Evaluierung von integrierten High-Speed Transceiver-Einheiten

Bachelorarbeit

DLR-IB-RM-OP-2021-245

Philipp Reisich

DLR

Hochschule München

Fakultät für Elektrotechnik und Informationstechnik

Studiengang Elektrotechnik und Informationstechnik

Bachelorarbeit von Philipp Reisich

Evaluierung von integrierten High-Speed Transceiver-Einheiten

Evaluation of integrated high-speed transceiver-units

Bearbeitungsbeginn: 01.04.2021 Abgabetermin: 26.12.2021

Lfd. Nr. gemäß Belegschein: 2104

Betreuer an der Hochschule: Prof. Dr. Joachim Schramm

**Betreuer am DLR: Markus Bihler MSc,
Dr. rer.nat. Thomas Bahls**

Erklärung des Bearbeiters:

Reisich	Philipp
Name	Vorname

1) Ich erkläre hiermit, dass ich die vorliegende Bachelorarbeit selbständig verfasst und nicht anderweitig zu Prüfungszwecken vorgelegt habe.

   Sämtliche benutzte Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate sind als solche gekennzeichnet.

München, 26.12.2021	
Ort, Datum	Unterschrift

2) Ich erkläre hiermit mein Einverständnis, dass die von mir erstellte Bachelorarbeit in die Bibliothek der Hochschule München eingestellt wird. Ich wurde darauf hingewiesen, dass die Hochschule in keiner Weise für die missbräuchliche Verwendung von Inhalten durch Dritte infolge der Lektüre der Arbeit haftet. Insbesondere ist mir bewusst, dass ich für die Anmeldung von Patenten, Warenzeichen oder Geschmacksmuster selbst verantwortlich bin und daraus resultierende Ansprüche selbst verfolgen muss.

München, 26.12.2021	
Ort, Datum	Unterschrift

Kurzfassung

Echtzeitkommunikation hat im Bereich der Robotik einen immer höheren Stellenwert. Die Latenz bei der Datenübertragung von Sensoren zu Aktoren hat einen direkten Einfluss auf die Regelbarkeit des robotischen Systems. Nicht nur eine schnelle, beziehungsweise möglichst latenzfreie Übertragung ist dabei wichtig, sondern auch die Integrität der übertragenen Daten. Eine entscheidende Rolle spielt hierbei die Feldbuskomponente, welche die einzelnen räumlich verteilten Einheiten sowohl untereinander als auch mit dem Echtzeitrechner verbindet. Diese Arbeit beschreibt die Implementierung und Evaluierung einer FPGA-basierten Feldbuskomponente zur Datenübertragung mit integrierten High-Speed Transceiver-Einheiten anhand von Xilinx 7 Series FPGAs. Diese Lösung soll dem aktuell eingesetzten externen Transceiver (TLK1221 von Texas Instruments) im Bezug auf Übertragungsgeschwindigkeit und Leistungsaufnahme gegenübergestellt und bewertet werden.

Abstract

Real-time communication is becoming increasingly important in the field of robotics. The latency of data transmission from sensors to actuators has a direct influence on the controllability of the robotic system. Not only a fast, or respectively a latency-free transmission is important, but also the integrity of the transmitted data. The fieldbus component plays a decisive role. It connects the physically separated devices with each other and the real-time computer. This thesis describes the implementation and evaluation of an FPGA-based fieldbus component for data transmission with integrated high-speed transceiver-units utilizing Xilinx 7 Series FPGAs. This solution is to be compared and evaluated with the currently used external transceiver (TLK1221 from Texas Instruments) in terms of transmission speed and power consumption.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Das Deutsche Zentrum für Luft- und Raumfahrt

Das Deutsche Zentrum für Luft- und Raumfahrt (DLR) agiert als das Forschungszentrum der Bundesregierung im Bereich Luft- und Raumfahrt. Das DLR kooperiert außerdem national und international in Forschungs- und Entwicklungsarbeiten in den Bereichen Luftfahrt, Raumfahrt, Energie, Verkehr, Sicherheit und Digitalisierung. Über die eigene Forschung hinaus ist das DLR für die Planung und Umsetzung der Deutschen Raumfahrtaktivität und als Dachorganisation für zwei Projektträger zur Forschungsförderung tätig. [2]

Die Geschichte des DLR geht zurück auf das Jahr 1969. Unter dem Zusammenschluss mehrerer Einrichtungen in den Bereichen Luft- und Raumfahrt entstand die „Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt“ (DFVLR). Nach der Fusion mit der „Deutschen Agentur für Raumfahrtangelegenheiten“ (DFL) im Jahr 1997 ist vom „Deutschen Zentrum für Luft- und Raumfahrt“ die Rede, so wie es auch heute noch heißt. [6]

Der Auftrag des DLR beinhaltet die Erforschung von Erde und Sonnensystem und die Forschung für den Erhalt der Umwelt. Dazu zählt die Entwicklung umweltverträglicher Technologien für die Energieversorgung und der Mobilität von morgen. Die Tätigkeiten des DLR reichen dabei von der Grundlagenforschung bis hin zur Entwicklung von Prototypen. Stand Februar 2021 beschäftigt das DLR in seinen 30 Standorten mehr als 10000 Mitarbeiterinnen und Mitarbeiter. Außerdem unterhält das DLR Büros in Brüssel, Paris, Tokyo und Washington D.C. Der Hauptsitz des DLR befindet sich in der Nähe von Köln. [2] Einer der Standorte des DLR befindet sich Oberpfaffenhofen in der Nähe von München und zählt mit 1800 Mitarbeitern zu den größten Forschungszentren Deutschlands.

Unter anderem befinden sich an diesem Standort die Institute für:

- Systemdynamik und Regelungssysteme
- Hochfrequenztechnik und Radarsysteme
- Kommunikation und Navigation
- Institut für Robotik und Mechatronik

Diese Bachelorarbeit wurde im Institut für Robotik und Mechatronik abgelegt, welches zuständig ist für Forschung und Entwicklung in den Bereichen Robotik, Mechatronik und optische Systeme. Dazu gehört die Entwicklung von robotischen Systemen, die Manipulations- und Fortbewegungsfähigkeiten des Menschen reproduzieren und erweitern sollen. Im Fokus steht dabei die Ermöglichung einer sichereren und effizienteren Interaktion des Menschen mit seiner Umwelt. Zur bekanntesten Entwicklung des Instituts zählt Rollin‘ Justin, ein humanoider Roboter, welcher als Forschungsplattform im Bereich der Servicerobotik dient. [3]

1.2. Motivation der Arbeit

Ob in der Robotik, oder am Arbeitsplatz im Homeoffice - Eine schneller Datentransfer gewinnt heutzutage immer mehr an Relevanz. So registrierte im März 2020 zu Beginn der Pandemie der Betreiber eines Internetknotens in Frankfurt einen Rekord von 9,1 Terabit pro Sekunde an Datenumsatz. Laut eines Artikels der Frankfurter Allgemeinen Zeitung „entspricht [dies] einer Datenmenge von zwei Millionen hochauflösenden Videofilmen oder dem Informationsgehalt von zwei Milliarden beschriebenen Din-A4-Seiten“. [21] Um den wachsenden Anforderungen gerecht zu werden, muss daher ständig an neuen Lösungen geforscht werden.

Auch in der Robotik hat die Technologie der High-Speed Datenkommunikation einen hohen Stellenwert. Durch einen stetigen Zuwachs der Rechenleistung können immer komplexere Systeme entwickelt werden. Damit geht allerdings auch ein erhöhtes Datenaufkommen einher, das zuverlässig an die Recheneinheiten übertragen werden muss. Diese Aufgabe wird von Transceivern übernommen, die aus einem Transmitter und einem Receiver bestehen. Solche Transceiver kommen in den robotischen Systemen des DLR zum Einsatz. Für diesen Zweck werden oft FPGAs verwendet, da diese in der Regel über integrierte Transceiver-Einheiten verfügen. Aus Gründen der Leistungseffizienz wurde jedoch bei der Entwicklung der externe Transceiver-Baustein TLK1221 von Texas Instruments verwendet. [20] Da seit Entwicklungsbeginn allerdings neue Generationen von FPGAs mit integrierten Transceiver-Einheiten entwickelt wurden, ist diese Aussage womöglich nicht mehr zutreffend.

1.3. Aufgabenstellung

Das Ziel dieser Arbeit ist die Untersuchung verschiedener Kommunikationsschnittstellen aktueller FPGAs, die für eine serielle differentielle High-Speed-Lösung des SpaceWire Physical-Layers verwendet werden können. Diese Untersuchungen sollen repräsentativ an Xilinx 7 Series FPGAs durchgeführt werden. Dafür wurden zwei verschiedene Kommunikationsschnittstellen implementiert und deren Funktionsfähigkeit validiert. Des Weiteren soll ein Konzept zur Messung der Leistungsaufnahme beider Lösungen ausgearbeitet werden. Die Ergebnisse der Messung sollen den Daten des aktuell zum Einsatz kommenden externen Transceiver-Baustein TLK1221 von Texas Instruments gegenübergestellt und bewertet werden.

2. Theoretische Grundlagen

2.1. Field Programmable Gate Arrays

Ein Field Programmable Gate Array, oder kurz FPGA, bezeichnet einen digitalen Schaltkreis, dessen Logikfunktion vom Anwender vorgegeben werden kann. [27, S. 77] FPGAs bestehen aus matrixförmig angeordneten Logikblöcken (eng. Look Up Table/LUT), die je nach Verschaltung unterschiedliche Funktionen erfüllen. So kann das gleiche FPGA für verschiedene Anwendungszwecke genutzt werden. [27, S. 85] Zu den Anwendungsgebieten von FPGAS zählen unter anderem Fahrerassistenzsysteme, Motorsteuerungen, Systeme zur Videoverarbeitung, Internet of Things (IoT) und 5G Mobilfunkmasten. [19, S. 5] Alle diese Aufgaben könnten jedoch auch von Chips übernommen werden, die spezifisch für diese Zwecke entwickelt wurden. Hierbei spricht man von integrierten Schaltungen (engl. Integrated Circuit/IC). Die Entwicklung einer Anwendung mit FPGAs bedeutet allerdings in der Regel einen deutlich geringeren Aufwand als bei ICs, da FPGAs durch die bereits vorhandenen Logikblöcke auf einem hohen Abstraktionsniveau entwickelt werden können. Dies reduziert die Zeit von der Idee bis zur fertigen Umsetzung drastisch (Time to market). [22, S. 1] Die Programmierung von FPGAs erfolgt mit Hardwarebeschreibungssprachen. Zu den bekanntesten Sprachen zählen VHDL und Verilog, wobei VHDL vermehrt Anwendung im europäischen Raum findet.

Im Rahmen dieser Arbeit wurden ausschließlich FPGAs der Firma Xilinx betrachtet. Eine ausführliche Beschreibung der Architektur des verwendeten FPGAs erfolgt in Kapitel 3.

2.2. Single-Ended vs Differentiell

Single-Ended und **Differentiell** sind zwei verschiedene Methoden zur Übertragung von elektrischen Signalen auf dem Physical-Layer. Bei einer Single-Ended Übertragung wird zwischen Datenquelle und Datensenke eine physische Verbindung über einen metallischen Leiter hergestellt, was hier den Physical-Layer darstellt. Zur Übertragung von Informationen werden konkrete Spannungspegel definiert, ab denen ein Signal als eine logische '1' oder '0' zu interpretiert ist. [1, S. 4]

Die differentielle Übertragungsmethode führt die Nutzung eines zweiten Leiters ein, an dem die selbe Spannung, aber mit invertierter Polarität anliegt. In Abbildung 2.1 ist der Signalverlauf der beiden Leiter zu erkennen. Dabei trägt der Leiter n das invertierte Signal von Leiter p.

Abbildung 2.1.: Signalverlauf mit differentiellem Leiterpaar [1, S. 5]

Anstatt das anliegende Potential mit dem Ground-Potential zu vergleichen, werden die beiden Signale miteinander verglichen. Besitzt der positive Leiter ein Signal mit einem höheren Spannungspegel als der negative Leiter, ist dies als eine logisch '1' zu werten. Ist der Signalpegel des positiven Leiters niedriger als der des negativen Leiters entspricht das übertragene Signal einer logischen '0'.

Bei der heutigen Nutzung von High-Speed Transceivern findet praktisch nur noch die differentielle Kommunikation Anwendung. Dies ist darin begründet, dass die differentielle Übertragungsmethode deutlich weniger störanfällig gegenüber externen elektromagnetischen Einflüssen ist. Abbildung 2.2 zeigt den Einfluss einer Störung auf das differentielle Leiterpaar:

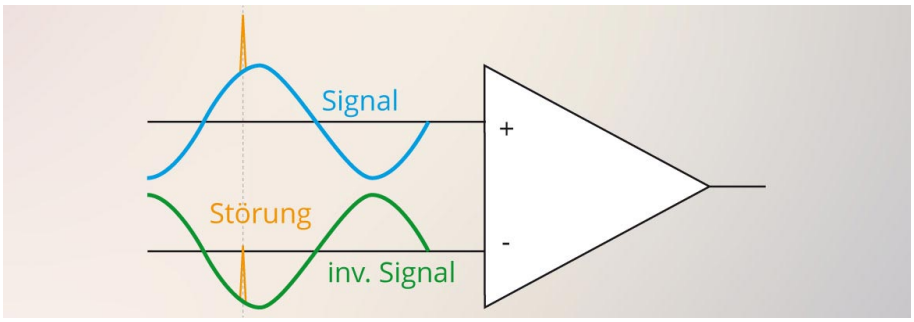

Abbildung 2.2.: Störung bei differentieller Übertragung [4]

Da die Spannungspegel der Signale nicht mit dem Ground-Potential, sondern miteinander verglichen werden, kann durch die Differenzbildung der beiden Signalpegel die Auswirkung der Störung herausgerechnet werden.

2.2.1. Low-Voltage Differential Signaling

Low-Voltage Differential Signaling (LVDS) ist eine spezieller Standard, der sich bei differentieller High-Speed Kommunikation durchgesetzt hat. Dazu wird das differentielle Leitungspaar mit einem 100 Ω Widerstand terminiert. Die differentiellen Signale werden mit einer 3,5mA Konstantstromquelle erzeugt. Vier Transistoren sind für die Richtung des Stromflusses auf den Datenleitungen und damit für den Signalpegel verantwortlich. (Vgl. Abb. 2.3) Dabei fällt nach dem Ohmschen Gesetze über den Terminierungswiderstand eine Spannung von 350mV ab. Da die Stromquelle selber die Richtung des Stromflusses jedoch nicht ändert, werden weniger elektromagnetische Störungen verursacht und die interne Schaltung ist weniger anfällig gegenüber Störungen. [24]

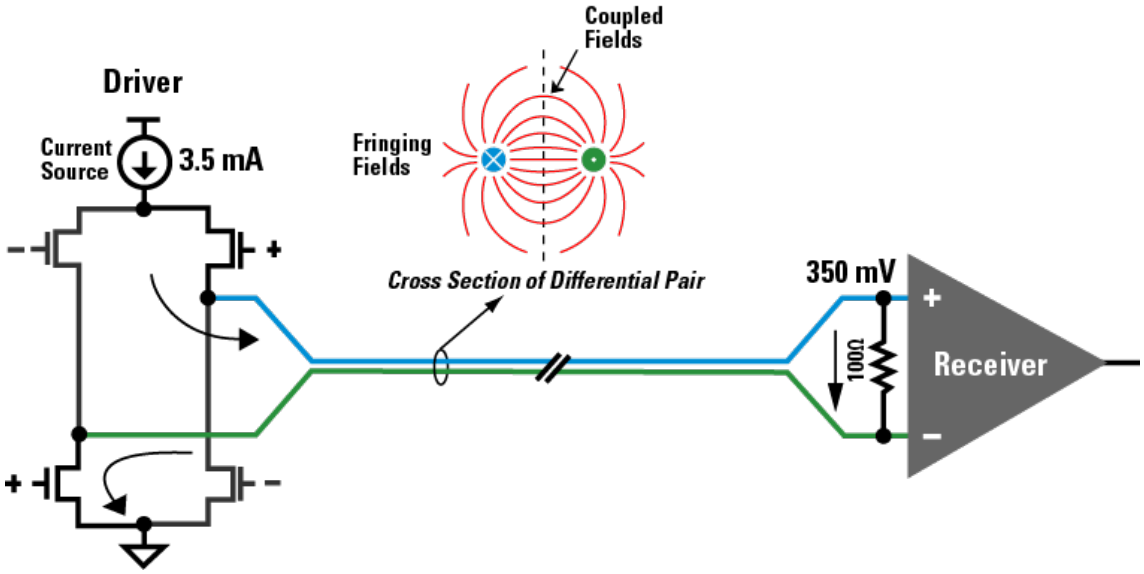

Abbildung 2.3.: LVDS-Treiber und Empfänger [5]

2.3. Seriell vs Parallel

Der Austausch von Daten zwischen den Logikzellen innerhalb des FPGAs geschieht parallel. So wird beispielsweise ein 8-Bit Signal auch mit 8 Datenleitungen an andere Logikzellen zur weiteren Verarbeitung übertragen. In der Sprache VHDL entsprechen diese Signale dem Datentyp **STD_LOGIC_VECTOR**.

Zur Verarbeitung von Daten, die außerhalb des FPGAs erzeugt wurden, existiert eine physische Schnittstelle (I/O), die einen beidseitigen Transfer von Informationen ermöglicht. Hierbei ist es jedoch nicht immer von Vorteil diese Daten parallel zu übertragen, da das FPGA nur über eine begrenzte Anzahl an I/O Pins verfügt. Außerdem kommt es

bei größeren Übertragungsgeschwindigkeiten vermehrt zu gegenseitigen elektromagnetischen Störungen der Leiter untereinander (Crosstalk). [1, S. 65] Diese Effekte wirken sich negativ auf die Integrität der übertragenen Daten aus.

Bei der seriellen Übertragung werden diese Daten nicht auf einmal über mehrere Leiter, sondern Bit für Bit über einen einzelnen Leiter übertragen. Um diese Informationen innerhalb des FPGAs verarbeiten zu können, müssen diese Datenströme jedoch wieder deserialisiert werden. Deshalb verfügen viele FPGAs über eine dedizierte Logik in Silizium, die die Aufgabe der Serialisierung und Deserialisierung von Daten zur Kommunikation mit der Außenwelt übernimmt.

Abbildung 2.4 zeigt das Prinzip der parallelen und seriellen Übertragung:

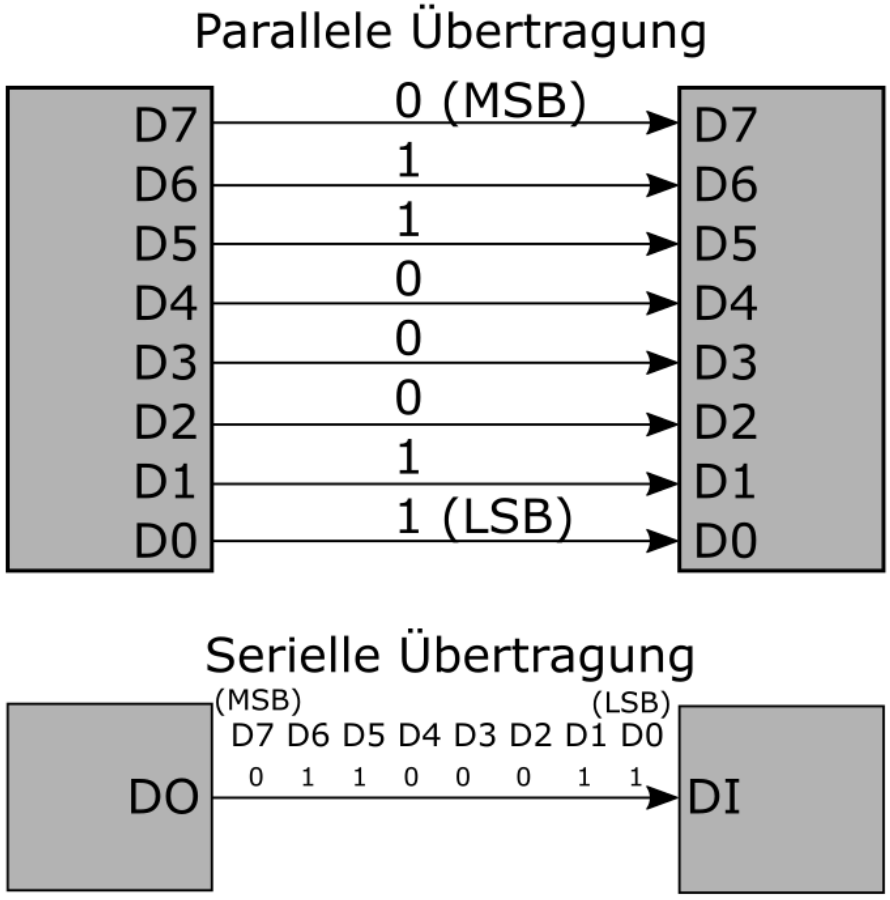

Abbildung 2.4.: Prinzip der seriellen und parallelen Übertragung

2.3.1. Serialisierer/Deserialisierer

Seriell-zu-parallel-Konverter, beziehungsweise Parallel-zu-seriell-Konverter (SerDes) bilden die Grundlage für einen schnellen und effizienten Austausch von Daten zwischen physisch getrennten Systemen. Ein SerDes besteht im Grunde genommen aus einem Schieberegister, welches die am parallelen Datenbus anliegenden Daten mit jeder positiven Taktflanke nacheinander ausgibt, beziehungsweise die Daten am seriellen Bus einliest und nach abgeschlossener Übertragung des Datenframes in einem Buffer zur weiteren Verarbeitung zwischenspeichert. Beträgt der zu übertragende Datenframe beispielsweise die Größe n Bit, so benötigt die vollständige Übertragung dieses Frames ebenfalls n Taktperioden. [1, S. 20] Die Abbildung 2.5 zeigt den Aufbau eines SerDes mit den zum Betrieb notwendigen Komponenten. Dazu gehören ein Taktgenerator, der die serielle Übertragungsgeschwindigkeit vorgibt, ein Serialisierer, der die parallel anliegenden Daten aus dem Buffer serialisiert und ein Deserialisierer, der den seriell empfangenen Datenstrom parallel in einem Buffer ausgibt. Vor dem Versand, beziehungsweise nach dem Empfang werden die Daten kodiert, beziehungsweise dekodiert.

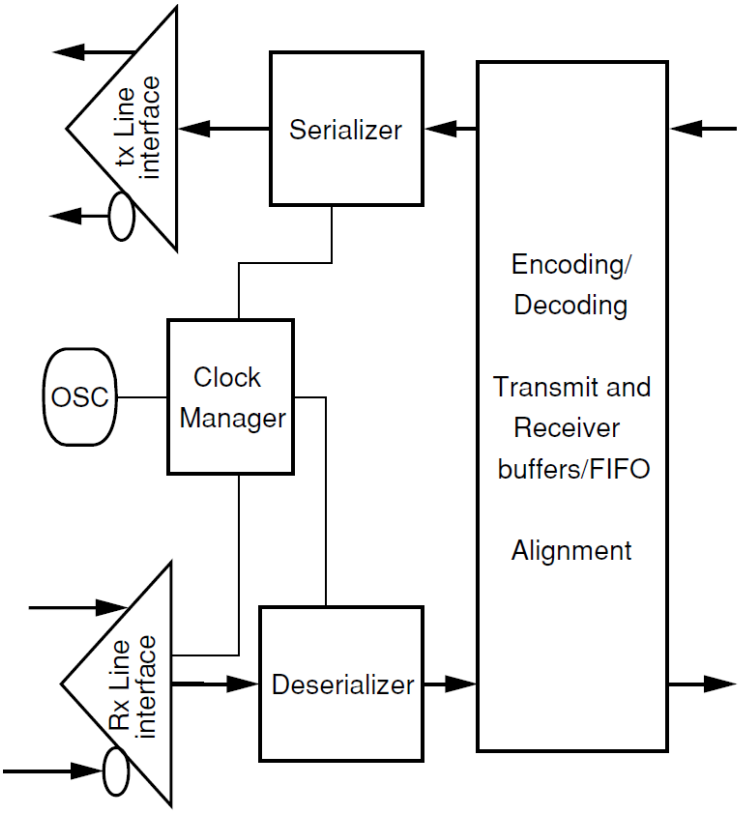

Abbildung 2.5.: SerDes Blockdiagramm [1, S. 20]

Warum seriell übertragene Daten oft kodiert werden soll im nächsten Kapitel 2.4 erläutert werden.

2.4. Leitungskodierung

Die Aufgabe der Leitungskodierung besteht darin die Fehleranfälligkeit bei der Datenübertragung zu minimieren. Dies wird erreicht durch Redundanz und eine möglichst gleichspannungfreie Übertragung (DC-Balancing). Dazu werden die Daten vor der Übertragung so modifiziert, dass die Anzahl der logischen Einsen und Nullen möglichst gleichmäßig ist. Zusätzlich werden oft weitere Informationen den Daten hinzugefügt, um die Erkennung von Fehlern zu ermöglichen und gegebenenfalls Fehler zu korrigieren. [1, S. 25]

Eine spezielle Leitungskodierung, die im Rahmen dieser Arbeit besondere Relevanz hat, soll im folgenden Abschnitt erläutert werden.

2.4.1. 8B/10B Kodierung

Die 8B/10B Kodierung (gesprochen „Acht aus Zehn Kodierung“) ist eine von IBM entwickelte Form der Leitungskodierung und wird unter anderem in der Kommunikation mit dem Gigabit Ethernet Protokoll genutzt. Dabei werden 8-Bit Wörter in 10-Bit Symbole übersetzt. Die effektive Übertragungsrate 8B/10B kodierter Daten beträgt somit nur 80% der seriellen Übertragungsgeschwindigkeit. Da ein 8-Bit Wort mit 256 verschiedene Zuständen auf ein 10-Bit Symbol mit 1024 verschiedenen Zuständen abgebildet wird, ist somit die Wahl von geeigneten Bitfolgen möglich, die durch häufigere Wechsel als das eigentlich Wort eine weitestgehend gleichspannungsfreie Übertragung ermöglichen. Die folgende Tabelle 2.1 zeigt zwei Beispiele, für eine gleichspannungfreie Übertragung durch die Wahl geeigneter 10-Bit Symbole:

8-Bit Wort	10-Bit Symbol
00000000	1001110100
00000001	0111010100

Tabelle 2.1.: Beispiele für gleichspannungsfreie Symbole [1, S. 26]

Da durch die begrenzte Anzahl an Symbolen allerdings nicht immer eine gleichmäßige Verteilung von Nullen und Einsen wie im obigen Beispiel möglich ist, wird ein Mechanismus eingeführt, der eine gleichspannungsfreie Übertragung durch eine sich ständig wechselnde Disparität (engl. *Running Disparity*) schafft. Dazu werden jedem 8-Bit Wort zwei mögliche 10-Bit Symbole zugewiesen. Diese Symbole werden je nach Disparität mit „+“ und „-“ gekennzeichnet. Die Folgende Tabelle 2.2 Zeigt einige Beispiel-Symbole mit Running Disparity:

Name	HEX	8-Bit	10-Bit RD -	10-Bit RD +
D10.7	EA	11101010	0101011110	0101010001
D31.7	FF	11111111	1010110001	0101001110
D4.5	A4	10100100	1101011010	0010101010
D0.0	00	00000000	1001110100	0110001011
D23.0	17	00010111	1110100100	0001011011
K28.7	FC	11111100	0011111000	1100000111

Tabelle 2.2.: Beispiele für 8B/10B Symbole mit Running Disparity (RD)[1, S. 26]

Durch den ständigen Wechsel der Disparität nach jedem übertragenen Symbol ist die Übertragung auf einen größeren Zeitraum gesehen gleichspannungsfrei. Außerdem kann der Empfänger Übertragungsfehler dadurch feststellen, wenn zwei aufeinander folgende Smybole die gleiche Disparität besitzen.

Neben den Symbolen für die Daten existieren auch sogenannte K-character, die den Beginn eines Symbols signalisieren. Diese werden so gewählt, dass deren Bitfolge einmalig ist und jederzeit durch einen Sequenzdetektor erkannt werden kann. Sobald eine dieser Bit-Sequenzen erkannt wird, weiß der Empfänger über die Grenzen der nächsten aufeinander folgenden 10-Bit Datensymbole Bescheid und kann die empfangenen Daten richtig anordnen. Diese speziellen K-character werden auch „Komma-Symbol“ genannt, weshalb dieser Prozess als *Comma Detection* bezeichnet wird. [1, S. 26ff]

Abbildung 2.6 zeigt den Ablauf einer seriellen Übertragung eines 8B/10B kodierten Datenstreams mit Comma Detection:

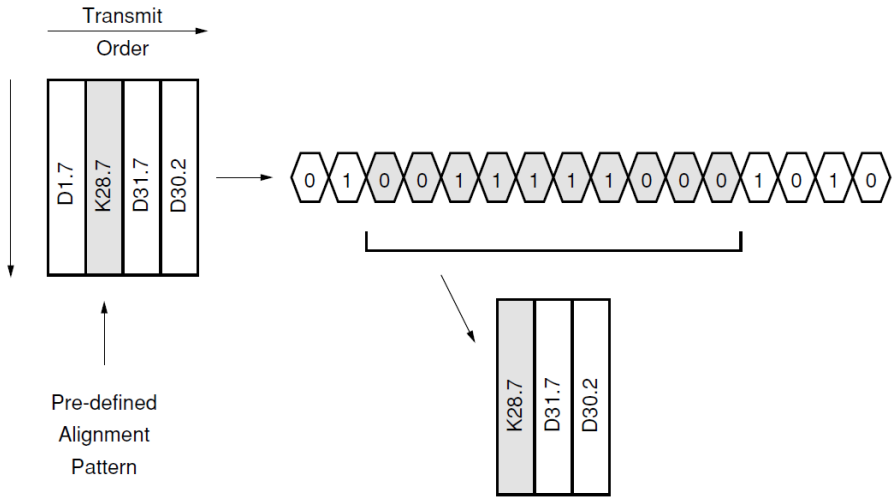

Abbildung 2.6.: 8B/10B kodierter Datenstream mit Comma Detection [1, S. 27]

3. Xilinx Zynq 7000 Architektur

Die Zynq 7000 Architektur des Herstellers Xilinx ermöglicht einen großen Funktionalitätsumfang durch die Kombination von Processing System (Mikrocontroller) und Programmable Logic (FPGA) in nur einem Chip (SoC). Dadurch können die Stärken der jeweiligen Systeme besser ausgenutzt werden. Das Konzept der Zusammenarbeit von FPGA und Mikroprozessor ist jedoch nicht neu. Herkömmliche PCBs werden für eine spezifische Anwendung entworfen und mit entsprechenden Komponenten bestückt. Allerdings hat die physische Trennung der Systeme auf der Leiterplatte einen negativen Einfluss auf Latenz und Bandbreite bei Datentransfers.
Durch die Unterbringung der beiden Systeme in einem Gehäuse bieten Zynq 7000 SoCs einen zuverlässigen und schnellen Datenaustausch zwischen Processing System und Programmable Logic über einen gemeinsamen Datenbus namens AXI. [19] (Vgl. Abb. 3.1)

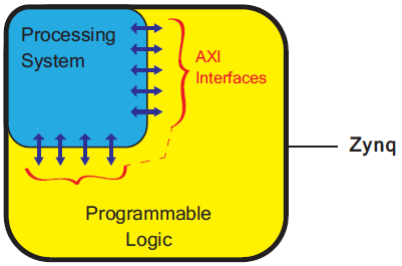

Abbildung 3.1.: Zynq 7000 SoC mit Processing System und Programmable Logic [25, S. 4]

Alle Zynq 7000 SoCs verfügen über einen Dual-core ARM Cortex-A9 Mikropozessor als Processing System, welcher je nach Zynq-Modell mit einer maximalen Taktfrequenz von 1 GHz betrieben werden kann. Dieser Prozessor ist in der Lage ganze Betriebssysteme wie zum Beispiel Linux auszuführen.

Im Rahmen dieser Arbeit liegt der Fokus allerdings auf der Programmable Logic, da diese über die zu implementierenden Transceiver-Einheiten verfügt. Daher sollen die Funktionen des Processing Systems nicht weiter vertieft werden.

Die Programmable Logic des Zynq 7000 SoCs besteht aus einem FPGA der 7 Series Generation, welche wiederum aus vier FPGA-Familien besteht. Diese werden unterteilt in Spartan-7, Artix-7, Kintex-7 und Virtex-7 FPGAs. Je nach Komplexität der implementierten Schaltung ist die Wahl aus einer der vier 7 Series Familien zu treffen. [8]

Einige Eigenschaften der einzelnen 7 Series Familien sind der Tabelle 3.1 zu entnehmen.

Max. Capability	Spartan-7	Artix-7	Kintex-7	Virtex-7
Logic Cells	102K	215K	478K	1,955K
Block RAM(1)	4.2 Mb	13 Mb	34 Mb	68 Mb
DSP Slices	160	740	1,920	3,600
DSP Performance(2)	176 GMAC/s	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
MicroBlaze CPU(3)	260 DMIPs	303 DMIPs	438 DMIPs	441 DMIPs
Transceivers	–	16	32	96
Transceiver Speed	–	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Serial Bandwidth	–	211 Gb/s	800 Gb/s	2,784 Gb/s
PCIe Interface	–	x4 Gen2	x8 Gen2	x8 Gen3
Memory Interface	800 Mb/s	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
I/O Pins	400	500	500	1,200
I/O Voltage	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V
Package Options	Low-Cost, Wire-Bond	Low-Cost, Wire-Bond, Bare-Die Flip-Chip	Bare-Die Flip-Chip and High-Performance Flip-Chip	Highest Performance Flip-Chip

Tabelle 3.1.: Eigenschaften von 7 Series FPGAs [8]

3.1. Input/Output

Xilinx FPGAs sind unterteilt in mehrere physische Bereiche. Diese Bereiche werden „Bank“ bezeichnet. Jeder Bank werden 50 I/O Pins zugeordnet, die als Ein- oder Ausgänge konfiguriert werden können. Diese werden wiederum in die Kategorien „high performance“ (HP) und „high range“ (HR) unterteilt und nutzen unterschiedliche Core-Spannungen. So nutzen HR-Banks einen Spannungsbereich von 1,2V–3,3V, HP-Banks werden im Bereich von 1,2V–1,8V betrieben. Zur Nutzung von LVDS können zwei nebeneinander liegende I/O Pins als differentielle Ein-/Ausgänge definiert werden. Dabei wird intern ein 100 Ω Terminierungswiderstand zwischen den differentiellen Eingängen zugeschaltet, um dem LVDS-Standard zu entsprechen. Zusätzlich verfügt jeder I/O Port über einen dedizierten 8-Bit SerDes. Dabei können auch mehrere SerDese kombiniert werden um den parallelen Datenbus auf bis zu 14 Bit zu erweitern. [8, S. 10]

3.2. High-Speed Serial Transceiver

Neben den SerDes für die I/O Pins verfügen Xilinx 7 Series FPGAs über dedizierte Multi-Gigabit-Transceiver, die je nach Familie unterschiedliche Geschwindigkeiten Erreichen. So sind Spartan-7 FPGAS optimiert für Anwendungen mit einem geringen Energieverbrauch. Daher sind in dieser Ausführung keine High-Speed Transceiver integriert. Artix-7 FPGAs sind ebenfalls für Anwendungen geeignet, wo ein geringer Energieverbrauch eine Rolle spielt. Im Gegensatz zur Spartan-7 verfügen Artix-7 FPGAs aber über integrierte High-Speed Transceiver-Einheiten, die Übertragungsgeschwindigkeiten von bis zu 6,6

Gb/s erreichen. Die Kintex-7 Familie bietet das beste Preis-Leistungs-Verhältnis und kommt mit integrierten Transceivern, die Geschwindigkeiten von bis zu 12,5 Gb/s erreichen. Die FPGA-Familie mit der höchsten Performance stellt die Virtex-7 Familie dar. Die integrierten Transceiver erreichen Geschwindigkeiten von bis zu 28,05 Gb/s und sind somit die schnellsten der 7 Series FPGAs. Jeder Transceiver besteht aus einem Transmitter und einem Receiver und kann in einer Voll-Duplex-Konfiguration betrieben werden. Jeder einzelne der Transceiver verfügt über eine in Silizium integrierte Logik zur 8B/10B Kodierung/Dekodierung und zur Clockrückgewinnung. Jeder der integrierten Transceiver verfügt über die Funktionalität zur Post-/Pre-Emphasis, was die Integrität der übertragenen Daten zusätzlich verbessert. [8, S. 11] Somit stellen die integrierten Gigabit-Transceiver von Xilinx eine kompakte Lösung für High-Speed Datentransfers dar.

4. Ansatz und Implementierung

4.1. Grundlegendes Gesamtkonzept

Folgend soll das Konzept zur Messung der Leistungsaufnahme des FPGAs im Betrieb mit integrierten Transceiver-Einheiten vorgestellt werden. Die beiden Hauptkomponenten des Aufbaus sind das zu messende System (FPGA mit integrierten Transceiver-Einheiten) und die Messgeräte zur Messung der Leistungsaufnahme. Das FPGA befindet sich auf einem Entwicklungsboard (ZC706), welches in Kapitel 4.1.1 behandelt wird. Der Aufbau des gesamten Systems ist in abstrahierter Form dem Blockschaltbild 4.1 zu entnehmen:

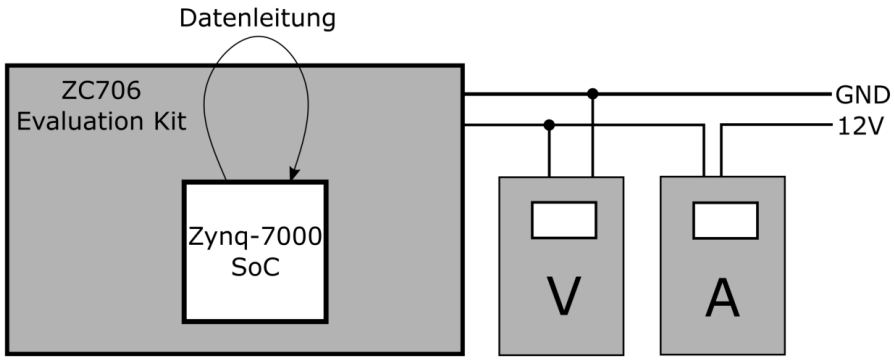

Abbildung 4.1.: Messaufbau

Um die Messung der Leistungsaufnahme der Transceiver im Betrieb durchführen zu können sind die folgenden Schritte notwendig:

1. Implementieren der Transceiver-Einheiten und Logik zum Erzeugen von Beispieldaten
2. Validieren der Schaltung mit Simulation
3. Validieren der Schaltung im Betrieb
4. Messung der Leistungsaufnahme

Diese Schritte sollen in den folgenden Abschnitten 4.2 und 4.3 für beide Ansätze getrennt erläutert werden.

4.1.1. Entwicklungsboard mit Zynq 7000 SoC

Das Entwicklungsboard „ZC706“ von Xilinx (vgl. Abb 4.2) vereinfacht den Entwicklungsprozess durch bereits vorhandene Peripherie. Die Peripherie beinhaltet:

- GPIO-Header, die mit den I/O-Ports des FPGAs verbunden sind
- SMA-Stecker zur differentiellen Datenübertragung mit den integrierten High-Speed Transceivern über Koaxialkabel
- mehrere Knöpfe, Schalter, Debug-LEDs zun imteragieren mit dem FPGA
- Schnittstelle zum Programmieren und Debuggen des Zynq SoCs

Das Entwicklungsboard verfügt über weitere Peripherie, die jedoch im Rahmen dieser Arbeit nicht verwendet wurde. Eine genaue Auflistung aller Funktionen sind dem Datenblatt [18] zu entnehmen.

Die zentrale Komponente stellt der Zynq 7000 SoC dar. Als *Programmable Logic* ist ein FPGA der Kintex-7 Familie verbaut. Dieses verfügt in der verwendeten Ausführung über 16 integrierte High-Speed Transceiver der GTX-Reihe, welche eine serielle Übertragungsrate von 12,5Gb/s erreichen. [9] Die Implementierung dieser Transceiver erfolgt in Kapitel 4.3. Wie alle 7 Series FPGAs sind die I/O Pins mit internen seriell-zu-parallel-Kovertern/parallel-zu-seriell-Konvertern ausgestattet. Diese Logik wird „SelectIO“ bezeichnet und soll in Kapitel 4.2 behandelt werden. [10]

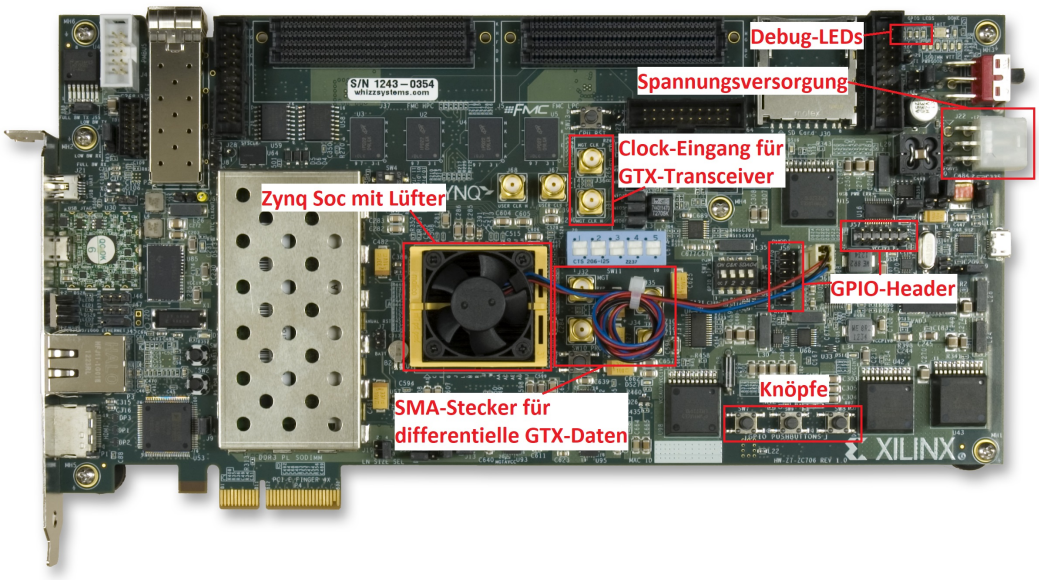

Abbildung 4.2.: ZC706 Entwicklungsboard [17]

4.1.2. Transceiver-Design

Untersucht wurden zwei verschiedene Ansätze zum High-Speed Datentransfer mit demselben FPGA. Das verwendete FPGA verfügt über integrierte Serialisierer/Deserialisierer-Einheiten (SelectIO) und 16 dedizierte High-Speed Transceiver-Einheiten (GTX). [16] [9] Der serielle Datenstrom wurde in dem verwendeten Messaufbau über eine Rückführung aus dem FPGA herausgeführt und wieder eingespeist. Dieses Konzept der Signalrückführung zu seiner Quelle wird als „Loopback“ bezeichnet. [23]

Da der TLK1221 bei einer seriellen Übertragungsgeschwindigkeit von 1,25Gb/s betrieben wird, sollen beide Ansätze zum besseren Vergleich ebenfalls diese Geschwindigkeit aufweisen. Außerdem soll die Kommunikation wie beim TLK1221 differentiell erfolgen. [7]

Zur einfacheren Implementierung von häufig verwendeten Funktionen stellt der Hersteller Xilinx innerhalb der mitgelieferten Entwicklungsumgebung Vivado fertige Pakete (*LogiCORE IP*) zur Verfügung. Diese beinhalten bereits vollständig implementierte Komponenten in Form von VHDL/Verilog-Templates oder Netzlisten. Diese Komponenten wurden zur Instantiierung der SelectIO-Ressourcen und GTX-Transceiver in das Vivado-Design integriert. Außerdem wurden IPs zur Überprüfung der internen Signale des FPGAs genutzt, um die Funktionalität der Implementierung im Betrieb zu bestätigen. [14] Der allgemeine Aufbau der beiden Ansätze bleibt dabei gleich und ist im Blockschaltbild 4.3 abgebildet.

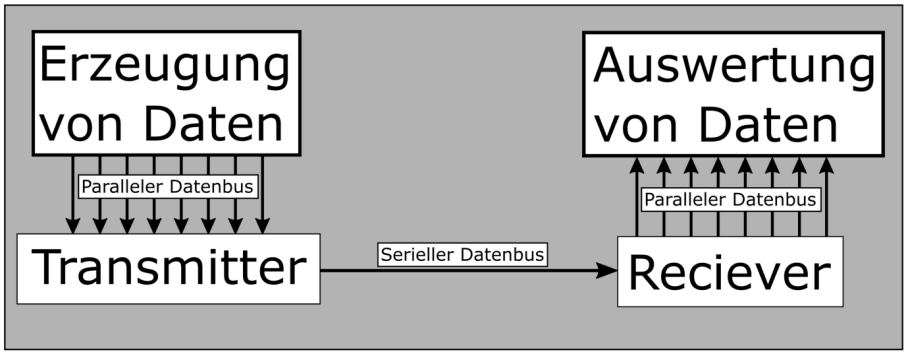

Abbildung 4.3.: Allgemeiner Aufbau der Transceiver-Designs

Das Design besteht aus:

- einem Frame Generator: Erzeugt zu übertragende Beispieldaten
- einem Transmitter: Die Einheit, welche Daten am parallelel Bus serialisiert und Bit für Bit ausgibt
- einem Receiver: Die Einheit, welche die seriell erhaltenen Daten am parallelen Bus ausgibt
- einem Frame Checker: Überprüft die empfangenen Daten auf Korrektheit

4.1.3. Messung

Nach vollständiger Implementierung und Überprüfung der Ansätze auf deren Funktionalität soll die Leistungsaufnahme ermittelt werden. Dazu wurde ein True-RMS Messgerät verwendet, um die anliegende Spannung und den Strom zu messen. Diese Messung wurde getrennt für den Betrieb mit SelectIO und GTX-Transceivern durchgeführt. Die Messergebnisse und die genaue Durchführung der Messung werden in Kapitel 5 wiedergegeben. Die Interpretation der ermittelten Messwerte erfolgt in Kapitel 6.

4.2. Implementierung SelectIO

Alle 7-Series FPGAs verfügen über eine in Silizium integrierte Logik, die das Serialisieren und Deserialisieren von Daten ermöglicht. Jeder I/O Port des FPGAs ist über diese Logik mit der nach außen angebunden. [10, S. 15]

Die folgenden Abschnitte geben die Konfiguration und Schritte der Implementierung der SelectIO-Logik wieder. Alle gezeigten Abbildungen von IP-Konfigurationsfenstern und Blockdesigns stammen aus der Entwicklungsumgebung Vivado 2019.1.

4.2.1. Design

Folgend soll das in Vivado erstellte Block-Design vorgestellt werden.

Das SelectIO-Design besteht aus mehreren Blöcken, von denen jeder eine andere Aufgabe erfüllt. Abbildung 4.4 zeigt das Block-Design und die Verschaltung der Blöcke untereinander.

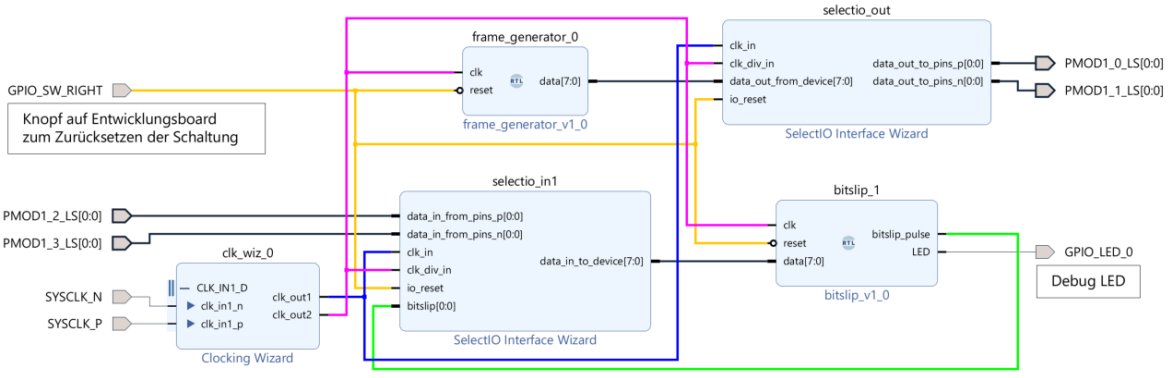

Abbildung 4.4.: Transceiver Design mit SelectIO

Zu erkennen sind unter anderem die beiden SelectIO-Blöcke zum Senden/Empfangen und (De-)Serialisieren von Daten. Der SelectIO_out-Block ist mit den beiden Ports **PMOD1_0_LS** und **PMOD1_1_LS** verbunden, über welche die serielle Kommunikation differentiell erfolgt. Das Pinpaar **PMOD1_2_LS** und **PMOD1_3_LS** bildet den differentiellen seriellen Eingang des SelectIO_in-Blocks. Diese Ports sind auf dem Entwicklungsboard über einen GPIO-Header zugänglich und mit zwei verdrillten Kupferkabeln miteinander verbunden.

Die Konfiguration und Implementierung der SelectIO-Blöcke und allen weiteren abgebildeten Blöcken (Clocking Wizard, Frame Generator und Bitslip) sollen separat in den folgenden Abschnitten thematisch behandelt werden.

4.2.2. SelectIO Interface Wizard

Sofern nicht anders gekennzeichnet wurden alle Informationen über den **SelectIO Wizard** dem Datenblatt [16] entnommen.

Mit Hilfe des **SelectIO Interface Wizards** lässt sich die SelectIO-Logik mit einer grafischen Oberfläche konfigurieren. (Vgl. Abb. 4.5)
Folgend sollen die gewählten Einstellungen im SelectIO Interface Wizard erklärt werden.

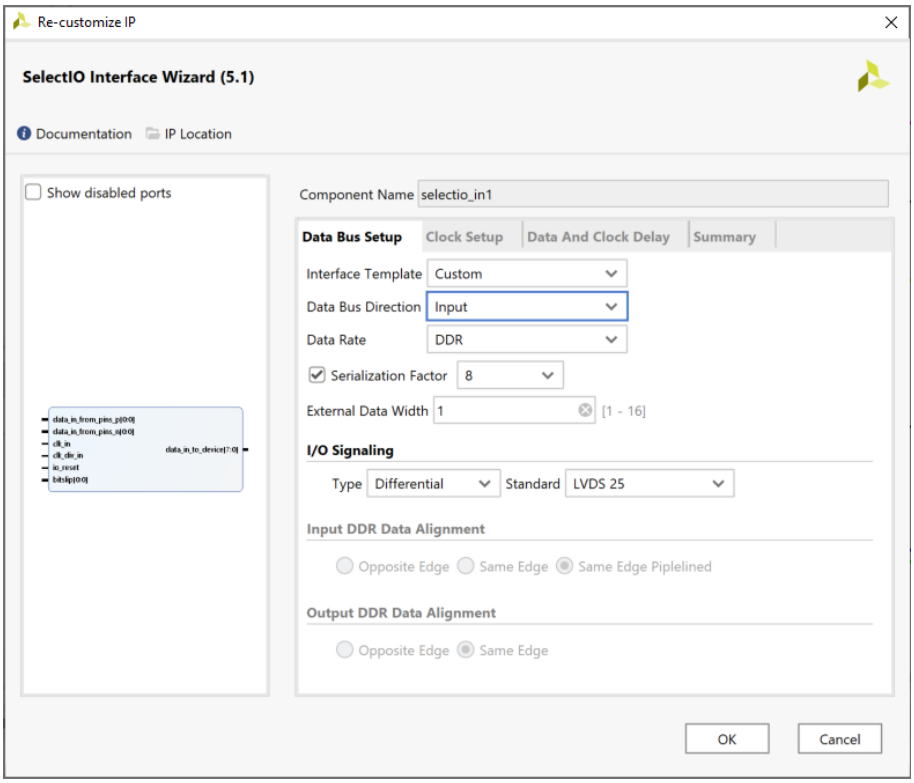

Abbildung 4.5.: SelectIO Interface Wizard

Data Bus Direction:
Die Einstellung **Data Bus Direction** legt die Richtung der Kommunikation fest. Mögliche Einstellungen sind **Input**, **Output**, **Bidirectional** und **Input And Output**. Hierbei wurde die Einstellung **Input**, beziehungsweise **Output** gewählt, wodurch zwei separate SelectIO-Blöcke für den Dateneingang und Datenausgang erzeugt werden.
(Vgl. Abb. 4.6)

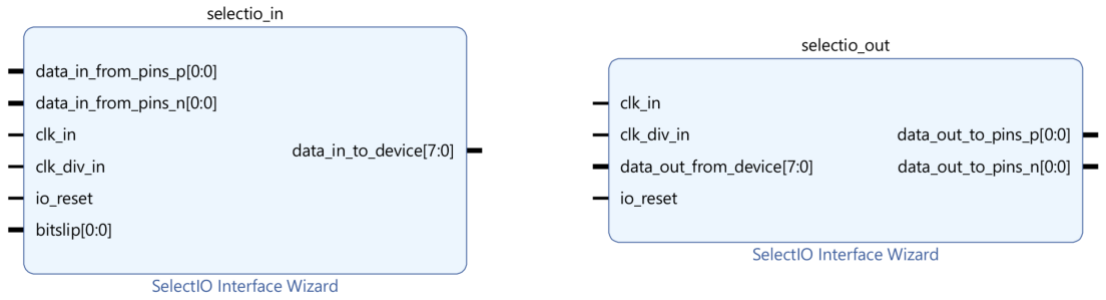

Abbildung 4.6.: SelectIO Input- und Output-Block

Data Rate:
Die Enstellung **Data Rate** legt fest, ob die serielle Übertragung mit Single Data Rate (SDR), oder mit Double Data Rate (DDR) erfolgt. Mit der Einstellung SDR erfolgt die serielle Übertragung eines einzelnen Bits mit jeder steigenden Taktflanke des am CLK Eingang anliegenden Taktsignals. Im DDR Modus erfolgt die Übertragung mit sowohl fallender, als auch steigender Taktflanke und ist somit bei gleicher Taktrate doppelt so schnell wie im SDR-Modus.

Serialization Factor:
Die Einstellung **Serialization Factor** bestimmt den Faktor der zu (de-)serialisierenden Daten. Die Wahl des Wertes ist abhängig von der **Busbreite** der parallel anliegenden Daten und der gewünschten Anzahl an seriellen Datenleitungen, welche über die Einstellung **External Data Width** festgelegt wird.

Die Formel 4.1 stellt den Zusammenhang von Busbreite und External Data Width mathematisch dar:

$$SerializationFactor = \frac{Busbreite}{ExternalDataWidth} \tag{4.1}$$

Bei einer Busbreite von 8 und der Nutzung von nur einer externen Datenleitung ergibt sich somit ein Wert von 8 für den Serialization Factor. Hierbei wird ein differenzielles Leitungspaar als eine Datenleitung gewertet.

I/O Signaling:
Die Einstellung **I/O Signaling** bietet die Wahl des Kommunikationsstandards. Dabei ist die Auswahl zwischen **Single-ended** und **Differential** möglich. (Vgl. Kapitel 2.2) Da als Kommunikationsstandard eine differentielle Kommunikation vorgegeben ist, wurde die Option **Differential** mit dem Standard **LVDS** gewählt.

4.2.3. Clocking

Die zwei generierten SelectIO-Blöcke haben jeweils die beiden Eingänge **CLK** und **CLK_DIV**. (Vgl. Abb. 4.6) Das am **CLK** anliegende Taktsignal gibt die Geschwindigkeit der seriellen Übertragung vor. Das **CLK_DIV** Signal dient dem taktsynchronen Laden, beziehungsweise Ausgeben, der Daten am parallelen Nus des jeweiligen SelectIO-Blocks. Jedes dieser Taktsignale wurde mit Hilfe des LogiCORE IPs Clocking Wizards generiert. Dieser erzeugt aus dem verbauten 200 MHz Taktgeber des Entwicklungsboards die gewünschten Taktsignale. [12] Die Konfiguration der beiden Taktsignale erfolgt nach folgenden Kriterien:

CLK:
Da bei der Konfiguration des SelectIO Transceiver Wizards für die Data Rate der DDR-Modus gewählt wurde, beträgt die resultierende Geschwindigkeit der serielle Übertragung das doppelte der anliegenden Taktrate. So ist bei einer seriellen Übertragungsrate von 1,25 Gb/s ein Taktsignal mit der halben Frequenz von **625 MHz** zu wählen.

CLK_DIV:
Die Wahl der Taktrate des CLK_DIV-Signals ist abhängig von mehreren Faktoren. Dazu gehören die Einstellung der Data Rate (SDR oder DDR), der gewählte Serialisierungsfaktor und die Einstellung der External Data Width.

Der Zusammenhang dieser Faktoren wird in den Formeln 4.2 und 4.3 getrennt nach SDR und DDR-Modus mathematisch dargestellt:

CLK_DIV im SDR-Modus:

$$CLK\_DIV = 1 \cdot CLK \cdot \frac{ExternalDataWidth}{SerializationFactor} \tag{4.2}$$

CLK_DIV im DDR-Modus:

$$CLK\_DIV = 2 \cdot CLK \cdot \frac{ExternalDataWidth}{SerializationFactor} \tag{4.3}$$

Mit den im SelectIO Wizard gewählten Einstellungen (DDR-Modus, Serialization Factor=8, External Data Width=1) und einer CLK-Frequenz von 625 MHz ergibt sich für das CLK_DIV Signal nach Formel 4.3 eine Frequenz von **156,25 MHz**.

4.2.4. Frame Generator

Da in dem aktuellen Aufbau keine Sensordaten vorliegen, müssen diese vor der Übertragung erzeugt werden. Realisiert wurde dies über eine Logik, die mit Hilfe eines 8-Bit Zählers Werte zwischen 0 und 255 am parallelen Bus des SelectIO_out-Blocks erzeugt. (Vgl. Abb. 4.7) Zum Betrieb des Frame Generators wird das CLK_DIV-Signal benötigt. Dieser Zähler wird mit jeder negativen CLK_DIV Taktflanke inkrementiert und zum Zeitpunkt der folgenden steigenden CLK_DIV Takflanke im Buffer des SelectIO_out-Blocks gespeichert. Bevor allerdings die eigentlichen Daten übertragen werden können, wird nach einem Reset eine festgelegte Sequenz für 72 CLK_DIV Perioden ausgegeben. Diese wird zur Initialisierung des SelectIO_in-Blocks benötigt. Der genaue Vorgang der Initialisierung wird im folgenden Abschnitt 4.2.5 erklärt.

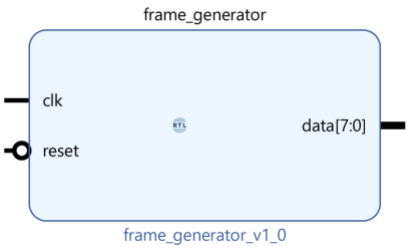

Abbildung 4.7.: Frame Generator

Der VHDL Quellcode des Frame Generators ist im Anhang A.1 einsehbar.

4.2.5. Bitslip

Bitslip bezeichnet die Verschiebung der Daten am parallelen Ausgang des SelectIO_in-Blocks. Diese Verschiebung wird durchgeführt, um die Anordnung des Ausgabewortes festzulegen. Die Bitslip-Operation erfolgt synchron zum CLK_DIV-Takt und verschiebt im SDR-Modus das ausgegebene Wort mit jeder Ausführung um jeweils eine Stelle nach Links. Im DDR-Modus hingegen wird abwechselnd eine Stelle nach Rechts und drei Stellen nach Links geschoben. (Vgl. Abb. 4.8)

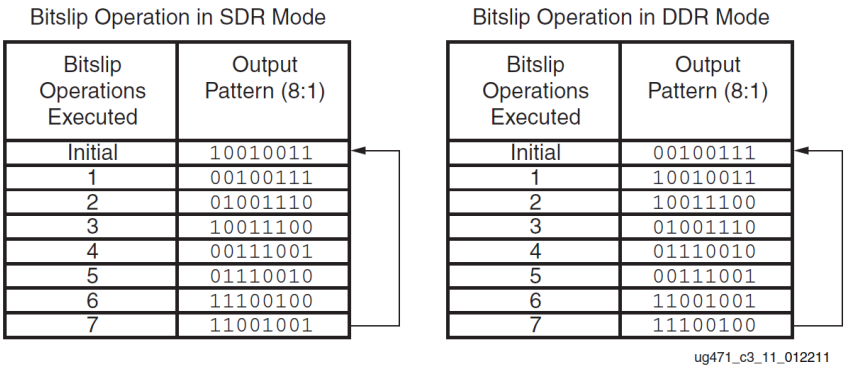

Abbildung 4.8.: Bitslip-Operation [16, S. 159]

Diese Bitslip-Operation wurde mit einer Logik automatisiert, um mit Hilfe eines Trainingspatterns vor Beginn einer Übertragung, oder nach einem Reset die Daten in der richtigen Anordnung auszugeben. Dieses Trainingspattern wird nach einem Reset vom Frame Generator für 72 CLK_DIV Perioden ausgegeben. Als Trainingspattern wurde der Wert x'FE' (=Dezimal 254) gewählt, da die binäre Darstellung des Wertes '11111110' ein leicht wiederzuerkennendes Muster darstellt. Der Bitslip-Block ist über eine Rückführung mit dem parallelen Datenbus des SelectIO_in-Blocks verbunden und überwacht die aktuell ausgegebenen Daten. Ist das Trainingspattern nicht am Ausgang des Blocks Erkennbar (Zeitpunkt 1 in Abb. 4.9) wird automatisch eine Bitslip-Operation ausgeführt. Dazu wird für eine CLK_DIV Periode ein Puls am Bitslip-Ausgang erzeugt (Zeitpunkt 2 in Abb. 4.9). Drei CLK_DIV Perioden danach ist die Bitslip-Operation abgeschlossen und der verschobene Wert ist am Ausgang des SelectIO_in-Blocks erkennbar (Zeitpunkt 3 in Abb. 4.9).

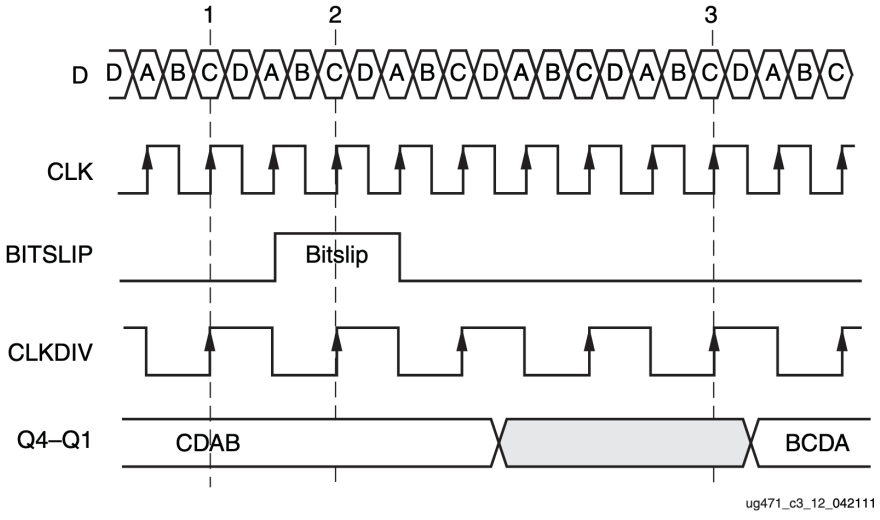

Abbildung 4.9.: Timing der Bitslip-Operation [10, S. 160]

Dieser Vorgang wird so oft wiederholt, bis das Trainingspattern am Ausgang erkennbar ist. Dabei sind bei einem 8-Bit Datenbus maximal 7 Bitslip-Operationen notwendig, um das gewünschte Pattern zu erhalten. Eine achte Bitslip-Operation ergibt wieder die Ursprüngliche Anordnung. Ist zu diesem Zeitpunkt das Trainingpattern noch nicht erkennbar, ist von einem Fehler in der Übertragung auszugehen. [10, S. 158]

Der VHDL Quellcode des Bitslip ist in Anhang A.2 einsehbar.

4.2.6. Simulation

Die Funktionalität des erstellten Designs wurde mit einer Simulation überprüft. Dazu wurde eine Testbench erstellt, die die erforderlichen Signale (Clock und Reset) erzeugt und die externen Datenleitungen der SelectIO-Blöcke miteinander verbindet. (Siehe Abbildung 4.10)

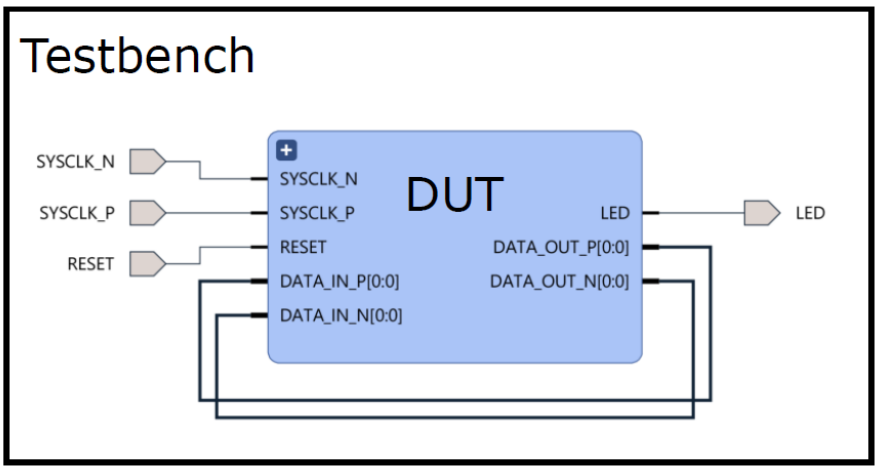

Abbildung 4.10.: Device Under Test (DUT) und Testbench

Zunächst soll überprüft werden, ob die Taktsignale richtig ausgegeben werden. Die Simulation in Abbildung 4.11 zeigt den vom Entwicklungsboard vorgegebenen differentiellen 200 MHz Systemtakt (SYSCLK_P und SYSCLK_N) und die beiden Ausganssignale des Clocking Wizard (CLK und CLK_DIV). Das CLK-Signal besitzt eine Periodendauer von 1,6 Nanosekunden, was umgerechnet einer Frequenz von 625 MHz entspricht. Das CLK_DIV-Signal besitzt die vierfache Periodendauer von 6,4 Nanosekunden und erfüllt somit ebenfalls die Vorgabe von 156,25 MHz.

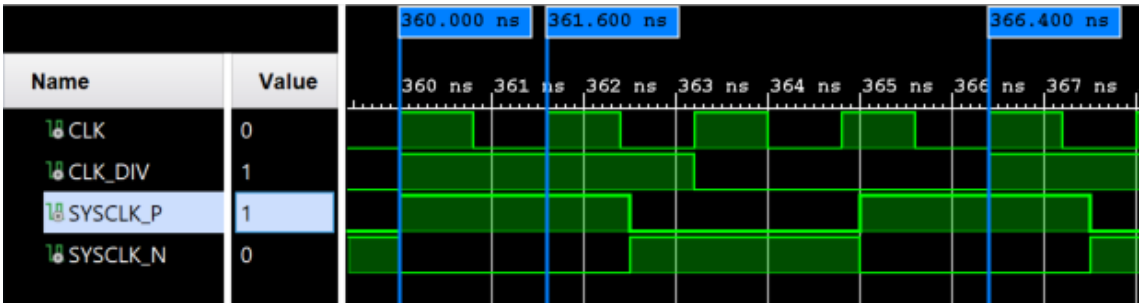

Abbildung 4.11.: Simulation der Ausgangssignale des Clocking Wizards

Da die Taktsignale richtig ausgegeben werden, soll die Funktionalität des **Frame Generators** als Nächstes untersucht werden. Zu Beginn der Simulation ist das Reset Signal 'HIGH'. Nach dem Deaktivieren des Reset gibt der Frame Generator das erwartete Trainingspattern x'FE' aus.

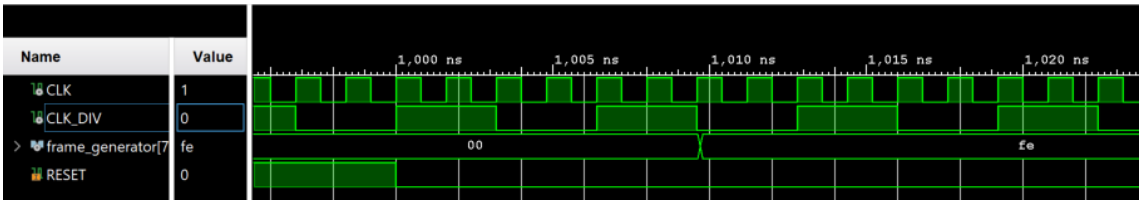

Abbildung 4.12.: Simulation der Init-Sequenz des Frame Generators

Nach 72 CLK_DIV Perioden wird der aktuelle Wert synchron zum CLK_DIV Takt inkrementiert. Dabei wird der aktuelle Wert immer mit der fallenden Flanke geändert. Da der Wert x'FF' (Binär '11111111') den maximalen Wert des 8-Bit Zählers darstellt, ergibt sich mit der Addition von 1 automatisch der Wert x'00'. (Vgl. Abb. 4.13)

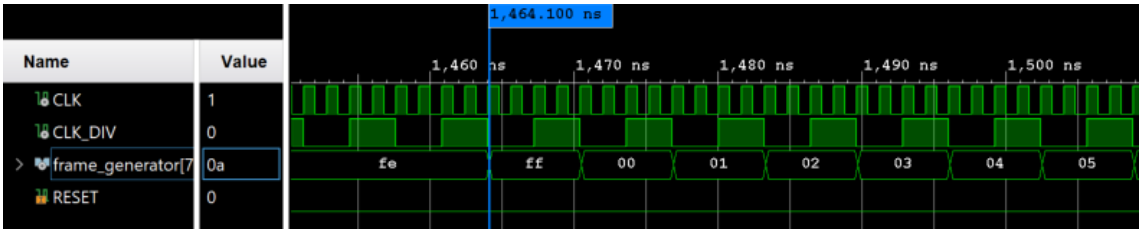

Abbildung 4.13.: Simulation der Zähler-Funktion des Frame Generators

Als Nächstes sollen die beiden SelectIO-Blöcke und die Bitslip-Logik untersucht werden. In Abbildung 4.14 sind unter anderem das Bitslip-Signal, die erzeugten Daten des Frame Generators und der parallele Datenbus des SelectIO_out-Blocks erkennbar. Nach dem

deaktivieren des Reset-Signals überprüft die Bitslip-Logik am Ausgang des SelectIO_in-Blocks mit der nächsten fallenden CLK_DIV-Flanke den ausgegebenen Wert. Nachdem das Ergebnis der Abfrage negativ ist, wird ein Puls über eine CLK_DIV-Periode ausgegeben. Die Bitslip-Operation selber erfolgt mit der darauf folgenden positiven Taktflanke und ist einen Takt später am Ausgang erkennbar. Zur besseren Übersicht wurden die Daten des SelectIO_out-Blocks binär dargestellt. So ist die abwechselnde Verschiebung um drei Stellen nach Rechts und eine nach Links bei der 2. und 3. Bitslip-Operation erkennbar.

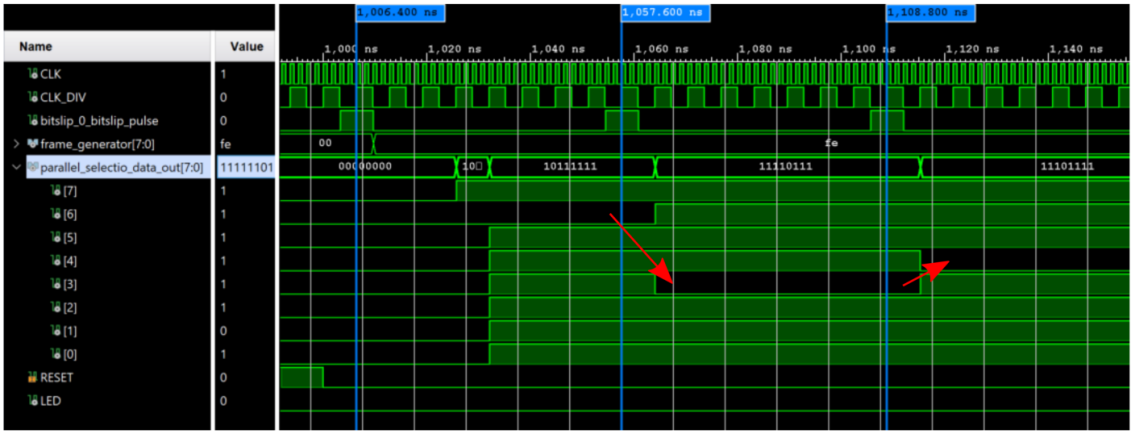

Abbildung 4.14.: Simulation der Bitslip-Operation nach Reset

Nach insgesamt 7 Bitslip-Pulsen erscheint das Trainingspattern x'FE' am Ausgang des SelectIO_in-Blocks. Das Signal der Debug-LED signalisiert mit einem Wechsel auf 'HIGH' den erfolgreichen Abschluss der Init-Sequenz. (Vgl. Abb. 4.15)

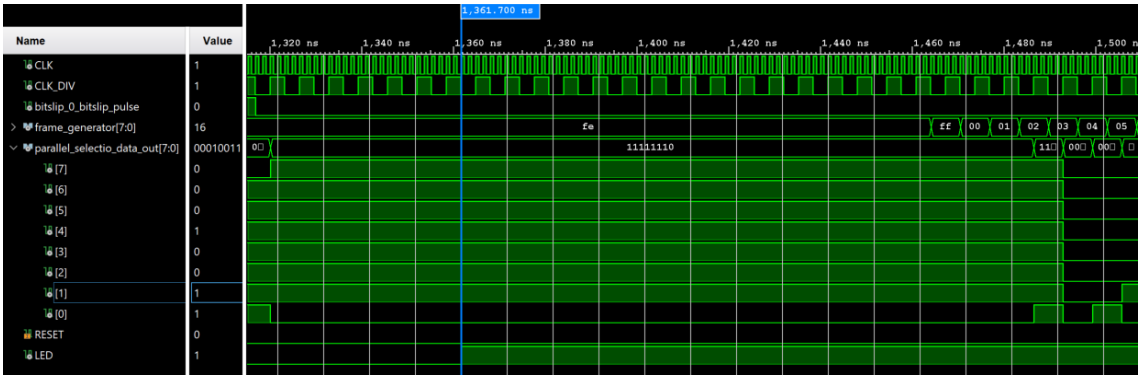

Abbildung 4.15.: Simulation der Bislip-Operation nach abgeschlossener Init-Sequenz

Durch die Simulation konnte somit erfolgreich nachgewiesen werden, dass das erstellte Block-Design die Anforderungen an die SelectIO-Logik erfüllt. Ein Plot des kompletten Signalsverlaufs der Init-Sequenz ist in Anhang C.1 einsehbar. Der Quellcode des SelectIO-Testbench befindet sich in Anhang A.3.

4.2.7. Validierung in Hardware

Nach erfolgreicher Simulation wurde eine Programmierdatei erstellt, um die Hardware des FPGAs zu programmieren. Da allerdings von außen nicht ersichtlich ist, ob die Kommunikationsvorgänge auch wirklich ordnungsgemäß stattfinden, müssen die internen Signale des FPGAs im Betrieb betrachtet werden.

Zu diesem Zweck wurde der **Integrated Logic Analyzer** (ILA) in das Design integriert. Dieser ermöglicht die Betrachtung von bis zu 4096 Signalverläufen über eine Debug-Schnittstelle. [13].

Die Abtastrate des ILA wird durch ein Taktsignal vorgegeben, das neben dem CLK und CLK_DIV mit Hilfe des **Clocking Wizards** erzeugt wird. Da die seriell übertragenen Daten nicht betrachtet werden, sondern nur der parallele Datenbus des SelectIO_in-Blocks, wurde das CLK_DIV-Signal (156,25 MHz) zur Taktung des ILA genutzt.

Da die Signalverläufe nicht in Echtzeit angezeigt werden können, ist das definieren von Triggern zum Starten der Aufzeichnung nötig. Als Trigger wurde hier die fallende Signalflanke des Resetsignals gewählt, um die darauf folgende Initialisierungssequenz mit Bistlip-Operation zu Betrachten. (Vgl. Abb. 4.16)

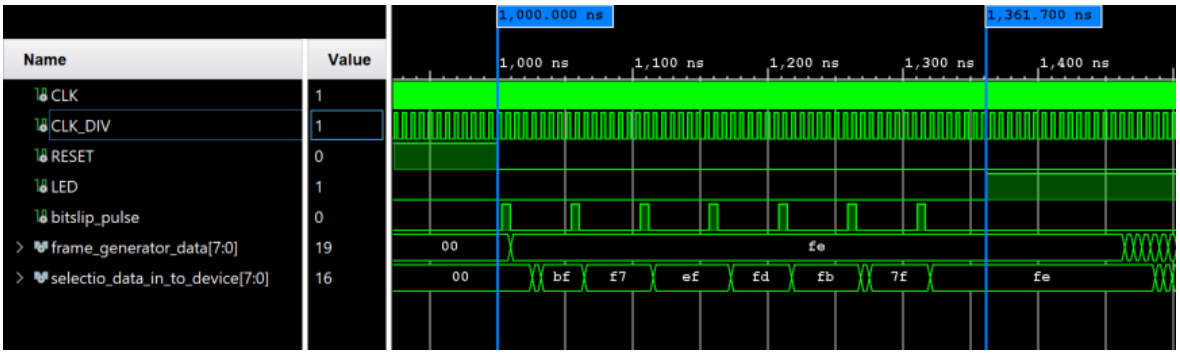

Abbildung 4.16.: ILA Signalverlauf nach Reset

Es ist zu erkennen, dass die Bitslip-Operation erfolgreich abgeschlossen werden konnte und das Tainingspatter x'FE' am Ausgang des SelectIO_in-Blocks ausgegeben und erkannt wurde. Zusätzlich gibt eine Leuchtdiode auf dem Entwicklungsboard eine visuelle Rückmeldung über den Status der Kommunikation.

Nach Abschluss der Initialisierungssequenz beginnt der Frame_Generator den Zähler zu inkrementieren. Mit einer Verzögerung von 3 CLK_DIV-Perioden sind auch diese Werte am Ausgang des SelectIO_out-Blocks erkennbar. (Vg. Abb. 4.17)

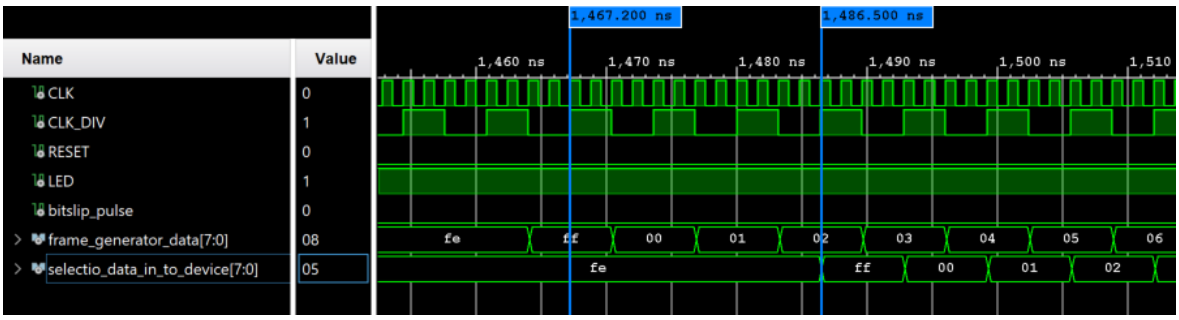

Abbildung 4.17.: ILA Signalverlauf nach Initialisierungssequenz

Somit kann angenommen werden, dass die SelectIO-Transceiver ordnungsgemäß funktionieren und das System bereit ist für die Messung der Leistungsaufnahme.

4.3. Implementierung GTX High-Speed Transceiver

Die 7 Series FPGAs von Xilinx verfügen neben der SelectIO-Logik auch über dedizierte Gigabit Transceiver-Einheiten, welche je nach Familie Geschwindigkeiten von bis zu 28.5 Gb/s erreichen. Dabei sind die Transceiver für jede der vier FPGA Familien unterschiedlich ausgelegt. Das hier verwendete FPGA ist Teil der Kintex-7 Familie und besitzt 16 integrierte GTX-Transceiver, die eine maximale Übertragungsgeschwindigkeit von 12,5 Gb/s erreichen. [8, S. 11]

Die folgenden Abschnitte geben die Konfiguration und Schritte der Implementierung der integrierten GTX-Transceiver wieder. Alle verwendeten Abbildungen von IP-Konfigurationsfenstern und Blockdesigns stammen aus der Entwicklungsumgebung Vivado 2019.1. Informationen zur Konfiguration des Aurora 8B/10B wurden dem Datenblatt [11] entnommen.

4.3.1. Design

Das Grundlegende GTX-Transceiver Design basiert auf einem von Xilinx zur Verfügung gestellten Example-Design, das für das verwendete Entwicklungsboard bereits voll funktionsfähig vorliegt. Zur Erzeugung dieses Example-Designs wurde der Xilinx IP **Aurora 8B/10B** verwendet, dessen Nutzen und Konfiguration im Kapitel 4.3.2 näher erläutert wird. [11, S. 87] Hierbei sind bereits ein Block zum Erzeugen von Daten (FRAME_GEN) und ein Block zum Überprüfen der empfangenen Daten (FRAME_CHECK) vorhanden. Die Abbildung 4.18 gibt den logischen Aufbau des Designs wieder. Eine ausführliche Abbildung des Block-Designs befindet sich im Anhang B.2.

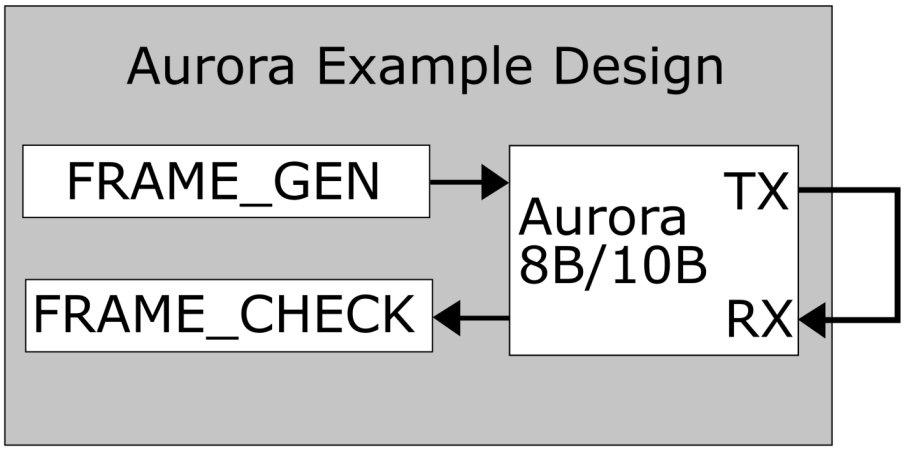

Abbildung 4.18.: Aurora Example Design

Die vom FRAME_GEN erzeugten Daten werden parallel an den Aurora 8B/10B-Block übergeben. Dort werden die Daten mit einer 8B/10B Kodierung kodiert und seriali-

siert. Die Verbindung zwischen Transmitter und Reciever findet in einer Loopback-Konfiguration statt. Dazu werden zwei Koaxialkabel zur differenziellen Übertragung genutzt, welche über die SMA-Stecker des Entwicklungsboards direkt mit einem der 16 GTX-Transceiver verbunden sind.

4.3.2. Aurora 8B/10B

Zur Implementierung der GTX-Transceiver wurde der Xilinx IP **Aurora 8B/10B** genutzt. Ähnlich wie bei SelectIO ist die Konfiguration über ein grafisches Interface möglich. (Vgl. Abb. 4.19)

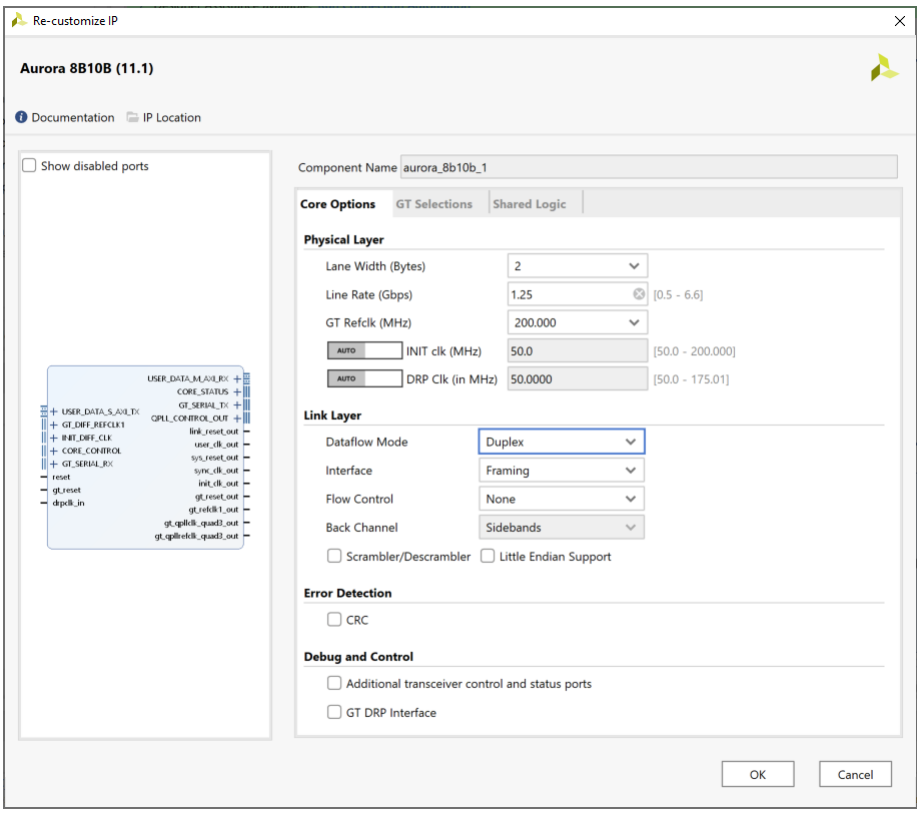

Abbildung 4.19.: Aurora 8B/10B Wizard

Folgend soll die Konfiguration des Aurora 8B/10B IPs erläutert werden.

Lane Width:
Die Einstellung **Lane Width** legt die Breite des parallelen Datenbusses in Byte fest. Dabei sind die Einstellungen 2 und 4 möglich. Hier wurde eine Busbreite von 2 Byte, also 16 Bit gewählt.

Line Rate:
Line Rate beschreibt die Einstellung der Übertragungsgeschwindigkeit der seriell übertragenen Daten. Durch die vorherige 8B/10B Kodierung beträgt die tatsächliche Datenrate der Nutzdaten:

$$DATA\_RATE = (0.8 \cdot LineRate) \tag{4.4}$$

GT Refclk:
Das Taktsignal zum Betreiben der GTX-Transceiver wird nicht innerhalb des FPGAs erzeugt und muss daher von außen eingespeist werden. Die Einstellung **GT Refclk** gibt die Taktrate dieses eingespeisten Clocksignals an. Dabei kann das eingespeiste Signal auch langsamer sein als die gewählt *Line Rate*. Eine interne Logik erzeugt aus diesem Signal die benötigten Taktsignale zum Betrieb der GTX-Transceiver.

INIT und DRP Clk: Der DRP (Dynamic Reconfiguration Port) und INIT Clock werden benötigt, um den Reset-Eingang zu entprellen und eine minimale Funktion ohne GT Refclk zu gewährleisten. Hier wurde der im Datenblatt angegebene Standardwert von 50 MHz für beide Einstellungen übernommen.

Dataflow Mode:
Die Einstellung **Dataflow Mode** legt die Richtung der Datenübertragung fest. Hier wurde die Einstellung „Duplex“ gewählt, um einen Block zu erzeugen, der sowohl als Transmitter als auch als Receiver fungiert.

4.3.3. Clocking

Zum Betreiben der internen GTX-Transceiver wird ein differentielles Clock-Signal benötigt, das auf externer Hardware erzeugt und in das FPGA eingespeist werden muss. [1, S. 36] Jedes der vier Transceiver-Quads besitzt zwei Clock-Eingänge zum Betrieb der Transceiver. Einer dieser Clock-Eingänge ist über ein differentielles SMA-Stecker-Paar auf dem Entwicklungsboard zugänglich. (Vgl. Abb. 4.2) Da im Rahmen der Bachelorarbeit jedoch kein externer Taktgenerator zur Verfügung stand, wurde dieses Signal mit dem Clocking Wizard IP intern erzeugt. Da ein differentielles Signal benötigt wird, wurde der IP **Utility Buffer** genutzt um das interne Single-Ended Signal über ein differentielles Pin-Paar nach außen zu führen und in den Clock-Eingang des GTX-Transceivers einzuspeisen. [15] Die Abbildung 4.20 zeigt den verwendeten IP.

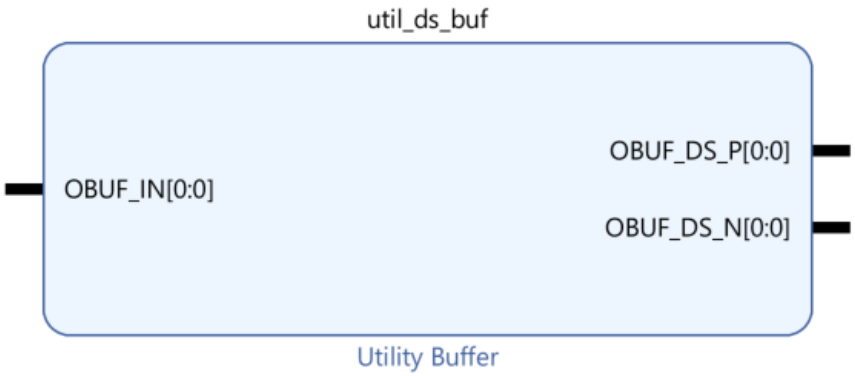

Abbildung 4.20.: Utility Buffer LogiCORE IP

Ein weiteres Taktsignal wird benötigt, um die vom Nutzer implementierte Logik synchron zu betreiben (user_clk). Dieses Taktsignal wird intern vom Aurora 8B/10b-Block erzeugt und beträgt im vorliegenden Fall 62,5MHz.

4.3.4. FRAME_GEN und FRAME_CHECK

Mit Hilfe des FRAME_GEN werden Beispieldaten zur seriellen Übertragung erzeugt. Ein Pseudo-Zufallsgenerators erzeugt Datenframes mit einem Linear-Feedback Shift Register (LFSR), das aus einem rückgekoppelten Schieberegister mit XOR-Verknüpfungen besteht. Da es sich allerdings um Pseudo-Zufallswerte handelt, sind die entstehenden Bitfolgen vorhersagbar. Dieser Umstand wird vom FRAME_CHECK genutzt, um die empfangenen Daten auf Korrektheit zu überprüfen. Dazu erzeugt dieser ebenfalls Datenframes mit einem LFSR und vergleicht diese mit den empfangenen Datenframes. Stimmen diese beiden Datenframes nicht miteinander überein, wird ein 8-Bit Zähler inkrementiert, um die Anzahl der aufgetretenen Übertragungsfehler anzugeben.

Folgend ist der Ausschnitt des VHDL-Quellcodes zur Erzeugung der Pseudo-Zufallswerte mit einem LFSR abgebildet. Dieser ist Teil des Aurora 8B/10B Example-Designs von Xilinx und wird sowohl für den FRAME_GEN als auch den FRAME_CHECK verwendet. [11, S. 87]

```
process(USER_CLK)
    begin
        if(USER_CLK'event and USER_CLK='1') then
            if(reset_c = '1') then
                data_lfsr_r     <=  X"ABCD" after DLY;  --random seed value
            elsif( (not TX_DST_RDY_N and not idle_r)='1' ) then
                data_lfsr_r <= (not(data_lfsr_r(3) xor data_lfsr_r(12) xor data_lfsr_r(14)
                                xor data_lfsr_r(15)) & data_lfsr_r(0 to 14)) after DLY;
            end if;
        end if;
    end process;
```

4.3.5. Simulation

Um das Example-Design auf Funktionalität zu überprüfen, sollen die erzeugten Signale in einer Simulation betrachtet werden. Da dieses Design bereits voll funktionsfähig vorliegt, sollen nur die Ein- und Ausgangssignale von FRAME_GEN und FRAME_CHECK überprüft werden, um eine ordnungsgemäße Übertragung mit GTX-Transceivern zu verifizieren. Das Example-Design beinhaltet bereits eine fertige Testbench, die zu diesem Zweck ausgeführt wurde. Abbildung 4.21 zeigt den Verlauf der Signale des FRAME_GEN und FRAME_CHECK.

Abbildung 4.21.: Simulation der Übertragung mit GTX

Es ist zu erkennen, dass das erzeugte Bitmuster des FRAME_GEN mit einer gewissen Verzögerung am Eingang des FRAME_CHECK anliegt. Der Zählerstand ERR_COUNT ist dabei durchgehend 0, was eine fehlerfreie Datenübertragung signalisiert.

4.3.6. Validierung in Hardware

Zur Verifizierung der Funktionalität der Schaltung im Betrieb wurden die internen Signale nach Programmierung des FPGAs mit dem ILA untersucht. Dazu wurden die erzeugten Datenframes des FRAME_GEN und und die Ausgangssignale nach der Übertragung mit den GTX-Transceivern betrachtet. Abbildung 4.22 zeigt die Signalverläufe von FRAME_GEN und FRAME_CHECK.

Abbildung 4.22.: ILA Waveform mit GTX

Es ist zu erkennen, dass die erzeugten Daten beim Empfänger ankommen. Der Zähler des FRAME_CHECK ist durchgehend 0, was bedeutet, dass während der ganzen Übertragung keine Fehler festgestellt wurden. Auch ist die erzeugte Sequenz des

FRAME_GEN mit einer Verzögerung von 39 user_clk-Perioden am Eingang des FRAME_CHECK erkennbar. Somit konnte nachgewiesen werden, dass die Übertragung mit GTX-Transceivern voll funktionsfähig ist.

5. Messung der Leistungsaufnahme

5.1. Durchführung der Messung

Nachdem die beiden Ansätze mit SelectIO und GTX vollständig implementiert wurden und deren Funktionsfähigkeit nachgewiesen wurde, kann die Messung der Leistungsaufnahme durchgeführt werden.

Zur Messung der Leistungsaufnahme wurde ein True-RMS Multimeter genutzt, um Strom und Spannung im Betrieb zu bestimmen. Dabei wurde die Spannung des Versorgungsnetzteils als konstant betrachtet. Da der Strom jedoch durch kapazitive/induktive Einflüsse womöglich variiert, würde eine Multiplikation dieser beiden Werte nur die Scheinleistung ergeben. Deshalb wurde ein Multimeter verwendet, das über über eine Funktion zur Mittelwertbildungn im DC-Messbereich verfügt und dadurch die Berechnung der Wirkleistung durch Multiplikation dieser beiden Werte erlaubt:

$$P = U \cdot I \tag{5.1}$$

Zur Messung der Stromstärke wurde die 12 Volt-Leitung zwischen dem Netzteil und dem Entwicklungsboard aufgetrennt und an den beiden Enden mit Bananensteckern verlötet. So können die Sonden des Messgeräts mit den beiden Leitungsenden zur Strommessung über eine Steckverbindung verbunden werden. (Vgl. Abb. 5.1)

Die Spannungmessung wurde zwischen der 12 Volt-Leitung und der Ground-Leitung am Versorgungsstecker des Entwicklungsboards durchgeführt. (Vgl. Abb. 4.1) Somit handelt sich um eine spannungsrichtige Messung, wobei ein minimaler Strom durch das Voltmeter fließt und den abgelesenen Wert der Stromstärke verfälscht. Dieser Messfehler ist allerdings vernachlässigbar, da der Strom durch das Voltmeter aufgrund eines Innenwiderstand von mehreren Megaohm kaum messbar ist und somit nur um einige Mikroampere abweicht. [26] Abbildung 5.2 zeigt die Unterschiede der spannungsrichtigen und stromrichtigen Messung.

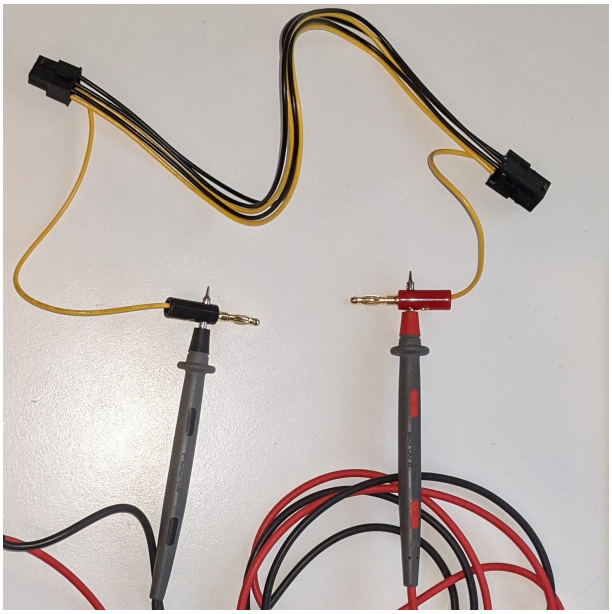

Abbildung 5.1.: Versorgungskabel mit Bananenstecker

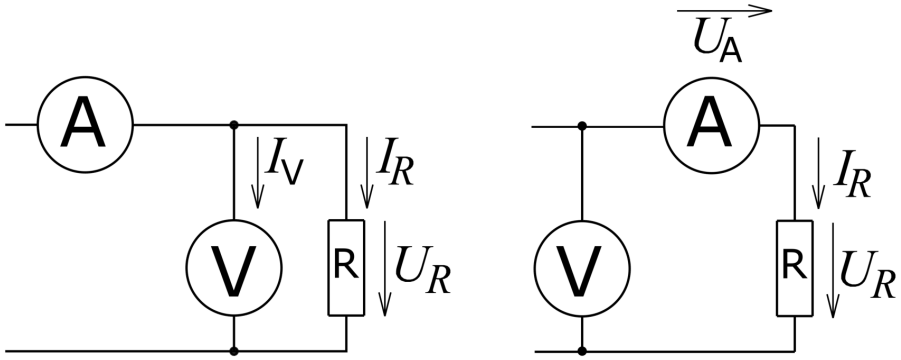

Abbildung 5.2.: Spannungsrichtiger (links) und stromrichtiger (rechts) Messaufbau [26]

Da auf dem Entwicklungsboard einige Komponenten verbaut sind (LEDs, Lüfter, Spannungswandler), die auch wenn nicht genutzt einen Teil zur Erhöhung des gesamten Leistungsverbrauchs beitragen, wurden zwei Messungen durchgeführt, um den Beitrag der Transceiver zu bestimmen. Dazu wurde die Leistungsmessung einmal mit implementierten, aber inaktiven Transceivern durchgeführt (es werden keine Daten erzeugt/übertragen) und einmal im Normalbetrieb (Daten werden erzeugt und übertragen). Diese Zustände wurden als **Aktiv** und **Inaktiv** bezeichnet. Die Differenz aus diesen Ergebnissen wurde als die absolute Leistungsaufnahme der Transceiver interpretiert.

Um während der Messung die Zustände der Transceiver (**Aktiv** oder **Inaktiv**) steuern zu können, wurde der im Design genutzte Hardware-Knopf zum Zurücksetzen der Schaltung durch den LogiCORE IP **VIO** (Virtual Input/Output) ersetzt. Durch diesen

IP-Core können Signale innerhalb der Programmierumgebung Vivado über die Debug-Schnittstelle des Entwicklungsboards gesteuert werden. Gleichzeitig können über einen Eingang Signale eingelesen werden und so Informationen über den Zustand des Transceivers liefern. (Vgl. Abb. 5.3)

Abbildung 5.3.: VIO LogiCORE IP

In den folgenden Abschnitten 5.2 und 5.3 sollen die Durchführung und Ergebnisse der Messungen der Leistungsaufnahme mit SelectIO und GTX bei einer seriellen Übertragungeschwindigkeit von 1,25 Gb/s wiedergegeben werden.

5.2. SelectIO

Vor der Durchführung der Messung der Leistungsaufnahme wurde das SelectIO-Design um einen VIO-Core ergänzt, um die Reset-Funktion über Vivado zu steuern. Dazu wurde das CLK_DIV Signal des Clocking Wizards mit dem clk-Eingang des VIO-Cores verbunden. Zusätzlich wurde das Signal der Debug-LED mit einem Eingang des VIO-Cores verbunden, um zu überprüfen, ob die Initialisierungssequenz erfolgreich abgeschlossen werden konnte und die Übertragung ordnungsgemäß abläuft. Gleichzeitig wurde so der Einfluss der Debug-LED eliminiert, da die LED ebenfalls zur gesamten Leistungsaufnahme beiträgt. Ein Blockschaltbild des resultierenden Designs ist im Anhang B.1 einsehbar.

Die Messung der Leistungsaufnahme wurde in den beiden Zuständen **Aktiv** und **Inaktiv** durchgeführt. In der folgenden Tabelle 5.1 sind die Ergebnisse der Messungen getrennt nach Zustand aufgeführt:

Zustand	Strom [mA]	Spannung [V]	Leistung [W]
Aktiv	643,8	11,776	7,5814
Inaktiv	640,8	11,782	7,5499

Tabelle 5.1.: Leistungsaufnahme der Implementierung mit SelectIO

Die Interpretation und Diskussion dieser Messergebnisse erfolgt in Kapitel 6.

5.3. GTX

Zur Messung der Leistungsaufnahme der integrierten GTX High-Speed Transceiver-Einheiten wurde ebenfalls ein VIO-Core in das Design integriert. Dieser wird bei der Erzeugung des Example-Deisgns mit den Einstellungen **Vivado Lab Tools** und **Additional transceiver control and status ports** im Aurora 8B/10B Wizard automatisch in das Design integriert. Damit lässt sich der power_down-Eingang des Aurora 8B/10B-Blocks steuern, um die Transceiver in den Zustand „Inaktiv“ zu versetzen.

Die Ergebnisse der Messung der Leistungsaufnahme in den Zuständen **Aktiv** und **Inaktiv** sollen in der folgenden Tabelle wiedergegeben werden:

Zustand	Strom [mA]	Spannung in [V]	Leistung [W]
Aktiv	674,4	11,677	7,8750
Inaktiv	648,0	11,688	7,5738

Tabelle 5.2.: Leistungsaufnahme mit implementiertem GTX-Transceiver

Die Interpretation und Diskussion dieser Messergebnisse erfolgt in Kapitel 6.

6. Diskussion der Ergebnisse

Um einen direkten Vergleich zwischen dem externen SerDes TLK1221 von Texas Instruments und integrierten High-Speed Transceiver-Einheiten herstellen zu können, wurden zwei verschiedene Ansätze zur High-Speed Datenübertragung mit einem FPGA der Firma Xilinx untersucht. Dazu wurden die beiden Ansätze mit SelectIO (vgl. Kapitel 4.2) und GTX (vgl. Kapitel 4.3) einer Messung der Leistungsaufnahme unterzogen. Die Ergebnisse dieser Messungen aus Kapitel 5 sollen nun interpretiert und diskutiert werden.

Tabelle 6.1 zeigt die absolute Leistungsaufnahme des FPGAs unterteilt nach Zustand (Aktiv/Inaktiv) und implementiertem Transceiver (SelectIO/GTX). Diese Werte wurden den Tabellen 5.1 und 5.2 entnommen.

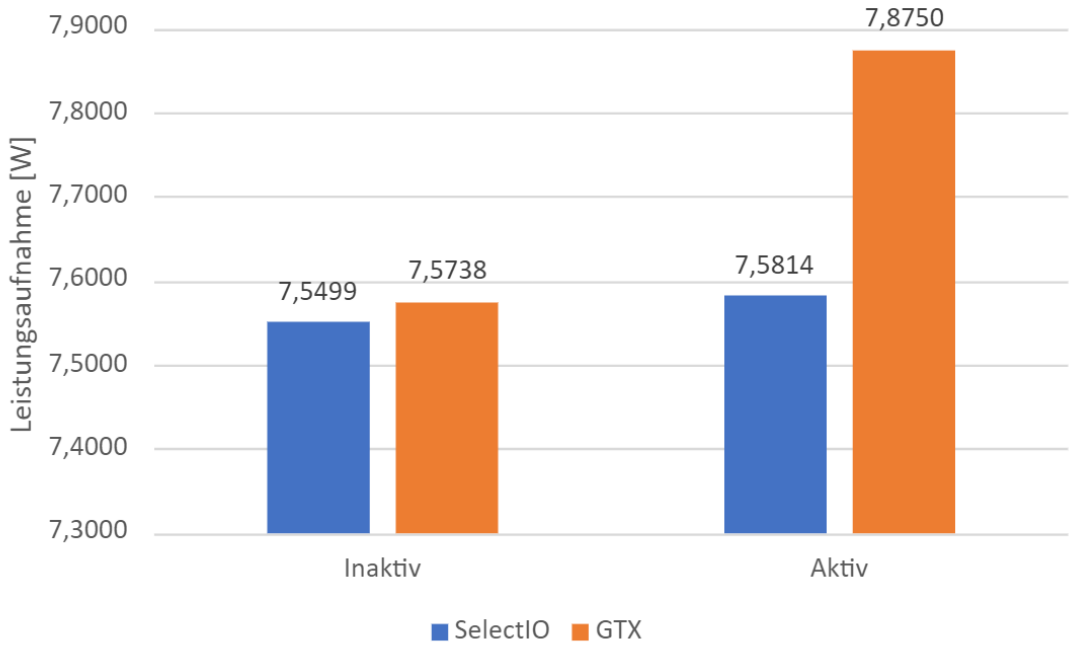

Abbildung 6.1.: Absolute Leistungsaufnahme des FPGAs nach Zustand der Transceiver

Es ist zu erkennen, dass die Implementierung mit SelectIO sowohl im aktiven, als auch im inaktiven Zustand eine geringere Leistungsaufnahme verursacht als die Implementierung mit GTX. Allerdings ist dabei zu beachten, dass der Grundverbrauch der beiden

Implementierungen im inaktiven Zustand bereits unterschiedlich ist. So ist der Grundverbrauch mit implementierten GTX-Transceivern 23,9mW höher als bei SelectIO, ohne dass eine Datenübertragung stattfindet. Da allerdings nur der absolute Verbrauch der integrierten Transceiver-Einheiten betrachtet werden soll, wurde die Differenz zwischen den Zuständen „Aktiv“ und „Inaktiv“ gebildet. Dieser Wert wurde als die absolute Leistungsaufnahme der integrierten Transceiver-Einheiten interpretiert. Abbildung 6.2 zeigt die absolute Leistungsaufnahme der beiden im FPGA integrierten Transceiver-Einheiten und des TLK1221. Der für den TLK1221 abgebildete Wert wurde dem Datenblatt entnommen und entspricht der Leistungsaufnahme bei einer Übertragungsgeschwindigkeit von 1,25Gb/s. [7]

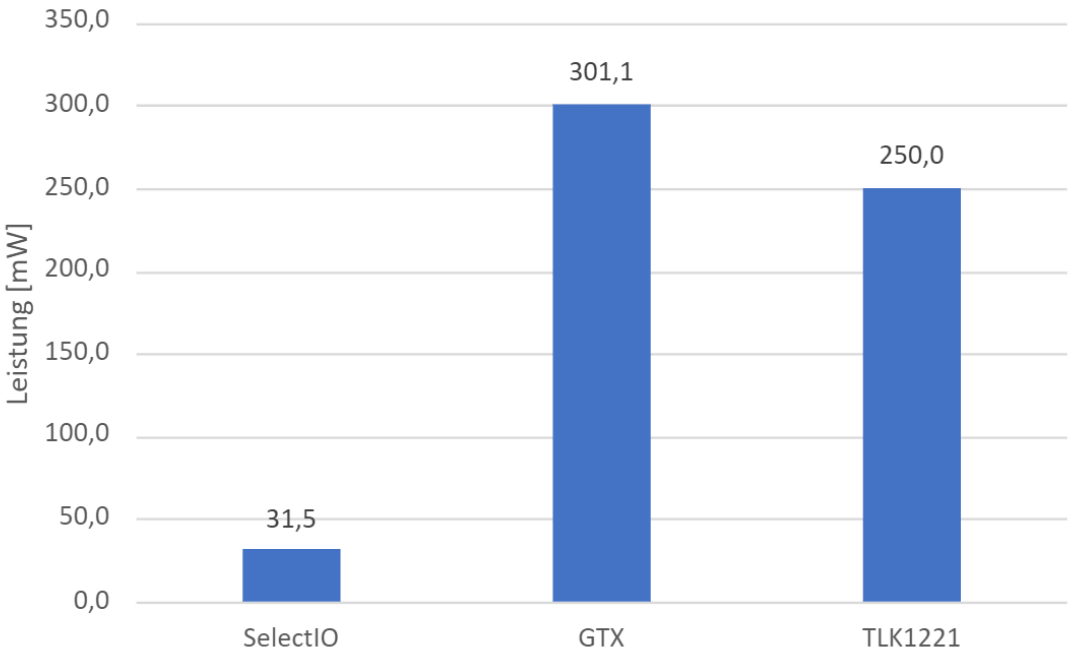

Abbildung 6.2.: Absolute Leistungsaufnahme der Transceiver-Einheiten

Mit 31,5mW beträgt die Leistungsaufnahme der Implementierung mit SelectIO nur 12,6% der Leistungsaufnahme des TLK1221. Die Implementierung mit GTX hingegen weist eine Leistungsaufnahme von 301,1mW auf und nutzt somit 120% der Leistung des TLK1221. Allerdings muss dabei berücksichtigt werden, dass aus den drei Varianten nur die GTX-Transceiver über eine in Silizium integrierte Logik zur 8B/10B Kodierung und Dekodierung verfügen. [9, S. 116] Dies könnte den 18% höheren Verbrauch der GTX-Transceiver gegenüber dem TLK1221 erklären.

Der Unterschied in der Leistungsaufnahme zwischen SelectIO und GTX/TLK1221 könnte darin begründet sein, dass sowohl TLK1221 als auch die GTX-Transceiver über eine integrierte Logik zur *Clock Recovery* und *Comma Detection*. (Vgl. Datenblatt [9, S. 116] und [7, S. 12])

Bei den betrachteten Messergebnissen muss jedoch beachtet werden, dass die Leistungsaufnahme des Entwicklungsboard mit allen verbauten Komponenten vorgenommen wurde. Dadurch ist eine Aussage über die Leistungsaufname des FPGAs alleine nicht möglich. Um eine genauere Messung durchzuführen, könnte eine Platine entwickelt werden die nur über die benötigte Peripherie verfügt und weitere Komponenten wie einen Lüfter über eine externe Energieversorgung speist. Jedoch erlauben die durchgeführten Messungen eine quantitative Aussage zu den jeweiligen Implementierungen.

7. Fazit

Bei einer isolierten Betrachtung der absoluten Leistungsaufnahmen der Transceiver stellt die Implementierung mit SelectIO die bevorzugte Variante dar. Allerdings verfügt SelectIO im Gegensatz zum TLK1221 und den GTX-Transceivern über keine integrierte Logik zur Clock Recovery und Comma Detection. Im Betrieb mit physisch getrenntem Sender und Empfänger müsste somit bei der Nutzung von SelectIO das Clock-Signal über einen zusätzlichen Leiter übertragen werden.

Die fehlende Logik zur Clock Recovery bei SelectIO führt dazu, dass in den bereits vorhandenen Systemen die Datenkabel ersetzt werden müssten. Da dies jedoch nicht erwünscht ist, stellt dieser Umstand ein Ausschlusskriterium für die Nutzung von SelectIO dar. Vorstellbar wäre jedoch der Einsatz von SelectIO zur Kommunikation mit anderem Komponenten auf der selben Platine.

Die fehlende Integration einer Logik zur 8B/10B (De-)Kodierung beim TLK1221 bedeutet einen größeren Platzaufwand bei der Entwicklung von Platinen. Gerade in der Robotik ist jedoch eine kleine Platinengröße von Vorteil. Durch die Integration der Logik zur 8B/10B-Kodierung stellt die Nutzung von GTX-Transceivern somit die kompakteste Lösung aus den betrachteten Systemen dar. Außerdem sind mit den GTX-Transceivern insgesamt höhere Geschwindigkeiten möglich. Durch die Funktionalität zur Post-/Pre-Emphasis ist ein Entwicklungssprung auf höhere Datenraten bei einer gleichzeitig größeren physischen Distanz der Systeme möglich.

Damit stellen die integrierten GTX-Transceiver des Kintex 7 FPGAs von Xilinx eine zukunftssichere Lösung dar und bieten sich zum Einsatz in den robotischen Systemen des DLR an.

Auf die Ergebnisse dieser Arbeit kann bei der Implementierung des Spacewire-Protokolls mit Xilinx FPGAs aufgebaut werden. Das Block-Design ist modular und kann um die gewünschten Funktionalitäten erweitert werden. Für eine umfassendere Bewertung aktueller Technologien zur High-Speed Datenübertragung mit integrierten Transceiver-Einheiten ist jedoch die Untersuchung von FPGAs anderer Hersteller möglich. Ein Vertiefung in die allgemeine Funktionsweise von Transceivern war jedoch durch die ausführliche Auseinandersetzung mit nur einem FPGA gut möglich.

A. Quellcode

A.1. SelectIO FRAME_GENERATOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity frame_generator is
    Port (
        clk, reset : in std_logic;
        data : out std_logic_vector(7 downto 0)
     );
end frame_generator;

architecture Behavioral of frame_generator is
signal buff : unsigned(7 downto 0);
signal counter : unsigned(6 downto 0);

begin
    process(clk)
    begin
        if(falling_edge(clk)) then
            if(reset = '1') then
                counter <= to_unsigned(0, counter'LENGTH);
                buff <= to_unsigned(0, buff'LENGTH);
                data <= "00000000";
            else
                if counter < 71 then
                    buff <= x"FE";
                    counter <= counter + 1;
                else
                    buff <= buff + 1;
                end if;

                data <= std_logic_vector(buff);

            end if;
        end if;


    end process;


end Behavioral;
```

A.2. SelectIO Bitslip

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity bitslip is
  Port (
    bitslip_pulse, LED : out std_logic;
    clk, reset : in std_logic;
    data : in std_logic_vector(7 downto 0)
    );
end bitslip;

architecture Behavioral of bitslip is
signal init_done : std_logic := '0';
--zaehler zur verzoegerung der Bitslip-Pulse
signal counter : unsigned(2 downto 0) := to_unsigned(7, counter'LENGTH);
begin
    process(clk, reset)
    begin

        if(falling_edge(clk)) then
            if(reset = '1') then
                init_done <= '0';
                counter <= to_unsigned(7, counter'LENGTH);
                LED <= '0';
                bitslip_pulse <= '0';

            elsif(init_done = '0') then
                if(counter = 0) then
                    if(data /= x"FE") then
                        bitslip_pulse <= '1';
                        counter <= to_unsigned(7, counter'LENGTH);

                    else
                        init_done <= '1';
                    end if;
                else
                    bitslip_pulse <= '0';
                    counter <= counter - 1;
                end if;

           else
                bitslip_pulse <= '0';

           end if;

        end if;
        LED <= init_done;
    end process;

end Behavioral;
```

A.3. SelectIO Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity selectio_tb is
end selectio_tb;

architecture Behavioral of selectio_tb is

COMPONENT selectio
 port (
    SYSCLK_P : in STD_LOGIC;
    SYSCLK_N : in STD_LOGIC;
    DATA_OUT_P : out STD_LOGIC_VECTOR ( 0 to 0 );
    DATA_OUT_N : out STD_LOGIC_VECTOR ( 0 to 0 );
    RESET : in STD_LOGIC;
    LED : out STD_LOGIC;
    DATA_IN_P : in STD_LOGIC_VECTOR ( 0 to 0 );
    DATA_IN_N : in STD_LOGIC_VECTOR ( 0 to 0 )
  );
END COMPONENT;



signal DATA_P, DATA_N : std_logic_vector(0 downto 0);
signal SYSCLK_N, SYSCLK_P, RESET, LED : std_logic;
CONSTANT CLK_PERIOD : time := 5ns;
CONSTANT CLK_DIV_PERIOD : time := 20ns;

begin

DUT : selectio
port map (
      DATA_IN_N => DATA_N,
      DATA_IN_P => DATA_P,
      DATA_OUT_N => DATA_N,
      DATA_OUT_P => DATA_P,
      LED => LED,
      RESET => RESET,
      SYSCLK_N => SYSCLK_N,
      SYSCLK_P => SYSCLK_P
    );

process
    begin
        SYSCLK_P <= '1';
        SYSCLK_N <= '0';
        wait for CLK_PERIOD/2;
        SYSCLK_P <= '0';
        SYSCLK_N <= '1';
        wait for CLK_PERIOD/2;
end process;

    process
    begin
        RESET <= '1';
        wait for 1000 ns;
        RESET <= '0';
        wait for 1000 ns;
        WAIT;
    end process;

end Behavioral;
```

B. Vivado Block-Designs

B.0.1. SelectIO-Design

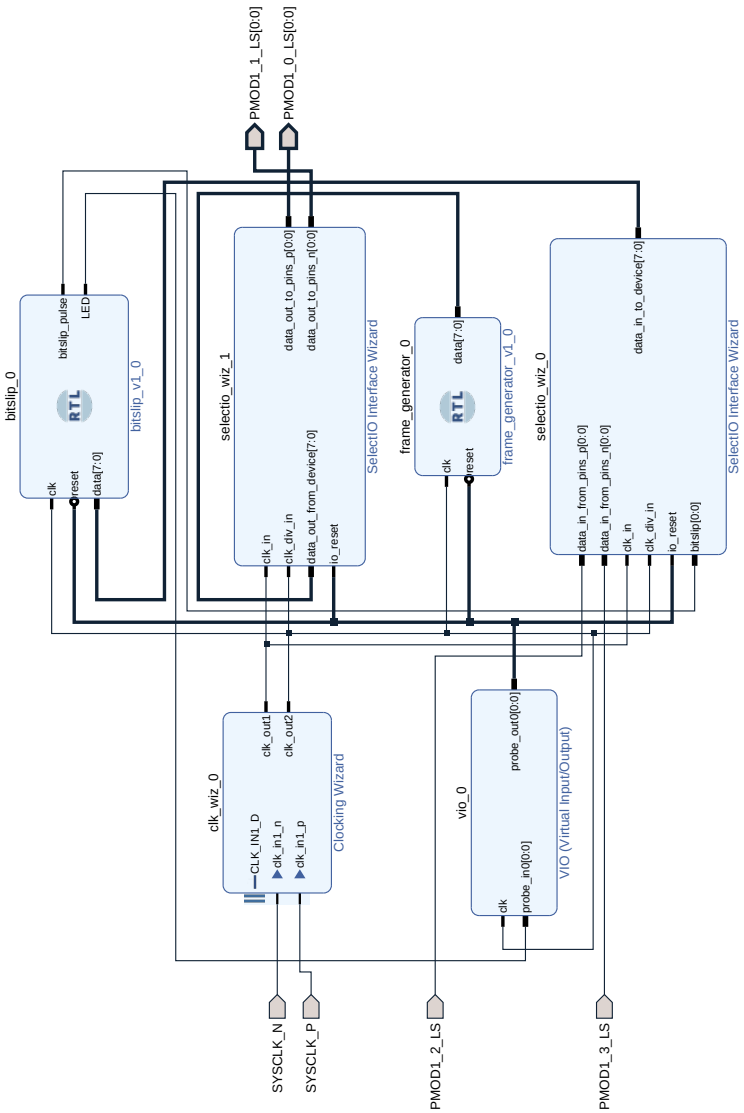

Abbildung B.1.: SelectIO-Design

B.0.2. Aurora 8B/10B-Design

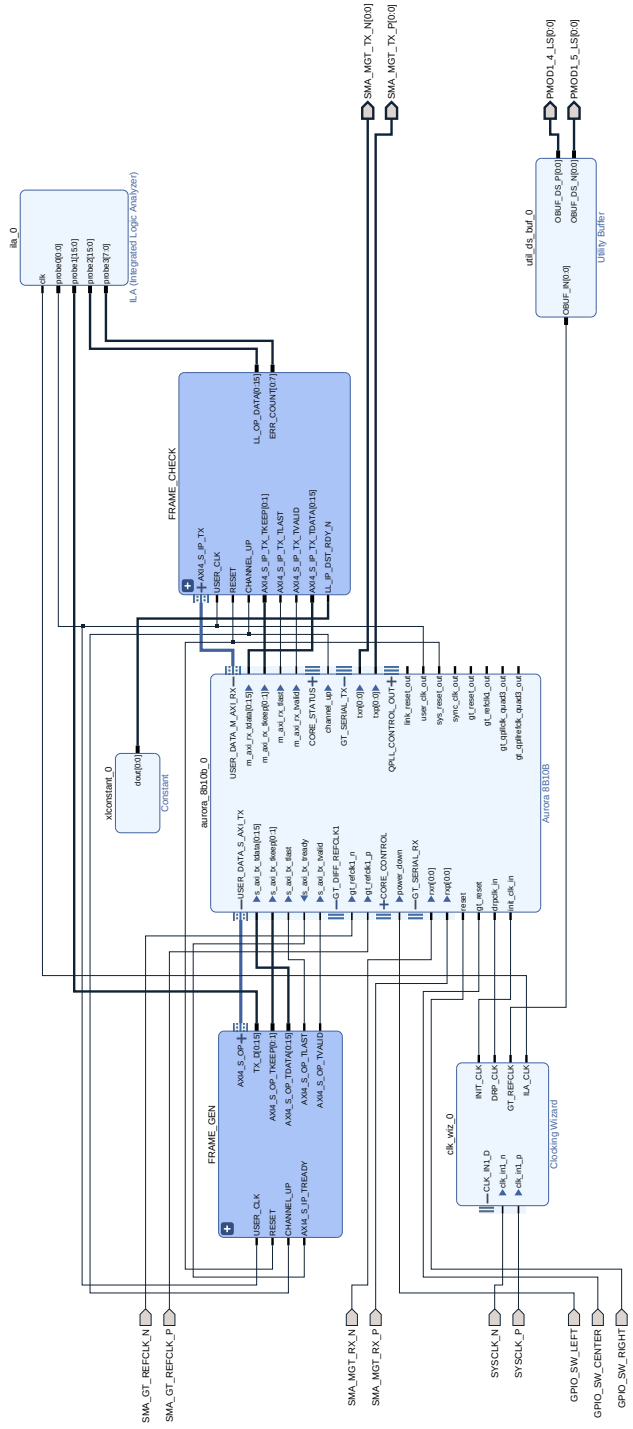

Abbildung B.2.: Aurora 8B/10B-Design

C. Signalverläufe

C.0.1. Simulation der Init-Sequenz mit Bitslip-Pulsen

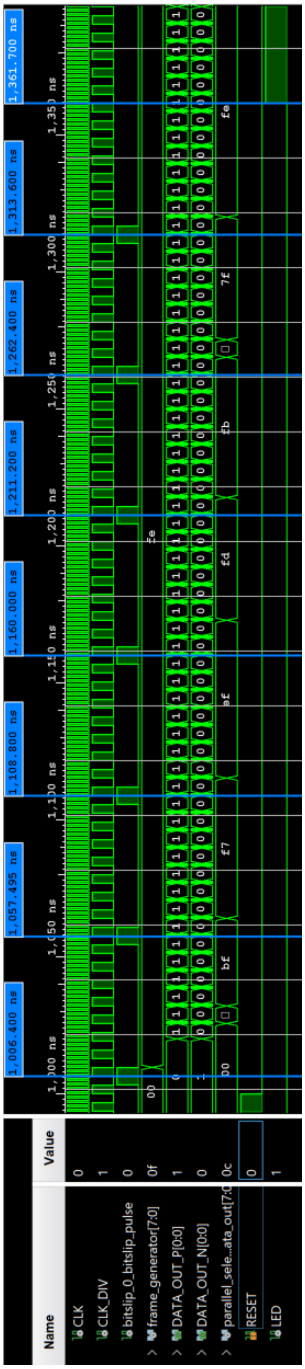

Abbildung C.1.: Simulation der Init-Sequenz mit Bitslip-Pulsen

Glossar

AXI **A**dvanced **e**Xtensible **I**nterface - Bus zum Datentransfer zwischen Programmable Logic und Proceessing System.

DDR **D**ouble **D**ata **R**ate - Datenübertragung erfolgt mit steigender und fallender Taktflanke.

DLR **D**eutsches Zentrum für **L**uft- und **R**aumfahrt.

DUT **D**evice **U**nder **T**est - Das System, das in einer Testumgebung auf Funktionalität untersucht wird.

FPGA **F**ield **P**rogrammable **G**ate **A**rray - Programmierbarer Logikbaustein zur Implementierung von Schaltungen mit Logikzellen.

GPIO **G**eneral **P**urpose **I**nput-**O**utput - Schnittstelle des FPGAs zum Datenaustausch mit externer Hardware.

ILA **I**ntegrated **L**ogic **A**nalyzer - Xilinx LogiCORE IP zum Aufzeichnen interner Signale des FPGAs über eine Debug-Schnittstelle.

LFSR **L**inear-**F**eedback **S**hift **R**egister - Ein Schieberegister, das über eine oder mehrere XOR-Verknüpfungen rückgekoppelt ist und Pseudo-Zufallswerte erzeugt.

LVDS **L**ow-**V**oltage **D**ifferential **S**ignaling - Übertragungsstandard zur High-Speed Datenübertragung.

SDR **S**ingle **D**ata **R**ate - Datenübertragung mit steigender Taktflanke.

SerDes **Ser**ialisierer/**Des**erialisierer - Bezeichnet seriell-zu-parallel-Konverter, bzw. parallel-zu-seriell-Konverter.

SoC **S**ystem **o**n **C**hip - Eine in Silizium integrierte Logik zur Lösung einer spezifischen Problemstellung.

Vivado Entwicklungsumgebung von Xilinx.

VHDL **V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage - Hardware-Beschreibungssprache zur Programmierung von FPGAs auf hohem Abstraktionsniveau.

VIO **V**irtual **I**nput/**O**utput - Xilinx LogiCORE IP zum Festlegen und Untersuchung interner Signale des FPGAs über eine Debug-Schnittstelle.

Bibliography

[1] Carl Christensen Abhijit Athavale. *High-Speed Serial I/OMade Simple*. 1st ed. Xilinx Connectivity Solutions, 2005. URL: https://www.xilinx.com/publications/archives/books/serialio.pdf (visited on 11/23/2021).

[2] DLR. *Das DLR*. URL: https://www.dlr.de/content/de/artikel/das-dlr.html (visited on 12/15/2021).

[3] DLR. *Institut für Robotik und Mechatronik*. URL: https://www.dlr.de/content/de/institutspraesentation/institut-fuer-robotik-und-mechatronik.html (visited on 12/15/2021).

[4] Meilhaus Electronic. *Wie funktioniert eigentlich differenzielle Signal-Übertragung?* URL: https://www.meilhaus.de/news-aktionen/blog/blog-differenziell/?b2b=0 (visited on 11/11/2021).

[5] Semiconductor Engineering. *LVDS (low-voltage differential signaling)*. URL: https://semiengineering.com/knowledge_centers/communications-io/off-chip-communications/i-o-enabling-technology/lvds-low-voltage-differential-signaling/ (visited on 12/14/2021).

[6] *Geschichte des DLR*. URL: https://web.archive.org/web/20150924015536/http://www.freunde-des-dlr.de/historie6.html (visited on 11/26/2021).

[7] Texas Instruments Inc. *TLK1221*. 2009. URL: https://www.ti.com/lit/ds/symlink/tlk1221.pdf (visited on 12/02/2021).

[8] Xilinx Inc. *7 Series FPGAs Data Sheet: Overview*. 2020. URL: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf (visited on 11/09/2021).

[9] Xilinx Inc. *7 Series FPGAs GTX/GTH Transceivers*. 2018. URL: https://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf (visited on 12/02/2021).

[10] Xilinx Inc. *7 Series FPGAs SelectIO Resources*. 2018. URL: https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf (visited on 11/25/2021).

[11] Xilinx Inc. *Aurora 8B/10B v11.1*. 2018. URL: https://www.xilinx.com/content/dam/xilinx/support/documentation/ip_documentation/aurora_8b10b/v11_1/pg046-aurora-8b10b.pdf (visited on 11/18/2021).

[12] Xilinx Inc. *Clocking Wizard v6.0*. 2021. URL: https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v6_0/pg065-clk-wiz.pdf (visited on 11/23/2021).

[13] Xilinx Inc. *Integrated Logic Analyzer v6.2*. 2016. URL: https://www.xilinx.com/support/documentation/ip_documentation/ila/v6_2/pg172-ila.pdf (visited on 12/01/2021).

[14] Xilinx Inc. *Intellectual Property (IP)*. URL: https://www.xilinx.com/products/intellectual-property.html (visited on 12/15/2021).

[15] Xilinx Inc. "LogiCORE IP Utility DifferentialSignaling Buffer (v1.01a)". In: *Product Specification* (2012). URL: https://www.xilinx.com/support/documentation/ip_documentation/util_ds_buf.pdf (visited on 12/08/2021).

[16] Xilinx Inc. *SelectIO Interface Wizard v5.1*. 2016. URL: https://www.xilinx.com/content/dam/xilinx/support/documentation/ip_documentation/selectio_wiz/v5_1/pg070-selectio-wiz.pdf (visited on 11/19/2021).

[17] Xilinx Inc. *ZC706 Built-In Self Test Flash Application - Präsentation*. 2015. URL: https://www.xilinx.com/member/forms/download/design-license.html?cid=413466&filename=xtp242-zc706-bist-c-2015-4.pdf (visited on 12/02/2021).

[18] Xilinx Inc. *ZC706 Evaluation Board for the Zynq-7000 XC7Z045 SoC*. 2019. URL: https://www.xilinx.com/content/dam/xilinx/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf (visited on 11/25/2021).

[19] Xilinx Inc. *Zynq-7000 SoC Data Sheet: Overview*. 2018. URL: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (visited on 11/09/2021).

[20] *Interne Dokumente des Instituts für Robotik und Mechatronik*.

[21] Inga Janović. *Coronavirus treibt Internetnutzung zu neuem Rekord*. 2020. URL: https://www.faz.net/aktuell/gesellschaft/gesundheit/coronavirus/coronavirus-treibt-internetnutzung-zu-neuem-rekord-16673785.html (visited on 12/15/2021).

[22] Bernd Schwarz Jürgen Reichardt. *VHDL-Simulation und -Synthese*. de Gruyter Oldenbourg, 2020. ISBN: 3110673452.

[23] Optech. *What is a Loopback Transceiver?* URL: https://sintrontech.com/definition-loopback-transceiver/ (visited on 11/18/2021).

[24] Manuel Padial Pérez. *Low Voltage Differential Signaling (LVDS) overview*. 2019. URL: https://www.doeeet.com/content/eee-components/actives/low-voltage-differential-signaling-lvds-overview/ (visited on 11/23/2021).

[25] Louise H. Crockett und Ross A. Elliot und Martin A. Enderwitz und Robert W. Stewart. *The Zynq Book - Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. 1st ed. Strathclyde Academic Media, 2014. URL: http://www.zynqbook.com/downloads/The_Zynq_Book_ebook_3.pdf (visited on 11/03/2021).

[26] *Stromrichtige und Spannungsrichtige Messung*. URL: https://www.leifiphysik.de/elektronik/halbleiterdiode/grundwissen/stromrichtige-und-spannungsrichtige-messung (visited on 12/13/2021).

[27] Roland Woitowitz. *Digitaltechnik : ein Lehr- und Übungsbuch*. Heidelberg New York: Springer, 2012. ISBN: 9783642208713.