ulm university universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Eingebettete Systeme / Echtzeitsysteme

Kursportfolio zur Vorlesung Architektur Eingebetteter Systeme

Sommersemester 2020

vorgelegt von:
Carolin Schindler
carolin.schindler@uni-ulm.de

verantwortlicher Dozent:
Prof. Dr.-Ing. Frank Slomka

begleitende Dozentent:
Marco Philippi
Marcel Rieß

Fassung 7. September 2020

Satz: PDF-LaTeX 2$_\varepsilon$

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 7. September 2020 .....

Carolin Schindler

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ARTEMIS Advanced Research & Technology for Embedded Intelligent Systems

ASIC Application Specific Integrated Circuits

BET Bounded-Execution-Time

CAN Controller Area Network

CMOS Complementary MOS

CRC Circular Redundancy Check

FPGA Field Programmable Gate Array

GENESYS generische Systemsoftware

INDEXYS industrielle Explorierung von GENESYS

IP-Core Intellectual Property Core

LET Logical-Execution-Time

MOS Metall-Oxid-Halbleiter

NMOS n-Kanal MOS

PET Physical-Execution-Time

PMOS p-Kanal MOS

VHDL Very High Speed Hardware Description Language

ZET Zero-Execution-Time

1 Einleitung

1.1 Motivation

Die Lehrveranstaltung Architektur eingebetteter Systeme, soll ein grundlegendes Verständnis für eingebette Systeme und deren Aufbau vermitteln. Dabei werden alle Teile des Systems von den Sensoren über den Mikrocomputer bis zu den Aktoren betrachtet. Des weiteren ist man anschließend in der Lage einfache eingebette Systeme selbst zu realisieren, beispielsweise auf einem FPGA. Die Veranstaltung sollte eine Basis bilden auf der durch weiterführende Vorlesungen aufgebaut werden kann oder durch Eigenrecherche komplexere Sachverhalte in Bezug auf eingebettete Systeme erarbeitet und verstanden werden können.
So auch bei dem Zusatzthema dieser Arbeit: Generische cross-domain Architektur für eingebettete Systeme am Beispiel von GENESYS. In der Vorlesung wurden verschiedene Architekturen und Technologien besprochen und wie diese kombiniert werden können. Zudem benötigt der Computer die Möglichkeit Eingaben, meist in Form von Sensordaten, zu verarbeiten und die Ergebnisse auszugeben oder für die Steuerung eines Aktors zu verwenden. Da verschiedene Systeme teilweise gleiche Aufgaben zu bewerkstelligen haben, bietet es sich an generische Lösungen für diese zu finden. Als Entwickler möchte man ständig wiederkehrende Aufgaben nicht immer wieder von neuem lösen, sondern eine gut funktionierende Lösung finden, die systemunabhängig wiederverwendet werden kann. Dieses Bestreben begegnet einem auch in der Arbeitswelt, beispielsweise im Rahmen einer Werkstudententätigkeit.
Da eingebette Systeme viele Berührungspunkte mit anderen Disziplinen haben, war das Wissen aus den Vorlesungen Signale und Systeme, Grundlagen der Elektrotechnik, Grundlagen der Rechnerarchitektur, Systemnahe Software I, Grundlagen verteilter Systeme und das Praktikum Mikrokontroller beim Verständnis der Lehrinhalte hilfreich.

1.2 Überblick

Diese Arbeit ist folgendermaßen aufgebaut. Kapitel 2 befasst sich mit der Kurseinlage. In dieser werden fünf Bereiche aus der Vorlesung, welche für das Zusatzthema in Kapitel 3 relevant sind, anhand von Literatur näher betrachtet. Zudem werden die beiden Fragen, die während der Vorlesung zur Bearbeitung gestellt wurden, beantwortet. Anschließend folgt die Ausformulierung des weiterführenden Themas in Kapitel 3. Zum diesem wird mit einer Motivation hingeführt. Anschließend wird Hintergrundwissen und das Thema selbst behandlet, gefolgt von einer Bewertung der gesammelten Erkenntnisse. Die Übungseinlage in Kapitel 4 befasst sich mit der Vorgehensweise bei der Lösung von den Übungsaufgaben und welches Wissen daraus erlangt werden konnte. Zum Schluss wird in Kapitel 5 ein Fazit zur gesamten Vorlesung gezogen.

2 Kurseinlage

Die Mitschrift zu allen Vorlesungen befindet sich im GitHub Repository [1] unter Vorlesungsmitschrift.

2.1 Inhaltliche Diskussion der Vorlesung

2.1.1 Einführung

Zu Beginn machen wir uns klar, was ein eingebettetes System ist.
Ein eingebettetes System ist

- „ein Computer, der steuert und regelt und in einem technischen Kontext eingebunden ist“. Der Benutzer sieht diesen nicht, er ist ihm also nicht zugänglich [10].
- „ein (günstiges) in Massen produziertes Element für ein größeres System und stellt diesem einen vorgegebenen, möglicherweise zeitbeschränkten Dienst zur Verfügung“ [2, übersetzt].
- „ein Rechensystem, [das] in einen technischen Kontext bzw. in ein übergeordnetes System eingebunden [ist] und vordefinierte Aufgaben [erfüllt]“, für die es ausschließlich entwickelt wird [1].

Damit ergibt sich der schematische Aufbau eines eingebetten Systems, der in Abbildung 2.1 dargestellt ist. Die Sensoren wandeln dabei physikalische Größen in elektrische Größen um, die Aktoren machen genau das Gegenteil. Die Wandler werden benötigt um analoge (zeit- und wertkontinuierliche) Signale in digitale (zeit- und wertdiskrete) Signale umzuwandeln und anders herum.

[1] `https://github.com/csacro/AES`

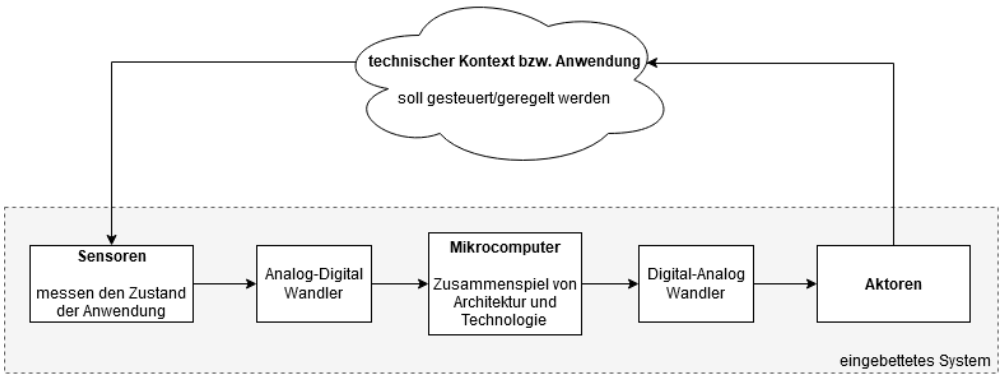

Abbildung 2.1: Schematischer Aufbau eines eingebetteten Systems [10]

Technische Kontexte können beispielweise Flugzeuge, Kraftfahrzeuge, Schiffe, Telefone oder Haushaltsgeräte sein. Eingebette Systeme werden durch die Miniaturisierung allgegenwärtig und finden Einsatz in immer mehr übergeordneten Systemen. Abhängig von der Systemkategorie werden verschiedene Anforderungen gestellt. Die Einteilung der eingebetteten Systeme in Kategorien kann dabei nach Systemeigenschaften wie der Definition der Schnittelle nach außen und der Bedienung der Anwendung erfolgen. Beispiele für besondere Anforderungen sind Echtzeitverhalten, Wiederverwendbarkeit und Skalierbarkeit, verteilte Implementierung, Sicherheit und geringe Verlustleistung [1].

Diskussion
Generell ist es wichtig einen guten Überblick über das Thema zu haben, damit man im späteren Verlauf immer weiß worum es geht, bei welchem Teil des eingebetten Systems man ist und wozu man diesen im Gesamtbild benötigt. Die Quelle [1] nennt nochmals mehr Beispiele und illustriert, neben dem in der Vorlesung besprochenen Kontext Flugzeug, die Kontexte Temperaturregelung und Smartphone.

2.1.2 VHDL - Hardwarebeschreibungssprachen

VHDL steht für Very High Speed Hardware Description Language. Mit dieser Sprache kann das Verhalten und die Struktur von Komponenten auf verschiedenen Ebenen definiert werden und unterstützt damit den Entwicklungsprozess. Zur graphischen Darstellung der Ebenen kann das Y-Diagramm in Abblidung 2.2 herangezogen werden.

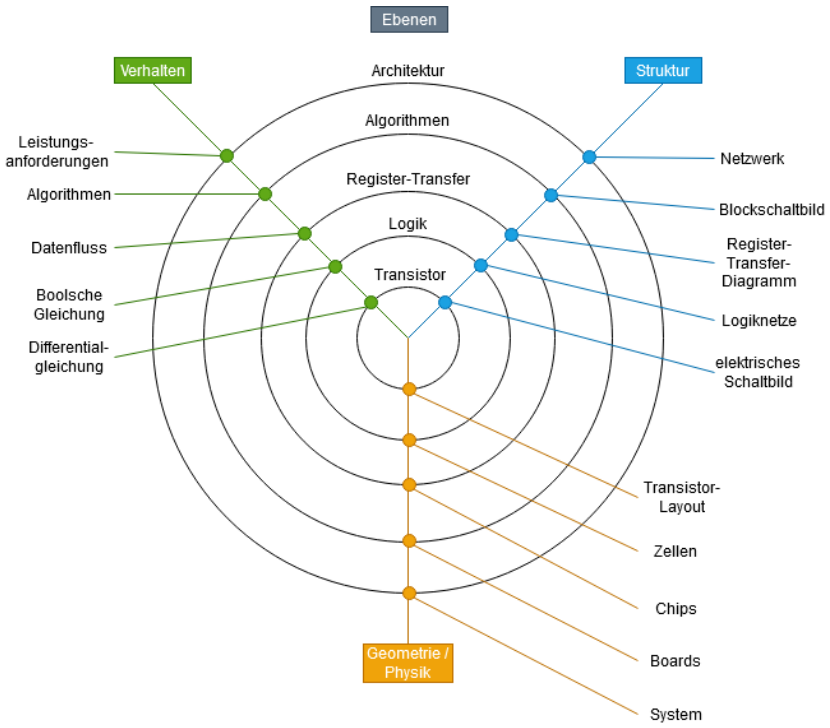

Abbildung 2.2: Y-Diagramm zu Darstellung der Abstraktionsebenen [5, 10]

Da sich das Zusatzthema mit generischen cross-domain Architekturen beschäftigt, betrachten wir in diesem Teil genauer, welche Mechanismen VHDL für eine generische Implementierung anbietet. Allgemein lassen sich sogenannte Entities definieren, welche Ports als Schnittstelle besitzen. Diese Entities - entweder selbst definiert oder aus einer Bibliothek entnommen - können als Komponenten in einer anderen Entity eingesetzt werden. Dabei müssen die Ports der Schnittstelle definiert werden, wodurch eine teilweise generische Verwendung der Entity möglich ist. Zusätzlich besteht die Möglichkeit generische Werte in der Entity selbst einzuführen, die das Verhalten der Komponente beeinflussen. Dies geschieht in der Definition der Entity mit folgendem Befehl:

```
generic (name_of_generic_variable :
            type_of_generic_variable :=
                default_value_of_generic_variable);
```

Will man bei der Einbindung der Komponente nicht den default-Wert der generischen Variable nutzen, kann dies analog zur port map in einer generic map angegeben werden [5].

Diskussion
Für das Verständnis von VHDL hat hauptsächlich die Übung zur Vorlesung, welche in Kapitel 4 näher betrachtet wird, beigetragen. Die Quelle [5] kann hierbei als Nachschlagwerk dienen und gibt Einblick in Aspekte von VHDL, die im Rahmen der Vorlesung nicht behandelt wurden.
In der Vorlesung wurde zuerst nicht klar, weshalb man in einer Entity mehrere Architectures implementieren sollte, die über eine Konfiguration auswählbar sind. Um dies vollständig zu verstehen, war die Erklärung eines vereinfachten Entwurfsablaufs im Kapitel Logic Fabrics der Vorlesung hilfreich. Eventuell könnte man diesen Teil auch vorziehen, damit man bereits in diesem Kapitel ein besseres Verständnis für die Verwendung von VHDL im Entwicklungsprozess bekommt.

2.1.3 Architektur der Komponenten

Eingebettet System können nach der Art ihres Designstiles, welche unterschiedliche Vor- und Nachteile mit sich bringen, unterschieden werden. Eine Auflistung dieser findet sich in Tabelle 2.1. Dabei steht + für eine positive und - für eine negative Eigenschaft. Mit dem Hardwar/Software Co-Design möchte man einen Kompromiss zwischen den zwei anderen Stilen finden, sodass die Vorteile beider Ansätze vereint werden können.

Designstil	Geschwindigkeit	Felixibilität
ASIC basiert	+	-
Mikroprozessor basiert	-	+
Hardware/Software Co-Design	(+)	(+)

Tabelle 2.1: Vor- und Nachteile verschiedener Designstile [2]

Zu einem eingebetteten System gehört meist auch ein eingebetteter Prozessor, der laut [2] in Massen produziert wird, anwendungsspezifisch und häufig statisch programmierbar ist. Zudem ist der Prozessor heterogen, da Hardware- und Softwaredesign und Designstile innerhalb der Software und Hardware vermischt werden. Viele eingebettete Prozessoren sind zudem echtzeitfähig, das heißt sie halten definierte Zeitschranken für die Ausfühung einer Aufgabe ein. Bereits im Jahr 2004 ging der Trend in Richtung programmierbare eingebette Prozessoren, da diese noch flexibler sind und eine Wiederverwendung von Softwaremodulen möglich ist, wenn

diese auf einer Prozessor-unabhängigen Ebene definiert wurden. Dies hat zur Folge, dass immer mehr konfigurierbare Hardware in eingebettete Prozessoren integriert wird [2].

Diskussion
In der Quelle [2] wurden eingebettete Prozessoren allgemein als anwendungsspezifisch bezeichnet. In der Vorlesung haben wir auch einen Instruction Set Processor und einen Application Specific Instruction Set Processor besprochen, welche offensichtlich zur Kategorie der programmierbaren Prozessoren zählen, die zur Zeit der Erstellung der Quelle erst noch im Kommen waren.

2.1.4 Logic Fabrics: Technologie und Entwurf

Logic Fabrics sind Chipfabriken, in denen integrierte Schaltungen hergestellt werden. Dabei kann zwischen drei Technologien unterschieden werden [9]:

- voll-kundenorientierter Entwurf: Layout und Schaltung werden komplett vom Kunden selbst entworfen.
- teil-kundenorientierter Entwurf: Vorgefertigte Elemente aus einer Bibliothek werden vom Kunden zusammengefügt. Beispiele hierfür sind:
  - Standardzellen: Die vorgefertigten Elemente sind Layouts, wodurch der Entwurf aus der Positionierung und Verdrahtung dieser Elemente besteht.
  - Maskenprogrammierung: Lediglich die Verdrahtungsmaske wird vom Kunden gewählt.
- programmierbare Logik: Die Verdrahtung kann vom Kunden über Speicherzellen programmiert werden.
  Hierzu zählt beispielsweise das Field Programmable Gate Array (FPGA), welches in der Übung in Kapitel 4 verwendet wurde.

Die Eigenschaften der unterschiedlichen Technologien werden in Abbildung 2.3 verglichen. Dabei zeigt der Pfeil in Richtung des Anstiegs der jeweiligen Eigenschaft.

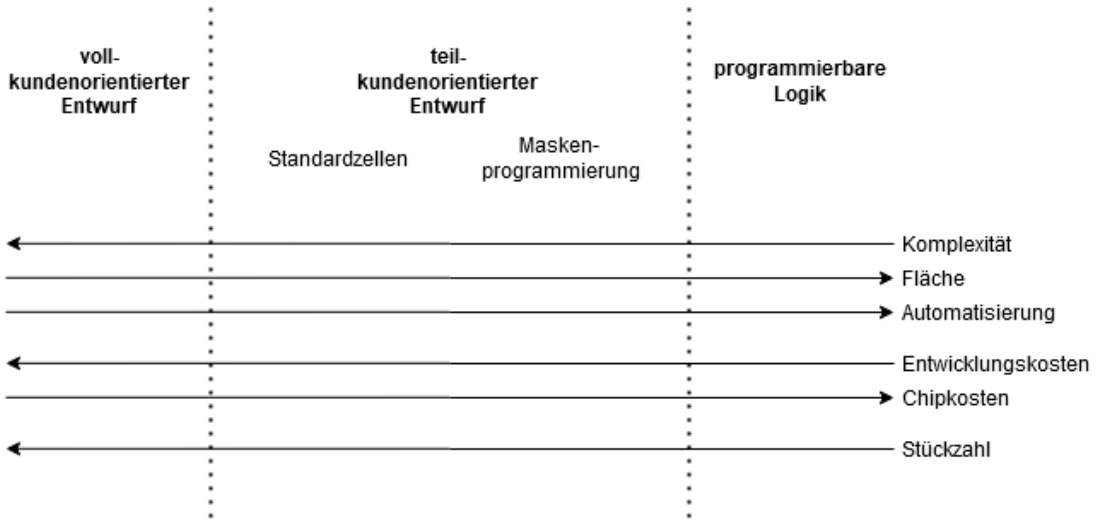

Abbildung 2.3: Eigenschaften der verschiedenen Technologien [10]

Möchte man möglichst flächensparende Systeme in hoher Stückzahl entwickeln, sollte man einen voll-kundenorientierten Entwurf wählen, der allerdings hohe Entwicklungskosten mit sich bringt.

Diskussion
Die Quelle [9] war bereits vor der Vorlesung zu diesem Kapitel bekannt. In ihr werden weitere Beispiele für Maskenprogrammierung und programmierbare Logik aufgeführt. Zudem sind die Erklärungen zu den einzelnen Technologien ausführlicher und mit mehr Grafiken illustriert. Da Grundlagen der Rechnerarchitektur zu den Voraussetzungen für diese Lehrveranstaltung zählt, war der Umfang, der in der Vorlesung vorgestellt wurde, angemessen.

2.1.5 Echtzeitsysteme

Um ein eichtzeitfähiges eingebettetes System zu fertigen, haben sich verschiedene Echtzeit-Programmierungs-Modelle entwickelt, die es ermöglichen ein Kontroll-Design in eine Implementierung mit entsprechenden zeitlichen Eigenschaften umzuwandeln. Um die Entwicklung aufzuzeigen werden die Modelle im folgenden in ihrer Entstehungsreihenfolge besprochen.

Physical-Execution-Time (PET) Programmierung wurde für die Verwendung auf Prozessoren mit einfachem Befehlssatz entworfen und nimmt konstante Ausführungszeiten an. Diese Programmierung hat eine hohe zeitliche Genauigkeit und Verzögerungen von Ein- und Ausgabe sind genau berechenbar. Allerdings ist keine Nebenläufigkeit möglich und ein Datenfluss mit mehreren Komponenten kann ebenfalls nicht realisiert werden. Mit der Entwicklung von Betriebssystemen und Echtzeit-Schedulern wurde PET durch Bounded-Execution-Time (BET) Programming abgelöst, welches die Realzeit und Parallelisierung berücksichtigt. Beim BET Modell kann die parallele Ausführung einer weiteren Aufgabe dazu führen, dass sich das I/O-Verhalten von bereits laufenden Aufgaben verändert. Dieser Nachteil wird durch die Einführung von Zero-Execution-Time (ZET) Programming, welches auf einer synchronen Sematik basiert, behoben. Dabei verliert ZET allerdings auch die Ausnutztung des Echtzeit-Schedulings. Um die Vorteile des BET und ZET Modells zu vereinen, wurde Logical-Execution-Time (LET) Programming entwickelt, welches auf einer logischen Zeit basiert [4].

Diskussion
Da die einzelnen Ausführungsmodelle und Arten von Zeit aus dem Vorlesungskapitel bekannt sind, wurde hier das Augenmerk auf den historischen Zusammenhang gelegt, welcher in der Quelle [4] beschrieben ist. Aus der Vorlesung wurde nicht klar, dass die Modelle für die Generierung einer automatisierten Implementierung mit der jeweiligen zeitlichen Ablaufcharakteristik verwendet werden können. Zudem illustriert die Quelle für jedes Modell die zeitliche Ausführung anhand eines vorgegebenen Beispiel-Codes und gibt weitere Details.

2.2 Beantwortung und Diskussion der Fragen des Kurses

2.2.1 Was ist ein eingebettetes System?

Ein eingebettetes System ist ein kleiner Computer, welcher Teil eines anderen Systems ist. Das eingebettete System erfüllt eine bestimmte, in sich abgeschlossene Funktion und kann somit in verschiedenen Systemen, die diese Funktion benöti-

gen, eingesetzt werden. Beispielhafte Funktionen sind: Signalverarbeitung, Regelung und Steuerung. Die Eingabe hierfür kann von anderen Teilen des Systems oder Sensoren sein. Die Ausgabe erfolgt an ein Teil des Gesamtsystems oder einen Aktor. Funktionen können sowohl in Hardware (MOS-Technik), als auch in Software (native Programmiersprachen) implementiert werden.
Qualitätsmerkmale für eingebettete Systeme sind zum Beispiel Zuverlässigkeit, Sicherheit (vor Angriffen) und Einhaltung von Zeitschranken.
In einem Gesamtsystem sind häufig mehrere eingebettete System verbaut, die alle miteinander arbeiten und kommunizieren müssen. Dabei muss beachtet werden, dass sich die Systeme nicht gegenseitig beeinflussen (Wärme, elektromagnetische Wellen, ...). Zudem werden die Systeme oftmals nicht seriell, sondern parallel verwendet. Das heißt Nebenläufigkeit muss bei der Umsetzung berücksichtigt werden. Ein weiterer Vorteil von eingebetteten System ist, dass ihre Funktion unabhängig vom Gesamtsystem getestet werden kann und das spätere System davon ausgehen kann, dass sich das eingebettete System wie erwartet verhält. Eventuelle Fehler müssen also in der Regel nicht innerhalb der Einheit des eingebetteten Systems gesucht werden, wenn dieses nach Spezifikation eingesetzt wurde.

Diskussion
Diese Frage diente zur Einführung und regte dazu an, sich selbst Gedanken über eingebettete Systeme zu machen, ohne zu recherchieren. Dabei sollten verschiedene Aspekte berücksichtigt und beleuchtet werden. Bei der Beantwortung der Frage ist Wissen aus verschiedenen Vorlesungen des Bachelorstudiums Informationssystemtechnik eingeflossen.

2.2.2 Herstellung einer integrierten Schaltung

Integrierte Schaltungen werden in Chipfabriken, sogenannten Fabs hergestellt. Zu Beginn wird ein Wafer benötigt, auf dem die gedruckten Schaltungen realisiert werden können. Wafer sind Silizium-Scheiben und entstehen durch folgende Schritte:

- Das Silizium wird durch Reduktion, z.B. mit Magnesium, aus Quarzsand gewonnen (Quarzsand ist oxidiertes Silizium).
- Das gewonnene Silizium wird geschmolzen und gereinigt.

- Das Silizium wird in eine Silizium-Schmelze getaucht und wieder herausgezogen, wodurch ein säulenförmiger Siliziumkristall entsteht.
- Nach nochmaliger Reinigung wird der Kristall in Scheiben geschnitten. Diese Scheiben sind die einzelnen Wafer.

Anschließend können die integrierten Schaltungen mit dem Verfahren des Planarprozesses hergestellt werden:

1. Oxidierung der Oberfläche des Wafers.
2. Ebenenweise Erzeugung der Struktur der Schaltung.
   Dies wird für jede Ebene mit Hilfe eines fotolithografischen Verfahrens umgesetzt, das in folgenden Schritten abläuft:
   - Eine Schicht Fotolack wird auf den Wafer aufgetragen.
   - Der Wafer wird durch eine Maske, eine Art Schablone, mit UV-Licht belichtet. An den freien Stellen der Maske wird der Fotolack durch die Belichtung fest.
   - Anschließend kann mit Flusssäure geätzt werden. Dabei wird Fotolack, welcher im vorherigen Schritt nicht gehärtet wurde entfernt und somit auch darunter liegende Schichten geätzt. Gehärteter Fotolack bleibt von der Ätzung unberührt.
   - Die geätzten Stellen können nun gezielt durch Diffusion mit Fremdstoffen bearbeitet werden
3. Passivierung des Wafers.
   Hierzu wird eine weitere Oxidschicht auf den Wafer aufgetragen.
4. Aufbrechung des Wafers in die einzelnen Chips.

Die einzelnen Prozessschritte sind aufwändig und teuer. Da aber viele Chips gleichzeitig auf einem Wafer gefertigt werden können, teilen sich die Kosten auf die einzelnen Chips auf.

Eine CMOS-Schaltung besteht aus PMOS- und NMOS-Transistoren. Bei der Herstellung wird zuerst in das positiv dotierte Silizium eine negativ dotierte Wanne eingebracht. Mit dem positiv dotierten Teil wird dann der NMOS-Transistor weiter

realisiert und mit dem negativ dotierten Teil der PMOS-Transistor. Da Löcher weniger beweglich sind als Elektronen, ist die Leitfähigkeit des positiv dotierten Siliziums niedriger als die des negativ dotierten Siliziums. Um dies auszugleichen ist der PMOS-Transistor breiter als der NMOS-Transistor ($R \sim \frac{l}{A}$).
Bei NMOS- bzw. PMOS-Transistoren wird die Struktur in folgenden Schritten ausgehend von einem passend dotierten Substrat hergestellt, welche in Abbildung 2.4 veranschaulicht sind:

1. Herstellung des Gateoxids
   - Oxidieren
   - Fotolack aufbringen
   - Maskieren und belichten ($\rightarrow$ Maske)
   - Ätzen
   - Gateoxid erzeugen
   - Siliziumnitrit entfernen
2. Diffusion des Transistors
   - Silizium aufbringen
   - Gatestruktur erzeugen ($\rightarrow$ Fotolack, Maske, Belichtung)
   - Silizium ätzen
   - Source und Drain diffundieren
3. Anschlüsse des Transistors
   - Oxidieren
   - Kontaktlöcher strukturieren ($\rightarrow$ Fotolack, Maske, Belichtung)
   - Kontaktlöcher ätzen
   - Metallisieren

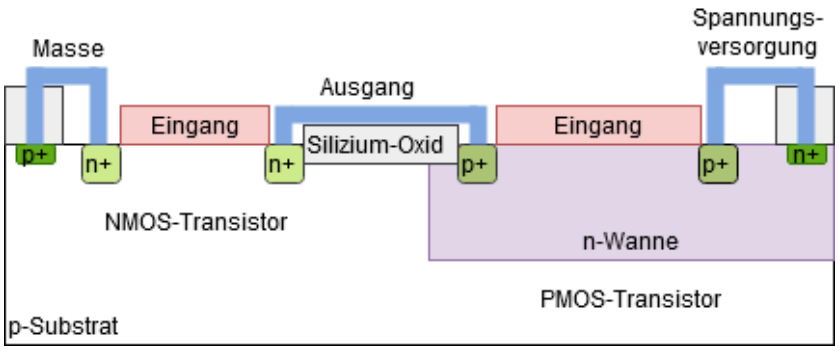

Abbildung 2.4: Struktur eines CMOS-Inverters [9]

Diskussion

Dieses Wissen ist bereits in der Vorlesung Grundlagen der Rechnerarchitektur vermittelt worden. Daher konnte die Frage unter Zuhilfenahme von [9] ohne weitere Probleme bearbeitet werden.

Da die Frage vor der entsprechenden Vorlesung gestellt wurde, hatte man die Chance sein Wissen aufzufrischen und sich somit selbstständig auf das Thema vorzubereiten.

3 Generische cross-domain Architektur für eingebettete Systeme am Beispiel von GENESYS

3.1 Motivation

Eingebettete Systeme haben sich in den letzten Jahren in verschiedenen Gebieten etabliert. Zuerst wurden sie einzeln als sogenannte stand-alone Komponenten eingesetzt. Später wurden die Systeme innerhalb derselben Domäne miteinander verbunden. Hierfür definierte man domänen-spezifische Echtzeit-Kommunikations-Protokolle, wie beispielweise CAN für die automobile Anwendung. Da nun eingebettete Systeme aus verschiedenen Anwendungsdomänen, zum Beispiel Multimedia-Systeme und Auto-Systeme oder Geräte im Internet of Things, miteinander interagieren sollen, stehen folgende Punkte im Vordergrund:

- Interoperabilität: Verbindung von eingebetteten Subsystemen verschiedener Domänen.
- Technologie-Einschränkungen: Durch die Miniaturisierung ist es sehr teuer neue Masken für die Herstellung von integrierten Schaltungen zu entwerfen. Daher ist es erstrebenswert generische Systeme zu erstellen, die in verschiedenen Gebieten eingesetzt werden können.
- personelle Ressourcen: Wenn jede Anwendung ihre eigenen Lösungen, Protokolle und Standards hat, wird für jede Domäne ein Training benötigt, um diese zu verstehen und mit diesen arbeiten zu können.

Daher bietet es sich an eine einheitliche generische cross-domain Architektur zu entwickeln [8].

3.2 Konzept

3.2.1 Generizität, generische Softwarekomponenten und Generatortechniken

Generizität ist ein Konzept, bei dem eine Struktur in variante und invariante Teile unterteilt wird. Eine generische Softwarekomponente realisiert dieses Konzept. In dieser sind die varianten Teile durch genersiche Parameter beschrieben, die es ermöglichen die Eigenschaften der Komponente je nach Anwendung festzulegen. Diese Parameter stellen auch den Ansatzpunkt für Generatoren dar, die Komonenten-Code aus einer abstrakten Beschreibung erstellen.
Durch wiederholte Auswahl, Konfiguration und Kombination von generischen Softwarekomponenten kann also eine anwendungsspezifische Laufzeitplattform erstellt werden [6].

3.2.2 ARTEMIS Strategic Research Agenda

Advanced Research & Technology for Embedded Intelligent Systems (ARTEMIS) ist eine europäische Technologieplattform für eingebettete Systeme und entwirft Strategic Research Agendas, welche die wichtigsten Forschungsbereiche für eingebettete Systeme enthalten. Dabei wurden im Jahr 2006 folgende Herausforderungen für eine generische cross-domain Architektur identifiziert [7, 8]:

1. Kombinierbarkeit: Die Architektur muss eine Zusammensetzung von einzelnen Komponenten und Subsystemen zu einem großen System ohne unkontrollierte Seiteneffekte unterstützen.
2. Vernetzung: Teilsysteme müssen untereinander und mit ihrer Umgebung auf verschiedenen Ebenen kommunizieren können. Dabei sind Zeit- und Zuverlässigkeitsbedingungen einzuhalten.
3. Sicherheit: Bösartige Angriffe auf das System müssen mit modernen Ansätzen gehandhabt werden, damit vertrauliche Daten und geistiges Eigentum geschützt sind. Darunter zählt auch der Schutz vor Manipulationen des Systems. Moderne Ansätze impliziert, dass das System so erstellt werden muss,

dass es in einem ständig andauernden Prozess verbessert und weiterentwickelt werden kann. Nur so ist sichergestellt, dass die Abwehr von Angriffen auf einem zeitgemäßen Stand ist und auch bisher unbekannte Angriffe gehandhabt werden können.

4. Robustheit: Fehler können zweitweise oder permanent auftreten und von verschiedenen Quellen wie Hardware, Design, Spezifikation oder Zufall ausgelöst werden. Trotz dieser muss eine aktzeptable Funktionalität vorhanden sein.
5. Diagnose und Wartung: Eine zuverlässige Fehlererkennung soll möglich sein, damit diese automatisch oder unter Anleitung eines Nutzers behoben werden können.
6. Integriertes Ressourcen-Management: Die Architektur muss wesentliche Ressourcen wie Energie, Zeit und Speicher verwalten können.
7. Entwicklungsfähigkeit: Die Architektur soll die Weiterentwicklung von Systemen oder bisher nicht bedachten Anforderungen unterstützen.

3.2.3 Europäisches Forschungsprojekt GENESYS

Ein weit verbreitetes Verfahren zur Softwareentwicklung ist der Top-Down-Ansatz. Bei diesem werden zuerst alle Anforderungen an das System formuliert und anschließend eine Lösung erstellt, die schrittweise verfeinert wird, bis diese implementiert werden kann. Allerdings entstehen dabei meist Unikate, die nicht wiederverwendet werden können und mit zunehmender Komplexität der heutigen Systeme ist eine vollständige Anfoderungsdefinition vor der Lösungsfindung nicht realistisch. Daher hat sich der europäische Sonderforschungsbereich 501 „Entwicklung großer Systeme mit generischen Methoden“ die Aufgabe gestellt, die Wiederverwendung von Software und den zu ihrer Fertigung benutzten Prozesse voranzutreiben. Dabei ist der Sonderforschungsbereich 501, wie in Abbildung 3.1 dargestellt, in verschiedene Projektbereiche unterteilt und diese besitzen wiederum mehrere Teilprojekte. Im folgenden soll genauer auf die Ergebnisse des Teilprojekts B5 generische Systemsoftware (GENESYS) eingegangen werden, welches genererische Ansätze für eingebettete Systeme auf der Basis von genersichen Softwarekomponenten und dem Einsatz von Generatortechniken untersucht [6].

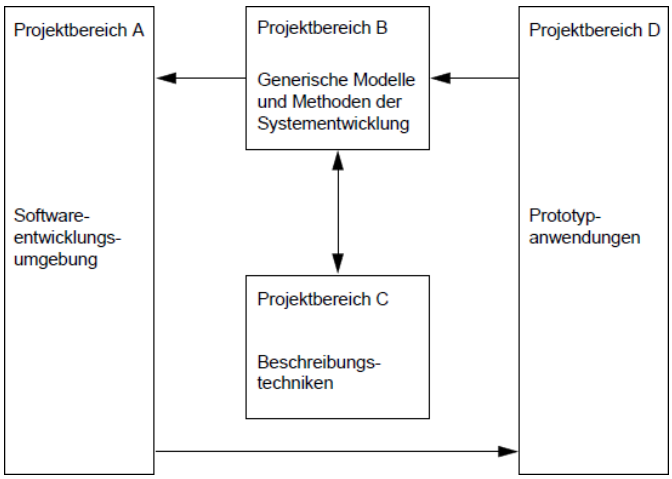

Abbildung 3.1: Organisationsstruktur des SFB 501 [6]

3.3 Umsetzung

3.3.1 Architekturprinzipien von GENESYS

Im folgenden werden die Architekturprinzipien von GENESYS aufgelistet und kurz erläutert [8].

- Strikte Komponenten-Orientierung
  Eine Komponente ist ein in sich abgeschlossenes Subsystem, dass unabhängig vom System entwickelt werden kann. Sie besitzt, wie in Abbildung 3.2 dargestellt, verschiedene Schnittstellen, wodurch eine Abstraktion als Designbaustein möglich ist.

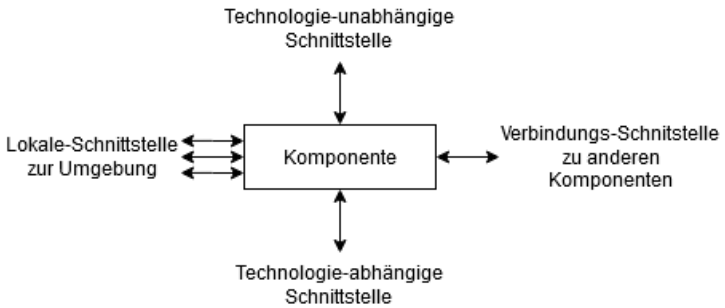

Abbildung 3.2: Schnittstellen einer Komponente [8]

- Strikte Trennung von Kommunikation und Berechnung
  Die Kommunikations-Struktur dient zur Integration einer Komponente in das Gesamtsystem. Durch die Trennung wird die Komplexität reduziert, da beide Teile getrennt entworfen und validiert werden können. Es kann einerseits die interne Implementierung verändert werden, ohne dass Änderungen an anderen Teilen des Systems notwendig sind, solange die Defintion der Kommunikations-Schnittstelle beibehalten wird. Andererseits kann durch Anpassung der Kommunikation eine implementierte Berechnung in verschiedenen Systemen eingebaut werden.

- Kommunikation
  Zur Realisierung der Kommunikation wird das Konzept des Nachrichtenaustauschs herangezogen. Eine Nachricht hat einen eindeutigen Sender und ein oder mehrere Empfänger. Zudem besitzen Nachrichten eine inhärente Konsistenz und können zur Synchronisation verwendet werden, wodurch ein höheres Abstraktionslevel verglichen mit einer Bus-Schnittstelle entsteht.
  Da verschiedene Anwendungen unterschiedliche Anforderungen an die Kommunikations-Struktur haben können, gibt es drei verschiedene Nachrichtentypen: periodische Nachrichten, sporadische Nachrichten und synchronisierte Datenströme.

- Verfügbarkeit einer gemeinsamen Zeit
  Eine einheitliche Zeitbasis ermöglicht die Koordination von verteilten Aufgaben und erlaubt es Zeitstempel von verschiedenen Komponenten in Beziehung zu setzen. In sicherheitskritischen Systemen muss die Zeitbasis zusätzlich fehlertolerant sein, da es sein kann, dass Dienste von dieser abhängen.

- Hierarchische Systemstruktur
  Abbildung 3.3 zeigt die Struktur, die in drei verschiedene Ebenen unterteilt ist. In jeder Ebene gibt es Dienste, die in die Kategorien Kern-Services und optionale Services eingeteilt sind. Die Unterteilung in Chip-Ebene, Geräte-Ebene und System-Ebene wurde gewählt, da sich die Eigenschaften der Dienste zwischen diesen Ebenen grundlegend unterscheiden. Komponenten, die auf der gleichen Ebene interagieren werden über die Verbindungs-Schnittstelle, welche in Abbildung 3.2 aufgeführt ist, verbunden. Für die Kommunikation zwischen den Ebenen werden zusätzlich Gateway-Komponenten verwendet.

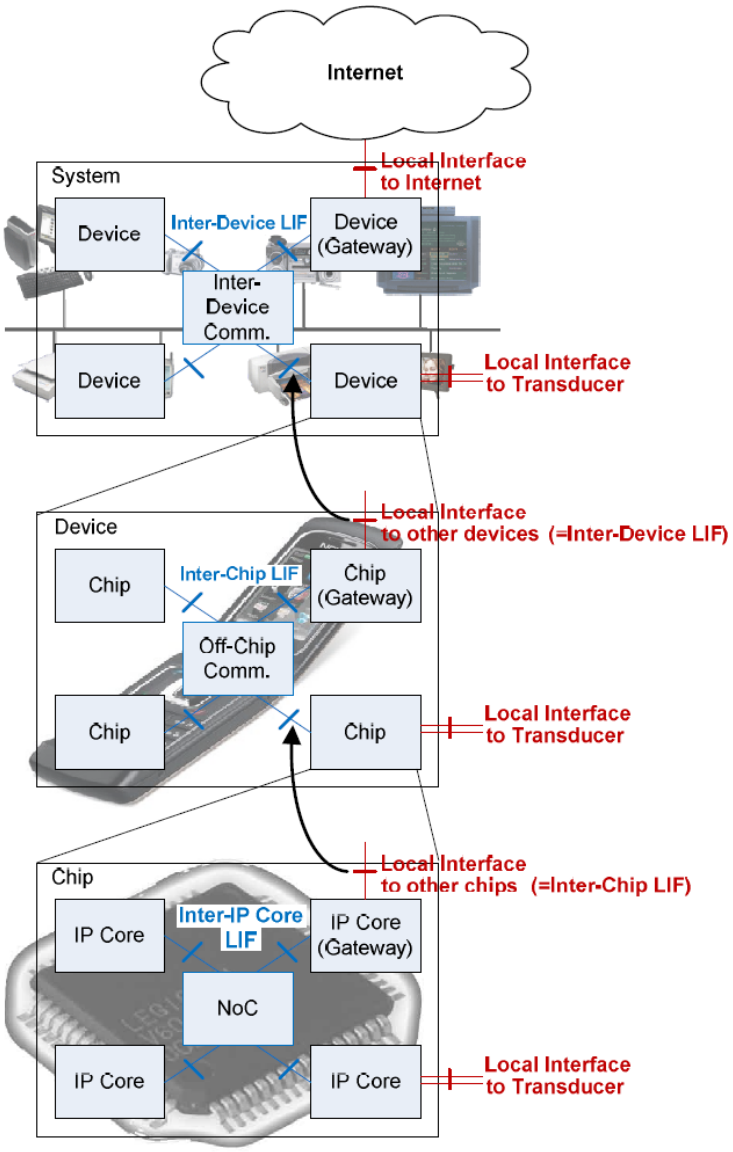

Abbildung 3.3: Integrationsebenen [8]

- Zustandsbewusstsein

  Dieses Prinzip wird benötigt, um den Zustand einer Komponente im System zu kennen und trägt zur Wiederherstellung des Zustands im Fall von temporären Fehlern bei. Hierzu wird ein sogenannter Reintegrations-Punkt für jede Komponente definiert. An diesem Punkt wird der Zustand für eine mögliche Zurücksetzung gespeichert und über Nachrichten nach außen propagiert. Dadurch ist eine online Diagnose möglich und Tests und Debugging können außerhalb des Systems durchgeführt werden.

- Fehlerisolation
  Es werden Fehler-Regionen definiert mit der Eigenschaft, dass ein Fehler innerhalb dieser Region keine Auswirkungen auf eine andere Region hat. Aufgrund der strikten Komponenten-Orientierung kann jede Komponente eine eigene Fehler-Region darstellen.
- Wissen und Kontrolle über Ressourcen
  Eingbettete Systeme müssen sich in fast jeder Domäne an den wechselnden Kontext anpassen. Deshalb wird ein integriertes Ressourcenmanagement vorgeschlagen, welches eine ganzheitliche Sicht auf die verschiedenen Ressourcen bietet. Bei der Zuweisung der Ressourcen an einzelne Komponenten muss dabei immer zwischen Energieeffizienz und Performance abgewägt werden. Um möglichst vielen Anwendungstypen gerecht zu werden, muss es mehrere Strategien für die Ressourcenzuweisung geben.
- Struktur von Diensten
  Es sollen mehrere standardisierte, validierte und zertifizierte Architektur-Dienste angeboten werden. Diese trennen die Funktionalität der Anwendung von der zur Realisierung verwendeten Technologie. Dadurch wird die Komplexität des Designs reduziert und dessen Wiederverwendung ermöglicht. Um die Dienste auf verschiedenen Domänen mit unterschiedlichen Anforderungen verwenden zu können, müssen sie erweiterbar und konfigurierbar sein.

3.3.2 Architektur von GENESYS

Aus den oben vorgestellten Architekturprinzipien ergibt sich folgende Umsetzung in eine Referenzarchitektur:
Das Gesamtsystem wird aus Komponenten zusammengesetzt, wie sie oben beschrieben sind. Auf der Chip-Ebene sind die Komponenten IP-Cores, auf der Geräte-Ebene sind sie Chips und auf der System-Ebene sind sie Geräte. Dies ist bereits in Abbildung 3.3 mit aufgenommen.
Die oben angesprochenen Dienste werden ebenfalls in einer Hierachie angeordnet. Diese ist in Abbildung 3.4 dargestellt. Zu den Kern-Services zählt der Zeit-Dienst, der die einheiltiche Zeit zur Verfügung stellt und der Kommunikations-Dienst, welcher den Nachrichtenaustausch realisiert. Der Ausführungskontroll-Dienst, welcher unter anderem die Ressourcenverteilung regelt und der Konfigurations-Dienst, der es ermöglicht Komponenten in das Gesamtsystem zu integrieren, zählen ebenfalls

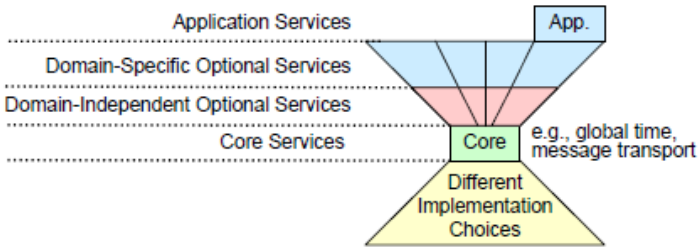

Abbildung 3.4: Service-Hierarchie in GENESYS [7]

zu den Kern-Services. Der Diagnose-Dienst und Sicherheits-Dienst sind Beispiele für Domänen-unabhänginge optionale Services, welche auf den Kern-Services aufbauen. Zuletzt gibt es Domänen-spezifische optionale Services, die zur Realisierung von Protokollen innerhalb einer Anwendungsdomäne verwendet werden können. So kann beispielsweise eine CAN Kommunikation für automotive Anwendung genutzt werden. Eine detailliertere Auflistung über alle zur Verfügung gestellten Dienste bietet Abbildung 3.5 [7].

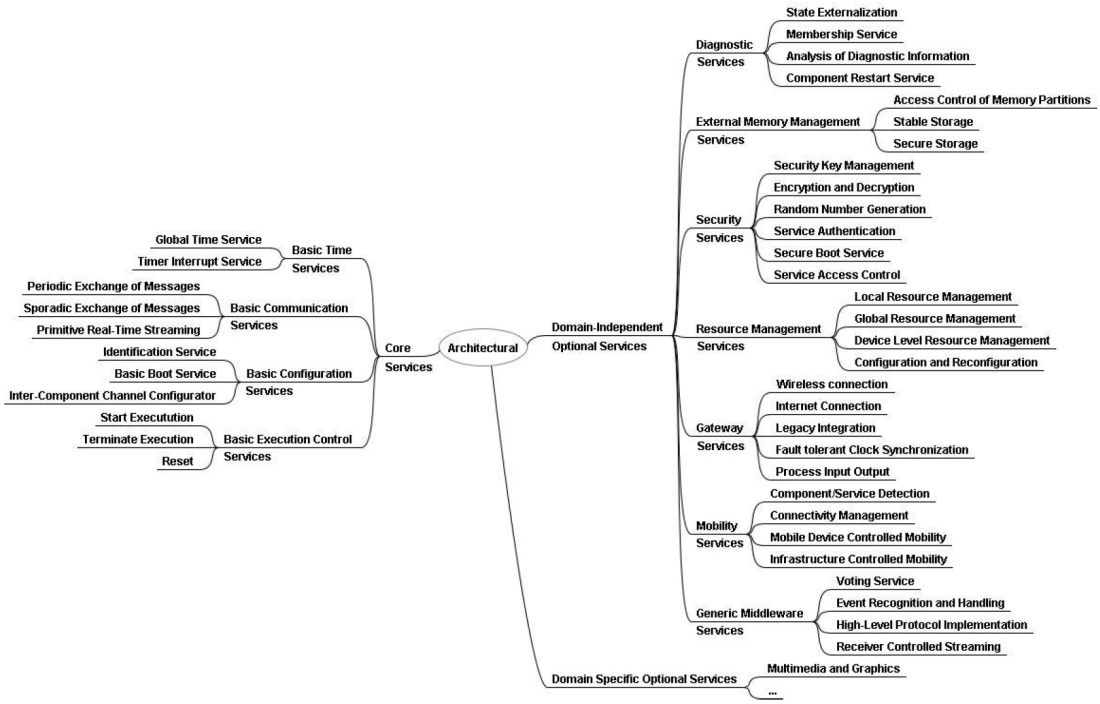

Abbildung 3.5: Übersicht über Services in GENESYS [7]

3.4 Validierung und Bewertung

GENESYS erfüllt durch die Grundlage der Architekturprinzipien die oben vorgestellten ARTEMIS Herausforderungen. In Tabelle 3.1 ist veranschaulicht welche Prinzipien zur Erfüllung welcher Herausforderung (Nummerierung entspricht Auflistung oben) beitragen und über die Anzahl der + kann abgelesen werden in welchem Maß [7, 8].

	ARTEMIS Herausforderungen						
GENESYS Architekturprinzipien	1	2	3	4	5	6	7
Strikte Komponenten-Orientierung	++	+	+	++	++	++	++
Strikte Trennung von Kommunikation und Berechnung	++	++	+	++	++	+	++
Verfügbarkeit einer gemeinsamen Zeit	++	+	+	+	+		
Hierarchische Systemstruktur	++		+	+	+	+	+
Kommunikation	++	++	+	+	+	+	+
Zustandsbewusstsein	+		+	++	++	+	+
Fehlerisolation	++		++	++	++		+
Wissen und Kontrolle über Ressourcen	+	+		+	+	++	+
Struktur von Services	++	++	+	+	+		++

Tabelle 3.1: GENESYS Architeturprinzipien und ARTEMIS Herausforderungen [8]

Des weiteren gibt es INDEXYS, das die industrielle Explorierung von GENESYS vorantreiben will. Hier wurde erfolgreich gezeigt, wie die Architektur von GENESYS mit den Services in der Automobil-, Flugzeug- und Bahnindustrie genutzt werden kann. Damit ist klar, dass die Ergebnisse von GENESYS in der Industrie anwendbar sind, wodurch ein praktischer Nutzen aus der Forschung gezogen werden kann [3].

4 Übung

Für die erstellten Lösungen zu den einzelnen Übungsaufgaben auf dem FPGA wird an dieser Stelle auf Anhang A verwiesen.
Im folgenden wird das allgemeine Vorgehen beim Bearbeiten einer Aufgabe beschrieben und anschließend wird die Übung zur Vorlesung selbst bewertet.

4.1 Konzept

Zuerst muss die Aufgabenstellung der Übung verstanden werden. Dabei wird die Frage was generell getan werden muss beantwortet, aber noch nicht wie es getan wird. Danach muss überlegt werden, wie die Ein- und Ausgabe erfolgen soll und welche Komponenten dafür benötigt werden. Anschließend muss sich um die Umsetzung der eigentlichen Aufgabe Gedanken gemacht werden. Dabei ist es hilfreich zu überlegen, ob Wissen aus einer zuvor bearbeiten Aufgabe verwendet werden kann, wobei insbesondere die bearbeiteten Tutorials als Richtlinie dienen können.
Eventuell besteht auch die Möglichkeit, die Aufgabe in mehreren Teilschritten zu lösen und somit die Komplexität und den Aufwand für eine eventuelle Fehlersuche zu minimieren. So kann beispielsweise bei Kapitel 12 der Übung vorgegangen werden: Zuerst wird die Eingabe in Software realisiert. Anschließend kann die Ausgabe mittels Polling und in einem weiteren Schritt mittels Interrupt-Signal in Software realisiert werden.

4.2 Lösung

Bei der Umsetzung des Knozepts wurde darauf geachtet Gedankengänge zu dokumentieren und Code zu kommentieren, um späteres Nachschlagen und Verstehen zu vereinfachen. Bei auftretenden Problemen oder Fehlverhalten in der Validierung

wurde versucht diese in Eigenrechereche zu beheben. Zudem wurden unbenutzte LEDs auf dem Board für Debug-Ausgaben genutzt. Insbesondere bei Kapitel 8 wurde dieses Verfahren verwendet, wo ein extra Projekt für das Debugging angelegt wurde. Im nächsten Schritt wurden Kommilitonen des gleichen Studiengangs um Unterstützung gebeten. Hierbei kam es meist zu einer vergleichenden und diskutierenden Zusammenarbeit mit Dominik Authaler, da wir meist auf gleichem Stand bei der Bearbeitung der Aufgaben waren und interessanterweise bei den gleichen Aufgaben Probleme hatten. Wenn dies alles nichts geholfen hat, wurde der Übungsleiter Marco Philippi um Unterstützung gebeten.

4.3 Validierung

Die ersten Aufgaben wurden noch mit dem Quartus II Simulations Tool simuliert und validiert. Ab Kapitel 6 der Übung fand die Validierung aus Komplexitätsgründen direkt auf dem Board statt. Wurde dabei festgestellt, dass die Implementierung nicht mit der Aufgabenspezifikation übereinstimmt, ging es an die Fehlersuche. Eine Testbench wurde im Rahmen der Übung nicht implementiert, da eine direkte Validierung auf dem Board immer möglich war und dort auch das Debugging live durchgeführt werden konnte.

4.4 Diskussion und Bewertung

Die Übung trägt zum Verständnis der Vorlesung bei. Insbesondere gibt sie Einblicke, wie eingebettete Systeme im Kleinen implementiert werden und welche Design-Entscheidungen dabei zu treffen sind. Der rote Faden ist während der gesamten Bearbeitung erkennbar. Die ersten Aufgaben, die sich noch nicht mit dem Circular Redundancy Check (CRC) beschäfigen, sind hilfreich, um sich mit dem Quartus II Tool und VHDL vertraut zu machen. Insgesamt steigern sich die Aufgaben von leicht nach schwer. Zudem ist es positiv, dass die Bearbeitung von Tutorials expliziter Teil der Übung ist. Diese führen gut in die neue Thematik des Kapitels ein, wodurch man sich nicht komplett eigenständig einarbeiten muss.

Etwas frustrienden waren Software-/Tool-bedingte Fehler, die sporadisch aufgetreten sind und nichts mit der eigenen Implementierung zu tun hatten. Diese wurden meist erst nach längerer Zeit der Fehlersuche in der eigenen Realisierung als solche erkannt, wenn sie dann plötzlich, ohne erkennbaren Grund, verschwunden waren.

5 Fazit

Insbesondere durch das eigene praktische Arbeiten in der Übung, wurden neue Erkennntnisse in der Entwicklung von eingebetteten Systemen erlangt und die Fähigkeiten Aufgaben und Probleme eigenständig zu lösen erweitert. Einzelne Teile der Vorlesung waren bereits aus anderen Veranstaltungen bekannt. Da die Zielgruppe aber heterogen ist, ist es sicherlich sinnvoll diese Teile nicht zu streichen, damit alle den gleichen Wissenstand haben. Die Veranschaulichung durch Beispiele, welche die Vorlesung ständig begleiten, hilft dabei die Inhalte besser miteinander verknüpfen zu können und lockert den Vortrag auf.
Im Zusatzthema der Arbeit wird die Notwendigkeit von generischen eingebetten Systemen dargestellt und dies sogar über die Grenzen einzelner Domänen hinweg. ARTEMIS veröffentlich jährlich eine Strategic Research Agenda[1], die aktuelle Herausforderungen für eingebette Systeme enthält. Daran sieht man, dass es in diesem Bereich noch viele ungelöste Punkte gibt, die es zu erforschen gilt. Die Herausforderungen für eine generische cross-domain Architektur wurden im Forschungsprojekt GENESYS addressiert. Diese Froschungsergebnisse können, wie INDEXYS zeigt, in der Industrie praktisch eingesetzt werden.
Da eingebette Systeme allgegenwärtig sind, ist es sinnvoll diese in ihren Grundzügen zu verstehen. Selbst wenn man nicht direkt an diesen entwickelt, sind sie dennoch Teil des Gesamtsystems und haben damit Einfluss auf die eigene Arbeit. Um sein Wissen zu vertiefen und mehr ins Detail zu gehen, kann der interessierte Student auf Literatur zurückgreifen oder eine der weiterführenden Vorlesungen besuchen, welche die Universität Ulm anbietet. Zudem können eigene eingebette Systeme, beispielsweise im Haushalt auf Basis eines Raspberry Pi, entwickelt werden.

[1] `https://artemis-ia.eu/documents.html`

A Lösung der Übung

Die Lösung und Implementierung zu den einzelnen Übungsaufgaben befindet sich im GitHub Repository [1] unter Uebung.
Das README.md File in diesem Verzeichnis wurde für aufgabenweise Notizen und zur Dokumentation von einzelnen Schritten und Gedankengängen genutzt. Zudem sind dort aufgetretene Probleme und gefundene Lösungen festgehalten.

[1] https://github.com/csacro/AES

Literatur

[1] Oliver Bringmann, Walter Lange und Martin Bogdan. *Eingebette Systeme - Entwurf, Modellierung und Synthese*. Berlin: Walter de Gruyeter GmbH & Co KG, 2018. ISBN: 978-3-110-51852-8.

[2] Sorin Cotofana, Stephan Wong und Stamatis Vassiliadis. *Embedded Processors: Characteristics and Trends*. Techn. Ber. Delft University of Technology, März 2004.

[3] Andreas Eckel u. a. „INDEXYS, a Logical Step beyond GENESYS“. In: *Proceedings of the 29th International Conference on Computer Safety, Reliability, and Security*. Aug. 2010, S. 431–451.

[4] Christoph M. Kirsch und Raja Sengupta. „The Evolution of Real-Time Programming“. In: *Handbook of Real-Time and Embedded Systems*. 2007.

[5] Andreas Mäder. *VHDL Kompakt*. Universität Hamburg: Fakultät für Mathematik, Informatik und Natruwissenschaften.

[6] Jürgen Münch und Bernd Schürmann. „Sonderforschungsbereich 501: Entwicklung großer Systeme mit generischen Methoden“. In: *it - Information Technology* 45 (Apr. 2003), S. 227–236.

[7] Roman Obermaisser und Hermann Kopetz. „From ARTEMIS Requirements to a Cross-Domain Embedded System Architecture“. In: *ERTS2 2010, Embedded Real Time Software & Systems*. Mai 2010.

[8] Roman Obermaisser u. a. „Fundamental Design Principles for Embedded Systems: The Architectural Style of the Cross-Domain Architecture GENESYS“. In: *2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. März 2009, S. 3–11.

[9] Prof. Dr.-Ing. Frank Slomka. *Grundlagen der Rechnerarchitektur - Von der Schaltung zum Prozessor*. Universität Ulm: Institut für eingebettete Systeme/Echtzeitsysteme, 2020.

[10] Prof. Dr.-Ing. Frank Slomka. *Vorlesung Architektur eingebetteter Systeme*. Sommersemster 2020.