Masterthesis

Im Fachbereich

Informatik und Kommunikationssysteme der Hochschule Merseburg (FH)

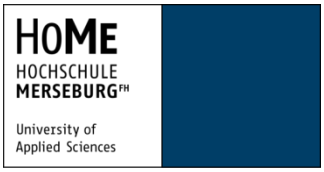
zum Thema

Hard- und Softwareentwicklung eines RGB-LED-Cubes mittels FPGA-Entwicklungstools und VHDL

Hardware and Software engineering of a RGB- LED cube with the aid of a FPGA development board and VHDL

---

eingereicht von:	Oliver Heumos, B.Eng., 18854
geboren am:	01.03.1983 in Zwenkau
abgegeben am:	Merseburg, den 16.12.2014
Erstprüfer:	Prof. Dr.-Ing. Rüdiger Klein
Zweitprüfer:	M.Eng. Ulrich Borchert
Fachbereich:	IKS
Studienrichtung:	Informatik und Kommunikationssysteme (MIKS)



Inhaltsverzeichnis

Abbildungsverzeichnis

Abbildungen ohne Quellenangaben wurden von dem Autor selbst erstellt.

Tabellenverzeichnis

Formelverzeichnis

Quellcodeverzeichnis

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

..... ……

Ort, Datum Unterschrift

Zusammenfassung

Im Rahmen der Masterarbeit des Masterstudienganges „Informatik und Kommunikationssysteme“ im Modul Nachrichtentechnik ist eine dreidimensionale Leuchtdioden-Farbmatrix realisiert worden. Diese dient zur Darstellung von visuellen Informationen, welche mit Hilfe von Effekten, zum Beispiel Farbspiele und Tiefenwirkungen, dargestellt werden.

Der Leser erhält einen Einblick über die Hardwareumsetzung einer Anzeige, deren Ansteuerung und einer Softwareumsetzung. Die Funktionsweise einzelner Hardwareelemente, deren Vor- und Nachteile sowie Anwendungsbeispiele werden in Text und Bild erläutert. Auf der im Anhang hinterlegten DVD sind Programmbeispiele in VHDL für einen FPGA zur Ansteuerung des Cubes vorhanden. Diese können unter Verwendung der in der Dokumentation beschriebenen Entwicklungsboards, angewandt und beliebig verändert werden.

Summary

Within the scope of the master thesis for the masters course "Informatics and Communication Systems", in the module Telecommunications, a three dimensional light-emitting diode colour matrix was realised.

It functions as a portrayal of visual information which is represented with help of effects such as colour play and depth effect.

The reader gains insight into the hardware transcriptions of a display, its selection, as well as a software transcription.

The functional principle of individual hardware elements, its advantages and disadvantages plus application examples are elucidated with video and text material. On the attached DVD are sample programs in VHDL for a FPGA to control the cube. These programs can be applied and arbitrary changes made by using the development boards being explained in the documentary.

1 Einleitung und Motivation

Leuchtdioden (LEDs) sind aktive Bauelemente. Sie verfügen über eine hohe Leuchtkraft und besitzen eine hohe Energieeffizienz. Für eine Realisierung eines Cubes werden mehrere LEDs in einer entsprechenden Form angeordnet und miteinander verbunden. Dies ermöglicht visuelle Darstellungen von dreidimensionalen Bildern.

Angesteuert wird der Cube durch einen Field Programmable Gate Array (FPGA) auf einem Entwicklungsboard zusammen mit der Hardwarebeschreibungssprache Very High Speed Integrated Circuit Hardware Description Language (VHDL). Die Ansteuerung erfolgt direkt über ein Inter Integrated Circuit (I^2C) Zwei-Draht-Bussystem auf einem „General-purpose input/output“ (GPIO) -Anschluss. Für die Verwendung des I^2C-Bussystems auf einem FPGA ist ein sogenannter Master-Controller erforderlich, welcher in dieser Dokumentation in VHDL vorliegt.

Der GPIO-Anschluss in Kombination mit dem I^2C-Bussystem ermöglicht es, den Cube an einem Embedded System, zum Beispiel einem Rasberry Pi, zu betreiben. Eine Programmierung kann ebenfalls mit der Programmiersprache JAVA erfolgen.

1.1 Aufgabenstellung

In diesem Modul soll eine eigenhändig gebaute 3D-Matrix in Form eines Würfels realisiert werden. Dieser besteht aus 512 Leuchtdioden (LEDs) jeweils in den Farben Rot, Grün und Blau (RGB), welche als RGB-LEDs bezeichnet werden. Die Ansteuerung erfolgt mittels einem FPGA auf einem oder mehreren Entwicklungsboards zusammen mit der Hardwarebeschreibungssprache VHDL.

1.2 Zielbeschreibung

Für die Umsetzung der Aufgabenstellung ist zunächst eine Auswahl von Hardwarekomponenten erforderlich. Diese Elemente stehen in einer direkten Beziehung mit der Dimensionierung und Konstruktion der 3D-Matrix. Für eine Softwareumsetzung stehen FPGA-Entwicklungsboards der Firma Altera und Altium zur Verfügung, welche mit VHDL beschrieben werden. Damit der zu erzeugende VHDL-Code auf unterschiedlichen FPGAs funktioniert, ist eine Einhaltung des VHDL-Standards 2008 vorgesehen. Eine Kommunikation zwischen dem jeweiligen Entwicklungsboard und dem Cube erfolgt über einen GPIO-Anschluss. Diese Boards verfügen über eine begrenzte Anzahl an GPIO-PINs. Für deren Erweiterung sind sogenannte GPIO-Expander, welche über ein Zwei-Draht-Bussystem angesteuert werden, vorgesehen. Für das Bussystem ist ein Master-Core in VHDL notwendig.

2 Angewandte Hardwarekomponenten

Dieses Kapitel gibt einen Einblick in die im Projekt angewandten Hardware-komponenten.

2.1 Elektrische Leiter

Elektrische Leiter werden in drei Kategorien eingeteilt. Diese lauten elektrischer Leiter, Halbleiter und Nichtleiter.

2.1.1 Elektrischer Leiter

Ein elektrischer Leiter ist ein Stoff, welcher durch seine chemischen und physikalischen Eigenschaften freibewegliche Ladungsträger besitzt und zum Transport geladener Teilchen (elektrischer Strom) benutzt werden kann. Für eine fortführende Recherche dienen die Referenzen [1,2].

2.1.2 Elektrischer Nichtleiter

Elektrische Nichtleiter sind Stoffe, welche aus physikalischer und chemischer Zusammensetzung keine elektrische Energie leiten. Sie werden fortführend als Isolatoren bezeichnet [1,3].

2.1.3 Elektrischer Halbleiter

Elektrische Halbleiter sind elektronische Bauelemente, welche sich durch bestimmte chemische und physikalische Eigenschaften auszeichnen. Sie können nur unter bestimmten Voraussetzungen in einen elektrisch leitenden Zustand überführt werden [1,4,5].

Die elektronische Leitfähigkeit obliegt unter anderen folgenden Abhängigkeiten:

- mechanische Kraft
- Temperatur
- Belichtung
- zugefügte Fremdstoffe

2.2 Halbleiter-Dotierung

Eine Dotierung ist eine gezielte Veränderung der Leitfähigkeit von Halbleiterbauelementen. Bei diesem Vorgang werden Fremdatome in einen vorhandenen Halbleiterwerkstoff eingebaut. Dabei wird in zwei Dotierungen unterschieden.

Bei einem n-dotierten Halbleiter wird der Halbleiterwerkstoff mit einem Donator (Atom) verunreinigt. Das Donatoratom schenkt dem Werkstoff ein zusätzliches Elektron. Dieses ist frei und kann zur Entstehung des Stromflusses beitragen. Durch diese zusätzlichen Donatoren wird die Leitfähigkeit des Halbleiters verändert [6]. Die folgende Abbildung 001 verdeutlicht den Elektronenfluss von Minus nach Plus in einem n-dotierten-Halbleiter.

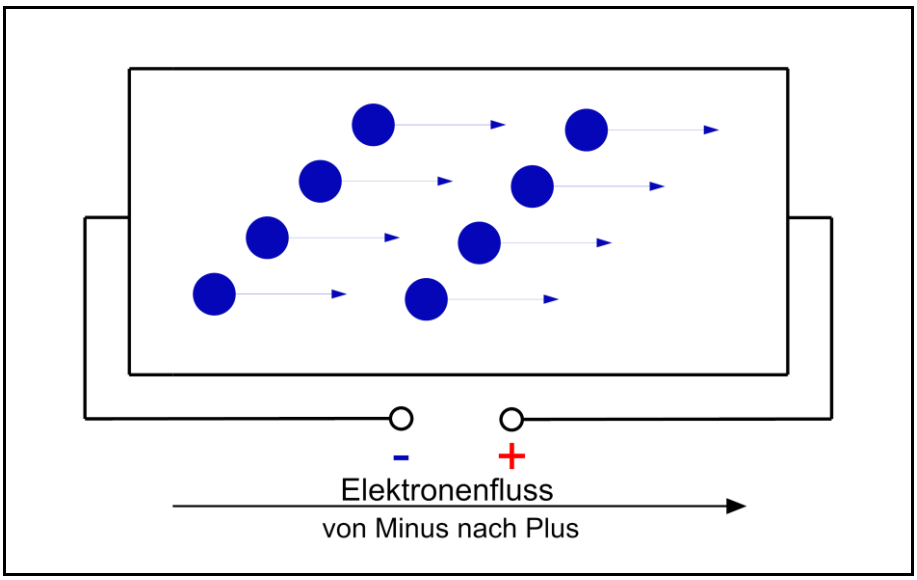


Abbildung 001 - Elektronenfluss eines n-dotierten Halbleiters bearbeitet unter Verwendung von [6]

Wird eine Spannungsquelle an den n-Leiter angeschlossen, so entzieht der positiv geladene Pol dem n-Leiter die Elektronen. Dadurch entsteht ein Elektronenstrom von Minus nach Plus.

Bei einem p-dotierten Halbleiter wird der Halbleiterwerkstoff mit einem Akzeptor (Atom) verunreinigt. Dem Akzeptoratom fehlt ein Elektron (Defektelektron). Dies führt in der Werkstoffstruktur zu dem sogenannten Elektronenloch. Existiert eine thermische Bewegung von Elektronen, werden diese von den Löchern angezogen und die offenen Stellen besetzt, wobei neue Löcher entstehen [6]. Die folgende Abbildung 002 verdeutlicht die Funktionsweise eines p-dotierten-Halbleiters.

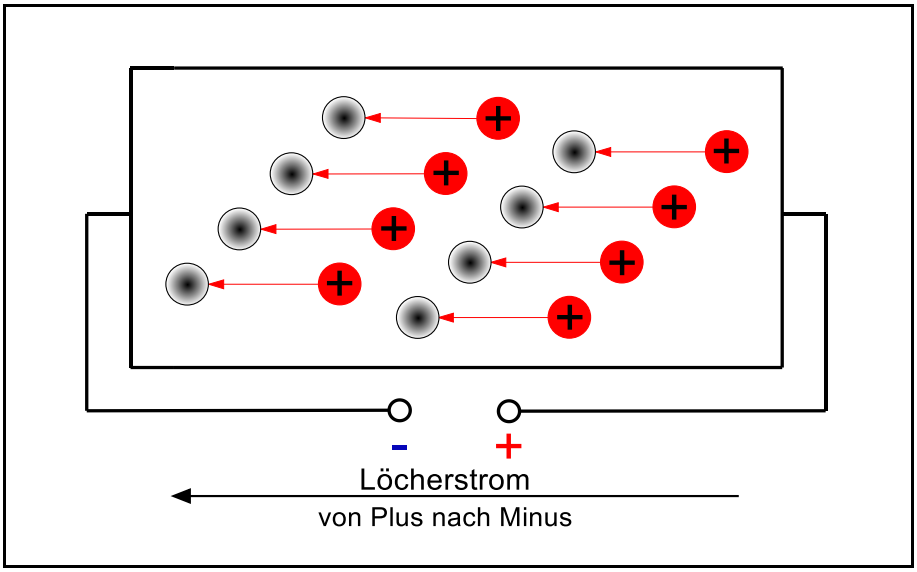


Abbildung 002 - Elektronenfluss eines p-dotierten Halbleiters bearbeitet unter Verwendung von [6]

Wird eine Spannungsquelle angelegt, wandern die Elektronen zum negativ geladenen Pol und ziehen dabei Elektronen in die freien Löcher, während der positiv geladene Pol dem p-Leiter erneut Elektronen entzieht. Bei diesem Effekt wird aus dem Isolator ein Leiter.

2.3 Metall-Oxid-Feldeffekt-Transistor

Aus der Funktionsweise der englischen Begriffe *transfer* für Bewegung und *resistor* für Widerstand setzt sich der Begriff Transistor zusammen. Er ist ein aktives Bauelement und wird als elektronischer Schalter (elektrischer Widerstand) oder Verstärker (elektronische Quelle) angewandt. Feldeffekttransistoren mit einer isolierten Gate-Elektrode (IGFET) sind unter der Bezeichnung Metall-Oxid-Feldeffekt-Transistoren (MOSFET) bekannt [7,8].

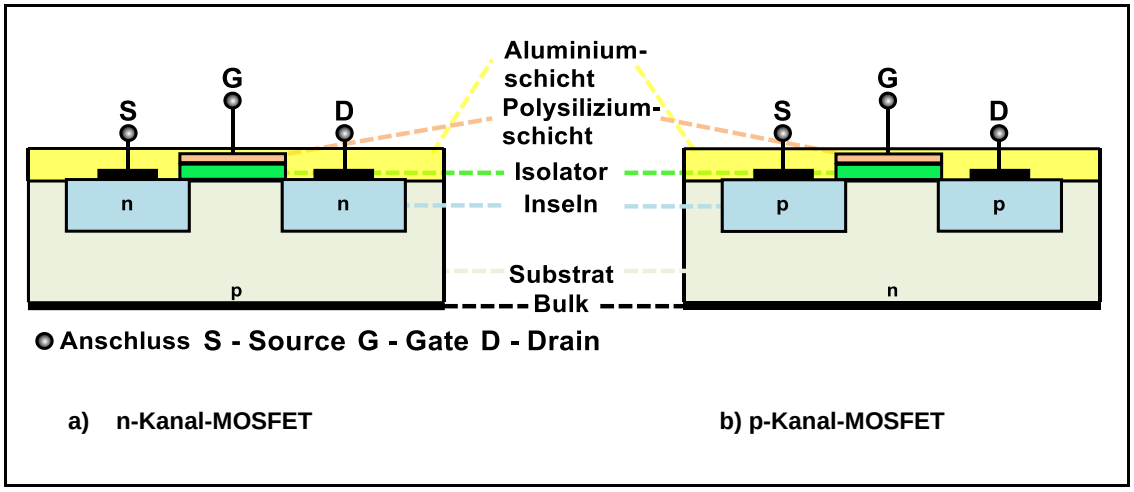


Abbildung 003 - Aufbau eines MOSFETs

Die Abbildung 003 zeigt jeweils den Aufbau eines MOSFETs. Dieser besteht aus einem Basismaterial (Silizium-Schicht p bzw. n), welches das Substrat bildet. Zwei Elektroden werden durch Ionenimplantation eingebracht. Unter Verwendung der Halbleitermaterialien wird in zwei MOSFET-Typen unterschieden. Besteht das Substrat aus einem p-Halbleitermaterial, beinhaltet dieses zwei n-dotierte Inseln. Dieser Typ wird als n-Kanal-MOSFET bezeichnet. Bei einem p-Kanal-MOSFET sind zwei p-dotierte Inseln in einem n-Halbleiter-Substrat vorhanden. Zwischen den Inseln und auf dem Substrat befindet sich eine dünne Isolationsschicht, welche meist aus Silizium-Dioxid besteht. Auf dieser ist eine dünne Aluminiumschicht aufgedampft, welche mit einer Polysiliziumschicht auf das Objekt aufgebracht ist. Diese beinhaltet Löcher bzw. Anschlüsse für elektrische Kontakte. Der MOSFET besitzt meist vier elektrische Anschlüsse bestehend aus Source S (Quelle), Drain D (Abfluss), Gate G (Tor) und einem Bulk (B).

Der Bulk ist meist intern mit dem Source-Anschluss verbunden. Für eine fortführende Recherche zur Herstellung eines MOSFETs dienen die Referenzen [7-9].

Von der Bauart abhängig wird ein MOSFET in selbstleitende (Verarmungstyp) und selbstsperrende (Anreicherungstyp) Typen unterschieden. Wird die Gate-Elektrode ohne Spannung betrieben sind Anreicherungstypen selbstsperrende Elemente. Diese werden leitend, sobald eine positive Spannung (*pinch off*) angelegt wird. Anzumerken ist, dass eine Berührung ausreicht, um eine Überschreitung der Durchschlags-spannung auszulösen. Der Verarmungstyp leitet bei niedriger Spannung von zum Beispiel $0V$. Wird dieser mit einer negativen Spannung betrieben, erfolgt eine Sperrung des Kanals. Dies kann zu Beschädigungen der Bauelemente führen. Je nach Einsatzbereich eines MOSFETS ist gegebenenfalls eine Schutzschaltung empfehlenswert.

In der Praxis sind n-Kanal-MOSFETs vorwiegend als Verarmungstyp und p-Kanal-MOSFETs als Anreichungstyp zu finden.

2.3.1 Funktionsweise n-Kanal-MOSFETs

Die folgende Abbildung 004 dient zur Veranschaulichung der Funktionsweise eines n-Kanal-Verarmungstyp-MOSFETs.

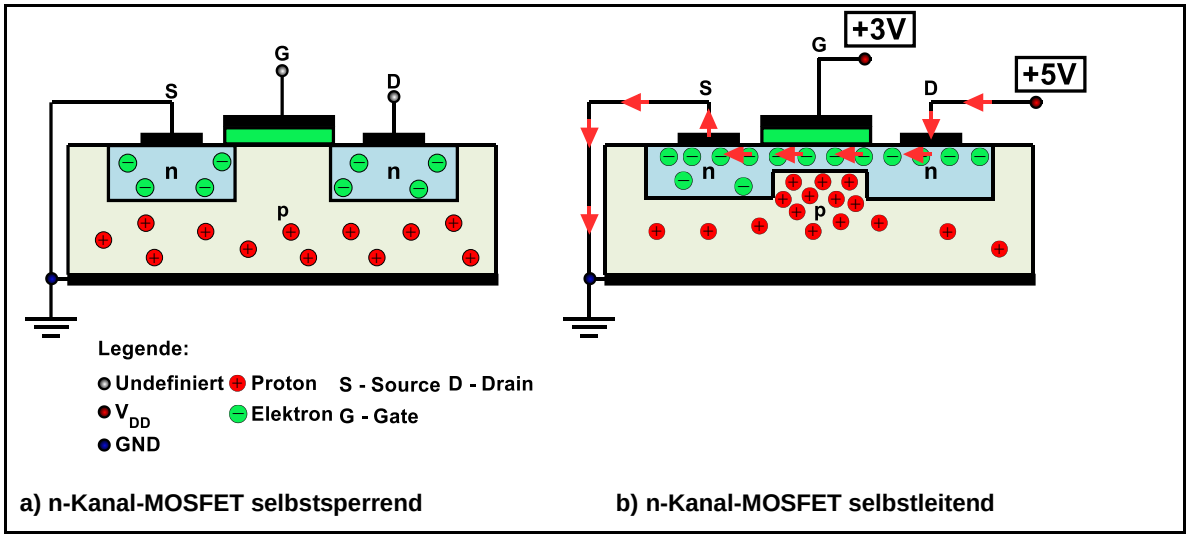


Abbildung 004 - Funktionsweise eines n-Kanal-MOSFET

Der Gate ist durch eine dünne Isolationsschicht vom Drain-Source-Kanal getrennt. Dadurch fließt kein Strom und der MOSFET kann leistungslos gesteuert werden (a). Erfolgt ein Anlegen einer Spannung, so werden die positiv geladenen Ladungsträger vom p-dotierten Substrat an- und negative Ladungsträger (Löcher) abgestoßen. Es bildet sich ein elektrisches Feld. Bei diesem Prozess werden n-dotierte Elektroden angeregt und es entsteht ein n-leitender Kanal. Mit einer Erhöhung der Gate-Spannung wird das elektrische Feld vergrößert. Das Funktionsprinzip wird in der nachfolgenden Abbildung 005 erläutert.

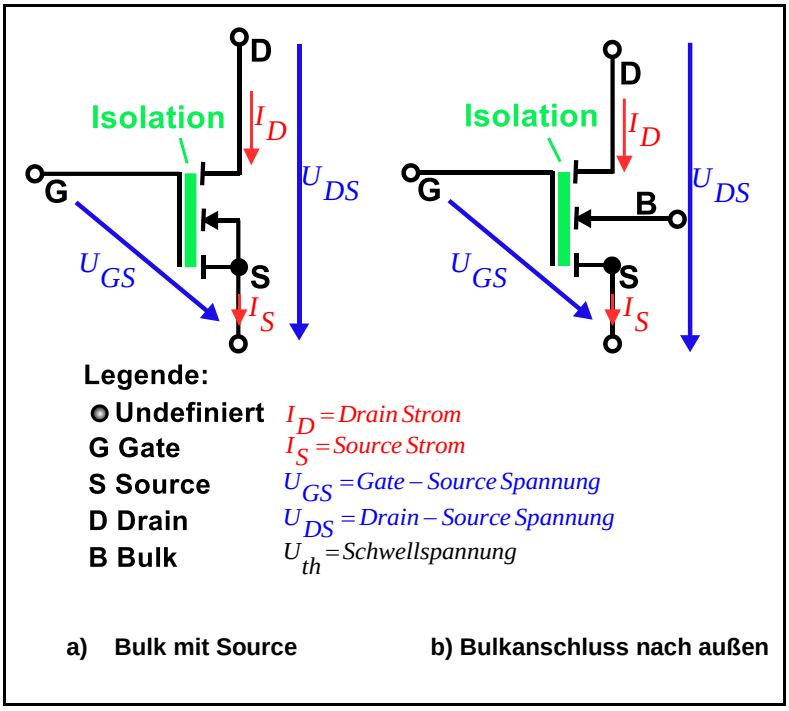


Abbildung 005 - Funktionsweise eines n-Kanal-MOSFET am Schaltbild

Über eine Gate-Source-Spannung U_{GS} kann die Größenordnung des Widerstandes zwischen *drain* und *source* R_{DS} und dadurch der Strom I_{DS} verändert werden. Wird eine Spannung zwischen Gate und Source größer als die Schwellspannung (*threshold voltage* U_{th}), beginnt ein Strom im Drain-Source-Kanal zu fließen. Die Schwellspannung ist vom MOSFET-Typ abhängig und liegt bei Niedervolt-MOSFETs meist zwischen $(0{,}8 \ldots 2)\mathrm{V}$ und bei Hochvolt-MOSFETs zwischen $(2 \ldots 6)\mathrm{V}$. Die Wirkungsweise eines n-Kanal-MOSFETs wird anhand folgender Abbildung 006 am Beispiel eines Leuchtmittels, welches durch diesen gesteuert wird, verdeutlicht.

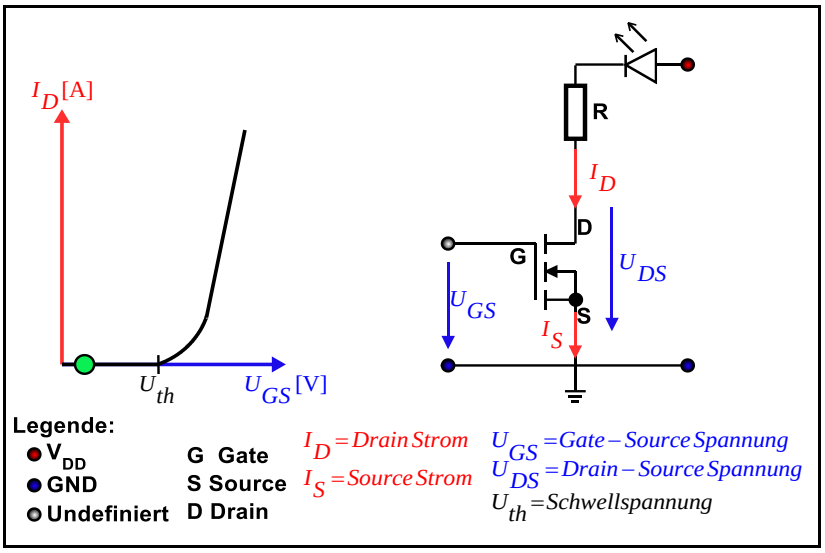


Abbildung 006 - n-Kanal-MOSFET Eingangskennlinie der Gate-Source-Spannung im Sperrbereich

Bei dieser Beispielschaltung wird eine Konstantstromquelle zwischen Gate- und Source-Anschluss geschalten. Eine weitere Stromquelle dient für die Versorgung des Leuchtmittels und wird ebenfalls mit dem GND verbunden. Liegt keine Gate-Source-Spannung ($U_{GS} = 0\,V$, $U_{GS} < U_{th}$) an, fließt kein Drain-Strom ($I_D = 0\,A$). Der MOSFET isoliert und besitzt einen sehr hohen Innenwiderstand R_{DS}. Eine anliegende Spannung zwischen Leuchtmittel und GND ist größer null Volt ($U_{GS} > 0\,V$). Es ist kein Stromkreis vorhanden und das Leuchtmittel ist aus.

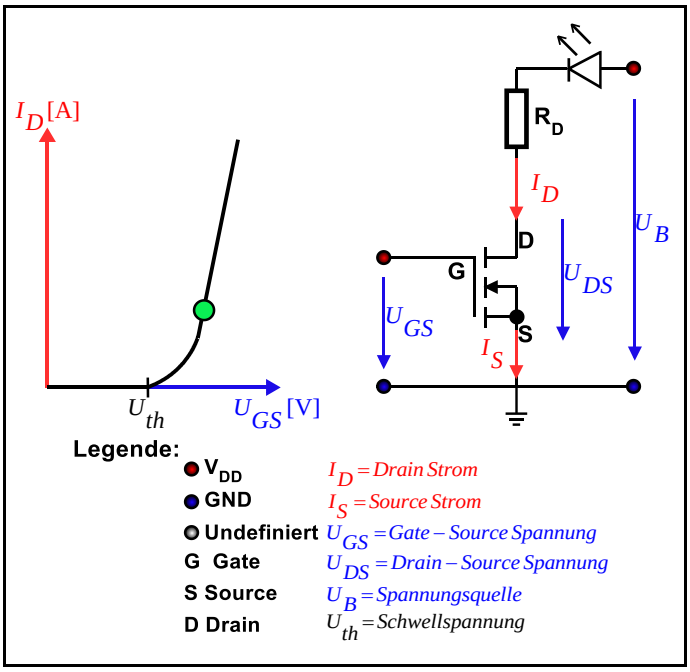


Abbildung 007 - n-Kanal-MOSFET Eingangskennlinie Gate-Source-Spannung im Durchlassbereich

Eine Überführung aus dem hochohmigen in einen niederohmigen Zustand wird erreicht, indem die Schwellspannung U_{th} mit zunehmender Gate-Source-Spannung überschritten wird $(U_{GS} > U_{th})$. Dabei entsteht ein Elektronenfluss, welcher einen aktiven Stromkreislauf bildet.

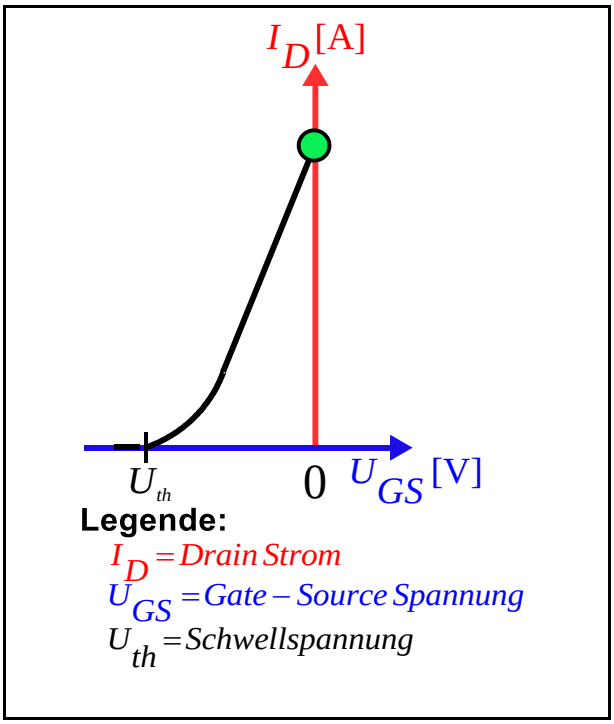


Abbildung 008 - n-Kanal-MOSFET Kennlinie Gate-Source-Spannung mit maximalen Drain-Strom

Durch den Herstellungsprozess bedingt, liegt bei selbstleitenden MOSFETs bereits eine Steuerspannung von $U_{GS} = 0\,V$ und somit der maximale Stromfluss vor.

Wird das Potential am Gate negativ gegenüber des Source-Anschlusses erhöht, werden freie Ladungsträger aus dem Kanal in das Substrat gezogen. Bei diesem Prozess findet eine Verarmung statt und der Kanal wird verengt. Der Stromfluss zwischen Source und Drain nimmt ab.

Wird die Gate-Sättigungsspannung erreicht, enthält der Kanal keine freien Ladungsträger mehr. Der MOSFET ist hochohmig und es fließt kein Drain-Strom ($I_D = 0\,A$).

2.3.2 Funktionsweise p-Kanal-MOSFETs

Bei einem p-Kanal-MOSFET sind alle Spannungen und Ströme entgegengesetzt des n-Kanal-Typs. Die folgende Abbildung 009 verdeutlicht diesen Zusammenhang.

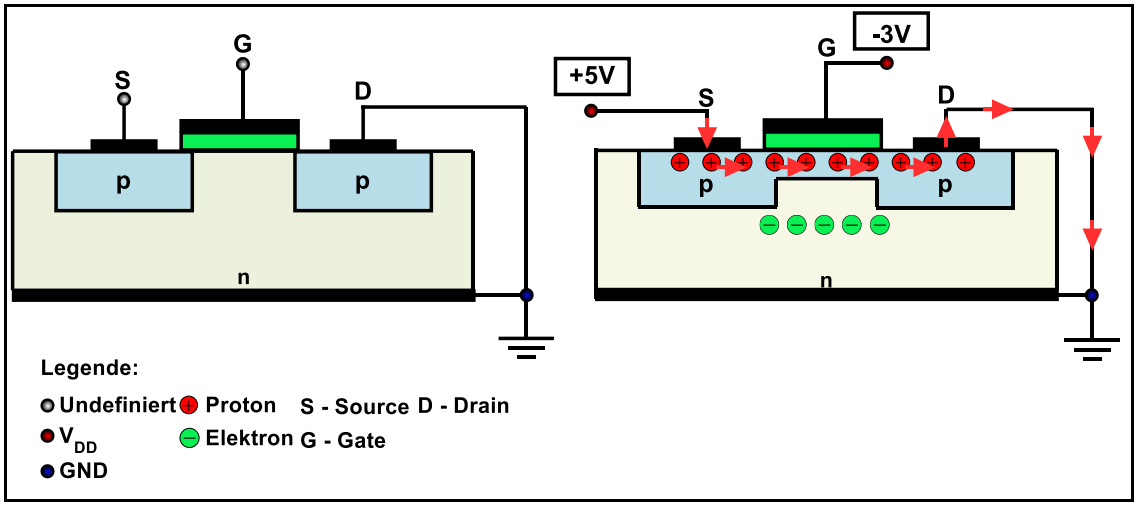


Abbildung 009 - Funktionsweise eines p-Kanal-MOSFET

Für einen Betrieb des p-Kanal-MOSFET-Anreicherungstyps ist eine Inversion einzuleiten. Dies erfolgt, indem eine negative Gate-Source-Spannung $U_{GS} < U_{th}$ anliegt. Die negative Drain-Source-Spannung U_{DS} führt zu einem Stromfluss durch den Transistor.

Die Eigenschaften eines jeden MOSFETs befinden sich im Datenblatt des jeweiligen Herstellers. Ein besonderes Merkmal liegt dabei in der $I_D - U_{DS}$ – Kennlinie. Diese Ausgangskennlinie gibt Auskunft über den Arbeits- und Sperrbereich sowie die benötigte Schwellspannung und den Sättigungsbereich.

2.3.3 Kennlinie eines n-Kanal-MOSFETs

Die folgende Abbildung 010 beinhaltet fiktiv gewählte Werte eines n-Kanal-MOSFETs vom Anreichungstyp.

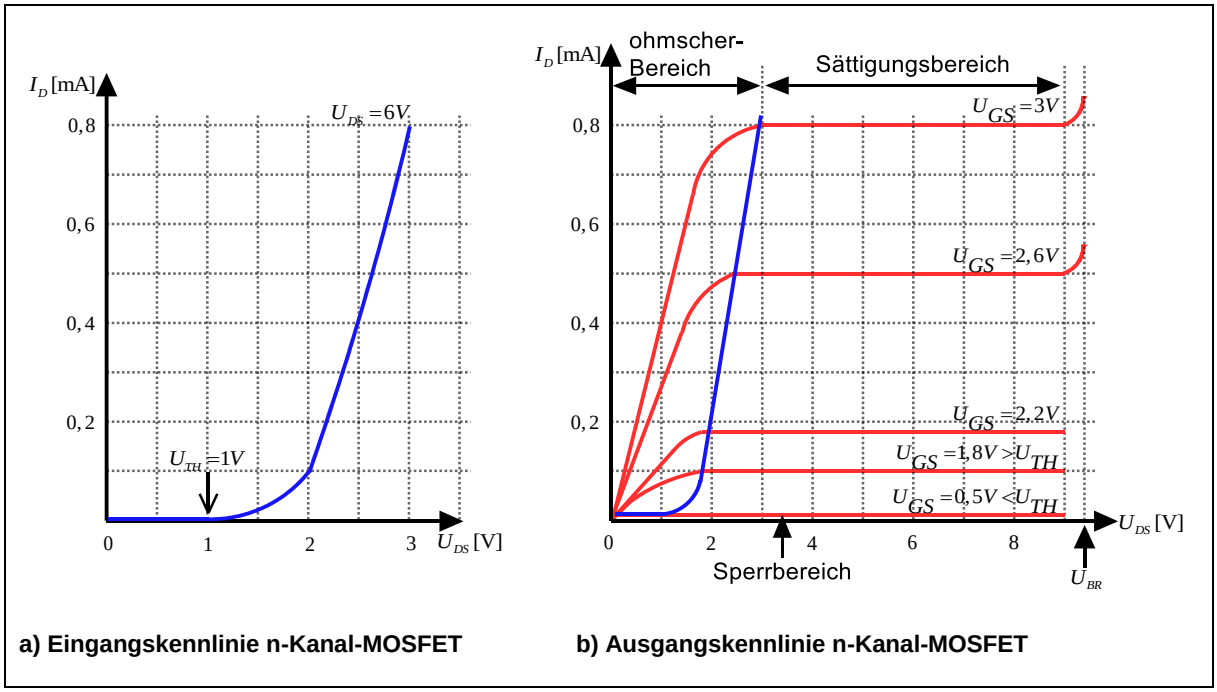


Abbildung 010 - Eingangs- und Ausgangskennlinie eines n-Kanal-MOSFET

Die Abbildung 010 a) zeigt, dass eine Schwellspannung von 1V nötig ist, damit der Kanal geöffnet wird und ein Stromfluss erfolgen kann. b) gibt eine Auskunft über den ohmschen Sättigungs- und Sperrbereich. Ersichtlich ist, dass im ohmschen Bereich der Strom proportional in Abhängigkeit der Gate-Source-Spannung steigt. In diesem öffnet sich der dotierte Kanal. Im Sättigungsbereich ist der Strom nur gering von der Spannung abhängig. Der Kanal hat seine maximal mögliche „Öffnung“ erreicht und es fließt der maximal mögliche Strom.

Wird das Bauelement über seinen größtmöglichen Bereich betrieben, so erfolgt ein Durchbruch der Gate-Kanal-Sperrschicht, welches den Defekt des Transistors bedeutet.

2.3.4 Schaltzeichen n- und p-Kanal-MOSFETs

Nachfolgend werden die Schaltzeichen der n- und p-Kanal-MOSFETS als Anreichungs- und Verarmungstyp aufgeführt.

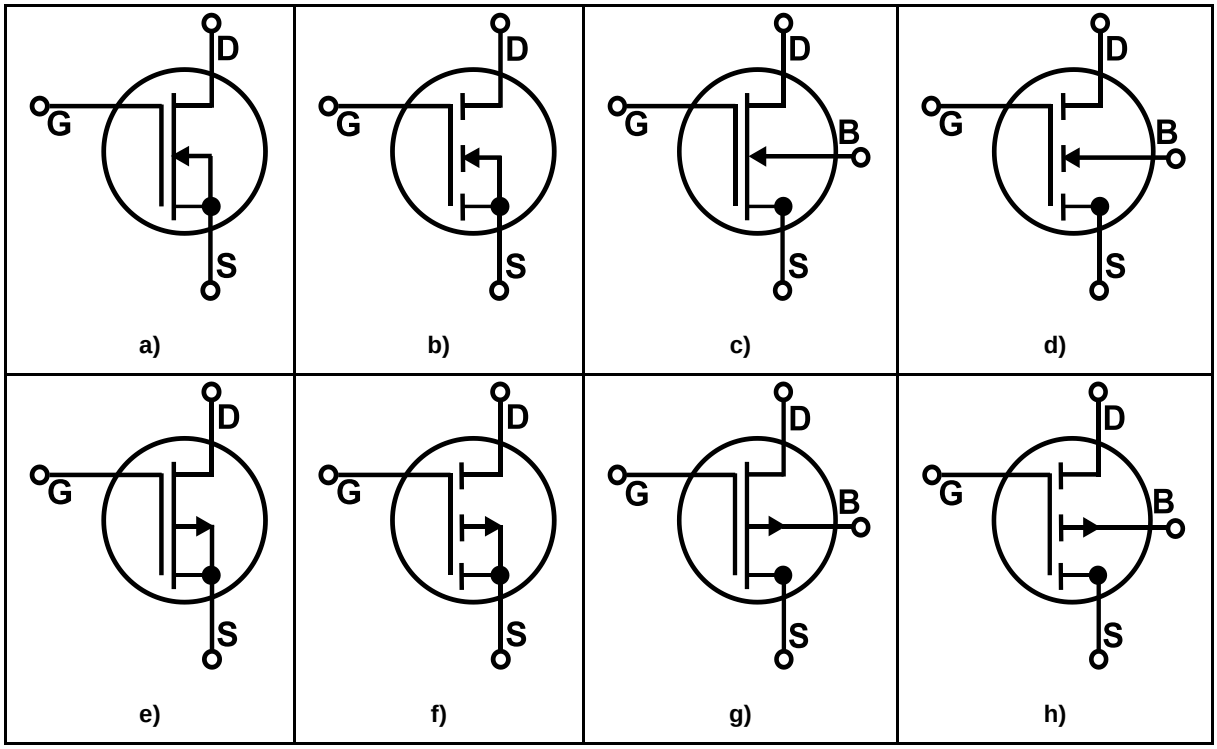


Abbildung 011 - Schaltzeichen der MOSFET-Typen bearbeitet unter Verwendung von [7]

- **a) n-Kanal-MOSFET Verarmungstyp mit verbundenen Bulk-Anschluss**
- **b) n-Kanal-MOSFET Anreicherungstyp mit verbundenen Bulk-Anschluss**
- **c) n-Kanal-MOSFET Verarmungstyp mit offenen Bulk-Anschluss**
- **d) n-Kanal-MOSFET Anreicherungstyp mit offenen Bulk-Anschluss**
- **e) p-Kanal-MOSFET Verarmungstyp mit verbundenen Bulk-Anschluss**
- **f) p-Kanal-MOSFET Anreicherungstyp mit verbundenen Bulk-Anschluss**
- **g) p-Kanal-MOSFET Verarmungstyp mit offenen Bulk-Anschluss**
- **h) p-Kanal-MOSFET Anreicherungstyp mit offenen Bulk-Anschluss**

Für eine fortführende Recherche dienen die Referenzen [7-9].

2.4 Lumineszenzdiode

Die Lumineszenzdiode (*light-emitting diode,* LED, Leuchtdiode) ist eine licht-emittierende Diode. Führende Unternehmen aus der Lichtindustrie haben sich weltweit zu einem Konsortium unter dem Namen „Zhaga“ zusammengeschlossen, um einen einheitlichen Standard für Schnittstellen von LED-Lichtmodulen zu entwickeln. Ziel ist es die fortlaufenden technologischen Entwicklungen schneller voranzutreiben und einen einheitlichen LED-Standard zu entwickeln [10]. Gründungsmitglieder der Zhaga sind unter anderem OSRAM, Philips, Panasonic und andere namhafte Leuchtmittelhersteller [10,11]. Zu den Standardfaktoren gehören:

- Abmessungen
- einheitliche Sockeltechniken
- lichttechnische und elektrische Kenngrößen
- Wärmeverhalten

Eine Leuchtdiode besteht aus einem n- und p-Halbleiter. Auf der n-leitenden Schicht ist eine dünnere p-Schicht aufgebracht.

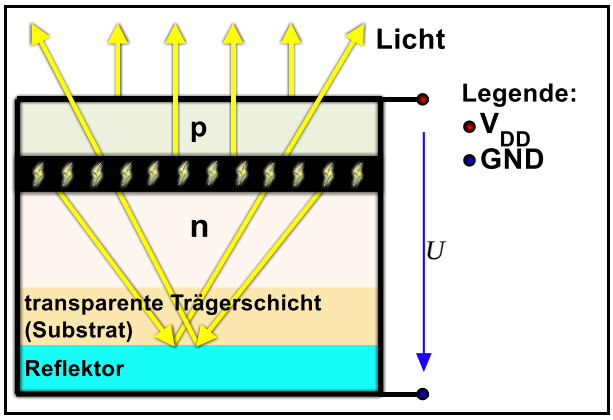


Abbildung 012 - physikalisches Funktionprinzip LED

Bezugnehmend auf das Kapitel 2.2, ist die Grenzschicht mit freien Ladungsträgern überschwemmt.

Durch Anlegen einer Spannung in Durchlassrichtung wandern die Elektronen durch Rekombination, wodurch Energie in Form von Licht am p-n-Übergang freigesetzt wird und aus der dünneren p-Schicht entweichen kann. Durch eine geeignete Auswahl der Halbleitermaterialien und der Dotierung lässt sich die freigesetzte Energie und deren Effizienz beeinflussen. Eine Steigerung der zugeführten Energie bewirkt eine höhere Wärmeentwicklung. Wird eine bestimmte Temperaturgrenze des Halbleiters erreicht, so erfolgt eine Sättigung der Ladungsträger und das Kristallgitter im Halbleiter wird zerstört. Die folgende Abbildung 013 verdeutlicht den Aufbau einer LED.

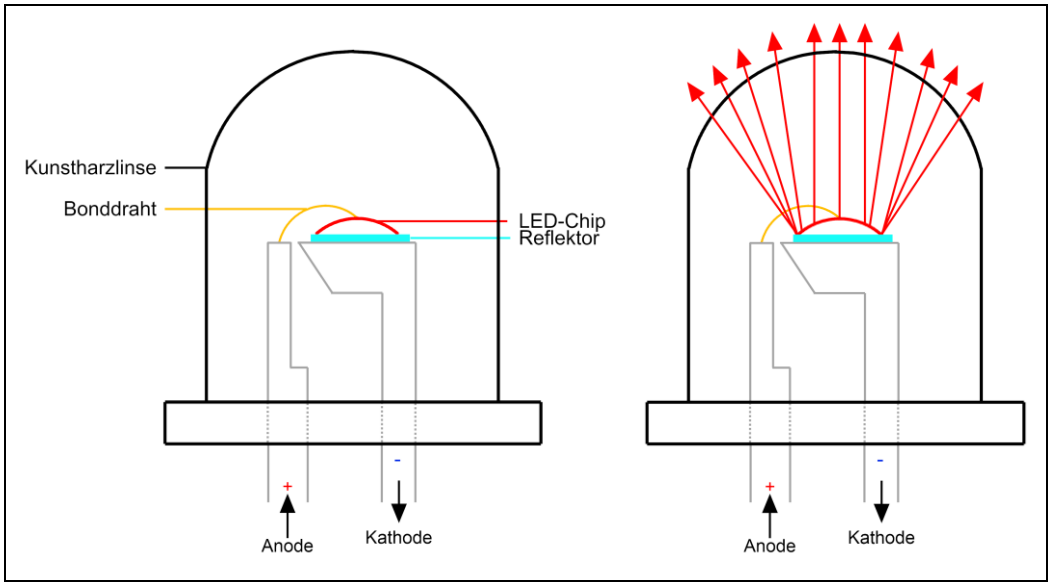


Abbildung 013 - Aufbau einer LED

Eine LED besteht aus Anode, Kathode, LED-Chip (p-n-dotierter-Halbleiter, Substrat), Reflektor, Gehäuse und einem Bonddraht.

Von dem Halbleiterkristall geht eine geringe Lichtstrahlung aus. Um eine höhere Lichtausbeute zu erreichen, wird ein Metall (Reflektor) angebracht, welches meist halbkugelförmig ist. Durch ein linsenförmiges Gehäuse wird das Licht gebündelt und eine bessere Streuung erreicht.

Bei herkömmlichen LEDs besteht das Gehäuse meist aus Kunstharz. Je nach Einsatzbereich kann dieses aus anderen Werkstoffen (zum Beispiel Glas) bestehen. Man unterscheidet klare LEDs und LEDs mit matter Gehäuseoberfläche, sogenannte diffuse LEDs.

Aufgrund dieser Eigenschaften ergibt sich je nach verwendeten Materialien ein unterschiedlicher Öffnungswinkel, woraus sich eine unterschiedliche Lichtausbeute ergibt. Die Angaben der Durchflussspannung U_{LED} und des Durchflussstroms I_{LED} sind im Datenblatt der Leuchtdiode enthalten. Die folgende Tabelle 01 gibt einen Überblick der üblichen Werte.

Standard-LED			
Farbe	Wellenlänge [nm]	U_{LED} [V]	I_{LED} [A]
rot	610 - 760	2,00	0,02
grün	500 - 570	3,00	0,02
blau	450 - 500	3,20	0,02

Tabelle 01 - Wellenlänge, Spannung und Stromstärke einer Standard-LED [12]

Diese Werte sind Richtwerte und können je nach Hersteller und verwendeten Halbleitermaterialien unterschiedlich sein. Ersichtlich ist, dass eine niedrigere Wellenlänge eine höhere Spannung benötigt.

2.4.1 Reihenschaltung

Bei einer Reihenschaltung fließt der gleiche Strom in jedes Bauelement. Die Gesamtspannung U_G ergibt sich aus einem Vorwiderstand R und der LED-Durchflussspannung U_{LED} unter Vernachlässigung der Leitungsverluste.

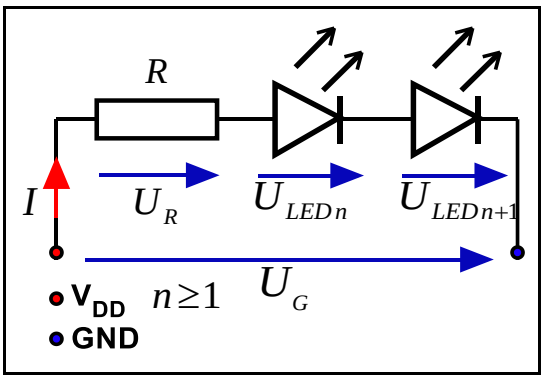


Abbildung 014 - LED-Reihenschaltung

Formel 01 - Ermittlung von U_{LED}

$$U_{LED} = U_{LED\,n} + U_{LED\,n+1} + \ldots \qquad n \geq 1$$

Dabei kann die Gesamtspannung wie folgt ermittelt werden:

Formel 02 - Berechnung der Gesamtspannung einer LED-Reihenschaltung

$$U_G[\mathrm{V}] = U_R[\mathrm{V}] + U_{LED}[\mathrm{V}]$$

Mit Hilfe des Ohmschen Gesetzes erfolgt die Berechnung des Vorwiderstandes nach Formel 03.

Formel 03 - Ermittlung der Widerstandsspannung U_R

$$U_R\,[V] = \frac{R_V[\Omega]}{I\,[A]}$$

Eine Umstellung nach R_V ergibt den gesuchten Widerstand:

Formel 04 - Berechnung des Vorwiderstands R_V

$$R_V[\Omega] = \frac{U_R\,[V]}{I\,[A]}$$

Daraus folgt die Ermittlung der am Vorwiderstand abfallenden Spannung:

Formel 05 - Am Vorwiderstand abfallende Spannung U_R

$$U_R[V] = [\,U_G - U_{LED}]V$$

Durch das Einsetzen der Formel 05 in Formel 04 ergibt sich die vollständige Formel zur Berechnung des Vorwiderstandes wie folgt:

Formel 06 - Berechnung eines LED-Vorwiderstands

$$R_V[\Omega] = \frac{[U_G - U_{LED}]V}{[I]A}$$

Die Wirkleistung (Verlustleistung) $\mathrm{P_V}$ wird aus dem Betriebsstrom und der daraus resultierenden Spannung ermittelt.

Formel 07 - Ermittlung der Verlustleistung (U_R und I bekannt)

$$P_V[W] = U_R[V] \cdot I[A]$$

Ist die Stromstärke nicht bekannt, so kann diese durch die folgende Gleichung errechnet werden:

Formel 08 - Ermittlung der Verlustleistung (U_R und R_V bekannt)

$$P_V\,[W] = \frac{U_R^2\,[V]}{R_V\,[\Omega]}$$

Ist die Stromstärke und die Größe des Vorwiderstandes bekannt, ergibt sich die Verlustleistung:

Formel 09 - Ermittlung der Verlustleistung (I und R_V bekannt)

$$P_V[W] = I^2[A] \cdot R_V[\Omega]$$

Als Beispiel dient ein Stromkreis, an welchem zwei LEDs mit einer Stromstärke von $30\mathrm{mA}$ bei $5\mathrm{V}$ in Reihe geschalten werden.

Gesucht sind der Vorwiderstand und die Verlustleistung der Schaltung, indem die Werte in die Formel 06 und Formel 09 eingesetzt werden.

Die Berechnungen ergeben folgende Ausgangswerte:

$$R_V = \frac{(5{,}00 - 4{,}00)V}{0{,}02A} = 50{,}00\Omega$$

$$P_V = 0{,}02^2A^2 \cdot 50{,}00\Omega = 0{,}02W$$

Ersichtlich ist, dass bei einer Betriebsspannung von 5V zwei LEDs nacheinander betrieben werden können. Bei der Verwendung einer weiteren LED ist die Betriebsspannung zu gering. Die Halbleiter würden nur mit einer schwachen Leuchtkraft leuchten.

2.4.2 Parallelschaltung mit einem Vorwiderstand

Bei einer Parallelschaltung addieren sich die einzelnen Ströme, welche durch jede LED fließen, zu einem Gesamtstrom.

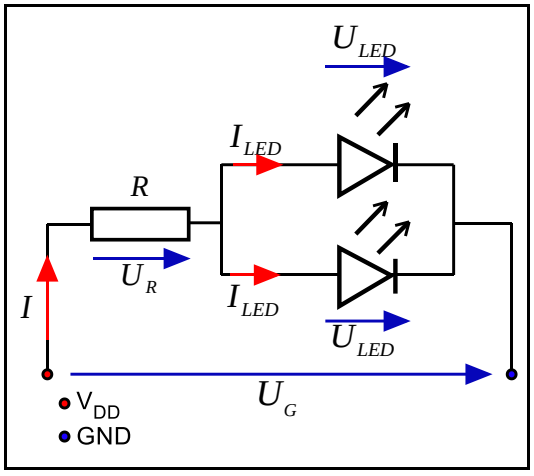


Abbildung 015 - LED-Parallelschaltung mit einem Vorwiderstand

Der zu ermittelte Wert des Vorwiderstandes setzt sich zusammen aus der gesuchten Spannung, welche am Widerstand abfällt, im Verhältnis zu der gewünschten Stromstärke aller parallelen LEDs.

Formel 10 - Berechnung LED-Vorwiderstand einer LED-Parallelschaltung

$$R_V[\Omega] = \frac{[U_R - U_{Led}]V}{(Anzahl\ der\ LEDs) \cdot I_{Led}[A]}$$

Für eine Betriebsspannung von $5\mathrm{V}$ und mehr als zwei LEDs ist eine Parallelschaltung mit einem gemeinsamen Vorwiderstand vorzuziehen. Als Beispiel dient ein Betrieb mit zwei LEDs.

$$R_V = \frac{(5{,}00 - 2.00)V}{(2\ LEDs) \cdot 0{,}02A} = 75{,}00\Omega$$

Das Beispiel verdeutlicht, dass ein Vorwiderstand von 50Ω benötigt wird, damit die LEDs mit $20\mathrm{mA}$ arbeiten.

$$P_V = 3V \cdot [2\ LEDs \cdot 0{,}02A] = 0{,}12W$$

Zu beachten ist, dass LEDs Toleranzen bzgl. ihrer Diodenspannung und Stromstärke aufweisen (Tabelle 01). Die Dioden mit einer geringeren Durchflussspannung benötigen weniger Strom. Die Stromverteilung bei diesem Verfahren erfolgt ungleichmäßig, so dass eine unterschiedliche Lichtausbeute existiert. Anzumerken ist, dass eine zu hohe Durchflussspannung zum Defekt führt.

Desweiteren fällt bei einem Ausfall einer LED mehr Spannung über den Vorwiderstand ab, so dass dieser ggf. unterdimensioniert ist. Daraus folgt eine Überspannung der weiteren Bauelemente.

2.4.3 Parallelschaltung mit jeweils einem Vorwiderstand

Bei dieser werden die LEDs parallel mit jeweils einem Vorwiderstand geschalten.

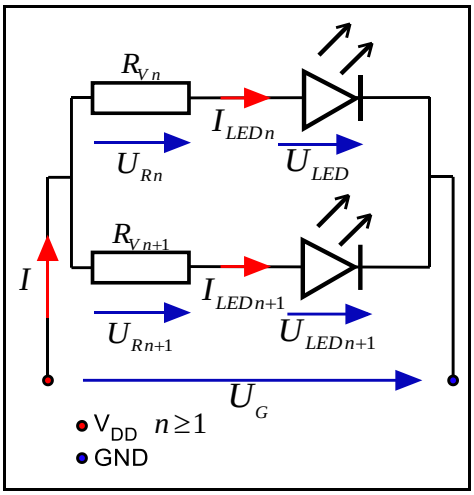


Abbildung 016 - LED-Parallelschaltung mit jeweils einem Vorwiderstand

Die Berechnung des LED-Vorwiderstands einer Parallelschaltung für jeden Zweig ergibt sich aus den folgenden Formeln.

Formel 11 - Berechnung Vorwiderstand einer LED-Parallelschaltung

$$R_{V\,n}\,[\Omega] = \frac{[U_G - U_{LED\,n}]V}{I_{LED\,n}\,[A]} \qquad n \geq 1$$

Formel 12 - Verlustleistung einer LED-Parallelschaltung

$$P_{V\,n}[W] = (I_{LED\,n}[A])^2 \cdot R_{V\,n}[\Omega] \qquad n \geq 1$$

Anhand der vorgestellten Funktionsweise ergeben sich Vor- und Nachteile von LEDs, welche in folgender Tabelle 02 kurz zusammengefasst werden.

Anwendung	Vorteile	Nachteile
Wirtschaft	• geringer Energieverbrauch • geringe Wärmeentwicklung • keine/kaum Wartungskosten • keine/kaum Reinigungskosten • geringe Entsorgungskosten • teilweise Recycling möglich	• hohe Fertigungskosten (stand Juni 2014) • hohe Energiebilanz (Anschaffungskosten größer als Nutzen) • geringe Lebensdauer bei falscher/höherer Energie-belastung
Umwelt	• enthält kein Quecksilber • geringerer Energiebedarf	• Fertigungsaufwand im Verhältnis zum erzeugten Lichtstrom
Technisch	• kein Hohlkörper der implodieren kann • stoß-und vibrationsfest • Farbwiedergabeeigenschaften • kleine Bauformen • regelbare Lichtfarbe	• Lebensdauer je nach Einsatz • Vorschaltgerät bzgl. exakt definierten Strömen und Spannung notwendig

Tabelle 02 - Vor- und Nachteile von LEDs

In folgenden Kapiteln werden unterschiedliche Bauformen mit ihren besonderen Eigenschaften kurz erläutert.

2.4.4 Kurzkopf LEDs

Eine mögliche Bauform ist eine LED mit verkürztem Kopf. Diese werden als Kurz- bzw. als Flachkopf-LED bezeichnet.

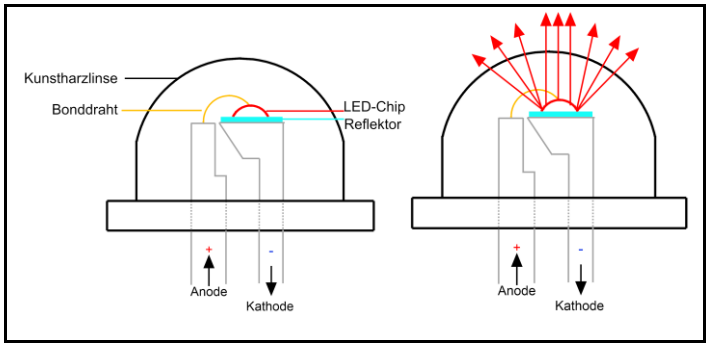


Abbildung 017 - Kurzkopf-LED

Diese bauliche Veränderung ermöglicht einen größeren Streuungswinkel gegenüber einer Standard LED. Laut Herstellerangaben bietet diese Bauform einen Reflexionswinkel von 90° bis 140°. Ein Einsatzbereich ist zum Beispiel das LED-Tagfahrlicht als Leuchtmittel am PKW.

2.4.5 Konkave LEDs

Eine besondere Bauform bietet eine nach innen gewölbte (konkave) Krümmung des Gehäuses einer LED. Die folgende Abbildung 018 verdeutlicht die Konstruktion und die Streuung.

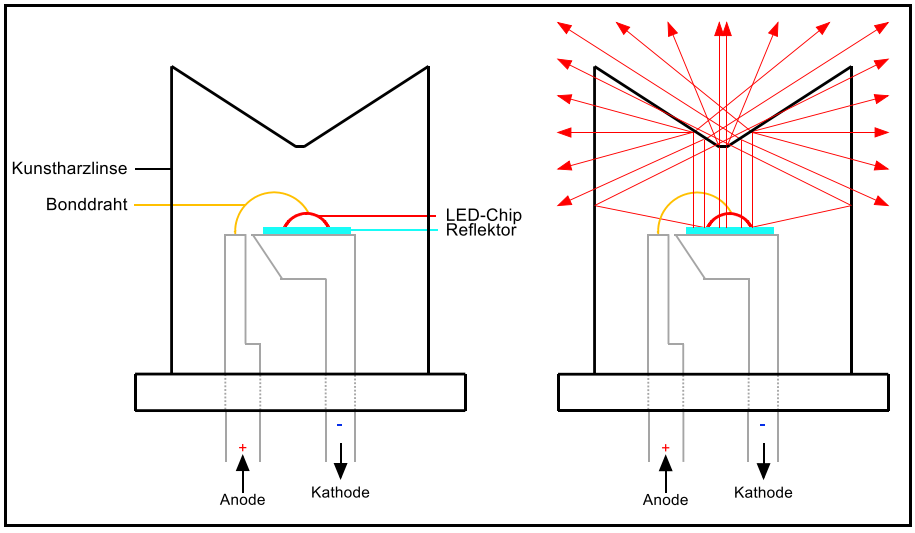


Abbildung 018 - Konkave-LED

Das gebündelte Licht wird an den Ecken und Kanten gebrochen und reflektiert. Laut Herstellerangaben beträgt der Streuungswinkel hier zwischen 90° bis 120°. Durch ihre besondere Bauform eignen sich diese LEDs für eine seitliche Betrachtung.

2.4.6 SMD LEDs

Eine weitere Bauform sind SMD-LEDs. Die folgende Abbildung 019 zeigt den Aufbau und ihre Abstrahlcharakteristik.

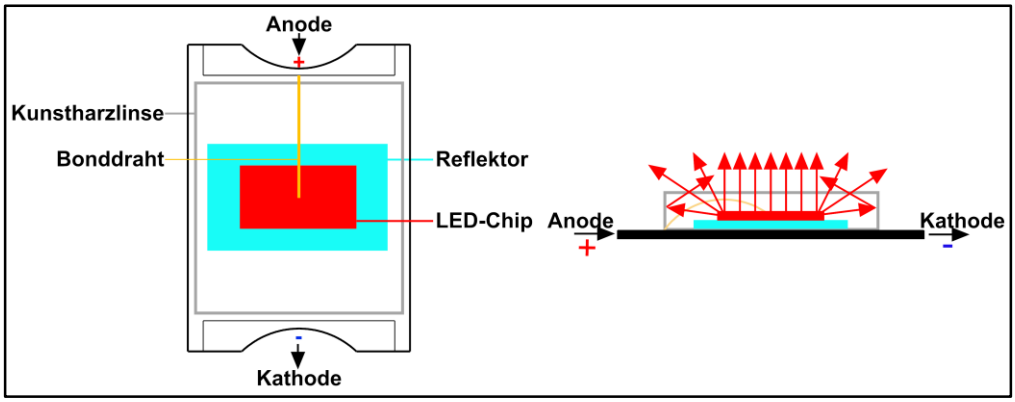


Abbildung 019 - SMD-LED

Ersichtlich ist, dass diese Bauform nicht besonders für eine seitliche Betrachtung geeignet ist.

Diese LEDs gibt es je nach Einsatzbereich in unterschiedlichen Ausführungen. Durch ihre geringe Bauform sind diese LEDs in unterschiedlichen Einsatzbereichen zu finden.

Die Abbildung 020 zeigen den Einsatz solcher LEDs in den Kfz-Bereichen.

a) Rückscheinwerfer VW-Passat Modell 2011	b) Rückscheinwerfer Audi-A8 Modell 2012

Abbildung 020 - Einsatz von LEDs im Kfz-Bereich Rückscheinwerfer

Die Abbildung 020 zeigt jeweils einen SMD-LED-Rückscheinwerfer zweier Automobilhersteller. Für eine fortführende Recherche dienen die Referenzen [10-14].

3 Bussysteme

Elektronische Systeme bestehen meist aus mehreren Komponenten. Eine Kommunikation zwischen diesen kann auf unterschiedlichen Wegen erfolgen. Damit eine Datenübertragung zwischen mehreren Hardwarekomponenten so wenige Leitungen wie möglich beansprucht, ist ein gemeinsamer Übertragungsweg (Bus) sinnvoll.

3.1 Seriel Peripheral Interface

Das Seriel Peripheral Interface (SPI) ist ein synchrones serielles Datenbussystem, welches von der Firma Motorola entwickelt wurde. Es ist ein lizenzfreies, nicht standardisiertes Protokoll. Neben der Bezeichnung SPI ist der Bus unter der Bezeichnung Microwire zu finden, welche das eingetragene Markenzeichen der Firma Nation Seminconductor ist [15].

3.1.1 SPI-Bus Portbezeichnungen

Die nachfolgende Tabelle 03 gibt einen Überblick der SPI-Portbezeichnungen. Diese besitzen in der Praxis unterschiedliche Bezeichnungen.

Portbezeichnung	Alternative Bezeichnung	Daten
SCLK	SCK	Takt (Serial Clock)
MOSI	SDO	Master Datenausgang/ Slave Dateneingang
MISO	SDI	Slave Datenausgang/ Master Dateneingang
SS	SS CS STE	LOW-aktiv Slave Select ($\overline{SS}$) Chip Select Slave Transmit Enable

Tabelle 03 - SPI-Portbezeichnungen

Zahlreiche Halbleiterhersteller, zum Beispiel Texas Instruments, Samsung und Fairchild Semiconductor bieten fertige SPI Peripherietypen an [16].

- Analoge- und Digitale Wandler (ADC, DAC)
- Speicher (EEPROM und FLASH)
- Real Time Clocks (RTC)
- Sensoren (Temperatur, Druck)
- Sonstige (Signalmixer, Potentiometer, Liquid Crystal Display (LCD) -Controller, Universal Asynchronous Receiver Transmitter (UART), Controller Area Network (CAN) -Controller, Universal Serial Bus (USB) -Controller, Amplifier)

3.1.2 SPI-Bus mit einem Master und Slave

Ein SPI-Bus arbeitet in einer Steuereinheit (Master) in Verbindung mit mindestens einer Peripherieeinheit (Slave). Das System besteht aus drei Steuerleitungen für eine synchronisierte Kommunikation zwischen Master und Slave. Die Datenübertragung kann in beide Richtungen parallel (Vollduplex) erfolgen. Zusätzlich wird eine Auswahlleitung für den Slave benötigt.

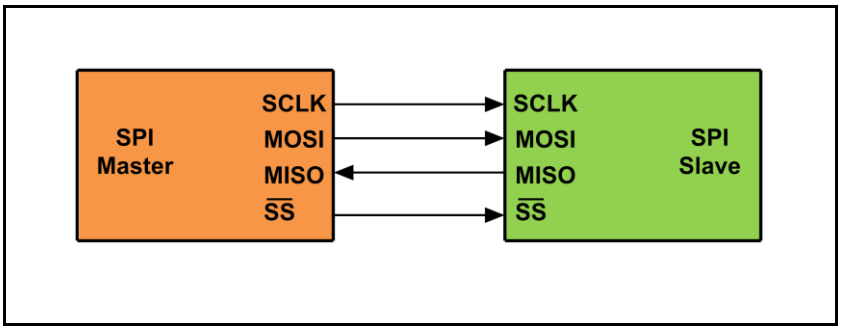


Abbildung 021 - SPI-Bus mit einem Master und Slave [15]

Der Master erzeugt und stellt ein Taktsignal für Slaves bereit, welches über eine SCLK-Leitung übertragen wird. Die zu übertragenden Daten werden auf zwei Signalleitungen übermittelt. Eine Datenübertragung vom Master erfolgt über eine "Master Output Slave Input" (MOSI) Leitung, während eine Datenübertragung vom Slave zum Master über eine "Master Input Slave Output" (MISO) Leitung gesendet wird. Eine Selektierung der Slave-Bausteine erfolgt über eine Slave-Select-Leitung (SS).

Eine SS-Leitung ist LOW-aktiv, wenn der Master einen Slave auf GND setzt. So lauscht dieser am MOSI und wartet auf Daten des Masters. Nach jenem Funktionsprinzip wird eine Datenübertragung zwischen Master und Slave und umgekehrt realisiert.

3.1.3 SPI-Bus Verbindung durch Kaskadierung der Slaves

Eine Verbindung und Anordnung der Slaves kann unterschiedlich erfolgen. Die anschließende Abbildung 022 zeigt den Aufbau einer Verkettung von Slave-Elementen.

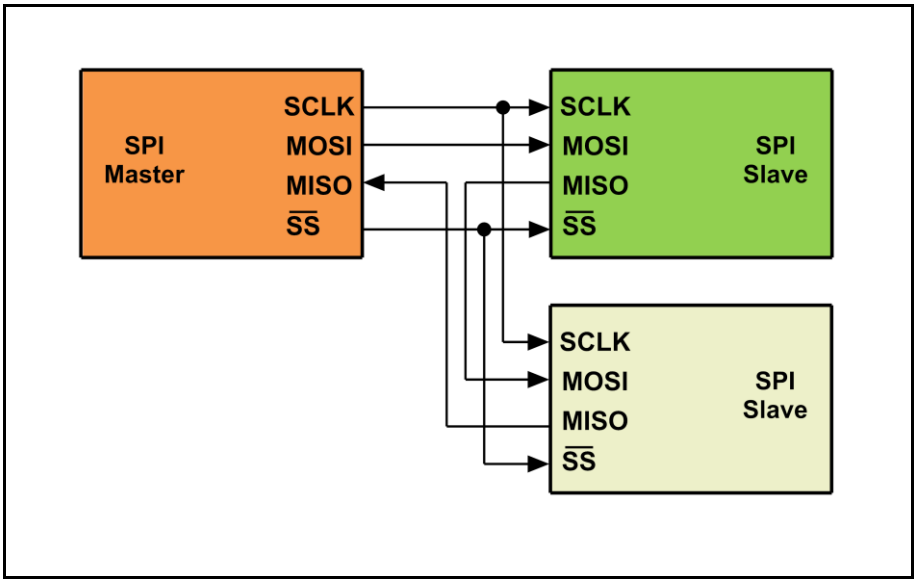


Abbildung 022 - SPI-Bus Verbindung durch Kaskadierung der Slaves [15]

Mehrere Slave-Bausteine können kaskadiert werden. Bei dieser Busstruktur werden die Daten des ersten Slaves an den Dateneingang des nachfolgenden Slave-Bausteines übertragen. Das System wird dadurch größer und bildet zum Beispiel ein breites Schieberegister[1].

[1] im Register stehender Bit kann um seine Position nach links oder rechts verschoben werden.

3.1.4 SPI-Bus Sternverbindung

Bei einer Verwendung von verschiedenen Slaves, zum Beispiel Temperatursensor und einem Display Controller, ist eine sternförmige Busstruktur möglich.

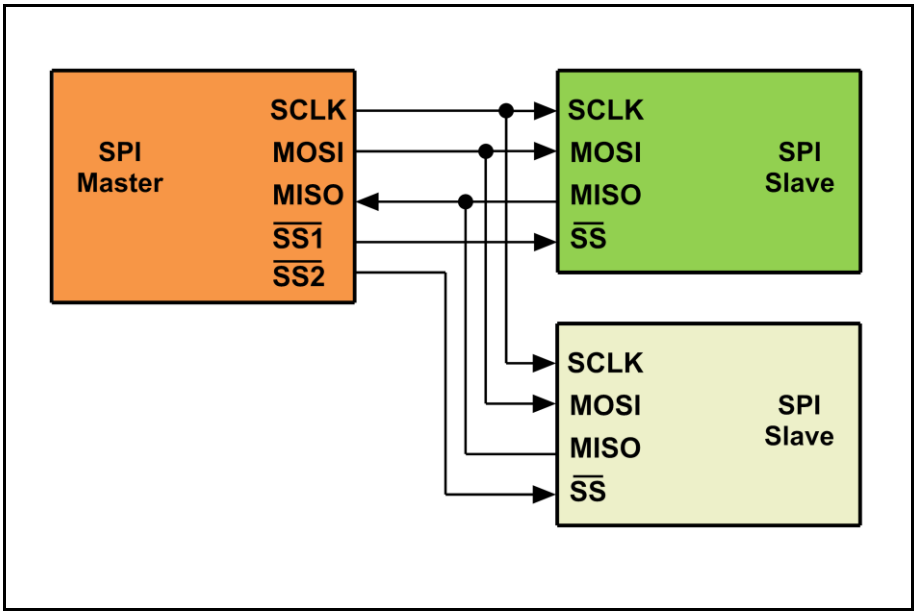


Abbildung 023 - SPI-Bus Sternverbindung [15]

Alle Slave-Bausteine, außer der SS, sind über gemeinsame Verbindungen direkt mit dem Master gekoppelt.

Für eine fortführende Recherche des SPI-Bussystems und den SPI-Protokollen bieten sich die Referenzen [15 - 18] an.

3.2 Inter-Integrated Circuit

Der Inter-Integrated Circuit (I²C) ist ein synchrones serielles Zweikanal-Bussystem, das von Next eXPerience Semiconductor (NXP) (Philips Semiconductor) spezifiziert und entwickelt wurde [19]. Aus dem Namen abgeleitet, handelt es sich um ein Bussystem, welches für eine Kommunikation von integrierten Schaltkreisen spezifiziert wurde.

Abbildung 024 - I²C-Logo [21]

Aus patentrechtlichen Gründen führte der Mikrocontroller[2] Hersteller ATMEL das technisch identische Two-Wire-Interface (TWI) ein [22]. Am 1. Oktober 2006 ließ NXP die Patente auslaufen. Ein Markenschutz besteht lediglich für das I²C-Logo. NXP führt die Weiterentwicklung fort und veröffentlichte im Jahr 2012 die Version 5 [23].

Zahlreiche Halbleiterhersteller zum Beispiel Texas Instruments, Samsung, Fairchild Semiconductor fertigen ebenfalls integrierte Schaltkreise [20] nach dem I²C-Standard an. Anwendungsbeispiele für einen I²C-Bus sind unter anderem:

- Analoge-Digitale-Wandler
- Digitale-Analoge-Wandler
- Display-LED-Treiber
- Eingabe-Ausgabe Erweiterungen
- Daten Konverter
- Speicher-Bausteine
- Uhr-Kalender Bausteine

[2] sind Halbleiterchips, welche je nach Einsatzbereich einen Prozessor, Arbeits- und Programmspeicher sowie Peripheriefunktionen enthalten.

3.2.1 I²C-Aufbau

Ein I²C-Bus besteht aus Spannungs-Leitung (V_{DD}), Masse (GND)- Leitung, Taktleitung (SCL) und einer Datenleitung (SDA).

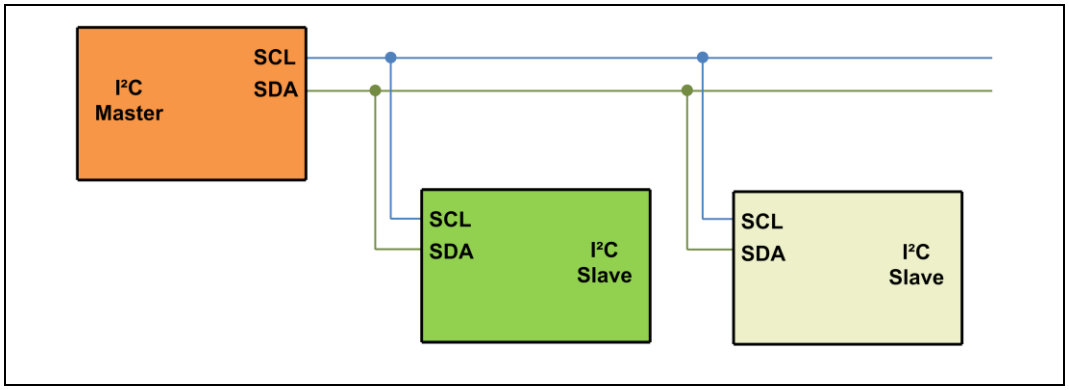


Abbildung 025 - I²C-Schema bearbeitet unter Verwendung von [23]

Das System besteht aus mindestens einem Master und einem oder mehreren Slaves. Alle Teilnehmer auf einem I²C-Bus werden mit der SCL- und SDA-Leitung verbunden.

Jeder Slave verfügt über eine eindeutige Adresse. Der Master selektiert anhand dieser einen Slave. Die Datenübertragung kann nur durch den Master eingeleitet werden. Die Slaves lauschen passiv auf dem Datenbus nach der Slave-Adresse. Erkennt ein Slave-Baustein seine Adresse im Datenkanal, greift er aktiv in das Bus-Geschehen ein.

Ein Betrieb mit mehreren Mastern (Multimaster Mode), wobei zwei Master-Geräte direkt miteinander kommunizieren, ist möglich und in der Spezifikation [23] definiert.

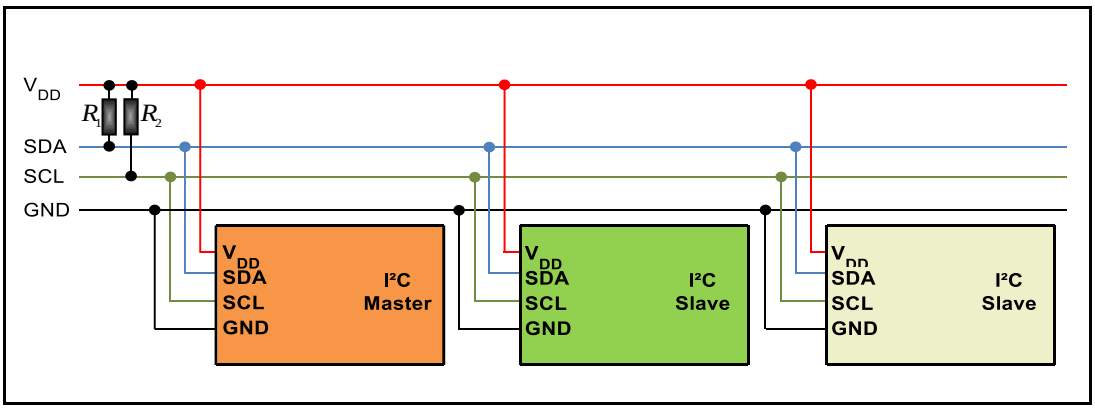


Abbildung 026 - I²C-Schema des Open-Collector-Bus bearbeitet unter Verwendung von [23]

Der I²C-Bus ist ein Open-Collector-Bus, das heißt die Spannungen richten sich nach der Spannung an den Pull-Up-Widerständen. Dies bedeutet, dass die Anschlüsse mit einem HIGH-Pegel betrieben werden. Erfolgt ein zu schneller Flankenwechsel des Masters, so kann der Slave diese auf LOW ziehen und der Master wartet bis der Slave den Pegel freigibt. Dieses Verfahren wird als sogenanntes Clock-Stretching bezeichnet. Die systematische Bitübertragung erfolgt im Big-Endian[3] und ist in der Spezifikation beschrieben. Die folgende Abbildung 027 verdeutlicht den Aufbau für eine sequentielle Lese- und Schreiboperation.

Abbildung 027 - I²C-Lese- und Schreibe-Konstruktion bearbeitet unter Verwendung von [24]

Zu erkennen ist, dass das I²C-Protokoll statisch aufgebaut ist, welches in den folgenden Teilkapiteln näher erläutert wird.

3.2.2 I²C-Idle-Zustand

Bei einer Inbetriebnahme oder nach einer fertigen Übertragung eines I²C-Busses befindet sich der Master in einem *idle* (freier Bus) - Zustand. Die Taktleitung und Datenleitung sind stets HIGH-aktiv.

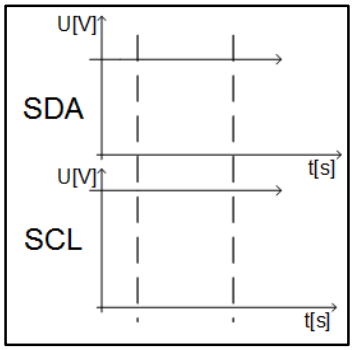


Abbildung 028 - I²C-Idle-Zustand bearbeitet unter Verwendung von [24]

[3] ist eine Schreibweise von Daten, beginnend mit dem größtwertigsten Bit (links nach rechts).

Befindet sich der Bus im Idle-Modus, kann mittels einer Startbedingung eine Datenübertragung initialisiert werden.

3.2.3 I²C-Start-Zustand

Dieser Zustand wird durch den Master mit einem Startsignal eingeleitet. Er zieht die SDA-Leitung von einem HIGH-Pegel (logisch „1“) auf einen LOW-Pegel (logisch „0“), während die Taktleitung auf einem HIGH-Pegel bleibt.

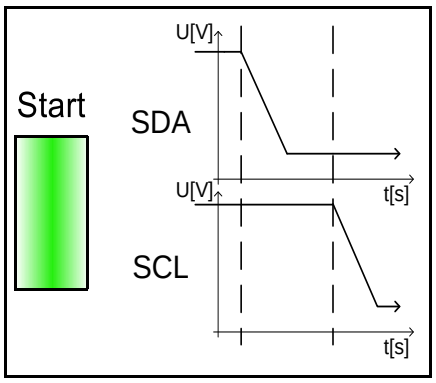


Abbildung 028 - I²C-Start-Zustand bearbeitet unter Verwendung von [24]

Die Abbildung 028 stellt eine I²C-Startbedingung für eine Übertragung dar.

3.2.4 I²C-Datenbit-Übertragungszustand

Anschließend erfolgt eine Übertragung von einem Byte. Dieses wird in zwei Kategorien eingeteilt.

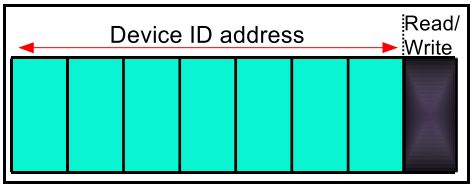


Abbildung 029 - Aufteilung eines I²C-Adressbyte bearbeitet unter Verwendung von [26]

Ein Datenbyte kann aus sieben Adressbits und einem Lese- oder Schreibbit gebildet werden (Abbildung 029).

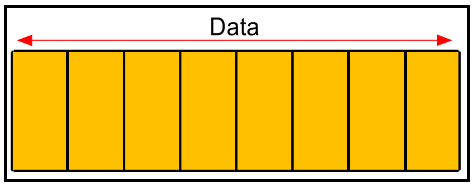


Abbildung 030 - Aufteilung eines I²C-Datenbyte bearbeitet unter Verwendung von [26]

Erfolgt eine Datenübertragung auf einen Slave, besteht dieses Byte aus den zu übertragenen Daten vom Master zum Slave (Abbildung 030). Ist eine Leseoperation gewünscht, so besteht dieses Byte aus 8-Lesebits vom Slave zum Master.

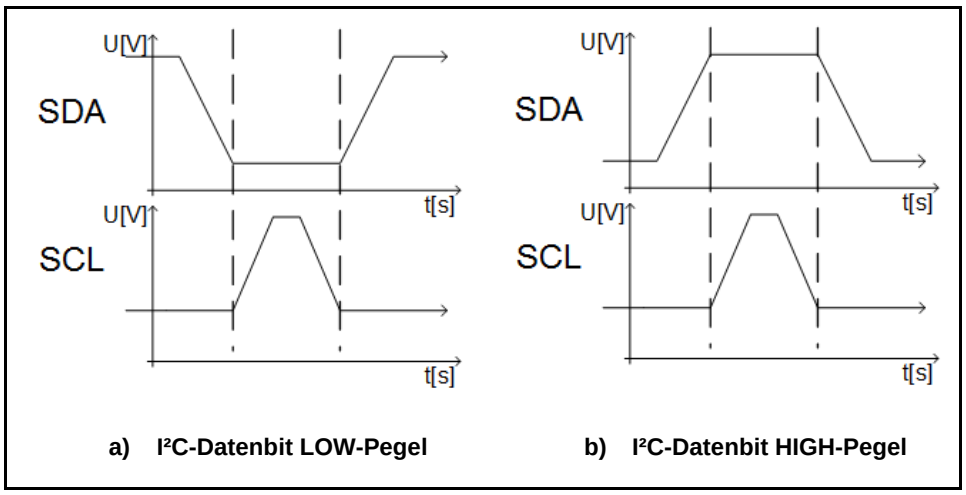


Abbildung 031 - Pegel eines I²C-Datenbits bearbeitet unter Verwendung von [24]

Anschließend erfolgt die Übertragung von einem Datenbyte. Ein Datenbit kann einen HIGH-Pegel und einen LOW-Pegel auf der SDA-Leitung annehmen, während die SCL-Flanke auf HIGH ist (Abbildung 031).

3.2.5 I²C-Acknowledge-Bit

Wurde dieses Byte erfolgreich vom Slave empfangen und geschrieben, quittiert er dies mit einem Acknowledge-Bit und signalisiert, dass die Übertragung erfolgreich war und weitere Daten empfangen werden können.

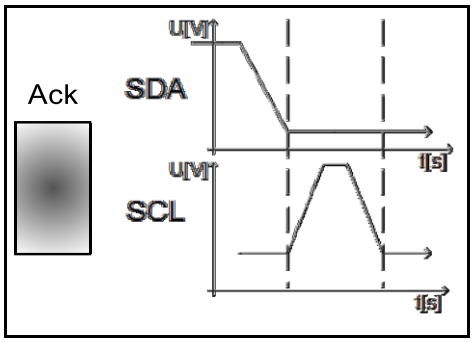


Abbildung 032 - I²C-Acknowledge-Bit bearbeitet unter Verwendung von [24]

Der Slave zieht die SDA-Leitung im neunten Takt auf einen LOW-Pegel.

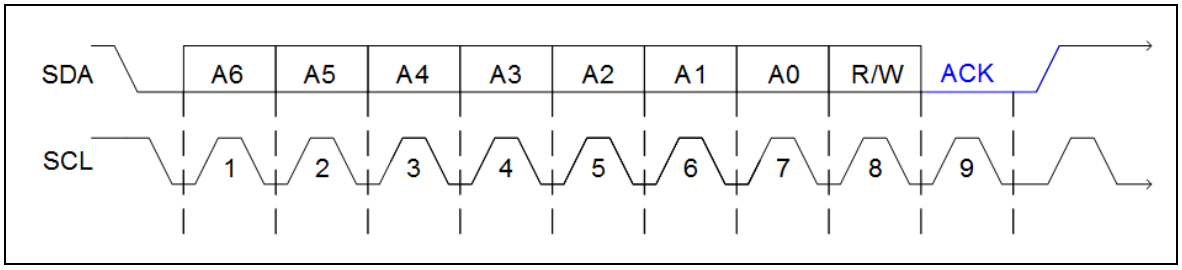


Abbildung 033 - I²C-Acknowledge-Bit Takt bearbeitet unter Verwendung von [26]

Erfolgt ein Lesezugriff vom Master zu einem Slave, erzeugt der Master ein Acknowledge-Bit. Der Slave erkennt diese Quittierung und sendet ein neues Byte. Ist die Datenübertragung abgeschlossen, wird der Bus freigegeben.

Anzumerken ist, dass das Steuersignal praktisch wie ein Datenbit gehandelt wird. Der muss dafür einen zusätzlichen neunten Taktimpuls auf SCL erzeugen.

3.2.6 I²C-Stop-Zustand

Das Ende einer Datenübertragung erfolgt durch eine Stop-Bedingung des Masters.

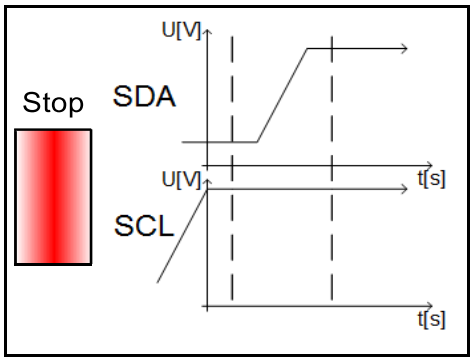


Abbildung 034 - I²C-Stoppbedingung bearbeitet unter Verwendung von [24]

Die SDA-Leitung wird von LOW auf HIGH gezogen, während die SCL-Leitung auf einem HIGH-Pegel bleibt. Durch diesen Prozess wird der Bus freigegeben und befindet sich im Idle-Modus.

3.2.7 I²C-Not-Acknowledge-Bit

Erfolgt eine Lesesequenz, sendet der I²C-Master nach dem Erhalt des letzten Datenbytes ein Not-Acknowledge-Signal. Dies bedeutet, dass er keine weiteren Daten mehr vom Slave erhalten möchte.

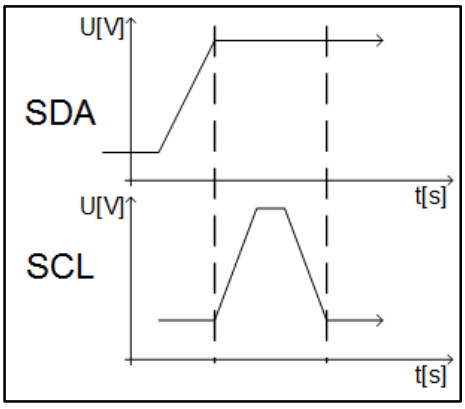


Abbildung 035 - I²C-Not Acknowledge bearbeitet unter Verwendung von [24]

Bei dieser Sequenz hält der Slave den SDA auf einen HIGH-Pegel (Abbildung 035).

Kommando	Funktion	Daten-Bit (SDA)	Takt (SCL)
Idle	Warte-Zustand	1	1
Start	Initialisierung I²C	0	1
Stop	Beende aktuelle Übertragung	1	1
Datenbit	Datenübertragung	0/1	1
Acknowledge	Übertragung erfolgreich	0	1
Not Acknowledge	Übertragung nicht erfolgreich	1	0
Read	logische ‘1‘	1	1
Write	logische ‘0‘	0	1

Tabelle 04 - I²C-Kommandos

Die Tabelle 04 gibt eine Übersicht der I²C-Zustände.

3.2.8 I²C-Adressraum

I²C verwendet einen Adressraum von sieben Bits. Dies entspricht einem Netzwerkknoten von maximal 128 (2^7) Slave-Bausteinen. Sechzehn Adressen sind für weitere Einsatzbereiche reserviert. Dadurch können theoretisch maximal 112 ICs in einem I²C-Netzwerkknoten existieren.

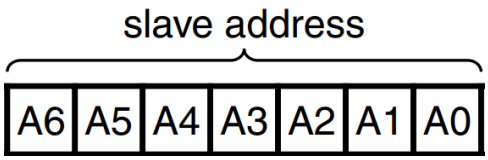


Abbildung 036 - I²C-Adressraum [26]

Die Adresse eines I²C-Bausteins wird in der Regel vom Hersteller festgelegt. Einige besitzen fixe Adressen, während andere die Möglichkeit bieten eine Teiladresse[4] (Subaddress), welche vom Benutzer vergeben wird, festzulegen. Bei vielen ICs sind dies die letzten drei Adressbits.

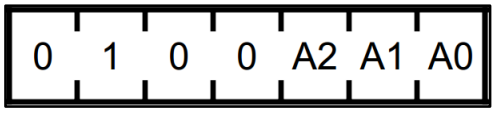


Abbildung 037 - I²C-Adressraum mit Teiladressen [26]

Somit können maximal 8 (2^3) gleiche ICs an einem I²C-Bus betrieben werden. Folgende Logiktabelle gibt einen Überblick des konfigurierbaren Subadressraumes.

A2	A1	A0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Tabelle 05 - I²C-Subadressraum

Damit der Bus ausbaufähig ist, wurden zwei Adressbereiche von jeweils einem Byte reserviert. Die nachfolgende Tabelle 06 gibt eine Übersicht der Adressen:

[4] benutzerdefinierte programmierbare Adressbits einer Adresse

Adresse	R/W Bit	Beschreibung
0000000	0	General Call Adresse
0000000	1	Startbyte
0000001	X	CBUS Adresse
0000010	X	reserviert für ein anderes Busformat
0000011	X	für zukünftige Erweiterungen reserviert
00001XX	X	für zukünftige Erweiterungen reserviert
11111XX	X	für zukünftige Erweiterungen reserviert
11110XX	X	10-Bit Adressierung

Tabelle 06 - I²C-Reservierte Adressen [26]

Aufgrund der Busentwicklung und dessen Einsatzmöglichkeiten wurden 10-Bit-Adressen eingeführt. Diese erlauben bis zu 1024 (2^{10}) Geräte an einem Bus. Durch die Reservierung der Adressen ‘1111 0XX‘ und des R/W Bits, werden mögliche 7-Bit-Geräte am Bus nicht gestört. An einem I²C-Bus können 7-Bit- und 10-Bit-Adress-Geräte zusammen betrieben werden. Die folgende Abbildung 038 zeigt einen systematischen Aufbau einer zu übertragenden 10-Bit-Adresse.

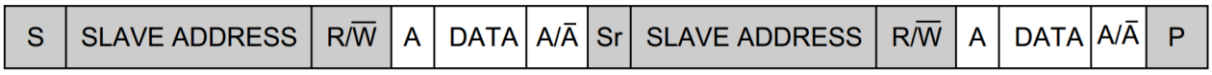


Abbildung 038 - I²C-Adressraum mit einer 10-Bit Adressierung [26]

Eingeleitet wird eine 10-Bit-Adresse durch die ersten fünf Bits (Tabelle 06) mit der Bitfolge ‘11110‘. Anschließend folgen die ersten zwei Bits der IC-Adresse, welche mit einem LOW-PEGEL für das Read/Write weitergeführt wird. Die Teilnehmer, die auf den ersten Teil der Adresse reagieren, übertragen ein Bestätigungsbit. Fortführend wird der zweite Teil der Adresse auf den Bus übertragen. Alle ICs, welche bereits auf den ersten Adressanteil reagierten, lauschen weiterhin auf diese folgenden Bits. Der angesprochene und korrekt adressierte Teilnehmer wird nach Erhalt der Bitfolge ein weiteres Bestätigungsbit senden. Anschließend beginnt der Datenaustausch. Für eine fortführende Recherche des SPI-Bussystems und den SPI-Protokollen bieten sich die Referenzen [19 - 26] an.

4 Vorstellung ausgewählter FPGA-Entwicklungsboards

In diesem Kapitel werden FPGAs und Entwicklungsboards für die Realisierung des Projekts vorgestellt.

4.1 Field Programmable Gate Array

FPGAs sind Halbleiter, die aus einer Matrix von konfigurierbaren Logikblöcken (CLBs) bestehen. FPGAs können für die jeweils gewünschte Anwendung oder Anforderung beschrieben werden. Dieses Merkmal unterscheidet FPGAs von Application Specific Integrated Circuits (ASICs), die speziell für bestimmte Designaufgaben hergestellt werden. Es existieren neben kleineren Herstellern (Atmel, Actel usw.) zwei bekannte FPGA-Hersteller, Xilinx und Altera [27].

4.2 Very High Speed Integrated Circuit Hardware Description Language

Die Very High Speed Integrated Circuit Hardware Description Language (VHDL) ist eine Hardwarebeschreibungssprache, welche seit 1987 im Institute of Electrical and Electronics Engineers (IEEE) -Standard verankert ist. Mit ihr ist eine textbasierte Beschreibung digitaler Systeme möglich [28,29]. In großen Abständen veröffentlicht die IEEE eine neue Revision des VHDL-Standards.

4.3 VHDL-Synthesetool

Ein Synthesetool ist eine Software, welche die Hardwarebeschreibung in ein FPGA-Layout überträgt. FPGA-Entwicklungsumgebungen[5] beinhalten zu meist dieses Tool und erlauben die Anwendung unterschiedlicher VHDL-Standards [30]. Aus Kompatibilitätsgründen wird in dieser Arbeit der VHDL-Standard 2008 vorwiegend angewandt.

4.4 Vorstellung Quartus II und Cyclone Entwicklungs-Boards

Die Altera Cooperation ist ein Hersteller von anwendungsspezifischen und programmierbaren integrierten Schaltungen und wurde 1983 gegründet. Die bekanntesten Produkte stellen FPGAs aus der Reihe Stratix, Arria und Cyclone dar. Desweiteren stellt diese hauseigene Software, zum Beispiel Electronic Design Automation (EDA), meist kostenlos zur Verfügung. Für eine Entwicklung von zum Beispiel prototypischen Projekten bietet diese Firma fertige Entwicklungsboards an [27 - 33]. Die Software Quartus II sowie zwei Entwicklungsboards werden in den folgenden Kapiteln kurz vorgestellt. Für eine fortführende Recherche bieten sich die Referenzen [27 - 50] an.

4.4.1 Quartus II

Altera Quartus II ist eine EDA-Anwendung. Sie dient unter anderem zur Analyse und Synthese von Hardwarebeschreibung (HDL) [34].

Abbildung 039 - Altera Quartus II Logo [32]

[5] ist ein Anwendungsprogramm in Form einer Programmierungsumgebung. Dieses besteht zu meist aus einer grafischen Benutzeroberfläche, welche die Funktionsvielfalt eines Editors mit Syntax-Hervorhebung, Interpreter, Simulator und Debugger vereint.

Eine herausragende Funktion des Programmes ist dabei die Möglichkeit der Simulation der beschriebenen Hardware sowie eingebundene Systeme zu hosten.

Grafisches Design mit HIGH-Level-IP-Blöcken soll dabei die Entwicklung erleichtern. Konkret bietet die Software Module für folgende Aufgaben [6]:

- FPGA-Beschreibung
- Embedded Software
- Schaltungssimulation

Diese Software kann als kostenlose Web-Edition oder als Vollversion 14.0 bezogen werden. Aus Kompatibilitätsgründen wird in diesem Projekt die kostenlose Web-Edition in der Version 13.1 angewandt [32].

Anzumerken ist, dass in der neuen Version nur noch aktuelle Cyclone-Modelle unterstützt werden. Aus diesem Grund wird für Cyclone 2 und 3-Modelle die Version 13.1 empfohlen.

Desweiteren bietet die Firma Altera eine eigene Simulationsumgebung für HDL mit dem Namen ModelSim an.

Abbildung 040 - Altera ModelSim Logo [33]

Die Software ermöglicht eine taktsynchrone oder zeitgenaue Simulation von digitalen Logikelementen. Die aktuelle Version 10.1 kann unter [33] kostenlos bezogen werden. Für eine fortführende Recherche von FPGAs, Altera sowie der Vergleich beider Quartus II Versionen ist in [31 - 37] zu finden.

4.4.2 Developer-Kit Cyclone III

Das *DE0 Development and Education board* (DE0 Board) ist ein preisgünstiges Entwicklungs- und Einsteigerboard für Design, *prototyping* und Implementierung von Elektronikprodukten, welches mit einem Cyclone III bestückt ist.

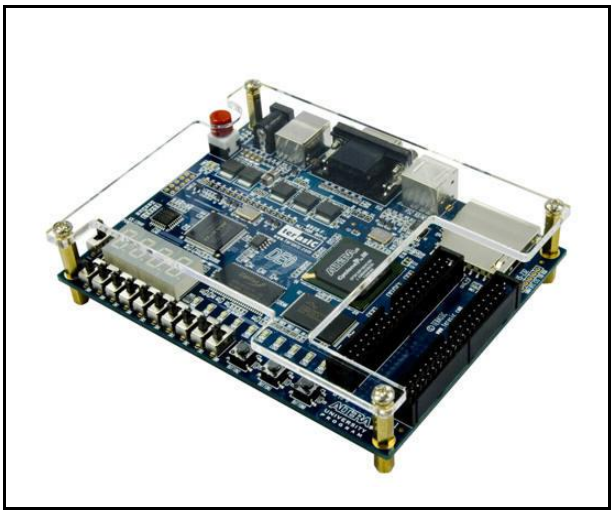


Abbildung 041 - Altera DE0 Board [38]

Das Board bietet unter anderem vier Siebensegmentanzeigen, drei Taster, zehn Schiebetaster, zehn grüne LEDs, zwei 40-PIN-GPIO-Anschlüsse und einen Cyclone III FPGA. Der Cyclone III (3C16) besitzt 15.408 konfigurierbare Logikelemente. Eine detaillierte Produktinformation bietet das Datenblatt [39].

Im Rahmen dieses Projektes wird nur der GPIO-Header betrachtet. Für eine Ansteuerung externer logischer Baukomponenten bieten sich die GPIO_0- und GPIO_1-Anschlüsse an. Die folgende Abbildung 042 zeigt eine dieser Schnittstellen.

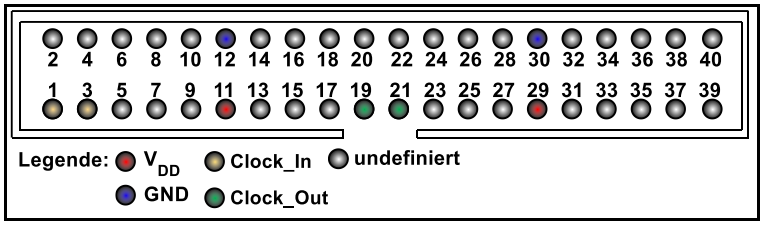


Abbildung 042 - Altera DE0 Board 40-PIN-GPIO_1 [39]

Aus dem Datenblatt des DE0 Boards ist ersichtlich, dass acht dieser Ports von Altera eine vordefinierte Ein- und Ausgabefunktion besitzen, welche in folgender Tabelle 07 zusammengefasst werden.

PIN-Nummer	Altera PIN	Altera Belegung
1	PIN_AB11	Clock_In
3	PIN_AA11	Clock_In
11		V_{DD}
12		GND
19	PIN_R16	Clock_Out
21	PIN_T16	Clock_Out
29		V_{DD}
30		GND

Tabelle 07 - vordefinierte Altera DE0 Board PIN-Belegung des GPIO_1 [39]

Die Anschlüsse 11 und 29 dienen für eine Spannungsversorgung (V_{DD}), während PIN 12 und 30 für GND reserviert sind. Die PINs 19 und 21 dienen zum Durchleiten des FPGA-Taktes. Über die Anschlüsse eins und drei kann ein externer Taktgeber angeschlossen werden. Alle weiteren Anschlüsse sind frei konfigurierbar und können für logische Ein- und Ausgabeoperationen angewandt werden. Jeder GPIO-Port liefert eine maximale Stromstärke von $16\mathrm{mA}$. Wird dieser Wert überschritten, kann dies zu einem Defekt des FPGAs führen.

4.4.3 Developer-Kit Cyclone IV

Eine nachfolgende Generation der Cyclone III FPGAs stellt der Cyclone IV dar. Dieser verfügt über mehr Logikelemente bei kleinerer Bauweise. Die folgende Abbildung 043 zeigt das derzeitig kleinste Entwicklungsboard mit der Bezeichnung Cyclone IV DE0 NanoBoard, welches direkt über USB (Stand November 2014) betrieben wird.

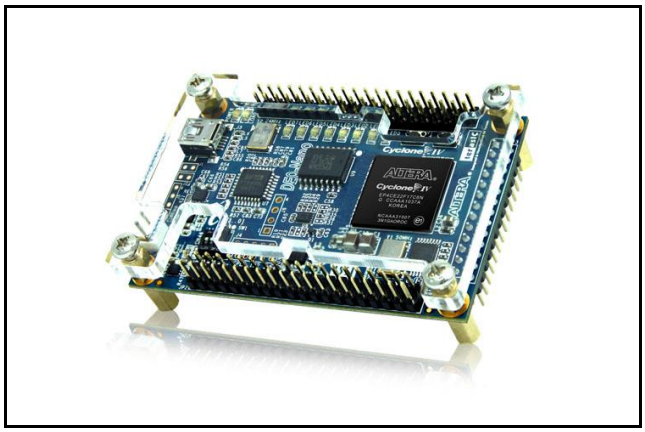


Abbildung 043 - Altera DE0 NanoBoard [40]

Dieses Board bietet unter anderem zwei Taster, vier Schiebetaster, acht grüne LEDs, zwei 40-PIN-GPIO-Anschlüsse, einen 26-PIN-GPIO-Anschluss und einem Cyclone IV FPGA. Der Cyclone IV EP4CE22F17C6N besitzt 22.320 konfigurierbare Logikelemente. Eine detaillierte Produktinformation bietet das Datenblatt [41].

Im Rahmen dieses Projektes werden die GPIO-Header genauer betrachtet. Aus dem Datenblatt ist zu entnehmen, dass ebenfalls PINs für vordefinierte Ein- und Ausgabefunktionen existieren.

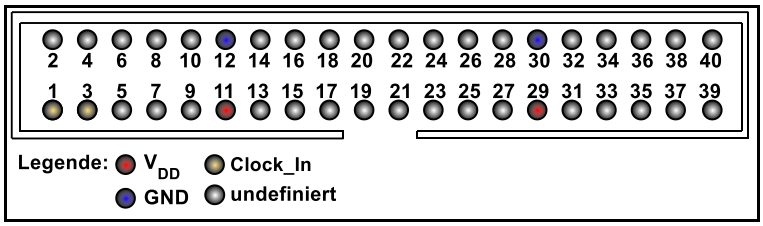


Abbildung 044 - Altera DE0 NanoBoard 40-PIN-GPIO_1 [41]

Die Abbildung 044 verdeutlicht, dass die Ports 11,12,29 und 30 von Altera ebenfalls für V_{DD} und GND fest belegt sind. Unterschiede gegenüber dem Cyclone III Board, existieren bei der PIN-Bezeichnung, welche in der Tabelle 08 zusammengefasst sind.

PIN-Nummer	FPGA-PIN	Altera Belegung
1	PIN_T9	Clock_In
3	PIN_R9	Clock_In
11		V_{DD}
12		GND
29		V_{DD}
30		GND

Tabelle 08 - vordefinierte Altera DE0 Nano Board PIN-Belegung des GPIO_1 [41]

Anhand dieser Tabelle 08 wird deutlich, dass sechs Anschlüsse für zwei externe Taktgeber, Spannung und Masse genutzt werden können.

Alle weiteren PINs sind frei konfigurierbar und können für logische Ein- und Ausgabeoperationen angewandt werden.

4.5 Vorstellung Altium-Designer und NanoBoard 3000

Die Altium Limited ist ein Hersteller, unter anderem spezialisiert auf elektronische Design-Software, mit ihrem Hauptsitz in Australien. Eines der bekanntesten Produkte ist der Altium-Designer. Für eine Entwicklung, von zum Beispiel prototypischen Projekten und Anwendungen, bietet diese Firma fertige Entwicklungsboards und umfangreiche Hilfestellungen auf ihrer Internetseite an [42 - 44]. Nachfolgend werden das NanoBoard 3000 und der Altium-Designer kurz erläutert.

4.5.1 Altium-Designer

Der Altium-Designer ist eine EDA-Anwendung der Firma Altium Limited. Er dient unter anderem zur Entwicklung von Leiterplatten in der Elektronik und einer Hardware-beschreibung [44].

Abbildung 045 - Logo des Altium-Designer 2014 [45]

Eine besondere Funktion des Programmes ist eine 3D-Ansicht der Leiterplatte während der Entwicklung. Diese Software bietet eine umfangreiche Möglichkeit zur Realisierung von FPGA-basiertem Design. Dadurch wird es möglich, ein Embedded System, wie z.B. einen I²C-Master-Core auf einem FPGA zu hosten. Ein grafisches Design mit High-Level-IP-Blöcken hilft die Entwicklung zu erleichtern. Konkret bietet die Software Module für folgende Aufgaben [43]:

- Schaltplan-Erfassung
- Leiterplatten-Layout
- FPGA-Programmierung
- Embedded Software
- Schaltungssimulation
- CAM (Computer Aided Manufacturing)

Die Verwendung der Software ist kostenpflichtig und wird als Testversion von Altium zum Download bereitgestellt [42]. Bei diesem Projekt wird der Altium-Designer in der Version 14.3 angewandt.

4.5.2 Altium NanoBoard 3000

Das NanoBoard 3000 (NB3K) ist ein FPGA-Entwicklungsboard für Design, *prototyping* und Implementierung von Elektronikprodukten, welches mit einem Xilinx-Spartan FPGA bestückt ist (Abbildung 046). Bei diesem Produkt handelt es sich um eine wiederprogrammierbare Hardware-Entwicklungsplattform, die die Leistung eines dedizierten Hochleistungsgeräts nutzt, um eine schnelle und interaktive Implementierung sowie das Debugging eigener Entwürfe zu ermöglichen [42,43].

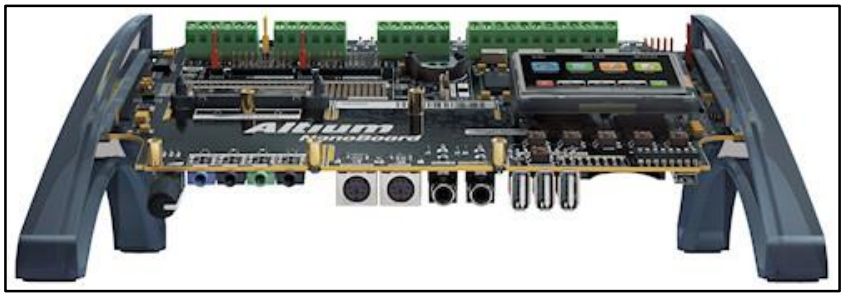


Abbildung 046 - Altium NanoBoard 3000 [46]

Damit eine Entwicklung auf diesem Board erfolgt, ist die Software Altium-Designer mit der EDA-Software Xilinx ISE WebPack[6] notwendig [48,49]. Besondere Vorkenntnisse der FPGA-Designerfahrungen werden mit Hilfe des Altium-Designers nicht benötigt.

Im Rahmen dieser Thesis liegt der Schwerpunkt ausschließlich auf der FPGA-Programmierung mit VHDL und der vom Board zur Verfügung stehenden GPIO-Header. Diese Schwerpunkte setzen allgemeine Kenntnisse der Hardwareentwicklung und den Umgang mit VHDL voraus. Für eine fortführende Recherche bietet sich die Referenz [42 - 51] an.

Für die Ansteuerung externer logischer Baukomponenten bietet das Board die GPIO-Header A und B an, welche wahlweise mit einer Spannung von $3{,}3\mathrm{V}$ bis $5\mathrm{V}$ betrieben werden können. Die folgende Abbildung 047 zeigt diese Schnittstellen.

[6] EDA-Software der Firma Xilinx

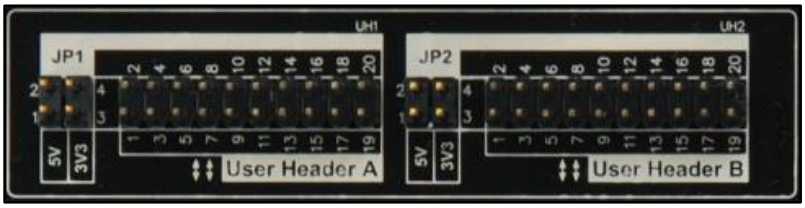


Abbildung 047 - Altium NanoBoard 3000 GPIO-Header A und B

Aus dem Datenblatt [47] ist ersichtlich, dass jeder GPIO-Port eine maximale Stromstärke von $16\mathrm{mA}$ liefert.

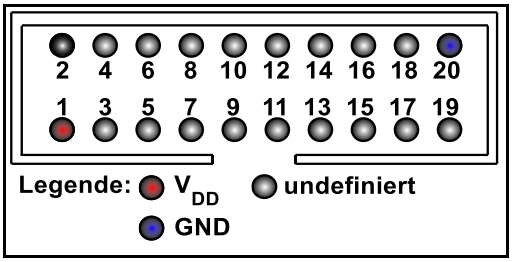


Abbildung 048 - Altium NanoBoard 3000 20-PIN-GPIO

Auf beiden Headern der Ports A und B, sind die Anschlüsse eins (V_{DD}) und zwanzig (GND) vordefiniert. Restliche PINs stehen dem Entwickler frei zur Verfügung und können für logische Ein- und Ausgabeoperationen angewandt werden. Die zwei von Altium belegten Anschlüsse sind in folgender Tabelle 09 zusammengefasst.

PIN-Nummer	Altium-PIN	Altium Belegung
1	HA1	V_{DD}
20	HA20	GND

Tabelle 09 - Altium NanoBoard 3000 PIN-Belegung des GPIO-Header A

Anzumerken ist, dass eine Ansteuerung der Ports über die Portbezeichnung HA für den GPIO-Header A und HB für den Header B erfolgt.

4.6 Vor- und Nachteile von FPGAs

Ein herausragendes Merkmal von FPGAs und der Hardwarebeschreibungssprache VHDL liegt im verankerten Standard. Chip- und herstellerspezifische Synthese sind von der anwendungsorientierten Hardwarebeschreibung entkoppelt. Entscheidet sich der Entwickler für einen VHDL-Standard, zum Beispiel Version 2008, so sollte seine Hardwarebeschreibung durch unterschiedliche Synthesetools interpretierbar sein.

Grundsätzlich ist ein Vergleich zwischen den zwei bekanntesten Herstellern Altera und Xilinx sehr umfangreich. Ein Zitat aus einem Altera Workshop vom 16.06.2013 beschreibt die Unterschiede wie folgt: „Die Firma Altera versucht vieles grundsätzlich anders zu realisieren, als die Firma mit X“[7].

Aus diesem Grund wird in dieser Dokumentation nur an den angewandten Stellen auf Unterschiede hingewiesen.

[7] Sprecher: Michael Fuhrmann Field Application Engineer (ALTERA), Altera Workshop, 16.06.2013, Atlanta Hotel Leipzig

5 RGB-LED-Würfel

Das Konzept des RGB-LED-Würfels wird in drei Teilbereiche gegliedert.

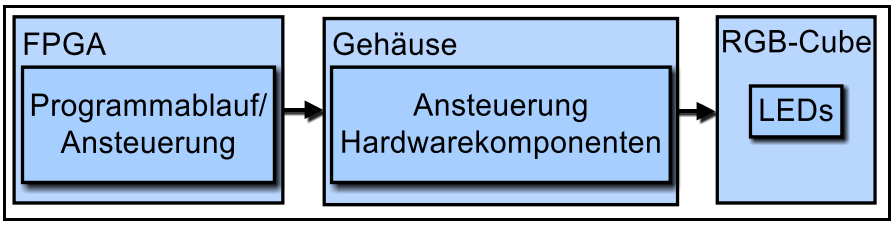


Abbildung 049 - Konzept RGB-LED-Cube

Der erste Abschnitt beinhaltet einen FPGA, welcher die Logik des Programmablaufs enthält. Diese wird im zweiten Abschnitt auf Hardwarekomponenten übertragen, welche sich in einem Gehäuse befinden. Im letzten Bereich erfolgt die Ausgabe der Daten in Form einer Lichtanimation auf der dreidimensionalen RGB-LED-Matrix. Die Konstruktion des Würfels, deren Hardwareumsetzung und dessen Ansteuerung mittels FPGA erfolgt in den nachfolgenden Kapiteln.

5.1 LED Auswahl

Im Kapitel 2.4 wurden bereits verschiedene LEDs mit Vor- und Nachteilen vorgestellt. Die Wahl richtet sich bei diesem Projekt nach einer Probandenmeinung, der Lieferbarkeit und nach dem Preis-Leistungs-Verhältnis.

Abbildung 050 - RGB-LED 5mm diffus [51]

Die Wahl ist eine $5mm$ Standard RGB-LED mit einem diffusen Gehäuse, einem Pluspol (Anode) und drei RGB-Kathoden mit folgenden Herstellerangaben:

Technische Daten	
Bauform	5mm Standard
PIN-Abstand	0,05" / 1,27mm
Kontakt	3x Kathode 1x Anode
Betriebstemperatur	-25 bis +85 °C

Tabelle 10 - Technische Daten RGB-LED [51]

Elektrische Eigenschaften	
Flussspannung	2,2, 3,3 3,4
max. Strom	3 x 30mA
Spitzenstrom	100mA für t<0,1ms

Tabelle 11 - Elektrische Eigenschaften RGB-LED [51]

Optische Eigenschaften	
Farbe	RGB steuerbar
Wellenlänge	630, 525,470nm
Helligkeit	6000mcd
Linse	diffus
Abstrahlwinkel	zirka 30°

Tabelle 12 - Optische Eigenschaften RGB-LED [51]

Durch das diffuse Gehäuse wird das Licht der LED stark seitlich verteilt. Dadurch eignet sich diese, neben der konkaven LED, für eine seitliche Betrachtung.

5.2 Konstruktion RGB-LED-Würfel

Dieses Kapitel beschreibt den Aufbau eines RGB-LED-Würfels. Eine vollständige Materialliste angewandter Hardwarekomponenten befindet sich im Anhang auf DVD unter dem Namen „Materialliste.pdf“.

5.2.1 Aufteilung des RGB-LED-Würfels in ein Koordinatensystem

Der Würfel wird zunächst als ein zweidimensionales Objekt (Ebenen und Säulen) betrachtet.

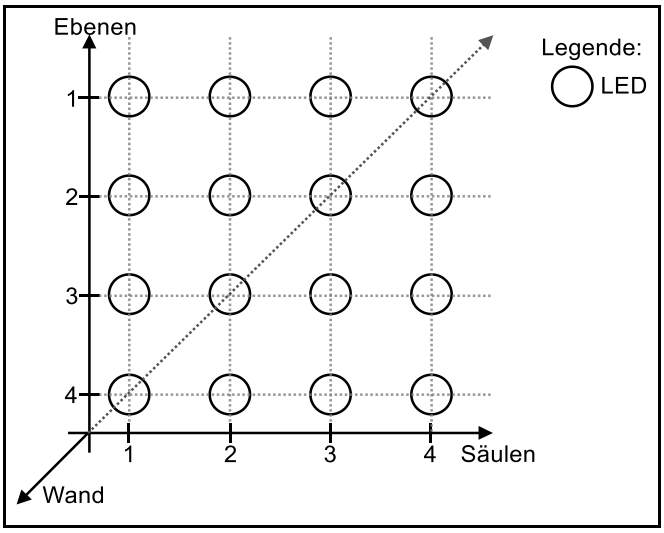


Abbildung 051 - Aufteilung der RGB-LED-Cube in Koordinaten

Die Anordnung der LEDs auf der Abszissenachse erscheint als einzelne Zeilen und bildet bei einem 3D-Objekt einzelne Ebenen. Die Ordinatenachse repräsentiert einzelne Spalten, welche als einzelne Säulen betrachtet werden. Die Z-Achse dient der räumlichen Betrachtung. Es entsteht eine 3D-Matrix, bei welcher jeweils die Ebenen durch Anoden und die Spalten durch Kathoden verbunden sind. Acht Dioden werden jeweils mit ihrer Anode in horizontaler Richtung miteinander verbunden.

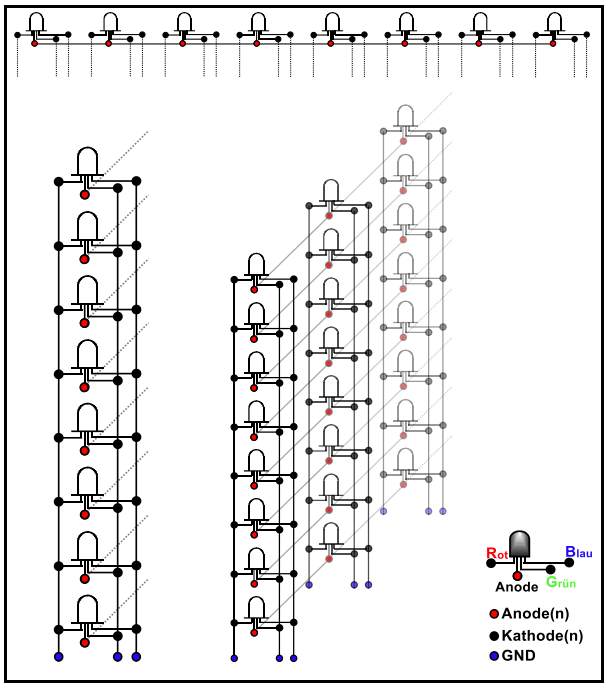


Abbildung 052 - Konzept Verdrahtung RGB-LED-Cube

Bei diesem Prozess entsteht eine Reihe aus acht LEDs. Für die Erzeugung eines Gitternetzes werden acht Reihen benötigt, welche anschließend jeweils mit ihren RGB-Kathoden in vertikaler Richtung verbunden werden. Bei der Umsetzung entsteht eine Wand bestehend aus jeweils acht Säulen und Reihen. Für die Errichtung des Cubes werden acht Wände benötigt.

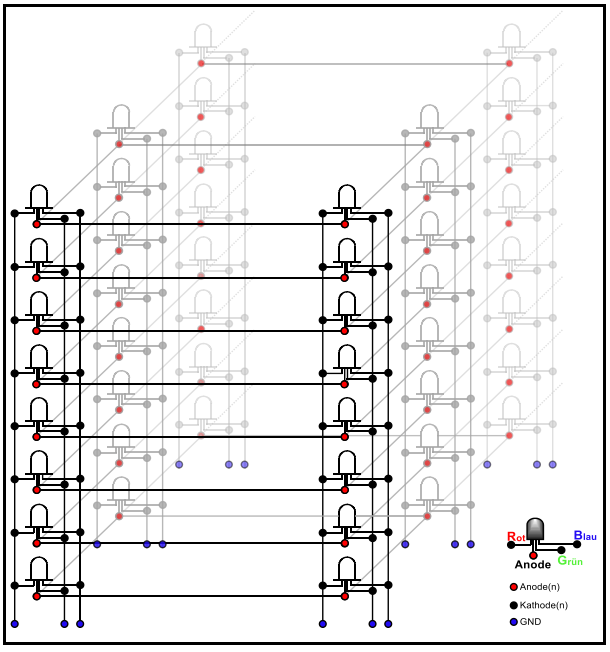


Abbildung 053 - Konzept Kathoden-Verdrahtung RGB-LED-Cube

Anschließend werden die einzelnen Wände über ihre Anoden miteinander verbunden.

5.2.2 Dimensionierung des Würfels

Die Darstellung eines Bildes auf einer 2D-Matrix entspricht einem Raster. Jede LED entspricht einem Pixel auf diesem. Wird die Matrix als ein dreidimensionales Objekt betrachtet, so entspricht jede einem Gitterpunkt in einem dreidimensionalen Raum und wird als Voxel bezeichnet. Bezugnehmend auf die Abstrahlcharakteristiken einer diffusen LED ist die Wahl eines geeigneten Rasters (Gittermaß und Diodenabstand) notwendig.

Abbildung 054 - Länge der Anode und Kathoden der angewandten RGB-LED

Eine RGB-LED besitzt drei Anschlussleitungen mit einer unterschiedlichen Länge von $(25 \pm 5)mm$. Eine Stichprobenanalyse ergab, dass jede zehnte LED eine Abweichung von $(25 \pm 10)mm$ aufwies. Um einen idealen Abstand benachbarter Dioden zu ermitteln, wurden drei Versuchsobjekte mit einem Leuchtiodenabstand von $25\mathrm{mm}$, $30\mathrm{mm}$ und $35\mathrm{mm}$ erstellt. Eine kleine Gruppe Probanden beurteilte die optische Eigenschaft und Wirkungsweise bei einem Abstand von zwei Metern zum Objekt. Dabei wurde eine Laufschrift und ein Standbild über die dreidimensionale Matrix gelegt. Die Testpersonen empfanden einen Abstand von $35\mathrm{mm}$ (Kopf zu Kopf) als geeignet.

Wird ein geringeres Gittermaß gewählt, so wirkt dies auf das menschliche Auge angenehm, da eine höhere Auflösung erreicht wird. Befinden sich die LEDs zu nah aneinander, trifft der Lichtkegel eine benachbarte Leuchtdiode. Bei einer Animation kann dies zu Irritationen führen.

Eine niedrigere Auflösung wird durch einen größeren Gitterabstand erreicht. Das Bild wirkt sehr grobkörnig, was durch einen größeren Abstand zum Objekt kompensiert werden kann. Bei einem $8x8x8$ Gitter ergibt sich eine Gesamtlänge von $245mm$. Damit dieser Abstand gewährleistet wird, werden die bestehenden Leitungen von jeder einzelnen LED gekürzt. Die folgenden Abbildungen zeigen diesen Vorgang.

Abbildung 055 - Biegen einer RGB-LED für den Cube

Bei einem Abstand von $4mm$ vom LED-Kopf zur Drahtmarkierung werden die Kathoden um $90°$ gebogen. Zu beachten ist, dass die äußere Kathode (Rote-Kathode) an der Markierung des Gehäuses um $90°$ gebogen wird.

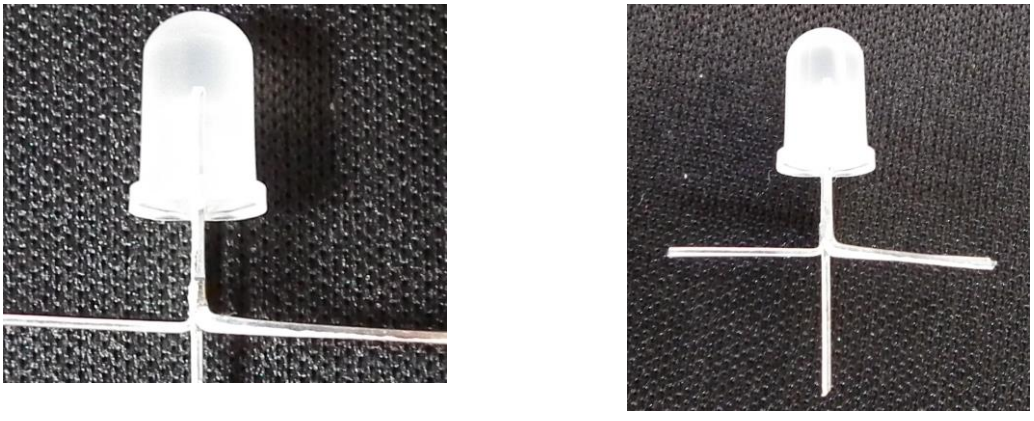


Abbildung 056 - Biegen der RGB-LED-Kathoden

Die Kathoden der Farbe grün und blau werden ebenfalls um $90°$, jedoch entgegengesetzt der roten Kathode, gebogen.

Anschließend werden alle vier Leitungen um zirka die Hälfte gekürzt. Um eine gute Leitfähigkeit und Stabilität des zu realisierenden Objekts zu gewährleisten, ist eine geeignete Wahl des Verbindungsdrahtes notwendig. Eine kleine Zusammenfassung der Leitfähigkeit von bekannten Metallen ist in folgender Tabelle 13 zusammengefasst.

Material	**Leitfähigkeit κ [S·m/mm²] = [m/Ω·mm²]**
Silber (Ag)	62,000
Kupfer (Cu)	56,000
Gold (Au)	43,500
Kohlenstoff (C)	00,100
Silizium (Si)	00,001

Tabelle 13 - Drahtleitfähigkeit [52]

Kupfer liefert neben Gold und Silber eine sehr gute Leitfähigkeit. Damit sich bei der Verarbeitung des Drahtes keine Korrosionserscheinungen bilden, wird verzinnter Kupferdraht genutzt.

Abbildung 057 - Verzinnter Kupferdraht bearbeitet unter Verwendung von [53]

Der verzinnte Kupferdraht hat eine silberne Farbe und bietet neben seiner hohen Leitfähigkeit ebenfalls gute Verarbeitungseigenschaften. Um eine Stabilität für das entstehende Gewicht des Würfels zu gewährleisten, wird ein $0{,}8\text{mm}$ Querschnitt gewählt.

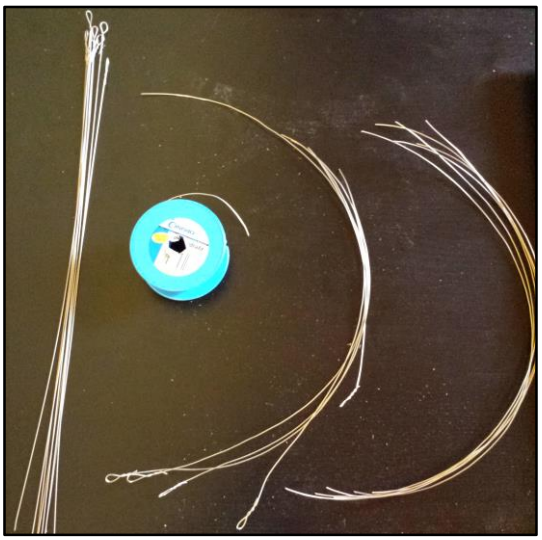


Abbildung 058 - Biegen des verzinnten Kupferdrahts

Für eine weitere Verarbeitung muss der Draht begradigt werden, indem er durch zwei Zangen gespannt wird. Zu beachten ist, dass das Ziehen des Drahtes langsam erfolgt, da er sonst reißen kann. Anschließend werden an den Enden Ösen gebogen.

Für einen $8\text{x}8\text{x}8$ Kubus werden 256 Drähte mit einer Länge von 300mm benötigt. Die Rechnung ergibt sich folgendermaßen:

Formel 13 - Berechnung der benötigten Drähte pro LED-Wand

$$\underbrace{\left(8\left[\frac{LED}{Zeile}\right]\cdot 1\,[PIN]\right)}_{\text{Anode}}+\underbrace{\left(8\left[\frac{LED}{Spalte}\right]\cdot 3[PIN]\right)}_{\text{Kathoden}}=32\left[\frac{Drähte}{Wand}\right]$$

Formel 14 - Berechnung der benötigten LED-Verbindungen

$$8\,[Reihen]\cdot 8\,[Spalten]\cdot 4\,[PIN]=256\,Verbindungen$$

Zusätzlich werden weitere 20 Verbindungsdrähte für ein Zusammensetzen des Gitternetzes benötigt.

5.2.3 Bau des Gitternetzes

Für die Konstruktion des Gitternetzes wird auf eine Acrylplatte ein $8x8$ Raster mit einem Abstand von $35mm$ aufgezeichnet. Eine Vorlage befindet sich im Anhang auf DVD unter dem Namen „Schablone_Gitternetz.pdf“.

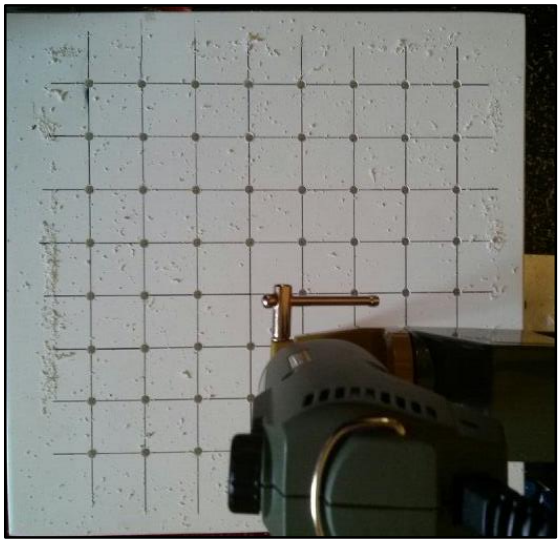


Abbildung 059 - Konstruktion eines Gitternetzes auf einer Acrylplatte

Anschließend erfolgt eine Bohrung von $5mm$ Durchmesser an jeder Linienkreuzung. Dadurch entsteht eine Grundplatte mit einer Gitternetzstruktur. Damit die einzelnen LEDs miteinander verlötet werden können, ist eine weitere Konstruktion erforderlich (Abbildung 060).

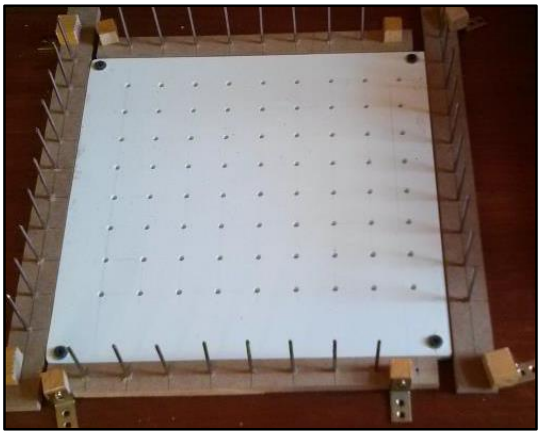


Abbildung 060 - Konstruktion zur Halterung und Einspannen eines Drahtes

Diese besteht aus einer Holzgrundplatte und vier MDF-Seitenstreifen, welche mit kleinen Winkeln arretiert sind, wodurch diese beweglich bleiben. An jedem Streifen werden jeweils acht Nägel im Abstand von $35\mathrm{mm}$ angebracht. Alternativ können Schrauben verwendet werden. Diese dienen zum Spannen des Kupferdrahtes. Desweiteren wird die bearbeitete Acrylplatte als Schablone eingelegt und arretiert. Anschließend werden die LEDs mit ihrem Kopf in die gebohrten Löcher der Acrylplatte gesetzt, so dass die Anoden nach oben zeigen. Daraufhin werden an den Nägeln Drähte gespannt, die im nächsten Schritt mit den Anoden verlötet werden (Abbildung 061).

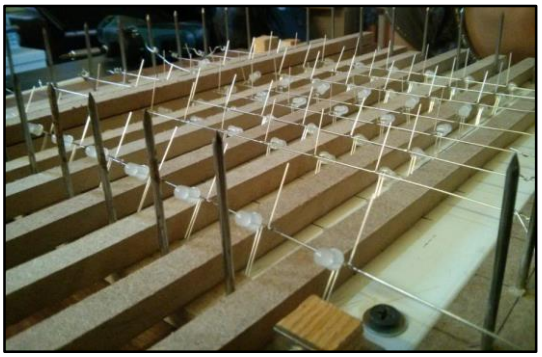


Abbildung 061 - Löten von Anoden in einer Reihe

Sind alle Anoden einer Reihe verbunden, so werden die Zeilen um 90° gedreht und wieder auf die Platte gelegt. Die zwei Kathoden (grün und blau) sollten in den gebohrten Löchern platziert werden.

Zur Stabilisierung werden zusätzlich $[275{,}0\text{x}0{,}5\text{x}300{,}0]\text{mm}$ MDF-Streifen zwischen Acrylplatte und Anodenverbindung geschoben. Damit die Köpfe der LEDs beim Verlöten genau im 90°-Winkel bleiben, wird ein Gitter aus $2{,}5\text{mm}$ starken MDF-Streifen zur Stabilisierung aufgelegt. Anschließend werden erneut Drähte gespannt, welche die roten Kathoden miteinander verbinden. Zwischen diese werden $[300\text{x}20\text{x}300]\text{mm}$ Hölzer für einen gleichmäßigen Gitterabstand aufgelegt.

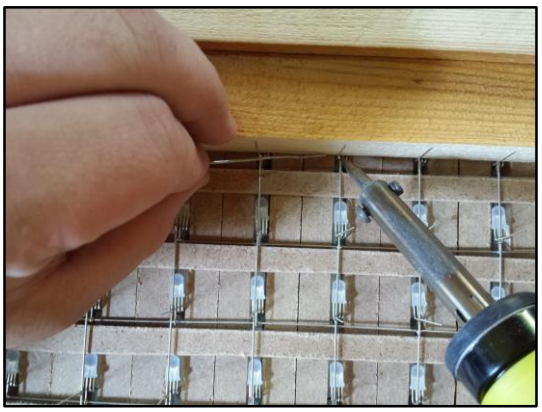


Abbildung 062 - Löten roter Kathode

Anzumerken ist, dass alle Verbindungen möglichst gerade gelötet werden.

Abbildung 063 - Löten grüner und blauer Kathoden

Nachdem die Anoden und die ersten Kathoden mit einem Draht verbunden sind, wird das Gitter um $180°$ gedreht und erneut auf die Schablone gelegt. Zur Arretierung dessen wird wiederholt das Stabilisierungsgitter aufgelegt. Anschließend werden nun gleich zwei Drähte um ein Nagelpaar gespannt.

Der Nagel fungiert somit als Abstandshalter zwischen beiden Drähten, um zu verhindern, dass versehentlich beim Löten beide Kathoden (grün und blau) miteinander verbunden werden. Dies hätte bei einer Inbetriebnahme einen unerwünschten optischen Effekt.

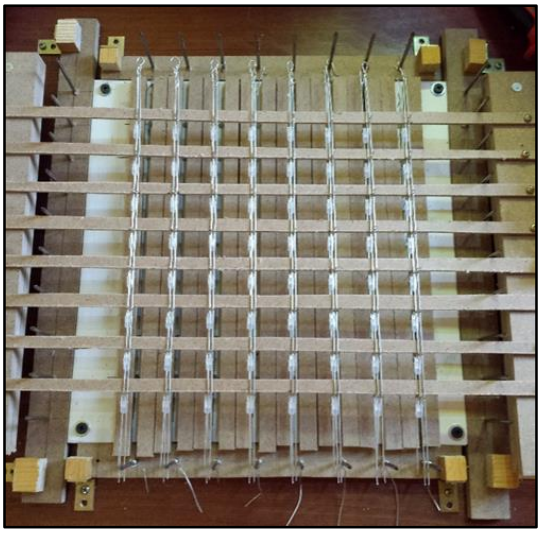


Abbildung 064 - fertige LED-Matrix

Die Abbildung 064 zeigt ein fertiges 2D-Gitternetz, bei dem die Anoden horizontal als Zeile und die Kathoden vertikal als Spalte, verbunden sind. Für die Realisierung eines Würfels werden weitere sieben Gitternetze benötigt. Damit diese zusammengesetzt werden können, ist eine weitere Hilfskonstruktion notwendig.

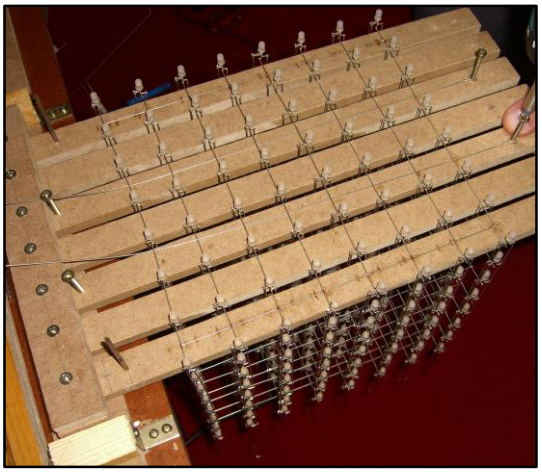


Abbildung 065 - Vereinigung aller Matrizen

Diese besteht aus einem Holzstreifen, welcher mit Winkeln auf einer Holzplatte arretiert ist. Sieben $[300\ x\ 10\ x\ 277]mm$ MDF Streifen dienen zur Halterung der Gitter und werden mit einem Abstand von $5\mathrm{mm}$ auf den Holzstreifen angebracht.

Anschließend können die Gitternetze auf die Konstruktion geschoben werden. Zur Stabilisierung werden am Ende zusätzlich Befestigungsstreifen angebracht.

Zum Spannen eines Verbindungsdrahtes dienen Schrauben und Nägel. Für die Einhaltung der Abstände können zusätzlich $[300\ x\ 20\ x\ 300]mm$ Holzpflöcke zwischen die LEDs geschoben werden. Ist eine Ebene miteinander verlötet, wird der Befestigungsstreifen abgeschraubt und die Gitter auf die nächste Ebene umgesetzt. Es werden erneut Drähte gespannt und mit den Anodenverbindungen verlötet.

Nachdem alle Gitternetze und zusätzliche Verbindungen zur Stabilität des Cubes gelötet sind, ist die 3D-Matrix vollständig (Abbildung 066).

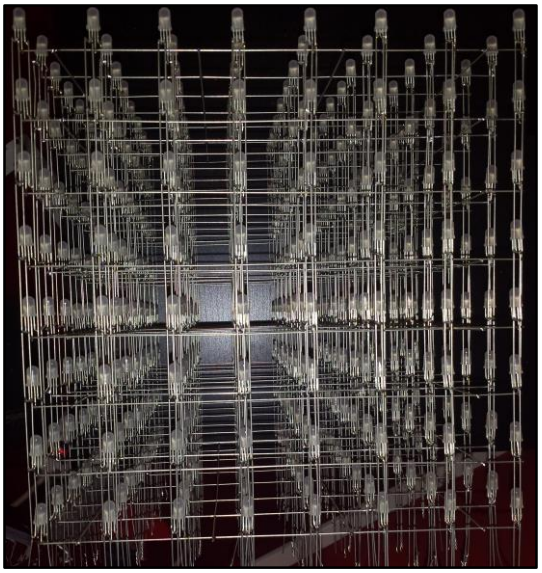


Abbildung 066 - vollständige LED 3D-Gittermatrix

Die Grafik zeigt die vollständige 3D-LED-Matrix bei der alle Anoden jeweils in einer Zeile als Ebenen und die farblichen Kathoden als Säulen miteinander verbunden sind.

5.2.4 Gehäuse

Für die Transportfähigkeit des Würfels ist ein festes Konstrukt notwendig. Das Gehäuse besteht aus einer Grundplatte, vier Seitenteilen, Abstandsstreifen, Abdeckplatte und Plexiglasscheiben. Die Abdeckplatte und zwei Seitenteile sollten nicht fest arretiert werden. Auf diese Weise wird erreicht, dass die Elektronik nicht sichtbar unter dem Würfel verstaut ist und für Vorführungszwecke frei zugänglich ist. Die Tabelle 14 gibt einen Überblick der benötigten Teile.

Nummer	Typ	Funktion	Maße [mm]	Menge
1.	Holzplatte	Bodenplatte	300 x 20 x 300	1
2.	Holzplatte	Deckplatte	300 x 20 x 300	1
3.	Holzstreifen	fest montierte Seitenwand	300 x 20 x 100	2
4.	Holzstreifen	abnehmbare Seitenwand	250 x 20 x 100	2
5.	Holzstreifen	Abstand- und Fixierstreifen	160 x 10 x 10	4
6.	Plexiglas	Wand	265 x 5 x 265	2
7.	Plexiglas	Wand	270 x 5 x 265	2
8.	Plexiglas	Deckel	270 x 5 x 270	1

Tabelle 14 - Benötigte Gehäusebauteile

Eine Bemaßung für die Konstruktion befindet sich im Anhang auf DVD unter dem Namen „Gehaeuse_Konstruktion.pdf“. Nachfolgend wird zur Erläuterung die Nummerierung der Teile benannt. Anzumerken ist, dass alle Bauelemente aus Holz nach ihrer Fertigstellung mit einem schwarzen Matt-Lack lackiert werden. Durch die schwarze Farbe wird erreicht, dass entstehendes Streuungslicht durch die Abdeckplatte absorbiert wird und unerwünschte Reflexionen vermieden werden.

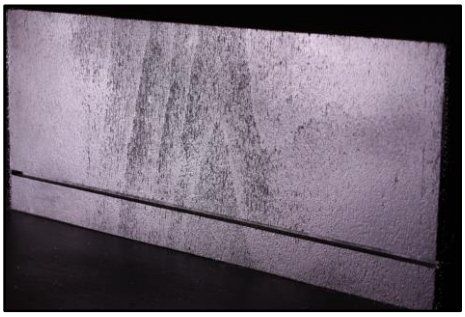


Abbildung 067 - Seitenteil mit Fräsnut für Leiterplatte

In die Seitenteile (3) sind Nuten mit einer Stärke von $3\mathrm{mm}$ gefräst. Diese dienen als Halterung für Leiterplatinen. Desweiteren werden Aussparungen für 20-PIN-Pfostenstecker, An- und Ausschalter sowie für die Stromversorgung eingearbeitet. Nach der Fertigung dieser Teile, werden sie auf der Bodenplatte montiert.

Die Seitenteile (4) sollten für Demonstrationszwecke abnehmbar sein.

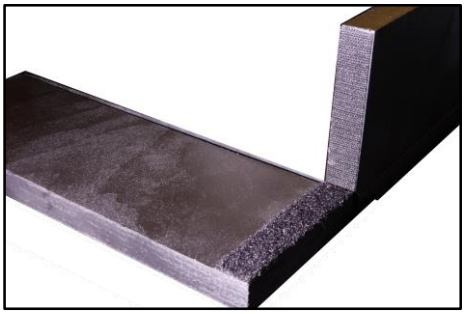


Abbildung 068 - Befestigung des Haken- und Flauschbandes zur Halterung der Seitenteile

Dies wird mittels Klettband realisiert. Das Hakenband befindet sich auf den zwei kurzen Seitenflächen der festen Seitenwände (3), während das Flauschband an den nicht-sichtbaren Seiten der beweglichen Seitenteile (4) bündig zum Rand befestigt wird. Um ein Verrutschen der abnehmbaren Deckplatte (2) zu vermeiden, dienen vier Abstands- bzw. Fixierstreifen, welche auf deren Unterseite so angebracht sind, dass diese mit allen vier Seitenwänden in Kontakt stehen.

Damit das 3D-Gitter mit dem Gehäuse verbunden werden kann, sind Bohrungen für die Anschlussdrähte in die Deckplatte notwendig.

Aufgrund des manuellen Herstellungsprozesses des 3D-Gitternetzes sind Abweichungen vorhanden. Ein LED-Abstand beträgt $(35 \pm 5)mm$. Eine Schablone der Bohrungen befindet sich im Anhang auf DVD unter dem Namen „Schablone_Bohrung.pdf“. Diese wird auf die Deckplatte (2) übertragen und anschließend Löcher für die Anschlussdrähte gebohrt. Jeder Draht ist mit einem Schrumpfschlauch von $0{,}5\mathrm{mm}$ Durchmesser versehen. Dies dient zum Schutz und Isolierung des Drahtes. Anschließend wird der LED-Würfel auf die Deckplatte gesteckt.

Um den Würfel gegen Umwelteinflüsse z.B. Wassertropfen und Staub zu schützen, ist ein Schutzgehäuse notwendig. Für dessen Herstellung wird klares, farbloses Plexiglas genutzt. Damit das Gehäuse nicht auf der Deckplatte (2) verrutscht, werden Nuten von $5\mathrm{mm}$ in die Abdeckplatte gefräst.

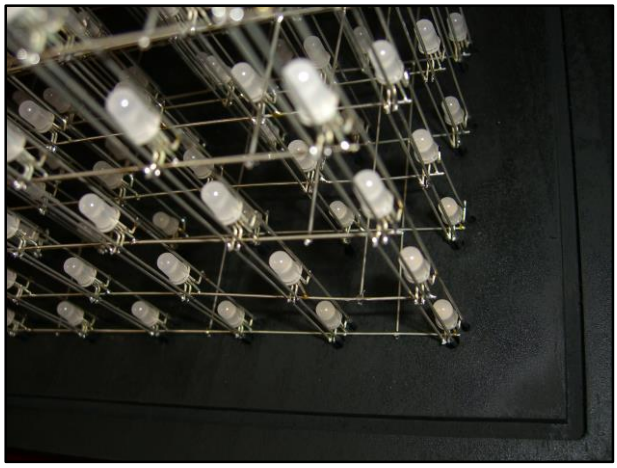


Abbildung 069 - Nut für Plexiglasschutzhaube

Somit kann der Plexiglaswürfel fixiert werden, was die Reinigung seiner Oberfläche erleichtert.

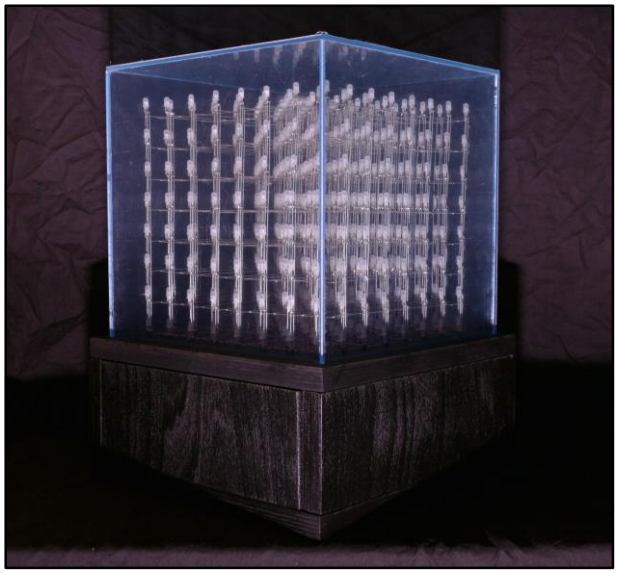


Abbildung 070 - Fertiger RGB-LED-Cube auf einem Holzgehäuse

Die Abbildung 070 zeigt den RGB-LED-Cube auf dem fertigen Gehäuse.

5.2.5 Verbindung- und Isolierung des Würfels

Zur Kopplung des Würfels mit der Ansteuerungselektronik werden Drähte mit einem Durchmesser von $0{,}25\mathrm{mm}$ genutzt. Dafür dient ein 80-PIN-Flachbandkabel. Die folgende Tabelle gibt einen Überblick über die Komponenten.

Nummer	Typ	Funktion	Maße [mm]	Menge
1.	80-PIN Flachbandkabel	Verbindung	-	3
2.	MQS Buchse	Verbindung	-	192
3.	Schrumpfschlauch	Isolierung	0,5 x 600	192
4.	Schrumpfschlauch	Isolierung	1,2 x 600	32
5.	Schrumpfschlauch	Isolierung	5,0 x 500	6
6.	Schrumpfschlauch	Isolierung	10,0 x 500	6
7.	40-PINBuchsenleiste	Verbindung	-	18

Tabelle 15 - Auflistung des Zubehörs zur Isolierung und Verbindung der Elemente

Die Anschlussdrähte sind entsprechend gekürzt, dass diese ca. $0{,}5\mathrm{mm}$ aus der Abdeckplatte herausragen.

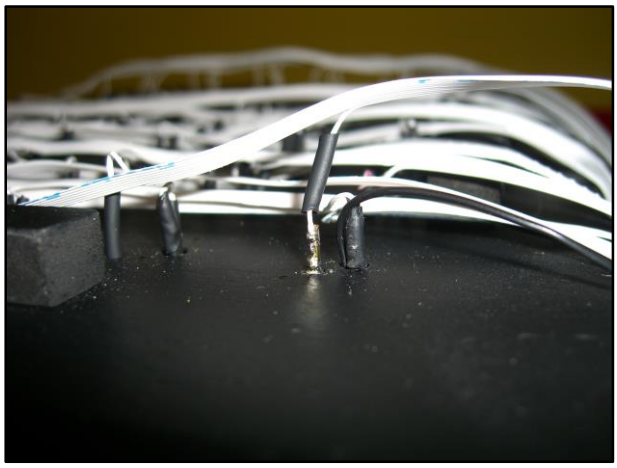


Abbildung 071 - Unterseite der Deckplatte mit LED-Anschlussdrähten

Eine MQS-Buchse auf jedem der Drähte dient der weiteren Verarbeitung. Innerhalb einer Wand sind alle acht Steckverbindungen einer Farbe jeweils mit den einzelnen Adern des Flachbandkabels verlötet und die Kontaktstellen mit Schrumpfschläuchen isoliert. Diese acht Drähte sind zu einem Bündel zusammengelegt, so dass pro Wand drei Drähte (rot, grün, blau) vorhanden sind. Schrumpfschläuche mit verschiedenen Durchmessern, welche ineinander verschachtelt werden, dienen für eine ausreichende Isolierung und gestalten aus mehreren Drähten einen Kabelbus.

Abbildung 072 - Kreation eines Kabelbussystems am RGB-LED-Cube

Eine Zusammenführung der Kabel von vier Wänden einer gemeinsamen Farbe (32 Säulen) ergibt einen Kabelbus. Daraus resultierend sind insgesamt sechs Kabelbusse vorhanden.

5.3 RGB-LED-Cube Elektronik

Für dieses Projekt liegen 512 RGB-LEDs vor. Die nachfolgende Tabelle 16 gibt einen Überblick der elektrischen Eigenschaften dieser LEDs.

Elektrische Eigenschaften			
Farbe	Rot	Grün	Blau
Flussspannung	2,2V	3,3V	3,4V
max. Strom	3 x 20mA		

Tabelle 16 - Elektrische Eigenschaften der angewandten RGB-LED

Formel 15 - Gesamtamperezahl für 1536 LEDs bei 0,02A pro LED

$$[(512\ LEDs) \cdot (3\ Farben)]\ \cdot 0{,}02A = 30{,}72A$$

Die Berechnung zeigt, dass ein Netzteil nötig ist, welches mindestens $31\mathrm{A}$ bei $20\mathrm{mA}$ liefert. Diese Werte sind sehr hoch und ineffizient. Ziel ist es den Würfel möglichst energieeffizient zu betreiben. Ein Zeit-Multiplex-Verfahren dient zur Ansteuerung einzelner Ebenen und Farben (Säulen). Anhand dieser Grundlage wird in der tatsächlichen Zeit nur eine Farbe einer LED auf einer Ebene leuchten. Die Säulen und Ebenen werden in kurzen Intervallen zu- und abgeschaltet, so dass durch die Trägheit des Auges alle LEDs als durchgängig leuchtend wahrgenommen werden.

Formel 16 - Gesamtamperezahl für 64 LEDs bei 0,02A pro LED

$$64\ LEDs \cdot 0{,}02A = 1{,}28A$$

Für einen Betrieb von 64 LEDs mit 20mA werden $1{,}28\text{A}$ benötigt. Aufgrund dieses Verfahrens und der ermittelten Werte, ist für den Betrieb des Würfels ein $+5\text{V}$ Netzteil mit einer maximalen Stromstärke von 3A ausreichend.

5.3.1 RGB-LED-Vorwiderstand

Durch den Herstellungsprozess bedingt, weisen LEDs unter anderem Toleranzen bezüglich der Durchlassspannungen auf. Die Ermittlung eines geeigneten Vorwiderstandes für die einzelnen Kathoden erfolgt rechnerisch durch die Formel 06.

$$R_V[\Omega] = \frac{[U_G - U_{LED}]V}{[I]A}$$

Bei einem Betrieb von $+5\text{V}$ und den angegebenen Werten aus der Tabelle 16 ergeben sich folgende Vorwiderstandswerte:

Farbe	Durchlassspannung [V]	Gleichung	Vorwiderstand R_V [Ω] (rechnerisch)	Nächst höherer Widerstand [Ω]
Rot	2,2	$\frac{[5 - 2{,}2]V}{0{,}02A}$	140	140
Grün	3,3	$\frac{[5 - 3{,}3]V}{0{,}02A}$	85	85
Blau	3,4	$\frac{[5 - 3{,}4]V}{0{,}02A}$	80	82

Tabelle 17 - Rechnerisch ermittelte Vorwiderstandswerte

Aus dem RGB-Farbraum bekannt, ergibt sich die Farbe Weiß anhand einer gleichmäßigen Abstrahlung aller drei Farbenanteile. Dominiert der rötliche Anteil wird ein warmer Weißton erreicht. Ein kühles Weiß wird durch ein stärkeres leuchtendes Blau erzeugt.

Aufgrund der Widerstandstoleranzen wird der nächst höhere verfügbare Widerstand gewählt. Daraus resultierend würde die Farbe Blau einen gleichen Vorwiderstand wie grün aufweisen, welches zu einem höheren Blauton führt. Eine optische Einschätzung unter Betrachtung der Toleranzen dient als Hilfe zur Anpassung der Widerstandwerte. Die folgende Tabelle 18 gibt einen Überblick der angewandten Widerstände.

Farbe	Durchlassspannung [V]	Gleichung	Vorwiderstand [Ω]
Rot	2,2	$\frac{[5 - 2{,}2]V}{0{,}02A}$	$R_{VR} = 140$
Grün	3,2	$\frac{[5 - 3{,}2]V}{0{,}02A}$	$R_{VG} = 91$
Blau	3,4	$\frac{[5 - 3{,}4]V}{0{,}02A}$	$R_{VB} = 82$

Tabelle 18 - Optisch ermittelte Vorwiderstandswerte

Ersichtlich ist, dass die optisch ermittelten Werte mit den rechnerischen Werten bis auf den Widerstand der Farbe Grün übereinstimmen. Die Farbe Grün sollte eine kleine Abweichung gegenüber der Farbe Blau aufweisen, so dass ein 91Ω Widerstand angewandt wird.

Durch die Verwendung einer Puls-Wide-Modulation (PWM) auf unterschiedlichen Farbkanälen, kann die Intensität des RGB-Farbraumes dynamisch angepasst werden. Daraus resultierend können auch andere Vorwiderstandswerte Verwendung finden. Die exakten Frequenzen der LED können zum Beispiel durch ein Spektrometer ermittelt werden. Aus zeitlichen und finanziellen Gründen wird dieses Verfahren nicht angewandt.

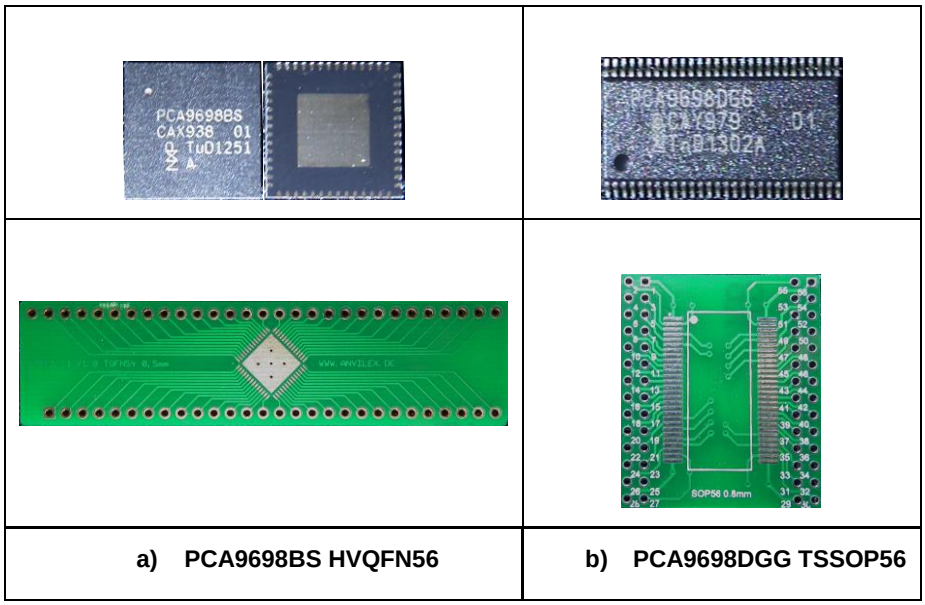
5.3.2 I²C-GPIO-Expander

Durch die Konstruktion des RGB-LED-Cubes sind die LEDs ebenenweise mit den Anoden verbunden. Diese Ebenen dienen einer Spannungsversorgung einer jeden LED. Damit diese leuchten, müssen die Kathoden auf GND geschalten werden.

Das Altium und die Altera Entwicklungsboards bieten nur eine begrenzte Anzahl an externen GPIO-Anschlüssen. Aus diesem Grund ist ein sogenannter GPIO-Expander (GPIO-Erweiterung) notwendig.

Jeder einzelne Port des Expanders ist über einen Vorwiderstand mit einer Säule (64 Säulen pro Farbe) des Würfels verbunden, welche in einem definierten Intervall auf GND geschalten wird. Durch das Schalten und Anliegen einer Spannung entsteht ein Stromfluss und die LEDs leuchten.

Angewandt wird ein PCA9698 40-Bit GPIO-Expander mit 56-PINs der Firma NXP [26]. Die nachfolgende Abbildung 073 zeigt zwei Bauformen der ICs mit der dazugehörigen Adapterplatine.



a) PCA9698BS HVQFN56 | **b) PCA9698DGG TSSOP56**

Abbildung 073 - Übersicht von NXP PCA9698 Bauformen und passende Adapterplatinen

Für die Ansteuerung des Cubes wird der PCA969BS in einer sogenannten HVQFN56 Bauform angewandt. Das q*uad flat no lead* (QFN-Package) sind SMD-Bauteile, die Anschlusskontakte auf der Unterseite besitzen, mit denen sie unmittelbar auf der Leiterplatte montiert werden.

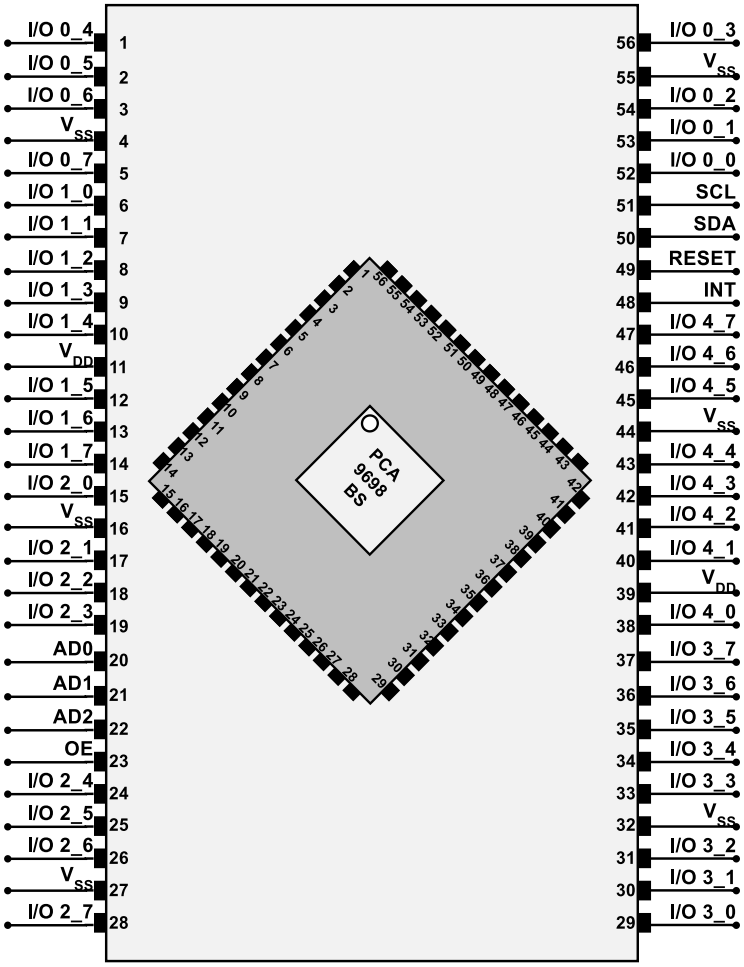


Abbildung 074 - PCA9698BS auf einer HVQFN56 Adapterplatine

Der IC besteht aus 56 Anschlüssen und hat einen PIN-Abstand von $0{,}5\mathrm{mm}$, welche gleichmäßig auf den vier Seiten verteilt sind. Dieser wird auf eine QFN56 Adapterplatine für eine weitere Bearbeitung und anschließend auf einer Lochrasterplatine aufgebracht.

Alle I/O Ports repräsentieren jeweils einen Anschluss, welcher als Ein- oder Ausgabe fungiert. Eine Spannungsversorgung erfolgt über $\mathrm{V_{DD}}$ -PINs. Eine Verbindung zum Minuspol wird durch die V_{SS}-PINs erreicht. In der nachfolgenden Tabelle 19 sind die Anschlüsse zusammengefasst.

Portbezeichnung	Funktion	Funktionsbeschreibung	Anzahl
50	SDA	I²C-Datenkanal	1
51	SCL	I²C-Taktkanal	1
49	Reset	Chip-Reset	1
48	INT	Interrupt	1
23	OE	Hardware PWM	1
11, 39	V_{DD}	Spannungsquelle	2
20, 21, 22	AD (X)	Hardwareadressierung	3
4, 16, 27 ,32 44 , 55	V_{SS}	GND	6
I/O X_0 … X_7	I/O	Input- und Output	40

Tabelle 19 - Zusammenfassung der Anschlüsse des PCA9698 auf einer Adapterplatine [26]

Die Funktionen einzelner Kontakte sind im Datenblatt beschrieben [26]. Eine Beispielschaltung zum Anschluss des IC befindet sich auf der Seite 31 des Datenblatts und im Anhang 1 dieser Dokumentation.

Die nachfolgende Abbildung 075 ist ein Ausschnitt einer Beispielschaltung für den Betrieb einzelner Säulen an einem PCA9698.

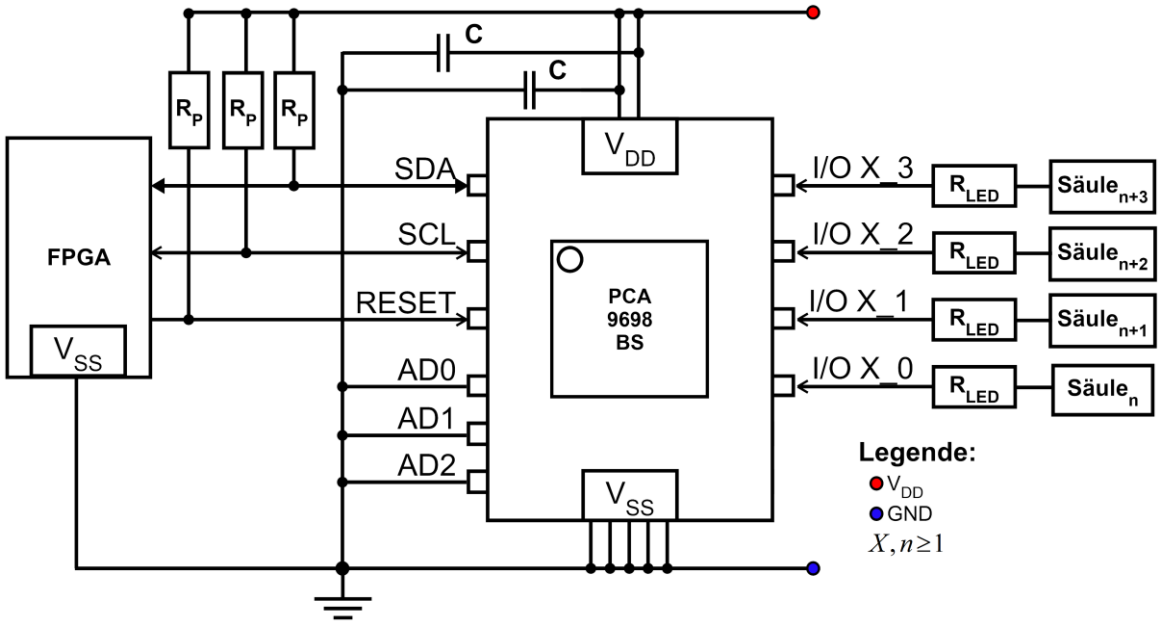


Abbildung 075 - Beispielschaltung PCA9698BS

Zur Glättung der Eingangsspannung dient jeweils ein 100nF Kondensator (C) zwischen V_{DD} und der Masse. Die Kanäle (SCL, SDA, Reset) des Open-Collector-I²C-Bussystems werden mit einem Pull-Up-Widerstand R_P von $4{,}7\text{k}\Omega$ betrieben. Das Schalten einer jeden einzelnen Säule des Würfels (192 Stück) erfolgt durch jeweils einen I/O-Anschluss des Expanders. Zwischen einer Säule und dem Port wird ein Vorwiderstand R_{LED} entsprechend der ermittelten Werte aus Tabelle 18 verbaut.

Für eine übersichtliche Aufteilung wird eine Frontseite des Würfels definiert. Als Zeichen wird an einer Ecke des Würfels der Draht schwarz lackiert.

Abbildung 076 - Frontseitenmarkierung des Cubes

Daraus resultierend sind 32 Säulen einer gemeinsamen Farbe mit einem Expander verbunden. Insgesamt sind 64 Säulen auf zwei Expander verteilt. Diese Verteilung erfolgt gemäß des Aufbaus des Cubes.

Dies ergibt eine Aufteilung von zwei „PCA9698BS“ pro Farbe, bei welchen jeweils acht PINs keinen Zustand besitzen und für fortführende Anwendungen verwendet werden können. Die nachfolgende Abbildung 077 verdeutlicht das Schema.

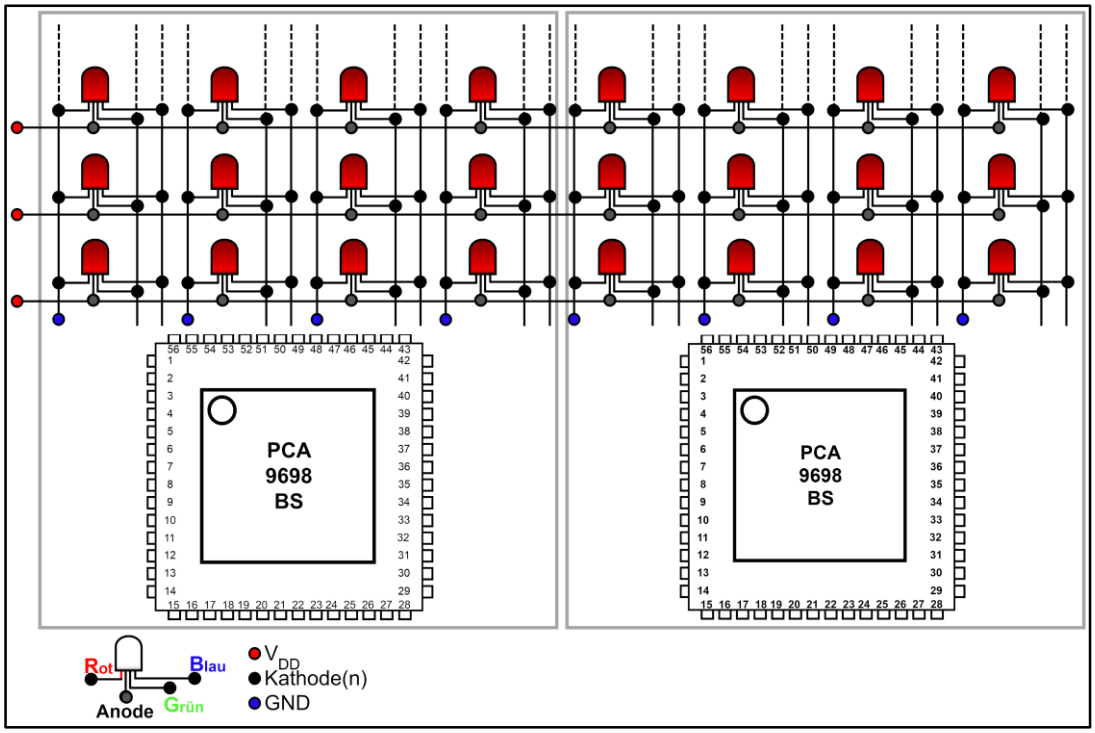


Abbildung 077 - Übersicht der angewandten GPIO-Expander für die Farbe Rot des RGB-LED-Cubes

Diese Konstruktion ermöglicht eine koordinierte Aufteilung für eine Ansteuerung des Expanders. Die einzelnen Register repräsentieren somit eine LED-Wand des Würfels, während die einzelnen Ports eine Säule darstellen.

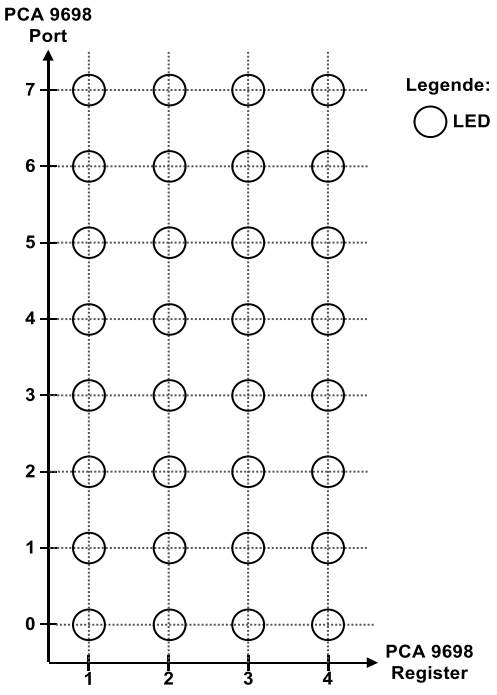


Abbildung 078 - Aufteilung des PCA9698 in ein Koordinatensystem zur Ansteuerung des Cubes

Die Abbildung 078 zeigt die Aufteilung des Expanders in ein zweidimensionales Koordinatensystem. Dies entspricht einer Draufsicht auf den Würfel. Die nachfolgende Abbildung 079 verdeutlicht das Konzept der Ansteuerung.

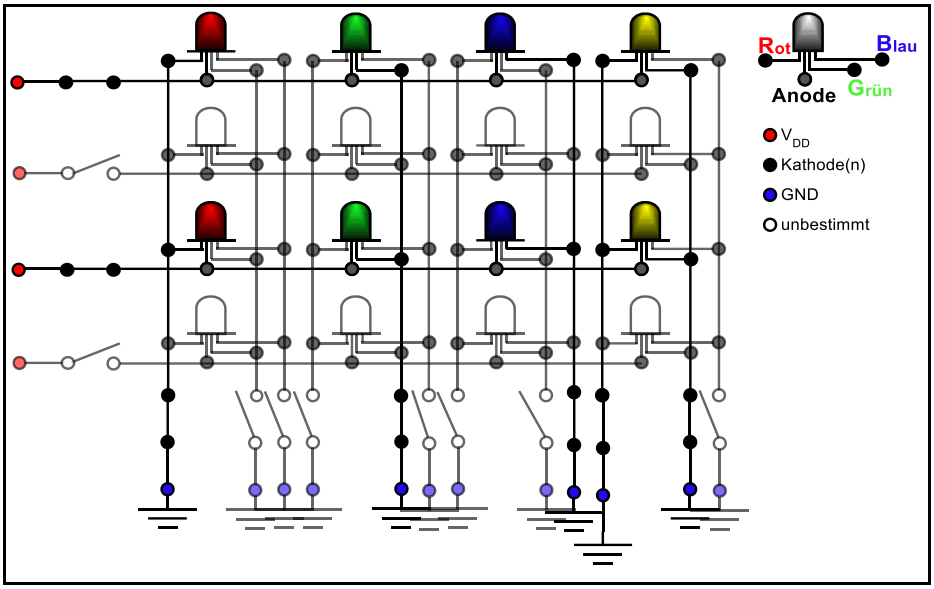


Abbildung 079 - Konzept für eine Ansteuerung von LEDs in einer Matrix

Wird eine Spannung auf eine Ebene angelegt und entsprechend die Farbe (Kathode) der LED von einer Säule auf GND gezogen, so leuchtet diese. Wird auf eine andere Ebene zur gleichen Zeit eine Spannung angelegt, so leuchten diese LEDs parallel in der gleichen Farbe. Auf Ebenen, wo keine Spannung angelegt wird, bleiben die LEDs offline. Werden unterschiedliche Farben einer LED in einer Säule auf GND gezogen, so entsteht eine einfache RGB-Farbmischung mit einer gleichbleibenden Farbintensität. Damit optisch einzelne LEDs leuchten und eine volle RGB-Farbmischung sowie eine unterschiedliche Lichtintensität erreicht wird, ist zum Beispiel ein Zeitverfahren notwendig.

Aufgrund der Trägheit des menschlichen Auges ist es möglich eine LED mit 50Hz zu betreiben. Umwelteinflüsse, z.B. Raumlicht, haben ebenfalls Auswirkungen auf die optische Wahrnehmung. Aus diesem Grund sind 60Hz ($\frac{1}{60\ Hz} = 0{,}016\ s$) ein idealer Wert. Für die optische Wahrnehmung genügt es, dass eine LED sechzig Mal pro Sekunde an- und ausgeschaltet wird, damit diese für das menschliche Auge konstant leuchtet.

Wird einer Ebene für $\frac{1}{60}\ s^{-1}$ eine Spannung angelegt und parallel die gewünschte Kathodenspalte auf GND gezogen, so leuchten nur diese LEDs. Anschließend wird die Ebene offline geschalten und eine andere Ebene unter Spannung gelegt. Parallel erfolgt erneut das Zuschalten des GND an der gewünschten Spalte. Bei diesem Projekt liegen acht Ebenen vor, woraus sich folgende Rechnung ergibt:

Formel 17 - Berechnung von acht Ebenen bei 60Hz

$$8\ Ebenen\ \cdot 60Hz = 480Hz$$

Für acht Ebenen sind 480Hz notwendig, damit jede Ebene abwechselnd für $0{,}016s$ leuchtet.

Die Helligkeitswerte der LEDs können durch eine Soft- oder Hardware-Pulsweiten-modulation reguliert werden. Der Ein- und Ausschaltzustand beträgt in der Regel 50% (Abbildung 080).

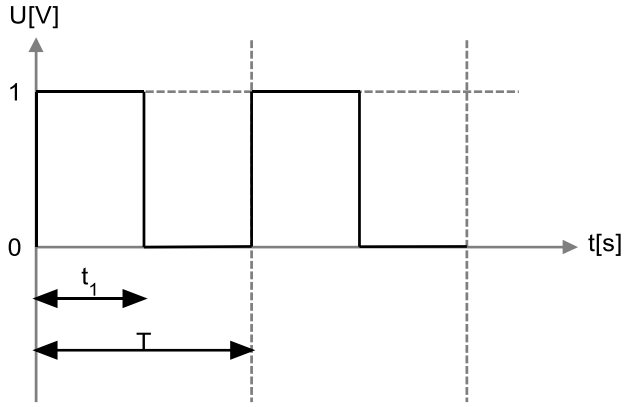


Abbildung 080 - Signalverlauf eines Tastverhältnisses

Die Grafik verdeutlicht, dass für eine Periodendauer von einer Sekunde die LED für $0{,}5s$ angeschaltet und für die restliche Zeit ausgeschaltet ist. Das Verhältnis der Zeit zwischen dem HIGH- und LOW-Anteil des Signals wird bei einer PWM verändert, während die Periodendauer konstant bleibt. Dieses Verhalten wird als Tastverhältnis bezeichnet. Die folgende Tabelle 20 zeigt die Auswirkung bei Veränderung des Verhältnisses einer roten LED.

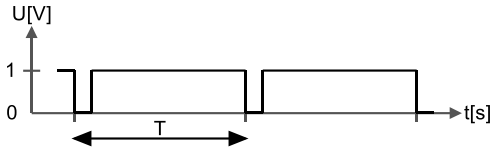
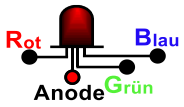
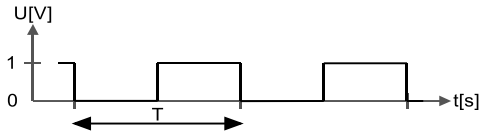
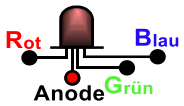
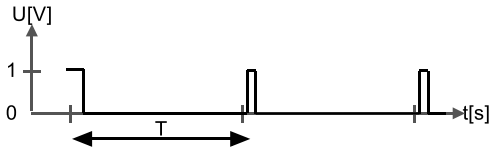
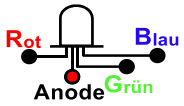
Nummer	PWM-Signal	LED-Helligkeit
1		
2		
3		

Tabelle 20 - Funktionsweise der PWM bei einer LED

Erfolgt eine Vergrößerung des HIGH-Pegel-Anteils (1), verkürzt sich die Ausschaltzeit der LED. Sie leuchtet länger und gegenüber eines Referenzwertes (2) heller. Wird der HIGH-Pegel-Anteil verkürzt (3), ist die Ausschaltzeit größer und somit die LED dunkler. Anzumerken ist, dass diese einige hundert Nanosekunden Einschaltzeit benötigt.

Resultierend aus dem theoretischen Ansatz zur Ansteuerung der 3D-Matrix und den Helligkeitswerten ergibt sich folgende Abbildung 081:

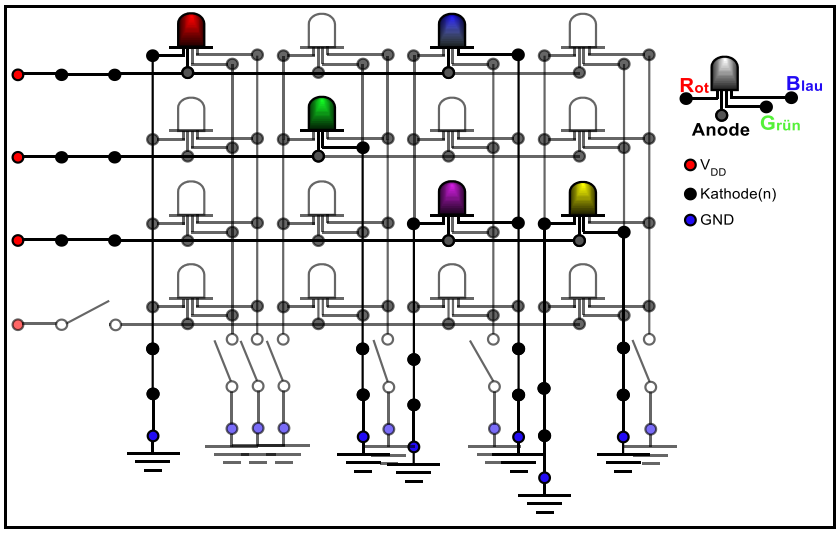


Abbildung 081 - Konzept für eine Ansteuerung einer 3D-Matrix

Deutlich ist, dass durch eine zeitliche Ansteuerung von Anoden und Kathoden die LEDs konstant leuchten und durch einen zeitlichen Verlauf eine Farbmischung möglich ist. Weitere Realisierungsansätze, zum Beispiel einzelne LEDs separat mit Spannung zu versorgen oder auf GND zu ziehen, werden aufgrund des höheren Bedarfs an Hardwarekomponenten nicht innerhalb der Thesis beschrieben.

Der PCA9698 bietet die Möglichkeit 64 7-Bit Teiladressen zu bilden. Für den RGB-Würfel werden sechs ICs des gleichen Typs verwendet. Eine Adressierung erfolgt durch die Teiladresse des IC-Bausteines, welcher aus dem LOW- oder HIGH Pegel (A2, A1, A0) gebildet wird. Die folgende Tabelle 21 dient zur Übersicht einer möglichen Bitvergabe.

	A2	A1	A0	Adresse
1	GND	GND	GND	000
2	GND	GND	V_{DD}	001
3	GND	V_{DD}	GND	010
4	GND	V_{DD}	V_{DD}	011
5	V_{DD}	GND	GND	100
6	V_{DD}	GND	V_{DD}	101

Tabelle 21 - Bitvergabe PCA9698

Eine Einteilung der Adressen erfolgt durch eine Verbindung der jeweiligen Ports. Ein HIGH-Pegel wird erreicht, indem dieser mit der Spannung verbunden wird. Der LOW-Pegel wird analog auf GND geschalten. Desweiteren verfügt dieser IC über zusätzliche Funktionen, zum Beispiel einen OE-Anschluss, welcher für eine hardwareseitige PWM Steuerung angewandt werden kann. In dieser Arbeit werden zusätzliche Anschlüsse OE, INT nicht näher erläutert. Der GPIO-Expander „PCA9698“ bietet 48 In-und Output-Anschlüsse, welche in fünf Register aufgeteilt sind. Jedes Register besteht aus sieben Teilanschlüssen über welche Daten gelesen und geschrieben werden können. Die folgende Tabelle 22 gibt einen Überblick der Bitfolgen.

Hex	D5	D4	D3	D2	D1	D0	Name	Type	Funktion
18h	0	1	1	0	0	0	IOC0	Lesen/Schreiben	I/O Konfiguration Register Bank 0
19h	0	1	1	0	0	1	IOC1	Lesen/Schreiben	I/O Konfiguration Register Bank 1
1Ah	0	1	1	0	1	0	IOC2	Lesen/Schreiben	I/O Konfiguration Register Bank 2
1Bh	0	1	1	0	1	1	IOC3	Lesen/Schreiben	I/O Konfiguration Register Bank 3
1Ch	0	1	1	1	0	0	IOC4	Lesen/Schreiben	I/O Konfiguration Register Bank 4
…	…	…	…	…	…	…	-	-	Reserviert

Tabelle 22 - I/O Konfiguration PCA9698 [26]

Ersichtlich ist, dass ein PCA9698 über ein fünf Port-Register verfügt. Eine kontinuierliche Datenübermittlung kann ab einem „Startregister“ erfolgen. Dabei wird das Register ausgewählt und Daten für dieses und nachfolgende Ports übertragen. Die Register können auch einzeln angewandt werden.

5.3.3 LED-Treiberschaltung

Ein FPGA bietet pro GPIO-Anschluss einen maximalen Strom von $20\mathrm{mA}$ bei $3{,}3\mathrm{V}$. Es werden zirka $1{,}28\mathrm{A}$ für einen Betrieb von 64 LEDs benötigt. Die Spannungsversorgung erfolgt über ein externes Netzteil. Die Ansteuerung einzelner Ebenen erfolgt über eine Treiberschaltung.

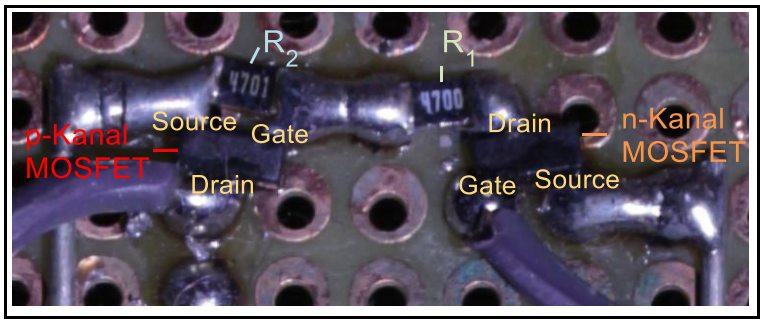


Abbildung 082 - LED-Treiberschaltung auf einer Lochrasterplatine

Diese besteht aus einem n-Kanal- und p-Kanal-MOSFET sowie zwei Widerständen, welche näher erläutert werden.

Für die Umsetzung dieser Schaltung wird ein n-Kanal-MOSFET vom Anreicherungstyp als Schalter angewandt.

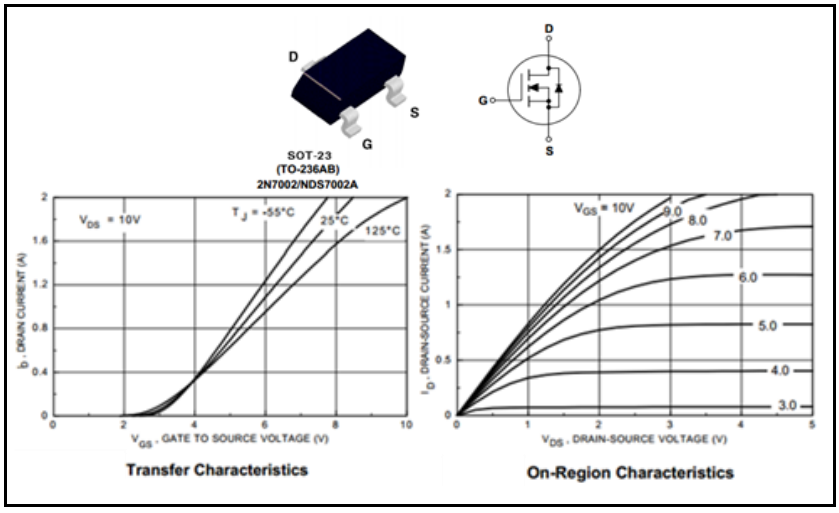


Abbildung 083 - n-Kanal-MOSFET Fairchild 2N7002 [54]

Die Abbildung 083 ist ein Auszug aus dem Datenblatt [54] eines „Fairchild 2N7002“. Ersichtlich ist, dass dieser Transistor ab einer geringen Schwellwertspannung von zirka $2\mathrm{V}$ arbeitet. Damit eignet sich der MOSFET für eine Gate-Ansteuerung direkt vom FPGA. Die Ausgangskennlinie zeigt, dass dieser bei einer Spannung von $+5\mathrm{V}$ ein $I_D \approx 0{,}8A$ im Sättigungsbereich betrieben wird. Ein Betrieb des gesamten Würfels ist mit diesem Transistor nicht möglich. Er fungiert als Schalter und öffnet das Gate eines p-Kanal-MOSFETs, welcher ebenfalls als Schalter fungiert.

Die Wahl ist ein „Vishay Si2323DS“ [55]. Dieser ist ein Anreicherungstyp p-Kanal-MOSFET, welcher mit einer maximalen Spannung von 20V betrieben werden kann.

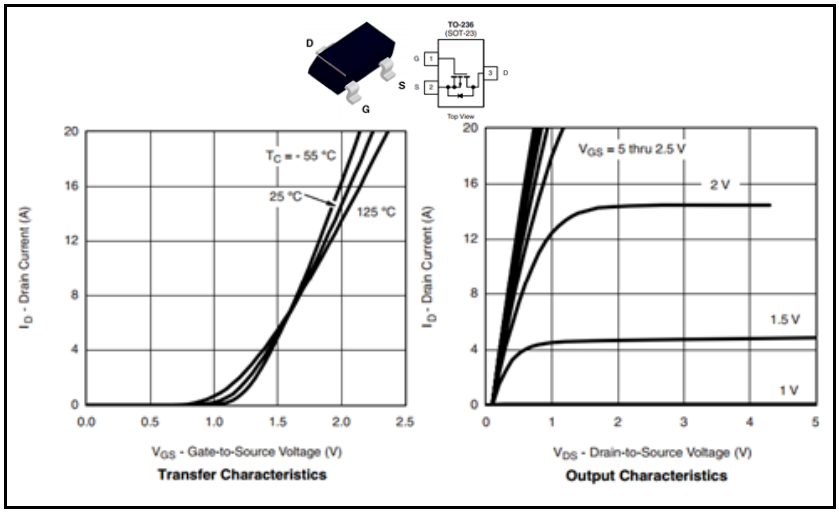


Abbildung 084 - p-Kanal-MOSFET Vishay Si2323DS [55]

Er arbeitet mit einer Schwellspannung von zirka $+0{,}75\text{V}$ und ein Betrieb mit $+5\text{V}$ ist mit mehreren Ampere möglich. Damit ist der MOSFET eine geeignete Wahl für die Umsetzung des Projekts.

Nachfolgend wird der Einsatz der MOSFETs als Treiberschaltung erläutert.

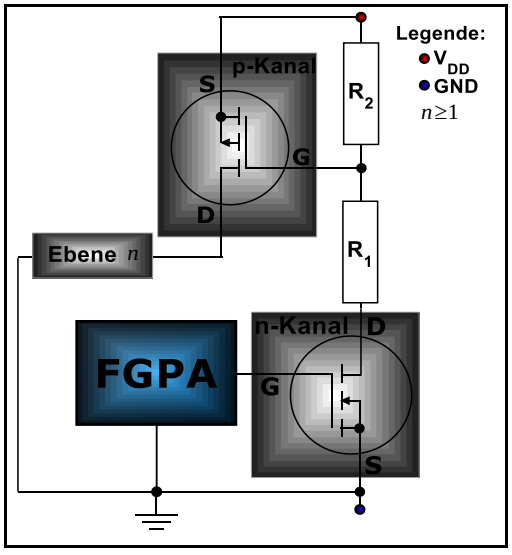


Abbildung 085 - Schaltplan der LED-Treiberschaltung

Liegt vom FGPA keine Spannung an, so ist der n-Kanal-MOSFET auf GND. Die Schwellspannung des n-Kanal-MOSFETs liegt bei $1V$. Kommt vom FPGA ein Signal mit einer Spannung von $+3{,}3V$ (mindestens $1{,}2V$), so befindet $U_{gs} > U_{th}$ der FET wird leitend. Dabei schaltet er das Gate des p-Kanal-MOSFETs, welcher mit $+5V$ Source anliegt. Dieser schaltet U_{DS} und ist somit leitend. Sobald der FPGA keinen Spannungsimpuls abgibt, sperrt sich der n-Kanal-MOSFET. Der Pull-Up-Widerstand R_2 ($4{,}7k\Omega$) zieht das Gate des p-Kanal-MOSFETs sofort auf HIGH. Der p-Kanal-MOSFET wird positiv und sperrt sich. R_1 dient als Pull-Down sowie als Schutzwiderstand und wird mit 470Ω gewählt. Die Umsetzung der Schaltung erfolgt auf einer Lochrasterplatine[8].

Für jede Ebene wird eine Treiberschaltung genutzt. Ein weiterer Realisierungsansatz besteht darin, mehrere Treiberschaltungen über einen GPIO-Expander zum Beispiel mittels des I²C-Protokolls anzusteuern. Bei diesem Ansatz entfällt der n-Kanal-MOSFET und R_1.

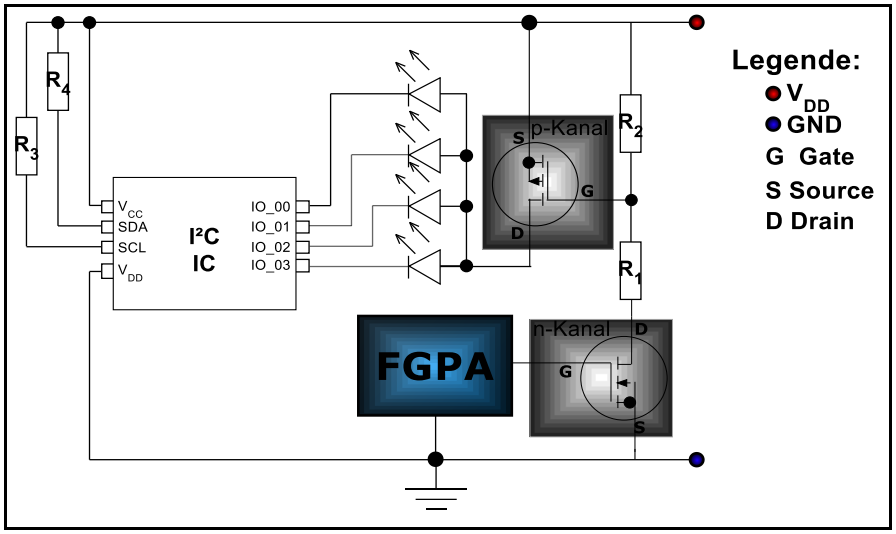


Abbildung 086 - Konzept eines Schaltplanes zur Ansteuerung der RGB-LED-Matrix

Die Abbildung 086 zeigt einen kompletten theoretischen Ansatz zur Ansteuerung einer LED-Matrix.

[8] universelle Platine mit Kupferstreifen oder -Punkten, die in einem Raster mit einem Abstand von 2,54mm aufgebracht sind.

Die Säulen der Matrix werden mit Hilfe eines I²C-Slaves (GPIO-Expander), welcher über einen I²C-Master (FPGA) angesprochen wird, in einem zeitlichen Intervall auf GND gezogen.

5.3.4 Schnittstellen und PIN-Belegung

Die Signalübertragung zwischen einem FPGA-Entwicklungsboard und dem Cube erfolgt über ein Bussystem. Durch die Wahl der PIN-Belegung ist ein Betrieb des Würfels mit einem Altera-Board sowie einem Altium NB3K möglich.

Auf einem Altera-Entwicklungsboard (Cyclone 3, Cyclone 4) sind 40-PIN-GPIO-Anschlüsse vorhanden. Für dieses Projekt, werden diese, wie folgt belegt:

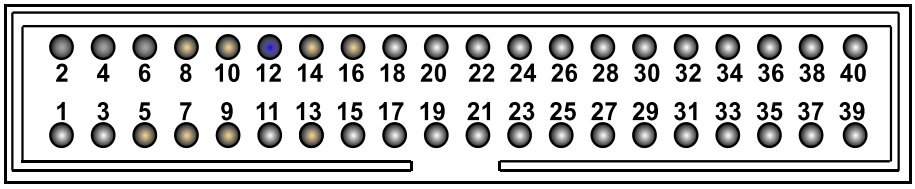


Abbildung 087 - RGB-LED-Würfel PIN-Belegung auf einem 40-PIN-GPIO_1 Altera DE0Board

Eine Datenübertragung des I²C-Protokolls erfolgt über die Ports 2, 4 und 6, welche grau markiert sind. Für einen gemeinsamen GND-Anschluss wird der PIN-12 verwendet (blau). Die einzelnen Signale der Ebenen werden über die gelb markierten PINs angesteuert. Zu beachten ist, dass ungenutzte PINs auf *„input tri-stated“* gelegt sind. Zur Sicherheit werden alle ungenutzten PINs des GPIO auf GND gelegt.

Eine vollständige PIN-Belegung für beide Boards befindet sich im Anhang unter dem jeweiligen Board-Bezeichnungs-Ordnern „Altera DE0 Board Cyclone III“ und „Altera DE0 Board Cyclone IV“ der Beispielprojekte. Die folgende Tabelle 23 gibt einen Überblick der GPIO_1 (JP2)-Belegung.

PIN-Nummer	Cyclone 3 JP2 Portbezeichnung	Cyclone 4 JP2 Portbezeichnung	Belegung
2	PIN_AA20	PIN_F13	I²C-SCL
4	PIN_AB20	PIN_T15	I²C-SDA
5	PIN_AA19	PIN_T14	Ebene_00
6	PIN_AB19	PIN_T13	I²C-Reset
7	PIN_AB18	PIN_R13	Ebene_01
8	PIN_AA18	PIN_T12	Ebene_03
9	PIN_AA17	PIN_R12	Ebene_02
10	PIN_AB17	PIN_T11	Ebene_04
11	-	-	V_{DD}
12	-	-	GND
13	PIN_Y17	PIN_T10	Ebene_06
14	PIN_W17	PIN_R11	Ebene_05
16	PIN_T15	PIN_R10	Ebene_07

Tabelle 23 - Auflistung angewandter Altera Boards GPIO-Anschlüsse

Das Altium NB3K verfügt über zwei GPIO-Anschlüsse. Eine neue PIN-Belegung ist nicht erforderlich und dadurch identisch zu der Altera Belegung.

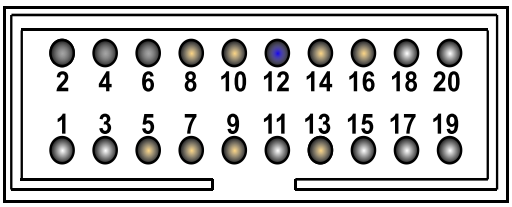


Abbildung 088 - RGB-Würfel PIN-Belegung auf dem 20-PIN-HA des NB3K

Die folgende Tabelle 24 gibt einen Überblick der GPIO-Belegung.

PIN-Nummer	Altium-Bezeichnung	Belegung
2	HA2	I²C-SCL
4	HA4	I²C-SDA
5	HA5	Ebene_00
6	HA6	I²C-Reset
7	HA7	Ebene_01
8	HA8	Ebene_03
9	HA9	Ebene_02
10	HA10	Ebene_04
12	HA12	GND
13	HA15	Ebene_06
14	HA14	Ebene_05
16	HA16	Ebene_07

Tabelle 24 - Auflistung angewandter Altium NB3K GPIO-Anschlüsse

Undefinierte GPIO-Anschlüsse werden mit GND verbunden. Eine 20-PIN-Pfostenbuchse am Gehäuse mit einer Weiterleitung zur Platine dient als Eingangsschnittstelle für eine Verbindung zwischen der Entwicklungsplattform und der Peripherie.

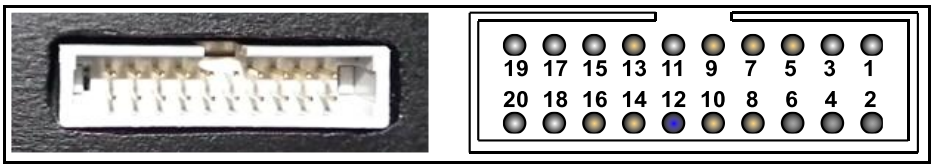


Abbildung 089 - 20-PIN-Gehäuseeingang

Die nachfolgende Tabelle 25 gibt einen Überblick der äußeren Anschlussbelegung des Gehäuseeinganges.

PIN-Nummer	Belegung
2	I²C-SCL
4	I²C-SDA
5	Ebene_00
6	I²C-Reset
7	Ebene_01
8	Ebene_03
9	Ebene_02
10	Ebene_04
12	GND
13	Ebene_06
14	Ebene_05
16	Ebene_07

Tabelle 25 - PIN-Belegung für Außenanschluss des Gehäuseeinganges

Für eine Ansteuerung der Ebenen und der Weiterleitung der FPGA-Signale, werden auf der LED-Treiberplatine ein 6-PIN, ein 20-PIN und zwei 14-PIN -Pfostenbuchsen verbaut, welche nachfolgend näher erläutert werden.

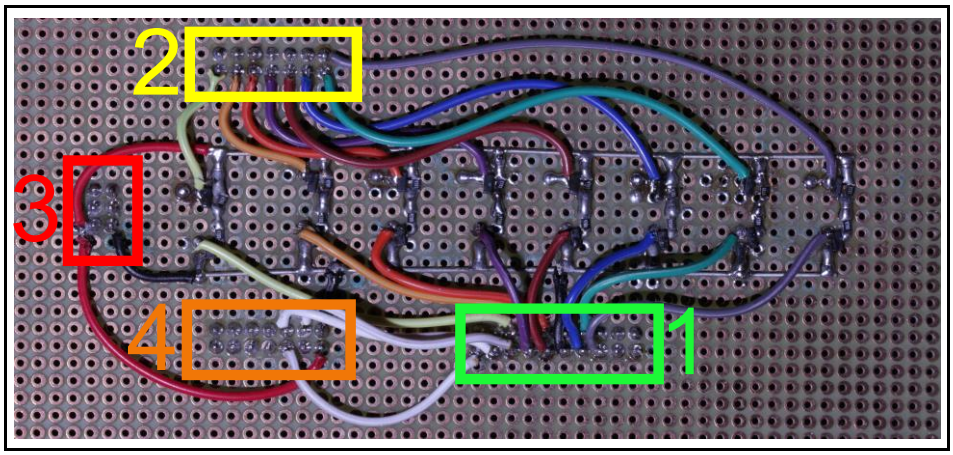


Abbildung 090 - Busverkabelung auf einer Treiberplatine

1: Vom Eingangsanschluss des Gehäuses erfolgt eine Verbindung zur Steuerplatine auf eine Eingangsbuchse.

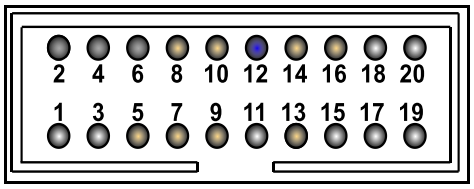


Abbildung 091 - PIN-Belegung der Eingangsbuchsen vom Gehäuse zur Steuerplatine

Die nachfolgende Tabelle 26 gibt einen Überblick über die Anschlussbelegung des Gehäuseeingangs.

PIN-Nummer	Belegung
2	I²C-SCL
4	I²C-SDA
5	Ebene_00
6	I²C-Reset
7	Ebene_01
8	Ebene_03
9	Ebene_02
10	Ebene_04
12	GND
13	Ebene_06
14	Ebene_05
16	Ebene_07

Tabelle 26 - PIN-Belegung der Eingangsbuchsen auf der Steuerungsplatine

Die Steuersignale der Ebene_00 bis Ebene_07 sind jeweils mit einer MOSFET-Treiberschaltung verbunden. Die I²C-Leitungen SCL, SDA und Reset werden über drei weiße Leitungen auf einem 14-PIN-Pfostenstecker durchgeschliffen.

2: Anschließend erfolgt die Verbindung der einzelnen Treiberschaltungen für die Ansteuerung der Ebenen des Würfels. Dazu wird von jedem einzelnen Drain eines p-Kanal-MOSFETs ein Kabel zu einer 14-PIN-Pfostenbuchse gelegt.

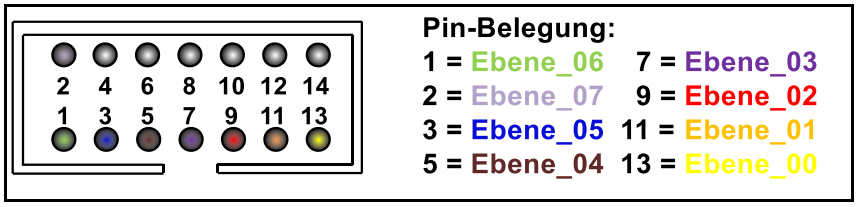


Abbildung 092 - PIN-Belegung für 14-PIN-Pfostenstecker für 8 Ebenen

3: Eine Stromversorgung der Elektronik erfolgt durch ein externes Netzteil, welches mit einem Wippschalter [56] verbunden ist. Die nachfolgende Abbildung 093 zeigt die Spannungsversorgung, den Schalter inklusive Schaltplan und dessen PIN-Belegung.

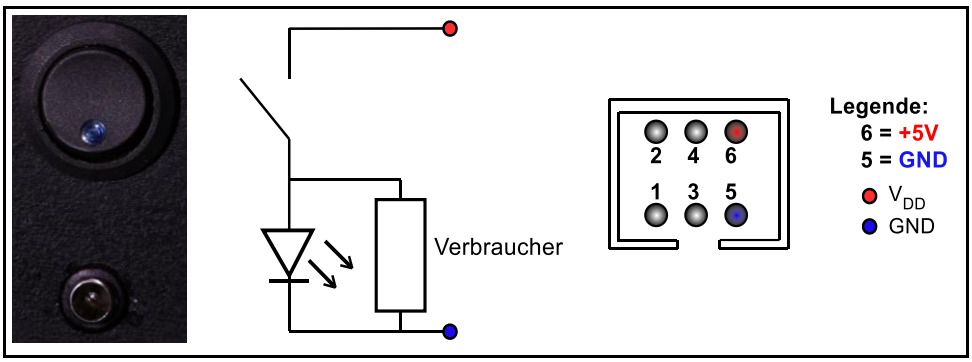


Abbildung 093 - PIN-Belegung Spannungsversorgungsanschluss mit Schalter inkl. Schaltplan

Die zwei Adern (+5V, GND) werden vom Schalter mit einem 6-PIN-Pfostenstecker verbunden. Diese dient der Versorgung der Treiberschaltung der einzelnen Ebenen und die I²C-Bausteine sind entsprechend verlötet.

4: Von der Stromversorgungsbuchse und der Eingangsbuchse werden die entsprechenden I²C-Signale und eine Spannungsversorgung auf eine 14-PIN-Buchse, entsprechend der Abbildung 094, verbunden.

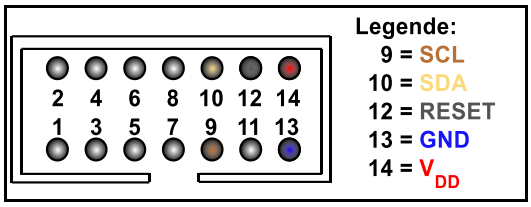


Abbildung 094 - PIN-Belegung für 14-PIN auf 6-PIN für einen PCA9698 am Pfostenstecker

Die Ansteuerung der I²C-Kanäle und die Spannungsversorgung der PCA9698 Expander wird durch ein Bussystem realisiert. Dazu wird auf der Lochrasterplatine ein 6-PIN-Pfostenstecker verbaut. Die PIN-Belegung ist in der folgenden Abbildung 095 zusammengefasst.

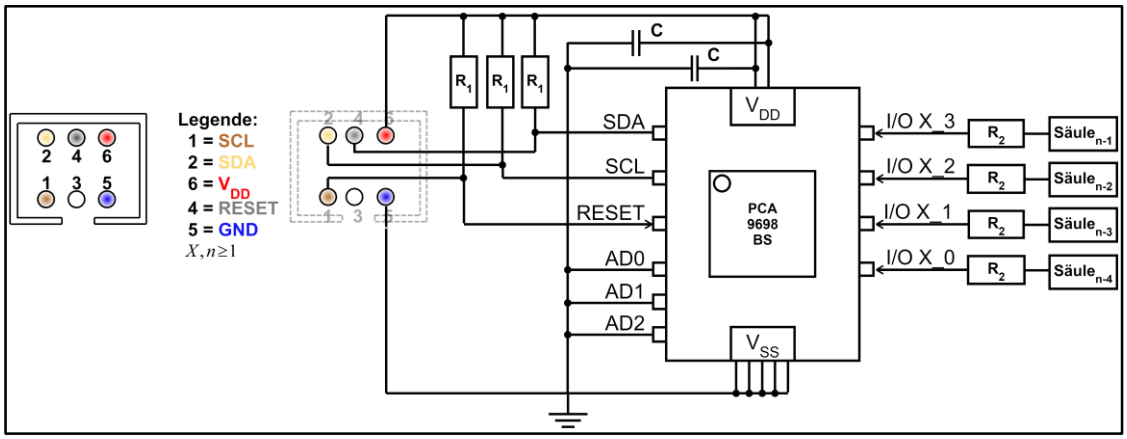


Abbildung 095 - PIN-Belegung PCA9698 für eine 6-PIN Busverbindung

Die Abbildung 096 verdeutlicht die Umsetzung der ICs auf einer Platine.

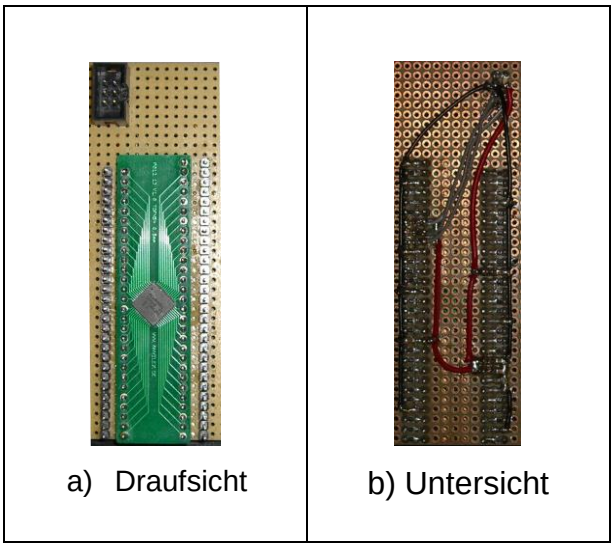
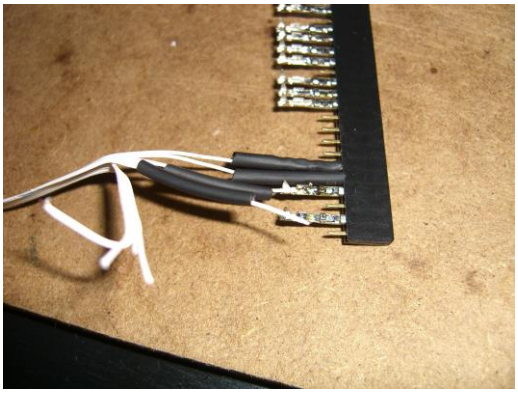


Abbildung 096 - PCA9698 auf einer Adapterplatine mit verlöteten Anschlüssen

Zum Verbinden der Säulen mit den I/O-Anschlüssen des GPIO-Expanders, werden MQS-Female Stecker angewandt, welche mit einem PIN-Adapter verbunden werden.



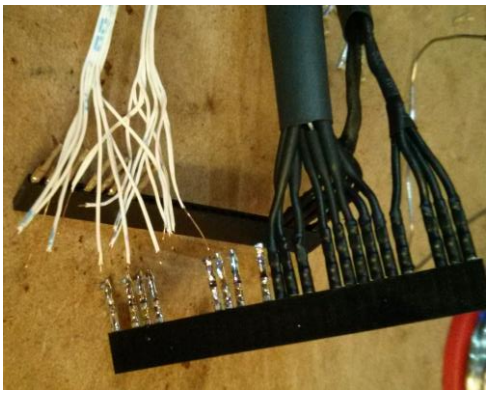


Abbildung 097 - Konstrukt einer Verbindung zwischen Säulensträngen mit einer Adapterleiste

Somit können die Kontakte leicht für Demonstrationszwecke entfernt und wieder verbunden werden. Die folgende Abbildung 098 zeigt das fertige System.

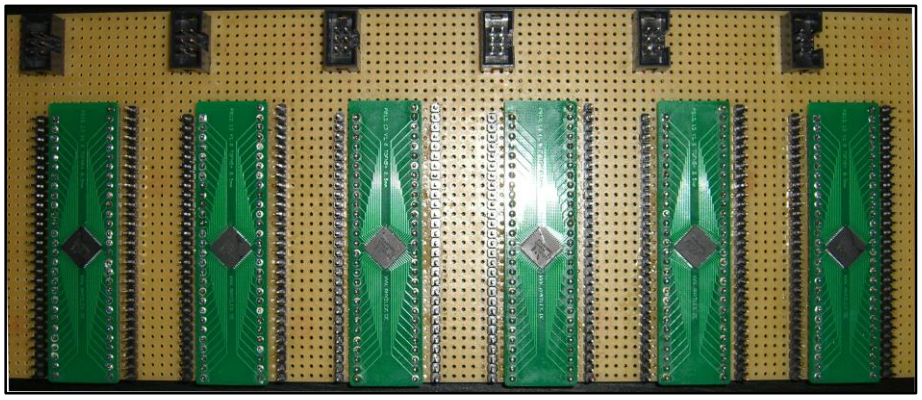


Abbildung 098 - Fertige Lochrasterplatine mit sechs PCA9698BS und Bus-Anschlüssen

Ein Beispielschaltplan für die Umsetzung der 3D-LED-Matrix befindet sich im Anhang auf DVD unter der Bezeichnung „Schaltplan_3D_LEDMatrix.pdf“.

5.4 Überprüfung der LED-Treiberschaltung mittels VHDL

Für eine theoretische Betrachtung erfolgt die Erstellung eines Konzepts. Es besteht aus Eingabe-, Verarbeitungs-, Ausgabe- und Hardwaremodul.

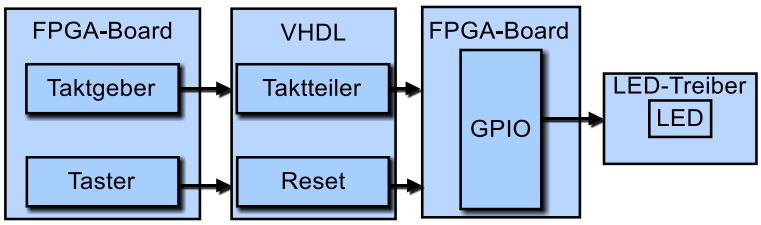


Abbildung 099 - Softwarekonzept zur Umsetzung der Ansteuerung des RGB-Cubes

Der Eingabeblock besteht aus einem Taktgeber und einer Reset-Logik. In der Verarbeitungseinheit teilt ein sogenannter Taktteiler einen 50Mhz -Systemtakt eines Entwicklungsboards auf einen geringeren Takt auf. Für einen Reset der VHDL-Logik, wird ein vom Board bereits entprellter Taster angewandt. Im letzten Abschnitt erfolgt die Ausgabe des niederfrequenten Taktes auf einem GPIO-Anschluss des Entwicklungsboards, an dem die Hardware angeschlossen ist.

Aus Kompatibilitätsgründen wird die kostenfreie EDA-Software Quartus II in der Version 13.1 und Altium-Designer 14.3 eingesetzt. Die folgende Abbildung 100 zeigt den kompletten Schaltungsentwurf eines „LED_Blinc“ -Projekts.

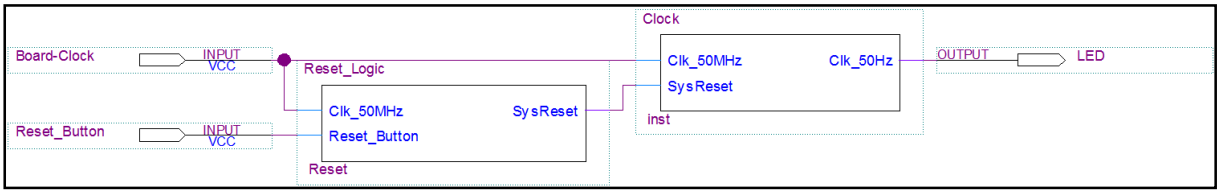


Abbildung 100 - Schaltungsentwurf des erzeugten VHDL-Taktteilers

Bei diesem wird eine LED am Entwicklungsboard betrieben. Dafür ist eine `Reset` -Logik und ein `Taktteiler` -Block (`Clock`) notwendig.

5.4.1 Umsetzung einer Reset-Logik mit Altera

Findet keine Reaktion auf eine Eingabe statt oder das System reagiert nicht ordnungsgemäß, ist für das Zurücksetzen der erzeugten Hard- und Software ein definierter Anfangszustand notwendig. Dies erreicht eine Reset-Routine, welche in jedem einzelnen Block definiert wird. Gemäß dem Xilinx-Datenblatt ist ein Reset-Logik-Modul empfehlenswert [50]. Die folgende Abbildung 101 zeigt ein Block-Symbol eines, in Altera erstellten, `Reset_Logic` -Moduls.

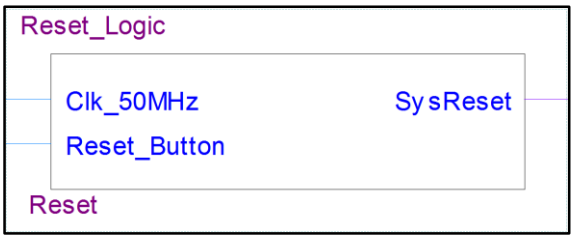


Abbildung 101 - Blockschaltbild einer Reset-Logik in Quartus II

Die Bedeutungen der einzelnen Ein- und Ausgabeanschlüsse und deren Funktionsbeschreibungen sind in folgender Tabelle 27 zusammengefasst.

Portname	Bild	Bitgröße	Funktion	Beschreibung
Clk_50Mhz		1	Eingang	50MHz Board-Takt
Reset_Button				Asynchroner LOW-aktiv-Pegel Reset
SysReset		1	Ausgang	0: Reset 1: kein Reset

Tabelle 27 - Übersicht der Ein- und Ausgabeanschlüsse der Reset-Logik

Für das Auslösen einer Reset-Routine, wird ein Taster angewandt. Die Funktionsbeschreibung erfolgt in einer „Reset_Logic.vhd“ -Datei.

```
01  library ieee;
02  use ieee.std_logic_1164.all;
03  use ieee.numeric_std.all;
```

Quellcode 01 - Import der IEEE-Bibliotheken in die Reset-Logik

Zunächst werden die benötigten IEEE-Bibliotheken importiert (01-03). Der Reset wird nicht auf eine Flanke des Systemtaktes ausgeführt (asynchron) und besitzt einen LOW-aktiv-Pegel.

```
01  …
02  Out_SysReset <= '0' when System_Reset = '0' else '1';
03  …
```

Quellcode 02 - Setzen des Ausgangsports der Reset-Logik

Dies bedeutet, dass am Ausgang erst bei einem Taster-Druck ein LOW-Pegel ausgegeben wird. Ist dieser nicht gedrückt, erfolgt stets die Ausgabe eines HIGH-Pegels.

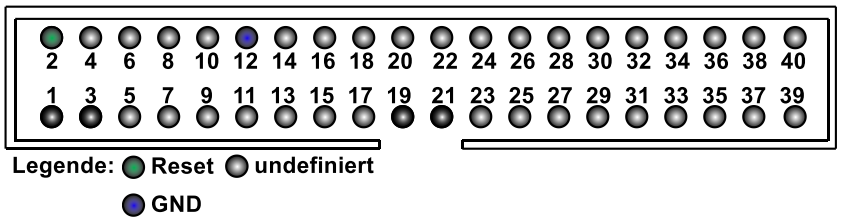


Abbildung 102 - PIN-Belegung eines Resets am GPIO_1 des Altera DE0 NanoBoard

Die Hardwareausgabe der Logik erfolgt am zweiten PIN des GPIO_1 des Altera DE0 NanoBoards. Ausgelöst wird das Signal über den „Key1“ des Boards.

Abbildung 103 - Reset-Button Altera DE0 NanoBoard

Die folgende Oszilloskop-Ausgabe verdeutlicht die Funktionalität der Logik an einer Hardwareausgabe.

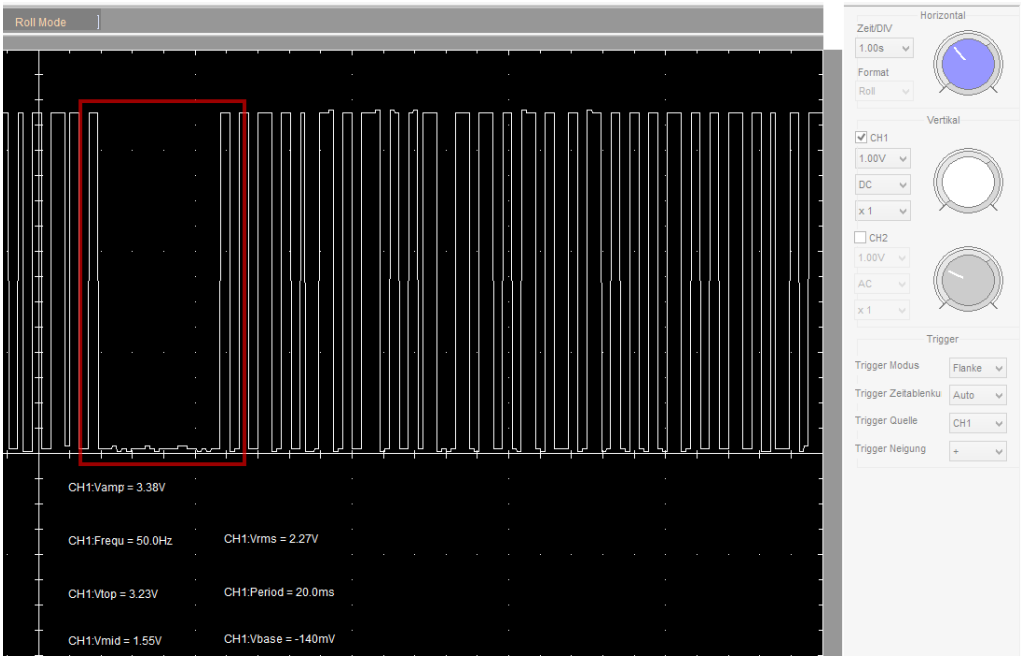


Abbildung 104 - Verifizierung der Reset-Logik am Oszilloskop

In der Abbildung 104 wird der Reset-Taster für rund $1s$ betätigt. Ersichtlich ist, dass ein $3{,}3V$ HIGH-Pegel in einem Takt von $50Hz$ ausgegeben wird. Erst beim Betätigen des Tasters, wird ein LOW-Pegel übertragen. Dies geschieht solange bis der Taster losgelassen wird.

5.4.2 Umsetzung eines Taktteilers mit Altera

Das Altera und Altium Entwicklungsboard verfügt jeweils über einen 50MHz-Taktgeber. Die Teilung des Taktes erfolgt durch die Zählung von steigenden Flanken des 50MHz-Systemtaktes. Für die Berechnung wird folgende Formel 18 angewandt:

Formel 18 - VHDL-Taktteiler

$$\frac{Zeit\ [Sekunden]}{FPGA-Takt\ [Z\ddot{a}hlerschritte]} = \frac{x\ [Sekunden]}{y\ [Z\ddot{a}hlerschritte]}$$

Diese Formel entspricht einem Dreisatz. Für den Wert „x in Sekunden“ wird eine gewünschte Periodendauer eingetragen. Anschließend wird diese nach y aufgelöst.

$$y = \frac{FPGA - Takt\ [Zählerschritte] \cdot x\ [Sekunden]}{Zeit\ [Sekunden]}$$

Daraus folgt für einen 50MHz -Taktgeber folgende Formel:

$$y = \frac{5 \cdot 10^7\ [Zählerschritte] \cdot x\ [Sekunden]}{Zeit\ [Sekunden]}$$

In diesem Beispiel wird der Systemtakt von 50Mhz auf 50Hz (entspricht einer Periodendauer von 0,02s) heruntergeteilt.

$$\frac{1\ [Sekunde]}{5 \cdot 10^7\ [Zählerschritte]} = \frac{0{,}02\ [Sekunden]}{y\ [Zählerschritte]}$$

$$y = \left[\frac{(5 \cdot 10^7\ Schritte) \cdot 0{,}02s}{1s}\right] = 1.000.000\ Schritte$$

$$\text{y} = \frac{1.000.000}{2} = 500.000\ Schritte$$

Die Berechnung verdeutlicht, dass für einen 50Hz-Takt 500.000 Flankenwechsel des 50MHz-Taktes abgezählt werden müssen.

Die folgende Abbildung 105 zeigt ein Block-Symbol einer in Altera erstellten Taktteiler-Logik.

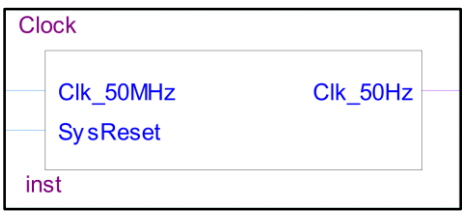


Abbildung 105 - Blockschaltbild eines 50Hz-Taktteilers

Für eine Umsetzung eines Taktteilers in VHDL werden Programmelemente benötigt, welche in der folgenden Tabelle 28 zusammengefasst sind.

Portname	Symbol	Bitgröße	Funktion	Beschreibung
Clk_50Mhz		1	Eingang	50MHz Board-Takt
SysReset				Asynchroner LOW-aktiv-Pegel Reset
Clk_50Hz		1	Ausgang	Takt: 50Hz

Tabelle 28 - Auflistung der Programmelemente des 50Hz-Taktteilers im Quartus II

Für die Erstellung eines Taktteilers wird eine VHDL-Datei mit den Namen „ClkDivider.vhd“ erzeugt.

```
01  library ieee;
02  use ieee.std_logic_1164.all;
03  use ieee.numeric_std.all;
```

Quellcode 03 - Import der IEEE-Bibliotheken im 50Hz-Taktteiler

Für die Umsetzung werden zuerst die benötigten Bibliotheken importiert (01-03). Es folgt die Initialisierung der benötigten In- und Outputs in der sogenannten *entity*.

```
01  …
02  entity Clock is
03  port
04  (
05    Clk_50MHz:in  std_logic;
06    Reset    :in  std_logic;
07    Clk_50Hz :out std_logic
08  );
09  …
```

Quellcode 04 - Definition der Ein- und Ausgänge des 50Hz-Taktteilers

Innerhalb dieser werden die Ports definiert, welche als Ein- und Ausgänge fungieren(05-07).

Nach der e*ntity* wird die Architektur des Taktteilers angelegt. In dieser befinden sich Variablendeklarationen und die in VHDL-Code umgesetzte Funktionalität des zu erzeugenden Bausteins.

```
01  …
02  architecture Main of Clock is
03  begin
04    Clk:
05    process (Clk_50MHz, Reset)
06      constant FPGA_Freq : integer := 50000000;
07      constant Clock_50Hz: integer := 50;
08      variable counter   : integer range 0 to FPGA_freq;
09    begin
10      -- async Reset
11      if (Reset = '0') then
12        Clk_50Hz<= '0';
13        counter := 0;
14      -- sync Rising Edge
15      elsif (rising_edge(Clk_50MHz)) then
16        -- First Periode Logic LOW
17        if (counter < ((FPGA_Freq/Clock_50Hz)/2)) then
18          Clk_50Hz<= '0';
19          counter := counter + 1;
20        -- Second Periode Logic HIGH
21        elsif (counter < ((FPGA_Freq/Clock_50Hz)-1)) then
22          Clk_50Hz<= '1';
23          counter := counter + 1;
24        else
25          counter := 0;
26        end if;
27      end if;
28    end process Clk;
29  end Main;
30  …
```

Quellcode 05 - Realisierung des Taktteilers von 50MHz auf 50Hz mit VHDL

Es werden zwei Konstanten und eine Variable von Typ Integer benötigt (06-08). Bevor eine Funktionsbeschreibung des Blocks stattfindet, wird eine asynchrone Reset-Routine initialisiert (11-13). Beim Auslösen wird die Variable `counter` zurückgesetzt. Für eine Teilung auf 50Hz müssen jeweils 500.000 Flanken gezählt werden, bevor am Ausgang des Taktteilers eine Pegeländerung stattfindet (15-25).

Für eine Funktionsüberprüfung wird ein Projekt mit dem Namen „LED-Blinc“ angelegt. In diesem wird dieser Block mit einer Reset-Logik und Ein- und Ausgängen verbunden.

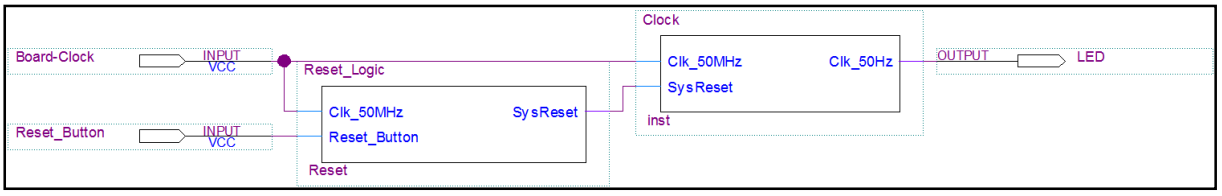


Abbildung 106 - Schematische Darstellung eines erzeugten 50Hz-Taktteilers im Quartus II

Bevor die erzeugte Hardwarebeschreibung praktisch ausgeführt wird, erfolgt eine Simulation des Funktionsprinzips. Eine Echtzeitsimulation ist aufgrund der zeitlich begrenzten Darstellung von *ModelSim* nicht möglich, da nur eine maximale Simulationszeit von 10.000ns unterstützt wird. Damit der erzeugte Taktteiler qualitativ simuliert werden kann, wird die Taktrate gegenüber der tatsächlich zu erzeugenden erhöht, um die prinzipielle Funktionsweise zu verifizieren.

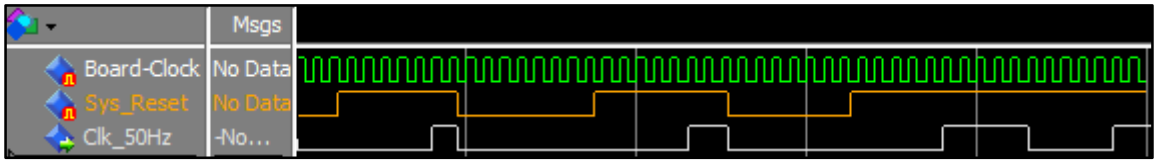


Abbildung 107 - Simulation eines 50Hz-Taktteilers im ModelSim

Zu erkennen ist, dass der 50MHz -Board-Takt zu einem niederfrequenten-Ausgangstakt gerechnet wird. Erfolgt eine Reset-Routine wird ein LOW-Pegel am Ausgang ausgegeben. Aufgrund der begrenzten Simulationsmöglichkeiten wurde eine Taktfrequenz von 5MHz gewählt. Ein qualitativer Nachweis des Ausgangstaktes von 50Hz erfolgt bei der praktischen Ausführung.

Für eine praktische Funktionsprüfung dient der zweite GPIO-Anschluss (GPIO_1) des Altera Cyclone 4 DE0 NanoBoard.

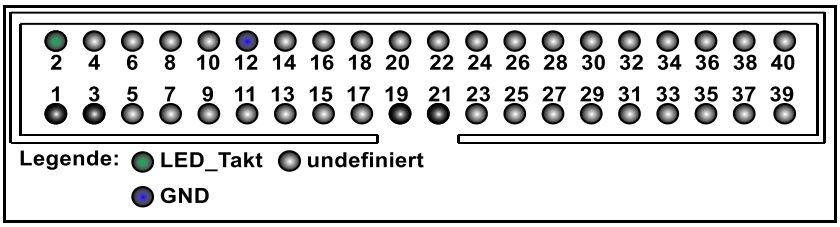


Abbildung 108 - PIN-Belegung des 50Hz-Taktteilers auf einem DE0 NanoBoard

Desweiteren werden Ein- und Ausgänge den entsprechenden FPGA-PINs zugeordnet. An diesem wird eine LED angeschlossen, welche mit einer Taktrate von 50Hz leuchtet. Dafür ist eine LED auf einer Lochrasterplatine aufgebracht und mit einem Sechs-PIN-Pfostenstecker verbunden.

Der GPIO_1-PIN-2 wird mit der Anode (Pluspol) der LED verbunden, während der GPIO_1-PIN-12 an den GND angeschlossen ist. Die folgende Tabelle 29 gibt einen Überblick der GPIO-Belegung.

PIN-Nummer	Altera-Bezeichnung	Belegung
2	PIN_F13	LED-Takt
12		GND
	PIN_R8	Board-Clock
	PIN_E1	Reset-Button

Tabelle 29 - Altera DE0 NanoBoard-GPIO_1 Belegung für einen Taktteiler

Anschließend wird die PIN-Belegung in den „Altera-PIN-Planner“ übertragen.

out LED	Output	PIN_F13	6	B6_N0	PIN_F13
in Reset-Button	Input	PIN_E1	1	B1_N0	PIN_E1
in Sys-Clock	Input	PIN_R8	3	B3_N0	PIN_R8

Abbildung 109 - Altera PIN-Planer des 50Hz Taktteilers auf einem DE0 NanoBoard

Nach erfolgreicher Synthetisierung erfolgt eine Verifikation des Taktteilers an einer LED.

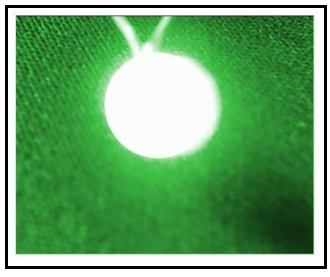


Abbildung 110 - Verifikation des 50Hz-Taktteilers an einer LED

Die Ausgabe zeigt, dass dieses Projekt funktionstüchtig ist und die LED konstant mit $50\mathrm{Hz}$ leuchtet. Eine graphische Ausgabe eines Oszilloskops verdeutlicht ebenfalls die Funktionstüchtigkeit der erzeugten VHDL-Blocks.

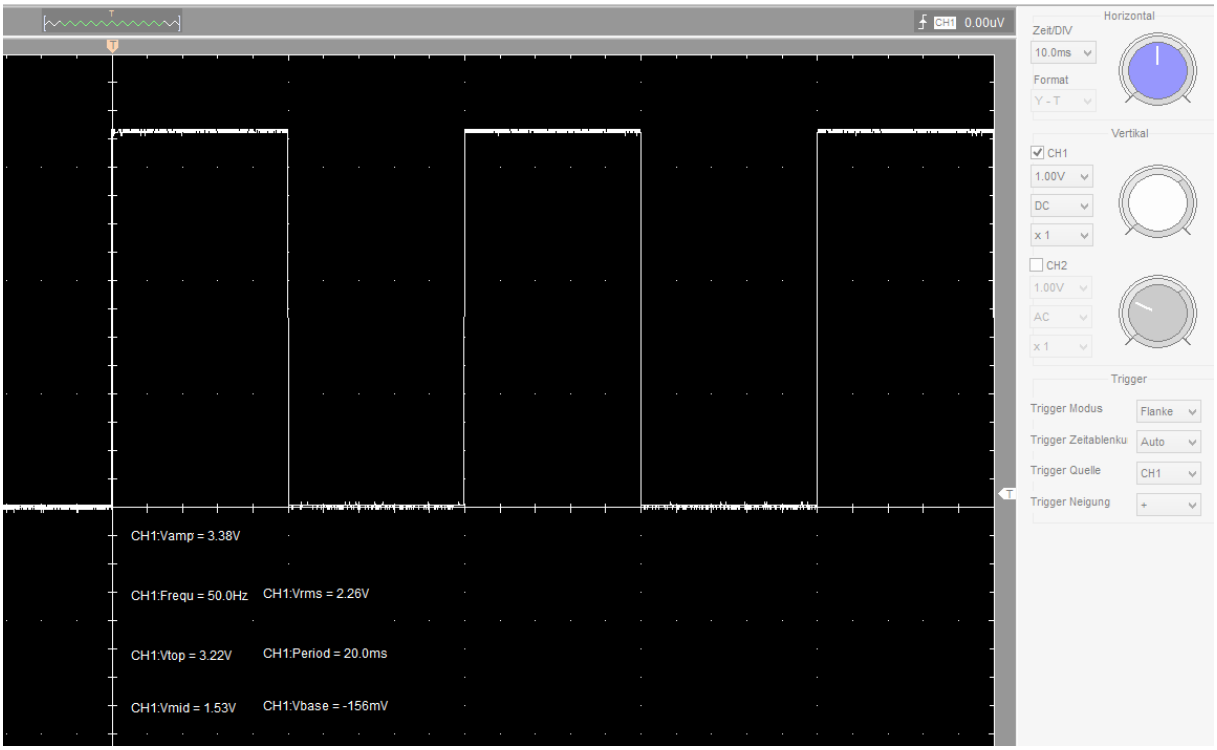


Abbildung 111 - Verifikation des 50Hz-Taktteilers am Oszilloskop

Die Ausgabe zeigt einen Spannungsverlauf von $3{,}3\text{V}$ in einem 50Hz-Takt.

5.4.3 Veränderung des erzeugten Taktteilers

Für Demonstrationszwecke wird ein zu erzeugender Takt von 1Hz gewählt.

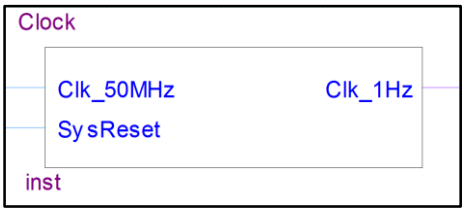


Abbildung 112 - Blockschaltbild eines 1Hz-Taktteilers im Quartus II

Dies wird erreicht indem die Konstante `Clock_50Hz` in einen Integer-Wert von `1` umgeschrieben wird.

```
01  …
02  constant Clock_1Hz : integer := 1;
03  …
```

Quellcode 06 - Änderung der Ausgabefrequenz auf 1Hz im 50Hz-Taktteiler

Aufgrund der Übersichtlichkeit wird ebenfalls der Ausgangs-Portname zu `Clock_1Hz` umbenannt (02).

Diese Umbenennung ist nicht zwingend erforderlich. Sie dient lediglich der Übersichtlichkeit und einer Überprüfung bei einem Block-Update.

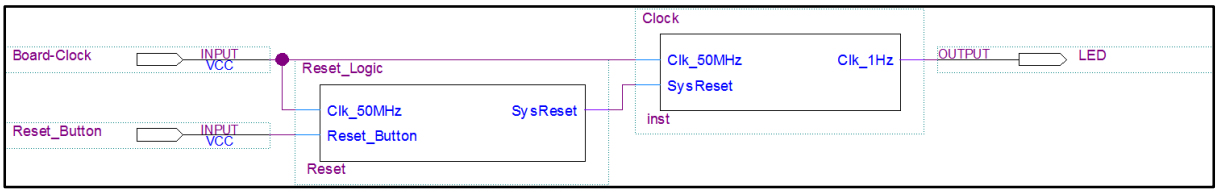


Abbildung 113 - Schematische Darstellung eines 1Hz-Taktteilers im Quartus II

Die Simulation des 50Hz-Taktteilers zeigte, dass die Funktionstüchtigkeit der beschriebenen Hardware gegeben ist. Aus diesem Grund wird auf eine erneute Simulation verzichtet.

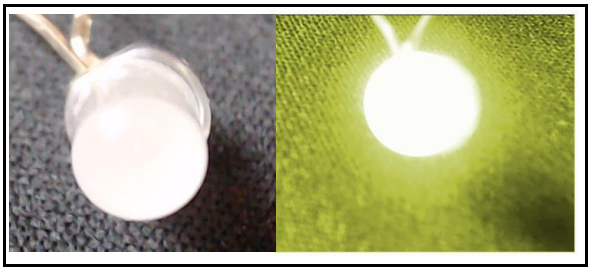


Abbildung 114 - Verifikation eines 1Hz-Taktteilers in Quartus II an einer LED

Die Abbildung 114 zeigt eine statische Hardwareausgabe der Funktionsfähigkeit dieser Hardwarebeschreibung. Ein Demonstrationsvideo ist im Anhang auf DVD unter der Bezeichnung „LED-Blink-Test.m4v“ vorhanden. Dieses zeigt eine blinkende LED in einem Zyklus von einem Hertz.

5.4.4 Umsetzung mit dem Altium-Designer auf dem NanoBoard 3000

Das Altium NB3K verfügt über einen $20MHz$ und einem $50MHz$ -Taktgeber. Aus Kompatibilitätsgründen wird der $50MHz$ -Taktgeber angewandt. Die folgende Tabelle 30 zeigt eine Auflistung der benötigten Elemente, die dazugehörigen Altium-Bibliotheken und deren Portbezeichnung.

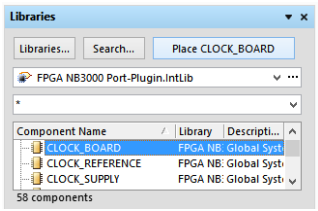
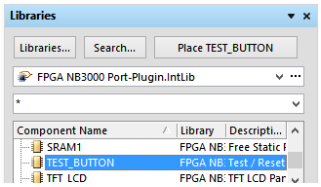
Element	Altium Bibliothek		Portname
		FPGA NB3000 Port-Plugin.IntLib	CLOCK-BOARD
			TEST-BUTTON
		Altium-Designer Toolbar	Place Port
			Virtuelle Single-Verbindungen inkl. Net Label
			Virtuelle BUS-Verbindungen inkl. Net Label

Tabelle 30 - Auflistung benötigter Programmelemente eines 1Hz-Taktteilers im Altium-Designer

Die bereits aus den Kapiteln 5.4.1 und 5.4.2 erstellten „Reset-Logic.vhd“ und „Clk_Divider.vhd“ -Dateien werden in ein neues Altium-FPGA-Projekt portiert und anschließend kompiliert.

Für eine schematische Darstellung wird in diesem Projekt ein „Schematic-Sheet“ erzeugt.

Auf diesem wird das erzeugte Symbol-Sheet (Block-Symbol-File) dargestellt. Ein Takt von 1Hz soll auf dem User Header A Port 2 und GND auf dem Port 4 erfolgen.

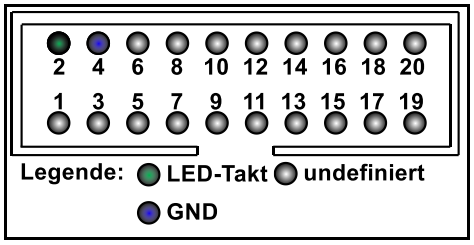


Abbildung 115 - PIN-Belegung eines 1Hz-Taktteilers am Altium NB3K

Angesteuert werden die GPIO-Header A und B über die Altium-Bezeichnung „HAxx“ und „HBxx“. Die entsprechenden Ports werden mit der Header-Nummer angegeben. Die folgende Abbildung 116 zeigt einen Konfigurationsdialog eines Ports.

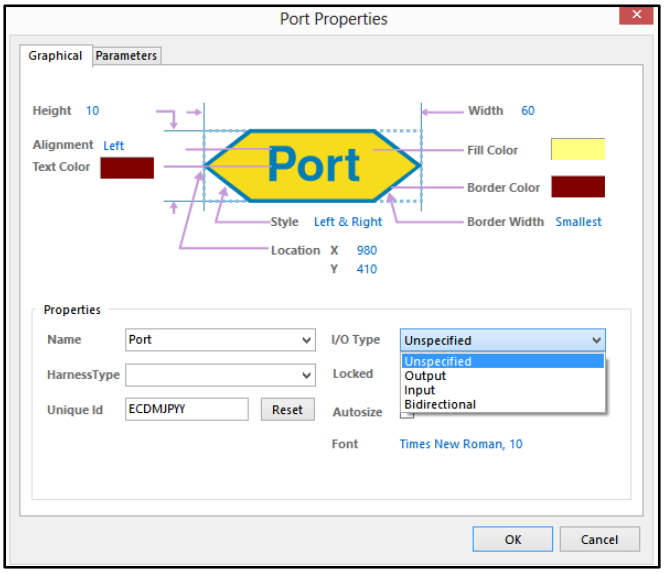


Abbildung 116 - Einstellmöglichkeiten eines Ports im Altium-Designer

Unter der Auswahl „Name“ wird die GPIO-Bezeichnung und dessen Port eingetragen. Mit dem Reiter „I/O Type“ kann der PIN entsprechend seines Typs als Eingang (Input), Ausgang (Output) oder als bidirektionale[9] (bidirectional) Anschluss konfiguriert werden. Für dieses Teilprojekt werden zwei Ausgangsanschlüsse benötigt.

[9] Anschluss fungiert als Ein- und Ausgang

Die folgende Abbildung 117 zeigt eine fertige grafische Darstellung des erzeugten Sheets sowie Ein- und Ausgabe-Anschlüsse.

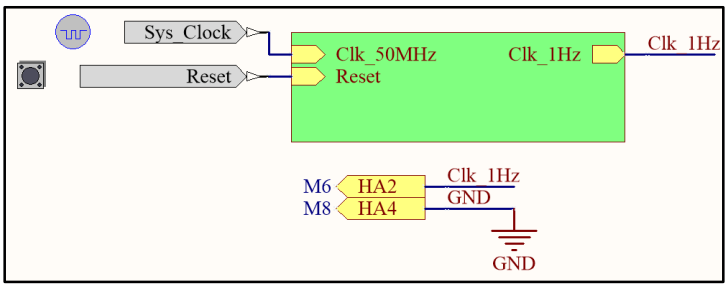


Abbildung 117 - Schematische Darstellung eines 1Hz-Taktteilers im Altium-Designer

Die Ein- und Ausgangsports werden mittels Leitungen verbunden. Dies kann über eine direkte Verbindung und/oder über ein sogenanntes „NetLabel“ umgesetzt werden. Beim NetLabel erhalten die Leitungen sowie Ports eine Beschriftung und sind virtuell verbunden. Anschließend wird das Projekt ausgewählt und das Programm der Reihe nach kompiliert, synthetisiert und für den FPGA übersetzt.

Ein qualitativer Nachweis des $1\mathrm{Hz}$ -Taktteilers erfolgt mittels FPGA an einer LED.

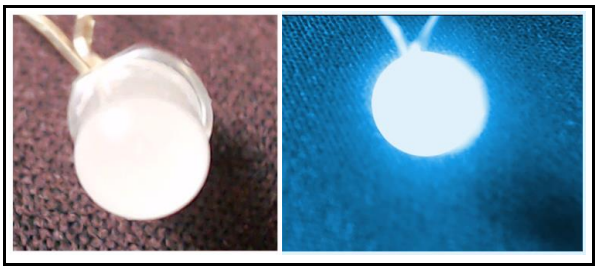


Abbildung 118 - Verifikation eines 1Hz-Taktteilers am NB3K an einer LED

Ein Reset der Anwendung wird mittels Test/Reset-Taster auf dem NB3K ausgelöst.

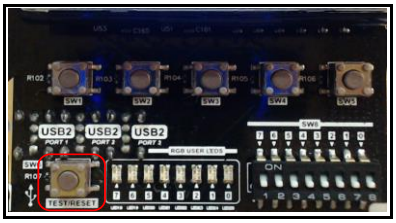


Abbildung 119 - Test/Reset-Taster des NB3K

5.4.5 Überprüfung eines n-Kanal-MOSFETs mittels eines Taktteilers

Mit Hilfe des erzeugten Taktteilers wird eine Funktionsprüfung auf einem Altera DE0 NanoBoard des für die Treiberschaltung angewandten n-Kanal-MOSFETs „Fairchild 2N7002“ erfolgen. Der folgende Schaltplan gibt einen Überblick über den Aufbau des Teilprojektes.

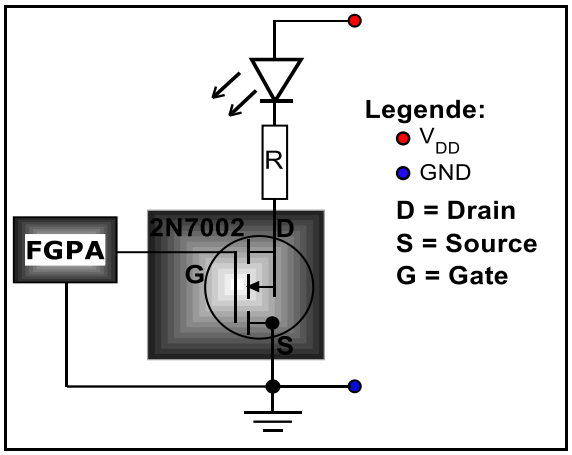


Abbildung 120 - Schaltplan eines n-Kanal-MOSFETs am FPGA

Der Source des n-Kanal-MOSFETs wird mit GND (PIN-12) verbunden, während das Gate an den Takt von 1Hz (PIN-2) angeschlossen wird.

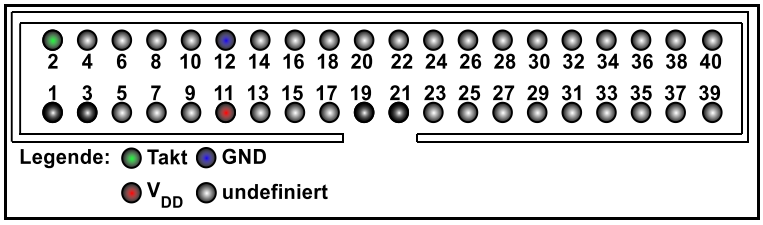


Abbildung 121 - PIN-Belegung eines n-Kanal-MOSFETS am DE0 NanoBoard

Zwischen einer Spannungsquelle (PIN-11) und dem Drain wird eine LED mit einem 100Ω Vorwiderstand angeschlossen.

Aufgrund der geringen Entwicklungsboard-Spannungsversorgung von $3{,}3\text{V}$, kann der LED-Vorwiderstand ausgelassen werden. Anschließend wird das Gate mit dem 1Hz -Takt geschaltet.

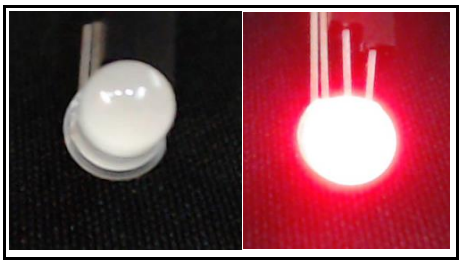


Abbildung 122 - Verifikation einer n-Kanal-MOSFET Schaltung am FPGA

Die Abbildung 122 verdeutlicht die Verifizierung der Hardware.

5.4.6 Überprüfung eines p-Kanal-MOSFET mittels eines Taktteilers

Ebenfalls wird eine Funktionsprüfung des angewandten „Si2323ds“ p-Kanal-MOSFET durchgeführt. Der folgende Schaltplan gibt einen Überblick über den Aufbau der p-Kanal-MOSFET-Schaltung mit einem FPGA.

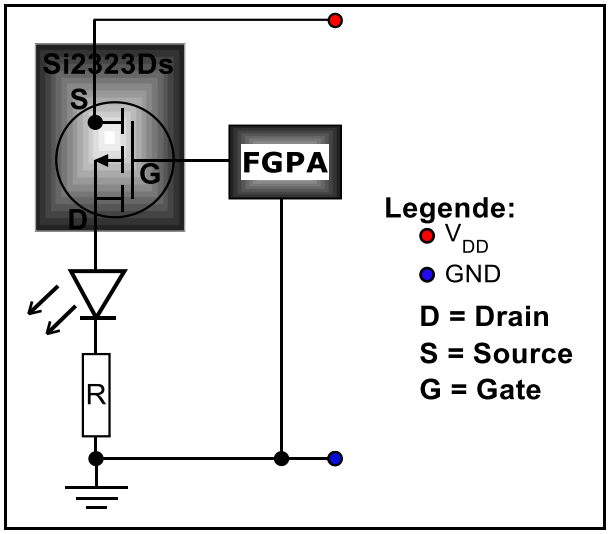


Abbildung 123 - Schaltplan eines p-Kanal-MOSFETs am DE0 NanoBoard

Anschließend erfolgt die Verifizierung an der Hardware.

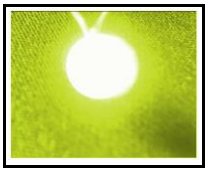


Abbildung 124 - Verifikation einer p-Kanal-MOSFET Schaltung am FPGA

Die Hardwareverifikation verdeutlicht, dass eine Ansteuerung des p-Kanal-MOSFETs aufgrund seiner Eigenschaften und der geringen FPGA-Spannung nicht möglich ist. Am Drain wird eine $+5V$ Spannung ausgegeben und die LED leuchtet dauerhaft.

5.4.7 Überprüfung der LED-Treiberschaltung

Eine Funktionsbeschreibung der Treiberschaltung wurde bereits im Kapitel 5.3.3 erläutert. Nachdem eine erfolgreiche Verifikation eines Taktteilers erfolgt ist, wird die LED-Treiberschaltung auf Funktionalität überprüft. Für eine Spannungsversorgung dient ein $5V$ Netzteil mit einer maximal möglichen Stromstärke von $3A$. Der folgende Schaltplan verdeutlicht den Aufbau.

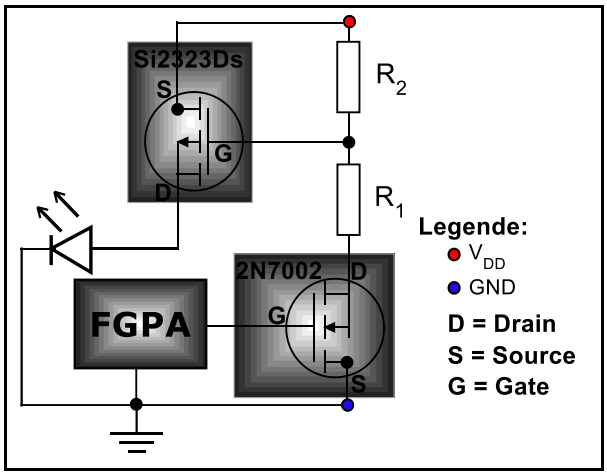


Abbildung 125 - Schaltplan zur Überprüfung der LED-Treiberschaltung an einer LED

Anzumerken ist, dass der V_{DD}-Anschluss des Netzteils mit dem Source-Anschluss des p-Kanal-MOSFETs verbunden wird. Die Anode der LED wird mit dem Drain des p-Kanal-MOSFETs verlötet und die Kathode auf GND gelegt.

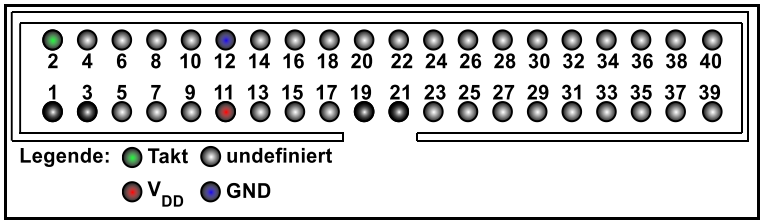


Abbildung 126 - PIN-Belegung der LED-Treiberschaltung am DE0 NanoBoard

Ebenfalls auf den gleichen GND wird der Source des n-Kanal-MOSFETs und der PIN-12 des GPIO-Ports des Altera Entwicklungsboards gelegt.

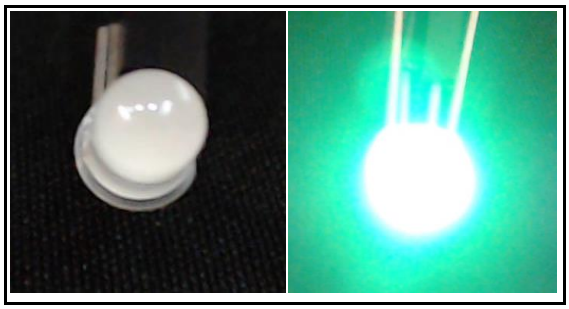


Abbildung 127 - Verifikation der LED-Treiberschaltung am FPGA

Die Ausgabe zeigt, dass die Schaltung ebenfalls funktionstüchtig ist.

5.5 Vorstellung eines I²C-Master-Core mit VHDL

Für die Anwendung eines I²C-Master-Core bieten die Hersteller Altera, Xilinx und Altium jeweils meist eigene proprietäre Lösungen an. Meist beziehen sich diese Angaben auf einen fertigen sogenannten „OpenCore“[10] [57].

Eine oberflächliche Analyse des Cores ergab, dass dieser keine reine VHDL-Entwicklung ist und andere Hardwarebeschreibungs- und Entwicklungssprachen in diesem Paket existieren. So befinden sich beispielsweise Dateien, welche mit der Programmiersprache C entwickelt sind, in diesem Paket. Desweiteren ist der I²C-Core nicht umfangreich dokumentiert, so dass ein schneller Einstieg nicht gegeben ist.

Ein Teilziel des Projektes ist es, einen 2008 standardisierten VHDL Code zu erzeugen und anzuwenden, welcher plattformübergreifend synthetisierbar ist. So bietet etwa das Dashboard „eewiki“, welches Teil der „Digi-Key Corporation“[11] ist, einen kostenlosen fertigen I²C-Master-Core in der Hardwarebeschreibungssprache VHDL an [58 - 61]. Dies ist eine Entwicklung von Scott Larson welcher den Core frei zur Verfügung stellt.

[10] ist eine Open-Source Hardwarevereinigung

[11] ist ein Verteiler für elektronische Bauteile, Baugruppen und Evaluation-Boards

```
--   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
```

Quellcode 07 - Code-Kommentar von Scott Larson zur Verwendung des I²C-Master-Core [61]

Diese Textpassage beschreibt, dass der Autor sowie die Plattform den Quelltext frei zur Verfügung stellen und sich von jeglicher Verantwortung distanzieren. Der Core ist in VHDL mit der EDA-Software Quartus II Version 11.1 entwickelt. Dies ermöglicht den Code herstellerunabhängig anzuwenden.

Angewandt wird der Master-Core in der Version 1.0 [61]. Aktuelle Versionen liegen in der Version 2.0 und 2.1 vor [60]. Änderungen des Cores sind bis Version 2.0 in dieser Dokumentation berücksichtigt und eingepflegt worden.

Anzumerken ist, dass der I²C-Master-Core von Scott Larson eine einfache Handhabung mit sich bringt. Es existiert jedoch keine umfangreiche Anwendungsdokumentation und der Core entspricht nicht dem VHDL Standard 2008.

Einige dieser Nachteile werden im Laufe des Kapitels näher erläutert.

Die folgende Abbildung 128 beschreibt einen theoretischen Ablaufplan der State Maschine des Entwicklers.

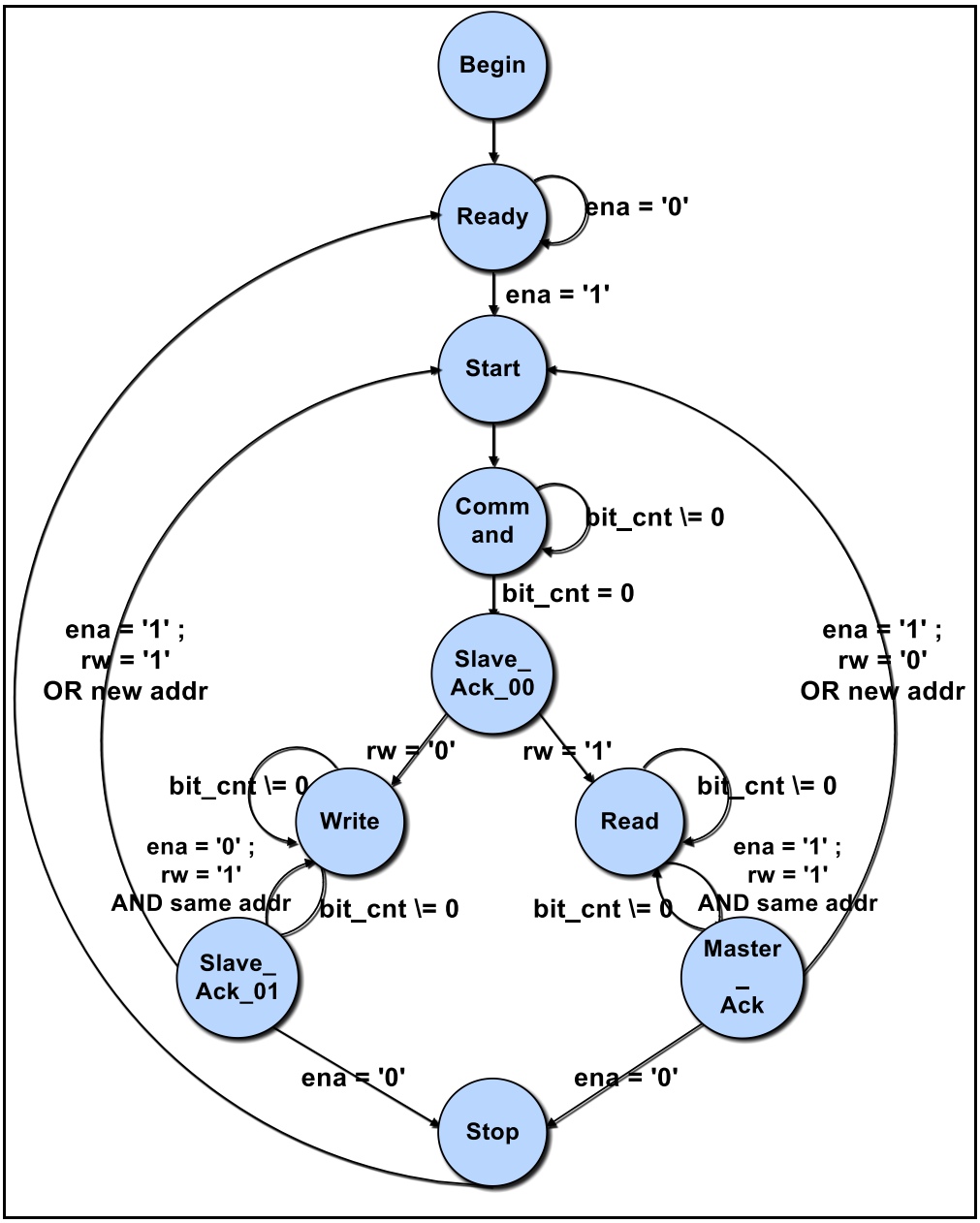


Abbildung 128 - Zustandsbeschreibung I²C-Master-Core Scott Larson bearbeitet unter Verwendung von [60]

Die einzelnen Ports und deren Funktionsbeschreibungen sind in folgender Tabelle 31 zusammengefasst.

Portname	Bitgröße	Funktion	Beschreibung
addr	7	Eingang	Slave Adresse (7-Bit)
clk	1	Eingang	Systemtakt
data_wr	8	Eingang	Zu übertragende Daten inkl. rw -Bit
ena	1	Eingang	0: keine Übertragung 1: Übertragung wird fortgeführt
reset_n	1	Eingang	LOW-aktiv-Pegel Reset
rw	1	Eingang	0: Daten schreiben 1: Daten lesen
data_rd	8	Ausgang	Gelesene Daten (8-Bit)
busy	1	Ausgang	0: I2C-Master im Leerlauf 1: Transaktion in Arbeit
ack_error	1	Buffer	0: keine Fehler 1: mindestens ein Fehler aufgetreten. Bit wird bei Beginn einer neuen Transaktion gelöscht
scl	1	Ein- und Ausgang	Serielle I²C-Systemtakt
sda	1	Ein- und Ausgang	I²C-Datenbus

Tabelle 31 - Ports und ihre Funktionen des I²C-Cores von Scott Larson [60]

Diese gibt eine Übersicht der angewandten PORTs des Master-Cores von Scott Larson. Der Core dient lediglich als Schnittstelle zwischen einer Anwendungslogik und der ICs. Anzumerken ist, dass die Variable `ack_error` ein `buffer` ist. Dieses ist nicht VHDL 2008 konform und kann bei einer Synthese nicht umgesetzt werden. Bei einer praktischen Anwendung mit einem Schreibzyklus auf einem I²C-Baustein mit einer maximalen 16-Bit Datenübertragung, funktioniert dieser zunächst wie erwartet gemäß der I²C-Spezifikation. Bei einem kontinuierlichen wiederholenden Schreibzyklus auf einen oder mehreren I²C-ICs oder bei einer Übertragung von mehr als 16-Bits (Daten), erfolgt keine Ausgabe.

Diese ist unter anderem auf das `busy` -Signal, die Anwendungslogik und die Zeitabläufe zurückzuführen. Die Flanken des `busy` -Signals signalisieren der Anwenderlogik, ob der Core bereit ist, neue Daten zu empfangen oder sich in einer Abarbeitung befindet. Durch Ver- und Bearbeiten des Signals kann es zu Zeitverzögerungen kommen. Einige dieser fehlerhaften Flanken sind aus den Simulationsergebnissen der einzeln erzeugten Module nicht ersichtlich. Aus diesen Gründen erfolgt eine Modifikation des Cores. Der Quelltext und die Zustandsbeschreibungen des veränderten Masters werden in Verbindung mit dem Original im nachfolgenden Kapitel näher erläutert. In Rahmen dieser Dokumentation wird nur der Schreibzyklus auf einen I²C-Baustein betrachtet. Der I²C-Core von Herrn Larson ist unter der Referenz [61] aufgelistet.

5.6 Konzept Ansteuerung RGB-LED-Würfel mittels VHDL

Für eine theoretische Betrachtung, für einen Betrieb des RGB-LED-Würfels, erfolgt die Erstellung eines Konzepts. Es besteht aus Eingabe-, Verarbeitungs-, Ausgabe- und Hardwaremodulen.

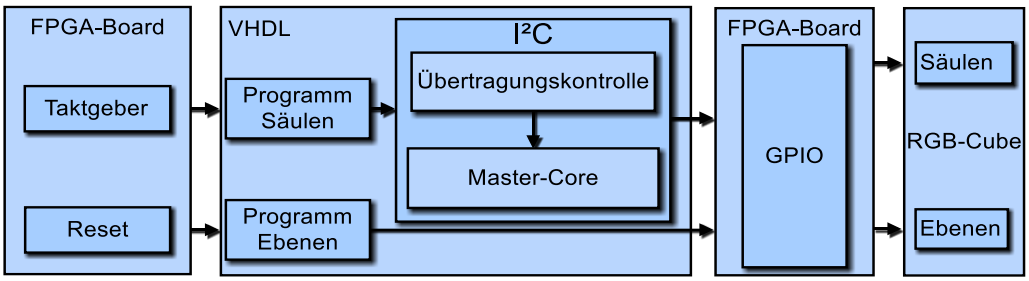


Abbildung 129 - Konzept der RGB-Cube Ansteuerung von einem FPGA

Der Eingabeblock besteht aus einem Taktgeber und einer Reset-Logik. Der Taktgeber teilt einen 50Mhz -Systemtakt eines Entwicklungsboards auf einen geringeren I²C-Takt auf. In der Verarbeitungseinheit wird eine entsprechende Hardwarelogik für eine Animation auf das Ausgangsprodukt deklariert. Im dritten Abschnitt werden die entsprechenden Signale über Leitungen an die Hardware des Würfels weitergegeben. Für einen Reset der VHDL-Logik und der GPIO-Expander wird ein vom Board bereits entprellter Taster angewandt.

5.7 Umsetzung der Ansteuerung des RGB-LED-Würfels in VHDL

Dieses Kapitel beschäftigt sich mit der Umsetzung eines I²C-Master-Core und einer animierten Anzeige in VHDL, mit der Laufschrift „HOME“, welche auf dem RGB-LED-Cube dargestellt wird. Die nachfolgende Abbildung 130 zeigt den kompletten Schaltungsentwurf eines „LED_Cube_Swipe_Home“ -Projekts.

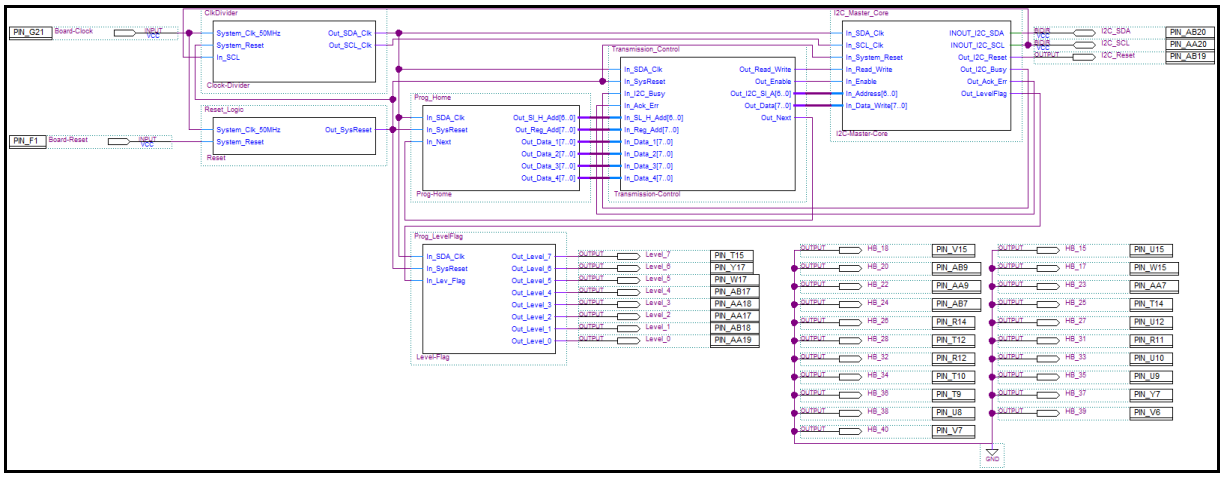


Abbildung 130 - Schematische Darstellung des gesamtes RGB-LED-Cube-VHDL-Projekts im Quartus II

Nachfolgend wird ein veränderter Core beschrieben. Parallel wird der ursprüngliche Quelltext aufgeführt, um Unterschiede zu verdeutlichen.

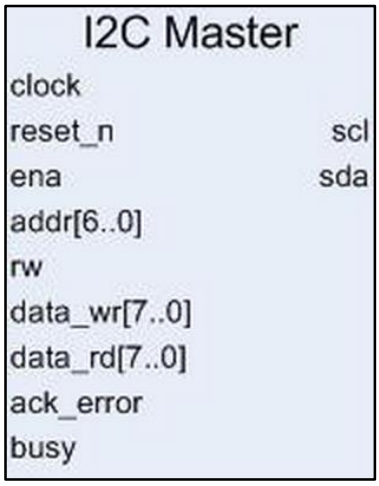


Abbildung 131 - Blockschaltbild I²C-Master-Core von Scott Larson [60]

Ersichtlich ist, dass dieser Core einen internen Taktteiler beinhaltet, welcher zunächst in einen eigenen Block ausgelagert wird. Für die Realisierung des Teilprojektes werden Programmelemente benötigt.

5.7.1 I²C-SDA- und SCL-Takt

Für die Umsetzung des I²C-Master-Core wird ein I²C-Taktteiler benötigt, welcher einen SDA- und SCL-Takt erzeugt. Die folgende Abbildung 132 zeigt das in Altera erstellte Block-Symbol des Taktteilers.

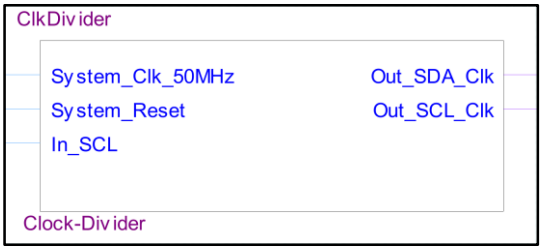


Abbildung 132 - Blockschaltbild eines I²C-SDA- und SCL-Taktteilers im Quartus II

Die Portnamen und deren Funktionsbeschreibungen sind in der nachfolgenden Tabelle 32 aufgelistet.

Portname	Bitgröße	Funktion	Beschreibung
System_Clk_50MHz	1	Eingang	System-Takt 50MHz vom Entwicklungsboard
In_SCL	1	Eingang	Serieller I²C-Systemtakt benötigt für Clock-Stretching
System_Reset	1	Eingang	Asynchroner LOW-aktiv-Pegel Reset
Out_SCL_Clock	1	Ausgang	Serielle I²C-Systemtakt
Out_SDA_Clock	1	Ausgang	Serieller I²C-Datentakt

Tabelle 32 - Portnamen und Funktionen des I²C-SDA- und SCL-Taktteilers

Anschließend erfolgt die Funktionsbeschreibung des Blocks in eine VHDL-Datei mit dem Namen „Clk_Divider.vhd“.

```
01  library ieee;
02  use ieee.std_logic_1164.all;
03  use ieee.numeric_std.all;
```

Quellcode 08 - Import der IEEE-Bibliotheken im I²C-SDA- und SCL-Taktteilers

In diese werden zuerst die benötigten Bibliotheken importiert (01-03). Es folgt die Initialisierung der benötigten In- und Outputs in der sogenannten `entity`.

	Modifiziert	Original
01	…	…
02	`port`	`PORT`
03	`(`	`(`
04	`System_Clk_50MHz : in std_logic;`	`clk : IN STD_LOGIC;`
05	`System_Reset : in std_logic;`	`reset_n : IN STD_LOGIC;`
06	`In_SCL : in std_logic;`	`ena : IN STD_LOGIC;`
07	`Out_SDA_Clk : out std_logic;`	`addr : IN STD_LOGIC_VECTOR`
08	`Out_SCL_Clk : out std_logic`	`(6 DOWNTO 0);`
09	`);`	`rw : IN STD_LOGIC;`
10	…	`data_wr : IN STD_LOGIC_VECTOR`
11		`(7 DOWNTO 0);`
12		`busy : OUT STD_LOGIC;`
13		`data_rd : OUT STD_LOGIC_VECTOR`
14		`(7 DOWNTO 0);`
15		`ack_error: BUFFER STD_LOGIC;`
16		`sda : INOUT STD_LOGIC;`
17		`scl : INOUT STD_LOGIC`
18		`);`
19		…

Quellcode 09 - Definition der Ein- und Ausgänge des I²C-SDA- und SCL-Taktteilers

Innerhalb dieser werden die Ports definiert, welche als Ein- und Ausgänge fungieren (04-08). Es wird ein Eingangsglied für einen 50MHz -Taktgeber und ein Port für das Auslösen einer Reset-Routine benötigt. Nach der `entity` wird die Architektur des Taktteilers angelegt.

	Modifiziert	Original
01	…	…
02	I2C_Divider:	CONSTANT divider : INTEGER :=
03	process (System_Clk_50MHz,	(input_clk/
04	System_Reset, In_SCL)	bus_clk)/4;
05	constant FPGA_Freq :integer :=	SIGNAL data_clk : STD_LOGIC;
06	50000000;	SIGNAL scl_clk : STD_LOGIC;
07	constant I2C_Bus_Clk:integer :=	SIGNAL stretch : STD_LOGIC :=
08	1;	'0';
09	constant Divider :integer :=	BEGIN
10	(FPGA_Freq/	…
11	I2C_Bus_Clk)/	
12	4;	
13	variable I2C_Cnt :integer	
14	range 0 to	
15	Divider*4;	
16	variable stretch :std_logic :=	
17	'0';	
18	…	

Quellcode 10 - Variableninitialisierung des I²C-SDA- und SCL-Taktteilers

In dieser befinden sich Variablendeklarationen und die in VHDL-Code umgesetzte Funktionalität des zu erzeugenden Bausteins. Ein vom Entwicklungsboard gegebener 50MHz -Takt wird auf einen gewünschten I²C-Takt heruntergerechnet. Dies erfolgt indem die Flanken des Taktes gezählt werden und neue LOW- und HIGH-Pegel in einem definierten Intervall ausgegeben werden. Der PCA9698 Expander kann von 1Hz bis maximal 1MHz betrieben werden. Für eine Testanwendung wird zunächst ein 1Hz -Takt gewählt. Gemäß der I²C-Spezifikation existieren vier Taktzustände. Der gewünschte Ausgangstakt erfolgt über die Integer-Konstante `I2C_Bus_Clk` (07). Die Konstante `Divider` (09) dient zur Aufteilung des FPGA-Taktes und I²C-Taktes in vier SCL-Takte. Mit Hilfe einer Integer-Zählervariablen `I2C_Cnt` (13) wird die Zeit für einen Flankenwechsel gezählt. Desweiteren dient die Logik-Variable `stretch` als Erkennungs-Flanke des Clock-Stretching. Anschließend wird ein Prozess für die Funktionalität des Blocks erzeugt.

	Modifiziert	Original
01	…	…
02	if(In_System_Reset = '0') then	IF (reset_n = '0') THEN
03	stretch := '0';	stretch <= '0';
04	I2C_Cnt := 0;	count := ;
05	…	…

Quellcode 11 - Asynchrone Reset-Routine des I²C-SDA- und SCL-Taktteilers

In diesem wird zunächst eine asynchrone Reset-Routine initialisiert, welche bei einem LOW-Pegel des `Reset` -Blocks ausgelöst wird. Beim Auslösen dieses Blocks werden die Variablen `stretch` und `I2C_Cnt` zurückgesetzt (02-04).

	Modifiziert	Original
01	…	…
02	`elsif(rising_edge`	`ELSIF(clk'EVENT AND`
03	`(System_Clk_50MHz))then`	`clk = '1') THEN`
04	`if (I2C_Cnt = Divider*4-1) then`	`IF(count = divider*4-1)`
05	`I2C_Cnt := 0;`	`THEN`
06	`elsif (stretch = '0') then`	`count := 0;`
07	`I2C_Cnt := I2C_Cnt + 1;`	`ELSIF(stretch = '0') THEN`
08	`end if;`	`count := count + 1;`
09	`case I2C_Cnt is`	`END IF;`
10	`when 0 to Divider-1 =>`	`CASE count IS`
11	`Out_Scl_Clk <= '0';`	`WHEN 0 TO divider-1 =>`
12	`Out_Sda_Clk <= '0';`	`scl_clk <= '0';`
13	`when Divider to`	`data_clk <= '0';`
14	`Divider*2-1 =>`	`WHEN divider TO`
15	`Out_Scl_Clk <= '0';`	`divider*2-1 =>`
16	`Out_Sda_Clk <= '1';`	`scl_clk <= '0';`
17	`when Divider*2 to`	`data_clk <= '1';`
18	`Divider*3-1 =>`	`WHEN divider*2 TO`
19	`if (In_Scl = '0') then`	`divider*3-1 =>`
20	`stretch := '1';`	`scl_clk <= '1';`
21	`else`	`IF(scl = '0') THEN`
22	`stretch := '0';`	`stretch <= '1';`
23	`end if;`	`ELSE`
24	`Out_Scl_Clk <= '1';`	`stretch <= '0';`
25	`Out_Sda_Clk <= '1';`	`END IF;`
26	`when others =>`	`data_clk <= '1';`
27	`Out_Scl_Clk <= '1';`	`WHEN OTHERS =>`
28	`Out_Sda_Clk <= '0';`	`scl_clk <= '1';`
29	`end case;`	`data_clk <= '0';`
30	…	`END CASE;`
31		`END IF;`
32		`END PROCESS;`
33		…

Quellcode 12 - Funktionsbeschreibung des I²C-SDA- und SCL-Taktteilers

Bei jeder steigenden Flanke erfolgt eine Überprüfung, ob der `I2C_Cnt` -Zähler sein Maximum erreicht hat (04). Ist dieses nicht erreicht und es findet kein Clock-Stretching statt, wird dieser Wert um eins erhöht. Anschließend erfolgt ein Übergang des Zustandes, welcher dem aktuellen Wert dieser Zählervariable entspricht. Anzumerken ist, dass im dritten Viertel der Taktaufteilung (17-25) eine Erkennung für ein Clock-Stretching implementiert ist. Hält der I²C-Slave die SCL-Flanke auf einem LOW-Pegel, so wird die Stretch-Logik auf `1` gesetzt und der I²C-Zähler wird bei einer nächsten steigenden Flanke nicht erhöht. Eine Verifizierung erfolgt an der Hardware.

Abbildung 133 - Verifikation des I²C-SDA- und SCL-Taktteilers am FPGA

Die Abbildung 133 zeigt, dass der erzeugte Block funktionstüchtig ist.

5.7.2 I²C-Master-Core Block

Der statische Aufbau des I²C-Protokolls ermöglicht eine Realisierung einzelner Zustände. Die folgende Abbildung 134 zeigt das Symbolschaltbild des zu erzeugenden modifizierten I²C-Master-Cores.

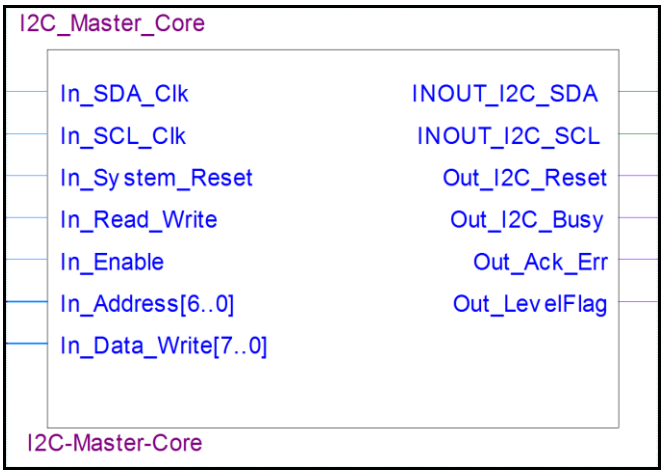


Abbildung 134 - Blockschaltbild des modifizierten I²C-Master-Cores im Quartus II

Die Bedeutungen der einzelnen Ein- und Ausgabeanschlüsse und deren Funktionsbeschreibungen sind in folgender Tabelle zusammengefasst.

Portname	Bitgröße	Funktion	Beschreibung
IN_SDA_Clock	1	Eingang	I²C-Datentakt
IN_SCL_Clock	1	Eingang	I²C-Systemtakt
In_System_Reset	1	Eingang	Asynchroner LOW-aktiv-Pegel Reset
In_Read_Write	1	Eingang	0: Daten schreiben 1: Daten lesen
In_Enable	1	Eingang	0: Übertragung 1: keine Übertragung erwünscht
In_Address	7	Eingang	Slave Adresse
In_Data_Write	8	Eingang	Datenvektor
INOUT_SDA_Clock	1	Ein- und Ausgang	Bidirektionaler Serieller I²C-Datenbus
INOUT_SCL_Clock	1	Ein- und Ausgang	Bidirektionaler Serielle I²C-Systemtakt
Out_I2C_Reset	1	Ausgang	0: Reset 1: kein Reset
Out_I2C_Busy	1	Ausgang	0: I2C-Master im Leerlauf 1: Transaktion in Arbeit
Out_Ack_Err	1	Ausgang	0: keine Fehler 1: Fehler
Out_LevelFlag	1	Ausgang	0: keine Aufforderung 1: Möglicher Ebenenwechsel

Tabelle 33 - Auflistung der Portnamen des I²C-Master-Cores

Die Funktionsbeschreibung erfolgt in einer „I2C_Master_Core.vhd“ -Datei.

```
01  library ieee;
02  use ieee.std_logic_1164.all;
03  use ieee.numeric_std.all;
```

Quellcode 13 - Import der IEEE-Bibliotheken im I²C-Master-Core

Zunächst werden die benötigten IEEE-Bibliotheken importiert (01-03). Es folgt die Initialisierung der benötigten In- und Outputs. Vor Beschreibung der Funktionalität des Moduls, wird eine asynchrone Reset-Routine initialisiert.

	Modifiziert	Original
01	…	…
02	`if (In_Sys_Rst = '0') then`	`IF(reset_n = '0') THEN`
03	`  LevelFlag     <= '0';`	`  state   <= ready;`
04	`  LevelFlag_Cnt := 0;`	`  busy    <= '1';`
05	`  Temp_RW       := '0';`	`  scl_ena <= '0';`
06	`  scl_ena       <= '0';`	`  sda_int <= '1';`
07	`  sda_int       <= '1';`	`  bit_cnt <= 7;`
08	`  bit_cnt       <= 7;`	`  data_rd <= "00000000";`
09	`  Out_Busy      <= '0';`	…
10	`  Out_Data_Read <= "00000000";`	
11	`  addr_rw       <= "00000001";`	
12	`  data_tx       <= "00000000";`	
13	`  state         <= Ready;`	
14	…	

Quellcode 14 - Asynchrone Reset-Routine des I²C-Master-Core bei einer steigenden Flanke

Erfolgt ein LOW-Pegel des `Reset` -Blocks, werden die Variablen auf einen vordefinierten Wert zurückgesetzt (02-13). Anschließend werden zwei parallel laufende Prozesse initialisiert. Der erste Prozess trägt den Namen `SDA_rising_edge`. In diesem werden Programmabläufe deklariert, welche bei einer HIGH-Flanke des SDA-Taktes verarbeitet werden. Zu Beginn einer jeden Transaktion befindet sich der Master-Core in einem Leerlauf, im sogenannten Idle (`Ready`) - Zustand.

	Modifiziert	Original
01	…	…
02	`when Ready =>`	`WHEN ready =>`
03	`  if (In_Enable = ,1') then`	`  IF(ena = ,1') THEN`
04	`    LevelFlag <= ,0';`	`    busy   <= ,1';`
05	`    LevelFlag_Cnt:= 0;`	`    addr_rw<= addr &`
06	`    bit_cnt   := 7;`	`rw;`
07	`    addr_rw   <= In_Address&In_Read_Write;`	`    data_tx<= data_wr;`
08	`    data_tx   <= In_Data_Write;`	`    state  <= start;`
09	`    state     <= Start;`	`  ELSE`
10	`  else`	`    busy   <= ,0';`
11	`    Out_Busy  <= ,0';`	`    state  <= ready;`
12	`    state     <= Ready;`	`  END IF;`
13	`  end if;`	…
14	…	

Quellcode 15 - Idle-Zustand des I²C-Master-Core

Darin verweilt er bis eine HIGH-Flanke des `Enable` -Signals erkannt wird, welches von der Anwendungslogik übermittelt wird (03). Ist dieses Signal vorhanden, werden die übertragenen Daten des vorherigen Blocks auf die entsprechenden Variablen bzw. Signale gesetzt und der `Start` -Zustand wird eingeleitet (04-09). Das `addr_rw` -Signal besteht aus der Hardwareadresse eines ICs und einem Lese- oder Schreib-Bit (07). Zu beachten ist, dass in diesem Zusammenhang die Zählervariable `bit_cnt` mit einem maximalen Integer-Wert von `7` zurückgesetzt wird (06). Die anschließende Abbildung 135 veranschaulicht das Verhältnis der beiden Variablen.

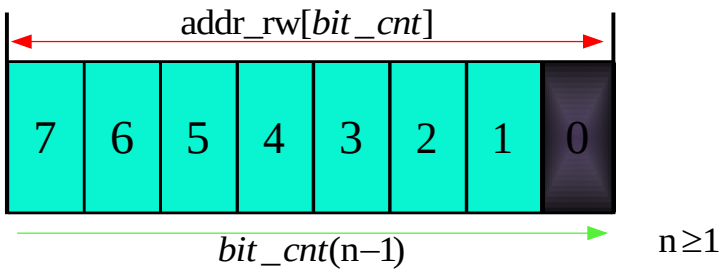


Abbildung 135 - Funktion der bit_cnt -Variable in einem I²C-Byte im I²C-Master-Core

Ersichtlich ist, dass die Variable `bit_cnt` ein Zeiger des Vektors `addr_rw` ist. Desweiteren ist ein `LevelFlag` -Signal deklariert, welches zu Beginn in diesem Zustand ein LOW-Pegel ausgibt (04). Die Bedeutung des Signals wird im nachfolgenden `Start` -Zustand ersichtlich.

	Modifiziert	Original
01	…	…
02	`when Start =>`	`WHEN start =>`
03	`  if (LevelFlag = 4) then`	`  busy    <= '1';`
04	`    LevelFlag     <= '1';`	`  scl_ena <= '1';`
05	`    LevelFlag_Cnt :=0;`	`  sda_int <= addr_rw(bit_cnt);`
06	`  end if;`	`  state   <= command;`
07	`  LevelFlag_Cnt:=`	…
08	`    LevelFlag_Cnt +1;`	
09	`  scl_ena     <= '1';`	
10	`  sda_int     <= addr_rw(bit_cnt);`	
11	`  state       <= Command;`	
12	…	

Quellcode 16 - Start-Zustand des I²C-Master-Core

Die Durchläufe des `Start` -Zustandes werden durch die Variable `LevelFlag_Cnt` gezählt (07,08). Die Pegel des `LevelFlag` -Signals signalisieren einem weiteren Modul `Level_Control` mit einem HIGH-aktiv, dass alle Daten für eine Ebene übertragen wurden und ein möglicher Ebenenwechsel eingeleitet wird. Dafür muss der `Start` -Zustand fünfmal durchlaufen (02). Ist dies erfolgt, wird eine HIGH-Flanke auf die Variable `LevelFlag` gesetzt und der Zähler zurückgesetzt (04,05). Anschließend muss mit der nächsten steigenden Flanke des SDA-Takes dieses Signal auf LOW gezogen werden. Dies erfolgt mit der Abarbeitung des nächsten `Command` -Zustandes. Desweiteren wird bei einer Abfolge des Start-Zustandes ein Flankenwechsel des seriellen I²C-Taktes (SCL) eingeleitet (09). Der erste Adressbit wird auf dem SDA-Kanal übertragen (10). Anschließend erfolgt der Übergang in einen `Command` -Zustand (11).

	Modifiziert	**Original**
01	…	…
02	`when Command =>`	`WHEN command =>`
03	`  LevelFlag <= '0';`	`  IF(bit_cnt = 0) THEN`
04	`  if (bit_cnt = 0) then`	`    sda_int <= '1';`
05	`    sda_int <= '1';`	`    bit_cnt <= 7;`
06	`    bit_cnt <= 7;`	`    state   <= slv_ack1;`
07	`    state   <= Slv_Ack_1;`	`  ELSE`
08	`  else`	`    bit_cnt <= bit_cnt - 1;`
09	`    bit_cnt <= bit_cnt - 1;`	`    sda_int <= addr_rw(`
10	`    sda_int <= addr_rw(bit_cnt-1);`	`                 bit_cnt-1);`
11	`    state   <= Command;`	`    state   <= command;`
12	`end if;`	`  END IF;`
13	…	…

Quellcode 17 - Command-Zustand des I²C-Master-Core

Bei jedem Durchlauf erfolgt die Ausgabe eines LOW-Pegels auf das `LevelFlag` -Signal, welches bereits erläutert wurde (03). Mit jeder steigenden Flanke wird das Byte abgearbeitet (09-11). Ebenfalls wird der Bit-Zähler überprüft, ob dieser den Wert `0` erreicht hat (04). Ist dieser Wert erreicht, wird der Zähler zurückgesetzt, der SDA-Takt zum Slave Acknowledge mit einem HIGH-Pegel freigegeben und der nächste Zustand eingeleitet (05-07). Gemäß des I²C-Protokolls wird ein Acknowledge-Bit bei einer erfolgreichen Übertragung vom Slave erwartet. Dieses Bit wird bei einer fallenden Flanke des SDA-Taktes vorbereitet und im weiteren Verlauf des Cores ausgewertet.

	Modifiziert	Original
01	…	…
02	`when Slv_Ack_1 =>`	`WHEN slv_ack1 =>`
03	`  if (addr_rw(0) = '0') then`	`  IF(addr_rw(0) = '0') THEN`
04	`    sda_int <= data_tx(bit_cnt);`	`    sda_int <= data_tx(bit_cnt);`
05	`    state   <= Data_Write;`	`    state   <= wr;`
06	`  else`	`  ELSE`
07	`    sda_int <= '1';`	`    sda_int <= '1';`
08	`    state   <= Data_Read;`	`    state   <= rd;`
09	`  end if;`	`  END IF;`
10	…	…

Quellcode 18 - Slv_Ack_1-Zustand des I²C-Master-Core

Im `Slv_Ack1` -Zustand wird zunächst das Lese- oder Schreibe-Bit überprüft (03). Ist dieses eine logische „0“, so wird der fortlaufende Schreib-Zustand eingeleitet, indem das erste Daten-Bit `data_tx` auf den SDA-Kanal übertragen und der nächste Zustand eingeleitet wird (04,05).

Um die Allgemeingültigkeit dieses I²C-Cores zu gewährleisten, existiert für eine Datenübertragung nur ein Datenvektor `data_tx`, welcher je nach Anwendung mit neuen Daten überschrieben wird.

Ist eine Lese-Operation erwünscht, so muss das rw-Bit mit einer logischen „1“ gesetzt sein, damit ein Übergang in diese erfolgt (07,08).

Der Datenbit-Übertragungszustand (`Data_Write`) -Zustand dient der Übertragung restlicher zu schreibender Datenbits.

Modifiziert

```
…
when Data_Write =>
  Case_Busy:
  case bit_cnt is
    when 0 =>
      sda_int  <= '1';
      bit_cnt  <= 7;
      addr_rw  <= In_Address &
                  In_Read_Write;
      data_tx  <= In_Data_Write;
      state    <= Slv_Ack_2;
    when 2 =>
      Temp_RW  := In_Read_Write;
      bit_cnt  <= bit_cnt - 1;
      sda_int  <= data_tx (bit_cnt
                           -1);
      State    <= Data_Write;
    when 3 =>
      Out_Busy <= '0';
      bit_cnt  <= bit_cnt - 1;
      sda_int  <= data_tx(bit_cnt
                           -1);
      state    <= Data_Write;
    when 4 =>
      Out_Busy <= '1';
      bit_cnt  <= bit_cnt - 1;
      sda_int  <= data_tx(bit_cnt
                           -1);
      State    <= Data_Write;
    when others =>
      bit_cnt  <= bit_cnt - 1;
      sda_int  <= data_tx(bit_cnt
                           -1);
      state    <= Data_Write;
    end case Case_Busy;
…
```

Original

```
…
WHEN wr =>
  busy <= '1';
  IF(bit_cnt = 0) THEN
    sda_int <= '1';
    bit_cnt <= 7;
    state   <= slv_ack2;
  ELSE
    bit_cnt <= bit_cnt - 1;
    sda_int <= data_tx(bit_cnt
                       -1);
    state   <= wr;
  END IF;
…
```

Quellcode 19 - Data_Write-Zustand des I²C-Master-Core

Eine Überführung in diesen Zustand erfolgt aus dem vorherigen `Slv_Ack_1` und dem nachfolgenden `Slv_Ack_2` -Zustand. Eine Herausforderung stellt bei diesem `Data_Write` -Zustand die Anforderung neuer Daten von der Programmlogik dar (02). Erfolgt eine vorzeitige Datenanforderung durch eine `Busy` -Flanke, werden die Adresse, die Read/Write-Flanke und die zu übertragenden Daten vorzeitig überschrieben. Eine exakte Vorhersage wann diese neuen Daten angefordert werden, steht in einem direkten Bezug zur Programmlogik (vorherige VHDL Blöcke). Je mehr Instanzen die Anforderung durchläuft, desto mehr erhöht sich die Verzögerungszeit (Latches). Die Teilzustände sollten entsprechend dieser Verzögerungen angepasst werden.

Zu beachten ist, dass die zu bearbeitende Anwenderlogik ebenfalls mindestens ein Takt zur Erkennung des rückgeführten Signals benötigt und die Übertragungszeit neuer Daten ebenfalls eine Beachtung findet. Die folgende Abbildung 136 zeigt den Bezug des `Busy` -Signals zwischen dem Master-Core und der vorherigen Logik.

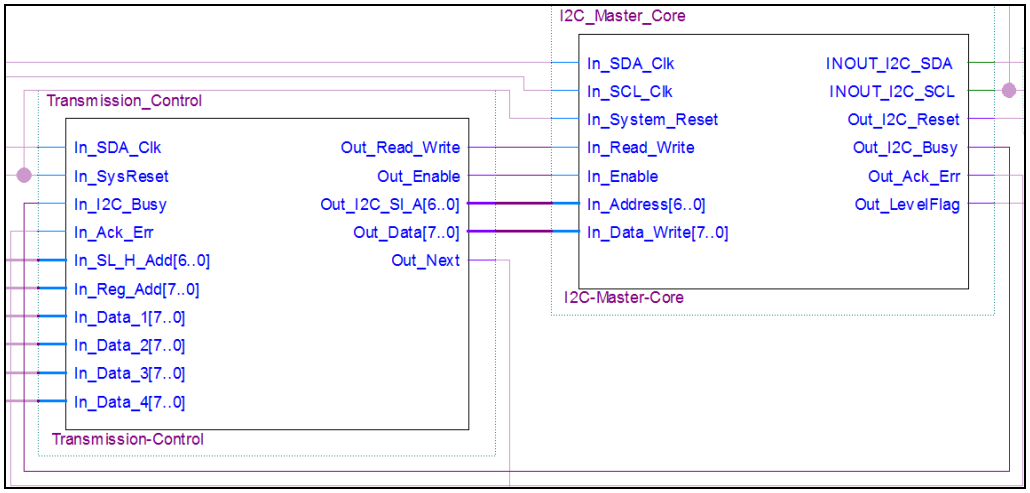


Abbildung 136 - Zusammenhang des Busy-Signals vom I²C-Master-Core und der Kontroll-Logik im Quartus II

Für eine zeitliche Koordination für den beschriebenen Zeitpunkt der Anforderung und das Setzen neuer Daten wird die Variable `bit_cnt` in Zustände aufgeteilt. Ist der Wert `4` erreicht, wird ein HIGH-Pegel auf den `Busy` -Ausgang gesetzt (24,25). Dieser muss unmittelbar mit der nächsten steigenden SDA-Flanke des Taktteilers (`bit_cnt = 3`) auf LOW gesetzt werden (19), damit vom vorherigen `Transmission_Control` -VHDL-Block keine neuen Daten angefordert werden.

Hat der Zähler den Wert `0` erreicht, sind alle Daten übertragen, die Variable zurückgesetzt und die neuen Daten entsprechend auf die initialisierten Variablen gesetzt (06-10).

	Modifiziert	Original
01	…	…
02	`when Slv_Ack_2 =>`	`WHEN slv_ack2 =>`
03	`  if (In_Enable = '1') then`	`  IF(ena = '1') THEN`
04	`    if (Temp_RW = '1') then`	`    busy     <= '0';`
05	`      addr_rw <= In_Address &`	`    addr_rw  <= addr & rw;`
06	`                 In_Read_Write;`	`    data_tx  <= data_wr;`
07	`      data_tx <= In_Data_Write;`	`    IF(addr_rw = addr & rw) THEN`
08	`      state   <= Start;`	`      sda_int<=`
09	`    else`	`        data_wr(bit_cnt);`
10	`      sda_int <= data_tx(bit_cnt);`	`      state  <= wr;`
11	`      state   <= Data_Write;`	`    ELSE`
12	`    end if;`	`      state  <= start;`
13	`  else`	`    END IF;`
14	`    scl_ena   <= '0';`	`  ELSE`
15	`    state     <= Stop;`	`    scl_ena  <= '0';`
16	`  end if;`	`    state    <= stop;`
17	…	`  END IF;`
18		…

Quellcode 20 - Slv_Ack_2-Zustand des I²C-Master-Core

Ist das erste Byte vollständig übermittelt, erfolgt die Überprüfung einer Abbruchbedingung (03). Ist diese nicht vorhanden, wird erneut geprüft, ob eine Lese- oder Schreiboperation erwünscht ist (04). Mit einer logischen „0“ des RW-Bits wird auf dem Datenkanal das erste Bit übertragen und der `Data_Write` -Zustand eingeleitet (10,11). Anschließend wird im nächsten Takt diese Abfolge im `Data_Write` -Zustand fortgeführt.

Ein besonderes Merkmal ist das Lese- oder Schreibe-Bit, welches stets überprüft wird. Dies signalisiert, dass von der Anwenderlogik eine neue Adresse und Daten bereitstehen, so dass der Core in den `Start` -Zustand überführt werden soll, um diese Daten zu übertragen (05-08). Wird weder eine Lese- oder Schreiboperation erwartet, so wird der SCL-Takt für den Slave auf einen LOW-Pegel gezogen und der `Stop` -Zustand eingeleitet (14,15).

Eine Leseoperation des Masters von einem Slave wurde im Rahmen dieser Arbeit nicht angewandt und die Hardwarebeschreibung nicht auf Veränderungen des Quelltextes angepasst. Aus diesem Grund wird der theoretische Ansatz von Herrn Larson betrachtet und auf eventuelle Anpassungen hingewiesen.

	Modifiziert	Original
01	…	…
02	`when Data_Read =>`	`WHEN rd =>`
03	`  if (bit_cnt = 0) then`	`  busy <= '1';`
04	`    if (In_Enable = '1' AND`	`  IF(bit_cnt = 0) THEN`
05	`        In_Read_Write = '1') then`	`    IF(ena = '1' AND addr_rw`
06	`      sda_int      <= '0';`	`        = addr & rw) THEN`
07	`    else`	`      sda_int <= '0';`
08	`      sda_int      <= '1';`	`    ELSE`
09	`    end if;`	`      sda_int <= '1';`
10	`    bit_cnt        <= 7;`	`    END IF;`
11	`    Out_Data_Read <= data_rx;`	`    bit_cnt <= 7;`
12	`    state          <= Master_Ack;`	`    data_rd <= data_rx;`
13	`  else`	`    state <= mstr_ack;`
14	`    bit_cnt        <= bit_cnt - 1;`	`  ELSE`
15	`    state          <= Data_Read;`	`    bit_cnt <= bit_cnt - 1;`
16	`  end if;`	`    state <= rd;`
17	…	`  END IF;`
18		…

Quellcode 21 - Data_Read-Zustand des I²C-Master-Core

Ersichtlich ist, dass erneut die Variable `bit_cnt` in der Funktion als Zeiger eines Zwischenspeichervektors `data_rx` fungiert. Hat der Wert `0` erreicht, erfolgt das Zurücksetzen des Zählers und die Daten werden der Anwenderlogik übertragen und in einen Master-Acknowledge-Zustand überführt (10-12).

Desweiteren findet in Zeile 04 und 05 eine Überprüfung statt, ob der Anwender eine weitere Leseoperation von der gleichen Slave-Adresse wünscht.

Bevor ein weiterer Lesevorgang eingeleitet wird, erfolgt eine Quittierung des empfangenen Bytes vom Master mit Acknowledge-Bit. Anderenfalls wird ein Not-Acknowledge-Bit vom Core übertragen.

	Modifiziert	Original
01	…	…
02	`when Master_Ack =>`	`WHEN mstr_ack =>`
03	`  if (In_Enable = '1') then`	`  IF(ena = '1') THEN`
04	`    if (In_Read_Write = '0') then`	`    busy <= '0';`
05	`      state   <= Start;`	`    addr_rw   <= addr & rw;`
06	`    else`	`    data_tx   <= data_wr;`
07	`      sda_int <= '1';`	`    IF(addr_rw = addr & rw) THEN`
08	`      state   <= Data_Read;`	`      sda_int <= '1';`
09	`    end if;`	`      state   <= rd;`
10	`    Out_Busy  <= '0';`	`    ELSE`
11	`    addr_rw   <= In_Address &`	`      state   <= start;`
12	`                 In_Read_Write;`	`    END IF;`
13	`    data_tx   <= In_Data_Write;`	`  ELSE`
14	`  else`	`    scl_ena   <= '0';`
15	`    scl_ena   <= '0';`	`    state     <= stop;`
16	`    state     <= Stop;`	`  END IF;`
17	`  end if;`	…
18	…	

Quellcode 22 - Master_Ack-Zustand des I²C-Master-Core

Bei Ausführung des `Master-Ack` -Zustandes wird zunächst geprüft, ob eine Transaktion erwünscht ist (03). Ist diese existent wird ebenfalls geprüft, ob ein neuer Lesemodus von der Anwendungslogik erwartet wird (04). Ist dies der Fall, gibt der Core den SDA-Kanal frei und der `Data_Read` -Zustand wird eingeleitet (06-09).

Wird eine neue Transaktion eingefordert, so findet eine Überführung in den `Start` -Zustand statt (05). Bei beiden Möglichkeiten wird die Adresse inklusive des Lese- oder Schreibe-Bits sowie die ersten zu übertragenden Daten der Anwendungslogik entsprechend der Variablennamen gesetzt.

Anzumerken ist, dass bei dieser Operation eine Koordination des `Out_Busy` -Signals, entsprechend möglicher Taktverzögerungen der VHDL-Blöcke und der Modifizierung des Core angepasst werden muss.

Erfolgt keine Aktion (`In_Enable = '0'`) wird ein HIGH-Pegel auf dem SCL-Takt ausgegeben und der `Stop` -Zustand eingeleitet (15,16).

	Modifiziert	Original
01	…	…
02	`when Stop =>`	`WHEN stop =>`
03	`  state  <= Ready;`	`  busy  <= '0';`
04	`end case;`	`  state <= ready;`
05	…	`END CASE;`
06		…

Quellcode 23 - Stop-Zustand des I²C-Master-Core

Dieser Zustand bewirkt eine Überführung in den Leerlauf (03). Empfangene Daten auf dem SDA-Kanal werden bei einer fallenden Flanke des SDA-Taktes des Taktteilers gelesen, während der SCL-Kanal einen HIGH-Pegel besitzt.

Auf dieser Grundlage findet in einem parallel laufenden Prozess, mit dem Namen `SDA_Falling_Edge`, bei einer fallenden Flanke des SDA-Kanals folgende Abhandlung statt.

	Modifiziert	Original
01	…	…
02	`if (In_Sys_Rst = '0') then`	`IF (reset_n = '0') THEN`
03	`  Out_Ack_Err <= '0';`	`  ack_err <= '0';`
04	…	…

Quellcode 24 - Asynchrone Reset-Routine des I²C-Master-Core bei einer fallenden Flanke

Bevor die Funktionalität des zweiten Prozesses erzeugt wird, ist ebenfalls eine asynchrone Reset-Routine initialisiert, welche das Ausgangssignal `Out_Ack_Err` zurücksetzt (03).

	Modifiziert	Original
01	…	…
02	`elsif (falling_edge`	`ELSIF(data_clk'EVENT AND`
03	`(In_Data_Clk)) then`	`data_clk = '0') THEN`
04	`case state is`	`CASE state IS`
05	`when Start =>`	`WHEN start =>`
06	`Out_Ack_Err <= '0';`	`IF(scl_ena = '0') THEN`
07	`when Slv_Ack_1 =>`	`ack_error <= '0';`
08	`if(sda /= '0' OR`	`END IF;`
09	`ack_error = '1') then`	`WHEN slv_ack1 =>`
10	`ack_error <= '1';`	`IF(sda /= '0' OR`
11	`when Data_Read =>`	`ack_error = '1') THEN`
12	`data_rx(bit_cnt) <=`	`ack_error <= '1';`
13	`INOUT_I2C_SDA;`	`END IF;`
14	`when Slv_Ack_2 =>`	`WHEN rd =>`
15	`Out_Ack_Err <= INOUT_I2C_SDA`	`data_rx(bit_cnt) <= sda;`
16	`OR`	`WHEN slv_ack2 =>`
17	`Out_Ack_Err;`	`IF(sda /= '0' OR`
18	`when others => null;`	`ack_error = '1') THEN`
19	…	`ack_error <= '1';`
20		`END IF;`
21		`WHEN OTHERS => NULL;`
22		`END CASE;`
23		…

Quellcode 25 - Funktionsbeschreibung des I²C-Master-Core bei einer fallenden Flanke

Bei einer fallenden Flanke wird im `Start` -Zustand bei einer neuen Transaktion der Acknowledge zurückgesetzt (06). Nachdem alle Bits im steigenden Flankenprozess im `Command` -Zustand übertragen wurden, findet eine Überführung in den `Slv_Ack_1` -Zustand statt.

Je nach dem SDA-Pegel kann eine Aussage getroffen werden, ob alle Daten erfolgreich oder nicht erfolgreich empfangen wurden. Bei einer negativen Übertragung wird dies der Anwenderlogik durch ein Fehlerbit (07-10) mitgeteilt. Analog zu dieser Bedingung ist ebenfalls der `Slv_Ack_2` -Zustand (14-17). Das Lesen der Daten von einem Slave geschieht ebenfalls über den SDA-Kanal bei einer fallenden Flanke. Ist der Lesemodus aktiv, so wird sequentiell der Vektor `data_rx` mit dem jeweiligen Bit beschrieben.

Das Setzen des SCL- und SDA-Taktes erfolgt unsynchron außerhalb eines Prozesses.

Modifiziert

```
01  …
02  with state select
03    sda_ena_n <= In_SDA_Clk
04                 when start,
05
06                 NOT In_SDA_Clk
07                 when stop,
08
09                 sda_int
10                 when others;
11
12    INOUT_I2C_SCL <= '0'
13                     when
14                     scl_ena = '1'
15                     else 'Z';
16    INOUT_I2C_SDA <= '0'
17                     when
18                     sda_ena_n = '0'
19                     else 'Z';
20    Out_LevelFlag <= '0'
21                     when
22                     LevelFlag = '0'
23                     else '1';
24    Out_Reset     <= '0'
25                     when
26                     In_Sys_Rst='0'
27                     else '1';
28  …
```

Original

```
…
WITH state SELECT
sda_ena_n <= data_clk
             WHEN
             start, NOT data_clk
             WHEN
             stop, sda_int
             WHEN OTHERS;

scl <= '0' WHEN (scl_ena = '1'
               AND scl_clk = '0')
           ELSE 'Z';
sda <= '0' WHEN sda_ena_n = '0'
           ELSE 'Z';
…
```

Quellcode 26 - Setzen der Ausgangsports des I²C-Master-Core

Eine Veränderung des SDA-Taktes erfolgt durch eine Initialisierung des `sda_ena_n` -Signals (03).

Ein Durchleiten des SDA-Taktes erfolgt bei Ausführung des `Start` -Zustandes (03,04). Invertiert wird dieser bei einer `Stop` -Bedingung (06,07). Eine Änderung findet ebenfalls bei einer Schreib- oder Leseoperation statt (09,10). Je nach Abhängigkeit des `sda_ena_n` -Signals wird der SDA-Takt verändert. Ein LOW-Pegel wird ausgegeben, wenn dieses ebenfalls LOW-aktiv ist (16-19). Anderenfalls wird dieser Kanal hochohmig. Der SCL-Takt wird auf einen LOW-Pegel gesetzt, unter der Bedingung, dass eine Konjunktion des `scl_ena` -Signals und des SCL-Eingangstaktes existiert. Anderenfalls wird dieser Ein-und Ausgang auf einen hochohmschen Zustand gelegt (12-15).

Einige I²C-ICs unterstützen einen Hardwarereset, welcher als LOW-aktiv durchgeleitet wird (23).

5.7.3 I²C-Übertragungskontroll-Block

Für einen koordinierten Datenaustausch zwischen einer Anwendungslogik und dem I²C-Master-Core wird ein Übertragungskontroll-Block (`Transmission_Control`) (Abbildung 137) erzeugt.

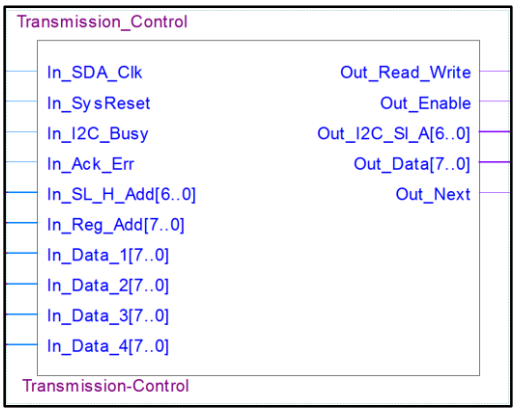


Abbildung 137 - Blockschaltbild der Übertragungskontrolle im Quartus II

In diesem werden die zu übermittelten Daten von einer Anwenderlogik angefordert, zwischengespeichert und zum Core übertragen. Desweiteren können in dieser Logik mögliche Acknowledge-Fehler ausgewertet und eine neue Übertragung initialisiert werden.

Anzumerken ist, dass diese Logik nur für eine Schreiboperation zu einem oder mehreren Slaves optimiert ist. Sie kann für Leseoperationen beliebig erweitert werden.

Die nachkommende Tabelle 34 gibt einen Überblick der erzeugten Ein- und Ausgangsports und deren Funktionen.

Portname	Bitgröße	Funktion	Beschreibung
IN_SDA_Clk	1	Eingang	I²C-Daten-Takt
In_SysReset	1	Eingang	Asynchroner LOW-aktiv-Pegel Reset
In_I2C_Busy	1	Eingang	0: I2C-Master im Leerlauf 1: Transaktion in Arbeit
In_Ack_Err	1	Eingang	0: keine Fehler 1: Fehler
In_SL-H-Add	7	Eingang	Slave Adresse
In_Reg_Add	8	Eingang	Slave Register (1 Byte)
In_Data_1	8	Eingang	Datenvektor
In_Data_1	8	Eingang	Datenvektor
In_Data_1	8	Eingang	Datenvektor
In_Data_1	8	Eingang	Datenvektor
Out_Read_Write	1	Ausgang	0: Daten schreiben 1: Daten lesen
Out_Enable	1	Ausgang	0; I²C-Start 1; I²C-Stop
Out_I2C_SL_H_A	8	Ausgang	Slave Adresse + Lese- oder Schreib-Bit
Out_Data	8	Ausgang	Datenvektor
Out_Next	1	Ausgang	0: Transfer aktiv 1: Transfer inaktiv (Anforderung neuer Daten)

Tabelle 34 - Portnamen und deren Funktionen in der Übertragungskontrolle

Eine Funktionsbeschreibung des Blocks erfolgt in einer „Transmission_Control.vhd“ -Datei.

```
01  library ieee;
02  use ieee.std_logic_1164.all;
03  use ieee.numeric_std.all;
```

Quellcode 27 - Import der IEEE-Bibliotheken in der Übertragungskontrolle

Zunächst werden die benötigten IEEE-Bibliotheken importiert (01-03). Es folgt die Initialisierung der benötigten In- und Outputs.

```
01  …
02  port
03  (
04    In_System_Clk, Button_One: in  std_logic;
05    In_System_Reset, In_I2C_Busy, In_Ack_Err : in  std_logic :='0';
06    In_SL_H_Add              : in  std_logic_vector (6 downto 0);
07    In_Reg_Add, In_Data_1, In_Data_2, In_Data_3, In_Data_4
08                             : in  std_logic_vector (7 downto 0);
09    Out_R_W, Out_Enable      : out std_logic :='0';
10    Out_I2C_Sl_A             : out std_logic_vector
11                               (6 downto 0) :="0100000";
12    Out_Data                 : out std_logic_vector
13                               (7 downto 0) :="00000000";
14    Out_Next                 : out std_logic := '0'
15  );
16  …
```

Quellcode 28 - Definition der Ein- und Ausgänge der Übertragungskontrolle

Der Quellcode 28 gibt einen Überblick der erzeugten Ein- und Ausgänge des Blocks. Ersichtlich ist, dass dieser Block als Vermittlungsstelle zwischen einem Programmablauf und dem I²C-Master-Core fungiert.

Ein `Out_Next` -Ausgang dient als Signalgeber für die Anforderung neuer Daten an das Programm. Anschließend wird die Architektur des Vermittlungsblocks angelegt. In dieser befinden sich Variablendeklarationen und die in VHDL-Code umgesetzte Funktionalität des zu erzeugenden Bausteins.

```
01  …
02  elsif (rising_edge(In_System_Clk)) then
03    Ack_Err     := In_Ack_Err;
04    next_out    <= '0';
05
06    if (Ack_Err = '0') then
07      next_cnt  :=0;
08    end if;
09
10    if (In_I2C_Busy = '1') then
11      next_cnt  := next_cnt+1;
12    end if;
13  …
```

Quellcode 29 - Funktionsbeschreibung der Übertragungskontrolle

Im Master-Core wurde eine Zustandsmaschine erstellt, welche sequentiell durchlaufen wird. Durch eine Signalisierung des `Busy` -Signals `In_I2C_Busy` werden neue Daten von diesem angefordert.

Eine Hilfsvariable `busy_cnt` dient als Zähler der HIGH-Flanken des `Busy` -Signals, welche für das Nachkommen dieser Daten angewandt wird.

```
01  …
02    Dataflow_Controll:
03    case busy_cnt is
04      when 0 =>
05        Enable      <= '1';
06        I2C_Sl_A    <= In_SL_H_Add;
07        R_W         <= '0';
08        Data        <= In_Reg_Add;
09        next_out    <= '0';
10      when 1 =>
11        Enable      <= '1';
12        Data        <= In_Data_1;
13        R_W         <= '0';
14        next_out    <= '0';
15      when 2 =>
16        Enable      <= '1';
17        Data        <= In_Data_2;
18        R_W         <= '0';
19        next_out    <= '0';
20      when 3 =>
21        Enable      <= '1';
22        Data        <= In_Data_3;
23        R_W         <= '0';
24        next_out    <= '0';
25      when 4 =>
26        Enable      <= '1';
27        Data        <= In_Data_4;
28        R_W         <= '0';
29        next_out    <= '0';
30      when 5 =>
31        Enable      <= '1';
32        R_W         <= '1';
33        next_out    <= '1';
34        next_cnt    :=   0;
35      when others =>
36        Enable      <= '0';
37        next_out    <= '0';
38        next_cnt    :=   0;
39      end case;
40  …
```

Quellcode 30 - Zustandsmaschine des busy_cnt-Zählers der Übertragungskontrolle

Für eine Übertragung von sechs Bytes werden sieben Zustände benötigt. Im ersten Zustand erfolgt die Initialisierung einer I²C-Operation (04).

Desweiteren werden die Daten der Slave-Hardware-Adresse, das `R_W` -Bit und der Port entsprechend auf die Variablen gesetzt (05-09). Mit einer erneuten Aufforderung des Cores wird der Zähler um eins erhöht und in den nächsten Zustand gesprungen (Quellcode 29 - 10-12). Mit jeder Aufforderung wird der nächste Zustand eingeleitet.

Sind alle Daten erfolgreich übertragen, wird dem Core mitgeteilt, dass der Wartezustand eingeleitet werden soll. Anschließend wird der Zähler zurückgesetzt und der Anwendungslogik eine HIGH-Flanke übermittelt, welche einen kompletten Durchlauf signalisiert (32,33).

```
01  …
02     Out_Enable     <= Enable;
03     Out_Read_Write<= R_W;
04     Out_I2C_Sl_A   <= I2C_Sl_A;
05     Out_Data       <= Data;
06     Out_Next       <= '0' when next_out = '0' else '1';
07  …
```

Quellcode 31 - Setzen der entsprechenden Ausgangsports der Übertragungskontrolle

Das Setzen der Werte an die entsprechenden Ausgangsports erfolgt außerhalb eines Prozesses.

5.7.4 Ansteuerung der RGB-LED-Cube-Ebenen

Für eine Ansteuerung einzelner Ebenen wird ein eigener `Prog_LevelFlag` -Block (Abbildung 138) erstellt.

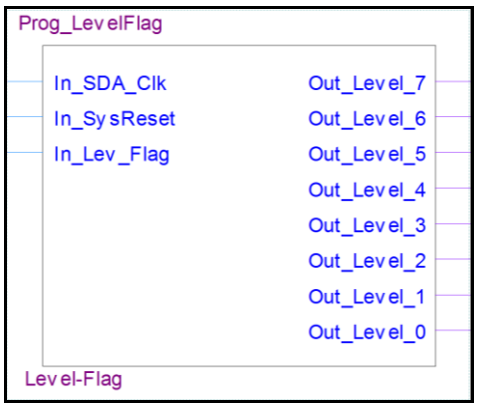


Abbildung 138 - Blocksymbol des Prog_LevelFlag im Quartus II

Die Bedeutungen der einzelnen Ein- und Ausgabe-Anschlüsse und deren Funktionsbeschreibungen sind in folgender Tabelle zusammengefasst.

Portname	Bitgröße	Funktion	Beschreibung
IN_SDA_Clk	1	Eingang	I²C-Daten-Takt
In_SysReset			Asynchroner LOW-aktiv-Pegel Reset
In_Lev_Flag			0: keine Aufforderung 1: Möglicher Ebenenwechsel
Out_Level_0…7	1	Ausgang	Ausgangstakt pro Ebene 0 bis 7

Tabelle 35 - Portnamen und deren Funktionen im Prog_LevelFlag

Die Funktionsbeschreibung erfolgt in einer „Prog_LevelFlag.vhd“ -Datei.

```
01  library ieee;
02  use ieee.std_logic_1164.all;
03  use ieee.numeric_std.all;
```

Quellcode 32 - Import der IEEE-Bibliotheken im Prog_LevelFlag

In dieser werden die benötigten IEEE-Bibliotheken importiert (01-03). Es folgt die Initialisierung der benötigten In- und Outputs. Innerhalb dieser werden die Ports definiert, welche als Ein- und Ausgänge fungieren.

```
01  …
02  signal Level_0, Level_1, Level_2, Level_3, Level_4, Level_5, Level_6,
03  Level_7 : std_logic :='0';
04  …
05  Process_LevelFlag:
06  process (In_SDA_Clk, In_SysReset, In_Lev_Flag)
07    constant Lev_On      : std_logic :='1';
08    constant Lev_Off     : std_logic :='0';
09    variable Level_Flag  : integer range 0 to 7 :=0;
10  …
```

Quellcode 33 - Initialisierung der Variablen im Prog_LevelFlag

Der Core signalisiert durch einen `Level_Flag`, dass die Daten für die Säulen einer Ebene übertragen wurden und ein möglicher Ebenenwechsel stattfinden kann. Es werden drei Eingangsglieder benötigt.

Ein Glied dient als Eingang für einen SDA-Takt und ein anderes Glied für das Auslösen eines Resets. Eine `Level_Flag` -Zählervariable dient zum Zählen des `In_Level_Flag` -Signals (09).

Bevor die Funktionalität des Blocks erzeugt wird, ist eine asynchrone Reset-Routine initialisiert.

```
01  …
02  if(In_SysReset = '0') then
03    Level_Flag := 0;
04    Level_0    <= Lev_Off;
05    …
06    Level_7    <= Lev_Off;
07  …
```

Quellcode 34 - Initialisierung der asynchronen Reset-Routine im Prog_LevelFlag

Beim Auslösen eines Reset-Vorgangs werden die erzeugten Variablen und Ausgangssignale auf einen definierten Standardwert gesetzt (03-06). Anschließend erfolgt die Funktionsbeschreibung des Blocks.

```
01  …
02  elsif (rising_edge(In_SDA_Clk)) then
03    if (In_Lev_Flag = '1') then
04      if (Level_Flag = 7) then
05        Level_Flag := 0;
06      else
07        Level_Flag := Level_Flag +1;
08  …
```

Quellcode 35 - Funktionsbeschreibung bei einer steigenden Flanke im Prog_LevelFlag

Mit jeder steigenden Flanke des SDA-Taktes wird der Pegel des `In_LevelFlag` -Signals geprüft (03). Ist dieses ein HIGH-Pegel, wird der Zähler erhöht. Anderenfalls bleibt der aktuelle Wert des Zählers erhalten. Desweiteren wird geprüft, ob dieser seinen maximalen Wert erreicht hat. Besitzt die Variable diesen Wert, wird er auf den Startwert zurückgesetzt (04,05).

Eine Initialisierung der Ausgangs-Pegel einzelner Ebenen erfolgt in einer Zustandsmaschine des Zählers.

```
01  …
02  case Level_Flag is
03    when 0 =>
04      Level_0 <= Lev_On ;
05      Level_1 <= Lev_Off;
06      …
07      Level_7 <= Lev_Off;
08    when 1 =>
09      Level_0 <= Lev_Off;
10      Level_1 <= Lev_On;
11      …
12      Level_7 <= Lev_Off;
13    when others =>
14      Level_Flag := 0;
15    end case;
16  …
```

Quellcode 36 - Abarbeitung einer Level_Flag-Zustandsmaschine im Prog_LevelFlag

Diese besteht aus acht Zuständen. Die ersten sieben Zustände entsprechen dem aktuellen Wert des `Level_Flag` -Zählers. Zu Beginn der Anwendung, soll nur die erste Ebene mit einer Spannung versorgt werden. Dies wird erreicht, wenn am Ausgangsport der ersten Ebene (`Level_0`) ein HIGH-Pegel und an den restlichen Level-Ports ein LOW-Pegel ausgegeben wird. Wird der Zähler um eins erhöht, erfolgt eine Spannungsversorgung des `Level_1` -Ports, während die restlichen Anschlüsse auf LOW gesetzt werden.

```
01  …
02  Out_Level_0  <= Level_0;
03  …
04  Out_Level_7  <= Level_7;
05  …
```

Quellcode 37 - Setzen der Ausgangsports im Prog_LevelFlag

Das Setzen der Werte an die entsprechenden Ausgangsports erfolgt außerhalb eines Prozesses.

5.7.5 Ansteuerung der RGB-LED-Cube-Säulen

Für die Erstellung einer visuellen Animation wird ein Anwendungslogik-Modul mit dem Namen „Prog_Home.vhd“ erstellt.

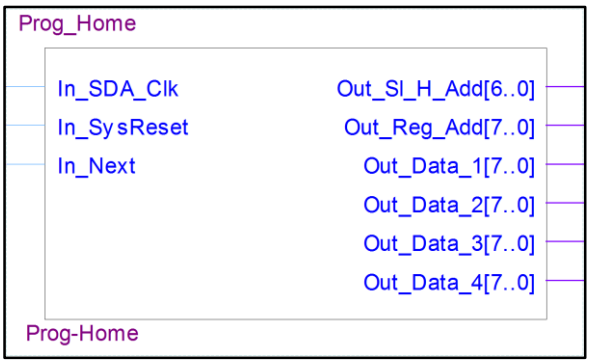


Abbildung 139 - Blockschaltbild des Prog_Home im Quartus II

In dieser Logik werden alle Ebenen nacheinander mit einer Farbe im Sekundentakt durchlaufen.

Ist ein kompletter Durchlauf erfolgt, wird die Farbe der LED auf grün und anschließend auf blau geändert. Diese wird durch eine Adressänderung des jeweiligen I²C-Expanders erreicht. Die Bedeutungen der einzelnen Ein- und Ausgabeanschlüsse des Moduls sind in folgender Tabelle zusammengefasst.

Portname	Bitgröße	Funktion	Beschreibung
In_SDA_Clk	1	Eingang	Serieller I²C-SDA-Takt
In_SysReset			Asynchroner LOW-aktiv-Pegel Reset
In_Next			Aufforderung neuer Daten vom Core
Out_Sl_H_Add	6	Ausgang	Slave(s) Adresse
Out_Reg_Add	7		Slaves (s) Register
Out_Data_1 … 4			Slave(s) Daten

Tabelle 36 - Portnamen und deren Funktion im Prog_Home

Anschließend erfolgt die Funktionsbeschreibung des Blocks in der `entity`.

```
01  …
02  entity Prog_Home is
03  port
04  (
05    In_Sys_Clk, In_Sys_Rst, In_Button_One, In_Ready, In_Next :
06      in  std_logic;
07    In_Lev_Flag  : in  std_logic;
08    Out_Sl_H_Add : out std_logic_vector (6 downto 0);
09    Out_Reg_Add  : out std_logic_vector (7 downto 0);
10    Out_Data_1, Out_Data_2, Out_Data_3, Out_Data_4 :
11      out std_logic_vector (7 downto 0);
12    );
13  end entity;
14  …
```

Quellcode 38 - Initialisierung Ein- und Ausgänge im Prog_Home

Für einen koordinierten Ablauf wird als Systemtakt der erzeugte SDA-Takt angewendet. Ein Reset-Eingang dient zum Zurücksetzen deklarierter Signale, Variablen und Ausgangsports. Ein `In_Next` -Signal signalisiert eine Aufforderung neuer Daten vom Core.

Eine Signalübertragung erfolgt jeweils über einen Busausgang für Slave-Hardware-Adresse, IC(s) Register und die I²C-Daten (`1…4`) der Ports. Zu Beginn der Architektur werden Signale initialisiert.

```
01  …
02  -- Array of Adresses
03  type Data_Sl_H_Add_Arr is array (natural range <>)
04    of std_logic_vector (6 downto 0);
05  signal Sl_H_Add: Data_Sl_H_Add_Arr (0 to 6):=
06  (
07    ("0100000"), -- Zero
08    ("0100001"), -- One
09    ("0100010"), -- Two
10    ("0100011"), -- Three
11    ("0100100"), -- Four
12    ("0100101"), -- Five
13    ("1101110")  -- ALL GPIO
14  );
15  …
```

Quellcode 39 - Hardwareadressen-Array im Prog_Home

Ein Signal vom Typ Array enthält angewandte Hardwareadressen, welche in absteigender Reihenfolge in den Spalten initialisiert werden. Ein PCA9698 besitzt fünf Portregister.

```
01  …
02  type Data_Reg_Add_Arr is array (natural range <>)
03    of std_logic_vector (7 downto 0);
04  signal Reg_Add: Data_Reg_Add_Arr (0 to 4):=
05  (
06    ("10011000"), -- Zero
07    ("10011001"), -- One
08    ("10011010"), -- Two
09    ("10011011"), -- Three
10    ("10011100")  -- Four
11  );
12  …
```

Quellcode 40 - Register-Array im Prog_Home

Diese werden in absteigender Reihenfolge in einem Array mit dem Namen `Data_Reg_Add_Arr` initialisiert.

Für die Erzeugung eines Bildes, welches auf der 3D-Matrix erscheint, wird der Cube von der Vorderansicht als Gesamtobjekt betrachtet. Die einzelnen Bilder einer Animation werden separat als Bitfolge in einem Array gespeichert. Fortlaufend wird pro Bild ein Array erzeugt. Die Abfolge der Arrays erfolgt in einem definierten Intervall.

Aufgrund der begrenzten Logikelemente eines FPGAs können nicht unendlich viele Arrays erzeugt werden. Daher empfiehlt es sich, bei größeren Bildabfolgen, die Arrays extern zu speichern. Diese können zum Beispiel durch einen Parser dynamisch ausgelesen werden. Desweiteren ist es möglich Schaltwerke, zum Beispiel ein Schieberegister, anzuwenden.

Säule / Ebene	0	1	2	3	4	5	6	7
7	00h	00h	00h	00h	00h	00h	00h	00h
6	00h	00h	00h	00h	00h	00h	00h	00h
5	00h	00h	00h	00h	00h	00h	00h	00h
4	00h	00h	00h	18h	18h	00h	00h	00h
3	00h	00h	00h	18h	18h	00h	00h	00h
2	00h	00h	00h	00h	00h	00h	00h	00h
1	00h	00h	00h	00h	00h	00h	00h	00h
0	00h	00h	00h	00h	00h	00h	00h	00h

Tabelle 37 - Beispielarray zur Übertragung eines Bildes auf den Cube im Prog_Home

Die Zeilen repräsentieren jeweils eine Säule, während die Spalten in Ebenen aufgeteilt werden. Das Beispiel aus Tabelle 37 zeigt mittig vier aktive LEDs. Die zu übertragenden Bits werden in Big-Endian angeordnet. Durch diese Verfahrensweise entsteht ein dreidimensionales Array.

Nach der `entity` wird die Architektur der Säulenansteuerung angelegt. In dieser befinden sich Variablendeklarationen und die in VHDL-Code umgesetzte Funktionalität des zu erzeugenden Bausteins.

```
01  …
02  -- Array Level Zero
03  type Data_HomeAni_Arr_0 is array (natural range <>,
04    natural range <>) of std_logic_vector (7 downto 0);
05  signal Data_HomeAni_0:
06  Data_HomeAni_Arr_0 (7 downto 0, 7 to 0) :=
07  (-- Level 0
08  ("00000000", "00000000", "00000000", "00000000",
09  "00000000", "00000000", "00000000", "00000000"),
10  -- Level 1
11  ("11111111", "11111111", "11111111", "11111111",
12  "11111111", "11111111", "11111111", "11111111")
13  …-- Level 3,4,5,6,7…
```

Quellcode 41 - Array für ein dreidimensionales Bild im Prog_Home

Zunächst wird das erste anzuzeigende Bild initialisiert. Bei diesem ist nur die erste Ebene (08,09) aktiv, während die restlichen Ebenen inaktiv sind (11-12).

Damit alle Ebenen in einem Sekundentakt durchlaufen werden, sind sieben weitere Arrays zu erzeugen.

```
01  …
02  type Data_HomeAni_Arr_0 is array (natural range <>,
03    natural range <>) of std_logic_vector (7 downto 0);
04  signal Data_HomeAni_0:
05  Data_HomeAni_Arr_0 (7 downto 0, 7 to 0) :=
06  (-- Level 0
07  ("11111111", "11111111", "11111111", "11111111",
08  "11111111", "11111111", "11111111", "11111111"),
09  -- Level 1
10  ("00000000", "00000000", "00000000", "00000000",
11  "00000000", "00000000", "00000000", "00000000"),
12  -- Level 2
13  ("11111111", "11111111", "11111111", "11111111",
14  "11111111", "11111111", "11111111", "11111111"),
15  … -- Level 3,4,5,6,7 …
```

Quellcode 42 - Array für nachfolgendes Bild im Prog_Home

Der Quellcode 42 verdeutlicht die Veränderungen des zweiten Arrays. Desweiteren wird ein Zwischenspeicher-Array erzeugt, welches im weiteren Verlauf näher erläutert wird.

```
01  …
02  type Cach_Arr is array (natural range <>,
03    natural range <>) of std_logic_vector (7 downto 0);
04  signal Data_Cach_Arr:
05  Cach_Arr (7 downto 0, 0 to 7) :=
06    (others => (others => "11111111"));
07  …
```

Quellcode 43 - Initialisierung eines Zwischenspeicher-Arrays für Bilder im Prog_Home

Nachdem die Arrays initialisiert sind, erfolgt die Deklaration weiterer benötigter Variablen.

```
01  …
02  constant Sek_Takt : integer :=52800;
03  variable Chip_Cnt : integer range 0 to ((Sek_Takt)*7)
04  := 0;
05  …
```

Quellcode 44 - Initialisierung eines Zählers für einen Bildwechsel im Prog_Home

Mit der Konstante `Sek_Takt` (02) werden die benötigten Flanken für eine I²C-Übertragung definiert. Diese ist proportional zu dem Systemtakt von `105600Hz`. Die Übertragung von I²C-Daten erfolgt in einer sequentiellen Abfolge. Resultierend wird ein Bit mit jeder steigenden Flanke des SDA-Taktes übertragen. Zustandsübergänge benötigen jeweils einen Taktzyklus. Für die Übertragung von 48 Bits auf vier Register des ICs werden 55 Taktzyklen benötigt. Die nachfolgenden Tabelle 38 gibt einen Überblick der angewandten Taktzyklen.

Abfolge	Nummer	I²C-Core Zustand	Taktzyklen	Funktion	Bits
1.	1.	Start	1	Hardwareadresse	1
	2.	Command	7		7
	3.	Command	1	Zustandsübergang	
2.	4.	Slv_Ack1	1	Port/Register	1
	5.	Write	7		7
	6.	Write	1	Zustandsübergang	
3.	7.	Slv_Ack2	1	Daten n	1
	8.	Write	7		7
	9.	Write	1	Zustandsübergang	
4.	10.	Slv_Ack2	1	Data n+1	1
	11.	Write	7		7
	12.	Write	1	Zustandsübergang	
5.	13.	Slv_Ack2	1	Data $n+1$	1
	14.	Write	7		7
	15.	Write	1	Zustandsübergang	
6.	16.	Slv_Ack2	1	Data $n+1$	1
	17.	Write	7		7
	18.	Write	1	Zustandsübergang	
	19.	Slv_Ack2	1	Zustandsübergang	
Gesamt:			55		48

Tabelle 38 - Zu beachtende Flanken des I²C-Master-Core im Prog_Home

Aus der Tabelle 38 ist ersichtlich, dass für eine Übertragung von 48 Bits 55 HIGH-Flanken benötigt werden.

Die Berechnung der Frequenz für einen kompletten Durchlauf bei einer Taktfrequenz von 60Hz ergibt sich wie folgt:

Formel 19 - Variante 1 der Herleitung des benötigten SDA-Taktes

$$60\text{Hz} \cdot 8\text{Ebenen} = 480\text{Hz}$$

$$480Hz \cdot 55Bits = 26400\text{Hz}$$

$$26400Hz \cdot 2ICs = 52800\text{Hz}$$

$$52800\text{Hz} \cdot 2\text{Flanken} = 105600\text{Hz}$$

Resultierend aus der Berechnung sind 105600Hz erforderlich, welche in dem zu erzeugenden SDA-Taktteilers initialisiert werden.

Ein weiterer Schritt ist, dass für eine Datenübertragung auf einem 4-Port IC 55 HIGH-Flanken benötigt werden. Für ein Zurücksetzen werden erneut 55 Bit benötigt.

	Durchläufe pro IC	Durchlauf	Flanken
	1	1 LEDs online	55
		2 LEDs offline	55
	2	3 LEDs online	55
		4 LEDs offline	55
Gesamt			220

Tabelle 39 - Benötigte Flanken zur Übertragung von Daten im Prog_Home

Formel 20 - Variante 2 der Herleitung des benötigten SDA-Taktes

$$220\text{HIGH} - \text{Flanken} \cdot 60\text{Hz} = 13200\text{Hz}$$

$$13200\text{Hz} \cdot 8\,\text{Ebenen} = 105600\text{Hz}$$

Die Ersatzrechnung zeigt, dass 105600Hz für die Übertragung von 220 Bits bei 60Hz pro Ebene benötigt werden.

Bevor die Funktionalität des Blocks beschrieben wird, erfolgt die Umsetzung einer asynchronen Reset-Routine.

```
01  …
02  if(In_SysReset = '0') then
03    DataColumn_Cnt   := 0;
04    FirstDataChg_Cnt := 0;
05    SecDataChg_Cnt   := 0;
06    ChipTime_Cnt     := 0;
07    DataChipTime_Cnt := 0;
08    Chip_A           := 0;
09    Chip_B           := 1;
10    In_Next_Cnt      := 0;
11    Data_Add         <= "0000000";
12    Data_Port        <= Offline;
13    Data_Home_1      <= Offline;
14    Data_Home_2      <= Offline;
15    Data_Home_3      <= Offline;
16    Data_Home_4      <= Offline;
17  …
```

Quellcode 45 - Asynchrone Reset-Routine des Prog_Home

Beim Auslösen dieser Routine werden alle erzeugten Signale und Variablen auf einen Standardwert gesetzt. Anschließend erfolgt die Funktionsbeschreibung des Blocks.

```
01  …
02  elsif (rising_edge(In_Sys_Clk)) then
03  Chip_Cnt := Chip_Cnt +1;
04  Case_Chip_Cnt_2:
05  case Chip_Cnt_2 is
06    when 0 =>
07      if (Chip_Cnt = Sek_Takt) then
08        Chip_Cnt_2  := Chip_Cnt_2+1;
09      else
10      Chip_Cnt_2    := 0;
11      end if;
12      for x in 0 to 7 loop
13        for y in 0 to 7 loop
14          Data_Cach_Arr (x, y)<=Data_HomeAni_0 (x, y);
15      end loop ;
16    end loop ;
17  …
```

Quellcode 46 - Funktionsbeschreibung des ersten Bild-Array im Prog_Home

Eingeleitet wird der Programmablauf mit einer Synchronisation auf den erzeugten SDA-Takt (02). Mit jeder steigenden Flanke des Taktes wird eine Zählervariable `Chip_Cnt` um jeweils eins erhöht (03). Desweiteren erfolgt der Einsatz eines zweiten Integer Zählers mit dem Namen `Chip_Cnt_2`.

Dieser Zähler ist zu Beginn des Funktionsblocks auf `0` gesetzt. Dabei erfolgt ein Sprung in eine `Case_Chip_Cnt_2` -Zustandsmaschine (05). In dieser wird, für die einzelnen Zustände, eine Funktionsbeschreibung deklariert.

Zunächst wird geprüft, ob der `Chip_Cnt` -Zähler sein Maximum von 1s (`52800` Schritte) erreicht hat (07). Ist der aktuelle Wert kleiner als dieser, wird die Variable `Chip_Cnt_2` auf `0` gesetzt, um in diesem Zustand bei einer nächsten steigenden Flanke zu verweilen (10). Anschließend wird das erste Array mittels einer Loop-Schleife auf ein Zwischenspeicher-Array übertragen (12-16). Sind `52800` Schritte erfolgt, findet mit der nächsten Flanke ein Zustandsübergang statt (08).

```
01  …
02  when 1 =>
03    if (Chip_Cnt = ((Sek_Takt)*2)) then
04      Chip_Cnt_2  := Chip_Cnt_2+1;
05    else
06      Chip_Cnt_2 := 1;
07    end if;
08    for x in 0 to 7 loop
09      for y in 0 to 7 loop
10        Data_Cach_Arr (x, y)<=Data_HomeAni_1 (x, y);
11      end loop ;
12    end loop ;
13  …
```

Quellcode 47 - Funktionsbeschreibung des zweiten Bild-Array im Prog_Home

Ersichtlich ist, dass die Funktionalität dem vorherigen Zustand entspricht. Dieser wird solange ausgeführt, bis die Zählervariable `Chip_Cnt` seinen zweiten Maximalwert erreicht hat (03). Für acht Ebenen sind Zustandswechsel erforderlich, indem das Zwischenspeicher Array mit neuen Daten gefüllt wird.

Zu beachten ist der nachfolgende letzte Zustand des `Chip_Cnt_2` -Zählers.

```
01  …
02  when 7 =>
03    if (Chip_Cnt = ((Sek_Takt)*8)) then
04      case Colour_Cnt is
05        when 0 =>
06          Chip_A         := 2;
07          Chip_B         := 3;
08          Colour_Cnt     := Colour_Cnt +1;
09        when 1 =>
10          Chip_A         := 4;
11          Chip_B         := 5;
12          Colour_Cnt     := Colour_Cnt +1;
13        when others =>
14          Chip_A         := 0;
15          Chip_B         := 1;
16          Colour_Cnt     :=0;
17      end case;
18      Chip_Cnt     := 0;
19      Chip_Cnt_2   := 0;
20    else
21      Chip_Cnt_2 := 7;
22    end if;
23    for x in 0 to 7 loop
24      for y in 0 to 7 loop
25        Data_Cach_Arr (x, y) <= Data_HomeAni_7 (x, y);
26      end loop;
27    end loop;
28  …
```

Quellcode 48 - Funktionsbeschreibung eines Farbwechsels im Prog_Home

Um einen Überlauf in einen unbestimmten Zustand zu vermindern, erfolgt beim Erreichen des letzten möglichen Maximalwertes (`(Sek_Takt)*8`) des `Chip_Cnt` - Zählers (03) ein Zurücksetzen der Variablen `Chip_Cnt` und `Chip_Cnt_2` (18,19). Ebenfalls wird der aktuelle Wert der Zählervariable `Colour_Cnt` geprüft. Durch Erhöhen des Zählers werden neue ICs durch die Variablen `Chip_A` und `Chip_B` ausgewählt und ein Farbwechsel wird initialisiert (04-17).

```
01  …
02  when others =>
03    Chip_Cnt     :=0;
04    Chip_Cnt_2   :=0;
05  end case Case_Chip_Cnt_2;
06  …
```

Quellcode 49 - Zurücksetzen einer Zählervariablen in der Chip_Cnt_2-Zustandsmaschine im Prog_Home

Erfolgt bei der Abarbeitung der Zustände ein Übergang in einen nicht erwünschten Zustand, so werden die Zustandszählervariablen in diesen zurückgesetzt (03,04).

Nachdem die Zustandsmaschine initialisiert ist, erfolgt eine Überprüfung der `In_Next` -Flanke im gleichen Prozess. Diese signalisiert eine Anforderung neuer Daten vom Core.

```
01  …
02  if (In_Next = '1') then
03    Column_Cnt  := Column_Cnt + 1;
04    In_Next_Cnt := In_Next_Cnt + 1;
05  end if;
06  …
```

Quellcode 50 - Überprüfung einer In_Next-Flanke im Prog_Home

Ist diese Flanke ein HIGH-Pegel, werden die Zählervariablen `Column_Cnt` und `In_Next_Cnt` erhöht (03,04).

Die Übertragung von I²C-Daten an nachfolgende Logikelemente erfolgt mit einer `Case_Next_Flag` -Zustandsmaschine. In dieser werden die Zählervariablen `CFlow_Cnt` und `CFlow2_Cnt` von Zustand zu Zustand neu initialisiert. Diese Variablen dienen einem synchronisierten Durchlauf der Säulen.

```
01  …
02  Case_Next_Flag:
03  case Next_Flag is
04    when 0 =>
05      if (In_Next_Cnt = 4) then
06        Next_Flag := 1;
07      else
08        Next_Flag  := 0;
09      end if;
10      CFlow_Cnt  := 0;
11      CFlow2_Cnt := 0;
12      when 1 =>
13        if (In_Next_Cnt = 8) then
14          Next_Flag := 2;
15        else
16          Next_Flag  := 1;
17        end if;
18        CFlow_Cnt  := 1;
19        CFlow2_Cnt := 1;
20  …
```

Quellcode 51 - Vorbereitung der I²C-Datenübertragung mittels eines Next_Flag-Zählers im Prog_Home

Die `Next_Flag` -Zustandsmaschine besitzt acht Zustände.

Für die Übertragung auf einem I²C-Expander werden vier Ports mit jeweils acht Bits angewandt. Werden gewünschte Daten auf diesen geschrieben, sollten sie anschließend durch logische „0“en gelöscht werden.

Um dies zu erreichen, wird im ersten Zustand geprüft, ob der Master-Core vier HIGH-Flanken gesendet hat (05). Ist dies geschehen, wird die Zählervariable um eins erhöht und mit der nächsten steigenden Flanke der Folgezustand eingeleitet (06). Anderenfalls wird der aktuelle Zustand beibehalten (08).

```
01  …
02  when 7 =>
03    if (In_Next_Cnt = 32) then
04      Next_Flag   := 0;
05      In_Next_Cnt :=0;
06    else
07      Next_Flag   := 7;
08    end if;
09    CFlow_Cnt     := 7;
10    CFlow2_Cnt    := 7;
11  …
```

Quellcode 52 - Achter Zustand des Next_Flag-Zählers im Prog_Home

Zu beachten ist, dass beim Erreichen des Maximalenwertes der `In_Next_Cnt` -Variable, alle angewandten Zählervariablen der Zustandsmaschine zurückgesetzt werden (04,05).

```
01  …
02  when others =>
03    Next_Flag    := 0;
04    In_Next_Cnt  := 0;
05  …
```

Quellcode 53 - Zurücksetzen der Zählervariablen in der Next_Flag-Zustandsmaschine im Prog_Home

Wird beim Ablauf ein undefinierter Zustand eingeleitet, so werden die zwei Zählervariablen zurückgesetzt (03,04).

Das Setzen neuer Werte wird mit vier Zwischenspeichervariablen, `Data_Home_1` bis `Data_Home_4`, erreicht. Dies erfolgt in einer `Flow_Column` -Zustandsmaschine.

```
01  …
02  Flow_Column:
03  case Column_Cnt is
04  --First Hardware Chip 1 Online
05  when 0 =>
06    Data_Add    <= Sl_H_Add (Chip_A);
07    Data_Port   <= Reg_Add (0);
08
09    Case_CFlow_1:
10    case CFlow_Cnt is
11      when 0 =>
12        Data_Home_1 <= Data_Cach_Arr (0,0);
13        Data_Home_2 <= Data_Cach_Arr (0,1);
14        Data_Home_3 <= Data_Cach_Arr (0,2);
15        Data_Home_4 <= Data_Cach_Arr (0,3);
16      when 1 =>
17        Data_Home_1 <= Data_Cach_Arr (1,0);
18        Data_Home_2 <= Data_Cach_Arr (1,1);
19        Data_Home_3 <= Data_Cach_Arr (1,2);
20        Data_Home_4 <= Data_Cach_Arr (1,3);
21  …
22       when others =>
23          CFLow_Cnt :=0;
24        end case Case_CFlow_1;
25  …
```

Quellcode 54 - Funktionsbeschreibung zur Ansteuerung des ersten IC im Prog_Home

In dieser werden die einzelnen Programmabläufe in einer Zustandsmaschine für den Wert des `Column_Cnt` -Zählers deklariert. Zu Beginn werden die Variablen `Data_Add` mit der gewünschten Hardwareadresse und `Data_Port` mit dem entsprechenden Startregister gesetzt (06,07). Je nach Wert des `CFlow_Cnt` -Zählers werden die entsprechenden Vektoren des Zwischenspeicher-Arrays zugewiesen.

```
01  …
02  when 1 =>
03    Data_Add    <= Sl_H_Add (Chip_A);
04    Data_Port   <= Reg_Add (0);
05    Data_Home_1 <= Offline;
06    Data_Home_2 <= Offline;
07    Data_Home_3 <= Offline;
08    Data_Home_4 <= Offline;
09  …
```

Quellcode 55 - Offlineschalten des ersten IC im Prog_Home

Der Quellcode 55 verdeutlicht die Veränderungen des zweiten Zustands.

Der PCA9698 behält die Pegel der Ports bis diese überschrieben oder durch eine Reset-Routine zurückgesetzt werden.

In diesem Beispiel sollen auf jeder Ebene unterschiedliche LEDs leuchten. Dazu müssen die Ports des ICs auf einen LOW-Pegel gesetzt werden, bevor der zweite IC Daten erhält.

```
01  …
02    Out_Sl_H_Add  <= Data_Add   ;
03    Out_Reg_Add   <= Data_Port  ;
04    Out_Data_1    <= Data_Home_1;
05    Out_Data_2    <= Data_Home_2;
06    Out_Data_3    <= Data_Home_3;
07    Out_Data_4    <= Data_Home_4;
08  …
```

Quellcode 56 - Setzen der Ausgangsports im Prog_Home

Das Setzen der Werte an die entsprechenden Ausgangsports erfolgt außerhalb eines Prozesses.

5.7.6 Puls-Wide-Modulation auf dem RGB-LED-Cube

Eine Software-PWM kann in dem Projekt unterschiedlich realisiert werden. Eine Variante wird in diesem Teilkapitel an einem einfachen Beispiel kurz erläutert und weitere Ansätze werden genannt. Die Umsetzung einer Software-PWM erfolgt im Projekt über die Ansteuerung der GPIO-Expander und findet im `Prog_Home` -Block in der „Prog_Home.vhd“ -Datei statt. Eine einfache Möglichkeit der Änderung der Helligkeitswerte oder ein Farbwechsel wird über eine Zählervariable `Chip_Cnt` und `Sek-Takt` realisiert.

```
01  …
02    if (Chip_Cnt < Sek_Takt) then
03      Chip_A := 0;
04      Chip_B := 1;
05      Chip_Cnt := Chip_Cnt +1;
06    elsif (Chip_Cnt < ((Sek_Takt)*2)) then
07      Chip_A := 2;
08    …
09    else
10      Chip_A :=0;
11      Chip_B :=0;
12      Chip_Cnt := 0;
13  …
```

Quellcode 57 - Initialisierung eines einfachen PWM-Zählers im Prog_Home

Mittels Anweisungen werden die Bedingungen abgearbeitet. Befindet sich die Variable `Chip_Cnt` in einem der Intervalle, wird diese hochgezählt und entsprechend die Variablen `Chip_A` und `Chip_B` initialisiert. Dies entspricht einer I²C-Adresse. Ein Hochzählen jeder Variable erfolgt synchron zum SDA-Takt, ist jedoch asynchron zum Aufbau des Bildwechsels. Bei einem gleichbleibenden Takt werden die zu übertragenden Bits auf einen „zufälligen“ IC übertragen. Dabei werden die An- und Ausschaltzeiten der LEDs verkürzt. Eine Erhöhung des SDA-Taktes ist im Beispiel nicht erforderlich.

Anschließend erfolgt das Setzen der Variablen in der bereits beschrieben `Column_Cnt` -Zustandsmaschine.

```
01  …
02  Flow_Column:
03  case Column_Cnt is
04    when 0 =>
05      Data_Add  <= Sl_H_Add (Chip_A);
06  …
07    when 1 =>
08      Data_Add  <= Sl_H_Add (Chip_A);
09  …
10    when 2 =>
11      Data_Add  <= Sl_H_Add (Chip_B);
12  …
13    when 3 =>
14      Data_Add  <= Sl_H_Add (Chip_B);
15  …
```

Quellcode 58 - Setzen der Variablen für I²C-Adressen im Prog_Home

Die Hardwareadressen werden entsprechend gesetzt (05-14). Ein Video der Animation befindet sich im Anhang auf DVD unter der Bezeichnung „3D_RGB_LED_Matrix_Farbspiel_PWM.m4v“.

Eine PWM kann ebenfalls über den `Prog_LevelFlag` -Modul erfolgen. Dadurch werden die An- und Ausschaltzeiten der LEDs einer gesamten Ebene gesteuert. Für diese Umsetzung ist gegebenenfalls die Erzeugung bzw. Umstrukturierung der Anwenderlogik erforderlich. Aufgrund der Vielfältigkeit der Umsetzung wurde kein PWM-Modul für eine PWM-Logik und eine Farb-Logik erzeugt. Diese kann individuell durch Abhängigkeiten von den darzustellenden Bildern implementiert werden.

5.8 Letzte Schritte und Programmierung des FPGA

Eine Umsetzung der erzeugten VHDL-Module erfolgt in den nachfolgenden Teilkapiteln auf einem Altera DE0 Board und auf einem Altium NanoBoard 3000.

5.8.1 Live-Test Altera DE0 Board

Nachdem alle Module erzeugt und platziert wurden, müssen diese mit Hilfe von Busleitungen und Leitungen miteinander verbunden werden. Die Abbildung 140 zeigt die vollständige Darstellung in Quartus II.

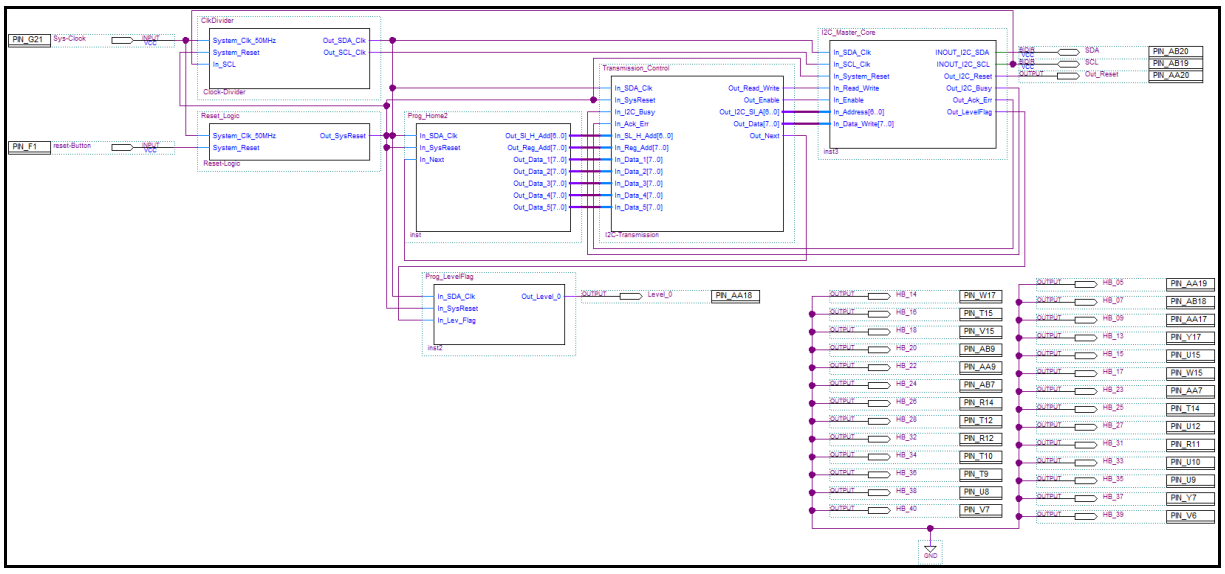


Abbildung 140 - Schematische Darstellung des gesamten Projektes im Quartus II

Angewandt werden zwei Eingangspins für den Board-Takt und den Taster. Der GPIO-1 (JP2) Port des Cyclone 3 Boards verfügt über 40-PINs. Eine Übertragung der I²C-Leitungen (SDA und SCL) erfolgt jeweils über eine bidirektionale Datenleitung. Für die verbleibenden Anschlüsse wird jeweils ein Ausgangsport angewandt. Nicht angewandte JP2-Anschlüsse werden von der EDA-Software auf einen Tri-State geschaltet. Um ein Fehlverhalten, zum Beispiel einen Kurzschluss, zu verhindern, werden die ungenutzten JP2-Ports auf GND geschaltet.

Nachdem alle Blöcke und die entsprechenden Ein- und Ausgänge miteinander verbunden sind, erfolgt die PIN-Belegung im „Altera PIN-Planner“.

	Status	From	To	Assignment Name	Value	En
1	✓		I2C_SDA	Location	PIN_AB20	Yes
2	✓		I2C_SCL	Location	PIN_AA20	Yes
3	✓		Board-Reset	Location	PIN_F1	Yes
4	✓		Board-Clock	Location	PIN_G21	Yes
5	✓		Level_7	Location	PIN_T15	Yes
6	✓		Level_6	Location	PIN_Y17	Yes
7	✓		Level_5	Location	PIN_W17	Yes
8	✓		Level_4	Location	PIN_AB17	Yes
9	✓		Level_3	Location	PIN_AA18	Yes
10	✓		Level_2	Location	PIN_AA17	Yes
11	✓		Level_1	Location	PIN_AB18	Yes
12	✓		Level_0	Location	PIN_AA19	Yes
13	✓		I2C_Reset	Location	PIN_AB19	Yes
14	✓		HB_40	Location	PIN_V7	Yes
15	✓		HB_39	Location	PIN_V6	Yes
16	✓		HB_38	Location	PIN_U8	Yes
17	✓		HB_37	Location	PIN_Y7	Yes
18	✓		HB_36	Location	PIN_T9	Yes
19	✓		HB_35	Location	PIN_U9	Yes
20	✓		HB_34	Location	PIN_T10	Yes
21	✓		HB_33	Location	PIN_U10	Yes
22	✓		HB_32	Location	PIN_R12	Yes
23	✓		HB_31	Location	PIN_R11	Yes
24	✓		HB_27	Location	PIN_U12	Yes
25	✓		HB_26	Location	PIN_R14	Yes
26	✓		HB_25	Location	PIN_T14	Yes
27	✓		HB_24	Location	PIN_AB7	Yes
28	✓		HB_23	Location	PIN_AA7	Yes
29	✓		HB_22	Location	PIN_AA9	Yes
30	✓		HB_20	Location	PIN_AB9	Yes
31	✓		HB_18	Location	PIN_V15	Yes
32	✓		HB_17	Location	PIN_W15	Yes
33	✓		HB_15	Location	PIN_U15	Yes
34	✓		HB_28	Location	PIN_T12	Yes

Abbildung 141 - PIN-Belegung des gesamten Projektes auf einem DE0 Board im Quartus II

Ein Quartus II PIN-Export der angewandten JP2-PINs für das Altera DE0 Board und DE0 NanoBoard sind im Anhang auf DVD unter der jeweiligen Boardbezeichnung hinterlegt.

Nach einer erfolgreichen Synthese wird das Programm auf den FPGA übertragen. Die Hardwareverifizierung zeigt einen sich drehenden Text „Home“ auf dem Würfel. Eine Videodemonstration befindet sich im Anhang auf DVD unter dem Namen „3D_RGB_LED_Matrix_Laufschrift_HOME“.

5.8.2 Live-Test NanoBoard 3000

Nachdem alle VHDL-Elemente erzeugt und auf der schematischen Darstellung platziert sind, müssen diese mit Hilfe von Busleitungen und Leitungen wie folgt miteinander verbunden werden (Abbildung 142).

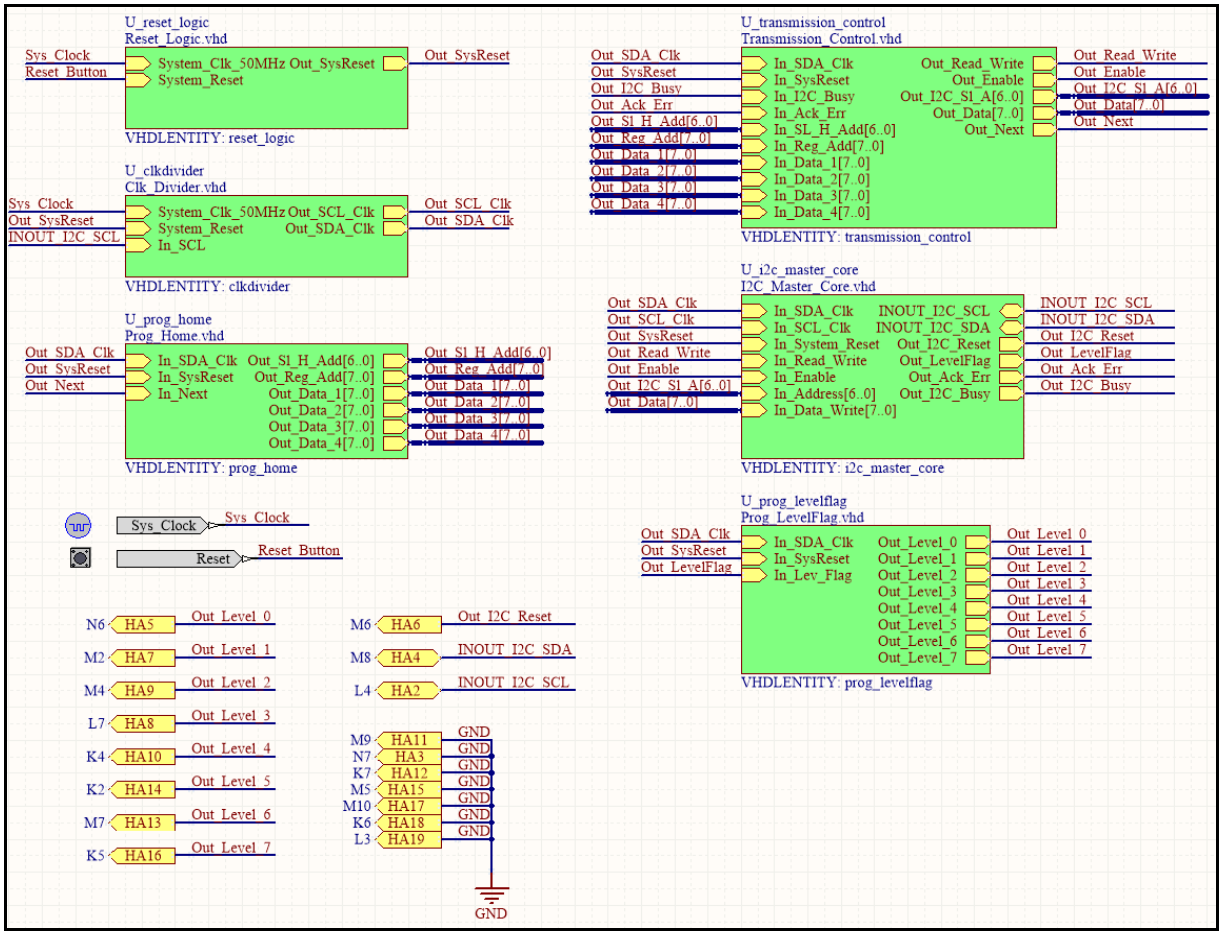


Abbildung 142 - Schematische Darstellung des gesamten Projektes im Altium-Designer

Angewandt werden zwei Eingangspins für den Board-Takt und den Taster. Der GPIO-Header A (HA) des Altium NB3K verfügt über 20-PINs. Für die I²C-Leitungen (SDA und SCL) werden bidirektionale Datenleitungen angewandt. Verbleibenden Anschlüssen wird jeweils ein Ausgangsport zugewiesen. Ungenutzte HA-Anschlüsse werden auf GND geschaltet. Im nächsten Schritt erfolgt die Übertragung auf den FPGA. Dabei wird das Projekt ausgewählt und das Programm der Reihe nach kompiliert, synthetisiert und für den FPGA übersetzt.

Nach erfolgreicher Programmierung des FPGAs wird die entsprechende Animation auf dem Cube dargestellt.

5.8.3 2-D Matrix

Zur Funktionsüberprüfung der beiden erstellten VHDL-Blöcke erfolgt ein Hardwaretest mit dem bereits vorgestellten Cyclone 4 Entwicklungsboard. Für das Teilprojekt wird eine 2D-Matrix bestehend aus 40 blauen LEDs erstellt. Auf dieser wird in Laufschrift der Text „HOME“ angezeigt. Die nachfolgende Abbildung 143 zeigt einen Überblick der Schaltung.

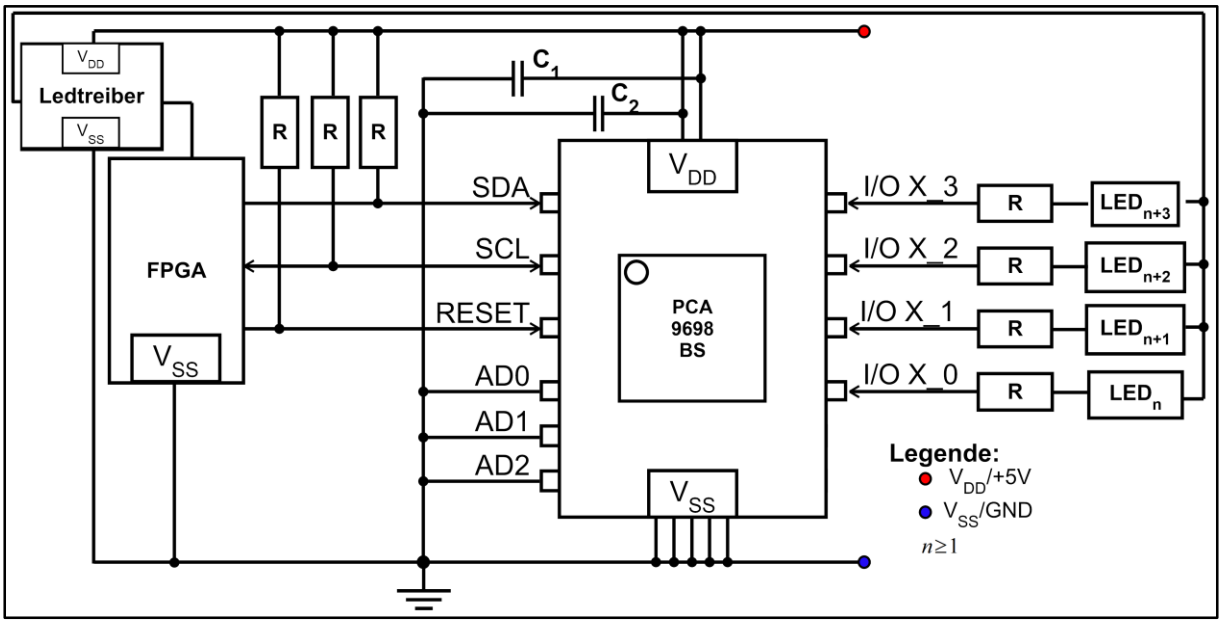


Abbildung 143 - Teilauszug eines Schaltplanes für eine 2D-LED-Matrix mit einem PCA9698BS

Die Kathoden der einzelnen LEDs werden jeweils mit einem Port des GPIO-Expanders verbunden. Eine Ansteuerung der Anoden erfolgt über eine gemeinsame Treiberschaltung, welche über einen GPIO-Anschluss eines FPGA-Entwicklungsboards gesteuert wird.

Die nachfolgende Abbildung 144 zeigt die fertige Platine der 2D-Matrix, den GPIO-Expander und alle vorhandenen Anschlüsse.

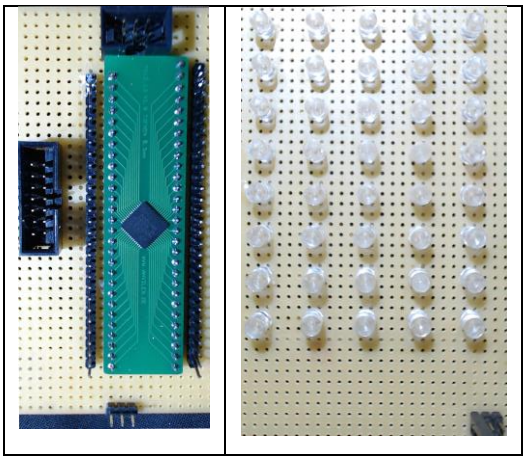


Abbildung 144 - Platine für die Ansteuerung einer 40-PIN 2D LED-Matrix

Ein Beispielschaltplan für die Umsetzung der 2D-LED-Matrix befindet sich im Anhang auf DVD unter der Bezeichnung „Schaltplan_2D_LEDMatrix.pdf“.

Die PIN-Belegungen für eine externe Stromversorgung, Ansteuerung des Expanders und die Treiberschaltung erfolgt nach Kapitel 5.3. Nachfolgend wird das Teilprojekt „2D_LED_Matrix“ in Quartus II schematisch dargestellt.

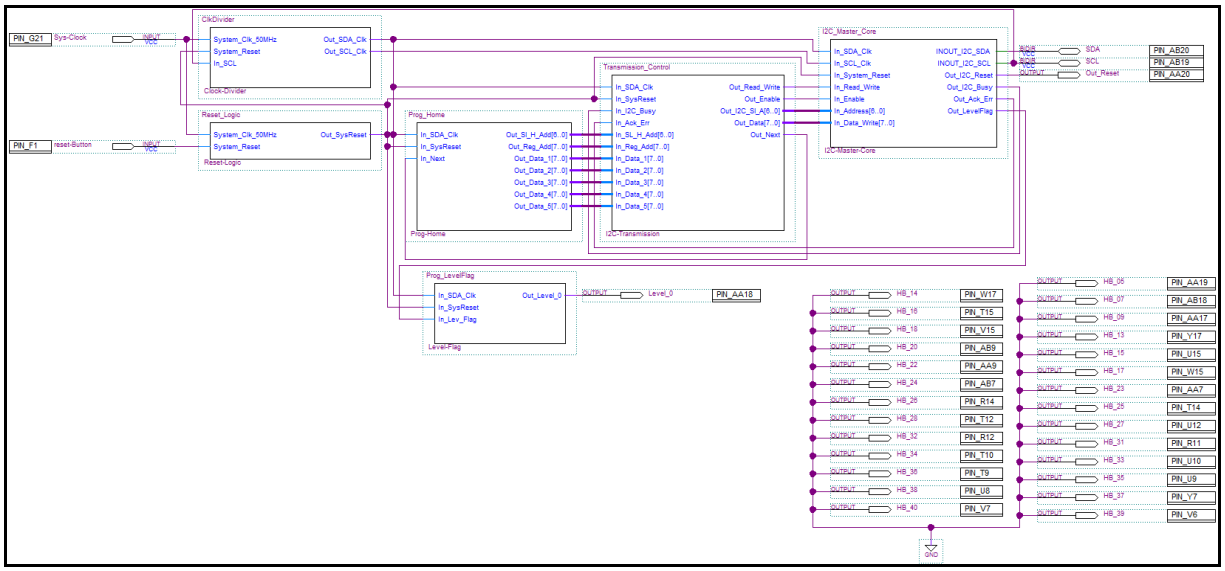


Abbildung 145 - Schematische Darstellung Projekt 2D-Matrix im Quartus II

Angesteuert wird diese Matrix mit einer Spannung von 5V. Für eine Übertragung von jeweils einem Byte auf fünf Ports (40-Bit) werden 64 Flanken benötigt.

Entsprechend wird die zu erzeugende Funktionsbeschreibung des I²C-Takts in der „Clock_Divider.vhd“ -Datei angepasst.

```
01  …
02    constant I2C_Bus_Clk : integer := 960;
03  …
```

Quellcode 59 - Anpassung des des I²C-SDA- und SCL-Taktteilers für eine 2D-Matrix im Taktteiler

Für eine angemessene Laufschrift wird ein I²C-Takt von 960Hz gewählt. Anschließend erfolgt die Umsetzung der zu übertragenden Bits in der „Prog_Home.vhd“ -Datei.

Auf der Matrix werden die LEDs so angesteuert, dass in einer Laufschrift „HOME“ angezeigt wird.

```
01  …
02  type Data_HomeAni_Arr_0 is array (natural range <>,
03    natural range <>) of std_logic_vector (7 downto 0);
04  signal Data_HomeAni_0:
05  Data_HomeAni_Arr_0 (0 to 30, 4 downto 0):=
06  (
07    ("11111111", "11111111", "11111111", "11111111",
08     "11111111"),
09    ("01111111", "01111111", "01111111", "01111111",
10     "01111111"),
11    …
```

Quellcode 60 - 3D-Array für eine 40-PIN 2D LED-Matrix im Prog_Home

Dafür wird ein dreidimensionales Array initialisiert. In diesem werden die zu übertragenden Bytes beschrieben.

```
01  …
02  case Chip_Cnt_2 is
03    when 0 =>
04      if (Chip_Cnt = Sek_Takt) then
05        Chip_Cnt_2  := Chip_Cnt_2+1;
06      else
07        Chip_Cnt_2    := 0;
08      end if;
09      Y_Coord         := 0;
10  …
```

Quellcode 61 - Array-Zeigervariable für eine 2D Matrix im Prog_Home

Anschließend wird eine `Chip_Cnt_2` -Zustandsmaschine erstellt (02). Mit jeder steigenden Flanke des System-Taktes wird der aktuelle Wert der `Chip_Cnt_2` -Variable überprüft und es wird in den entsprechenden Zustand gesprungen (03). In jedem Zustand findet eine Überprüfung der `Chip_Cnt` -Variable statt (04).

Hat diese einen maximalen Wert erreicht, wird die Variable `Chip_Cnt_2` um eins erhöht und es wird mit der nächsten steigenden Flanke in den entsprechenden Zustand gesprungen (05). Ist der Wert noch nicht erreicht, wird der aktuelle Zustand erhalten (07). Jeder Zustand entspricht einem Intervall, in welchem eine Zeile des Arrays übertragen wird (09).

```
01  …
02  if (In_Next = '1') then
03  …
04    Data_Home_1 <= Data_HomeAni_0 (Y_Coord,0);
05  …
```

Quellcode 62 - Funktionsbeschreibung bei einer Aufforderung neuer Daten vom Core für eine 2D Matrix im Prog_Home

Mit jeder neuen Datenaufforderung vom Master-Core werden die Koordinaten entsprechend gesetzt (04).

```
01  …
02    Out_Data_1 <= Data_Home_1;
03  …
```

Quellcode 63 - Setzen der Ausgangsports für eine 2D Matrix im Prog_Home

Eine Übertragung der Daten erfolgt außerhalb eines Prozesses, indem die Ausgangssignale gesetzt werden (02). Die Verifizierung der erzeugten Codes erfolgt auf der Hardware.

Anschließend wird das Teilprojekt kompiliert, synthetisiert, die Ports entsprechend gesetzt und auf den FPGA übertragen. Die Funktionsweise des erzeugten Codes befindet sich als Demonstrationsvideo im Anhang auf DVD unter der Bezeichnung „2D_Matrix_HOME_Laufschrift_HOME.mp4“.

5.9 Fehlersuche des erzeugten Master-Core-Moduls

Die Komplexität der erzeugten Module kann zu einem unkoordinierten Ablauf führen. Mit der erfolgreichen Synthese der Übertragungskontrolle und des Master-Cores kann ein Live-Test auf der Hardware erfolgen. Für diesen werden die folgenden Hilfsvariablen in der Datei „Transmission_Control.vhd“ erstellt:

```
01  …
02  constant Cache_Reg_Add : std_logic_vector
03    (6 downto 0):="0100000"
04  constant Cache_Reg_Port: std_logic_vector
05    (7 down 0)   := "10011000";
06  constant Cache_Data_1, Cache_Data_2, Cache_Data_3,
07    Cache_Data_4 : std_logic_vector
08    (7 downto 0):="11111111"
09  ;
10  …
```

Quellcode 64 - Initialisierung von Testvariablen in der Übertragungskontrolle

Die Konstante `Reg_Add` wird mit einer I²C-IC-Adresse initialisiert (02,03). Eine Bitübertragung erfolgt ab dem ersten Chip-Register (04,05). Dabei werden vier Register des Expanders angewandt, deren Ports fortlaufend mit einem Byte bestehend aus logischen „1“en beschrieben werden. Nach der Initialisierung der Variablen erfolgt eine Anpassung in den einzelnen Zustandsabfolgen.

```
01  …
02  I2C_Sl_A <= Cache_SL_H_Add;
03  Data     <= Cache_Reg_Add;
04  …
05  Data     <= Cache_Data_1;
06  …
07  Data     <= Cache _Data_2;
08  …
09  Data     <= Cache _Data_3;
10  …
11  Data     <= Cache_Data_4;
12  …
```

Quellcode 65 - Anpassung der zu übertragenden Vektoren in der Übertragungskontrolle

Existieren keine Fehler in der Übertragungskontrolle und im Master-Core, so leuchten alle angesteuerten LEDs auf dem Cube.

6 Fazit und Anmerkungen

Diese Dokumentation verdeutlicht, dass eine LED eine kompakte Bauweise, eine gute spektrale Darstellung und eine hohe Energieeffizienz gegenüber herkömmlichen Leuchtmitteln aufweist. Durch ihre unterschiedlichen Bauformen und Eigenschaften bieten sich diese für den Bau eines RGB-LED-Cubes an.

Das Projekt veranschaulicht die Vorteile einer Verwendung von VHDL mit einem FPGA. Auf einem FPGA können Systeme gehostet werden sowie Funktionsbeschreibungen erfolgen. Moderne FPGAs, zum Beispiel die Altera Cyclone 5 Generation, verfügen über einen Dual Core ARM V7 CPU kombiniert mit einem FPGA. Dies ermöglicht die Verwendungen fertiger Embedded Systeme, zum Beispiel Google-Android, auf der ARM-Architektur. Eine Hardwarebeschreibung kann parallel erfolgen und muss nicht speziell für einen ARM angepasst werden.

Aus der Dokumentation ist ersichtlich, dass ein FPGA eine begrenzte Anzahl an Logikelementen besitzt. Dies erfordert Kenntnisse in Logik, Schaltwerken und einer Hardwarebeschreibungssprache, zum Beispiel VHDL, zur Erzeugung von einer effektiven ressourcenschonenden Anwenderlogik. Parallele Prozesse und eine flankengenaue Erkennung ermöglichen komplexe Aufgabenabarbeitungen und HIGH-Performance gegenüber herkömmlichen Controllern.

Das Projekt zeigt, dass unter Einhaltung von VHDL-Standards eine Kompatibilität der erzeugten VHDL-Module zu unterschiedlichen FPGA-Herstellern gewährleistet ist. Durch die verschiedenen Portvergaben und Eigenschaften von FPGAs und deren Entwicklungsboards ist es erforderlich, dass Ein- und Ausgangsports spezifisch konfiguriert werden. Aus Kompatibilitätsgründen sind diese PIN-Belegungen bei der Projektplanung ebenfalls zu berücksichtigen.

Das NanoBoard 3000 präsentiert sich gemeinsam mit dem Altium-Designer als eine gute Plattform für den Einstieg in die FPGA-Programmierung, fordert allerdings viel Geduld während der Installation, Lizenzierung und Einarbeitung. Der Hersteller Altium

bietet umfangreiche Videodemonstrationen, Hilfestellungen, Anwendungsbeispiele und Workshops an, welche meist mit anderen Programmiersprachen und grafischen Programmelementen kombiniert sind. Daraus resultierend sind Grundkenntnisse im Bereich der VHDL-Entwicklung und im Umgang mit dem Altium-Designer empfehlenswert.

Als Kommunikationsschnittstelle zwischen einem FPGA-Entwicklungsboard und dem Cube dient jeweils ein GPIO-Port. Dies ermöglicht eine Softwareumsetzung zur Ansteuerung des Cubes auf unterschiedlichen Systemen. So kann der Würfel über einen Personal-Computer oder ein Embedded System, z.B. Rasberry Pi, betrieben werden. Desweiteren bietet die Programmiersprache Java eine entsprechende GPIO-API an.

Eine Verwendung von MOSFETs als Schalter in Form einer LED-Treiberschaltung dient für eine Spannungsversorgung einer Ebene. Deren Ansteuerung erfolgt jeweils über einen GPIO-PIN. Für eine GPIO-Erweiterung dient ein I²C-Zwei-Draht-Bussystem. Jeweils 32 Säulen einer Farbe werden auf einem PORT einer I²C-Erweiterung auf GND geschaltet. Helligkeit und Farbwerte können über eine Software realisiert werden, um die Hardwarekosten zu reduzieren.

Das standardisierte I²C-Zwei-Draht-Bussystem besitzt einen statischen Aufbau. Dies ermöglicht dessen Verwendung auf unterschiedlichen Plattformen unter der Voraussetzung, dass ein I²C-Master-Core existiert. Proprietäre Lösungen für einen FPGA werden meist vom Hersteller zur Verfügung gestellt. Dies ermöglicht eine schnellere Implementierung und Anwendung für eine Anwenderlogik. Die Realisierung/Verwendung eines in VHDL erzeugten Master-Cores benötigt gegebenenfalls weniger Logikelemente und kann beliebig erweitert oder auf einen Anwendungsbereich, zum Beispiel einer Leseoperation, optimiert werden.

Auf einem FPGA können mehrere I²C-Master-Core-Module und entsprechende Programmabläufe umgesetzt werden. Parallele Projekte können ebenfalls auf einem FPGA in ein vorhandenes Projekt implementiert werden. Durch diesen Prozess werden die Hardwarekosten reduziert, Leistungen optimiert und die Ressourcen effektiv angewandt.

Das erstellte Projekt bietet viele Möglichkeiten einer Fortsetzung. Auf dem Cube können unterschiedliche 3D-Farbanimationen dargestellt werden, welche auf 512 RGB-LEDs begrenzt sind. Der Master-Core bietet die Möglichkeit Daten aus einem I²C-Baustein auszulesen. Unter Betrachtung der Modifizierung und Anpassungen können I²C-Bausteine, zum Beispiel Temperatursensor, Echtzeituhr und Barometer ausgelesen, deren Daten interpretiert und auf dem Cube angezeigt werden. Für eine Farb- und/oder Helligkeitsänderung kann ein separates VHDL-Modul erzeugt werden, welches mit weiteren Logikmodulen z.B. Anwenderlogik und der Ebensteuerung verbunden ist. Dies ermöglicht eine Darstellung von akustischen Spektralbereichen.

Abkürzungsverzeichnis

CLK	Clock
2D	Zweidimensional
3D	Dreidimensional
Ack	Acknowledge
ADC	Analog to Digital Converter
API	Application Programming Interface
ARM	Advanced RISC Machines
ASIC	Application Specific Integrated Circuits
CAM	Computer Aided Manufacturing
CAN	Controller Area Network
CLB	Configurable Logic Block
DAC	Digital to Analog Converter
DE	Development and Education Board
DSP	Digital Signal Processor
DVD	Digital Versatile Disc
EDA	Electronic Design Automation
EEPROM	Electrically Erasable Programmable Read only Memory
FPGA	Field Programmable Gate Array
GND	Ground
GPIO	General Purpose Input/Output
HA	Header A
HB	Header B

HDL	Hardware Description Language
Hz	Hertz
I/O	Input/Output
I²C	Inter Integrated Circuit
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IGFET	Insulated Gate Field-Effect Transistor
LED	Light Emitting Diode
MHz	Megahertz
MISO	Master Input Slave Output
MOSFET	Metal Oxide Semiconductor Field-Effect Transistor
MOSI	Master Out Slave Input
NB3K	NanoBoard 3000
NXP	Next eXPerience
PKW	Personenkraftwagen
PWM	Pulsweitenmodulation
QFN	Quad Flat No Lead
R	Resistor
R/W	Read/Write
RGB	Red Green Blue
RST	Reset
RTC	Real Time Clocks
SCK	Takt
SCL	Serial Clock Lin
SDA	Serial Data Line

SMD	Surface Mounted Device
SPI	Seriel Peripheral Interface
SS	Slave Select
TWI	Two Wire Interface
TXT	Textdatei
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Referenzen

[1] Elektrische Leiter
B130; Hütte Das Ingenieur-Wissen
H. Czochos und M.Hennecke
33. Auflage, Spinger-Verlag
ISBN: 978-3-540-71851-2

[2] Elektrische Leiter
Wikipedia Foundation Inc.
http://de.wikipedia.org/wiki/Leiter_(Physik)
Stand 01.07.2014

[3] Elektrischer Nichtleiter
Wikipedia Foundation Inc.
http://de.wikipedia.org/wiki/Isolator
Stand 01.07.2014

[4] Halbleiter
Wikipedia Foundation Inc.
http://de.wikipedia.org/wiki/Halbleiter
Stand 01.07.2014

[5] Halbleiter
Das ELKO
http://www.elektronik-kompendium.de/sites/bau/1502121.htm
Stand 01.07.2014

[6] Dotierung
Das ELKO - Kompendium
http://www.elektronik-kompendium.de/sites/grd/1007251.htm
Stand 01.07.2014

[7] Metall-Oxid-Feldeffekt-Transistor
Wikipedia Foundation Inc.
http://de.wikipedia.org/wiki/Metall-Oxid-Halbleiter-Feldeffekttransistor
Stand 01.07.2014

[8] MOSFET
G717; Hütte Das Ingenieur-Wissen
H. Czochos und M.Hennecke
33. Auflage, Spinger-Verlag
ISBN: 978-3-540-71851-2

[9] MOSFET
Elemente der angewandten Elektronik
Prof. Dr.Ing.- Böhmer, Prof. Dr. Ing.- Ehrhardt, Prof. Dr. Ing. - Oberschelp
15. Auflage; Vieweg Verlag
ISBN: 9783834801241

[10] Luminesenzenzdiode
Vogel Business Media GmbH & Co. KG
http://www.elektronikpraxis.vogel.de/opto/articles/249214/
Stand 01.07.2014

[11] Zhaga
Zhaga Consortium
http://www.zhagastandard.org/
Stand 01.07.2014

[12] LED
Wikipedia Foundation Inc.
http://de.wikipedia.org/wiki/Leuchtdiode
Stand 01.07.2014

[13] LED

G176; Hütte Das Ingenieur-Wissen

H. Czochos und M.Hennecke

33. Auflage, Spinger-Verlag

ISBN: 978-3-540-71851-2

[14] LED

Das ELKO - Kompendium

http://www.elektronik-kompendium.de/sites/bau/0201111.htm

Stand 01.07.2014

[15] SPI

Wikipedia Foundation Inc.

http://de.wikipedia.org/wiki/Serial_Peripheral_Interface

Stand 01.08.2014

[16] SPI Hersteller

FH München; Fachbereich Elektrotechnik und Informationstechnik

http://www.netzmafia.de/skripten/hardware/SPI-Hersteller.html

Stand 01.08.2014

[17] SPI

Motorola Spezifikation

http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf

Stand 01.08.2014

[18] SPI

Universität Koblenz Landau

http://www.uni-koblenz.de/~physik/informatik/MCU/SPI.pdf

Stand 01.08.2014

[19] NXP

NXP Semiconductors

http://www.nxp.com/

Stand 01.08.2014

[20] NXP I²C

NXP Semiconductors

http://www.nxp.com/products/interface_and_connectivity/i2c/

Stand 01.08.2014

[21] I²C Logo

Wikipedia Foundation Inc.

http://de.wikipedia.org/wiki/Datei:I2c_logo.svg

Stand 01.08.2014

[22] Mikrocontroller

AVR-Mikrocontroller-Lehrbuch: Einführung in die Welt der AVR-RISC-Mikrocontroller am Beispiel des ATmega8

Roland Walter, Denkholz Buchmanufaktur

Auflage: 3., aktualisierte Aufl. (1. Juli 2009)

ISBN-13: 978-3981189445

[23] I²C

Wikipedia Foundation Inc.

http://de.wikipedia.org/wiki/I%C2%B2C

Stand 01.08.2014

[24] I²C Zustände

Roboternetz Forum; Frank Brall

http://rn-wissen.de/wiki/index.php/I2C

Stand 01.08.2014

[24] I²C Spezifikation

NXP Semiconductors Revision 6 - April 2014

http://www.nxp.com/documents/user_manual/UM10204.pdf

Stand 01.08.2014

[25] I²C Anwendungshinweis

NXP Semiconductors

http://www.nxp.com/documents/application_note/AN10216.pdf

Stand 01.08.2014

[26] I²C PCA9698 Datenblatt

NXP Semiconductors

http://www.nxp.com/documents/data_sheet/PCA9698.pdf

Stand 01.08.2014

[27] FPGA

Wikipedia Foundation Inc.

http://de.wikipedia.org/wiki/Field_Programmable_Gate_Array

Stand 31.08.2014

[28] VHDL

Wikipedia Foundation Inc.

http://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language

Stand 01.09.2014

[29] IEEE

Institute of Electrical and Electronics Engineers

https://www.ieee.org/index.html

Stand 01.09.2014

[30] Synthesetool
Wikipedia Foundation Inc.
http://de.wikipedia.org/wiki/Synthesetool
Stand 01.09.2014

[31] Altera
Altera Corporation
http://www.altera.com/
Stand 01.09.2014

[32] Quartus II
Altera Corporation
http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html
Stand 01.11.2014

[33] ModelSim
Altera Corporation
http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html
Stand 01.11.2014

[34] EDA
Wikipedia Foundation Inc.
http://de.wikipedia.org/wiki/Electronic_Design_Automation
Stand 01.09.2014

[35] HDL
Wikipedia Foundation Inc.
http://de.wikipedia.org/wiki/Hardwarebeschreibungssprache
Stand 01.09.2014

[36] VHDL

Kompaktkurs VHDL: mit vielen anschaulichen Beispielen
Paul Molitor; Jörg Ritter
Oldenbourg Wissenschaftsverlag (19.12.2012)
ISBN-13: 978-3486712926

[37] VHDL

VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme
Jürgen Reichardt; Bernd Schwarz
Oldenbourg Wissenschaftsverlag; Auflage: aktualisierte Auflage (5.12.2012)
ISBN-13: 978-3486716771

[38] DE0 Board

Altera Corporation
http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=364
Stand 01.09.2014

[39] DE0 Benutzerhandbuch

Altera Corporation
http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=364&FID=0c266381d75ef92a8291c5bbdd5b07eb
Stand 01.09.2014

[40] DE0 NanoBoard

Altera Corporation
http://www.altera.com/education/univ/materials/boards/de0-nano/unv-de0-nano-board.html
Stand 01.09.2014

[41] DE0 NanoBoard Benutzerhandbuch
Altera Corporation
ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Boards/DE0-Nano/DE0_Nano_User_Manual.pdf
Stand 01.09.2014

[42] Altium-Designer
Altium Limited
http://products.live.altium.com/
Stand 12.09.2014

[43] Altium NanoBoard 3000
Altium Limited
http://nb3000.altium.com/intro.html
Stand 12.09.2014

[44] Belegarbeit Altium NanoBoard 3000
Inbetriebnahme des Altium NanoBoard 3000 und Dokumentation ausgewählter FPGA-Beispiele
Florian Sauerbrei, B.Eng. ; Oliver Heumos B.Eng.
Hochschule Merseburg; Fachbereich: Informatik und Kommunikationssysteme
19.03.2013

[45] Altium NanoBoard 3000 Logo 2014
Altium Limited
http://www.altium.com
Stand 12.09.2014

[46] Altium NanoBoard 3000 Frontansicht
Altium Limited
http://www.altium.com/files/envision/sep09/NB3000front.jpg
Stand 12.09.2014

[47] Altium NanoBoard 3000 Datenblatt
Altium Limited
http://nb3000.altium.com/PDFs/NB3000%20Product%20Data%20Sheet_de.pdf
Stand 12.09.2014

[48] ISE WebPack
Xilinx Corp.
http://www.xilinx.com/support/download/index.htm
Stand 12.09.2014

[49] Place and Route Tools
Altium Limited
http://wiki.altium.com/display/ADOH/Xilinx+Place+and+Route+Tools+Configuration
Stand 12.09.2014

[50] Xilinx Reset
Xilinx Corp.
http://www.xilinx.com/support/documentation/white_papers/wp272.pdf
Stand 12.09.2014

[51] LED 5mm diffus
KT-electronic
http://www.ebay.de/itm/100-Stuck-LED-5mm-RGB-diffus-4-PIN-gem-Plus-6000mcd-/351158407507?pt=Bauteile&hash=item51c2ac1153
Stand 01.10.2014

[52] Elektrische Leitfähigkeit
Wikipedia Foundation Inc.
http://de.wikipedia.org/wiki/Elektrische_Leitf%C3%A4higkeit
Stand 01.10.2014

[53] Kupferdraht
Conrad Electronic SE
http://www.conrad.de/medias/global/ce/6000_6999/6000/6050/6054/605450_RB_00_FB.EPS_1000.jpg
Stand 01.10.2014

[54] N-Kanal-MOSFET 2N7002
Fairchild Semi
https://www.fairchildsemi.com/datasheets/2N/2N7000.pdf
Stand 01.10.2014

[55] P-Kanal-MOSFET
Vishay
http://www.vishay.com/docs/72024/72024.pdf
Stand 01.10.2014

[56] Wippschalter
highend-elektronik
http://www.ebay.de/itm/DOT-Wippschalter-Druckschalter-Illuminated-Switch-KFZ-Kippschalter-12V-/161260823848?pt=Elektromechanische_Bauelemente&var=&hash=item258be4e528
Stand 01.10.2014

[57] OpenCore
Open Source Hardware Community
http://opencores.org/
Stand 01.10.2014

[58] Dashboard
Digi-Key Corporation
http://www.eewiki.net
Stand 01.10.2014

[59] Dashboard

Digi-Key Corporation

http://www.digikey.de/?WT.srch=1&WT.medium=cpc&WT.mc_id=IQ6359341

7-VQ2-g-VQ6-53842680020-VQ15-1t1-VQ16-c

Stand 01.10.2014

[60] I²C Master-Core Version Scott Larson

Digi-Key Corporation

http://www.eewiki.net/display/LOGIC/I2C+Master+(VHDL)

Stand 01.10.2014

[61] I²C Master-Core Version 1.0 Scott Larson

Digi-Key Corporation

https://www.eewiki.net/download/attachments/10125324/i2c_master_v1_0.vh

d?version=1&modificationDate=1403293989363&api=v2

Stand 02.01.2014

Anhang

DVD :

- ✓ Materialliste.pdf
- ✓ Gehaeuse_Konstruktion.pdf
- ✓ Schablone_Gitternet.pdf
- ✓ Schablone_Bohrung.pdf
- ✓ Schaltplan_2D_LEDMatrix.pdf
- ✓ Schaltplan_3D_LEDMatrix.pdf

- ✓ I²C
  - Spezifikation
  - Datenblatt PCA9698
  - VHDL-Master-Core von Scott Larson Version 1.0, 2.0 und 2.1

- ✓ Beispiel Projekte Videos
  - 2D_Matrix_HOME Laufschrift „HOME“
  - 3D RGB-LED-Matrix Laufschrift „HOME“
  - 3D RGB-LED-Matrix Ebenentest
  - 3D RGB-LED-Matrix Farbspiel/PWM
  - 3D RGB-LED-Matrix Farbspiel/PWM 2
  - 3D_RGB_LED_Matrix_SwipeTest
  - LED-Blink-Rest/Takt-Test
  - RGB-LED-Blink-Test/Takt-Test

- ✓ Altera DE0 Board Cyclone III
  - Datenblatt

- ✓ Altera Quartus II Version 13.1 Web-Edition DE0 Board Projekte
  - Takt/Reset-Test (LED_Blinc)
  - RGB-LED (RGB_Led_Blinc)
  - 2D_LED_Matrix (2D_LED_Matrix)
  - 3D RGB-LED-Matrix 3D Effekt (LED_Cube_Swipe_Test)
  - 3D RGB-LED-Matrix Laufschrift „HOME“ (LED_Cube_Swipe_Home)
  - 3D RGB-LED-Matrix Ebenentest (LED_RGB_Cube_Levels)
  - 3D RGB-LED-Matrix Farbspiel/PWM (LED_Cube_Colour_Test)

- ✓ Altera DE0 NanoBoard Cyclone IV
  - Datenblatt

- ✓ Altium Quartus II Version 13.1 Web-Edition DE0 NanoBoard Projekte
  - Takt/Reset-Test (LED_Blinc)
  - RGB-LED (RGB_Led_Blinc)
  - 2D_LED_Matrix (2D_LED_Matrix)
  - 3D RGB-LED-Matrix 3D Effekt (LED_Cube_Swipe_Test)
  - 3D RGB-LED-Matrix Laufschrift „HOME“ (LED_Cube_Swipe_Home)
  - 3D RGB-LED-Matrix Ebenentest (LED_RGB_Cube_Levels)
  - 3D RGB-LED-Matrix Farbspiel/PWM (LED_Cube_Colour_Test)

- ✓ Altium-Designer NanoBoard 3000
  - Datenblatt
  - User I/O Headers
  - Schematic NanoBoard3000XN Xilinx-FPGA

- ✓ Altium-Designer NanoBoard 3000 Projekte
  - Takt/Reset-Test (LED_Blinc)
  - RGB-LED (RGB_Led_Blinc)
  - 2D_LED_Matrix (2D_LED_Matrix)
  - 3D RGB-LED-Matrix 3D Effekt (LED_Cube_Swipe_Test)
  - 3D RGB-LED-Matrix Laufschrift „HOME“ (LED_Cube_Swipe_Home)
  - 3D RGB-LED-Matrix Ebenentest (LED_RGB_Cube_Levels)
  - 3D RGB-LED-Matrix Farbspiel/PWM (LED_Cube_Colour_Test)

Anhang 1:

Beispielschaltung PCA9698