UNIKASSEL
VERSITÄT

Fachgebiet Rechnerarchitektur
und Systemprogrammierung
Prof. Dr.-Ing. habil. Josef Börcsök

Integration eines 32-Bit Softcore Prozessors und die Anwendung auf der komplexen Sensorik

Emil Gracic
Matrikelnummer 25249338

Erstgutachter: Prof. Dr.-Ing. habil. Josef Börcsök
Zweitgutachter: Prof. Dr. Michael Schwarz

Betreuer: Dr.-Ing. Ali Hayek

Erklärung

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Kassel, den _____ _____

Danksagung

Ich bedanke mich herzlich bei Herr Prof. Dr.-Ing. habil. Josef Börcsök für ein anregendes Thema sowie für sehr gute Arbeitsbedingungen.
Herrn Prof. Dr. Michael Schwarz danke ich für die Übernahme der Rolle des zweiten Prüfers.
Meinem Betreuer, Herrn Dr. Ali Hayek, danke ich für ständige Ansprechbarkeit, sowie für sein Engagement, dass diese Arbeit effektiv abgeschlossen werden konnte. Besonders seine Beratung bei der Zeitplanung war sehr hilfreich.
Am Ende bedanke ich mich bei allen, die mich während dieser Arbeit unterstützt haben, vor allem bei meiner Familie.

Inhaltsverzeichnis

Abstract

This thesis describes the development of an SoC (**S**ystem **O**n **C**hip), namely a Nios II system, which consists of the Nios II Softcoreprocessor, several bus systems and a variety interfaces to the peripherals. One of the most important parts of the periphery are sensors - an LM74 temperature sensor and a TSL250R light sensor.
The DBC3C40 Cyclone III Development Board of the firma Altera® is used. This board allows very wide spectrum of the develompment possibilities (ethernet interface, CAN-bus interface, TFT-LCD controller interface etc.).

The hardware design of the Nios II System will be provided by the Quartus II 9.0 software. The most important part of this software is the SOPC Builder (**S**ystem **O**n **P**rogrammable **C**hip) tool. With this tool, the single system components are selected, and the connections between them are determinated. After the successful compilation process, the hardware design is loaded on the develompment board.

To perform the test of the hardware design, the Linux operating system (Microcontroller Linux) is used. Two tasks, the driver for the temperature and light sensor, are processed asynchronously by this operating system. With help of the driver for the ethernet interface, the results delivered by the sensors are displayed via a simple Boa webserver.

Kurze Zusammenfassung

Diese Diplomarbeit beschreibt die Entwicklung eines SoCs (**S**ystem **O**n **C**hip). Das Thema ist ein Nios II System, das aus dem Nios II Softcoreprozessor, verschiedenen Bussystemen sowie den Schnittstellen zu einer Vielzahl der Peripherie besteht. Einer der wichtigsten Teile dieser Peripherie sind Sensoren – der LM74 Temperatursensor und TSL250R Lichtsensor. Das DBC3C40 Entwicklungsboard der Firma Altera® kommt zum Einsatz, welches dem Entwickler ein sehr breites Spektrum von Entwicklungsmöglichkeiten zur Verfügung stellt (Ethernet-Schnittstelle, CAN-Bus-Schnittstelle, TFT-LCD Controller-Schnittstelle etc.).

Das Hardwaredesign des Nios II Systems erfolgt mit der Quartus II 9.0 Software. Von größter Bedeutung ist das SOPC Builder (**S**ystem **O**n **P**rogrammable **C**hip) Tool. Mit diesem Tool werden die einzelnen Systemkomponenten ausgewählt, und die Verbindungen zwischen ihnen festgelegt. Nach dem erfolgreichen Kompilierungsprozess, wird das Hardwaredesign auf das Board geladen.

Um den Test des Hardwaredesigns durchzuführen, wird das Linux Betriebssystem (Mikrocontroller Linux) eingesetzt. Zwei Tasks – der Treiber für den Temperatur- bzw. Lichtsensor, werden asynchron vom Betriebssystem abgearbeitet. Mit Hilfe des Treibers für die Ethernetschnittstelle werden die von Sensoren gelieferten Ergebnisse über einen einfachen Boa Webserver bereitgestellt.

1 Einleitung

Das Prozessor-Design auf FPGA-Basis (**F**ield **P**rogrammable **G**ate **A**rray) bietet dem Entwickler ganz neue Freiheitsgrade an – er kann genau den gewünschten Prozessor-Typ mit dem Satz an Peripherie und der optimalen Funktionalität bzw. Performance für seine Applikation entwickeln.

Ein SCP (**S**oft **C**ore **P**rozessor) besteht aus mehreren Source-Code-Dateien, welche in einer Hardware-Beschreibungssprache geschrieben wurden, ganz ähnlich wie ein C- oder Java-Programm. Diese Datei kann von speziellen HDL (**H**ardware **D**escription **L**anguage)-Compilern zu einer digitalen Schaltung synthetisiert werden, welche anschließend in einem FPGA geladen werden kann. Erst dann ist der SCP lauffähig und kann wie ein normaler Hardwareprozessor z. B. in C programmiert werden. Aus Softwaresicht ist nun kein Unterschied mehr erkennbar. Er von einem SOPC Builder generiert, was bedeutet, dass das Entwicklungssystem aufgrund einer vom Entwickler durchgeführten Konfiguration die SCP-Datei generiert. Der wesentliche Vorteil besteht darin, dass der SCP jederzeit wieder verändert und erweitert werden kann, wenn dies für die Zielapplikation vom Vorteil ist.

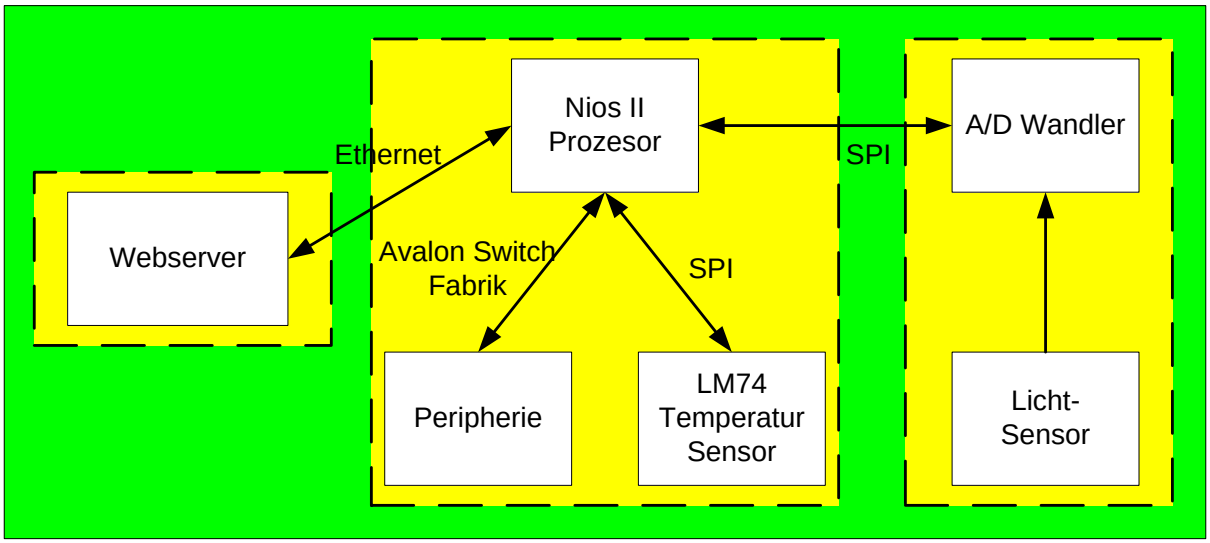

Bild 1-1: Blockschaltbild vom entwickelten System-on-Chip

1.1 Motivation

Falls auf einem FPGA kein Prozessorkern vorhanden ist, und man einen Prozessor einsetzen will, dann kann dies nur mittels Softcore Prozessor geschehen.
Die Firma Altera bietet eine sehr effiziente Software – Quartus II 9.0, für einfaches Hardware-Design mit Alteras FPGAs an. Diese Software enthält ein Tool - SOPC Builder-Tool, mit dem anhand der vorgefertigten IP (**I**ntellectual **P**roperty)-Komponenten ein komplettes SoC erstellt werden kann. Diese IP-Komponenten werden auch IP-Cores genannt (Nios II Prozessor ist auch ein IP-Core).
Als IP-Core wird ein wiederbenutzbarer Teil eines Chipdesigns (im Sinne von Bauplänen) in der Halbleiterindustrie bezeichnet. Diese enthält das geistige Eigentum des Entwicklers/Herstellers.
Alle diesen Komponenten liegen in einer Hardwarebeschreibungssprache (HDL oder VHDL – **V**ery High Speed Integrated Circuit HDL) vor, und können durch den Prozess der Synthese mit der Quartus II 9.0 auf dem FPGA simuliert werden

1.2 Aufgabestellung

Das Ziel dieser Arbeit ist ein komplettes SoC zu entwickeln, das durch eine Sensorik erweitert wird, so dass am Ende ein komplexes System vorliegt, dessen Funktionalität ständig erweitert werden kann.
Als Endprodukt verfügt man über ein SoC, ein eingebettetes Linux Betriebssystem auf dem Nios II Prozessor und eine Testapplikation, welche die Arbeit der Sensoren überwacht und die aktuell abgelieferten Sensorwerte über eine Ethernetschnittstelle auf dem Boa Webserver bereitstellt.
Für das Hardware- und Softwaredesign wurden die Produkte der Firma Altera benutzt. Als Entwicklungsboard wird das DBC3C40 Board benutzt, als Software Quartus II 9.0 für das Hardwaredesign und Nios II 9.0 IDE für die Erstellung der Testapplikation.
Die eingesetzten Sensoren sind: der LM74 Temperatursensor und TSL250R Lichtsensor. Der Temperatursensor ist auf dem Board integriert. Um die Kommunikation mit ihm bereitzustellen, soll ein Treiber geschrieben werden. Der Lichtsensor liegt

extern zum Board und kommuniziert mit dem über den AD7707 Analog-Digital-Wandler. Für diese Kommunikation soll auch ein Treiber geschrieben werden.
Diese zwei Treiber werden vom Betriebssystem als zwei Tasks asynchron bearbeitet.

1.3 Gliederung der Arbeit

Um die Schritte für den Aufbau eines SoC zu verstehen, müssen erst einmal die Grundlagen der SoC-Komponenten erläutert werden. Dies wird im **Kapitel 2 „Stand der Technik“** beschrieben.

Als zentraler Punkt des in dieser Arbeit entwickelten SoC steht ein Softcore Prozessor der Firma Altera, deswegen wird im **Kapitel 3 „Softcore Prozessor Nios II“** die Struktur und Funktionalität dieses Prozessors vorgestellt.

Nachdem die Grundlagen erläutert sind, lässt sich mit der Beschreibung der praktischen Seite dieser Arbeit beginnen. **Kapitel 4 „System-on-Chip Hardwaredesign“** beschreibt den Umgang mit der Software Quartus II 9.0, die benutzt wird um ein komplettes System-on-Chip auf dem DBC3C40 Entwicklungsboard zu realisieren. Dabei werden alle zur Entwicklung nötigen Schritte erklärt.

Um sicher zu stellen, dass das SoC wirklich korrekt funktioniert, wird ein eingebettetes Linux Betriebssystem auf dem Nios II Prozessor eingesetzt. Dies wird im **Kapitel 5 „Softwaredesign – Testapplikation“** beschrieben.

Im Anschluss werden die angeschlossenen Sensoren beschrieben, sowie die Art der Interaktion mit dem Nios II System. Dies war die Aufgabe des **Kapitels 6 „Sensorik“**.

Kapitel 7 „Die Sensortreiber als unabhängige Tasks unter uClinux und Alteras® Triple Speed Ethernet“ beschreibt, wie die Sensortreiber unter Mikrocontroller-Linux als unabhängigen Task gebildet werden können. Da die Ergebnisse dieser Treiber über die Ethernetschnittstelle auf dem Boa Webserver bereitgestellt werden, wird in diesem Kapitel auch Alteras® Triple Speed Ethernet vorgestellt.

Im **Kapitel 8 „Zusammenfassung und Ausblicke“** wird die ganze Arbeit kurz zusammengefasst und die potentielle, zukünftige Arbeit mit diesem SoC erörtert.

2 Stand der Technik

2.1 System-on-Chip

System-on-Chip könnte als Integration aller oder eines großen Teils der Systemfunktionen auf einem Stück Silizium definiert werden, so genannte monolithische Integration. Einsatzsgebiet sind üblicherweise die eingebetteten Systeme.

Früher bestanden die Systeme aus einem Mikroprozessor- oder Mikrocontroller-IC (**I**ntegral **C**ircuit) und vielen anderen auf einer Platine gelöteten ICs. Heute, geht der Trend in der Richtung alle Funktionen auf einer integrierten Schaltung zu realisieren, wobei die digitalen, analogen und mixed-signal Funktionseinheiten integriert werden. Zwei Ziele sind gleich anzumerken – die Miniaturisierung und die Kosteneinsparung.

Jedes SoC besteht aus mindestens drei Komponenten:

- Einem Prozessor
- Einer Speichereinheit
- Und den Peripherie-Einheiten

Diese drei Komponenten sind miteinander mittels einer bestimmten Busstruktur verbunden. Weiteren Komponenten eines SoC sind z. B. A/D Wandler, Grafikschnittstellen z. B. für LCD (**L**iquid **C**rystal **D**isplay), LVDS (**L**ow **V**oltage **D**ifferential **S**ignaling), der Pulsweitenmodulator (PWM) für Motorsteuerung etc.

Bei dem SoC-Entwurf unterscheidet man zwischen fünf Entwurfsebenen:

- Systemebene: ist die am wenigsten detaillierte Ebene. Hier werden die grundlegenden Eigenschaften des Systems beschrieben. Das System wird in Form verschiedener Blöcken, wie z. B. Speicher oder Prozessor, dargestellt. Auf dieser Ebene findet die Partitionierung der gesamten Schaltungsfunktion statt.

- Algorithmische Ebene: das ganze System wird in der Form eines Programms betrachtet, welches aus den Funktionen, Prozeduren, Prozessen und weiteren Kontrollstrukturen besteht.

- Register-Transfer-Ebene: das System wird als ein synchroner Block betrachtet, welcher aus einem operativ arbeitenden Teil (Datenpfad) und einem Steuerwerk besteht. Die Struktur des Systems wird durch die Verknüpfung verschiedener Register, Multiplexer und logischen Einheiten beschrieben. Das Systemverhalten wird in Form endlicher Automaten dargestellt.

- Logikebene: basiert auf der booleschen Algebra.

- Schaltungsebene: Die Systemstruktur wird in Form einer Netzliste dargestellt. Netzliste ist die semantische Beschreibung der elektrischen Verbindungen zwischen bestimmten Bauelementen. Hier kommen die elektrischen Bauelemente wie Transistoren, Dioden, Kapazitäten und Widerstände zum Einsatz. Das Systemverhalten auf dieser Ebene wird durch die Differentialgleichungen beschrieben.

Diese Ebenen werden im Bild 2-1 verdeutlicht.

Der reale Entwurfszyklus eines SoC wird im Bild 2-2 dargestellt. Dieses Zyklus besteht aus drei Phasen:

- Die Spezifikationsphase: formale Beschreibung des Modelles in Form von Texten, Tabellen oder Zeichnungen und das erste VHDL-Modell.

- Die Entwurfsphase: der synthetisierbare VHDL-Code wird in eine technologieunabhängige Beschreibung in der Logikebene überführt.

- Die Implementierungsphase: die Angaben über die zu verwendende Technologie (FPGA, ASIC usw.).

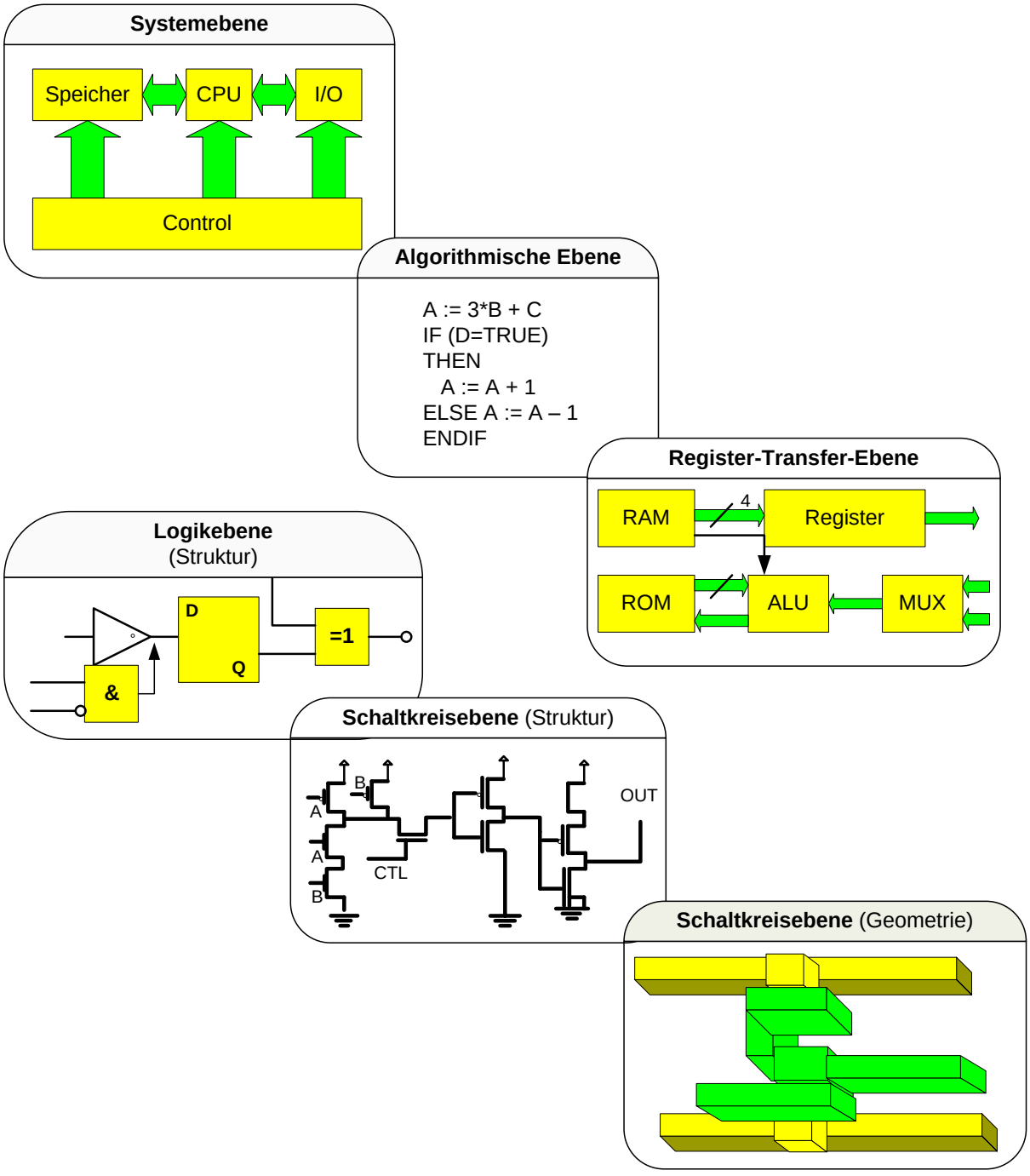

Bild 2-1: Entwurfsebenen des SoC [1]

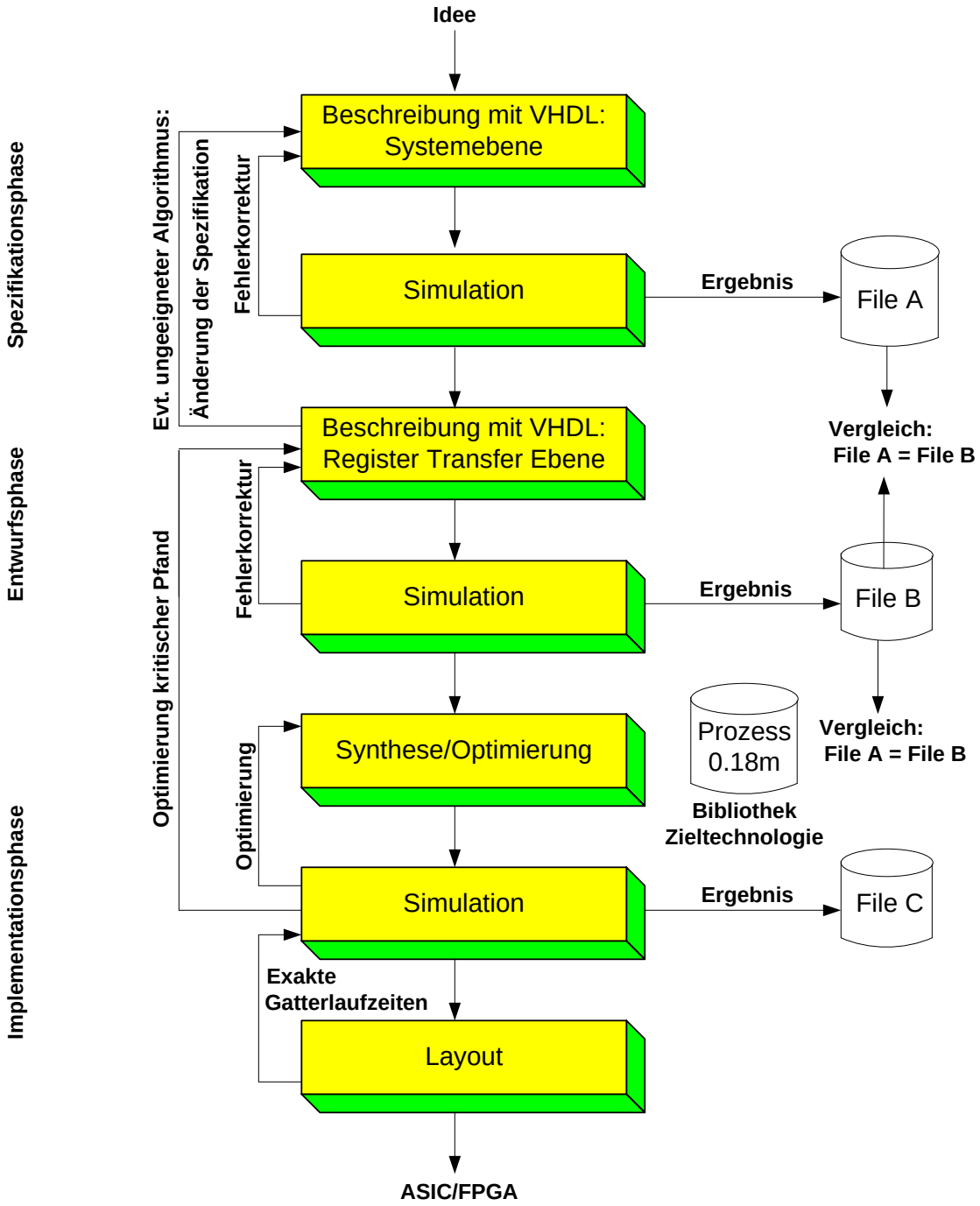

Bild 2-2: Entwurfszyklus eines SoC [1]

Der Entwickler kümmert sich um die Systemebene, die Aufgaben anderer Ebenen werden durch eine bestimmte Software (in diesem Fall durch Quartus II 9.0) automatisch erledigt.

2.2 FPGA

FPGA (**F**ield **P**rogrammable **G**ate **A**rray) ist ein programmierbarer Halbleiterbaustein. Da FPGAs programmierbare, logische Komponenten und programmierbare Verbindungen zwischen diesen Komponenten beinhalten, gelten sie als feinkörnig rekonfigurierbar.

Das FPGA wird als Feld bzw. Matrix gefertigt. Seine Komponenten können zu grundlegenden logischen Bausteinen wie AND, OR, NOR, NOT und NAND aber auch zu der komplexeren Logik wie Decoder, Encoder oder mathematischen Funktionen programmiert und verknüpft werden.

Die Programmierung eines FPGA findet nach der Herstellung statt, so dass die Anforderungen des Anwenders berücksichtigt werden. Die Funktion des FPGAs wird ausschließlich durch die Konfiguration festgelegt und somit kann der gleiche Baustein für verschiedene Schaltungen verwendet werden, was bei den Prototypen und Kleinserien sehr vorteilhaft ist. Diese Eigenschaft macht das FPGA sehr kostengünstig im Vergleich zu einer anwendungsspezifischen integrierten Schaltung (ASIC).
Auf der anderen Seite sind die FPGAs im Allgemeinen langsamer als ASICs und nicht beliebig groß programmierbar. Diese Einschränkung der programmierbaren Logik ist das Ergebnis der Vorbereitung des Herstellers. Die Stufe der Programmierbarkeit eines FPGAs wird an der Anzahl der logischen Komponenten, der I/O Ports, der Flipflops, der Gates usw. festgelegt.

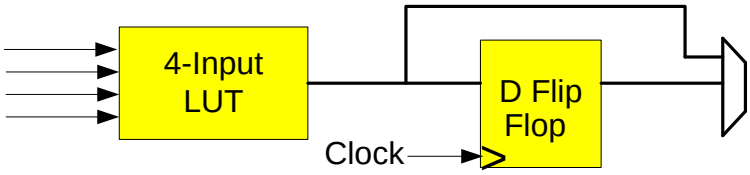

Bild 2-3: Logik-Element eines FPGAs mit Look-Up-Table (LUP) und Flipflop

Wie schon erwähnt, sind die zentralen Elemente des FPGAs programmierbare Logikelemente. In diesen können die grundlegenden sowie komplexeren Funktionen realisiert werden. Wenn eine rein kombinatorische Funktion benötigt wird, kommt ein

nachgeschaltetes Speicherelement zum Einsatz. Dieses Speicherelement kann als Flipflop oder Latch verwendet werden. Die kombinatorische Logik wird meist durch sog. LUTs (**L**ook **U**p **T**able) gebildet, das heißt in einem kleinen Speicher wird für jeden Zustand der Eingänge der Wert abgelegt, den der Ausgang annehmen soll. Der LUT-Speicher kann auch als Teil von Rechenfunktionen eingesetzt werden. An die Logikelemente sind Schaltmatrizen angeschlossen, mit denen über Leitungen die Verbindung zu anderen Elementen des FPGAs hergestellt wird.

Eingangs-/Ausgangs-Blöcke dienen der Kommunikation mit der Außenwelt, über sie werden die Pins des FPGAs mit der Schaltmatrix verbunden. Auch diese Blöcke können an die jeweilige Anwendung angepasst werden, z. B. kann die Ausgangsspannung an den jeweiligen I/O-Standard angepasst werden (TTL/CMOS usw.).
Eine Taktaufbereitung sorgt dafür, dass überall auf dem Chip ein synchroner Takt zu Verfügung steht, zusätzlich kann dieser mit Hilfe von Phase Locked Loop (PLL) Regelkreisen oft noch verdoppelt, reduziert und in der Phase verändert werden. In vielen FPGAs sind außerdem noch zusätzliche fest verdrahtete Funktionen enthalten, wie z. B. Speicher (sog. Block RAM), der sich in vielfältiger Weise konfigurieren lässt. Rein konventionell organisierter Speicher kann hier untergebracht werden und belegt keine LUTs bzw. Logikzellen. Vor allem für Aufgaben der Signalverarbeitung sind Multiplizierer integriert, die resourcenschonender und schneller sind, als solche, die aus Logikzellen zusammengesetzt sind.

Reprogrammierbare FPGAs haben einen speziellen Bereich der Computertechnik erst in nutzbarem Umfang realisierbar gemacht: Selbst konfigurierende Systeme. Diese konfigurieren sich zur Laufzeit entsprechend der geforderten Eigenschaften (z. B. spezielle mathematische Algorithmen) um und erreichen damit gezielte Verarbeitungsgeschwindigkeiten und Parallelität. Als besondere Herausforderung kann man hierbei die Compiler-Entwicklung sehen. Ziel ist es, objektorientiert Logik-Kapazitäten bei Bedarf zur Benutzung zu konfigurieren und nach der Benutzung freizugeben. FPGAs werden gerne zur Echtzeit-Verarbeitung einfacher Algorithmen genutzt, speziell zur Signalverarbeitung (z. B. FFT, FIR), Protokoll-Abarbeitung (Ethernet MAC-Layer, GPRS etc.), Kodierung, Fehlerkorrektur usw., das heißt immer dann, wenn die Bearbeitung eines Datenstroms nicht mehr von einer CPU bewältigt werden kann (Ausgangsdatenstrom gleich groß wie Eingangsdatenstrom, in der Regel mit einer gewissen Latenz). Besonders in Bereichen, in denen Algorithmen bzw. Protokolle einer schnellen Weiterentwicklung unterliegen, ist die Verwendung rekonfigurierbarer FPGAs statt ASICs angebracht (schnelle Marktreife, nachfolgende Fehlerbehebungen, Anpassung an neue Entwicklungen), weil dann nur noch die Firmware aktualisiert werden muss, anstatt der Neuanfertigung und dem Austausch einer integrierten Schaltung.

2.3 Alteras Cyclone III Device Familie

Die Cyclone® III Device Familie bietet eine einzigartige Kombination der hohen Funktionalität, niedrigen Energieverbrauch und niedrigen Kosten an. Sie basiert auf den Halbleiter-Produktionsgesellschaft von Taiwan (TSMC), low-power (LP) Prozess-Technologie, Silikonoptimierungen und verschiedene Softwarefeatures, um den Energieverbrauch zu minimieren. Deswegen ist diese Familie eine gute Lösung für die große Produktionsserien, low-power und kostenorientierte Anwendungen. Um das Design an eigene Bedürfnisse anzupassen, bietet die Cyclone III Familie folgende Möglichkeiten:

- Cyclone III: niedriger Energieverbrauch, höhe Funktionalität mit niedrigen Kosten.
- Cyclone III LS: Sichere FPGAs mit niedrigem Energieverbrauch.

Mit den Cyclone III LS Geräten können sicherheitsgerichteten Anwendungen am Silicon implementiert werden. Sie schützen die IPs vor einer Manipulation, revereser Engineering und illegalem Kopieren. Anschließend, können redundanten Teile auf dem Chip integriert werden, um die Größe der Anwendung zu reduzieren.

Tabelle 2.1: Eigenschaften der Cyclone III Familie

Familie	Gerät	Logische Elemente	Anzahl der M9K Blöcke	Total RAM Bits	18x18 Multiplizierer	PLLs	Globale Clk-Netze	Maximum Nutzer I/Os
Cyclone III	EP3C5	5,136	46	423,936	23	2	10	182
	EP3C10	10,320	46	423,936	23	2	10	182
	EP3C16	15,408	56	516,096	56	4	20	346
	EP3C25	24,624	66	608,256	66	4	20	215
	EP3C40	39,600	126	1,161,216	126	4	20	535
	EP3C55	55,856	260	2,396,160	156	4	20	377
	EP3C80	81,264	305	2,810,880	244	4	20	429
	EP3C120	119,088	432	3,981,312	288	4	20	531

Tabelle 2.2: Eigenschaften der Cyclone III (LS) Familie

Familie	Gerät	Logische Elemente	Anzahl der M9K Blöcke	Total RAM Bits	18x18 Multi-pliers	PLL	Globa-le Clk-Netze	Maximum Nutzer I/Os
Cyclone III LS	EP3CLS80	70,208	333	3,068,928	200	4	20	413
	EP3CLS100	100,448	483	4,451,328	276	4	20	413
	EP3CLS150	150,848	666	6,137,856	320	4	20	413
	EP3CLS200	198,464	891	8,211,456	396	4	20	413

Die Cyclone III (LS) Familie stellt dem Entwickler ein Featureset zur Verfügung, das für portable Anwendungen optimiert ist, und bietet eine Reihe verschiedener Speicherelemente, eingebetteter Multiplizierer und I/O-Optionen an. Sie unterstützt zahlreiche externe Speicherschnittstellen und Eingabe/Ausgabe-Protokolle, die in großen Produktionsserien üblich sind.

2.3.1 Logik-Elemente und Logik-Feld-Blöcke (Logik Array Blocks LAB)

Logische Elemente (LE) sind die kleinsten Einheiten der Cyclone III Architektur. Sie sind kompakt gebaut und ermöglichen komplexere Features mit einem effizienten Logikgebrauch. Jedes LE hat folgende Eigenschaften:

- 4 Input Look-Up-Tabelle, die jede Funktion mit vier Eingängen implementieren kann.
- programmierbare Register
- Carry-Chain Verbindung
- Register-Chain Verbindung
- verschiedene Interverbindungsmöglichkeiten

Jedes programmierbare Register kann als D, T, JK oder RS Flipflop konfiguriert werden. Jedes hat folgende Eingänge: Daten, Clock, Clock-Enable und Clear. Alle Signale, die den globalen Clock nutzen, die General-Purpose I/O Pins sowie die jede interne Logik, können den Clock treiben und die Kontrollsignale des Registers löschen.
Bei den kombinatorischen Funktionen umgeht der LUT-Ausgang die Register und geht direkt zu den LUT-Ausgängen.
Das LE funktioniert auf zwei Arten - im normalen oder im arithmetischen Modus. Der normale Modus ist für logische und kombinatorische Funktionen geeignet, der arithmetische Modus für arithmetische Funktionen (Addierer, Zähler, Akkumulator, Komparator…).

Logik-Feld-Blöcke (Logik Array Blocks LAB) sind im Prinzip gruppierte LEs. Sie besitzen folgende Eigenschaften:

- 16 LEs
- LAB Kontrollsignale
- LE Carry-Chain
- Register-Chain
- Lokale Interverbindung

Lokale Verbindungen transferieren Signale zwischen den LEs innerhalb eines LABs.

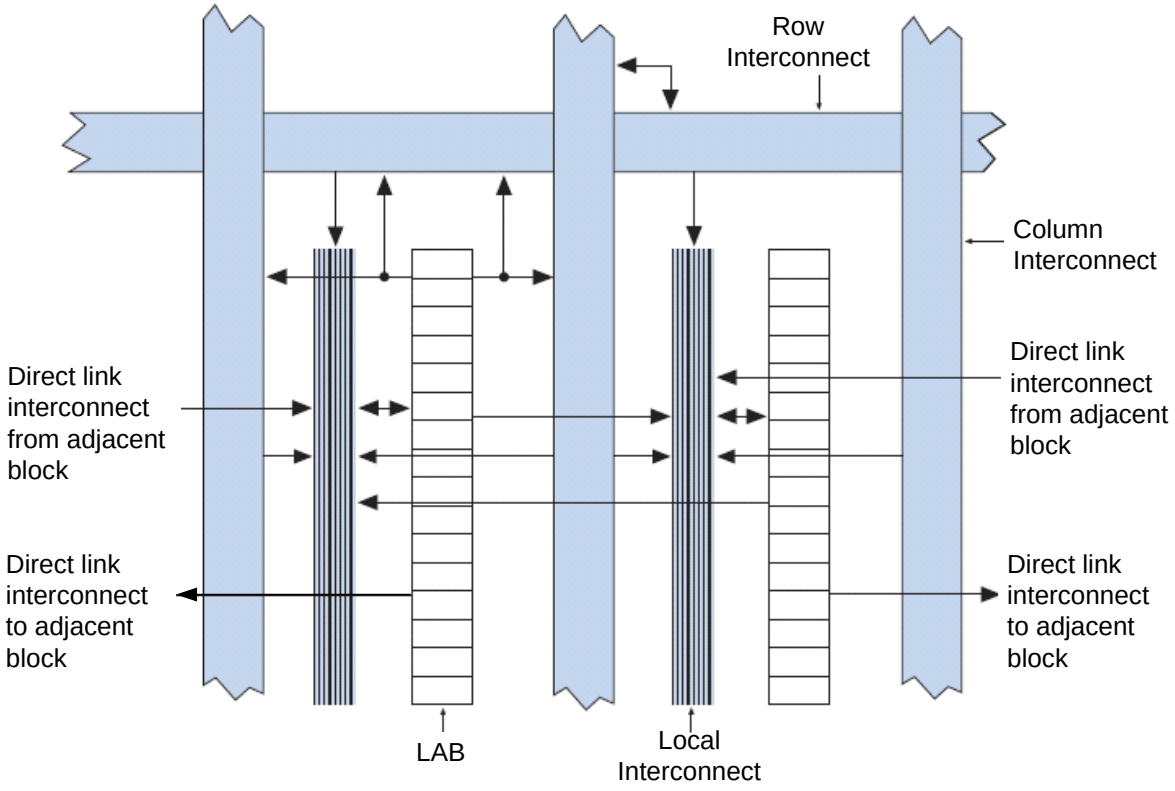

Bild 2-4: Die Struktur eines Cyclone III LABs

(Für mehr Informationen über LE bzw. LAB siehe die [2])

2.3.2 Speicherblöcke

Von der Cyclone III Device Familie werden die eingebetteten Speicherstrukturen angeboten, um die Bedürfnisse nach dem On-Chip Speicher zu erfüllen. Diese eingebetteten Speicherstrukturen bestehen aus den Spalten der M9K Speicherblöcken, die so konfiguriert werden können, um die verschiedenen Speicherfunktionen wie z. B. RAM, ROM, Schiebregister oder FIFO Puffer zu implementieren. Ein M9K (die Bedeutung dieser Abkürzung konnte nicht gefunden werden) Speicherblock ist ein synchroner Dual-Port Speicherblock.

M9K Speicherblöcke charakterisiert folgendes:

- 8 192 Speicherbits pro Block (9,216 Bits pro Block einschließlich die Paritätsbits)
- unabhängiges Read/Write-Enable Signal für jeden Port
- variable Portkonfigurationen
- Speichervariante, wo ein M9K Block in zwei Single-Port RAMs zerlegt wird
- Single-Port-Variante und einfache Dual-Port-Variante für alle Portbreiten
- richtige Dual-Port-Funktionalität
- Byte-Enable Signal für Maskierung des Dateneingangs während des Schreibens
- Clock-Enable Kontrollsignal für jeden Port
- Initialisierungsdatei, um den Speicherinhalt im ROM oder RAM vorzuladen

(Weitere Informationen über Speicherblöcke der Cyclone III Familie können in [3] gefunden werden)

2.3.3 Eingebetteter Multiplizierer

Die eingebetteten Multiplizierer werden als ein 18 x 18 oder als zwei 9 x 9 Multiplizierer konfiguriert. Es gibt keine Beschränkungen an der Datenbreite, aber es muss berücksichtigt werden, dass größere Datenbreite (ab 18x18) den Multiplikationsprozess verlangsamen.
Bei der Cyclone III Familie ist es möglich den „soft“ Multiplizierer mit den M9K Speicherblöcken zu implementieren.

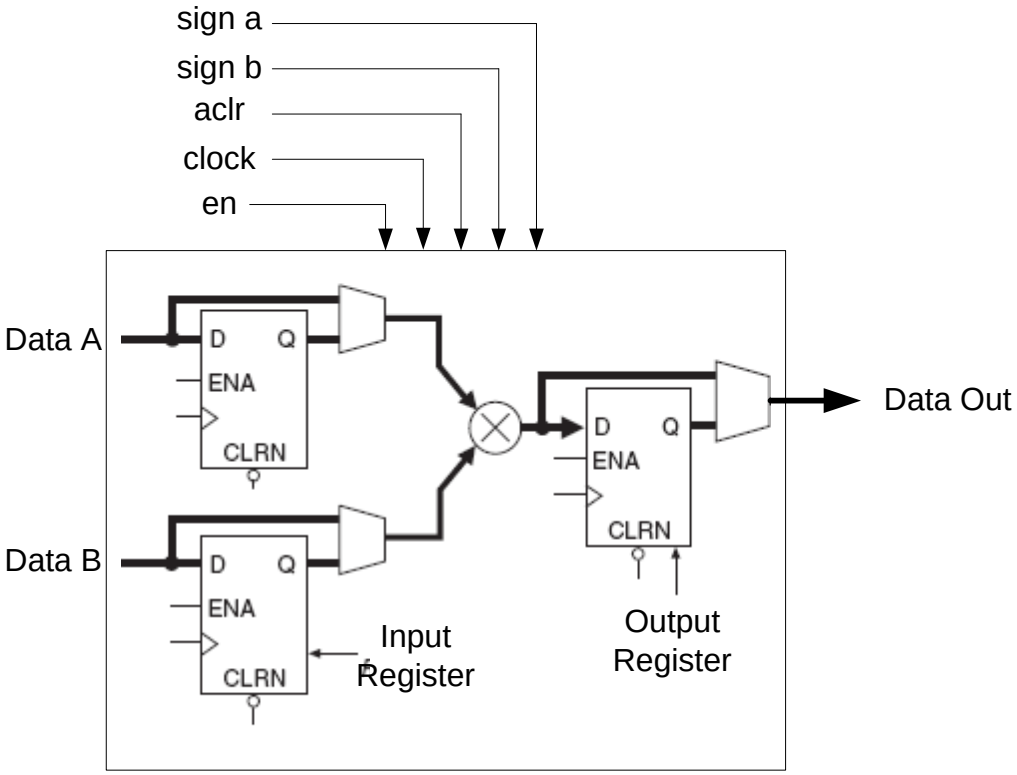

Bild 2-5: Eingebetteter Multiplizierer der Cyclone III Familie

(für weitere Informationen über eingebetteter Multiplizierer siehe [4])

2.3.4 Clock-Netze und PLLs

Die Cyclone III Familie stellt 16 Clockpins zur Verfügung, welche die globale Clocks (GCLKs) treiben können. Sie sieht für jede Seite des Gerätes vier Clockpins vor, außer für die EP3C5 und EP3C10 Gerättypen. Bei diesen zwei Gerättypen sind nur vier Clockpins auf der rechten und linken Seite des Gerätes vorgesehen.
Die globalen Clocks sind überall im Gerät erreichbar. Sie können auch von den Kontrollsignale sowie von der internen Logik verwendet werden. Die Clockkontrollblöcke treiben die GCLKs. Sie liegen an jeder Seite des Gerätes nahe zu den Clockpins. Die GCLKs sind für die minimale Asymmetrie und Verzögerung optimiert.
Der Kontrollblock hat zwei Funktionen:

- dynamisches Selektieren der GCLK-Quelle
- GCLK-Netze ausschalten

Der Cyclone III bietet auch bis zu vier PLLs (**P**hase **L**ocked-**L**oop) an. Diese stellen das robuste Clockmanagement, seine Synthese, externes Systemclockmanagement und die Hochleistungs-I/O-Schnittstellen zur Verfügung.
(Mehr Informationen über dieses Thema kann aus [5] entnommen werden)

2.3.5 I/O Schnittstellen

Die I/O-Elemente(IOE) der Cyclone III Familie enthalten den bidirektionalen I/O-Puffer und fünf Register für die Registrierung der Ein- und Ausgänge. Eine vollständige, eingebettete, bidirektionale Single-Daten-Übertragung wird auch gewährleistet. Diese fünf Register sind: ein Inputregister, zwei Outputregister und zwei Output-Enable-Register.
IOE bieten eine Reihe programmierbarer Funktionen für jeden I/O-Pin. Sie vergrößern die Flexibilität der I/O-Anwendung und stellen eine Alternative zur Verfügung, um den Gebrauch von den extern-getrennten Komponenten zu reduzieren.
Die Cyclone III Familie unterstützt vielfache single-ended und verschiedene differentiale I/O-Standards. Neben den 3.3-, 3.0-, 2.5-, 1.8-, 1.5 V I/O-Standards werden auch die 1.2 V I/O-Standards unterstützt.

Alle I/O-Pins sind zusammen in I/O-Bänke gruppiert. Jeder I/O-Pin gehört zu genau einer Bank. Es gibt 8 solche Bänke. Jede Bank hat einen VREF-Bus, um die Spannung der verschiedenen I/O-Standards anzupassen. Jeder VREF-Pin bezieht sich auf seine VREF-Gruppe.

Die Cyclon III Familie kann die Daten über LVDS-Signale senden bzw. empfangen. Die Input- und Output-Pins unterstützen für die LVDS-Transmitter und Empfänger die Seriallisierung und Deseriallisierung durch die interne Logik.
Die BLVDS (**B**us **L**ow **V**oltage **D**ifferential **S**ignaling) erweitern die Vorteile von LVDS um Multipoint-Anwendungen.
Die RSDS(**R**educed **S**wing **D**ifferential **S**ignaling) und mini-LVDS Standards sind die Ableitungen des LVDS-Standards. Prinzipiell sind diese zwei Standards im elektrischen Sinn sehr ähnlich zum LVDS-Standard, aber sie haben kleinere Spannungsschwänkung und bieten dadurch Möglichkeit der Energiesparrung.
Der PPDS (**P**oint-to-**P**oint **D**ifferential **S**ignaling)-Standard ist die nächste Generation des RSDS-Standards. Dieser wurde von der Firma National Semiconductor Corporation eingeführt.

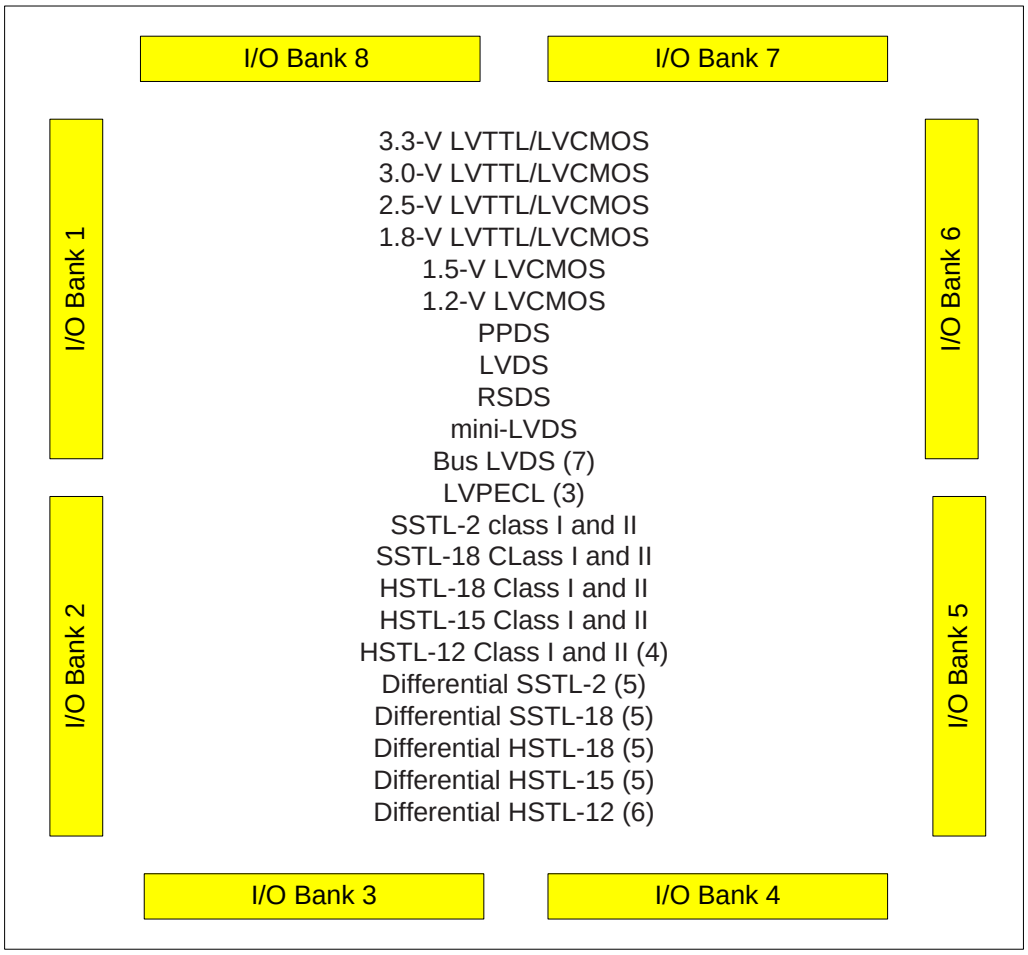

Bild 2-6: I/O Banken der Cyclone III Familie

Der Cyclon III unterstützt auch die I/O-Standards, die erforderlich sind, um mit einer breiten Reihe der externen Speicherschnittstellen (DDR SDRAM, DDR2 SDRAM und QDRII SRAM) zu kommunizieren.
(Für mehr Informationen über I/O-Schnittstellen siehe [6])

3 Softcore Prozessor Nios II

Nios II ist ein eingebetteter 32-bit, RISC (Reduced Instruction Set Computer), Softcore Prozessor der Firma Altera.

Mit dem Begriff Softcore-Prozessor werden zur heutigen Zeit Prozessoren bezeichnet, die in einer Hardware-Beschreibungssprache wie VHDL oder Verilog modelliert und für den Einsatz in programmierbaren Logikbausteinen vom Typ FPGA oder CPLD vorgesehen sind. Solche Prozessoren lassen sich vollständig (einschließlich der Daten- und Programmspeicher) in einem FPGA/CPLD integrieren. Als Gegenbegriff zu Soft-Core-Prozessor wird heute auch der Begriff Hard-Core-Prozessor verwendet, mit dem man Prozessoren bezeichnet, die in einem programmierbaren Logikbaustein bereits als Hardware-Module eingebettet sind, wie z. B. der PowerPC in dem FPGA-Baustein Virtex-4 FX von Xilinx.

Vom Nios II gibt es heute drei Varianten: Nios II/e (economy), Nios II/s (standard) und Nios II/f (fast).
Fast-Core ist optimiert auf System Performance, Economy-Core wurde auf kleinstmöglichen Logikbedarf optimiert, und ein Standard-Core ist die Mischung aus den ersten zwei Cores – ausgewogene Performance + geringer Logikbedarf.

Die Architektur eines Nios II Prozessor besteht aus folgenden Einheiten:

- Registerbank
- Arithmetic Logic Unit (ALU)
- Schnittstelle zur Befehlslogik
- Exception Controller
- Internem und externem Interruptcontroller
- Befehlsbus
- Datenbus
- Memory Management Unit (MMU)
- Memory Protection Unit (MPU)
- Befehls- und Datencachespeicher
- JTAG Debug Modul
- „eng gekoppeltem Speicher“ (tightly coupled memory)

Diese Architektur wird im Bild 3-1 verdeutlicht.

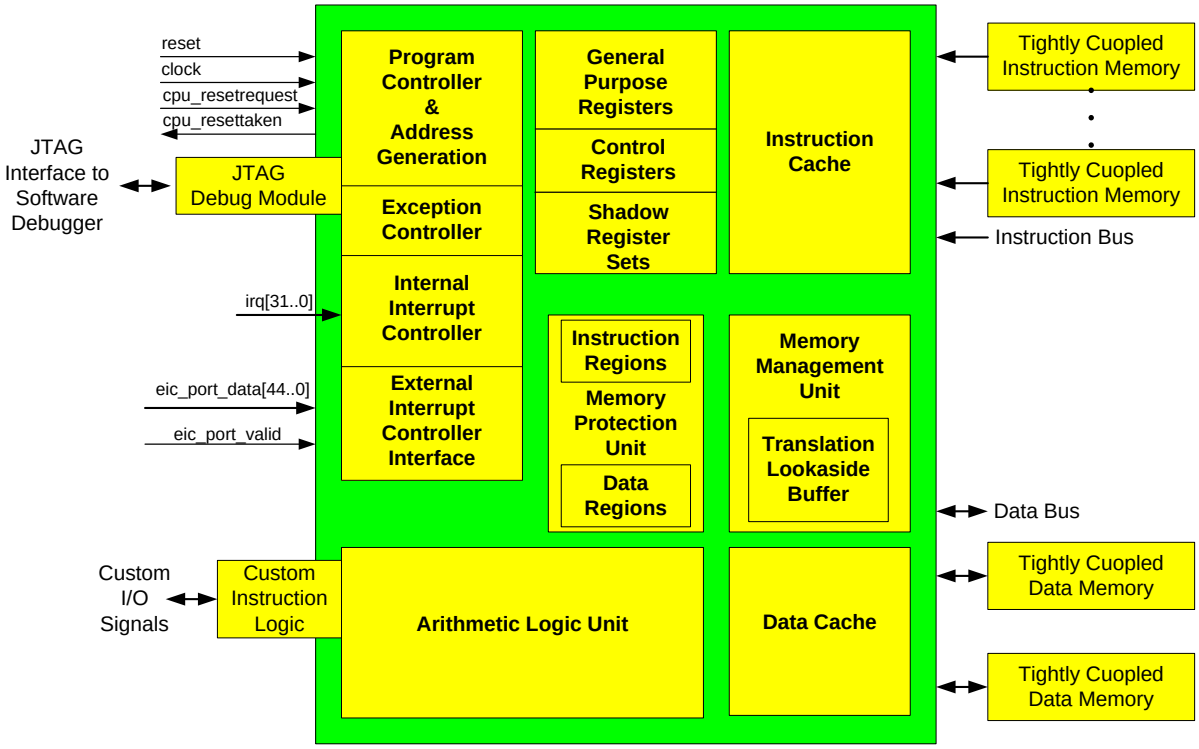

Bild 3-1: Blockdiagramm vom Nios II Prozessor Core

3.1 Registerbank

Die Nios II Architektur unterstützt eine flache Registerbank, die aus zweiunddreißig 32-Bit General-Purpose Register und bis zu zweiunddreißig 32-Bit Kontrollregister besteht. Es werden zwei Modi unterstützt – Supervisor- und Usermodus, die dem Systemcode ermöglichen, die Kontrollregister von den möglichen Fehlern schützen zu können. (weitere Informationen sind unter [7] verfügbar)

3.2 Nios II ALU

Die Nios II ALU bearbeitet die Daten, die in den General Purpose Registern (GPR) gespeichert sind. ALU Funktionen können auf einem oder zwei Register ausgeführt

werden. Das erhaltende Ergebnis dieser Funktion wird auch auf im Register gespeichert.
Die von Nios II ALU unterstützten Funktionen werden in der folgenden Tabelle dargestellt:

Tabelle 3.1: Funktionen der Nios II ALU

Funktionskategorie	Funktionen
Arithmetik	Addieren, Subtrahieren, Multiplizieren und Dividieren
Relational	==, !=, >=, < (gleich, ungleich, größer gleich und kleiner)
Logik	AND, OR, NOR und XOR
Shift und Rotation	Shift und Rotation vom 0 bis 31 Bits. Arithmetisch shift rechts, logisch shift rechts und links, Rotation links und rechts

Es besteht auch die Möglichkeit benutzerdefinierte Funktionen zu implementieren. Diese entstehen als Kombination der fundamentalen Funktionen aus der Tabelle 3.1 (für mehr Details über Nios II ALU siehe [7]).

3.3 Exception Controller

Der Nios II Prozessor unterstützt einen einfachen Exception Controller, der alle Arten der Ausnahmesituationen steuert. Jede Ausnahmesituation, einschließlich der Hardwareinterrupts, zwingt den Prozessor den aktuell ausgeführten Code auf einer Ausnahmeadresse zu übertragen. Auf dieser Adresse analysiert ein Exception Handler die Ursache der Ausnahmesituation und schickt eine passende Ausnahmeroutine zurück zum Prozessor. Die Ausnahmeadressen werden in SOPC Builder während der Systemgenerierungszeit spezifiziert.
Alle Exceptions sind ganz präzise festgelegt. In diesem Fall bedeutet präzise, dass der Prozessor alle Instruktionen vor der fehlerhaften Instruktion vollständig ausgeführt hat, und dass er zu diesem Zeitpunkt weitere (fehlerhaftere Instruktionen) nicht ausführt, sondern auf die Antwort vom Exception Handler wartet, und dementsprechend den weiteren Ablauf der Instruktionen definiert. (weitere Informationen sind in [7] zu finden)

3.4 Der Interruptcontroller

Die Nios II Architektur unterstützt 32 interne Hardware-Interrupts. Der Prozessorkern hat 32 Level-sensitive Interruptquellen (IRQ) irq0 bis irq31, und stellt für jede Interruptquelle einen getrennten Input bereit. Die IRQ Priorität kann auch durch die Software bestimmt werden.

Die Software kann durch die *Ienable Controll Register* jede Interruptquelle aktivieren bzw. deaktivieren. Ienable Controll Register enthält ein Interrupt-Enable-Bit für jeden IRQ Input. Das Aktivieren bzw. Deaktivieren der Interrupts durch die Software kann auch global passieren, und zwar durch das Setzen des PIE Bits vom *Status Controll Register*. (weitere Informationen sind aus [7] zu entnehmen)

3.5 Die Speicher- und I/O-Organisation

Charakteristisch für Speicher- und I/O-Organisation vom Nios II Prozessor ist ihre Flexibilität. Dies ist ein gewaltiger Unterschied zwischen Nios II und traditionellen Mikrocontroller. Da die Nios II Systeme konfigurierbar sind, variieren die Speichermodelle und Peripherien von System zu System. Das bedeutet auch, dass die Speicher- und I/O-Organisation von System zu System unterschiedlich sind.
Der Zugriff auf Speicher und I/O wird auf die folgenden Arten realisiert:

- Befehlsmaster Port – ein Avalon Memory-Mapped (Avalon-MM) Master Port, der die Verbindung zum Befehlsspeicher via System-Interconnect-Fabric stellt (mehr über dieses Thema im Abschnitt „Avalon Bus“).
- Befehlscache – Schneller, interner Cachespeicher
- Datenmaster Port – ein Avalon-MM Master Port, der die Verbindung zum Datenspeicher via System-Interconnect-Fabric stellt.
- Datencache – Schneller, interner Cachespeicher
- „eng gekoppelter Speicher“ (tightly coupled memory) – Schnittstelle zum schnellen externen on-chip-Speicher

Die Nios II Architektur abstrahiert die Hardwaredetails vom Programmierer, so dass dieser eine Nios II Applikation ohne die Kenntnis über die Hardwareimplementierung schreiben kann.
Das Bild 3-2 sollte die Speicher- und I/O Organisation vom Nios II verdeutlichen.

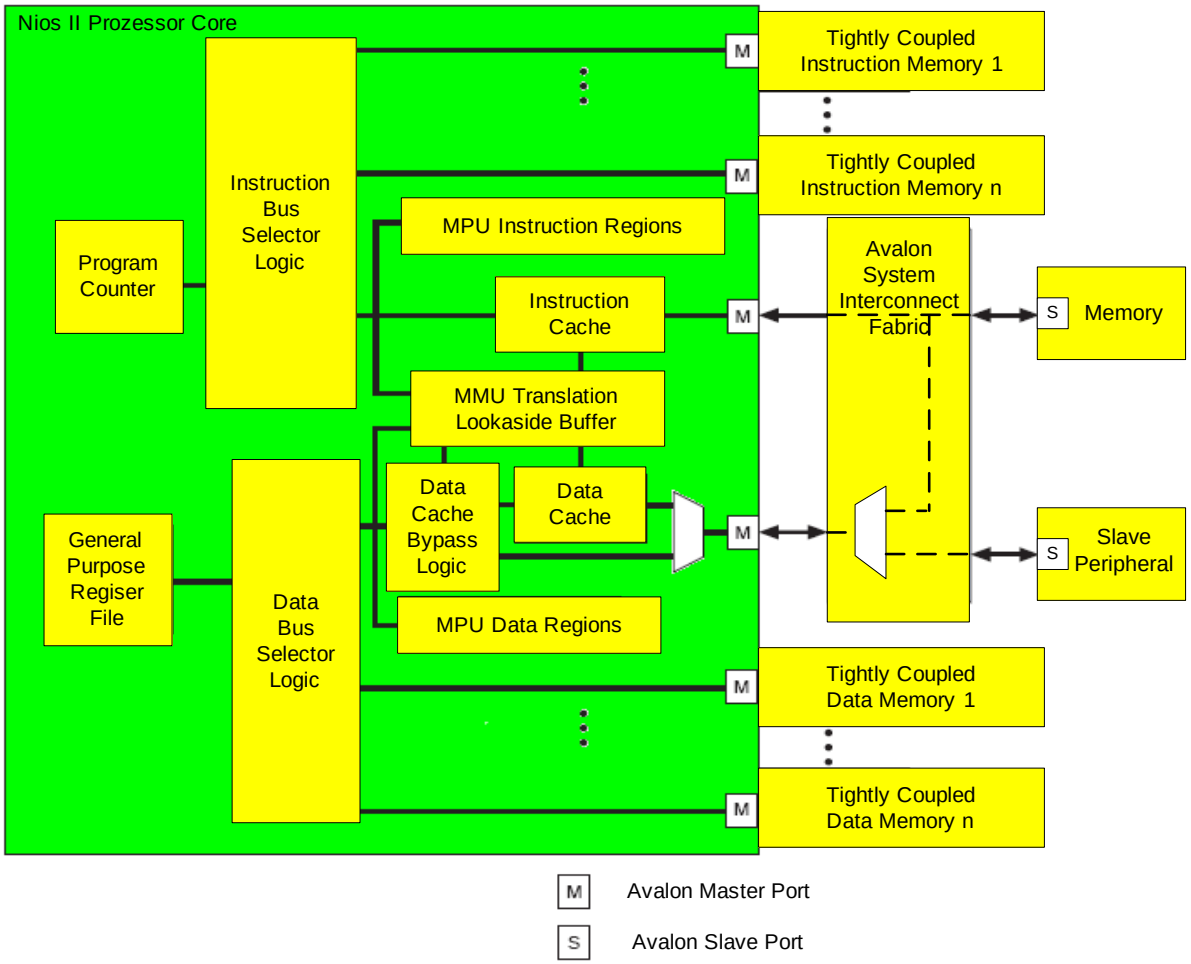

Bild 3-2: Nios II Speicher- und I/O-Organisation

3.6 Befehls- und Datenbus

Nios II Prozessor besitzt getrennte Befehls- und Datenbus, wodurch es zur Harward-Prozessor-Architektur gehört. Beiden Buse sind als Avalon-MM Masterport implementiert. Der Datenbus stellt die Verbindung zu den Speicherelementen und zu den Peripheriekomponenten her. Der Befehlsbus stellt die Verbindung nur zu den Speicherelementen her.

Der Nios II Befehlsbus ist als ein 32-Bit Avalon-MM Masterport implementiert. Er realisiert nur eine Funktion: Er holt die Anweisungen, die vom Prozessor ausgeführt werden sollen. (Anmerkung: Befehlsmasterport führt keine Schreib-Operation aus)

Der Befehlsmasterport ist ein „pipelined“ Avalon-MM Masterport. Der Befehlsmasterport kann aufeinander folgende Lese-Anträge herausgeben, bevor die Daten von den vorherigen Anträgen zurückgegeben sind.
Der Befehlsbus bekommt immer eine Datenmenge von 32 Bit. Befehlsmasterport beruht auf der dynamischen Bus-Kalibrierung-Logik. Aufgrund der dynamischen Bus-Kalibrierung-Logik gibt jede Befehlsanfrage ein volles Anweisungswort zurück, unabhängig von der Breite des Zielspeichers. Infolgedessen, brauchen die Programme Breite des Speichers im Nios II Prozessorsystem nicht zu berücksichtigen.

Der Nios II Datenbus ist auch als 32-Bit Avalon-MM Masterport implementiert. Er realisiert zwei Funktionen:

- Lesen der Daten aus dem Speicher oder einer Peripherie, wenn der Prozessor die „Load“ Operation ausführt.
- Schreiben der Daten in dem Speicher oder in einer Peripherie, wenn der Prozessor die „Store“ Operation ausführt.

Byte-enable Signale auf dem Datenmasterport entscheiden, welche von den vier Bytes während der Ausführung der „Store“ Operation geschrieben werden. Wenn der Nios II Core mit dem Datencachespeicher konfiguriert ist, dessen Datenbreite größer als vier Bytes ist, dann findet die „pipelined“ Avalon-MM Übertragung statt.
Falls der Datenmasterport mit dem zero-wait-state-Speicher verbunden ist, können die „Load“ und „Store“ Operationen in einem einzigen Clockzyklus vollständig abgeschlossen werden.
(weitere Informationen sind in [7] zu finden)

3.7 Cachespeicher

Die Nios II Architektur unterstützt den Cachespeicher auf dem Befehlsmasterport (Befehlscache) und auf dem Datenmasterport (Datencache).
Der Cachespeicher ist der integrierte Teil des Nios II Prozessor Cores, und kann als On-Chip-Speicher betrachtet werden. Bei den Nios II Prozessorsystemen, die einen

langsamen Off-Chip-Speicher (wie z. B SDRAM) für die Programm-, bzw. Datenspeicherung nutzen, kann der Cachespeicher die durchschnittliche Speicherzugriffszeit verbessern.

Die Befehls- und Datencachespeicher sind während der Ausführung des Programms dauerhaft benutzbar. Der Nios II Befehlssatz definiert auch die Befehle für das Cachemanagement.

Die Cachespeicher sind optional, sie müssen nicht unbedingt eingesetzt werden. Der Bedarf nach hoher Speicherperformanz ist von Applikation zu Applikation verschieden. Es gibt viele Applikationen, die möglichst kleine Prozessorkonfiguration anfordern, so dass der Einsatz vom Cachespeicher nicht so viel Sinn macht.

Ein Nios II Prozessor kann entweder einen, zwei oder keinen der Cachespeicher beinhalten. Außerdem ist für die Prozessoren, die einen Cachespeicher einsetzen, die Speichergröße frei konfigurierbar. Der Einsatz dieser Art des Speichers hat keinen Einfluss auf die Funktionalität des Programms, doch es beeinflusst die Prozessorperformanz, da der Zugriff auf die Befehle effizienter ist. Schreiben und Lesen der Daten erfolgt demnach schneller.
(für mehr Informationen siehe [7])

3.8 Eng gekoppelter Speicher - Tightly Coupled Memory

Tightly Coupled Memory (TCM) ermöglicht einen “low-latency” Speicherzugriff für performanzkritische Applikationen. Die wichtigsten Eigenschaften dieses Speichers sind:

- Performanz ist sehr ähnlich zum Cache
- Software kann garantieren, dass der performanzkritischer Code oder Daten in TCM abgelegt sind.
- Kein Echtzeit-Caching Zusatz wie z. B. Speicher-Leerung.

Physisch ist der TCM-Port ein getrennter Masterport des Nios II Prozessorcores, der zum Befehls-, oder Datenmasterport ähnlich ist. In der Nios II Architektur sind keine, eine oder mehrere TCM-s einzusetzen, und können für den Zugriff auf die Daten und Befehle benutzt werden. Jeder stellt eine direkte Verbindung zu genau einem Speicher her, und garantiert dabei eine kleine, gleichbleibende Latenz.

TCM verfügt über klar definiertem Adressbereich, der zur Systemgenerierungszeit erzeugt und zugeordnet wird. Sie benutzt normale “Load” und “Store” Befehle. Aus der Softwaresicht gibt es kein Unterschied zwischen TCM und anderen Speicher. (für mehr Informationen siehe [7])

3.9 Zuordnung der Adressen

Die Zuordnung der Adressen für die Speicher und Peripherie im Nios II Prozessorsystem hängt von dem Design ab. Die Adresszuordnung wird während der Systemgenerierungszeit definiert. Man unterscheidet zwischen drei Adressarten, die für den Nios II Prozessor von großer Bedeutung sind:

- Reset-Adressen
- Exception-Adressen
- Break-Handler-Adressen

Die Programmierer greifen auf den Speicher und Peripherie zu, indem sie die verschiedenen Makros und Treiber nutzen. Folglich, hat die flexible Art der Adresszuordnung keinen Einfluss auf die Applikation des Entwicklers.

3.10 Memory Management Unit und Memory Protection Unit

Eine ***Memory Management Unit (MMU)*** wird benutzt, wenn ein Betriebssystem zum Einsatz kommt. Die MMU löst folgende Aufgaben:

- Übersetzt die virtuellen Adressen zu physischen
- Speicherschutz
- Cachekontrolle
- Adressverteilung durch die Software

Einsatz der MMU bei dem Nios II Core ist optional. Es wurde schon erwähnt, dass der Einsatz nur dann sinnvoll ist, wenn ein Betriebssystem auf dem Nios II Prozessor vorhanden ist.
Bei den Systemen, die ganz einfach aufgebaut sind, reicht die Nutzung der Hardware-Abstraction-Library (HAL) vollständig aus. Diese Bibliotheken ermöglichen direkten Zugriff auf die Hardwarekomponenten.

Der Nios II Prozessor ermöglicht den Einsatz von einer ***Memory Protection Unit (MPU)*** für Betriebssysteme und Laufzeitumgebungen, die den Speicherschutz anstatt des virtuellen Speichermanagements bevorzugt.
Wenn die MPU auf einem System vorhanden und eingeschaltet ist, überwacht sie das Holen der Anweisungen, den Zugriff auf die Daten und Befehlen und schützt somit gegen unerwünschter Programmausführung.
Die MPU kann als ein Hardwarezusatz betrachtet werden, der von der Software zur Definition der Adressbereiche und der Art des Zugriffes benutzt wird.
Diese Einheit ermöglicht dem Nios II Prozessor zwei verschieden Modus, nämlich den User-Modus und den Supervisor-Modus. Normalerweise, läuft die Systemsoftware im Supervisor-Modus und die Endapplikation des Benutzers läuft im User-Modus, obwohl jede Software im Supervisor-Modus laufen könnte, falls dies erwünscht wäre.
Die Systemsoftware definiert, welche MPU-Bereiche zum Supervisor-Modus und welche zum User-Modus gehören.

Speicherbereiche: Die MPU besteht aus bis zum 32 Befehlsspeicher-, und 32 Datenspeicherbereiche. Jeder Bereich wird durch folgende Attribute gekennzeichnet:

- **Basisadresse:** spezifiziert die kleinste Adresse eines Speicherbereiches. Diese Adresse wird durch die Speicherbereichsgröße bestimmt (z. B. wenn die Größe eines Speicherbereiches 4Kbyte beträgt, dann muss die Basisadresse mit 4Kbyte multipliziert werden). Wenn diese Adresse nicht korrekt festgelegt ist, dann ist der Umgang mit dem Speicherbereich undefiniert.
- **Bereichstyp**: jeder Speicherbereich der MPU ist entweder der Daten- oder Befehlsspeicherbereich.
- **Bereichsindex**: jeder Speicherbereich hat einen Index zwischen 0 und n. (n = Zahl der Speicherbereiche eines Bereichstypes. Index 0 hat die größte Priorität)
- **Bereichsgröße:** die minimale Größe eines Speicherbereiches beträgt 64 Bytes. Die maximale Größe eines Speicherbereiches ist gleich der Größe des Nios II Adressraumes.
- **Zugriffsrechte:** das Zugriffsrecht bezieht sich auf die Ausführungsrechte, wenn es sich um ein Befehlsspeicherbereich handelt, bzw. auf Schreib-

/Leserechte für die Datenspeicherbereiche. Jeder Zugriff ohne Rechte verursacht eine Ausnahmesituation.

(weitere Informationen über MMU und MPU sind aus [8] zu entnehmen)

3.11 JTAG Debug Modul

Nios II Architektur unterstützt ein JTAG Debug Modul für die on-Chip-Simulation, damit der Prozessor von dem PC gesteuert werden kann. Durch dieses Modul werden folgenden Aufgaben gelöst:

- Download des Programms
- Starten und Stoppen der Programmausführung
- Setzen der Breakpoints und watchpoints
- Analyse der Register und Speicher

Die JTAG Target Connection stellt die Verbindung zum Prozessor über JTAG Pins auf dem Altera FPGA her. Dies ermöglicht das Starten und Stoppen des Prozessors, sowie die Überprüfung und Änderung der Register und Speicher.
Es gibt bestimmte Software (Quartus II oder Nios II DIE) die den Download des ausführbaren Codes und der Daten auf dem Prozessorspeicher über die JTAG-Verbindung ermöglichen.
Softwarebreakpoints werden auf die Befehle gesetzt, die sich im RAM Speicher befinden.
Hardwarebreakpoints werden auf die Befehle gesetzt, die sich im Flash Speicher befinden. (für mehr Informationen siehe [7])

3.12 Avalon Bus

Der Avalon-Bus ist ein synchroner Bus und wurde von Altera speziell für den Datentransfer zwischen den Komponenten eines SOPC-Systems entwickelt. Es wird zwi-

schen Avalon-Master und Avalon-Slave-Ports unterschieden. Ein Master kann einen Bustransfer ohne Beteiligung der CPU initiieren und durchführen. Ein Slave-Port hingegen muss von einem Master angestoßen werden, um einen Bustransfer zu starten. Der Datenfluss geht immer nur in eine Richtung, deswegen gibt es für das Schreiben und Lesen getrennte Datenkanäle. So wird im FPGA intern ein bidirektionaler Bus mit Tri-State-Zuständen vermieden.

Man kann beliebig viele Avalon-Busse einführen, diese können zueinander ihre Daten parallel transferieren, sofern sie nicht zum exakt gleichen Zeitpunkt auf eine gleiche Ressource zugreifen. In diesem Fall wird über eine priorisierte Arbitration der Zugriff geregelt.

Auf dem Bild 3-3 initiiert der Master M1 einen Read-Bustransfer indem er die gewünschte Adresse „M1_address“ anlegt und das „M1_readn“ Signal auf logisch Null legt. Über den Avalon-Bus wird die Adresse an den Slave gelegt und mit dem „S1-chipselect“ der Lesezugriff gestartet. Die gelesenen Daten werden dann zum Master 1 übertragen, welcher sie mit der nächsten positiven Taktflanke übernimmt. In diesem Beispiel kann also pro Taktzyklus maximal ein 32Bit-Wort übertragen werden. Die Taktraten und Busbreiten sowie die Art der Steuersignale können über den SOPC Builder frei konfiguriert werden. Als Clock-Signal sollte am besten der CPU-Takt verwendet werden, es sei denn, die zu verbindende Komponente arbeitet mit einem anderen Taktsignal, dann ist eine Synchronisation von Nöten.

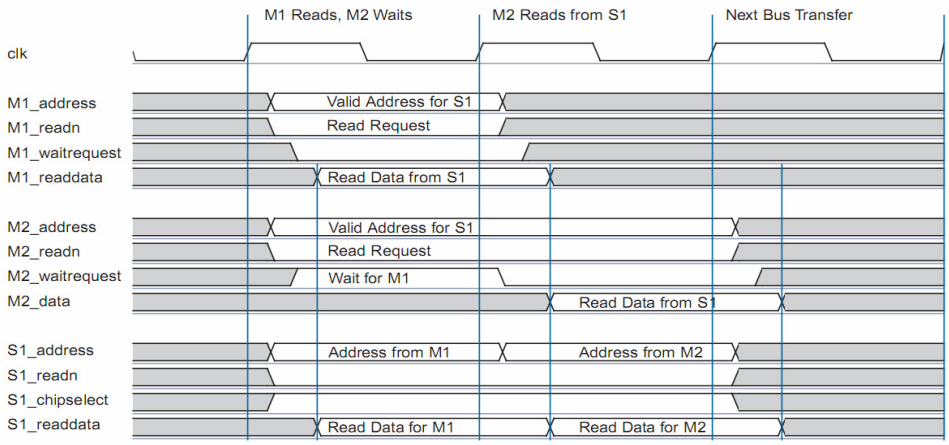

Bild 3-3: Zwei Avalon Master greifen auf einen Slave zu

Avalon® stellt eine Schnittelle dar, die die Portverbindungen zwischen Master- und Slavekomponenten und das Timing bei der Kommunikation dazwischen bestimmt. Die Schnittstelle erleichtert das Systemdesign, indem sie das Verbinden von Komponenten in FPGA vereinfacht. Es gibt sechs unterschiedlichen Schnittstellentypen:

- Avalon Memory Mapped Interface
- Avalon Memory Mapped Tristate Interface
- Avalon Streaming Interface
- Avalon Clock
- Avalon Interrupt
- Avalon Conduit

Zusätzliche Informationen über diese Schnittstellen können in [9] gefunden werden.

3.13 System-Interconnect Fabric

System-Interconnect-Fabric (SIB) ist eine Kommunikationsstruktur, die so implementiert ist, dass die Master-Slave Kommunikation in einem System immer bereitgestellt werden kann. SIB besteht aus einer synchronen Logik und Routingressourcen innerhalb des FPGA und handhabt die ganze Kommunikation im SoC.
Master- und Slaveschnittstellen können unterschiedliche Signale enthalten. Um miteinander kommunizieren zu können, muss SIB bestimmte Anpassungen an den Signalen vornehmen.
SIB realisiert folgende Funktionen:

- Adressdekodierung
- Datenpfad Multiplexing
- Wait-State-Insertion
- Gepippelte Leseübertragung
- Arbitrierung für Multimaster Systeme
- Interrupts
- Reseten des Systems

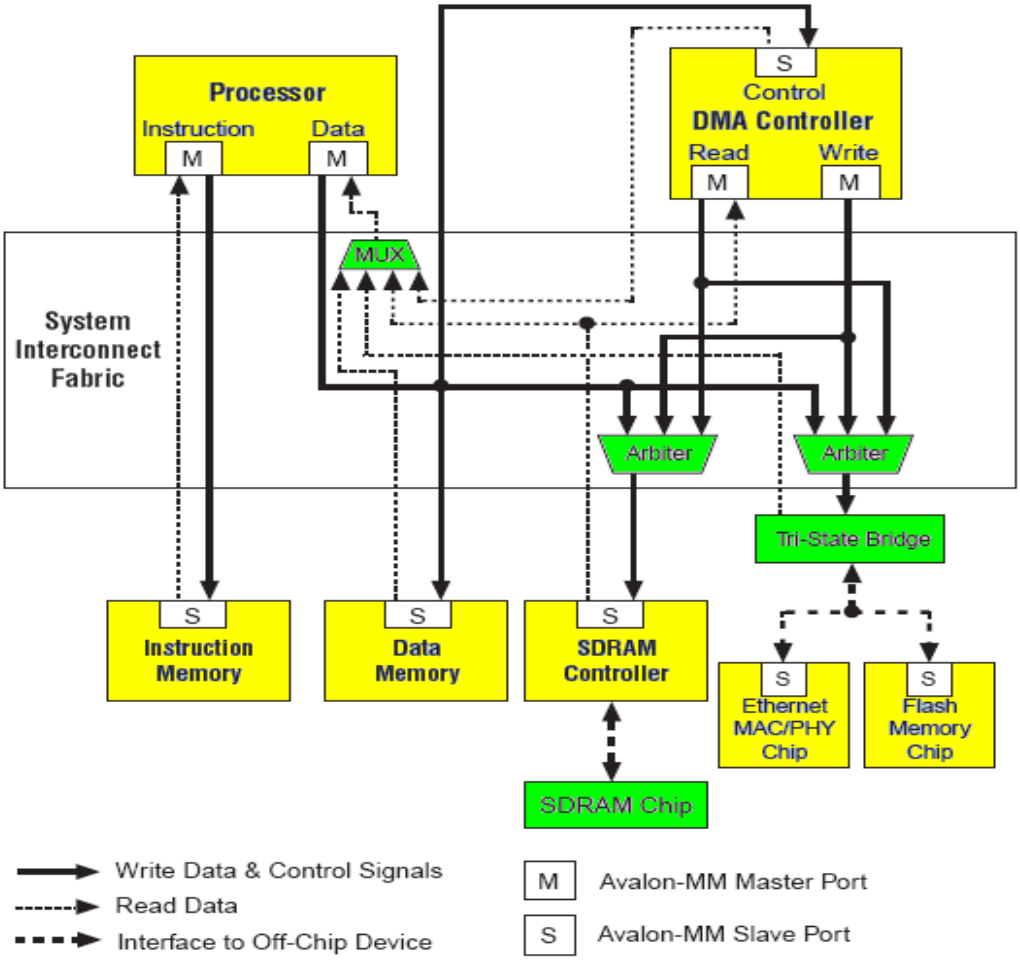

Bild 3-4: Beispielsystem einer System-Interconnect-Fabric

Weiteres über System-Interconnect-Fabric ist aus [10] zu entnehmen.

4 System-on-Chip Hardwaredesign

In dieser Arbeit wird das DBC3C40 Etwiklungsboard der Firma Altera benutzt. Das Hardwaredesign wurde mit der Alteras Software Quartus II 9.0 erstellt.
Bevor man mit dem Hardwaredesign anfängt, sollten erst einmal die Komponenten des Entwicklungsboards genannt und deren Funktionen kurz besprochen werden.

4.1 DBC3C40 Entwicklungsboard

Das DBC3C40 Board ist ein Cyclone III Development Board der FirmaEBV Elektronik mit einigen Schnittstellen für industrielle Kommunikationszwecke.

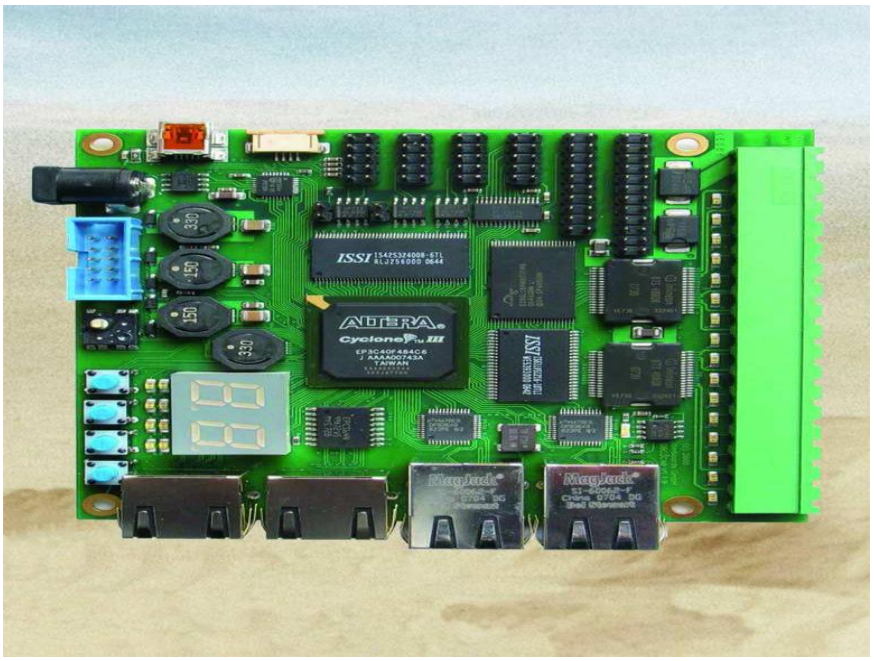

Bild 4-1: Das DBC3C40 Board

Die Komponenten dieses Boards sind im nachfolgenden Bild anzusehen:

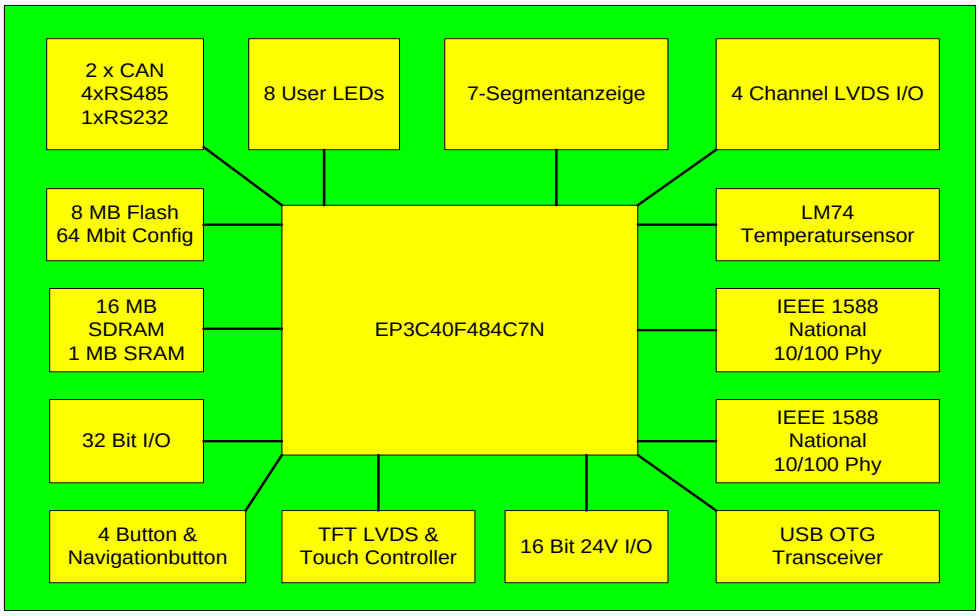

Bild 4-2: Blockdiagramm des DBC3C40 Boards

4.2 Hardwarekomponenten des DBC3C40 Boards

Auf dem Board befinden sich folgende Bauteile:

• EP3C40F484C7N - eine Cyclone III FPGA-Matrix von Altera

• EPCS64 Configuration Device - ein serieller Flashchip, in dem die Konfiguration des FPGAs gespeichert werden kann, um später beim Einschalten automatisch hochgeladen zu werden (siehe [11]).

• 2x DB83640 10/100 Ethernet PHY - Der PHY-Chip verbindet die Vermittlungsschicht mit der physikalischen Schicht (siehe [12]).

• 2x 4 Channel LVDS23 link on RJ45 sockets - eine Schnittstelle für Hochgeschwindigkeit-Datenübertragung

• USB OTG24 Transceiver - eine USB-Schnittstelle

• LVDS TFT Interface - ein TFT Bildschirm mit LVDS Interface kann angeschlossen werden

• 16Mbyte SDRAM - ein ISSI 128 Mbit SDRAM-Chip

• 1Mbyte SRAM - ein ISSI 512Kx16bit SRAM-Chip

• 8Mbyte Flash - ein 4Mx16bit Flash-Chip von Spansion

• LM74 I2C Temperature Sensor - ein Temperatursensor vom National Semiconductor mit SPI/Microwire kompatibler Schnittstelle

• 1x UART Transceiver - wird mit einem RS232 Transceiver von Texas Instruments MAX3238 benutzt

• 2x CAN Transceiver - zwei Texas Instruments SN65HVD233

• 4x RS485 Transceiver - vier RS458 Transceiver von Texas Instruments SN75HVD11D

• 32 pin I/O Connector - diese sind direkt mit dem FPGA verbunden

• 16bit 24V I/O Interface - ein Phoenix Contact Combicon Header. Jeder Combicon-Pin ist mit einem LED verbunden, das den Pegel des Pins anzeigt

• 8x User LEDs

• 2 Digit Seven Segment Display

• 4x User Buttons

• Navigation Key - kann für die graphischen Routinen für das TFT Display benutzt werden

• JTAG Interface - wird zusammen mit den Altera-Tools zur Programmierung Benutzt

• Touch Screen Controller - ein TSC2200 Touchscreen-Kontroller von Texas Instruments. Implementiert 4-Draht Touchschnittstelle, 2 ADC25 Kanäle und ein DAC26 Kanal für Steuerung der Hintergrundbeleuchtung des TFTs

• Security Eeprom - EEPROM27 AT24C16, wird für die Benutzung von EtherCat28 auf dem Board erfordert

• On Board 12V, 5V, 3.3V and 1.2V power supply

4.3 Hardwaredesign des Nios II Systems

Sobald ein Überblick über die in dieser Arbeit benutzte Hardware geschafft ist, lässt sich mit der Beschreibung des Hardwaredesigns anfangen.
Um ein Hardwaresystem mit den Bauteilen der Firma Altera zu entwickeln, muss man die Software Quartus II 9.0 benutzen. Diese Software ist von Alteras Homepage herunterzuladen.

Als zentrales Werkzeug dieser Software wird das SOPC Builder Tool bezeichnet.

Das zu entwickelnde System wird durch folgende Komponenten definiert:

- Nios II Prozessor
- JTAG UART
- UART RS232
- On-Chip-ROM
- 1 Mbyte SRAM
- 8 Mbyte Flash
- 16 Mbyte SDRAM
- LM74 I2C Temperatur Sensor
- EPCS64 Konfiguration Device
- DB83640 10/100 Ethernet PHY
- 8 User LEDS
- 4 User Buttons
- 7 Segmentanzeige
- System Identifikation Komponente
- 3,3V I/O für Kommunikation mit der Sensorik

Das folgende Bild soll die Phasen der Entwicklung verdeutlichen:

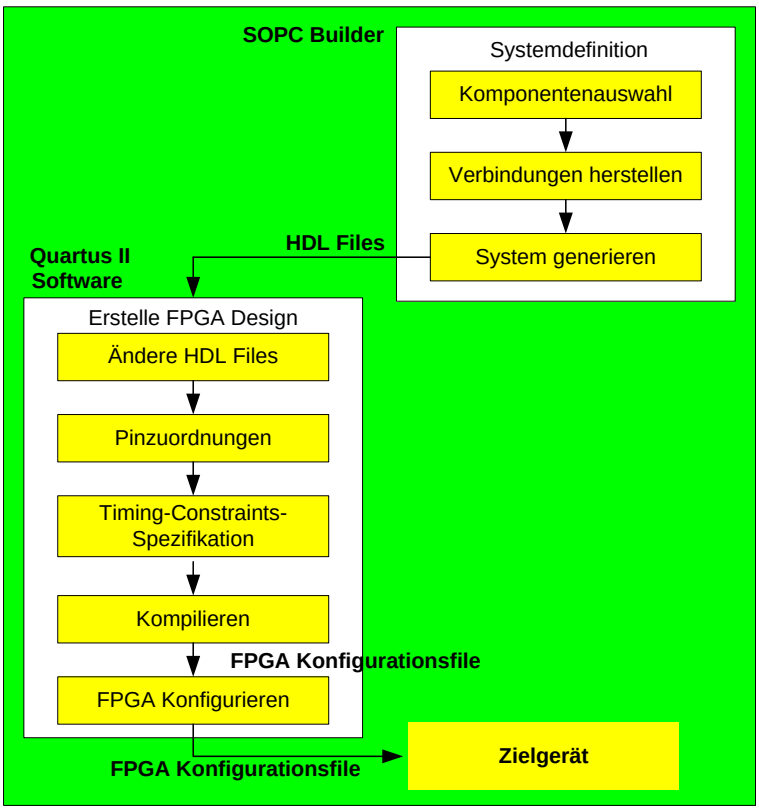

Bild 4-3: Hardwaredesignfluss

4.3.1 Phase 1 des Hardwaredesigns: SOPC Builder

Nachdem die Bestandteile des Systems definiert wurden, kann man mit dem SOPC Builder Werkzeug diese Bestandteile spezifizieren. Mit diesem Werkzeug werden also die Komponenten des zu entwickelnden Systems konfiguriert, und die Verbindungen zwischen den hergestellt. Zwei sehr wichtigen Eigenschaften von SOPC Builder sind Flexibilität und Kompatibilität. Man kann jeder Zeit das System anpassen – neue Komponenten einfügen, oder bestimmte Komponente löschen. Es besteht auch die Möglichkeit selbstdefinierte Hardware Komponenten einzusetzen, die in VHDL geschrieben sind. Die primären Ausgaben von SOPC Builder sind folgende Files:

- SOPC Builder Design File (**.sopc**) – enthält die Hardwarebestandteile des SOPC Builder Systems.

- SOPC Informationsfile (**.sopcinfo**) – für den Mensch lesbare Beschreibung der Komponenten vom .sopc File.

- Hardware Description Language (**HDL**) Files – dies sind die Hardwaredesignfiles, die das SOPC Builder System beschreiben.

(für mehr Informationen über SOPC Builder Werkzeug, siehe [13])

Designsschritte:

Schritt 1: Quartus II 9.0 starten – Start → Programme → Quartus 9.0 → Open
Die für DBC3C40 installierten Dateien sollen gefunden und nios_II_lab Ordner geöffnet werden. Wähle nios_II_lab.qpf (Projektfile) aus.
Man könnte auch ein ganz neues Projekt machen, wobei die erste Möglichkeit ein bisschen Arbeit spart, da die PLL Komponente schon vorhanden ist.

Schritt 2: SOPC Builder starten – Tools → SOPC Builder. Im Fenster Create new System muss der Name des Systems eingegeben werden. VHDL soll als Target HDL markiert werden. Auf OK klicken.

Schritt 3: Im Targetfenster als Device Family Cyclone III auswählen. Das folgende Bild zeigt die GUI vom SOPC Builder:

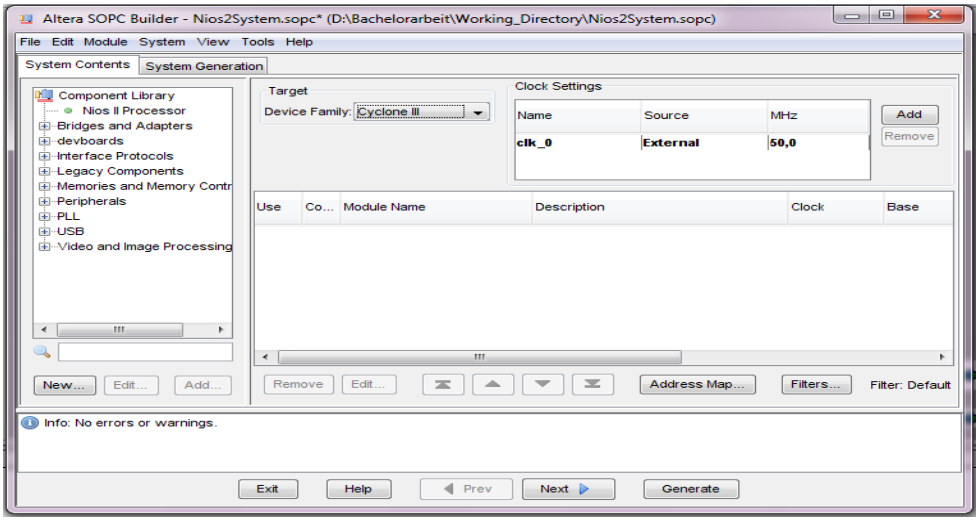

Bild 4-4: GUI von SOPC Builder

Schritt 4: Komponente Nios 2 Prozessor einfügen – Aus dem linken Fenster „System Contents“ wähle Nios II Prozessor aus, und dann Doppelklick.

Folgenden Einstellungen sind vorzunehmen:
Im Tab „Core Nios II“ wähle Nios II/f als Nios II Core aus, als „Hardware Multiply“ Embedded Multipliers, später wenn auch Speicherkomponenten eingefügt sind, wird als Memory für Reset und Exception Vector sdram ausgewählt.
Wähle im Tab Caches and Memory Interfaces für Instruction Master 4 Kbytes Instructioncache und für Daten Master 2 Kbytes Instructioncache. Daten Cache Line Size muss auf 32 Bytes gesetzt werden.
Include tightly coupled instruction (Daten) master muss markiert werden, und number of ports auf 1 gesetzt.

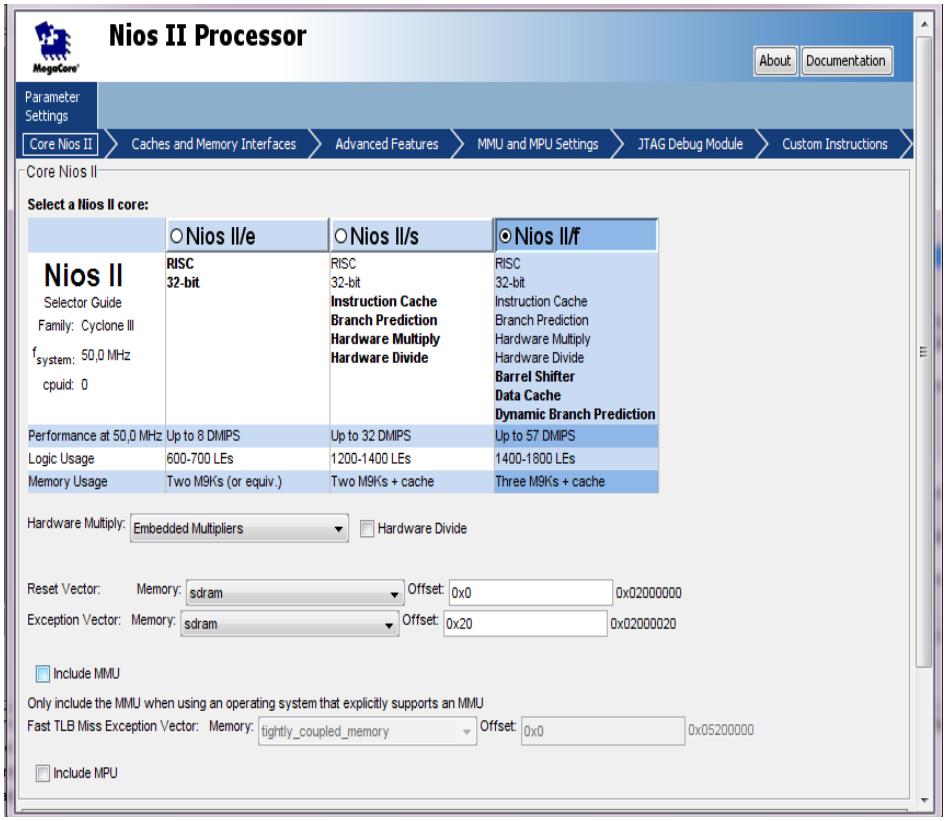

Bild 4-5: Generierung vom Nios II Prozessor

Im Tab JTAG Debug Module sollte Level 1 ausgewählt werden, die anderen Einstellungen sind als Default Einstellungen zu übernehmen.
Im Tab Custom Instructions soll nichts geändert werden. Auf finish klicken.

Schritt 5: Komponente SRAM einfügen - Aus dem Fenster System Contents wähle devboards → DBC3C40 → **IS61WV51216** aus. Dann auf finish klicken.

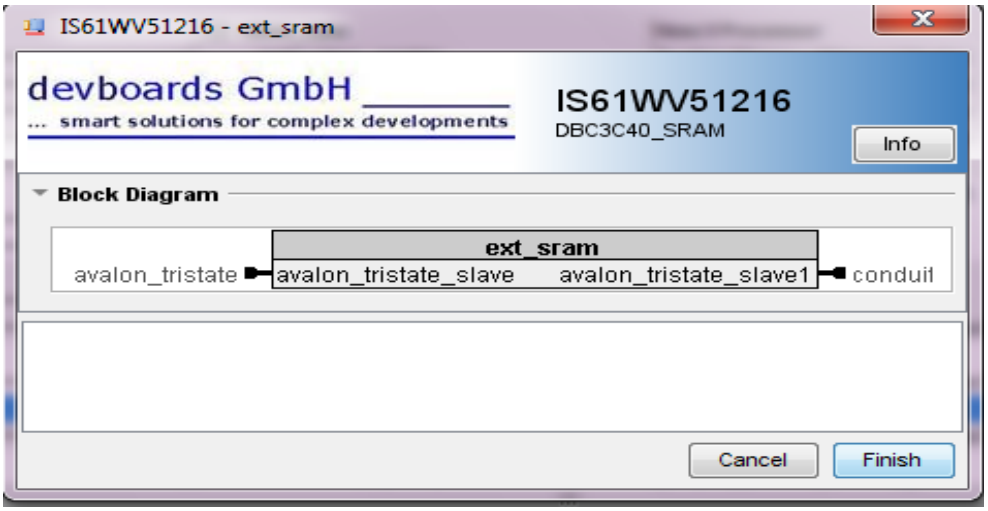

Bild 4-6: Generierung von SRAM

Schritt 6: Komponente Flash Speicher einfügen - Aus dem Fenster System Contents wähle Memories and Memory Controllers aus → Flash → Flash Memory Interface (CFI) aus.

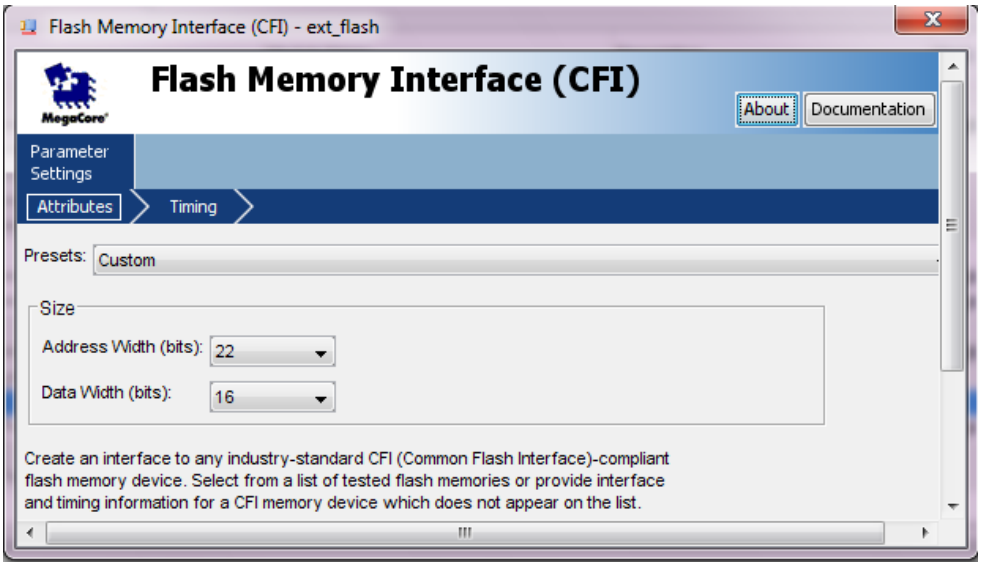

Bild 4-7: Generierung von Flash Speicher

Im Tab Attributes soll Adressbreite auf 22 und die Datenbreite auf 16 gestellt werden. Im Tab Timing Setup soll auf 20, wait auf 100, hold auf 20 und units auf ns gestellt werden. Dann auf finish klicken.

Schritt 7: Komponente Avalon Tristate-MM Bridge einfügen - Aus dem Fenster System Contents wähle Bridges and Adapters → Memory Mapped → Avalon-MM Tristate Bridge aus.

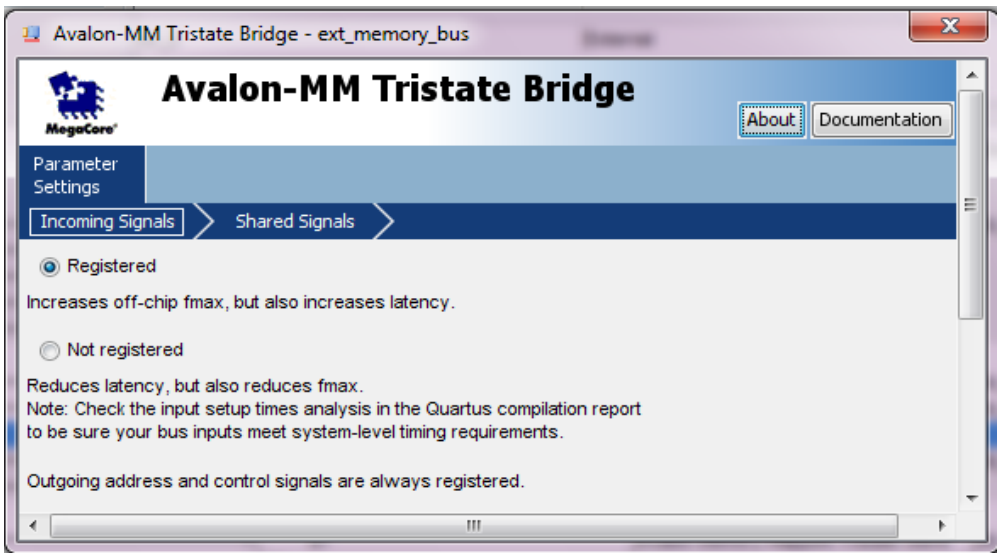

Bild 4-8: Generierung von Avalon-MM Tristate Bridge (Memory Bus)

Im Tab Incoming Signal soll „Registered“ ausgewählt werden. Im Tab Shared Signals sollen alle sieben Checkboxen aktiviert werden. Dann auf finish klicken.

Anmerkung: Avalon Tristate Slave Port von SRAM und s1 Port vom Flash Speicher sollen mit dem Tristate Master Port von Avalon-MM Tristate Bridge verbunden werden.

Schritt 8: Komponente SDRAM einfügen - Aus dem Fenster System Contents wähle Memories and Memory Controllers → SDRAM → SDRAM Controller aus.
Im Tab Memory Profile soll für Presets **single Micron MT48LC4M32B2-7 chip** ausgewählt werden. Klick auf finish.

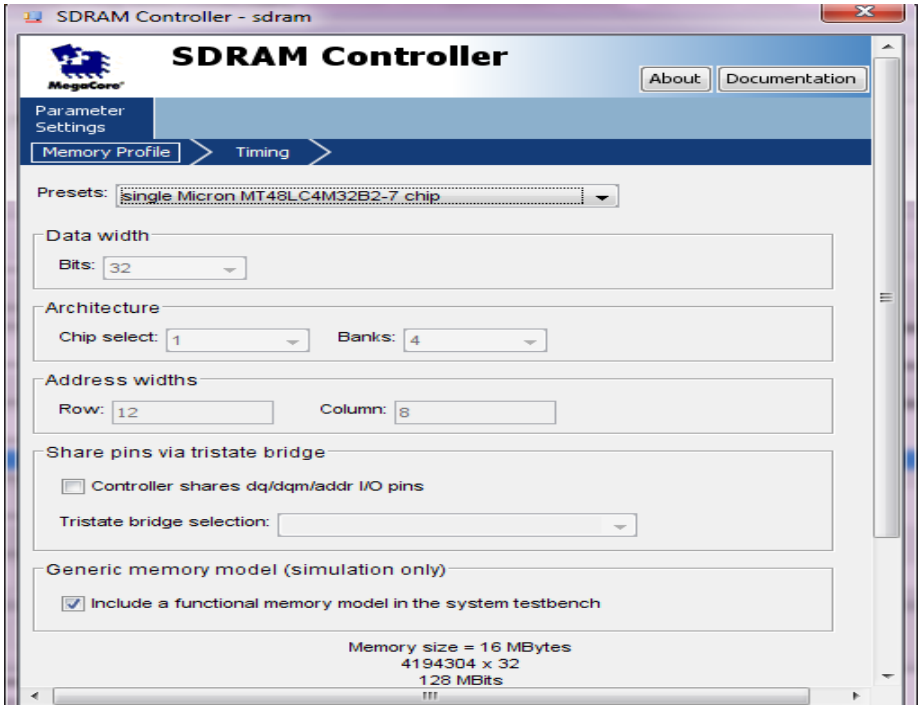

Bild 4-9: Generierung von SDRAM Controller

Schritt 9: Komponente JTAG UART einfügen - Aus dem Fenster System Contents wähle Interface Protocols → Serial → JTAG UART aus. Die Default Einstellungen sind zu übernehmen.

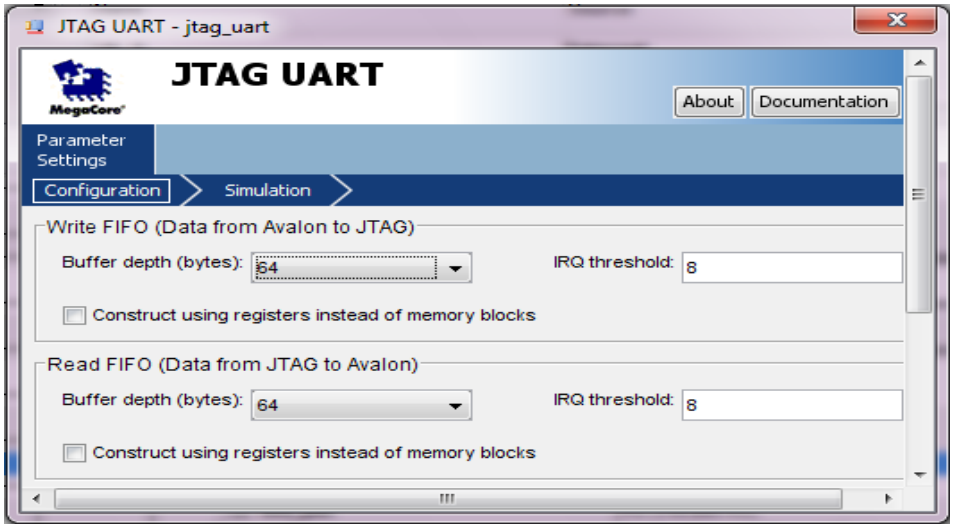

Bild 4-10: Generierung von JTAG UART

Schritt 10: Komponente UART RS232 einfügen - Aus dem Fenster System Contents wähle Interface Protocols → Serial → UART (RS-232 Serial Port) aus.

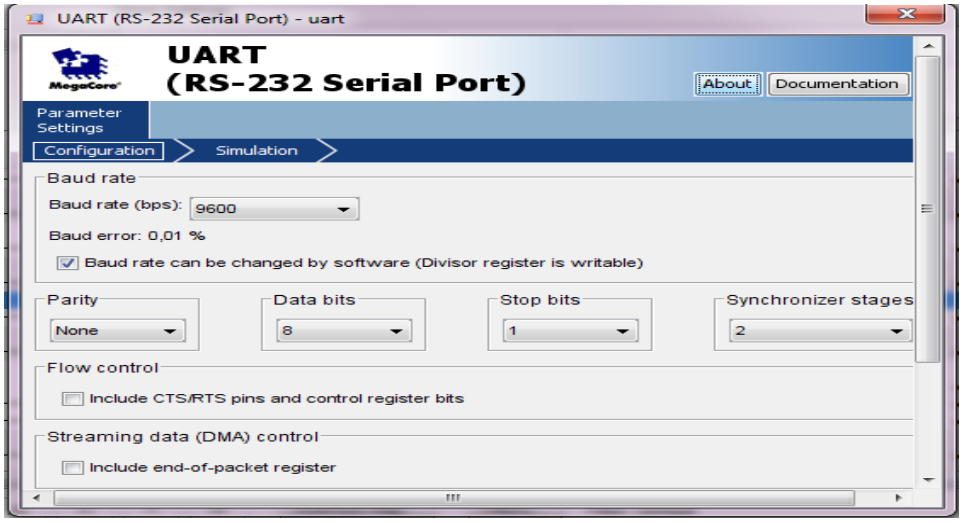

Bild 4-11: Generierung von UART RS232

Bei dem UART RS232 können die Default Einstellungen übernommen werden. Falls man die Baudrate durch die Software ändern möchte, muss der Checkbox „Baud rate can be changed by Software“ aktiviert werden.

Schritt 11: Komponente 7-Segmentanzeige einfügen - Aus dem Fenster System Contents wähle Peripherals → Microcontroller Peripherals → PIO (Parallel I/O) aus.

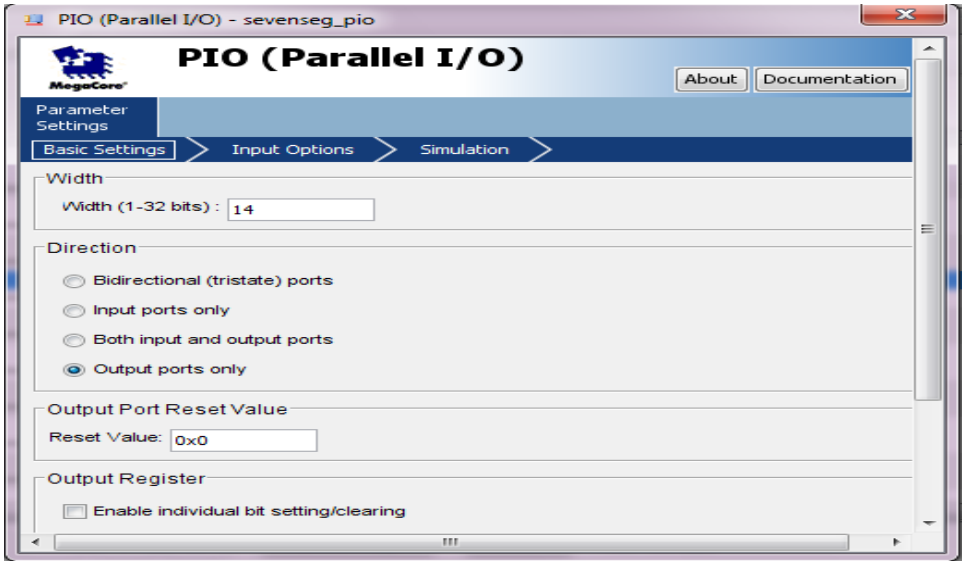

Bild 4-12: Generierung von PIO für 7-Segmentanzeige

Für die Breite (Width) soll 14 geschrieben und als Richtung (Direction) „Output ports only“ ausgewählt werden. Dann auf finish klicken.

Schritt 12: Komponente 8 LED-s einfügen - Aus dem Fenster System Contents wähle Peripherals → Microcontroller Peripherals → PIO (Parallel I/O) aus.
Für die Breite (Width) soll 8 geschrieben und als Richtung (Direction) „Output ports only“ ausgewählt werden. Dann auf finish klicken.

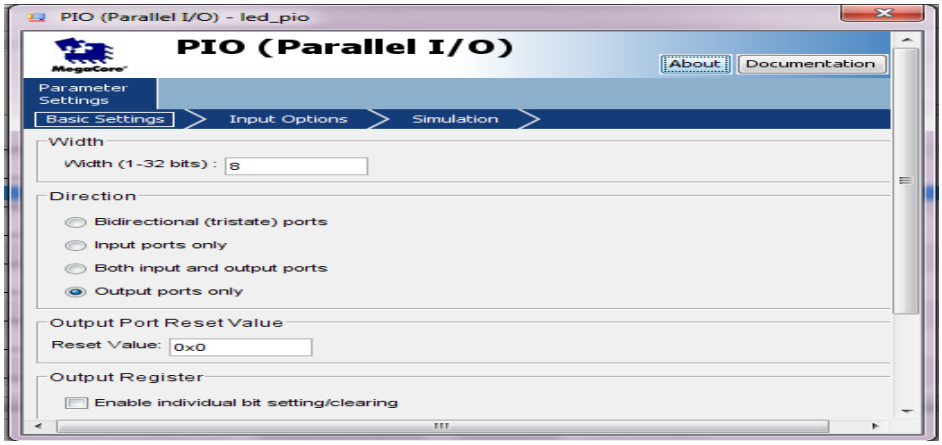

Bild 4-13: Generierung von PIO für 8 LED-s

Schritt 13: Komponente 4 Buttons einfügen - Aus dem Fenster System Contents wähle Peripherals → Microcontroller Peripherals → PIO (Parallel I/O) aus.

Für die Breite (Width) soll 4 geschrieben und als Richtung (Direction) „Input ports only“ ausgewählt werden. Dann auf finish klicken.

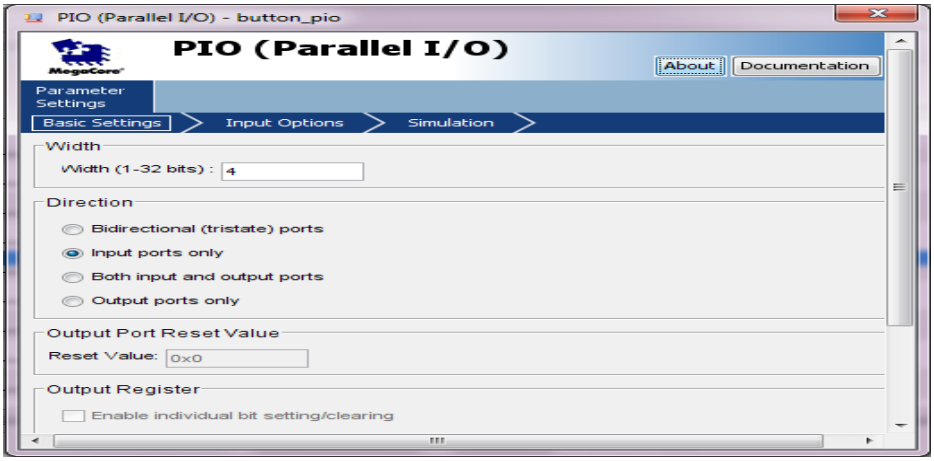

Bild 4-14: Generierung von PIO für 4 Buttons

Schritt 14: Komponente System ID einfügen - Aus dem Fenster System Contents wähle Peripherals → Debug and Performance → System ID Peripheral aus. Klick auf finish.

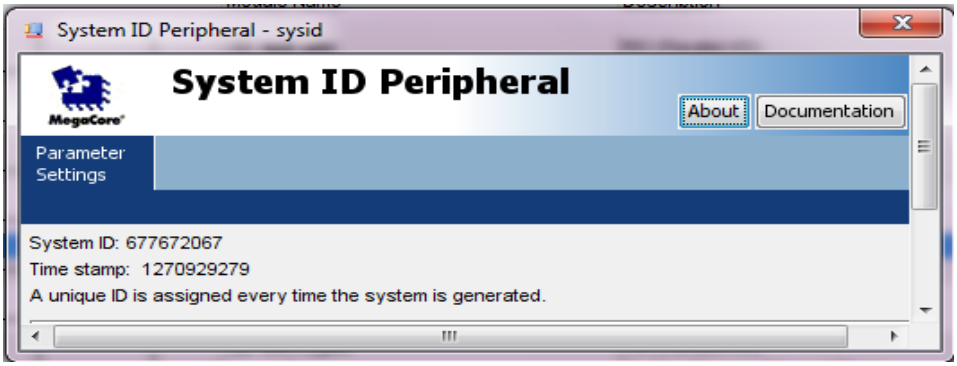

Bild 4-15: Generierung von System ID

Schritt 15: Komponente Temperatursensor einfügen - Aus dem Fenster System Contents wähle Peripherals → Microcontroller Peripherals → PIO (Parallel I/O) aus. Für Width soll 3 geschrieben und als Direction „Bidirectional Tristate Ports“ ausgewählt werden. Dann auf finish klicken

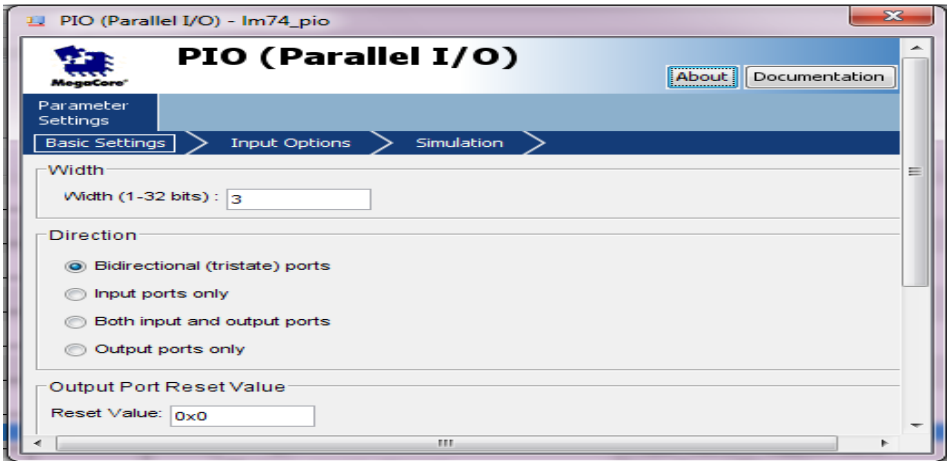

Bild 4-16: Generierung vom Temperatursensor

Schritt 16: Komponente EPCS Controller einfügen - Aus dem Fenster System Contents wähle Memory and Memory Controllers → Flash → EPCS Serial Flash Controller. Default Einstellungen sind beizubehalten, dann auf finish klicken.

Bild 4-17: Generierung vom EPCS Controller

Schritt 17: Komponente Tightly Coupled Memory einfügen - Aus dem Fenster System Contents wähle Memory and Memory Controllers → On-Chip → On-Chip-Memory (RAM or ROM) aus.

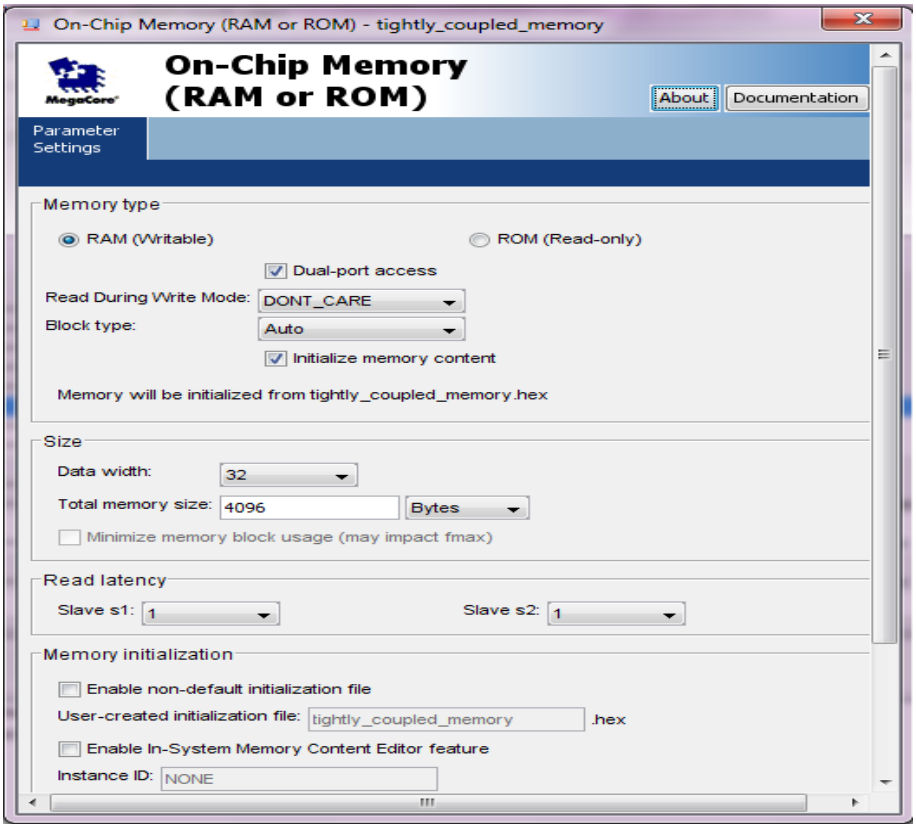

Bild 4-18: Generierung von Tightly Coupled Memory

Wie an oben gezeigtem Bild sollen RAM (Writable) Radiobox, Dual-port access und Initialize memory content Checkboxen aktiviert werden. Data width soll auf 32 und Total memory size auf 4096 Bytes gesetzt werden. Anschließend auf finish klicken.

Schritt 18: Komponente Ethernet einfügen - Aus dem Fenster System Contents wähle Interface Protocols → Ethernet → Triple-Speed Ethernet aus.

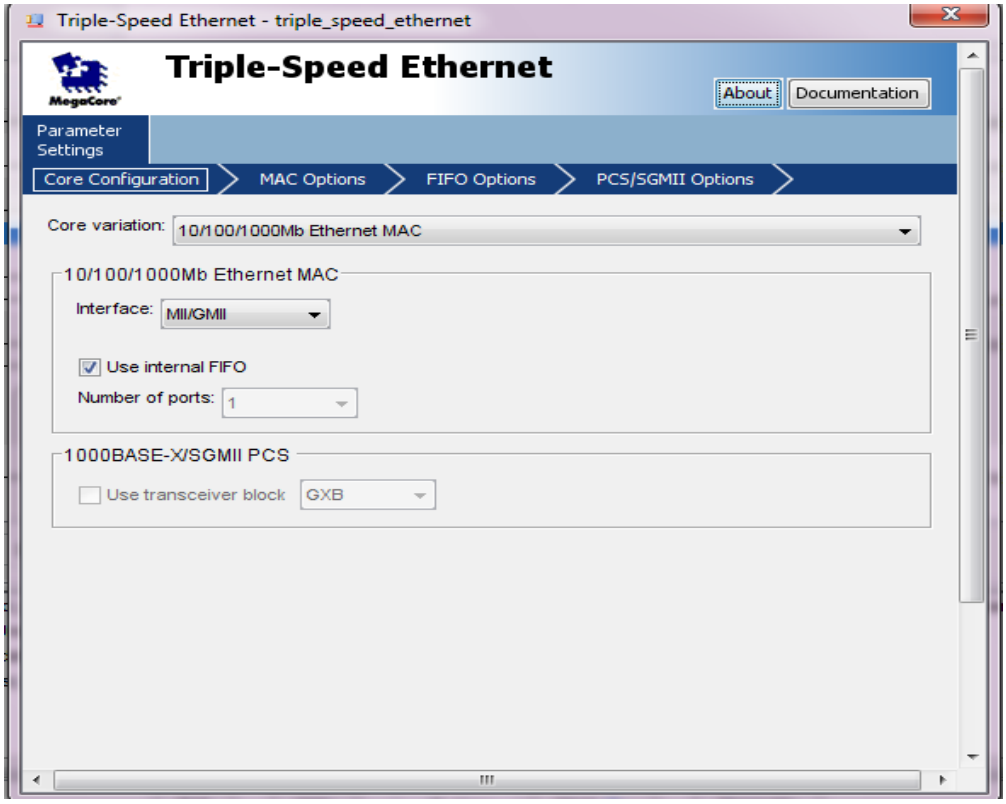

Bild 4-19: Generierung von Ethernet

Im Tab Core Configuration soll als Interface MII/GMII ausgewählt werden, und Checkbox Use Internal FIFO aktiviert werden.
Im Tab MAC Options sollen folgenden Checkboxen aktiviert werden: Enable MAC 10/100 half duplex support, Include statistics counters, Align packet headers to 32-bit boundary, Enable magic packet detection und Include MDIO Module. Host clock divisor soll auf 40 gesetzt werden. Im Tab FIFO Options soll Memory Block auf AUTO und Width auf 32 Bits gestellt werden. Im Anschluss auf finish klicken.

Schritt 19: Nun sind noch zwei Scatter-Gather DMA Controller für Ethernet zu generieren– SGDMA_TX mit dem Transfermodus „Memory to Stream“ und SGDMA_RX mit dem Transfermodus „Stream to Memory“. Data Width soll bei beiden SGDMA Controllern auf 32 gestellt werden, Source Error Width auf 1 und Daten Transfer FIFO Depth auf 64.

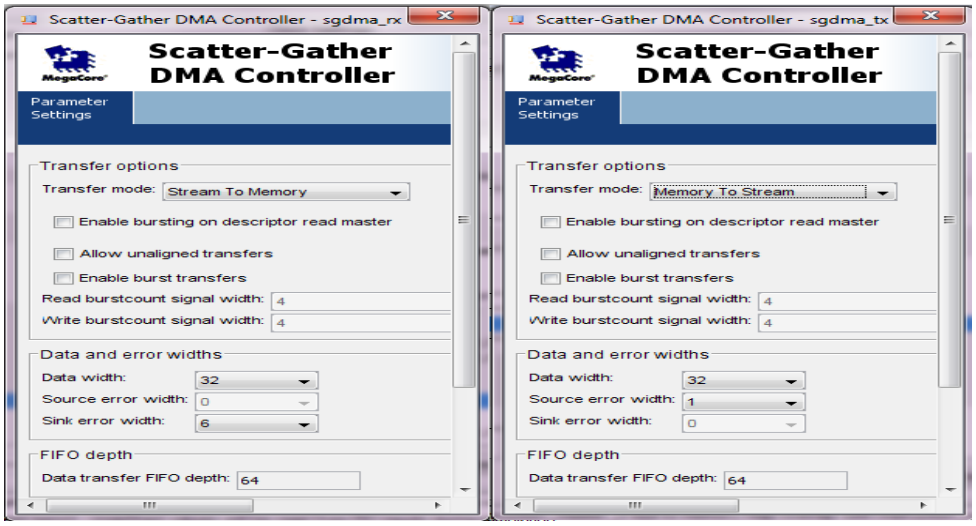

Bild 4-20: Generierung vom Scatter-Gather DMA Controller

Schritt 20: Damit die Ethernetschnittstelle korrekt funktioniert, müssen noch ein Deskriptor Speicher und Deskriptor Offset Bridge eingefügt werden. Die Deskriptor Offset Bridge dient zur Verbindung zwischen Ethernet, SGDMA Controller und Deskriptor Speicher. Als Deskriptor Speicher wird wieder eine On-Chip Memory Komponente benutzt, und für die Deskriptor Offset Bridge wird die Avalon-MM Pipeline Bridge aus der Kategorie Bridges and Adapters genommen.

Die Einstellungen, die vorgenommen werden sollen, sind aus dem Bild 4-21 abzulesen.

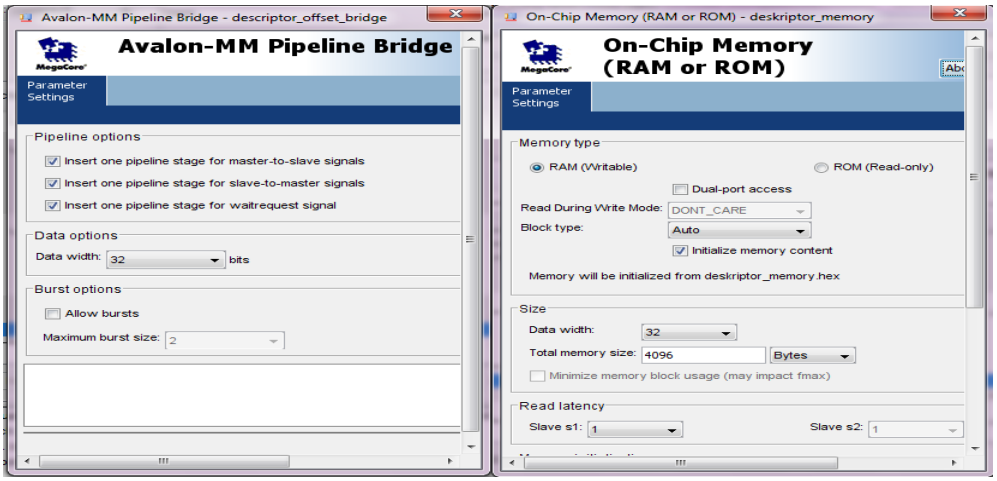

Bild 4-21: Generierung von Deskriptor Speicher und Deskriptor Offset Bridge

Unmittelbar nachdem alle Komponenten des Systems ausgewählt sind, sollen Verbindungen hergestellt werden. Die folgenden Bilder zeigen, wie die Systemkomponenten interagieren und wie die Verbindungen realisiert sind. Entsprechend der gezeigten Bilder sollen in der Spalte „Connections“ von SOPC Builder GUI die Verbindungen gesetzt werden.

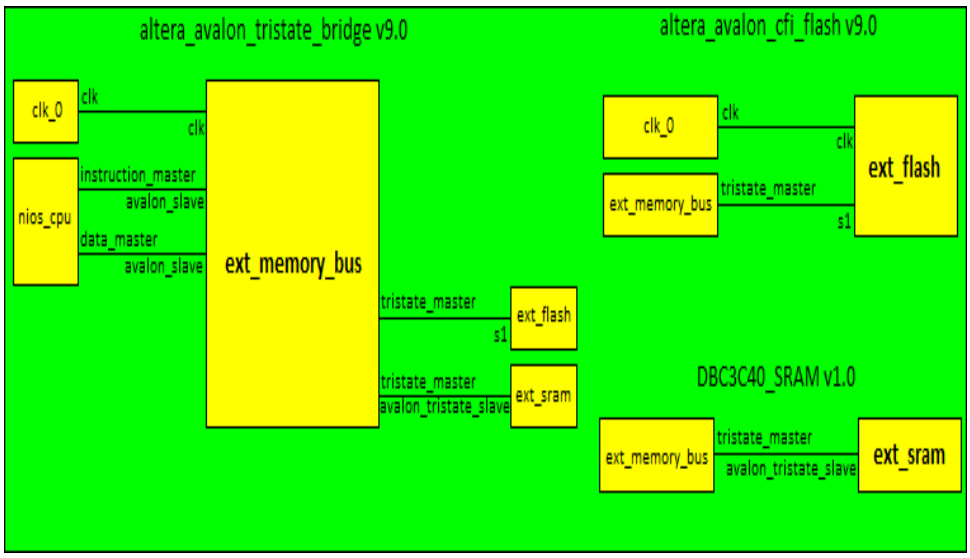

Bild 4-22: Die Verbindungen von den Flash, SRAM und Memorybus

Im nächsten Bild werden die Verbindungen vom Temperatursensor, LED-s, Buttons und 7-Segmentanzeige dargestellt.

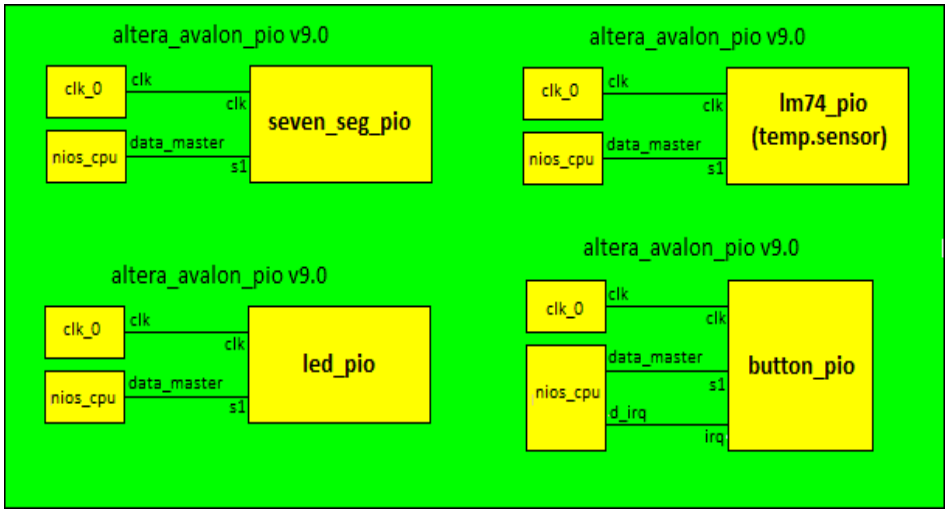

Bild 4-23: Die Verbindungen von 7-Segmentanzeige, Temperatursensor, LED-s und Buttons

Das Bild 4-24 soll die Verbindungen von der JTAG_UART, UART RS-232, System ID und Timer verdeutlichen.

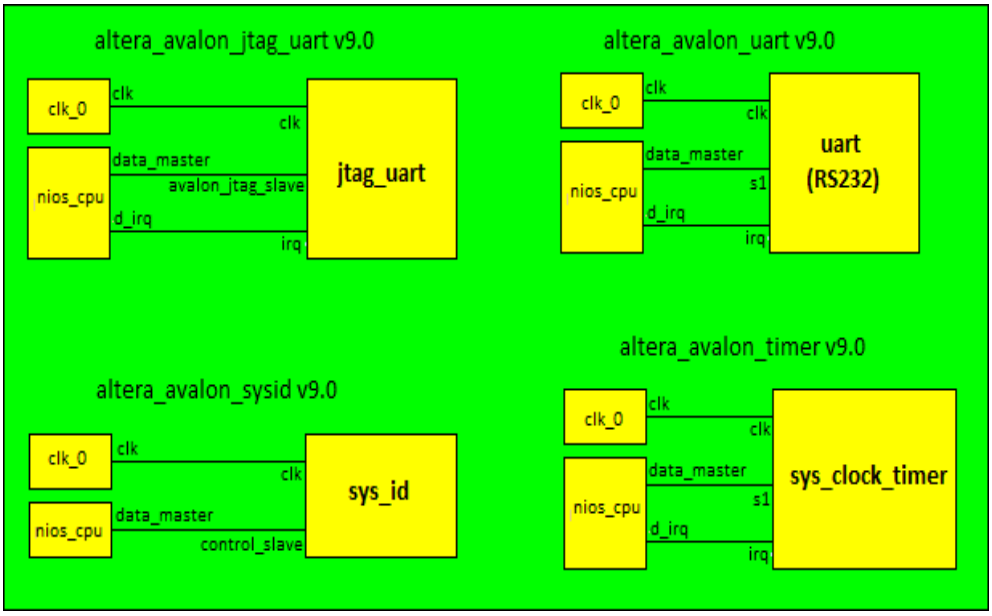

Bild 4-24: Die Verbindungen von der JTAG UART, UART RS-232, System ID und Timer

Die zwei Controller des Systems werden im folgenden Bild dargestellt:

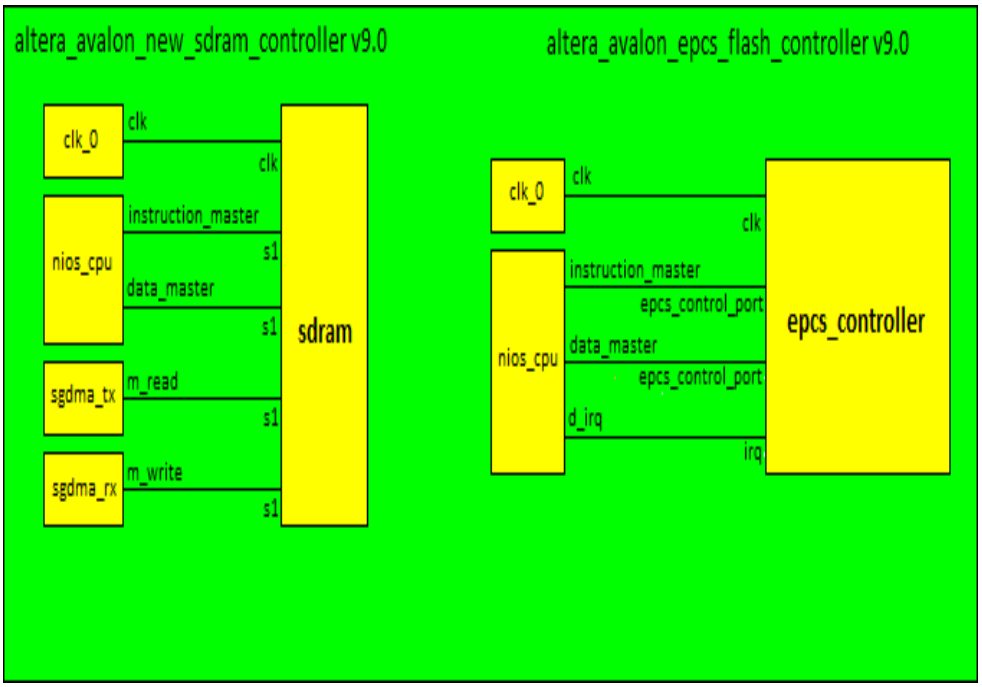

Bild 4-25: Die Verbindungen von den SDRAM-, und EPCS-Controller

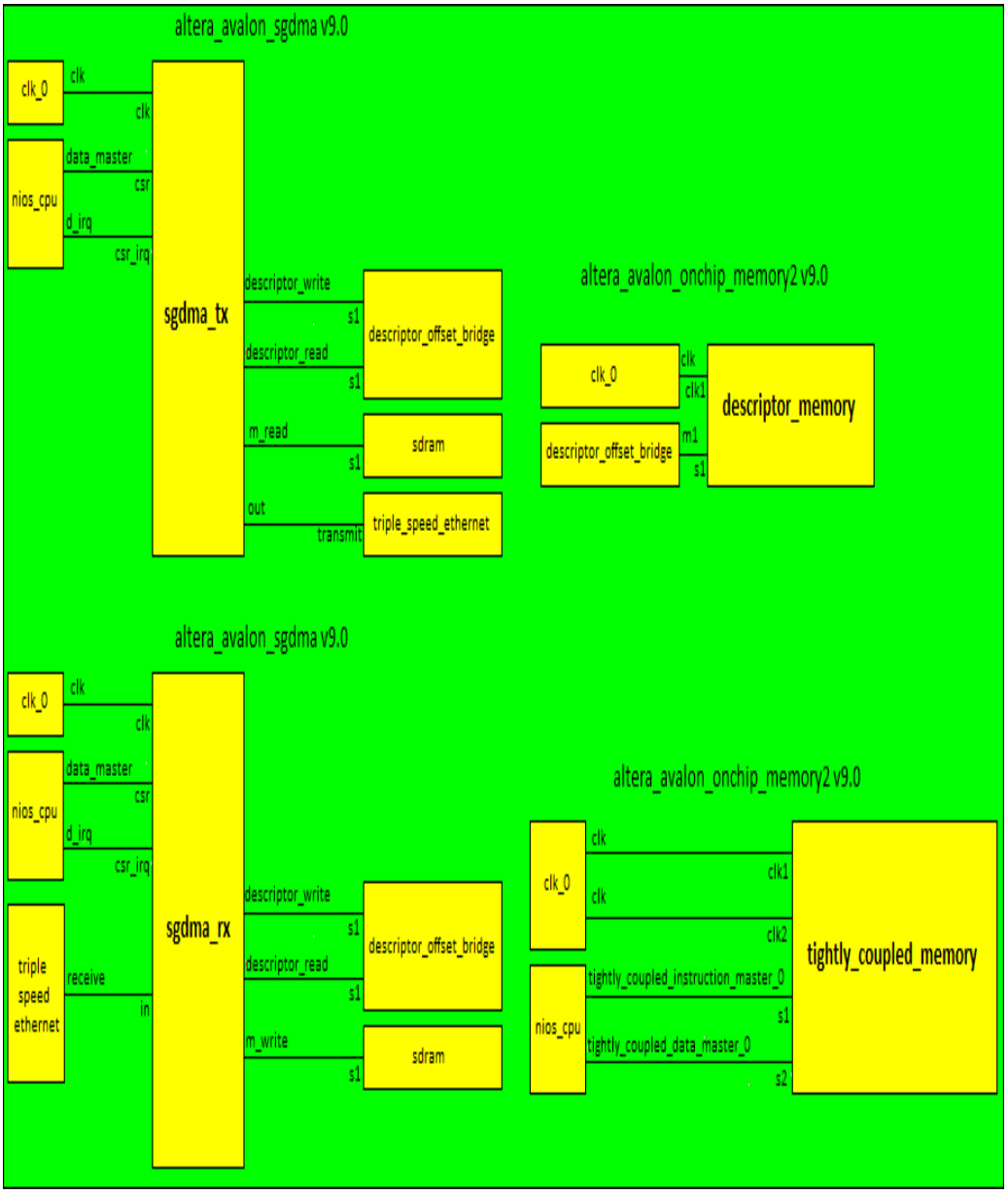

Bild 4-26: Die Verbindungen von SGDMA_TX, SGDMA_RX, Tightly Coupled Memory und Descriptor Memory

Es müssen nur noch die Verbindungen vom Ethernet und der Descriptor Bridge erläutert werden.

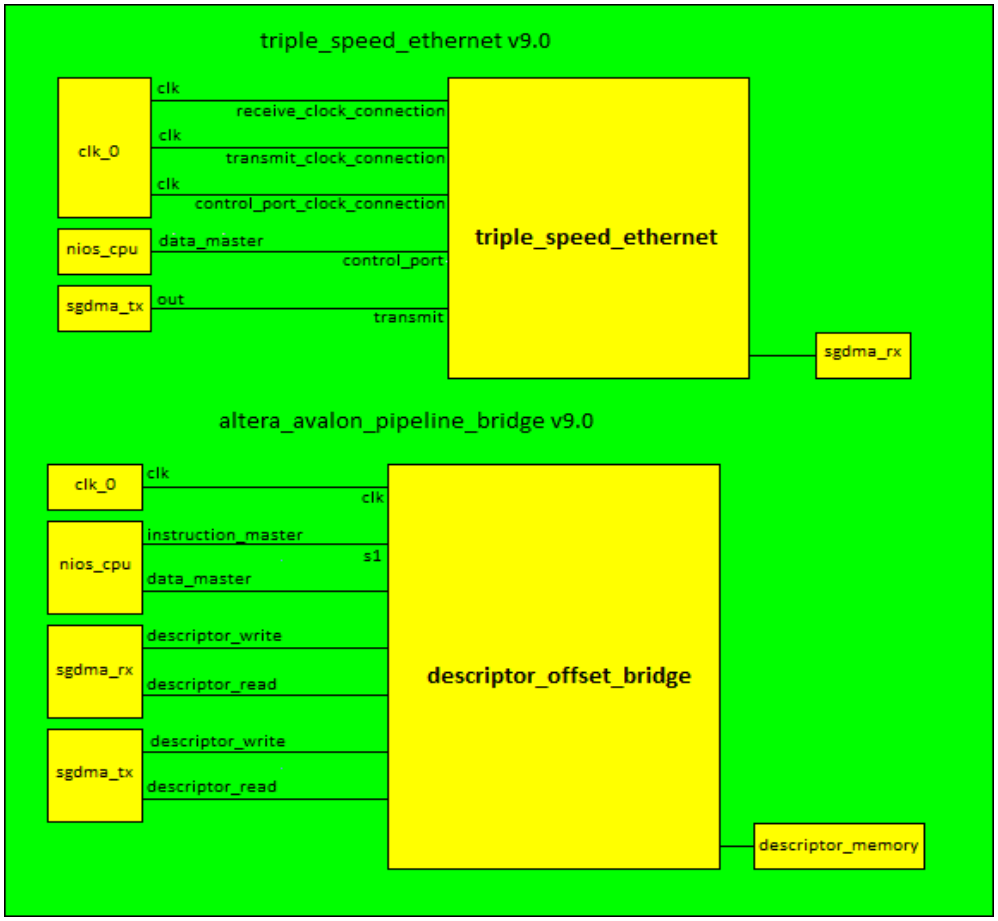

Bild 4-27: Die Verbindungen von den Ethernet und Deskriptor Offset Bridge

Um die Komponenten korrekt zu adressieren, muss man folgendes vornehmen: SOPC Builder → System → Auto-Assign Base Adresses und System → Auto-Assign IRQs. Somit entstehen keine Konflikte zwischen den Komponentenadressbereichen. Letzter Schritt mit dem SOPC Builder: auf Generate klicken.

4.3.2 Phase zwei des Hardwaredesigns – Erstellung des FPGA-Designs

Nachdem das System erfolgreich generiert wurde, kommt die zweite Phase des Hardwaredesigns – FPGA Design mit der Software Quartus II 9.0.
Das SOPC Builder Tool kann nun geschlossen werden. Im Fenster „Project Navigator“ von Quartus II 9.0 durch den Doppelklick auf .bdf File des Projektes wird ein neuer Fenster mit dem Blockschaltbild des ganzen Systems geöffnet. Jetzt müssen noch die einzelnen Systemkomponenten verbunden werden. Wie das gemacht wurde, ist im File Triple_Speed_Ethernet.bdf des Projektordners „Hardware_Design“ zu finden. Es muss erwähnt werden, dass die Eingänge bzw. Ausgänge aller Systemkomponenten mit bestimmten FPGA-Pins verbunden werden sollen.
Durch den Doppelklick auf die leere Fläche des Systemblockschaltungsfensters öffnet sich ein Fenster, wo allen Systemkomponenten als bestimmte Blockschaltungen

zu finden sind, sowie alle anderen Bauteile, die zur Entwicklung des Hardwaredesigns nötig sind. Von größter Bedeutung sind die Pins.

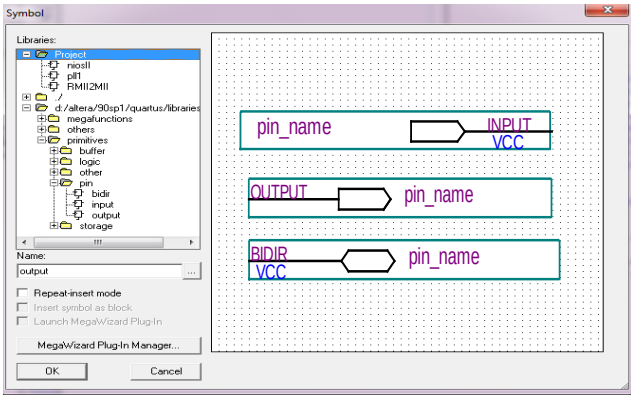

Bild 4-28: Arten der Pins für die Systemeingänge bzw. Systemausgänge

Als Pins-Namen sollen die mit dem Datenblatt vom DBC3C40 Entwicklungsboard gelieferten Pins-Namen ausgewählt werden.
Nachdem alle Ein- und Ausgänge mit den FPGA Pins verknüpft sind, kann man den Kompilierungsprozess starten – Prozessing → Start Compilation.
Der Kompilierungsprozess umfasst die folgenden Prozesse:

- Analysis and Synthesis
- Filter
- Assembler
- Classic Timer Analyzer

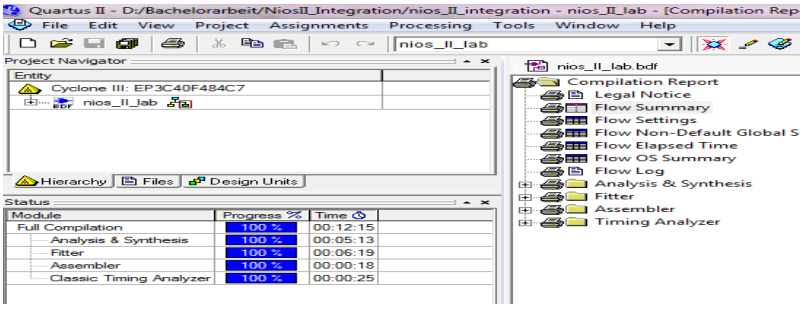

Bild 4-29: Kompilierungsprozess mit Quartus II 9.0

Nach der erfolgreichen Kompilierung wird eine Konfigurationsdatei (sof - SRAM Object File) vom Quartus II Assembler erzeugt. Diese kann auf folgender Art auf das Zielgerät (FPGA) geladen werden:

> Tools → Programmer. In „Hardware Setup“ soll als Hardware der JTAG USB-Blaster ausgewählt werden. Dann soll mit „Add File“ von dem Quartus II Assembler erzeugtes .sof File eingefügt werden. Die Checkbox „Program/Configure“ aktivieren und auf „Start“ klicken.

Somit wurde das Hardwaredesign auf den FPGA geladen.

5 Softwaredesign – Portierung von uClinux

Um den Test des Prozessors durchzuführen, gibt es verschiedene Möglichkeiten. Eine davon ist ein ganzes Betriebssystem zu portieren und auf dem Prozessor laufen zu lassen. Es können dann verschiedene Testapplikationen geschrieben werden, die als unabhängige Tasks vom Betriebssystem bearbeitet werden. In dieser Arbeit wurde ein Mikrocontroller-Linux (uClinux) als Betriebssystem eingesetzt.

5.1 uClinux

Wegen seiner Vorteile und Stärken kann Linux auch im Embedded Bereich eingesetzt werden. Immer öfter wird eine Internetkommunikation für Embedded Anwendungen gefordert (auch der Fall mit dieser Arbeit). Linux stellt eine Reihe von wichtigen Protokollen und Programmen zur Vernetzung von Anwendungen über das Internet zur Verfügung.

Vorteile beim Einsatz von Linux sind:

- keine Lizenzkosten
- hohe Investitionssicherheit (der Quell-Code kann im Notfall selbst gepflegt werden)
- hohe Stabilität
- sehr sicher
- Unterstützung verschiedenster Hardware
- Netzwerkfähigkeit
- im Kernel integriertes Firewalling-Framework
- Multi-Tasking und Multi-User
- Große Zahl unterstützter Dateisysteme
- Riesiges Software-Angebot
- Sehr gut konfigurierbar
- moderne Desktop-Oberflächen als GUI (Gnome, KDE usw.)

Der Name „uClinux“ besteht aus „uC“ (eigentlich µC) für Mikrocontroller und „Linux“. uClinux ist ein Linux-Betriebssystem, das speziell auf Systeme ohne MMU (Memory Management Unit) zugeschnitten ist. Derzeit gibt es folgende Versionen der Kernel-Serien 2.0.x, 2.4.x und 2.6.x. Alle Linux-Kernel beinhalten bereits eine TCP/IP-Unterstützung, Firewall-Möglichkeiten sowie die Treiber für die unterschiedlichsten Hardware-Komponenten.
Die letzte Version des Linux-Kernels (2.6.x) ist die stabilste. Ab dieser Version wurde uClinux mit in den offiziellen Linux-Kernel integriert. Damit ist ein riesiger Schritt für eine weitere Verbreitung, aber auch für eine weitere Annäherung und Verschmelzung der Linux-Architektur, getan.

5.1.1 Die Verzeichnisstruktur der uClinux-Distribution

Die uClinux-Distribution ist hierarchisch aufgebaut. Die wichtigsten Unterverzeichnisse sind:

config: Dateien und Skripte mit der Konfiguration für die Anwendungsprogramme.
lib: Bibliotheken für die Anwendungsprogramme.
uClibc: Implementierung der Libc auf Embedded Plattformen.
bin: Skripte und Binaries für die Erzeugung der bootfähigen Images für jeweiliges Board.
images und **romfs**: im images-Verzeichnis liegen erstellte Images (Binärdateien), und im romfs liegen die Dateien, die in das Image des ROM-Filesystems übernommen werden.
(Mehr Informationen über Mikrocontroller-Linux können aus [27] entnommen werden)

5.2 Installierungsschritte

Bevor man mit der Konfiguration von uClinux beginnt, ist es notwendig zu entscheiden, ob man unter Windows oder unter Linux arbeiten will. Erfahrungen aus der Praxis zeigen aber, dass es immer komfortabler, flexibler und effizienter ist, unter Linux zu arbeiten.
Schritt 1: Linux-Desktop installieren: hier stehen drei Linuxversionen zur Verfügung: Red Hat Enterprise Linux 4.0 und 5.0, SUSE Linux Enterprise 12.0 oder CentOS 4.0 oder 5.0. Für diese Arbeit wurde Open SUSE 12.0 ausgewählt.
Die für Open SUSE notwendigen Pakete können mit folgendem Kommando installiert werden:

*sudo zypper install git-core git-gui make gcc ncurses-devel bison byacc *
flex gawk gettext ccache zlib-devel lzo-devel pax-utilslibglade2-devel

Schritt 2: Quartus II 9.0 für Linux installieren. Folgende Dateien sind zu herunterladen: 91sp1_quartus_free_linux.tar, 91_nios2eds_linux.tar, 91sp1_nios2eds_linux.tar. Diese Dateien sollen in einem Ordner gespeichert werden. Anschließlich ist folgendes auszuführen:

tar xvf 91sp1_quartus_free_linux.tar
tar xvf 91_nios2eds_linux.tar
mkdir nios2eds_sp1
tar -C nios2eds_sp1 -xvf 91sp1_nios2eds_linux.tar

Die folgenden Programme wurden installiert:

Quartus = /opt/altera9.1sp1/quartus
IP Megacore = /opt/altera9.1sp1/ip
Nios II EDS = /opt/altera9.1sp1/nios2eds

Bei diesem Installationsschritt muss als letztes noch JTAG konfiguriert werden. Eine Datei /etc/udev/rules.d/51-usbblaster.rules ist zu erstellen. Dieser Datei sollen folgende Zeilen eingefügt werden:

 # USB-Blaster
BUS=="usb", SYSFS{idVendor}=="09fb", SYSFS{idProduct}=="6001",MODE="0666", PROGRAM="/bin/sh -c 'K=%k; K=$${K#usbdev}; printf /proc/bus/usb/%%03i/%%03i $${K%%%%.} $${K#*.}'", RUN+="/bin/chmod 0666 %c"*

Anschließend muss im Homeverzeichnis eine leere Datei „.jtag.config“ (mit der Kommandozeile *touch ~/.jtag.config*) erstellt werden.
(Für mehr Informationen über diesen Installierungsschritt siehe [14])

Schritt 3: Linux-Distribution herunterladen:

wget http://www.niosftp.com/pub/uclinux/nios2-linux-20090730.tar
tar xf <Pfad_zu>nios2-linux-20090730.tar
cd nios2-linux
./checkout

Man kann nun die uClinux-Toolchain kompilieren, oder die prekompilierte BinaryToolchain benutzen.

wget http://www.niosftp.com/pub/gnutools/...080203.tar.bz2

sha1sum nios2gcc-20080203.tar.bz2
6873249d8eae7c2981aac6791f044ddaab507566 nios2gcc-0080203.tar.bz2

sudo tar jxf nios2gcc-20080203.tar.bz2 -C /

Der Cross-Compiler ist im Verzeichnis /opt/nios2 installiert. Der aktuelle Pfad soll auf den Cross-Compiler umgestellt werden:
Folgende Zeile ist am Ende der Datei .profile im Homeverzeichnis einzufügen:

PATH=$PATH:/opt/nios2/bin

(für mehr Informationen über diesen Installierungsschritt siehe [15])

Somit wurde die notwendige Installation abgeschlossen

5.3 Erste Konfiguration und erstes Booten

Terminal starten → mount –a (wegen JTAG) → jtagconfig → quartus
Falls beim Start der Quartus Software Probleme auftreten, muss folgende Zeile eingegeben werden, und das Kommando “quartus” wieder ausgeführt:

xhost +local:root

Mit dem Quartus Software kann .sof File auf den FPGA des DBC3C40-Boards geladen werden:
Tools → Programmer → add File (.sof) → Start (siehe Details aus dem Abschnitt 4.3.2). Somit ist die Konfigurationsdatei des Hardwaredesigns auf dem FPGA vorhanden.

Ein neues Terminal starten → *cd ~/nios2-linux/uClinux-dist*
make vendor_hwselect SYSPTF=/Pfad_zum_Hardwaredesignprojekt/Sys_Name.ptf

Die PTF-Datei beschreibt die Konfiguration des Hardwaredesigns. Sie beinhaltet alle Bauteile, Adressen und Verbindungen zwischen den einzelnen Systemkomponenten. Diese Datei ist von größter Bedeutung für die Kernelkonfiguration, weil daraus andere Dateien erzeugt werden, welche bei dem Compilierungsvorgang eine sehr große Rolle spielen: hardware.mk (Basisinformationen über die Systemkonfiguration) und nios2.h (Adressen der einzelnen Komponenten und deren Namen in Form von Makros).

Nach der letzten Kommandozeile wird gefragt, welche CPU und welcher Arbeitsspeicher benutzt werden. Für die CPU gibt es nur eine Möglichkeit - Nios2, für den Arbeitsspeicher werden alle aus dem PTF abgelesenen Speicherchips angeboten – es sollte SDRAM ausgewählt werden.

Nächste Kommandozeile ist:
make menuconfig

```
Vendor/Product Selection --->                  # auswählen
   --- Select the Vendor you wish to target
       Vendor (Altera)  --->                   # Altera auswählen
   --- Select the Product you wish to target
       Altera Products (nios2)   --->          # nios2 auswählen

Kernel/Library/Defaults Selection --->         # auswählen
   --- Kernel is linux-2.6.x
       Libc Version (None)  --->               # None auswählen
   [*] Default all settings (lose changes)     # auswählen
   [ ] Customize Kernel Settings
   [ ] Customize Vendor/User Settings
   [ ] Update Default Vendor Settings
```

Dann <exit> <exit> <yes> und anschließlich das Kommando *make* ausführen.
Die Kompilierung des Kernels und der Applikationen hat begonnen. Nach dem erfolgreichen Compilierungsvorgang kann man das erste Image auf dem Board booten. Kommandozeilen dafür sind:
nios2-download –g images/zImage
nios2-terminal

(Dieser Vorgang basiert auf [16])

5.4 Kernel- und Applikationskonfiguration anpassen

Um die Kernel- und Applikationskonfiguration auf eigene Bedürfnisse anzupassen, muss folgendes getan werden:
make menuconfig

Kernel/Library/Defaults Selection --->
(linux-2.6.x) Kernel Version
(None) Libc Version
[ ] Default all settings (lose changes)
[*] Customize Kernel Settings ← zum Ändern der Kernelkonfiguration
[*] Customize Vendor/User Settings ← zum Ändern der Applikationskonfiguration
[ ] Update Default Vendor Settings

Bei der Kernelkonfiguration muss folgendes angepasst werden, alle anderen Default-Einstellungen können übernommen werden:

- Prozessor type and Features:
  - Platform → Altera Cyclone Development Board Support
  - [*]Support of DMA controller with Avalon interface

- [*] Networking support

- Device Drivers
  - [*] Block devices
  - [*] Network device support
    - [*] Ethernet (10 or 100Mbit)
      - [*] Altera Triple Speed Ethernet MAC support
      - [*] PHY Device support and infrastructure
        - [*] Drivers for National Semiconductor PHYs
  - [*] I2C support
    - [*] I2C device interface
    - [*] Autoselect pertinent helper modules
  - [*] SPI support
    - [*] Altera SPI Controller
  - [*] DMA Engine support

Dann <exit> <exit> <yes>.
Nun soll noch die Applikationskonfiguration angepasst werden:

- Library Configuration
  - [*] bulid realy old libnet

- Network Applications
  - [*] mii-tool

Wieder <exit> <exit> <yes> und anschließend *make* ausführen.
(Dieser Vorgang basiert auch auf [16])

Die Kompilierung der angepassten uClinux-Distribution hat begonnen. Da der Treiber für Alteras Triple Speed Ethernet relativ neu ist, ergab sich mit ihm eine Reihe unterschiedlicher Probleme. Einen komplett neuen Treiber zu schreiben, wäre bei dieser Arbeit aus mehreren Gründen nicht möglich gewesen. Deshalb war die einzige Möglichkeit nach dem potentiellen Lösungsansatz zu recherchieren.

Nach der Anpassung der Dateien - atse.c (Zeilen 1645, 1672-1681, 1690-1699) und altera_tse.c (Zeilen 853, 1660-1669, 1683-1692, 1708-1717), wurden die Probleme mit dem Ethernet Treiber beseitigt. Diese Files sind unter ~/nios2-linux/linux-2.6/drivers/net/ zu finden.

Mit dem *ping <ip_adresse>* Kommando kann getestet werden, ob der Ethernettreiber wirklich korrekt funktioniert. Es hat sich ergeben, dass die MAC Adresse nicht abgelesen werden konnte und dass die Netzmaskadresse falsch gesetzt ist, deswegen mussten sie manuell gesetzt werden. Dieses Problem wurde auch behoben (siehe im Anhang Eth_Init).

6 Sensorik

Der Nios II Prozessor interagiert mit zwei Sensoren – einem LM74 Temperatursensor und einem TSL250R Lichtsensor. Der LM74 Temperatursensor ist auf dem DBC3C40-Board integriert, da aber Kernelversion 2.6.30 (letzte Kernelversion) keinen Treiber für diesen Sensor anbietet, musste dieser programmiert werden. Der Quellcode dieses Treibers ist im Anhang zu finden.
Der Lichtsensor kommuniziert mit dem Nios II System über einen AD-Wandler, weil er nicht auf dem Board vorhanden ist, musste er extern angeschlossen werden.

6.1 LM74 Temperatursensor

Der LM74 Temperatursensor vereinigt einen „band-gap“ Temperatursensortyp und einen 12-Bit Delta-Sigma Analog/Digital-Wandler. Die Kompatibilität der Drei-Leitung-Seriellenschnittstelle dieses Sensortyps mit SPI bzw. MICROWIRE ermöglicht die Kommunikation mit den verschiedenen Mikrocontrollern bzw. Prozessoren. Das ID-Register identifiziert diesen Sensor als ein Produkt der Firma National Semiconductor.
Immer, wenn der LM74 Temperatursensor eingeschaltet ist, operiert er in einem kontinuierlichen Modus. Dies bedeutet, dass der aktuelle Temperaturwert aus dem Temperaturregister ständig abgelesen werden kann. Unmittelbar nach der Einschaltung liefert der Sensor einen falschen Wert bis die erste Temperaturumwandlung vollständig abgeschlossen ist.
Dieser Sensor funktioniert immer als ein Slave und ist, wie schon erwähnt, kompatibel mit einer SPI- oder MICROWIRE-Schnittstelle. Ein vollständiges Übertragen/Empfangen Zyklus braucht 32 Taktsignale, 16 zur Übertragung und 16 zum Empfangen.
Der LM74 Sensor besitzt drei Register: Konfigurationsregister (um den Operationsart auszuwählen, da aber kontinuierlicher Modus automatisch nach der Einschaltung konfiguriert wird, ist dieser Register bei der Treiberentwicklung von keiner großen Bedeutung), ID-Register und Temperaturregister.

Tabelle 6.1: Temperaturregister des LM74 Sensor

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
MSB	Bit 11	Bitt 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	LSB	1	X	X

D0 und D1 sind undefiniert.
D2 ist immer auf 1 gesetzt.
D3-D15 sind Temperaturdaten.
Der Temperaturwert wird aus 13 Bits gebildet. Es handelt sich um ein Zweierkomplement Word mit einem LSB = 0,0625° C.
Ein Beispiel dazu: Vom Dataregister wurde folgender Wert abgelesen: 0000 1100 1000 0111. Letzte 13 Bits werden berücksichtigt, also 0000 1100 1000 0. In dezimaler From entspricht dieser Wert der Zahl 400.
400 * 0,0625° C = 25° C.

Tabelle 6.2: Temperaturwerte des Temperaturregister

Temperatur	Digitale Ausgabe	
	Binär	Hexa
+150° C	0100 1011 0000 0111	4B 07h
+125° C	0011 1110 1000 0111	3E 87h
+25° C	0000 1100 1000 0111	0C 87h
+0,0625° C	0000 0000 0000 1111	00 0Fh
0° C	0000 0000 0000 0111	00 07h
-0,0625° C	1111 1111 1111 1111	FF FFh
-25° C	1111 0011 1000 0111	F3 87h
-55° C	1110 0100 1000 0111	E4 87h

Ein Lesezyklus ist auf der folgenden Abbildung anzusehen:

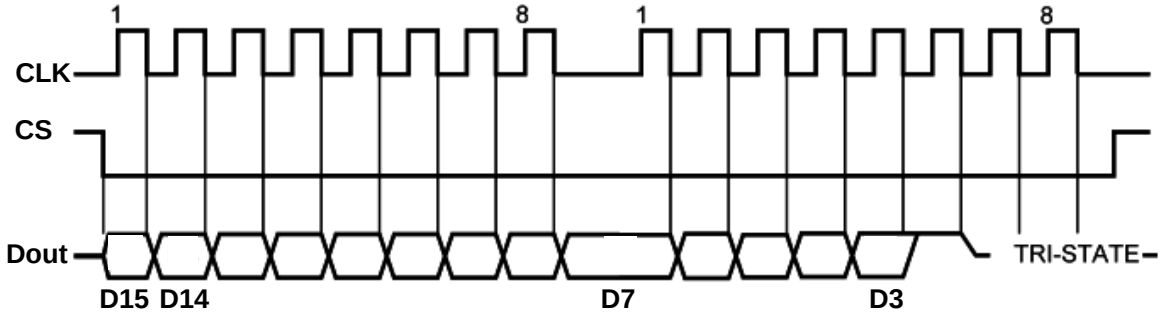

Bild 6-1: Lesezyklus beim LM74 Temperatursensor

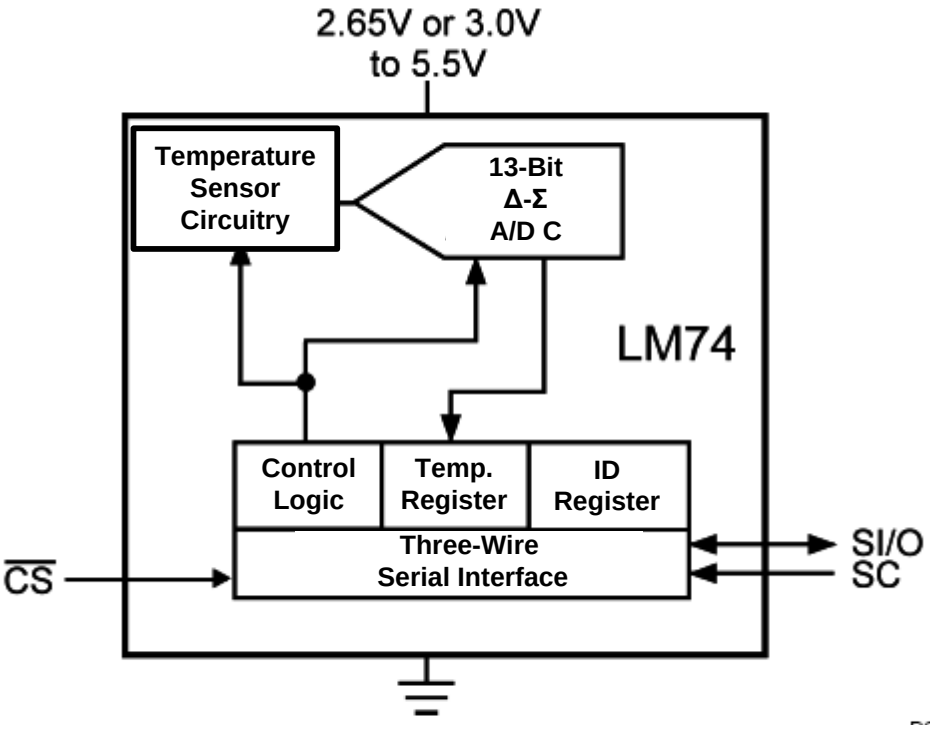

Bild 6-2: Blockschaltbild des LM74 Temperatursensor

Wie dieser Sensor mit dem Nios II System interagiert wird im nächsten Bild vorgestellt. Es handelt sich um eine sehr einfache SPI-Schnittstelle.

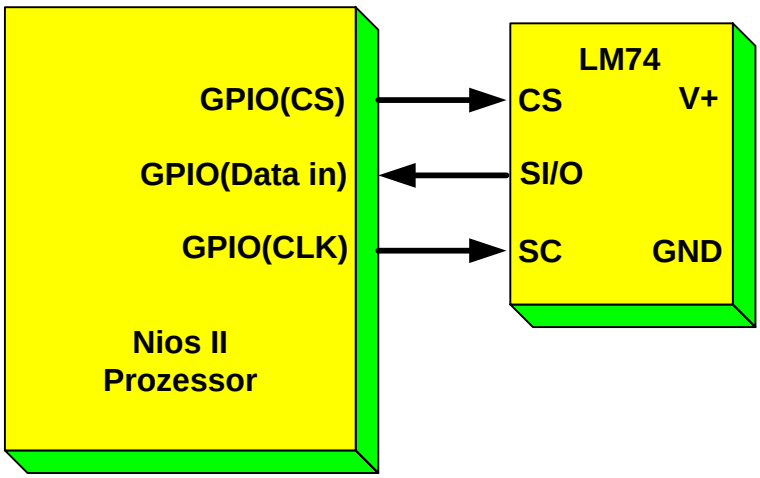

Bild 6-3: Interaktion zwischen Nios II Prozessor und LM74 Temperatursensor

(Ausführliche Informationen über diesen Sensor sind unter [17] verfügbar)

6.2 TSL250R Lichtsensor

Der TSL250R besteht aus einer Fotodiode und einem integrierten Verstärker. Zum Anschluss an die Schaltung benötigt der TSL250R ausschließlich Versorgungsspannung (Pin 2) und Masse (Pin 1). An Pin 3 liefert der Sensor eine Ausgangsspannung

welche proportional zum einfallenden Licht ist. Die spektrale Empfindlichkeit reicht von Ultraviolett bis Infrarot und hat ihr Maximum im sichtbaren Bereich des Lichtspektrums.

Der TSL250R arbeitet mit Betriebsspannungen zwischen 2,70 und 5,50 V und einem typischen Strom von 1,1 mA.
Das Maximum der Ausgangsspannung (unter voller Bestrahlung) für diesen Sensor liegt unter 4V, wenn der Baustein mit 5V betrieben wird. Somit hat man die Wahl, den Sensor direkt ohne zusätzliche Beschaltung mit einer Referenzspannung von 5V über einen ADC-Kanal an den Kontroller anzuschließen. Ohne zusätzliche Verstärkung verliert man aber über den möglichen Spannungsbereich hinweg 1V Auflösung.

Für einen 8-Bit-ADC bedeutet eine 4V Eingangsspannung ein Messwert von 0xCC. Der Messwertebereich geht damit von 0x00 bis 0xCC. Abhängig von der Applikation ist dies kein Problem. Wenn es aber darum geht präzisere Lichtmessungen durchzuführen sollte der gesamte verfügbare ADC-Bereich ausgenutzt werden. Für diesen Zweck könnte ein Operationsverstärker verwendet, welcher das Sensor-Ausgangssignal mit einem Verstärkungsfaktor von 1,25 auf die vom ADV möglichen 5V Maximum verstärkt. Da in diesem Projekt der AD7708 benutzt wird, der maximal 2,5 Volt aufnehmen kann, reicht eine Eingangsspannung von 3,3 Volt aus. [18]

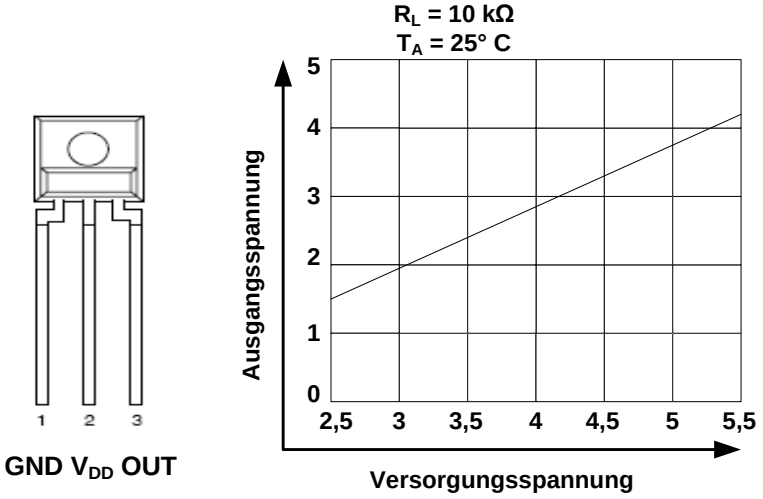

Bild 6-4: Pinbelegung und Spannungsabhängigkeit des TSL250R Lichtsensors

Da das DBC3C40-Board über keinen A/D-Wandler verfügt, an dem der Lichtsensor angeschlossen werden kann, kommt ein externer A/D-Wandler zum Einsatz.
Es handelt sich um einen AD7708 Analog-Digital-Wandler. Dieser A/D-Wandler wird durch mehrere Register konfiguriert und gesteuert. Das sind folgende Register:

- Kommunikationsregister
- Statusregister
- Moderegister
- ADC Kontrollregister
- I/O Kontrollregister
- Filterregister
- ADC Datenregister
- ADC Offsetregister
- ADC Gainregister
- Testregister
- ID Register

Alle diese Register haben unterschiedliche Funktion und Größe. Im Folgenden werden nur in dieser Arbeit angesprochene Register beschrieben, nämlich: Kommunikationsregister, Moderegister, ADC Kontrollregister und Datenregister. Weitere Informationen sind im Datenblatt vom A/D Wandler zu finden (siehe[19]).

6.3 AD7708 Registerset

Kommunikationsregister ist ein Schreibregister, das 8 Bit groß ist. Jede Kommunikation beginnt mit dem Schreiben in dieses Register. Durch das Schreiben ins Kommunikationsregister wird festgestellt, ob die nächste Operation eine Schreib-, oder Leseoperation ist, und auf welchem Register die Operation ausgeführt wird.

Tabelle 6.3: Bit-Belegung des Kommunikationsregister

CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
WEN	R/W	0	0	A3	A2	A1	A0

CR7: ist ein low-aktiv. Hier muss eine Null geschrieben werden, damit man das Kommunikationsregister ansprechen kann.
CR6: eine Null an dieser Stelle bedeutet, dass die nächste Operation zum spezifizierten Register eine Schreiboperation ist. Eine Eins stellt die Leseoperation dar.
CR5, CR4: an diesen Stellen müssen Nullen geschrieben werden, damit das Kommunikationsregister korrekt arbeitet.

CR3 – CR0: Die Kombination dieser vier Bits bestimmt auf welchem Register die Operation ausgeführt wird.

Moderegister ist 8 Bit groß, zu ihm können die Daten geschrieben werden und von ihm können die Daten gelesen werden. Dieses Register konfiguriert den Operationsmodus des A/D Wandlers. Von der größten Bedeutung sind die ersten drei Bits dieses Registers.

Tabelle 6.4: Bit-Belegung des Moderegister

MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
CHOP	NEGBUF	REFSEL	CHCON	OSCPD	MD2	MD1	MD0

Die Bits MR2 – MR0 müssen wie folgend belegt werden: 011. Dies bedeutet, dass ein kontinuierlicher Modus konfiguriert ist, in dem das Datenregister kontinuierlich Updates macht, so dass immer der aktuellste Wert vom Datenregister abgelesen werden kann. Andere Bits dieses Registers sind für diese Arbeit irrelevant.

ADC Kontrollregister ist wie das Moderegister 8 Bit groß, zu ihm können auch die Daten geschrieben werden und von ihm können die Daten gelesen werden. Dieser Register wird benutzt um z. B. den Kanal zum Datenempfang zu selektieren.

Tabelle 6.5: Bit-Belegung des Kontrollregister

ADCON7	ADCON6	ADCON5	ADCON4	ADCON3	ADCON2	ADCON1	ADCON0
CH3	CH2	CH1	CH0	U/B	RN2	RN1	RN0

Die letzten vier Bits dieses Registers sind mit den Nullen belegt, das bedeutet, dass der Kanal 1 benutzt werden kann.

Datenregister ist ein Leseregister, das 16 Bit groß ist. Das Ergebnis der Konvertierung wird in diesem Register gespeichert und kann von ihm abgelesen werden, wenn der A/D Wandler das READY-Signal auf 0 setzt.

6.4 Kommunikation zwischen Nios II Prozessor und Lichtsensor

Diese Kommunikation wird über den externen A/D-Wandler AD7708 bereitgestellt. Der Lichtsensor ist direkt mit dem A/D-Wandler verbunden. Es handelt sich um eine rein physikalische Verbindung, bei welcher der Ausgangsspannungspin des Lichtsensors mit dem Kanal1-Pin des A/D-Wandler verbunden wird. Der Lichtsensor wandelt das Licht in der Spannung um, und übergibt das Ergebnis an A/D-Wandler, der A/D-Wandler wandelt Spannung (den analogen Wert) in digitalen Wert um, und übergibt das Ergebnis weiter an Nios II Prozessor.

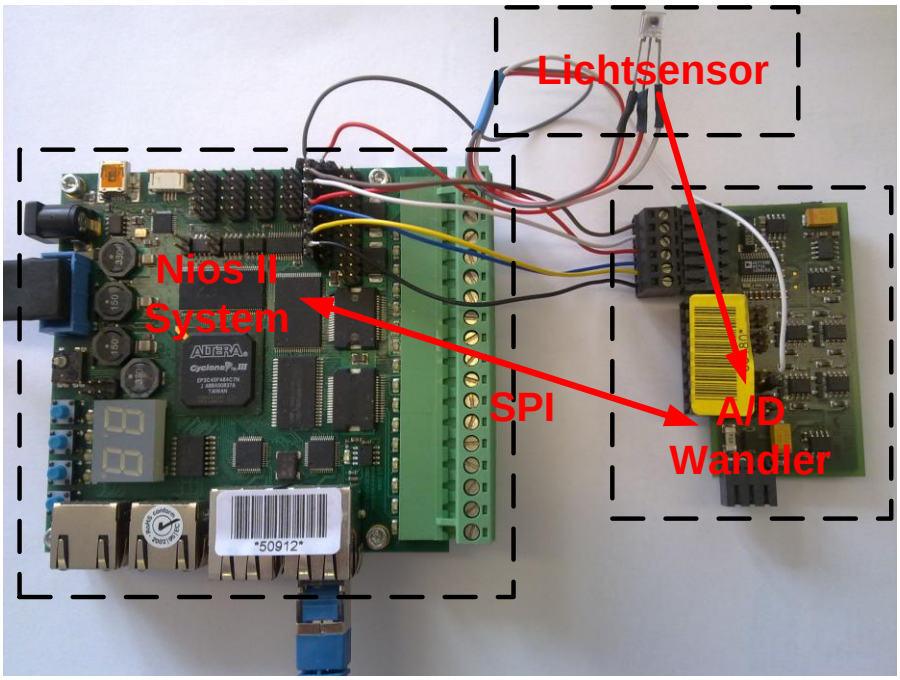

Bild 6-5: Interaktion zwischen Nios II System, A/D-Wandler und Lichtsensor

Die Kommunikation zwischen Nios II System und A/D-Wandler ist komplexer. Es kommt nicht nur der Einsatz einer SPI-Schnittstelle zum Einsatz, sondern es muss die sogenannte „On-Chip-Register“ Kommunikation verstanden und implementiert werden. Jede Kommunikation mit dem A/D-Wandler (egal ob Schreiben oder Lesen) beginnt mit dem Schreiben zum Kommunikationsregister. Durch das Schreiben ins Kommunikationsregister wird auch festgestellt, ob die nächste Operation eine Schreib-, oder Leseoperation ist, und auf welchem Register diese Operation ausgeführt wird (siehe Methode COMREG_implement vom File adwandler.c). Dieses Prinzip wird auf der folgenden Abbildung verdeutlicht:

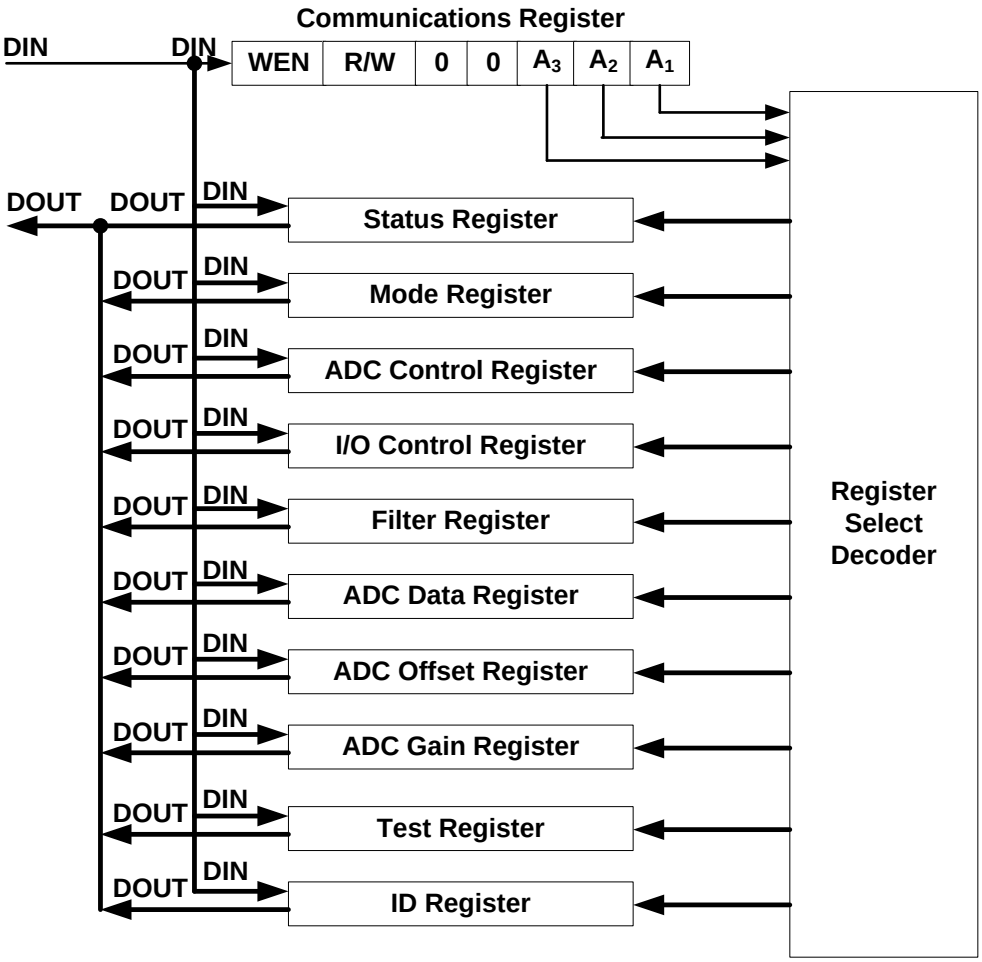

Bild 6-6: On-Chip Registerkommunikationsablauf

Beim Schreiben auf einem Register passiert folgendes:
Der Master, also der Nios II Prozessor zieht das CS Signal auf Masse (LOW), erzeugt die Taktsignale, wobei bei der negativen Flanke ein Bit gesendet wird, und bei der nachfolgenden, positiven Flanke ist der gesendete Bit beim Slave – A/D Wandler AD7708. Nachdem alle Bits zum AD7708 übertragen wurden, zieht der Nios II das CS Signal auf HIGH und somit ist ein Schreibzyklus beendet (siehe File adwandler.c, Methode ADC_write_to_reg).
Lesen von einem Register erfolgt auf folgender Weise:
Erst mal muss man im Kommunikationsregister schreiben, wobei an Stelle des R/W Bits eine Eins geschrieben werden muss. Wie es schon im Abschnitt " Register" erklärt ist, die Kombination der ersten vier Bits bestimmt von welchem Register gelesen wird.

Nach dem Schreiben im Kommunikationsregister, beginnt der Lesezyklus:
Der Nios II Prozessor setzt das Signal CS auf Masse (LOW), erzeugt die Taktsignale, wobei bei der positiven Flanke ein Bit abgelesen werden kann (der A/D Wandler sendet die Daten zum Nios II), bei der nachfolgenden negativen Flanke ist das Bit schon beim Master-Nios II angekommen. Dies wiederholt sich, bis alle Bits vom A/D Wandler zum Nios II übertragen sind. Danach zieht der Prozessor das CS Signal auf HIGH und der Lesezyklus ist somit beendet (siehe File adwandler.c, Methode ADC_read_from_reg). Beim Lesen vom Datenregister beginnt der Zyklus erst dann, wenn der A/D Wandler das RDY Signal auf 0 gesetzt hat. ([19])

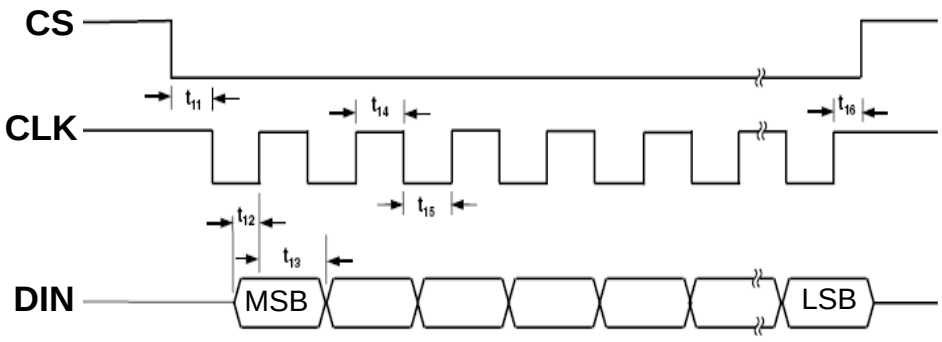

Bild 6-7: Schreiben zum A/D-Wandler

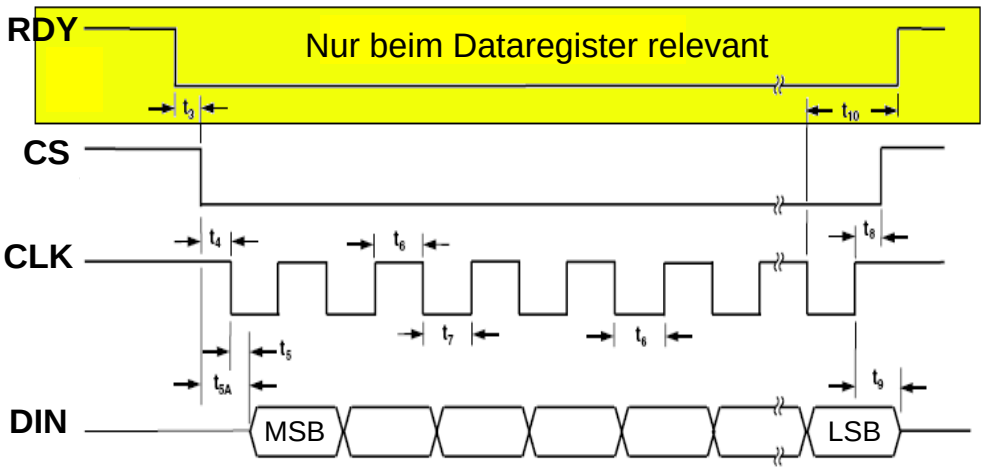

Bild 6-8: Lesen vom A/D-Wandler

Wie der A/D-Wandler Treiber als einziger Task unter uClinux gestartet wird, wird im nächsten Kapitel dieser Arbeit beschrieben.

7 Die Sensortreiber als unabhängige Tasks unter uClinux und Alteras® Triple Speed Ethernet

Es reicht nicht nur den Treiber für die Kommunikation zwischen den A/D-Wandler und Nios II Prozessor zu schreiben. Dieser Treiber soll mit anderen Linux-Files kompiliert werden. Dies wird auf folgender Weise realisiert:

- In dem Verzeichnis ~nios2-linux/uClinux-dist/user/ wird ein neuer Verzeichnis mit dem Name *adwandler* erstellt:
  cd ~linux/uClinux-dist/user/
  mkdir adwandler

- Zum Verzeichnis *adwandler* wechseln und eine Datei *adwandler.c* erstellen.
  cd adwandler
  echo –n >adwandler.c

- Den Inhalt des Treibers in *adwandler.c* kopieren.

- Folgende Zeile in ~linux/uClinux-dist/user/Makefile einfügen:
  dir_$(CONFIG_USER_ADWANDLER_ADWANDLER) += *adwandler*

- Folgende Zeilen in ~linux/uClinux-dist/user/Kconfig nach dem Menü „Miscellaneous Applications“ einfügen:
  config USER_ADWANDLER_ADWANDLER
  bool "adwandler"

- Folgende Datei ~linux/uClinux-dist/user/adwandler/Makefile erzeugen:
  echo –n >Makefile , anschließend in dieser Datei folgende Zeilen einfügen:
  EXEC = hello
  OBJS = hello.o

  all: $(EXEC)

```
$(EXEC): $(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

romfs:
    $(ROMFSINST)    /bin/$(EXEC)

clean:
    -rm -f $(EXEC) *.elf *.gdb *.o
```

Nun kann man mit dem Kommando *menuconfig* unter “Miscellaneous Applications“ *adwandler* auswählen. Mit der *make* Kommandozeile wird Linux neu kompiliert und damit auch der hinzugefügte Treiber. Nach der erfolgreichen Kompilierung, kann man uClinux wieder booten. Mit der Kommandozeile *adwandler* wird der Treiber gestartet. (Siehe [20])

Der A/D-Wandler muss als unabhängiger Task gestartet werden. Dieser Task ermöglicht die Initialisierung des A/D-Wandler und die Konfiguration des kontinuierlichen Modus (0x03 wird im Moderegister geschrieben), so dass jetzt die Ergebnisse der Lichtumwandlung ständig abgefragt werden können.

Der zweite und vor allem der wichtigste Task ist der Boa-Webserver (siehe [21]). Der korrekt funktionierende Triple-Speed-Ethernet Treiber ermöglicht den Einsatz dieses Webserver, der von uClinux frei zur Verfügung gestellt wird. Man bekommt mit diesem Webserver eine Test-HTML-Seite, die auf eigene Bedürfnisse angepasst werden kann. Dies hat die Arbeit viel einfacher gemacht. Diese Test-HTML-Seite wurde wie folgend angepasst (siehe template.c und cgi.c unter ~nios2-linux/uClinux-dist/user/cgi-generic/):

- Titelmarke
- Headermarke
- Da der Treiber des Temperatursensor nicht komplex ist, wird er aus den Performanzgründen im File cgi.c geschrieben (sonst musste er als unabhängiger Task gestartet werden).
- Abfrage der Sensorwerte.
- Ausgabe der Sensorwerte.
- Ausgabe des Temperaturwertes auf der Sieben-Segment-Anzeige
- Neuladen der Testseite in jeder Sekunde, damit immer die aktuellsten Sensorwerte ausgegeben werden.

Dies wird auf der Abbildung 7.1 auf der nächsten Seite dargestellt:

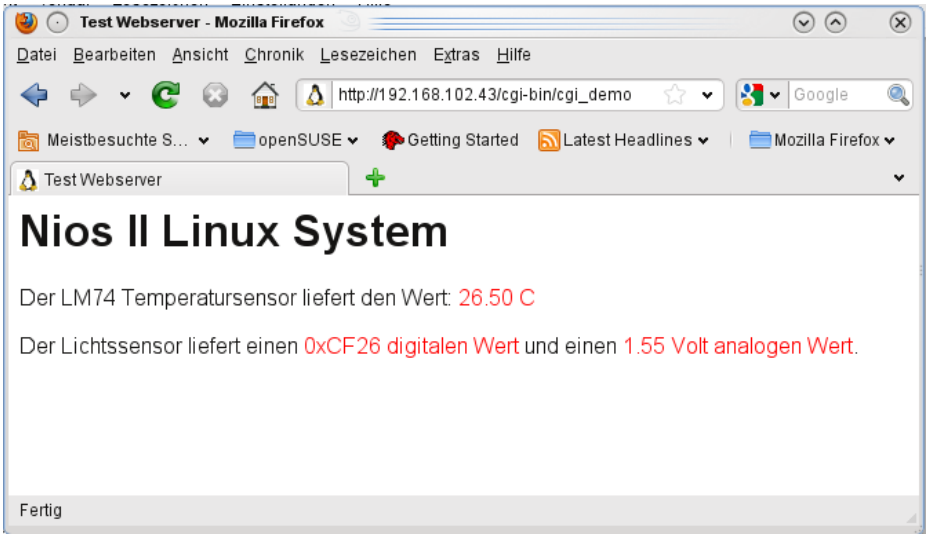

Abbildung 7.1: Testwebserver

Für die Kommunikation zwischen Nios II System und Sensoren war eine SPI-Schnittstelle nötig. Um die von den Sensoren gelieferten Ergebnisse auf dem Webserver auszugeben, war eine Ethernetschnittstelle nötig. In diesem Fall handelt sich um einen „Triple-Speed-Ethernet“, der im nächsten Abschnitt vorgestellt wird.

Laut dem IEEE (Institute of Electrical and Electronics Engineers), das für die Ethernetstandards verantwortlich ist, ist **Ethernet** eine Netzwerktechnik, die überwiegend in lokalen Netzwerken aber auch zur Verbindung großer Netzwerke eingesetzt wird. Seitdem eine Arbeitsgruppe für eine Technik für lokale Netzwerke eingerichtet wurde, ist der Name Ethernet das Synonym für alle unter der Arbeitsgruppe 802.3 vorgeschlagenen und standardisierten Spezifikationen.
Ethernet ist eine paketvermittelnde Netzwerktechnik. Die Adressierung und die Zugriffskontrolle finden auf der Schicht 1 und 2 des OSI-Schichtenmodells statt. In Form von Pakten kommen die Daten von darüber liegenden Schichten (TCP/IP). Diese Datenpakete werden um einen Header und eine Prüfsumme erweitert und danach übertragen. Wegen dieser Technik der Datenübertragung breitet Ethernet Probleme bei den zeitkritischen Anwendungen. Es besteht keine Garantie, dass die Daten innerhalb einer bestimmten Zeit den Empfänger erreichen. [22]

Altera® bietet eine IP-Core - Triple-Speed-Ethernet. Sie ermöglicht die Lösungen mit Netzwerkunterstützung zu bauen. TSE besteht aus 10/100/1000-Mbit/s Ethernet MAC (Media Access Controller) und PCS7 IP. Diese IP-Core ermöglicht, dass ein Altera® FPGA an einem externen PHY (in diesem Fall National DP83640 PHY) angeschlossen werden kann und damit eine Schnittstelle zu einem Netzwerk darstellt.
Allgemein, MAC kontrolliert die Zugriffe auf das Übertragungsmedium und bildet eigentliche Schnittstelle zum Medium. Die Schnittstelle zwischen MAC und PHY (physikalische Schnittstelle) ist standardisiert und bildet das so genannte Media Independed Interface MII. Der MAC befindet sich im Adress- und Datenbus des Prozessors und wird von diesem gesteuert.
Die wichtigste Eigenschaften vom Alteras Triple Speed Ethernet sind:

- 10/100/1000 Mbps Ethernet MAC im halb-Duplex und voll-Duplex Modus
- 10/100 Mbps oder 1000 Mbps kleiner MAC
- Multi-Kanal MAC
- kann MegaWizard® Schnittstelle benutzen
- Unterstützung für OpenCore Plus
- Unterstützung für Virtual Local Area Network (VLAN)

Mehr Informationen über Triple Speed Ethernet und seine Konfigurationsmöglichkeiten und Einstellungen können in [23] gefunden werden.

8 Fazit

Das Endergebnis dieser Arbeit wird in drei großen Schritten erreicht:

1. Hardwaredesign
2. Portierung des Mikrocontroller-Linux
3. Softwaredesign

Beim ersten Schritt wird die Quartus II 9.0 Software von Altera® eingesetzt. Der Akzent wird auf das SOPC Builder Tool gesetzt. Mit diesem Softwaretool werden die Hardwarekomponenten ausgewählt und die Verbindungen zwischen ihnen hergestellt. Die potentielle Problemstelle ist die Verbindungsherstellung. Vom SOPC Builder werden nicht alle Verbindungen des Hardwaresystems hergestellt, deswegen müssen diese manuell gesetzt werden (z. B. Verbindung zwischen Nios II Prozessor und UART-232-Schnittstelle).
Nach dem die Aufgabe des SOPC Builder Tools erfolgreich abgeschlossen wurde, bekommt man eine graphische Darstellung des Hardwaresystems. Alle Hardwarekomponenten sind mit ihren Ein- bzw. Ausgängen an dieser Graphik vorhanden. Jeder dieser Ein- bzw. Ausgänge muss ein bestimmter Pin graphisch zugeordnet werden. Falls ein falscher oder kein Pin zugeordnet wird, treten die Probleme bei dem Compilierungsprozess auf. Nachdem der Compilierungsprozess fehlerfrei abgeschlossen wurde, kann die Konfigurationsdatei des Hardwaresystems auf das DBC3C40 Entwicklungsboard geladen werden. Dies erfolgt mit Hilfe des „Programer“ Tools von Quartus II 9.0.

Der zweite Schritt umfasst eine Vorbereitung für die Portierung des uClinux und die eigentliche Portierung. Die Vorbereitung erfolgt durch die Installation einer Desktopversion des Linux – Open SUSE, und durch die Installation der Quartus II 9.0 und Nios 2 IDE Software. Problematisch war die Quartus II 9.0 Software. Nach der Installation konnte sie nicht gestartet werden. Dieses Problem wurde mit der Kommandozeile „xhost +local:root“ beseitigt.
Die Portierung des uClinux war kein einfacher Prozess. Der Treiber für die Altera Triple Speed Ethernet-Schnittstelle hat viele Probleme gemacht. Es hat sich heraus-

gestellt, dass der Speicher für diesen Treiber nicht korrekt reserviert wurde, und dass weder die MAC- noch die Netzmaskadresse gesetzt wurden. Nach dem diese Probleme behoben wurden, konnte das Image erfolgreich gebootet werden.

Schritt drei befasst sich mit der Treiberentwicklung für den LM74 Temperatur- bzw. TSL250R Lichtsensor. Da der Lichtsensor extern über den A/D-Wandler angeschlossen ist, musste die SPI-Schnittstelle für die A/D-Wandler – Kommunikation eingesetzt wurde. Da die von Altera angebotene SPI-Schnittstelle nicht wie gewünscht funktionierte, kam eine simulierte SPI zum Einsatz. Diese zwei Treiber werden als zwei Tasks asynchron vom uClinux bearbeitet.
Der letzte Schritt des Softwaredesigns ist die Datenbereitstellung der von den Sensoren gelieferten Ergebnisse über den Boa Webserver. Mit einem Webbrowser sind diese Ergebnisse anzuschauen.
Das Endergebnis dieser Arbeit ist in der folgenden Abbildung dargestellt:

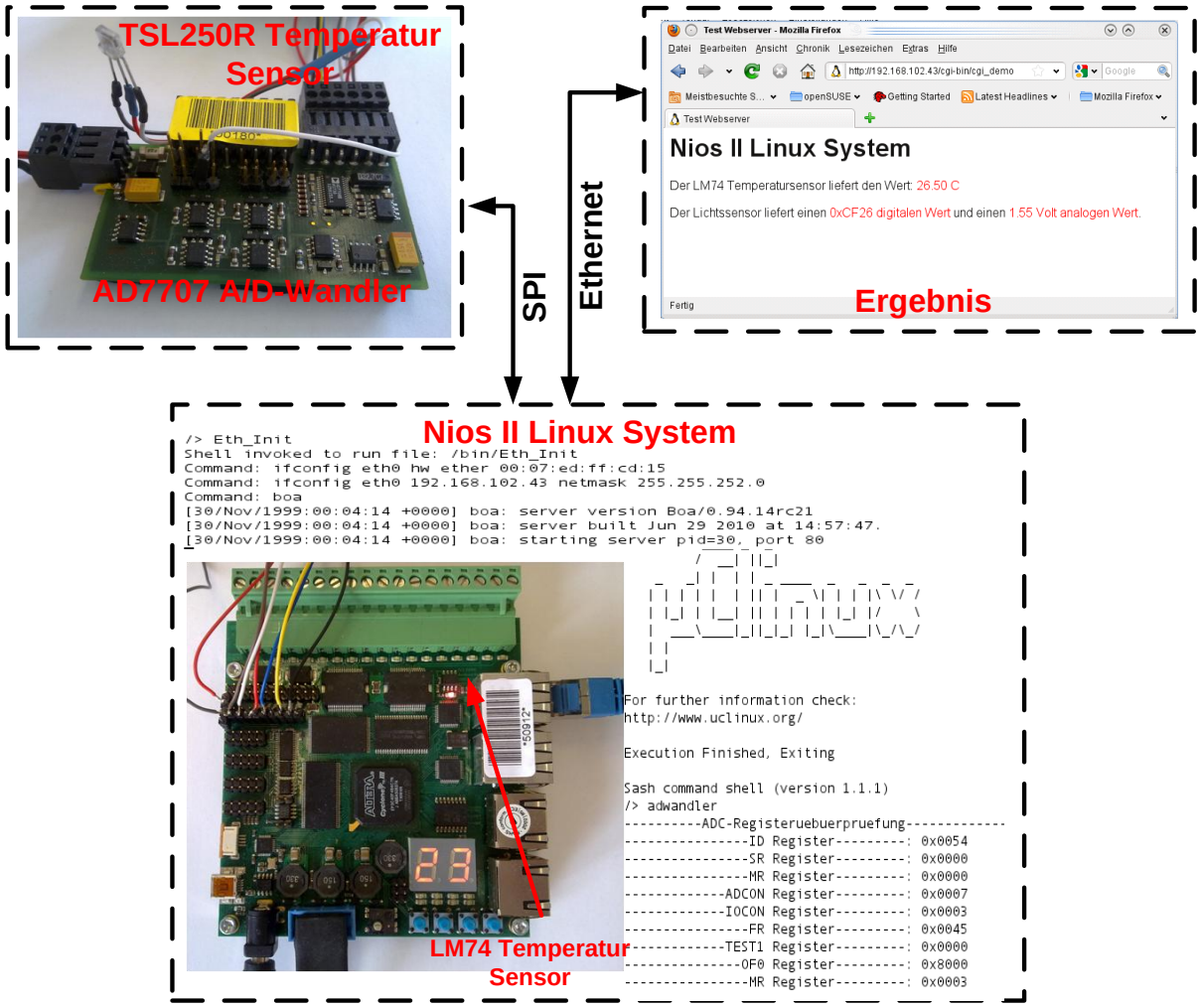

Bild 8-1: Das Endergebnis

Das gewünschte Ziel wurde erreicht. Es öffnet die Tür für weitere Entwicklungen. Das Nios II System mit der Sensorik könnte um einen Aktor erweitert werden, um ein Nios II Sensor-Aktor System zu bekommen (z. B. im Fall dass die Temperatur über 50 Grad ist, sollte ein Alarm eingeschaltet werden). Bei so einem Sensor-Aktor System wäre der Einsatz einer CAN-Busschnittstelle geeignet.
Nicht alle Hardwarekomponenten des DBC3C40 Entwicklungsboards wurden angesprochen. Eine wurde schon genannt, nämlich der CAN-Bus. Für die Ausgabe der Sensorergebnisse könnte auch ein TFT-Controller eingesetzt werden. Die Entwicklung einer GUI (**G**raphical **U**ser **I**nterface) für die Sensorergebnisse wäre auch sinnvoll.

9 Anhang

9.1 Literaturverzeichnis

[1] *Modellierung integrierter Schaltungen* (Skript zur Vorlesung „Ausgewählte Kapitel der Rechnerarchitektur – Josef Börcsök

[2] Altera. *Logic Elements and Logic Array Blocks in the Cyclone III Device Family*, Dezember 2009

[3] Altera. *Memory Blocks in the Cyclone III Device Family*, Dezember 2009

[4] Altera. *Embedded Multipliers in the Cyclone III Device Family*, Dezember 2009

[5] Altera. *Clock Networks and PLLS in the Cyclone III Device Family*, Dezember 2009

[6] Altera. *I/O Interfaces*, Januar 2010

[7] Altera. *Nios II Processor Reference Handb*ook, November 2009

[8] Altera. *Programming Model*, November 2009

[9] Altera. *Avalon Interface Specifications*, April 2009

[10] Altera. *System Interconnect Fabric for Memory-Mapped Interfaces*, November 2009

[11] Altera. *EPCS64 Configuration Device*, März 2005

[12] National Semiconductor. *DP83640 Precision PHYTER IEEE 1588*, Januar 2009

[13] Altera. *Quartus II Handbook Version 10.0, SOPC Builder*, Juli 2010

[14] Nios Wiki, *Quartus for Linux* http://www.nioswiki.com/OperatingSystems/UClinux/QuartusforLinux

[15] Nios Wiki, *Install Nios2Linux*
http://www.nioswiki.com/InstallNios2Linux

[16] Nios Wiki, *uClinux-Distribution*
http://www.nioswiki.com/OperatingSystems/UClinux/UClinuxDist

[17] National Semmiconductor. *SPI MICROWIRE™ 12-Bit Plus Sign Temperature Sensor*, Juni 200

[18] TAOS. *TSL250R Light to Voltage Sensor*, September 2007

[19] Analog Devices. *AD7708/AD7718*, 2001

[20] Nios Wiki, *Compile Hello*
http://www.nioswiki.com/OperatingSystems/UClinux/CompileHello

[21] Wikipedia, *Boa Webserver*
http://de.wikipedia.org/wiki/Boa_%28Webserver%29

[22] IEEE 802.3, Ethernet
http://www.ieee802.org/3/

[23] Altera. *Triple Speed Ethernet MegaCore*, November 2009

[24] *Processor Design, System-On-Chip Computing for ASICs and FPGAs* – Jari Nurmi; Springer, April 2007

[25] *Digital system designs and practices: using Verilog HDL and FPGA* – Ming-Bo Lin; Wiley, 2008

[26] *High-speed digital system design: a handbook of interconnect theory and design practices* – Stephen Hall; Wiley, 2008

[27] *Einführung in den Prozessorentwurf: von der Planung bis zum Prototyp* – Dieter Wecker; Renningen: expert-Verl., 2008

[28] *Embedded Linux, Praktische Umsetzung mit uClinux* – Thomas Brinker, Heiko Degenhardt, Gerald Kupris; VDE Verlag, 2007

9.2 Abbildungsverzeichnis

9.3 Tabellenverzeichnis

9.4 Abkürzungsverzeichnis

SoC – System-on-Chip
FPGA – Field Programmable Gate Array
SCP – Soft-Core-Processor
HDL – Hardware Description Language
SOPC – System-On-Programmable-Chip
IP-Core – Intelectual Property Core
PWM – Pulsweitenmodulator
VHDL – Very High Speed Integrated Circuit Hardware Description Language
ASIC – Application Specific Integrated Circuit
LUP – Look-up-Table
PLL – Phase Locked Loop
RAM – Random Access Memory
TSMC - Taiwan Semiconductor Manufacturing Company
LP – Low Power
LE – Logic Element
LAB – Logic Array Block
ROM – Read Only Memory
GCLK – Global Clock
IOE – Input Output Element
VREF – Voltage Reference
LVDS – Low Voltage Differential Signaling
BLVDS – Bus Low Voltage Differential Signaling
RSDS – Reduced Swing Differential Signaling
PPDS – point-to-point differential signaling
RISC – Reduced Instruction Set Computer
CPLD – Complex Programmable Logic Device
ALU – Arithmetic Logic Unit
MMU – Memory Management Unit
MPU – Memory Protection Unit
GPR – General Purpose Register
IRQ – Interrupt Quelle
MM – Memory Management
TCM – Tightly Coupled Memory
SIB – System-Interconnect-Fabric
PIO – Parallel Input Output
MAC – Media Access Controll
uClinux – MicroController Linux
ADC – Analog Digital Converter
(V)LAN – (Virtual) Local Area Network

Eth_Init

```
ifconfig eth0 hw ether 00:07:ed:ff:cd:15
ifconfig eth0 192.168.102.43 netmask 255.255.252.0
boa
```

LM74_Temperatursensor.c

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>

#include "altera_avalon_pio_regs.h"       // PIO access
#include "custom_nios2.h"
#include "../adwandler/adwandler.h"

//defines for Temperatursensor
#define lm74_dq  0x01  // LM74 Signal definition
#define lm74_clk 0x02  // LM74 Signal definition
#define lm74_cs  0x04  // LM74 Signal definition

/*****************************************************************************/
// LM74 Temperature Sensor
/*****************************************************************************/
float read_temp (int base)
{
  alt_u32 temp;
  float celsius;

   // DQ = In, CLK, CS = OUT
IOWR_ALTERA_AVALON_PIO_DIRECTION(base,0x06);
IOWR_ALTERA_AVALON_PIO_DATEN(base,0xf);
```

```
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_cs);
IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK1
temp = IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01;

IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK2
temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK3
temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK4
temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK5
temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK6
temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK7
temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK8
temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK9
temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK10
temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);
```

```
  IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
  IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK11
  temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

  IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
  IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK12
  temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

  IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
  IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK13
  temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

  IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
  IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK14
  temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

  IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
  IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK15
  temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

  IOWR_ALTERA_AVALON_PIO_DATEN(base,0);
  IOWR_ALTERA_AVALON_PIO_DATEN(base,              lm74_clk);   //CLK16
  temp = (temp << 1) | (IORD_ALTERA_AVALON_PIO_DATEN(base) & 0x01);

  IOWR_ALTERA_AVALON_PIO_DATEN(base,        lm74_cs + lm74_clk);

  celsius = (temp >> 3) * 0.0625;
  return(celsius);
}
```

```
//set the temperature sensor value on the seven segment display
void sevenseg_set_dec(int dec)
{
	alt_u16 Daten = 0;
	static alt_u8 segments[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D,
	0x07, 0x7F, 0x6F}; /* 0-9 */
	Daten = segments[dec/10] | (segments[(dec%10)] << 7);
	IOWR_ALTERA_AVALON_PIO_DATEN(na_seven_seg_pio, Daten);
}
```

adwandler.h

```
//----------------------------------ADC Registers-------------------------------------------------------
#define SR	0x01		//Statusregister

#define MR		0x02		/* Moderegister. Dieses Register konfiguriert den
Oprationmodus vom AD7708/AD7718 */

#define ADCON	0x03	/* Kontrollregister. Dieser Register wird benutzt zur
Kanalselektierung, und zur uni/bipolar Kodierung */

#define IOCON	0x04	/* Dieser Register wird benutzt, um den I/O Port zu konfigu-
rieren und zu kontrollieren */

#define FR	0x05		//Filterregister

#define DATEN	0x06	/* Datenregister, das Ergebnis der AD Wandlung wird in
diesem Register gespeichert */

#define OF0	0x07		//Offset-Calibration-Coefficient-Register

#define GNO	0x08	//Gain-Calibration-Coefficient-Register.

#define  ID	0x09		//ID Register.

#define TEST1	0x0C	//Testregister.
//-------------------------------------------------------------------------------------------------------
#define RD	0x01
#define WR	0x00
```

```
//---------------------------------------Funktionen-----------------------------------------------
void wait();
void set_DIN(char val);
void set_CLK(char val);
void set_CS(char val);
void set_RESET(char val);
void COMREG_implement(char rw, char reg);
void ADC_write_to_reg (char reg, int val);
int ADC_read_from_reg (char reg);
```