



Gert Freiberger BSc

Implementierung von speziellen Modulationsarten mit einem Mach-Zehnder Modulator

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Telematik / Masterstudium ICE 2015

eingereicht an der

Technischen Universität Graz

Betreuer

Ao. Univ. -Prof.-Ing Dr.techn. Erich Leitgeb

Institut für Hochfrequenztechnik

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum

Unterschrift

Implementierung von speziellen Modulationsarten mit einem Mach- Zehnder Modulator

Masterarbeit

Gert Freiberger

Institut für Hochfrequenztechnik



Betreuer:

Ao. Univ. -Prof.-Ing. Dr.techn. Erich Leitgeb

In Kooperation mit:



Kurzfassung

In dieser Arbeit wird ein optisches Übertragungssystem mittels eines Mach-Zehnder Modulators aufgebaut und eine PPM Modulation implementiert. Als optische Quelle wird ein Laser verwendet, welcher in einem früheren Projekt bereits entwickelt wurde. Dieser Mach-Zehnder Modulator, dient als externer Modulator, welcher sein Modulationssignal von einem FPGA erhält. Der FPGA dient dabei als Erzeuger und gleichermaßen als Empfänger des Signales, nachdem der optische Empfänger Dieses in ein elektrisches Signal umgewandelt hat. Für die Erzeugung des Signales besitzt der FPGA zwei Möglichkeiten. Entweder werden Daten über den PC, mittels UART übermittelt oder es wird eine festgelegte Nachricht, welche im FPGA implementiert ist, verwendet.

Der erste Teil der Arbeit beschäftigt sich mit den Hardwarekomponenten des Übertragungssystems. Es werden die einzelnen Teile genauer studiert, darunter die optische Quelle, der Mach-Zehnder Modulator, der FPGA und der optische Empfänger.

Der zweite Teil beschäftigt sich vorwiegend mit dem Programm, welches das PPM Signal für den Modulator erzeugt und die verschiedenen Schnittstellen, des FPGAs die implementiert sind, beinhaltet. Diese Schnittstellen sind zuerst die UART, welche den Datenaustausch mit dem PC verwaltet, der Transceiver der die Daten codiert und sendet und der Receiver der die Daten wieder empfängt und decodiert.

Der dritte und letzte Teil der Arbeit beinhaltet, alle wichtigen Zwischenergebnisse während der Entwicklung des Systems und die durchgeführten Tests zur Überprüfung der Funktionalität der einzelnen Komponenten und ihr wirken miteinander. Darunter sind die einzelnen Schritte bei der Verifizierung der UART, des Transceivers und des Receivers genau beschrieben.

Abstract

In this master thesis, an optical communications system with a Mach-Zehnder modulator is build and a PPM Modulation is implement. For the optical source, a laser is used, which was develop in an earlier project. The external modulator is the Mach-Zehnder modulator, which get its modulation signal from the FPGA. The FPGA is for creating the PPM Signal and receiving the Signal, after the optical receiver has converted the Signal to the electrical region. The Data for the Signal can create by two ways. The first one by sending the Data from the PC to the FPGA, via UART. The other way is by using the hardcoded data in the FPGA. This data from the FPGA used for creating the PPM Signal.

The first part of the thesis consists of the description of the hardware components. These components are the optical source, the Mach-Zehnder modulator, the FPGA and the optical receiver.

The second part shows the core components of the FPGA program. This program creates the PPM signal for the modulator. The program also includes the ports for important hardware components. This Ports used for the communication system. The important ports are the UART, the transceiver and the receiver. The UART control the communication between the PC and the FPGA. The transceiver used for coding and sending the data. For receiving and decoding the data, the receiver is used.

The last part of the thesis includes the intermediate results of the tests, from the communication system and its components. These tests are important for verification of the single components and the combination of them. The most important tests are from the UART, the transceiver and the receiver.

Inhalt

1. Einführung	1
2. Mögliche Modulationsschemen in der Optik	2
2.1. Analoge Intensitätsmodulation	5
2.2. Digitale Basisbandmodulationstechniken	6
2.2.1 „ON-OFF Keying“	6
2.2.2 „Pulse Position Modulation“	7
2.2.3 „Pulse Interval Modulation“	11
2.3. „Subcarrier“ Intensitätsmodulation	16
2.4. „Orthogonal Frequency Division Multiplexing“	17
2.5. „Optical Polarization Shift Keying“	18
3. Aufbau des optischen Kommunikationssystems	21
3.1. Optische Quelle	21
3.2. Externer Modulator	23
3.3. FPGA	27
3.4. Optischer Empfänger	30
4. Erstellung des FPGA Programms	32
4.1. Erstellung des Frameworks	32
4.2. UART	42
4.3. Transceiver	52
4.4. Receiver	56
4.5. Transfer des Programms auf den FPGA	60
5. Ergebnisse	66
5.1. Tests und Ergebnisse der UART	66
5.2. Ergebnisse der Tests für das Programm	71
5.3. Ergebnisse des PPM Signals	75
6. Schlussfolgerung	79
Literaturverzeichnis	80

1. Einführung

Dieses Kapitel erläutert die Aufgabenstellung, die Motivation dahinter und bietet eine Übersicht über den Aufbau der Masterarbeit. Die Aufgabenstellung besteht darin, ein optisches Übertragungssystem aufzubauen, in welchem spezielle Modulationsarten implementiert werden. Dazu sollen bereits vorhandene Komponenten verwendet und fehlende Zwischenstücke konstruiert werden. Die Grundstruktur des optischen Übertragungssystems besteht im Wesentlichen aus vier Komponenten, eine optische Quelle, ein externer Modulator, ein optischer Empfänger und ein FPGA, welcher für die Erschaffung des Modulationssignales dient. Die speziellen Modulationsarten werden mit dem gleichen FPGA realisiert. Der Grund für diese Arbeit ist, dass noch kein solches System im optischen Labor existiert, dazu wurde die optische Quelle in einem früheren Projekt aufgebaut. Hinzu kommt, dass es ein sehr spannendes Thema ist, die genauen Hintergründe eines solchen Systems zu ergründen und die Theorie in der Praxis anzuwenden.

Die Arbeit selbst gliedert sich in vier große Bereiche. Kapitel 2 ist der erste Teil, welcher einen Überblick über viele verschiedene Modulationsarten, welche in der optischen Nachrichtentechnik eingesetzt werden, gibt. Darunter auch das sehr verbreitete „On-OFF-Keying“. Im Kapitel 3 werden die vier Hauptkomponenten des optischen Übertragungssystem genauer beleuchtet. Dort werden neben technischen Details auch ihr Zweck im Übertragungssystem beschrieben. Direkt im nächsten großen Kapitel, wird sehr genau auf das Programm, welches auf dem FPGA läuft, eingegangen. Neben der Grundstruktur dieses Programms werden die einzelnen Teilbereiche beleuchtet und deren Zweck im gesamten System. Das Kapitel 5 stellt die Ergebnisse dar, welche während des Aufbaus erzielt wurden. Das letzte Kapitel beinhaltet die Schlussfolgerung über die Arbeit und die Probleme die aufgetaucht sind.

2. Mögliche Modulationsschemen in der Optik

Um einen Überblick über die möglichen Modulationsschemen zu bekommen, werden in diesem Kapitel die gängigsten Arten von optischen Modulationsformen auf ihre Vor- und Nachteile untersucht. Das folgende Abbild bietet einen ersten Einblick über viele der möglichen Schemen.

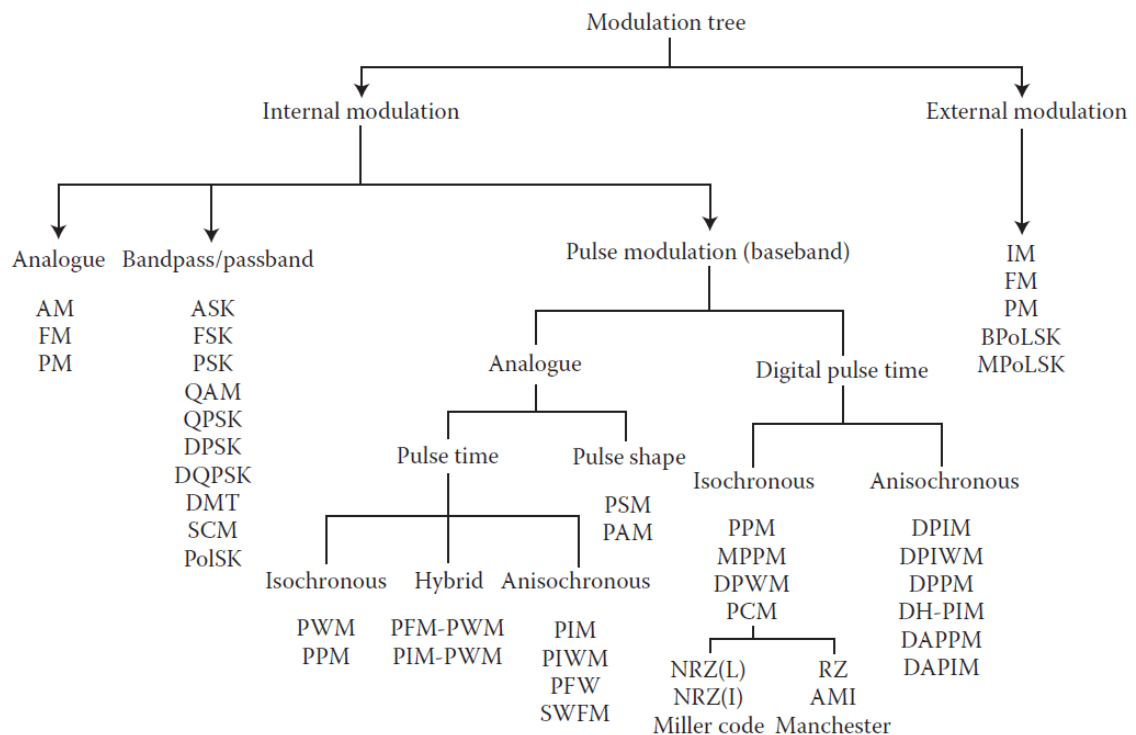


Abbildung 1: Übersicht über optische Modulationsschemen [1]

Zuerst soll ein Überblick geschaffen werden, welche Limitierungen es für optische Modulationsschemen gibt.

Energie-Effizienz: Diese Limitierung ist einer der wichtigsten Parameter für optische Systeme, denn sie gibt an, mit wie viel Energie der optische Sender betrieben werden darf. Sie muss begrenzt werden wegen den Sicherheitsbestimmungen vom Hersteller des Senders, besonders wenn das optische System eine drahtlose Komponente besitzt.

Durch diese Einschränkung kann man nicht jede Interferenz, einfach mit Erhöhung der Leistung des Senders kompensieren. Mathematisch lässt sich die Energie-Effizienz η_p ausdrücken:

$$\eta_p = \frac{E_{pulse}}{E_b} \quad \text{Formel 2.1}$$

Wobei E_{pulse} die Energie pro Impuls und E_b die durchschnittliche Energie pro Bit ist [1].

Dadurch ist die Wahl einer geeigneten Modulation ein fundamentaler Schritt beim Bau eines optischen Systems.

Bandbreiten-Effizienz: Theoretisch betrachtet, besitzt der optische Träger eine „unlimitierte Bandbreite“, jedoch wird diese durch verschiedene Eigenschaften des optischen Systems selbst limitiert. Eigenschaften wie die Kanalkapazität oder die Photodetektorfläche haben einen großen Einfluss auf die verfügbare Bandbreite eines optischen Systems mit einer verzerrungsfreien Übertragung. Auch die Mehrwegausbreitung wirkt begrenzend auf die Bandbreite und dadurch spielt die verfügbare Bandbreite eine wichtige Rolle bei der Wahl eines Modulationsschemas. Um die Bandbreiten-Effizienz eines Modulationsschemas zu bestimmen nutzt man folgende Formel:

$$\eta_B = \frac{R_b}{B} \quad \text{Formel 2.2}$$

Hierbei ist R_b die verfügbare Bitrate und B die Bandbreite des IR Transmitters.

Da die beiden bisher aufgelisteten Parameter (Energie-Effizienz und Bandbreiten-Effizienz) eng miteinander verbunden sind bei der Auswahl eines Modulationsschemas, besitzen diese auch eine Beziehung zueinander, welche sich mit der folgenden Formel ausdrücken lässt:

$$\eta_P = \frac{\eta_B}{\gamma} \quad \text{Formel 2.3}$$

Dabei ist γ der Auslastungsgrad.

Ein wichtiger Punkt ist wäre noch zu erwähnen, dass Modulationsschemen mit hohen Bandbreitenanforderungen anfälliger für „intersymbol interference“ (ISI) sind und damit auch einen höheren Energieaufwand mit sich bringen [1].

Zuverlässigkeit der Transmissionen: Dieser Punkt ist weniger ein Parameter, als eher eine unumgängliche Notwendigkeit, denn ein Modulationsschema mit guter Energie- und Bandbreiten-Effizienz bringt nicht viel, wenn die Fehlerrate bei ungünstigen Bedingungen (bei Freiraum wäre dies Nebel und im Kabel wäre dies ein Knick) in die Höhe schießt. Auch bei Schwankungen des DC-Anteils des Datensignals, bei induzierten ISI von Mehrweganteilen und beim Auftreten von Dispersion sollte das Modulationsschema eine gewisse Resistenz aufweisen. Das Modulationsschema sollte auch zu viele „high pulses“ und auch eine zu lange Zeit zwischen dem Übergang von „low“ zu „high“ vermeiden. Zusammengefasst sollte das Modulationsschema gegenüber vielen Problemen, die bei der Übertragung auftreten können, resistent sein [1]. Dies ist ein etwas schwer bestimmbarer Wert für ein Modulationsschema, denn es spielen hier viele Faktoren eine Rolle. Man sollte die Wahl des geeignetsten Modulationsschemas immer an das vorhandene optische System anpassen, um den schwerwiegendsten Störungen entgegenzuwirken.

Weitere Faktoren: Einer der wichtigsten Faktoren, der noch nicht erwähnt wurde, sind die Kosten eines Modulationsschemas. Diese sollten so gering wie möglich ausfallen. Ein Schema kann noch so viel Energie- und Bandbreiten-Effizienz besitzen, wenn es sich von den Kosten her nicht rentiert, scheidet es einfach aus. Neben den Kosten spielt auch noch die Komplexität eine Rolle. Ein leicht einzubauendes, bzw. konfigurierbares Schema wird einem komplexen Schema meistens vorgezogen, deswegen findet das simpelste Modulationsschema „ON-OFF Keying“ (OOK) sehr häufig den Weg in optische Systeme. Ein weiterer Faktor, vor allem bei Freiraumübertragung in geschlossenen Räumen, ist die Fähigkeit, die Interferenz vom künstlichen Umgebungslicht nicht anzunehmen [1].

Diese Punkte bilden die wichtigsten Kriterien bei der Auswahl eines geeigneten Modulationsverfahrens für ein optisches Kommunikationssystem. Die nachfolgenden Unterkapitel befassen sich mit den wichtigsten Modulationsverfahren für optische Systeme, diese standen auch zur Auswahl für mein optisches System.

Bei einem „intensity modulation / direct detection“ IM/DD System handelt es sich einfach wie der Name schon sagt um einen Überbegriff für Modulationsverfahren, welche die Intensität des Lichts modulieren und Sender und Empfänger eine „Line of site“ LOS zueinander haben, bedeutet, die beiden sehen sich ohne Objekte dazwischen.

2.1. Analoge Intensitätsmodulation

Grundlegend gibt es zwei Möglichkeiten für analoge Intensitätsmodulationen (AIM). Die Erste ist die IM/DD und ist zugleich die einfachere Art der AIM, hierbei wird die Intensität der Lichtquelle, mittels eines analogen Signals (oder einem vormodulierten „radio frequency“ RF Signal), moduliert und mit einem Detektions-basiertem Photodetektor direkt empfangen. Diese Methode besitzt nur einen großen Nachteil, Sie ist nicht für hohe Frequenzen zu gebrauchen, wegen der limitierten Bandbreite der Lichtquelle und ihren nichtlinearen Charakteristiken. In Abbildung 2 kann man den schematischen Aufbau eines solchen Systems sehen.

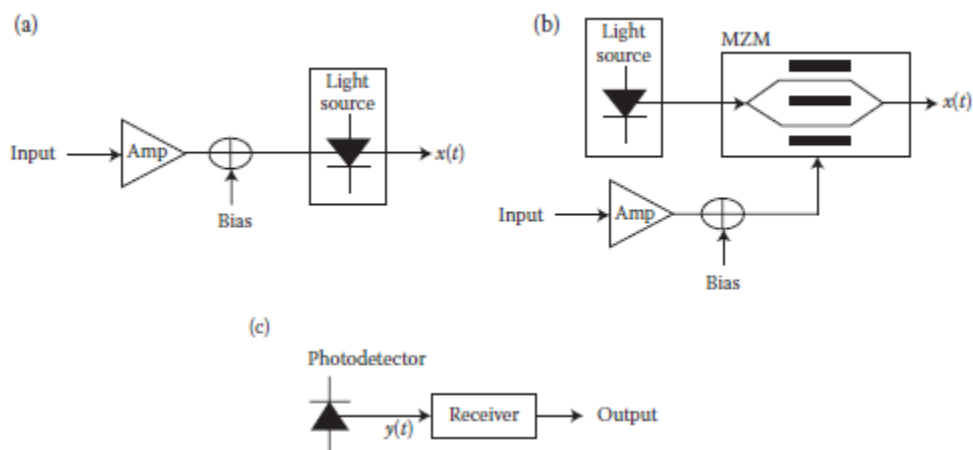


Abbildung 2: Schematischer Aufbau der zwei möglichen AIM Systeme a) Intensitätsmodulation b) externe Modulation und c) Empfänger [1]

Die zweite Möglichkeit eines Systems mit AIM ist die externe Modulation, bei dieser Methode wird, wie in Abbildung 2 Punkt b zu erkennen ist, ein externer Modulator (wie z.B. Mach-Zehnder Modulatoren [MZM]) verwendet, um damit die Intensität des Laserstrahls zu modulieren. Damit dies empfangen werden kann, wird ein Kohärenter Empfänger eingesetzt. In diesen Systemen kann man viele Modulationsarten anwenden, wie die Amplitudenmodulation (AM), „Quadrature amplitude modulation“ (QAM), „pulse amplitude modulation“ (PAM) und auch „subcarrier multiplexing“ (SCM) kann implementiert werden [1].

Man erreicht zwar mit der externen Modulation höhere Frequenzen, muss dafür aber viele Probleme lösen, wie das nichtlineare Verhalten der Lichtquelle. Auch Einbußen der Energie-Effizienz sind hinzunehmen, besonders die erhöhten Kosten stellen eines der Hauptprobleme dar. Falls man jedoch keine hohen Frequenzen erreichen muss, ist die AIM eine gute Wahl, da diese viele Vorteile gegenüber digitalen Modulationsschemen bietet, wie geringe Kosten und eine einfache Implementierung [1].

2.2. Digitale Basisbandmodulationstechniken

Dieses Unterkapitel befasst sich mit den verschiedenen Basisbandmodulationstechniken in der Optik, darunter die bekanntesten wie OOK, „pulse position modulation“ (PPM) und den Abwandlungsmöglichkeiten beider Modulationen. Doch zuerst eine kurze Beschreibung wie digitale Basisbandmodulationstechniken definiert sind. Wie der Name schon sagt, werden die Daten vor der Intensitätsmodulation, der optischen Quelle, nicht auf eine höhere Trägerfrequenz gemischt, damit bleibt ein Anteil der Energie im „direct current“ (DC) Bereich. Das Spektrum der modulierten Daten befindet sich demnach in der Nähe des DC Bereiches. Doch die Basisbandschemen haben den Vorteil, dass sie toleranter gegenüber den Effekten von Mehrwegausbreitungen sind [1].

2.2.1 „ON-OFF Keying“

Bei OOK gibt es grundsätzlich 2 Methoden „no return to zero“ (NRZ) oder „return to zero“ (RZ), wobei NRZ die geläufigere ist, welche wir uns zuerst anschauen [1].

OOK-NRZ ist wohl das meist genutzte Modulationsschema für digitale Übertragungen in der Optik, wegen seiner Einfachheit. Der Sender sendet dabei einen Rechteckimpuls aus, der die Länge von $\frac{1}{R_b}$ ist, wobei R_b ist hierbei die Bitrate, und die Intensität $2P$, P beschreibt die Übertragungsenergie, hat, um eine „1“ zu signalisieren. Für eine „0“ wird einfach kein Impuls ausgesendet. Die benötigte Bandbreite beträgt $B = R_b = \frac{1}{T}$ (T ist die Periodendauer, welche der Pulsbreite entspricht), die Inverse der Pulsbreite [2].

OOK-RZ unterscheidet sich zu OOK-NRZ nur bei der Übertragung einer „1“, denn hier bleibt der Impuls nicht die volle Periodendauer auf „high“. Bei einem $\gamma = 0,5$ ist die Pulsbreite genau die halbe Periodendauer. Diesen Unterschied kann man sehr gut in Abbildung 3 erkennen. Daraus ergibt sich der Vorteil, das OOK-RZ weniger Energie benötigt als OOK-NRZ für die gleiche Fehlerperformance, dafür aber eine höhere Bandbreitenanforderung besitzt. Bei $\gamma = 0,5$ wird bereits die doppelte Bandbreite benötigt und bei $\gamma = 0,25$ schon die vierfache [1]. Damit hat man bei diesen beiden Varianten die Wahl zwischen besserer Energie-Effizienz oder besserer Bandbreiten-Effizienz.

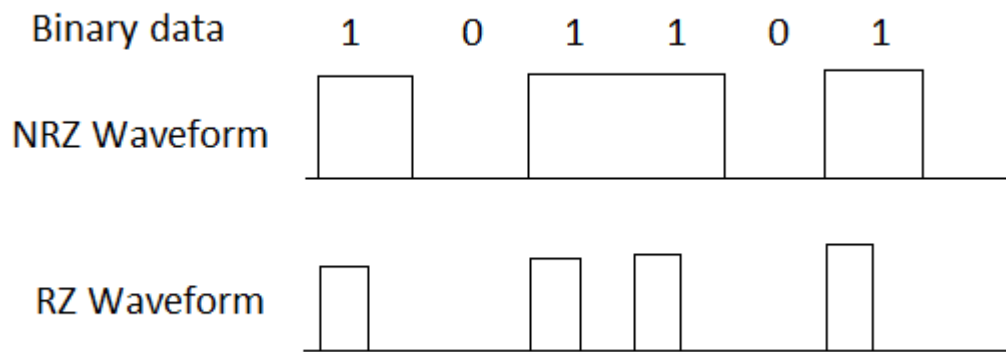


Abbildung 3: Signalform für OOK-NRZ und OOK-RZ bei $\gamma = 0,5$ [13]

2.2.2 „Pulse Position Modulation“

Wenn man von „Pulse Position Modulation“ (PPM) spricht, ist meistens die „Single Pulse Position Modulation“ (L-PPM) Variante damit gemeint. Denn PPM umfasst viele verschiedene Verfahren, darunter „Multi Pulse Position Modulation“ (MPPM) oder auch „Differential Pulse Position Modulation“ (DPPM). Doch alle diese Formen bieten dieselben Vorteile, vor allem die hohe Energie-Effizienz, Übertragungs-Effizienz und die gute Sicherheit gegenüber Störungen machen PPM zu einer beliebten und weit verbreiteten Modulation in der optischen Nachrichtentechnik. Verglichen mit OOK hat PPM eine geringe durchschnittliche Energie und eine hohe Spitzenenergie, dies führt zu einer guten Sicherheit und einem hohen „signal to noise ratio“ (SNR) [3]. Im Weiteren werden nun einige ausgewählte Varianten von PPM genauer betrachtet.

L-PPM: Die meist genutzte Ausführung von PPM besitzt eine feste Größe von L Zeiteinheiten, eine Zeiteinheit wird „chip“ genannt. Wobei alle chips „0“ sind, bis auf einen der „1“ ist, wie in Abbildung 4 zu sehen ist. Damit ergeben sich L verschiedene Symbole. Jedes Symbol entspricht nun einem Bitmuster und ergibt damit das „mapping“ für die Übertragung. Da nun pro Symbol immer nur eine „1“ ist, ergibt sich eine extrem gute Energie-Effizienz, aber man benötigt eine hohe Bandbreite im Gegenzug dafür. Die benötigte Bandbreite lässt sich leicht bestimmen und entspricht der Anzahl der chips. Um den Vorteil der Energie-Effizienz zu behalten und dabei die Bandbreitenanforderung zu verringern, entstanden viele Varianten [3].

DPPM: Diese, verbessert vor allem die Bandbreiten-Effizienz, da sie wie PPM eine „1“ und sonst „0“ besitzt, aber alle „0“ nach der „1“ wegschneidet und sofort mit dem nächsten Symbol fortfährt. In Abbildung 4 ist dies verdeutlicht und man erkennt ebenfalls, dass der Durchsatz sich enorm erhöht. Ein Nachteil dieser Modulation ist, dass ein Fehler sich auch auf das nachfolgende Symbol auswirkt. Es

gibt dabei zwei Arten von Fehlern die auftreten können. Der Erste ist, dass eine „0“ als eine „1“ detektiert wird und somit ein zusätzliches Symbol erkannt wird und das eigentliche Symbol als ein anderes Symbol detektiert wird. Der Zweite ist, dass eine „1“ als „0“ erkannt wird und somit ein Symbol nicht erkannt wird und das Folgende falsch detektiert wird. Natürlich kann auch eine Kombination dieser beiden Fehlertypen auftreten. Doch das größte Problem dabei ist, dass diese Fehler nur erkannt werden können, wenn eine Folge von „0“ größer als $(L-1)$ chips ist. Im Vergleich zu L-PPM besitzt DPPM, zwar eine etwas höhere Energie Anforderung, dafür aber eine viel geringere Bandbreitenanforderung. L-PPM ist des Weiteren bei Fehlern besser, da hier Fehler immer nur das betreffende Symbol verfälschen, auch die Fehlererkennung ist besser, denn wenn ein Symbol mehr als eine oder keine „1“ besitzt, wird ein Fehler erkannt [1].

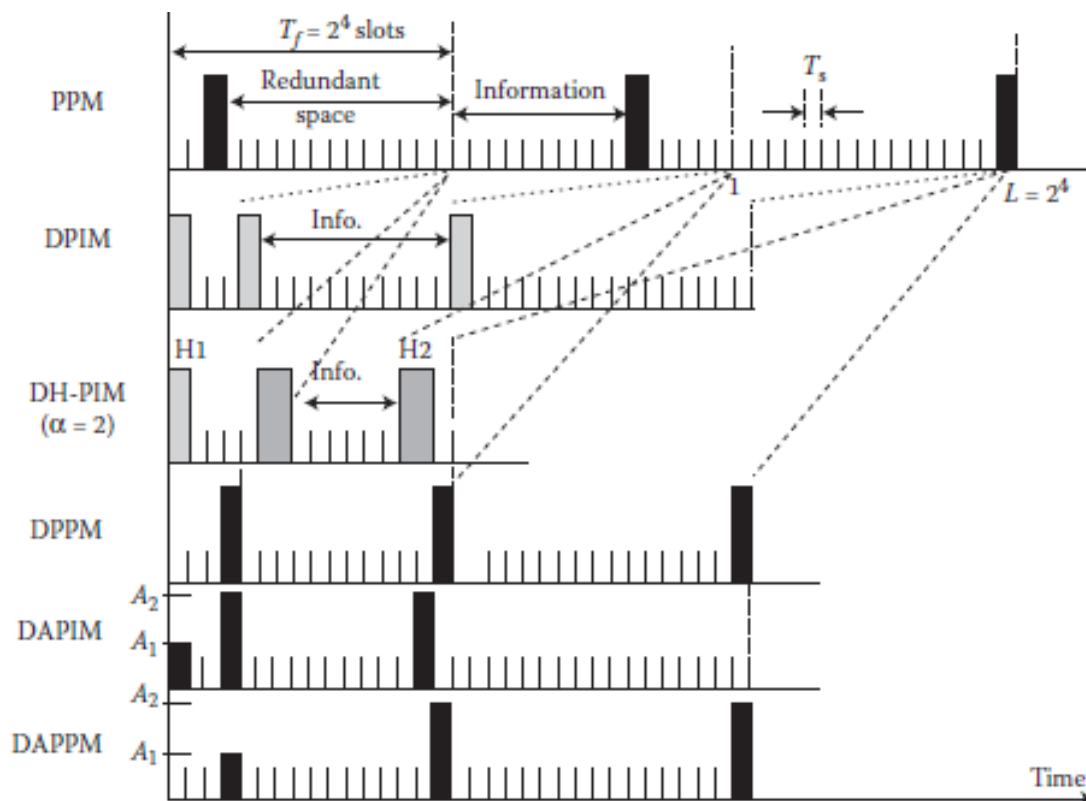


Abbildung 4: Signalform für PPM, DPIM, DH-PIM, DPPM, DAPIM und DAPPM [1]

MPPM: Bei MPPM besteht das Symbol wieder aus einer fixen Größe von chips, nur treten hier mehr „1“ Impulse pro Symbol auf. Die Abbildung 5 zeigt wie MPPM aussieht, im speziellen 2-MPPM (2-MPPM hat zwei „1“ pro Symbol) [1].

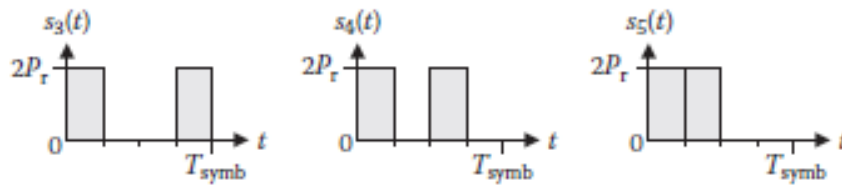


Abbildung 5: gültige Signalformen für 2-MPPM [1]

Diese Modulationsvariante von PPM bringt den großen Vorteil der sehr guten Bandbreiten-Effizienz, dafür steigt im Gegenzug der Energiebedarf.

Beim Einsatz von MPPM werden meistens nicht alle möglichen Signalformen verwendet. Dies hat mehrere Gründe, sie werden nicht gebraucht oder man will eine geringere Fehlerrate erzielen. Dies wird erreicht indem man die Symbole so auswählt, dass eine große minimale Hamming-Distanz erzielt wird [1].

Für MPPM gibt es wiederum mehr Ausführungen. Die Einfachste ist jene bei der alle „1“ Impulse die gleichen Parameter besitzen. Doch durch verändern der Parameter lässt sich die Bandbreiten-Effizienz weiter steigern, so kann man die Höhe oder Breite der „1“ Impulse verändern, soweit das jeder „1“ Impuls pro Symbol seine eigenen Charakteristiken besitzt. Der Aufwand jedoch für eine solche Modulation ist sehr hoch und komplex und deswegen kommt sie nur sehr selten zur Anwendung [3].

„Differential Amplitude Pulse Position Modulation“ (DAPPM): Bei dieser Abweichung ändert sich wie der Name schon sagt die Amplitude der „1“ Pulse. Vom Grundprinzip her funktioniert DAPPM gleich wie DPPM nur mit dem Unterschied, dass hier die Höhe der „1“ Impulse verändert wird, siehe Abbildung 4. Damit werden gleich zwei Kernparameter verbessert, man erhöht die Bandbreiten-Effizienz und verringert gleichzeitig die benötigte Energie. Ein Nachteil der dadurch entsteht, ist die erhöhte Komplexität des Modulationsverfahrens [1].

„PPM-Based Four-Dimensional Modulation“: Die Idee hinter dieser Modulation, ist es in einem L-PPM Signal die „1“ mit den verschiedenen Farben zu kombinieren, im Grunde handelt es sich dabei um L-PPM mit „quadratic color shift keying“ (QCSK). Es ist darauf zu achten das diese Art der Modulation nur mit sichtbarem Licht, welches man in die einzelnen Farben aufspalten kann, funktioniert. Dabei wird mittels einer „red green blue amber“ RGBA-LED, welche wegen ihrer hohen Modulationsbandbreite gerne eingesetzt wird, das L-PPM Signal übertragen. In der Abbildung 6 sieht man wie Sender und Empfänger für den Einsatz der RGBA-LED aussehen. Die Daten kommen hier

bereits als PPM Signal in den 4-D Modulator, welches mittels der LEDs übertragen wird. Zum Empfangen dienen dann optische Bandpass-Filter [4].

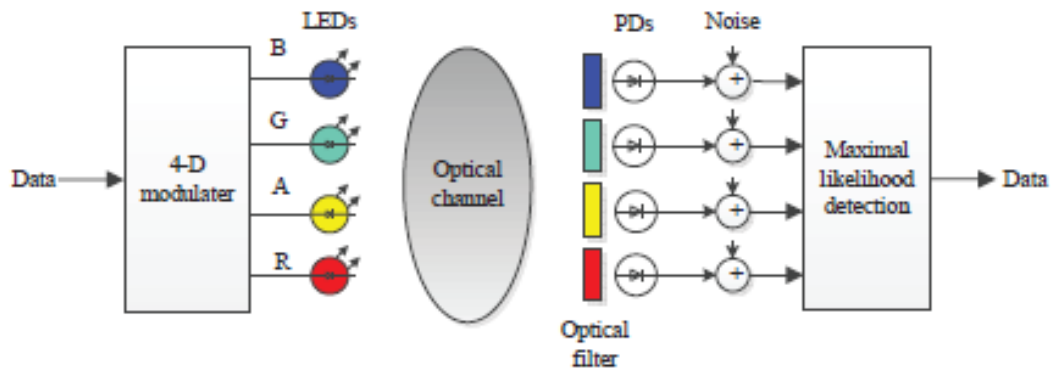


Abbildung 6: Blockdiagramm eines RGB-A-LED sichtbare Lichtkommunikation Systems [4]

Die Modulation selbst sieht man in Abbildung 7, man erkennt dort, dass immer zwei Farben bei einem „1“ Impuls aktiv sind, dies hat den Grund den „cross-talk“ der Farben untereinander zu reduzieren. Der Vorteil einer solchen Modulation besteht darin, dass die Bandbreiten-Effizienz, im Vergleich zu L-PPM deutlich erhöht wird. Auch die Energie-Effizienz steigt minimal, wenn man „PPM-Based Four-Dimensional Modulation“ mit L-PPM im Spektrum des sichtbaren Lichts vergleicht. Nachteile dabei wären, die höhere Komplexität, aber auch die LED kann Probleme bereiten, wenn sie außerhalb ihrer linearen Grenzen betrieben wird [4].

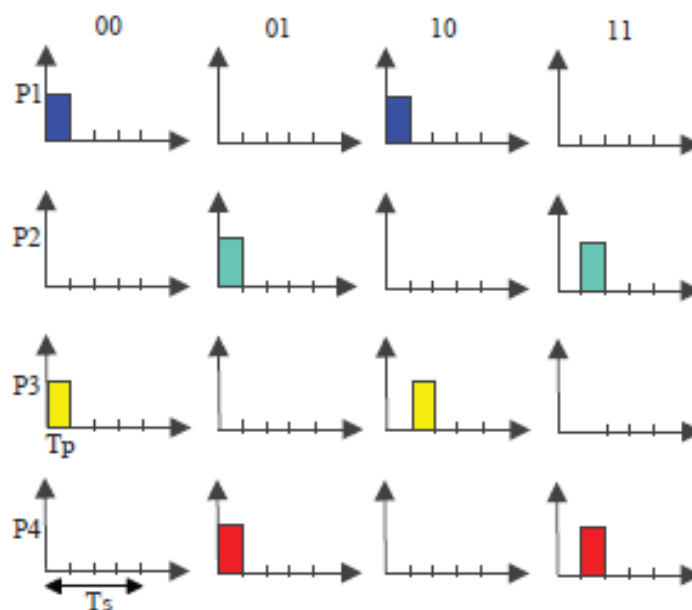


Abbildung 7: 4-Stufen PPM-basierte 4-D Modulation [4]

2.2.3 „Pulse Interval Modulation“

Die Gruppe von Modulationen unterscheidet sich im Grunde von den PPM Arten nur darin, dass hier nicht die Position des Impulses aussagekräftig ist, sondern das Intervall zwischen zwei Impulsen. „Pulse Interval Modulation“ (PIM) reduziert die Komplexität von L-PPM in folgender Weise, dass PIM besitzt eine eingebaute Symbolsynchronisierung. Die einfachste Implementierung von PIM ist „Digital Pulse Interval Modulation“ DPIM, welche den Vorteil gegenüber L-PPM bietet, dass die Überschüssigen „0“ entfernt werden [1]. Der Unterschied von DPIM zu PIM besteht darin, dass die Zeiteinheiten diskret sind. Es gibt zwei mögliche Ausführungen von DPIM, einmal ohne „guardslot“ und einmal mit. Der „guardslot“ wird dazu benötigt, dient der Vermeidung, dass zwei „1“ Impulse direkt aufeinander folgen [2]. Im Weiteren betrachten wir nun diese beiden Varianten zusammen mit weiteren Möglichkeiten für PIM Techniken.

DPIM ohne „guardslot“ GS: Wie oben schon erwähnt ist dies die einfachste Art einer PIM. Hierbei signalisiert ein „1“ Impuls den Beginn eines Symbols, dieser Impuls besitzt eine Breite eines chips. Nach diesem Impuls kommt nun eine Folge von „0“ chips, die Anzahl dieser „0“ chips gibt das jeweilige Symbol an, zu sehen in Abbildung 4. Im Grunde ist DPIM eine Invertierung von DPPM und damit können auch die gleichen Fehlerfälle eintreffen. Der Vorteil von DPIM zu PPM ist ebenfalls eine Verbesserung der Bandbreiten-Effizienz und der Datenrate [1].

DPIM mit GS: Der feine Unterschied besteht darin, dass nach einem „1“ Impuls immer ein „0“ chip folgt (siehe Abbildung 8), damit lässt sich verhindern, dass zwei „1“ Impulse nacheinander auftreten. Eine Möglichkeit wäre diesen Fall beim „symbolmapping“ auszuschließen. Doch damit würde es nicht vollkommen vermieden werden, denn es könnte der erste chip nach dem „1“ Impuls fehlerhaft detektiert werden und um solch einen Fall vorzubeugen wird ein GS eingesetzt. Der GS ist immer ein „0“ chip, daher wird beim Empfänger immer nach einem „1“ Impuls der darauffolgende chip direkt auf „0“ gesetzt und deswegen gar nicht weiter betrachtet.




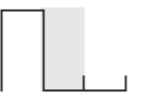
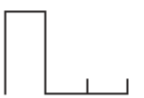
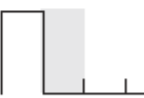


Source data	4-DPIM symbols	
	NGB	1GS
00		
01		
10		
11		

Abbildung 8: 4-DPIM ohne GS und mit einem GS [1]

Dies kann natürlich zu neuen Fehlerarten führen, wovon viele das darauffolgende Symbol ebenfalls verfälschen. Die Arten wurden bereits bei DPPM genauer erläutert, die einzige Art von Fehler, die mit einem GS entstehen kann, wäre, dass genau der letzte „0“ chip vor dem nächsten „1“ Impuls falsch detektiert wird. Somit hat der darauffolgende chip, welcher den „1“ Impuls enthält, als GS direkt auf „0“ gesetzt wird. Dies wäre das schlimmste Szenario, denn es würde das nächste Symbol sofort verfälschen.

Wenn man nun DPIM ohne GS und DPIM mit GS vergleicht, in Hinsicht auf Energie-Effizienz, liegt DPIM mit GS leicht vorne, verbraucht mehr Bandbreite, wobei DPIM ohne GS eine leicht besser Bandbreiteneffizienz besitzt [1].

Bevor nun die restlichen Varianten von PIM erläutert werden, vergleichen wir OOK, L-PPM und DPIM auf ihre Energie- und Bandbreiten-Effizienz. In Abbildung 9 ist die Gegenüberstellung der Energie-Effizienz der drei Modulationsverfahren. OOK dient hierbei als Referenz. Auf der X-Achse sind die Anzahl der chips aufgetragen, welche hier als M benannt wurden, und auf der Y-Achse die benötigte Übertragungsenergie in dB. Man erkennt ganz klar, dass sowohl L-PPM und DPIM eine deutlich bessere Energie-Effizienz, als OOK, bei höheren M besitzen. Daraus folgt, dass je mehr chips man nutzt, die Energie-Effizienz damit verbessert wird. Doch man kann aus Abbildung 9 auch ablesen, das L-PPM eine bessere Energie-Effizienz besitzt als DPIM [2].

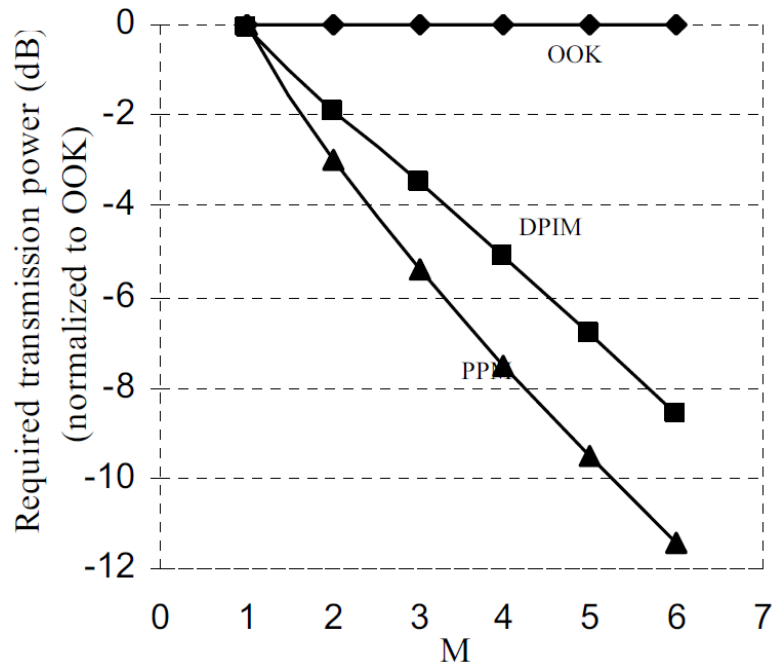


Abbildung 9: Energie-Effizienz Vergleich von L-PPM, DPIM und OOK [2]

Um eine bessere Aussage der Drei treffen zu können ist in Abbildung 10 ein Vergleich der Bandbreiten-Effizienz dargestellt. Ganz klar gewinnt OOK hier, aber OOK benötigt eine Symbolsynchronisierung, das die anderen beiden nicht benötigen. L-PPM ist das Bandbreitenineffizienteste dieser drei Modulationsarten, man erkennt, dass mit steigendem M die benötigte Bandbreite sehr schnell wächst. Bei DPIM ist dieser Anstieg deutlich geringer, da DPIM keine fixe Datenrate besitzt, hierfür wurde die durchschnittliche Bitrate basierend auf der durchschnittlichen Symbolrate benutzt [2].

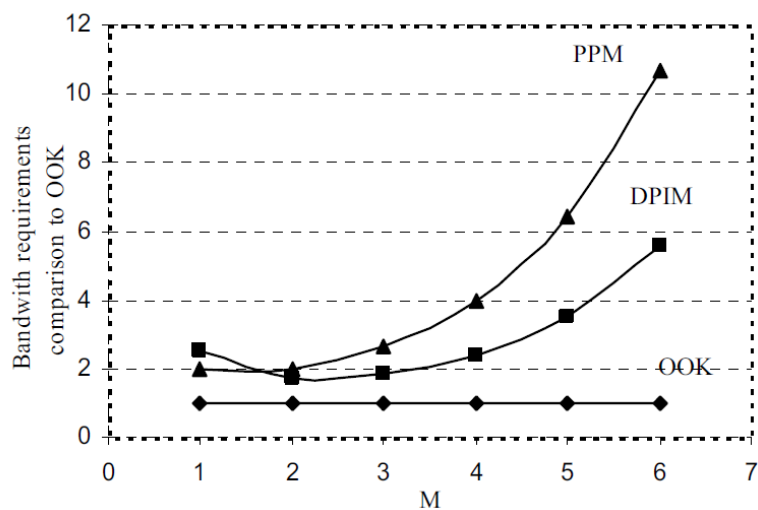


Abbildung 10: Bandbreiten-Effizienz Vergleich von L-PPM, DPIM und OOK [2]

Um ein Fazit ziehen zu können, benötigen wir noch einen Vergleich der Fehlerwahrscheinlichkeit der Drei, welche in Abbildung 11 zu sehen sind. Es wurde auf der Y-Achse die Paketfehlerrate aufgetragen, da bei einem Fehler in DPIM immer zwei Symbole verfälscht werden. Man erkennt klar, das L-PPM zwar mehr Bandbreite benötigt, aber eine geringere Fehlerwahrscheinlichkeit und ebenfalls die stärkste Energie-Effizienz besitzt. Doch DPIM kann in Punkto Bandbreite gegenüber L-PPM punkten, doch man erkennt den klaren Trend, dass die Fehlerwahrscheinlichkeit mit höherem M einen größeren Abstand zu L-PPM annimmt. Während OOK zwar in Bandbreiten-Effizienz nicht zu schlagen ist, schneidet diese Modulationsart bei Energie-Effizienz und Fehlerwahrscheinlichkeit am schlechtesten ab. [2].

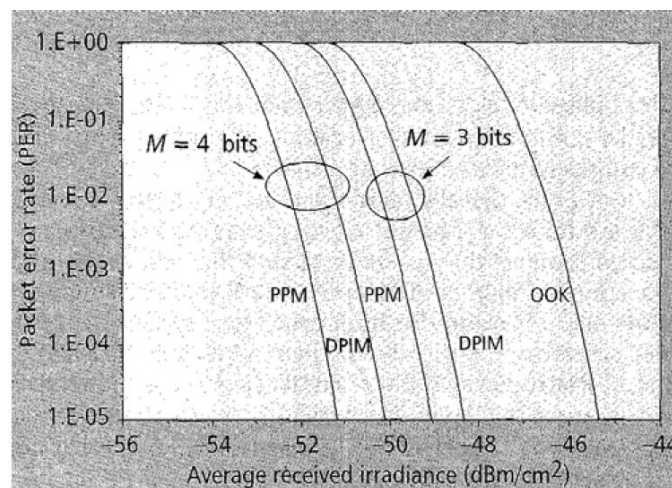


Abbildung 11: Paketfehlerrate für L-PPM, DPIM und OOK gegenüber der durchschnittlichen empfangenen Bestrahlungsstärke [2]

Damit sieht man, dass es nicht das beste Modulationsverfahren gibt, sondern man immer von Anwendung zu Anwendung abwägen muss, welches die beste Performance bietet.

„Dual-Header“ PIM (DH-PIM): Diese Modulationsart benutzt zwei „headers“, H_1 und H_2 , für die Übertragung. Dabei besitzt H_1 eine Breite von $0.5\alpha T_s$ und H_2 αT_s , T_s steht für die Dauer eines chips und α ist eine positive Ganzzahl größer 0. Beide „headers“ besitzen einen „guard space“ um sicherzustellen, dass nach einem „header“ keine „1“ auftreten kann. Bei DH-PIM wird im Grunde wie bei DPIM mit GS, die Information durch die Anzahl der leeren chips gekennzeichnet, in Abbildung 4 sieht man wie Signalform DH-PIM aussieht. Um eine genauere Funktionsweise zu verstehen ist in Abbildung 12 ein detaillierter Aufbau zu sehen. Man sieht deutlich die unterschiedliche Breite von H_1 und H_2 sowie den

„guard space“ dahinter. Die Gesamtlänge von „header“ und „guard space“ ist dabei immer dieselbe. Dahinter sind nun die Information chips, welche wie bei DPIM alle „0“ sind, bis der nächste „header“ kommt. Welcher „header“ nun genutzt wird hängt vom „most significant bit“ (MSB) des binären Eingangswortes ab, ist dies 0 wird H1, bei 1 wird H2 benutzt. Wenn H1 genutzt wird entspricht d_n dem Dezimalwert des binären Eingangswortes, bei H2 entspricht d_n dem Dezimalwert des Einserkomplementes des binären Eingangswortes [5].

Durch diese Modulationsart lassen sich nicht nur die redundanten chips die nach einem L-PPM Impulses folgen, eliminieren, sondern auch die durchschnittliche Framedauer verkürzen und zwar zirka auf die Hälfte von DPIM und ein Viertel von PPM [5]. Dadurch erhöht sich die Datenrate enorm und auch die Bandbreiten-Effizienz [1].

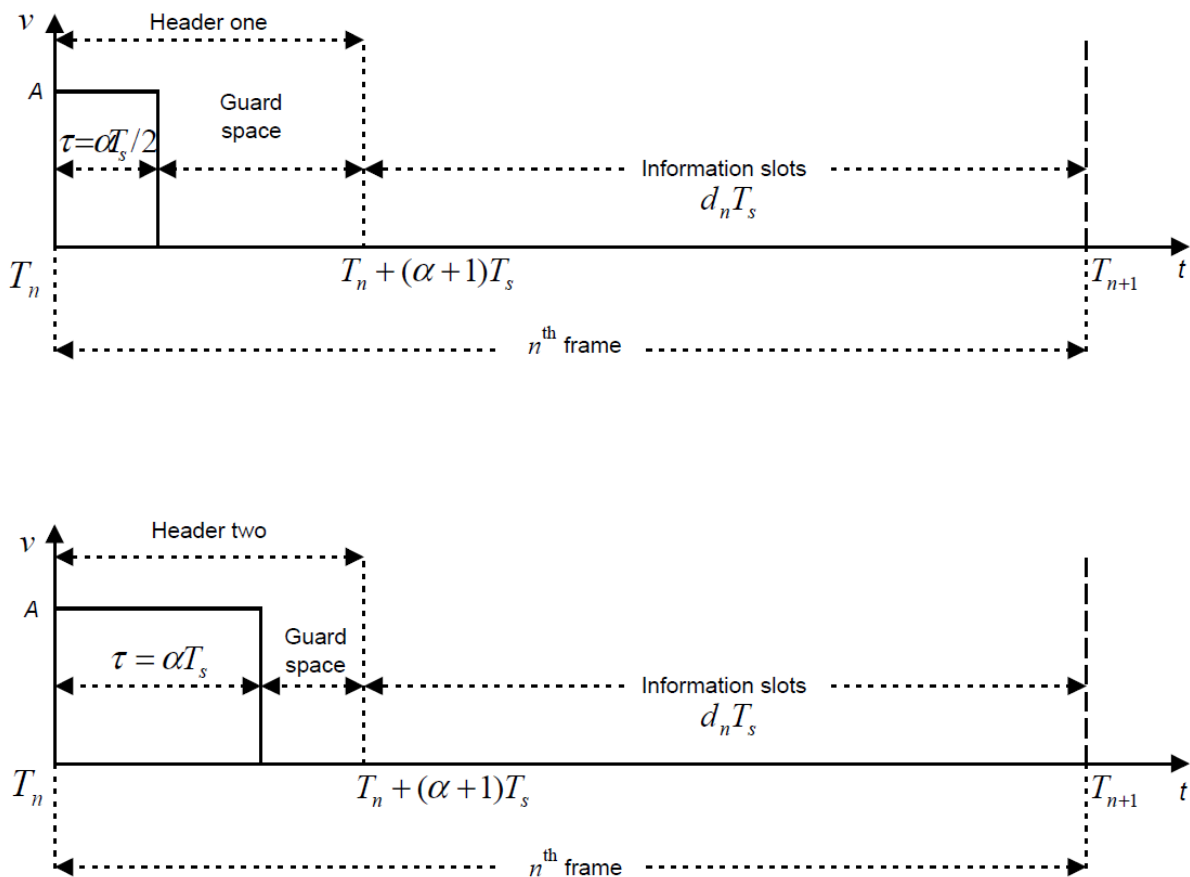


Abbildung 12: Aufbau des DH-PIM Signals, H₁ (oben) und H₂ (unten)

„Multilevel“ DPIM (MDPIM): Ähnlich wie DH-PIM, wird hier der Impuls verändert, der den Anfang eines Frames definiert. Nur wird bei MDPIM (oder auch „dual amplitude“ PIM [DAPIM] genannt) nicht die Breite des Impulses verändert, sondern die Amplitude, auch ein GS ist nach dem Impuls vorhanden.

Genau wie bei DH-PIM gibt das MSB wieder Auskunft darüber welche Amplitude verwendet wird, dabei steht MSB = 0 für die Höhe A und für MSB = 1 die Höhe 2A. Die Anzahl der leeren chips die folgen wird gleich wie bei DH-PIM gelöst, Höhe A entspricht d_n dem Dezimalwert des binären Eingangswortes und bei Höhe 2A dem Dezimalwert des Einserkomplementes des binären Eingangswortes. Zu sehen ist die Signalform in Abbildung 4, bezeichnet mit DAPIM bezeichnet. Die Vorteile sind dieselben wie bei DH-PIM, bessere Bandbreiten-Effizienz und höhere Datenrate [1].

2.3. „Subcarrier“ Intensitätsmodulation

Die „subcarrier“ (SC) Modulationstechnik ist eine alte, leichte und kosteneffektive Möglichkeit um die Bandbreite in analogen optischen Kommunikationssystemen auszunutzen. Es werden viele analoge Basisband und digitale Signale hinaufgemischt in Hinsicht auf die Intensität, Frequenz oder Phasenmodulation des optischen Trägers. Im einfachsten Fall (IM/DD) erhält man dadurch im optischen Spektrum den originalen optischen Träger ω_0 , und zwei Seitenbänder $\omega_0 \pm \omega_{sc}$, wobei ω_{sc} die SC Frequenz ist. Der größte Nachteil ist die schlechte durchschnittliche optische Energie-Effizienz, diese entsteht da sich ein Seitenband im negativen Bereich befindet und mittels eines DC-Offsets kompensiert wird. Das Signale $x(t)$ soll nicht negativ sein. Wenn man die Anzahl der SC erhöht, verringert sich damit auch die Energie-Effizienz, da für jeden SC ein DC-Offset notwendig ist.

Diese Art der Modulation wurde für optische Freiraumübertragung übernommen, weil die induzierte ISI, die von der Mehrwegausbreitung kommt, reduziert wird und eine Immunität gegenüber den Störungen von fluoresziertem Licht im DC Bereich des Spektrums besitzt. Da die optische Energie begrenzt ist (Augensicherheit), wurde zur Verbesserung der Energie-Effizienz eine zweite Stufe der Modulation hinzugefügt. Beispiele dafür sind „binary phase shift keying“ (BPSK) und „quadrature phase shift keying“ (QPSK). Es gibt auch Modulationstechniken für „frequency division multiplexing“ um mehreren Benutzern zu erlauben zeitgleich zu kommunizieren. So ein System kann man in Abbildung 13 sehen [1].

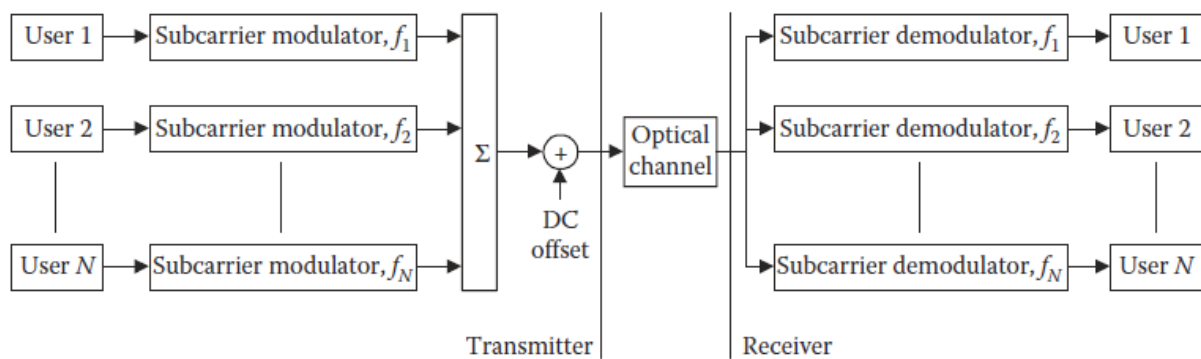


Abbildung 13: Basisimplementierung für mehrfache SC Modulation [1]

Jeder Benutzer moduliert seine Daten mit einem eigenen SC und die Summe aus allen wird dann genutzt um die optische Quelle zu modulieren. Die Demodulatoren besitzen jeder einen eigenen Bandpassfilter, um die richtigen Daten herauszufiltern. Durch dieses Verfahren wird eine hohe Bandbreiten-Effizienz und hohe Bitraten erzielt. Jedoch ist die schlechte Energie-Effizienz ein Nachteil. Eine Möglichkeit die Energie-Effizienz zu erhöhen wäre, den DC-Offset, der die negativen Bänder in den positiven Bereich schiebt, zu entfernen und damit „clipping“ zu erlauben, dadurch kommt es aber zu nichtlinearen Verzerrungen [1].

2.4. „Orthogonal Frequency Division Multiplexing“

Im Grunde ist „orthogonal frequency division multiplexing“ (OFDM) eine spezielle Art der SC Modulation, in der alle SC orthogonal zueinander sind. Für eine bessere Übersicht, wie ein solches orthogonales Signal aussieht, bevor es die optische Quelle moduliert, sieht man in Abbildung 14. Dabei ist zwischen jedem SC ein Sicherheitsintervall um ISI und Störungen durch den Kanal zu vermeiden. Die Vorteile von OFDM sind eine gute Bandbreiten-Effizienz und eine gute Störungssicherheit gegenüber anderen Verfahren [1].

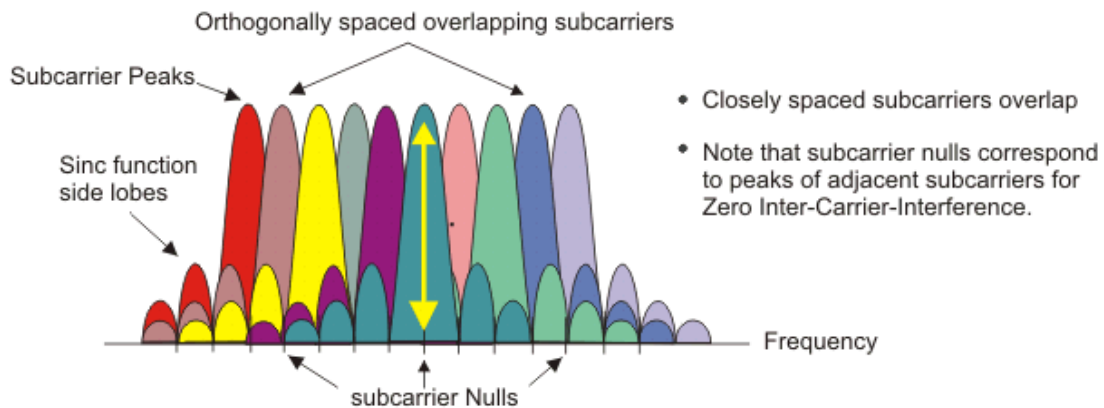


Abbildung 14: OFDM Spektrum [14]

Das Spektrum ist bei OFDM auch schmaler, da durch die Orthogonalität der SC zueinander, diese eng aneinander platzierbar sind. Doch der große Nachteil ist auch hier die schlechte Energie-Effizienz, da wie bei den anderen SC Modulationen ein DC-Offset, für die negativen Anteile, nötig ist. Um dies zu verbessern gibt es die Möglichkeit des asymmetrischen „clippings“, hier wird kein DC-Offset verwendet. Dadurch besitzt das OFDM Signal nur mehr die halbe Bandbreiten-Effizienz [1].

2.5. „Optical Polarization Shift Keying“

Wie der Name schon vermuten lässt handelt es sich bei „polarization shift keying“ (PoSK) um ein Modulationsverfahren, welches die Information in der Polarisation versteckt. In der optischen Nachrichtentechnik ist dieses Verfahren vor allem für Freiraumübertragungen vorteilhaft. Denn die Polarisation ist ein sehr stabiler Parameter, wenn man es mit der Phase oder der Amplitude vergleicht. Im Folgenden werden wir drei verschiedene Arten genauer betrachten [1].

„Binary“ (B)PoSK: In Abbildung 15 ist das Blockdiagramm des Senders zu erkennen, dabei wechselt der Modulator immer zwischen den beiden orthogonalen „state of polarization“ (SOP)s hin und her und verändert damit die Polarisation der optischen Quelle.

Dieser Modulator basiert auf den „lithium niobate“ (LiNbO_3) Mach-Zehnder interferometrie (MZI) Modulator, welcher auf einer Wellenlänge von 1550nm arbeitet [1].

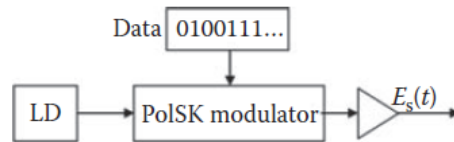


Abbildung 15: Blockdiagramm des PolSk Sender [1]

„Multilevel“ (M)PolSK: Wie bei BPolSK wird die Polarisation moduliert, nur wird ebenfalls die Amplitude und die Phase moduliert. In Abbildung 16 sieht man das Blockdiagramm eines Senders für MPolSK. Die Leuchtdiode (LD) ist linear polarisiert mit einem Winkel von $\pi/4$, mit Bezug auf die Senderreferenzachse. Dieser Strahl wird in einen Polarisationsstrahlsplitter (PBS) geführt. Danach ist \hat{x} horizontal und \hat{y} vertikal polarisiert mit gleicher Amplitude und 0° Phasendifferenz. \hat{y} wird nun um $\pi/2$ Phasenverschoben, anschließend werden beide orthogonal polarisierten Komponenten synchron Amplituden- und Phasenmoduliert bevor sie mittels des Polarisationsstrahlkombinierer (PBC), wieder zusammengefügt werden [1].

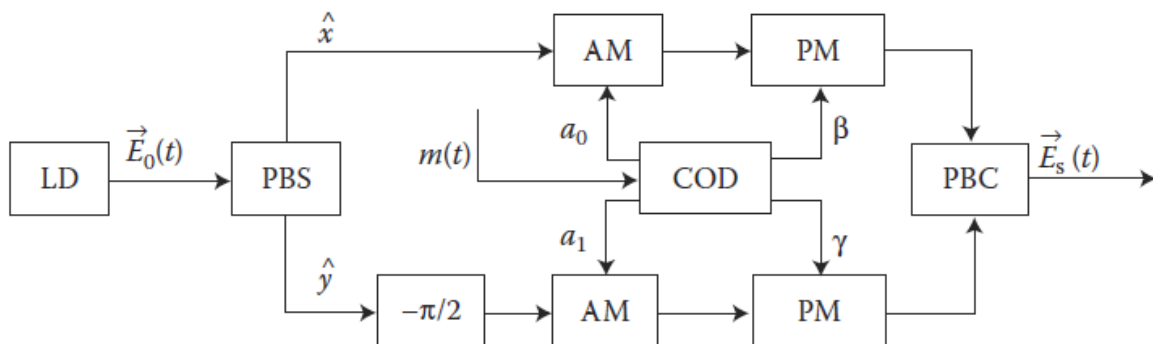


Abbildung 16: Blockdiagramm eines MPolSK Senders für ein kohärentes optisches Kommunikationssystem [1]

„Differential Circle Polarization Shift Keying“ (DCPolSK): PolSK Modulationen sind im speziellen sehr attraktiv für Systeme die eine Spitzenleistungslimitierung besitzen, da sie ein konstantes umhüllendes Modulationsschema sind und gegenüber dem Phasenrauschen des Lasers fast immun sind. Im Vergleich zu linearen PolSK Verfahren, wo die Polarisationskoordinate vom Sender genau auf den Empfänger ausgerichtet sein muss, ist dies bei „circular“ PolSk (CPolSK) nicht der Fall, denn bei zirkularer Polarisation hat man zwei Rotationszustände, dies bietet eine gute Flexibilität für optische Freiraumsysteme, im speziellen für sich bewegende Objekte. Diese Modulationsart besitzt sogar einen

Energiegewinn von zirka 3 dB gegenüber OOK. Für den Überblick ist in Abbildung 17 der DCPoISK kohärente Transmitter zu sehen [1].

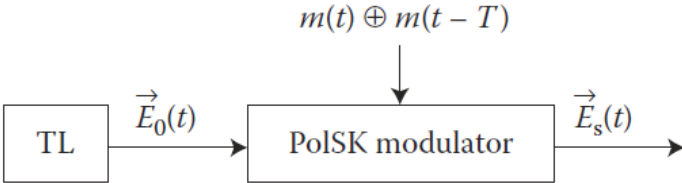


Abbildung 17: Blockdiagramm des DCPoISK Senders für ein kohärentes optisches Kommunikationssystem.
TL („transmitting laser“); PolSK, externer Polarisationsmodulator [1]

3. Aufbau des optischen Kommunikationssystems

In diesem Kapitel werden die einzelnen Komponenten, des optischen Systems aufgezählt und näher erläutert. Während die optische Quelle, der Modulator und der FPGA bereits vorhanden waren, aufgrund der Arbeit an einem anderen Projekt, musste für den optischen Empfänger ein Board entworfen werden. Infolge dessen wird für diese Komponenten auf [6] verwiesen, für genauere Informationen.

3.1. Optische Quelle

Die optische Quelle ist ein Laser der Firma Oclaro (LambdaFLEX™ iTLA TL5000 Integrated Tunable Laser Assembly). Er ist vom Typ TL5000WDJ welcher bereits in einem vorherigen Projekt verwendet wurde. Für die Bedienung des Lasers wurde eine Spannungsversorgung und ein Programm entworfen. Der Laser kann zwischen 7.3dBm und 14dBm operieren und bei Wellenlängen zwischen 1528nm und 1563nm arbeiten. Diese Parameter lassen sich mittels des Programms in den Rahmenbedingungen ändern [6].

Gesteuert wird der Laser mittels eines Raspberry Pi, welcher die gesamte Kommunikation mit dem Laser übernimmt, auf diesem befindet sich auch das konsolenbasierte Programm, welches zur Steuerung dient. Starten kann man es, indem man in einem neuen Terminal zum Ordner „Kommunikationsprogramm Laser“ navigiert und das darin enthaltene Executable „main“ ausführt. Die Befehle und deren Bedeutung sind in [6] genau beschrieben. Die wichtigsten davon sind:

- `laseron`: zum Einschalten des Lasers
- `laseroff`: zum Ausschalten des Lasers
- `power`: zum Einstellen der Leistung
- `wavelength`: zum Einstellen der Wellenlänge
- `info`: für die aktuellen Einstellungen
- `quit`: zum Beenden des Programms

Diese reichen für den normalen Betrieb aus. Der Ausgang des Lasers ist mit einem „*subscriber connector*“ SC-Stecker abgeschlossen [6]. Wichtig ist noch, das der Laser immer auf einem Kühlkörper liegt, da er sonst zu heiß wird und sich somit nicht mehr einschalten lässt. Die schematische Darstellung in Abbildung 18 sorgt für eine Übersicht aller Hardwarekomponenten, die für den Betrieb des Lasers benötigt werden. Wie zu erkennen ist, wird der Laser über ein ATX-Netzteil versorgt, welches zugleich den Raspberry Pi mitversorgt [6].

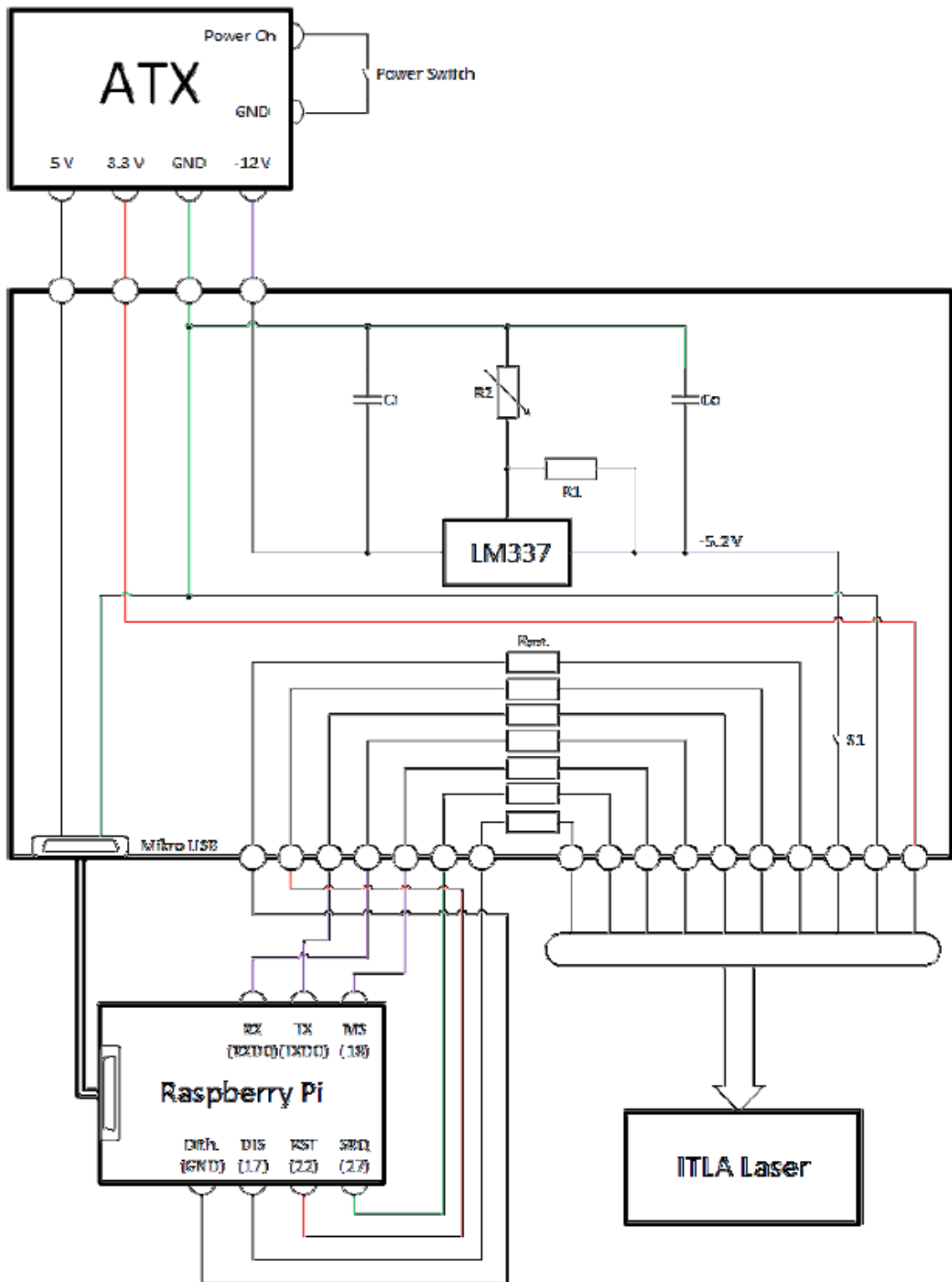


Abbildung 18: Hardware Übersicht für den Laser [6]

3.2. Externer Modulator

Beim verwendeten Modulator handelt es sich um einen LiNbO_3 (Lithiumniobat) Mach-Zehnder Modulator, von der Firma Fujitsu, mit einer Bitrate von 12 GBps (Gigabit pro Sekunde) [7]. Dieser wurde für das Projekt ausgewählt, da dieser Modulator im Labor vorhanden ist und er die nötigen Voraussetzungen erfüllt. Diese Voraussetzungen wären eine Bitrate von 10 GB/s sowie die Kompatibilität mit unserem Laser, Wellenlänge 1550nm, sowie mit dem FPGA.

Der Modulator selbst besitzt vier Anschlüsse, zwei davon für die Bias-Spannung und die anderen zwei für eine Monitoring Diode, mit deren Hilfe es möglich wäre, die BIAS-Spannung dynamisch einzustellen. Dies wird aber nicht implementiert, da für den Aufbau die Bias Spannung auf 5V eingestellt wird, wo sie in der Nähe des Arbeitspunktes liegt mit dem eine gute Funktionalität gewährleistet wird. In Abbildung 19 sieht man wie der Mach-Zehnder Modulator aussieht, die beiden Leitungen dienen für erste Tests mit dem Modulator.



Abbildung 19: Mach Zehnder Modulator

Die Abbildung 20 zeigt eine genaue Darstellung des Modulators. Die vier Anschlüsse sind von links nach rechts, G = GND, B = Bias, A = Anode und C = Cathode [7]. Die Anode und Cathode werden nicht verwendet.

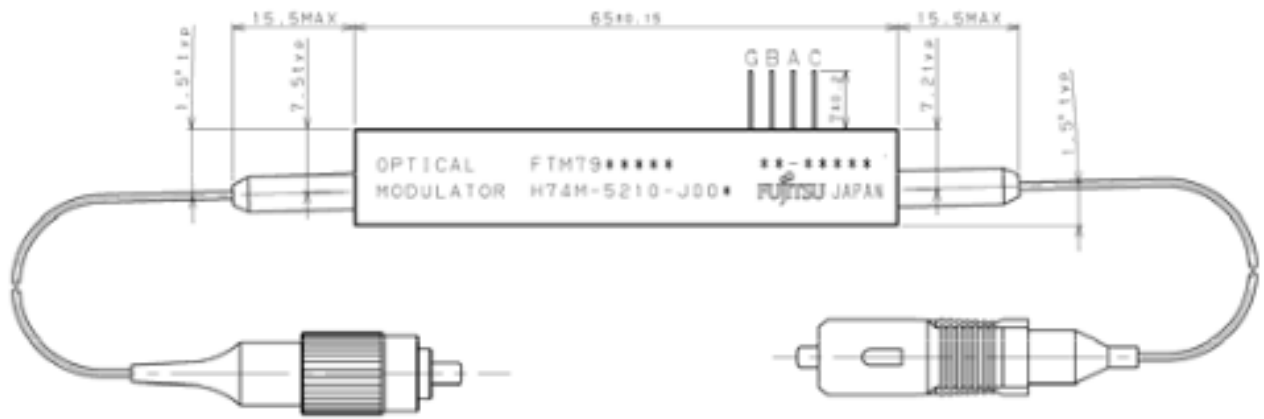


Abbildung 20: Genaue Darstellung des Modulators mit seinen Anschlüssen und Abmessungen [7]

Bei der Abbildung 21 erkennt man inwiefern die Bias-Spannung mit der Ausgangsleistung zusammen hängt. Es ist dabei sehr wichtig wo man den Arbeitspunkt einstellt und wie groß die ER (Extinction ratio) ist. Hier ist unser Arbeitspunkt mit 5V Bias Spannung eingestellt, in Abbildung 21 wäre dies in der Nähe des rechten „quadrature point“, dies hat zur Folge, dass die Ausgangsleistung des Modulators erhöht wird, wenn der FPGA eine logische 1 sendet und verringert wird, wenn eine logische 0 gesendet wird. Dies bedeutet für unser PPM Signal, das die optische Ausgangsleistung ansteigt bei einer logischen 1 und abnimmt bei einer logischen 0. Man könnte den Arbeitspunkt auch auf eine andere Position setzen, nur dann würde z.B. beim linken „quadrature point“ bei einer logischen 1 eine Verringerung der Ausgangsleistung entstehen und man müsste das Signal beim Empfangen erst invertieren bevor man weiterarbeiten kann. Aus diesem Grund wird ein Arbeitspunkt verwendet der keine unnötigen Operationen verlangt.

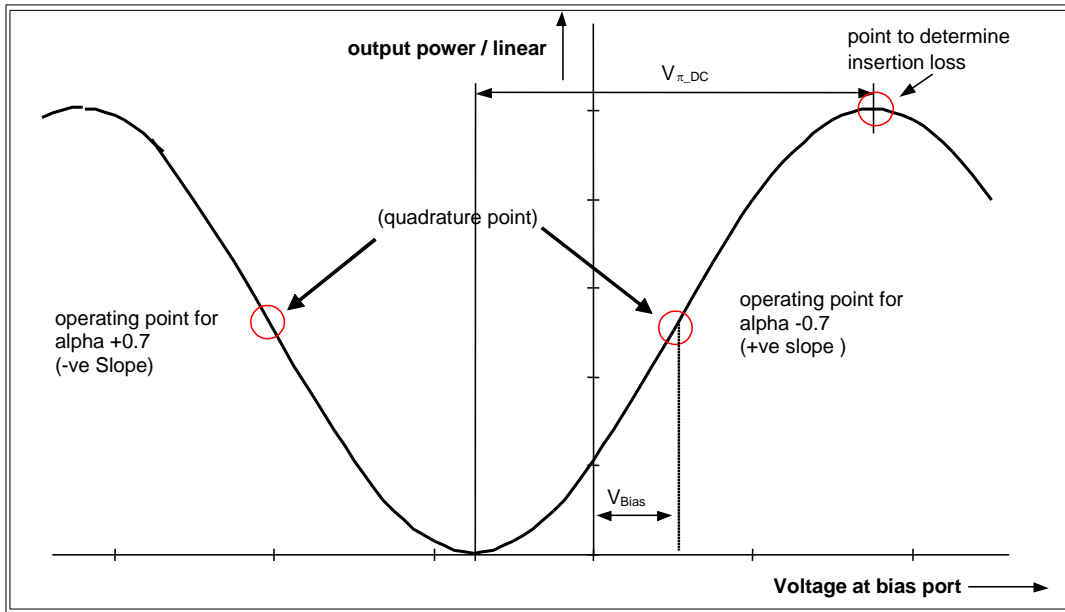


Abbildung 21: Kennlinie für Bias-Spannung zur Ausgangsleistung [15]

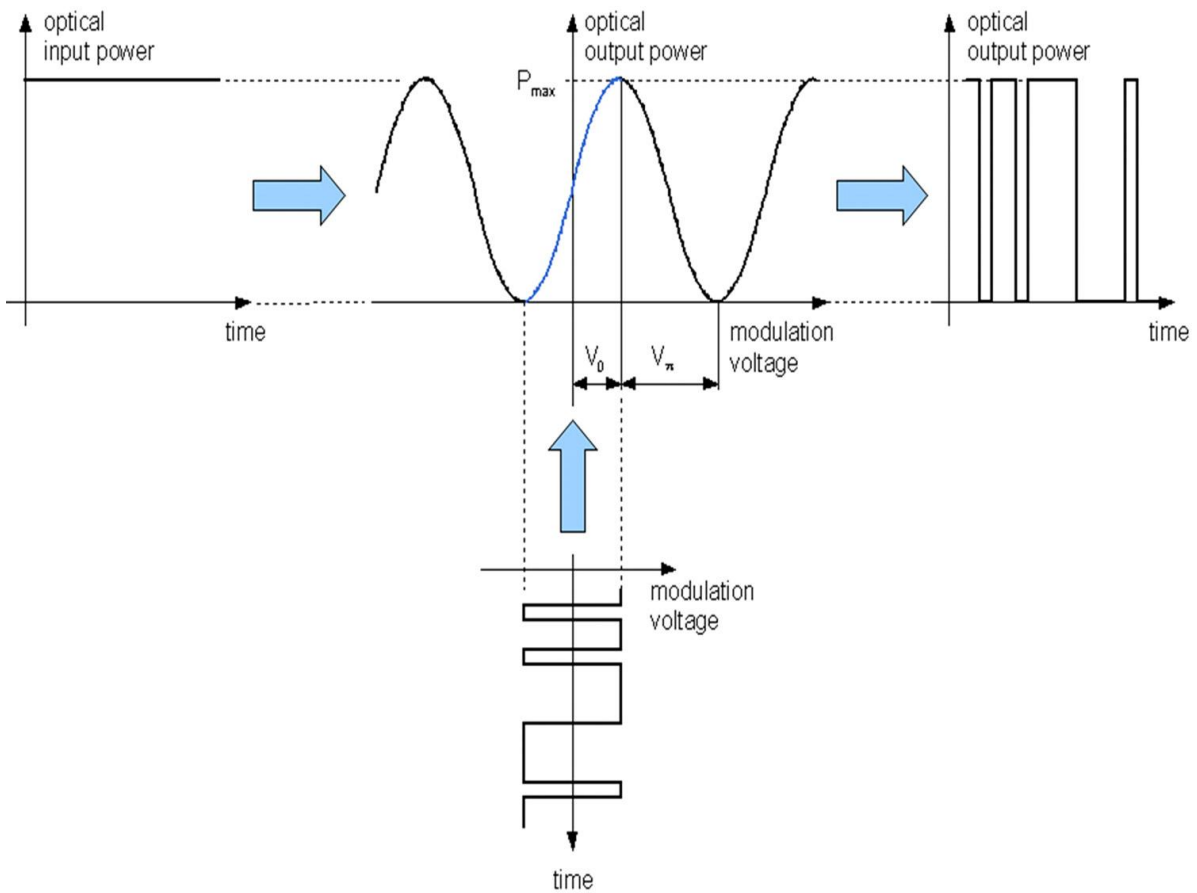


Abbildung 22: Beispiel für Mach Zehnder Modulator für PPM Modulation [17]

Abbildung 22 dient als Beispiel wie die PPM Modulation im Modulator aussieht. Man erkennt deutlich den beschriebenen Anstieg der optischen Ausgangsleistung bei einer logischen 1 und die Abnahme bei einer logischen 0. Ebenfalls sieht man sehr schön welchen Einfluss der gewählte Arbeitspunkt besitzt.

Weiteres besitzt der Modulator auf seinem Eingang einen SC-Stecker, welchen auch der Laser besitzt und am Ausgang besitzt er einen MU- (miniature unit-coupling) Stecker oder auch Mini SC genannt. Da der MU-Stecker sich nicht mit dem LC- (lucent connector) Stecker des Empfängers verbinden lässt, wird ein MU/MU Verbinder zusammen mit einem Kabel, dass auf der einen Seite einen MU-Stecker und auf der anderen einen LC-Stecker besitzt, verwendet. Diese Lösung wurde gewählt, da sie viel kostengünstiger ist, als einen Verbinder von MU auf LC zu kaufen.

Auch besitzt der Modulator einen HF-Eingang, dieser wird, wie in Abbildung 22 zu erkennen ist, für die „modulation Voltage“ verwendet. Das Signal, welches der FPGA an den Modulator sendet, wird in diesen Eingang gespeist. Somit ergibt sich aus Bias-Spannung und dem HF-Signal, das optische Ausgangssignal am Modulator.

Da der Eingang des HF-Signals am Modulator ein einzelnes Signal erwartet und das Ausgangssignal des FPGAs aber ein Differenzielles ist, gibt es zwei Lösungen für dieses Problem.

Für die erste Möglichkeit, wird eine Schaltung gebaut, welche das differenzielle Signal umwandelt. Dies könnte mit einem Balun erfolgen. Die zweite Möglichkeit wäre, nur eines der beiden Signale zu verwenden und somit auf die Entwicklung einer Schaltung zu verzichten. Doch dafür müsste das einzelne Signal groß genug sein. Um diese Möglichkeit überprüfen zu können musste zuerst das Programm soweit geschrieben werden, um dieses Signal zu erzeugen und messen zu können. Dieser Teil wird in der Ergebnissektion erläutert.

3.3. FPGA

Hierbei dreht es sich um den Spartan-6 FPGA SP605 von der Firma Xilinx, warum dieser ausgewählt wurde wird in [6] genau beschrieben. Der FPGA besitzt ein eingebautes Transceiver Modul, welches zum Senden der Daten an den Modulator und zum Empfangen der Daten vom optischen Empfänger genutzt wird. Der Hauptgrund warum ein FPGA das Modulationssignal an den Modulator sendet, besteht darin, dass normale Mikrokontroller keine Datenraten im GBit/s unterschützen können [6]. Eine Übersicht über den FPGA und seiner Komponenten ist in Abbildung 23 zu sehen. Dort sieht man auch die Anschlüsse für das Senden und Empfangen der Daten (Lower-power GTP Transceiver (RX/TX)).

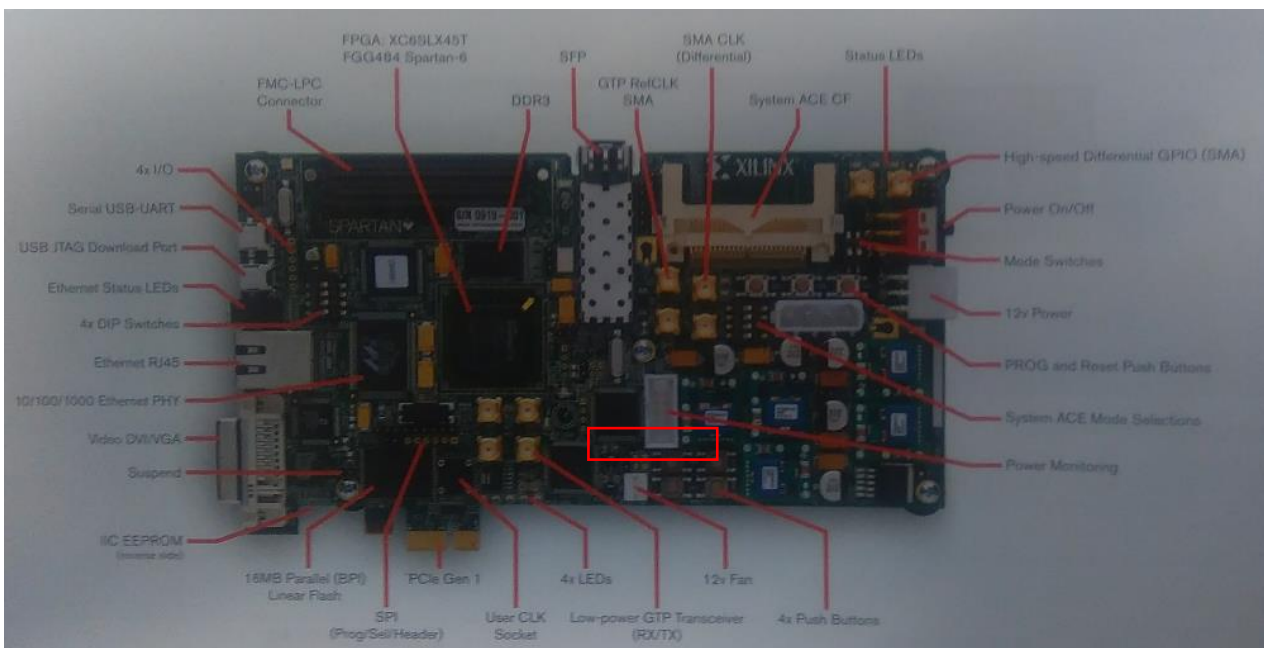


Abbildung 23: Übersicht über den FPGA

Für einen genaueren Überblick über die Anschlüsse für den Transceiver und den Receiver dient Abbildung 24. Die beiden oberen Anschlüsse RXP(J34) und RXN(J35) sind für den Receiver und die unteren TXP(J32) und TXN(J33) für den Transceiver.

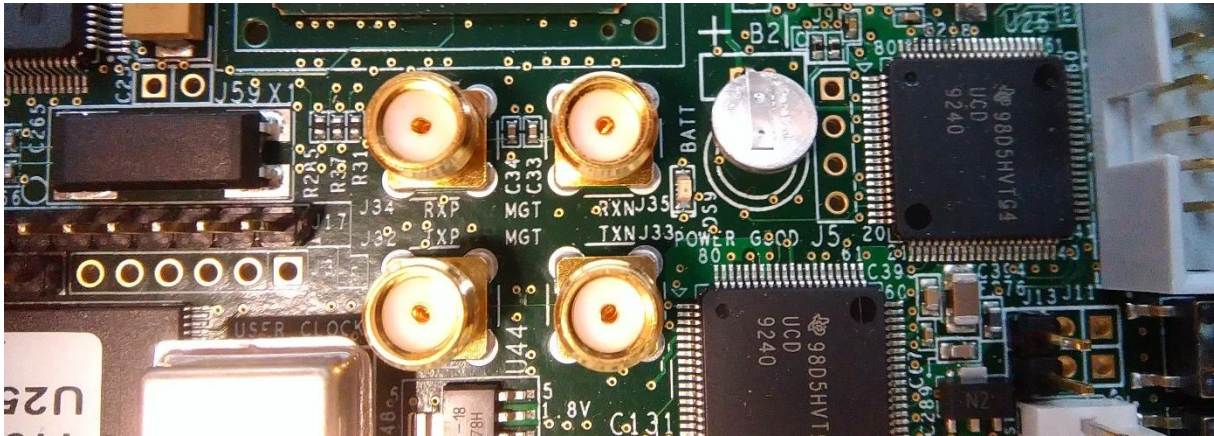


Abbildung 24: Transceiver und Receiver Anschlüsse auf dem FPGA

Die Abbildung 25 zeigt die Anschlüsse J40 und J39, welche für verschiedene Tests benutzt werden.

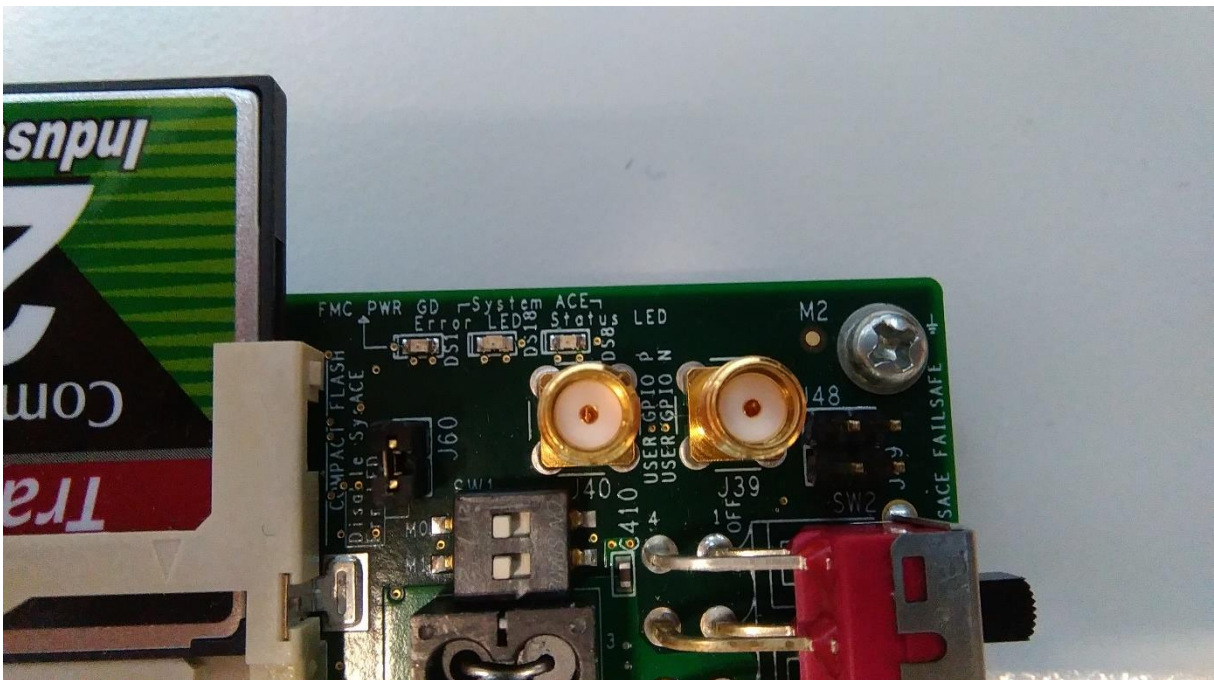


Abbildung 25: SMA Anschlüsse für Tests

Weiteres ist mit dem FPGA auch eine Kommunikation mit einem PC möglich, um somit auch dynamisch Daten versenden zu können. Natürlich wird dabei die Datenrate von der genutzten Übertragung zwischen FPGA und PC beschränkt. Es gibt eine Vielzahl an Möglichkeiten, auf dem FPGA, um mit Geräten zu kommunizieren, um einige zu nennen:

- Ethernet
- USB-UART
- SPI

Abbildung 26 zeigt zwei dieser Anschlüsse. Es gibt natürlich noch einige mehr, aber diese Drei waren in der engeren Auswahl für das Senden von Daten zwischen FPGA und PC. Die einfachste Form einer solchen Übertragung stellt eine UART (Universal Asynchronous Receiver Transmitter) dar. Der FPGA bietet dafür eine sehr gute Lösung mittels der USB-UART-bridge. Diese könnte dazu verwendet werden eine UART mittels USB zu realisieren. Eine genaue Beschreibung wie diese implementiert wird ist in Kapitel 4 zu finden.

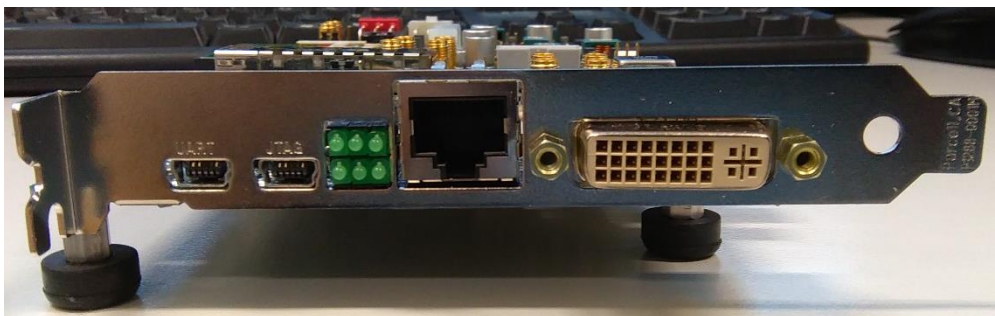


Abbildung 26: Port Anschlüsse für den FPGA

3.4. Optischer Empfänger

Der für dieses Projekt genutzte Empfänger ist der ERA1402GT von der Firma Sumitomo Electric. Dieser Empfänger besitzt die Möglichkeit zum Empfangen von Übertragungen im 10 GB/s Bereich. In Abbildung 27 sieht man den Empfänger ohne sein Board zur Ansteuerung. Er selbst besitzt 17 Pins, sein Layout für Diese kann man in Abbildung 28 sehen [8], wie man dort erkennt besitzt der Empfänger die Möglichkeit das empfangene Signal sowohl als differenzielles oder einzelnes Signal auszugeben. Für die differenzielle Variante steht eine maximale Ausgangsspannung von 1100mVpp (Voltage peak-to-peak) zur Verfügung und für die einzelne Variante stehen 550mVpp zur Verfügung [9]. Dabei spielt es keine Rolle ob man den negativen oder positiven Ausgang benutzt, einzig eine Invertierung des Signals ist beim negativen Ausgang von Nöten. Für dieses Projekt bietet sich der differenzielle Ausgang perfekt an, da der FPGA selbst ein differenzielles Signal erwartet.

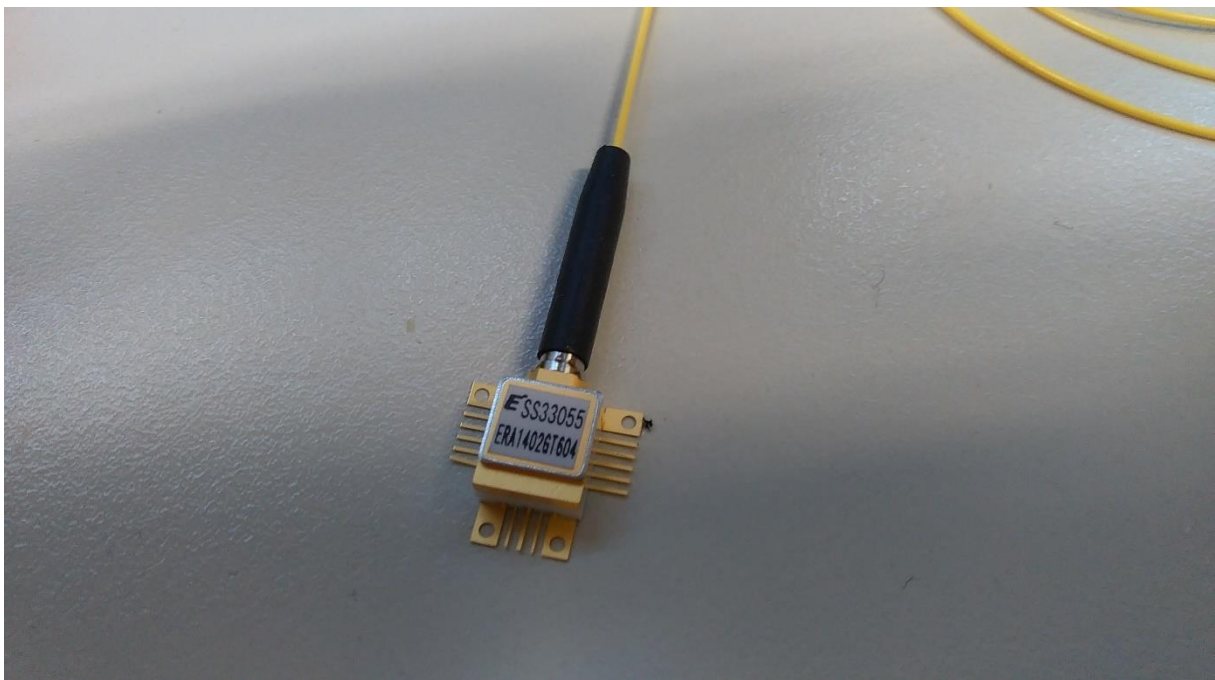


Abbildung 27: Optischer Empfänger ERA1402GT

Als Versorgungsspannung des Empfängers werden zwischen 6-8V gebraucht, die Wellenlänge der empfangenen Signale kann zwischen 1528-1564nm liegen und ist damit geeignet für unser verwendetes 1550nm PPM Signal. Weitere Kenndaten des Empfängers können aus dem Datenblatt im Anhang Entnommen werden [8].

Table: DC Bias Pinning		
No	Symbol	Function
1	GND	Case Ground
2	V _{PD}	PD Supply Voltage
3	NC	No connection
4	NC	No connection
5	NC	No connection
6	GND	Case Ground
7	GND	Case Ground
8	OUTN	Negative Output
9	GND	Case Ground
10	OUTP	Positive Output
11	GND	Case Ground
12	GND	Case Ground
13	NC	No connection
14	V _{CC}	TIA Supply Voltage
15	NC	No connection
16	NC	No connection
17	GND	Case Ground

Abbildung 28: Pinlayout des Empfängers

4. Erstellung des FPGA Programms

In diesem Kapitel werden die Erstellung des Frameworks, die einzelnen Elemente und das transferieren des Programm auf den FPGA erläutert und beschrieben. Darunter die UART, welche für das empfangen und senden von Daten zwischen FPGA und PC verantwortlich ist. Der Transmitter, der das PPM Signal erzeugt und an den Modulator schickt und der Receiver, welcher das Signal wieder empfängt und demoduliert.

4.1. Erstellung des Frameworks

Dieser Abschnitt beschäftigt sich mit der Erstellung des Frameworks für den Transmitter und Receiver. Das erstelle Framework selbst diente nur als Grundgerüst und ist nur geeignet die Transmitter und Receiver Module auf dem FPGA zu testen. Erstellt und geschrieben wurde das Programm mittels der „ISE Desing Suite: WebPack Edition“ Software von der Firma Xilinx, diese wurde mit dem FPGA mitgeliefert und deswegen auch für das Projekt genutzt. Die verwendete Programmiersprache ist Verilog, da diese sehr Hardware nahe und übersichtlich ist. Neben Verilog wäre noch die Sprache VHDL möglich für den FPGA. Nun kommen wir zur Erstellung des Frameworks.

Der Erste Schritt ist es zuerst ein neues Projekt zu erstellen, dies geschieht mit einem Klick auf "New Projekt", in Abbildung 29 sieht man wo diese Anweisung zu finden ist.

Im nächsten Schritt wird der Name und der Speicherort für das Projekt ausgewählt. Darauf folgend wird mit einem Klick auf "Next" die Hardwareeinstellung getroffen, diese sieht für den FPGA wie in Abbildung 30 zu sehen ist aus, das wichtigste bei dieser Einstellung ist es das richtige FPGA-Board auszuwählen. Dieses wurde in einem vorherigen Kapitel schon genauer beschrieben.

Ist dies passiert, Klick man auf "Next" und danach auf "Finish". Somit erstellt der Editor ein neues Projekt.

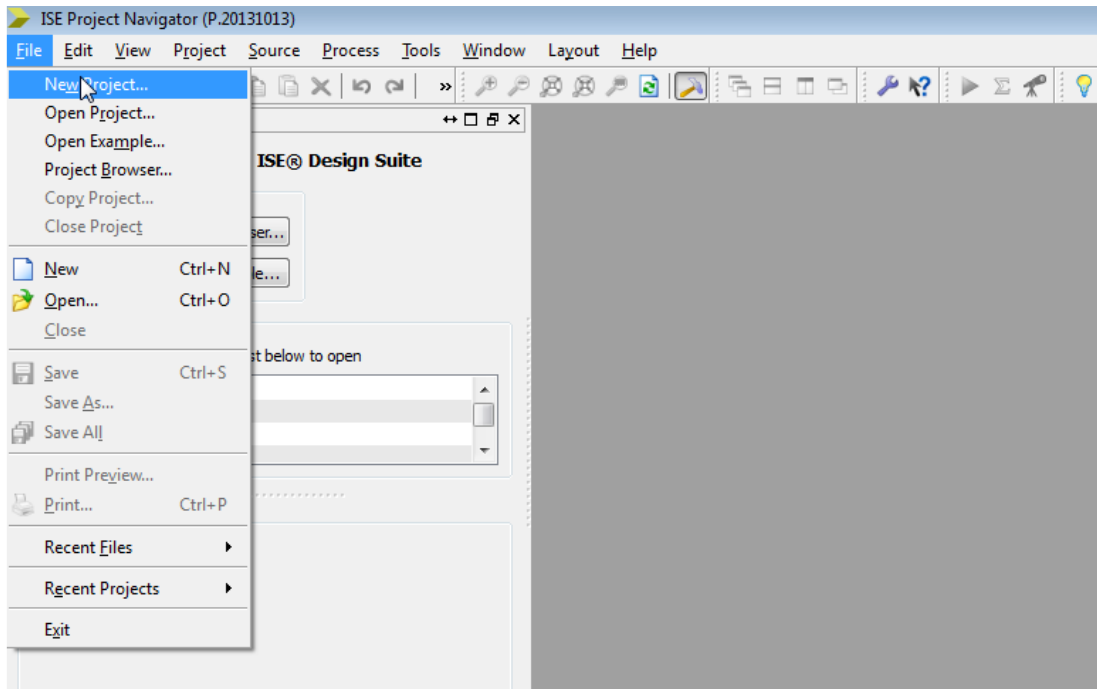


Abbildung 29: Neues Projekt erstellen

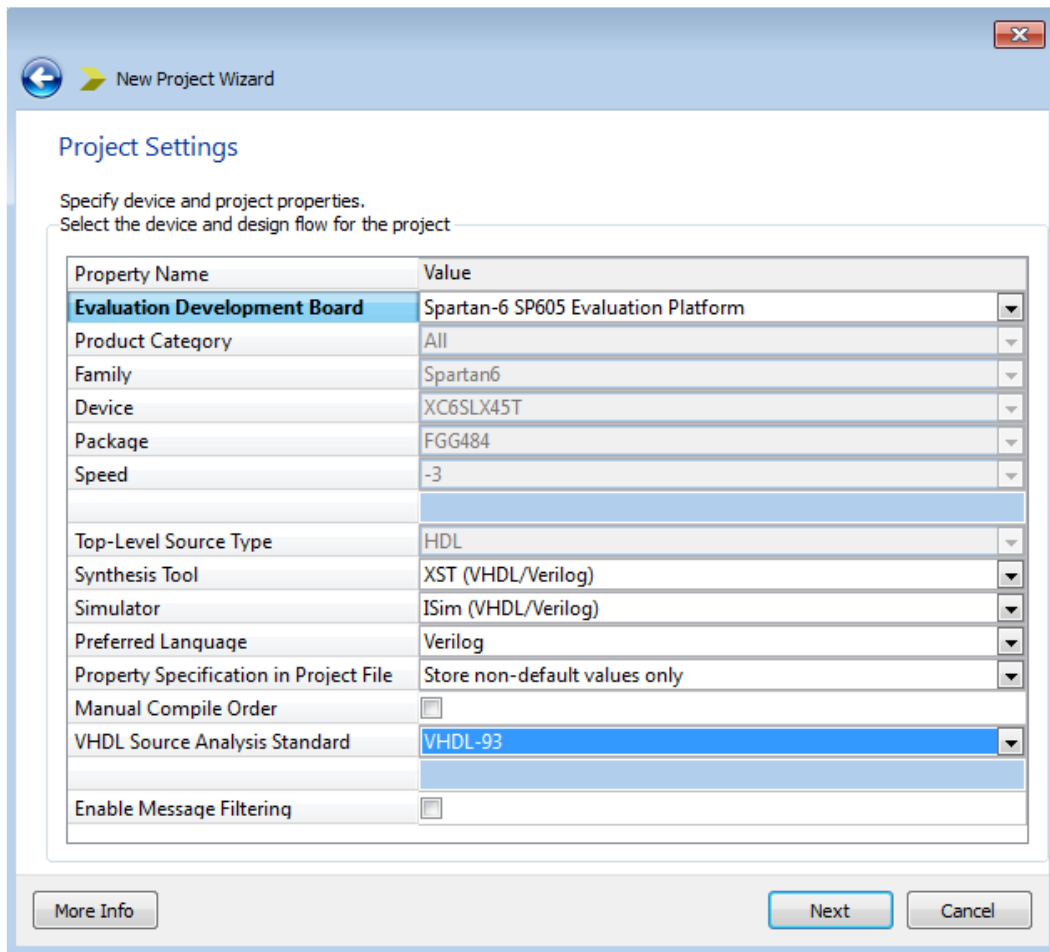


Abbildung 30: Hardwareeinstellung für den FPGA

Der nächste Schritt ist es ein "Coregen" Projekt zu erstellen. Diese Art der Projekte besitzen viele vorgefertigten Frameworks für die verschiedenen Anwendungen des FPGAs. Zuerst öffnet man den "Core Generator" wie in Abbildung 31 zu sehen ist.

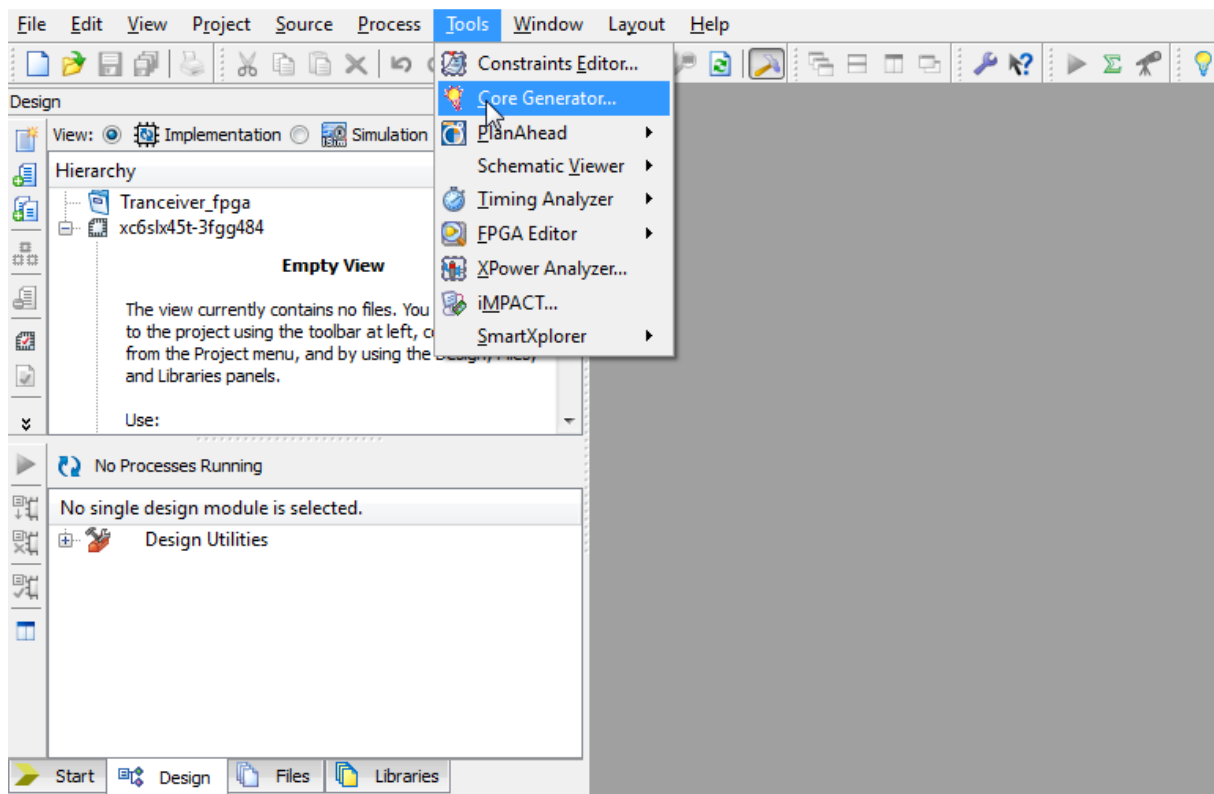


Abbildung 31: Öffnen des Core Generator Tools

Der "Core Generator" öffnet nun ein neues Fenster, indem ebenfalls ein neues Projekt erstellt werden muss, wie in Abbildung 32 abgebildet. Nach dem Klick für das Erzeugen eines neuen Projektes, muss ein Name und ein Speicherort gewählt werden. Es ist wichtig, dass der Speicherort unverändert bleibt, aus dem einfachen Grund, dass die Struktur der erstellten Projekte nicht beschädigt wird und keine Probleme entstehen.

Nachdem Name und Speicherort gewählt sind, werden anschließend Parameter für das Framework festgelegt, der Wichtigste ist die Auswahl des richtigen FPGA-Boards. Abbildung 33 zeigt das der verwendete FPGA mit xc6slx45t bezeichnet wird.

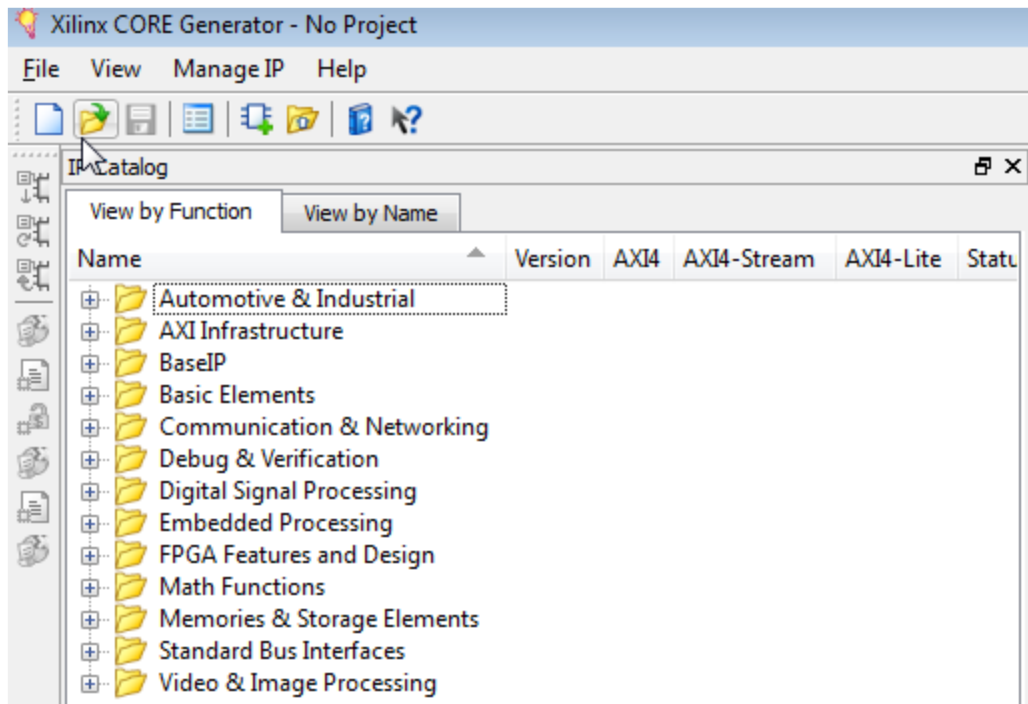


Abbildung 32: Core Generator Fenster

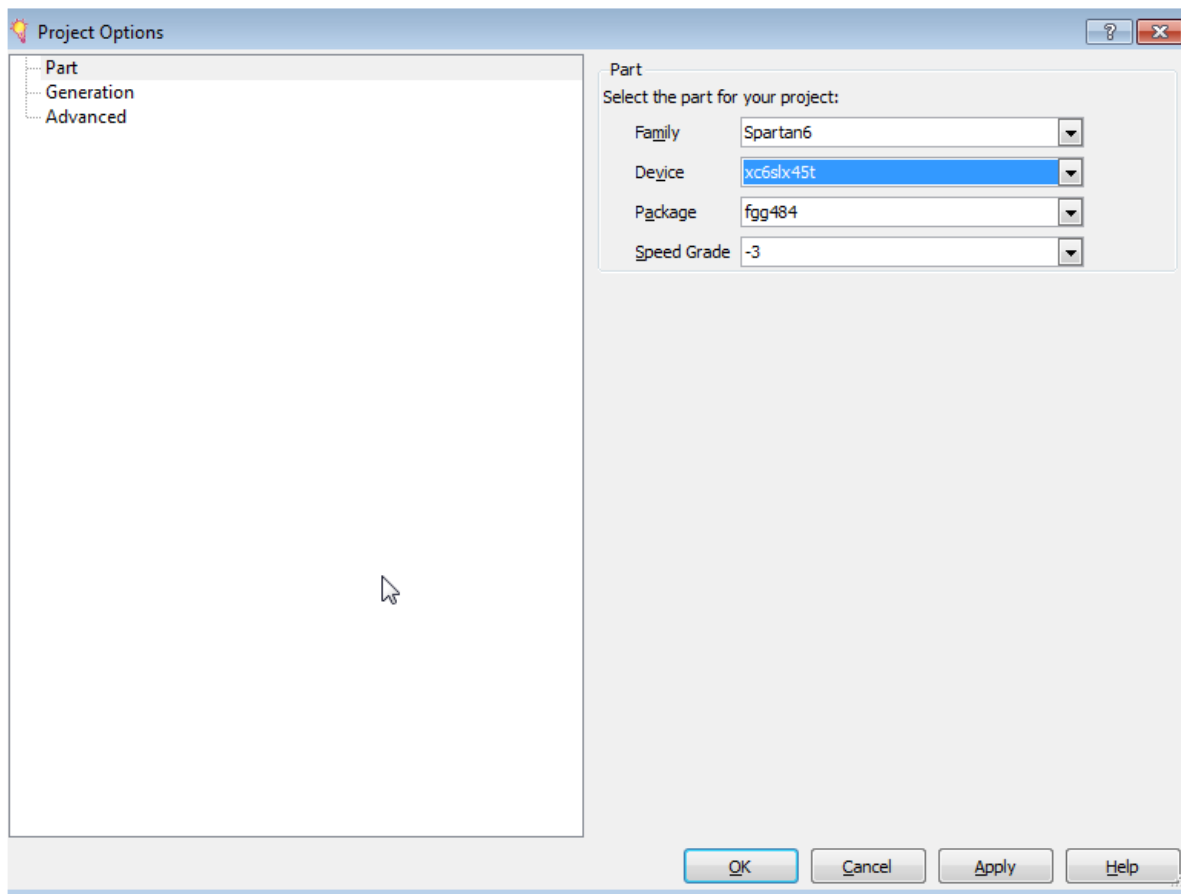


Abbildung 33: Parameter Einstellung für das Core Projekt

Nach diesen Einstellungen wird auf der linken Seite von Abbildung 33 der „Generation“ Abschnitt ausgewählt. Im „Generation“ Abschnitt wird nun die Programmiersprache Verilog gewählt. Die Abbildung 34 zeigt diese Einstellung. Zum Schluss wird mit dem Klick auf OK das Projekt angelegt.

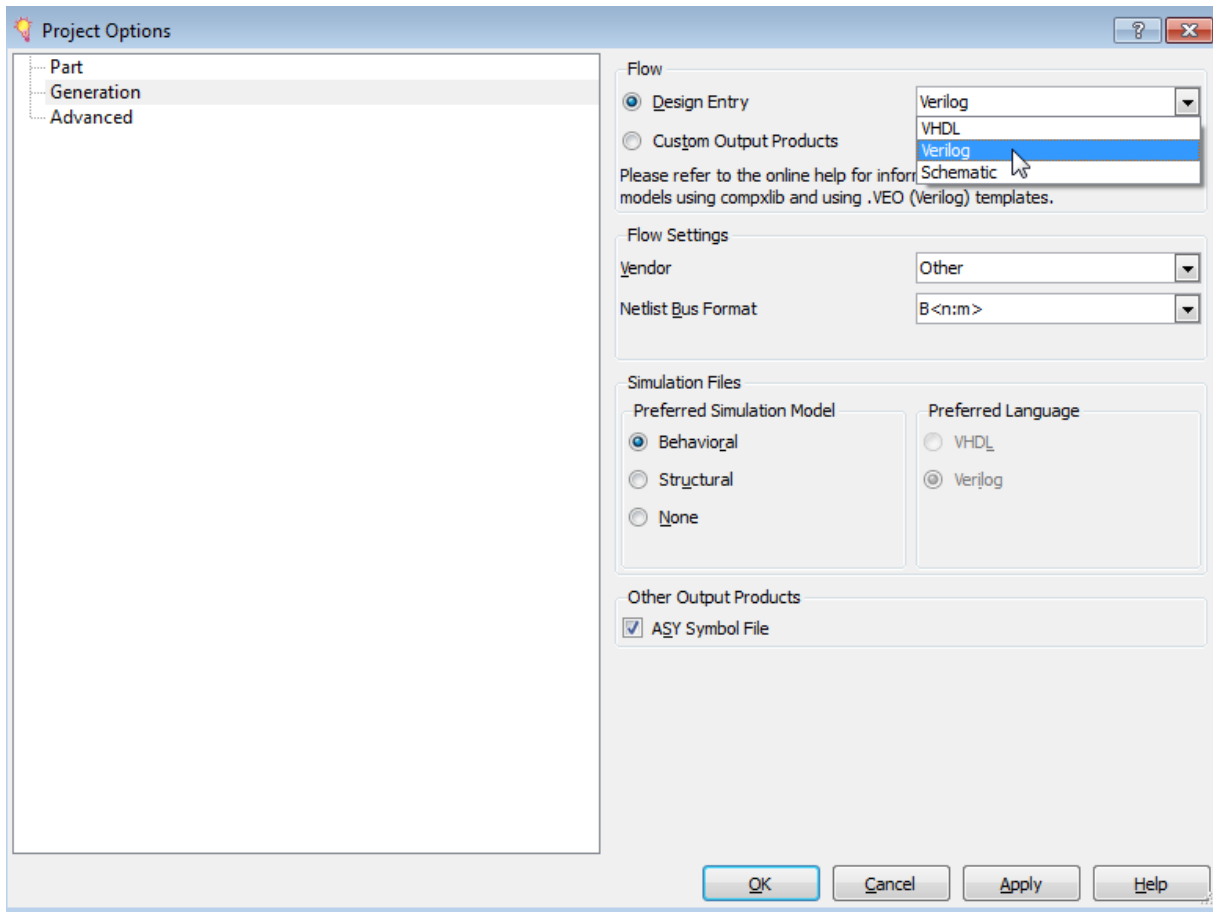


Abbildung 34: Einstellung der Programmiersprache

Ab diesem Zeitpunkt ist es möglich eines der vielen Frameworks die der FPGA besitzt zu erzeugen. Da der Transceiver benötigt wird, ist der „Transceiver Wizard“ das perfekte Tool um das Framework mit einigen wichtigen Parametern zu erstellen. Die Abbildung 35 zeigt wo der „Transceiver Wizard“ zu finden ist. Dieser wird gestartet um ein Transceiver Framework zu erzeugen.

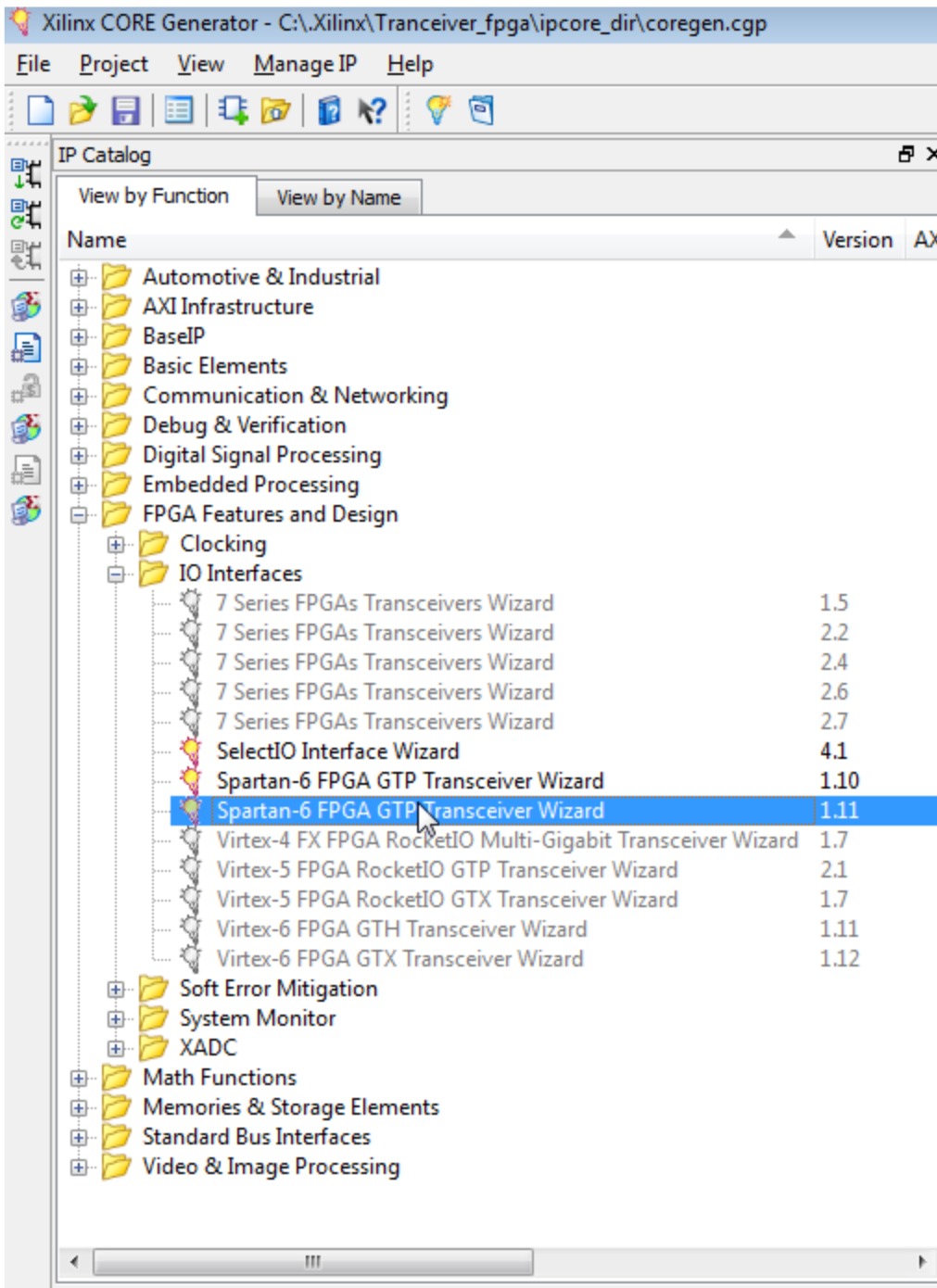


Abbildung 35: Navigation zum Transceiver Wizard

Nach dem Starten des „Transceiver Wizard“ wird als Erstes ein Name vergeben. Ebenfalls ist die Auswahl eines Transceivers zu treffen, der benutzt werden soll. Es gibt insgesamt vier verschiedene Transceiver, welche immer nur als Paar auswählbar sind. Wie Abbildung 36 darstellt heißen die beiden Paare GTPA1_DUAL_X1_Y0 und GTPA1_DUAL_X0_Y0. Jedes dieser Paare besitzt 2 Transceiver, diese sind wie folgend aufgelistet.

- GTPA1_DUAL_X0_Y0_0 = PCIe
- GTPA1_DUAL_X0_Y0_1 = SMA
- GTPA1_DUAL_X1_Y0_0 = SFP
- GTPA1_DUAL_X1_Y0_1 = FMC_LPC

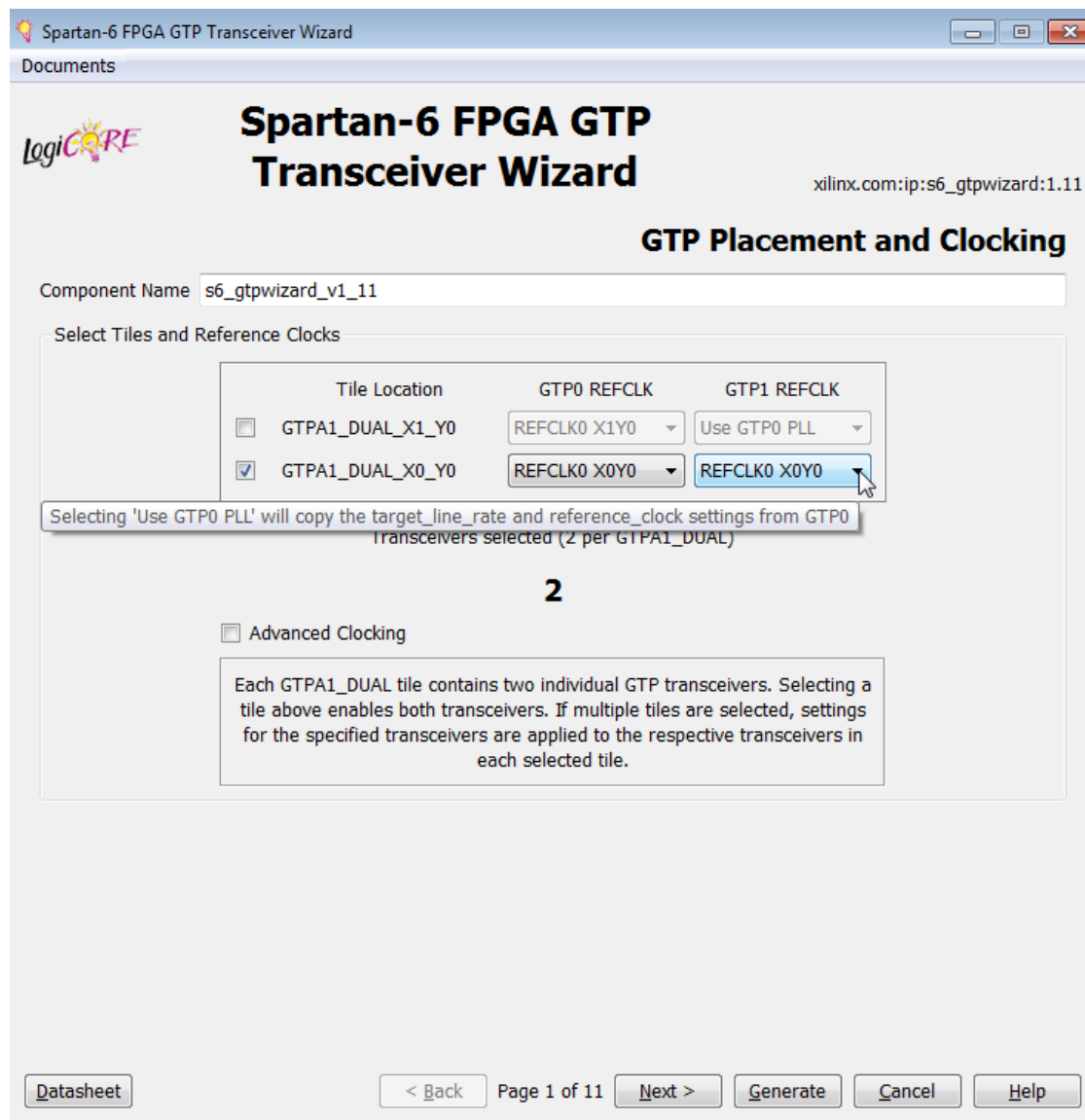


Abbildung 36: Startseite des Transceiver Wizard mit Auswahl der möglichen Transceiver

Da nur der Transceiver mit dem SMA Ausgang von Interesse ist, wird GTPA1_DUAL_X1_Y0 wie in Abbildung 36 ausgewählt. Als „Referenzclock“ wird die REFCLK0 X0Y0 gewählt, diese Einstellung bestimmt nur, welche „Clock“ genutzt werden soll. Mit dieser Einstellung wird die „Clock“ die zum Transceiver Paar X0Y0 gehört verwendet [10].

Auf der nächsten Seite des „Transceiver Wizard“ wird die „Line Rate“, das Coding und die Bandbreite für den Transceiver, sowie für den Receiver eingestellt, dies wird in Abbildung 37 abgebildet.

Als Template wird „Start from scratch“ gewählt, somit können alle Parameter selbst eingestellt werden. Die „Line Rate“ wird auf 2,7Gb/s gesetzt, die höchst mögliche Einstellung in diesem Framework. Das Coding wird deaktiviert, da für PPM kein Coding nötig ist und somit mehr Bandbreite zur Verfügung steht. Die Bandbreite wird auf 32 Bits gesetzt, damit erreicht man eine 32 PPM und somit steht einem das gesamte Alphabet und einige Satzzeichen zur Verfügung.

Spartan-6 FPGA GTP Transceiver Wizard

Documents

LogiCORE

Spartan-6 FPGA GTP Transceiver Wizard

xilinx.com:ip:s6_gtpwizard:1.11

Line Rate and Protocol Template

Use Dynamic Reconfiguration Port

GTP0

Protocol Template
Start from scratch

Select optional protocol template above

Common Settings
Target Line Rate 0.675 Gbps Reference Clock 135.00 MHz

TX Settings
Line Rate 0.675 Gbps
Encoding 8B/10B
Data Path Width 32 Bits

RX Settings
Line Rate 0.675 Gbps
Decoding 8B/10B
Data Path Width 32 Bits

GTP1

Protocol Template
Use GTP0 settings
Settings will be copied from GTP0

Common Settings
Target Line Rate 0.675 Gbps Reference Clock 135.00 MHz

TX Settings
Line Rate 0.675 Gbps
Encoding 8B/10B
Data Path Width 32 Bits

RX Settings
Line Rate 0.675 Gbps
Decoding 8B/10B
Data Path Width 32 Bits

Datasheet < Back Page 2 of 11 Next > Generate Cancel Help

Abbildung 37: Parameter Einstellungen für den Transceiver

Der Receiver wird mit den gleichen Einstellungen wie der Transceiver eingestellt. Der zweite Transceiver wird auf die Einstellung „Use GTP0 settings“ gesetzt, damit haben beide Transceiver die gleichen Einstellungen.

Die nächste und damit letzte Seite von Relevanz, beschäftigt sich mit den „Coding ports“, Abbildung 38 stellt diese dar. Diese werden alle abgewählt und danach wird mit einem Klick auf „Generate“ das Framework erzeugt.

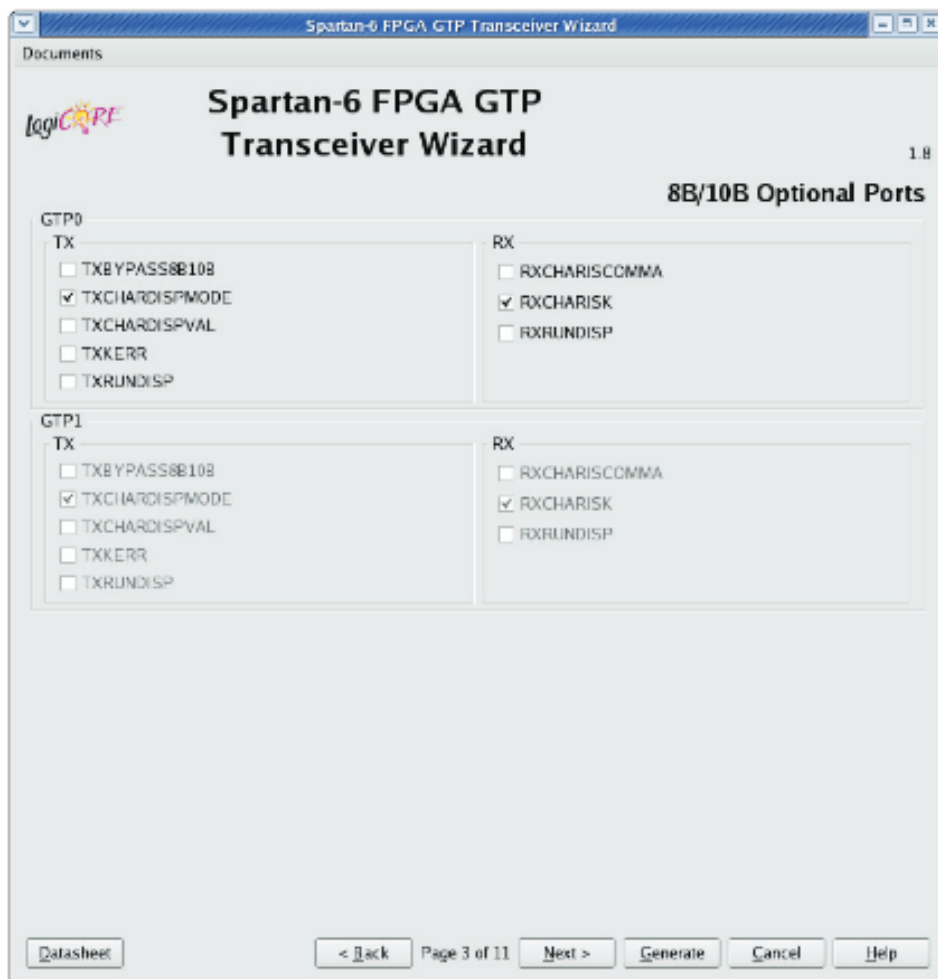


Abbildung 38: Coding Einstellungen für den Transceiver

Das erstellte Framework besitzt nun die in Abbildung 39 dargestellte Struktur. FRAME_GEN ist dabei für die Erzeugung der Daten für den Transceiver verantwortlich, während FRAME_CHECK die empfangenen Daten auf ihre Richtigkeit überprüft. Der Wrapper Abschnitt umfasst dabei die Kommunikation mit den Hardware Komponenten und allen Software Abschnitten. Auf diesem Framework kann nun die UART eingebaut werden und Transceiver und Receiver auf eine PPM Struktur umgebaut werden.

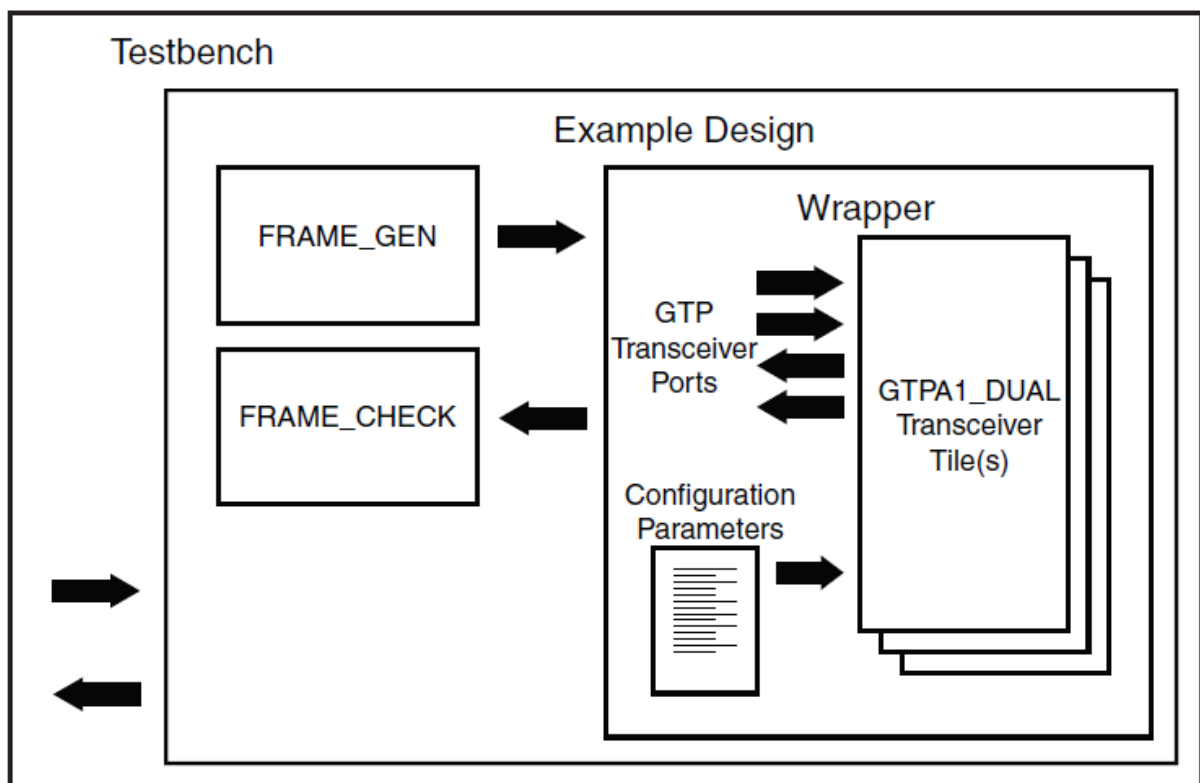


Abbildung 39: Struktur des Frameworks [10]

4.2. UART

Wie bereits erwähnt wurde, wird die USB-UART hier genau erläutert. Dies umfasst die Definition und Implementierung der USB-UART.

Eine UART ist eine der einfachsten Übertragungsformen, Sie besitzt lediglich eine Sende –und eine Empfängerleitung zwischen den beiden Komponenten. Diese beiden Leitungen sind seriell, also wird Bit für Bit übertragen, somit ist es nötig Daten von parallel zu seriell und umgekehrt umzuwandeln. Diese seriellen Daten benötigen natürlich auch ein Start- und Stoppsbit, wobei die Anzahl der Startbits immer 1 ist, aber die Anzahl der Stoppsbits zwischen 1, 1.5 und 2 variieren kann. Auch die Anzahl der Datenbits kann variieren und es gibt die Möglichkeit ein Paritätsbit einzufügen, um damit die Fehlererkennung zu verbessern. In Abbildung 40 ist ein solcher „Bitstream“ zu sehen, man erkennt deutlich das der „idle“ Zustand bei einer UART logisch 1 ist und die Spannung zwischen 3-15V (logisch 1) und -3- -15V(logisch 0) beträgt. Wichtig ist nur das beide Geräte die eine UART nutzen, die gleiche Anzahl an Datenbits bzw. Stoppsbits benutzen und wissen ob das Paritätsbit verwendet wird.

Neben dem Bitaufbau ist auch das richtige Taktsignal wichtig, eine UART nutzt aber keine zusätzliche Taktleitung, daher müssen die Sender und Empfänger mit dem gleichen Takt senden bzw. empfangen und um dies zu gewährleisten kommt die Baudrate ins Spiel. Diese gibt an wieviel Bits pro Sekunde übertragen werden sollen. Um nun den richtigen Takt berechnen zu können werden folgende Formeln verwendet.

$$TX_CLK = \frac{CLK}{Baudrate} \quad \text{Formel 4.1}$$

$$RX_CLK = \frac{TX_CLK}{16} \quad \text{Formel 4.2}$$

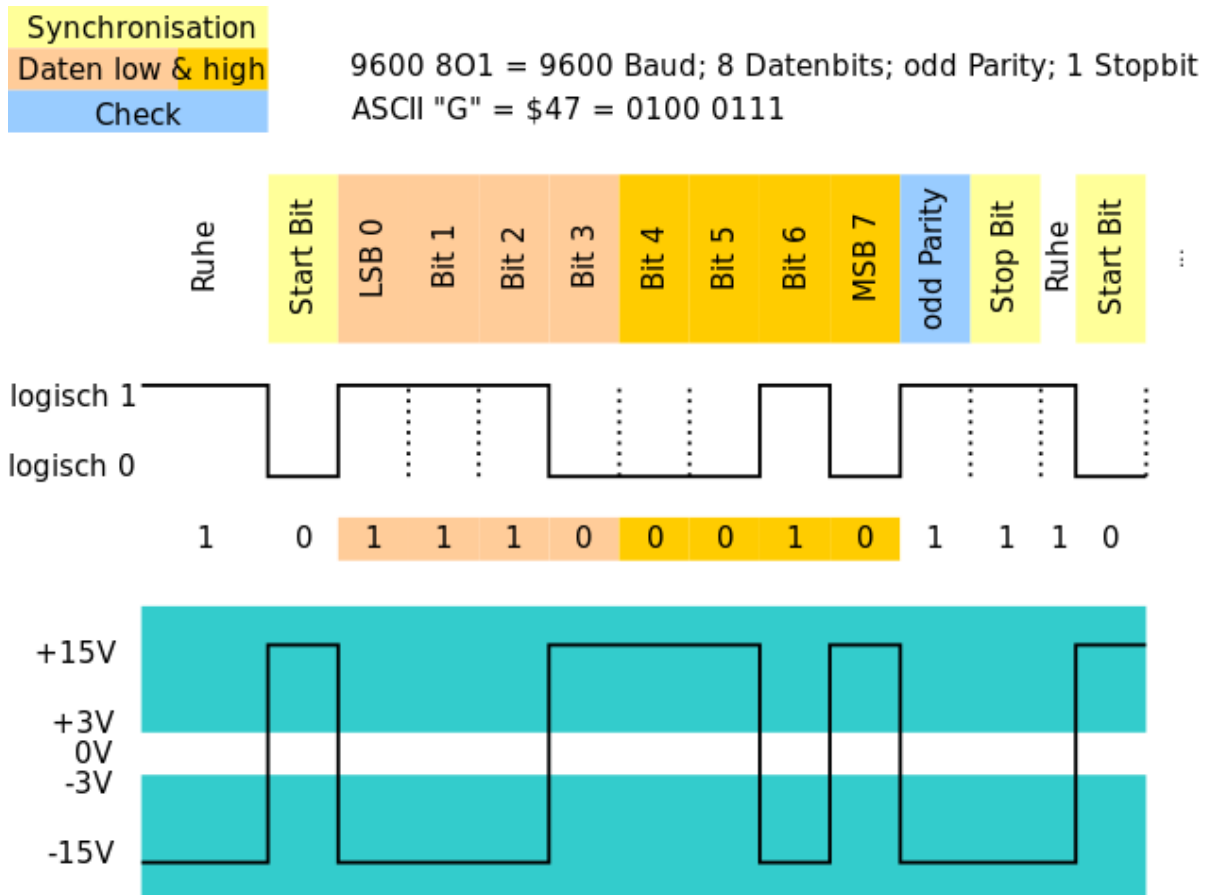


Abbildung 40: Aufbau des Bitstreams einer UART [11]

Wobei in Formel 4.1 der CLK („clock“) die Taktfrequenz der Komponente angibt und TX_CLK den Faktor angibt, mit welcher man die Taktfrequenz teilen muss, um die gewünschte Baudrate zu erzielen. Formel 4.2 errechnet den Faktor für den Empfänger, die Teilung durch 16 hat sich als guter Wert in den Jahren bewährt. Der Grund für die Teilung durch 16 ergibt sich daraus, dass das empfangene Bit in der Mitte abgetastet wird. Als nächstes wird erläutert wie die UART im FPGA implementiert wurde.

Der erste Schritt bei der Implementierung war es sich auf eine Baudrate und auf eine Bitfolge zu einigen. Es wurde eine Baudrate von 9600 genutzt, da diese bei den meisten Geräten standardmäßig genutzt wird. Weiteres wurde sich auf das Bitmuster 801 geeinigt, 8 Datenbits, 0 Paritätsbits und 1 Stopbit. 8 Datenbits, um immer ein Byte zu übertragen und man kann somit mit ASCII Zeichen arbeiten. Ein Paritätsbit sowie mehr als ein Stopbit, war bei unserer Anwendung nicht nötig. Nach diesen wichtigen Einigungen konnte mit der Erzeugung der Taktsignale begonnen werden.

Der FPGA besitzt dazu 2 Taktsignale, welche genutzt werden können, eines mit 200MHz und eines mit 27MHz. Da das Taktsignal noch geteilt wurde reichten 27MHz vollkommen. Die Berechnung der Teilungsfaktoren der UART ergab für TX_CLK = 2812,5, da aber nur positive Flanken im Code als Trigger dienen, musste der Wert durch zwei geteilt werden und ergab damit TX_CLK = 1406,25. Da keine ganze Zahl herauskam, musste jeder vierte Takt mit einer zusätzlichen Flanke auf 1407 korrigiert werden. Der Teilungsfaktor für RX_CLK = 87,89 wurde auf 88 gerundet. Da die Abweichung hier sehr gering ist, musste keine Korrektur, für das Empfängertaktsignal vorgenommen werden. Da bereits beim Taktsignal des Senders darauf geachtet wurde, dass nur positive Flanken getriggert werden, mussten für den Empfänger keine weiteren Berechnungen durchgeführt werden. In Abbildung 41 ist das Codefragment dargestellt, welches die beiden Taktsignale erzeugt.

```

always@(posedge UART_CLK)
begin
    tx_counter <= tx_counter + 1;
    rx_counter <= rx_counter + 1;

    //txclk
    if(tx_counter == uart_clk_correct)
        begin
            tx_counter <= 0;
            txclk <= ~ txclk;

            //2812,5/2 = 1406,25, to correct the error from 0,25, every 4th txclk will come at 1407, for clks errors
            if(uart_clk_correction == 3)
                begin
                    uart_clk_correct <= 1407;
                    uart_clk_correction <= 0;
                end
            else
                begin
                    uart_clk_correct <= 1406;
                    uart_clk_correction <= uart_clk_correction + 1;
                end
            end
        end

    //rxclk 16 times faster then txclk, to minimize errors 1406/16= 87,89 ~ 88
    if(rx_counter == 88)
        begin
            rx_counter <= 0;
            rxclk <= ~ rxclk;
        end
    end
end

```

Abbildung 41: Aufbau der Taktsignale für Sender und Empfänger der UART

In Abbildung 42 sieht man die Grundschematik der implementierten UART.

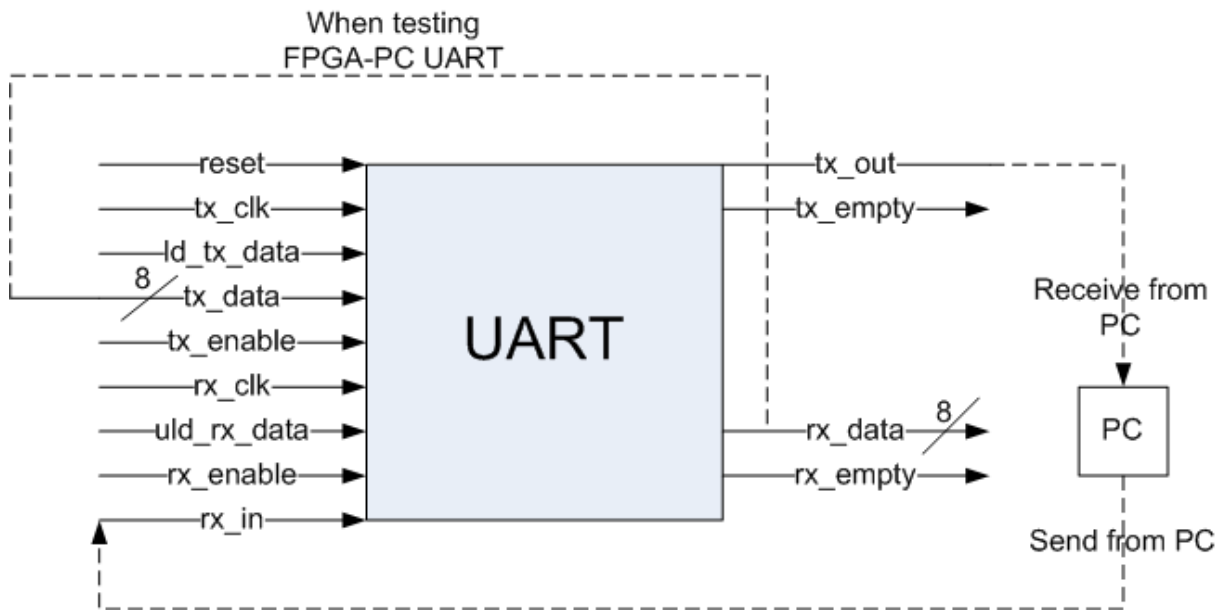


Abbildung 42: UART Schematik [12].

Diese besteht aus einer Sendeleitung (tx_out) – und einer Empfangsleitung (rx_in), sie werden zum Senden bzw. Empfangen der einzelnen Bits genutzt. Weiteres besteht die Schematik aus den zwei vorher beschriebenen Taktsignalen für Sender und Empfänger, dazu gibt es auch ein „enable“ Bit, mit dem das Senden bzw. Empfangen deaktiviert bzw. aktiviert werden kann. Die 8-Bit Register (tx_data und rx_data) stellen die Datenbits dar, diese werden mittels (ld_tx_data und uld_rx_data) geladen bzw. geleert, während (tx_empty und rx_empty) angeben, ob das nächste Symbol gesendet bzw. empfangen werden kann.

```

//Transmitter

always @ (posedge txclk or posedge tile0_tx_system_reset0_c)
begin
  if (tile0_tx_system_reset0_c) begin
    tx_reg      <= 0;
    tx_empty    <= 1;
    tx_over_run <= 0;
    tx_out      <= 1;
    tx_cnt      <= 0;
  end
  else begin

    //check if new data can be load to transmit
    if (ld_tx_data)
      begin
        if (!tx_empty)
          begin
            tx_over_run <= 0;
          end
        else
          begin
            tx_reg  <= tx_data;
            tx_empty <= 0;
          end
      end

    if (tx_enable && !tx_empty) begin
      tx_cnt <= tx_cnt + 1;
      //send start bit, always 0
      if (tx_cnt == 0) begin
        tx_out <= 0;
      end

      //send data bits
      if (tx_cnt > 0 && tx_cnt < 9) begin
        tx_out <= tx_reg[tx_cnt - 1];
      end

      //send stop bit always 1
      if (tx_cnt == 9) begin
        tx_out <= 1;
        tx_cnt <= 0;
        tx_empty <= 1;
      end
    end
    if (!tx_enable) begin
      tx_cnt <= 0;
    end
  end
end
end

```

Abbildung 43: Sender der UART

Die Abbildung 43 zeigt den Code des implementierten Senders der UART. Zu Beginn werden alle Register in ihren „IDLE“ Zustand gebracht, dies geschieht auch wenn ein „reset“ des Transceiver Moduls erfolgt. Der nächste Abschnitt überprüft ob neue Daten geladen werden können, dazu wird das Register ld_tx_data überprüft ob neue Daten zum Senden vorhanden sind. Damit ld_tx_data eine positive Rückmeldung gibt ist der Codeabschnitt in Abbildung 44 verantwortlich.

Dieser überprüft nur bei jedem neuen Takt, des UART-Sender Taktsignals, ob alle Daten gesendet wurden und ob der UART-Sender wieder bereit ist. Nach diesen Überprüfungen kommt der letzte Abschnitt in Abbildung 43, das senden der Daten, dazu wird als erstes eine logische 0 gesendet, das Startbit, danach kommen die 8 Datenbits und zum Schluss wird das Stopbit gesendet.

```
//check if the transmitter is ready for new data to send

if(tx_empty)
begin
    if(tx_delay_enable && tx_data != 8'h0)
    begin
        tx_enable <= 1;
        ld_tx_data <= 1;
        tx_delay_enable <= 0;
    end
end
else
begin
    ld_tx_data <= 0;
end
```

Abbildung 44: Überprüft ob der UART-Sender bereit ist neue Daten zu senden

Der letzte Teil des UART-Senders wird in Abbildung 45 dargestellt. Dieser hat die Aufgabe, nachdem ein Sendevorgang gestartet wurde, den Sender nach 10 Takten des UART-Sender Taktsignals, wieder abzuschalten. Dies hat den einfachen Grund, dass es 10 Takte braucht für die Übertragung eines Symbols. Würde dies nicht geschehen würde der UART-Sender einfach das letzte Symbol erneut senden, wenn keine neuen Daten angelegt werden.

```
if(tx_enable)
begin
    count_disable_tx <= count_disable_tx +1;
    if(count_disable_tx == 10)
    begin
        count_disable_tx <= 0;
        tx_enable <= 0;
    end
end
```

Abbildung 45: Schaltet den UART-Sender ab

```

//RX

always @ (posedge rxclk or posedge tile0_tx_system_reset0_c)
begin
    if (tile0_tx_system_reset0_c)
        begin
            rx_reg          <= 0;
            rx_data         <= 0;
            rx_sample_cnt <= 0;
            rx_cnt          <= 0;
            rx_frame_err   <= 0;
            rx_over_run    <= 0;
            rx_empty       <= 1;
            rx_d1          <= 1;
            rx_d2          <= 1;
            rx_busy        <= 0;
        end
    else
        begin
            // Synchronize the asynch signal
            rx_d1 <= RX_UART;
            rx_d2 <= rx_d1;
            // Uload the rx data
            if (uld_rx_data)
                begin
                    rx_data <= rx_reg;
                    rx_empty <= 1;
                end
            // Receive data only when rx is enabled
            if (rx_enable)
                begin
                    // Check if just received start of frame
                    if (!rx_busy && !rx_d2)
                        begin
                            rx_busy          <= 1;
                            rx_sample_cnt <= 1;
                            rx_cnt          <= 0;
                        end
                    // Start of frame detected, Proceed with rest of data
                    if (rx_busy)
                        begin
                            rx_sample_cnt <= rx_sample_cnt + 1;
                        end
                end
        end
    end
end

```

Abbildung 46: UART-Empfänger oberer Abschnitt

```

// Logic to sample at middle of data
if (rx_sample_cnt == 7)
begin
    if ((rx_d2 == 1) && (rx_cnt == 0))
    begin
        rx_busy <= 0;
    end
    else
    begin
        rx_cnt <= rx_cnt + 1;
        // Start storing the rx data
        if (rx_cnt > 0 && rx_cnt < 9)
        begin
            rx_reg[rx_cnt - 1] <= rx_d2;
        end
        if (rx_cnt == 9)
        begin
            rx_busy <= 0;
            // Check if End of frame received correctly
            if (rx_d2 == 0)
            begin
                rx_frame_err <= 1;
            end
            else
            begin
                rx_empty <= 0;
                rx_frame_err <= 0;
                // Check if last rx data was not unloaded,
                rx_over_run <= (rx_empty) ? 0 : 1;
            end
        end
    end
end
end
end
end
end
if (!rx_enable)
begin
    rx_busy <= 0;
end

```

Abbildung 47: UART-Empfänger unterer Abschnitt

Da nun der UART-Sender genauer beschrieben wurde, wird nun der UART-Empfänger etwas genauer beleuchtet. Die Abbildung 46 zeigt den Beginn des UART-Empfängers, auch dieser setzt zuerst alle Register auf „IDLE“. Der Abschnitt danach ist nur zum Synchronisieren der Daten und zum Überprüfen ob die vorher empfangenen Daten bereits aus dem „Buffer“ geladen wurden.

Anschließend beginnt der Abschnitt der die Daten empfängt. Zuerst wird überprüft ob der UART-Empfänger empfangen darf, danach wird auf ein Startbit gewartet. Ist dieses Startbit empfangen worden, beginnt der UART-Empfänger mit dem empfangen der Bits. Dieser Vorgang ist in Abbildung 47 zu sehen. Dort wird zuerst auf die Mitte des empfangenen Bits gesampelt. Danach werden alle 8 Datenbits und das Stoppbit empfangen, Das Stoppbit wird zum Schluss überprüft und falls es nicht einer 0 entspricht, wird ein Fehler ausgegeben.

Die Abbildung 48 zeigt den Codeabschnitt der überprüft ob ein neues Symbol am UART-Empfänger empfangen wurde und falls dies der Fall ist, wird ein Register gesetzt damit der „Buffer“ entleert wird und weiter empfangen kann.

```

:
//check if new character was received

if(rx_empty)
begin
    uld_rx_data <= 0;
    if(tx_delay_enable)
        begin
            rx_new_data <= ~ rx_new_data;
        end
    end
else
begin
    uld_rx_data <= 1;
    tx_delay_enable <= 1;
end

```

Abbildung 48: Überprüft ob ein neues Symbol am UART-Empfänger empfangen wurde

Um die UART zu testen wurde zunächst das Programm „Realterm“ installiert, dieses bietet eine deutlich bessere Übersicht als das bekannte „Hyperterminal“ und mehr Darstellungsformen der gesendeten bzw. empfangenen Daten. Neben ASCII, kann Realterm auch binär und hexadezimal darstellen, dies hat die Fehleranalyse deutlich vereinfacht. Bei Display wählt man die Darstellungsform, es besteht die Möglichkeit sich mit dem Haken bei „Half Duplex“ die gesendeten Daten anzeigen zu lassen. Beim Reiter Port stellt man die Kenndaten für die UART ein, sucht das entsprechende Board und baut eine Verbindung auf. Nun kann man über die Tastatur Daten senden. Eine kurze Übersicht über das Programm erhält man in Abbildung 49. Um nun die UART zu testen werden die empfangenen Daten vom FPGA einfach an den PC zurückgesendet und nebenher über einen SMA Ausgang am FPGA mittels Oszilloskop verfolgt.

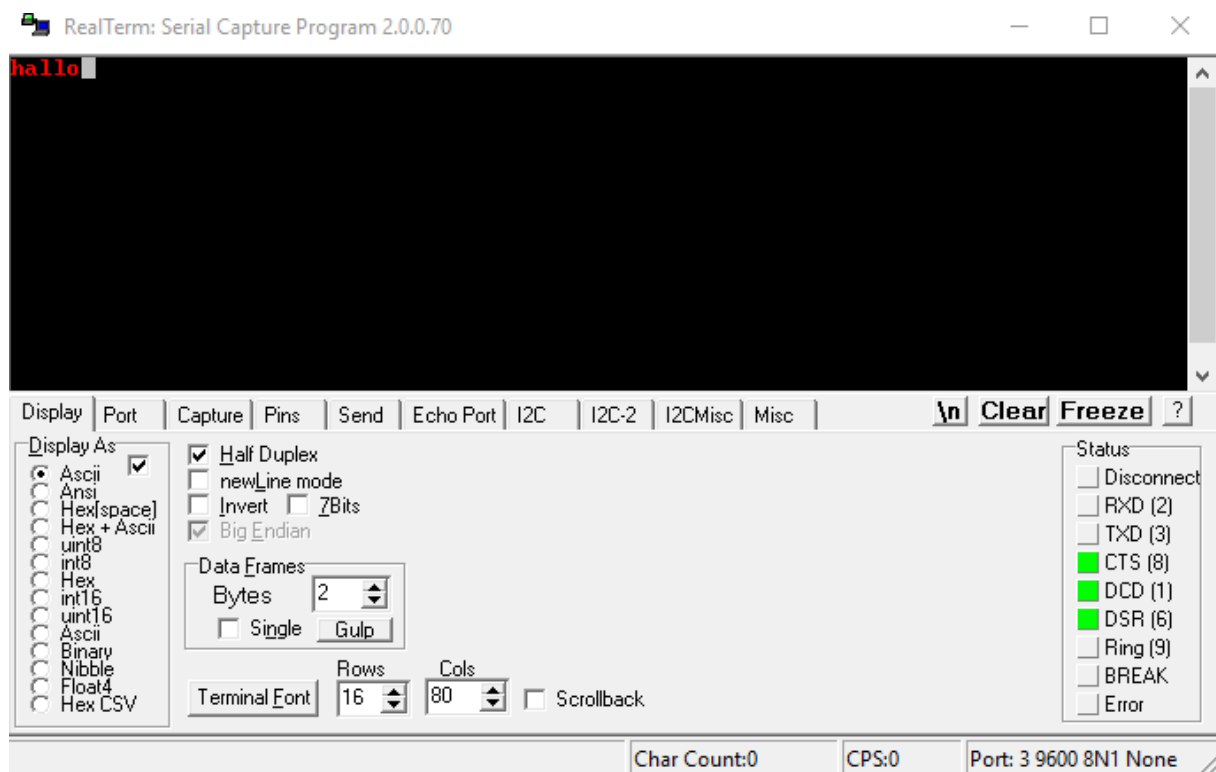


Abbildung 49: Realterm Übersicht

Der erfolgreiche Abschluss dieser Tests, führte nun zum nächsten Teil der UART Testung, dem eigenständigen senden von Daten. Dazu wurde das Programm so umgeschrieben, um erhaltene Symbole an das Transceiver Modul weiter zu leiteten. Dieses Modul, nach erhalten des Symbols, eine Nachricht über den Transceiver schickt und diese zur Überprüfung ebenfalls an die UART weiterleitet, damit diese es zur Überprüfung an den PC schicken kann. Dieser Test hatte vor allem den Zweck zu überprüfen ob, gesendete Symbole korrekt an den Transceiver übergeben werden. Die Testergebnisse der UART werden in der Ergebnis Sektion beschrieben.

4.3. Transceiver

In diesem Abschnitt wird der implementierte Transceiver genauer erläutert. Dazu werden wichtige Codefragmente und Hardwarekomponenten betrachtet. Der Transceiver selbst wird primär vom Framework angesprochen, siehe Abbildung 39, Welches Testdaten vom „frame_gen“ zum GTPA1_DUAL schickt, dies ist eine Stammfunktion des FPGAs und das Bindeglied zwischen Software und Hardware. Diese Funktion leitet die Daten direkt auf den FPGA Transceiver weiter und stellt alle Parameter ein. Die Abbildung 50 zeigt wie die Stammfunktion mit dem gewählten Transceiver Paar verbunden ist.

```
##----- Set placement for tile0_rocketio_wrapper_i/GTPA1_DUAL -----  
INST "s6_gtpwizard_v1_10_i/tile0_s6_gtpwizard_v1_10_i/gtpal_dual_i" LOC = GTPA1_DUAL_X0Y0;
```

Abbildung 50: Hardwareverbindung mit dem GTPA1_DUAL_X0Y0 Transceiver Paar

Da diese Verbindungen im Framework bereits implementiert sind, handelt sich der nächste Schritt um die Generierung der PPM Daten. Um diese zu erzeugen ist es vorab notwendig eine Codierung festzulegen. Aufgrund der Tatsache, dass nicht alle existierenden Symbole, mittels der 32 PPM übertragbar sind, muss eine Auswahl der wichtigsten Symbole für eine Kommunikation getroffen werden. Die Abbildung 51 zeigt die LUT („Look up table“), für das PPM Signal, mit den ausgewählten Symbolen. Neben dem Alphabet sind die wichtigsten Satzzeichen inklusive Leerzeichen und Zeilenumbruch enthalten. Es sind keine Großbuchstaben möglich, da diese im ASCII Format erneut 26 Symbole benötigen würden und Umlaute wurden genauso ausgeschlossen. Doch für einfache Nachrichten sind die gewählten Symbole vollkommen ausreichend.

Der nächste Schritt des Transceivers besteht darin, die erhaltenen Daten der UART in die PPM Codierung umzuwandeln. Wenn die UART ein neues Symbol empfängt, wird dem Transceiver mitgeteilt, dass Dieser ein neues Symbol senden kann, siehe Abbildung 48. Das empfangene Symbol wird an das frame_gen Modul weitergeleitet und dort in ein PPM Symbol umgewandelt. Dieses wird danach in den RAM des FPGAs geladen.

LUT	
Symbol	PPM Codierung
a	10000000000000000000000000000000
b	01000000000000000000000000000000
c	00100000000000000000000000000000
d	00010000000000000000000000000000
e	00001000000000000000000000000000
f	00000100000000000000000000000000
g	00000010000000000000000000000000
h	00000001000000000000000000000000
i	00000000100000000000000000000000
j	00000000010000000000000000000000
k	00000000001000000000000000000000
l	00000000000100000000000000000000
m	00000000000010000000000000000000
n	00000000000001000000000000000000
o	00000000000000100000000000000000
p	00000000000000010000000000000000
q	00000000000000001000000000000000
r	00000000000000000100000000000000
s	00000000000000000010000000000000
t	00000000000000000001000000000000
u	00000000000000000000100000000000
v	00000000000000000000010000000000
w	00000000000000000000001000000000
x	00000000000000000000000100000000
y	00000000000000000000000010000000
z	00000000000000000000000001000000
\n	00000000000000000000000000100000
	00000000000000000000000000010000
.	00000000000000000000000000001000
!	00000000000000000000000000000100
?	00000000000000000000000000000010
,	00000000000000000000000000000001
default	00000000000000000000000000000000

Abbildung 51: LUT für die PPM Codierung

In Abbildung 52 sind die Eingänge und Ausgänge des RAMs abgebildet. Dieser RAM besteht aus zwei synchronen Ports, A und B. Die einzelnen Ports wiederum besitzen wichtige Datenleitungen die für den Transceiver essentiell sind, um Synchronisationsprobleme mit dem Receiver zu vermeiden. Daher werden die Daten vorher in den RAM geladen und erst mit dem nächsten Clock des Transceivers gesendet. Für den Transceiver selbst, wird aber nur ein Port benötigt.

ADDR ist der Adressbus, dieser bestimmt auf welche Speicherstelle geschrieben wird, in Abbildung 53 sieht man die wechselnden Adressen für die PPM Symbole oder für eine festgelegte Übertragung, auf welche später noch eingegangen wird.

DI wird als Datenbus verwendet. Auf diesen Bus werden die Daten gelegt, welche als nächstes in den RAM geschrieben werden. Dies betrifft das PPM Symbol.

DIP ist ein Datenbus für Paritätsbits, für das PPM-Signal werden Nuller beigefügt.

EN gibt an, ob Daten in den RAM geschrieben werden dürfen, falls dies nicht der Fall ist, verharrt der Datenausgangsbuss im selben Zustand wie er bereits ist. Ein sehr wichtiges Bit, das gesetzt sein muss um überhaupt neue Daten senden zu können. Es wird immer auf eins gesetzt.

WE ist das „write_enable“ Bit und bietet zusätzlich die Möglichkeit vom Schreiben von 8 Bits (1 Byte) in den RAM. Es existieren 4 dieser Bits um den 32 Bit Eingang aufzuteilen. Diese Möglichkeit wird nicht benötigt.

DO ist der Datenausgang, auf ihm liegen die Daten welche an den Transceiver geleitet werden. Er wird auch für den Receiver verwendet, um die empfangenen Daten, mit denen des Transceivers, zu vergleichen und auf Fehler zu überprüfen.

DOP sind die Paritätsbits auf einem eigenen Bus. Diese Bits werden ebenso überprüft ob eine Übereinstimmung mit den Bits von DIP besteht.

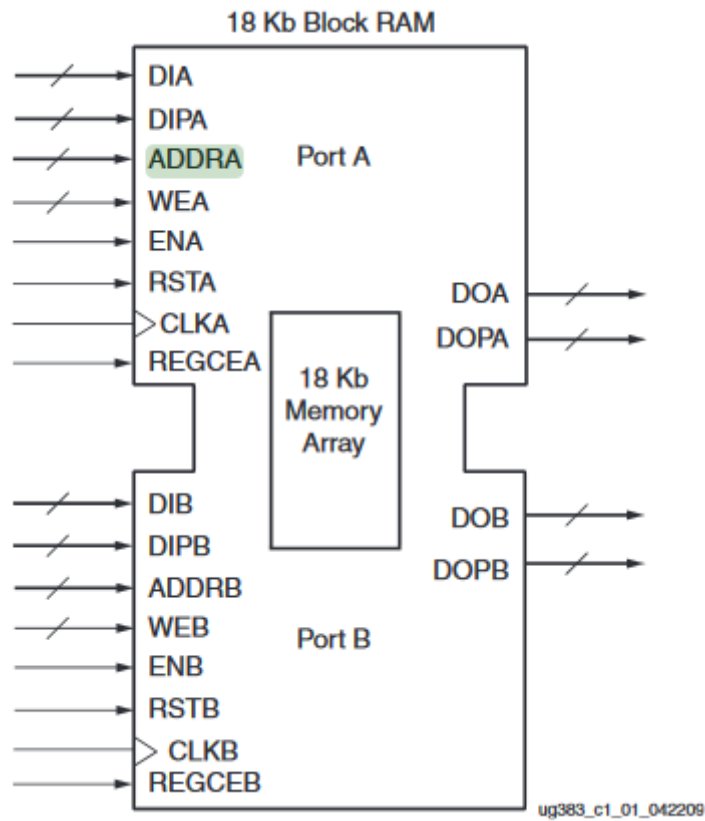


Abbildung 52: Blockschaltbild des Datenflusses für den DUAL-Port RAMs [16]

```
// _____ Counter to read from BRAM _____
always @(posedge USER_CLK)
  if (SYSTEM_RESET | (read_counter_i == (WORDS_IN_BRAM-1)))
    begin
      read_counter_i <= 9'd0;
    end
  else read_counter_i <= read_counter_i + 9'd1;
```

Abbildung 53: Zähler für die verschiedenen Adressen

Der Clock der für CLKA und den Transceiver genutzt wird, ist vom Framework erzeugt worden, siehe Abbildung 54.

```
NET "tile0_txusrclk20_i" TNM_NET = "tile0_txusrclk20_i";  
TIMESPEC TS_tile0_txusrclk20_i = PERIOD "tile0_txusrclk20_i" 14.815 ns;
```

Abbildung 54: Clock für den RAM

Dadurch die UART die limitierende Komponente darstellt, gibt es ebenfalls die Möglichkeit Daten ohne UART-Eingang zu versenden. In diesem Fall wird die Zeile in Abbildung 55 verwendet, mit dieser ist es möglich das volle Potenzial des Transceivers zu nutzen. Da diese Symbole jedoch fest im Code verankert sind, muss man für das Ändern der Symbole den Programmcode selbst verändern. Jedoch ist dies kein großes Problem.

Ein anderer Weg wäre es, statt einer UART, eine andere Verbindung zwischen PC und FPGA zu implementieren, als Beispiel SPI.

```
FRAME_GEN #  
(  
    .WORDS_IN_BRAM(EXAMPLE_WORDS_IN_BRAM),  
    .MEM_00(256'h00000000000000000000000000000000000000000000000000000000000000000000000000000001),
```

Abbildung 55: Festgelegte Übertragung ohne UART-input

4.4. Receiver

Dieses Kapitel behandelt den Receiver des Programmes und die Überprüfung der empfangenen Daten. Dieser Bereich des Programmes befindet sich im File „frame_check“, Abbildung 39 zeigt die Übersicht des Programmes. Das Signal wird direkt von den in Abbildung 24 gezeigten Eingängen in frame_check geführt.

Dort wird als erster Schritt das anliegende Signal in ein 32bit Register geladen. Die Abbildung 56 zeigt dieses wichtige Codesegment.

```

always @(posedge USER_CLK)
    rx_data_r <= RX_DATA;

```

Abbildung 56: Daten am Receiver werden in ein Register geladen

Dieses Register bedarf einer Fehlerüberprüfung, dazu werden die Daten, die der Transceiver in den RAM gespeichert hat, verwendet. Um die richtige Speicheradresse zu finden, sind zwei Wege im Programm implementiert. Wenn der Modus mit der UART verwendet wird, leitet der Transceiver in `frame_gen`, die Adresse des gesendeten Symbols an `frame_check` weiter. Wird jedoch der deutlich schnellere Weg genutzt, liegen die Daten bereits im RAM. In diesem Fall wird der Codeabschnitt in Abbildung 57 verwendet, dieser Zähler geht die Adressen im RAM von vorne bis hinten durch und da der Transceiver mit diesem Clock die Daten sendet, weiß man immer welches Symbol als nächstes am Receiver zu erwarten ist.

```

// _____ Counter to read from BRAM _____
generate
if (RX_DATA_WIDTH > 20)
begin : four_byte
    always @(posedge USER_CLK)
        if (SYSTEM_RESET | (read_counter_i == (WORDS_IN_BRAM-1)))
            begin
                read_counter_i <= 9'd0;
            end
        else if (start_of_packet_detected_r & !track_data_r)
            begin
                read_counter_i <= 9'd1;
            end
        else read_counter_i <= read_counter_i + 9'd1;
    end
else
begin: one_or_two_byte
    always @(posedge USER_CLK)
        if (SYSTEM_RESET | (read_counter_i == (WORDS_IN_BRAM-1)) | (start_of_packet_detected_r & !track_data_r))
            begin
                read_counter_i <= 9'd0;
            end
        else read_counter_i <= read_counter_i + 9'd1;
    end
endgenerate

```

Abbildung 57: Zähler für die richtige Adresse des zu überprüfenden Symbols

Da der RAM für den Receiver nur im „read_mode“ gebraucht wird, müssen die Eingänge und Ausgänge anders belegt werden. Die Abbildung 58 stellt dieses sehr wichtige Codefragment dar. Neben dem Zähler für die richtige Adresse, werden die beiden Eingangsdatenleitungen DIA und DIPA auf 0 gesetzt,

da nur Leseoperation benötigt wird. Ebenfalls wird WEA auf 0 gesetzt, doch ENA wird auf 1 gesetzt, ansonsten würde sich der Datenausgangsbuss nicht ändern. Der wichtigste Teil ist hier der DOA, dieser beinhaltet das Referenzsymbol, mit dem die Überprüfung durchgeführt wird.

```
.ADDRA (read_counter_i),
.DIA (tied_to_ground_vec_i[31:0]),
.DIPA (tied_to_ground_vec_i[3:0]),
.DOA (bram_data_i),
.DOPA ( ),
.WEA (tied_to_ground_i),
.ENA (tied_to_vcc_i),
```

Abbildung 58: RAM Eingänge und Ausgänge für den Receiver

Die Abbildung 59 zeigt die Überprüfung des empfangenen Symbols. Falls bei dieser Überprüfung ein Fehler detektiert wird, setzt sich das „error_detected_c“ Flag auf 1.

```
generate
if (RX_DATA_WIDTH==40)
begin : use_40bit
assign error_detected_c = track_data_r4 && (rx_data_r_track != {tied_to_ground_vec_i[7:0],bram_data_r});
end
else
begin : not_40bit
assign error_detected_c = track_data_r4 && (rx_data_r_track != bram_data_r[(RX_DATA_WIDTH-1):0]);
end
endgenerate
```

Abbildung 59: Überprüfung des empfangenen Symbols

Dieses Flag wird in Abbildung 60 überprüft und ein Zähler wird inkrementiert, um die Anzahl der Fehler zu registrieren. Das Flag wird in einem anderen Fragment wieder auf 0 gesetzt.

```
always @(posedge USER_CLK)
if (SYSTEM_RESET)
error_count_r <= 9'd0;
else if (error_detected_r)
error_count_r <= error_count_r + 1;
```

Abbildung 60: Zähler für die detektierten Fehler

Nach dieser Überprüfung wird das PPM-Symbol zur Decodierung weitergeleitet. Siehe Abbildung 61 , dort erkennbar werden die PPM-Symbole wieder in ASCII Zeichen transformiert.

Es werden jedoch fehlerhafte Symbole zuvor auf 0 gesetzt, damit diese leichter in RealTerm zu erkennen sind. Eine Fehlerkorrektur wäre sehr einfach, da fehlerhafte Symbole einfach durch ihr zu überprüfendes Symbols ersetzt würden. Dies wird aber nicht durchgeführt, um Fehler zu verdeutlichen.

```

always @(posedge USER_CLK)
begin
    case (rx_data_r_d)
        32'b10000000000000000000000000000000: rx_data_decode = 8'h61;
        32'b01000000000000000000000000000000: rx_data_decode = 8'h62;
        32'b00100000000000000000000000000000: rx_data_decode = 8'h63;
        32'b00010000000000000000000000000000: rx_data_decode = 8'h64;
        32'b00001000000000000000000000000000: rx_data_decode = 8'h65;
        32'b00000100000000000000000000000000: rx_data_decode = 8'h66;
        32'b00000010000000000000000000000000: rx_data_decode = 8'h67;
        32'b00000001000000000000000000000000: rx_data_decode = 8'h68;
        32'b00000000100000000000000000000000: rx_data_decode = 8'h69;
        32'b00000000010000000000000000000000: rx_data_decode = 8'h6A;
        32'b00000000001000000000000000000000: rx_data_decode = 8'h6B;
        32'b00000000000100000000000000000000: rx_data_decode = 8'h6C;
        32'b00000000000010000000000000000000: rx_data_decode = 8'h6D;
        32'b00000000000001000000000000000000: rx_data_decode = 8'h6E;
        32'b00000000000000100000000000000000: rx_data_decode = 8'h6F;
        32'b00000000000000010000000000000000: rx_data_decode = 8'h70;
        32'b00000000000000001000000000000000: rx_data_decode = 8'h71;
        32'b00000000000000000100000000000000: rx_data_decode = 8'h72;
        32'b00000000000000000010000000000000: rx_data_decode = 8'h73;
        32'b00000000000000000001000000000000: rx_data_decode = 8'h74;
        32'b00000000000000000000100000000000: rx_data_decode = 8'h75;
        32'b00000000000000000000010000000000: rx_data_decode = 8'h76;
        32'b00000000000000000000001000000000: rx_data_decode = 8'h77;
        32'b00000000000000000000000100000000: rx_data_decode = 8'h78;
        32'b00000000000000000000000010000000: rx_data_decode = 8'h79;
        32'b00000000000000000000000001000000: rx_data_decode = 8'h7A;
        32'b00000000000000000000000000100000: rx_data_decode = 8'h0D; //new line
        32'b00000000000000000000000000010000: rx_data_decode = 8'h20;
        32'b000000000000000000000000000001000: rx_data_decode = 8'h2E;
        32'b0000000000000000000000000000000100: rx_data_decode = 8'h21;
        32'b00000000000000000000000000000000010: rx_data_decode = 8'h3F;
        32'b00000000000000000000000000000000001: rx_data_decode = 8'h2C;
        32'b00000000000000000000000000000000000: /*$display("default")*/ rx_data_decode = 8'b0;
        default: /*$display("EOF or anything else")*/ rx_data_decode = 8'b0;
    endcase
end

```

Abbildung 61: Decodierung der PPM-Symbole

Nach der Decodierung werden die Symbole an den Sender, der UART, geleitet und von Diesem an den PC übertragen. Dort können die Daten dann verifiziert werden, ob diese tatsächlich korrekt empfangen und überprüft wurden.

4.5. Transfer des Programms auf den FPGA

In diesem Kapitel wird beschrieben, wie das Programm auf den FPGA übertragen wird. Bevor mit diesem Vorgang begonnen werden kann, muss der FPGA eingeschaltet und das JTAG Port, siehe Abbildung 26, mit dem PC verbunden sein.

Sind diese Vorbereitungen passiert, wird in den ISE Project Navigator gewechselt, indem das FPGA Projekt geöffnet ist. Dort wird das Top Modul des Projekts ausgewählt, wie es in Abbildung 62 dargestellt ist.

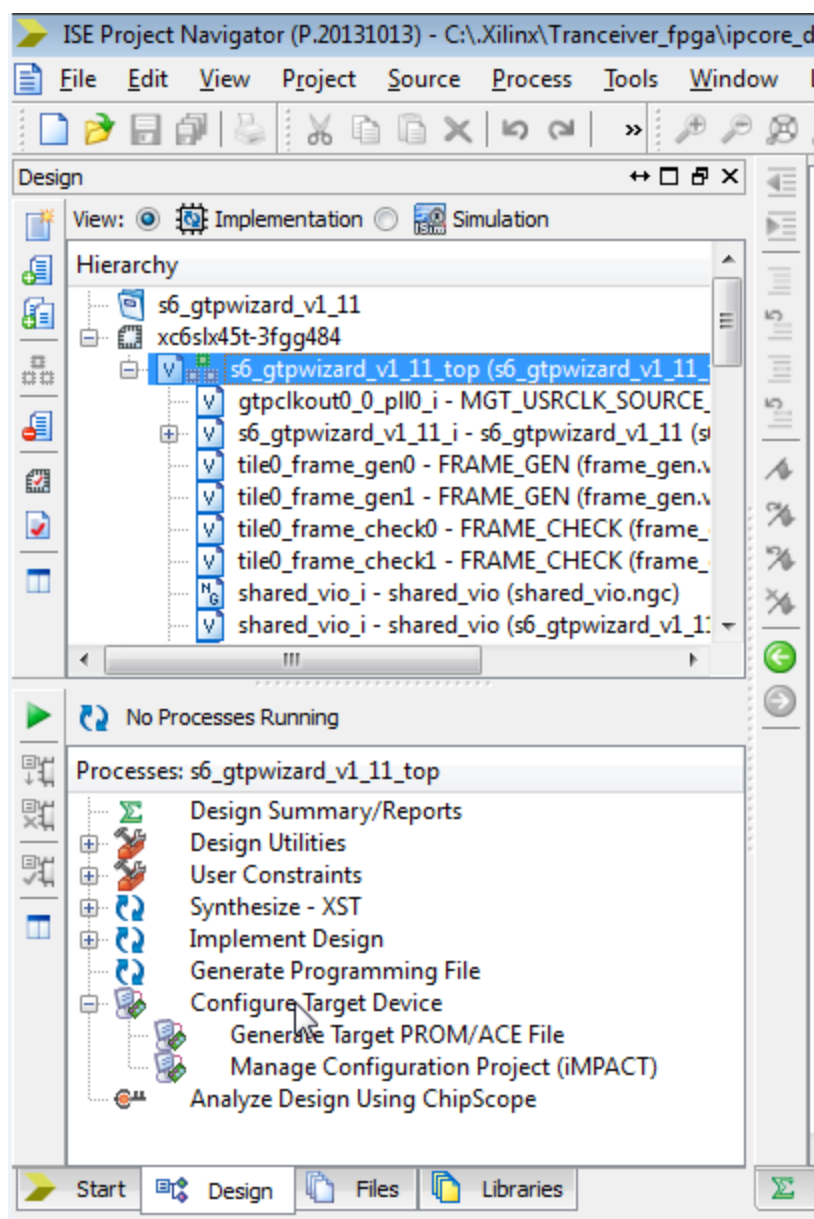


Abbildung 62: Auswahl des Top Moduls im ISE Project Navigator

Danach muss ein Konfigurationsfile für das Zielsystem erstellt werden, dies wird erreicht indem man den „Configure Target Device“ Vorgang startet. Dieser überprüft vorher ob Fehler im Programm vorliegen und startet danach den „ISE iMPACT“. Das erscheinende Fenster dient der Mitteilung, dass kein Konfigurationsfile für das Zielsystem vorhanden ist und dass eines definiert werden muss. Mit dem Klick auf „OK“ startet der „ISE iMPACT“.

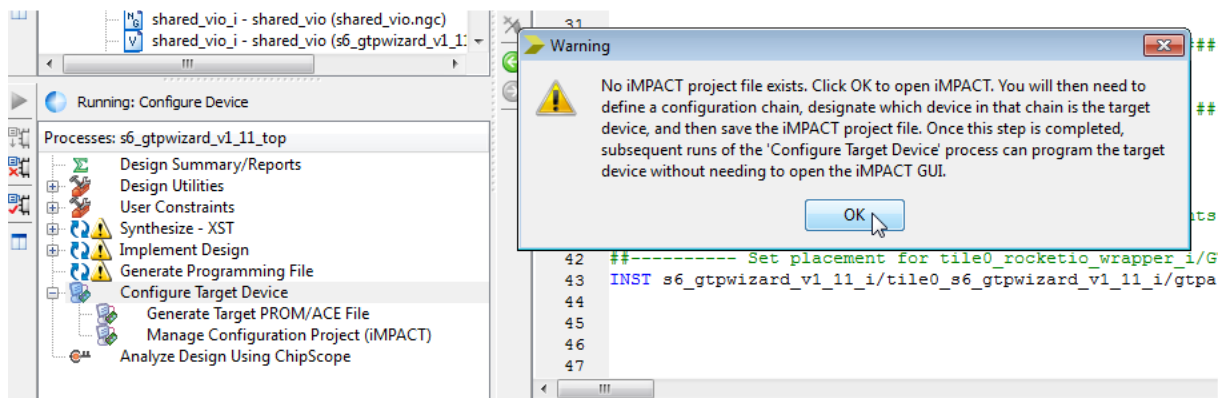


Abbildung 63: Starten des „Configure Target Device“ zum Erzeugen eines Konfigurationsfiles für den FPGA

Im „ISE iMPACT“ wird zuerst nach angeschlossener Hardware gescannt, dies wird mittels „Boundary Scan“ eingeleitet. Der FPGA wird damit gefunden, ist aber noch nicht verbunden mit dem „ISE iMPACT“.

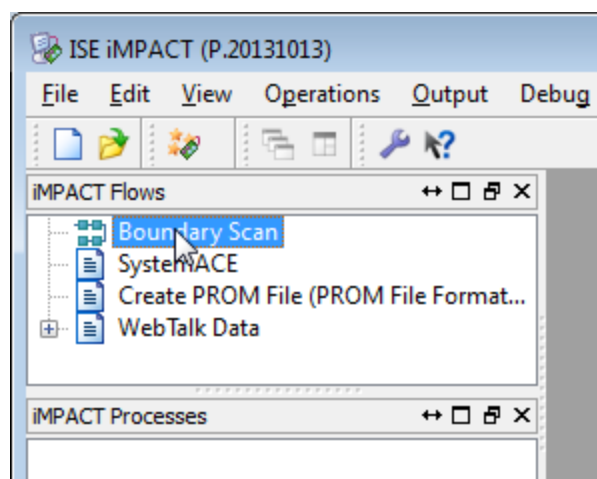


Abbildung 64: Boundary Scan

Um die Verbindung herzustellen wird in der Menüleiste „Output → Cable Auto Connect“ ausgewählt. Ist der Verbindungsaufbau erfolgreich, wird in der Konsole neben einigen Daten auch der Satz „Cable connection established“ ausgegeben. Sollte dies nicht der Fall sein, müssen die Stromversorgung des FPGA bzw. ob dieser eingeschaltet ist, überprüft werden. Auch ob das JTAG Port wirklich mit dem PC verbunden ist.

Wenn die Verbindung hergestellt ist, folgt der nächste Schritt, die Erzeugung des Konfigurationsfiles. Dies wird, wie in Abbildung 65, mit einem Klick auf „Initialize Chain“ gestartet.

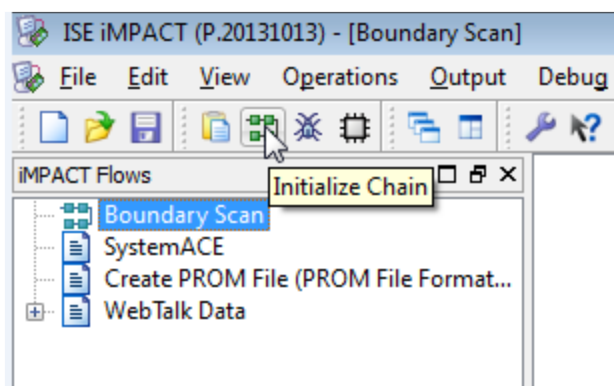


Abbildung 65: Initialize Chain

Es öffnet sich ein Fenster (Abbildung 66), mit der Option, ob man das erstellte Konfigurationsfile hinzufügen möchte, mit „YES“ bestätigen und der Dateibrowser öffnet sich. Um die Sichtbarkeit von Binärdateien sichergestellt zu stellen, wird ein Klick auf „Bypass“ gesetzt.

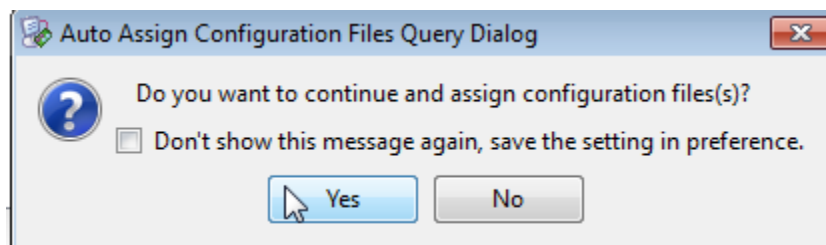


Abbildung 66: Fenster ob man das Konfigurationsfile hinzufügen möchte

Man navigiert nun zum Projektverzeichnis und wählt die dortige Binärdatei mit der Endung „.bit“ aus und bestätigt dies mit „OK“.

Nach dem hinzufügen der Binärdatei erscheint ein Fenster, welches die Information enthält, dass der FPGA einen „Flash PROM“ besitzt und ob man SPI oder BPI PROM hinzufügen möchte. Diese Funktion wird nicht benötigt und das Fenster wird mit „No“ geschlossen, siehe Abbildung 67.

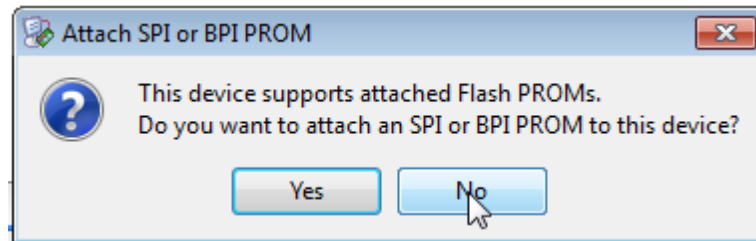


Abbildung 67: Fenster ob SPI oder BPI PROM hinzugefügt werden sollen

Ein weiteres Fenster öffnet sich, in dem man das Zielsystem auswählen muss. Wie in Abbildung 68 zu sehen ist, wird der FPGA als „Device 2“ erkannt. Man wählt dieses Device aus und bestätigt mit „Apply“ und schließt das Fenster mit „OK“.

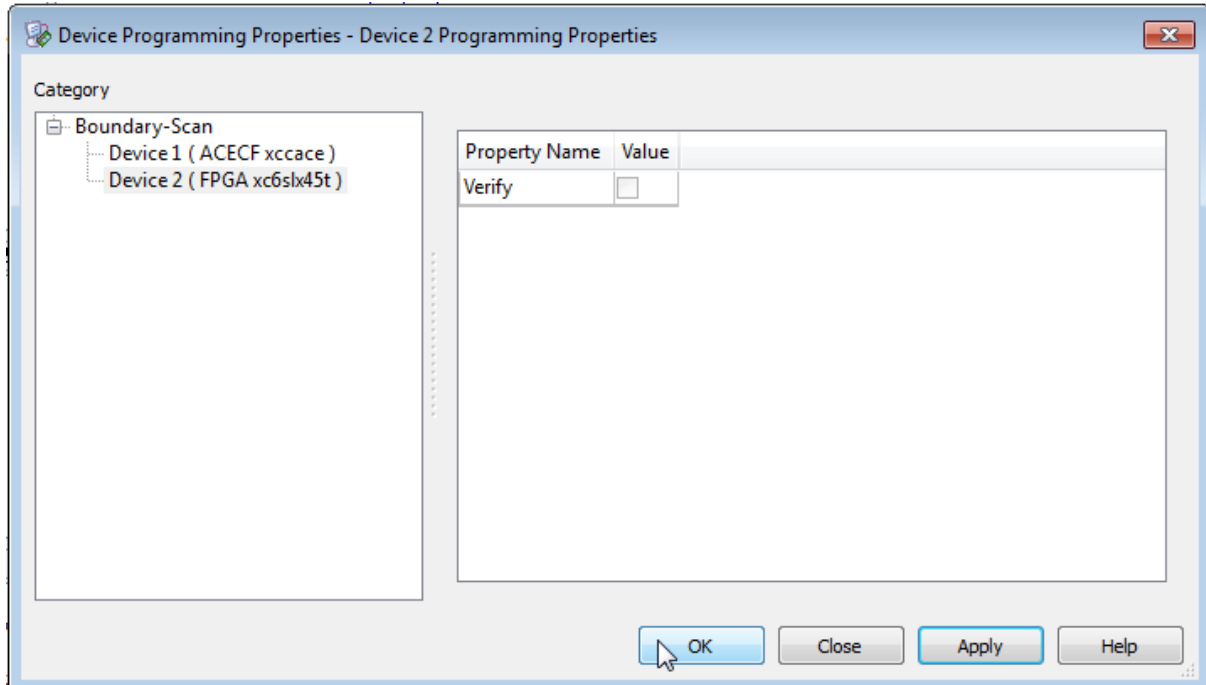


Abbildung 68: Auswahl des Zielsystems

Die Konfiguration ist soweit abgeschlossen, dass es möglich ist, das Programm auf den FPGA zu spielen. Der letzte Schritt um das Programm auf den FPGA zu übertragen, besteht darin, in der Menüleiste “Operations → Program” den Programmiervorgang zu starten, wie in Abbildung 69 dargestellt.

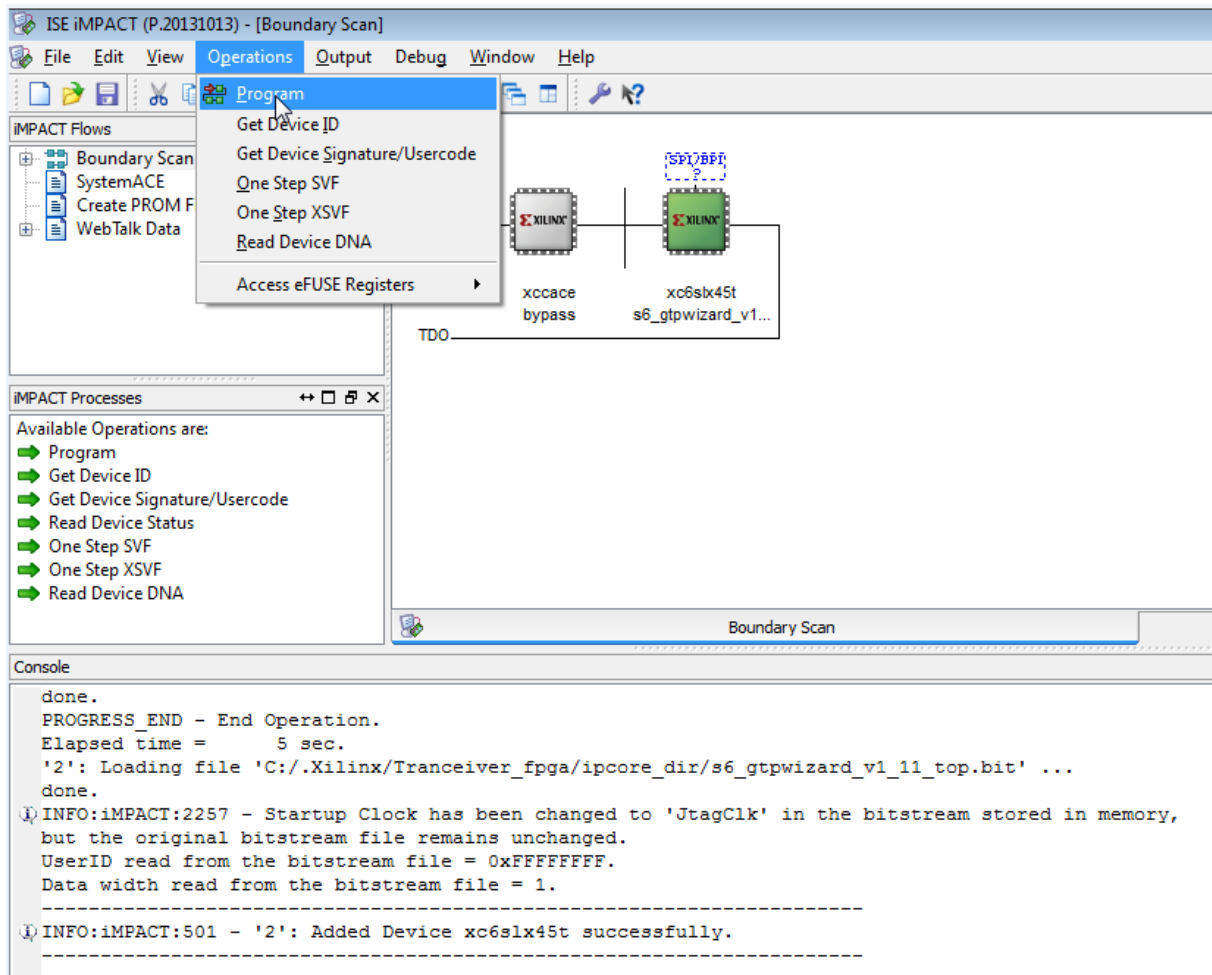


Abbildung 69: Übertragen des Programms auf den FPGA

Nach einer erfolgreichen Übertragung erscheint am Bildschirm “Program Succeeded”, wie es in Abbildung 70 zu sehen ist. Der FPGA resettet sich nun und beginnt mit der Ausführung des Programms.

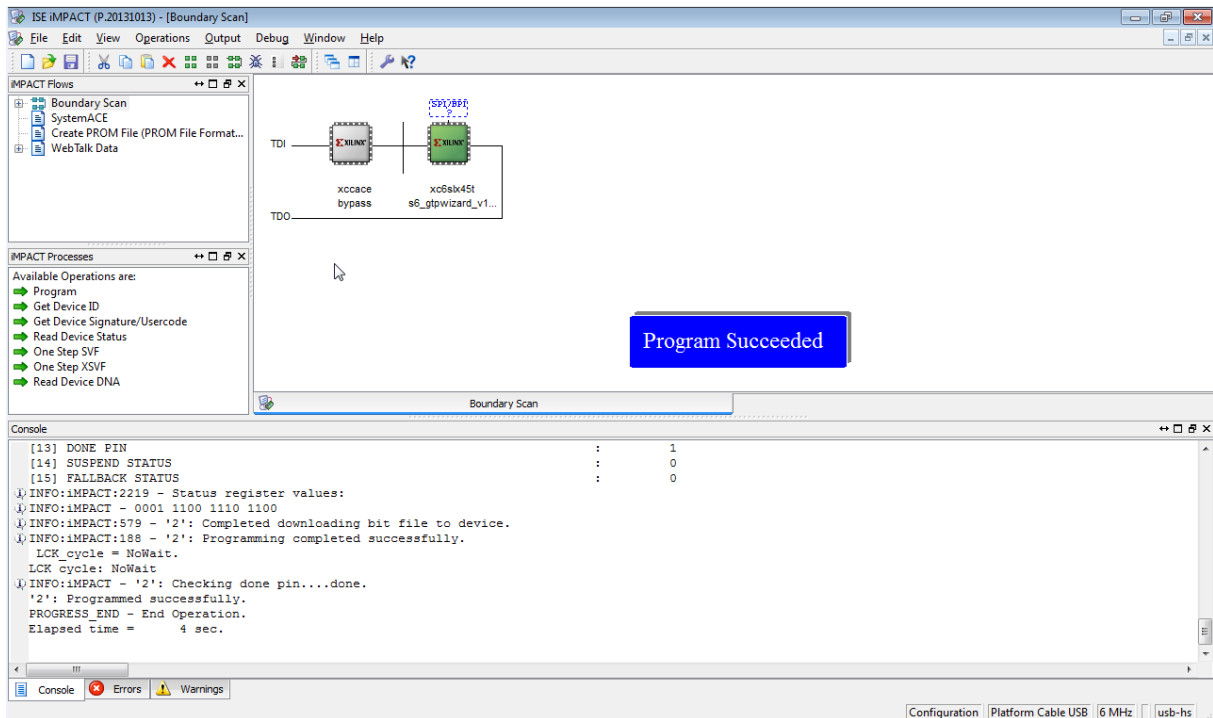


Abbildung 70: Erfolgreiche Übertragung des Programms auf den FPGA

Wenn der FPGA mit dem laufenden Programm resettet oder ausgeschaltet wird, sind diese Schritte erneut nötig. Da nach einem Reset die vorhandenen Beispielprogramme am Flashspeicher des FPGA geladen werden. Diese sollten nicht überschrieben werden, da sie nicht nur einen Einstieg in den FPGA bieten, sondern auch alle Funktionalitäten überprüfen.

5. Ergebnisse

In diesem Kapitel werden alle Messungen, Tests und Ergebnisse genau behandelt und beschrieben. In folgender Reihenfolge, zuerst die Tests der UART, danach Die mit dem Programm und zum Schluss die Tests mit dem PPM Signales.

5.1. Tests und Ergebnisse der UART

Bevor die UART selbst getestet wird, muss zuvor überprüft werden ob die beiden erzeugten Taktsignale auch korrekt sind. Diese Überprüfung wird mittels Oszilloskop erstellt.

Der Messaufbau für die Messung ist in Abbildung 71 dargestellt, es wird der SMA Testausgang in Abbildung 25 verwendet. Dieser Ausgang (J39) wird mit einem Oszilloskop verbunden, damit dort etwas zu erkennen ist, werden die Taktsignale im Programm zusätzlich auf diesen Ausgang gelegt.

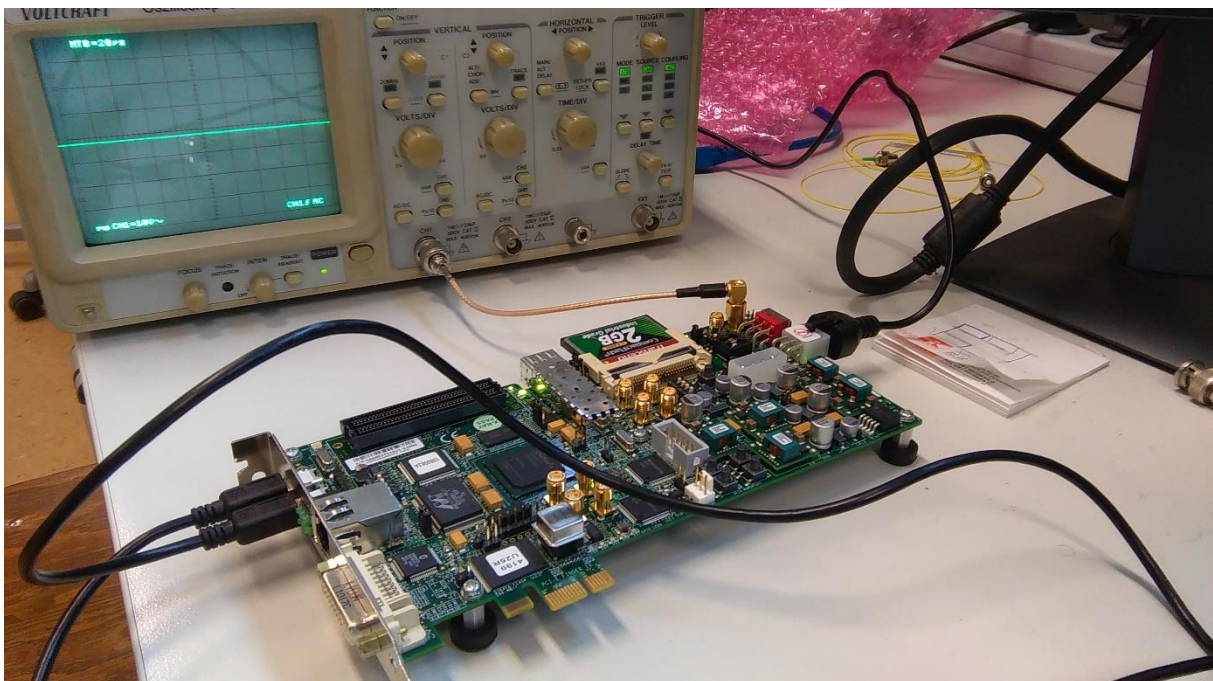


Abbildung 71: Messaufbau für die Messung der Taktsignale

In Abbildung 72 ist das Taktsignal für den Sender der UART zu sehen. Man erkennt dort eine „Time per Division“ von 20µs und eine „Voltage per Division“ von 10V. Da eine positive und negative Flanke genau 5,1 Divisions lang ist, lässt sich daraus eine Periodendauer von 104µs berechnen und diese ist genau zu erreichen, bei einer Baudrate von 9600. Auch die Vpp passt in die Rahmenbedingungen einer UART, sie beträgt hier zirka 25V. Somit ist jede Flanke zirka 12,5V hoch.

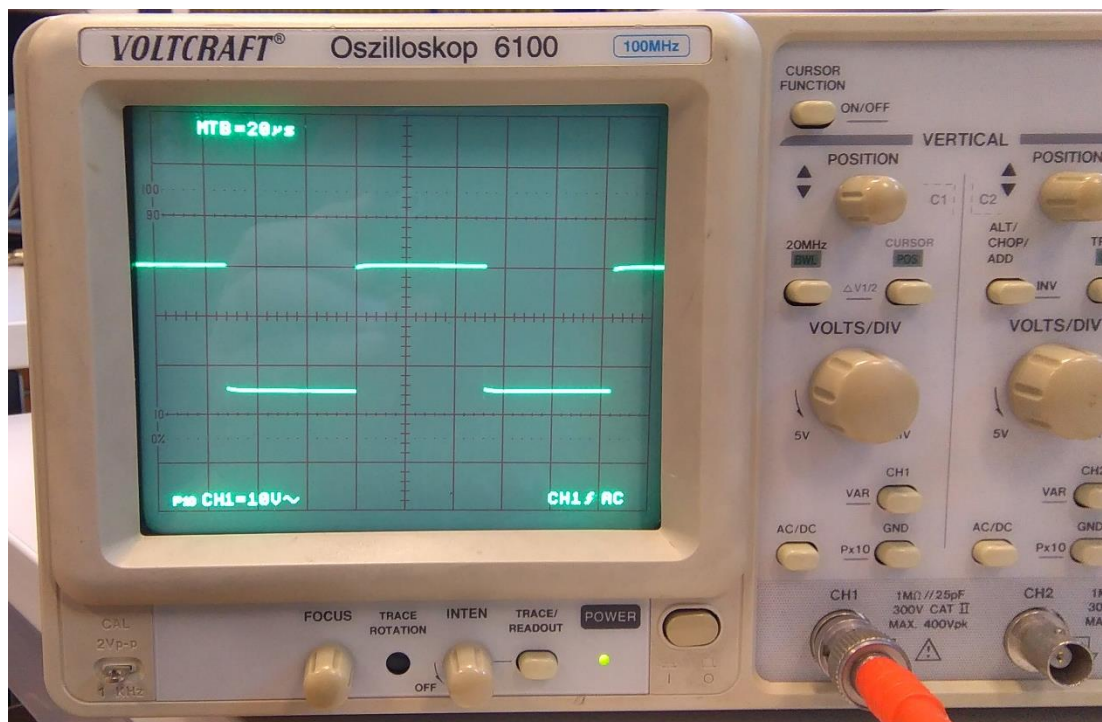


Abbildung 72: UART Sender Taktsignal

Das zweite Taktsignal, des Empfängers, ist in Abbildung 73 dargestellt. Man erkennt mit einem Blick das dieses Signal deutlich schneller ist, als das Sender Signal in Abbildung 72. Da das Empfängersignal 16mal schneller sein soll, als das Sendertaktsignal, muss eine bessere Darstellung eingestellt werden.

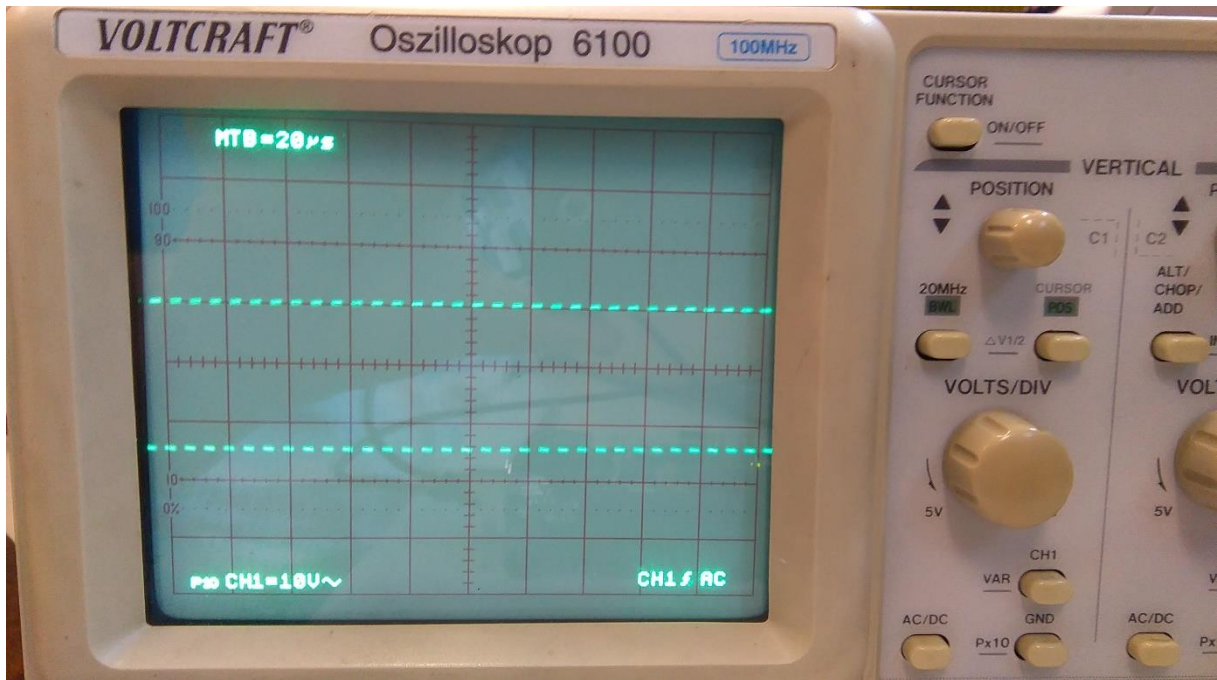


Abbildung 73: Taktsignal des UART-Empfängers



Abbildung 74: Taktsignal des UART-Empfängers mit optimalen Einstellungen

Die Abbildung 74 zeigt nun eine bessere Darstellung des Empfängersignals, zu erkennen durch eine höhere Einstellung der „Time per Division“, auf 2ys. Für die neue Einstellung beträgt eine positive und negative Flanke genau 3,25 Divisions, dies entspricht 6,5ys. Da der Empfänger 16mal schneller, als der Sender sein soll, muss man 104ys nur durch 16 teilen und erhält 6,5ys. Es entspricht genau der gemessenen Periodendauer des Empfängersignals. Da feststeht, dass die beiden Taktsignale korrekt sind, können nun die richtigen Tests an der UART beginnen.

Der erste Test der UART besteht darin zu testen ob die Daten den FPGA erreichen und zurückkommen, zu diesem Zweck wurde im Code die Empfängerleitung direkt auf die Sendeleitung gelegt. Dies bedeutet, das Sender- und Empfänger der UART, die im Code implementiert sind, nicht verwendet werden, sondern das Signal nur durchgereicht wird. Senden und empfangen liegt alleine beim Programm RealTerm in diesem Test. Der Zweck für diesen Test ist es herauszufinden, ob die Hardwareports des FPGAs korrekt verbunden sind.

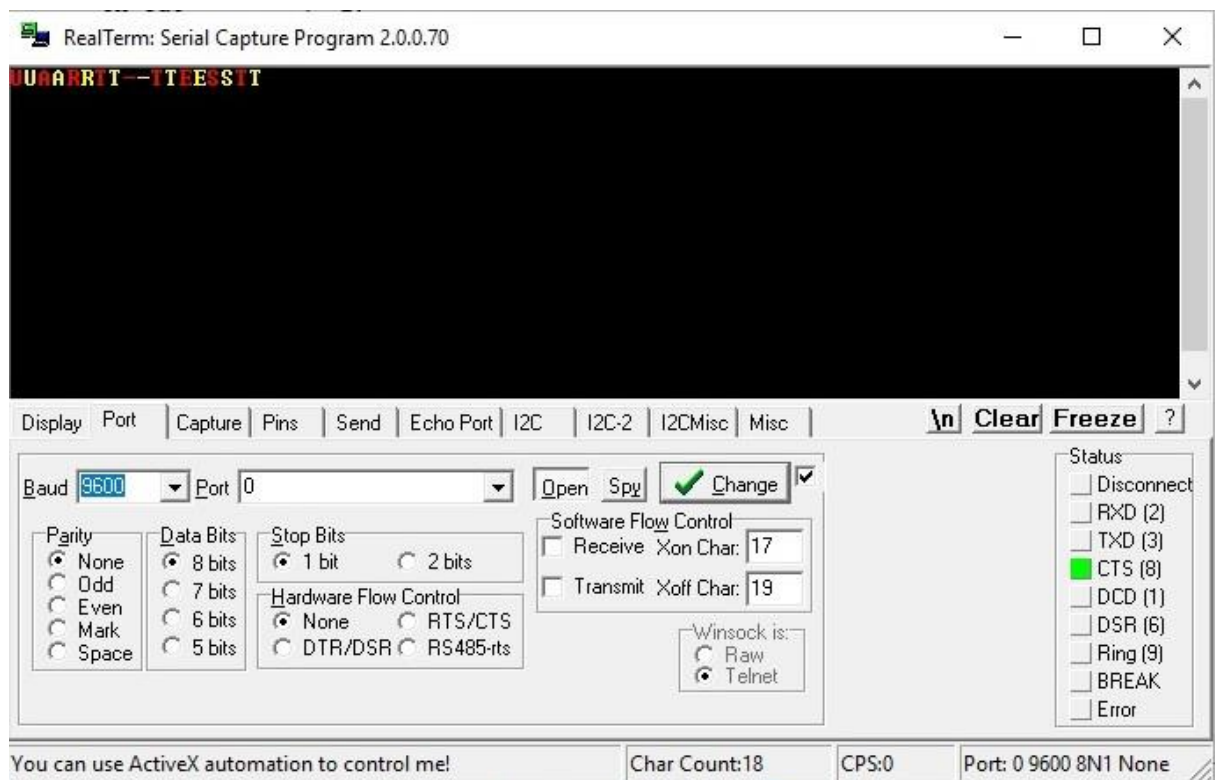


Abbildung 75: Erster UART-Test mit einfacher Durchreicheinstellung

Das Ergebnis dieses Tests ist in Abbildung 75 dargestellt. Die roten Buchstaben zeigen dabei die gesendeten Symbole und die gelben die Empfangenen. Es ist eindeutig zu erkennen, dass alle gesendeten Symbole wieder empfangen wurden. Damit kann festgehalten werden, dass alle Hardwareports für die UART korrekt implementiert sind. Der nächste Test beschäftigt sich mit der tatsächlich implementierten UART.

Der zweite und letzte Test der UART testet die in Abbildung 43, Abbildung 46 und Abbildung 47 dargestellten Codesegmente. Für diesen Test werden die empfangenen Symbole am FPGA an den UART-Sender des FPGA direkt weitergeleitet. Dies heißt von RealTerm gesendete Symbole werden am FPGA empfangen und von Diesem direkt wieder an RealTerm gesendet. Durch diesen Test lässt sich sagen ob die implementierte UART funktioniert.

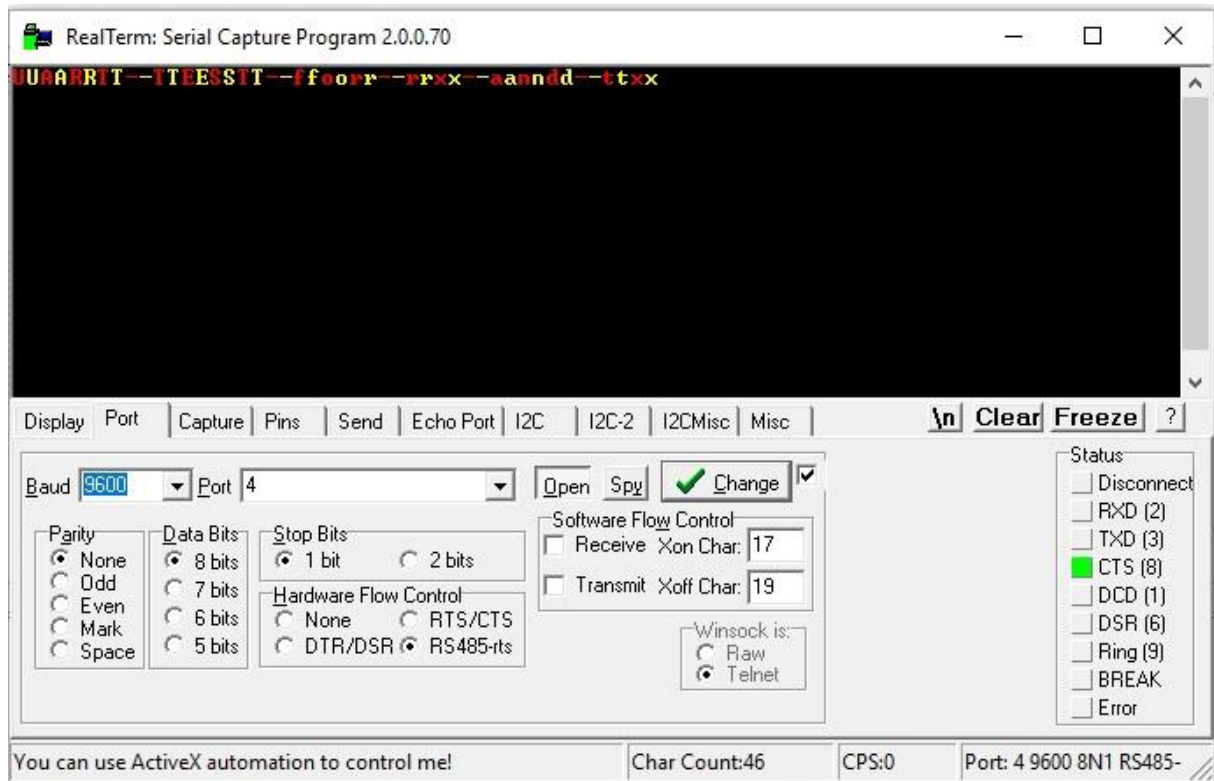


Abbildung 76: Testergebnis des zweiten UART Tests

Das Ergebnis in Abbildung 76, weist auf einen erfolgreichen Test hin, dass die UART damit voll funktionsfähig ist. Viele weitere Tests bauen auf die UART auf. Neben der Funktion, dynamische Daten zu versenden, bietet die UART eine der wenigen Möglichkeiten, zu prüfen, ob die korrekten Symbole, vom Receiver empfangen werden. Ebenfalls eine weitere Anwendung der UART, besteht darin, die empfangenen Symbole grafisch darzustellen. Doch diese Tests und deren Ergebnisse werden im nächsten Abschnitt behandelt.

5.2. Ergebnisse der Tests für das Programm

Dieser Abschnitt beschreibt die Abfolge der Tests und Ergebnisse für das Programm, welches in vorherigen Kapitel genauer beschrieben ist.

Der erste Test stellt dabei fest, ob der Transceiver funktioniert, dazu wird das Symbol welches gerade gesendet wird, zusätzlich in ein Register gespeichert, dort decodiert und anschließend über die UART auf den PC übertragen. Für diese Messung wird der in Abbildung 71 dargestellte Messaufbau verwendet. Neben der UART wird das Symbol ebenfalls auf dem Oszilloskop abgefangen. Damit ein gutes Bild am Oszilloskop entsteht wird das Symbol „h“ dauerhaft vom PC an den FPGA gesendet und dort an den Transceiver geleitet. Ohne diesen Vorgang würde kein klares Bild entstehen, da für die Messung ein analoges Oszilloskop verwendet wird.

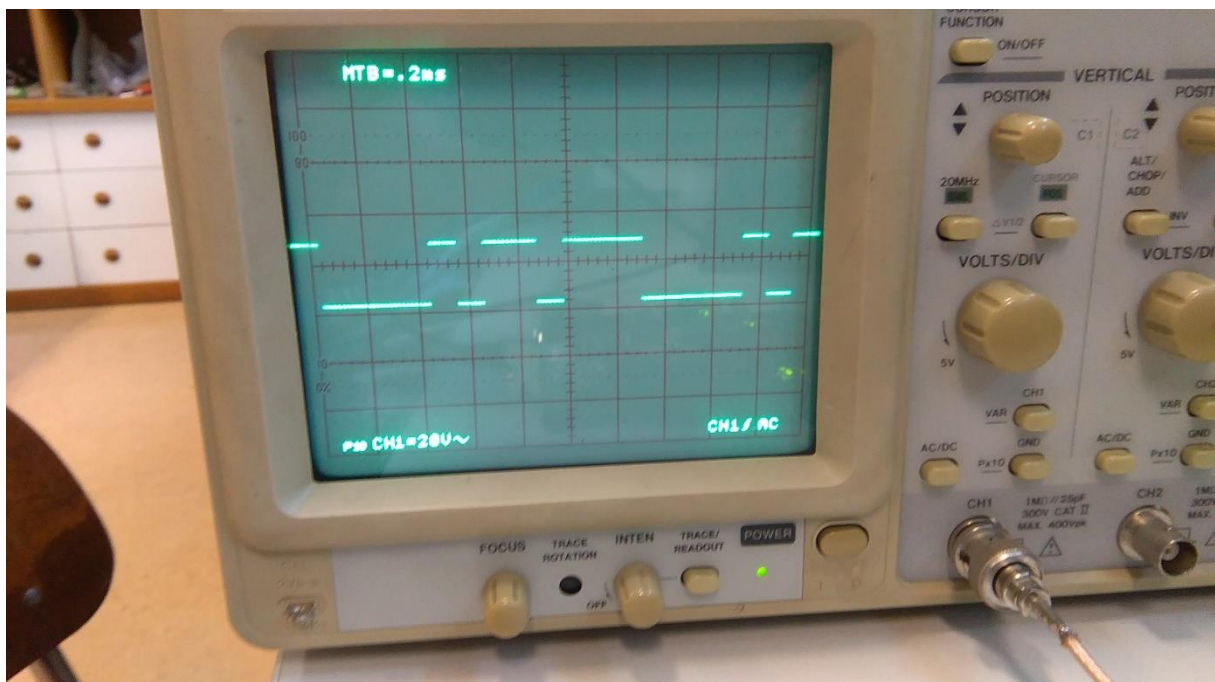


Abbildung 77: Signal des Buchstaben h am Oszilloskop

Dieser Test sollte nur testen ob der Transceiver und die UART richtig miteinander verbunden sind und ob das PPM-Signal korrekt an den Transceiver Ausgang des FPGA weitergeleitet wird. Man kann in Abbildung 77 das Bitmuster erkennen und damit ist gezeigt das der Transceiver, vor allem aber die PPM Codierung und Decodierung korrekt implementiert sind.

Die nächste Messung sollte nun überprüfen, wie das Signal am Transceiver Ausgang aussieht. Dazu wird der Differenzielle Ausgang, wie er in Abbildung 24 zu sehen ist, mit einem Oszilloskop verbunden. Die Messung des PPM-Signals ist in Abbildung 78 abgebildet. Die beiden differenziellen Ausgänge sind mittels der Math-Funktion im unteren Teil von Abbildung 78 übereinander gelegt. Man erkennt sehr schön die einzelnen Piken des PPM-Signal. Daraus ergibt sich, das der Transceiver korrekt arbeitet und somit ein Test mit dem Receiver folgen kann.

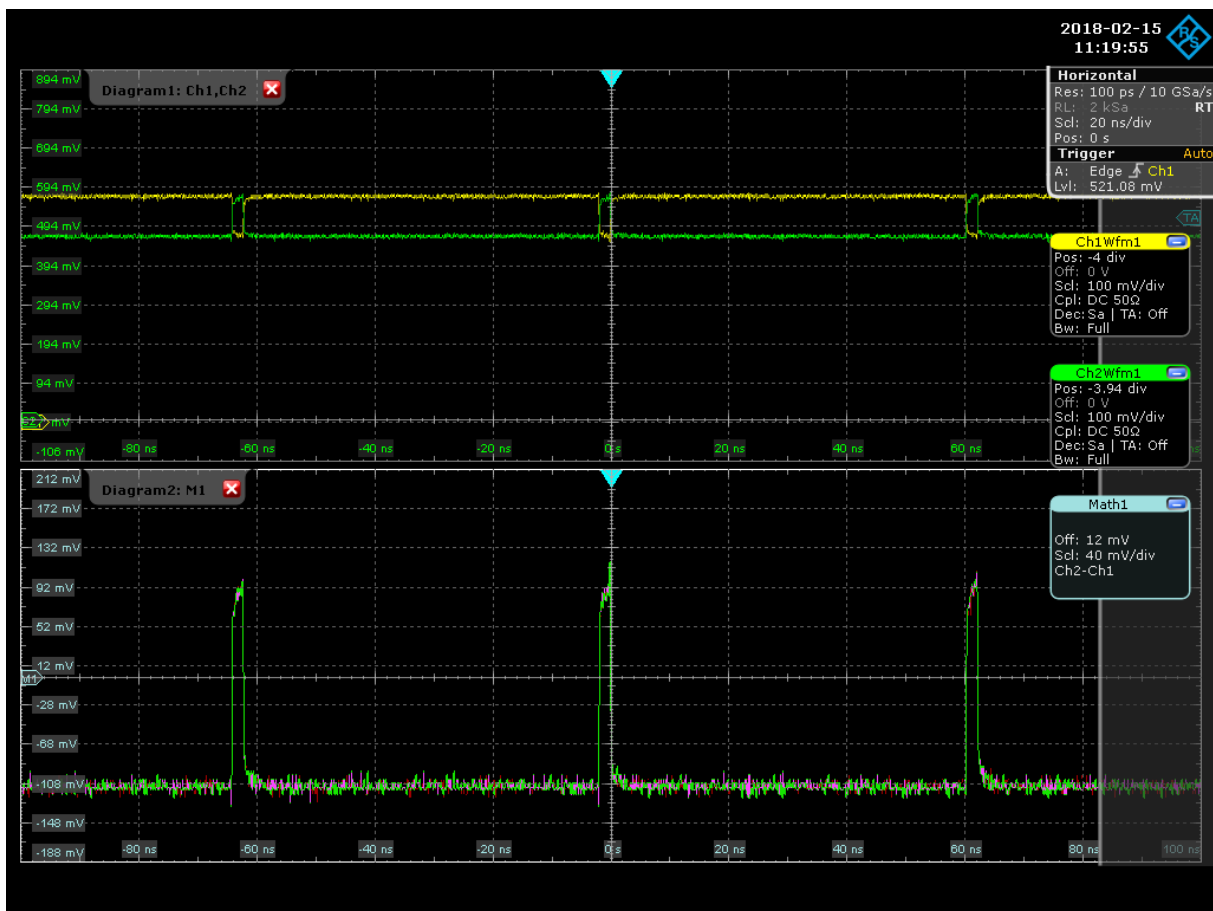


Abbildung 78: PPM-Signal am Transceiver Ausgang

Doch bevor dieser Test mit dem Receiver beschrieben wird, zeigt Abbildung 79 und Abbildung 80, das Spektrum des PPM-Signals am Transceiver Ausgang. Dazu ist festzuhalten, das in Abbildung 80 eine längere Folge des PPM-Signals betrachtet wird. Man erkennt sehr gut die Hauptkeule und die Nebenkeulen des Signals.

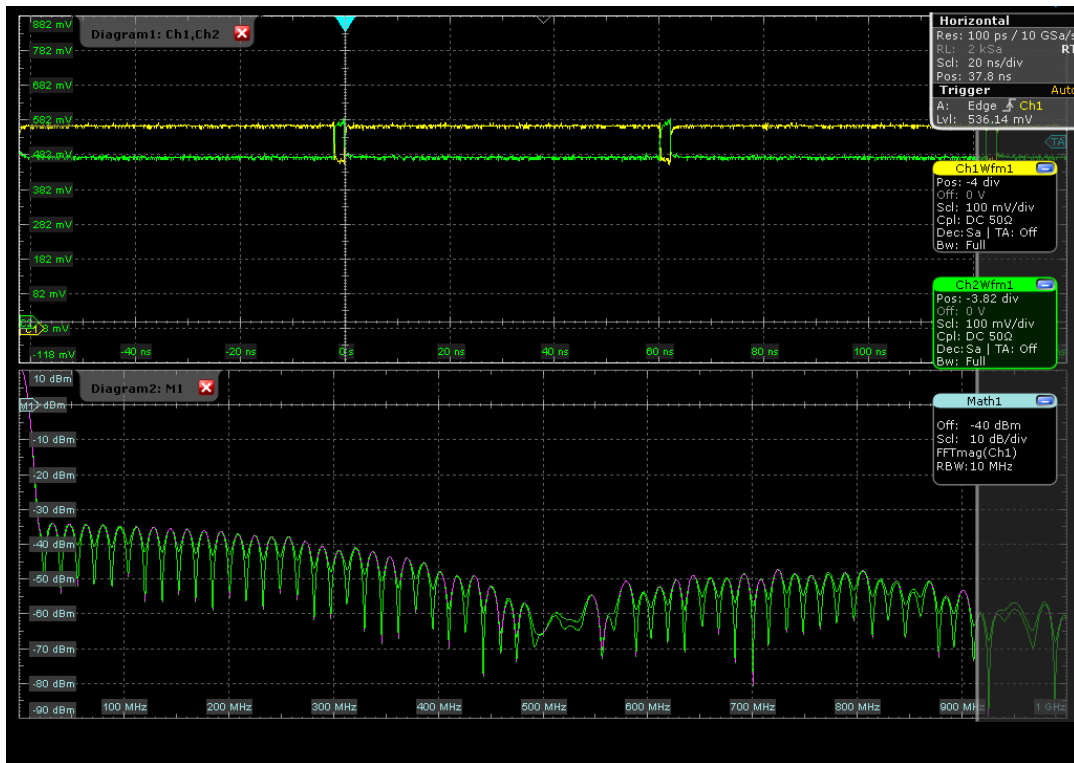


Abbildung 79: FFT von CH1des Transceiver Signals mit RBW von 10MHz

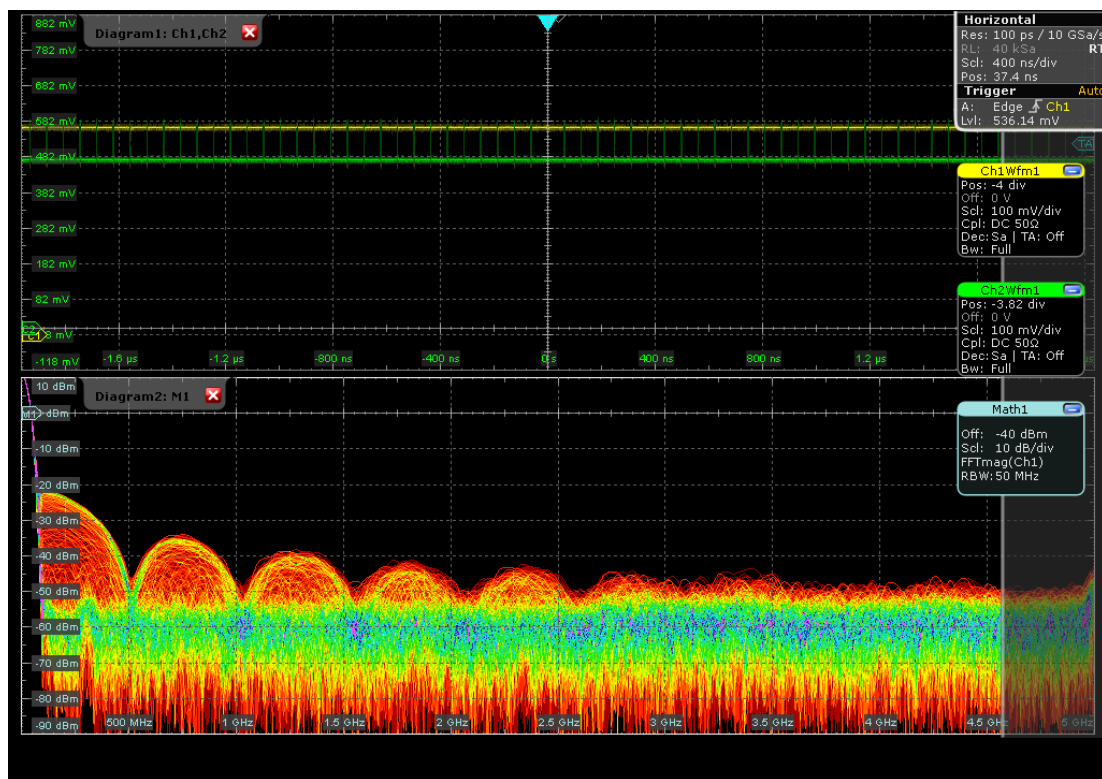


Abbildung 80: FFT von CH1 des Transceiver Signals mit RBW von 50MHz

Der letzte Test für das Programm besteht darin, Symbole vom PC über die UART an den Transceiver des FPGAs zu schicken, diese Symbole vom Receiver des FPGAs empfangen zu lassen, zu decodieren und über die UART zurück an den PC zu senden.

Dazu sind die differenziellen Ausgänge des Transceivers direkt mit den Eingängen des Receivers verbunden. Dieser Test soll die Funktionsfähigkeit des Receivers zeigen. Das Ergebnis dieses Tests ist in Abbildung 81 zu sehen. Es zeigt, dass die Symbole korrekt empfangen wurden. Da die UART aber sehr viel langsamer ist als der Transceiver, ist es keine Überraschung das keine Fehler vorhanden sind, diese treten jedoch auf, sobald die Methode mit einer vorher festgelegten Nachricht verwendet wird.

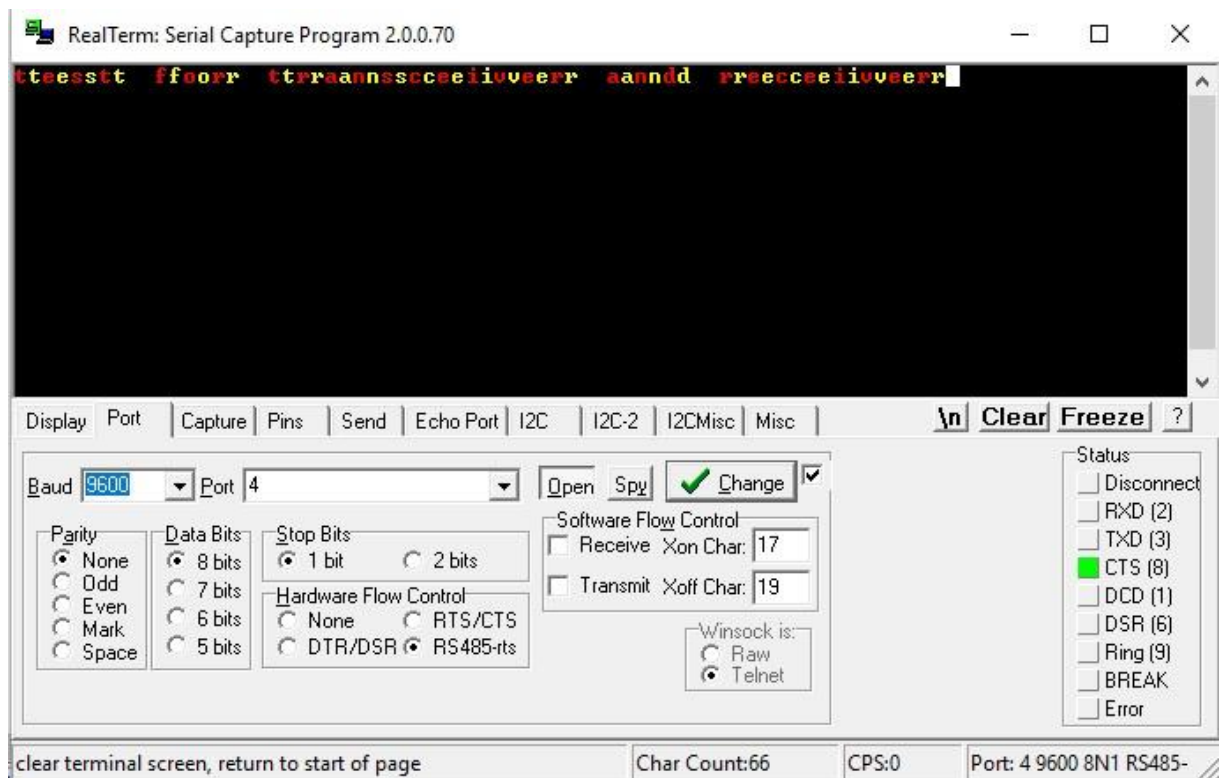


Abbildung 81: Test für Transceiver und Receiver

Als Abschluss dieser Tests kann festgehalten werden, dass die implementierten Transceiver und Receiver am FPGA ihre Arbeit korrekt ausführen. Der nächste Teil beschreibt nun die Ergebnisse für das gesamte System.

5.3. Ergebnisse des PPM Signals

In diesem Abschnitt über die Tests und Ergebnisse wird beschrieben wie sich das gesamte Übertragungssystem verhält. Es sei gesagt, das keine Messdaten aus dem optischen Bereich des Übertragungssystems vorhanden sind, da für eine solche Messung ein optisches Oszilloskop benötigt wird, jedoch Keines im Labor vorhanden ist.

Um einen kurzen Überblick über das System zu erhalten, wird kurz zusammengefasst aus welchen Komponenten es aufgebaut ist. Als Optische Quelle wird der Laser aus 3.1 verwendet, dieser wird mit dem Externen Modulator verbunden, für das Modulationssignal wird der Transceiver des FPGAs verwendet. Der optische Ausgang des Modulators wird mit dem optischen Empfänger verbunden und Dessen Ausgang wird in den Receiver Eingang des FPGAs geführt.

In Abbildung 82 ist das PPM-Signal zu erkennen, welches in den Receiver Eingang des FPGAs geleitet wird. Man erkennt sehr gut die einzelnen Piken des Signals. Daraus lässt sich folgern, dass der Receiver ein ähnliches Signal, welches der Transceiver aussendet, empfängt.

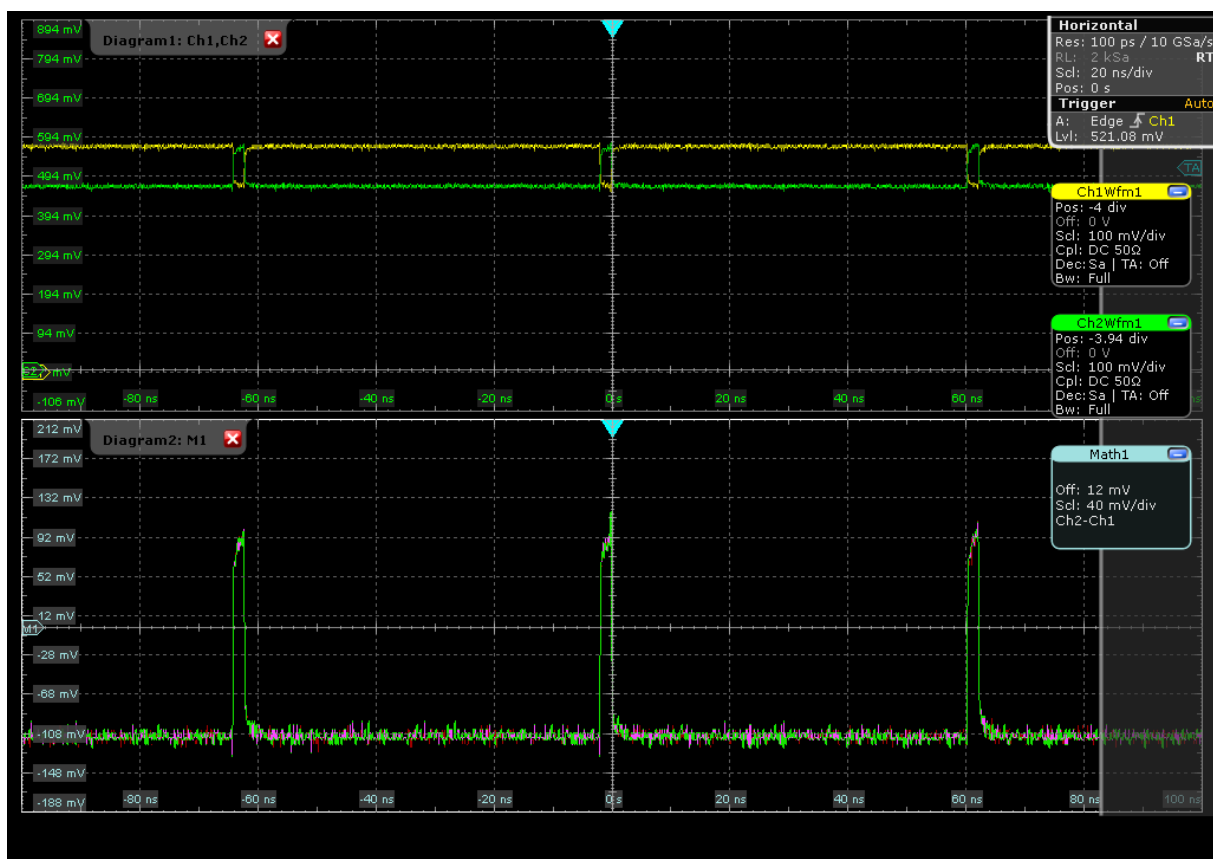


Abbildung 82: PPM-Signal am Receiver Eingang

Da nun erfolgreich das Signal am Receiver ankommt, kann der letzte große Test gestartet werden. Dazu werden alle möglichen Symbole vom PC ausgesendet und hoffentlich korrekt empfangen.

Doch bevor dieser Test zur Durchführung kommt, wird ein Blick auf das Spektrum des Signals geworfen, wie in Abbildung 83 für wenige Symbole gezeigt wird.. Man erkennt ebenfalls das hier mehr Symbole in kurzer Zeit übertragen werden. Die längeren Nullphasen bedeuten nur das keine Symbole in dieser Zeit gesendet werden, um das System nicht sofort am Maximum zu betreiben.

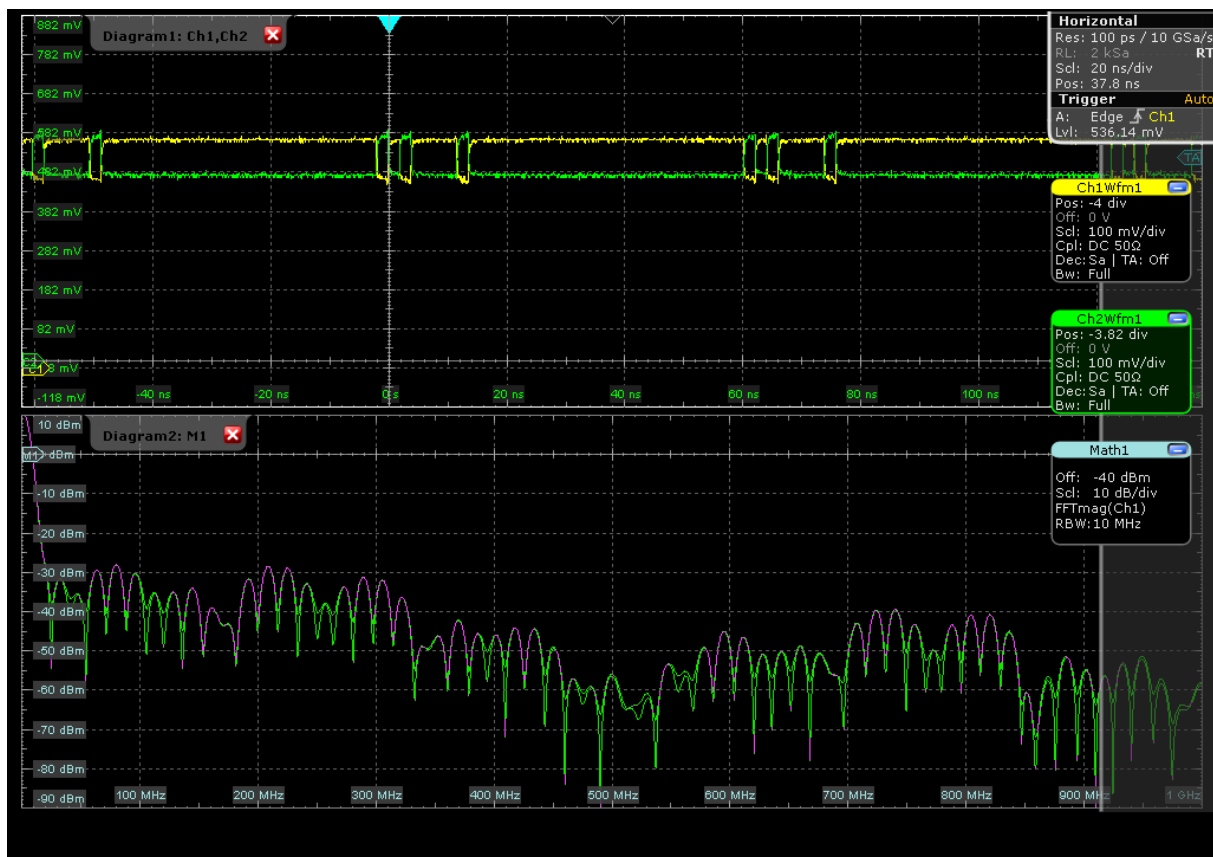


Abbildung 83: FFT des Signals am Receiver Eingang mit einem RBW von 10MHz

Auch das Spektrum mit einem größeren Ausschnitt des Signals wird in Abbildung 84 dargestellt. Es ist erkennbar, dass mehrere Symbole hintereinander, einen Einfluss auf das Spektrum haben und sich gegenseitig stören, im Vergleich zu Abbildung 80.

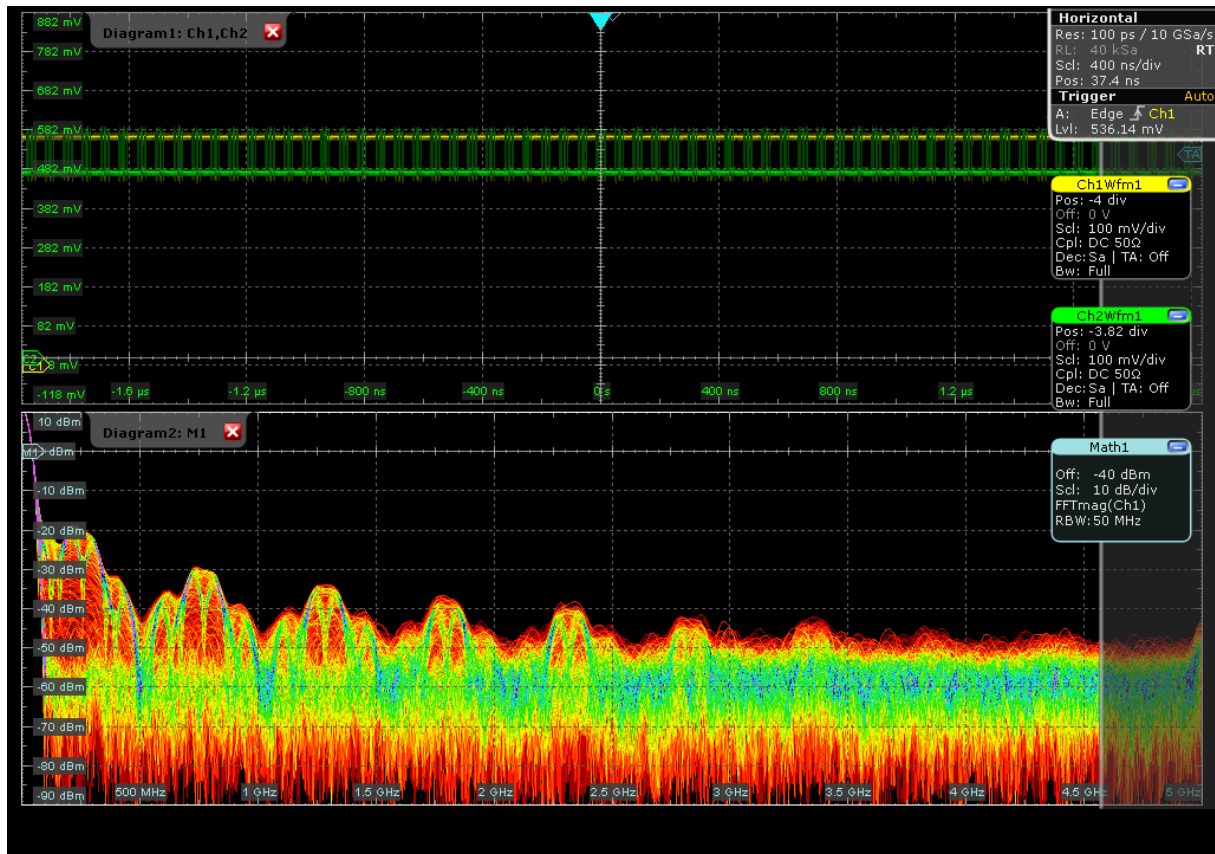


Abbildung 84: Spektrum des PPM-Signal am Receiver Eingang mit einem RBW von 50MHz

Für den letzten großen Test wird die Übertragungsform mittels UART verwendet, da wir bei dieser Variante ein anschauliches Ergebnis bekommen. Wie in Abbildung 85 leicht zu erkennen ist, gibt es ein erfolgreiches Testergebnis. Das erste Symbol ist „new Line“, RealTerm löst zwar selbst einen Zeilenumbruch aus, aber beim empfangen des Symbols wählt Es eine andere Darstellungsform. Die restlichen Symbole zeigen das jedes Einzelne aus Abbildung 51 empfangen wird. Symbole, welche nicht in der Codierung vorhanden sind, werden als NULL dargestellt, also der ASCII Code 00 in hexadezimal.

Leider wird NULL in RealTerm ebenfalls wie eine „Leertaste“ dargestellt.

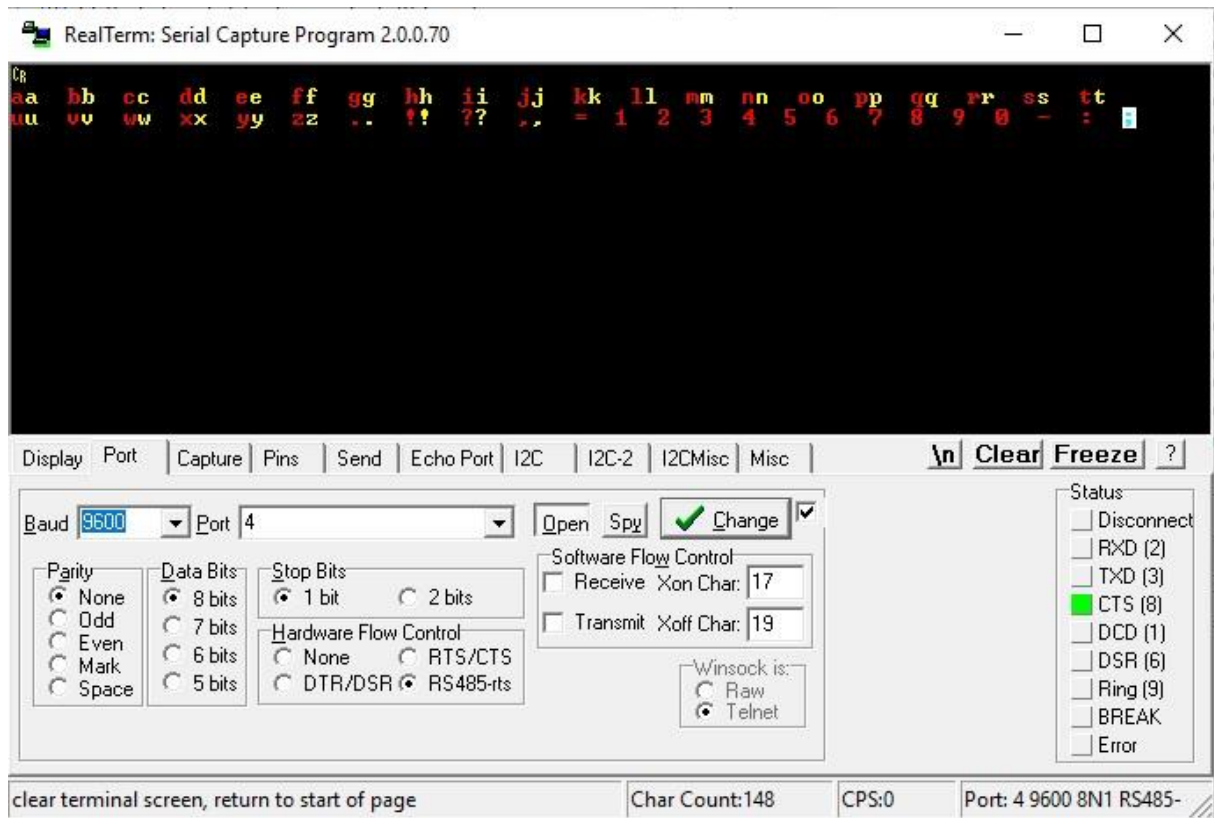


Abbildung 85: Testergebnis des gesamten Übertragungssystem mit allen möglichen Symbolen und einigen nicht möglichen Symbolen

Im Abschluss sollte noch erwähnt werden, dass die Tests mit der Variante der festen Übertragung nicht so erfolgreiche Ergebnisse liefert. Dies ist auch der Grund warum keine Ergebnisse darüber aufgeführt sind, aufgrund keiner verwertbaren Ergebnisse.

6. Schlussfolgerung

Das optische Übertragungssystem für spezielle Modulationsarten, mittels eines Mach-Zehnder Modulators liefert passable Ergebnisse, dennoch wurde nur eine Art der Modulation implementiert. Die vorerst integrierte Modulationsart ist 32-PPM. Der Grund warum nur eine Modulationsart umgesetzt wurde ist, dass die Erzeugung, bzw. das Verständnis des Frameworks des Programmes deutlich komplexer war als zu Beginn angenommen. Doch trotz dieser Einschränkung arbeitet das erstellte System sehr gut. Vor allem der dynamische Dateneingang der mittels UART umgesetzt wurde, funktioniert ausgezeichnet und ist einer der wichtigsten Schnittstellen für zukünftige Implementationen und Erweiterungen. Mit dieser Schnittstelle ist es möglich, Daten aus dem FPGA zu extrahieren. Ein gutes Beispiel dafür, sind neben den übertragenen Daten, ebenfalls Kennwerte, Zustände und Fehlermeldungen verschiedenster Anwendungen. Basierend auf diesem Fundament wäre es jedoch möglich, ohne größeren Zeitaufwand, andere Modulationsarten aus dem Kapitel 2 zu implementieren.

Die größte Herausforderung bei dieser Arbeit war es die verschiedenen Möglichkeiten des FPGAs verstehen zu lernen und herauszufinden wie gewisse Funktionalitäten des FPGAs korrekt angesprochen werden. Eine weitere interessante Herausforderung war die Verbindung des FPGAs mit dem PC und der in Abschnitt 4.2 genau beschriebenen UART. Diese ist zwar eine der einfacheren Möglichkeiten des FPGAs zu kommunizieren, dennoch war es eine herausforderndere Aufgabe als angenommen, die perfekt gelöst wurde. Auch der Transceiver des FPGAs war eine Thematik die eine sehr große Problemstellung mit sich brachte, es wurden zwar viele Kernstücke des Transceivers mittels des Frameworks erstellt, dennoch musste neben dem Verständnis für Dieses, eine Menge über den FPGA in Erfahrung gebracht werden, unter anderem die Funktionsweise des RAMs auf dem FPGA.

Abschließend ist zu sagen, dass dieses optische Übertragungssystem eine gute Funktionalität bietet, indem Erweiterungen deutlich schneller und unkomplizierter implementierbar sind.

Als kleiner Ausblick für zukünftige Verbesserungen, wäre als erster Punkt, die Integrierung von weiteren Modulationsarten und einem Menü um zwischen Diesen auszuwählen. Eine weitere sinnvolle Erweiterung wäre, eine weitere Schnittstelle zwischen FPGA und PC zu erzeugen, als Beispiel SPI oder Ethernet, um damit die Geschwindigkeit des dynamischen Systems zu erhöhen.

Literaturverzeichnis

- [1] Z. Gasshemlooy, W. Popoola und S. Rajbhandari, *Optical Wireless Communications: System and Channel Modelling with MATLAB*, CRC Press, 2013.
- [2] G. A. Mahdiraji und E. Zahedi, „Comparison of Selected Digital Modulation Schemes (OOK, PPM and DPIM) for Wireless Optical Communications,“ Student Conference on Research and Development, IEE, 2006.
- [3] X. Fu, G. Chen, T. Tang, Y. Zhao, P. Wang und Y. Zhang, „Research and Simulation of PPM Modulation and Demodulation System on Spatial Wireless Optical Communication,“ IEEE Conference Publications, IEEE, 2010.
- [4] L. Kong, W. Xu, H. Zhang, C. Zhao und X. You, „A PPM-Based Four-Dimensional Modulation Scheme for Visible Light Communications,“ IEEE Conference Publications, IEEE, 2015.
- [5] N. M. Aldibbiat, Z. Ghassemlooy und R. McLaughlin, „Performance of Dual Header-Pulse Interval Modulation (DH-PIM) for OpticalWireless Communication Systems,“ CiteSeerx, UK., 2000.
- [6] D. Veit und G. Freiberger, *Bau eines kohärenten Kommunikationssystems*, Graz: TU Graz Projekt, 2016.
- [7] Fujitsu, *Low Drive Voltage, Single Drive 12GB/s LiNbO3 External Modulator*, 2007.
- [8] Eudyna, *Eudyna(Sumitomo Electric) PIN*.
- [9] Eudyna, *ERA1402GT*.
- [10] Xilinx, „Logi Core IP Spartan-6 FPGA GTP Transceiver Wizard v1.11,“ 2011.
- [11] wikipedia, „wikipedia.org,“ [Online]. Available: https://de.wikipedia.org/wiki/Universal_Asynchronous_Receiver_Transmitter#/media/File:RS-232_timing.svg. [Zugriff am 24 1 2018].
- [12] UMBC, „UMBC CSEE,“ [Online]. Available: <https://www.csee.umbc.edu/~tinoosh/cmpe650/slides/UART.pdf>. [Zugriff am 13 12 2017].
- [13] R. W. World, „RF Wireless World,“ 8 Jänner 2017. [Online]. Available: <http://www.rfwireless-world.com/Tutorials/SDH-modulation.html>. [Zugriff am 6 September 2017].
- [14] Keysight Technologies, Inc., „Keysight Technologies,“ 24 Juli 2017. [Online]. Available: http://rfmw.em.keysight.com/wireless/helpfiles/89600b/webhelp/subsystems/wlan-ofdm/content/ofdm_basicprinciplesoverview.htm. [Zugriff am 20 September 2017].
- [15] C. R. H. Z. Frank Wüst, *Component Specification 11,352Gbps Single Drive LN Mach-Zehnder Modulator*, CoreOptics, 2009.
- [16] Xilinx, „Spartan-6 FPGA Block RAM Resources,“ 2011.

[17] JENOPTIK AG, „JENOPTIK,“ JENOPTIK, [Online]. Available:
<https://www.jenoptik.com/products/optoelectronic-systems/light-modulation/integrated-optical-modulators-fiber-coupled>. [Zugriff am 29 Jänner 2020].