IntraAction	$0,6(2) \cdot 10^{-4}$	$0,7(2) \cdot 10^{-4}$
$\langle \Delta\Phi \rangle \ (\pi)$	Brimrose	$0,7(2) \cdot 10^{-4}$	$1,9(9) \cdot 10^{-4}$
	Gooch & Housego	$1,2(2) \cdot 10^{-4}$	$2,0(5) \cdot 10^{-4}$
	IntraAction	$1,0(2) \cdot 10^{-4}$	$1,3(3) \cdot 10^{-4}$

TABELLE 5.5: Gegenüberstellung der Mittelwerte der Flächen- und Phasendrift zwischen zwei aufeinanderfolgenden Impulsen in einer Sequenz. $\langle {}^{\Delta B}/_{B_0} \rangle$ und $\langle \Delta\Phi \rangle$ sind für 70% und 100% des Leistungspegels für jeden Modulator angegeben.

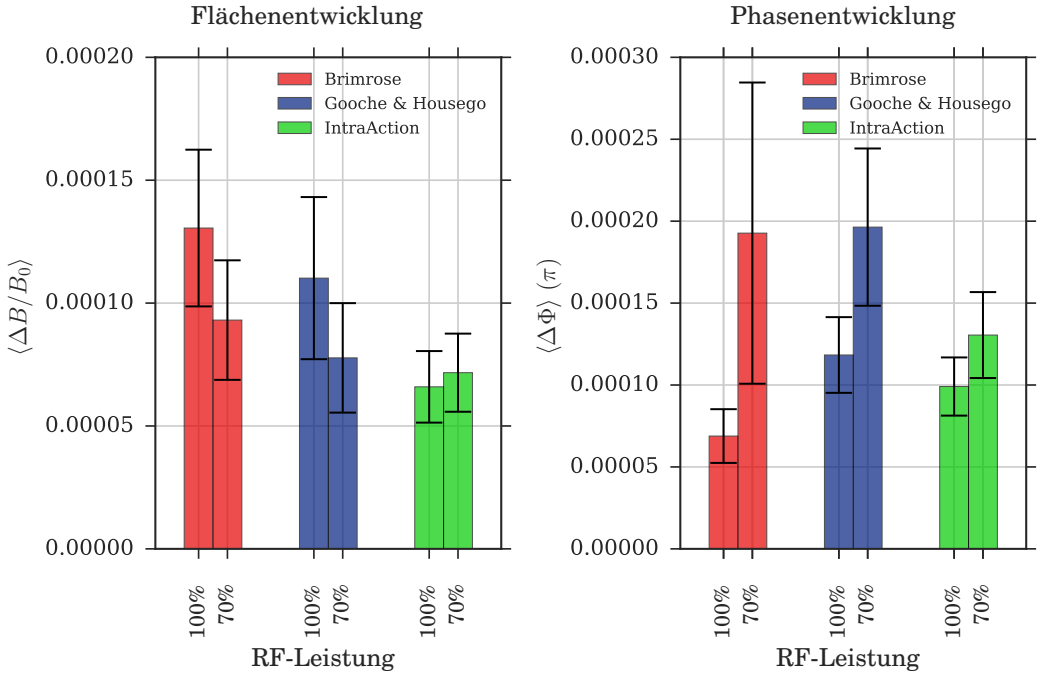


ABBILDUNG 5.21: Vergleich der mittleren Drift in der Sequenz – dargestellt sind die Mittelwerte der Flächen- und Phasendrift $\langle {}^{\Delta B}/_{B_0} \rangle$ und $\Delta\Phi$ zwischen zwei Impulsen; die Unsicherheiten ergeben sich aus den Standardabweichungen.

Kapitel 6

Erwartete Auswirkungen auf Gatteroperationen

In diesem Kapitel werden die Auswirkungen der in Kapitel 5 diskutierten Ergebnisse auf Quantengatter abgeschätzt. In Abschnitt 6.1 werden die Rohdaten als Störgrößen in das Fehlermodell einer Simulation eingearbeitet. Es wird eine Gatterstichprobe simuliert, bei der eine zufällige Sequenz aus Clifford-Gattern Cl, die aus physikalischen Gattern zusammengesetzt sind, erzeugt und auf den Ausgangszustand $|0\rangle = |S\rangle$ eines Qubits angewandt wird. Das letzte Gatter der Sequenz wird so gewählt, dass ein idealer Prozess den Endzustand $|1\rangle = |D\rangle$ ergibt. Die Population in $|D\rangle$ wird bestimmt und über viele zufällige Sequenzen gemittelt. Aus diesem Mittel wird die Fehlerrate der Clifford-Gatter $\mathcal{I}(Cl)$ abgeschätzt, woraus die Fehlerrate der einzelnen physikalischen Gatter $\mathcal{I}(Ph)$ ermittelt wird. Diese Größen werden in 6.2 mit experimentellen Daten verglichen [5].

6.1 Simulation der Gatterstichprobe

Die Auswirkungen des in dieser Arbeit vorgestellten Modells der akusto-optischen Modulatoren werden im Rahmen der Simulation einer Gatterstichprobe betrachtet [16]. Hier handelt es sich um eine Sequenz aus zufälligen Clifford-Gattern Cl, welche sich im Mittel aus 2,25 physikalischen Gattern R zusammensetzen. In Abschnitt 6.1.1 wird die Modellierung der Fluktuationen der Impulsflächen und -phasen in der Simulation beschrieben – das Fehlermodell aus [5] wird um die Daten der AOMs erweitert. Das Resultat der Gatterstichprobe ist in Abschnitt 6.1.2 diskutiert.

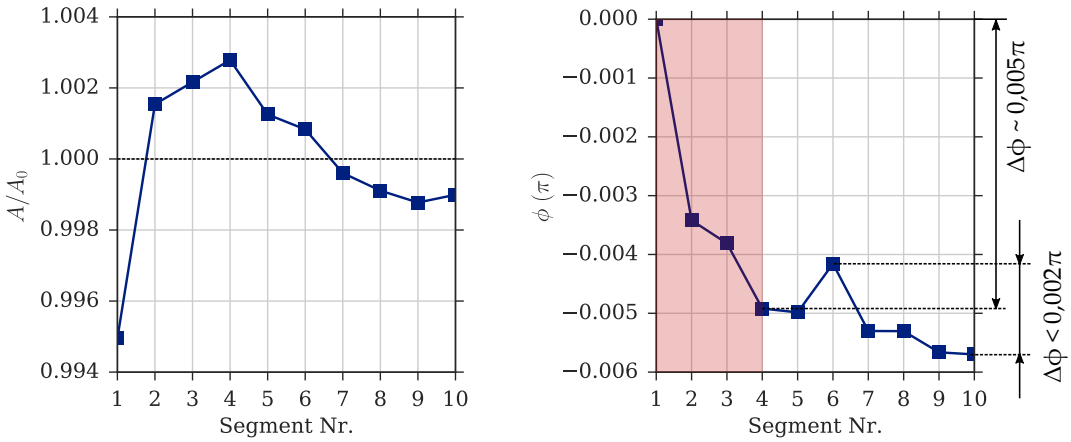
6.1.1 Simulation fehlerbehaftete Quantengatter

Es wird versucht, den Einfluss der Modulatoren auf die Quantengatter R so zu modellieren, dass diese möglichst realitätsnah in der Simulation wiedergegeben werden können. Dafür werden die Rohdaten der Impulsflächen- und Phasenentwicklung direkt verwendet. Mit der gewählten Rabifrequenz von $\Omega = 2\pi \cdot 50\,\mathrm{kHz}$ beträgt die Dauer eines Übergangs zwischen dem Zustand $|S\rangle$ und $|D\rangle$ $t_\pi = 10\,\mu\mathrm{s}$. Der Datensatz mit den Zeitparametern $t_{\mathrm{Puls}} = t_{\mathrm{Pause}} = 10\,\mu\mathrm{s}$ wird gewählt, um einen Impuls dieser Dauer darstellen zu können. Es werden die Daten zu 70% und 100% der maximalen Radiofrequenzleistung betrachtet.

Modellierung der Dynamik innerhalb eines Impulses

Die Entwicklung der Fläche A und Phase ϕ innerhalb eines einzelnen Impulses wird nach Abschnitt 4.3 durch Zerlegung des Impulses in 10 Segmente gleicher Länge und Bestimmung der Anteile $A^{(i)}$ und $\phi^{(i)}$ mit $i \in \{1, \ldots 10\}$ modelliert. Dies ist in

Abb. 6.1 für einen Einzelimpuls dargestellt, wobei die Flächenentwicklung relativ zum Mittelwert A_0 und die Phasenentwicklung $\phi^{(i)}$ als Funktion der Segmentnummer i aufgetragen ist[1]. Die rot hinterlegten Segmente in Abb. 6.1b zeigen eine deutliche Änderung zu Beginn des Impulses, die den Phasenfehler des Gatters dominiert.



(A) Flächenentwicklung (B) Phasenentwicklung

ABBILDUNG 6.1: Darstellung des Flächen- (A) und Phasensignals (B) für den Modulator Gooch & Housego 3200-124 für die maximale Radiofrequenzleistung. Die rot hinterlegten Segmente in (B) dominieren den Phasenfehler des Gatters R.

Im folgenden werden zwei unterschiedliche Fehlermodelle betrachtet:

(1) Zerlegung des Gatters in 10 Segmente und Verarbeitung der Rohdaten – das Gatter wird in eine Sequenz aus Drehungen zerlegt, sodass[2] $R = R_{10} \circ \ldots \circ R_1$. Die Änderung zwischen zwei Datenpunkten des Phasensignals $\Delta\phi^{(i)} = \phi^{(i)} - \phi^{(i-1)}$ wird als Fehler auf die jeweilige R_i mit $i \in \{1, \ldots, 10\}$ aufgeschlagen – die relativen Flächen $A^{(i)}/A_0$ werden direkt berücksichtigt. Mit der Definition von R in Gl. 2.9 und der Diskussion der Gatterfehler in Abschnitt 2.3.3 lässt sich das fehlerhafte Teilgatter R_i durch Gl. 6.1 modellieren[3].

$$R_i(\theta, \phi) = \cos\left(\frac{1}{10} \cdot \frac{\theta}{2}\sqrt{\frac{A^{(i)}}{A_0}}\right) \mathbb{1} + \\ + i \sin\left(\frac{1}{10} \cdot \frac{\theta}{2}\sqrt{\frac{A^{(i)}}{A_0}}\right) \cdot \left(\cos(\phi + \Delta\phi^{(i)})\sigma_x + \sin(\phi + \Delta\phi^{(i)})\sigma_y\right) \tag{6.1}$$

Der Vorfaktor $1/10$ für den Polarwinkel θ ergibt sich durch Teilung von R in 10 Teilgatter R_i; für jede Drehung R_i ist $\theta_i = \theta/10$.

1 Modulator Gooch & Housego 3200-124, RF-Leistung 100%, erster Impuls der Sequenz zu den Zeitparametern $t_{\text{Puls}} = t_{\text{Pause}} = 10\,\mu\text{s}$

2 Hier ist $f \circ g$ die Hintereinanderausführung zweier Funktionen f und g.

3 Die Definition der Impulsfläche A weicht von der Definition von $\mathcal{A}$ in Gl. 2.27 ab – A ist durch $I_0 \cdot t_{\text{Puls}}$ gegeben, während $\mathcal{A} = \sqrt{I_0} t_{\text{Puls}}$ ist. Durch Betrachtung der relativen Größen (bezogen auf das Mittel A_0 bzw. $\mathcal{A}_0$) und Bildung der Wurzel gilt $\sqrt{A/A_0} = \mathcal{A}/\mathcal{A}_0$

(2) Approximation der Fluktuationen durch lineare Drift – die Dynamik der AOMs wird durch die in Abschnitt 5.2.1 und Abschnitt 5.2.2 ermittelten linearen Änderungen ΔA und $\Delta\phi$ modelliert. Die Daten eines in 10 Segmente zerlegten Impulses werden durch

$$\frac{A(i)}{A_0} = \frac{A_y}{A_0} + i \cdot \frac{\Delta A}{A_0} \quad \forall i \in \{1, \ldots, 10\} \quad \text{und} \quad \phi(i) = \begin{cases} \phi_y + i \cdot \Delta\phi & i = 1 \\ \Delta\phi \quad \forall i \in \{2, \ldots, 10\} \end{cases}$$

genähert. Mit diesen Annahmen wird das fehlerbehaftete Gatter R_i durch Gl. 6.2 beschrieben.

$$\begin{aligned} R_i(\theta, \phi) = &\cos\left(\frac{1}{10} \cdot \frac{\theta}{2}\sqrt{\frac{A_y + i\Delta A}{A_0}}\right) \mathbb{1} + \\ &+ i \sin\left(\frac{1}{10} \cdot \frac{\theta}{2}\sqrt{\frac{A_y + i\Delta A}{A_0}}\right) \cdot (\cos(\phi + \phi_y + \Delta\phi)\sigma_x + \sin(\phi + \phi_y + \Delta\phi)\sigma_y) \end{aligned} \tag{6.2}$$

Der Unterschied beider Methoden wird durch die Prozessgüte $\mathcal{F}(\sigma, \varrho)$ gezeigt – diese ist für zwei Prozesse ϱ und σ durch Gl. 6.3 definiert [42]. Dabei sind ϱ und σ die Dichtematrizen des fehlerbehafteten und des idealen Prozesses. Zum Vergleich der Fehlermodelle wird für jede Ausführung eines Gatters R_i ϱ_i und σ_i berechnet und daraus $\mathcal{F}(\sigma_i, \varrho_i)$ bestimmt. Mit der rekursiven Definition $|\psi\rangle_i = R_i |\psi_{i-1}\rangle$ für den Zustand nach der Ausführung des i-ten Gatters sind σ und ϱ wie in Gl. 6.4 gegeben – hier beschreibt R_i^{ideal} ein ideales Gatter.

$$\mathcal{F}(\sigma, \varrho) = \mathrm{Sp}\left(\sqrt{\varrho^{1/2}\sigma\varrho^{1/2}}\right) \tag{6.3}$$

$$\begin{aligned} \sigma_i &= (|\psi_i\rangle \langle\psi_i|)^{\text{ideal}} = R_i^{\text{ideal}} |\psi_{i-1}\rangle \langle\psi_{i-1}| (R_i^{\text{ideal}})^\dagger \\ \varrho_i &= |\psi_i\rangle \langle\psi_i| = R_i |\psi_{i-1}\rangle \langle\psi_{i-1}| R_i^\dagger \end{aligned} \tag{6.4}$$

In Tab. 6.1 ist die Fehlerrate des Prozesses $1 - \mathcal{F}$ für beide Fehlermodelle angegeben. Als Datensatz wird der in Abb. 6.1 dargestellte Impuls gewählt sowie Mittewert und Standardabweichung über alle Impulse der Sequenz zu den Zeitparametern $t_{\text{Puls}} = t_{\text{Pause}} = 10\,\mu\text{s}$ für jeden der untersuchten Modulatoren gebildet[4].

	$1 - \mathcal{F}\ (10^{-6})$	
	Rohdaten	lineare Drift
erster Impuls	0,798	0,803
Brimrose EF-270	11,5 ± 1,6	11,1 ± 1,7
Gooch & Housego 3200	0,38 ± 0,14	0,36 ± 0,15
IntraAction ATM-235	0,51 ± 0,17	0,39 ± 0,16

TABELLE 6.1: Fehlerrate der Impulse – für jeden Modulator wird das Mittel der Fehlerrate für jeden Impuls einer Sequenz berechnet und für alle Modelle angegeben.

4 Die Sequenz wird für 100% des Signalpegels betrachtet.

Wenn auch die errechneten Fehlerraten beider Modelle innerhalb ihrer Unsicherheiten übereinstimmen, zeigt sich, dass das Fehlermodell (2) den Gatterfehler im Mittel unterschätzt. Da die Rohdaten in (1) direkt in das Modell eingehen kann davon ausgegangen werden, dass (1) für lange Gattersequenzen eine genauere Abbildung der physikalischen Realität erlaubt. Es wird daher dieses Fehlermodell für die weitere Analyse gewählt.

Modellierung der Drift in der Sequenz

Für die Simulation der Gatterstichprobe werden Sequenzen von über 600 physikalischen Gatter betrachtet. Die Effekte zwischen zwei aufeinanderfolgenden Quantengattern werden durch die Bestimmung der Drift einer Sequenz modelliert. Mit dem Flächenmittel B_0 wird die relative Flächenänderung $^{\Delta B}/_{B_0}$ zwischen den Impulsen bestimmt. Diese wird durch

$$\theta = \theta_0 \cdot \sqrt{\frac{B_0 + \Delta B}{B_0}}$$

im Polarwinkel θ berücksichtigt; θ_0 beschreibt den ungestörten Winkel. Für das Phasensignal wird analog vorgegangen. Die Drift $\Delta\Phi$ ist hier eine absolute Größe, sodass

$$\Phi = \Phi_0 + \Delta\Phi$$

gilt. Diese Störgrößen werden vor Ausführung des ersten Teilgatters R_i einmalig in den jeweiligen Winkeln als Konstante berücksichtigt.

6.1.2 Gatterstichprobe

Im Folgenden wird eine numerische Simulation präsentiert, die Einflüsse der akustooptischen Modulatoren auf Gatteroperationen untersucht. Effekte der Umgebung [43] sowie Fehler in der Zustandspräparation und -detektion (SPAM) sind im Modell nicht enthalten. Es werden jeweils Gattersequenzen mit $l \in \{10, 70, 130, 190, 250, 310\}$ Clifford-Gattern Cl erzeugt [16]. Diese Längen l sind bewusst gewählt, sodass ein direkter Vergleich mit [5] möglich ist. Das Qubit wird im Zustand $|S\rangle$ präpariert; das letzte Gatter der Sequenz wird berechnet, sodass der Endzustand auf $|D\rangle$ projeziert wird. Nach Ablauf jeder Sequenz wird die Wahrscheinlichkeit $p(|D\rangle)$ den Zustand $|D\rangle$ zu erhalten bestimmt. Zu jeder Länge l werden 50 zufällige Sequenzen generiert, sodass eine statistische Bestimmung von $p(|D\rangle)$ möglich ist. Das Resultat für ein bestimmtes l kann als Histogramm dargestellt werden – Abb. 6.2 zeigt so ein Histogramm für $l = 250$ bei der maximalen Ausgangsleistung von M-ActION. Es werden 20 Klassen (Bins) für jeden Modulator betrachtet. Hier zeigt sich eine um den Faktor ≈ 2 größere Breite $\Delta p = 0,016$ der Verteilung für den Modulator Brimrose EF-270-100; die Modelle der Hersteller Gooch & Housego und IntraAction liefern dagegen $\Delta p = 0,01$.

Die Mittelwerte $\bar{p}(|D\rangle, l)$ werden für jede Sequenzlänge gebildet. Diese erlauben eine Abschätzung des in Abschnitt 2.1.3 definierten Gatterfehlers $\mathcal{I}(Cl)$ der Clifford-Gatter durch das Modell Gl. 2.12. Da jedes Cl im Mittel aus 2,25 physikalischen Gattern Ph gebildet wird, ist der Fehler eines Gatters $\mathcal{I}(Ph)$ durch

$$\mathcal{I}(Ph) = \mathcal{I}(Cl)^{1/2,25} \tag{6.5}$$

bestimmt.

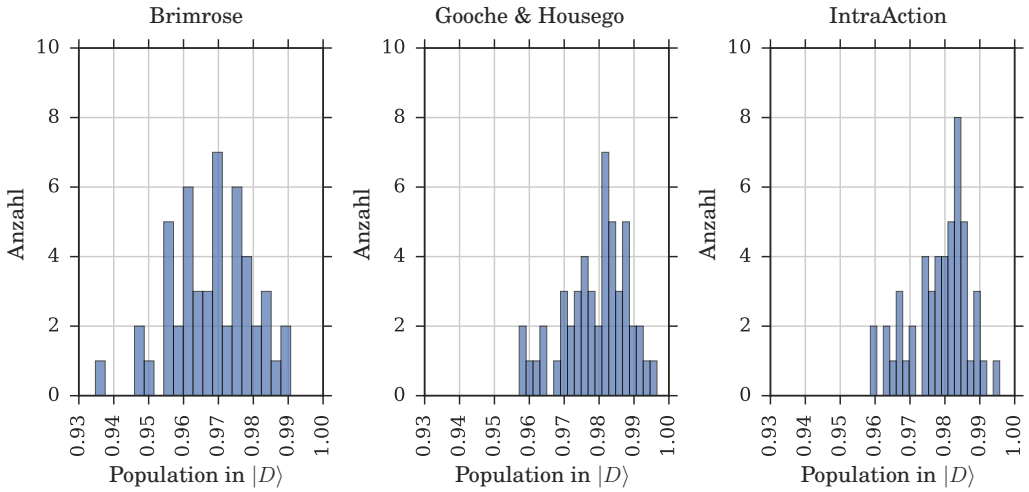


ABBILDUNG 6.2: Histogramm der Population in $|D\rangle$ für eine Sequenz aus 250 Clifford-Gattern bei 100% der Radiofrequenzleistung von M-ActION. Es sind 20 Klassen aufgetragen.

Die Simulation wird nach folgendem Schema ausgeführt:

(1) Es wird eine Länge $l \in \{10, 70, 130, 190, 250, 310\}$ gewählt.

(2) Eine Sequenz aus l zufälligen Cl wird erzeugt.

(3) Die Daten der AOMs werden verarbeitet – da die Sequenzlänge l wesentlich größer als die in dieser Arbeit diskutierten Sequenzen aus 60 Impulsen ist, wird für jedes physikalische Gatter der Cl ein zufälliger Impuls ausgewählt und im Gatter Gl. 6.1 berücksichtigt.

(4) Schritt (3) wird wiederholt, bis alle Cl ausgeführt worden sind; die Population in $|D\rangle$ wird berechnet.

(5) Es werden die Schritte (2)-(4) für eine neue Zufallsequenz gleicher Länge wiederholt – insgesamt wird dieser Prozess für jedes l 50 mal ausgeführt. Anschließend wird die mittlere Population in $|D\rangle$ bestimmt.

(6) Die Sequenzlänge wird angepasst und (1)-(5) erneut ausgeführt, bis alle l abgearbeitet sind. Durch Näherung der mittleren Population $\bar{p}(|D\rangle, l)$ wird der Gatterfehler $I(Cl$ und daraus $I(Ph)$ abgeschätzt.

Die Fehlerraten der Gatter sind in Tab. 6.2 angeführt. Die Daten der AOMs sind berücksichtigt, während Einflüsse des Lasersystem aus [5] zuächst vernachlässigt werden. Der Beitrag der Modulatoren ist mit $I(Ph) = \mathcal{O}(10^{-6})$ geringer als vermutet.

Modulator	Leistung 100%		Leistung 70%	
	$\mathcal{I}(Cl)\ 10^{-5}$	$\mathcal{I}(Ph)\ 10^{-5}$	$\mathcal{I}(Cl)\ 10^{-5}$	$\mathcal{I}(Ph)\ 10^{-5}$
Brimrose	$6,0(1)$	$2,8(1)$	$0,400(6)$	$0,182(2)$
Gooch & Housego	$0,35(1)(1)$	$0,158(2)$	$0,250(3)$	$0,113(2)$
IntraAction	$0,33(1)$	$0,150(2)$	$0,265(4)$	$0,121(2)$

TABELLE 6.2: Vergleich der durch die AOMs hervorgerufenen Gatterfehler. Der Beitrag der Modulatoren zu I(Ph) ist in der Größenordnung $\mathcal{O}(10^{-6})$.

Für den AOM Brimrose EF-270 wird bei 100% des Signalpegels eine höhere Fehlerrate ermittelt – dieser Effekt wird nach Betrachtung der Ergebnisse aus Abschnitt 5.2 erwartet. Für 70% Leistung liefert dieses Modell mit $I(Ph) = 0,182(2) \cdot 10^{-5}$ einen Gatterfehler, der, wenn auch weiterhin um den Faktor 2 größer, mit den übrigen Typen vergleichbar ist. Anhand dieser Daten wird bei der Verwendung der Modulatoren Gooch & Housego 3200 und IntraAction ATM-235 in Experimenten eine höhere Gattergüte erwartet.

Die Fluktuationen des Lasers sollen im nächsten Schritt berücksichtigt werden. Die Simulation wird wiederholt und die zeitliche Entwicklung der Intensität und Phase des Lichts mit

$$\theta \longrightarrow \theta\sqrt{\frac{I}{I_0}} \quad \text{und} \quad \phi \longrightarrow \phi + \Delta_L\phi$$

auf das Gatter R_i aus Gl. 6.1 aufgeschlagen. Zur Modellierung der Fluktuationen werden die Rohdaten aus [5] verwendet. Das Ergebnis ist in Tab. 6.3 angeführt und in Abb. 6.3 dargestellt. Die mittleren Anregungen in $|D\rangle$ für jede Sequenzlänge l sind blau hinterlegt. Diese errechnen sich durch Bildung des Mittelwerts der 50 Zufallssequenzen für jedes l. Die Unsicherheiten ergeben sich aus den Standardabweichungen der Anregungen. Der rot abgebildete Verlauf beschreibt die Näherung der $\bar{p}(|D\rangle, l)$ durch Gl. 2.12.

Modulator	Leistung 100%		Leistung 70%	
	$\mathcal{I}(Cl)\ 10^{-5}$	$\mathcal{I}(Ph)\ 10^{-5}$	$\mathcal{I}(Cl)\ 10^{-5}$	$\mathcal{I}(Ph)\ 10^{-5}$
Brimrose	$13,7(5)$	$6,2(2)$	$8,8(2)$	$4,0(1)$
Gooch & Housego	$9(3)$	$4,1(2)$	$8,2(1)$	$3,7(1)$
IntraAction	$8,5(5)$	$3,85(1)$	$8,2(1)$	$3,7(1)$

TABELLE 6.3: Ermittelte Gatterfehler mit Berücksichtigung des Lasersystems. Angeführt sind die Fehler eines einzelnen *Clifford*- Gatters Cl sowie der physikalischen Gatter Ph.

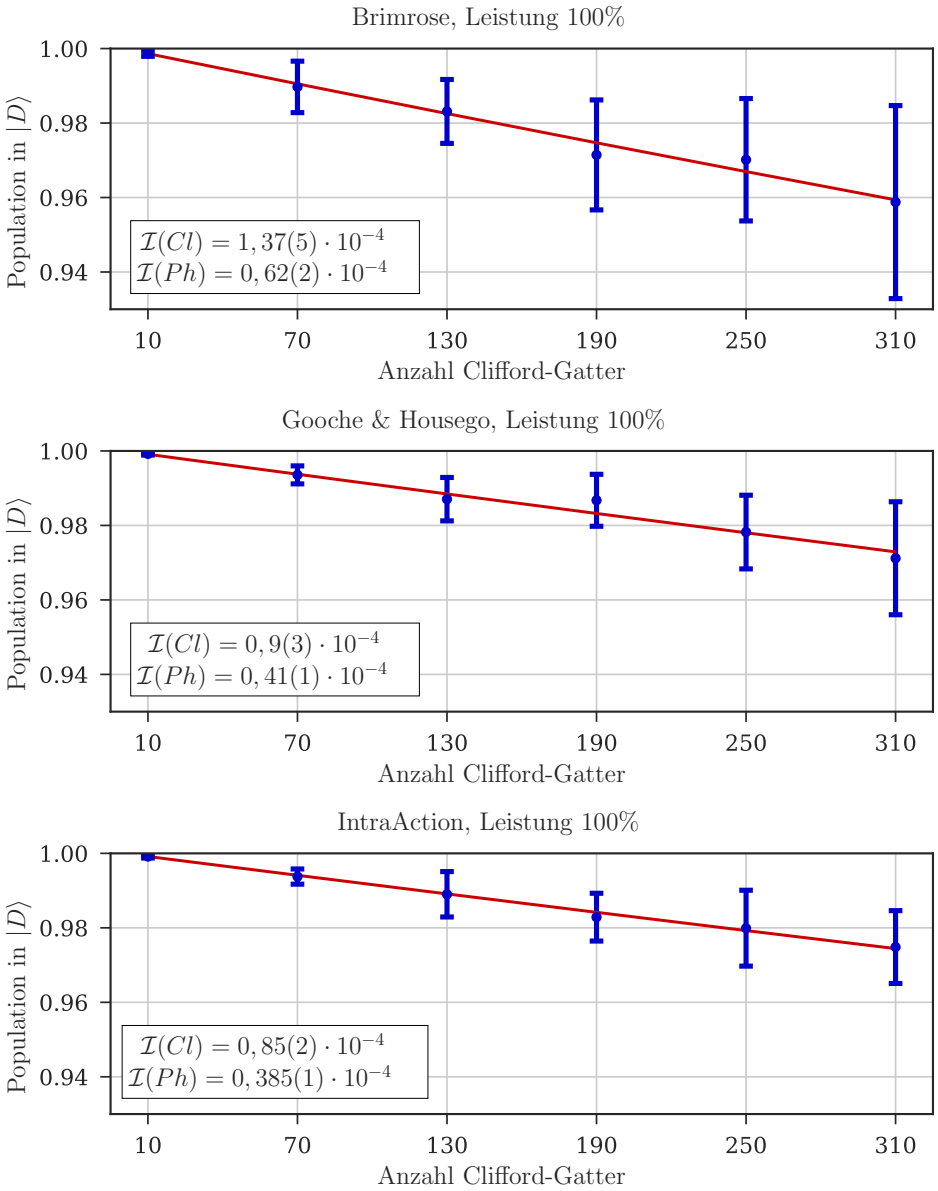


ABBILDUNG 6.3: Darstellung der ermittelten Populationen in $|D\rangle$ für die untersuchten Modulatoren. Die mittlere Anregung wird in blau dargestellt; die Approximation der Daten erfolgt durch das Modell in Gl. 2.12 und liefert den Verlauf in rot.

Zum Test der Funktionalität der Simulation wurde gezeigt, dass die Fehlerrate aus [5] für das reine Lasersystem reproduzierbar ist. Die Berücksichtigung der Intensitäts- und Phasenfluktuationen des Lichts führen zu einem um den Faktor 10 größeren Gatterfehler $\mathcal{I}(Ph) = \mathcal{O}(10^{-5})$. Für das Modell Brimrose EF-270-100 zeigt sich bei maximaler Ausgangsleistung der größte Gatterfehler von $I(Ph) = 6,2(2) \cdot 10^{-5}$. Die Verteilungen sind im Vergleich zu den anderen Modulatoren breiter gestreut. Aus dem Vergleich in Abschnitt 5.5.1 kann geschlossen werden, dass die Modelle IntraAction ATM2351A2.14 und Gooch & Housego 3200-124 vergleichbare Resultate liefern; dies wird durch die Simulation bestätigt. In Tab. 6.3 sind die ermittelten Gatterfehler der Clifford- sowie der physikalischen Gatter für beide gemessenen Leistungspegel angeführt. Nach Abschnitt 5.5.1 wird 70% für der Leistung ein nahezu identisches Verhalten für alle Modulatoren erwartet – die Ergebnisse bestätigen

diese Vermutung. Weiterhin zeigt der Modulator Brimrose EF-270 den größten Gatterfehler.

6.2 Auswirkung der Resultate

Die Simulation zeigt einen Einfluss der Modulatoren auf die Gatteroperationen auf einem Qubit. Ein Vergleich mit Resultaten aus dem Experiment ist nötig, damit die Auswirkungen dieser Effekte auf zukünftige Messungen abgeschätzt werden können. Für alle Modelle werden die Gatterfehler mit den Daten aus [5] verglichen.

Eine in [5] durchgeführte Simulation, welche Effekte des Lasersystems als einzige Störgröße berücksichtigt, liefert $\mathcal{I}(Ph) = 3,82(4) \cdot 10^{-5}$ – davon sind $1,58(2) \cdot 10^{-6}$ Intensitäts- und $3,43(4) \cdot 10^{-5}$ den Phasenfluktuationen zuzuschreiben[5]. Die durch das Intensitätsrauschen hervorgerufene Fehlerrate ist nach Tab. 6.2 mit den Einflüssen der Modulatoren auf die Gatterstichprobe vergleichbar. Das Phasenrauschen des Lasers scheint die erreichbare Gattergüte zu limitieren.

Im Experiment wurden wie in der Simulation Sequenzen aus 10, 70, 130, 190, 250 und 310 Clifford-Gattern erzeugt und für jede Länge 50 Zufallskombinationen der Gatter erstellt und ein Fehler von $\mathcal{I}(Ph) = 6,2(2) \cdot 10^5$ gemessen. In Abb. 6.4 ist das experimentelle Resultat aus [5] zusammen mit den Fehlerraten aus Tab. 6.3 dargestellt. Für den im Experiment eingesetzten Modulator Brimrose EF-270 wird eine perfekte Übereinstimmung erzielt. Die Modelle IntraAction ATM-235 und Gooch & Housego 3200 werden $\mathcal{I}(Ph) = 3,86(1) \cdot 10^{-5}$ und $\mathcal{I}(Ph) = 4,1(2) \cdot 10^{-5}$ erzielt; dies deutet darauf hin, dass ein Einsatz dieser Modelle in zukünftigen Experimenten sich vorteilhaft auf die Güte der Quantengatter auswirkt. Bei 70% der Signalleistung bleibt die Fehlerrate mit $\approx 4 \cdot 10^{-5}$ für alle AOMs unter dem im Experiment bestimmten Wert. Anhand dieses Ergebnisses wird vermutet, dass der Einfluss der Modulatoren auf die Quantengatter für aktuelle Anwendungen nicht als limitierender Faktor zu sehen ist. Eine eindeutige Aussage über die Auswirkungen der in dieser Arbeit diskutierten Störgrößen ist erst möglich, sobald die unterschiedlichen Modelle im Experiment implementiert und im Rahmen einer Tomographie auf einem Ion untersucht worden sind.

5 Es ist anzumerken, dass die Summe beider Komponenten nicht $\mathcal{I}(Ph) = 3,82(4) \cdot 10^{-5}$ liefert – Intensitäts- und Phasenfluktuationen wurden in [5] separat simuliert, bevor beide Anteile in der Simulation berücksichtigt werden; das Zusammenspiel beider Effekte liefert einen größeren Gatterfehler.

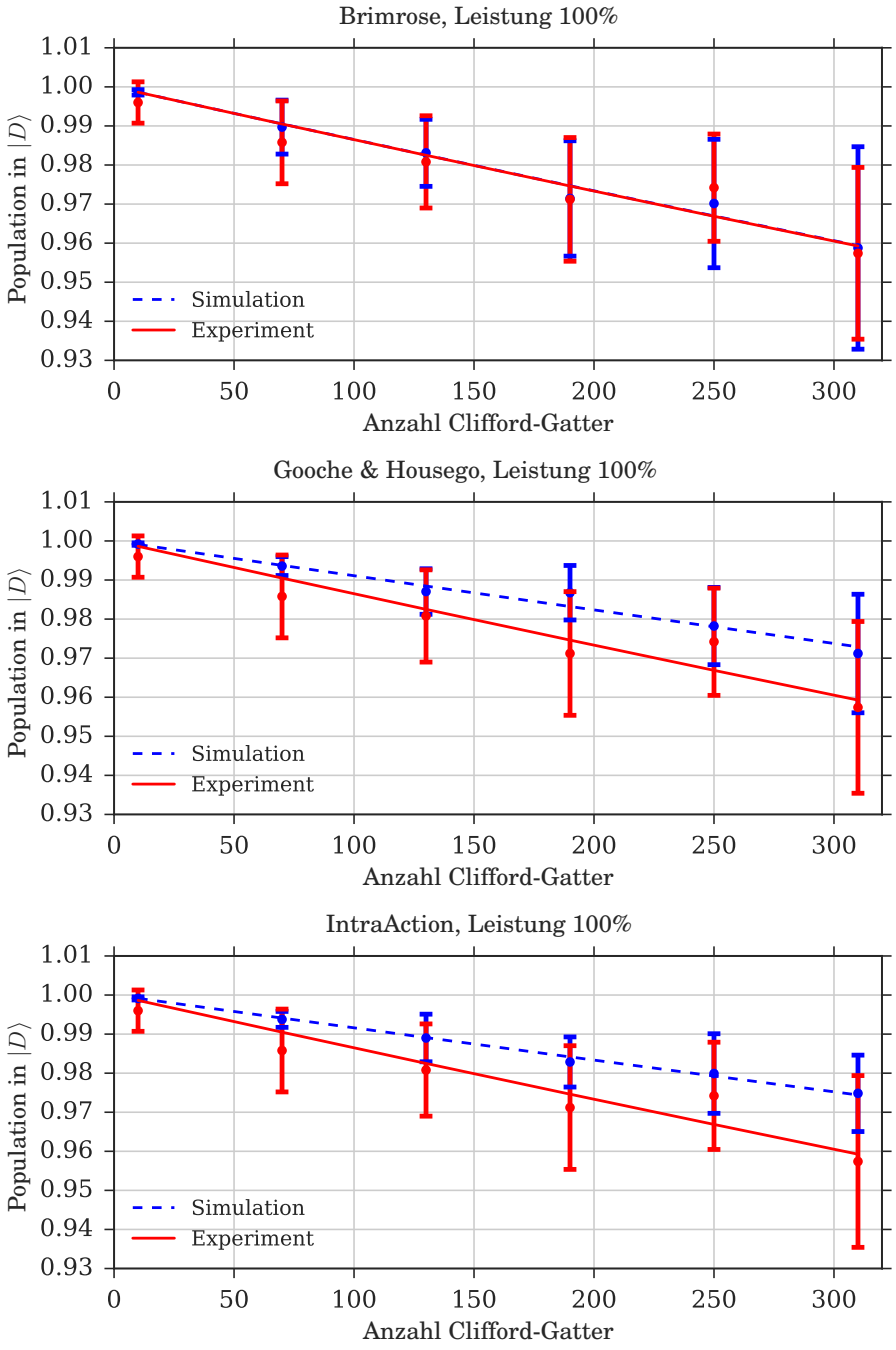


ABBILDUNG 6.4: Darstellung der ermittelten Populationen in $|D\rangle$ für die simulierte Gatterstichprobe im Vergleich zu den experimentellen Resultaten aus [5].

Kapitel 7

Zusammenfassung und Ausblick

In dieser Arbeit wurden experimentelle Imperfektionen auf Quantengatter diskutiert. Mit M-ActION wurde ein neues System zur Erzeugung dieser Impulse getestet. Dieses soll den zur Zeit verwendeten Impulsgenerator *PulseBox* ablösen und neue Experimente ermöglichen. Nachdem ein Verständnis über die Programmierung des Systems erarbeitet wurde, wurden Impulssequenzen erzeugt, welche zur Charakterisierung von akusto-optischen Modulatoren (AOM) eingesetzt werden. Sowohl die Impulsfläche als auch die Phasenlage des den AOM treibenden Radiofrequenzsignals sind für die Ausführung von Quantengattern von Bedeutung – diese Größen wurden im Rahmen dieser Arbeit bestimmt. Es wurden Modulatoren unterschiedlicher Hersteller untersucht und charakterisiert. Aus den gewonnenen Daten wurde versucht, eine Abschätzung über das Verhalten dieser Komponenten im Experiment zu treffen.

In Kapitel 3 wurde das Impulsgeneratorsystem erklärt und die Funktionsweise und Programmierung anhand von Beispielen erklärt. Mit diesem werden Impulssequenzen erzeugt, die zur Charakterisierung dreier Modulatoren unterschiedlicher Hersteller in einer Doppelpass-Konfiguration dienen. Der frequenzverschobene Laserstrahl wurde mit einer Referenz überlagert und die entstehende Schwebung mit einer Photodiode detektiert. Als Messgröße diente die von der Diode ausgegebene Spannung, welche ein Maß für die Intensität und Phase des Signals darstellt. Eine Impulssequenz aus 60 RF-Impulsen wurde an den Modulator angelegt die Schwebung detektiert und die Impulsfläche und -phase wie in Kapitel 4 beschrieben für die einzelnen Impulse bestimmt. Die Entwicklung dieser Größen wurde für die vollständige Sequenz und auch genauer für den ersten, zweiten und letzten Impuls der Sequenz untersucht; das Zerlegen jedes Impulses in 10 Segmente gleicher Länge erlaubte die Bestimmung der Phasen- und Flächenentwicklung innerhalb eines Impulses. Für jedes Modell wurden die Impulsdauer und die Pause zwischen den Impulsen variiert, um Einflüsse dieser Parameter auf Fläche und Phase nachzuweisen.

Die experimentellen Resultate wurden in Kapitel 5 präsentiert. In Abschnitt 5.2.1 wird gezeigt, dass der untersuchte Modulator EF-270-100 des Herstellers Brimrose deutliche Änderungen der Fläche und Phase innerhalb eines Impulses zeigt. Bei 100% Leistung ändert sich die Impulsamplitude deutlich, was sich auf die Impulsfläche niederschlägt. Die mittlere relative Flächenänderung innerhalb eines Impulses von $\Delta A/A_0 = 1,7(1) \cdot 10^{-3}$ wird für den ersten Impuls der Sequenz bestimmt – diese beträgt für den zweiten und letzten Impuls $\Delta A/A_0 = 1,4(3) \cdot 10^{-3}$. Bei den Modellen IntraAction ATM2351A2.14 und Gooch & Housego 3200-124 wird eine geringere Drift von $\Delta A/A_0 = 8 \cdot 10^{-4}$ beobachtet. Es wird vermutet, dass es sich hier um thermische Effekte des Signalumformers (Transducer) handelt, der das RF-Signal in eine Schallwelle konvertiert. Die Radiofrequenz führt zu einer Erwärmung

des Umformers und beeinflusst so dessen Effizienz. Es ist zudem denkbar, dass sich die Temperaturänderung auf das Kontaktmaterial zwischen Transducer und den im Modulator integrierten TeO_2-Kristalls auswirkt und so die Ausbreitung der Schallwelle beeinflusst. Je nach thermischer Kopplung mit dem Gehäuse erhitzt sich der AOM mehr oder weniger schnell. Besonders für kurze Impulse mit geringem Zeitabstand zueinander kann das System nicht ins thermische Gleichgewicht gelangen. Analog verhält es sich für lange Impulse mit langen Pausen – hier kühlt der Kristall wieder ab, sodass der nächste Impuls zu einer neuen Temperaturänderung führt. Die Drift ist für das Modell Brimrose EF-270 stark leistungsabhängig – für 70% der Radiofrequenzleistung nimmt diese bereits deutlich ab. Die Betrachtung der Dynamik der Impulsphase in Abschnitt 5.2.2 liefert ein ähnliches Resultat. Die Modelle der Hersteller Gooch & Housego und IntraAction zeigen bei maximaler Leistung eine um den Faktor 3 geringere Phasenänderung von $\Delta\phi \approx 0,5(1) \cdot 10^{-2}\pi$ entlang eines Impulses; für den Typ Brimrose werden $\approx 1,3(3) \cdot 10^{-2}\pi$ bestimmt. Eine modellunabhängige Abnahme von $\Delta\phi$ mit der Impulsposition wird beobachtet, sodass auch hier auf thermische Einflüsse geschlossen werden kann. Bei geringerer Leistung sind die Modelle erneut vergleichbar. Anhand dieser Daten wird der Einsatz der Modulatoren IntraAction ATM235 und Gooch & Housego 3200 für zukünftige Experimente empfohlen. Besonders für hohe RF-Leistungen und kurze Impulsdauern wird vom Modell Brimrose EF-270 abgeraten. In Abschnitt 5.3 wurde das Verhalten in der Sequenz untersucht. Es zeigt sich, dass alle Typen für Flächen- und Phasendrift vergleichbare Resultate liefern. Für die Flächenentwicklung ist auch hier ein Zusammenhang mit den Zeitparametern erkennbar – für lange Impulse wird eine Abnahme der Drift $^B/_{B_0}$ beobachtet, während für Kurzimpulse keine Beziehung erkennbar ist. Hier wird erneut ein thermischer Effekt als Ursache dieses Verhaltens vermutet. Die Quelle der Drift wird hier im Telluriumdioxid-Kristall des Modulators vermutet, der durch den Schallumformer und die propagierende Schallwelle erwärmt wird. Die Drift $\Delta\Phi$ erscheint dagegen eher zufällig und beträgt unabhängig des Modells $\Delta\Phi = \mathcal{O}(10^{-4})\pi$. Es ist jedoch fraglich, ob die in Abschnitt 4.2 beschriebene Methode zur Bestimmung der Impulsphase derart kleine Änderungen genau wiedergeben kann.

Die Auswirkungen der Flächen- und Phasenentwicklung auf eine Gatterstichprobe wurde anhand einer Simulation abgeschätzt. Es wurde aufbauend auf [5] ein erweitertes Fehlermodell implementiert, das auf den Daten der aufgezeichneten Laserimpulse basiert. Nach Annahme einer Rabifrequenz $\Omega = 2\pi 50\,\mathrm{kHz}$ wurde der durch die Impuls- und Pausendauer $t_{\mathrm{Puls}} = t_{\mathrm{Pause}} = 10\,\mu\mathrm{s}$ identifizierte Datensatz ausgewählt und in das Fehlermodell implementiert. Für jede Gatterstichprobe wurden Sequenzen aus Clifford-Gattern unterschiedlicher Länge mit je 50 Zufallssequenzen erzeugt, wobei ein Clifford-Gatter im Mittel aus 2,25 physikalischen Gattern besteht. Der Vergleich der Modulatoren wurde für beide aufgezeichneten Leistungspegel durchgeführt. Bei 100% des RF-Signalpegels lieferte das Modell Brimrose EF-270-100 eine deutlich größere Fehlerrate der physikalischen Gatter von $\mathcal{I}(Ph) = 0,62(2) \cdot 10^{-4}$, die mit dem experimentellen Ergebnis aus [5] übereinstimmt. Bei 70% Leistung näherten sich alle Modulatoren einander an. Die Modelle Gooch & Housego 3200-124 und IntraAction ATM2351A2.14 lieferten für beide Leistungen nahezu identische Ergebnisse mit einem Gatterfehler von $\mathcal{I}(Ph) = 0,37 \cdot 10^{-4}$. Das Lasersystem wurde als Hauptstörquelle identifiziert; die Beiträge zu $\mathcal{I}(Ph)$ der Modulatoren sind mit $\mathcal{O}(Ph) \approx 10^{-6}$ gering. Es wird vermutet, dass der limitierende Faktor nicht in der Impulserzeugung, sondern im Frequenz- und Phasenrauschen des Lasers zu suchen ist. Eine genaue Aussage lässt sich erst durch Wiederholung des Experiments

am Ion treffen, nachdem M-ActION in den bestehenden Aufbau vollständig integriert worden ist.

7.1 Weiterentwicklung von M-ActION

An der Weiterentwicklung des Kontrollsystems wird laufend gearbeitet. Hauptaugenmerk gilt der Verbesserung des Programmcodes der DDS-basierten Frequenzgeneratoren, welche ab einer Länge von 80 Impulsen ein fehlerhaftes Verhalten zeigen. Mit einem „Trick“ lässt sich dieses Limit auf ≈ 700 Impulse erweitern[1] – in zukünftigen Experimenten werden jedoch mehr als 10000 Impulse benötigt. Der Grund für diese Beschränkung kann sowohl der Hardware als auch dem Programmcode zugeschrieben werden; die genaue Ursache ist bis dato nicht bekannt. Durch die Zusammenarbeit mit dem auf die Entwicklung von FPGA-Systemen spezialisierten Unternehmen *Enclustra*[2] wird gehofft, dass zukünftige Schaltungen direkt von Spezialisten auf alle Funktionalitäten getestet werden können, sodass ein fehlerfreier Einsatz im Labor möglich ist. Bereits zum Zeitpunkt der Fertigstellung dieser Arbeit ist ein neuer Entwurf der DDS-Schaltungen verfügbar. Wann dieser jedoch in Innsbruck eingesetzt werden kann ist noch offen.

Kleinere Fehler im Programmcode können durch Anpassung der Datenverarbeitung am Kontrollcomputer ausgeglichen werden. M-ActIONs Zählerschaltungen liefern einen zusätzlichen Datenpunkt, der den Zählerstand Null ausgibt. Bei einer Mittelwertbildung wird das Ergebnis so verfälscht – dieser Datenpunkt kann durch Anpassung der empfangenen Daten am Computer entfernt werden. Ein weitreichendes Problem ist die Verarbeitung langer Impulssequenzen. Der externe Speicher der DDS-Schaltungen wird aktuell nicht angesprochen, wodurch das theoretische Limit von mehr als 5000 Impulsen nicht erreicht werden kann.

Die Implementierung in bestehenden Experimenten in Innsbruck stellt ein weiteres Hindernis dar. Das bisher verwendete Kontrollprogramm TrICS wurde für den Betrieb mit dem Impulsgenerator PulseBox entwickelt. Neben der Kommunikation zwischen PulseBox und Computer via Ethernet werden Digitalsignale über Hardwareschnittstellen übertragen – diese dienen der Kontrolle des Status des Impulsgenerators und dem Start/Stopp der externen Zählerschaltungen. Eine Ausführung eines Experiments ist ohne diese Signale nicht möglich. Um TrICS mit M-ActION einsetzen zu können, sind diese Kontrollsignale zu deaktivieren und einige Änderungen (z.B. das Auslesen von M-ActIONs Zählerschaltungen) nötig, die einen tiefen Eingriff in den Kern des Programms notwendig machen. Es ist anzuraten, über die Entwicklung einer neuen Software nachzudenken, die auf den Betrieb mit M-ActION zugeschnitten ist und eine einfache Anbindung neuer Geräte erlaubt.

1 M-ActION unterscheidet zwischen einzelnen Impulsen und `looped_settings`, die periodisch aktualisiert werden (siehe Abschnitt B.4.3 für eine detaillierte Beschreibung). Für `looped_settings` tritt ein Fehler nach 80 Parameteränderungen auf. Mit Einzelimpulsen, die mit `make` erzeugt werden (siehe Abschnitt 3.2.2), können bis zu 700 Impulse erreicht werden

2 Webpräsenz unter http://www.enclustra.com

7.2 Programm zur Steuerung von Experimenten

Zum Betrieb des Impulsgenerators PulseBox sind zusätzlich externe Hardwarekomponenten und eine Kette an Programmen notwendig. Das Erfassen von Daten von Zählerschaltungen und Kameras, deren Analyse und das Konfigurieren der Box benötigt mehrere $100\,\mu s$ und stellt ein Laufzeithindernis dar. Die Wiederholrate von zwei aufeinanderfolgenden Experimenten und das schnelle, ereignisbasierte Umschalten zwischen Impulssequenzen ist durch diese Abläufe limitiert. Die Programmkette der PulseBox ist jedoch so tief im aktuellen Kontrollprogramm TrICS eingebunden, dass eine Anbindung von M-ActION nur bedingt möglich ist:

- PulseBox und TrICS sind auf eine externe Hardware zur Datenerfassung[3] angewiesen. Wird diese Komponente von TrICS nicht im System erkannt, so ist ein Start des Programms und somit auch der Betrieb des Impulsgenerators nicht möglich.
- die zu TrICS ergänzenden Komponenten dienen der Weiterleitung von Signalen an die PulseBox, die Erfassung der Daten sowie die Überwachung der Rückmeldungen des Impulsgenerators. Diese können von M-ActION zwar emuliert werden, wobei jedoch auf einige Digitalausgänge verzichtet werden muss. Die Notwendigkeit dieser Signale gestaltet auch die Anbindung weiterer Hardware an TrICS als schwierig.
- eine Umsetzung von bedingten Operationen, welche das Resultat eines Experiments als Entscheidungsgrundlage für unmittelbar folgende Messungen liefern, sind in TrICS nicht implementiert. M-ActION erlaubt eine Vorverarbeitung der Ergebnisse und ermöglicht so einen schnellen Wechsel zwischen Operationen.

Das neue System M-ActION übernimmt Datenerfassung, Analyse und die Entscheidungsfindung für den Wechsel zwischen Experimenten selbst vor – eine zusätzliche Hard- und Softwarekette entfällt. Die in der Auflistung genannten Hindernisse sind tief im Kern von TrICS verankert und können nur sehr schwer abgeändert werden. Es ist daher ratsam, ein neues, auf M-ActION abgestimmtes Programm zu entwickeln. Dieser Nachfolger von TrICS muss wie der Impulsgenerator selbst modular aufgebaut sein, um die Anbindung neuer Hardware oder eigener Codefragmente so einfach wie möglich zu gestalten. Ein Demonstrationsprojekt wurde im Laufe der Anfertigung dieser Arbeit bereits umgesetzt. Aufbauend auf diesem Prototyp lässt sich unter Einbeziehung aller Arbeitsgruppen des Instituts ein Kontrollprogramm entwickeln, welches die Funktionalität von TrICS übersteigt.

3 National Instruments NI-PCI6733, Hochgeschwindigkeits-Analogausgangsmodul: 1 MS/s, 16 bit, 8 Kanäle, 2 Zählerschaltungen.

Anhang A

Hardware and Software Setup Guide

In this chapter, the procedure for setting up a development environment to program and use the M-ActION pulse generator is discussed. Each step will be outlined in detail, with menu interactions in a given program written in italic font and arrows indicating submenus. A quick overview of the software development kit is given and an easy to follow guide on how to set up the hardware is presented.

A.1 Development environment

A.1.1 Application projects

Before setting up the development environment, a set of predefined application projects are required. These projects define M-ActION's core and contain prebuilt experiments. The projects are available on a local university server, and can be downloaded using GIT. GIT will keep track of any changes of the project files, ensures an easy way of updating content and – in case of fatal programming errors – allows a "roll back" to restore previous versions of the project. The following files are required:

- ionpulse_sdk – this folder contains the necessary framework (the core) of M-ActION. Before continuing, check if the folders "ionpulse_sw", "sharedlib" and "testing_sandbox" are present.
- repositories "blacktop", "hiway" and "pulser" – these folders act as libraries for predefined content that is shared throughout ionpulse_sdk.
- the hardware definition file (HDF) "ionpulse_wrapper.hdf" – the hardware specifications (device information, pin assignments, etc.) are governed by this file.

A.1.2 Xilinx Software Development Kit (XSDK) – Setup

The following steps are outlined for, and tested in, XSDK versions 2015.1 and 2015.2. Software bundles can be found on the vendor's website www.xilinx.com in the download area. For each product, a node locked licence is mandatory; the licence (ISE webpack licence) can be generated for free after signing up to the Xilinx webpage.

After downloading the aforementioned files in App. A.1.1, open the Xilinx Software Development Kit (XSDK) – the dialogue window "Workspace Launcher" will show. Select a file path of your choice, e.g *E:\my_workspace* and click *ok*. The location specified is now registered as the workspace XSDK will operate on.

Once XSDK is up and running, skip the "Welcome" dialogue and from the *File* menu select *File*$\longrightarrow$ *Import*. In the window which will appear, select *General*$\Longrightarrow$*Existing projects into workspace* and select the folder *ionpulse_sdk* from A.1.1 as a root directory by clicking *Browse*. Three different projects (testing_sandbox, ionpulse_sw and shared-lib) are now present in the *Projects* field. Make sure that the checkboxes next to all three projects are ticked and click *ok*[1]. The projects will now appear in the left sidebar of XSDK.

Proceed by adding the repositories *blacktop*, *hiway* and *pulser* by navigating to *Xilinx Tools*$\longrightarrow$ *Repositories* and selecting *new* next to *local repositories* in the upcoming window. Add the downloaded folders and select *Rescan Repositories* before clicking *Ok*. XSDK will now check these repositories for files referenced in the ionpulse_sdk projects.

The final steps involve generating a Board Support Package (BSP) and a Linker Script (LS). For the BSP, select *File*$\Longrightarrow$*New*$\Longrightarrow$*Board Support Package* and click *Specify*. Navigate to and select the hardware definition file *ionpulse_wrapper.hdf* from A.1.1 and click *Finish*. A new window (*Xilinx Board Support Package Project*) will appear – hit *Finish* again without performing any changes. In the BSP's settings make sure to check the tickbox next to *lwip14X*[2] to add the TCP-IP support library.

For each of the projects (*ionpulse_sw*, *sharedlib* and *testing_sandbox*) in the project explorer of the XSDK, the Linker Script has to be generated. In order to generate the scripts, right-click on one of the projects and select *Generate Linker Script*. The values for HEAP- and STACK-size need to be adjusted to 16MB by entering 16777216[3]. Create the script by clicking *Generate*. This process must be repeated for all projects.

A.1.3 XSDK – brief overview

The XSDK main window is divided into different panels, with the important panels shown in Fig. A.1 described below.

(1) Project explorer – the contents of the workspace are displayed in this window. Each entry resembles a tree which can be expanded to access the individual items. Active projects are identified by an icon resembling an open folder next to their name, while closed projects are depicted as blue folders. Errors within a project are indicated with a small red "X" – all errors can be traced to their respective faulty element by expanding the file tree.

(2) Code editor – this window shows the code of a selected file. The editor supports syntax highlighting and declaration lookup[4].

(3) Console window – outputs of the XSDK, detected problems while performing compilations and terminal connections[5] are shown in this window.

1 If the checkbox "Copy projects" into workspace is checked, a local copy of ionpulse_sdk is created in the XSDK workspace.
2 LWIP = light weight IP, internet protocol library, current version 1.41, 1.1 $\longrightarrow$ lwip141
3 This value corresponds to 16MB in bytes (binary).
4 When selecting a code segment and right-clicking, the option *Open Declaration* shows the declaration and occurence of the segment within different files.
5 RS232 command line terminal

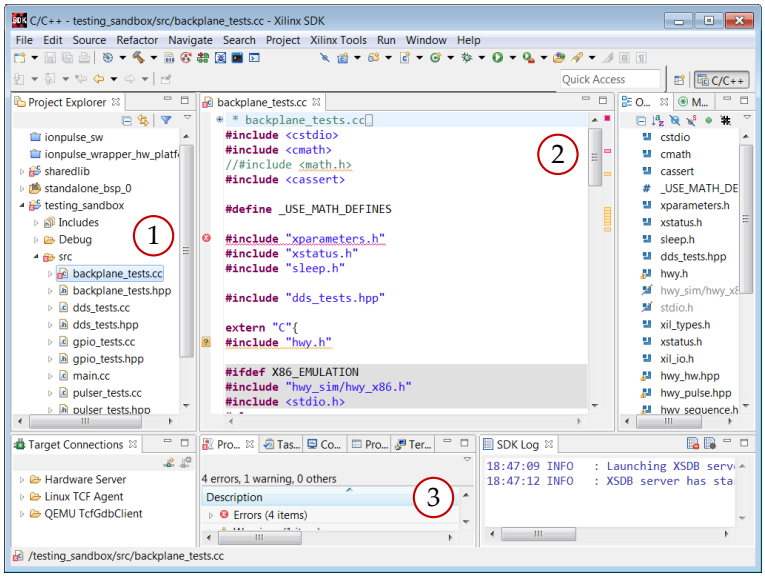


FIGURE A.1: XSDK main window overview – the three main panels are the project explorer (1), the code editor (2) and the console window (3).

Compiling projects

A compilation process is started whenever the "hammer" symbol shown in Fig. A.2 is pressed. The compiler will build all active projects within the workspace – projects which should be excluded from the build should be closed before starting the process. Note that any dependencies between different projects require the constituents to be open during the build.

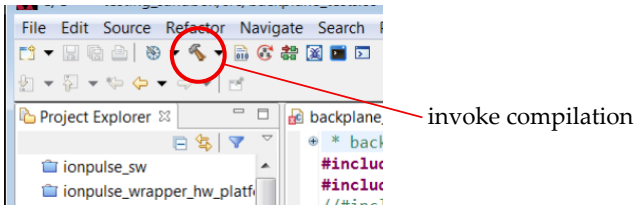


FIGURE A.2: XSDK invoke compilation – a compilation is triggered by pressing the "hammer" icon. All active/open projects within the workspace will be built by the compiler.

If any errors occur during the compilation process, any faulty code segments found by the compiler will be highlighted within the project. Any error can be tracked down by looking either at the error descriptions in the console window (Problems tab) or by expanding the tree view in the project explorer. Warnings thrown by the compiler should be considered, but rarely prevent the code from executing safely.

Hardware target programming

Before any code can run on the target platform, the hardware must be programmed by means of a hardware description file (HDF). This file describes all necessary components within an FPGA/PSoC[6] and how they are connected with the environment. If a hardware platform is connected to the computer, XSDK will recognise the target and the FPGA can be programmed via the menu *Xilinx Tools⟶Programm FPGA*. Should a device not appear as a target even if it is connected and powered, further troubleshooting is needed. It is then recommended to look at the Xilinx Answer Records available at www.xilinx.com.

On-Target execution and system debugger

XSDK supports the launch of compiled projects on connected hardware platforms – the code is executed using the external hardware resources (processor and peripherials) allowing the programmer to step through a program, interrupt it on chosen breakpoints and jump to different events within the program flow. XSDK distinguishes here between the "on-target launch" and the "on-target system debugger":

- Launch on Hardware (GDB, GNU debugger) – the compiled code will be executed without any interrupt capabilities from within XSDK. The code is executed from start to end, but can be terminated at any time.
- Launch on Hardware (System Debugger) – connected hardware will execute the code just like in GDB, but can be suspended and analysed in XSDK. Values of variables within the code are visible for the programmer during the debugging process, making it a powerful tool to thoroughly analyse a project.

Even though GDB is in general a debugging run itself, it is widely used as a quick launch-suspend mechanism without any code analysis aspect in mind. For any thorough inspection of the code it is recommended to use the system debugger.

Note that any connected hardware using GDB or the system debugger will immediately terminate any running code as soon as it is disconnected from the computer.

Run configuration

To execute code on the hardware a "Run configuration" must be created. This configuration ensures correct mapping of the code project to the target device and links present input/output interfaces (IO) with XSDK. Run configurations can be created by right-clicking on a project in the project explorer and selecting *Run As* or using the menu bar shown in Fig. A.3. It features the following options:

(1) Run (green "play" button) – this menu option allows the creation of new run configurations as well as launching previously created ones (in Fig. A.3 testing_sandbox.elf and ionpulse_sw.elf are already present). If an existing configuration is selected, the program code is immediately executed on the connected hardware.

(2) Debug (green "bug" icon) – debug runs are handled analogous to (1), but XSDK's debug perspective is opened.

6 PSoC = Programmable System on Chip, a hybrid between FPGA and microprocessor.

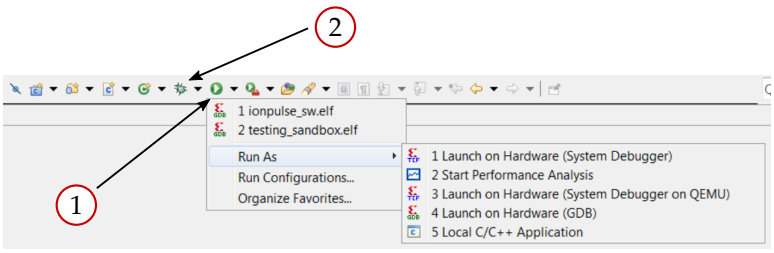


FIGURE A.3: XSDK run configuration – a run configuration can be built using the menu shown above. Two ready-built configurations (testing_sandbox.elf and ionpulse_sw.elf) are already present and can be reused by selecting them directly.

Note that whenever a new launch is created (Launch on Hardware GDB or System Debugger) a new run configuration is created on the fly. Existing configurations should be reused whenever possible.

To interface any peripherial inputs and outputs with XSDK, open *Run configurations* from the menu shown in Fig. A.3. In the now-appearing window's left tree view, the existing configurations are present and can be edited by the user as shown in Fig. A.4. Selecting the tab *STDIO Connection* allows connection of the standard IO (RS232 COM port) to the console window (Fig. A.1 (3)) of XSDK. Data sent via this interface will be displayed in the console, provided the Baudrate is set correctly.

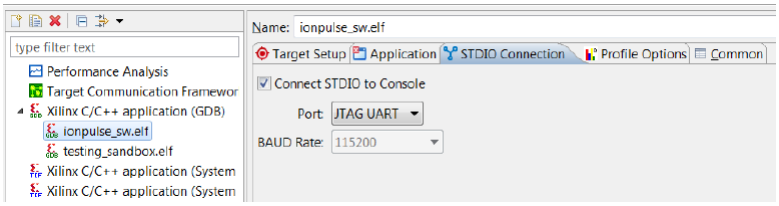


FIGURE A.4: XSDK run configuration editor – properties of existing run configurations can be altered using this editor window.

Note: set the Baudrate to 115200 when establishing a connection with M-ActION. Incorrect settings lead to random characters being displayed in the console window, rendering the output useless.

A.2 Hardware setup guide

A brief discussion on how to properly set up M-ActION's hardware is given in this section. The following components are needed to assemble the pulse generator system:

(1) Xilinx AVNET Zedboard – 1 piece, the mainboard on M-ActION. The Zedboard is available as a bundle, which contains the board, a memory card, a software development kit on a CD with a free registration key, a power supply, two Micro-USB to USB cables and a CAT-5 network cable.

(2) Enterpoint Milldown DDS1 Channelcards – up to 4 pieces (16 channels) are supported

(3) Enterpoint Milldown Backplane 1 – 1 piece, connects the Zedboard with the DDS cards

(4) TIQI backplane breakout board – 1 piece, connection between mainboard and backplane

(5) UIBK isolated digital output cards – 2 pieces, provide digital outputs on 5 Volt (TTL) logic level, connected to the backplane breakout board

(6) ATX Power Supply (more than 650W) – 1 piece

(7) Additional components

- Micro-USB to USB cable – 2 pieces, if not covered by (1)
- CAT-5 network cable – 1 piece, if not covered by (1)
- $12\,\mathrm{V}$ power supply for Zedboard – 1 piece, if not covered by (1)
- DSUB25 cables for digital outputs – 2 pieces
- TIQI to UIBK digital output card adapter – 1 piece
- $12\,\mathrm{V}$ cooling fans

Assemble the hardware step by step according to the following guide:

- Connect the TIQI backplane adapter card (4) with the Zedboard (1) using the FMC LPC connector on the Zedboard shown in Fig. A.5. For mechanical stability, use two screws and two spacers to attach (4) to (1). Use the two Micro-USB to USB cables (7) to connect Zedboard and computer – the Xilinx Software Development Kit (XSDK) is now able to detect the board, once it is powered by a 12V power supply (7). Plug the CAT-5 network cable (7) into the ethernet jack and connect it to the computer or a router.
- Use the 2 DSUB25 cables to connect the TIQI backplane adapter card (4) with the UIBK isolated digital output cards (5) by using the adapter card (7).
- Plug the TIQI backplane breakout board – now attached to the Zedboard – into the backplane (3). Use the connector marked in Fig. A.6. Make sure that the backplane is powered and properly ground by using an ESD clip and wristband.
- Insert the DDS cards (2) into the Micro-TCA connectors highlighted in Fig. A.6. The chosen socket has a direct effect on the channel number of each output – the first slot carries channels 1 to 4, the second 5 to 8 and so on.
- Connect the ATX power supply (6) to the backplane (3) using the ATX connector. Switch on the power supply and the small DIP switch on the backplane[7].
- Finally, connect an external $1\,\mathrm{GHz}$ clock source to the clock input of the DDS channel cards (2). The level of the clock signal should never exceed $-10\,\mathrm{dBm}$.

7 ATX power supplies rely on a "power good"-signal generated by an external power control circuit – this ensures stable power levels when powering a computer. To create a power-good signal, simply bridge pins 14 and 15 on the ATX connector using a wire.

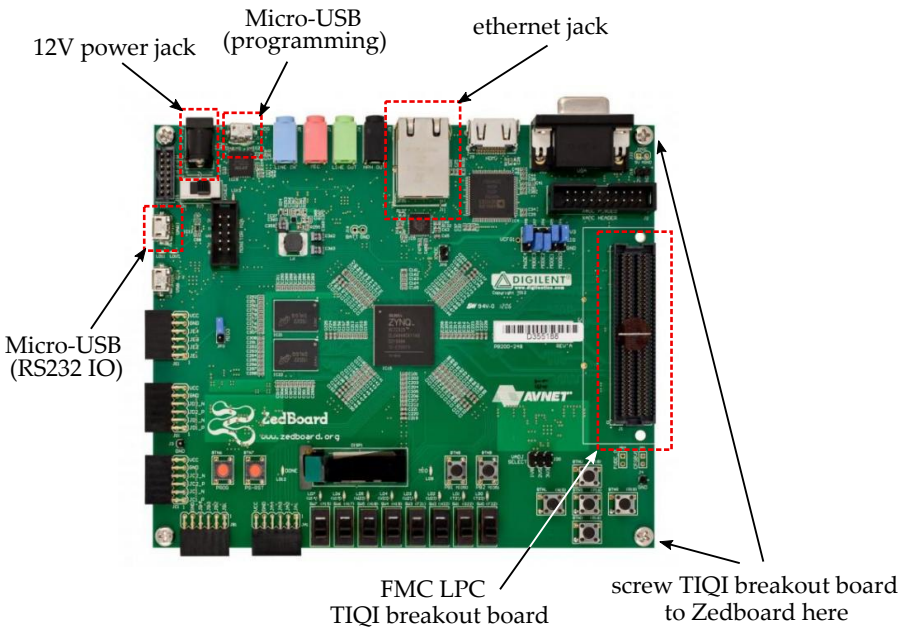


FIGURE A.5: Zedboard – the relevant connectors and hardpoints are shown. The two Micro-USB ports enable programming and console communication, while the ethernet jack provides the connection with M-ActION's control software. The FMC-LPC header serves as an interface to the TIQI backplane breakout board, which should be screwed together with the Zedboard. *Figure taken from www.zedboard.org, edited.*

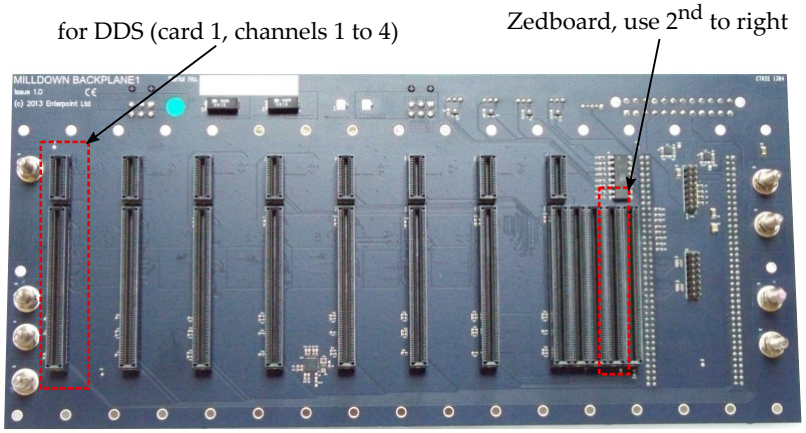


FIGURE A.6: Backplane – use the Micro-TCA headers to connect DDS channel cards and Zedboard. Use only the highlighted connector for the Zedboard (second header to the right). *Figure taken from www.enterpoint.co.uk, edited.*

A.3 Test Application

The application "testing_sandbox" provides basic pulse sequences to test the functionality of the DDS channel cards. If the development system is configured as outlined in App. A.1.2, the code is ready to be executed on the target platform. Provided that the hardware is assembled as descibed in App. A.2, the following steps are necessary to run the application on the Zedboard:

(1) Perform the hardware target programming – invoke the programming toolchain by selecting *Xilinx Tools⟶Program FPGA* from the XSDK menu and select *Program*. If this process is successful, a blue LED will light up on the Zedboard. The device is now ready for the program code.

(2) Compile the code of "testing_sandbox" and check if any errors occur during the compilation. If errors are present, check if the Linker Script (LS) is generated correctly (HEAP and STACK size set to 16 MB), that the software repositories are loaded (*Xilinx Tools⟶Repositories* according to App. A.1.2) and that the Board Support Package Settings are correct (*Xilinx Tools⟶Board Support Package Settings*).

(3) After successfully compiling the code, create a Run configuration with STDIO connected to the XSDK console as described in App. A.1.3. Set the Baudrate to 115200 and launch the application on the Zedboard by *Run As⟶Launch on Hardware (GDB)*.

Once the application is running, ensure that the LEDs (two pairs) are flashing on the DDS channel cards. Each pair should blink with a slightly different frequency. Check the output of the XSDK console window – if everything is working accordingly, a "Hello World" message should is displayed, followed by "1: info, 0: exit".

In the XSDK console window, enter "1" to display the info menu – the different numbers represent the possible tests the Zedboard can run. Perform the following steps:

(1) Connect an oscilloscope to DDS channel number 1 (first card, first channel).

(2) Set the start channel of *testing_sandbox* by typing "10", followed by the desired channel (0 to 15 – choose 0 here) and "0" to enable the backplane trigger[8].

(3) Enter "14" (menu entry *Boss Cap selftest, DDS reset*) – this will prepare a random sequence of pulses on the DDS.

(4) In the XSDK console enter "5" and specify the number of repetitions the pulse sequence should execute – as soon as this command is sent, a pulse sequence should appear on the oscilloscope.

8 A pulse sequence is executed by sending a specific command via the backplane to the DDS cards – this command can either be a command word (*backplane command*) or logic signal (*backplane trigger*). If no pulse sequence is executed using the backplane trigger, try using the backplane command by entering "10" + channel number + "1" and execute the sequence before further troubleshooting.

Anhang B

Working with M-ActION – code description

In this section, the structure of experiments/pulse sequences on M-ActION is discussed. The programming of pulses is shown by means of easy-to-follow code examples. As M-ActION is based on the object-oriented programming language *C++*, all code examples are presented in `C++ code style`.

B.1 A brief introduction to various *C++* concepts

C++ is an object-oriented programming language, which extends the capabilities of standard *C* (*C99* and also *C11*, see for example http://en.cppreference.com/w/c). In addition to known concepts, objects are introduced. An object is invoked from a class, which acts as its blueprint. In turn, a class is a collection of functions and variables, referred to as members, which are available (with some restrictions) to all objects inherited from it. A simple example is presented in the following code snippet:

```
class rectangle {
public:
  int length;
  int width;

  int area(){
     return length*width;
  }
  void set_values(int,int);
};

void rectangle::set_values(int len, int wdth){
  length = len;
  width = wdth;
}

// in main routine, use

rectangle r;
r.set_values(2,3);
int area = r.area(); // integer area is now 6
```

The members of the class `rectangle` are the integers `length` and `width` and the functions `area` (returns an integer) and `set_values`. An object `r` is created from the class, as soon as `rectangle r` is called within the code. From this point on, members of `r` can be accessed by the "dot"-operator, as shown by the call of the member function `r.set_values(2,3)` in the code.

Note that all members of `rectangle` are declared as **`public`**. The keyword **`public`** in conjunction with **`protected`** and **`private`** provides access control[1] to members of a class:

- **`public`** – public members are accessible from anywhere within the code.
- **`protected`** – if a member is protected, only objects who implement the class and inherited classes can access these code segments.
- **`private`** – only members of the class itself can access private code.

The concept of protected members is used whenever a class (the *child*) is inherited from another class (referred to as the *parent*). While protected code is not accessible to anyone not directly related to the parent, the child has access to these members. In the following example, the class `rectangle` is inherited from the parent `form`, which contains just two public and two private members:

```
class form {
public:
  void set_x(int x){
    form_x = x;
  }
  void set_y(int y){
    form_y = y;
  }

protected:
  int form_x;
  int form_y;
};

class rectangle: public form {
public:
  int area(){
          return form_x * form_y;
  }
};
```

1 This concept is known as *information hiding*.

The protected members `form_x` and `form_y` are passed from the parent `form` to the child `rectangle`. While these are known to `rectangle` itself, they cannot be accessed directly from the "outside":

```
// from previous code example, use in code...
/* this code will compile */
rectangle r;
r.set_x(2);
r.set_y(3);
int area = r.area();

/* this code will not */
rectangle s;
s.form_x = 2; // try to access form_x directly -- fails!
s.form_y = 3;
int area = s.area();
```

The faulty code will result in an error, which reads *'int form::form_x' is protected [...] within this context*.

While there are many more interesting concepts regarding classes, objects and inheritance, this section will now close with a short introduction to pointers. For a detailed tutorial on *C++* and also basic *C*, refer to http://en.cppreference.com/w/.

In a computer, variables are stored in certain positions within a memory. The job of a pointer is, as the name implies, to point at a specific memory cell, allowing access to a variable by its memory address instead of calling it by its "name":

```
int a = 5;
int *p = &a;  // assign pointer p the address of a using '&'

int c = p; // c is now the address of a!
c = *p; // c is now '5', the value of a!
```

A pointer is created by adding the "star"-operator `*` as shown above. The memory address of a variable is accessed by the "ampersand"-operator `&`, while the value stored within a certain address is again obtained using `*`.

The use of pointers is heavily encouraged whenever code is written for systems with limited memory – every time a variable is passed to a function within a code, a temporary copy of the variable is registered in the memory. Each function call increases the load on the memory. In addition, as a function operates on a local copy of the variable, the original will not be affected. If a function should alter the specified variable directly, its address should be passed instead of its name.

Note that pointers can also point to an object, like the `rectangle r` defined above. To access its members, instead of using the "dot"-operator, the "arrow"-operator `->` is used.

B.2 Structure of M-ActION

The structure of experiments created for M-ActION will now be discussed – demonstrating how an experiment is created, how to define and use templates and how to add experiments to M-ActION's experiment list.

B.2.1 Structure of an experiment

In M-ActION, each experiment is represented by a class. A common parent class `experiment` is predefined and all newly written code inherits from it – multiple inheritance is common and is encouraged, whenever more complex experiments should be implemented. Whenever an experiment is called, the following process chain is executed:

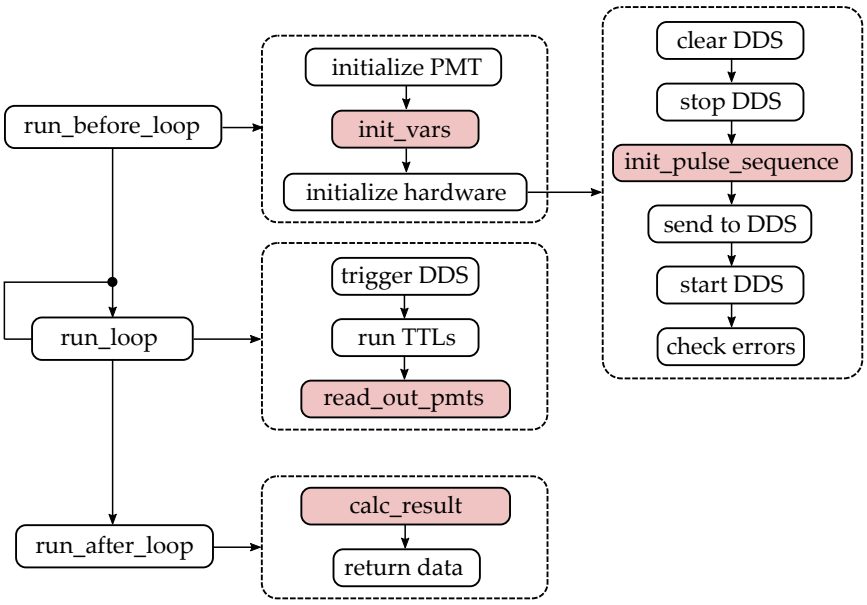


FIGURE B.1: Overview of the process chain, which is executed whenever an experiment is called. The routines highlighted in red must be overridden by the user.

The highlighted routines in Fig. B.1 resemble "empty" member functions, which must be programmed by the user. These functions contain the actual pulse sequence which should be executed. A new experiment `new_experiment`, which inherits from the class `experiment`, declares those members as shown below:

```
class new_experiment: public experiment{
public:
  // constructor:
  new_experiment(list_t* exp_list, const std::string& name);
protected:
  void init_pulse_sequence() override;
private:
  // everything new_experiment should keep private
};
```

Note that the constructor of an experiment always has the arguments `list_t* exp_list` *and* `const std::string& name`*. Those arguments must be present and ensure that the experiment can be registered in M-ActION.*

Note that it is recommended to move the class definition, like the one shown in the example above, into a designated header file. This header file is then included in the code file, where the member functions are implemented.

The class `new_experiment` above shows that the routine `init_pulse_sequence`, which is part of the process chain depicted in Fig. B.1, is implemented. This function contains all the relevant code to create radiofrequency and digital pulses. All highlighted member functions will be explained in App. B.2.3, when the implementation of experiment templates is discussed.

B.2.2 Objects and parameters

M-ActION introduces a variety of objects and parameters which are directly involved in generating radiofrequency pulses, digital pulses and in passing arguments from the control computer to the pulse generator. Objects and parameters can be distinguished between:

(1) remote parameters[2] – these classes contain information which is passed to and received from the experiment control computer. Remote parameters can be easily identified by the prefix `rp_`; for example, an integer can be passed to M-ActION by means of the parameter `rp_int`.

(2) pulse-, FPA- and time-objects – objects are used when a radiofrequency or digital pulse is created or certain, often shared, parameter objects are created from remote parameters.

Remote parameters

A remote parameter is used to distribute information between the control computer and M-ActION – they act as an interface to the "outside world". M-ActION utilises the parameter types shown in Tab. B.1. Other, unused parameter classes like `rp_matrix` or `rp_led` are currently not implemented[3].

remote parameter	data type	description	value (e.g.)
rp_unsigned	unsigned integer	strictly positive integer	1, 2
rp_int	signed integer	pos. and neg. integer values	-1, 2
rp_double	double	floating point number	0.012
rp_bool	bool	boolean, `true` or `false`	true
rp_string	string	ordinary text string	"text"
rp_fpa	–	triple of floating point values	[100,0,80]

TABLE B.1: Overview of the commonly used remote parameters supported by M-ActION.

2 Each remote parameter itself is a class with a variety of members, used to access values or calculate quantities.

3 For a more detailed description of these as well as other parameters, see *ionpulse_sdk/sharedlib/src/shared_params.h* and *ionpulse_sdk/sharedlib/src/shared_params.c* in M-ActION's program code.

Remote parameters are only passed to the control computer if they are defined within the experiment class's constructor, as shown in the example below. The first segment of the code shows the declaration of the class `new_experiment` in the header file `new_experiment.h`, while the code implementation[4] is explained in the second part.

```
// source: HEADER file

class new_experiment: public experiment {
public:
  new_experiment(list_t* exp_list, const std::string& name);
protected:
  void init_pulse_sequence() override;
private:
  rp_int new_integer;
};

// source: CPP file

new_experiment::new_experiment(list_t* exp_list,
                               const std::string& name):
      experiment(exp_list,name),
      new_integer("the remote parameter", &params, 1)
{
}

new_experiment::init_pulse_sequence(){
        // the pulse sequence goes here!
}
```

The remote parameter `new_integer` is added to the class constructor – it has a description[5], is added to the parameter list[6] `&params` and is assigned a default value[7].

In contrast to the example shown above, remote parameters can also be added from a different source than the class's header file. These parameters will then be "pushed" onto the parameter list[8] within the class constructor.

The code example presented above will now be extended – a shared parameter called `external_parameter` is included from the external source file *external_experiment.h* and pushed to the parameter list. Note that its memory address is pushed onto the vector, not the parameter itself.

4 The code is written in a code file, e.g. `new_experiment.cpp`
5 The description is a simple text string, which describes the purpose of the parameter.
6 The parameter list is globally shared among all experiments and contains the parameter of every experiment on M-ActION.
7 In the example shown, the value of the parameter `new_integer` is set to 1.
8 The parameter list itself is a vector (see http://www.cplusplus.com/reference/vector/vector/). Whenever elements are added to a vector, they are "pushed" onto it (appended at the end) by means of `vector.push_back(data)`.

```
// source: HEADER file
#include "external_experiment.h"

class new_experiment: public experiment {
public:
  new_experiment(list_t* exp_list, const std::string& name);
protected:
  void init_pulse_sequence() override;
private:
  rp_int new_integer;
};

// source: CPP file

new_experiment::new_experiment(list_t* exp_list,
                               const std::string& name):
      experiment(exp_list,name),
      new_integer("the remote parameter", &params, 1)
{
  params.push_back(&(external_parameter)); // ext. parameter
}
```

It is recommended to define shared remote parameters in a separate file, which is then saved in a subfolder of *ionpulse_sdk/ionpulse_sw/src*. Global functions accessed by all experiments can also be defined here.

Pulse-, FPA- and Time-objects

In contrast to remote parameters, these objects are not directly accessible from a remote source. Pulse-, FPA- and Time-objects are solely used to describe radiofrequency or digital pulses, which are then executed by M-ActION. Each object is initialised by certain routines which translate remote parameters or a list of variables (different data types, e.g. integer, double, etc.) into an executable pulse. The following objects are commonly used:

object	description	remote parameter
dds::FPA	bundles frequency, amplitude and phase information	rp_fpa
dds::Time	time information	rp_double, rp_int
dds::ttl_dds_pul	pulse object, executable pulse	–

TABLE B.2: Object types used to create radiofrequency pulses. A detailed description on how to initialize these objects is given in App. B.4.

Apart from the objects listed in Tab. B.2 other, rarely used objects are supported[9]. For a detailed description of these objects refer to *ionpulse_sw/src/bp_dds.cpp and [...]/bp_dds.h*. A discussion of the routines involved in initialising pulse-, FPA- and Time-objects is given in App. B.4.

9 An example would be `VgaLutSubsetSetting`.

B.2.3 Working with experiment templates

When using an experimental setup in the laboratory, some routines like the cooling and detection processes will occur on a regular basis – it is thus useful to write a template, tailored to the laboratory setup, in order to minimise the effort when writing new pulse sequences. New experiments can simply be derived from the template, similar to the inheritance of all experiment files from the class `experiment`.

As previously shown in Fig. B.1, the routines `init_vars`, `init_pulse_sequence`, `read_out_pmts` and `calc_result` must be defined by the user – otherwise, no pulse sequence will be executed. The template should fill those routines with code describing cooling, detection, initialization, the PMT readout and the post processing of the data obtained from `read_out_pmts`. The layout of such a template is discussed in the following part.

- Before running the sequence (`init_vars`) – this routine is executed before the actual sequence is processed and sent to the DDS function generators. It can be used to initialise some variables, perform some calculations or print some text to the console output[10] but, in general, it is barely utilized. It is not mandatory to implement this function.
- Processing the pulse sequence (`init_pulse_sequence`) – overriding this member is mandatory. It contains the structure of the template, which can be similar to the process chain shown in Fig. B.2. Like in the parent class `experiment`, overrideable functions can be implemented which must be configured by the user. It is recommended to keep the PMT readout as flexible as possible, as there might be some changes needed depending on the user code. A similar approach for the data analysis routines is suggested[11].

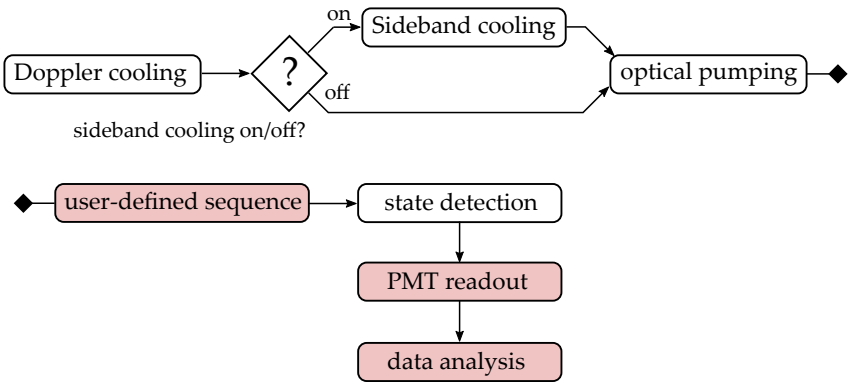


FIGURE B.2: Possible structure of `init_pulse_sequence` within a template. The routine `execute_user_sequence` is kept overrideable for the sequence defined by the user.

10 The console output is an RS232 interface (STDIO, see XSDK description in Anhang A) and can display simple text strings.

11 To add flexibility, the data analysis could be split into two different routines, with one performing tasks needed in every experiment (background counts, etc.) and the second function performing user-specific operations.

- Reading the data (`read_out_pmts`) – performs the data aquisition at the end of each sequence. It is suggested to split this routine into multiple sub-functions, in order to implement general (for every experiment) and specific (depending on the user code) tasks.
- Analyse the data (`calc_results`) – M-ActION can post-process the data itself. The routine `calc_results` should not be implemented by the template, but should rather be "open" for each derived experiment. A default code can be added, which can be overridden at any time.

The following code example shows a possible structure of the template code of the member `init_pulse_sequence`. Doppler, Sideband cooling and optical pumping are implemented, as well as a detection and readout function.

```
new_template::init_pulse_sequence(){
  doppler_cooling();
  if(sideband_cooling == true){
          sideband_cooling();
  }
  pumping();
  execute_user_sequence();
  detection();
  read_out_pmts();
  analysis();
}
```

Using templates, the experiments created on M-ActION can be visualised as depicted in Fig. B.3. All templates inherit directly from the parent class `experiment`, while each laboratory can define its own template and adjust it to its needs, before writing the actual pulse sequences required in the experiment.

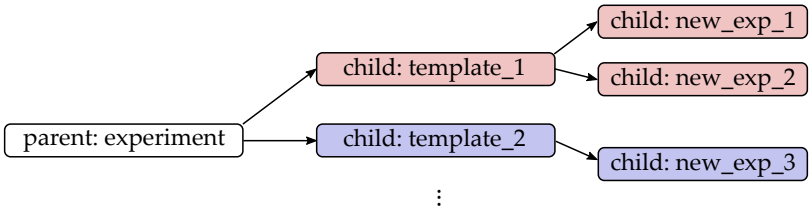


FIGURE B.3: Experiments derived from templates – the highlighted templates are directly derived from the "base class" `experiment`, while the final experiments are children of the individual templates.

B.2.4 Registering experiments on M-ActION

Each newly written experiment class has to be registered in M-ActION to make it accessible to the user via the control computer. An experiment is registered by its class name and has a unique (visible) ID assigned to it – this ID can differ from the name chosen when declaring the class[12]. When running M-AcTION the ID is then

12 For example, an experiment might be declared as `new_experiment`, while its ID can be `"this is a new experiment"`.

shown to the user on the control computer.

M-ActION's experiment list is created within the file *config_local.cpp* which is located in a subfolder of *ionpulse_sdk/ionpulse_sw/src*[13]. An experiment is registered when its header file is included in *config_local.cpp* and the following line is added to the code:

```
void init_experiments(){
  /* some program code
     [...]
    register new experiments here */

  new new_experiment(&global_exp_list, "a new experiment");
  /* the line above registers the experiment!
          [...]
     other programm code */
}
```

Note that each experiment is not only identified by the ID or name assigned to it, but also by a number. This number is governed by the order in which they appear in *config_local.cpp*, and thus their "position" within the experiment list. For example:

```
// config_local.cpp

  new new_experiment(&global_exp_list, "a new experiment");
  // "a new experiment" will appear first --> ID number 0
  new sec_experiment(&global_exp_list, "another experiment");
  // "another experiment" will appear second --> ID no. 1
```

B.3 Network configuration

M-ActION communicates with the experiment computer with a pre-defined IP address – by default M-ActION's address is set to *192.168.2.5*. Different laboratories might have different requirements when it comes to managing their local network. If M-ActION is directly connected to the experiment control computer then, in general, no changes are necessary. Whenever M-ActION is implemented within a local network, the IP address needs to be changed in order to fit the network's address space. This can be done in the file *config_local.h*[14] – the following lines of code must be located within the file:

```
/* file: config_local.h

   includes and other code [...] */

#define LAB_IP1 (192) // change those lines to suit the lab
#define LAB_IP2 (168)
#define LAB_IP3 (2)
#define LAB_IP4 (5)
```

13 The name of the subfolder may change whenever a new version of M-ActION is released, but is usually specific to the laboratory using M-ActION. For example, a current release includes the LinTrap/Ca40 laboratory in the folder *Linear*.

14 As previously mentioned, this file is located in *ionpulse_sdk/ionpulse_sw/src*.

B.4 Programming of pulses

The focus of the previous sections was to discuss the basic structure of M-ActION, including how to create basic experiments and register them in M-ActION's memory. This section considers the programming of pulse sequences, which are a crucial part of the body of the function `init_pulse_sequence`. An overview of the different pulse types is first given, before the execution of loops and looped settings is discussed.

B.4.1 Pulse types and single pulse execution

M-ActION supports a variety of different pulse types, which will be now discussed in detail. The routines which create the Pulse-objects (see App. B.2.2) are presented and the initialisation of FPA- and Time-objects is shown.

Initialising FPA- and Time-objects

Before a radiofrequency pulse is created, its characteristic pulse length, frequency, amplitude and phase must be defined. This is done by assigning a remote parameter or variables by means of integers or floating point numbers directly.

- Initialising an FPA object – an FPA can be created from the remote parameter `rp_FPA` or an ordered triple of values (frequency in MHz, phase in degrees and amplitude in percent). The FPA is declared as a pointer within the header file before it is then initialised in the code by invoking the following method:

```
// header file
class new_experiment: public experiment {
  /* some code */
private:
  dds::FPA *new_FPA = nullptr; // never skip nullptr reference!
// code file
  /* code of init_pulse_sequence [...] */
  bp->use_FPA(&new_FPA, 80,0,100);

  /* more code [...] */
```

Note that upon declaration of the FPA, `new_FPA` is referenced to the memory null pointer **`nullptr`** (or `NULL`). If this reference is not present, M-ActION cannot create an FPA – an error will be thrown[15].

The FPA is created using the routine `bp->use_FPA(...)` with the FPA as the first argument. In the example shown above the FPA's parameters are set by the triple (80, 0, 100). A different approach would be to directly pass a remote parameter `rp_fpa` (in the following called `fpa_parameter`):

```
// code file
  /* code of init_pulse_sequence [...] */
  bp->use_FPA(&new_FPA, fpa_parameter);
```

15 The error message mentions that an attempt has been made to use an already defined FPA (as its pointer is not referenced to null).

An analogous procedure must be applied to Time-objects `dds::Time`. They are also declared as pointers and referenced to **`nullptr`**. Accepted time values assigned to Time-objects are given as floating point or integer values, representing a time in microseconds. The objects are initialised using the routine `bp->use_Time(...)` – the arguments of this function are the object and `rp_double`, `rp_int` or simply integer/floating point numbers. Note that due to M-ActION's program code, only time values $> 1.4\,\mu$s are supported. The declaration and initialisation are shown in the following code:

```
/* header file
   some code [...] */

private:
  rp_double new_time;  // a remote parameter
  dds::Time *time_obj = nullptr;
  dds::Time *time_obj_two = nullptr;

/* more code in header [...] */

/* .CPP file [...] */

  bp->use_Time(&time_obj, 10); // set time 10µs
  bp->use_Time(&time_obj_two, new_time); // from parameter
```

Note that all objects within the code examples are declared as **`private`**. Declaring them as **`public`** or **`protected`** does not affect the initialisation procedure.

Pulse types

M-ActION supports different pulse types – each pulse is defined by a selection of parameters, of which some are not necessary when creating the pulse; these will be marked as optional. Every Pulse-object carries a TTL word, a TTL mask and a PMT flag, which are in general optional parameters. Their effect will be briefly discussed before continuing with the explanation of the pulse types.

- TTL word – the TTL word is a 32 bit integer pattern which is passed to the Pulse-object. Each bit of the TTL word resembles one of the 32 digital output channels of M-ActION. The TTL pattern represents a digital signal which appears on M-ActION's digital outputs when the Pulse-object is executed.
- TTL mask – when passing a TTL word to a pulse the 32 bit wide TTL mask ensures that only those bits which are contained in the mask are altered. Consider the following example[16]:

```
ttl_mask = 0xFFFFFFFF; // all bits accessible
ttl_word = 0x00000001; // only first bit is set

ttl_mask = 0b1010101; // TTLs 0,2,4 and 6 accessible
ttl_word = 0b0000101; // TTLs 0 and 2 are set to 1
```

16 Note that the values are set in hexadecimal as well as in binary format. A number will be treated as binary if the prefix "0b" is detected; hexadecimal numbers are identified by "0x".

- PMT flag – M-ActION triggers its internal counter registers if this flag is not zero. Up to 4 counter inputs are supported – the number of each input governs which counter register is triggered (PMT0 = 0x01, PMT1 = 0x02, PMT2 = 0x04 and PMT3 = 0x08). Passing a PMT flag to a Pulse-object triggers the counters which are synchronous to the executed pulse.

In the following paragraphs the different pulse types are presented. A figure illustrating each pulse shape and its characteristic parameters is presented and a brief description of the pulse, its parameters and its initialisation method is given.

- Wait pulse – a wait pulse is not suited to create a radiofrequency pulse, but rather, to alter the digital output pattern or to gate PMT counter inputs. Its characteristic property is the "on time" t_{on}, i.e. the delay, until an event is triggered.

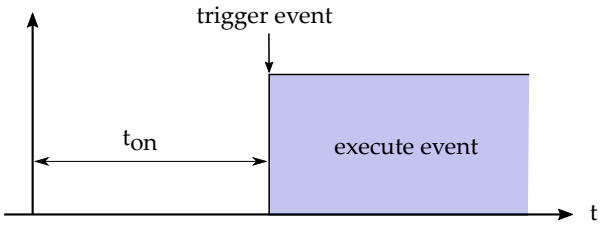


FIGURE B.4: Illustration of a Wait pulse. As no radiofrequency signal is generated, the action triggered by the pulse is depicted by the labelled "execute event"- box.

A Wait pulse is created from a Pulse-object `dds::ttl_dds_pul`, which is then passed to the routine `bp->make_wait`. The following example shows how a Wait pulse is created with the object `t_on` as the time argument. The parameters given in brackets are optional[17].

```
/* header file
   some code [...] */
private:
  dds::ttl_dds_pul *new_pulse = nullptr;

/* .CPP file
   some code [...] */
bp->make_wait(&new_pulse, t, [TTL word], [TTL mask], [PMT]);
```

- Edge pulse – an Edge pulse is a radiofrequency signal which is executed until it is interrupted by another pulse (like a second edge). It is defined by an FPA-object[18] `FPA` governing the frequency, amplitude and phase passed to the pulse, and an optional Time-object `t_on` which serves as a delay.

  An Edge pulse is created using the "Make" routine `bp->make_edge` as shown in the following example. Parameters and objects are defined within the header file, while "Make" is invoked within the code file. The example shows also the initialisation of the FPA- and Time-objects.

17 As a final parameter an additional flag (8 bit integer) can be passed to the pulse. These are in general neglected as they (for now) serve no particular purpose.

18 The characteristic properties of the FPA are the frequency ν, the phase ϕ and the amplitude a.

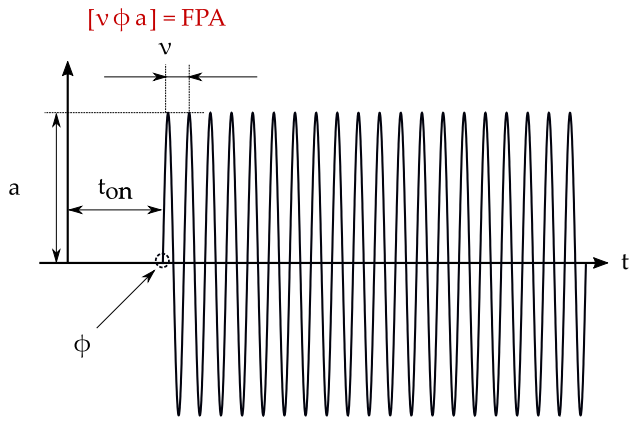


FIGURE B.5: Illustration of an Edge pulse. It is defined by the frequency ν, phase ϕ and amplitude a – these parameters are grouped into an FPA-object. The delay t_{on} is an optional parameter, which can be ignored.

```
/* header file */
private:
  rp_fpa param_FPA; // the parameters passed to the objects
  rp_double param_t_on;

  dds::FPA *FPA = nullptr;
  dds::Time *t_on = nullptr;
  dds::ttl_dds_pul *edge_pulse = nullptr; // Pulse-object

/* .CPP code file
   some code [...] */
  bp->use_FPA(&FPA, param_FPA);
  bp->use_Time(&t_on, param_t_on);

  // create the edge pulse:
  bp->make_edge(&edge_pulse,FPA,[t_on],[TTL word],
                                    [TTL mask],[PMT]);
```

- Cap pulse – Cap pulses are concatenations of edge pulses. They are defined by at least one FPA-object `FPA_on` and one Time-object `t_off` representing frequency ν_{on}, phase ϕ_{on}, amplitude a_{on} and pulse length t_{off}. In this case of only one FPA a second ("hidden") FPA with amplitude $a_{\text{off}} = 0$ is used to switch off the pulse after a time t_{off}.

  A more general pulse is defined by two FPA's – governing the "on"-state and the "off"-state as shown in Fig. B.6. Up to two Time-objects `t_on` and `t_off` define the pulse delay t_{on} and the pulse length t_{off}. If no value for t_{on} is passed it is by default set to the minimum value of $t_{\text{on}} = 1.4\,\mu$s. Note that the "off"-state of a Cap pulse behaves in similar fashion to that of an Edge pulse – it is executed until it is interrupted by another pulse. The following code example shows how to create a general Cap pulse; optional parameters are shown in brackets.

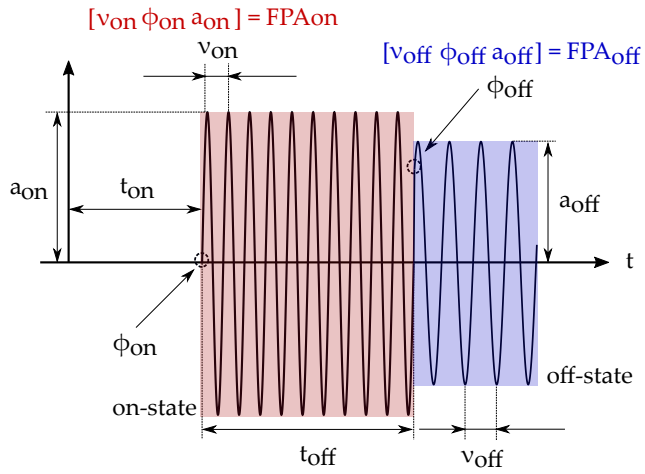


FIGURE B.6: Illustration of a Cap pulse – depicted is the general case of a Cap pulse described by two FPA- and two Time-objects. As a Cap pulse is described by two Edge pulses, the second part of the Cap (the "off"-state) is executed until it is interrupted by another pulse.

```
/* header file */
private:
  rp_fpa param_fpa_on, param_fpa_off;
  rp_double param_t_on, param_t_off;

  dds::FPA *FPA_on = nullptr,
           *FPA_off = nullptr;
  dds::Time *t_on = nullptr,
            *t_off = nullptr;

  dds::ttl_dds_pul *cap_pulse = nullptr;

/* .CPP code file
   [...] /*

  bp->use_FPA(&FPA_on,param_fpa_on); // initialise FPA's
  bp->use_FPA(&FPA_off,param_fpa_off);

  bp->use_Time(&t_on,param_t_on); // initialise Times
  bp->use_Time(&t_off,param_t_off);

  // now make the actual Cap pulse...
  bp->make_cap(&cap_pulse,FPA_on,[FPA_off],[t_on],
                t_off,[TTL word],[TTL mask],[PMT]);
```

- Shaped pulse – the final, commonly used pulse is the Shaped pulse. The amplitude of this pulse follows a certain pre-programmed profile stored in a Look-Up-Table (LUT). A general statement about the profile is not possible, as it might be altered by the user. However, by default, a Blackman lineshape[19] will be created. In addition to FPA- and Time-objects a new object type `VgaSetting` will now be introduced:

  The object `VgaSetting` defines the steepness of the amplitude profile when ramping up the amplitude to the value set by an FPA-object. Minimum and maximum steepness are given by the integer values `0x00000` (minimum) and `0x3FFFF` (maximum). A `VgaSetting` is initialised analogous to FPA- and Time-objects as shown in the following example:

```
/* header file */
private:
  dds::VgaSetting *vga = nullptr;
/* .CPP code file */
  bp->use_vga(&vga,0x01AB1); // ramp steepness 0x01AB1
```

  As `VgaSetting` accepts integer values, remote parameters suited to pass data of a `VgaSetting` from the control computer to M-ActION are `rp_int` and `rp_unsigned`. Note that the object works directly with integer values – it cannot cast `rp_int` to `int`. In order to extract the value of a remote parameter, the member function `getValue()` is called[20].

  In order to simplify the use of Shaped pulses two methods, each with a predefined ramp steepness, are defined. Their "Make"-routines are (with flat ramp steepness) `bp->make_slow_shaped`, and `bp->make_med_shaped` (medium ramp steepness) define the rise time t_r[21]. The pulse is defined by two FPA-objects `fpa_on` and `fpa_off` and a Time-object `t_off` governing the plateau time as shown in Fig. B.7. Note that for this example `fpa_off` carries the amplitude $a_{off} = 0$, thus switching off the pulse. The following example shows how to initialise a slow Shaped and a medium Shaped pulse:

```
/* header file */
private:
  rp_double param_t_off;
  rp_fpa param_fpa;

  dds::Time *t_off = nullptr;
  dds::FPA *fpa_on = nullptr,
           *fpa_off = nullptr;

  dds::ttl_dds_pul *slow_shaped = nullptr;
  dds::ttl_dds_pul *medium_shaped = nullptr;
```

19 The blackman window is defined numerically as given by [38].

20 To get the numerical value of an `rp_double param` within the code, the following line is invoked:
```
double the_value = param.getValue();
```

21 The rise time is defined as the time it takes until the amplitude has reached 90% of its final value.

```
/* .CPP code file
   some code [...] /*

  bp->use_Time(&t_off,param_t_off); // pulse length

  bp->use_FPA(&fpa_on,param_fpa);
  bp->use_FPA(&fpa_off,0,0,0); // switch off (off-state)

  bp->make_med_shaped(&medium_shaped,fpa_on,
                  fpa_off,t_off,[TTL word],[TTL mask],[PMT]);
  bp->make_slow_shaped(&medium_shaped,fpa_on,
                  fpa_off,t_off,[TTL word],[TTL mask],[PMT]);
```

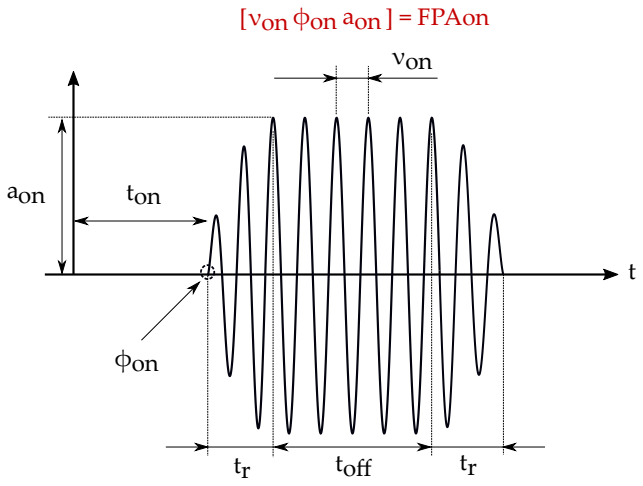


FIGURE B.7: Illustration of a Shaped pulse created with `make_med_shaped` and/or `make_slow_shaped`. Both routines lead to Shaped pulses with a different rise time/amplitude ramp for each pulse.

More generally, a Shaped pulse incorporating a user-defined ramp steepness is created in a similar way. In addition to two FPA's and the pulse length the `dds::VgaSetting` and a collection of Time-objects describing the behaviour of the VGA data transfer have to be considered. Initialising such a pulse is best described by means of investigating the original code found in the files *ionpulse_sdk/ionpulse_sw/src/bp_dds.cpp* and *[...]/bp_dds.h*.

Executing a pulse with `bp->run`

Pulse-objects which have already been created can be re-used multiple times within an experiment file[22]. A pulse is executed when the code `bp->run(...)` is invoked. The arguments of this method are the Pulse-object and the DDS channel(s) the radiofrequency signal is executed on – M-ActION supports 16 channels which are addressed by a 16-bit integer, each bit resembling a single channel. Note that a pulse can also be executed on mulitple channels simultaneously. After a pulse is assigned

22 The Pulse-object can be re-used, but not re-programmed, by means of `bp->run(...)`.

to a DDS channel, the command `bp->sync()` must be called. This command ensures that the pulse is actually executed – if this command is missing, M-ActION will interrupt the code and will be stuck in a loop; M-ActION then must be rebooted.

The next example shows the initialisation of a Cap pulse, which is first simultaneously executed on the channels 1 and 2 for a time of $10\,\mu$s, before a second Cap appears with minimum delay (equal to $1.4\,\mu$s) on channel 2 only. Note that the FPA of the "off"-state is automatically set with amplitude zero (default value).

```
/* header file
   [...] */
private:
  rp_fpa param_fpa_on;

  dds::FPA *fpa_on = nullptr;
  dds::Time *t_on = nullptr;

  dds::ttl_dds_pul *cap_pulse = nullptr;

/* .CPP code file
   [...] */
  bp->use_FPA(&fpa_on,param_fpa_on); // the FPA
  bp->use_Time(&t_on,10); // Time obj. w. 10µs

  bp->make_cap(&cap_pulse,fpa_on,t_on); // no TTL or PMT

  bp->run(*cap_pulse,0x0003); // 0x0001 and 0x0002
  // command above executes pulse on channels 1 and 2

  bp->sync(); // synchronize, IMPORTANT!

  bp->run(*cap_pulse,0x0002); // on channel 2 only
  bp->sync();
```

Local (throw-away) pulses

The previous example showed how a Pulse-object is used multiple times within an experiment file – the same Cap pulse is executed twice. If a pulse occurs only once within an experiment file, and won't be utilised elsewhere within the code, a local (throw-away) pulse can be created. These pulse types create the necessary Pulse-, Time and FPA-objects within their "Make"-routine – this improves the readability of the code in the case of long pulse sequences. Note that whenever a local pulse is created, it is executed immediately (according to the respective parameters passed to the function) on the given DDS channel. The `bp->sync()` command must be invoked by the user after calling such local functions.

For each pulse type discussed in this section, a routine to create a local pulse exists. As an example a local Cap pulse with frequency $100\,$MHz, phase 0, amplitude 80 and pulse length $25\,\mu$s is created and executed on channel 1 by the code shown in the following example.

```
/* .CPP code file */
  bp->make_local_cap(100,0,80,25,0x0001,
                [TTL word],[TTL mask],[PMT]);
  bp->sync();
```

A full list of all local pulse methods is contained in the file *ionpulse_sdk/ionpulse_sw/src/bp_dds.h* – all relevant parameters can be easily identified within the declarations.

Parameter to pulse conversion

In contrast to local pulses, it can be useful to create a pulse from an FPA without the previous initialisation of FPA- and Time-objects. This approach enhances the readability of the code significantly and allows the user to re-use the Pulse-object. Routines performing the conversion of a set of objects into an executable pulse are identified by `bp->fpa_t_[PULSE TYPE](...)` with the pulse type (Edge or Cap in lowercase letters) within the method's name. When such a function is called, the pulse is executed immediately without "throwing away" the Pulse-object – after each call of such a routine, the command `bp->sync()` must be invoked.

The example below shows how a Cap pulse is created from previously uninitialised objects – the pulse is then executed on channel 1.

```
/* header file */
private:
  rp_fpa param_fpa_on;
  rp_double param_t;

  dds::FPA *fpa_on = nullptr;
  dds::Time *t_on = nullptr;

  dds:ttl_dds_pul *cap_pulse = nullptr;

/* .CPP code file */
  bp->fpa_t_cap(&fpa_on, param_fpa_on,&t_on,param_t_on,
            &cap_pulse, 0x0001, [TTL word], [TTL mask], [PMT]);
  bp->sync();
```

Note that `bp->sync()` is called immediately after the method `bp->fpa_t_cap(...)` is executed. For a detailed list of all supported conversion routines refer again to *ionpulse_sdk/ionpulse_sw/src/bp_dds.h*.

B.4.2 Writing a sequence

Radiofrequency pulses are usually part of a more or less complex sequence comprised of many single pulses. Sequences can be naively written by generating each pulse with its characteristic properties individually – note that for each Pulse-object one or more FPA- and Time-objects must exist, and these objects take up valuable memory space within M-ActION[23]. However, it has been shown, that Pulse-objects can be re-used multiple times within a pulse sequence, as long as their properties remain unchanged (same amplitude/frequency/phase and length).

23 More precisely: each object passed to a DDS will be registered within the fairly small memory of the DDS – the DDS is the current bottleneck regarding the number of pulses that can be executed.

The following code serves as an example of a sequence of 200 identical Cap pulses which are executed on channels 1 to 3 depending on an integer number.

```
/* header file */
private:
  dds::Time *t_on = nullptr;
  dds::FPA *fpa_on = nullptr;

  dds::ttl_dds_pul *cap_pulse = nullptr;

/* .CPP code file */
  bp->use_FPA(&fpa_on,80,20,100);
  bp->use_Time(&t_on,10);

  bp->make_cap(&cap_pulse,fpa_on,t_on,
                      0x0000FFFF,0x0000FFFF,1);
  /* cap pulse, freq. 80MHz, phase 20 deg, amplitude 100%
     TTL word --> lowest 16 bits to 1
     TTL mask --> act on lowest 16 bits
     gate PMT channel 1 (open)  */

  /* run the sequence */
  for(unsigned i = 0; i < 200; i ++){
    bp->run(*cap_pulse,i%7); // channel is i modulus 7
    bp->sync();
  }
```

By re-using the Pulse-objects, less data is stored M-ActION's memory – this is advantageous for long sequences. The main downside of this practice is that properties of the Pulse-object cannot be altered by means of changing the FPA's; doing so will result in a fatal error and M-ActION must then be rebooted.

B.4.3 Updating in sequence – looped settings

In order to enable the programming of complex sequences a different approach to App. B.4.2 must be made. FPA- and Time-objects can be updated by means of a parameter vector, whose values define the property of the object. A vector of the type `dds::looped_setting` is defined in the code which allows the parameters of FPA- and Time-objects to be updated. The following code shows the creation of the vector `sequence_vect`, which is filled with amplitude values of an FPA `fpa_on` (from 0 to 99). The amplitude values are first loaded into another vector called `param_vector` before mapping them to `sequence_vect` and thus to `fpa_on`.

```
/* .CPP code file
   assume FPA fpa_on is already initialised */

  new param_vector = new std::vector<double>();
  auto sequence_vec = new std::vector<dds::looped_setting *>;
```

```
/* .CPP code file
   fill param_vector with values from 0 to 99) */

  for(unsigned i = 0; i < 100; i++){
    param_vector->push_back(i);
  }
  // fill the sequence_vect

  sequence_vect->push_back(
          new dds::looped_setting(*(fpa_on->a()),
                                          param_vector));
  /* execute sequence
     the FPA fpa_on belongs to an initialized
     Cap Pulse-object called cap_pulse */

  bp->run_loop(sequence_vect, [&] {
    bp->run(*cap_pulse, 0x0001);
    bp->sync();
  });
```

This example shows the values being pushed onto the `sequence_vect` – more precisely, the mapping between entries of `param_vector` and the amplitude value of `fpa_on` is established. Access to the amplitude values of `fpa_on` is gained by the member function `fpa_on->a()` – an understanding of the member functions of FPA- and Time-objects can be gained from the files *bp_dds.h* and *bp_dds.cpp* in *ionpulse_sdk/ionpulse_sw/src*.

After the `dds::looped_setting sequence_vect` is filled with the amplitudes of `fpa_on` the sequence can be executed. This is achieved by calling the method `bp->run_loop(...)` with `sequence_vect` as the first argument. The second argument is a function (here a local Lambda function[24] as indicated by `[&]`). The code following the ampersand is executed for every entry in `sequence_vect` with the Pulse-object now configured to accept amplitude values from `sequence_vect`.

B.5 Final note

M-ActION is constantly under development; software and hardware are updated almost on a monthly basis. Thus, this guide on pulse programming cannot considered to be complete – it will be updated continuously as the M-ActION project progresses over the next years. Thus, this guide on pulse programming cannot be considered to be complete – it will be updated continuously as the M-ActION project progresses over the next years. The guide is also restricted to commonly used objects, parameters and functions which make up a part of the whole, more complex system. However, it offers an intuitive introduction to the software development kit, as well as a set of easy to follow instructions on how to set up the hardware.

24 Refer to http://de.cppreference.com/w/cpp/language/lambda for a description on Lambda functions.

Literatur

1. Feynman, R. & Shor, P. W. Simulating Physics with Computers. *SIAM Journal on Computing* **26,** 1484–1509 (1982).
2. Shor, P. W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing* **26,** 1484–1509 (1997).
3. Grover, L. K. *A fast quantum mechanical algorithm for database search* 1996. eprint: arXiv:quant-ph/9605043.
4. DiVincenzo, D. P. & IBM. The Physical Implementation of Quantum Computation. doi:10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E. eprint: arXiv:quant-ph/0002077 (2000).
5. Stricker, R. *Gatteroperationen hoher Güte in einem optischen Quantenbit,* Magisterarb. (Universität Innsbruck, Institut für Experimentalphysik, AG Quantenoptik und Spektroskopie, 2017).
6. Negnevitsky, V. persönliche Rücksprachen über den Zeitraum der Arbeit. 2016 - 2017.
7. Pham, P. T. T. *A general-purpose pulse sequencer for quantum computing,* Magisterarb. (Massachusetts Institute Of Technology, Department of Electrical Engineering und Computer Science, 2005).
8. Born, M. Zur Quantenmechanik der Stoßvorgänge. *Zeitschrift für Physik* **37,** 863–867. ISSN: 0044-3328 (1926).
9. Hassani, S. *Mathematical Physics - A Modern Introduction to Its Foundations* ISBN: 9783319011943. doi:10.1007/978-3-319-01195-0 (Springer, 2013).
10. Haeffner, H., Roos, C. F. & Blatt, R. Quantum computing with trapped ions. doi:10.1016/j.physrep.2008.09.003. eprint: arXiv:0809.4368 (2008).
11. Barton, P. A. *u. a.* Measurement of the lifetime of the $3d^2D_{5/2}$ state in $^{40}Ca^+$. *Phys. Rev. A* **62,** 032503 (3 2000).
12. Staanum, P., Jensen, I. S., Martinussen, R. G., Voigt, D. & Drewsen, M. Lifetime measurement of the metastable $3d^2D_{5/2}$ state in the $^{40}Ca^+$ ion using the shelving technique on a few-ion string. *Phys. Rev. A* **69,** 032503 (3 2004).
13. Jin, J. & Church, D. A. Precision lifetimes for the Ca^+ 4p ^{2}P levels: Experiment challenges theory at the 1% level. *Phys. Rev. Lett.* **70,** 3213–3216 (21 1993).
14. Gerritsma, R. *u. a.* Precision measurement of the branching fractions of the 4p 2P3/2 decay of Ca II. *The European Physical Journal D* **50,** 13–19. ISSN: 1434-6079 (2008).
15. Chuang, I. L. & Nielsen, M. A. Prescription for experimental determination of the dynamics of a quantum black box. *Journal of Modern Optics* **44,** 2455–2467 (1997).

16. Knill, E. *u. a.* Randomized benchmarking of quantum gates. *Phys. Rev. A* **77,** 012307 (1 2008).

17. Gottesman, D. The Heisenberg Representation of Quantum Computers. eprint: arXiv:quant-ph/9807006 (1998).

18. King, C. The capacity of the quantum depolarizing channel. *IEEE Transactions on Information Theory* **49,** 221–229. ISSN: 0018-9448 (2003).

19. Magesan, E., Gambetta, J. M. & Emerson, J. Scalable and Robust Randomized Benchmarking of Quantum Processes. *Phys. Rev. Lett.* **106,** 180504 (18 2011).

20. Epstein, J. M., Cross, A. W., Magesan, E. & Gambetta, J. M. Investigating the limits of randomized benchmarking protocols. *Phys. Rev. A* **89,** 062321 (6 2014).

21. Roos, C. F. *Controlling the Quantum State of Trapped Ions,* Dissertation, Universität Innsbruck (2000).

22. Cohen-Tannoudji, C., Dupont-Roc, J. & Grynberg, G. *Atom – Photon Interactions, Basic Processes and Applications* ISBN: 0471625566 (Wiley-VCH Verlag GmbH, 1992).

23. Korpel, A. Acousto-optics - A review of fundamentals. *Proceedings of the IEEE* **69,** 48–53. ISSN: 0018-9219 (1981).

24. Young, J. E. H. & Yao, S.-K. Design considerations for acousto-optic devices. *Proceedings of the IEEE* **69,** 54–64. ISSN: 0018-9219 (1981).

25. Bragg, W. L. The Structure of Some Crystals as Indicated by Their Diffraction of X-rays. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* **89,** 248–277. ISSN: 0950-1207 (1913).

26. Schindler, P. *Frequency synthesis and pulse shaping for quantum information processing mit trapped ions,* Dissertation, Universität Innsbruck, (2008).

27. *Zynq-7000 All Programmable SoC Overview* DS190, V1.10. Xilinx Incorporated (Sep. 2016). https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

28. *Fundamentals of Direct Digital Synthesis (DDS)* MT-085. Analog Devices (Okt. 2008). http://www.analog.com/media/en/training-seminars/tutorials/MT-085.pdf.

29. *AD9910 1 GSPS, 14-Bit, 3.3 V CMOS Direct Digital Synthesizer* Rev. E. Analog Devices (Feb. 2017). http://www.analog.com/media/en/technical-documentation/data-sheets/AD9910.pdf.

30. *Cyclone FPGA Family Datasheet* Version 1.2. Altera Corporation (März 2003). https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ds/ds_cyc.pdf.

31. Negnevitsky, V. *u. a. Repeated QND measurements and feedback control: M-ActION progress,* ETH Zürich, am 11. 5. 2017 im Rahmen des EQUAL Site Visit in Innsbruck. (2017).

32. *MessagePack efficient binary serialization format* online abgerufen, zuletzt am 24.05.2017. MessagePack, Sadayuki Furuhashi (2008-2013). https://www.msgpack.org.

33. Donley, E. A., Heavner, T. P., Levi, F., Tataw, M. O. & Jefferts, S. R. Double-pass acousto-optic modulator system. *Review of Scientific Instruments* **76,** 063112 (2005).

34. *Adjustable Gain Avalanche Photodetectors* APD430X Operation Manual. Thorlabs (Sep. 2015). https://www.thorlabs.com/drawings/bf587a2f1215ffd8-E97F064F-FB08-4F43-600E23C53B4DCF9B/APD430A_M-Manual.pdf.
35. *Free Space Acousto-Optic Frequency Shifters* Specifications. Brimrose (). http://www.brimrose.com/pdfandwordfiles/AO_Frequency_Shifter.pdf.
36. *Acousto Optic Frequency Shifters* Model ATM-A1-A2 Series. IntraAction Corp. (Jan. 2011).
37. *Reference Specifications* AOMO 3200-124. Gooch & Housego (Juni 2002). https://www.goochandhousego.com/wp-content/pdfs/3200_124_97_01544_01rF.pdf.
38. Oppenheim, A. & Schafer, R. *Discrete Time Signal Processing* 2. Aufl. ISBN: 978-1292025728 (Pearson, 1999).
39. Wong, K. K. *Properties of Lithium Niobate* ISBN: 0852967993 (Institution of Electrical Engineers (INSPEC), 2002).
40. Marshall, G. & Stutz, G. *Handbook of Optical and Laser Scanning, Second Edition* ISBN: 9781439808801. https://books.google.at/books?id=PJjLBQAAQBAJ (CRC Press, 2016).
41. Tietze, U. & Schenk, C. *Advanced Electronic Circuits* ISBN: 9783642812439 (Springer, 1978).
42. Nielsen, M. A. & Chuang, I. L. *Quantum Computation and Quantum Information* ISBN: 978-1-107-00217-3 (Cambridge University Press, 2010).
43. Brouard, S. & Plata, J. Internal-state dephasing of trapped ions. *Phys. Rev. A* **68,** 012311 (1 2003).