Daniel Kolednik

Configurable Adaptive Mixed Signal Test Pattern Generator

DIPLOMARBEIT

zur Erlangung des akademischen Grades eines

Diplom- Ingenieurs

der Studienrichtung Elektrotechnik-Informationstechnik

eingereicht an der

Technischen Universität Graz

Betreut von:

Ass.Prof. Dipl.-Ing. Dr.techn. Peter Söser

Institut für Elektronik

Technische Universität Graz

Leiter: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Pribyl

Graz, Mai 2011

Danksagung

An dieser Stelle möchte ich mich herzlich bei Herrn Ass.Prof. Dipl.-Ing. Dr.techn. Peter Söser bedanken, der mir die Erstellung dieser Diplomarbeit ermöglichte.

Weiterer Dank gilt Herrn Dipl.-Ing Klaus Strohmayer, der mir seitens Dialog Semiconductor immer mit Rat und Tat zur Seite stand. Herzlichen Dank möchte ich auch Herrn Dipl.-Ing Dr. Wolfgang Mayerwieser aussprechen, der mir die Diplomarbeit, in Zusammenarbeit mit Dialog Semiconductor ermöglichte.

Bedanken möchte ich mich auch bei allen Mitarbeitern von Dialog Semiconductor, die mir beim Entstehen dieser Arbeit behilflich waren.

Besonderen Dank auch an meine Familie, ohne deren Unterstützung und Motivation die Durchführung meines Studiums nicht möglich gewesen wäre.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum	Unterschrift

Kurzfassung

Im Rahmen dieser Diplomarbeit wurde ein System entwickelt, welches digitale Testpattern und analoge Signale, die zur Evaluierung und Fehleranalyse von Chips eingesetzt werden, generiert, und die Ausgabewerte des DUTs (Device Under Test) mit erwarteten Werten vergleicht. Grundlage dafür war eine Projektarbeit mit dem Titel „FPGA Test Pattern Generator“, die bereits am Institut für Elektronik abgeschlossen wurde.

Das Gesamtsystem basiert auf einem FPGA Board, welches bei der kooperierenden Firma Dialog Semiconductor vorhanden war. Darauf wurde ein, auf einem Mikrocontroller basierendes System aufgebaut, welches alle dafür nötigen Einheiten beinhaltet. Auch eine Software, die das Gesamtsystem steuert, wurde entwickelt. Das Hauptaugenmerk lag dabei auf einem Parser, welcher die in Textform geschickten Testdaten in den Flash Speicher ablegt und die Einheiten konfiguriert. Die Testdaten werden, von einem PC aus, über eine USB Schnittstelle übertragen.

Die gesamte Arbeit beinhaltet auch ein Testboard, welches für die Verbindungen vom FPGA zum getesteten Chip verantwortlich zeichnet. Dabei wurde darauf geachtet, so viele Bauteile wie möglich, die von Dialog Semiconductor entwickelt wurden, zu unterstützen. Dazu wurden verschiedene Vorkehrungen getroffen, die es ermöglichen das Testboard auf die zu testenden Bauteile abzustimmen. Das System macht es auch möglich, Chips, welche auf unterschiedlichen Spannungsniveaus arbeiten, zu testen. Es werden dementsprechend verschiedene Versorgungsspannungen zur Verfügung gestellt und auch die Testsignale können auf verschiedenen Niveaus liegen. Mittels eines DACs und eines ADCs können auch analoge Signale erzeugt, beziehungsweise überprüft werden.

Abschließend wurden Testpattern auf das neu entwickelte System geladen und damit vorhandene Chips gemessen und verifiziert.

Abstract

A system was developed for creating and analysing digital and analog testpattern, which are used for evaluation and fault analysis of chips. The thesis is based on a project titled “FPGA Test Pattern Generator”, which was already completed at the “Institut für Elektronik” before.

The system is based on an FPGA board, which was previously used within Dialog Semiconductor. A microcontroller is used as a base unit for the hardware system on the FPGA board. For the proposed function of the pattern generator, different units were implemented in hardware within the FPGA. The communication between the testboard and the PC is done via an USB interface. The firmware, which is only placed inside the microcontroller, is responsible for the communication between the FPGA and the PC. This is achieved via commands directly sent from a text-terminal on PC side. The main part of the software is the parser, which interprets the text commands, saves data into the flash memory and configures the device.

Additionally, the work also includes a testboard, which provides the signals generated from the FPGA board for the DUT. The main goal of the testboard is flexibility to make different products from Dialog Semiconductor testable. The board is developed to be easy configurable. It is possible to supply chips with different supply voltages. Also the digital signals can be created on different voltage levels. To enable the generation and testing of analog signals an ADC as well as a DAC is added to the testboard.

To conclude the thesis, the test pattern generator was configured, test data were loaded into flash memory and chips were verified.

Inhaltsverzeichnis

1 Begriffserklärung

SFR - Special Function Register

- sind Register, die direkt über einen Bus des Mikrocontrollers beschrieben werden können. Sie dienen zur Konfiguration der unterschiedlichen Hardware Einheiten über den Mikrocontroller. Um Informationen über die Hardware zur Verfügung zu stellen können SFRs auch als Statusregister ausgeführt werden.

ADC - Analog Digital Converter

- setzt analoge Eingangssignale in digitale Daten um.

DAC - Digital Analog Converter

- setz digitale Daten in analoge Ausgangssignale um.

DMA - Direct Memory Access

- ist ein unabhängig vom Mikrocontroller ausgeführter Datentransfer zwischen zwei Speicherbereichen.

FPGA - Field Programmable Gate Array

- ist ein integrierter Schaltkreis, in den eine logische Schaltung programmiert werden kann.

VLSI - Very Large Scale Integration

- ist eine Technik für integrierte Schaltungen (IC), die sich durch eine hohe Integrationsdichte auszeichnet.

IC - Integrated Circuit

- ist eine auf einem einzelnen Substrat untergebrachte elektronische Schaltung.

PXI - PCI eXtensions for Instrumentation

- ist ein System zur Messung und Generierung von Signalen.

USB - Universal Serial Bus

- ist ein serielles Bussystem zur Verbindung eines Computers mit externen Geräten

UART - Universal Asynchronous Receiver Transmitter

- ist eine asynchrone serielle Schnittstelle zum Empfangen und Senden von Daten.

IO - Input / Output

- sind Ein- und Ausgänge

LED - Light Emitting Diode

- ist eine Leuchtdiode

SPI - Serial Peripheral Interface

- ist ein synchrones, serielles Bussystem, mit dem digitale Schaltungen verbunden werden können

SOIC - Small Outline Integrated Circuit

- ist ein Packagetyp für Chips

QFN - Quad Flat No Leads

- ist ein Packagetyp für Chips

2 Aufgabenstellung

Die Aufgabe bestand daraus einen Test Pattern Generator für den Laborbetrieb bei Dialog Semiconductor zu entwickeln. Es sollte, mittels des Test Pattern Generators, möglich sein verschiedene Produkte von Dialog Semiconductor testen und evaluieren zu können.

Hauptaugenmerk sollte dabei auf der einfachen Erweiterbarkeit und der einfachen Konfigurierbarkeit des Systems liegen.

Um das System einfach zu erweitern, wurde automatische Code Generierung sowohl bei den Hardwarekomponenten als auch bei der Software verwendet.

Die Steuerung des Systems sollte eine Software übernehmen, welche ausschließlich am System laufen sollte. Dadurch ist zur Steuerung seitens des PCs nur ein Text Terminal notwendig, ohne eine Software am momentan verwendeten PC installieren zu müssen.

Die Konfiguration des Test Pattern Generators sollte durch einfaches Senden eines standardisierten Testdateienformats an das System erfolgen. Als Dateiformat sollte eine „.wgl“ (Waveform Generation Language) verwendet werden. Diese zeichnet sich durch einfache Lesbarkeit aus.

Das Testboard sollte 3 unterschiedliche Spannungsdomains bieten um alle zu testenden Produkte versorgen zu können.

Zum Überprüfen und Erzeugen von analogen Signalen sollten auch ein ADC und ein DAC am Testboard untergebracht werden.

Die Entwicklung wurde in zwei Teile unterteilt, wobei der erste Teil, welcher als Projektarbeit am Institut für Elektronik an der Technischen Universität Graz durchgeführt wurde, sich hauptsächlich mit dem Design der Hardwarekomponenten des Systems beschäftigte. Der zweite Teil, welcher als Diplomarbeit durchgeführt wurde, befasste sich mit dem Entwurf einer Software, dem Entwurf des Testboards und auch mit dem Hardware Design einer Direct Memory Access Einheit, welche auch noch einige Anpassung der bestehenden Komponenten notwendig machte.

3 Test Methoden und Architekturen

3.1 Design für Testbarkeit

Die Steuerbarkeit und die Beobachtbarkeit von kombinatorischer Logik wird verringert, je mehr kombinatorische Logik in einem Bauteil vorhanden ist. Das größere Problem jedoch ist, dass gute Testbarkeit für sequentielle Logikschaltungen nur schwer erreichbar ist. Da sehr viele interne Zustände existieren, bedarf es meist einer sehr großen Anzahl von Eingangswerten um die Schaltung in einen bestimmten Zustand zu versetzen. In weiterer Folge ist es auch sehr schwierig, den exakten internen Zustand anhand der primären Ausgänge zu identifizieren. Ein strukturierterer Ansatz um große sequentielle Schaltungen zu testen ist ein wichtiger Teil des ***design for testability*** (DFT) Ansatzes.

Anfangs wurden verschiedene ***ad hoc*** Techniken entwickelt, um die Testbarkeit zu verbessern. Diese Techniken basieren auf Modifikationen der Schaltungen, um bessere Testbarkeit zu erhalten. Jedoch sind diese Techniken nicht methodisch und können somit nicht direkt auf andere Schaltungen übertragen werden. Ein weiteres Problem ist die Unvorhersehbarkeit des Aufwands, um die DFT Funktionen zu implementieren.

Der strukturierte Ansatz wurde entwickelt, um DFT Ingenieuren methodische Ansätze zu bieten, mit denen sie die Testbarkeit ihrer Produkte verbessern können. Das bedeutet, dass die Anzahl der erkennbaren Fehler möglichst hoch ist. Diese Technik ermöglicht es, den Testaufwand kalkulierbar zu machen. Zusätzlich dazu sind diese Techniken auch weit einfacher zu automatisieren. Verschiedene ***electronic design automation*** (EDA) Anbieter stellen DFT Werkzeuge zur Verfügung.

In den folgenden zwei Unterpunkten werden kurz unterschiedliche ***ad hoc*** DFT Techniken und strukturierte DFT Ansätze gezeigt.

3.1.1 Ad Hoc Ansatz

Der ***ad hoc*** Ansatz beinhaltet einige Reihen von Design Praktiken und Modifikationsrichtlinien zur Verbesserung der Testbarkeit. Solch ein Ansatz setzt meist große Erfahrung des Designers voraus. Die folgende Tabelle listet typische ***ad hoc*** Techniken auf. Danach wird noch ***test point inseration*** genauer erklärt, da diese die wohl am weitest verbreitete Technik darstellt.

A1	Test Punkte einsetzen
A2	Asynchrone Setz/Rücksetz Signale für Speicherelemente vermeiden
A3	Kombinatorische Feedback-Schleifen vermeiden
A4	Redundante Logik vermeiden
A5	Asynchrone Logik vermeiden
A6	Große Schaltungen in kleine Blöcke teilen

Tabelle 3-1 Typische ad hoc Techniken

3.1.1.1 Test Point Inseration

Diese Technik wird häufig verwendet, um Steuerbarkeit und Beobachtbarkeit von internen Knoten zu verbessern. Testbarkeitsanalysen werden verwendet, um interne Knoten zu finden, welche als Testknoten ausgeführt werden sollen.

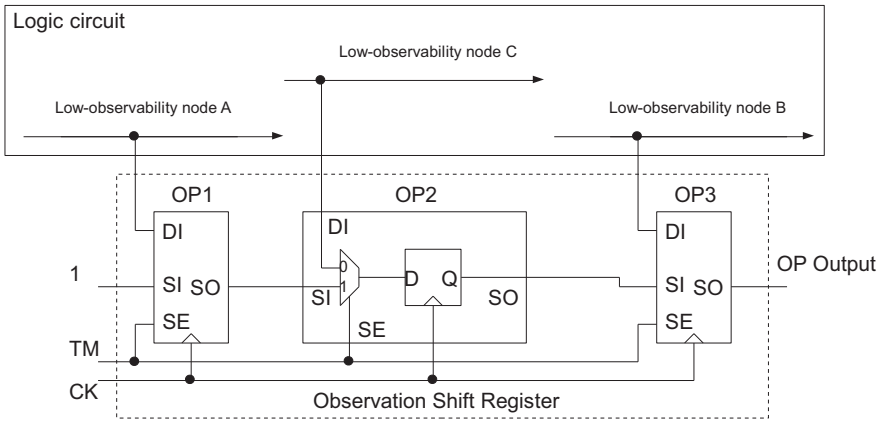


Abbildung 3-1 Observation Point Insertion

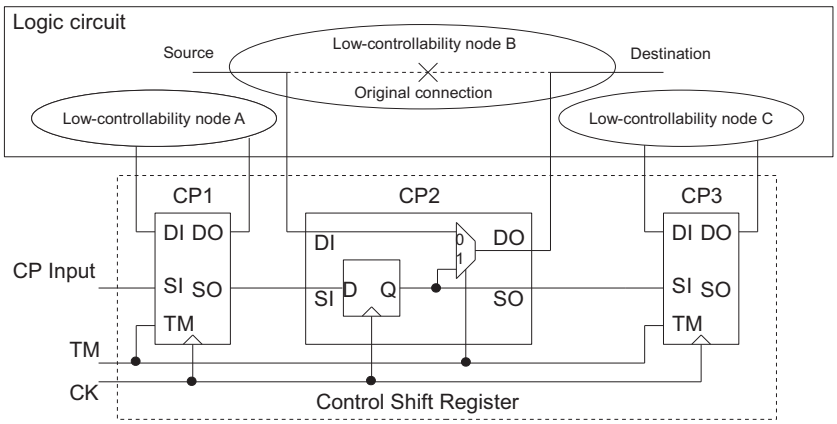


Abbildung 3-2 Control Point Insertion

In Abbildung 3-1 wird das Prinzip der ***observation point insertion*** anhand einer zu beobachtenden Logikschaltung mit drei ***low-observability*** Knoten illustriert. ***OP2*** zeigt die Struktur eines Beobachtungspunktes, welcher aus einem Multiplexer und einem D Flip-Flop aufgebaut ist. Ein ***low-observability*** Knoten ist, im Falle eines Beobachtungspunktes, mit dem Anschluss 0 des Multiplexers verbunden. Alle Beobachtungspunkte sind über den Multiplexeranschluss 1 seriell verbunden. Damit wird ein Beobachtungsschieberegister gebildet. Ein ***SE*** Signal wird verwendet um den Multiplexer zu schalten. Wenn ***SE*** auf logisch 0 liegt und gleichzeitig eine positive Flanke des Taktes ***CK*** auftritt, werden die Logikwerte der ***low-observability*** Knoten in den D Flip-Flops gespeichert. Wenn ***SE*** auf logisch 1 liegt, werden die D Flip-Flops in den Punkten ***OP1***, ***OP2*** und ***OP3*** als Schieberegister fungieren. Dies ermöglicht die sequentielle Beobachtung der einzelnen gespeicherten Logikwerte am Ausgang ***OP Output***. Damit ist die Beobachtbarkeit des Knoten weit verbessert.

In Abbildung 3-2 wird anhand eines Beispiels ***control point insertion*** erklärt. Dies wird mittels einer Logikschaltung mit drei ***low-controllability*** Knoten illustriert. ***CP2*** zeigt die Struktur eines Steuerpunktes, welcher aus einem Multiplexer und einem D Flip-Flop aufgebaut ist. Die ursprüngliche Verbindung bei einem ***low-controllability*** Knoten wird aufgetrennt und ein Multiplexer zwischen ***Source*** und ***Destination*** eingefügt. Im normalen Operationsmodus wird der Testmodus (***TM***) null gesetzt und ***Source*** und ***Destination*** werden über den Multiplexer verbunden. Während des Tests wird das TM Signal auf logisch 1 gesetzt, um mit dem Wert des D Flip-Flops, über den Multiplexer, Destination zu beaufschlagen. Die D Flip-Flops in ***CP1***, ***CP2*** und ***CP3*** bilden ein Schieberegister. Dadurch werden die Werte an ***CP Input*** verwendet, um Destination damit zu beaufschlagen. Damit wird die Steuerbarkeit des Knotens immens verbessert. Es werden dadurch jedoch Signallaufzeiten verändert. Deshalb muss darauf geachtet werden, solche Knoten ausschließlich in nicht zeitkritischen Pfaden einzusetzen.

Grundsätzlich sind jedoch ***scan points***, welche eine Kombination aus Steuerpunkt und Beobachtungspunkt darstellen, zu bevorzugen.

3.1.2 Strukturierter Ansatz

Der strukturierte DFT Ansatz verbessert die gesamte Testbarkeit einer Schaltung mittels ***test-oriented design methodology*** [Williams 1983] [McClusky 1986]. Dieser Ansatz ist ein methodischer und systematischer mit besser absehbaren Resultaten.

3.1.2.1 Scan Design

Scan Design, die am meisten verwendete DFT Methodik, versucht die Testbarkeit von Schaltungen zu verbessern, indem Steuerbarkeit und Beobachtbarkeit von Speicherelementen in sequentiellen Entwürfen verbessert werden. Typischer Weise kann

dies erreicht werden, wenn das sequentielle Design in ***Scan Design*** konvertiert wird. Dabei wird eine sequentielle Logik auf eine kombinatorische reduziert. Dies reduziert die Komplexität der Algorithmen zur Testpatterngenerierung. Dies kann in drei Betriebsarten passieren: ***normal mode***, ***shift mode*** und ***capture mode***.

Im ***normal mode*** sind alle Testsignale abgeschaltet und das Scan Design arbeitet in funktioneller Konfiguration. In beiden Modi, ***shift mode*** und ***capture mode***, wird ein Testmodus Signal (***TM***) verwendet, um alle Testoptionen einzuschalten. Diese sind notwendig um Tests zu vereinfachen, zum Debugging, für Diagnoseaufgaben, um Fehlerabdeckung zu verbessern und um einen sicheren Betrieb der Schaltung im Testmodus sicherzustellen.

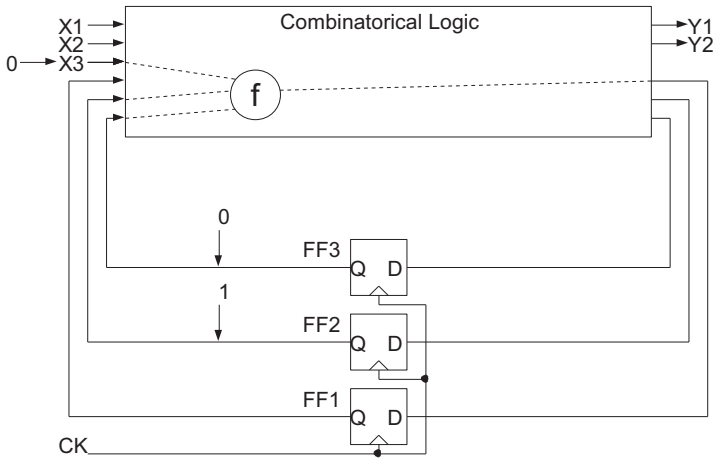


Abbildung 3-3 Schwierigkeit beim Testen sequentieller Logik

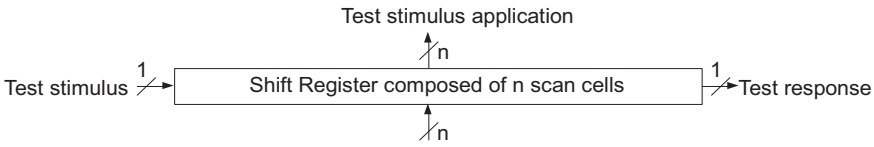


Abbildung 3-4 Scan Design Konzept

In Abbildung 3-3 wird Scan Design anhand einer sequentiellen Schaltung dargestellt. Die Schaltung beinhaltet kombinatorische Logik und drei D Flip-Flops. Ein ***stuck-at*** Fehler (***f***) in der kombinatorischen Logikschaltung benötigt den Eingang ***X3*** (logisch 0), das Flip-Flop ***FF2*** (logisch 1) und das Flip-Flop ***FF3*** (logisch 0) um den Fehlereffekt im Flip-Flop ***FF1*** zu erfassen. Da die abgespeicherten Werte in ***FF2*** und ***FF3*** nicht direkt über Eingänge steuerbar sind, ist eine sehr lange Sequenz nötig, um die beiden Flip-Flops auf gewünschte Werte zu setzen. Um den Fehlereffekt, welcher in ***FF1*** erfasst wurde, am Ausgang sichtbar zu machen, muss eine lange Testprozedur ablaufen. Aus diesem Beispiel kann man entnehmen, dass die Hauptschwierigkeit beim Testen sequentieller Schaltungen die Beobachtbarkeit und Steuerbarkeit interner Zustände ist.

Scan Design, das in Abbildung 3-4 illustriert wird, soll diese Schwierigkeiten mittels Zugängen zu Speicherelementen vereinfachen. Dies kann erreicht werden, indem man ausgewählte Speicherelemente in ***scan cells*** umwandelt und sie anschließend zu Schieberegistern verbindet, sogenannte ***scan chains***. In Abbildung 3-4 werden die ***n*** Speicherelemente als Schieberegister konfiguriert. Alle Testsignale können in die ***n scan cells*** in ***n*** Taktzyklen hinein und heraus geschoben werden.

Das Beispiel in Abbildung 3-3 kann somit weit einfacher behandelt werden:

1- In den Schiebemodus wechseln und die gewünschten Werte 1 und 0 in die Flip-Flops FF2 und FF3 schieben.

2- Logisch 0 am Eingang X3 anlegen.

3- In den Erfassungsmodus schalten und einen Takt anlegen, um den Wert in FF1 abzuspeichern.

4- In den Schiebemodus wechseln und die Werte aus FF1, FF2 und FF3 herausschieben und vergleichen.

Da Scan Design Zugang zu den internen Speicherelementen bietet, kann die Testgenerierung vereinfacht werden.

3.2 Fehler Modelle

Wegen der Zunahme von VLSI Defekten, müssen immer komplexere Tests entwickelt werden, um diese Fehler zu detektieren. Fehlermodelle sind notwendig zur Generierung und Evaluierung von Testvektoren.

Ein gutes Fehlermodell muss zwei Kriterien erfüllen:

1- Es muss das Verhalten des Fehlers genau widerspiegeln.

2- Effizienz bei Fehlersimulation und Patterngenerierung.

Es wurden viele Fehlermodelle entwickelt, jedoch keines kann auf sich alleine gestellt alle Fehler eines Systems wiedergeben, die auftreten können. Daraus resultiert, dass unterschiedliche Fehlermodelle kombiniert werden, um Test Vektoren zu erzeugen und damit ICs zu testen.

Für ein Fehlermodell können *k* unterschiedliche Typen von Fehlern an potentiellen Fehlerstellen auftreten. (bei den meisten Fehlermodellen ist *k* = 2). Eine Logikschaltung beinhaltet *n* mögliche Fehlerstellen, abhängig vom Fehlermodell. Bei der Annahme, es gäbe nur einen Fehler in der Schaltung, ergibt sich eine Gesamtanzahl der möglichen Einzelfehler aus folgender Gleichung.

$$single - fault\ Modell: Anzahl\ der\ Einzelfehler = k * n$$

Normalerweise treten jedoch mehrere Fehler gleichzeitig in einer Schaltung auf. Die absolute Anzahl der möglichen Kombinationen von multiplen Fehlern lautet daher.

$$multiple - fault\ Modell: Anzahl\ der\ Einzelfehler = (k+1)^n - 1$$

Hierbei ist hinzuzufügen, dass der Term -1 die fehlerfreie Logikschaltung darstellt.

Das multiple-fault Modell entspricht mehr der Realität als der single-fault Ansatz, jedoch impliziert dieser auch einen sehr hohen Mehraufwand. Die Anzahl der möglichen Fehler wird sehr hoch. Schon bei einer geringen Anzahl von Fehlertypen und Fehlerstellen ist die Anzahl der möglichen Fehler so hoch, dass das System höchst unpraktisch wird. Glücklicherweise hat sich gezeigt, dass eine hohe Fehlerabdeckung mittels single-fault Ansatz eine hohe Fehlerabdeckung im multiple-fault Ansatz zur Folge hat, da man davon ausgehen kann, dass die einzelnen Fehler des multi-fault Modells unabhängig voneinander sind und daher wie single-fault Fehler behandelt werden können.

Auch beim single-fault Ansatz gibt es äquivalente Fehler, welche zu Vereinfachungen herangezogen werden können. Dies nennt man ***fault collapsing***. Dadurch wird die Simulationszeit reduziert und auch der Aufwand zur Testgenerierung.

3.2.1 Stuck-At Fehler

Das Stuck-At Fehlermodell wird bereits seit Jahrzehnten erfolgreich genutzt. Dieses Fehlermodell beeinflusst den logischen Zustand der Signale in Logikschaltungen. Die beeinflussten Signale sind primäre Eingänge, primäre Ausgänge, interne Gate Eingänge und Ausgänge. Dieses Modell legt fehlerhafte Leitungen der Logik auf definierte konstante Logikwerte, entweder logisch „1“ oder logisch „0“.

Diese Zustände werden auch ***stuck-at 0*** (***SA0***) oder ***stuck-at 1*** (***SA1***) genannt.

Im Falle der anschließenden Logikschaltung besitzt diese 9 Signalleitungen. Bei einem single-fault Ansatz gibt es 18 (2*9) mögliche fehlerhafte Schaltungsmöglichkeiten.

Folgende Tabelle zeigt die Wahrheitstabelle für die fehlerhaften Schaltungen und die fehlerfreie Schaltung.

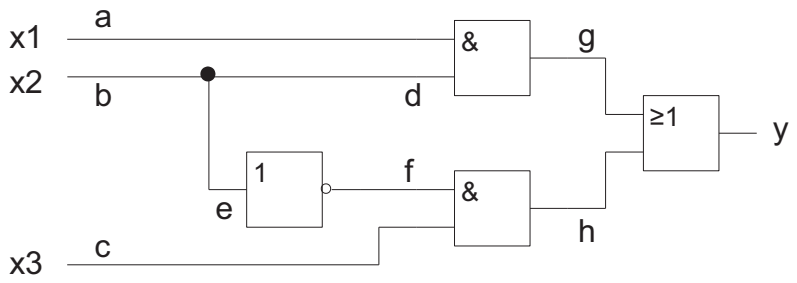


Abbildung 3-5 Stuck-At Fehler Beispiel

x1,x2,x3	000	001	010	011	100	101	110	111
y	0	1	0	0	0	1	1	1
a SA0	0	1	0	0	0	1	0	0
a SA1	0	1	1	1	0	1	1	1
b SA0	0	1	0	1	0	1	0	1
b SA1	0	0	0	0	1	1	1	1
c SA0	0	0	0	0	0	0	1	1
c SA1	1	1	0	0	1	1	1	1
d SA0	0	1	0	0	0	1	0	0
d SA1	0	1	0	0	1	1	1	1
e SA0	0	1	0	1	0	1	1	1
e SA1	0	0	0	0	0	0	1	1
f SA0	0	0	0	0	0	0	1	1
f SA1	0	1	0	1	0	1	1	1
g SA0	0	1	0	0	0	1	0	0
g SA1	1	1	1	1	1	1	1	1
h SA0	0	0	0	0	0	0	1	1
h SA1	1	1	1	1	1	1	1	1
i SA0	0	0	0	0	0	0	0	0
i SA1	1	1	1	1	1	1	1	1

Tabelle 3-2 Wahrheitstabelle mit stuck-at Fehlern

Die grauen Felder in der Wahrheitstabelle zeigen die Fälle an, welche einen anderen Zustand des Ausganges hervorrufen, als es bei einer fehlerfreien Schaltung sein würde. Die Eingangswerte für die grau hinterlegten Fehlerfälle ergeben die gewünschten Testvektoren um die ***stuck-at*** Fehler zu detektieren. Mit Ausnahme der Signale d ***SA1***, e ***SA0*** und f ***SA1***, können alle anderen Fehler mit zwei oder mehr Testvektoren detektiert werden. Deshalb, um 100% Fehlerabdeckung zu erreichen, müssen in jedem Set Testvektoren die Testvektoren 011 und 100 vorkommen.

Um auch noch kurz auf das ***fault collapsing***, die Reduktion von Fehlern, einzugehen können zwei einfache Beispiele herangezogen werden. Man nutzt dabei das Verhalten von Logikgliedern aus. Liegt zum Beispiel ein ***SA0*** Wert an einem Eingang eines UND Gatters an, so kann der Ausgang des Gatters nur ***SA0*** annehmen, egal was am anderen Eingang des Gatters anliegt. Im Falle eines ODER Gatters liegt der Ausgang immer auf ***SA1*** sollte ein Eingang auf ***SA1*** liegen. Der zweite Eingang spielt dabei keine Rolle. Dies wird zur Vereinfachung genutzt. Dabei wird der zweite Eingang ignoriert, wenn der erste Eingangswert einen definierten Ausgangswert zur Folge hat, egal was am anderen Eingang anliegt.

3.2.2 Transistor Faults

Je nach Schaltpegel kann ein Transistor ***stuck-open*** oder ***stuck-short*** sein. Das ***stuck-at*** Fehlermodell kann wegen der hohen Anzahl von CMOS Transistoren nicht das Verhalten von ***stuck-open*** oder ***stuck-short*** Fehlern in CMOS Schaltungen wiedergeben. Zur Verdeutlichung des Problems wird das Beispiel eines CMOS NOR Gatters, siehe Abbildung 3-6, herangezogen.

Nehmen wir an, dass Transistor N2 ***stuck-open*** ist. Wird nun der Eingangsvektor AB = 01 angelegt, sollte der Ausgang auf logisch 0 liegen. Jedoch wird durch den ***stuck-open*** Zustand des Transistors N2 der Ausgang Z von Ground isoliert. Weil die Transistoren P2 und N1 nicht leitend sind, bleibt der Ausgang auf dem vorherigen Logikpegel, entweder 0 oder 1. Um solch einen Fehler zu detektieren sind zumindest zwei Testvektoren notwendig, zum Beispiel AB = 00 → 01. Im fehlerfreien Fall wird durch die Eingangswerte 00 der Ausgang Z = 1 und durch die Eingangswerte 01 der Ausgang Z = 0. Im Fehlerfall jedoch wird beim Übergang der Eingangswerte 00 auf 01 der Ausgangswert Z unverändert auf 1 liegen. Damit verhält sich die Schaltung wie ein Pegel sensitives Latch.

Solch ein ***stuck-open*** Fehler in einer CMOS Schaltung benötigt zumindest zwei Testvektoren zur Detektion, nicht einen einzelnen wie bei einem ***stuck-at*** Fehler.

Stuck-short Fehler produzieren leitende Verbindungen zwischen Vdd und Vss. Wenn zum Beispiel der Transistor N2 ***stuck-short*** ist, entsteht eine Verbindung zwischen Vdd und Vss im Falle der Eingangswerte 00. Dadurch entsteht ein Spannungsteiler, welcher durch die Widerstände der leitenden Transistoren definiert wird. Diese Spannung kann von anderen Logikschaltungen als fehlerhafter Logikpegel interpretiert werden. Solche Fehler werden durch Messungen des Stromes detektiert. Dies nennt man auch IDDQ Testen.

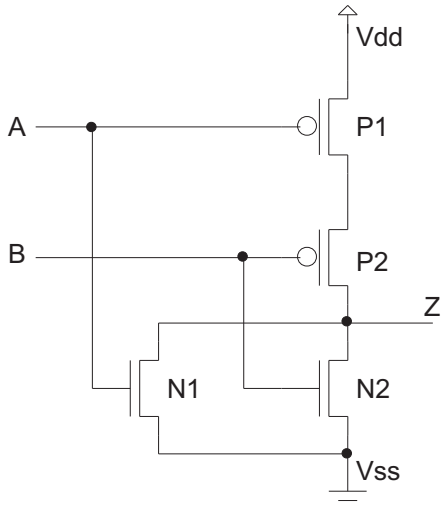


Abbildung 3-6 CMOS NOR

Die Schaltung in Abbildung 3-6 hat 8 (2*4) mögliche Transistor Fehler. Auch hier gibt es Potential zur Vereinfachung. Wenn ein Transistor einer Serienschaltung ***stuck-open*** ist, ist der zweite belanglos. Dies gilt auch für parallele Transistoren, sollte einer davon ***stuck-short*** sein, ist der Zustand des zweiten Transistors für die Funktion irrelevant. Dies ergibt wiederum Potential für ***fault collapsing***.

AB	00	01	10	11
Z	1	0	0	0
N1 stuck-open	1	0	Last-Z	0
N1 stuck-short	IDDQ	0	0	0
N2 stuck-open	1	Last-Z	0	0
N2 stuck-short	IDDQ	0	0	0
P1 stuck-open	Last-Z	0	0	0
P1 stuck-short	1	0	IDDQ	0
P2 stuck-open	Last-Z	0	0	0
P2 stuck-short	1	IDDQ	0	0

Tabelle 3-3 Wahrheitstabelle NOR Gatter mit Fehler

3.2.3 Open und Short Fehler

Fehler in VLSI Bauteilen können auch Unterbrechungen (***opens***) und Kurzschlüsse (***shorts***) in den Verbindungen der Transistoren der Schaltung sein. Unterbrochene Verbindungen verhalten sich wie ***stuck-open*** Transistorfehler, aber auch wie ***stuck-at*** Fehler, je nachdem, welche Verbindungen betroffen sind. In solch einem Falle würde ein Test mit hoher ***stuck-at*** Fehlerabdeckung und Transistor Fehlerabdeckung auch den open Fehler detektieren.

Trotzdem verhalten sich resistive ***open*** Fehler nicht gleich wie Transistorfehler oder ***stuck-at*** Fehler. Sie haben Einflüsse auf die Signallaufzeit.

Ein ***short*** zwischen zwei Elementen wird auch ***bridging fault*** genannt. Diese Elemente können Transistoranschlüsse oder Verbindungen zwischen Transistoren und Gates sein. Wird ein Element gegen Vdd oder Vss kurzgeschlossen, ist dies äquivalent einem ***stuck-at*** Fehler. Wenn jedoch zwei Signalleitungen miteinander kurzgeschlossen werden, sind bridging fault Modelle notwendig.

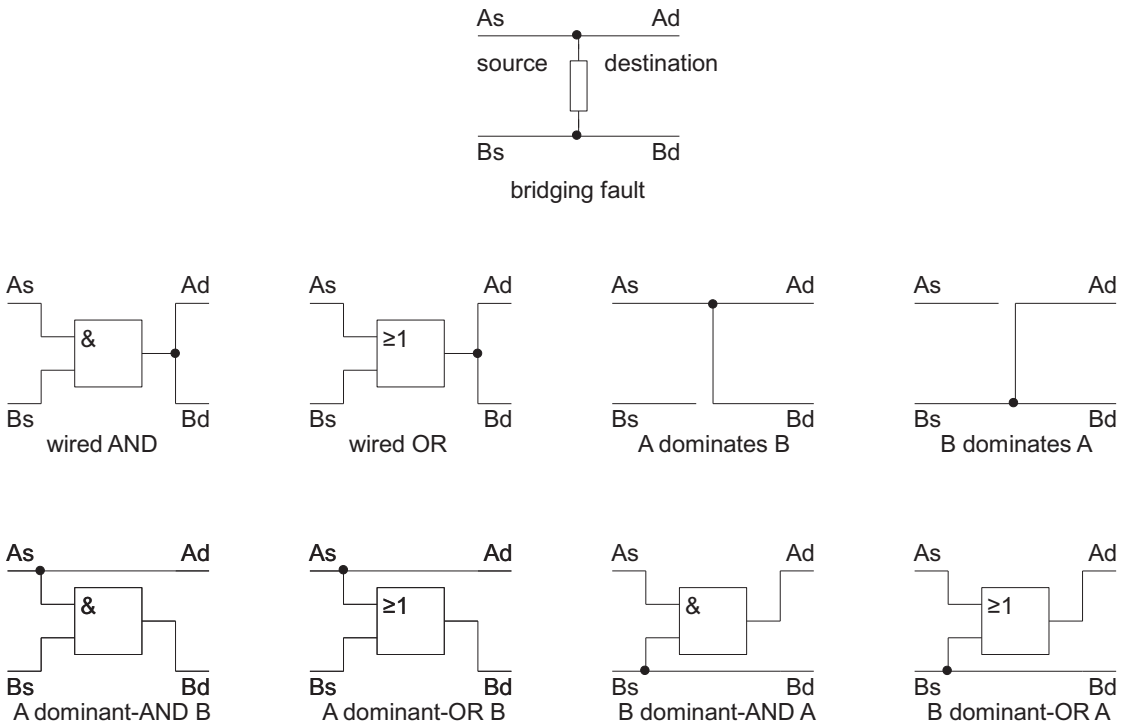


Abbildung 3-7 bridging fault Modelle

Die Verhalten der ***bridging faults*** werden in folgender Wahrheitstabelle beschrieben.

AsBs	0	0	0	1	1	0	1	1
AdBd	0	0	0	1	1	0	1	1
Wired AND	0	0	0	0	0	0	1	1
Wired OR	0	0	1	1	1	1	1	1
A dominates B	0	0	0	0	1	1	1	1
B dominates A	0	0	1	1	0	0	1	1
A dominant-AND B	0	0	0	0	1	0	1	1
B dominant-AND A	0	0	0	1	0	0	1	1
A dominant-OR B	0	0	0	1	1	1	1	1
B dominant-OR A	0	0	1	1	1	0	1	1

Tabelle 3-4 Wahrheitstabelle bridging fault Modelle

Die ***wired AND*** und ***wired OR bridging*** Fehlermodelle wurden ursprünglich für Bipolar Schaltungen entwickelt. Deshalb spiegelt es nicht das ***bridging*** Fehlerverhalten von CMOS Schaltungen wider. Um das Verhalten korrekt zu beschreiben wurde das ***dominant bridging*** Fehlermodell entwickelt. Dabei wird ein Signal zum dominanten und gibt den Pegel des kurzgeschlossenen Pfades an.

Das ***dominant bridging*** Fehlermodell ist schwieriger zu detektieren, weil das fehlerhafte Verhalten nur am dominanten Signal betrachtet werden kann, im Gegensatz zu den beiden Netzen im Falle von ***wired OR*** und ***wired AND bridging*** Fehlermodellen.

Trotzdem zeigt sich durch die Wahrheitstabelle, dass die Testvektoren, welche die ***dominant bridging*** Fehler abdecken, auch garantiert die ***wired AND*** und ***wired OR bridging*** Fehler abdecken.

Bridging Fehler treten in der Praxis häufig auf und können mittels IDDQ Testen detektiert werden. Es hat sich auch gezeigt, dass die meisten ***bridging*** Fehler durch Testvektoren mit hoher ***stuck-at*** Fehlerabdeckung detektiert werden.

3.2.4 Delay Faults und Crosstalk

Fehlerfreies Verhalten von logischen Schaltungen setzt nicht nur die richtige logische Funktion voraus. Zusätzlich sollten die korrekten Logiksignale entlang des Signalpfades innerhalb einer gewissen Zeit anliegen. Ein ***delay*** Fehler führt zu Verspätungen der Signale entlang eines Pfades, bis die Signallaufzeit aus dem definierten Rahmen fällt. Dafür gibt es verschiedene ***delay*** Fehlermodelle. Diese Fehlermodelle werden erst bei kleinen Geometrien (<130 nm) relevant.

Im ***gate-delay*** Fehlermodell und dem ***transition*** Fehlermodell tritt dann ein Fehler auf, wenn die Zeitspanne eines Übergangs vom ***gate*** Eingang zu dessen Ausgang eine definierte Zeitspanne übersteigt. Sollten mehrere gleichzeitige Signalübergänge an den Eingängen eines ***gates*** auftreten, ändern sich die ***gate-delay*** Zeiten empfindlich. Dies ist auf die Aktivierung durch mehrere Lade- und Entladepfade zurückzuführen.

Ein weiteres Modell ist das ***path-delay*** Fehlermodell. Dieses berücksichtigt die gesamte Signallaufzeit entlang eines Signalpfades, also die Summe aller ***gate-delays*** entlang eines Pfades. Deshalb ist das ***path-delay*** Fehlermodell praktisch besser anwendbar als das ***gate-delay*** Fehlermodell. Ein großes Problem stellt die hohe Anzahl an möglichen Signalpfaden in einer Schaltung dar. Die Anzahl der Signalpfade ist im schlimmsten Fall die exponentielle Anzahl der Leitungen. Meistens ist es nicht möglich alle ***path-delay*** Fehler zu ermitteln, die zur Testgenerierung und Fehlersimulation dienen.

Delay Fehler benötigen auch definierte Testvektoren um einen Signalpfad durch die logische Schaltung zu finden und eine Signaländerung entlang dieses Pfades zu bewirken, um das ***path-delay*** zu messen. Als Beispiel dient die folgende Schaltung. Die Werte in den UND und ODER Blöcken geben die ***gate-delays*** der einzelnen Glieder an. Die beiden Testvektoren v1 und v2 werden benutzt, um das ***path-delay*** von x2 nach y zu ermitteln. Die Signalübergangszeit bei v1 auf v2 ist exakt bei t = 0. Durch das ***path-delay*** des Pfades wird der Ausgang y nach t = 7 seinen Zustand ändern.

	v1	v2
x1	0	0
x2	0	1
x3	1	1

Tabelle 3-5 path-delay Testvektoren

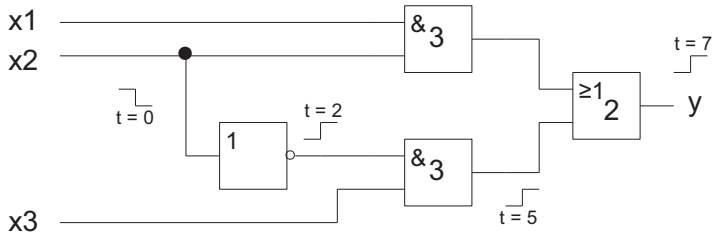


Abbildung 3-8 path-delay Fehlertest

Solche Messungen benötigen high-speed high-precision Tester.

Zusätzlich ist zu beachten, dass nicht ausschließlich ***gate-delays*** für das ***path-delay*** verantwortlich zeichnen. Auch Signalleitungen verursachen, je nach Layout, gewisse Verzögerungen. Das gesamte ***path-delay*** ist somit die Summe aller Verzögerungen.

Je kleiner die Technologien werden, umso mehr steigt die ***cross-coupling*** Kapazität zwischen Verbindungen. Dies führt zu sogenannten ***crosstalk*** Effekten. Diese können zu Fehlverhalten in Chips führen. ***Crosstalk*** Effekte können in zwei Kategorien unterteilt werden: und zwar in ***crosstalk-glitches*** und ***crosstalk-delays***.

Crosstalk-glitches sind Spannungspulse, die über unerwünschte Kopplungen von einer Störquelle auf eine Leitung übertragen werden. Die Amplitude der Störung ist dabei von der Kopplungskapazität und der Kapazität zu Ground abhängig.

Crosstalk-delays sind Signalverzögerungen, die auch durch parasitäre Kopplungen zwischen den Leitungen auftreten. Diese Verzögerungen wirken nun zusätzlich zu den ***gate-delays*** und den ***interconnect-delays*** auf das gesamte ***path-delay*** ein.

Um diese Effekte simulieren zu können, wurden physikalische Design- und Analysetools entwickelt. Diese helfen bei der Reduktion von ***crosstalk*** Problemen. Wegen der ***crosstalk*** Effekte ist es dringend notwendig, weitere Testtechniken zu entwickeln, um Produktionsfehler, die auf diesen Effekten basieren, detektieren zu können.

3.2.5 Pattern Sensitivity und Coupling Faults

Produktionsdefekte haben ein großes Spektrum von Möglichkeiten, sich in Schaltungen zu manifestieren. Es gibt mehrere Fehler, die mit den bis jetzt besprochenen Fehlermodellen nicht abgedeckt werden können. In sehr dichten Speicherblöcken, wie zum Beispiel RAM Blöcken, kann es durch Fehler zu Beeinflussungen durch Speicherinhalte benachbarter Zellen kommen, welche als ***pattern-sensitivity*** Fehler bezeichnet werden. Eine Wertänderung in Speicherzellen, die auch andere Zellen zu einer Wertänderung anregen, werden ***coupling*** Fehler genannt. Dadurch wird es notwendig dass, wenn Speicher getestet werden, zu den Vektoren für ***stuck-at*** Fehler zusätzliche Testvektoren für ***pattern-sensitivity*** und ***coupling*** Fehler hinzugefügt werden. Für Speichertests wurden eigene Testalgorithmen entwickelt.

Der effizienteste Testalgorithmus für RAM Tests bezüglich Testzeit und Fehlerabdeckung ist der March LR Algorithmus. Dieser Algorithmus hat eine Testzeit von 16N, wobei N für die Anzahl der Speicheradressen steht. Der Algorithmus ermöglicht die Abdeckung von pattern-***sensitivity*** Fehlern, ***coupling*** Fehlern und ***bridging*** Fehlern in einem RAM Block. In Wort orientierten Speicherblöcken muss eine ***background data sequence*** (***BDS***) hinzugefügt werden, um alle Fehler in jedem Wort des Speichers abzudecken.

Der ***March LR Algorithmus*** mit ***BDS*** wird in folgender Tabelle anhand eines RAM Speichers mit 2-bit Worten gezeigt. Generell ist die Anzahl der ***BDS*** = log2(K) + 1, wobei K für die Anzahl der Bits pro Wort steht.

Test Algortihmus	March Testsequenz
March LR ohne BDS	↕(w0); ↓(r0,w1); ↑(r1,w0,r0,r0,w1); ↑(r1,wo); ↑(r0,w1,r1,r1,w0); ↑(r0)
March LR mit BDS	↕(w00); ↓(r00,w11); ↑(r11,w00,r00,r00,w11); ↑(r11,w00); ↑(r00,w11,r11,r11,w00); ↑(r00,w01,w10,r10); ↑(r10,w01,r01); ↑(r01)

Tabelle 3-6 March LR RAM Testalgorithmus

w0 = write 0 (oder all 0's); r1 = read 1 (oder all 1's); ↑ = Adresse erhöhen; ↓ = Adresse erniedrigen; ↕ = Adressänderung

Die physikalischen Adressen sind in den meisten Fällen nicht mit den logischen Adressen ident. Dies muss bei der Generierung der Testvektoren entsprechend berücksichtigt werden.

3.3 Produktions- und Funktionstest

3.3.1 Produktionstest

Bei der Produktion von Chips kann nicht garantiert werden, dass die Funktion aller Chips, nach dem letzten Schritt der Fertigung, auch wirklich gewährleistet ist. Dies erfordert unterschiedliche Testmethoden. Diese umfassen Testmethoden direkt auf dem Wafer und nach dem Packaging.

Die Primärziele von Produktionstests sind die maximale Testabdeckung und die minimale Testzeit am ASIC Tester zu erzielen. Ein weiterer wichtiger Punkt ist, dass der Entwicklungsaufwand dieser Tests, möglichst gering gehalten werden soll. Es wurden standardisierte Testmethoden entwickelt, um diesen Aufwand gering und kalkulierbar zu halten. Um gute Testbarkeit zu erzielen, wird zusätzliche, automatisch generierte, Hardware in die Schaltung eingefügt. Diese DFT Strukturen sind, je nach Anspruch, unterschiedlich ausgeführt, um immer die optimale Lösung für die zu testende Logik darzustellen.

3.3.2 Funktionstest

Funktionstests dienen dazu, um vorher festgelegte Spezifikationen zu überprüfen. Die ersten Funktionstests werden auch am Wafer, durch ***Probing*** durchgeführt.

Im Gegensatz zu den Produktionstests, werden bei Funktionstests die Chips wie in ihrer späteren Umgebung überprüft. Deshalb können diese Tests sehr lange dauern und sind daher sehr kostenintensiv. Dies ist der Fall weil interne Schaltungselemente nur sehr schwer von außen zugänglich sind. Solche Funktionstests sind sehr stark vom Design abhängig. Um solche Tests entwickeln zu können, muss eine detaillierte Kenntnis des Designs vorhanden sein. Da solche Tests sehr spezifisch sind, ist es nicht möglich, sie automatisch zu generieren. Dies ist der Grund, warum für jeden ASIC eine eigene Teststrategie entwickelt werden muss. Einer Wiederverwendung in anderen Schaltungen ist nicht möglich.

Eine gewisse Beschleunigung von solchen Tests, kann durch die ***stop at fail*** Methode erreicht werden. Dabei wird der gesamte Funktionstest sofort beendet, sobald ein Test nicht bestanden wird. Anfangs werden bei solchen Tests Ausfälle durch Kurzschluss- und Kontaktmessung entstehen.

3.4 Automatische Test Pattern Generierung (ATPG)

Ein kompletter ATPG Algorithmus, der sogenannte ***D-Algorithmus***, wurde 1966 veröffentlicht. Der D-Algorithmus verwendet logische Werte um beide, die Werte der fehlerhaften- und der fehlerfreien Schaltung, gleichzeitig darzustellen. Er kann für jeden ***stuck-at*** Fehler einen Test generieren, solange ein Test für diesen Fehler existiert. Auch wenn der Rechenaufwand des ***D-Algorithmus*** sehr hoch ist, ist seine theoretische Bedeutung allgemein anerkannt.

Der nächste Meilenstein in ATPG war der ***PODEM*** Algorithmus 1981, welcher den Wertebereich der Eingänge aufgrund von Simulationen erfasst, um damit die Berechnungseffizienz zu steigern. Seit damals wurden ATPG Algorithmen zu immer wichtigeren Themen für Forschung und Entwicklung. Es wurden viele Verbesserungen erzielt und viele ATPG Tools wurden auf den Markt gebracht. Zum Beispiel ***FAN*** (1983) und ***SOCRATES*** (1988) waren wichtige Beiträge in Sachen Beschleunigung des ATPG Prozesses.

Den aktuellen ATPG Tools liegt meist eine zufällige Reihe von Test Pattern zu Grunde. Durch Fehlersimulation wird festgestellt, wie viele potentielle Fehler detektiert wurden. Zusätzlich zu den Resultaten der Fehlersimulation können zusätzliche Testvektoren generiert werden, um sogenannte ***hard-to-detect*** Fehler zu erfassen, um die gewünschte Fehlerabdeckung zu erzielen.

Das ***International Symposium on Circuits and Systems*** (ISCAS) kündigte kombinatorische Benchmark-Schaltungen (1985) und sequentielle Benchmark-Schaltungen (1989) an, um die ATPG Forschung und Entwicklung der internationalen Testvereinigung zu unterstützen. Das Hauptproblem von großen Logikschaltungen war die Identifikation von nicht detektierbaren Fehlern.

In den 1990er Jahren wurden sehr schnelle ATPG Systeme entwickelt. Diese ermöglichen eine Beschleunigung um einen Faktor 5 gegenüber dem ***D-Algorithmus*** bei 100% Fehlerabdeckung.

Daraus resultiert, dass die ATPG für kombinatorische Logik kein Problem mehr darstellt. ATPG für sequentielle Logik ist weiterhin schwierig, da eine Zustandsfolge angelegt werden muss, um den Fehler bis zu den Ausgängen durchzuschleifen, wo er dann beobachtet und detektiert werden kann.

Für große sequentielle Schaltungen ist es somit schwierig 100% Fehlerabdeckung zu erzielen, da die Rechenzeit zu groß wird. Deshalb müssen DFT Techniken angewandt werden.

3.5 Fehlerklassen

ATPG Tools ermöglichen die Handhabung von unterschiedlichen Fehlern. Sie ermöglichen die Zuordnung von Fehlern zu unterschiedlichen Fehlerklassen, je nach Nachweisbarkeit der Fehler. Fehlerklassen sind in Kategorien unterteilt.

3.5.1 Detektierte Fehler (detected)

Diese können in drei Fehlerklassen unterteilt werden:

- ***Detected robustly*** Fehler werden durch ***path delay*** Testen oder Fehlersimulation erkannt. Während der ATPG wird zumindest ein Pattern, welches solch einen Fehler auslöst, erzeugt.
- ***Detected by simulation*** Fehler werden durch Pattern Generierung und Simulation dieser Pattern detektiert.
- ***Detected by implication*** Fehler müssen nicht direkt von Pattern detektiert werden, da diese Fehler aus dem Schieben der ***scan chains*** resultieren. Diese Fehler treten entlang der ***scan chain*** Pfaden auf und beinhalten Taktanschlüsse und Datenanschlüsse der Scan Zellen.

3.5.2 Vielleicht detektierte Fehler (possibly detected)

Diese können in zwei Fehlerklassen unterteilt werden:

- ***ATPG possibly detected*** Fehler treten auf, wenn die fehlerhafte Schaltung als simulierten Ausgangswert X aufweist. Die fehlerfreie Schaltung hingegen liefert den Ausgangswert logisch 0 oder logisch 1. Die Analyse zeigt, dass der Fehler nicht garantiert detektiert werden kann, nur möglicherweise.
- Die ***Not analyzed-possibly detected*** Fehler treten auf, wenn die fehlerhafte Schaltung als simulierten Ausgangswert X aufweist. Die fehlerfreie Schaltung hingegen liefert den Ausgangswert logisch 0 oder logisch 1. Die Analyse des Fehlers war nicht eindeutig.

In beiden Fällen kann ein Wert X am zu testenden Chip nicht generiert werden. Dadurch ist der Ausgangswert entweder 1 oder 0. Wenn der Ausgang 1 ist und der Erwartungswert 0, kann der Fehler detektiert werden. Wenn jedoch der Ausgang 0 und der Erwartungswert 0 ist, kann der Fehler nicht detektiert werden.

3.5.3 Nicht detektierbare Fehler (undetectable faults)

Diese Kategorie beinhaltet Fehler, welche weder durch ATPG, funktionelle Testgenerierung, parametrische Testgenerierung oder irgendeine andere Testgenerierung erkennbar sind. Normalerweise werden solche Fehler aus der Gesamtfehlerabdeckung entfernt.

- Die ***undetectable unused*** Fehlerklasse beinhaltet Fehler, die an nicht verwendeten Ausgängen auftreten. Solche Fehler haben keine Auswirkung auf die interne Logik des Designs oder die Fehlersimulation.
- Die ***undetectable tied*** Fehlerklasse beinhaltet Fehler, die an Anschlüssen auftreten, welche auf logisch 0 oder logisch 1 gezogen sind. Dies sind meist nicht verwendete Eingänge. Ein ***stuck-at*** 1 Fehler an einem Anschluss, der auf logisch 1 gezogen ist, kann nicht detektiert werden und hat keine Auswirkung auf die Schaltung. Ein ***stuck-at*** 0 Fehler, welcher an einem Anschluss der auf logisch 0 auftritt, hat auch keinen Effekt.
- Die ***undetectable blocked*** Fehlerklasse beinhaltet Fehler, welche an Punkten auftreten, die in Beobachtbarkeit und Steuerbarkeit durch redundante Fehler eingeschränkt sind. Solche Fehler sind Begleitfehler von ***undetectable redundant*** Fehlern.
- Die ***undetectable redundant*** Fehlerklasse beinhaltet Fehler, für die redundante Logikpfade zur Verfügung stehen. Diese Fehler können nicht detektiert werden, da diese von der fehlerfreien redundanten Logik ausmaskiert werden. Dadurch können einzelne Fehler in solchen redundanten Designs nicht detektiert werden.

3.5.4 ATPG untestbare Fehler (ATPG untestable faults)

Diese Kategorie beinhaltet Fehler, welche nicht intern untestbar sondern nur mittels ATPG Ansatz nicht testbar sind. Diese Fehler können testbar sein, wenn andere Methoden, zum Beispiel funktionelle Tests, verwendet werden. Diese Kategorie hat nur eine Klasse.

- ***ATPG untestable, not detected*** Fehler können nicht detektiert werden, da die benötigten Pattern eine ATPG Beschränkung verletzen würden. Auch wenn Fehler sequentieller Logikteile oder Latches mittels einfacher ATPG Methoden nicht detektierbar sind, handelt es sich um untestbare Fehler.

3.5.5 Nicht detektierte Fehler (not detected faults)

Diese Fehler treten auf, wenn die Fehleranalyse frühzeitig beendet wird. Solche abgebrochenen Analysen können durch Erreichen des Iterationslimits oder durch zu komplexe Designs für die ATPG Algorithmen auftreten.

- Die ***not controlled*** Fehlerklasse beinhaltet Fehler, welche der ATPG Algorithmus nicht darstellen kann.
- Die ***not observed*** Fehlerklasse beinhaltet Fehler, welche zwar steuerbar, aber nicht in eine ***scan chain*** oder an einen Ausgang geführt werden können, um dort beobachtbar zu sein.

4 FPGA Implementierung

4.1 Aufbau FPGA Board

4.1.1 FPGA Board

Das FPGA Board (siehe Abbildung 4-1 und 4-2) ist ein bereits verwendetes der Firma Dialog Semiconductor. Es bietet Speichermöglichkeiten, unterschiedliche Verbindungen und weitere Hardware.

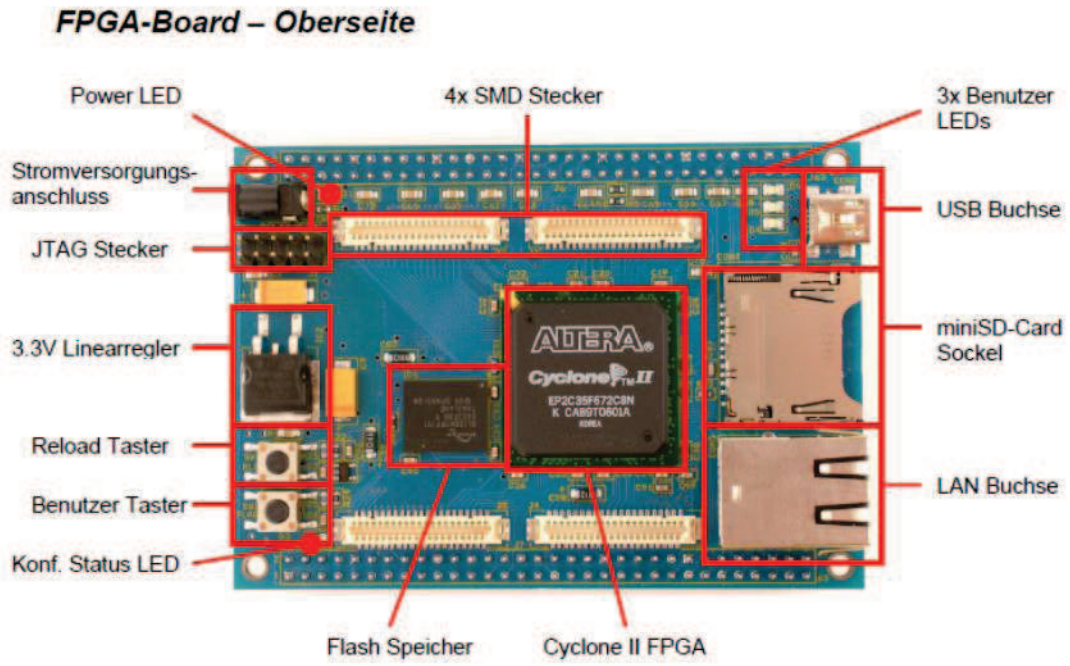


Abbildung 4-2 FPGA Board Oberseite

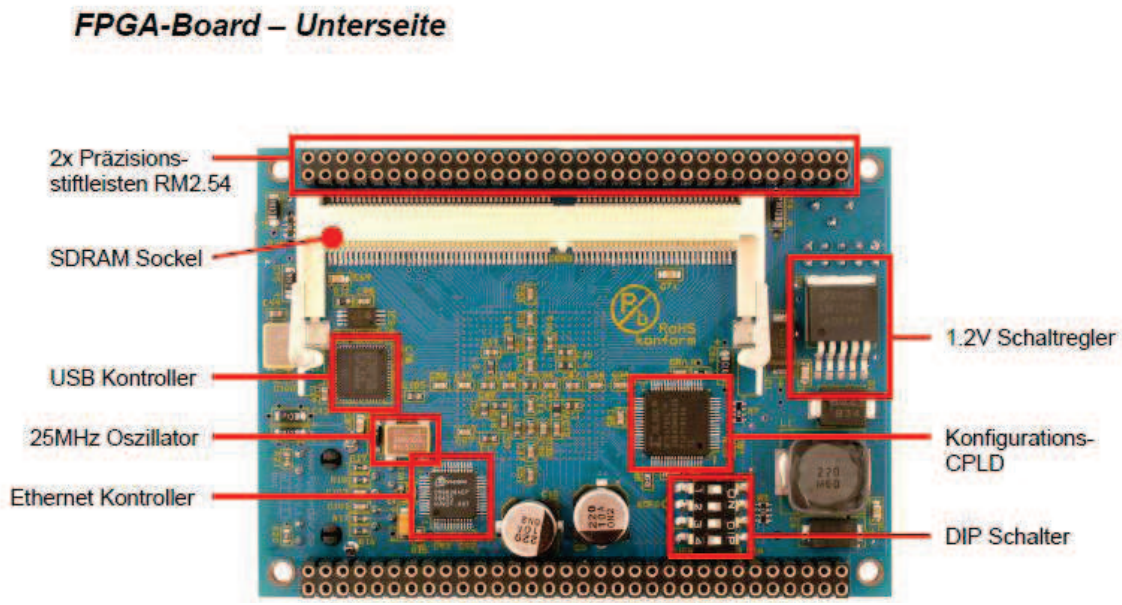


Abbildung 4-1 FPGA Board Unterseite

4.1.2 Extention Board

Das Extention Board (siehe Abbildung 4-3 und 4-4) bietet drei Konnektoren, welche zum Beispiel zum Debugging eines Controllers verwendet werden können. Zusätzlich erweitert das Extention Board den Speicher durch 2 RAM Blöcke.

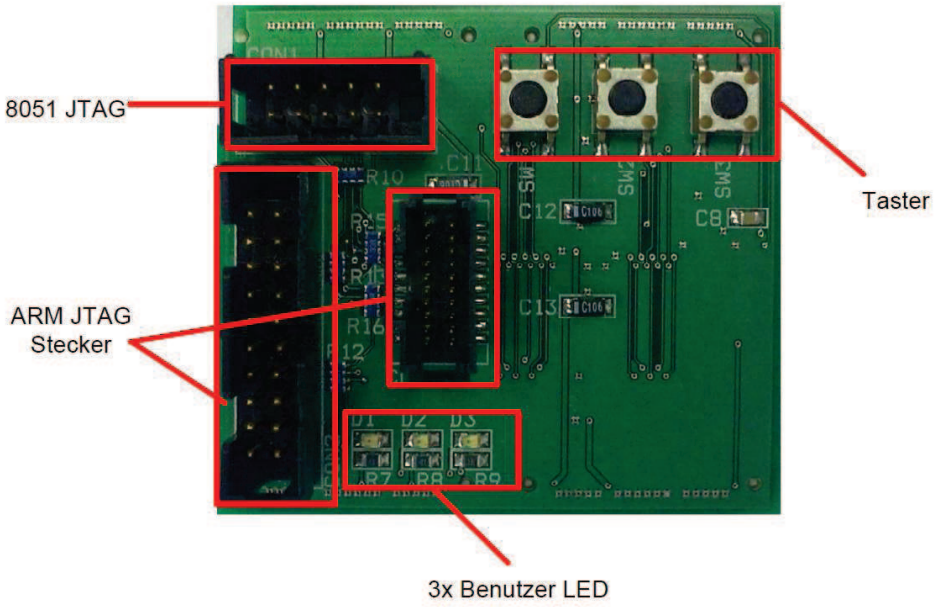


Abbildung 4-3 Extention Board Oberseite

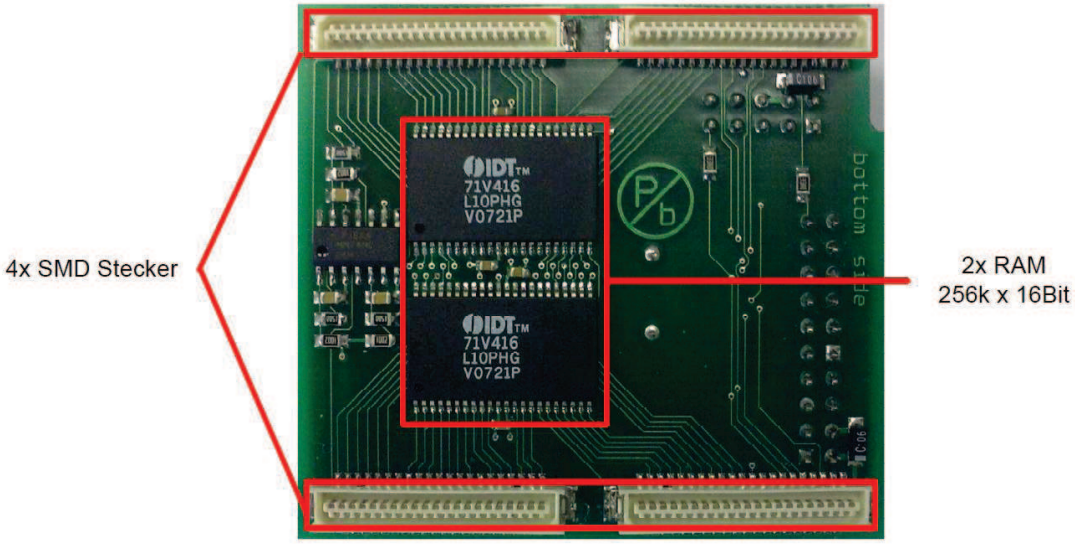


Abbildung 4-4 Extention Board Unterseit

4.2 Gliederung der Komponenten

4.2.1 Modulare Gliederung

Das gesamte Hardwaredesign lässt sich schematisch in folgender Grafik überblicken. Die Generiereinheiten und die Vergleichseinheiten können, je nach Konfiguration, 1 mal bis maximal 10 mal vorhanden sein, was die Variierung der Eingangs- und Ausgangszahl möglich macht.

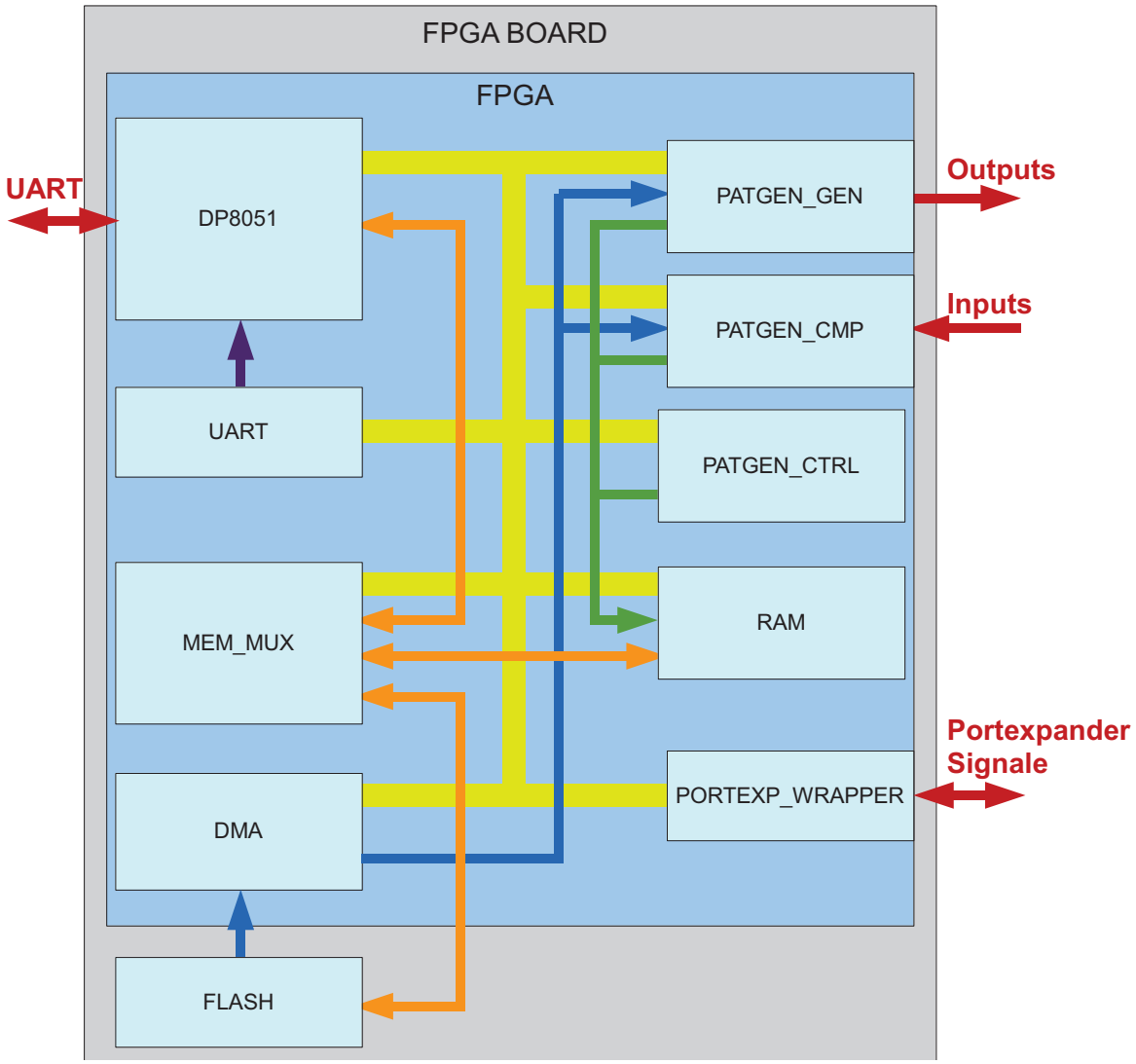


Abbildung 4-5 FPGA Hardwareübersicht

Agenda:	SFR Bus	-	Gelb
	Speicherzugriff	-	Orange
	DMA Zugriff	-	Blau
	DMA Fehlerfall	-	Grün

4.2.2 Tabellarische Gliederung

Komponenten:	Files:	Aufgabe:	Interfaces / Ports:
DP8051	----------	Prozessor	----------
Memory Mux	mem_mx.sv	Zugriff auf Speicher	patgen_ctrl_sys_if patgen_gen_sys_if patgen_cmp_sys_if dp_prgmem_if dp_imem_if dp_xmem_if ram128x08_mem_if ram128x16_mem_if ram128x24_mem_if ram1792x08_mem_if ram256x08_mem_if
RAM128x08	ram_128x08.sv TM035FSSRAM128x08.v ram128x08_mem_if.sv	Fehlerfall: Generate Value	ram128x08_mem_if
RAM128x16	ram_128x16.sv TM035FSSRAM128x16.v ram128x16_mem_if.sv	Fehlerfall: Compare Value	ram128x16_mem_if
RAM128x24	ram_128x24.sv TM035FSSRAM128x24.v ram128x24_mem_if.sv	Fehlerfall: Vector Counter	ram128x24_mem_if
RAM49152x08	ram_49152x08.sv TM035FSSRAM49152x08.v Ram49152x08_mem_if.sv	Program Memory	Ram49152x08_mem_if
RAM256x08	ram_256x08.sv TM035FSSRAM256x08.v Ram256x08_mem_if.sv	Internal Memory	ram256x08_mem_if
Patgen CTRL	patgen_ctrl.sv patgen_ctrl_sfr_if.sv patgen_ctrl_sys_if.sv	Pat.Gen. Kontrolleinheit	patgen_ctrl_sfr_if patgen_ctrl_sys_if
Patgen GEN	patgen_gen.sv patgen_gen_core.sv patgen_gen_sfr_if.sv patgen_gen_sys_if.sv	Pat.Gen. Generiereinheit	patgen_gen_sfr_if patgen_gen_sys_if
Patgen CMP	patgen_cmp.sv patgen_cmp_core.sv patgen_cmp_sfr_if.sv patgen_cmp_sys_if.sv	Pat.Gen. Vergleichseinheit	patgen_cmp_sfr_if patgen_cmp_sys_if
Patgen DMA	patgen_dma.sv patgen_dma_sfr_if.sv patgen_dma_sys_if.sv	Direkter Speicherzugriff	patgen_dma_sfr_if patgen_dma_sys_if
Patgen UART	patgen_uart.sv patgen_uart_sfr_if.sv patgen_uart_sys_if.sv	UART Einheit	patgen_uart_sfr_if patgen_uart_sys_if
Patgen Wrapper	patgen_portexp_wrapper.sv patgen_portexp_wrapper_sfr_if.sv patgen_portexp_wrapper_sys_if.sv	Wrapper für Portexpander Logik	patgen_portexp_wrapper_sfr_if patgen_portexp_wrapper_sys_if

Patgen Top	patgen_dig_top.sv	Top Level Verbindungen	ram128x08_mem_if ram128x16_mem_if ram128x24_mem_if ram1792x08_mem_if ram256x08_mem_if patgen_ctrl_sfr_if patgen_ctrl_sys_if patgen_gen_sfr_if patgen_gen_sys_if patgen_cmp_sfr_if patgen_cmp_sys_if patgen_dma_sfr_if patgen_dma_sys_if patgen_uart_sfr_if patgen_uart_sys_if patgen_portexp_wrapper_sfr_if patgen_portexp_wrapper_sys_if

Tabelle 4-1 Hardware Übersicht

Agenda:	Erweiterte Einheiten	-	Blau unterlegt
	Übernommenen Einheiten	-	Grün unterlegt
	Neue Einheiten	-	Rot unterlegt

Die unterschiedlichen Einheiten wurden von einer bereits beendeten Projektarbeit übernommen, adaptiert oder gänzlich neu entwickelt. Die Veränderungen an den bestehenden Einheiten wurden durch den DMA Zugriff auf den Flash-Speicher notwendig. Dazu wurden die Systeminterfaces der einzelnen Einheiten erweitert und die Vergleichseinheit wurde auf ein Doppelbuffersystem umgebaut. Der Programmspeicher musste auf 49 kByte vergrößert werden, da die für das System entwickelte Software einen sehr großen Speicher benötigt.

Die Direct Memory Access Einheit wurde neu implementiert, sowie der Portexpander Wrapper und die UART Einheit.

4.3 Verhaltensbeschreibung der Komponenten

4.3.1 Gegebene Einheiten

4.3.1.1 Mikrocontroller (Digital Core Design)

Der DP8051 Controller ist ein geschwindigkeitsoptimierter Hochleistungs - Controller. Beim Design von Digital Core Design wurde besonders auf Rechenleistung im Vergleich zur Leistungsaufnahme geachtet. Dieses Verhältnis wir durch eine Leistungsmanagement Einheit (PMU) verbessert.

Der DP8051 ist 100% binär kompatibel mit dem Standard 8051 8-bit Mikrocontroller.

Der DP8051 weist eine Pipeline RISC (Reduced Instruction Set Computing) Architektur auf, welche es ermöglicht, 85-200 Millionen Instruktionen pro Sekunde abzuarbeiten. Diese Rechenleistung wird genutzt, um einen Vorteil im Bereich von Applikationen mit geringem Leistungsverbrauch zu erzielen. Dies kann dadurch erreicht werden, dass der Core bei gleicher Rechenleistung über 10 mal langsamer getaktet werden kann als in der originalen Implementation des 8051 Controllers.

DP8051 Features:

- 100 % Software kompatibel mit Standard 8051
- 24 mal schnellere Multiplikation
- 12 mal schnellere Addition
- 256 bytes interner Datenspeicher
- 64k bytes interner oder externer Programmspeicher
- 16M bytes externer Datenspeicher
- Einfache Verbindung mit Speicher
- Interface für Special Function Registers

Peripherie:

- 2-wire DoCD Debug Einheit
- Power Management Einheit
- Erweiterter Interrupt Controller
- Vier 8-bit I/O Ports
- Drei 16-bit Timer / Counter
- Full-duplex Serielle Schnittstelle
- I2C Bus Controller
- 32-bit Fixkomma arithmetischer Prozessor

(Digital Core Design)

4.3.1 Eigene Einheiten

4.3.1.1 Patterngenerator Control Unit

Diese Einheit wurde zur Steuerung der Generier- und Vergleichseinheiten implementiert. Sie dient auch dazu, die einzelnen Einheiten mit einem Zeitbasis zu versorgen. Diese Zeitbasis legt die Anzahl der Pattern, die pro Sekunde generierbar sind, fest. Durch einen Prescaler (7 Bit) und einen Teiler (8 Bit) kann ein beliebiger Teilungsfaktor im Bereich von 1 bis 32385 eingestellt werden. Der Teiler gibt zusätzlich die Skalierung der einzelnen Pattern und somit die Auflösung der einzelnen Pattern an.

Der Wert des Prescalers wird über das Special Function Register *PRDIV*, der des Teilers über das Special Function Register *DIV* eingestellt.

Die Kontrolleinheit wird auch dafür verwendet, im Falle eines auftretenden Fehlers, den erwarteten Wert und den tatsächlich erhaltenen Wert in einem Special Function Register abzulegen.

Zusätzlich wurden in dieser Einheit noch 2 Zähler realisiert. Einer davon dient zum Zählen der durchlaufenen Pattern (Vectorcounter) und der andere zum Zählen der detektierten Fehlerfälle (Errorcounter *ERRCNT*).

Im Hinblick auf weiterführende Entwicklungen ist es möglich, den Vectorcounter mit einem Wert, der mittels SFR Zugriff eingestellt werden kann, im Vorhinein zu setzen. Dies ermöglicht in Zukunft das Starten des Generators von einem gewünschten Testpattern aus.

Registername:	**Funktion:**
DIV	Einstellung des Dividers
PRDIV	Einstellung des Predividers
ERRCNT	Errorcounter
ERRnVEC	Erwartungswert im Fehlerfall
ERRnGOT	Empfangener Wert im Fehlerfall
CTRL	Kontroll Register
STAT	Status Register
PAGESEL	Pageselect Register
UNITNR	Anzahl der Einheiten

Tabelle 4-2 Kontrolleinheit SFR Übersicht

Register:

DIV							
7	6	5	4	3	2	1	0
DIV							

DIV - Wert des Teilers

Der Teiler stellt die Auflösung der Pattern ein. Je größer der Teiler, desto genauer werden die einzelnen Testpattern aufgelöst. Jedoch wird die Frequenz der Pattern durch Erhöhung der Auflösung verringert.

PRDIV							
7	6	5	4	3	2	1	0
EN	PRDIV						

PRDIV - Wert des Predividers

EN - Enable des Predividers

Wenn Pattern mit niedrigeren Frequenzen generiert werden sollen, muss die Taktfrequenz vorgeteilt werden. Wichtig dabei ist, darauf zu achten, dass der Predivider erst dann erhöht wird, wenn die gewünschte Patternfrequenz nicht mehr alleine durch den Teiler eingestellt werden kann. Dies ermöglicht eine höhere Auflösung der Pattern.

ERRCNT							
7	6	5	4	3	2	1	0
ERRCNT							

ERRCNT - Wert des Errorcounters

In diesem SFR ist der aktuelle Wert des Errorcounters gespeichert.

ERRnVEC							
7	6	5	4	3	2	1	0
EEXP3		EEXP2		EEXP1		EEXP0	

EEXP0 – EEXP3... Erwartungswerte der Vergleichseinheit, die den Fehler detektiert

Dieses Register hält die erwarteten Werte der Compareeinheit, in der ein Fehler detektiert wurde.

ERRnGOT							
7	6	5	4	3	2	1	0
EGOT3		EGOT2		EGOT1		EGOT0	

EGOT0 – EGOT3- Tatsächlich erhaltene Werte der Vergleichseinheit, die den Fehler detektiert

Dieses Register hält die empfangenen Werte der Vergleichseinheit, in der ein Fehler detektiert wurde.

In diesem SFR ist der aktuelle Wert des Errorcounters gespeichert.

CTRL							
7	6	5	4	3	2	1	0
ni						RST	STRT

STRT - Starts Pattern Generator

RST - Resets Pattern Generator

Durch das Setzen des Start Bits startet der Patterngenerator mit den voreingestellten Parametern, die Patternerzeugung und der Patternvergleich. Durch das Setzen des Reset Bits können alle Einheiten auf deren Resetwerte zurückgesetzt werden.

STAT							
7	6	5	4	3	2	1	0
ni			CPFULL	CBFULL	GPFULL	GBFULL	RUN

RUN - Shows that Generator is running

GBFULL - Generate Buffer full Bit

GPFULL - Generate Pattern Buffer full Bit

CBFULL - Compare Buffer full Bit

CPFULL - Compare Pattern Buffer full Bit

Das Status Register gibt den momentanen Status des Patterngenerators an. Es zeigt, ob der Patterngenerator läuft und welche der Buffer des Doppelbuffersystems momentan voll bzw. leer sind.

PAGESEL							
7	6	5	4	3	2	1	0
ni				PAGESEL			

PAGESEL - Pagenummer

Da die Anzahl der maximalen Special Function Register durch die Adressierung des DP8051 beschränkt wird, musste bei einer Anzahl von mehr als einem Ausgangsmodul oder mehr als einem Eingangsmodul eine Lösung gefunden werden. Deshalb wurde ein Multipage System implementiert, welches über ein Special Function Register die verschiedenen Pages selektieren kann. Je nach Einstellung durch das *PAGESEL* Register kann die jeweilige Page selektiert und beschrieben werden.

Die Anzahl der Pages wird ebenfalls durch die Konfigurationsdatei, die SFR Liste, eingestellt. Dabei verändert sich der SFR Multiplexer, welcher die Selektierung und Beschreibung der Pages durchführt. Die Anzahl der maximal verwendbaren Pages ist auf 10 beschränkt. Dadurch ist es möglich, maximal 10 Generiereinheiten und 10 Vergleichseinheiten zu verwalten.

Wird das *PAGESEL* Register mit 8x00 beschrieben, kann auf Page 0 zugegriffen werden, wenn *PAGESEL* mit 8x09 beschrieben wird, auf Page 9, welche die letzte verwaltbare Page darstellt.

UNITNR							
7	6	5	4	3	2	1	0
PCUNIT				PGUNIT			

PGUNIT - Anzahl der Generiereinheiten

PCUNIT - Anzahl der Compareeinheiten

Dieses Register dient zur Einstellung der Anzahl der Generier- und Vergleichseinheiten. Diese Register wurden im Hinblick auf zukünftige Erweiterungen (z.B.: DMA – Einheit) notwendig.

4.3.1.2 Patterngenerator Generate Unit

Diese Einheit ist für die Generierung der Test Pattern und deren Ausgabe verantwortlich. Pro Einheit stehen 4 Ausgänge zur Verfügung, welche durch einzelne Register einzeln konfigurierbar sind. Damit können die Zeiten für die Flanken der Signale, die Art der Signale und auch die Ausgänge einzeln freigeschaltet werden. Durch Letzteres ist es auch möglich, einen Tri-State Ausgang zu simuliernen.

Zusätzlich ist zu erwähnen, dass bis zu 10 Einheiten mit je 4 Ausgängen durch einfaches Ändern eines Feldes in der SFR Liste und der Pinliste automatisch erzeugbar sind.

n…Anzahl der Generiereinheiten

Konfiguration der Einheit mittels Special Function Register:

Registername:	**Funktion:**
PGnCMPF0 - PGnCMPF3	Einstellung des Startpunktes der ersten Flanke
PGnCMPS0 - PGnCMPS3	Einstellung des Startpunktes der zweiten Flanke
PGnCFG	Einstellung des Typs der Kanäle
PGnCTRL	Register zum Freischalten der Ausgänge
PGnVEC	Register, das den Ausgabewert hält
PGnSTAT	Status Register

Tabelle 4-3 Generiereinheit SFR Übersicht

Register:

PGnCMPF0 - PGnCMPF3							
7	6	5	4	3	2	1	0
CMPF							

CMPF - Startpunkt der ersten Flanke

Diese Register dienen zum Einstellen der ersten Flanke, ab welcher die Pattern übernommen werden. Die Auflösung der Pattern und damit auch die Anzahl der auswählbaren Möglichkeiten hängen von der Voreinstellung im Teiler ab. Je größer dieser ist, desto feiner können die Pattern eingestellt werden.

PGnCMPS0 - PGnCMPS3							
7	6	5	4	3	2	1	0
CMPS							

CMPS - Startpunkt der zweiten Flanke

Diese Register dienen zum Einstellen der zweiten Flanke, bis zu der die Pattern anliegen. Danach wird je nach Einstellung des Typs der weitere Zustand ausgegeben. Die Auflösung der Pattern und damit auch die Anzahl der auswählbaren Möglichkeiten hängen von der Voreinstellung im Teiler ab. Je größer dieser ist, desto feiner können die Pattern eingestellt werden.

PGnCFG							
7	6	5	4	3	2	1	0
TYP3		TYP2		TYP1		TYP0	

TYP0 – TYP3 - Einstellung des Pattern Typs für jede Generiereinheit

TYP0 – TYP3	**Typ:**
00	Non return to zero.
01	Return to zero.
10	Return to one.
11	Return to complement.

Tabelle 4-4 Signaltypen

Je nach Einstellung dieses Registers besitzen die 4 Ausgänge dieser Einheit unterschiedliches Verhalten. Diese unterschiedlichen Verhalten sind auch die Standard-Signaltypen, die von unterschiedlichen Tools zur Patternerzeugung unterstützt werden.

PGnCTRL							
7	6	5	4	3	2	1	0
ni				EN3	EN2	EN1	EN0

EN0 – EN3 - Enable Register der Generiereinheit

Mittels dieses Registers können die einzelnen 4 Kanäle der Generiereinheit separat geschaltet werden. Dieses Register ist sowohl über SFR Zugriff zugänglich kann aber auch durch die Hardware direkt beschrieben werden. Dies geschieht dann, wenn am Ausgang der Einheit HIGH-Z anliegen soll. In diesem Falle wird der gewünschte Ausgang deaktiviert.

PGnVEC							
7	6	5	4	3	2	1	0
GVEC3		GVEC2		GVEC1		GVEC0	

TYP0 – TYP3 - Einstellung des Pattern Typs für jede Generiereinheit

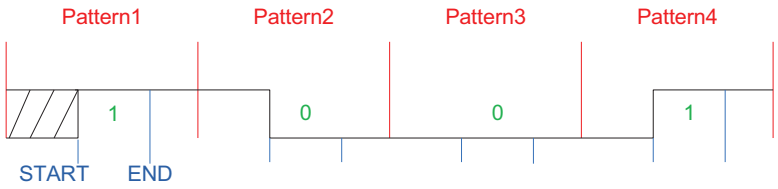
GVEC0 – GVEC3	**Typ:**
00	LOW
01	HIGH
10	HIGH Z
11	HIGH Z

Tabelle 4-5 Generierwerte

Mit diesem Register können die unterschiedlichen Werte angezeigt und festgelegt werden, welche in den aufgespannten Intervallen anliegen.

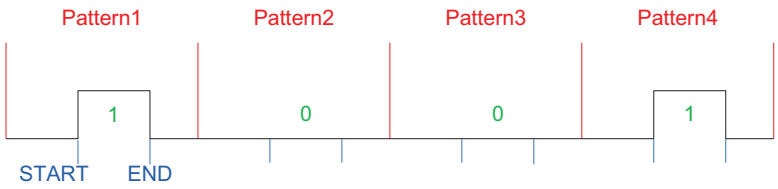
Non return to zero:

Bits gehen zum Zeitpunkt *PGnCMPF* 1 zu 1 in Ausgang über. Das bedeutet, dass jeder Zustand von *PGnVEC* direkt in die Ausgänge übergeht.



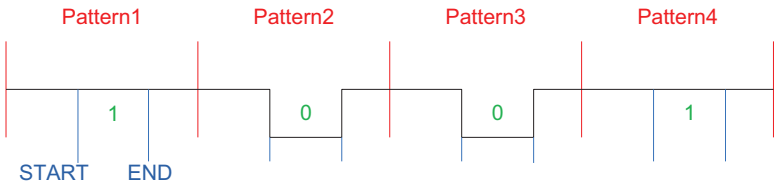
Return to zero:

Der Ausgang geht in dem mit Registern *PGnCMPF* und *PGnCMPS* aufgespannten Intervall auf den Wert von *PGnVEC* und dann wieder zurück auf LOW.



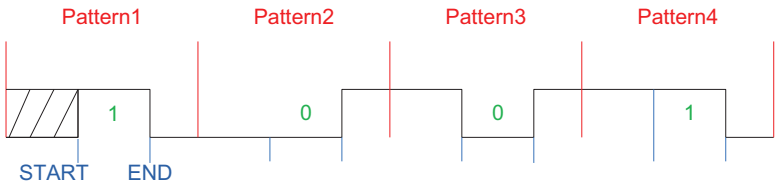
Return to one:

Der Ausgang geht in dem mit Registern *PGnCMPF* und *PGnCMPS* aufgespannten Intervall auf den Wert von *PGnVEC* und dann wieder zurück auf HIGH.



Return to complement:

Der Ausgang geht in dem mit Registern *PGnCMPF* und *PGnCMPS* aufgespannten Intervall auf den Wert von *PGnVEC* und dann auf dessen Komplement.



n…Anzahl der Generiereinheiten

PGnSTAT							
7	6	5	4	3	2	1	0
ni					RUN	PFULL	BFULL

BFULL - Pattern Buffer 1 Status Bit

PFULL - Pattern Buffer 2 Status Bit

RUN - Pattern Run Bit

PFULL und BFULL zeigen an, welche Buffer des Doppelbuffer Systems momentan voll beziehungsweise leer sind. Das RUN Bit zeigt an, ob die Generiereinheit die Ausgangssignale generiert.

4.3.1.3 Patterngenerator Compare Unit

Diese Einheit dient zum Vergleich der empfangenen Signale mit einem voreingestellten Wert. Die Einstellung des Wertes und der Vergleichszeiten wird wiederum über SFR Zugriffe abgehandelt. Somit können Intervalle eingestellt werden, in denen der tatsächlich empfangene Wert mit einem erwarteten verglichen wird. Für den Fall, dass die Werte nicht übereinstimmen bzw. der empfangene Wert sich innerhalb des Intervals ändert, wird ein Fehler detektiert. Im Fehlerfall wird der Vektorenzähler, die Generiervektoren, der erwartete Wert und der empfangene Wert im RAM abgelegt. Der Errorcounter wird inkrementiert.

Zusätzlich ist zu erwähnen, dass bis zu 10 Einheiten mit je 4 Ausgängen durch einfaches Ändern eines Feldes in der SFR Liste und der Pinliste automatisch erzeugbar sind.

Konfiguration der Einheit mittels Special Function Register:

Registername:	**Funktion:**
PCnCMPF0 - PCnCMPF3	Einstellung des Startpunktes des Intervalls
PCnCMPS0 - PCnCMPS3	Einstellung des Endpunktes des Intervalls
PCnCTRL	Register zum Freischalten der Eingänge
PCnVEC	Register mit Vergleichswerten
PCnGOT	Register mit empfangenen Werten
PCnSTAT	Status Register

Tabelle 4-6 Vergleichseinheit SFR Übersicht

Register:

PCnCMPF0 - PCnCMPF3							
7	6	5	4	3	2	1	0
CMPF							

CMPF - Startpunkt der ersten Flanke

Diese Register dienen zum Einstellen der ersten Flanke, die den Beginn des zu überprüfenden Intervalls darstellt. Die Auflösung der Pattern und damit auch die Anzahl der auswählbaren Möglichkeiten hängen von der Voreinstellung im Teiler ab. Je größer dieser ist, desto feiner können die Pattern eingestellt werden.

PCnCMPS0 - PCnCMPS3							
7	6	5	4	3	2	1	0
CMPS							

CMPS - Startpunkt der zweiten Flanke

Diese Register dienen zum Einstellen der zweiten Flanke, bis zu der die Eingänge überprüft werden. Danach werden die eingehenden Signale nicht mehr mit den zu erwarteten Werten verglichen. Die Auflösung der Pattern und damit auch die Anzahl der auswählbaren Möglichkeiten hängen von der Voreinstellung im Teiler ab. Je größer dieser ist, desto feiner können die Pattern eingestellt werden.

PCnCTRL							
7	6	5	4	3	2	1	0
ni				EN3	EN2	EN1	EN0

EN0 – EN3 - Enable Register der Compareeinheit

Mittels dieses Registers können die einzelnen 4 Kanäle der Compareeinheit einzeln konfiguriert (enabled/disabled) werden.

PCnVEC							
7	6	5	4	3	2	1	0
CEXP3		CEXP2		CEXP1		CEXP0	

CEXP0 – CEXP3 - Erwartungswerte der 4 Kanäle der Vergleichseinheit

CEXP0 – CEXP3	**Typ:**
00	LOW
01	HIGH
10	Don’t Care
11	Don’t Care

Tabelle 4-7 Erwartungswerte

Mittels dieses Registers können die Erwartungswerte für die 4 Eingangskanäle eingestellt werden. Diese Werte werden im aufgespannten Intervall mit den tatsächlich empfangenen Werten verglichen. Sollten die Werte voneinander abweichen, wird ein Fehler detektiert.

PCnGOT							
7	6	5	4	3	2	1	0
CGOT3		CGOT2		CGOT1		CGOT0	

CGOT0 – CGOT3- Eingangswerte der 4 Kanäle der Vergleichseinheit

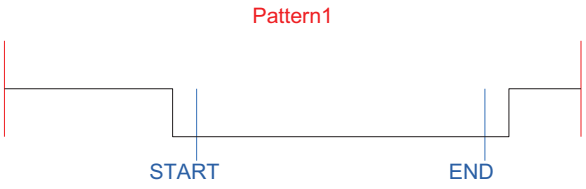
CGOT0 – CGOT3	**Samplewerte:**
00	LOW
01	HIGH
10	HIGH-Z
11	HIGH-Z

Tabelle 4-8 Empfangene Werte

In diesem Register sind die tatsächlichen Eingangswerte gespeichert. Diese werden bei jeder positiven Taktflanke gesampled. Diese Werte werden zur Fehlerdetektion mit denen des PCnVEC Registers verglichen

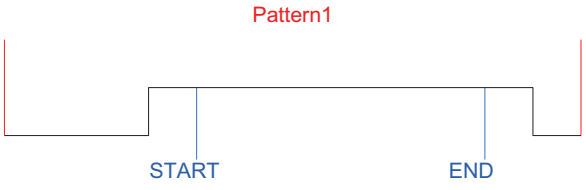
Expected Value ist “Zero”:

Der erwartete Wert ist LOW.

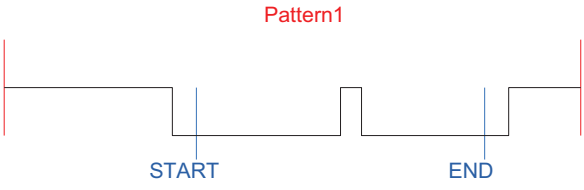


Expected Value = Got Value = OK

Expected Value ≠ Got Value = Error

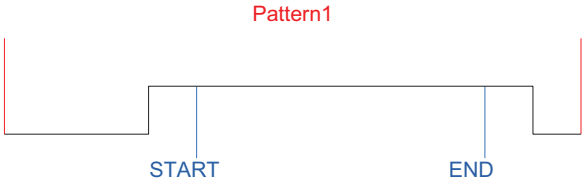


Got Value ≠ konstant = Error



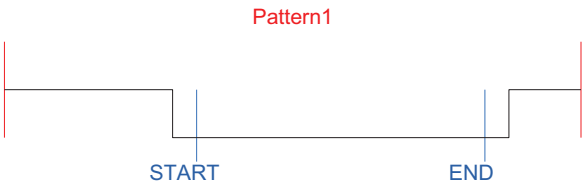
Expected Value ist “One”:

Der erwartete Wert ist HIGH.

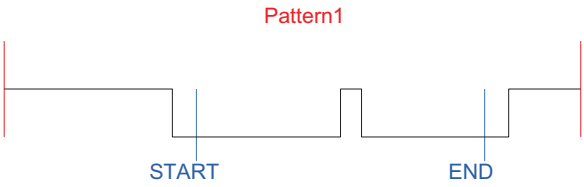


Expected Value = Got Value = OK

Expected Value ≠ Got Value = Error



Got Value ≠ konstant = Error



Expected Value ist “HIGH Z ”:

Kein Fehlerfall kann auftreten, da Eingangswert egal ist. Status immer OK.

PCnSTAT							
7	6	5	4	3	2	1	0
ni					RUN	PFULL	BFULL

PFULL - Pattern Compare Buffer 1 Status Bit

BFULL - Pattern Compare Buffer 2 Status Bit

RUN - Pattern Compare Run Bit

PFULL und BFULL zeigen an, welche Buffer des Doppelbuffer Systems momentan voll beziehungsweise leer sind. Das RUN Bit zeigt an, ob die Vergleichseinheit die Eingangssignale empfängt und mit den Vergleichswerten vergleicht.

4.3.1.4 Direct Memory Access Unit

Diese Einheit wurde durch die gewünschte Pattern Frequenz notwendig. Diese sollte bei 1 MHz liegen. Um die Generierwerte und Vergleichswerte in kürzerer Zeit vom Flash-Speicher zu den Einheiten zu bringen, wurde ein direkter Speicherzugriff nötig. Sollte der Mikrocontroller diese Aufgabe übernehmen, würden die Exekutionszeiten der Software zu viel Zeit benötigen.

Die DMA Einheit stellt, je nach Anzahl der Generier- und Vergleichseinheiten, die Daten vom Flash-Speicher zu Verfügung und speichert sie in den Einheiten ab. Im Falle der Generiereinheit wird das Register PGnVEC mit den Daten beschrieben, im Falle der Vergleichseinheit das PCnVEC Register beschrieben.

Hierbei musste jedoch auch eine Zeitverzögerung implementiert werden, da auch der Flash Speicher gewisse Latenzen aufweist. Dies und die Anzahl der Einheiten begrenzt die maximal mögliche Patternfrequenz.

Konfiguration der Einheit mittels Special Function Register:

Registername:	**Funktion:**
STARTVEC0 – STARTVEC2	Startvektor für Tests
COUNTER0 – COUNTER2	Endvektor für Tests
DMASTAT	Status Register
DMACTRL	Kontrollregister
PGUNITCNT	Anzahl der Generiereinheiten
PCUNITCNT	Anzahl der Vergleichseinheiten
DMASTATCLR	Status Clear Register

Tabelle 4-9 DMA SFR Übersicht

Register:

STARTVEC0 – STARTVEC2							
7	6	5	4	3	2	1	0
START							

START - Startwert der Pattern

In diesen Registern werden die Startnummern für den DMA Zugriff festgehalten. Mittels Spezial Function Register Zugriff kann der Startvektor für die Tests eingegeben werden. Der maximale Wert dabei ist durch die Anzahl von 24 Bit gegeben.

COUNTER0 – COUNTER2							
7	6	5	4	3	2	1	0
CNT							

CNT - Endwert der Pattern

In diesen Registern werden die Anzahl der Testpattern für den DMA Zugriff festgehalten. Mittels Spezial Function Register Zugriff kann der Endvektor für die Tests eingegeben werden. Der maximale Wert dabei ist durch die Anzahl von 24 Bit gegeben. Es ist zu beachten, dass, würde eine größere Zahl als die tatsächlich im Flash-Speicher vorhandene Anzahl der Pattern eingestellt werden, beliebige Daten vom Flash zu den Einheiten transportiert werden. Dies hätte eine Erzeugung und den Vergleich mit zufälligen Daten zu Folge.

DMASTAT							
7	6	5	4	3	2	1	0
ni						RDY	RUN

RUN - Run Bit

RDY - Ready bit

Das RUN Bit zeigt an, ob die DMA Einheit arbeitet. Das RDY Bit zeigt an, ob die DMA Einheit konfiguriert und arbeitsfähig ist.

DMACTRL							
7	6	5	4	3	2	1	0
ni						LOOP	START

START - Start Bit

LOOP - Loop Bit

Durch Setzen des START Bits wird der Direkt Memory Access, je nach Wert des RDY Bits im DMASTAT Register, gestartet. Das LOOP Bit gibt die Möglichkeit, die Testvektoren im Kreis laufen zu lassen. Dabei wird das Intervall zwischen STARTVEC und COUNTER wiederholt durchlaufen. Dies gibt die Möglichkeit, Pattern zu wiederholen und längere Tests laufen zu lassen.

PGUNITCNT							
7	6	5	4	3	2	1	0
UNITNR							

UNITNR - Anzahl der Generiereinheiten

Mittels dieses Registers kann die DMA Einheit bezüglich der Anzahl der Generiereinheiten konfiguriert werden. Dies ist notwendig, um die Länge des DMA Zugriffes einzustellen.

PCUNITCNT							
7	6	5	4	3	2	1	0
UNITNR							

UNITNR - Anzahl der Generiereinheiten

Mittels dieses Registers kann die DMA Einheit bezüglich der Anzahl der Vergleichseinheiten konfiguriert werden. Dies ist notwendig, um die Länge des DMA Zugriffes einzustellen.

DMASTATCLR							
7	6	5	4	3	2	1	0
ni						RDYCLR	RUNCLR

RUNCLR - Run Bit

RDYCLR - Ready bit

Dieses Register wird verwendet, um die Statusbits des DMASTAT Registers zurückzusetzen.

4.3.1.5 UART Unit

Diese Einheit wurde benötigt, um eine größere Anzahl an Baudraten, schneller als die Standard Baudraten des 8051 Controllers, für die serielle Schnittstelle einzustellen. Sie ermöglicht sowohl eine Erhöhung der Baudrate durch einen kleineren Vorteiler, als auch eine Verringerung durch Erhöhen des Vorteilers. Dazu wurde der bereits vorhandene Controller verändert. Diese Einheit besteht nur aus einem Register und ist simpel konfigurierbar. Die Taktfrequenz des entwickelten Systems liegt bei 30 MHz.

Die UART Einheit des verwendeten DP8051 besteht aus folgenden Registern.

SBAUD0							
7	6	5	4	3	2	1	0
EN	ni			BAUD			

EN - Enable Bit

BAUD - UART Baudrate Register

Baudrateberechnung:

$$TH1 = 256 - \frac{Quarzfrequenz}{BAUD} * \frac{2 * SMOD}{Baudrate * 32}$$

SBUF0							
7	6	5	4	3	2	1	0
SBUF0							

SBUF0 - UART Datenbuffer

Dieses Register enthält die jeweils empfangenen oder zu sendenden Daten. Je nachdem ob ein Schreibbefehl oder ein Lesebefehl auf dieses Register erfolgt, zeigt es unterschiedliche Daten. Bei einem Schreibbefehl darauf werden die geschriebenen Daten in das Senderegister transferiert und übertragen. Wird ein Lesebefehl auf das SBUF0 Register ausgeführt, werden die Daten des UART Empfangsregisters gelesen.

SCON0							
7	6	5	4	3	2	1	0
SM00	SM01	SM02	REN0	TB08	RB08	TI0	RI0

SM00 - Baudrateneinstellung

SM01 - Baudrateneinstellung

SM02 - Enable für Multiprozessor Kommunikation

REN0 - Enable für seriellen Empfang

TB08 - Neuntes zu sendendes Datenbit

RB08 - Neuntes empfangenes Datenbit

TI0 - Transmit Interrupt Bit

Ri0 - Receive Interrupt Bit

Dieses Register dient zum Einstellen der Baudrate und des Modus, der Schnittstelle. Dieses Register hält auch das Enable Bit der UART Schnittstelle. Es hält auch die neunten Bits für Senden und Empfangen im 9 Bit Modus. Auch die Interrupt Flags, welche empfangene Daten oder gesendete Daten signalisieren, sind in diesem Register lokalisiert.

Folgende Tabelle zeigt die möglichen Einstellungen der UART Schnittstelle.

SM00	SM01	Modus	Beschreibung	Baudrate
0	0	0	Shieberegister	fclk/12
0	1	1	8-bit UART	Variabel
1	0	2	9-bit UART	fclk/32 oder fclk/64
1	1	3	9-bit UART	variabel

Tabelle 4-10 UART Modus

Die Baudratenberechnung ist, je nach Modus, abhängig von Systemtakt, Vorteiler und dem TH1 Register, welches für die Überlaufsfrequenz (T1ov) des Timers 1 verantwortlich zeichnet.

Modus	Baudrate	
0	fclk/12	
1,3	SMOD0 = 0	T1ov/32
	SMOD1 = 1	T1ov/16
2	SMOD0 = 0	fclk/64
	SMOD0 = 1	fclk/32

Tabelle 4-11 UART Baudrate

Das SMOD0 Bit befindet sich im Register PCON an Stelle der zwei hochwertigsten Bits.

4.3.1.6 Portexpander Wrapper

Der Port Expander Wrapper wurde zur Ansteuerung des Port Expanders implementiert, welcher zur Erhöhung der Anzahl der IOs führt. Es gab bereits eine vorhandene Ansteuerungseinheit für den Port Expander, jedoch war dieser auf ein ganz anderes System ausgelegt. Dazu wurde diese Einheit erstellt, die die Ansteuerungseinheit dem System anpasst.

Konfiguration der Einheit mittels Special Function Register:

Registername:	**Funktion:**
PE_CTRL0	Portexpander Kontrollregister
PE_CTRL1	Portexpander Kontrollregister
PE_STAT	Status Register
PE_DOUT0 – PE_DOUT3	Daten Ausgangsregister
PE_DIN0 – PE_DIN3	Daten Eingangsregister
PE_DIR0 – PE_DIR3	Konfigurationsregister
PE_MOD0 – PE_MOD3	Modusregister

Tabelle 4-12 Portexpander SFR Übersicht

Register:

PE_CTRL0							
7	6	5	4	3	2	1	0
ni							PE_S

PE_S - Port Expander Shutdown

Dieses Kontrollregister enthält ausschließlich das Port Expander Shutdown Bit, welches die Einheit zum Abschalten bringt.

PE_CTRL							
7	6	5	4	3	2	1	0
PE_RUN	PE_DATEN	PE_PCFGEN	PE_GCFGEN	PE_PREDIV			

PE_PREDIV - Wert des Predividers

PE_GCFGEN - Hinausschieben aus Portexpander Globalmodus

PE_PCFGEN - Hinausschieben aus Portexpander

PE_DATEN - Hinausschieben der Daten Enable

PE_RUN - Hinausschieben Enable

Dieses Register ermöglicht unterschiedliche Einstellungen, welche das Verhalten des Portexpanders bestimmen.

PE_STAT							
7	6	5	4	3	2	1	0
ni							PE_RUN

PE_RUN - Status Bit

Dieses Status Register zeigt an, ob der Port Expander läuft oder nicht.

PE_DOUT0 – PE_DOUT3							
7	6	5	4	3	2	1	0
PE_DOUT0 – PE_DOUT3							

PE_DOUT - Ausgangsdaten

Diese Register halten die Daten, die an den Ausgängen des Portexpanders hinausgeschoben werden. Das Register PE_DOUT0 besteht nur aus 4 Bit, da der Portexpander über 28 Kanäle verfügt.

PE_DIN0 – PE_DIN3							
7	6	5	4	3	2	1	0
PE_DIN0 – PE_DIN3							

PE_DIN - Eingangsdaten

Diese Register halten die Daten, die an den Eingängen des Portexpanders anliegen. Das Register PE_DIN0 besteht nur aus 4 Bit, da der Portexpander über 28 Kanäle verfügt.

PE_DIR0 – PE_DIR3							
7	6	5	4	3	2	1	0
PE_DIR0 – PE_DIR3							

PE_DIR - Richtungskonfiguration

Diese Register legen fest, ob die Kanäle des Portexpanders als Eingang oder als Ausgang betrieben werden. Das Register PE_DIR0 besteht nur aus 4 Bit, da der Portexpander über 28 Kanäle verfügt.

PE_MOD0 – PE_MOD3							
7	6	5	4	3	2	1	0
PE_MOD0 – PE_MOD3							

PE_MOD - Moduskonfiguration

Diese Register dienen zur Einstellung des Modus, in dem die Kanäle arbeiten. Das Register PE_MOD0 besteht nur aus 4 Bit, da der Portexpander über 28 Kanäle verfügt.

4.3.1.7 Memory Multiplexer Unit

Diese Einheit dient dazu, alle Zugriffe auf Speicherbereiche zu verwalten. Dabei wird unterschieden, ob ein Zugriff des Mikrocontrollers auf den Programmspeicher erfolgt, oder ob dieser auf seinen internen Speicher zugreift, oder auf seinen externen. Zusätzlich gibt es auch die Situation, dass ein Fehlerfall auftritt. Dabei wird der Mikrocontroller, durch Setzen seines READY Bits, in einen WAIT Zustand versetzt und ein direkter Speicherzugriff durchgeführt. Dies ist notwendig, da im Fehlerfall sofort der Wert des Vektorzählers, die Werte der generierten Pattern, die erwarteten Werte und die tatsächlich erhaltenen Werte abgespeichert werden müssen. Der direkte Speicherzugriff hat Priorität.

In dieser Einheit war die automatische Code Erzeugung von extremer Wichtigkeit, da immer nur so viel Speicher erzeugt werden muss, wie vom FPGA verwendet wird. Dadurch musste sich auch die Adressierung mit den gegebenen Konfigurationen mitändern.

4.3.1.8 Memory Mapping

Im Fehlerfall werden durch einen direkten Speicherzugriff der Vector Counter, der Inhalt des PGnVEC Registers als auch der des PCnVEC Registers und des PCnGOT Registers im RAM abgelegt. Der DP8051 muss jedoch auch auf die gespeicherten Werte zugreifen können. Dies wurde durch einen XRAM Zugriff des Mikrocontrollers realisiert. Dabei werden die Speicherbereiche der abgelegten Daten als linearer XRAM Bereich des DP8051 angesehen.

XRAM Adresse	Speicherbereich Fehlerfall
0x0010FF	**EXP9 [127]**
.	**GOT9 [127]**
.	**.**
.	**EXP9 [0]**
0x001000	**GOT9 [0]**
.	**.**
0x0008FF	**EXP1 [127]**
.	**GOT1 [127]**
.	**.**
.	**EXP1 [0]**
0x000800	**GOT1 [0]**
0x0007FF	**EXP0 [127]**
.	**GOT0 [127]**
.	**.**
.	**EXP0 [0]**
0x000700	**GOT0 [0]**

DMA Zugriff durch
PATGEN_CTRL
PATGEN_GEN
PATGEN_CMP

XRAM Adresse	Speicherbereich Fehlerfall
0x0006FF	**GEN9 [127]**
.	**.**
0x000680	**GEN9 [0]**
.	**.**
0x0002FF	**GEN1 [127]**
.	**.**
0x000280	**GEN1 [0]**
0x00027F	**GEN0 [127]**
.	**.**
0x000200	**GEN0 [0]**
0x0001FF	**ni**
.	**VEC2 [127]**
.	**VEC1 [127]**
.	**VEC0 [127]**
.	**.**
.	**ni**
.	**VEC2 [1]**
.	**VEC1 [1]**
.	**VEC0 [1]**
.	**ni**
.	**VEC2 [0]**
.	**VEC1 [0]**
0x000000	**VEC0 [0]**

Tabelle 4-13 Memory Mapping

5 Software

Die Software wurde so implementiert, dass diese ausschließlich auf dem entwickelten System läuft und keine zusätzliche Software auf Seiten des PCs notwendig ist. Die Steuerung des Systems läuft über einfache Textbefehle, die in einem Terminal am PC eingegeben werden. Die Software interpretiert mittels eines Parsers die empfangenen Befehle und setzt diese um.

5.1 Übersicht der Applikationen

Dateien:	Funktionen:	Aufgabe:
patgen.c	main	Hauptschleife
patgen_global.h	main header	Header der Hauptschleife
app_patgen_uart_driver.c	app_patgen_uart_init app_patgen_uart_interrupt app_patgen_uart_rcbuffer_handling app_patgen_uart_rcbuffer2_full app_patgen_uart_rcbuffer2_clr app patgen_uart_rcbuffer_get app_patgen_uart_buffer_overrun app_patgen_uart_tmbuffer_handling app_patgen_uart_transmit app_patgen_uart_send_buffer app_patgen_uart_transmit_string app_patgen_uart_echo app_patgen_uart_header app_patgen_uart_version	UART Konfiguration und Funktionen zum Senden und Erhalten von Daten
app_patgen_uart_driver.h	UART header	Header der UART Funktionen
app_patgen_parser.c	app_patgen_parser app_patgen_parser_format app_patgen_parser_command app_patgen_parser_getatt app_patgen_parser_compares app_patgen_parser_dividers app_patgen_parser_pbuffer2_clr app_patgen_parser_getsfr app_patgen_file_timeout app_patgen_file_timeout_stop app_patgen_timer0_interrupt	Parser zur Erkennung von Kommandos und zur Handhabung von Konfigurationsfiles.
app_patgen_parser.h	parser header	Header der Parser Funktion
app_patgen_hd_flash.c	app_read_byte_from_flash app_write_byte_to_flash app_sector_erase_flash app_complete_erase_flash app_header_erase_flash app_pattern_erase_flash app_check_flash app_check_head_flash	Kontrollfunktion für den Flash Speicher. Diese steuert sämtliche Zugriffe auf den Speicher des Flash.

	app_check_pattern_flash	
app_patgen_hd_flash.h	Flash handling header	Header der Flash Kontrollfunktion
app_patgen_file_flash.c	app_patgen_file_tm app_patgen_flash_init app_patgen_vec_flash app_patgen_head_flash app_patgen_head_sfr app_patgen_head_crc app_patgen_patt_crc app_patgen_head_cfg	Kontrolleinheit zum Ablegen der Daten des Konfigurationsfiles im Flash
app_patgen_file_flash.h	File to Flash header	Header der File to Flash Funktion
app_patgen_mem.c	app_patgen_xram_write app_patgen_xram_read app_patgen_iram_write app_patgen_iram_read app_patgen_pmem_read app_patgen_sfr	Funktionen zum Speicherzugriff inklusive Zugriff auf Special Function Register
app_patgen_mem.h	Memory header	Header der Speicher Kontrollfunktion
app_patgen_output.c	app_aptgen_help app_patgen_error app_patgen_print	Funktionen zum Ausgeben der Hilfe und von Zahlenwerten
app_patgen_output.h	Output header	Header der Ausgabefunktionen
app_patgen_usr_cmd.c	app_patgen_usr_cmd	Anwenderspezifische Kommandos
	app_turnau_scan_mode app_turnau_test_mode app_turnau_spi_write_reg app_turnau_send_byte app_turnau_end	Diese Funktionen werden später nicht genauer beschrieben, da es sich dabei um produktspezifische Details handelt.
app_patgen_usr_cmd.h	User command header	Header für anwenderspezifische Kommandos

Tabelle 5-1 Software Übersicht

5.2 Verhaltensbeschreibungen der Funktionen

Die Software wurde so entwickelt, dass seitens des PC's nur ein einfaches Text Terminal ausreicht, welches über die USB Schnittstelle mit dem System kommuniziert. Durch Eingabe von vordefinierten Kommandos kann das Verhalten des Pattern Generators definiert werden.

Zusätzlich wurde damit ermöglicht, den Pattern Generator mittels der einfachen Übermittlung einer WGL-Datei zu konfigurieren. Diese wird textbasiert, Zeile für Zeile, übermittelt.

Dies bietet den Vorteil, keine Benutzerschnittstelle am PC implementieren zu müssen. Deshalb wurde jedoch eine umfangreiche Software entwickelt, welche auf dem DP8051 Mikrokontroller läuft.

5.2.1 Main Loop

Dieser Teil der Software beinhaltet zwei einfache Funktionen zur Initialisierung der Einheiten. Hierbei wird die UART Schnittstelle konfiguriert und das Echo aktiviert, welches die übermittelten Zeichen direkt an das Textterminal am PC zurücksendet.

Die Hauptschleife beinhaltet auch eine Endlosschleife, in der zwei Funktionen für die Kommunikation über die serielle Schnittstelle und die Interpretation der erhaltenen Kommandos zuständig sind.

5.2.2 UART Treiber (app_patgen_uart_driver.c)

In dieser Datei sind alle, für die UART Kommunikation notwendigen, Funktionen beinhaltet.

5.2.2.1 UART Initialisierung (app_patgen_uart_init)

Diese Funktion wird anfangs in der Hauptschleife aufgerufen und dient der Initialisierung der UART Einheit und der Baudrateneinstellung dieser. Zusätzlich werden durch das Setzen der dafür vorgesehenen Bits in den Registern IE (Interrupt Enable) und IP (Interrupt Priority) der Interrupt der seriellen Schnittstelle eingeschaltet und dessen Priorität definiert.

Es wird auch der Timer selektiert, der die Baudrate der UART Schnittstelle vorgibt, in diesem Falle Timer 1. Durch Setzen der Register TH1 und SMOD kann die Baudrate angepasst werden. Die Kommunikation des Pattern Generators läuft mit einer Baudrate von 19.2 kBaud. Diese relativ geringe Geschwindigkeit entsteht durch die Zugriffszeiten auf den Flash Speicher. Wenn der Pattern Generator über die WGL-Datei konfiguriert wird, kann diese

nicht schneller übertragen werden, als dass die Attribute im Speicher abgelegt werden können.

Der Wert des TH1 Registers berechnet sich wie folgt:

$$TH1 = 256 - \frac{Quarzfrequenz}{BAUD} * \frac{2 * SMOD}{Baudrate * 32}$$

Wenn der Echo Modus der Kommunikation eingeschaltet ist, wird mit der Initialisierung auch ein Header übertragen, welcher genauere Informationen über Software und Hardware Version anzeigt.

5.2.2.2 Interrupt Service Routine (app_patgen_uart_interrupt)

Wird ein Byte erfolgreich übertragen, wird ein Interrupt ausgelöst. Durch einen einfachen Lesezugriff auf das SBUF Register kann der empfangene Wert ausgelesen werden. Über Pointer Operationen werden die erhaltenen Werte in einen Buffer geschrieben. Wenn der Echo Modus eingeschaltet ist, wird der Wert auch direkt in das SBUF Register zurückgeschrieben, um ihn wieder zurück zu senden und am Textterminal auszugeben.

Bei einem Zeilenumbruch oder beim Erreichen der maximalen Zeilenlänge wird der Zeilenzähler inkrementiert.

Auch das Senden von Daten löst einen Interrupt aus, welcher zeigt, dass ein Byte erfolgreich transferiert worden ist.

5.2.2.3 Receive Buffer Verarbeitung (app_patgen_uart_rcbuffer_handling)

Diese Funktion wird in einer Endlosschleife ausgeführt. Sie dient dazu, eine empfangene Zeile in einen zweiten Buffer zu kopieren. Wenn dieser voll ist, wird er von einer weiteren Anwendung, dem Parser, verarbeitet. Gleichzeitig wird der Zeilenzähler des ersten Buffers dekrementiert.

5.2.2.4 Receive Buffer 2 Full (app_patgen_uart_rcbuffer2_full)

Diese Funktion gibt zurück, ob der zweite Empfangsbuffer voll ist oder nicht.

5.2.2.5 Receive Buffer 2 Clear (app_patgen_uart_rcbuffer2_clr)

Diese Funktion setzt das Buffer zwei voll Flag zurück.

5.2.2.6 Buffer Lokalisierung (app_patgen_uart_rcbuffer_get)

Diese Funktion gibt die Position des Pointers auf den Empfangsbuffer zurück.

5.2.2.7 Buffer Overrun (app_patgen_uart_buffer_overrun)

Diese Funktion dient dazu, einen Fehler anzuzeigen und ihn an das Text Terminal auszugeben. Der auftretende Fehler ist dabei ein Schreibbefehl auf einen bereits vollen Buffer.

5.2.2.8 Transmission Buffer Verarbeitung (app_patgen_uart_tmbuffer_handling)

Ist der Buffer zum Senden der Daten leer, kann dieser beschrieben werden. Wird ein Zeilenumbruch in den Buffer geschrieben oder die maximale Buffergröße erreicht, so wird der Inhalt des Buffers übertragen. Dies wird durch die Funktion app_patgen_uart_send_buffer ausgeführt.

Sollte der Buffer voll sein, kann nicht auf ihn geschrieben werden und es wird gewartet, bis die Daten übermittelt sind.

5.2.2.9 Zeichen Transmission (app_patgen_uart_transmit)

Diese Funktion dient dazu, einzelne Zeichen in den Sendebuffer zu schreiben. Sie ruft auch die vorher beschriebene Funktion der Buffer Verarbeitung auf.

5.2.2.10 Sende Transmission Buffer (app_patgen_uart_send_buffer)

Hierbei wird der aktuelle Wert im Sendebuffer in das Register SBUF geschrieben, was das Senden über die UART Schnittstelle auslöst.

5.2.2.11 Transmit String (app_patgen_uart_transmit_string)

Diese Funktion erhält einen String als Übergabeparameter, splittet diesen auf und sendet die einzelnen Zeichen mit der app_patgen_uart_transmit Funktion. Es wird an den zu übertragenden String noch ein Zeichen mit dem ASCII Wert für Return angehängt, um den String abzusenden.

5.2.2.12 UART Echo (app_patgen_uart_echo)

Je nach Übergabeparameter wird der Echo Modus ein- beziehungsweise ausgeschaltet.

5.2.2.13 Transmit Header (app_aptgen_uart_header)

Diese Funktion sendet den Header, der Informationen über das Projekt beinhaltet, an das Textterminal am PC.

5.2.2.14 Transmit Version (app_patgen_uart_version)

Diese Funktion sendet Informationen bezüglich Software- und Hardwareversion zum Textterminal am PC.

5.2.3 Datei und Kommando Parser

Der Parser ist dafür verantwortlich, aus den von der seriellen Schnittstelle empfangenen Daten die richtigen Schlüsse zu ziehen. Je nachdem welche Daten als Zeilen durch die UART Schnittstelle empfangen werden, muss der Parser unterscheiden, ob es sich um Speicherzugriffskommandos handelt, ob Kontrollkommandos eingegeben werden oder zum Beispiel Zugriffe auf Special Function Register erfolgen.

Er beinhaltet auch unterschiedliche Funktionen, um aus den als String empfangenen Daten Attribute abzuleiten. Auch zur Gewährleistung einer korrekten Dateiübertragung sind einige Kontrollfunktionen vorhanden.

5.2.3.1 Parser (app_patgen_parser)

Ist der Empfangsbuffer voll, wird ein Pointer auf den Buffer gesetzt. Daraufhin wird die Formatierungsfunktion aufgerufen. Ist diese abgeschlossen, wird die Kommandofunktion aufgerufen, welche die Interpretation der empfangenen Zeile übernimmt.

5.2.3.2 Formatierung (app_patgen_parser_format)

Um die empfangenen Zeilen vor dem Verarbeiten zu formatieren, wurde diese Funktion implementiert. Sie wandelt den gesamten String in Kleinbuchstaben um und ersetzt, sollten zwei oder mehrere Leerzeichen aufeinander folgen, diese durch ein einzelnes Leerzeichen. Sollte eine Zeile mit Leerzeichen enden, werden diese abgeschnitten.

Zeilenumbrüche werden durch Null ersetzt.

5.2.3.3 Kommando Funktion (app_patgen_parser_command

Der Parser kann zwischen folgenden Kommandos unterscheiden. Sollte ein falsches Kommando oder eine Syntaxverletzung vorliegen, wird diese am Textterminal am PC angezeigt.

Kommandoliste					
Kommando	**Adresse**	**Daten**	**Daten**	**Aktion**	**Wert**
help	o	o		Hilfe anzeigen	
iram_write	add	dat		Byte in IRAM schreiben	hex,bin,dec
iram_read	add	o		Byte aus IRAM lesen	hex,bin,dec
xram_write	add	dat		Byte in XRAM schreiben	hex,bin,dec
xram_read	add	o		Byte aus XRAM lesen	hex,bin,dec
sfr_write	add	dat		Byte in SFR schreiben	hex,bin,dec
sfr_read	add	o		Byte aus SFR lesen	hex,bin,dec
pmem_read	add	o		Byte aus PMEM lesen	hex,bin,dec
file_tm				Datei senden	
file_tm_hd				Datei senden, nur Header	
file_tm_pt				Datei senden, nur Pattern	
pat_head_ld				Header in SFR laden	
head_cfg				Zeige Headerkonfiguration	
pat_run		startvec	endvec	Starte Pattern	hex,bin,dec
pat_run_lp		startvec	endvec	Starte Pattern in Loop Modus	hex,bin,dec
pat_stop				Stoppe Pattern	
head_crc				Überprüfe Header CRC	
pat_crc				Überprüfe Pattern CRC	
flash_er_hd				Lösche Header aus Flash	
flash_er_pt				Lösche Pattern aus Flash	
flash_er				Lösche gesamten Flash	
flash_chk				Überprüfe ob Flash leer ist	

Tabelle 5-2 Kommandoliste

Die Daten und Adressen für die unterschiedlichen Kommandos können in Hexadezimal-format, Binär-format oder einfach dezimal eingegeben werden. Beim hexadezimalen Format ist den Daten ein 0x vorzusetzen, beim binären Format 0b und im Dezimalformat ist die gewünschte Zahl einfach einzugeben.

Wie auch beim Kommando Parser werden im Dateitransfermodus die empfangenen Zeilen auf deren Zugehörigkeit untersucht. Der Parser gliedert die zur Konfiguration gesendete WGL-Datei in unterschiedliche Teile, um die Verarbeitung zu optimieren.

Empfangene Zeile	Aktion wenn detektiert
waveform	Startet Timeout für Dateitransfer
signal	Startet Reihung der Signale
pattern	Startet Ablegen der Pattern in Flash
end	Beendet Datentransfer, stoppt Timeout

Tabelle 5-3 WGL-Dateistruktur

Wird die Startzeile mit dem Attribut „waveform“ detektiert, schaltet sich ein Timer ein, welcher dazu dient, die Korrektheit des Dateitransfers zu überwachen. Sollte aus irgendwelchen Gründen der Transfer abbrechen, läuft der Timer über und zeigt die Störung an, worauf der Dateitransfer endet.

Wird eine Zeile mit dem Attribut „signal“ empfangen, so weiß der Parser, dass die folgenden Zeilen Signalnamen und deren Richtung beinhalten. So wird die Anzahl der Ein- und Ausgänge des Pattern Generators detektiert. Dieser Block wird mit einer Zeile, in der „end“ steht, geschlossen.

Detektiert der Parser eine Zeile, die mit „timeplate“ beginnt, so wird dieser Zeile die Frequenz der Pattern entnommen. Je nach gefundenem Wert (z.B.: 1000ns) wird nun der Wert des Predividers und des Dividers ermittelt und vorrübergehend im RAM abgelegt. Die weiters folgenden Zeilen geben Aufschluss auf das Timing und die Typen der unterschiedlichen Signale. Ihnen werden die Zeiten der Vergleichswerte und das Verhalten der Signale entnommen und im RAM abgelegt. Auch dieser Block wird mit einer „end“ Zeile beendet.

Der Pattern Block wird mit dem detektierten Attribut „pattern“ gestartet. Alle folgenden Zeilen sind Testvektoren, die entschlüsselt und im Flash-Speicher abgelegt werden. Durch diesen Vorgang wird die maximale Baudrate der Übertragung festgelegt, da die Dauer des Entschlüsselns und die Ablage in den Flash-Speicher eine gewisse Zeit beansprucht. Dieser Block wird auch mit einer „end“ Zeile beendet.

Der gesamte Dateitransfer wird mit einer „end“ Zeile beendet.

Das Ende der Datei startet, außer es wurde der Modus für ausschließliches Übertragen von Pattern gewählt, die Speicherung des Headers im Flash-Speicher.

5.2.3.4 Attribut Erkennung (app_patgen_parser_getatt)

Hierbei wird ein Startpunkt des Strings übergeben, um ab dort nach Zahlenwerten zu suchen. Diese können in Hexadezimalformat, Binärformat und Dezimalformat vorliegen. Der detektierte Wert wird als „unsigned long integer“ Rückgabewert übergeben. Das heißt, es sind Werte von bis zu 4 Bytes möglich. Zusätzlich wird auch die Zeichenlänge der Zahl abgelegt. Dies ist wichtig, um zwei Zahlen hintereinander zu erfassen.

5.2.3.5 Einstellung der Divider (app_patgen_parser_dividers)

Als Parameter werden dieser Funktion die Zeitkonstante der Pattern und ein Skalierungswert übergeben. Dieser gibt an, ob es sich bei der Zeitkonstante um Nanosekunden oder Mikrosekunden handelt. Diesen beiden Werten zufolge wird der Predivider und der Divider gesetzt. Dabei wird darauf geachtet, die maximal mögliche Skalierung der Pattern zu erreichen, um möglichst viele Abstimmungsmöglichkeiten zu erhalten.

5.2.3.6 Vergleichswerte Erkennung (app_patgen_parser_compares)

Dieser Funktion werden zwei Werte übergeben, der Vergleichswert und ein Skalierungswert. Aus diesen beiden Werten und der Periodendauer der Pattern können die Start- oder Endwerte der Signale berechnet werden, die das Vergleichs- oder Datenintervall angeben.

5.2.3.7 Parser Buffer löschen (app_patgen_parser_pbuffer2_clr)

Diese Funktion gibt den Parser Buffer zum erneuten Überschreiben frei und setzt den Timer zurück, um dessen Überlauf zu verhindern. Passiert dies nicht innerhalb einer gewissen Zeit, löst der Timer einen Interrupt aus.

5.2.3.8 Special Function Register Erkennung (app_patgen_parser_getsfr)

Diese Funktion dient dazu, aus dem Registernamen das gewünschte Special Function Register zu detektieren und dessen Adresse zurückzugeben. Dabei wird zuerst die Länge des Registernamens ermittelt, um aufgrund dieser eine Vorunterteilung vorzunehmen. Dadurch kann die Zeit für das Finden der richtigen Adresse reduziert werden. Erst danach wird der String mit den Namen der Register verglichen. Diese Funktion wird mittels der automatischen Code Generierung automatisch erweitert, sollten neue Special Function Register zur Registerliste und dem System hinzugefügt werden.

5.2.3.9 Timeout Start (app_patgen_file_timeout)

In dieser Funktion wird der Timer zur Kontrolle der Richtigkeit des Dateitransfers eingeschaltet. Dazu werden die Interrupts des Timers 0 ermöglicht und ihnen Prioritäten zugewiesen.

5.2.3.10 Timeout Stop (app_patgen_timeout_stop)

Diese Funktion stoppt den Timer.

5.2.3.11 Timer Interrupt Service Routine (app_patgen_timer0_interrupt)

In dieser Routine wird der Timer auch abgeschaltet, der Dateitransfer abgebrochen und eine Fehlermeldung ausgegeben.

5.2.4 Flash-Speicher Zugriffe

Hierbei handelt es sich um Schreib- und Lesezugriffe auf den Flash-Speicher. Auch für das Löschen des Speichers wurden einige Funktionen implementiert. Der Flash Speicher wird als 1 Wort breiter Speicher benutzt. Die Zugriffe auf den Flash-Speicher wurden daran angepasst. Es sind XRAM Zugriffe des DP8051 Mikrokontrollers.

5.2.4.1 Byte lesen (app_read_byte_from_flash)

Dieser Funktion wird eine Adresse als long integer übergeben. Die Adresse wird aufgespaltet und in drei Bytes in den internen RAM Speicher geschrieben. Der weitere XRAM Zugriff, welcher vom Flash Speicher liest, wurde in Assembler geschrieben, da nur so auf den großen Speicherbereich zugegriffen werden konnte. Zurückgegeben wird ein unsigned char mit dem Wert des gelesenen Bytes.

5.2.4.2 Byte schreiben (app_write_byte_to_flash)

Als Übergabeparameter erhält diese Funktion zusätzlich noch ein Byte Daten als unsigned char zusätzlich zur Adresse, die im gleichen Format vorliegt wie beim Lesezugriff. Auch der Schreibzugriff wurde in Assembler implementiert.

5.2.4.3 Sektor löschen (app_sector_erase_flash)

Diese Funktion löscht einen Sektor im Flash-Speicher. Der Sektor wird je nach Wert der übergebenen Variable ausgewählt. Diese Funktion wird durch gewisse Reihenfolgen von Schreibzugriffen auf den Speicher ausgelöst. Auch diese Funktion wurde in Assembler implementiert.

5.2.4.4 Lösche Flash (app_complete_erase_flash)

Mittels einer Schleife werden durch die Funktion „Sektor löschen“ alle Sektoren, die zum Ablegen des Headers und der Pattern verwendet werden, gelöscht.

5.2.4.5 Lösche Header (app_header_erase_flash)

Damit wird der Header, welcher im Flash-Speicher abgelegt ist, gelöscht. Dies entspricht einem Löschen des Sektors 1.

5.2.4.6 Lösche Pattern (app_pattern_erase_flash)

Damit werden die Pattern aus dem Flash Speicher gelöscht. Dies entspricht dem Löschen der Sektoren 2-64.

5.2.4.7 Überprüfe Flash (app_check_flash)

Ob der Speicher beschrieben oder leer ist, wird mittels dieser Funktion ermittelt.

5.2.4.8 Überprüfe Header (app_check_header_flash)

Hierbei wird überprüft, ob der Header im Flash abgespeichert ist oder nicht.

5.2.4.9 Überprüfe Pattern (app_check_pattern_flash)

Diese Funktion ermittelt, ob Pattern im Flash abgelegt sind oder nicht.

5.2.5 Datei in Flash

Hierbei werden alle Funktionen zusammengefasst, die dazu dienen, Konfigurationen und Pattern im Flash abzulegen und die Daten im Flash Speicher zu überprüfen.

5.2.5.1 Datei Transfer (app_patgen_file_tm)

Mit dieser Funktion werden Voreinstellungen getroffen, um eine WGL-Datei zu übertragen. Es wird auch angezeigt, dass das System bereit ist und das Zeichenecho wird abgeschaltet.

5.2.5.2 Vektor speichern (app_patgen_vec_flash)

Diese Funktion wird verwendet, um Testvektoren im Flash-Speicher abzulegen. Die erste Speicherstelle der Testvektoren hat die Adresse 0x820000. Mit jedem abgespeicherten Vektor wird dieser auch zum CRC Wert hinzugezählt.

5.2.5.3 Header speichern (app_patgen_head_flash)

Während des Dateitransfers werden die Konfigurationsdaten im RAM abgelegt. Mit Ende des Daten Transfers wird diese Funktion aufgerufen, welche die Daten vom RAM in den Flash-Speicher ablegt. Die Konfiguration im Header wird wie folgt abgelegt.

Flash Speicher		
FORMAT		**INFO**
0x800000	PRDIV	Predivider
0x800001	DIV	Divider
Generiersignale		
	en	Enable
	type	Typ
	cmpf	Startwert
	cmps	Stopwert
Vergleichssignale		
	en	Enable
	cmpf	Startwert
	cmps	Stopwert
Je nach Anzahl der Einheiten gibt es unterschiedlich viele Generiersignale und Vergleichssignale		
0x81FFFA		Anzahl Pattern (LSB)
0x81FFFB		Anzahl Pattern
0x81FFFC		Anzahl Pattern (MSB)
0x81FFFE		CRC (MSB)
0x81FFFF		CRC (LSB)

Tabelle 5-4 Flash-Speicher Header

5.2.5.4 Header in SFR (app_patgen_head_sfr)

Wird das Kommando „head_ld“ am Text Terminal des PCs eingegeben, wird diese Funktion aufgerufen. Damit werden die Headerdaten aus dem Flash-Speicher in die Special Function Register geladen, um die Einheiten zu konfigurieren.

5.2.5.5 Header CRC Kontrolle (app_patgen_head_crc)

Diese Funktion vergleicht den CRC Wert des Headers mit den im Flash-Speicher gespeicherten Daten des Headers. Ist die Überprüfung korrekt, wird deren Richtigkeit angezeigt, wenn nicht eine Fehlermeldung ausgegeben wird.

5.2.5.6 Pattern CRC Kontrolle (app_patgen_patt_crc)

Diese Funktion vergleicht den CRC Wert der Pattern mit den im Flash gespeicherten Daten des Headers. Ist die Überprüfung korrekt, wird deren Richtigkeit angezeigt, wenn nicht eine Fehlermeldung ausgegeben wird.

5.2.5.7 Header Konfiguration (app_patgen_header_cfg)

Durch das Textkommando „head_cfg“ wird diese Funktion aufgerufen. Sie gibt die Daten des Headers am Text Terminal aus. Die Werte des Predividers, des Teilers, die Anzahl der Signale, die Anzahl der Einheiten, die Reihenfolge der Signale und deren Konfigurationen werden angezeigt.

5.2.6 Speicherzugriffe, SFR Zugriffe

Um auf den externen RAM Speicher, den internen RAM Speicher, den Programmspeicher und die Special Function Register zugreifen zu können, mussten einige Funktionen dafür implementiert werden.

5.2.6.1 Schreibe in XRAM (app_patgen_xram_write)

Diese Funktion holt sich die Daten und die Adresse aus dem empfangenen String und speichert mittels Assembler Code die Daten im Speicher an der gewünschten Adresse des externen RAM Speichers ab.

5.2.6.2 Lese aus XRAM (app_patgen_xram_read)

Diese Funktion holt sich die Adresse aus dem empfangenen String und liest mittels Assembler Code die Daten aus dem externen RAM Speicher.

5.2.6.3 Schreibe in IRAM (app_patgen_iram_write)

Diese Funktion holt sich die Daten und die Adresse aus dem empfangenen String und speichert mittels Assembler Code die Daten im Speicher an der gewünschten Adresse des internen RAM Speichers ab.

5.2.6.4 Lese aus IRAM (app_patgen_iram_read)

Diese Funktion holt sich die Adresse aus dem empfangenen String und liest mittels Assembler Code die Daten aus dem internen RAM Speicher.

5.2.6.5 Lese aus PMEM (app_patgen_pmem_read)

Diese Funktion holt sich die Adresse aus dem empfangenen String und liest mittels Assembler Code die Daten aus dem Programmspeicher.

5.2.6.6 SFR Zugriff (app_patgen_sfr)

Dieser Funktion werden die Adresse des zu behandelnden Special Function Registers, die Art des Zugriffs, Schreiben oder Lesen, und die zu schreibenden Daten übergeben. Nun wird aus der Adresse und der Pagenummer eine neue Variable erzeugt, welche mittels eines Switch Befehls abgearbeitet wird.

5.2.7 Ausgabe

Um Fehler, die Hilfe und um Zahlen in verschiedenen Formaten anzuzeigen, wurden drei Funktionen implementiert.

5.2.7.1 Hilfe (app_patgen_help)

Diese Funktion wird durch den Textbefehl „help“ aufgerufen und zeigt die Auswahlmöglichkeiten der Hilfe an. Je nach Auswahl werden die einzelnen Kommandos aufgelistet und beschrieben. Die Erklärungen werden durch den Anfangsbuchstaben der Untergruppen selektiert, indem man diese über das Text Terminal eingibt und mit Enter bestätigt.

5.2.7.2 Error (app_patgen_error)

Je nach übergebenem Parameter werden unterschiedliche Fehlermeldungen ausgegeben. Dies erleichtert das Lokalisieren des Fehlers und zeigt die Art des Fehlers an.

5.2.7.3 Zahlenausgabe (app_patgen_print)

Diese Funktion gibt die ihr übergebene Zahl als Dezimalzahl, Hexadezimalzahl und Binärwert aus.

5.2.8 Benutzerdefinierte Kommandos

Diese Rubrik wurde eingeführt, um der bestehenden Software Funktionen hinzuzufügen ohne mehrere Dateien ändern zu müssen. Es steht dem Anwender frei, Funktionen dort zu implementieren und diese auch über das Text Terminal aufzurufen.

5.2.8.1 Benutzerkommando (app_patgen_usr_cmd)

Dies ist die übergeordnete Funktion, welche je nach übergebenem Wert, Kommandos aufruft. Der Befehl, der diese Funktion aufruft, ist „usr_cmd $“. Der $ Wert kann ein beliebiger Zahlenwert sein, welcher die Auswahl tätigt.

Bei diesen Kommandos handelt es sich um spezifische Anwendungen, wie zum Beispiel ein gewünschtes DUT in den Scan Modus zu versetzen.

6 Testboard

6.1 Grundlagen

Um alle gewünschten Chips der Firma Dialog Semiconductor testen zu können, mussten verschiedene Vorkehrungen auf der Platine getroffen werden. Es mussten verschiedene Spannungsdomains der Logiksignale ermöglicht werden. Um alle Pins des DUTs mit Signalen versorgen oder vergleichen zu können, mussten sämtliche Ein- und Ausgänge des Pattern Generators mit jedem gewünschten Pin verbunden werden können.

Auch mussten Spannungsversorgungen für 12 V, 5 V und 3.3 V zur Verfügung gestellt werden.

Um auch analoge Signale generieren und vergleichen zu können, sollte ein Analog-Digital-Umsetzer und ein Digital-Analog-Umsetzer im System vorhanden sein.

6.2 Konzept

Das Konzept des Testboards wird in folgender Grafik illustriert.

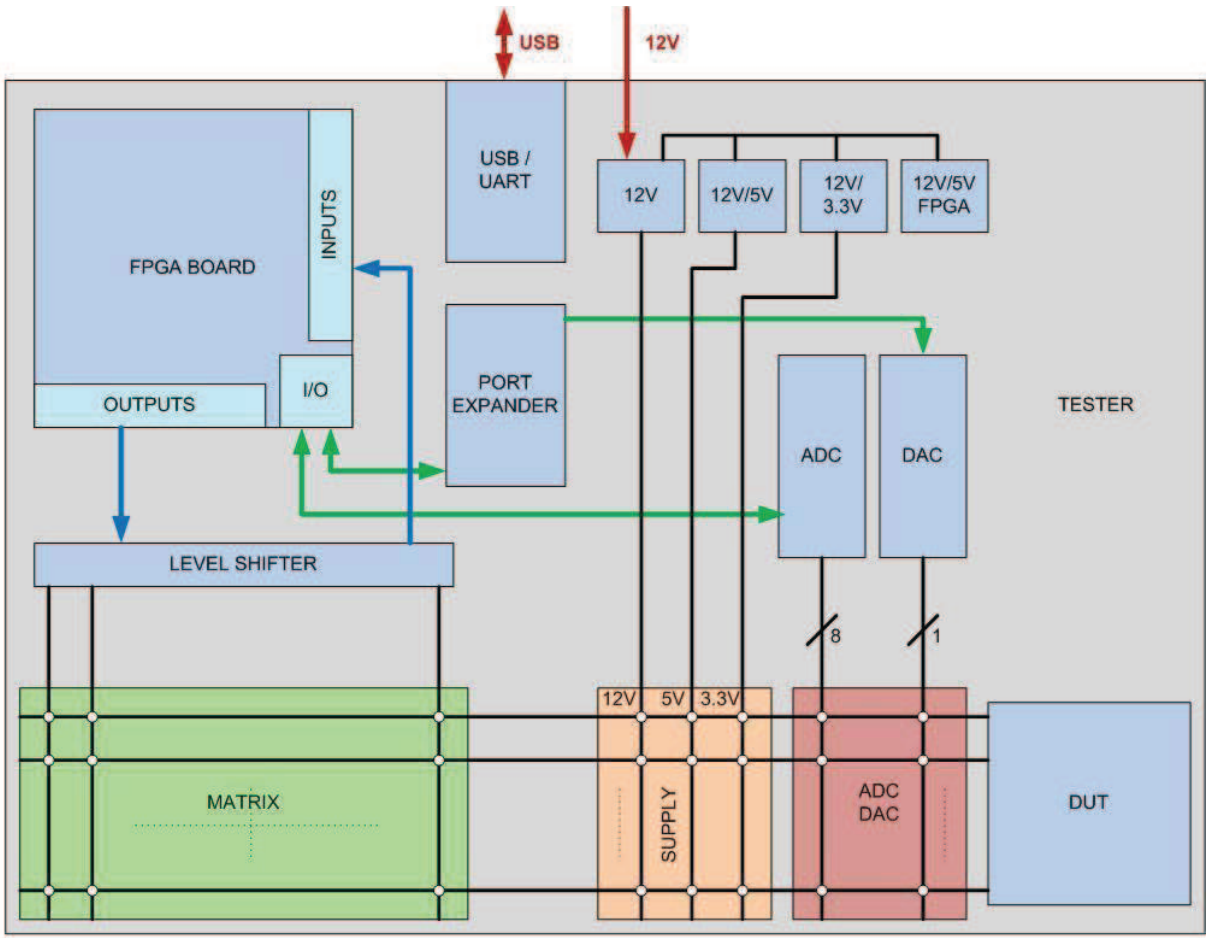


Abbildung 6-1 Testboard Konzept

6.3 Boarddesign

Das Testboard wurde mit einer 6 Layer Platine aufgebaut. Beim Design des Boards wurde darauf geachtet, es so aufzubauen um eine möglichst hohe Flexibilität zu gewährleisten. Diese soll es ermöglichen, verschiedenste Chips mittels dieses Systems zu testen.

Anhand der Board Oberseite werden die Platzierungen der unterschiedlichen Einheiten illustriert.

6.3.1 Board Oberseite

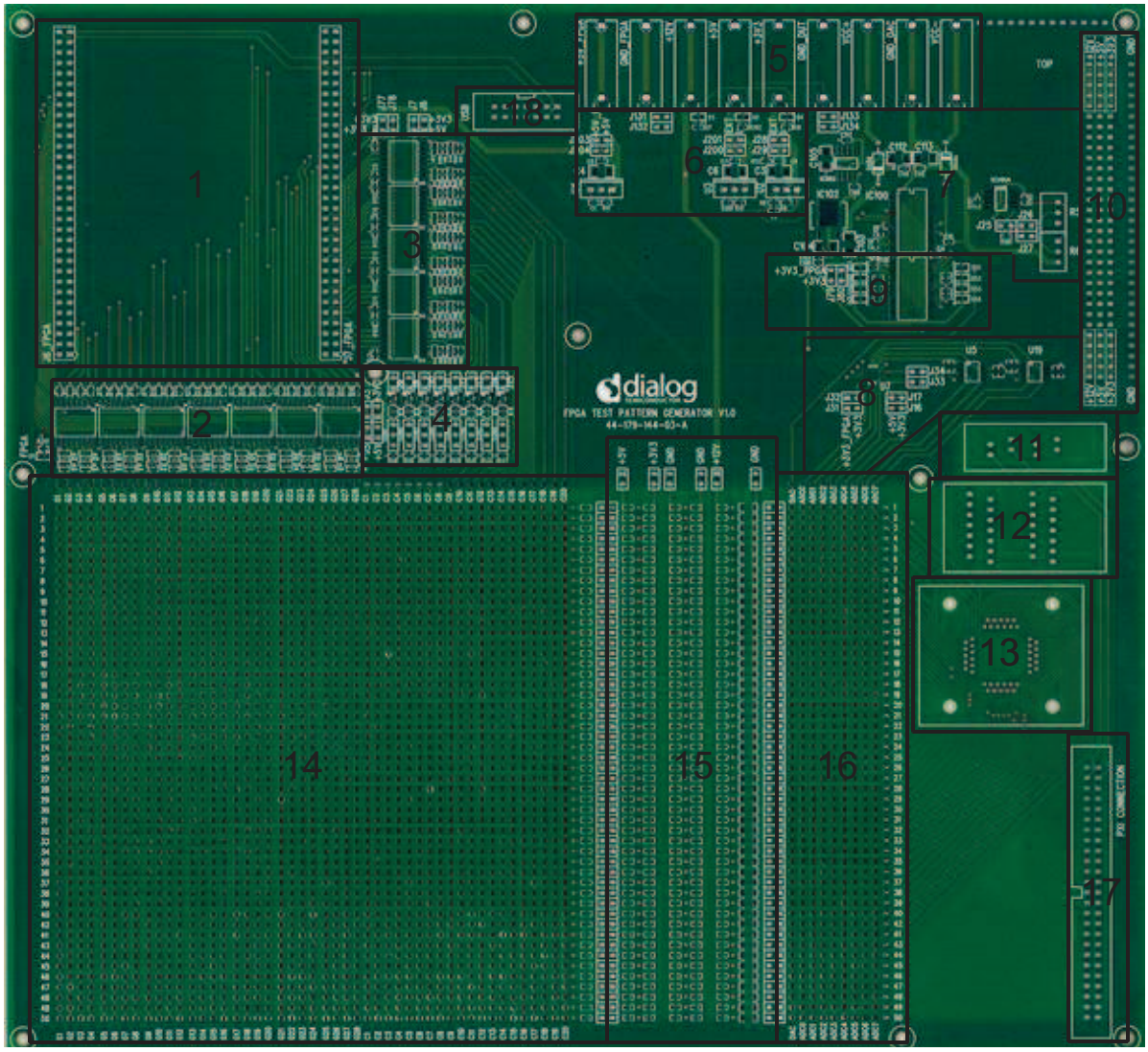


Abbildung 6-2 Testboard Oberseite

6.4 Übersicht der Komponenten

Nr.	Dateien:	Funktionen:	Aufgabe:
1	**FPGA Verbindung**	Stecker	Stiftleisten zum Aufstecken des FPGA Boards
2	**Level Shifter**	Pegelschieber	Umsetzung der Logiksignale auf gewünschte Pegel, Digitale Ausgänge
3	**Level Shifter**	Pegelschieber	Umsetzung der Logiksignale auf gewünschte Pegel, Digitale Eingänge
4	**Level Shifter**	Pegelschieber	Umsetzung der Logiksignale auf gewünschte Pegel, Digitale Eingänge
5	**Buchsen**	Supply	Hirschmann Verbindungsbuchsen
6	**DC/DC Umsetzer 12V/5V**	Supply	DC / DC Umsetzer von 12V auf 5V
	DC/DC Umsetzer 12V/5V FPGA	Supply	DC / DC Umsetzer von 12V auf 5V, um FPGA Board zu versorgen
	DC/DC Umsetzer 12V/3.3V	Supply	DC / DC Umsetzer von 12V auf 3.3V
7	**ADC**	Analoge Signale	Einheit zum Messen analoger Signale
8	**DAC**	Analoge Signale	Einheit zum Erzeugen analoger Signale
9	**Lochrasterfeld**	Aufbaufeld	Bietet Möglichkeit, zusätzlich Schaltungen aufzubauen
10	**Port Expander**	Erweiterung	Vergrößert die Anzahl der IOs des FPGA Boards
11	**SOIC Sockel**	DUT Sockel	8 Pin SOIC 150mil Sockel
12	**SOIC Sockel**	DUT Sockel	28 Pin SOIC 300mil Sockel
13	**QFN Sockel**	DUT Sockel	44 Pin QFN Sockel
14	**Matrix**	Verbindung	Variable Verbindungseinheit
15	**Supply**	Verbindung	Versorgungsverbindungen
16	**ADC / DAC**	Verbindung	ADC / DAC Verbindungen
17	**PXI Konnektor**	Stecker	Steckverbindung für PXI System
18	**USB / UART Konverter**	Schnittstelle	Einheit zum Umsetzen von USB Signalen auf UART Signale

Tabelle 6-1 Testboard Komponenten

6.4.1 FPGA Verbindung

Hierbei handelt es sich um zwei doppelte 64 Pin Stiftleisten. Auf diese kann das verwendete FPGA Board aufgesteckt werden.

6.4.2 Level Shifter

Diese Level Shifter machen es möglich, digitale Signale auf unterschiedlichen Spannungsniveaus zu erzeugen. Die wichtigsten Eigenschaften der Level Shifter sind in folgender Tabelle angeführt.

Eigenschaften
Bidirektionale Kommunikation
3V – 5V Umsetzung
Output Enable Funktion
2ns maximale Pulsweitenstörung

Tabelle 6-2 Level Shifter Eigenschaften

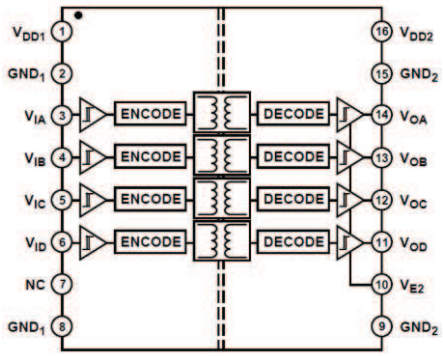


Abbildung 6-3 Funktionelles Blockdiagramm

Das Blockdiagramm in obiger Abbildung zeigt die Schmitt Trigger, die galvanische Trennung und die ausgangsseitig schaltbaren Schmitt Trigger.

Die Level Shifter, die als digitale Ausgänge verwendet werden, werden auf der Seite des zu testenden Chips mittels Zenerdioden geschützt (siehe Abbildung 6-4). Es wurden auch Serienwiderstände eingefügt. Die Eingänge werden hochohmig auf Ground gelegt.

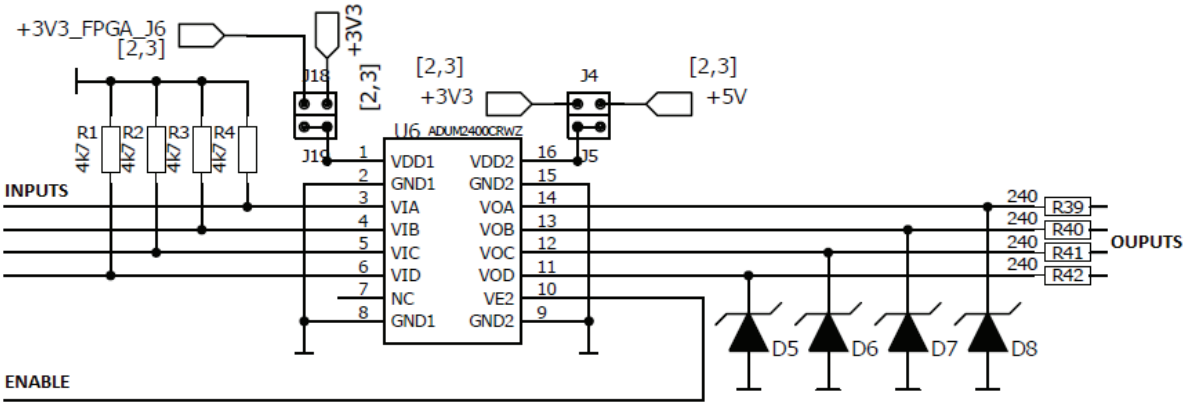


Abbildung 6-4 Level Shifter digitale Ausgänge

Die Level Shifter, die für die digitalen Eingänge verantwortlich zeichnen, werden auch mittels Zenerdioden seitens der zu testenden Einheit geschützt. Jedoch werden diese auch zusätzlich mit pull down Widerständen auf Ground gezogen (siehe Abbildung 6-5). Die Enable Signale der Level Shifter werden konstant auf Ground gelegt, um die Einheiten dauerhaft einzuschalten.

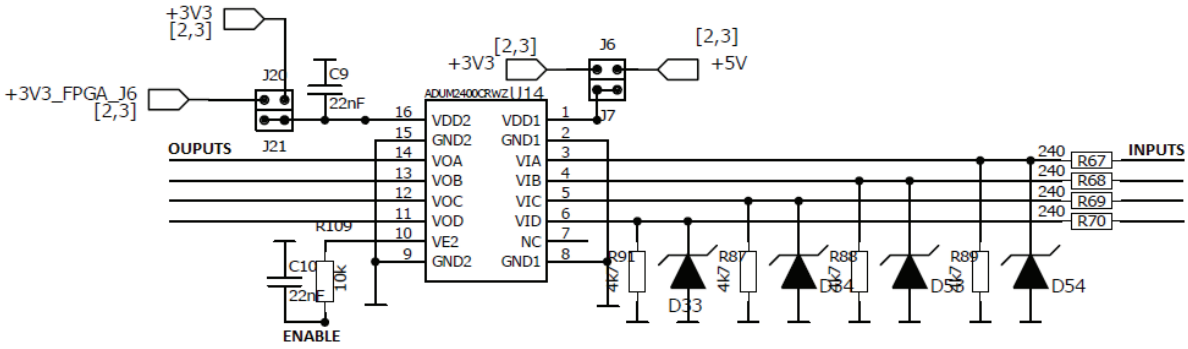


Abbildung 6-5 Level Shifter digitale Eingänge

6.4.3 Level Shifter

Um auch kleinere Spannungen als Eingang testen zu können, werden diese mittels der 8 Transistoren auf ein höheres Spannungsniveau von 3.3 V gebracht (siehe Abbildung 6-6).

Die Widerstände dienen zur Konfiguration der Transistorschaltung.

Mittels der Jumper kann der Kanal aber auch als einfacher Eingang konfiguriert werden.

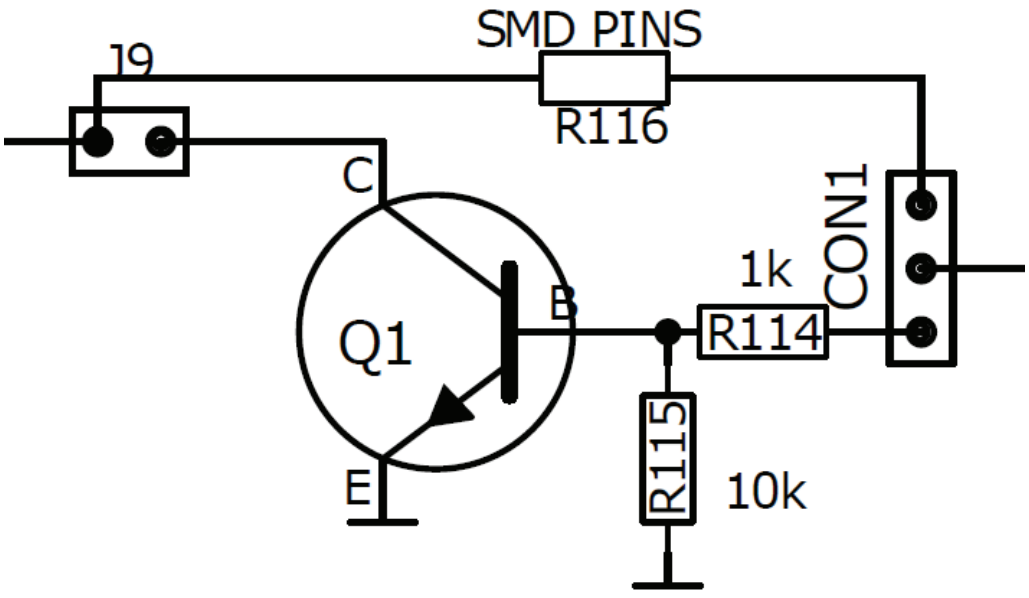


Abbildung 6-6 Bipolartransistor-Beschaltung

6.4.4 Buchsen

Um das gesamte Board zu versorgen, stehen unterschiedliche Buchsen zur Verfügung. Es ist möglich, alle Spannungen, 12 V, 5 V, 3.3 V, FPGA einzeln zu versorgen. Dabei sind auch die Massen des FPGA Boards und des zu testenden Chips trennbar ausgeführt. Jedoch ist es auch möglich, das gesamte Board mit 12 V zu versorgen. Dafür sind die Massen zu verbinden.

6.4.5 DC / DC Umsetzer

Mittels der DC DC Umsetzer werden alle Komponenten des Testboards mit den notwendigen Spannungen versorgt. Mit Jumpern kann man die Versorgungen konfigurieren (siehe Abbildung 6-7).

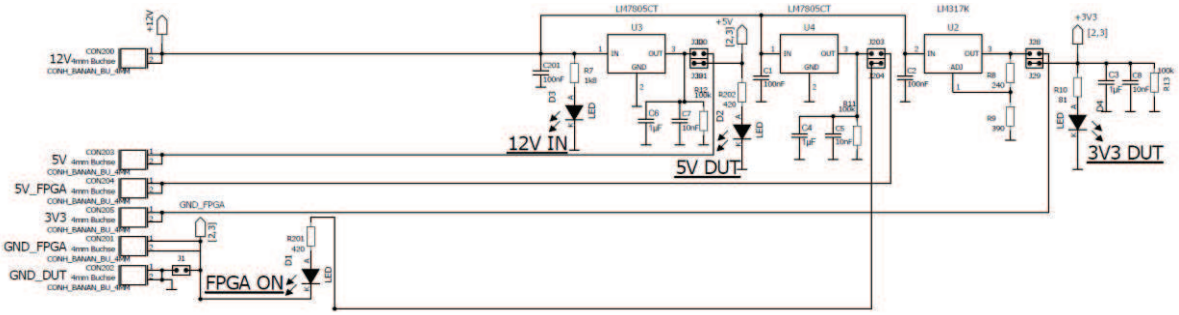


Abbildung 6-7 Spannungsversorgungen

6.4.6 Analog Digital Umsetzer

Hierbei handelt es sich um einen 8 Kanal ADC mit 14 Bit Auflösung. Die wichtigsten Eigenschaften sind in folgender Tabelle zusammengefasst. 4 Kanäle des ADCs werden über nicht invertierende Verstärker angesteuert, welche über die Widerstandswerte einstellbar sind. Die anderen 4 Eingänge des ADCs werden direkt angesprochen. Der ADC ist ein Sukzessiver Approximations Analog Digital Umsetzer.

Eigenschaften
14 Bit Auflösung ohne missing code
8 Kanal Multiplexer
Durchsatz 250 kSPS
INL / DNL: ±0.5 / ±0.25 LSB typisch
Eingangsbereich: 0V – Vdd
Multiple Spannungsreferenzen: Intern, Extern gebuffert (bis 4.096V), Extern (bis Vdd)
Interner Temperatursensor
Serielle Schnittstelle kompatibel mit SPI, MICROWIRE, QSPI und DSP
Geringes Übersprechen: -125dB

Tabelle 6-3 ADC Eigenschaften

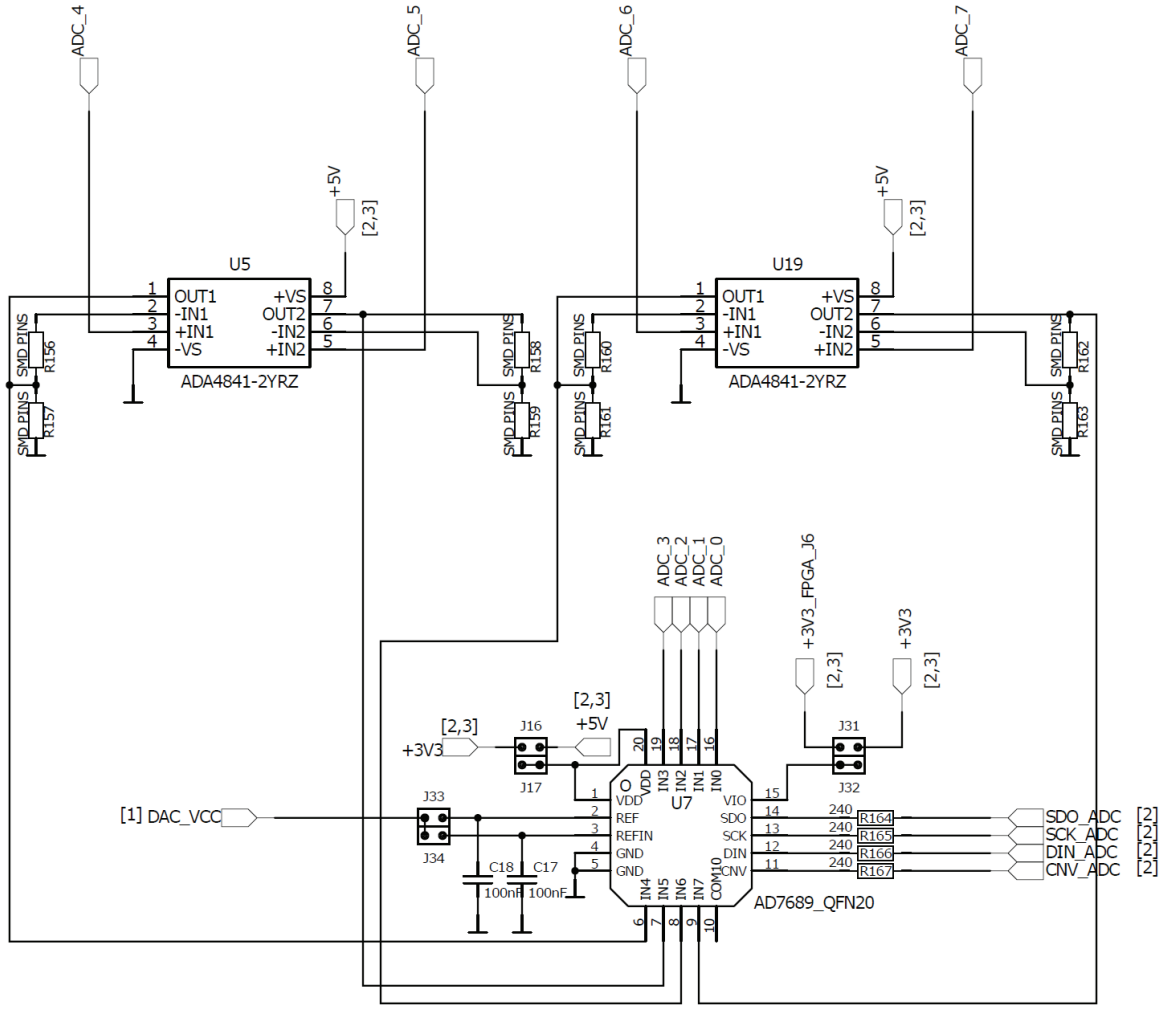


Abbildung 6-8 ADC Beschaltung

6.4.7 Digital Analog Umsetzer

Im Falle dieser Arbeit wurde ein LTC1821 DAC verwendet. Es handelt sich dabei um einen 16 Bit DAC. Die wichtigsten Eigenschaften werden in folgender Tabelle angeführt. Der DAC muss mit einer Versorgungsspannung und einer sinusförmigen Referenzspannung beaufschlagt werden. Ein Spannungsabgleich kann mittels zwei Potentiometern durchgeführt werden. Diese sind mittels Jumpern selektierbar. Die 16 Daten Eingänge werden vom Port Expander bereitgestellt (siehe Abbildungen 6-8 und 6-9).

Eigenschaften
1LSB maximale INL und DNL im industriellen Temperaturbereich
Geringes Rauschen: 13nV/√Hz
Geringer Glitch: 2nV*s
Power-On Reset
Asynchroner Clear Pin

Tabelle 6-4 DAC Eigenschaften

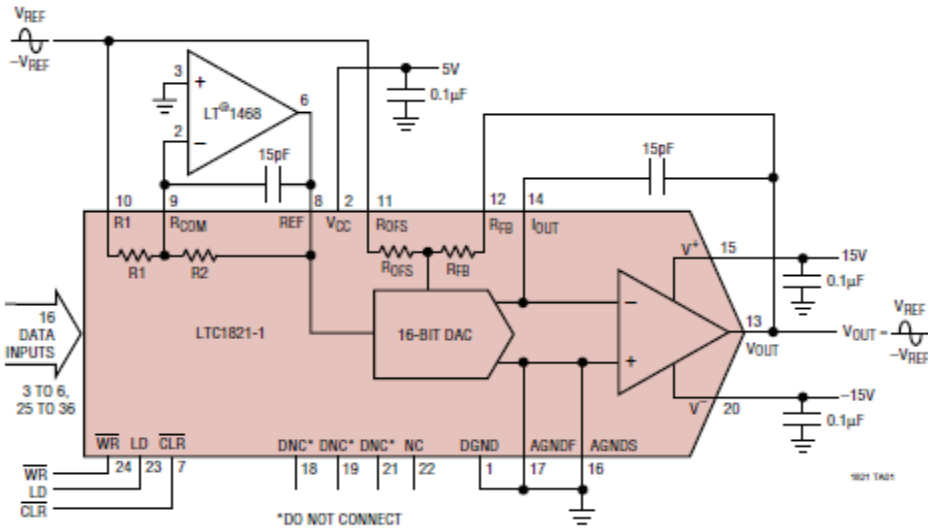


Abbildung 6-9 DAC Minimalbeschaltung (Linear-Technology)

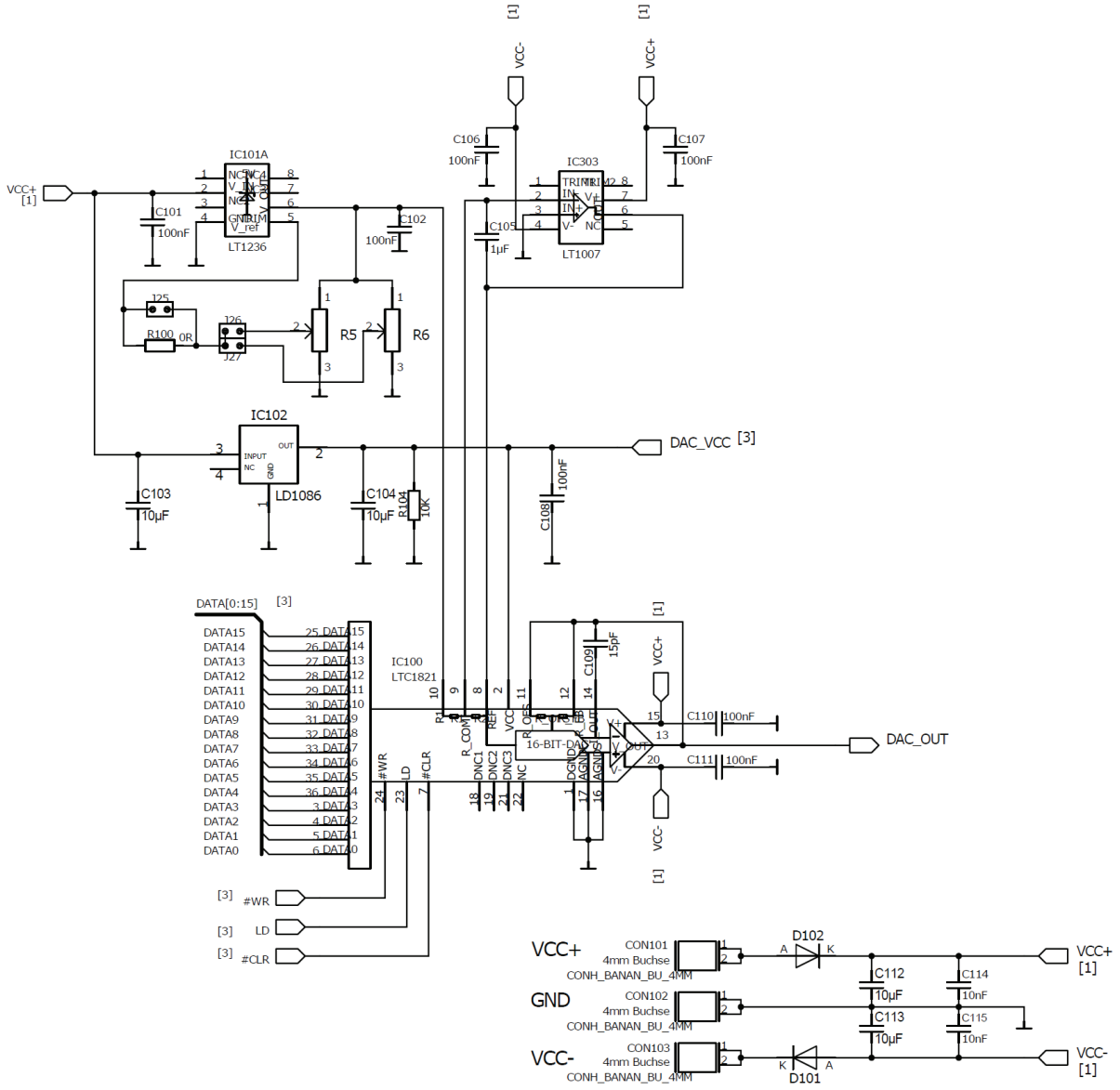


Abbildung 6-10 DAC Beschaltung und Versorgung

6.4.8 Lochrasterfeld

Um zusätzlich benötigte Schaltungen auf dem Testboard aufbauen zu können, wurde ein Lochrasterfeld bereitgestellt. Es besteht aus unbelegten Punkten, Versorgungsspannungspins und Massepins.

6.4.9 Port Expander

Um die Anzahl der IOs des FPGA Boards zu erweitern, wurde ein Port Expander auf dem Testboard untergebracht. Dieser wird mittels SPI Kommandos angesprochen. Die dafür notwendige Einheit wurde am FPGA Board implementiert. Der Port Expander verfügt über 28 IOs, welche getrennt als Eingang, Ausgang oder LED Treiber konfiguriert werden können. 8 Ausgänge des Port Expanders sind mit LEDs verbunden,19 weitere IOs des Port Expander sind mit dem DAC verbunden (siehe Abbildung 6-10).

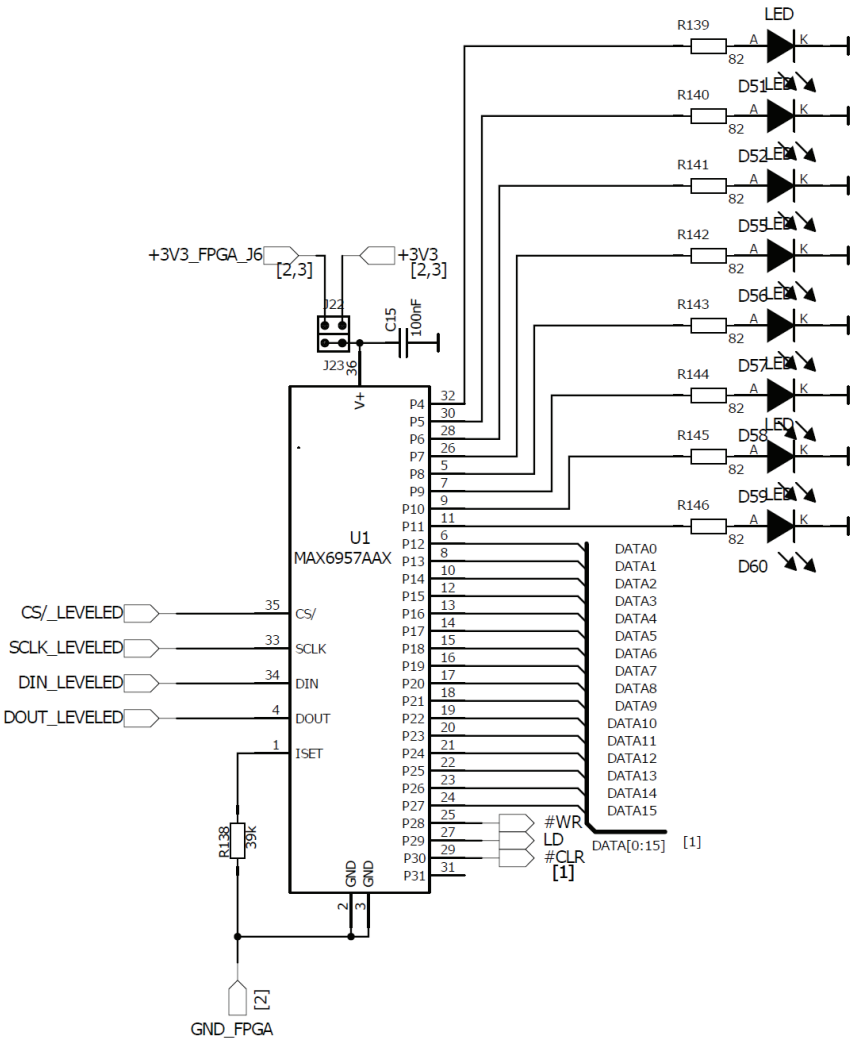


Abbildung 6-11 Port Expander Beschaltung

6.4.10 Sockel

Es wurde ein SOIC 8 Pin 150 mil Sockel, ein SOIC 28 Pin 300 mil Sockel und ein 44 Pin QFN Sockel auf dem Testboard platziert, um 4 Produkte von Dialog Semiconductor testen zu können.

6.4.11 Matrix Verbindungen

Die Verbindungsmatrix ermöglicht es, jeden Ein- oder Ausgang des FPGA Boards mit jedem Ein- oder Ausgang des zu testenden Chips zu verbinden. Dies geschieht durch einfache Durchkontaktierung mittels beidseitig verlöteten Metallpins.

6.4.12 Supply Verbindungen

Ähnlich wie im Falle der Kontaktiermatrix ist es auch hierbei möglich, jeden Pin des getesteten Chips mit einer der Spannungen zu beaufschlagen. Diese können mittels SMD Bauteilen mit den gewünschten Pins verbunden werden.

6.4.13 ADC / DAC Verbindungen

Jeder Pin des getesteten Chips kann auch mit dem Ausgang des DACs verbunden oder mit den 8 Eingängen des ADCs verbunden werden.

6.4.14 PXI Konnektor

Um das, bei Dialog Semiconductor vorhandene, PXI System mit dem Testboard verbinden zu können, um damit Messungen durchzuführen, wurde eine Steckverbindung dafür vorgesehen. Dabei handelt es sich um eine 2*25 Pin Stiftleiste.

6.4.15 USB / UART Konverter

Um mit dem DP8051 Mikrocontroller kommunizieren zu können, wurde dieser Baustein auf dem Testboard untergebracht. Der DP8051 wird über dessen serielle Schnittstelle angesprochen. Seitens des PC‘s wird die Verbindung mittels der USB Schnittstelle hergestellt.

7 Ergebnisse

Der Pattern Generator wurde zur Validierung mittels eines bereits getesteten Produktes ausprobiert. Die Ergebnisse dieser Versuche zeigten, dass das Gerät die gewünschten Anforderungen erfüllt. Es folgt nun eine kurze Beschreibung des Versuchaufbaus und der gesammelten Ergebnisse.

7.1 Hardware Einstellungen

Mittels der Versorgungsspannungsverbindungen wurde der Chip mit den benötigten Potentialen beaufschlagt. Zusätzlich wurden die Verbindungen der einzelnen Generier- und Vergleichseinheiten auf der Verbindungsmatrix aufgebaut. Dies geschah im rot markierten Feld in der folgenden Abbildung.

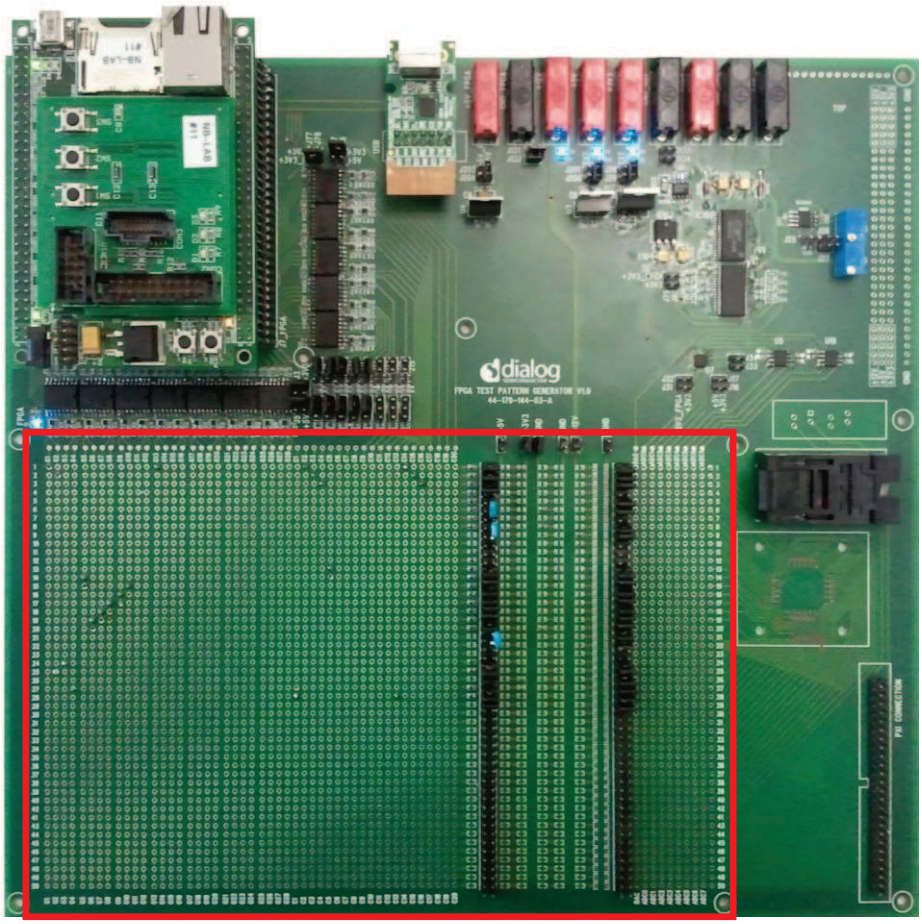


Abbildung 7-1 Testboard

7.2 Benutzerkommandos

Es wurden zwei Benutzerkommandos in C implementiert, um SPI Kommandos zu replizieren, welche das DUT zuerst in den Testmodus und dann in den Scan Modus versetzen. Dies kann über Software erreicht werden, da der Patterngenerator als digitaler Signalgenerator und nicht nur zur Generierung von funktionalen Pattern verwendet werden kann.

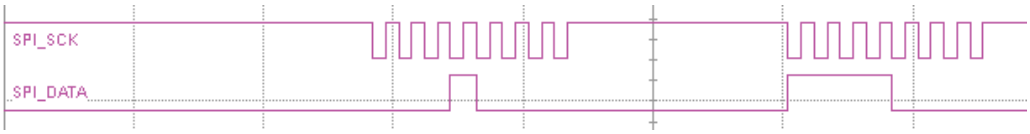


Abbildung 7-2 SPI Testmodus

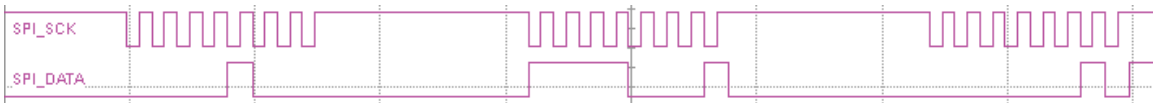


Abbildung 7-3 SPI Scanmodus

7.3 WGL Übertragung

Die WGL- Datei zum Konfigurieren der FPGA Hardware und zum Ablegen der Testvektoren im Flash-Speicher wurde übertragen.

7.4 Pattern starten

Mittels Textkommandos wurden dann die Testpattern gestartet. Durch LEDs wurde angezeigt, ob ein Fehler detektiert wurde oder nicht. Anhand der Testpattern ist sehr gut zu sehen, dass diese in die Scan Flip-Flops hinein- beziehungsweise wieder herausgeschoben werden. Die herausgeschobenen Werte werden verglichen und verifiziert.

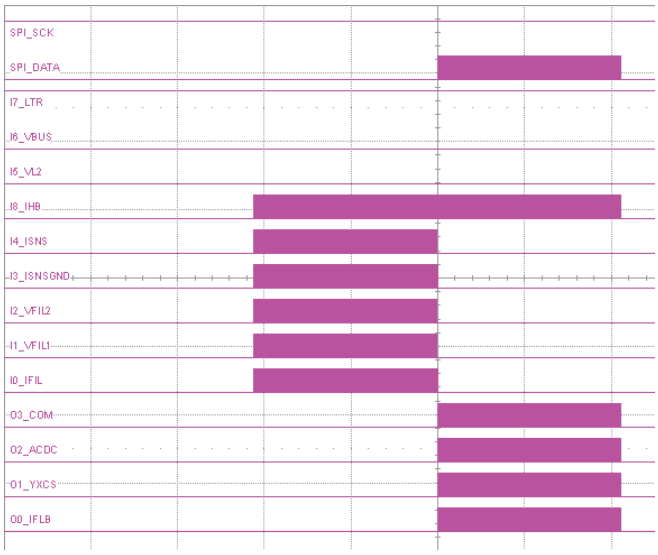


Abbildung 7-4 Testpattern

7.5 Testergebnisse

Es wurden verschiedene Testpattern abgearbeitet, darunter auch fehlerhafte. Dabei wurden alle Fehler detektiert und lokalisiert. Durch Auslesen der Fehlerdaten aus den RAM Blöcken können die Fehler mittels der Vektornummer, der empfangenen Daten, der erwarteten Daten und den momentan generierten Signalen bestimmt und behandelt werden.

8 Verzeichnisse

8.1 Abbildungsverzeichnis

8.2 Tabellenverzeichnis

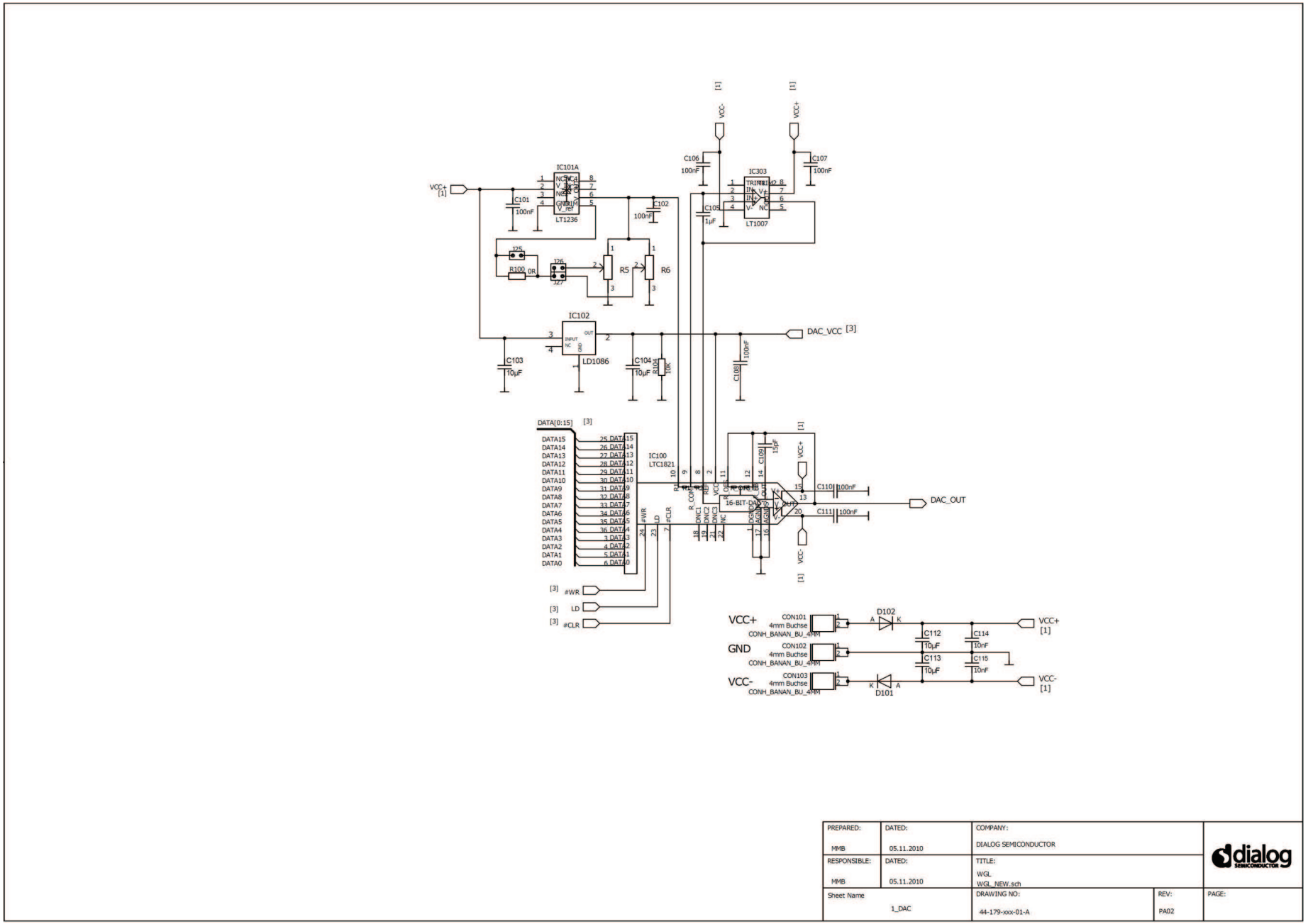
9 Literaturverzeichnis

1. *SDCC Compiler User Guide.* (2011). Abgerufen am 2011 von http://sdcc.sourceforge.net/.
2. Altera. (2007). *Cyclone II Design Handbook.* Abgerufen am 2011 von http://www.altera.com.
3. Digital Core Design. (kein Datum). *DP8051CPU Pipelined High Performance Microcontroller.*
4. Linear-Technology. (kein Datum). 16-Bit Ultra Precise Fast Settling Vout DAC.
5. Synopsis. (Juni 2009). TetraMAX ATPG User Guide.
6. Wen, L.-T. W.-W. (2006). *VLSI Test Principles And Architectures.* Morgan Kaufmann Publishers.

10 Schematics und Layout

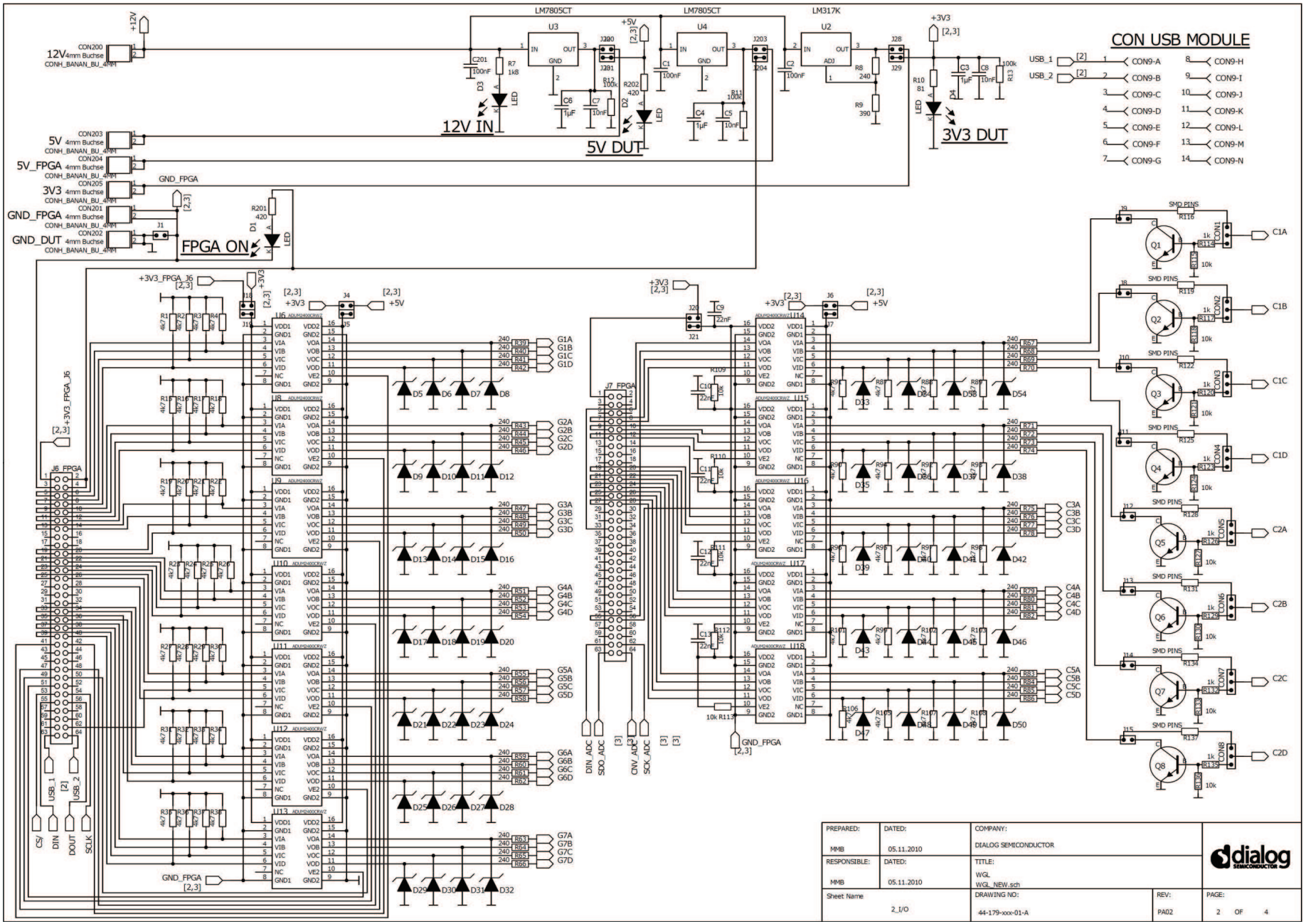
Die folgenden Seiten beinhalten die Schematics der einzelnen Komponenten des Testboards. Zusätzlich dazu wurde auch das Layout des gesamten Testboards, von allen 6 Layern, hinzugefügt.

Digital Analog Umsetzer

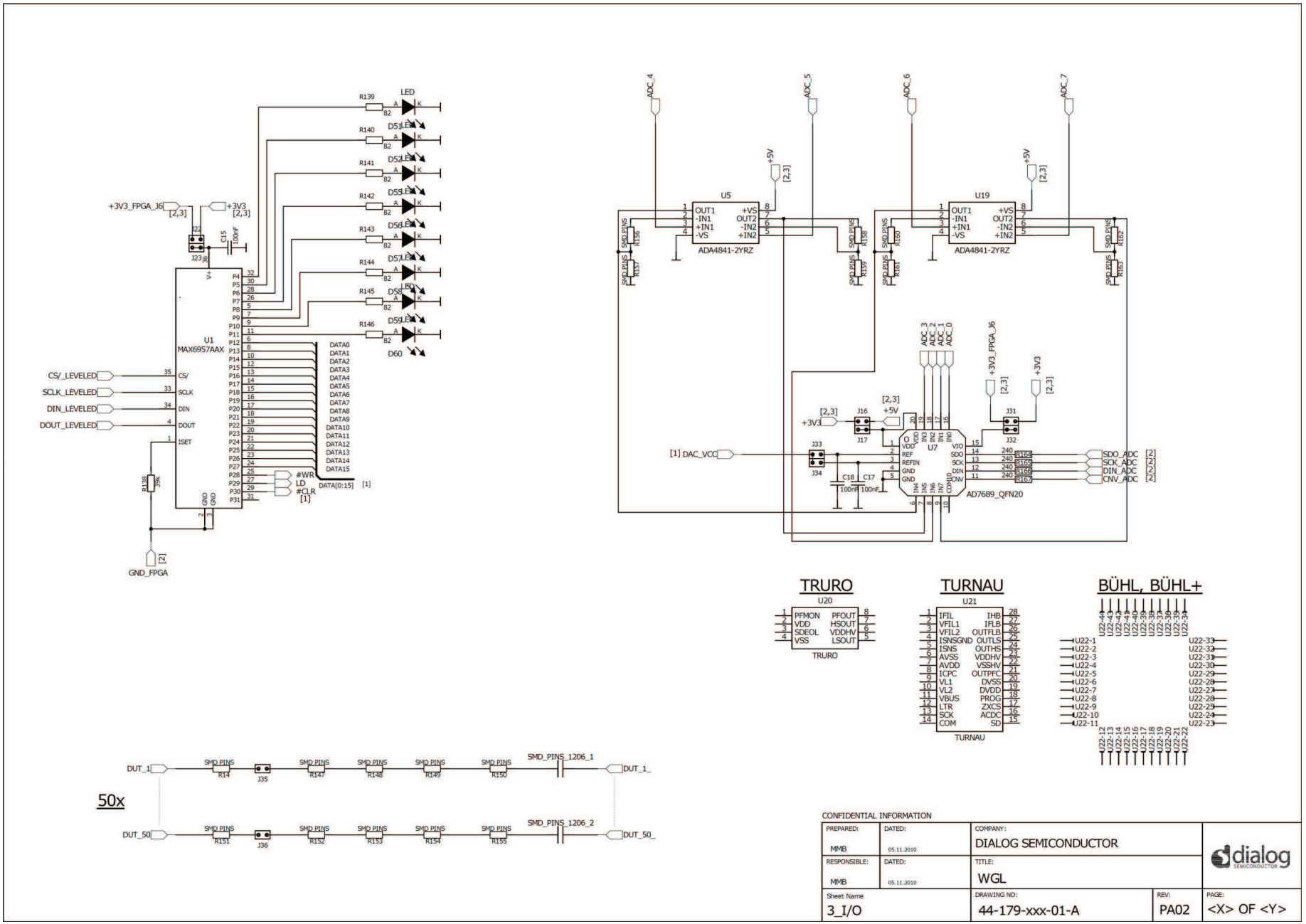


PREPARED: MMB	DATED: 05.11.2010	COMPANY: DIALOG SEMICONDUCTOR	
RESPONSIBLE: MMB	DATED: 05.11.2010	TITTLE: WGL WGL_NEW.sch	
Sheet Name 1_DAC		DRAWING NO: 44-179-xxx-01-A	REV: PA02
			PAGE:

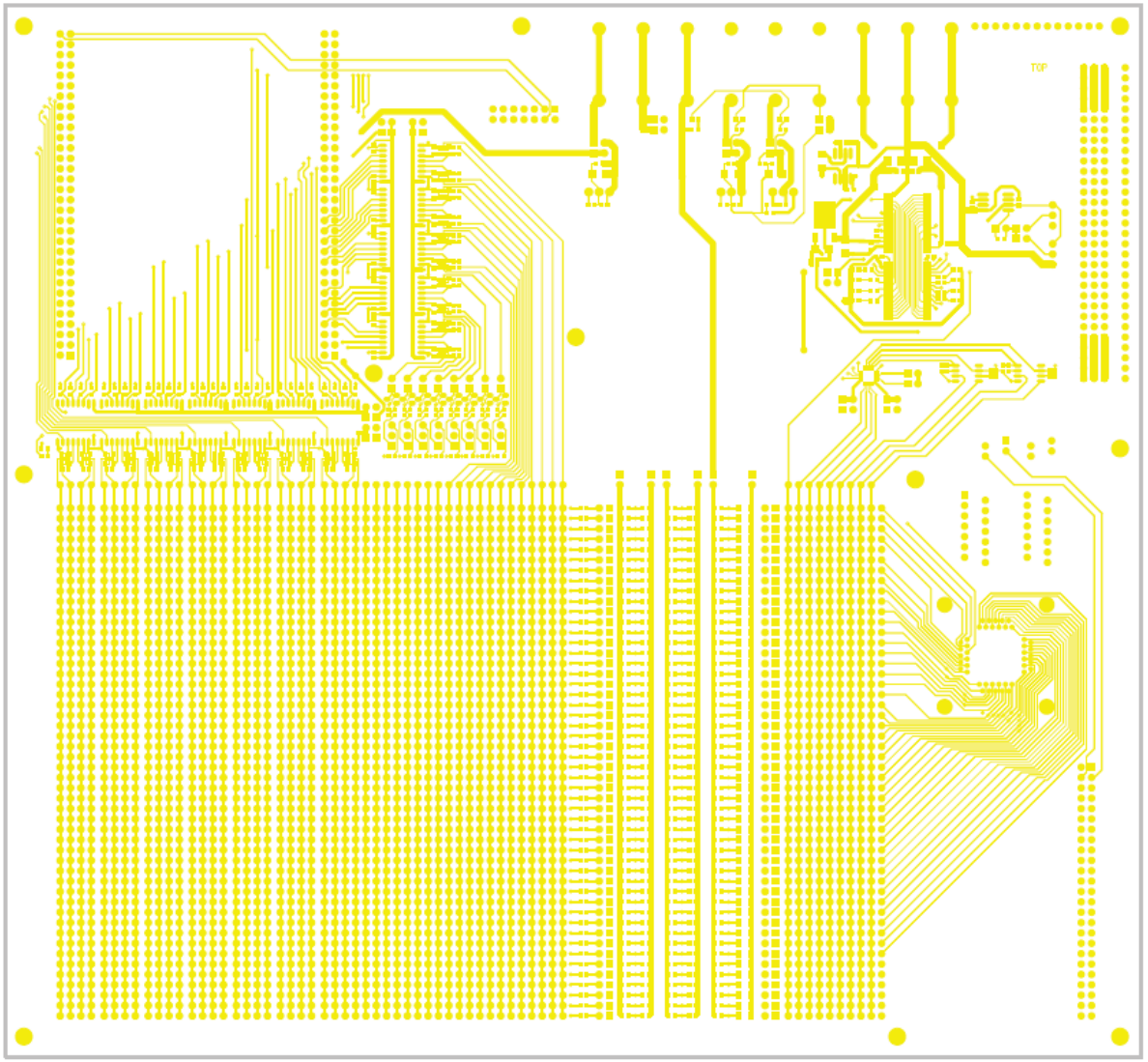
Supplies, Level Shifter, Bipolartransistoren



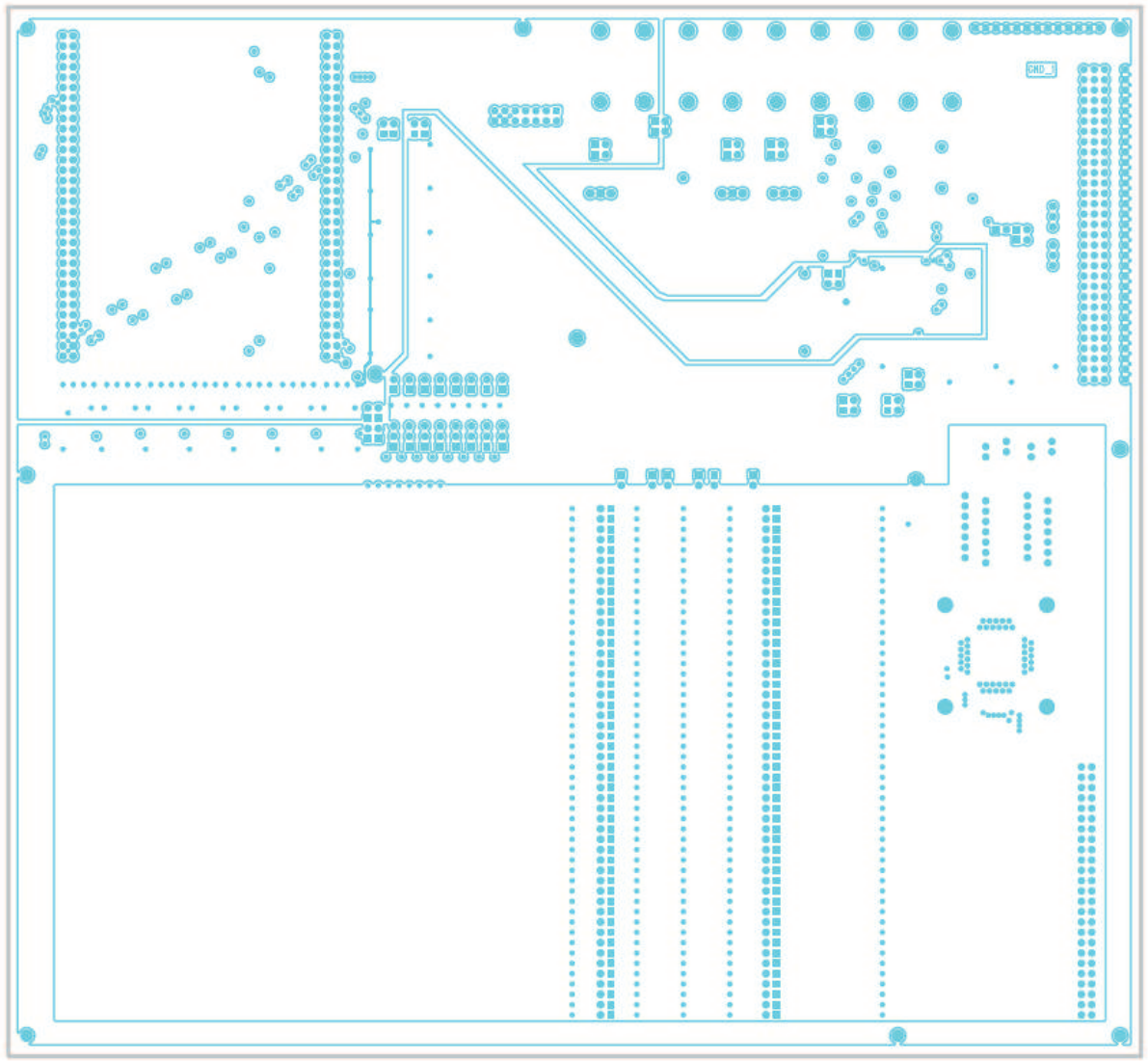
Port Expander, Analog Digital Umsetzer, Sockel



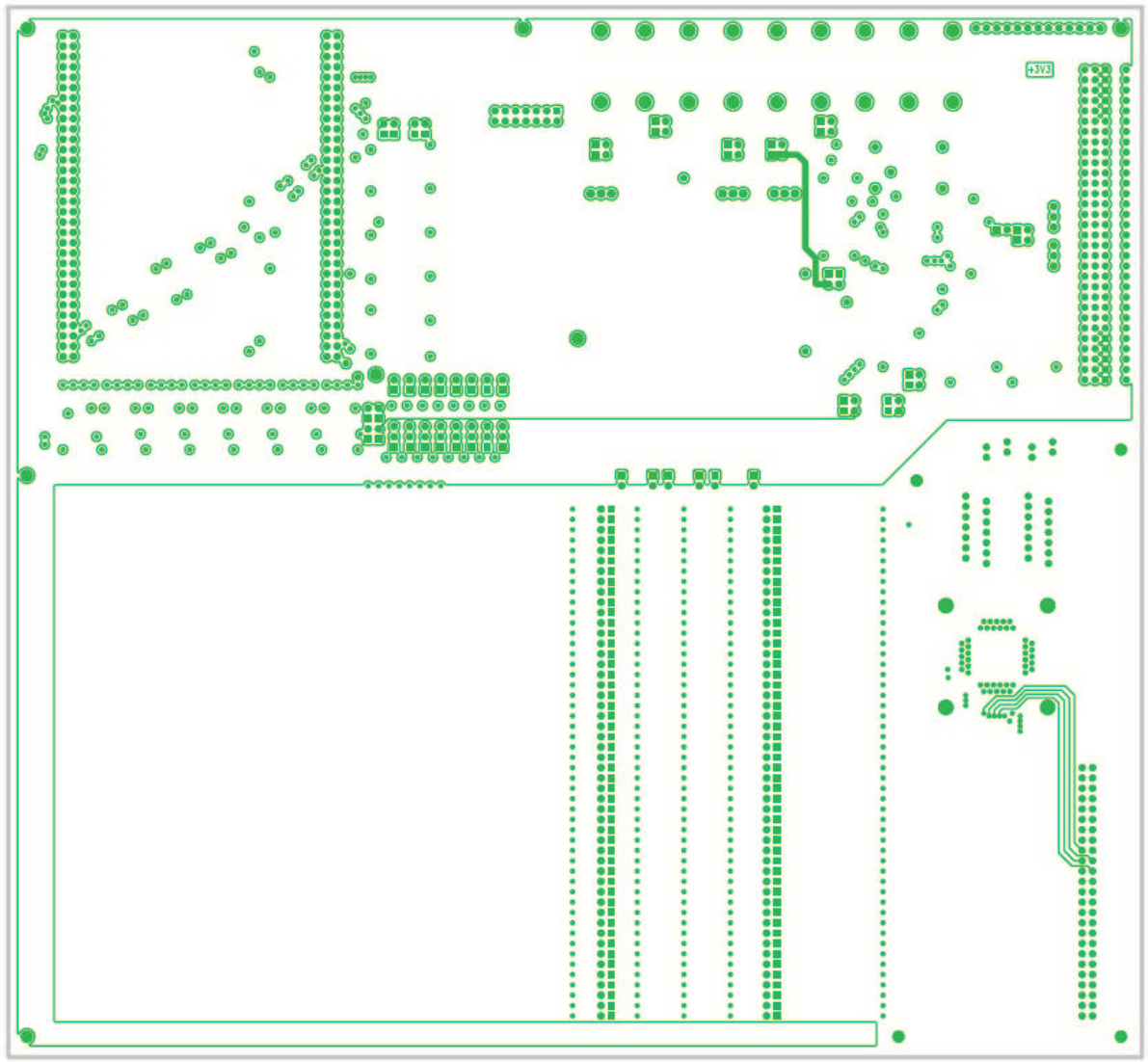
TOP Layout



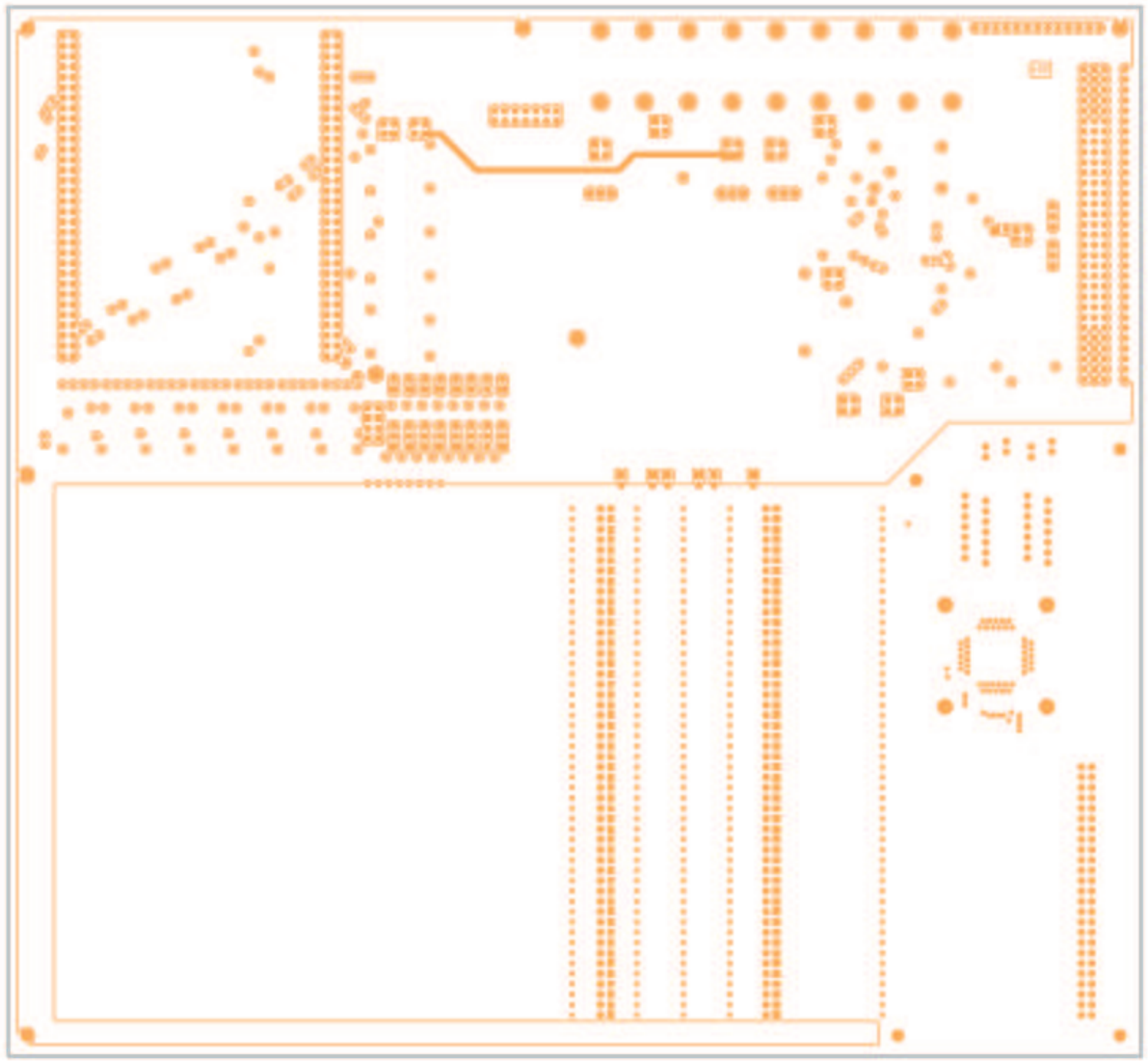
GND 1 Layout



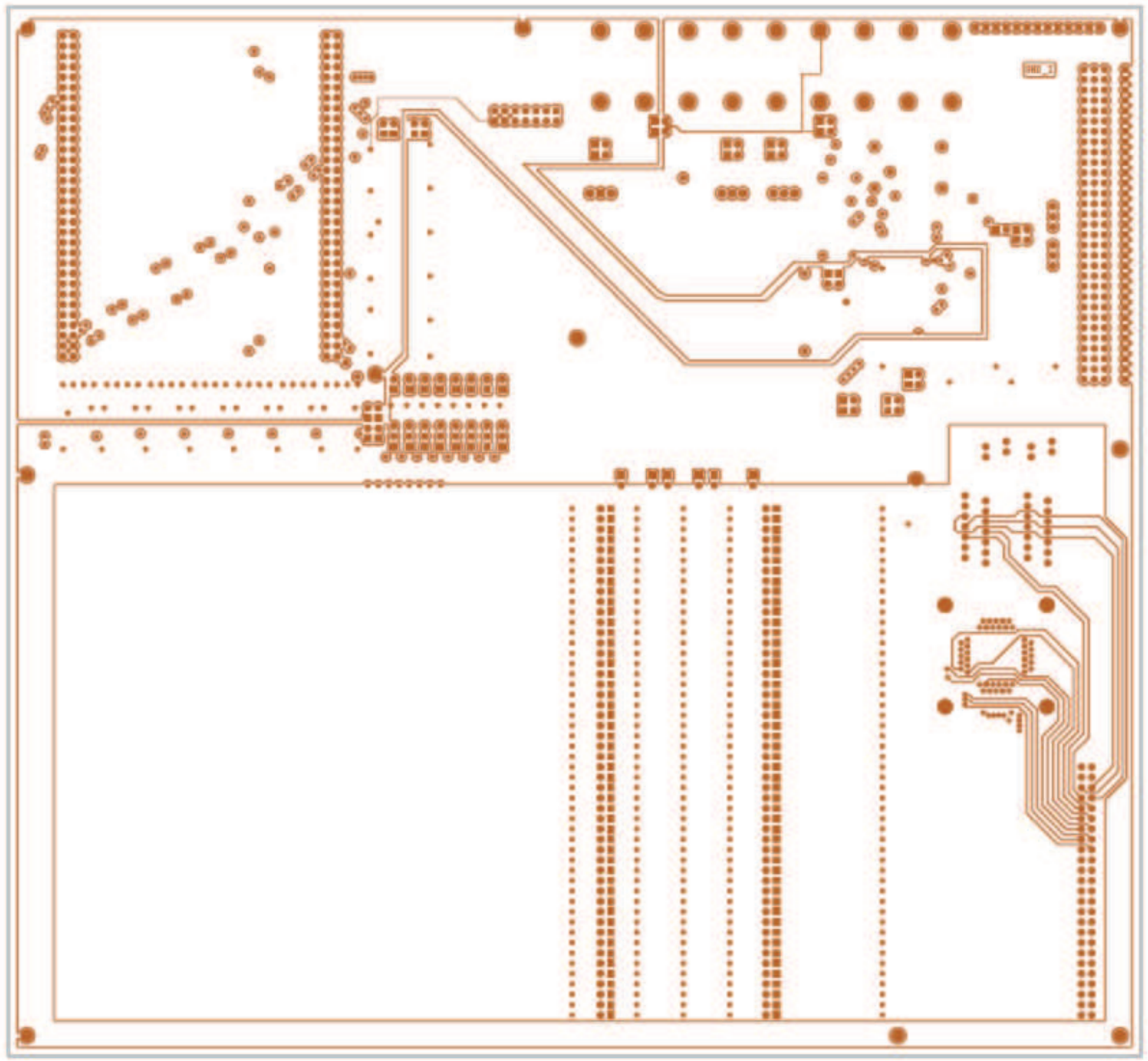
3V3 Layout



5V Layout



GND 2 Layout



BOTTOM Layout