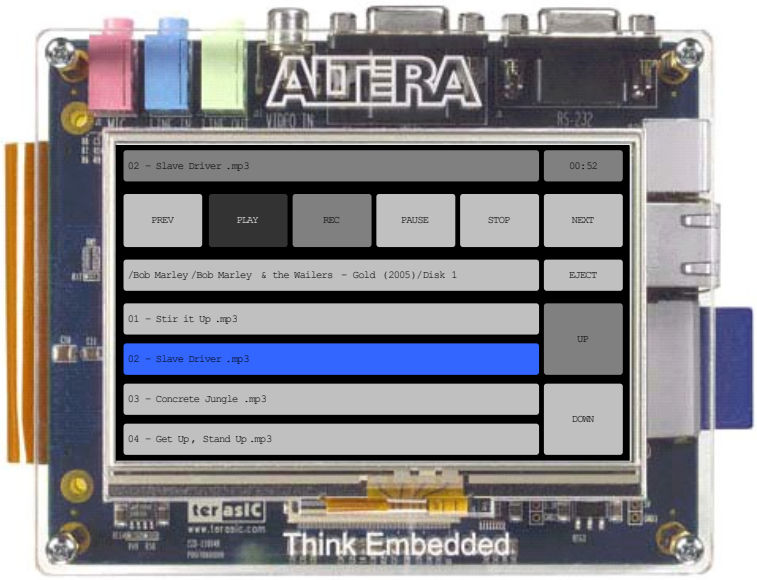
Fachhochschule Bielefeld
University of Applied Sciences

Diplomarbeit

Entwicklung einer

FPGA basierten SOPC Audio-Plattform mit konfigurierbarem Steuerinterface

Birger Zimmermann



Erstprüfer: Prof. Dr. Ing. Norbert Schmidt

Zweitprüfer: Prof. Dr. math. Wolfgang Bunse

Danksagung

An dieser Stelle möchte ich allen herzlich danken, die mich bei der Anfertigung dieser Diplomarbeit unterstützt haben.

Ein ganz besonderer Dank gilt meinen beiden Prüfern, Herrn Prof. Dr. Ing. Norbert Schmidt, der mich durch seine hilfreichen Anregungen sehr unterstützt hat, sowie Herrn Prof. Dr. math. Wolfgang Bunse für seine Bereitschaft, als Zweitgutachter zu fungieren.

Darüber hinaus danke ich Herrn Dipl.-Ing. Frank Engert für die Unterstützung bei der Hardwarebeschaffung.

Kurzfassung

Gegenstand der hier vorliegenden Arbeit ist die Entwicklung einer Audio-Plattform, die aus Hard- und Software-Komponenten besteht. Das entwickelte System beinhaltet Funktionen zum Aufnehmen und Abspielen von Audiodaten in CD-Qualität. Die Integration eines speziell optimierten FAT-Filesystems ermöglicht eine auf SD-Karten basierende Datenspeicherung. Der integrierte MP3-Decoder kann für Multimedia-Anwendungen eingesetzt werden. Die veränderbare Hardwarebasis (FPGA) ermöglicht verschiedenen Applikationen ein konfigurierbares Steuerinterface. Zur Darstellung der Systemfunktionen wurde eine Jukebox als Referenzanwendung entwickelt. Diese ist mit einem grafischen Benutzerinterface ausgestattet und erlaubt das Abspielen und Aufnehmen von WAV- sowie die Wiedergabe von MP3-Dateien. Der technische Kern der Plattform besteht aus einem FPGA, auf dem das entwickelte System-On-a-Chip (SOC) mit integriertem Softcore-Prozessor die Steuerung einer Anwendung übernimmt. Speziell in dieser Arbeit entworfene IP-Cores unterstützen den Prozessor bei der Kommunikation mit den angeschlossenen Peripheriebausteinen. Eine durch Custom-Instructions realisierte ALU erweitert den Prozessor um mathematische Befehle und ermöglicht damit die MP3-Decodierung in Echtzeit. Im Gegensatz zu vorgefertigten Komponenten wie MP3-Dekodier-Bausteinen oder mikrocontrollerbasierten Lösungen, bietet diese Plattform durch den Einsatz der FPGA Technik ein großes Maß an Flexibilität. Dies erlaubt ein kosten- und zeitoptimiertes System-Design.

Abstract

Subject of this diploma thesis is the development of a FPGA based embeddable audio platform. It provides functions for audio recording and playback in CD-quality. The optimized FAT file system implementation allows data storage on SD-cards. An integrated MP3 decoder can be used for multimedia application development. The programmable hardware (FPGA) programmed with a full featured system-on-a-chip provides a configurable control interface for different contexts. A jukebox reference application was built to proof the flexibility and capabilities of the developed platform. The application contains a graphical user interface and functions to play and record audio files. Specially designed IP-cores support the embedded soft-core processor in the communication with the peripherals. Custom instructions enhance the power of the processor for MP3-playback. In contrast to off-the-shelf components like MP3 decoder chips and micro controller solutions, this platform provides a high degree of flexibility by using the FPGA technology. This enables cost effective and time optimized system design.

Inhaltsverzeichnis

Inhaltsverzeichnis....1

1 Einleitung....4

1.1 Ziel....4

1.2 Stand der Technik und vorhandene Lösungen....5

1.3 Aufbau der Dokumentation....5

2 Instrumentarium und Methodik....6

2.1 Entwicklungssystem....6

2.2 Schnelleinstieg mit „Hello World“ Programm....7

2.3 Hardware Design....9

2.3.1 Die Hardwarebeschreibungssprache Verilog....9

2.3.2 Quartus II EDA Suite....9

2.3.3 IP-Core Design....10

2.3.4 System Design....12

2.3.5 In-System-Programming....15

2.3.6 Verifikation....15

2.4 Softwareentwicklung....16

2.4.1 Programmiersprache....16

2.4.2 Nios II- Entwicklungsumgebung....17

2.4.3 Treiber Entwicklung....17

2.4.4 System Library Project....17

2.4.5 Performance-Messung mit GNU-Profiler und Performance Counter....18

2.4.6 Fehlersuche - Debugging....19

2.4.7 Software Entwicklungsprozess....19

3 Konzept....20

4 IP-Core Entwicklung....21

4.1 SPI-Master IP-Core....21

4.1.1 Ziel....22

4.1.2 Konzept....22

4.1.3 Implementierung....24

4.1.4 Ergebnis....29

4.2 WM8731 IP-Core....31

4.2.1 Ziel....32
4.2.2 Konzept....32
4.2.3 Implementierung....33
4.2.4 Ergebnis....40
4.3 64-Bit-ALU IP-Core....41
4.3.1 Ziel....41
4.3.2 Konzept....41
4.3.3 Implementierung....42
4.3.4 Ergebnis....45
4.4 Opencores.org IPs....46
5 Systementwicklung....47
5.1 Konzept....47
5.2 Implementierung....48
5.2.1 Projektstruktur....48
5.2.2 SOPC-Builder-Projekt....49
5.2.3 System-Generierung und Synthese (Kompilierung)....52
5.2.4 FPGA-Ressourcen-Verbrauch....53
5.3 Ergebnis....53
6 Softwareentwicklung....54
6.1 Audio-Plattform-Softwarebibliothek....54
6.1.1 Datenspeicherung....55
6.1.2 Audio....58
6.1.3 Grafisches Benutzerinterface....61
6.1.4 Steuerung....64
6.1.5 Integration der Softwarebibliothek....65
6.2 MP3-Decoder Integration....66
6.2.1 Implementierung....66
6.2.2 Verwendungshinweise....68
6.2.3 Ergebnis....68
6.3 Entwicklung der „open juke“-Referenzanwendung....69
6.3.1 Ziel....69
6.3.2 Konzept....69
6.3.3 Zustandsautomat....70
6.3.4 Implementierung....70
6.3.5 Ergebnis....72

7 Zusammenfassung und Ausblick....73

Anhang....74

Abbildungsverzeichnis....194

Quelltextverzeichnis....197

Tabellenverzeichnis....199

Abkürzungsverzeichnis....201

Literaturverzeichnis....203

1 Einleitung

Eingebettete Audioapplikationen, z.B. eingesetzt als Warnsystem im Auto oder Instruktionssystem an Maschinen, benötigen spezielle, von örtlichen Gegebenheiten abhängige Steuerungs- und Kommunikationsschnittstellen. Dies können einfache Taster oder komplexe Busanbindungen, wie der CAN- oder LIN-Bus, sein. Um den Entwicklungsaufwand für neue Systeme zu reduzieren und damit die Kosten zu senken, ist es sinnvoll eine möglichst flexible und konfigurierbare technische Basis für vielfältige Anwendungen zu schaffen. Eine Plattform aus Hard- und Software-Komponenten, die grundlegende, in diesem Umfeld benötigte Funktionen in modularer Form bereitstellt. Die Zielgruppe dieses Frameworks sind Anwendungsentwickler, Produktdesigner und Ingenieure der Elektrotechnik, die das System zur Implementierung spezieller Anwendungen nutzen und erweitern können.

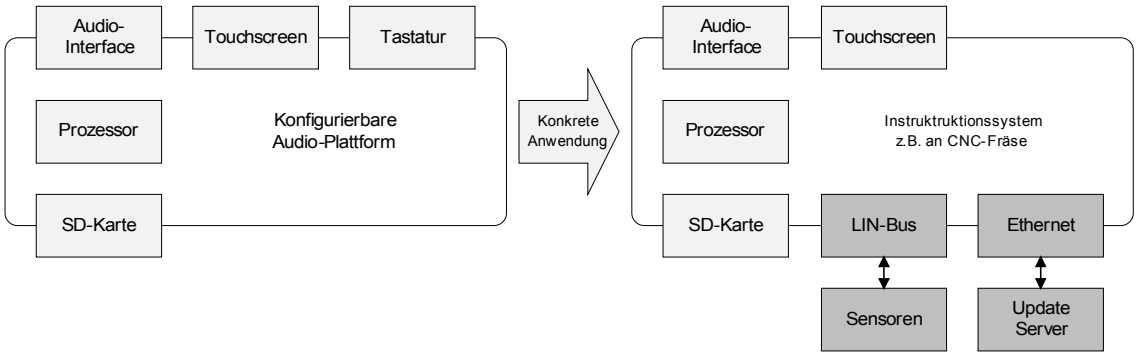


Abbildung 1: Implementierungsbeispiel Instruktionssystem

1.1 Ziel

Das Ziel der vorliegenden Arbeit ist die Realisierung der oben beschriebenen Plattform als System-on-a-Chip (SOC) in einem FPGA. Darüber hinaus wird eine Referenzanwendung entwickelt, mit der die Nutzbarkeit des Systems dargestellt wird. Das System wird mit folgenden Funktionen erstellt:

- Aufnehmen und Abspielen von Audiodateien in CD-Qualität[1]
- FAT-Filesystem-basierte Datenspeicherung auf SD-Karte
- Konfigurierbares, touchscreen-basiertes, grafisches Benutzerinterface
- PS/2 Tastatur-Steuerung
- Abspielen von MP3-Dateien

[1] CD-Qualität: 16 Bit, 44,1 kHz Stereo

1.2 Stand der Technik und vorhandene Lösungen

Im Rahmen dieser Arbeit wurde keine ausführliche Marktanalyse durchgeführt. Eine Internetrecherche lässt jedoch folgende Markttendenzen erkennen: Es gibt auf der einen Seite statische, mikrocontroller-basierte Systeme, die überwiegend auf der ARM-Architektur aufbauen. Diese werden vom Hersteller zusätzlich mit diversen Peripherie-Schnittstellen ausgestattet um viele Anwendungsbereiche abzudecken. Ein Beispiel ist der von NXP entwickelte LPC2880, der neben einem ARM Kern einen 16-Bit Audio-CODEC und eine MMC/SD-Karten-Schnittstelle integriert . Auf der anderen Seite gibt es konfigurierbare Systeme, die, wie das vorliegende, durch IP-Cores erweitert werden können. Ein Beispiel ist die vom Fraunhofer Institut entwickelte Audio-Plattform , die auf einem von ARC-International hergestellten ARCtangent™-A4 Softcore basiert .

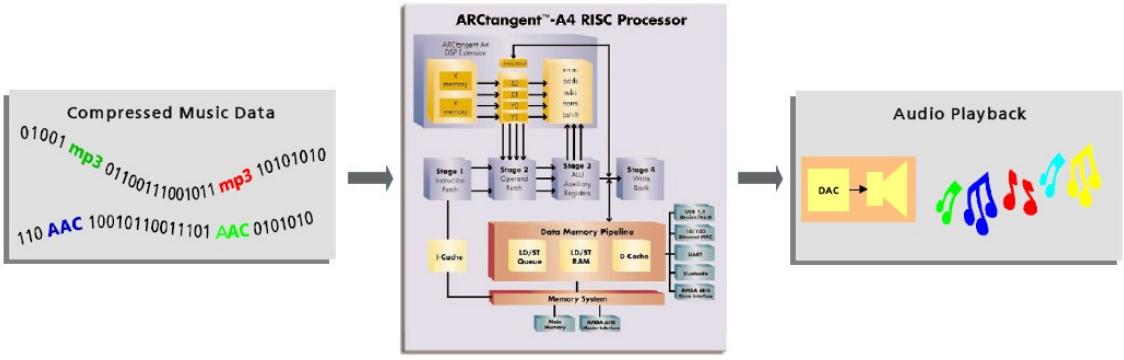


Abbildung 2: Fraunhofer Audio-Plattform, Quelle: Fraunhofer Institute, CorePool, 2002

1.3 Aufbau der Dokumentation

Die Gliederung der Arbeit folgt chronologisch den durchgeführten Entwicklungsschritten. Ausgehend von der Beschaffung des Entwicklungssystems bis hin zur Erstellung der Beispielapplikation. Literaturangaben sind in eckige Klammern gestellt und im Text fortlaufend nummeriert. Programmquelltext, Signal- und Dateinamen sowie Variablen sind typographisch durch den Schrifttyp `Courier-New` hervorgehoben. Eigennamen, wie *Nios II*, sind kursiv gestellt. Weiterführende Informationen, wie Programmquelltexte, verwendete Werkzeuge und Messergebnisse sind in den Anhang eingefügt.

2 Instrumentarium und Methodik

Im Folgenden werden die zur Erstellung dieser Arbeit verwendeten Werkzeuge und angewandten Methoden beschrieben. Die Darlegung umfasst die Wahl des Entwicklungssystems sowie die damit verbundene Software und die durchgeführten Entwicklungsabläufe für IP-Core-, System- und Softwareentwicklung. Verwendete Werkzeuge werden zusätzlich mit Angabe der Bezugsquelle in der Tabelle 32 im Anhang aufgeführt.

2.1 Entwicklungssystem

In der Vorbereitungsphase wurde das Entwicklungssystem *Nios II Embedded Evaluation Kit, Cyclone III Edition* der Firma Altera durch die Fachhochschule bereitgestellt.

Die Wahl auf dieses System fiel aufgrund der folgenden Eigenschaften:

- Komplettsystem mit on-board FPGA, RAM, Audio-CODEC, LCD-Touchscreen sowie PS/2- und SD-Karten-Verbindung
- Die mitgelieferte Entwicklungssoftware ist für nicht-kommerzielle Zwecke kostenlos
- Die Softwareentwicklung basiert auf der Eclipse IDE
- JTAG-Interface für Debugging und In-System-Programming (ISP)

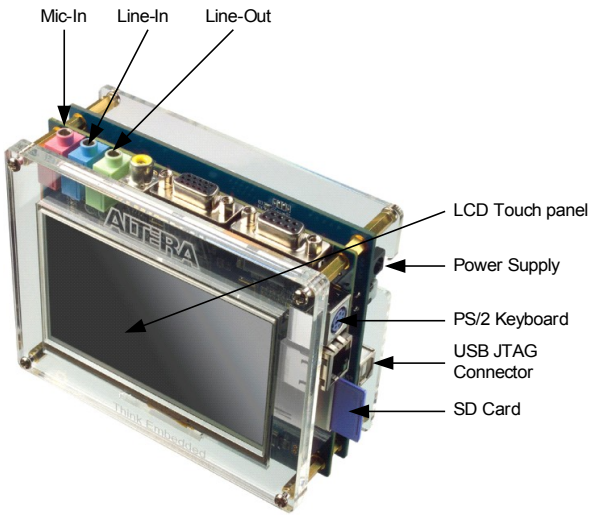


Abbildung 3: Entwicklungssystem mit den verwendeten Schnittstellen.
Quelle: Nios II Embedded Evaluation Kit Quick start guide

Eine detaillierte Beschreibung des Entwicklungssystems kann den Referenz-Handbüchern entnommen werden. Diese enthalten neben allgemeinen Erläuterungen auch Blockschaltbilder, Schaltungsausschnitte und Pin-Belegungen .

2.2 Schnelleinstieg mit „Hello World“ Programm

Das verwendete Entwicklungssystem wird mit allen zur Entwicklung nötigen Werkzeugen und Beispielen ausgeliefert. Im Folgenden wird der Einstieg erläutert:

1. Installation der Entwicklungssoftware (bestehend aus *Quartus II*, *Nios II IDE*, *MegaCore IP Library*, *ModelSim-Altera*) von der mitgelieferten DVD *Altera Complete Design Suite*
2. Installation der Evaluations-Lizenz:
    a. Altera bietet auf seiner Homepage unter folgender URL eine freie Lizenz für Evaluationszwecke an:

        https://www.altera.com/support/licensing/free_software/lic-q2web.jsp

    b. Die Lizenz wird per Email-Anhang zugestellt. Dazu muss unter obiger URL ein Entwickler-Konto angelegt werden. Hierbei sind einige personenbezogene Daten einzutragen. Zusätzlich muss die NIC Nummer des Host-PCs, auf dem entwickelt werden soll, angegeben werden

    c. Nachdem die Lizenz-Datei vorliegt, muss diese in das *Quartus II* Verzeichnis kopiert werden. Anschließend wird das Programm gestartet und das Menu „License Setup“ im Tools-Menu aufgerufen. Dort wird unter der Option „License File“ die Lizenz-Datei mit Pfad angegeben
3. Installation der Tutorials und Beispielprogramme von der mitgelieferten CD-ROM *Nios II Embedded Evaluation Kit Cyclone III Edition*. Auf die Installation des *„.NET Frameworks“* kann verzichtet werden
4. Anschluss des Kits:
    a. Die USB-Kabelverbindung zum Host und die Spannungsversorgung werden hergestellt

    b. Nach dem ersten Einschalten des Kits wird der Benutzer ggf. über eine neue Hardware informiert (*Byteblaster*). In der folgenden Installation muss der Pfad für den USB-Treiber angegeben werden. Dieser befindet sich im *Quartus II* Installationspfad im Unterordner `drivers`
5. Erstellen eines „Hello World“ Programms:
    a. Start der *NIOS II IDE*

    b. Über den Menupunkt „File→New→NIOS II C/C++ Application“ wird der „New Project“-Dialog geöffnet

    c. Innerhalb des Dialogs muss die Ziel-Hardware ausgewählt werden. Es empfiehlt sich, das installierte Standard-System auszuwählen:
    `cycloneIII_embedded_evaluation_kit_standard_sopc.ptf`

d. Ist dies nicht in der vorgegebenen Liste vorhanden, muss es im Dateisystem unterhalb des `altera` Ordners gesucht werden (siehe Abbildung 4)

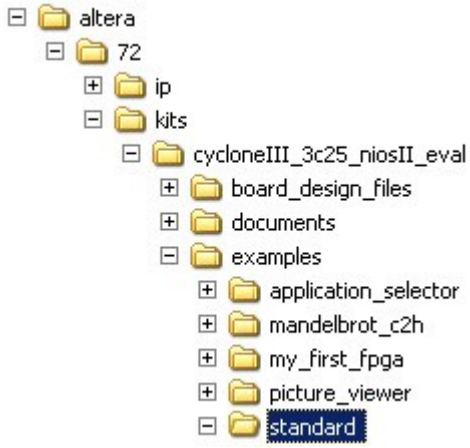


Abbildung 4: Ordnerstruktur der Beispielanwendungen

e. In der Beispiel-Liste wird der „Hello World“-Eintrag ausgewählt und der Projekterstellungsprozess wird mittels „Finish“-Button beendet

6. Starten des Beispiels

a. Über das Tools-Menu der *Nios II IDE* wird der *Quartus II Programmer* gestartet

b. Im Programm-Fenster wird über den „Add File“-Button die Datei `cycloneIII_embedded_evaluation_kit_standard.sof` ausgewählt. Diese befindet sich ebenfalls im oben angegebenen Ordner

c. Durch Betätigen des „Start“-Buttons wird die Datei in den FPGA übertragen

d. Wenn die Hardware nicht richtig erkannt wird, empfiehlt es sich, den USB-Stecker zu ziehen und erneut einzustecken. Danach wird über den „Hardware-Setup“-Button erneut das *Byteblaster* USB-Interface ausgewählt

e. Nachdem das System in den FPGA übertragen ist, kann das „Hello World“-Programm kompiliert und gestartet werden. Hierzu wird mit der rechten Maustaste in der *Nios II IDE* auf das erstellte Projekt im Project-Explorer geklickt und der Menupunkt „Run as→Nios II Hardware“ ausgeführt

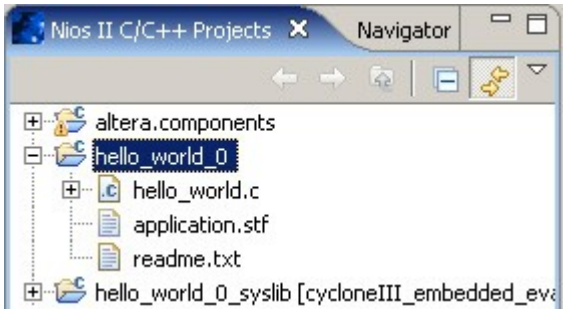


Abbildung 5: Projekt-Explorer mit „Hello World“-Projekt

2.3 Hardware Design

Hardware Design wird hier als der System-On-a-Chip (SOC) Erstellungsprozess verstanden. Dieser umfasst das Modellieren von Logik in einer Hardwarebeschreibungssprache (IP-Core Design), das Generieren des Systems aus mehreren Komponenten (System Design) und die anschließende Funktions-Verifikation.

2.3.1 Die Hardwarebeschreibungssprache Verilog

Die moderne Chipentwicklung ist eng an die Verhaltensmodellierung in einer Hardwarebeschreibungssprache (HDL) gekoppelt . Da in dieser Arbeit ein komplettes System mit eigenen Komponenten entwickelt wird, ist die Einarbeitung in eine HDL Voraussetzung. Es gibt eine Vielzahl von unterschiedlichen Sprachen wie VHDL, Verilog, AHDL, SystemC, usw. Welche Sprache für die Modellierung zum Einsatz kommt, hängt von syntaktischen Präferenzen, den entwicklungstechnischen Rahmenbedingungen und der Verfügbarkeit von Entwicklungswerkzeugen ab . Hier fiel die Entscheidung auf Verilog, da diese Sprache eine C-ähnliche Syntax besitzt und sehr verbreitet ist. Verilog-Dateien enthalten in der Regel die Implementierung eines vollständigen Moduls (Funktionsblock). Dieses kann in anderen Modulen durch Erstellung von Instanzen wieder verwendet werden (vgl. Objektorientierung). Hierdurch lassen sich komplexe Systeme im „Baukastenprinzip“ durch Verbindung mehrerer Module aufbauen. Eine Einführung in die Sprache bot das Buch *„Verilog – Modellbildung für Synthese und Verifikation“* von B. Hoppe .

2.3.2 Quartus II EDA Suite

Die verwendete Software *Quartus II* wird von Altera für den FPGA und CPLD-Chip-Entwurf ausgeliefert. Sie unterstützt die Standardsprachen VHDL und Verilog. Darüber hinaus können Module auch über eine grafische Oberfläche mit Schaltsymbolen zusammengefügt werden. Es ist eine Bibliothek enthalten, die Standard-Logikelemente bereitstellt. *Quartus II* wird für nicht-kommerzielle Zwecke als *Web Edition* kostenfrei zur Verfügung gestellt. Für kommerzielle Entwicklungen gibt es eine kostenpflichtige *Subscription Edition*, die erweiterte Funktionen bereitstellt . Das in der vorliegenden Arbeit vorgestellte System ist mithilfe der *Web Edition* entwickelt worden. Ein Nachteil der nicht-kommerziellen Version ist, dass einige von Altera zur Verfügung gestellte *MegaCores*[1] nur zeitlich begrenzt funktionieren. Diese Limitierung lässt sich verlängern, wenn das entwickelte System via JTAG Interface an den *Quartus II Programmer* angeschlossen bleibt (Tethered Mode) . Eine Einführung in die Entwicklungsumgebung bot das in elektronischer Form mitgelieferte Benutzerhandbuch .

2.3.2.1 Quartus II Projekt

Das *Quartus II* Projekt vereint alle für die Systemerstellung nötigen Dateien und Einstellungen. Dies gilt sowohl für die IP-Core- als auch für die Systementwicklung. Die Tabelle 34 im Anhang enthält eine Aufzählung der erstellten Projekte. Diese befinden sich auf der beiliegenden CD-ROM im `projects` Verzeichnis.

[1] MegaCores sind von Altera entwickelte und getestete IP-Cores

2.3.2.2 MegaWizard Plugin Manager

Der *MegaWizard Plugin Manager* ist ein Unterprogramm der *Quartus II EDA*. Dieses Werkzeug hilft über ein grafisches Benutzerinterface bei der Konfiguration von vorgefertigten, parametrisierbaren Funktionsblöcken (*MegaFunctions*). In dieser Arbeit wurde das Werkzeug zur Erstellung von PLLs, FIFOs und On-Chip-RAM verwendet.

2.3.3 IP-Core Design

Unter einem IP-Core (Intellectual Property Core), hier auch als Komponente oder Modul bezeichnet, wird ein Funktionsblock oder Bauplan mit definierter Schnittstelle und Funktion verstanden. Ein Core kann als Quelltext in einer HDL (hier Verilog) oder als bereits synthetisierte Netzliste vorliegen . Die Firma Altera liefert mit ihrer Entwicklungsumgebung eine *MegaCore IP Library* mit diversen vorgefertigten IP-Cores aus. Fehlende Komponenten müssen neu entwickelt oder über Drittanbieter bezogen werden. Eine weitere Möglichkeit besteht in der Integration von freien Modulen der Entwicklergemeinde *Opencores.org*, siehe Kapitel 2.3.4.2.

2.3.3.1 IP-Core Entwicklungsprozess

Im Folgenden wird erläutert, wie bei der Erstellung von IP-Cores vorgegangen wird. Der Gesamtprozess wird zusammenfassend in Abbildung 9 dargestellt.

Definition- und Spezifikationsphase:

1. Die Anforderung wird definiert, z.B. *„Das System soll mit einer PS/2 Tastatur verbunden werden“.*
2. Funktionsdefinition der Komponente. Hierzu sind ggf. genaue Kenntnisse der zugrunde liegenden Protokolle nötig, siehe Abbildung 6.

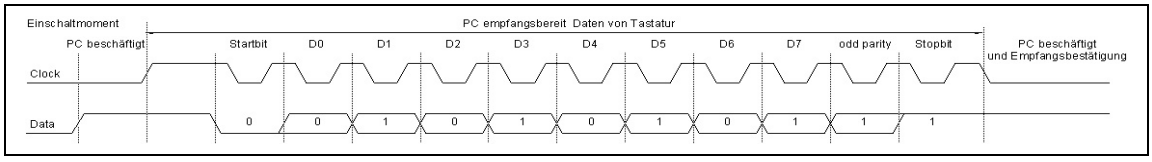


Abbildung 6: PS/2 Timing, Quelle: Wikipedia

3. Festlegung der externen Schnittstellen. Wahl der *Avalon*-Bus-Anbindung. Folgende kombinierbare Varianten sind möglich:
   - Der Core bekommt einen eigenen Adressraum zugewiesen (Memory Mapping), über den andere Bus-Master, wie die CPU, mittels Lese- und Schreibzugriffen auf die Komponente zugreifen können (Slave-Betrieb), siehe Abbildung 7
   - Die Komponente verarbeitet Streaming-Daten oder stellt sie zur Verfügung
   - Die Komponente ist ein Master und kann eigenständig auf den Bus zugreifen (Master-Betrieb)

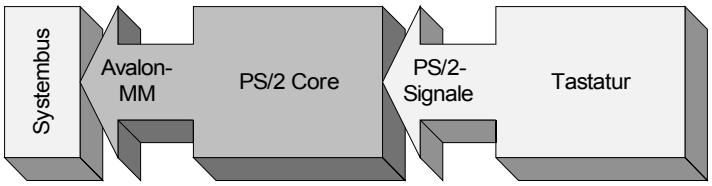


Abbildung 7: PS/2 Core mit Avalon-MM (Memory Mapping)

4. Soll ein Mikroprozessor die Komponente steuern, muss das Application Program Interface (API) und das Registerset definiert werden.
5. Untersuchung und Festlegung der Takt-Anforderungen. Welche Takte werden benötigt? Welche Signale sind synchron bzw. asynchron zu welchen Taktquellen?

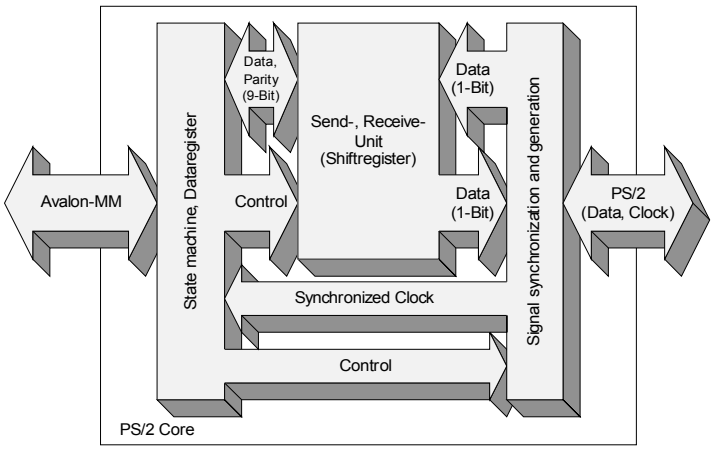


Abbildung 8: PS/2 Blockdiagramm

6. Definition der Funktionsblöcke (siehe Abbildung 8). Eine Teilung des Systems in logische Untergruppen, wie z.B. in Speicher- und Empfangseinheit, hilft bei der Komplexitätsreduktion. Die Ein- und Ausgangssignale der entstandenen Subsysteme werden spezifiziert.
7. Definition der Zustandsautomaten.

Implementierungs- und Verifikationsphase:

1. Für komplexe Logiken empfiehlt es sich, ein leicht simulierbares Verhaltensmodell anzufertigen. Dies kann z.B. mit Hilfe von bekannten Programmiersprachen wie C oder MATLAB geschehen.
2. Modellierung in einer HDL auf Register Transfer Ebene (RTL). Im Gegensatz zur Definition der Funktionsblöcke, in der die Top-Down-Methode angewendet wird, wird in der Implementierungsphase die Bottom-Up-Methode verwendet. Es wird also zuerst das kleinste Subsystem entwickelt und getestet. Sobald dies die gewünschte Funktion erfüllt, wird das nächst Größere, bzw. in der Hierarchie Übergeordnete, Modul implementiert.
3. Verifikation der Komponente im Simulator und in der Zielhardware, siehe Kapitel 2.3.6.

Systemkomponenten Erstellung:

1. Erstellung einer SOPC-Beschreibungsdatei, siehe Kapitel 2.3.4.2.
2. Erstellung einer Komponenten-Instanz im SOC

Entwicklung der Treiber:

1. Erstellung einer C-Header-Datei mit Register- und Statusdefinitionen
2. Erstellung und Test der Treiberfunktionen
3. Test des Registerzugriffs und der Steuerung mittels Prozessor (Simulation möglich)

2.3.3.2 Übersicht

Der oben beschriebene Entwicklungsprozess ist in folgender Abbildung dargestellt:

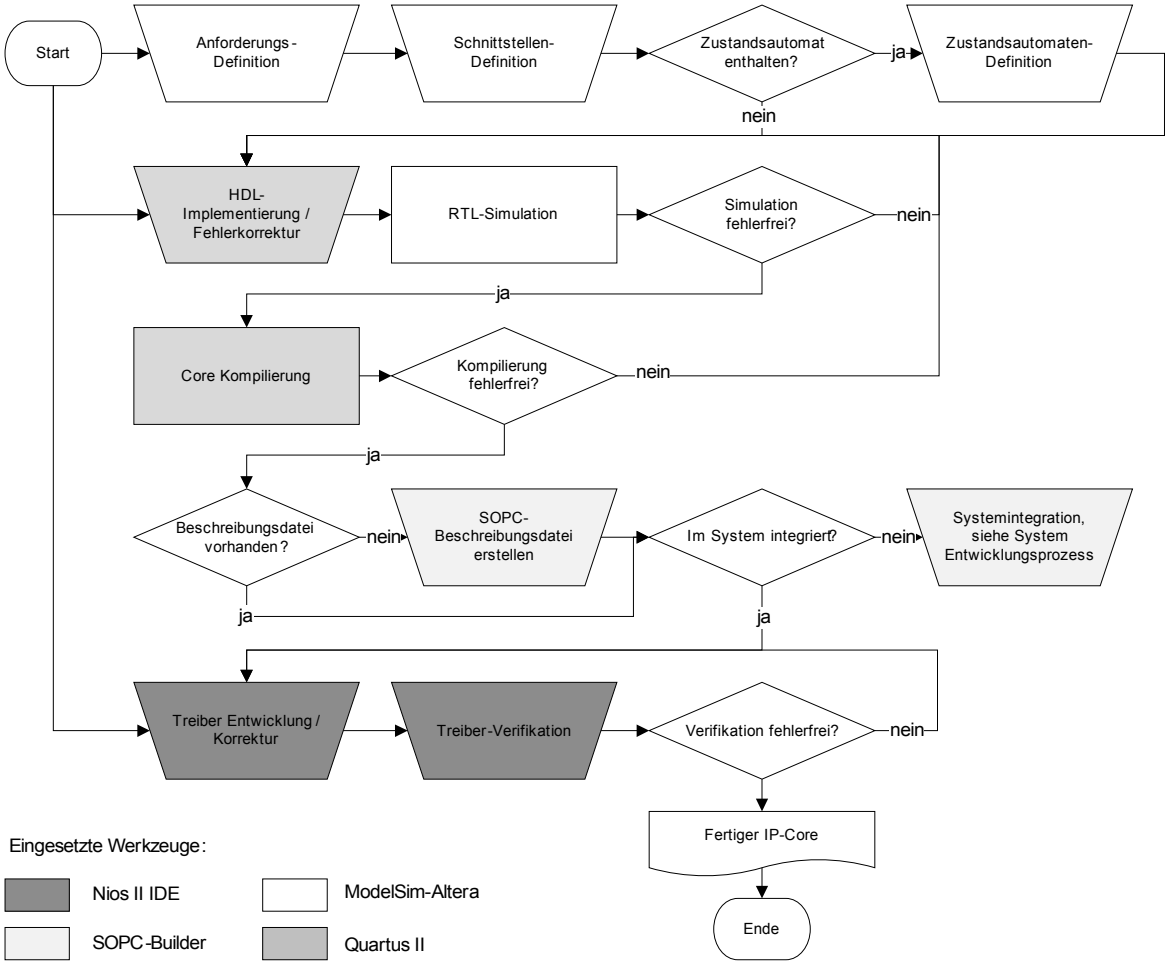


Abbildung 9: IP-Core Entwicklungsprozess Übersicht

2.3.4 System Design

Das System Design ist ein wichtiger Baustein im Hardwareentwicklungsprozess. Er umfasst das Generieren eines Systems aus mehreren Komponenten. Das Zusammensetzen des Systems erfolgt dabei auf abstrakter Ebene. Busverbindungslogiken zwischen den Komponenten werden automatisch generiert.

2.3.4.1 SOPC-Builder

Der *SOPC-Builder* ist ein Systembau-Werkzeug, dass mit der *Quartus II* Entwicklungsumgebung zur Verfügung gestellt wird. Das Werkzeug besitzt ein GUI, mit dem *Avalon*-Bus kompatible IP-Cores (siehe 2.3.4.2) zu einem Gesamtsystem kombiniert werden können (siehe Abbildung 10). Zusätzlich unterstützt das Programm den Entwickler in der überschneidungsfreien Interrupt- und Adressvergabe. Erkannte Überschneidungen werden automatisch angezeigt. In dieser Arbeit wurde auf die Arbitrierungslogik[1] Einfluss genommen um der verwendeten Audio-DMA mehr Speicherzugriffrechte einzuräumen als der CPU. Generiert wird ein synthetisierbares Gesamtsystem, das alle Teilkomponenten in sich vereint und mittels der *System Interconnect Fabric* verbindet .

Use	Conne...	Module Name	Description	Clock	Base	End	IRQ
☑		**cpu**	Nios II Processor				
		instruction_master	Avalon Memory Mapped Master	**clk**			
		data_master	Avalon Memory Mapped Master			IRQ 0	IRQ 31
		jtag_debug_module	Avalon Memory Mapped Slave		**0x00000800**	0x00000fff	
☑		**onchip_mem**	On-Chip Memory (RAM or ROM)				
		s1	Avalon Memory Mapped Slave	**clk**	**0x00002000**	0x00003fff	
☑		**jtag_uart**	JTAG UART				
		avalon_jtag_slave	Avalon Memory Mapped Slave	**clk**	**0x00000000**	0x00000007	

Abbildung 10: SOPC-Builder Systemdefinition mit Minimalprojekt, bestehend aus CPU, On-Chip RAM und JTAG UART

Ein so entstandenes System kann in ein bestehendes *Quartus II* Projekt integriert und externe Ports mit den EA-Pins des FPGA verbunden werden (siehe Abbildung 11). Mehr Informationen dazu können im Buch *„Rapid Prototyping of Digital Systems - SOPC Edition“* von O.J. Hamblen nachgelesen werden .

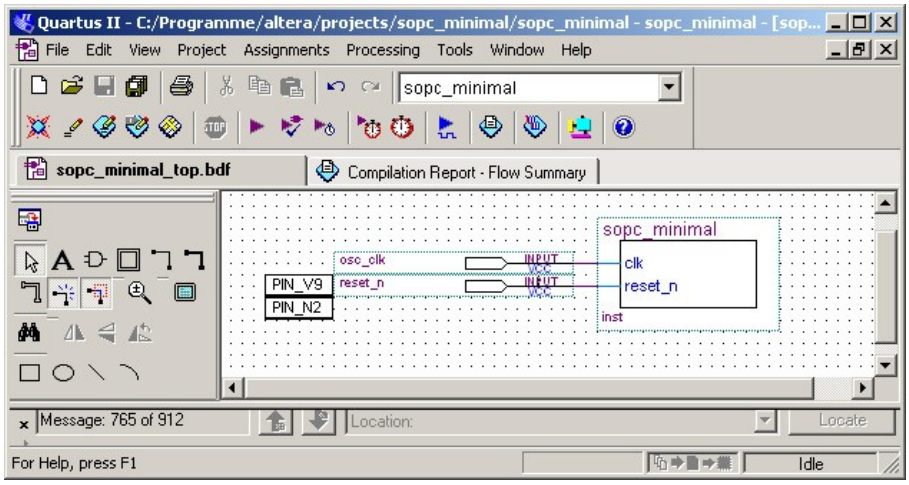


Abbildung 11: Generiertes System integriert in Quartus II Projekt und grafisches Top-Modul

[1] Unter Arbitrierung wird die möglichst gerechte Verteilung von Ressourcen (Buszeit) auf verschiedene Geräte verstanden.

2.3.4.2 Systemkomponenten

Für die *SOPC-Builder* basierte Systemgenerierung (siehe Kapitel 2.3.4.1) müssen die zu verwendenden IP-Cores eine SOPC-Beschreibungsdatei[1] besitzen. Diese kann mithilfe *Component Editors* (Unterprogramm des *SOPC-Builder*) erstellt werden (siehe Abbildung 12).

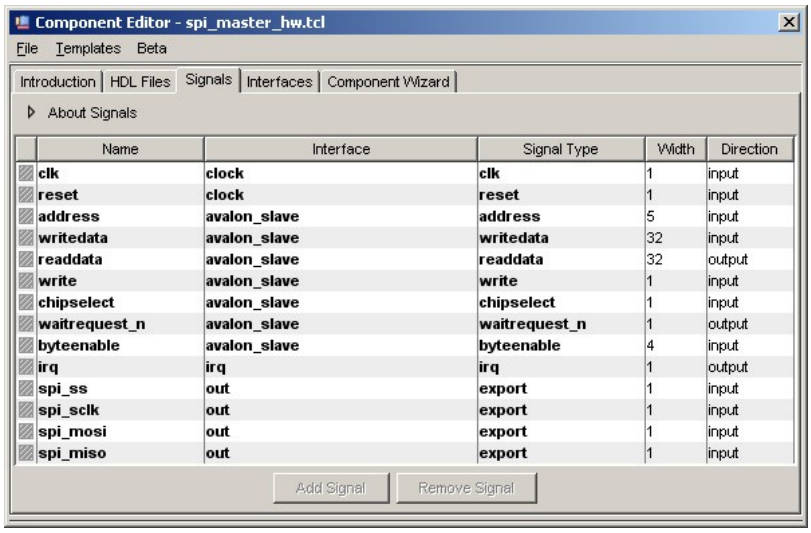


Abbildung 12: Altera's Component Editor mit Signalzuordnung

Bevor eine Beschreibungsdatei erstellt werden kann, muss sichergestellt sein, dass die zu integrierende Komponente über eine *Avalon*-Schnittstelle verfügt .

Bei den in der Arbeit verwendeten *Opencore*-IPs müssen die Wishbone-Bus-Signale[2] an den *Avalon*-Bus angepasst werden (siehe Abbildung 13). Die Signalzuordnung kann aus der Tabelle 33 im Anhang entnommen werden.

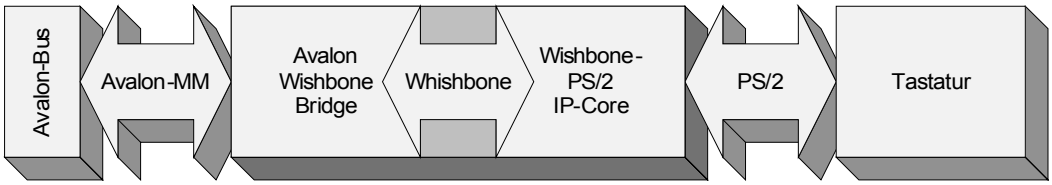


Abbildung 13: Wishbone- zu Avalon-Bus wrapping

2.3.4.3 System Entwicklungsprozess Übersicht

Als ein sinnvoller Entwicklungsablauf hat sich die iterative Integration neuer Komponenten herausgestellt (siehe Abbildung 14). Dabei wird auf der Basis eines funktionalen Systems eine Erweiterung vorgenommen. Jede Änderung des Systems erzwingt eine Generierungs-, Kompilier- und Testphase, um sicherzustellen, dass die Veränderung keine Fehler verursacht hat. Ist ein Fehler entstanden, wird das System so lange korrigiert und getestet, bis wieder ein funktionales Ausgangssystem zur Verfügung steht. Erst dann wird das System erneut erweitert. Dieses Verfahren ist am Anfang aufwändig, hilft aber den späteren Einsatz für die Fehlersuche zu reduzie-

[1] Die SOPC-Beschreibungsdatei hat eine _hw.tcl Dateiendung

[2] Der Wishbone-Bus ist ein Open Source Bussystem

ren, da eine schrittweise Veränderung nur einen begrenzten Wirkungskreis hat und somit die entstandenen Fehler leichter zugeordnet und behoben werden können.

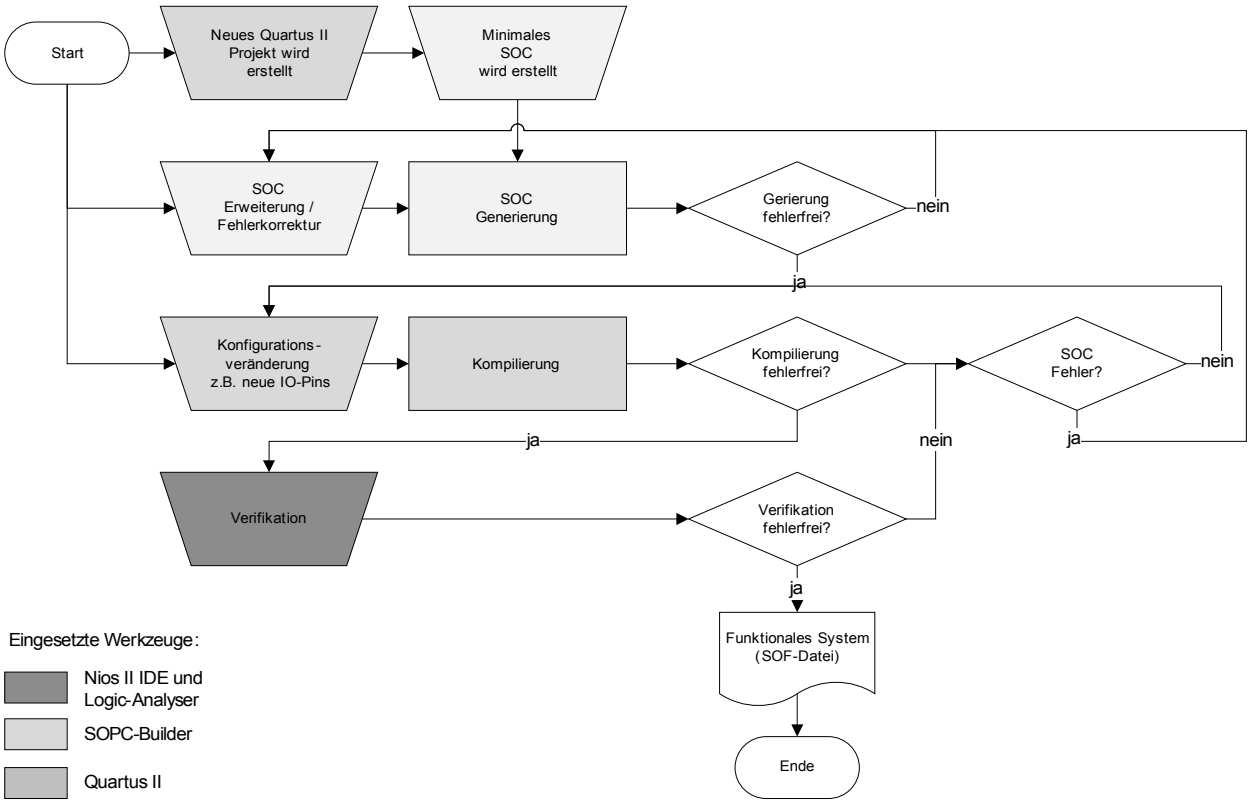


Abbildung 14: Iterativer System-Entwicklungsprozess

2.3.5 In-System-Programming

Das In-System-Programming (ISP) über das JTAG-Interface wird sowohl für die Konfiguration des FPGA verwendet als auch für das transferieren von Programmdaten an den *Nios II* Prozessor. Bei der Programmierung des FPGA kommt der *Quartus II Programmer* zum Einsatz. Dieser kann über *Quartus II* oder die *Nios II IDE* gestartet werden. Für den Programmdownload gibt es das Kommandozeilen-Werkzeug `nios2-download` oder die einfachere Möglichkeit, direkt in der *Nios II IDE* eine „Run Configuration“ anzulegen, die das Downloaden automatisiert.

2.3.6 Verifikation

Unter dem Begriff Verifikation wird hier die Überprüfung der Komponenten- bzw. Systemfunktionen verstanden. Diese gliedert sich in eine Verifikation durch Simulation in der Modellierungsphase und in eine Verifikation der Signale in der Zielhardware im Anschluss an die Synthese. Die in dieser Arbeit entwickelten IP-Cores werden in der Modellierungsphase mittels Testbench und *ModelSim-Altera* simuliert. Die Überprüfung der Signale im fertigen System erfolgt mit einem Oszilloskop bzw. Logic-Analyser. Eine Erläuterung der Methoden folgt:

2.3.6.1 Testbench-Simulation

Eine Testbench ist eine HDL-Datei, in der eine Instanz der zu untersuchenden Komponente erstellt wird. Zusätzlich werden in der Datei Funktionen definiert, die die nötigen Prüfsignale für die Komponente generieren. Während der Simulation können Ausgangsgrößen mit zuvor definierten Werten verglichen werden. Tritt bei dem Vergleich eine Unstimmigkeit auf, so kann die Simulation mit einem Fehler abgebrochen werden. Die in dieser Arbeit entworfenen Testbenches werden mit dem von Altera zur Verfügung gestellten Werkzeug *ModelSim-Altera* überprüft. Dieses kann nach Konfiguration über *NativeLink* direkt aus *Quartus II* gestartet werden . Das wichtigste Hilfsmittel dabei ist die Wellenform-Ansicht der untersuchten Signale (siehe Abbildung 15). Fehlfunktionen können anhand des dargestellten Signalverlaufs erkannt werden. Eine weitere Möglichkeit ist die textbasierte Ausgabe von Signal- und Register-Werten. Dies ist mit Debug-Ausgaben innerhalb eines C-Programms vergleichbar.

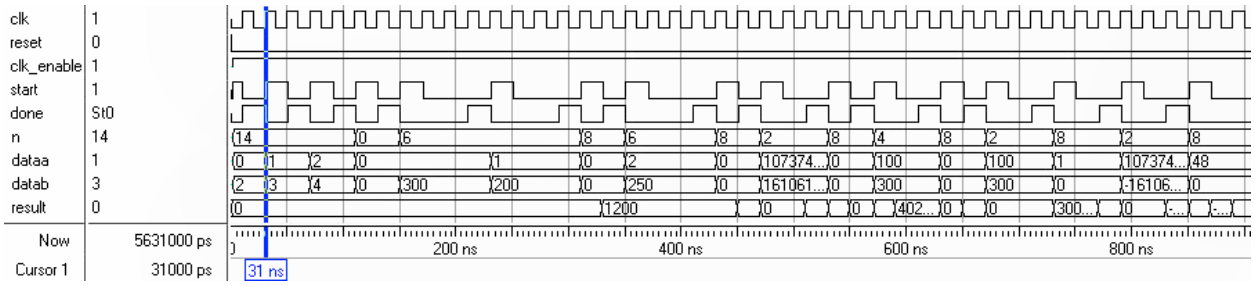


Abbildung 15: Wellenformdarstellung einer Testbench-Simulation

2.3.6.2 Verifikation in der Zielhardware mittels Logic-Analyser

Um Fehler im Zielsystem zu untersuchen, werden zu überprüfende interne Signale durch eine Design-Änderung an die EA-Pins des FPGAs geleitet und mittels Logic-Analyser oder Oszilloskop überprüft. Dies ist zeitintensiv, da die Änderung eine Neu-Kompilierung des Systems erfordert. Eine Alternative dazu ist das Einsetzen des *SignalProbe* Debugging-Cores, ein von Altera zur Verfügung gestellter *MegaCore*, der das Überprüfen interner Signale vereinfacht. Altera bietet für seine Entwicklungsumgebung weitere Debugging-Werkzeuge an, die in dieser Arbeit nicht verwendet werden .

2.4 Softwareentwicklung

Die in dieser Arbeit durchgeführten Software-Entwicklungsschritte konnten mit den von Altera zur Verfügung gestellten Werkzeugen durchgeführt werden. Diese werden im Folgenden beschrieben.

2.4.1 Programmiersprache

Die Softwareentwicklung wurde ausschließlich in der Programmiersprache *C* durchgeführt. Die verwendete Standard-Bibliothek baut auf der *newlib*[1] auf. In den erstellten Quelltexten wird

[1] Newlib Homepage: http://sourceware.org/newlib

Funktionsnamen ein Präfix vorangestellt, um diese modulbezogen zu gruppieren und Linker-Probleme bei gleichen Benennungen im Voraus zu vermeiden. Funktionsparameternamen wird ein Unterstrich (_) vorangestellt, um diese von globalen und funktionsinternen Variablen zu unterscheiden.

2.4.2 Nios II- Entwicklungsumgebung

Altera stellt mit seiner EDA-Suite eine *Nios II* Entwicklungsumgebung zur Verfügung. Diese ist eine speziell für die *Nios II* Entwicklung angepasste Eclipse-Version. Darin sind Beispiele und Tutorials für den Einstieg enthalten. Neue Projekte können über Wizards erstellt werden. Die integrierte Entwicklungsumgebung (IDE) kümmert sich um den Build-Prozess. Es müssen keine Make-Dateien manuell erstellt oder angepasst werden. Der *Quartus II Programmer* kann über das Tools-Menu gestartet werden. Wichtig sind die integrierten Debugging- und Profiling-Funktionen, die einen durchgängigen Entwicklungsablauf erlauben, siehe folgende Abschnitte.

2.4.3 Treiber Entwicklung

Altera stellt eine *Hardware Abstraction Layer* (HAL) genannte Treiberschicht zur Verfügung. Die HAL ist mit der Standard-Bibliothek *newlib* gekoppelt. Sie ermöglicht den Zugriff auf Dateien und Devices mit den Standard EA-Funktionen . Instanzvariablen werden an die Treiberfunktionen über Zeiger auf eine File-Struktur übergeben, die wiederum den Zugriff auf die Device-Struktur erlaubt. In dieser können z.B. Port-Adressen oder Interrupt-Nummern definiert sein. Hierdurch können die Treiberfunktionen für unterschiedliche Device-Instanzen wieder verwendet werden. Einige in dieser Arbeit entwickelte Treiber nutzen diese Schicht zur Funktions-Kapselung (siehe Abbildung 48).

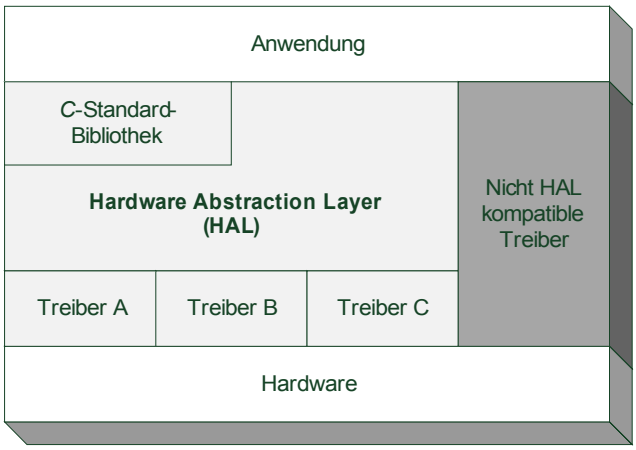


Abbildung 16: Ebenen eines HAL-basierten Systems

2.4.4 System Library Project

Um die Software-Entwicklung mit einem *SOPC-Builder*-generierten System zu ermöglichen wird ein *System Library Project* in der *Nios II IDE* erstellt (siehe Abbildung 17). Dieses Bibliotheks-Projekt erzeugt eine `system.h` Datei, in der die EA-Adressen, IRQ-Nummern usw. für das Sys-

tem definiert sind. Die Altera-spezifischen *Newlib* Funktions- und Treiberimplementierungen sind in form einer Softwarebibliothek enthalten. Der HAL-Treiber-Initialisierungs-Code befindet sich in der `alt_sys_init.c` Datei. Dieser wird vom Startup-Code vor der `main` Funktion ausgeführt.

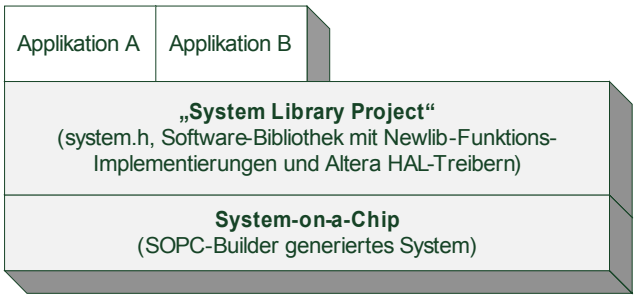


Abbildung 17: System-Library-Project als Verbindung zwischen SOC und Applikation

2.4.5 Performance-Messung mit GNU-Profiler und Performance Counter

Um die Ausführungsgeschwindigkeit einer Anwendung zu verbessern, müssen langsame Programmstellen identifiziert werden. Im Anschluss werden die gefundenen Programmabschnitte genauer analysiert und, soweit möglich, Optimierungen durchgeführt. Für das Auffinden von langsamen Funktionen, ohne direkten Anhaltspunkt, bietet sich die Untersuchung mittels *GNU-Profiler* an. Dieses Werkzeug kann in der *Nios II IDE* in der System-Library Konfiguration aktiviert werden („Link with profiling library“). Nach Beendigung des untersuchten Programms steht die Datei `gmon.out` mit den gemessenen Funktionszeiten zur Verfügung und kann innerhalb der Entwicklungsumgebung geöffnet werden. Anhand der enthaltenen Informationen können die genauer zu untersuchenden Abschnitte identifiziert werden.

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 28.62      1.35     1.35   540189     0.00     0.00  push_on_queue
 20.12      2.30     0.95        1     0.95     4.14  main
 15.46      3.03     0.73        6     0.12     0.12  alt_file_locked
 11.78      3.59     0.56   540189     0.00     0.00  ad7843_get_next_msg
  4.61      3.80     0.22   540189     0.00     0.00  get_pen_state
  3.95      3.99     0.19   540189     0.00     0.00  oc_ps2_has_next
```

Abbildung 18: Ausschnitt aus der GNU-Profiler-Ausgabe

Stehen die zu untersuchenden Programmabschnitte fest, kann mittels des *Performance Counter* Cores und dessen API eine taktzyklengenaue Messung angeschlossen werden. Dies geschieht, indem an entsprechenden Stellen im Programm Start- und End-Aufzeichnungsmakros eingefügt werden. Die Verwendung des *Performance Counters* setzt eine Integration im SOC voraus .

2.4.6 Fehlersuche - Debugging

Um Programmfehler zu finden, werden unterschiedliche, aufeinander aufbauende Verfahren eingesetzt. Im ersten Schritt werden Programmteile mit Log-Befehlen ausgestattet, die Ergebnisse in Textform in einer Konsole ausgegeben. Anhand dieser Ausgaben kann erkannt werden, ob bestimmte funktionsspezifische Rahmenbedingen eingehalten werden. Parameterprüfbedingungen (Asserts) werden zu wichtigen Funktionen hinzugefügt. Ist ein fehlerhafter Programmabschnitt identifiziert, wird der Einstiegspunkt mit einem Breakpoint markiert und mittels Debugger im Single-Step-Verfahren durchlaufen. Dabei werden die Variablen und Speicherinhalte auf Korrektheit überprüft. Bei der Untersuchung von Interrupt Service Routinen (ISR) kommt es vor, dass ein Einkreisen des Fehlers mit Logs und Debugger nicht möglich ist. Hier müssen Prüfbedingungen erstellt werden, die im Fehlerfall z.B. eine LED ansteuern. Der verwendete Debugger ist in die *Nios II IDE* integriert.

2.4.7 Software Entwicklungsprozess

Nachdem das FPGA mit dem SOC konfiguriert ist, verläuft der Software Entwicklungsprozess wie folgt (siehe Abbildung 19): Bei einer neuen Anwendung wird zuerst ein Projekt erzeugt. Wird ein Programm erweitert, muss dies anschließend in das FPGA übertragen und getestet werden. Treten Programmfehler auf, wird der Fehler eingekreist und das Programm ggf. mit dem Debugger durchlaufen. Die gefundenen Programmfehler werden korrigiert. Anschließend wird das Programm erneut übertragen und gestestet. Dieser Vorgang wird bis zur vollständigen Programmimplementierung wiederholt. Am Ende des Prozesses steht eine getestete Programmdatei.

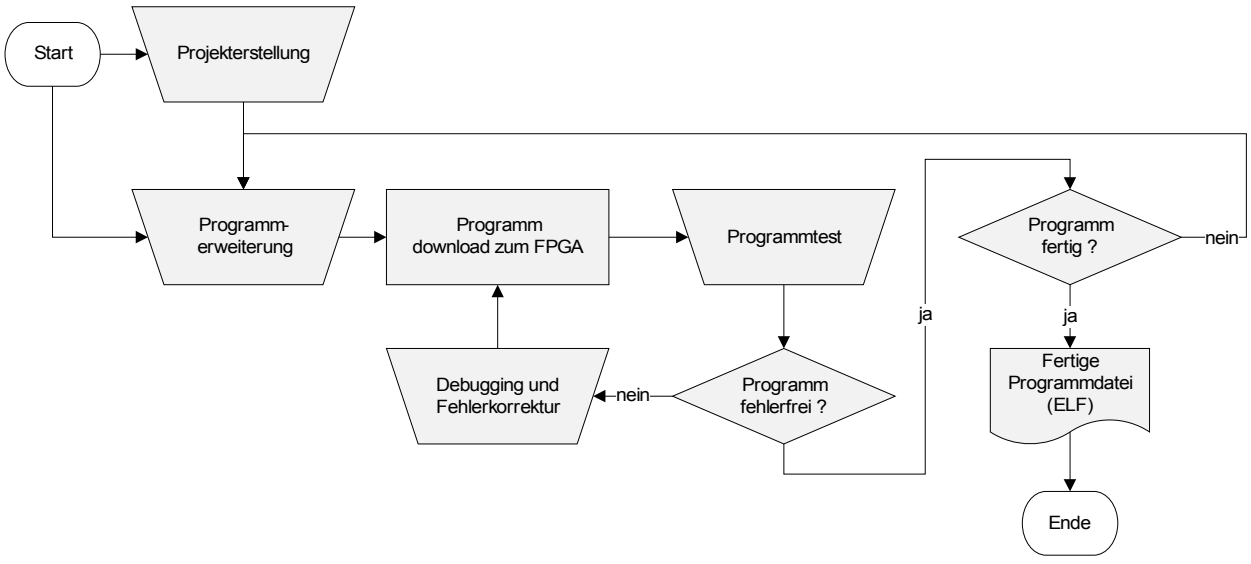


Abbildung 19: Software Entwicklungsprozess

3 Konzept

Das in der Einleitung definierte Ziel wird erreicht, indem zum einen ein System-on-a-Chip (SOC) und zum anderen eine Softwarebibliothek mit Treibern zur Steuerung der Hardware entworfen werden.

Die folgende Abbildung zeigt den geplanten Aufbau der Audio-Plattform zusammen mit der erstellten Referenzanwendung:

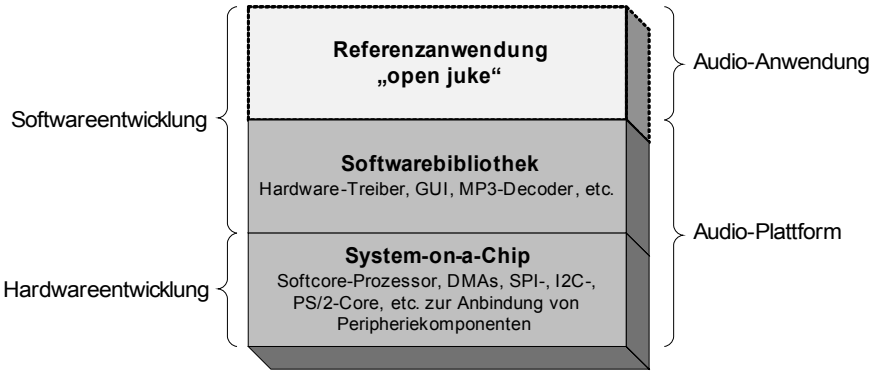


Abbildung 20: Teilung des Systemaufbaus in Hard- und Software

Konzeptionell wird das weitere Vorgehen in eine Hardwareentwicklungs- und Softwareentwicklungsphase geteilt:

Die Hardwareentwicklung gliedert sich in die **IP-Core-Entwicklung**, in der die Systemkomponenten, die zur Anbindung der vorhandenen Peripheriebausteine benötigt werden, erstellt bzw. an das verwendete Bussystem angepasst werden, sowie die **Systementwicklung** in der die verwendeten Komponenten zu einem Gesamtsystem zusammengeführt werden.

In der Softwareentwicklung wird die **Audio-Plattform-Softwarebibliothek** mit den Treibern für die verwendeten Systemkomponenten entwickelt. Zusätzlich werden hardwareunabhängige Module integriert, die bei der Anwendungsentwicklung helfen sollen. Zur Darstellung der Flexibilität der konfigurierbaren Hardware wird ein **MP3-Decoder** integriert, der durch die Verwendung eines speziell in der IP-Core-Entwicklung erstellten Co-Prozessors (64-Bit-ALU) beschleunigt wird.

Den Abschluss bildet die Entwicklung der **„open juke“-Referenzanwendung**, die auf der erstellten Hard- und Softwarebasis aufbaut.

4 IP-Core Entwicklung

Das geplante System-on-a-Chip (SOC) wird aus verschiedenen Systemkomponenten zusammengesetzt (siehe Abbildung 21). Diese sind für die Anbindung der Peripherie, wie Audio-CODEC, SD-Karte, usw. notwendig. Die meisten der benötigen Komponenten, wie der *Nios II* Prozessor, werden in der *Mega-Core-IP-Library* von Altera zur Verfügung gestellt. Darin nicht enthaltene Funktionsblöcke werden aus dem Pool der Entwicklergemeinde *Opencores.org* verwendet (siehe Kapitel 4.4). Für spezielle Funktionen müssen neue Cores entwickelt werden, wenn diese nicht in den verfügbaren Pools enthalten sind bzw. den gestellten Anforderungen nicht entsprechen. Ein Beispiel dafür ist der Core für die Anbindung des WM8731-CODECs (siehe Kapitel 4.2).

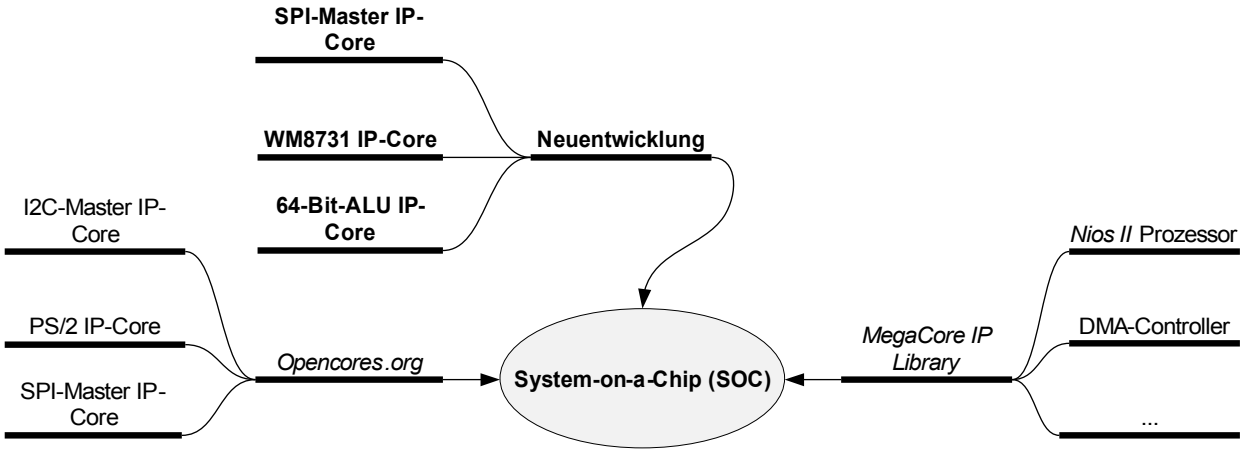


Abbildung 21: Komponenten aus unterschiedlichen Quellen werden im System kombiniert

In den folgenden Kapiteln wird die Entwicklung bzw. die Anpassung der Komponenten beschrieben. Der Entwicklungsablauf orientiert sich an dem im Instrumentarium- und Methodikteil der Arbeit vorgestellten Entwicklungsprozess (siehe Kapitel 2.3.3.2). Die erstellten Verilog Quelltextdateien sind in den H eingefügt.

4.1 SPI-Master IP-Core

SD-Karten unterstützen zwei Busverbindungsmodi: den 4-Bit-Parallel- und den 1-Bit-SPI-Modus. In dieser Arbeit wird aufgrund der nicht verfügbaren Dokumentation des 4-Bit-Modus und der bereits vorhanden Hardware die Anbindung per SPI-Bus vorgenommen. Tests im Vorfeld ergaben, dass die maximal auf dem verwendeten Entwicklungssystem erreichbare SPI-Busfrequenz ca. 16 MHz beträgt. Dies begründet sich darin, dass eine Signalpegeltransformation von 2.5V (FPGA) auf 3.3V (SD-Karte) durchgeführt werden muss und der eingesetzte MAX3378-IC lt. Datenblatt mit maximal 16 Mbit/s arbeitet . Eine weitere Einschränkung der Übertragungsgeschwindigkeit begründet sich in der Core-zu-Speicheranbindung. Die zwei verfügbaren Komponenten, der *Opencores* SPI-Core und der Altera SPI-*MegaCore*, besitzen ein byteorientiertes

Avalon-Slave-Interface und sind für den Datenaustausch mit dem Speicher auf den Prozessor angewiesen, der die Daten Byte für Byte zwischen Speicher und Core überträgt. Diese Art der Anbindung führt zu einem Geschwindigkeitsverlust, der im oberen Signalverlauf der folgenden Abbildung dargestellt ist.

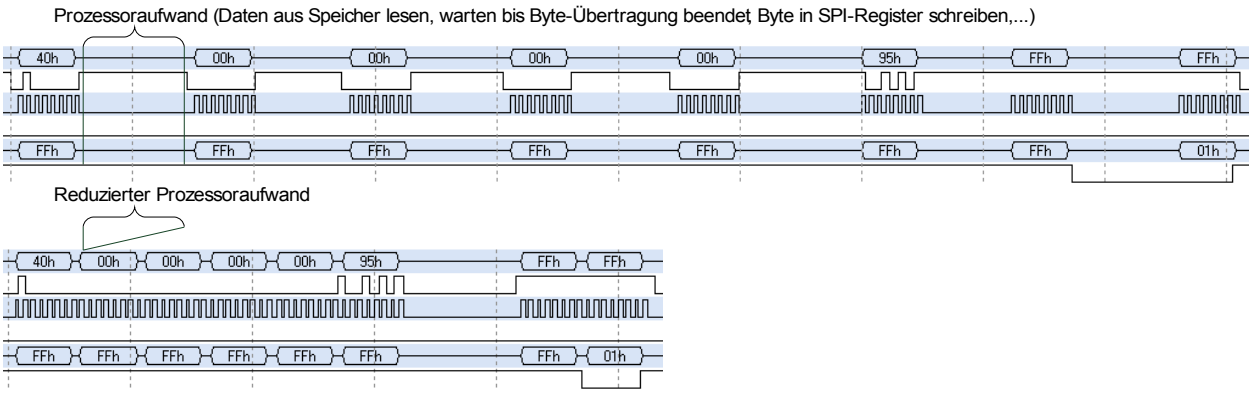


Abbildung 22: Prozessoraufwand bei der SPI-Datenübertragung und Optimierung im Vergleich

4.1.1 Ziel

Das Speicherinterface der oben genannten Cores ist für die schnelle Übertragung von großen Datenmengen (z.B. Audiostreaming in CD-Qualität) ungeeignet. Um dieses Problem zu lösen, wird ein SPI-Core entwickelt, der die volle Busbreite (32-Bit) bei der Datenübergabe von Speicher-zu-Core ausnutzt und eine parallele Datenübertragung ermöglicht. So kann, während der Prozessor die Daten an den Core übergibt, bereits die Übertragung auf dem SPI-Bus gestartet werden. Durch diese Optimierung, die anhand des unteren Signalverlaufs in der Abbildung 22 zu erkennen ist, soll eine Performanceverbesserung um bis zu 100 Prozent erreicht und somit das Audiodatenstreaming ermöglicht werden.

4.1.2 Konzept

Das definierte Ziel wird erreicht indem jeweils zwischen Übertragungseinheit (SPI-Bus-Seite) und Speicherinterface ein Zwischenspeicher (FIFO) eingesetzt wird. Hierdurch wird ermöglicht, dass parallel zur Übertragung bereits neue Daten zur Verfügung gestellt werden können. Für die Steuerung der Datenübertragung wird ein Transfersize-Register benötigt, in dem die Anzahl der zu übertragenden Bytes definiert wird. Innerhalb des Transfer-Registers werden zusätzliche Bits für die Operationswahl vorgesehen. Zusätzlich wird ein Statusregister benötigt, über das der Prozessor mit Zustandsinformationen versorgt wird. Über ein Flag im Kontroll-Register können die FIFOs gelöscht werden. Die folgende Tabelle zeigt die definierten Register und ihre Funktion:

Register	**Beschreibung**
`REG_FIFO`	Daten-Schreib-Leseregister (32-Bit)
`REG_FIFO_FILL_COUNTS`	Füllzustände der FIFOs jeweils 16-Bit
`REG_TRANSFER_SIZE`	Übertragungsgröße in Bytes, Start-Flag, Job-Bits

REG_CONTROL	Zusatzfunktionen wie FIFOs löschen etc.
REG_STATUS	Statusbits (BUSY, FIFO-Füllzustände)
REG_SCLK_DIVIDER	Slave Clock Taktteiler-Register (8-Bit)
REG_SS	Slave Select Register (8-Bit)
REG_TOKEN	Token (8-Bit)

Tabelle 1: SPI-Master EA-Register

Für die interne Steuerung wird ein Zustandsautomat (Finite State Machine ─ FSM) implementiert (siehe Abschnitt 4.1.3.1), der folgende zusätzliche Funktionen ermöglicht: Das Warten auf den Empfang eines definierbaren 1-Byte-Tokens[1] sowie das Warten auf ein Token, das sich von dem zuvor definierten unterscheidet. Diese Funktionen werden im Zusammenhang mit dem SD-Karten-Protokoll benötigt und stellen eine weitere Performance-Verbesserung in Aussicht. Folgende Operationen werden für den Core definiert:

Name	**Beschreibung**
TRANSFER	Datenübertragung Master→Slave
RECEIVE	Datenübertragung Slave→Master
WAIT_FOR_TOKEN	Warten auf spezielles Token
WAIT_WHILE_TOKEN	Warten bis ein anderes als das definierte Token empfangen wird

Tabelle 2: SPI-Master Operationen (Jobs)

Die Abbildung 23 zeigt den schematischen Aufbau des entwickelten Cores. Links ist das Speicherinterface dargestellt, über das der Prozessor angebunden ist. Rechts ist die Schnittstelle zum SPI-Bus eingezeichnet. Die in der Mitte angeordnete Zustandsmaschine wird über die EA-Register vom Prozessor kontrolliert und steuert die angebundenen Schieberegister sowie die Empfangs- bzw. Sendezwischenspeicher.

[1] Ein Token wird hier als 1-Byte Symbol verstanden, das in der seriellen Kommunikation z.B. als Start- oder Stopp-Bedingung eingesetzt wird.

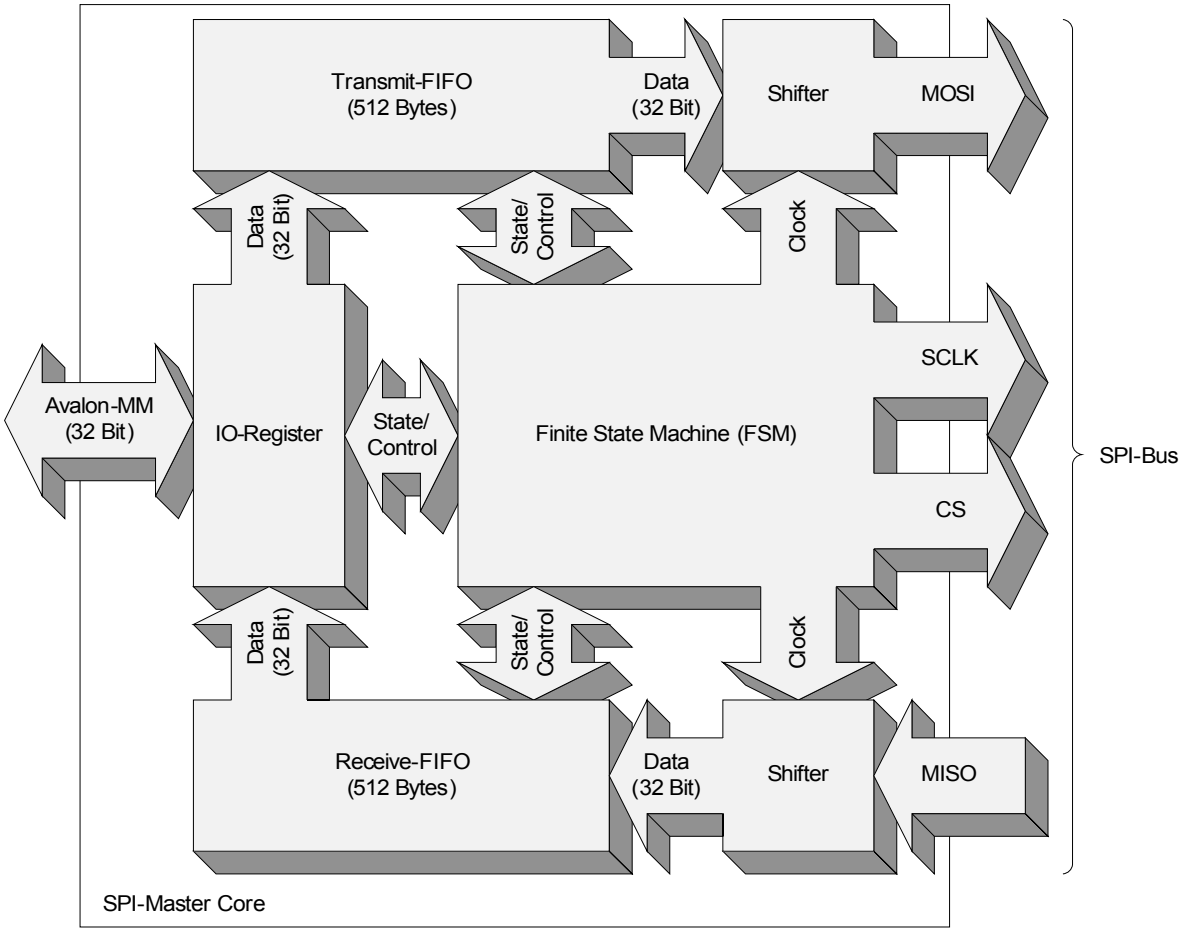


Abbildung 23: SPI-Master Blockdiagramm

4.1.3 Implementierung

Die Implementierung beginnt mit dem Entwurf des Zustandsautomaten, der anschließend in Verilog modelliert wird. Eine Simulation der Komponente wird durchgeführt. Die Treiber-Entwicklung und System-Integration folgen.

4.1.3.1 Zustandsautomat

Die Übertragungslogik wird anhand des in Abbildung 24 dargestellten und im Folgenden beschriebenen Zustandsautomaten implementiert. Der Automat startet nach einem Reset-Signal in den *Idle*-Zustand. Dieser wird nach dem Empfangen des Start Flags (`start_transfer`) entweder in Empfangs- (*Receive-Bits*) oder Übertragungsrichtung (*Load-Data*) verlassen.

Im *Load-Data*-Zustand wird aus dem Transfer-FIFO ein Datenwort (32-Bit) ausgelesen und in den *Transfer-Bits*-Zustand gewechselt. Dort wird der SPI-Takt über Teiler generiert und die Datenbits seriell an den Ausgabeport übertragen. Sind alle Bytes übertragen (`transfer_count==0`), wird in den *Finish*-Zustand gewechselt. Dort werden die Ausgangssignale zurückgesetzt und wieder in den *Idle*-Zustand gewechselt.

Im *Receive-Bits*-Zustand werden die seriell empfangenen Daten in ein Datenregister geschrieben. Ist ein Byte empfangen worden, wird je nach gewählter Operation (`job`) unterschiedlich

verzweigt. Ist `RECEIVE` ausgewählt und das Datenregister ist voll (4 Bytes empfangen), wird in den *Store-Data*-Zustand gewechselt, in dem die Daten in das Empfangs-FIFO gespeichert werden. Sind weitere Bytes zu empfangen, wird je nach Operation in den *Receive-Bits*-Zustand zurückgekehrt. Ansonsten wird aus dem *Store-Data*-Zustand das Empfangen durch den Übergang in den *Pre-Finished2*-Zustand beendet. Ist die `WAIT_FOR_TOKEN` Operation ausgewählt, wird nur dann in den Store-*Data* Zustand verzweigt, wenn das empfangene Byte dem zuvor definierten Token entspricht. Bei der `WAIT_WHILE_TOKEN` Operation ist dies genau umgekehrt. Hier wird dann in den *Store-Data*-Zustand verzweigt, wenn das empfangene Byte ungleich ist. Der *Receive-Bits*-Zustand kann auch direkt in den *Pre-Finish2*-Zustand übergehen, wenn keine Bytes mehr zu empfangen sind und die Token Operationen nicht erfolgreich verlaufen sind.

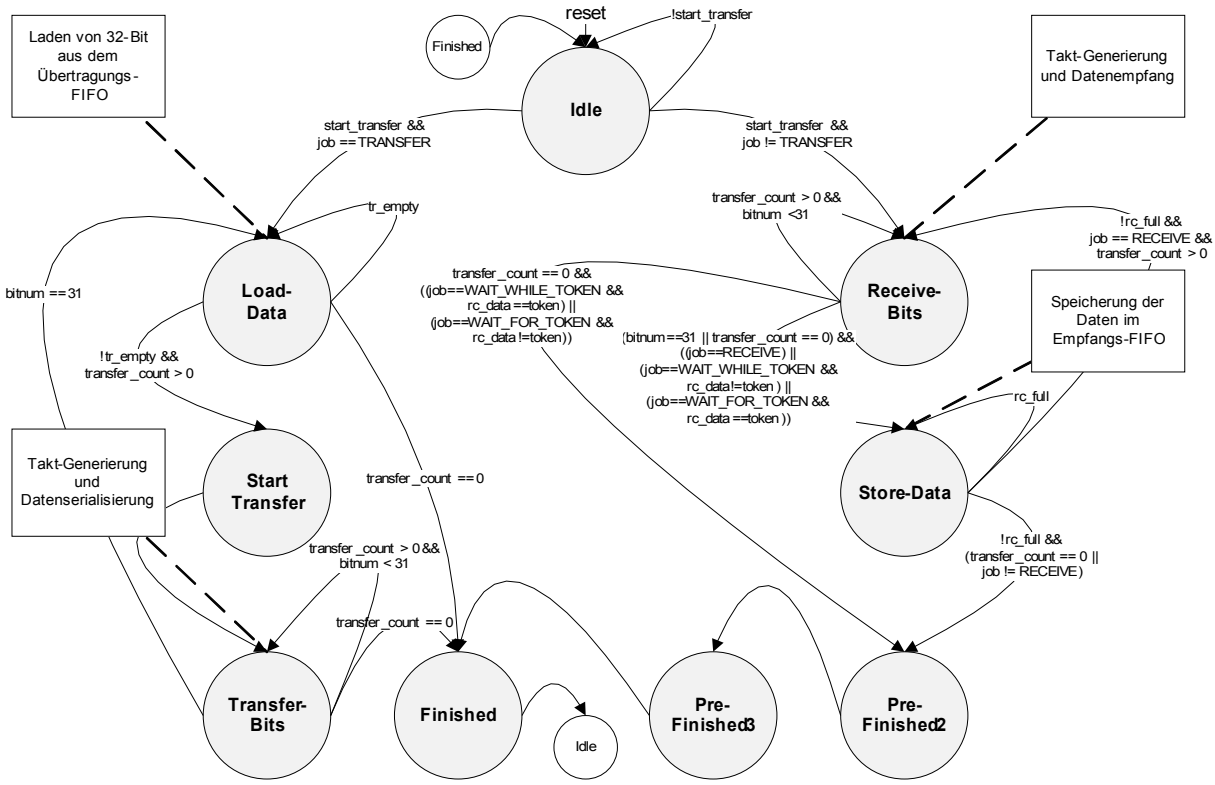


Abbildung 24: SPI-Master Transceiver- Zustandsautomat

4.1.3.2 Projekt- und Modul-Struktur

In *Quartus II* wird das `spi_master` Projekt und ein `hdl` Unterverzeichnis für die Verilog-Dateien zur besseren Strukturierung angelegt. Die Empfangs- und Sende-FIFOs werden mit dem *MegaWizard Plugin Manager*, basierend auf dem Single-Clock-FIFO[1], generiert . Die Definitionsdatei, in der die Konstanten (Register, Status, usw.) definiert werden, wird angelegt. Das Transceiver-Modul, das die Hauptlogik mit Zustandsautomat enthält, wird erstellt und die Ports (Ein- und Ausgänge) werden definiert. Das Top-Modul mit der *Avalon*-Bus-Anbindung wird angelegt. Anschließend wird eine Testbench-Datei für das Transceiver-Modul erzeugt. In folgender Abbildung ist die entstandene Modul-Hierarchie dargestellt:

[1] Die bei der Generierung eingestellten Parameter können den generierten FIFO-Dateien entnommen werden (spi_master_receive_fifo.v u. spi_master_tranfer_fifo.v).

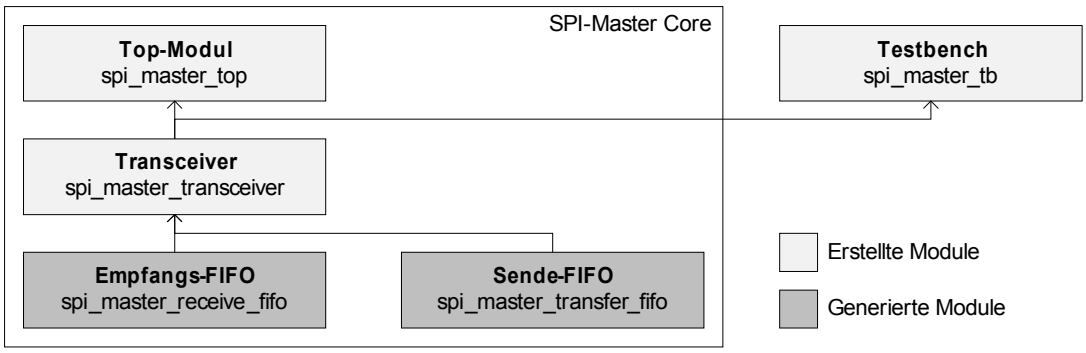


Abbildung 25: SPI-Master Modul-Hierarchie

4.1.3.3 Hardwarebeschreibung

Im Folgenden werden die Hardwarebeschreibungsdateien vorgestellt:

Die **Definitionsdatei** (siehe Anhang H.1.1) ist mit einer C-Header-Datei vergleichbar. In dieser werden die Konstanten für den Core definiert.

Das **Top-Modul** (siehe Anhang H.1.2) verfügt über ein *Avalon-MM*[1] Interface und stellt die Schnittstelle zum System dar. In diesem Modul werden die EA-Register deklariert und die Funktionen für den Registerzugriff implementiert. Der Aufbau des Top-Moduls gliedert sich in folgende Abschnitte:

1. Moduldefinition mit *Avalon-MM* sowie SPI-Ein- und Ausgangssignalen (siehe S. 87)
2. Parameterdefinition (siehe S. 88)
3. Register- und Variablendefinition, Erzeugung der Transceiver Instanz (siehe S. 88)
4. Logische Gleichungen (siehe S. 89)
5. Schreib- und Lese-Unterprogramme sowie Reset-Funktion für die EA-Register (siehe S. 90)
6. Taktabhängige Register-Zuweisungen und Unterprogrammaufruf (siehe S. 92)

Der **Transceiver** (siehe Anhang H.1.3) implementiert den Zustandsautomaten mit Sende- und Empfangslogik. Das Modul ist wie folgt aufgebaut:

1. Moduldefinition mit Ein- und Ausgängen (siehe S. 93)
2. Parameterdefinition (siehe S. 93)
3. Register- und Variablendefinition (siehe S. 93)
4. Erzeugung der FIFO Instanzen (siehe S. 94)
5. Logische Gleichungen (siehe S. 94)
6. Unterprogramme für Init und Reset (siehe S. 94)
7. Implementierung des in Abschnitt 4.1.3.1 definierten Zustandsautomaten (siehe S. 96)

[1] Das Avalon-MM (Memory-Mapped) Interface ist eine Busschnittstelle, über die Komponenten in den Adressraum des Systems integriert werden .

Die **Testbench** (siehe Anhang H.1.4) implementiert die Test-Funktionen für den Transceiver. Die Datei ist wie folgt aufgebaut:

1. Moduldefinition (siehe S. 99)
2. Register- und Variablendefinition, Erzeugung der Transceiver Instanz
3. Logische Gleichungen (siehe S. 100)
4. Takt-Generierung, Slave-Clock-Counter
5. Transceiver Test- Unterroutinen (siehe S. 100)
6. Haupttestprogramm; in diesem werden die Funktionen des Transceivers über den Aufruf von Unterroutinen getestet (siehe S. 103)

4.1.3.4 Testbench-Simulation

Während der Implementierungsphase wurde für jede SPI-Master-Operation eine Test-Funktion in der Testbench angelegt. Die Abbildung 26 zeigt die Wellenformdarstellung der Transfer-Operation in der RTL-Simulation. Das `state_idle` Signal ist zu Beginn der Operation auf *High* und signalisiert damit Operationsbereitschaft. Die auszuführende Operation wird über die `job` Variable festgelegt (0=Transfer Operation). Die Anzahl der zu übertragenden Bytes wird in der Variable `transfer_count` festgelegt. Das Transfer-FIFO wurde zuvor mit der Bytefolge 0x04, 0x03, 0x02, 0x01 gefüllt. Dies ist in der `tr_write_data` Variable abzulesen. Die Operation wird durch das Toggeln des `start_transfer` Signals gestartet. Der Core zieht `state_idle` auf einen *Low*-Pegel und die Operation wird ausgeführt. Die `sclk` (Slave Clock) wird generiert und die Daten werden an `mosi` (Master-Out-Slave-In) seriell ausgegeben. Die entworfene Testbench führt alle Operationen nacheinander aus.

Die resultierenden Signalverläufe werden optisch mit den erwarteten verglichen. Eine vollständige Test-Automation kann durch die Implementierung eines SPI-Slaves erreicht werden, der die empfangen Daten als Echo an den Master zurücksendet. Dieser kann dann die empfangenen Daten mit den gesendeten Daten automatisch vergleichen und ggf. einen Übertragungsfehler melden.

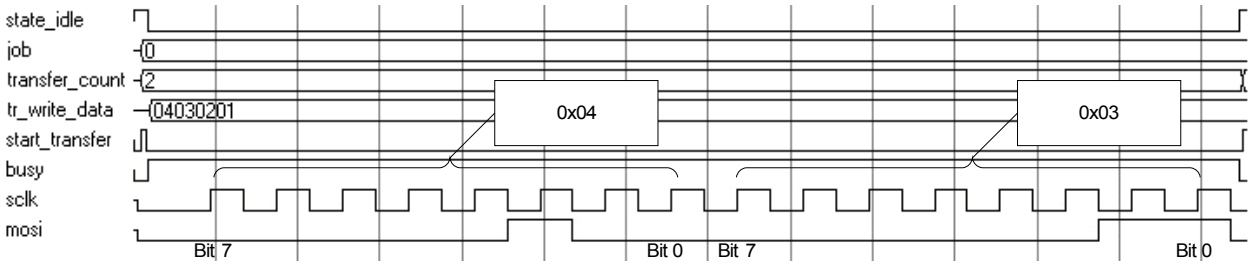


Abbildung 26: SPI-Master Simulation, Wellenformdarstellung der Transfer-Operation

4.1.3.5 System-Integration

Für den entwickelten Core wird, wie in Kapitel 2.3.4.2 erläutert, eine SOPC-Beschreibungsdatei angelegt. Anschließend wird die System-Integration durchgeführt. Der Core wird in die Periphe-

rie-Clock-Domain aufgenommen (siehe Abbildung 45). Diese wird mit 60 MHz betrieben. So kann durch eine vierfache Takt-Teilung eine SPI-Busfrequenz von 15 MHz erreicht werden. Dies liegt, wie einleitend beschrieben, nahe der maximal möglichen Takt-Frequenz von 16 MHz.

Die Anbindung erfolgt über eine Clock-Crossing-Bridge, um den Datentransfer mit dem mit 100-MHz getakteten Prozessor zu ermöglichen (siehe Abbildung 27) .

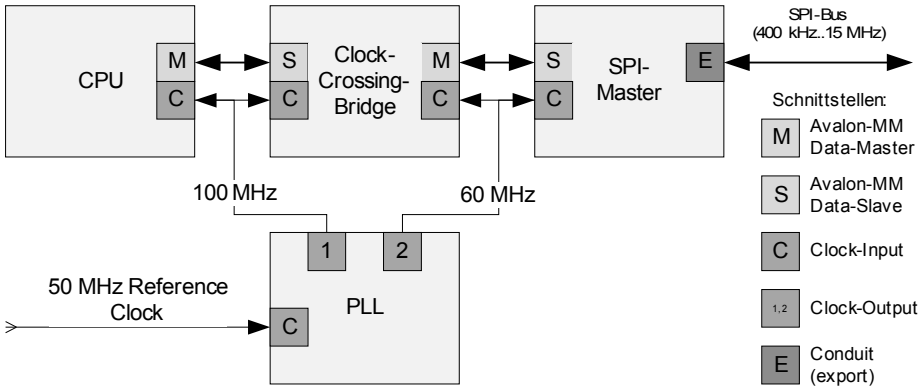


Abbildung 27: SPI-Master System Integration (Bussystem Darstellung)

Ein Test mit einem 65-MHz-Takt (16 MHz SPI-Frequenz) führte zu Datenübertragungsfehlern. Wegen der geringen, zu erwartenden, Performance-Steigerung bei der Takt-Erhöhung von 15- auf 16 MHz, wurde eine Untersuchung des Fehlers unterlassen.

4.1.3.6 Treiber Entwicklung

Die Treiber-Dateien werden in der Audio-Plattform-Bibliothek innerhalb des `device/spi_master` Verzeichnisses angelegt. Es wird eine `spi_master_regs.h` Datei für die EA-Register und Konstanten wie Start-Flag und Job-Bits erstellt (siehe Anhang I.4.1). Die `spi_master.h` Datei wird mit den Prototypen für die Treiber-Funktionen versehen (siehe Anhang I.4.2). In der `spi_master.c` werden die in der Tabelle 3 beschriebenen Funktionen implementiert (siehe Anhang I.4.3). Um zukünftig mehrere SPI-Cores parallel in einem System betreiben zu können, z.B. für den Betrieb mehrerer SD-Karten, werden alle Funktionen mit einem Zeiger auf eine `spi_dev_t` Device-Struktur versehen. Diese ist innerhalb des `device/spi` Verzeichnisses in der `spi.h` Datei definiert und enthält neben der Basisadresse und Busfrequenz einen „privaten“ Zeiger auf eine treiberspezifische Datenstruktur. Ein Testprogramm ist im Anhang G.1 zu finden.

Funktion	**Beschreibung**
`spi_master_init_device`	Initialisiert den Treiber
`spi_master_get_freq`	Liest die eingestellte SPI-Frequenz aus
`spi_master_set_freq`	Setzt die SPI-Frequenz
`spi_master_wait_for_token`	Wartet auf den Empfang eines Tokens (1-Byte-Datum)
`spi_master_wait_while_token`	Wartet, solange das angegebene Token empfangen wird

Funktion	Beschreibung
`spi_master_write_constant`	Schreibt eine Konstante n-Mal auf den SPI-Bus
`spi_master_read`	Liest Daten vom SPI-Bus
`spi_master_write`	Schreibt Daten auf den SPI-Bus
`spi_master_cs_enable`	Setzt das Chip-Select (CS) Signal auf Low
`spi_master_cs_disable`	Setzt das Chip-Select (CS) Signal auf High

Tabelle 3: SPI-Master Treiber-Funktionen

4.1.4 Ergebnis

Eine Messung der Datenübertragungsgeschwindigkeit wurde vor (mit dem *Opencore* SPI-Master) und nach der Implementierung des neu entwickelten Cores durchgeführt. Diese erfolgte mit dem *Performance Counter*. Dabei wurde die benötigte Zeit für das Lesen von 8192 Bytes gemessen[1]. Der Messvorgang wurde in einer Programmschleife mehrfach ausgeführt und anschließend das Ergebnis gemittelt. Die so ermittelten Werte zeigen bei der getesteten 1-GB Hama SD-Karte, eine Performanceverbesserung um den Faktor 2,3[2] (siehe Abbildung 28).

Der Geschwindigkeitsgewinn ist auf die Änderung der Busanbindung zurückzuführen. Aus diesem Grund ist davon auszugehen, dass die erzielte Verbesserung Karten unabhängig ist.

Die Abbildung enthält weitere Messergebnisse, die zwischen einzelnen Optimierungsphasen aufgezeichnet wurden (von links nach rechts zu lesen). Auf einen Teil davon wird in Kapitel 6.1.1.2 weiter eingegangen. Folgende Optimierungen wurden durchgeführt:

1. Puffer optimiert: Der SD-Treiber wurde so optimiert, dass er anhand einiger Parameter entscheiden kann, ob die Pufferung eines Blocks sinnvoll ist oder die direkte Speicherung der Daten im Zielspeicher durchgeführt werden soll
2. Pufferanzahl erhöht: Setzen der Pufferanzahl auf 5 (empirisch ermittelter max. Wert, eine Größere Pufferanzahl führte zu keiner messbaren Geschwindigkeitssteigerung)
3. Leseoptimiert: Es wurde der Bus-Wait-Mechanismus[3] im SPI-Master implementiert (hierbei wird der Bus solange in den Wartezustand versetzt, bis Daten im FIFO vorhanden sind)
4. Multi-Block-Read: Implementierung des Multi-Block-Read SD-Karten-Befehls, der die Übertragung von mehreren Blöcken hintereinander erlaubt

---

[1] Bei 8192 Bytes wird der SD-Treiber und der FAT-Treiber optimal genutzt, siehe Kapitel 6.1.1.5ff

[2] $\frac{6265\,\text{kbit/s}}{2777\,\text{kbit/s}} \cong 2{,}3$

[3] Der Bus-Wait-Mechanismus signalisiert einem Bus-Master (z.B. CPU), dass ein Slave (z.B. SPI-Core) noch nicht bereit ist für einen Datentransfer es resultiert ein Warte-Zyklus

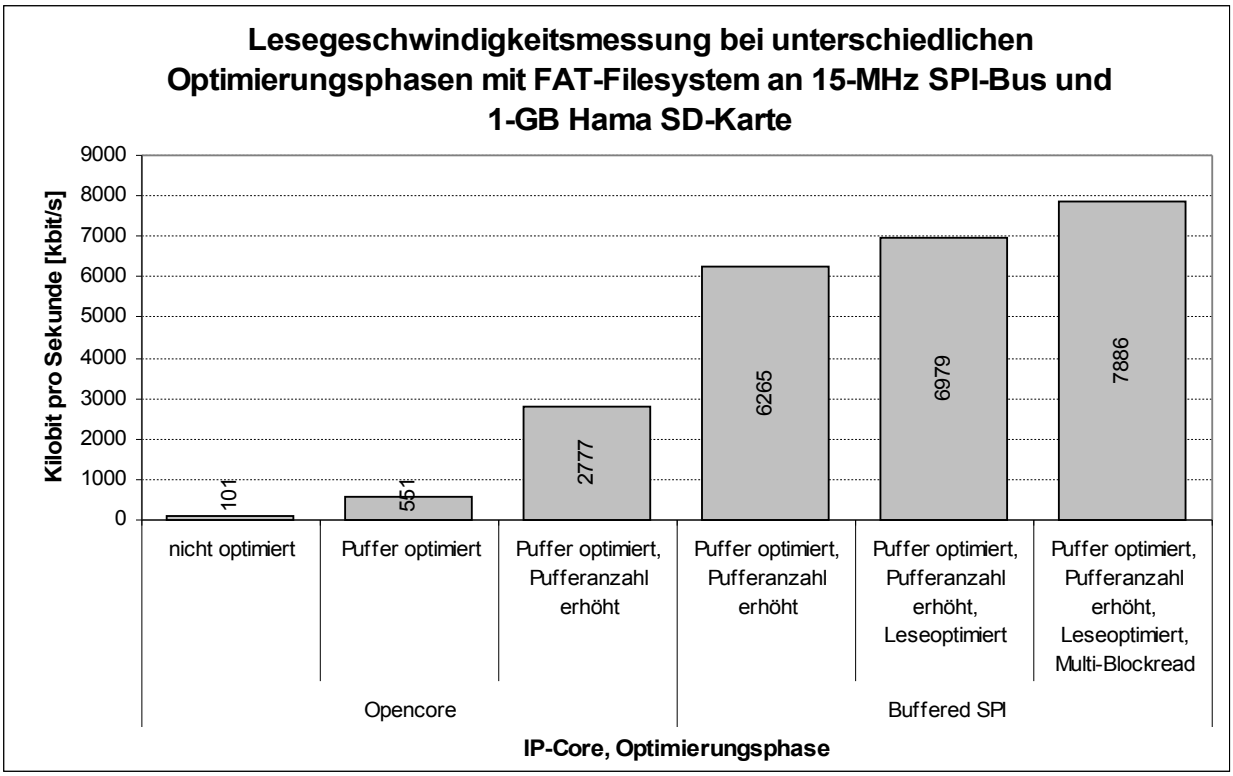


Abbildung 28: SD-Karten Lesegeschwindigkeitsmessung bei unterschiedlichen Optimierungsphasen

4.2 WM8731 IP-Core

Der auf dem Entwicklungssystem vorhandene WM8731-CODEC besitzt zwei mit dem FPGA verbundene Schnittstellen (siehe Abbildung 29).

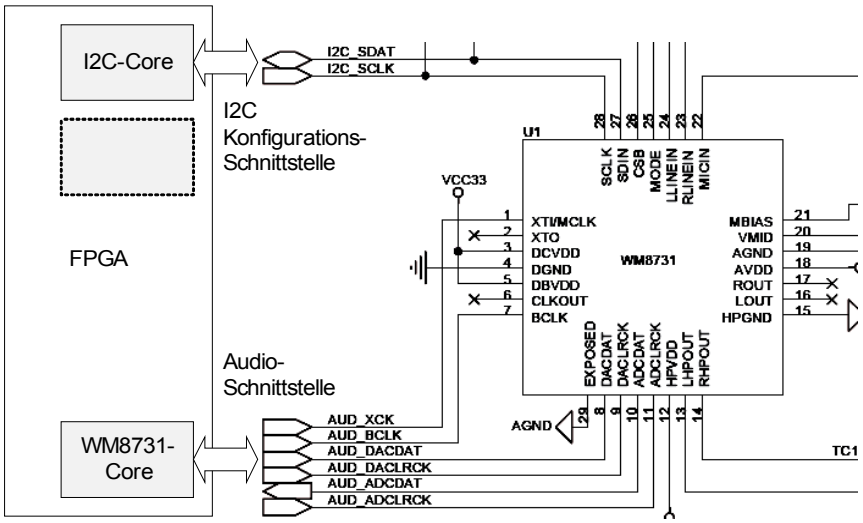


Abbildung 29: WM8731-CODEC Schnittstellen (Schaltungsausschnitt). Quelle: LCD Multimedia Daughtercard Reference Manual

Die I2C-Schnittstelle wird verwendet, um Steuerbefehle an den CODEC zu senden. Dies umfasst z.B. das Regeln der Ein- und Ausgangs-Lautstärke sowie die Konfiguration der Samplerate und -breite. Der gesamte Funktionsumfang kann aus dem Datenblatt entnommen werden . Das I2C-Konfigurations-Interface wird mit einem in das System zu integrierenden I2C-Core verbunden (siehe Kapitel 4.2.3.8). Die zweite Schnittstelle ist für die Verbindung des Bausteins mit den Audiodaten zuständig. Diese werden über ein serielles Protokoll[1] übertragen (siehe Abbildung 30).

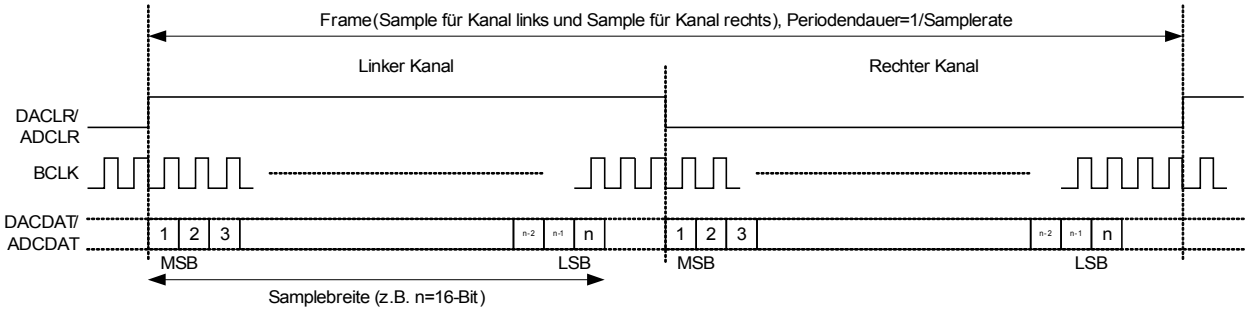


Abbildung 30: WM8731 Audiodaten Protokoll („Left Justified Mode“)

Für die Generierung des seriellen Datenstroms und der damit verbundenen Takte wird ein weiterer Core benötigt. Dieser wird neu entwickelt, da die von Altera verfügbare Komponente[2] ohne

[1] Der CODEC unterstützt verschiedene Protokolle zur seriellen Audiodatenübertragung. In der Arbeit wird der „Left Justified Mode“ verwendet.

[2] Teil der SD-Card-Audio Beispielanwendung, siehe: http://www.terasic.com/downloads/cd-rom/mtdb

ein Audiodaten-Eingangs-Interface implementiert ist und keine Laufzeit-Konfiguration der Sample-Takte (DACLR, ADCLR und BCLK) zulässt.

4.2.1 Ziel

Um den vollen Funktionsumfang des WM8731-ICs nutzen zu können, wird ein Core entwickelt, der folgende Anforderungen erfüllt: 32-Bit DMA Anbindung über *Avalon-ST*-Interface[1], konfigurierbare Samplerate und -breite sowie freie Ein- und Ausgangs-Kanalwahl.

4.2.2 Konzept

Der zu erstellende Core lässt sich in mehrere funktionale Einheiten gliedern. Diese sind in der folgenden Abbildung dargestellt. Auf der linken Seite sind die Schnittstellen für die System Integration abgebildet. Diese umfassen das *Avalon-ST*-Interface für die FIFO-zu-Speicher-Anbindung und das *Avalon-MM*-Interface für die Adressraum-Integration der Register. Auf der rechten Seite sind die Audio-Signale, die mit dem WM8731-IC verbunden sind, eingezeichnet.

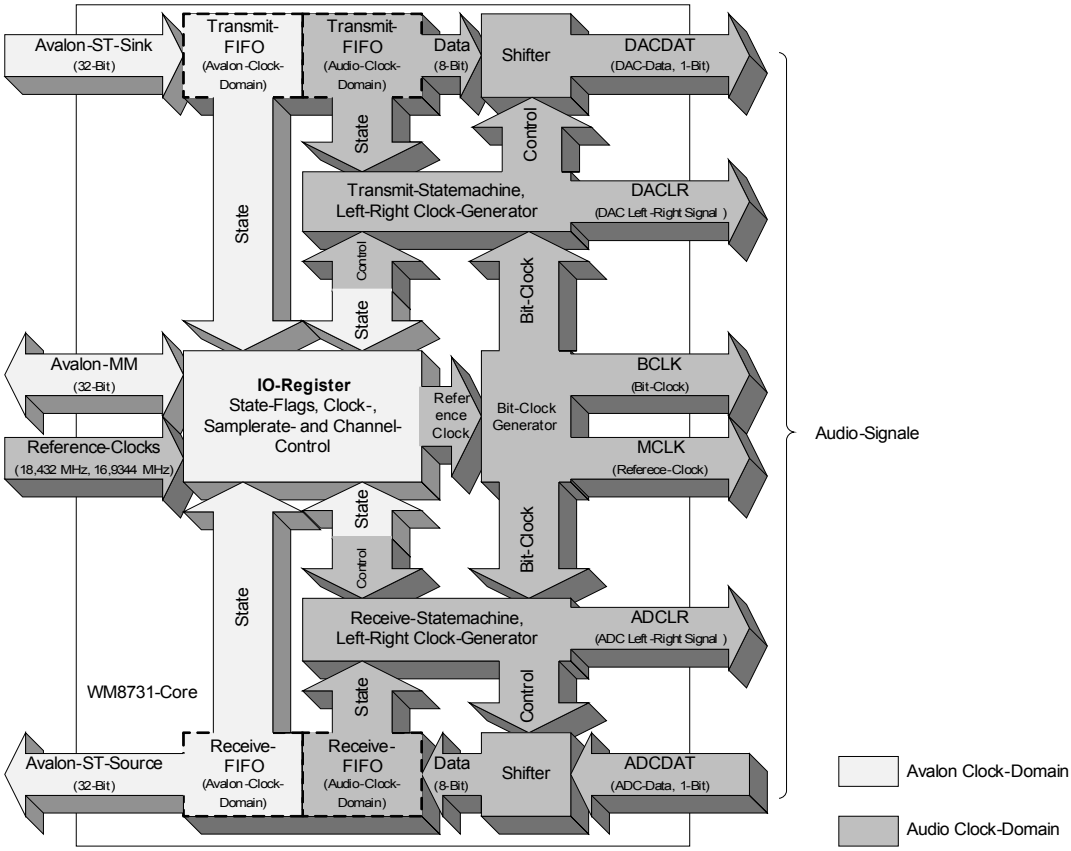


Abbildung 31: WM8731 IP-Core Blockdiagramm mit Darstellung der Clock-Domains

Der Core wird mit zwei unterschiedlichen Referenz-Takten versorgt[2], da es nicht möglich ist, alle vom CODEC unterstützen Sampleraten (z.B. 48 kHz und 44,1 kHz) aus einer Takt-Quelle mit einem ganzzahligen Teilerverhältnis zu generieren. Für die Auswahl des Referenz-Taktsi-

[1] Das Avalon-ST-Interface ist eine Busschnittstelle für Datenströme, die entweder Source (Quelle) oder Sink (Senke) sein kann. Sie besitzt im Gegensatz zum Avalon-MM keine Adressraum-Integration .

[2] 18,432 MHz für die Sampleraten 8-, 32-, 48- und 96 kHz oder 16,9344 MHz für 44,1- und 88,2 kHz

gnals (MCLK), die Teiler für den Bit-Takt (BCLK) und für die DACLR- und ADCLR-Takte werden Register benötigt. Diese sind, zusammen mit dem Status- und Kontroll-Register, über das eine Re-Konfiguration initiiert wird, in der folgenden Tabelle aufgeführt:

Register	**Beschreibung**
`REG_BCLK_DIVIDER`	Bit-Clock Teiler Register
`REG_DAC_DIVIDER`	DACLR Teiler Register
`REG_ADC_DIVIDER`	ADCLR Teiler Register
`REG_BYTES_PER_SAMPLE`	Anzahl der Sample-Bytes (2 oder 3)
`REG_TRANSMIT_FIFO_FILL_COUNT`	Sende-FIFO Füllzustand in Bytes
`REG_RECEIVE_FIFO_FILL_COUNT`	Empfangs-FIFO Füllzustand in Bytes
`REG_CLOCK_SELECT`	Referenz-Takt-Auswahl Register (0 oder 1)
`REG_CONTROL`	Steuerregister mit Konfigurations-Flag
`REG_STATUS`	Statusregister mit Transceiver- und FIFO-Zustand

Tabelle 4: WM8731-Core EA-Register

4.2.3 Implementierung

Vor der eigentlichen Modellierung wird der Zustandsautomat entworfen und Synchronisationsproblematiken erörtert. Anschließend wird die Logik in Verilog modelliert und mit *ModelSim-Altera* simuliert. Der Treiber wird implementiert und getestet.

4.2.3.1 Zustandsautomat

Der Core wird mit zwei Zustandsautomaten (siehe Abbildung 32) für die Sende- und Empfangslogik implementiert, die einen gemeinsamen *Idle*-Zustand besitzen. In diesem werden keine Daten übertragen oder empfangen. In den *Running*-Zustand wird nur dann gewechselt, wenn sich Sende- und Empfangslogik zuvor im *Idle*-Zustand befinden. Beim Wechsel in den *Running*-Zustand (`reconfiguration_req==1`) werden Sende- und Empfangslogik mit den neuen Einstellungen für die Takt-Teiler, Kanalwahl und der Sample-Breite konfiguriert. Wenn das Top-Modul eine Anfrage per `do_job` Signal stellt, wird aus dem *Running* Zustand in den *On-Even-Idle*-Zustand gewechselt. Dabei wird das `reconfigure_req` Register entsprechend des `job` Signals (0=go idle, 1=reconfigure) gesetzt. Aus dem *On-Even-Idle*-Zustand wird erst dann in den *Idle*-Zustand gewechselt, wenn die Anzahl der bisher übertragenen bzw. empfangenen Bytes durch zwölf teilbar ist. Diese Logik soll dabei helfen, nur „vollständige“ Frames[1] in bzw. aus den FIFOs zu transferieren (siehe Kapitel 4.2.3.2).

[1] Ein Frame ist hier definiert als: 2-Byte bei 16-Bit Einkanal, 6 Byte bei 24-Bit Stereo, usw.

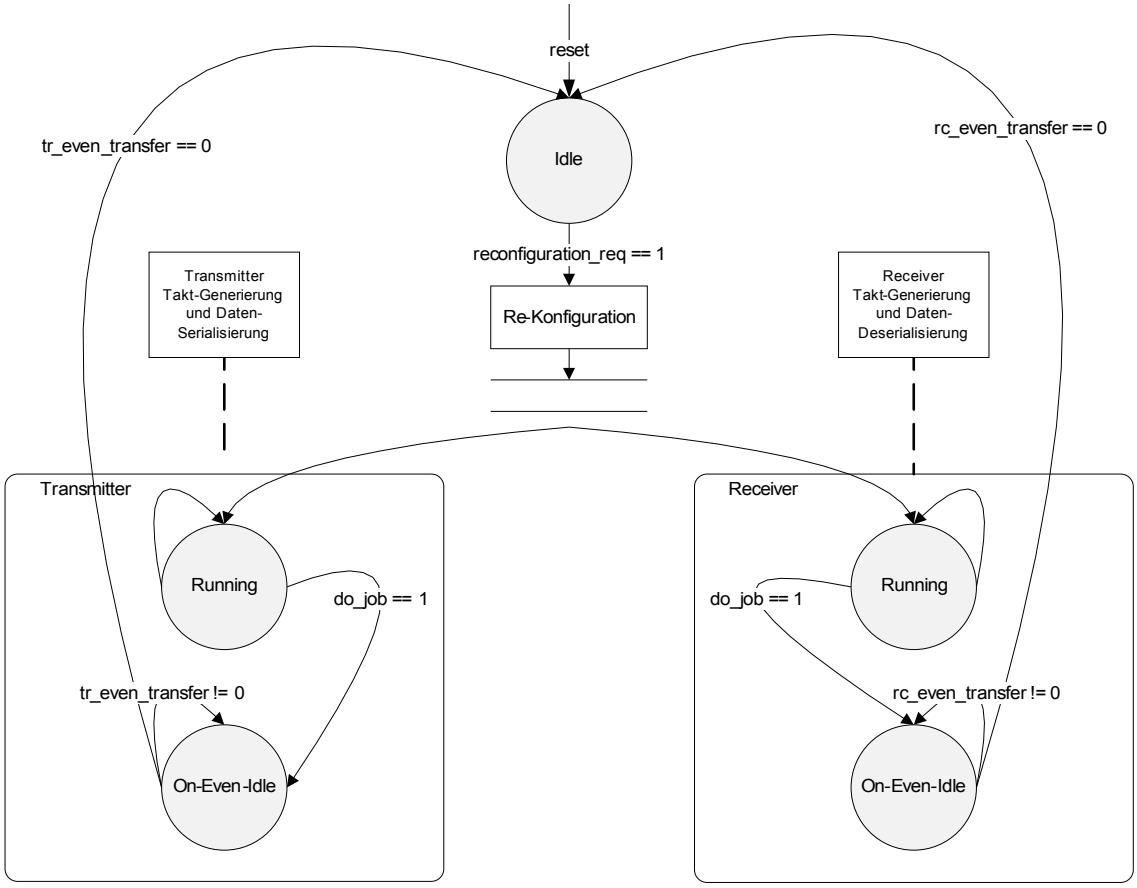


Abbildung 32: WM8731 IP-Core Sende- und Empfangs-Zustandsautomaten mit gemeinsamem Idle-Zustand

4.2.3.2 Konfigurations-Problematik

Folgendes Szenario bedarf näherer Betrachtung: Die DMAs auf der System-Seite füllen und leeren die angeschlossenen FIFOs. Dabei werden busbedingt immer mindestens vier Byte parallel gelesen und geschrieben. Die Audio-CODEC-Seite muss, um die Anforderungen zu erfüllen, mit unterschiedlichen Sample-Breiten im Stereo- sowie im Einkanal-Betrieb arbeiten können. Eine Umschaltung von einer Sample-Breite zur anderen erfordert eine Systematik, die mit dem Umstand zurechtkommt, dass ein FIFO noch teilweise mit Samples der „alten“ Breite gefüllt sein kann. Dieses Problem wird durch folgende Grafik erläutert:

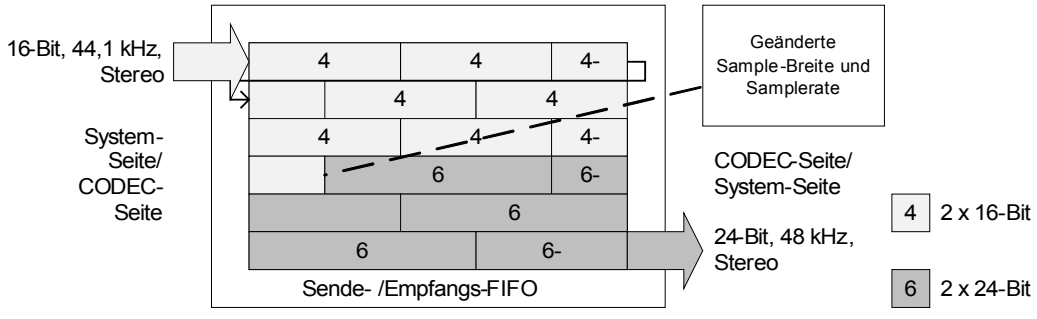


Abbildung 33: Sende- bzw. Empfangs-FIFO mit unterschiedlich breiten Samples gefüllt

Wie oben zu erkennen, ist das FIFO mit unterschiedlichen Frames gefüllt. Dieses Problem tritt dann auf, wenn z.B. zuerst eine Musikdatei mit 24-Bit-breiten Samples abgespielt wurde und anschließend eine mit 16-Bit-breiten. Die zu findende Logik muss die Re-Konfiguration zur richtigen Zeit durchführen, damit keine „halben“ Samples in den FIFOs verbleiben. Dies könnte passieren, wenn zum Konfigurationszeitpunkt die ersten vier Byte eines 6-Byte-Frames bereits übertragen wurden. Dies würde alle nachfolgenden Samples um zwei Byte aus der Reihe schieben und damit einen nicht wieder herstellbaren Synchronisationsverlust auslösen.

Eine Lösung kann darin liegen, die FIFOs vor der Re-Konfiguration zu leeren. Dann muss zusätzlich sichergestellt sein, dass die sich im Speicher befindlichen Ring-Puffer, DMA-interne FIFOs usw., ebenfalls geleert sind – eine sehr gute, aber schwer zu implementierende Lösung.

Der gefundene Lösungsweg ist einfacher, aber nicht fehlerfrei. Dabei werden die Samples bzw. Samplepaare (Frames) zu Paketen von zwölf Bytes gruppiert. Dies entspricht dem kgV[1] von 2-Byte für 16-Bit-Einkanal, 4-Byte für 16-Bit-Stereo, 3-Byte für 24-Bit-Einkanal und 6-Byte für 24-Bit-Stereo. Es muss sichergestellt sein, dass immer nur ein Vielfaches von zwölf Bytes aus bzw. in die FIFOs gelangt. Eine Re-Konfiguration kann dann an einer „Paket“-Grenze durchgeführt werden, ohne die Synchronisation zu verlieren. Für die Zeit einer FIFO-Leerung können Samples mit falscher Samplerate oder -breite übertragen werden, da die Logik keine Kenntnis über den Inhalt der FIFOs hat (Blackbox). Ein vollständiger Synchronisationsverlust wird so jedoch vermieden. Diese Logik wird durch einen Byte-Zähler pro FIFO im Zustandsautomaten erreicht. Zusätzlich muss im Treiber dafür gesorgt werden, dass immer nur ganze Pakete in die Ring-Buffer und zur DMA gelangen.

4.2.3.3 Asynchrone Datenübertragung

Der Zustandsautomat arbeitet im Audio-Referenz-Takt, wahlweise mit 18,432 MHz oder 16,9344 MHz. Die EA-Registeranbindung des Cores erfolgt über den *Avalon*-Bus mit 66,5 MHz (siehe 4.2.3.8). Hierdurch kommt es bei der Verarbeitung von Steuerinformationen, z.B. beim Setzen der Samplerate und der anschließenden Konfiguration der Zustandsmaschine, zu asynchronen Signal-Übergängen. Diese werden als Clock-Domain-Crossing (CDC) bezeichnet. In Registern kann dies zu Setup- und Hold-Violations führen, die einen metastabilen Zustand[2] auslösen können. Um dies zu verhindern, wird dem Core für die Ansteuerung des Zustandsautomaten eine Signal-Synchronisations-Logik hinzugefügt (siehe Abbildung 34) .

Für die Audio-Datenübergabe von der *Avalon*- zur Zustandsautomaten-Seite werden Dual-Clock FIFOs eingesetzt, die jeweils für den Datenein- und Datenausgang unterschiedliche Taktsignale verwenden .

---

1 Kleinste gemeinsame Vielfache

2 Zustände, die weder logisch Null noch Eins sind

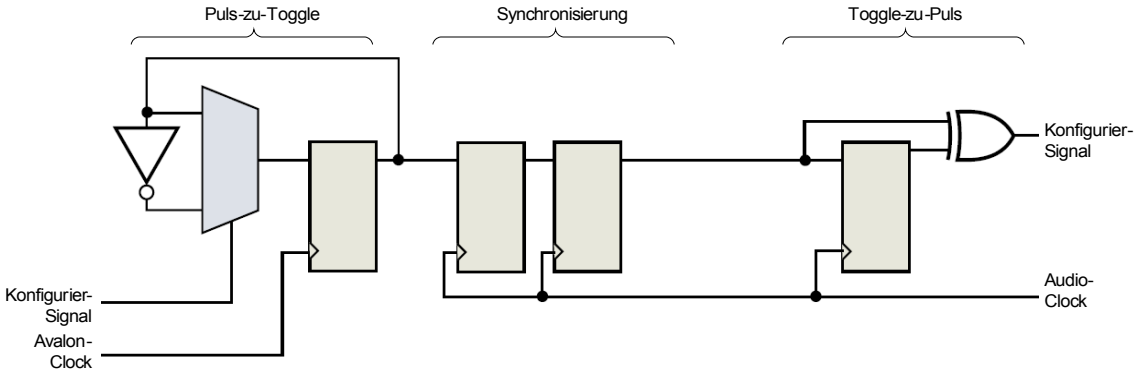


Abbildung 34: Signal Synchronisations-Logik

4.2.3.4 Glitch-freie Takt-Umschaltung

Eine Problematik betrifft die Audio-Referenz-Takt-Umschaltung. Wie im Konzept beschrieben, wird der Audio-Takt gewechselt, um alle Sampleraten des CODECS zu unterstützen. Ein einfaches „multiplexing“ kann hier nicht verwendet werden, da das resultierende Signal ein Taktsignal ist und hierfür besondere Stabilität gefordert ist. Durch eine nicht-stabilisierte Umschaltung der Quell-Takte können Glitches[1] entstehen, die in nachfolgenden Registern metastabile Zustände auslösen können. Folgende Abbildung zeigt die zur Lösung verwendete Logik :

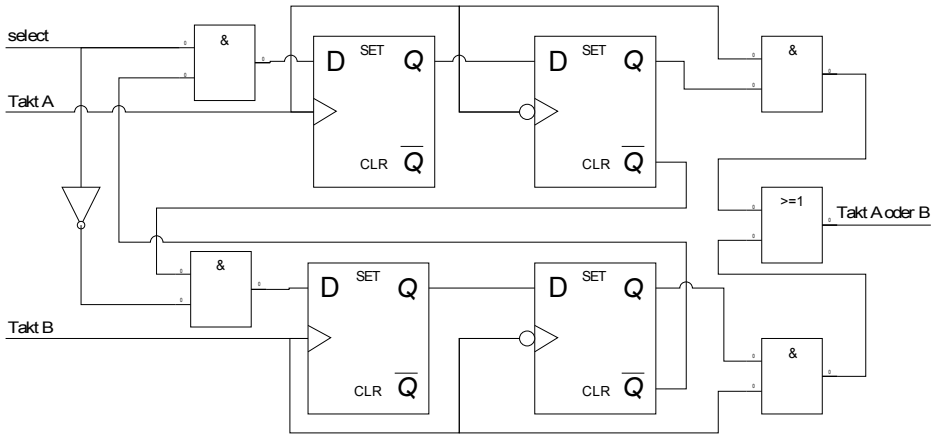


Abbildung 35: Glitch-freie Takt-Umschaltung

4.2.3.5 Projekt- und Modul-Struktur

In *Quartus II* wird das `wm8731` Projekt sowie ein `hdl` Unterverzeichnis für die Verilog-Dateien angelegt. Die Aus- und Eingangs-FIFOs werden mit dem *MegaWizard Plugin Manager*, basierend auf dem Dual-Clock-FIFO[2], generiert . Die Konstanten-Definitionsdatei wird angelegt. Das Transmitter-Modul, das die Hauptlogik mit Zustandsautomat und den Takt-Generatoren enthält, sowie das Top-Modul mit der *Avalon*-Bus Anbindung, werden erstellt. Anschließend wird eine

[1] Kurzzeitige Falschaussage in logischen Schaltungen

[2] Die eingestellten Parameter können den generierten FIFO-Dateien entnommen werden.

Testbench für den Transmitter angelegt. Folgende Abbildung zeigt die entstandene Modul-Hierarchie:

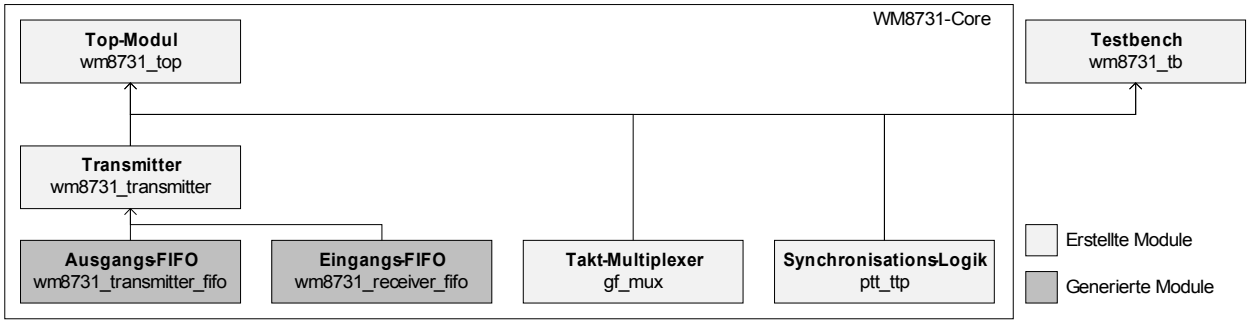


Abbildung 36: WM8731 IP-Core Modul-Hierarchie

4.2.3.6 Verilog Hardwarebeschreibung

Im Folgenden werden die Hardwarebeschreibungsdateien vorgestellt:

Die **Definitionsdatei** (siehe Anhang H.2.1) enthält Konstanten für die EA-Register, Zustände, Register-Größe-Definitionen usw.

Das **Top-Modul** (siehe Anhang H.2.2) verfügt über ein *Avalon-MM* Interface für den Registerzugriff und zwei *Avalon-ST*-Schnittstellen (Sink und Source) für den Datenaustausch per DMA. Der Aufbau des Top-Moduls gliedert sich in folgende Abschnitte:

1. Moduldefinition mit Ein- und Ausgängen (siehe S. 105)
2. Register- und Variablen- Definition
3. Erstellung der Transmitter-, der Datensynchronisationslogik- und der Audio-Referenz-Takt-Umschaltungs Instanz (siehe S. 106)
4. Logische Gleichungen für die Bus-Steuerung
5. Eintaktung der Transmitter-Zustandsautomaten Status (siehe S. 107)
6. Steuerung für den Registerzugriff (*Avalon-MM*)

Der **Transmitter** (siehe Anhang H.2.3) enthält die eigentliche Datenübertragungslogik. In ihm werden die FIFO Instanzen erzeugt und der Zustandsautomat mit den Takt-Generatoren implementiert. Die Datei ist wie folgt aufgebaut:

1. Moduldefinition mit Ein- und Ausgängen (siehe S. 109)
2. Register- und Variablen-Definition
3. Erstellung der FIFO Instanzen
4. Logische Gleichungen für die Statuszuweisung, den FIFO Takt und DAC-Signal-Out
5. Unterprogramme zur Zustandsautomat-Initialisierung und Frame-Übertragung
6. Transceiver-Zustandsautomat und Takt-Generatoren (siehe S. 113)

Der **Takt-Multiplexer** zur Glitch-freien Taktumschaltung (siehe 4.2.3.4) ist Teil des separaten `gf_mux` Projektes und ist im Anhang H.2.4 abgedruckt.

Die **Signal-Synchronisations-Logik** zur Signalübertragung zwischen Clock-Domains (siehe 4.2.3.3) ist in dem separaten `cross_domain_sync` Projekt definiert und im Anhang H.2.5 zu finden.

Die **Testbench** (siehe Anhang H.2.6) implementiert die Funktionen zum Testen des Transmitter-Moduls. Die Testbench ist wie folgt aufgebaut.

1. Moduldefinition (siehe S. 118)
2. Register- und Variablen-Definition
3. Instanziierung des Transmitters und der Synchronisations-Logik
4. Test-Takt-Generierung
5. FIFO-Füll- und Leer-Funktionen, Re-Konfigurations-Unterprogramm (siehe S. 119)
6. Testprogramm (in dem zuerst ein Reset des Transmitters ausgeführt wird und anschließend unterschiedliche Konfigurationen initialisiert werden) (siehe S. 121)

4.2.3.7 Testbench-Simulation

Für Simulation des Cores wurde eine Testbench angelegt. Durch diese konnte während der Implementierungsphase die Funktionalität des Transmitters verifiziert werden. Der hier abgebildete Signalverlauf zeigt einen Ausschnitt aus der RTL-Simulation. Zu sehen ist das generierte Bit-Clock-Signal (`bclk`) sowie das Frame-Synchronisations-Signal (`daclr_clk`). Die Ausgabe der Daten erfolgt seriell über das `dac_o` Signal. Außerdem sind die Steuerleitungen für die Sample-Breite `bytes_per_sample` und der Sende-FIFO-Füllstand `tr_rdusedw` zu erkennen.

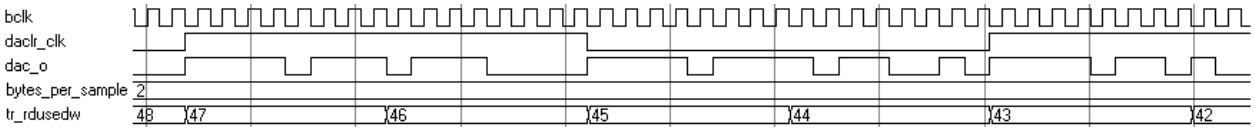


Abbildung 37: Wellenformdarstellung der RTL-Simulation

Zur Überprüfung der generierten Signale wurden diese manuell mit dem im Datenblatt und in Abbildung 30 dargestellten Signalverlauf verglichen.

4.2.3.8 System-Integration

Für den entwickelten Core wird, wie in Kapitel 2.3.4.2 erläutert, eine Komponenten-Beschreibungsdatei angelegt. Hierbei ist zu beachten, dass der Core über drei unterschiedliche *Avalon*-Schnittstellen verfügt (*Avalon-MM*-Slave, *Avalon-ST*-Sink und *Avalon-ST*-Source). Anschließend wird der Core mit dem *SOPC-Builder* in das System integriert. Zusätzlich wird ein PLL für die Audio-Referenz-Takt-Generierung hinzugefügt. Die Speicheranbindung zum RAM wird durch das Hinzufügen einer Receive- und Transfer-DMA erreicht. Die Abbildung 38 stellt die System Integration schematisch dar:

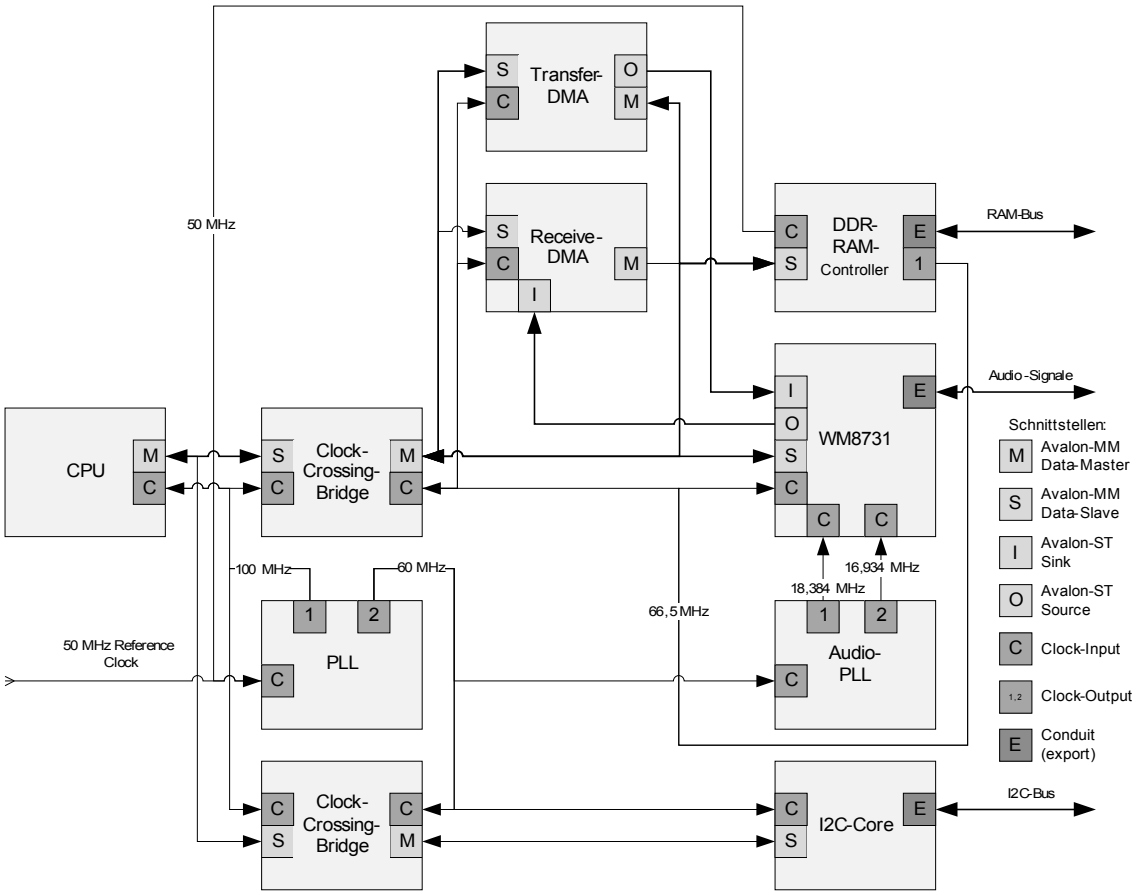


Abbildung 38: WM8731-Core System Integration (Bussystem Darstellung)

Nach der Implementierung des PLLs in das System stellte sich heraus, dass die geforderten Audio-Takt Referenz-Frequenzen nicht exakt zu generieren sind. Die geforderten Referenz-Frequenzen und die gemessenen Frequenzen sind in der folgenden Tabelle dargestellt. Durch die Abweichung entsteht eine minimale Tonhöhenveränderung, die ggf. bei der Anwendungsentwicklung berücksichtigt werden muss.

Referenz-Takt [Hz]	Samplerate [Hz]	Gemessener-Takt [Hz]	Res.-Samplerate [Hz]	Rel.-Abweichung [%]
18432000	48000	18389421	47889,11719	0,231
16934400	44100	16937622	44108,39063	0,019

Tabelle 5: Abweichungen von den Referenz-Frequenzen

4.2.3.9 Treiber Entwicklung

Die Treiber-Dateien werden in der Audio-Plattform-Softwareibliothek im `device/wm8731` Verzeichnis erstellt.

Das Konfigurations-Interface befindet sich in der `wm8731.h` Header-Datei. In dieser sind die Prototypen und Konstanten für die Steuerung des CODEC definiert. Die zugehörige Implementierung befindet sich in der `wm8731.c` Datei. Nachfolgend werden die wichtigsten Funktionen in

Tabellenform dargestellt. Die zugehörigen CODEC-Register sind in runden Klammern angegeben .

Funktion	Beschreibung
wm8731_set_lineout_vol	Line-out Lautstärkeregelung (R0+R1)
wm8731_set_linein_vol	Line-in Lautstärkeregelung (R2+R3)
wm8731_set_analog_path	Analoge Signalpfad Steuerung (R4)
wm8731_set_digital_path	Digitale Signalpfad Steuerung (R5)
wm8731_set_power_reg	Power Register (R6)
wm8731_set_interface	Digital Audio Interface Format (R7)
wm8731_set_sampling	Einstellung der Samplerate (R8)
wm8731_set_active	An-/Aus Funktion(R9)
wm8731_reset	Reset Funktion (R10)

Tabelle 6: WM8731 I2C-Konfigurations-Interface-Funktionen

Das Steuer-Interface des WM8731 IP-Cores wird in der wm8731_device.h Header-Datei definiert. Diese enthält die wm8731_audio_dev_t Device-Struktur, in der die EA-Basisadresse, die Stream- und Konfigurations-Device Strukturen referenziert werden. Die Register- und Konstantendefinitionen sind in der wm8731_regs.h definiert. Die Implementierung ist in der Datei wm8731_device.c zu finden. Die Treiber-Funktionen sind in folgender Tabelle aufgeführt:

Funktion	Beschreibung
wm8731_audio_init_device	Initialisierungsfunktion, integriert den Treiber in die HAL
wm8731_audio_exit_device	Entfernt den Treiber aus der HAL
wm8731_audio_set_sampling_rate	Sampleraten Konfigurationsfunktion

Tabelle 7: WM8731 IP-Core Treiber-Funktionen

4.2.4 Ergebnis

Die Erstellung des WM8731 IP-Cores und die System-Integration konnten wie geplant durchgeführt werden. Der Treiber wurde mit verschiedenen Audio-Dateien[1] in unterschiedlichen Sampleraten (8-, 32-, 41,1 und 48 kHz) mit 16-Bit Samplebreite im Einkanal- bzw. Stereo-Betrieb erfolgreich getestet.

Es wird eine minimale Tonhöhenveränderung durch die nicht exakt generierbaren Referenz-Frequenzen erzeugt. Anwendungen, die auf eine exakte Audio-Referenz-Frequenz angewiesen sind müssen einen externen Taktgeber integrieren.

Die Re-Konfigurations-Problematik konnte durch die Gruppierung der Samples zu Zwölf-Byte-Paketen umgangen werden.

[1] Die Audiotestdateien befinden sich auf der CD-ROM im Audiotest Verzeichnis

4.3 64-Bit-ALU IP-Core

Eine Echtzeit-MP3-Decodierung mit dem Helix-Decoder ist auf dem *Nios II*, ohne die in der Dokumentation geforderte 32-Bit-Multiplikation mit 64-Bit-Ergebnis, nicht zu erreichen (siehe Kapitel 6.2). Das ergab eine Geschwindigkeitsmessung[1], die vor der Implementierung der Prozessor-Erweiterung durchgeführt wurde.

4.3.1 Ziel

Um den MP3-Decoder-Algorithmus in Echtzeit auszuführen, ist der Standard-Befehlssatz des *Nios II* ist nicht ausreichend. Aus diesem Grund wird eine ALU entwickelt, die den Prozessor um die oben genannten mathematischen Funktionen erweitert. Eine weitere Geschwindigkeitsverbesserung wird durch die Hardwareimplementierung von mathematischen Spezialfunktionen erwartet. Dies betrifft die im Decoder verwendete Count-Leading-Zeros[2]- sowie die Absolut- und die 32-zu-16-Bit-Clip-Funktion.

4.3.2 Konzept

Der Standard-Befehlssatz des *Nios II* kann über das *Custom Instruction Interface* erweitert werden . Die Zusatz-Befehle werden über freie Opcodes adressiert. Es können bis zu 256 zusätzliche Befehle integriert werden. Jeder Befehl kann mit zwei 32-Bit Quell-Registern und einem 32-Bit Ziel-Register ausgeführt werden. Die Rückgabe eines 64-Bit Wertes ist nicht direkt möglich. Dies führt zur Aufteilung der gewünschten mathematischen Funktionen in mehrere Befehle. So wird die 64-Bit-Multiplikation auf zwei Befehle aufgeteilt. Der erste Befehl multipliziert die beiden 32-Bit-Operanden und speichert das 64-Bit-Ergebnis in einem internen Akkumulator ab. Anschließend werden die höherwertigen 32-Bit als Ergebnis zurück gegeben. Der zweite Befehl dient dazu, einen gespeicherten Zahlenwert aus dem Akkumulator in das Zielregister zu laden und diesen gleichzeitig um eine bestimmte Anzahl von Bits nach rechts zu verschieben (Division durch 2^n). Der gesamte Befehlssatz ist in Tabelle 36 im E beschrieben. Nachträglich wurde ein weiterer Akkumulator hinzugefügt um zwei Summen parallel bilden zu können.

Abbildung 39 zeigt den schematischen Aufbau der ALU. Die Eingangsoperanden und das Ergebnis sind dunkel hervorgehoben. Die mathematischen Blöcke sind schraffiert dargestellt.

---

[1] Es wurde die Zeit gemessen, die der Prozessor (ohne Optimierung) für die Decodierung einer MP3-Datei mit 30 Sekunden Länge und 128 kbit/s Bitrate benötigt. Diese lag deutlich über 30 Sekunden.

[2] Es wird durch diese Funktion die Anzahl der zusammenhängenden Null-Bits in einem Zahlenwert ermittelt, beginnend mit dem höchsten Bit.

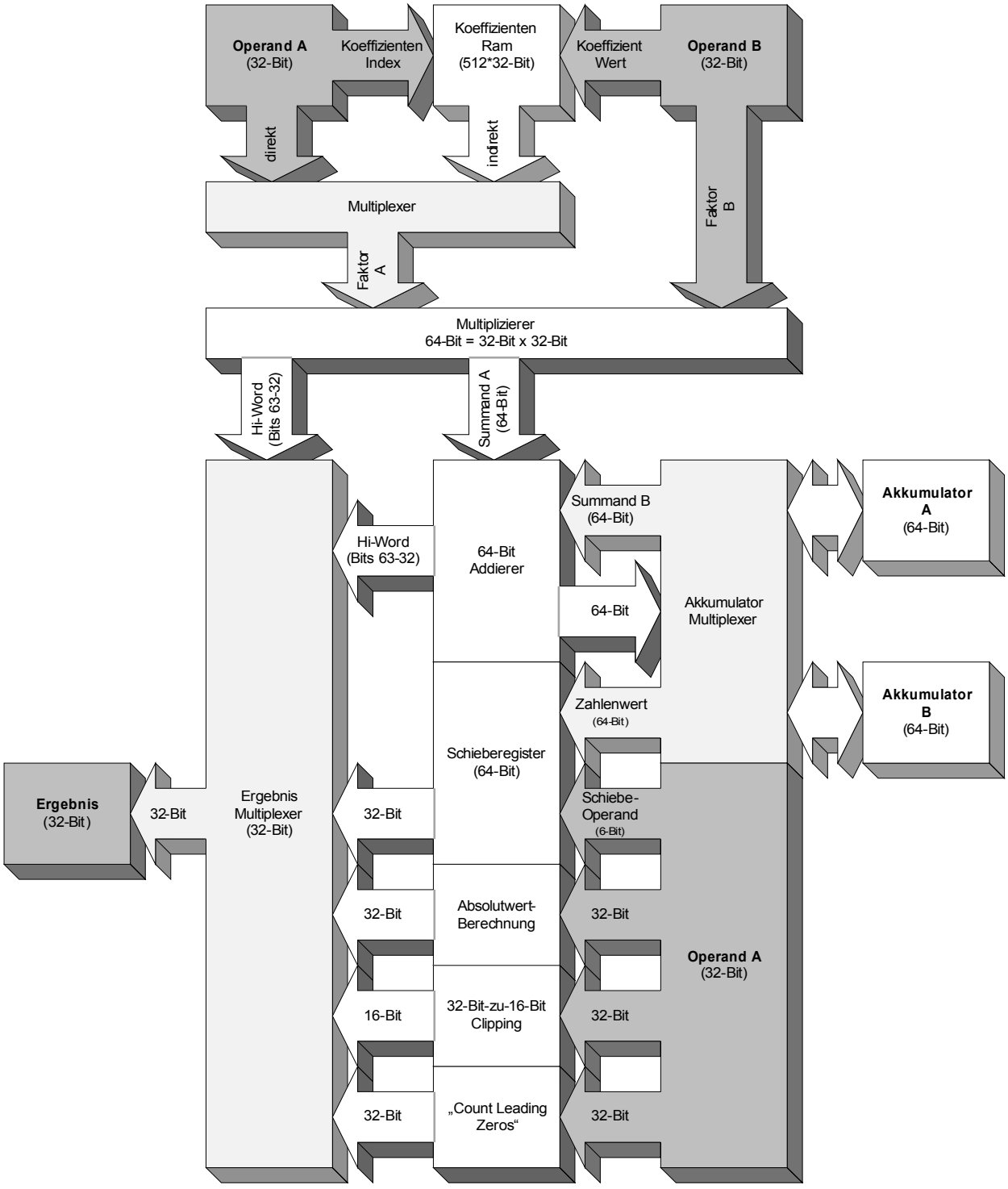


Abbildung 39: Schematischer Aufbau der 64-Bit-ALU

4.3.3 Implementierung

Bevor die Hardware modelliert wird werden Strukturmaßnamen durchgeführt. Anschließend erfolgt die Modellierung in Verilog. Eine Testbench wird entworfen und die Simulation mit *Altera-ModelSim* durchgeführt. Danach wird die ALU in das System integriert und eine Header-Datei für die Anwendungsentwicklung erstellt.

4.3.3.1 Projekt- und Modul-Struktur

In *Quartus II* wird das `mult64` Projekt und ein `hdl` Unterverzeichnis für die Verilog-Dateien angelegt. Der Koeffizientenspeicher wird basierend auf der 1-Port-RAM *Megafunction* mit dem *MegaWizard Plugin Manager* erstellt . Eine Definitionsdatei mit den definierten Opcodes und weiteren Konstanten wird angelegt. Das Top-Modul mit der Prozessor-Anbindung wird unter Zuhilfenahme der in der *Custom-Instruction*-Dokumentation gezeigten Vorlage erzeugt . Die Schiebe- (lpm_clshift) und Multiplikations-*Megafunction* (lpm_mult) werden im Top-Modul instanziiert . Abschließend wird die Testbench-Datei mit den Testroutinen für den Core angelegt. Folgende Abbildung zeigt die entstandene Modul-Hierarchie:

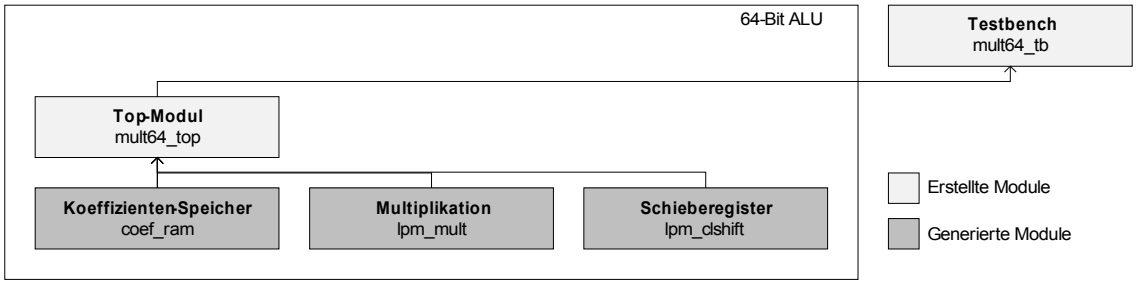


Abbildung 40: 64-Bit-ALU Modul-Hierarchie

4.3.3.2 Verilog Hardwarebeschreibung

Im Folgenden werden die Hardwarebeschreibungsdateien vorgestellt:

Die **Definitionsdatei** (siehe Anhang H.3.1) enthält die Opcode-Konstanten.

Das **Top-Modul** (siehe Anhang H.3.2) enthält die Programmlogik und beinhaltet die *Custom-Instruction*-Schnittstelle zur Anbindung an den Prozessor. Der Aufbau der Datei gliedert sich in folgende Abschnitte:

1. Moduldefinition mit *Custom Instruction* Interface (siehe S. 122)
2. Parameterdefinition (siehe S. 123)
3. Register- und Variablen-Definition (siehe S. 123)
4. Instanziierung des Koeffizientenspeichers, Multiplikators und des Schieberegisters mit den zugehörigen Parametern (siehe S. 123)
5. Logische Gleichungen für Multiplexer usw. (siehe S. 124)
6. Unterprogramm zur Initialisierung der gewählten Operation (siehe S. 124)
7. Hauptlogik mit taktabhängigen Register-Zuweisungen (siehe S. 126)

Die **Testbench** (siehe Anhang H.3.3) implementiert die Test-Funktionen für die ALU und ist wie folgt aufgebaut:

1. Moduldefinition (siehe S. 129)
2. Register- und Variablen-Definition (siehe S. 129)

3. Instanziierung des Top-Moduls (siehe S. 129)
4. Takt-Generierung (siehe S. 129)
5. Unterprogramme für die ALU-Befehle (siehe S. 129)
6. Testprogramm zur Verifikation der ALU durch Unterprogrammaufrufe und Ergebnisvergleich (siehe S. 130)

4.3.3.3 Testbench-Simulation

Zur Verifikation der ALU mittels RTL-Simulation wurde für jeden Befehl (siehe E) eine Funktion in der Testbench angelegt. Fehlfunktionen werden durch Ergebnisvergleiche automatisch erkannt und der Testvorgang ggf. mit einem Fehler abgebrochen. Die Abbildung 41 zeigt die Wellenformdarstellung des folgenden Testbench Ausschnitts.

```
1 STORE(0, 2);              // Setzt Werte im Koeffizientenspeicher
2 STORE(1, 3); STORE(2, 4);
3 LOAD_A(0);                // Lädt Accu A mit 0
4 MULT_A_ADD_Q(0, 300);     // Multipl. den Koeff. 0 mit 300 und addiert
5 MULT_A_ADD_Q(1, 200);     // Multipl. den Koeff. 1 mit 200 und addiert
6 GET_A(0);                 // Liefert das Ergebnis (ungeschiftet)
```

Quelltext 1: 64-Bit-ALU Testbench Quelltext-Ausschnitt

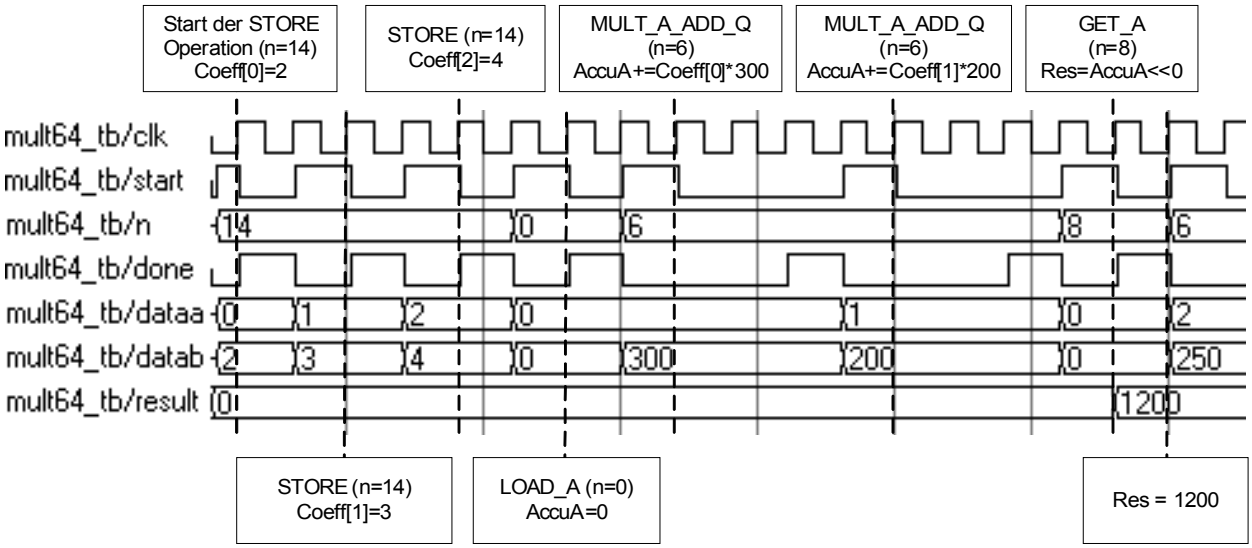


Abbildung 41: 64-Bit-ALU RTL-Simulation

4.3.3.4 System-Integration

Um die ALU in das System zu integrieren, wird (wie in Kapitel 2.3.4.1 beschrieben) eine SOPC-Beschreibungsdatei[1] angelegt. Anschließend wird die ALU über die „Custom Instructions“-Option des *Nios II* Einstellungs-Dialogs in das System eingefügt. Bei der Integration können folgende Parameter verändert werden:

[1] mult64_hw.tcl

Parameter	Beschreibung
MAX_SHIFT_DISTANCE_BITS_PER_CYCLE	Definiert die maximal pro Zyklus zu schiebenden Bits. Default Wert: 3
MULT_CYCLES	Defiiniert die Anzahl der Multiplikations-Zyklen. Default Wert: 2

Tabelle 8: 64-Bit-ALU Parameter

Die folgende Abbildung zeigt einen Ausschnitt der erfolgten SOPC-Builder Integration:

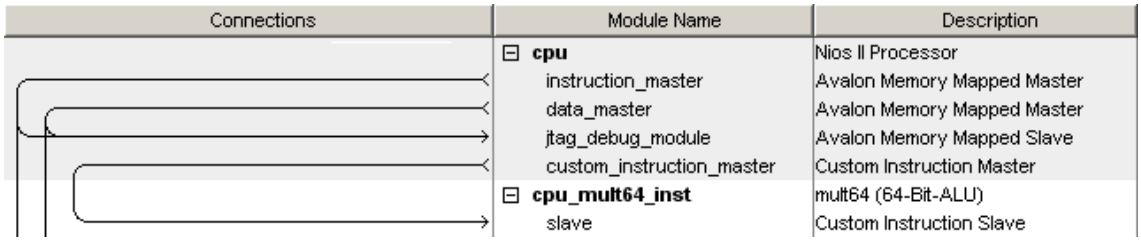


Abbildung 42: 64-Bit-ALU System-Integration (SOPC-Builder Ausschnitt)

4.3.3.5 Header-Datei

Die `alu64.h` Datei definiert den ALU-Befehlssatz über spezielle *Nios II* Compiler Built-In Functions (siehe Anhang I.1) .

Bei der Integration weiterer *Custom Instructions* in das System ist ggf. eine Anpassung der „Custom Instruction Number“[1] nötig.

4.3.4 Ergebnis

Die 64-Bit-ALU ist voll funktionsfähig und besitzt durch die Definition des Befehlssatzes in einer Header-Datei ein komfortables Anwendungs-Interface. Mit der Implementierung der Befehle in die MP3-Decoder Funktionen konnte die geforderte Echtzeitfähigkeit erreicht werden (siehe Kapitel 6.2.3).

[1] Diese Nummer bestimmt den Opcode-Index der ALU-Befehle und wird im SOPC-Builder bei der Integration angegeben. Die zugehörige Konstante (ALU64_CI_N) ist in der ALU-Header-Datei definiert.

4.4 Opencores.org IPs

Die Internet-Recherche nach frei verfügbaren Cores (für die fehlenden Funktionen) führte zur *Opencores.org* Entwicklergemeinde. Diese bietet über ihre Homepage (www.opencores.org) jedem registrierten Benutzer den Zugang zu unterschiedlichen IPs an. Die vorhandenen Projekte sind nach unterschiedlichen Kategorien wie Microprocessor, Communication-Controller usw. gruppiert.

Die angebotenen Komponenten haben in der Regel ein Wishbone-Bus-Interface und benötigen zur Verwendung in einem Altera-System einen „Bus-Wrapper“ (siehe 2.3.4.2).

Für die angepassten Cores wird zum Zweck der Strukturierung jeweils ein eigenes *Quartus II* Projekt erstellt (siehe Anhang D.1). Die „Bus-Wrapper-Module“ sind zur Dokumentation in den Anhang H.4 eingefügt. Bevor die Komponenten in das System integriert werden können, ist die Erstellung einer SOPC-Beschreibungsdatei erforderlich (siehe Kapitel 2.3.4.2).

Im Folgenden werden die verwendeten Cores beschrieben:

Der **I2C-Master** wird für die Anbindung des WM8731 I2C-Konfigurations-Interface und des auf dem Board befindlichen EEPROMs verwendet (siehe Abbildung 29). Bei der System-Integration stellte sich heraus, dass das I2C-Slave-Clock-Signal (SCLK) nicht der erwarteten Signalform entspricht. Dies konnte auf einen fehlenden Pull-Up-Widerstand zwischen dem FPGA und dem MAX II CPLD[1] zurückgeführt werden. Ein Versuch, die FPGA Pin-Konfiguration so zu ändern, dass ein interner Pull-Up-Widerstand bei dem Ausgangs-Port verwendet wird, scheiterte. Gelöst wird das Problem durch eine Modifikation des Bus-Wrappers, sodass anstatt des Tristate-Ausgangs ein normaler High-Pegel erzeugt wird (siehe H.4.3)

Der eingesetzte **PS/2-Core** wird für die Anbindung der Tastatur benötigt. Hier ist anzumerken, dass der Core-Quelltext in VHDL geschrieben ist und der Bus-Wrapper in Verilog. Dies zeigt die Interoparabilität beider Sprachen und die Stärken der Quartus II Software.

Der **SPI-Master** IP-Core der Entwicklergemeinde wurde zu Beginn der Entwicklung einem Geschwindigkeitstest unterzogen. Der Test ergab eine unzureichende Performance für die geforderte Anwendung (siehe Abbildung 28) und wurde daraufhin durch eine Neuentwicklung ersetzt (siehe Kapitel 4.1).

[1] Der MAX II CPLD wird auf dem Entwicklungskit als Bus-Multiplexer und Pegel-Transformator zwischen FPGA und einigen Peripherie-Bausteinen verwendet .

5 Systementwicklung

Die Hardware-Plattform besteht aus mehreren Komponenten (IP-Cores), die mit dem *SOPC-Builder* zu einem Gesamtsystem generiert werden (siehe Kapitel 2.3.4.1).

5.1 Konzept

Der konzeptionelle Systemaufbau mit der Gliederung in Module ist in der folgenden Abbildung dargestellt: Der eckig umrandete, mittlere Teil symbolisiert die Grenzen des FPGA. Die Komponenten, die ober- und unterhalb angeordnet sind, zeigen die Peripheriebausteine, die über EA-Pins mit dem FPGA verbunden sind. Die weißen Pfeile und der waagrechte Balken symbolisieren das Bussystem. Zur besseren Übersichtlichkeit sind die Komponenten nach Kategorien gruppiert.

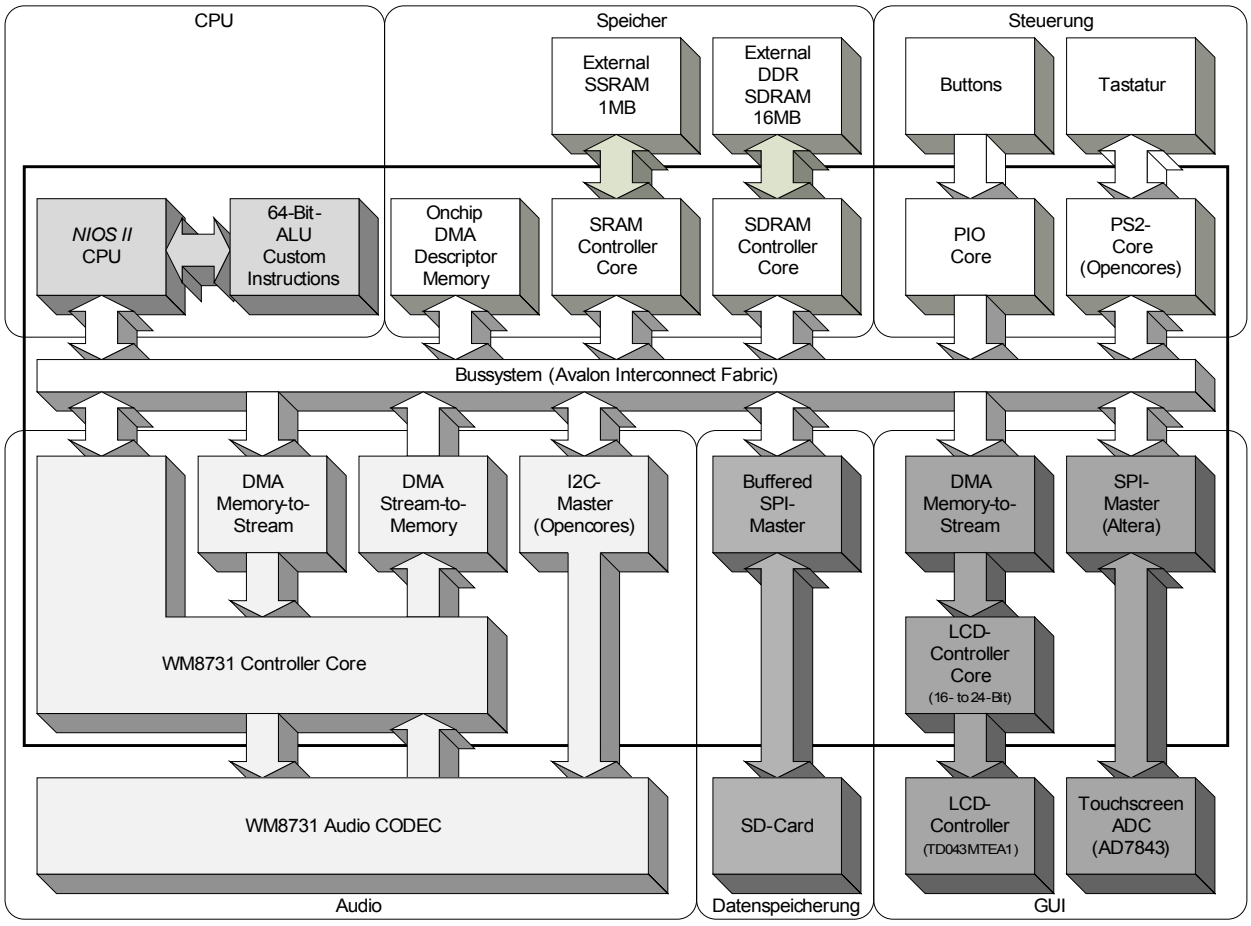


Abbildung 43: Hardwareorientierte Darstellung des Systemaufbaus

5.2 Implementierung

5.2.1 Projektstruktur

Zu Beginn der Implementierung des in der Abbildung 43 dargestellten Systems wird eine Projektstruktur erzeugt (siehe Abbildung 44). Diese besteht aus dem *Quartus II* **Audio-Plattform-Projekt**, das in weitere Projektdateien aufgeteilt ist. Dies ist das ***SOPC-Builder*-Projekt**, das die einzelnen Komponenten über Busverbindungen zu einem Gesamtsystem vereint (siehe 5.2.2) schließlich das **Top-Modul**, das das generierte **SOPC-Modul** und weitere Hilfsmodule wie den **Touchpanel-Clock-Multiplexer** und das **Audio-Clock-Join-Modul** verbindet. Die **Projekteinstellungen** sowie die **Clock-Constraints** werden für den Kompilierungsprozess benötigt. Diese Art der Strukturierung wird größtenteils durch die Verwendung der *Quartus II* Software vorgegeben.

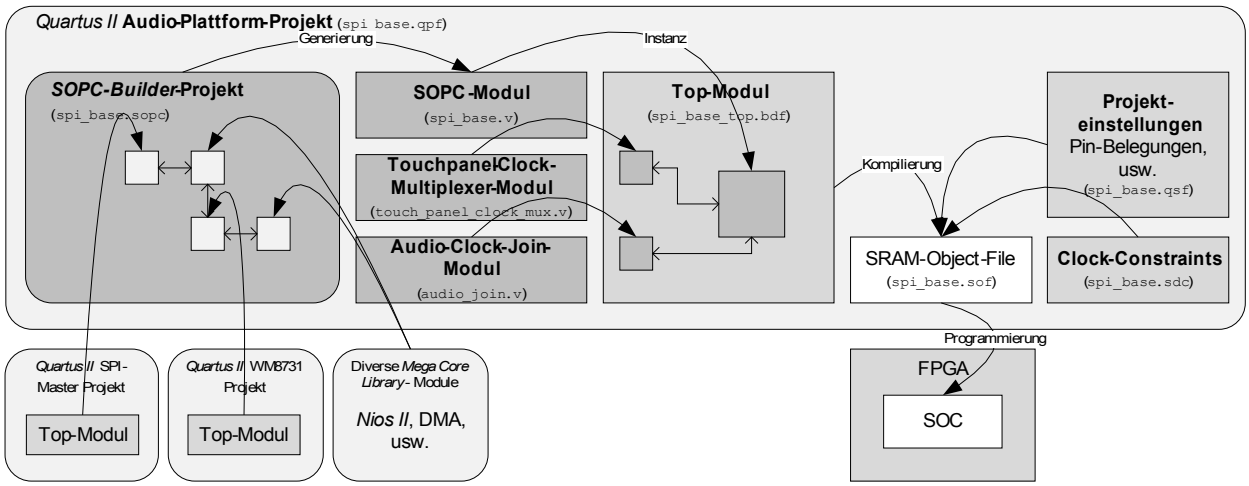


Abbildung 44: Audio-Plattform Projektstruktur

Das *Quartus II* **Audio-Plattform-Projekt** befindet sich im `spi_base` Verzeichnis (siehe D.1).

Die **Projekt-Einstellungen** sind in der `spi_base.qsf`[1] Datei definiert und enthalten die Pin-Zuweisungen sowie weitere Konfigurationseinstellungen (siehe Anhang J.2). Die Datei lässt sich mit einem Texteditor oder komfortabel über den *Quartus II* Projekt-Einstellungs-Dialog bearbeiten .

Die **Clock-Constraints** werden innerhalb der `spi_base.sdc`[2] Datei definiert (siehe Anhang J.1). Diese werden für die Timing-Analyse benötigt, die während der Systemsynthese (Kompilierung) in *Quartus II* durchgeführt wird. In der Datei werden die nicht synchronen Takte (siehe Kapitel 5.2.2.1) mit False-Path-Constraints[3] markiert.

---

1 Quartus II Settings File

2 Synopsys Design Constraints File

3 False-Path-Constraints werden verwendet, um zwei voneinander unabhängige Taktsignale für die Timing-Analyse zu markieren. Dadurch werden die sonst entstehenden False-Positives bei der Berechnung von Register-Setup- bzw. Hold-Violations vermieden.

Der **Touchpanel-Clock-Multiplexer** ist ein einfaches Verilog-Modul (keine SOPC-Komponente), um zwei Taktquellen zu multiplexen. Dieser wird verwendet, da sich der LCD-Controller und der Touchpanel-ADC das Slave-Clock-Signal (SCLK) auf dem Entwicklungskit teilen. So werden die beiden SCLKs, die von unterschiedlichen Cores generiert werden, auf übergeordneter Ebene gemultiplext (siehe H.4.5).

Das **Audio-Clock-Join-Modul** dient zur Zusammenfassung von zwei Einzelsignalen zu einem Bussignal.

5.2.2 SOPC-Builder-Projekt

Das *SOPC-Builder*-Projekt `spi_base.sopc` enthält alle Informationen zur Generierung des SOC. Diese umfassen die verwendeten Komponenten mit Adressvergabe, die Bus- und IRQ-Verbindungen sowie die Aufteilung in unterschiedliche Takt-Zonen.

5.2.2.1 Takt-Zonen

Das System wird in drei Takt-Zonen (Clock-Domains) aufgeteilt: Die 100-MHz **CPU-Clock-Domain** wird für den Prozessor, das SRAM und den Flashspeicher vorgesehen, damit diese mit einer möglichst hohen Takt-Frequenz Daten untereinander austauschen können. Für die Peripherie-Controller ist ein langsamerer Takt ausreichend. Hierfür wird die 60-MHz **Peripheral-Clock-Domain** erstellt. Die an das SDRAM-angebundenen Komponenten werden in die 65,5-MHz **SDRAM-Clock-Domain** integriert. Dieser Takt wird durch die Verwendung einer 133-MHz SDRAM-Anbindung vorgegeben. Zur Erzeugung der Audio-Referenz-Takte wird ein zusätzliches PLL integriert (siehe Kapitel 4.2.3.8). Die folgende Abbildung zeigt den erläuterten Aufbau, zusätzlich sind die in den Zonen enthaltenen Module mit den im *SOPC-Builder* vergebenen Instanznamen dargestellt:

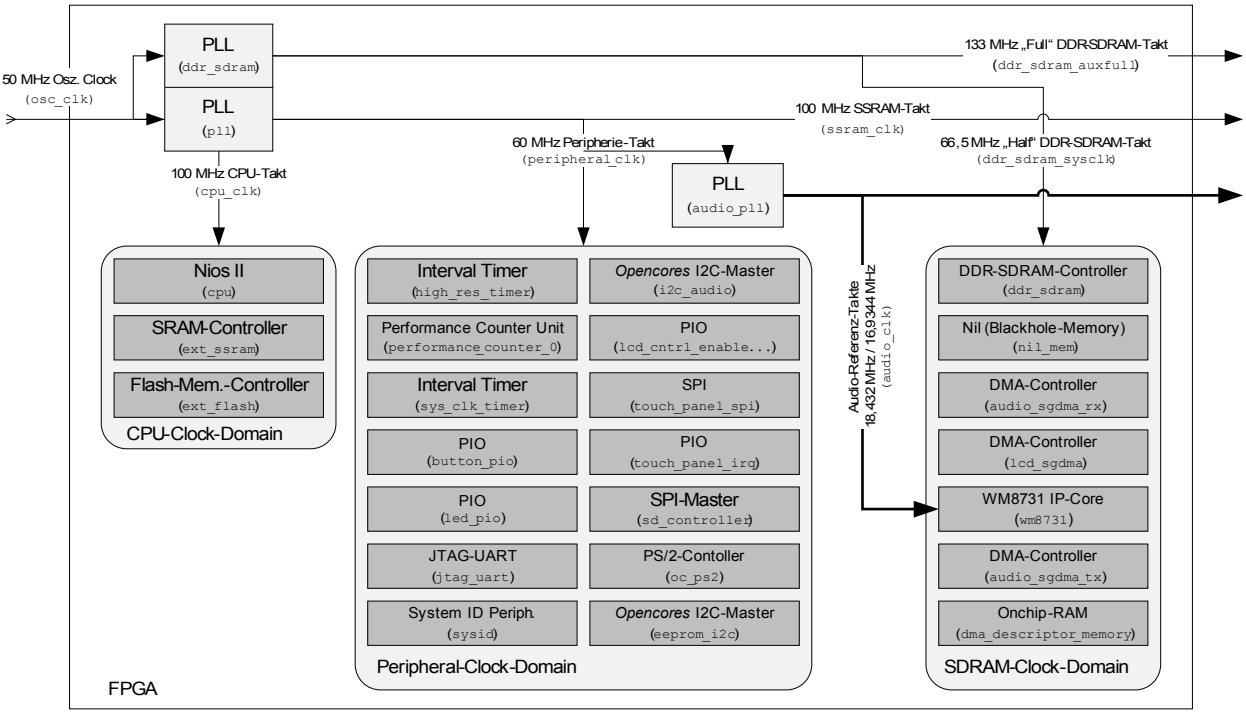


Abbildung 45: System Clock-Domains

5.2.2.2 Systemkomponenten (IP-Cores) mit Avalon-Busverbindung

Die folgende Abbildung zeigt die Verbindung der Komponenten über den *Avalon*-Bus: Zu erkennen ist die zentrale Rolle der CPU (A), die mit den anderen Modulen über Bridges (C, D, G, W) verbunden ist. Aus Gründen der Übersichtlichkeit sind die in der LCD-Chain enthaltenen Format-Adapter (3, 4, 5) nicht dargestellt. An den eingezeichneten DMAs ist der Übergang von Speicher-Interface zu Streaming-Interface (*Avalon-MM →Avalon-ST*) zu erkennen.

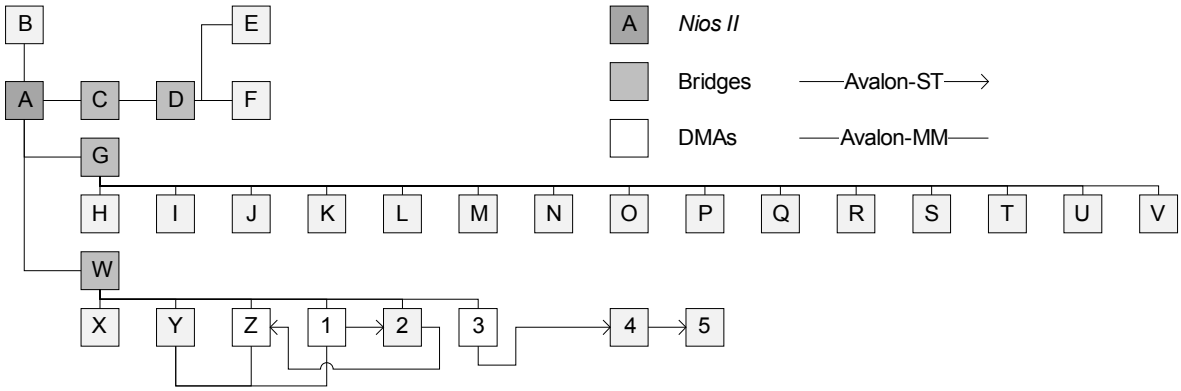


Abbildung 46: Systemkomponenten mit Avalon-Busverbindung

Das System wird aus den in der folgenden Tabelle aufgeführten Komponenten zusammengesetzt:

	Beschreibung	Instanzname	IRQ		Beschreibung	Instanzname	IRQ
A	Nios II	`cpu`		**P**	Opencores I2C-Master	`i2c_audio`	6
B	64-Bit-ALU	`cpu_mult64_inst`		**Q**	PIO	`lcd_cntrl_enable, *_scl, *_sda`	
C	Pipeline-Bridge	`cpu2ram_bridge`		**R**	SPI	`touch_panel_spi`	11
D	Tristate-Bridge	`tristate_bridge`		**S**	PIO	`touch_panel_irq`	10
E	Cypress SSRAM	`ext_ssram`		**T**	SPI-Master IP-Core	`sd_controller`	
F	Flash Memory	`ext_flash`		**U**	Opencores PS/2	`oc_ps2`	8
G	Clock-Crossing-Bridge	`cpu2peripheral`		**V**	Opencores I2C-Master	`eeprom_i2c`	12
H	Interval Timer	`high_res_timer`	1	**X**	DDR-SDRAM Controller	`ddr_sdram`	
I	Performance Counter	`performance_counter_0`		**Y**	On-Chip Memory	`dma_descriptor_memory`	
J	Interval Timer	`sys_clk_timer`	2	**Z**	Scatter-Gather DMA-Controller	`audio_sgdma_rx`	4
K	Phase-Locked-Loop (PLL)	`pll`		**1**	Scatter-Gather DMA	`audio_sgdma_tx`	3
L	PIO	`button_pio`	9	**2**	WM8731 IP-Core	`wm8731`	
M	PIO	`led_pio`		**3**	Scatter-Gather DMA	`lcd_sgdma`	5
N	JTAG UART	`jtag_uart`	7	**4**	LCD-16-24-Converter	`lcd_converter`	
O	System ID Peripheral	`sysid`		**5**	Video Sync Generator	`altera_avalon_video_sync_generator_inst`	

Tabelle 9: Systemkomponenten

In der Tabelle sind zusätzlich die zugeordnete IRQ-Nummer und der Instanzname aufgeführt. Der Name wird bei der Generierung der `system.h` Datei (siehe Kapitel 2.4.4), in der sich die Konstanten für die Softwareentwicklung befinden, verwendet.

Der ***Nios II*** ist ein 32-Bit RISC-Prozessor und wird von Altera mit der *MegaCore IP-Library* zur Verfügung gestellt . Dieser kann bei der Erstellung im System mit unterschiedlichen Einstellungen konfiguriert werden:

Einstellung	**Wert**
Nios II Core	Nios II/fast
Hardware Multiply	Yes
Hardware Divide	Yes
Instruction Cache	4 KB ohne Burst
Data Cache	2 KB
Data Cache Line Size	16 Bytes
Debug Level	1

Tabelle 10: Nios II Konfiguration

Die Einstellungen sind so gewählt, dass der Prozessor mit einer möglichst hohen Performance arbeitet. Der dabei erhöhte Ressourcenverbrauch (Logik Elemente und Register) stellt hier keinen Nachteil dar.

Die **Phase-Locked-Loop** (PLL) wird innerhalb des Systems mehrmals verwendet (siehe Abbildung 45). Eine PLL erzeugt aus dem extern zugeführten 50-MHz Oszillatorsignal die Systemtakte für CPU, SRAM, FLASH und Peripherie. Eine weitere PLL generiert die Audio-Referenz-Takte (siehe Kapitel 4.2.3.8). Die dritte PLL wird automatisch durch die Verwendung des DDR-SDRAM-Controller-Cores in das System integriert.

Der **LCD-16-24-Converter** ist ein IP-Core, der eine 16-Bit-Pixel zu 24-Bit-Pixel-Konvertierung durchführt und als *Avalon-ST*-Komponente implementiert ist. Der Quelltext ist im Anhang H.4.4 abgedruckt.

Der **Scatter-Gather-DMA-Controller** wird im System zur Verbindung des Hauptspeichers (DDR-SDRAM) mit den Streaming-Schnittstellen (*Avalon-ST*) des LCD-Controllers und des WM8731 IP-Cores verwendet. An die eingesetzten DMAs wird ein On-Chip Speicher (Y) angebunden, in dem über „Descriptor-Chains“ die DMA Steuerung erfolgt .

Die **PIO**-Komponete ist ein Altera-*MegaCore* zur direkten Ansteuerung von FPGA-Pins und wird hier für die Anbindung der Taster und LEDs verwendet.

Die **System ID** Komponente dient zur Identifikation des erstellten Systems. So wird vermieden, dass ein fertiges Softwareprojekt auf ein veraltetes bzw. nicht passendes System übertragen wird.

Auf die neu entwickelten Komponenten wird hier nicht weiter eingegangen, da die Systemintegration in den jeweiligen Kapiteln erläutert wird.

Der **Interval-Timer** wird zweimal mit unterschiedlichen Zeitauflösungen (10µs und 10ms) erstellt und dient als Zeitgeber im System.

Da bei der Verwendung unterschiedlicher Takt-Zonen verschieden getaktete Komponenten miteinander kommunizieren müssen (z.B. beim Zugriff der 100-MHz CPU auf die 60-MHz Peripherie, siehe Abbildung 45) ist eine Synchronisationslogik erforderlich. Es gibt folgende Möglichkeiten zur Lösung: Es wird dem SOPC-Builder überlassen, „unsichtbare“ Clock-Domain-Adapter zwischen die Master-Slave-Verbindungen zu integrieren oder es werden manuell **Clock-Crossing-Bridges** (CCB) zur Entkopplung der Komponenten hinzugefügt. Hier wird die manuelle Variante bevorzugt, da diese durch die Verwendung von FIFOs einen höheren Datendurchsatz ermöglicht . Das System mit den integrierten CCBs ist in Abbildung 47 zu sehen. Bei der Erstellung der Instanzen im *SOPC-Builder* wurden die Standardwerte für die FIFOs übernommen.

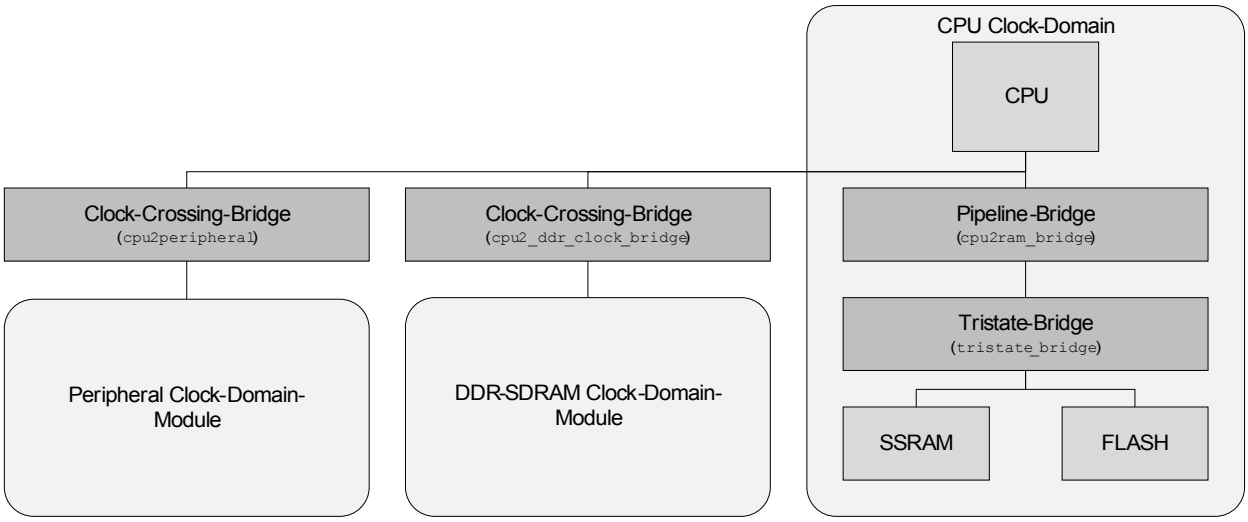


Abbildung 47: System-Bridges (Clock-Crossing-, Pipeline- und Tristate-Bridges)

Lange Signalwege innerhalb eines Systems führen zur Verringerung der maximal möglichen Systemfrequenz (Fmax). Zur Reduktion des Signalwegs pro Zyklus wird die **Pipeline-Bridge** verwendet. Diese integriert eine Registerstufe zwischen die angeschlossenen Komponenten. Hierdurch werden die Signallaufzeiten reduziert und die Fmax erhöht. Eine Erhöhung der Zyklenanzahl wird dabei akzeptiert.

Die **Tristate-Brige** dient zum Multiplexen von Signalen und wird im Zusammenhang mit dem SSRAM und dem Flash-Speicher eingesetzt (siehe Abbildung 47). Diese teilen sich den Daten- und Adressbus. Hierdurch werden EA-Pins gespart.

Die Adressvergabe für die *Avalon-MM*-Komponenten ist trivial und erfolgt bei der Konfiguration der Komponente im *SOPC-Builder*.

5.2.3 System-Generierung und Synthese (Kompilierung)

Das so erstellte System kann im *SOPC-Builder* bei kollisionsfreier Adress- und Interrupt-Vergabe generiert werden. Anschließend wird *Quartus II* ein grafisches Symbol für das erstellte System (Verilog-Datei) erzeugt bzw. aktualisiert. Das System-Schaltungssymbol lässt sich in das

zuvor angelegte grafische Top-Modul einfügen und mit den EA-Pins verbinden. Den Abschluss der Erstellungsphase bildet die Synthese des Systems. Die dabei entstehende SOF-Datei kann in das FPGA programmiert und anschließend verifiziert werden.

5.2.4 FPGA-Ressourcen-Verbrauch

Folgende Tabelle enthält den FPGA-Ressourcen-Verbrauch des erstellten Systems:

Ressource	Einheiten
Total logic elements	20,498 / 24,624 (83 %)
Total combinational functions	16,269 / 24,624 (66 %)
Dedicated logic registers	12,378 / 24,624 (50 %)
Total registers	12612
Total pins	164 / 216 (76 %)
Total memory bits	145.222 / 608.256 (24 %)
Embedded Multiplier 9-bit elements	12 / 132 (9 %)
Total PLLs	3 / 4 (75 %)

Tabelle 11: FPGA-Ressourcen-Verbrauch

5.3 Ergebnis

Der Systemerstellungsprozess wird durch die von Altera zur Verfügung gestellten Werkzeuge sehr vereinfacht. Sobald ein Basissystem vorhanden ist, können Änderungen durch das Hinzufügen neuer Komponenten einfach durchgeführt werden (Konfigurierbarkeit!). Hierbei ist auf den Ressourcenverbrauch zu achten. Das entstandene System benötigt bereits ca. 83% der verfügbaren Logik-Elemente. Um z.B. einen Netzwerk-Controller mit DMA-Anbindung zu integrieren, muss der Verbrauch zuvor durch Optimierung bzw. Entfernung nicht benutzter Komponenten verringert werden.

Ein für das Entwicklungskit spezielles Problem soll nicht unerwähnt bleiben: So ist vom Hersteller die Verbindung der LEDs an das FPGA in derselben EA-Bank durchgeführt worden, in der auch das DDR-SDRAM angeschlossen ist. Es scheint, dass durch die Verwendung des DDR-SDRAM-Cores in dem System sehr viele Pin-nahe Logik-Ressourcen verbraucht werden und eine Parallel-Nutzung einiger LED-Pins aus diesem Grund nicht möglich ist.

6 Softwareentwicklung

6.1 Audio-Plattform-Softwarebibliothek

Der konzeptionelle Aufbau der Audio-Plattform-Softwarebibliothek mit den Treibern und APIs wird anhand der folgenden Abbildung erläutert:

In der Mitte ist die Anwendung abgebildet, die die Softwarebibliotheks-Funktionen verwendet. Diese kann mit den hardwareunabhängigen APIs für den Dateizugriff, dem grafischen Benutzerinterface und der Audio-API verbunden werden. Die PS/2- und Touchpanel-Treiber sind hardwareabhängig und werden ohne Abstraktion eingebunden. Die anderen Treiber werden von der Anwendung entkoppelt. Dies soll den Austausch der zugrunde liegenden Hardware (SOC, IP-Cores) erleichtern. Die mit „Device“ benannten Module sind zusätzlich in die *Hardware Abstraction Layer* (HAL) integriert. Diese ermöglicht den Zugriff über Standard-Bibliotheks-Funktionen (siehe Kapitel 2.4.3).

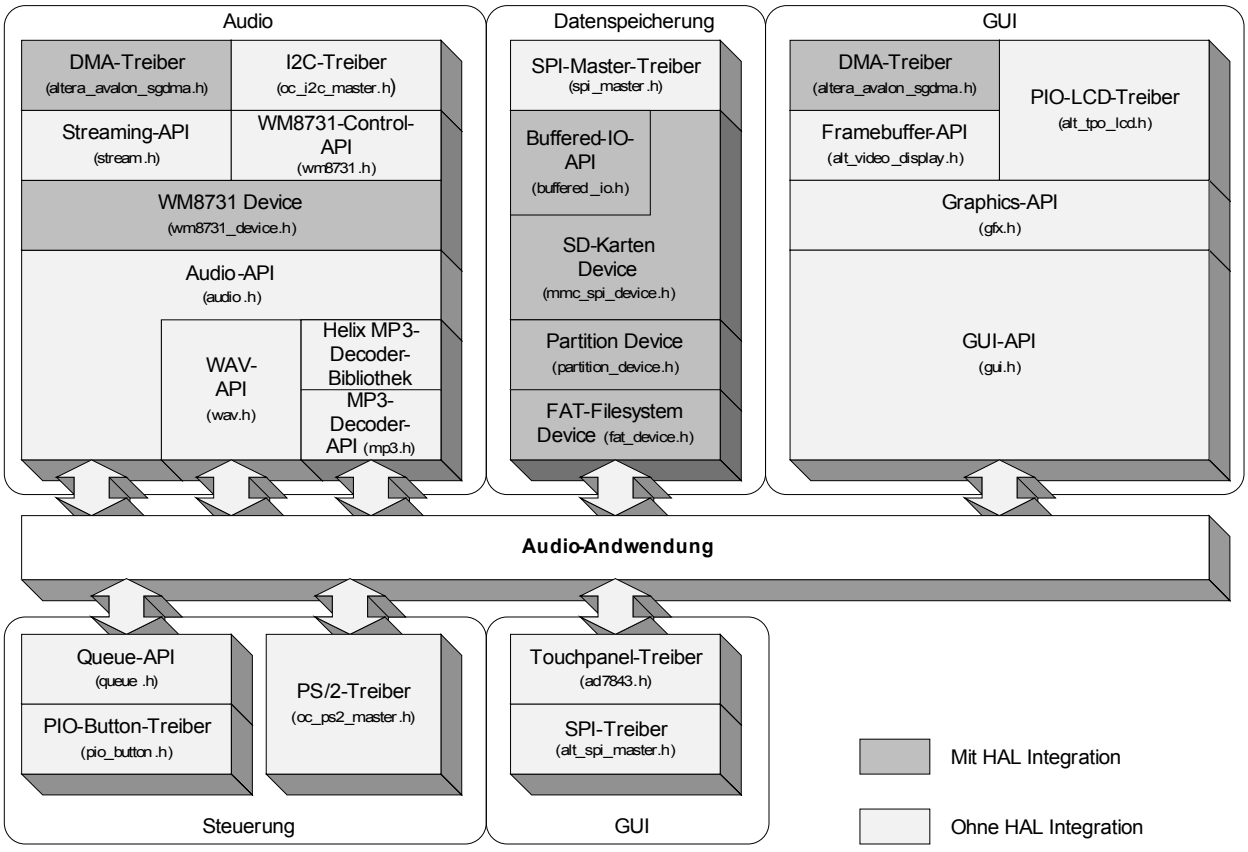


Abbildung 48: Schematischer Aufbau der Audio-Plattform-Softwarebibliothek mit Kennzeichnung der HAL-Devices

Die Bibliothek befindet sich im `nios2_audio_plattform` Projekt (siehe D.2). Im Folgenden werden die entwickelten Treiber und APIs beschrieben.

6.1.1 Datenspeicherung

6.1.1.1 SPI-Master-Treiber

Der SPI-Treiber wird in Kapitel 4.1.3.6 erläutert.

6.1.1.2 SD-Karten-Treiber

Der SD-Karten-Treiber basiert auf einer 8-Bit-*Atmel-AVR*-Implementierung, die vom Autor innerhalb der „LogFrog“ Datenlogger-Applikation erstellt wurde . Die Treiberdateien befinden sich im `device/mmc` Verzeichnis. Folgende Tabelle beschreibt den Inhalt der Treiberdateien:

Datei	**Beschreibung**
`mmc_card.h`	Definition der MMC/SD-Karten- und Protokoll spezifische Konstanten
`mmc_spi_device.h`	Treiber-Funktions-Prototypen
`mmc_spi.c`	Implementierung der MMC/SD-Karten Protokollfunktionen
`mmc_spi_device.c`	Implementierung der HAL Treiberfunktionen
`buffered_io.h`	Prototypen für die Block-Pufferung
`buffered_io.c`	Implementierung der Block-Pufferung

Tabelle 12: SD-Karten Treiber-Dateien

Im Unterschied zur ursprünglichen Implementierung wird hier die SPI-Schnittstelle besser gekapselt. Dies ermöglicht einen einfachen Austausch des zugrunde liegenden SPI-Treibers bei zukünftigen System-Änderungen. Zusätzlich wurde der SD-Treiber um die Multiple-Block-Read und Mutliple-Block-Write[1] SD-Protokoll-Funktionen ergänzt (siehe Anhang I.5.2) . Dies führte zu einer Lese- und Schreibgeschwindigkeits-Verbesserung, die durch eine Messung dokumentiert wird und in der Abbildung 28 dargestellt ist. Bei der getesteten 1-GB Hama SD-Karte ist eine Steigerung der Performance von 6979 kbit/s auf 7886 kib/s (ca. 13 Prozent) zu erkennen. Die Messung ist nicht repräsentativ, da keine Vergleichsmessungen (vorher/nachher) mit anderen Karten durchgeführt wurden. Es wird jedoch davon ausgegangen, dass sich die Verbesserung auch auf andere Karten auswirkt, da das Datenzugriffs-Prinzip verbessert wurde.

Der Treiber ist mit einer Block-Pufferung ausgestattet, die SD-Karten-Blöcke bei einem wiederholten Zugriff im RAM zwischenspeichert. Dies optimiert Zugriffe mit kleiner Schreib-Lesegröße. Beim Daten-Streaming[2] mit Schreib-Lesegrößen ab 512 Bytes wird der Pufferungsmechanismus bei einem vollständigen Blockzugriff umgangen. Die Anzahl der zu puffernden Blöcke ist konfigurierbar. Tendenziell gilt: Je mehr Puffer, umso schneller ist der Treiber. Hier ist zwischen Performancegewinn und Speicherverbrauch abzuwägen.

Hierfür wurde ein Testprogramm erstellt, das im Anhang G.2 zu finden ist.

---

[1] Die Multiple-Block Funktionen erlauben das Lesen- bzw. Schreiben mehrerer Blöcke sequentiell hintereinander. Hierdurch kann die SD-Karte, während Daten übertragen werden, bereits Speicher für den nächsten Block löschen bzw. reservieren.

[2] Kontinuierliches Schreiben bzw. Lesen

6.1.1.3 Partitions-Device

Das Partitions-Device ist in dem `device/partition` Verzeichnis definiert und wurde entwickelt, um den Zugriff auf die Dateisystem-Partitionen zu abstrahieren. Die `partition_device.h` Datei enthält die Device-Struktur und den Prototypen für die Initialisierungsfunktion. Durch die Integration des Treibers in die HAL kann eine Anwendung nach Initialisierung des Treibers auf die vorhandenen Partitionen mit den Standard EA-Funktionen zugreifen. Die Zuweisung der `parentDevice` Variable in der Device-Struktur ermöglicht die Angabe des Speichermediums, auf das der Partitions-Treiber nach der Initialisierung zugreifen soll.

6.1.1.4 FAT-Filesystem

Die Implementierung des FAT-Filesystems basiert auf der vom Autor innerhalb des Studiums angefertigten Datenlogger Applikation . Die Treiberdateien befinden sich im `device/fat` Verzeichnis. Folgende Tabelle beschreibt den Inhalt der wichtigsten Treiberdateien:

Datei	**Beschreibung**
`fat.h`	Definition der Konstanten und FAT-spezifische Funktions-Prototypen
`fat_device.h`	Treiber-Funktions-Prototypen
`fat_device.c`	Implementierung der HAL Treiberfunktionen
`create.c`	Implementierung der create-, unlink- und rename-Funktion
`dir.c`	Verzeichniszugriffs-Funktionen
`mount.c`	Implementierung der mount-Funktion
`open.c`	Implementierung der open-Funktion
`readwrite.c`	Implementierung der read- und write-Funktionen

Tabelle 13: FAT-Filesystem Treiber-Dateien

Die Integration des Treibers aus der *Atmel-AVR*-Programmbasis erfolgte ohne große Anpassungen. Ein Programmfehler, der durch die *Nios II* 32-Bit-Architektur hervorgerufen wird, konnte gelöst werden (es wurde auf 16-Bit Werte an ungeraden Adressen zugegriffen, das führte zu einem Bus-Fehler, die fehlerhafte `unicode_utf16to8`[1] Funktion wurde umgeschrieben).

6.1.1.5 Schreib-Lesegeschwindigkeitsmessung

In der Testphase wurden mehrere unterschiedliche SD-Karten auf ihre Schreib- und Lesegeschwindigkeit mit dem FAT-Filesystem getestet. Für die grafische Darstellung (siehe Abbildung 49 und Abbildung 50) wurden jeweils vier Kennlinien von unterschiedlichen Kartengrößen ausgewählt. Die zugehörigen Messwerte sind in der Tabelle 37 im F aufgeführt. Die Messung wurde mit einer 8 MB großen Datei und mit einer Pufferung von fünf Blöcken durchgeführt. Das Messergebnis, die Datei-Schreib- bzw. -Lesezeit, wurde gemittelt und in kbit/s umgerechnet. Das Messprogramm ist im Anhang G.4 abgedruckt.

[1] Die unicode_utf16to8 Funktion ist innerhalb der charset/unicode.c Datei implementiert.

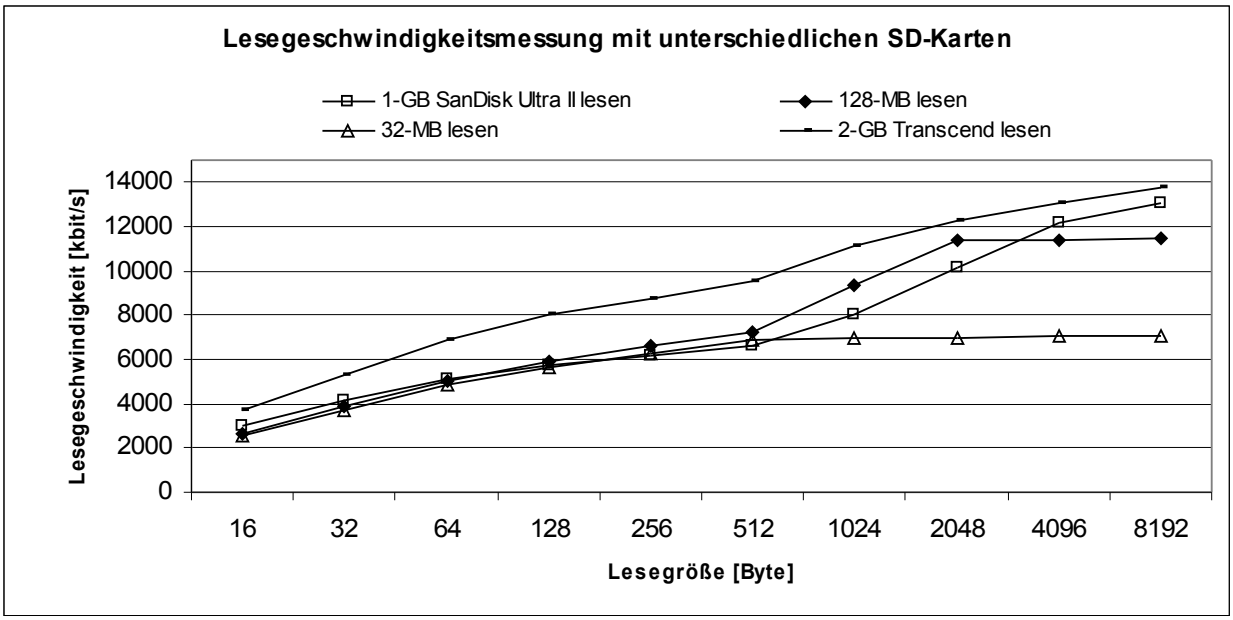


Abbildung 49: Lesegeschwindigkeitsmessung mit unterschiedlichen SD-Karten

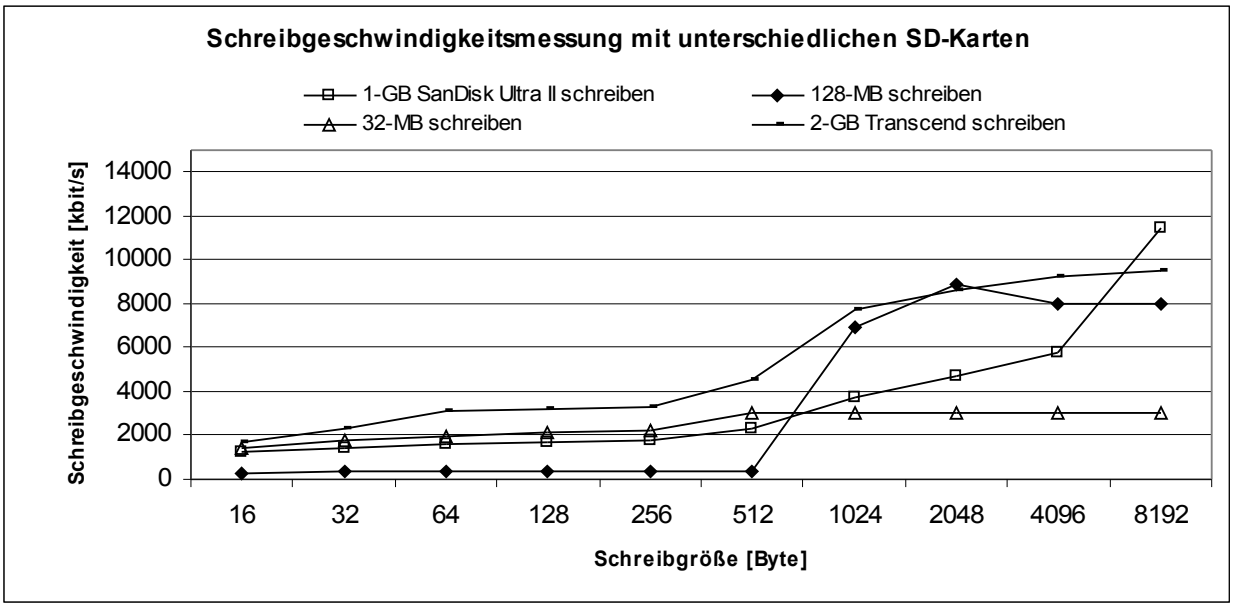


Abbildung 50: Schreibgeschwindigkeitsmessung mit unterschiedlichen SD-Karten

6.1.1.6 Messung der Rechenzeitverteilung

Es wird eine Messung zur Untersuchung der Rechenzeitverteilung beim Zugriff auf die SD-Karte durchgeführt. Die beiden folgenden Abbildungen zeigen das Ergebnis dieser Messung. Der obere Teil des Balkens im Diagramm repräsentiert die Zeit, die der FAT-Treiber für das Datei-Handling[1] benötigt. Der untere Balken stellt die Datenübertragungs-Zeit dar, die zwischen SD-

[1] Das Datei-Handling beinhaltet folgende Funktionen: a. beim Lesen: Suche des nächsten Clusters in der FAT, b. Beim Schreiben: Suche des nächsten freien Clusters und dessen Allokation. Die Datei-Zeiger Berechnung, um den Clusterübergang zu prüfen und die Daten-Zwischenpufferung, erfolgt bei beiden Operationen.

Karte und Hauptspeicher verbraucht wird. Die zugehörigen Messwerte sind in der Tabelle 38 im F abgebildet. Das Messprogramm ist im Anhang G.4 abgedruckt.

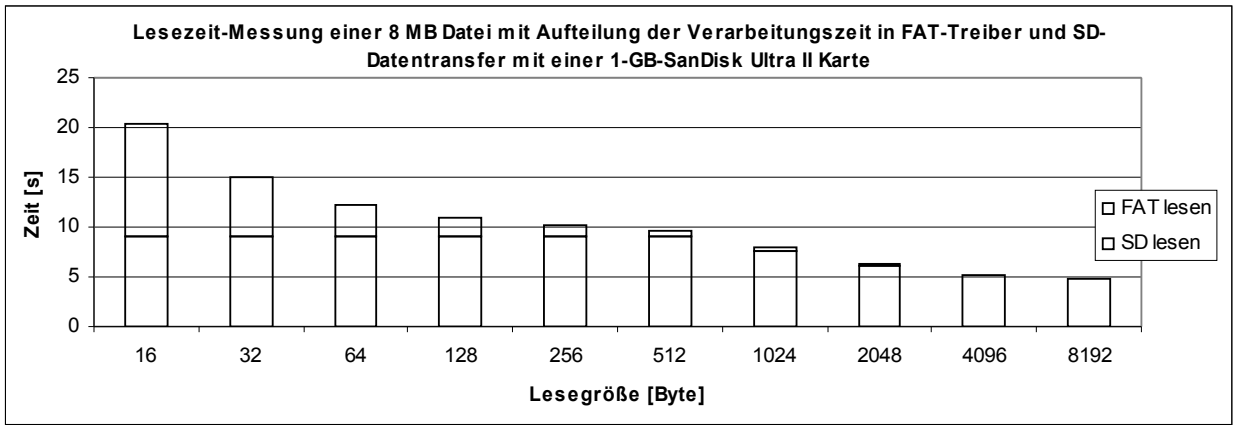


Abbildung 51: Rechenzeitverteilung der Lesezeit

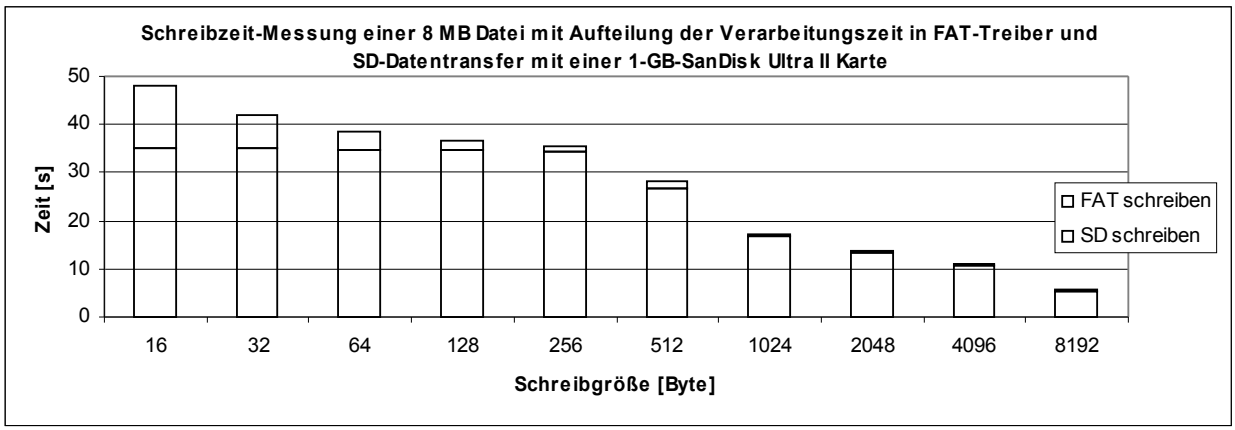


Abbildung 52: Rechenzeitverteilung der Schreibzeit

Anhand der Abbildungen ist zu erkennen, dass der FAT-Treiber bei Schreib-Lesegrößen im Bereich von 16 bis 512 Byte Rechenzeitrelevant ist. Ab einer Größe von 512 Byte verringert sich die SD-Karten-Datenübertragungszeit sichtbar Dies ist auf den Multi-Block-Read und -Write-Mode zurückzuführen, der bei Übertragungen mit mehr als einem Block (512 Bytes) verwendet wird (siehe Kapitel 6.1.1.2). Mit steigender Zugriffsgröße verringert sich die Aufruf-Frequenz der FAT-Lese- bzw. -Schreib-Funktion und somit die Rechenzeit für das Datei-Handling.

6.1.2 Audio

6.1.2.1 I2C-Treiber

Der Treiber implementiert die Steuerung für den Opencore I2C-Core. Dieser basiert auf einer Linux-Version von P. Korsgaard[1], die in dieser Arbeit für die Altera-Infrastruktur reimplementiert wurde. Die Treiber-Dateien befinden sich im `device/oc_i2c` Verzeichnis. In der

[1] http://www.linuxhq.com/kernel/v2.6/18/drivers/i2c/busses/i2c-ocores.c

`oc_i2c_master_regs.h` Datei werden die Core-Register und Status-Flags definiert. Die Device-Struktur und die Funktions-Prototypen werden in der `oc_i2c_master.h` Datei definiert.

Funktion	Beschreibung
`oc_i2c_init_device`	Initialisiert den Treiber und registriert in der HAL
`oc_i2c_exit_device`	Entfernt den Treiber

Tabelle 14: Opencore I2C-Treiber

6.1.2.2 WM8731 Device-Treiber

Der Treiber für den Audio-CODEC wird in Kapitel 4.2.3.9 beschrieben.

6.1.2.3 Streaming-API

Die Streaming-API ist eine speziell entwickelte Ring-Puffer-Implementierung, die direkt mit einer DMA für den Datentransfer verbunden ist (siehe Abbildung 53). Bei der Initialisierung wird die Transfergröße und die Gesamt-Puffergröße spezifiziert. Einer Anwendung werden Schreib- bzw. Lesefunktionen für die Datenübertragung mit den Ring-Puffern zur Verfügung gestellt. Die angeschlossene DMA übernimmt den Transfer der Daten zwischen Ring-Puffer und *Avalon-ST*-Sink oder -Source. In der vorliegenden Arbeit wird diese API verwendet, um die Audioschnittstelle des WM8731 IP-Cores mit der Anwendung zu verbinden.

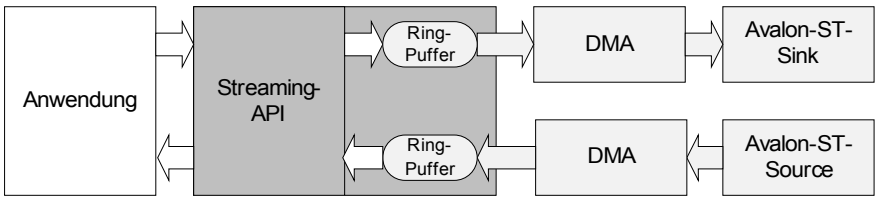


Abbildung 53: Streaming-API

Die Treiberdateien sind im `device/stream` Verzeichnis zu finden. Die `stream.h` Datei definiert die Treiberfunktions-Prototypen und die Device-Struktur. Die `stream.c` Datei implementiert die API-Funktionen.

6.1.2.4 Audio-API

Die Audio-API stellte eine generalisierte Audio-Schnittstelle für die Anwendungsentwicklung zur Verfügung. Damit soll eine Kapselung der zugrunde liegenden Treiberebene geschaffen werden. So greifen die MP3- und WAV-APIs nicht direkt auf die WM8731-IP-Core Funktionen zu. Dies soll zukünftige Erweiterungen (z.B. die Integration eines anderen CODECs) erleichtern. Die Treiber-Dateien sind im `device/audio` Verzeichnis zu finden. Die `audio.h` Header-Datei definiert die folgenden öffentlichen Funktionen:

Funktion	Beschreibung
`audio_set_sampling_rate`	Setzt die Samplerate
`audio_get_sampling_rate`	Liest die Samplerate aus

audio_clear_stream	Löscht den Audio-Ausgabedatenstrom
audio_stream_to_file	Liest Daten aus dem Audio-Eingangsstrom in eine Datei
audio_stream_from_file	Schreibt Daten aus einer Datei in den Audio-Ausgabestrom
audio_set_line_out_volume	Stellt die Line-Out Lautstärke ein

Tabelle 15: Audio-API-Funktionen

6.1.2.5 WAV-API

Die WAV-API wurde entwickelt, um den Zugriff auf WAV-Dateien zu vereinfachen. So kann über die entstandene API eine WAV-Datei abgespielt und aufgenommen werden. Die API-Dateien sind im audio/wav Verzeichnis zu finden. Die wav.h Header-Datei definiert die folgenden Funktionen:

Funktion	**Beschreibung**
wav_init_playback	Initialisiert die Datei-Wiedergabe
wav_play_next	Spielt die nächsten n-Bytes ab
wav_exit_playback	Beendet die Wiedergabe
wav_get_seconds_left	Gibt einen Sekundenzähler für die verbleibende Spielzeit zurück
wav_init_recording	Initialisiert die Datei-Aufnahme
wav_record_next	Nimmt die nächsten n-Bytes auf
wav_exit_recording	Beendet die Aufnahme

Tabelle 16: WAV-API Funktionen

6.1.2.6 MP3-Decoder-API

Innerhalb der Audio-Plattform wird die Helix-MP3-Decoder-API (siehe Kapitel 6.2.1.4) in folgende Funktionen gekapselt. Die API-Dateien sind im audio/mp3 Verzeichnis zu finden.

Funktion	**Beschreibung**
mp3_init_decode	Initialisiert den Decoders
mp3_exit_decode	Gibt den Decoderspeicher frei
mp3_init_playback	Initialisiert die Playback-Struktur, Öffnet die abzuspielende Datei
mp3_exit_playback	Schließt die Datei, Leert den Audio-Puffer
mp3_play_next	Liest MP3-Daten aus der Datei, decodiert MP3-Daten und füllt den Audio-Puffer mit Sample-Daten
mp3_get_seconds_left	Gibt die Anzahl der verbleibenden Sekunden zurück

Tabelle 17: Gekapselte MP3-Decode- API innerhalb der Audio-Plattform

6.1.3 Grafisches Benutzerinterface

6.1.3.1 PIO-LCD-Treiber

Der PIO-LCD-Treiber wird dazu verwendet, den LCD-Controller zu initialisieren. Die Treiber-Dateien sind im `device/alt_tpo_lcd` Verzeichnis zu finden. Der Quelltext stammt aus dem Picture-Viewer-Beispielprogramm, das dem Entwicklungssystem beiliegt. Der Treiber wird mit drei 1-Bit PIO IP-Cores verbunden, deren EA-Adressen bei der Initialisierung der Treiber-Struktur angegeben werden müssen.

Funktion	Beschreibung
`alt_tpo_lcd_read_config_register`	Liest LCD-Register
`alt_tpo_lcd_write_config_register`	Beschreibt LCD-Register
`alt_tpo_lcd_init`	Initialisiert LCD

Tabelle 18: PIO-LCD-Treiber API

6.1.3.2 Framebuffer-API

Die Framebuffer-API stellt die programmatische Schnittstelle zwischen Hauptspeicher und LCD-Display dar. In der Initialisierungsfunktion wird eine DMA gestartet, die die RGB Pixel-Daten kontinuierlich aus dem Speicher an einen Pixel-Konverter weiterreicht. Die `alt_video_display` Struktur definiert Variablen für die Bildschirmauflösung und Pixelbreite, die bei der Initialisierung angegeben werden können. Die API ist in dem `device/alt_video_display` Verzeichnis definiert und stammt aus dem Picture-Viewer-Beispielprogramm. Sie wurde um einen Frame-Counter (Bildzähler) erweitert, der es dem Prozessor ermöglicht einen Bildwechsel zu erkennen.

Funktion	Beschreibung
`alt_video_display_init`	Initialisiert Framebuffer und DMA
`alt_video_display_close`	Gibt Framebuffer-Speicher frei und stoppt DMA

Tabelle 19: Framebuffer-API

6.1.3.3 Grafik API

Die Grafik API stellt Funktionen zum Zeichnen von Linien, Rechtecken und Zeichenketten zur Verfügung. Es wird ein Rechteck-Clipping für die Linien-Funktionen unterstützt. Intern greift die API auf den Framebuffer zu, der sich im Hauptspeicher befindet und von der Framebuffer-API kontrolliert wird (siehe Kapitel 6.1.3.2). Die `gfx_context_t` Struktur, die jeder Zeichenfunktion übergeben wird, definiert ausgewählte Farbe, Schrift und Clip-Region. Zur Initialisierung der Kontext-Struktur wird ein Zeiger auf die `alt_video_display` Struktur benötigt. Die API ist im `gfx` Verzeichnis zu finden. Die folgende Tabelle enthält die wichtigsten Zeichenfunktionen. Diese unterstützen nur den 16-Bit Farbmodus (Hi-Color) vollständig. Andere Modi müssen bei Bedarf nachgerüstet werden. Die Implementierung unterstützt bisher nur eine Festpunktschrift.

Diese ist im `gfx/font` Unterordner zu finden und kann als Vorlage für Erweiterungen genutzt werden:

Funktion	**Beschreibung**
`gfx_draw_line`	Zeichnet eine Linie
`gfx_draw_string`	Zeichnet eine Zeichenkette
`gfx_fill_rect`	Füllt ein Rechteck
`gfx_set_pixel`	Setzt einen Pixel

Tabelle 20: Grafik API

6.1.3.4 Altera-SPI Treiber

Im System wird für die Anbindung des Touchpanel-AD-Wandlers der Altera SPI-MegaCore verwendet. Der entwickelte Treiber unterstützt Interrupt-gesteuerte Transaktionen. Diese bestehen aus einer Schreib- und Lese-Aktion, wobei für beide eine bestimmte Anzahl von Bytes definiert werden kann. Hierdurch kann die Kommunikation mit einem Peripherie-Baustein ohne Warteschleife des Prozessors durchgeführt werden. Die Treiber-Dateien sind im `device/alt_spi` Verzeichnis erstellt (siehe Anhang I.3).

Folgende Tabelle enthält die SPI-Treiber-Funktionen:

Funktion	**Beschreibung**
`alt_spi_init_device`	Initialisiert den Treiber, Interrupt Registrierung
`alt_spi_exit_device`	Interrupt Unregistrierung
`alt_spi_init_transaction`	Initialisiert Transaktions-Struktur
`alt_spi_start_transaction`	Startet eine Transaktion
`alt_spi_transaction_active`	Teste ob eine Transaktion beendet ist
`alt_spi_read`	Lese Daten
`alt_spi_write`	Schreibe Daten
`alt_spi_write_constant`	Schreibe Konstante
`alt_spi_cs_enable`	Chipselect
`alt_spi_cs_disable`	Chipdeselect

Tabelle 21: Altera-SPI Treiber Funktionen

6.1.3.5 Touchpanel-Treiber

Der Touchpanel-Treiber hat die Aufgabe, den Touchpanel-AD-Wandler über das SPI-Interface abzufragen und die ausgelesenen Werte in Form einer Message-Struktur mit x- und y-Koordinate in einer Queue[1] abzulegen. Die Queue kann von einem Anwendungsprogramm ausgelesen werden um die Daten zu verarbeiten. Der Treiber ist im `device/ad7843` Verzeichnis zu finden.

[1] Eine Queue (Warteschlange) ist ein Software basierter Daten-FIFO-Speicher zur zeitlichen Entkopplung von Datenerzeugung und Datenverarbeitung

Die ad7843.h Header-Datei definiert die Device-Struktur und die folgenden Funktions-Prototypen:

Funktion	Beschreibung
`ad7843_init_device`	Initialisiert die Device-Struktur und alloziiert Queue Speicher
`ad7843_exit_device`	Gibt Speicher frei
`ad7843_get_next_msg`	Liest die nächste Nachricht -falls vorhanden- aus der Queue aus

Tabelle 22: Touchpanel-Treiber

6.1.3.6 GUI-API

Das GUI-API hilft bei der Entwicklung von grafischen Benutzer-Oberflächen. Es stellt Funktionen zur Verfügung, um eine grafische Komponenten-Strukturen aufzubauen. Eine Komponente besitzt eine `event` Funktion, die bei unterschiedlichen Ereignissen aufgerufen wird. Hierdurch werden z.B. Tastatur- und Maus-Nachrichten von einer übergeordneten Komponente an ihre Kind-Komponenten bis zur Verarbeitung weitergereicht. Der Aufruf an die Komponente, sich in einem Bereich neu zu zeichnen (Paint-Mechanismus), wird ebenfalls über die `event` Funktion durchgeführt. Für die grafische Oberfläche der Referenzanwendung wurde eine Schaltflächen-Komponente (Button) entwickelt, die mit verschiedenen Farbschemas konfiguriert werden kann. Die API Funktionen sind im `gui` Verzeichnis der Softwarebibliothek zu finden. Die `gui.h` Datei enthält die Komponenten-Basis-Struktur und Funktionen für die Verteilung von Nachrichten an die Komponenten (Event-Dispatching).

Die folgende Tabelle enthält einen Ausschnitt aus der API:

Funktion	Beschreibung
`gui_set_first_child`	Definiert die Basis-Komponente in der Baumstruktur
`gui_add_comp`	Fügt eine Kind-Komponente zu einer anderen hinzu
`gui_add_repaint_comp`	Markiert das Komponentenrechteck zum neu zeichnen
`gui_fire_key_event`	Delegiert eine Keyboardnachricht an die Komponenten
`gui_fire_mouse_event`	Delegiert eine Mausnachricht an die Komponenten
`gui_repaint`	Zeichnet Komponenten neu
`gui_set_disabled`	Disabled die Komponente; bei Buttons wird das „disabled“ Farbschema ausgewählt.

Tabelle 23: Ausschnitt aus den GUI-API Funktionen

Für spezielle Komponenten, wie der Schaltfläche, werden separate Dateien angelegt. Die `gui_button.h` Header-Datei enthält die Button-Struktur und die Prototypen für die Button-Funktionen. Die Implementierung mit `event` Funktion befindet sich in der `gui_button.c` Datei.

Die folgende Tabelle enthält die button-spezifischen Funktionen:

Funktion	Beschreibung
`gui_button_set_text`	Setzt den darzustellenden Text
`gui_button_select`	Selektiert einen Button (wählt das „selected“ Farbschema aus)

Tabelle 24: GUI-API Button Funktionen

Eine Anwendung wird über eine Listener-Funktion von der Button-Komponente über eine Zustandsänderung (z.B. Knopfdruck) informiert. Die Zuweisung der Listener-Funktion erfolgt über die `listener` Variable bei der Struktur-Initialisierung. Der Nachrichten-Verteilvorgang wird anhand der folgenden Abbildung illustriert.

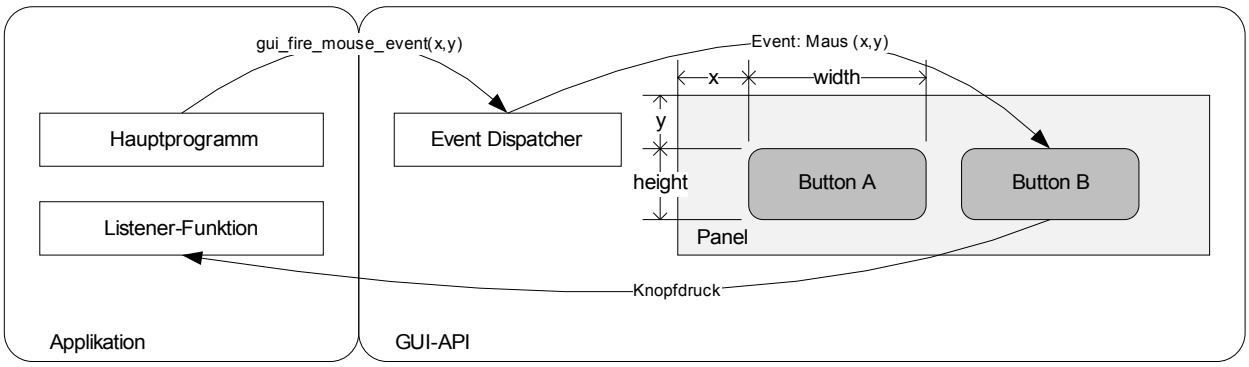


Abbildung 54: GUI-API Nachrichten-Verteilung (Event-Dispatching)

6.1.4 Steuerung

6.1.4.1 PS/2 Tastatur-Treiber

Der PS/2 Treiber implementiert die Steuerfunktionen für den Opencore PS/2-Core (siehe Kapitel 4.4) . Dieser ist in dem `device/oc_ps2` Verzeichnis zu finden. Die `oc_ps2_master.h` Datei enthält die Device-Struktur und die Funktions-Prototypen. Eine Anwendung deklariert diese Struktur mit der EA-Adresse, IRQ-Nummer und der Größe der Tastatur-Queue. Anschließend wird die oc_ps2_init_device Funktion zur Initialisierung des Treibers aufgerufen und die Anwendung kann mittels der `oc_ps2_has_next` und `oc_ps2_get_next` Funktionen die Tasten-Codes aus der Queue auslesen.

Funktion	Beschreibung
`oc_ps2_init_device`	Initialisiert den PS/2 Core und alloziiert Speicher für die Tastatur-Queue
`oc_ps2_exit_device`	Gibt den Speicher frei
`oc_ps2_has_next`	Überprüft, ob eine Tastatur-Code in der Queue liegt
`oc_ps2_get_next`	Liest den nächsten Code aus der Queue aus

Tabelle 25: PS/2 Tastatur-Treiber-Funktionen

6.1.4.2 PIO-Button-Treiber

Der PIO-Button-Treiber wurde entwickelt, um die auf dem Entwicklungsboard befindlichen Taster als Steuersignalquelle in das System einzubinden. Die Tasten bekommen einen Code (Queue-Index) zugeordnet, dessen Status durch den Treiber in eine Queue geschrieben wird. Die `pio_button.h` Header-Datei definiert die `pio_button_dev_t` Device-Struktur. Diese wird bei der Deklaration in der Anwendung mit der IRQ-Nummer und der PIO IP-Core EA-Adresse initialisiert. Implementiert wird der Treiber in der `pio_button.c` Datei. Auf die Tastenzustände wird über die Queue-API zugegriffen (siehe Kapitel 6.1.4.3):

Funktion	Beschreibung
`pio_button_init_device`	Initialisiert den Treiber und registriert die ISR
`pio_button_exit_device`	Entfernt die ISR

Tabelle 26: PIO-Button-Treiber

6.1.4.3 Queue-API

Die Queue-API wurde konzipiert um der verwendenden Anwendung einen einheitlichen Zugang zu Steuerinformationen zu verschaffen. Diese Informationen bestehen aus (Tasten-)Code und Zustand (gedrückt, losgelassen oder wiederholt). Ein Timer generiert bei Bedarf Wiederholungs-Codes. Aus Zeitgründen wurde bisher nur der PIO-Button-Treiber in die Queue-API integriert. Eine Integration des PS/2- und Touchpanel-Treibers steht aus. Die Queue-API-Dateien befinden sich im `device/queue` Verzeichnis. Ein Testprogramm ist im Anhang G.3 zu finden. Folgende Tabelle enthält die definierten Treiber-Funktionen:

Funktion	Beschreibung
`queue_init_device`	Initialisiert die Queue (Speicher-Allokation), Startet den Timer
`queue_exit_device`	Gibt den Queue-Speicher frei, Stoppt den Timer
`queue_get_next`	Auslesen der nächsten Queue-Nachricht
`queue_set_current_state`	Setzen des aktuellen Zustands eines Eintrags

Tabelle 27: Queue-API

6.1.5 Integration der Softwarebibliothek

Zur Verwendung der Audio-Plattform-Softwarebibliothek in Anwendungs-Projekten müssen die Linker- und Compiler-Einstellungen des verwendenden Projektes erweitert werden. Diese sind in der folgenden Tabelle aufgeführt. Die Angabe bezieht sich auf die Debug-Version der Bibliothek.

Werkzeug	Option	Wert	Beschreibung
Compiler	`-I`	`../nios2_audio_platform/src`	Compiler-Include-Pfad
Linker	`-L`	`../nios2_audio_platfom/Debug`	Bibliotheksverzeichnis (Debug-Version)

Linker	`-l`	nios2_audio_platform	Bibliotheksname

Tabelle 28: Audio-Plattform Integration, Linker- und Compiler-Einstellungen

6.2 MP3-Decoder Integration

Der Integrierte MP3-Decoder stammt aus dem Helix-Projekt[1] und wurde von RealNetworks entwickelt. Der Quelltext wird über CVS zum Download angeboten[2]. Die Implementierung ist als 32-Bit-Fixedpoint-Version verfügbar und damit für Prozessoren ohne Fließkomma-Einheit geeignet. Es existiert eine ARM-Architektur Version, die als Vorlage für die vorzunehmende *Nios II* Implementierung dient. Eine Optimale Decodier-Performance wird erreicht, wenn der Prozessor über folgende mathematische Funktionen verfügt:

- Lange Multiplikation (zwei 32-Bit Faktoren erzeugen ein 64-Bit Ergebnis)
- Lange Multiplikation mit 64-Bit Ergebnis Akkumulation

Dies ist der Decoder-Dokumentation auf der Helix-Website ist zu entnehmen. Der verwendete *Nios II* Prozessor unterstützt diese Befehle nicht direkt. Deshalb wird diese Vorraussetzung durch die Entwicklung einer 64-Bit-ALU, siehe Kapitel 4.3, geschaffen.

6.2.1 Implementierung

6.2.1.1 Projekt-Struktur Erstellung

Es wird ein neues *Nios II C/C++ Library* Projekt namens `nios2_mp3_decoder` angelegt. Innerhalb des Projektpfades wird ein `src` Unterverzeichnis erzeugt, in welches die MP3-Decoder Quelltexte kopiert werden. Nicht benötigte Dateien werden anschließend entfernt. Folgende Abbildung zeigt die entstandene Verzeichnisstruktur:

```
nios2_mp3_decoder/src:
LICENSE.txt  RCSL.txt  RPSL.txt  mp3dec.c  mp3tabs.c  pub  readme.txt  real

nios2_mp3_decoder/src/pub:
mp3common.h  mp3dec.h  mpadecobjfixpt.h  statname.h

nios2_mp3_decoder/src/real:
alu64.h      buffers.c  dequant.c  hufftabs.c         scalfact.c  trigtabs.c
assembly.h   coder.h    dqchan.c   imdct.c            stproc.c
bitstream.c  dct32.c    huffman.c  nios2_polyphase.c  subband.c
```

Abbildung 55: MP3-Decoder-Verzeichnisstruktur

6.2.1.2 Nios II Anpassungen

Aus der Dokumentation des MP3-Decoders geht hervor, dass die `assembly.h` Datei für einen neuen Prozessor angepasst werden muss. Die durchgeführte Erweiterung wird im folgenden

[1] https://datatype.helixcommunity.org/Mp3dec

[2] https://helixcommunity.org/viewvc.cgi/datatype/mp3/codec/fixpt

Quelltext-Ausschnitt dargestellt. Es wird ein neuer `define` Block für den *Nios II* Prozessor eingefügt. Dieser beinhaltet einen `include` für die `alu64.h` Header-Datei und implementiert die benötigten Funktionen unter Verwendung der 64-Bit-ALU.

```
... #elif defined(__GNUC__) && defined(__nios2__)
 typedef long long Word64;
#include "alu64.h"  // 32-bit * 32-bit with hi-word 32-bit result  static __inline int
MULSHIFT32(int _x, int _y){   return ALU64_A_MULT_LOAD(_x, _y); } // abs function
static __inline int FASTABS(int _x){     return ALU64_FASTABS(_x); } // count leading
zeros static __inline int CLZ(int _x){   return ALU64_CLZ(_x); } // multiply and
accumulate static __inline Word64 MADD64(Word64 _sum, int _x, int _y){      return
ALU64_A_MADD(_sum, _x, _y); } // shift arithmetic right static __inline Word64
SAR64(Word64 x, int n){     return x>>n; } // shift left static __inline Word64
SAL64(Word64 x, int n){     return x<<n; }  #else ...
```

Quelltext 2: MP3-Decoder Erweiterung unter Verwendung der 64-Bit-ALU Befehle

Die Integration der 64-Bit-ALU Befehle in die `assemby.h` Datei (siehe oben) ist nicht ausreichend um Echtzeit bei der Decodierung zu erreichen. Daraufhin wurde eine ausführliche Performance-Analyse des Decoders mittels *GNU-Profiler* durchgeführt, als Ergebnis konnten die „Polyphase“-Funktionen in der `polyphase.c` Datei als Geschwindigkeits-Schwachpunkte identifiziert werden. Diese wurden anschließend an die 64-Bit-ALU angepasst und in der `nios2_polyphase.c` Datei gespeichert (siehe Anhang I.2).

6.2.1.3 Compiler-Einstellungen

Um eine maximale Performance der erzeugten Bibliothek zu erreichen, wird in den Compiler-Einstellungen die Optimierungsstufe auf „Optimize most“ (`-O3`) eingestellt, das der höchsten Stufe entspricht. Als zusätzlicher Compiler-Include-Pfad wird `src/pub` angegeben.

6.2.1.4 Helix MP3-Decoder-API

Der Decoder stellt über die `mp3dec.h` Datei folgende öffentliche Funktionen zur Verfügung:

Funktionsname	**Beschreibung**
`MP3InitDecoder`	Initialisierung des Decoders mit Speicherallokation

Funktionsname	Beschreibung
`MP3FreeDecoder`	Freigabe des Decoderspeichers
`MP3FindSyncWord`	Suchen der Frame-Start Markierung (Sync-Word) innerhalb des Datenstroms
`MP3Decode`	Decodier Funktion
`MP3GetLastFrameInfo`	Informationsrückgabe über den letzten Frame an das aufrufende Programm

Tabelle 29: Helix MP3-Decoder-API

6.2.2 Verwendungshinweise

Zur Verwendung der MP3-Bibliothek in anderen Projekten (z.B. in der Referenzanwendung) müssen die Linker- und Compiler-Einstellungen des verwendenden Projektes erweitert werden. Diese sind in der folgenden Tabelle aufgeführt:

Tool	Option	Wert	Beschreibung
Compiler	`-I`	`../nios2_mp3_decoder/src/pub`	Compiler-Include-Pfad
Linker	`-L`	`../nios2_mp3_decoder/Debug`	Bibliotheksverzeichnis
Linker	`-l`	nios2_mp3_decoder	Bibliotheksname

Tabelle 30: MP3-Decoder Integration, Linker- und Compiler Einstellungen

6.2.3 Ergebnis

Nach der Integration der MP3-Bibliothek in die Referenzanwendung wurde eine Geschwindigkeitsmessung mit dem *Performance-Counter MegaCore* durchgeführt. Dabei wurde die Ausführungszeit der MP3-Decode-Funktion während des Abspielens mehrerer unterschiedlicher Musikstücke gemessen. Die Anzahl der dabei decodierten Bytes wurde summiert und anschließend durch die ermittelte Zeit geteilt. Das Ergebnis zeigt mit unterschiedlichen MP3-Bitraten eine durchschnittliche Decodiergeschwindigkeit von 393 KB/s. Das liegt deutlich über den benötigten 172,3 KB/s bei einer Samplerate von 44,1 kHz in 16-Bit Stereo und somit können Audiodateien in Echtzeit (ohne Zeitverzögerung) wiedergegeben werden.

6.3 Entwicklung der „open juke“-Referenzanwendung

Die Entwicklung der Referenzanwendung erfolgte, nachdem die Hardwareplattform und die Softwarebibliothek mit den Treibern fertig gestellt waren. Der Name „open juke“ ist an Jukebox angelehnt und soll durch das „open“ signalisieren, dass die entwickelte Anwendung offen für Erweiterungen ist. Der erste Entwurf der Anwendung wurde ohne MP3-Decoder implementiert. Die Funktionen beschränkten sich auf das Auf- und Abspielen von unkomprimierten WAV-Dateien. Zu einem späteren Zeitpunkt stellte sich jedoch bei der Internet-Recherche heraus, dass es einen frei verfügbaren Software-MP3-Decoder gibt, der in einem Mikrocontroller-Projekt eingesetzt wird . Dies führte schließlich zur Implementierung des MP3-Decoders auf dem *Nios II* Prozessor (siehe Kapitel 6.2) und zur anschließenden Integration in die Referenzanwendung.

Folgende Abbildung zeigt das erstellte grafische Benutzerinterface:

Abbildung 56: Grafisches Benutzerinterface

Eine vollständige Funktionsbeschreibung der Anwendung ist in der Bedienungsanleitung im A zu finden.

6.3.1 Ziel

Die erstellte Audioplattform stellt ein Rahmenwerk für die Anwendungsentwicklung bereit. Um die Funktionalität und die Anwendbarkeit der API zu verifizieren (Proof-Of-Conzept) wird eine Referenzanwendung erstellt, die außer den üblichen Funktionen eines MP3-Players die Aufnahme von Audiodaten in CD-Qualität ermöglicht.

6.3.2 Konzept

Der Funktionsumfang einer Jukebox umfasst im Wesentlichen das Auswählen und Abspielen von Musiktiteln aus einer zuvor definierten Liste. Dieses übliche Konzept wird hier aufgegriffen.

Die Anwendung gliedert sich in zwei Bereiche: Den Steuerbereich mit den Abspiel-, Aufnahme-, Pause- und Stopp-Funktionen und den Titel-Navigationsbereich, über den der Verzeichnisbaum der SD-Karte durchsucht werden kann. Die Wiedergabe eines Titels erfolgt, sobald dieser im Navigationsbereich ausgewählt wird. Das aktuelle Verzeichnis mit allen verfügbaren Titeln wird in einer internen Liste gespeichert. Dies ermöglicht die Titelnavigation über Vor- und Rück-Taste, auch wenn das Verzeichnis im Navigationsbereich zwecks Suche neuer Titel gewechselt wurde. Eine Sortierfunktion sorgt dafür, dass die Titel in alphabetischer Reihenfolge dargestellt werden. Die Aufnahmefunktion speichert die aufgenommenen Daten in dem im Navigationsbereich ausgewählten Verzeichnis. Der Name der aufzunehmenden Datei wird aus einem Präfix (recording) und der aktuellen Systemzeit gebildet. Es wird geprüft ob ein Name bereits vergeben wurde. Ist dies der Fall, so wird dem Dateinamen eine Nummerierung angehängt. Hierdurch soll eine Doppelbenennung vermieden werden. Der aktuell wiedergegebene Titel wird zusammen mit der verbleibenden Abspielzeit in einer Statuszeile angezeigt.

6.3.3 Zustandsautomat

Die interne Steuerung der Anwendung erfolgt über einen Zustandsautomaten. Jede Benutzeraktion, wie das Betätigen der „Play“-Taste, löst einen Zustandswechsel aus. Dabei wird anhand des neuen und alten Zustands bestimmt, welche Übergangsfunktionen ausgeführt werden. Diese aktualisieren das GUI und starten bzw. stoppen die Wiedergabe oder die Aufnahme. Die Pause-Funktion ist ein einfaches Flag und wird nicht über einen Zustand realisiert. Folgende Abbildung stellt die Zustände mit ihren Übergängen dar. Diese sind mit den auslösenden Benutzer-Aktionen beschriftet:

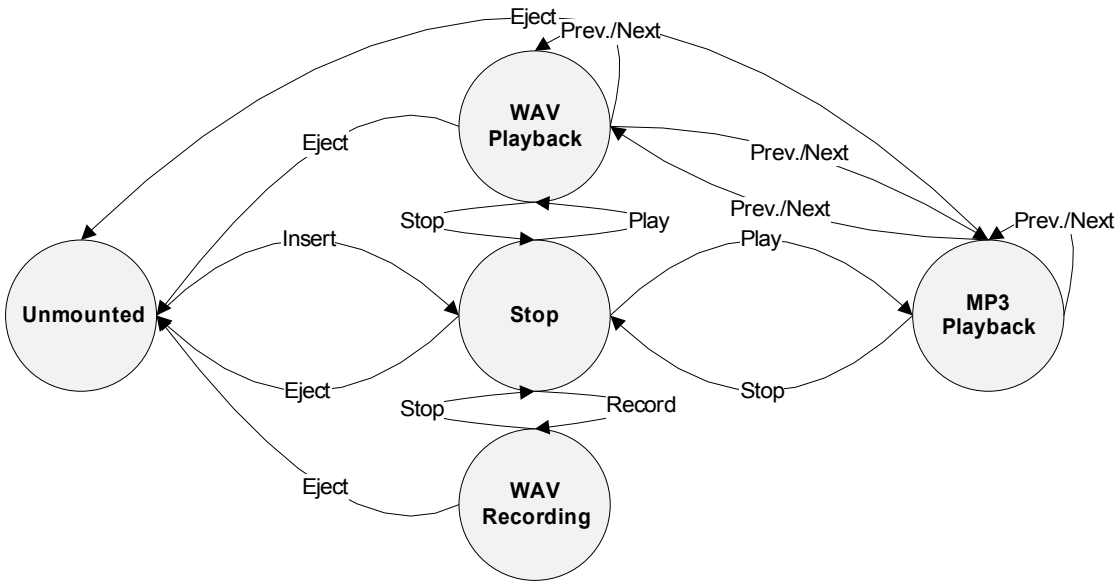


Abbildung 57: „open juke" Zustandsautomat

6.3.4 Implementierung

Für die Implementierung wird ein neues *Nios II* Applikations-Projekt in der IDE angelegt. Die Compiler und Linker-Einstellungen werden um die Audio-Plattform-Softwarebibliothek erweitert

(siehe Kapitel 6.1.5). Die Anwendung wird in mehrere Dateien aufgeteilt. Die `oj_gui.c` Datei enthält die Initialisierung der GUI Komponenten-Strukturen. Die `oj_gui.h` Datei definiert Farbkonstanten, Schaltflächennamen und Größen. Die `main.c` Datei enthält die Treiber-Strukturen und die main-Funktion. Die `list.h` und `list.c` Datei enthalten Funktionen zur Playlist-Steuerung und Sortierung.

6.3.4.1 Grafisches Benutzerinterface

Das grafische Benutzerinterface wird mit Hilfe des GUI-API erstellt (siehe Kapitel 6.1.3.6). Die Wurzel der Komponentenhierarchie stellt ein unsichtbares Panel dar, das nur als Container für die Kindelemente fungiert. Die Schaltflächen werden in der `init_gui` Funktion zum Panel hinzugefügt. Jede Schaltfläche besitzt in der `oj_gui.c` Datei eine eigene Struktur, in der die Button-ID, -Größe und das Farbschema sowie Referenzen auf die Event- und Listener-Funktionen zugeordnet werden. Eine Konfiguration der Schaltelemente kann über die Veränderung der GUI Konstanten in der `oj_gui.h` Datei erfolgen (siehe Anhang I.6.2).

6.3.4.2 Initialisierung der Treiber-Strukturen

In der `main.c` Datei werden die Treiber-Strukturen deklariert und mit den Konstanten für die EA-Adressen und IRQ-Nummern aus der `system.h` Datei beschrieben.

6.3.4.3 Button-Listener-Funktion

Die Steuerung des Zustandsautomaten erfolgt über die `ui_button_listener` Funktion (siehe Abbildung 58). Diese ist in der `main.c` Datei definiert. Bei einem Aufruf wird der Funktion eine Referenz auf die auslösende Button-Komponente übergeben. Diese enthält eine Komponenten-ID, anhand der innerhalb der Listener-Funktion entschieden wird, welche Zustands-Änderung durchgeführt werden muss (siehe S. 182).

6.3.4.4 Hauptprogramm

Die Initialisierung der Treiber erfolgt zu Beginn der `main` Funktion über die mit `init` bezeichneten Funktionen (siehe S. 183). Bei der Initialisierung des Audio-Treibers wird standardmäßig das Mikrofon als Eingangssignal eingestellt. Dies kann über die AUDIO_MICROPHONE_ENABLE Konstante geändert werden. Da das Entwicklungssystem keine batteriegepufferte RTC besitzt, wird anstatt des Auslesens der Uhrzeit aus einem RTC-Baustein, die Systemzeit in der `init_time` Funktion auf einen konstanten Wert gesetzt. Im Anschluss an die Initialisierung der Treiber wird die Hauptschleife durchlaufen. Diese überprüft die Nachrichten-Queues der angeschlossenen Steuerungs-Treiber und leitet ggf. Nachrichten an die GUI-API weiter. Je nach Zustand wird das Abspielen oder Aufnehmen der Audiodaten ausgeführt. Am Schluss der Hauptschleife wird die GUI bei Veränderungen neu gezeichnet. Die folgende Abbildung stellt das Hauptprogramm in einem Flussdiagramm dar.

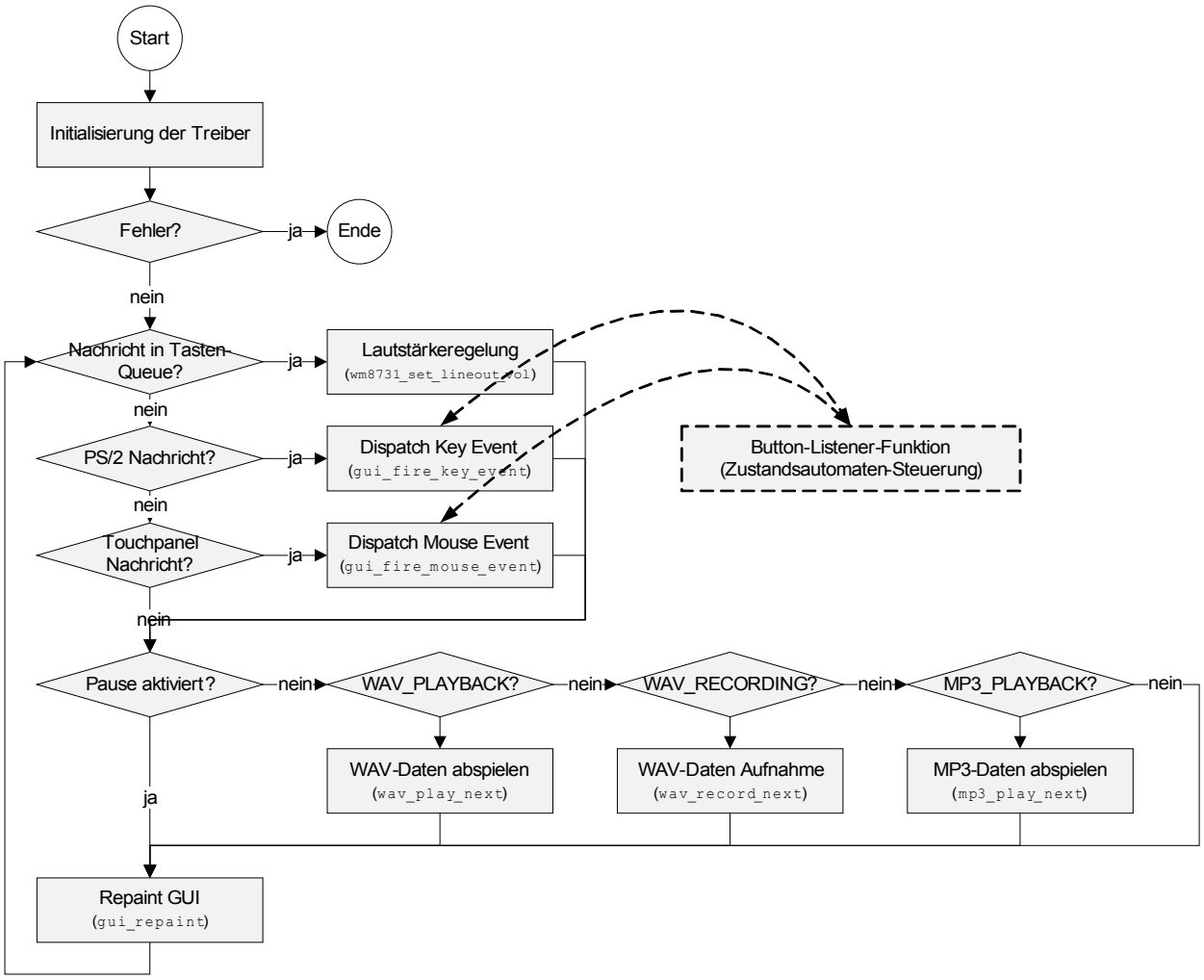


Abbildung 58: "open juke"-Flussdiagramm der main-Funktion

6.3.5 Ergebnis

Die Jukebox-Anwendung zeigt die volle Funktionsfähigkeit der Audio-Plattform. Das Steuerinterface kann durch die Integration zusätzlicher Cores und deren Treiber erweitert werden. Der Speicherverbrauch der Debug-Version mit vielen zusätzlichen Textausgaben beträgt ca. 278 KB. Es bleiben damit ca. 795 KB für Heap und Stack übrig. Eine Speicheroptimierung wurde nicht durchgeführt. Der Verbrauch der Hardware Ressourcen kann in Kapitel 5.2.4 nachgelesen werden. Einige wünschenswerte Funktionen, wie das Löschen von Aufnahmen und die Anzeige der Rest-Aufnahmezeit, wurden nicht implementiert (Die nötigen Treiber-Funktionen sind vorhanden).

7 Zusammenfassung und Ausblick

Das in der Einleitung definierte Ziel der Entwicklung einer Audio-Plattform mit konfigurierbarem Steuerinterface konnte mit den in der vorliegenden Arbeit dokumentierten Mitteln realisiert werden.

Ein Proof-Of-Concept wurde mit der Erstellung der „open juke“-Referenzanwendung erfolgreich durchgeführt. Diese zeigt durch ihr konfigurierbares grafisches Benutzerinterface die Variabilität des entstandenen Systems. Die Entwicklung des optimierten SPI-Masters für die SD-Karten-Datenspeicherung ermöglicht eine Aufnahme von Audiodaten in CD-Qualität. Die durchgeführten Messungen zeigen gegenüber dem frei verfügbaren Opencore SPI-Core eine deutliche Geschwindigkeitssteigerung in der Datenübertragung. Die Integration des MP3-Decoders mit der zusätzlich entwickelten 64-Bit-ALU rundet den Funktionsumfang des Systems ab. Testprogramme, die während der Entwicklung entstanden sind, können in weiteren, auf diesem System aufbauenden Arbeiten, verwendet werden.

Hierfür gibt es folgende Ansätze:

- Integration eines Ethernet IP-Cores für ein Internet-Radio
- Integration eines MP3-Encoders für ein MP3-Diktiergerät
- Integration weiterer CODECs wie AAC (Helix Projekt) oder Ogg Vorbis (Tremor)
- Erweiterung des WM8731 IP-Cores um die Mono-Kanalwahl
- Änderung der SD-Karten-Anbindung von SPI zu 4-Bit-Parallel zur Steigerung der Datenübertragungsgeschwindigkeit
- Entwicklung eines Grafik-IP-Cores (GPU) zur Unterstützung des Prozessors bei der Grafikdarstellung (Flächen füllen, Linen zeichnen, etc.)

Anhang

A Bedienungsanleitung der Referenzapplikation

A.1 Inbetriebnahme

Vorraussetzung für die Inbetriebnahme ist die vollständige Installation der Entwicklungsumgebung mit *Nios II IDE* auf dem Host-Computer (siehe Kapitel 2.2). Das Entwicklungssystem wird an die Stromversorgung und an den Host via USB angeschlossen. Ein Kopfhörer oder PC-Lautsprecher wird mit dem Audioausgang verkabelt. Das `projects` Verzeichnis wird von der beiliegenden CD-ROM in ein Verzeichnis auf die Festplatte kopiert (z.B. `c:/Programme/altera`). Die *Nios II IDE* wird gestartet und die im `projects/eclipse-workspace` befindlichen Projekte werden über den Befehl „File→Import→General→Existing Projects into Workspace“ von der Festplatte importiert. Anschließend wird die Datei *spi_base_time_limited.sof* aus dem *projects/spi_base* Verzeichnis mittels *Quartus II Programmer* in den FPGA programmiert. Die open_juke Anwendung wird aus der IDE per Rechtsklick auf das Projekt und mit „Run As→Nios II Hardware“ gestartet.

A.2 Funktionsbeschreibung

Die Referenzapplikation verfügt über einen Touchscreen mit Schaltflächen, über die der Benutzer die Anwendung mit dem Finger oder einem Stift steuern kann, siehe Abbildung 56. Zusätzlich lassen sich einige Funktionen über eine angeschlossene PS/2 Tastatur steuern. Eine Lautstärkeregelung erfolgt per Board-Taster. Alle Steuerbefehle sind in der folgenden Tabelle aufgeführt. Die Benutzerführung wird durch eine zustandsabhängige Farbdarstellung erreicht. Ein helles Grau signalisiert, dass eine Schaltfläche auswählbar ist. Ein mittleres Grau zeigt an, dass die Funktion nicht wählbar ist. Ausgewählte Schaltflächen werden dunkelgrau hinterlegt. Die Dateiliste besteht ebenfalls aus Schaltflächen, die ein anderes Farbschema besitzen. So wird ein ausgewählter Eintrag in der Dateiliste blau dargestellt. Auf dem Touchscreen werden zusätzlich der ausgewählte Titel und die verbleibende Abspielzeit in Sekunden angezeigt. Bei der Record-Funktion wird eine neue Datei im aktuell ausgewählten Verzeichnis angelegt. Der Dateiname wird aus einem Präfix und einem Systemzeitstempel erstellt.

Touch-Fläche	**PS/2 Tastatur**	**Board-Taste**	**Bechreibung**
PREV	←		Spielt den vorherigen Titel ab
NEXT	→		Spielt den nächsten Titel ab
PLAY	Plus		Spielt den ausgewählten Titel ab

Touch-Fläche	PS/2 Tastatur	Board-Taste	Bechreibung
REC	Enter		Startet die Aufnahme
PAUSE			Unterbricht das Abspielen bzw. die Aufnahme, durch ein erneutes Auswählen wird die Pause wieder aufgehoben
STOP	Minus		Unterbricht das Abspielen und die Aufnahme
EJECT			Beendet das Abspielen und Aufnehmen und ermöglicht das Austauschen der SD-Karte
INSERT			Meldet eine neue SD-Karte an und liest das Verzeichnis der Karte ein
/			Der mit dem Schrägstrich beginnende oberste Dateilisteneintrag wechselt in die Übergeordnete Verzeichnisebene
UP	↑		Scrollt die Dateiauswahlliste nach oben
DOWN	↓		Scrollt die Dateiauswahlliste nach unten
		Button 1	Lautstärkeregelung: leiser
		Button 2	Lautstärkeregelung: lauter

Tabelle 31: Steuerbefehle der Referenzapplikation

B Entwicklungswerkzeuge

In der folgenden Tabelle werden verwendete Werkzeuge mit Bezugsquelle aufgeführt:

Name	Beschreibung
Quartus II Software-Web Edition	Altera FPGA Entwicklungsumgebung http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html
ModelSim-Altera-	ModelSim Simulationsprogramm

Web Edition	http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html
Nios II Embedded Design Suite	Nios II Entwicklungsumgebung http://www.altera.com/products/ip/processors/nios2/tools/ni2-development_tools.html
Folgende Tools befinden sich ebenfalls auf der beiliegenden CD-ROM im `tools` Verzeichnis.	
Audacity	Audiodatei-Bearbeitungsprogramm http://www.audacity.de
LogicPort	Logic Analyser (erfordert Hardware) http://www.pctestinstruments.com
Frhed	Hexeditor http://www.kibria.de

Tabelle 32: Entwicklungswerkzeuge

C Avalon zu Wishbone Signalzuordnung

Mithilfe der Signalzuordnungen in folgender Tabelle können Wishbone-Slave IP-Cores an das Avalon-Bussystem angebunden werden:

Avalon-Slave-Signal	**Type**	**Wishbone-Slave-Signal**
clk_i	Input	wb_clk_i
reset_i	Input	wb_rst_i
address_i	Input	wb_adr_i
writedata_i	Input	wb_dat_i
readdata_o	Output	wb_dat_o
write_i	Input	wb_we_i
chipselect_i	Input	wb_stb_i
chipselect_i	Input	wb_cyc_i
waitrequest_n	Output	wb_ack_o
byteenable	Input	wb_sel_i
n/a	Output	wb_err_o
irq_i		wb_int_o

Tabelle 33: Avalon zu Wishbone Signalzuordnung

D Erstellte Projekte

D.1 Quartus II Projekte

Die erstellten *Quartus II* Projekte können von der beiliegenden CD-ROM aus dem Verzeichnis `projects` geöffnet werden:

Projektname	**Beschreibung**
spi_base	Audio-Plattform-Projekt
cross_domain_sync	IP-Core zur asynchronen Datenübertragen zwischen zwei Clock-Domains
gf_mux	IP-Core um zwei asynchrone Clock-Signale „glitch-free“ zu multiplexen
lcd_16bit	16-Bit Hi-Color zu 32-Bit True-Color Konvertierung mit *Avalon-ST* Schnittstellen
mult64	64-Bit ALU mit *Avalon Custom-Instruction*-Schnittstelle
nil	Nirwana RAM, Lesezugriffe liefern 0 und Schreibzugriffe werden ignoriert mit *Avalon-MM* Schnittstelle.
oc_i2c	*Opencores* I2C-Master IP-Core mit *Avalon-MM* Schnittstelle
oc_ps2_master	*Opencores* PS/2 IP-Core mit *Avalon-MM* Schnittstelle
oc_spi	*Opencores* SPI-Master IP-Core mit *Avalon-MM* Schnittstelle
spi_master	SPI-Master mit *Avalon-MM* Schnittstelle.
wm8731	WM8731 Audio-CODEC IP-Core mit *Avalon-MM* und *Avalon-ST* Schnittstellen

Tabelle 34: Erstellte Quartus II Projekte

D.2 Nios II Projekte

Die erstellten *Nios II* Projekte können von der beiliegenden CD-ROM aus dem Verzeichnis `projects/eclipse-workspace` in die *Nios II IDE* importiert werden:

Projektname	Beschreibung
`test_spi_master`	SPI-Master IP-Core Testprogramm
`test_mmc_sd_spi`	SD-Treiber Testprogramm
`test_fat_read_write`	SD-Karten Performance-Messprogramm
`test_control`	PS/2- und Board-Taster-Testprogramm
`open_juke`	„open juke“ Referenzapplikation
`nios2_mp3_decoder`	MP3-Decoder-Bibliothek
`nios2_audio_platform`	Audio-Plattform-Softwarebibliothek

Tabelle 35: Erstellte Nios II Projekte

E 64-Bit-ALU-Befehle

Die ALU-Befehle besitzen zwei 32-Bit Operanden `A` und `B` sowie einen 32-Bit-Rückgabewert `R`. Die internen Akkumulatoren `AccuA` und `AccuB` haben eine Breite von 64-Bit. Die folgende Tabelle zeigt den Befehlssatz, der in der `alu64.h` Header-Datei definiert ist:

Befehl	Beschreibung
`LOAD_BOTH`	Lädt beide Akkumulatoren mit einem 64-Bit Zahlenwert, der auf `A` und `B` aufgeteilt ist. `A` enthält die höherwertigen 32-Bit. Ohne Rückgabewert. Pseudocode: `AccuA=AccuB=(A<<32)|B`
`CLIP_TO_SHORT`	Kürzt den in `A` übergebenen 32-Bit Zahlenwert auf 16-Bit. `B` wird nicht verwendet. Pseudocode: `R=A>32767 ? 32767:(A<-32768?-32768:A)`
`FASTABS`	Absolut Funktion. `A` enthält 32-Bit Zahlenwert. Das 32-Bit-Ergebnis ist der positive `A` Inhalt. `B` wird nicht verwendet. Pseudocode: `R=A<0?-A:A`
`STORE`	Speichert den Zahlenwert in `B` in den internen Koeffizienten Speicher an Adresse `A`. Ohne Rückgabewert. Pseudocode: `Coeff[A]=B`
`CLZ`	Es werden die dem Zahlenwert in `A` vorangestellten 0-Bits gezählt und zurückgegeben. (Count leading zeros)

Befehl	Beschreibung
A_LOAD	Lädt den AccuA mit 64-Bit Zahlenwert. Siehe LOAD_BOTH. Pseudocode: AccuA=(A<<32)|B
A_MULT_LOAD	Multiplikation von A und B. Das 64-Bit Ergebnis wird in AccuA gespeichert. Pseudocode: AccuA=A*B
A_MULT_ADD	Multiplikation von A und B. Das 64-Bit Ergebnis wird zum AccuA addiert. Ohne Rückgabewert. Pseudocode: AccuA=AccuA+A*B
A_MULT_ADD_COEF	Multiplikation von Koeffizientenspeicherwert an Adresse A mit B. Das 64-Bit Ergebnis wird zum AccuA addiert. Ohne Rückgabewert. Pseudocode: AccuA=AccuA+Coeff[A]*B
A_GET	Liefert die unteren 32-Bit von dem um den Wert in A nach rechts verschobenen Inhalt von AccuA. Pseudocode: R=AccuA>>A
B_LOAD	Siehe A_LOAD mit Akkumulator B anstatt Akkumulator A
B_MULT_LOAD	s.o.
B_MULT_ADD	s.o.
B_MULT_ADD_COEF	s.o.
B_GET	s.o.

Tabelle 36: 64-Bit-ALU Befehlssatz

F Messergebnisse

F.1 Ergebnisse der Schreib- und Lesegeschwindigkeitsmessung

	Schreib-Lesegröße [Byte]									
	16	32	64	128	256	512	1024	2048	4096	8192
Karte	Schreib-Lesegeschwindigkeit [kbit/s]									
1-GB SanDisk Ultra II schreiben	1200	1433	1606	1702	1769	2283	3716	4720	5742	**11453**
1-GB SanDisk Ultra II lesen	3041	4142	5075	5705	6150	6613	8039	10147	12158	13102
128-MB schreiben	295	312	325	327	332	349	6921	**8868**	7955	7970
128-MB lesen	2636	3869	5043	5945	6618	7232	9397	11342	11394	11435
32-MB schreiben	1400	1732	1968	2098	2203	3035	3001	3034	3057	3019
32-MB lesen	2578	3748	4843	5672	6285	6840	6961	7014	7043	7050
2-GB SanDisk schreiben	1083	1211	1279	1320	1339	1715	2295	5788	8568	8585
2-GB SanDisk lesen	2934	3871	4634	5122	5414	5691	8188	10268	12086	12084
2-GB SanDisk Extreme III schreiben	1304	1546	1708	1764	1813	2380	3607	5478	6419	7056

2-GB SanDisk Extreme III lesen	3039	4075	4919	5481	5820	6134	7904	10039	11569	12920
2-GB Transcend schreiben	**1644**	**2278**	**3128**	**3227**	**3246**	**4568**	**7690**	8613	**9201**	9535
2-GB Transcend lesen	**3683**	**5327**	**6870**	**8019**	**8769**	**9505**	**11129**	**12303**	**13027**	**13726**
2-GB OCZ schreiben	517	579	586	594	611	695	1484	2596	4250	6954
2-GB OCZ lesen	3117	4216	5888	6712	7226	7719	9714	11364	12421	13474
1-GB Kingston schreiben	894	985	1036	1098	1100	1252	2037	5284	5927	10643
1-GB Kingston lesen	3266	4643	5881	6783	7352	7876	9759	11254	12747	13365
1-GB Sharkoon schreiben	820	875	920	951	986	1221	2429	5236	6189	11740
1-GB Sharkoon lesen	2962	4051	5648	6478	6996	7464	9429	11025	12751	13393

Tabelle 37: Ergebnisse der Schreib-Lesegeschwindigkeits-Messung (Maximal-Werte sind hervorgehoben)

F.2 Ergebnisse der Schreib- und Lesezeitmessung

	Schreib-Lesegröße [Bytes]									
	16	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**
Ebene	**Zeit [s]**									
FAT schreiben	12,992	6,6477	3,5481	1,9804	1,2318	1,3988	0,2925	0,2071	0,1208	0,0713
SD schreiben	35,123	35,185	34,839	34,782	34,294	26,854	16,856	13,402	10,851	5,5241
FAT lesen	11,182	5,8228	3,0884	1,7591	1,072	0,4947	0,2688	0,1585	0,0819	0,0444
SD lesen	9,1369	9,1369	9,138	9,1386	9,1451	9,1426	7,6812	6,1435	5,1814	4,838

Tabelle 38: Ergebnisse der Schreib- und Lesezeitmessung mit Verteilung der Rechenzeit auf FAT-Dateihandling und SD-Datentransfer

G Testprogramme

G.1 SPI-Master-Treiber

Das folgende Testprogramm ist in dem `test_spi_master` Projekt zu finden und illustriert die Funktionsweise des SPI-Master-Treibers. Die `spi_dev_t` Struktur wird global deklariert und mit Konstanten aus der `system.h`[1] Datei beschrieben. In der `main` Funktion wird der Treiber initialisiert und anschließend wird der Zahlenwert `0xFF` zehnmal auf den SPI-Bus ausgegeben.

```
1  #include "system.h"
2  #include "device/spi_master/spi_master.h"
3  // device structure declaration
4  static spi_dev_t spi={
5     .base       = (void *)SD_CONTROLLER_BASE,    // Core io address
6     .clk        = SD_CONTROLLER_FREQ,            // Core bus freq
7     .maxClock   = 15000000,                      // 15 MHz max bus freq
8     .slaveNum   = 0
9  };
10 // main test function
11 int main(){
12    spi_master_init_device(&spi);                // Init device
13    spi_cs_enable(&spi);                         // CS enable
14    spi_write_constant(&spi, 0xFF, 10);          // Send 10 bytes of 0xFF
15    spi_cs_disable(&spi);                        // CS disable
16   return 0;
17 }
```

Quelltext 3: SPI-Master Testprogramm

[1] Die system.h Datei wird durch das „System Library Project“ generiert, siehe Kapitel 2.4.4

G.2 SD-Karten-Treiber

Das folgende Testprogramm ist in dem `test_mmc_sd_spi` Projekt zu finden und zeigt die Initialisierung der SPI- und SD-Karten-Treiber sowie einen SD-Karten Lesezugriff.

```
1  #include <unistd.h>
2  #include <fcntl.h>
3  #include "system.h"
4  #include "device/spi_master/spi_master.h"
5  #include "device/mmc/mmc_spi_device.h"
6  // spi device structure declaration
7  static spi_dev_t spi={
8     .base    = (void *)SD_CONTROLLER_BASE, // Core IO address
9     .clk        = SD_CONTROLLER_FREQ,              // Core bus freq
10    .maxClock   = 15000000,                        // 15 MHz max bus freq
11    .slaveNum   = 0
12 };
13 // mmc sd device structure declaration
14 static mmc_spi_dev_t
15   mmcDev={
16    .device = {.name = "/mnt/mmc0"},
17    .numOfBuffers = 5                              // 5 buffers
18   };
19 // main test function
20 int main(){
21    int file; short unsigned int signature;
22    // init spi bus
23    if (spi_master_init_device(&spi) < 0)
24       return -1;
25   // create the mmc block device
26   if (mmc_spi_init_device(&mmcDev, &spi) < 0)
27      return -1;
28   // open the card
29   file = open("/mnt/mmc0", O_RDONLY, 0);
30    // seek to signature position
31   lseek(file, 510, SEEK_SET);
32   // read signature
33   read(file, &signature, 2);
34   // close
35   close(file);
36   return 0;
37 }
```

Quelltext 4: SD-Treiber Testprogramm

G.3 PS/2 -, PIO-Button Treiber und Queue API

Das folgende Testprogramm ist im `test_control` Projekt zu finden und zeigt die Initialisierung der PS/2-, PIO-Button und Queue-Treiber. Eine Testfunktion liest die empfangenen Steuercodes aus und gibt diese über die `printf`-Funktion in der Konsole aus.

```
1  #include <stdio.h>
2  #include "system.h"
3  #include "device/queue/queue.h"
4  #include "device/pio_button/pio_button.h"
5  #include "device/oc_ps2/oc_ps2_master.h"
6  // init queue structure
7  static queue_dev_t queue={
8     .numberOfEntries  = 4, // 4 buttons
9     .totalQueueSize   = 16, .repeatCycles = 1
10 };
11 // init button structure
12 static pio_button_dev_t buttons={
13    .queue   = &queue, .queueStateIndex = 0,
14    .base    = BUTTON_PIO_BASE,
15    .irqNum  = BUTTON_PIO_IRQ,
16    .bits    = BUTTON_PIO_DATA_WIDTH
17 };
18 // init PS/2 structure
19 static oc_ps2_dev_t
20    ps2={
21    .device     = {.name = OC_PS2_NAME},
22    .base       = (void *)OC_PS2_BASE,
23    .irqNum     = OC_PS2_IRQ, .queueSize = 16
24    };
25 // main test function
26 int main(){
27    // init the queue
28    if (queue_init_device(&queue) <0) return -1;
29    // init the buttons
30    if (pio_button_init_device(&buttons) <0) return -1;
31    // init the ps2
32    if(oc_ps2_init_device(&ps2) <0) return -1;
33    while(1){ // test loop
34       queue_msg_t msg;
35       if (queue_get_next(&queue, &msg)){ // read queue message
36          printf("msg code: %d, state:%d\n", msg.code, msg.state);
37       }else
38       if (oc_ps2_has_next(&ps2)){ // read ps/2 key code
39          uint32_t
40             data = oc_ps2_get_next(&ps2);
41          printf("ps2 code: %d, ascii:%c, pressed:%d\n",
42             KEY_GET_CODE(data), KEY_GET_ASCII(data),
43             (data & KEY_STATE_PRESSED)?((data & KEY_STATE_REPEAT)?3:1):0
44          );
45       }
46    }
47    return 0;
48 }
```

Quelltext 5: PS/2, PIO-Button Treiber und Queue API Testprogramm

G.4 Schreib-Lesegeschwindigkeitsmess-Programm

Das folgende Programm wurde verwendet um die Schreib-Lesegeschwindigkeit von unterschiedlichen SD-Karten zu ermitteln (siehe Kapitel 6.1.1.5). Der Programmquelltext ist ebenfalls auf der beiliegenden CD-ROM zu finden (siehe D.2).

```
1  #include <unistd.h>
2  #include <fcntl.h>
3  #include <stdlib.h>
4  #include "system.h"
5  #include "device/spi_master/spi_master.h"
6  #include "device/mmc/mmc_spi_device.h"
7  #include "device/fat/fat_device.h"
8  #include "device/partition/partition_device.h"
9  #include "altera_avalon_performance_counter.h"
10
11 // File size definition (8MB)
12 #define FILE_SIZE                      1000*8192
13
14 // spi device structure declaration
15 static spi_dev_t spi={
16    .base       = (void *)SD_CONTROLLER_BASE, // Core IO address
17    .clk          = SD_CONTROLLER_FREQ,              // Core bus freq
18    .maxClock     = 15000000,                        // 15 MHz max bus freq
19    .slaveNum     = 0
20 };
21 // mmc sd device structure declaration
22 static mmc_spi_dev_t
23   mmcDev={
24    .device = {.name = "/mnt/mmc0"},
25    .numOfBuffers = 5 // 5 buffers
26   };
27 // partition device structure
28 static fs_dev_t
29    fs={
30       .device ={.name = "/mnt/fs"},
31       .parentDevice = "/mnt/mmc0"
32 };
33 // fat volume structure
34 static fat_volume_t
35    fat={
36       .device ={.name = "/mnt/root"},             // root file system
37       .parentDevice = "/mnt/fs/0",                // partition 0
38       .numOfFiles    = 10
39 };
40 // write data to file
41 int write_file(const char*_file, int _total, int _piece){
42    int
43       i, file;
44   // open the file
45   if ((file = open(_file, O_RDWR|O_TRUNC|O_CREAT|FAT_RW_NOUPDATE, 0)) < 0)
46       return file;
47    unsigned char
48       *mem = malloc(_piece);
49
50    // fill with values
51    for (i=0; i < _piece; i++)
52       mem[i] = (uint8_t)i&0xFF;

53    while (_total > 0){
54       PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE, 1);
55      // write piece
56      write(file, mem, _piece);
57      PERF_END(PERFORMANCE_COUNTER_0_BASE, 1);
58       _total -= _piece;
59    }
60   // close
61   close(file);
62
63   free(mem);
64   return 0;
65 }
66 // read data from file
```

```
1  int read_file(const char *_file, int _total, int _piece){
2     int
3        file;
4    // open the file
5    if ((file = open(_file, O_RDONLY|FAT_RW_NOUPDATE, 0)) < 0)
6        return file;
7     unsigned char
8        *mem = malloc(_piece);

9     while (_total > 0){
10       PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE, 1);
11      // write piece
12      read(file, mem, _piece);
13      PERF_END(PERFORMANCE_COUNTER_0_BASE, 1);
14       _total -= _piece;
15    }
16   free(mem);
17   // close
18   close(file);
19   return 0;
20 }
21 // read test function
22 int read_test(int _blockSize){
23    int ret;

24    PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
25    PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);
26    ret = read_file("/mnt/root/test.bin", FILE_SIZE, _blockSize);
27    PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
28
29    return ret;
30 }
31 // write test function
32 int write_test(int _blockSize){
33    int ret;

34    PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
35    PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);
36    ret = write_file("/mnt/root/test.bin", FILE_SIZE, _blockSize);
37    PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
38
39    return ret;
40 }
41 // main test function
42 int main(){
43    int
44       n, i;
45    double
46       allTime[10], sdTime[10];
47    // init spi bus
48    if (spi_master_init_device(&spi) < 0)
49       return -1;
50   // create the mmc block device
51   if (mmc_spi_init_device(&mmcDev, &spi) < 0)
52      return -1;
53    // init partition driver
54    if (partition_init_device(&fs) < 0)
55       return -1;
56    // init fat driver
57    if (fat_init_device(&fat) < 0)
58       return -1;

59    for (n = 16; n <= 8192; n += n){
60       printf("%d\t", n);
61    }
62    printf("\n---Write performance Test with kbit/s result---");
63    printf("\n");
64    for (i=0,n = 16; n <= 8192; n += n){
65       if (write_test(n) < 0)
66          printf("ERROR\t");
67       allTime[i] = perf_get_section_time(PERFORMANCE_COUNTER_0_BASE,
68                         1)*1.0/60000000;
69       sdTime[i] = perf_get_section_time(PERFORMANCE_COUNTER_0_BASE,
70                         2)*1.0/60000000;
71       printf("%g\t", FILE_SIZE/allTime[i]*8/1024);
72       i++;
73    }
74    printf("\n---Write Test with processing-time [s] for FAT-driver---\n");
```

```
   for (i=0,n = 16; n <= 8192; n += n){
      printf("%g\t", allTime[i]-sdTime[i]);
      i++;
   }
   printf("\n---Write Test with processing-time [s] for SD card access---\n");
   for (i=0,n = 16; n <= 8192; n += n){
      printf("%g\t", sdTime[i]);
      i++;
   }
   printf("\n---Read performance Test with kbit/s result---\n");
   for (i=0,n = 16; n <= 8192; n += n){
      if (read_test(n) <0)
         printf("ERROR\t");
      allTime[i] = perf_get_section_time(PERFORMANCE_COUNTER_0_BASE,
                        1)*1.0/60000000;
      sdTime[i] = perf_get_section_time(PERFORMANCE_COUNTER_0_BASE,
                        2)*1.0/60000000;
      printf("%g\t", FILE_SIZE/allTime[i]*8/1024);
      i++;
   }
   printf("\n---Read Test with processing-time [s] for FAT-driver---\n");
   for (i=0,n = 16; n <= 8192; n += n){
      printf("%g\t", allTime[i]-sdTime[i]);
      i++;
   }
   printf("\n---Read Test with processing-time [s] for SD card access---\n");
   for (i=0,n = 16; n <= 8192; n += n){
      printf("%g\t", sdTime[i]);
      i++;
   }
   printf("\n");
   // unmount fat
   fat_exit_device(&fat);
   // unmound fs
   partition_exit_device(&fs);
  return 0;
}
```

Quelltext 6: Schreib-Lesegeschwindigkeitsmess-Programm

H Verilog-Quelltexte

H.1 SPI-Master Verilog-Quelltexte

H.1.1 Definitionsdatei

```
/*
*
* FIFO-buffered SPI Master
*
* State, register, etc. definitions
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
// main state machine
`define STATE_IDLE                  4'd0
`define STATE_START_TRANSFER        4'd1
`define STATE_LOAD_DATA             4'd2
`define STATE_TRANSFER_BITS         4'd3
`define STATE_STORE_DATA            4'd4
`define STATE_RECEIVE_BITS          4'd9
`define STATE_PRE_FINISHED2         4'd13
`define STATE_PRE_FINISHED3         4'd14
`define STATE_FINISHED              4'd15
`define STATE_REG_BITS              4
```

```
1
2  // register sizes
3  `define MAX_TRANSFER_BYTES_BITS       16
4  `define MAX_REF_CLKS_PER_BIT_BITS     8
5
6  // register addresses
7  `define REG_TRANSFER_FIFO             3'd0  // data register
8  `define REG_FIFO_FILL_COUNTS          3'd1  // fill counts
9  `define REG_TRANSFER_SIZE             3'd2  // transfer size, start job
10 `define REG_CONTROL                   3'd3  // clear fifo etc.
11 `define REG_STATUS                    3'd4  // status infos
12 `define REG_SCLK_DIVIDER              3'd5  // slave clock divider
13 `define REG_SS                        3'd6  // slave select
14 `define REG_DATA                      3'd7  // token register
15
16 `define REG_SIZE_BITS                 3
17
18 `define CNTRL_BIT_RECEIVE_CLEAR       31
19 `define CNTRL_BIT_RECEIVE_BUS_HOLD    30
20 `define CNTRL_BIT_TRANSFER_CLEAR      29
21 `define CNTRL_BIT_TRANSFER_BUS_HOLD   28
22 `define CNTRL_BIT_RESET               27
23 `define CNTRL_BIT_JOB_START           26
24 `define CNTRL_BIT_JOB_H               25
25 `define CNTRL_BIT_JOB_L               24
26 `define CNTRL_BIT_LSB_FIRST           23
27
28 `define STATUS_FLAG_TRANSFER_FULL     0
29 `define STATUS_FLAG_RECEIVE_EMPTY     1
30 `define STATUS_FLAG_BUSY              2
31 `define STATUS_FLAG_WRITE_ERROR       3
32 `define STATUS_FLAG_READ_ERROR        4
33 `define STATUS_FLAG_TRANSFER_EMPTY    5
34 `define STATUS_FLAG_RECEIVE_FULL      6
35
36 // default values
37 `define DEFAULT_SCLK_DIVIDER          50
38
39 `define JOB_TRANSFER                  2'd0
40 `define JOB_RECEIVE                   2'd1
41 `define JOB_WAIT_FOR_TOKEN            2'd2
42 `define JOB_WAIT_WHILE_TOKEN          2'd3
43
44 `define JOB_BITS                      2
```

Quelltext 7: SPI-Master Definitionsdatei (spi_master_defines.v)

H.1.2 Top-Modul

```
45 /*
46 *
47 * FIFO-buffered SPI Master
48 *
49 * Top module, for avalon registered slave access
50 *
51 * (c) 2008-2011 by Birger Zimmermann
52 *
53 */
54 `timescale 1ns/1ns
55 `include "spi_master_defines.v"
56
57 // ------------------------------------------------------------------------
58 // modul and port definition
59 // ------------------------------------------------------------------------
60 module spi_master_top(
61    input    wire  clk_i,                 // avalon clock
62    input    wire  reset_n_i,             // reset signal
63    input    wire  [2:0] as_address_i,    // address
64    input wire  as_read_i,                // read transfer signal
65    input    wire  [31:0] as_writedata_i, // write data (32-bit)
66    output   wire  [31:0] as_readdata_o,  // read data (32-bit)
67    input    wire  as_write_i,            // write transfer signal

68    input wire  begintransfer_i,          // begin transfer signal
```

```
    output   wire  as_waitrequest_o,       // signal a bus wait request
    input wire  [3:0] as_byteenable_i,     // not used

    output   wire  [7:0] spi_ss_o,         // slave select
    output   wire  sclk_o,                 // serial clock
    output   wire  mosi_o,                 // master out slave in
    input    wire  miso_i                  // master in slave out
);

// ------------------------------------------------------------------------
// parameter definitions
// ------------------------------------------------------------------------
parameter PRE_REQ_DELAY    = 2;
parameter S2M_DELAY        = 1; // slave to master signal delay (after clk_i)

// ------------------------------------------------------------------------
// register and wire definitions
// ------------------------------------------------------------------------
reg   [32-1:0] as_readdata;
reg   [`MAX_REF_CLKS_PER_BIT_BITS-1:0] sclk_divider_reg;
reg   [`MAX_TRANSFER_BYTES_BITS-1:0] transfer_count_reg;

reg   read_fifo_error;
reg   write_fifo_error;
reg   [7:0] spi_ss;
reg   [7:0] token;

reg   waitrequest_for_transfer_fifo_full;
reg   waitrequest_for_receive_fifo_empty;
reg   lsb_first;

reg   delayed_rc_rdreq;
reg   delayed_tr_wrreq;

//reg    [8-1:0] check_reg;

// strobes
wire  start_transfer_strobe;
wire  transfer_clear_strobe;
wire  receive_clear_strobe;
wire  tr_wrreq_strobe;
wire  rc_rdreq_strobe;
wire  reset_strobe;
wire  transfer_strobe;
wire  receive_strobe;
wire  [`JOB_BITS-1:0] job_strobe;

wire  read_error_strobe;
wire  write_error_strobe;
wire  state_idle;
wire  tr_full;
wire  [7:0] tr_wrusedw;
wire  [32-1:0] rc_read_data;
wire  [32-1:0] tr_write_data;
wire  rc_empty;
wire  [7:0] rc_rdusedw;
wire  tr_empty;
wire  rc_full;
wire  busy;

// ------------------------------------------------------------------------
// instance definition
// ------------------------------------------------------------------------
spi_master_transceiver transceiver_inst(
    .ref_clock_i(clk_i),
    .reset_n_i(reset_n_i),

    .sync_reset_i(reset_strobe),
    .token_i(token),
    .job_i(job_strobe),
    .start_transfer_i(start_transfer_strobe),

    .sclk_divider_i(sclk_divider_reg),
    .transfer_count_i( as_writedata_i[`MAX_TRANSFER_BYTES_BITS-1:0] ),
    .state_idle_o(state_idle),

    //.fifo_clk_i(clk_i),
```

```
    // transmit fifo signals
    .tr_sclr_i(transfer_clear_strobe),
    .tr_wrreq_i(tr_wrreq_strobe),
    .tr_write_data_i(tr_write_data),
    .tr_full_o(tr_full),
    .tr_empty_o(tr_empty),
    .tr_usedw_o(tr_wrusedw),

    // receive fifo signals
    .rc_sclr_i(receive_clear_strobe),
    .rc_rdreq_i(rc_rdreq_strobe),
    .rc_read_data_o(rc_read_data),
    .rc_empty_o(rc_empty),
    .rc_full_o(rc_full),
    .rc_usedw_o(rc_rdusedw),

    // spi signals
    .sclk_o(sclk_o),
    .mosi_o(mosi_o),
    .miso_i(miso_i)
);
// --------------------------------------------------------------------------
// continous assignments
// --------------------------------------------------------------------------
// address, access wires
wire  fifo_adr_and_rd   = (as_address_i == `REG_TRANSFER_FIFO?
                                    as_read_i: 1'b0);
wire  fifo_adr_and_wr   = (as_address_i == `REG_TRANSFER_FIFO?
                                    as_write_i: 1'b0);
wire  control_wr        = (as_address_i == `REG_CONTROL  ? as_write_i: 1'b0);
wire  size_wr           = (as_address_i == `REG_TRANSFER_SIZE?
                                    as_write_i: 1'b0);

// write request strobe for transmit fifo
assign tr_wrreq_strobe     = fifo_adr_and_wr & ~tr_full & ~delayed_tr_wrreq;
// write request did not happended last cycle and no wait cycle is requested
assign write_error_strobe  = fifo_adr_and_wr & ~delayed_tr_wrreq &
                                ~as_waitrequest_o;

// read request strobe for receive fifo
assign rc_rdreq_strobe     = fifo_adr_and_rd & ~rc_empty & ~delayed_rc_rdreq;
// read request did not happended last cycle and no wait cycle is requested
assign read_error_strobe   = fifo_adr_and_rd & ~delayed_rc_rdreq &
                                ~as_waitrequest_o;

// reset transceiver strobe
assign reset_strobe        = (control_wr & begintransfer_i ?
                                as_writedata_i[`CNTRL_BIT_RESET] : 1'b0);

// transfer fifo clear strobe
assign transfer_clear_strobe =
    ((control_wr & begintransfer_i) ? as_writedata_i[`CNTRL_BIT_TRANSFER_CLEAR]:1'b0)
    |
    ((size_wr & begintransfer_i) ? as_writedata_i[`CNTRL_BIT_TRANSFER_CLEAR]:1'b0)
    |
    reset_strobe;

// receive fifo clear strobe
assign receive_clear_strobe  =
    ((control_wr & begintransfer_i) ? as_writedata_i[`CNTRL_BIT_RECEIVE_CLEAR]:1'b0)
    |
    ((size_wr & begintransfer_i) ? as_writedata_i[`CNTRL_BIT_RECEIVE_CLEAR]:1'b0)
    |
    reset_strobe;

// start transfer strobe
assign start_transfer_strobe  = (size_wr & begintransfer_i ?
                                    as_writedata_i[`CNTRL_BIT_JOB_START]:1'b0);
assign job_strobe             = (size_wr & begintransfer_i ?
                                    {as_writedata_i[`CNTRL_BIT_JOB_H],
                                    as_writedata_i[`CNTRL_BIT_JOB_L]} : 2'd0);

// wait request strobe
assign as_waitrequest_o =
    (as_write_i & (
       (as_address_i != `REG_TRANSFER_FIFO ? begintransfer_i:1'b0)
    ))
```

```
   |
   (as_read_i & (
      (as_address_i != `REG_TRANSFER_FIFO ?
         begintransfer_i:1'b0)
   )
   |
      (fifo_adr_and_rd&(waitrequest_for_receive_fifo_empty?
          (~delayed_rc_rdreq):begintransfer_i))
   |
      (fifo_adr_and_wr&(waitrequest_for_transfer_fifo_full?
          (~delayed_tr_wrreq):begintransfer_i))
   );
// slave select
assign spi_ss_o = ~spi_ss;
// read data
assign as_readdata_o =
   (
   (as_address_i != `REG_TRANSFER_FIFO ? as_readdata: 0)
   |
   (as_address_i == `REG_TRANSFER_FIFO ?
   (
      lsb_first? (
      {
         {rc_read_data[7:0], rc_read_data[15:8]},
         {rc_read_data[23:16], rc_read_data[31:24]}
      }
      ): rc_read_data
   )

    : 0)
   );
// write data
assign tr_write_data = lsb_first? (
   {
      {as_writedata_i[7:0],
         as_writedata_i[15:8]},
         {as_writedata_i[23:16], as_writedata_i[31:24]}
   }
   ): as_writedata_i;

// busy bit
assign busy = ~state_idle;

// ------------------------------------------------------------------------
// read task
// ------------------------------------------------------------------------
task read_task();
begin
   // first cycle of transfer
   if (begintransfer_i) begin

      case (as_address_i) // synopsis parallel_case
         `REG_FIFO_FILL_COUNTS: begin
            as_readdata       <= #S2M_DELAY {{{(14-8){1'b0}},
               {rc_rdusedw, 2'b0}}, {{(14-8){1'b0}}, {tr_wrusedw, 2'b0}}};
         end
         `REG_TRANSFER_SIZE:  begin
            as_readdata       <= #S2M_DELAY {{(32-
                                  `MAX_TRANSFER_BYTES_BITS){1'b0}},
                                  transfer_count_reg};
         end
         `REG_CONTROL: begin
            as_readdata                          <= #S2M_DELAY 0;
            as_readdata[`CNTRL_BIT_TRANSFER_BUS_HOLD]    <=
                     #S2M_DELAY waitrequest_for_transfer_fifo_full;
            as_readdata[`CNTRL_BIT_RECEIVE_BUS_HOLD]     <=
                     #S2M_DELAY waitrequest_for_receive_fifo_empty;
            as_readdata[`CNTRL_BIT_LSB_FIRST]               <=
                     #S2M_DELAY lsb_first;
         end
         `REG_STATUS: begin
            as_readdata       <= #S2M_DELAY 0;
            //as_readdata[4:0]   <=
               #S2M_DELAY {{read_fifo_error, write_fifo_error},
               {busy, {rc_empty, tr_full}}};

            as_readdata[`STATUS_FLAG_TRANSFER_FULL]      <= #S2M_DELAY tr_full;
            as_readdata[`STATUS_FLAG_RECEIVE_EMPTY]      <= #S2M_DELAY rc_empty;
```

```
            as_readdata[`STATUS_FLAG_BUSY]                  <= #S2M_DELAY busy;
            as_readdata[`STATUS_FLAG_WRITE_ERROR]           <=
                    #S2M_DELAY write_fifo_error;
            as_readdata[`STATUS_FLAG_READ_ERROR]            <=
                    #S2M_DELAY read_fifo_error;
            as_readdata[`STATUS_FLAG_TRANSFER_EMPTY]    <= #S2M_DELAY tr_empty;
            as_readdata[`STATUS_FLAG_RECEIVE_FULL]      <= #S2M_DELAY rc_full;

            // reset the error flags
            write_fifo_error  <= #S2M_DELAY 1'b0;
            read_fifo_error     <= #S2M_DELAY 1'b0;

            // set the debug check register
            //as_readdata[31:24] <= #S2M_DELAY check_reg;
            //check_reg       <= #S2M_DELAY 8'd0;
         end
         `REG_SCLK_DIVIDER: begin
            as_readdata       <=
                    #S2M_DELAY {{(32-`MAX_REF_CLKS_PER_BIT_BITS){1'b0}},
                    sclk_divider_reg};
         end
         `REG_SS: begin
            as_readdata       <= #S2M_DELAY {{(32-8){1'b0}}, spi_ss};
         end
         `REG_DATA: begin
            as_readdata       <= #S2M_DELAY {{(32-8){1'b0}}, token};
         end
         default: begin
            as_readdata       <= #S2M_DELAY {{(32-`REG_SIZE_BITS){1'b0}},
                                    as_address_i};
         end
      endcase
   end
end
endtask

// ------------------------------------------------------------------------
// write task
// ------------------------------------------------------------------------
task write_task();
begin
   // first cycle of transfer
   if (begintransfer_i) begin
      case (as_address_i) // synopsis parallel_case
         `REG_FIFO_FILL_COUNTS: begin
         end
         `REG_TRANSFER_SIZE: begin
            transfer_count_reg                    <=
                  as_writedata_i[`MAX_TRANSFER_BYTES_BITS-1:0];
            waitrequest_for_transfer_fifo_full  <=
                  as_writedata_i[`CNTRL_BIT_TRANSFER_BUS_HOLD];
            waitrequest_for_receive_fifo_empty  <=
                  as_writedata_i[`CNTRL_BIT_RECEIVE_BUS_HOLD];
            lsb_first                                 <=
                  as_writedata_i[`CNTRL_BIT_LSB_FIRST];
         end
         `REG_CONTROL: begin
            waitrequest_for_transfer_fifo_full  <=
                  as_writedata_i[`CNTRL_BIT_TRANSFER_BUS_HOLD];
            waitrequest_for_receive_fifo_empty  <=
                  as_writedata_i[`CNTRL_BIT_RECEIVE_BUS_HOLD];
            lsb_first                     <= as_writedata_i[`CNTRL_BIT_LSB_FIRST];
         end
         `REG_SCLK_DIVIDER: begin
            sclk_divider_reg              <=
                  as_writedata_i[`MAX_REF_CLKS_PER_BIT_BITS-1:0];
         end
         `REG_SS: begin
            spi_ss                        <= as_writedata_i[8-1:0];
         end
         `REG_DATA: begin
            token                         <= as_writedata_i[8-1:0];
         end
         default:
            begin
            end
      endcase
   end
```

```
end
endtask

// -----------------------------------------------------------------------
// reset task
// -----------------------------------------------------------------------
task reset_task();
begin
   sclk_divider_reg                  <= `DEFAULT_SCLK_DIVIDER;
   transfer_count_reg                <= 0;

   spi_ss                            <= 0;
   waitrequest_for_transfer_fifo_full <= 1'b0;
   waitrequest_for_receive_fifo_empty <= 1'b0;

   delayed_rc_rdreq                  <= 1'b0;
   delayed_tr_wrreq                  <= 1'b0;

   read_fifo_error                   <= 1'b0;
   write_fifo_error                  <= 1'b0;
   token                             <= 8'd0;

   //    check_reg                   <= 8'd0;

   // default little endian style
   lsb_first                         <= 1'b0;
end
endtask

// -----------------------------------------------------------------------
// clock dependent assignments
// -----------------------------------------------------------------------
always @(posedge clk_i, negedge reset_n_i) begin
   if (!reset_n_i) begin
      // asynchronous reset
      reset_task();
   end
   else
   // positive clock edge?
   if (clk_i) begin
      // read error check
      if (read_error_strobe) begin
         read_fifo_error      <= 1'b1;
      end

      // write error check
      if (write_error_strobe) begin
         write_fifo_error     <= 1'b1;
      end

//    if (tr_wrreq_strobe)
//       check_reg <= check_reg + 1'b1;

      // did a read from the fifo happend this cycle
      delayed_rc_rdreq       <= #PRE_REQ_DELAY rc_rdreq_strobe;
      // did a write to the fifo happend this cycle
      delayed_tr_wrreq       <= #PRE_REQ_DELAY tr_wrreq_strobe;

      // avalon read request ?
      if (as_read_i) begin
         read_task();
      end
      else
      // avalon write request ?
      if (as_write_i) begin
         write_task();
      end
   end
end

endmodule
```

Quelltext 8: SPI-Master Top-Modul (spi_master_top.v)

H.1.3 Transceiver

```
/*
*
* FIFO-buffered SPI Master
*
* Transceiver module
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
`timescale 1ns/1ns
`include "spi_master_defines.v"

// ------------------------------------------------------------------------
// modul and port definition
// ------------------------------------------------------------------------
module spi_master_transceiver(
   // clock signals
   input wire ref_clock_i,
   input wire reset_n_i,

   // synchronous reset
   input wire sync_reset_i,

   // slave clock divider
   input wire [`MAX_REF_CLKS_PER_BIT_BITS-1:0] sclk_divider_i,
   // which job (opcode) should be started
   input wire [`JOB_BITS-1:0] job_i,
   input wire [8-1:0] token_i,
   input wire start_transfer_i,
   input wire [`MAX_TRANSFER_BYTES_BITS-1:0] transfer_count_i,
   output wire state_idle_o,

   // transmit fifo signals
   input wire  tr_sclr_i,
   input wire  tr_wrreq_i,
   input wire  [31:0]tr_write_data_i,
   output wire tr_full_o,
   output wire tr_empty_o,
   output wire [7:0]tr_usedw_o,

   // receive fifo signals
   input wire  rc_sclr_i,
   input wire  rc_rdreq_i,
   output wire [31:0]rc_read_data_o,
   output wire rc_full_o,
   output wire rc_empty_o,
   output wire [7:0]rc_usedw_o,

   // spi signals
   output reg  sclk_o,
   output reg  mosi_o,
   input wire  miso_i
);
// ------------------------------------------------------------------------
// parameter defitions
// ------------------------------------------------------------------------
parameter RECEIVE_DUMMY      = 8'hff;
parameter MOSI_IDLE_STATE    = 1'b0;
parameter SCLK_IDLE_STATE    = 1'b0;

parameter STATE_CHANGE_DELAY  = 2;
parameter COUNTER_CHANGE_DELAY   = 2;

// ------------------------------------------------------------------------
// register and wire definitions
// ------------------------------------------------------------------------
reg            [5-1:0] bit_num;                      // bit number register
reg               [`STATE_REG_BITS-1:0] state;             // state register

reg               [`MAX_TRANSFER_BYTES_BITS-1:0] transfer_count;   // transfer size register

reg               [`JOB_BITS-1:0] job;                  // operation register
reg               [8-1:0] token;                        // data token register
reg               [8-1:0] rc_token;                     // receive token register
reg            [`MAX_REF_CLKS_PER_BIT_BITS-1:0] bit_trigger_divider;
reg            [`MAX_REF_CLKS_PER_BIT_BITS-1:0] bit_trigger_divider_compare;
```

```
1
2  // fifo connections
3  wire           rc_wrreq_strobe;
4  wire           tr_rdreq_strobe;
5  reg               [32-1:0] rc_write_data;
6  wire           [32-1:0] tr_read_data;
7  wire           [7:0] rc_usedw;
8  wire           [7:0] tr_usedw;
9  // ---------------------------------------------------------------------------
10 // instance definitions
11 // ---------------------------------------------------------------------------
12 spi_master_transfer_fifo transfer(
13    .aclr(~reset_n_i),
14    .clock(ref_clock_i),
15    .sclr(tr_sclr_i),
16    .wrreq(tr_wrreq_i),
17    .full(tr_full_o),
18    .usedw(tr_usedw),
19    .data(tr_write_data_i),
20
21    .rdreq(tr_rdreq_strobe),
22    .empty(tr_empty_o),
23    .q(tr_read_data)
24 );
25
26 spi_master_receive_fifo receive(
27    .aclr(~reset_n_i),
28    .clock(ref_clock_i),
29    .sclr(rc_sclr_i),
30    .wrreq(rc_wrreq_strobe),
31    .full(rc_full_o),
32    .usedw(rc_usedw),
33    .data(rc_write_data),
34
35    .rdreq(rc_rdreq_i),
36    .empty(rc_empty_o),
37    .q(rc_read_data_o)
38    );
39
40 // ---------------------------------------------------------------------------
41 // continous assignments
42 // ---------------------------------------------------------------------------
43 assign state_idle_o = (state == `STATE_IDLE) ? 1'b1: 1'b0;
44
45 assign rc_wrreq_strobe = (state == `STATE_STORE_DATA ? ~rc_full_o: 1'b0);
46
47 assign tr_rdreq_strobe = (state == `STATE_LOAD_DATA ? ~tr_empty_o: 1'b0);
48
49
50 assign rc_usedw_o = {rc_full_o, rc_usedw[6:0]};
51
52 assign tr_usedw_o = {tr_full_o, tr_usedw[6:0]};
53
54 // ---------------------------------------------------------------------------
55 // init task
56 // ---------------------------------------------------------------------------
57 task transfer_init_task();
58 begin
59    $display("state is in transfer_init_task");

60    transfer_count            <= #COUNTER_CHANGE_DELAY transfer_count_i;

61    // setup the sclk divider
62    bit_trigger_divider_compare <= sclk_divider_i-1'b1;

63    // setup bit divider
64    bit_trigger_divider     <= 0;

65    //rc_wrreq              <= #WRREQ_DELAY 1'b0;
66
67    // store which job we want to do
68    job                  <= job_i;
69    // store the token
70    token                <= token_i;

71    // start with bit 0, for shift out it is inverted
72    bit_num                 <= 5'd0;
73    case (job_i)
```

```
      `JOB_TRANSFER: begin
         $display("start transfer job");
         $display("transfer bytes:%d", transfer_count_i);

         // next state -> load data
         state          <= #STATE_CHANGE_DELAY `STATE_LOAD_DATA;

      end
      `JOB_RECEIVE: begin
         $display("start receive job");
         $display("receive bytes:%d", transfer_count_i);

         // set data in transfer register
         //transfer_data   <= {4{RECEIVE_DUMMY}};

         // clock low
         sclk_o         <= 1'b0;

         // msb first
         mosi_o         <= RECEIVE_DUMMY[3'd7];

         // go to receive state
         state          <= #STATE_CHANGE_DELAY `STATE_RECEIVE_BITS;
      end
      `JOB_WAIT_FOR_TOKEN: begin
         $display("start wait for token: %x job", token_i);
         $display("max wait for bytes:%d", transfer_count_i);

         // set data in transfer register
         //transfer_data      <= {4{RECEIVE_DUMMY}};

         // clock low
         sclk_o         <= 1'b0;

         // msb first
         mosi_o         <= RECEIVE_DUMMY[3'd7];

         // go to receive state
         state          <= #STATE_CHANGE_DELAY `STATE_RECEIVE_BITS;
      end
      `JOB_WAIT_WHILE_TOKEN: begin
         $display("start wait while token: %x job", token_i);
         $display("max wait for bytes:%d", transfer_count_i);
         // set data in transfer register
         //transfer_data      <= {4{RECEIVE_DUMMY}};

         // clock low
         sclk_o         <= 1'b0;

         // msb first
         mosi_o         <= RECEIVE_DUMMY[3'd7];

         // go to receive state
         state          <= #STATE_CHANGE_DELAY `STATE_RECEIVE_BITS;
      end
   endcase
end
endtask

// -----------------------------------------------------------------------------
// reset task
// -----------------------------------------------------------------------------
task reset_task();
begin
   $display("reset");

   // init registers
   transfer_count            <= #COUNTER_CHANGE_DELAY 1'b0;

   bit_trigger_divider       <= 1'b0;
   bit_trigger_divider_compare <= 1'b0;

   bit_num                <= 5'd0;

   state                  <= #STATE_CHANGE_DELAY `STATE_IDLE;

   job                    <= 0;
   token                <= 0;
```

```
end
endtask

// -----------------------------------------------------------------------------
// state machine
// -----------------------------------------------------------------------------
always @(posedge ref_clock_i, negedge reset_n_i) begin
   if (!reset_n_i) begin
      reset_task();
   end
   else begin
      if (sync_reset_i) begin
         reset_task();
      end
      else
      case(state)
         // -----------------------------------------------------------------
         `STATE_IDLE: begin
            $display("in state STATE_IDLE");
            sclk_o            <= SCLK_IDLE_STATE;
            mosi_o            <= MOSI_IDLE_STATE;

            if (start_transfer_i) begin
               transfer_init_task();
            end
         end

         // -----------------------------------------------------------------
         `STATE_START_TRANSFER: begin
            $display("data loaded %x", tr_read_data);
            // set clock low
            sclk_o         <= 1'b0;
            // start with bit 0, for shift out it is inverted
            bit_num        <= 5'd0;
            // msb first
            mosi_o         <= tr_read_data[5'd31];
            // data loaded, goto start transfer of bytes
            state          <= #STATE_CHANGE_DELAY `STATE_TRANSFER_BITS;
         end
         // -----------------------------------------------------------------
         `STATE_LOAD_DATA: begin
            $display("state is in STATE_LOAD_DATA transfer_count: %d, strobe: %d",
               transfer_count, tr_rdreq_strobe);

            if (transfer_count > 0) begin
               if (~tr_empty_o) begin
                  // data loaded, goto start transfer of bytes
                  state       <= #STATE_CHANGE_DELAY `STATE_START_TRANSFER;
               end
            end
            else begin
               // error must not be here normally
               $display("error in wrong state STATE_LOAD_DATA with no transfer count");
               state          <= #STATE_CHANGE_DELAY `STATE_FINISHED;
            end
         end
         // -----------------------------------------------------------------
         `STATE_TRANSFER_BITS: begin
            // generate sclk
            if (bit_trigger_divider >= bit_trigger_divider_compare) begin

               // toggle clock
               sclk_o <= ~sclk_o;

               // falling edge of sclk
               if (sclk_o) begin
                  //$display("falling edge of sclk in bit: %d", bit_num);
                  // increase bit num
                  bit_num  <= bit_num+5'd1;

                  // last bit of byte
                  if (bit_num[2:0] == 3'd7) begin
                     // one byte transferred
                     $display("one byte transferred");

                     // decrease transfer_count
                     transfer_count             <= #COUNTER_CHANGE_DELAY transfer_count-1'b1;
```

```
                  // was it the last byte we want to transfer ?
                  if (transfer_count == 1) begin
                     // so we are finished up
                     state               <= #STATE_CHANGE_DELAY `STATE_FINISHED;
                     mosi_o              <= MOSI_IDLE_STATE;
                  end
                  else begin
                     // we need want more...

                     // last bit of data
                     if (bit_num == 5'd31) begin
                        $display("next state load new data");
                        // last bit of byte transferred go to byte load state
                        state            <= #STATE_CHANGE_DELAY `STATE_LOAD_DATA;

                        // set to one because we do an data load state step now
                        bit_trigger_divider <= 1'b1;
                     end
                     else begin
                        // we have more bytes in this word
                        // setup next shift out bit
                        mosi_o           <= tr_read_data[5'd31-bit_num-1'b1];
                        bit_trigger_divider <= 1'b0;
                     end
                  end

               end
               else begin
                  // setup next shift out bit
                  mosi_o           <= tr_read_data[5'd31-bit_num-1'b1];
                  // reset to zero
                  bit_trigger_divider <= 0;
               end
            end
            // rising edge of sclk
            else begin
               //$display("rising edge of sclk bit_num: %d", bit_num);
               // reset to zero
               bit_trigger_divider  <= 0;
            end
         end
         else begin
            // increase divider
            bit_trigger_divider     <= bit_trigger_divider+1'b1;
         end
      end
      // -----------------------------------------------------------------------
      `STATE_STORE_DATA: begin
         if (~rc_full_o) begin
            // data stored
            if (transfer_count > 0) begin
               // do we can do more ?
               case (job)
                  `JOB_RECEIVE: begin
                     //rc_write_data   <= 0;
                     state          <= #STATE_CHANGE_DELAY `STATE_RECEIVE_BITS;
                  end
                  default: begin
                     // last bit of byte transferred
                     state          <= #STATE_CHANGE_DELAY `STATE_PRE_FINISHED2;
                  end
               endcase
            end
            else begin
               state               <= #STATE_CHANGE_DELAY `STATE_PRE_FINISHED2;
            end
         end
         else begin
            $display("wait for space in receive fifo");
         end

      end
      // -----------------------------------------------------------------------
      `STATE_RECEIVE_BITS: begin
         // generate sclk
         if (bit_trigger_divider >= bit_trigger_divider_compare) begin
```

```
1                  // toggle clock
2                  sclk_o <= ~sclk_o;

3                  // falling edge of sclk
4                  if (sclk_o) begin
5                     bit_num             <= bit_num+1'b1;

6                     // is one byte transferred ?
7                     if (bit_num[2:0] == 3'd7) begin
8                        // one byte received
9                        $display("one byte received");

10                       // decrease transfer_count
11                       transfer_count      <= #COUNTER_CHANGE_DELAY transfer_count-1'b1;

12                       case (job)
13                          `JOB_RECEIVE: begin
14                             // last bit of data
15                             if (bit_num == 5'd31) begin
16                                $display("next state store received data");
17                                // last bit of byte transferred go to byte load state
18                                state               <= #STATE_CHANGE_DELAY `STATE_STORE_DATA;
19                             end
20                             else begin
21                                // how ever the last byte was received
22                                // so we need to store what we have
23                                if (transfer_count == 1) begin
24                                   // last bit of byte transferred go to byte load state

25                                   state            <= #STATE_CHANGE_DELAY `STATE_STORE_DATA;
26                                end
27                             end
28                          end
29                          `JOB_WAIT_FOR_TOKEN: begin
30                             if (token == rc_token) begin
31                                $display("JOB_WAIT_FOR_TOKEN succeed");
32
33                                rc_write_data[31:24] <= rc_token;
34                                rc_write_data[23:0]     <= 0;
35                                state               <= #STATE_CHANGE_DELAY `STATE_STORE_DATA;
36                             end
37                             else
38                             if (transfer_count == 1) begin
39                                $display("JOB_WAIT_FOR_TOKEN failed");
40                                // last byte received was not the requested one so timeout occurs
41                                state               <= #STATE_CHANGE_DELAY `STATE_PRE_FINISHED2;
42                             end
43
44                          end
45                          `JOB_WAIT_WHILE_TOKEN: begin
46                             if (token != rc_token) begin
47
48                                rc_write_data[31:24] <= rc_token;
49                                rc_write_data[23:0]     <= 0;
50                                state               <= #STATE_CHANGE_DELAY `STATE_STORE_DATA;
51                             end
52                             else
53                             if (transfer_count == 1) begin
54                                $display("JOB_WAIT_WHILE_TOKEN failed");
55                                // last byte received was not the requested one so timeout occurs
56                                state               <= #STATE_CHANGE_DELAY `STATE_PRE_FINISHED2;
57                             end
58                          end

59                       endcase

60                       // restart bit divider
61                       bit_trigger_divider    <= 1'b0;
62                    end
63                    else begin
64                       // reset to zero
65                       bit_trigger_divider    <= 1'b0;
66                    end
67                 end
68                 // rising edge of sclk
69                 else begin
70                    //$display("in state STATE_RECEIVE_BITS posedge sclk, bitnum: %d", bit_num);
```

```
                  if (bit_num == 0) begin
                     // set all bits of other 3 bytes to zero
                     rc_write_data[23:0]     <= 0;
                  end

                  // read bit
                  rc_write_data[5'd31-bit_num]  <= miso_i;
                  rc_token[3'd7-bit_num[2:0]]   <= miso_i;

                  // reset to zero
                  bit_trigger_divider       <= 0;
               end

            end
            else begin
               // increase divider
               bit_trigger_divider <= bit_trigger_divider+1'b1;
            end

         end

         `STATE_PRE_FINISHED2: begin
            $display("in state STATE_PRE_FINISHED2");

            // just put an intermediate cycle for the rdusdw to be setup right at finish
            state       <= #STATE_CHANGE_DELAY  `STATE_PRE_FINISHED3;
         end
         `STATE_PRE_FINISHED3: begin
            $display("in state STATE_PRE_FINISHED3");

            // just put an intermediate cycle for the rdusdw to be setup right at finish
            state       <= #STATE_CHANGE_DELAY  `STATE_FINISHED;
         end
         `STATE_FINISHED: begin
            $display("in state STATE_FINISHED");
            sclk_o      <= SCLK_IDLE_STATE;
            mosi_o      <= MOSI_IDLE_STATE;
            state       <= #STATE_CHANGE_DELAY  `STATE_IDLE;
         end
         default: begin
            state       <= #STATE_CHANGE_DELAY  `STATE_IDLE;
         end
      endcase
   end
end

endmodule
```

Quelltext 9: SPI-Master Transceiver (spi_master_transceiver.v)

H.1.4 Testbench

```
/*
*
* FIFO-buffered SPI Master
*
* Testbench for the transceiver
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
`timescale 1ns/1ns
`include "spi_master_defines.v"

// ------------------------------------------------------------------------
// modul definition
// ------------------------------------------------------------------------
module spi_master_tb();
// ------------------------------------------------------------------------
// register and wire definitions
// ------------------------------------------------------------------------
reg ref_clk;
reg   reset_n;
```

```
1
2  wire state_idle;
3  wire [7:0] tr_usedw;
4  wire [7:0] rc_usedw;
5  wire rc_empty;
6  wire mosi;
7  wire miso;
8  wire sclk;
9
10 reg [`MAX_TRANSFER_BYTES_BITS-1:0]transfer_count;
11 //reg [`MAX_TRANSFER_BYTES_BITS-1:0]receive_count;
12 reg start_transfer;
13 reg [`JOB_BITS-1:0] job;
14 reg [8-1:0] token;
15
16 reg tr_wrreq;
17 reg [32-1:0] tr_write_data;
18
19 reg rc_rdreq;
20 wire [32-1:0] rc_read_data;
21 reg [31-1:0] sclk_counter;
22
23 reg rc_sclr;
24 wire busy;
25 // ------------------------------------------------------------------------
26 // instance definition
27 // ------------------------------------------------------------------------
28 spi_master_transceiver transceiver(
29    .ref_clock_i(ref_clk),
30    .reset_n_i(reset_n),
31
32    .sclk_divider_i(4),
33    .transfer_count_i(transfer_count),
34    .token_i(token),
35    .job_i(job),
36    .start_transfer_i(start_transfer),
37    .state_idle_o(state_idle),
38
39    //.fifo_clk_i(ref_clk),
40
41    .tr_sclr_i(1'b0),
42    .tr_wrreq_i(tr_wrreq),
43    .tr_write_data_i(tr_write_data),
44    .tr_usedw_o(tr_usedw),
45
46    .rc_sclr_i(rc_sclr),
47    .rc_usedw_o(rc_usedw),
48    .rc_rdreq_i(rc_rdreq),
49    .rc_read_data_o(rc_read_data),
50    .rc_empty_o(rc_empty),
51    .sclk_o(sclk),
52    .mosi_o(mosi),
53    .miso_i(miso)
54 );
55 // ------------------------------------------------------------------------
56 // continous assignments
57 // ------------------------------------------------------------------------
58 assign busy = ~state_idle;
59
60 assign miso = ((sclk_counter >= 0*8 && sclk_counter < 1*8) ? 1'b1: 1'b0)
61 |
62  ((sclk_counter >= 2*8 && sclk_counter < 3*8) ? 1'b1: 1'b0)
63 |
64  ((sclk_counter >= 7*8-1 && sclk_counter < 7*8) ? 1'b1: 1'b0)
65 |
66  ((sclk_counter >= 14*8 && sclk_counter < 15*8) ? 1'b1: 1'b0)
67 ;
68
69 // ------------------------------------------------------------------------
70 // clock generation
71 // ------------------------------------------------------------------------
72 always #5 ref_clk = ~ref_clk;
73
74 always @(posedge sclk) begin
75    sclk_counter <= #5 sclk_counter + 1'b1;
76 end
77
78 // ------------------------------------------------------------------------
```

```
1  // write data to fifo sub routine
2  // ------------------------------------------------------------------------
3  task write_fifo(input wire [32-1:0] _data);
4  begin
5     @(negedge ref_clk);
6     tr_write_data = _data;
7     tr_wrreq = 1'b1;
8     @(posedge ref_clk);
9     #1 tr_wrreq = 1'b0;
10 end
11 endtask
12
13 // ------------------------------------------------------------------------
14 // start transfer
15 // ------------------------------------------------------------------------
16 task start_transfer_task(
17    input wire[`MAX_TRANSFER_BYTES_BITS-1:0]_transfer);
18 begin
19    wait(~busy);
20    @(negedge ref_clk);
21    // setup transfer count
22    transfer_count          = _transfer;
23    job                     = `JOB_TRANSFER;
24    start_transfer          = 1'b1;
25    @(posedge ref_clk);
26    #1 start_transfer       = 1'b0;
27    //@(negedge ref_clk);
28    wait(busy);
29 end
30 endtask
31
32 // ------------------------------------------------------------------------
33 // start receive
34 // ------------------------------------------------------------------------
35 task start_receive_task(
36    input wire [`MAX_TRANSFER_BYTES_BITS-1:0]_transfer);
37 begin
38    wait(~busy);
39    @(negedge ref_clk);
40    // setup transfer count
41    transfer_count          = _transfer;
42    job                     = `JOB_RECEIVE;
43    start_transfer          = 1'b1;
44    @(posedge ref_clk);
45    #1 start_transfer       = 1'b0;
46    //@(negedge ref_clk);
47    wait(busy);
48 end
49 endtask
50 // ------------------------------------------------------------------------
51 // start wait for token
52 // ------------------------------------------------------------------------
53 task start_wait_for_token_task(
54    input wire [`MAX_TRANSFER_BYTES_BITS-1:0]_transfer,
55    input wire [8-1:0] _token);
56 begin
57    wait(~busy);
58    @(negedge ref_clk);
59    // setup transfer count
60    transfer_count          = _transfer;
61    job                     = `JOB_WAIT_FOR_TOKEN;
62    token                   = _token;
63    start_transfer          = 1'b1;
64    rc_sclr                 = 1'b1;
65    @(posedge ref_clk);
66    #1 start_transfer       = 1'b0;
67    rc_sclr                 = 1'b0;
68    @(negedge ref_clk);

69    wait(~busy);
70
71    if (rc_usedw > 0) begin
72       $display("token found within %d cycles", _transfer);
73       display_received_task(1);
74    end
75    else
76       $display("token was NOT found within %d cycles", _transfer);
77 end
```

```
1  endtask
2
3  // ---------------------------------------------------------------------------
4  // start wait while token
5  // ---------------------------------------------------------------------------
6  task start_wait_while_token_task(
7     input wire [`MAX_TRANSFER_BYTES_BITS-1:0]_transfer,
8     input wire [8-1:0] _token);
9  begin
10    wait(~busy);
11    @(negedge ref_clk);
12    // setup transfer count
13    transfer_count          = _transfer;
14    job                     = `JOB_WAIT_WHILE_TOKEN;
15    token                   = _token;
16    start_transfer          = 1'b1;
17    rc_sclr                 = 1'b1;
18    @(posedge ref_clk);
19    #1 start_transfer       = 1'b0;
20    rc_sclr                 = 1'b0;
21    @(negedge ref_clk);
22    wait(~busy);
23
24    if (rc_usedw > 0) begin
25       $display("token found within %d cycles", _transfer);
26       display_received_task(1);
27    end
28    else
29       $display("token was NOT found within %d cycles", _transfer);
30 end
31 endtask
```

```
1  // ----------------------------------------------------------------------------
2  // display received data
3  // ----------------------------------------------------------------------------
4  task display_received_task(
5     input integer _num);
6  integer i;
7  begin
8     $display("start receiving of %d bytes", _num);
9     for(i=0; i < (_num+3)/4; i=i+1) begin
10       wait (~rc_empty);
11       @(negedge ref_clk);
12       rc_rdreq <= 1'b1;
13       @(posedge ref_clk);
14       #1 rc_rdreq <= 1'b0;
15       @(negedge ref_clk);
16       $display("read data [%d]: %08x", i*4,  rc_read_data);
17    end
18 end
19 endtask
20
21 // ----------------------------------------------------------------------------
22 // main test program
23 // ----------------------------------------------------------------------------
24 initial begin
25    // init variables
26    sclk_counter      = 0;
27    start_transfer = 1'b0;
28    rc_sclr        = 1'b0;
29    ref_clk     = 1'b0;
30    tr_wrreq       = 1'b0;
31    token          = 0;
32    rc_rdreq       = 0;

33    // perform reset
34    reset_n        = 1'b1;
35    #1 reset_n     = 1'b0;
36    #1 reset_n     = 1'b1;
37
38    // transfer 2 bytes
39    start_transfer_task(2);
40    write_fifo(32'h04030201);

41    // transfer 6 bytes
42    start_transfer_task(6);
43    write_fifo(32'h01020304);
44    write_fifo(32'h05060000);

45    // receive 10 bytes
46    start_receive_task(10);
47    display_received_task(10);

48    // wait for token 1
49    start_wait_for_token_task(10, 8'd1);

50    // wait for token 0
51    start_wait_while_token_task(10, 8'd0);

52    wait(~busy);
53    $display("testbench successfully finished!");
54    $stop;
55    $finish;
56 end
57 endmodule
```

Quelltext 10: SPI-Master Testbench (spi_master_tb.v)

H.2 WM8731 IP-Core Verilog-Quelltexte

H.2.1 Definitionsdatei

```
/*
*
* WM8731 Audio-CODEC core, support different sample rates and runtime configuration
*
* Constant definitions
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
`define PACKET_SIZE                   12 // sizeof sample packets
`define CHANNEL_NUM                   2  // number of channels

`define MAX_BYTES_PER_SAMPLE          3
`define MAX_BYTES_PER_SAMPLE_BITS     2//log2(`MAX_BYTES_PER_SAMPLE)

`define MAX_REF_CLKS_PER_FRAME        2304  // 18384Khz/8Khz
//log2(`MAX_REF_CLKS_PER_FRAME)
`define MAX_REF_CLKS_PER_FRAME_BITS   12

`define MAX_BIT_CLKS_PER_FRAME_BITS   8
`define  FIFO_DATA_OUT_BITS           8
`define  FIFO_DATA_IN_BITS            32

`define BCLK_BYTES_PER_SAMPLE         4
`define BCLK_BYTES_PER_SAMPLE_BITS    2
//log2(`MAX_REF_CLKS_PER_FRAME/(`BCLK_BYTES_PER_SAMPLE*8*`CHANNEL_NUM))
`define  MAX_REF_CLKS_DIVIDER_BITS    7
//log2(`MAX_REF_CLKS_PER_FRAME/(`BCLK_BYTES_PER_SAMPLE*`CHANNEL_NUM))
`define  MAX_LATCH_CLKS_DIVIDER_BITS  10

// defaults
`define DEFAULT_REF_CLK_DIVIDER       384
`define DEFAULT_BYTES_PER_SAMPLE      2

// registers
`define CLOCK_DIVIDER_REG             3'd0
`define DACLR_DIVIDER_REG             3'd1
`define ADCLR_DIVIDER_REG             3'd2
`define BYTES_PER_SAMPLE_REG          3'd3
`define TRANSMIT_FIFO_FILL_COUNT_REG  3'd4
`define RECEIVE_FIFO_FILL_COUNT_REG   3'd5
`define CLOCK_SELECT_REG              3'd6
`define CONTROL_REG                   3'd7
`define STATUS_REG                    3'd7
// fifo size
`define FIFO_BYTE_SIZE                512

// divider register channel enable bits
`define CHANNEL_L_BIT                 16
`define CHANNEL_R_BIT                 17

// control register job bits
`define TRANSCEIVER_DO_JOB_BIT        31
`define TRANSCEIVER_RECONFIGURE_BIT   30

// clock control bits in clock select register
`define CLOCK_SELECT_BIT              0

// state machine states
`define WM8731_STATE_IDLE             2'd0
`define WM8731_STATE_ON_EVEN_IDLE     2'd1
//`define WM8731_STATE_PRE_IDLE       2'd2
`define  WM8731_STATE_RUNNING         2'd3
`define WM8731_STATE_BITS             2
```

Quelltext 11: WM8731 IP-Core Definitionsdatei (wm8731_defines.v)

H.2.2 Top-Modul

```
/*
*
* WM8731 Audio-CODEC core, support different sample rates and runtime configuration
*
* Top module, for avalon registered slave- and DMA based streaming access
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
`timescale 1ns/1ns
`include "wm8731_defines.v"

// --------------------------------------------------------------------------
// modul and port definition
// --------------------------------------------------------------------------
module wm8731_top(
   // avalon clock interface
   input    wire  clk_i,                    // avalon clock
   input    wire  reset_n_i,                // asynchronous reset

   // avalon streaming interface transmitter (sink)
   output   wire  transmit_ready_o,         // avalon st backpressure signal (1=accept, 0=full)
   input    wire  transmit_valid_i,         // source has valid data
   input    wire  [31:0] transmit_data_i,   // data (32 bit, 4 Symbols a 8 Bit)
   input    wire  transmit_sop_i,           // start of packet
   input    wire  transmit_eop_i,           // end of packet
   input    wire  [1:0]transmit_empty_i,    // empty symbols

   // avalon streaming interface receiver (source)
   output   wire  [31:0] receive_data_o,    // data (32 bit, 4 Symbols a 8 Bit)
   output   wire  receive_valid_o,          // data valid
   input    wire  receive_ready_i,          // sink ready to accept data
   output   reg   receive_sop_o,            // start of packet
   output   reg   receive_eop_o,            // end of packet
   output   reg   [1:0]receive_empty_o,     // empty symbols

   // avalon slave signals
   input    wire  [2:0] as_address_i,       // lower address bits

   input    wire  as_read_i,                // read transfer
   output   reg   [31:0] as_readdata_o,     // read data

   input    wire  as_write_i,               // write transfer
   input    wire  [31:0] as_data_i,         // write data

   output   wire  as_wait_request_o,        // slave to master wait signal
   input    wire  begintransfer_i,          // master to slave begin transfer signal
   input    wire  [3:0] as_byteenable_i,    // enable byte lane (1111 = write full 32 bits)

   // audio interface
   input    wire  [1:0]ref_clk_i,

   output   wire  bclk_o,
   output   wire  daclr_clk_o,
   output   wire  adclr_clk_o,
   output   wire  dac_o,
   input    wire  dac_i,
   output   wire  xclk_o
);

// --------------------------------------------------------------------------
// register and wire definitions
// --------------------------------------------------------------------------
// internal variables
reg   [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] bclk_divider_reg;
reg   [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] daclr_divider_reg;
reg   [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] adclr_divider_reg;
reg   [`MAX_BYTES_PER_SAMPLE_BITS-1:0] sample_size_reg; // (e.g. 2= 2xBytes (16Bit)
reg   clock_control_reg;

reg   rc_channel_l_reg;
reg   rc_channel_r_reg;
reg   tr_channel_l_reg;
reg   tr_channel_r_reg;

reg   tr_was_empty_state;
```

```
1
2  wire  [7:0] transmit_fifo_count;
3  wire  transmit_fifo_full;
4
5  wire  [7:0] receive_fifo_count;
6  wire  receive_fifo_empty;
7
8  wire  write;
9  wire  read;
10
11 wire  [`WM8731_STATE_BITS-1:0] tr_state;
12 wire  [`WM8731_STATE_BITS-1:0] rc_state;
13 reg   [`WM8731_STATE_BITS-1:0] tr_state_a;
14 reg   [`WM8731_STATE_BITS-1:0] rc_state_a;
15 reg   [`WM8731_STATE_BITS-1:0] tr_state_b;
16 reg   [`WM8731_STATE_BITS-1:0] rc_state_b;
17 wire  [9:0] tr_rdusedw;
18
19 wire  ref_clk;
20
21 // pulse to toggle because signal crosses clock domain
22 reg   transceiver_job;
23 wire  transceiver_reconfigure_latch;
24 reg   transceiver_do_job;
25
26 initial begin
27    receive_sop_o           <= 1'b0;
28    receive_eop_o           <= 1'b0;
29    receive_empty_o         <= 2'b0;

30    transceiver_do_job      <= 1'b0;
31    transceiver_job         <= 1'b0;
32 end
33
34 // ---------------------------------------------------------------------------
35 // instance definitions
36 // ---------------------------------------------------------------------------
37 // transmitter instance
38 wm8731_transmitter transmitter(
39    .ref_clk_i(ref_clk),
40    .reset_n_i(reset_n_i),
41
42    .do_job_i(transceiver_reconfigure_latch),
43    .job_i(transceiver_job),

44    .bclk_divider_i(bclk_divider_reg),
45    .daclrc_divider_i(daclr_divider_reg),
46    .adclrc_divider_i(adclr_divider_reg),

47    .tr_channel_l_i(tr_channel_l_reg),
48    .tr_channel_r_i(tr_channel_r_reg),
49    .rc_channel_l_i(rc_channel_l_reg),
50    .rc_channel_r_i(rc_channel_r_reg),

51    // control
52    .bytes_per_sample_i(sample_size_reg),
53
54    // transmit fifo
55    .tr_fifo_wrclk_i(clk_i),                 // write clock
56    .tr_fifo_data_i(transmit_data_i),        // input data 32Bit
57    .tr_fifo_wreq_i(transmit_valid_i),       // write request for new data
58    .tr_fifo_wrfull_o(transmit_fifo_full),
59    .tr_fifo_wrusedw_o(transmit_fifo_count), // fill count
60
61    .rc_fifo_rdclk_i(clk_i),                 // read clock
62    .rc_fifo_q_o(receive_data_o),            // output data 32Bit
63    .rc_fifo_rdreq_i(receive_ready_i),       // read request for new data
64    .rc_fifo_rdempty_o(receive_fifo_empty),  // empty signal
65    .rc_fifo_rdusedw_o(receive_fifo_count),  // fill count
66
67    // debug signals
68    .tr_state_o(tr_state),
69    .rc_state_o(rc_state),
70    .tr_rdusedw_o(tr_rdusedw),
71
72    // audio control signals
73    .adclr_clk_o(adclr_clk_o),
74    .dac_i(dac_i),
```

```
    .dac_o(dac_o),
    .daclr_clk_o(daclr_clk_o),
    .bclk_o(bclk_o),
    .xclk_o(xclk_o)
);

// start strobe clock synchronization
ptt_ttp start_job(
    .do_job_i(transceiver_do_job),
    .src_clk_i(clk_i),
    .dest_clk_i(ref_clk),
    .do_job_o(transceiver_reconfigure_latch)
);

// clock multiplexer
gf_mux clock_mux(
    .clk_i(ref_clk_i),
    .clk_o(ref_clk),
    .select_i(clock_control_reg)
);

// ------------------------------------------------------------------------
// continous assignments
// ------------------------------------------------------------------------
// generate streaming read enable signal
assign   receive_valid_o   = ~receive_fifo_empty;
assign   transmit_ready_o  = ~transmit_fifo_full;
assign   as_wait_request_o = begintransfer_i;

// ------------------------------------------------------------------------
// clock dependent assignments
// ------------------------------------------------------------------------
// state stabilizing clock domain transition
always @(posedge clk_i) begin
    // tr state
    tr_state_a <= tr_state;
    tr_state_b <= tr_state_a;
    // rc state
    rc_state_a <= rc_state;
    rc_state_b <= rc_state_a;
end

// generate avalon-mm
always @(posedge clk_i, negedge reset_n_i) begin
    if (!reset_n_i) begin

        bclk_divider_reg     <= `DEFAULT_REF_CLK_DIVIDER;
        daclr_divider_reg    <= `DEFAULT_REF_CLK_DIVIDER;
        adclr_divider_reg    <= `DEFAULT_REF_CLK_DIVIDER;
        sample_size_reg      <= `DEFAULT_BYTES_PER_SAMPLE;

        clock_control_reg    <= 1'b0;

        rc_channel_l_reg     <= 1'b1;
        rc_channel_r_reg     <= 1'b1;
        tr_channel_l_reg     <= 1'b1;
        tr_channel_r_reg     <= 1'b1;

        transceiver_do_job      <= 1'b0;
        transceiver_job         <= 1'b0;

        tr_was_empty_state         <= 1'b0;
    end
    else begin

        // sample tr_rdempty signal
        if (transmit_fifo_count == 0)
            tr_was_empty_state <= 1'b1;

        // shut of the reconfiguration request
        transceiver_do_job      <= #1 1'b0;

        if (begintransfer_i) begin
            if (as_write_i) begin
              case (as_address_i) // synopsis parallel_case
               `CLOCK_DIVIDER_REG: begin
                  if (as_data_i >= (`BCLK_BYTES_PER_SAMPLE*8*2*`CHANNEL_NUM))
                     bclk_divider_reg     <= #1 as_data_i[`MAX_REF_CLKS_PER_FRAME_BITS-1:0];
```

```
            else
               bclk_divider_reg     <= (`BCLK_BYTES_PER_SAMPLE*8*2*`CHANNEL_NUM);
         end

         `DACLR_DIVIDER_REG: begin
            if (as_data_i >= (`BCLK_BYTES_PER_SAMPLE*8*2*`CHANNEL_NUM))
               daclr_divider_reg    <= #1 as_data_i[`MAX_REF_CLKS_PER_FRAME_BITS-1:0];
            else
               daclr_divider_reg    <= #1 (`BCLK_BYTES_PER_SAMPLE*8*2*`CHANNEL_NUM);

            tr_channel_l_reg        <= #1 as_data_i[`CHANNEL_L_BIT];
            tr_channel_r_reg        <= #1 as_data_i[`CHANNEL_R_BIT];
         end

         `ADCLR_DIVIDER_REG: begin

            if (as_data_i >= (`BCLK_BYTES_PER_SAMPLE*8*2*`CHANNEL_NUM))
               adclr_divider_reg    <= #1 as_data_i[`MAX_REF_CLKS_PER_FRAME_BITS-1:0];
            else
               adclr_divider_reg    <= #1 (`BCLK_BYTES_PER_SAMPLE*8*2*`CHANNEL_NUM);

            rc_channel_l_reg        <= #1 as_data_i[`CHANNEL_L_BIT];
            rc_channel_r_reg        <= #1 as_data_i[`CHANNEL_R_BIT];
         end

         `BYTES_PER_SAMPLE_REG:  begin
            sample_size_reg         <= #1 as_data_i[`MAX_BYTES_PER_SAMPLE_BITS-1:0];

         end
         `CLOCK_SELECT_REG: begin
            clock_control_reg       <= #1 as_data_i[`CLOCK_SELECT_BIT];
         end
         `CONTROL_REG:  begin
            transceiver_do_job      <= #1 as_data_i[`TRANSCEIVER_DO_JOB_BIT];
            transceiver_job         <= as_data_i[`TRANSCEIVER_RECONFIGURE_BIT];
         end
        endcase
      end
      else
      if (as_read_i) begin
         case (as_address_i) // synopsis parallel_case
            `CLOCK_DIVIDER_REG:          as_readdata_o <=
                    #1 {{(32-`MAX_REF_CLKS_PER_FRAME_BITS){1'b0}}, bclk_divider_reg};
            `DACLR_DIVIDER_REG:  begin
               as_readdata_o <= 0;
               as_readdata_o[`MAX_REF_CLKS_PER_FRAME_BITS-1:0] <= #1 daclr_divider_reg;
               as_readdata_o[`CHANNEL_L_BIT]    <= #1 tr_channel_l_reg;
               as_readdata_o[`CHANNEL_R_BIT]    <= #1 tr_channel_r_reg;
            end
            `ADCLR_DIVIDER_REG: begin
               as_readdata_o <= 0;
               as_readdata_o[`MAX_REF_CLKS_PER_FRAME_BITS-1:0] <= #1 adclr_divider_reg;
               as_readdata_o[`CHANNEL_L_BIT]    <= #1 rc_channel_l_reg;
               as_readdata_o[`CHANNEL_R_BIT]    <= #1 rc_channel_r_reg;
            end
            `BYTES_PER_SAMPLE_REG: begin
               as_readdata_o             <= #1 sample_size_reg;
            end
            `CLOCK_SELECT_REG: begin
               as_readdata_o <= #1 clock_control_reg;
            end
            `TRANSMIT_FIFO_FILL_COUNT_REG: begin
               as_readdata_o <= #1 {22'b0, transmit_fifo_count, 2'b00};
            end
            `RECEIVE_FIFO_FILL_COUNT_REG: begin
               as_readdata_o <= #1 {22'b0, receive_fifo_count, 2'b00};
            end
            `STATUS_REG: begin
               as_readdata_o     <=
                     #1 {tr_rdusedw, tr_was_empty_state, rc_state_b, tr_state_b,
                      receive_fifo_empty, transmit_fifo_full};
               tr_was_empty_state   <= 1'b0;
            end

            default:
               begin
                  as_readdata_o  <= #1 {29'b0, as_address_i};
               end
```

```
            endcase
         end
      end
   end
end
endmodule
```

Quelltext 12: WM8731 IP-Core Top-Modul (wm8731_top.v)

H.2.3 Transmitter

```
/*
*
* WM8731 Audio-CODEC core, support different sample rates and runtime configuration
*
* Tranceiver module, receive data, send data, statemachine handling
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
`timescale 1ns/1ns
`include "wm8731_defines.v"

// ---------------------------------------------------------------------------
// modul and port definition
// ---------------------------------------------------------------------------
module wm8731_transmitter(
   // clocks
   input wire  ref_clk_i,        // reference clock
   input wire  reset_n_i,        // async reset (for 1. init)

   input wire  do_job_i,         // reconfigure strobe
   input wire  job_i,            // go idle=0, reconfigure=1

   input wire  tr_channel_l_i,   // use transmit channel left (0=disabled, 1=enabled)
   input wire  tr_channel_r_i,   // use transmit channel right (0=disabled, 1=enabled)
   input wire  rc_channel_l_i,   // use receive channel left (0=disabled, 1=enabled)
   input wire  rc_channel_r_i,   // use receive channel right (0=disabled, 1=enabled)

   input wire  [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] bclk_divider_i,  // e.g. 384=(18432Khz/48Khz)
   input wire  [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] daclrc_divider_i,// e.g. 384=48Khz (@18432Khz)
   input wire  [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] adclrc_divider_i,

   // bytes per sample eg. 24bit sample => 3 byte
   input wire  [`MAX_BYTES_PER_SAMPLE_BITS-1:0] bytes_per_sample_i,

   // data fifo
   input wire  tr_fifo_wrclk_i,                        // write clock
   input wire  [`FIFO_DATA_IN_BITS-1:0] tr_fifo_data_i, // input data 32Bit
   input wire  tr_fifo_wreq_i,                         // write request for new data
   output   wire  [7:0] tr_fifo_wrusedw_o,             // fill count
   output   wire  tr_fifo_wrfull_o,

   input wire  rc_fifo_rdclk_i,                        // read clock
   input wire  rc_fifo_rdreq_i,
   output   wire  [7:0] rc_fifo_rdusedw_o,
   output   wire  rc_fifo_rdempty_o,
   output   wire  [32-1:0] rc_fifo_q_o,

   output   wire  [`WM8731_STATE_BITS-1:0] tr_state_o,  // transmit state output
   output   wire  [`WM8731_STATE_BITS-1:0] rc_state_o,  // receive state output
   output   wire  [9:0] tr_rdusedw_o,                   // test signal for transmit fill count
   // audio control signals
   output   wire  xclk_o,                           // MCLK
   output   reg      bclk_o,                        // BCKL
   output   reg      daclr_clk_o,                   // DACLR
   output   reg      adclr_clk_o,                   // ADCLR
   output   wire  dac_o,                            // DAC
   input wire  dac_i                                // ADC
);


// ---------------------------------------------------------------------------
```

```
1  // register and wire definitions
2  // ------------------------------------------------------------------------------
3  reg            [3:0] even_tranceive_compare;
4
5  wire           tr_rdempty;
6  wire           tr_rdfull;
7  wire           [9:0] tr_rdusedw;
8  wire           [`FIFO_DATA_OUT_BITS-1:0] tr_fifo_q;
9  wire           tr_fifo_rdclk;
10
11 reg            tr_fifo_rdreq;
12 reg            tr_new_data_req;
13 reg            [8-1:0] tr_read_data;
14
15 reg            [`MAX_BIT_CLKS_PER_FRAME_BITS-1:0] bclk_divider;
16 reg            [`MAX_BIT_CLKS_PER_FRAME_BITS-1:0] bclk_divider_compare;
17
18 reg            [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] adclr_clk_div;
19 reg            [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] adclr_clk_div_compare;
20
21 reg            [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] rc_byte_trigger_divider;
22 reg            [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] byte_trigger_divider_compare;
23
24 reg            [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] daclr_clk_div;
25 reg            [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] daclr_clk_div_compare;
26
27 reg            [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] tr_byte_trigger_divider;
28
29 reg            [`MAX_BIT_CLKS_PER_FRAME_BITS-1:0] bit_trigger_divider;
30 reg            [`MAX_BIT_CLKS_PER_FRAME_BITS-1:0] bit_trigger_divider_compare;
31
32 reg            [3-1:0] tr_bit_num;
33 reg            [8-1:0] tr_byte_num;
34 reg            tr_channel_l;
35 reg            tr_channel_r;
36 reg            [`MAX_BYTES_PER_SAMPLE_BITS+1-1:0] tr_bytes_per_frame;
37
38 reg            [3:0] tr_even_transfer;
39 reg            tr_frame_read_enable;
40 reg            tr_sample_read_enable;
41 reg            [`WM8731_STATE_BITS-1:0] tr_state;
42
43 wire           rc_fifo_wrclk;
44 wire           rc_fifo_wrfull;
45 wire           [9:0] rc_fifo_wrusedw;
46 reg            [3:0] rc_even_transfer;
47 reg            [3-1:0] rc_bit_num;
48 reg            [8-1:0] rc_byte_num;
49 reg            [8-1:0] rc_data;
50
51 reg            [8-1:0] rc_fifo_data;
52 reg            rc_frame_write_enable;
53 reg            rc_sample_write_enable;
54 reg            rc_fifo_wrreq;
55 reg            rc_channel_l;
56 reg            rc_channel_r;
57 reg            [`MAX_BYTES_PER_SAMPLE_BITS+1-1:0] rc_bytes_per_frame;
58 reg            [`MAX_BYTES_PER_SAMPLE_BITS-1:0] bytes_per_sample;
59 reg            [`WM8731_STATE_BITS-1:0] rc_state;
60
61 reg            reconfiguration_req;
62
63 // ------------------------------------------------------------------------------
64 // instance definition
65 // ------------------------------------------------------------------------------
66 // transfer fifo
67 wm8731_transmitter_fifo tr_inst(
68    .aclr(~reset_n_i),
69    .data(tr_fifo_data_i),
70    .rdclk(tr_fifo_rdclk),
71    .rdreq(tr_fifo_rdreq),
72    .rdusedw(tr_rdusedw),
73    .wrclk(tr_fifo_wrclk_i),
74    .wrreq(tr_fifo_wreq_i),
75    .q(tr_fifo_q),
76    .rdempty(tr_rdempty),
77    .rdfull(tr_rdfull),
78    .wrfull(tr_fifo_wrfull_o),
```

```
    .wrusedw(tr_fifo_wrusedw_o)
);

// receive fifo
wm8731_receiver_fifo rc_inst(
    .aclr(~reset_n_i),
    .data(rc_fifo_data),
    .wrclk(rc_fifo_wrclk),
    .wrreq(rc_fifo_wrreq),
    .wrfull(rc_fifo_wrfull),
    .wrusedw(rc_fifo_wrusedw),

    .rdclk(rc_fifo_rdclk_i),
    .rdreq(rc_fifo_rdreq_i),
    .rdusedw(rc_fifo_rdusedw_o),
    .q(rc_fifo_q_o),
    .rdempty(rc_fifo_rdempty_o)
);

// --------------------------------------------------------------------------
// continous assignments
// --------------------------------------------------------------------------
// state assignments
assign tr_state_o       = tr_state;
assign rc_state_o       = rc_state;
assign tr_rdusedw_o  = tr_rdusedw;

// fifo clock assignments
assign tr_fifo_rdclk    = ~ref_clk_i;
assign rc_fifo_wrclk    = ~ref_clk_i;

// assign audio clock
assign xclk_o        = ~ref_clk_i;
assign dac_o         = tr_read_data[~tr_bit_num];

// --------------------------------------------------------------------------
// init statemachine task
// --------------------------------------------------------------------------
task init_task(
    input wire [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] _bclk_divider,
    input wire [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] _adclrc_divider,
    input wire [`MAX_REF_CLKS_PER_FRAME_BITS-1:0] _daclrc_divider,

    input wire [`MAX_BYTES_PER_SAMPLE_BITS-1:0] _bytes_per_sample,

    input wire _tr_channel_l,
    input wire _tr_channel_r,

    input wire _rc_channel_l,
    input wire _rc_channel_r
    );
begin
    // bclk reset
    bclk_divider              <= 0;
    bclk_divider_compare         <= _bclk_divider/(_bytes_per_sample*8*`CHANNEL_NUM*2)-1;
    bclk_o                    <= 1'b0;


    // adclr reset
    adclr_clk_div             <= 0;
    adclr_clk_o               <= 1'b1;
    adclr_clk_div_compare        <= _adclrc_divider/2-1;

    // daclr reset
    daclr_clk_div             <= 0;
    daclr_clk_o               <= 1'b1;
    daclr_clk_div_compare        <= _daclrc_divider/2-1;

    even_tranceive_compare       <= `PACKET_SIZE-1;//(_bytes_per_sample == 2'd3) ? 12-1:4-1;

    bit_trigger_divider          <= 0;
    bit_trigger_divider_compare     <= _bclk_divider/(_bytes_per_sample*`CHANNEL_NUM*8)-1;
    byte_trigger_divider_compare  <= _bclk_divider/(_bytes_per_sample*`CHANNEL_NUM)-1;
    bytes_per_sample          <= _bytes_per_sample;

    $display("bytes_per_sample: %d", bytes_per_sample);
    $display("bclk_divider_compare: %d", bclk_divider_compare);
    $display("bit_trigger_divider_compare: %d", bit_trigger_divider_compare);
```

```
   $display("byte_trigger_divider_compare: %d", byte_trigger_divider_compare);

   // init transmitter
   tr_byte_num               <= 0;
   tr_frame_read_enable         <= 1'b0;
   tr_sample_read_enable        <= 1'b0;
   tr_fifo_rdreq             <= 1'b0;
   tr_new_data_req              <= 1'b0;

   tr_byte_trigger_divider      <= 0;
   tr_bit_num                <= 0;
   tr_read_data              <= 0;
   tr_bytes_per_frame           <= _bytes_per_sample*(_tr_channel_l+_tr_channel_r);

   tr_channel_l              <= _tr_channel_l;
   tr_channel_r              <= _tr_channel_r;

   tr_even_transfer          <= 0;

   // init receiver
   rc_byte_trigger_divider      <= 0;

   rc_bit_num                <= 0;
   rc_byte_num               <= 0;

   rc_frame_write_enable        <= 1'b0;
   rc_sample_write_enable       <= 1'b0;

   rc_bytes_per_frame           <= _bytes_per_sample*(_rc_channel_l+_rc_channel_r);
   rc_channel_l              <= _rc_channel_l;
   rc_channel_r              <= _rc_channel_r;

   rc_fifo_wrreq             <= 1'b0;
   rc_fifo_data              <= 0;

   rc_even_transfer          <= 0;
end
endtask

// ------------------------------------------------------------------------------
// transfer init frame task
// ------------------------------------------------------------------------------
task tr_init_frame_task();
begin
   if ( tr_rdempty || tr_rdusedw < tr_bytes_per_frame) begin
      tr_frame_read_enable <= 1'b0;
      tr_sample_read_enable   <= 1'b0;
      tr_fifo_rdreq        <= 1'b0;
   end
   else begin
      tr_frame_read_enable    <= 1'b1;
      tr_sample_read_enable   <= tr_channel_l;
      tr_fifo_rdreq        <= tr_channel_l;
   end
end
endtask

// ------------------------------------------------------------------------------
// receive init frame task
// ------------------------------------------------------------------------------
task rc_init_frame_task();
begin
   // define if the hole frame fits in the receive fifo
   if (rc_fifo_wrfull || (rc_fifo_wrusedw+rc_bytes_per_frame) >= `FIFO_BYTE_SIZE) begin
      rc_frame_write_enable      <= 1'b0;
      rc_sample_write_enable     <= 1'b0;
   end
   else begin
      rc_frame_write_enable      <= 1'b1;
      rc_sample_write_enable     <= rc_channel_l;
   end
end
endtask

// shift dac data in
always@(posedge bclk_o) begin
   rc_data[~rc_bit_num] <= dac_i;
end
```

```
1
2  // ------------------------------------------------------------------------
3  // transceiver machine
4  // ------------------------------------------------------------------------
5  always @(posedge ref_clk_i, negedge reset_n_i) begin
6     if (~reset_n_i) begin
7        // init
8        init_task(
9           `DEFAULT_REF_CLK_DIVIDER,
10          `DEFAULT_REF_CLK_DIVIDER,
11          `DEFAULT_REF_CLK_DIVIDER,
12          `DEFAULT_BYTES_PER_SAMPLE,
13          1'b1,
14          1'b1,
15
16          1'b1,
17          1'b1
18       );

19       // state setup

20       tr_state       <= `WM8731_STATE_IDLE;
21       rc_state       <= `WM8731_STATE_IDLE;

22       reconfiguration_req  <= 1'b0;
23    end
24    else
25    if (ref_clk_i) begin

26       // synchronous reconfiguration
27       if (reconfiguration_req && tr_state == `WM8731_STATE_IDLE &&
28             rc_state == `WM8731_STATE_IDLE) begin
29          $display("reconfiguration of transceiver, channels tr:%b, rc:%b",
30             {tr_channel_l_i, tr_channel_r_i}, {rc_channel_l_i, rc_channel_r_i});

31          // setup reconfigured values
32          init_task(
33             bclk_divider_i,
34             adclrc_divider_i,
35             daclrc_divider_i,
36             bytes_per_sample_i,
37             tr_channel_l_i,
38             tr_channel_r_i,

39             rc_channel_l_i,
40             rc_channel_r_i
41          );

42          rc_state       <= `WM8731_STATE_RUNNING;
43          tr_state       <= `WM8731_STATE_RUNNING;

44          // reconfiguration done...
45          reconfiguration_req  <= 1'b0;
46       end
47       else
48       begin
49          // was there a job request?
50          if (do_job_i) begin
51             $display("do job request");
52             // set the state
53             if (tr_state == `WM8731_STATE_RUNNING) begin
54                tr_state <= `WM8731_STATE_ON_EVEN_IDLE;
55             end

56             if (rc_state == `WM8731_STATE_RUNNING) begin
57                rc_state <= `WM8731_STATE_ON_EVEN_IDLE;
58             end

59             reconfiguration_req <= job_i;
60          end

61          // ------------------------------------------------------------------------------------------
62          // bclk generation
63          if(bclk_divider >= bclk_divider_compare) begin
64             bclk_divider   <= 0;
65             bclk_o         <= ~bclk_o;
66          end else
67             bclk_divider   <= bclk_divider+1'b1;
```

```
1
2           // ---------------------------------------------------------------------------------------
3           // bit generation
4           if(bit_trigger_divider >= bit_trigger_divider_compare) begin
5              bit_trigger_divider  <= 0;

6              // increment rc bit num
7              rc_bit_num        <= rc_bit_num+1'b1;

8              // increment tr bitnum
9              tr_bit_num        <= tr_bit_num+1'b1;
10          end
11          else begin
12             bit_trigger_divider  <= bit_trigger_divider+1'b1;
13          end
14
15          // ---------------------------------------------------------------------------------------
16          // daclr generation
17          if(daclr_clk_div >= daclr_clk_div_compare) begin
18             // channel change
19             daclr_clk_div  <= 0;
20             daclr_clk_o    <= ~daclr_clk_o;

21             tr_bit_num     <= 0;
22          end
23          else begin
24             daclr_clk_div  <= daclr_clk_div+1'b1;
25          end


26          // ---------------------------------------------------------------------------------------
27          // transfer even byte amount check
28          if (tr_fifo_rdreq) begin
29             if (tr_even_transfer >= even_tranceive_compare) begin
30                tr_even_transfer  <= 0;
31             end
32             else begin
33                tr_even_transfer  <= tr_even_transfer+1'b1;
34             end
35          end

36          // ---------------------------------------------------------------------------------------
37          // dac byte generation
38          if(daclr_clk_div == daclr_clk_div_compare-1) begin
39             tr_byte_num               <= 0;
40             tr_byte_trigger_divider      <= 0;
41             tr_new_data_req              <= 1'b1;

42             // frame start (left channel)
43             if (~daclr_clk_o) begin
44                // state check
45                case (tr_state)
46                   `WM8731_STATE_IDLE: begin
47                      tr_frame_read_enable    <= 1'b0;
48                      tr_sample_read_enable     <= 1'b0;
49                      tr_fifo_rdreq          <= 1'b0;
50                   end
51                   `WM8731_STATE_RUNNING: begin
52                      tr_init_frame_task();
53                   end
54                   `WM8731_STATE_ON_EVEN_IDLE: begin
55                      if (tr_even_transfer == 0) begin
56                         tr_state            <= `WM8731_STATE_IDLE;
57                         tr_frame_read_enable <= 1'b0;
58                         tr_sample_read_enable   <= 1'b0;
59                         tr_fifo_rdreq       <= 1'b0;
60                      end
61                      else begin
62                         tr_init_frame_task();
63                      end
64                   end
65                endcase
66             end
67             // right channel
68             else begin
69                tr_sample_read_enable      <= tr_frame_read_enable & tr_channel_r;
70                tr_fifo_rdreq          <= tr_frame_read_enable & tr_channel_r;
71             end
```

```
1           end
2           else
3           begin /*if (tr_frame_read_enable)*/
4              // next byte
5              if(tr_byte_trigger_divider >= byte_trigger_divider_compare) begin
6                 tr_byte_trigger_divider    <= 0;
7                 tr_byte_num                <= tr_byte_num+1'b1;

8                 tr_new_data_req            <= 1'b1;
9                 tr_fifo_rdreq              <= tr_sample_read_enable &&
10                                              (tr_byte_num+1'b1) < bytes_per_sample?
11                                              1'b1: 1'b0;
12             end
13             else begin
14                tr_byte_trigger_divider    <= tr_byte_trigger_divider+1'b1;
15                if (tr_new_data_req) begin
16                   if (tr_fifo_rdreq)
17                      tr_read_data      <= tr_fifo_q;
18                   else
19                      tr_read_data      <= 8'd0;
20                end

21                tr_fifo_rdreq           <= 1'b0;
22                tr_new_data_req            <= 1'b0;
23             end
24          end

25          // ---------------------------------------------------------------------------------------
26          // adclr generation
27          if(adclr_clk_div >= adclr_clk_div_compare) begin
28             adclr_clk_div  <= 0;
29             adclr_clk_o    <= ~adclr_clk_o;

30             rc_bit_num     <= 0;
31          end
32          else begin
33             adclr_clk_div  <= adclr_clk_div+1'b1;
34          end
35
36          // ---------------------------------------------------------------------------------------
37          // receive even byte amount check
38          if (rc_fifo_wrreq) begin
39             if (rc_even_transfer >= even_tranceive_compare) begin
40                rc_even_transfer  <= 0;
41             end
42             else begin
43                rc_even_transfer  <= rc_even_transfer+1'b1;
44             end
45          end
46
47          // ---------------------------------------------------------------------------------------
48          // receive
49          if(rc_byte_trigger_divider >= byte_trigger_divider_compare) begin
50             rc_byte_trigger_divider    <= 0;

51             // start of a sample (left or right)
52             if(adclr_clk_div >= adclr_clk_div_compare) begin
53                rc_byte_num         <= 0;

54                // frame start (left channel)
55                if (~adclr_clk_o) begin
56                   //$display("rc frame start state: %d", rc_state);
57                   case (rc_state)
58                      `WM8731_STATE_IDLE: begin
59                         rc_frame_write_enable      <= 1'b0;
60                         rc_sample_write_enable     <= 1'b0;
61                      end
62                      `WM8731_STATE_RUNNING: begin
63                         rc_init_frame_task();
64                      end
65                      `WM8731_STATE_ON_EVEN_IDLE: begin

66                         // if writing will happen
67                         if (rc_even_transfer == 0 || (rc_byte_num < bytes_per_sample &&
68 rc_sample_write_enable && rc_even_transfer >= even_tranceive_compare)) begin
69                            rc_state                <= `WM8731_STATE_IDLE;
70                            rc_frame_write_enable      <= 1'b0;
71                            rc_sample_write_enable     <= 1'b0;
```

```
                        end
                        else begin
                           rc_init_frame_task();
                        end
                     end
                  endcase
               end
               else begin
                  rc_sample_write_enable     <= rc_frame_write_enable & rc_channel_r;
               end
            end
            else begin
               rc_byte_num          <= rc_byte_num+1'b1;
            end

            // if at the end of a byte
            rc_fifo_wrreq           <= rc_byte_num < bytes_per_sample &&
                                       rc_sample_write_enable? 1'b1 : 1'b0;
            rc_fifo_data            <= rc_data;

         end
         else begin
            rc_byte_trigger_divider    <= rc_byte_trigger_divider+1'b1;
            rc_fifo_wrreq              <= 1'b0;
         end
      end
   end // ref_clk_i
end


endmodule
```

Quelltext 13: WM8731 IP-Core Transmitter (wm8731_transmitter.v)

H.2.4 Takt-Multiplexer

```
/*
*
* Glitch-free clock multiplexing
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
`timescale 1ns/1ns
module gf_mux(
   input wire [1:0]clk_i,
   //input wire reset_n_i,
   input wire select_i,
   output wire clk_o
);
reg [1:0]clk_a;
reg [1:0]clk_b;

initial begin
   clk_a  <= 2'b0;
   clk_b  <= 2'b0;
end

always @(posedge clk_i[0]/*, negedge reset_n_i*/) begin
/* if (~reset_n_i)
      clk_a[0] <= 0;
   else
   if (clk_i[0])*/
      clk_a[0] <= ~select_i & ~clk_b[1];
end

always @(negedge clk_i[0]/*, negedge reset_n_i*/) begin
   /*if (~reset_n_i)
      clk_a[1] <= 0;
   else
   if (~clk_i[0])*/
      clk_a[1] <= clk_a[0];
end

```

```
always @(posedge clk_i[1]/*, negedge reset_n_i*/) begin
/* if (~reset_n_i)
      clk_b[0] <= 0;
   else
   if (clk_i[1])*/
      clk_b[0] <= select_i & ~clk_a[1];
end

always @(negedge clk_i[1]/*, negedge reset_n_i*/) begin
/* if (~reset_n_i)
      clk_b[1] <= 0;
   else
   if (~clk_i[1])*/
      clk_b[1] <= clk_b[0];
end

assign clk_o = clk_i[0]&clk_a[1] | clk_i[1]&clk_b[1];

endmodule
```

Quelltext 14: Takt-Multiplexer (gf_mux.v)

H.2.5 Synchronisations-Logik

```
/*
* Design entry
*
* For asynchronous data transfer between clock domains
*
* (c) 2008-2011 by Birger Zimmermann
*
* Design principle taken from:
* Crews M.; Yuenyongsgool Y.:
* Practical design for transferring signals between clock domains, Philips Semiconductors, 2003.
*/
`timescale 1ns/1ns
// module definition with io port
module ptt_ttp(
   input wire  do_job_i,   // do job signal in
   input wire  src_clk_i,  // source clock input

   input wire  dest_clk_i, // dest clock input
   output   wire  do_job_o // do job signal out
);
// parameter definition
parameter OUTPUT_DELAY = 1;
// register definition
reg      take_it_tg  = 1'b0;
reg      b        = 1'b0;
reg      c        = 1'b0;
reg      d        = 1'b0;

// logical assignment (toggle to pulse)
assign #OUTPUT_DELAY do_job_o = c ^ d;

// source clock dependent register assignment (pulse to toggle)
always @(posedge src_clk_i) begin
   take_it_tg <= do_job_i ? ~take_it_tg : take_it_tg;
end

// destination clock dependent register assignment (synchronizer)
always @(posedge dest_clk_i) begin
   b  <= take_it_tg;
   c  <= b;
   d  <= c;
end

endmodule
```

Quelltext 15: Synchronisations-Logik (ptt_ttp.v)

H.2.6 Testbench

```
/*
*
* WM8731 Audio-CODEC core, support different sample rates and runtime configuration
*
* Testbench
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
`timescale 1ns/1ns
`include "wm8731_defines.v"

// ---------------------------------------------------------------------------
// modul definition
// ---------------------------------------------------------------------------
module wm8731_tb();
// ---------------------------------------------------------------------------
// register and wire definitions
// ---------------------------------------------------------------------------
reg      dest_clk;

wire     bclk;
wire     daclr_clk;
wire     adclr_clk;
wire     xclk;
wire     dac_o;
reg      [7:0] dat;
reg      [2-1:0] bytes_per_sample;
reg      [12-1:0] bclk_divider=12'd384;
reg      [12-1:0] daclrc_divider=12'd384;
reg      [12-1:0] adclrc_divider=12'd384;
reg      reset_n;

reg      src_clk;
reg      [32-1:0] fifo_data;
reg      fifo_wreq;
reg      receive_rdreq;

reg      tr_channel_r;
reg      tr_channel_l;
reg      rc_channel_r;
reg      rc_channel_l;

wire     fifo_full;
wire     [7:0] tr_fifo_wrusedw;
wire     [9:0] tr_rdusedw;

wire     [32-1:0] receive_data;
wire     receive_fifo_empty;
wire     [7:0] receive_fifo_count;
wire     [1:0] tr_state;
wire     [1:0] rc_state;

reg      do_job_strobe;
wire     do_job_latch;
reg      job;

reg   [32-1:0]i;
reg   [32-1:0] read_data;

// ---------------------------------------------------------------------------
// instance definition
// ---------------------------------------------------------------------------
// transmitter instance
wm8731_transmitter trans(
   .reset_n_i(reset_n),
   .ref_clk_i(dest_clk),

   .do_job_i(do_job_latch),
   .job_i(job),
   // control
   .bclk_divider_i(bclk_divider),
   .daclrc_divider_i(daclrc_divider),
   .adclrc_divider_i(adclrc_divider),
   .bytes_per_sample_i(bytes_per_sample),
```

```
    .tr_channel_l_i(tr_channel_l),
    .tr_channel_r_i(tr_channel_r),

    .rc_channel_l_i(rc_channel_l),
    .rc_channel_r_i(rc_channel_r),

    // fifo
    .tr_fifo_wrclk_i(src_clk),               // write clock
    .tr_fifo_data_i(fifo_data),              // input data 32Bit
    .tr_fifo_wreq_i(fifo_wreq),              // write request for new data
    .tr_fifo_wrfull_o(fifo_full),
    .tr_fifo_wrusedw_o(tr_fifo_wrusedw),     // fill count
    .tr_rdusedw_o(tr_rdusedw),

    .rc_fifo_rdclk_i(src_clk),               // read clock
    .rc_fifo_q_o(receive_data),              // output data 32Bit
    .rc_fifo_rdreq_i(receive_rdreq),         // read request for new data
    .rc_fifo_rdempty_o(receive_fifo_empty),  // empty signal
    .rc_fifo_rdusedw_o(receive_fifo_count),  // full count

    .rc_state_o(rc_state),
    .tr_state_o(tr_state),

    // outputs
    .daclr_clk_o(daclr_clk),
    .dac_o(dac_o),
    .bclk_o(bclk),
    .xclk_o(xclk),
    .adclr_clk_o(adclr_clk),
    .dac_i(dac_o)
);

// start strobe generation (synch logic)
ptt_ttp start_job(
    .do_job_i(do_job_strobe),
    .src_clk_i(src_clk),
    .dest_clk_i(dest_clk),
    .do_job_o(do_job_latch)
);

// ---------------------------------------------------------------------------
// continous assignments
// ---------------------------------------------------------------------------
// generate clock
always #5 dest_clk = ~dest_clk;

// generate clock
always #2 src_clk = ~src_clk;

// ---------------------------------------------------------------------------
// write fifo task (fill fifo)
// ---------------------------------------------------------------------------
task write_fifo(input wire [32-1:0] _data);
begin
    @(negedge src_clk);
    fifo_data = _data;
    fifo_wreq = 1'b1;
    @(posedge src_clk);
    #1 fifo_wreq = 1'b0;
end
endtask
// ---------------------------------------------------------------------------
// read fifo task (read data from fifo)
// ---------------------------------------------------------------------------
task read_fifo(output reg [32-1:0] _data);
begin
    @(negedge src_clk);
    receive_rdreq = 1'b1;
    @(posedge src_clk);
    #1 _data = receive_data;
    #1 receive_rdreq = 1'b0;
end
endtask
```

```
1  // ------------------------------------------------------------------------
2  // reconfiguration task
3  // ------------------------------------------------------------------------
4  task reconfigure_transceiver(
5     input wire [1:0] _tr_channels,
6     input wire [1:0] _rc_channels,
7     input wire [2-1:0] _bytes_per_sample );
8  begin
9     tr_channel_r      = _tr_channels[0];
10    tr_channel_l      = _tr_channels[1];
11    rc_channel_r      = _rc_channels[0];
12    rc_channel_l      = _rc_channels[1];
13    bytes_per_sample  = _bytes_per_sample;
14    job               = 1'b1; // reconfigure
15    @(negedge src_clk);
16    do_job_strobe     = 1'b1;
17    @(negedge src_clk);
18    do_job_strobe     = 1'b0;
19 end
20 endtask
21
22 // ------------------------------------------------------------------------
23 // fill FIFO task
24 // ------------------------------------------------------------------------
25 `define MONO   1'b0
26 `define STEREO 1'b1
27
28 task read_samples_from_file(input integer _size, input wire _stereo);
29 integer file, r;
30 reg      [32-1:0] stereo;
31 reg      [16-1:0] mono;
32 begin
33
34    file = $fopen("testdata/music.wav", "rb");
35    if (!file) begin
36       $display("could not open wav file");
37       $finish;
38    end

39    // skipping wav header
40    r = $fseek(file, 44, 0);

41    // read wav data
42    for (i=0;i < _size; i=i+1) begin
43       if (!_stereo) begin
44          r = $fread(mono, file);
45          stereo[31:16] = mono;
46          r = $fseek(file, 2, 1);
47          r = $fread(mono, file);
48          stereo[15:0] = mono;
49       end
50       else
51          r = $fread(stereo, file);
52
53       wait(!fifo_full);
54
55       write_fifo(stereo); // write data in fifo
56    end
57
58    $fclose(file);
59 end
60 endtask
```

```
1  // ---------------------------------------------------------------------------
2  // Testprogram
3  // ---------------------------------------------------------------------------
4  initial begin
5     // init clocks
6     dest_clk    = 1'b0;
7     src_clk        = 1'b0;
8     // init job strobe
9     do_job_strobe  = 1'b0;

10    // reset the transceiver
11    reset_n = 1'b1;
12    #5
13    reset_n = 1'b0;
14    #5
15    reset_n = 1'b1;

16    #1000

17    // reconfigure, only left channels, 16-Bit (2 Bytes)
18    reconfigure_transceiver(2,2,2);

19    // fill fifo with test data
20    read_samples_from_file(1*`PACKET_SIZE, `MONO);

21    // wait for fifo empty
22    wait (tr_fifo_wrusedw == 0);

23    // reconfigure, only right channel, 24-Bit (3 Bytes)
24    reconfigure_transceiver(1,1,3);

25    // fill fifo with test data
26    read_samples_from_file(1*`PACKET_SIZE, `MONO);

27    // wait for fifo empty
28    wait (tr_fifo_wrusedw == 0);
29    // reconfigure, both channels, 16-Bit (2 Bytes)
30    reconfigure_transceiver(3,3,2);

31    // fill fifo with test data
32    read_samples_from_file(1*`PACKET_SIZE, `STEREO);

33    // wait for fifo empty
34    wait (tr_fifo_wrusedw == 0);
35
36 /* #36000
37    read_fifo(read_data);
38    $display("read:%x", read_data);
39    #50
40    read_fifo(read_data);
41    $display("read:%x", read_data);
42    #50
43    read_fifo(read_data);
44    $display("read:%x", read_data);  */
45    /*#1500000*/ $stop;
46    $finish;
47 end
48
49 endmodule
```

Quelltext 16: WM8731 IP-Core Testbench (wm8731_tb.v)

H.3 64-Bit-ALU Verilog-Quelltexte

H.3.1 Definitionsdatei

```
60 /*
61 *
```

```
1  * 64-Bit-ALU
2  *
3  * Defines
4  *
5  * (c) 2008-2011 by Birger Zimmermann
6  *
7  */
8  // opcodes used in top module
9  `define OP_LOAD         4'b000?
10 `define OP_MULT         4'b001?
11 `define OP_MULT_ADD     4'b010?
12 `define OP_MULT_ADD_Q   4'b011?
13 `define OP_GET          4'b100?
14
15 `define OP_ABS          4'b1010
16 `define OP_CLZ          4'b1011
17 `define OP_CLIP_16      4'b1100
18 `define OP_LOAD_BOTH    4'b1101
19 `define OP_STORE        4'b1110
20
21 `define ACCU_NUM_BIT    0
22
23 // opcodes used in test bench
24 // accu a
25 `define OP_A_LOAD       4'b0000
26 `define OP_A_MULT       4'b0010
27 `define OP_A_MULT_ADD   4'b0100
28 `define OP_A_MULT_ADD_Q 4'b0110
29 `define OP_A_GET        4'b1000
30
31 // accu b
32 `define OP_B_LOAD       4'b0001
33 `define OP_B_MULT       4'b0011
34 `define OP_B_MULT_ADD   4'b0101
35 `define OP_B_MULT_ADD_Q 4'b0111
36 `define OP_B_GET        4'b1001
37
38 // cycle size, names
39 `define CYCLE_BITS      3
40
41 `define CYCLE_PROCESS   3'd2
42 `define CYCLE_PRE_FETCH 3'd1
43 `define CYCLE_FETCH     3'd0
```

Quelltext 17: 64-Bit-ALU Definitionsdatei (mult64_defines.v)

H.3.2 Top-Modul

```
44 /*
45 *
46 * 64-Bit-ALU
47 *
48 * Top-Module
49 *
50 * (c) 2008-2011 by Birger Zimmermann
51 *
52 */
53 // synopsys translate_off
54 `timescale 1ns/1ns
55 // synopsys translate_on
56 `include "mult64_defines.v"
57 // ----------------------------------------------------------------------------
58 // modul and port definition
59 // ----------------------------------------------------------------------------
60 module mult64_top(
61    input wire clk_i,          // CPU's master-input clk <required for multi-cycle>
62    input wire reset_i,        // CPU's master asynchronous reset <required for multi-cycle>
63    input wire clk_en_i,       // Clock-qualifier <required for multi-cycle>
64    input wire start_i,        // True when this instr. issues <required for multi-cycle>
65    output   reg done_o,       // True when instr. completes <required for variable muli-cycle>
66    input wire [31:0]dataa_i,  // operand A <always required>
67    input wire [31:0]datab_i,  // operand B <optional>
68    input wire [3:0] n_i,      // N-field selector <required for extended>
69
```

```
   output wire [31:0] result_o // result <always required>
);

// ------------------------------------------------------------------------
// parameter definitions, do not modify until you know what you are doing
// ------------------------------------------------------------------------
parameter   MULT_CYCLES                  = 2;  // Cycles need for multiplication
parameter   MAX_SHIFT_DISTANCE_BITS_PER_CYCLE   = 3;  // Maximum number of bit shift per cycle
specparam   DONE_DELAY                   = 1;  // Logic delay for rising or releasing signals
(ns)
specparam   MAX_SHIFT_DISTANCE_BITS         = 5;  // 5bit= max 2^5-1 steps + whole word (32) bit
shift
specparam   MAX_SHIFT_PER_CYCLE             = (1<<MAX_SHIFT_DISTANCE_BITS_PER_CYCLE)-1;

// ------------------------------------------------------------------------
// register and wire definitions
// ------------------------------------------------------------------------
// internal registers
reg         [63:0] c0;
reg         [63:0] c1;
reg         [31:0] shift_register_hi;
reg         [MAX_SHIFT_DISTANCE_BITS-1:0] desired_shift;
reg      [31:0] result_reg;
// control register
reg         done;
reg         [`CYCLE_BITS-1:0] cycleNum;
reg         mult_clk_enable;
reg         [5:0]zCnt;
reg         [5:0]bitPos;

// internal signals
wire     [63:0] mult_result;
wire     [63:0] shift_data_out;
wire     [MAX_SHIFT_DISTANCE_BITS-1+1:0] desired_shift_strobe;
wire     [MAX_SHIFT_DISTANCE_BITS_PER_CYCLE-1:0] shift_distance;
wire     [3:0]opcode;
wire     accu_num;
wire     coef_op;
wire     [63:0]sum;
//wire      [63:0]sum1;
wire     [63:0]selected_accu;
wire     mult_clk_enable_strobe;
wire     coef_wren;
wire     coef_rden;
wire     [31:0]coef_q;
// ------------------------------------------------------------------------
// instance definitions
// ------------------------------------------------------------------------
coef_ram coef_ram_inst(
   .address(dataa_i[8:0]),// data address operand 1
   .clken(clk_en_i),
   .clock(clk_i),
   .data(datab_i),// data in operand 2
   .rden(coef_rden),
   .wren(coef_wren),
   .q(coef_q)
   );

lpm_mult lpm_mult_component (
            .dataa(coef_op?coef_q:dataa_i),
            .datab(datab_i),
            .clken(mult_clk_enable_strobe),
            .aclr(reset_i),
            .clock(clk_i),
            .result(mult_result),
            .sum(1'b0)
            );
defparam
   lpm_mult_component.lpm_hint = "MAXIMIZE_SPEED=5",
   lpm_mult_component.lpm_pipeline = MULT_CYCLES,
   lpm_mult_component.lpm_representation = "SIGNED",
   lpm_mult_component.lpm_type = "LPM_MULT",
   lpm_mult_component.lpm_widtha = 32,
   lpm_mult_component.lpm_widthb = 32,
   lpm_mult_component.lpm_widthp = 64;

lpm_clshift lpm_clshift_component (
            .distance (shift_distance),
```

```
            .direction (1'b1),
            .data ( {shift_register_hi, result_reg} ),
            .result (shift_data_out)
            // synopsys translate_off
            ,
            .overflow (),
            .underflow ()
            // synopsys translate_on
            );

defparam
   lpm_clshift_component.lpm_shifttype = "ARITHMETIC",
   lpm_clshift_component.lpm_type = "LPM_CLSHIFT",
   lpm_clshift_component.lpm_width = 64,
   lpm_clshift_component.lpm_widthdist = MAX_SHIFT_DISTANCE_BITS_PER_CYCLE;

// ------------------------------------------------------------------------
// continous assignments
// ------------------------------------------------------------------------
assign      desired_shift_strobe    = dataa_i[MAX_SHIFT_DISTANCE_BITS-1+1:0];
assign      shift_distance          = desired_shift > MAX_SHIFT_PER_CYCLE ? MAX_SHIFT_PER_CYCLE :
desired_shift;

assign      mult_clk_enable_strobe  = (mult_clk_enable || start_i) ? clk_en_i : 1'b0;

//assign    sum0              = c0+mult_result;
assign      sum               = selected_accu+mult_result;
//assign    sum1              = c1+mult_result;
assign      selected_accu     = ~accu_num?c0:c1;
assign      accu_num          = n_i[`ACCU_NUM_BIT];
assign      opcode            = n_i[3:0];
assign      coef_op           = (opcode == `OP_A_MULT_ADD_Q || opcode == `OP_B_MULT_ADD_Q);

assign      coef_wren         = opcode == `OP_STORE ? start_i:1'b0;
assign      coef_rden         = coef_op ? start_i:1'b0;

assign      result_o          =
   ((opcode == `OP_A_MULT) || (opcode == `OP_B_MULT)) ?
      mult_result[63:32] :
   (

      ((opcode == `OP_A_MULT_ADD) || (opcode == `OP_B_MULT_ADD)) ? sum[63:32]:result_reg
   );
// ------------------------------------------------------------------------
// init operation task
// ------------------------------------------------------------------------
task init_operation_task(); begin
   // which opcode
   casex (opcode)
      //-------------------------------------------------------
      `OP_LOAD: begin
         if (~accu_num) begin
            c0          <= {dataa_i, datab_i};
         end
         else begin
            c1          <= {dataa_i, datab_i};
         end

         cycleNum    <= `CYCLE_FETCH;//1;
         done        <= 1'b1;
         done_o         <= #DONE_DELAY 1'b1;
      end
      //-------------------------------------------------------
      `OP_LOAD_BOTH: begin
         c0          <= {dataa_i, datab_i};
         c1          <= {dataa_i, datab_i};

         cycleNum    <= `CYCLE_FETCH;//1;
         done        <= 1'b1;
         done_o         <= #DONE_DELAY 1'b1;
      end
      //-------------------------------------------------------
      `OP_MULT: begin
         // start cycle
         cycleNum       <= MULT_CYCLES-1;
         mult_clk_enable   <= 1'b1;
         done        <= 1'b0;
      end
```

```
1        //-------------------------------------------------------
2        `OP_MULT_ADD: begin
3           // start cycle
4           cycleNum       <= MULT_CYCLES-1;
5           mult_clk_enable   <= 1'b1;
6           done        <= 1'b0;
7        end
8        `OP_MULT_ADD_Q: begin
9           // start cycle
10          cycleNum       <= MULT_CYCLES;
11          mult_clk_enable   <= 1'b1;
12          done        <= 1'b0;
13       end
14       //-------------------------------------------------------
15       `OP_GET: begin

16          // shift < 32
17          if (~desired_shift_strobe[5]) begin
18             {shift_register_hi, result_reg} <= selected_accu;
19          end
20          else begin
21             // hi 32 bits of accu 0
22             result_reg           <= selected_accu[63:32];
23             // load hi shift register
24             shift_register_hi    <= selected_accu[63] ? -1:0;
25          end

26          desired_shift        <= desired_shift_strobe[MAX_SHIFT_DISTANCE_BITS-1:0];
27
28          // depends on dataa_i (shift), no shift?
29          if (desired_shift_strobe[MAX_SHIFT_DISTANCE_BITS-1:0] == 0) begin
30             // goto fetch cycle
31             cycleNum       <= `CYCLE_FETCH;
32             done           <= 1'b1;
33             done_o            <= #DONE_DELAY 1'b1;
34          end
35          else
36          if (desired_shift_strobe[MAX_SHIFT_DISTANCE_BITS-1:0] <= MAX_SHIFT_PER_CYCLE) begin
37             // goto prefetch cycle
38             cycleNum       <= `CYCLE_PRE_FETCH;
39             done           <= 1'b0;
40          end
41          else begin
42             // need intermediate cycles
43             cycleNum          <= `CYCLE_PROCESS;
44             done           <= 1'b0;
45          end

46       end
47       //-------------------------------------------------------
48       `OP_STORE: begin
49          cycleNum    <= `CYCLE_FETCH;
50          done        <= 1'b1;
51          done_o         <= #DONE_DELAY 1'b1;
52       end
53       //-------------------------------------------------------
54       `OP_ABS: begin
55          // is signed?
56          if (dataa_i[31])
57             result_reg  <= -dataa_i;
58          else
59             result_reg  <= dataa_i;

60          cycleNum    <= `CYCLE_FETCH;
61          done        <= 1'b1;
62          done_o         <= #DONE_DELAY 1'b1;
63       end
64       //-------------------------------------------------------
65       `OP_CLIP_16: begin

66          if (dataa_i[31]) begin
67             // negative
68             if (dataa_i < -32768) begin
69                result_reg  <= -32768;
70             end
71             else begin
72                result_reg  <= dataa_i;
73             end
```

```
         end
         else begin
            // positive
            if (dataa_i >= 32768) begin
               result_reg  <= 32767;
            end
            else begin
               result_reg  <= dataa_i;
            end
         end

         cycleNum    <= `CYCLE_FETCH;
         done        <= 1'b1;
         done_o         <= #DONE_DELAY 1'b1;
      end
      //-------------------------------------------------------------
      `OP_CLZ: begin
         $display("%b", dataa_i);

         bitPos         <= 15;

         casex (dataa_i[31:30])
            2'b00: begin
                  zCnt        <= 2;
                  cycleNum       <= `CYCLE_PROCESS;
                  done        <= 1'b0;
                  result_reg     <= {dataa_i[29:0], 2'b0};
            end
            2'b01: begin
                  zCnt        <= 1;
                  cycleNum       <= `CYCLE_PROCESS;
                  done        <= 1'b0;
                  result_reg     <= {dataa_i[29:0], 2'b0};
            end
            2'b1?: begin
                  //zCnt         <= 0;
                  cycleNum       <= `CYCLE_FETCH;
                  done        <= 1'b1;
                  done_o         <= #DONE_DELAY 1'b1;
                  result_reg     <= 0;
            end
         /* 2'b11: begin
                  zCnt        <= 0;
                  cycleNum       <= 1;
                  done        <= 1'b1;
                  done_o         <= #DONE_DELAY 1'b1;
                  result_reg     <= 0;
            end*/
         endcase
      end
   endcase

end
endtask
// ------------------------------------------------------------------------------
// clock dependent assignments
// ------------------------------------------------------------------------------
always@(posedge clk_i, posedge reset_i) begin
   if (reset_i) begin
      // idle cycle
      cycleNum             <= 0;
      mult_clk_enable         <= 1'b0;
      done              <= 1'b1;
      done_o               <= 1'b0;
      c0                <= 0;
      c1                <= 0;
      result_reg           <= 0;
   end
   else begin
      if (clk_en_i) begin
         if (start_i) begin
            init_operation_task();
         end
         else
         case (cycleNum)
            `CYCLE_PROCESS: begin
               casex (opcode)
                  //-------------------------------------------------------
```

```
                `OP_GET: begin
                   // shift operation
                   shift_register_hi          <= shift_data_out[63:32];
                   result_reg                 <= shift_data_out[31:0];

                   desired_shift              <= desired_shift-shift_distance;

                   if (desired_shift-shift_distance <= MAX_SHIFT_PER_CYCLE) begin
                      cycleNum                <= 1;
                   end
                end
                //-------------------------------------------------------
                `OP_CLZ: begin
                   //if (bitPos >= 1) begin
                      $display("%b", result_reg);
                      casex (result_reg[31:30])
                         2'b00: begin

                            if (bitPos == 1) begin
                               cycleNum       <= `CYCLE_FETCH;
                               done_o         <= #DONE_DELAY 1'b1;
                               done        <= 1'b1;
                               result_reg     <= zCnt+2;
                            end
                            else begin
                               zCnt           <= zCnt+2'd2;
                               cycleNum       <= `CYCLE_PROCESS;
                               result_reg     <= {result_reg[29:0], 2'b0};
                            end
                         end
                         2'b01: begin
                            //zCnt            <= zCnt+1;
                            result_reg        <= zCnt+1;
                            done_o            <= #DONE_DELAY 1'b1;
                            done           <= 1'b1;
                            cycleNum          <= `CYCLE_FETCH;
                         end
                         2'b1?: begin
                            result_reg        <= zCnt;
                            done_o            <= #DONE_DELAY 1'b1;
                            done           <= 1'b1;
                            cycleNum          <= `CYCLE_FETCH;
                         end
                         /*2'b11: begin
                            result_reg        <= zCnt;
                            done_o            <= #DONE_DELAY 1'b1;
                            done              <= 1'b1;
                            cycleNum          <= 1;
                         end*/
                      endcase

                      bitPos                  <= bitPos-1'b1;
                   //end
                end
                //-------------------------------------------------------
                default: begin
                   // decrease cycle
                   cycleNum                <= cycleNum-1'b1;
                end
             endcase
          end
          //-------------------------------------------------------
          `CYCLE_PRE_FETCH: begin
             // pre fetch cycle
             if (~done) begin
                // assert done
                //done            <= 1'b1;
                casex (opcode)
                   //-------------------------------------------------------
                   `OP_MULT: begin
                      mult_clk_enable   <= 1'b0;
                   end
                   //-------------------------------------------------------
                   `OP_MULT_ADD: begin
                      mult_clk_enable   <= 1'b0;
                   end
                   `OP_MULT_ADD_Q: begin
                      mult_clk_enable   <= 1'b0;
```

```
                    end
                    //-------------------------------------------------------------
                    //`OP_CLZ: begin
                    // result_reg     <= zCnt;
                    //end
                    //-------------------------------------------------------------
                    `OP_GET: begin
                       // shift operation
                       result_reg     <= shift_data_out[31:0];

                    end
                 endcase
                 done_o            <= #DONE_DELAY 1'b1;
              end
              else begin
                 done_o            <= #DONE_DELAY 1'b0;
              end

              cycleNum <= `CYCLE_FETCH;
           end
           //-----------------------------------------------------------
           `CYCLE_FETCH: begin
              // fetch cycle
              if (~done) begin
                 casex (opcode)
                    //-------------------------------------------------------------
                    `OP_MULT: begin
                       if (~accu_num) begin
                          c0       <= mult_result;
                       end
                       else begin
                          c1       <= mult_result;
                       end

                    end
                    //-------------------------------------------------------------
                    `OP_MULT_ADD: begin
                       if (~accu_num) begin
                          // 64 bit add
                          c0          <= sum;
                       end
                       else begin
                          // 64 bit add
                          c1          <= sum;
                       end
                    end
                    //-------------------------------------------------------------

                    `OP_MULT_ADD_Q: begin
                       if (~accu_num) begin
                          // 64 bit add
                          c0          <= sum;
                       end
                       else begin
                          // 64 bit add
                          c1          <= sum;
                       end
                    end
                 endcase

                 // assert done
                 done           <= 1'b1;
              end

              done_o               <= #DONE_DELAY 1'b0;
           end
           //-----------------------------------------------------------
           default: begin
              // decrease cycle
              cycleNum          <= cycleNum-1'b1;
           end
        endcase
     end
  end
end

endmodule
```

Quelltext 18: 64-Bit-ALU Top-Modul (mult64_top.v)

H.3.3 Testbench

```
1  /*
2  *
3  * 64-Bit-ALU
4  *
5  * Testbench
6  *
7  * (c) 2008-2011 by Birger Zimmermann
8  *
9  */
10 // synopsys translate_off
11 `timescale 1ns/1ns
12 // synopsys translate_on
13 `include "mult64_defines.v"
14 // ---------------------------------------------------------------------------
15 // modul definition
16 // ---------------------------------------------------------------------------
17 module mult64_tb();
18 // ---------------------------------------------------------------------------
19 // register and wire definitions
20 // ---------------------------------------------------------------------------
21 reg      clk;
22 reg      reset;
23 reg      clk_enable;
24 reg      start;
25 reg      [31:0]dataa;
26 reg      [31:0]datab;
27 reg      [3:0] n;
28 wire     [31:0]result;
29 reg      [63:0] bigresult;
30 integer  res;
31 wire     done;
32 integer  i;
33 reg      [31:0]value;
34 // ---------------------------------------------------------------------------
35 // instance definition
36 // ---------------------------------------------------------------------------
37 mult64_top the_mult(
38    .clk_i(clk),
39    .reset_i(reset),
40    .clk_en_i(clk_enable),
41    .start_i(start),
42    .done_o(done),       // True when instr. completes <required for variable muli-cycle>
43    .dataa_i(dataa),     // operand A <always required>
44    .datab_i(datab),     // operand B <optional>
45    .n_i(n),             // N-field selector <required for extended>
46    .result_o(result)    // result <always required>
47 );
48 // ---------------------------------------------------------------------------
49 // Clock generation
50 // ---------------------------------------------------------------------------
51 always #10 clk = ~clk;
52 // ---------------------------------------------------------------------------
53 // Subroutines
54 // ---------------------------------------------------------------------------
55 task do_command_task(
56    input wire [3:0]_cmd,
57    input integer _a,
58    input integer _b);
59 begin
60    $display("excecute cmd: %d, a:%d, b:%d", _cmd, _a, _b);
61    n    = _cmd;
62    dataa = _a;
63    datab = _b;

64    start = 1'b1;
65    @(posedge clk);
66    #1
67    start = 1'b0;
68    wait(done);
```

```
   @(posedge clk);
   wait(~done);
   res = result;
end
endtask

task LOAD_A(input wire [63:0]_in); begin
   do_command_task(`OP_A_LOAD, _in[63:32], _in[31:0]);
end
endtask

task LOAD_B(input wire [63:0]_in); begin
   do_command_task(`OP_B_LOAD, _in[63:32], _in[31:0]);
end
endtask

task MULT_A_ADD(input integer _a, input integer _b); begin
   do_command_task(`OP_A_MULT_ADD, _a, _b);
end
endtask

task MULT_B_ADD(input integer _a, input integer _b); begin
   do_command_task(`OP_B_MULT_ADD, _a, _b);
end
endtask

task MULT_A_ADD_Q(input integer _a, input integer _b); begin
   do_command_task(`OP_A_MULT_ADD_Q, _a, _b);
end
endtask

task MULT_B_ADD_Q(input integer _a, input integer _b); begin
   do_command_task(`OP_B_MULT_ADD_Q, _a, _b);
end
endtask

task GET_A(input integer _a); begin
   do_command_task(`OP_A_GET, _a, 0);
end
endtask

task GET_B(input integer _a); begin
   do_command_task(`OP_B_GET, _a, 0);
end
endtask

task STORE(input integer _a, input integer _b); begin
   do_command_task(`OP_STORE, _a, _b);
end
endtask

task ABS(input integer _a); begin
   do_command_task(`OP_ABS, _a, 0);
end
endtask

task CLIP_16(input integer _a); begin
   do_command_task(`OP_CLIP_16, _a, 0);
end
endtask

task CLZ(input integer _a); begin
   do_command_task(`OP_CLZ, _a, 0);
end
endtask
task assert_equal(input _a); begin
   if (~_a) begin
      $display("assertion failed");
      $stop;
   end
end
endtask
// --------------------------------------------------------------------------
// main test program
// --------------------------------------------------------------------------
initial begin
   clk = 1'b0;
   start = 1'b0;
```

```
    reset = 1'b1;
    #1
    reset = 1'b0;
    #1
    clk_enable = 1'b1;
    $display("----------------");
    STORE(0, 2);
    STORE(1, 3);
    STORE(2, 4);

    LOAD_A(0);
    MULT_A_ADD_Q(0, 300);
    MULT_A_ADD_Q(1, 200);
    bigresult[63:32] = result;
    GET_A(0);
    bigresult[31:0] = result;
    $display("result: %d =1200 (300*q[0]+200*q[1], q[0]=2, q[1]=3)", bigresult);
    assert_equal(1200 == bigresult);

    MULT_A_ADD_Q(2, 250);
    GET_A(0);
    bigresult[31:0] = result;
    $display("result: %d =2200(300*q[0]+200*q[1]+250*q[2], q[0]=2, q[1]=3, q[2]=4)", bigresult);
    assert_equal(2200 == bigresult);

    $display("----------------");
    do_command_task(`OP_A_MULT, 1073741824, 1610612736);
    $display("accu a mult result hi: %d", result);
    bigresult[63:32] = result;
    GET_A(0);
    bigresult[31:0] = result;
    $display("accu a mult result: %d =1729382256910270464 (1073741824*1610612736)", bigresult);
    assert_equal(64'd1729382256910270464 == bigresult);

    $display("----------------");
    MULT_A_ADD(100, 300);
    $display("accu a mult add result hi: %d", result);
    bigresult[63:32] = result;
    GET_A(0);
    bigresult[31:0] = result;
    $display("accu a mult add result: %d =1729382256910300464 (1729382256910270464+100*300)",
bigresult);
    assert_equal(64'd1729382256910300464 == bigresult);

    $display("----------------");
    do_command_task(`OP_A_MULT, 100, 300);
    GET_A(1);
    $display("accu a shifted result: %d =15000 (100*300/2^1)",  result);
    assert_equal(15000 == result);

    do_command_task(`OP_A_MULT, 1073741824, -1610612736);
    GET_A(48);
    $display("accu a shifted result: %d =-6144 (1073741824*1610612736/2^48)",  res);
    assert_equal(-6144 == res);

    $display("----------------");
    do_command_task(`OP_B_MULT, 100, 300);
    GET_B(2);
    $display("accu b shifted result: %d =7500 (100*300/2^2)",  result);
    assert_equal(7500 == result);

    $display("----------------");
    MULT_B_ADD(200, 150);
    $display("accu b mult add result hi: %d", result);
    bigresult[63:32] = result;
    GET_B(1);
    bigresult[31:0] = result;
    $display("accu b mult add result: %d =30000 (100*300+200*150)/2^1", bigresult);
    assert_equal(30000 == bigresult);

    $display("----------------");
    /*do_command_task(`OP_B_LOAD, -1, -1000000);


    for (i=0; i < 8; i=i+1) begin
       do_command_task(`OP_B_MULT_ADD, 100, 1000);
    end
    do_command_task(`OP_B_GET, 16, 0);
```

```
    $display("b mult add result %d= (%d)", res, (-1000000+8*100*1000)>>16);
    */

    LOAD_B(33554432);
    LOAD_A(33554432);
    MULT_A_ADD(2, 0);
    MULT_A_ADD(0, -116);
    MULT_B_ADD(0, 0);
    MULT_B_ADD(0, -116);
    MULT_A_ADD(0, 852);
    MULT_A_ADD(0, -1836);
    MULT_B_ADD(0, 852);
    MULT_B_ADD(0, -1836);
    MULT_A_ADD(0, 8148);
    MULT_A_ADD(0, -20612);
    MULT_B_ADD(0, 8148);
    MULT_B_ADD(0, -20612);
    MULT_A_ADD(0, 26296);
    MULT_A_ADD(0, -149956);
    MULT_B_ADD(0, 26296);
    MULT_B_ADD(0, -149956);
    MULT_A_ADD(0, 300152);
    MULT_A_ADD(0, 149956);
    MULT_B_ADD(0, 300152);
    MULT_B_ADD(0, 149956);
    MULT_A_ADD(0, 26296);
    MULT_A_ADD(0, 20612);
    MULT_B_ADD(0, 26296);
    MULT_B_ADD(0, 20612);
    MULT_A_ADD(0, 8148);
    MULT_A_ADD(0, 1836);
    MULT_B_ADD(0, 8148);
    MULT_B_ADD(0, 1836);
    MULT_A_ADD(0, 852);
    MULT_A_ADD(-4, 116);
    MULT_B_ADD(0, 852);
    MULT_B_ADD(0, 116);

    GET_A(20);//=31
    $display("a res:%d=31", res);
    assert_equal(31 == res);
    GET_B(20);//=32
    $display("b res:%d=32", res);
    assert_equal(32 == res);

    $display("----------------");
    ABS(-10);
    $display("abs result %d=10", result);
    assert_equal(10 == res);

    ABS(100);
    $display("abs result %d=100", result);
    assert_equal(100 == res);

    // test clip
    $display("----------------");
    CLIP_16(-10);
    $display("clip result %d=>%b", res, result);
    assert_equal(-10 == res);

    CLIP_16(-32768);
    $display("clipresult %d=>%b", res, result);
    assert_equal(-32768 == res);

    CLIP_16(-32768*2);
    $display("clipresult %d=>%b", res, result);
    assert_equal(-32768 == res);

    CLIP_16(32768);
    $display("clipresult %d=>%b", res, result);
    assert_equal(32767 == res);

    // test clz
    $display("----------------");

    value = 32'b1111_1111_1111_1111_1111_1111_1111_1111;

    for (i=0; i <= 32; i=i+1) begin
```

```
1        CLZ(value);
2        $display("clz result: %d=%d", result, i);
3        assert_equal(i == res);
4        value = value>>1;
5     end
6
7     #1000 $stop;
8     $finish;
9  end
10
11 endmodule
```

Quelltext 19: 64-Bit-ALU Testbench (mult64_tb.v)

H.4 Diverse Verilog-Quelltexte

H.4.1 Opencore PS/2 Wishbone-zu-Avalon-Wrapper

```
/*
*
* Wrapper for Opencores PS/2-Core from Wishbone to Avalon
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
module oc_ps2_master(
   input    wire  clk_i,
   input    wire  reset_i,
   input    wire  [0:0]address_i,
   input    wire  [7:0]writedata_i,
   output   wire  [7:0]readdata_o,
   input    wire  write_i,
   input    wire  chipselect_i,
   output   wire  irq_o,
   output   wire  waitrequest_n_o,
   inout    wire  ps2_clk_o,
   inout    wire  ps2_dat_io
);
// registers
// 0 is data register
// 1 is status register (read)
// 1 is control register (write)

// control register bits
// 0 = clear input buffer if bit is 0 ?
// 3 = error clear if bit is 0 ?
// 6 = irq rx enable
// 7 = irq tx enable

// status register bits
// 0 = busy
// 1 = input buffer full
// 2 = output buffer full
// 3 = parity or frame error
// 4,5 = reserved
// 6 = irq rx enable
// 7 = irq tx enable

ps2_wb the_ps2_wb(
.wb_clk_i(clk_i),
.wb_rst_i(reset_i),
.wb_dat_i(writedata_i),
.wb_dat_o(readdata_o),
.wb_adr_i(address_i),
.wb_stb_i(chipselect_i),
.wb_we_i(write_i),
.wb_ack_o(waitrequest_n_o),
.irq_o(irq_o),
.ps2_clk(ps2_clk_o),
.ps2_dat(ps2_dat_io)
);

endmodule
```

Quelltext 20: Opencore PS/2 Wishbone-zu-Avalon-Wrapper (oc_ps2_master.v)

H.4.2 Opencore SPI Wishbone-zu-Avalon-Wrapper

```
/*
*
* Opencore SPI IP-Core Wishbone to Avalon Wrapper
```

```
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
`include "spi_defines.v"

module spi_master(
   input    clk,
   input    reset,
   input    [4:0]address,
   input    [31:0]writedata,
   output   [31:0]readdata,
   input    write,
   input    chipselect,
   output   irq,
   output   waitrequest_n,
   input    [3:0]byteenable,

   output   [7:0]spi_ss,
   output   spi_sclk,
   output   spi_mosi,
   input    spi_miso
);

/*parameter SPI_SLAVE_SELECT_LINES  8

localparam SPI_SS_NB = `SPI_SS_NB-1;

*/
wire wb_err_o;

spi_top the_spi_top(

   .wb_clk_i(clk),
   .wb_rst_i(reset),
   .wb_adr_i(address),
   .wb_dat_i(writedata),
   .wb_dat_o(readdata),
   .wb_we_i(write),
   .wb_stb_i(chipselect),
   .wb_cyc_i(chipselect),
   .wb_ack_o(waitrequest_n),
   .wb_int_o(irq),
   .wb_sel_i(byteenable),
   .wb_err_o(wb_err_o),

   // SPI signals
   .ss_pad_o(spi_ss),
   .sclk_pad_o(spi_sclk),
   .mosi_pad_o(spi_mosi),
   .miso_pad_i(spi_miso)
);

endmodule
```

Quelltext 21: Opencore SPI Wishbone-zu-Avalon-Wrapper (oc_spi_master_top.v)

H.4.3 Opencore I2C-Master Wishbone-zu-Avalon-Wrapper

```
module i2c_master(
   input wire clk,
   input wire reset_n,
   input wire [2:0] address,
   input wire [7:0] writedata,
   output   wire [7:0] readdata,
   input wire write,
   input wire chipselect,
   output   wire irq,
   output   wire waitrequest_n,
   inout wire i2c_scl,
   inout wire i2c_sda
);

// parameters
```

```
parameter DRIVE_SCLK = 0; // logic value if master (me) not drive the pin up/down

wire scl_pad_o;
wire scl_padoen_o;
wire scl_pad_i;

wire sda_pad_o;
wire sda_padoen_o;
wire sda_pad_i;

// wrapp signals
assign i2c_sda    = sda_padoen_o ? 1'bZ : sda_pad_o;
assign i2c_scl    = scl_padoen_o ? (DRIVE_SCLK?1'b1:1'bZ) : scl_pad_o; // Tristate or Dirve Logic
assign sda_pad_i  = i2c_sda;
assign scl_pad_i  = i2c_scl;

// create i2c master top instance
i2c_master_top the_i2c_master_top(
   .wb_clk_i(clk),
   .wb_rst_i(1'b0),
   .arst_i(reset_n),
   .wb_adr_i(address),
   .wb_dat_i(writedata),
   .wb_dat_o(readdata),
   .wb_we_i(write),
   .wb_stb_i(chipselect),
   .wb_cyc_i(chipselect),
   .wb_ack_o(waitrequest_n),
   .wb_inta_o(irq),

   .scl_pad_i(scl_pad_i),
   .scl_pad_o(scl_pad_o),
   .scl_padoen_o(scl_padoen_o),
   .sda_pad_i(sda_pad_i),
   .sda_pad_o(sda_pad_o),
   .sda_padoen_o(sda_padoen_o)
    );
endmodule
```

Quelltext 22: Opencore I2C Wishbone-zu-Avalon-Wrapper (i2c_master.v)

H.4.4 16-24-Bit Pixel-Converter

```
/*
*
* 16-32-Bit Pixel-Converter IP-Core
*
* For converting (expanding) 16-Bit Hi-Color to 24-Bit True-Color
*
* (c) 2008-2011 by Birger Zimmermann
*
*/
`timescale 1ns/1ns
module lcd_16bit_top(
  // global signals
  input  clk,
  input  reset_n,

  // Avalon-ST Sink
  output ready_out,
  input  valid_in,
  input  [15:0] data_in,
  input  sop_in,
  input  eop_in,
  input [0:0] empty_in,

  // Avalon-ST Source
  input  ready_in,
  output valid_out,
  output [23:0] data_out,
  output sop_out,
  output eop_out,
  output [1:0] empty_out
);
```

```
1
2  assign ready_out  = ready_in;
3  assign valid_out  = valid_in;
4  assign sop_out    = sop_in;
5  assign eop_out    = eop_in;
6  assign empty_out  = empty_in;
7
8  //werden meist je fünf Bit für die Anteile Rot und Blau und sechs Bit für Grün
9  //Bit    15    14    13    12    11    10    09    08    07 06  05  04  03  02  01 00
10 //Data   R  R  R  R  R  G  G  G  G  G  G  B  B  B  B  B
11
12 // data_in is 16 bit
13 //Bit 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
14 //Bit    07 06    05    04    03    02    01 00 15    14    13  12  11  10  09  08
15 //Data   G  G  G  B  B  B  B     B  R  R  R  R  R  G  G  G
16
17 wire  [7:0]b         = {data_in[12:8], 3'b0};
18 wire  [7:0]r         = {data_in[7:3], 3'b0};
19 wire  [7:0]g         = {data_in[2:0], data_in[15:13], 2'b0};
20 assign   data_out    = {b,g,r};
21 endmodule
```

Quelltext 23: 16-24-Bit Pixel-Converter (lcd_16bit_top.v)

H.4.5 Touchpanel Clock-Multiplexer

```
22 /*
23 *
24 * Simple Multiplexer
25 *
26 * (c) 2008-2011 by Birger Zimmermann
27 *
28 */
29 module touch_panel_clock_mux(
30    input wire spi_ss_n_i,
31    input wire spi_sclk_i,
32    input wire i2c_enable_n_i,
33    input wire i2c_sclk_i,
34    output wire sclk_o
35 );
36
37 assign sclk_o = ~spi_ss_n_i ? spi_sclk_i: (~i2c_enable_n_i ? i2c_sclk_i: 1'b0);
38
39 endmodule
```

Quelltext 24: Touchpanel Clock-Multiplexer touch_panel_clock_mux.v

I C-Quelltexte

I.1 64-Bit-ALU Header-Datei

```
40 /*
41  * Nios II 64-Bit-ALU custom instructions for extended math operations
42  *
43  * Copyright 2008-2011 by Birger Zimmermann, all rights reserved.
44  *
45  * $Id: $
46  */
47 #ifndef ALU64_H_
48 #define ALU64_H_
49
50 #define ALU64_CI_N 0x00000000 // custom instruction number
51 #define ALU64_CI_N_MASK ((1<<4)-1)
52 #define ALU64_CI(n,A,B) __builtin_custom_inii(ALU64_CI_N+(n&ALU64_CI_N_MASK),(A),(B))
```

```
#define ALU64_CI_2(n,A) __builtin_custom_ini(ALU64_CI_N+(n&ALU64_CI_N_MASK),(A))

// helper struct
typedef union {
      long long sum;
      unsigned int r[2];
   } hilo_t;

/**
 * Load both (A and B) accumulators
 */
static __inline void ALU64_LOAD_BOTH(long long _sum){
   hilo_t x;
   x.sum = _sum;
   ALU64_CI(13, x.r[1], x.r[0]);
}

/**
 * Clip 32-Bit value to 16-Bit
 * return _i>32767 ? 32767:(_i<-32768?-32768:_i)
 */
static __inline short ALU64_CLIP_TO_SHORT(int _i){
   return (short)ALU64_CI(12, _i, 0);
}

/**
 * Abs
 */
static __inline int ALU64_FASTABS(int _x){
   return ALU64_CI_2(10, _x);
}

/**
 * Count leading zeros
 */
static __inline int ALU64_CLZ(int _x){
   return ALU64_CI_2(11, _x);
}
/**
 * Store factor in internal memory
 */
static __inline void ALU64_STORE(int _reg, int _value){
   ALU64_CI(14, _reg, _value);
}

/**
 * Load accu A with 64-bit
 */
static __inline void ALU64_A_LOAD(long long _sum){
   hilo_t x;
   x.sum = _sum;
   ALU64_CI(0, x.r[1], x.r[0]);
}

/**
 * Multiply _x*_y and set result in accu A
 * return upper 32-bit
 */
static __inline int ALU64_A_MULT_LOAD(int _x, int _y){
   return ALU64_CI(2, _x, _y);
}

/**
 * Multiply _x*_y and add result to accu A
 */
static __inline void ALU64_A_MULT_ADD(int _x, int _y){
   ALU64_CI(4, _x, _y);
}

/**
 * Multiply with coeffizient and add to accu A
 */
static __inline void ALU64_A_MULT_ADD_COEF(int _coef, int _y){
   ALU64_CI((6), _coef, _y);
}

/**
 * Get accu A 64-bit result shifted by _shift to a 32-bit result
```

```
 */
static __inline int ALU64_A_GET(int _shift){
   return ALU64_CI_2(8, _shift);
}

/**
 * Load accu A and add _x*_y
 * return 64-Bit result
 */
static __inline long long ALU64_A_MADD(long long _sum, int _x, int _y){
   hilo_t x;

   x.sum = _sum;

   // load sum
   ALU64_CI(0, x.r[1], x.r[0]);
   x.r[1] = ALU64_CI(4, _x, _y); // hi
   x.r[0] = ALU64_CI_2(8, 0);    // lo

   return x.sum;
}

/**
 * Load accu B with 64-bit
 */
static __inline void ALU64_B_LOAD(long long _sum){
   hilo_t x;
   x.sum = _sum;
   ALU64_CI((0|1), x.r[1], x.r[0]);
}

/**
 * Multiply _x*_y and set result in accu B
 * return upper 32-bit
 */
static inline int ALU64_B_MULT_LOAD(int _x, int _y){
   return ALU64_CI((2|1), _x, _y);
}
/**
 * Multiply _x*_y and add result to accu B
 */
static __inline void ALU64_B_MULT_ADD(int _x, int _y){
   ALU64_CI((4|1), _x, _y);
}

/**
 * Multiply with coeffizient and add to accu B
 */
static __inline void ALU64_B_MULT_ADD_COEF(int _coef, int _y){
   ALU64_CI((6|1), _coef, _y);
}

/**
 * Get accu A 64-bit result shifted by _shift to a 32-bit result
 */
static __inline int ALU64_B_GET(int _shift){
   return ALU64_CI_2((8|1), _shift);
}

#endif /*ALU64_H_*/
```

Quelltext 25: 64-Bit-ALU Header-Datei alu64.h

I.2 Angepasste MP3-Decoder Polyphase-Routinen

```
/*****************************************************************************
 * Fixed-point MP3 decoder
 * Jon Recker (jrecker@real.com), Ken Cooke (kenc@real.com)
 * June 2003
 *
 * nios2_polyphase.c
 *
 * This is the optimized polyphase algorithm for the nios2 processor
```

```
 * using an 64-Bit-ALU custom-instruction-set
 *
 * Done by Birger Zimmermann, 2008
 ******************************************************************************/

#include "coder.h"
#include "assembly.h"

#if defined(__GNUC__) && defined(__nios2__)

/* input to Polyphase = Q(DQ_FRACBITS_OUT-2), gain 2 bits in convolution
 *  we also have the implicit bias of 2^15 to add back, so net fraction bits =
 *    DQ_FRACBITS_OUT - 2 - 2 - 15
 *  (see comment on Dequantize() for more info)
 */
#define DEF_NFRACBITS   (DQ_FRACBITS_OUT - 2 - 2 - 15)
#define CSHIFT 12 /* coefficients have 12 leading sign bits for early-terminating mulitplies */

#define MC0M(x) {\
   vLo = *(vb1+(x));    vHi = -*(vb1+(23-(x))); \
   ALU64_A_MULT_ADD_COEF(c1, vLo); ALU64_A_MULT_ADD_COEF(c2, vHi); \
   c1+=2;c2+=2;\
}

#define MC1M(x) {\
      vLo = *(vb1+x); \
      ALU64_A_MULT_ADD_COEF(coef++, vLo); \
}

#define MC2M(x) { \
         c1 = coef++;   c2 = coef++;\
         vLo = *(vb1+(x)); vHi = *(vb1+(23-(x)));\
         ALU64_A_MULT_ADD_COEF(c1, vLo); ALU64_A_MULT_ADD_COEF(c2, -vHi); \
         ALU64_B_MULT_ADD_COEF(c2, vLo);  ALU64_B_MULT_ADD_COEF(c1, vHi); \
}
/******************************************************************************
 * Function:    PolyphaseMono
 *
 * Description: filter one subband and produce 32 output PCM samples for one channel
 *
 * Inputs:      pointer to PCM output buffer
 *              number of "extra shifts" (vbuf format = Q(DQ_FRACBITS_OUT-2))
 *              pointer to start of vbuf (preserved from last call)
 *              start of filter coefficient table (in proper, shuffled order)
 *              no minimum number of guard bits is required for input vbuf
 *                (see additional scaling comments below)
 *
 * Outputs:     32 samples of one channel of decoded PCM data, (i.e. Q16.0)
 *
 * Return:      none
 *
 * TODO:        add 32-bit version for platforms where 64-bit mul-acc is not supported
 *                (note max filter gain - see polyCoef[] comments)
 ******************************************************************************/
void PolyphaseMono(short *pcm, int *vbuf, const int *coefBase)
{
   int i;
   int coef;
   int *vb1;
   int vLo, vHi, c1, c2;
   Word64 rndVal;

   rndVal = (Word64)( 1 << (DEF_NFRACBITS - 1 + (32 - CSHIFT)) );

   /* special case, output sample 0 */

   vb1 = vbuf;

   c1 = 0;  c2 = c1+1;
   ALU64_LOAD_BOTH(rndVal);
   MC0M(0)
   MC0M(1)
   MC0M(2)
   MC0M(3)
   MC0M(4)
   MC0M(5)
   MC0M(6)
   MC0M(7)
```

```
1
2     *(pcm + 0) = ALU64_CLIP_TO_SHORT(ALU64_A_GET((32-CSHIFT)+DEF_NFRACBITS));
3
4     /* special case, output sample 16 */
5     coef = 256;
6     vb1 = vbuf + 64*16;
7     ALU64_LOAD_BOTH(rndVal);
8
9     MC1M(0)
10    MC1M(1)
11    MC1M(2)
12    MC1M(3)
13    MC1M(4)
14    MC1M(5)
15    MC1M(6)
16    MC1M(7)
17
18    *(pcm + 16) = ALU64_CLIP_TO_SHORT(ALU64_A_GET((32-CSHIFT)+DEF_NFRACBITS));
19
20    /* main convolution loop: sum1L = samples 1, 2, 3, ... 15   sum2L = samples 31, 30, ... 17 */
21    coef = 16;
22    vb1 = vbuf + 64;
23    pcm++;
24
25    /* right now, the compiler creates bad asm from this... */
26    for (i = 15; i > 0; i--) {
27
28       ALU64_LOAD_BOTH(rndVal);
29       MC2M(0)
30       MC2M(1)
31       MC2M(2)
32       MC2M(3)
33       MC2M(4)
34       MC2M(5)
35       MC2M(6)
36       MC2M(7)
37
38       vb1 += 64;
39
40       *(pcm)       = ALU64_CLIP_TO_SHORT(ALU64_A_GET((32-CSHIFT)+DEF_NFRACBITS));
41       *(pcm + 2*i) = ALU64_CLIP_TO_SHORT(ALU64_B_GET((32-CSHIFT)+DEF_NFRACBITS));
42       pcm++;
43    }
44 }
45
46 #define MC0S(x) {\
47    vLo = *(vb1+(x));    vHi = -*(vb1+(23-(x))); \
48    ALU64_A_MULT_ADD_COEF(c1, vLo); ALU64_A_MULT_ADD_COEF(c2, vHi); \
49    vLo = *(vb1+32+(x)); vHi = -*(vb1+32+(23-(x))); \
50    ALU64_B_MULT_ADD_COEF(c1, vLo); ALU64_B_MULT_ADD_COEF(c2, vHi); \
51    c1+=2;c2+=2;\
52 }
53
54 #define MC1S(x) {\
55       vLo = *(vb1+x); \
56       ALU64_A_MULT_ADD_COEF(coef, vLo); \
57       vLo = *(vb1+32+x); \
58       ALU64_B_MULT_ADD_COEF(coef++,vLo); \
59 }
60
61 #define MC2S_L(x) { \
62          c1 = coef++;   c2 = coef++;\
63          vLo = *(vb1+(x)); vHi = *(vb1+(23-(x)));\
64          ALU64_A_MULT_ADD_COEF(c1, vLo); ALU64_A_MULT_ADD_COEF(c2, -vHi); \
65          ALU64_B_MULT_ADD_COEF(c2, vLo);  ALU64_B_MULT_ADD_COEF(c1, vHi); \
66 }
67
68 #define MC2S_R(x) { \
69          c1 = coef++;   c2 = coef++;\
70          vLo = *(vb1+32+(x)); vHi = *(vb1+32+(23-(x))); \
71          ALU64_A_MULT_ADD_COEF(c1, vLo); ALU64_A_MULT_ADD_COEF(c2, -vHi); \
72          ALU64_B_MULT_ADD_COEF(c2, vLo);  ALU64_B_MULT_ADD_COEF(c1, vHi); \
73 }
74 /*****************************************************************************
75  * Function:    PolyphaseStereo
76  *
77  * Description: filter one subband and produce 32 output PCM samples for each channel
78  *
```

```
 * Inputs:      pointer to PCM output buffer
 *              number of "extra shifts" (vbuf format = Q(DQ_FRACBITS_OUT-2))
 *              pointer to start of vbuf (preserved from last call)
 *              start of filter coefficient table (in proper, shuffled order)
 *              no minimum number of guard bits is required for input vbuf
 *                (see additional scaling comments below)
 *
 * Outputs:     32 samples of two channels of decoded PCM data, (i.e. Q16.0)
 *
 * Return:      none
 *
 * Notes:       interleaves PCM samples LRLRLR...
 *
 * TODO:        add 32-bit version for platforms where 64-bit mul-acc is not supported
 ************************************************************************************/
void PolyphaseStereo(short *pcm, int *vbuf, const int *coefBase){
   int i;
   int *vb1;
   int vLo, vHi, c1, c2;
   int coef;
   Word64 rndVal;

   rndVal = (Word64)( 1 << (DEF_NFRACBITS - 1 + (32 - CSHIFT)) );

   /* special case, output sample 0 */

   vb1 = vbuf;
   c1 = 0;  c2 = c1+1;

   ALU64_LOAD_BOTH(rndVal);

   MC0S(0)
   MC0S(1)
   MC0S(2)
   MC0S(3)
   MC0S(4)
   MC0S(5)
   MC0S(6)
   MC0S(7)

   *(pcm + 0) = ALU64_CLIP_TO_SHORT(ALU64_A_GET((32-CSHIFT)+DEF_NFRACBITS));
   *(pcm + 1) = ALU64_CLIP_TO_SHORT(ALU64_B_GET((32-CSHIFT)+DEF_NFRACBITS));

   /* special case, output sample 16 */
   coef = 256;
   vb1 = vbuf + 64*16;

   ALU64_LOAD_BOTH(rndVal);

   MC1S(0)
   MC1S(1)
   MC1S(2)
   MC1S(3)
   MC1S(4)
   MC1S(5)
   MC1S(6)
   MC1S(7)

   *(pcm + 2*16 + 0) = ALU64_CLIP_TO_SHORT(ALU64_A_GET((32-CSHIFT)+DEF_NFRACBITS));
   *(pcm + 2*16 + 1) = ALU64_CLIP_TO_SHORT(ALU64_B_GET((32-CSHIFT)+DEF_NFRACBITS));

   /* main convolution loop: sum1L = samples 1, 2, 3, ... 15   sum2L = samples 31, 30, ... 17 */

   vb1 = vbuf + 64;
   pcm += 2;
   coef = 16;

   /* right now, the compiler creates bad asm from this... */
   for (i = 15; i > 0; i--) {
      ALU64_LOAD_BOTH(rndVal);

      MC2S_L(0)
      MC2S_L(1)
      MC2S_L(2)
      MC2S_L(3)
      MC2S_L(4)
      MC2S_L(5)
      MC2S_L(6)
```

```
      MC2S_L(7)

      *( pcm + 0)               = ALU64_CLIP_TO_SHORT(ALU64_A_GET((32-CSHIFT)+DEF_NFRACBITS));
      *( pcm + 2*2*i + 0)     = ALU64_CLIP_TO_SHORT(ALU64_B_GET((32-CSHIFT)+DEF_NFRACBITS));

      ALU64_LOAD_BOTH(rndVal);

      coef -= 16;
      MC2S_R(0)
      MC2S_R(1)
      MC2S_R(2)
      MC2S_R(3)
      MC2S_R(4)
      MC2S_R(5)
      MC2S_R(6)
      MC2S_R(7)

      *( pcm + 1)               = ALU64_CLIP_TO_SHORT(ALU64_A_GET((32-CSHIFT)+DEF_NFRACBITS));
      *( pcm + 2*2*i + 1)     = ALU64_CLIP_TO_SHORT(ALU64_B_GET((32-CSHIFT)+DEF_NFRACBITS));

      pcm += 2;
      vb1 += 64;
   }
}
#endif
```

Quelltext 26: Angepasste MP3-Decoder Polyphase-Routinen nios2_polyphase.c

I.3 Altera SPI-Master Treiber

I.3.1 Header-Datei

```
/*
 * Altera SPI-Master Core Application Interface
 *
 * API for accessing the SPI-bus
 *
 * Copyright 2008-2011 by Birger Zimmermann, all rights reserved.
 *
 * $Id:$
 */
#ifndef ALT_SPI_MASTER_H_
#define ALT_SPI_MASTER_H_

#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>

/**
 * Transaction structure
 */
typedef struct alt_spi_ta{
   uint8_t                         *txData;
   uint32_t                        txLength;
   uint8_t                         *rxData;
   uint32_t                        rxLength;
   void                            *context;
   void (*callback)(void *);
   uint8_t                         dummy;
}alt_spi_ta_t;

typedef void (*alt_spi_callback_t)(void *);

/**
 * Device structure
 */
typedef struct{
   void           *base;              /**< port io base address  */
   uint32_t       irqNum;
   uint8_t        slaveNum;           /**< which slave to be accessed, normally 0..7  */
   alt_spi_ta_t   *transaction;
```

```
   volatile int   transactionNum;
}alt_spi_dev_t;

/**
 * init device
 */
int alt_spi_init_device(alt_spi_dev_t *_device);
/**
 * exit device
 */
void alt_spi_exit_device(alt_spi_dev_t *_device);

/**
 * Checks if a transaction is in progress
 */
bool alt_spi_transaction_active(alt_spi_dev_t *_device);
/**
 * Init a transaction
 */
void alt_spi_init_transaction(alt_spi_ta_t *_ta, void *_txData, uint32_t _txLength, void
*_rxData, uint32_t _rxLength, uint8_t _dummy, alt_spi_callback_t _callback, void *_context);
/**
 * Start transactions
 */
int alt_spi_start_transaction(alt_spi_dev_t *_device, alt_spi_ta_t *_ta, int _num);

/**
 * Read data
 */
void alt_spi_read(alt_spi_dev_t *_device, uint8_t * _readData, uint16_t _readLen, uint8_t
_dummy);
/**
 * Write constant data
 */
void alt_spi_write_constant( alt_spi_dev_t *_device,
                          uint8_t _byte,
                          uint16_t _length);
/**
 * Write data
 */
void alt_spi_write(alt_spi_dev_t *_device,
                          const uint8_t *_data,
                          uint16_t _length);
/**
 * Disable chipselect
 */
void alt_spi_cs_disable(alt_spi_dev_t *_device);
/**
 * Enable chipselect
 */
void alt_spi_cs_enable(alt_spi_dev_t *_device);

#endif /*ALT_SPI_MASTER_H_*/
```

Quelltext 27: Altera SPI-Master Treiber Header-Datei alt_spi_master.h

I.3.2 Treiber-Implementierung

```
/*
 * Altera SPI-Master Core Application Interface
 *
 * API for accessing the SPI-bus
 *
 * Copyright 2008-2011 by Birger Zimmermann, all rights reserved.
 *
 * $Id:$
 */
#include <stdio.h>
#include "alt_spi_master.h"
#include "altera_avalon_spi_regs.h"

#define DUMMY_BYTE   0xFF
static void alt_spi_start_transaction_int(alt_spi_dev_t *_device, alt_spi_ta_t *_ta){
   // set the active transaction
```

```
    _device->transaction = _ta;

    // data to transmit
   if (_ta->txLength > 0){
    IOWR_ALTERA_AVALON_SPI_TXDATA(_device->base, *_ta->txData++);
   }else
   // data to receive
   if (_ta->rxLength > 0){
    IOWR_ALTERA_AVALON_SPI_TXDATA(_device->base, _ta->dummy);
   }
}

static void alt_spi_isr(void* _context, uint32_t _irq){
  alt_spi_dev_t
   *dev = (alt_spi_dev_t *) _context;
  alt_spi_ta_t
   *ta = dev->transaction;

  // get data from rx data
  uint8_t
   rxd   = IORD_ALTERA_AVALON_SPI_RXDATA(dev->base);

  // still data to transmit
  if (ta->txLength > 0){
   ta->txLength--;
   if (ta->txLength > 0){
      // send an other byte
      IOWR_ALTERA_AVALON_SPI_TXDATA(dev->base, *ta->txData++);
      return;
   }else
   if (ta->rxLength > 0){
      // send an other dummy byte (to receive one byte)
      IOWR_ALTERA_AVALON_SPI_TXDATA(dev->base, ta->dummy);
      return;
      }
  }

  // still data to receive
  if (ta->rxLength > 0){
   ta->rxLength--;

   // store the received byte
   *ta->rxData++ = rxd;

      if (ta->rxLength > 0){
      // send an other dummy byte (to receive one byte)
      IOWR_ALTERA_AVALON_SPI_TXDATA(dev->base, ta->dummy);
      return;
      }
  }

   // end of transaction ?
   if (ta->txLength==0&&ta->rxLength==0){
   // transaction finished

   // call callback ?
   if (ta->callback != NULL)
      ta->callback(ta->context);

   dev->transactionNum--;

   // start an other transaction ?
   if (dev->transactionNum > 0){
      alt_spi_start_transaction_int(dev, ++ta);
   }else{
      // deassert cs
      IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(dev->base, 0);
      IOWR_ALTERA_AVALON_SPI_CONTROL(dev->base, 0);

   }
  }
}

void alt_spi_init_transaction(alt_spi_ta_t *_ta, void *_txData, uint32_t _txLength, void
*_rxData, uint32_t _rxLength, uint8_t _dummy, alt_spi_callback_t _callback, void *_context){
   // init transaction structure
   _ta->txData          = _txData;
   _ta->txLength     = _txLength;
```

```
    _ta->callback     = _callback;
    _ta->context        = _context;
    _ta->dummy          = _dummy;
    _ta->rxData       = _rxData;
    _ta->rxLength     = _rxLength;
}

int alt_spi_start_transaction(alt_spi_dev_t *_device, alt_spi_ta_t *_ta, int _num){

    _device->transactionNum = _num;

   IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(_device->base, 1 << _device->slaveNum);
    IOWR_ALTERA_AVALON_SPI_CONTROL (_device->base,
                ALTERA_AVALON_SPI_CONTROL_IRRDY_MSK |
                ALTERA_AVALON_SPI_CONTROL_SSO_MSK   );

    alt_spi_start_transaction_int(_device, _ta);

    return 0;
}

void alt_spi_read(alt_spi_dev_t *_device, uint8_t * _readData, uint16_t _readLen, uint8_t _dummy)
{
    void *base = _device->base;
   uint32_t status;

   IORD_ALTERA_AVALON_SPI_RXDATA(base);

   /* Keep clocking until all the data has been processed. */
   for ( ; _readLen>0; _readLen--){
       do {
       status = IORD_ALTERA_AVALON_SPI_STATUS(base);
    }
    while (((status & ALTERA_AVALON_SPI_STATUS_TRDY_MSK) == 0));

     IOWR_ALTERA_AVALON_SPI_TXDATA(base, _dummy);

     do {
       status = IORD_ALTERA_AVALON_SPI_STATUS(base);
     }
     while ((status & ALTERA_AVALON_SPI_STATUS_RRDY_MSK) == 0);

     *_readData++ =IORD_ALTERA_AVALON_SPI_RXDATA(base);
   }

   /* Wait until the interface has finished transmitting */
   do {
     status = IORD_ALTERA_AVALON_SPI_STATUS(base);
   }
   while ((status & ALTERA_AVALON_SPI_STATUS_TMT_MSK) == 0);
}

void alt_spi_write_constant( alt_spi_dev_t *_device,
                           uint8_t _byte,
                           uint16_t _length) {
   uint32_t status;
    void *base = _device->base;
   IORD_ALTERA_AVALON_SPI_RXDATA(base);

   for(;_length > 0;_length--) {

       do {
       status = IORD_ALTERA_AVALON_SPI_STATUS(base);
    }
    while ((status & ALTERA_AVALON_SPI_STATUS_TRDY_MSK) == 0);

     IOWR_ALTERA_AVALON_SPI_TXDATA(base, _byte);

     do {
       status = IORD_ALTERA_AVALON_SPI_STATUS(base);
     }
     while ((status & ALTERA_AVALON_SPI_STATUS_RRDY_MSK) == 0);

     IORD_ALTERA_AVALON_SPI_RXDATA(base);
   }

   do  {
     status = IORD_ALTERA_AVALON_SPI_STATUS(base);
```

```
  }
  while ((status & ALTERA_AVALON_SPI_STATUS_TMT_MSK) == 0);
}

void alt_spi_write(alt_spi_dev_t *_device,
                         const uint8_t *_data,
                         uint16_t _length) {
  uint32_t status;
   void *base = _device->base;
  IORD_ALTERA_AVALON_SPI_RXDATA(base);

  for(;_length > 0; _length--) {
   uint8_t data = *_data++;

   do {
      status = IORD_ALTERA_AVALON_SPI_STATUS(base);
   }
   while (((status & ALTERA_AVALON_SPI_STATUS_TRDY_MSK) == 0));

    IOWR_ALTERA_AVALON_SPI_TXDATA(base, data);

    do {
      status = IORD_ALTERA_AVALON_SPI_STATUS(base);
    }
    while ((status & ALTERA_AVALON_SPI_STATUS_RRDY_MSK) == 0);

    IORD_ALTERA_AVALON_SPI_RXDATA(base);
  }

  do  {
    status = IORD_ALTERA_AVALON_SPI_STATUS(base);
  }
  while ((status & ALTERA_AVALON_SPI_STATUS_TMT_MSK) == 0);
}

void alt_spi_cs_disable(alt_spi_dev_t *_device){
  // Wait until the interface has finished transmitting
  while ((IORD_ALTERA_AVALON_SPI_STATUS(_device->base) & ALTERA_AVALON_SPI_STATUS_TMT_MSK) == 0);

  // deselect slave IORD_ALTERA_AVALON_SPI_SLAVE_SEL(_device->base) & ~(1 << _device->cardNum)
  IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(_device->base, 0);
  IOWR_ALTERA_AVALON_SPI_CONTROL(_device->base, 0);
}

void alt_spi_cs_enable(alt_spi_dev_t *_device){
  IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(_device->base, 1 << _device->slaveNum);
  IOWR_ALTERA_AVALON_SPI_CONTROL(_device->base, ALTERA_AVALON_SPI_CONTROL_SSO_MSK);
}
void alt_spi_set_freq(alt_spi_dev_t *_device, uint32_t _freqKhz){
}

bool alt_spi_transaction_active(alt_spi_dev_t *_device){
   return _device->transactionNum > 0;
}

int alt_spi_init_device(alt_spi_dev_t *_device){
   _device->transactionNum = 0;

  IOWR_ALTERA_AVALON_SPI_CONTROL(_device->base, 0);

  alt_irq_register(_device->irqNum,
             (void*) _device,
             alt_spi_isr);

   return 0;
}

void alt_spi_exit_device(alt_spi_dev_t *_device){
   alt_irq_register(_device->irqNum, NULL, NULL);
}
```

Quelltext 28: Altera SPI-Master Treiber alt_spi_master.c

I.4 SPI-Master Treiber

I.4.1 Register Header-Datei

```
/*
 * Buffered SPI Core Application Interface
 *
 * API for accessing the SPI Bus
 *
 * Copyright 2008-2011 by Birger Zimmermann, all rights reserved.
 *
 * $Id: $
 */
#ifndef SPI_MASTER_REGS_H_
#define SPI_MASTER_REGS_H_

/**
 * Memory mapped registers
 */
#define SPI_MASTER_REG_FIFO                         0
#define SPI_MASTER_REG_FIFO_FILL_COUNTS             1
#define SPI_MASTER_REG_TRANSFER_SIZE                2
#define SPI_MASTER_REG_CONTROL                      3
#define SPI_MASTER_REG_STATUS                       4
#define SPI_MASTER_REG_SCLK_DIVIDER                 5
#define SPI_MASTER_REG_SS                           6 /*< slave select register */
#define SPI_MASTER_REG_TOKEN                        7

/**
* Fifo control and transfer register special bits (write operation)
*/
#define SPI_MASTER_CNTRL_CLEAR_RECEIVE_FIFO         (1<<31)
#define SPI_MASTER_CNTRL_HOLD_BUS_ON_EMPTY_FIFO     (1<<30)
#define SPI_MASTER_CNTRL_CLEAR_TRANSFER_FIFO        (1<<29)
#define SPI_MASTER_CNTRL_HOLD_BUS_ON_FULL_FIFO      (1<<28)
#define SPI_MASTER_CNTRL_RESET                      (1<<27)
#define SPI_MASTER_CNTRL_START_TRANSFER             (1<<26)
#define SPI_MASTER_CNTRL_JOB_H                      (1<<25)
#define SPI_MASTER_CNTRL_JOB_L                      (1<<24)
#define SPI_MASTER_LITTLE_ENDIAN                    (1<<23)
// bits to shift the job num
#define SPI_MASTER_CNTRL_JOB_SHIFT                  24

/**
 * Operations (jobs)
 */
#define SPI_MASTER_JOB_TRANSFER                     0
#define SPI_MASTER_JOB_RECEIVE                      1
#define SPI_MASTER_JOB_WAIT_FOR_TOKEN               2
#define SPI_MASTER_JOB_WAIT_WHILE_TOKEN             3

/**
 * Fifo fill size register (read operation)
 */
#define SPI_MASTER_GET_TRANSFER_FILL_COUNT(n)       (n&0xFFFF)
#define SPI_MASTER_GET_RECEIVE_FILL_COUNT(n)        (n>>16)

/**
 * Transfer register (write operation) sizes
 */
#define SPI_MASTER_START_JOB(job, size)
   ((((uint32_t)job)<<SPI_MASTER_CNTRL_JOB_SHIFT)|SPI_MASTER_CNTRL_START_TRANSFER|(size&0xFFFF))

/**
 * Status register
 */
#define SPI_MASTER_FLAG_BUSY                        (1<<2)
#define SPI_MASTER_FLAG_RECEIVE_EMPTY               (1<<1)
#define SPI_MASTER_FLAG_TRANSFER_FULL               (1<<0)
#define SPI_MASTER_FLAG_WRITE_ERROR                 (1<<3)
#define SPI_MASTER_FLAG_READ_ERROR                  (1<<4)

#define SPI_MASTER_GET_TEST_COUNT(status)           (status>>24)

#define SPI_MASTER_FIFO_SIZE                        512
```

```
#define SPI_MASTER_MAX_TRANSFER_SIZE                ((1<<16)-1)
#endif /*SPI_MASTER_REGS_H_*/
```

Quelltext 29: SPI-Master Register Header-Datei spi_master_regs.h

I.4.2 Header-Datei

```
/*
 * Buffered SPI Core Application Interface
 *
 * API for accessing the SPI Bus
 *
 * Copyright 2008-2011 by Birger Zimmermann, all rights reserved.
 *
 * $Id: $
 */
#ifndef SPI_MASTER_H_
#define SPI_MASTER_H_

#include "device/spi/spi.h"

/**
 * Init device
 */
int spi_master_init_device(spi_dev_t *_dev);
/**
 * Wait for token
 */
int spi_master_wait_for_token(spi_dev_t *_device, uint8_t _token, uint32_t _timeout);
/**
 * Wait while token
 */
int spi_master_wait_while_token(spi_dev_t *_device, uint8_t _token, uint32_t _timeout, uint8_t
*_data);
/**
 * Disable chip select
 */
int spi_master_cs_disable(spi_dev_t *_device);
/**
 * Enable chip select
 */
int spi_master_cs_enable(spi_dev_t *_device);
/**
 * Write data
 */
int spi_master_write(spi_dev_t *_device,
                         const uint8_t *_data,
                         uint16_t _length);
/**
 * Write constant
 */
int spi_master_write_constant( spi_dev_t *_device,
                         uint8_t _byte,
                         uint16_t _length);
/**
 * Read data
 */
int spi_master_read( spi_dev_t *_device,
                         uint8_t * _data,
                         uint16_t _length);
/**
 * Get the sclk frequency
 */
uint32_t spi_master_get_freq(spi_dev_t *_device);
/**
 * Set the sclk frequency
 */
uint32_t spi_master_set_freq(spi_dev_t *_device, uint32_t _clkInHz);

#ifdef DEBUG_SPI_MASTER
void spi_master_test_case(spi_dev_t *_device);
#endif

```

```
#endif /*SPI_MASTER_H_*/
```

Quelltext 30: SPI-Master Header-Datei spi_master.h

I.4.3 Treiber-Implementierung

```
/*
 * Buffered SPI Core Application Interface
 *
 * API for accessing the SPI Bus
 *
 * Copyright 2008-2011 by Birger Zimmermann, all rights reserved.
 *
 * $Id: $
 */
#include <io.h>
#include <errno.h>
#include <string.h>
#include "debug/debug.h"
#include "sys/alt_cache.h"

#include "spi_master.h"
#include "spi_master_regs.h"

#define LOWEST_FREQUENCY_HZ     400000
#define IDLE_TIMEOUT            1000000
#define CHECK_UNEVEN_ADDRESS  1
#define ERROR_UNEVEN_ADDRESS  "uneven address error"
#define ERROR_IDLE_TIMEOUT    "idle timeout error"
// Improvements:
// Direct Byte-Transfer Register ?
// master interface, direct write or read from memory

// Done:
// remove separate transfer and receive counters. one counter fits both
// wait for receive pattern function
// three operation modes 0=transfer 1=receive 2=wait for byte pattern
// hold bus bit in transfer size register until data ready
// e.g bit 30 and 14 to fill or read from fifo staled
// reset command for the transceiver force to idle state
// clear fifo flags also in transfer size register e.g bit 31 and 15

#define MIN(a,b) (a<=b?a:b)
#define MAX(a,b) (a>=b?a:b)

static inline int wait_until_idle(void * _base, uint32_t _timeout){
   /*uint32_t status;*/
   while ((/*status = */IORD(_base, SPI_MASTER_REG_STATUS))&SPI_MASTER_FLAG_BUSY &&
(_timeout--));
   return _timeout > 0?0:-1;
}

static inline void wait_until_receive_fill_count(void * _base, uint16_t _fillcount){
   while ((SPI_MASTER_GET_RECEIVE_FILL_COUNT(IORD(_base, SPI_MASTER_REG_FIFO_FILL_COUNTS)) <
_fillcount));
}

static inline uint32_t read_status(void * _base){
   return IORD(_base, SPI_MASTER_REG_STATUS);
}

/**
 * Wait until a token is send on the bus
 */
int spi_master_wait_for_token(spi_dev_t *_device, uint8_t _token, uint32_t _timeout){
   uint8_t read;

   // setup token
   IOWR(_device->base, SPI_MASTER_REG_TOKEN, _token);

   while (_timeout > 0){
      uint32_t
         len = MIN(SPI_MASTER_MAX_TRANSFER_SIZE, _timeout);
```

```
1        // wait for a running job
2        wait_until_idle(_device->base, IDLE_TIMEOUT);

3        // start token job
4        IOWR(_device->base, SPI_MASTER_REG_TRANSFER_SIZE,
5           SPI_MASTER_START_JOB(SPI_MASTER_JOB_WAIT_FOR_TOKEN, len)|SPI_MASTER_LITTLE_ENDIAN);

6        // wait until ready
7        wait_until_idle(_device->base, IDLE_TIMEOUT);

8        uint32_t
9           status = read_status(_device->base);

10       if (status&SPI_MASTER_FLAG_RECEIVE_EMPTY)
11          _timeout -= len;
12       else{
13          read = IORD(_device->base, SPI_MASTER_REG_FIFO)&0xFF;

14          return read==_token?0:-ETIME;
15       }
16    }
17    return -ETIME;
18 }
19
20 /**
21  * Wait while a token is send on the bus
22  */
23 int spi_master_wait_while_token(spi_dev_t *_device, uint8_t _token, uint32_t _timeout, uint8_t
24 *_data){
25    uint8_t read;

26    // setup token
27    IOWR(_device->base, SPI_MASTER_REG_TOKEN, _token);

28    while (_timeout > 0){
29       uint32_t
30          len = MIN(SPI_MASTER_MAX_TRANSFER_SIZE, _timeout);

31       // wait for a running job
32       wait_until_idle(_device->base, IDLE_TIMEOUT);

33       // start token job
34       IOWR(_device->base, SPI_MASTER_REG_TRANSFER_SIZE,
35          SPI_MASTER_START_JOB(SPI_MASTER_JOB_WAIT_WHILE_TOKEN, len)|SPI_MASTER_LITTLE_ENDIAN);

36       // wait until ready
37       wait_until_idle(_device->base, IDLE_TIMEOUT);

38       uint32_t
39          status = read_status(_device->base);

40       if (status&SPI_MASTER_FLAG_RECEIVE_EMPTY)
41          _timeout -= len;
42       else{
43          read = IORD(_device->base, SPI_MASTER_REG_FIFO)&0xFF;

44          // store data
45          if (_data != NULL)
46             *_data = read;

47          return 0;
48       }
49    }
50    return -ETIME;
51 }
52 /**
53  * Init the driver
54  */
55 int spi_master_init_device(spi_dev_t *_device){
56
57    // init functions
58    _device->get_freq         = &spi_master_get_freq;
59    _device->set_freq         = &spi_master_set_freq;
60    _device->cs_disable       = &spi_master_cs_disable;
61    _device->cs_enable        = &spi_master_cs_enable;
62    _device->write            = &spi_master_write;
63    _device->write_constant   = &spi_master_write_constant;
```

```
1     _device->read              = &spi_master_read;
2     _device->wait_for_token    = &spi_master_wait_for_token;
3     _device->wait_while_token  = &spi_master_wait_while_token;
4
5     // reset the transceiver
6     IOWR(_device->base, SPI_MASTER_REG_CONTROL, SPI_MASTER_CNTRL_RESET);

7     // check the register read/write
8     IOWR(_device->base, SPI_MASTER_REG_SCLK_DIVIDER, 150);

9     if (IORD(_device->base, SPI_MASTER_REG_SCLK_DIVIDER) != 150)
10       return -EIO;
11
12    spi_set_freq(_device, LOWEST_FREQUENCY_HZ);

13    return 0;
14 }
15
16 uint32_t spi_master_get_freq(spi_dev_t *_device){
17    uint32_t
18       divider = IORD(_device->base, SPI_MASTER_REG_SCLK_DIVIDER);
19    return _device->clk/(divider<<1);
20 }
21
22 uint32_t spi_master_set_freq(spi_dev_t *_device, uint32_t _clkInHz){
23    uint32_t
24       divider =  MAX((_device->clk/_device->maxClock)>>1, (_device->clk/_clkInHz)>>1);

25    IOWR(_device->base, SPI_MASTER_REG_SCLK_DIVIDER,  divider);

26    return _device->clk/(divider<<1);
27 }
28
29 int spi_master_cs_disable(spi_dev_t *_device){
30    if (wait_until_idle(_device->base, IDLE_TIMEOUT) < 0){
31       ERROR(ERROR_IDLE_TIMEOUT);
32       return -1;
33    }
34    IOWR(_device->base, SPI_MASTER_REG_SS, 0);
35    return 0;
36 }
37
38 int spi_master_cs_enable(spi_dev_t *_device){
39    if (wait_until_idle(_device->base, IDLE_TIMEOUT) < 0){
40       ERROR(ERROR_IDLE_TIMEOUT);
41       return -1;
42    }
43    IOWR(_device->base, SPI_MASTER_REG_SS, 1<<_device->slaveNum);
44    return 0;
45 }
46
47 int spi_master_write(spi_dev_t *_device,
48                         const uint8_t *_data,
49                         uint16_t _length){
50   // assert aligned address
51   #if CHECK_UNEVEN_ADDRESS
52    if ((((uint32_t)_data)&3)!=0){
53       ERROR(ERROR_UNEVEN_ADDRESS);
54       return -1;
55    }
56    #endif
57    uint32_t n;
58    uint32_t *data = (uint32_t *)_data;

59    while (_length > 0){
60       uint16_t
61          transfer = MIN(SPI_MASTER_FIFO_SIZE, _length);

62       if (wait_until_idle(_device->base, IDLE_TIMEOUT) < 0){
63          ERROR(ERROR_IDLE_TIMEOUT);
64          return -1;
65       }
66       uint32_t trans = (transfer+3)>>2;
67       // transfer is stalled as long as no data is there
68       IOWR(_device->base, SPI_MASTER_REG_TRANSFER_SIZE,
69          SPI_MASTER_START_JOB(SPI_MASTER_JOB_TRANSFER, transfer)|SPI_MASTER_LITTLE_ENDIAN);

70       for (n=trans; n > 0; n--){
```

```
1           IOWR(_device->base, SPI_MASTER_REG_FIFO, *data++);
2        }
3        _length -= transfer;
4     }
5
6     return 0;
7  }
8
9  int spi_master_write_constant( spi_dev_t *_device,
10                            uint8_t _byte,
11                            uint16_t _length){
12    uint8_t
13       data[32];

14    memset(data, _byte, MIN(sizeof(data), _length));

15    while (_length > 0){
16       uint16_t transfer = MIN(sizeof(data), _length);

17       if (spi_write(_device, data, transfer) < 0)
18          return -1;

19       _length -= transfer;
20    }

21    return 0;
22 }
23
24 int spi_master_read( spi_dev_t *_device,
25                            uint8_t * _data,
26                            uint16_t _length){
27
28    int
29       n;

30    if (wait_until_idle(_device->base, IDLE_TIMEOUT) < 0){
31       ERROR(ERROR_IDLE_TIMEOUT);
32       return -1;
33    }

34    if (_length < 4){
35       IOWR(_device->base, SPI_MASTER_REG_TRANSFER_SIZE,
36          SPI_MASTER_START_JOB(SPI_MASTER_JOB_RECEIVE,
37          _length)|SPI_MASTER_CNTRL_HOLD_BUS_ON_EMPTY_FIFO|SPI_MASTER_LITTLE_ENDIAN);
38       //wait_until_idle(_device->base);
39       uint32_t
40          data = IORD(_device->base, SPI_MASTER_REG_FIFO);
41       for (; _length > 0; _length--){
42          *_data++ = data&0xFF;
43          data = data>>8;
44       }

45       return 0;
46    }

47    #if CHECK_UNEVEN_ADDRESS
48    if ((((uint32_t)_data)&3)!=0){
49       ERROR(ERROR_UNEVEN_ADDRESS);
50       return -1;
51    }
52    #endif

53    int
54       alignedLength = _length&(~3);

55    //if (alignedLength > 0){
56    uint32_t *alignedData = (uint32_t *)_data;
57    //_length -= alignedLength;

58    do{
59       uint16_t
60          transfer = MIN(SPI_MASTER_FIFO_SIZE, _length);

61       IOWR(_device->base, SPI_MASTER_REG_TRANSFER_SIZE,
62          SPI_MASTER_START_JOB(SPI_MASTER_JOB_RECEIVE,
63          transfer)|SPI_MASTER_CNTRL_HOLD_BUS_ON_EMPTY_FIFO|SPI_MASTER_LITTLE_ENDIAN);
64
65       //wait_until_idle(_device->base);
```

```
1        // wait until receive fifo size is transfer/2
2        wait_until_receive_fill_count(_device->base, transfer>>1);

3        // read from fifo
4        for (n=transfer>>2; n > 0; n--){
5           *alignedData++ = IORD(_device->base, SPI_MASTER_REG_FIFO);
6        }

7        alignedLength -= transfer&(~3);
8     }  while (alignedLength > 0);

9     _data = (uint8_t *)alignedData;
10    _length = _length&3;

11    if (_length > 0){
12       uint32_t
13          data = IORD(_device->base, SPI_MASTER_REG_FIFO);

14       for (; _length > 0; _length--){
15          *_data++ = data&0xFF;
16          data = data>>8;
17       }
18    }
19    //}
20    return 0;
21 }
```

Quelltext 31: SPI-Master Treiber-Implementierung spi_master.c

I.5 MMC/SD-Karten-Treiber

I.5.1 MMC/SD-Karten Header-Datei (Protokoll-Konstanten)

```
27 /*
28  * MMC Card interface
29  *
30  * API for accessing SD/MMC-cards
31  *
32  * Copyright 2005-2007 by Birger Zimmermann, all rights reserved.
33  *
34  * $Id:$
35  */
36 #ifndef MMC_CARD_H_
37 #define MMC_CARD_H_
38
39 #include <stdint.h>
40 #include <stdbool.h>
41
42 #if defined(__DOXYGEN__)
43    #define  __attribute__(x)
44 #endif
45
46 #define MMC_BLOCK_SIZE              512         /* this is always 512 */
47 #define MMC_BLOCK_SIZE_LD           9
48 #define MMC_LOWEST_FREQUENCY_HZ     400000
49
50 /**
51  * @name MMC SPI mode result codes
52  * @{
53  */
54 typedef uint8_t   mmc_r1_t;
55 typedef uint16_t  mmc_r2_t;
56 typedef uint32_t  mmc_r3_t;
57 /** @} */
58
59 #define MMC_VDD_145_150 0x00000001  /* VDD voltage 1.45 - 1.50 */
```

```
1  #define MMC_VDD_150_155 0x00000002  /* VDD voltage 1.50 - 1.55 */
2  #define MMC_VDD_155_160 0x00000004  /* VDD voltage 1.55 - 1.60 */
3  #define MMC_VDD_160_165 0x00000008  /* VDD voltage 1.60 - 1.65 */
4  #define MMC_VDD_165_170 0x00000010  /* VDD voltage 1.65 - 1.70 */
5  #define MMC_VDD_17_18   0x00000020  /* VDD voltage 1.7 - 1.8 */
6  #define MMC_VDD_18_19   0x00000040  /* VDD voltage 1.8 - 1.9 */
7  #define MMC_VDD_19_20   0x00000080  /* VDD voltage 1.9 - 2.0 */
8  #define MMC_VDD_20_21   0x00000100  /* VDD voltage 2.0 ~ 2.1 */
9  #define MMC_VDD_21_22   0x00000200  /* VDD voltage 2.1 ~ 2.2 */
10 #define MMC_VDD_22_23   0x00000400  /* VDD voltage 2.2 ~ 2.3 */
11 #define MMC_VDD_23_24   0x00000800  /* VDD voltage 2.3 ~ 2.4 */
12 #define MMC_VDD_24_25   0x00001000  /* VDD voltage 2.4 ~ 2.5 */
13 #define MMC_VDD_25_26   0x00002000  /* VDD voltage 2.5 ~ 2.6 */
14 #define MMC_VDD_26_27   0x00004000  /* VDD voltage 2.6 ~ 2.7 */
15 #define MMC_VDD_27_28   0x00008000  /* VDD voltage 2.7 ~ 2.8 */
16 #define MMC_VDD_28_29   0x00010000  /* VDD voltage 2.8 ~ 2.9 */
17 #define MMC_VDD_29_30   0x00020000  /* VDD voltage 2.9 ~ 3.0 */
18 #define MMC_VDD_30_31   0x00040000  /* VDD voltage 3.0 ~ 3.1 */
19 #define MMC_VDD_31_32   0x00080000  /* VDD voltage 3.1 ~ 3.2 */
20 #define MMC_VDD_32_33   0x00100000  /* VDD voltage 3.2 ~ 3.3 */
21 #define MMC_VDD_33_34   0x00200000  /* VDD voltage 3.3 ~ 3.4 */
22 #define MMC_VDD_34_35   0x00400000  /* VDD voltage 3.4 ~ 3.5 */
23 #define MMC_VDD_35_36   0x00800000  /* VDD voltage 3.5 ~ 3.6 */
24 #define MMC_CARD_BUSY   0x80000000  /* Card Power up status bit */
25
26
27 #define MMC_CSD_SIZE            16
28 #define MMC_DUMMY               0xFF
29
30 /**
31  * @name MMC SPI r1 card response codes
32  * @{
33  */
34 #define MMC_R1_NO_ERROR         0x00     // no error result
35 #define MMC_R1_BUSY             0x01     // busy result code
36 #define MMC_R1_IN_IDLE_STATE    (1<<0)   // in idle state
37 /** @} */
38
39 /**
40  * @name MMC commands
41  * First bit 0 (start bit), second 1 (transmission bit); CMD-number + 0ffset 0x40
42  * @{
43  */
44 #define MMC_GO_IDLE_STATE             0 // CMD0
45 #define MMC_SEND_OP_COND              1 // CMD1
46 #define MMC_READ_CSD                  9
47 #define MMC_SEND_CID                  10
48 #define MMC_STOP_TRANSMISSION         12
49 #define MMC_SEND_STATUS               13
50 #define MMC_SET_BLOCKLEN              16
51 #define MMC_READ_SINGLE_BLOCK         17
52 #define MMC_READ_MULTIPLE_BLOCK       18
53 #define MMC_CMD_WRITEBLOCK            20
54 #define MMC_WRITE_SINGLE_BLOCK        24
55 #define MMC_WRITE_MULTIPLE_BLOCK      25
56 #define MMC_WRITE_CSD                 27
57 #define MMC_SET_WRITE_PROT            28
58 #define MMC_CLR_WRITE_PROT            29
59 #define MMC_SEND_WRITE_PROT           30
60 #define MMC_TAG_SECTOR_START          32
61 #define MMC_TAG_SECTOR_END            33
62 #define MMC_UNTAG_SECTOR              34
63 #define MMC_TAG_EREASE_GROUP_START    35
64 #define MMC_TAG_EREASE_GROUP_END      36
65 #define MMC_UNTAG_EREASE_GROUP        37
66 #define MMC_EREASE                    38
67 #define MMC_READ_OCR                  58
68 #define MMC_CRC_ON_OFF                59
69 /** @} */
70
71 /**
72  * @name MMC Protocol Tokens
73  * @{
74  */
75 #define MMC_TOKEN_BUSY                          0x00  /**< data out line low */
76 #define MMC_TOKEN_START_BLOCK_READ              0xfe  /**< data token start byte, Start Single
77 Block Read */
78 #define MMC_TOKEN_START_MULTIPLE_BLOCK_READ     0xfe  /**< data token start byte, Start Multiple
```

```
Block Read */
#define MMC_TOKEN_START_BLOCK_WRITE            0xfe  /**< data token start byte, Start Single
Block Write */
#define MMC_TOKEN_START_MULTIPLE_BLOCK_WRITE   0xfc  /**< data token start byte, Start Multiple
Block Write */
#define MMC_TOKEN_STOP_MULTIPLE_BLOCK_WRITE    0xfd  /**< data token stop byte, Stop Multiple
Block Write */

#define MMC_TOKEN_DATA_RESPONSE_MASK           0x11
#define MMC_TOKEN_DATA_RESPONSE_STATUS_MASK    0x0E
#define MMC_TOKEN_DATA_RESPONSE_ACCEPTED       0x04  /**<xxx0 010 1 - data accepted. */
#define MMC_TOKEN_DATA_RESPONSE_CRC_ERROR      0x0A  /**<xxx0 101 1 - data rejected due to a CRC
error. */
#define MMC_TOKEN_DATA_RESPONSE_WRITE_ERROR    0x0C  /**<xxx0 110 1 - data rejected due to a
Write error. */
/** @} */

/**
 * @name MMC SPI Timings
 * @{
 */
#define  MMC_TIMING_MAX_CR               8     // max 8*8 Clockcycles
#define MMC_TIMING_GO_IDLE               10
#define MMC_TIMING_WAIT_GO_IDLE          4000 // 4 ms
#define MMC_TIMING_SEND_OP               200
#define MMC_TIMING_WAIT_SEND_OP          4000 // 4 ms
/** @} */

/**
 * CID (Card Identification Number: Card ID Register)
 * The CID register is 16 bytes long and contains a unique card identification number as shown in
the table
 * below. It is programmed during card manufacturing and can not be changed by MultiMediaCard
hosts.
 */
typedef struct mmc_cid{
   uint8_t     manufacturerID; /**< The manufacturer IDs are controlled and assigned by the
MultiMediaCard Association. */
   uint16_t applicationID; /**< Identifies the card OEM and/or the card contents. */
   uint8_t     productName[6];   /**< 6 ASCII characters long */
   /**
    * The product revision is composed of two Binary Coded Decimal (BCD) digits,
      four bits each, representing an n.m
    * revision number. The n is the most significant nibble and the m is the least
      significant nibble.
    * Example: The PRV binary value filed for product revision 6.2 will be: 0110 0010.
   */
   uint8_t     productRevision;
   uint8_t     serialNumber[4];
   /**
    * BCD date - mm/yy (offset from 1997)
    * As an example, this field value for a March 2001 manufacturing date will be: 0011 0100.
    */
   uint8_t     manufacturingDateCode;  /**< Manufacturing date - mm/yy (offset from 1997) */
   uint8_t     crc;
} __attribute__ ((packed)) mmc_cid_t;

/**
 * CSD (Card Specific data: Card Specific data Register)
 * The CSD register contains all the configuration information required in order to
   access the card data.
 */
typedef struct mmc_csd{
   /**
    * TRAN_SPEED—Table 3-13 defines the maximum data transfer rate TRAN_SPEED.
    * Table 3-13. Maximum Data Transfer Rate Definition
    * TRAN_SPEED Bit Code
    * 2:0 transfer rate unit
    * 0=100kbit/s, 1=1Mbit/s, 2=10Mbit/s, 3=100Mbit/s, 4... 7=reserved
    * 6:3 time value
    * 0=reserved, 1=1.0, 2=1.2, 3=1.3, 4=1.5, 5=2.0, 6=2.5, 7=3.0,
    * 8=3.5, 9=4.0, A=4.5, B=5.0, C=5.5, D=6.0, E=7.0, F=8.0
    * 7 Reserved
    */
   uint32_t tran_speed;
   /**
    * C_SIZE (Device Size)-This parameter is used to compute the card capacity.
      The memory capacity of the
```

```
     * card is computed from the entries C_SIZE, C_SIZE_MULT and READ_BL_LEN
     */
    uint32_t c_size;
    /**
     * C_SIZE_MULT (Device Size Multiplier)-This parameter is used for coding a factor MULT for
     * computing the total device size (see C_SIZE).
       The factor MULT is defined as 2^(C_SIZE_MULT+2).
     */
    uint8_t     c_size_mult;
    /**
     * READ_BL_LEN- The data block length is computed as 2^READ_BL_LEN.
       The block length might therefore be
     * in the range 1, 2, 4...2048 bytes
     */
    uint8_t     read_bl_len;
    /**
     * READ_BL_PARTIAL-Defines whether partial block sizes can be used in block read commands.
     * READ_BL_PARTIAL=0 means that only the READ_BL_LEN block size can be used for block-oriented
     * data transfers.
     * READ_BL_PARTIAL=1 means that smaller blocks can be used as well. The minimum block size
will be
     * equal to minimum addressable unit (one byte)
     */
    bool        read_bl_partial;
    /**
     * READ_BLK_MISALIGN-Defines if the data block to be read by one command can be spread over
     * more than one physical block of the memory device.
       The size of the memory block is defined in
     * READ_BL_LEN.
     * READ_BLK_MISALIGN=0 signals that crossing physical block boundaries is invalid.
     * READ_BLK_MISALIGN=1 signals that crossing physical block boundaries is allowed.
     */
    bool     read_bl_misalign;
    /**
     * TAAC-Defines the asynchronous part (relative to the MultiMediaCard clock (CLK))
       of the read access time.
     */
    uint32_t taac; //[ns]
    /**
     * NSAC-Defines the worst case for the clock dependent factor of the data access time.
       The unit for NSAC
     * is 100 clock cycles. Therefore, the maximal value for the clock dependent part of the
       read access time is
     * 25.5k clock cycles.
     * The total read access time NAC as expressed in the Table 5-12 is the sum of TAAC and NSAC.
       It has to be
     * computed by the host for the actual clock rate. The read access time should be interpreted
       as a typical
     * delay for the first data bit of a data block or stream from the end bit on the read
       commands.
     */
    uint16_t nsac; // [cycles]
    uint8_t     r2w_factor;
} mmc_csd_t;


#endif /*MMC_CARD_H_*/
```

Quelltext 32: MMC/SD-Karten Protokoll-Header-Datei mmc_card.h

I.5.2 Treiber-Implementierung

```
/*
 * MMC Card interface
 *
 * API for accessing SD/MMC-cards
 *
 * Copyright 2008-2011 by Birger Zimmermann, all rights reserved.
 *
 * $Id: $
 */
#include <unistd.h>
#include "mmc_spi_device.h"
```

```
1  #include "device/spi/spi.h"
2
3  #define min(a,b) (a<=b?a:b)
4  #define DUMMY_CLOCK_CLYCLES(spiDev) spi_write_byte(spiDev, MMC_DUMMY)
5
6  static inline void mmc_spi_end_transaction(mmc_spi_dev_t *_device){
7      spi_cs_disable(_device->spiDevice);
8      DUMMY_CLOCK_CLYCLES(_device->spiDevice);
9  }
10
11 static inline mmc_res_t mmc_spi_go_idle_r1(mmc_spi_dev_t *_device, mmc_r1_t *_r1);
12
13 static inline mmc_res_t mmc_spi_map_error(mmc_r1_t _error){
14    return _error==MMC_R1_NO_ERROR?MMC_SUCCESS:MMC_IOERROR;
15 }
16
17 const static uint8_t __attribute__ ((aligned))
18       start_single_write_tokens[2] = {MMC_DUMMY, MMC_TOKEN_START_BLOCK_WRITE} ;
19
20 const static uint8_t __attribute__ ((aligned))
21       start_multi_write_tokens[2] = {MMC_DUMMY, MMC_TOKEN_START_MULTIPLE_BLOCK_WRITE};

22 /**
23 * Computation of CRC-7 0x48 polynom (x**7 + x**3 + 1)
24 * @return the calculated crc
25 */
26 #if (MMC_CRC_CHECK)
27 inline uint8_t mmc_crc7_update(uint8_t crc, uint8_t c){
28    uint8_t
29       ibit;
30    for (ibit = 0; ibit < 8; ibit++){
31       crc <<= 1;
32       if ((c ^ crc) & 0x80)
33          crc ^= 0x09;
34       c <<= 1;
35    }

36    crc &= 0x7F;
37    return crc;
38 }
39 uint8_t mmc_crc7(uint8_t crc, void *_pc, uint16_t len){
40    uint16_t i;
41
42    uint8_t
43       *pc = (uint8_t *)_pc;

44    for (i = 0; i < len; i++, pc++){
45       crc= mmc_crc7_update(crc, *pc);
46    }
47
48    return crc;
49 }
50
51 inline uint16_t _crc_xmodem_update(uint16_t crc, uint8_t data){
52    uint8_t i;
53    crc = crc ^ ((uint16_t)data << 8);
54    for (i=0; i<8; i++){
55        if (crc & 0x8000)
56            crc = (crc << 1) ^ 0x1021;
57        else
58            crc <<= 1;
59    }

60    return crc;
61 }
62 #endif
63
64 #if (MMC_CRC_CHECK)
65 inline uint8_t spi_crc_read(spi_dev_t *_device, uint16_t *_crc){
66    uint8_t
67       b = spi_read_byte(_device);
68
69    *_crc = _crc_xmodem_update(*_crc, b);
70    return b;
71 }
72
73 inline void spi_crc_write(spi_dev_t *_device, uint16_t *_crc, uint8_t _data){
74    *_crc = _crc_xmodem_update(*_crc, _data);
```

```
1     spi_write_byte(_device, _data);
2  }
3
4  #endif
5
6  #define KBPS 1
7  #define MBPS 1000
8
9  const static uint32_t ts_exp[] = { 100*KBPS, 1*MBPS, 10*MBPS, 100*MBPS, 0, 0, 0, 0 };
10 const static uint32_t ts_mul[] = { 0, 1000, 1200, 1300, 1500, 2000, 2500, 3000,
11          3500, 4000, 4500, 5000, 5500, 6000, 7000, 8000 };
12
13 //0=1ns, 1=10ns, 2=100ns, 3=1us, 4=10us, 5=100us, 6=1ms, 7=10ms
14 uint32_t mmc_spi_taac(uint8_t _ts){
15    uint32_t
16       val=1;
17    uint8_t n, exp = (_ts&0x07);
18    for (n=0; n < exp; n++)
19       val *= 10;
20    val *= ts_mul[(_ts)>>3];
21
22    return val;
23 }
24
25 uint32_t mmc_spi_tran_speed(uint8_t _ts){
26    uint32_t rate = ts_exp[(_ts & 0x7)] * ts_mul[_ts >> 3];
27
28    if ( rate <= 0 ) {
29       MMC_DEBUG(DEBUG_LEVEL_INIT, ": error - unrecognized speed 0x%02x", _ts);
30       return 1;
31    }
32
33    return rate;
34 }
35 /**
36  * Set card in idle state
37  */
38 mmc_res_t mmc_spi_go_idle(mmc_spi_dev_t *_device){
39    int
40       retry =MMC_TIMING_GO_IDLE;
41    mmc_res_t
42       res;
43    mmc_r1_t
44       r1;
45    do {
46       // resetting card, go to SPI mode
47       if ((res = mmc_spi_send_r1_cmd(_device, MMC_GO_IDLE_STATE, 0, &r1)) != MMC_SUCCESS){
48          MMC_DEBUG(DEBUG_LEVEL_INIT, "Could not send CMD0");
49          return res;
50       }
51       if(!(retry--)){
52          MMC_DEBUG(DEBUG_LEVEL_INIT, "Busy while go idle");
53          return MMC_BUSY;
54       }
55       if (r1 == MMC_R1_IN_IDLE_STATE)
56          break;
57       // wait 2 milli seconds
58       usleep(MMC_TIMING_WAIT_GO_IDLE);
59    } while(true);
60    return MMC_SUCCESS;
61 }
62 /**
63  * Set card in operation condition state
64  */
65 mmc_res_t mmc_spi_wait_for_op_condition(mmc_spi_dev_t *_device){
66    int
67       retry = MMC_TIMING_SEND_OP;
68    mmc_res_t
69       res;
70    mmc_r1_t
71       r1;
72
```

```
1     do{
2        // initializing card for operation
3        if ((res = mmc_spi_send_r1_cmd(_device, MMC_SEND_OP_COND, 0, &r1)) == MMC_SUCCESS){
4           // is still in idle state?
5           if (!(r1&MMC_R1_IN_IDLE_STATE))
6              break;
7        }

8        // do retry counter
9        if(!(retry--)){
10          MMC_DEBUG(DEBUG_LEVEL_INIT, "Busy while send op condition");
11          return MMC_R1_BUSY;
12       }
13       // wait some milli seconds
14       usleep(MMC_TIMING_WAIT_SEND_OP);
15    } while(true);
16
17 // MMC_DEBUG(DEBUG_LEVEL_INIT, "result:%d", r1);
18
19    return MMC_SUCCESS;
20 }
21 /**
22  * Reset the card. Switch card to SPI Mode.
23  * @return r1 result code, MMC_R1_NO_ERROR on success
24  */
25 mmc_res_t mmc_spi_reset(mmc_spi_dev_t *_device){
26    mmc_res_t
27       res;
28    mmc_r3_t
29       r3;

30    // CS high = inactive
31   spi_cs_disable(_device->spiDevice);   // cs disable
32    // send dummy bytes with CS high before accessing
33    // issues at least 76 dummy clocks on spi bus for MMC initialization
34    spi_write_constant(_device->spiDevice, MMC_DUMMY, 10);

35    // set card in idle state
36    if ((res = mmc_spi_go_idle(_device)) != MMC_SUCCESS){
37       MMC_DEBUG(DEBUG_LEVEL_INIT, "Go idle was not successful");
38       return res;
39    }
40
41    if ((res = mmc_spi_wait_for_op_condition(_device)) != MMC_SUCCESS){
42       MMC_DEBUG(DEBUG_LEVEL_INIT, "No op condition came up");
43       return res;
44    }

45    // TODO: check card parameters for voltage compliance
46    // before issuing initialize command
47    if (( res = mmc_spi_read_ocr(_device, &r3)) < 0){
48       MMC_DEBUG(DEBUG_LEVEL_INIT, "Could not read OCR");
49       return res;
50    }
51
52    // check voltage constraints
53    if (!(r3&MMC_VDD_33_34)){
54       MMC_DEBUG(DEBUG_LEVEL_CMD, "Voltage out of range");
55       return MMC_VOLTAGE_ERROR;
56    }
57
58   // bit 31 is the busy bit if it is set the card is powered up
59 /*  if (!(r3&MMC_CARD_BUSY)){
60       MMC_DEBUG(DEBUG_LEVEL_CMD, "card did not power up");
61       return 1;
62    }*/
63
64    MMC_DEBUG(DEBUG_LEVEL_INIT, "Result: %d", res);
65
66    return res;
67 }
68 /**
69  * Init MMC in SPI mode
70  * @param _device the mmc volume
71  * @return r1 result code, MMC_R1_NO_ERROR on success
72  */
73 mmc_res_t mmc_spi_init(mmc_spi_dev_t *_device){
74    mmc_csd_t
```

```
1       csd;
2    mmc_res_t
3       res;
4    mmc_r2_t
5       r2;
6    mmc_r1_t
7       r1;
8
9    // spi clock freq
10   spi_set_freq(_device->spiDevice, MMC_LOWEST_FREQUENCY_HZ);
11
12   // reset the card
13   if ((res = mmc_spi_reset(_device)) != MMC_SUCCESS){
14      MMC_DEBUG(DEBUG_LEVEL_INIT, "Reset was not successful");
15      return res;
16   }
17
18   #if (MMC_CRC_CHECK)
19   // turn on CRC checking
20   if ((res = mmc_spi_send_r1_cmd(_device, MMC_CRC_ON_OFF, 1, &r1)) != MMC_SUCCESS)
21      return res;
22   if (r1 != MMC_R1_NO_ERROR)
23      return mmc_spi_map_error(r1);

24   #else
25   // turn off CRC checking to simplify communication
26   if ((res = mmc_spi_send_r1_cmd(_device, MMC_CRC_ON_OFF, 0, &r1)) != MMC_SUCCESS){
27      return res;
28   }
29   if (r1 != MMC_R1_NO_ERROR){
30      return mmc_spi_map_error(r1);
31   }
32   #endif
33

34   if ((res = mmc_spi_read_csd(_device, &csd)) != MMC_SUCCESS){
35      MMC_DEBUG(DEBUG_LEVEL_CSD, "Could not read csd");
36      return res;
37   }
38
39   // SPI Bus auf max Geschwindigkeit
40   uint32_t
41      f =  spi_set_freq(_device->spiDevice, csd.tran_speed);
42   uint32_t
43      NAC = csd.taac/100;
44   NAC *= f/100000;
45   NAC /=100000;
46   //min(((_device->NAC + csd.nsac)>>3), ((f/10)>>3));
47   _device->readTimeout               = ((NAC + csd.nsac)>>3)*100;
48   _device->writeTimeout              = _device->readTimeout*csd.r2w_factor;

49   MMC_DEBUG(DEBUG_LEVEL_CSD, "NAC: %d [8 cycles] (transmiited Bytes) => %f [s]",
50      NAC, NAC*8.0f/f);
51
52   _device->totalSectors      = ((uint32_t)csd.c_size+1)<<(csd.c_size_mult+2);
53   _device->sectorSize        = 1<<csd.read_bl_len;
54
55   _device->byteSize          = _device->sectorSize;
56   _device->byteSize          *= _device->totalSectors;
57
58   _device->blockPartial      = csd.read_bl_partial;
59
60   // set the block length to default (512)
61   if ((res = mmc_spi_set_block_length(_device, MMC_BLOCK_SIZE_LD)) != MMC_SUCCESS)
62      return res;

63   // read card status
64   if ((res = mmc_spi_read_status(_device, &r2)) < 0)
65      return res;

66   return MMC_SUCCESS;
67 }
68 /**
69  * send a cmd to the sd card
70  */
71 static void mmc_spi_send_cmd(mmc_spi_dev_t *_device, uint8_t _cmd, uint32_t _param){
72
73   MMC_DEBUG(DEBUG_LEVEL_CMD, "cmd:%d", (uint16_t)_cmd);
```

```
1
2     uint8_t cmdToken[8];
3
4     // transmission bit = 1
5     _cmd|=0x40;
6     cmdToken[0] = _cmd;
7     cmdToken[1] = _param>>24;
8     cmdToken[2] = _param>>16;
9     cmdToken[3] = _param>>8;
10    cmdToken[4] = _param;
11
12    #if (MMC_CRC_CHECK)
13    cmdToken[5] = mmc_crc7(0, cmdToken, 5)<<1)+1;
14    //(mmc_crc7(mmc_crc7(0, &_cmd, 1), &_param, 4)<<1)+1;
15    MMC_DEBUG(DEBUG_LEVEL_CMD, "crc:%d", (uint16_t)cmdToken[5]);
16    #else
17   // crc valid only for MMC_GO_IDLE_STATE
18    cmdToken[5] = 0x95;
19    #endif
20
21    // send command
22    spi_write(_device->spiDevice, cmdToken, 6);
23 }
24 /**
25  * Sends a mmc command without chip select
26  * @param _cmd the command to send to the card
27  * @param _param the parameter
28  * @return r1 result code, MMC_R1_NO_ERROR on success
29  */
30 static mmc_res_t mmc_spi_r1_cmd(mmc_spi_dev_t *_device,
31    uint8_t _cmd, uint32_t _param, mmc_r1_t *_r1){
32    mmc_res_t
33       res;
34    MMC_DEBUG(DEBUG_LEVEL_CMD, "send r1 cmd:%d", _cmd);
35    // send the command
36    mmc_spi_send_cmd(_device, _cmd, _param);

37    // wait for response
38    if ((res = spi_wait_while_token(_device->spiDevice, MMC_DUMMY, _device->cmdTimeout, _r1)) < 0)
39       return res;
40
41    MMC_DEBUG(DEBUG_LEVEL_CMD, "r1 response:%d", (uint16_t)*_r1);
42    // return response
43    return res;
44 }
45
46 static inline mmc_res_t mmc_spi_r1b_cmd(mmc_spi_dev_t *_device,
47    uint8_t _cmd, uint32_t _param, mmc_r1_t *_r1){
48    mmc_res_t
49       res;

50    mmc_spi_send_cmd(_device, _cmd, _param);

51    //The received byte immediataly following CMD12 is a stuff byte,
52    //it should be discarded before receive a response of the CMD12.
53    spi_read_byte(_device->spiDevice);

54    // wait for response
55    if ((res = spi_wait_while_token(_device->spiDevice, MMC_DUMMY, MMC_TIMING_MAX_CR, _r1)) < 0)
56       return res;

57    // wait, while busy
58    if ((res = spi_wait_while_token(_device->spiDevice, MMC_TOKEN_BUSY, _device->readTimeout,
59        NULL)) < 0)
60       return res;
61
62    MMC_DEBUG(DEBUG_LEVEL_CMD, "r1b response: %d", (uint16_t)*_r1);
63    // return response
64    return res;
65 }
66
67 /**
68  * Send Command to MMC/SD via SPI with chip select
69  * @param _device the volume to send the command to
70  * @param _cmd the command to send to the card
71  * @param _param the parameter
72  * @return r1 result code.
73  */
```

```
1  mmc_res_t mmc_spi_send_r1_cmd(mmc_spi_dev_t *_device, uint8_t _cmd,
2  uint32_t _param, mmc_r1_t *_r1){
3     mmc_res_t
4        res;

5     // assert chip select
6    spi_cs_enable(_device->spiDevice);

7     // issue the command
8     res = mmc_spi_r1_cmd(_device, _cmd, _param, _r1);
9
10    // release chip select
11    mmc_spi_end_transaction(_device);

12    return res;
13 }
14 /**
15  * Read data from MMC
16  * @param _device the mmc volume to read from
17  * @param _cmd to issue
18  * @param _param the params
19  * @param _buffer to receive data
20  * @param _length data length
21  * @return r1 result code.
22  */
23 static mmc_res_t mmc_cmd_read_data(mmc_spi_dev_t *_device, uint8_t _cmd,
24 uint32_t _param, uint8_t *_buffer, uint16_t _length){
25    #if (MMC_CRC_CHECK)
26    uint16_t
27       crc=0;
28    #else
29    uint8_t
30       temp[4];
31    #endif
32    mmc_res_t
33       res;
34    mmc_r1_t
35       r1;

36    // CS low
37    spi_cs_enable(_device->spiDevice); // cs enable
38
39    // issue command
40    if ((res = mmc_spi_r1_cmd(_device, _cmd, _param, &r1)) != MMC_SUCCESS){
41       ERROR("r1 cmd not successful");
42
43       mmc_spi_end_transaction(_device);
44       return res;
45    }
46    // check r1 result code
47    if (r1 != MMC_R1_NO_ERROR)
48       return mmc_spi_map_error(r1);
49
50    // wait until MMC_TOKEN_START_BLOCK_READ
51    if ((res = spi_wait_for_token(_device->spiDevice, MMC_TOKEN_START_BLOCK_READ,
52       _device->readTimeout)) < 0){
53       ERROR("wait for token failed");
54
55       mmc_spi_end_transaction(_device);
56       return res;
57    }
58
59    //Lesen des Blocks von MMC/SD-Karte
60    #if (MMC_CRC_CHECK)
61    for (;_length >0; _length--){
62       *_buffer++ = spi_crc_read(&crc);
63    }
64    #else
65       spi_read(_device->spiDevice, _buffer, _length);
66    #endif
67
68    // read crc
69    #if (MMC_CRC_CHECK)
70       spi_crc_read(&crc);
71       spi_crc_read(&crc);
72       MMC_DEBUG(DEBUG_LEVEL_READ, "crc:%d", crc);
73    #else
74       spi_read(_device->spiDevice, temp, 2);
```

```
1     #endif
2
3     // CS high
4     mmc_spi_end_transaction(_device);
5
6     #if (MMC_CRC_CHECK)
7     if (crc != 0)
8        return MMC_CRC_ERROR;
9     #endif
10
11    return MMC_SUCCESS;
12 }
13 /**
14  * Write data to MMC
15  * @param _device the mmc volume to write data to
16  * @param _cmd the command to issue
17  * @param _param the parameters to send
18  * @param _buffer the buffer of the data to be written
19  * @param _length the length of the data to send
20  * @return result code. 0 on no error < 0 on error
21  */
22 static mmc_res_t mmc_cmd_write_data(mmc_spi_dev_t *_device,
23    uint8_t _cmd, uint32_t _param, const uint8_t *_buffer, uint16_t _length){
24    mmc_res_t
25       res;
26    mmc_r1_t
27       r1;

28    #if (MMC_CRC_CHECK)
29    uint16_t
30       crc=0;
31    #endif
32    spi_cs_enable(_device->spiDevice); // cs enable
33
34    // issue command
35    if ((res = mmc_spi_r1_cmd(_device, _cmd, _param, &r1)) != MMC_SUCCESS){
36       ERROR("r1 command failed");
37       // CS high
38       mmc_spi_end_transaction(_device);
39       return res;
40    }

41    if (r1 != MMC_R1_NO_ERROR)
42       return mmc_spi_map_error(r1);
43
44    // issue 8-clock cycles wait and start single write block
45    MMC_DEBUG(DEBUG_LEVEL_WRITE, "send block write start token");
46    spi_write(_device->spiDevice, &start_single_write_tokens[0], 2);
47
48    #if (MMC_CRC_CHECK)
49    // write block
50    for (;_length >0; _length--){
51          spi_crc_write(&crc, *_buffer++);
52    }
53    #else
54       spi_write(_device->spiDevice, _buffer, _length);
55    #endif

56    #if (MMC_CRC_CHECK)
57       spi_write_byte(_device, crc>>8);
58       spi_write_byte(_device, crc);
59    #else
60       // write crc (dummy)
61       spi_write_constant(_device->spiDevice, MMC_DUMMY, 2);
62    #endif
63    // read data response
64    r1 = spi_read_byte(_device->spiDevice);
65    MMC_DEBUG(DEBUG_LEVEL_WRITE, "data write response before mask:%d", (uint16_t)r1);

66    r1 &=MMC_TOKEN_DATA_RESPONSE_STATUS_MASK;

67    MMC_DEBUG(DEBUG_LEVEL_WRITE, "data write response:%d", (uint16_t)r1);

68    // wait while busy
69    if (spi_read_byte(_device->spiDevice)!=MMC_TOKEN_BUSY){
70       ERROR("card not busy");
71    }
72    if ((res = spi_wait_while_token(_device->spiDevice,
```

```
      MMC_TOKEN_BUSY, _device->writeTimeout, NULL)) < 0){
      ERROR("wait while token BUSY failed");
      mmc_spi_end_transaction(_device);
      return res;
   }

   mmc_spi_end_transaction(_device);
   switch(r1){
      case MMC_TOKEN_DATA_RESPONSE_ACCEPTED:
         return MMC_SUCCESS;
      case MMC_TOKEN_DATA_RESPONSE_CRC_ERROR:
         return MMC_CRC_ERROR;
      case MMC_TOKEN_DATA_RESPONSE_WRITE_ERROR:
         return MMC_WRITE_ERROR;
      default:
         return MMC_SUCCESS;
   }
}
/**
 * Write a block to the card
 * @param _device the volume to write the block to
 * @param _blocknum the block number
 * @param _buffer the buffer containing the block data
 * @return r1 result, MMC_R1_NO_ERROR on success
 */
mmc_res_t mmc_spi_write_block(mmc_spi_dev_t *_device,
   uint32_t _blocknum, uint16_t _num, const void *_buffer){
   mmc_res_t
      res;
   #if (MMC_CRC_CHECK)
   uint16_t
      crc=0;
   #endif
   mmc_r1_t
      r1;

   if (_num == 1)
      return mmc_cmd_write_data(_device,
         MMC_WRITE_SINGLE_BLOCK, _blocknum<<_device->blockSizeLd,
         (uint8_t *)_buffer, _device->blockSize);
   else{
      spi_cs_enable(_device->spiDevice); // cs enable

      // issue command
      if ((res = mmc_spi_r1_cmd(_device,
         MMC_WRITE_MULTIPLE_BLOCK, _blocknum<<_device->blockSizeLd, &r1)) != MMC_SUCCESS){
         mmc_spi_end_transaction(_device);
         return res;
      }
      // check r1 result code
      if (r1 != MMC_R1_NO_ERROR)
         return mmc_spi_map_error(r1);

      for (; _num > 0; _num--){

         // issue 8-clock cycles wait (Nwr) and write start token
         MMC_DEBUG(DEBUG_LEVEL_WRITE, "send block write start token");
         spi_write(_device->spiDevice, &start_multi_write_tokens[0], 2);

         #if (MMC_CRC_CHECK)
            // write block
            for (;_length >0; _length--){
               spi_crc_write(&crc, *_buffer++);
            }
         #else
            spi_write(_device->spiDevice, _buffer, _device->blockSize);
         #endif

         #if (MMC_CRC_CHECK)
            spi_write_byte(_device, crc>>8);
            spi_write_byte(_device, crc);
         #else
            // write crc (dummy)
            spi_write_constant(_device->spiDevice, MMC_DUMMY, 2);
         #endif

         // Data response
         r1 = spi_read_byte(_device->spiDevice)&MMC_TOKEN_DATA_RESPONSE_STATUS_MASK;
```

```
1           MMC_DEBUG(DEBUG_LEVEL_WRITE, "data write response:%d", (uint16_t)r1);

2           // wait, while busy
3           if ((res = spi_wait_while_token(_device->spiDevice,
4              MMC_TOKEN_BUSY, _device->writeTimeout, NULL)) < 0){
5              ERROR("wait while busy failed");
6              mmc_spi_end_transaction(_device);
7              return res;
8           }

9           if (r1 != MMC_TOKEN_DATA_RESPONSE_ACCEPTED){
10             break;
11          }

12          _buffer+= _device->blockSize;
13       }

14       // issue 8-clock cycles wait (NWR)
15       spi_write_byte(_device->spiDevice, MMC_DUMMY);

16       // sent write stop token
17       MMC_DEBUG(DEBUG_LEVEL_WRITE, "send block stop token");
18       spi_write_byte(_device->spiDevice, MMC_TOKEN_STOP_MULTIPLE_BLOCK_WRITE);
19
20       // issue 8-clock cycles wait (NBR)
21       spi_write_byte(_device->spiDevice, MMC_DUMMY);

22       // wait, while busy
23       if (spi_read_byte(_device->spiDevice)!=MMC_TOKEN_BUSY){
24          ERROR("Card not busy");
25       }

26       if ((res = spi_wait_while_token(_device->spiDevice,
27          MMC_TOKEN_BUSY, _device->writeTimeout, NULL)) < 0){
28          ERROR("Wait while busy failed");
29          mmc_spi_end_transaction(_device);
30          return res;
31       }

32       mmc_spi_end_transaction(_device);

33       switch(r1){
34          case MMC_TOKEN_DATA_RESPONSE_ACCEPTED:
35             return MMC_SUCCESS;
36          case MMC_TOKEN_DATA_RESPONSE_CRC_ERROR:
37             return MMC_CRC_ERROR;
38          case MMC_TOKEN_DATA_RESPONSE_WRITE_ERROR:
39             return MMC_WRITE_ERROR;
40          default:
41             return MMC_SUCCESS;
42       }
43    }
44 }
45 /**
46  * Read a block from the card
47  * @param _device the volume to read the block from
48  * @param _blocknum the block number
49  * @param _buffer the buffer containing the block data
50  * @return the r1 result, MMC_R1_NO_ERROR on success
51  */
52 //#include <io.h>
53 //#include "system.h"
54 mmc_res_t mmc_spi_read_block(mmc_spi_dev_t *_device,
55    uint32_t _startBlock, uint16_t _num, void *_buffer){
56    mmc_res_t
57       res;
58    mmc_r1_t
59       r1;
60
61    //log(1, "block num:%d", _blocknum);
62    if (_num == 1)
63       return mmc_cmd_read_data(_device,
64       MMC_READ_SINGLE_BLOCK, _startBlock<<_device->blockSizeLd,
65       (uint8_t *)_buffer, _device->blockSize);
66    else{
67       // multiblock read
68
```

```
1        #if (MMC_CRC_CHECK)
2        uint16_t
3           crc=0;
4        #else
5        uint8_t
6           temp[4];
7        #endif
8        // CS low
9        spi_cs_enable(_device->spiDevice); // cs enable

10       // issue command
11       if ((res = mmc_spi_r1_cmd(_device,
12          MMC_READ_MULTIPLE_BLOCK, _startBlock<<_device->blockSizeLd, &r1)) < 0){
13          // CS high
14          mmc_spi_end_transaction(_device);
15          return res;
16       }
17       if (r1 != MMC_R1_NO_ERROR)
18          return mmc_spi_map_error(r1);

19       // loop for _num blocks
20       for (; _num >0 ; _num--){
21          // wait until MMC_TOKEN_START_BLOCK_READ
22          if ((res = spi_wait_for_token(_device->spiDevice,
23             MMC_TOKEN_START_BLOCK_READ, _device->readTimeout)) < 0){
24             ERROR("wait for data read token failed");
25             // CS high
26             mmc_spi_end_transaction(_device);
27             return res;
28          }

29          // read block
30          #if (MMC_CRC_CHECK)
31          for (;_length >0; _length--){
32                *_buffer++ = spi_crc_read(&crc);
33          }
34          #else
35             spi_read(_device->spiDevice, _buffer, _device->blockSize);
36          #endif

37          // read crc
38          #if (MMC_CRC_CHECK)
39             spi_crc_read(&crc);
40             spi_crc_read(&crc);
41             MMC_DEBUG(DEBUG_LEVEL_READ, "crc:%d", crc);
42          #else
43             spi_read(_device->spiDevice, temp, 2);
44          #endif

45          _buffer+= _device->blockSize;
46       }

47       // issue command
48       if ((res = mmc_spi_r1b_cmd(_device, MMC_STOP_TRANSMISSION, 0, &r1)) < 0){
49          // CS high
50          mmc_spi_end_transaction(_device);
51          return res;
52       }

53       if (r1 != MMC_R1_NO_ERROR)
54          return mmc_spi_map_error(r1);

55       // CS high
56       mmc_spi_end_transaction(_device);

57       return MMC_SUCCESS;
58    }
59 }
60
61 /**
62  * Read the CSD register
63  * @param _device to read the csd from
64  * @param _csd pointer to csd structure
65  * @return the r1 result, MMC_R1_NO_ERROR on success
66  */
67 mmc_res_t mmc_spi_read_csd(mmc_spi_dev_t *_device, mmc_csd_t *_csd){
68    uint8_t
69       buffer[MMC_CSD_SIZE];
```

```
    mmc_res_t
       res;

    if ((res = mmc_cmd_read_data(_device,
       MMC_READ_CSD, 0, buffer, MMC_CSD_SIZE)) != MMC_SUCCESS)
       return res;

    // version
    MMC_DEBUG(DEBUG_LEVEL_CSD, "version: %d", (uint16_t)buffer[0]);

    // TAAC
    _csd->taac = mmc_spi_taac(buffer[1]);

    MMC_DEBUG(DEBUG_LEVEL_CSD, "taac: %f [ns] max data setup time (%d,%d)",
       _csd->taac*1.0f/1000, buffer[1]&0x07, buffer[1]>>3);

    // NSAC
    _csd->nsac = ((uint16_t)buffer[2])*100;

    MMC_DEBUG(DEBUG_LEVEL_CSD, "nsac: %d [cycles]", _csd->nsac);

    // get the transpeed
    _csd->tran_speed = mmc_spi_tran_speed(buffer[3]);
    MMC_DEBUG(DEBUG_LEVEL_CSD, "tran speed: %d [Hz]", _csd->tran_speed);

    // get the C_SIZE value. bits [73:62] of data
    // [73:72] == data[6] && 0x03
    // [71:64] == data[7]
    // [63:62] == data[8] && 0xc0
    _csd->c_size = buffer[6] & 0x03;
    _csd->c_size <<= 8;
    _csd->c_size += buffer[7];
    _csd->c_size <<= 2;
    _csd->c_size += buffer[8] >> 6;

    MMC_DEBUG(DEBUG_LEVEL_CSD, "c_size: %d", (uint16_t)_csd->c_size);

    // get the val for C_SIZE_MULT. bits [49:47] of data
    // [49:48] == data[5] && 0x03
    // [47]    == data[4] && 0x80
    _csd->c_size_mult = buffer[9] & 0x03;
    _csd->c_size_mult <<= 1;
    _csd->c_size_mult += buffer[10] >> 7;

    MMC_DEBUG(DEBUG_LEVEL_CSD, "c_size_mult: %d", (uint16_t)_csd->c_size_mult);

    _csd->read_bl_len = buffer[5]&0xF;
    MMC_DEBUG(DEBUG_LEVEL_CSD, "read_bl_len: %d", (uint16_t)_csd->read_bl_len);

    _csd->read_bl_partial = buffer[6]>>7;
    MMC_DEBUG(DEBUG_LEVEL_CSD, "read_bl_partial: %d", (uint16_t)_csd->read_bl_partial);

    _csd->read_bl_misalign = (buffer[6]&0x20) != 0;
    MMC_DEBUG(DEBUG_LEVEL_CSD, "read_bl_misalign: %d", (uint16_t)_csd->read_bl_misalign);

    _csd->r2w_factor = 1<<((buffer[12] & 0x1c) >> 2);//1<<((buffer[12]>>2)&0x07);
    MMC_DEBUG(DEBUG_LEVEL_CSD, "r2w_factor: %d", _csd->r2w_factor);

    return res;
}
/**
 * Read the mmc/sd card status register
 * @return r2 response (SR) the upper 8-bits are the r1 result
 */
mmc_res_t mmc_spi_read_status(mmc_spi_dev_t *_device, mmc_r2_t *_r2){
    mmc_r2_t
       r2;
    mmc_res_t
       res;
    mmc_r1_t
       r1;

    // issue chip select
    spi_cs_enable(_device->spiDevice); // cs enable

    if ((res = mmc_spi_r1_cmd(_device, MMC_SEND_STATUS, 0, &r1)) != MMC_SUCCESS)
       return res;
```

```
    r2 = r1;
    r2 = r2 << 8;
    r2 += spi_read_byte(_device->spiDevice);

    // issue chip deselect
    mmc_spi_end_transaction(_device);

    *_r2 = r2;

    // return response
    return 0;
}
/**
 * Read the mmc/sd card ocr register
 * @return r3 response (OCR)
 */
mmc_res_t mmc_spi_read_ocr(mmc_spi_dev_t *_device, mmc_r3_t *_r3){
    uint8_t
       buffer[4];
    mmc_res_t
       res;
    mmc_r3_t
       r3;

    // issue chip select
    spi_cs_enable(_device->spiDevice); // cs enable

    mmc_r1_t
       r1;
    if((res = mmc_spi_r1_cmd(_device, MMC_READ_OCR, 0, &r1)) != MMC_SUCCESS){
       MMC_DEBUG(DEBUG_LEVEL_INIT, "OCR read command execution failed");
       // issue chip deselect
       mmc_spi_end_transaction(_device);
       return res;
    }

    if (r1 == MMC_R1_NO_ERROR){
       spi_read(_device->spiDevice, buffer, 4);
       r3 = buffer[0];
       r3 = r3 << 8;
       r3 += buffer[1];
       r3 = r3 << 8;
       r3 += buffer[2];
       r3 = r3 << 8;
       r3 += buffer[3];
       *_r3 = r3;
    }else{
       MMC_DEBUG(DEBUG_LEVEL_INIT, "OCR read failed with r1:%d" ,r1);
       return mmc_spi_map_error(r1);
    }
    // issue chip deselect
    mmc_spi_end_transaction(_device);

    // return response
    return res;
}
/**
 * Read the CID Register of the card
 * @param _device the volume to read the cid from
 * @param _cid the pointer to the cid structure
 * @return the result code
 */
mmc_res_t mmc_spi_read_cid(mmc_spi_dev_t *_device, mmc_cid_t *_cid){
    return mmc_cmd_read_data(_device, MMC_SEND_CID, 0, (uint8_t *)_cid, sizeof(mmc_cid_t));
}
/**
 * Set the block length default is 9, (2^9)=512
 * This is not in bytes it is sized in log2(bytes)
 * @param _device the volume
 * @param _length the block length in ld
 * @return the r1 result code
 */
mmc_res_t mmc_spi_set_block_length(mmc_spi_dev_t *_device, uint8_t _length){
    mmc_res_t
       res;
    mmc_r1_t
       r1;
```

```
1
2     int
3        length = 1<<_length;
4     if ((res = mmc_spi_send_r1_cmd(_device, MMC_SET_BLOCKLEN, length, &r1)) != MMC_SUCCESS)
5        return res;
6     if (r1 == MMC_R1_NO_ERROR){
7        _device->blockSize = length;
8        _device->blockSizeLd = _length;
9     }

10    return mmc_spi_map_error(r1);
11 }
```

Quelltext 33: MMC/SD-Karten Treiber-Implementierung mmc_spi.c

I.6 „open juke“-Referenzanwendung

I.6.1 Hauptprogramm

```
13 /*
14  * "open juke" - A NIOS2 based configurable SOPC Audiosystem
15  *
16  * The main program with initialization of device drivers and gui.
17  *
18  * Copyright 2008-2011 by Birger Zimmermann, all rights reserved.
19  *
20  * $Id: $
21  */
22 #include <stdio.h>
23 #include <unistd.h>
24 #include <fcntl.h>
25 #include <stdlib.h>
26 #include <io.h>
27 #include <string.h>
28 #include <time.h>
29 #include <sys/time.h>
30
31 #include "altera_avalon_sgdma.h"
32 #include "altera_avalon_sgdma_regs.h"
33 #include "altera_avalon_performance_counter.h"
34 #include "altera_avalon_pio_regs.h"
35 #include "sys/alt_alarm.h"
36 #include "sys/alt_irq.h"
37 #include "sys/alt_cache.h"
38 #include "sys/alt_errno.h"
39
40 // Debugging header file
41 #include "debug/debug.h"
42
43 // Driver header files
44 #include "device/mmc/mmc_spi_device.h"
45 #include "device/buffer/buffer_device.h"
46 #include "device/partition/partition_device.h"
47 #include "device/fat/fat_device.h"
48 #include "device/alt_tpo_lcd/alt_tpo_lcd.h"
49 #include "device/alt_video_display/alt_video_display.h"
50 #include "device/oc_i2c/oc_i2c_master.h"
51 #include "device/spi/spi.h"
52 #include "device/wm8731/wm8731_device.h"
53 #include "device/stream/stream.h"
54 #include "device/ad7843/ad7843.h"
55 #include "device/spi_master/spi_master.h"
56 #include "device/queue/queue.h"
57 #include "device/pio_button/pio_button.h"
58 #include "device/oc_ps2/oc_ps2_master.h"
59 #include "device/alt_spi/alt_spi_master.h"
60
61 // GFX
62 #include "gfx/gfx.h"
63 #include "gfx/fonts/courier10.h"
64
```

```
1  // Audio
2  #include "audio/mp3/mp3_decode.h"
3  #include "audio/wav/wav.h"
4
5  // Application specific includes
6  #include "oj_gui.h"
7  #include "util.h"
8  #include "list.h"
9
10 #ifndef NIL_MEM_BASE
11 #define NIL_MEM_BASE       0x2008000              // Address of NIL memory
12 #define NIL_MEM_SIZE       4096
13 #endif
14
15 //#define PERFORMANCE_TEST
16 //extern long long sampleCount;
17
18 // LCD config
19 #define LCD_WIDTH                          800              // LCD screen width
20 #define LCD_HEIGHT                         480              // LCD sreen height
21 #define LCD_COLORDEPTH                     16               // LCD screen color depth (bits per
22 pixel)
23 #define LCD_NUMBUFFER                      1                // Number of buffers for LCD driver
24 // Baseaddress of the display DMA descriptors
25 #define LCD_DESCRIPTOR_BASE              (DMA_DESCRIPTOR_MEMORY_BASE+8*32)
26
27 // AUDIO config
28 #define AUDIO_MICROPHONE_ENABLE          1                 // 1=Mic, 0=Line-In
29 #define AUDIO_TRANSFER_BLOCK_SIZE        4096
30 #define AUDIO_BUFFERS_PER_STREAM         3
31 #define AUDIO_DAC_DESCRIPTOR_BASE        (DMA_DESCRIPTOR_MEMORY_BASE)
32 #define AUDIO_ADC_DESCRIPTOR_BASE        (DMA_DESCRIPTOR_MEMORY_BASE+2*32)
33 #define AUDIO_SAMPLE_SPACE               (AUDIO_TRANSFER_BLOCK_SIZE*AUDIO_BUFFERS_PER_STREAM*2)
34
35 #define UI_UNSELECTED_ENTRY              -1
36
37 // jukebox states
38 typedef enum{STATE_UNMOUNTED, STATE_MP3_PLAYBACK, STATE_WAV_PLAYBACK, STATE_WAV_RECORDING,
39 STATE_STOP} state_t;
40
41 // internal function prototypes
42 void ui_button_listener(gui_button_t *_button, bool _state);
43 void ui_chooser_refresh();
44 void ui_set_pause(bool _pause);
45 void ui_set_next_state(state_t _nextState);
46 void ui_playlist_create_by_chooser();
47 int ui_playlist_select_entry(int _selected);
48 void ui_chooser_show_pos(int _pos);
49 void list_create_by_dir(const char *_dir, list_t *_list);
50 void unmount_card();
51 bool mount_card();
52
53 // ----------------------------------------------------------------------------
54 // Touchpanel init
55 // ----------------------------------------------------------------------------
56 static alt_spi_dev_t touchpanelSpi={
57    .base = (void *)TOUCH_PANEL_SPI_BASE,
58    .irqNum = TOUCH_PANEL_SPI_IRQ
59 };
60 // touch panel calibration data x
61 static ad7843_transform_t
62    xTransform={
63       .s0 = 132,
64       .s1 = 3946,
65       .t0 = 0,
66       .t1 = 799
67    };
68 // touch panel calibration data y
69 static ad7843_transform_t
70    yTransform={
71       .s0 = 148,
72       .s1 = 3849,
73       .t0 = 479,
74       .t1 = 0
75    };
76 static ad7843_dev_t touchpanel={
77    .spi           = &touchpanelSpi,
78    .penBase    = (void *)TOUCH_PANEL_IRQ_BASE,
```

```
    .xTransform = &xTransform,
    .yTransform = &yTransform
};
// ------------------------------------------------------------------------
// Filesystem init
// ------------------------------------------------------------------------
static spi_dev_t spi={
    .base       = (void *)SD_CONTROLLER_BASE,
    .clk        = SD_CONTROLLER_FREQ,
    .maxClock   = 15000000, // 15Mhz max bus freq
    .slaveNum   = 0
};
static mmc_spi_dev_t
  mmcDev={
    .device = {.name = "/mnt/mmc0"},
    .numOfBuffers = 5 // 5 buffers
  };
static fs_dev_t
    fs={
       .device ={.name = "/mnt/fs"},
       .parentDevice = "/mnt/mmc0"
};
static fat_volume_t
    fat={
       .device ={.name = "/mnt/root"},
       .parentDevice = "/mnt/fs/0",
       .numOfFiles = 10
    };
// ------------------------------------------------------------------------
// Audio init
// ------------------------------------------------------------------------
static i2c_device_t i2cAudioControl={
    .device = {.name = I2C_AUDIO_NAME},
    .base = (void *)I2C_AUDIO_BASE,
    .clk = I2C_AUDIO_FREQ,
    .sclk = 400000,
    .irqNum = I2C_AUDIO_IRQ
};
static uint32_t
       frequencies[] = {18432000, 16934400};
static wm8731_dev_t
    wm8731_control={
    I2C_AUDIO_NAME,
    frequencies,
    2
    };
static wm8731_audio_dev_t
    wm8731={
       .device={.name=WM8731_NAME},
    .base = (void *)WM8731_BASE,
    .control = &wm8731_control
    };
static int
    audioDeviceHandle;
// ------------------------------------------------------------------------
// Control init
// ------------------------------------------------------------------------
static queue_dev_t queue={
    .numberOfEntries  = 4,
    .totalQueueSize      = 16,
    .repeatCycles        = 1
};
static pio_button_dev_t buttons={
    .queue   = &queue,
    .queueStateIndex  = 0,
    .base    = BUTTON_PIO_BASE,
    .irqNum = BUTTON_PIO_IRQ,
    .bits    = BUTTON_PIO_DATA_WIDTH
};
static oc_ps2_dev_t
    ps2={
    .device        = {.name = OC_PS2_NAME},
    .base          = (void *)OC_PS2_BASE,
    .irqNum     = OC_PS2_IRQ,
    .queueSize  = 16
    };
// ------------------------------------------------------------------------
// GUI init
```

```
1  // ------------------------------------------------------------------------
2  static alt_video_display
3     display;
4  gfx_context_t
5     gfx;
6  // ------------------------------------------------------------------------
7  // UI
8  // ------------------------------------------------------------------------
9  static state_t
10    uiState=STATE_UNMOUNTED;
11
12 static bool
13    uiPause=false;
14
15 static char
16    currentFilename[256];
17 static char
18    currentDisplayname[256];
19
20 static char
21    uiChooserDirectory[256];
22 static char
23    secondsLeft[16];
24 static int
25    lastSecondsLeftNum;
26
27 static int
28    chooserPosition = -1;
29 static list_t
30    chooserList;
31
32 static list_t
33    uiPlaylist;
34 static int
35    uiPlaylistSelectedEntry=UI_UNSELECTED_ENTRY;
36 static char
37    uiPlaylistDirectory[256];
38 static bool
39    uiPlaylistDirty=true; // recreate playlist if new entry is selected
40 static mp3_play_t
41    mp3Play;
42 static wav_play_t
43    wavPlay;
44 static wav_record_t
45    wavRecord;
46 static int
47    recordingNum=0;
48 /**
49  * Init GUI (buttons)
50  */
51 void init_gui(){
52    // set the gfx context
53    gui.gfx = &gfx;

54    // set the first child in the gui
55    gui_set_first_child(&gui, &panel);

56    // chain the components
57    gui_add_comp(&panel, &btPrev);
58    gui_add_comp(&panel, &btPlay);
59    gui_add_comp(&panel, &btRec);
60    gui_add_comp(&panel, &btPause);
61    gui_add_comp(&panel, &btStop);
62    gui_add_comp(&panel, &btNext);
63
64    gui_add_comp(&panel, &labelTrack);
65    gui_add_comp(&panel, &labelTrackTime);
66    gui_add_comp(&panel, &btChooserDirectory);
67    gui_add_comp(&panel, &btEject);
68    gui_add_comp(&panel, &btUp);
69    gui_add_comp(&panel, &btDown);

70    int
71       i;
72    for (i=0; i< UI_CHOOSER_ENTRIES; i++)
73       gui_add_comp(&panel, &labelDirEntries[i]);

74    gui_add_comp(&panel, &labelCopyright);
```

```
   gui_add_repaint_comp(&panel);

   gfx_set_bg_color(&gfx, GFX_RGB16(155,155,155));
   gfx_set_color(&gfx, GFX_RGB16(255,255,255));
}
/**
 * Init the LCD Display
 */
bool init_lcd(){
   alt_tpo_lcd
      lcd={
   // Specify base addresses of the each communication bus PIO,
   // as defined in system.h
         .scen_pio = LCD_CNTRL_ENABLE_BASE,
         .scl_pio  = LCD_CNTRL_SCL_BASE,
         .sda_pio  = LCD_CNTRL_SDA_BASE
      };         // lcd configurator struct
   int result;

   // Write the default gamma curve, use 800x480 resolution (default),
   // and verify that the LCD is alive by reading back its Chip/ID
   // register. Most users will only need to call this routine, and
   // verify that it completed successfully.
   result = alt_tpo_lcd_init(&lcd, LCD_WIDTH, LCD_HEIGHT);
   if(result) {
      ERROR("Failed to initialize display");
      return false;
   }

   result = alt_video_display_init(&display,
      LCD_SGDMA_NAME,                 // char* sgdma_name
      LCD_WIDTH,                      // int width
      LCD_HEIGHT,                     // int height
      LCD_COLORDEPTH,                 // int color_depth
      ALT_VIDEO_DISPLAY_USE_HEAP,     // int buffer_location (malloc buffers)
      LCD_DESCRIPTOR_BASE, // int descriptor_location (malloc descriptors)
      LCD_NUMBUFFER);                 // int num_buffers

   if(result == 0) {
      gfx.display = &display;
      gfx.font    = &fontCourier10;
      gfx.color      = 0xFFFFFF;
      gfx.bgColor =  0x000000;
      gfx.opaque  = false;

      // set default clipping region
      gfx_set_clip_rect(&gfx, 0, 0, LCD_WIDTH, LCD_HEIGHT);
   }  else {
     ERROR("Failed to initialize display");
     return false;
   }

   return true;
}
/**
 * Init the user interface
 */
bool init_ui(){
   // init touchpanel
   alt_spi_init_device(&touchpanelSpi);
   ad7843_init_device(&touchpanel, 5);

   if (queue_init_device(&queue) <0)
      return false;

   if (pio_button_init_device(&buttons) <0)
      return false;

   if(oc_ps2_init_device(&ps2) <0)
      return false;

   return true;
}
/**
 * Init audio device
 */
bool init_audio(){
```

```
   // init audio control bus
   oc_i2c_init_device(&i2cAudioControl);

   // allocate sample buffer
   volatile void
      *sampleSpace = alt_uncached_malloc(AUDIO_SAMPLE_SPACE);

   // init the wm8731 connection
   if (wm8731_audio_init_device(&wm8731, AUDIO_SGDMA_TX_NAME, AUDIO_DAC_DESCRIPTOR_BASE,
   AUDIO_SGDMA_RX_NAME, AUDIO_ADC_DESCRIPTOR_BASE, sampleSpace, AUDIO_SAMPLE_SPACE,
   AUDIO_BUFFERS_PER_STREAM, NIL_MEM_BASE, NIL_MEM_SIZE) < 0){
      ERROR("Can not init wm8731 audio device");
      return false;
   }

   // open audio device
   audioDeviceHandle = open(WM8731_NAME, O_RDWR, 0);

   //wm8731_set_analog_path(&wm8731, WM8731_AANALOGUE_INSEL|WM8731_AANALOGUE_SIDEATT_N15 |
WM8731_AANALOGUE_SIDETONE | WM8731_AANALOGUE_DACSEL | WM8731_AANALOGUE_BYPASS);  // 0xF8
   #if AUDIO_MICROPHONE_ENABLE
   if (!wm8731_set_analog_path(&wm8731_control,
         WM8731_AANALOGUE_MICBOOST| WM8731_AANALOGUE_INSEL| WM8731_AANALOGUE_SIDEATT_N15 |
         WM8731_AANALOGUE_SIDETONE | WM8731_AANALOGUE_DACSEL| WM8731_AANALOGUE_BYPASS))
      return false;
   #else
   if (!wm8731_set_analog_path(&wm8731_control, WM8731_AANALOGUE_SIDEATT_N15 |
      WM8731_AANALOGUE_SIDETONE | WM8731_AANALOGUE_DACSEL| WM8731_AANALOGUE_BYPASS))
      return false;
   #endif

   // set volume
   if (!wm8731_set_linein_vol(&wm8731_control, WM8731_VOL_BOTH, 0x17)) // 0x17 0dB gain
      return false;

   return true;
}
/**
 * Init the SD-Card driver
 */
bool init_sd(){
   // init spi bus
   if (spi_master_init_device(&spi) < 0)
      return false;

  // create the mmc block device
  if (mmc_spi_init_device(&mmcDev, &spi) == 0){
      INFO("SD/MMC device created");
     return true;
  }

   return false;
}
/**
 * Exit the SD-Card driver
 */
void exit_sd(){
   mmc_spi_exit_device(&mmcDev);
}
/**
 * Set the position of the chooser
 */
void ui_chooser_set_pos(int _pos){
   chooserPosition = max(0, min(_pos, chooserList.listEntriesNum-UI_CHOOSER_ENTRIES));

   ui_chooser_refresh();
}
/**
 * Shows a given chooser position
 */
void ui_chooser_show_pos(int _pos){
   // ahead?
   if (_pos >= chooserPosition+UI_CHOOSER_ENTRIES)
      ui_chooser_set_pos(_pos-UI_CHOOSER_ENTRIES-1);
   else
   // above
   if (_pos < chooserPosition)
      ui_chooser_set_pos(_pos);
```

```
   else
      ui_chooser_refresh();
}
/**
 * Set the current directory
 */
void ui_chooser_set_directory(const char *_dir){
   // set the current dir
   strncpy(uiChooserDirectory, _dir, sizeof(uiChooserDirectory));

   // set the labels text
   gui_button_set_text(&btChooserDirectory, uiChooserDirectory);

   // enable/disable (parent dir selector)
   gui_set_disabled(&btChooserDirectory, str_equals(uiChooserDirectory, "/"));

   // create the dir list
   list_create_by_dir(uiChooserDirectory, &chooserList);

   // scroll up
   ui_chooser_set_pos(0);
}
/**
 * Mount the SD-Card, init partition device and fat filesystem
 */
bool mount_card(){
   // mount partitions
  if (partition_init_device(&fs) == 0){

   // mount filesystem
   if (fat_init_device(&fat) == 0){
         return true;
   }else
      partition_exit_device(&fs);
  }

   return false;
}
/**
 * Unmount the SD-Card
 */
void unmount_card(){
   // unmount fat
   fat_exit_device(&fat);
   // unmound fs
   partition_exit_device(&fs);
}
/**
 * Create a file list
 */
void list_create_by_dir(const char *_dir, list_t *_list){
   // clear the chooser list
   list_clear(&chooserList);

   // read directory of root filesystem
   fat_file_t
      *file;
   fat_result_t
      res;

   log(1, "reading directory: '%s'", _dir);
   if ((res=fat_open(&fat, _dir, -1, O_RDONLY, 0, &file)) >=0){
      fat_find_t
         find;
      entry_type_t
         type;
    while(fat_readdir(file, &find)==0){
      int
         len = strlen(find.filename);
      if (fat_is_direntry(&find)){
         if (stricmp(find.filename, ".") != 0 && stricmp(find.filename, "..") != 0)
            type = DIR_ENTRY;
         else
            continue;
      }
      else
      if (stricmp(find.filename+len-4, ".mp3") == 0)
         type = MP3_ENTRY;
```

```
      else
      if (stricmp(find.filename+len-4, ".wav") == 0)
         type = WAV_ENTRY;
      else
         continue;

      if (list_add_entry(&chooserList, find.filename, type) < 0)
         break;

      }

   fat_close(file);
   }else{
      ERROR("could not open dir");
   }

   list_sort(&chooserList);
}
/**
 * Stop playback or recording
 */
void ui_stop_playback_or_recording(){
      // stop the current state
   if (uiState == STATE_MP3_PLAYBACK){
      log(1, "stop mp3 playback");
      mp3_exit_playback(&mp3Play);
   }
   else
   if (uiState == STATE_WAV_PLAYBACK){
      log(1, "stop wav playback");
      wav_exit_playback(&wavPlay);
   }
   else
   if (uiState == STATE_WAV_RECORDING){
      log(1, "stop recording");
      if (wav_exit_recording(&wavRecord) == 0){
         list_add_entry(&chooserList, currentDisplayname, WAV_ENTRY);
         list_sort(&chooserList);

         ui_playlist_create_by_chooser();

         // set selected entry
         ui_playlist_select_entry(list_find_index_by_name(&uiPlaylist, currentDisplayname));


         int
            selected = list_find_index_by_name(&chooserList, currentDisplayname);
         // refresh chooser, position the selection on top
         ui_chooser_show_pos(selected);
      }
   }
}
/**
 * Reset the playlist (clear all entries)
 */
void ui_playlist_reset(){
   uiPlaylistSelectedEntry = UI_UNSELECTED_ENTRY;
   strcpy(uiPlaylistDirectory, "");
   list_clear(&uiPlaylist);
}
/**
 * Reset the chooser list
 */
void ui_chooser_reset(){
   gui_set_disabled(&btChooserDirectory, true);
   list_clear(&chooserList);
   strcpy(uiChooserDirectory, "");
   ui_chooser_set_pos(0);
}
/**
 * Reset the ui
 */
void ui_reset(){
   ui_playlist_reset();
   ui_chooser_reset();
   // un-select buttons
   ui_set_pause(false);
   // un-select play
```

```
1     gui_button_select(&btPlay, false);
2     // un-select rec
3     gui_button_select(&btRec, false);
4     // disable buttons
5     gui_set_disabled(&btRec, true);
6     gui_set_disabled(&btPlay, true);
7     gui_set_disabled(&btPause, true);
8     gui_set_disabled(&btNext, true);
9     gui_set_disabled(&btPrev, true);
10    gui_set_disabled(&btStop, true);

11    gui_button_set_text(&labelTrack, "");
12    gui_button_set_text(&labelTrackTime, "00:00");
13 }
14 /**
15  * Init playback
16  */
17 void ui_init_playback(){
18    // un-select pause
19    ui_set_pause(false);

20    // enable/disable prev button
21    gui_set_disabled(&btPrev, uiPlaylistSelectedEntry <= 0);

22    // enable/disable next button
23    gui_set_disabled(&btNext, uiPlaylistSelectedEntry >= uiPlaylist.listEntriesNum-1);
24 }
25 /**
26  * State transition function
27  */
28 void ui_set_next_state(state_t _nextState){
29    // stop playback or recording...(depends on current state)
30    ui_stop_playback_or_recording();

31    // start time
32    lastSecondsLeftNum = -1;
33
34    // is next state the "unmounted" state
35    if (_nextState == STATE_UNMOUNTED){
36       // unmount the file system etc.
37       unmount_card();

38       // reset the ui
39       ui_reset();

40       // set the new text on the button
41       gui_button_set_text(&btEject, UI_LABEL_MOUNT_TEXT);
42    }
43    else
44    // set the next state
45    if(_nextState == STATE_MP3_PLAYBACK){
46       ui_init_playback();
47       mp3_init_playback(&mp3Play, audioDeviceHandle, currentFilename);
48       gui_button_set_text(&labelTrack, currentDisplayname);
49
50       // select play
51       gui_button_select(&btPlay, true);
52       // disable rec
53       gui_set_disabled(&btRec, true);
54    }else
55    if(_nextState == STATE_WAV_PLAYBACK){
56       ui_init_playback();
57       wav_init_playback(&wavPlay, audioDeviceHandle, currentFilename);
58       gui_button_set_text(&labelTrack, currentDisplayname);

59       // select play
60       gui_button_select(&btPlay, true);

61       // disable rec
62       gui_set_disabled(&btRec, true);
63    }else
64    if(_nextState == STATE_WAV_RECORDING){

65       if (ui_create_record_name() < 0){
66          ERROR("could not generate name");
67          return;
68       }
69       wav_init_recording(&wavRecord, audioDeviceHandle,
```

```
1           currentFilename, 44100, 16, AUDIO_CHANNEL_BOTH);
2        gui_button_set_text(&labelTrack, currentDisplayname);
3        // select rec
4        gui_button_select(&btRec, true);
5        // disable play
6        gui_set_disabled(&btPlay, true);
7     }else
8     if(_nextState == STATE_STOP){

9        if (uiState == STATE_UNMOUNTED/* && _nextState != STATE_UNMOUNTED*/){
10          if (mount_card()){
11             // set the new text on the button
12             gui_button_set_text(&btEject, UI_LABEL_UNMOUNT_TEXT);

13             // enable stop button
14             gui_set_disabled(&btStop, false);

15             // read the directory
16          ui_chooser_set_directory("/");
17          uiPlaylistDirty = true;

18          // set the chooser to position 0
19          ui_chooser_set_pos(0);
20          }else
21             return;
22       }

23       // un-select play
24       gui_button_select(&btPlay, false);
25       // un-select rec
26       gui_button_select(&btRec, false);

27       // un-select pause
28       ui_set_pause(false);

29       // enable buttons
30       gui_set_disabled(&btRec, false);
31       gui_set_disabled(&btPlay, false);
32       gui_set_disabled(&btPause, false);
33    }
34    uiState = _nextState;
35 }
36 /**
37  * Searches the list and finds next playable entry
38  */
39 int ui_find_next_entry(){
40    int
41       selected = uiPlaylistSelectedEntry+1;

42    while (selected >= 0 && selected < uiPlaylist.listEntriesNum){
43       if (uiPlaylist.listEntries[selected].type != DIR_ENTRY){
44          return selected;
45       }
46       selected++;
47    }
48    return uiPlaylist.listEntriesNum;
49 }
50 /**
51  * Searches the list and finds previous playable entry
52  */
53 int ui_find_prev_entry(){
54    int
55       selected = uiPlaylistSelectedEntry-1;

56    while (selected >= 0 && selected < uiPlaylist.listEntriesNum){
57       if (uiPlaylist.listEntries[selected].type != DIR_ENTRY){
58          return selected;
59       }
60       selected--;
61    }
62    return UI_UNSELECTED_ENTRY;
63 }
64 /**
65  * Create a playlist by the chooser list
66  */
67 void ui_playlist_create_by_chooser(){
68    // clear playlist
69    list_clear(&uiPlaylist);
```

```
    int
       i;

    for (i=0; i < chooserList.listEntriesNum; i++){
       if (chooserList.listEntries[i].type != DIR_ENTRY){
          list_add_entry(&uiPlaylist, chooserList.listEntries[i].name,
          chooserList.listEntries[i].type);
       }
    }

    // set the playlists directory
    strcpy(uiPlaylistDirectory, uiChooserDirectory);

    uiPlaylistDirty = false;
}
/**
 * Refresh the chooser ui
 */
void ui_chooser_refresh(){
   bool
       selectedInCurrentDir = str_equals(uiPlaylistDirectory, uiChooserDirectory);

   int
       i, minEntries = min(chooserList.listEntriesNum, UI_CHOOSER_ENTRIES);

    for (i=0; i < UI_CHOOSER_ENTRIES; i++){
       gui_button_select(&labelDirEntries[i], uiPlaylistSelectedEntry != UI_UNSELECTED_ENTRY &&
       selectedInCurrentDir &&
       (i+chooserPosition >=0 && i+chooserPosition < chooserList.listEntriesNum) &&
       (str_equals(uiPlaylist.listEntries[uiPlaylistSelectedEntry].name,
       chooserList.listEntries[i+chooserPosition].name)));

       if (i < minEntries){
          gui_button_set_text(&labelDirEntries[i],
             chooserList.listEntries[chooserPosition+i].name);
          gui_set_disabled(&labelDirEntries[i], false);
          switch(chooserList.listEntries[chooserPosition+i].type){
             case DIR_ENTRY:{
                labelDirEntries[i].unselectedSchema = &csDirEntryUnselected;
                break;
             }
             default:{
                labelDirEntries[i].unselectedSchema = &csEntryUnselected;
                break;
             }
          }
       }
       else{
          gui_set_disabled(&labelDirEntries[i], true);
          gui_button_set_text(&labelDirEntries[i], "");
       }
    }

    // set the up/down buttons enabled/disabled
    gui_set_disabled(&btUp, chooserPosition <= 0);
    gui_set_disabled(&btDown, chooserPosition >= chooserList.listEntriesNum-UI_CHOOSER_ENTRIES);

}
/**
 * Set pause (indirect state)
 */
void ui_set_pause(bool _pause){
       gui_button_select(&btPause, _pause);
       uiPause = _pause;
}
/**
 * Create a name for the recording file
 */
int ui_create_record_name(){
   int
       errorCount=1000; // TODO Constant
   char
       formattedTime[32];
   time_t timeVal;
   if (time(&timeVal) != -1){
       struct tm *stm = localtime(&timeVal);
      sprintf(formattedTime, "_%02i.%02i.%04i_%02ih%02im%02is",
          stm->tm_mday, stm->tm_mon+1, stm->tm_year+1900, stm->tm_hour, stm->tm_min, stm->tm_sec);
```

```
1
2        recordingNum = 0;
3     }else
4        formattedTime[0] = 0;
5
6     do{
7        strcpy(currentFilename, "/mnt/root");
8        strcat(currentFilename, uiChooserDirectory);
9        if (!str_equals(uiChooserDirectory, "/"))
10          strcat(currentFilename, "/");
11       if (recordingNum == 0)
12          sprintf(currentDisplayname, "recording%s.wav", formattedTime);
13       else
14          sprintf(currentDisplayname, "recording%s_%d.wav", formattedTime, recordingNum);
15       recordingNum++;
16       strcat(currentFilename, currentDisplayname);
17       // try to open the file
18       int
19          file = open(currentFilename, O_RDONLY, 0);
20       if (file >= 0)
21          close(file);
22       else
23       if (ALT_ERRNO == ENOENT)
24          return 0;
25
26    }while(errorCount--);
27    return -EIO;
28 }
29 /**
30  * Create names for display and file access
31  */
32 void ui_create_selection_names(int _selected){
33    strcpy(currentFilename, "/mnt/root");
34    strcat(currentFilename, uiPlaylistDirectory);
35    if (!str_equals(uiPlaylistDirectory, "/"))
36       strcat(currentFilename, "/");
37    strcat(currentFilename, uiPlaylist.listEntries[_selected].name);
38    strcpy(currentDisplayname, uiPlaylist.listEntries[_selected].name);
39 }
40 /**
41  * Select an entry in the playlist
42  */
43 int ui_playlist_select_entry(int _selected){
44    if (_selected>=0 && _selected < uiPlaylist.listEntriesNum){
45          ui_create_selection_names(_selected);
46          uiPlaylistSelectedEntry = _selected;
47       return _selected;
48    }
49    return -1;
50 }
51 /**
52  * Start playback of an entry
53  */
54 int ui_play_entry(int _selected){
55    if (ui_playlist_select_entry(_selected) >= 0){
56 /*
57    int
58       selected = list_find_index_by_name(&chooserList, currentDisplayname);
59    // refresh chooser, position the selection on top
60    ui_chooser_show_pos(selected);
61 */
62       if (uiPlaylist.listEntries[_selected].type == MP3_ENTRY){
63          ui_set_next_state(STATE_MP3_PLAYBACK);
64          return 0;
65       }else
66       if (uiPlaylist.listEntries[_selected].type == WAV_ENTRY){
67          ui_set_next_state(STATE_WAV_PLAYBACK);
68          return 0;
69       }
70    }
71    ui_set_next_state(STATE_STOP);
72    return -1;
73 }
```

```
1  /**
2   * Callback function from the ui buttons
3   */
4  void ui_button_listener(gui_button_t *_button, bool _state){
5     int
6        id = _button->comp.id;
7     char
8        concatBuffer[256];
9     //log(1, "button: %d state:%d", id, _state);
10    if (id == UI_RECORD_ID){
11       ui_set_next_state(STATE_WAV_RECORDING);
12    }else
13
14    if (id == UI_EJECT_ID){
15       if (uiState != STATE_UNMOUNTED){
16          // unmount
17          ui_set_next_state(STATE_UNMOUNTED);
18       }else
19          ui_set_next_state(STATE_STOP);
20
21    }else
22    if (id == UI_BUTTON_STOP_ID){
23       ui_set_next_state(STATE_STOP);
24    }else
25    if (id == UI_BUTTON_PLAY_ID){
26       ui_play_entry(uiPlaylistSelectedEntry);
27       ui_chooser_refresh();
28    }else
29    if (id == UI_BUTTON_NEXT_ID){
30       ui_play_entry(ui_find_next_entry());
31       ui_chooser_refresh();
32    }else
33    if (id == UI_BUTTON_PREV_ID){
34       ui_play_entry(ui_find_prev_entry());
35       ui_chooser_refresh();
36    }else
37    if (id == UI_BUTTON_PAUSE_ID){
38       uiPause = _state;
39    }else
40    if (id == UI_DIRECTORY_ID){

41       strcpy(concatBuffer, uiChooserDirectory);
42       char *
43          ptr = strrchr(concatBuffer, '/');
44       // set parent directory
45       if (ptr >= concatBuffer){
46          if (ptr == concatBuffer)
47             concatBuffer[1] = 0;
48          else
49             *ptr=0;
50          ui_chooser_set_directory(concatBuffer);
51          uiPlaylistDirty = true;
52       }
53    }else
54    if (id ==UI_BUTTON_UP_ID){
55       ui_chooser_set_pos(chooserPosition-1);
56    }
57    else
58    if (id ==UI_BUTTON_DOWN_ID){
59       ui_chooser_set_pos(chooserPosition+1);
60    }
61    else
62    if (id >= UI_ENTRIES_START_ID){
63       int
64          selected = chooserPosition+id-UI_ENTRIES_START_ID;

65       if (chooserList.listEntries[selected].type == DIR_ENTRY){

66          strcpy(concatBuffer, uiChooserDirectory);
67          if (strlen(uiChooserDirectory) != 1)
68             strcat(concatBuffer, "/");
69          strcat(concatBuffer, chooserList.listEntries[selected].name);

70          ui_chooser_set_directory(concatBuffer);
71          uiPlaylistDirty = true;
72       }else{
73          // new dir ?
74          if (uiPlaylistDirty){
```

```
1              // create a new playlist
2              ui_playlist_create_by_chooser();
3           }
4           selected = list_find_index_by_name(&uiPlaylist, chooserList.listEntries[selected].name);
5           ui_play_entry(selected);
6           ui_chooser_refresh();
7        }
8     }
9  }
10 /**
11  * Init the internal timer (no RTC present in system, so fixed date is used)
12  */
13 void init_time(){
14       struct tm now={
15       .tm_sec  = 0,
16       .tm_min  = 0,
17       .tm_hour = 0,
18       .tm_mday = 29,
19       .tm_mon  = 6,
20       .tm_year = 108,
21       .tm_wday = 2,
22       .tm_yday = 0,
23       .tm_isdst= 0
24    };
25    struct timeval now_time_val;
26    time_t
27       time= mktime(&now);
28    now_time_val.tv_sec = time;
29    settimeofday(&now_time_val, NULL);
30 }
31 /**
32  * The MAIN routine, contains the control logic for all peripheral access
33  */
34 int main() {
35    #ifdef PERFORMANCE_TEST
36    PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
37    PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);
38    #endif

39    // init system clock
40    init_time();

41    // init the lcd display and gfx context
42    if (!init_lcd()){
43       return 1;
44    }
45    // init ps2, buttons, touchpanel
46    if (!init_ui()){
47       return 1;
48    }
49    // init audio hw/sw
50    if (!init_audio()){
51       return 1;
52    }
53    // init mp3 decoding
54    if (mp3_init_decode() < 0)
55       return 1;

56    // init sd device
57    if (!init_sd()){
58       return 1;
59    }
60    // init graphical user interface
61    init_gui();

62    // init the button states etc.
63    ui_reset();

64    // set the ui state (tries to mount the card) if unmounted
65    ui_set_next_state(STATE_STOP);
66
67    uint32_t
68       oldFrameNum=0; // frame counter

69    uint8_t
70       lineOutVol=WM8731_LINEOUT_DEFAULT_VOL; // default volume

71    // main loop
```

```
1     do{
2        ad7843_msg_t
3           msg;
4        queue_msg_t
5           state;

6        // is a button (on the bord) pressed?
7        if (queue_get_next(&queue, &state)){
8           /* && !state.repeat*/
9           if (state.state== QUEUE_STATE_PRESSED || state.state==QUEUE_STATE_REPEAT){
10             // decrease volume
11             if (state.code == 0){
12                if (lineOutVol >= WM8731_LINEOUT_MIN_VOL)
13                   wm8731_set_lineout_vol(wm8731.control, WM8731_VOL_BOTH, lineOutVol--);
14             }else
15             // increase volume
16             if (state.code == 1){
17                if (lineOutVol < WM8731_LINEOUT_MAX_VOL)
18                   wm8731_set_lineout_vol(wm8731.control, WM8731_VOL_BOTH, lineOutVol++);
19             }
20             #ifdef PERFORMANCE_TEST
21             else
22             if (state.code == 2)
23                break;
24             #endif
25          }
26          //return 0;
27       }
28       else
29       // key on ps2 keyboard pressed/release etc.?
30       if (oc_ps2_has_next(&ps2)){
31          gui_key_t
32             key;
33          uint32_t
34             data = oc_ps2_get_next(&ps2);

35          key.ascii = KEY_GET_ASCII(data);
36          key.code    = KEY_GET_CODE(data);
37          key.type = (data & KEY_STATE_PRESSED)?
38                      ((data & KEY_STATE_REPEAT)?GUI_KEY_REPEAT:GUI_KEY_DOWN):GUI_KEY_UP;

39          log(1, "key code: %d", key.code);

40          // delegate the key to the gui
41          gui_fire_key_event(&gui, &key);

42       }else
43       // a message from the touchpanel?
44       if (ad7843_get_next_msg(&touchpanel, &msg)){

45          // delegate the mouse
46          gui_fire_mouse_event(&gui, msg.x, msg.y, msg.penDown);
47       }
48       // paused ?
49       if (!uiPause){
50          switch(uiState){
51             // do the mp3 continues playback
52             case STATE_MP3_PLAYBACK:{

53                mp3_play_next(&mp3Play);

54                if (mp3Play.errorCode != 0){
55                   ui_play_entry(ui_find_next_entry());
56                   ui_chooser_refresh();
57                }else{
58                   // create track time display
59                   int
60                      secondsLeftNum = mp3_get_seconds_left(&mp3Play);
61                   if (secondsLeftNum >= 0 &&
62                      (secondsLeftNum < lastSecondsLeftNum || lastSecondsLeftNum == -1) ){
63                      sprintf(secondsLeft, "%.2d:%.2d", secondsLeftNum/60, secondsLeftNum%60);
64                      gui_button_set_text(&labelTrackTime, secondsLeft);
65                      lastSecondsLeftNum = secondsLeftNum;
66                   }
67                }
68                break;
69             }
70             // do the wav playback
```

```
            case STATE_WAV_PLAYBACK:{
               wav_play_next(&wavPlay);

               if (wavPlay.errorCode != 0){
                  ui_play_entry(ui_find_next_entry());
                  ui_chooser_refresh();
               }else{
                  // create track time display
                  int
                     secondsLeftNum = wav_get_seconds_left(&wavPlay);
                  if (secondsLeftNum >= 0 &&
                     (secondsLeftNum < lastSecondsLeftNum || lastSecondsLeftNum == -1) ){
                     sprintf(secondsLeft, "%.2d:%.2d", secondsLeftNum/60, secondsLeftNum%60);
                     gui_button_set_text(&labelTrackTime, secondsLeft);
                     lastSecondsLeftNum = secondsLeftNum;
                  }
               }
               break;
            }
            // record further bytes
            case STATE_WAV_RECORDING:{
               wav_record_next(&wavRecord, 2048);
               break;
            }
            default:
               break;
         }
      }

      // repaint the components
      uint32_t
         newFrameNum = gfx.display->frameNum;

      // a new frame has begun?
      if (newFrameNum != oldFrameNum){
         gui_repaint(&gui);
         oldFrameNum = newFrameNum;
      }

   }while(1);
   //--------------------------------------------------------
   // program never reaches here, but for completeness
   // show how to de-init all devices
   //--------------------------------------------------------
   // unmount the card
   unmount_card();
   // exit the sd device
   exit_sd();

   #ifdef PERFORMANCE_TEST
   PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
   perf_print_formatted_report(PERFORMANCE_COUNTER_0_BASE, SD_CONTROLLER_FREQ, 1,
         "mp3 decode bytes");
   printf("samplecount:%lu", sampleCount);
   #endif

   wm8731_audio_exit_device(&wm8731);

   // exit button device
  pio_button_exit_device(&buttons);
  // exit the button queue
  queue_exit_device(&queue);

  // de-allocate mp3 decode memory
  mp3_exit_decode();

  return 0;
}
```

Quelltext 34: "open juke"-Referenzanwendung Hauptprogramm main.c

I.6.2 GUI-Header-Datei

```
/*
 * "open juke" - A NIOS2 based configurable SOPC Audiosystem
 *
 * The gui structure
 *
 * Copyright 2008-2011 by Birger Zimmermann, all rights reserved.
 *
 * $Id: $
 */
#ifndef OJ_GUI_H_
#define OJ_GUI_H_

#include "gui/gui.h"
#include "gui/gui_button.h"
#include "gui/gui_panel.h"

// Color definitions
#define COL_MY_RED                          GFX_RGB16(200,100,100)
#define COL_LTRED                           GFX_RGB16(120,50,50)
#define COL_DKRED                           GFX_RGB16(120,50,50)
#define COL_LTGREY                          GFX_RGB16(200,200,200)

#define COL_GREY                            GFX_RGB16(160,160,160)
#define COL_MIDGREY                         GFX_RGB16(120,120,120)
#define COL_DKGREY                          GFX_RGB16(100,100,100)
#define COL_DKDKGREY                        GFX_RGB16(80,80,80)
#define COL_MY_BLUE                         GFX_RGB16(050,050,200)

#define COL_BUTTON_CENTER                   COL_LTGREY
#define COL_BUTTON_SELECTED                 COL_GREY
#define COL_CHOOSER_BUTTON_SELECTED         COL_MY_BLUE
#define  COL_BUTTON_TOP                     COL_WHITE
#define  COL_BUTTON_BOTTOM                  COL_DKGREY
#define  COL_BUTTON_TEXT                    COL_DKGREY

// UI position definitions
#define UI_POS_Y                            0
#define UI_POS_X                            0
#define UI_WIDTH                            800
#define UI_HEIGHT                           480

#define UI_BUTTON_MARGIN                    6
#define UI_BUTTON_MARGIN_Y                  12
#define UI_BUTTON_WIDTH                     ((UI_WIDTH-5*UI_BUTTON_MARGIN)/6)
#define UI_BUTTON_HEIGHT                    90
#define UI_LABEL_HEIGHT                     52
#define UI_TRACK_WIDTH                      (4*UI_BUTTON_MARGIN+5*UI_BUTTON_WIDTH)
#define UI_TRACK_HEIGHT                     (UI_LABEL_HEIGHT)
#define UI_BUTTON_2ROW_Y                    (UI_LABEL_HEIGHT+UI_BUTTON_MARGIN_Y)

#define UI_CHOOSER_TITLE_POS_Y         (UI_LABEL_HEIGHT+UI_BUTTON_MARGIN_Y*2+UI_BUTTON_HEIGHT)
#define UI_CHOOSER_TITLE_HEIGHT             52
#define UI_CHOOSER_TITLE_MARGIN             12

#define UI_CHOOSER_POS_Y (UI_CHOOSER_TITLE_POS_Y+UI_CHOOSER_TITLE_MARGIN+UI_CHOOSER_TITLE_HEIGHT)
#define UI_CHOOSER_MARGIN                   6

#define UI_CHOOSER_ROW_WIDTH              (UI_WIDTH-UI_CHOOSER_MARGIN-UI_BUTTON_WIDTH)
#define UI_CHOOSER_ROW_HEIGHT               52
#define UI_CHOOSER_ENTRIES                  4

// UI message ids
#define UI_RECORD_ID                        3
#define UI_EJECT_ID                         4
#define UI_BUTTON_STOP_ID                   5
#define UI_BUTTON_PLAY_ID                   6
#define UI_BUTTON_NEXT_ID                   7
#define UI_BUTTON_PREV_ID                   8
#define UI_BUTTON_PAUSE_ID                  9
#define UI_BUTTON_UP_ID                     10
#define UI_BUTTON_DOWN_ID                   11
#define UI_ENTRIES_START_ID                 20
#define UI_DIRECTORY_ID                     19

// UI texts
#define UI_LABEL_MOUNT_TEXT                 "INSERT"
```

```
1  #define UI_LABEL_UNMOUNT_TEXT               "EJECT"
2  #define UI_LABEL_COPYRIGHT                  "open juke, (c) 2008-2011 by Birger Zimmermann"
3  #define UI_LABEL_PAUSE                      "PAUSE"
4  #define UI_LABEL_STOP                       "STOP"
5  #define UI_LABEL_NEXT                       "NEXT"
6  #define UI_LABEL_UP                         "UP"
7  #define UI_LABEL_DOWN                       "DOWN"
8  #define UI_LABEL_RECORD                     "REC"
9  #define UI_LABEL_PLAY                       "PLAY"
10 #define UI_LABEL_PREV                       "PREV"
11
12 // Components
13 extern gui_t gui;
14 extern gui_panel_t panel;
15 extern gui_button_t btPrev;
16 extern gui_button_t btPlay;
17 extern gui_button_t btRec;
18 extern gui_button_t btPause;
19 extern gui_button_t btStop;
20 extern gui_button_t btNext;
21 extern gui_button_t labelTrack;
22 extern gui_button_t labelTrackTime;
23 extern gui_button_t btChooserDirectory;
24 extern gui_button_t btEject;
25 extern gui_button_t btUp;
26 extern gui_button_t btDown;
27 extern gui_button_t labelDirEntries[];
28 extern gui_button_t labelCopyright;
29
30 // Color schemas
31 extern gui_color_schema_t csDirEntryUnselected;
32 extern gui_color_schema_t csEntryUnselected;
33
34 // Callback
35 extern void ui_button_listener(gui_button_t *_button, bool _state);
36 #endif /*OJ_GUI_H_*/
```

Quelltext 35: "open juke"-Referenzanwendung GUI-Header-Datei oj_gui.h

J Hardware-Plattform Projekt-Dateien

J.1 Synopsys-Design-Constraints-File

```
37 ## Creating and setting variables for clock paths to make code look cleaner
38 set System_Clock_int *|the_pll|the_pll|altpll_component|auto_generated|pll1|clk[0]
39 set SSRAM_Clock_ext *|the_pll|the_pll|altpll_component|auto_generated|pll1|clk[1]
40 set Slow_Clock_int *|the_pll|the_pll|altpll_component|auto_generated|pll1|clk[2]
41
42 set DDR_Local_Clock *|the_ddr_sdram|ddr_sdram_controller_phy_inst|alt_mem_phy_inst|
43 ddr_sdram_phy_alt_mem_phy_ciii_inst|clk|pll|altpll_component|auto_generated|pll1|clk[1]
44 set DDR_Controller_Clock *|the_ddr_sdram|ddr_sdram_controller_phy_inst|alt_mem_phy_inst|
45 ddr_sdram_phy_alt_mem_phy_ciii_inst|clk|pll|altpll_component|auto_generated|pll1|clk[0]
46
47 set audio_18432_int the_audio_pll|altpll_component|auto_generated|pll1|clk[0]
48 set audio_16934_int the_audio_pll|altpll_component|auto_generated|pll1|clk[1]
49
50 ## cut paths between audio pll and system clk
51 set_false_path -from [get_clocks $DDR_Controller_Clock] -to [get_clocks $audio_18432_int]
52 set_false_path -from [get_clocks $DDR_Controller_Clock] -to [get_clocks $audio_16934_int]
53 set_false_path -from [get_clocks $audio_18432_int] -to [get_clocks $DDR_Controller_Clock]
54 set_false_path -from [get_clocks $audio_16934_int] -to [get_clocks $DDR_Controller_Clock]
55
56 set_false_path -from [get_clocks $audio_18432_int] -to [get_clocks $audio_16934_int]
57 set_false_path -from [get_clocks $audio_16934_int] -to [get_clocks $audio_18432_int]
58
59 ## Cutting the paths between the system clock and ddr controller clock
60 ## since there is a clock crossing bridge between them (FIFOs)
61 set_false_path -from [get_clocks {osc_clk}] -to [get_clocks $Slow_Clock_int]
62 set_false_path -from [get_clocks $Slow_Clock_int] -to [get_clocks {osc_clk}]
```

```
1
2  ## Cutting the paths between the system clock and ddr controller clock
3  ## since there is a clock crossing bridge between them (FIFOs)
4  set_false_path -from [get_clocks $Slow_Clock_int] -to [get_clocks $System_Clock_int]
5  set_false_path -from [get_clocks $System_Clock_int] -to [get_clocks $Slow_Clock_int]
6
7  ## Cutting the paths between the system clock and ddr controller clock
8  ## since there is a clock crossing bridge between them (FIFOs)
9  set_false_path -from [get_clocks $Slow_Clock_int] -to [get_clocks $DDR_Controller_Clock]
10 set_false_path -from [get_clocks $DDR_Controller_Clock] -to [get_clocks $Slow_Clock_int]
11
12 ## Cutting the paths between the system clock and ddr controller clock
13 ## since there is a clock crossing bridge between them (FIFOs)
14 set_false_path -from [get_clocks {osc_clk}] -to [get_clocks $DDR_Controller_Clock]
15 set_false_path -from [get_clocks $DDR_Controller_Clock] -to [get_clocks {osc_clk}]
16
17 ## Cutting the paths between the system clock and ddr controller clock
18 ## since there is a clock crossing bridge between them (FIFOs)
19 set_false_path -from [get_clocks $System_Clock_int] -to [get_clocks $DDR_Controller_Clock]
20 set_false_path -from [get_clocks $DDR_Controller_Clock] -to [get_clocks $System_Clock_int]
21
22 ## Cutting the paths between the system clock and ddr local clock
23 ##since there is a clock crossing bridge between them (FIFOs)
24 set_false_path -from [get_clocks $System_Clock_int] -to [get_clocks $DDR_Local_Clock]
25 set_false_path -from [get_clocks $DDR_Local_Clock] -to [get_clocks $System_Clock_int]
26
27 ## Cutting the paths between the external oscillator clock
28 ## and the system clock since there is an asyncronous clock crosser between them
29 set_false_path -from [get_clocks {osc_clk}] -to [get_clocks $System_Clock_int]
30 set_false_path -from [get_clocks $System_Clock_int] -to [get_clocks {osc_clk}]
31
32 ## Cutting the paths between the external oscillator clock
33 ## and the system clock since there is an asyncronous clock crosser between them
34 set_false_path -from [get_clocks {osc_clk}] -to [get_clocks $DDR_Local_Clock]
35 set_false_path -from [get_clocks $DDR_Local_Clock] -to [get_clocks {osc_clk}]
36
37 ##SSRAM Constraints
38 set_output_delay -clock [get_clocks $SSRAM_Clock_ext] -reference_pin [get_ports {ssram_clk}] 2.4
39 [get_ports {ssram_adsc_n ssram_bw_n* ssram_bwe_n ssram_ce_n ssram_oe_n flash_ssram_a*
40 flash_ssram_d*}]
41 set_input_delay -clock [get_clocks $SSRAM_Clock_ext]  -reference_pin [get_ports {ssram_clk}] 4.1
42 [get_ports {flash_ssram_d*}]
43 set_multicycle_path -from [get_ports {flash_ssram_d*} ] -setup -end 2
```

Quelltext 36: Synopsys-Design-Constraints-File (spi_base.sdc)

J.2 Quartus II Settings File

```
set_global_assignment -name FAMILY "Cyclone III"
set_global_assignment -name DEVICE EP3C25F324C8
set_global_assignment -name TOP_LEVEL_ENTITY spi_base_wrapper
set_global_assignment -name ORIGINAL_QUARTUS_VERSION "7.2 SP2"
set_global_assignment -name PROJECT_CREATION_TIME_DATE "14:46:41  MARCH 12, 2008"
set_global_assignment -name LAST_QUARTUS_VERSION "8.0 SP1"
set_global_assignment -name USE_GENERATED_PHYSICAL_CONSTRAINTS OFF -section_id eda_palace
set_global_assignment -name USER_LIBRARIES "C:/Programme/altera/72/ip/ddr_high_perf/lib;"
set_global_assignment -name DEVICE_FILTER_PACKAGE FBGA
set_global_assignment -name DEVICE_FILTER_PIN_COUNT 324

set_location_assignment PIN_V9 -to osc_clk
set_location_assignment PIN_N2 -to reset_n
set_location_assignment PIN_D18 -to flash_write_n
set_location_assignment PIN_E2 -to flash_select_n
set_location_assignment PIN_D17 -to flash_read_n
set_location_assignment PIN_C3 -to flash_reset_n
set_instance_assignment -name VIRTUAL_PIN ON -to flash_ssram_adr[0]
set_location_assignment PIN_E12 -to flash_ssram_adr[1]
set_location_assignment PIN_A16 -to flash_ssram_adr[2]
set_location_assignment PIN_B16 -to flash_ssram_adr[3]
set_location_assignment PIN_A15 -to flash_ssram_adr[4]
set_location_assignment PIN_B15 -to flash_ssram_adr[5]
set_location_assignment PIN_A14 -to flash_ssram_adr[6]
set_location_assignment PIN_B14 -to flash_ssram_adr[7]
set_location_assignment PIN_A13 -to flash_ssram_adr[8]
set_location_assignment PIN_B13 -to flash_ssram_adr[9]
set_location_assignment PIN_A12 -to flash_ssram_adr[10]
set_location_assignment PIN_B12 -to flash_ssram_adr[11]
set_location_assignment PIN_A11 -to flash_ssram_adr[12]
set_location_assignment PIN_B11 -to flash_ssram_adr[13]
set_location_assignment PIN_C10 -to flash_ssram_adr[14]
set_location_assignment PIN_D10 -to flash_ssram_adr[15]
set_location_assignment PIN_E10 -to flash_ssram_adr[16]
set_location_assignment PIN_C9 -to flash_ssram_adr[17]
set_location_assignment PIN_D9 -to flash_ssram_adr[18]
set_location_assignment PIN_A7 -to flash_ssram_adr[19]
set_location_assignment PIN_A6 -to flash_ssram_adr[20]
set_location_assignment PIN_B18 -to flash_ssram_adr[21]
set_location_assignment PIN_C17 -to flash_ssram_adr[22]
set_location_assignment PIN_C18 -to flash_ssram_adr[23]
set_location_assignment PIN_G14 -to flash_ssram_adr[24]
set_location_assignment PIN_B17 -to flash_ssram_adr[25]
set_location_assignment PIN_H3 -to flash_ssram_dat[0]
set_location_assignment PIN_D1 -to flash_ssram_dat[1]
set_location_assignment PIN_A8 -to flash_ssram_dat[2]
set_location_assignment PIN_B8 -to flash_ssram_dat[3]
set_location_assignment PIN_B7 -to flash_ssram_dat[4]
set_location_assignment PIN_C5 -to flash_ssram_dat[5]
set_location_assignment PIN_E8 -to flash_ssram_dat[6]
set_location_assignment PIN_A4 -to flash_ssram_dat[7]
set_location_assignment PIN_B4 -to flash_ssram_dat[8]
set_location_assignment PIN_E7 -to flash_ssram_dat[9]
set_location_assignment PIN_A3 -to flash_ssram_dat[10]
set_location_assignment PIN_B3 -to flash_ssram_dat[11]
set_location_assignment PIN_D5 -to flash_ssram_dat[12]
set_location_assignment PIN_B5 -to flash_ssram_dat[13]
set_location_assignment PIN_A5 -to flash_ssram_dat[14]
set_location_assignment PIN_B6 -to flash_ssram_dat[15]
set_location_assignment PIN_C16 -to flash_ssram_dat[16]
set_location_assignment PIN_D12 -to flash_ssram_dat[17]
set_location_assignment PIN_E11 -to flash_ssram_dat[18]
set_location_assignment PIN_D2 -to flash_ssram_dat[19]
set_location_assignment PIN_E13 -to flash_ssram_dat[20]
set_location_assignment PIN_E14 -to flash_ssram_dat[21]
set_location_assignment PIN_A17 -to flash_ssram_dat[22]
set_location_assignment PIN_D16 -to flash_ssram_dat[23]
set_location_assignment PIN_C12 -to flash_ssram_dat[24]
set_location_assignment PIN_A18 -to flash_ssram_dat[25]
set_location_assignment PIN_F8 -to flash_ssram_dat[26]
set_location_assignment PIN_D7 -to flash_ssram_dat[27]
set_location_assignment PIN_F6 -to flash_ssram_dat[28]
set_location_assignment PIN_E6 -to flash_ssram_dat[29]
```

```
set_location_assignment PIN_G6 -to flash_ssram_dat[30]
set_location_assignment PIN_C7 -to flash_ssram_dat[31]
set_location_assignment PIN_E9 -to ssram_outputenable_n
set_location_assignment PIN_F9 -to ssram_chipenable_n
set_location_assignment PIN_F10 -to ssram_bw_n[0]
set_location_assignment PIN_F11 -to ssram_bw_n[1]
set_location_assignment PIN_F12 -to ssram_bw_n[2]
set_location_assignment PIN_F13 -to ssram_bw_n[3]
set_location_assignment PIN_F7 -to ssram_adsc_n
set_location_assignment PIN_G13 -to ssram_bwe_n
set_location_assignment PIN_A2 -to ssram_clk
set_location_assignment PIN_A1 -to hsmc_clk
set_location_assignment PIN_D14 -to cpu_clk
set_location_assignment PIN_F1 -to button[0]
set_location_assignment PIN_F2 -to button[1]
set_location_assignment PIN_A10 -to button[2]
set_location_assignment PIN_B10 -to button[3]
set_location_assignment PIN_P13 -to led[0]
set_location_assignment PIN_P12 -to led[1]
set_location_assignment PIN_N12 -to led[2]
set_location_assignment PIN_N9 -to led[3]
set_location_assignment PIN_P13 -to led_0
set_location_assignment PIN_P12 -to led_1
set_location_assignment PIN_N12 -to led_2
set_location_assignment PIN_N9 -to led_3
set_location_assignment PIN_N8 -to sd_cs[0]
set_location_assignment PIN_L6 -to sd_mosi
set_location_assignment PIN_M2 -to sd_sclk
set_location_assignment PIN_M3 -to sd_miso
set_location_assignment PIN_E1 -to audio_sda
set_location_assignment PIN_F3 -to audio_scl
set_location_assignment PIN_M1 -to HC_AUD_ADCLRCK
set_location_assignment PIN_A9 -to HC_AUD_ADCDAT
set_location_assignment PIN_R2 -to HC_AUD_DACLRCK
set_location_assignment PIN_R1 -to HC_AUD_DACDAT
set_location_assignment PIN_E17 -to HC_AUD_BCLK
set_location_assignment PIN_A1 -to HC_AUD_XCK
set_location_assignment PIN_H6 -to eeprom_scl
set_location_assignment PIN_D3 -to eeprom_sda
set_location_assignment PIN_M5 -to HC_PS2_CLK
set_location_assignment PIN_T1 -to HC_PS2_DAT
set_location_assignment PIN_U2 -to mem_clk
set_location_assignment PIN_V2 -to mem_clk_n
set_location_assignment PIN_V1 -to mem_cs_n
set_location_assignment PIN_R13 -to mem_cke
set_location_assignment PIN_U1 -to mem_addr[0]
set_location_assignment PIN_U5 -to mem_addr[1]
set_location_assignment PIN_U7 -to mem_addr[2]
set_location_assignment PIN_U8 -to mem_addr[3]
set_location_assignment PIN_P8 -to mem_addr[4]
set_location_assignment PIN_P7 -to mem_addr[5]
set_location_assignment PIN_P6 -to mem_addr[6]
set_location_assignment PIN_T14 -to mem_addr[7]
set_location_assignment PIN_T13 -to mem_addr[8]
set_location_assignment PIN_V13 -to mem_addr[9]
set_location_assignment PIN_U17 -to mem_addr[10]
set_location_assignment PIN_V17 -to mem_addr[11]
set_location_assignment PIN_U16 -to mem_addr[12]
set_location_assignment PIN_V11 -to mem_ba[0]
set_location_assignment PIN_V12 -to mem_ba[1]
set_location_assignment PIN_V16 -to mem_ras_n
set_location_assignment PIN_T4 -to mem_cas_n
set_location_assignment PIN_U15 -to mem_we_n
set_location_assignment PIN_U4 -to mem_dq[0]
set_location_assignment PIN_V4 -to mem_dq[1]
set_location_assignment PIN_R8 -to mem_dq[2]
set_location_assignment PIN_V5 -to mem_dq[3]
set_location_assignment PIN_P9 -to mem_dq[4]
set_location_assignment PIN_U6 -to mem_dq[5]
set_location_assignment PIN_V6 -to mem_dq[6]
set_location_assignment PIN_V7 -to mem_dq[7]
set_location_assignment PIN_U13 -to mem_dq[8]
set_location_assignment PIN_U12 -to mem_dq[9]
set_location_assignment PIN_U11 -to mem_dq[10]
set_location_assignment PIN_V15 -to mem_dq[11]
set_location_assignment PIN_U14 -to mem_dq[12]
set_location_assignment PIN_R11 -to mem_dq[13]
set_location_assignment PIN_P10 -to mem_dq[14]
```

```
set_location_assignment PIN_V14 -to mem_dq[15]
set_location_assignment PIN_U3 -to mem_dqs[0]
set_location_assignment PIN_T8 -to mem_dqs[1]
set_location_assignment PIN_V3 -to mem_dm[0]
set_location_assignment PIN_V8 -to mem_dm[1]
set_location_assignment PIN_R4 -to HC_LCD_DATA[0]
set_location_assignment PIN_T17 -to HC_LCD_DATA[1]
set_location_assignment PIN_T18 -to HC_LCD_DATA[2]
set_location_assignment PIN_L16 -to HC_LCD_DATA[3]
set_location_assignment PIN_M17 -to HC_LCD_DATA[4]
set_location_assignment PIN_N6 -to HC_LCD_DATA[5]
set_location_assignment PIN_M13 -to HC_LCD_DATA[6]
set_location_assignment PIN_N13 -to HC_LCD_DATA[7]
set_location_assignment PIN_D14 -to HC_NCLK
set_location_assignment PIN_R17 -to HC_DEN
set_location_assignment PIN_M14 -to HC_HD
set_location_assignment PIN_L13 -to HC_VD
set_location_assignment PIN_R18 -to HC_GREST
set_location_assignment PIN_M6 -to HC_SCEN
set_location_assignment PIN_T2 -to HC_SDA
set_location_assignment PIN_N17 -to HC_ADC_PENIRQ_N
set_location_assignment PIN_L18 -to HC_ADC_DOUT
set_location_assignment PIN_K18 -to HC_ADC_BUSY
set_location_assignment PIN_U18 -to HC_ADC_DIN
set_location_assignment PIN_V18 -to HC_ADC_DCLK
set_location_assignment PIN_R5 -to HC_ADC_CS_N

set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to ssram_adsc_n
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to ssram_bw_n
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to ssram_chipenable_n
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to ssram_outputenable_n
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to ssram_bwe_n
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to ssram_clk

set_instance_assignment -name TCO_REQUIREMENT "3.3 ns" -from * -to ssram_adsc_n
set_instance_assignment -name TCO_REQUIREMENT "3.3 ns" -from * -to ssram_bw_n
set_instance_assignment -name TCO_REQUIREMENT "3.3 ns" -from * -to ssram_bwe_n
set_instance_assignment -name TCO_REQUIREMENT "3.3 ns" -from * -to ssram_chipenable_n
set_instance_assignment -name TCO_REQUIREMENT "3.3 ns" -from * -to ssram_outputenable_n
set_instance_assignment -name TCO_REQUIREMENT "3.3 ns" -from * -to flash_ssram_adr
set_instance_assignment -name TSU_REQUIREMENT "6 ns" -from * -to flash_ssram_dat

set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -section_id Top
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
set_global_assignment -name PARTITION_COLOR 2147039 -section_id Top
set_global_assignment -name LL_ROOT_REGION ON -section_id "Root Region"
set_global_assignment -name LL_MEMBER_STATE LOCKED -section_id "Root Region"

set_global_assignment -name RESERVE_DATA0_AFTER_CONFIGURATION "USE AS REGULAR IO"
set_global_assignment -name RESERVE_DATA1_AFTER_CONFIGURATION "USE AS REGULAR IO"
set_global_assignment -name RESERVE_DATA7_THROUGH_DATA2_AFTER_CONFIGURATION "USE AS REGULAR IO"
set_global_assignment -name RESERVE_FLASH_NCE_AFTER_CONFIGURATION "USE AS REGULAR IO"
set_global_assignment -name RESERVE_OTHER_AP_PINS_AFTER_CONFIGURATION "USE AS REGULAR IO"
set_global_assignment -name SEED 4
set_global_assignment -name CYCLONEII_OPTIMIZATION_TECHNIQUE SPEED
set_global_assignment -name ROUTER_TIMING_OPTIMIZATION_LEVEL MAXIMUM

set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "2.5 V"

set_global_assignment -name ENABLE_ADVANCED_IO_TIMING ON
set_global_assignment -name DEVICE_FILTER_SPEED_GRADE 8
set_global_assignment -name POWER_PRESET_COOLING_SOLUTION "23 MM HEAT SINK WITH 200 LFPM AIRFLOW"
set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE (CONSERVATIVE)"
set_global_assignment -name CYCLONEIII_CONFIGURATION_SCHEME "ACTIVE PARALLEL"
set_global_assignment -name ON_CHIP_BITSTREAM_DECOMPRESSION OFF
set_global_assignment -name NUM_PARALLEL_PROCESSORS 2

set_global_assignment -name OPTIMIZE_FAST_CORNER_TIMING ON
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[0]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[10]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[11]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[12]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[1]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[2]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[3]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[4]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[5]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[6]
```

```
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[7]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[8]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_addr[9]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_ba[0]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_ba[1]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_cas_n
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_cke
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_clk
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_clk_n
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_cs_n
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dm[0]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dm[1]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[0]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[10]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[11]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[12]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[13]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[14]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[15]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[1]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[2]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[3]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[4]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[5]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[6]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[7]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[8]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dq[9]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dqs[0]
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to mem_dqs[1]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_ras_n
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to mem_we_n
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[0]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[10]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[11]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[12]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[1]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[2]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[3]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[4]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[5]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[6]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[7]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[8]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_addr[9]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_ba[0]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_ba[1]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_cas_n
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_cke
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_clk
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_clk_n
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_cs_n
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dm[0]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dm[1]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[0]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[10]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[11]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[12]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[13]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[14]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[15]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[1]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[2]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[3]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[4]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[5]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[6]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[7]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[8]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dq[9]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dqs[0]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_dqs[1]
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_ras_n
set_instance_assignment -name IO_STANDARD "SSTL-2 CLASS I" -to mem_we_n
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dm[0]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dm[1]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[0]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[10]
```

```
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[11]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[12]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[13]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[14]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[15]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[1]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[2]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[3]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[4]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[5]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[6]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[7]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[8]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dq[9]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dqs[0]
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1244174944 -to mem_dqs[1]
set_global_assignment -name TIMEQUEST_DO_REPORT_TIMING ON
set_global_assignment -name ENABLE_DA_RULE "C101, C102, C103, C104, C105, C106, R101, R102, R103,
R104, R105, T101, T102, A101, A102, A103, A104, A105, A106, A107, A108, A109, A110, S101, S102,
S103, S104, D101, D102, D103, H101, H102, M101, M102, M103, M104, M105"
set_global_assignment -name PHYSICAL_SYNTHESIS_COMBO_LOGIC ON
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION ON
set_global_assignment -name PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING ON
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_RETIMING ON
set_global_assignment -name PHYSICAL_SYNTHESIS_EFFORT EXTRA
set_global_assignment -name PHYSICAL_SYNTHESIS_COMBO_LOGIC_FOR_AREA ON
set_global_assignment -name PHYSICAL_SYNTHESIS_MAP_LOGIC_TO_MEMORY_FOR_AREA ON
set_global_assignment -name NOMINAL_CORE_SUPPLY_VOLTAGE 1.2V
set_global_assignment -name SDC_FILE ddr_sdram_phy_ddr_timing.sdc
set_global_assignment -name BDF_FILE spi_base_wrapper.bdf
set_global_assignment -name QIP_FILE spi_base.qip
set_global_assignment -name SDC_FILE spi_base.sdc
set_global_assignment -name VERILOG_FILE audio_join.v
set_global_assignment -name VERILOG_FILE touch_panel_clock_mux.v
```

Quelltext 37: Quartus II Settings File (spi_base.qsf)

Abbildungsverzeichnis

Abbildung 1: Implementierungsbeispiel Instruktionssystem....4

Abbildung 2: Fraunhofer Audio-Plattform, Quelle: Fraunhofer Institute, CorePool, 2002…5

Abbildung 3: Entwicklungssystem mit den verwendeten Schnittstellen. Quelle: Nios II Embedded Evaluation Kit Quick start guide....6

Abbildung 4: Ordnerstruktur der Beispielanwendungen....8

Abbildung 5: Projekt-Explorer mit „Hello World“-Projekt....8

Abbildung 6: PS/2 Timing, Quelle: Wikipedia....10

Abbildung 7: PS/2 Core mit Avalon-MM (Memory Mapping)....11

Abbildung 8: PS/2 Blockdiagramm....11

Abbildung 9: IP-Core Entwicklungsprozess Übersicht....12

Abbildung 10: SOPC-Builder Systemdefinition mit Minimalprojekt, bestehend aus CPU, On-Chip RAM und JTAG UART....13

Abbildung 11: Generiertes System integriert in Quartus II Projekt und grafisches Top-Modul....13

Abbildung 12: Altera's Component Editor mit Signalzuordnung....14

Abbildung 13: Wishbone- zu Avalon-Bus wrapping....14

Abbildung 14: Iterativer System-Entwicklungsprozess....15

Abbildung 15: Wellenformdarstellung einer Testbench-Simulation....16

Abbildung 16: Ebenen eines HAL-basierten Systems....17

Abbildung 17: System-Library-Project als Verbindung zwischen SOC und Applikation…18

Abbildung 18: Ausschnitt aus der GNU-Profiler-Ausgabe....18

Abbildung 19: Software Entwicklungsprozess....19

Abbildung 20: Teilung des Systemaufbaus in Hard- und Software....20

Abbildung 21: Komponenten aus unterschiedlichen Quellen werden im System kombiniert....21

Abbildung 22: Prozessoraufwand bei der SPI-Datenübertragung und Optimierung im Vergleich....22

Abbildung 23: SPI-Master Blockdiagramm....24

Abbildung 24: SPI-Master Transceiver- Zustandsautomat....25

Abbildung 25: SPI-Master Modul-Hierarchie....26

Abbildung 26: SPI-Master Simulation, Wellenformdarstellung der Transfer-Operation.…27

Abbildung 27: SPI-Master System Integration (Bussystem Darstellung).....28

Abbildung 28: SD-Karten Lesegeschwindigkeitsmessung bei unterschiedlichen Optimierungsphasen.....30

Abbildung 29: WM8731-CODEC Schnittstellen (Schaltungsausschnitt). Quelle: LCD Multimedia Daughtercard Reference Manual 31

Abbildung 30: WM8731 Audiodaten Protokoll („Left Justified Mode“).....31

Abbildung 31: WM8731 IP-Core Blockdiagramm mit Darstellung der Clock-Domains.……32

Abbildung 32: WM8731 IP-Core Sende- und Empfangs-Zustandsautomaten mit gemeinsamem Idle-Zustand.....34

Abbildung 33: Sende- bzw. Empfangs-FIFO mit unterschiedlich breiten Samples gefüllt 34

Abbildung 34: Signal Synchronisations-Logik.....36

Abbildung 35: Glitch-freie Takt-Umschaltung.....36

Abbildung 36: WM8731 IP-Core Modul-Hierarchie.....37

Abbildung 37: Wellenformdarstellung der RTL-Simulation.....38

Abbildung 38: WM8731-Core System Integration (Bussystem Darstellung).....39

Abbildung 39: Schematischer Aufbau der 64-Bit-ALU.....42

Abbildung 40: 64-Bit-ALU Modul-Hierarchie.....43

Abbildung 41: 64-Bit-ALU RTL-Simulation.....44

Abbildung 42: 64-Bit-ALU System-Integration (SOPC-Builder Ausschnitt).....45

Abbildung 43: Hardwareorientierte Darstellung des Systemaufbaus.....47

Abbildung 44: Audio-Plattform Projektstruktur.....48

Abbildung 45: System Clock-Domains.....49

Abbildung 46: Systemkomponenten mit Avalon-Busverbindung.....50

Abbildung 47: System-Bridges (Clock-Crossing-, Pipeline- und Tristate-Bridges).....52

Abbildung 48: Schematischer Aufbau der Audio-Plattform-Softwarebibliothek mit Kennzeichnung der HAL-Devices.....54

Abbildung 49: Lesegeschwindigkeitsmessung mit unterschiedlichen SD-Karten.....57

Abbildung 50: Schreibgeschwindigkeitsmessung mit unterschiedlichen SD-Karten.....57

Abbildung 51: Rechenzeitverteilung der Lesezeit.....58

Abbildung 52: Rechenzeitverteilung der Schreibzeit 58

Abbildung 53: Streaming-API.....59

Abbildung 54: GUI-API Nachrichten-Verteilung (Event-Dispatching).....64

Abbildung 55: MP3-Decoder-Verzeichnisstruktur.....66

Abbildung 56: Grafisches Benutzerinterface.....69

Abbildung 57: „open juke" Zustandsautomat.....70

Abbildung 58: "open juke"-Flussdiagramm der main-Funktion.....72

Quelltextverzeichnis

Quelltext 1: 64-Bit-ALU Testbench Quelltext-Ausschnitt.....44

Quelltext 2: MP3-Decoder Erweiterung unter Verwendung der 64-Bit-ALU Befehle.....67

Quelltext 3: SPI-Master Testprogramm.....83

Quelltext 4: SD-Treiber Testprogramm.....84

Quelltext 5: PS/2, PIO-Button Treiber und Queue API Testprogramm.....85

Quelltext 6: Schreib-Lesegeschwindigkeitsmess-Programm.....88

Quelltext 7: SPI-Master Definitionsdatei (spi_master_defines.v).....90

Quelltext 8: SPI-Master Top-Modul (spi_master_top.v).....95

Quelltext 9: SPI-Master Transceiver (spi_master_transceiver.v).....102

Quelltext 10: SPI-Master Testbench (spi_master_tb.v).....105

Quelltext 11: WM8731 IP-Core Definitionsdatei (wm8731_defines.v).....106

Quelltext 12: WM8731 IP-Core Top-Modul (wm8731_top.v).....111

Quelltext 13: WM8731 IP-Core Transmitter (wm8731_transmitter.v).....118

Quelltext 14: Takt-Multiplexer (gf_mux.v).....119

Quelltext 15: Synchronisations-Logik (ptt_ttp.v).....119

Quelltext 16: WM8731 IP-Core Testbench (wm8731_tb.v).....123

Quelltext 17: 64-Bit-ALU Definitionsdatei (mult64_defines.v).....124

Quelltext 18: 64-Bit-ALU Top-Modul (mult64_top.v).....131

Quelltext 19: 64-Bit-ALU Testbench (mult64_tb.v).....135

Quelltext 20: Opencore PS/2 Wishbone-zu-Avalon-Wrapper (oc_ps2_master.v).....136

Quelltext 21: Opencore SPI Wishbone-zu-Avalon-Wrapper (oc_spi_master_top.v).....137

Quelltext 22: Opencore I2C Wishbone-zu-Avalon-Wrapper (i2c_master.v).....138

Quelltext 23: 16-24-Bit Pixel-Converter (lcd_16bit_top.v).....139

Quelltext 24: Touchpanel Clock-Multiplexer touch_panel_clock_mux.v.....139

Quelltext 25: 64-Bit-ALU Header-Datei alu64.h.....142

Quelltext 26: Angepasste MP3-Decoder Polyphase-Routinen nios2_polyphase.c.....145

Quelltext 27: Altera SPI-Master Treiber Header-Datei alt_spi_master.h.....147

Quelltext 28: Altera SPI-Master Treiber alt_spi_master.c.....150

Quelltext 29: SPI-Master Register Header-Datei spi_master_regs.h.....151

Quelltext 30: SPI-Master Header-Datei spi_master.h.....152

Quelltext 31: SPI-Master Treiber-Implementierung spi_master.c.....156

Quelltext 32: MMC/SD-Karten Protokoll-Header-Datei mmc_card.h.....160

Quelltext 33: MMC/SD-Karten Treiber-Implementierung mmc_spi.c.....172

Quelltext 34: "open juke"-Referenzanwendung Hauptprogramm main.c.....188

Quelltext 35: "open juke"-Referenzanwendung GUI-Header-Datei oj_gui.h.....189

Quelltext 36: Synopsys-Design-Constraints-File (spi_base.sdc).....191

Quelltext 37: Quartus II Settings File (spi_base.qsf).....196

Tabellenverzeichnis

Tabelle 1: SPI-Master EA-Register....23

Tabelle 2: SPI-Master Operationen (Jobs)....23

Tabelle 3: SPI-Master Treiber-Funktionen....29

Tabelle 4: WM8731-Core EA-Register....33

Tabelle 5: Abweichungen von den Referenz-Frequenzen....39

Tabelle 6: WM8731 I2C-Konfigurations-Interface-Funktionen....40

Tabelle 7: WM8731 IP-Core Treiber-Funktionen....40

Tabelle 8: 64-Bit-ALU Parameter....45

Tabelle 9: Systemkomponenten....50

Tabelle 10: Nios II Konfiguration....51

Tabelle 11: FPGA-Ressourcen-Verbrauch....53

Tabelle 12: SD-Karten Treiber-Dateien....55

Tabelle 13: FAT-Filesystem Treiber-Dateien....56

Tabelle 14: Opencore I2C-Treiber....59

Tabelle 15: Audio-API-Funktionen....60

Tabelle 16: WAV-API Funktionen....60

Tabelle 17: Gekapselte MP3-Decode- API innerhalb der Audio-Plattform....60

Tabelle 18: PIO-LCD-Treiber API....61

Tabelle 19: Framebuffer-API....61

Tabelle 20: Grafik API....62

Tabelle 21: Altera-SPI Treiber Funktionen....62

Tabelle 22: Touchpanel-Treiber....63

Tabelle 23: Ausschnitt aus den GUI-API Funktionen....63

Tabelle 24: GUI-API Button Funktionen....64

Tabelle 25: PS/2 Tastatur-Treiber-Funktionen....64

Tabelle 26: PIO-Button-Treiber....65

Tabelle 27: Queue-API....65

Tabelle 28: Audio-Plattform Integration, Linker- und Compiler-Einstellungen....66

Tabelle 29: Helix MP3-Decoder-API....68

Tabelle 30: MP3-Decoder Integration, Linker- und Compiler Einstellungen.....68

Tabelle 31: Steuerbefehle der Referenzapplikation.....75

Tabelle 32: Entwicklungswerkzeuge.....76

Tabelle 33: Avalon zu Wishbone Signalzuordnung.....77

Tabelle 34: Erstellte Quartus II Projekte.....78

Tabelle 35: Erstellte Nios II Projekte.....79

Tabelle 36: 64-Bit-ALU Befehlssatz.....81

Tabelle 37: Ergebnisse der Schreib-Lesegeschwindigkeits-Messung (Maximal-Werte sind hervorgehoben).....82

Tabelle 38: Ergebnisse der Schreib- und Lesezeitmessung mit Verteilung der Rechenzeit auf FAT-Dateihandling und SD-Datentransfer.....82

Abkürzungsverzeichnis

AHDL	Altera Hardware Description Language
ALU	Arithmetic Logical Unit
API	Application Programming Interface
ARM	Acorn Risc Machine
CAN	Controller Area Network
CCB	Clock Crossing Bridge
CDC	Clock Domain Crossing
CODEC	Coder Decoder
CPU	Central Processing Unit
CS	Chip Select
CVS	Concurrent Versions System
DMA	Direct Memory Access
EA	Eingang/Ausgang
EDA	Electronic Design Automation
EDS	Embedded Design Suite
EEPROM	Electrically Erasable Programmable Read-Only Memory
ELF	Executable and Linking Format
FAT	File Allocation Table
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
IC	Integrated Circuit
IDE	Integrated Development Environment
IP	Intellectual Property
IRQ	Interrupt Request
ISR	Interrupt Service Routine

JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LE	Logic Element
LIN	Local Interconnect Network
MOSI	Master out slave in
MISO	Master in slave out
NIC	Network Interface Card
PIO	Parallel Input/Output
PLL	Phase-locked loop
RAM	Random Access Memory
RTL	Register Transfer Level
SCLK	Slave Clock
SD	Secure Digital
SOC	System On a Chip
SOF	SRAM Object File
SOPC	System On a Programmable Chip
SPI	Synchronous Serial Interface
UART	Universal Asynchronous Receiver Transmitter
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Literaturverzeichnis

[1] **NXP**: LPC2880; LPC2888 - 16/32-bit ARM microcontrollers; 8 kB cache, up to 1 MB flash, Hi-Speed USB 2.0 device, and SDRAM memory interface, www.nxp.com/pip/LPC2880_LPC2888_3.html, letzter Zugriff 24.07.08

[2] **Fraunhofer Institute Integrated Circuits**: Audio Platform, http://www.corepool.com/products/audio/index.htm, letzter Zugriff 24.07.08

[3] **ARC International**: Market Ready Audio, Video, and Imaging Solutions, http://www.arc.com/productsandsolutions/index.html, letzter Zugriff 24.07.08

[4] **Altera Corporation**: Nios II Embedded Evaluation Kit - Cyclone III Edition, 10/2007, http://www.altera.com/products/devkits/altera/kit-cyc3-embedded.html#documentation, letzter Zugriff: 14.08.08

[5] **Kesel, F.; Bartholomä, R.:** Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs, Oldenbourg, 2006

[6] **Smith, D.J.:** VHDL & Verilog Compared & Contrasted, http://www.angelfire.com/in/rajesh52/verilogvhdl.html, letzter Zugriff: 10.07.08

[7] **Hoppe, B.**: Verilog: Modellbildung für Synthese und Verifikation, Oldenburg Verlag, 2006 München.

[8] **Altera Corporation**: Quartus II Software—Subscription Edition and Web Edition Comparison, http://www.altera.com/products/software/products/quartus2web/features/sofquarweb_features.html, letzter Zugriff: 24.07.08

[9] **Altera Corporation**: OpenCore Plus Evaluation of Megafunctions, 11/2007, http://www.altera.com/literature/an/an320.pdf, letzter Zugriff: 19.07.08

[10] **Altera Corporation**: Quartus II Development Software Literature, http://www.altera.com/literature/lit-qts.jsp, letzter Zugriff: 11.07.08

[11] **Wikipedia:** IP-Core, http://de.wikipedia.org/wiki/IP-Core, letzter Zugriff: 22.06.08

[12] **Altera Corporation**: System Interconnect Fabric for Memory-Mapped Interfaces, 03/2008, http://www.altera.com/literature/hb/qts/qts_qii54003.pdf, letzter Zugriff: 10.07.08

[13] **Hamblen, J. O.**: Rapid Prototyping of Digital Systems - SOPC Edition, Springer Verlag, 2008

[14] **Altera Corporation**: Developing Components for SOPC Builder, Quartus II 7.2 Handbook, Volume 4, 10/2007, http://www.altera.com/literature/lit-qts.jsp, letzter Zugriff: 07.07.08

[15] **Herveille, R.; OPENCORES.ORG**: WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, http://www.opencores.org/projects.cgi/web/wishbone/wbspec_b3.pdf, letzter Zugriff: 07.07.08

[16] **Altera Corporation**: Mentor Graphics ModelSim Support, 05/2008, http://www.altera.com/literature/hb/qts/qts_qii53001.pdf, letzter Zugriff: 12.08.08

[17] **Altera Corporation**: Quartus II Verification Methods, http://www.altera.com/products/software/products/quartus2/verification/qts-design_flow-verification2.html, letzter Zugriff: 12.07.08

[18] **Altera Corporation**: Developing Device Drivers for the Hardware Abstraction Layer, 5/2008, http://www.altera.com/literature/hb/nios2/n2sw_nii52005.pdf. Letzter Zugriff: 26.07.08

[19] **Altera Corporation**: Profiling Nios II Systems, 7/2008, http://www.altera.com/literature/an/an391.pdf. Letzter Zugriff: 27.07.08

[20] **MAXIM**: MAX3378, ±15kV ESD-Protected, 1µA, 16Mbps, Dual/Quad Low-Voltage Level Translators in UCSP, 11/2007 Rev.2

[21] **Altera Corporation**: Single- and Dual-Clock FIFO Megafunction, 05/2007 Version 4.0, http://www.altera.com/literature/ug/ug_fifo.pdf, letzter Zugriff: 01.08.08

[22] **Altera Corporation**: Avalon Interface Specifications, 03/2008 Version 1.0, http://www.altera.com/literature/manual/mnl_avalon_spec.pdf, letzter Zugriff: 07.07.08

[23] **Altera Corporation**: Avalon Memory-Mapped Bridges, Quartus II 8.0 Handbook, Volume 4, 05/2008, http://www.altera.com/literature/hb/qts/qts_qii54020.pdf, letzter Zugriff: 02.08.08

[24] **Altera Corporation**: LCD Multimedia Daughtercard - Reference Manual, 11/2007

[25] **Crews, M.; Yuenyongsgool, Y.:** Practical design for transferring signals between clock domains, http://www.edn.com/contents/images/276202.pdf, Philips Semiconductors, 2003,letzter Zugriff: 14.08.08

[26] **Altera Corporation**: Using DCFIFO for Data Transfer between Asynchronous Clock Domains, 12/2007, http://www.altera.com/literature/an/an473.pdf, letzter Zugriff: 09.08.08

[27] **Rafey, M.:** Glitch protection for unrelated clock sources, http://www.design-reuse.com/articles/5827/techniques-to-make-clock-switching-glitch-free.html, EEdesign, 2003,letzter Zugriff: 14.08.08

[28] **Wolfson Microelectronics**: WM8731, Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates, Datenblatt, http://www.wolfsonmicro.com/products/WM8731, letzter Zugriff: 07.07.08

[29] **Altera Corporation**: Nios II Processor, 05/2008, http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1_01.pdf, letzter Zugriff: 27.08.08

[30] **Zimmermann, B.; Wolf, D.**: LogFrog - Ein Langzeitdatenerfassungssystem, Studienarbeit, Fachhochschule Bielefeld, Juli 2007

[31] **SanDisk**: Secure Digital Card, Product Manual, 2003 Version 1.9, http://www.cs.ucr.edu/~amitra/sdcard/ProdManualSDCardv1.9.pdf, letzter Zugriff: 07.07.08

[32] **Schwarz, A.:** ARM MP3/AAC Player, http://www.mikrocontroller.net/articles/ARM_MP3/AAC_Player, letzter Zugriff: 08.08.08

[33] **Altera Corporation**: Nios II Custom Instruction - User Guide, 2007 Version 1.4, http://www.altera.com/literature/ug/ug_nios2_custom_instruction.pdf, letzter Zugriff: 07.07.08

[34] **Altera Corporation**: RAM Megafunction, User Guide, 03/2007, http://www.altera.com/literature/ug/ug_ram.pdf, letzter Zugriff: 07.08.08

[35] **Altera Corporation**: The LPM Quick Reference Guide, 12/1996, http://www.altera.com/literature/catalogs/lpm.pdf, letzter Zugriff: 07.08.08

[36] **Altera Corporation**: Scatter-Gather DMA Controller Core, http://www.altera.com/literature/hb/nios2/qts_qii55003.pdf, Letzter Zugriff: 03.09.08

[37] **Analog Devices**: AD7843, Touch Screen Digitizer, Datenblatt, http://www.analog.com/en/analog-to-digital-converters/ad-converters/AD7843/products/product.html, Letzter Zugriff: 07.07.08