Westfälische Hochschule

Fachbereich Elektrotechnik und Angewandte Naturwissenschaften

D O K U M E N T A T I O N

Implementierung eines FFT-Algorithmus auf einer Softcore CPU zur Frequenzanalyse von Audiosignalen

Wahlmodul Entwicklung digitaler Systeme

von

Lukas Wöhle

betreut von

Betreuer: Dipl.-Ing.(FH) Oliver Gießelmann
Dozent: Prof. Dr. Udo Jorczyk

Gelsenkirchen, 21. Februar 2017

Disclaimer - REV 1

Dieses Dokument dient der Erstellung eines Softcore Prozessors (NIOS 2) auf einem FPGA der Firma Altera (Cyclone 2 und 4). Innerhalb des Projektes wurde das Entwicklungsboard DE2-115 der Firma Terasic verwendet.

- Der Autor übernimmt keinerlei Gewähr für die Aktualität, Korrektheit, Vollständigkeit oder Qualität der bereitgestellten Informationen. Haftungsansprüche gegen den Autor, welche sich auf Schäden materieller oder ideeller Art beziehen, die durch die Nutzung oder Nichtnutzung der dargebotenen Informationen bzw. durch die Nutzung fehlerhafter und unvollständiger Informationen verursacht wurden, sind grundsätzlich ausgeschlossen, sofern seitens des Autors kein nachweislich vorsätzliches oder grob fahrlässiges Verschulden vorliegt. Alle Angebote sind freibleibend und unverbindlich. Der Autor behält es sich ausdrücklich vor, Teile der Seiten oder das gesamte Angebot ohne gesonderte Ankündigung zu verändern, zu ergänzen, zu löschen oder die Veröffentlichung zeitweise oder endgültig einzustellen.

- Bei direkten oder indirekten Verweisen auf fremde Webseiten ("Hyperlinks"), die außerhalb des Verantwortungsbereiches des Autors liegen, würde eine Haftungsverpflichtung ausschließlich in dem Fall in Kraft treten, in dem der Autor von den Inhalten Kenntnis hat und es ihm technisch möglich und zumutbar wäre, die Nutzung im Falle rechtswidriger Inhalte zu verhindern. Der Autor erklärt hiermit ausdrücklich, dass zum Zeitpunkt der Linksetzung keine illegalen Inhalte auf den zu verlinkenden Seiten erkennbar waren. Auf die aktuelle und zukünftige Gestaltung, die Inhalte oder die Urheberschaft der verlinkten/verknüpften Seiten hat der Autor keinerlei Einfluss. Deshalb distanziert er sich hiermit ausdrücklich von allen Inhalten aller verlinkten /verknüpften Seiten, die nach der Linksetzung verändert wurden. Diese Feststellung gilt für alle innerhalb des eigenen Internetangebotes gesetzten Links und Verweise sowie für Fremdeinträge in vom Autor eingerichteten Gästebüchern, Diskussionsforen, Linkverzeichnissen, Mailinglisten und in allen anderen Formen von Datenbanken, auf deren Inhalt externe Schreibzugriffe möglich sind. Für illegale, fehlerhafte oder unvollständige Inhalte und insbesondere für Schäden, die aus der Nutzung oder Nichtnutzung solcherart dargebotener Informationen entstehen, haftet allein der Anbieter der Seite, auf welche verwiesen wurde, nicht derjenige, der über Links auf die jeweilige Veröffentlichung lediglich verweist.

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des Wahlmoduls EDS (Entwicklung digitaler Systeme)gilt es, mittels einer Hardware Beschreibungssprache (VHDL) ein digitales System zu entwickeln, um damit spezifische Aufgaben zu lösen. Innerhalb dieser Arbeit wird eine Softcore CPU auf einem FPGA Board implementiert, um über diese eine Online Frequenzanalyse eines Audiosignals zu ermöglichen.

Für dieses Vorhaben bietet sich die Softcore Lösung an, da diese variabel dimensioniert werden kann, vor allem bezüglich des nutzbaren Speichers. Der ASIC Hersteller Altera bietet zu diesem Zwecke die Softcore CPU, namentlich NIOS II, an, welcher innerhalb eines System Design Tools (QSYS) variabel dimensioniert und erweitert werden kann. Auf diese Weise kann ein beliebig komplexes System erschaffen werden, welches u.a. eine Schnittstelle zu einem Host-PC besitzt, welches die Daten aufnehmen bzw. grafisch darstellen kann (z.B. Ethernet oder JTAG).

2 Aufgabenstellung

Wie Eingangs bereits erwähnt soll eine Frequenzanalyse von Audiosignalen auf einem FPGA der Firma Altera implementiert werden. Hierfür stand eine bereits Implementierung Frequenzanalyse in Form des FFT-Algorithmus (Fast Fourier Transformation) in der Programmiersprache „*C*“ zur Verfügung. Diese soll genutzt werden, um ein entsprechendes Audiosignal auszuwerten.

Weiterhin soll der oben beschriebene Prozess auf einem FPGA Board implementiert werden, um so die Problematik eines statischen Systems (z.B. Mikrocontroller) zu umgehen (variable Speichergröße - beliebig komplexe Digitalschaltungen).

Die Lösung des oben genannten Problems lässt sich in folgende Teilaufgaben gliedern:

Softcore Prozessor Aufsetzen eines **Softcore Prozessors** - *NIOS II*. Dieser Prozessor ermöglicht die Implementierung des bereits vorhandenen C-Codes in das System. Der Prozessor muss mit genügend großem Speicher ausgelegt werden, um eine Frequenzanalyse mit einer hohen Auflösung, respektive Anzahl Samples durchführen zu können.

Anbindung des Audio Codec Die Frequenzauswertung soll über ein Audiosignal erfolgen. Dieses muss dem Prozessor bzw. der Applikation zugänglich sein. Dafür wird ein Audio Codec benötigt, welcher über eine entsprechende Schnittstelle sowohl Daten an den Prozessor weiterleitet, als auch direkt zum Audioausgang des Codec selbst, um eine Online Auswertung und gleichzeitige Audiowiedergabe zu gewährleisten. Dieser Codec muss durch den Prozessor variabel konfigurierbar sein, um die Auflösung bzw. die Sample-rate u.ä. zu verändern bei anderen Audiosignalen.

Implementierung des FFT-Alg. Der bereits vorliegende C-Code der FFT muss im Softcore Prozessor implementiert werden. Hier muss also ein Array aufgebaut werden, welches die Samples des Audiosignals beinhaltet über welche dann die FFT durchgeführt wird. Weiterhin soll die Ausgabe der Frequenz zu Debugg-Zwecken in der Konsole des angeschlossenen Host-Rechners ausgegeben werden.

Die drei Arbeitspakete werden in den nachfolgenden Kapiteln näher erläutert.

3 QSYS - System Design

Das System soll auf der Entwicklungsplattform DE2-115 (Evaluierungsboard von Terasic (Cyclone 4)) implementiert werden. Hierfür liefert der FPGA Hersteller Altera die nötige Software (Quartus). Innerhalb von Quartus ist es möglich vordefinierte IP-Cores (generischer VHDL-Code) für ein System auszuwählen und diese grafisch zu verbinden. Weiterhin müssen bzw. können hier wichtige Voreinstellungen getroffen werden (Registerbreite, Speicherplatz u.ä.). Nachfolgend ist die Herangehensweise zur Erstellung einer Softcore CPU , namentliche dem NIOS II, und der benötigten Peripherie aufgeführt.

3.1 NIOS II

1. **QSYS öffnen**
   - In der Leiste wo ein neues File erstellt werden kann, neben dem Chipplaner ist QSYS zu finden. Durch einen Klick öffnet sich ein neues Fenster.

2. **Neues System designen**
   - Erstellung eines neuen Systems (Clock ist bereits initialisiert). Bei Clock_in_reset: Export anklicken um ein Signal nach außen zuführen. Der Reset soll extern durch einen Schalter erfolgen und nicht durch den Clock-Reset. Dieser kann innerhalb der Programmierung des Prozessors zu Problemen führen.

3. **Hinzufügen des Prozessors**
   - Unter Komponenten (linke Seite): Hinzufügen des Prozessors (NIOS II) -> *Add* zum bestätigen klicken. Nun muss die Version des Prozessors ausgewählt werden. Es können zwei unterschiedliche Typen gewählt werden: Dabei ist die Version f die schnellere Variante des Prozessors, allerdings muss diese auch durch die Lizenz abgedeckt sein. Daher wurde der e-Typ gewählt, da dieser frei verfügbar ist ohne Mehrkosten.

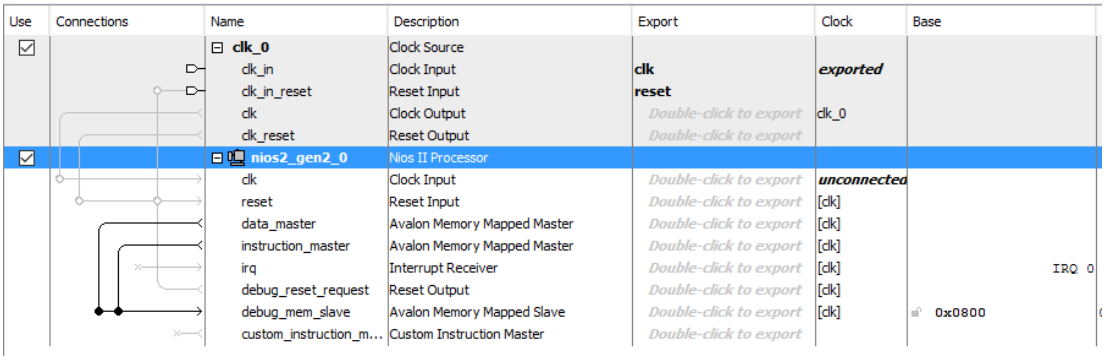

Abbildung 3.1: Screenshot der Komponenten- Prozessor und Clock.

4. **Verbinden des NIOS**

   a) Verbinden der Komponenten: Die Komponenten sind generische VHDL-Files. Die Eingangs- bzw. Ausgangssignale müssen mit den entsprechenden Aus- und Eingängen der übrigen Komponenten verknüpft werden, es erfolgt also eine Zuweisung der Signale untereinander. Verbindungen: Vorerst nur Clock (clk) und Clock_reset. Durch einen Klick auf eine grau hinterlegte Leitung (Knotenpunkt) wird die Verbindung aktiv (schwarz).

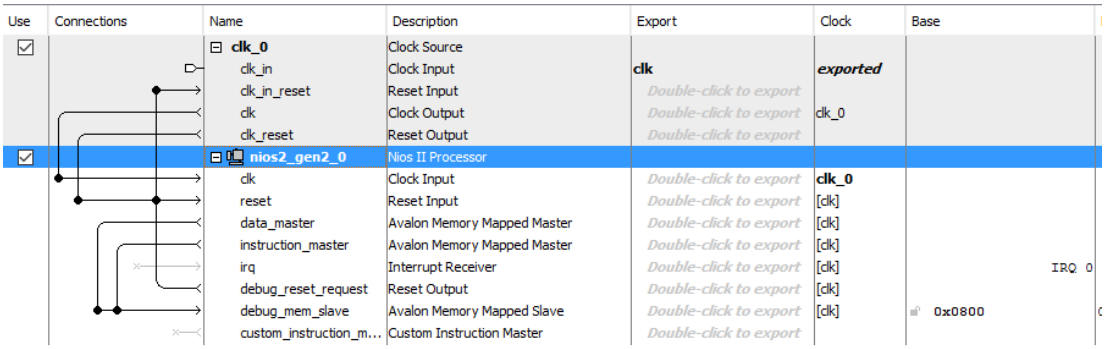

Abbildung 3.2: Screenshot der verbundenen Komponenten- Prozessor und Clock.

5. **Einfügen einer Programmierschnittstelle (JTAG-UART)**
   - Hinzufügen einer JTAG-UART- Schnittstelle zur Programmierung des NIOS II. Verbindungen : CLK, Reset (nur clk_reset!), avalon_jtag_Slave (Nur Data_Master des NIOS).

6. **Speicher hinzufügen**
   - Speicher hinzufügen (On-Chip Memory - RAM). In Parameterliste genügend Speicher (Bytes) anlegen. Hier 256 KByte Speicher. Verbindungen: CLK, S1 (Data und Instruktion Master), Reset (auch nur CLK_reset).
   - Nun den NIOS Prozessor anwählen (Parameter). Unter Vectors (Reset & Exception Vector Memory) den ON-Chip Memory Baustein als Vector Memory auswählen (Vector offset Prüfen! Min 0x20 bei Exception mehr!).

7. **Optionale PIO**
   - Falls gewünscht PIO (parallel Input / Output) einfügen. Verbindungen: CLK, Reset, s1 (data und Instruktion). Außerdem bei external connection Auf Export doppelt klicken (nach außen verfügbar machen). Hierdurch können z.B. LEDs angeschlossen werden, welche ein visuelles Feedback geben. Weiterhin ist es über diese Schnittstelle möglich andere VHDL-Komponenten dem Prozessor verfügbar zu machen. Innerhalb dieser Arbeit wurde ein einfacher 8 Bit Counter in VHDL erzeugt und über die PIO an den Prozessor angeschlossen. Der Prozessor kann den Counter durch ein Enable bit entsprechend an bzw. ausschalten.

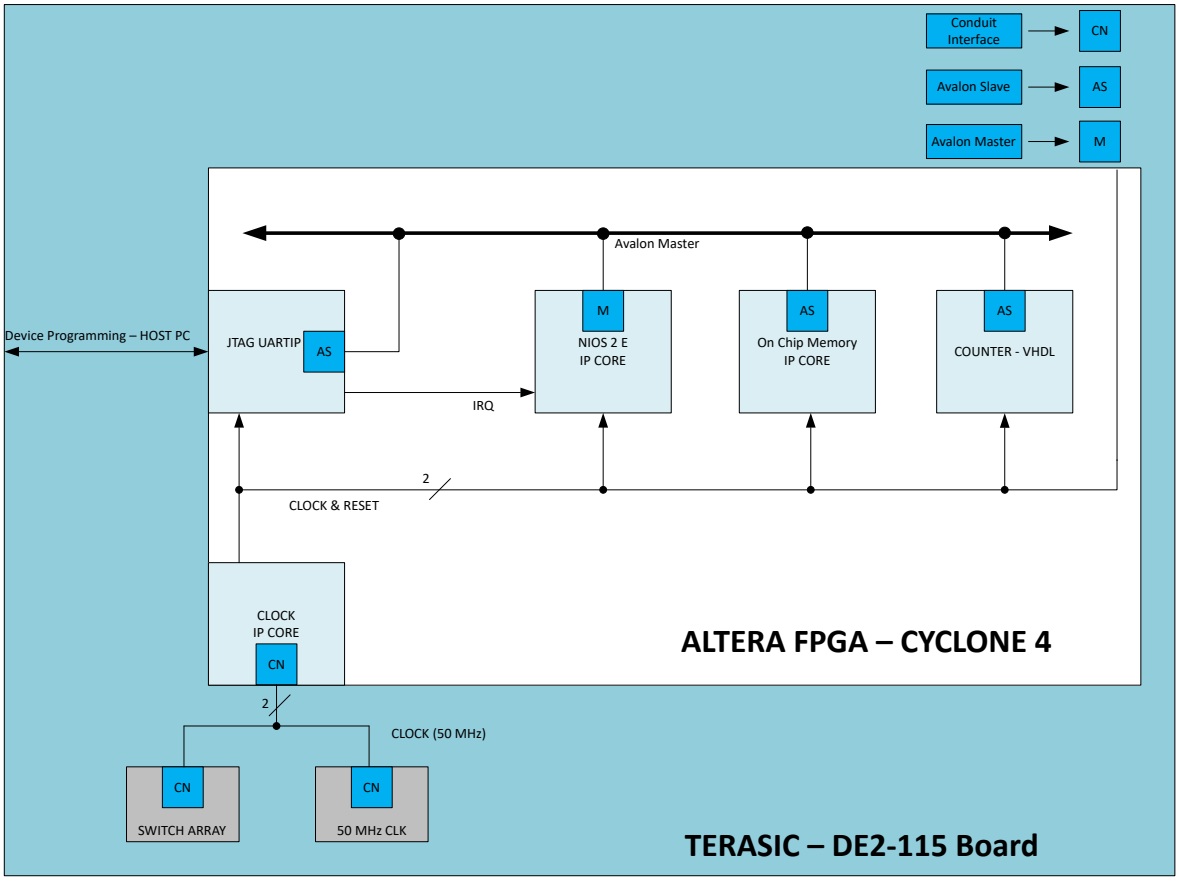

Abbildung 3.3: Schema des Systemdesigns für eine Softcore CPU mit einer externen Counter Komponente.

Die benötigten Komponenten, um einen Softcore Prozessor auf einem FPGA zu implementieren sind in Abb.3.3 schematisch dargestellt. Weiterhin können die benötigten Verbindungen bzw. Interfaces und Ankopplungen dieser an externe Hardware und andere IP Cores eingesehen werden.

3.2 AUDIO-Codec

Innerhalb dieses Unterabschnittes werden die Komponenten, welche für die Anbindung des Audio Codec benötigt werden, vorgestellt. Dieser kann durch zwei IP-Cores mit dem NIOS verbunden werden. Die beiden Cores erzeugen dafür die Schnittstelle zur Konfiguration des Codec über I2C und stellen eine Schnittstelle zum Auslesen der FIFO des Audio Codecs bereit.

1. **Audio PLL**
   - Hinzufügen eines PLL (Phase Locked Loop). Der Codec benötigt ein 12.88 MHz Taktsignal, welches aus der 50 MHz Clock des Boards abgeleitet werden muss. Hierfür kann der Audio PLL IP-Core genutzt werden. Diese Komponenten muss hinzugefügt werden und entsprechend auf die benötigte Frequenz eingestellt werden. Verbindungen : ref_CLK auf CLK, ref_Reset auf CLK_reset. Weiterhin muss der PLL-Ausgang (audio_clk) nach außen exportiert werden, um diesen in der Top Entity des Projektes auf den Codec zu routen.

2. **Audio Komponente hinzufügen**
   - Hinzufügen des Audio IP Cores. Der Codec speichert Samples, welche am Ausgang des ADC (Analgo Digital Converters) in einer FIFO (First in First Out Speicher). Diese muss vom Prozessor ausgelesen werden. Die Audio Komponente ermöglicht dies. Hier muss die Breite der Daten angegeben werden (in dieser Applikation 24 Bit). Verbindungen : CLK auf CLK, Reset auf CLK_reset und auf reset_source der Audio PLL, Avalon_audio_slave auf data_master und instruction_master des NIOS, interrupt ref_Reset auf CLK_reset. Weiterhin muss das external_interface (Conduit-Schnittstelle) durch einen Doppelklick nach außen exportiert werden. Hierbei handelt es sich um die Datenleitungen des Codecs, also der Left Right Clock und die Datenleitungen des ADC bzw. DAC. Diese müssen innerhalb der Top Entity mit dem Codec verbunden werden (vgl. Anhang, VHDL-Code 7.2)

3. **Audio Config hinzufügen**
   - Hinzufügen des Audio und Video Config IP Cores. Der auf dem DE 115-2 Board befindliche Codec muss bzw. kann durch diesen Core vorkonfiguriert werden bzw. über diesen später programmiert werden. Hier muss nun das entsprechende Entwicklungsboard ausgewählt werden. Außerdem ist es möglich den Codec hier bereits mit entsprechenden Daten zu konfigurieren (Sample Rate, Datawidth usw.). Der Codec wird in diesem Projekt zuvor Automatisch initialisiert. Innerhalb der Applikation, welche auf dem NIOS implementiert wird, können diese Werte durch die entsprechenden Registerbezüge abgeändert bzw. programmiert werden (siehe Kapitel 5, Programmierung des Softcore Prozessors in C, Anhang, *AUDIO_CODEC_REGISTER_CONFIG.h*).

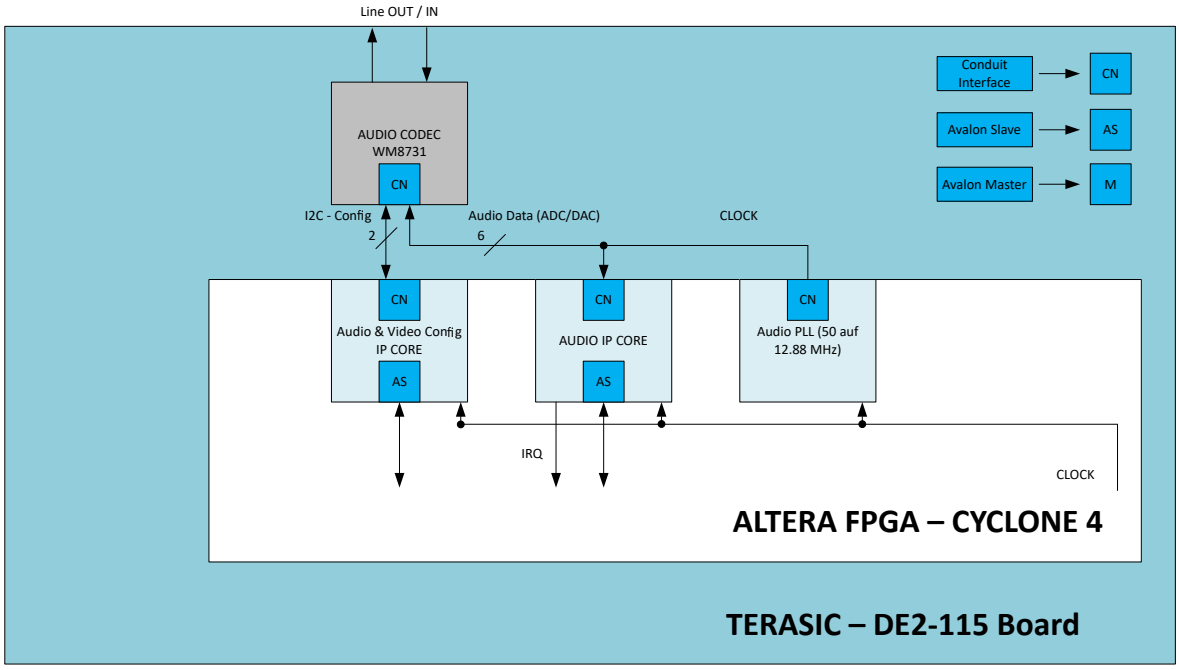

Abbildung 3.4: Schema des Systemdesigns der Audio Komponenten. Es fehlt lediglich die Anbindung an den Avalon Bus.

3.3 VHDL generieren und Instanziierung

1. **Adressen zuweisen**
   - Abschließend müssen die einzelnen Komponenten, deren Register usw., noch einen Adressbezug also eine Adresse erhalten. Dies geschieht über die Adress-zuweisen: System -> Assign Base Adresses. Das Ergebnis bzw. die endgültige Struktur könnte wie folgt aussehen.

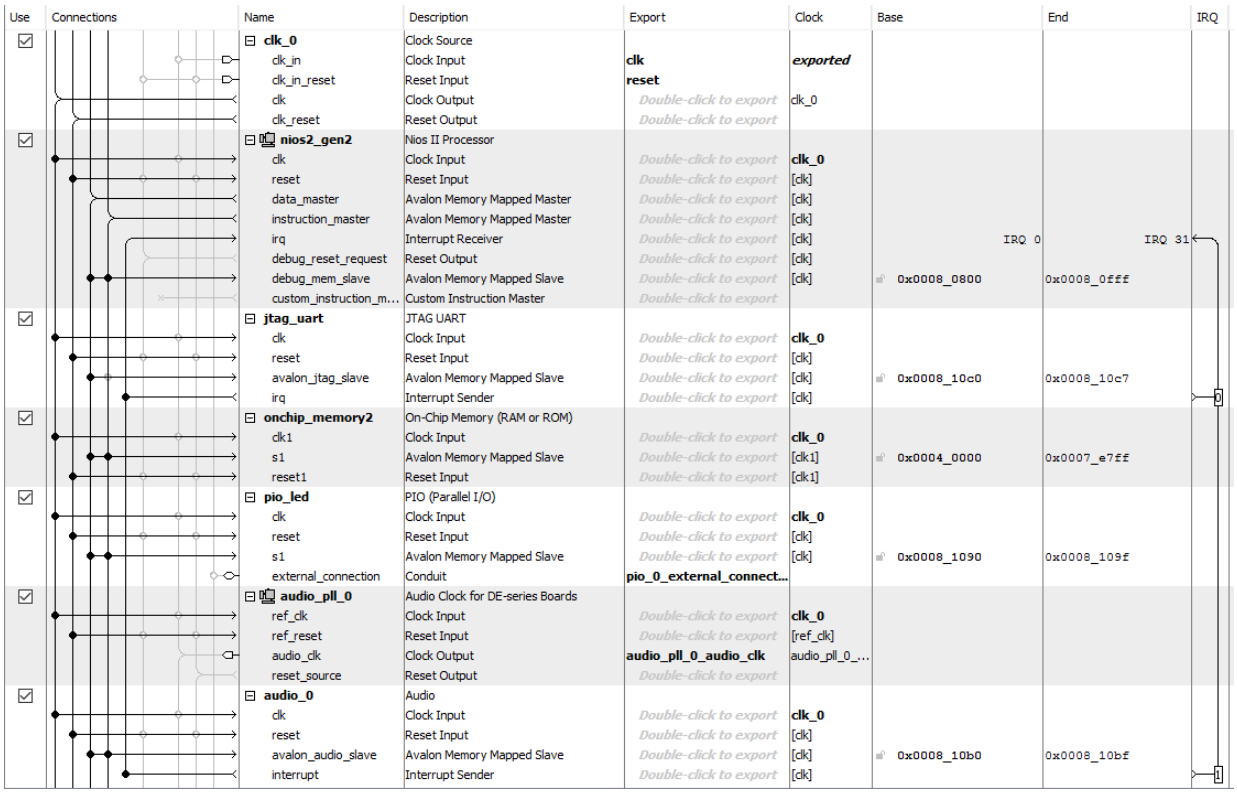

Abbildung 3.5: Beispielhaftes System inklusive Verbindungen. Die genaue Struktur kann in Abb.3.6 auf Seite 11 eingesehen werden.

2. **Projekt abspeichern.**

3. **VHDL generieren**
   - Erzeugen der VHDL - Komponenten des NIOS: Generate -> Generate HDL -> Anwählen der gewünschten HDL (VHDL).

4. **Instanziierungstemplate erzeugen**
   - Instanziierung des NIOS für Top Entity des Codes für den FPGA anzeigen und kopieren (Generate ->Show Instatiation Template).

```
1
2     component NIOS2_tut_audio is
3         port (
4             audio_0_external_interface_ADCDAT           : in   std_logic                  := 'X';           --
                  ADCDAT
5             audio_0_external_interface_ADCLRCK          : in   std_logic                  := 'X';           --
                  ADCLRCK
6             audio_0_external_interface_BCLK             : in   std_logic                  := 'X';           --
                  BCLK
7             audio_0_external_interface_DACDAT           : out  std_logic;                                   --
                  DACDAT
8             audio_0_external_interface_DACLRCK          : in   std_logic                  := 'X';           --
                  DACLRCK
9             audio_and_video_config_0_external_interface_SDAT : inout std_logic            := 'X';           --
                  SDAT
10            audio_and_video_config_0_external_interface_SCLK : out std_logic;                               --
                  SCLK
11            audio_pll_0_audio_clk_clk                   : out  std_logic;                                   --
                  clk
12            clk_clk                                     : in   std_logic                  := 'X';           --
                  clk
13            cnt_en_external_connection_export           : out  std_logic;                                   --
                  export
14            counter_out_external_connection_export      : in   std_logic_vector(7 downto 0) := (others => 'X
                  '); -- export
15            pio_0_external_connection_export            : out  std_logic_vector(7 downto 0);                --
                  export
16            reset_reset_n                               : in   std_logic                  := 'X';           --
                  reset_n
17            vol_en_external_connection_export           : in   std_logic                  := 'X'            --
                  export
18        );
19    end component NIOS2_tut_audio;
20
21    u0 : component NIOS2_tut_audio
22        port map (
23            audio_0_external_interface_ADCDAT           => CONNECTED_TO_audio_0_external_interface_ADCDAT,
24            audio_0_external_interface_ADCLRCK          => CONNECTED_TO_audio_0_external_interface_ADCLRCK,
25            audio_0_external_interface_BCLK             => CONNECTED_TO_audio_0_external_interface_BCLK,
26            audio_0_external_interface_DACDAT           => CONNECTED_TO_audio_0_external_interface_DACDAT,
27            audio_0_external_interface_DACLRCK          => CONNECTED_TO_audio_0_external_interface_DACLRCK,
28            audio_and_video_config_0_external_interface_SDAT =>
                  CONNECTED_TO_audio_and_video_config_0_external_interface_SDAT,
29            audio_and_video_config_0_external_interface_SCLK =>
                  CONNECTED_TO_audio_and_video_config_0_external_interface_SCLK,
30            audio_pll_0_audio_clk_clk                   => CONNECTED_TO_audio_pll_0_audio_clk_clk,
31            clk_clk                                     => CONNECTED_TO_clk_clk,
32            cnt_en_external_connection_export           => CONNECTED_TO_cnt_en_external_connection_export,
33            counter_out_external_connection_export      =>
                  CONNECTED_TO_counter_out_external_connection_export,
34            pio_0_external_connection_export            => CONNECTED_TO_pio_0_external_connection_export,
35            reset_reset_n                               => CONNECTED_TO_reset_reset_n,
36            vol_en_external_connection_export           => CONNECTED_TO_vol_en_external_connection_export
37        );
```

VHDL-Code 3.1: Instanzzierungstemplate des aufgesetzten Systems

Nach diesen Schritten sollte ein entsprechendes .qip File erstellt worden sein, welches in das Projekt eingebunden werden kann (in Quartus). Dieses .qip-File enthält alle VHDL-Komponenten des QSYS-System Designs. Nun muss lediglich die Komponente und die Instanziierung des Prozessors erfolgen. Diese muss in der Top-Entity des Projektes erfolgen.

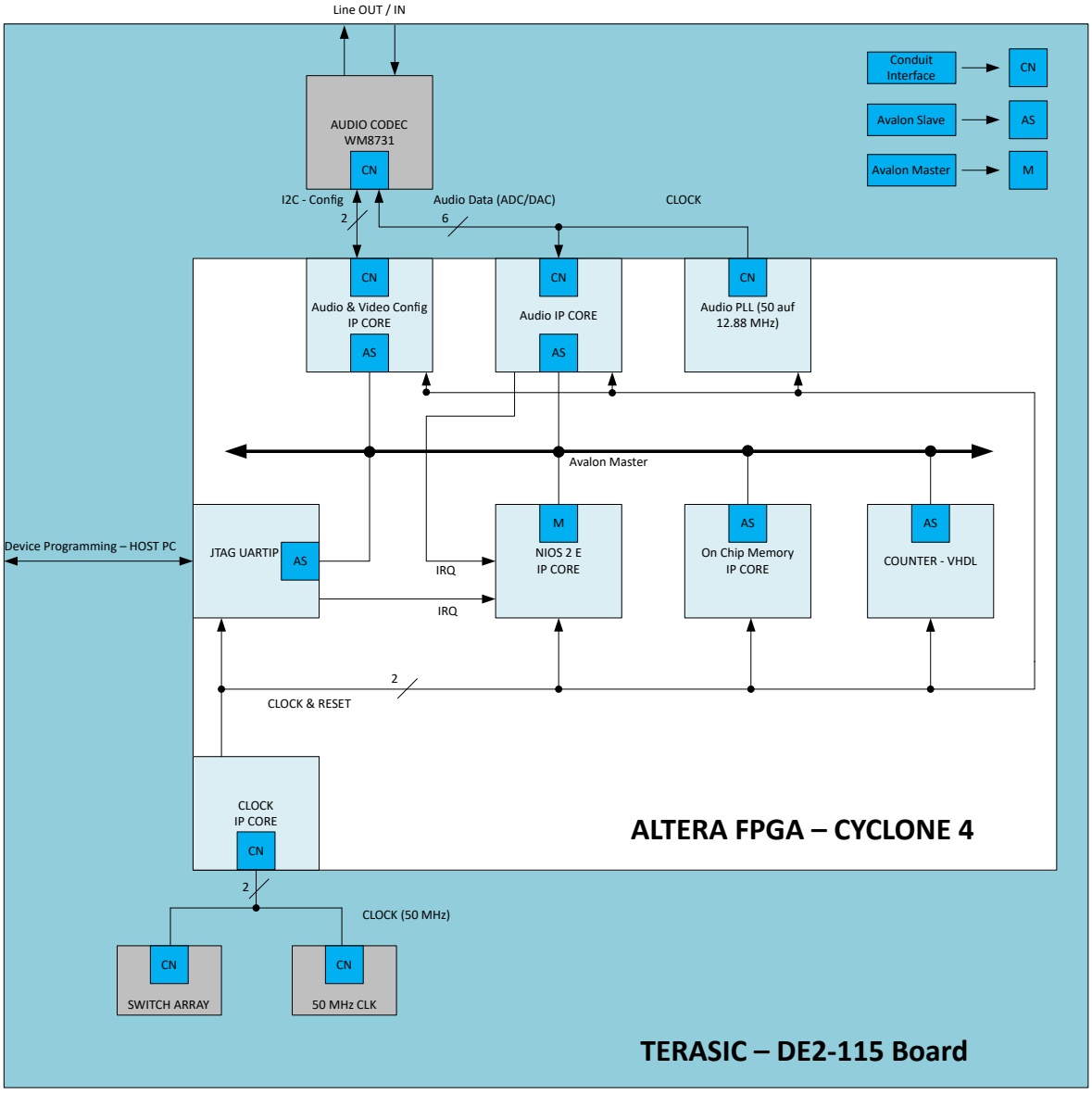

Abbildung 3.6: Schematischer Aufbau der einzelnen Komponenten und deren Wechselwirkung auf FPGA, Board und externer Ebene.

4 Quartus - Top Entity und Integration der Komponenten

Das in Kapitel 3 vorgestellte und entwickelte System muss nun innerhalb von Quartus in ein entsprechendes Projekt eingebunden werden. Weiterhin muss das Instanziierungstemplate innerhalb der Top Entity des Projektes eingebettet werden und mit den nötigen Signalen versorgt werden. Die nötigen Schritte sind nachfolgend aufgelistet:

1. **Quartus Projekt anlegen - VHDL Top Entity**
   - Projekt öffnen mit VHDL-Top Entity für die Board Komponenten, also dem Pin-Mapping (Clock, LEDs, I2C, uvm.). Über ein mitgeliefertes Tool von Terasic (System Builder) kann ein solches File, jedoch in Verilog, einfach erstellt werden. Hier können alle benötigten Hardware Komponenten über eine GUI-Applikation ausgewählt werden. Anschließend kann über generate ein Quartus Projekt erstellt werden. In diesem ist die Top Entity in Verilog vorhanden. Dieses File kann einfach in VHDL übersetzt werden. Ein bereits übersetztes VHDL File kann dem Anhang entnommen werden (siehe Anhang).
2. **Instanziierungstemplate in Top Entity einfügen**
   - Einfügen des im Qsys kopierten Instanziierungstemplates in die Architecture des Top Files. Hier müssen den Signalen des Prozessors nun noch die externen Pins zugewiesen werden (CLOCK, RESET, etc). Der Reset soll in diesem Projekt kontrolliert durch das Betätigen eines Switches ausgelöst werden. Dadurch können eventuell auftretende Probleme bei der Programmierung unterbunden werden. Es muss darauf geachtet werden, ob der Switch am Prozessor high oder low aktiv ist.
3. **QSYS VHDL-Files einbinden**
   - Im Projekt Verzeichnis (Links), Files anzeigen. Rechtsklick auf Projekt -> add Files. Zum Verzeichnis des NIOS navigieren und das **.qip** File einbinden (hier sind ALLE Komponenten drin).
   - Gegebenenfalls eigene Komponenten hinzufügen (hier Counter Entity).
4. **Synthetisieren und auf das FPGA-Board übertragen**
   - Alles kompilieren und Synthetisieren (Fehler ausmerzen).
   - Erzeugtes SOF-File auf das FPGA-Board übertragen.

5 Programmierung des Softcore Prozessors in C

Diese Kapitel erläutert die Grundlagen zur Programmierung des Softcore Prozessors (NIOS II) in der Programmiersprache C. Altera liefert bei der Installation von Quartus ein Software Buildtool, welches auf Eclipse basiert, zur Generierung von C-files bzw. Projekten für den NIOS II. Innerhalb von dieser Entwicklungsumgebung kann frei in C programmiert werden und das kompilierte Projekt kann über diese IDE auf das FPGA Boards, respektive den Prozessor übertragen werden.

5.1 Neues Projekt erstellen

1. **Öffnen der Software - Eclipse**
   - Öffnen der Software NIOS 2 Software Build Tools (Eclipse Makro).
   - Hier ein neues Projekt anlegen. Es empfiehlt sich eine vordefinierte Applikation inklusive BSP (Board Support Packages) zu öffnen und diese entsprechend anzupassen. Hier muss nun das SOPC file des NIOS eingebunden werden. Die Software generiert nun aus alle benötigten Abhängigkeiten die Board Support Packages und bindet diese entsprechend ein. Es werden zwei Source Ordner im Verzeichnis angelegt. Das eigentliche Projekt, in welchem der eigentliche Applikations- Code eingebettet wird und die benötigten Header, als auch ein Ordner der das Kürzel *bsp* angehängt hat. Dieser enthält Treiber u.ä. welche für den Betrieb des NIOS und des Boards benötigt werden.

2. **Code bearbeiten und BSP-aktualisieren**
   - Vor der eigentlichen Programmierung sollten die BSP (Board Support Packages) auf den aktuellsten Stand gebracht werden. Dieser Schritt ist sehr wichtig und sollte **immer** vor einer Übertragung des Codes auf den Zielprozessor geschehen. Das zweite Verzeichnis (*GenerischerName_bsp* anwählen mit einem Rechtsklick. Dann unter NIOS 2 -> Generate BSP anwählen.
   - Nun kann das Main file bearbeitet werden. Innerhalb dieser Applikation wurde ein FFT.Alg eingebettet. Dafür muss die FIFO des Codecs Software-seitig ausgelesen werden, in einem entsprechenden Buffer abgelegt werden und bei erreichen der Sample Schwelle an den FFT-Algorithmus übergeben werden. Weiterhin

wird in dieser Routine die Funktionsweise und damit die Konfiguration des Audiocodec durchgeführt. Der Codec wird auf einen 24 Bit Modus eingestellt. Die Samples werden aktuell lediglich vom rechten Kanal des Codecs herangezogen. Die Software enthält eine kleine Routine, welche auf eine Eingabe in der Konsole wartet, wodurch der Auslesevorgang gestartet wird. Außerdem wird hierdurch der Counter gestartet, dieser gibt ein visuelles Feedback auf dem Board, damit sich ein Nutzer von der Funktion überzeugen kann bzw. als Debugg Anzeige. Der vollständige Code kann dem Anhang entnommen werden.

Es ist an dieser Stelle wichtig zu erwähnen, dass der verfügbare FFT. Algorithmus sehr langsam in der Ausführung für größere Anzahlen an Samples war, weshalb ein FFT Algorithmus aus Matlab implementiert wurde. Matlab besitzt ein Tool (Matlab Coder), durch welches Matlab-Funktionen in C-Code konvertiert werden können. Durch dieses Tool wurde eine deutlich schnellere Variante des FFT Alg. nach Cooley und Tukey auf dem NIOS implementiert.

```
/*
 * NIOS FFT Audio
 *
 * This example calculates the dominante frequency of an Audiostream. It runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It runs with or without the MicroC/OS-II RTOS and requires a STDOUT
 * device in your system's hardware.
 *
 * The Qsys System design needs to have an Audio-configdevice in order to work right.
 *
 */
//Author: Lukas Woehle Version 2.1 (08.02.2017)




/*****************************************************/
#define samples 1024;   // Number of Samples for FFT
/*****************************************************/


#include <math.h>
#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "altera_up_avalon_audio.h"
#include "altera_up_avalon_audio_and_video_config.h"
#include "AUDIO_CODEC_REGISTER_CONFIG.h"
#include "FFT_Matlab.h"


int main(void)
{   int Samples=samples;  // Samples
    int count=0xFF;       // Counter var for LEDs
    int CNT_Start=0;      // Var to hold State (Start/Stop)
    int x=1;              // Break var for while loop
    int i=0;              // Samplecount


    alt_up_av_config_dev* Audio_conf; //Opens the Audio/Video Configuration device specified by name
        .
```

```
42      alt_up_audio_dev * audio_dev;    //Opens the Audio device
43
44
45      signed int Buff_R[1024]={0};     // Buffer for Audiosamples
46      double Buff_R_doub[1024]={0};    // Conversion Buffer (signed to double)
47      double Dominante;                // Output Var for dominant Frequency
48
49      // open the config Port
50
51      Audio_conf = alt_up_av_config_open_dev("/dev/audio_and_video_config_0");
52
53      // Configure Audio-Codec - see documentation and AUDIO_CODEC_REGISTER_CONFIG.h
54      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_00 ,REG_00_DEFAULT);
55      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_01 ,REG_01_DEFAULT);
56      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_02 ,REG_02_DEFAULT);
57      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_03 ,REG_03_DEFAULT);
58      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_04 ,REG_04_DEFAULT);
59      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_05 ,REG_05_DEFAULT);
60      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_06 ,REG_06_DEFAULT);
61      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_07 ,REG_07_DEFAULT);
62      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_08 ,REG_08_DEFAULT);
63      alt_up_av_config_write_audio_cfg_register(Audio_conf, REG_09 ,REG_09_DEFAULT);
64
65      // open the Audio port
66      audio_dev = alt_up_audio_open_dev ("/dev/audio_0");
67
68      // catch exception when Device is not present
69      if ( audio_dev == NULL)
70       printf ("Error:␣could␣not␣open␣audio␣device␣\n");
71      else
72       printf ("Opened␣audio␣device␣\n");
73
74      // Debug print
75      printf("Start␣Loopback␣with␣1,␣end␣with␣0");
76
77      // Start Counter (7-segment) and Turn off/on LEDs
78      IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED_BASE,count & 0xFF);
79                  scanf("%i",&CNT_Start);
80                  printf("%i",CNT_Start);
81                  switch (CNT_Start){
82                  case 0:
83                      IOWR_ALTERA_AVALON_PIO_DATA(CNT_EN_BASE, 0x00);
84                      IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED_BASE,0xFF);
85                      break;
86                  case 1:
87                      IOWR_ALTERA_AVALON_PIO_DATA(CNT_EN_BASE, 0x01);
88                      IOWR_ALTERA_AVALON_PIO_DATA(PIO_LED_BASE,0x00);
89                      break;
90                  default:
91                      printf("nutin");
92                  }
93      while(x==1)
94      {
95          // Read Audio Data, check if samples are available
96          int fifospace = alt_up_audio_read_fifo_avail (audio_dev, ALT_UP_AUDIO_LEFT);
97          if ( fifospace != 0 )  // check if data is available
98      {
99          if (i==Samples){      // if Samples == 1024, Start FFT
100             printf("success"); // Debug print
101 /*************************************************************************
102                 Convert unsigned integer to double
103 Codec values are only positve, Starting one (sign) is not recognized
104                 as signed - > needs to be converted
105 *************************************************************************/
106             for(int a=0;a<=Samples;a++){
```

```
107                 if (Buff_R[a] & (1<<23)) {
108                   Buff_R[a]|= (1<<31)|(1<<30)|(1<<29)|(1<<28)|(1<<27)|(1<<26)|(1<<25)|(1<<24);
109                   Buff_R_doub[a]=(double)Buff_R[a];
110                 }
111                 else{
112                    Buff_R_doub[a]=(double)Buff_R[a];
113                 }
114             }
115             i=0;   // Set FFT-Counter var to zero
116             Dominante= FFT_Matlab(Buff_R_doub); // Calculate dominante frequency
117             printf("Frequenzcy is:");       // print frequency
118             printf("%f",Dominante);
119             printf("\n");
120         }
121         // Read Audio Buffer data and save to Buffer
122         alt_up_audio_read_fifo (audio_dev, &(Buff_R[i]), 1, ALT_UP_AUDIO_LEFT);
123         // increment Counter
124         i++;
125     }
126
127 }
128
129 }
```

VHDL-Code 5.1: Main Code des Projektes

3. **Projekt kompilieren und Ausführen**

   - Sobald das BSP als auch der eigentliche Code in das Projekt eingeflossen sind, so kann das Projekt kompiliert werden. Dazu muss ein Rechtsklick auf den Source Ordner der Main Datei erfolgen und *Build Project* ausgewählt werden. Es ist möglich, dass an dieser Stelle Fehler auftreten, die nicht auf einen falschen Code zurückzuführen sind. Ein prominenter Fehler beruht auf einer zu geringen Speichergröße des NIOS. Dieser muss innerhalb von QSYS dann entsprechend erweitert werden.

     Konnte das Projekt richtig kompiliert werden (*Build succesfull*) so kann die Software auf den NIOS geflasht werden. Dazu muss erneut der Source Ordner der Main Datei mittels Rechtsklick ausgewählt werden. Nun kann unter dem Reiter *Run as* der Button *NIOS II Hardware* ausgewählt werden. Hierdurch wird der Code auf das Board und respektive den NIOS geflasht. In einigen Fällen muss die Verbindung *aufgelöst* werden. Ist dies der Fall öffnet sich ein Fenster (Run config). In diesem muss die Verbindung neu hergestellt werden durch einen Klick auf Connections und anschließend auf den Button Resolve Connection. Sobald die Verbindung bzw. der Zielprozessor gefunden wurde kann durch einen Klick auf Run die Applikation auf dem Softcore ausgeführt werden.

     > Es kann unter Umständen zu einem Fehler kommen (*.elf could not...*) welcher auf die falsche Einstellung der Reset Leitung zurückzuführen ist. Falls der Reset auf einen Switch geführt wurde, sollte versucht werden, die Switch Stellung zu verändern.

Nun sollte die Software geflasht werden können.

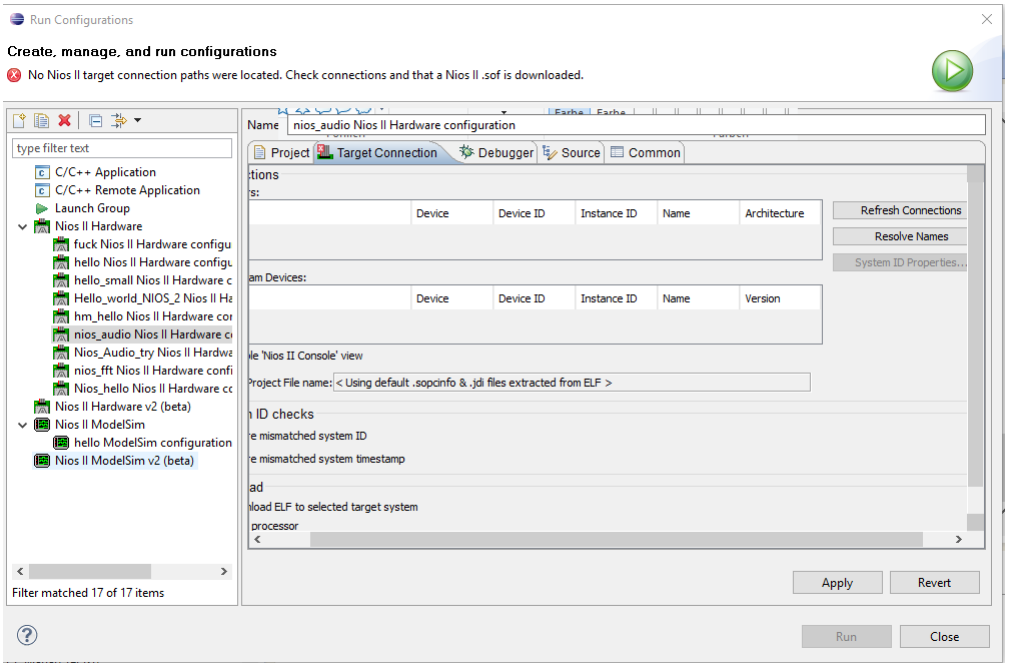

Abbildung 5.1: Run config-Fenster

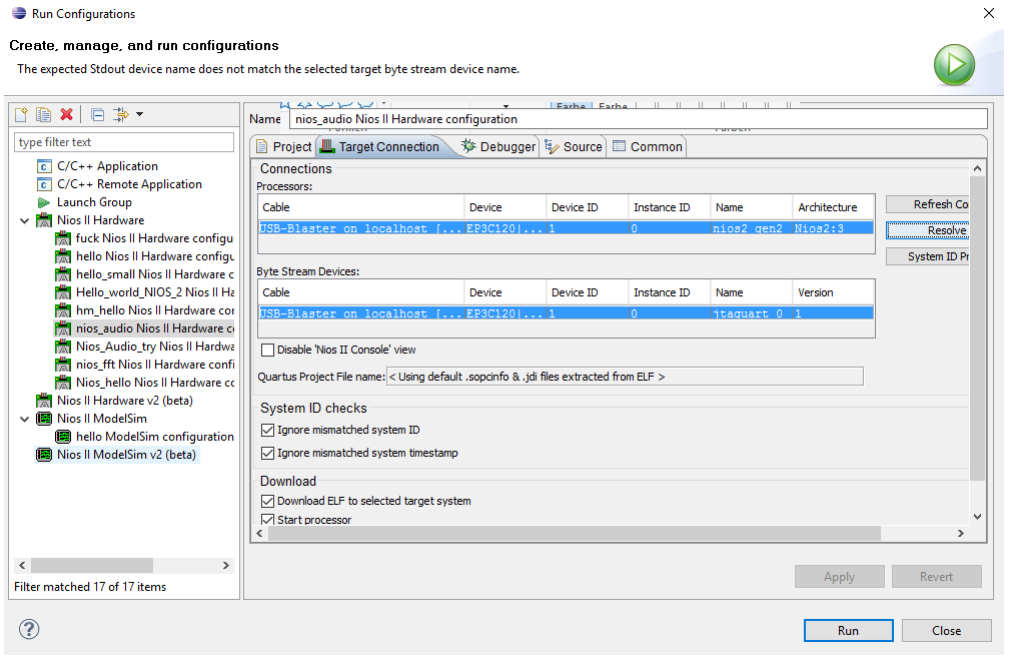

Abbildung 5.2: Verbindung hergestellt mit dem Zielsystem. Nun kann die Applikation durch Run auf dem Zielsystem ausgeführt werden.

4. **Betrieb**

   - Sobald die Software erfolgreich übertragen wurde, so sollten die grünen LEDs des Terasic Boards leuchten und zwei 7-Segment Anzeigen sollten jeweils eine 0 anzeigen. Nun kann über den Host-PC ein Startkommando gegeben werden(*1 über die Konsole an den NIOS senden*) wodurch die FFT und der Counter gestartet werden. Nach einem Durchlauf wird dann die Dominante der Frequenz angezeigt. Die Anzeige der Frequenz erfolgt in der aktuellen Version auf der Konsole des Host-PC. Nachfolgend kann die Ausgabe des Prozessors bei der Beaufschlagung des Audiocodecs mit einem 1 kHz Sinus eingesehen werden:

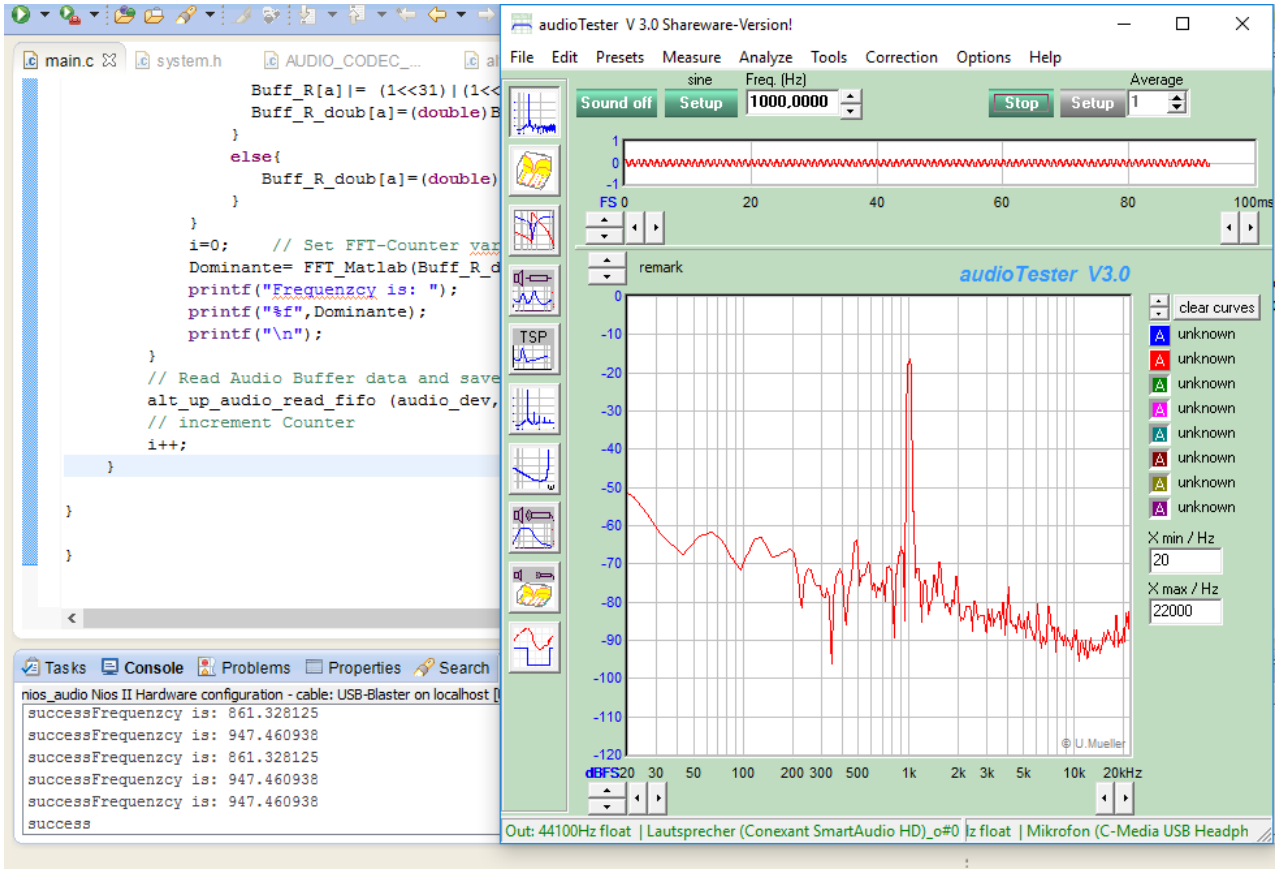

Abbildung 5.3: FFT Ergebnis bei Beaufschlagung des Codecs mit einem 1 kHz Sinus Signals.

6 Fazit

Die in Kapitel 2 beschriebene Aufgabenstellung konnte auf dem Zielsystem (FPGA Cyclone 4) umgesetzt werden. Allerdings stellte sich die Suche nach geeigneter Literatur zur Implementierung der Softcore CPU (NIOS) als sehr schwierig heraus. Das System ist daher bei weitem nicht an der Leistungsgrenze und könnte durch eine Hardware FFT auf dem FPGA deutlich schneller gestaltet werden. Jedoch nimmt die Hardware deutlich mehr Fläche ein, wodurch eine große Anzahl an Samples hierbei ebenfalls zu Problemen führen kann.

Bei der Evaluation des fertigen FFT-Alg. zeigte sich schnell, dass dieser sehr langsam war. Bei einer Sample Anzahl von 1024 dauerte die Auswertung bereits einige Sekunden. Diese Zeit konnte deutlich reduziert werden durch die Matlab FFT respektive den generierten C-Code konnte die Ausführungszeit deutlich reduziert werden (allerdings immer noch vergleichsweise langsam). Die Ausführungszeit könnte durch einen angepassten Algorithmus deutlich angehoben werden. Weiterhin besteht die Möglichkeit die schneller Variante des NIOS zu verwenden (Version f). Diese muss allerdings innerhalb eines Lizenz Paketes erworben werden, damit diese als Standalone Lösung funktioniert.

Weiterhin könnte eine Schnittstelle geschaffen werden, welche es ermöglicht die FFT-Daten innerhalb von Matlab oder einer anderen Grafischen Applikation anzuzeigen bzw. diese weiter zu nutzen. Dafür kann die Blaster Schnittstelle genutzt werden, allerdings mit signifikantem Mehraufwand. Eine andere Lösung besteht darin die Daten auf dem vorhandenen LCD des Terasic Boards anzuzeigen.

Literaturverzeichnis

[Altera Corporation, 2010] Altera Corporation (2010). Sopc builder user guide. https://www.altera.com/en_US/pdfs/literature/ug/ug_sopc_builder.pdf.

[Altera Corporation, 2011a] Altera Corporation (2011a). Audio core for altera de-series boards: For quartus ii 11.0. ftp://ftp.altera.com/up/pub/Altera_Material/11.1/University_Program_IP_Cores/Audio_Video/Audio.pdf.

[Altera Corporation, 2011b] Altera Corporation (2011b). Audio/video configuration core for de-series boards: For quartus ii 11.0. ftp://ftp.altera.com/up/pub/Altera_Material/11.0/University_Program_IP_Cores/Audio_Video/Audio_and_Video_Config.pdf.

[Altera Corporation, 2015] Altera Corporation (2015). Avalon interface specifications. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf.

[Wolfson microelectronics, 2004] Wolfson microelectronics (2004). Portable internet audio codec with headphone driver and programmable sample rates: (wm8731 rev3.4). https://www.cirrus.com/cn/pubs/proDatasheet/WM8731_v4.9.pdf.

7 ANHANG

7.1 VHDL Top Entity

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


entity top is
port(



   --//=======================================================
   --// PORT declarations
   --//=======================================================
--VCC_t : in std_logic:='1';

   --////////////// CLOCK //////////
CLOCK_50 : in std_logic;
CLOCK2_50 : in std_logic;
CLOCK3_50 : in std_logic;

   --////////////// Sma //////////
SMA_CLKIN : in std_logic;
SMA_CLKOUT : out std_logic;

   --////////////// LED //////////
LEDG: out std_logic_vector(8 downto 0);
LEDR: out std_logic_vector(17 downto 0);

   --////////////// KEY //////////
KEY : in std_logic_vector(3 downto 0);

   --////////////// EX_IO //////////
EX_IO: inout std_logic_vector(6 downto 0);

   --////////////// SW //////////
SW: in std_logic_vector(17 downto 0);

   --////////////// SEG7 //////////
      HEX0 :out std_logic_vector(6 downto 0);
      HEX1 :out std_logic_vector(6 downto 0);
      HEX2 : std_logic_vector(6 downto 0);
      HEX3 : std_logic_vector(6 downto 0);
      HEX4 : std_logic_vector(6 downto 0);
      HEX5 : std_logic_vector(6 downto 0);
      HEX6 : std_logic_vector(6 downto 0);
      HEX7 : std_logic_vector(6 downto 0);

   --////////////// LCD //////////
LCD_BLON : out std_logic;
LCD_DATA: inout std_logic_vector(7 downto 0);
LCD_EN : out std_logic;
```

```
51  LCD_ON : out std_logic;
52  LCD_RS : out std_logic;
53  LCD_RW : out std_logic;
54
55     --////////////// RS232 ///////////
56  UART_CTS: in std_logic;
57  UART_RTS : out std_logic;
58  UART_RXD : in std_logic;
59  UART_TXD : out std_logic;
60
61     --////////////// PS2 for Keyboard and Mouse ///////////
62  PS2_CLK : inout std_logic;
63  PS2_CLK2 : inout std_logic;
64  PS2_DAT : inout std_logic;
65  PS2_DAT2 : inout std_logic;
66
67     --////////////// SDCARD ///////////
68  SD_CLK : out std_logic;
69  SD_CMD : inout std_logic;
70  SD_DAT: inout std_logic_vector(3 downto 0);
71  SD_WP_N : in std_logic;
72
73     --////////////// VGA ///////////
74  VGA_B : out std_logic_vector(7 downto 0) ;
75  VGA_BLANK_N : out std_logic;
76  VGA_CLK : out std_logic;
77  VGA_G : out std_logic_vector(7 downto 0);
78  VGA_HS : out std_logic;
79  VGA_R : out std_logic_vector(7 downto 0);
80  VGA_SYNC_N : out std_logic;
81  VGA_VS : out std_logic;
82
83     --////////////// Audio ///////////
84  AUD_ADCDAT : in std_logic;
85  AUD_ADCLRCK : inout std_logic;
86  AUD_BCLK : inout std_logic;
87  AUD_DACDAT : out std_logic;
88  AUD_DACLRCK : inout std_logic;
89  AUD_XCK : out std_logic;
90
91     --////////////// I2C for EEPROM ///////////
92  EEP_I2C_SCLK : out std_logic;
93  EEP_I2C_SDAT : inout std_logic;
94
95     --////////////// I2C for Audio Tv-Decoder ///////////
96  I2C_SCLK : out std_logic;
97  I2C_SDAT : inout std_logic;
98
99     --////////////// Ethernet 0 ///////////
100 ENET0_GTX_CLK : out std_logic;
101 ENET0_INT_N: in std_logic;
102 ENET0_LINK100: in std_logic;
103 ENET0_MDC : out std_logic;
104 ENET0_MDIO : inout std_logic;
105 ENET0_RST_N : out std_logic;
106 ENET0_RX_CLK: in std_logic;
107 ENET0_RX_COL: in std_logic;
108 ENET0_RX_CRS: in std_logic;
109 ENET0_RX_DATA: in std_logic_vector(3 downto 0);
110 ENET0_RX_DV: in std_logic;
111 ENET0_RX_ER: in std_logic;
112 ENET0_TX_CLK: in std_logic;
113 ENET0_TX_DATA: out std_logic_vector(3 downto 0);
114 ENET0_TX_EN : out std_logic;
115 ENET0_TX_ER : out std_logic;
```

```
116 ENETCLK_25: in std_logic;
117
118    --////////////// Ethernet 1 //////////
119 ENET1_GTX_CLK : out std_logic;
120 ENET1_INT_N: in std_logic;
121 ENET1_LINK100: in std_logic;
122 ENET1_MDC : out std_logic;
123 ENET1_MDIO : inout std_logic;
124 ENET1_RST_N : out std_logic;
125 ENET1_RX_CLK: in std_logic;
126 ENET1_RX_COL: in std_logic;
127 ENET1_RX_CRS: in std_logic;
128 ENET1_RX_DATA: in std_logic_vector(3 downto 0);
129 ENET1_RX_DV: in std_logic;
130 ENET1_RX_ER: in std_logic;
131 ENET1_TX_CLK: in std_logic;
132 ENET1_TX_DATA: out std_logic_vector(3 downto 0);
133 ENET1_TX_EN : out std_logic;
134 ENET1_TX_ER : out std_logic;
135
136    --////////////// TV Decoder //////////
137 TD_CLK27: in std_logic;
138 TD_DATA : in std_logic_vector(7 downto 0);
139 TD_HS: in std_logic;
140 TD_RESET_N : out std_logic;
141 TD_VS: in std_logic;
142
143    --////////////// USB 2.0 OTG (Cypress CY7C67200) //////////
144 OTG_ADDR: out std_logic_vector(1 downto 0);
145 OTG_CS_N : out std_logic;
146 OTG_DATA : inout std_logic_vector(15 downto 0);
147 OTG_INT: in std_logic;
148 OTG_RD_N : out std_logic;
149 OTG_RST_N : out std_logic;
150 OTG_WE_N : out std_logic;
151
152    --////////////// IR Receiver //////////
153 IRDA_RXD: in std_logic;
154
155    --////////////// SDRAM //////////
156 DRAM_ADDR: out std_logic_vector(12 downto 0);
157 DRAM_BA: out std_logic_vector(1 downto 0);
158 DRAM_CAS_N : out std_logic;
159 DRAM_CKE : out std_logic;
160 DRAM_CLK : out std_logic;
161 DRAM_CS_N : out std_logic;
162 DRAM_DQ: inout std_logic_vector(31 downto 0);
163 DRAM_DQM : out std_logic_vector(3 downto 0);
164 DRAM_RAS_N : out std_logic;
165 DRAM_WE_N : out std_logic;
166
167    --////////////// SRAM //////////
168 SRAM_ADDR: out std_logic_vector(19 downto 0);
169 SRAM_CE_N : out std_logic;
170 SRAM_DQ: inout std_logic_vector(15 downto 0);
171 SRAM_LB_N : out std_logic;
172 SRAM_OE_N : out std_logic;
173 SRAM_UB_N : out std_logic;
174 SRAM_WE_N : out std_logic;
175
176    --////////////// Flash //////////
177 FL_ADDR: out std_logic_vector(22 downto 0);
178 FL_CE_N : out std_logic;
179 FL_DQ : inout std_logic_vector(7 downto 0);
180 FL_OE_N : out std_logic;
```

```
181 FL_RST_N : out std_logic;
182 FL_RY: in std_logic;
183 FL_WE_N : out std_logic;
184 FL_WP_N : out std_logic;
185
186 --GPIO
187 GPIO: inout std_logic_vector(35 downto 0));
188 end top;
189
190 architecture BEH_NIOS of top is
191
192
193 end BEH_NIOS;
194
195 -- //=======================================================
196 -- // REG/WIRE declarations
197 -- //=======================================================
198
199
200
201
202 -- //=======================================================
203 -- // Structural coding
204 -- //=======================================================
```

VHDL-Code 7.1: VHDL Top Entity für das DE 115-2 Board

7.2 VHDL Instanziierung

```
1  architecture BEH_NIOS of top is
2  signal EN_able : std_logic := '0';
3  signal I_zw : std_logic_vector(7 downto 0);
4  signal s_AUD_ADCDAT : std_logic;
5  signal s_AUD_ADCLRCK : std_logic;
6  signal s_AUD_BCLK :  std_logic;
7  signal s_AUD_DACDAT : std_logic;
8  signal s_AUD_DACLRCK : std_logic;
9
10
11
12
13    component NIOS2_tut_audio is
14        port (
15            clk_clk                                 : in  std_logic                     := 'X';           --
                  clk
16            cnt_en_external_connection_export       : out std_logic;                                      --
                  export
17            counter_out_external_connection_export  : in  std_logic_vector(7 downto 0) := (others => 'X
                  '); -- export
18            pio_0_external_connection_export        : out std_logic_vector(7 downto 0);                   --
                  export
19            reset_reset_n                           : in  std_logic                     := 'X';           --
                  reset_n
20            audio_0_external_interface_ADCDAT       : in  std_logic                     := 'X';           --
                  ADCDAT
21            audio_0_external_interface_ADCLRCK      : in  std_logic                     := 'X';           --
                  ADCLRCK
22            audio_0_external_interface_BCLK         : in  std_logic                     := 'X';           --
                  BCLK
23            audio_0_external_interface_DACDAT       : out std_logic;                                      --
                  DACDAT
24            audio_0_external_interface_DACLRCK      : in  std_logic                     := 'X';           --
                  DACLRCK
```

```
25            audio_and_video_config_0_external_interface_SDAT : inout std_logic           := 'X';          --
                  SDAT
26            audio_and_video_config_0_external_interface_SCLK : out std_logic ;
                  -- SCLK
27            audio_pll_0_audio_clk_clk                  : out  std_logic ;
                  -- clk
28            vol_en_external_connection_export          : in   std_logic                 := 'X'           --
                  export
29            );
30     end component NIOS2_tut_audio;
31
32
33    component Counter_Simple_HEX is
34       port(
35          CLOCK1: in std_logic;
36          LEDG1:  out std_logic_vector(7 downto 0); -- OUT Bit Vektor
37          SW1:    in std_logic;
38          outi:   out std_logic_vector(7 downto 0);
39          I_out:  out std_logic;
40          HEX_n:  out std_logic_vector(6 downto 0);
41          HEX_n2:  out std_logic_vector(6 downto 0);
42          EN_C:   in std_logic
43          );
44    end component Counter_Simple_HEX;
45
46
47
48
49 begin
50
51 u1 : component Counter_Simple_HEX
52          port map (
53             CLOCK1 => CLOCK_50,
54             LEDG1 => LEDR(7 downto 0),
55             outi => I_zw,
56             SW1 => SW(0),
57             HEX_n=> HEX0,
58             HEX_n2=> HEX1,
59             EN_C => EN_able
60          );
61
62
63    s_AUD_ADCDAT<=AUD_ADCDAT;
64    s_AUD_ADCLRCK<=AUD_ADCLRCK;
65    AUD_DACDAT<=s_AUD_ADCDAT;
66    AUD_DACLRCK<=s_AUD_ADCLRCK;
67
68 u0 : component NIOS2_tut_audio
69          port map (
70             clk_clk                                  => CLOCK_50,      --clk.clk
71             cnt_en_external_connection_export        => EN_able,       --cnt_en_external_connection.export
72             counter_out_external_connection_export   => I_zw,          --counter_out_external_connection.
                   export
73             pio_0_external_connection_export         => LEDG(7 downto 0),--pio_0_external_connection.
                   export
74             reset_reset_n                            => not(SW(0)),    --reset.reset_n
75             audio_0_external_interface_ADCDAT        => s_AUD_ADCDAT,  --audio_0_external_interface.ADCDAT
76             audio_0_external_interface_ADCLRCK       => s_AUD_ADCLRCK, --.ADCLRCK
77             audio_0_external_interface_BCLK          => AUD_BCLK,      --.BCLK
78             audio_0_external_interface_DACDAT        => s_AUD_DACDAT,  --.DACDAT
79             audio_0_external_interface_DACLRCK       => s_AUD_DACLRCK, --.DACLRCK
80             audio_and_video_config_0_external_interface_SDAT => I2C_SDAT,   --
                   audio_and_video_config_0_external_interface.SDAT
81             audio_and_video_config_0_external_interface_SCLK => I2C_SCLK,    --.SCLK
82             audio_pll_0_audio_clk_clk                => AUD_XCK ,      --audio_pll_0_audio_clk.clk
```

```
83            vol_en_external_connection_export           => not(SW(1))     --vol_en_external_connection.export
84        );
85 end BEH_NIOS;
```

VHDL-Code 7.2: Vollständige Instanziierung inklusive Signalzuweisung

7.3 Audio Register Header

```
1  // HEADER for CODEC INIT --
2  // Define Register Adress (WM8731)
3  #define REG_00 0x00
4  #define REG_01 0x01
5  #define REG_02 0x02
6  #define REG_03 0x03
7  #define REG_04 0x04
8  #define REG_05 0x05
9  #define REG_06 0x06
10 #define REG_07 0x07
11 #define REG_08 0x08
12 #define REG_09 0x09
13
14 // CODEC Config - see WM 8731 Datasheet for more Information
15 #define REG_00_DEFAULT 0b000010111
16 #define REG_01_DEFAULT 0b000010111
17 #define REG_02_DEFAULT 0b001111001
18 #define REG_03_DEFAULT 0b001111001
19 #define REG_04_DEFAULT 0b000010010
20 #define REG_05_DEFAULT 0b000000000
21 #define REG_06_DEFAULT 0b000000010
22 #define REG_07_DEFAULT 0b001000011
23 #define REG_08_DEFAULT 0b000000000
24 #define REG_09_DEFAULT 0b000000001
```

VHDL-Code 7.3: Register Definitionen des Audio Codec WM8731