Bernburg
Dessau
Köthen

Hochschule Anhalt
Anhalt University of Applied Sciences

emw
Fachbereich
Elektrotechnik, Maschinenbau
und Wirtschaftsingenieurwesen

Belegarbeit

Software-Hardware-Codedesign

Alexander Wilke

Vorname Nachname

Elektro- und Informationstechnik
Fernstudium

Studiengang

Thema:

Entwurf und Implementierung eines 3-Achsen-Beschleunigungssensors

Prof. Dr. Michael Brutscheck

Prüfer/in

30.01.2015

Abgabe am

Selbstständigkeitserklärung

Hiermit erkläre ich, dass die Arbeit selbständig verfasst, in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt wurde und keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet wurden.

Mosbach, 27.01.2015	Alexander Wilke
Ort, Datum	Unterschrift/en der/des Studierenden

Kurzfassung

Die Arbeit beschreibt die Modellbildung und Implementierung eines über die Programmiersprache C programmierbaren Mikrocontrollers (Nios II) innerhalb eines FPGAs. Hierfür wird das *terasIC* Entwicklungsboard DE0Nano mit dem FPGA *Cyclone® IV EP4CE22F17C6N* verwendet. Der auf dem Entwicklungsboard befindliche 3D-Beschleunigungssensor ADXL345 wird hierzu mittels einer I2C-Verbindung über C angesprochen und ausgewertet. Zur Anzeige der Beschleunigungswerte in X-, Y- und Z-Richtung kommt ein eigens entwickeltes LED-Erweiterungsmodul zum Einsatz. In dieser Arbeit sind die notwendigen Schritte zur Modellbildung, Implementierung und Programmierung (C und VHDL) beschrieben. Des Weiteren wird auf aufgetretene hard- und softwareseitige Probleme eingegangen, um Nachfolgearbeiten zu erleichtern.

Inhaltsverzeichnis

1 Motivation und Zielsetzung

Neben der vorlesungsbedingten Vorgabe zur Durchführung dieser Arbeit stand das persönliche Interesse an der Verbindung eines FPGAs mit einem in C programmierbaren Mikrocontrollers im Vordergrund.

1.1 Einleitung in die Thematik

In dieser Arbeit werden keine allgemeinen und umfassenden Grundlagen zur Modellbildung eines Mikrocontrollers innerhalb eines FPGAs zusammengetragen. Des Weiteren werden auch keine Grundlagen in den Programmiersprachen C und VHDL beschrieben. Speziell für letzteres sei auf entsprechende Literatur verwiesen.

Die vorliegende Arbeit umfasst gezielt die nötigen Schritte zur Realisierung der Aufgabenstellung. Zur Einführung in die Thematik sei auf die Vorlesung von Herrn Prof. Dr. Brutscheck und dessen Skript [Bru14] sowie auf das Handbuch zum DE0Nano [Ter13] verwiesen, in welchen Beispiele zur Umsetzung aufgeführt sind.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist die Modellbildung und Implementierung eines über die Programmiersprache C programmierbaren Mikrocontrollers (Nios II) innerhalb eines FPGAs. Hierfür wird das *terasIC* Entwicklungsboard DE0Nano mit dem FPGA *Cyclone® IV EP4CE22F17C6N* verwendet. Der auf dem Entwicklungsboard befindliche 3D-Beschleunigungssensor ADXL345 wird hierzu mittels einer I2C-Verbindung über C angesprochen und ausgewertet. Zur Anzeige der Beschleunigungswerte in X-, Y- und Z-Richtung wird ein eigens entwickeltes LED-Erweiterungsmodul eingesetzt.

Im Folgenden sind die notwendigen Schritte zur Modellbildung, Implementierung und Programmierung (C und VHDL) beschrieben.

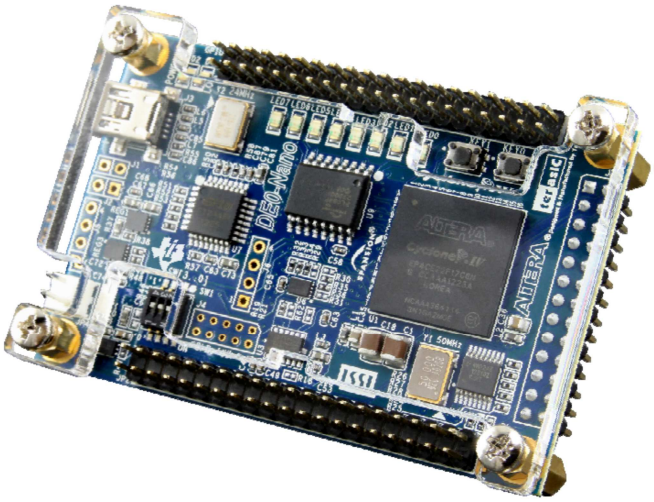


Abbildung 1.1: Das DE0Nano Entwicklungsboard, [Ter13, S. 5]

2 Nios II

Zur Implementierung eines Mikrocontrollers innerhalb des FPGAs müssen zunächst im Zuge einer Modellbildung die späteren Funktionen des Mikrocontrollers zugewiesen werden. Dies erfolgt über das in Qaurtus II befindliche Programm Qsys. Auf Basis der festgelegten Funktionen werden anschließend im VHDL-Code in Quartus II die externen Verbindungen (Ein- und Ausgänge) zugewiesen. Nach Einbindung aller über Qsys erzeugten Bibliotheken sowie der Pin-Zuordnung im Pin-Planner wird das Programm final kompiliert und anschließend auf den FPGA aufgespielt. Nachfolgend werden die einzelnen Schritte näher erläutert.

2.1 QSYS

Über Qsys wird die Modellbildung des späteren Mikrocontrollers durchgeführt, um dessen spätere Funktionen und Eigenschaften festzulegen. Das finale Qsys Modell mit den Verbindungen (Connections) ist in *Anhang A* aufgeführt. Die Funktionen der einzelnen Komponenten werden im Folgenden beschrieben.

Abbildung 2.1: Startfenster von Qsys, [Qua13]

- clk_50
  Zur Takterzeugung wird standardmäßig eine Taktquelle vorgegeben, welche auf 50 MHz festgelegt und später im Pin-Planner dem 50 MHz Quarz des DE0Nano zugewiesen wird.

- nios2_cpu
  Der Nios-II-Kern beschreibt den Mikrocontroller selbst, über welchen der spätere Befehlsaustausch zum FPGA und damit verbundener Peripherie stattfindet.

- onchip_memory_cpu
  Zur Implementierung des Mikrocontrollers und des im Nachfolgenden erstellen C-Code innerhalb des FPGAs wird Speicher benötigt, welcher mit vorgesehen wird.
  Als Speicherplatz wurden 20400 Bytes gewählt.

- jtag_uart
  Der JTAG-UART Core wird benötigt, um eine serielle Verbindung zum FPGA/Nios-II-System herstellen zu können. Hierüber erfolgt die spätere Programmierung. Eine separate RS232-Verbindung wird daher nicht benötigt.

- key
  Über diesen 2 Bit breiten IO werden im Folgenden die beiden Taster (KEY[0] u. KEY[1]) des DE0Nano als INPUT angebunden.

Signal Name	FPGA Pin No.	Description	I/O Standard
KEY[0]	PIN_J15	Push-button[0]	3.3V
KEY[1]	PIN_E1	Push-button[1]	3.3V

Abbildung 2.2: KEY-Übersicht, [Ter13, S.14]

- switch
  Über den 4 Bit breiten IO werden im Folgenden die vier Schalter (DIP-Switch[0] – DIO-Switch[3]) des DE0Nano als INPUT angebunden.
  Hinweis: Im späteren Programm sind diese funktionslos.

Signal Name	FPGA Pin No.	Description	I/O Standard
DIP Switch[0]	PIN_M1	DIP Switch[0]	3.3V
DIP Switch[1]	PIN_T8	DIP Switch[1]	3.3V
DIP Switch[2]	PIN_B9	DIP Switch[2]	3.3V
DIP Switch[3]	PIN_M15	DIP Switch[3]	3.3V

Abbildung 2.3: Switch-Übersicht, [Ter13, S.15]

- led
  Über den 8 Bit breiten IO werden im Folgenden die acht LEDs (LED[0] – LED[7]) des DE0Nano als OUTPUT angebunden.

Signal Name	FPGA Pin No.	Description	I/O Standard
LED[0]	PIN_A15	LED Green[0]	3.3V
LED[1]	PIN_A13	LED Green[1]	3.3V
LED[2]	PIN_B13	LED Green[2]	3.3V
LED[3]	PIN_A11	LED Green[3]	3.3V
LED[4]	PIN_D1	LED Green[4]	3.3V
LED[5]	PIN_F3	LED Green[5]	3.3V
LED[6]	PIN_B1	LED Green[6]	3.3V
LED[7]	PIN_L3	LED Green[7]	3.3V

Abbildung 2.4: LED-Übersicht, [Ter13, S.14]

- gpio
  Über den 32 Bit breiten IO wird im Folgenden der Expansion Header GPIO-0 des DE0Nano als bidirektionaler IO angebunden (GPIO_00 – GPIO_31). Hierüber wird das spätere LED-Erweiterungsmodul angesteuert (siehe Kapitel 3.4).
  Wichtig: Um im späteren C-Code Ausgänge einzeln schalten zu können siehe Kapitel 5.7.

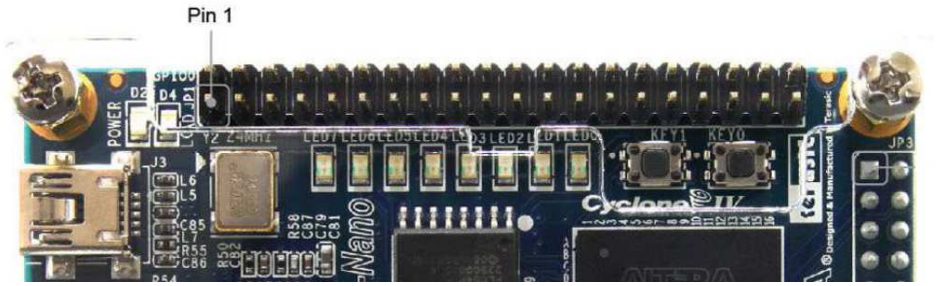


Abbildung 2.5: Ausschnitt des GPIO-0 Expansion Header, [Ter13, S.18]

Zur Pinbelegung des GPIO-0 siehe Abbildung 2.6.

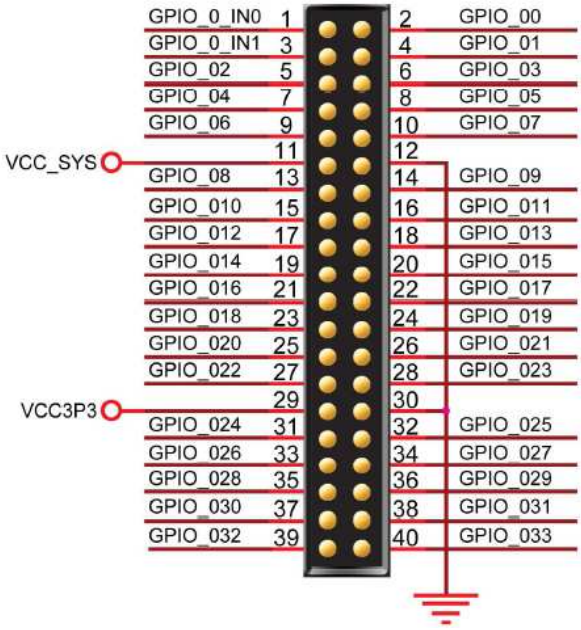


Abbildung 2.6: Pinbelegung des GPIO-0 Expansion Header, [Ter13, S.18]

- i2c
  In der Standardinstallation von Quartus II existiert kein I^2C-Core. Daher wurde der frei verfügbare I^2C-Core von opencores eingebunden.
  Download unter: http://www.alterawiki.com/wiki/I2C_(OpenCores)

  Der I^2C-Core liefert die nötigen C-Bibliotheken zur späteren Kommunikation über die I^2C-Verbindung mit dem Beschleunigungssensor ADXL345.

  Die notwendigen Schritte zur Einbindung des I^2C-Cores sind in Abschnitt 5.1 aufgeführt.

- CS_n
  Über den 1 Bit breiten IO wird im Folgenden der Chip-Select Eingang des ADXL345 als OUTPUT angebunden.

- uart
  Zur Implementierung einer seriellen Schnittstelle für weitere Kommunikationswege (außerhalb der Nios-II-Console) wird der uart-core mit eingebunden. Für Einsatzmöglichkeiten siehe Kapitel 4.
  Einstellungen: Baud rate: 9600, Parity: none, Data bits: 8, Stop bits: 1

Das fertige Qsys-Modell muss final selbst kompiliert werden, damit alle nötigen Bibliotheken mit den festgelegten Signalnamen in den Projektordner kopiert werden. Diese Bibliotheken werden für die spätere Kompilierung des Haupfiles in Quartus II benötigt.
Zur Erzeugung der Bibliotheken siehe Abschnitt 5.2.

2.2 VHDL Quartus II

Um eine Verbindung des Qsys-Modells mit realen IOs und Signalen des FPGAs zu schaffen, wird in Quartus II ein neues VHDL-File angelegt. Der komplette VHDL-Code ist in Anhang B aufgeführt.
Das Programm selbst lässt sich in vier Abschnitte unterteilen.

1. Einbindung relevanter Bibliotheken
2. Initiierung benötigter Signale (als *in*, *out* oder *inout*)
   Hinweis: An dieser Stelle die Signalnamen des FPGA verwenden (siehe auch Abschnitt 5.6)
3. Einbindung der im Qsys-Modell festgelegten Signale.
   Hinweis: Qsys selbst liefert hierzu den nötigen HDL-Code. Siehe Abschnitt 5.3.
4. Zuweisung der unter 2. initiierten Signalen mit den und 3. festgelegten Signalen aus Qsys.
   Hinweis: Generell ist zu beachten, dass die durch 3. vorgegebene Signalrichtung nicht geändert wird (siehe Abschnitt 5.4).

Das VHDL-File beschreibt somit lediglich die Signalvernetzung des Qsys-Modells mit Signalen des FPGAs. Weitere VHDL-Logik ist (in diesem Projekt) nicht enthalten.

2.3 Kompilierung / Synthese

Zur erfolgreichen Kompilierung sind die nachfolgenden Schritte zwingend der Reihe nach durchzuführen:

1. Einbindung der erzeugten Qsys-Bibliotheken (siehe Abschnitt 5.5)
2. Einbindung der DE0Nano Assignments (siehe Abschnitt 5.6) zur Zuordnung der Signalnamen an die Pins des FPGAs.
3. Start der Kompilierung.

Mit erfolgreicher Kompilierung kann das Programm auf den FPGA gespielt werden.

2.4 Bespielen des DE0Nano

Das Nios-II-System kann auf zwei Arten auf den FPGA gespielt werden.

1. Über die .sof-Datei, welche beim Kompilieren (siehe Abschnitt 2.3) mit erzeugt wird.
   Die .sof-Datei wird jedoch nur temporär auf dem FPGA gespeichert. Wird dieser von der Stromversorgung getrennt muss der FPGA erneut bespielt werden.
2. Über die .jic-Datei. Zur Erzeugung siehe [Bru14, S. 122].
   Die .jic-Datei wird erst nach dem Reboot (kurzzeitiges abziehen des FPGAs von der Stromversorgung) in den Speicher geschrieben.

Für beide Programmier-Dateien muss der Programmer (siehe Abbildung 2.7) gestartet und die gewünschte Datei über „*Add File…*“ ausgewählt werden. Mit Betätigung von „*Start*“ wird das Programm auf den FPGA geschrieben.
Hinweis: An dieser Stelle ist noch kein C-Programm im Speicher des Nios-II-Systems, dies muss nachfolgend programmiert werden. Siehe hierzu Kapitel 3.

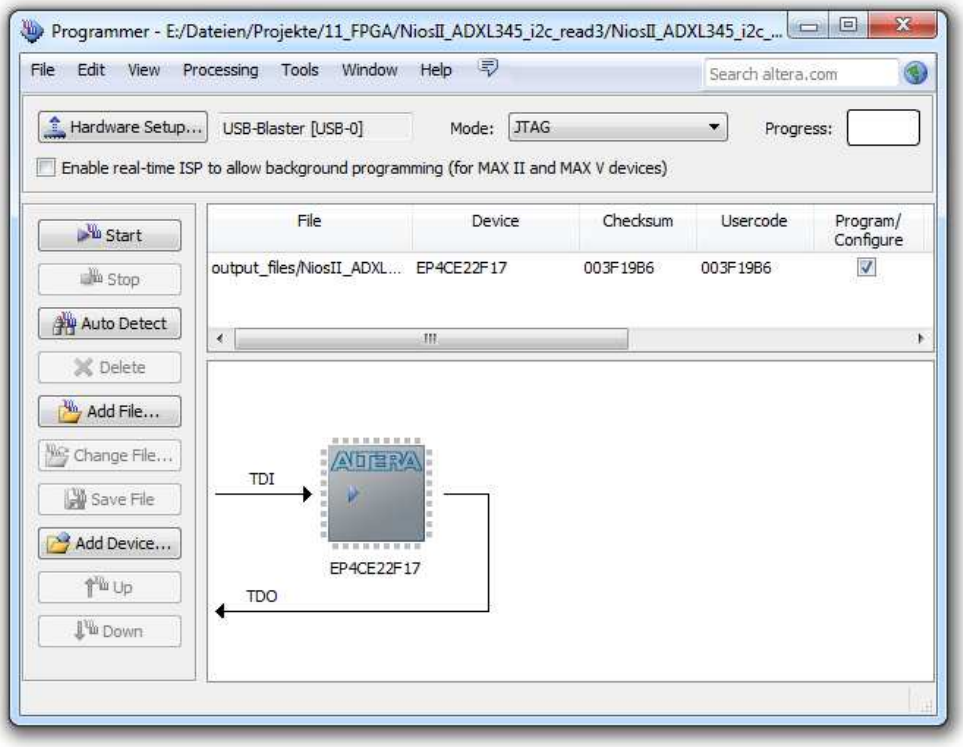


Abbildung 2.7: Programmer [Qua13]

3 C-Programmierung mit Eclipse

In dieser Dokumentation wird darauf verzichtet, den kompletten Quellcode zu erläutern bzw. diesen im Anhang wiederzugeben. Auf der im Anhang angefügten CD ist dieser enthalten und kommentiert. Die wesentlichen Programmbestandteile werden jedoch im Folgenden näher betrachtet.

3.1 I²C

Die c-Bibliotheksfunktionen aus „i2c_opencores.c“ sowie „i2c_opencores.h“ [Her03] werden zulasten der Übersichtlichkeit direkt in das Hauptprogramm übernommen und nicht als #include, um bei der eigentlichen Programmierung nicht zwischen den Dateien hin und her springen zu müssen. Gerade bei der Einarbeitung in die Funktionsweise erwies sich dies als vorteilhaft.
Folgende Funktionen, mit kurzen Beschreibungen, entstammen [Her03]:

(1) **void I2C_init**(alt_u32 base, alt_u32 clk, alt_u32 speed);
Hierüber wird die I2C-Schnittstelle initialisiert.
base: Ist die Adresse des I²C-Kerns aus Abschnitt 2.1. Diese ist system.h in der Variablen I2C_BASE gespeichert
clk: Ist der Takt des Nios-II-Systems [Angabe in Hz]. Dieser wurde im Quellcode in der Variablen „**#define** i2c_clk 50000000“ gespeichert.
speed: Ist der Clock-Takt der I²C Verbindung [Angabe in Hz]. Dieser wurde im Quellcode in der Variablen „**#define** i2c_speed 100000“ gespeichert.

(2) **int I2C_start**(alt_u32 base, alt_u32 add, alt_u32 read);
Hierüber wird die Kommunikation zum Device (ADL345) gestartet. Wird zum Lesen und Schreiben gleichermaßen zu Beginn benötigt.
base: Ist die Adresse des I²C-Kerns aus Abschnitt 2.1. Diese ist in system.h in der Variablen I2C_BASE gespeichert.
add: Ist die Adresse des ADXL345. Gemäß [AnD10, S. 17] ist dies der Hex-Wert 0x1D.
read: Gemäß [Her03, S. 16 f.] ist der Wert für die Variable read zum Lesen 1 und zum Schreiben 0.

(3) alt_u32 **I2C_read**(alt_u32 base, alt_u32 last);
Die I2C_read-Funktion wird zum Lesen eines 8 Bit (1 Byte) Wertes verwendet und dient zudem als Stop-Funktion am Ende eines Lese- und Schreibbefehls.
base: Ist die Adresse des I²C-Kerns aus Abschnitt 2.1. Diese ist in system.h in der Variablen I2C_BASE gespeichert.
last: Gibt an, ob es sich (speziell beim Lesen mehrerer Bytes) um den letzten Lese-Befehl handelt (last=1) oder nicht (last=0).

(4) alt_u32 **I2C_write**(alt_u32 base, alt_u8 data, alt_u32 last);
Die I2C_write-Funktion wird zum Schreiben eines 8 Bit (1 Byte) Wertes verwendet.
base: Ist die Adresse des I²C-Kerns aus Abschnitt 2.1. Diese ist in system.h in der Variablen I2C_BASE gespeichert
data: Beschreibt den zu sendenden 8 Bit-Datensatz.
last: Gibt an, ob es sich (speziell beim Schreiben mehrerer Bytes) um den letzten Schreibbefehl handelt (last=1) oder nicht (last=0).

Folgende #defines wurden ebenfalls [Her03] entnommen:

```
(5)  #define I2C_OK (0)
(6)  #define I2C_ACK (0)
(7)  #define I2C_NOACK (1)
(8)  #define I2C_ABITRATION_LOST (2)
(9)  #define I2C_OPENCORES_INSTANCE(name, dev) extern int alt_no_storage
(10) #define I2C_OPENCORES_INIT(name, dev) while (0)
```

Im Zuge der Verwendung obiger Funktionen wurde beim Aufbau der Kommunikation festgestellt, dass der Sende- bzw. Schreibbefehl nicht alleine genutzt werden kann, um eine komplette Kommunikation abzubilden. Hierfür wurden eigene Funktionen angelegt, über welche die komplette Kommunikation zum Senden und Schreiben abgebildet wird. Der benötige Ablauf zum Senden bzw. Schreiben ist in [Her03, S. 16] ersichtlich. Hierauf beruht der Aufbau folgender eigener Funktionen.

(11) alt_u32 **I2C_read_8b**(alt_u8 reg);
Diese eigene Funktion deckt die kompletten Befehle zum Lesen eines 8 Bit-Wertes über die I²C-Schnittstelle ab(Quellcode mit Kommentaren siehe Anhang D).
reg: Beschreibt das zu lesende Register

(12) alt_u16 **I2C_read_16b**(alt_u8 reg);
Diese eigene Funktion deckt die kompletten Befehle zum Lesen eines 16-Bit-Wertes über die I²C-Schnittstelle ab (Quellcode mit Kommentaren siehe Anhang D). Das Lesen eines Registers (z.B. 0x00) liest das nächst höhere (0x01) hierbei automatisch mit aus.
reg: Beschreibt das zu lesende Register

(13) **void I2C_write_8b**(alt_u8 data, alt_u8 reg);
Diese eigene Funktion deckt die kompletten Befehle zum Schreiben eines 8 Bit-Wertes über die I²C-Schnittstelle ab(Quellcode mit Kommentaren siehe Anhang D).
data: Beschreibt den zu sendenden 8 Bit-Datensatz
reg: Beschreibt das zu beschreibende Register

Zum Test der Funktionen wurden die für die I²C-Kommunikation relevanten Ausgänge des ADXL345 am Oszilloskop (4 Kanal Oszilloskop MS0-X 3034A 350Mhz Agilent Technologies) gemessen. Hierfür wurden die Ausgänge auf eine separate Stiftleisten gelegt (siehe Abbildung 3.1).

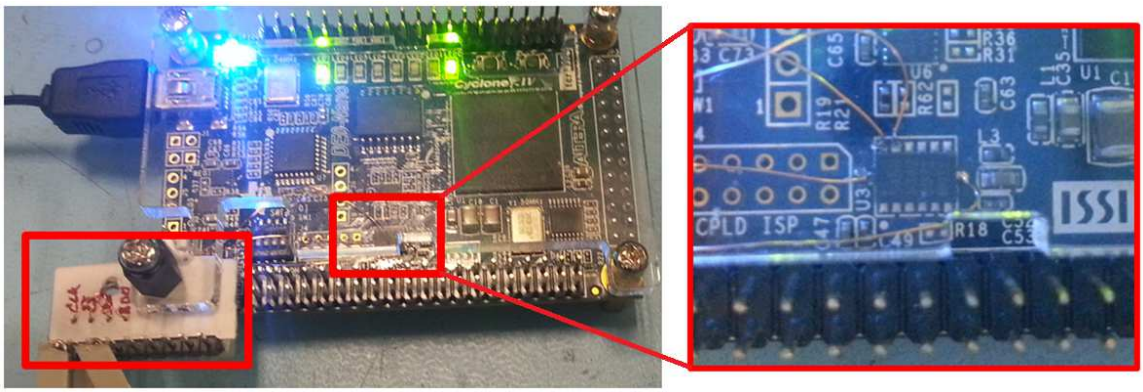


Abbildung 3.1: I²C-Kommunikationspins auf separater Stiftleiste, Eigene Darstellung

Als Beleg für die korrekte Ausführung der Kommunikation seien nachfolgende Beispiele mit Abbildungen aufgeführt. Im verwendeten Oszilloskop ist die Auswertung und Anzeige einer I^2C-Kommunikation integriert, sodass die Befehle mit entsprechenden Hex-Werten angezeigt werden. Die Belegung des Oszilloskops ist durchweg Clk (Channel 1), CS_n (Channel 2), SDA (Channel 3) und SD0 (Channel 4).

Beispiel 01: Lesen der Device-ID aus Register 0x00. Das Ergebnis ist der Hex-Wert 0xE5 [AnD10, S. 23].
Funktion: I2C_read_8b(0x00);

Erläuterung:

3AWa: Adresse des ADXL345 0x1D mit nachfolgender 0x00 als Schreibbefehl ergibt 0x3A (binär: 11101-0 → 111010).
Das „W“ deutet den führenden Write-Befehl.
Das „a“ steht für Acknowledge.

00a: Beschreibt das Senden des zu lesenden Registerwertes 0x00.
Das „a“ steht für Acknowledge.

3BRa: Adresse des ADXL345 0x1D mit nachfolgender 0x01 als Lesebefehl ergibt 0x3B (binär: 11101-1 → 111011).
Das „R“ deutet den ausgeführten Read-Befehl.
Das „a“ steht für Acknowledge.

E5~a: E5 steht für den vom ADXL345 gelieferten Hexwert.
Das „a“ steht für Acknowledge.

Abbildung 3.2: Lesen der Device-ID, Eigene Darstellung

Beispiel 02: Beschreiben des Data-Format-Registers 0x31 mit dem Wert 0x0B.
Funktion: I2C_write_8b(0x0B, 0x31);

Erläuterung:
3AWa: Adresse des ADXL345 0x1D mit nachfolgender 0x00 als Schreibbefehl ergibt 0x3A (binär: 11101-0 → 111010).
Das „W“ deutet den führenden Write-Befehl.
Das „a“ steht für Acknowledge.

31a: Beschreibt das Senden des zu beschreibenden Registerwertes 0x31.
Das „a“ steht für Acknowledge.

0Ba: 0B steht für den zu schreibenden Hexwert.
Das „a“ steht für Acknowledge.

Abbildung 3.3: Beschreiben des REG_DATA_FORMAT-Registers, Eigene Darstellung

Beispiel 03: Überprüfung von Beispiel 02. Einlesen des Wertes im Data-Format-Registers 0x31. Das Ergebnis ist der Hex-Wert 0x0B.
Funktion: I2C_read_8b(0x31);

Erläuterung:
3AWa: Adresse des ADXL345 0x1D mit nachfolgender 0x00 als Schreibbefehl ergibt 0x3A (binär: 11101-0 → 111010).
Das „W“ deutet den führenden Write-Befehl.
Das „a“ steht für Acknowledge.

31a:	Beschreibt das Senden des zu lesenden Registerwertes 0x31. Das „a“ steht für Acknowledge.
3BRa:	Adresse des ADXL345 0x1D mit nachfolgender 0x01 als Lesebefehl ergibt 0x3B (binär: 11101-1 → 111011). Das „R“ deutet den ausgeführten Read-Befehl. Das „a“ steht für Acknowledge.
0B~a:	0B steht für den vom ADXL345 gelieferten Hexwert. Das „a“ steht für Acknowledge.

Abbildung 3.4: Lesen des Wertes im REG_DATA_FORMAT-Register, Eigene Darstellung

Mit Hilfe der eigenen Funktionen ist eine einfache Kommunikation zum Lesen und Schreiben des ADXL345 ermöglicht.

3.2 delay Funktion

Analog zum Programmieren mit anderen Mikrocontrollern (z.B. ATMega-Reihe) wurde eine Funktion erzeugt, welche definiert eine Pausenfunktion im Millisekundenbereich durchführt. Basis der Funktion ist die in Eclipse vorhandene „usleep“ Funktion, welche Pausen im µ-Sekundenbereich durchführt. Der Aufbau der Funktion ist in Anhang E ersichtlich.

(14) **void _delay_ms** (**int** msec);
msec**:** Zahl zur Angabe der Wartezeit in Millisekunden.
Hinweis: Die zeitliche Ungenauigkeit der Funktion durch die vorhandene for-Schleife wird vernachlässigt.

3.3 ADXL345

Mit der Implementierung der I²C-Funktionen aus Abschnitt 3.1 sind die notwendigen Bedingungen gegeben, um die gewünschten Beschleunigungswerte des ADXL345 auslesen zu können.

Grundlegend muss vorab der Chip-Select-Pin (s. CS_n aus Abschnitt 2.1) auf logisch 1 gesetzt werden:

```
(15) IOWR_ALTERA_AVALON_PIO_DATA(CS_N_BASE, 0x01);
```

Hinweis: Dies ist in den Beispielen aus Abschnitt 3.1 bereits der Fall.

Des Weiteren muss die I²C-Kommunikation zu Beginn des C-Codes initialisiert und Grundeinstellungen zur Kommunikation in die Registerwerte des ADXL345 geschrieben werden. Dies geschieht zu Beginn der main-Funktion.

```
(16) I2C_init(I2C_BASE, i2c_clk, i2c_speed);
```
Initialisiert die I²C-Verbindung mit den Angaben für clk und speed.

```
(17) I2C_write_8b((XL345_RANGE_16G | XL345_FULL_RESOLUTION),
     ADXL345_REG_DATA_FORMAT);
```
Setzt die Bits im DATA_FORMAT-Register des ADXL345 zur Operation im 16g-Betrieb mit voller Auflösung (Skalierungsfaktor 4 [AnD10, S. 26]).

```
(18) I2C_write_8b(XL345_RATE_100, ADXL345_REG_BW_RATE);
```
Setzt die Bits im BW_RATE-Register zur Operation der Messausgabe mit 100 Hz.

```
(19) I2C_write_8b(XL345_MEASURE, ADXL345_REG_POWER_CTL)
```
Der ADL345 initialisiert im Standby-Modus nach Unterbrechung der Stromversorgung, daher muss dieser durch Setzen des Measure-Bits im POWER_CTL Register gestartet werden.

```
(20) // I2C_write_8b(XL345_STANDBY, ADXL345_REG_POWER_CTL);
```
Zum Stoppen der Messung kann der ADXL345 durch Setzen des STANDBY-Bits in den Standby-Modus gesetzt. Diese Funktion ist im finalen Quellcode lediglich als Kommentarblock enthalten.

Nach der Initialisierung werden über folgende Funktionen die Beschleunigungswerte in X-, Y- und Z-Richtung eingelesen und direkt mit dem Skalierungsfaktor [AnD10, S. 26] verrechnet:

```
(21) read_val_x = I2C_read_16b(ADXL345_REG_DATAX0);
     val_x = read_val_x*scale_factor;

(22) read_val_y = I2C_read_16b(ADXL345_REG_DATAY0);
     val_y = read_val_y*scale_factor;

(23) read_val_z = I2C_read_16b(ADXL345_REG_DATAZ0);
     val_z = read_val_z*scale_factor;
```

Die Werte stehen anschließend in den Variablen val_x, val_y und val_z zur Verfügung, um beispielsweise über die Nios-II-Console mit ggf. angepasster printf-Funktion (siehe Abbildung 3.5), über die serielle Schnittstelle (siehe Abschnitt 3.5) oder intern zur Anzeige der Werte auf LEDs (siehe Abschnitt 3.4) weiter verwendet werden.

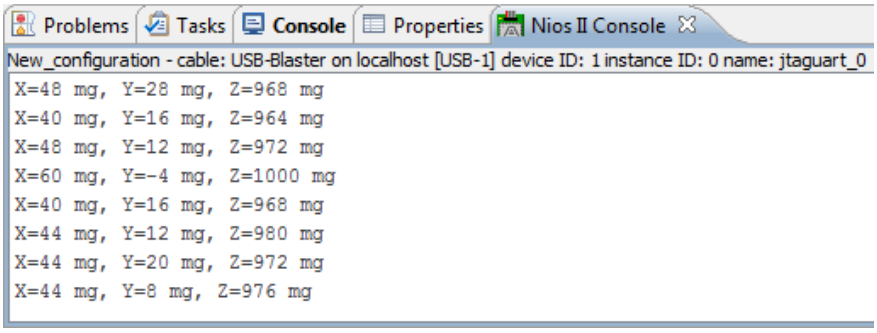


Abbildung 3.5: Beispiel eingelesener Beschleunigungswerte, Eigene Darstellung

3.4 LED-Erweiterungsplatine

Um die Beschleunigungswerte über LEDs zur Anzeige zu bringen, wurde die in Abbildung 3.6 dargestellte Platine mit je 17 LEDs für jede Raumrichtung entworfen. Auf den Schaltplan und die zugrundeliegenden Ideen zur Erstellung wird an dieser Stelle nicht eingegangen.
Zur Ansteuerung der LED-Erweiterungsplatine wird diese über ein 40-adriges Flachbandkabel per Wannenstecker mit dem GPIO (siehe gpio Abschnitt 2.1) verbunden.

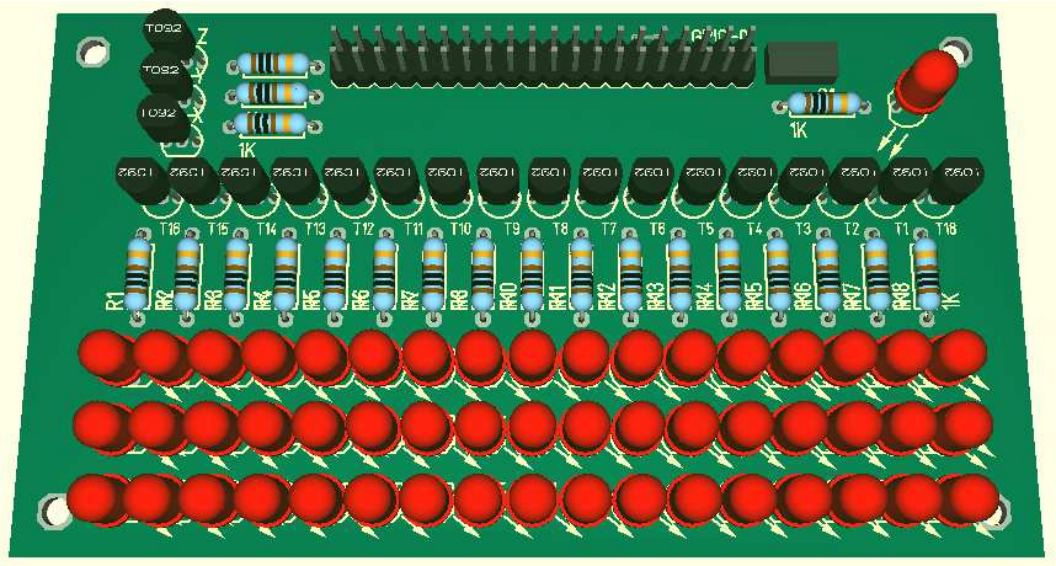


Abbildung 3.6: 3D Ansicht der LED-Erweiterungsplatine, Eigene Darstellung

Da es sich bei dem LED-Erweiterungsmodul um ein „Einzelstück“ handelt wird an dieser Stelle auf eine komplexe Beschreibung des hierfür erstellten C-Codes verzichtet. Die grundlegende Funktionsweise ist in Abschnitt 3.6 zusammengefasst.

Generell ist zu beachten, dass im C-Code bei Nutzung der GPIO-Erweiterung zwingend auf die Port-Definition geachtet werden muss, welche zu Beginn der main-Schleife erfolgen sollte.

(24) IOWR_ALTERA_AVALON_PIO_DIRECTION(GPIO_BASE, 0xFFFFFFFF);
Port des GPIO als Ausgang definiert.

Zur Übersicht seien an dieser Stelle lediglich die zur Funktionsweise des LED-Erweiterungsmoduls erstellen Grundfunktionen aufgeführt:

(25) **void reset** (**void**);
Setzt alle IOs des GPIO in Grundstellung, damit keine LED leuchtet.

(26) **void xyz_set** (**int** axis, **int** value);
Setzt in Abhängigkeit des ermittelten g-Wertes (value) die Ausgänge zur Ansteuerung der LEDs gemäß deren Raumrichtungen (axis).
axis: Aktiviert eine Achse je nach Wert (1=x-Achse; 2=y-Achse; 3=z-Achse)
value: in Abhängigkeit des Wertes werden LEDs angeschaltet

(ggf. spätere Erweiterung dieses Abschnittes)

3.5 UART

Grundidee der UART-Implementierung ist die Auswertung und Anzeige der ermittelten g-Werte mittels der Programmierumgebung LabVIEW, siehe hierzu Kapitel 4.

Da die standardmäßig durch die Implementierung der UART vorhandene Funktion „IOWR_ALTERA_AVALON_UART_TXDATA(base, data)“ lediglich ein Zeichen (data) senden kann, wurde eine eigene Funktion erstellt, über welche aus mehreren Zeichen bestehende Strings gesendet werden können (siehe Anhang F). Hierüber können die ermittelten Beschleunigungswerte über die serielle Schnittstelle geschickt werden.

(27) **void send_string**(**char** *tx_string);
Funktion zum Senden einer Zeichenkette.
Hinweis zur UART: Baud rate: 9600, Parity: none, Data bits: 8, Stop bits: 1

Abbildung 3.7 verdeutlicht die Funktionsweise über das Schnittstellenprogramm Hterm (Freeversion 0.8.1beta). Es wird je der Beschleunigungswert, welcher in seine Einzelzeichen aufgeteilt wurde, mit führendem Zeichen der Raumrichtung (x, y und z) nacheinander fortlaufend gesendet.

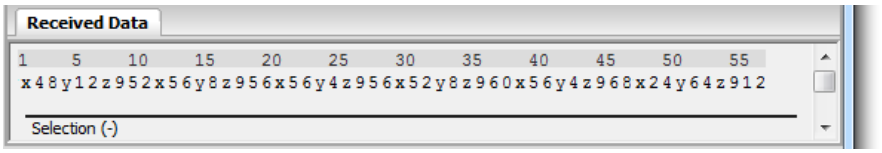


Abbildung 3.7: Auszug der Kommunikation mit Hterm, Eigene Darstellung

(ggf. spätere Erweiterung dieses Abschnittes)

3.6 Gesamtprojektübersicht

Der finale Stand des Projektes ist in Abbildung 3.8 ersichtlich. Das DE0Nano ist über das 40-adrige Flachbandkabel mit dem LED-Erweiterungsmodul verbunden. Die aktuelle Programmierung sieht vor, dass die mittleren LEDs jeder Reihe (oben für X, mittig für Y und unten für Z) „0g“ widerspiegeln und als Referenz immer aktiv sind. Je nach ermitteltem Beschleunigungswert werden ausgehend von der Mitte Richtung rechts positive g-Werte und nach links negative g-Werte von -8g bis +8g angezeigt. Jede LED steht somit für „Δ1g“. Die ermittelten Beschleunigungswerte werden im C-Programm hierfür entsprechend auf- oder abgerundet.

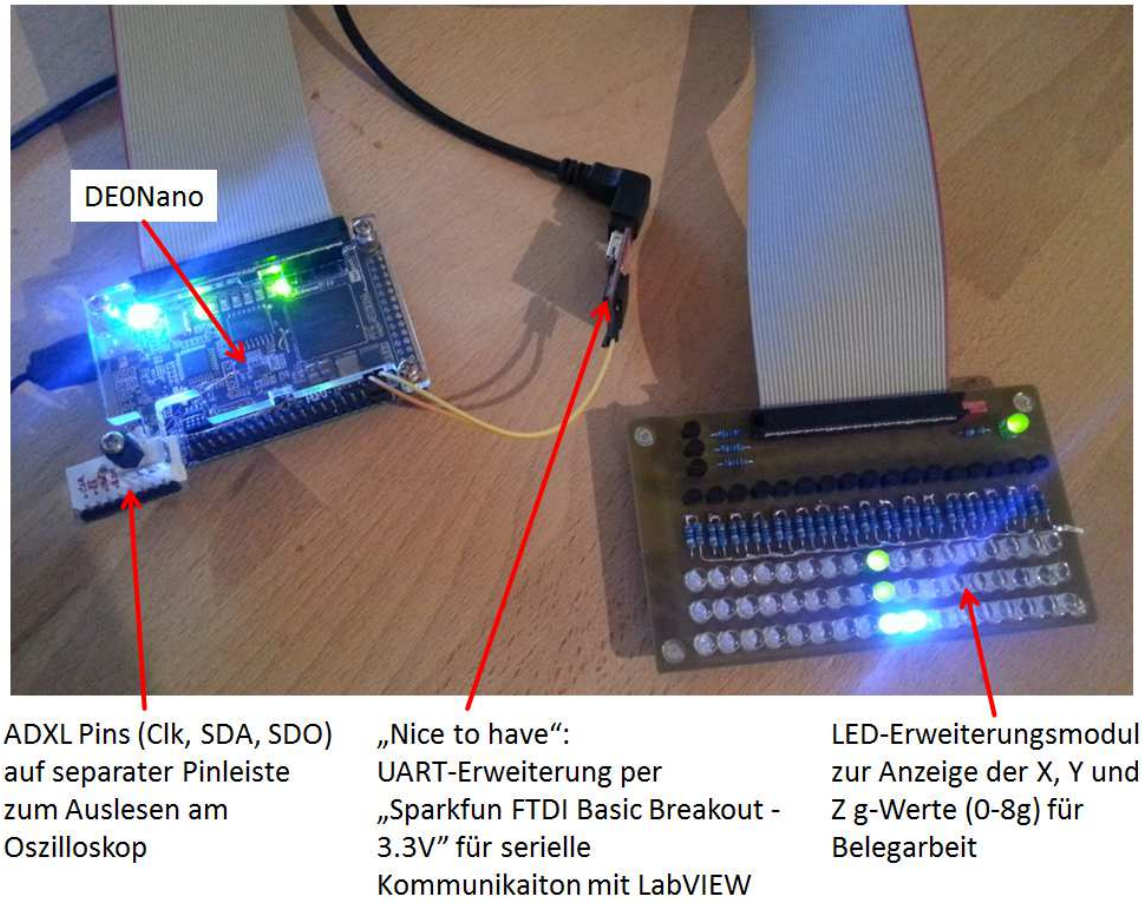


Abbildung 3.8: Projektübersicht mit Beschreibung, Eigene Darstellung

3.7 Bespielen des DE0Nano mit dem C-Programm

Um das C-Programm auf den FPGA zu spielen werden an dieser Stelle zwei Möglichkeiten aufgezeigt.

1. Das C-Programm temporär aufspielen.
   Hierbei wird das Projekt zunächst kompiliert. Dazu wird in Eclipse per Rechtsklick auf den Projektnamen „Build Project“ ausgewählt (siehe Abbildung 3.9). Wird das geschriebene C-Programm ohne Fehler kompiliert wird es über die Auswahl „Run As“ und „Nios II Hardware“ aufgespielt. Dies setzt allerdings voraus, dass die in Abschnitt 2.4 beschriebene Implementierung des Nios-II-Systems durchgeführt wurde.

Zur Programmierung mit Eclipse sei an dieser Stelle noch auf Abschnitt 5.8 bezüglich Fehlerquellen verwiesen.

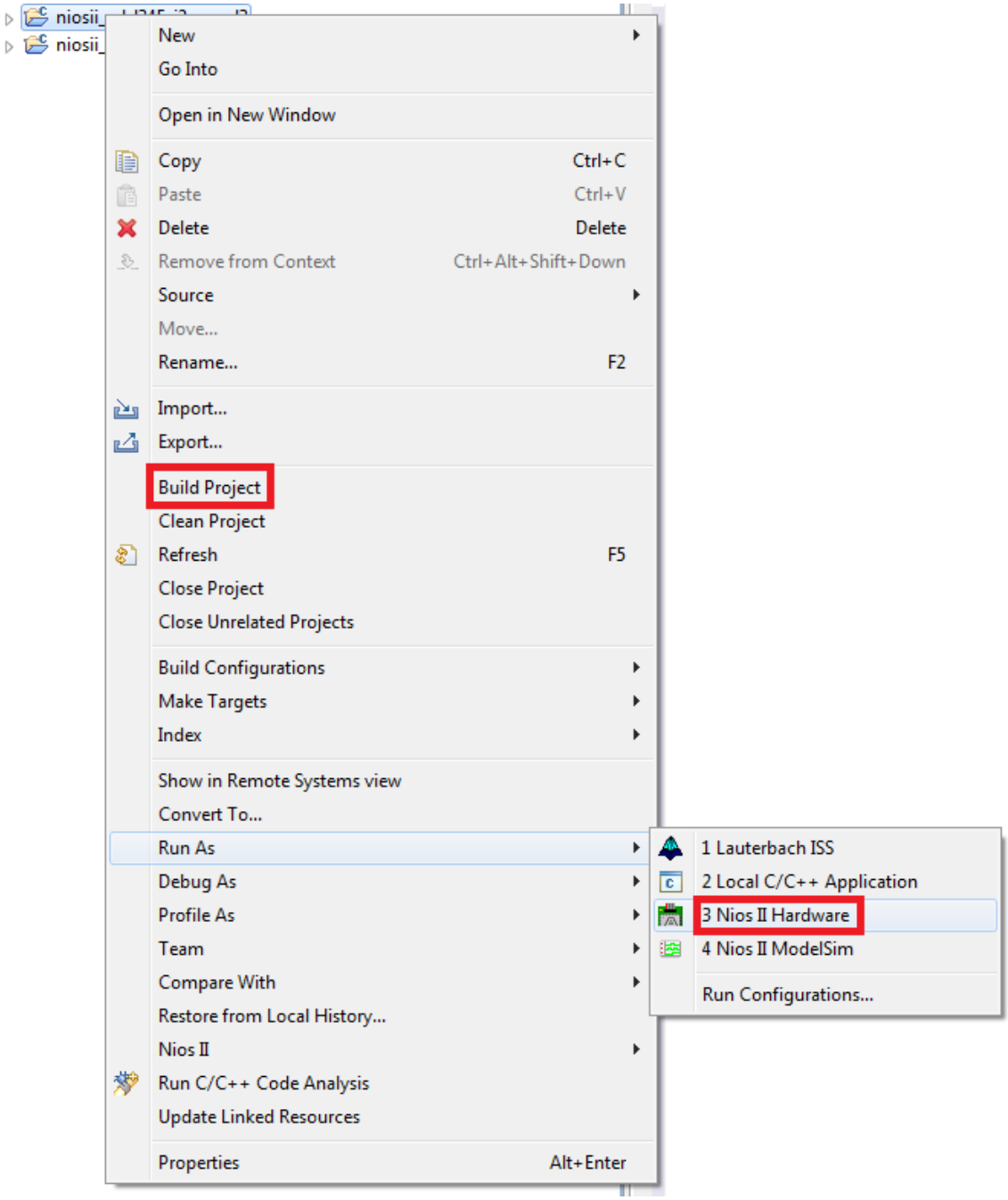


Abbildung 3.9: C-Programm temporär aufspielen, Eigene Darstellung

2. Das C-Programm dauerhaft aufspielen. Hierzu muss in den Eigenschaften der in Abschnitt 2.1 angelegten onchip_memory_cpu in Qsys das Häkchen „Initialize memory content“ gesetzt werden (siehe Abbildung 3.10). Die hierbei erwarteten Initialisierungs-Dateien (.hex, .qip und .spd) werden in Eclipse per Rechtsklick auf den Projektnamen und der Auswahl von „Make Targets“, „Build“ sowie im nachfolgend erscheinenden Fenster über die Auswahl von „mem_init_generate“ und Bestätigung über „Build“ erzeugt (siehe Abbildung 3.11).

Die Dateien müssen in Quartus II in das Projekt mit eingebunden werden. Anschließend wird das gesamte Projekt neu kompiliert und gemäß Abschnitt 2.4 auf das DE0Nano gespielt. [Dav15]

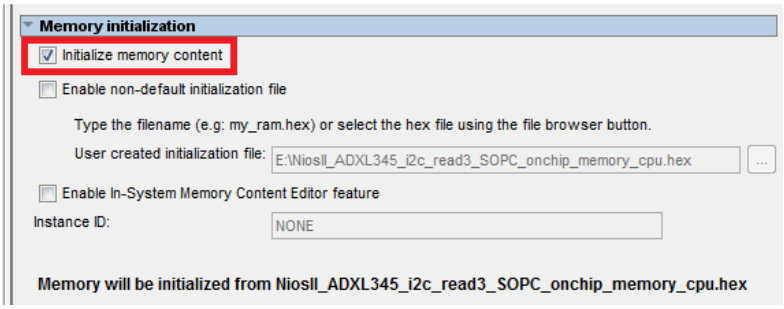


Abbildung 3.10: Initialize memory content, Eigene Darstellung

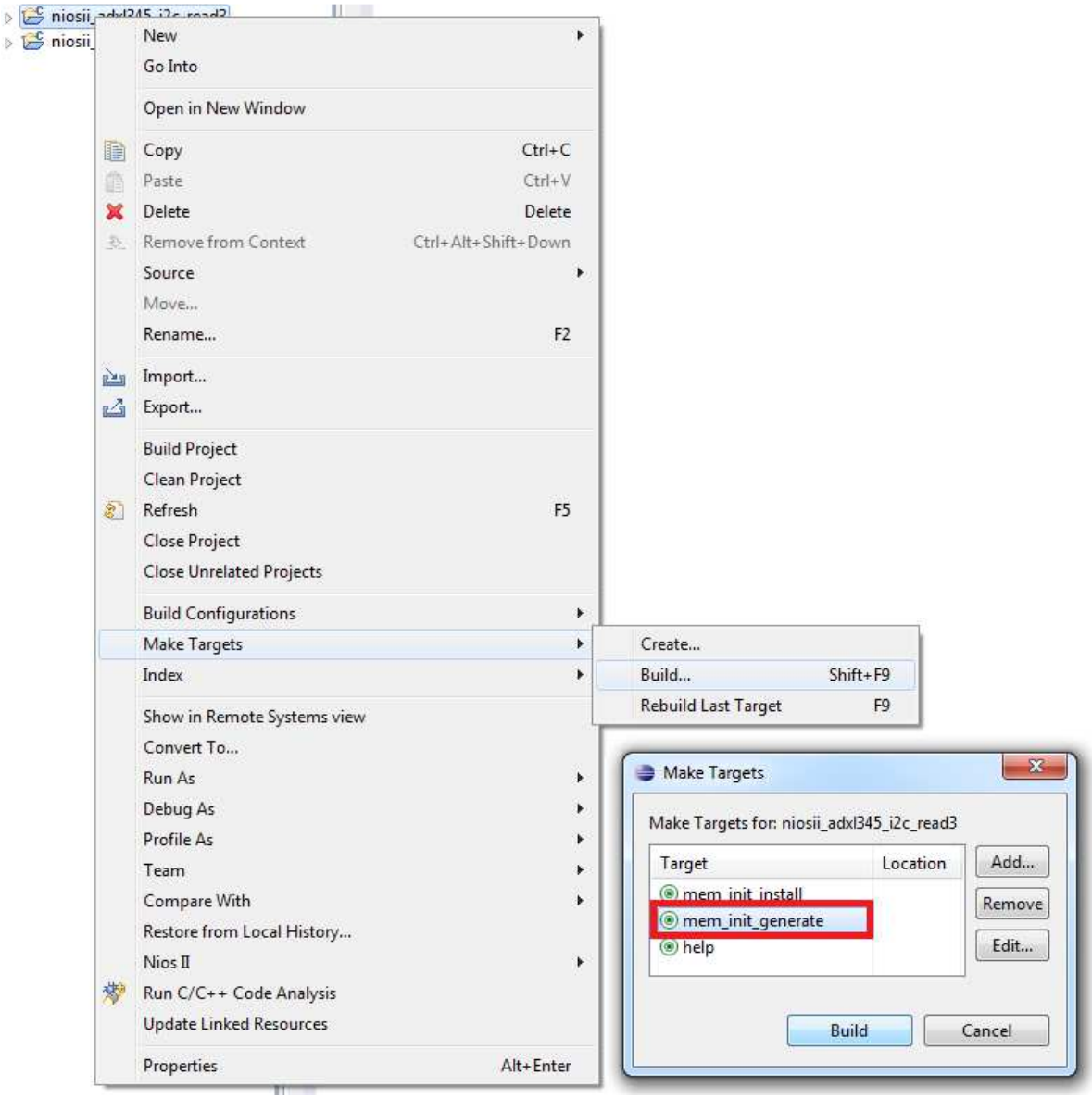


Abbildung 3.11: Initialisierungs-hex-file, Eigene Darstellung

4 LabVIEW

Dieses Kapitel erläutert das zur Auswertung der Beschleunigungswerte geschriebene LabVIEW Programm.

(ggf. spätere Erweiterung dieses Abschnittes)

5 Fehlerursachen

Die im Verlauf dieser Arbeit aufgetretenen Probleme (soft- u. hardwareseitig) und Schwierigkeiten werden für Nachfolgearbeiten in diesem Kapitel kurz zusammengefasst.

5.1 Qsys zusätzliche Module

Zur Einbindung weitere Module muss Qsys zunächst geöffnet werden. Über das Menü *„Tools\Options…“* öffnet sich das in Abbildung 5.1 ersichtliche Fenster.

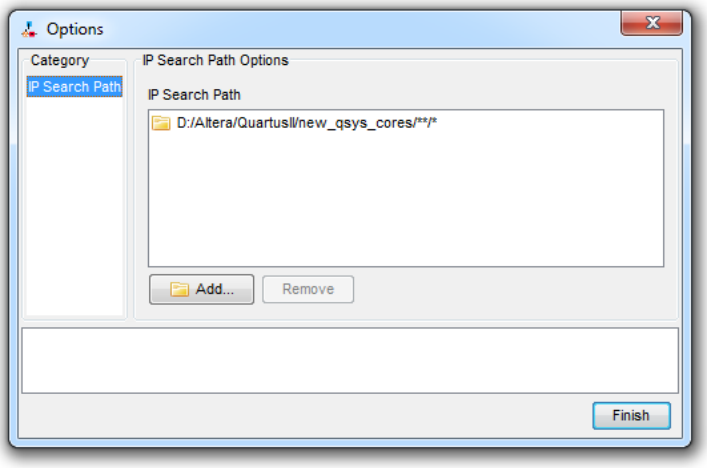


Abbildung 5.1: Qsys – Einbindung weiterer Module, [Qua13]

Über „Add…“ wird der Ordner ausgewählt, in welchem die (z.B. aus dem Internet) bezogenen Dateien liegen. Es empfiehlt sich die Erstellung eines Hauptordners für sämtliche zusätzlichen Module. Qsys aktualisiert sich nach der Auswahl selbst. Die neuen Module können anschließend über das Auswahlfenster ausgewählt und eingebunden werden. Abbildung 5.2 veranschaulicht dies am Beispiel des eingebundenen I^2C-Cores von opencores.

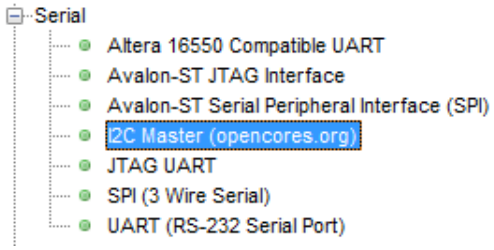


Abbildung 5.2: Eingebundener I^2C-Core, [Qua13]

5.2 Qsys Kompilierung

Zur Erzeugung der Qsys-Bibliotheken muss im Menü des Qsys-Modells unter *„Generation“* zunächst die Programmiersprache ausgewählt und anschließend die Kompilierung per Betätigung der Schaltfläche *„Generate“* gestartet werden. Eine Abbildung zur Veranschaulichung ist in Anhang C aufgeführt.

5.3 Qsys HDL Example

Im Menü des Qsys-Modells wird unter „*HDL Example*“ der in Quartus II nötige Code (in VHDL oder Verilog) bereitgestellt, um die nötige Signalzuweisung aus Abschnitt 2.2 durchführen zu können.

5.4 Qsys Signalrichtung

Die Signalrichtung in Qsys, wie durch Abschnitt 5.3 vorgegeben, darf im VHDL-Code (siehe Abschnitt 2.2) nicht verändert werden. Jede Änderung führt beim späteren kompilieren zu einer nicht umgehbaren Fehlermeldung.

5.5 Einbindung der Qsys-Bibliotheken

Zur Einbindung der über Qsys erzeugten Bibliotheken müssen in Quartus II unter „Files“ (siehe Abbildung 5.3) per Rechtsklick auf den Ordner „Files“ im nachfolgend erscheinenden Menü sämtliche .v-Dateien ausgewählt werden, welche sich in Folgenden Ordner befinden:

1. „Dateipfad“\xxx_SOPC\synthesis\
2. „Dateipfad“\xxx_SOPC\submodules\

Zudem muss die .vhdl-Datei aus Abschnitt 2.2 ebenfalls mit ausgewählt werden, wenn nicht bereits aufgeführt.

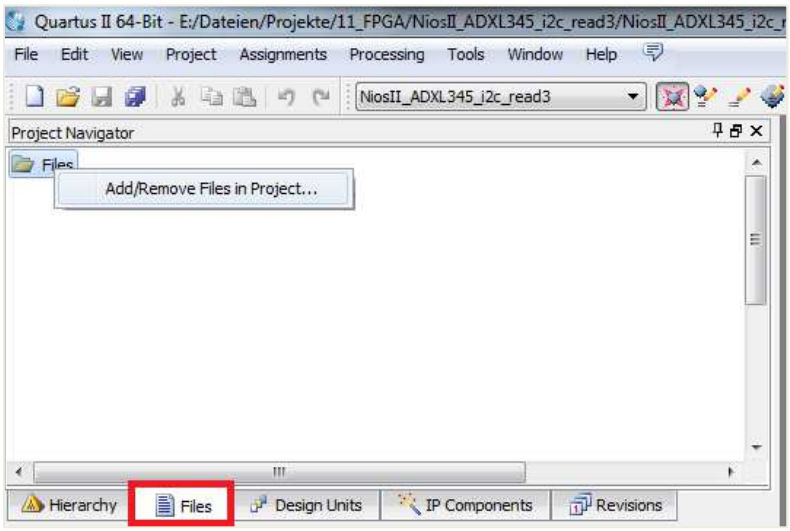


Abbildung 5.3: Einbindung von Qsys Bibliotheken, [Qua13]

Wichtig: Es ist zwingend darauf zu achten, dass die .qsys-Datei, welche das Qsys-Modell beschreibt, nicht ausgewählt wird. Dies führt gerade bei zusätzlich eingebundenen Modulen (wie z.B. I^2C von opencores) zu Fehlern.

Wenn beim kompilieren auf fehlende .v-Dateien verwiesen wird, müssen diese per Hand in einen der beiden obig genannten Ordner kopiert werden. Dies ist beispielsweise bei der Datei „timescale.v“ aus dem I^2C-Modul von opencores der Fall.

5.6 Einbindung von Assignments

Im Lieferumfang des FPGAs befindet sich auf der CD die Datei „DE0_Nano.qsf“, in welcher die Signalnamen mit Pinbelegung hinterlegt sind. Die Datei wird nach Auswahl von „Import Assignments“ aus Abbildung 5.4 ausgewählt.
Diese Namen wurden bereits beim Punkt „Initiierung benötigter Signale“ unter Abschnitt 2.2 verwendet. Durch die Einbindung der Assignments werden die Signalnamen den entsprechenden Pins des FPGAs automatisch zugewiesen.

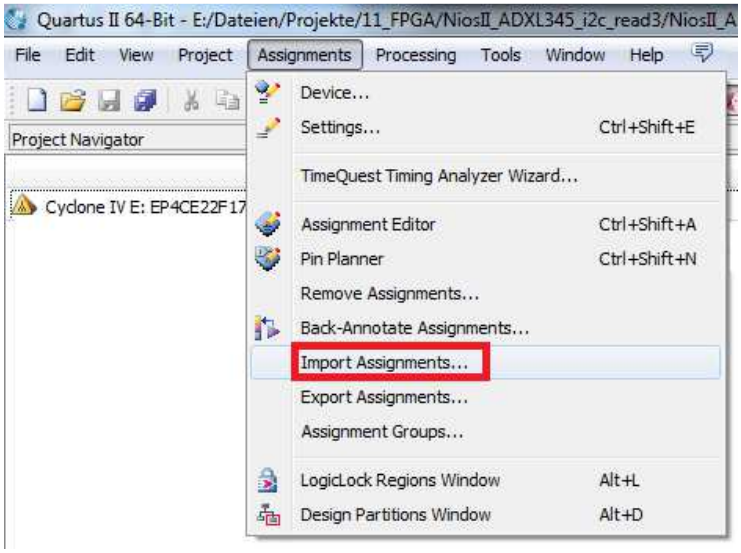


Abbildung 5.4: Einbindung von Assignments, [Qua13]

5.7 Setzen und Rücksetzen einzelner IO-Bits im C-Code

Um im späteren C-Code bequem einzelne IO-Bits setzen und rücksetzen zu können, empfiehlt sich die Aktivierung der Funktion „Enable individual bit setting/clearing“ in den Einstellungen der PIO in Qsys (z.B. für die GPIO-Erweiterung).

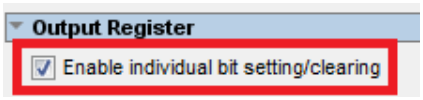


Abbildung 5.5: Ausschnitt PIO Einstellungen, [Qua13]

Hierüber werden folgende Funktionen altera_avalon_pio_regs.h aktiviert:

1. IOWR_ALTERA_AVALON_PIO_SET_BITS(base, data);
   Diese Funktion setzt die unter „data“ genannten Bits im Register „base“ auf logisch 1.

2. IOWR_ALTERA_AVALON_PIO_CLEAR_BITS(base, data);
   Diese Funktion setzt die unter „data“ genannten Bits im Register „base“ auf logisch 0.

5.8 Target Connection Problem in Eclipse

Nach jedem Neustart von Quartus II und Eclipse erscheint bei der ersten darauf folgenden Programmierung des DE0Nano die in Abbildung 5.6 ersichtliche Fehlermeldung, die darauf hinweist, dass keine Quelle mit einem Nios-II-System an den Rechner angebunden ist.

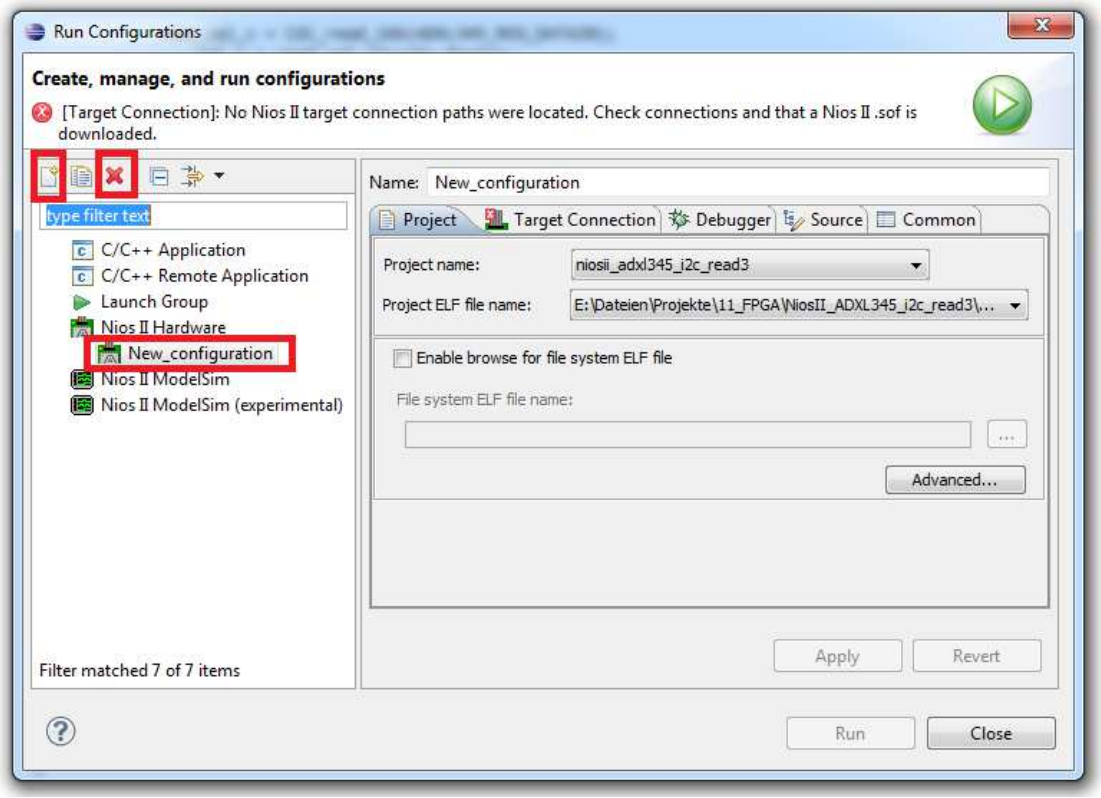


Abbildung 5.6: Target Connection Problem in Eclipse, Eigene Darstellung

Um den USB-Blaster (das DE0Nano-Board) wieder auswählen zu können, muss zunächst die „New_configuration“ gewählt und per rotem „**X**“ gelöscht werden. Nach der Auswahl der „New lauch configuration (Blatt mit kleinem gelben Kreuz) muss auf den Reiter „Target Connection“ gewechselt werden (siehe Abbildung 5.8). Nach einem Klick auf „Refresh Connections“ wird nach einer kurzen Pause der USB-Blaster zur Auswahl angezeigt. Über den abschließenden Klick auf „Run“ wird das Projekt wieder dauerhaft (bis zum Neustart beider Programme) auf den FPGA gespielt.

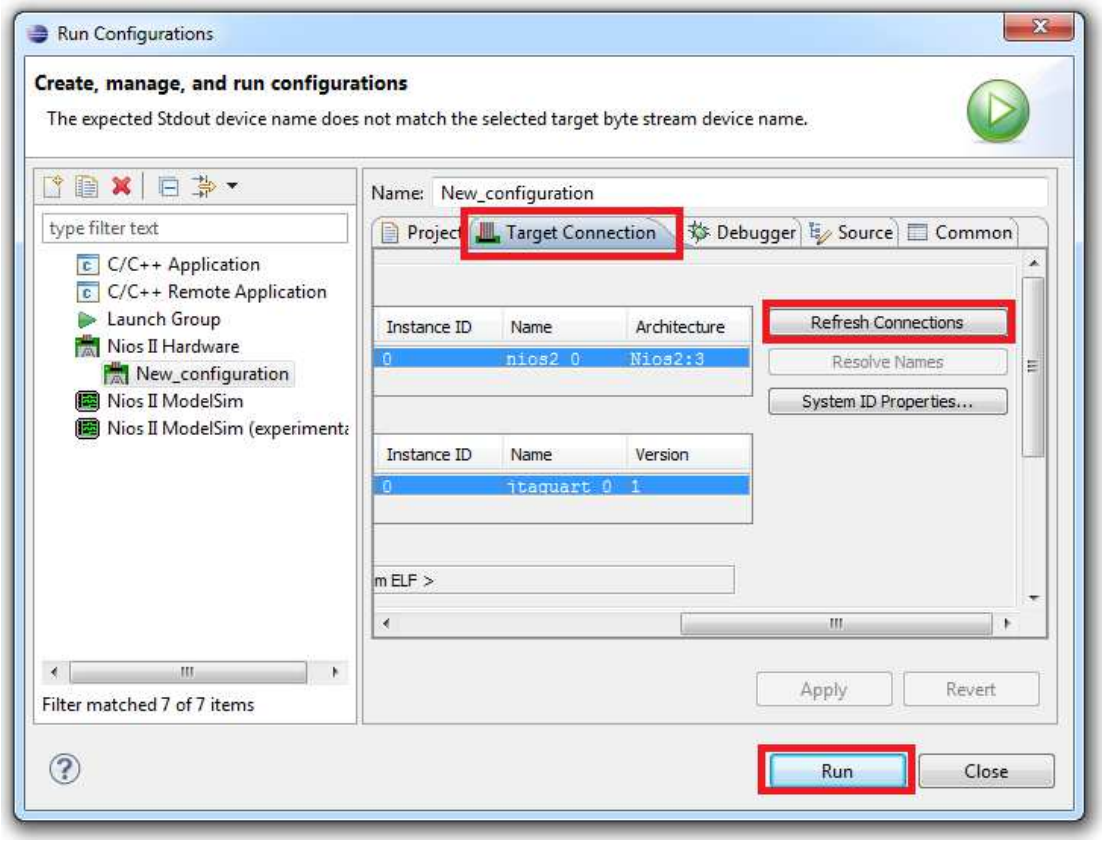


Abbildung 5.7: Target Connection Neuauswahl, Eigene Darstellung

6 Zusammenfassung und Ausblick

Nach aktuellem Stand ist die Belegarbeit mit Implementierung der I^2C-Kommunikation und dem Anzeigen der ermittelten Beschleunigungswerte über das LED-Erweiterungsmodul erfüllt. Die erstellte Dokumentation bietet einen Überblick über die Vorgehensweise und fasst aufgetretene Probleme und Fehlerquellen für Nachfolgearbeiten und Anfänger auf diesem Gebiet zusammen.

Die anfänglichen Versuche zur Verwendung der 3-Wire-SPI-Verbindung wurden verworfen, da keine Kommunikation aufgebaut werden konnte. Aufgrund mangelnder Informationen und Anwendungsbeispielen konnten die über Qsys verfügbaren C-Bibliotheken nicht erfolgreich angewandt werden.
Aufgrund der in Bezug auf den I^2C-Bus gewonnenen Erkenntnisse liegt die Vermutung nahe, dass die SPI-Verbindung ebenfalls aus einer „Reihe von Einzelbefehlen“ (analog dem I^2C) hätte aufgebaut werden müssen. Dies sollte weiterführend betrachtet und dokumentiert werden.

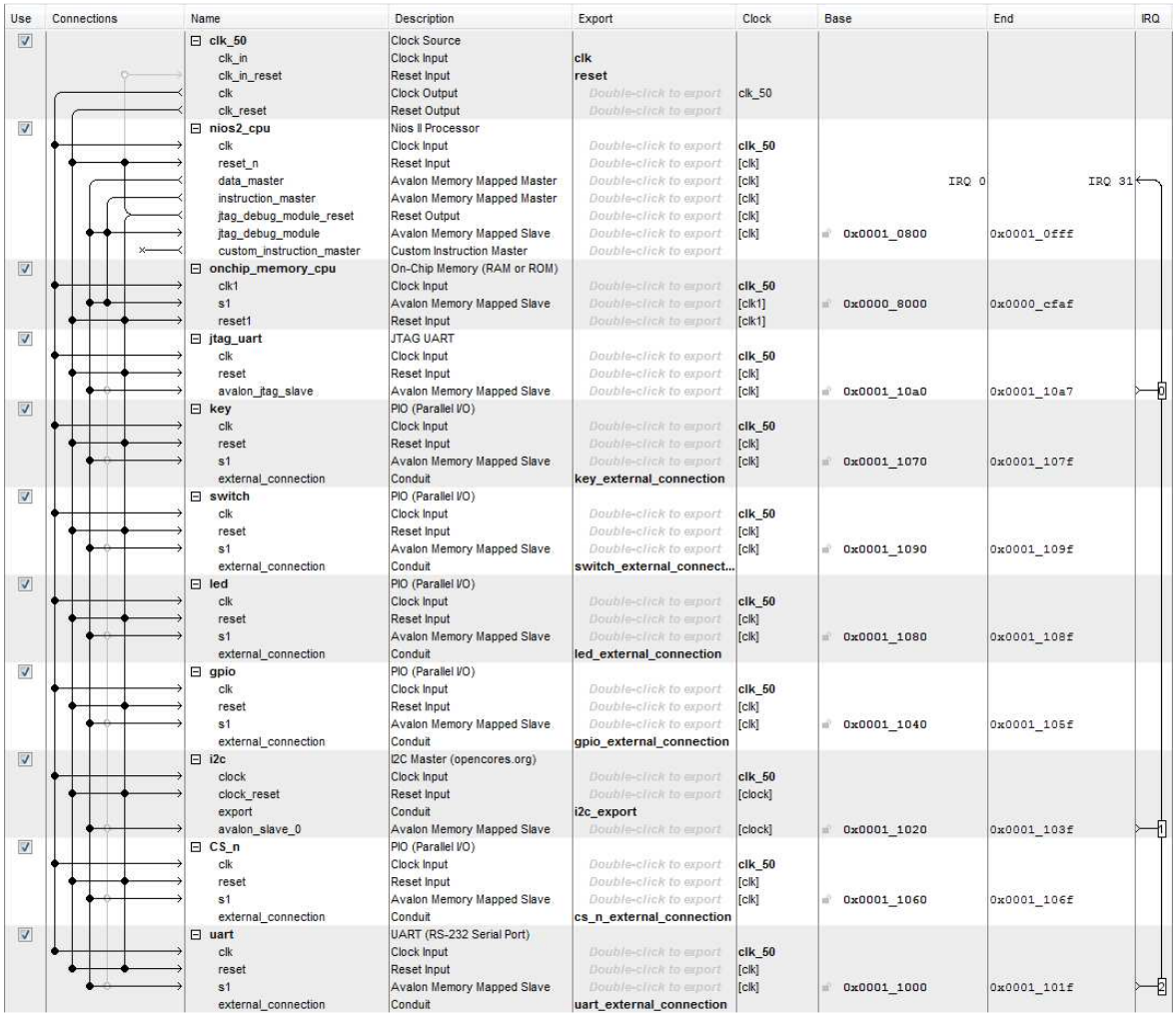
Als Erweiterung soll das in Kapitel 4 angesprochene LabVIEW Programm zur Auswertung und Anzeige der ermittelten Beschleunigungswerte über die serielle Schnittstelle programmiert und ebenfalls dokumentiert werden.

Anhang

Übersicht verwendeter Software:

- Zur Niederschrift dieser Arbeit wurde Microsoft Word aus dem Office Paket *Microsoft Office Home and Student 2010*, Version 14.0.7116.5000 (32Bit), verwendet.
  Die Vorlage dieser Arbeit wurde von der Hochschule Anhalt (VorlageMA_Fall1.docx) bereitgestellt.
- Zur Programmierung des DE0Nano per VHDL sowie zur Erzeugung des Qsys-Modells wurde mit der *Quartus II 64-Bit Version 13.0.0 Build 156 04/24/2013 SJ Web Edition* gearbeitet. Qsys liegt in der Standardinstallation als Version *13.0 Build 156* vor.
- Zur Programmierung des über Qsys erzeugten Mikrocontroller-Modells wurde die mit *Quartus II* mitgelieferte Eclipse-Version *Eclipse IDE for C/C++ Developers, Version: Indigo Service Release 2, Build id: 20120216-1857* verwendet.
- Die *LabVIEW Studentenversion 13.0f2 (32-Bit)* wurde zur Programmierung der graphischen Auswertung der Beschleunigungssensorwerte über die serielle Schnittstelle verwendet.
- Die in dieser Arbeit aufgeführte LED-Erweiterungsplatine wurde mit dem Layoutprogramm *Target3001! V15 (15.9.0.63)* erstellt und über den Platinenfräser *Protomat S103* von der Firma *lpkf* gefräst.

Anhang A: Qsys-Modell

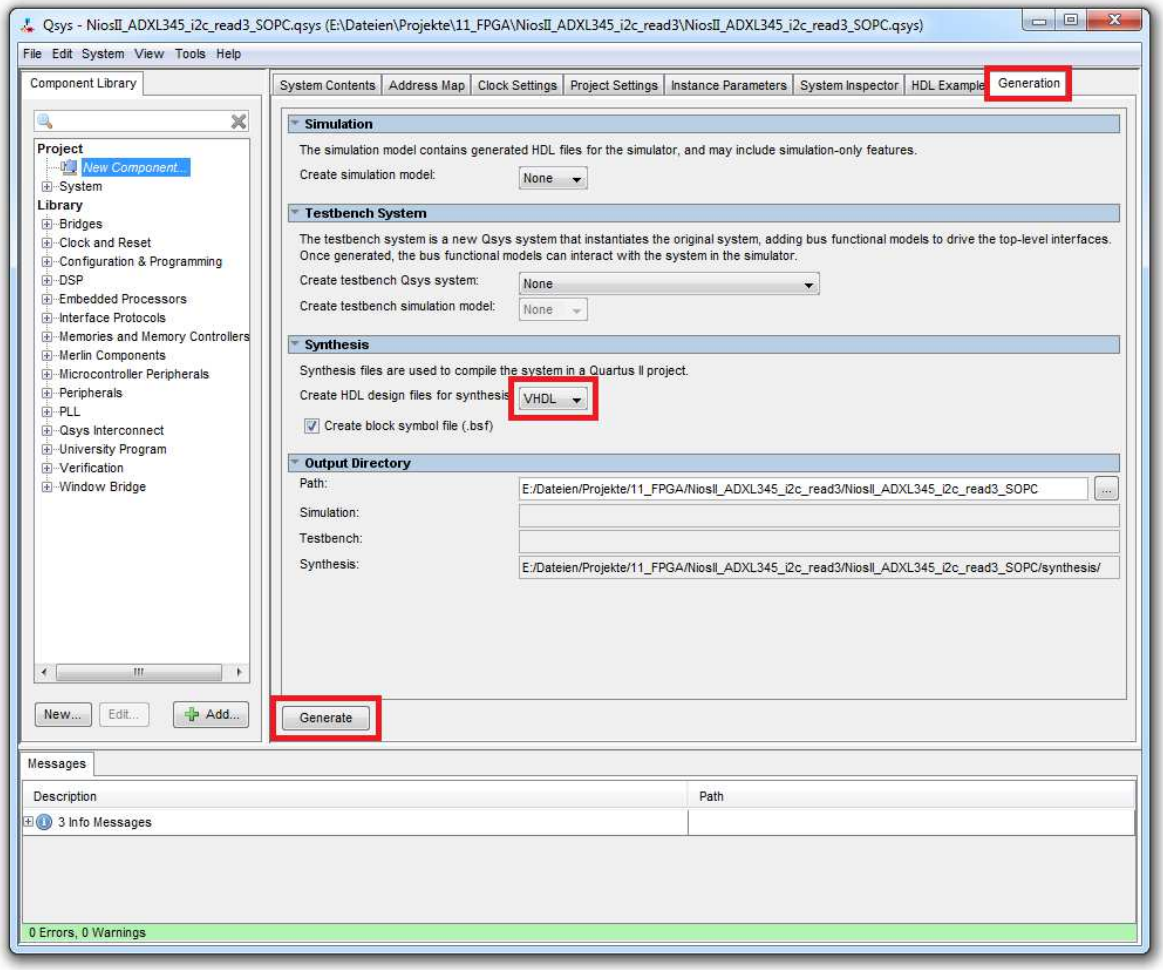


Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
☑		**clk_50**	Clock Source					
		clk_in	Clock Input	**clk**				
		clk_in_reset	Reset Input	**reset**				
		clk	Clock Output	Double-click to export	clk_50			
		clk_reset	Reset Output	Double-click to export				
☑		**nios2_cpu**	Nios II Processor					
		clk	Clock Input	Double-click to export	**clk_50**			
		reset_n	Reset Input	Double-click to export	[clk]			
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			IRQ 31
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
		jtag_debug_module_reset	Reset Output	Double-click to export	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_0800	0x0001_0fff	
		custom_instruction_master	Custom Instruction Master	Double-click to export				
☑		**onchip_memory_cpu**	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	Double-click to export	**clk_50**			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0000_8000	0x0000_cfaf	
		reset1	Reset Input	Double-click to export	[clk1]			
☑		**jtag_uart**	JTAG UART					
		clk	Clock Input	Double-click to export	**clk_50**			
		reset	Reset Input	Double-click to export	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_10a0	0x0001_10a7	0
☑		**key**	PIO (Parallel I/O)					
		clk	Clock Input	Double-click to export	**clk_50**			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1070	0x0001_107f	
		external_connection	Conduit	**key_external_connection**				
☑		**switch**	PIO (Parallel I/O)					
		clk	Clock Input	Double-click to export	**clk_50**			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1090	0x0001_109f	
		external_connection	Conduit	**switch_external_connect...**				
☑		**led**	PIO (Parallel I/O)					
		clk	Clock Input	Double-click to export	**clk_50**			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1080	0x0001_108f	
		external_connection	Conduit	**led_external_connection**				
☑		**gpio**	PIO (Parallel I/O)					
		clk	Clock Input	Double-click to export	**clk_50**			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1040	0x0001_105f	
		external_connection	Conduit	**gpio_external_connection**				
☑		**i2c**	I2C Master (opencores.org)					
		clock	Clock Input	Double-click to export	**clk_50**			
		clock_reset	Reset Input	Double-click to export	[clock]			
		export	Conduit	**i2c_export**				
		avalon_slave_0	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0001_1020	0x0001_103f	1
☑		**CS_n**	PIO (Parallel I/O)					
		clk	Clock Input	Double-click to export	**clk_50**			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1060	0x0001_106f	
		external_connection	Conduit	**cs_n_external_connection**				
☑		**uart**	UART (RS-232 Serial Port)					
		clk	Clock Input	Double-click to export	**clk_50**			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1000	0x0001_101f	2
		external_connection	Conduit	**uart_external_connection**				

Anhang B: VHDL-Code

```
1 –Alexander Wilke MEF2013
2
3 LIBRARY ieee;
4 USE ieee.std_logic_1164.all;
5 USE ieee.std_logic_arith.all;
6 USE ieee.std_logic_unsigned.all;
7
8 ENTITY NiosII_ADXL345_i2c_read3 IS
9 PORT (
10 CLOCK_50          : IN STD_LOGIC;
11 KEY               : IN STD_LOGIC_VECTOR(1 downto 0);
12 SW                : IN STD_LOGIC_VECTOR(3 downto 0);
13 LED               : OUT STD_LOGIC_VECTOR(7 downto 0);
14 GPIO_0            : INOUT STD_LOGIC_VECTOR(31 downto 0);
15 I2C_SDAT          : INOUT STD_LOGIC;
16 I2C_SCLK          : INOUT STD_LOGIC;
17 G_SENSOR_CS_N     : out STD_LOGIC;
18 GPIO_1            : inout STD_LOGIC_VECTOR(1 downto 0)
19 );
20
21 END NiosII_ADXL345_i2c_read3;
22
23 ARCHITECTURE Structure OF NiosII_ADXL345_i2c_read3 IS
24
25          component NiosII_ADXL345_i2c_read3_SOPC is
26                    port (
27                              clk_clk                                 : in std_logic                          := 'X';
28                              reset_reset_n                           : in std_logic                          := 'X';
29                              --IOs
30                              led_external_connection_export          : out std_logic_vector (7 downto 0);
31                              switch_external_connection_export       : in std_logic_vector (3 downto 0)      := (others => 'X');
32                              key_external_connection_export          : in std_logic_vector (1 downto 0)      := (others => 'X');
33                              gpio_external_connection_export         : inout std_logic_vector (31 downto 0);
34                              --I2C
35                              i2c_export_scl_pad_io                   : inout std_logic                       := '1';
36                              i2c_export_sda_pad_io                   : inout std_logic                       := '0';
37                              cs_n_external_connection_export         : out std_logic ;
38                              uart_external_connection_rxd            : in std_logic                          := 'X';
39                              uart_external_connection_txd            : out std_logic
40                    );
41          end component NiosII_ADXL345_i2c_read3_SOPC ;
42
43
44 BEGIN
45 -- Instantiate the Nios II system entity generated by the SOPC Builder
46 NiosII : NiosII_ADXL345_i2c_read3_SOPC port map (
47          clk_clk                                 => CLOCK_50 ,
48          reset_reset_n                           => KEY(0),
49          --IOs
50          led_external_connection_export          => LED,
51          switch_external_connection_export       => SW,
52          key_external_connection_export          => KEY,
53          gpio_external_connection_export         => GPIO_0 ,
54          --I2C
55          i2c_export_scl_pad_io                   => I2C_SCLK ,
56          i2c_export_sda_pad_io                   => I2C_SDAT ,
57          cs_n_external_connection_export         => G_SENSOR_CS_N ,
58          --UART
59          uart_external_connection_rxd            => GPIO_1 (1),
60          uart_external_connection_txd            => GPIO_1 (0)
61 );
62
63 END Structure ;
64
```

Anhang C: Qsys-Kompilierung



Anhang D: Eigene I²C-Routinen

```
//*************************************************************************
// I2C_read_8b ************************************************************
alt_u32 I2C_read_8b(alt_u8 reg){

        int status;
        alt_u8 read_data;

        // Befehlsanordnung zum Lesen eines Byte über den I2C von Open
        status = I2C_start(I2C_BASE, 0x1D, 0);          //read =  1== read    0== write
        /*
        if (status == 0) { printf("Adresse ist anerkannt \n");
        }
        else{ printf("Adresse ist nicht anerkannt \n");
        }
        */
        I2C_write(I2C_BASE, reg, 0);                    //z.B. ADXL345_REG_POWER_CTL
        status = I2C_start(I2C_BASE, 0x1D, 1);          //read =  1== read    0== write
        read_data = I2C_read(I2C_BASE, 1);
        status = I2C_read(I2C_BASE, 1);                 //stop-Funktion!!!

        return read_data;
}

//*************************************************************************
// I2C_read_16b ***********************************************************
alt_u16 I2C_read_16b(alt_u8 reg){

        int status;
        alt_u8 read_data1, read_data2;
        alt_u16 read_data;

        // Befehlsanordnung zum Lesen eines Byte über den I2C von OpenCore
        status = I2C_start(I2C_BASE, 0x1D, 0);          //read =  1== read    0== write
        /*
        if (status == 0) { printf("Adresse ist anerkannt \n");
        }
        else{ printf("Adresse ist nicht anerkannt \n");
        }
        */
        I2C_write(I2C_BASE, reg, 0);                    //z.B. ADXL345_REG_POWER_CTL
        status = I2C_start(I2C_BASE, 0x1D, 1);          //read =  1== read    0== write
        read_data1 = I2C_read(I2C_BASE, 0);
        read_data2 = I2C_read(I2C_BASE, 1);
        status = I2C_read(I2C_BASE, 1);                 //stop-Funktion!!!

        // zwei 8Bit Werte zu einem 16Bit Wert zusammenfügen.
        read_data = ( read_data2 << 8 ) | read_data1;

        return read_data;
}

//*************************************************************************
// I2C_write_8b ***********************************************************

void I2C_write_8b(alt_u8 data, alt_u8 reg){

        int status;

        // Befehlsanordnung zum Schreiben eines Bytes über den I2C von Open
        status = I2C_start(I2C_BASE, 0x1D, 0);          //read =  1== read    0== write
        /*
        if (status == 0) { printf("Adresse ist anerkannt \n");
        }
        else{ printf("Adresse ist nicht anerkannt \n");
        }
        */
        I2C_write(I2C_BASE, reg, 0);
        I2C_write(I2C_BASE, data, 1);                   //z.B. bit3 -> 0x08
        status = I2C_read(I2C_BASE, 1);                 //stop-Funktion!!!

}
```

Anhang E: _delay_ms-Funktion

```
//***************************************************************************
// DELAY ********************************************************************
void _delay_ms (int msec){
        int i;
        for (i=0; i<=msec; i++){
                usleep(1000);          // 1ms warten
        }
}
```

Anhang F: send_string-Funktion

```
//#############################################################################################
// UART

//*****************************************************************************
// send_string ****************************************************************

void send_string(char *tx_string){

        int i;
        int laenge = strlen(tx_string);
        char *string = tx_string;          // oder = &tx_string[0]

        for (i=0; i<=(laenge-1);i++){
        // Transmit ready (TRDY 0x40) muss 1 sein bevor neues Zeichen gesendet werden kann
        while (!(IORD_ALTERA_AVALON_UART_STATUS(UART_BASE) & ALTERA_AVALON_UART_STATUS_TRDY_MSK)){}
        // sende 1. - n. Zeichen
        IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE, string[i]);
        }
}
```

Abkürzungsverzeichnis

LED	Licht emmitierende Diode (engl.: light emitting diode)
FPGA	Licht emmitierende Diode (engl.: field programmable gate array)

Abbildungsverzeichnis

Tabellenverzeichnis

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

Literaturverzeichnis

[AnD10]	Analog Devices *ADXL345 User Manual, Rev. A* Bedienungsanleitung zum Beschleunigungssensor ADXL345 Herstellers Analog Devices, 2009-2010 Kostenlos beziehbar auf: http://pdf1.alldatasheet.com/datasheet-pdf/view/254714/AD/ADXL345.html
[Bru14]	Brutscheck, M. *Hardware-Software-Codesign* Vorlesungsskript als PDF, Fakultät Elektrotechnik, Hochschule Anhalt, Köthen, 2014
[Dav15]	Dave (User im Internet) Forumsbeitrag auf der Internetplattform von mikrocontroller.net, Januar 2015 Internetseite: http://www.mikrocontroller.net/topic/355246?goto=3965953#3965953
[Her03]	Herveille, R. *I²C-Master Core Specification* Handbuch zum I²C-Core, 2003 Kostenlos beziehbar auf: http://www.alterawiki.com/wiki/I2C_(OpenCores)
[Qua13]	Altera Corporation *Software Quartus II* *Quartus II 64-Bit Version 13.0.0 Build 156 04/24/2013 SJ Web Edition*, Copyright (C) 1991-2013 Altera Corporation
[Ter13]	TerasIC Technologies *DE0-Nano User Manual* Bedienungsanleitung zum FPGA DE0Nano des Herstellers TerasIC, 2003-2013 Kostenlos beziehbar auf: ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Boards/DE0-Nano/DE0_Nano_User_Manual.pdf