Bachelorarbeit

Voruntersuchungen zur Konstruktion von Experimentalanordnungen
für Fluoreszenzlebensdauermessung unter Nutzung
eingebetteter Systeme

Verfasser:	Lutz Krätzer
Fakultät:	Informatik / Mathematik
Betreuer (IWS):	Dr.-Ing. Frank Sonntag
Betreuer (HTW):	Prof. Dr.-Ing. Robert Baumgartl

Datum: 1. Oktober 2012

Inhaltsverzeichnis

1. Einleitung **4**
1.1. Motivation . . . 4
1.2. Problemstellung . . . 5
1.3. Konventionen . . . 6

2. Theoretische Grundlagen **7**
2.1. Fluoreszenzlebensdauermessung . . . 7

3. Systematische Herangehensweise **9**
3.1. Kontextanalyse . . . 9
3.2. Zustandsanalyse . . . 10
3.3. Komponentenanalyse . . . 12
3.4. Analyse der zeitlichen Abläufe . . . 12
3.5. Einordnung in harte und weiche Echtzeitsysteme . . . 13
3.6. Systematisierung vor dem Hintergrund der Aufgabenstellung . . . 14

4. Material und Methoden **20**
4.1. Hardware-Plattform . . . 20
4.1.1. Mikroprozessor . . . 22
4.1.2. Schnittstellen . . . 22
4.2. Digitales Speicheroszilloskop . . . 22
4.3. Methodik . . . 24

5. Systementwicklung **26**
5.1. Ansteuerung der GPIO-Schnittstelle . . . 26
5.2. Programmentwicklung unter einem Betriebssystem . . . 29
5.3. Programmentwicklung ohne Betriebssystem . . . 31
5.3.1. Crosscompiler / Toolchain . . . 32
5.3.2. Linker-Script . . . 32
5.3.3. Make-File . . . 34
5.3.4. Assembler . . . 34

6. Ergebnisse **36**
6.1. Generierung eines Rechtecksignals mit Betriebssystem . . . 37
6.1.1. Szenario 1 . . . 37
6.1.2. Szenario 2 . . . 37

6.1.3. Szenario 3 . . . 38
6.2. Generierung eines Rechtecksignals ohne Betriebssystem . . . 38
6.2.1. Szenario 4 . . . 38
6.2.2. Szenario 5 . . . 39
6.2.3. Szenario 6 . . . 39
6.2.4. Szenario 7 . . . 40

7. Diskussion **41**
7.1. Zusammenhang Konfiguration Taktmanager und Signalverlauf . . . 44

8. Alternativen **47**
8.1. Feldprogrammierte Gate Arrays (FPGAs) . . . 47

9. Zusammenfassung und Ausblick **48**

10. Eigenständigkeitserklärung **50**

A. Installation des Betriebssystems **51**
A.1. Konfiguration und Administration . . . 52
A.1.1. Netzwerkadapter . . . 52

B. Bildschirmbilder Speicheroszilloskop **54**

C. Quelltexte **59**

D. U-Boot - Essenzielle Befehle **62**

1. Einleitung

1.1. Motivation

> „Die zentralen ökonomischen und gesellschaftlichen Herausforderungen in Deutschland lassen sich ohne die Querschnittstechnologie Embedded Systems nicht lösen.“[3, Seite 6]

> „Embedded Systems sind zunehmend produktprägender Bestandteil mindestens in den drei umsatzstärksten Branchen Deutschlands.“[3, Seite 6]

Diese zwei Thesen, welche in der nationalen Roadmap für eingebettete Systeme ausgeführt werden, geben einen Hinweis auf die wachsende industrielle Bedeutung dieser Technologie.

Der Grund für diese Entwicklung darf in den besonderen Merkmalen von eingebetteten Systemen selbst gefunden werden.

Eingebettete Systeme sind unmittelbar in den Kontext einer technischen Umgebung eingebunden. Innerhalb dieser Umgebung können sie Steuerungs-, Regelungs- und Datenverarbeitungsaufgaben übernehmen. Durch Sensoren und Aktoren wird die Verbindung zur Außenwelt geschaffen.

Durch Leistungssteigerungen der eingesetzten Prozessoren, innerhalb von eingebetteten Systemen, können immer komplexer werdende Anforderungen beispielsweise im Automobilbau, in der Energietechnik und nicht zuletzt in der Medizintechnik bewältigt werden.

Für diese Entwicklung gibt es viele innovative Beispiele. Ein interessanter Vertreter aus dem Bereich der Energietechnik ist das intelligente Energienetz (Smart Grid), welches für die sogenannte Energiewende[1] in Deutschland einen Schlüsselfaktor darstellt.

Ein intelligentes Netz steuert selbständig die Energieflüsse in einem Stromnetz. Durch den großflächigen Ausbau regenerativer Energiequellen werden die Energieerzeuger der Zukunft in einer dezentralen Struktur angeordnet sein und sind durch ein volatiles Erzeugerverhalten gekennzeichnet. Deshalb braucht es ein System, welches die Beschaffenheit

[1] Ein Energiekonzept des Bundesamtes für Umweltschutz, welches bis zum Jahr 2050 vorsieht, dass erneuerbare Energien einen Anteil von 60 % am Bruttoendenergieverbrauch erreichen. [2]

auf der Erzeugerseite und der Verbraucherseite überwacht und automatisch Anpassungen an die entsprechende Bedarfssituation vornimmt.

Dieses Beispiel zeigt, dass ein eingebettetes System unmittelbar in eine physikalische Umgebung eingebunden ist und direkt auf sie einwirkt. Aus dieser Eigenschaft ergeben sich nicht funktionale Anforderungen an ein eingebettetes System, die unbedingt erfüllt werden müssen, damit kein Schaden an Menschen oder Sachen entsteht.

Wichtige nicht funktionale Anforderungen, die bei eingebetteten Systemen eine tragende Rolle spielen, ist die Echtzeitfähigkeit und die Robustheit gegen Störungen des umgebenden Systems. Für die Sicherung der Qualität der Produkte, die auf eingebetteten Systemen basieren, ist die Erfüllung solcher Anforderungen entscheidend.

Im Rahmen der vorliegenden Arbeit soll die Ausführungsgeschwindigkeit und Echtzeitfähigkeit einer konkreten Mikrocontrollerplattform untersucht werden.

Diese beiden Eigenschaften sind für die Entwicklung einer Experimentalanordnung, die den physikalischen Effekt der Fluoreszenz in der Zeitdomäne erfassen soll, wesentlich.

Die Erkenntnisse, die sich aus den Untersuchungen gewinnen lassen, werden in ein Projekt einfließen, in welchem an der Entwicklung eines Systems zur automatischen Substanztestung gearbeitet wird.

Dieses System ist in der Lage verschiedene Untersuchungen an lebenden Zellen automatisiert und über einen langen Zeitraum (<90 Tage) durchzuführen und zu dokumentieren. Eine spezielle Untersuchung ist die Erfassung der Fluoreszenzlebensdauer, welche u.a. Stoffwechselvorgänge von Zellen charakterisiert.

Die Fluoreszenzlebensdauer ist ein Parameter eines physikalischen Vorgangs und wird in 2.1 beschrieben.

1.2. Problemstellung

Die Motivation der vorliegenden Arbeit besteht im Kern darin, Untersuchungen durchzuführen, um einschätzen zu können, inwieweit sich eingebettete Systeme eignen, um die Fluoreszenzlebensdauer zu erfassen.

Dabei muss man sich vergegenwärtigen, dass ein Vorgang quantifiziert werden soll, der im Nanosekundenbereich abläuft.

Dieses Verhalten des physikalischen Systems zieht folgende Fragen nach sich:

1. Kann handelsübliche Rechentechnik, welche in eingebetteten Systemen verbaut ist,

eingesetzt werden, um mit einer physikalischen Umgebung im Nanosekundenbereich zu interagieren?

2. Ist es möglich definierte Zeitschranken einzuhalten?

3. Welche minimale zeitliche Auflösung ist möglich?

4. Was ergibt ein Vergleich zwischen einem System mit und ohne Betriebssystem?

1.3. Konventionen

Zur Unterstützung der besseren Lesbarkeit wurde bei hexadezimalen Adressen und Werten ein senkrechter Strich zwischen Doppeloktetts eingefügt.

2. Theoretische Grundlagen

2.1. Fluoreszenzlebensdauermessung

Der vorliegende Abschnitt beschäftigt sich rudimentär mit dem Thema Fluoreszenz und Fluoreszenzlebensdauermessung. Der Schwerpunkt liegt in der Erarbeitung einiger wichtiger physikalischer Zusammenhänge und entscheidender Rahmenparameter, die als Ausgangspunkt weiterer Überlegungen in den nächsten Kapiteln dienen sollen. Das folgende Zitat lenkt den Blick auf die Motivation, den Effekt der Fluoreszenzlebensdauer verfahrenstechnisch nutzbar zu machen.

> „The measurement of fluorescence signals is of increasing importance in medical and bio-chemical analytics. An important topic of scientific work is the development of fluorescence-based methods for the in-vivo characterisation of human and animal tissues. Aims of these effort among others are the localisation and characterisation of cancer tissue, the characterisation and monitoring of the tissue metabolism and the investigation of ischemic tissue states. In bio-analytics fluorescence measurements are routine in various applications in molecular biology, drug screening and medical diagnostics.“ [30, Seite 423]

Die Vorüberlegungen dieser Arbeit werden in ein Forschungsprojekt einfließen, das sich mit automatischer Substanztestung an lebenden Zellen und Gewebe beschäftigt. Damit die Testergebnisse aussagekräftig sind, wird es entscheidend sein, den Zustand der Proben bis zu einer Zeitspanne von 90 Tagen zu überwachen. Dafür ist die Fluoreszenzlebensdauermessung ein vielversprechendes Verfahren. Ein erheblicher Vorteil der Fluoreszenzlebensdauermessung liegt darin, dass sie eine nicht invasive Methode darstellt, um Zellen und Gewebe zu charakterisieren.

Ein aussichtsreicher Ansatz zur Überwachung von Stoffwechselvorgängen bei lebenden Zellen und Gewebe ist die Quantifizierung der NADH[1]-Fluoreszenzlebensdauer.

Die Fluoreszenzlebensdauermessung basiert auf den physikalischen Effekt der Fluoreszenz. Die Fluoreszenz ist die Emission von Photonen, welche entsteht, nachdem ein Mo-

[1] NADH ist die Abkürzung für Nicotinamidadenindinukleotid. NADH ist ein wichtiges Trägermolekül, welches für die Energieübertragung in Zellen eine entscheidende Rolle einnimmt. Dieses aktivierte Trägermolekül ist auf die Übertragung von energiereichen Elektronen und Wasserstoffatomen spezialisiert.[33, Seite 114, 118]

lekül aus einem angeregten Zustand in seinen Grundzustand zurückfällt. [32, Seite 12] Die Anregung der Proben erfolgt mit einem Laserpuls im Nano- oder Sub-Sekundenbereich. [30, Seite 423] Das Zurückfallen der Moleküle in den Grundzustand bewegt sich bei den meisten Fluorophoren im Spektrum von einigen Pikosekunden bis zu 10 ns. [30, Seite 423] Aus der Abklingkurve von NADH in Abbildung 2.1 lässt sich erkennen, dass die Fluoreszenz bei NADH bereits nach 5 ns abgeklungen ist und danach somit nicht mehr nachzuweisen ist. Die Detektion dieses Vorgangs erfordert deshalb ausreichend schnelle Erfassungstechnik, die Thema der vorliegenden Arbeit ist. Aus mindestens zwei Werten der Intensität lässt sich die Fluoreszenzlebensdauer nach Formel 2.1 bestimmen. [31, Seite 801]

$$\tau_{eff} = \frac{\Delta t}{\ln(\frac{I_a}{I_b})} \tag{2.1}$$

Die Abklingkurve folgt einer exponentiellen Funktion. Schneckenburger gibt die Anregung von NADH bei λ=391 nm und die Detektion bei λ=440-550 nm an. [31, Seite 801] Pfeifer verwendet λ=340 nm zur Anregung und detektiert bei λ=465 nm. Die Fluoreszenzlebensdauer von NADH wird mit τ_{eff}=0,4 ns angegeben. [30, Seite 427]

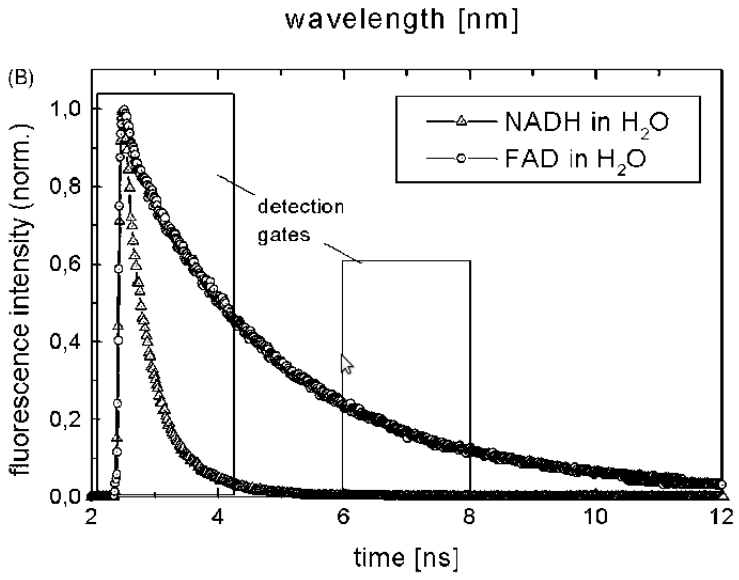


Fig. 4. (a) Normalised fluorescence spectra of NADH and FAD in water. (b) Fluorescence decay curves of NADH and FAD (measured with a time-correlated single photon counting set up).

Abbildung 2.1.: NADH Abklingkurve

3. Systematische Herangehensweise

In der frühen Phase einer Systementwicklung empfiehlt es sich, die Anforderungen an das System zu spezifizieren. Die Frage, die dabei im Mittelpunkt der Betrachtung steht, ist, welche Funktionen beziehungsweise Dienste ein System erbringen soll, um bestimmte Probleme zu lösen oder ein bestimmtes Ziel zu erreichen.

Für die formale Beschreibung des vorliegenden Systems werden Techniken des Requirement Engineering eingesetzt.

3.1. Kontextanalyse

Bei der Kontextanalyse werden alle einzelnen Elemente, die alle Teil des Gesamtsystems sind, identifiziert. Des weiteren wird hervorgehoben, welche Wirkung die Elemente auf das Gesamtsystem ausüben.

Aus dem Kontextdiagramm in Abbildung 3.1 lassen sich die Steuerflüsse und Datenflüsse der einzelnen Elemente erkennen, wobei die Messung der Fluoreszenzdauer als Anwendungsfall der Bezugspunkt ist.

Die Funktionen und Eigenschaften der hervorgehobenen Elemente soll nun näher beschrieben werden.

Subtanztester Der Substanztester ist das eigentliche Produkt und das übergeordnete System, welches alle Funktionen zur automatischen Substanztestung verantwortet. Es sendet einen Auslöseimpuls, welcher die Fluoreszenzlebensdauermessung startet.

Lasersystem Das Lasersystem dient zur Anregung der Probe. Es benötigt einen Steuerimpuls, welcher das Lasersystem aktiviert.

Sensor Der Sensor in der Ausprägung einer CCD-Zeile[1] wandelt emittierendes Licht einer spezifischen Wellenlänge in ein elektrisches Signal um.

A/D-Wandler Der A/D-Wandler digitalisiert das analoge Signal, welches vom Sensor

[1] Eine CCD-Zeile wandelt die Intensität des Lichtes in ein diskretes Signal um. Die Wandlung erfolgt, indem das einfallende Licht in Form von Ladungen übertragen und gespeichert wird. CCD ist die Abkürzung von Charge-coupled Device. [4]

abgegriffen wird. Das Ergebnis steht in Form eines binären Wertes zur Weiterverarbeitung bereit. Dieser Wert repräsentiert die Intensität des Fluoreszenzeffektes.

Sample/Hold-Glied Die beiden Sample/Hold-Glieder sind Teil des A/D-Wandlers. Dies muss aber nicht zwingend der Fall sein. Sie könnten auch als eigenständige Baugruppe im System angeordnet sein. Ob dies für die finale Konzeption zur Entwicklung des Messsystems sinnvoll ist, soll an dieser Stelle vorerst offen gelassen werden. Die Sample/Hold-Glieder stellen sicher, dass der Spannungspegel eines sich schnell ändernden Signals für die Zeit der Quantifizierung konstant gehalten wird. Da für die Berechnung der Fluoreszenzlebensdauer mindestens zwei Intensitätswerte innerhalb kürzester Zeitspannen aufgenommen werden müssen, empfiehlt es sich zwei Sample/Hold- Glieder einzusetzen.

Die Sample/Hold-Glieder benötigen zur Aktivierung ein Steuersignal vom Messsystem.

Datenspeicher Im Datenspeicher werden alle Intensitätswerte abgelegt. Sie stehen hier zur Weiterverarbeitung bereit. Beispielsweise könnten statistische Bereinigungen durchgeführt werden. Es ist sinnvoll den Vorgang der Quantifizierung und der Berechnung zu entkoppeln, damit alle Systemressourcen dem Prozess der Signalwandlung zur Verfügung stehen.

Dieser Gedanke sollte bei der Entwicklung der Softwarearchitektur einfließen.

3.2. Zustandsanalyse

Nachdem der Systemkontext beschrieben wurde und die einzelnen Elemente des Kontextes bekannt sind, ist es möglich das Systemverhalten näher zu untersuchen. Das Systemverhalten offenbart sich durch die Funktionalität der einzelnen Elemente, die zu unterschiedlichen Zeitpunkten zur Verfügung gestellt werden. Das Gesamtsystem wird demnach aus dynamischen Objekten gebildet, die Verhalten zeigen. [5, Seite 230]

Für die Darstellung dieses Verhaltens eignet sich ein Zustandsdiagramm, welches in der Abbildung 3.2 dargestellt ist.

Das abgebildete Zustandsdiagramm zeigt eine neue interessante Sicht auf das zu entwickelnde System. Es wird deutlich, dass es Abhängigkeiten zwischen den einzelnen Objekten des Systems gibt.

Die einzelnen Elemente aus dem Kontextdiagramm finden sich in der Definition ihrer Zustände im Zustandsdiagramm wieder. Zustände sind auch ineinander verschachtelt. Dies gibt einen Hinweis darauf, dass es Objekte geben wird, die Teil eines übergeordneten Objektes sind. Entsprechend des Diagramms in Abbildung 3.2 gibt es beispielsweise

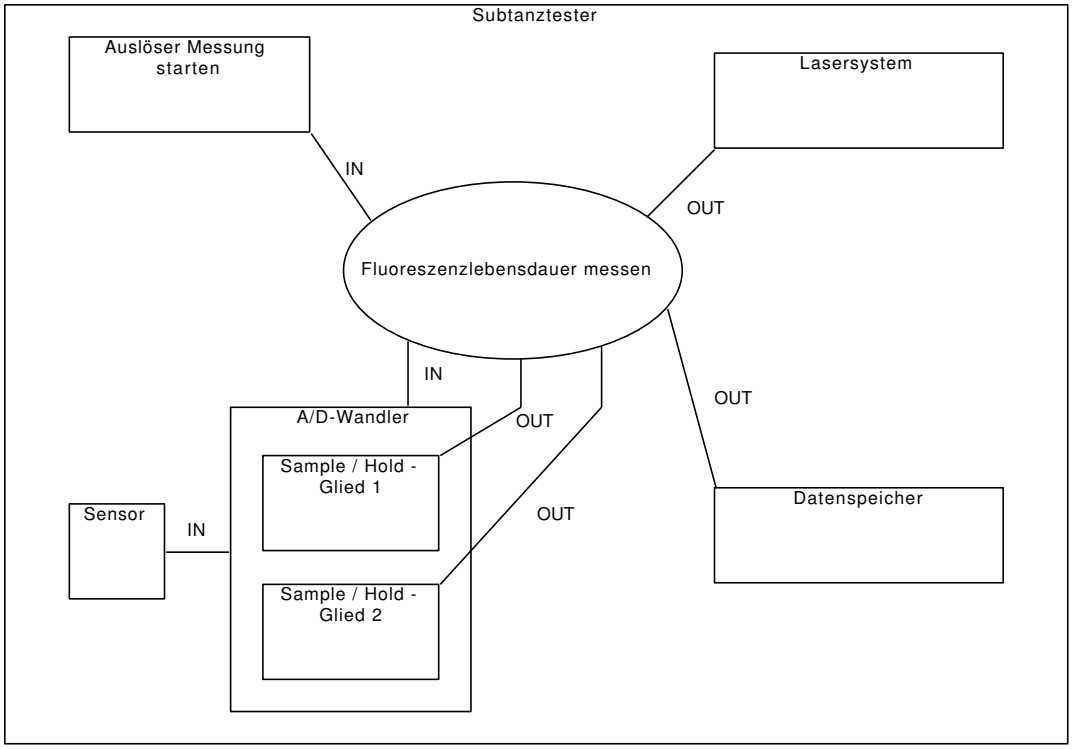


Abbildung 3.1.: Kontextdiagramm Messsystem

den Zustand *A/D-Wandler aktiv*, welcher die Zustände der Sample/Hold-Glieder einschließt. Im Kapitel *Kontextanalyse* wurde bereits darauf hingewiesen, dass die beiden Sample/Hold-Glieder nicht zwingend Teil des A/D-Wandlers sein müssen. Diese Frage soll in einem weiterführenden Kapitel im Rahmen einer Diskussion erörtert werden.

Ein weiterer interessanter Aspekt zeigt sich in der Betrachtung der Transitionen. Es wird deutlich, dass es im System Informationen in Form von Nachrichten gibt, die Transitionen bewirken. Die Überlegung, in welcher Art diese Nachrichten vorliegen und übertragen werden, führt zu den Schnittstellen. Eine Vertiefung erfolgt in der Komponentenanalyse auf Seite 12.

Eine besondere Möglichkeit einer Transition wird durch den Faktor *Zeit* verursacht. Dies ist bei einem Sample/Hold-Glied der Fall. Die *Einstellzeit* ist hier ein wichtiger technischer Parameter. Erkennbar ist, dass eine Quantifizierung eines analogen Wertes nur durchgeführt werden darf, wenn sich ein stabiler Wert am Sample/Hold-Glied eingestellt hat.

Eine Sonderstellung nimmt der Sensor im Diagramm ein. Er besitzt nur einen Zustand. Einfallendes Licht wird kontinuierlich in ein elektrisches Signal gewandelt und steht des-

halb dem A/D-Wandler dauerhaft zur Verfügung. Ein Auslöseimpuls ist nicht notwendig. Er kann deshalb als passives Element angesehen werden.

3.3. Komponentenanalyse

Nach der Kontextanalyse und Zustandsanalyse ist es nun sinnvoll, die Grenzen und die Schnittstellen des Systems näher zu untersuchen.

Zur Darstellung eignet sich dafür ein Kompositstrukturdiagramm, welches in Abbildung 3.3 zu sehen ist.

Es fördert das Verständnis über die interne Zusammensetzung des Systems. Die abgebildeten Objekte sind nun nicht mehr nur abstrakt bezeichnet, sondern repräsentieren tatsächlich vorkommende Baugruppen. Dies gilt auch für die Schnittstellen.

Die Kardinalität der vorkommenden Elemente lässt sich ebenfalls ablesen. Der A/D-Wandler als Baugruppe ist zweifach ausgeführt. Die anderen Baugruppen einfach.

Mit Hilfe des Kompositstrukturdiagramms lässt sich weiterführend analysieren, wo eine Kapselung von Funktionalität sinnvoll ist. Es kann nun eine Abgrenzung stattfinden zwischen Baugruppen bei denen eine Eigenentwicklung notwendig ist und Baugruppen, deren Entwicklung man auslagert. An dieser Stelle zeigt sich der Sinn von Schnittstellen. Über Schnittstellen lässt sich im Allgemeinen auf eine Funktionalität zurückgreifen, deren genaue Implementierung verborgen ist. Deutlich wird dies beim A/D-Wandler. Ein A/D-Wandler lässt sich benutzen, ohne dass man detailliert über seinen inneren Aufbau Kenntnis haben muss. Gleiches gilt für das Rechnersystem und das Lasersystem.

Abschließend soll an dieser Stelle festgehalten werden, dass eine Spezifikation der Schnittstellen unumgänglich ist. Des weiteren muss herausgearbeitet werden, bei welchen Komponenten eine Eigenentwicklung sinnvoll ist und worin ihre Funktionalität besteht.

Diese Überlegungen werden unter Systematisierung vor dem Hintergrund der Aufgabenstellung angestellt.

3.4. Analyse der zeitlichen Abläufe

Die bisherigen Betrachtungen haben zeitliche Aspekte des zu entwickelnden Systems ausgeblendet. Das ist aber im Sinne der Lösung, die angestrebt wird, eine unvollständige Sichtweise. Entscheidend sind natürlich nicht nur die Präzedenzen zwischen den Aktivierungszuständen der Systemelemente (siehe 3.2), wenn Funktionalität bereit gestellt wird, sondern auch in welchen zeitlichen Schranken dieses zu erfolgen hat.

Unter dem Abschnitt Fluoreszenzlebensdauermessung auf Seite 7 wurde ausgeführt, dass ein physikalischer Vorgang quantifiziert werden soll, der bestimmten zeitlichen Gesetzmäßigkeiten folgt.

Diese zeitlichen Gesetzmäßigkeiten, die Beachtung finden müssen, werden im folgenden hervorgehoben:

1. Die exponentielle Abklingkurve, welche erfasst werden soll, bewegt sich im zeitlichen Intervall von $\{\tau \in \mathbb{R} \mid 1\ ns \leq \tau \leq 3\ ns\}$.

2. Die zeitliche Auflösung beträgt $1\ ns$.

3. Der Abstand zwischen der Anregung und dem maximalen Extrema der Abklingkurve ist $\geq 1\ ns$.

   Die Anregung wird bei $\tau_0 = 0\ ns$ angenommen.

Die hervorgehobenen zeitlichen Rahmenbedingungen werden im gezeiteten Sequenzdiagramm in Abbildung 3.4 dokumentiert.

3.5. Einordnung in harte und weiche Echtzeitsysteme

Nach Stankovic sind harte Echtzeitsysteme dadurch gekennzeichnet, dass die Korrektheit des Systems nicht nur davon abhängt, ob das System logisch richtige Ergebnisse liefert, sondern auch innerhalb welcher Zeit das Ergebnis zur Verfügung steht. [8, Seite 1]

Falls ein hartes Echtzeitsystem definierte Zeitschranken verletzt, kann diese ernste Konsequenzen nach sich ziehen.

Die zeitliche Spezifikation eines Echtzeitsytems kann aber auch durch die Betrachtung einer Nutzen- (oder auch Kosten-) Funktion vorgenommen werden. Diese Funktion gibt den Nutzen an, den die Bereitstellung des Ergebnisses eines Services zu jedem möglichen Zeitpunkt zum Gesamtnutzen des Systems beiträgt. Wird dieser Nutzen negativ, so entsteht ein finanzieller oder personeller Schaden. [9, Seite 18]

In Abbildung 3.5 ist die Nutzenfunktion eines weichen und eines harten Echtzeitsystems dargestellt. Im Gegensatz zur Nutzenfunktion des weichen Echtzeitsystems ist die Nutzenfunktion des harten Echtzeitsystems sprunghaft. Wird eine Zeitschranke (Deadline) verpasst, nimmt der Nutzen abrupt einen negativen Wert an.

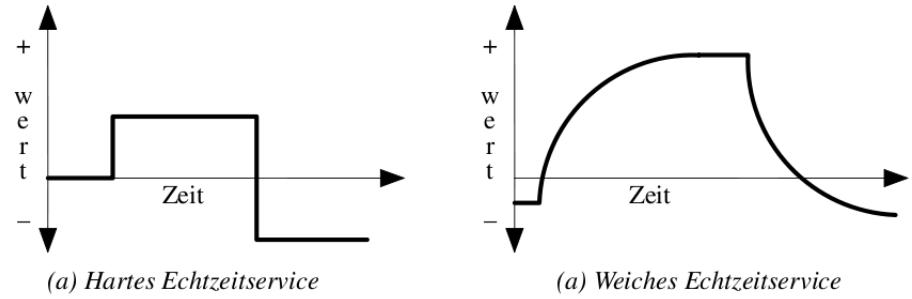
Es soll nun eine Einschätzung getroffen werden, welches Verhalten das zu entwickelnde System, welches Gegenstand der vorliegenden Arbeit ist, aufweist.

Der Nutzen dieses Systems erwächst dann, wenn es korrekte und brauchbare Messer-

gebnisse liefert. Falls Zeitschranken nicht eingehalten werden, liefert das System keine Messwerte, oder sie sind unbrauchbar. Das ist der Tatsache geschuldet, dass eine zeitliche exakte Steuerung des A/D-Wandlers und des Lasersystems zur Anregung der Probe entscheidend für die korrekte Quantifizierung der Fluoreszenzlebensdauer ist.

Wenn eine Langzeituntersuchung einer Probe (<90 Tage) durchgeführt wird und die Aussagekraft der Untersuchungsergebnisse nicht gegeben ist, weil die Dokumentation der Fluoreszenzlebensdauer nicht lückenlos erfolgte, hat dies unmittelbare Konsequenzen. Es kommt bei der Verfehlung einer Zeitschranke nicht nur zu einer Minimierung des Nutzens, sondern es entsteht finanzieller Schaden.

Deshalb darf davon ausgegangen werden, dass es sich um ein hartes Echtzeitsystem handelt.



(a) Hartes Echtzeitservice *(a) Weiches Echtzeitservice*

Abbildung 3.5.: Zeitanforderungen für Echtzeitservices

3.6. Systematisierung vor dem Hintergrund der Aufgabenstellung

Das Ziel der vorliegenden Arbeit ist es, Vorüberlegungen zur Lösung eines Teilproblems für die Entwicklung einer prototypischen Experimentalanordnung anzustellen. Damit die Fragestellungen der Arbeit, welche unter dem Abschnitt Problemstellung auf Seite 5 aufgezählt sind, beantwortet werden können, ist eine Eingrenzung auf die Teile des Systems notwendig, die für das Problem relevant sind.

Darüber hinaus werden die Ergebnisse der Analyse aus den Abschnitten Kontextanalyse, Zustandsanalyse, Komponentenanalyse und Analyse der zeitlichen Abläufe zusammengefasst und diskutiert. Auf dieser Grundlage werden dann weitere Überlegungen angestellt, die zum Aufbau eines konkreten Systems führen sollen, an dem Messungen vorgenommen werden können.

Folgendes wird aus den letzten Abschnitten abgeleitet:

1. Das Lasersystem und die Sample/Hold-Glieder benötigen für ihrer Aktivierung ein Steuersignal. Das korrekte Zeitverhalten dieser Steuersignale ist entscheidend für die Funktion des Systems. Werden Zeitschranken nicht eingehalten, liefert das System keine brauchbaren Ergebnisse. Der Nutzen des Systems ist direkt mit seiner Echtzeitfähigkeit verknüpft.

2. Eine Kapselung von Funktionalität ist sinnvoll. Eine Eigenentwicklung aller Komponenten, die zum System gehören, ist nicht empfehlenswert. Deutlich wird das bei der Realisierung der Wandlung des analogen Signals in ein digitales Signal. Es wurde bereits angedeutet, dass die Sample/Hold-Glieder Teil des A/D-Wandlers sein können, aber nicht müssen. Zur Reduzierung des Schaltungsaufwandes, der auch zu einer Verkürzung der Entwicklungszeit führen dürfte, empfiehlt es sich einen integrierten Schaltkreis einzusetzen, in dem die komplette Funktionalität eines Wandlungsprozesses vereinigt ist. Es muss lediglich sichergestellt werden, dass über eine geeignete Schnittstelle auf die Funktionen zugegriffen werden kann. Eventuell sind auch Pegelanpassungen zwischen den Komponenten des Systems nötig.

3. Die charakteristischen technischen Parameter des A/D-Wandlers müssen mit den Anforderungen des Systems abgestimmt sein.

   Es soll betont werden, dass die *Umsetzungszeit (Conversion Time)*[2] des A/D-Wandlers ein entscheidender Parameter für das Erreichen der Echtzeitfähigkeit ist.

4. Echtzeitfähigkeit wird nur erreicht, wenn alle zeitkritischen Komponenten, zur Lösung der Problemstellung, betrachtet werden. Anders ausgedrückt könnte die Verletzung von Zeitschranken einer Komponente die Echtzeitfähigkeit des gesamten Systems gefährden.

   Zur Messung der Fluoreszenzlebensdauer sind verschiedene Komponenten notwendig, die als zeitkritisch eingestuft werden müssen. Dazu gehört der A/D-Wandler, das Lasersystem und das System, welches die Steuersignale generiert.

   Die vorliegende Arbeit konzentriert sich dabei auf die Betrachtung des Systems zur Generierung der Steuersignale. Untersuchungen über das Lasersystem als Komponente ist nicht Gegenstand der Arbeit. Es gehört aber zum Systemkontext und es offenbart sich, dass es einen Logikeingang zur Verfügung stellen muss, über welchen die Anregung gestartet werden kann.

5. Zur Generierung von Steuersignalen soll eine geeignete Mikrocontrollerplattform

[2] Die Zeit die vergeht, vom Start der Umsetzung bis zur Ausgabe der gültigen Bitkombination. Die meisten Umsetzer verfügen über einen speziellen Logikeingang (Start of Conversion), über den der Start ausgelöst wird und einen Logikeingang (End of Conversion), der das Ende der Umsetzung anzeigt. [6, Seite 377]

zum Einsatz kommen. Eine zweckmäßige Schnittstelle für den Abgriff der Steuersignale stellen die General-Purpose-I/O-PortPins (GPIO)[3] dar.

[3] Eine GPIO-Schnittstelle kann einzelne Port-Pins eines Mikrocontrollers in den High- bzw. Low-Zustand versetzen. Sie weist aber auch Eigenschaften auf, die sie interessant für die Schnittstelle zu einer Analogschaltung werden lassen. Zusätzlich zum digitalen Ein- und Ausgang enhalten diese Designs programmierbare Pull-up-/Pull-down-Lastbausteine sowie alternative Funktionen, bei denen On-Chip-Peripherieeinheiten wie UARTs, Zähler/Timer, Codec-Schnittstellen usw. die Steuerung der Portfunktionen übernehmen. [7]

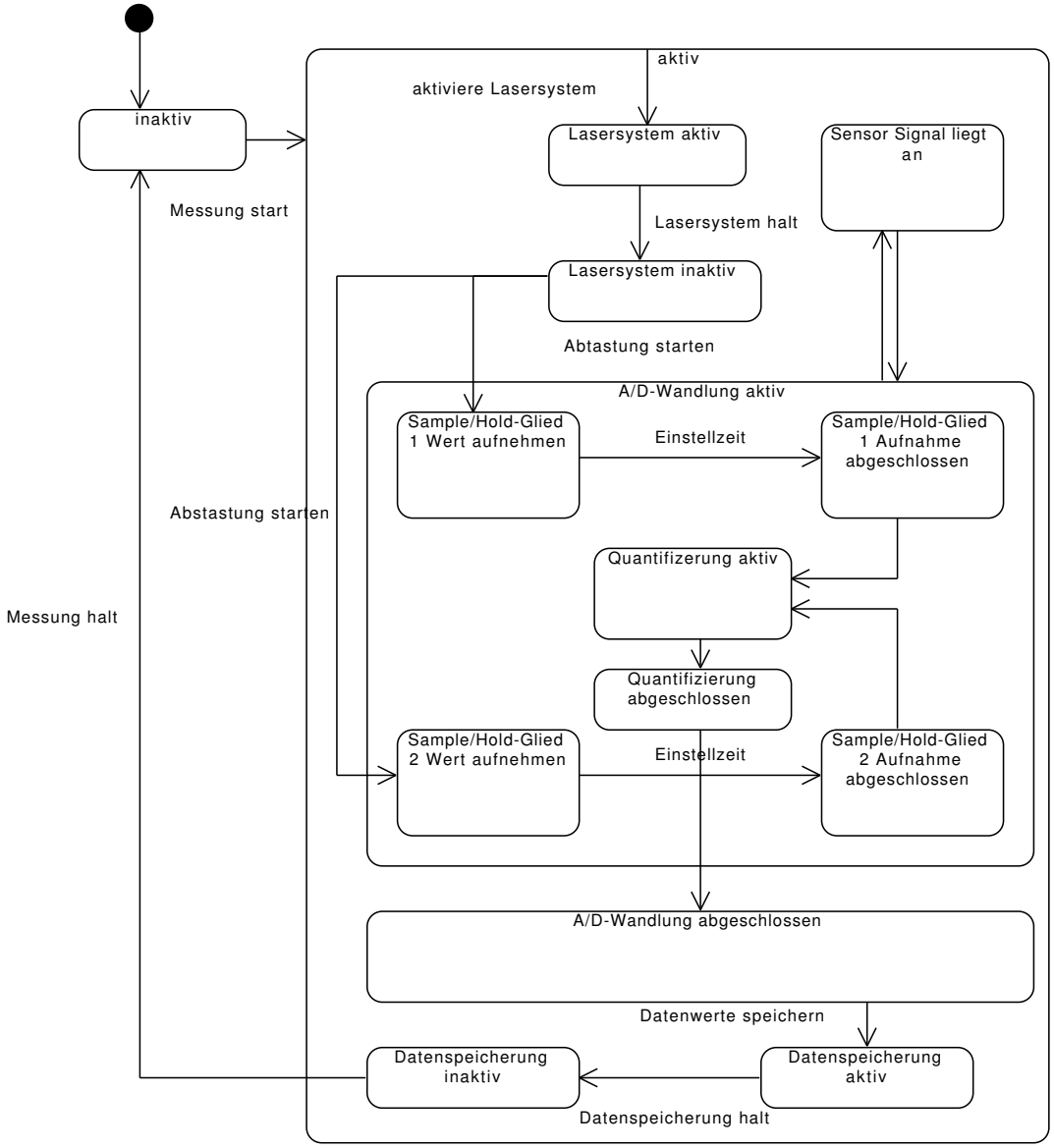


Abbildung 3.2.: Zustandsdiagramm Messsystem

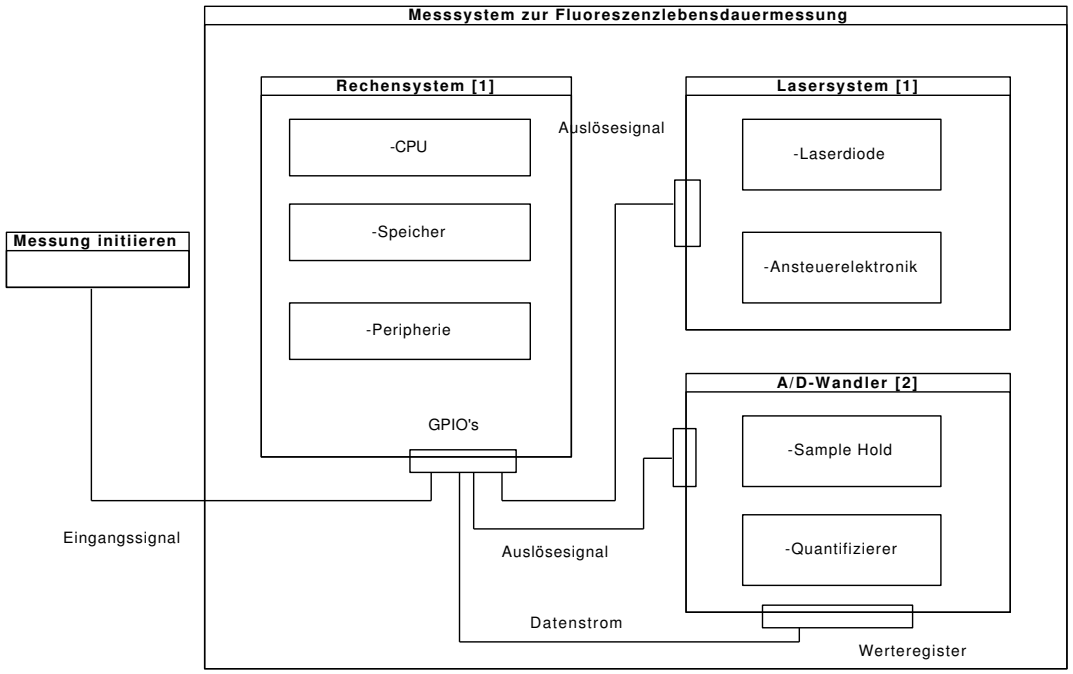


Abbildung 3.3.: Strukturdiagramm Messsystem

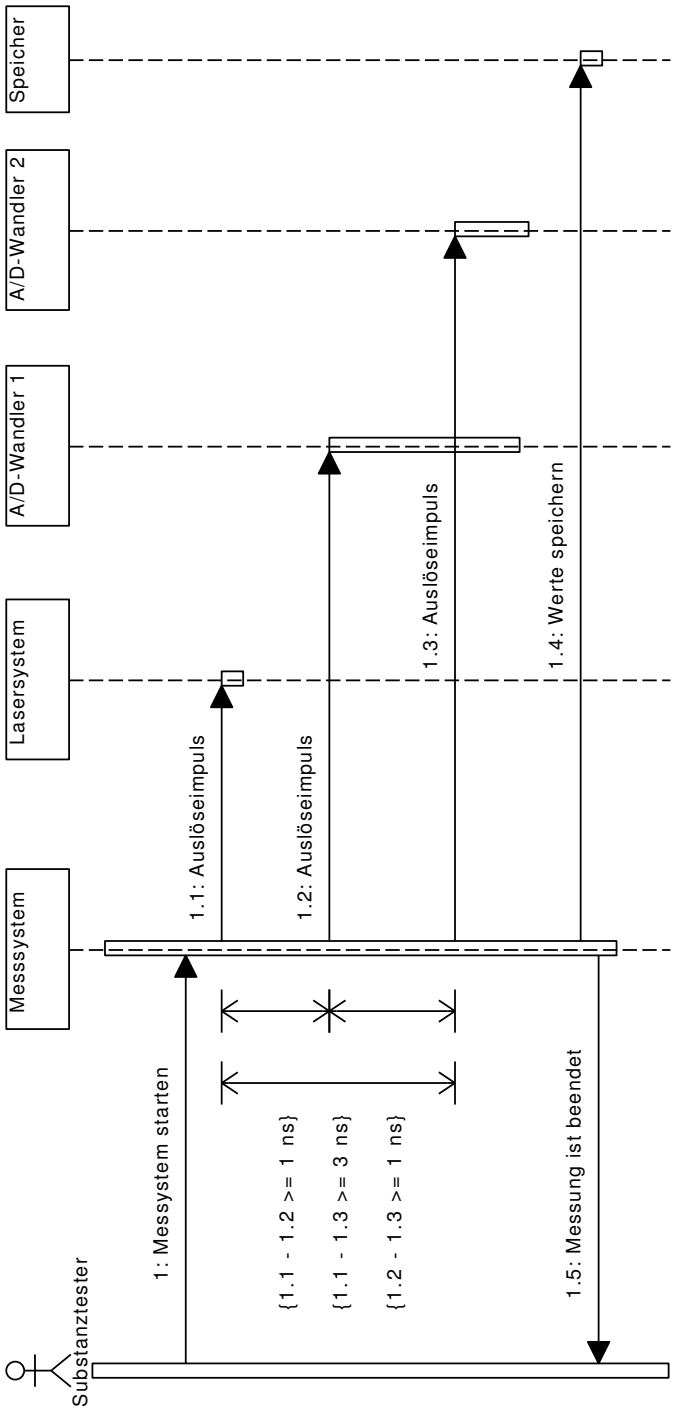


Abbildung 3.4.: Sequenzdiagramm Messsystem

4. Material und Methoden

Ausgehend von den Ergebnissen des letzten Kapitels soll nun erörtert werden, welche konkrete Hardware eingesetzt werden soll, um eine Lösung der Problemstellung entwickeln zu können. Weiterführend wird die Methode beschrieben, die für die Ermittlung der Ergebnisse zur Anwendung kommt.

4.1. Hardware-Plattform

Es wurde bereits erwähnt, dass ein handelsübliches eingebettetes System vor dem Hintergrund der Aufgabenstellung untersucht werden soll. Dieses System sollte einige Anforderungen erfüllen, die im folgenden formuliert werden:

1. Es muss eine Schnittstelle vorhanden sein, über die es möglich ist, die digitalen Signale zur Steuerung des A/D-Wandlers und des Lasersystems abzugreifen. Ein Zugriffspunkt zur Herstellung einer elektrischen Verbindung muss zur Verfügung stehen.

2. Die Plattform sollte eine breite Unterstützung für offene Betriebssysteme anbieten. Da es sich um die Entwicklung eines Prototyps handelt, macht es noch keinen Sinn sich auf ein bestimmtes Betriebssystem festzulegen. Es ist empfehlenswert sich die Möglichkeit offen zu halten, nach einem Leistungsvergleich von verschiedenen Betriebsystemen, das geeignete auszuwählen.

3. Das Kennzeichen einer prototypischen Entwicklung ist die Tatsache, dass die Spezifikation von bestimmten Komponenten und die Art ihrer Schnittstellen in dieser Phase der Systementwicklung noch nicht vollständig bekannt ist. Deshalb sollte die Hardware-Plattform eine großes Spektrum an Schnittstellen besitzen, damit potentielle Systemkomponenten unterstützt werden können.

4. Betrachtet man Zweck des zu entwickelnden Systems, wird deutlich, dass ein physikalischer Vorgang digitalisiert werden soll, der relativ schnell abläuft (siehe Analyse der zeitlichen Abläufe). Dieser Aspekt zieht die Überlegung nach sich, dass die Verarbeitungsleistung des Mikroprozessors ausreichend dimensioniert sein muss.

   Die Verarbeitungsleistung eines Mikroprozessors wird wesentlich durch zwei Para-

meter bestimmt:

Taktrate Ein Mikroprozessor folgt einem internen Befehlszyklus, der den Ablauf für den Befehlszugriff und die Befehlsausführung beinhaltet. Die Schrittdauer t_S, welche benötigt wird, um einen Zyklus zu durchlaufen wird durch die Taktrate bestimmt. Die Taktrate C_{CPU} ist der Reziprokwert der Schrittdauer (siehe 4.1) .

$$C_{cpu} = \frac{1}{t_s} \tag{4.1}$$

Es sei darauf hingewiesen, dass es sich hier exakt gesehen, um den Maschinenbefehlszyklus handelt. Die vorkommenden Mikrobefehle eines Mikroprozessors sind für den Programmierer nicht zugänglich und deren Ausführungsgeschwindigkeit wird durch die oben definierte Taktrate auch nicht unmittelbar beschrieben. [10, Seite 28]

ICC Der ICC (Instruction Cycle Count) gibt an, wieviele Befehle pro Takt abgearbeitet werden können. Dieser Parameter lässt auf die Qualität der eingesetzten Rechnerarchitektur schließen. Durch den Einsatz einer Pipeline[1] ist eine positive Beeinflussung des Parameters möglich.

Nachdem Anforderungen an eine Hardware-Plattform formuliert wurden, soll ein konkretes Evaluations-Board und seine relevanten Eigenschaften vorgestellt werden. Abbildung 4.1 zeigt das sogenannte Beagle-Board, welches im Handel zu einem Stückpreis von $\approx$ 130 € erworben werden kann. (`http://www.digikey.de/product-detail/de/BEAGLEXM/296-25798-ND/2224866`)

Abbildung 4.1.: Beagle-Board

[1] Eine Pipeline ist eine Einheit eines Mikroprozessors, die eine zeitliche Überlappung der einzelnen auszuführenden Befehle ermöglicht. [11, Seite 238]

4.1.1. Mikroprozessor

Nach [12, Seite 14] ist im Beagle-Board ein ARM-Cortex A8 kompatibler Mikroprozessor mit der Bezeichung **DM3730** verbaut. Dieser Mikroprozessor wird durch Texas Instruments gefertigt und vertrieben.

Taktrate Es werden Taktraten von 300 MHz, 600 Mhz, 800 Mhz und 1 Ghz unterstützt. [14, Seite 1]

Speicher Die Speichergröße beträgt 512 MByte. [12, Seite 16] Es ist eine MMU[2] integriert, die zur Verwaltung des virtuellen Speichers eingesetzt wird.

Pipeline Die superskalare Pipeline besitzt 13 Stufen. [17]

Cache Der Cache hat ein Größe von 32 KByte (Level 1) bzw. 256 KByte (Level 2). [14]

Befehlssatz Der Thumb-Befehlssatz wird unterstützt. [17] Dieser Befehlssatz besteht nur aus 16 Bit breiten Befehlen. Dadurch wird die Code-Dichte signifikant erhöht.

4.1.2. Schnittstellen

GPIO → 22 GPIO-Pins können über den Expansions-Steckplatz (28-polig) erreicht werden. Der Spannungspegel der als Eingang beziehungsweise Ausgang geschalteten Pins beträgt 5V.

Ethernet → 10/100 Mbit/s

USB → 4 LS/FS/HS

RS232 → 1 D-Sub 9-polig (UART3[3])

I2C → 4

4.2. Digitales Speicheroszilloskop

Für die Erfassung elektrischer Signale in Abhängigkeit der Zeit benötigt man ein Oszilloskop. Bei der Bearbeitung der vorliegenden Problemstellung kamen zwei digitale

[2] Die Abkürzung MMU steht für Memory Management Unit. Diese Einheit bildet den virtuellen Adressraum eines Prozesses auf den tatsächlichen Adressraum des Hauptspeichers ab. Es besteht deshalb ein Zugriffsschutz des Speichers. Verschiedene Prozesse können nicht gleichzeitig auf den gleichen Adressraum des Hauptspeichers zugreifen.

[3] Universal Asynchronous Receiver Transmitter zur Realisierung von serieller Kommunikation

Speicheroszilloskope zum Einsatz. Im folgenden sollen wichtige Kenngrößen dieser Speicheroszilloskope vorgestellt werden. Des weiteren wird diskutiert, ob sich die konkreten Geräte eignen, um die spezifischen Signale, welche im Blick auf die Problemstellung interessant erscheinen, zu erfassen.

Die entscheidenden Kenngrößen eines digitalen Speicheroszilloskops ist die **Bandbreite** und die **Abtastrate**.

Bandbreite Die Bandbreite kennzeichnet den Abstand zwischen minimaler und maximaler Frequenz eines angelegten Signals, bei dem die Signaldämpfung 3 dB beträgt. Da bei Oszilloskopen der Übertragungsbereich bei 0 Hz beginnt, entspricht der Wert der Bandbreite der oberen Grenzfrequenz f_{max}. Bei der oberen Grenzfrequenz ergibt sich ein Betrag der Signalamplitude von $\frac{V_{max}}{\sqrt{2}}$.

Abtastrate Die Abtastrate gibt an, wie groß der Abstand zwischen den Abtastzeitpunkten ist. Die Einheit ist Hz oder $\frac{Samples}{s}$.

Zwischen der Signalfrequenz f_s des Signals, welches abgetastet werden soll und der Abtastrate des Oszilloskops f_a muss ein wichtiger Zusammenhang beachtet werden. Diesen Zusammenhang beschreibt das Nyquist-Shannon-Abtasttheorem (siehe 4.2).

$$f_a > 2 \cdot f_s \tag{4.2}$$

Es sagt aus, dass die Abtastfrequenz mindestens doppelt so hoch sein muss wie die Frequenz des abzutastenden Signals, damit kein Informationsverlust bei der Rückgewinnung des Ursprungssignals auftritt. Die Erfüllung dieser wichtigen Bedingung soll nun überprüft werden.

Im Abschnitt *Analyse der zeitlichen Abläufe* wurde ausgeführt, dass ein Vorgang in der Zeitdomäne mit einer Auflösung von 1 ns erfasst werden soll. Daraus lässt sich ableiten, dass eine maximale Frequenz von 0,5 Ghz zu untersuchen ist. (siehe 4.3)

$$f_s = \frac{1}{T} = \frac{1}{2ns} = 0,5Ghz \tag{4.3}$$

Die eingesetzten digitalen Speicheroszilloskope besitzen folgende entscheidenden Leistungsdaten: [18], [19]

Typ	Bandbreite	Abstastrate	Kanäle	Verfügbar bei
Tektronix DPO3034	300 Mhz	2,5 GS/s	4	Fraunhofer IWS Dresden
Tektronix TDS784D	1000 Mhz	4 GS/s	4	TUD Nachrichtentechnik

Für die Analyse bestimmter Anomalien im Signalverlauf schien das **Tektronix TDS784D** geeigneter zu sein. Dabei handelte es sich insbesondere um Signalsprünge, die sporadisch auftraten. Auf diese Problematik wird im Kapitel *Diskussion* näher eingegangen.

Abschließend soll festgehalten werden, dass grundsätzlich beide Oszilloskope, ihren Leistungsgrenzen nach, zur Untersuchung der vorkommenden Signale eingesetzt werden können Das Nyquist-Shannon-Abtasttheorem (siehe 4.2) ist erfüllt.

4.3. Methodik

Dieser Abschnitt beschreibt die grundlegende Idee, die der Herangehensweise zur Beantwortung der gegebenen Fragestellung zu Grunde liegt. Es soll deutlich werden, welcher Messaufbau entwickelt wurde, um die Ergebnisse, die der Ausgangspunkt zur Formulierung der Thesen sind, zu ermitteln. Die Messungen, welche durchgeführt wurden, sollen nachvollziehbar und wiederholbar sein.

Diese Arbeit ist maßgeblich durch die Frage getrieben, ob die bereits vorgestellte Hardware-Plattform mit einer physikalischen Umgebung im Nanosekundenbereich interagieren kann. Dies führt zur der Überlegung, dass man einen entsprechenden Systemaufbau konstruieren muss, mit dem es möglich ist einen Interaktionsprozess nachzubilden.

Es sind dafür wenigstens folgende Komponenten erforderlich:

1. Eine **Nachricht**, die bei dem Empfänger ein bestimmtes Verhalten bewirkt. Eine einfache Form einer Nachricht stellt ein Rechtecksignal dar, welches als ein Träger von Informationen angesehen werden kann.

2. Eine **Schnittstelle**, über welche die Nachricht übermittelt werden kann. Dafür bietet sich die bereits erwähnte GPIO-Schnittstelle an. An ihr kann ein elektrisches Signal, in Form einer Rechteckfunktion, abgegriffen werden.

3. Ein **Akteur**, welcher eine Nachricht generiert. Diese Aufgabe verantwortet das eingebettete System selbst. Die Generierung des Rechtecksignals erfolgt durch ein entsprechendes Programm. Innerhalb des Programms kann die Charakteristik des Rechtecksignals festgelegt werden.

4. Der **Empfänger** der Nachricht, welcher die Nachricht qualitativ und quantitativ abbildet. Das digitale Speicheroszilloskop besitzt alle Voraussetzungen diese Funktion zu erfüllen.

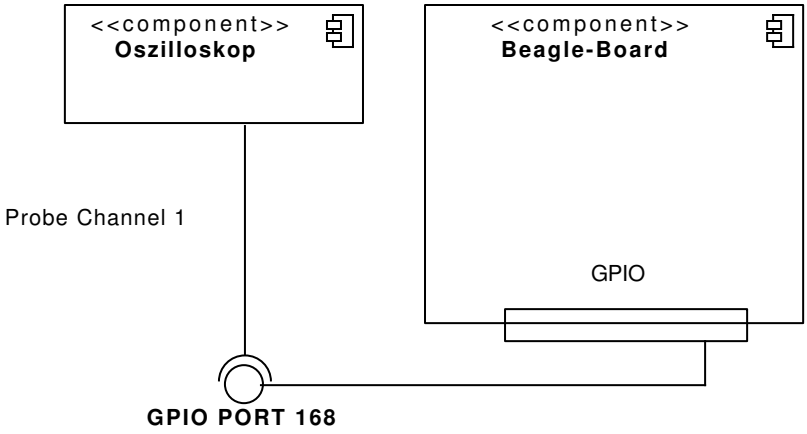


Abbildung 4.2.: Komponentendiagramm

Die aktiven Komponenten können nun miteinander verbunden werden. Abbildung 4.2 zeigt die Realisierung. Ausgehend von diesem Verbund sollte sich ein generiertes Rechtecksignal qualifizieren und quantifizieren lassen.

Daraus lassen sich folgende Erkenntnisse ableiten:

1. Aus der **Impulsbreite** eines High- oder Low-Zustandes lässt sich die mögliche zeitliche **Auflösung** ermitteln.

2. Zeitliche **Abweichungen** (Jitter[4])oder **Anomalien** der Impulsfolge können erkannt werden. Aus diesen Erscheinungen lässt sich unter Umständen schließen, dass das System kein **deterministisches** Verhalten aufweist.

3. Es werden zwei unterschiedliche Methoden angewendet. Die eine Methode führt das Programm, welches das Rechtecksignal generiert, unter einem Betriebssystem aus. Die andere Methode sieht vor, dass dieses Programm ohne Betriebssystem abgearbeitet wird. Daraus lässt sich ein Vergleich zwischen den Methoden gewinnen. Darüber hinaus sollte sich der Einfluss von **Latenzen** und **Interrupts** des Betriebssystems auf das Rechtecksignal einschätzen lassen.

[4] „Jitter is defined as the short-term variations of a digital signal's significant instants from their ideal positions in time. This captures the essence of jitter, but some of the individual terms (short-term, significant instants, ideal positions) need to be more specific before this definition can be unambiguously used. In all real applications, jitter has a random component, so it must be specified using statistical terms. Metrics such as mean value and standard deviation, and qualifiers such as confidence interval, must be used to establish meaningful, repeatable measurements.“[20, Seite 4]

5. Systementwicklung

Im vorherigen Kapitel wurden die Rahmenbedingungen für die Methoden, die zur Beantwortung der gegebenen Problemstellung führen sollen, ausformuliert. Dieses Kapitel beschreibt die konkreten Schritte, die diese Rahmenbedingungen realisieren. Es geht letztlich um die Entwicklung von zwei Programmen, die ein Rechtecksignal generieren. Ein Programm wird in die Laufzeitumgebung eines Betriebssystems eingebettet sein und kann deshalb auf vorhandene Abstraktionsebenen aufsetzen. Essenzielle Initialisierungen werden durch das Betriebssystem realisiert.

Das andere Programm soll ohne Betriebssystem ausgeführt werden. Bevor das Programm auf der Hardware-Plattform lauffähig wird, sind einige zusätzliche Vorüberlegungen erforderlich.

5.1. Ansteuerung der GPIO-Schnittstelle

Die GPIO-Schnittstelle ist in sechs Module aufgeteilt. Jedes Modul besitzt 32 zugehörige GPIO-Pins, die prinzipiell als Ausgang oder Eingang geschaltet werden können. Es werden folgende Anwendungsfälle unterstützt: [15, Seite 3477]

- Dateneingang / Datenausgang
- Schnittstelle zur Tastaturansteuerung mit Entprellung
- Auslösung eines Interrupts bei externen Ereignissen
- Anfrage zum Aufwachen (Wake-up request) bei der Detektion von externen Ereignissen

Pro GPIO-Modul sind zwei Anschlüsse für Interrupts vorhanden.

Das Blockdiagramm 5.1 gibt einen Überblick über den SOC (System on a Chip) DM3730 und zeigt, wie sich die GPIO-Module in das Gesamtsystem einordnen. Ein Aspekt, der auch für spätere Erklärungen nicht unerheblich ist, offenbart sich in seiner Struktur. Die GPIO-Module befinden sich, von der Mikroprozessoreinheit aus gesehen, in einer höheren Schicht des Gesamtsystems. Die Verbindung zwischen diesen Einheiten erfolgt

über zwei Bussysteme, welche als **L3 Interconnect** und **L4 Interconnect** bezeichnet sind. Eine Recherche offenbarte, dass Texas Instruments eine Lösung der Firma Sonics zur Realisierung des Bussystems einsetzt. Das Open Core Protocol (OCP) beschreibt die Datenübertragung. [21] Die maximale Datenübertragungsrate wird mit 300 Mhz angegeben. Eine Vertiefung in das Protokoll erfolgt an dieser Stelle nicht. Es soll aber festgehalten werden, dass es innerhalb des SOC eine Kommunikationsschicht gibt, die beim Ansprechen von Einheiten der Peripherie eine maßgebliche Bedeutung hat.

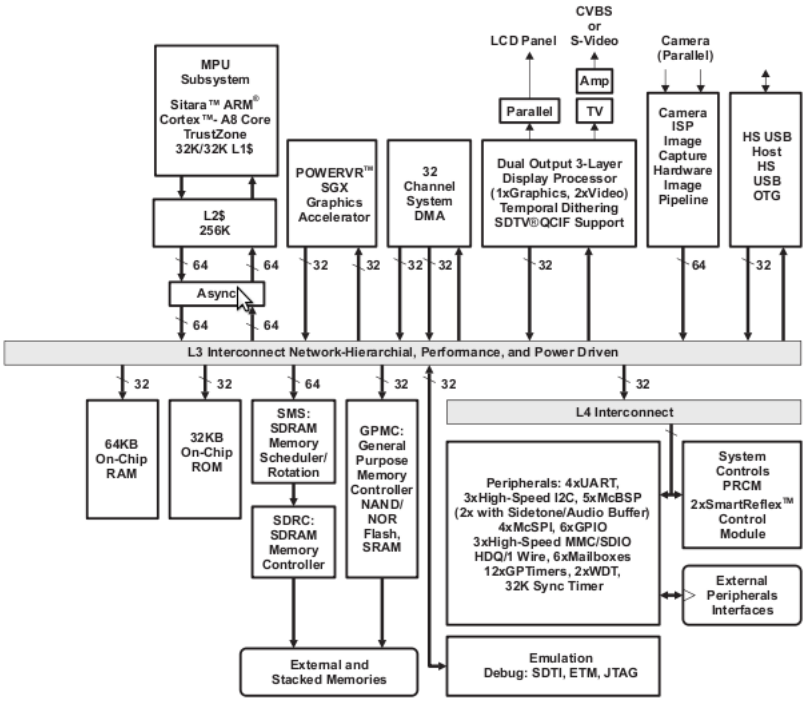


Abbildung 5.1.: Blockdiagramm

Die nächste folgerichtige Aktion ist die **Konfiguration** der GPIO-Schnittstelle. Dafür sind einige Schritte erforderlich, die exemplarisch sind, wenn ein Port der GPIO-Schnittstelle als Ausgang geschaltet werden soll:

1. Die **Auswahl** eines geeigneten GPIO-Pins muss getroffen werden. Ein elektrischer Zugangspunkt sollte vorhanden sein.

2. Die meisten GPIO-Pins haben eine multiple Funktionalität. Deshalb ist es vor der Benutzung erforderlich, die entsprechende **Funktion** zu bestimmen. Die Realisierung erfolgt durch einen Multiplexer. Ein Multiplexer entspricht einer Schaltung zur Selektion von elektrischen Signalen. Einen Multiplexer kann man sich in der Analogie zu einem Drehschalter vorstellen. Durch einen Drehschalter wird aus einer

Folge von Eingangssignalen eine Auswahl getroffen und ein bestimmtes Eingangssignal auf den Ausgang gelegt. Der eigentliche Schaltvorgang wird innerhalb des Mikroprozessors elektronisch umgesetzt.

3. Nachdem die Entscheidung getroffen wurde, dass ein bestimmter GPIO-Pin als Eingang beziehungsweise Ausgang fungieren soll, ist die Festlegung der **Datenrichtung** notwendig.

4. Im letzten Schritt kann nun der **elektrische Pegel** des GPIO-Pins bestimmt werden.

Die einzelnen genannten Schritte werden nun durchgeführt und kommentiert. Die Bearbeitung orientiert sich am Technical Reference Manual [13] und Beagle-Board System Reference Manual [12]:

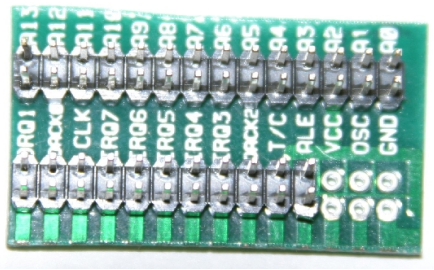


Abbildung 5.2.: Steckerleiste

GPIO-PIN Für den Zugriff auf die Funktionalität der GPIO-Schnittstelle ist der Expansions-Steckplatz prädestiniert. An Pin 24 kann im Betriebsmodus 4 der GPIO-PIN 168 abgegriffen werden [12, 108]. Um einen sicheren mechanischen und elektrischen Kontakt zum Expansion-Steckplatz zu gewährleisten, wurde eine geeignete Steckerleiste (siehe Abbildung 5.2) hergestellt. An Pin 27 oder Pin 28 liegt Massepotential an.

Funktion Das System Control Module (SCM) ist für die Konfiguration der GPIO-Pins verantwortlich. Dieses Modul stellt eine Reihe von 32-Bit Register zur Verfügung, über welche durch Software die Konfiguration gesetzt wird. Jedes Register steuert zwei Pins. [13, Seite 2456] Für die Einstellungen eines Pins stehen deshalb 16 Bit zur Verfügung. Bit 0-3 beziehungsweise 16-18 legen den Betriebsmodus (Muxmode) fest. Die Tabelle 13-4 gibt Aufschluss über das Register *CONTROL_PADCONF_I2C1_SDA[31:16]*, welches mit dem GPIO-Pin 168 korrespondiert. Im Betriebsmodus 4 wird das GPIO-PIN als Eingang beziehungsweise Ausgang definiert. [13, Seite 2467]

Das Register ist über die Adresse $4800|21BC_{16}$ zu erreichen. Über den Befehl *md*

<adress>, welcher durch den Boot-Loader (Das U-Boot [22]) zur Verfügung gestellt wird, kann der Initialwert des Registers ausgelesen werden. Auf die entscheidenden Aufgaben des Boot-Loaders wird in späteren Abschnitten eingegangen. Das Register ist mit dem Initialwert $0000|0118_{16}$ belegt. Daraus folgt, dass nach der Initialisierung der GPIO-Pin 168 in Betriebsmodus 0 betrieben wird. Das entspricht der Datenleitung des I^2C-Busses der Schnittstelle 1. Mittels *mw* $4800|21BC_{16}$ 0004|0118 kann der Betriebsmodus 4 aktiviert werden.

Datenrichtung Die Datenrichtung legt fest, ob ein GPIO-Pin als Eingang oder Ausgang verwendet werden soll. Die Abbildung 25-1 [13, Seite 3495] zeigt, dass die GPIO-Pins 160-186 in Bank 6 organisiert sind. Das Funktions-Register für Bank 6 besitzt eine Basis-Adresse von $4905|8000_{16}$ [13, Seite 3517]. In Tabelle 25-12 [13, Seite 3520] findet man das Datenrichtungsregister *GPIO_OE* unter Adresse $4905|8034_{16}$. Der Wert 0_{16} legt die Datenrichtung eines GPIO-Pins als Ausgang fest. Der Initialwert des Registers beträgt $0000|0000_{16}$. Eine Änderung ist deshalb nicht notwendig.

Pegel Der elektrische Pegel der GPIO-Pins wird durch die Register *GPIO_CLEARDATAOUT* und *GPIO_SETDATAOUT* manipuliert. Damit der Wert, welcher in das Register geschrieben wird, mit GPIO-Pin 168 korrespondiert, ist es entscheidend die richtige Stelle, an der ein Bit auf 1 gesetzt wird, zu bestimmen. Vom niederwertigsten Bit aus gesehen korrespondiert Stelle 8 im Register mit GPIO-Pin 168. Deshalb ist ein entsprechender Wert von $0000|0100_{16}$ in das Register *GPIO_CLEARDATAOUT* einzutragen, um einen logischen Low-Pegel anzulegen. Einen logischen High-Pegel erhält man, wenn das Register *GPIO_SETDATAOUT* mit dem Wert $0000|0100_{16}$ belegt wird.

5.2. Programmentwicklung unter einem Betriebssystem

Die Register und die Werte über welche die GPIO-Schnittstelle angesprochen werden kann, um ein Rechtecksignal auszugeben, sind nun bekannt. Ausgehend von dem Betriebssystem Debian GNU/Linux 6.0 soll nun ein Programm entwickelt werden, welches ein Rechtecksignal generiert.

Die Installation des Betriebssystems wird an dieser Stelle vorausgesetzt. Damit aber alle Schritte, die zur Beantwortung der Fragestellung führten, nachvollzogen werden können, ist eine Installationsanleitung im Anhang A enthalten.

Die erste Überlegung bezieht sich auf den Zugriff auf die Register des Mikroprozessors. Da der Mikroprozessor über eine MMU verfügt, die unter dem Betriebssystem aktiviert ist, ist ein direkter Zugriff auf den physischen Speicherbereich prinzipiell nicht möglich.

Linux stellt aber eine zeichenorientierte Gerätedatei */dev/mem* zur Verfügung, deren Byte-Adressen als physische Speicheradressen interpretiert werden. Als Superuser ist auf

diese Weise ein Zugriff auf den physischen Speicher zu bewerkstelligen.

Quelltext C.1 zeigt die Implementierung der zuvor erläuterten Schritte. Das Programm lässt den logischen Pegel des GPIO-Pins 168 zyklisch alternieren. Die wichtigsten Instruktionen des Programms verdienen eine Erläuterung:

Zeile 11 Es wird ein Eintrag in die Systemtabellen vorgenommen, indem der zurückgegebene **Dateideskriptor** registriert wird. Die Gerätedatei */dev/mem* wird zum Lesen und Schreiben geöffnet. Der Datenfluss erfolgt synchron (*O_SYNC*). Das bedeutet, dass während eines Schreibvorgangs ein anderer Prozess blockiert wird, bis der Schreibvorgang beendet wurde.

Zeile 20 An dieser Stelle erfolgt der entscheidende Schritt. Der Befehl *mmap* blendet die geöffnete Datei in den virtuellen Speicherbereich des rufenden Prozesses ein. Die **Verbindung** zwischen dem Programm und dem **physischen Speicher** ist geschaffen. Es werden Seiten zum Lesen und Schreiben geöffnet. Änderungen an der Datei sind auch für andere Prozesse sichtbar (*MAP_SHARED*). Der 6. Parameter ist wesentlich. Er gibt den Offset an, welcher die **Startadresse** der eingeblendeten Datei kennzeichnet.

Zeile 29 In dieser Zeile wird eine konkrete **Wertzuweisung** durchgeführt. Das Register *CONTROL_PADCONF_I2C1_SDA* wird manipuliert. Die Indexierung auf ein bestimmtes Element des Speicherbereichs geschieht, indem die 4 Byte Adresse des entsprechenden Registers durch 4 dividiert wird. Das Ergebnis zeigt dann auf die richtige Position des Arrays. Durch die Wertzuweisung ist der GPIO-Pin 168 als Eingang beziehungsweise Ausgang konfiguriert.

Zeile 39 Diese Instruktion unterscheidet sich zu Zeile 20 nur durch die **Startadresse** des physischen Speicherbereichs. Es wird der Speicherbereich eingeblendet, der die Register beinhaltet, die für die Pegeländerung des GPIO-Pin 168 verantwortlich sind.

Zeile 49 Das Register *GPIO_CLEARDATAOUT* wird mit $0000|0100_{16}$ belegt. Damit ist vor dem **Schleifeneintritt** für GPIO-Pin 168 ein logischer Low-Pegel definiert.

Zeile 50-59 Nach Eintritt in die Endlosschleife bewirken die Instruktionen in Zeile 52 und Zeile 56 einen **zyklischen Wechsel** des logischen Pegels von GPIO-Pin 168. Durch *sleep* wird eine Entschleunigung des Vorgangs erreicht. Damit konnte bei den ersten Untersuchungen der Vorgang durch ein Digitalmultimeter sichtbar gemacht werden. In späteren Experimenten wurde *sleep* entfernt, um die maximale Grenze der Frequenz des Rechtecksignals bestimmen zu können.

5.3. Programmentwicklung ohne Betriebssystem

Die Rahmenbedingungen für ein Programm, welches ohne Betriebssystem für ein bestimmtes System lauffähig sein soll, unterscheiden sich signifikant zu der Programmentwicklung unter einem Betriebssystem. Prinzipiell gibt es keine Schnittstellen und auch keine Funktionen, die für die Programmentwicklung verwendbar wären. Darüber hinaus sind wichtige Systemkomponenten nicht initialisiert und deshalb nicht einsetzbar.

Deshalb soll zunächst beleuchtet werden, welche minimalen Voraussetzungen geschaffen werden müssen, damit ein Programm ausgeführt werden kann. Da für die Bewältigung der gesamten Aufgabenstellung ein stark begrenztes Zeitkontingent zur Verfügung steht, soll eine Lösung gefunden werden, die zeitnah umgesetzt werden kann.

Betrachtet man den Zeitpunkt nach dem Einschalten des Systems wird erkennbar, dass als erstes Programm der sogenannte ROM-Code ausgeführt wird. Die CPU enthält einen Reset-Vektor, der auf das Programm zeigt. Der ROM-Code ist fester Teil der Systemhardware und kann deshalb nicht verändert werden. Im nächsten Schritt gibt es verschiedene Optionen. Der ROM-Code ist in der Lage selbstständig zu erkennen, über welchen Weg weitere Programme nachgeladen werden sollen. Bei dem vorliegenden System kam eine MMC (Multimedia Card) zur Anwendung. Findet der ROM-Code eine MMC, durchsucht er die Partitionstabelle und transferiert den X-Loader, welcher sich auf der ersten FAT32-Partition befinden muss, in den Hauptspeicher und führt in aus. [23]

Damit dieser Prozess überhaupt möglich wird, müssen grundsätzliche Initialisierungen des Systems vorgenommen werden. Der ROM-Code initialisiert den Systemtakt, den Peripherietakt, den Speicher und bestimmte Einheiten der Peripherie. Dazu gehört beispielsweise der MMC-Controller und die serielle Schnittstelle. Die serielle Schnittstelle ist deshalb ein geeigneter Zugriffspunkt, um schon während der ersten Schritte der Bootsequenz, mit dem System zu kommunizieren. Die serielle Schnittstelle ist initial konfiguriert für 115200 bit/s, 8 Bit Datenlänge, 1 Stopbit und keine Parität. [23]

Der X-Loader (First Stage Loader) initialisiert die Multiplexer, die für die Auswahl der Funktionalität der Pins entscheidend sind. Des weiteren lädt er das U-Boot (Second Stage Loader) in den Hauptspeicher. Die zentrale Aufgabe des U-Boots ist es, das Kernel-Image des Betriebssystems zu entpacken und in den Hauptspeicher zu laden. Auf diese Funktionalität soll aber genau an dieser Stelle nicht zurückgegriffen werden. Das U-Boot bietet auch eine Benutzerschnittstelle an, die es erlaubt das System zu manipulieren und Programme über die serielle Schnittstelle oder Ethernet zu übertragen und auszuführen. Essenzielle Befehle dieser Benutzerschnittstelle können dem Anhang D entnommen werden.

Somit ist das U-Boot eine aussichtsreicher Ausgangspunkt, um auf dieser Grundlage ein Programm zu entwickeln und es ohne Betriebssystem auszuführen. Für die Installation des X-Loaders und des U-Boots wurde nach der Anleitung vorgegangen, die im Anhang

A zu finden ist.

Für die Entwicklung des eigentlichen Programms, welches als Binärcode in den Hauptspeicher übertragen wird, sind einige Vorbereitungen und Schritte nötig, welche in den nächsten Abschnitten ausgeführt werden.

5.3.1. Crosscompiler / Toolchain

Der Crosscompiler ist erforderlich, weil das zu entwickelnde Programm auf einer Plattform kompiliert wird, deren Architektur von der Zielarchitektur verschieden ist. Die Toolchain enthält essenzielle Werkzeuge, die für das Bauen des Objektes unentbehrlich sind. Dazu gehören der Assembler, der Linker und Programme zum Ausgeben von Objektinformationen (*objdump*) und Transformieren eines Objektes in verschiedene Formate (*objcopy*).

Zum Einsatz kommt die Toolchain Yagarto [24]. Die Installation der Toolchain ist trivial. Es müssen lediglich symbolische Links in */usr/bin* angelegt werden, die auf die erforderlichen Werkzeuge im Installationspfad von Yagarto zeigen.

5.3.2. Linker-Script

Im Linker-Script werden die Speicherbereiche der Zielplattform definiert. Des weiteren wird spezifiziert, wie die Sektionen der Eingabedateien auf den Speicher der Zielplattform abgebildet werden. [25] Damit das Programm auf der Zielplattform ausgeführt werden kann, ist die Position des Programm im Speicher entscheidend. Abbildung 5.3 gibt Aufschluss über die Aufteilung der unterschiedlichen Speicherbereiche der Zielplattform.

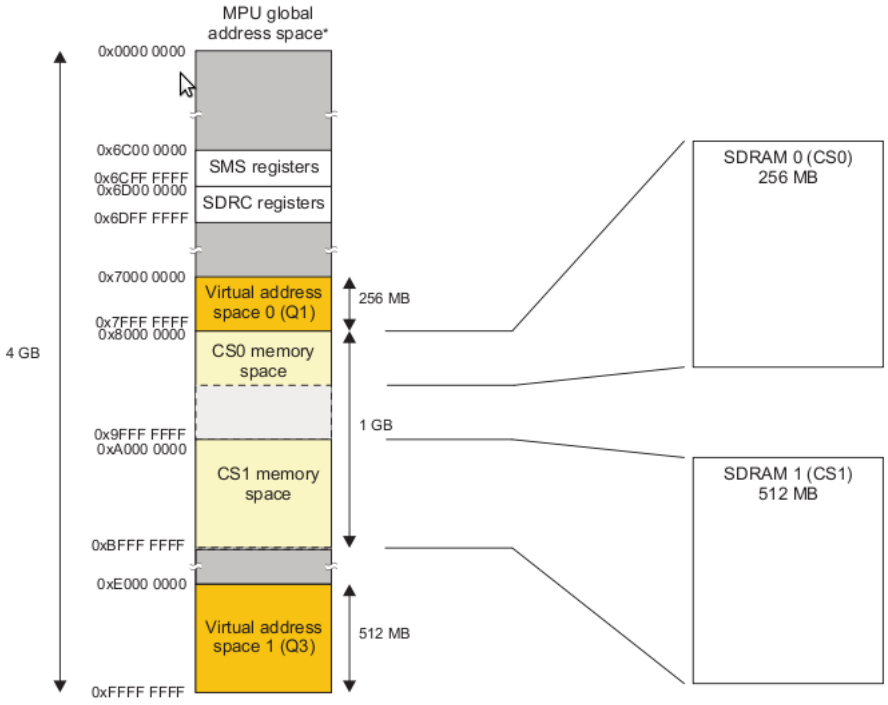


Abbildung 5.3.: Speicherkarte

Aus ihr geht hervor, dass der SDRAM bei der Adresse $8000|0000_{16}$ beginnt und eine Länge von 256 MByte besitzt. Die Umgebungsvariable *loadaddr* des U-Boots legt die Zieladresse im SDRAM fest. Ein serieller Transfer eines binären Abbildes von einem Programm lädt es zur Position dieser Zieladresse. Mittels *go <Zieladresse>* wird das Programm ausgeführt. Die Konfiguration des U-Boots legt die Umgebungsvariable *loadaddr* auf einen Wert von $8020|0000_{16}$ fest.

Ausgehend von diesen Vorbedingungen werden nun die Instruktionen des Linker-Scripts C.2 verständlich:

Zeile 5 An dieser Stelle erfolgt die Definition des Zielspeicherbereichs. *MEMORY* beschreibt die Lokation im Speicher. *ORIGIN* kennzeichnet die Startadresse der Lokation. *LENGTH* gibt die Größe des definierten Speicherbereichs in Byte an. [25] Die Instruktion bewirkt, dass ein Speicherbereich mit der Länge von 64 KByte definiert wird, welcher bei der Adresse $8020|0000_{16}$ beginnt.

Zeile 10 *SECTION* legt fest, wie die Codesektionen der Eingabedateien auf den Speicher abgebildet werden. Die Codesektion, welche den Programmcode enthält, wird durch das Symbol *.text* repräsentiert. Das Wildcardzeichen bewirkt, dass die Anweisung auf alle Programmcodesektionen aller Eingabedateien angewendet wird. Sämtlicher

Programmcode wird dem Speicherfeld, welches in Zeile 5 definiert wurde, über das Symbol *ram* zugeteilt.

5.3.3. Make-File

Das Make-File, welches in Abschnitt C.3 dargestellt ist, enthält die Beschreibung wie das Zielobjekt erstellt werden soll. Die einzelnen Schritte sind auf den ersten Blick nicht selbsterklärend und sollen eine Erläuterung erfahren:

Zeile 1 An dieser Stelle wird das Präfix definiert, welches den Werkzeugnamen vorangestellt wird. Das Präfix dient der Namensunterscheidung zwischen den Werkzeugen des nativen Systems und den Werkzeugen des Crosscompilers.

Zeile 4 Der Assembler transformiert den Quellcode in Maschinencode.

Zeile 5 In dieser Zeile wird der Linker gestartet, welcher aus dem assemblierten Objektfile ein Programm im ELF-Format erstellt.

Zeile 6 Nach der Disassemblierung steht eine Datei *gpio.list* bereit, welche zum Zweck der Fehlersuche oder Optimierung verwendet werden kann.

Zeile 7 Hier wird das Objektfile in ELF-Format gekürzt, indem alle Informationen, außer der Programmsektion, entfernt werden. Die entstehende Datei liegt im BIN-Format und kann für die Übertragung zum Zielsystem benutzt werden.

5.3.4. Assembler

Der Assemblercode enthält die eigentliche Logik des Programms. Die Funktion des Programms wurde bereits in vorangegangen Abschnitten beschrieben und unterscheidet sich nur unwesentlich zum Programm C.1.

Im folgenden sollen die wichtigsten Instruktion in gpio.asm kommentiert werden:

Zeile 3 Hier wird durch *.equ* ein Ausdruck einem symbolischen Namen gleichgesetzt. Die Gleichsetzung bezieht sich auf die Basis-Adresse des UART3 der Zielplattform. Der UART3 realisiert die RS232-Schnittstelle, welche durch einen Port von außen zu erreichen ist. Die Adresse $4902|0000_{16}$ ist dem UART3 zugeordnet. [13, Tabelle 19-38, Seite 2950]

Zeile 6-8 Die Zeilen bewirken, dass über die serielle Schnittstelle das Zeichen „H“ ausgegeben wird. Die Ausgabe signalisiert, dass ein Programm sich in Ausführung befindet.

Zeile 10-12 Der Codeabschnitt bewirkt, dass GPIO-Pin 168 als Eingang beziehungsweise Ausgang geschalten wird, indem das entsprechende Register des zugehörigen Multiplexers manipuliert wird. Die Adressierung erfolgt direkt und ohne Offset. [26] In Zeile 10 wird der Wert der Adresse in Register 8 geladen. Der eigentliche Wert, welcher gespeichert werden soll, lädt die Instruktion in Zeile 11 in Register 1. Durch *str* in Zeile 12 wird der Inhalt von Register 1 auf die Adresse gespeichert, welche sich in Register 8 befindet.

16-18 Durch das Aktivieren von *GPIO_CLEARDATAOUT* wird der elektrische Pegel von GPIO-Pin 168 auf ein logisches Low gesetzt. Die Prozedur zum Beschreiben des entsprechenden Registers ist analog der Kommentierung der Zeilen 16-18.

27-29 Hier wird das Register *GPIO_SETDATAOUT* verändert. Die Änderung bewirkt ein logisches High am GPIO-Pin 168.

31-36 Diese Zeilen entsprechen in etwa einer *for*-Schleife und bewirken eine Zeitverzögerung zwischen den Pegeländerungen.

6. Ergebnisse

Der vorliegende Abschnitt dient der Präsentation der Ergebnisse. Auf Basis der Entwicklungen der vorangegangenen Abschnitte ist es möglich konkrete Werte zu ermitteln. Eine Bewertung der Ergebnisse erfolgt unter dem Abschnitt *Diskussion*.

Im folgenden werden die Bedingungen, die zu den Ergebnissen führten, im einzelnen beschrieben. Richtungweisende Darstellungen des Bildschirmes des digitalen Speicheroszilloskops findet man im Anhang. Nach den ersten Experimenten zeigt sich sehr schnell, dass die Konfiguration des Taktmanagers einen maßgeblichen Einfluss auf die Impulszeiten des Rechtecksignal hat. Deshalb wurde die Konfiguration des Taktmanagers in kleinen Schritten verändert und die Auswirkungen untersucht. Die Register zur Konfiguration des Taktmanagers werden später erläutert.

Es werden nur die Umgebungsparameter explizit erwähnt, die sich zum vorherigen Szenario geändert haben. Die Einstellungen des digitalen Speicheroszilloskops können der Darstellung des Bildschirmes entnommen werden. Die Nachleuchtzeit des Tektronix TDS784D wurde auf 1 Sekunde eingestellt um auch sporadisch auftretende Anomalien erkennen zu können. Beim Verblassen tritt ein Farbwechsel (rot-gelb-grün) ein.

6.1. Generierung eines Rechtecksignals mit Betriebssystem

6.1.1. Szenario 1

Szenario	→ Ermittlung der Impulslänge
Betriebssystem	→ Debian GNU/Linux 6.0
DSO	→ Tektronix DPO3034
Bildschirm Oszilloskop	→ B.1
	Registerwerte Taktmanager
CM_CLKSEL1_PLL_MPU	→ 0013|$200C_{16}$
CM_CLKSEL2_PLL_MPU	→ 0000|0001_{16}
CM_CLKSEL1_PLL	→ 0990|$0C00_{16}$
CM_CLKSEL_CORE	→ 0000|$130A_{16}$
	Rechtecksignal
Impulslänge	→ 200 ns
Periodendauer	→ 400 ns
Frequenz	→ 2,5 MHz
Tastgrad	→ 50 %
	Bemerkungen
Anomalien	→ keine
Auffälligkeiten	→ Einschwingvorgang von ca. 40 ns nach Flankenwechsel

6.1.2. Szenario 2

Szenario	→ Darstellung von Interrupts
Bildschirm Oszilloskop	→ B.2, B.3
	Bemerkungen
Auffälligkeiten	→ Störung des Signals für bis zu 18 ms
	→ Lastsimulation bei B.3
	→ keine Last bei B.2
	→ bei Last sind Störungen häufiger und länger

6.1.3. Szenario 3

Szenario	→ Test auf Anomalien
DSO	→ Tektronix TDS784D
Bildschirm Oszilloskop	→ B.8
	Bemerkungen
Anomalien	→ Signalsprung (dauerhaft): Δ=10 ns B.8
	→ Impulsverlängerungen um bis zu 80 ns (sporadisch) B.8

6.2. Generierung eines Rechtecksignals ohne Betriebssystem

6.2.1. Szenario 4

Szenario	→ Ermittlung der Impulslänge
U-Boot Version	→ 2010.03-dirty (Mar 11 2011 – 08:40:41)
X-Loader Version	→ 1.4.4ss (Mar 10 2011 – 11:47:05)
	Registerwerte Taktmanager
CM_CLKSEL1_PLL_MPU	→ $0012|580C_{16}$
CM_CLKSEL2_PLL_MPU	→ $0000|0001_{16}$
CM_CLKSEL1_PLL	→ $094C|0C00_{16}$
CM_CLKSEL_CORE	→ $0000|130A_{16}$
	Rechtecksignal
Impulslänge	→ 400 ns
Periodendauer	→ 800 ns
Frequenz	→ 1,25 MHz
Tastgrad	→ 50 %
	Bemerkungen
Anomalien	→ keine
Auffälligkeiten	→ Einschwingvorgang (40 ns) nach Flankenwechsel

6.2.2. Szenario 5

Szenario	→ Ermittlung der Impulslänge
U-Boot Version	→ SPL 2012.07-00005-ge63776f (Aug 16 2012 – 07:18:18)
X-Loader Version	→ unbekannt
	Registerwerte Taktmanager
CM_CLKSEL1_PLL_MPU	→ $0012\|580C_{16}$
CM_CLKSEL2_PLL_MPU	→ $0000\|0001_{16}$
CM_CLKSEL1_PLL	→ $0990\|0C00_{16}$
CM_CLKSEL_CORE	→ $0000\|130A_{16}$
	Rechtecksignal
Impulslänge	→ 200 ns
Periodendauer	→ 400 ns
Frequenz	→ 2,5 MHz
Tastgrad	→ 50 %
	Bemerkungen
Anomalien	→ keine
Auffälligkeiten	→ Einschwingvorgang (40 ns) nach Flankenwechsel

---

6.2.3. Szenario 6

Szenario	→ Kontrolle von Störungen
Bildschirm Oszilloskop	→ B.4
	Bemerkungen
Anomalien	→ keine
Auffälligkeiten	→ Einschwingvorgang (40 ns) nach Flankenwechsel → das Rechtecksignal ist stabil, keine Störungen

---

6.2.4. Szenario 7

Szenario	→ Test auf Anomalien
DSO	→ Tektronix TDS784D
Bildschirm Oszilloskop	→ B.5,B.6, B.7
	Registerwerte Taktmanager
CM_CLKSEL1_PLL_MPU	→ $000B\|E80C_{16}$
CM_CLKSEL2_PLL_MPU	→ $0000\|0001_{16}$
CM_CLKSEL1_PLL	→ $0990\|0C00_{16}$
CM_CLKSEL_CORE	→ $0000\|130A_{16}$
	Rechtecksignal
Impulslänge	→ 160 ns
Periodendauer	→ 320 ns
Frequenz	→ 3,125 MHz
Tastgrad	→ 50 %
	Bemerkungen
Anomalien	→ Signalsprünge Δ=10 ns B.6
	→ Signalverschiebung wächstB.7
	→ nach zweiter Periode: Δ=50 ns B.5

7. Diskussion

Die Beobachtungen und Ergebnisse des vorangegangenen Kapitels sollen nun bewertet und eingeordnet werden. Die Bewertung erfolgt unter dem Blickwinkel, inwiefern ein bestimmtes Verhalten erwartungsgemäß auftrat, oder ob es von der Theorie abweicht.

Aufgrund des begrenzten Zeitkontingents können die Ursachen bestimmter Anomalien des Signals nicht vollständig erklärt werden. Das liegt auch nicht zuletzt daran, dass für eine tiefgründige Ursachenforschung detailliertes Wissen über die Architektur des Mikrocontrollers zwingend ist. Der Hersteller des vorliegenden Mikrocontrollers bietet aber nur begrenzt Einblicke in die Systemarchitektur. Blockschaltbilder geben eine Übersicht über die Struktur und die Vernetzung der verschiedenen Systemkomponenten. Daraus wachsen Erkenntnisse über das Zusammenwirken der Systemkomponenten. Oftmals ist es aber nicht möglich nachzuvollziehen, wie eine bestimmte Systemkomponente im Inneren aufgebaut ist und welche elektrische und zeitliche Charakteristik sie aufweist.

Zur Beantwortung der Fragestellung dieser Arbeit ist es auch nicht zwingend notwendig, die Ursache von unerwarteten Beobachtungen bis ins Detail zu erforschen, sondern entscheidend ist, was die Folge dieser Beobachtungen vor dem Hintergrund der Aufgabenstellung ist.

Es folgt die Diskussion einzelner Beobachtungen:

1. Eine offensichtlich unerwartete Beobachtung liegt in dem Vergleich der Impulsdauer zwischen den Rechtecksignalen, die auf der Plattform mit und ohne Betriebssystem ausgeführt werden. Erwartungsgemäß würde man davon ausgehen, dass die Impulsdauer bei dem Rechtecksignal, welches ohne Betriebssystem generiert wird, deutlich kürzer ist, als bei dem Rechteckgenerator, welcher unter einem Betriebssystem ausgeführt wird. Die Begründung dieser Annahme liegt darin, dass es weniger Ursachen für Latenzen des Systems gibt. Beispielsweise werden keine Interrupts ausgelöst und es finden keine Kontextwechsel bei Zustandsänderungen von Prozessen statt. Trotzdem beträgt die Impulsdauer 400 ns (ohne Betriebssystem) im Unterschied zu 200 ns (mit Betriebssystem).

   Diese Diskrepanz führte zu der Vermutung, dass die Taktfrequenz des Prozessors oder des Peripheriebusses die Erklärung für das Verhalten sein könnte. Eine sichere Aussage über die Taktfrequenzen lässt sich nur treffen, wenn man die Konfiguration des Taktmanagers untersucht. Im Abschnitt 7.1 wird die Wertermittlung der Takt-

frequenzen hergeleitet und ihre Auswirkung auf die Signalcharakteristik dargestellt. Tatsächlich lässt sich nachweisen, dass diese Taktfrequenzen signifikanten Einfluss auf das Rechtecksignal haben. Die erwähnte Diskrepanz lässt sich so auflösen. Das U-Boot mit dem Versionsstand vom 11.03.11 initialisiert den Takt des Peripheriebusses auf 166 Mhz (L3 Interconnect) beziehungsweise 83 Mhz (L4 Interconnect). Bei dem Versionstand des U-Boots vom 16.08.12 stellte sich ein erwartungsgemäßes Verhalten ein. Die Konfiguration des Taktmanagers entspricht der Konfiguration unter dem Betriebssystem.

2. Die nächste Beobachtung bezieht sich auf den Einfluss von Interrupts auf den Signalverlauf. Ausgehend von der Theorie sollte es bei dem Rechteckimpuls, welcher unter einem Betriebssystem erzeugt wird, Signalstörungen geben. Dieser Einfluss lässt sich anhand der Ergebnisse zeigen. In den Bildschirmbildern B.2, B.3 sieht man deutlich, dass es zu Signalunterbrechungen kommt, die bis zu 18 ms anhalten. Der logische Pegel verharrt dann entweder im High- oder Low-Zustand. Simuliert man eine Last, indem ein Programm gestartet wird, welches fortlaufend Werte auf den Bildschirm ausgibt, erhöht sich die Wahrscheinlichkeit, dass die Unterbrechungen auftreten und die Unterbrechungsdauer ist größer (siehe B.3 im Vergleich zu B.2). Dieses Verhalten ist offensichtlich nicht deterministisch.

   Aus Abbildung B.4 kann man schließen, dass bei einem Rechtecksignal, das ohne Betriebssystem generiert wird, keine offensichtlichen Störungen in Form von Unterbrechungen auftreten.

3. Durch Manipulation der Taktfrequenzen ist es möglich, die Impulszeiten des Rechtecksignals zu verkürzen. Eine minimale Impulsdauer von 140 ns konnte durch eine Erhöhung des Prozessortaktes und Peripheriebustaktes erreicht werden. Die zeitliche Granularität mit der das System auf seine umgebende physikalische Umgebung einwirken kann beträgt somit 140 ns.

   Dieser Wert, der experimentell ermittelt wurde, ist analytisch schwer nachzuvollziehen. Theoretisch hätte man eine Granularität im Bereich von wenigen Nanosekunden erwarten können. Es wurde bereits erwähnt, dass der Mikroprozessor eine Pipeline besitzt, die 13 Stufen enthält. Wenn man unterstellt, dass der Prozessor mit einer Frequenz von 1 Ghz getaktet ist und pro Taktzyklus tatsächlich eine Instruktion ausgeführt werden kann, indem durch die Pipeline eine parallele Verarbeitung der Mikrobefehle ausgeführt wird, könnte man kleinere Ausführungszeiten, als die ermittelten, annehmen. Der vorliegende Prozessor weist allerdings einen hohen Komplexitätsgrad auf, so dass eine Vorhersage von Latenzen und Ausführungszeiten nur mit hohem Aufwand zu erreichen ist. Das folgende Zitat stützt diese Behauptung.

   > „The complexity of the processor makes it impossible to guarantee precise timing information with hand calculations. The timing of an instruction

> is often affected by other concurrent instructions, memory system activity, and additional events outside the instruction flow. Describing all possible instruction interactions and all possible events taking place in the processor is beyond the scope of this document. Only a cycle-accurate model of the processor can produce precise timings for a particular instruction sequence.“[28]

In dem hohen Komplexitätsgrad des Prozessors liegt aber nicht nur allein die Begründung für eine aufwändige Vorhersage des Zeitverhaltens bei der Befehlsverarbeitung. Die Befehlsverabeitung ist auch nur ein Aspekt, der bei der Ermittlung von Latenzen bei der Ansteuerung der GPIO-Schnittstelle betrachtet werden muss. Aus der Abbildung 5.1 geht hervor, dass die GPIO-Schnittstelle über einen Peripheriebus mit der MPU verbunden ist. Die Kommunikation wird durch das OCP[1] realisiert. Die Kommunikation über dieses Protokoll ist ein zweiter Aspekt, der bei Betrachtung des Zeitverhaltens der GPIO-Schnittstelle berücksichtigt werden sollte.

Ein dritter Aspekt liegt sicher in der Analyse der elektrischen Charakteristik der GPIO-Schnittstelle. Eine Recherche über die verwendete Elektronik und ihre Spezifikation gestaltet sich schwierig. Es ließ sich nicht ermitteln welche Transistoren beziehungsweise Schaltkreise die GPIO-Pins schalten. Die Schaltfrequenz könnte an dieser Stelle beispielsweise ein Faktor sein, der die Impulszeiten des Reckteckssignals limitiert.

4. Bei der Ermittlung der Impulszeiten zeigten sich auch verschiedene Signalanomalien. Im Bildschirmbild B.6 lässt sich eine Verschiebung des Signals um 10 ns erkennen. Da der Signalverlauf rot eingefärbt ist, kann man davon ausgehen, dass dies kein sporadischer Effekt ist, sondern es ist anzunehmen, dass die Ursache systematisch ist. Möglicherweise ist dieser Effekt als Jitter zu bezeichnen. Die Definition von Jitter beschreibt diesen Effekt als eine zeitliche Verschiebung eines Signal, ausgehend von seiner idealen Position. Bei der Datenübertragung kann Jitter Bitfehler verursachen. Auffällig ist, dass es keine zufällige gleichmäßige Verteilung der Verschiebungen gibt. Die Anomalie stellt sich als ein Signalsprung dar, der sowohl unter einem Betriebssystem als auch ohne Betriebssystem 10 ns beträgt. Die Ursache dieser systematischen Erscheinung sollte deshalb in den Systemkomponenten des Mikrocontrollers zu finden sein. Eine Vermutung ist, dass fundiertes Wissen über eine PLL Erklärungen über die Ursache liefern könnte. Auf eine Ursachenforschung soll an dieser Stelle aus Zeitgründen verzichtet werden. Es soll festgehalten werden, dass es einen Effekt gibt, den man als Jitter bezeichnen kann. Die Ursache ist höchstwahrscheinlich nicht zufällig sondern systematisch. Der Wert des Jitters beträgt 10 ns.

[1] OCP-IP is dedicated to proliferating a common standard for intellectual property (IP) core interfaces, or sockets, that facilitate „plug and play “ System-on-Chip (SoC) design [29]

Ein zweiter Effekt folgt einem sporadischen Verhalten. Im Bildschirmbild B.8 sind zwei Erscheinungen sichtbar. Der bereits diskutierte Signalsprung von 10 ns ist erkennbar. Zusätzlich gibt es Signalsprünge, die länger als 10 ns sind und seltener auftreten. Die Beurteilung dieser Sprünge ist schwierig. Je länger diese Signalsprünge, ausgehend vom Triggerpunkt sind, desto seltener treten sie auf. Möglicherweise ist dies auch kein neuer Effekt, sondern Teil der bekannten Erscheinung.

Aus den Bildschirmbildern B.5, B.7 kann man schließen, dass ausgehend vom Triggerpunkt, der Jitter immer größer wird. Nach der fünften Periode beträgt er bereits 96 ns. Es gibt ein Element, das die Erscheinung verstärkt. Bei den Experimenten zeigte es sich, dass die Erhöhung der Taktraten Einfluss auf die Erscheinungen hatte und sie offensichtlich intensivierte.

Im Kapitel *Zusammenfassung und Ausblick* soll der Schwerpunkt auf die Folge dieser Erscheinungen gelegt werden. Eine erschöpfende Analyse der Ursachen dieser Erscheinungen bleibt offen.

7.1. Zusammenhang Konfiguration Taktmanager und Signalverlauf

Eine Recherche im Technical Reference Manual [13] und im BeagleBoard-xM Rev C System Reference Manual [12] führt zur Ermittlung der Taktfreuqenzen. Zunächst muss man ein prinzipielles Verständnis über die Erzeugung der Taktsignale erhalten. Ausgehend von einem hoch genauen Taktsignal, welches durch einen Quarz erzeugt wird, kann über eine sogenannte PLL[2] eine Auswahl von neuen Taktquellen generiert werden. Über Multiplizierer und Dividierer innerhalb einer PLL kann die Frequenz der Taktquelle bestimmt werden.

Aus [12, Seite 73] kann man entnehmen, dass am Prozessor eine Referenzfrequenz von 26 Mhz anliegt. Diese Frequenz wird durch zwei geteilt und steht dann als Systemtakt (SYS_CLK) zur Verfügung. [13, Seite 292]. Aus der Gleichung 7.1 lässt sich die Taktfrequenz der MPU[3] ermitteln. [13, Seite 350]

$$C_{MPU} = \frac{(SYS_CLK) \cdot M}{((N+1) \cdot M2} \tag{7.1}$$

Die entsprechenden Faktoren der Multiplizierer und Dividierer sind durch die Registerwerte, welche unter [13, Seite 350] beschrieben sind, festgelegt. Im Abschnitt 5.2 wurde

[2] PLL ist die Abkürzung für Phase-Locked Loop (Phasenregelschleife). Eine Phasenregelschleife synchronisiert die Phase eines erzeugten Signals mit der eines Referenzsignals. Eine PLL kann nur Signale erzeugen, die einem ganzzahligen Vielfachen des Referenzsignals entsprechen.[27]

[3] MPU ist die Abkürzung für Micro Processor Unit.

ausgeführt, wie man unter einem Betriebssystem Zugriff auf den physischen Speicherbereich erhalten kann. Auf diesem beschriebenen Weg lassen sich die Registerwerte hardwarenah auslesen.

Die Berechnung erfolgt exemplarisch für alle Szenarios in Gleichung 7.2. Um die Faktoren zu erhalten müssen die hexadezimalen Registerwerte zunächst in Binärwerte umgerechnet werden. Aus den 32-Bit Werten kann man die Faktoren extrahieren. Die Positionen sind der Quelle [13, Tabelle 3-58, Seite 350] zu entnehmen.

$$C_{MPU} = \frac{\frac{26MHz}{2} \cdot 800}{((12+1) \cdot 1} = 800Mhz \tag{7.2}$$

Die Berechnung der Taktfrequenz des Peripheriebusses ergibt sich aus den Gleichungen 7.3, 7.4, 7.5, 7.6. [13, Seite 350], [13, Tabelle 3-60, Seite 351], [13, Tabelle 3-196, Seite 492], [13, Tabelle 3-151, Seite 473]

$$f_{CLKOUT} = \frac{f_{SYS_CLK} \cdot M}{N+1} = \frac{\frac{26MHz}{2} \cdot 400}{(12+1) \cdot 1} = 400Mhz \tag{7.3}$$

$$f_{CORE_CLK} = \frac{f_{CLKOUT}}{M2} = \frac{400Mhz}{1} = 400Mhz \tag{7.4}$$

$$f_{L3_ICLK} = \frac{f_{CORE_CLK}}{DIV_L3} = \frac{400Mhz}{2} = 200Mhz \tag{7.5}$$

$$f_{L4_ICLK} = \frac{L3_ICLK}{DIV_L4} = \frac{200Mhz}{2} = 100Mhz \tag{7.6}$$

Es darf festgehalten werden, dass unter dem Betriebssystem Debian GNU/Linux 6.0 die MPU mit einer Taktfrequenz von 800 Mhz betrieben wird. Die Taktfrequenz des Peripheriebusses beträgt 200 Mhz (L3 Interconnect) beziehungsweise 100 Mhz (L4 Interconnect).

Über die vorangegangene Herleitung lässt sich nun eine Übersicht erstellen, die den Zusammenhang zwischen den Taktfrequenzen und der Charakteristik des Rechtecksignals deutlich macht.

Bemerkung	MPU (Mhz)	L3 (Mhz)	L4 (Mhz)	ti (ns)	fsig (Mhz)
Debian GNU/Linux 6.0	800	200	100	200	2,5
U-Boot(11.03.11)	600	166	83	400	1,25
U-Boot(16.08.12)	600	200	100	200	2,5
	800	200	100	180	2,8
	1000	200	100	160	3,2
	600	200	200	160	3,2
	1000	200	200	140	3,6
	600	100	50	380	1,3
	100	200	100	560	0,9

8. Alternativen

In der Diskussion im vorherigen Kapitel zeichnete sich bereits ab, dass der Ansatz der vorliegenden Arbeit sich nicht eignet, das Verfahren der Fluoreszenzlebensdauermessung zu applizieren. Dieses Kapitel stellt ansatzweise einen alternativen Weg vor, der eine Möglichkeit sein könnte, die herausgearbeiteten Probleme zu lösen beziehungsweise zu umgehen.

8.1. Feldprogrammierte Gate Arrays (FPGAs)

> „Bei FPGAs handelt es sich um wiederverwendbare programmierbare Siliziumchips. Mittels vorgefertigter Logikblöcke und programmierbarer Routing-Ressourcen können Chips so konfiguriert werden, dass sie bestimmte Funktionen in Hardware implementieren. Im Gegensatz zu ASICs bieten FPGAs hardwaregetaktete Geschwindigkeit und Zuverlässigkeit bei gleichzeitiger Kosteneffizienz. Die wiederprogrammierbaren Siliziumchips sind genauso flexibel wie Software, die auf einem Prozessor ausgeführt wird. Im Unterschied zu Prozessoren bieten FPGAs echte Parallelität, so dass verschiedene Verarbeitungsoperationen nicht auf die gleiche Ressource angewiesen ist.“[34]

Dem Zitat logisch folgend wäre es denkbar, die Funktionalität die für die Fluoreszenzlebensdauermessung erforderlich ist, vollständig in Hardware zu implementieren. Beispielsweise könnte man eine Zählerschaltung realisieren, die mit einer Voreinstellung versehen werden kann und bei einem definierten Zählerstand die erwähnten Steuersignale ausgibt. Da es kein Bussystem und auch keine Befehlsverarbeitung gibt, ist nur die elektrische Charakteristik der eingesetzten Logik-Gatter ein limitierendes Element in Bezug auf das Zeitverhalten. Es ist davon auszugehen, dass ein FPGA ein aussichtsreicher Ansatz ist, um ein System mit hartem Echtzeitverhalten im Sinne der vorgestellten Anwendung zu entwickeln.

9. Zusammenfassung und Ausblick

Die vorliegende Bachelorarbeit beschreibt das wissenschaftliche Arbeiten in einem interdisziplinären Themengebiet. Es wurde erforscht, inwieweit sich handelsübliche Hardwareplattformen, welche eingebettete Systeme realisieren, eignen, um das Verfahren der Fluoreszenzlebensdauermessung aus dem Bereich der Biomedizin, zu applizieren.

Ausgehend von einer Recherche, in medizinischen Datenbanken über das Verfahren wurden die Anforderungen an das System spezifiziert. Dabei kamen Techniken des Requirements Engineering zum Einsatz. Nachfolgend konnte Systemhardware ausgewählt werden, um ein System zu schaffen, welches eine Möglichkeit zur konkreten Wertermittlung darstellte.

Vor dem Hintergrund der Aufgabenstellung wurde eine Methode erarbeitet, die zur Beantwortung der vorliegenden Fragen führte. Durch eine systematische Herangehensweise an die Problemstellung konnte dann die Systemsoftware entwickelt werden, mit welcher es möglich war bestimmte Funktionen aus dem biomedizinischen Verfahren abzubilden.

In der Diskussion wurden die Ergebnisse eingeordnet und bewertet. Unerwartetes Verhalten konnte punktuell erklärt werden. Aufgrund eines begrenzten Zeitkontingents wurden nicht alle Erscheinungen ursächlich erforscht.

Trotzdem wurden erfolgreich alle Fragen der Aufgabenstellung beantwortet und ein richtungweisender Ausblick auf die zukünftigen Entwicklungen im Forschungsprojekt formuliert.

Die folgenden Thesen wurden aus den Ergebnissen der vorliegenden Arbeit abgeleitet und schließen die Bachelorarbeit ab:

These 1 Handelsübliche Rechentechnik, welche in eingebetteten Systemen eingesetzt wird, eignet sich **nicht**, um im Bereich von wenigen Nanosekunden mit einer physikalischen Umgebung zu interagieren. Sie erfüllt die Anforderungen, um den physikalischen Vorgang der Fluoreszenzlebensdauer zu quantifizieren, deshalb **nicht**.

These 2 Das vorliegende System kann eine minimale zeitliche Auflösung von 140 ns erreichen. Diese These sollte man auch für Systeme mit verwandter Rechenarchitektur anwenden können.

These 3 Das Verfahren der Fluoreszenzlebensdauermessung erfordert, dass Zeitschran-

ken eingehalten werden müssen, die im Bereich von 1 ns liegen. Das vorliegende System würde diese Zeitschranken verletzen.

These 4 Ein System, das für die Anwendung der Fluoreszenzlebensdauermessung zum Einsatz kommt, muss hartes Echtzeitverhalten aufweisen. Verletzungen von Zeitschranken führen zu hohen Kosten.

These 5 Das vorliegende eingebettete System mit oder ohne Betriebssystem ist im Sinne von These 4 **nicht** echtzeitfähig.

These 6 Das vorliegende System ohne Betriebssystem kann stabile Rechtecksignale generieren. Es ist so gesehen in seinem Verhalten zu seiner technischen Umgebung deterministisch.

These 7 Das vorliegende System mit dem eingesetzten Betriebssystem Debian GNU/Linux 6.0 ist im Sinne von These 6 **nicht** deterministisch. Es treten Unterbrechungen der Rechtecksignale auf. Diese These kann sehr wahrscheinlich für alle Betriebssysteme mit Linux-Kernel ohne Echtzeitpatch erweitert werden.

These 8 Das vorliegende System mit oder ohne Betriebssystem weist in Bezug auf seine zeitliche Granularität **keine** signifikanten Unterschiede auf.

These 9 Das vorliegende System mit oder ohne Betriebssystem zeigt Signalsprünge, die sich als Jitter im Bereich von 10 ns interpretieren lassen. Es ist aus dieser Sicht nicht deterministisch. Auch aus diesem Grund ist das vorliegende System für die Anwendung der Fluoreszenzlebensdauermessung **nicht** einsetzbar.

These 10 Aufgrund des hohen Komplexitätsgrades ist eine analytische Vorhersage des zeitlichen Verhaltens des vorliegenden Systems ohne herstellerspezifisches Wissen **nicht** möglich.

These 11 Um das Verfahren der Fluoreszenzlebensdauermessung erfolgreich zu applizieren ist ein Einsatz einer FPGA-Plattform ein aussichtsreicher Weg.

10. Eigenständigkeitserklärung

Ich versichere, dass ich die vorgelegte Bachelorarbeit eigenständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen verwendet habe und die den benutzten Quellen entnommenen Passagen als solche kenntlich gemacht habe.

_____

Lutz Krätzer

Dresden, den 1. Oktober 2012

A. Installation des Betriebssystems

Im folgenden sollen die Schritte erläutert werden, die notwendig sind, um Debian auf einem Beagle-Board zu installieren. Dabei wurde sich weitgehend an einer Anleitung, welche unter *http://elinux.org/BeagleBoardDebian* zu erreichen ist, orientiert.

Im ersten Schritt wird ein Script von Robert Nelson mittels *git* heruntergeladen.

```
1  jean@enterprise4:~/Desktop/beagle/install git clone git://github.com/
       RobertCNelson/netinstall.git
2  Cloning into netinstall...
3  remote: Counting objects: 827, done.
4  remote: Compressing objects: 100% (336/336), done.
5  remote: Total 827 (delta 534), reused 778 (delta 485)
6  Receiving objects: 100% (827/827), 135.80 KiB, done.
7  Resolving deltas: 100% (534/534), done.
8  jean@enterprise4:~/Desktop/beagle/install ls
9  netinstall
10 jean@enterprise4:~/Desktop/beagle/install
```

Danach wird das Script aufgerufen. Es bereitet eine μSD-Card für eine Installation von Debian vor.

Es muss ermittelt werden über welche Gerätedatei die eingelegte μSD-Card zu erreichen ist.

```
df -h
```

Aus der Übersicht kann man entnehmen, das die μSD-Card über die Gerätedatei */dev/sdb* angesprochen wird.

```
1  jean@enterprise4:~/Desktop/beagle/install/netinstall df -h
2  Dateisystem            Größe Benut  Verf Ben% Eingehängt auf
3  /dev/sda1               46G  8,4G   36G  20% /
4  none                   1,9G  304K  1,9G   1% /dev
5  none                   1,9G  184K  1,9G   1% /dev/shm
6  none                   1,9G   88K  1,9G   1% /var/run
7  none                   1,9G     0  1,9G   0% /var/lock
8  none                   1,9G     0  1,9G   0% /lib/init/rw
9  /dev/sda2               61G   58G   26M 100% /home
10 /dev/sdb1              7,4G   32K  7,4G   1% /media/3564-3233
11 jean@enterprise4:~/Desktop/beagle/install/netinstall
```

Das Installation kann nun gestartet werden.

```
./mk_mmc.sh –mmc /dev/sdb –uboot beagle –distro squeeze
```

Wenn keine Fehlermeldungen aufgetreten sind, sollte die μSD-Card für die Netzinstallation von Debian vorbereitet sein. Die μSD-Card muss ausgehängt werden.

```
umount /dev/sdb1
```

Die weitere Installation, die konform zu einer Debian-Standardinstallation ist, kann nun durch Einlegen der μSD-Card auf dem Beagle-Board fortgesetzt werden.

Bei der Partitionierung der μSD-Card zeigte es sich, dass die Laufwerke als logische Laufwerke angelegt werden sollten, da sonst das System das Root-Dateisystem nicht finden kann.

Die Partitionstabelle A.1 des Evaluationssystems gestaltet sich dann wie folgt.

Listing A.1: Partitionstabelle

```
1  Disk /dev/mmcblk0: 3951 MB, 3951034368 bytes
2  4 heads, 16 sectors/track, 120576 cylinders
3  Units = cylinders of 64 * 512 = 32768 bytes
4  Sector size (logical/physical): 512 bytes / 512 bytes
5  I/O size (minimum/optimal): 512 bytes / 512 bytes
6  Disk identifier: 0x000e0e54
7
8          Device Boot      Start         End      Blocks   Id  System
9  /dev/mmcblk0p1   *           1        2128       68045    e  W95 FAT16 (LBA
       )
10 Partition 1 does not end on cylinder boundary.
11 /dev/mmcblk0p2            2144      120544     3788801    5  Extended
12 Partition 2 does not end on cylinder boundary.
13 /dev/mmcblk0p5            2145      114080     3581952   83  Linux
14 /dev/mmcblk0p6          114113      120544      205824   82  Linux swap /
       Solaris
```

A.1. Konfiguration und Administration

A.1.1. Netzwerkadapter

Damit Fernwartung auf dem Beagle-Board möglich ist, muss der Netzwerkadapter entsprechend konfiguriert werden. Für einen Direktverbund, bei dem man mit einem Notebook auf das Beagle-Board zugreifen möchte, wurden folgende Einstellungen gewählt.

Adressierung → statisch

Host-IP → 192.168.1.1

Client-IP → 192.168.1.2

Für die Durchführung dieser Einstellungen ist die Editierung der Datei */etc/network/interfaces* wie folgt notwendig.

Listing A.2: Auszug aus /etc/network/interfaces

```
1  # The loopback network interface
2  auto lo
3  iface lo inet loopback
4
5  # The primary network interface
6  #allow-hotplug usb1
7  #iface usb1 inet dhcp
8
9  # Luk, 23.03.11, Umstellung von DHCP auf statische IP-Zuweisung
10
11 allow-hotplug usb1
12 iface usb1 inet static
13  address   192.168.1.1
14  gateway   192.168.1.1
15  netmask   255.255.255.0
16  network   192.168.1.0
17  broadcast 192.168.1.255
```

B. Bildschirmbilder Speicheroszilloskop

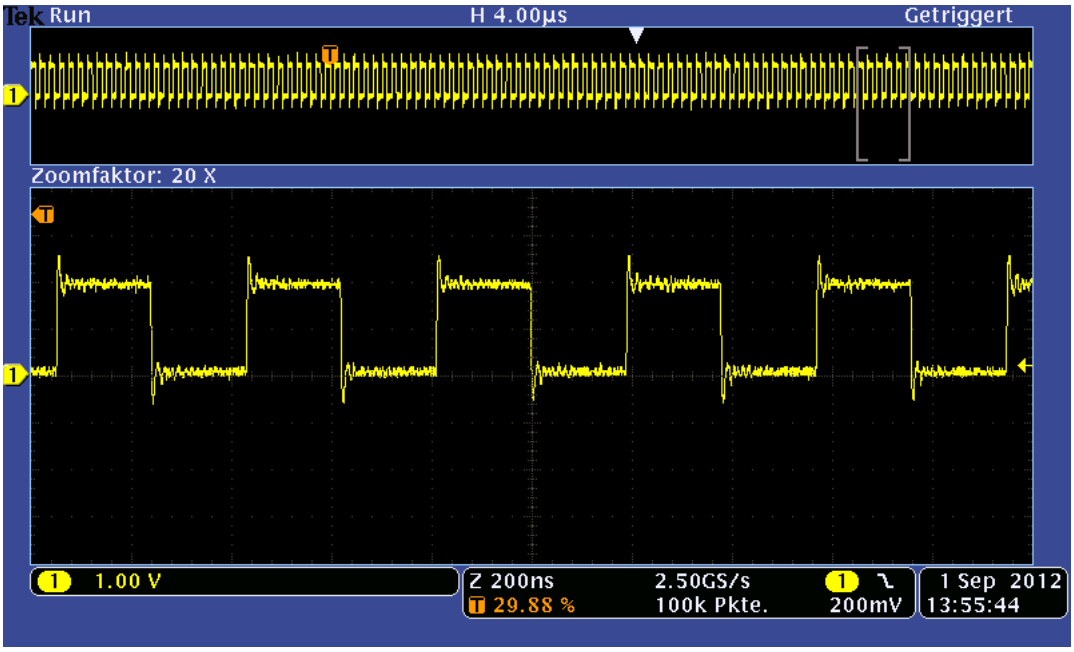


Abbildung B.1.: tek 0

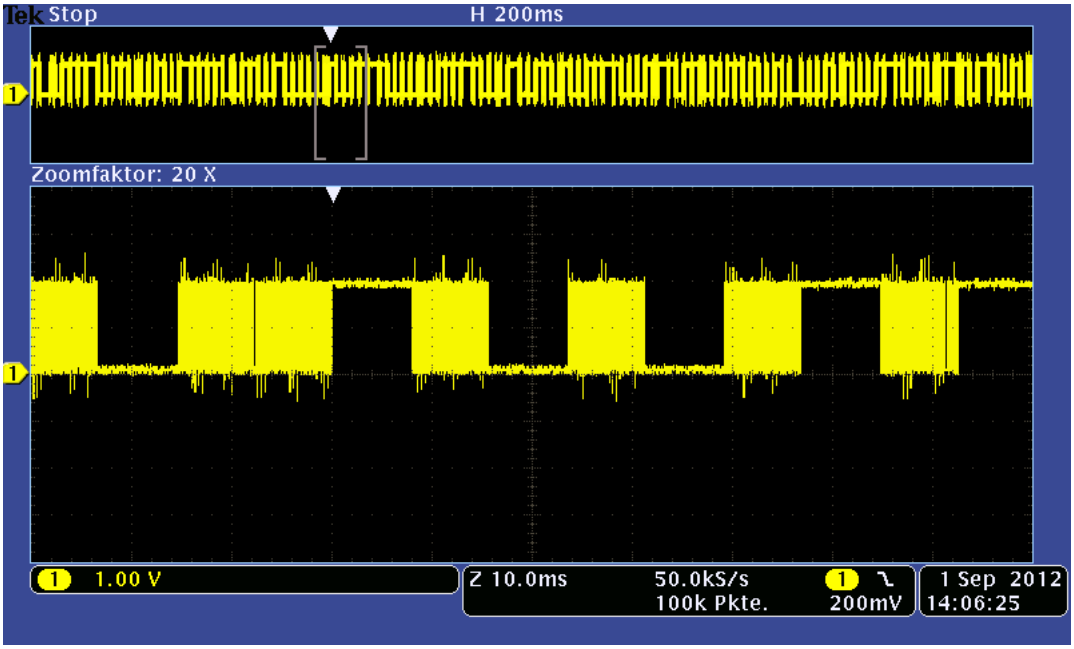


Abbildung B.2.: tek 2

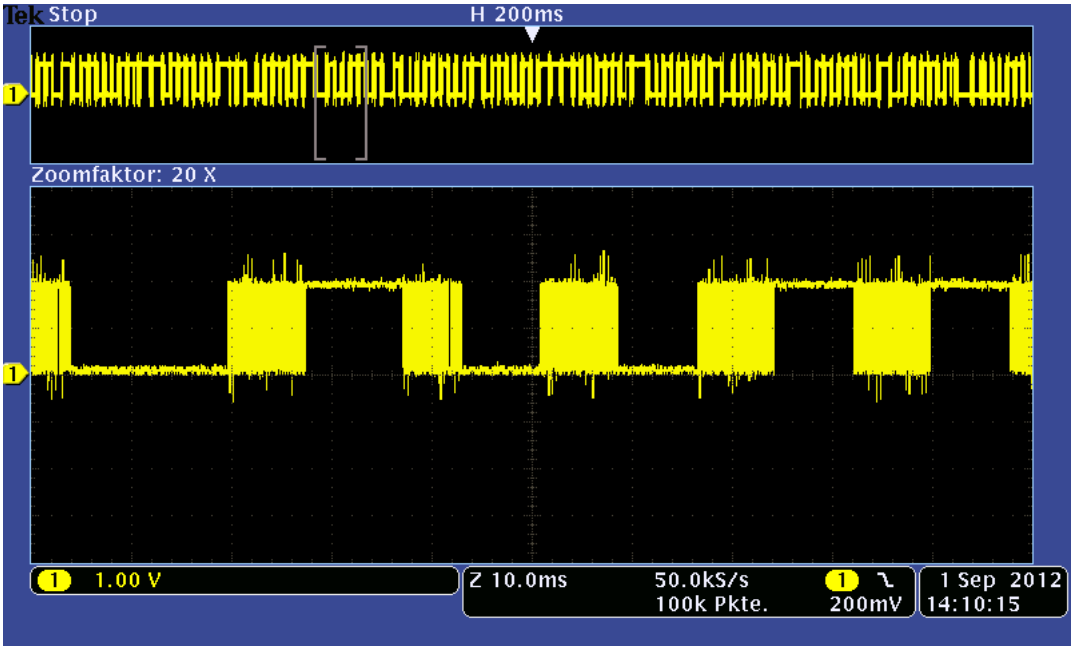


Abbildung B.3.: tek 9

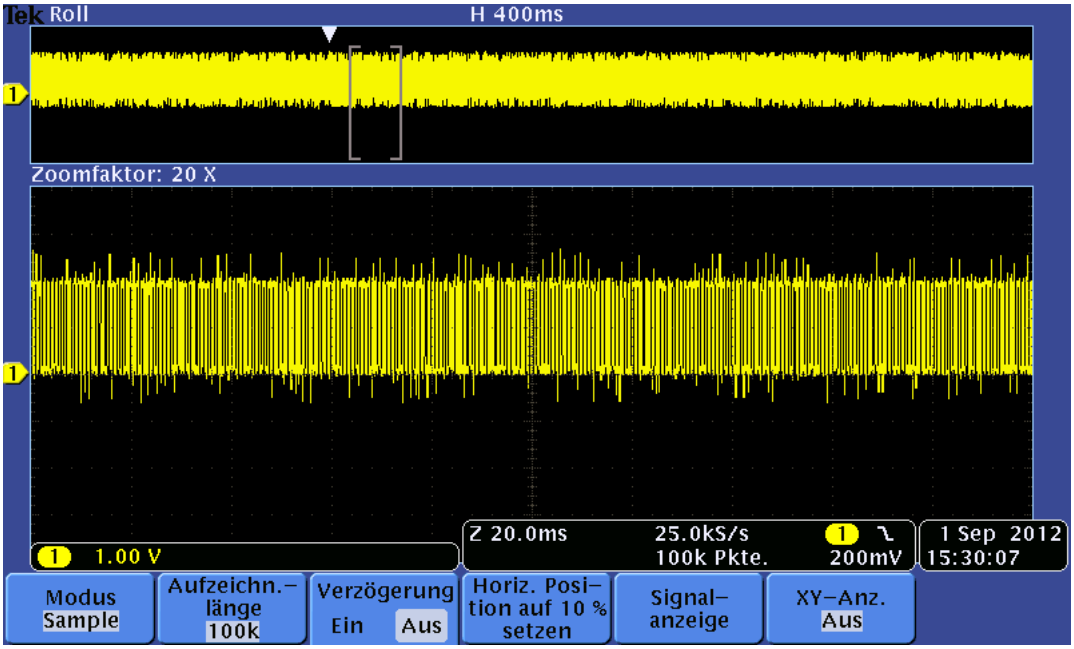


Abbildung B.4.: tek 17

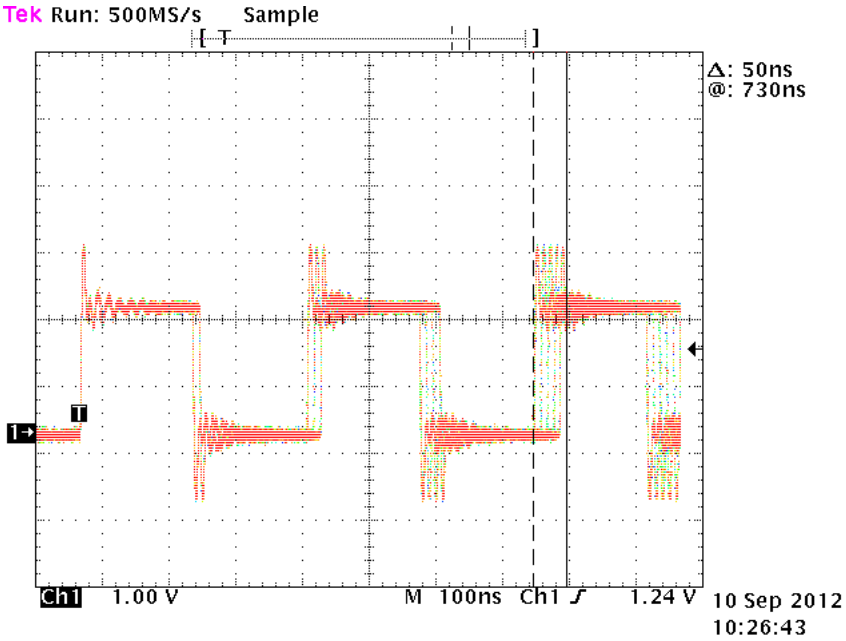


Abbildung B.5.: tek8tds

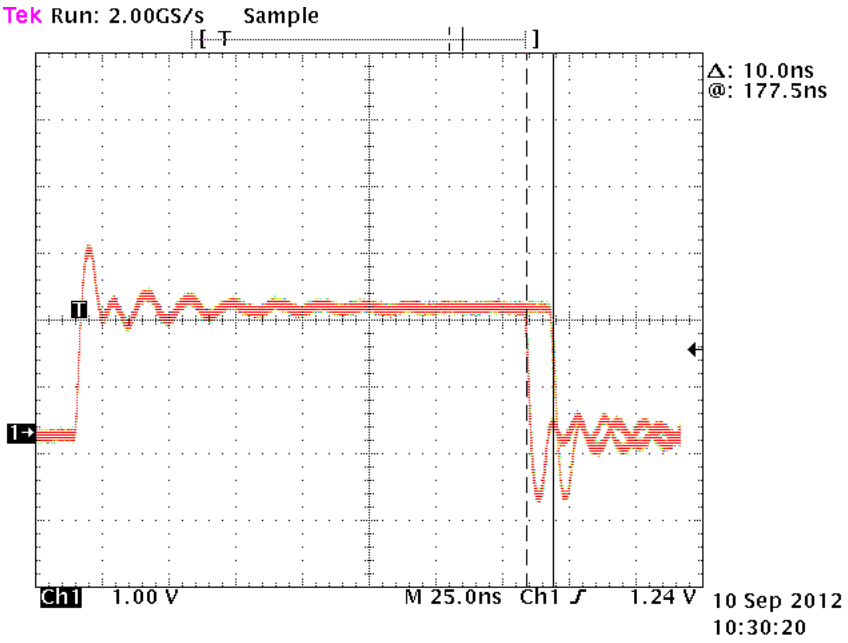


Abbildung B.6.: tek9tds

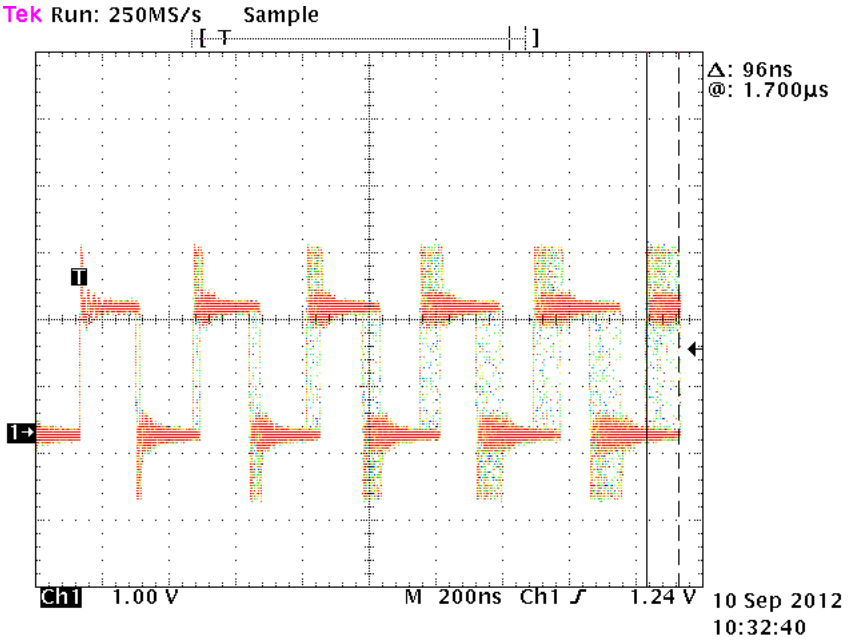


Abbildung B.7.: tek10tds

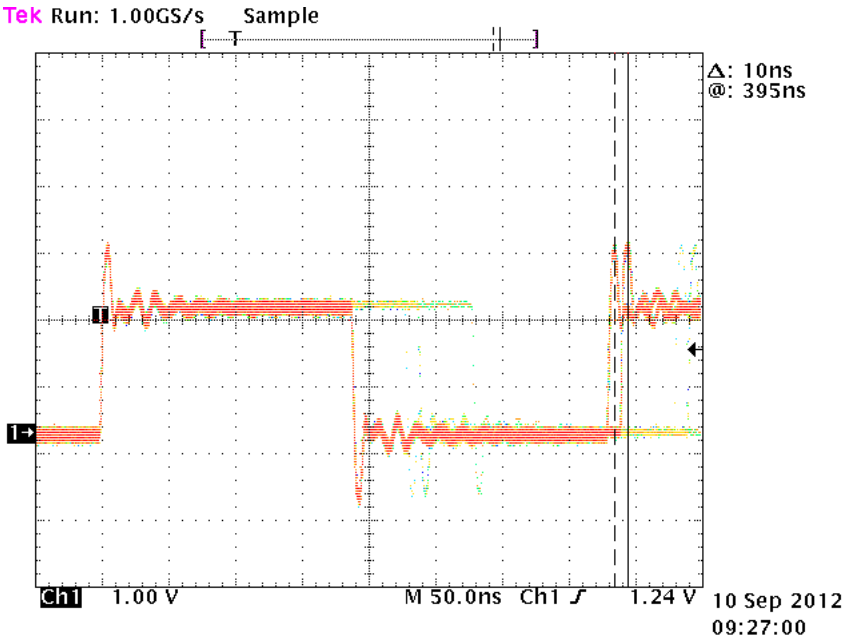


Abbildung B.8.: tek3tds

C. Quelltexte

Listing C.1: gpio.c

```
1
2  /* Lutz Krätzer, 15.08.2012 */
3
4  #include <stdio.h>
5  #include <unistd.h>
6  #include <fcntl.h>
7  #include <sys/mman.h>
8
9  int main()
10 {
11    int fd = open("/dev/mem", O_RDWR | O_SYNC);
12
13    if(fd < 0)
14    {
15       printf("Could not open memory\n");
16       return 0;
17    }
18
19    volatile ulong* pinconf;
20    pinconf = (ulong*) mmap(NULL, 0x10000, PROT_READ | PROT_WRITE,
          MAP_SHARED, fd, 0x48000000);
21
22    if(pinconf == MAP_FAILED)
23    {
24       printf("Pinconf Mapping failed\n");
25       close(fd);
26       return 0;
27    }
28
29    pinconf[0x21BC/4] = 0x00040118;
30
31    close(fd);
32
33   /* ********************************************************** */
34
35
36    fd = open("/dev/mem", O_RDWR | O_SYNC);
37
38    volatile ulong* gpio;
39    gpio = (ulong*) mmap(NULL, 0x10000, PROT_READ | PROT_WRITE, MAP_SHARED,
          fd, 0x49050000);
40
41    if(gpio == MAP_FAILED)
```

```
42      {
43          printf("GPIO Mapping failed\n");
44          close(fd);
45          return 0;
46      }
47
48
49      gpio[0x8090/4] = 0x00000100;
50      while(1)
51      {
52          gpio[0x8094/4] = 0x00000100;
53
54          sleep(1);
55
56          gpio[0x8090/4] = 0x00000100;
57
58          sleep(1);
59      }
60
61      close(fd);
62
63
64      return 0;
65  }
```

Listing C.2: linker.ld

```
1
2   /* rammap */
3   MEMORY
4   {
5       ram : ORIGIN = 0x80200000, LENGTH = 0x10000
6   }
7
8   SECTIONS
9   {
10      .text : { *(.text*) } > ram
11
12  }
```

Listing C.3: Makefile

```
1   ARMGNU = arm
2
3   gpio.bin: gpio.asm
4     $(ARMGNU)-as gpio.asm -o gpio.o
5     $(ARMGNU)-ld -T linker.ld gpio.o -o gpio.elf
6     $(ARMGNU)-objdump -D gpio.elf > gpio.list
7     $(ARMGNU)-objcopy gpio.elf -O binary gpio.bin
```

Listing C.4: gpio.asm

```
1  #UART base locations from the TRM
2  #According to the Beagleboard-xM System Reference Manual, UART3 is
       connected to the serial port.
3  .equ UART3.BASE,      0x49020000
4  .arm
5  _start:
6      ldr r0,=UART3.BASE
7      mov r1,#'H'
8      strb r1,[r0]
9  ############################### set MUX to GPIO
10     ldr   r8,=0x480021BC
11     ldr   r1,=0x00040118
12     str   r1,[r8]
13 ############################### set CLEARDATAOUT
14 forever:
15
16     ldr   r8,=0x49058094
17     ldr   r1,=0x00000100
18     str   r1,[r8]
19 ############################### Delay
20 #mov    r4, #100
21 #next_sec:
22 #  mov    r0, #1000
23 #  bl     delay_ms
24 #  subs   r4, r4, #1
25 #  bne    next_sec
26 ############################### set SETDATAOUT
27     ldr   r8,=0x49058090
28     ldr   r1,=0x00000100
29     str   r1,[r8]
30 ############################### Delay
31 # mov    r4, #100
32 #next_sec2:
33 #  mov    r0, #1000
34 #  bl     delay_ms
35 #  subs   r4, r4, #1
36 #  bne    next_sec2
37
38     b   forever
39
40 delay_ms:
41    ldr    r1, =14750
42 delay_ms_1:
43    subs   r1, r1, #1
44    bne    delay_ms_1
45    subs   r0, r0, #1
46    bne    delay_ms
47    mov    pc, lr
48
49 #exit:   swi 0x11
```

D. U-Boot - Essenzielle Befehle

BDINFO Print board info structure

GO Start application at address *addr*
go addr [arg ...]

LOADY Load binaryfile over serial line

MD Memory display
md addr

MW Memory write
md addr value

PRINTENV Print environment variables

RESET Perform reset of the cpu

SAVEENV Save environment variable to persistent storage

SETENV Set environment variable *setenv name value*

TFTPBOOT Boot image via network using TFTP protocol
tftpboot loadAddress bootfilename

Literaturverzeichnis

[1] Laura, Marcu. (2012). Fluorescence Lifetime Techniques in Medical Applications. *Annals of Biomedical Engineering*, 2012(2), 304-331.

[2] Das Energiekonzept und seine beschleunigte Umsetzung. (2012). Verfügbar unter `http://www.bmu.de/energiewende/beschluesse_und_massnahmen/doc/47892.php` [09.09.2012].

[3] Nationale Roadmap Embedded Systems. (2009). Verfügbar unter `http://www.bitkom.org/60376.aspx?url=NRMES_2009_einseitig.pdf` [09.09.2012].

[4] CHARGE-COUPLED DEVICES. (2002). Verfügbar unter `www.ece.iit.edu/~pfelber/ccd/project.pdf` [11.09.2012].

[5] Korff, Andreas. (2005). Embedded/Real-time UML. In Peter Liggesmeyer & Dieter Rombach (Hrsg.), Software Engineering eingebetteter Systeme (S. 227-247). München: Elsevier GmbH.

[6] Schmusch, Wolfgang. (2001). Elektronische Meßtechnik Band 6. Würzburg: Vogel.

[7] GPIO-Ports-Multitalente der I/O-Kommunikation. (2007). Verfügbar unter `http://www.elektroniknet.de/bauelemente/technik-know-how/halbleiter/article/340/0/` [15.09.2012].

[8] Stankovic, John A. & Krithi Ramamritham. (1988). Hard Real-Time Systems. Los Alamitos: IEEE Computer Society Press.

[9] Reisinger, Johannes. (1993). Konzeption und Analyse eines zeitgesteuerten Betriebssystems für Echtzeitanwendungen. Verfügbar unter `http://www.vmars.tuwien.ac.at/documents/intern/337/Dissertation_Reisinger_Johannes.pdf` [16.09.2012].

[10] Flik, Thomas. (2005). Mikroprozessortechnik und Rechnerstrukturen. Heidelberg: Springer.

[11] Herrmann, Paul. (2011). Rechnerarchitektur. Wiesbaden: Vieweg+Teubner.

[12] BeagleBoard-xM Rev C System Reference Manual. (2010). Verfügbar unter http://beagleboard.org/static/BBxMSRM_latest.pdf [17.09.2012].

[13] DM3730, DM3725 Digital Media Processors. (2011). Verfügbar unter http://www.ti.com/product/dm3730 [17.09.12].

[14] AM/DM37x Overview. Verfügbar unter http://processors.wiki.ti.com/index.php/AM/DM37x_Overview [17.09.11].

[15] AM/DM37x Multimedia Device Silicon Revision 1.x Technical Reference Manual. (2012). Texas Instruments. Verfügbar unter http://www.ti.com/litv/pdf/sprugn4q [17.09.2012].

[16] ARM-Cortex-A8 Technical Reference Manual. (2010). ARM Limited. Verfügbar unter http://infocenter.arm.com/help/topic/com.arm.doc.ddi0344k/DDI0344K_cortex_a8_r3p2_trm.pdf [17.09.12].

[17] Architecture and Implementation of the ARM Cortex-A8 Microprocessor http://www.design-reuse.com/articles/11580/architecture-and-implementation-of-the-arm-cortex-a8-microprocessor.html [25.09.12].

[18] MSO/DPO3000 Mixed Signal Oscilloscope. Verfügbar unter http://www.tek.com/oscilloscope/mso3000-dpo3000 [19.09.2012].

[19] Digital Phosphor Oscilloscope. Verfügbar unter http://www.tek.com/datasheet/digital-phosphor-oscilloscope [19.09.2012].

[20] Understanding and Characterizing Timing Jitter. Verfügbar unter www.to-way.com/tf/tekjit.pdf [19.09.12].

[21] SonicsMX ® SMART InterconnectTM Solution. Verfügbar unter http://www.sonicsinc.com/uploads/pdfs/SMX_121007.pdf [20.09.12].

[22] Das U-Boot-the Universal Boot Loader. Verfügbar unter http://www.denx.de/wiki/U-Boot [23.09.12].

[23] Bootloader Project. Verfügbar unter http://omappedia.org/wiki/Bootloader_Project [24.09.12].

[24] Yet another GNU ARM toolchain. Verfügbar unter http://www.yagarto.de/ [24.09.12].

[25] Sourceware - Linker Scripts. Verfügbar unter http://sourceware.org/binutils/docs/ld/Scripts.html [25.09.12].

[26] ARM Infocenter - memory access instructions. Verfügbar unter `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0068b/Bcfihdhj.html` [25.09.12].

[27] NI Developer Zone. Verfügbar unter `http://www.ni.com/white-paper/3781/de` [26.09.12].

[28] ARM-Infocenter - Instruction Cycle Timing. Verfügbar unter `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344k/Babeghic.html` [27.09.12].

[29] Open Core Protocol. Verfügbar unter `http://www.ocpip.org/` [27.09.12].

[30] Pfeifer, Lutz (2005). Improved Routine Bio-Medical and Bio-Analytical Online Fluorescence Measurements Using Fluorescence Lifetime Resolution. *Journal of Fluorescence*, 2005(3), 423-432. Verfügbar unter `http://www.springerlink.com/content/g91023353x672176/fulltext.pdf` [28.09.12].

[31] Schneckenburger, H. (2005). Laser-Assisted Fluorescence Microscopy and Biomedical Screening. *Laser Physics*, 2006(5), 799-807.

[32] Troy, T.L., Sevick-Muraca, E.M. (1999). Fluorescence Lifetime Imaging and Spectroscopy in Random Media. In W. Rettig, B. Strehmel, S. Schrader, H. Seifert (Hrsg.), Applied Fluorescence in Chemistry, Biology and Medicine (S. 3-38). Berlin, Heidelberg: Springer.

[33] Alberts, Bray, Hopkin (2012). Essential Cell Biology. Weinheim: Wiley-VCH.

[34] Einführung in die FPGA-Technologie. Verfügbar unter `http://www.ni.com/white-paper/6984/de` [28.09.12].

Abbildungsverzeichnis

2.1. NADH Abklingkurve, Journal of Fluorescence 8

3.1. Kontextdiagramm Messsystem, Lutz Krätzer 11
3.5. Zeitanforderungen für Echtzeitservices, Johannes Reisinger 14
3.2. Zustandsdiagramm Messsystem, Lutz Krätzer 17
3.3. Strukturdiagramm Messsystem, Lutz Krätzer 18
3.4. Sequenzdiagramm Messsystem, Lutz Krätzer 19

4.1. Beagle-Board, beagleboard.org 21
4.2. Komponentendiagramm, Lutz Krätzer 25

5.1. Blockdiagramm, Texas Instruments 27
5.2. Steckerleiste, Lutz Krätzer 28
5.3. Speicherkarte, Texas Instruments 33

B.1. tek0, Tektronix DPO3034 54
B.2. tek2, Tektronix DPO3034 55
B.3. tek9, Tektronix DPO3034 55
B.4. tek17, Tektronix DPO3034 56
B.5. tek8tds, Tektronix TDS784D 56
B.6. tek9tds, Tektronix TDS784D 57
B.7. tek10tds, Tektronix TDS784D 57
B.8. tek3tds, Tektronix TDS784D 58