Institut für integrierte Photonik

der

RWTH Aachen University

Institutsprojekt

Optischer Link

Teilnehmer:
Raphael Brandis
Kevin Läufer

Betreuer:
Johannes Hauck

31. August 2014

Inhaltsverzeichnis

1 Aufgabenstellung

Wir sind Studenten der Elektrotechnik an der RWTH Aachen und haben unser Viertsemester-Institutsprojekt am Institut für Integrierte Photonik (IPH) absolviert. Im Rahmen dieses Projekts haben wir uns mit der Übertragung von Informationen über optische Links beschäftigt.

Im ersten von zwei Projektteilen ging es zunächst um Informationsübermittlung mittels eines extern modulierten Laserstrahls. Anschließend sollte im zweiten Abschnitt des Projekts eine Laserdiode direkt moduliert und auf diese Weise digitale Daten übertragen werden.

2 Extern modulierter Laser

Um mit einem mit konstanter Intensität strahlenden Laser Informationen zu übertragen, muss mindestens ein Parameter des Laserstrahls extern moduliert werden. Mögliche Modulationsparameter sind beispielsweise die Amplitude oder die Phase des Laserstrahls. In unseren Versuchen kam ausschließlich Amplitudenmodulation zum Einsatz.

Wenn die Amplitude eines Laserstrahls extern moduliert werden soll, sind wiederum verschiedene Aufbauten als Modulator denkbar. Ein Lautsprecher mit einem in der Mitte der Membran angebrachten kleinen Spiegel kann je nach Auslenkung der Membran die Intensität des auf der Photodiode des Empfängers auftreffenden Lichts verändern; mit dieser Technik hat sich eine zweite Projektgruppe genauer beschäftigt. Wir haben uns stattdessen für die Variante entschieden, bei der ein einzelner LCD-Pixel[1] zum Einsatz kommt.

2.1 LCD-Modul als Modulator

Ein einzelner Pixel eines LCD besteht üblicherweise aus einer Flüssigkristallschicht, an die über zwei Elektroden ein elektrisches Feld angelegt werden kann, sowie einem horizontalen und einem vertikalen Polarisationsfilter, je einer vor und einer hinter der Kristallschicht.

Wären die Flüssigkristalle nicht vorhanden, würden beide Polarisationsfilter zusammen das gesamte einfallende Licht herausfiltern. Die zusätzliche Kristallschicht zwischen beiden Filtern sorgt jedoch für eine Drehung der Polarisation des durch den ersten Filter polarisierten Lichts. Da sich die Ausrichtung der Kristalle über das angelegte Feld beziehungsweise über die angelegte Spannung steuern lässt, lässt sich somit auch die Menge des insgesamt durchgelassenen Lichts steuern.

[1] engl. *liquid crystal display*

Da in unseren Versuchsaufbauten eine Laserdiode als Lichtquelle diente, war das in den Modulator einfallende Licht bereits polarisiert. Der von uns verwendete LCD-Pixel musste daher nur über einen Polarisationsfilter hinter der Kristallschicht verfügen.

2.1.1 Übertragungscharakteristik

Um das Übertragungsverhalten des verwendeten LCD-Modulators zu ermitteln, haben wir zunächst einen sehr einfachen Versuchsaufbau verwendet: Der Modulator wurde direkt an einen *Rigol DG1022*-Signalgenerator angeschlossen, als Empfänger diente eine kommerziell erhältliche Photodiode mit integrierter Biasspannung. Mit dem Oszilloskop konnten wir nun die Amplitude des empfangenen Signals bei Übertragung von Sinusschwingungen verschiedener Frequenzen messen.

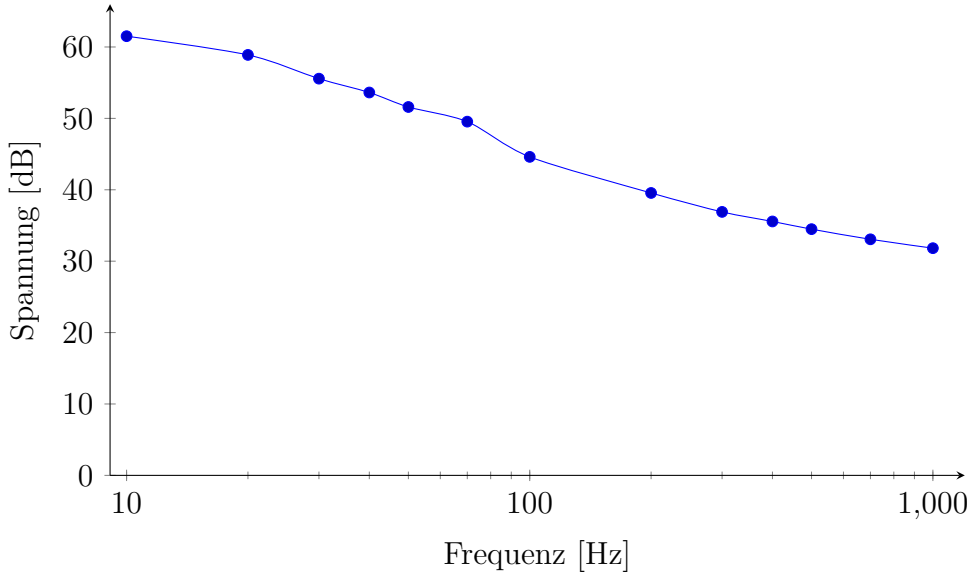


Abbildung 1: Übertragungscharakteristik des LCD-Modulators (Trägerfrequenz: 100 kHz, Amplitude: ±6 V)

In unseren Messungen fiel die Amplitude schon im niedrigen zweistelligen Frequenzbereich stark ab. Bei einer von 10 Hz ausgehenden Messreihe war der −3 dB-Punkt bereits bei cirka 20 Hz erreicht. Durch Verwendung einer hochfrequenten Trägerwelle, die mit dem niederfrequenten Nutzsignal amplitudenmoduliert wurde[2], ließ sich das Übertragungsverhalten geringfügig verbessern (siehe Abbildung 2). Insgesamt mussten wir dennoch feststellen,

[2] Diese Funktion stellte der verwendete Signalgenerator von Haus aus bereit.

dass sich der von uns verwendete LCD-Pixel aufgrund seiner geringen Geschwindigkeit nur eingeschränkt als Modulator zur Informationsübertragung eignet (siehe Frequenzgang in Abbildung 1).

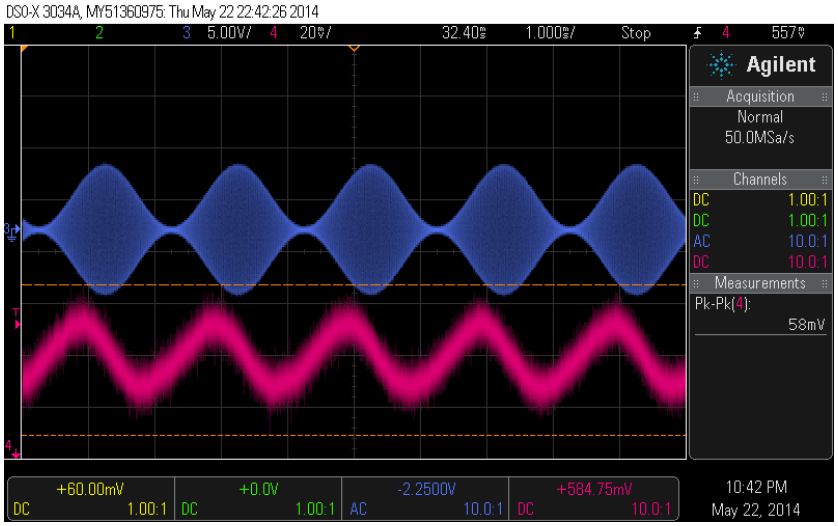


Abbildung 2: Blau: AM-modulierte Trägerwelle, Rosa: Empfangenes Signal

2.2 Übertragung von Audiosignalen

Im nächsten Schritt sollten statt einfachen Sinussignalen, die sich direkt im Signalgenerator erzeugen lassen, komplexere Audiosignale mithilfe unseres analogen Links übertragen werden — beispielsweise Musik.

Aufgrund der in Abschnitt 2.1.1 erläuterten Versuchsergebnisse war es unser Ziel, den Modulator weiterhin mit einer hochfrequenten, amplitudenmodulierten Trägerwelle anzusteuern. Da der Modulationseingang des Signalgenerators nach einem Pegel von $\pm 5\,\mathrm{V}$ verlangte, haben wir zunächst auf dem Breadboard eine diskrete Kopfhörerverstärkerschaltung aufgebaut, die das aus einem Smartphone oder einem Computer kommende Musiksignal auf den nötigen Pegel verstärkt.

Auf der Empfängerseite kam eine ebenfalls diskret aufgebaute Transimpedanzverstärkerschaltung zum Einsatz, an die zwei aktive Lautsprecher angeschlossen wurden. Der Aufbau aus Laserdiode, LCD-Modulator und Photodiode mit Transimpedanzverstärker ist in Abbildung 3 zu sehen.

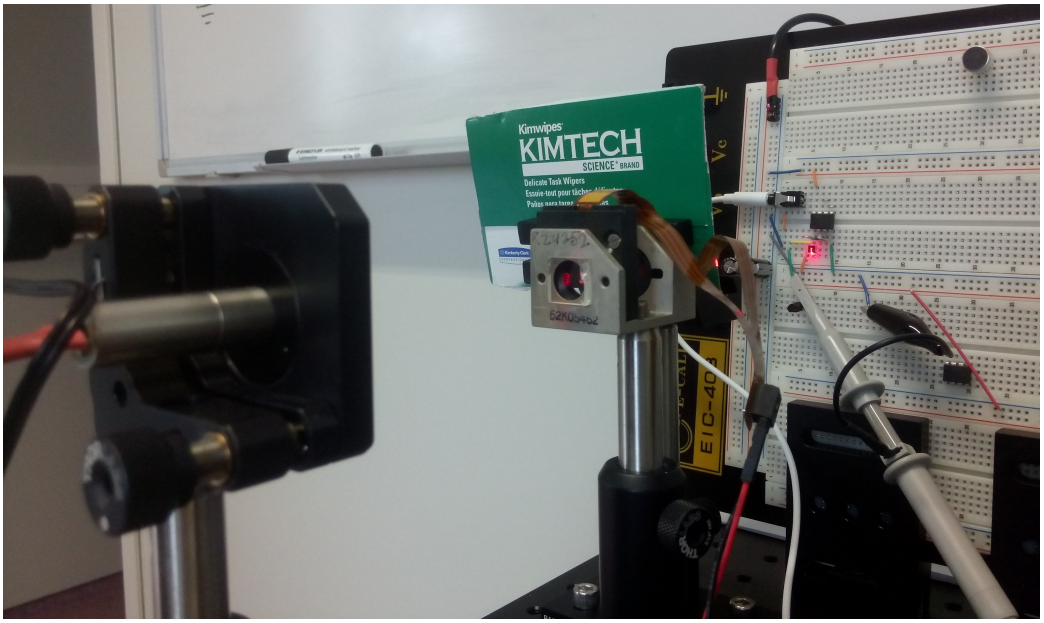
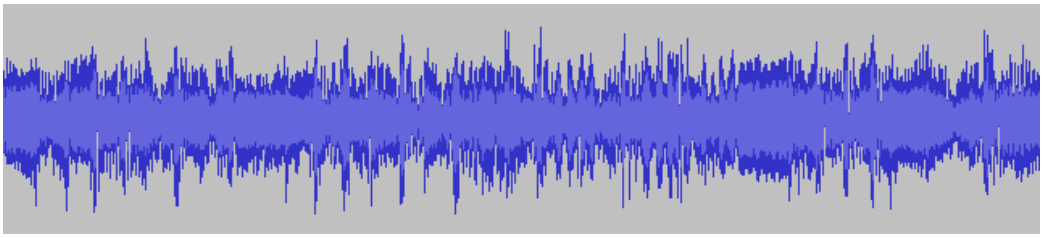
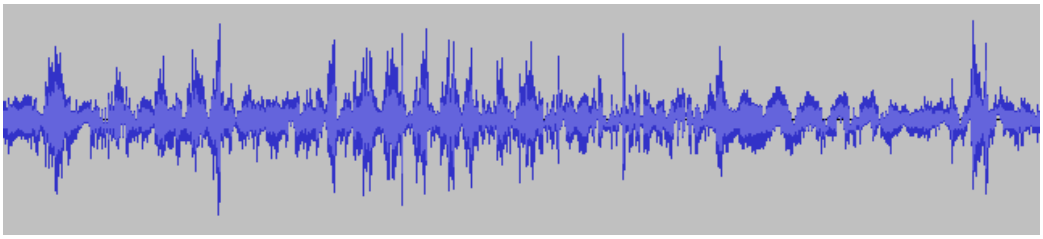


Abbildung 3: Versuchsaufbau zur Übertragung von Audiosignalen

War bei der Übertragung einzelner Sinustöne der Ton noch bis in hohe Frequenzbereiche (>10 kHz) hörbar, so verschlechterte sich das Übertragungsverhalten für breitbandige Musiksignale sehr stark. Erst durch eine Bandbreitenbegrenzung der Musik auf Frequenzen bis maximal 750 Hz wurden einzelne Basstöne aus dem Ursprungssignal erkennbar, die jedoch weiterhin von starken Störgeräuschen und Verzerrungen begleitet wurden.



(a) Bandbreitenbegrenztes Ursprungssignal



(b) Mit einem Mikrofon aufgezeichnete Ausgabe der Lautsprecher

Abbildung 4: Übertragung eines Musiksignals

In Abbildung 4 ist die geringe Übereinstimmung der Wellenformen des zu übertragenden Musiksignals (4a) und des aus den Lautsprechern zu hörenden Signals (4b) zu erkennen.

Auch bei reinen Sprachsignalen reichte die Übertragungsqualität bei weitem nicht aus, um Sprachverständlichkeit zu gewährleisten.

2.3 Übertragung digitaler Daten

Nach den Experimenten mit der analogen Signalübertragung war es von Interesse, auch einmal digitale Daten über den bestehenden optischen Link zu übertragen.

2.3.1 USB-UART-Dongle

In einem ersten Ansatz wurden günstige UART[3]-USB-Adapter verwendet. Die Adapter arbeiten mit 3.3 V-CMOS-Signalen, die mittels eines Komparators auf den nötigen Pegel von ±5 V gebracht werden. Mit diesem Pegel lässt sich der Eingang des *Rigol DG1022*-Signalgenerators ansteuern. Bei Amplitudenmodulation führen dann +5 V zur maximalen Amplitude und -5 V zu einer Amplitude von fast Null. So ließ sich ein einfaches On-Off-Keying umsetzen.

[3] Universal Asynchronous Receiver / Transmitter

Leider war es nicht möglich, bei den verwendeten UART-Adaptern Baudraten unter 1 kHz einzustellen. Dies war aber deutlich zu schnell für das LCD, wie man in Abbildung 5 sehen kann. Hier wurde am Computer eine Taste im Terminal gedrückt gehalten, um immer den gleichen Wert zu senden.

Man beachte, dass bei dieser und allen folgenden Messungen der grüne Kanal, also die Messung am Ausgang des Transimpedanzverstärkers, auf AC Coupling eingestellt war. Somit sind die Messergebnisse nicht verlässlich und es lassen sich höchstens Tendenzen feststellen. Hier wäre es interessant, die Messungen noch einmal mit richtigen Einstellungen zu wiederholen.

Dennoch ist klar erkennbar, dass ein kurzes *High* am Ausgang des UART-Adapters (blauer Kanal) zu wenig Änderung am Ausgang des Transimpedanzverstärkers führt. Das LCD kann dem Signal vermutlich nicht schnell genug folgen.

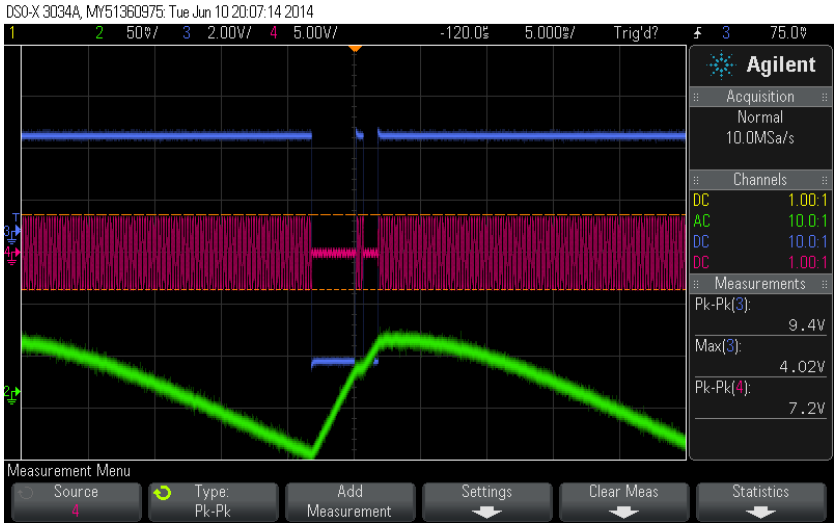


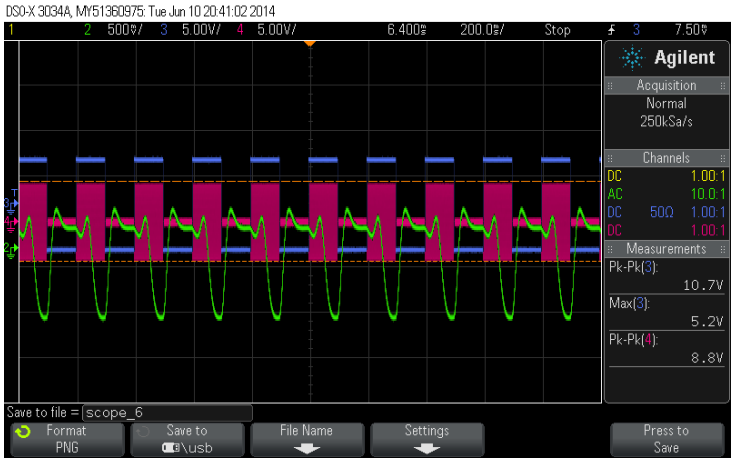
Abbildung 5: Hier wird kontinuierlich der gleiche Wert mittels UART gesendet.

2.3.2 Datenübertragung mit 10 Hz

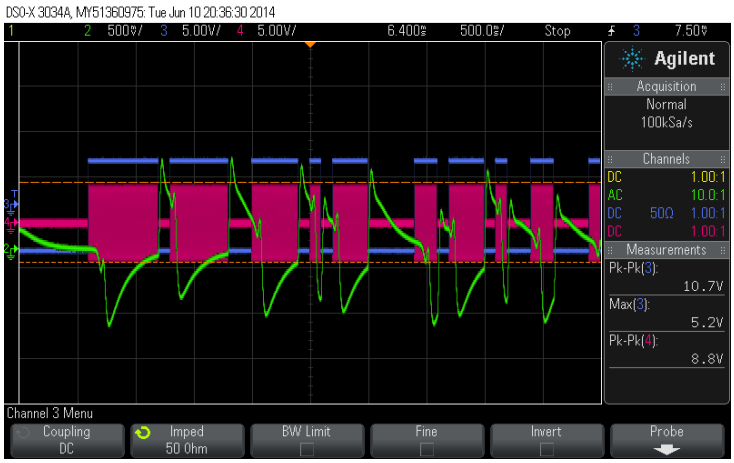
Nachdem die UART-Adapter nicht langsam genug Daten liefern konnten, wurde als nächstes ein weiterer Signalgenerator verwendet, der in der Lage war, Bitsequenzen mit einer bestimmten Frequenz zu erzeugen, in unserem Fall mit 10 Hz. Diese Bitsequenzen wurden genutzt, um mittels On-Off-Keying das LCD anzusteuern.

Interessant ist hierbei der Unterschied zwischen einem determinierten Muster aus sich abwechselnden Nullen und Einsen und einem Zufallsmuster. Wie in Abbildung 6 gut zu sehen ist, kann sich das LCD auf eine konstante

Frequenz einschwingen. Bei einem Zufallssignal hingegen hängt das Signal von der Vorgeschichte ab. Hier wird deutlich, warum bei Messungen der Signalqualität (z.B. bei der Erstellung von Augendiagrammen, siehe 3.4) ein Zufallssignal verwendet werden sollte und ein deterministisches Signal nicht ausreicht.



(a) Abwechselnde Nullen und Einsen



(b) Zufälliges Signal

Abbildung 6: Übertragung verschiedener 10 Hz-Signale

2.3.3 Software-UART mit Mikrocontrollern

Um nun endlich digitale Nutzdaten zu übertragen, haben wir auf einem Mikrocontroller UART in Software implementiert, um die verwendete Baudrate beliebig niedrig wählen zu können. Hierzu wurden zwei *STMicroelectronics Discovery Boards* verwendet, eines mit *STM32F4*- und eines mit

STM32F3-Mikrocontroller. Der Quellcode nutzt die *XPCC*-Bibliothek und ist auf `http://github.com/ekiwi/iph` abrufbar. Die wichtigsten funktionalen Bestandteile sind aber auch im Anhang A zu finden.

Zur Verbesserung der Robustheit der Übertragung kam ein (7,4)-Hamming-Code zum Einsatz, bei dem alle gültigen Codewörter untereinander einen Hammingabstand von mindestens drei besitzen. Somit kommen auf vier Datenbits drei Paritätsbits, was die Korrektur einfacher Bitfehler erlaubt. Ein möglicher Algorithmus hierfür sieht wie folgt aus (siehe auch Quellcode in Anhang A.1):

Für ein empfangenes Wort wird der Hammingabstand zu jedem gültigen Codewort errechnet. Beträgt der Abstand eins oder null, wird davon ausgegangen, dass das empfangene Wort dem Codewort entspricht. Bei zweifachen Bitfehlern führt dies zu einer falschen Zuordnung: Da der Hammingabstand zwischen zwei gültigen Codeworten drei beträgt und bei zwei Bitfehlern der Abstand zum ursprünglichen Codewort zwei ist, gibt es ein anderes Codewort, zu dem die empfangene Sequenz nur einen Hammingabstand von eins hat. Diesem Codewort wird das empfangene Wort fälschlicherweise zugeordnet.

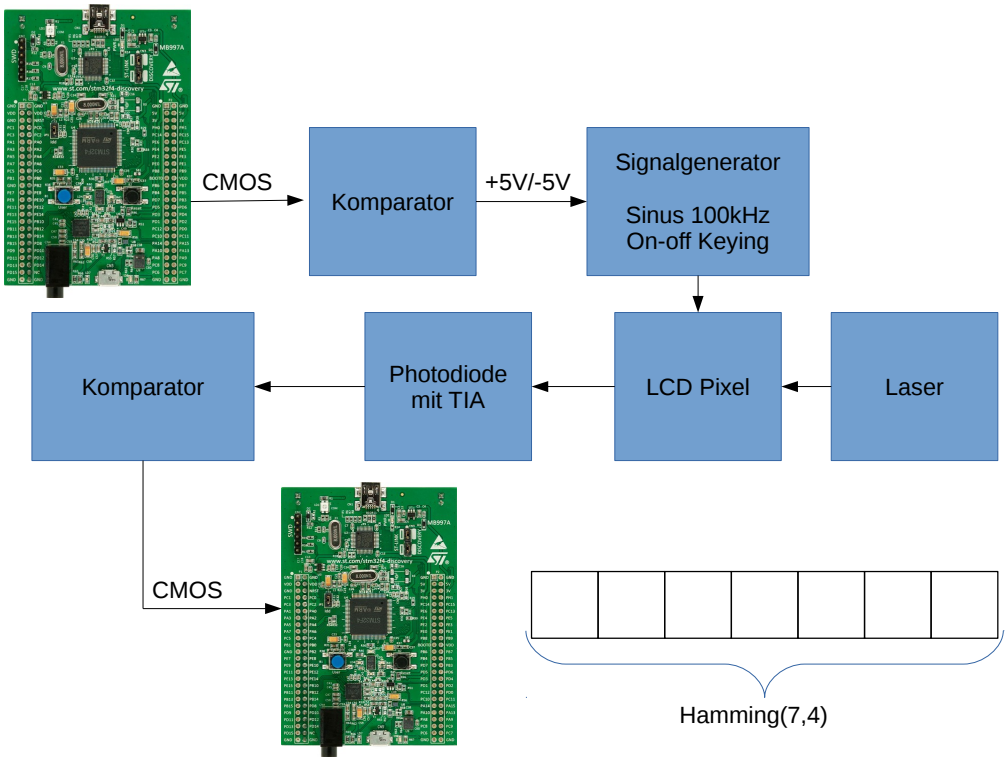


Abbildung 7: Software-UART-Signalkette

Zum Test der Übertragung wurden 294 Datenwörter in (7,4)-Hamming-Codierung übertragen. Der Versuchsaufbau ist in Abbildung 7 dargestellt. Die Positionierung des Lasers und der Empfängerdiode entsprachen dem in Abbildung 3 gezeigten Aufbau. Der Inhalt der Datenpakete waren die Zahlen von 0 bis 15. Dies reduziert die Aussagekraft des Experiments etwas im Vergleich zu randomisierten Daten, erleichterte aber die Auswertung erheblich. Um nämlich zwei oder mehr Bitfehler erkennen zu können, müssen die gesendeten Daten am Empfänger bekannt sein. Somit gelangten wir zu der in Abbildung 8 dargestellten Fehlerverteilung.

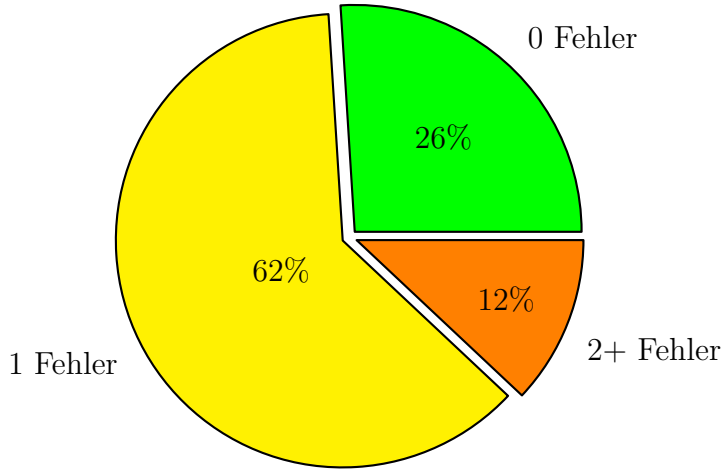


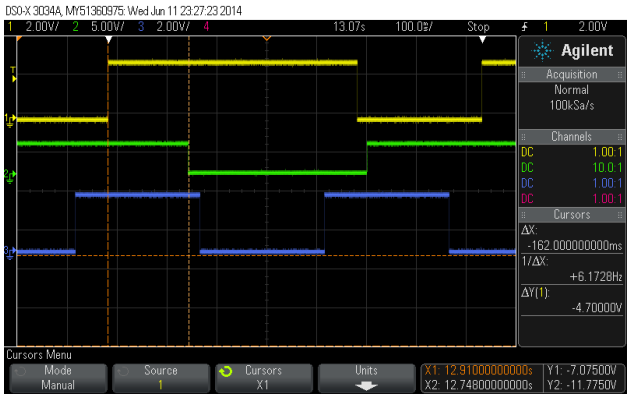
Abbildung 8: Fehler bei der Übertragung von 294 (7,4)-Hamming-codierten, deterministischen Datenworten mit einem 4 Hz-UART-Signal

Demnach sind, nach Fehlerkorrektur durch den Hammingcode, immer noch 12 % der empfangenen Daten falsch und müssten in einem echten Übertragungssystem durch höhere Layer abgefangen werden. Die Rate der Datenbits berechnet sich wie folgt:

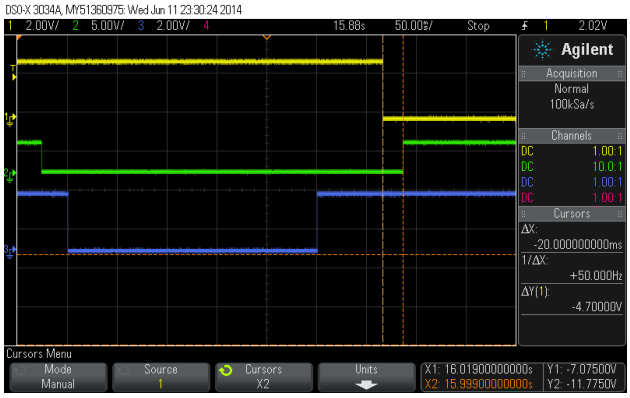
$$4\,\mathrm{bit/s} * \frac{4\,\mathrm{bit}}{7\,\mathrm{bit}} \approx 2.3\,\mathrm{bit/s} \tag{1}$$

Somit wurden über etwa 30 cm mit 2.3 bit/s und einer recht hohen Fehlerrate von 12 % Daten übertragen — kein sehr gutes Ergebnis.

Doch wie ist dieses schlechte Resultat zu erklären? Für eine Erklärungsversuch lohnt es sich, die Messungen in Abbildung 9 zu betrachten. Hier sieht man, dass eine zufällig ausgewählte steigende Flanke (9a) 162 ms benötigt, um durch das System zu propagieren. Eine fallende Flanke (9b) hat mit 20 ms eine deutlich kleinere Verzögerung. Weitere beobachtete Flanken wiesen Verzögerungen in der gleichen Größenordnung auf.



(a) steigende Flanke: 162 ms



(b) fallende Flanke: 20 ms

Abbildung 9: Verzögerung bei einem 4 Hz-UART-Signal; Quellensignal in gelb und invertiertes Ausgangssignal in grün

Da bei einem zufälligen Bit eine steigende Flanke sowohl an dessen Anfang als auch an dessen Ende auftreten kann, müsste man demnach mehr als 162 ms nach Anfang und 162 ms vor Ende eines Bits sampeln, um Fehler ganz zu vermeiden. Dies führt zu folgender maximaler Datenrate:

$$T_{min} = 162\,\text{ms} * 2 = 324\,\text{ms} \tag{2}$$

$$f_{max} = \frac{1}{T_{min}} = \frac{1}{324\,\text{ms}} \approx 3\,\text{bit/s} \tag{3}$$

Ein Absenken der Bitrate auf 3 bit/s in Kombination mit einer intelligenten Wahl des Samplingzeitpunktes könnte also eine Hamming-Codierung überflüssig machen und somit zu einer höheren Datenrate führen. Hierfür müssen aber die Latenzen bekannt sein und bei der Synchronisation durch die erste fallende Flanke (siehe Anhang A.3, Zeile 39) berücksichtigt werden.

Als wir bei einem Test ohne Kenntnis der Latenz mit einer Datenrate von 2 bit/s Daten übertragen haben, traten fast keine Bitfehler auf.

Die mit den oben aufgeführten Berechnungen ermittelte maximale theoretische Datenrate von etwa 3 bit/s ist jedoch weiterhin sehr niedrig. Interessant wäre es daher, die Ursache der Latenz zu untersuchen. Hierbei sollte bei verschiedenen Ansteuerungsarten des LCD die erzielte Latenz gemessen werden. Mit den oben genannten Berechnungen lässt sich daraus die mögliche Datenrate berechnen.

3 Direkt modulierter Laser

Im zweiten Teil des Projekts sollte ein Diodenlaser direkt moduliert werden, um damit digitale Daten zu übertragen. Dazu hatte die andere Gruppe bereits eine Modulationsmethode untersucht, bei der auf einen konstanten Strom ein Datensignal aufgekoppelt wurde.

Unser Ziel war es, eine Stromquelle direkt mit einem Datensignal zu modulieren und danach Empfängerdesigns zu evaluieren.

3.1 Sender: Modulierte Konstantstromquelle

Aufgrund der exponentiellen Kennlinie einer Laserdiode muss diese mit einem konstanten Strom versorgt werden. Eine einfache Konstantstromquelle lässt sich mit zwei Bipolartransistoren aufbauen, wie in Abbildung 10 zu sehen ist.

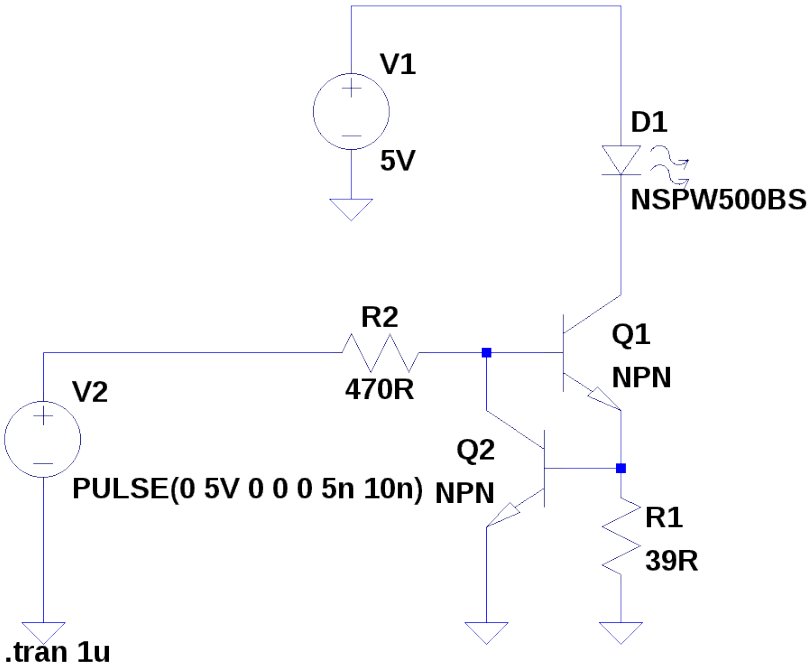


Abbildung 10: Modulierte Stromquelle.

Der Strom I_{D1}, der durch die Diode fließt, entspricht hierbei in etwa dem Strom I_{R1}, der durch den Widerstand $R1$ fließt. An diesem müssen, durch den Transistor $Q2$ vorgegeben, etwa 0.7 V abfallen, wodurch sich folgender Strom einstellt:

$$I_{D1} = I_{R1} = \frac{U_{R1}}{R1} \approx \frac{0.7\,\mathrm{V}}{R1} \tag{4}$$

Diese Konstantstromquelle lässt sich leicht ein- und ausschalten, indem über einen Widerstand die Summe der Basis-Emitter-Spannungen beider Transistoren, $U_{BE1} + U_{BE2}$, vorgegeben wird.

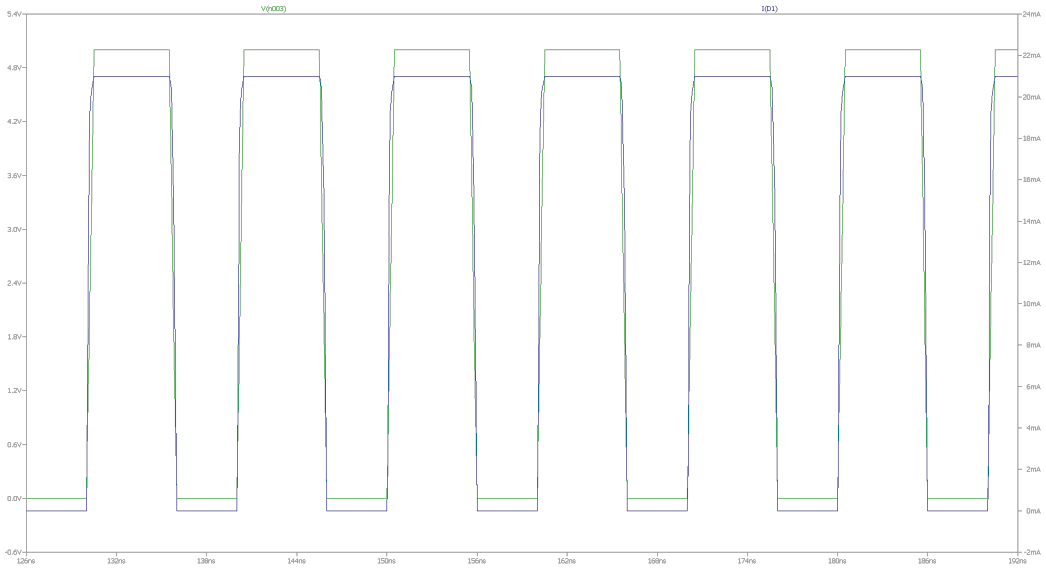


Abbildung 11: Transientenanalyse der modulierten Stromquelle

Eine Simulation des Aufbaus in LTSpice zeigt, dass der Strom I_{D1} durch die Diode von der Eingangsspannung U_2 abhängt. Bei den in Abbildung 11 dargestellten Ergebnissen wurde zwar mit einem vereinfachten Transistormodell simuliert, sie sind also mit Vorsicht zu genießen. Das Funktionsprinzip wird durch die Simulation dennoch gut veranschaulicht.

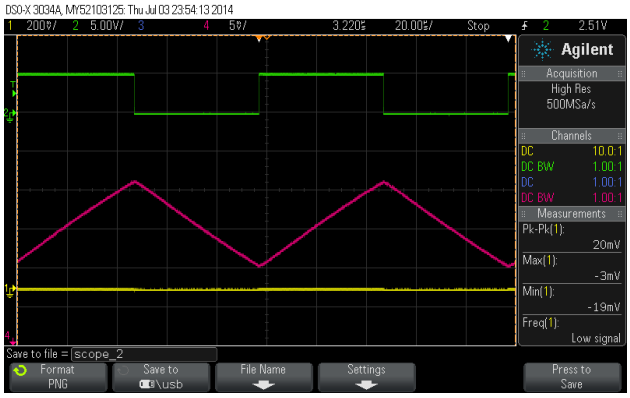
Abbildung 12: Reflexionen bei einem 20 MHz-Eingangssignal

Mit diesem Aufbau ließen sich in unseren Versuchen recht schnelle Übertragungen von zufälligen Bits realisieren (siehe auch Abschnitt 3.2). Es gab jedoch aufgrund der fehlenden Terminierung auf der Eingangsseite unserer Schaltung Probleme mit Reflexionen auf der Leitung (siehe Abbildung 12).

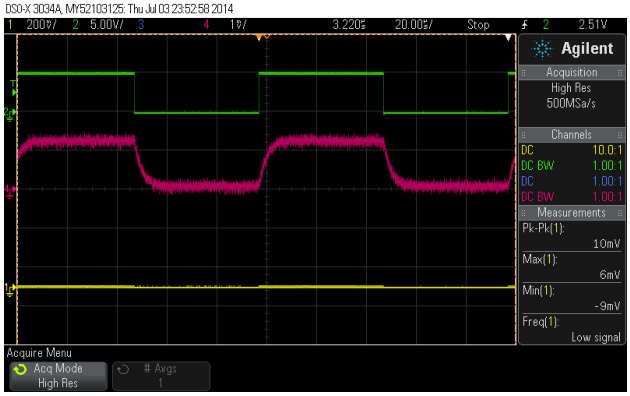
Hier wäre eine Untersuchung verschiedener Terminierungsvarianten interessant.

Auch verhält sich die Schaltung bei Frequenzen im Megahertzbereich immer weniger wie in der Simulation: So entspricht die Spannung am Widerstand $R1$ nicht mehr den erwarteten 0.7 V. Hier wären eine Verbesserung der Spice-Simulation durch die Auswahl eines passenden Transistormodells und durch bessere Modellierung der Quelle sowie weitere empirische Versuche angebracht.

3.2 Photodiode mit Lastwiderstand als Empfänger



(a) 10 kΩ



(b) 50 Ω

Abbildung 13: Grün: Gesendetes 20 kHz-Rechtecksignal; Rosa: Empfangenes Signal bei unterschiedlichem Lastwiderstand

Zum Testen der Sendeschaltung nutzten wir zunächst eine Photodiode mit Biasspannung und umschaltbarem Lastwiderstand (*NewFocus 1623*) als Empfänger.

In Abbildung 13 ist die Abhängigkeit des Ausgangssignals vom Lastwiderstand zu erkennen. Zur Aufnahme dieser Messungen wurde der Ausgang der Photodiode mit einem hochohmig terminierten Eingang eines Oszilloskops verbunden und auf den Eingang unserer Sendeschaltung ein 20 kHz-Rechtecksignal gegeben. Bei einem hohen Lastwiderstand (13a) erhöht sich die Verstärkung: Die Amplitude beträgt hier in etwa 10 mV. Jedoch sinkt die Bandbreite, was sich durch sehr lange Anstiegs- und Abfallzeiten und die daraus resultierende Dreiecksform des Ausgangssignals bemerkbar macht. Mit einem kleineren Lastwiderstand (13b) ergibt sich ein weniger verzerrtes Signal, die Amplitude beträgt jedoch nur noch in etwa 1 mV.

Die kleinen Amplituden und die Verzerrung beim Empfang eines 20 kHz-Signals weisen darauf hin, dass für schnellere Signale ein besserer Verstärker benötigt wird.

3.3 Empfänger mit integriertem *AD8015* Transimpedanzverstärker

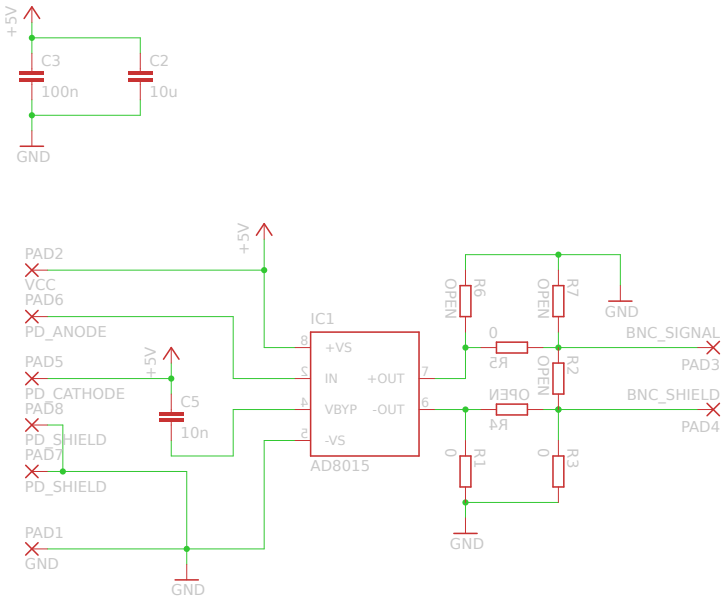


Abbildung 14: Beschaltung des *AD8015*-Transimpedanzverstärkers

Anstatt selbst eine Empfängerschaltung aufzubauen, bietet es sich an, einen integrierten Transimpedanzverstärker wie den *AD8015* zu verwenden. Dieser ist von Grund auf dafür konzipiert, den Photostrom in eine proportional verstärkte Spannung zu wandeln, die differentiell mit einer Impedanz von

50 Ω am Ausgang abgegriffen werden kann. Es werden nur wenige externe Komponenten benötigt, was zu einer einfachen Schaltung führt (siehe Abbildung 14).

Mit diesem Aufbau lassen sich auch schnelle Signale verstärken, wie in der Betrachtung der Augendiagramme im nächsten Abschnitt deutlich wird.

3.4 Augendiagramme

Augendiagramme können mit dem Oszilloskop aufgenommen werden. Hierzu wird der Trigger auf steigende und fallende Flanken des Signals eingestellt und Persistenz aktiviert. Hierdurch erhält man eine Überlagerung von steigenden und fallenden Flanken. Verbleibt zwischen diesen eine klar erkennbare Öffnung (das Auge), kann man davon ausgehen, dass die Bits auch von einer Empfängerschaltung richtig interpretiert werden können.

Wichtig ist, als Eingangssignal eine Zufallssequenz von Bits zu wählen, damit sich das System nicht einfach auf eine Frequenz einschwingt.

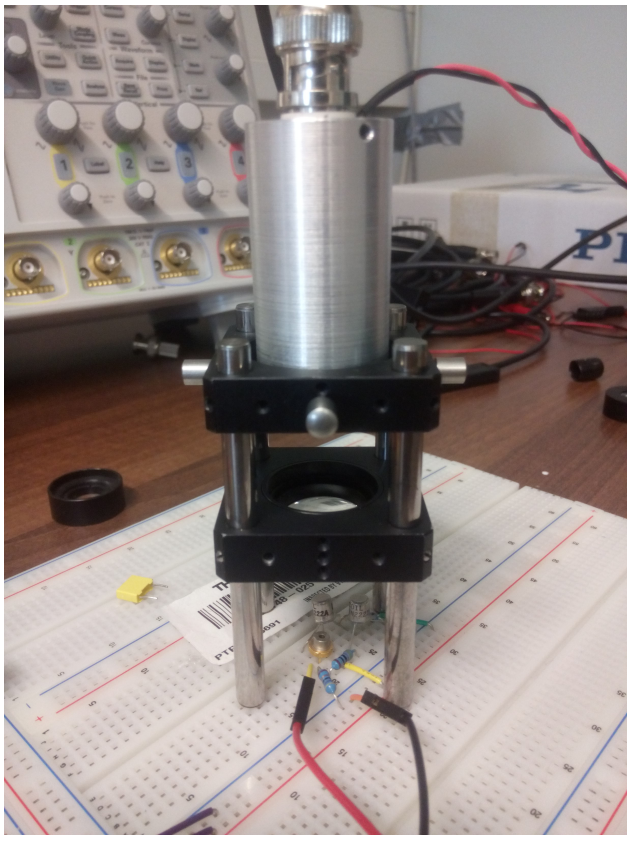
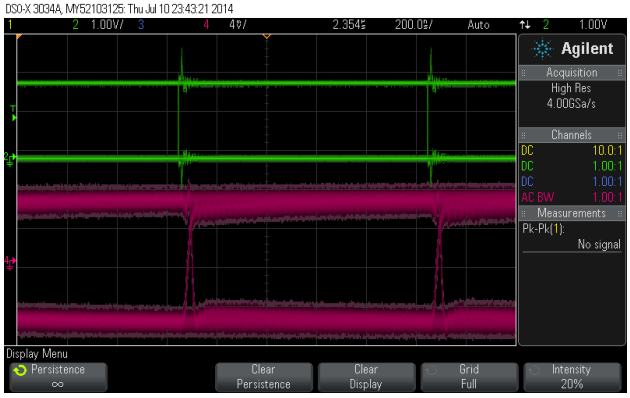


Abbildung 15: Versuchsaufbau zur Erstellung der Augendiagramme

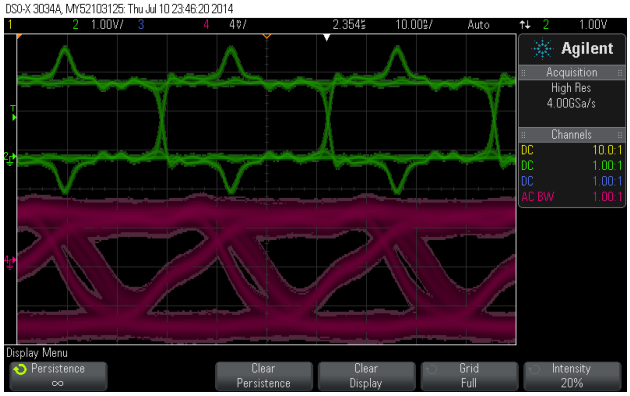
In unserem Fall wurden die Augendiagramme in Abbildung 16 und Abbildung 17 mit der Sendeschaltung aus Abschnitt 3.1 und der Empfängerschaltung aus Abschnitt 3.3 aufgenommen. Der Aufbau ist in Abbildung 15 zu sehen.



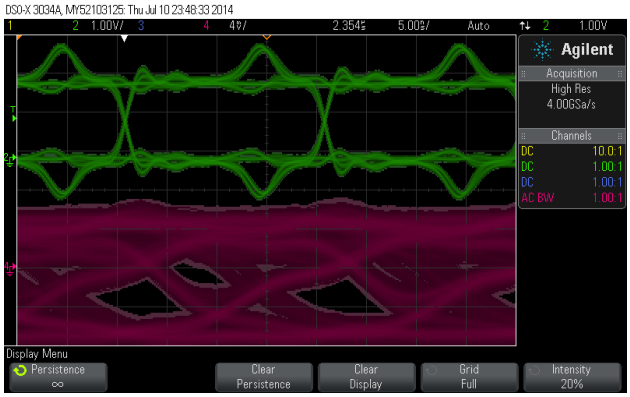
(a) 1 MHz

(b) 10 MHz

Abbildung 16: Augendiagramme (gesendetes Signal in grün)



(a) 30 MHz



(b) 50 MHz

Abbildung 17: Augendiagramme (gesendetes Signal in grün)

Ein 1 MHz-Bitsignal wird ohne Probleme übertragen (16a), auch bei einem 10 MHz-Signal ist die Augenöffnung noch unproblematisch (16b). Ein 30 MHz-Signal hat bereits eine deutlich kleinere Öffnung (17a), die Übertragung sollte hier aber auch noch funktionieren. Bei 50 MHz scheint die Grenze erreicht; hier ist eine distinkte Öffnung kaum noch zu erkennen (17b).

3.5 Ergebnisse

Mit dem System aus selbst gebauter modulierbarer Stromquelle und *AD8015*-Verstärker lassen sich Signale mit einer Geschwindigkeit von bis zu 50 MHz gut übertragen. Die Photodiode ohne Verstärker ist hingegen in diesem Geschwindigkeitsbereich nicht mehr zu gebrauchen.

Bezüglich des Senders sollte man als nächstes versuchen, die verwendete Schaltung wie in Abschnitt 3.1 beschrieben zu verbessern, um sie dann auf einer Platine aufzubauen. Zusammen mit einem Gehäuse ließe sich damit ein

Versuchsaufbau erstellen, mit dem die Schaltung noch besser untersucht und der Laser auch in eine Glasfaser eingekoppelt werden könnte.

Der Empfänger mit *AD8015* hat sich bewährt und muss wohl erst einmal nicht verändert werden. Es wäre aber interessant, ihn noch besser zu vermessen. Unter anderem sollte mit einer professionellen Lichtquelle untersucht werden, bis zu welchen Frequenzen verstärkt werden kann. Bisher wurde hier nur das Gesamtsystem aus Empfänger und Sender untersucht, was es erschwert, eindeutige Aussagen über die Einzelkomponente zu treffen. Interessant könnte es auch sein, die *AD8015*-Schaltung mit verschiedenen Photodioden aufzubauen und zu untersuchen, wie sich diese auf das Verstärkungsverhalten auswirken.

4 Fazit

Auch wenn wir, insbesondere mit dem LCD, keine besonders großen Erfolge erzielen konnten, so war das Praktikum für uns dennoch sehr interessant und lehrreich. Es wurde klar, dass die Beschreibung als LTI-System nur eine Näherung ist, die sich nicht ohne Weiteres auf das LCD übertragen lässt. Es ist daher wichtig, mit Zufallsbitfolgen zu messen, um ein Einschwingen zu verhindern. Auch bedeutet ein einzelner Sinuston, der sich in guter Qualität übertragen lässt, noch lange nicht, dass man dynamische Musiksignale, die Frequenzen aus dem gesamten hörbaren Spektrum enthalten, übertragen kann.

Im Nachhinein fallen einem zahlreiche Dinge ein, die man hätte besser machen können. So sei Studenten, die in Zukunft dieses Praktikum durchführen, empfohlen, von Anfang an strukturiert vorzugehen. So sollte man sich erst überlegen, welche Kenndaten des LCDs benötigt werden, und anschließend, wie diese zu messen sind. Sehr wichtig ist außerdem, immer daran zu denken, den Versuchsaufbau zu einer Messung genau zu dokumentieren, um Wiederholbarkeit sicherzustellen. So haben wir es leider versäumt, den genauen Aufbau bei Aufnahme der ersten Messungen mit dem LCD festzuhalten.

Darüber hinaus wäre interessant gewesen, einige Messreihen mit unterschiedlichen Arten der Ansteuerung des LCDs aufzunehmen. Hier wären Latenz (also Gruppenlaufzeit) und Amplitudenübertragungsfunktion interessant. Unter anderem könnte man den LCD mit verschiedenen Amplituden sowie mit oder ohne hochfrequenter Trägerwelle ansteuern.

Interessant war auch der praktische Einsatz von Kenntnissen aus *Grundgebiete der Informatik II* bei der Übertragung von Daten mittels eines Hamming-Codes. Hier zeigte sich, dass eine niedrigere Übertragungsgeschwindigkeit ef-

fizienter sein kann, da weniger Fehler erzeugt und somit weniger Paritätsbits benötigt werden. Für die Übertragung von digitalen Daten wäre es bei einer erneuten Durchführung des Praktikums interessant, nach Charakterisierung des LCDs eine Übertragungsstrecke mit niedrigerer Latenz zu konstruieren.

Bei der direkten Modulation des Lasers empfiehlt es sich, die Schaltung des Senders wie oben beschrieben zu optimieren und dann fest aufzubauen, um Messungen einfacher und genauer durchführen zu können. Beim Empfänger mit *AD8015*-Verstärker könnte man Messungen mit einer modulierten Lichtquelle, deren Bandbreite bekannt ist, durchführen, um seine maximale Bandbreite zu ermitteln.

Gelohnt hat sich definitiv auch, in der Exkursionswoche für zwei ganze Tage am Projekt zu arbeiten. So kann man einmal etwas mehr am Stück erledigen, ohne sich Woche für Woche erneut in das Projekt hineindenken zu müssen. Wir raten aber davon ab, dass Praktikum komplett an einem Stück durchzuführen, da dann nicht genug Zeit bleibt, um Teile zu bestellen oder Platinen fertigen zu lassen.

Zu guter Letzt möchten wir den zukünftigen Teilnehmern dieses Praktikums viel Erfolg wünschen. Ihr könnt euch auf jeden Fall auf die Unterstützung des Instituts verlassen. Egal ob ihr Fragen habt, Hilfe oder Teile für die Durchführung braucht, wendet euch einfach an euren Betreuer.

Wir hoffen, dass wir euch mit unserem Projektbericht helfen konnten und würden uns freuen, in einem Jahr eure Ergebnisse zu sehen.

Viel Spaß!

A C++-Quellcode-Ausschnitte

A.1 Hamming-Code

hamming.hpp

```
1 #ifndef HAMMING_HPP
2 #define HAMMING_HPP
3
4 #include <xpcc/debug/logger.hpp>
5
6 #undef    XPCC_LOG_LEVEL
7 #define    XPCC_LOG_LEVEL xpcc::log::INFO
8
9 namespace Hamming
```

```
10  {
11
12  static constexpr uint8_t
13  hammingLut[16] = {
14  0b00000000,    // 0
15  0b00001011,    // 1
16  0b00010110,    // 2
17  0b00011101,    // 3
18  0b00100111,    // 4
19  0b00101100,    // 5
20  0b00110001,    // 6
21  0b00111010,    // 7
22  0b01000101,    // 8
23  0b01001110,    // 9
24  0b01010011,    // 10
25  0b01011000,    // 11
26  0b01100010,    // 12
27  0b01101001,    // 13
28  0b01110100,    // 14
29  0b01111111,    // 15
30  };
31
32  // encodes least significant 4 bits
33  static uint8_t
34  encode(uint8_t data) {
35      return(hammingLut[data & 0xf]);
36  }
37
38  static int
39  decode(uint8_t data) {
40      int cur_distance, min_distance = 8;
41      uint8_t data_copy, hamming_word;
42      int index = 0;
43
44      for(int ii = 0; ii < 16; ++ii) {
45          cur_distance = 0;
46          data_copy = data;
47          hamming_word = hammingLut[ii];
48          for(int jj = 0; jj < 8; ++jj) {
49              cur_distance += (static_cast<bool>(data_copy & 0x1) !=
50                  static_cast<bool>(hamming_word & 0x1));
```

```
51              data_copy = data_copy >> 1;
52              hamming_word = hamming_word >> 1;
53          }
54
55          if(cur_distance < min_distance) {
56              min_distance = cur_distance;
57              index = ii;
58          }
59      }
60      XPCC_LOG_INFO << min_distance << "," << xpcc::endl;
61      XPCC_LOG_DEBUG << "Minimum Hamming distance: " << min_distance
62          << xpcc::endl;
63      return index;
64  }
65
66  }
67
68
69  #endif // HAMMING_HPP
```

https://github.com/ekiwi/iph/blob/master/stm32/lcd_transmitter/hamming.hpp

A.2 Software-UART: Quelle

transmitter.cpp

```
1   #include "transmitter.hpp"
2   #include "../settings.hpp"
3   #include "../hamming.hpp"
4
5   #include <xpcc/architecture/platform.hpp>
6   #include <xpcc/debug/logger.hpp>
7
8   using namespace xpcc::stm32;
9
10  typedef GpioOutputA5 SendPin;
11
12
13  Transmitter::Transmitter()
14  : delay(0)
15  {
```

```
16      this->stop();
17  }
18
19  void
20  Transmitter::initialize()
21  {
22      SendPin::setOutput(xpcc::Gpio::High);
23  }
24
25
26  bool
27  Transmitter::run()
28  {
29      static bool bit;
30      static int ii;
31
32      PT_BEGIN();
33
34      delay.restart(Settings::BitPeriod);
35
36      // send startbit
37      SendPin::reset();
38      PT_WAIT_UNTIL(delay.isExpired());
39
40      // send 7 databits
41      ii = 0;
42      while(ii < 7) {
43          ++ii;
44
45          bit = static_cast<bool>(data & (1<<6));
46          // Send single bit
47          SendPin::set(bit);
48          if(bit) XPCC_LOG_DEBUG << "1";
49          else XPCC_LOG_DEBUG << "0";
50
51          PT_WAIT_UNTIL(delay.isExpired());
52
53          data = data << 1;
54      }
55
56      // send stopbit
```

```
57      SendPin::set();
58      PT_WAIT_UNTIL(delay.isExpired());
59
60      delay.restart(0);     // stop delay
61      PT_END();
62  }
63
64
65  void
66  Transmitter::send(uint8_t data) {
67      if(this->isRunning()) {
68          XPCC_LOG_ERROR << "Cannot send Data. Not finished!" << xpcc::endl;
69          return;
70      }
71
72      XPCC_LOG_DEBUG << xpcc::endl << "Send: " << data << xpcc::endl;
73      this->data =  Hamming::encode(data);
74      this->restart();
75  }
```

https://github.com/ekiwi/iph/blob/master/stm32/lcd_transmitter/source/transmitter.cpp

A.3 Software-UART: Empfänger

receiver.cpp

```
1   #include "receiver.hpp"
2   #include "../settings.hpp"
3   #include "../hamming.hpp"
4
5   #include <xpcc/architecture/platform.hpp>
6   #include <xpcc/debug/logger.hpp>
7
8   using namespace xpcc::stm32;
9
10  typedef xpcc::GpioInverted<GpioInputA5> ReceivePin;
11  typedef GpioOutputA7 SampleIndicator;
12
13
14  #undef    XPCC_LOG_LEVEL
15  #define    XPCC_LOG_LEVEL xpcc::log::INFO
```

```
16
17  Receiver::Receiver()
18  : delay(0)
19  {
20      this->stop();
21  }
22
23  void
24  Receiver::initialize()
25  {
26      ReceivePin::setInput();
27      SampleIndicator::setOutput(xpcc::Gpio::High);
28  }
29
30
31  bool
32  Receiver::run()
33  {
34      static bool bit;
35      static int ii;
36
37      PT_BEGIN();
38
39      // wait for falling edge
40      PT_WAIT_UNTIL(ReceivePin::read());
41      PT_WAIT_UNTIL(!ReceivePin::read());
42      XPCC_LOG_DEBUG << "[";
43
44      // wait for startbit + 1.bit to get transmitted
45      delay.restart(Settings::ReceiveStartDelay);
46      PT_WAIT_UNTIL(delay.isExpired());
47
48      // receive bits
49      ii = 0;
50      while(ii < 7) {
51          ++ii;
52
53          bit = ReceivePin::read();
54          SampleIndicator::toggle();
55          if(bit) XPCC_LOG_DEBUG << "1";
56          else XPCC_LOG_DEBUG << "0";
```

```
57          data = (data << 1) | (bit? 1 : 0);
58
59          delay.restart(Settings::BitPeriod);
60          PT_WAIT_UNTIL(delay.isExpired());
61      }
62
63      // sample stopbit
64      bit = ReceivePin::read();
65      SampleIndicator::toggle();
66      if(bit) {
67          XPCC_LOG_DEBUG << "]";
68      } else {
69          XPCC_LOG_DEBUG << "*E*";
70      }
71
72      // done
73      PT_END();
74  }
75
76
77
78
79  void
80  Receiver::startReceive() {
81      if(this->isRunning()) {
82          XPCC_LOG_ERROR << "Cannot receive Data. Not finished!" << xpcc::endl;
83          return;
84      }
85
86      XPCC_LOG_DEBUG << xpcc::endl << "Start receiving"<< xpcc::endl;
87      this->data = 0;
88      this->restart();
89  }
90
91  uint8_t
92  Receiver::getResult() {
93      if(this->isRunning()) {
94          XPCC_LOG_ERROR << "No result. Not finished!" << xpcc::endl;
95          return 0;
96      }
97
```

```
98      return Hamming::decode(data);
99  }
```

https://github.com/ekiwi/iph/blob/master/stm32/lcd_transmitter/sink/receiver.cpp