

BTNode FPGA

Mobiler FPGA mit Bluetooth Kommunikation

Peter Fercher



DIPLOMARBEIT
DA-2003.08

Wintersemester 2002/2003

Betreuer: Matthias Dyer
Jan Beutel
Professor: Lothar Thiele

Zusammenfassung

An der ETH wurde ein kleiner und mobiler Sensorknoten entwickelt - der BTnode. Sein Mikrokontroller sorgt zusammen mit dem Bluetooth Modul für die Anbindung an die Aussenwelt. Tatsache ist aber, dass es dem BTnode für viele Anwendungen an Rechenleistung fehlt.

Deshalb wurde im Rahmen dieser Arbeit ein kleines und mobiles FPGA Board entwickelt - der BTnode_FPGA. Er verschafft dem BTnode die nötige Rechenpower. Damit ist der BTnode in der Lage, rechenintensive Signalverarbeitung zu betreiben.

Im folgenden erkläre ich alle Schritte die nötig waren um den BTnode_FPGA zu realisieren. Vom Ausarbeiten des Konzepts über das Hardware Design bis hin zur Bestückung, der Inbetriebnahme und Test des Boards. Auch die implementierte Software zur Ansteuerung wird erläutert. Eine Demo Applikation rundet das ganze ab, und zeigt dass die gestellten Anforderungen an die Hardware erfüllt werden konnten.

Abstract

At the ETH Zurich, a small and mobile sensornode has been developed - the BTnode. Its microcontroller connects to the world using a Bluetooth module. Fact is that the BTnode lacks computing power for many applications.

The goal of this thesis was to develop an FPGA extension board for the BTnode - The BTnode_FPGA. It provides sufficient computing power to the BTnode for digital signal processing applications.

This report guides through all the steps of realising the FPGA module. From elaborating a concept, hardware design, assembly, to initial operation and testing of the board. The implemented software is also explained. A demo application shows that the hardware constraints could be met.

Inhaltsverzeichnis

1	Einführung	4
1.1	Motivation und thematischer Hintergrund	5
1.1.1	BTnode	6
1.2	Ziele der Arbeit	7
1.2.1	BTnode_FPGA	7
1.2.2	Aufgabenstellung	7
2	Anforderungen	14
2.1	Low-Power	14
2.2	Mobilität	14
2.3	FPGA Konfiguration über Bluetooth	14
3	Konzept FPGA-Modul	15
3.1	Designentscheidungen	15
3.1.1	Wieso ein Flash Memory?	15
3.1.2	Wieso ein CPLD?	16
3.2	Szenarien	16
3.2.1	Neue Konfiguration ins Flash schreiben	17
3.2.2	Daten aus Flash Lesen	17
3.2.3	(Re)Konfiguration des FPGAs	18
3.3	Konfigurations Konzept und BurchED B5-X300 Board Evaluation	18
4	Umsetzung in Hardware/Software	19
4.1	Komponentenauswahl	19
4.1.1	FPGA	19
4.1.2	CPLD	19
4.1.3	FLASH	20
4.1.4	SRAM	20
4.1.5	Power Regulators und Stromversorgung	20
4.2	Oszillator	21
4.3	Part List PCB	21
4.4	HW-Design	23
4.4.1	Part Developer	23
4.4.2	Footprints	23
4.4.3	Schema	23
4.4.4	Layout	24
4.4.5	Einholen von Offerten	25
4.5	SW-Design	25
4.5.1	VHDL für CPLD und FPGA	26
4.6	Inbetriebnahme und Test	30
4.7	Bestückung PCB	30
4.8	Kabel	30
4.9	Funktionalitätstests mit VC++	32
4.10	Demoanwendung	35
4.11	Bandbreiten und Bottleneck	35
4.12	Power Berechnungen	36
5	Schlussfolgerung	37
6	Ausblick	37

A Schemas und Layers	38
A.1 BNode	38
A.2 BNode_FPGA Schemas	38
A.3 BNode_FPGA Layers	38

1 Einführung

In meiner ersten Semesterarbeit beschäftigte ich mich mit der Implementierung des Bluetooth Protokollstacks auf einem 8-Bit Mikroprozessor. In der zweiten Semesterarbeit implementierte ich einen Sigma-Delta Audio D/A Wandler auf einem Virtex FPGA. Mit den gesammelten Erfahrungen aus diesen Arbeiten, und weil ich der Meinung bin das ein Elektoringenieur mindestens einmal in seinem Leben Hardware selber entwickeln sollte ;-) hab ich diese Diplomarbeit angenommen. Dies war mein erstes - und sicher nicht mein letztes - Hardware-Design! Ich konnte in den 4 Monaten sehr viel lernen. Da wäre zum einen das Konzeptuelle: Was soll die Hardware können? Wie erreiche ich die gesetzten Designziele und Spezifikationen? Herausfinden welcher Hersteller bei welchem Distributor ist, kriege ich Samples und überhaupt: Was mache ich, wenn der Disti das Teil dann doch nicht liefern kann - oder nur zu 10000 Stück? Wichtig war auch: Akzeptieren von Zeitlimits und das Erkennen von persönlichen Grenzen. Das coolste Teil nützt nichts wenn es nicht lieferbar ist und: Ich bin keine Maschine! Am liebsten wäre ich rund um die Uhr an der Arbeit gewesen, doch musste ich erkennen das das nicht möglich war. Während der ganzen Diplomarbeit konnte ich weitgehend selbstständig arbeiten. Es stand mir viel Handlungsspielraum zur Verfügung. Wichtige Designentscheidungen konnte ich mit den Betreuern Matthias und Jan besprechen. An dieser Stelle möchte ich folgenden Leuten danken:

Meinen Betreuern, **Matthias Dyer** der mir bei FPGA und Konzeptuellen Entscheidungen half und **Jan Beutel**, von dessen Board-Design und Hardware Erfahrung ich profitieren konnte. Weiterhin möchte ich dem **Institut TIK** und insbesondere **Professor Lothar Thiele** für die Ermöglichung und Finanzierung der Arbeit danken. Ein grosses Merci auch an die **Mitarbeiter vom TIK** und die **Mitstudenten im G69**. Merci für die gute Zeit!

Peter Fercher

1.1 Motivation und thematischer Hintergrund

Mobile Sensorknoten sind heutzutage Gegenstand von Forschung und Entwicklung. Einen solchen Sensorknoten zeigt Abbildung 1. Er besteht aus einem Sensor, einem Kommunikationsteil und einer Recheneinheit. Der Kommunikationsteil dient der Anbindung an die Aussenwelt. Der Recheneteil verarbeitet und komprimiert Daten die der Sensor liefert. Anwendungsgebiete von solchen mobilen Sensorknoten sind Wearable Computing und Sensor Netzwerke. Anwendung in diesen Bereichen finden auch die an der ETH entwickelten BTnodes [1].

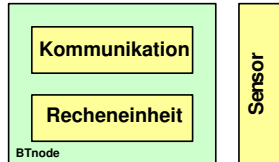


Abbildung 1: Aufbau eines Sensorknotens

Wearable Systems sind eingebettete Systeme. Beispiele dafür sind Sensoren wie Beschleunigungsmesser, Mikrofone und Displays eingebettet in unsere Kleidung. Auch die Energieerzeugung soll in die Kleidung eingebaut werden. Koordiniert wird alles von einer Hauptrecheneinheit.

Drahtlose Kommunikation ist die bevorzugte Verbindungsart zwischen Sensorknoten und Hauptrecheneinheit. Um möglichst wenig Daten von den Sensoren übertragen zu müssen, und somit die energieaufwendige drahtlose Kommunikation reduzieren zu können, werden Verfahren wie Kompression etc. idealerweise direkt in den Sensorknoten integriert. Zugriff auf externe Netzwerke kann durch bestehende Technologien wie Wireless LAN oder Bluetooth erreicht werden.

Field Programmable Gate Arrays (FPGAs) und General Purpose CPUs (GP CPUs) haben beide ihre Vorzüge. Das Anwendungsfeld von FPGAs liegt in rechenintensiven Aufgaben die möglichst wenig Kontrollfluss benötigen. Diese können von FPGAs schneller und mit weniger Energiebedarf gelöst werden als von GP CPUs. Paradebeispiele finden sich zu hauf in der digitalen Signalverarbeitung. Alle heutigen multimedia Applikationen benötigen Verfahren zur (De)Komprimierung. Die Idee der Kombination von GP CPU und FPGA liegt darin, die Vorzüge der beiden Architekturen gleichzeitig auszunützen. Der FPGA nimmt der CPU die rechenintensiven Aufgaben wie Audiodaten samplen ab. Im Gegenzug können auf einer CPU kontrollintensive Aufgaben wie Kommunikation einfacher implementiert werden. So kann eine highspeed CPU durch eine leistungsschwächere CPU und einen FPGA ersetzt werden. Wird dieses Konzept richtig angewandt, kann man Aufgaben schneller und mit weniger Energiebedarf lösen.

Ein weiterer Vorteil von FPGAs gegenüber Application Specific Integrated Circuits (ASICs): Sie sind rekonfigurierbar. Man spricht auch von ASICs-On-Demand. Also Anwendungsspezifische Integrierte Schaltungen die für die jeweilige Aufgabe konfiguriert werden können. Diese ermöglicht es, baugleiche Knoten in Sensornetzwerken für verschiedene Anwendungsbereiche zu verwenden. Der Knoten wird lediglich an einen anderen Sensor angehängt. Die Rekonfigurierbarkeit von FPGAs ermöglicht auch eine viel kürzere und billigere Entwicklung als die für statische Hardware (ASICs).

1.1.1 BTnode

Am TIK wurde im Rahmen der Projekte Terminodes [2] und Smart-ITs [3] der kleine und mobile BTnode [1] entwickelt (Abbildung 2). Er besteht aus einem Bluetooth-Modul und einem Mikrokontroller. Der Mikrokontroller wird einerseits für die höheren Schichten des Bluetooth-Stacks benötigt, kann aber andererseits auch für einfache Benutzerprogramme genutzt werden. Tatsache ist, dass die Rechenkraft und die Speicherkapazität sehr beschränkt sind.

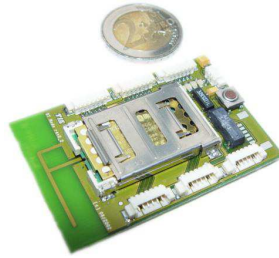


Abbildung 2: BTnode Rev. 2.2 (61x41mm) von Jan Beutel

1.2 Ziele der Arbeit

1.2.1 BTnode_FPGA

Für rechenintensive Aufgaben schreitet der BTnode geradezu nach einer FPGA-Erweiterung. Damit wird es möglich, ihn auch als rechenintensiven Sensorknoten einzusetzen. Eine solche Erweiterung (den BTnode_FPGA) durfte ich im Rahmen dieser Diplomarbeit am TIK entwickeln (Abbildung 3).

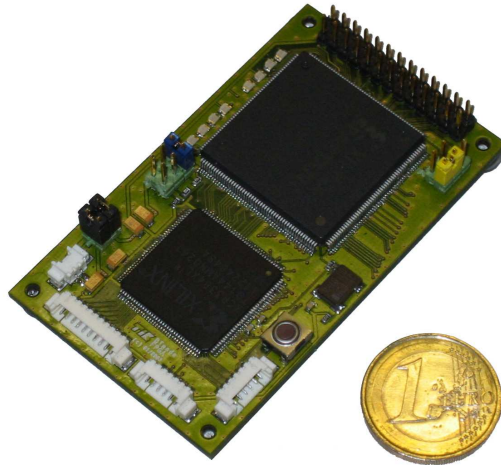


Abbildung 3: BTnode_FPGA Board (77x46mm)

1.2.2 Aufgabenstellung

DA-2003-8: Mobiler FPGA mit Bluetooth Kommunikation

Matthias Dyer

Institut für Technische Informatik und Kommunikationsnetze
ETH Zürich

21. Oktober 2002

Betreuer: Matthias Dyer <dyer@tik.ee.ethz.ch>
Jan Beutel <beutel@tik.ee.ethz.ch>
Student: Peter Fercher <pfercher@ee.ethz.ch>
Dauer: 21. Oktober 2002 - 28. Februar 2003

1 Thematischer Hintergrund

Jüngste Fortschritte in drahtloser Kommunikation und Elektronik haben die Entwicklung von preisgünstigen, low-power, multifunktionalen Sensorknoten ermöglicht, welche ungebunden über kurze Distanzen miteinander kommunizieren. Diese kleine Sensorknoten, bestehend aus Sensoren, Datenverarbeitung und Kommunikationskomponenten, werden in sog. Sensornetzwerken eingesetzt. Sensornetzwerke bestehen aus einer grossen Anzahl Sensorknoten, welche dicht zueinander in oder in unmittelbarer Nähe des Phänomens ausgesetzt werden. Eine Eigenschaft von Sensornetzwerken ist, dass die Sensorknoten eine gemeinsame Aufgabe erledigen.

Sensorknoten sind mit einem on-board Prozessor bestückt. Statt die rohen Sensordaten zu senden benutzen sie ihre Rechenkraft um lokal einfache Datenverarbeitungen durchzuführen um dann nur die notwendigen und vorprozessierten Daten senden zu müssen. Dies ist sehr effizient, da die drahtlose Kommunikation, welche energieaufwändig ist, reduziert werden kann.

Es hat sich gezeigt, dass trotz allem Fortschritt in der Computerarchitektur einige Applikationen nach noch mehr Rechenleistung verlangen. Dazu gehören besonders Anwendungen aus dem Bereich Kryptographie und Multimedia, wie Audio-, Video- und Bildverarbeitung. Um für diese Algorithmen die benötigte Rechenleistung bereitzustellen, greift man meist auf Applikationsspezifische Integrierte Schaltungen.

gen (ASICs) zurück, die in den Sensorknoten integriert werden um eine bestimmte Aufgabe zu beschleunigen.

Eine Möglichkeit für die Realisierung von anwendungsspezifischer Hardware sind Field-Programmable Gate Arrays (FPGAs). Die Rekonfigurierbarkeit dieser Bausteine erlaubt nicht nur für verschiedene rechenintensive Anwendungen ein und dieselbe Hardware zu verwenden, sie ermöglicht auch eine viel kürzere und billigere Entwicklung als die für statische Hardware.

Warum ein FPGA ?

Leistungsfähige FPGAs haben im Vergleich zu klassischen Mikrocontrollern und Signalverarbeitungsprozessoren (DSPs) einen relativ hohen Stromverbrauch. Es wurde jedoch gezeigt, dass vor allem bei Programmen von Eingebetteten Systemen, schon mit heutigen Bausteinen ein rekonfigurierbares Array deutliche Einsparungen bringen kann [3]. Solche Programme verbringen einen Grossteil ihrer Rechenzeit in kleinen einfachen Schlaufen, welche in Hardware um mehrere Grössenordnungen schneller ablaufen als in Software.

Abgesehen vom Energieverbrauch bringt ein FPGA einschlägige Vorteile für einen Sensorknoten:

- **Rechenleistung:** Heutige FPGAs erreichen ASIC-ähnliche Rechenleistungen für datenpfad-orientierte Anwendungen und übertreffen somit für nicht-floating-point Algorithmen auch die schnellsten DSPs.
- **Flexibilität:** Im Gegensatz zu ASICs sind FPGAs rekonfigurierbar. Durch partielle Rekonfiguration können auch nur gewisse Teile der FPGA Konfiguration geändert werden.
- **Multifunktionalität:** Ein FPGA kann eine Vielzahl von Anwendungen ausführen. Es existieren verschiedene Komponenten (IP-Cores) für FPGAs, sowohl als freie Designs als auch kommerzielle Produkte, welche in das eigene Design integriert werden können. Die Auswahl umfasst unter anderem diverse Arithmetik-Kernel, Digitale Filter, En-/Decoders, verschiedene Kommunikations-Interfaces, Krypto-Cores, sowie komplette CPUs.
- **I/O:** Ein FPGA hat eine grosse Anzahl benutzerprogrammierbare Input-/Output-Pins. Somit kann der Sensorknoten mit mehreren, auch komplexen Sensoren bestückt werden. Mit dem Sigma-Delta Prinzip können auch ohne grossen zusätzlichen Hardwareaufwand gute Analog-Digital- und Digital-Analog-Wandler realisiert werden.

- **Parallelität:** Ein FPGA kann Teile von Algorithmen oder ganze Anwendungen parallel zueinander ausführen. In einem Sensor-knoten könnte somit ein FPGA mehrere Sensoren gleichzeitig unterhalten (Grenzwerte überwachen, Vorprozessierung, Filterung, usw.).

Anhand zweier Beispiele kann man diese Vorteile aufzeigen. Diese Beispiele sind als mögliche Anwendungen zu verstehen.

Bsp. 1: Künstliche intelligente Maus

Forscher, die sich mit Künstlicher Intelligenz beschäftigen, entwickeln oft kleine autonome Roboter, mit welchen sie ein gewisses Verhalten zeigen oder erforschen wollen. Die *künstliche Maus* ist solch ein Roboter, basierend auf dem mobilen Roboter Khepera. Als Fühler hat diese Maus 19 Schnurrbarthaare, implementiert je mit einem Mikrophon und einem Haar. Um die Daten von den Mikrophonen verarbeiten zu können, werden zur Zeit noch 19 analoge Leitungen von der Maus zu einer Analog-Digital-Wandler Karte für den PC geführt.

Ein mobiler und mit einem FPGA bestückter Sensorknoten könnte direkt auf der Maus die 19 analogen Signale parallel erfassen, vorprozessieren, verpacken und über Funk zum PC übertragen, oder gar selber auf einer Compact-Flash Karte abspeichern.

Bsp. 2: Acoustic Ranging

Ein anderes Beispiel ist die Positionsbestimmung, welche für verschiedene Anwendungen erforderlich ist. Es gibt verschiedene Methoden für die Positionsbestimmung, wovon *Acoustic Ranging* eine ist. Mittels Ultraschall wird von einem Knoten eine charakteristische Sequenz gesendet und dann von einem anderen Knoten empfangen. Die Distanz kann dann über die gemessene Flugzeit der Schallwellen berechnet werden. Um die charakteristische Sequenz genau zu detektieren, werden in der Signalverarbeitung typischerweise Filter und Kreuzkorrelationen angewendet.

Solche Signalverarbeitungsalgorithmen sind gut geeignet für FPGAs. Das ganze Lokalisierungs-Subsystem (mit Triangulation) kann parallel und unabhängig vom Rest in einem Teil des FPGAs ablaufen. Ausserdem können weitere Sensoren zur Verbesserung der Genauigkeit (Temperatur, Feuchtigkeit) einfach integriert werden. Das Subsystem könnte von einem Soft-Mikrokontroller (im FPGA) angesteuert werden.

2 Problemstellung

An unserem Institut wurde im Rahmen der Projekte Terminodes und Smart-Its der kleine und mobile *BTNode rev.2_2* [1] entwickelt. Er besteht aus einem Bluetooth-Modul und einem Mikrokontroller. Der Mikrokontroller wird einerseits für die höheren Schichten des Bluetooth-Stacks benötigt, kann aber andererseits auch für einfache Benutzerprogramme genutzt werden. Tatsache ist, dass die Rechenkraft und die Speicherkapazität sehr beschränkt sind.

Das Ziel dieser Arbeit ist, den *BTNode* mit einem *FPGA-Modul* zu erweitern. Diese Erweiterung sollte die oben erwähnten Vorteile ermöglichen. Die Entwicklung des FPGA-Moduls kann folgende Punkte beinhalten (Machbarkeit/Implementation):

- **Low-Power:** Da der *BTNode* sowie das *FPGA-Modul* in den meisten Fällen von einer Batterie versorgt werden, sollte der Energieverbrauch so gut wie möglich reduziert werden. Dabei soll auch untersucht werden, ob es von Vorteil ist, wenn zwischendurch das ganze *FPGA* abgeschaltet werden kann. Natürlich müsste dann die *FPGA-Konfiguration* mit dem aktuellen Kontext automatisch neu geladen werden können. Um den Stromverbrauch des *FPGA-Moduls* im Betrieb messen zu können, sollte eine geeignete Speisung gewählt werden.
- **Konfiguration über Bluetooth:** Könnte man den Mikrokontroller des *BTNodes* sowie das *FPGA* über Bluetooth programmieren, würde das erheblich zur Flexibilität beitragen. Ausserdem würden interessante neue Forschungsarbeiten im Bereich *Reconfigurable Computing* ermöglicht werden. Damit über Bluetooth nicht immer eine ganze *FPGA-Konfiguration* versendet werden muss, sollte die partielle Rekonfiguration [9] ermöglicht werden.
- **FPGA:** Leistungsfähigkeit und Energieverbrauch, das sind zwei Kriterien, die gegenläufig sind. Für diese Arbeit soll durch eine Evaluation ein *FPGA* ausgesucht werden, der den Anforderungen am besten genügt. Dabei spielen auch Kriterien wie Entwicklungssoftware und Verfügbarkeit eine Rolle.

Die Diplomarbeit umfasst drei Gebiete. In einem ersten Schritt soll ein Konzept vom *FPGA-Modul* erarbeitet werden. Dieses beinhaltet auch die Evaluation verschiedener Hardware-Komponenten. Um mit der Anbindung des *FPGAs* an den *BTNode* und der *FPGA Konfiguration* zu experimentieren, steht das Entwicklungsboard *B5-X300* [2] der Firma Burch Electronic Design zur Verfügung. Dieses bietet einen Xi-

linx Spartan IIE XC2S300E¹ FPGA [4] und diverse Peripheriemodule. Im zweiten Schritt, sollte dieses Konzept dann in Hardware umgesetzt werden.

Im dritten Teil soll das FPGA-Modul in Betrieb genommen werden. Anhand einer Beispielanwendung kann das Funktionsprinzip demonstriert werden.

3 Teilaufgaben

1. **Evaluation FPGA:** Arbeiten sie sich in die Grundlagen von FPGAs ein und wählen sie einem geeigneten Baustein aus.
2. **Grundlagen BTNode:** Verschaffen sie sich einen Überblick über die BTNode Architektur und deren Anbindungsmöglichkeiten.
3. **Konzept Konfiguration:** Testen sie verschiedene Alternativen zur Konfiguration des BTNodes und des FPGAs [5, 6, 10, 11, 12]. Machen sie sich dazu auch mit dem BurchED B5-X300 Board vertraut.
4. **Konzept FPGA-Modul:** Erstellen sie ein Konzept des gesamten FPGA-Moduls, welches die oben erwähnten Punkte berücksichtigt.
5. **HW-Design:** Realisieren Sie Ihr Konzept in Hardware. Arbeiten Sie sich zuerst in die Grundlagen vom FPGA-Schaltungsentwurf [7, 8] ein und machen Sie sich mit den Design-Tools vertraut.
6. **Betrieb/Test:** Nehmen Sie das FPGA-Modul in Betrieb und testen Sie es zusammen mit dem BTNode.
7. **Anwendungen:** Demonstrieren Sie das FPGA-Modul mit einer geeigneten Anwendung.

4 Organisatorisches

- **Zeitplan** Erstellen Sie am Anfang Ihrer Arbeit zusammen mit dem Betreuer einen realistischen Zeitplan. Halten Sie Ihren Arbeitsfortschritt laufend fest.
- **Dokumentation** Dokumentieren Sie Ihre Arbeit sorgfältig. Legen Sie dabei besonderen Wert auf die Beschreibung Ihrer Überlegungen und Designentscheide.

¹Bedingt durch seinen relativ hohen Stromverbrauch, mag der Spartan IIE FPGA für das FPGA-Modul nicht besonders geeignet zu sein. Andere Bausteine verhalten sich aber in der Programmierung sehr ähnlich.

Literatur

- [1] Jan Beutel. Btnode rev2_2, July 2002. http://www.tik.ee.ethz.ch/~beutel/projects/btnode/btnode_rev2_2/btnode2_2.html.
- [2] Burch Electronic Design. *B5-X300 / B5-Spartan2e+ Quickstart*, August 2002. <http://www.burched.biz/tutorials/B5S2eQuickStart7Aug2002.pdf>.
- [3] J. Villarreal G. Stitt, B. Grattan and F. Vahid. Using on-chip configurable logic to reduce embedded system software energy. In *IEEE Symposium on Field-Programmable Custom Computing Machines, Napa Valley*, April 2002.
- [4] Xilinx. *Xilinx Spartan-IIE 1.8V FPGA Family*, v1.1 edition, June 2002.
- [5] Xilinx Inc. *Xilinx Application Note XAPP058: Xilinx In-System Programming Using an Embedded Microcontroller*, v3.0 edition, 1 2001. <http://www.xilinx.com/xapp/xapp058.pdf>.
- [6] Xilinx Inc. *Xilinx Application Note XAPP138: Virtex FPGA Series Configuration and Readback*, v2.4 edition, 7 2001. <http://www.xilinx.com/xapp/xapp138.pdf>.
- [7] Xilinx Inc. *Xilinx Application Note XAPP450: Power-On Current Requirements for the Spartan-II and Spartan-IIE Families*, v1.0 edition, 1 2001. <http://www.xilinx.com/xapp/xapp450.pdf>.
- [8] Xilinx Inc. *Xilinx Application Note XAPP451: Power-Assist Circuits for the Spartan-II and Spartan-IIE Families*, v1.0 edition, 1 2001. <http://www.xilinx.com/xapp/xapp451.pdf>.
- [9] Xilinx Inc. *Xilinx Application Note XAPP290: Two Flows for Partial Re-configuration: Module Based or Small Bit Manipulations*, v1.0 edition, 5 2002. <http://www.xilinx.com/xapp/xapp290.pdf>.
- [10] Xilinx Inc. *Xilinx Application Note XAPP501: Configuration Quick Start Guidelines*, v1.3 edition, 6 2002. <http://www.xilinx.com/xapp/xapp501.pdf>.
- [11] Xilinx Inc. *Xilinx Application Note XAPP502: Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode*, v1.1 edition, 6 2002. <http://www.xilinx.com/xapp/xapp502.pdf>.
- [12] Xilinx Inc. *Xilinx Application Note XAPP632: Programming an FPGA via E-mail*, v1.0 edition, 5 2002. <http://www.xilinx.com/xapp/xapp632.pdf>.

2 Anforderungen

Ein mobiles eingebettetes System stellt spezielle Anforderungen an die Hardware. Ein Begriff der immer wieder auftaucht ist der Duty-Cycle. Wenn es heisst: Diese Anwendung hat einen kleinen Duty-Cycle, so ist gemeint das die Anwendung einen grossen Teil ihrer Zeit schlafend verbringt. Dann wacht sie (z.B. durch Timer oder durch Sensorsignal angestossen) auf, verarbeitet Daten und sendet diese weg. Dann schläft der Knoten wieder. Mit einem kleinen Duty-Cycle kann man viel Energie sparen. Dies ist bei allen mobilen Systemen wichtig da damit die Einsatzdauer verlängert wird. Schlafen bedingt, dass man die Verbraucher ein- und ausschalten kann.

2.1 Low-Power

Da der BTnode sowie das FPGA-Modul in den meisten Fällen von einer Batterie versorgt werden, musste der Energieverbrauch soweit möglich reduziert werden. Dazu wurde vorgesehen das man das gesamte FPGA-Modul via BTnode aus- und einschalten kann. Nach jedem Powercycle (also dem Aus- und wieder Einschalten der Stromversorgung) muss der FPGA neu konfiguriert werden.

2.2 Mobilität

Da das FPGA-Modul als Erweiterung zum BTnode geplant war, und Mobilität immer im Vordergrund stand, wurde bewusst die Fläche des FPGA-Moduls minimiert.

2.3 FPGA Konfiguration über Bluetooth

Die Möglichkeit den FPGA über Bluetooth neu zu konfigurieren ermöglicht in Zukunft Firmware updates in Senserknoten, Routern etc. über Bluetooth vorzunehmen. Den Zusammenhang zeigt Abbildung 4. Der Mikrokontroller des BTnodes hat 2 Hardware UARTs. Am einen hängt das

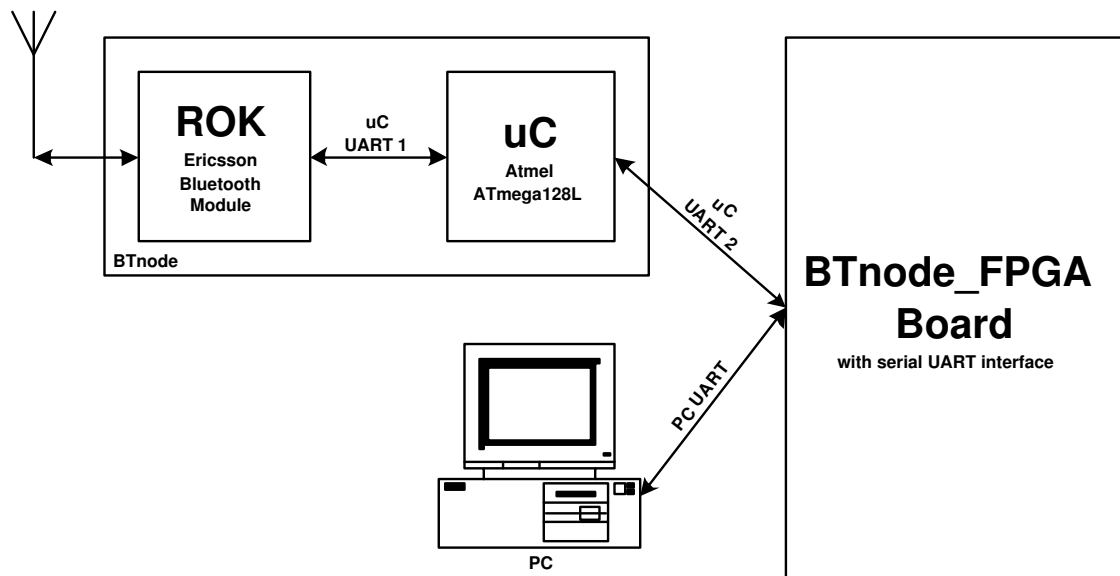


Abbildung 4: BTnode_FPGA kommuniziert via UART mit BTnode oder PC

Bluetooth Modul von Ericsson, der andere ist frei. Deshalb hat das BTnode_FPGA Board ein serielles Interface. Das serielle Interface lässt sich zudem auch einfach von einem PC aus ansteuern. So kann die FPGA Konfiguration auch per PC seriell zum Board übertragen werden.

3 Konzept FPGA-Modul

Das Konzept wurde in 2 Schritten erarbeitet. In einem ersten Schritt wurden wegen Vorgaben aus der Aufgabenstellung (Energie ist begrenzt, Erweiterung zum BTnode) zwei Designentscheidungen getroffen. Im zweiten Schritt wurden dann diese Designentscheidungen anhand von Szenarien durchgespielt und überprüft.

Die Ausgangssituation zeigt Abbildung 5. Wir wollen einen FPGA über den Mikrokontroller im BTnode programmieren.



Abbildung 5: Konzept: FPGA soll via BTnode Mikrokontroller programmiert werden

3.1 Designentscheidungen

3.1.1 Wieso ein Flash Memory?

Mobile Systeme haben immer begrenzte Energiereserven mit denen sie leben müssen (Batterien). Es macht also Sinn, die „Stromfresser“ nur dann einzuschalten wenn man sie braucht. Da der FPGA den weitaus grössten Teil der Energie aller Komponenten auf dem Board benötigt, steht er an dieser Stelle stellvertretend für das gesamte Board. Das heisst es macht Sinn, das gesamte FPGA Board vom Mikrokontroller aus ausschalten zu können.

Das Design in Abbildung 5 hat folgenden Nachteil: Bei jedem Ausschalten und wieder Einschalten (also eines Powercycles) des FPGAs muss dieser wieder neu konfiguriert werden. Die Architektur in Abbildung 6 schafft Abhilfe. Durch den Einsatz eines Flash Speichers (nicht flüchtiger Speicher) steht die FPGA Konfiguration über mehr Powercycles zur Verfügung und muss nicht neu über Bluetooth übertragen werden. Das spart Zeit und Energie, da drahtlose Kommunikation langsam und Energieaufwendig ist im Vergleich zum Auslesen eines Flash Speichers. Natürlich

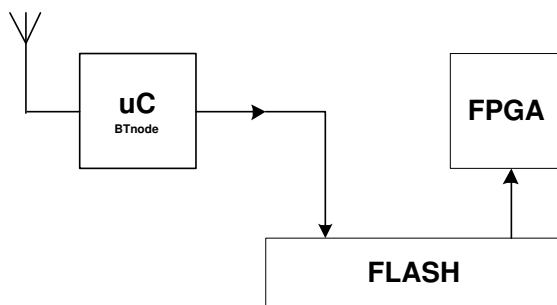


Abbildung 6: FPGA Konfiguration im Flash speichern

muss in Abbildung 6 der Mikrokontroller im BTnode jetzt das Flash programmieren können. Das Problem liegt hierbei in den limitierten I/O Möglichkeiten, die der BTnode bietet. Will man ein Flash programmieren, so braucht man ca. 5 Bit Steuerleitungen, 8 Bit Datenleitungen und 20 Bit Adressleitungen (Total ca. 33 Bit). Soviel I/O steht auf dem BTnode nicht mehr zur Verfügung. Es wäre auch eine Möglichkeit gewesen, ein serielles Flash zu nehmen. Da wäre man mit 4 Bit Ansteuerleitungen aausgekommen. Dies hätte aber folgende Nachteile mit sich gebracht: Es wäre

langsamer gewesen da keine parallele Konfiguration möglich gewesen wäre. Man hätte auch nicht so einfach und wahlfrei auf mehrere gespeicherte Konfigurationen zugreifen könne.

3.1.2 Wieso ein CPLD?

Für die Anbindung des BTnode_FPGA Boards an den BTnode stehen nur ca. 4 Bit zur Verfügung. Deshalb ist es nicht möglich, dass der Mikrokontroller die Aufgabe der Adressgenerierung für den Flash Speicher übernimmt. Der Trick liegt darin, dass ein Complex Programmable Logic Device (CPLD) als Glue-Logic die Kommunikation mit Flash, FPGA und Mikrokontroller kontrolliert. Die Daten zwischen Mikrokontroller und CPLD werden seriell übermittelt (benötigt 2 Bit: rx und tx). Steuerleitungen braucht es zwischen dem Mikrokontroller und dem CPLD deshalb nur wenige. Abbildung 7 zeigt den neuen Sachverhalt. Der Einsatz eines CPLDs für die FPGA Konfiguration wird in [14] erklärt.

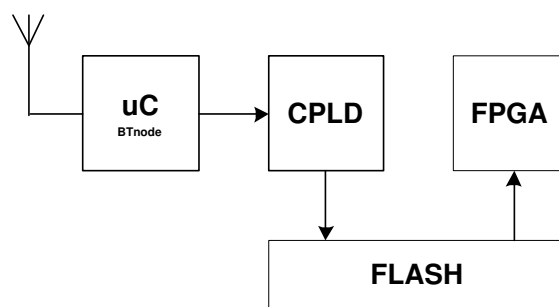


Abbildung 7: CPLD als Interface Konverter

3.2 Szenarien

Einen Überblick der Sensorknoten-Architektur zeigt Abbildung 8. Man sieht die 3 Hauptkomponenten: Den BTnode, das BTnode_FPGA Board und einen angehängten Sensor. Auf dem Board findet man Stromversorgung, Clock und FLASH für CPLD und FPGA. Das SRAM ist als zusätzlicher temporärer schneller Speicher für den FPGA gedacht.

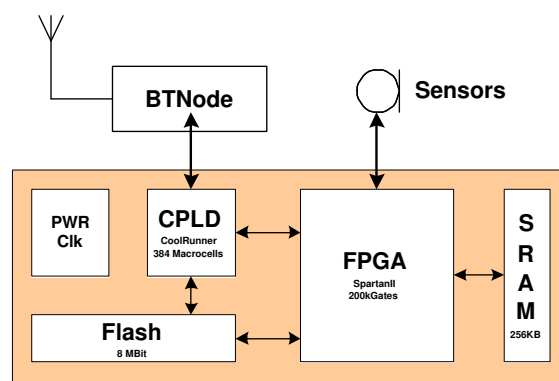


Abbildung 8: BTnode_FPGA im Überblick

Folgende Szenarien wurden durchdacht:

3.2.1 Neue Konfiguration ins Flash schreiben

In Abbildung 9 wird das .bit File das die FPGA Konfiguration beinhaltet von einem PC aus via Bluetooth zum BTnode übertragen. Der BTnode schickt die Daten seriell an den CPLD auf dem BTNode_FPGA Board. Der CPLD schreibt die Konfigurationsdaten ins Flash Memory.

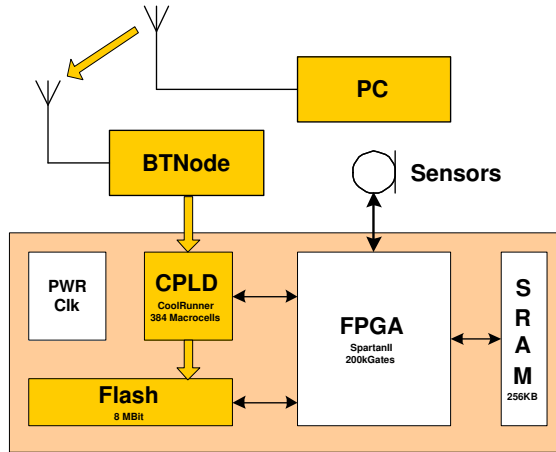


Abbildung 9: Neue Konfiguration ins Flash speichern

3.2.2 Daten aus Flash Lesen

Will der FPGA Daten über mehrere Power On/Off Cycles nutzen, oder dem BTnode zur Verfügung stellen, so speichert der FPGA diese Daten im Flash. Der BTnode kann diese Daten mittels CPLD vom Flash lesen. Ein weiterer wichtiger Punkt ist das ermöglichen einer *verify* Funktion. Dies ist sehr wichtig fürs Debuggen. Sind die Daten einmal im Flash gespeichert, kann man sie zurücklesen und mit dem original .bit File vergleichen. Abbildung 10 fasst dies zusammen.

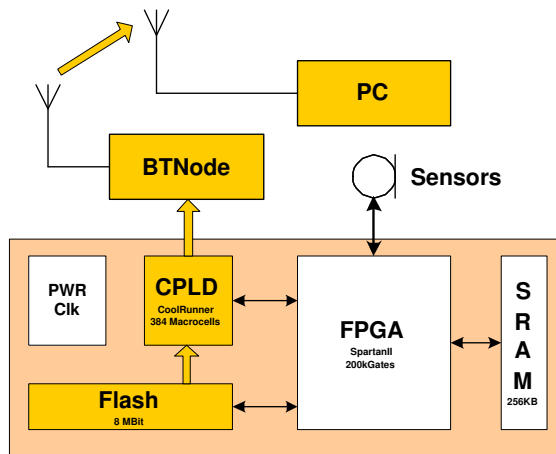


Abbildung 10: (Zurück)Lesen des Flash Speichers

3.2.3 (Re)Konfiguration des FPGAs

Abbildung 11 zeigt: Beim Einschalten des BTnode_FPGA Boards durch den BTnode wird der FPGA via Slave Parallel Mode neu konfiguriert. Der CPLD generiert dabei die Steuersignale und die Adressen für den Flash Speicher und redet mit dem FPGA via Slave Parallel Interface. Dies erlaubt Ein- und Ausschalten des FPGA ohne die Konfiguration für den FPGA erneut über Bluetooth zu versenden.

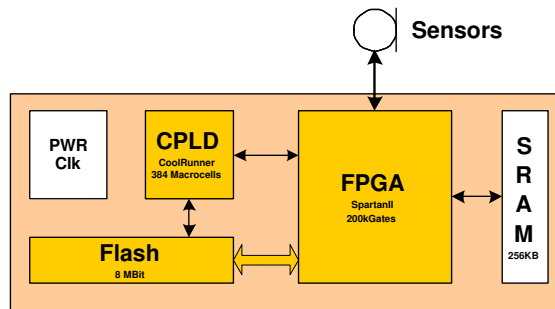


Abbildung 11: (Re)Konfiguration des FPGAs beim Einschalten

3.3 Konfigurations Konzept und BurchED B5-X300 Board Evaluation

Es stand mir das Super-Value-Pack der Firma Burch Electronic Design zur Verfügung [4]. Die Komponenten die ich nutzte waren das B5-X300 Development Board, das Flash Modul (serielles PROM von Xilinx) sowie das Download Cable/Board (Abbildung 22) zum Programmieren der Chips. Xilinx FPGAs unterstützen mehrere Konfigurations Modi. Mit dem BurchED Board konnte ich 2 verschiedene Konfigurationsvarianten für den FPGA evaluieren.

- **JTAG:** Ein weit verbreiteter serieller Konfigurations- und Teststandard (auch unter dem Namen Boundary Scan bekannt) und
- **Master Serial:** Dabei generiert der FPGA selber einen Clock und taktet die Daten aus einem seriellen Flash PROM wie beispielsweise dem Xilinx 18V02 raus.

Da das Flash PROM auch wieder über JTAG programmiert werden musste, schaute ich mir zunächst einmal den JTAG Standard an. Es ist ein ziemlich umfangreicher Standard und die State Machine ist sehr gross. Ich fand die nötigen Infos unter [12]. Schnell war klar, das man JTAG besser nur vom PC aus benutzt (zu debugging Zwecken und via Xilinx ISE, das JTAG sprechen kann). Ich kam zum Schluss, das es zu aufwendig gewesen wäre, JTAG auf dem BTnode zu implementieren. Es wurde eine einfachere Lösung gesucht und gefunden: Durch den Einsatz eines standard 8Bit parallelen Flash boten sich zwei wesentlich Vorteile:

1. Ein standard Flash kann mit standard Mikroprozessor Timings programmiert werden. Ganz ohne JTAG. Dies ist - verglichen mit JTAG - relativ einfach in einem FPGA oder CPLD zu implementieren.
2. Mit einem parallelen Flash ist es möglich, den FPGA über den schnellsten Modus (Slave Parallel) zu konfigurieren und sich gleichzeitig Readback vom FPGA offen zu halten.

Auf dem BTnode_FPGA Board bilden CPLD und FPGA eine JTAG chain. Der CPLD wird immer über den JTAG Port programmiert, der FPGA *kann* via JTAG programmiert werden. Die Konfigurationsdaten für den FPGA werden seriell vom BTnode (über den CPLD) ins Flash gespeichert. Der CPLD generiert dann die nötigen Signale und Adressen, liest die Konfiguration aus dem Flash und konfiguriert den FPGA via Slave Parallel Mode.

4 Umsetzung in Hardware/Software

4.1 Komponentenauswahl

Da der Zeitraum für die Diplomarbeit begrenzt auf 4 Monate war, spielte die Lieferbarkeit der einzelnen Komponenten eine grosse Rolle. Ungefähr 2 Wochen wurde für die Auswahl und die Abklärung der Lieferbarkeit verwendet: „...grüezi da isch Fercher vo der ETH Zürich...“ ;-) Es bleibt zu bemerken, dass für den FPGA und den CPLD keine anderen Hersteller ausser Xilinx in Betracht gezogen wurden, da das ganze Institut TIK auf Xilinx als Hersteller setzt und die Entwicklungsumgebungen auch nur für Xilinx vorhanden sind.

4.1.1 FPGA

FPGAs benutzen SRAM zum Speichern der Konfiguration. Deshalb müssen sie beim Powercyclen neu konfiguriert werden. Um möglichst gut mit ASICs zu konkurrenzieren sind die primären Designziele der FPGA Hersteller High-Speed und High-Density. Mit grösseren Dichten erhöhen sich auch die Leckströme, da es mehr Transistoren pro FPGA hat. Zudem werden die Transistoren in einem halb on halb off Zustand gehalten, um schneller schalten zu können. Bei Xilinx nimmt man an, das sich der Stromverbrauch in den nächsten Jahren verzehnfachen wird. Bei FPGAs gilt als Faustregel: Je neuer desto mehr Strom wird benötigt. Es standen Virtex, VirtexE, SpartanII [7] und SpartanIIE [8] zur Diskussion. Da der FPGA der grösste Stromverbraucher ist wurde mit dem Power Estimator jeweils das gleiche Design auf die 4 Plattformen gemapped (200 slices ADPCM Decoder). Es stellte sich heraus (siehe Tabelle 1), dass bei den FPGAs die E Varianten jeweils über 540mW im Core verheizen. Da SpartanII auch noch sparsamer als Virtex war, wurde entschieden ein SpartanII FPGA zu verbauen. Es wurde die grösstmögliche Version genommen mit 200k Gates. Als Package wurde das grösste nicht Ball Grid Array genommen (PQ208) da der Prototyp von Hand bestückt wurde.

FPGA	Statisch [mW]	Dynamisch [mW]	Total [mW]
SpartanII	14	16	30
SpartanIIE	547	3	550
Virtex	30	16	46
VirtexE	547	3	550

Tabelle 1: Leistungsverbrauch bei den untersuchten FPGAs

4.1.2 CPLD

CPLDs speichern ihre Konfiguration in nicht flüchtigem Speicher. Sie müssen deshalb nach einem Powercyclen nicht neu konfiguriert werden. Bei CPLDs liegen die Design Ziele der Hersteller bei Ultra-Low-Power. Deshalb gilt bei den CPLDs als Faustregel: Je älter, desto höher der Stromverbrauch. Zur Auswahl standen CoolRunner XPLA3 [9] und der neuere CoolRunnerII [10]. Leider konnte über den schweizer Distributor der CoolRunnerII nicht rechtzeitig geliefert werden. So war ich gezwungen den älteren Bruder zu verbauen. Da der CoolRunnerII über interne Pull-Ups verfügt wäre dies sicherlich eine lohnende Option bei einem Redesign. Die Auswahl der Grösse und des Package war primär von den zur Verfügung stehenden User I/O Pins abhängig. Da mehr als 60 User I/O Pins benötigt wurden (40 für Flash und Slave Parallel Ansteuerung + 16 I/O zum FPGA + 4 für LEDs + 8 als Interface zum BNode/PC + 3 für Power On/Off/Status) kam nur ein TQ144er in Frage. Dann wurde die grösste Variante dieses Package ausgewählt (eine 384 Macrocell Variante). Genaue Power Schätzungen wie bei FPGAs konnten leider nicht gemacht werden, weil der Power-Estimator von Xilinx CoolRunner XPLA3 nicht unterstützt.

4.1.3 FLASH

Flash ist nicht flüchtiger Speicher und dient in unserem Design der Speicherung von FPGA Konfigurationen über mehrere Powercycles. Beim Programmieren von Flash Speicher muss man (im Vergleich zu SRAM) folgendes beachten:

- **program:** Mit *program* kann man ein Bit von 1 nach 0 programmieren. Eine erneute (Rück)Programmierung des gleichen Bits von 0 nach 1 ist nicht direkt möglich. Man kann also nicht eine 0 wieder zu einer 1 machen. Dazu muss die Bit Zelle zuerst gelöscht werden.
- **erase:** Mit *erase* wird eine Zelle von 0 nach 1 gelöscht. Die heutigen Flash Chips erlauben allerdings Löschen nur sektorweise. Man kann also nicht ein einzelnes Bit Löschen.

Flashs werden in Uniform und Non-Uniform Sektor Varianten angeboten. Für unseren Zweck wurde bewusst ein Uniform-Sektor Flash ausgewählt, da unser Flash 2 Funktionen übernehmen muss.

1. FPGA Konfiguration .bit File speichern. Dazu werden max 2MBit benötigt. Hier wäre also ein 2MBit grosser Sektor ideal, denn dieser könnte auf einmal gelöscht werden.
2. Der Rest des Speichers kann als Datenspeicher dem FPGA dienen. Hier sind also viele kleine Sektoren idealer.

Da diese Wollmilchsau unter den Flash Chips nicht zu finden war wurde ein Uniform-Sektor Chip mit 16 x 64kByte Sektoren ausgewählt. Es wurde ein 8 x 1MBit Flash von AMD ausgewählt [23]. Ein Vorteil des ausgewählten Chips ist der eingebaute 200nA (!!!) Automatic-Sleep-Current Ultra-Low-Power Mode.

4.1.4 SRAM

Es gibt unzählige SRAM Anbieter die alle nahezu gleiche Produkte liefern. Da beim Design vom BTnode gute Erfahrungen mit AMIC gemacht wurden, habe ich mich ebenfalls für diesen Hersteller entschieden (zumal dieser innert nützlicher Frist gratis Samples liefern konnte). 4MBit wurden als Grösse für genug befunden da der SRAM dem FPGA nur zum Speichern von temporären Daten dienen soll. Da viele Audio Signale mit 16 Bit Breite arbeiten wurde ein 16 x 256kBit Chip ausgewählt [21]. Zu diesem Chip existieren zudem auch Ersatztypen von Allience, Cypress, Mitsubishi und Hitachi.

4.1.5 Power Regulators und Stromversorgung

Für den FPGA und den CPLD sowie für SRAM, Flash und Clock mussten nun die passenden Versorgungsspannungen generiert werden. Als Batterie wurden drei 1.2V Akkuzellen vorgesehen. Diese können selbst bei Grossverteilern (inkl. Ladegerät) für ca 50 Franken gekauft werden (NiMH version ca 2000mAh). Die Batterie liefert also 3.6V. Der FPGA braucht 2 Versorgungsspannungen: 2.5V für den Core und 3.3V für I/O. Die 3.3V I/O Spannung wurde bewusst gewählt, da das Flash, das SRAM und der Clock mit diesen Spannungen arbeiten. Der CPLD braucht ebenfalls 3.3V Versorgungsspannung. Diese 2 Spannungen wurden mit 2 Maxim MAX1818 Linear Voltage Regulators realisiert [22]. Die Empfehlung für diesen Power Regulator findet man in [17]. Hier finden sich auch Beispielschaltungen zur Stromversorgung von FPGAs.

Der FPGA braucht für ein erfolgreiches Power-On während 50ms einen Strom von 500mA. Dies war die wichtigste Anforderung an den Spannungsregler. Die gewählte Variante ist ein Low-Dropout Typ. Als Dropout bezeichnet man den Verlust an Spannung zur gewünschten Ausgangsspannung. Diese beträgt beim MAX1818 maximal 120mV. Da diese Spannung mal den durchfliessenden Strom verheizt wird, war es für unser Low-Power Design ein wichtiges Kriterium.

Der maximale Input auf die Powerregulators (und damit auf das Board) ist 5.5V. Dies darf auf Dauer nicht überschritten werden! Der minimale Input liegt bei 3.3V + 120mV Drop-Out. Sollte also 3.42V nicht unterschreiten.

4.2 Oszillator

Der Oszillator treibt 3 Inputs auf dem Board: Den Clock Input des CPLDs, den Clock Input des FPGAs und den CCLK Input für das Slave Parallel Interface. Deshalb wurde eine kräftige Variante ausgesucht. Es ist ein Oszillator der bis zu 30pF Last treiben kann. Die Clock Frequenz wurde mit 10MHz bewusst in einen Bereich gelegt wo man genug Raum hat für Audio- und Sprachanwendungen. Beispielsweise 44.1kHz mit 128 Fach Oversampling (5.68448MHz) A/D Konverter. Der Oszillator ist von der Firma Jauch [20].

4.3 Part List PCB

In der Tabelle 2 findet man eine Part List mit Bestellnummern, Distributoren und Preisen. Es sind nicht alle abgeklärten Distributoren aufgelistet sondern nur jene, bei denen bestellt wurde. Nach einem letzten abklärenden Telefongespräch bezüglich der Lieferbarkeit der Bausteine war die von den Distributoren bevorzugte Bestellart eine Email. Ein kleiner Tip am Rande: Es empfiehlt sich die Email Bestellung jeweils an eine Hauptadresse zu senden. Ansonsten kann es vorkommen das der Sachbearbeiter mit dem man telefoniert hat eine Zeit lang nicht da ist und die Bestellung versendet (ich spreche aus eigener Erfahrung).

Part	Order Code	Hersteller	Distributor	Kontakt	Tel.	Anzahl	inkl. Tax+Porto [CHF]
CPLD CoolRunner	XCR3384XL-12TQ144C	XILINX	Insight Memec CH	Fr. Schneider	062 919 07 24	5	406.10
FPGA Spartan II	XC2S200-5PQ208C	XILINX	Insight Memec CH	Fr. Schneider	062 919 07 24	5	230.50
FLASH	AM29LV801B-70EC	AMD	Spoerle Electronic	Hr. Josefa Moreno	01 817 62 11	5	80.40
SRAM	LP62S16256EV-55LLT	AMIC	Anatec AG	Hr. Stadlin	041 748 32 32	10	samples
Power Regulators	MAX1818-EUT25	MAXIM	Maxim CH		01 828 80 99	5	12.80
	MAX1818-EUT33	MAXIM	Maxim CH		01 828 80 99	10	46.80
Clock 10MHz,50ppm	6801662	JAUCH	Distrelec CH	distrelec.ch		5	51.50
PCB			Walter-Schoch AG	Fr. Verena Fischer	01 762 41 41	10	583.20
TOTAL						5	1411.30
TOTAL pro Stück						1	282.30

Tabelle 2: Part List PCB: Am Mittwoch, den 19. Februar 2003 war 1 USD = 1.38 CHF

4.4 HW-Design

Das Board wurde mit Cadence Board Design (14.2) unter Solaris auf einer SunBlade 100 erstellt. Nachfolgend die wichtigsten Schritte:

4.4.1 Part Developer

Mit dem Part Developer erstellt man die Schema Symbole für die eigenen Parts. Folgende Parts wurden erstellt: Die Power Regulators, der Clock-Chip, das Flash, das SRAM, der CPLD und der FPGA. Dies ist eine sehr mühselige Aufgabe da man die gesamte Pinliste der Parts eintippen muss (welche Pins sind I/O, welche VCC, welche GND...). Bei 4-6 Pins ist das sicherlich kein Problem, bei einem 208er Package dauert das allerdings etwas länger. Zugleich ist es aber extrem wichtig das man hier keinen Fehler macht (VCC/GND Vertauschungen werden an der dünnsten Stelle getrennt, sofern das Netzteil genügend Strom liefert ;-). Es wurde an dieser Stelle zu zweit gearbeitet. Einge tippt von mir und Nachkontrolliert von Matthias und mir.

4.4.2 Footprints

Im nächsten Schritt wurden zu den entwickelten Parts die Footprints generiert. Dies wurde mit Hilfe eines Perl Skriptes von Jan erledigt. Dies ist eine Sache von einer halben Stunde (Dank dem Skript). Man benötigt lediglich die Chip Dimensionen die man in den Datenblättern [20], [21], [22], [23] und [11] findet.

4.4.3 Schema

Dann konnte mit dem Schemazeichnen begonnen werden. Das Schema erstreckt sich über 3 Seiten. Im folgenden möchte ich auf einige Einzelheiten die mir wichtig erscheinen eingehen. Man findet die Schemas im Anhang. Als Hilfe für mein eigenes Schema diente auch das Schema zum XESS Board [6] und das Schema zum BurchED Board (Dies ist nicht online verfügbar, es befindet sich aber eine Kopie auf der CD zur Diplomarbeit).

1. Schemaseite 1 von 3 (FPGA: Abbildung 28):

Ganz unten sieht man die Stromversorgung. Alle Power Pins wurden mit 100N gestützt. Ganz oben links sieht man das Slave Parallel Interface. Die Konfigurations Pins M0, M1 und M2 wurden als Pull-Ups implementiert. Dies erlaubt durch Stecken eines Jumpers die entsprechende Leitung auf '0' zu ziehen.

2. Schemaseite 2 von 3 (CPLD und Stecker: Abbildung 29):

Der Resettaster S1 ist active-low. Oben rechts sieht man die LEDs: 0-3 sind für den FPGA und 4-7 sind für den CPLD. Zu den Steckern: J1 ist die Pfostenleiste für FPGA I/O. J2 ist der JTAG programmier Port. J3 ist das Interface zum BTnode/PC. J4 ist der Power Input. J5 dient zum Ein/Ausschalten des Boards und zum Statuscheck der Power Regulators. Unterhalb des CPLDs sieht man noch die Konfigurationsjumper für die JTAG Chain. Damit kann man den CPLD vom FPGA isolieren (Nur CPLD: J6 1 und 2 verbinden. CPLD und FPGA: J6 2 und 3 verbinden und J7 1 und 2 verbinden).

3. Schemaseite 3 von 3 (Speicher, Power und Clock: Abbildung 30):

Diese Seite enthält die 2 Fehler, die beim Design gemacht wurden: Der FLASH_RY Pin ist Open-Drain. Man muss ihn mit einem Pull-Up Widerstand (zBsp. 100k) nach +3.3V ziehen. Die Schaltung zu den Power-Regulators findet man in den Datenblättern. Die Schaltung vom Clock wurde vom XESS Board abgeschaut. Der zweite Fehler im Design ist der Pull-Up von den Shutdown Pins der Power-Regulators (Pin 3). Dieser Pull-Up Widerstand (R12) muss mit VBAT anstelle +3.3V verbunden werden.

4.4.4 Layout

War das Schema erstellt, konnte mit dem Physical Design angefangen werden. Es standen 2 Layer zur Verfügung. Routen auf 2 Layern war nur möglich da mit den vielen User I/O Pins des FPGA und des CPLD viele Freiheitsgrade zum Swappen von Pins zur Verfügung standen.

- **Placen der Komponenten und Pinswappen**

Als erstes wurden CPLD und FPGA geplaced. Dann FLASH und SRAM. Danach wurden die Stecker, Clock und das andere Kleinzeugs geplaced. Bei den 100N Kondensatoren war wichtig, diese so nahe wie möglich an die Pins zu placen. Durch die Vielzahl an User I/O Pins konnte das Routen optimiert werden. Wichtig ist hierbei: Wenn im Allegro (physical) Pins geswappt wurden, dort Export physical machen und dann wenn man wieder ins Schema geht dort *zu allererst* Import physical macht. Dies updated automatisch das Schema und tauscht die entsprechenden Pin Nummern aus.

- **Routen der Signalverbindungen von Hand (Breite = 125)**

Zuerst wurden die Signalleitungen geroutet.

- **Clock Leitung Routen (Breite = 125)**

Da die Clockleitung sowohl CPLD als auch FPGA versorgt, ist es wichtig das die Länge der Zuleitungen (da 10MHz Design) in etwa gleich lang sind. Zudem sollte man über möglichst wenige Vias gehen. Wichtig ist auch das man Einkopplungen soweit als möglich verhindert. Deshalb gilt: Keine sich schnell ändernden Signale parallel zur Clockleitung routen.

- **Power Leitungen Routen (Breite = 250 - 500)**

Die Hauptadern der Powerleitungen sind viermal so breit ausgelegt wie die Signalleitungen. Mindestdicke von Power und GND anschlüssen ist doppelt so breit wie bei Signalen. Hauptadern wurden bei Layer Wechsel mit mindestens 3 Vias verbunden.

- **Restlichen Platz mit GND Planes füllen**

Um Einkopplungen vorzubeugen und GND Anschlüsse zu vereinfachen wurden freie Flächen von Top und Bottom Layers anschliessend mit GND Plane gefüllt. Top und Bottom GND Planes wurden dann noch durch zahlreiche Vias miteinander verbunden, um das Potential möglichst gleich zu halten.

- **Design Rules Checken und Printdaten vorbereiten zum Abschicken**

Folgende Checks sind wichtig:

- Unplaced Components: Falls man vergessen hat irgendwas zu placen.
- Design Rules Check: Überprüft Abstände, Grössen usw.
- Unconnected Pins: Dies ist oft ein graphisches Problem. Man findet, dass auf dem Monitor alles gut verbunden ist aber das Tool findet das nicht ;-). Soll heissen, optisch sieht alles ok aus ;-). Dann ist es am Einfachsten, wenn man die Verbindung löscht und neu routet.

Die Printdaten bestehen aus Gerber Files. Dies werden in Cadence über den Menüpunkt Create Artwork erstellt. Versendet wurden:

- PADTOP.art (Abbildung 31)
- PADBOTTOM.art (Abbildung 32)
- SOLDERTOP.art (Abbildung 33)
- SOLDERBOTTOM.art (Abbildung 34)
- ncdrill1.tap
- nctape.log
- art_param.txt

Vor dem Versenden an den PCB Hersteller wurde als letzter Check das Schema mit der Netzliste vom Layout Tool verglichen. Pin für Pin. Es wurden keine Fehler gefunden.

4.4.5 Einholen von Offerten

Es wurden insgesamt Offerten von 4 Herstellern eingeholt:

- PCB-Pool Deutschland, www.pcb-pool.de
Der PCB-Pool ist interessant für Kleinserien (es ist möglich auch nur 1 Stk. herstellen zu lassen, man zahlt pro Quadratdezimeter). Vor allem wirds billig, wenn man bereit ist 10 Arbeitstage in Kauf zu nehmen (halber Preis verglichen mit 5 Arbeitstagen).
- Wiko AG, www.wiko.ch
War teurer als Walter-Schoch AG
- db electronic, www.db-electronic.ch
War ebenfalls teurer als Walter-Schoch AG
- Ausgewählt wurde, da preisgünstig, die Firma Walter Schoch AG. Bei ihr wurde auch das BTnode Board hergestellt. Eine sorgfältige Bearbeitung war also sichergestellt.

Nach ca. 10 Tagen hatten wir unsere 10 Boards. Die Qualität der Leiterplatten war Einwandfrei.

4.5 SW-Design

Da ich diese Diplomarbeit alleine machte, konnte die Software zur Hardware nicht wie sonst üblich parallel entwickelt werden. Ich konnte damit beginnen sobald die Printdaten abgeschickt waren. Es wurde VHDL für den CPLD und den FPGA programmiert. Zum Testen vom VHDL wurde am Anfang das Program *Hyperterminal* unter Windows und später C Programme benutzt, die ich mit MSVC++ entwickelt habe.

Da das Schwergewicht dieser Arbeit in der Entwicklung der Hardware lag, war es mir aus zeitlichen Gründen leider nicht mehr möglich, Software für den BTnode zu entwickeln. Der aufwendigere und vorallem hardware orientiertere Teil wurde aber implementiert. Folgende Roadmap wurde von mir aufgestellt (Prioritätsliste):

1. VHDL für CPLD und FPGA [IMPLEMENTIERT]
 - (a) Test Programme (LED-Flasher für basic hardware Funktionalitätstests)
 - (b) UART Test
 - (c) FLASH read
 - (d) FLASH erase und program
 - (e) Slave Parallel Interface
 - (f) Demo Applikation für Präsentation
2. C Code [IMPLEMENTIERT]
 - (a) UART Test
 - (b) Flash read via UART
 - (c) Flash erase, program and read via UART
 - (d) Konfigurationsfile erzeugen aus .bit File
 - (e) Schreiben von generiertem Konfigurationsfile ins Flash und Readback und dann verify ob erfolgreich.
3. BTnode C Code [NICHT IMPLEMENTIERT]
 - (a) Von PC aus mit BTnode kommunizieren via Linux sockets
 - (b) Ansteuerung des UARTS auf dem BTnode

Konzeptmässig ist der BTnode aber vollständig integriert (durch das serielle 8E1 UART Interface des BTnode_FPGA Boards das im Moment über den UART eines PCs angesprochen wird). Die entwickelte Software beweist einerseits die Funktionalität der entwickelten Hardware und zeigt andererseits dass das erstellte Konzept aufgeht.

4.5.1 VHDL für CPLD und FPGA

Das Rad neu erfinden macht keinen Sinn. So machte ich mich auf die Suche nach VHDL Code der UART implementiert. Auf der Xilinx Webpage fand ich eine Netzliste für Spartan/Virtex. Dies nützte mir nichts, da ich den Code auf dem CPLD laufen lassen musste. Es gab auf der Xilinx Seite auch noch VHDL Code. Diesen Code habe ich dann ausprobiert - erfolglos. Er funktionierte nicht. So suchte ich weiter und fand schliesslich auf dem Web noch Code der funktionierte. Diesen Code habe ich dann 2 Tage lang zusammen mit einem VHDL Model von einem verwandten FLASH Typus getestet.

Zum schrittweisen Debuggen wurde wie folgt vorgegangen.

- **cpld_test**

Nachdem Power, Clock und CPLD aufs Board gelötet waren, wurde mittels dieses Codes der CPLD auf dem Board auf seine Funktionalität getestet. *cpld_test* ist nur ein kleines Laufflicht, das die 4 LEDs des CPLDs regelmässig flashed. Programmiert wird über JTAG.

- **fpga_test**

Nachdem der CPLD funktionierte wurde der FPGA aufs Board gelötet. Dieser wurde dann mit *fpga_test* getestet. Auch dieser Code ist nur ein einfaches Laufflicht.

- **uart_ctrl**

Mit diesem Code wurden verschiedene UART Baudraten getestet. Der Code ist nur eine Echo State Machine. Schreibt man ein Byte zum CPLD so schreibt dieser das Byte gleich wieder zurück. Dies wurde mittels *Hyperterminal* getestet. Nach dem dies funktionierte wurde noch mittels eines VC++ Programms gleichen Namens ein gesamntes .bit File für den FPGA übertragen, wieder empfangen und dann verifiziert.

- **cpld_small**

Nachdem die grundsätzliche Funktionalität des UARTs, des CPLDs und des FPGAs sichergestellt wurde ging es an die Implementation des eigentlichen Codes für den CPLD. Der Code für den CPLD besteht aus 3 wesentlichen Komponenten:

- **Serielle Datenübertragung zwischen BTnode und CPLD**

Durch die Startbitdetektion beim Empfänger wird bei jedem Datenbyte neu synchronisiert. Diese Detektion wird durch ein 16-faches Oversampling relativ zum UART Speed erreicht. Ein Teiler im CPLD kann nur ganzzahlig sein. Dabei entsteht relativ zu dem verwendeten UART Speed ein Fehler. Ein kleine Abweichung von <2% ist dabei tolerierbar da bei jedem Datenbyte neu synchronisiert wird. Der verbaute Clock hat eine Frequenz von 10MHz. In der Tabelle 3 sind die Fehlerabweichungen für verschiedene UART standard Baudraten ausgerechnet. Das Bestmögliche Ergebnis mit 10MHz war 57600 Bit/s. Mit einem 18.432 MHz Oscillator (gleicher Hersteller) lassen sich alle UART Speeds ohne Fehler durch ganze Teiler erreichen. Siehe Tabelle 4

- **Ansteuerung des Flash Speichers**

Die Steuersignale wurden anhand einer 1MBit AMD Variante getestet, für die man unter [24] ein Simulationsmodel in VHDL findet. Diese Model unterstützte zwar nicht alle Modes aber standard *program*, *read* und *erase* konnte getestet werden.

- **Signalgenerierung für Slave Parallel Rekonfiguration des FPGA**

Das Xilinx Slave Parallel Interface ist unter [18] erklärt.

Im folgenden wird das Interface der VHDL Module vorgestellt: Abbildung 12 erklärt die verwendeten Symbole.

uart speed [bit/s]	uart x 16 [Hz]	10MHz/uart x 16	divisor	error [%]
9600	153600	65.10	65	0.16
19200	307200	32.55	33	1.38
38400	614400	10.28	10	1.70
57600	921600	10.85	11	1.38
115200	1843200	5.43	5	7.84

Tabelle 3: Fehler beim verwendeten 10MHz Oszillator

uart speed [bit/s]	uart x 16 [Hz]	18.432 MHz/uart x 16	divisor	error [%]
9600	153600	120	12	0
19200	307200	60	6	0
38400	614400	30	3	0
57600	921600	20	2	0
115200	1843200	10	1	0

Tabelle 4: Keine Abweichung beim Verwenden eines speziellen Oszillators mit 18.432 MHz

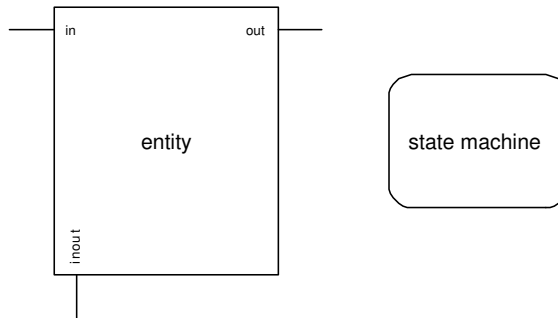


Abbildung 12: Symbole die im folgenden verwendet werden: *entity* und *state machine*

- **rxover.vhd:** Dies ist der rx Teil des UARTs [25]. Er empfängt Daten seriell (1 Start Bit, 8 Daten Bits, 1 Even Parity Bit und 1 Stop Bit) und gibt diese parallel aus. `mclkx16` muss 16 mal so schnell sein, wie Bits seriell übertragen werden. Sobald `rxrdy = '1'` ist kann man mit einem '0' Strobe am `read` Port die Daten am `data[7..0]` Port auslesen. Zudem liefert das Modul die drei Fehlersignale `parityerr`, `framingerr` und `overrun`.
- **txmit.vhd:** Ist der tx Teil des UARTs [25]. Auch er benötigt einen 16 mal so schnellen Clock wie man Bits überträgt. Sobald `txrdy = '1'` anzeigt, kann man die Daten parallel an `data[7..0]` anlegen und einen Strobe an `write` erzeugen (d.h. das Signal kurz nach '0' ziehen und dann wieder nach '1' releasen). Die Daten werden dann seriell über `tx` übertragen (1 Start Bit, 8 Daten Bits, 1 Even Parity Bit und 1 Stop Bit).
- **smallcpld.vhd: sp und reconf** (Abbildung 15)
sp: Diese State Machine konfiguriert den FPGA über den Slave Parallel Mode neu. Sie liest die Daten aus dem Flash mittels der Signale `oe`, `ce`, `we`, `addr`, `ry`. Am Slave Parallel Interface des FPGA werden die notwendigen Signale generiert (`wr`, `cs`, `program`) und gelesen (`busy`, `init`, `done`).
reconf: Liest neue Konfiguration via UART und speichert diese im Flash. Danach folgt ein read und tx via UART um ein verify zu ermöglichen.
- **slow_clock_generic.vhd:** UARTs laufen mit verschiedenen standard Bit/s Raten. Die meisten Implementationen verwenden zum Detektieren vom Startbit ein 16 faches Oversampling

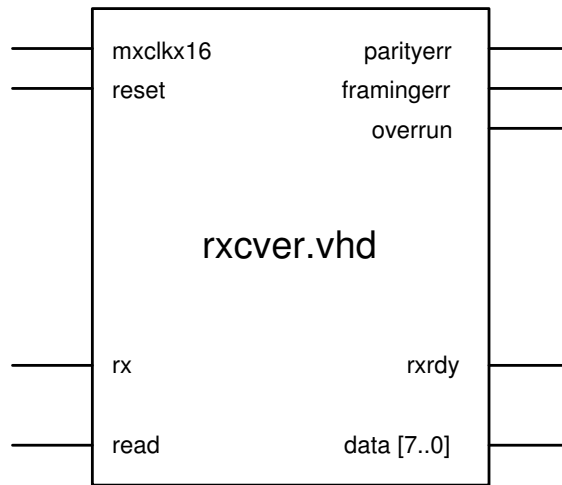


Abbildung 13: Interface vom *rxover* VHDL Modul

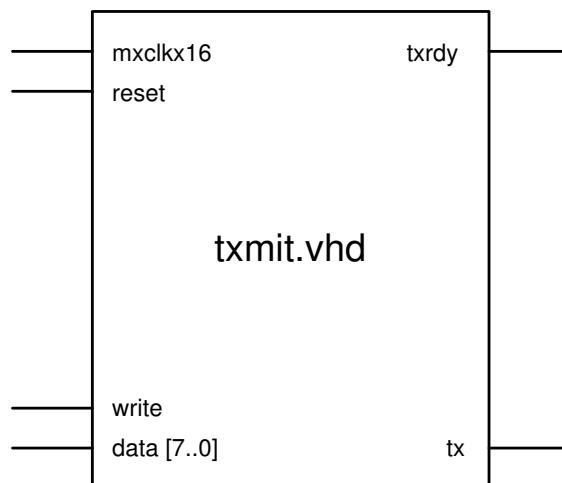


Abbildung 14: Interface vom *txmit* VHDL Modul

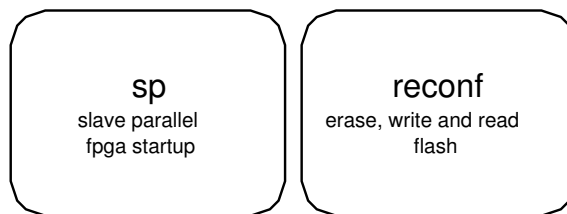


Abbildung 15: *sp* and *reconf* State Machine

bezogen auf den UART Speed. Deshalb brauchen *rxover* und *txmit* Module einen speziellen Clock. Dieser muss abhängig vom Verwendeten Oszillator heruntergeteilt werden. Diese Aufgabe wird von dem in Abbildung 16 gezeigten *slow_clock_generic* Entity erledigt. Es handelt sich dabei um einen modulo n Counter. Er gibt immer nur jeden n-ten Clock Puls vom

Eingang `clk` zum Ausgang `slow_clock` aus.

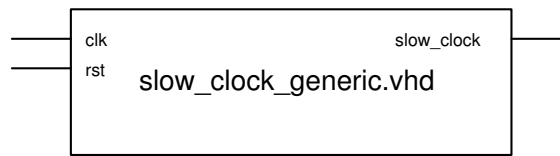


Abbildung 16: Clock divider für UART

- **cpld_small:** Das Toplevel Modul in Abbildung 17 enthält alle bisher erklärten VHDL Module und State Machines. Das Mapping von Signalnamen auf die entsprechenden Pins findet man im `cpld_small.ucf` File.

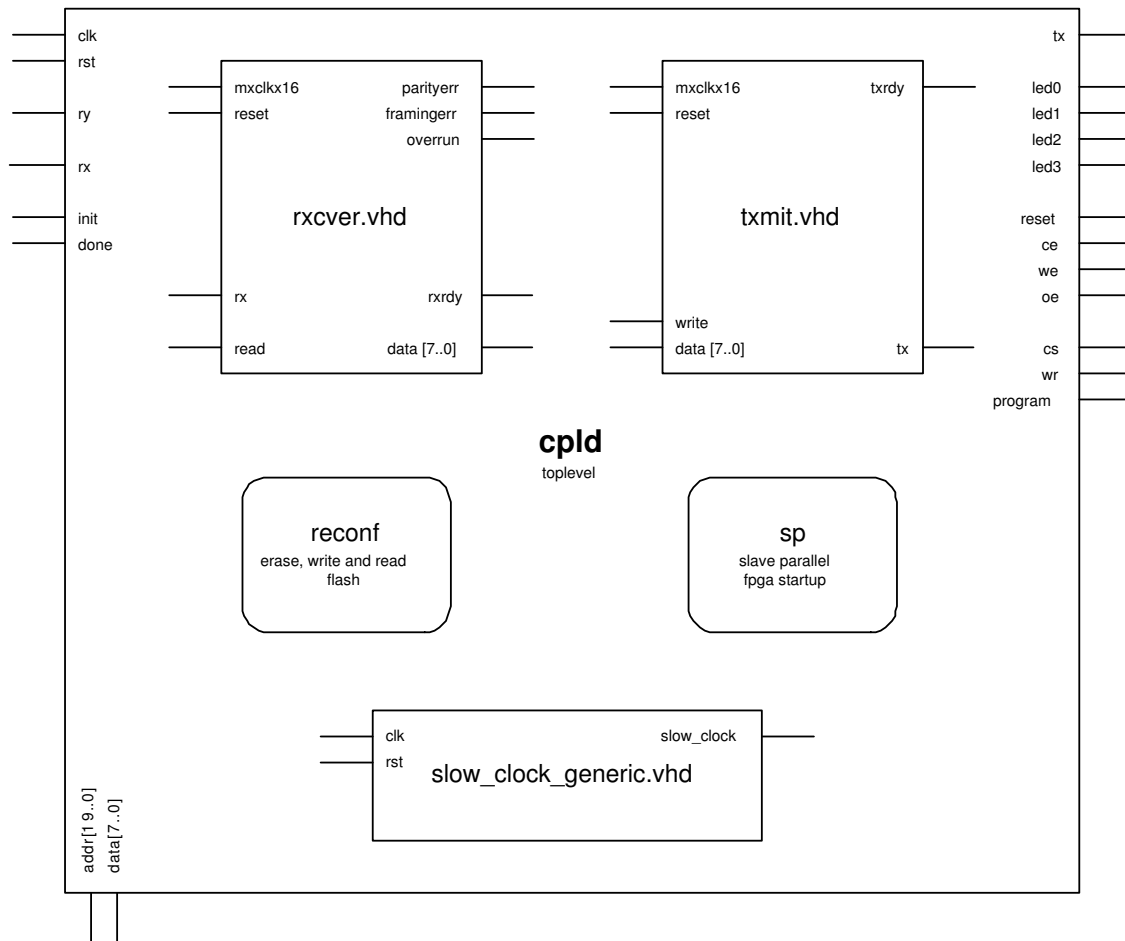


Abbildung 17: `cpld` Toplevel VHDL Modul

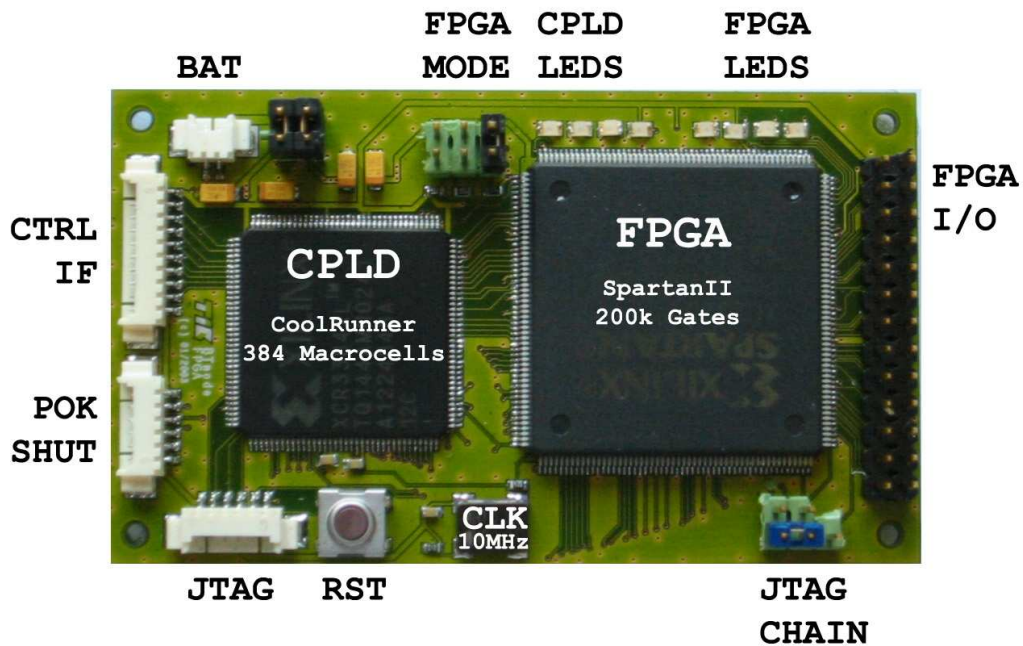


Abbildung 18: Blick von oben auf das Board: TOP Layer

4.6 Inbetriebnahme und Test

Da ich selber noch keine Erfahrung im SMD löten hatte, hat mir Jan das erste Board bestückt (um Fehlerquellen zu minimieren). Zwei weitere Boards habe ich dann selber bestückt. Wenn man selber Bestücken muss, hat man aber den Vorteil, dass man Schritt für Schritt bestücken und testen kann.

4.7 Bestückung PCB

Begonnen wurde mit den Power Regulators. Dabei wurde schon der erste Fehler im Design entdeckt. Der Shutdown Pin der Regulators muss per VBAT hochgezogen werden (Pull-Up). Dies liess sich einfach mit dem Drehen eines Widerstandes und einer kleinen Drahtbrücke erledigen. Dies sieht man auf der Abbildung 19 (mit PATCH 1 bezeichnet). Danach wurde der Oszillator aufgelötet und mit einem KO überprüft. Anschliessend wurde der CPLD gelötet und getestet. Dann folgten FPGA, FLASH und SRAM. Das fertig Bestückte Board zeigt Abbildung 18 und Abbildung 19. Ein zweiter Patch war nötig da der ready Pin des Flashs ein Open-Drain Pin ist. Dies ist in Abbildung 19 mit PATCH 2 bezeichnet.

4.8 Kabel

Abbildung 18 zeigt 4 Stecker von denen 3 gelötet wurden:

- **BAT**: Power Input (2 Pin Molex). Abbildung 20 zeigt eine Variante wie man den Strom vom USB nutzen kann. Zu Test und Messzwecken (Strom/Spannung) wurde aber ein Power-Supply und 2 Messgeräte benutzt.

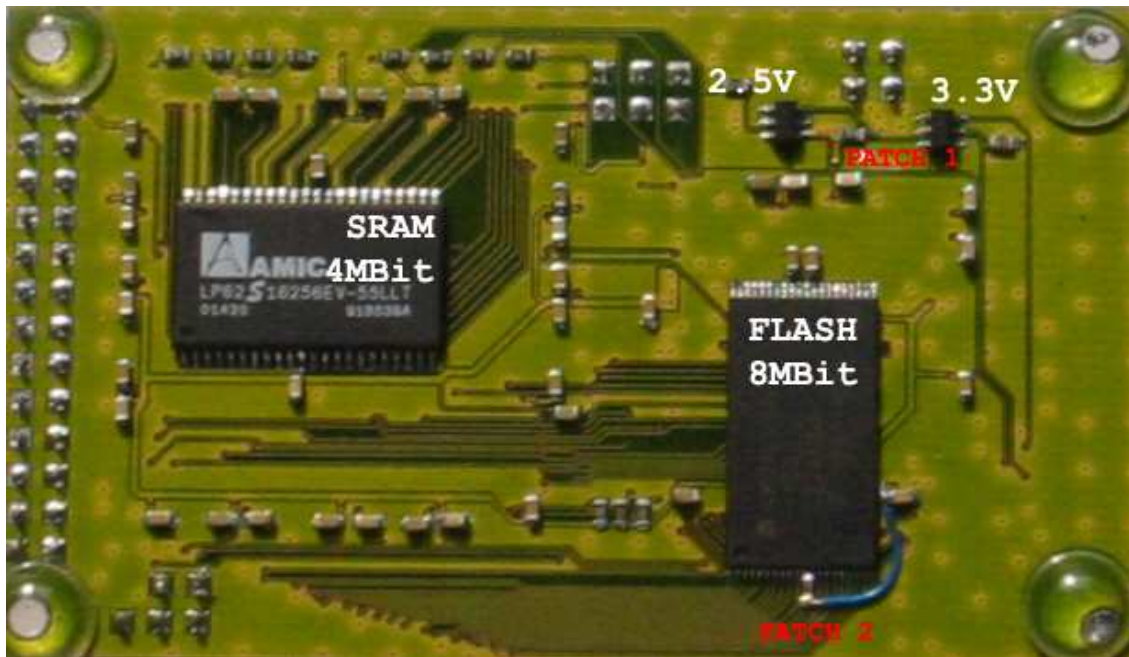


Abbildung 19: Blick auf die Unterseite des Boards: BOTTOM Layer

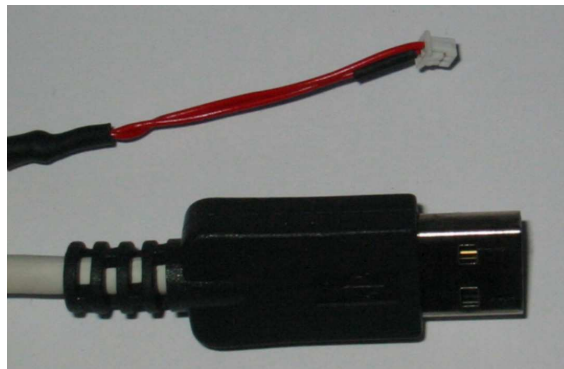


Abbildung 20: Stromversorgung aus dem USB Hub eines PC's

- **CTRL IF:** Interface zum Controller/PC (10 Pin Molex). Wurde nur für rx/tx Pins genutzt. Damit das BTnode_FPGA Board (3.3V Signalpegel) mit dem UART vom PC aus reden kann (12V Signalpegel), braucht es zwischengeschaltet noch einen Levelshifter. Verbunden wird dann mittels folgender Komponenten: Ein gerades RS232 Kabel, ein Nullmodem und ein Gender-Chagner Adapter, zu sehen in Abbildung 21.
- **JTAG:** JTAG Programmier Port (6 Pin Molex). Abbildung 23 zeigt den Aufbau. Abbildung 22 zeigt das Verwendete Download Board von BurchED. Das BurchED Board wird am Parallel Port angehängt und mittels Xilinx ISE wird der CPLD/FPGA via JTAG programmiert.

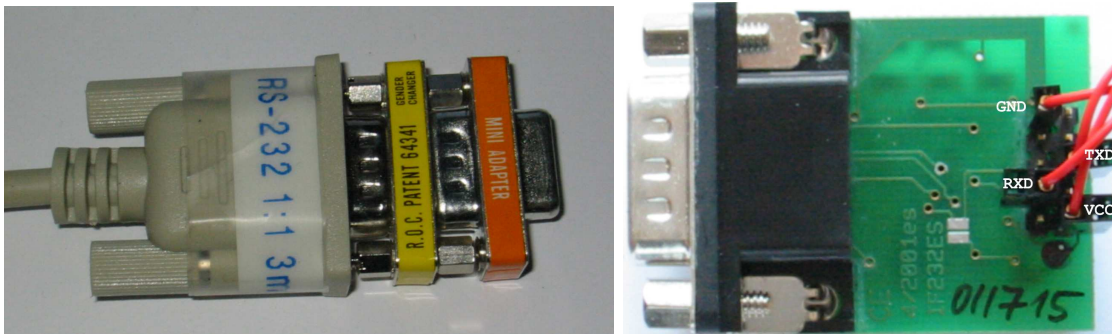


Abbildung 21: Levelshifter 12V PC / 3.3V BTnode_FPGA

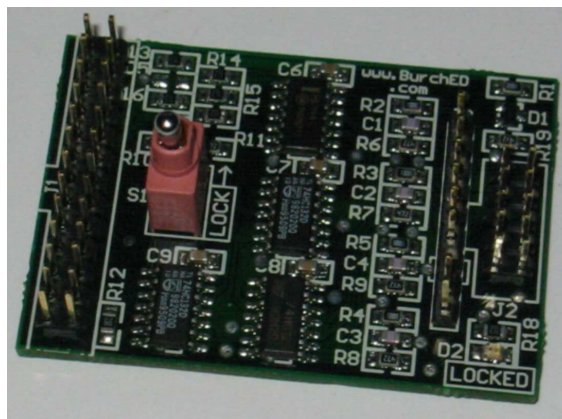


Abbildung 22: Download: Mittels des Download Boards von BurchED (JTAG)

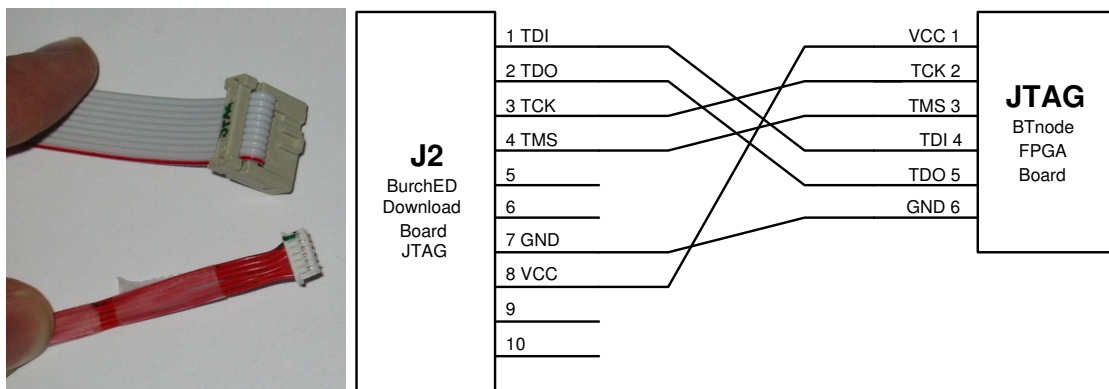


Abbildung 23: JTAG Verbindung von BurchED Download Board und BTnodeFPGA

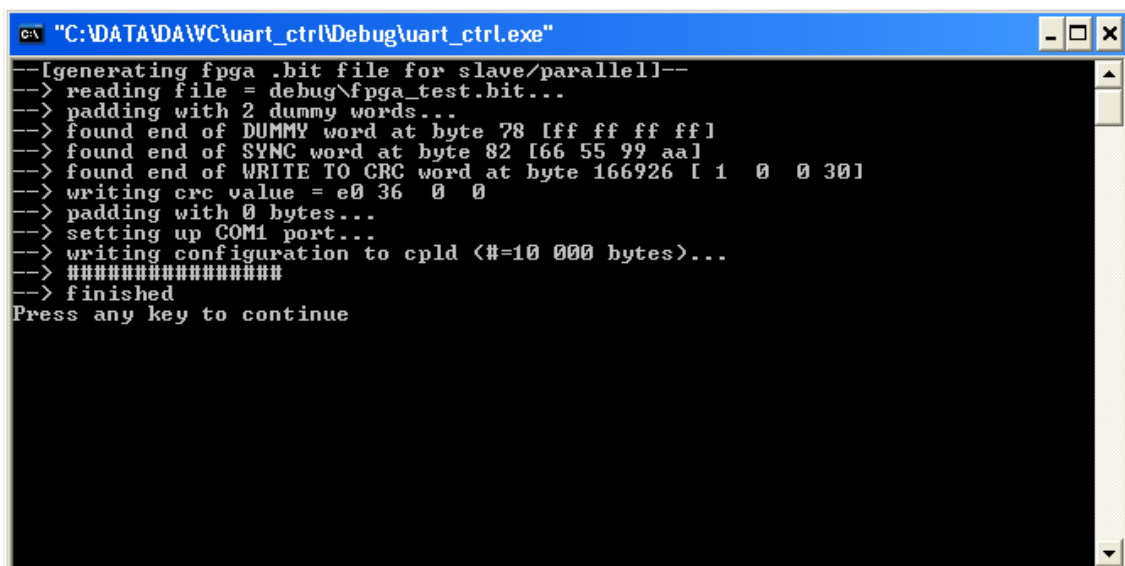
4.9 Funktionalitätstests mit VC++

An dieser Stelle werden die implementierten C Programme zur Verifikation des VHDLs und zur seriellen Ansteuerung des BTnode_FPGA Boards vorgestellt.

- **uart_ctrl.exe:**
Ein .bit File wird Byte für Byte zum CPLD und geschrieben und wieder gelesen. Ein einfacher

Vergleich verifiziert das Echo vom CPLD. Um die UART Signale vom PC Level (12V) zum BTnode_FPGA Board Level (3.3V) zu shiften, wurde ein Levelshifter benutzt. Verbunden wurde das ganze mittels eines geraden RS232 Kabels, eines Nullmodem und eines Gender-Changer Adapters. Zu sehen ist das in Abbildung 21. Will man das Program laufen lassen, so geht man folgendermassen vor

1. CTRL IF Kabel anhängen (10 Pin molex). VCC und GND mit dem Levelshifter verbinden. Dann nach dem .ucf File vom CPLD und den Schemas im Anhang RX und TX mit dem Levelshifter verbinden (RX = Pin 2 nahe an VBAT Stecker, TX = Pin 3). Wo man auf dem Levelshifter einstecken muss zeigt Abbildung 21.
2. JTAG Kabel anhängen an BTnode_FPGA Board und an Download Board (zu sehen in Abbildung 22).
3. Überprüfen ob die Power Jumper gesetzt sind (die neben dem VBAT Stecker). Dann die Speisung anlegen an VBAT. Maximaler Input ist 5.5V. Da der FPGA bei dieser Anwendung nicht benutzt wird, kann als Stromversorgung auch das USB Stromkabel eingesetzt werden. Das Kabel zeigt Abbilgung 20. Es ist einfach selber herzustellen und liefert 5V.
4. Mit Xilinx ISE *uart_ctrl* Projekt öffnen und auf CPLD Downloaden. Die JTAG Chain kann dabei so konfiguriert werden, das nur der CPLD angesprochen werden kann. Siehe Abbildung 18. Im File *slow_clock_generic.vhd* ist die UART Baud Rate eingestellt (Vordefiniert auf 57600 kBit/s)
5. Durch Drücken des RST Tasters auf dem Board (Abbildung 18) die State Machine in den idle State bringen.
6. Jetzt kann man zum Beispiel das Program Hyperterminal unter Windows starten und dann die serielle Schnittstelle auf 57600, 8 Daten Bits, Even Parity Bit und 1 Stop Bit einstellen. Dann kann man drauflostippen im Eingabefenster und sieht immer was getippt wurde (sein eigenes Echo).
7. Um die Übertragung eines ganzen .bit Files zu testen kann man nun (auch nach einem RST) das *uart_ctrl.exe* Program starten. Abbildung 24 zeigt das Ergebnis.



```
C:\DATA\DAWC\uart_ctrl\Debug\uart_ctrl.exe
[generating fpga .bit file for slave/parallel1]--
--> reading file = debug\fpga_test.bit...
--> padding with 2 dummy words...
--> found end of DUMMY word at byte 78 [ff ff ff ff]
--> found end of SYNC word at byte 82 [66 55 99 aa]
--> found end of WRITE TO CRC word at byte 166926 [ 1 0 0 30]
--> writing crc value = e0 36 0 0
--> padding with 0 bytes...
--> setting up COM1 port...
--> writing configuration to cpld <#=10 000 bytes>...
--> #####
--> finished
Press any key to continue
```

Abbildung 24: Output vom *uart_ctrl* Test Program

- **cpld_small.exe:**

Das Hauptprogramm (Abbildung 25):

1. Öffnet .bit File von Xilinx ISE

Da wir den FPGA über den Slave Parallel Mode aufstarten werden, muss man beim Generieren des .bit Files für den FPGA den Startup Clock auf CCLK stellen (Navigationsleiste -> Generate Programming File -> right click -> Properties... -> Tab Startup Options -> FPGA Start-Up Clock auf CCLK setzen).

2. Generiert daraus File für Slave Parallel Mode

Das Datenformat ist in [18] angegeben. Es empfiehlt sich das .bit File mit einem HEX Editor zu öffnen. Der Datenstrom wird in Blöcken von 4 Bytes (1 Word) verstanden. Tabelle 5 zeigt die Hexcodes die man finden muss um das Bitfile zu generieren. Am Anfang von dem von Xilinx ISE generierten .bit File stehen Informationen wie Package, Datum, Uhrzeit usw. Begin der Konfigurationsdaten ist nach dem Synchronisation Word AA 99 55 66. Vor diesem werden allerdings noch 2 Dummy Words FF FF FF FF geschrieben. Das Ende der Konfigurationsdaten kommt nach dem Final Write To CRC 30 00 00 01. Dann folgt noch der CRC Value und dann kommen 16 0x00 Bytes.

Data Type	Data Field
...	
FF FF FF FF	Dummy Word
AA 99 55 66	Synchronisation Word
30 00 80 01	First Real Data
...	
30 00 00 01	Write To CRC
xx xx xx xx	CRC Value
...	
30 00 80 01	Write next 4 Bytes to CMD Register
00 00 00 05	Begin Start-Up Sequence
...	
30 00 00 01	Final Write To CRC
xx xx xx xx	Final CRC Value
00 00 00 00	16 zero Bytes for Slave Parallel
00 00 00 00	
00 00 00 00	
00 00 00 00	
00 00 00 00	

Tabelle 5: Xilinx .bit Bytestream Format

3. TX via UART 3 Bytes Limit(Filesize) zum CPLD

Dies gibt der *reconf* State Machine an, wie lange sie am UART lesen soll.

4. Sendet generiertes Bitstream File via UART zum CPLD, dieser schreibt es ins FLASH

5. CPLD liest FLASH wieder aus und sendet es via UART zurück zum PC/Controller

6. File wird als *readback* File gespeichert

7. Beide Files werden miteinander verglichen (verify)

8. Der CPLD startet den FPGA via Slave Parallel auf (lädt Konfiguration aus FLASH)

9. Der FPGA läuft.

Wenn man also den FPGA neu konfigurieren will, geht man folgenderweise vor:

1. BNode_FGPA Board richtig konfigurieren: Die Mode Jumpers setzen. Für Slave Parallel braucht man M0=0, M1=1 und M2=1 (also Jumper bei M0 setzen, den am nächsten zum FPGA). Diese Info findet man auch unter [18].

2. CTRL IF Kabel anhängen (10 Pin Molex). VCC und GND mit dem Levelshifter verbinden. Dann nach dem .ucf File vom CPLD und den Schemas im Anhang RX und TX mit dem Levelshifter verbinden (RX = Pin 2 nahe an VBAT Stecker, TX = Pin 3). Wo man auf dem Levelshifter einstecken muss zeigt Abbildung 21.
3. JTAG Kabel anhängen an BTnode_FPGA Board und an Download Board, zu sehen in Abbildung 22.
4. Überprüfen ob die Power Jumper gesetzt sind (die neben dem VBAT Stecker). Dann die Speisung anlegen an VBAT. Maximaler Input ist 5.5V! Der FPGA muss über einen kurzen Zeitraum (50ms) 500mA ziehen können für einen erfolgreichen Start-Up. Deshalb muss das Netzteil diesen Strom liefern können.
5. Mit Xilinx ISE *cpld_small* Projekt öffnen und auf CPLD downloaden. Die JTAG Chain kann dabei so konfiguriert werden, das nur der CPLD angesprochen werden kann. Siehe Abbildung 18.
6. Durch Drücken des RST Tasters auf dem Board (siehe Abbildung 18) Beginnt der CPLD das Flash zu löschen. An dieser Stelle möchte ich die Bedeutung der LEDs beim *cpld_small* Projekt erklären (siehe auch VHDL Code):
 - LED0, nahe an den Mode Jumpfern, zeigt an wenn ein UART Fehler eintritt.
 - LED1 zeigt das ready Signal vom Flash.
 - LED2 zeigt an das der *reconf* Vorgang abgeschlossen ist (End *reconf* State Machine).
 - LED3 zeigt an das der Start-Up Vorgang abgeschlossen ist (End *sp* State Machine).
 Begin vom Löschkvorgang erkennt man am Erlöschen der LED1. Ende des Löschkvorgangs erkennt man am erneuten Aufleuchten der LED1 (dann ist das Flash wieder ready).
7. Jetzt (LED1 leuchtet wieder nach RST drücken) ist der CPLD bereit, Daten via UART zu empfangen. Jetzt startet man das Program *cpld_small.exe*. Dieses läuft in den oben beschriebenen Schritten ab.
8. Nachdem *cpld_small.exe* erfolgreich beendet ist leuchtet auch die LED2 (d.h. *reconf* State Machine erfolgreich abgeschlossen). Jetzt startet die *sp* State Machine den FPGA via Slave Parallel auf. Dies geschieht innerhalb von ca 1/40 Sekunde. Der Start-Up Vorgang ist erfolgreich beendet, sobald die LED3 leuchtet.
9. Der FPGA läuft jetzt !

4.10 Demoanwendung

Für die Präsentation wurden 2 FPGA .bit Files gemacht. Ein Light-Flasher links nach rechts und ein Light-Flasher rechts nach links. Diese 2 .bit Files können nacheinander übertragen werden. Man sieht das das Konzept des BTnode_FPGA Boards (Rekonfiguration des FPGAs über serielle Schnittstelle die z.B. vom BTnode oder wie hier vom PC angesteuert wird) aufgeht.

4.11 Bandbreiten und Bottleneck

Abbildung 26 zeigt die Bandbreiten zwischen den verwendeten Komponenten. Klar ist ersichtlich, das die serielle Übertragung der limitierende Faktor ist (57 600 kBit/s). Nun soll noch berechnet werden, wie lange eine Neukonfiguration des FPGAs dauert. Das Start-Up via Slave Parallel dauert nur ca. 1/40 Sekunde für ein 2MBit File bei 80MBit/s und wird deshalb vernachlässigt. Ein .bit File für SpartanII hat eine Grösse von 167 055 Bytes. Durch den verwendeten UART Mode von 1 Start Bit, 8 Daten Bits, 1 Parity Bit und einem Stop Bit, kann man von 57 600 Bit/s nur 8/11 nutzen. Das sind (abgerundet) 41 890 Bit/s. Das reine Übertragen dauert also $167\,055 \cdot 8 / 41\,890 = 31$ Sekunden. Wird ein *readback* und *verify* auch noch durchgeführt wie das im *cpld_small.exe* Program der Fall ist, so dauert eine Neukonfiguration ca. 60 Sekunden. Bei der Verwendung eines

```

c:\ "C:\data\adrtest6WC++Code\cpld_small\Debug\cpld_small.exe"
--[BTnode_FPGA: reconf]--
-> generating fpga .bit file for slave/parallel debug\fpga_test_out.bit...
> reading file = debug\fpga_test_cclk.bit
> writing 2 dummy words (FF FF FF FF)
> found end of DUMMY word at byte 78 [ff ff ff ff]
> found end of SYNC word at byte 82 [aa 99 55 66]
> found first WRITE TO CRC word at byte 166926 [30 0 0 1]
> found final WRITE TO CRC word at byte 167034 [30 0 0 1]
> writing final crc value = 0 0 8 ff
> writing 16 bytes (0x00)...
> filesize = 166984
-> opening com1...
> setting up com1 port (57600, 8E1)
-> writing limit = 166987 to cpld...
-> writing flash up to limit... (&# = 10 percent)
> #####
-> readback of flash data...
> dropping flash content to file debug\fpga_test_readback.bit
> #####
-> verifying...
> #####
> found 0 errors :)
-> finished
Press any key to continue

```

Abbildung 25: Output vom *cpld_small.exe* Test Programm

18.432 MHz Oszillators, der den doppelt so schnellen 115 200 Bit/s UART Mode ermöglicht, kann diese Zeit inkl. *verify* auf ca. 30 Sekunden halbiert werden. Lässt man *verify* weg, kann man eine Neukonfiguration innerhalb ca. 15 Sekunden durchführen. Dies ist ein Akzeptabler Wert da er im Bereich vom Programmieren via JTAG mit Xilinx ISE liegt.

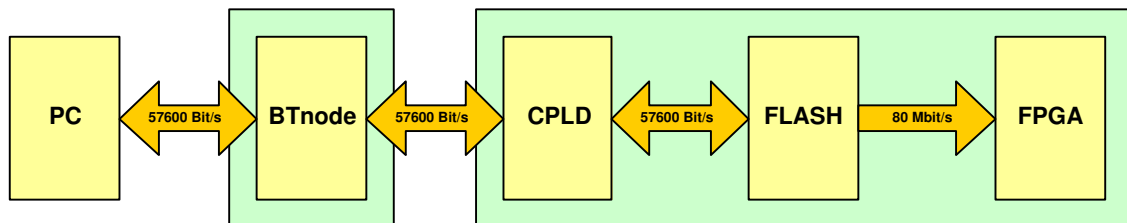


Abbildung 26: Bandbreiten zwischen den verwendeten Komponenten

4.12 Power Berechnungen

Bei der Demoanwendung wurde das gesamte Board mit 5V gespeisen. Der Stromverbrauch lag bei 70mA. Das ergibt 350mW. Dabei lief im FPGA allerdings nicht viel. Realistisch ist ein Stromverbrauch von ca 500mW für das gesamte Board bei 5V. Nimmt man nun an, dass das Board mit 3 mal 1.2V 2000mAh Nickel-Metal-Hydrid Akkus betrieben wird, stehen insgesamt $3.6V \times 2000mAh = 7200mWh$ zur Verfügung. Damit kann das Board dann $7200mWh / 350mW = 20.57h$ also ca 20 Stunden lang *ununterbrochen* betrieben werden. Je nach Duty-Cycle kann die Energie so Wochenlang reichen.

5 Schlussfolgerung

In dieser Diplomarbeit wurde aufgrund der Voraussetzungen (Konzept/Schema BTnode und Anforderungen der Aufgabenstellung: Niedriger Energieverbrauch, Mobilität und Bluetooth Kommunikation) ein Konzept für ein mobiles FPGA Board erstellt: Den BTnode_FPGA. Die Komponenten wurden dementsprechend ausgewählt und bestellt. Die erforderliche Hardware (PCB) wurde designed, bestückt und mittels Software (VHDL- und C Code) getestet. Die gestellten Anforderungen an die Hardware konnten erfüllt werden.

Mit dem BTnode_FPGA steht dem BTnode nun die nötige Rechenpower für Datenpfad orientierte Anwendungen zur Verfügung.

6 Ausblick

Das serielle Interface des BTnode_FPGA Boards zur Rekonfiguration des FPGAs wurde zum Testen der Funktionalität mit dem PC angesteuert. Zur Ermöglichung der Rekonfiguration via Bluetooth bleibt die Arbeit auf der BTnode Seite. Wobei die Aufgabe hier darin besteht, das .bit File vom PC via Bluetooth zum BTnode zu übertragen (Linux sockets). Der BTnode schreibt es dann seriell (wie der PC) zum BTnode_FPGA Board (ein UART Treiber ist bereits vorhanden auf dem BTnode).

Durch die Flexibilität des hardware Designs öffnet sich ein breites Anwendungsfeld im Bereich Sensornetzwerke und Wearable Systems.

A Schemas und Layers

A.1 BTnode

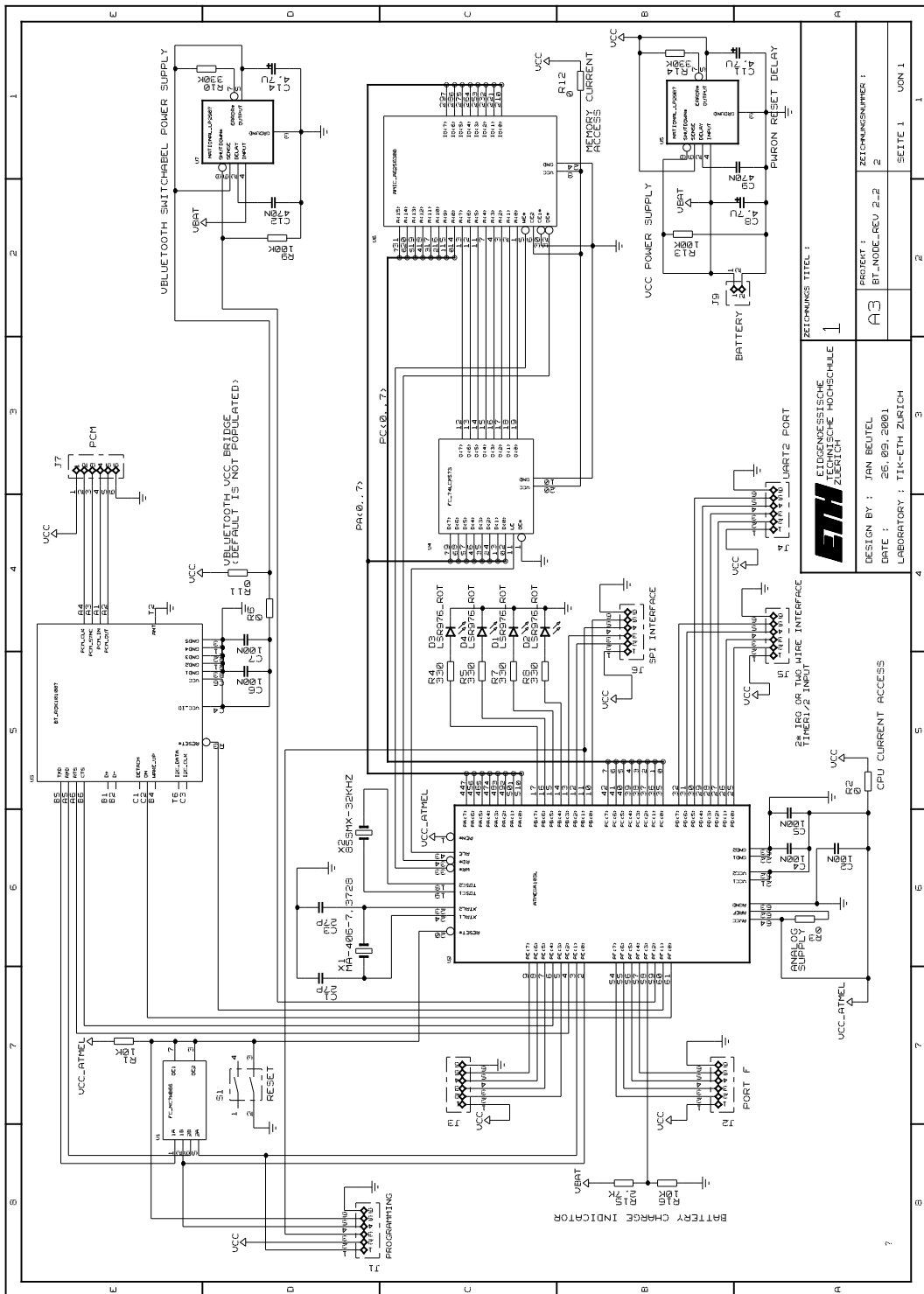
Der Vollständigkeit halber habe ich hier auch das Schema vom BTnode [1] eingebunden. Dieses Schema ist von Jan Beutel (Abbildung 27).

A.2 BTnode_FPGA Schemas

Hier nun die Schemas von dem in dieser Diplomarbeit entwickelten BTnode_FPGA Board (Abbildungen 28, 29 und 30).

A.3 BTnode_FPGA Layers

Abbildungen 31 und 32 zeigen Top und Bottom Layer. Die Pads von Top und Bottom zeigen Abbildungen 33 und 34.



ZEICHNUNGS TITEL :		1	
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE ZÜRICH		1	
DESIGN BY :	JAN BEUTEL	PROJEKT :	BT_NODE_REV_2.2
DATE :	26.03.2001	ZEICHNUNGSNUMMER :	2
LABORATORY :	TIK-ETH ZÜRICH	SEITE 1	VON 1

Abbildung 27: BTnode rev2.2 Schema von Jan Beutel

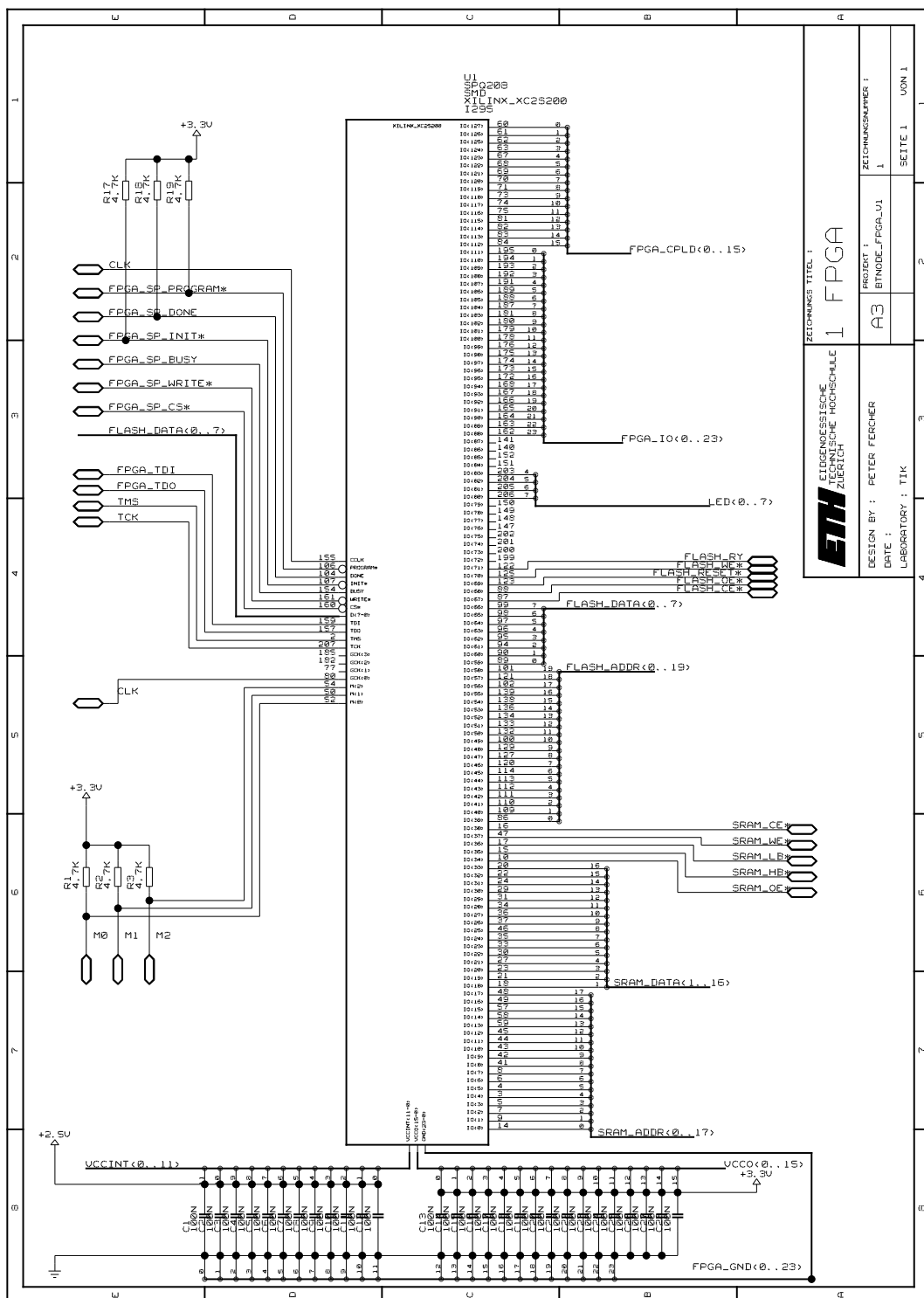


Abbildung 28: BTnode FPGA V1 Schemaseite 1 von 3

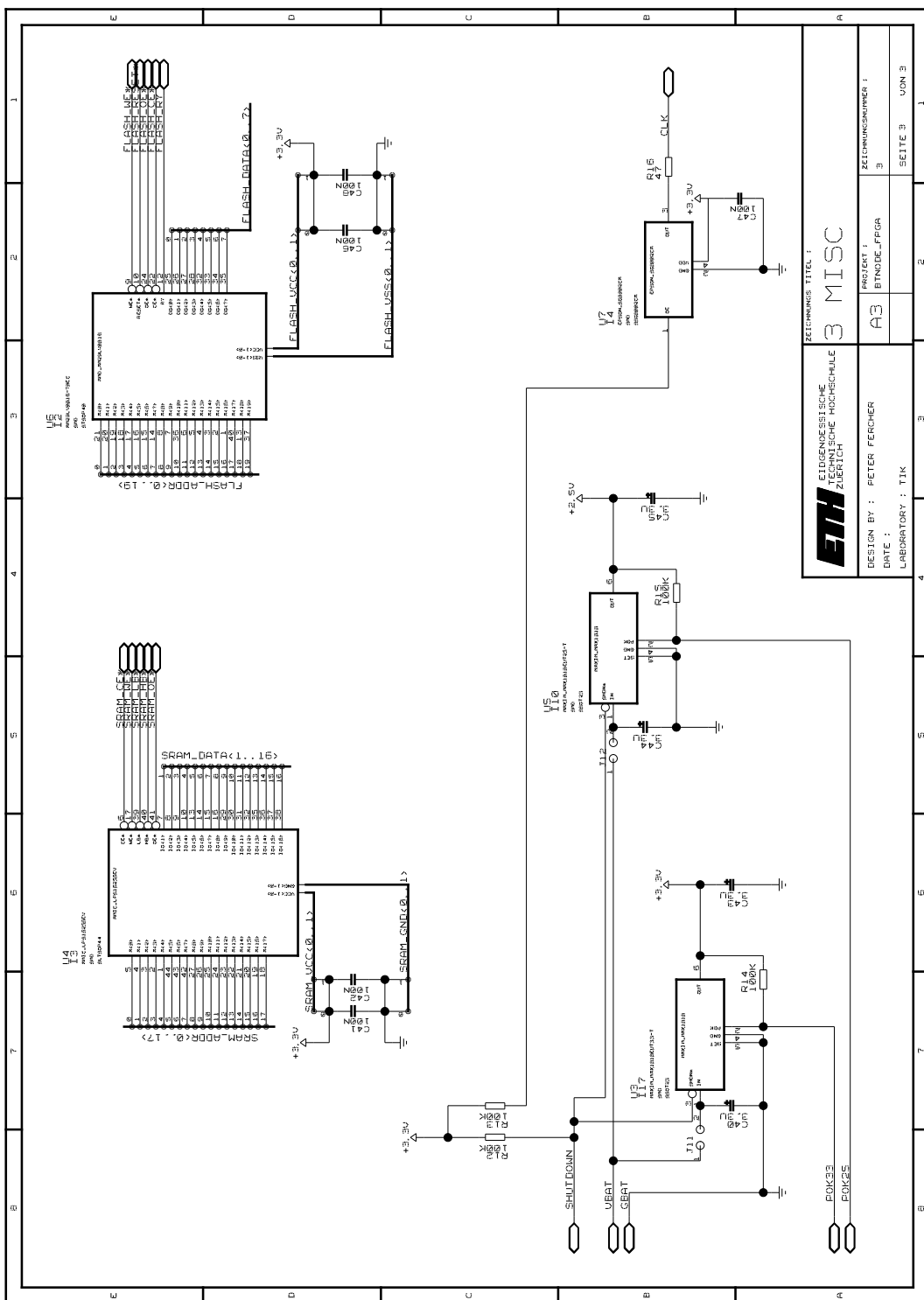


Abbildung 30: BTnode FPGA V1 Schemaseite 3 von 3

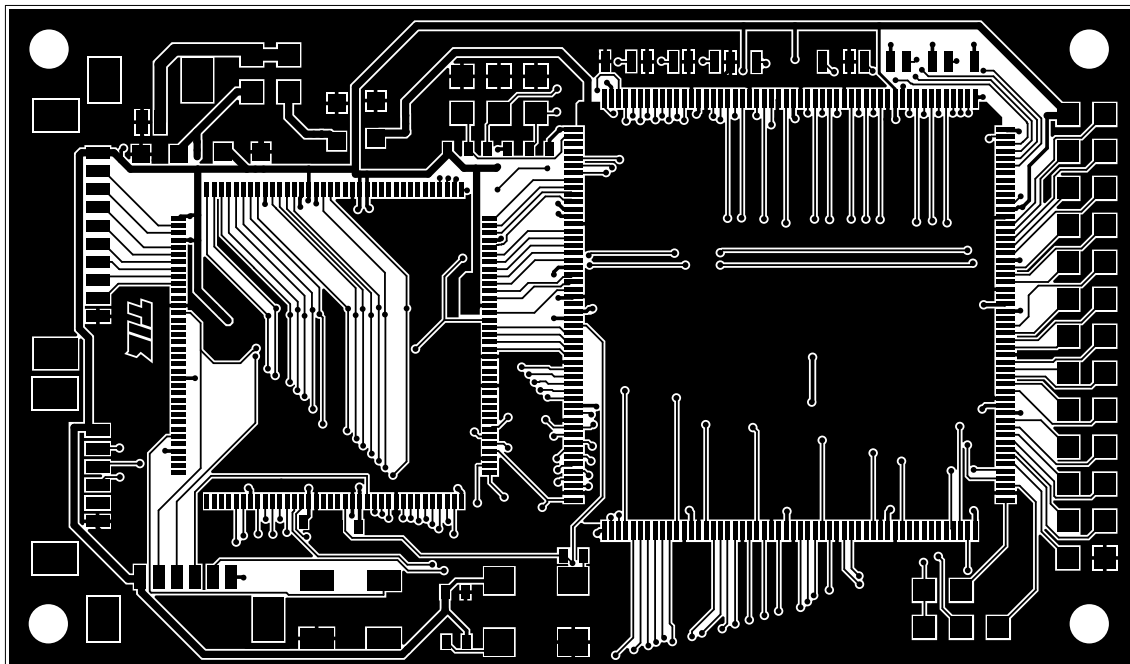


Abbildung 31: PADTOP

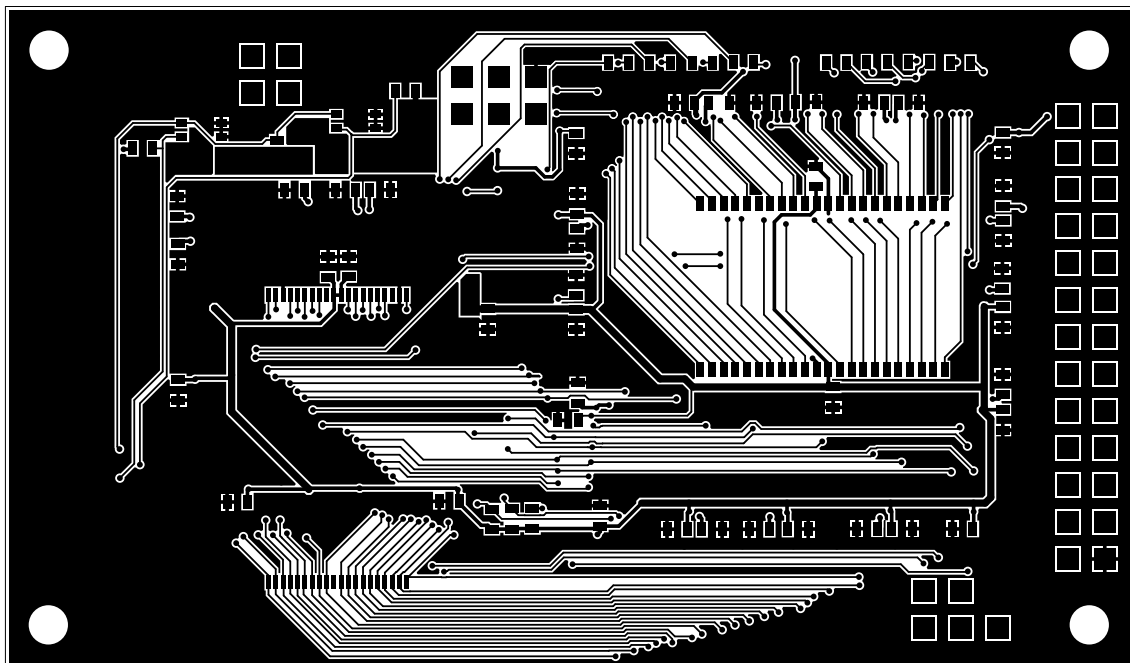


Abbildung 32: PADBOTTOM

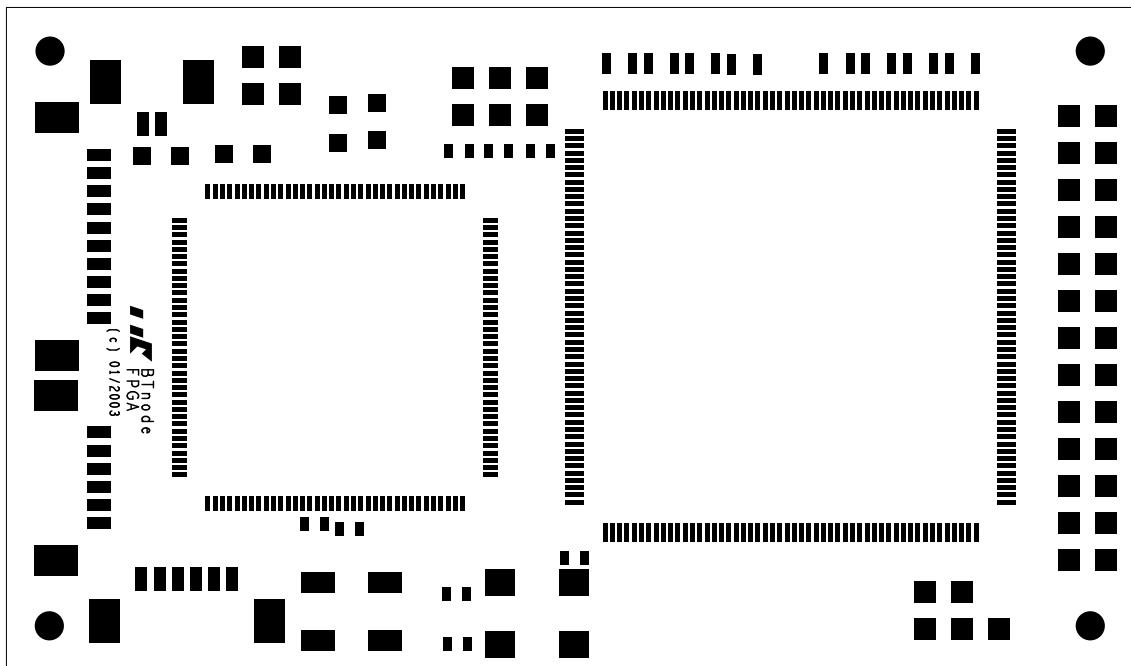


Abbildung 33: SOLDERTOP

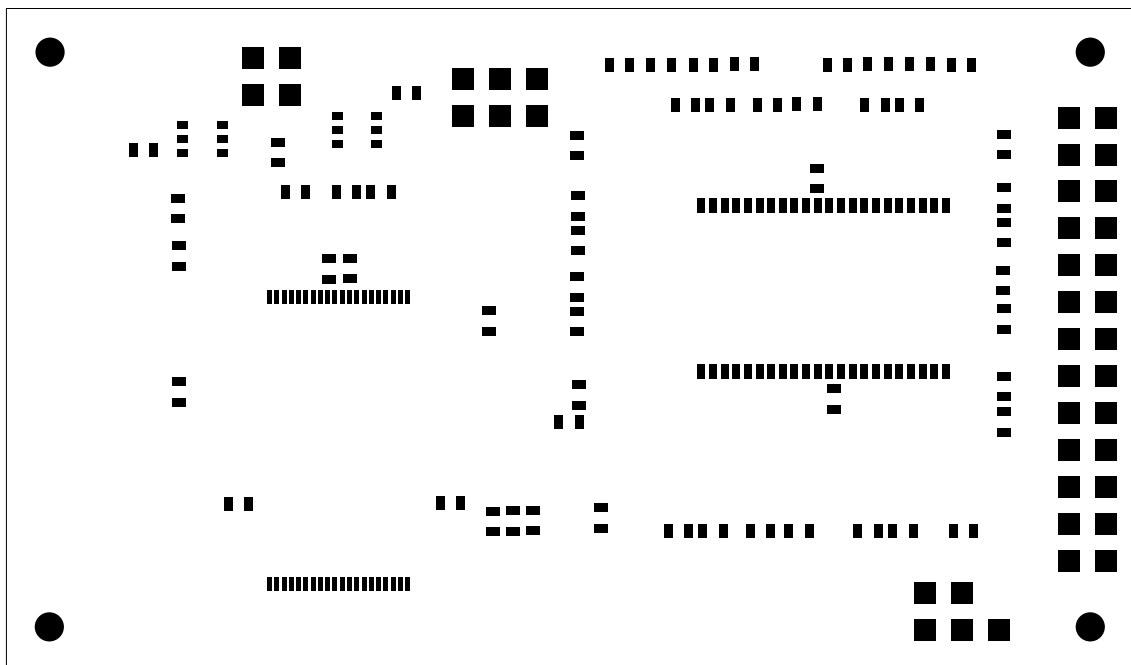


Abbildung 34: SOLDERBOTTOM

Literatur

- [1] Jan Beutel, Doktorant am TIK, ETHZ. BTNode rev2_2, July 2002:
http://www.tik.ee.ethz.ch/~beutel/projects/btnode/btnode_rev2_2
- [2] NCCR MICS Homepage: <http://www.mics.ch>
- [3] Smart-ITs: <http://www.inf.ethz.ch/vs/res/proj/smartits.html>
- [4] Burch Electronic Design, FPGA Boards for System-On-Chip prototyping and education: Super Value Pack. <http://www.burched.biz/b5xsvp.html>
- [5] XESS XSV-800 Virtex Prototyping Board with 2.5V, 800,000-gate FPGA http://www.xess.com/prod014_4.php3
- [6] XESS Board Schemas http://www.xess.com/manuals/xsv-manual-v1_1.pdf
- [7] Xilinx Spartan-II FPGAs, Data Sheets
Introduction and Ordering Information:
http://direct.xilinx.com/bvdocs/publications/ds001_1.pdf
Functional Description:
http://direct.xilinx.com/bvdocs/publications/ds001_2.pdf
DC and Switching Characteristics:
http://direct.xilinx.com/bvdocs/publications/ds001_3.pdf
Pinout Tables:
http://direct.xilinx.com/bvdocs/publications/ds001_4.pdf
- [8] Xilinx Spartan-III FPGAs, Data Sheets
Introduction and Ordering Information:
http://direct.xilinx.com/bvdocs/publications/ds077_1.pdf
Functional Description:
http://direct.xilinx.com/bvdocs/publications/ds077_2.pdf
DC and Switching Characteristics:
http://direct.xilinx.com/bvdocs/publications/ds077_3.pdf
Pinout Tables:
http://direct.xilinx.com/bvdocs/publications/ds077_4.pdf
- [9] Xilinx CoolRunnerII CPLD Family:
<http://direct.xilinx.com/bvdocs/publications/ds090.pdf>
Advance Product Specification:
<http://direct.xilinx.com/bvdocs/publications/ds095.pdf>
- [10] Xilinx CoolRunner XPLA CPLD Family:
<http://direct.xilinx.com/bvdocs/publications/ds012.pdf>
384 Macrocell CPLD:
<http://direct.xilinx.com/bvdocs/publications/ds024.pdf>
- [11] Xilinx package informations PQ208 and TQ144:
<http://www.xilinx.com/partinfo/pkgs.htm>
- [12] Xilinx Application Note, Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan xapp139.pdf: <http://www.xilinx.com/xapp/xapp139.pdf>
- [13] Xilinx Application Note, Configuration Quickstart Guidelines: <http://www.xilinx.com/xapp/xapp501.pdf>
- [14] Xilinx Application Note, Configuring Spartan-II FPGAs from Parallel EPROMs <http://www.xilinx.com/xapp/xapp178.pdf>

- [15] Xilinx Application Note, Power-Assist Circuits for the Spartan-II and Spartan-IIE Families
<http://www.xilinx.com/xapp/xapp451.pdf>
- [16] Xilinx Application Note, Powering Xilinx Spartan-II FPGAs <http://www.xilinx.com/xapp/xapp189.pdf>
- [17] Xilinx Application Note, Power-On Requirements for the Spartan-II and Spartan-IIE Families
<http://www.xilinx.com/xapp/xapp450.pdf>
- [18] Xilinx Application Note, Spartan-II FPGA Family Configuration and Readback <http://www.xilinx.com/xapp/xapp176.pdf>
- [19] Xilinx Application Note, Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode <http://www.xilinx.com/xapp/xapp501>
- [20] Jauch Quarz GmbH Oscillator VX3 3.3V:
http://www.jauch.de/pdf/international/o_vx3_3v.pdf
- [21] AMIC LP62S16256E-I Series, 256K X 16 BIT LOW VOLTAGE CMOS SRAM:
<http://www.amictechnology.com/pdf/LP62S16256E.pdf>
- [22] Maxim MAX1818, 500mA Low-Dropout Linear Regulator:
<http://pdfserv.maxim-ic.com/arpdf/MAX1818.pdf>
- [23] AMD Am29LV081B, 8 Megabit (1 Mb x 8-Bit), CMOS 3.0 Volt only Uniform Sector Flash Memory: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/21525.pdf
- [24] FMF Model List. Simulationsmodelle für Speicher etc.
http://vhdl.org/vi/fmf/wwwpages/model_list.html
- [25] Implementation of a digital UART by VHDL (that works ;-)
<http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1999f/UART/uart.html>