FACHHOCHSCHULE
KOBLENZ

University of Applied Science

Fachbereich Elektro- und Informationstechnik

Diplomarbeit

Theoretische und praktische Analyse serieller Bussysteme bzgl. Störemission und Echtzeitverhalten mit anschließender Implementierung in einer mikrocontrollerbasierten Empfängersteuereinheit

Bearbeiter: Torsten Krause

Betreuer Hochschule: Prof. Dr. Uwe Gärtner

Bearbeitungsort: Max-Planck-Institut für Radioastronomie
Auf dem Hügel 69
53121 Bonn

Betreuer Institut: Dipl.-Ing. (FH) Thomas Berenz

Tag der Einreichung: 15.Oktober 2010

Danksagung

Zum Gelingen dieser Diplomarbeit und damit dem erfolgreichen Abschluss meines Studiums an der Fachhochschule Koblenz, haben einige Personen beigetragen, bei denen ich mich recht herzlich bedanken möchte.

- Zunächst danke ich Professor Dr. Uwe Gärtner, der mir während der Diplomarbeit immer mit Rat und Tat zur Seite stand.

- Zum Zweiten möchte ich mich bei Dipl.-Ing. (FH) Thomas Berenz für die sehr gute Zusammenarbeit bedanken. Er hat mir stets bei all meinen Problemen freundlich und kompetent weitergeholfen.

- Des Weiteren danke ich allen Mitarbeitern des Max - Planck – Instituts, die in jeglicher Form mit meiner Diplomarbeit zu tun hatten. Hier natürlich besonders den Kollegen aus dem Hochfrequenzlabor.

- Ein ganz besonderer Dank gilt meinen Eltern Paul und Petra Krause, meinem Bruder Daniel Krause und meiner Freundin Anna Hüllen, ohne deren Unterstützung dieses Studium nie möglich gewesen wäre.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die dem Prüfungsausschuss der Fachhochschule Koblenz eingereichte Diplomarbeit zum Thema „Theoretische und praktische Analyse serieller Bussysteme bzgl. Störemission und Echtzeitverhalten mit anschließender Implementierung in einer mikrocontrollerbasierten Empfängersteuereinheit“ vollkommen selbstständig verfasst und ausschließlich die angegebenen Quellen verwendet habe.
Alle Fotografien, Blockschaltbilder und Grafiken, wurden selbst erstellt. Es wurden keine fremden Abbildungen kopiert.

Koblenz, den 15. Oktober 2010

Torsten Krause

Inhaltsverzeichnis

1. Einleitung

1.1 Institut

1.1.1 Max – Planck – Gesellschaft

Im Jahre 1911 wurde die Kaiser – Wilhelm – Gesellschaft gegründet. Diese wurde am 26. Februar 1948 in Max – Planck – Gesellschaft zur Förderung der Wissenschaft e.V. (kurz: MPG) umbenannt.
Heute gelten die 80 Institute, Forschungsstellen, Laboratorien und Arbeitsgruppen als nationale bzw. internationale „Centres of Excellence“[1] der Grundlagenforschung. Es sind mehr als 13.300 Mitarbeiter, davon etwa 4.800 Wissenschaftler, bei der Max – Planck – Gesellschaft angestellt. Hinzu kommen noch ca. 7.000 Nachwuchs- und Gastwissenschaftler.

MAX-PLANCK-GESELLSCHAFT

Abbildung 1.1: *Logo Max – Planck – Gesellschaft*

Die einzelnen Institute forschen in den Natur-, Bio-, Geistes- und Sozialwissenschaften im Dienste der Allgemeinheit. Sie arbeiten mit anderen Forschungsinstituten zusammen und sind auf Grund ihrer interdisziplinären Arbeit mit Hochschulen vernetzt. Die Max – Planck – Gesellschaft unterstützt besonders Forschungsrichtungen, die von deutschen Universitäten auf Grund ihres neuen und innovativen Charakters wenig berücksichtigt sind. Aufwändige Geräte und Einrichtungen, wie zum Beispiel wissenschaftliche Bibliotheken oder dem Radioteleskop in Effelsberg, werden einem großen Kreis von Wissenschaftlern zur Verfügung gestellt.
Ein spezielles Programm für junge, aufstrebende Wissenschaftler ermöglicht diesen, zeitlich begrenzt, eine Max – Planck – Forschungsgruppe an einem der Institute aufzubauen. Diese sind frei in der Gestaltung ihrer wissenschaftlichen Forschung.

1: Centres of Excellence: Experten Zentrum, Spitzenforschungszentrum

1.1.2 Max – Planck – Institut für Radioastronomie

Die Arbeit des 1966 gegründeten Max – Planck – Instituts für Radioastronomie (kurz: MPIfR) beruht wesentlich auf Beobachtungen, die am Radio-Observatorium Effelsberg durchgeführt werden. Neben der Radioastronomie sind auch Infrarotastronomie und theoretische Astrophysik wichtige Hauptaufgabengebiete.
Da die vom Radioteleskop Effelsberg empfangenen Signale sehr schwach sind und das Umgebungsrauschen um ein vielfaches stärker ist, entwickelt die Elektronikabteilung des Max – Planck – Instituts für Radioastronomie extrem rauscharme, cryogenische Empfänger .

Abbildung 1.2: *Logo und Gebäude Max-Planck-Institut für Radioastronomie*

Regelmäßig beteiligt sich das Institut an verschiedenen internationalen Netzwerken, bei denen mit Hilfe vieler Teleskope ein virtuelles Radioteleskop mit einem Durchmesser von mehreren hundert Kilometern simuliert wird. Diese Radiointerferometriebeobachtungen erreichen die höchsten räumlichen Auflösungen, die zurzeit möglich sind. Die Auswertung der gewonnenen Daten wird, in Zusammenarbeit mit dem Bundesamt für Kartographie und Geodäsie, mit einem speziellen Korrelationsrechner realisiert.
Die wissenschaftliche Arbeit des Instituts ist in vier Forschungsgruppen gegliedert:

- Radioastronomische Fundamentalphysik
- Millimeter- und Submillimeter- Astronomie
- Infrarot - Astronomie
- Radioastronomie / Very Long Baseline Interferometry

1.1.3 Radioteleskop Effelsberg

Das 100 m - Radioteleskop Effelsberg (Abb. 1.3) ist das zweitgrößte vollbewegliche Teleskop der Welt. Es ist seit 1972 in Betrieb und dient seitdem für radioastronomische Beobachtungen. Kontinuierlich wird an der Verbesserung der genutzten Technik gearbeitet.

Abbildung 1.3: *Radioteleskop Effelsberg*

2352 Paneelen bilden die 7850 m^2 große Antennenoberfläche. Das Teleskop hat ein Gesamtgewicht von 3200 t und kann in 12 Minuten mit einer Genauigkeit von 0,3 mm eine 360° Drehung vollziehen.
Radiowellen mit Wellenlängen zwischen 3,5 mm und 90 cm können empfangen und ausgewertet werden. Der Standort wurde in einem Tal mit geringer Bewohnung gewählt um den Einfluss von Fremdstrahlung zu minimieren. Des Weiteren hat das Tal eine Nord-Süd-Ausrichtung. Dies ist von großem Vorteil, da so eine Beobachtung des Zentrums der Milchstraße möglich ist, die in südlicher Richtung liegt.
Hauptbeobachtungsziele sind:

- Pulsare
- Schwarze Löcher
- Galaxiekerne
- Kalte Gas- und Staubwolken

1.2 Aufgabenstellung

Im Radioteleskop Effelsberg werden die Signale zur Steuerung der astronomischen Empfangseinheiten wegen geringer Störemission parallel übertragen. Da zukünftige Empfänger mit deutlich mehr Kanälen gebaut werden, soll diese Kommunikation serialisiert werden. Neben der Störstrahlungsemission steht die Datenkapazität des Bussystems im Vordergrund.
Das zurzeit im Radioteleskop verwendete Kommunkationssystem nennt sich DÜSY. Es handelt sich um ein betriebseigenes serielles Bussystem. Da DÜSY durch ein anderes Bussystem ersetzt werden soll, muss dieses messtechnisch erfasst werden, um die Mindestanforderungen zu kennen.
Auf Grundlage der Mindestanforderungen werden im Anschluss kommerziell erhältliche serielle Bussysteme theoretisch auf Nutzbarkeit als Ersatzsystem untersucht.
Da keines der Bussysteme diesen Anforderungen entspricht, wird anschließend ein Eigenbau verwirklicht. Dies beinhaltet die Festlegung des Protokolls, das Suchen nach geeigneten Bausteinen, den Entwurf der notwendigen Platinen. Die Platinen müssen bestückt werden und anschließend getestet.
Zur Erzeugung der Daten wird das „Spartan 3E FPGA Evaluationsboard“ programmiert, das außerdem die gesendeten Daten mit denen nach der Übertragung vergleicht, um die Bitfehlerrate des Systems zu bestimmen.
Des Weiteren wird zur Minimierung der Störstrahlung die Steuerplatine mit dem Mikrocontroller in ein selbstkonstruiertes Metallgehäuse gebettet. Zum Vergleich wird die Störemission der Platine mit und ohne Gehäuse gemessen und gegenübergestellt.

2 Technische Grundlagen

2.1 OSI - Referenzmodell

Das Open - Systems - Interconnect - Referenzmodell (kurz: OSI - Referenzmodell) dient der Beschreibung von Netzwerken (vgl. [CIE], [KLE] und [TAN]). Es wurde im Jahre 1983 von der ISO[1] veröffentlicht und seither mehrfach überarbeitet.
Notwendig wurde das OSI – Modell, da die Kommunikation zwischen verschiedenartigen Partnern mit der Zeit immer komplexer wurde und es keine internationalen Standards gab. Dies führte zu großen Problemen bei der Vernetzung unterschiedlicher Systeme.
Beschrieben wird mit Hilfe des OSI - Modells die Kommunikation von offenen Systemen. Diese heißen so, da sie für die Kommunikation mit anderen Systemen, beispielsweise anderer Hersteller, bereit sind.
Eingeteilt wird die Kommunikation in sieben abstrakte Ebenen oder Schichten. Jede einzelne Schicht hat dabei ein klar definiertes Aufgabengebiet und unterstützt unterschiedliche Funktionalitäten der Kommunikation. Jede Ebene ist jeweils eine funktionale Erweiterung der Darunterliegenden.

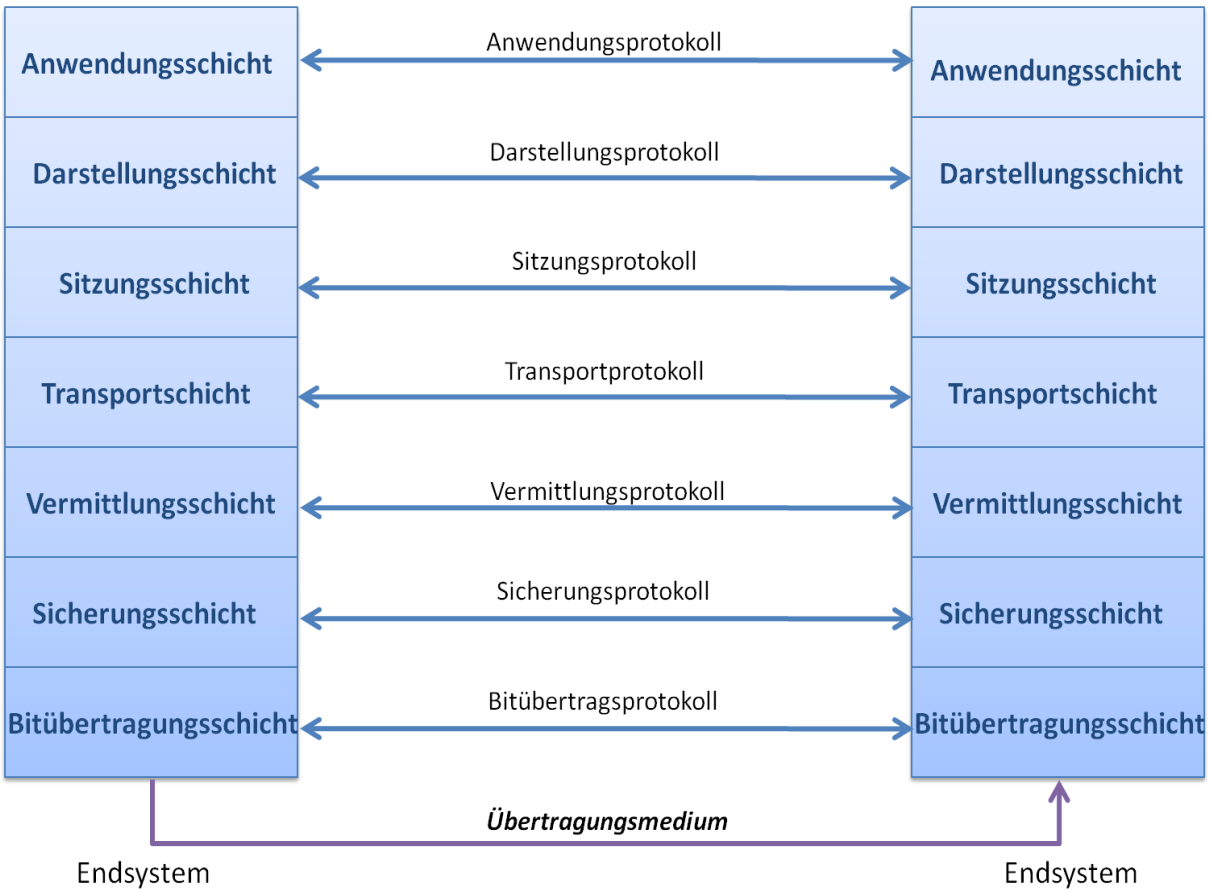


Abbildung 2.1: *OSI - Referenzmodell*

Über die **Bitübertragungsschicht** (Physical Layer) werden alle mechanischen, elektrischen und funktionalen Parameter übermittelt, die zur Übertragung eines Bitstroms notwendig sind. Dies sind zum Beispiel Informationen welcher Spannung welchem logischen Pegel entspricht, wie eine Erstverbindung zustande kommt oder wie lange ein Bit dauert.
Die **Sicherungsschicht** (Data Link Layer) gewährleistet, dass die Kommunikation zweier benachbarter Systeme fehlerfrei funktioniert. Die Daten werden in sogenannte Rahmen (Frames) eingepackt, so dass Übertragungsfehler von den teilnehmenden Stationen erkannt werden oder auch behoben werden können. Auch die Datenüberschwemmung, ein Problem das auftritt, wenn der Sender schneller Daten auf das Übertragungsmedium schickt, als dass der Empfänger sie lesen kann, wird in dieser Schicht behandelt.
Für den geregelten Sendebetrieb in einem Netzwerk mit mehr als zwei Knoten ist die **Vermittlungsschicht** (Network Layer) verantwortlich. Die Auswahl der Paketrouten vom Ursprungsort zum Bestimmungsort muss festgelegt werden. Dies kann über statische Tabellen geschehen oder aber durch eine Terminalsitzung, die auf aktuelle Gegebenheiten wie derzeitige Netzauslastung Rücksicht nimmt. Diese Sitzung kann für jedes gesendete Datenpacket einzeln vorgenommen werden, um eine optimale Netzauslastung zu gewährleisten. Eine weitere wichtige Aufgabe ist das Vermeiden von Datenstaus durch eine Begrenzung der Anzahl von Daten, die sich in einem Netzwerk befinden.
Die vierte Schicht des OSI – Referenzmodells stellt sicher, dass Nachrichtenpakete in der richtigen Reihenfolge beim Empfänger ankommen, dass keine doppelt oder gar nicht gesendet wurden. Des Weiteren gewährleistet die **Transportschicht** (Transport Layer) auch den sogenannten Quality of Service. Dieser gibt die Dienstgüte an, die durch bestimmte Parameter beschrieben wird. So wird beispielsweise der Datenverkehr so beeinflusst, dass Verkehrsstaus vermieden werden und abgestufte Bandbreiten garantiert werden.
Die **Sitzungsschicht** (Session Layer), häufig auch als Kommunikationssteuerungsschicht bezeichnet, ermöglicht das Einrichten von Sitzungen an verschiedenen Rechner. Hier ist es wichtig, dass die Kommunikation nacheinander erfolgt. Auch der Auf- und Abbau der Sitzungen wird überwacht und die Synchronisation der Datenströme vorgenommen.
Die bisher genannten Schichten des OSI-Referenzmodells stellen eine korrekte Übertragung der Bits sicher. Die **Darstellungsschicht** (Presentation Layer) hingegen ist für die Semantik und Syntax zuständig. Dies bedeutet, dass die Darstellung bestimmter Zeichen bei unterschiedlichen Kommunikationspartnern varrieren kann. Funktionen, die hier festgelegt werden, sind verwendeter Zeichensatz, Informationen zur Codierung der Daten und Darstellungsformat für Drucker oder Bildschirm. Zusätzlich wird auch eine Vereinbarung bezüglich einer eventuellen Komprimierung der Daten zur Verringerung der Datenmenge getroffen.
Die oberste Schicht des OSI-Referenzmodells ist die **Anwendungsschicht** (Application Layer). Diese beinhaltet alle von den unteren Schichten definierten Protokolle und Dienste. Es werden alle Funktionen bereitgestellt, mit denen der Benutzer auf das Kommunikationssystem zugreifen kann.

2.2 Netzwerkstrukturen

Sobald mehrere Rechner bzw. Kommunikationseinheiten miteinander verbunden sind, um Informationen auszutauschen, wird dies nach Schnell [SWI] als Netzwerk bezeichnet.
Unter Netzwerkstruktur, oft auch Netztwerktopologie genannt, wird die Gliederung der einzelnen Einheit untereinander verstanden.
Die Wahl der Netzwerktopologie beeinflusst entscheidend die Art der Kommunikation und verschiedene Parameter wie Ausfallsicherheit und maximale Anzahl an übertragbaren Nachrichten pro Zeiteinheit.
Oft ist ein Informationsaustausch zwischen zwei Stationen möglich, ohne dass diese physikalisch über eine Leitung direkt verbunden sind.

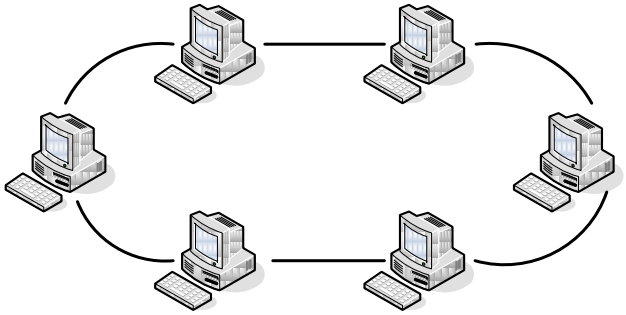


Abbildung 2.2: *Ringtopologie*

Bei der Ringtopologie (Abbildung 2.2) bilden alle Knoten mittels Zweipunktverbindungen einen geschlossenen Ring. Nachrichtenpakete werden von Teilnehmer zu Teilnehmer weitergeleitet. Jeder Einzelne prüft einen Adressteil der Nachricht und entscheidet, ob die Nachricht weitergeleitet werden muss oder für diesen Knoten bestimmt ist.
Das verwendete Protokoll muss sicherstellen, dass es zu keinen Kollisionen der Nachrichten kommt, da es sich ausschließlich um aktive Knoten in einer Ringtopologie handelt. Dies bedeutet, dass die Nachricht von jeder beteiligten Einheit neu zum nächsten benachbarten Knoten versendet wird, daher sind sehr große Entfernungen zwischen den Kommunikationsteilnehmer möglich.
Da es sich jeweils um Punkt – zu – Punkt – Verbindungen handelt, ist eine Übertragung über Lichtwellenleiter leicht realisierbar. Auch bei der Kommunikation über Kupferleitungen ist nur eine Leitung notwendig.
Probleme entstehen, wenn es zu einem Kurzschluss oder Kabelbruch kommt. Auch der Ausfall eines Teilnehmers führt zu Komplikationen. Um diese zu beheben, kann der Ring redundant konzipiert werden. Dabei ist eine Kommunikation in beide Richtungen möglich.

Eine weitere Netzwerkstruktur ist die Sterntopologie (Abbildung 2.3). Diese ist durch eine zentrale Einheit charakterisiert, an die alle Anderen sternförmig angeschlossen sind.
Es ist möglich als zentralen Teilnehmer einen Hub einzusetzten, der die Nachricht analysiert und an die richtige Schnittstelle weiterleitet, oder einen Rechner mit höherer Intelligenz zu verwenden, so dass beispielsweise eine Priorisierung einzelner Nachrichten möglich ist.

So kommt es zu einem Netzwerk, das nur aus Punkt – zu – Punkt – Verbindungen besteht. Fällt einer dieser Leitung oder eine Einheit aus, so ist eine Kommunikation zwischen allen übriggebliebenen Einheiten möglich.

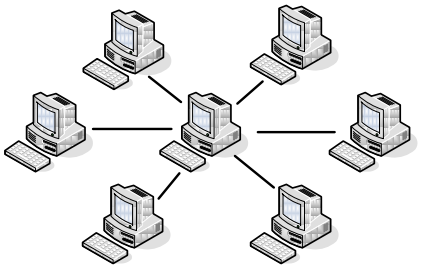


Abbildung 2.3: *Sterntopologie*

Dahingegen hat ein Ausfall der zentralen Einheit einen Komplettausfall des Netzwerkes zur Folge. Auch im normalen Betrieb ohne jegliche Ausfälle ist die zentrale Einheit eine Engstelle. Jegliche Nachricht muss diese Einheit passieren. Besteht ein Netzwerk aus N Teilnehmern, so benötigt die zentrale Kommunikationseinheit N-1 Schnittstellen. Der Zugriff auf das Netzwerk kann auf zwei verschiedene Arten realisiert werden. Beim Polling fragt ständig die zentrale Einheit bei den anderen Einheiten an, ob der Bedarf besteht Daten zu senden. Da dies jedoch ein langsames Verfahren ist, ist es auch möglich die Kommunikation so aufzubauen, dass jede sendewillige Einheit eine Anfrage an den Zentralrechner stellt, ob schon jemand sendet und dann eine Sendefreigabe erhält, wenn noch keiner sendet. Das Einbinden neuer Teilnehmer oder die Kommunikation über Lichtwellenleiter ist sehr einfach. Der Kabelaufwand ist im Vergleich zu anderen Netzwerktopologien erhöht.
Die in Abbildung 2.4 dargestellte Topologie nennt sich Bustopologie und wird auch Linienstruktur genannt. Hierbei gibt es ein gemeinsames Übertragungsmedium und alle Teilnehmer sind über eine im Allgemeinen kurze Stichleitung passiv angekoppelt.
Jeder Teilnehmer ist gleichberechtigt. Wie bei der Ringtopologie handelt es sich hierbei um ein Diffusionsnetz, bei dem sich eine Nachricht im gesamten Netz ausbreitet und jede Einheit entscheidet, ob die Nachricht für sie bestimmt ist.

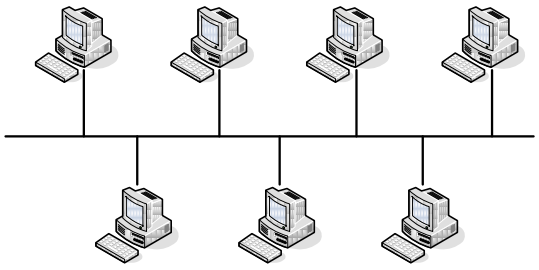


Abbildung 2.4: *Bustopologie*

Die Verkabelung und das Hinzufügen neuer Teilnehmer ist sehr einfach. Allerdings ist die Buslänge und maximale Anzahl an Kommunikationsteilnehmern stark begrenzt. Auch die Übertragung über Lichtwellenleiter ist kompliziert, da die Signale aus dem gemeinsamen Übertragungsmedium ausgekoppelt werden müssen.
Die Übertragunsleitung muss an beiden Enden abgeschlossen werden, damit es zu keinen Reflexionen kommt. Um Kollisionen zu vermeiden, wird der Zugriff auf das

Übertragungsmedium geregelt. Hierbei „horcht“ jeder Teilnehmer ständig, ob die Leitung belegt ist.
Kommt es zu Ausfällen von Teilnehmer, so fällt nur die an dem Teilnehmer beteiligte Kommunikation aus. Fällt allerdings eine Leitung aus, kommt es zu Ausfällen von Netzwerkteilen.

2.3 Übertragungsmedien

Ein Kanal zur Übertragung von Nachrichten verbindet allgemein zwei Kommunikationspartner. Dieser kann aus unterschiedlichen Übertragungsmedien realisiert werden. Abbildung 2.5 zeigt eine Klassifizierung der möglichen Übertragungsmedien.

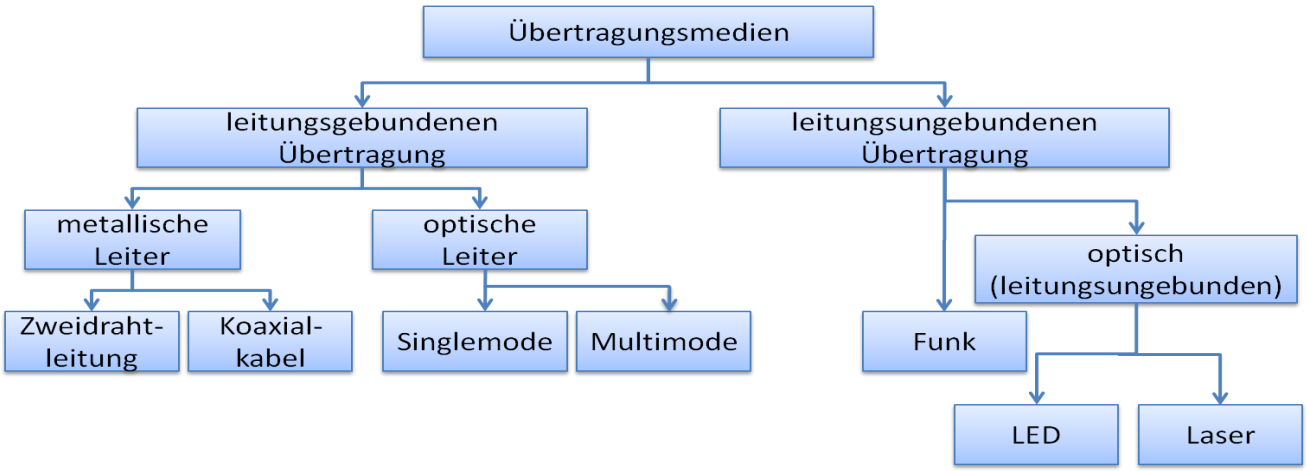


Abbildung 2.5: *Klassifikation Übertragungsmedien*

Die unterschiedlichen Kabeltypen besitzen verschiedene Eigenschaften, die durch die folgenden Kenngrößen beschrieben werden:

- **Störungen**: diese sind meistens durch eingekoppelte Fremdsignale oder Eigenrauschen des Leiters hervorgerufen.
- **Dämpfung:** hat eine Verringerung der Signalamplitude zur Folge. Die Dämpfung ist frequenzabhängig.
- **Laufzeit:** Zeit, die ein Signal zur vollständigen Übertragung benötigt. Diese ist wie die Dämpfung auch frequenzabhängig, aber auch vom Aufbau des Mediums.
- **Bandbreite:** nur in bestimmten Frequenzbereichen ist eine Übertragung möglich, Frequenzanteile außerhalb dieses Frequenzbandes werden von den verwendeten Materialien gedämpft. Den Bereich zwischen maximal und minimal übertragbarer Frequenz, wird als Bandbreite bezeichnet.

Die Art der Übertragung elektrischer Signale wird in zwei große Gruppen aufgeteilt. Zum Einen in die leitungsgebundene Übertragung, bei der ein physikalisches Medium zur Übertragung genutzt wird. Zum Anderen in leitungsungebundene Übertragung bei der die Luft bzw. der Freiraum als Trägermaterial genutzt wird.

Bei der leitungsungebunden Übertragung, auch drahtlose Übertragung (wireless) oder Freiraumübertragung genannt, wird die zu übertragende Information über sich in der Luft ausbreitende elektromagnetische Wellen übermittelt.
Die leitungsgebundenen Medien lassen sich noch weiter in metallische und optische Wellenleiter unterteilen.
Bei den Leitern aus Metall, gibt es eine Vielzahl von verschiedenen Varianten, die Gängigsten sind die in Abbildung 2.6 dargestellte Zweidrahtleitung oder Twisted-Pair-Leitung und die Koaxialleitung.

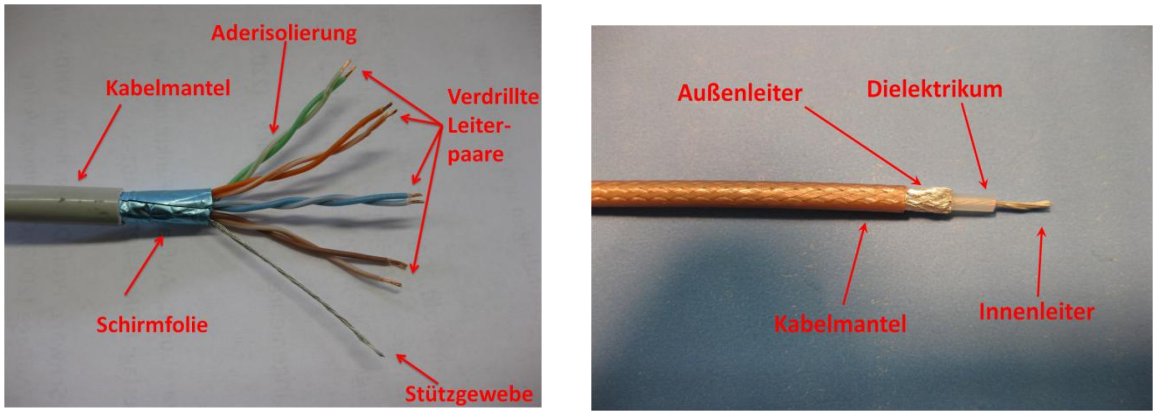


Abbildung 2.6: *links: Twisted-Pair-Kabel rechts: Koaxialkabel*

Twisted-Pair-Kabel:
Eine Twisted-Pair-Leitung besteht aus mindestens zwei Adern, die miteinander verdrillt sind. Jede Ader ist einzeln isoliert. Die spiralförmige Verdrillung der Leitungen bewirkt, dass störende Einflüsse von Außen, zum Beispiel eines anderen stromdurchflossenen Leiters, weniger Auswirkungen auf das Signal haben. Die Schirmfolie reduziert den Einfluss niederfrequenter Störungen. Das Stützgewebe zwischen den verdrillten Leitungen und der Folie dient ausschließlich zum Füllen der Zwischenräume und dem Verhindern statischer Ladungen bei Reibung.
Das Ganze wird vom Kabelmantel umschlossen, dieser schützt das Kabel vor mechanischen Einflüßen und begrenzt auch den Biegeradius, damit die Leitungen nicht beschätigt werden.

Koaxialleitung:
Die Koaxialleitung besteht, ähnlich dem Twisted-Pair-Kabel, aus zwei Leitern. Allerdings ist der Aufbau unterschiedlich. Die Koaxialleitung besteht aus einem Innenleiter, auch Kern genannt. Dieser wird von einem Dielektrikum umschlossen. Dieser Isolator wird wiederum von einem zylindrischen Leiter umgeben, oft in Form eines eng geflochtenen Netzes.
Der Kabelmantel, als äußerste Hülle, schützt das gesamte Kabel wie bei der Twisted-Pair-Leitung.

Lichtwellenleitung
Die Übertragung von Lichtwellen über optische Medien, auch Lichtwellenleiter, Glasfaserkabel oder optische Faser genannt, hat vier Vorteile im Vergleich zu metallischen Leitern:

- Optische Übertragungsmedien haben eine bedeutend kleinere Dämpfung. Dies bedeutet, dass ohne Signalverstärker viel größere Entfernungen überbrückt werden können.

- Durch die höhere Frequenz des Lichtes steht eine größere Bandbreite zur Verfügung.
- Elektromagnetische Einflüsse beeinträchtigen die Signalqualität bei Lichtstrahlen weniger.
- Die optischen Kabel strahlen aber auch selbst weniger Emissionen aus, so dass andere elektrischen Komponenten in der Umgebung nicht gestört werden.

Glasfaserkabel bestehen aus einem Kern und einem Mantel. Der Mantel besteht aus zwei Komponenten, dem Cladding und dem Coating.
Der optische Mantel umschließt den Kern und durch Totalreflexion an der Grenzfläche der beiden Komponenten wird der Lichtstrahl im Kern geführt.
Damit es zur Toatalreflexion kommt, muss nach Lindner [LIN] unter Verwendung des Snelliusschen Brechungsgesetzes der Winkel mit dem das Licht auf die Grenzfläche zwischen Kern und Mantel trifft, mindestens minimal größer sein, wie der „Grenzwinkel der Totalreflexion“ β_g oder es müssen die Materialien für Mantel und Kern mit passenden Brechungsindices $n_1 \; und \; n_2$ gewählt werden.

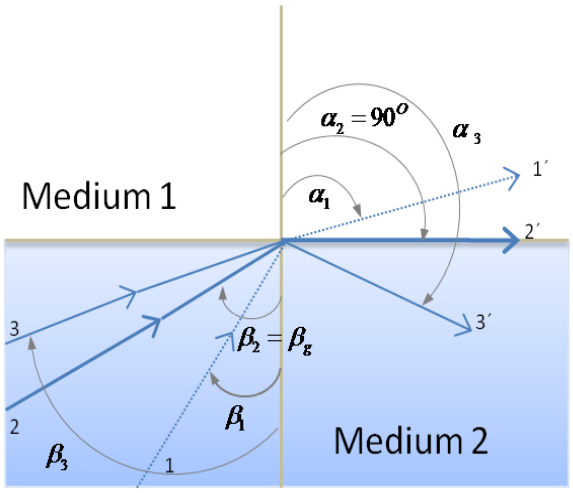


Abbildung 2.7: *Entstehung der Totalreflexion*

Snelliussches Brechungsgesetz: $\frac{\sin(\alpha)}{\sin(\beta)} = \frac{n_1}{n_2}$

Berechnung Grenzwinkel:

$$\beta_g = \arcsin\left(\frac{n_1}{n_2}\right) = \arcsin\left(\frac{n_{MANTEL}}{n_{KERN}}\right)$$

In Abbildung 2.7 ist die Entstehung der Totalreflexion verdeutlicht. Während es bei Lichtstrahl 1 zu einer Brechung kommt, wird Lichtstrahl 3 totalreflektiert. Der Strahl, der im Winkel $\beta_2 = \beta_g$ auf die Grenzfläche trifft, wird so reflektiert, dass er im 90° Winkel zum Lot, also genau an der Grenzfläche entlang verläuft. Dieser Winkel wird „Grenzwinkel der Totalreflexion“ genannt.

Die Veränderung der Brechungsindices kann bei Glas durch Dotierung mit Fremdatomen erreicht werden.
Lichtwellenleiter lassen sich nach dem Weg, den der Lichtstrahl im Kern zurückgelegt hat, klassifizieren. Diese unterschiedlichen Wege werden als Moden bezeichnet. Wie in Abbildung 2.8 verdeutlicht ist, gibt es Multimodefasern und Singlemodefasern.
Bei den Multimodefasern ist der Radius des Kerns so gewählt, dass Lichtstrahlen die in bestimmten Winkeln einfallen, durch mehrmalige Totalreflexion durch den Lichtwellenleiter geführt werden. Da bei derartigen Fasern mehrere Lichteinfallswinkel zulässig sind, nehmen die Lichtstrahlen unterschiedliche Moden und haben somit auch verschiedene Laufzeiten durch den Leiter. Durch diese Eigenschaft der Multimodefasern, der Modendispersion, kommt es selbst bei niedrigen Datenraten zu Signalverzerrungen.

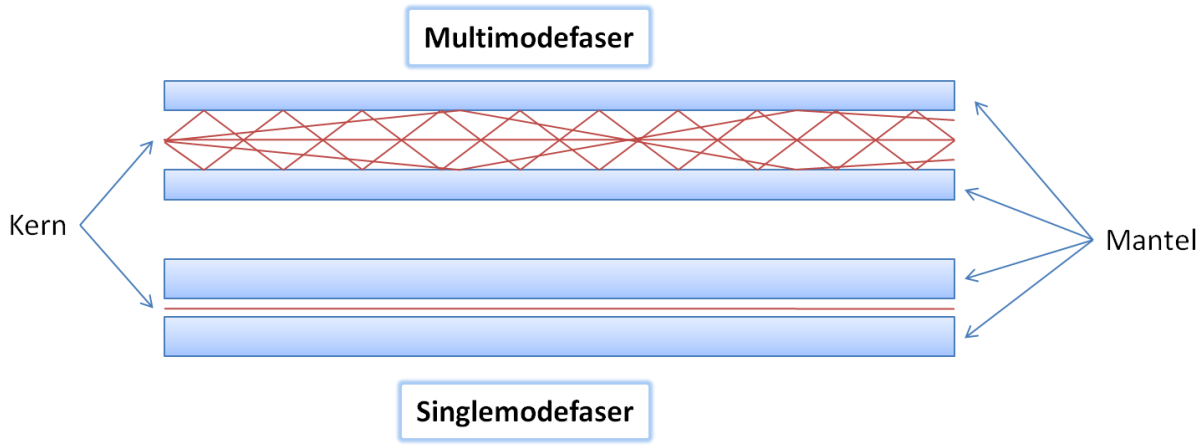


Abbildung 2.8: *Strahlengang Lichtwellenleiter*

Ist der Durchmesser des Lichtwellenleiters so gering gewählt, dass nur noch eine Mode möglich ist, so handelt es sich um eine Singlemodefaser. Bei diesen Fasern werden Verzerrungen durch Modendispersion vermieden. Auf Grund des geringen Kerndurchmessers ist jedoch ein hochpräziser Aufbau nötig.

2.4 Symmetrische Datenübertragung

Um bei der Signalübertragung den Einfluss von äußeren Störungen zu minimieren, werden häufig Signale symmetrisch versendet. Dabei wird die gleiche Information über zwei getrennte Leitungen gleichzeitig übertragen. Allerdings werden die Pegel gegeneinander vertauscht, häufig einfach negiert. Entspricht der Pegel U_0 auf der Leitung A einem logischen Low-pegel und U_1 einem Highpegel, so ist es auf der Leitung B genau umgekehrt.
Am Ende der Leitung ist ein Operationsverstärker angebracht, der Signal- von Signal+ subtrahiert.

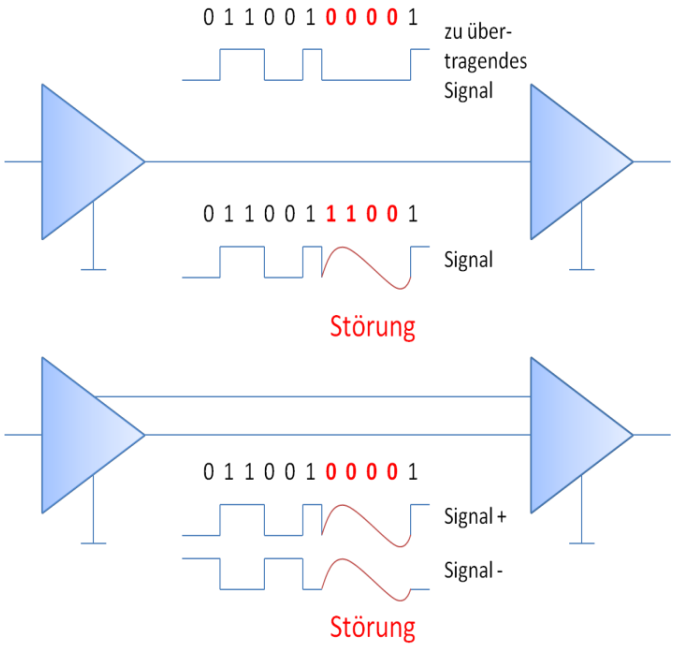


Abbildung 2.9: *Symmetrische Datenübertragung*

Soll logisch „1“ übertragen werden, so liegt auf Leitung A der Pegel U_1 und auf Leitung B der Pegel U_0 an. Nach dem Operationsverstärker ist der Pegel $U_1 - U_0$ zu messen. Folglich wäre dieser beim Übermitteln einer logischen „0“ $U_0 - U_1$. Kommt es zu Störungen, beispielsweise durch das Magnetfeld eines anderen stromdurchflossenen Leiters, so wirkt dieser Störpegel $U_{STÖR}$ auf beide Signalleitungen gleichermaßen. Diese Störung hat allerdings auf das Signal hinter dem Operationsverstärker, wie folgende Rechnung zeigt $(U_1 + U_{STÖR}) - (U_0 + U_{STÖR}) = U_1 - U_0$, keinen Einfluss.

3 Serielle Bussysteme

3.1 Voruntersuchungen

3.1.1 Übertragungssystem DÜSY

Im Radioteleskop Effelsberg fallen eine Vielzahl von Daten und Signalen an, die übertragen werden. Ein Teil dieses komplexen Kommunikationsgeflechts ist das DÜSY.
DÜSY steht für DatenÜbertragungsSYstem und ist ein vom Max-Planck-Institut selbst entwickeltes System, das die Kommunikation zwischen „Frontend Drive Control“ und „Frontend Drive Interface“ regelt.

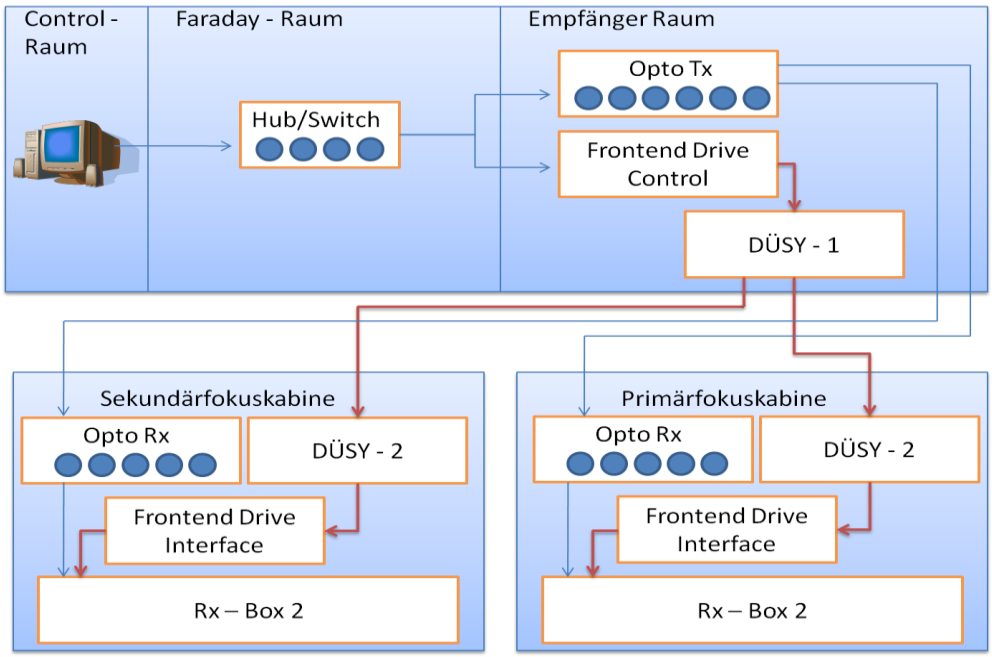


Abbildung 3.1: *Kommunikationsschema Steuersignale*

Bei radioastronomischen Beobachtungen können immer nur Signale mit einer bestimmten Wellenlänge empfangen werden. Jede mögliche Wellenlänge hat eine eigene Empfangseinheit. Wobei 14 Empfänger in der Primärfokuskabine und 7 in der Sekundärfokuskabine stehen. Um die Empfänger zu steuern, wird jeder einzeln mit 24 Schaltsignalen angesteuert.
Diese Daten werden, wie in Abbildung 3.1 veranschaulicht, im Controlraum erzeugt. Von da kommen sie in den Faradayraum, wo sie von anderen Daten selektiert und in die „Frontend Drive Control“ im Empfängerraum weitergeleitet werden. Hier werden diese Daten dann auf das DÜSY gebracht. DÜSY überträgt die Signale seriell in die beiden Empfängerkabinen, wo sie in der „Frontend Drive Interface“ auf die einzelnen Empfänger verteilt werden. Das „Frontend Drive Interface“ arbeitet ähnlich wie ein Hub.

Das Übertragungssystem DÜSY arbeitet paketorientiert. Es werden immer 8 Bit zu einem Unterrahmen zusammengefasst. 8 Unterrahmen bilden einen Unterrahmenblock. Ein Hauptrahmen besteht aus insgesamt 481 Bit; 256 Datenbit und 225 Kontrollbit.
Die Übertragungsrate liegt bei 8,5 MHz, was einer Bit-Länge von 117,6 ns entspricht.
Damit dauert ein Hauptrahmen 56,57 µs. Die Übertragungszeit über das Kupferkabel ist bei einer Kabellänge von rund 400 m mit unter 2 µs vernachlässigbar.

3.1.2 Vermessung des bestehenden Übertragungssystems

Um die Anforderungen an ein neues Übertragungssystem festlegen zu können, müssen erst einige Parameter des alten Systems DÜSY bestimmt werden. Die meisten Parameter gingen aus der technischen Dokumentation hervor (*siehe Kapitel 3.1.1*).
Der Jitter und die Laufzeit einer Nachricht wurden messtechnisch erfasst. Hierzu wurde in der Primärfokuskabine ein Ausgangspin und ein Eingangspin gebrückt. Ein gesendeter Impuls durchlief die Kommunikationsstrecke Empfängerraum → Fokuskabine → Empfängerraum.
In Abbildung 3.2 ist das Ergebnis der Messung der Signallaufzeit dargestellt. Das gelbe Eingangssignal hat zu einem bestimmten Zeitpunkt eine fallende Flanke. Das rosa dargestellte Ausgangssignal 98 µs später die fallende Flanke. Dies bedeutet, dass das Signal genau diese Zeit gebraucht hat um die Strecke zu durchlaufen. Somit ist, mit einer gewissen vernachlässigbaren Ungenauigkeit, die Signallaufzeit bei einfacher Übertragung Empfängerraum → Fokuskabine 49 µs.

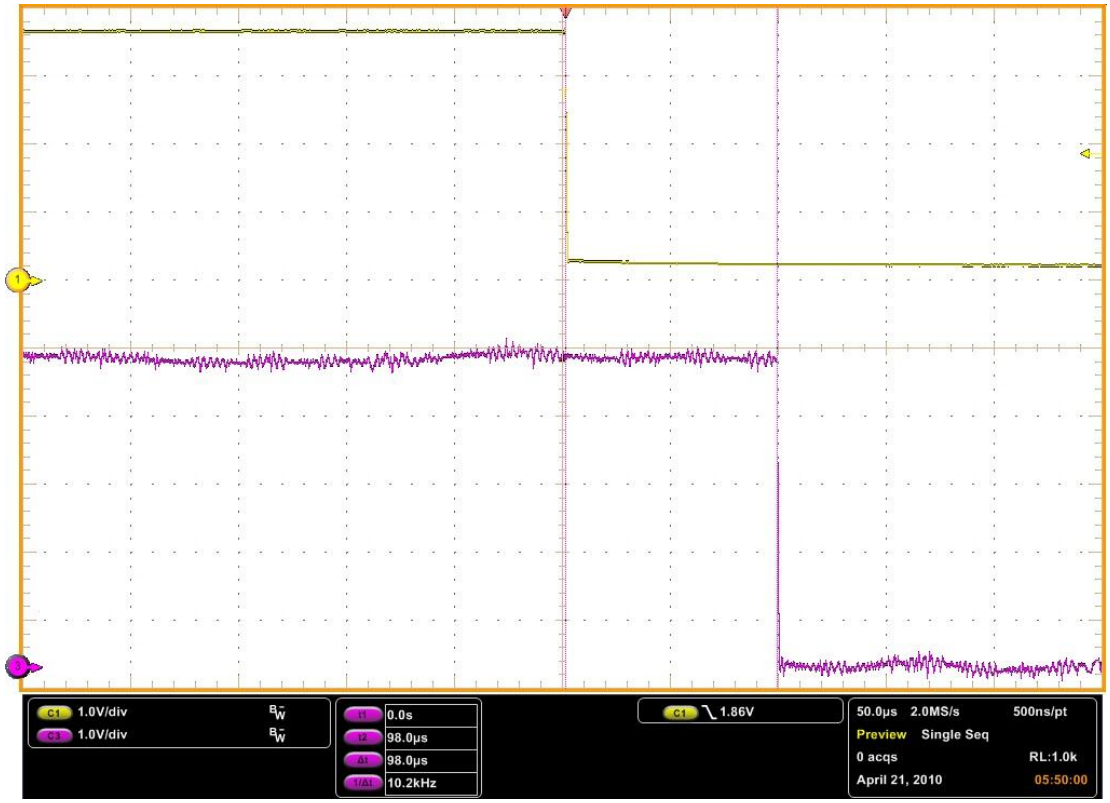


Abbildung 3.2*: Messung der Signallaufzeit von DÜSY*

In einer zweiten Messung wurde der Systemjitter bestimmt. Der Jitter beschreibt die Varianz der Signallaufzeit, also die Taktungenauigkeit der Übertragung. Der Takt ist nicht konstant, sondern unterliegt gewissen Schwankungen.
In der Messung, deren Ergebnis in Abbildung 3.3 zu sehen ist, sind eine Vielzahl von Pegelwechseln (Lowpegel auf Highpegel) in einem Augendiagramm dargestellt. Die Farbskallierung zeigt, wie viele Signale an einem bestimmten Punkt überlappen. Grün entspricht wenigen Überlappungen, über Gelb bis Rot zu vielen.

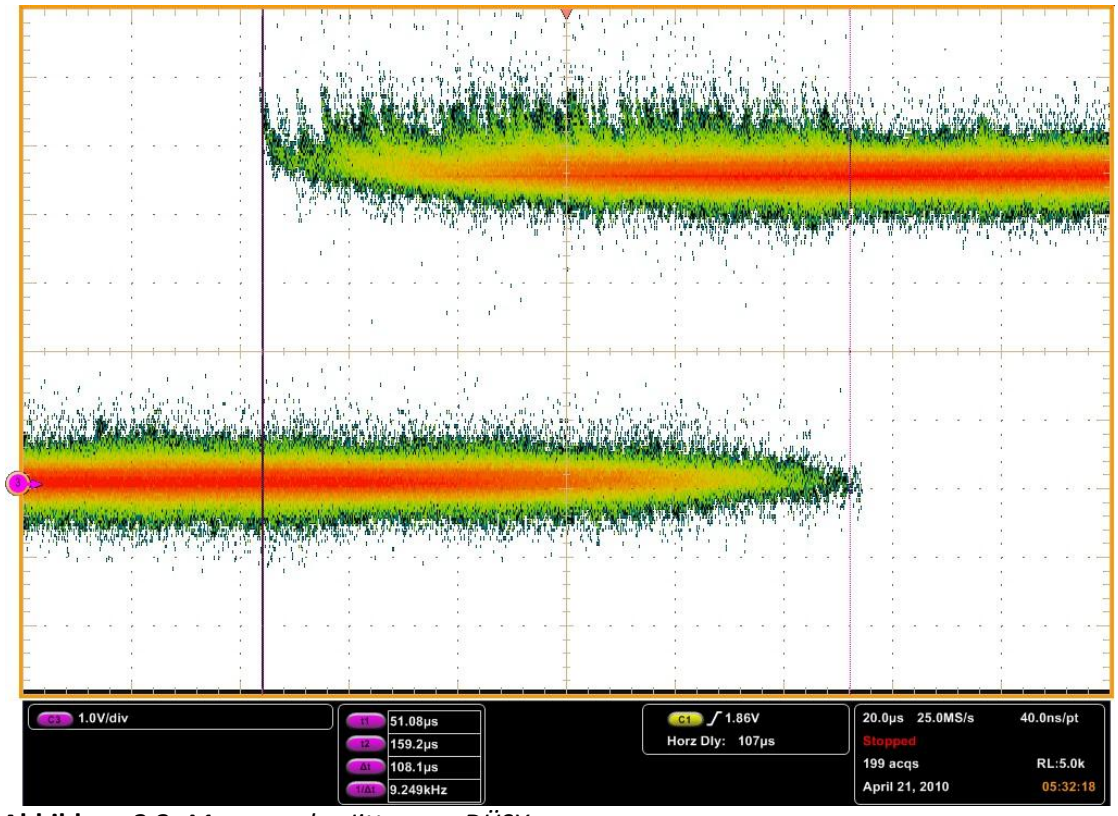


Abbildung 3.3: *Messung des Jitter von DÜSY*

Diese Schwankungen ergeben sich, da die Daten nicht synchron mit dem Systemtakt von DÜSY eingespeist werden.

3.1.3 Anforderungen an das Übertragungssystem

Um festzulegen, ob ein Bussystem als Ersatz für DÜSY und zum anschließenden Datentransfer zu den Empfängern geeignet ist, müssen Mindestanforderungen an das System definiert werden. All diese Anforderungen müssen zwingend von dem neuen System erfüllt werden.

Folgende vier Anforderungen sind gegeben:

- Das System muss **echtzeitfähig** sein. Aus den Messungen (siehe Kapitel 3.1.2) hat sich ergeben, dass DÜSY einen Systemjitter 49 µs hat. Dies entspricht somit auch der maximalen Zeit, die für das Senden von Daten benötigt wird. Für das neue System wurde festgelegt, dass es eine maximale Übertragungszeit von 10 µs haben darf. Damit wäre das System, nach eigener Definition, echtzeitfähig.

- Eine weitere wichtige Anforderung ist eine **geringe Störemission**. Da das Gesamtgebilde Radioteleskop Effelsberg sehr empfindlich ist und eine starke Emission von Störstrahlung Messergebnisse verfälschen könnte, muss diese auf ein absolutes Minimum reduziert werden.

- Da im Kabelbaum Platz für andere Kabel geschaffen werden soll, wird ein **serielles Bussysteme** vorrausgesetzt.

- Als letzter wichtiger Parameter des Systems ist die **Übertragungssicherheit** zu nennen. Da es aufgrund der oben genannten Echtzeitfähigkeit ausgeschlossen ist, dass ein Übertragungsverfahren gewählt wird, das Fehler erkennt und anschließend die Daten erneut sendet, muss gewährleistet werden, dass die Daten mit einer hohen statistischen Genauigkeit richtig übertragen werden.

3.2 Untersuchung serieller Bussysteme

Busse sind Systeme, bei denen Kommunikation zwischen mehreren Teilnehmern über ein gemeinsames Übertragungsmedium realisiert wird. Grundlegend wird zwischen parallelen und seriellen Bussen unterschieden. Bei der parallelen Technik werden die Daten über mehrere Leitungen zeitgleich gesendet. Serielle Bussysteme haben dagegen nur eine Datenleitung, über die Bit für Bit die komplette Nachricht versendet wird.

Vorteile serieller Bussysteme:

- Geringer Kabelaufwand
- Größere Distanzen zwischen Sender und Empfänger realisierbar

Nachteile serieller Bussysteme:

- Bei gleicher Bitdauer nimmt die Übertragungszeit im Vergleich zur parallelen Datenübertragung zu.

Im Folgenden werden einige serielle Bussysteme dahingehend theoretisch untersucht, ob sie als Ersatzsystem für DÜSY in Frage kommen.

3.2.1 Can – Bus

Der Controller – Area – Network Bus (kurz: Can-Bus) wurde im Jahre 1987 von den beiden Firmen Bosch und Intel vorgestellt.
Der Bus überträgt die Daten seriell über zwei Leitungen, CanH[1] und CanL[2], symmetrisch (vgl. [MEM] und [ETS]). Dies dient (wie in Kapitel 2.1.4 beschrieben) zur Fehlervermeidung. Can unterstützt die Linientopologie (auch Bustopologie genannt) und unter größerem Aufwand auch die Sterntopologie.
Beim dominanten Zustand muss zwischen den beiden Leitungen mindestens ein Spannungsunterschied von 3,5 V und beim Rezessiven höchstens 1,5 V anliegen.
Da an allen Busteilnehmern eine gesendete Nachricht annähernd gleichzeitig anliegen muss, ist die maximale Kabellänge von der verwendeten Bitrate abhängig. Die nebenstehende Tabelle zeigt die empfohlenen Kabellängen bei bestimmten Bitraten.

Bitrate	Max. Kabellänge
10 kbit/s	6700 m
20 kbit/s	3300 m
50 kbit/s	1300 m
125 kbit/s	530 m
250 kbit/s	270 m
500 kbit/s	130 m
1 Mbit/s	40 m

Damit es zu keinen Reflexionen an den Leitungsenden kommt, sollten diese reflexionsfrei mit einem 120 Ω Widerstand an beiden Enden terminiert werden.
Die maximale Anzahl der unterstützen Knoten ist vom verwendeten Treiber abhängig. Leistungsstarke Treiber unterstützen bis zu 150 Teilnehmer pro Bus.

SOF	Ident	RTR	IDE	r0	DLC	DATA	CRC	ACK	EOF
1	11	1	1	1	4	0 – 64	15	2	10

Abbildung 3.4: *Frameaufbau Can-Bus*

Ein Nachrichtenframe, auch Nachrichtenpaket genannt, wird wie in Abbildung 3.4 dargestellt[3], verwirklicht. Hierbei handelt es sich um den Aufbau eines Standardframes, der Extended Frame ist etwas länger, wird hier jedoch nicht näher betrachtet.

In einem Frame können bis zu 64 Bit Information übertragen werden. Des Weiteren besteht ein Nachrichtenpaket noch aus 46 Overhead-Bit:

- **Start of Frame (kurz: SOF):** Das Startbit ist immer dominant und kennzeichnet den Beginn eines Frames. Auf dieses Bit synchronisieren sich alle Busteilnehmer.
- **Identifier (kurz: Ident):** Es erfolgt keine direkte Adressierung eines Knotens, stattdessen werden Parameter übertragen, wie zum Beispiel Systemtemperatur. Jeder Knoten entscheidet, ob er diesen Paramter lesen soll.
  Des Weiteren enthalten die Identifierbit eine Priorisierung der Nachricht.
- **Remote Transfer Request (kurz: RTR):** Unterscheidet, ob es sich um ein Daten- oder ein Datenanforderungstelegramm handelt. Datenanforderungstelegramme enthalten keine Datenbit.
- **Identifier Extension (kurz: IDE):** Gibt an, ob es sich um einen Standard- oder einen Extended Frame handelt.

1: CanH: Can High 2: CanL: Can Low 3: untere Beschriftung in Abb. 3.4 gibt Größe in Bit an

- **r0:** Ist immer rezessiv, dient dem Framecheck (*siehe weiter unten*), für spätere Entwicklungen varriabel belegbar.
- **Data Length Code (kurz: DLC):** Informiert Knoten über die Länge des folgenden Datenblocks.
- **DATA:** Informationsbit, zu übertragende Nachricht mit einer maximalen Länge von 8 Byte.
- **Cyclic Redundarncy Check (kurz: CRC):** Redundante Prüfbit, zur Fehlerkorrektur *(siehe weiter unten)*
- **Acknowledgement (kurz: ACK):** Rückmeldung, ob Daten richtig an den Empfangs-knoten angekommen sind (siehe weiter unten)
- **End of Frame (kurz:EOF):** Kennzeichnet das Ende des Nachrichtenframes.

Der Can-Bus hat fünf verschiedene Mechanismen implementiert, die mögliche Fehler erkennen sollen:

1) **Monitoring:** Jeder Knoten vergleicht beim Senden seiner Nachricht gleichzeitig die Pegel auf der Leitung. Kommt es zu Differenzen, so liegt ein Fehler vor. Dies geschieht beispielsweise, wenn zwei Knoten gleichzeitig senden.

2) **Bit Stuffing:** Durch das Verwenden einer Non-Return-to-Zero Codierung, werden nach fünf aufeinanderfolgenden Bit eines Pegels, ein Bit mit dem gegenteiligen Pegel eingefügt und später vom Empfänger wieder entfernt. Somit entstehen keine langen Folgen eines Pegels, der dazu führen könnte, dass die Synchronität verloren geht.

3) **Cyclic Redundancy Check:** Der sendende Knoten fügt der Nachricht die CRC- Prüfsumme bei. Diese wird mit Hilfe einer genau festgelegten Rechenvorschrift bestimmt. Der Empfänger berechnet diese Summe ebenfalls selbstständig und vergleicht sein Ergebnis mit der mitgesendeten Prüfsumme.

4) **Framecheck:** Beim Framecheck überprüft jeder Empfänger, ob der empfangene Frame das richtige Format hat. Dies beinhaltet die richtige Länge und die richtigen Pegel von SoF, r0 und Eof.

5) **ACK-Fehler:** Der korrekte Empfang einer Nachricht wird von jedem Knoten durch das Setzen eines Acknowledgement bestätigt. Bleibt diese Nachricht aus, so kann es sein, dass der Empfänger nicht erkannt wurde, sei es durch Fehlen des Knotens oder durch Kabelschäden oder dass der Empfänger einen Fehler erkannt hat.

Bei der Verwendung des Can-Busses darf jeder Knoten ohne vorherige Zustimmung eines Masters Daten senden. Durch das Verfahren Carrier-Sense-Multiple-Access/Collision-Resolution (kurz: CSMA/CR) wird durch Bit-Arbitrierung eine Priorisierung der Nachrichten realisiert. Jeder sendende Knoten kontrolliert seine eigene Nachricht. Ist im Identifier ein dominanter Zustand durch einen Rezessiven überschrieben, so sendet zeitgleich ein Knoten mit einer höheren Priorisierung.
In diesem Fall wird die Nachricht nach einer Zufallszeit erneut gesendet. Dadurch ist der Can-Bus nicht deterministisch, das heißt es ist nicht sicher vorhersagbar, wann ein Nachrichtenpaket übermittelt wird.

3.2.2 I²C – Bus

Der Inter-IC-Bus, kurz I²C-Bus genannt, ist ebenfalls ein 2-Drahtbus. Die Daten werden nicht symmetrisch übertragen, sondern nur über eine der beiden Leitungen. Diese heißt: serial data (kurz: SDA). Über das zweite Kabel, serial clock (kurz: SCL), wird ein Taktsignal übertragen. Dieses dient zur Synchronisation aller Kommunikationsteilnehmer.
Der Buszugriff erfolgt nach dem Master – Slave – Verfahren. Dabei regelt ein Kommunikationsteilnehmer, der Master, welcher Knoten senden darf. Dies geschieht über das Carrier-Sense-Multiple-Access/Collision-Avoidance- Verfahren (kurz: CSMA/CA). In Abbildung 3.5 ist die Funktionsweise verdeutlicht. Wenn eine Station senden möchte, überprüft sie, ob bereits ein anderer Knoten sendet. Sobald die Leitung frei ist, beginnt der Knoten mit der Datenübertragung. Dabei horcht er immer auf der Leitung, ob nicht doch noch ein Teilnehmer sendet. Sollte es zu einer Kollision zweier Nachrichten kommen, so erkennen beide sendenden Knoten dies und übertragen jeweils ein JAM-Signal. Nach einer Zufallszeit beginnt der sendewillige Knoten wieder mit der Überwachung des Mediums. Sollte die vorher konfigurierte maximale Anzahl an Übertragungsversuchen erreicht sein, so wird eine Fehlermeldung an die Netzwerkkarte geschickt.

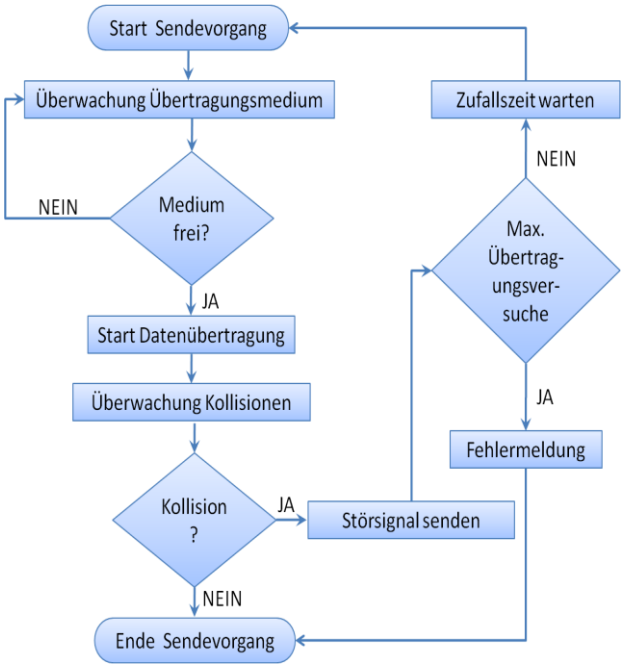


Abbildung 3.5: *CSMA/CA*

Die beschriebene Masterfunktion kann von unterschiedlichen Einheiten dynamisch geregelt werden. Diese Eigenschaft wird Multimasterprinzip genannt.
Der Aufbau einer I²C-Nachricht (Abbildung 3.6)[1] ist sehr einfach. Nach einem Startbit, das den Anfang einer Nachricht kennzeichnet, folgt die Adresse des Zielknotens. Diese ist im Allgemeinen 7 Bit lang, kann aber mittlerweile bei großen Netzwerken auf 10 Bit erweitert werden. Es folgt das Read/Write-Bit, das zur Unterscheidung von Lese- bzw. Schreibaufforderungen dient. Die Acknowledgement-Bit vor und hinter den Datenbit werden von der Empfangseinheit versendet. Mit dem Ersten wird bestätigt, dass der Empfänger bereit ist und mit dem Zweiten kann er weitere Daten anfordern. Die Daten können eine Maximallänge von einem Byte haben. Abschließend wird das Stopbit versendet.

Start	Adresse	R/W	ACK	Daten	ACK	Stop
1	7	1	1	8	1	1

Abbildung 3.6: *Frameaufbau I²C*

Bitrate	Bezeichnung
100 kbit/s	Standard Mode
400 kbit/s	Fast Mode
1Mbit/s	Fast Mode Plus
3,4 Mbit/s	High Speed Mode

I²C hat keine Maßnahmen zur Fehlererkennung implementiert. Somit kann eine sichere Übertragung nicht gewährleistet werden. Maximal unterstützt der Bus eine Übertragungsrate von 3,4 Mbit/s. Vorraussetzung dafür ist, dass High Speed Mode verwendet werden. In der nebenstehenden Tabelle sind alle gängigen Übertragungsraten dargestellt.

1: untere Beschriftung in Abb. 3.6 gibt Größe in Bit an

3.2.3 RS485 - Bus

Der RS485- oder EIA485- Bus überträgt alle Daten symmetrisch, damit der Einfluss von Gleichstromstörungen minimiert wird (siehe Kap. 2.1.4).
Dieser kann wahlweise über eine, zwei oder vier Drähte betrieben werden. Bei der Realisierung über zwei Drähte ist nur ein Halbduplexbetrieb möglich, das heißt dass immer nur eine Station senden kann. Ein großer Vorteil dieser Lösung ist die Multimasterfähigkeit, so dass jede Station mit jeder anderen Daten austauschen kann.

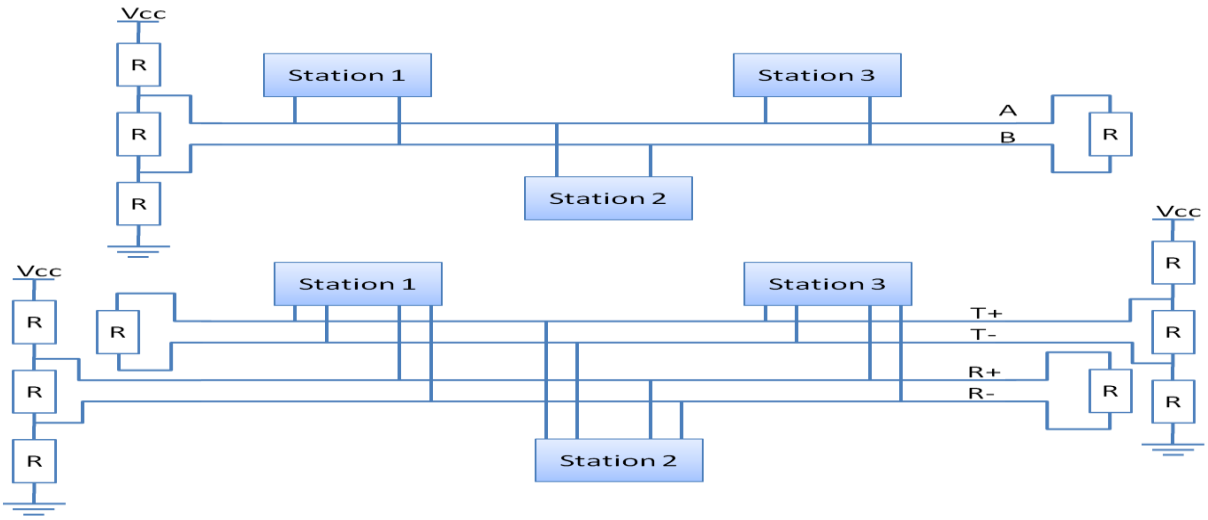


Abbildung 3.7: *RS485 oben Zweidrahtlösung, unten Vierdrahtlösung*

Bei der Variante mit vier Drähten, ist ein Drahtpaar für gesendete Daten vom Master und das Andere ausschließlich für Antworten von den einzelnen Slaves zum Master hin.
Somit ist ein Vollduplexbetrieb, jedoch kein Multimasterbetrieb möglich.
Damit es zu keinen Timing-Problemen kommt, müssen die Stichleitungen möglichst kurz gewählt werden, so dass sie 5 m nicht überschreiten.

Adresse	Framelänge	Daten	Checksum
8	8 v 16	0 – 512k	8 v 16

Abbildung 3.8: *Frameaufbau RS485*

Ein Frame ist wie in Abbildung 3.8[1] dargestellt, sehr einfach aufgebaut. Nach acht Adressbit folgen die Bit, die die Framelänge mitteilen. Wenn das folgende Datenpaket kleiner als 255 kByte ist, reichen 8 Bit um die Framelänge darzustellen. Sollten die Datenmenge größer sein, so muss das Framelängenfenster auf 16 Bit ausgeweitet werden.
Am Ende des Frames können optional verschiedene Checksummen zur Fehlererkennung angefügt werden. Möglich sind 8 Bit- oder 16 Bit-Checksum oder die 16 Bit lange CRC – Prüfsumme. Die in der Reihenfolge aufsteigend eine Vielzahl von Fehlern erkennen. Auch bei dem RS485-Bus ist die Bitrate von der verwendeten Kabellänge abhängig. Standardmäßig lassen sich 128 Geräte anschließen. Einige sehr leistungsstarke Treiber unterstützen 255 Knoten.

Bitrate	Max. Kabellänge
93,75 kbit/s	1200 m
500 kbit/s	400 m
10 Mbit/s	10 m

1: untere Beschriftung in Abb. 3.8 gibt Größe in Bit an

3.2.4 Flexray

Flexray ist ein neuartiges Bussytem, das aus der Automobilindustrie kommt. Datenpakete werden symmetrisch über Zweidrahtleitungen übertragen. Mit der standardisierten Übertragungsrate von 10 Mbit/s ist eine Kabelgesamtlänge von 22 m möglich. Bei einer deutlich kürzeren Übertragungsstrecke können maximal 22 Knoten implementiert werden. Es wird nicht nur die Bustopologie unterstützt, sondern auch die normale und die kaskadierte Sterntopologie. Bei Letzterer ist eine Kabellänge von 72 m möglich.

Abbildung 3.9: *Zyklusaufbau Flexray*

Ein Sendezyklus bei Flexray ist in vier ungleichgroße Segmente aufgeteilt (siehe Abbildung 3.9). Im static segment werden deterministisch Daten übertragen. Dies geschieht in dem dieses Segment in gleichgroße Slots unterteilt ist, deren Anzahl immer konstant ist. Jedem Kommunikationsteilnehmer sind bestimmte Slots zugeordnet. Somit sind die Nachrichten hier immer gleich lang. Sollte ein Knoten in einem Zyklus keine oder weniger Daten senden wollen, so können die Slots frei bleiben. Jede Einheit hat einen Slotzähler um die Daten, die für ihn bestimmt sind zu erkennen.

Das dynamic segment ist ebenfalls in Slots eingeteilt. Diese sind jedoch bedeutend kleiner als die im static segment. Sollte ein Knoten zusätzlichen Bedarf haben Daten zu senden, wird ihm hier ein Slot zugewissen, der sich der Länge der Nachricht anpasst. Sollte es nicht möglich sein in der Restzeit dieses Segmentes die Nachricht zu senden, so muss bis zum nächsten Zyklus gewartet werden. Durch die Reihenfolge der Slotzuteilung werden die Nachrichten indirekt priorisiert, da die Wahrscheinlichkeit bei späteren Nachrichten größer ist, das sie nicht mehr ins Segment passen.

Im dritten Bereich, dem symbol window werden ausschließlich Steuerbefehle für die Knoten übertragen. Dies sind beispielsweise Wake Up- oder Shut Down- Befehle.

In der Network Idle Time (in Abbildung 3.9: NET) werden Synchronisationsmuster übertragen, um die Uhren der einzelnen Teilnehmer einheitlich laufen zu lassen.

Das Verhältnis von static segment zu dynamic segment lässt sich frei konfigurieren. Um einen Netzausfall auszuschließen, ist während Wartezeiten ohne Übertragung ein Pegel angelegt, der sich von den Signalpegeln unterscheidet. Sollte es zu einem Pegelabfall kommen, so wird dies als Fehler erkannt. Jedes einzelne Segment ist mit Fehlererkennungsmechanismen wie CRC-Prüfsummen gesichert.

3.2.5 Ethernet

Der Standard IEEE 802.3 beschreibt eine Vielzahl von Ethernetvarianten. Diese unterscheiden sich nach Schnabel [SCP], wie in unten stehender Tabelle[1] dargestellt, in ihren Eigenschaften wie Übertragungsrate und zu verwendendes Übertragungsmedium.

Ethernet Standard	Bezeichnung	Datenrate	Kabeltyp / Maximale Kabellänge
802.3	10Base5	10 Mbit/s	Koaxialkabel / 500 m
802.3a	10Base2	10 Mbit/s	Koaxialkabel / 185 m
802.3i	10Base-T	10 Mbit/s	Twisted-Pair-Kabel
802.3j	10Base-FL	10 Mbit/s	Glasfaserkabel
802.3u	100Base-TX	100 Mbit/s	Twisted-Pair-Kabel / 100 m
802.3u	100Base-FX 100Base-SX	100 Mbit/s	Glasfaserkabel
802.3z	1000Base-SX 1000Base-LX	1 Gbit/s	Glasfaserkabel
802.3ab	1000Base-T	1 Gbit/s	Twisted-Pair-Kabel / 100 m
802.3ae	10GBase-SR 10GBase-SW 10GBase-LR 10GBase-LW	10 Gbit/s	Glasfaserkabel
802.3an	10GBase-T	10 Gbit/s	Twisted-Pair-Kabel / 100 m

Ein Standardframe wie in Abbildung 3.10[2] aufgebaut. Bei einzelnen Varianten weicht der Aufbau geringfügig ab.

Abbildung 3.10: *Frameaufbau Ethernet*

- **Präambel (PA):** 010101-Folge zur Synchronisation aller Teilnehmer
- **Destination adress (DA):** MAC-Adresse des Empfängers; muss im Netzwerk eindeutig sein
- **Source adress (SA):** MAC-Adresse des Senders; muss im Netzwerk eindeutig sein
- **Type:** Enthält je nach Variante Informationen für das verwendete Osi-Schicht-3-Protokoll oder über die Gesamtlänge der Frames
- **Data und Padding (PAD):** Dieses Feld enthält die zu übertragenden Daten. Es darf allerdings eine Größe von 46 Byte nicht unterstreiten, da diese zu Problemen bei der Kollisionserkennung führen würde. Sind zu wenig Daten zu übermitteln, so wird das Datenfeld mit den Padding-Bit aufgefüllt.
- **Frame Check Sequenz (FCS):** Prüfsumme zur Fehlererkennung

1: Tabelle ist nur ein Ausschnitt der Standards und hat keinen Anspruch auf Vollständigkeit
2: untere Beschriftung in Abb. 3.10 gibt Größe in Bit an

Da alle Standard-Ethernet-Varianten mit CSMA/CD Verfahren (siehe Kapitel 3.2.2) arbeiten, ist es nicht deterministisch. Werden Kollisionen zweier Nachrichten erkannt, wird eine zufällige Zeit gewartet, bis erneut gesendet wird.

Um diesen entscheidenden Nachteil auszugleichen, gibt es Real-Time-Ethernet-Varianten. Diese stellen mit unterschiedlichen Mechanismen die Datenübertragung in bestimmten Zeiten sicher. Im Folgenden werden drei Real-Time-Ethernet-Varianten vorgestellt, die für die gegebene Problemstellung als Lösung in Frage kommen.

3.2.6 Realtime Ethernet Applikation: Powerlink

Powerlink ist ein streng deterministisches Echtzeitprotokoll. Es arbeitet mit einer Übertragungsrate von 100 Mbit/s. Es verbessert zwei Eigenschaften die „normales“ Ethernet Nicht-Echtzeitfähig macht. Zum Ersten müssen Kollisionen ausgeschlossen werden. „Normales“ Ethernet arbeitet mit CSMA/CD, hier horcht zwar die Sendestation, ob die Leitung frei ist, es kann aber zu Kollisionen kommen, wenn zwei Stationen gleichzeitig mit dem Senden beginnen. Powerlink segmentiert zum Einen das Netz durch den Gebrauch von Switches. Dadurch werden so genannte „Collision Domains“ gebildet, Quasi-Punkt-zu-Punkt-Verbindungen. Zum Anderen wird die Kommunikation sehr strengen Regeln unterworfen. So bekommt jede Sendestation einen festen Zeitrahmen, in dem nur sie senden darf. Die zweite Verbesserung ist das Ersetzen des TCP Stacks durch einen so genannten Slot-Communication-Network-Manager-Stack (kurz: SCNM Stack). Dieser Stack ist echtzeitfähig, da er die Latenzzeit, also die Reaktionszeit stark reduziert und auch jedes einzelne Datenpaket schneller bearbeiten kann.
Bei dem Ethernet-Echtzeit-Ansatz „Powerlink“ startet nach Prof. Dr. Schwanger [SC2] der Master mit einer Broadcastnachricht einen Sendezyklus, dem „Start of Cyclic Frame (SoC)“. Mit Hilfe dieser Nachricht werden alle Teilnehmer synchronisiert.
Nun beginnt der zyklische Datenaustausch. Der Manager gibt der ersten Station das Kommunikationsrecht mit einer kurzen Nachricht (Abbildung 3.11-1). Diese sendet dann auf direktem Wege eine Broadcast-Nachricht an alle Stationen (Abbildung 3.11-2). Jede einzelne Station kann nun entscheiden, ob die Nachricht für sie interessant ist, oder nicht. Diese Prozedur wiederholt sich nun für jede Station (siehe Abbildung 3.11).

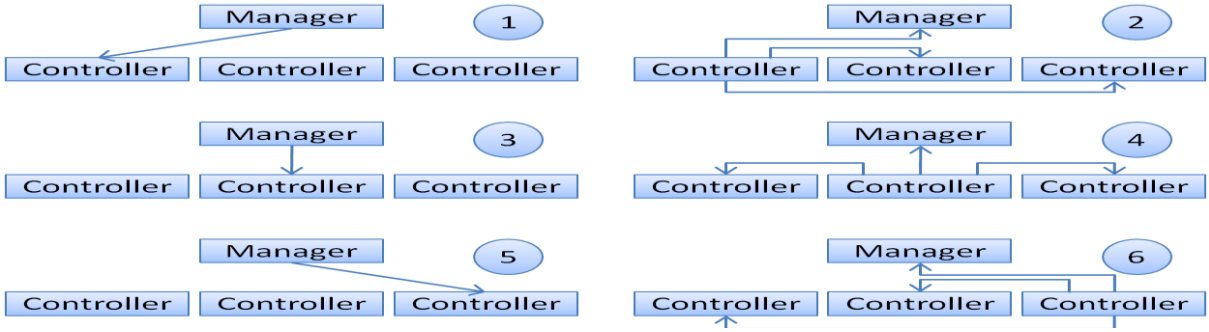


Abbildung 3.11: *Nachrichtenfolge Cycle Period Powerlink*

Ist die letzte Station mit dem Senden fertig, beginnt die „Asynchron Period“. In dieser kann eine Station die Bedarf hat noch weitere, dringende Daten in einem bestimmten, frei definierbaren Zeitfenster senden. Den Sendebedarf muss die Station allerdings bereits in der „Cycle Period“ im Header ihrer Nachricht anmelden. Ist dies abgeschlossen, so kommt noch die „Idle-Period“. In diesem Zeitfenster wartet das System auf den Start eines neuen Zyklus. Die Dauer dieser Periode ist abhängig von der Dauer der Asynchronen Phase. Nun kann ein neuer Zyklus Starten.

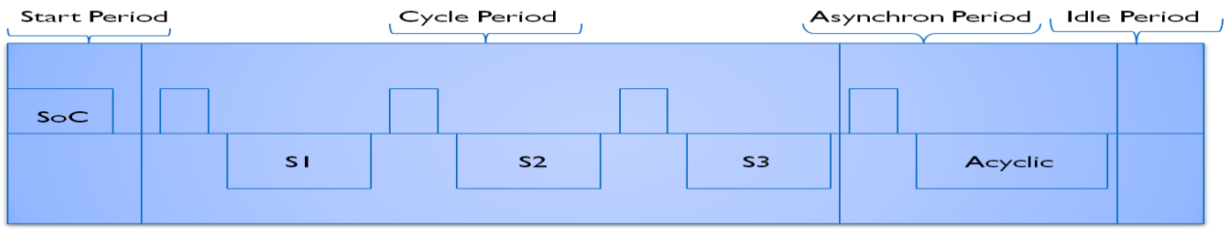


Abbildung 3.12: *Powerlink Period*

Ein weiterer Vorteil von Powerlink ist, dass bei Verwendung von Hubs beliebige Topologien realisierbar sind.
Nachteilig ist, dass redundante Pfade vermieden werden müssen und dass man keine Standard Ethernet- Bausteine nutzen kann, sondern spezielle Powerlink-Geräte.

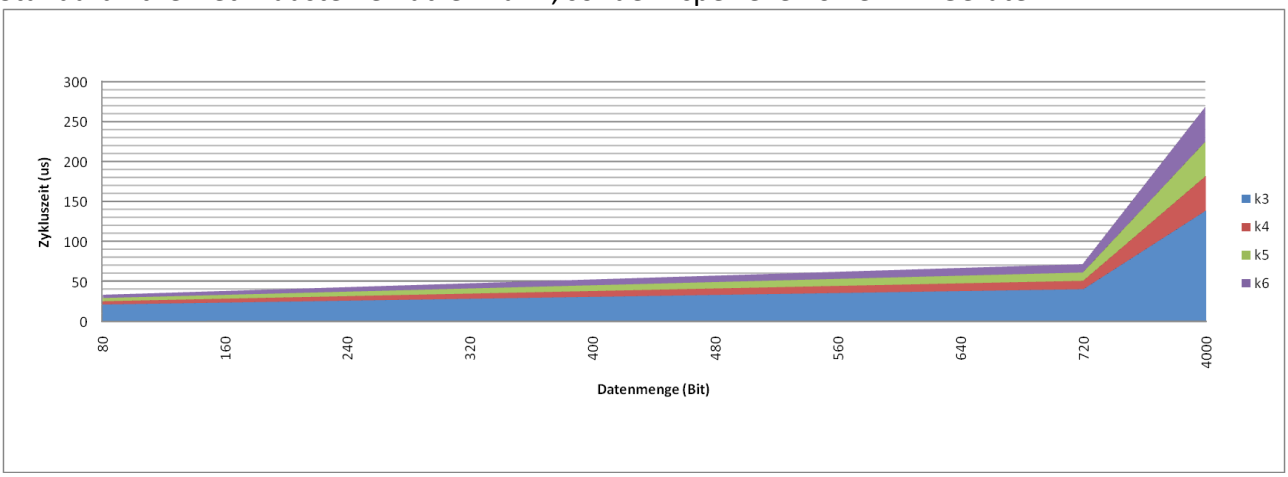


Abbildung 3.13: *Zeitverhalten Powerlink mit voller Datenlast*

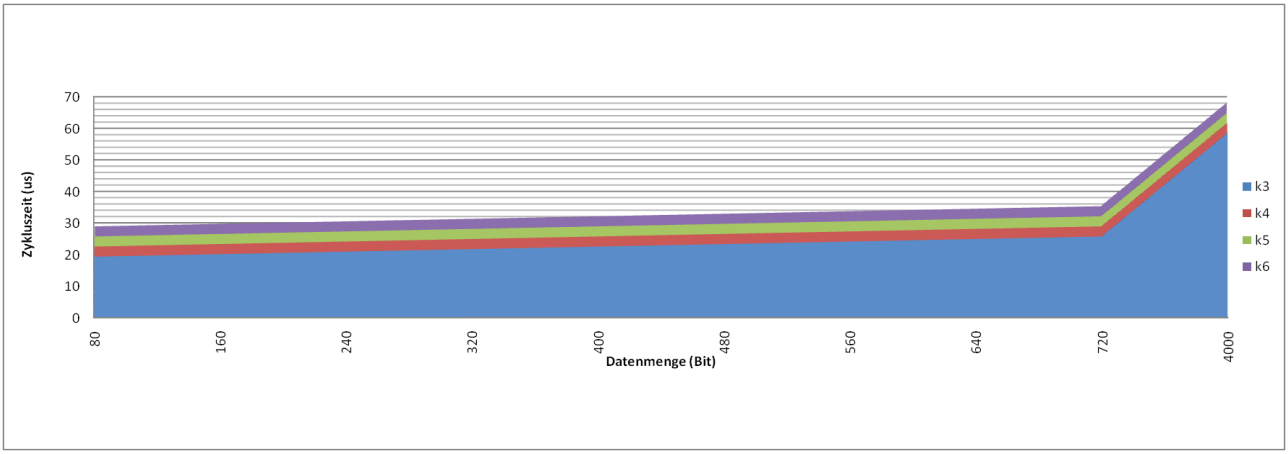


Abbildung 3.14: *Zeitverhalten Ethernet Powerlink mit Datenlast an einem Knoten*

In den Abbildungen 3.13 und 3.14[1] ist das Zeitverhalten von Powerlink unter der Annahme dargestellt, dass keine Daten im asynchronen Fenster gesendet werden und ohne Berücksichtigung von Totzeiten zwischen einzelnen Nachrichten. Im oberen Diagramm wird dargestellt, wie sich die Zykluszeit verändert, wenn an allen Knoten Daten anliegen. Die blaue Fläche verdeutlicht das Verhalten bei 3 Knoten bis zur lila Fläche mit 6 Knoten. Es ist zu sehen, dass bei der minimalen Datenmenge von 80 Bit, die für das System geforderten 10 µs nicht erreicht werden.

Im Betrieb ist es meistens der Fall, dass nicht an allen Knoten Daten anliegen, sondern nur an einem Einzelnen. Dies ist im unteren Diagramm dargestellt. Es ist zu sehen, dass die Zykluszeiten mit steigender Datenmenge weniger stark zunehmen, aber auch hier die Anforderungen nicht erreicht werden können.

1: Waagerechte Achsen nicht linear ! Senkrechte Skalierung unterschiedlich

3.2.7 Realtime Ethernet Applikation: Profinet

Profinet wird in zwei verschiedenen Ausführungen angeboten. Einmal die im Folgenden betrachtete Profinet IO und zum anderen Profinet CBA (Component Based Automation). Profinet IO beschreibt die Lösung zur Übertragung von Daten im Feldbereich über Ethernet. Profinet CBA hingegen ist ausschließlich zur Vernetzung verteilter Anlagen.
Profinet IO teilt seine Nachrichten in drei große Kategorien ein.

- Normale Nachrichten: Dies sind die zeitunkritischen Daten, die über TCP/IP übertragen werden können.
- Real-Time Nachrichten (asynchroner Modus): Dieses sind zeitkritische Daten, die schneller übertragen werden müssen. Durch eine Priorisierung im Header wird diesen Nachrichten eine Wertigkeit gegeben. Es handelt sich um eine reine Softwarelösung.
- Isochrone Real- Time Nachrichten (synchroner Modus): Dies sind Nachrichten, für die selbst der Realtime Mode noch zu langsam ist. Hier ist nicht nur eine Softwarepriorisierung vorgenommen, sondern spezielle IRT Bausteine verwendet worden. Diese beinhalten einen Switch, so dass der separate Switch gespart werden kann. Allerdings sind derartige Nachrichten nur im gemeinsamen Adressraum zu übertragen, was ein gleiches Subnetz voraussetzt. Gleichzeitig wird damit aber neben der Stern- auch die Linien- und Ringtopologie möglich.

Abbildung 3.15: *Frameaufbau Profinet*

Es ist auch eine Realisierung möglich, die nur aus dem Standard-Kanal besteht, auch die Größe der einzelnen Phasen ist frei konfigurierbar.
Bei diesem Verfahren müssen nicht zwangsläufig Switches eingesetzt werden, da keine Segmentierung gefordert wird.
Der IRT Modus arbeitet nur im Vollduplexbetrieb, was zwingend Kabelpaare benötigt. Da alle Nachrichten gleich schnell übertragen werden müssen, ist eine Priorisierung der Nachrichten für das System in Effelsberg unvorteilhaft. Aus diesem Grund kommt die Echtzeitvariante Profinet nicht als DÜSY-Ersatz in Frage.

3.2.8 Realtime Ethernet Applikation: EtherNet/IP

EtherNet/IP arbeitet mit einer Übertragungsrate von 10 Mbit/s oder 100 Mbit/s. Es hat einen ähnlichen Ansatz wie Powerlink. Kollisionen werden durch den Einsatz von Switches verhindert. Auch hier werden dadurch Quasi-Punkt-zu-Punkt Verbindungen geschaffen. Des Weiteren beinhaltet EtherNet/IP eine Protokollerweiterung namens CIP (Common Industrial Protocol), das zur zeitlichen Verbesserung der Übertragung zeitkritische Daten nicht über TCP (sogenannte explizite Nachrichten) sondern über UDP (sogenannte implizite Nachrichten) sendet. Dieses Protokoll hat einen kleinen Header und hat ein berechenbares Zeitverhalten, da es nicht zu einer unberechenbaren Anzahl von Wiederholungen führt. Es arbeitet nach dem Producer/Consumer - Modell, d.h. Geräte melden ihren Bedarf für den Empfang bestimmter Daten als „Consumer“ an und stellen wiederrum den Netzwerkteilnehmern eigene Daten als „Producer“ zur Verfügung. Bevor Cip- Dateien in die Ethernet/IP- Umgebung gesendet werden können, werden diese mit einem zusätzlichen Header versehen. Dieser enthält unter anderem Informationen über das Format der Nachricht und den Sendestatus. Dieses Verfahren wird Encapsulation genannt.

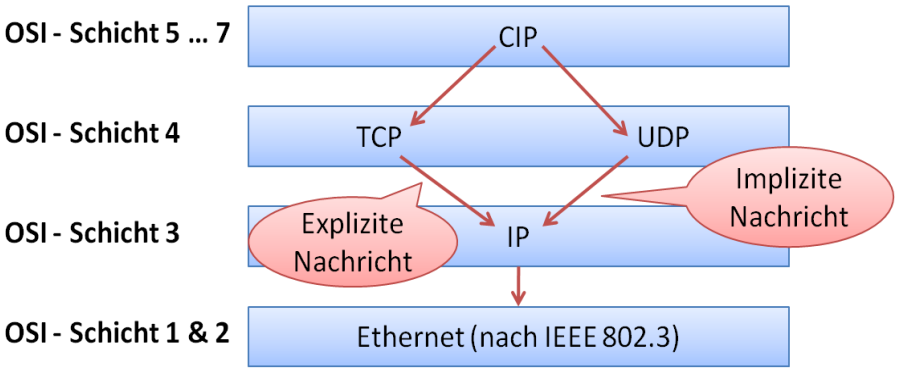


Abbildung 3.16: *Schichtenmodell Ethernet/IP mit CIP Erweiterung*

Zur weiteren Verbesserung der Echtzeitfähigkeit wird nach Prof. Dr. Schwanger [SC1] eine Implementierung des IEEE-1588 Protokolls namens CIPsync empfohlen. Durch CIPsync werden zum Einen alle Teilnehmer mit einer Genauigkeit von 0,5 µs synchronisiert. Die Synchronisationsdaten werden ca. einmal pro Sekunde gesendet. Des Weiteren ist eine Priorisierung zeitkritischer Daten möglich.
Ethernet/IP ist nur in einer Sterntopologie mit einem Anschluss pro Strahl realisierbar. Außerdem arbeitet Ethernet/IP im Vollduplexbetrieb, dies bedeutet auch hier, dass immer Leitungspaare verlegt werden müssen.
Auch hier ist, wie bei Profinet, eine Priorisierung der Nachrichten notwendig und somit für die gegebene Problemstellung ungeeignet.

3.2.9 Zwischenfazit

Die theoretischen Untersuchungen der seriellen Bussysteme haben ergeben, dass kein System die Anforderungen erreicht.
So erreicht der Can-Bus bei einer Kabellänge von ca. 400m nur eine Übertragungsrate von 125 kBit/s. Der I²C-Bus ist nicht deterministisch, da es mit CSMA/CD arbeitet. Dasselbe gilt für Standard-Ethernet. RS485 überträgt auch bei der gegebenen Kabellänge nicht ausreichend schnell. Flexray unterstützt keine Übertragungsstrecken dieser Länge.
Die Echtzeit-Varianten von Ethernet sind auch nicht geeignet. Powerlink ist vom Protokoll her zu komplex, so dass selbst bei minimaler Auslastung die Übertragung um den Faktor drei zu lange dauern würde. Profinet arbeitet mit einer Priorisierung der Nachrichten. Dies ist für unser Problem nicht geeignet, da über das System ausschließlich die 24 Steuersignale für die Empfänger übertragen werden sollen und somit alle Nachrichten gleich wichtig sind. Ähnliches gilt auch für Ethernet/IP. Auch hier wird mit Priorisierungen gearbeitet. Des Weitern wird bei zeitkritischen Daten TCP durch UDP ersetzt. Dieses Netzwerkprotokoll bietet keine ausreichende Sicherheit gegenüber Datenverlusten.
Da kein System für das gegebene Problem geeignet ist, muss ein eigenes System entworfen werden.
Das neue System, SERELECS, hat dieselben Systemanforderungen, die ein kommerziell erworbenes gehabt hätte (siehe Kapitel 3.1.3).

4 Entwurf SERELECS

4.1 Konzept

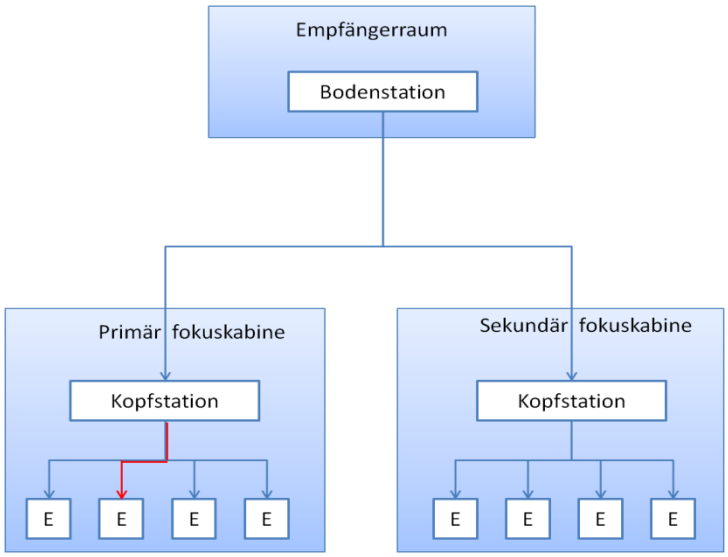


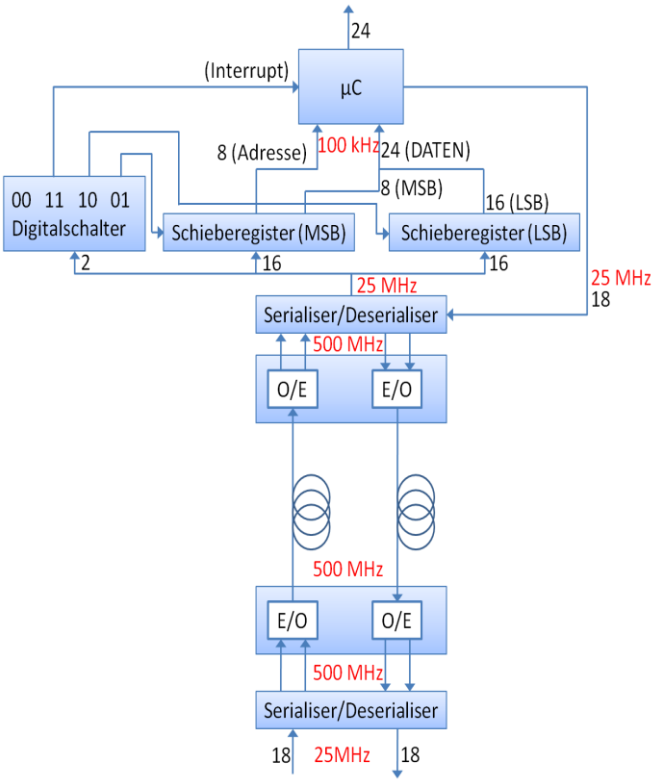
Abbildung 4.1: *Übersicht SERELECS*

Da wie in Kapitel 3 beschrieben kein kommerziell erhältliches System zur Erneuerung der Kommunikationsstrecke vom Empfängerraum zu den einzelnen Empfängern geeignet ist, wurde ein komplett eigenes Kommunikationssystem geplant. Der Name SERELECS setzt sich zusammen aus:

- SE → Serial
- RE → Realtime
- LE → Low Emission
- CS → Communication System

Damit beschreibt der Name SERELECS die Kerneigenschaft, die das zukünftige System auszeichnen soll. Die Daten sollen seriell, in einer fest definierten Zeit übertragen werden und dabei den Empfangsbetrieb im Radioteleskop Effelsberg nicht durch Störstrahlung beeinflussen.

In Abbildung 4.1 ist der Aufbau von SERELECS verdeutlicht. Die Daten sollen im Empfängerraum in einer Baueinheit namens Bodenstation in das Serelecsprotokoll eingebettet werden und über Lichtwellenleiter entweder in die Primär- oder in die Sekundärfokuskabine übertragen werden. In den Kabinen sollen die Daten von der Einheit Kopfstation übernommen werden. Diese soll einen Mikrocontroller beinhalten, der die Daten an die einzelnen Empfänger verteilt. Dazu wird der Nachricht eine Adresse mitgegeben. Dies geschieht ebenfalls über Lichtwellenleiter.
Als Prototyp soll nur der in Abbildung 4.1 durch einen roten Pfeil markierte Teil, von einer Kopfstation zu einem Empfänger, aufgebaut werden. Die Empfängerseite soll mit komplettem Funktionsumfang realisiert werden, sowie sie später im Radioteleskop Effelsberg eingebaut werden soll. Dies beinhaltet das Senden und Empfangen der Zustände der 24 zu steuernden Schalter. Des Weiteren ein Bedienpanel, an dem die Schalter manuell zu Testzwecken geschaltet werden können, sowie die Anzeige aktuellen Zustände mit Hilfe von LEDs. Die Kopfstation hingegen soll nur Teilfunktionen des späteren Komplettsystems beinhalten. So werden die Daten nicht auf verschiedene Empfänger selektiert, sondern alle Daten, die die Kopfstadion verlassen, sind für den einen angeschlossenen Empfänger bestimmt. Infolgedessen ist auch eine Schnittstelle zur Bodenstation hin notwendig.



In Abbildung 4.2 ist der Aufbau der Versuchsstrecke schematisch dargestellt. In schwarzen Zahlen sind die Anzahl der parallelen Leitungen verdeutlicht; steht keine Zahl an einer Leitung, so steht der Pfeil für eine Einzelne.

Die roten Zahlen stehen für die Übertragungsrate der anliegenden Signale an den jeweiligen Leitungen.

Die Probedaten, die benötigt werden um den Aufbau zu testen, sollen mit Hilfe eines FPGA-Evaluationsboardes generiert werden. Außerdem soll das Board auch die Daten, die es generiert hat, mit den anliegenden Daten an bestimmten, fest definierten Orten vergleicht.

Abbildung 4.2: *Teilkonzept SERELECS*

Die von dem FPGA-Board generierten Daten werden über 18 Leitungen parallel auf die TxBussystem-Platine, die die Bodenstation realisiert, übermittelt. Diese Leitungen gehen auf einen Serialiser/Deserialiser Baustein, der die Daten auf zwei Leitungen differentiell (siehe Kapitel 2.1.4) ausgibt. Dadurch erhöht sich die Übertragungsrate von 25 MHz auf 500 MHz. Diese Signale werden dann an eine optische Schnittstelle weitergeleitet.

Über Lichtwellenleiter, zur Reduktion der Störemission, werden die Daten an die RxBussystem-Platine übertragen. Diese Platine realisiert einen Großteil der Funktionalitäten der Empfängerseite. Die fehlenden Funktionen sind lediglich die Anzeige der Schalterzustände und die Schalter zur manuellen Änderung der Zustände. Diese sind auf der RxControlUnit-Platine realisiert.

Auf der RxBussystem-Platine werden die Signale wieder von optisch auf elektrisch gewandelt und differenziell an einen Serialiser/Deserialiser-Baustein übertragen. Dieser parallelisiert die Daten wieder auf 18 Leitungen, so dass die Taktfrequenz der Daten nur noch 25 MHz schnell sind. Da der verwendete Mikrocontroller nur Signale mit einer Frequenz von 100 kHz entgegen nehmen kann, müssen die Datenraten weiter reduziert werden. Dies geschieht über zwei Schieberegister. Diese bringen auch die Daten in die gewünschte Form aus 8 Adressbit und 24 Datenbit. Um nacheinander die Schieberegister und anschließend dem Mikrocontroller den Befehl zum Laden der Daten zu geben, wird ein Digitalschalter verwendet. Von dem Mikrocontroller gehen zum Einen die 24 Datenleitungen zur RxControlUnit und wieder 18 Datenleitungen zum Serialiser/Deserialiser-Baustein, damit die Schalterzustände auch in den Control-Raum (siehe Abbildung 3.1) gemeldet werden können.

4.2 TxBussystem

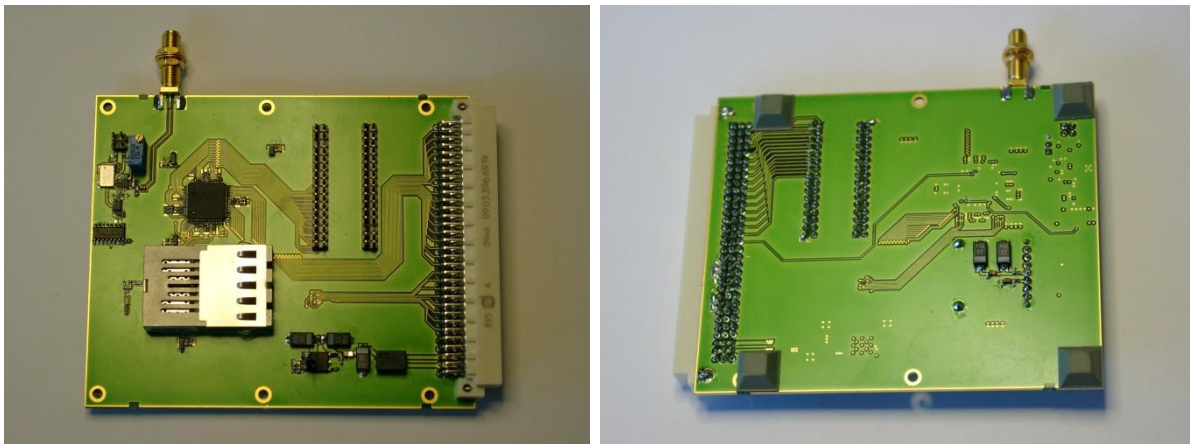


Abbildung 4.3: *Vorder- und Rückseite TxBussystem*

Die in Abbildung 4.3 dargestellte TxBussystem – Platine stellt die Sendeseite des Prototypen von SERELECS dar.
Diese Platine enthält nicht den kompletten Leistungsumfang, die eine spätere Realisierung im Radioteleskop haben wird. So fehlt unter anderem die Steuereinheit, die entscheidet für welchen Empfänger die Daten sind. Des Weiteren hat diese Transceiver – Platine lediglich eine optische Schnittstelle, so dass die angesprochene Verteilung der Daten auf mehrere Empfänger nicht möglich ist.
Die vom FPGA generierten Zufallsdaten (siehe Kapitel 5.4) werden über den 98 poligen Hartingstecker parallel eingespeist. Von dort werden sie direkt an den Serialiser /Deserialiser – Chip übertragen, dieser wandelt die 18 parallelen Datenströme in einen Seriellen um. Hierdurch wird die Taktrate von 25 MHz auf 500 MHz verzwanzigfacht.
Der Baustein zur Serialisierung kann parallele Datenraten zwischen 15 MHz und 66 MHz verarbeiten. Da diese beschriebene Umwandlung sehr taktempfindlich ist und auch die Taktrate während Versuchsmessungen variiert werden soll, wurde neben einem 100 MHz – Quarzoszillator, dessen Takt auf 25 MHz geviertelt wird, aber auch ein SMA - Anschluss implementiert. Von diesem kann ein externes sinusförmiges Taktsignal eingespeist werden, das in ein rechteckförmiges Signal transformiert wird.
Sind die Daten serialisiert, so werden sie an eine optische Schnittstelle weitergeleitet, von wo aus sie über Glasfaserkabel an die Empfängerstation gesendet werden.
SERELECS kann nicht nur Daten an den Empfänger senden, sondern auch von ihm Daten entgegennehmen. Diese Daten kommen an der gleichen zweikanaligen optischen Schnittstelle an und werden seriell, über zwei differentielle Leitungen, an den Serialiser/Deserialiser – Baustein gesendet. Dieser reduziert die Datenrate um den Faktor achtzehn und gibt die Daten parallel aus. Diese Leitungen sind ebenfalls an den Stecker geführt, von dem aus die Daten vom FPGA kommen. Zwischen den Serialiser/Deserialiser – Chip und dem Stecker zum FPGA sind die Leitungen nicht direkt geführt, sondern laufen jeweils auf einen 32 poligen Stecker. Diese können gebrückt werden oder im Zusammenspiel mit der RxBussystem-Platine die Signale an verschieden Position beider Platinen analysiert oder miteinander verglichen werden.

4.2.1 Serialisierung / Deserialisierung

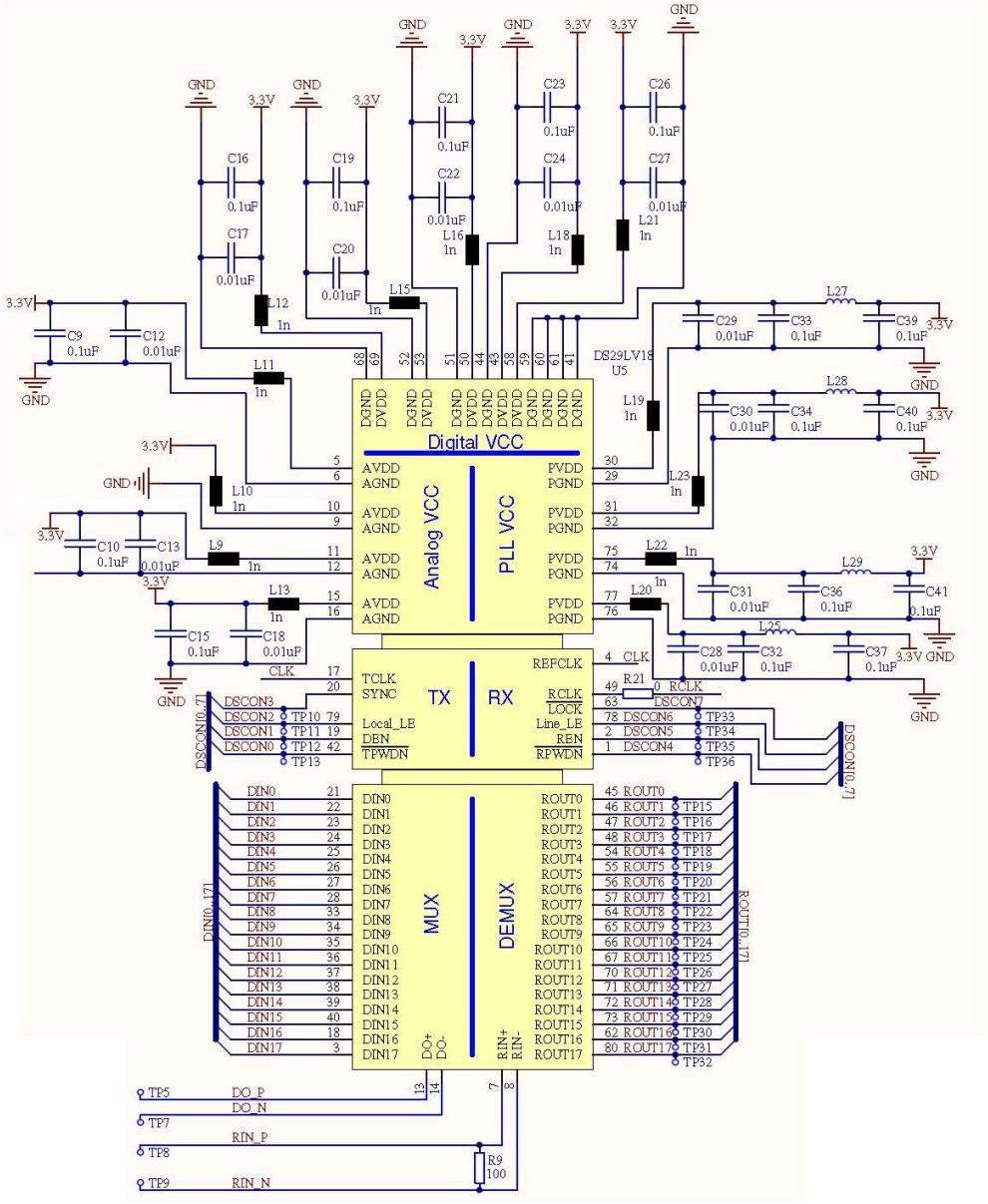


Abbildung 4.4: *Schaltplanteil Serialiser/Deserialiser*

Die Grundfunktion der TxBussystem-Platine ist die Serialisierung der parallelen Daten und anschließende Übertragung über Lichtwellenleiter zur RxBussystem-Platine. Die Wandlung von einem parallelen zu einem seriellen Datenstrom wird mit Hilfe des DS92LV18 „ 18-Bit Bus LVDS Serialser/Deserialiser – 15-66 MHz“ Bausteins (Abbildung 4.4) von National Semiconductor verwirklicht.
Der DS92LV18 kann Daten multiplexen. Dazu gibt er die an den Pins DIN0 – DIN17 anliegenden Signalpegel seriell über die Leitungen DO+ und DO- aus. Diese Leitungen übertragen das Signal differentiell (siehe Kapitel 2.1.4) um die Störsicherheit erheblich zu erhöhen. Dies geschieht wie in Abbildung 4.5 dargestellt synchron zu dem Takt, der am Pin TCLK anliegt. Damit die Gegenseite, die die Daten entgegennimmt, weiß, wo der Übergang von einem zum nächsten Datenwort ist, fügt der Chip zu den achtzehn Datenbit zusätzlich ein Start- und ein Stopbit hinzu.

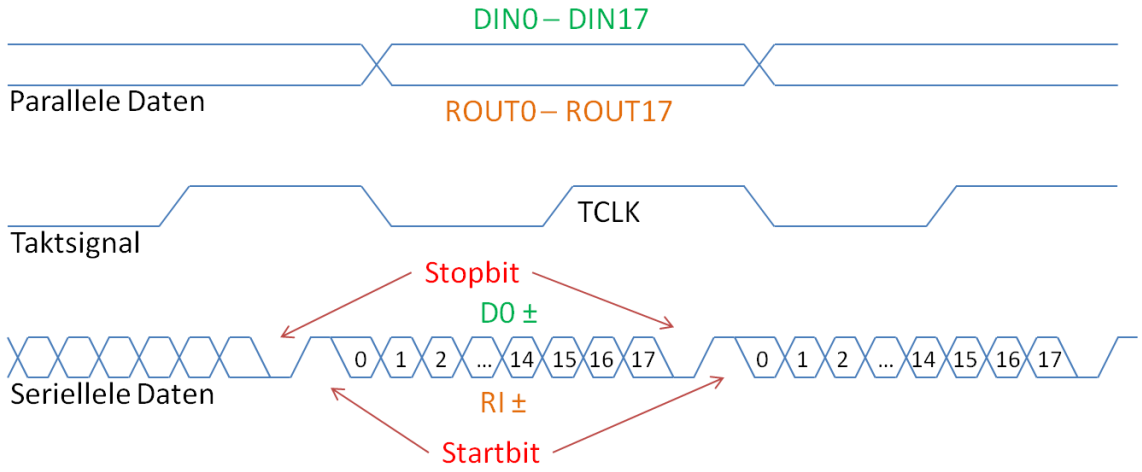


Abbildung 4.5: *Arbeitsprinzip Serialiser/Deserialiser*

Der Serialiser/Deserialiser – Baustein hat auch einen Demultiplexer[1] implementiert. Dieser wandelt den an den Pins RIN+ und RIN- anliegenden seriellen, differentiellen Datenstrom in einen parallelen und legt diesen an die Ausgänge ROUT0 – ROUT17.
Synchronisiert wird auf den Übergang von Startbit zum Stopbit- Es kann zu Problemen kommen, wenn über mehrere Sendezyklen ein solcher Pegelwechsel von Low auf High an einer bestimmten Position bei den Datenbit vorkommt, da der DS92LV18 nicht sicher zuordnen kann, welcher Übergang den Anfang eines neuen Datenwortes ankündigt.
Die Leitungen RIN± werden mit einem Widerstand abgeschlossen, damit es zu keinen Reflexionen auf den Leitungen kommt. Dies würde zu Überlagerungen der Signale führen, so dass es zu Fehlern in der Übertragung der Daten kommt. Bei den anderen beiden seriellen Eingängen DO± ist dies nicht notwendig, da diese Leitungen intern an der Gegenseite der optischen Schnittstelle reflexionsfrei abgeschlossen sind.
Neben den Datenein- und Datenausgängen hat der Baustein eine Reihe von Pins, die der Initialisierung des Bausteins dienen. Dies ist notwendig um gewisse Arbeitseinstellungen vorzunehmen oder bestimmte Status abzufragen.
Der Ausgangspin[2] LOCK zeigt an, ob die Empfänger – PLL synchronisiert ist. Mit dem Eingang SYNC kann gewählt werden, ob der Baustein die anliegenden Daten weiterleiten soll oder ob ein Synchronisationsmuster versendet werden soll. Die Signale REN bzw. DEN schalten die Receiver- bzw. Transmitterausgänge bei Highpegel ab. Das vom Oszillator bereitgestellte Taktsignal wird an die Eingänge TCLK für den Sender und REFCLK für den Empfänger angelegt. Am RCLK wird zu Kontrollzwecken der PLL Takt nochmal ausgegeben.
Die in Abbildung 4.4 im oberen Drittel dargestellten Pins, dienen der Spannungsversorgung des Bausteins.

1: Für den Demuxbetrieb (orange dargestellt) bei Zeichnung unten serieller Eingang und oben paraller Ausgang.
2: Es werden nur die Initialisierungspins erwähnt, die für diese Diplomarbeit wichtig sind.

4.2.2 Signalwandlung optisch auf elektrisch

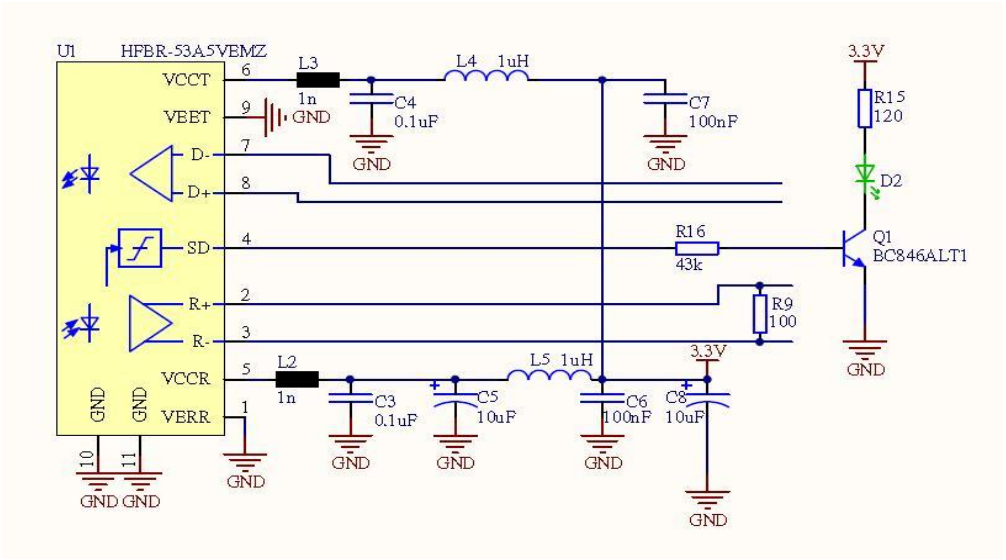


Abbildung 4.6: *Schaltplanteil Signalwandlung*

Das TxBussystem-Board tauscht seriell Daten mit der RxBussystem-Platine aus. Damit diese Kommunikation möglichst wenig Störstrahlung emittiert, wurde als Übertragungsmedium Multimodeglasfaserkabel gewählt (siehe Kapitel 2.1.3).
Als Schnittstelle dient das HFBR-53A5EMZ – Modul der Firma Avago. Diese Schnittstelle kann Signale mit einer Übertragungsrate von bis zu 2 GHz wandeln. Die Daten, die elektrisch, differentiell an den Leitungen D± anliegen, werden in optische Signale gewandelt. Umgekehrt werden die optischen Signale, die am Receiver ankommen, in differentielle Signale gewandelt und an den beiden Pins R± ausgegeben.
Der Ausgang „Signal Detect“, in Abbildung 4.6 mit SD abgekürzt, signalisiert durch einen Pegelwechsel, ob Daten empfangen wurden. Dieses Signal wurde auf eine LED geführt. Da diese maximal von einen Strom von 10 mA durchflossen werden dürfen, mussten die Werte der beiden Widerstände rechnerisch ermittelt werden (Berechnung siehe Kapitel 9.1.2).
Die Spulen, Kondensatoren und Ferrite an den Versorgungspins dienen der Stabilisierung und Störungsunterdrückung der Betriebsspannungen.
Als optisches Übertragungsmedium werden Multimodeglasfaserkabel mit einem Kerndurchmesser von 50 µm und einem Manteldurchmesser von 125 µm mit einem SC-Stecker verwendet.
Der Baustein überwacht intern die Pegel der Signale und regelt diese nach. Dies führt beim Senden von langen Folgen gleicher Pegel dazu, dass er bei vielen Nullen die Leistung hochregelt, so dass diese am Empfänger als Highpegel interpretiert werden und umgekehrt bei langen Einsfolgen die Leistung runterfährt, bis der Empfänger fälschlicherweise eine logische ‚1‘ als logische ‚0‘ wahrnimmt. Diese Fehler können nur durch Kanalcodierung reduziert werden. In diesem Projekt wurden die längeren Folgen gleicher Pegel durch eine 4Bit/5Bit – Codierung unterdrückt. Die vom FPGA generierten Daten werden von diesem durch eine eindeutige Zuweisung codiert und die Empfangsseite muss sie wieder decodieren (siehe Kapitel 5.4.4).

4.2.3 Taktgenerierung

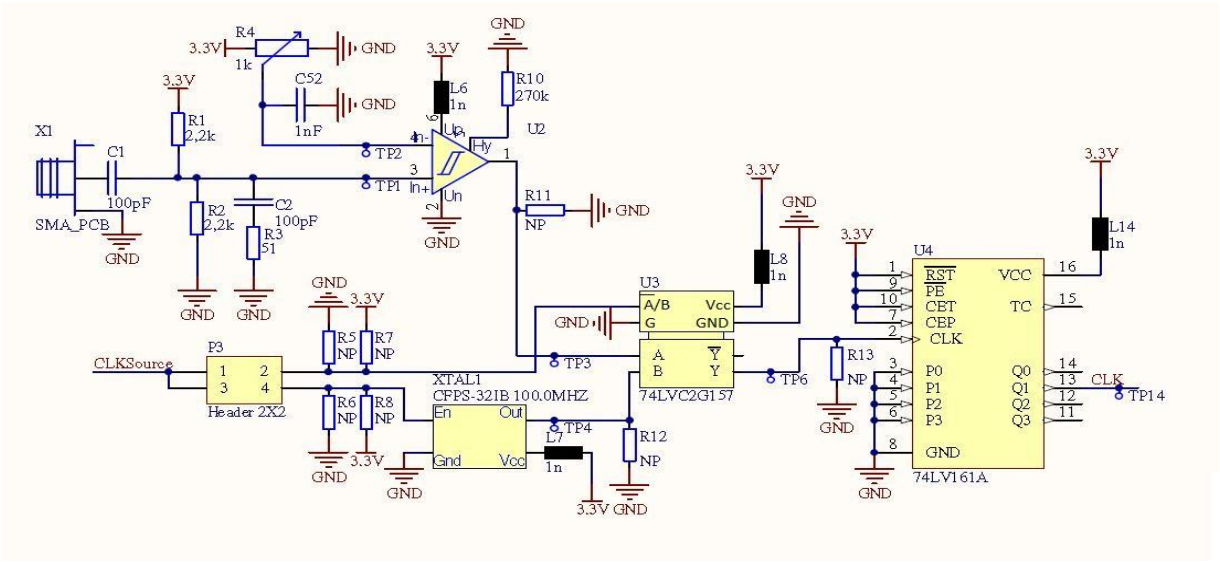


Abbildung 4.7: *Schaltplanteil Taktgenerierung*

Dieser Teil des Schaltplans ist für die Taktgenerierung verantwortlich. Das hier erzeugte Signal wird an den Serialiser/Deserialiser – Baustein übertragen. Die Frequenz dieses Taktsignals regelt, mit welcher Geschwindigkeit der Chip die parallel anliegenden Daten seriell und die seriell anliegenden Daten parallel ausgibt.
Es gibt zwei Möglichkeiten das Taktsignal zu erzeugen. Zum Einen gibt es auf dem Board ein präzisions Quarzoszillator, der ein 100 MHz getaktetes Signal erzeugt. Geschaltet wird der Oszillator mit dem Signal CLKSource, das zum Enabel – Eingang führt. Der Dataselector (74LVC2G157) hat neben den beiden Dateneingängen, Pin A und B einen Select – Eingang mit dem festlegt wird welcher der beiden Eingänge an die Ausgänge, einmal normal und einmal negiert, durchgeschaltet werden soll. Hat das Signal CLKSource einen Highpegel um den Quarzoszillator freizuschalten, so wird automatisch auch das Signal am Dataselector durchgeschaltet. Im umgekehrten Fall, arbeitet der Oszillator nicht und das Signal der externen Takteinspeisung wird durchgeschaltet. Diese beiden Möglichkeiten ein Taktsignal zu erzeugen, hat für die Realisierung zwei Vorteile. Es kann der Takt geändert werden und somit die Übertragungsgeschwindigkeit variieren und bei einer späteren Realisierung im Radioteleskop kann die Übertragung auf den Systemtakt synchronisiert werden, der von allen elektronischen Komponenten genutzt wird. Über den SMA-Stecker wird dieses Signal eingespeist (siehe Rechnung Kapitel 9.1.1), die Widerstände an dieser Leitung (R1 – R3) sorgen für eine Pegelanpassung und werden von dem Comparator von einem sinusförmigen- in ein rechteckförmiges Signal gewandelt. Mit Hilfe des Potentiometers lässt sich die Schaltschwelle verändern. Im Betrieb kann mit einem Oszilloskop der Widerstandswert so verändert werden, dass das Rechtecksignal symmetrisch ist.
Da beide Taktsignale eine Frequenz von 100 MHz haben, aber nur 25 MHz benötigt werden, teilt der 4-Bit-Counter-Baustein (74LV161A) den Takt um den Faktor vier herunter und leitet dieses Signal an den Serialiser/Deserialiser- Chip weiter.

4.2.4 Spannungsversorgung

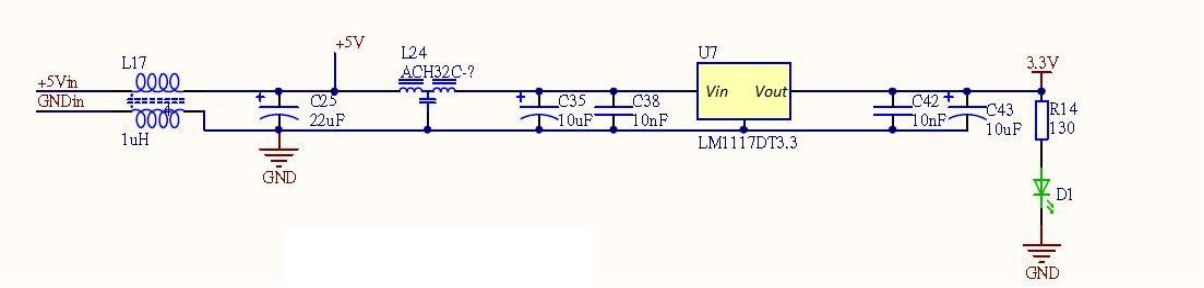
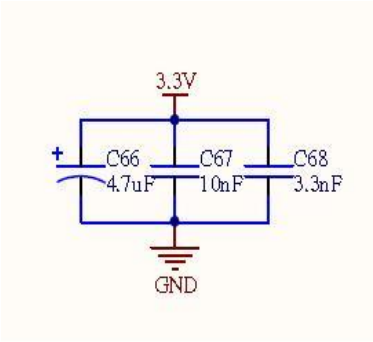


Abbildung 4.8: *Schaltplanteil Spannungsversorgung*

Vor allem im Hochfrequenzbereich ist die Spannungsversorgung ein oft unterschätztes Problem. Ohne entsprechende Vorkehrungen können sehr viele Störungen über die Spannungsversorgung eingespeist und abgestrahlt werden.
Die meisten der verwendeten Bausteine benötigen eine Betriebsspannung von 3,3 V, der Decoder für die Siebensegmentanzeige benötigt hingegen 5 V Betriebsspannung. Dementsprechend wird die höchste benötigte Spannung, also 5 V eingespeist. Der Festspannungsregler LM1117DT3.3 wandelt die eingespeisten 5 V Gleichspannung auf einen festen Pegel von 3,3 V. Somit stehen alle benötigten Betriebsspannungen zur Verfügung.
Die einzelnen Kondensatoren (C25, C35, C38, C42 und C43) dienen als Stützkondensatoren, die durch ihre teilweise unterschiedlichen Kapazitäten im niederen Frequenzbereich Störungen unterdrücken. Da der Oszillator auf dem Board mit 100 MHz getaktet ist, sind in diesem Frequenzbereich stärkere Störungen zu erwarten. Diese werden vom In-Line-Filter (ACH32C) unterdrückt. Dieser LCL-Bandpassfilter hat genau bei 100 MHz eine starke Störunterdrückung.
Zur Gleichtaktunterdrückung wird ein Common-Mode-Filter verwendet. Aufgebaut ist der Filter aus einem Ferrit, um den zwei Leitungen in umgekehrter Richtung gewickelt sind und somit als Spulen mit umgekehrter Polarität wirken. Fließt ein zeitlich veränderlicher Strom auf beiden Leitungen in die gleiche Richtung, so bilden sich um beide Drähte magnetische Felder, die entgegengesetzt wirken. Diese überlagern sich und heben sich teilweise gegenseitig auf. Da das magnetische und elektrische Feld untrennbar miteinander gekoppelt sind, schwächt sich auch der Strom proportional.
Die Leuchtdiode zeigt an, ob die 3,3 V Spannung anliegt.

4.2.5 Vcc-Gnd-System



Auf der Platine sind vier Kondensatorengruppen (*Bezeichnung im Schaltplan: C60 bis C71*) platziert, die das Vcc-Gnd-System bilden.
Jede Gruppe besteht aus drei Kondensatoren, zwei Plattenkondensatoren und einem Elko-Tantal-Kondensator. Wie in Abbildung 4.9 dargestellt, haben die Plattenkondensatoren eine Kapazität von 3,3 nF und 10 nF. Der Tantal-Kondensator mit einer Kapazität von 4,7 µF wurde gewählt, da er im niederen Frequenzbereich bessere Eigenschaft hat und in höheren Kapazitäten erhältlich ist.

Abbildung 4.9: *Vcc-Gnd-System*

Für die Übertragung von Leistung in eine Last sorgt das Vcc-Gnd-System. Es muss sichergestellt sein, dass in einer fest definierten Zeit genügend Ladung zur Verfügung steht, ohne dass es zu einem Spannungseinbruch kommt. Um dies zu gewährleisten, müssen Stützkonstruktionen installiert werden.
In den meisten Fällen wird diese Stützkonstruktion durch Kondensatoren verwirklicht, die möglichst nah an der Last zwischen Betriebsspannung und Masse platziert werden. Damit es zu möglichst geringen Spannungsabfällen an der Konstruktion selbst kommt und die Spannung fast ausschließlich an der Last ansteht, müssen die Kondensatoren möglichst niederohmig sein.

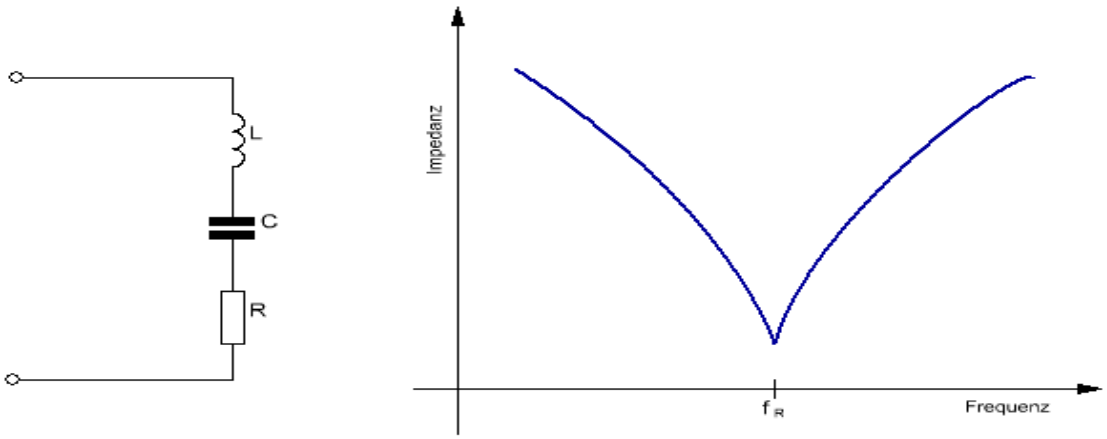


Abbildung 4.10: links: *Ersatzschaltbild-* rechts: *Impedanzverlauf- realer Kondensator*

Ein realer Kondensator hat neben dem kapazitiven auch einen ohmschen und induktiven Anteil. Bei niederfrequenten Signalen überwiegt der kapazitive Anteil. Wird die Frequenz erhöht, so vergrößert sich der Einfluss des induktiven Anteils. Die Frequenz, bei der sich der kapazitive und der induktive Einfluss aufheben, wird Resonanzfrequenz genannt (in Abbildung 4.10: f_R). In diesem Resonanzpunkt, bei dem ein rein ohmsches Verhalten gegeben ist, ist die Impedanz am geringsten.
Da Digitalbausteine, häufig sehr steile Stromimpulse fordern, müsste aber ein Stützsystem breitbandig niederohmig sein. Ist dies nicht gewährleistet, so kann die Spannung an der Last bei steigenden Flanken einbrechen oder bei fallenden Flanken kann es zu Spannungsüberhöhungen kommen.

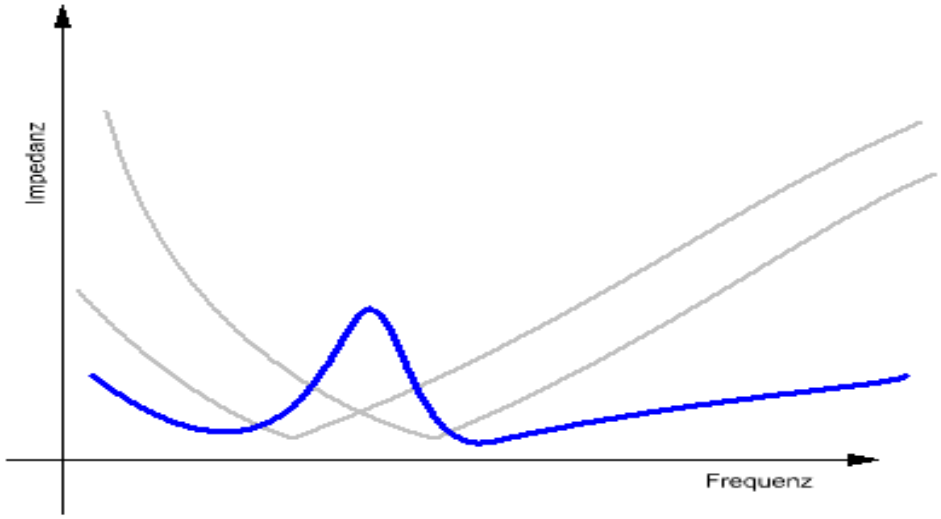


Abbildung 4.11: *Impedanzverlauf Parallelschaltung zweier Kondensatoren*

Durch die parasitären Induktivitäten wird das Vcc-Gnd-System erheblich gestört. Um den Impedanzgang breitbandiger niederohmig zu halten, können mehrere Kondensatoren mit unterschiedlichen Kapazitäten parallel geschaltet werden. In Abbildung 4.11 ist der Impedanzgang einer Parallelschaltung zweier Kondensatoren in blau verdeutlicht. In dem Bereich in dem der eine Kondensator noch kapazitives Verhalten und der Andere schon indukives Verhalten hat, kommt es zu einer Paralellresonanz. Der genaue Verlauf der Impedanz ist abhängig von den Kapazitätswerten und der Güte der gewählten Bausteine.
Bessere Ergebnisse können erzielt werden, wenn drei oder mehr Kondensatoren parallel geschaltet werden und diese möglichst nah an die Last mit breiten Leiterbahnen angebunden werden.

Mit einem flächigen Vcc-Gnd-System und mehren parallelen Kondensatorgruppen lassen sich sehr gute Ergebnisse erzielen.
Mit diesem Verfahren lassen sich bis in den zweistelligen GHz-Bereich niederohmige Systeme realisieren. Das Vcc-Gnd-Sytem kann als leerlaufende Leitung mit kapazitiven Eigenschaften gesehen werden. Um den Wellenwiderstand der Leitungen zu minimieren, werden dünne Substrate verwendet. Empfohlen werden Substratdicken < 120 µm. Wir verwenden Platinen mit einer Substratdicke von 50 µm.

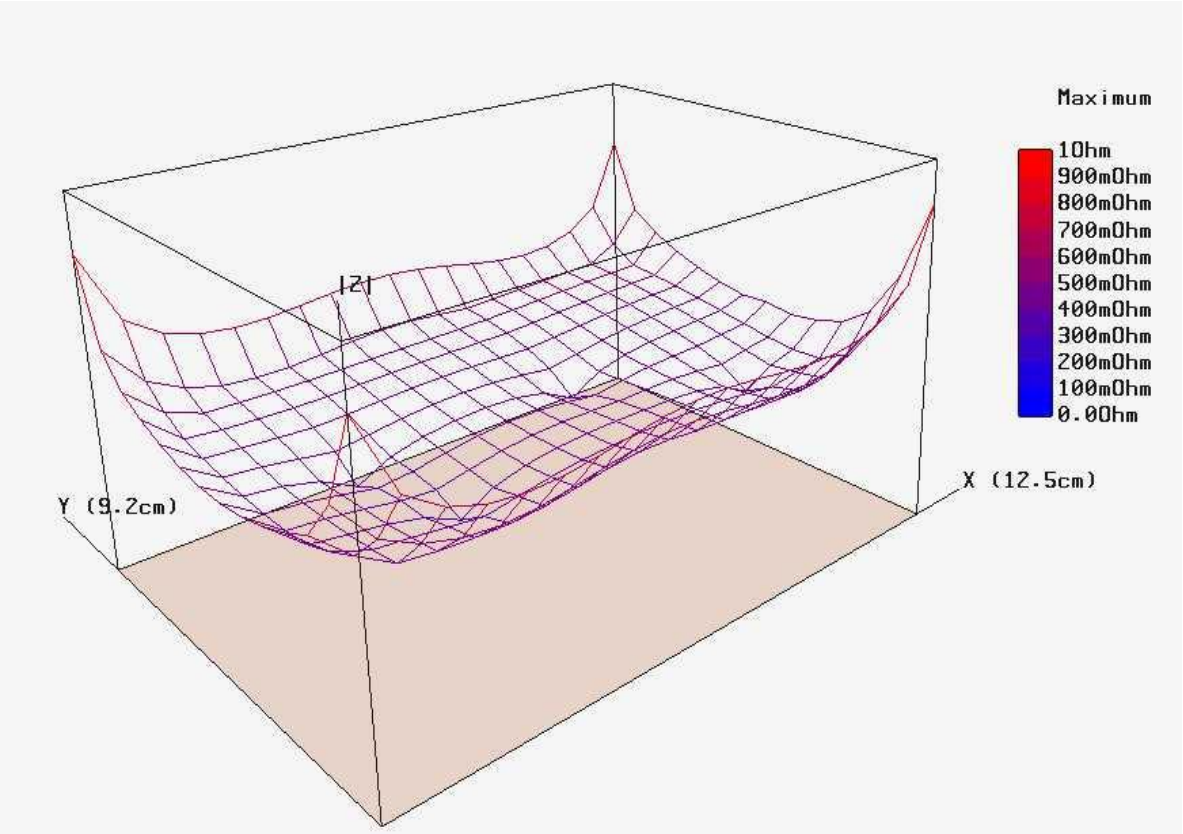


Abbildung 4.12: *Simulation Vcc-Gnd Impedanz*

Da diese Kondensatorgruppen nicht einzeln an jede Last, beispielsweise an jeden Pin eines ICs gesetzt werden, müssen diese eine bestimmte Verteilung auf der Platine haben.
Mit der Software „Silent V4“ wurde die Positionierung dieser Stützkonstrukte simuliert und optimiert. Es wurden (siehe Abbildung 4.12) vier Kondensatorgruppen vorausgesetzt, je einer für die optisch/elektrisch Wandlung, einer für den Serialiser, einer für den Mikrocontroller und einer für die Taktgenerierung. Die Positionen der Gruppen wurden bei der Simulation variiert, bis das Ergebnis zufriedenstellend war.

Störungen, die von außen in das Gesamtsystem eindringen wollen, werden sehr schnell absorbiert. Ein hochfrequenter Störer muss beim Eintritt einen großen Impedanzsprung überwinden. Der geringe Teil, der diesen überwinden kann, trifft auf ein stark verlustbehaftetes System, da mit kleiner werdendem Wellenwiderstand, die Kupferverluste zunehmen. Die restliche Energie wird in Wärmeenergie umgewandelt.

Um das System noch weiter zu verbessern, wurden sämtliche Anbindungen von Bausteinen ans Vcc-Gnd-System mit SMD Ferriten angebunden. Diese haben bei der Betriebsfrequenz ihre Resonanzfrequenz, also ein rein ohmsches Verhalten, so dass Störungen, die über die Vcc - Pins in die Spannungsversorgungsfläche eintreten könnten, vorher stark gedämpft werden.

4.2.6 Steckverbinder TxBussystem

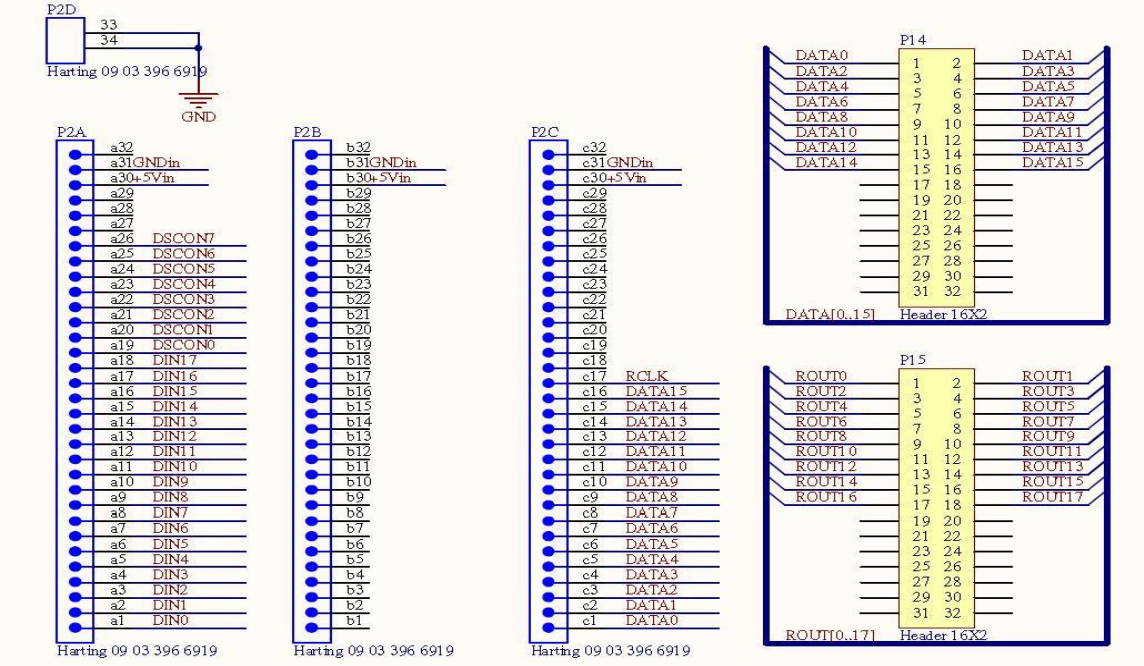


Abbildung 4.13: *Schaltplanteil Steckverbinder*

Um Daten vom FPGA- Board entgegen nehmen zu können und Daten an verschieden Punkten der Übertragungsstrecke vergleichen zu können, wurden Steckverbinder angebracht.
Der 98 polige Harting Steckverbinder bildet die Schnittstelle über ein entsprechendes Kabel zum FPGA – Evaluationsboardes. Übertragen werden:

- **Din0 – Din17:** Datenbus vom FPGA zufällig generiert um die Übertragungsstrecke testen zu können.
- **DATA0 – DATA15:** Daten, die die Übertragungsstrecke durchlaufen haben und zum Spartan 3E Board gesendet werden um die fehlerfreie Übermittlung der Daten zu überprüfen.
- **RCLK:** Ausgangssignal, das von der internen PLL generiert wurde.
- **GNDin:** Überträgt das Groundpotential.
- **+ 5Vin:** Überträgt die 5 V Gleichspannung wahlweise vom FPGA – Board oder über eine extere angeklemmte Spannungsquelle.
- **DSCON0 – DSCON7:** Überträgt die Initialisierungssignale des Serialiserbausteins.

Über die Steckerleisten P 14 und P15 können die Daten vom FPGA auf den Serialiser gebrückt werden. Im Zusammenspiel mit der RxBussystemplatine sind auch andere Konfigurationen möglich, das heißt Signale von anderen Positionen der Übertragungsstrecke brückbar (genaueres siehe Kapitel 4.3.5).

4.3 RxBussystem

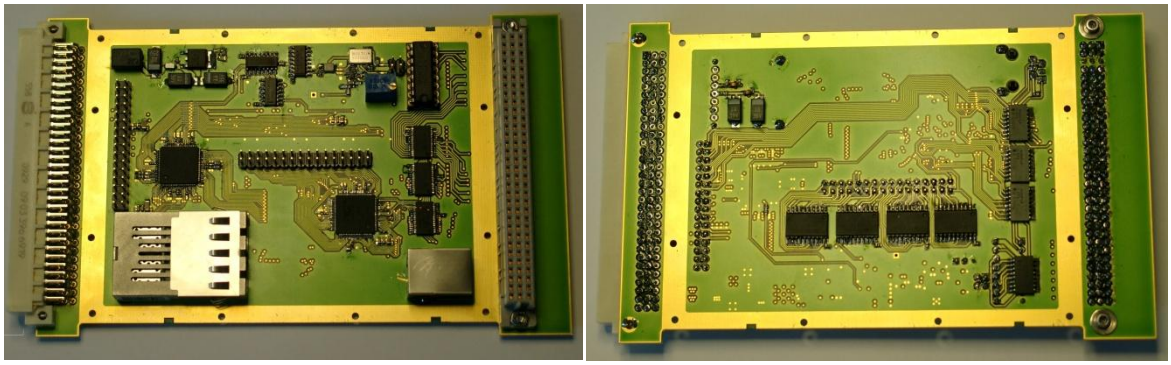


Abbildung 4.14: *Vorder- und Rückseite RxBussystem*

Die Einheit, die die gesendeten Daten von der Kopfstation am Empfänger entgegennimmt, ist das RxBussystem. Die Aufgaben, die dieses Teilsystem erfüllt, können in drei Einzelgebiete aufgeteilt werden:

- Aktuelle Schalterstellungen vom Kontrollraum empfangen und am Empfänger einstellen.
- Aktuelle Zustände der Signale S0 bis S23 an Bedienpanel weiterleiten.
- Signale der Schalter auf dem Bedienpanel empfangen und Signale S0 – S23 aktualisieren und an den Kontrollraum melden.

Alle in Kapitel 4.2 beschriebenen Funktionen des TxBussystems sind identisch in diesem Teil der Kommunikationsstrecke realisiert. Zusätzlich wird die Kommunikation zwischen RxControlUnit und RxBussystem geregelt, ein Mikrocontroller implementiert und die Daten an den Empfänger weitergeleitet. Der Mikrocontroller regelt, beispielsweise die Signale S0 bis S23 und entscheidet welche Signale priorisiert sind, die die vom Kontrollraum gesendet oder die am Bedienpanel eingestellt werden. Kommt es zu einer Zustandsänderung eines der Signale, so müssen alle beteiligten Einheiten, Rechner im Kontrollraum sowie Bedienpanel, dies mitgeteilt bekommen.

4.3.1 Datenratenreduktion

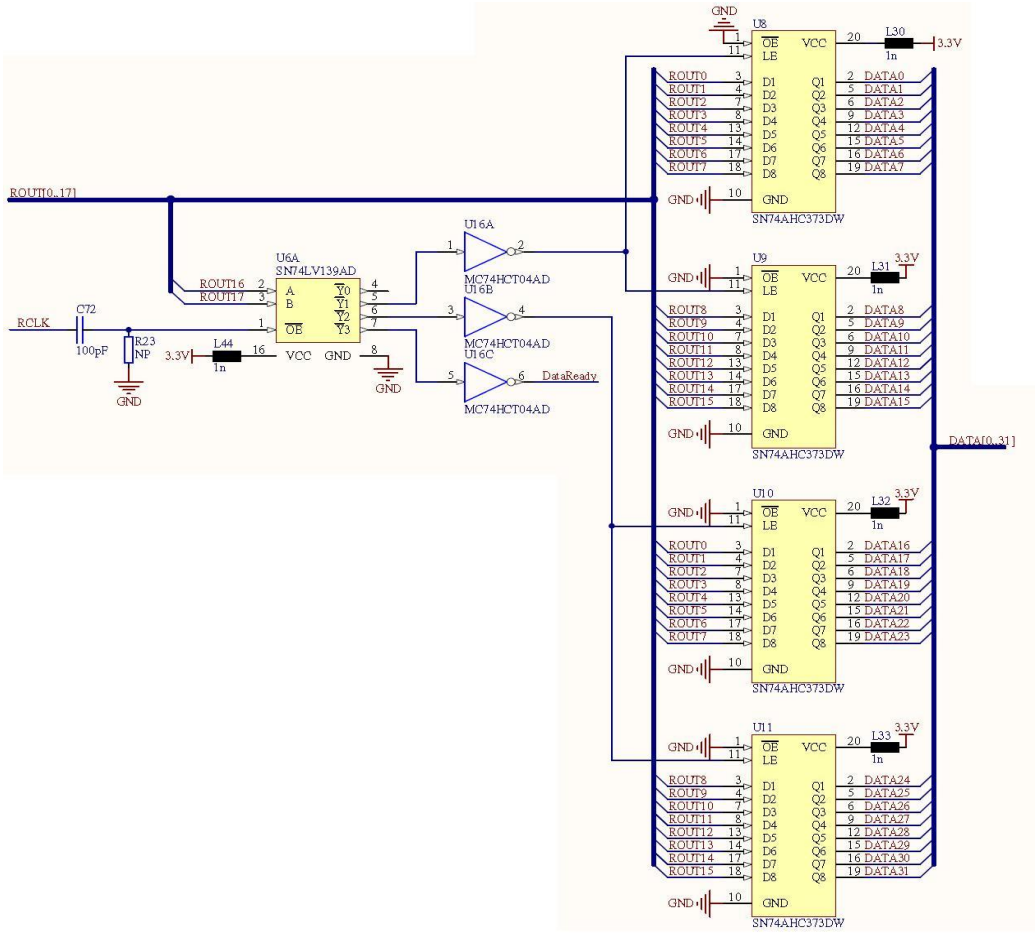


Abbildung 4.15: *Schaltplanteil Datenratenreduktion*

Die an das RxBussystem gesendeten Daten werden, wie in Kapitel 4.2.2 beschrieben, von optischen Signalen in elektrische umgewandelt und anschließend parallelisiert (siehe Kapitel 4.2.1). Da die Daten mit einer Datenrate von 25 MHz anliegen, ist es notwendig diese zu reduzieren, da der verwendete Mikrocontroller in dieser Geschwindigkeit die Daten nicht entgegen nehmen kann.
Damit der Mikrocontroller die Daten einlesen kann, wurde eine Zykluszeit von 5 µs festgelegt. Somit stehen 125 Zyklen mit einer Zykluszeit von je 40 ns zur Verfügung.
Da ein komplettes Datenwort nicht nur aus den übertragenen 16 Datenbit sondern aus 32 Bit besteht, müssen die Daten zwischengespeichert werden und anschließend zu einem kompletten Datenwort zusammengefügt werden. Da von den 18 Bit die pro Zyklus übertragen werden, 16 Datenbit sind, bleiben zwei Bit für eine interne Adressierung.
Ein kompletter Übertragungszyklus besteht aus folgenden Phasen:

- Übertragung der ersten 16 Datenbit, als Adresse wird eine 01 gesendet. Die Daten werden in die oberen beiden Schieberegister geschrieben.
- 6 Zyklen wird die Adresse auf 00 gesetzt, und somit kein Speicher oder Mikrocontroller angesprochen.
- 16 Datenbit werden übertragen und als Adressbit 10 gesendet. Die Daten werden in die beiden anderen Schieberegister geschrieben.

- Erneut 7 Zykuszeiten wird die Adresse 00 übertragen, die Datenbit werden nicht gespeichert.

- Anschließend einmalig Adresse auf 11 setzen, dies gibt dem Mikrocontroller den Interrupt, dass die Daten an den Schieberegistern anliegen und geladen werden können.

- Die restlichen Sendezyklen wird wieder die Adresse 00 übertragen. Nach 125 Zyklen ist ein Gesamtsendezyklus abgeschlossen, indem ein komplettes 32 Bit Wort in 5 µs vom Mikrocontroller entgegengenommen wurde.

Die beschrieben Zuweisungen von Adressen erfolgt über die Signale ROUT16 und ROUT17, die vom Serialiser – Chip kommen. Der „Dual 2 Line 4 Line Decoder“ - Baustein SN74LV139 gibt je nach Zustand der beiden Eingangsbit an einem der vier Ausgangsbit einen Lowpegel aus. Da sich mit zwei Eingangsbit vier mögliche Zustandspaare bilden lassen, ist die Zuweisung eindeutig und auch vollständig; es gibt keine Eingangskombination die keinem Ausgang zugewiesen ist.
Da sowohl der Mikrocontroller als auch die vier Schieberegister highaktiv sind, werden die Signale aus dem „Dual 2 Line 4 Line Decoder“ – Baustein mit Hilfe des Digitalbausteins MC74HCT04AD negiert. Dabei wird aus einem Highpegel ein Lowpegel und umgekehrt.

4.3.2 Mikrocontroller

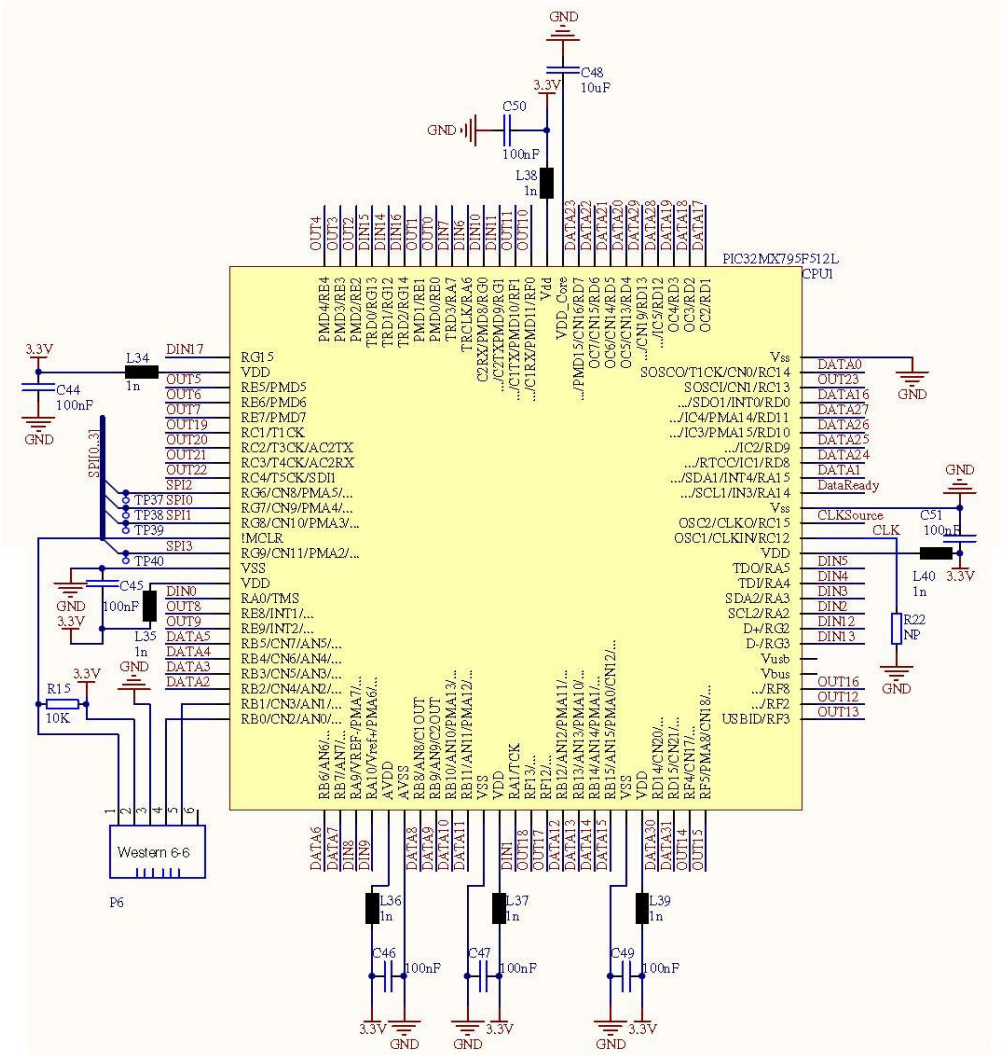


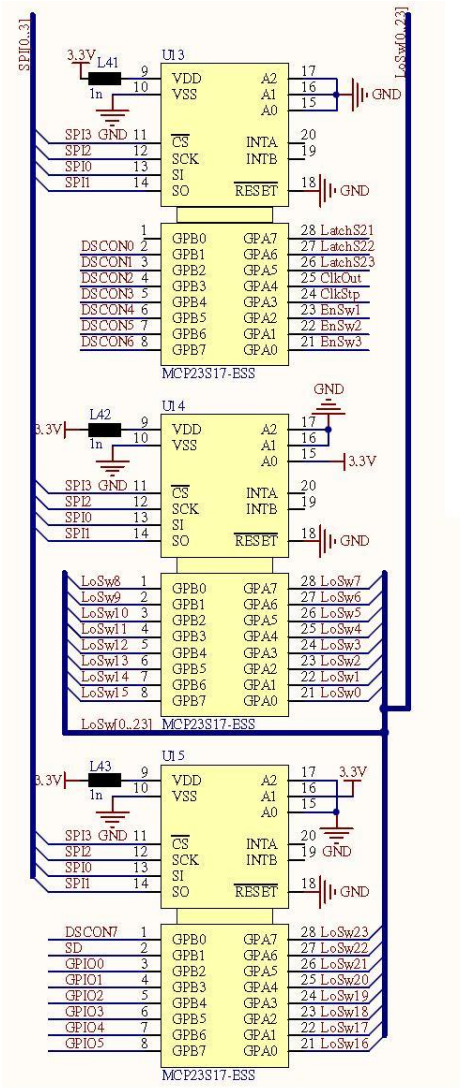
Abbildung 4.16: *Schaltplanteil Mikrocontroller*

Der verwendete Mikrocontroller PIC32MX795F512 übernimmt auf der RxBussystem – Platine die Datenkonditionierung und steuert das gesamte System. Er codiert die ausgehenden und decodiert die ankommenden Daten mit der 4Bit/5Bit-Codierung (siehe Kapiel 5.4.4). Er sendet die zeitunkritischen Daten über den SPI-Bus an das RxControlUnit-System und hat, ähnlich wie das FPGA, eine Funktion, die die gesendeten mit empfangenen Daten vergleicht.

Der in C programmierte Quellcode wird vom Compiler in Maschinencode übersetzt und über den 6 poligen Westernstecker in den Mikrocontroller geladen.

Die Programmierung des Mikrocontrollers ist nicht Teil dieser Diplomarbeit und ist deshalb auch nicht ausführlich beschrieben.
An dieser Stelle möchte ich mich bei Dipl.-Ing. (FH) Thomas Berenz bedanken, der diesen Teil des Projektes übernommen hat.

4.3.3 SPI – Bus



Hauptaufgabe von SERELECS ist die Übertragung der 24 Schalterstellungen zu den einzelnen Empfängern. Es ist aber auch möglich die Schalter bei jedem Empfänger manuell zu verstellen. Damit dies auch den Rechnern im Kontrollraum übermittelt wird, müssen ständig die aktuellen Positionen der Schalter an die entsprechenden Rechner übermittelt werden. Da der verwendete Mikrocontroller allerdings nicht ausreichend Ein- und Ausgangspins hat, werden die zeitunkritischen Daten über den „Serial Peripheral Interface Bus“ (kurz SPI - Bus) übertragen. Der SPI – Bus ist ein serieller synchroner Datenbus, der für die lokale Anbindung von Digitalbausteinen an einen Mikrocontroller konzipiert ist.

Dieser Bus arbeitet nach dem Master – Slave Verfahren, bei dem eine zentrale Rechnereinheit, bei diesem Projekt der Mikrocontroller, die Kommunikation mit allen anderen Teilnehmer, den Slaves, regelt.

Bei den verwendeten MCP23S17-ESS Bausteinen erfolgt eine feste 3 Bit Adressierung über die Eingänge A0-A2.

Soll ein Slave beispielsweise die Adresse „010“ haben, so muss am Pin A1 ein Potential von 3,3 V und an den anderen beiden Groundpotential anliegen.

Abbildung 4.17: *Schaltplanteil SPI - Bus*

Der Bus besteht aus insgesamt vier Leitungen:

- **Serial Data Out (kurz: SO):** Über diese Leitung werden Daten vom Slavebaustein an den Master gesendet.
- **Serial Data In (kurz: SI):** Daten, die vom Master an den Slavebaustein , sowie Adressanfragen gehen über diesen Bus.
- **Serial Clock Input (kurz: SCK):** Um die Kommunikation zu synchronisieren, müssen alle über den SPI-Bus kommunizierenden Teilnehmer den gleichen Takt anliegen haben. Dieser wird vom Master erzeugt und über diese Leitung an alle Slaves übermittelt.
- **Chip Select (kurz: CS):** Um zu unterscheiden, ob die ankommende Nachricht vom Masterteilnehmer eine Adresse beinhaltet oder Informationsdaten, wird diese Leitung während der Übertragung einer Adresse auf Lowpegel gezogen. Alle folgenden Daten sind nun für den entsprechenden Slave, bis eine neue Adressierung erfolgt. Alle Slaves, bei denen die gesendete Adresse nicht mit der eigenen

übereinstimmt, schalten alle Eingänge auf hochohmig, so dass die Signale nicht gelesen werde.

4.3.4 Siebensegment und Stecker zur RxControlUnit

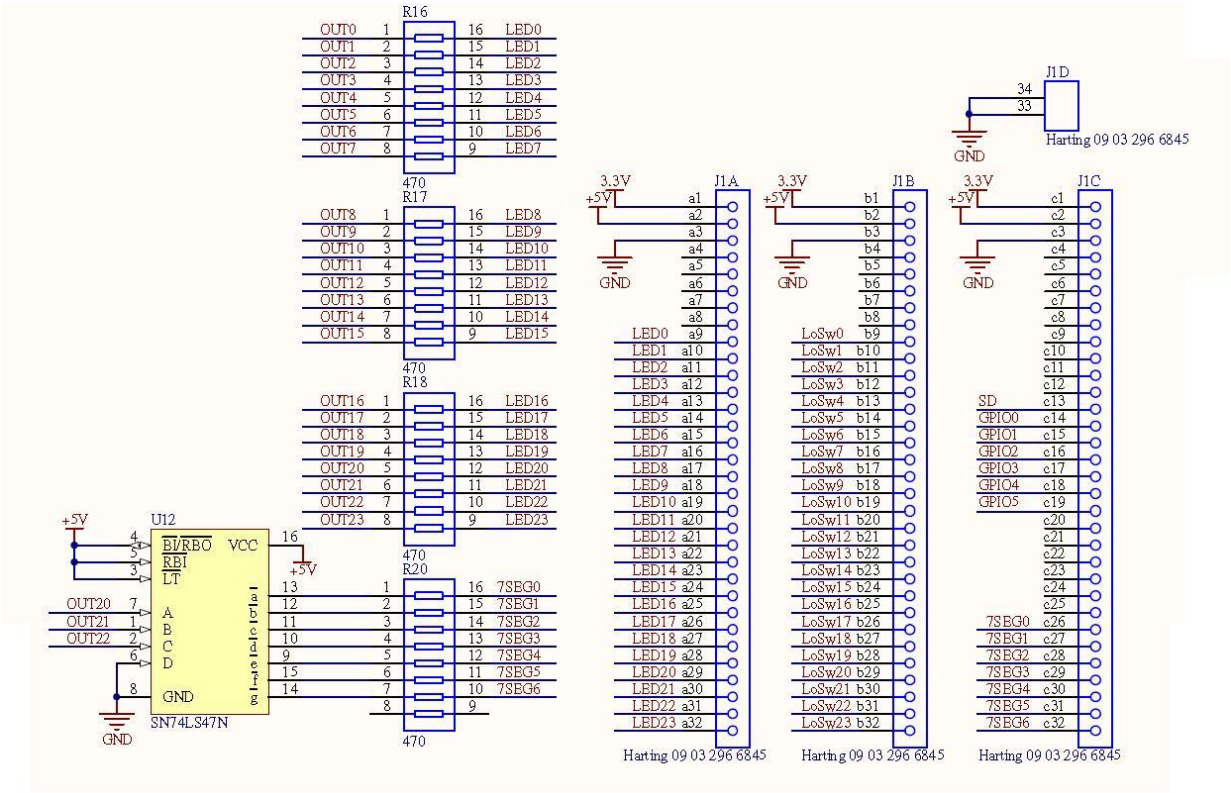


Abbildung 4.18: *Schaltplanteil 7-Segmentanzeige und Stecker*

Die übertragenen 24 Signale sind Steuersignale für Empfänger im Radioteleskop, die vom Kontrollraum an die einzelnen Empfänger gesendet werden. Allerdings muss auch eine Möglichkeit geschaffen werden, manuell die Schalter zu setzen. Dafür wird je ein Bedienpanel an jedem Empfänger angebracht, die RxControlUnit (siehe Kapitel 4.4).
Die Schnittstelle zwischen RxBussystem und RxControlUnit ist der Hartingstecker. Über ihn werden die 24 aktuellen Zustände der Signale S0 bis S24 zur Darstellung an die RxControlUnit übertragen. Diese Signale kommen direkt vom Mikrocontroller. Da es 24 Kippschalter auf dem Bedienpanel gibt, werden diese Signale von der ControlUnit an den Mikrocontroller gesendet, um sie den Signalen S0 bis S 23 zuzuordnen. Zusätzlich werden noch 6 GPIO Signale an den Stecker angelegt, über die verschiedene Status angezeigt werden können, die im Moment allerdings noch nicht verwendet werden und somit für zukünftige Erweiterungen der Empfänger zur Verfügung stehen. Das Signal „Signal Detect (SD)“ vom Serialiser-Chip, das anzeigt, wenn der Baustein gelockt ist, ist auch an den Stecker gelegt. Die Schalterzustände S20 bis S22 (im Schaltplan mit OUT bezeichnet), zeigen binär an welcher Empfänger in einer Mehrempfängebox ausgewählt wurde. Der Baustein SN74LS47N wandelt diese drei Eingangssignale in sieben Ausgangssignale, so dass diese auf einer Siebensegmentanzeige die Empfängerwahl anzeigen.

4.3.5 Steckverbinder RxBusystem

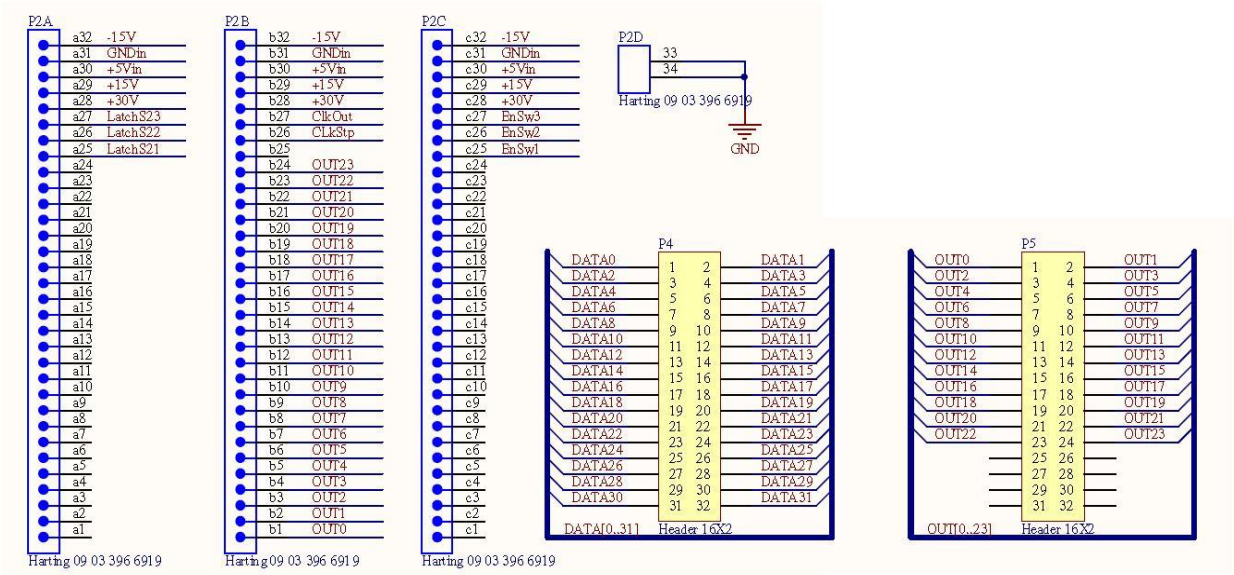


Abbildung 4.19: Schaltplanteil *Steckverbinder*

Der Stecker P2 stellt die Schnittstelle zwischen dem Kommunikationssystem SERELECS und dem Empfänger dar.

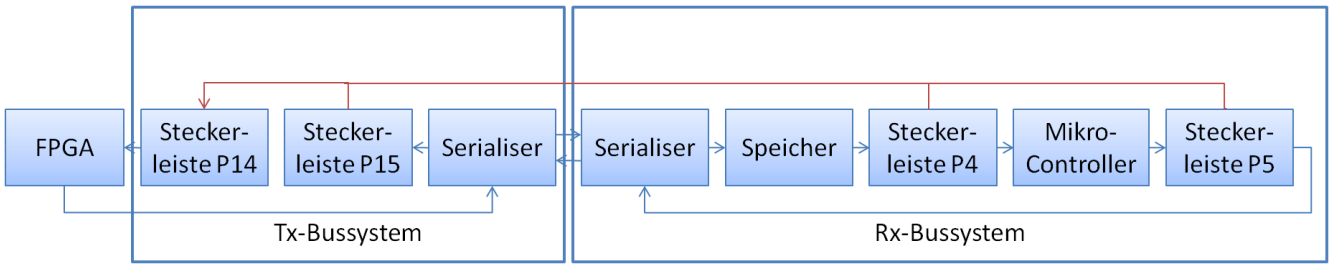


Abbildung 4.20: *Signallaufplan SERELECS*

In Abbildung 4.20 ist ein stark vereinfachter Signallaufplan des gesamten Prototypen von SERELECS dargestellt. Die mit blauen Pfeilen dargestellten Signalpfade sind fest durch Leiterbahnen bzw. zwischen den Serialiser-Chips mit Lichtwellenleiter verbunden. Für die Übertragung von dem Empfänger zurück zur Kopfstation, also von der Rx-Bussystem- zur Tx-Bussystem – Platine, ist es möglich die Signale für den Vergleich im FPGA mit den gesendeten Daten an drei verschiedenen Position abzugreifen. Einmal direkt hinter dem Mikrocontroller, ohne dass der komplette Rückweg über die Lichtwellenleiter durchlaufen werden muss, einmal bereits vor dem Mikrocontroller um zu testen, ob die gesendeten Daten richtig am FPGA ankommen oder die „kurze“ Brücke zwischen Steckerleiste P14 und P15 um den kompletten Signalweg zu testen.

4.4 RxControlUnit

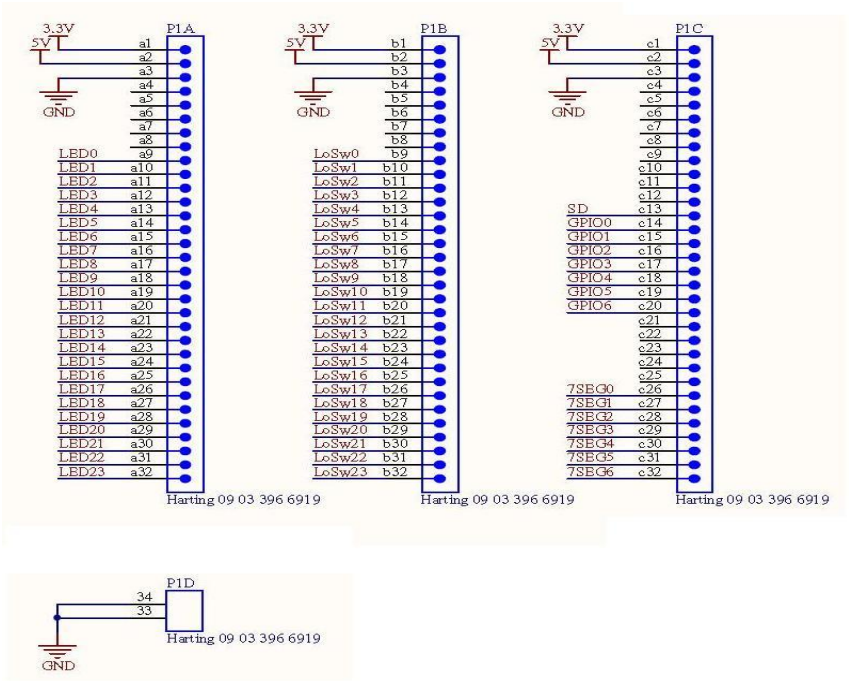


Abbildung 4.21: *Schaltplanteil Schnittstelle zum RxBussystem*

Das RxControlUnit - Teilsystem ist eine Erweiterung zum RxBussystem. Es hat zwei grundlegende Funktionen. Zum Einen sind 24 Schalter angebracht, mit denen die Zustände der Signale S0 – S23 manipuliert werden können. Jeder Schalter negiert bei Betätigung den aktuellen Zustand eines der Signale. Dies wird über den Hartingstecker der Schnittstelle zum RxBussytem an den Mikrocontroller übermittelt. Dieser sendet den neuen aktuellen Signalwert in den Kontrollraum.
Da der normale Messbetrieb des Radiotelekops allerdings aus dem Kontrollraum geregelt wird und die Schalter am Bedinpanel nur bei Wartungsarbeiten und bei der Inbetriebnahme eines Empfängers genutzt werden, müssen die aktuellen Signalpegel der 24 Steuersignale aktualisiert werden um an den in Abbildung 4.22 dargestellten Dioden D1 bis D24 angezeigt zu werden.
Die Signale für die Siebensegmentanzeige werden von dem Hartingstecker direkt auf die Anzeige übertragen.
Für die GPIO Signale 0 bis 3 sind ebenfalls Schalter vorgesehen, diese Signale haben noch keine Belegung und sind für spätere Erweiterungen vorgesehen.

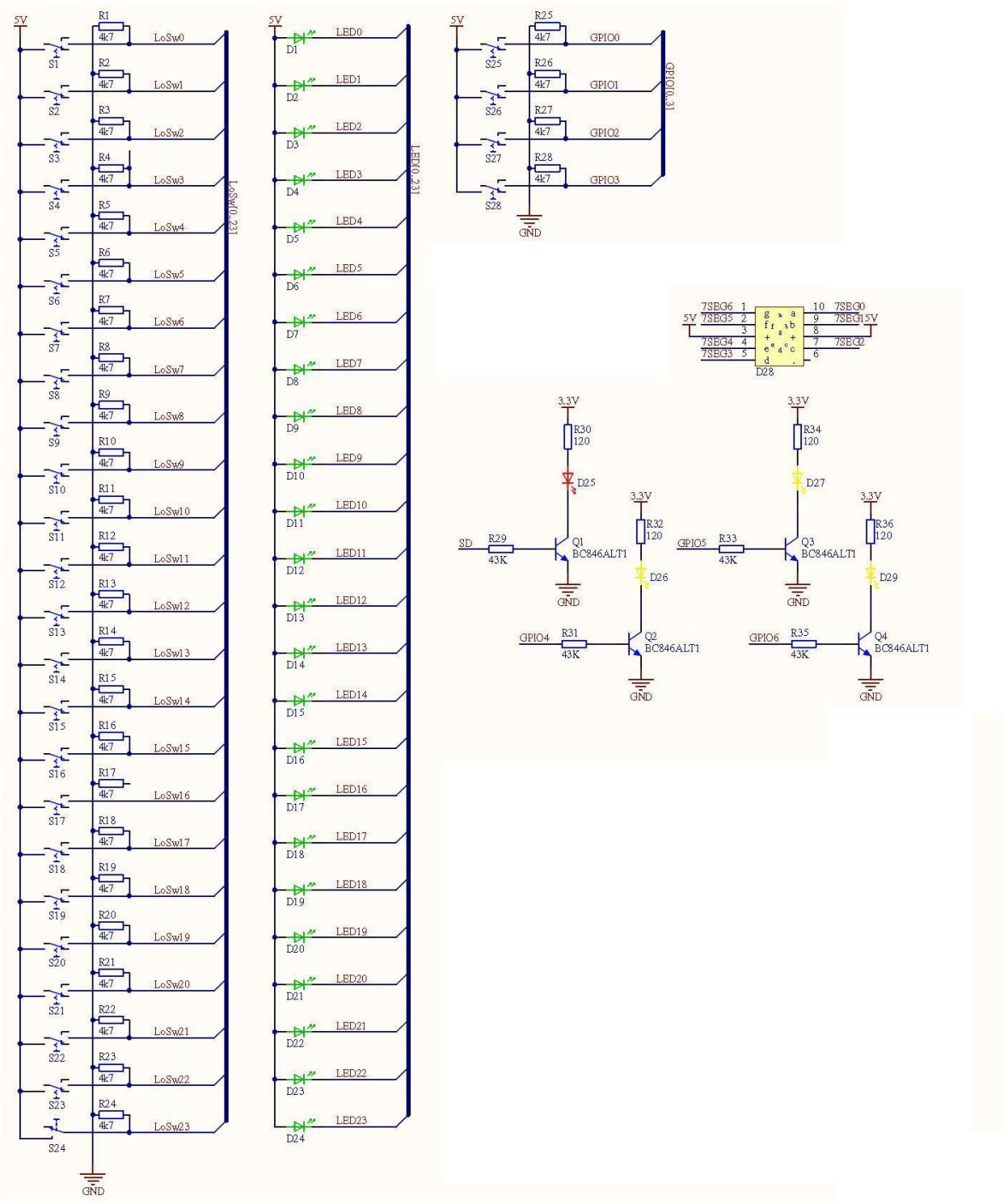


Abbildung 4.22: *Schaltplanteil Statusanzeigen*

Die anderen GPIO Signale (GPIO4 bis GPIO6), die ebenfalls noch keine Verwendung im Moment haben, sind auf drei gelbe LEDs gelegt. Die rote LED zeigt an, ob der Serialiser/Deserialiser- Chip gelockt ist.

5 Testdatengenerierung und Vergleich

Um die Funktionalitäten der beiden Platinen TxBussystem und RxBussystem zu testen, sowie die Kommunikation zwischen ihnen, ist es notwendig Zufallsdaten zu generieren, die übertragen werden können.
Diese Datenpakte werden in einem FPGA auf dem Evaluationsboard „Spartan 3E“ der Firma Xilinx erzeugt.
Zusätzlich werden die Daten nach der Übertragung wieder ins FPGA zurückgelesen um sie mit den gesendeten Daten zu vergleichen. So kann die Zuverlässigkeit des Gesamtsystems überprüft werden. Die Ergebnisse dieses Vergleichs werden an einem LC-Display angezeigt.

Da wie in Abschnitt 4.2.2 beschrieben, eine Kanalcodierung notwendig ist, werden die Daten mit einer 4Bit/5Bit – Zuweisung codiert. Somit muss von dem ursprünglichen Aufbau eines Datenwortes abgewichen werden, da aus den 12 Datenbit pro Wort nun 15 werden. Der vorgesehene Adressbereich ist somit nicht mehr realisierbar. Dies ist allerdings nicht weiter schlimm, da es aktuell im Radioteleskop keine Baugruppen gibt, die diese benötigen und diese auch nur als zusätzliches Feature gedacht waren.

A	B	0	Datenwort 0	Datenwort 1	Datenwort2
1	1	1	5	5	5

Abbildung 5.1: *Aufbau Datenpaket SERELECS*

Ein Datenpaket sendet, wie in Abbildung 5.1[1] dargestellt drei 4Bit/5Bit- codierte Datenwörter, die aufgrund dieser Zuweisung allerdings nur 12 Bit Information enthalten. Mit den oberen beiden Bit, A und B, werden die Schieberegister auf der RxBussystemplatine angesprochen (siehe Kapitel 4.3.1). Zwischen diesen Steuerbit und den Datenbit, liegt das Null - Bit. Dieses wird immer mit einem Lowpegel übertragen. Dieses Bit kann bei späteren Ausführungen des Systems kontrolliert werden. Hat es einen Highpegel, so liegt ein Übertragungsfehler vor. Es könnte auch als Paritätsbit verwendet werden. Dabei wird das Bit so belegt, dass das Gesamtdatenwort entweder eine gerade oder eine ungerade Anzahl an Highpegeln aufweist.

Ein Sendezyklus besteht aus drei dieser Datenpakete. Um die 24 Steuersignale zu verschicken, muss ein bestimmtes Zeitverhalten eingehalten werden. Mit den ersten beiden Datenpaketen werden die Schieberegister beschrieben und mit dem Dritten der Interrupt verschickt, dass die Daten bereit sind und vom Mikrocontroller entgegengenommen werden können. Da der Mikrocontroller dies nicht schnell genug kann, können die Daten nicht direkt hintereinander gesendet werden, sondern es müssen Wartezeiten zwischen den Paketen liegen.
Die Daten werden mit einer Datenrate von 25 MHz parallel vom FPGA ausgegeben, was einer Bitdauer von 40 ns entspricht. Festgelegt wurde eine Gesamtzykluszeit von 5 µs. Somit stehen pro Gesamtzyklus 125 Einzelzyklen zur Verfügung.

1: untere Beschriftung in Abb. 5.1 gibt Größe in Bit an

5.1 FPGA

Die Zufallsdaten werden im Zuge dieses Projektes mit Hilfe eines „Field Programmable Gate Array“ (kurz: FPGA) generiert. Ein FPGA ist ein integrierter Schaltkreis zur Realisierung digitaler Schaltungen.
Es ermöglicht eine komplexe Schaltung zu verändern, ohne eine physikalische Veränderung der Platine vornehmen zu müssen.
Ein FPGA hat folgende Bestandteile (vgl. [URB]):

- **Ein- und Ausgangsblöcke:** diese Blöcke verbinden die Logikelemente mit den Pins des Bausteins. Zusätzlich werden die Ausgangsspannungspegel bestimmten Standards, wie TTL, angepasst.
- **Logikelemente:** bestehen aus Logikgattern mit nachgeschaltetem Flipflops. Die Gatter sind in den meisten Fällen Lookup – Table (kurz LUT). Diese LUTs haben vier oder sechs Eingänge und zwischen einem und sechs Ausgängen, auf die sie jede beliebige kombinatorische Funktion abbilden können.
- **Pfade:** werden auch Kanäle genannt. Sie dienen zur Verbindung verschiedener LUTs, so dass Signale über den gesamten Chip geleitet werden können. Die Pfade sind programmierbar und lassen sich immer wieder neu routen.
- **Taktgenerierung:** zur Synchronisation aller Prozesse und Signalen generiert ein FPGA einen internen Takt, der sich ganzzahlig runter teilen lässt.
- **Speicherblöcke:** sind festverdrahtet integriert und nicht durch Speicherschaltungen aus Flipflops verwirklicht. Diese Speicher dienen ausschließlich zur temporären Speicherung verschiedener Signalpegel.
- **System on Chip:** sind wenige Standardfunktionen, die allerdings häufig gebraucht werden. Sie sind als platzsparende Hard-Cores fest implementiert und können nicht verändert werden. Dies können beispielsweise Ansteuerungen für Schnittstellen sein.

Nachteilig bei der Nutzung des FPGAs auf dem Spartan 3 E Evaluationsboard ist, dass die Programmierung nur temporär ist und bei Abschaltung der Spannungsversorgung verloren geht.

5.2 VHDL

Die Hardwarebeschreibungssprache „Very High Speed Integrated Circuit Hardware Description Language“ (kurz: VHDL) ist ein weltweit anerkannter Standard zur Beschreibung funktionalen Simulation und Datenaustausch beim Entwurf digitaler Schaltungen.
Der Funktionsumfang wurde erweitert durch:

- die Synthese von Hardware
- die Modellierung von Hardware und Zellbibliotheken
- die Einführung mathematischer Typen und Funktionen

5.2.1 Programmaufbau

Ein einzelnes VHDL-Programm kann aus mehreren Unterprogrammen, sogenannten Components, zusammengesetzt sein. Die Kommunikation zwischen den einzelnen Unterprogrammen wird im TOP-Component geregelt. Hier wird festgelegt, welche Signale extern sind, also nach außen hin sichtbar sind oder welche Signale nur intern genutzt werden. Zusätzlich wird im TOP-Component definiert, welche Unterprogramme es gibt, welche Ein- und Ausgänge diese haben und von welchem Typ die Signale sind (siehe Kapitel 5.2.2).

```
library IEEE;
use …;


entity <Entityname> is
 port (<Deklaration der Ein- und Ausgänge>);
end <Entityname>;

architecture <Architekurname> of <Enityname> is
<Architekturdeklaration>;
begin
 process (<Empfindlichkeitsliste>)
 <Prozessdeklaration>
 begin
 {<VHDL-Anweisungen>;}
 end process;
end <Architekurname>;
```

Code 5.1: *Programmaufbau Übersicht*

Jeder Programmteil, ob TOP-Component oder ein Unterprogramm, ist wie in Code 5.1 dargestellt, gegliedert:

- **Verwendete Bibliotheken:** In diesem ersten Block wird festgelegt, welche Bibliotheken (engl.: libraries) eingebunden werden sollen. Während dieses Projektes wurde ausschließlich die IEEE-Bibliothek verwendet. Die use-Anweisung spezifiziert, welches der in der Bibliothek abgelegten Packages eingebunden werden soll.

- **Entity:** In der Entity werden jeweils die Schnittstellen beschrieben, die außerhalb des Objektes sichtbar sind. Somit wird die Anzahl der Anschlüsse festgelegt. Es wird in der Entity des TOP-Components die Schnittstellen des Gesamtprogramms (siehe Kapitel 5.4.2) beschrieben und die der anderen Components untereinander. Diese sind intern und somit bei FPGAs nicht auf einen PIN plazierbar.
- **Architecture:** beschreibt die Funktionalität des Codes, also das Innenleben des Components.
- **Process:** Mit Hilfe der Prozesse lassen sich die programmierten Codes besser strukturieren. Eine Architektur kann mehrere Prozesse enthalten. Zu beachten ist, dass alle Prozesse parallel abgearbeitet werden, aber dass ein Prozess für sich sequenziell abgearbeitet wird. Also zuerst die oberen Anweisungen dann Zeile für Zeile nach unten. Während der Prozessausführung ist es nicht möglich auf die aktuellen Werte zuzugreifen. Erst wenn der Prozess beendet ist, werden die aktuellen Werte übergeben. Alle Zwischenwerte, die im Laufe der Abarbeitung vorkamen sind somit verloren. In der Empfindlichkeitsliste, auch sensitivity list, werden alle Signale aufgeführt, die mit einem positiven Pegelwechsel den Prozess aktivieren.

5.2.2 Signaltypen

Bei der Programmierung von VHDL-Code sind eine Vielzahl von Signaltypen zu unterscheiden. Hier werden nur die im Projekt verwendeten Typen von Signalen vorgestellt. Signaleigenschaften lassen sich in drei große Gruppen aufteilen. Und jedes verwendete Signal ist eine Kombination aus allen drei Eigenschaften:

- **Verfügbarkeit:** Diese Eigenschaft beschreibt, wo das Signal gelesen und verwendet werden kann.
    - Ist das Signal in der Entity des Top-Components deklariert, so ist das Signal nach außen sichtbar und kann mit jedem Component verbunden werden. Wird das Signal in einem Component verändert, muss dieses auch in der Entity des Components stehen.
    - Wenn das Signal in der Architekturdeklaration des Top-Components definiert ist, dann ist es nur innerhalb des Programms sichtbar. Auch diese Signale müssen in jeder Entity der Unterprogramme stehen, wenn es dort manipuliert wird.
    - Soll ein Signal ausschließlich innerhalb eines Unterprogrammes nutzbar sein, so wird es in der Architekturdeklaration des betreffenden Components definiert. Es ist möglich lokale Signale mit gleichem Namen in mehreren Unterprogrammen zu erzeugen.
    - Da ein Unterprogramm aus mehreren Prozessen bestehen kann, ist es möglich auch Signale zu deklarieren, die nur für den Prozess nutzbar sind. Diese werden in der Prozessdeklaration erzeugt.
- **Portmodus:** Der Portmodus gibt an, ob das Signal nur gelesen werden kann oder ob es verändert werden darf.
    - **in – Modus:** Diese Signale sind Eingangssignale und können innerhalb des Componets, in dem sie als in-Modus-Signal definiert wurden, nur gelesen werden und der aktuelle Wert zu Berechnungen genutzt werden. Es ist nicht möglich diesem Signal andere Werte zuzuweisen.
    - **out – Modus:** Signale vom Modus out sind Ausgangssignale und können nicht als Teil einer Funktion anderen Signalen zugewiesen werden.
    - **inout – Modus:** Die als inout deklarierten Signale sind bidirektionale Signale und vereinen die Eigenschaften von Ein- und Ausgangssignalen. Sie können sowohl verändert werden, als auch mittels einer beliebigen Funktion andere Signale beeinflussen.
- **Signalart:** Diese Eigenschaft beschreibt, welche Werte das Signal annehmen kann und welche Form es hat. Signale unterschiedlicher Signalart lassen sich nicht miteinander verknüpfen.
    - **Integer:** Signale vom Typ Integer können alle Dezimalzahlen im Bereich ± 2147483647 als Wert annehmen.
    - **STD_LOGIC:** STD_LOGIC-Signale sind binär. Dies bedeutet, dass sie nur die logischen Werte „0“ oder „1“ annehmen können.
    - **STD_LOGIC_VECTOR:** diese Signalart beschreibt einen binären Bus. Die Breite des Busses, also die Anzahl der Einzelsignale lässt sich frei konfigurieren. Bei der Deklaration wird festgelegt welches das „Most significant Bit“ und welches das „Last significant Bit“ ist. Es ist möglich jedes Bit einzeln zu lesen oder zu schreiben, ohne dass die anderen Bit des Busses beachtet werden müssen.

5.2.3 Verwendete Konstrukte

```
if <Bedingung1> then
 { <Sequentielle Anweisung1> }
   elsif <Bedingung2> then
   { <Sequentielle Anweisung2> }
    ….
    else
     { <Sequentielle Anweisung n> }
end if;
```

Code 5.2: *if-then-else Anweisung*

```
case <Kontrollausdruck> is
 when <Kontrollausdruck1> => { <Sequentielle Anweisung1> }
 when <Kontrollausdruck2> => { <Sequentielle Anweisung2> }
 …
 when others => { <Sequentielle Anweisung n> }
end case;
```

Code 5.3: *case Anweisung*

Bei der Programmierung des VHDL-Codes wurden zwei Konstrukte sehr häufig genutzt. Es ist zum Verständnis des Quellcodes wichtig die Unterschiede des in Code 5.2 dargestellten if-then-else Anweisung und der case-Anweisung (Code 5.3) zu kennen.

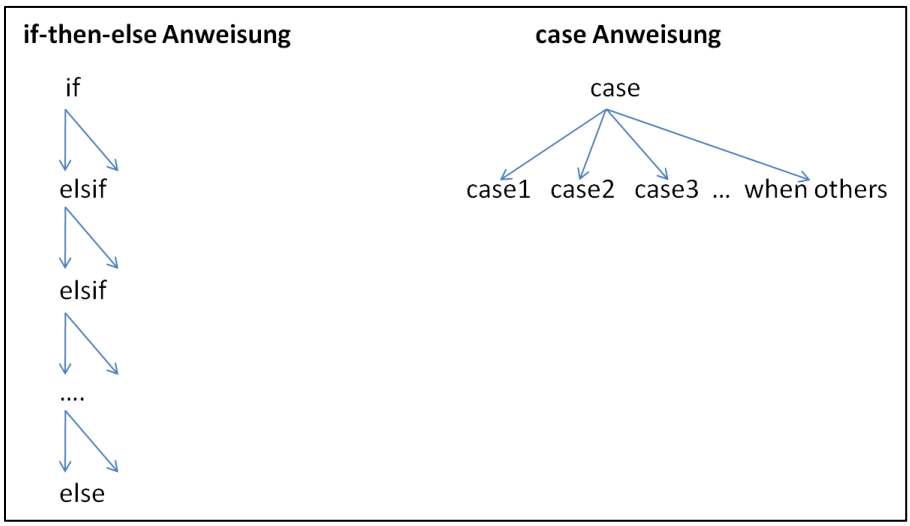


Abbildung 5.2: *Unterschied If- und Case-Anweisung*

Bei der case-Anweisung wird (vgl. [REI]) das Signal, das als Kontrollausdruck eingetragen wurde, gelesen. Abhängig von dem aktuellen Wert dieses Signals zweigt das Programm in die entsprechende Anweisung. Es ist wichtig, dass die Testausdrücke eindeutig sind und dass alle möglichen Werte berücksichtigt werden. Um dies sicherzustellen, ist die „when others“ hilfreich, da sie alle vorher nicht genannten Kombinationen abdeckt.
Die if-Anweisung (cgl. [ZAI]) dahingegen ist eine Verkettung von Bedingungen, die von oben nach unten durchlaufen werden. Die Anzahl der elsif-Pfade ist beliebig, kann also auch Null sein und somit kann es zu erheblich unterschiedliche Signallaufzeiten kommen. Zu beachten ist, dass die Signalparameter erst nach dem Prozess aktualisiert werden, so dass die Veränderung eines Signalpegels bei einer if-Anweisung erst beim nächsten Aufruf des Prozesses wirksam ist und nicht schon während der Anweisung.

5.3 Evaluationsboard „Spartan 3E“

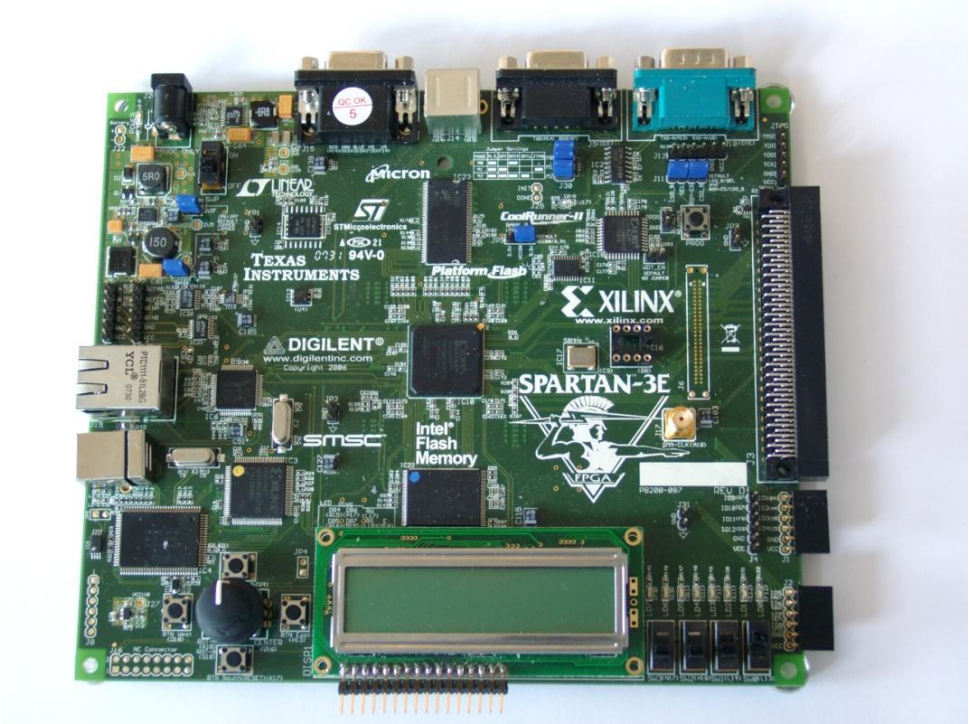


Abbildung 5.3: *Spartan 3E*

Zur Generierung der Zufallszahlen und zum Vergleich von gesendeten Daten und empfangenen Daten, wird das Evaluationsboard Spartan 3E (Abbildung 5.3) der Firma Xilinx genutzt.
Es wird von einem 50 MHz Quarzoszillator getaktet und hat unter anderem folgende Elemente:

- **Speicher:** 16 Mbit SPI Flash, 128 Mbit paralleles Flash, 64 MByte DDR SDRAM

- **Anschlüsse:** Ethernet, JTAG USB Kabel, zwei 9-pin RS-232 Ports, PS/2, Drehinkrementgeber mit Tastfunktion, vier Schiebeschalter, acht LED Ausgänge, vier Taster, 100-Pin Hirose Erweiterungsstecker und drei 6-Pin Stiftleisten

- **Anzeige:** 2 Zeilen mit je 16 Zeichen LCD

5.4 Programmierter VHDL-Code

Zu jedem Unterkapitel im Abschnitt „Programmierter VHDL-Code“ existiert im Anhang ein gleichnamiges Kapitel, in dem der komplette VHDL Quelltext dargestellt ist.

5.4.1 Top

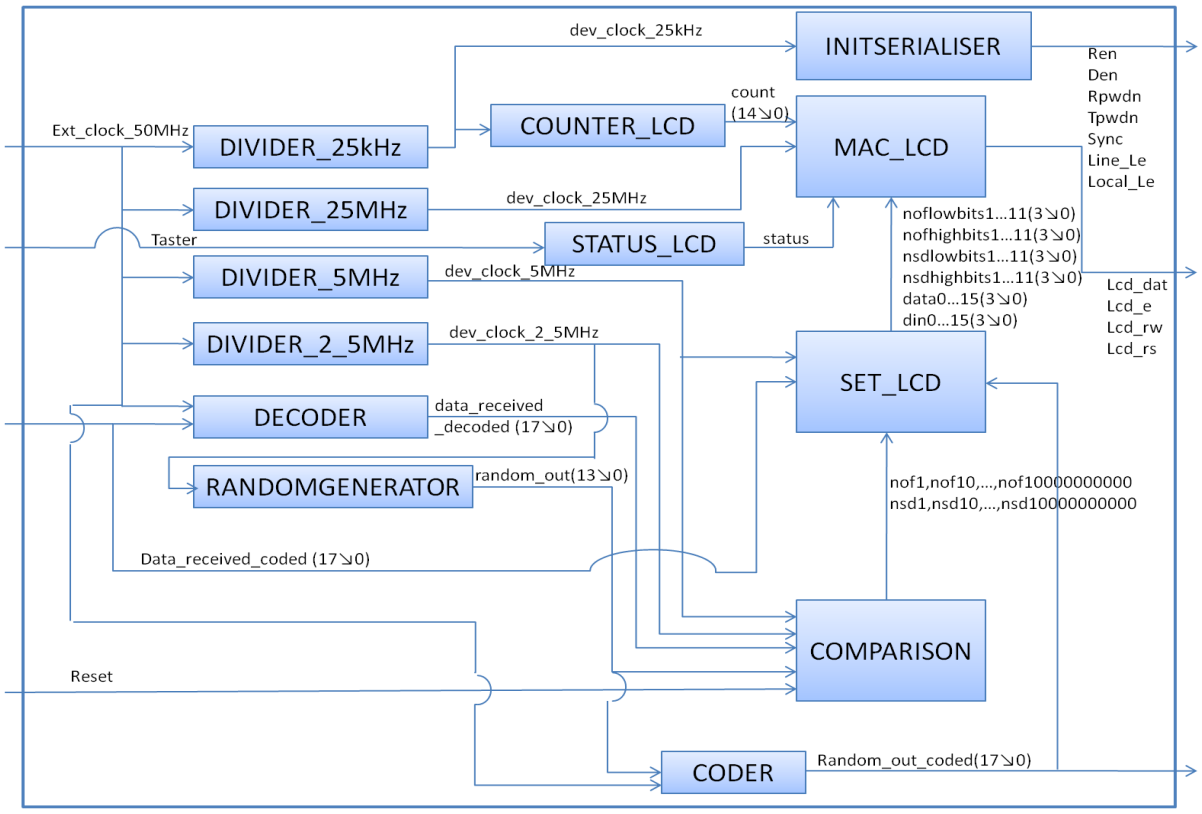


Abbildung 5.4: *Übersicht VHDL-Code*

Um das komplexe Verhalten, das das FPGA realisieren soll, übersichtlicher zu programmieren und um Arbeitschritte zu Gruppen zusammenzufassen, wurde der VHDL-Code in verschiedene Komponenten (engl.: components) aufgeteilt. Diese sind in Abbildung 5.4 mit blauen Kästen dargestellt. So ist es möglich bei bestimmten definierten Ereignissen ein bestimmtes Unterprogramm auszuführen. Jeder Pfeil zwischen den einzelnen Components steht für Signale[1] oder auch Signalgruppen, die in Pfeilrichtung kommunizieren. Die Signale, die nicht nur innerhalb des Programms existieren, sondern auf Ein- oder Ausgangspins des FPGAs geführt sind, haben einen Großbuchstaben die restlichen Kleinbuchstaben. Ist ein Signal komplett in Kleinbuchstaben aufgeführt, so ist es ein internes Signal.
In der Entitydeklaration des Top – Components sind die externen Signale definiert und in der Architekturdeklaration die internen.
Es folgen, die wie in Code 5.4 beispielhaft für die Komponente “CODER” dargestellten, Definitionen jedes einzelnen Components. Es wird neben dem Namen des Unterprogramms auch die Ein- und Ausgangssignale sowie deren Typ festgelegt.

1: Ist Abbildung 5.4 ein Signal „Signalname (Zahl1 ↘ Zahl2)“ dargestellt, so ist die Breite des Bussignales (Zahl1 downto Zahl2) verdeutlicht

```
COMPONENT CODER is
 PORT(
      Ext_clk_50MHz           : in STD_LOGIC;
      random_out              : in STD_LOGIC_VECTOR (13 downto 0);
      random_out_coded        : inout STD_LOGIC_VECTOR (17 downto 0)
 );
end COMPONENT;
```

Code 5.4: *Deklaration Component*

Nach der Deklaration aller Komponenten folgt die Instanziierung (Code 5.5). Dabei wird jedem Port des Unterprogramms ein Signal zugewiesen.

```
INST_CODER : CODER PORT MAP
      (
      Ext_clk_50MHz           => Ext_clk_50MHz,
      random_out              => random_out,
      random_out_coded        => Din
      );
```

Code 5.5: *Signalzuweisung*

5.4.2 Divider

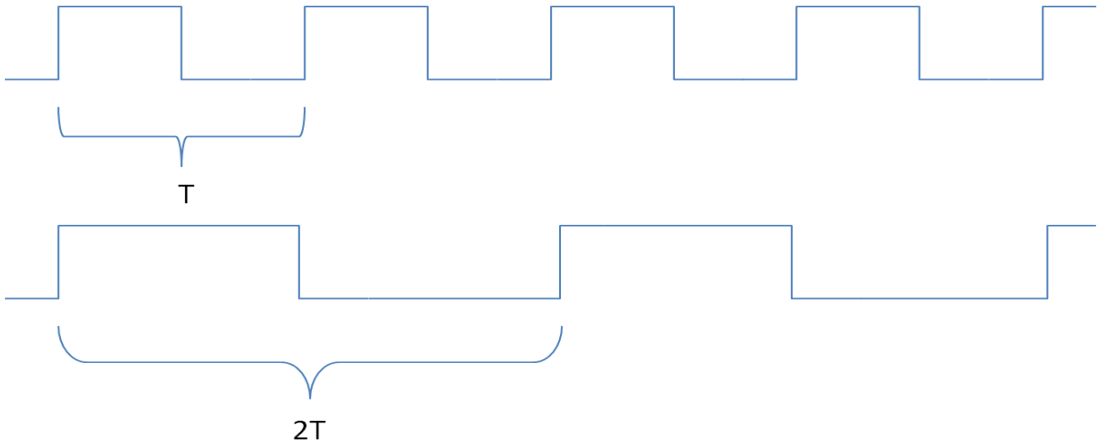
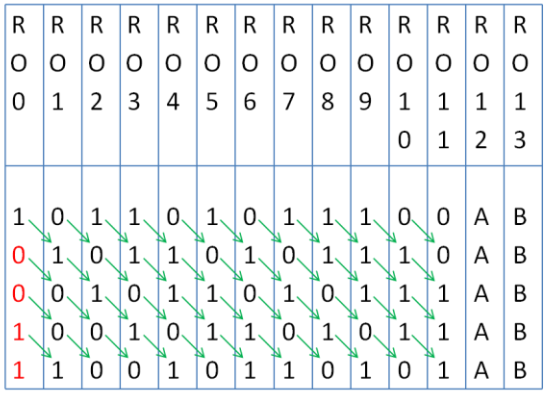


Abbildung 5.5: *Taktteilung*

Die insgesamt vier Unterprogramme Divider, teilen alle den vom FPGA – Board bereitgestellten 50 MHz Takt um einen ganzzahligen Faktor. In Abbildung 5.5 ist dies schematisch verdeutlicht.
Zu beachten ist, dass die verwendete if – Anweisung nur auf steigende Flanken des Eingangssignals reagiert. Dies hat zur Folge, dass wenn man den Takt durch 2n teilen will, die Abfrage nur bis n zählen darf.

5.4.3 Randomgenerator



Der Randomgenerator erzeugt die Zufallsdaten, die zum Testen der Übertragungsstrecke versendet werden.
Das Signal ist 14 Bit breit. Die unteren 12 sind Datenbit und die oberen Beiden, Bit 13 und 14, dienen der internen Adressierung um die Schieberegister und den Mikrocontroller anzusprechen (genaue Beschreibung: siehe Kapitel 4.3.1).

Abbildung 5.6: *Übersicht Signalgenerierung Randomgenerator*

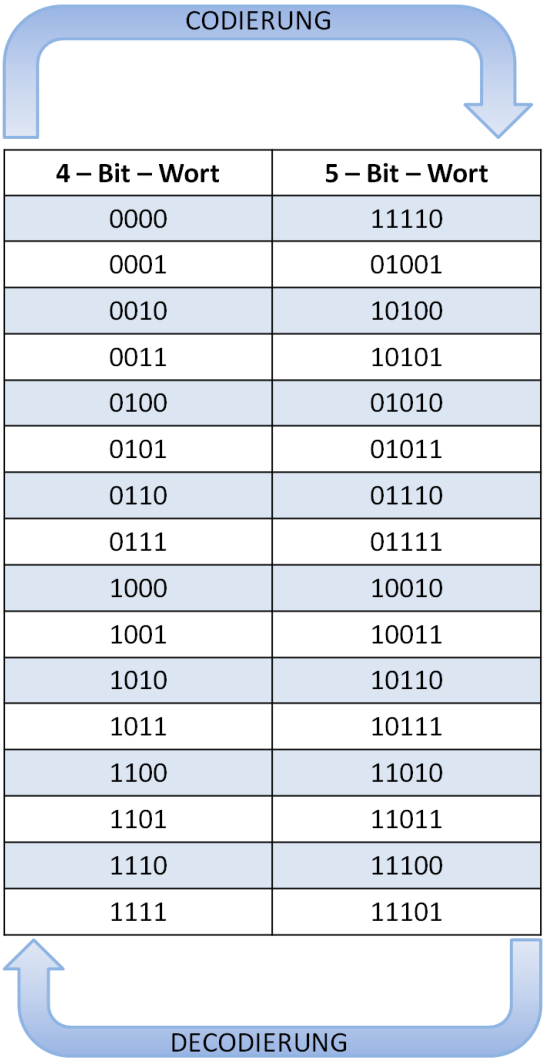
Das Programm erzeugt keine wirklichen Zufallszahlen, sondern Pseudozufallszahlen. Diese sind bei jedem Durchlauf anders, lassen sich aber genau rekonstruieren. Wie in Abbildung 5.6 dargestellt, wird bei jedem Durchlauf jedes Bit eine Stelle weiter zum nächst höheren Bit geschoben. Das Bit random_out11 wird aus dem digital erzeugten Schieberegister verdrängt.

```
if rising_edge (dev_clock_25MHz) then
 interncounter <= interncounter + 1; -- 125 Zyklen für Übertragungszeit 5us
 random(11 downto 1) <= random(10 downto 0);
 random(0)   <=   not(random(0))   XOR   random(8)   XOR   random(2)   XOR
(not(random(7))) XOR random(4);
 case interncounter is
  when "0000000" => random_out(13 downto 12) <= "01";
                    random_out(11 downto 0) <= random(11 downto 0);
  when "0000100" => random_out(13 downto 12) <= "10";
                    random_out(11 downto 0) <= random(11 downto 0);
  when "0001000" => random_out(13 downto 12) <= "11";
  when "1111101" => interncounter <= "0000000";
  when others => random_out <= "00000001111111";
 end case;
end if;
```

Code 5.6: *Randomgenerator (Teilcode)*

Das random_out0 – Bit wird für jede Pseudozufallszahl neu berechnet. Die Rechenvorschrift ist im Codeausschnitt 5.6 in Zeile vier und fünf programmiert: $\boldsymbol{ro0 = \overline{ro0}\ \text{xor ro8 xor } ro2 \text{ xor } \overline{\text{ro7}} \text{ xor ro4.}}$
Die verwendete XOR-Verknüpfung hat eine symmetrische Ergebnisverteilung. Das bedeutet, dass von den vier möglichen Eingangszuständen bei zwei Eingängen zweimal eine Eins als Lösung folgt und zweimal eine Null. Somit ist die Wahrscheinlichkeit gleich verteilt, welches Ergebnis die Rechnung hat.
Die im Code 5.6 folgende case – Anweisung realisiert, das ebenfalls in Kapitel 4.3.1 beschrieben Zeitverhalten der Adressbit. Zuerst wird das eine Schieberegister angesprochen, es folgen mehrere Zyklen, bei denen die Adresse 00 ist und somit die Datensignale nicht gelesen werden. Gefolgt von dem Befehl zum Beschreiben des zweiten Schieberegisters und einer weiteren 00 – Adressphase, wird der Interrupt für den Mikrocontroller ausgelöst, dass die Daten an den Schieberegistern anliegen. Nach einer erneuten Wartezeit, in der der Mikrocontroller die Daten einlesen kann, beginnt ein neuer Sendezyklus.

5.4.4 Coder und Decoder



4 – Bit – Wort	5 – Bit – Wort
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

Abbildung 5.7: *4Bit / 5Bit – Codierung*

Bei der ersten Inbetriebnahme waren die beiden Instanzen Coder und Decoder noch nicht realisiert. Notwendig wurden sie, da es zu Fehlern bei der Übertragung langer Folgen gleicher Pegel kam. Die Ursachen sind in Kapitel 4.2.2 beschrieben.
Um die langen Folgen gleicher Pegel zu vermeiden und somit häufigere Pegelwechsel zu garantieren, werden die ausgehenden Daten mit der 4Bit/5Bit – Codierung übertragen.
Ankommende Datenpakete werden nach der gleichen Vorschrift, die in der nebenstehenden Tabelle dargestellt ist, wieder decodiert.
Die Zuordnung ist so gewählt, dass maximal 8 gleiche Pegel hintereinander vorkommen können. Dies ist für die optische Schnittstelle kein Problem und die Daten werden fehlerfrei übertragen.
Abbildung 5.8 zeigt, wie sich die Datenpakete durch die 4Bit/5Bit – Codierung verändern. Durch grüne Pfeile ist die Codierung dargestellt. Aus den 4 Bit – Datenwörtern werden 5 Bit lange. Die beiden Adressbit A und B werden übernommen und an die vorgesehene Stelle geschrieben. Das freibleibende Bit wird immer mit einer logischen Null beschrieben.

Mit den orangenen Pfeilen ist die Decodierung verdeutlicht. Aus den langen 5 Bit breiten Datenwörtern werden wieder kurze 4 Bit breite. Die Adresse wird verschoben und die vorher zugefügte Null verfällt. Somit ist das in Abbildung 5.8 obere Datenpaket identisch mit dem unteren dargestellten Datenpaket.
Gesendet und empfangen werden ausschließlich Datenpaket von dem mittleren Format (Abbildung 5.8).

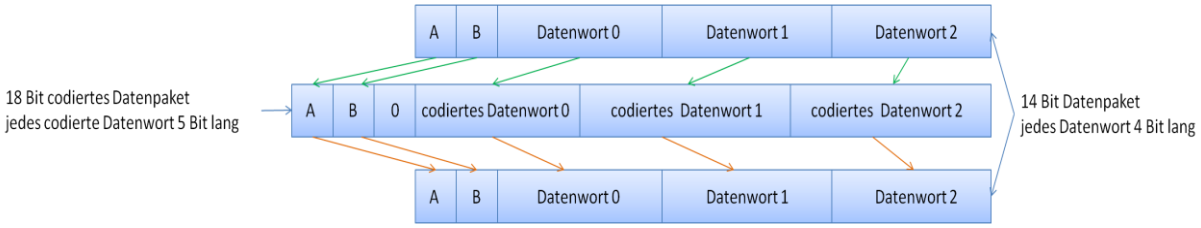


Abbildung 5.8*: Vergleich codiertes und uncodiertes Datenwort*

5.4.5 Comparison

Um die Übertragungsfehlerrate des Systems zu bestimmen, müssen die gesendeten Daten mit den wieder Empfangenen verglichen werden. Die Instanz Comparison (deutsch: der Vergleich) realisiert diesen Vergleich und zählt die Anzahl der gesendeten Datenpakete.
Verglichen werden die Pakete vor dem Coder und nach dem Decoder, also jeweils die uncodierten Daten.
Die Anzahl der gesendeten Daten wird in den Integersignalen nsd1 bis nsd10000000000 (nsd = number send data) gespeichert. Diese stellen jeweils eine Zehnerpotenz der Summe dar.

```
if rising_edge (dev_clock_2_5MHz) then
if nsd1 < 9 then
  nsd1 <= nsd1 +1;
  else
        nsd1 <= 0;
     if nsd10 < 9 then
      nsd10 <= nsd10 +1;
    else
      nsd10 <= 0;
```

Code 5.7: *Zählschleife*

Die Funktionsweise dieser Zählschleife ist im Code 5.7 veranschaulicht. Die Schleife wird bei jeder positiven Flanke des Taktsignales angestoßen. Bei dem Anstoß wird die Variable eins hochgezählt. Hat sie den Wert 9, so wird sie wieder auf 0 gesetzt und die nächste Stelle wird um eins erhöht. Da die Daten im späteren Betrieb mit 25 MHz gesendet werden sollen, wären die verfügbaren elf Stellen für Langzeittests nicht geeignet. Deshalb wurden Schleifen mit den Signale nsdnotshown1, nsdnotshown10 und nsdnotshown100 vor die eigentliche Zählschleife gesetzt. Der Inhalt dieser Signale wird nicht am LCD dargestellt. Somit kann diese Schleife bis 10^{13} laufen.
Die Schleife, die die Daten vergleicht, funktioniert annähernd identisch. Es bestehen zwei Unterschiede zu oben genannter Schleife. Zum Einen wechselt nicht automatisch bei jedem Taktsignal das unterste Signal seinen Zustand, sondern bei jedem Taktsignal wird das gesendete mit dem fast zeitgleich ankommenden Daten verglichen. Zum Anderen werden alle Stellen weitergeleitet und somit am Display dargestellt. Es muss sichergestellt werden, dass zum Zeitpunkt des Vergleichs dieselben Datenpakete noch anliegen. Bei 25 MHz ist dies nicht ohne größeren Programmieraufwand möglich, der aus Zeitgründen nicht mehr realisiert werden konnte. 25 MHz entspricht einer Bitdauer von 40 ns. Dies bedeutet, dass alle 40 ns ein neues Datenpaket an den Ausgängen anliegt. Da die Daten aber über die Gesamtstrecke länger als 40 ns brauchen, liegt beim Vergleich immer das aktuell zu sendende Datenpaket an, aber an den Eingängen hingegen noch das letzte Datenpaket.
Deshalb können alle Langzeittests nur mit einer parallelen Datenrate von 2,5 MHz ausgeführt werden.
Bei Druck des Resetknopfes werden alle Zählvariablen auf Null gesetzt.

5.4.6 Set_LCD

	0011xxxx	0100xxxx	0101xxxx	0110xxxx	0111xxxx
xxxx0000	0	@	P	`	p
xxxx0001	1	A	Q	a	q
xxxx0010	2	B	R	b	r
xxxx0011	3	C	S	c	s
xxxx0100	4	D	T	d	t
xxxx0101	5	E	U	e	u
xxxx0110	6	F	V	f	v
xxxx0111	7	G	W	g	w
xxxx1000	8	H	X	h	x
xxxx1001	9	I	Y	i	y
xxxx1010	:	J	Z	j	z
xxxx1011	;	K	[	k	{
xxxx1100	<	L	¥	l	|
xxxx1101	=	M	]	m	}
xxxx1110	>	N	^	n	←
xxxx1111	?	O	_	o	→

Die in der Instanz Comparison erzeugten Integersignale sollen auf dem LC – Diplay angezeigt werden. Da es sich um ein digitales System handelt, können Intergerzahlen nicht einfach übermittelt werden, sondern müssen einer Binärzahl zugewiesen werden.
Bei dem verwendeten Display auf dem Spartan 3E Evaluationsboard müssen, damit ein Zeichen dargestellt werden kann, zwei vier Bit breite Binärzahlen hintereinander an das Display gesendet werden. Ein Ausschnitt der möglichen Zeichen ist in der obenstehenden Tabelle dargestellt.
Jedem darstellbaren Symbol ist eine eindeutige Bitfolge zugeordnet. Die Highbit sind in einer Spalte gleich und stehen ganz oben in der Tabelle. In einer Reihe der Tabelle sind die Lowbit konstant. Diese stehen in der linken Spalte. Die Kombination aus High- und Lowbit beschreiben ein Symbol eindeutig.
Es ist notwendig für jedes Symbol, das dargestellt wird, zwei Signale zu generieren, eins für die Highbit und eins für die Lowbit.

5.4.7 Status_LCD

Es sind zwei Anzeigen am LCD möglich. Einmal wird in der oberen Reihe die Fehleranzahl (number of failure, kurz: nof) und die Anzahl gesendeter Daten (number send data, kurz: NSD) dargestellt oder es werden wie in Abbildung 5.9 links dargestellt, gesendete und empfangene Daten gegenübergestellt. In dieser Instanz wird ein Schalter abgefragt und bei Betätigung des Schalters wechselt die Anzeige.

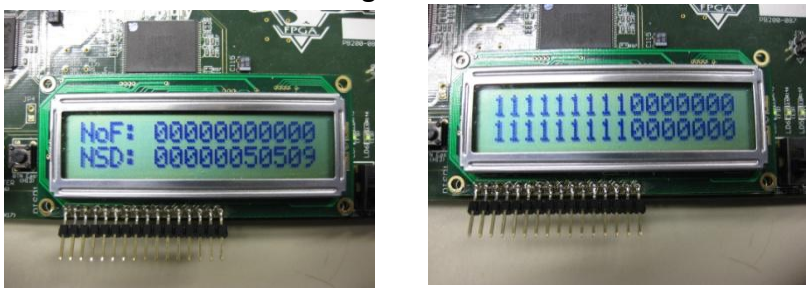


Abbildung 5.9*: LCD - Anzeigen*

5.4.8 Counter

Um dem LC – Display Befehle zeitgenau senden zu können, zählt die Instanz Counter eine 15 Bit breite Variable hoch. Mit einer Frequenz von 25 kHz, also alle 40 µs, wird binär eine eins addiert.
Ein Schreibzyklus dauert bis die Variable einmal alle möglichen Werte angenommen hat. Dies ist erreicht, wenn das Signal aus 15 Einsen besteht, also nach 32.767 Durchläufe. Somit wird das Display alle 1,3 Sekunden neu beschrieben.

5.4.9 Mac_LCD

Das LC – Display wird mit Hilfe von vier Signalen gesteuert:

- **lcd_dat:** Vierbitbus, über den die Datenwörter oder Steuerbefehle an das Display gesendet werden.
- **lcd_e:** Enableleitung, bei einer positiven Flanke auf dieser Leitung werden die anstehenden Daten gelesen.
- **lcd_rs:** Register Select, je nach Pegel unterscheidet das Display zwischen Steuerbefehlen oder Daten, die dargestellt werden sollen.
- **lcd_rw:** Read/Write Status: gibt an, ob anliegende Daten gelesen werden sollen oder bestimmte Daten ausgegeben werden soll.

Die Instanz Mac_LCD fragt ständig den Zustand der Zählvariablen der Instanz Counter ab. Beim ersten Durchlauf der Variablen werden festdefinierte Befehlsketten an das LCD gesendet, mit denen die Grundeinstellungen vorgenommen werden. Ab dem zweiten Durchlauf werden ausschließlich die Variablen abgefragt, die dargestellt werden sollen. Dies sind entweder die Vierbit – Signale aus der Instanz Set_LCD, die die Datenwörter für die gesendeten und empfangenen Daten beinhalten oder die Datenworter der Zählvariablen für gesendete Daten und erkannte Fehler. Je nach Zustand der Variablen status werden die entsprechenden Datenwörter nacheinander über die lcd_dat Leitung an das Display angelegt. Immer gefolgt von einem Impuls auf der Enable – Leitung, wenn die Daten anstehen.

5.4.10 Initserialiser

Der Serialiser/Deserialiser – Baustein DS92LV18 hat eine Reihe von Intitialisierungspins. In diesem VHDL – Modul Initserialiser können die Pegel gesetzt werden. Da es für dieses Projekt nicht notwendig ist eine dieser vorgenommen Einstellungen während des Betriebes zu ändern, sind diese Signale nicht taktabhängig programmiert. Dies spart eine taktgesteuerte Abfrage der Signalwerte.

```
process(dev_clock_25kHz)
begin
                  Sync        <= '0';
                  Local_Le    <= '0';
                  Den         <= '1';
                  Tpwdn       <= '1';
                  Line_Le     <= '0';
                  Ren         <= '1';
                  Rpwdn       <= '1';
end process;
```

Code 5.8: Zuweisung Inititialisierung

Im Code 5.8 ist die für dieses Projekt standardisierte Einstellung dargestellt.

- **Sync = 0:** Mit dieser Einstellung sendet der Baustein die anliegenden Daten an die Ausgänge, bei einem Highpegel würden Synchronisationsmuster gesendet und die Eingänge ignoriert.
- **Line_Le = 0:** Ist dieser Pin Low, so gibt der Baustein die an DIN0 bis Din17 parallel anliegenden Daten seriell an die Ausgangspins DO± aus. Ist der Local_Le – Pin dagegen High, so würde der Baustein die an RIN ± seriell ankommenden Daten seriell ausgeben.
- **Local_Le = 0:** Mit dieser Einstellung werden die seriell ankommenden Daten an RIN± parallelisiert und an den Ausgängen ROUT0 bis ROUT17 angelegt. Andernfalls würden die an den DIN – Pins parallel anliegenden Daten parallel ausgegeben.
- **TPWDN = 1:** Schaltet die Sender – PLL an.
- **RPWDN = 1:** Schaltet die Empfänger – PLL an.
- **DEN = 1:** Sendeeingangssignale werden durchgeschaltet, bei einer logischen Null wären die Ausgänge hochohmig.
- **REN = 1:** Empfängereingangssignale werden durchgeschaltet, bei einer logischen Null wären die Ausgänge hochohmig.

6 Gehäuse zur Reduktion der Störemission

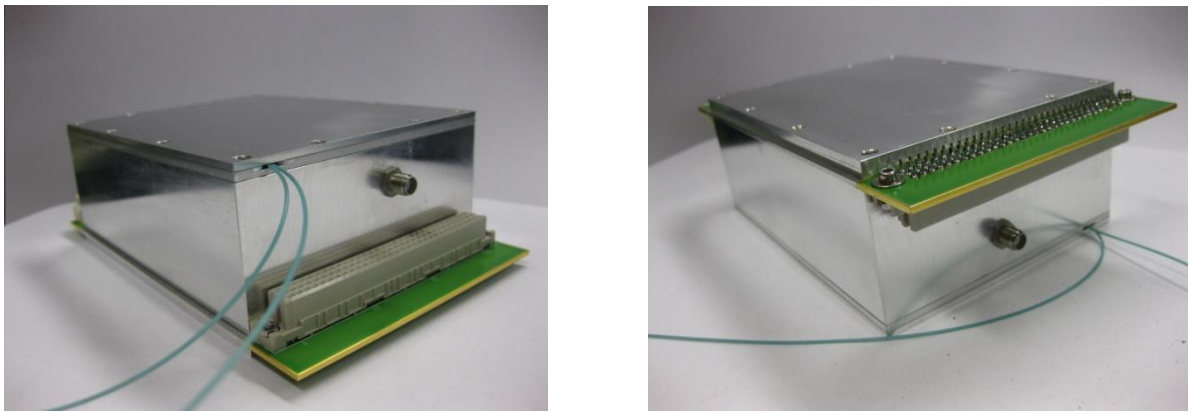


Abbildung 6.1: *Gehäuse RxBussystem links: Oberseite rechtes: Unterseite*

Die von dem Radioteleskop Effelsberg empfangenen Signale sind um ein vielfaches schwächer als das Umgebungsrauchen, das größtenteils durch den Menschen verursacht wird.
Da sich Störungen, die im Fokus des Spiegels erzeugt werden, sehr stark auf die Messergebnisse auswirken, ist es wichtig das keine Störstrahlung austritt.
Um diese Störungen zu unterdrücken wurde das in Abbildung 6.1 gezeigte Gehäuse für das RxBussystem entworfen[1].
Das Gehäuse besteht aus insgesamt vier Teilen, dem Bodenteil, Deckelteil, Deckelteil Innen und Deckelteil Außen. Jedes der vier Einzelteile ist aus je einem Alublock gefräst, um optimale Störunterrückung zu gewährleisten.
Über den SMA – Stecker im Deckelteil kann das externe Taktsignal für den Serialiser / Deserialiser – Chip eingespeistet werden. Dieser Stecker ist intern direkt mit dem SMA – Stecker, der sich auf der Platine befindet, verbunden.
In dem Bauteil Deckelteil Innen (siehe Pläne Kapitel 9.3) ist eine Führung eingefräst, durch die die Lichtwellenleiter nach außen gelegt werden. Das Deckelteil Außen schließt das Gehäuse nach oben hin ab, so dass auch durch die Öffnung, durch die die Lichtwellenleiter führen, keine Störstrahlung emittieren kann.
Für das TxBussystem wurde kein Gehäuse entworfen, da im Rahmen dieser Diplomarbeit nur Probemessungen (siehe Kapiel 7.2) durchgeführt wurden um den Erfolg dieses Verfahrens zur Störunterdrückung zu belegen.

1: Die Ausarbeitung der Idee zu dem Gehäuse ist eine Gemeinschaftsarbeit von Dipl. – Ing. (FH) Thomas Berenz und Torsten Krause. Die Pläne zur Fertigung für die Werkstatt wurden angefertigt von Dipl. – Ing (FH) Thomas Berenz und Peter Winkelmann.

7 Messungen SERELECS

7.1 Messung Signalqualität

7.1.1 Bestimmung der Bitfehlerrate

Nachdem die Platinen bestückt und das System soweit getestet wurde, dass eine Übertragung möglich ist, konnten genauere Untersuchungen durchgeführt werden:

- **Funktionstest TxBussystem (25 MHz parallele Datenrate):** Bei der ersten Untersuchung wurde nur das TxBussystem untersucht. Dafür wurde die Sende- und die Empfangseinheit der optischen Schnittstelle gebrückt.
  Somit wurden die gesendeten Daten sofort wieder auf den Eingang gegeben. Da wie in Kapitel 5.4.5 beschrieben, ein Vergleich mit dem FPGA in dieser Sendegeschwindigkeit nicht möglich ist, wurden die gesendeten und empfangenen Daten mit einem Oszilloskop verglichen. Es wurde über 2,5 Stunden ein willkürlich gewähltes Bit auf der Sende- und der Empfangsseite gemessen und stichprobenartig durchschnittlich 10 mal pro Minute verglichen. Es konnte kein Fehler festgestellt werden. Da diese Messung wenig aussagekräftig ist und nicht sichergestellt ist, dass es nicht zu Übertragungsfehlern kommt, wurde die Datenrate reduziert.

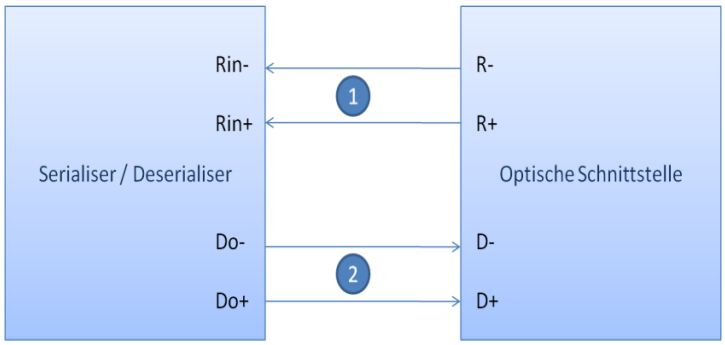
- **Funktionstest TxBussystem (2,5 MHz parallele Datenrate):** Bei einer Datenrate von 2,5 MHz bei den parallelen Daten, also 50 MHz seriell, war das FPGA in der Lage die Datenpakete zu vergleichen. Bei einer Messung von 20 Stunden wurden bei 2,5 MHz Übertragungsrate $180 \cdot 10^9$ Datenpakete übermittelt. Dies entspricht $3{,}24 \cdot 10^{12}$ Datenbit. Damit wurden genug Daten übertragen um mit einer großen statistischen Genauigkeit Übertragungsfehler auszuschließen.

- **Funktionstest RxBussystem (2,5 MHz parallele Datenrate):** Bei der RxBussystem – Platine wurde ebenfalls die optische Schnittstelle kurzgeschlossen. Der Mikrocontroller hat Pseudozufallsdaten gesendet und diese mit den empfangenen Daten verglichen. Auch dieser Test wurde über 20 Stunden durchgeführt, so dass ebenfalls $3{,}24 \cdot 10^{12}$ Datenbit übermittelt wurden. Auch hier kam es zu keinem Fehler.

- **Funktionstest gesamte Übertragungsstrecke:** Da die beiden Platinen für sich mit einer großen statistischen Genauigkeit fehlerfrei arbeiten, wurde die Gesamtkommunikationsstrecke getestet. Das heißt, das FPGA erzeugt Daten, die vom TxBussystem über Lichtwellenleiter an das RxBussystem gesendet werden. Der Mikrocontroller liest die Daten und sendet das gleiche Datenpaket wieder zurück. Hier lagen ebenfalls am FPGA beim Abfragezeitpunkt für den Vergleich der gesendeten und empfangenen Daten nicht mehr dasselbe Datenpaket an. Mit dem Oszilloskop wurde, wie bei der ersten Messung, ca. 1500 Probemessungen vorgenommen und es kam zu keinem Fehler.

Die oben beschriebenen Messungen haben gezeigt, dass das System mit einer sehr hohen Sicherheit richtig überträgt. Aus zeitlichen Gründen war es nicht möglich im FPGA ein Halteglied oder Speicher zu realisieren, der die gesendeten Daten so lange zwischenspeichert, bis das gleiche Datenpaket wieder ankommt und verglichen wurde.
Um das System im Teleskop einsetzten zu können sind weitere Tests notwendig.

7.1.2 Augendiagramme und Signalformen



Die Übertragung der schnellen seriellen Daten zwischen der optischen Schnittstelle und dem Serialiser / Deserialiser – Baustein ist der kritische Punkt bei der Datenübertragung, da bei so hohen Datenraten selbst kleine Störungen zu Fehlern führen können.

Abbildung 7.1: *Übersicht Messpunkte*

Verglichen werden die Signale zwischen optischer Schnittstelle und dem Serialiser / Deserialiser in beide Richtungen.
Da die Daten symmetrisch übertragen werden, wurde zur Messung der Signale ein differentieller Tastkopf verwendet, der beide Leitungen zeitgleich misst und intern subtrahiert.

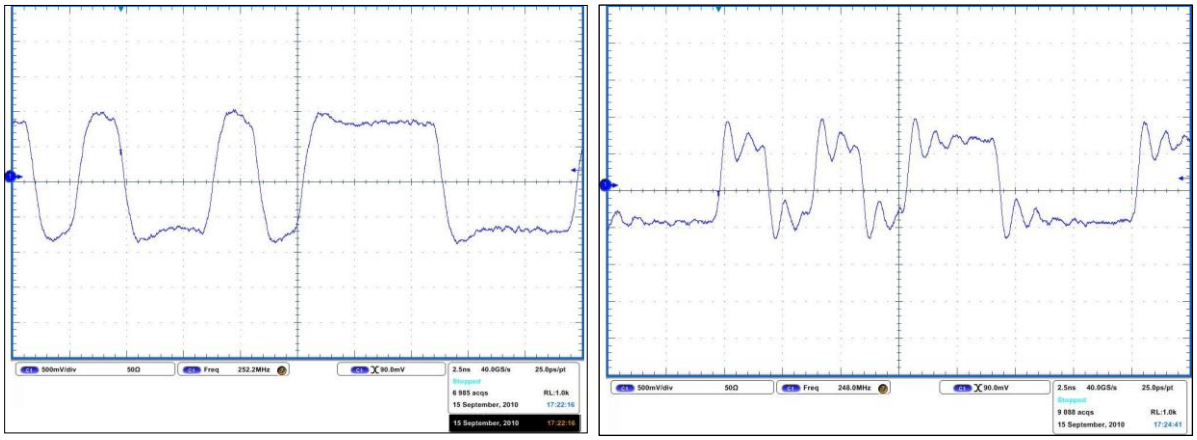


Abbildung 7.2: *Signalform links: Messpunkt 1 rechts: Messpunkt 2*

Abildung 7.2 zeigt deutlich, dass die beiden Signale sich stark unterscheiden. Zu beachten ist, dass die beiden Hälften der Abbildung nicht die gleiche Bitfolge zeigen.
In der Qualität der Signale sind Unterschiede zu erkennen. So haben die Signale die von dem Serialiser – Baustein zur optischen Schnittstelle gesendet werden viel ausgeprägtere Überschwinger als die Daten in umgekehrter Richtung.
Die Leitungen an Messpunkt 1 sind mit einem Widerstand reflexionsfrei abgeschlossen. Bei den Leitungen mit der schlechteren Signalqualität wurde darauf verzichtet, da die entsprechenden Eingänge der optischen Schnittstelle abgeschlossen sind (siehe Kapitel 4.2.1).
Versuche haben ergeben, dass ein Widerstand parallel zwischen den Leitungen nicht die Lösung des Problems ist, da verschiedene Widerstände eingepasst wurden und es zu keiner sichtbaren Signalverbesserung kam.

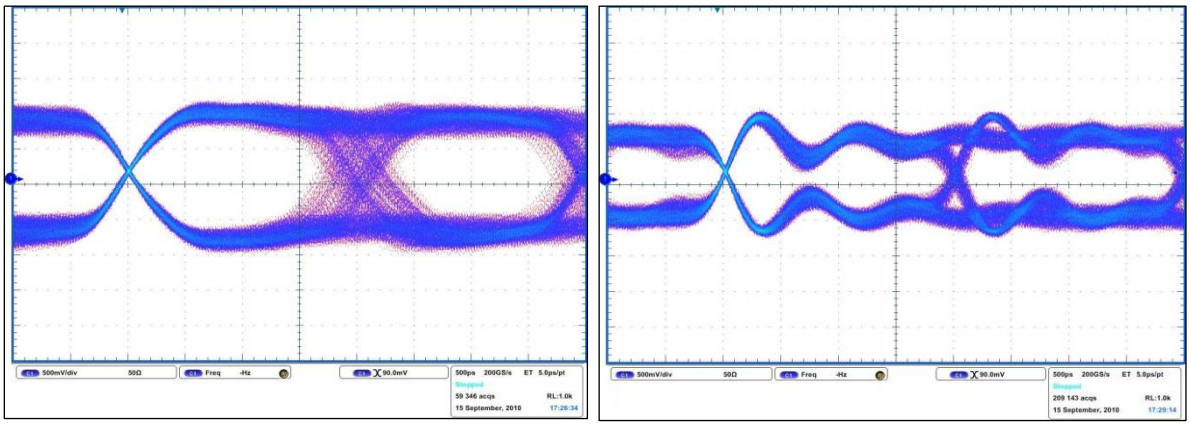


Abbildung 7.3: *Augendiagramme links: Messpunkt 1 rechts: Messpunkt 2*

Die Augendiagramme (Abbildung 7.3) dienen hauptsächlich zur Beurteilung der Signalqualität. Es werden die Zustandsübergänge zwischen den einzelnen Symbolen dargestellt.
Je weiter das Auge in vertikaler Richtung geöffnet ist, desto unempfindlicher ist das Signal gegenüber dem Rauschen. Je kleiner das Auge in horizontaler Richtung geöffnet ist, desto empfindlicher ist es gegenüber Jitter (Abtastzeitschwankungen).
Auch hier sind deutliche Unterschiede zu sehen. Die Augenöffnungen in horizontaler Richtung sind annähernd gleich. Die Öffnung in vertikaler Richtung jedoch nicht. Die bei den Einzelsignalaufnahmen zu sehenden Überschwinger zeigen sich auch hier.
Wie in Kapitel 7.1.1 beschrieben, kommt es trotz dieser unsauberen Signale zu keinen Fehlern bei der Übertragung. In den Augendiagrammen ist eine Region in der Mitte zu erkennen, durch die kein einziges Signal läuft. Alle Signalpegel werden zum Abtastzeitpunkt richtig interpretiert.

Eine mögliche Verbesserung könnte eine Serienterminierung bringen, bei der ein Widerstand jeder Leitung in Serie geschaltet wird. Da dafür jedoch eine neue Platine bestellt werden müsste, konnte dies nicht mehr realisiert werden.

7.2 Messung EMV - Empfindlichkeit

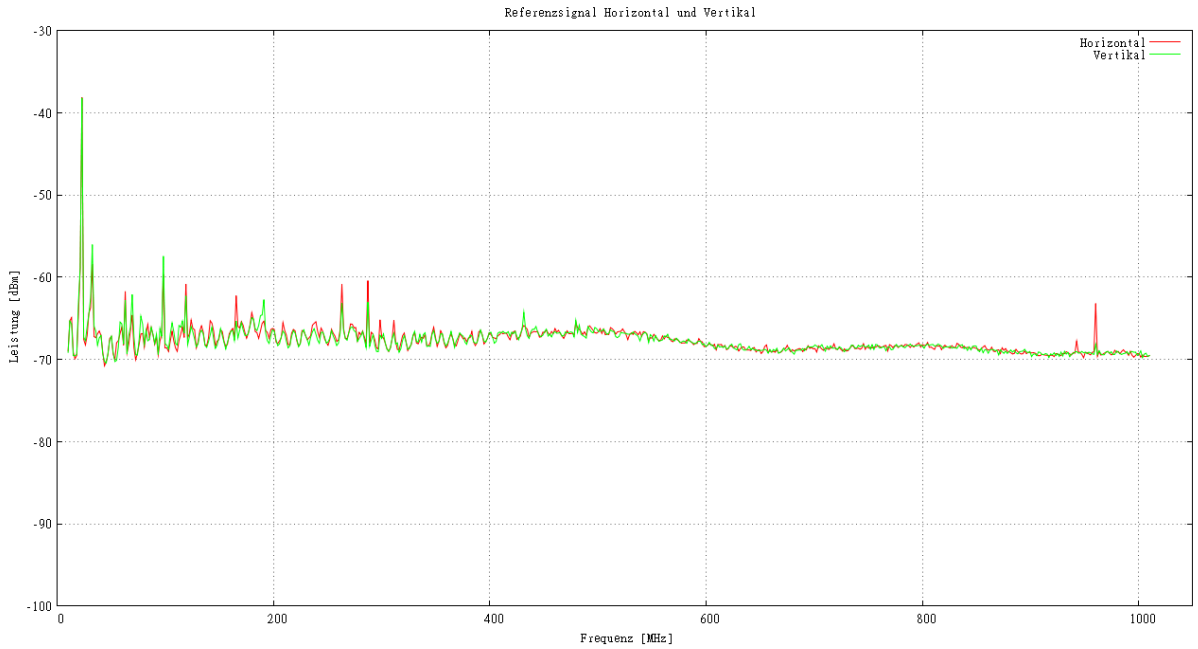
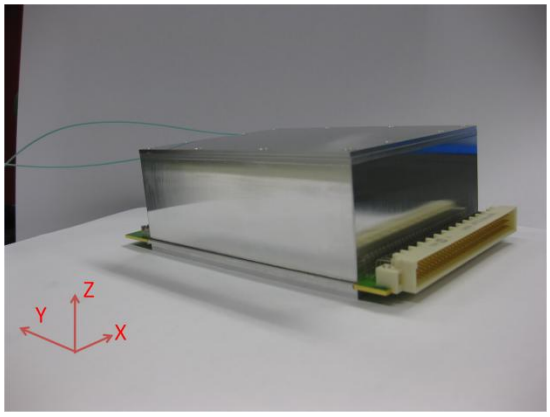


Abbildung 7.4: *Referenzsignal Antenne*

Mit Hilfe einer Messantenne wird die elektromagnetische Störstrahlung gemessen, die das RxBussystem emittiert.
Zum Einen wird gemessen wie viel Störemmision mit und ohne Gehäuse auftritt und zum Anderen wird mit einer Nahfeldsonde bestimmt welches Bauteil welche Störungen verursacht.
Jedes elektromagnetische Feld besteht aus einer elektrischen- und einer magnetischen Komponente. Diese stehen senkrecht aufeinander. Da sich aber die Ausrichtung dieses lotrechten Paares ändert, müssen um ein Feld zu beschreiben in horizontaler und in vertikaler Richtung die elektrischen Strahlungen gemessen werden.



Um festzustellen in welche Richtung wieviel Störemission abgestrahlt wird, wurde die Antenne auf unterschiedliche Seiten der RxBussystem – Platine gerichtet.
Um die Richtungen eindeutig beschreiben zu können, wurde das in Abbildung 7.5 dargestellte Koordinatenssystem verwendet.

Abbildung 7.5: *Koordinatenfestlegung*

In Abbildung 7.4 sind die beiden Referenzsignale dargestellt. Diese zeigen die Messwerte, die ohne Messobjekt im leeren Messraum aufgenommen wurden. Dieser Raum ist mit Absorptionsmaterial ausgekleidet, so dass keine elektromagnetische Strahlung von außen in den Raum dringt. Die Antenne hat eine untere Grenzfrequenz von 300 MHz und der Vorverstärker eine obere Grenzfrequenz von 1 GHz.

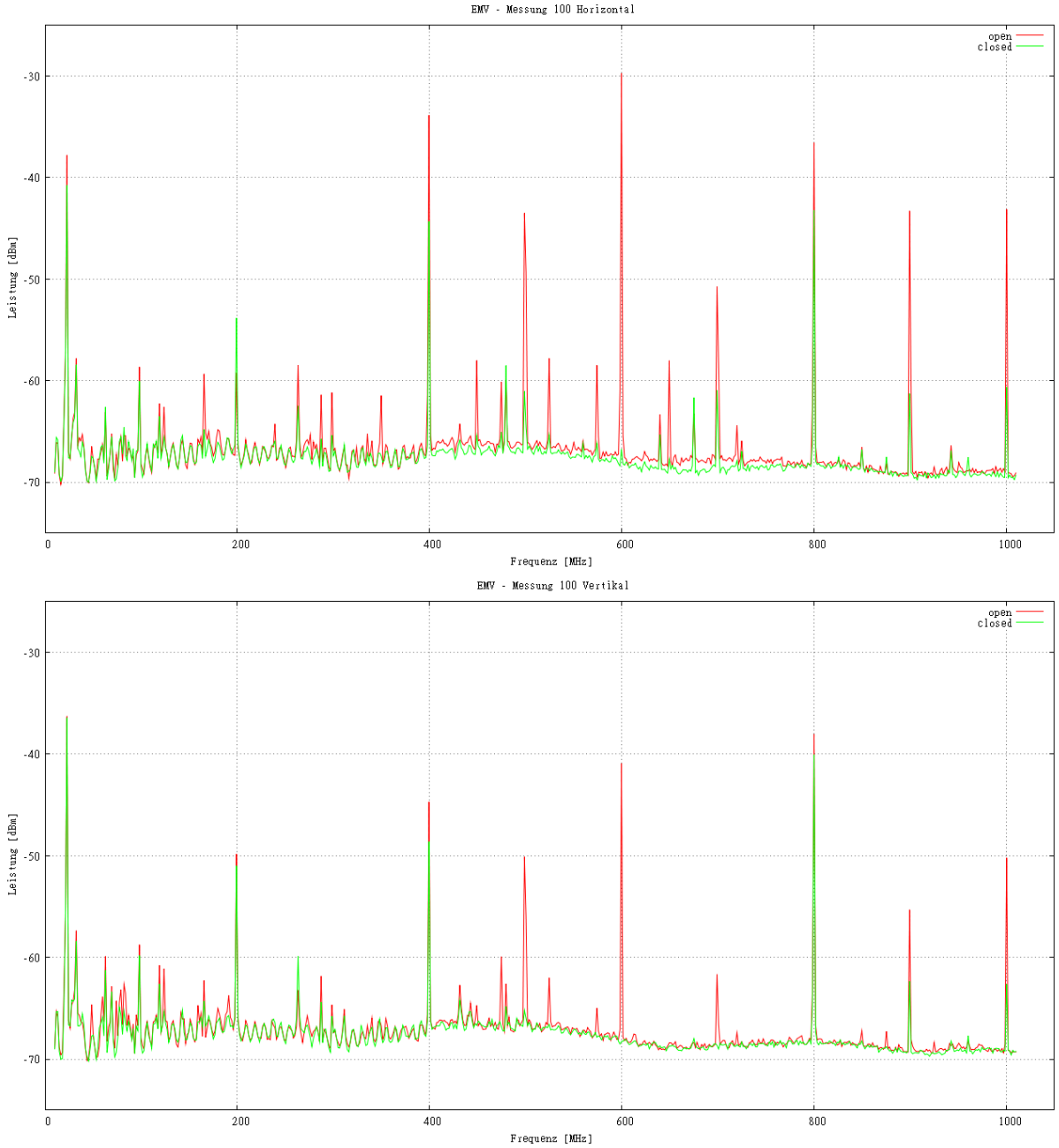
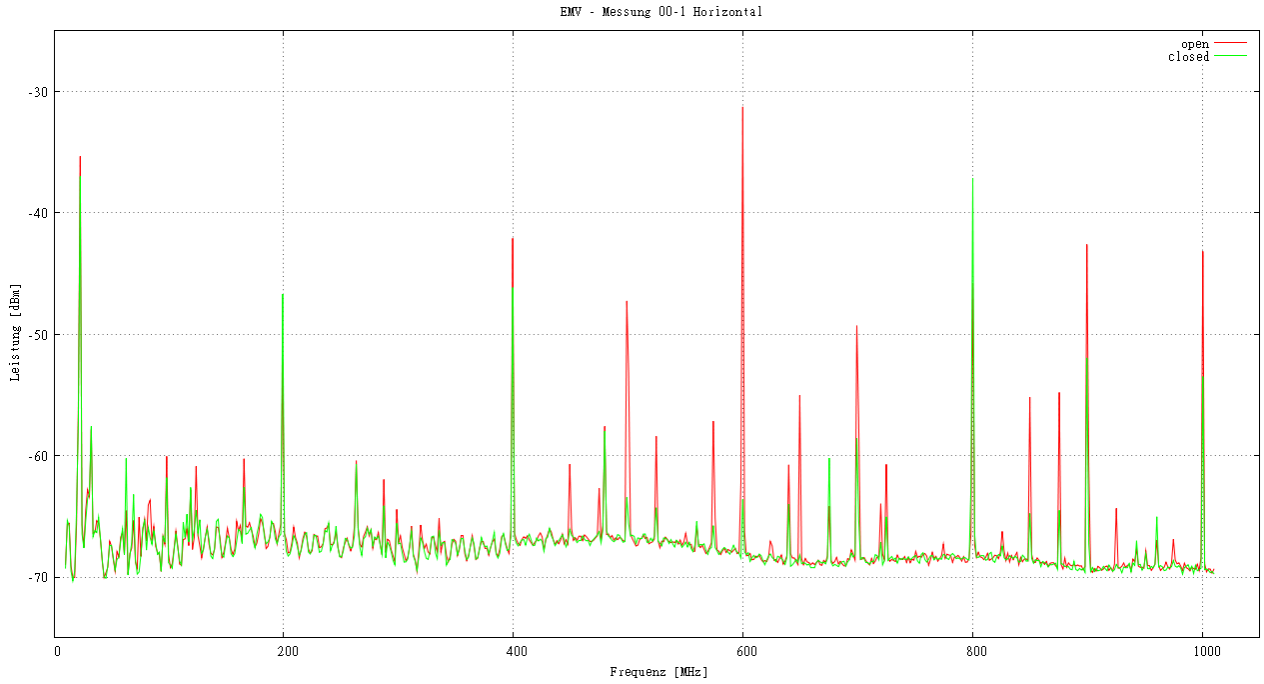


Abbildung 7.6: EMV – Messung in X - Richtung



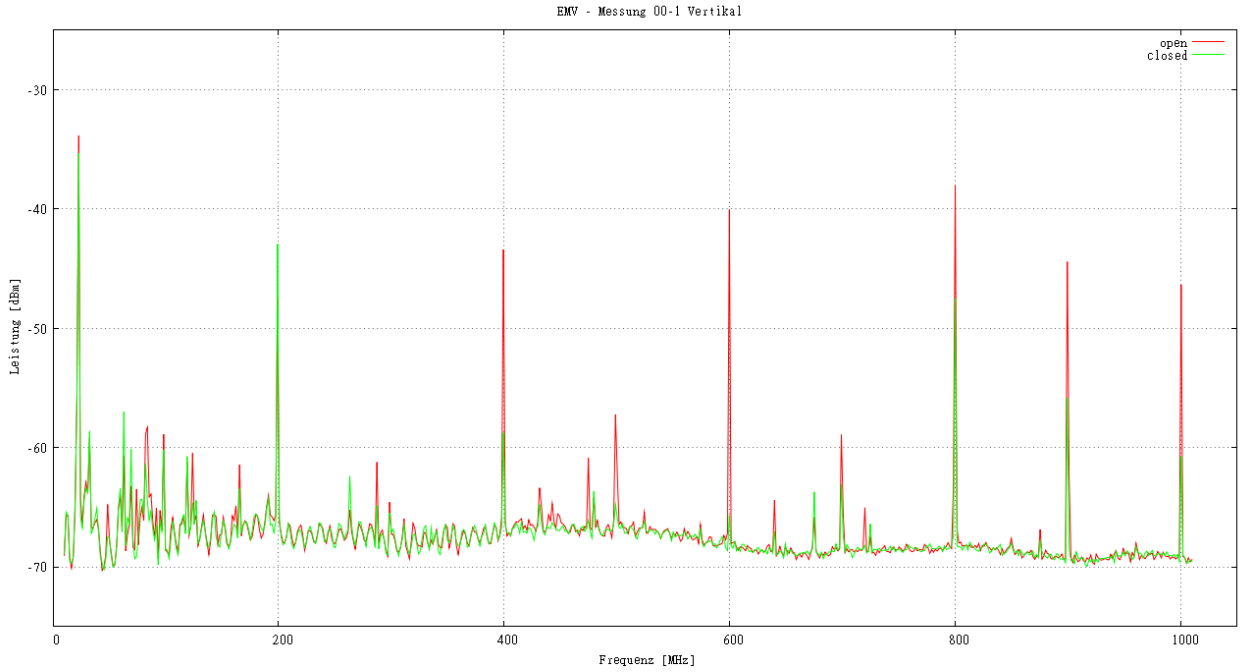


Abbildung 7.7: *EMV – Messung in (-Z) - Richtung*

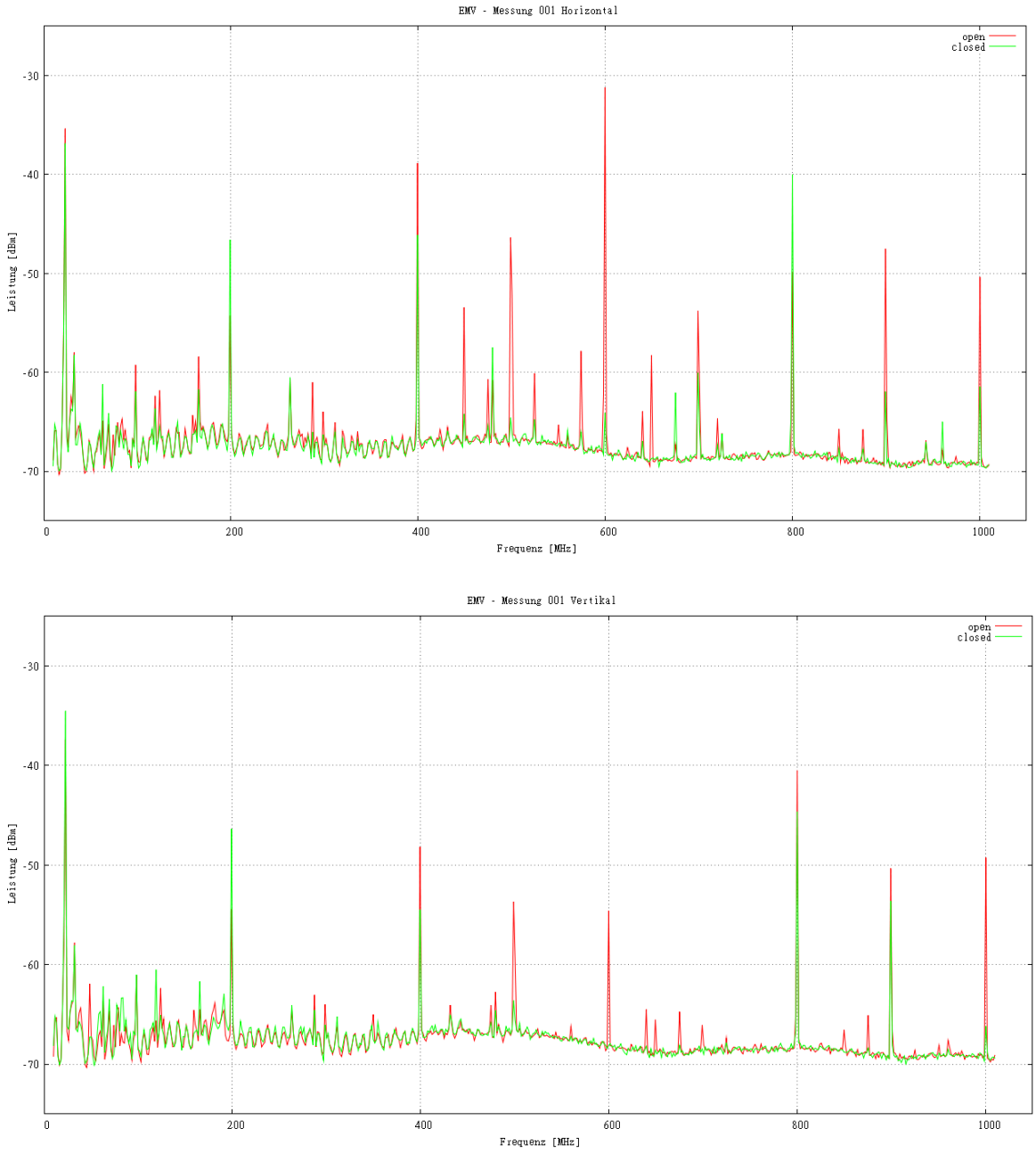


Abbildung 7.8: *EMV – Messung in Z - Richtung*

In den vorrausgegangenen drei Abbildungen sind drei der Messreihen dargestellt. Jedes ist aus einer anderen Richtung gemessen. Der rote Graph zeigt das Störspektrum ohne das Gehäuse und der grüne das Spektrum mit Gehäuse. In der oberen Hälfte sind die horizontalen- und in der unteren Hälfte die vertikalen Messungen dargestellt.
Die Messungen haben gezeigt, dass die Störemmision nicht ausreichend von dem Gehäuse unterdrückt wird. Bei zwei der dargestellten Messungen, waren bei 800 MHz sogar mit Gehäuse mehr Störstrahlung zu messen als ohne. Der Grund für diese Erhöhung ist unbekannt, wurde aber mit anderen Messungen bestätigt. Dahingegen können bei 600 MHz sehr gute Ergebnisse erzielt werde. Messungen mit der Nahfeldsonde haben gezeigt, dass die meiste Störstrahlung von den Steckern verursacht wird, die nach außen geführt sind und bei der Messung Signale anliegen hatten.

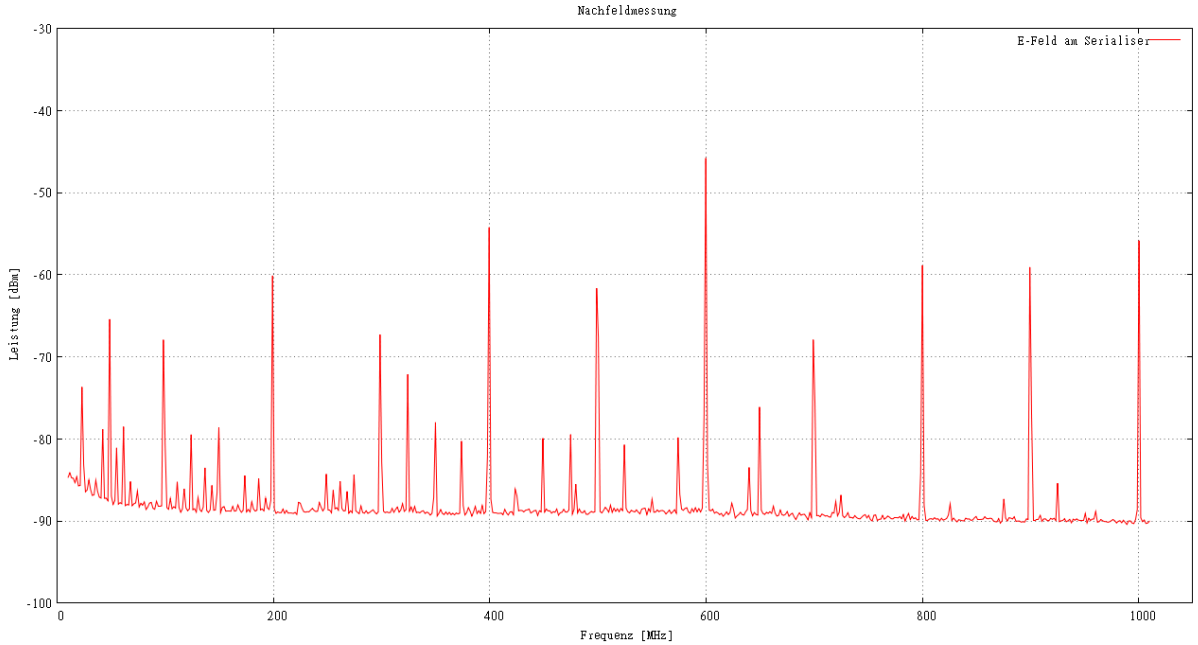


Abbildung 7.9: *Nahfeldmessung am Serialiser*

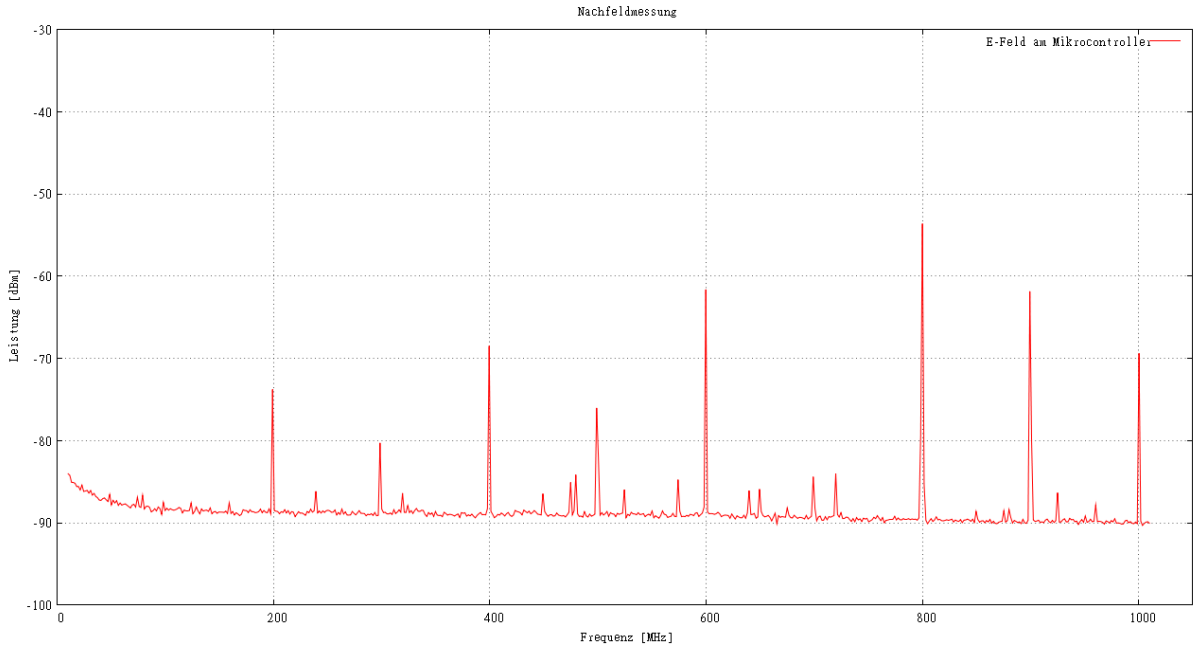


Abbildung 7.10: *Nahfeldmessung am Mikrocontroller*

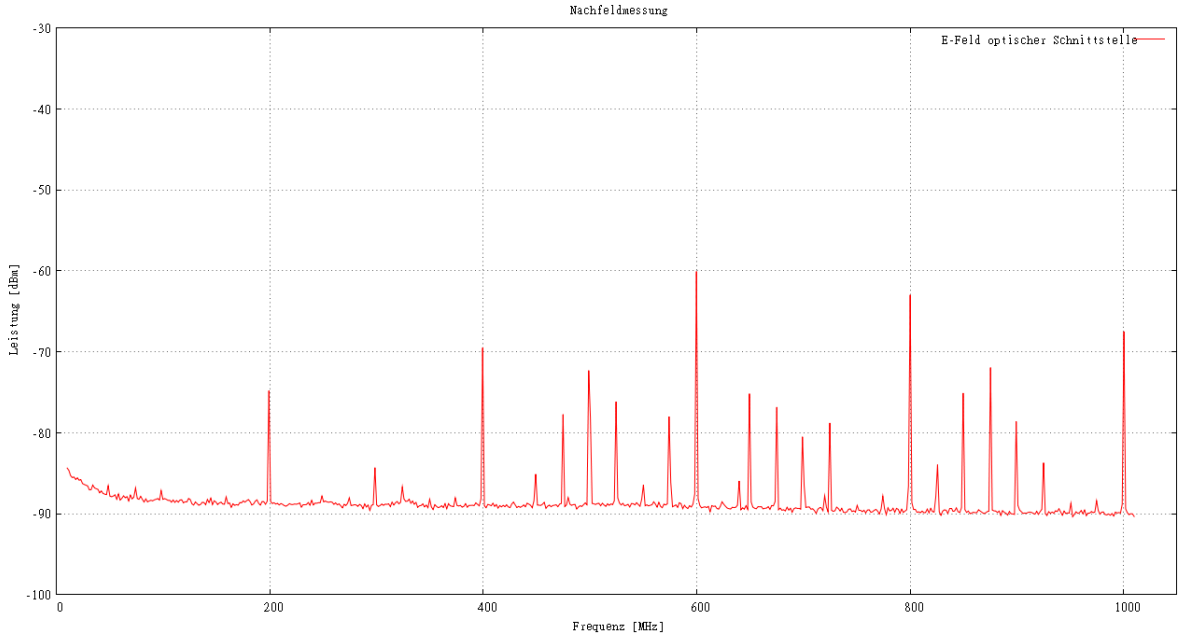


Abbildung 7.11: *Nahfeldmessung an optischer Schnittstelle*

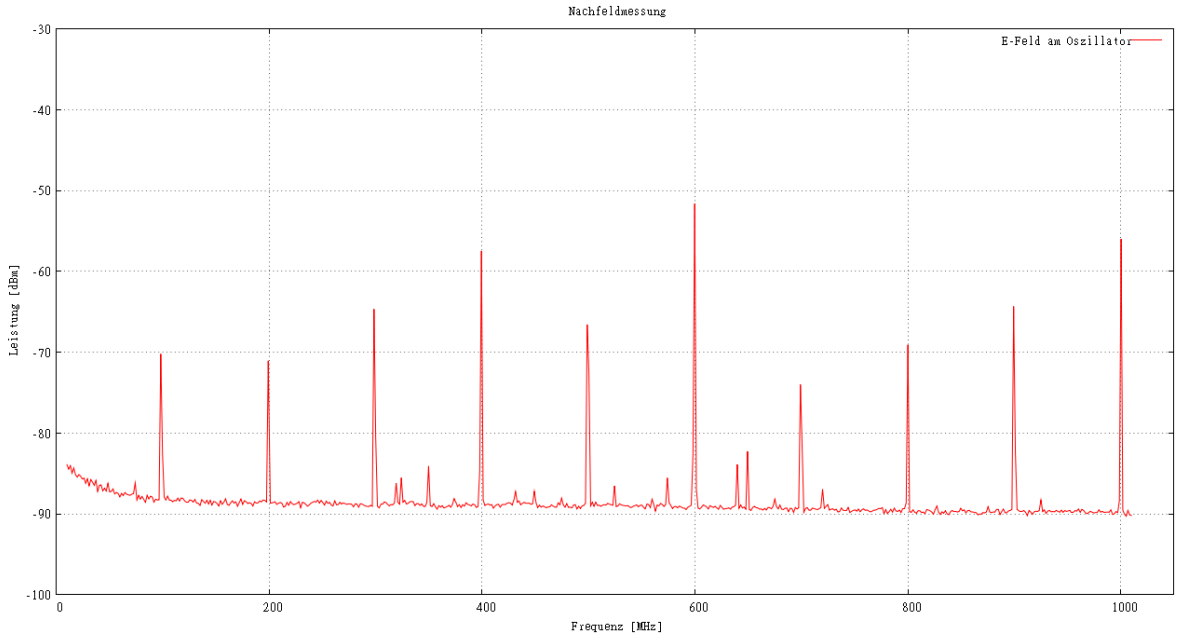


Abbildung 7.12: *Nahfeldmessung am Oszillator*

Weitere Messungen mit der Nahfeldsonde haben die Störemission in der Nähe der einzelnen Bausteine aufgezeichnet. Hier wurden nicht die Einzelspektren der Bausteine gemessen, sondern das Störsignal in der Nähe des Bausteins, dieses ist aber durch andere Störer beeinflusst.
Bei dem Spektrum am Serialiser zeigen sich viele Spektralanteile im 25 MHz Abständen. Dies ist die Frequenz, mit der dieser Baustein arbeitet. Das selbe zeigt sich beim Oszillator im 100 MHz Abstand. Die Spektren am Mikrocontroller und der optischen Schnittstelle zeigen dahingegen Peaks, die sich nicht genau einer Quelle zuordnen lassen.

8 Fazit und Aussichten für das Projekt

Das entwickelte System wurde als Prototyp entworfen. Es umfasst nicht die kompletten, späteren Funktionalitäten, aber schon einen sehr großen Teil. Das Grundkonzept funktioniert und es kommt zu keinen Problemen bei der Kommunikation zwischen den einzelnen Bausteinen.
Da die ersten zwei Monate des Projektes mit der Untersuchung von kommerziellen Bussystemen verbracht wurde, aber keines den Anforderungen dieses Projektes genügt hat, standen für die Konzeptionierung, Erprobung und Dokumentation des Systems lediglich vier Monate zur Verfügung.
Die Ergebnisse, die erzielt wurden, sind sehr vielversprechend. Vor allem die Tatsache, dass keine Übertragungsfehler auftreten, ist sehr positiv.
Mit einer Zykluszeit von 5 µs ist das System doppelt so schnell wie gefordert und zehnmal so schnell wie das alte DÜSY – System.
Das neuartige Vcc – Gnd – System arbeitet hervorragend. Es konnten keine Störungen festgestellt werden.
Ausgereift ist das System jedoch noch nicht. So sollten noch die Signalqualität der schnellen seriellen Signale verbessert werden.
Auch der Vergleich der Datenpakete bei der Betriebsfrequenz von 25 MHz muss noch angepasst werden.
Nicht zufriedenstellend waren die Ergebnisse der EMV – Messungen. Weitere Messungen müssen zeigen, ob das Gehäuse verbessert werden kann. Um die Störemission zu verringern, müssen mit großer Wahrscheinlichkeit die Stecker in das Gehäuse verlegt werden und die Kabel EMV-Dicht nach außen geführt werden.
Stolz macht mich, dass das Konzept von SERELECS übernommen und ein ähnliches System im Radioteleskop Effelsberg eingebaut werden wird.
Ich bin dankbar für diese sehr interessante Diplomarbeit, die mir viele theoretische und praktische Einblicke in viele Bereiche der Nachrichtentechnik gegeben hat.

9 Anhänge

9.1 Berechnungen

Zeichen	Bezeichnung	Einheit
α, β	Winkel	rad
f	Frequenz	1/s , Hz
I	Stromstärke	A
P	Leistung	W
R	Widerstand	Ω
U	Spannung	V
U_{BE}, U_{CE}	Spannungen am Transistor: Basis-Emitter Spannung, Collector-Emitter Spannung	V
U_{CC}, V_{cc}	Betriebspannung	V
U_{eff}, $\hat{U}_s$, $\hat{U}_{ss}$	Effektivspannung, Scheitelspannung, Spannung Spitze - Spitze	V

9.1.1 Erwartete Spitzen – Spitzen – Spannung SMA Stecker

In den SMA – Stecker wird eine Leistung von 0 dBm eingespeist. Um die Hysterese des Comperators richtig einstellen zu können muss der zu erwartende Spitzen – Spitzen – Spannungswert bestimmt werden.

Geben:

$\mathrm{P} = 0\mathrm{dBm} = 1\mathrm{mW};\ \ \mathrm{R} = 50\Omega$

Rechnung:

$P = \mathrm{U_{eff}} \cdot \mathrm{I}$ (1)

$I = \frac{U_{eff}}{R}$ (2)

$P = \frac{U_{eff}^2}{R}$ *(2) in (1) eingesetzt* (3)

$U_{eff} = \sqrt{P \cdot R}$ *(3) umgestellt* (4)

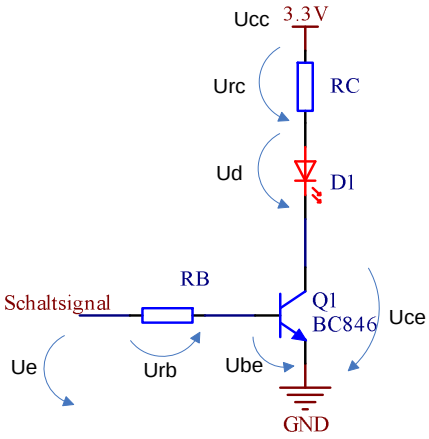
$U_{eff} = \sqrt{P \cdot R} = \sqrt{0{,}001W \cdot 50\Omega} = 0{,}223V$ (4b)

$\widehat{U}_s = \sqrt{2} \cdot U_{eff}$ (5)

$\widehat{U}_{ss} = 2 \cdot \widehat{U}_s = 2 \cdot \sqrt{2} \cdot U_{eff}$ (6)

$\widehat{U}_{ss} = 2 \cdot \widehat{U}_s = 2 \cdot \sqrt{2} \cdot U_{eff} = 2 \cdot \sqrt{2} \cdot 0{,}223V = 0{,}632V$ (6b)

9.1.2 NPN-Transistor als Schalter



In der dargestellten Schaltung (Abb. 9.1) wird ein Transistor als Schalter betrieben. Zur Regelung der Spannung U_d auf 1,9 V mussten die Widerstände R_B und R_C bestimmt werden.

Abbildung 9.1: *Schaltung Transistor als Schalter*

Geben:

$U_{BE} = 0{,}7V;\ U_{CE} = 0{,}2V; U_E = 3{,}3V;\ U_D = 1{,}9V;\ I_D = I_C = 10mA\,;\ \beta = 180$

Rechnung:

$$U_{RB} = U_E - U_{BE} \quad (1)$$

$$U_{RB} = U_E - U_{BE} = 3{,}3V - 0{,}7V = 2{,}4V \quad (1b)$$

$$I_B = \frac{I_C}{\beta} \quad (2)$$

$$I_B = \frac{I_C}{\beta} = \frac{10mA}{180} = 55\mu A \quad (2b)$$

$$R_B = \frac{U_{RB}}{I_B} \quad (3)$$

$$R_B = \frac{U_{RB}}{I_B} = \frac{2{,}4V}{55\mu A} = 43k\Omega \quad (3b)$$

$$U_{RC} = U_{CC} - U_D - U_{CE} \quad (4)$$

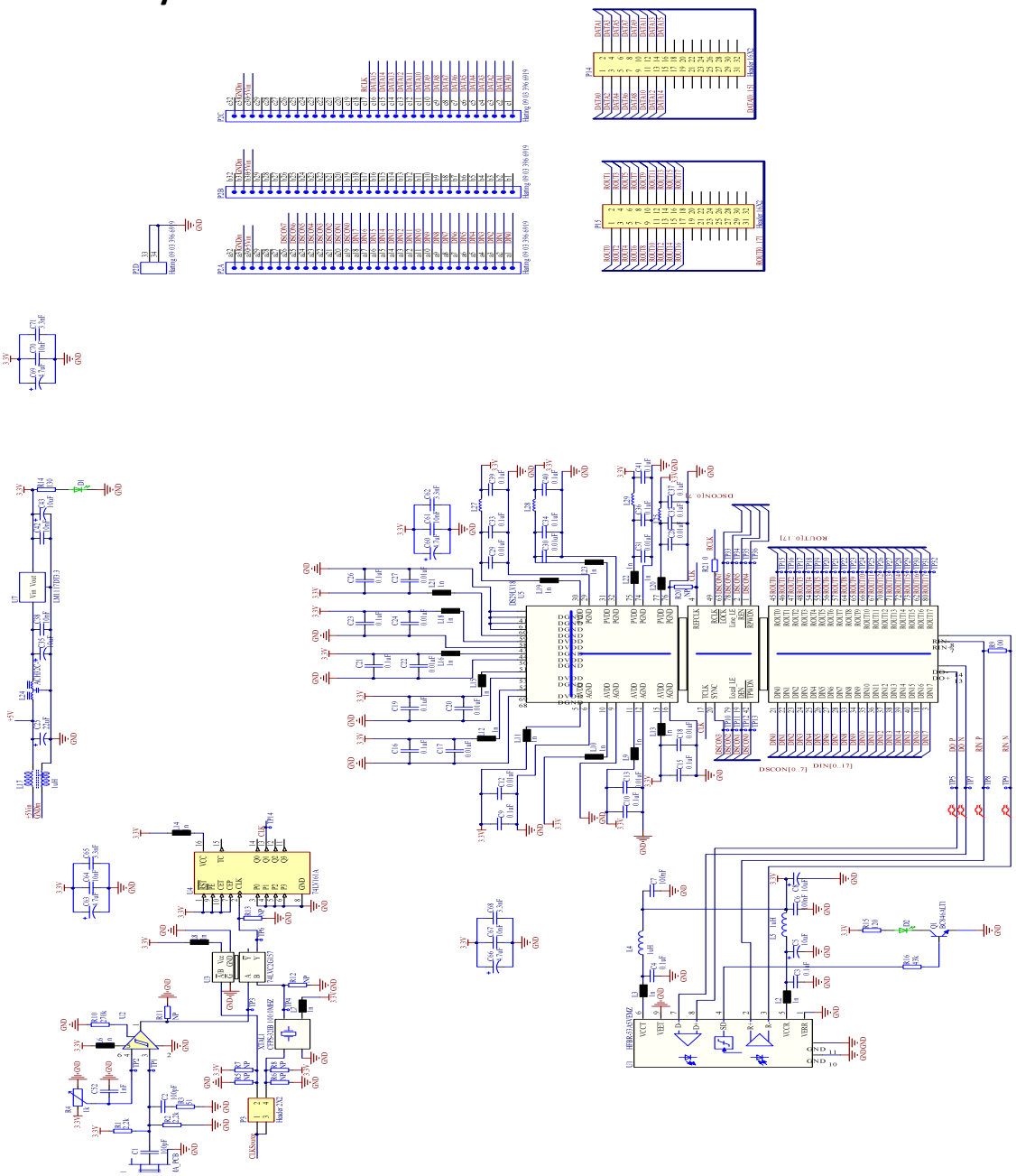
$$U_{RC} = U_{CC} - U_D - U_{CE} = 3{,}3V - 1{,}9V - 0{,}2V = 1{,}2V \quad (4b)$$

$$R_C = \frac{U_{RC}}{I_C} \quad (5)$$

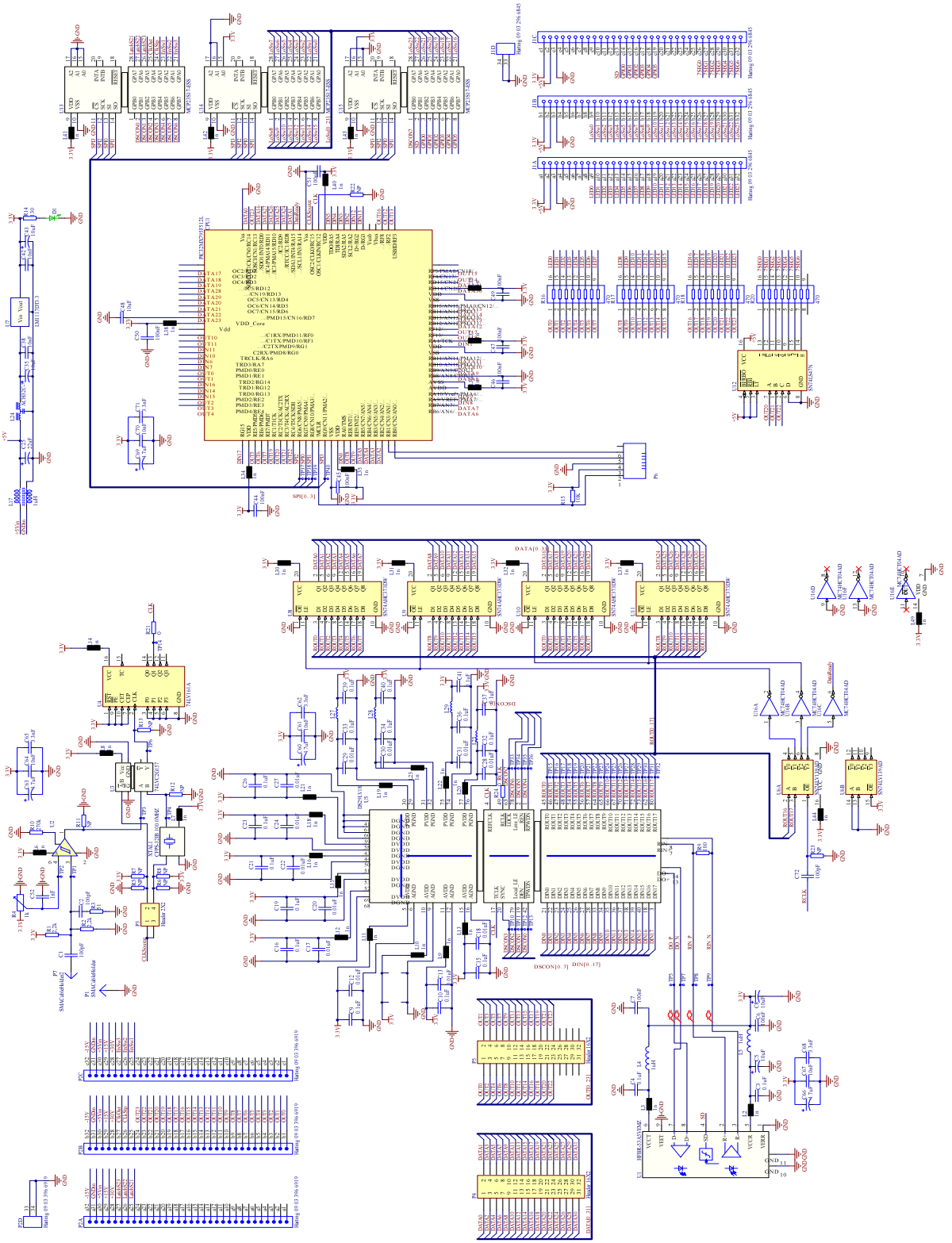
$$R_C = \frac{U_{RC}}{I_C} = \frac{1{,}2V}{10mA} = 120\Omega \quad (5b)$$

9.2 Schaltpläne

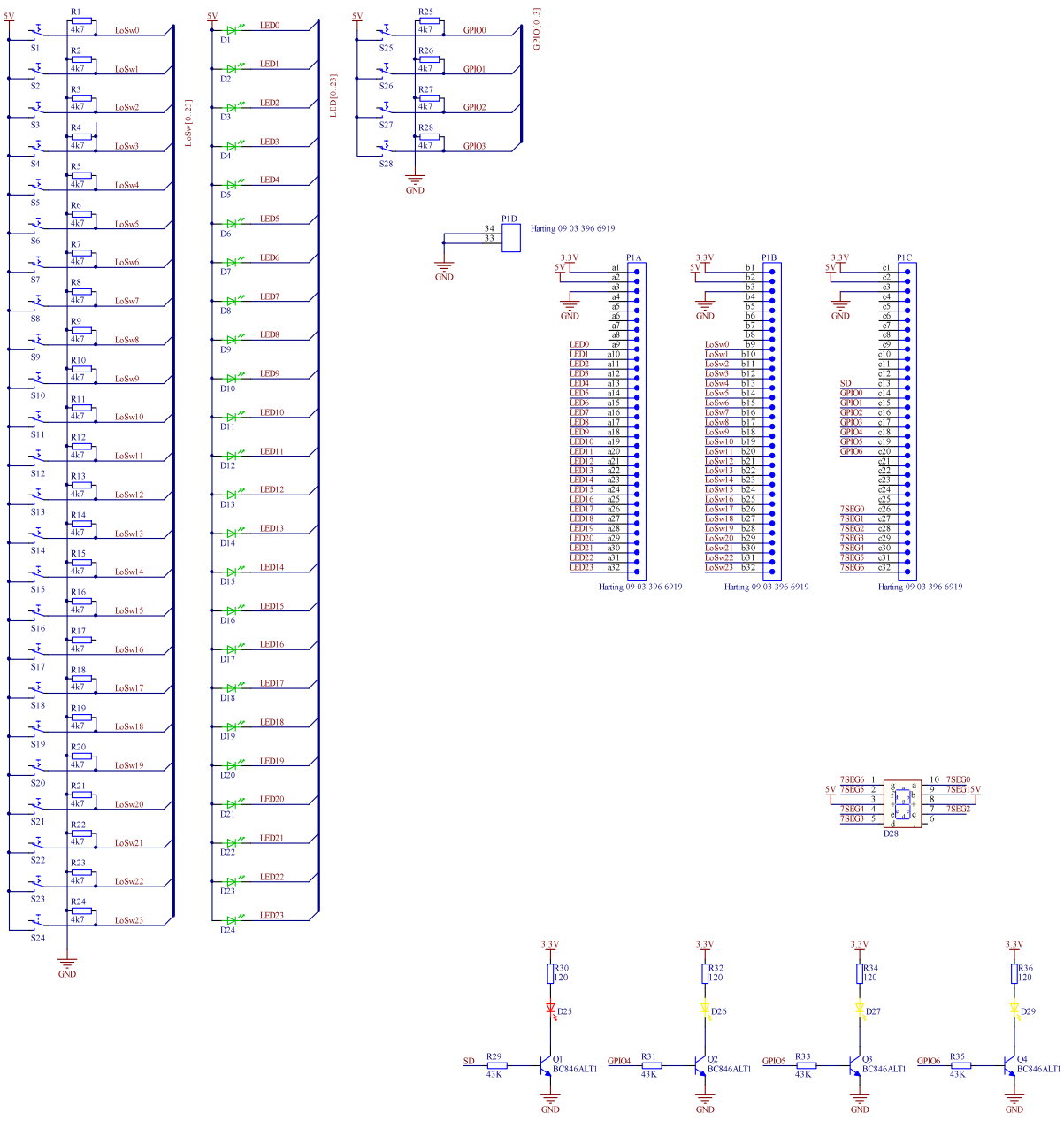
9.2.1 TX – Bussystem



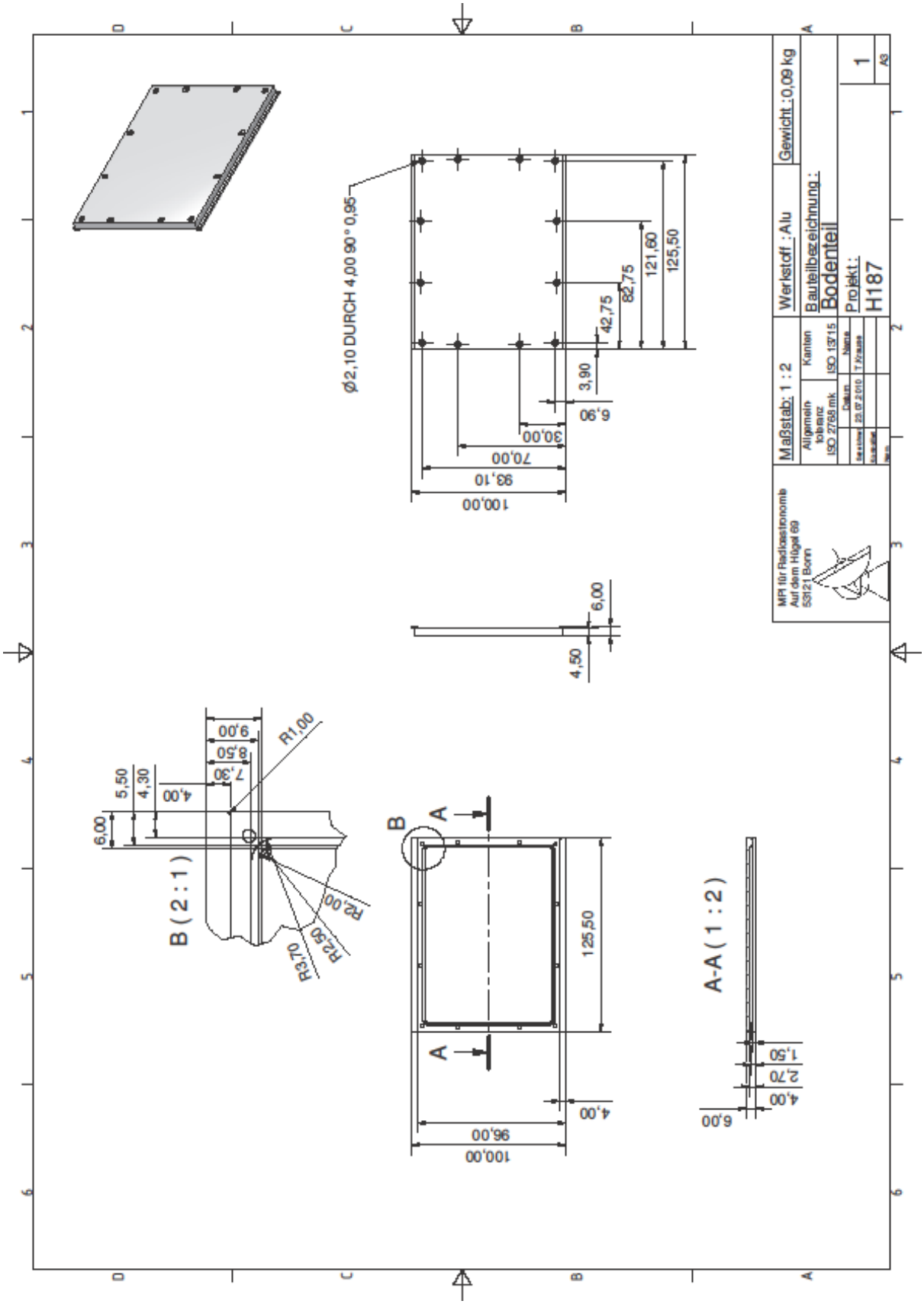
9.2.2 RX Bussystem

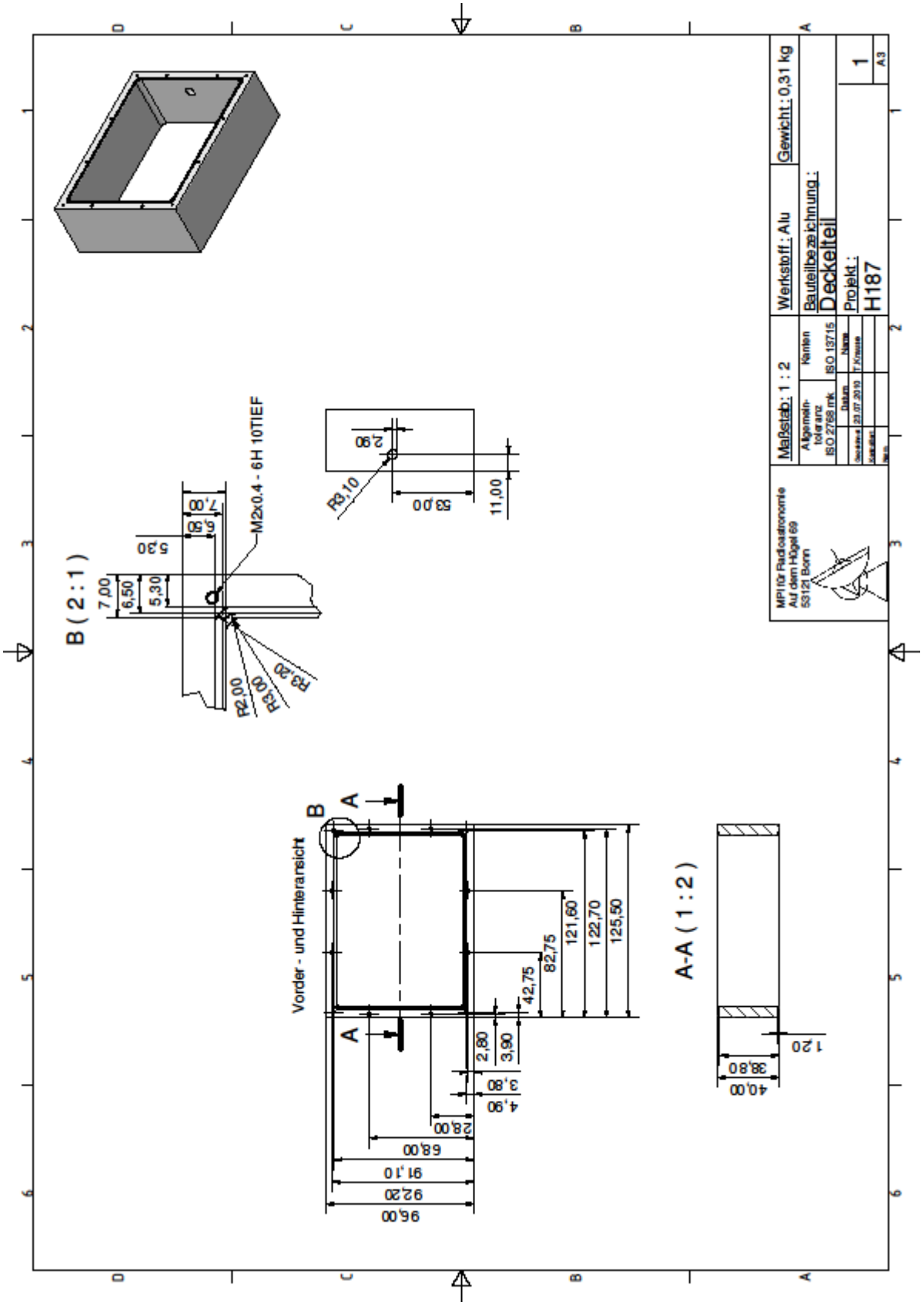


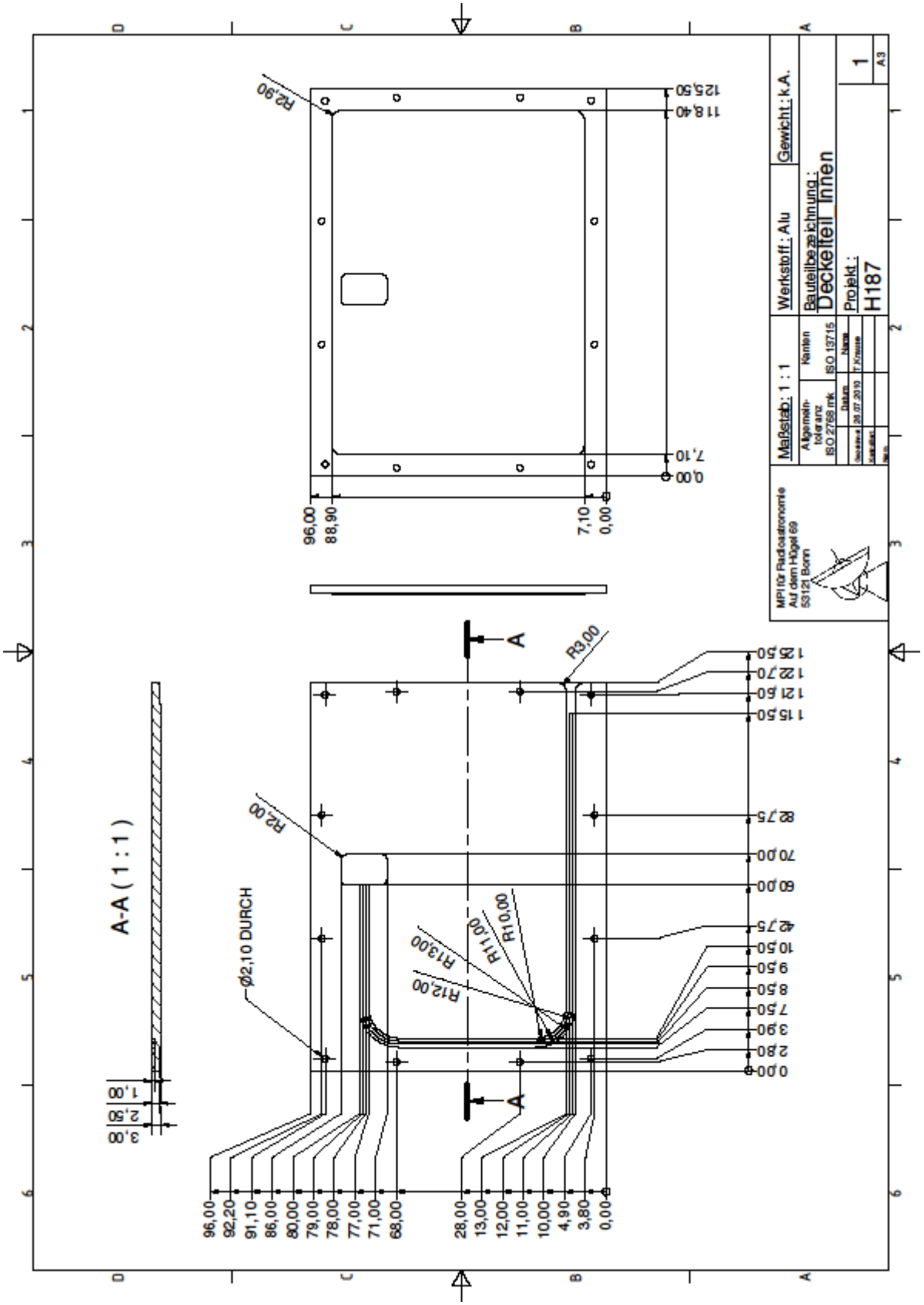
9.2.3 RX Bussystem

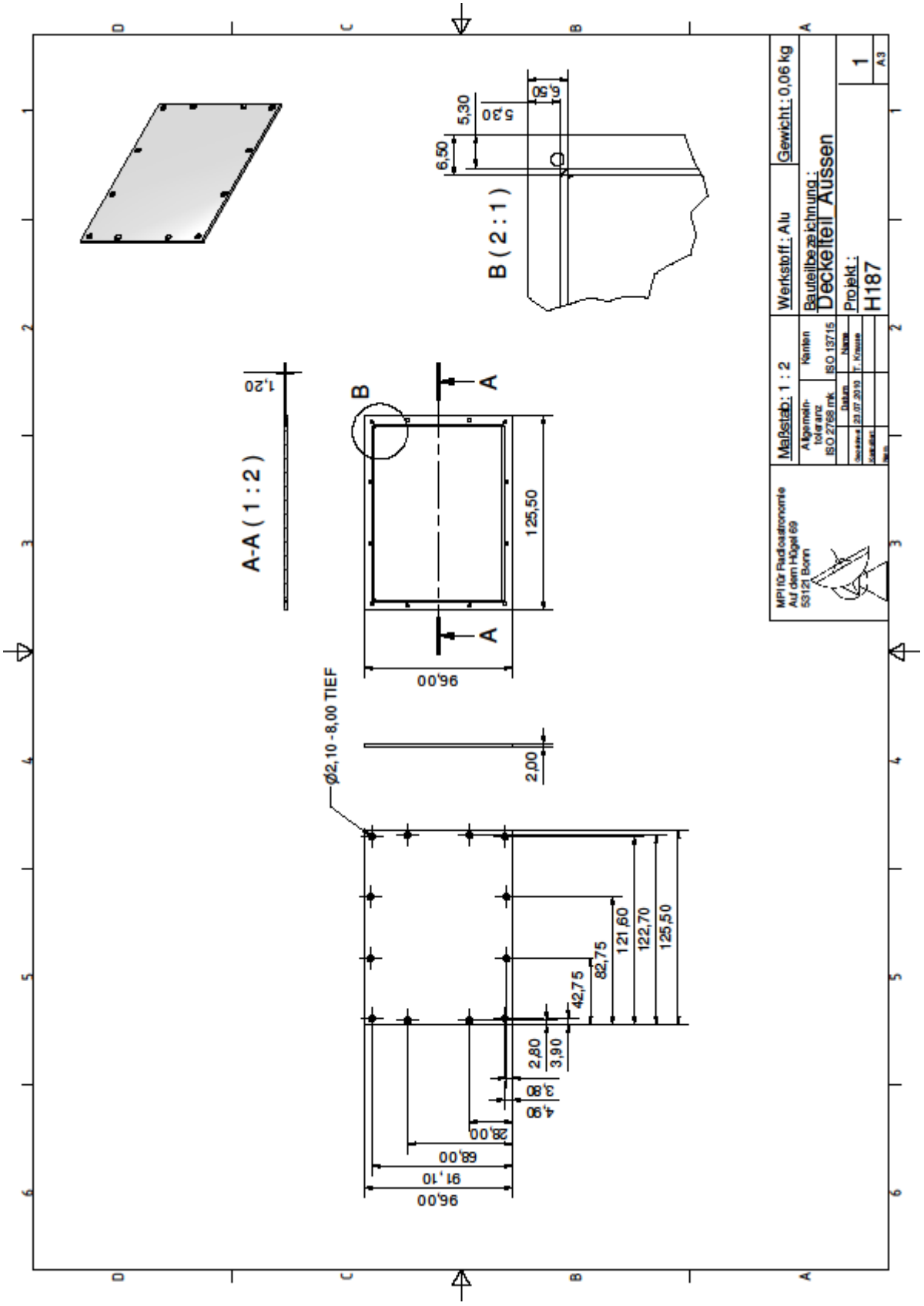


9.3 Gehäusepläne




Gewicht: 0,31 kg
Werkstoff: Alu
Bauteilbezeichnung:
Deckelteil
Projekt:
H187
1
A3
Maßstab: 1 : 2
Kanten
ISO 13715
Allgemeintoleranz
ISO 2768 mK
Datum
23.07.2019
T. Krause
Name
MPI für Radioastronomie
Auf dem Hügel 69
53121 Bonn
B (2 : 1)
M2x0,4 - 6H 10TIEF
7,00
6,50
5,30
5,30
7,80
5,50
R2,00
R3,00
R3,20
R3,10
2,90
53,00
11,00
Vorder - und Hinteransicht
A
B
96,00
92,20
91,10
68,00
28,00
4,90
3,80
2,80
3,90
42,75
82,75
121,60
122,70
125,50
A-A (1 : 2)
40,00
38,80
1,20


Maßstab: 1 : 1
Werkstoff: Alu
Gewicht: k.A.
Bauteilbezeichnung: Deckelteil_Innen
Projekt: H187
MPI für Radioastronomie
Auf dem Hügel 69
53121 Bonn
A-A (1 : 1)
Ø2,10 DURCH
R2,90
R3,00
R2,00
R13,00
R12,00
R11,00
R10,00


B (2 : 1)
5,30
6,50
A-A (1 : 2)
1,20
125,50
96,00
Ø2,10 -8,00 TIEF
2,00
121,60
122,70
125,50
82,75
42,75
2,80
3,90
3,80
4,90
28,00
68,00
91,10
96,00
Maßstab: 1 : 2
Werkstoff: Alu
Gewicht: 0,06 kg
Bauteilbezeichnung:
Deckelteil_Aussen
Projekt:
H187
MPI für Radioastronomie
Auf dem Hügel 69
53121 Bonn

9.4 VHDL-Code

9.4.1 Top

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

entity TOP is

      PORT (
            Ext_clk_50MHz           : in STD_LOGIC;
            Data                    : inout STD_LOGIC_VECTOR (17 downto 0);
            Din                     : inout STD_LOGIC_VECTOR (17 downto 0);
            Sync                    : inout STD_LOGIC;
            Local_Le                : inout STD_LOGIC;
            Den                     : inout STD_LOGIC;
            Tpwdn                   : inout STD_LOGIC;
            Lock                    : inout STD_LOGIC;
            Line_Le                 : inout STD_LOGIC;
            Ren                     : inout STD_LOGIC;
            Rpwdn                   : inout STD_LOGIC;
            Lcd_dat                 : inout STD_LOGIC_VECTOR (3 downto 0);
            Lcd_rw                  : out   STD_LOGIC;
            Lcd_e                   : out   STD_LOGIC;
            Lcd_rs                  : out   STD_LOGIC;
            Taster                  : in    STD_LOGIC;
            Reset                   : in    STD_LOGIC
      );
end TOP;

architecture Behavioral of TOP is


------------------------------------------------------------
--Verwendete interne Signale ohne Schnittstelle nach Aussen
------------------------------------------------------------
signal dev_clock_25kHz        : STD_LOGIC;
signal dev_clock_5MHz         : STD_LOGIC;
signal dev_clock_2_5MHz       : STD_LOGIC;
signal dev_clock_25MHz        : STD_LOGIC;
signal noflowbits1            : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits2            : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits3            : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits4            : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits5            : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits6            : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits7            : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits8            : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits9            : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits10           : STD_LOGIC_VECTOR (3 downto 0);
signal noflowbits11           : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits1           : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits2           : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits3           : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits4           : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits5           : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits6           : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits7           : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits8           : STD_LOGIC_VECTOR (3 downto 0);
```

```
signal nofhighbits9          : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits10         : STD_LOGIC_VECTOR (3 downto 0);
signal nofhighbits11         : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits1           : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits2           : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits3           : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits4           : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits5           : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits6           : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits7           : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits8           : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits9           : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits10          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdlowbits11          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits1          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits2          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits3          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits4          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits5          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits6          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits7          : STD_LOGIC_VECTOR (3 downto 0);
signal nsd1                  : integer;
signal nsd10                 : integer;
signal nsd100                : integer;
signal nsd1000               : integer;
signal nsd10000              : integer;
signal nsd100000             : integer;
signal nsd1000000            : integer;
signal noF1                  : integer;
signal noF10                 : integer;
signal noF100                : integer;
signal noF1000               : integer;
signal noF10000              : integer;
signal noF100000             : integer;
signal noF1000000            : integer;
signal noF10000000           : integer;
signal noF100000000          : integer;
signal noF1000000000         : integer;
signal noF10000000000        : integer;
signal nsdhighbits8          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits9          : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits10         : STD_LOGIC_VECTOR (3 downto 0);
signal nsdhighbits11         : STD_LOGIC_VECTOR (3 downto 0);
signal count                 : STD_LOGIC_VECTOR (14 downto 0);
signal nsd10000000           : integer;
signal nsd100000000          : integer;
signal nsd1000000000         : integer;
signal nsd10000000000        : integer;
signal status                : STD_LOGIC;
signal din0                  : STD_LOGIC_VECTOR (3 downto 0);
signal din1                  : STD_LOGIC_VECTOR (3 downto 0);
signal din2                  : STD_LOGIC_VECTOR (3 downto 0);
signal din3                  : STD_LOGIC_VECTOR (3 downto 0);
signal din4                  : STD_LOGIC_VECTOR (3 downto 0);
signal din5                  : STD_LOGIC_VECTOR (3 downto 0);
signal din6                  : STD_LOGIC_VECTOR (3 downto 0);
signal din7                  : STD_LOGIC_VECTOR (3 downto 0);
signal din8                  : STD_LOGIC_VECTOR (3 downto 0);
signal din9                  : STD_LOGIC_VECTOR (3 downto 0);
signal din10                 : STD_LOGIC_VECTOR (3 downto 0);
signal din11                 : STD_LOGIC_VECTOR (3 downto 0);
signal din12                 : STD_LOGIC_VECTOR (3 downto 0);
signal din13                 : STD_LOGIC_VECTOR (3 downto 0);
```

```
signal din14                    : STD_LOGIC_VECTOR (3 downto 0);
signal din15                    : STD_LOGIC_VECTOR (3 downto 0);
signal data0                    : STD_LOGIC_VECTOR (3 downto 0);
signal data1                    : STD_LOGIC_VECTOR (3 downto 0);
signal data2                    : STD_LOGIC_VECTOR (3 downto 0);
signal data3                    : STD_LOGIC_VECTOR (3 downto 0);
signal data4                    : STD_LOGIC_VECTOR (3 downto 0);
signal data5                    : STD_LOGIC_VECTOR (3 downto 0);
signal data6                    : STD_LOGIC_VECTOR (3 downto 0);
signal data7                    : STD_LOGIC_VECTOR (3 downto 0);
signal data8                    : STD_LOGIC_VECTOR (3 downto 0);
signal data9                    : STD_LOGIC_VECTOR (3 downto 0);
signal data10                   : STD_LOGIC_VECTOR (3 downto 0);
signal data11                   : STD_LOGIC_VECTOR (3 downto 0);
signal data12                   : STD_LOGIC_VECTOR (3 downto 0);
signal data13                   : STD_LOGIC_VECTOR (3 downto 0);
signal data14                   : STD_LOGIC_VECTOR (3 downto 0);
signal data15                   : STD_LOGIC_VECTOR (3 downto 0);
signal random_out               : STD_LOGIC_VECTOR (13 downto 0);
signal Data_received_decoded    : STD_LOGIC_VECTOR (13 downto 0);

------------------------------------------------------------------------------

COMPONENT CODER is
 PORT(
      Ext_clk_50MHz             : in STD_LOGIC;
      random_out                : in STD_LOGIC_VECTOR (13 downto 0);
      random_out_coded          : inout STD_LOGIC_VECTOR (17 downto 0)
 );
end COMPONENT;

COMPONENT COUNTER_LCD is
 PORT(
      dev_clock_25kHz           : in STD_LOGIC;
      count                     : inout STD_LOGIC_VECTOR (14 downto 0)
);
end COMPONENT;

COMPONENT DECODER is
 PORT(
      Ext_clk_50MHz             : in STD_LOGIC;
      data_received_decoded     : inout STD_LOGIC_VECTOR (13 downto 0);
      Data_received_coded       : inout STD_LOGIC_VECTOR (17 downto 0)
 );
end COMPONENT;

COMPONENT DIVIDER_25kHz is
 PORT(
      Ext_clk_50MHz             : in STD_LOGIC;
      dev_clock_25kHz           : inout STD_LOGIC
 );
end COMPONENT;

COMPONENT DIVIDER_2_5MHz is
 PORT(
      Ext_clk_50MHz             : in STD_LOGIC;
      dev_clock_2_5MHz          : inout STD_LOGIC
 );
end COMPONENT;

COMPONENT DIVIDER_5MHz is
 PORT(
      Ext_clk_50MHz             : in STD_LOGIC;
```

```
      dev_clock_5MHz          : inout STD_LOGIC
 );
end COMPONENT;

COMPONENT DIVIDER_25MHz is
 PORT (
       Ext_clk_50MHz          : in STD_LOGIC;
       dev_clock_25MHz        : inout STD_LOGIC
 );
end COMPONENT;

COMPONENT COMPARISON is
 PORT(
      dev_clock_2_5MHz        : in STD_LOGIC;
      dev_clock_5MHz          : in STD_LOGIC;
      random_out              : in STD_LOGIC_VECTOR(13 downto 0);
      nsd1                    : inout integer;
      nsd10                   : inout integer;
      nsd100                  : inout integer;
      nsd1000                 : inout integer;
      nsd10000                : inout integer;
      nsd100000               : inout integer;
      nsd1000000              : inout integer;
      nsd10000000             : inout integer;
      nsd100000000            : inout integer;
      nsd1000000000           : inout integer;
      nsd10000000000          : inout integer;
      nof1                    : inout integer;
      nof10                   : inout integer;
      nof100                  : inout integer;
      nof1000                 : inout integer;
      nof10000                : inout integer;
      nof100000               : inout integer;
      nof1000000              : inout integer;
      nof10000000             : inout integer;
      nof100000000            : inout integer;
      nof1000000000           : inout integer;
      nof10000000000          : inout integer;
      data_received_decoded   : in STD_LOGIC_VECTOR(13 downto 0);
      Reset                   : in STD_LOGIC
 );
end COMPONENT;

COMPONENT INITSERIALIZER is
 PORT(
      Sync                    : inout     STD_LOGIC;
      Local_Le                : inout     STD_LOGIC;
      Den                     : inout     STD_LOGIC;
      Tpwdn                   : inout     STD_LOGIC;
      Lock                    : inout     STD_LOGIC;
      Line_Le                 : inout     STD_LOGIC;
      Ren                     : inout     STD_LOGIC;
      Rpwdn                   : inout     STD_LOGIC;
      dev_clock_25kHz         : in STD_LOGIC
      );
end COMPONENT;

COMPONENT MAC_LCD is
PORT (
      dev_clock_25MHz         : in STD_LOGIC;
      status                  : in STD_LOGIC;
      lcd_dat                 : out STD_LOGIC_VECTOR (3 downto 0);
      lcd_e                   : out STD_LOGIC;
```

```
lcd_rw                  : out STD_LOGIC;
count                   : in STD_LOGIC_VECTOR (14 downto 0);
lcd_rs                  : out STD_LOGIC;
noflowbits1             : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits2             : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits3             : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits4             : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits5             : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits6             : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits7             : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits8             : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits9             : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits10            : in STD_LOGIC_VECTOR (3 downto 0);
noflowbits11            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits1            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits2            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits3            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits4            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits5            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits6            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits7            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits8            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits9            : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits10           : in STD_LOGIC_VECTOR (3 downto 0);
nofhighbits11           : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits1             : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits2             : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits3             : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits4             : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits5             : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits6             : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits7             : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits8             : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits9             : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits10            : in STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits11            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits1            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits2            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits3            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits4            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits5            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits6            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits7            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits8            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits9            : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits10           : in STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits11           : in STD_LOGIC_VECTOR (3 downto 0);
din0                    : in STD_LOGIC_VECTOR (3 downto 0);
din1                    : in  STD_LOGIC_VECTOR (3 downto 0);
din2                    : in  STD_LOGIC_VECTOR (3 downto 0);
din3                    : in  STD_LOGIC_VECTOR (3 downto 0);
din4                    : in  STD_LOGIC_VECTOR (3 downto 0);
din5                    : in  STD_LOGIC_VECTOR (3 downto 0);
din6                    : in  STD_LOGIC_VECTOR (3 downto 0);
din7                    : in  STD_LOGIC_VECTOR (3 downto 0);
din8                    : in  STD_LOGIC_VECTOR (3 downto 0);
din9                    : in  STD_LOGIC_VECTOR (3 downto 0);
din10                   : in  STD_LOGIC_VECTOR (3 downto 0);
din11                   : in  STD_LOGIC_VECTOR (3 downto 0);
din12                   : in  STD_LOGIC_VECTOR (3 downto 0);
din13                   : in  STD_LOGIC_VECTOR (3 downto 0);
din14                   : in  STD_LOGIC_VECTOR (3 downto 0);
din15                   : in  STD_LOGIC_VECTOR (3 downto 0);
```

```
      data0                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data1                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data2                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data3                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data4                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data5                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data6                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data7                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data8                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data9                     : in  STD_LOGIC_VECTOR (3 downto 0);
      data10                    : in  STD_LOGIC_VECTOR (3 downto 0);
      data11                    : in  STD_LOGIC_VECTOR (3 downto 0);
      data12                    : in  STD_LOGIC_VECTOR (3 downto 0);
      data13                    : in  STD_LOGIC_VECTOR (3 downto 0);
      data14                    : in  STD_LOGIC_VECTOR (3 downto 0);
      data15                    : in  STD_LOGIC_VECTOR (3 downto 0)
      );
end COMPONENT;

COMPONENT RANDOMGENERATER is
 PORT (
      dev_clock_2_5MHz          : in STD_LOGIC;
      random_out                : inout STD_LOGIC_VECTOR (13 downto 0)
      );
end COMPONENT;

COMPONENT SET_LCD is
 PORT(
      dev_clock_5MHz            : in STD_LOGIC;
      Data                      : inout STD_LOGIC_VECTOR (17 downto 0);
      Din                       : inout STD_LOGIC_VECTOR (17 downto 0);
      noflowbits1               : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits2               : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits3               : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits4               : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits5               : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits6               : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits7               : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits8               : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits9               : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits10              : out STD_LOGIC_VECTOR (3 downto 0);
      noflowbits11              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits1              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits2              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits3              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits4              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits5              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits6              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits7              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits8              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits9              : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits10             : out STD_LOGIC_VECTOR (3 downto 0);
      nofhighbits11             : out STD_LOGIC_VECTOR (3 downto 0);
      nsd1                      : in integer;
      nsd10                     : in integer;
      nsd100                    : in integer;
      nsd1000                   : in integer;
      nsd10000                  : in integer;
      nsd100000                 : in integer;
      nsd1000000                : in integer;
      nsd10000000               : in integer;
      nsd100000000              : in integer;
      nsd1000000000             : in integer;
```

```
nsd10000000000          : in integer;
nof1                    : in integer;
nof10                   : in integer;
nof100                  : in integer;
nof1000                 : in integer;
nof10000                : in integer;
nof100000               : in integer;
nof1000000              : in integer;
nof10000000             : in integer;
nof100000000            : in integer;
nof1000000000           : in integer;
nof10000000000          : in integer;
nsdlowbits1             : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits2             : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits3             : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits4             : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits5             : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits6             : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits7             : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits8             : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits9             : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits10            : out STD_LOGIC_VECTOR (3 downto 0);
nsdlowbits11            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits1            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits2            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits3            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits4            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits5            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits6            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits7            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits8            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits9            : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits10           : out STD_LOGIC_VECTOR (3 downto 0);
nsdhighbits11           : out STD_LOGIC_VECTOR (3 downto 0);
din0                    : out STD_LOGIC_VECTOR (3 downto 0);
din1                    : out STD_LOGIC_VECTOR (3 downto 0);
din2                    : out STD_LOGIC_VECTOR (3 downto 0);
din3                    : out STD_LOGIC_VECTOR (3 downto 0);
din4                    : out STD_LOGIC_VECTOR (3 downto 0);
din5                    : out STD_LOGIC_VECTOR (3 downto 0);
din6                    : out STD_LOGIC_VECTOR (3 downto 0);
din7                    : out STD_LOGIC_VECTOR (3 downto 0);
din8                    : out STD_LOGIC_VECTOR (3 downto 0);
din9                    : out STD_LOGIC_VECTOR (3 downto 0);
din10                   : out STD_LOGIC_VECTOR (3 downto 0);
din11                   : out STD_LOGIC_VECTOR (3 downto 0);
din12                   : out STD_LOGIC_VECTOR (3 downto 0);
din13                   : out STD_LOGIC_VECTOR (3 downto 0);
din14                   : out STD_LOGIC_VECTOR (3 downto 0);
din15                   : out STD_LOGIC_VECTOR (3 downto 0);
data0                   : out STD_LOGIC_VECTOR (3 downto 0);
data1                   : out STD_LOGIC_VECTOR (3 downto 0);
data2                   : out STD_LOGIC_VECTOR (3 downto 0);
data3                   : out STD_LOGIC_VECTOR (3 downto 0);
data4                   : out STD_LOGIC_VECTOR (3 downto 0);
data5                   : out STD_LOGIC_VECTOR (3 downto 0);
data6                   : out STD_LOGIC_VECTOR (3 downto 0);
data7                   : out STD_LOGIC_VECTOR (3 downto 0);
data8                   : out STD_LOGIC_VECTOR (3 downto 0);
data9                   : out STD_LOGIC_VECTOR (3 downto 0);
data10                  : out STD_LOGIC_VECTOR (3 downto 0);
data11                  : out STD_LOGIC_VECTOR (3 downto 0);
data12                  : out STD_LOGIC_VECTOR (3 downto 0);
```

```
      data13                  : out STD_LOGIC_VECTOR (3 downto 0);
      data14                  : out STD_LOGIC_VECTOR (3 downto 0);
      data15                  : out STD_LOGIC_VECTOR (3 downto 0)
      );
end COMPONENT;

COMPONENT STATUS_LCD is
 PORT (
      taster                  : in  STD_LOGIC;
      status                  : inout STD_LOGIC
      );
end COMPONENT;


begin

INST_INITSERIALIZER : INITSERIALIZER PORT MAP
      (
      Sync                    => Sync,
      Local_Le                => Local_Le,
      Den                     => Den,
      Tpwdn                   => Tpwdn,
      Lock                    => Lock,
      Line_Le                 => Line_Le,
      Ren                     => Ren,
      Rpwdn                   => Rpwdn,
      dev_clock_25kHz         => dev_clock_25kHz
      );

INST_COMPARISON : COMPARISON PORT MAP
      (
      dev_clock_5MHz          => dev_clock_5MHz,
      dev_clock_2_5MHz        => dev_clock_2_5MHz,
      random_out              => random_out,
      nsd1                    => nsd1,
      nsd10                   => nsd10,
      nsd100                  => nsd100,
      nsd1000                 => nsd1000,
      nsd10000                => nsd10000,
      nsd100000               => nsd100000,
      nsd1000000              => nsd1000000,
      nsd10000000             => nsd10000000,
      nsd100000000            => nsd100000000,
      nsd1000000000           => nsd1000000000,
      nsd10000000000          => nsd10000000000,
      nof1                    => nof1,
      nof10                   => nof10,
      nof100                  => nof100,
      nof1000                 => nof1000,
      nof10000                => nof10000,
      nof100000               => nof100000,
      nof1000000              => nof1000000,
      nof10000000             => nof10000000,
      nof100000000            => nof100000000,
      nof1000000000           => nof1000000000,
      nof10000000000          => nof10000000000,
      data_received_decoded   => data_received_decoded,
      Reset                   => Reset
      );

INST_COUNTER_LCD : COUNTER_LCD PORT MAP
      (
      dev_clock_25kHz         => dev_clock_25kHZ,
```

```
      count                   => count
      );


INST_CODER : CODER PORT MAP
      (
      Ext_clk_50MHz           => Ext_clk_50MHz,
      random_out              => random_out,
      random_out_coded        => Din
      );

INST_DECODER : DECODER PORT MAP
      (
      Ext_clk_50MHz           => Ext_clk_50MHz,
      data_received_decoded   => data_received_decoded,
      Data_received_coded     => Data
      );

INST_DIVIDER_25kHz : DIVIDER_25kHz PORT MAP
 (
      Ext_clk_50MHz           => Ext_clk_50MHz,
      dev_clock_25kHz         => dev_clock_25kHz
 );

 INST_DIVIDER_2_5MHz : DIVIDER_2_5MHz PORT MAP
      (
      Ext_clk_50MHz           => Ext_clk_50MHz,
      dev_clock_2_5MHz        => dev_clock_2_5MHz
      );

 INST_DIVIDER_5MHz : DIVIDER_5MHz PORT MAP
      (
      Ext_clk_50MHz           => Ext_clk_50MHz,
      dev_clock_5MHz          => dev_clock_5MHz
      );

INST_DIVIDER_25MHz : DIVIDER_25MHz PORT MAP
       (
      Ext_clk_50MHz           => Ext_clk_50MHz,
      dev_clock_25MHz         => dev_clock_25MHz
      );

INST_RANDOMGENERATER : RANDOMGENERATER PORT MAP
      (
      dev_clock_2_5MHz        => dev_clock_2_5MHz,
      random_out              => random_out
      );

INST_Status_LCD : Status_LCD PORT MAP
      (
      taster                  => taster,
      status                  => status
 );

INST_SET_LCD : SET_LCD PORT MAP
      (
      dev_clock_5MHz          => dev_clock_5MHz,
      noflowbits1             => noflowbits1,
      noflowbits2             => noflowbits2,
      noflowbits3             => noflowbits3,
      noflowbits4             => noflowbits4,
      noflowbits5             => noflowbits5,
      noflowbits6             => noflowbits6,
```

```
noflowbits7             => noflowbits7,
noflowbits8             => noflowbits8,
noflowbits9             => noflowbits9,
noflowbits10            => noflowbits10,
noflowbits11            => noflowbits11,
nofhighbits1            => nofhighbits1,
nofhighbits2            => nofhighbits2,
nofhighbits3            => nofhighbits3,
nofhighbits4            => nofhighbits4,
nofhighbits5            => nofhighbits5,
nofhighbits6            => nofhighbits6,
nofhighbits7            => nofhighbits7,
nofhighbits8            => nofhighbits8,
nofhighbits9            => nofhighbits9,
nofhighbits10           => nofhighbits10,
nofhighbits11           => nofhighbits11,
nsdlowbits1             => nsdlowbits1,
nsdlowbits2             => nsdlowbits2,
nsdlowbits3             => nsdlowbits3,
nsdlowbits4             => nsdlowbits4,
nsdlowbits5             => nsdlowbits5,
nsdlowbits6             => nsdlowbits6,
nsdlowbits7             => nsdlowbits7,
nsdlowbits8             => nsdlowbits8,
nsdlowbits9             => nsdlowbits9,
nsdlowbits10            => nsdlowbits10,
nsdlowbits11            => nsdlowbits11,
nsdhighbits1            => nsdhighbits1,
nsdhighbits2            => nsdhighbits2,
nsdhighbits3            => nsdhighbits3,
nsdhighbits4            => nsdhighbits4,
nsdhighbits5            => nsdhighbits5,
nsdhighbits6            => nsdhighbits6,
nsdhighbits7            => nsdhighbits7,
nsdhighbits8            => nsdhighbits8,
nsdhighbits9            => nsdhighbits9,
nsdhighbits10           => nsdhighbits10,
nsdhighbits11           => nsdhighbits11,
nsd1                    => nsd1,
nsd10                   => nsd10,
nsd100                  => nsd100,
nsd1000                 => nsd1000,
nsd10000                => nsd10000,
nsd100000               => nsd100000,
nsd1000000              => nsd1000000,
nsd10000000             => nsd10000000,
nsd100000000            => nsd100000000,
nsd1000000000           => nsd1000000000,
nsd10000000000          => nsd10000000000,
nof1                    => nof1,
nof10                   => nof10,
nof100                  => nof100,
nof1000                 => nof1000,
nof10000                => nof10000,
nof100000               => nof100000,
nof1000000              => nof1000000,
nof10000000             => nof10000000,
nof100000000            => nof100000000,
nof1000000000           => nof1000000000,
nof10000000000          => nof10000000000,
din0                    => din0,
din1                    => din1,
din2                    => din2,
```

```
      din3                    => din3,
      din4                    => din4,
      din5                    => din5,
      din6                    => din6,
      din7                    => din7,
      din8                    => din8,
      din9                    => din9,
      din10                   => din10,
      din11                   => din11,
      din12                   => din12,
      din13                   => din13,
      din14                   => din14,
      din15                   => din15,
      data0                   => data0,
      data1                   => data1,
      data2                   => data2,
      data3                   => data3,
      data4                   => data4,
      data5                   => data5,
      data6                   => data6,
      data7                   => data7,
      data8                   => data8,
      data9                   => data9,
      data10                  => data10,
      data11                  => data11,
      data12                  => data12,
      data13                  => data13,
      data14                  => data14,
      data15                  => data15,
      Data                    => Data,
      Din                     => Din
      );

INST_MAC_LCD :  MAC_LCD PORT MAP
      (
      dev_clock_25MHz         => dev_clock_25MHz,
      lcd_dat                 => lcd_dat,
      lcd_e                   => lcd_e,
      lcd_rw                  => lcd_rw,
      status                  => status,
      count                   => count,
      lcd_rs                  => lcd_rs,
      noflowbits1             => noflowbits1,
      noflowbits2             => noflowbits2,
      noflowbits3             => noflowbits3,
      noflowbits4             => noflowbits4,
      noflowbits5             => noflowbits5,
      noflowbits6             => noflowbits6,
      noflowbits7             => noflowbits7,
      noflowbits8             => noflowbits8,
      noflowbits9             => noflowbits9,
      noflowbits10            => noflowbits10,
      noflowbits11            => noflowbits11,
      nofhighbits1            => nofhighbits1,
      nofhighbits2            => nofhighbits2,
      nofhighbits3            => nofhighbits3,
      nofhighbits4            => nofhighbits4,
      nofhighbits5            => nofhighbits5,
      nofhighbits6            => nofhighbits6,
      nofhighbits7            => nofhighbits7,
      nofhighbits8            => nofhighbits8,
      nofhighbits9            => nofhighbits9,
      nofhighbits10           => nofhighbits10,
```

```
      nofhighbits11           => nofhighbits11,
      nsdlowbits1             => nsdlowbits1,
      nsdlowbits2             => nsdlowbits2,
      nsdlowbits3             => nsdlowbits3,
      nsdlowbits4             => nsdlowbits4,
      nsdlowbits5             => nsdlowbits5,
      nsdlowbits6             => nsdlowbits6,
      nsdlowbits7             => nsdlowbits7,
      nsdlowbits8             => nsdlowbits8,
      nsdlowbits9             => nsdlowbits9,
      nsdlowbits10            => nsdlowbits10,
      nsdlowbits11            => nsdlowbits11,
      nsdhighbits1            => nsdhighbits1,
      nsdhighbits2            => nsdhighbits2,
      nsdhighbits3            => nsdhighbits3,
      nsdhighbits4            => nsdhighbits4,
      nsdhighbits5            => nsdhighbits5,
      nsdhighbits6            => nsdhighbits6,
      nsdhighbits7            => nsdhighbits7,
      nsdhighbits8            => nsdhighbits8,
      nsdhighbits9            => nsdhighbits9,
      nsdhighbits10           => nsdhighbits10,
      nsdhighbits11           => nsdhighbits11,
      din0                    => din0,
      din1                    => din1,
      din2                    => din2,
      din3                    => din3,
      din4                    => din4,
      din5                    => din5,
      din6                    => din6,
      din7                    => din7,
      din8                    => din8,
      din9                    => din9,
      din10                   => din10,
      din11                   => din11,
      din12                   => din12,
      din13                   => din13,
      din14                   => din14,
      din15                   => din15,
      data0                   => data0,
      data1                   => data1,
      data2                   => data2,
      data3                   => data3,
      data4                   => data4,
      data5                   => data5,
      data6                   => data6,
      data7                   => data7,
      data8                   => data8,
      data9                   => data9,
      data10                  => data10,
      data11                  => data11,
      data12                  => data12,
      data13                  => data13,
      data14                  => data14,
      data15                  => data15
      );

end Behavioral;
```

9.4.2 Divider 25 kHz

```
----------------------------------------------------------------------------
-- Teilt Takt von 50 MHz auf 25 KHz
-- Takt wird bei Inizialisierung des LCD benötigt
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity DIVIDER_25kHz is
 Port (
            Ext_clk_50MHz           : in STD_LOGIC;
            dev_clock_25kHz         : inout STD_LOGIC --25 kHz
 );
end DIVIDER_25kHz;

architecture Behavioral of DIVIDER_25kHz is
signal divide_counter : integer := 0;
begin

 Process(Ext_clk_50MHz)
 begin
 if (rising_edge (Ext_clk_50MHz)) then
      if (divide_counter /= 1000) then
            divide_counter <= divide_counter + 1;
      else
            divide_counter <= 0;
            dev_clock_25kHz <= not (dev_clock_25kHz);
      end if;
 end if;
 end process;

end Behavioral;
```

9.4.3 Divider 2,5 MHz

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



entity DIVIDER_2_5MHz is
 Port (
            Ext_clk_50MHz           : in STD_LOGIC;
            dev_clock_2_5MHz        : inout STD_LOGIC
 );
end DIVIDER_2_5MHz;

architecture Behavioral of DIVIDER_2_5MHz is

signal divide_counter : integer := 0;
begin

 Process(Ext_clk_50MHz)
 begin
 if (rising_edge (Ext_clk_50MHz)) then
      if (divide_counter /= 10) then
            divide_counter <= divide_counter + 1;
      else
            divide_counter <= 0;
            dev_clock_2_5MHz <= not (dev_clock_2_5MHz);
      end if;
 end if;
 end process;



end Behavioral;
```

9.4.4 Divider 5 MHz

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


entity DIVIDER_5MHz is
 Port (
            Ext_clk_50MHz     : in STD_LOGIC;
            dev_clock_5MHz    : inout STD_LOGIC
 );
end DIVIDER_5MHz;


architecture Behavioral of DIVIDER_5MHz is

signal divide_counter : integer := 0;
begin

 Process(Ext_clk_50MHz)
 begin
 if (rising_edge (Ext_clk_50MHz)) then
      if (divide_counter /= 5) then
            divide_counter <= divide_counter + 1;
      else
            divide_counter <= 0;
            dev_clock_5MHz <= not (dev_clock_5MHz);
      end if;
 end if;
 end process;


end Behavioral;
```

9.4.5 Divider 25 MHz

```
----------------------------------------------------------------------------
-- Teilt Takt von 50 MHz auf 25 MHz
-- Takt wird bei inizialisierung des LCD benötigt
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity DIVIDER_25MHz is
 Port (
            Ext_clk_50MHz           : in STD_LOGIC;
            dev_clock_25MHz         : inout STD_LOGIC
 );
end DIVIDER_25MHz;

architecture Behavioral of DIVIDER_25MHz is
signal divide_counter : integer := 0;
begin

 Process(Ext_clk_50MHz)
 begin
 if (rising_edge (Ext_clk_50MHz)) then
      if (divide_counter /= 0) then
            divide_counter <= divide_counter + 1;
      else
            divide_counter <= 0;
            dev_clock_25MHz <= not (dev_clock_25MHz);
      end if;
 end if;
end process;

end Behavioral;
```

9.4.6 Randomgenerator

```
----------------------------------------------------------------------------
-- Zufallszahlengerator
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RANDOMGENERATOR is
      PORT (
            dev_clock_2_5MHz        : in STD_LOGIC;
            random_out              : inout STD_LOGIC_VECTOR (13 downto 0);
      );
end RANDOMGENERATOR;

architecture Behavioral of RANDOMGENERATOR is
signal random : STD_LOGIC_VECTOR(13 downto 0) := "00000001111111";
signal interncounter : STD_LOGIC_VECTOR (6 downto 0);
begin
process(dev_clock_2_5MHz)
begin
if rising_edge (dev_clock_2_5MHz) then
 interncounter <= interncounter + 1; -- 125 Zyklen für Übertragungszeit 5us
 random(11 downto 1) <= random(10 downto 0);
 random(0)    <=    not(random(0))    XOR    random(8)    XOR    random(2)    XOR
(not(random(7))) XOR random(4);
 case interncounter is
  when "0000000" => random_out(13 downto 12) <= "01";
                    random_out(11 downto 0) <= random(11 downto 0);
  when "0000100" => random_out(13 downto 12) <= "10";
                    random_out(11 downto 0) <= random(11 downto 0);
  when "0001000" => random_out(13 downto 12) <= "11";
  when "1111101" => interncounter <= "0000000";
  when others => random_out <= "00000001111111";
 end case;
end if;
end process;
end Behavioral;
```

9.4.7 Coder

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

entity CODER is
 PORT(
      Ext_clk_50MHz           : in        STD_LOGIC;
      random_out              : in STD_LOGIC_VECTOR (13 downto 0);
      random_out_coded        : inout STD_LOGIC_VECTOR (17 downto 0)
 );
end CODER;

architecture Behavioral of CODER is

begin
process (Ext_clk_50MHz)
begin
if rising_edge (Ext_clk_50MHz) then
      random_out_coded(15) <= '0';
      random_out_coded(17 downto 16) <= random_out (13 downto 12);
 if random_out(13 downto 0) = "00000001111111" then
      random_out_coded(17 downto 0) <= "000000000111111111";
 else
 case random_out(3 downto 0) is
  when "0000" => random_out_coded(4 downto 0) <= "11110";
  when "0001" => random_out_coded(4 downto 0) <= "01001";
  when "0010" => random_out_coded(4 downto 0) <= "10100";
  when "0011" => random_out_coded(4 downto 0) <= "10101";
  when "0100" => random_out_coded(4 downto 0) <= "01010";
  when "0101" => random_out_coded(4 downto 0) <= "01011";
  when "0110" => random_out_coded(4 downto 0) <= "01110";
  when "0111" => random_out_coded(4 downto 0) <= "01111";
  when "1000" => random_out_coded(4 downto 0) <= "10010";
  when "1001" => random_out_coded(4 downto 0) <= "10011";
  when "1010" => random_out_coded(4 downto 0) <= "10110";
  when "1011" => random_out_coded(4 downto 0) <= "10111";
  when "1100" => random_out_coded(4 downto 0) <= "11010";
  when "1101" => random_out_coded(4 downto 0) <= "11011";
  when "1110" => random_out_coded(4 downto 0) <= "11100";
  when "1111" => random_out_coded(4 downto 0) <= "11101";
  when others => random_out_coded(4 downto 0) <= random_out_coded(4 downto
0);
 end case;
 case random_out(7 downto 4) is
  when "0000" => random_out_coded(9 downto 5) <= "11110";
  when "0001" => random_out_coded(9 downto 5) <= "01001";
  when "0010" => random_out_coded(9 downto 5) <= "10100";
  when "0011" => random_out_coded(9 downto 5) <= "10101";
  when "0100" => random_out_coded(9 downto 5) <= "01010";
  when "0101" => random_out_coded(9 downto 5) <= "01011";
  when "0110" => random_out_coded(9 downto 5) <= "01110";
  when "0111" => random_out_coded(9 downto 5) <= "01111";
  when "1000" => random_out_coded(9 downto 5) <= "10010";
  when "1001" => random_out_coded(9 downto 5) <= "10011";
  when "1010" => random_out_coded(9 downto 5) <= "10110";
  when "1011" => random_out_coded(9 downto 5) <= "10111";
  when "1100" => random_out_coded(9 downto 5) <= "11010";
  when "1101" => random_out_coded(9 downto 5) <= "11011";
  when "1110" => random_out_coded(9 downto 5) <= "11100";
```

```
  when "1111" => random_out_coded(9 downto 5) <= "11101";
  when others => random_out_coded(9 downto 5) <= random_out_coded(4 downto
0);
 end case;
 case random_out(11 downto 8) is
  when "0000" => random_out_coded(14 downto 10) <= "11110";
  when "0001" => random_out_coded(14 downto 10) <= "01001";
  when "0010" => random_out_coded(14 downto 10) <= "10100";
  when "0011" => random_out_coded(14 downto 10) <= "10101";
  when "0100" => random_out_coded(14 downto 10) <= "01010";
  when "0101" => random_out_coded(14 downto 10) <= "01011";
  when "0110" => random_out_coded(14 downto 10) <= "01110";
  when "0111" => random_out_coded(14 downto 10) <= "01111";
  when "1000" => random_out_coded(14 downto 10) <= "10010";
  when "1001" => random_out_coded(14 downto 10) <= "10011";
  when "1010" => random_out_coded(14 downto 10) <= "10110";
  when "1011" => random_out_coded(14 downto 10) <= "10111";
  when "1100" => random_out_coded(14 downto 10) <= "11010";
  when "1101" => random_out_coded(14 downto 10) <= "11011";
  when "1110" => random_out_coded(14 downto 10) <= "11100";
  when "1111" => random_out_coded(14 downto 10) <= "11101";
  when others => random_out_coded(14 downto 10) <= random_out_coded(14
downto 10);
 end case;
end if;
end if;
end process;
end Behavioral;
```

9.4.8 Decoder

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


entity DECODER is
 PORT(
            Ext_clk_50MHz           : in        STD_LOGIC;
            Data_received_decoded   : inout STD_LOGIC_VECTOR (13 downto 0);
            Data_received_coded     : inout STD_LOGIC_VECTOR (17 downto 0)
 );
end DECODER;

architecture Behavioral of DECODER is

begin
process (Ext_clk_50MHz)
begin
if rising_edge (Ext_clk_50MHz) then
 Data_received_decoded(13 downto 12) <= Data_received_coded(17 downto 16);
if Data_received_coded = "000000000111111111" then
 Data_received_decoded <= "00000001111111";
else
 case Data_received_coded(4 downto 0) is
  when "11110" => Data_received_decoded(3 downto 0) <= "0000";
  when "01001" => Data_received_decoded(3 downto 0) <= "0001";
  when "10100" => Data_received_decoded(3 downto 0) <= "0010";
  when "10101" => Data_received_decoded(3 downto 0) <= "0011";
  when "01010" => Data_received_decoded(3 downto 0) <= "0100";
  when "01011" => Data_received_decoded(3 downto 0) <= "0101";
  when "01110" => Data_received_decoded(3 downto 0) <= "0110";
  when "01111" => Data_received_decoded(3 downto 0) <= "0111";
  when "10010" => Data_received_decoded(3 downto 0) <= "1000";
  when "10011" => Data_received_decoded(3 downto 0) <= "1001";
  when "10110" => Data_received_decoded(3 downto 0) <= "1010";
  when "10111" => Data_received_decoded(3 downto 0) <= "1011";
  when "11010" => Data_received_decoded(3 downto 0) <= "1100";
  when "11011" => Data_received_decoded(3 downto 0) <= "1101";
  when "11100" => Data_received_decoded(3 downto 0) <= "1110";
  when "11101" => Data_received_decoded(3 downto 0) <= "1111";
  when others => Data_received_decoded(3 downto 0)  <=
Data_received_decoded(3 downto 0);
 end case;
 case Data_received_coded(9 downto 5) is
  when "11110" => Data_received_decoded(7 downto 4) <= "0000";
  when "01001" => Data_received_decoded(7 downto 4) <= "0001";
  when "10100" => Data_received_decoded(7 downto 4) <= "0010";
  when "10101" => Data_received_decoded(7 downto 4) <= "0011";
  when "01010" => Data_received_decoded(7 downto 4) <= "0100";
  when "01011" => Data_received_decoded(7 downto 4) <= "0101";
  when "01110" => Data_received_decoded(7 downto 4) <= "0110";
  when "01111" => Data_received_decoded(7 downto 4) <= "0111";
  when "10010" => Data_received_decoded(7 downto 4) <= "1000";
  when "10011" => Data_received_decoded(7 downto 4) <= "1001";
  when "10110" => Data_received_decoded(7 downto 4) <= "1010";
  when "10111" => Data_received_decoded(7 downto 4) <= "1011";
  when "11010" => Data_received_decoded(7 downto 4) <= "1100";
  when "11011" => Data_received_decoded(7 downto 4) <= "1101";
  when "11100" => Data_received_decoded(7 downto 4) <= "1110";
```

```
  when "11101" => Data_received_decoded(7 downto 4) <= "1111";
  when others => Data_received_decoded(7 downto 4) <=
Data_received_decoded(7 downto 4);
 end case;
 case Data_received_coded(14 downto 10) is
  when "11110" => Data_received_decoded(11 downto 8) <= "0000";
  when "01001" => Data_received_decoded(11 downto 8) <= "0001";
  when "10100" => Data_received_decoded(11 downto 8) <= "0010";
  when "10101" => Data_received_decoded(11 downto 8) <= "0011";
  when "01010" => Data_received_decoded(11 downto 8) <= "0100";
  when "01011" => Data_received_decoded(11 downto 8) <= "0101";
  when "01110" => Data_received_decoded(11 downto 8) <= "0110";
  when "01111" => Data_received_decoded(11 downto 8) <= "0111";
  when "10010" => Data_received_decoded(11 downto 8) <= "1000";
  when "10011" => Data_received_decoded(11 downto 8) <= "1001";
  when "10110" => Data_received_decoded(11 downto 8) <= "1010";
  when "10111" => Data_received_decoded(11 downto 8) <= "1011";
  when "11010" => Data_received_decoded(11 downto 8) <= "1100";
  when "11011" => Data_received_decoded(11 downto 8) <= "1101";
  when "11100" => Data_received_decoded(11 downto 8) <= "1110";
  when "11101" => Data_received_decoded(11 downto 8) <= "1111";
  when others => Data_received_decoded(11 downto 8) <=
Data_received_decoded(11 downto 8);
 end case;
end if;
end if;
end process;
end Behavioral;
```

9.4.9 Comparison

```
----------------------------------------------------------------------------
--Dieses Instanz COMPARISON soll die gesendeten Daten mit den Empfangenen
--vergleichen und
--Fehler zaehlt bestimmen
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;

entity COMPARISON is
      PORT(
            dev_clock_2_5MHz        : in STD_LOGIC;
            dev_clock_5MHz          : in STD_LOGIC;
            random_out              : in STD_LOGIC_VECTOR(13 downto 0);
            nsd1                    : inout integer;
            nsd10                   : inout integer;
            nsd100                  : inout integer;
            nsd1000                 : inout integer;
            nsd10000                : inout integer;
            nsd100000               : inout integer;
            nsd1000000              : inout integer;
            nsd10000000             : inout integer;
            nsd100000000            : inout integer;
            nsd1000000000           : inout integer;
            nsd10000000000          : inout integer;
            nof1                    : inout integer;
            nof10                   : inout integer;
            nof100                  : inout integer;
            nof1000                 : inout integer;
            nof10000                : inout integer;
            nof100000               : inout integer;
            nof1000000              : inout integer;
            nof10000000             : inout integer;
            nof100000000            : inout integer;
            nof1000000000           : inout integer;
            nof10000000000          : inout integer;
            nof                     : inout STD_LOGIC_VECTOR (10 downto 0);
            data_received_decoded   : inout STD_LOGIC_VECTOR(13 downto 0);
            reset                   : in STD_LOGIC
            );
end COMPARISON;

architecture Behavioral of COMPARISON is
signal nsd1notshown : integer := 0;
signal nsd10notshown: integer := 0;
signal nsd100notshown : integer := 0;
begin

process (dev_clock_5MHz, dev_clock_2_5MHz)
begin
if rising_edge (dev_clock_2_5MHz) then
if reset = '1' then
nsd1notshown <= 0;
nsd10notshown <= 0;
nsd100notshown <= 0;
nsd1 <= 0;
nsd10 <= 0;
nsd100 <= 0;
```

```
nsd1000 <= 0;
nsd10000 <= 0;
nsd100000 <= 0;
nsd1000000 <= 0;
nsd10000000 <= 0;
nsd100000000 <= 0;
nsd1000000000 <= 0;
nsd10000000000 <= 0;
else if nsd1notshown < 9 then
  nsd1notshown <= nsd1notshown +1;
  else
        nsd1notshown <= 0;
 if nsd10notshown < 9 then
  nsd10notshown <= nsd10notshown +1;
  else
        nsd10notshown <= 0;
 if nsd100notshown < 9 then
  nsd100notshown <= nsd100notshown +1;
  else
        nsd100notshown <= 0;
 if nsd1 < 9 then
  nsd1 <= nsd1 +1;
  else
        nsd1 <= 0;
     if nsd10 < 9 then
      nsd10 <= nsd10 +1;
    else
      nsd10 <= 0;
      if nsd100 < 9 then
        nsd100 <= nsd100 +1;
      else
        nsd100 <= 0;
        if nsd1000 < 9 then
          nsd1000 <= nsd1000 +1;
        else
          nsd1000 <= 0;
          if nsd10000 < 9 then
            nsd10000 <= nsd10000 +1;
          else
            nsd10000 <= 0;
            if nsd100000 < 9 then
            nsd100000 <= nsd100000 +1;
             else
             nsd100000 <= 0;
             if nsd1000000 < 9 then
             nsd1000000 <= nsd1000000 +1;
              else
              nsd1000000 <= 0;
              if nsd10000000 < 9 then
              nsd10000000 <= nsd10000000 +1;
               else
               nsd10000000 <= 0;
               if nsd100000000 < 9 then
               nsd100000000 <= nsd100000000 +1;
                else
                nsd100000000 <= 0;
                if nsd1000000000 < 9 then
                nsd1000000000 <= nsd1000000000 +1;
                 else
                 nsd1000000000 <= 0;
                 if nsd10000000000 < 9 then
                 nsd10000000000 <= nsd10000000000 +1;
                  else
```

```
                        nsd10000000000 <= 0,
                    end if ;
                  end if;
                 end if;
                end if;
               end if;
              end if ;
             end if;
            end if;
           end if;
          end if;
        end if;
       end if;
      end if;
     end if;
     end if;
if rising_edge (dev_clock_5MHz) and dev_clock_2_5MHz = '0'   then
if reset = '1' then
nof1 <= 0;
nof10 <= 0;
nof100 <= 0;
nof1000 <= 0;
nof10000 <= 0;
nof100000 <= 0;
nof1000000 <= 0;
nof10000000 <= 0;
nof100000000 <= 0;
nof1000000000 <= 0;
nof10000000000 <= 0;
else if (Data_received_decoded(11 downto 0) /= random_out(11 downto 0))
then
if nof1 < 9 then
     nof1 <= nof1 +1;
  else
       nof1 <= 0;
    if nof10 < 9 then
      nof10 <= nof10 +1;
    else
      nof10 <= 0;
      if nof100 < 9 then
        nof100 <= nof100 +1;
      else
        nof100 <= 0;
        if nof1000 < 9 then
          nof1000 <= nof1000 +1;
        else
          nof1000 <= 0;
          if nof10000 < 9 then
            nof10000 <= nof10000 +1;
          else
            nof10000 <= 0;
            if nof100000 < 9 then
            nof100000 <= nof10000 +1;
             else
             nof10000 <= 0;
             if nof1000000 < 9 then
             nof1000000 <= nof1000000 +1;
              else
              nof1000000 <= 0;
              if nof10000000 < 9 then
              nof10000000 <= nof10000000 +1;
               else
```

```
                nof10000000 <= 0;
                if nof100000000 < 9 then
                nof100000000 <= nof100000000 +1;
                 else
                 nof100000000 <= 0;
                 if nof1000000000 < 9 then
                 nof1000000000 <= nof1000000000 +1;
                  else
                  nof1000000000 <= 0;
                  if nof10000000000 < 9 then
                  nof10000000000 <= nof10000000000 +1;
                   else
                   nof10000000000 <= 0;

                 end if ;
                end if;
               end if;
             end if;
            end if;
          end if ;
        end if;
      end if;
     end if;
    end if;
  end if ;
 end if;
end if;
end if;
end process;
end Behavioral;
```

9.4.10 Set_LCD

```
----------------------------------------------------------------------------------
-- Signalzuweisung für LCD
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity SET_LCD is
      PORT(
            dev_clock_5MHz          : in  STD_LOGIC;
            noflowbits1             : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits2             : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits3             : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits4             : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits5             : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits6             : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits7             : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits8             : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits9             : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits10            : out STD_LOGIC_VECTOR (3 downto 0);
            noflowbits11            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits1            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits2            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits3            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits4            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits5            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits6            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits7            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits8            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits9            : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits10           : out STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits11           : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits1             : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits2             : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits3             : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits4             : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits5             : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits6             : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits7             : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits8             : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits9             : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits10            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits11            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits1            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits2            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits3            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits4            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits5            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits6            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits7            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits8            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits9            : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits10           : out STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits11           : out STD_LOGIC_VECTOR (3 downto 0);
            nof1                    : in integer;
```

```
            nof10                   : in integer;
            nof100                  : in integer;
            nof1000                 : in integer;
            nof10000                : in integer;
            nof100000               : in integer;
            nof1000000              : in integer;
            nof10000000             : in integer;
            nof100000000            : in integer;
            nof1000000000           : in integer;
            nof10000000000          : in integer;
            nsd1                    : in integer;
            nsd10                   : in integer;
            nsd100                  : in integer;
            nsd1000                 : in integer;
            nsd10000                : in integer;
            nsd100000               : in integer;
            nsd1000000              : in integer;
            nsd10000000             : in integer;
            nsd100000000            : in integer;
            nsd1000000000           : in integer;
            nsd10000000000          : in integer;
            din0                    : out STD_LOGIC_VECTOR (3 downto 0);
            din1                    : out STD_LOGIC_VECTOR (3 downto 0);
            din2                    : out STD_LOGIC_VECTOR (3 downto 0);
            din3                    : out STD_LOGIC_VECTOR (3 downto 0);
            din4                    : out STD_LOGIC_VECTOR (3 downto 0);
            din5                    : out STD_LOGIC_VECTOR (3 downto 0);
            din6                    : out STD_LOGIC_VECTOR (3 downto 0);
            din7                    : out STD_LOGIC_VECTOR (3 downto 0);
            din8                    : out STD_LOGIC_VECTOR (3 downto 0);
            din9                    : out STD_LOGIC_VECTOR (3 downto 0);
            din10                   : out STD_LOGIC_VECTOR (3 downto 0);
            din11                   : out STD_LOGIC_VECTOR (3 downto 0);
            din12                   : out STD_LOGIC_VECTOR (3 downto 0);
            din13                   : out STD_LOGIC_VECTOR (3 downto 0);
            din14                   : out STD_LOGIC_VECTOR (3 downto 0);
            din15                   : out STD_LOGIC_VECTOR (3 downto 0);
            data0                   : out STD_LOGIC_VECTOR (3 downto 0);
            data1                   : out STD_LOGIC_VECTOR (3 downto 0);
            data2                   : out STD_LOGIC_VECTOR (3 downto 0);
            data3                   : out STD_LOGIC_VECTOR (3 downto 0);
            data4                   : out STD_LOGIC_VECTOR (3 downto 0);
            data5                   : out STD_LOGIC_VECTOR (3 downto 0);
            data6                   : out STD_LOGIC_VECTOR (3 downto 0);
            data7                   : out STD_LOGIC_VECTOR (3 downto 0);
            data8                   : out STD_LOGIC_VECTOR (3 downto 0);
            data9                   : out STD_LOGIC_VECTOR (3 downto 0);
            data10                  : out STD_LOGIC_VECTOR (3 downto 0);
            data11                  : out STD_LOGIC_VECTOR (3 downto 0);
            data12                  : out STD_LOGIC_VECTOR (3 downto 0);
            data13                  : out STD_LOGIC_VECTOR (3 downto 0);
            data14                  : out STD_LOGIC_VECTOR (3 downto 0);
            data15                  : out STD_LOGIC_VECTOR (3 downto 0)
      );
end SET_LCD;

architecture Behavioral of SET_LCD is

begin

process (dev_clock_5MHz)
begin
```

```
if rising_edge(dev_clock_5MHz)then
 if nof1 = 0 then
 noflowbits11  <= "0000";
 nofhighbits11 <= "0011";
 else
  if nof1 = 1 then
  noflowbits11  <= "0001";
  nofhighbits11 <= "0011";
  else
   if nof1 = 2 then
   noflowbits11  <= "0010";
   nofhighbits11 <= "0011";
   else
    if nof1 = 3 then
    noflowbits11  <= "0011";
    nofhighbits11 <= "0011";
    else
     if nof1 = 4 then
     noflowbits11  <= "0100";
     nofhighbits11 <= "0011";
     else
      if nof1 = 5 then
      noflowbits11  <= "0101";
      nofhighbits11 <= "0011";
      else
       if nof1 = 6 then
       noflowbits11  <= "0110";
       nofhighbits11 <= "0011";
       else
        if nof1 = 7 then
        noflowbits11  <= "0111";
        nofhighbits11 <= "0011";
       else
        if nof1 = 8 then
        noflowbits11  <= "1000";
        nofhighbits11 <= "0011";
        else
         if nof1 = 9 then
         noflowbits11  <= "1001";
         nofhighbits11 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof10 = 0 then
 noflowbits10  <= "0000";
 nofhighbits10 <= "0011";
 else
  if nof10 = 1 then
  noflowbits10  <= "0001";
  nofhighbits10 <= "0011";
  else
   if nof10 = 2 then
   noflowbits10  <= "0010";
   nofhighbits10 <= "0011";
```

```
   else
    if nof10 = 3 then
    noflowbits10  <= "0011";
    nofhighbits10 <= "0011";
    else
     if nof10 = 4 then
     noflowbits10  <= "0100";
     nofhighbits10 <= "0011";
     else
      if nof10 = 5 then
      noflowbits10  <= "0101";
      nofhighbits10 <= "0011";
      else
       if nof10 = 6 then
       noflowbits10  <= "0110";
       nofhighbits10 <= "0011";
       else
        if nof10 = 7 then
        noflowbits10  <= "0111";
        nofhighbits10 <= "0011";
       else
        if nof10 = 8 then
        noflowbits10  <= "1000";
        nofhighbits10 <= "0011";
        else
         if nof10 = 9 then
         noflowbits10  <= "1001";
         nofhighbits10 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof100 = 0 then
 noflowbits9  <= "0000";
 nofhighbits9 <= "0011";
 else
  if nof100 = 1 then
  noflowbits9  <= "0001";
  nofhighbits9 <= "0011";
  else
   if nof100 = 2 then
   noflowbits9  <= "0010";
   nofhighbits9 <= "0011";
   else
    if nof100 = 3 then
    noflowbits9  <= "0011";
    nofhighbits9 <= "0011";
    else
     if nof100 = 4 then
     noflowbits9  <= "0100";
     nofhighbits9<= "0011";
     else
      if nof100 = 5 then
      noflowbits9  <= "0101";
      nofhighbits9<= "0011";
```

```
      else
       if nof100 = 6 then
       noflowbits9  <= "0110";
       nofhighbits9 <= "0011";
       else
        if nof100 = 7 then
        noflowbits9  <= "0111";
        nofhighbits9 <= "0011";
       else
        if nof100 = 8 then
        noflowbits9  <= "1000";
        nofhighbits9 <= "0011";
        else
         if nof100 = 9 then
         noflowbits9  <= "1001";
         nofhighbits9 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof1000 = 0 then
 noflowbits8  <= "0000";
 nofhighbits8 <= "0011";
 else
  if nof1000 = 1 then
  noflowbits8  <= "0001";
  nofhighbits8 <= "0011";
  else
   if nof1000 = 2 then
   noflowbits8  <= "0010";
   nofhighbits8 <= "0011";
   else
    if nof1000 = 3 then
    noflowbits8  <= "0011";
    nofhighbits8 <= "0011";
    else
     if nof1000 = 4 then
     noflowbits8  <= "0100";
     nofhighbits8<= "0011";
     else
      if nof1000 = 5 then
      noflowbits8  <= "0101";
      nofhighbits8<= "0011";
      else
       if nof1000 = 6 then
       noflowbits8  <= "0110";
       nofhighbits8 <= "0011";
       else
        if nof1000 = 7 then
        noflowbits8  <= "0111";
        nofhighbits8 <= "0011";
       else
        if nof1000 = 8 then
        noflowbits8  <= "1000";
        nofhighbits8 <= "0011";
```

```
        else
         if nof1000 = 9 then
         noflowbits8  <= "1001";
         nofhighbits8 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof10000 = 0 then
 noflowbits7  <= "0000";
 nofhighbits7 <= "0011";
 else
  if nof10000 = 1 then
  noflowbits7  <= "0001";
  nofhighbits7 <= "0011";
  else
   if nof10000 = 2 then
   noflowbits7  <= "0010";
   nofhighbits7 <= "0011";
   else
    if nof10000 = 3 then
    noflowbits7 <= "0011";
    nofhighbits7 <= "0011";
    else
     if nof10000 = 4 then
     noflowbits7  <= "0100";
     nofhighbits7<= "0011";
     else
      if nof10000 = 5 then
      noflowbits7  <= "0101";
      nofhighbits7<= "0011";
      else
       if nof10000 = 6 then
       noflowbits7  <= "0110";
       nofhighbits7 <= "0011";
       else
        if nof10000 = 7 then
        noflowbits7  <= "0111";
        nofhighbits7 <= "0011";
       else
        if nof10000 = 8 then
        noflowbits7 <= "1000";
        nofhighbits7 <= "0011";
        else
         if nof10000 = 9 then
         noflowbits7  <= "1001";
         nofhighbits7 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
```

```
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof100000 = 0 then
 noflowbits6  <= "0000";
 nofhighbits6 <= "0011";
 else
  if nof100000 = 1 then
  noflowbits6  <= "0001";
  nofhighbits6 <= "0011";
  else
   if nof100000 = 2 then
   noflowbits6  <= "0010";
   nofhighbits6 <= "0011";
   else
    if nof100000 = 3 then
    noflowbits6 <= "0011";
    nofhighbits6 <= "0011";
    else
     if nof100000 = 4 then
     noflowbits6  <= "0100";
     nofhighbits6<= "0011";
     else
      if nof100000 = 5 then
      noflowbits6  <= "0101";
      nofhighbits6<= "0011";
      else
       if nof100000 = 6 then
       noflowbits6  <= "0110";
       nofhighbits6 <= "0011";
       else
        if nof100000 = 7 then
        noflowbits6  <= "0111";
        nofhighbits6 <= "0011";
       else
        if nof100000 = 8 then
        noflowbits6 <= "1000";
        nofhighbits6 <= "0011";
        else
         if nof10000 = 9 then
         noflowbits6  <= "1001";
         nofhighbits6 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof1000000 = 0 then
 noflowbits5  <= "0000";
 nofhighbits5 <= "0011";
 else
  if nof1000000 = 1 then
  noflowbits5  <= "0001";
  nofhighbits5 <= "0011";
  else
```

```
   if nof1000000 = 2 then
   noflowbits5  <= "0010";
   nofhighbits5 <= "0011";
   else
    if nof1000000 = 3 then
    noflowbits5 <= "0011";
    nofhighbits5 <= "0011";
    else
     if nof1000000 = 4 then
     noflowbits5  <= "0100";
     nofhighbits5<= "0011";
     else
      if nof1000000 = 5 then
      noflowbits5  <= "0101";
      nofhighbits5<= "0011";
      else
       if nof1000000 = 6 then
       noflowbits5  <= "0110";
       nofhighbits5 <= "0011";
       else
        if nof1000000 = 7 then
        noflowbits5  <= "0111";
        nofhighbits5 <= "0011";
       else
        if nof1000000 = 8 then
        noflowbits5 <= "1000";
        nofhighbits5 <= "0011";
        else
         if nof100000 = 9 then
         noflowbits5  <= "1001";
         nofhighbits5 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof10000000 = 0 then
 noflowbits4  <= "0000";
 nofhighbits4 <= "0011";
 else
  if nof10000000 = 1 then
  noflowbits4  <= "0001";
  nofhighbits4 <= "0011";
  else
   if nof10000000 = 2 then
   noflowbits4  <= "0010";
   nofhighbits4 <= "0011";
   else
    if nof10000000 = 3 then
    noflowbits4 <= "0011";
    nofhighbits4 <= "0011";
    else
     if nof10000000 = 4 then
     noflowbits4  <= "0100";
     nofhighbits4<= "0011";
     else
```

```
      if nof10000000 = 5 then
      noflowbits4  <= "0101";
      nofhighbits4<= "0011";
      else
       if nof10000000 = 6 then
       noflowbits4  <= "0110";
       nofhighbits4 <= "0011";
       else
        if nof10000000 = 7 then
        noflowbits4  <= "0111";
        nofhighbits4 <= "0011";
       else
        if nof10000000 = 8 then
        noflowbits4 <= "1000";
        nofhighbits4 <= "0011";
        else
         if nof1000000 = 9 then
         noflowbits4  <= "1001";
         nofhighbits4 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof100000000 = 0 then
 noflowbits3  <= "0000";
 nofhighbits3 <= "0011";
 else
  if nof100000000 = 1 then
  noflowbits3  <= "0001";
  nofhighbits3 <= "0011";
  else
   if nof100000000 = 2 then
   noflowbits3  <= "0010";
   nofhighbits3 <= "0011";
   else
    if nof100000000 = 3 then
    noflowbits3 <= "0011";
    nofhighbits3 <= "0011";
    else
     if nof100000000 = 4 then
     noflowbits3  <= "0100";
     nofhighbits3 <= "0011";
     else
      if nof100000000 = 5 then
      noflowbits3  <= "0101";
      nofhighbits3 <= "0011";
      else
       if nof100000000 = 6 then
       noflowbits3  <= "0110";
       nofhighbits3 <= "0011";
       else
        if nof100000000 = 7 then
        noflowbits3  <= "0111";
        nofhighbits3 <= "0011";
       else
```

```
        if nof100000000 = 8 then
        noflowbits3 <= "1000";
        nofhighbits3 <= "0011";
        else
         if nof10000000 = 9 then
         noflowbits3  <= "1001";
         nofhighbits3 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof1000000000 = 0 then
 noflowbits2  <= "0000";
 nofhighbits2 <= "0011";
 else
  if nof1000000000 = 1 then
  noflowbits2  <= "0001";
  nofhighbits2 <= "0011";
  else
   if nof1000000000 = 2 then
   noflowbits2  <= "0010";
   nofhighbits2 <= "0011";
   else
    if nof1000000000 = 3 then
    noflowbits2 <= "0011";
    nofhighbits2 <= "0011";
    else
     if nof1000000000 = 4 then
     noflowbits2  <= "0100";
     nofhighbits2 <= "0011";
     else
      if nof1000000000 = 5 then
      noflowbits2  <= "0101";
      nofhighbits2 <= "0011";
      else
       if nof1000000000 = 6 then
       noflowbits2  <= "0110";
       nofhighbits2 <= "0011";
       else
        if nof1000000000 = 7 then
        noflowbits2  <= "0111";
        nofhighbits2 <= "0011";
       else
        if nof1000000000 = 8 then
        noflowbits2 <= "1000";
        nofhighbits2 <= "0011";
        else
         if nof100000000 = 9 then
         noflowbits2  <= "1001";
         nofhighbits2 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
```

```
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nof10000000000 = 0 then
 noflowbits1  <= "0000";
 nofhighbits1 <= "0011";
 else
  if nof10000000000 = 1 then
  noflowbits1  <= "0001";
  nofhighbits1 <= "0011";
  else
   if nof10000000000 = 2 then
   noflowbits1  <= "0010";
   nofhighbits1 <= "0011";
   else
    if nof10000000000 = 3 then
    noflowbits1 <= "0011";
    nofhighbits1 <= "0011";
    else
     if nof10000000000 = 4 then
     noflowbits1  <= "0100";
     nofhighbits1 <= "0011";
     else
      if nof10000000000 = 5 then
      noflowbits1  <= "0101";
      nofhighbits1 <= "0011";
      else
       if nof10000000000 = 6 then
       noflowbits1  <= "0110";
       nofhighbits1 <= "0011";
       else
        if nof10000000000 = 7 then
        noflowbits1  <= "0111";
        nofhighbits1 <= "0011";
       else
        if nof10000000000 = 8 then
        noflowbits1 <= "1000";
        nofhighbits1 <= "0011";
        else
         if nof1000000000 = 9 then
         noflowbits1  <= "1001";
         nofhighbits1 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd1 = 0 then
 nsdlowbits11  <= "0000";
 nsdhighbits11 <= "0011";
 else
  if nsd1 = 1 then
```

```
  nsdlowbits11  <= "0001";
  nsdhighbits11 <= "0011";
  else
   if nsd1 = 2 then
   nsdlowbits11  <= "0010";
   nsdhighbits11 <= "0011";
   else
    if nsd1 = 3 then
    nsdlowbits11  <= "0011";
    nsdhighbits11 <= "0011";
    else
     if nsd1 = 4 then
     nsdlowbits11  <= "0100";
     nsdhighbits11 <= "0011";
     else
      if nsd1 = 5 then
      nsdlowbits11  <= "0101";
      nsdhighbits11 <= "0011";
      else
       if nsd1 = 6 then
       nsdlowbits11  <= "0110";
       nsdhighbits11 <= "0011";
       else
        if nsd1 = 7 then
        nsdlowbits11  <= "0111";
        nsdhighbits11 <= "0011";
       else
        if nsd1 = 8 then
        nsdlowbits11  <= "1000";
        nsdhighbits11 <= "0011";
        else
         if nsd1 = 9 then
         nsdlowbits11  <= "1001";
         nsdhighbits11 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd10 = 0 then
 nsdlowbits10  <= "0000";
 nsdhighbits10 <= "0011";
 else
  if nsd10 = 1 then
  nsdlowbits10  <= "0001";
  nsdhighbits10 <= "0011";
  else
   if nsd10 = 2 then
   nsdlowbits10  <= "0010";
   nsdhighbits10 <= "0011";
   else
    if nsd10 = 3 then
    nsdlowbits10  <= "0011";
    nsdhighbits10 <= "0011";
    else
     if nsd10 = 4 then
```

```
      nsdlowbits10  <= "0100";
      nsdhighbits10 <= "0011";
      else
       if nsd10 = 5 then
       nsdlowbits10  <= "0101";
       nsdhighbits10 <= "0011";
       else
        if nsd10 = 6 then
        nsdlowbits10  <= "0110";
        nsdhighbits10 <= "0011";
        else
         if nsd10 = 7 then
         nsdlowbits10  <= "0111";
         nsdhighbits10 <= "0011";
        else
         if nsd10 = 8 then
         nsdlowbits10  <= "1000";
         nsdhighbits10 <= "0011";
         else
          if nsd10 = 9 then
          nsdlowbits10  <= "1001";
          nsdhighbits10 <= "0011";
          end if;
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd100 = 0 then
 nsdlowbits9  <= "0000";
 nsdhighbits9 <= "0011";
 else
  if nsd100 = 1 then
  nsdlowbits9  <= "0001";
  nsdhighbits9 <= "0011";
  else
   if nsd100 = 2 then
   nsdlowbits9  <= "0010";
   nsdhighbits9 <= "0011";
   else
    if nsd100 = 3 then
    nsdlowbits9  <= "0011";
    nsdhighbits9 <= "0011";
    else
     if nsd100 = 4 then
     nsdlowbits9  <= "0100";
     nsdhighbits9<= "0011";
     else
      if nsd100 = 5 then
      nsdlowbits9  <= "0101";
      nsdhighbits9<= "0011";
      else
       if nsd100 = 6 then
       nsdlowbits9  <= "0110";
       nsdhighbits9 <= "0011";
       else
        if nsd100 = 7 then
```

```
        nsdlowbits9  <= "0111";
        nsdhighbits9 <= "0011";
       else
        if nsd100 = 8 then
        nsdlowbits9  <= "1000";
        nsdhighbits9 <= "0011";
        else
         if nsd100 = 9 then
         nsdlowbits9  <= "1001";
         nsdhighbits9 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd1000 = 0 then
 nsdlowbits8  <= "0000";
 nsdhighbits8 <= "0011";
 else
  if nsd1000 = 1 then
  nsdlowbits8  <= "0001";
  nsdhighbits8 <= "0011";
  else
   if nsd1000 = 2 then
   nsdlowbits8  <= "0010";
   nsdhighbits8 <= "0011";
   else
    if nsd1000 = 3 then
    nsdlowbits8  <= "0011";
    nsdhighbits8 <= "0011";
    else
     if nsd1000 = 4 then
     nsdlowbits8  <= "0100";
     nsdhighbits8<= "0011";
     else
      if nsd1000 = 5 then
      nsdlowbits8  <= "0101";
      nsdhighbits8<= "0011";
      else
       if nsd1000 = 6 then
       nsdlowbits8  <= "0110";
       nsdhighbits8 <= "0011";
       else
        if nsd1000 = 7 then
        nsdlowbits8  <= "0111";
        nsdhighbits8 <= "0011";
       else
        if nsd1000 = 8 then
        nsdlowbits8  <= "1000";
        nsdhighbits8 <= "0011";
        else
         if nsd1000 = 9 then
         nsdlowbits8  <= "1001";
         nsdhighbits8 <= "0011";
         end if;
        end if;
```

```
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd10000 = 0 then
 nsdlowbits7  <= "0000";
 nsdhighbits7 <= "0011";
 else
  if nsd10000 = 1 then
  nsdlowbits7  <= "0001";
  nsdhighbits7 <= "0011";
  else
   if nsd10000 = 2 then
   nsdlowbits7  <= "0010";
   nsdhighbits7 <= "0011";
   else
    if nsd10000 = 3 then
    nsdlowbits7 <= "0011";
    nsdhighbits7 <= "0011";
    else
     if nsd10000 = 4 then
     nsdlowbits7  <= "0100";
     nsdhighbits7<= "0011";
     else
      if nsd10000 = 5 then
      nsdlowbits7  <= "0101";
      nsdhighbits7<= "0011";
      else
       if nsd10000 = 6 then
       nsdlowbits7  <= "0110";
       nsdhighbits7 <= "0011";
       else
        if nsd10000 = 7 then
        nsdlowbits7  <= "0111";
        nsdhighbits7 <= "0011";
       else
        if nsd10000 = 8 then
        nsdlowbits7 <= "1000";
        nsdhighbits7 <= "0011";
        else
         if nsd10000 = 9 then
         nsdlowbits7  <= "1001";
         nsdhighbits7 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd100000 = 0 then
 nsdlowbits6  <= "0000";
```

```
 nsdhighbits6 <= "0011";
 else
  if nsd100000 = 1 then
  nsdlowbits6  <= "0001";
  nsdhighbits6 <= "0011";
  else
   if nsd100000 = 2 then
   nsdlowbits6  <= "0010";
   nsdhighbits6 <= "0011";
   else
    if nsd100000 = 3 then
    nsdlowbits6 <= "0011";
    nsdhighbits6 <= "0011";
    else
     if nsd100000 = 4 then
     nsdlowbits6  <= "0100";
     nsdhighbits6<= "0011";
     else
      if nsd100000 = 5 then
      nsdlowbits6  <= "0101"
      nsdhighbits6<= "0011";
      else
       if nsd100000 = 6 then
       nsdlowbits6  <= "0110";
       nsdhighbits6 <= "0011";
       else
        if nsd100000 = 7 then
        nsdlowbits6  <= "0111";
        nsdhighbits6 <= "0011";
       else
        if nsd100000 = 8 then
        nsdlowbits6 <= "1000";
        nsdhighbits6 <= "0011";
        else
         if nsd10000 = 9 then
         nsdlowbits6  <= "1001";
         nsdhighbits6 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd1000000 = 0 then
 nsdlowbits5  <= "0000";
 nsdhighbits5 <= "0011";
 else
  if nsd1000000 = 1 then
  nsdlowbits5  <= "0001";
  nsdhighbits5 <= "0011";
  else
   if nsd1000000 = 2 then
   nsdlowbits5  <= "0010";
   nsdhighbits5 <= "0011";
   else
    if nsd1000000 = 3 then
    nsdlowbits5 <= "0011";
```

```
    nsdhighbits5 <= "0011";
    else
     if nsd1000000 = 4 then
     nsdlowbits5  <= "0100";
     nsdhighbits5<= "0011";
     else
      if nsd1000000 = 5 then
      nsdlowbits5  <= "0101";
      nsdhighbits5<= "0011";
      else
       if nsd1000000 = 6 then
       nsdlowbits5  <= "0110";
       nsdhighbits5 <= "0011";
       else
        if nsd1000000 = 7 then
        nsdlowbits5  <= "0111";
        nsdhighbits5 <= "0011";
       else
        if nsd1000000 = 8 then
        nsdlowbits5 <= "1000";
        nsdhighbits5 <= "0011";
        else
         if nsd100000 = 9 then
         nsdlowbits5  <= "1001";
         nsdhighbits5 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd10000000 = 0 then
 nsdlowbits4  <= "0000";
 nsdhighbits4 <= "0011";
 else
  if nsd10000000 = 1 then
  nsdlowbits4  <= "0001";
  nsdhighbits4 <= "0011";
  else
   if nsd10000000 = 2 then
   nsdlowbits4  <= "0010";
   nsdhighbits4 <= "0011";
   else
    if nsd10000000 = 3 then
    nsdlowbits4 <= "0011";
    nsdhighbits4 <= "0011";
    else
     if nsd10000000 = 4 then
     nsdlowbits4  <= "0100";
     nsdhighbits4<= "0011";
     else
      if nsd10000000 = 5 then
      nsdlowbits4  <= "0101";
      nsdhighbits4<= "0011";
      else
       if nsd10000000 = 6 then
       nsdlowbits4  <= "0110";
```

```
        nsdhighbits4 <= "0011";
        else
         if nsd10000000 = 7 then
         nsdlowbits4  <= "0111";
         nsdhighbits4 <= "0011";
        else
         if nsd10000000 = 8 then
         nsdlowbits4 <= "1000";
         nsdhighbits4 <= "0011";
         else
          if nsd1000000 = 9 then
          nsdlowbits4  <= "1001";
          nsdhighbits4 <= "0011";
          end if;
         end if;
        end if;
       end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd100000000 = 0 then
 nsdlowbits3  <= "0000";
 nsdhighbits3 <= "0011";
 else
  if nsd100000000 = 1 then
  nsdlowbits3  <= "0001";
  nsdhighbits3 <= "0011";
  else
   if nsd100000000 = 2 then
   nsdlowbits3  <= "0010";
   nsdhighbits3 <= "0011";
   else
    if nsd100000000 = 3 then
    nsdlowbits3 <= "0011";
    nsdhighbits3 <= "0011";
    else
     if nsd100000000 = 4 then
     nsdlowbits3  <= "0100";
     nsdhighbits3 <= "0011";
     else
      if nsd100000000 = 5 then
      nsdlowbits3  <= "0101";
      nsdhighbits3 <= "0011";
      else
       if nsd100000000 = 6 then
       nsdlowbits3  <= "0110";
       nsdhighbits3 <= "0011";
       else
        if nsd100000000 = 7 then
        nsdlowbits3  <= "0111";
        nsdhighbits3 <= "0011";
       else
        if nsd100000000 = 8 then
        nsdlowbits3 <= "1000";
        nsdhighbits3 <= "0011";
        else
         if nsd10000000 = 9 then
         nsdlowbits3  <= "1001";
```

```
          nsdhighbits3 <= "0011";
          end if;
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if nsd1000000000 = 0 then
 nsdlowbits2  <= "0000";
 nsdhighbits2 <= "0011";
 else
  if nsd1000000000 = 1 then
  nsdlowbits2  <= "0001";
  nsdhighbits2 <= "0011";
  else
   if nsd1000000000 = 2 then
   nsdlowbits2  <= "0010";
   nsdhighbits2 <= "0011";
   else
    if nsd1000000000 = 3 then
    nsdlowbits2 <= "0011";
    nsdhighbits2 <= "0011";
    else
     if nsd1000000000 = 4 then
     nsdlowbits2  <= "0100";
     nsdhighbits2 <= "0011";
     else
      if nsd1000000000 = 5 then
      nsdlowbits2  <= "0101";
      nsdhighbits2 <= "0011";
      else
       if nsd1000000000 = 6 then
       nsdlowbits2  <= "0110";
       nsdhighbits2 <= "0011";
       else
        if nsd1000000000 = 7 then
        nsdlowbits2  <= "0111";
        nsdhighbits2 <= "0011";
       else
        if nsd1000000000 = 8 then
        nsdlowbits2 <= "1000";
        nsdhighbits2 <= "0011";
        else
         if nsd100000000 = 9 then
         nsdlowbits2  <= "1001";
         nsdhighbits2 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
```

```
if rising_edge(dev_clock_5MHz)then
 if nsd10000000000 = 0 then
 nsdlowbits1  <= "0000";
 nsdhighbits1 <= "0011";
 else
  if nsd10000000000 = 1 then
  nsdlowbits1  <= "0001";
  nsdhighbits1 <= "0011";
  else
   if nsd10000000000 = 2 then
   nsdlowbits1  <= "0010";
   nsdhighbits1 <= "0011";
   else
    if nsd10000000000 = 3 then
    nsdlowbits1 <= "0011";
    nsdhighbits1 <= "0011";
    else
     if nsd10000000000 = 4 then
     nsdlowbits1  <= "0100";
     nsdhighbits1 <= "0011";
     else
      if nsd10000000000 = 5 then
      nsdlowbits1  <= "0101";
      nsdhighbits1 <= "0011";
      else
       if nsd10000000000 = 6 then
       nsdlowbits1  <= "0110";
       nsdhighbits1 <= "0011";
       else
        if nsd10000000000 = 7 then
        nsdlowbits1  <= "0111";
       else
        if nsd10000000000 = 8 then
        nsdlowbits1 <= "1000";
        nsdhighbits1 <= "0011";
        else
         if nsd1000000000 = 9 then
         nsdlowbits1  <= "1001";
         nsdhighbits1 <= "0011";
         end if;
        end if;
       end if;
      end if;
     end if;
    end if;
   end if;
  end if;
 end if;
end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(0) = '0' then
  din0 <= "0000";
 else
  din0 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(1) = '0' then
  din1 <= "0000";
 else
  din1 <= "0001";
 end if;
```

```
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(2) = '0' then
  din2 <= "0000";
 else
  din2 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(3) = '0' then
  din3 <= "0000";
 else
  din3 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(4) = '0' then
  din4 <= "0000";
 else
  din4 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(5) = '0' then
  din5 <= "0000";
 else
  din5 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(6) = '0' then
  din6 <= "0000";
 else
  din6 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(7) = '0' then
  din7 <= "0000";
 else
  din7 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(8) = '0' then
  din8 <= "0000";
 else
  din8 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(9) = '0' then
  din9 <= "0000";
 else
  din9 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(10) = '0' then
  din10 <= "0000";
 else
  din10 <= "0001";
 end if;
```

```
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(11) = '0' then
  din11 <= "0000";
 else
  din11 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(12) = '0' then
  din12 <= "0000";
 else
  din12 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(13) = '0' then
  din13 <= "0000";
 else
  din13 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(14) = '0' then
  din14 <= "0000";
 else
  din14 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DIN(15) = '0' then
  din15 <= "0000";
 else
  din15 <= "0001";
 end if;
end if;

if rising_edge(dev_clock_5MHz)then
 if DATA(0) = '0' then
  data0 <= "0000";
 else
  data0 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(1) = '0' then
  data1 <= "0000";
 else
  data1 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(2) = '0' then
  data2 <= "0000";
 else
  data2 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(3) = '0' then
  data3 <= "0000";
 else
  data3 <= "0001";
```

```
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(4) = '0' then
  data4 <= "0000";
 else
  data4 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(5) = '0' then
  data5 <= "0000";
 else
  data5 <= "0001";
 end if;
end if;if rising_edge(dev_clock_5MHz)then
 if DATA(6) = '0' then
  data6 <= "0000";
 else
  data6 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(7) = '0' then
  data7 <= "0000";
 else
  data7 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(8) = '0' then
  data8 <= "0000";
 else
  data8 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(9) = '0' then
  data9 <= "0000";
 else
  data9 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(10) = '0' then
  data10 <= "0000";
 else
  data10 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(11) = '0' then
  data11 <= "0000";
 else
  data11 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(12) = '0' then
  data12 <= "0000";
 else
  data12 <= "0001";
 end if;
```

```
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(13) = '0' then
  data13 <= "0000";
 else
  data13 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(14) = '0' then
  data14 <= "0000";
 else
  data14 <= "0001";
 end if;
end if;
if rising_edge(dev_clock_5MHz)then
 if DATA(15) = '0' then
  data15 <= "0000";
 else
  data15 <= "0001";
 end if;
end if;
end process;
end Behavioral;
```

9.4.11 Mac – LCD

```
--------------------------------------------------------------------------------
-- Instanz MAC-LCD legt die Daten mit einem genauen Zeitverhalten an die
-- Ausgangspins, damit sie korrekt dargestellt werden.
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity MAC_LCD is
PORT (
            dev_clock_25MHz         : in STD_LOGIC;
            status                  : in STD_LOGIC;
            lcd_dat                 : out STD_LOGIC_VECTOR (3 downto 0);
            lcd_e                   : out STD_LOGIC;
            lcd_rw                  : out STD_LOGIC;
            count                   : in STD_LOGIC_VECTOR (14 downto 0);
            lcd_rs                  : out STD_LOGIC;
            noflowbits1             : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits2             : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits3             : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits4             : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits5             : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits6             : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits7             : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits8             : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits9             : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits10            : in STD_LOGIC_VECTOR (3 downto 0);
            noflowbits11            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits1            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits2            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits3            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits4            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits5            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits6            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits7            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits8            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits9            : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits10           : in STD_LOGIC_VECTOR (3 downto 0);
            nofhighbits11           : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits1             : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits2             : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits3             : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits4             : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits5             : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits6             : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits7             : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits8             : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits9             : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits10            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdlowbits11            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits1            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits2            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits3            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits4            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits5            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits6            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits7            : in STD_LOGIC_VECTOR (3 downto 0);
```

```
            nsdhighbits8            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits9            : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits10           : in STD_LOGIC_VECTOR (3 downto 0);
            nsdhighbits11           : in STD_LOGIC_VECTOR (3 downto 0);
            din0                    : in STD_LOGIC_VECTOR (3 downto 0);
            din1                    : in STD_LOGIC_VECTOR (3 downto 0);
            din2                    : in STD_LOGIC_VECTOR (3 downto 0);
            din3                    : in STD_LOGIC_VECTOR (3 downto 0);
            din4                    : in STD_LOGIC_VECTOR (3 downto 0);
            din5                    : in STD_LOGIC_VECTOR (3 downto 0);
            din6                    : in STD_LOGIC_VECTOR (3 downto 0);
            din7                    : in STD_LOGIC_VECTOR (3 downto 0);
            din8                    : in STD_LOGIC_VECTOR (3 downto 0);
            din9                    : in STD_LOGIC_VECTOR (3 downto 0);
            din10                   : in STD_LOGIC_VECTOR (3 downto 0);
            din11                   : in STD_LOGIC_VECTOR (3 downto 0);
            din12                   : in STD_LOGIC_VECTOR (3 downto 0);
            din13                   : in STD_LOGIC_VECTOR (3 downto 0);
            din14                   : in STD_LOGIC_VECTOR (3 downto 0);
            din15                   : in STD_LOGIC_VECTOR (3 downto 0);
            data0                   : in STD_LOGIC_VECTOR (3 downto 0);
            data1                   : in STD_LOGIC_VECTOR (3 downto 0);
            data2                   : in STD_LOGIC_VECTOR (3 downto 0);
            data3                   : in STD_LOGIC_VECTOR (3 downto 0);
            data4                   : in STD_LOGIC_VECTOR (3 downto 0);
            data5                   : in STD_LOGIC_VECTOR (3 downto 0);
            data6                   : in STD_LOGIC_VECTOR (3 downto 0);
            data7                   : in STD_LOGIC_VECTOR (3 downto 0);
            data8                   : in STD_LOGIC_VECTOR (3 downto 0);
            data9                   : in STD_LOGIC_VECTOR (3 downto 0);
            data10                  : in STD_LOGIC_VECTOR (3 downto 0);
            data11                  : in STD_LOGIC_VECTOR (3 downto 0);
            data12                  : in STD_LOGIC_VECTOR (3 downto 0);
            data13                  : in STD_LOGIC_VECTOR (3 downto 0);
            data14                  : in STD_LOGIC_VECTOR (3 downto 0);
            data15                  : in STD_LOGIC_VECTOR (3 downto 0)
 );
end MAC_LCD;

architecture Behavioral of MAC_LCD is

begin
process (dev_clock_25MHz)
 variable Init_LCD_finished: STD_LOGIC := '0';
 begin
if rising_edge (dev_clock_25MHz) then
 if Init_LCD_finished = '0' Then
 case count is
 when "0000010111101110" => lcd_dat <= "0011";   ----------
 lcd_e <= '0';
 lcd_rs <= '0';
 lcd_rw <= '0';
 when "0000010111101111" => lcd_e <= '1'; ----- 0x30 Init Function Set 751
 when "0000011110111101" => lcd_e <= '1'; ----- 0x30 Init Function Set 957
 when "0000011111000011" => lcd_e <= '1'; ----- 0x30 Init Function Set 963
 when "0000011111000110" => lcd_dat <= "0010";  ---------
 when "0000011111000111" => lcd_e <= '1'; --    0x20 Function Set 967
 when "0000011111001010" => lcd_dat <= "0000";
 when "0000011111001011" => lcd_e <= '1';       ----------
 when "0000011111001110" => lcd_dat <= "0010";  ----------
 when "0000011111001111" => lcd_e <= '1';       -- 0x28 Display Off
 when "0000011111010001" => lcd_dat <= "1000";  --
 when "0000011111010010" => lcd_e <= '1';       ----------
```

```
when "000001111010100" => lcd_dat <= "0000";  ----------
when "000001111010101" => lcd_e <= '1';       --    0x06 Display Clear
when "000001111010110" => lcd_dat <= "0110";  --
when "000001111010111" => lcd_e <= '1';       ----------
when "000001111011001" => lcd_dat <= "0000";  ----------
when "000001111011010" => lcd_e <= '1';       --    0x0c Entry Mode Set
when "000001111011100" => lcd_dat <= "1100";  --
when "000001111011101" => lcd_e <= '1';       ----------
when "111111111111111" => Init_LCD_finished := '1';
when others => lcd_e <= '0';
end case;



elsif Init_LCD_finished = '1' then
if status = '1' then
case count is

when "000000000000000" => lcd_dat <= "0000";
                          lcd_rs <= '0';
when "000000000000011" => lcd_e <= '1';
when "000000000001110" => lcd_dat <= "0001";
when "000000000010010" => lcd_e <= '1'; --------- CURSOR HOME
-------------------------------------------------------------
when "000011000000101" => lcd_rs <= '1';
when "000011000001000" => lcd_dat <= "0100";
when "000011000001010" => lcd_e <= '1';
when "000011000001100" => lcd_dat <= "1110";
when "000011000010000" => lcd_e <= '1'; ----- N number
-------------------------------------------------------------
when "000011000011000" => lcd_dat <= "0110";
when "000011000011011" => lcd_e <= '1';
when "000011000011110" => lcd_dat <= "1111";
when "000011000100010" => lcd_e <= '1'; --------- o of
-------------------------------------------------------------
when "000011000101000" => lcd_dat <= "0100";
when "000011000101011" => lcd_e <= '1';
when "000011000101111" => lcd_dat <= "0110";
when "000011000110010" => lcd_e <= '1'; --------- F failures
-------------------------------------------------------------
when "000011000110100" => lcd_dat <= "0011";
when "000011000111000" => lcd_e <= '1';
when "000011000111010" => lcd_dat <= "1010";
when "000011000111100" => lcd_e <= '1'; --------- :
-------------------------------------------------------------
when "000011000111110" => lcd_dat <= "0010";
when "000011001000001" => lcd_e <= '1';
when "000011001000100" => lcd_dat <= "0000";
when "000011001000111" => lcd_e <= '1';
-------------------------------------------------------------
when "000011001001001" => lcd_dat <= nofhighbits1;
when "000011001001011" => lcd_e <= '1';
when "000011001001101" => lcd_dat <= noflowbits1    ;
when "000011001001111" => lcd_e <= '1';
-------------------------------------------------------------
when "000011001010001" => lcd_dat <= nofhighbits2;
when "000011001010011" => lcd_e <= '1';
when "000011001010101" => lcd_dat <=  noflowbits2   ;
when "000011001010111" => lcd_e <= '1';
-------------------------------------------------------------
when "000011001011001" => lcd_dat <= nofhighbits3;
when "000011001011011" => lcd_e <= '1';
```

```
when "000011001011101" => lcd_dat <=  noflowbits3   ;
when "000011001011111" => lcd_e <= '1';
-------------------------------------------------------
when "000011001100011" => lcd_dat <= nofhighbits4;
when "000011001100110" => lcd_e <= '1';
when "000011001101001" => lcd_dat <=  noflowbits4   ;
when "000011001101011" => lcd_e <= '1';
-------------------------------------------------------
when "000011001110000" => lcd_dat <= nofhighbits5;
when "000011001110010" => lcd_e <= '1';
when "000011001110100" => lcd_dat <= noflowbits5;
when "000011001110111" => lcd_e <= '1';
-------------------------------------------------------
when "000011010000000" => lcd_dat <= nofhighbits6;
when "000011010000011" => lcd_e <= '1';
when "000011010000101" => lcd_dat <= noflowbits6;
when "000011010001000" => lcd_e <= '1';
-------------------------------------------------------
when "000011010001010" => lcd_dat <= nofhighbits7;
when "000011010001101" => lcd_e <= '1';
when "000011010001111" => lcd_dat <= noflowbits7;
when "000011010010010" => lcd_e <= '1';
-------------------------------------------------------
when "000011010010100" => lcd_dat <= nofhighbits8;
when "000011010011010" => lcd_e <= '1';
when "000011010011101" => lcd_dat <= noflowbits8;
when "000011010100010" => lcd_e <= '1';
-------------------------------------------------------
when "000011010100110" => lcd_dat <= nofhighbits9;
when "000011010101001" => lcd_e <= '1';
when "000011010101100" => lcd_dat <= noflowbits9;
when "000011010101111" => lcd_e <= '1';
-------------------------------------------------------
when "000011010110010" => lcd_dat <= nofhighbits10;
when "000011010110100" => lcd_e <= '1';
when "000011010110111" => lcd_dat <= noflowbits10;
when "000011010111010" => lcd_e <= '1';
-------------------------------------------------------
when "000011010111101" => lcd_dat <= nofhighbits11;
when "000011011000000" => lcd_e <= '1';
when "000011011000011" => lcd_dat <= noflowbits11;
when "000011011000110" => lcd_e <= '1';
-------------------------------------------------------
when "000011111000111" => lcd_rs <= '0';
when "000011111001001" => lcd_dat <= "1100"; --springt in naechste zeile
when "000011111001100" => lcd_e <= '1';
when "000011111001111" => lcd_dat <= "0000";
when "000011111010011" => lcd_e <= '1';
when "000011111010101" => lcd_rs <=  '1';
-------------------------------------------------------
when "000101000000101" => lcd_rs <= '1';
when "000101000001000" => lcd_dat <= "0100";
when "000101000001010" => lcd_e <= '1';
when "000101000001100" => lcd_dat <= "1110";
when "000101000010000" => lcd_e <= '1'; ----- N number
-------------------------------------------------------
when "000101000011000" => lcd_dat <= "0101";
when "000101000011011" => lcd_e <= '1';
when "000101000011110" => lcd_dat <= "0011";
when "000101000100010" => lcd_e <= '1'; --------- S send
-------------------------------------------------------
when "000101000101000" => lcd_dat <= "0100";
when "000101000101011" => lcd_e <= '1';
```

```
when "000101000101111" => lcd_dat <= "0100";
when "000101000110010" => lcd_e <= '1'; --------- D data
-------------------------------------------------------
when "000101000110100" => lcd_dat <= "0011";
when "000101000111000" => lcd_e <= '1';
when "000101000111010" => lcd_dat <= "1010";
when "000101000111100" => lcd_e <= '1';
-------------------------------------------------------
when "000101000111110" => lcd_dat <= "0010";
when "000101001000001" => lcd_e <= '1';
when "000101001000100" => lcd_dat <= "0000";
when "000101001000111" => lcd_e <= '1';
-------------------------------------------------------
when "000101001001001" => lcd_dat <= nsdhighbits1;
when "000101001001011" => lcd_e <= '1';
when "000101001001101" => lcd_dat <= nsdlowbits1;
when "000101001001111" => lcd_e <= '1';
-------------------------------------------------------
when "000101001010001" => lcd_dat <= nsdhighbits2;
when "000101001010011" => lcd_e <= '1';
when "000101001010101" => lcd_dat <= nsdlowbits2;
when "000101001010111" => lcd_e <= '1';
-------------------------------------------------------
when "000101001011001" => lcd_dat <= nsdhighbits3;
when "000101001011011" => lcd_e <= '1';
when "000101001011101" => lcd_dat <= nsdlowbits3;
when "000101001011111" => lcd_e <= '1';
-------------------------------------------------------
when "000101001100011" => lcd_dat <= nsdhighbits4;
when "000101001100110" => lcd_e <= '1';
when "000101001101001" => lcd_dat <= nsdlowbits4;
when "000101001101011" => lcd_e <= '1';
-------------------------------------------------------
when "000101001110000" => lcd_dat <= nsdhighbits5;
when "000101001110010" => lcd_e <= '1';
when "000101001110100" => lcd_dat <= nsdlowbits5;
when "000101001110111" => lcd_e <= '1';
-------------------------------------------------------
when "000101010000000" => lcd_dat <= nsdhighbits6;
when "000101010000011" => lcd_e <= '1';
when "000101010000101" => lcd_dat <= nsdlowbits6;
when "000101010001000" => lcd_e <= '1';
-------------------------------------------------------
when "000101010001010" => lcd_dat <= nsdhighbits7;
when "000101010001101" => lcd_e <= '1';
when "000101010001111" => lcd_dat <= nsdlowbits7;
when "000101010010010" => lcd_e <= '1';
-------------------------------------------------------
when "000101010010100" => lcd_dat <= nsdhighbits8;
when "000101010011010" => lcd_e <= '1';
when "000101010011101" => lcd_dat <= nsdlowbits8;
when "000101010100010" => lcd_e <= '1';
-------------------------------------------------------
when "000101010100110" => lcd_dat <= nsdhighbits9;
when "000101010101001" => lcd_e <= '1';
when "000101010101100" => lcd_dat <= nsdlowbits9;
when "000101010101111" => lcd_e <= '1';
-------------------------------------------------------
when "000101010110010" => lcd_dat <= nsdhighbits10;
when "000101010110100" => lcd_e <= '1';
when "000101010110111" => lcd_dat <= nsdlowbits10;
when "000101010111010" => lcd_e <= '1';
-------------------------------------------------------
```

```
when "000101010111101" => lcd_dat <= nsdhighbits11;
when "000101011000000" => lcd_e <= '1';
when "000101011000011" => lcd_dat <= nsdlowbits11;
when "000101011000110" => lcd_e <= '1';
-------------------------------------------------------
when "111111111111111" => Init_LCD_finished := '1';
when others => lcd_e <= '0';

end case;
else
case count is

when "000000000000000" => lcd_dat <= "0000";
                          lcd_rs <= '0';
when "000000000000011" => lcd_e <= '1';
when "000000000001110" => lcd_dat <= "0001";
when "000000000010010" => lcd_e <= '1'; --------- CURSOR HOME
-------------------------------------------------------
when "000011000000101" => lcd_rs <= '1';
when "000011000001000" => lcd_dat <= "0011";
when "000011000001010" => lcd_e <= '1';
when "000011000001100" => lcd_dat <= din0;
when "000011000010000" => lcd_e <= '1';
-------------------------------------------------------
when "000011000011000" => lcd_dat <= "0011";
when "000011000011011" => lcd_e <= '1';
when "000011000011110" => lcd_dat <= din1;
when "000011000100010" => lcd_e <= '1';
-------------------------------------------------------
when "000011000101000" => lcd_dat <= "0011";
when "000011000101011" => lcd_e <= '1';
when "000011000101111" => lcd_dat <= din2;
when "000011000110010" => lcd_e <= '1';
-------------------------------------------------------
when "000011000110100" => lcd_dat <= "0011";
when "000011000111000" => lcd_e <= '1';
when "000011000111010" => lcd_dat <= din3;
when "000011000111100" => lcd_e <= '1';
-------------------------------------------------------
when "000011000111110" => lcd_dat <= "0011";
when "000011001000001" => lcd_e <= '1';
when "000011001000100" => lcd_dat <= din4;
when "000011001000111" => lcd_e <= '1';
-------------------------------------------------------
when "000011001001001" => lcd_dat <= "0011";
when "000011001001011" => lcd_e <= '1';
when "000011001001101" => lcd_dat <= din5;
when "000011001001111" => lcd_e <= '1';
-------------------------------------------------------
when "000011001010001" => lcd_dat <= "0011";
when "000011001010011" => lcd_e <= '1';
when "000011001010101" => lcd_dat <=  din6;
when "000011001010111" => lcd_e <= '1';
-------------------------------------------------------
when "000011001011001" => lcd_dat <= "0011";
when "000011001011011" => lcd_e <= '1';
when "000011001011101" => lcd_dat <=  din7;
when "000011001011111" => lcd_e <= '1';
-------------------------------------------------------
when "000011001100011" => lcd_dat <= "0011";
when "000011001100110" => lcd_e <= '1';
when "000011001101001" => lcd_dat <=  din8;
when "000011001101011" => lcd_e <= '1';
```

```
-------------------------------------------------------
when "000011001110000" => lcd_dat <= "0011";
when "000011001110010" => lcd_e <= '1';
when "000011001110100" => lcd_dat <= din9;
when "000011001110111" => lcd_e <= '1';
-------------------------------------------------------
when "000011010000000" => lcd_dat <= "0011";
when "000011010000011" => lcd_e <= '1';
when "000011010000101" => lcd_dat <= din10;
when "000011010001000" => lcd_e <= '1';
-------------------------------------------------------
when "000011010001010" => lcd_dat <= "0011";
when "000011010001101" => lcd_e <= '1';
when "000011010001111" => lcd_dat <= din11;
when "000011010010010" => lcd_e <= '1';
-------------------------------------------------------
when "000011010010100" => lcd_dat <= "0011";
when "000011010011010" => lcd_e <= '1';
when "000011010011101" => lcd_dat <= din12;
when "000011010100010" => lcd_e <= '1';
-------------------------------------------------------
when "000011010100110" => lcd_dat <= "0011";
when "000011010101001" => lcd_e <= '1';
when "000011010101100" => lcd_dat <= din13;
when "000011010101111" => lcd_e <= '1';
-------------------------------------------------------
when "000011010110010" => lcd_dat <= "0011";
when "000011010110100" => lcd_e <= '1';
when "000011010110111" => lcd_dat <= din14;
when "000011010111010" => lcd_e <= '1';
-------------------------------------------------------
when "000011010111101" => lcd_dat <= "0011";
when "000011011000000" => lcd_e <= '1';
when "000011011000011" => lcd_dat <= din15;
when "000011011000110" => lcd_e <= '1';
-------------------------------------------------------
when "000011111000111" => lcd_rs <= '0';
when "000011111001001" => lcd_dat <= "1100"; --springt in naechste zeile
when "000011111001100" => lcd_e <= '1';
when "000011111001111" => lcd_dat <= "0000";
when "000011111010011" => lcd_e <= '1';
when "000011111010101" => lcd_rs <=  '1';
-------------------------------------------------------
when "000101000000101" => lcd_rs <= '1';
when "000101000001000" => lcd_dat <= "0011";
when "000101000001010" => lcd_e <= '1';
when "000101000001100" => lcd_dat <= data0;
when "000101000010000" => lcd_e <= '1';
-------------------------------------------------------
when "000101000011000" => lcd_dat <= "0011";
when "000101000011011" => lcd_e <= '1';
when "000101000011110" => lcd_dat <= data1;
when "000101000100010" => lcd_e <= '1';
-------------------------------------------------------
when "000101000101000" => lcd_dat <= "0011";
when "000101000101011" => lcd_e <= '1';
when "000101000101111" => lcd_dat <= data2;
when "000101000110010" => lcd_e <= '1';
-------------------------------------------------------
when "000101000110100" => lcd_dat <= "0011";
when "000101000111000" => lcd_e <= '1';
when "000101000111010" => lcd_dat <= data3;
when "000101000111100" => lcd_e <= '1';
```

```
-------------------------------------------------------
when "000101000111110" => lcd_dat <= "0011";
when "000101001000001" => lcd_e <= '1';
when "000101001000100" => lcd_dat <= data4;
when "000101001000111" => lcd_e <= '1';
-------------------------------------------------------
when "000101001001001" => lcd_dat <= "0011";
when "000101001001011" => lcd_e <= '1';
when "000101001001101" => lcd_dat <= data5;
when "000101001001111" => lcd_e <= '1';
-------------------------------------------------------
when "000101001010001" => lcd_dat <= "0011";
when "000101001010011" => lcd_e <= '1';
when "000101001010101" => lcd_dat <= data6;
when "000101001010111" => lcd_e <= '1';
-------------------------------------------------------
when "000101001011001" => lcd_dat <= "0011";
when "000101001011011" => lcd_e <= '1';
when "000101001011101" => lcd_dat <= data7;
when "000101001011111" => lcd_e <= '1';
-------------------------------------------------------
when "000101001100011" => lcd_dat <= "0011";
when "000101001100110" => lcd_e <= '1';
when "000101001101001" => lcd_dat <= data8;
when "000101001101011" => lcd_e <= '1';
-------------------------------------------------------
when "000101001110000" => lcd_dat <= "0011";
when "000101001110010" => lcd_e <= '1';
when "000101001110100" => lcd_dat <= data9;
when "000101001110111" => lcd_e <= '1';
-------------------------------------------------------
when "000101010000000" => lcd_dat <= "0011";
when "000101010000011" => lcd_e <= '1';
when "000101010000101" => lcd_dat <= data10;
when "000101010001000" => lcd_e <= '1';
-------------------------------------------------------
when "000101010001010" => lcd_dat <= "0011";
when "000101010001101" => lcd_e <= '1';
when "000101010001111" => lcd_dat <= data11;
when "000101010010010" => lcd_e <= '1';
-------------------------------------------------------
when "000101010010100" => lcd_dat <= "0011";
when "000101010011010" => lcd_e <= '1';
when "000101010011101" => lcd_dat <= data12;
when "000101010100010" => lcd_e <= '1';
-------------------------------------------------------
when "000101010100110" => lcd_dat <= "0011";
when "000101010101001" => lcd_e <= '1';
when "000101010101100" => lcd_dat <= data13;
when "000101010101111" => lcd_e <= '1';
-------------------------------------------------------
when "000101010110010" => lcd_dat <= "0011";
when "000101010110100" => lcd_e <= '1';
when "000101010110111" => lcd_dat <= data14;
when "000101010111010" => lcd_e <= '1';
-------------------------------------------------------
when "000101010111101" => lcd_dat <= "0011";
when "000101011000000" => lcd_e <= '1';
when "000101011000011" => lcd_dat <= data15;
when "000101011000110" => lcd_e <= '1';
-------------------------------------------------------
when "111111111111111" => Init_LCD_finished := '1';
when others => lcd_e <= '0';
```

```
 end case;
 end if;
 end if;
end if;

 end process;
end Behavioral;
```

9.4.12 Status LCD

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


entity Status_LCD is
PORT (
            taster                  : in  STD_LOGIC;
            status                  : inout STD_LOGIC
 );
end Status_LCD;

architecture Behavioral of Status_LCD is

begin

 Process(taster)
 begin
 if (rising_edge (taster)) then
      status <= not (status);
 end if;
 end process;



end Behavioral;
```

9.4.13 Counter

```
----------------------------------------------------------------------------
-- Zähler für LCD
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity COUNTER_LCD is
PORT (
            dev_clock_25kHz         : in STD_LOGIC;
            count                   : inout STD_LOGIC_VECTOR (14 downto 0)
 );
end COUNTER_LCD;

architecture Behavioral of COUNTER_LCD is

begin
process (dev_clock_25kHz)
 begin
 if rising_edge (dev_clock_25kHz) THEN
      count <= count + '1';
 end if;
end process;

end Behavioral;
```

9.4.14 Initserialiser

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

entity INITSERIALIZER is
      PORT(
            Sync                    : inout     STD_LOGIC;
            Local_Le                : inout     STD_LOGIC;
            Den                     : inout     STD_LOGIC;
            Tpwdn                   : inout     STD_LOGIC;
            Lock                    : inout     STD_LOGIC;
            Line_Le                 : inout     STD_LOGIC;
            Ren                     : inout     STD_LOGIC;
            Rpwdn                   : inout     STD_LOGIC;
            dev_clock_25kHz         : in        STD_LOGIC
      );
end INITSERIALIZER;

architecture Behavioral of INITSERIALIZER is

begin
process(dev_clock_25kHz)
begin

                  Sync        <= '0';
                  Local_Le    <= '0';
                  Den         <= '1';
                  Tpwdn       <= '1';
                  Lock        <= '1';
                  Line_Le     <= '0';
                  Ren         <= '1';
                  Rpwdn       <= '1';

end process;
end Behavioral;
```

10 Quellen- & Abbildungsverzeichnisse

10.1 Literaturverzeichnis

[ETS] Etschberger, Konrad: Controller – Area- Network: Grundlagen, Protokolle, Bausteine, Anwendungen; 2. Auflage (2000); Carl Hanser Verlag; ISBN: 3-446-19431-2

[GAE] Gärtner, Prof. Dr.-Ing. Uwe F.: Hochfrequenztechnik I – Teil 1.: Manuskriptentwurf (2001)

[GOE] Göbel, Jürgen: Kommunikationstechnik: Grundlagen und Anwendungen; 1. Auflage (1999); Hüthig Verlag, ISBN: 3-7785-3904-3

[KRI] Kriesel, Prof. Dr. habil. Werner; Heimbold, Dr.-Ing. Tilo; Telschow, Dipl.-Ing. Telschow: Bustechnologien für die Automation: Vernetzung, Auswahl und Anwendung von Kommunikationssystemen; 2. Auflage (2000); Hüthig Verlag; ISBN: 3-7785-2778-9

[LIN] Lindner, Helmut: Physik für Ingenieure, 16. Auflage (2003); Fachbuchverlag; ISBN: 3-44621703-7

[MES] Messmer, Hans Peter: PC-Hardwarebuch: Aufbau, Funktionsweise, Programmierung, 6. Auflage (2003); Addison Wesley Verlag; ISBN: 3-8273-1461-5

[REI] Reichardt, Prof. Dr. rer. Nat. Jürgen; Schwarz, Prof. Dr.-Ing. Bernd: VHDL-Synthese, Entwurf digitaler Schaltungen und Systeme; 3. Auflage (2003);Oldenburg Verlag; ISBN: 3-486-27384-1

[SIK] Sikora, Prof. Dr.-Ing. Dipl.-Ing. Dipl.-Wirt.-Ing. Axel: Technische Grundlagen der Rechner – kommunikation: Internet-Protokolle und Anwendungen; 1. Auflage (2003); Fachbuchverlag Hanser; ISBN: 3-446-22455-6

[SCH] Schleicher, Manfred: Digitale Schnittstellen und Bussysteme: Grundlagen und praktische Hinweise zur Anbindung von Feldgeräten; 1. Auflage (2008); JUMO Verlag; ISBN: 978-3-935742-02-3

[SC1] Schwanger, Prof. Dr. Jürgen: Ethernet erreicht das Feld: Sechs Echtzeit-Varianten im Vergleich – Teil 1 in: Elektronik: Fachzeitschrift für industrielle Anwender und Entwickler; Ausgabe 11/2004; WEKA Fachzeitschriften Verlag; Seiten 48 – 54

[SC2] Schwanger, Prof. Dr. Jürgen: Ethernet erreicht das Feld: Sechs Echtzeit-Varianten im Vergleich – Teil 2 in: Elektronik: Fachzeitschrift für industrielle Anwender und Entwickler; Ausgabe 13/2004; WEKA Fachzeitschriften Verlag; Seiten 38 – 43

[SWI] Schnell, Prof. Dr. Ing. Gerhard; Wiedemann, Dipl.-Ing. Bernhard: Bussysteme in der Automatisierungstechnik und Prozesstechnik: Grundlagen, Systeme und Trends der industriellen Kommunikation; 6. Auflage (2006); Friedrich Vieweg & Sohn Verlag; ISBN: 3-8348-0045-7

[TAN] Tanenbaum, Andrew: Computernetzwerke; 4.Auflage (2003); Pearson Verlag; ISBN: 3-8273-7046-9

[URB] Urbanski, Prof. Dr.-Ing. Klaus; Woitowitz, Dr.-Ing. Roland: Digitaltechnik, Ein Lehr- und Übungsbuch; 4. Auflage (2003); Springer Verlag; ISBN: 3-540-40180-6

[ZAI] Zainalabedin, Navabi: VHDL, Analysis and Modeling of Digital Systems; 2. Auflage (1998); McGraw Hill Companies; ISBN: 0-07-046479-0

10.2 Internetquellen

[CIE] Cieply, Nina; Fachbericht: ISO / OSI-Referenzmodell: Allgemeine Beschreibung des ISO / OSI-Modells der Datenübertragung und seine Motivation; Abruf: 26. Mai 2010
URL: http://www.ninacieply.de/ausbildung/OSI_Referenzmodell.pdf

[KLE] Kleine, Matthias; Das OSI-Referenzmodell; Abruf: 26. Mai 2010
URL: http://www.selflinux.org/selflinux-devel/pdf/osi.pdf

[MEM] ME-Meßsysteme GmbH, Grundlagen zum CAN – Bus; Abruf 12.Juni 2010
URL: http://www.me-systeme.de/de/basics/kb-canbus.pdf

[SCP] Schnabel, Patrick; Elektronik Kompendium; Abruf: 01. Juli 2010
URL: http://www.elektronik-kompendium.de/sites/net/1406171.htm

10.3 Abbildungsverzeichnis