Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

René Büscher

Architektur zur Unterstützung von Redundanzkonzepten auf FPGAs mittels Laufzeitrekonfiguration

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

René Büscher

Architektur zur Unterstützung von Redundanzkonzepten auf FPGAs mittels Laufzeitrekonfiguration

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Thomas Lehmann
Zweitgutachter: Prof. Dr. Michael Schäfers

Eingereicht am: 31. August 2017

René Büscher

Thema der Arbeit
Architektur zur Unterstützung von Redundanzkonzepten auf FPGAs mittels Laufzeitrekonfiguration

Stichworte
FPGA, Partiell, Rekonfiguration, DPR, Sicherheit, Zuverlässigkeit, UAV, Infrastruktur, TMR, Architecture, FMEA

Kurzzusammenfassung
Ein Ansatz zur Erhöhung der Sicherheit kleiner UAV ist die Verwendung von klassischen Redundanztechniken, die sich bereits in Luftfahrt- und Raumfahrtsystemen etabliert haben, beispielsweise die TMR. Um die TMR in kleine UAVs kosten- und platzsparend zu integrieren, kann die Flexibilität moderner FPGAs genutzt werden. Jedes Modul und der Voter wird dazu in verschiedene Partitionen des FPGAs verteilt. Mithilfe der partiellen Rekonfiguration kann ein Selbstheilungsprozess integriert werden. Dieser Prozess ermöglicht es, defekte Module zu verschieben, ersetzen oder neu zu laden. Dieses Sicherheitskonzept kann die Zuverlässigkeit und Verfügbarkeit des Autopilotensystems für kleine UAVs erhöhen.

René Büscher

Title of the paper
Architecture to Support Redundancy Concepts on FPGAs by means of Runtime Reconfiguration

Keywords
FPGA, Partial, Reconfiguration, DPR, Safety, Reliability, UAV, Infrastructure, TMR, Architecture, FMEA

Abstract
One approach to increase the safety of small UAV is the use of classical redundancy techniques, which are already established in aviation and space systems, for example the TMR. In order to integrate the TMR into small UAVs in a cost- and space-efficient manner, we will use the flexibility of modern FPGAs. Therefore, each module and voter will be distributed in different partitions of the FPGA. The means of partial reconfiguration supports self-healing processes. It can be integrated a self-healing process. This process allows to relocate, replace or reload a faulty module. This safety concept increases the reliability and availability of the autopilot system of the small UAV and hence its safety.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einführung

Die Entwicklung von kleinen Unmanned Aerial Vehicle (UAV) schreitet im Moment rasant voran. UAV werden bereits in verschiedenen Geschäftsbereichen vieler Unternehmen eingesetzt, beispielsweise in der Landwirtschaft, dem Vermessungswesen oder dem Windenergiesektor. Etabliert haben sich in diesem Umfeld verschiedene proprietäre[1] oder freie[2] avionische Steuersysteme (Opensource-Flugcontroller). Der wesentliche Unterschied zwischen beiden Varianten ist die Sicherheit, die Größe und das Gewicht. Um kleine UAV schnell und einfach nutzen zu können, werden häufig Opensource-Flugcontroller verwendet. Die Flugcontroller sind günstig, leicht und kompakt. Es ergibt sich aber ein Sicherheitsproblem, da die Flugcontroller nicht redundant sind und nur über eine primitive Fehlerbehandlung verfügen. Daher ist die Idee entstanden, die Flexibilität von modernen FPGA zu nutzen, um die Sicherheit durch Redundanzen und einer aktiven Fehlerbehandlung zu erhöhen und dabei eine möglichst kompakte und leichte Bauweise zu erreichen. Die Idee beruht auf der dynamisch partiellen Rekonfiguration (DPR), die von modernen FPGA unterstützt wird. Dadurch ist es möglich, einzelne funktionale Module während der Laufzeit durch andere zu ersetzen, entfernen oder verschieben. Mithilfe dieses Ansatzes kann innerhalb eines Chipsatzes *„redundante Hardware“* implementiert werden. Des weiteren können die Rechenressourcen effizienter genutzt werden und an die Mission dynamisch angepasst werden. Vor allem in der Luft- und Raumfahrtechnik gilt ein solcher Ansatz als vielversprechend, wie beispielsweise die Autoren Lüdtke et al. [6] und Sabena et al. [11] beschrieben haben.
Der Schwerpunkt dieser Arbeit widmet sich der Sicherheit, im Sinne von Safety. Die Security soll in dieser Arbeit nicht weiter betrachtet werden. Ziel ist es, mehrere funktionale Module sowie den zugehörigen Voter in verschiedene Partitionen eines FPGA zu verteilen, wodurch ein redundantes Teilsystem entsteht. Dazu muss die notwendige Infrastruktur und das Redundanzkonzept betrachtet und definiert werden.

[1] Zum Beispiel Micropilot[35] oder U-Pilot[38]
[2] Zum Beispiel Pixhawk[37] oder PaparazziUAV[36]

1.1 Zielsetzung

Um die notwendigen Ziele abzuleiten, werden zunächst die bestehenden (Sicherheits-)Probleme analysiert, die in verschiedenen avionischen Systemen für UAV vorhanden sind. Die Analyse stellt lediglich eine oberflächliche Recherche dar und kann unvollständig sein. Die Daten wurden aus den frei verfügbaren Informationen und Datenblättern der Hersteller ermittelt.
Wie bereits erwähnt, kann die Entwicklung avionischer Systeme für UAV in zwei Teilbereiche gegliedert werden: Opensource-Projekte[1] und proprietäre Systeme[2]. Die Entwickler der Openesource-Projekte bieten Hard- und Softwarelösungen an. Die angebotenen Lösungen sind nur unzureichend sicher und nicht redundant aufgebaut. Die Architekturen der Projekte sehen nicht den Einsatz von Redundanzen vor. Es existieren keine Routinen, die eine Fehlererkennung ermöglichen und dessen Behebung steuern oder einen Fehler isolieren. Die Opensource-Projekte werden zumeist im Hobbybereich oder der Wissenschaft verwendet. Der Schwerpunkt liegt nicht auf einer sicheren Architektur, sondern auf eine schnelle Integration.
Die proprietären Lösungen werden beispielsweise in UAV für den militärischen oder industriellen Einsatz verwendet. Die angebotenen Lösungen bestehen aus Hard- und Software und sind redundant ausgelegt. Das bedeutet, dass mindestens zwei Systeme gleicher Art vorhanden sind. Laut Hersteller sind Routinen zur Fehlererkennung und -behebung implementiert worden. Jedoch ist der Preis und das Gewicht der Systeme sehr hoch. Beispielsweise wiegt *„MP21283X Triple Redundant“* von Micropilot insgesamt 860g (ohne Sensorik) bei einer Größe von 227x127x54mm [28]. Der Anschaffungspreis liegt bei geschätzten 20000 USD. Das System von Micropilot wurde nicht für kleine UAV entwickelt. Die Kosten, die Systemgröße und das Gewicht übersteigen das Fassungsvermögen der meisten kleinen UAV. Durch die Verwendung eines proprietären Systems würde die Flugdauer und Nutzlast rapide sinken.
Zu der Systemanalyse können zunächst generische Ziele abgeleitet werden. Diese Ziele sollen nicht mit dieser Arbeit gelöst werden, stellen aber die grundlegenden Problematiken avionischer Systeme für kleine UAV dar:

- Hohe Ausfallsicherheit und Zuverlässigkeit
- Fehlererkennung/-behebung
- Integration eines Selbstheilungsprozesses
- Miniaturisierung und Gewichtsreduktion
- Hohe Flexibilität und Energieeffizienz

Wie bereits erwähnt, soll die Flexibilität moderner FPGA verwendet werden. Moderne FPGA können zur Laufzeit definierte Partitionen neu konfigurieren. Dies geschieht mithilfe der

DPR. Dadurch können einzelne Module innerhalb einer Partition getauscht, verschoben oder entfernt werden. Die so freigewordenen Rechenkapazitäten können von einem anderen Modul genutzt werden. Durch eine effiziente Nutzung der Rechenkapazitäten können innerhalb eines FPGA sicherheitsrelevante Module mehrfach geladen werden, wodurch bereits ein einziger FPGA die notwendigen Redundanzen beinhalten kann. Durch eine Fehlererkennung können die fehlerhaften Module isoliert und ausgetauscht oder entfernt werden. Auf diese Art können zufällige Fehler abgefangen werden, jedoch keine systematischen Fehler, da diese bereits zur Designphase vorhanden sind. Das Ziel dieser Arbeit ist es, zum Einen eine sichere Infrastruktur bereitzustellen, welche es ermöglicht fehlerhafte Komponenten zu erkennen und entsprechende Gegenmaßnahmen einzuleiten. Zum Anderen muss die Infrastruktur eine sichere Rekonfiguration ermöglichen, da die Rekonfiguration eine zentrale Maßnahme zur Fehlerbehebung darstellt. Des Weiteren muss die klassische Variante der TMR betrachtet und angepasst werden, da die Variante nicht für eine Rekonfiguration entworfen wurde. Die Abbildung 1.1 zeigt eine Ideenskizze eines möglichen Systemaufbaus. Dabei kontrolliert das *"Configuration Management"* die Rekonfiguration und entscheidet welche Gegenmaßnahme eingeleitet werden muss. Die *"Failure Detection"* erkennt verschiedene Fehler und meldet diese dem *"Configuration Management"*. Der *"Voter"* und die *"Module"* stellen die klassische TMR dar. Alle Daten werden über einen Bus übertragen.

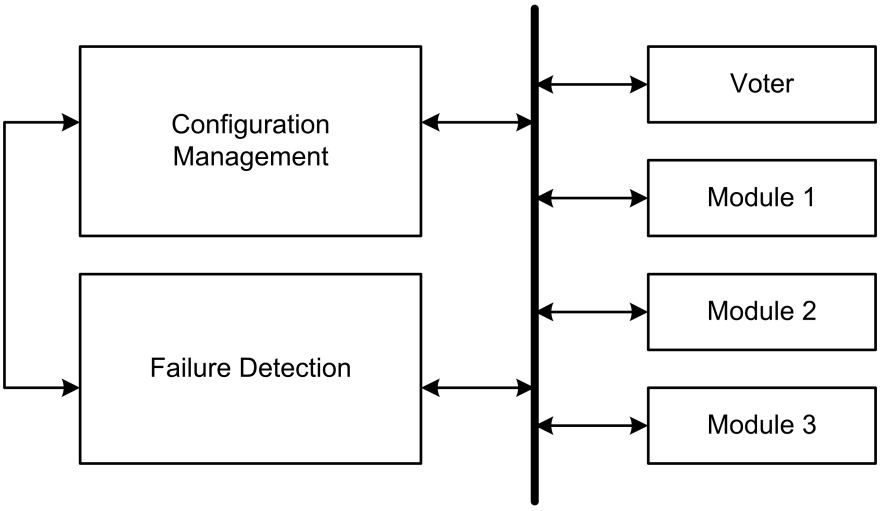

Abbildung 1.1: Ideenskizze eines möglichen Systemaufbaus, dabei ist links die Systemsteuerung zu sehen und rechts die TMR.

1.2 Arbeitsumfeld

Das Projekt Airborne Embedded System (AES) entstand Anfang 2013. Zentrales Ziel war die Entwicklung einer autonom fliegenden Drohne, auch UAV genannt. Die Drohne soll später beispielsweise für den Katastrophenschutz eingesetzt werden. In ferner Zukunft soll es möglich sein, den Flugroboter für einen Flug in Deutschland zuzulassen zu können. Daher ist eines der Kernthemen die Zuverlässigkeit und Sicherheit. Das erste Flugmodell für das System wurde ursprünglich vom Departement Fahrzeugtechnik und Flugzeugbau der HAW Hamburg bereitgestellt. Im diesem Departement war das Project BWB AC20.30 ansässig, welches für den Träger verantwortlich ist. Das BWB-Team forscht an neuen Flugzeugformen und dazu benötigte das Projekt-Team ein neues Messsystem, für den experimentellen Träger, um aerodynamische Messflüge durchführen zu können. Da sich die Interessen der Teams überschnitten, nutzen beide Teams dies, um sich gegenseitig zu unterstützen und einen Wissenstransfer zu schaffen. Seit 2015 haben sich die Wege der beiden Projektteams getrennt und das PO AES entwickelt, in Zusammenarbeit mit der *"Forschungsgruppe Optronik und Avionik"* das bestehende avionische System und die Simulationsumgebung stetig weiter. Die verschiedenen Systeme kommen dabei in verschiedenen Teilprojekten der Forschungsgruppe zum Einsatz. Zudem wurden mehrere eigene Modelle in Form von Starrflüglern und Multikoptern angeschafft, um auch Tests unter realen Bedingungen durchführen zu können. Teilbereiche der aktuellen Forschung sind Antikollisionssysteme, FPGA-Flugcontroller, die Erweiterung der Simulationsumgebung und die Integration von CANaerospace[3] als generisches Kommunikationsprotokoll für UAV.

1.3 Aufbau der Arbeit

Diese Arbeit ist in mehrere Kapitel gegliedert. Folgende Auflistung gibt einen Überblick über die Arbeit und die darin enthaltenen Kapitel. Dem Anhang kann der Inhalt und die Struktur der beigelegten CD-ROM sowie die Auflistung der verwendeten Programme und dessen Versionen entnommen werden.

Kapitel 1 gibt eine kurze Einführung in die Thematik und zeigt die Motivation und Zielsetzung der Arbeit.
Kapitel 2 beschreibt die grundlegenden Begriffe und technischen Grundlagen der Arbeit. Zudem wird in diesem Kapitel eine Einführung in die Sicherheitsanalysen gegeben.
Kapitel 3 beschreibt und diskutiert die Ansätze aus den vergleichbaren Arbeiten.

[3] Protokoll, welches auf dem Controller Area Network (CAN) aufgesetzt wird und speziell für die Luftfahrt entworfen wurde [12].

Kapitel 4 erläutert den zugrunde liegenden Systementwurf und beschreibt die Schnittstellen und das Kommunikationsprotokoll.
Kapitel 5 analysiert den vorgeschlagenen Systementwurf mithilfe einer Sicherheitsanalyse. Des Weiteren werden in diesem Abschnitt die Ergebnisse der Analyse diskutiert.
Kapitel 6 beschreibt den angepassten Systementwurf auf Basis der vorhergegangenen Analyse.
Kapitel 7 erläutert die Implementierungsschritte und beschreibt alle Systemmodule im Detail. Zudem werden in diesem Abschnitt die Simulations- und Testergebnisse beschrieben und diskutiert.
Kapitel 8 fasst die Resultate dieser Arbeit zusammen und zeigt im Ausblick die möglichen Weiterentwicklungen.

2 Grundlagen

Dieses Kapitel beschreibt die Grundlagenm die für den weiteren Verlauf der Arbeit wichtig sind. Zunächst werden verschiedene Redundanzkonzepte betrachtet und dessen Funktionalität beschrieben. Im Anschluss daran werden die Möglichkeiten beschrieben, mit welchen man einen FPGA von Xilinx der Artix-7 Familie (re-)konfiguriert. Des Weiteren wird ausführlich die partielle Rekonfiguration und der Zugriff auf die FPGA-Fabrik mittels Internal Configuration Access Port (ICAP) beschrieben, sowie der Entwicklungsworkflow von Xilinx Vivado für die partiellen Rekonfiguration. Zusätzlich dazu werden im letzten Abschnitt die etabliertesten Sicherheitsanalysen beschrieben, da die im weiteren Verlauf der Arbeit benötigt werden.
Für diese Arbeit wird nur der Artix-7[4] von Xilinx betrachtet, da dieser Chipsatz an der Hochschule verwendet wird. Als Software zur Synthese der Very High Speed Integrated Circuit Hardware Description Language (VHDL)-Dateien wurde Xilinx Vivado[5] in der Version 2016.1 gewählt.

2.1 Sicherheitskonzept

In diesem Abschnitt werden relevante Redundanzkonzepte präsentiert und verschiedene Verfahren erläutert. Ziel soll es sein einen Überblick über die Konzepte zu geben, da diese in der Arbeit immer wieder verwendet werden. Dabei wird im Detail ein Blick auf die Dual Modular Redundancy (DMR), Triple Modular Redundancy (TMR) und N-Modular Redundancy (NMR) geworfen. Außerdem werden die Standy-Verfahren erklärt, da auch diese häufiger in der Arbeit verwendet werden.

2.1.1 Redundanzkonzepte

Die hier beschriebenen Redundanzkonzepte basieren auf der Beschreibung von Storey[13]. Der Abbildung 2.1 können dabei die drei beschriebenen Varianten entnommen werden: *"A stellt die*

[4] Informationen zum FPGA: https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html.
[5] Informationen zur Software: https://www.xilinx.com/products/design-tools/vivado.html. In der Version 2016.2 wurde der Entwicklungsprozess der partiellen Rekonfiguration geändert. Aus Kompatibilitätsgründen wird in dieser Arbeit noch mit der Version 2016.1 gearbeitet.

DMR", *"B stellt die TMR"* und *"C stellt die NMR"* dar. Dabei münden die Ausgangssignale der einzelnen Module in einem Voter. Bei der TMR vergleicht der Voter die Eingangsdaten und kann anhand der Signale entscheiden, ob und welches Modul nicht ordnungsgemäß funktioniert. Wenn jedoch alle Module unterschiedliche Daten liefern, kann nur ein Fehlverhalten ermittelt werden, aber nicht spezifiziert werden, welches Module fehlerhaft ist. Die Funktionalität ist bei der DMR identisch, nur dass bei diesem Aufbau nie ermittelt werden kann, welches Modul fehlerhaft ist. Die NMR stellt eine Erweiterung mit mehr als drei Modulen dar. Dabei ist diese Variante aber nicht zwingend sicherer. Auch hier kann es unter verschiedenen Umständen dazu kommen, das nicht genau gesagt werden kann, welches Modul fehlerhaft ist und eben welches nicht.

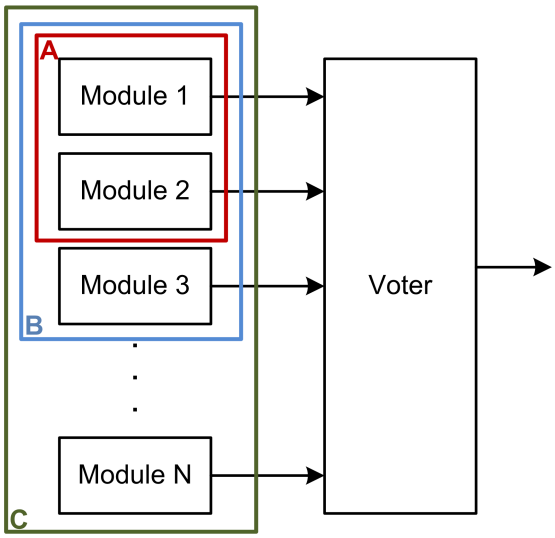

Abbildung 2.1: Schematische Darstellung der verschiedenen Variationen von Redundanzkonzepten: (A) DMR, (B) TMR, (C) NMR.

2.1.2 Standby-Verfahren

Die hier beschriebenen Verfahren basieren auf der Beschreibung von Storey[13]. Dabei werden zentral zwei Varianten unterschieden: Hot-Standby und Cold-Standby. Bei einer Variante, die Hot-Standby verwendet, ist das redundante Modul „passiv aktiv“ und kalkuliert beispielsweise durchgehend die Berechnungen mit. Dadurch kann dieses Modul zu jederzeit und ohne größere Pausen die Arbeit des aktiven Moduls übernehmen und ersetzen. Diese Variante ist aus Sicht der Implementierung einfach umzusetzen, verbraucht aber durchgehend Energie und erzeugt Abwärme. Beim Cold-Standby ist das Modul passiv und muss beim Erkennen eines

Fehlverhalten zunächst geweckt und eventuell synchronisiert werden. Daher ist das Modul nicht sofort einsatzbereit. Jedoch wird dieses Modul kaum Energie benötigen und Abwärme erzeugen, bis zum Zeitpunkt dessen Nutzung. Je nach Priorisierung und Aufgabe des Moduls muss der Entwickler entscheiden, welche der beiden Varianten die richtige ist.

2.2 FPGA (Re-)Konfiguration

Ein FPGA bietet dem Benutzer verschiedene Möglichkeiten diesen zu konfigurieren. Es existieren zwei unterschiedliche Konfigurationsdatenpfade: Der serielle Datenpfad und der parallele Datenpfad. Außerdem wird zwischen dem Master- und Slave-Konfigurationsmodus unterschieden. Der Unterschied zwischen den Datenpfaden stellt letztlich die Art der Datenübertragung dar. Beim seriellen Datenpfad werden die Konfigurationsdaten seriell übertragen und beim parallelen Datenpfad parallel. Beispiele für die serielle Übertragung ist die Serial Peripheral Interface (SPI)- oder auch die Joint Test Action Group (JTAG)-Schnittstelle. Für den parallelen Datenpfad werden beispielsweise der ICAP oder Quad SPI als Schnittstelle verwendet. Im Master-Konfigurationsmodus steuert der FPGA selbstständig seine (Re-)Konfiguration, beispielsweise beim automatischen Laden eines Bitfiles aus einem Flash-Speicher oder von einer SD-Karte. Im Slave-Modus wird der FPGA durch ein anderes System (re-)konfiguriert, beispielsweise durch einen Mikroprozessor oder eine Workstation. Dabei ist die Wahl der Schnittstelle und somit des Datenpfades so gut wie keine Grenze gesetzt. Zumeist wird ein FPGA über den seriellen Datenpfad mithilfe von JTAG programmiert. Der FPGA befindet sich dabei im Slave-Konfigurationsmodus.
Nach dem erfolgreichen Programmieren kann zusätzlich die Konfigurationsdatei in einem Flash-Speicher abgelegt werden, wodurch der FPGA nach dem Einschalten der Spannung die Möglichkeit hat, die Daten im Master-Konfigurationsmodus selbständig über einen seriellen Datenpfad zu laden und sich selber zu konfigurieren. Da es durch die Zieldefinitionen notwendig ist, dass innerhalb des FPGA eine eigene Rekonfigurationsroutine implementiert wird, benötigen wir den Zugriff auf die FPGA-Fabrik. Dies gelingt mithilfe des ICAP, welcher im Abschnitt 2.5 ausführlich beschrieben wird.
Bisher wurden die physikalischen Zugriffsmöglichkeiten beschrieben. Für den Zugriff wird ein bestimmtes Protokoll verwendet. Das am weitesten verbreitete, ist das SelectMAP-Protokoll, welches auch beim ICAP verwendet wird. Das SelectMAP-Protokoll nutzt den parallelen Datenpfad. Das Protokoll kommt sowohl im Master- als auch im Slave-Konfigurationsmodus zum Einsatz. Für den bidirektionalen Datenpfad des ICAP kann zwischen einem 8-, 16- oder 32-bit breiten Datenpfad gewählt werden. Der ICAP kann zur Konfiguration, für ein Readback der

Konfigurationsdaten, zum Auslesen der Statusregister oder dem Übertragen von Kommandos an den FPGA verwendet werden. Die Abbildung 2.2 zeigt dabei beispielhaft das Senden von mehreren Datenbytes. Die Steuerung des Ablaufs erfolgt über einzelne Steuerleitungen und

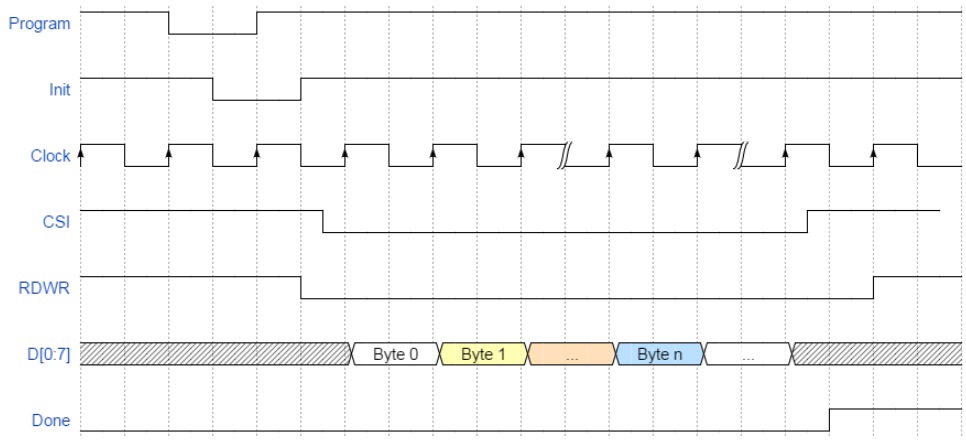

Abbildung 2.2: Darstellung des Signalablauf des SelectMAP-Protokoll für einen lesenden Zugriff auf ein FPGA-Register [23, S. 46].

Kommandos, die über den Datenpfad übertragen werden. Je nach gewählter Datenbreite kann mit der steigenden Flanke ein Byte, Halbwort oder Wort gelesen oder geschrieben werden. Beim Lesen und Schreiben muss die Bitanordnung beachtet werden. Das Interface erwartet die Daten in jedem Byteblock im Little-Endian Format, das bedeutet, dass das kleinstwertigste Bit am Anfang stehen muss (LSB zu MSB). Das Lesen (*"RDWR=1"*) und Schreiben (*"RDWR=0"*) kann jederzeit durch eine Signaländerung auf eine logische *"0"* auf der *"CSI"*-Leitung unterbrochen werden. Die Konfigurationsverfahren von Xilinx FPGAs werden in den Handbüchern [23] und der Arbeit von Russek [10] ausführlich beschrieben.

2.3 Partielle Rekonfiguration

Die partielle Rekonfiguration stellt eine Spezialform der Konfigurationverfahren eines FPGA dar. Dadurch ist es möglich, Partitionen eines FPGA während der Laufzeit zu rekonfigurieren, ohne dass andere Partitionen des FPGA davon betroffen sind oder ihre Funktionalität einschränken müssen. Dies gilt nur, wenn zwischen den einzelnen Partitionen, keine Interaktionen vorgesehen sind. Normalerweise ist ein FPGA nach der Programmierung statisch und während der Laufzeit unveränderbar. Die Abbildung 2.3 zeigt dabei schematisch den Aufbau der statischen und dynamischen Partitionen. Durch das Verwenden der partiellen Rekonfigu-

rationen wird der FPGA zunächst in statische, in der Abbildung als *"FPGA"* bezeichnet, und dynamische Partitionen, in der Abbildung als *"Reconfigurable Partition Block A"* bezeichnet, aufgeteilt. Der Inhalt der dynamischen Partitionen wird durch verschiedene Module, in der Abbildung als *"A1-A4.bit"* beschrieben, dargestellt. Dabei wird jedes Modul durch eine Konfigurationsdatei repräsentiert. Die Funktionalität einer rekonfigurierbaren Partition wird durch

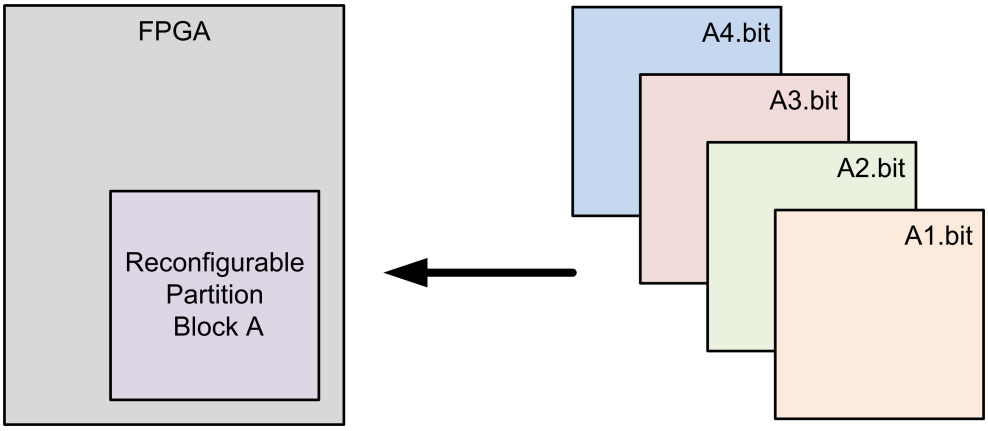

Abbildung 2.3: Grafische Darstellung des Aufbaus der statischen und dynamischen Partitionen [23, S. 8].

ein oder mehrere rekonfigurierbare Module beschrieben. Jede Partition besitzt eine begrenzte Anzahl an Ressourcen. Die Größe der Partition wird während des Entwicklungszyklus definiert. Die einzelnen Module enthalten die konkrete Funktionalität. Alle Module, die in der selben Partition untergebracht werden sollen, benötigen die selbe Schnittstelle. Jede Partition erhält während der Synthese und Implementierung automatisch sogenannte Partition-Pins. Diese Pins stellen die Schnittstelle zwischen der statischen und der rekonfigurierbaren Logik dar. Innerhalb einer rekonfigurierbaren Partition können nur Lookup-Table (LUT), Flip-Flop (FF), Block Random Access Memory (BRAM) und Digitaler Signalprozessor (DSP) verwendet werden. Andere dedizierte Blöcke, wie beispielsweise ein Mixed-Mode Clock Manager (MMCM), können nicht in diesen speziellen Partitionen implementiert werden und müssen in die statische Logik ausgelagert werden. Zur Rekonfiguration kann beispielsweise der ICAP verwendet werden, aber auch via JTAG können einzelne Partitionen rekonfiguriert werden. Die partielle Rekonfiguration von Xilinx FPGA werden im Handbuch [23] und [17] ausführlich beschrieben.

2.4 Xilinx Vivado Workflow

Um die partielle Rekonfiguration in den Entwicklungsprozess zu integrieren, muss ein angepasster Workflow von Vivado verwendet werden. Zum Verständnis werden zunächst die verschiedenen Projektverwaltungsvarianten beschrieben. Vivado unterstützt zwei verschiedene Varianten: den Project-Mode und Non-Project-Mode. Diese basieren auf der Arbeitsweise von Viavado. Alle Daten die Vivado zum Bearbeiten eines Projektes benötigt, werden in den Arbeitsspeicher geladen und dort vorgehalten. Daher sind die Daten nur flüchtig vorhanden und müssen in regelmäßigen Abständen auf der Festplatte gespeichert werden. Im Project-Mode erfolgt das Sichern nach jedem erfolgreichem Projektschritt vollautomatisch. Dazu wird nach jedem Schritt ein Checkpoint erstellt, der die letzten Ergebnisse beinhaltet sowie den aktuellen Projektstand. Zudem werden im Project-Mode alle Einstellungen in einer Projektdatei gespeichert und jedes Projekt wird mit einer allgemeinen Standarteinstellung geladen, welche für die meisten Hardwareprojekte vollkommen ausreichend ist. Auch die Abhängigkeiten zwischen den einzelnen HDL-Dateien werden vollautomatisch erzeugt und verwaltet. Im Non-Project-Mode wiederum werden weder Checkpoints automatisch erzeugt, noch die Abhängigkeiten zwischen den HDL-Dateien automatisch verknüpft. Alle Einstellungen müssen vom Entwickler händisch vorgenommen werden. Dazu wird der Entwickler dazu angehalten, eigene Tool Command Language (TCL)-Skripte zu entwerfen, welche die Tätigkeiten übernehmen, die im Project-Mode bereits vorhanden sind. Dadurch ist dieser Modus wesentlich aufwändiger in der Handhabung, bietet aber einen sehr hohen Freiheitsgrad bei der Entwicklung. Unabhängig vom Modus basieren alle Schritte die Vivado durchführt auf TCL-Skripte. Dabei greift Vivado im Project-Mode auf einen stark simplifizierten Befehlssatz zurück. Dieser verwendet bereits die vorgeschlagenen Parameter von Xilinx. Dieser simple Befehlssatz basiert im Grunde auf den umfangreicheren Befehlssatz des Non-Project-Mode. Jedoch muss in diesem Modus jeder Parameter selber festgelegt werden. Mehr Informationen zum Thema TCL und Vivado kann im Handbuch [18] gefunden werden. Mehr Informationen zu den einzelnen Modes kann in den Handbüchern [20] und [22] nachgelesen werden.
Um die partielle Rekonfiguration im Projekt verwenden zu können, muss in Xilinx Vivado 2016.1 der Non-Project-Mode verwendet werden. Die Abbildung 2.4 zeigt den Ablauf zur Erstellung der verschiedenen Bitfiles für die statischen und dynamischen Partitionen. Zunächst müssen alle Module, sowohl die statischen als auch die dynamischen, unabhängig von einander synthetisiert werden. Das Ergebnis ist eine Netlist jedes einzelnen Moduls. Die statische Netlist wird nun zusammen mit den rekonfigurierbaren Partition und einem Modul, das der Partition zugeordnet ist, verknüpft. Das so entstandene Design wird implementiert (Mapping & Routing).

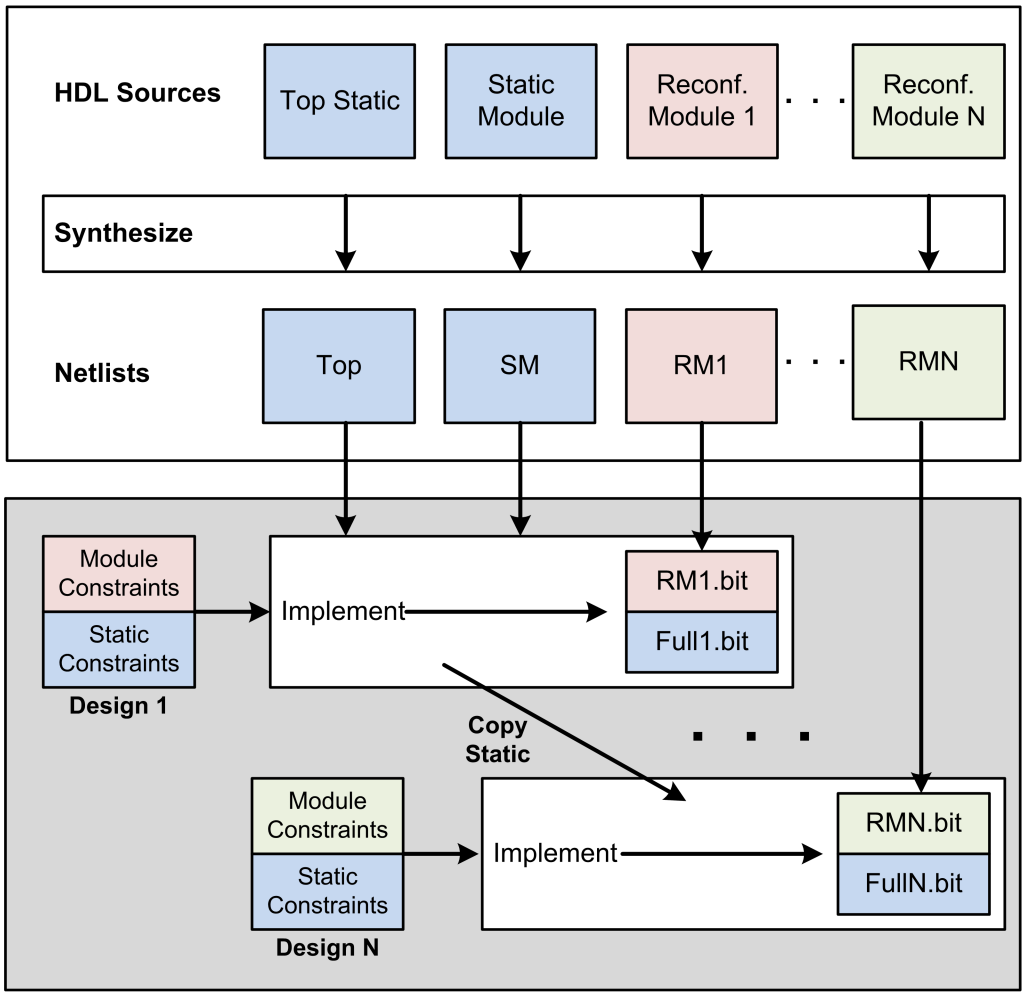

Abbildung 2.4: Schematische Darstellung des Workflows zur Generierung der Konfigurationsdaten für die statischen und dynamischen Partitionen [17, S. 19].

Daraus entsteht jeweils ein Bitfile für die statische Partition, welches die jeweiligen Partitionen beinhaltet und die einzelnen Bitfiles für jeweils ein Modul je Partition. Das bereits implementierte statische Modul wird kopiert und mit dem nächsten Modulen der Partition verknüpft. Die verknüpften Daten werden wieder implementiert. Als Ergebnis liegen entsprechende Bitfiles vor. Dieser Schritt wird solange wiederholt, bis alle rekonfigurierbaren Module implementiert und die zugehörigen Bitfiles erzeugt wurden. Bei der Implementierung sollte immer mit dem umfangreichsten Modul begonnen werden, da das Modul im Normalfall die größte Anzahl an Hardware inferiert. Dadurch kann sichergestellt werden, dass die Partition-Pins für das größte Modul passend instantiiert werden. Dadurch stellt man frühzeitig fest, ob es Platz- oder Timingprobleme innerhalb der definierten Partition gibt. Weitere Informationen zum Workflow der partiellen Rekonfiguration von Vivado kann den Handbüchern [17] und [24] entnommen werden.

2.5 Internal Configuration Access Port

Der ICAP wurde bereits mehrfach erwähnt und wird in diesem Abschnitt ausführlich behandelt. Der Port stellt eine dedezierte Komponente eines Xilinx FPGA dar und ist als Hardwaremakro innerhalb von VHDL oder Verilog zugänglich. Xilinx als Hersteller ist der Schnittstelle gegenüber verschwiegen und es ist mühselig Informationen zu erhalten. Bekannt ist, dass die Schnittstelle das SelectMAP-Protokoll verwendet. Dazu existieren eine Reihe von wichtigen Dokumenten: [23], [19] und [21]. Somit ist bekannt, wie die Schnittstelle genutzt werden kann und wie das Protokoll funktioniert. Das SelectMAP-Protokoll wurde bereits im Kapitel 2.2 ausführlich beschrieben. Unbekannt hingegen ist der konkrete Befehlssatz, mit denen beispielsweise ein FPGA-Register ausgelesen werden kann oder eine Rekonfiguration gestartet wird. Für diese Informationen wurden zum einen die Simulationsdateien der UNISIM-Bibliothek von Xilinx (UNISIM.vcomponents.all) herangezogen und zum anderen verschiedene Foreneinträge der Community betrachtet: [33], [34], [32], [31], [29] und [30]. Mithilfe der zusätzlichen Informationen war es möglich, die Befehlsfolge für das Auslesen der gerätespezifischen Identifikationsnummer zu extrahieren und zu testen. Der Befehlsablauf und der Zustand der Steuerleitungen kann der Tabelle 2.1 entnommen werden. Dadurch war es möglich festzustellen, in welchem Format die Konfigurationsdatei vorliegen muss. Die Daten müssen in einem binären Format vorliegen, da sonst zusätzlich ein Interpreter auf dem FPGA notwendig wäre. Dadurch ergibt sich der Nachteil, dass eine Konfigurationsdatei immer eine feste Größe hat. Die Größe ist abhängig von der Größe der jeweiligen Zielpartition. Zudem konnte beim Betrachten der Konfigurationsdatei mit einem HEX-Editor festgestellt werden, dass es nicht

Befehl(Hex)	Enable (LA)	Read/Write	Erklärung
20000000	High	High	Dummy Word
20000000	High	Low	Dummy Word
AA995566	Low	Low	Sync Word
20000000	Low	Low	Dummy Word
28018001	Low	Low	Read ID Command
20000000	High	Low	Dummy Word
20000000	High	High	Dummy Word
20000000	Low	High	Dummy Word
20000000	Low	High	Dummy Word
20000000	Low	High	Dummy Word
20000000	Low	High	Dummy Word
20000000	Low	High	Dummy Word
20000000	High	High	Dummy Word
20000000	High	Low	Dummy Word
30008001	Low	Low	Desync Send Command
0000000D	Low	Low	State for Desync Register
20000000	High	Low	Dummy Word
20000000	High	High	Dummy Word

Tabelle 2.1: Befehlsablauf zum Auslesen der gerätespezifischen Identifikationsnummer des FPGA über den ICAP, mithilfe des SelectMAP-Protokolls.

mehr nötig ist, vor einer Rekonfiguration Befehle via ICAP an den FPGA zu übertragen, da bereits alle notwendigen Befehle in der Konfigurationsdatei enthalten sind. Ausschließlich die Steuerleitung müssen korrekt angesteuert werden. Dadurch ist es wesentlich simpler, den ICAP zu implementieren, da es nur notwendig ist die übertragenen Rohdaten im richtigen Format weiterzugeben. Zudem ist es möglich die Rekonfiguration mithilfe des *"Enable"*-Signals zu pausieren, wodurch es nicht nötig ist, die Daten sehr schnell an den Port weiterzureichen.

2.6 Sicherheitsanalysen

In diesem Abschnitt werden die wichtigsten Sicherheitsanalysen vorgestellt. Diese dienen als Grundlage für die im späteren Verlauf der Arbeit durchgeführte Sicherheitsanalyse im Kapitel 5. Betrachtet und beschrieben werden im Folgenden die HAZARD-Analyse, Fault Tree Analysis (FTA) und FMEA. Die Beschreibungen der Analysen basieren auf den Büchern von Storey [13] und Bozzano et al. [2].

2.6.1 HAZARD-Analyse

Während der HAZARD-Analyse werden durch ein interdisziplinäres Expertenteam die (Teil-) Systeme betrachtet. Das Team ermittelt die Verbindungen zwischen den Komponenten und prüft die Abhängigkeiten. Dabei werden in einer Diskussion, über die Parameter der Links und Attribute der Komponenten, Leitwörter verwendet, um Gefahren zu identifizieren. Die Leitworte sollen das Team auf zu beachtende Aspekte und Probleme aufmerksam machen. Im Folgenden ein Beispielsatz für das Verständnis: *"Die Sensorspannungsversorgung liefert **keine** Spannung mehr. Der Grund dafür könnte ein Spannungsregulator sein. Die Konsequenz ist ein fehlendes Signal des Sensors."*. Diese Sätze werden tabellarische abgebildet. Ziel ist es, mögliche Abweichungen vom normalen Betrieb zu finden und die aus den Abweichungen resultierenden Gefahren abzuleiten. Es ist nicht das Ziel, alle möglichen Ursachen zu finden, diese sollen durch eine FTA oder FMEA gefunden werden. Das Ergebnis der HAZARD-Analyse wird in einer HAZOP-Studie festgehalten. Ein Beispiel einer solchen Studie kann der Tabelle 2.2 entnommen werden.

2.6.2 Fault Tree Analysis

Die FTA wird zum Finden möglicher Fehler verwendet. Die Analyse beginnt mit dem Ausfall einer Komponente oder eines Systems, dem sogenannten Top Level Event (TLE). Von dem TLE ausgehend, wird versucht, ein Baum bis zu den auslösenden Events aufzuspannen, den

Item	Inter-connec-tion	Attribute	Leitwort	Ursache	Folge	Empfehlung
1	Sensor Versor-gungs-leitung	Spannungs-versorgung	Nein	PSU, Reg-ler oder Kabelbruch	Mangelhaftes Sensor-Signal erkannt, Sys-tem wird herunterge-fahren	
2			Mehr	Reglerfehler	Mögliche Beschädi-gung des Sensors	Überspannungsschutz erwägen
3		Sensor-strom	Mehr	Sensorfehler	Falsche Temperatur gelesen	Monitor muss Strom-versorgung gewähr-leisten
4			Weniger	Sensorfehler	Falsche Temperatur gelesen	Wie oben
5	Sensor-ausgang	Spannung	Nein	PSU, Sen-sor oder Kabelbruch	Mangelhaftes Sensor-Signal erkannt, Sys-tem wird herunterge-fahren	
6			Mehr	Sensorfehler	Temperaturmessung zu hoch? Führt zu einer Verringerung der Anlageneffizienz	Betrachte die Verwen-dung von redundan-ten Sensoren

Tabelle 2.2: Beispielhafte Darstellung einer HAZOP-Studie aus [13, S. 43].

sogenannten Basic Events (BE). Zwischen dem TLE und den BE existieren eine Menge von Intermediate Events (IE). Die IE können aus mehreren BE und/oder IE bestehen. Ein IE wird immer über ein logisches Gater *"AND"* und *"OR"* erzeugt. Problematisch ist die mögliche Größe eines solchen Baumes. Daher sollten die verwendeten Komponenten und der Systemzustand, für eine solche Analyse, klar abgegrenzt und definiert werden. Ein FTA wird meist im späteren Projektverlauf verwendet oder zusätzlich zu anderen Analysenmethoden. In diesem Abschnitt wurde nur das allgemeine Format der Analyse präsentiert. Es existieren inzwischen viele Abwandlungen und Erweiterungen.

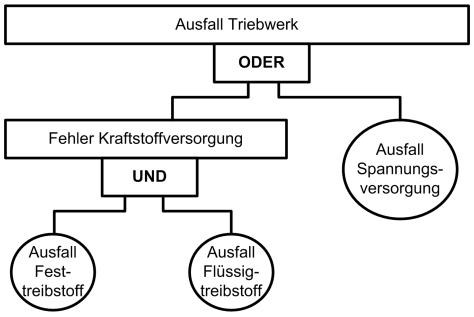

Abbildung 2.5: Beispielhafte Darstellung eines Fehlerbaumes aus [13, S. 65].

2.6.3 Failuremode and Effect Analysis

Die FMEA wird in verschiedenen Phasen eines Projektes eingesetzt, um mögliche Auswirkungen auf die zu analysierenden Komponenten oder (Teil-)Systeme und die angrenzenden Komponenten und (Teil-)Systeme zu finden. Die Durchführung der Analyse verläuft tabellarisch und wird meist durch ein interdisziplinarisches Expertenteam und mindestens einem Moderator durchgeführt. Dabei wird je nach Abgrenzung für eine oder mehrere Komponenten bis zum gesamten System ermittelt, welche Fehler möglicherweise auftreten können. Zu allen ermittelten Fehlern wird festgehalten, welche Ursachen zugrunde liegen. Die Auflistung der Ursachen muss nicht vollständig sein, da die Ursachen ebenfalls von der Betrachtungstiefe abhängig ist. Außerdem werden Effekte, die auf die Umgebung und das System wirken können, in der Tabelle festgehalten. Ein Beispiel für eine FMEA zeigt die Tabelle 2.3.

FMEA für einen Mikroschalter						
Ref.-Nr.	**Komponente**	**Ausfallmodi**	**Mögliche Ursache**	**Lokale Effekte**	**System Effekte**	**Abhilfemaßnahmen**
1	Werkzeugschutzschalter	Leerlaufkontakte	(a) Fehlerhafte Komponente (b) Übermäßiger Strom (c) Extreme Temperatur	Fehler beim Erkennen des Werkzeugschutzes	Verhindert den Einsatz des Werkzeuges? System scheitert Sicher	Schalter für hohe Zuverlässigkeit und geringe Wahrscheinlichkeit eines gefährlichen Ausfalls wählen Starre Qualitätskontrolle bei der Beschaffung von Schaltungen
2		Kurzschlusskontakte	(a) Fehlerhafte Komponente (b) Übermäßiger Strom	System stellt fälschlicherweise fest, dass der Schutz geschlossen ist	Erlaubt das Werkzeug zu verwenden? Gefährliches Versagen	Ändern der Software, um den Fehler zu erkennen und entsprechende Maßnahmen zu ergreifen
3		Übermäßiges prallen des Schalter	(a) Alterungseffekte (b) Längere hohe Ströme	Leichte Verzögerung beim Erkennungszustand des Schutzes	Unerheblich	Sicherstellen, dass Hardware-Design übermäßigen Strom durch Schalter verhindert

Tabelle 2.3: Beispielhafte Darstellung einer FMEA aus [13, S. 38].

3 Relevante Arbeiten

Bevor der Systemvorschlag im Kapitel 4 diskutiert wird und die Problemstellung im Abschnitt 4.1 ausführlich betrachtet wird, wurden verschiedene relevante Arbeiten begutachtet. Ziel dieses Kapitels ist die Diskussion und das Prüfen der Arbeiten auf mögliche Übertragungen in das eigene Konzept. Dabei steht im Vordergrund, dass hier Techniken mit der partiellen Rekonfiguration verwendet wurden, da dies ein zentrales Element der Arbeit darstellt. Dabei werden zunächst aktuelle Redundanzkonzepte betrachtet. Im Anschluss daran werden verschiedene Arbeiten zur Steuerung der partiellen Rekonfiguration herangezogen. Der letzte Abschnitt beschäftigt sich speziell mit der Thematik der Datensynchronität vor, während und nach einer Rekonfiguration.
Die partielle Rekonfiguration und verschiedene Redundanzkonzepte miteinander zu verbinden ist nicht neu. Die Entwicklungen in diesem Feld haben in den letzten Jahren zugenommen. Mögliche Gründe für die Zunahme sind die preiswerten Komponenten und die hohe Ressourcen- und Leistungsverfügbarkeit auf den Chipsätzen. Auch die partielle Rekonfiguration hat sich seitens der Hersteller immer weiterentwickelt und wurde immer besser in den Entwicklungsprozess integriert. Dadurch ist es deutlich einfacher geworden, Entwicklungen in diesem Bereich voranzutreiben. Beispielsweise beschrieben Zhang et al. bereits 2006 [27] in einem Artikel verschiedene Standby-Verfahren, die mittels der partiellen Rekonfiguration umgeschaltet werden konnten. Paulsson et al. [8] stellten 2006 ein weiteres Verfahren vor, mit welchen kontinuierliche Built-In Tests ausgeführt werden können. Dabei wurde der FPGA in mehrere Partitionen unterteilt, in denen die Module ausgeführt werden. Ein Teil der Partitionen wurde statt mit einem funktionalen Modul mit einem Testmodul geladen, welches kontinuierlich verschiedene Testroutinen ausführt. Dadurch sollten permanente Fehler identifiziert werden. Durch die partielle Rekonfiguration war es möglich, einen durchgängingen Wechsel zwischen Built-In-Test und funktionalen Modul zu erreichen.

3.1 Redundanzkonzept

Wie bereits erwähnt, wird zum Einen ein Redundanzkonzept benötigt, welches eine partielle Rekonfiguration unterstzützt. Zum Anderen ist eine Infrastruktur nötig, welche die Fehlereken-

nung durchführt und die Konfiguration durchführen kann. In der ersten Arbeit schlagen die Autoren Vít et al.[16] eine alternative Architektur zur klassischen TMR vor. Die Architektur verwendet eine Checker-Komponente, mithilfe derer Fehler erkannt werden sollen. Das System besteht aus zwei FPGA. Auf den FPGA wird jeweils die identische Implementierung ausgeführt. Jeder FPGA wird in mehrere Partitionen unterteilt. Die einzelnen Partitionen können mithilfe der partiellen Rekonfiguration zur Laufzeit rekonfiguriert werden, wodurch im Fehlerfall nicht der vollständige FPGA konfiguriert werden muss. Die Partitionen jedes FPGA besitzen einen Vorgänger und Nachfolger, wodurch eine geschlossene Kette entsteht. Der Checker wird in mehrere verschiedene Partitionen jedes FPGA implementiert, sodass jeder Checker zwischen verschiedenen funktionalen Modulen eingehangen wird. Jeder Datenstrang wird durch zwei separate Checker geprüft. Durch diesen Aufbau ist es möglich, eine vergleichbar hohe Zuverlässigkeit zu erreichen, wie es bei der TMR möglich wäre, jedoch mit einem geringeren Ressourcenaufwand. Ein Nachteil des Aufbaus ist es, dass nicht eindeutig festgestellt werden kann, welches funktionale Modul fehlerhaft ist, sondern nur, dass eines der Module zwischen zwei Checkern fehlerhaft ist. Daher ist es dann notwendig alle Module neu zu konfigurieren und darauf zu hoffen, dass der Fehler nicht wieder auftritt. Außerdem wird durch die Verkettung bereits die Architektur vorgegeben, wodurch sich dieser Ansatz nicht generisch für alle Systeme nutzen lässt.

Einen weiteren Ansatz liefert die Arbeit von Straka et al.[14]. In dieser Arbeit verwenden die Autoren die TMR. Dabei wird jedes Modul innerhalb einer dynamische Partition geladen. Der Voter wird in einer statischen Partition des FPGA geladen. Die Fehler werden mittels Checker erkannt. Dieser übermittelt einen Bericht an den *"Reconfiguration Manager"*. Der Manager kann eine Rekonfiguration einleiten. Das System der Autoren besteht aus drei Partitionen, in denen jeweils ein Modul geladen werden kann. Der Voter entscheidet auf Basis des Checker-Berichts, welche Daten über die Schnittstelle aus dem System transportiert werden. Für andere Komponenten des Systems ist die Rekonfiguration und ein Fehlverhalten transparent, da der Manager, Voter und dessen Module als eine geschlossene Komponente nach außen agieren. Dadurch mangelt es dem Ansatz jedoch auch an Flexibilität, da die Module immer nur in der selben Partition geladen werden können. Das bedeutet, dass es keine Möglichkeit gibt, die Partition beispielsweise anders zu verwenden. Ein permanenter Fehler innerhalb einer Partition stellt für diesen Ansatz ein großes Problem dar, denn ein Verschieben des Moduls ist nicht möglich.

Die Autoren Al-Haddad et al. [1] beschreiben in einem Artikel ein System zur adaptiven

Redundanz. Dabei verwenden die Autoren die Selbstorganisation aus dem Feld *"Organic Computing"*. Das Ziel von *"Organic Computing"* ist es, ein System zu entwerfen, welches eine Selbstkonfiguration, -organisation und -heilung ermöglicht. Die Systeme wurden durch die Biologie inspiriert und sollen in der Arbeit mithilfe eines FPGA implementiert werden. Als Redundanzkonzept ist das Reconfigurable Adaptive Redundancy System interessant. Diese Komponente besteht aus maximal drei funktionalen Modulen. Die Ausgabedaten aller Module werden an das Autonomic Element (AE) weitergegeben. Dieses Element prüft die Ausgaben und es ist dem Element möglich, jedes einzelne funktionale Modul zu aktivieren und deaktivieren. Dazu wird der Redundancy Controller verwendet, welcher Bestandteil des AE ist. Dadurch ist es möglich, zwischen einem funktionalen Modul (keine Redundanz), über die DMR bis hin zur TMR alle Variationen abzubilden. Dem Redundancy Controller ist es möglich, zur Laufzeit zwischen den verschiedenen Varianten zu wechseln und im Fehlerfall einzelne Module neuzuladen, beziehungsweise auszutauschen. Der Komponente ist bekannt, wann eine Rekonfiguration durchgeführt wird, wodurch die anderen Module darauf reagieren können und die Rekonfiguration keinen negativen Einfluss auf das Verhalten ausüben kann. Die Autoren schlagen einen Hybrid-Modus vor. Dabei wird die Redundanz regelmäßig geändert, um Energie zu sparen. Dieser Ansatz ist sehr interessant, vor allem die Möglichkeit die Redundanzen in einer Art Hybrid-System zu betreiben. Jedoch ist auch dieser Ansatz kaum flexibel, da die Partitionen ausschließlich für die selben Module vorgesehen sind. Zudem wird die Rekonfiguration durch eine externe Workstation eingeleitet und gesteuert, wodurch der mögliche Einsatz weiter einschränkt wird.

Die Autoren Dumitriu et al.[3] verwenden keine Variante der NMR, sondern einen ganz anderen Ansatz. Dessen Architektur besteht aus mehreren Slots, welche alle unabhängig geladen werden können. Jeder Slot ist an einen Bus angeschlossen, wodurch eine N:M-Kommunikation erreicht werden kann. Jedes Modul beinhaltet einen Built-In Self-Test (BIST). Mithilfe dessen kann festgestellt werden, ob innerhalb des Moduls Fehler auftreten. Jedes Modul besitzt zudem ein Control- und Report-Interface. Durch die Schnittstellen ist es möglich, die Rekonfiguration eines Slots, beziehungsweise das Verschieben in einen anderen Slot, einzuleiten. Ob eine Rekonfiguration notwendig ist, wird durch ein externes System auf Basis des BIST-Reports entschieden. Alle Informationen werden über das Report-Interface verteilt, sodass auch andere Komponenten über den Status informiert werden. Durch den Bus werden die Single-Point-of-Failure (SPF) stark reduziert und die Flexibilität des Systems erhöht. Jedoch reicht ein BIST nicht aus, um alle Fehlervarianten richtig identifizieren zu können. Vor allem weil alle anderen Komponenten, welche beispielsweise für die Steuerung verantwortlich sind, nicht vor einem

Ausfall geschützt sind. Außerdem ist die gesamte Steuerung nur extern verfügbar, auf einer separaten Workstation, wodurch der Einsatz stark beschränkt wird.

In der letzten Arbeit von den Autoren Eapen et al.[4] wurde ein detaillierter Blick auf die Anordnung und Ausdehnung von Partition geworfen. Dazu wurden mehrere Partition definiert. Jede Partition hat eine andere physikalische Größe und Ausdehnung. Ziel dabei ist es bei, einem Ausfall eines Moduls dieses in eine andere Partition zu verschieben. Dabei soll im Idealfall die Partition gewählt werden, welche eine minimale Anzahl von Ressourcen zur Verfügung stellt, jedoch ausreichend um das Modul fehlerfrei auszuführen. Dadurch sollte bei einem Ausfall eines größeren Moduls eine Partitionen mit ausreichend Ressourcen vorhanden sein. Es wird also versucht, die Anzahl der nicht mehr nutzbaren Ressourcen zu minimieren und den FPGA am effizientesten zu nutzen. Dies stellt eine sehr interessante Herangehensweise dar, da es hierdurch möglich ist, die begrenzten Ressourcen eines FPGA möglichst effizient und schonnend zu nutzen. Jedoch kann es auch hierbei möglich sein, das bestimmte Partitionen nicht mehr genutzt werden können, wenn andere Module zu groß sind.

3.2 Steuerung

Ein wichtiger Aspekt für die Infrastruktur ist die Steuerung der partiellen Rekonfiguration, sowie die Systemüberwachung. Dabei muss die Infrastruktur ebenfalls entscheiden, wann und vor allem welche Gegenmaßnahemen eingeleitet werden muss. Dabei spielt auch das Scheduling der Rekonfiguration eine zentrale Rolle, da beispielsweise entschieden werden muss welche Partition oder welches Modul zuerst konfiguriert wird. Daher wurden insgesamt zwei Arbeiten ausgewertet, die sich am Rande mit dieser Thematik im breiten Feld der partiellen Rekonfiguration auseinandersetzen.

Dabei wird ein vielversprechender Ansatz von Lüdtke et al.[6] verfolgt, um die Steuerung der Rekonfiguration zu realisieren. Der gesamte Aufbau des Systems ist dezentral, in einzelne Nodes organisiert. Jede Node erhält mindestens eine Rolle: Worker, Master, Observer. Für die Steuerung sind die Rollen Master und Observer besonders interessant. Ein Observer überwacht ausschließlich des Systems. Wenn der Oberserver einen Fehler feststellt, teilt er den Fehler einem Master mit. Der Master hat nun die Aufgabe eine entsprechende Gegenmaßnahme einzuleiten. Dabei müssen in einem System immer mindestens zwei Master und Observer vorhanden sein, wodurch ein SPF ausgeschlossen werden kann. Die Observer überwachen zusätzlich die Master und sich selber, beziehungsweise gegenseitig. Dabei ist immer ein Master

aktiv, während die anderen im Hot-Standby verweilten. Dadurch ist es zu jeder Zeit möglich, die Arbeit eines fehlerhaften Master zu übernehmen. Zusätzlich wird beim Eintreten eines solchen Ausfalls, also eines Masters oder Observers, direkt eine neue Node ausgewählt, welche die Tätigkeit im Hot-Standby übernimmt. Hierdurch wird sichergestellt, dass kein SPF innerhalb des Gesamtsystems vorhanden ist. Dieser Ansatz ist sehr umfangreich in der Umsetzung, stellt aber eine sehr interessante und fehlertoleranten Entwurf dar. Durch diesen Entwurf wird zudem sichergestellt, dass auch die Infrastruktur vor Gefahren und Schäden geschützt ist und nicht nur die funktionalen Module, die dies beispielsweise mithilfe der NMR erreichen.

Die Arbeit von den Autoren Al-Haddad et. al. [1] beschäftigt sich zum Teil mit der Steuerung und dem Scheduling der Rekonfiguration. Das System wird permanent, durch einen Observer, überwacht. Wenn der Observer ein Fehlverhalten erkennt, wird zunächst versucht festzustellen, von welchem genauen Typ der Fehler ist, wodurch es möglich ist, festzustellen mit welcher Gegenmaßnahme der Fehler behoben werden kann. Zunächst versucht ein Automat den Fehler durch ein weiteres Modul zu maskieren. Zusätzlich besteht die Möglichkeit, das Modul nach dem Maskieren zu verschieben. Dazu muss zunächst geprüft werden, ob eine Partition vorhanden ist, zu der ein Verschieben möglich ist oder ob das Verschieben erst durch das Entfernen eines anderen Moduls möglich ist. Dabei ist das Entfernen abhängig von den Prioritäten. Der Entwurf muss nicht zu jederzeit vollständig redundant sein. Durch das wiederholte Wechseln zwischen verschiedenen Redundanzvarienten, können die Ressourcen und der Stromverbrauch reduziert werden und es wird trotzdem ein hohes Maß an Sicherheit gewährleistet. Es ist möglich, dass nur ein Modul allein ausgeführt wird, ohne jegliche Redundanz. Gelegentlich wird ein identisches Modul in eine freie Partition geladen, um zu prüfen, ob die Werte noch korrekt sind. Wenn die Werte unauffällig sind, kann das gerade geladene Modul entfernt werden. Unterscheiden sich die Ausgabewerte, wird ein drittes Modul geladen, wodurch die Fehlerquelle ermittelt werden kann. Diese Herangehensweise ist in ihrer Art simpel, aber wohl überlegt. Der Implementierungsaufwand hält sich hierbei in Grenzen. Vor allem das hybride Prüfen der Fehler stellt ein interessantes Konzept dar. Ein Nachteil, auch bei diesem System, ist die Abhängigkeit von einer externen Workstation. Diese übernimmt auch in dieser Arbeit die Steuerung der Abläufe.

3.3 Datensynchronität und Kommunikation

Ein wichtiger Aspekt, welcher mithilfe der Infrastruktur für die partielle Rekonfiguration gelöst werden muss, ist die Datensynchronität zwischen gleichartigen redundanten Modulen.

Dieser Punkt ist wichtig, da durch falsche Daten ein Modul im Normalfall falsche Ergebnisse an den Voter weitergibt und dieser daraufhin das Modul als fehlerhaft kennzeichnet. Daher ist es wichtig, dass die Daten der Module auf irgendeine Art und Weise, beispielsweise nach dem Verschieben eines Moduls, synchronisiert werden, sodass das eben geladenen Modul auf einem identischen Wissenstand arbeitet, wie die gleichartigen Module. Dazu wurden verschiedene wissenschaftliche Quellen betrachtet und ausgewertet, um einen Mechanismus für die eigene Infrastruktur ableiten zu können.

Die Autoren Szurman et al.[15] haben einen Vorschlag zur Zustandssynchronisation nach einer erfolgreichen partiellen Rekonfiguration beschrieben. Dazu wurde ein Synchronize Arbiter realisiert, welcher nach der Rekonfiguration die Synchronisation steuert. Dazu wurde der Arbiter mit jedem beteiligten Modul verbunden. Jedes Modul muss zusätzlich einen Synchronization Controller implementieren. Wenn ein Modul synchronisiert werden soll, gibt der Arbiter allen anderen Modulen das Signal, dass die Arbeit eingestellt werden muss. Danach tauschen die einzelnen Module, die zu synchronisierenden Daten miteinander aus. Dabei überwacht der Arbiter den Synchronisationsprozess. Nachdem alle Module synchronisiert wurden, kann der Arbiter wieder alle Module freigeben und das System kann seine Arbeit fortsetzen. Durch die externe Steuerung ist eine gute Überwachung des Prozesses möglich, jedoch stellt eben dieses Modul eine SPF dar. Außerdem muss bei diesem Vorgehen zusätzlicher Aufwand von einem Applikationsentwickler aufgebracht werden, da es nötig ist, dass jedes Modul einen eigenen Controller implementiert.

Die Autoren Lüdtke et al.[6] setzen auf ein voll-vermaschtes Netzwerk. Das Bussystem wird mithilfe von Spacewire[6] aufgebaut. Dadurch ist es allen Komponenten möglich, miteinander zu kommunizieren. Zusätzlich kann das Netz in mehrere Segmente eingeteilt werden. Dies geschieht mithilfe von Routern, die mehrere Segmente miteinander verbinden können. Durch den vollvermaschten Aufbau wird eine ausfallsichere Kommunikationsstruktur integriert. Die Zustandssynchronisation erfolgt mithilfe von sogennaten Interface Nodes, die beispielsweise auf einen Massenspeicher oder ähnliches zugreifen können. Eine Interface Node erhält zusätzlich die Rolle Storage. Dieser Node ist es möglich, die Zustände einzelner Process Nodes in einen Festwertspeicher abzulegen oder nach einem Ausfall und der anschließenden Rekonfiguration zu synchronisieren. Dazu muss sichergestellt werden, dass die Daten des Speichers auch den aktuellen gültigen Stand wiedergeben. Prinzipiell ist dieser Ansatz sehr robust, jedoch wird nicht geklärt, wann und wie die Daten auf dem Speicher abgelegt werden. Damit ist unklar, ob die gesamte Kommunikation zwischengespeichert wird oder einzelne Module gezielt ein

Checkpoint setzen. Dafür ist das integrierte Bussystem und die erreichte Voll-Vermaschung robust und hochverfügbar.

In der Arbeit von Pilotto et al.[9] spielt die Synchronisation nach einer erfolgreichen Rekonfiguration die zentrale Rolle. Dabei betrachten die Autoren einen beispielhaften Automaten und ermitteln anhand diesem, welche Zustände sich am besten für das Speichern eines Checkpoints verwenden lassen. Dazu nehmen sie alle minimalen Schleifen des Automaten, um am Ende jeder Schleife einen Checkpoint einzufügen. Der Checkpoint wird nicht extern gespeichert, sondern nach einer Rekonfiguration direkt von einem benachbarten Modul angefordert. Auf Basis des Checkpoints kann das neu geladene Modul mit einer deutlich geringeren Verzögerung weiterarbeiten. Dieser Ansatz stellt nur eine Lösung für den in der Arbeit beschriebenen Automaten dar. Eine solche Implementierung sollte auf Seiten des Applikationsentwicklers erfolgen und kann nicht mithilfe der Infrastruktur gewährleistet werden.

Letztlich kann festgehalten werden, dass es viele Möglichkeiten gibt, die Synchronisation zu gewährleisten. Jedoch existiert hierzu kein generischer Ansatz, da dies zu stark von der konkreten Applikation abhängt. Daher sollte die Infrastruktur Möglichkeiten bieten, eine Synchronisation zu starten und überwachen, als zu versuchen, diese generisch abzubilden.

[6] Spacewire ist ein Feldbus, welcher speziell im Hinblick auf Robustheit an die Anforderungen im Weltraum entwickelt wurde (http://spacewire.esa.int/content/Home/HomeIntro.php).

4 Systemvorschlag

Dieses Kapitel beschreibt einen ersten Systemvorschlag. Die Basis für den Vorschlag stellen die Arbeiten aus Kapitel 3 sowie die im folgenden Abschnitt beschriebene Problemstellung (Abschnitt 4.1). Ziel dieses Kapitels ist es, orientiert an den definierten Zielen, einen ersten Systementwurf zu beschreiben, in dem eine Infrastruktur präsentiert wird, mit dessen Hilfe es möglich ist, die partielle Rekonfiguration in Verbindung mit verschiedenen Varianten der NMR zu nutzen. Von der Infrastruktur soll dazu das *"entfernen/hinzufügen"*, *"verschieben"* und *"neuladen"* eines funktionalen Moduls ermöglicht werden, wie es in Abbildung 4.1 schematisch dargestellt wird. Weiter soll eine Fehlerdetektion möglich sein. Das Kapitel analysiert zunächst die Probleme, um zu zeigen, was von der Infrastruktur und dem Redundanzkonzept berücksichtigt werden muss. Im Anschluss daran wird der Entwurf sowie die Kommunikation und das Protokoll ausführlich beschrieben.

4.1 Problemstellung

Als Grundlage für die Definition der Problemstellung wird die TMR herangezogen. Dabei wird im ersten Abschnitt betrachtet, welche Probleme vor, während oder nach der partiellen Rekonfiguration für das Redundanzkonzept bestehen und was bei einem Entwurf beachtet werden muss. Im Anschluss daran wird ein Blick auf die Herausforderungen an die Systemsteuerung geworfen, um zu ermitteln welche Aspekte von der Steuerung berücksichtigt werden müssen. Der nächste Abschnitt betrachtet noch einmal die Datensynchronität und die Kommunikation des Systems. Im letzten Abschnitt wird ein kurzer Blick auf das Startverhalten des Systems geworfen.

4.1.1 TMR-Redundanzkonzept

Wie bereits erwähnt, besteht das Konzept aus drei funktionalen Modulen und einem Voter. Dabei überprüft der Voter die Ausgangsdaten aller Module, beispielsweise auf Similarität. Der Voter kann unterschiedlich komplex sein, je nach Art und Weise der Datenprüfung. Auf welche Art die Daten geprüft werden müssen, muss durch eine konkrete Applikation entschieden

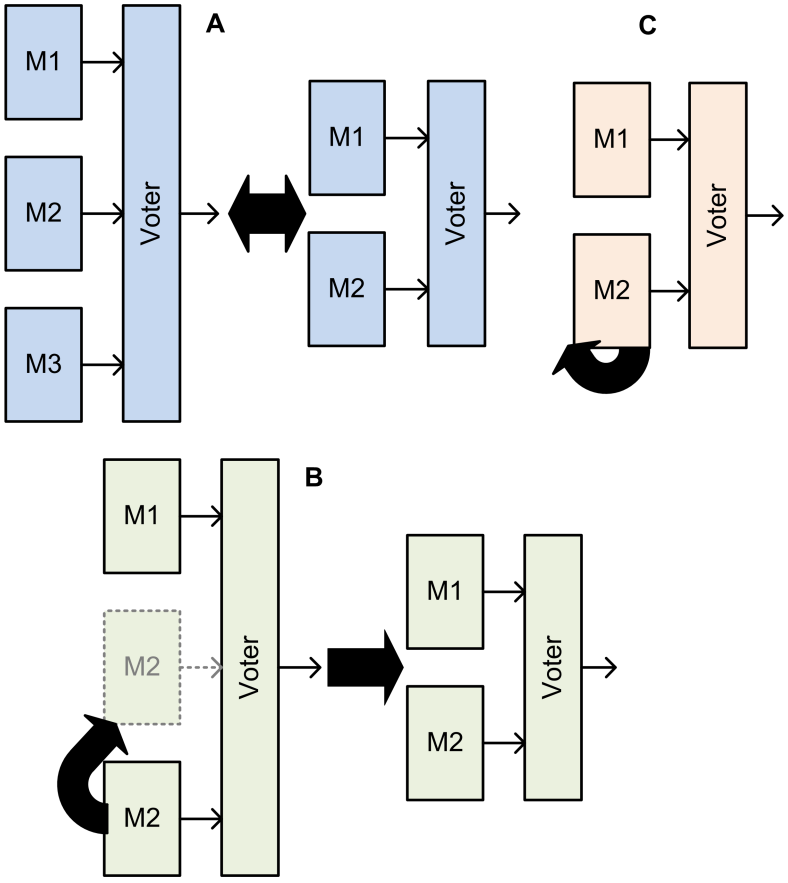

Abbildung 4.1: Schematische Darstellung der Leistungsreferenz der Infrastruktur: (A) Modul entfernen/hinzufügen, (B) Modul verschieben, (C) Modul neuladen.

werden. In diesem Abschnitt wird die TMR betrachtet und geprüft, welche Auswirkungen die Rekonfiguration auf das Konzept hat. Zunächst wird geprüft, welche Auswirkungen das Entfernen eines funktionalen Moduls auf das Konzept hat (vgl. Abbildung 4.1 (A)). Durch das Entfernen des Moduls würde der Voter möglicherweise einen Fehler erkennen, da dadurch am Eingang ein undefinierter Zustand anliegt. Der Voter würde entscheiden, dass das Modul fehlerhaft und nicht mehr funktionstüchtig ist, da dieser keine Kenntnis über das Entfernen hat. Außerdem würde ab dem Moment des Entfernens nur noch die DMR vorhanden sein. Wenn ein weiteres Modul ausfällt, kann der Voter nicht mehr feststellen, welches Modul den Fehler erzeugt. Es kann nur noch erkennen das ein Fehlverhalten vorhanden ist.

Der nächste Aspekt stellt das Hinzufügen eines Moduls dar (vgl. Abbildung 4.1 (A)). Ausgangspunkt dazu stellt die DMR dar, welche fehlerfrei funktioniert. Für den Voter stellt das Hinzufügen eine zusätzliche Herausforderung. Als Grundvoraussetzung muss der Voter zunächst einmal auch drei Module unterstützen und nicht nur zwei. Das bedeutet, dass der Voter eine gewisse Flexibilität benötigt. Da die Module je nach konkreter Anwendung interne Zustände und Variablen besitzen, auf dessen beispielsweise Berechnungen aufbauen, kann es zu einem Synchronitätsproblem kommen. Das neue Modul hat dadurch möglicherweise zu Beginn andere Ergebnisse und der Voter würde dieses Modul direkt wieder als fehlerhaft kennzeichnen. Daher ist eine Synchronisation der Daten oder eine Wartezeit für den korrekten Betrieb unabdingbar. Diese Informationen müssten auf Seiten der konkreten Applikation definiert und in der Infrastruktur berücksichtigt werden.

Eine weitere Betrachtung stellen die Auswirkungen des Neuladens eines Moduls innerhalb seiner Partition und das Verschieben in eine andere Partition dar (vgl. Abbildung 4.1 (B) und (C)). Beim Neuladen und Verschieben treten zwei bereits erwähnte Probleme auf: die Totzeit und Datensynchronität. Vom Beginn des Neuladens oder Verschiebens bis zum lauffähigen Modul vergeht eine gewisse Zeit und während dessen liegen am Ausgang undefinierte Zustände an. Dadurch ist es möglich, dass der Voter das Modul fälschlicherweise als fehlerhaft markiert. Auch die Synchronität der Daten muss wieder sichergestellt werden, da es auch hierbei wieder Probleme in der Zusammenarbeit mit dem Voter gibt. Speziell für das Verschieben stellt die Verbindung zwischen Modulausgang und Votereingang eine zusätzliche Problematik dar. Durch die Infrastruktur muss sichergestellt werden, dass auch die Verknüpfung angepasst wird.

Aus dieser Analyse ergeben sich folgende Probleme, die beim Entwurf der Infrastruktur und des

Redundanzkonzeptes berücksichtigt werden sollten, damit ein robustes und fehlertolerantes System entworfen wird:

- Der Voter muss Kenntnis über die Rekonfiguration haben. Auch die Art sollte hierbei berücksichtigt werden.
- Es wird ein Mechanismus benötigt, welcher die Datensynchronität sicherstellt und den Datenaustausch überwacht.
- Es wird ein Zeitmangement benötigt, welches prüft, ob das Modul bereits vollständig geladen wurde und bereits korrekte Daten liefern kann.
- Die Lösung benötigt einen flexiblen Kommunikationskanal, wodurch physikalische Verbindungsänderungen möglich sind.
- Der Voter stellt einen SPF dar und daher sollte zusätzlich eine Lösung gefunden werden die diese Problematik auflöst.

4.1.2 Systemsteuerung

In diesem Abschnitt wird betrachtet, was von der Steuerung der Rekonfiguration beachtet werden muss. Ein spezielles Augenmerk wird hierbei auf die Steuerung und dem Überwachen der Rekonfiguration gelegt. Bisher ist nicht gänzlich geklärt, welches Modul die Verantwortung für die Rekonfiguration übernimmt und welches Modul eben diese überwacht. Eine Möglichkeit wäre es, dass der Voter diese Tätigkeit übernimmt, da es zummindest für ein Teilsystem die zentrale Komponente darstellt. Jedoch müsste dadurch jeder Voter für jedes Teilsystem diese Tätigkeit übernehmen und es müsste eine umfangreiche Absprache zwischen allen Voter implementiert werden. Außerdem würde hierdurch keine klare Trennung zwischen den Aufgaben eingehalten werden. Eine weitere Möglichkeit wäre es, diese Tätigkeit auf mehrere Module zu verteilen. Hierbei wären die gleichen Probleme, wie bereits beim Voter vorhanden. Außerdem gibt es innerhalb des FPGA nur eine beschränkte Anzahl an dedizierte Komponenten mit denen eine Rekonfiguration realisiert werden kann, wodurch sich bereits ein Flaschenhals ergibt. Daher scheint es am sinnvollsten, hierfür ein eigenes unabhängiges Modul vorzusehen.

Ein weiterer Aspekt stellt das Prüfen der Vor- und Nachbedingungen der Rekonfiguration dar. Es muss zum Einen geprüft werden, in welcher Partition das Modul platziert werden kann und ob dazu beispielsweise ein vorheriges Entfernen eines anderen Moduls nötig ist. Auch muss an irgendeiner Stelle definiert werden, ob ein Verschieben nötig ist oder ein Neuladen ausreicht. Nach der Rekonfiguration muss zudem geprüft werden, ob diese erfolgreich war. Ein weiteres

Problem stellt das Scheduling dar. Es kann möglich sein, dass mehrere Module zur gleichen Zeit, beispielsweise verschoben oder neugeladen werden sollen. Hierbei könnten zusätzlich Prioritäten vorhanden sein, welche bei der Reihenfolge berücksichtigt werden müssen. Auch stellt hier wieder die Koordination der Zielpartitionen einen wichtigen Aspekt dar. Daher scheint es auch durch diese Punkte sinnvoll zu sein, ein eigenes Modul für diese Tätigkeiten vorzusehen, da hierdurch eine zentrale Stelle genau die angegebenen Tätigkeiten koordinieren und ausführen kann.

Zudem stellt sich die Frage, ob eine Rekonfiguration unterbrechbar sein muss, unabhängig davon, wie es technisch umgesetzt werden kann. Außerdem sollte es möglich sein, auch andere Module über einen entsprechenden Kanal mitzuteilen, dass eine Rekonfiguration durchgeführt wird. Ansonsten kann es in anderen Modulen zu Fehlern kommen, da diese auf Daten warten oder durch Unwissenheit falsche Daten verarbeiten.

Aus dieser Analyse ergeben sich folgende Probleme, die beim Entwurf der Infrastruktur und des Redundanzkonzeptes berücksichtigt werden sollten, damit ein robustes und fehlertolerantes System entworfen wird:

- Es wird eine oder mehrere Komponenten zur Steuerung und Überwachung der Rekonfiguration benötigt.
- Es wird eine Art Scheduling nötig, welches es ermöglicht auch mehrere Rekonfigurationsanfragen zur Zeit zu verarbeiten und priorisieren.
- Zum Prüfen der Vor- und Nachbedingungen sowie zum Ermitteln einer gültigen Zielpartition muss ein Modul vorgesehen werden oder eine andere Lösung gefunden werden.
- Es muss eine Lösung gefunden werden, mithilfe dessen definiert wird, welche Rekonfigurationstätigkeit ausgeführt wird: hinzufügen, verschieben, neuladen oder entfernen.
- Auf einem zentralen Kanal muss die Rekonfiguration verbreitet werden, um Fehler in anderen Modulen zu verhindern.
- Optional sollte die Möglichkeit einer Unterbrechung geprüft werden.

4.1.3 Datensynchronität und Kommunikation

Wie bereits erwähnt, stellt die Kommunikation und Datensynchronität einen wesentlichen Punkt dar. Problemtisch sind vor allem die statischen Kommunikationskanäle zwischen den funktionalen Modulen und dem Voter. Durch diese Gegebenheit ist es vor allem wichtig, dass

die Infrastruktur eine Möglichkeit schafft, die Verknüpfungen dynamisch neu zu allokieren. Es stellt ein weiteres Problem dar, dass von der klassischen TMR keine direkte Kommunikation zwischen den einzelnen Modulen möglich ist. Eine solche Kommunikation könnte aber nötig werden, da es möglich sein muss, die Daten zwischen den Modulen zu synchronisieren. Das bedeutet, dass die aktuelle Punkt-zu-Punkt-Verbindung zwischen den funktionalen Modulen und dem Voter, gegen eine andere Art der Verbindung ersetzt werden muss.

Des Weiteren ist es mit der TMR nicht möglich, dass alle Module des Teilsystems mit einem anderem Teilsystem kommunizieren. Nur der Voter kann mit seinem Signalausgang mit genau einem anderen Teilsystem verbunden werden. Dadurch ist der Aufbau statisch. Für die Zustandssynchronisation sollte somit eine Möglichkeit geschaffen werden, welche das statische Konzept dahingehend verändert, dass ein Datenaustausch auch untereinander und mit anderen Modulen möglich ist.

Aus dieser Analyse ergeben sich folgende Probleme, die beim Entwurf der Infrastruktur und des Redundanzkonzeptes berücksichtigt werden sollten, damit ein robustes und fehlertolerantes System entworfen wird:

- Die Kommunikationskanäle zwischen den einzelnen Modulen und dem Voter müssen veränderlich sein.
- Ein Austausch zwischen mehreren Teilsystemen soll ermöglicht werden.
- Es wird eine Möglichkeit benötigt, um Daten zwischen einzelnen Modulen und Teilsystemen auszutauschen.

4.1.4 Startverhalten

Ein bisher nicht betrachteter Punkt stellt das Startverhalten und die Startkonfiguration des Systems dar. Dies stellt nicht nur für das Redundanzkonzept einen kritischen Faktor dar, sondern gilt für das gesamte System, welches auf dem FPGA geladen wird. Zum einen ist es notwendig zu definieren, in welchen Partitionen die Module und die Voter geladen werden und wie die Verbindungen der Systemteile aussehen. Aus technischer Sicht ist es zum anderen problematisch direkt das System auszuführen, da es mehrere Sekunden dauern kann bis der gesamte FPGA betriebsbereit ist. Wenn dies nicht beachtet wird, wäre es möglich, dass bereits zum Systemstart Fehler vom Voter identifiziert werden. Im Idealfall würde nach dem Laden zunächst eine Art Power-On Self-Test (POST) ausgeführt werden, um sicherzustellen, dass das System nach dem Starten fehlerfrei und betriebsbereit ist. Grund hierfür sind Fehler, die

während des Betriebes auftreten und auf ein falsches Starten zurückzuführen sind.

Aus dieser Analyse ergeben sich folgende Probleme, die beim Entwurf der Infrastruktur und des Redundanzkonzeptes berücksichtigt werden sollten, damit ein robustes und fehlertolerantes System entworfen wird:

- Es muss sichergestellt werden, dass die verschiedenen Module und Voter nach dem Zuschalten der Spannung in einer passenden Partition geladen werden und die Verbindungen untereinander richtig verknüpft werden.
- Nach dem erfolgreichen Start sollte eine Art POST zunächst das System überprüfen.
- Es sollte ein zentrales Freigabesignal implementiert werden, nach dessen Senden erst alle Teilsysteme ihre Arbeit aufnehmen, um sicherzustellen, dass gleichartige Module nicht zu verschiedenen Zeiten ihre Arbeit aufnehmen und dadurch direkt Fehler vom Voter erkannt werden.

4.2 Systementwurf

In diesem Abschnitt wird der Systementwurf beschrieben. Mithilfe des Entwurfs werden die ermittelten Probleme behoben. Zusätzlich wurden Entwurfsentscheidungen auf Basis der präsentierten wissenschaftlichen Arbeiten getroffen. Der erste Abschnitt beschreibt zunächst die Systemkommunikation und das gewählte Bussystem. Im Anschluss daran wird die Systemsteuerung mit den jeweiligen Komponenten beschrieben. Danach wird ein Blick auf die Partitionen, welche als Ziel für die DPR dienen, geworfen. Im letzten Abschnitt wird das Redundanzkonzept präsentiert und die Funktion des Konzeptes beschrieben. Der Systementwurf kann der Abbildung 4.2 entnommen werden und dient als Basis für die nachfolgenden Kapitel.

4.2.1 Kommunikation - Bussystem und Protokoll

Für die Lösung wurden zwei Bussysteme vorgesehen: Der Datenbus und der Steuerbus. Beide Bussysteme sind Multi-Master-fähig, wodurch eine M-zu-N-Kommunikation zwischen allen funktionalen Modulen möglich ist. Um die einzelnen Module und Datenpakete adressieren zu können und jeweils eine eindeutige Kennung zu erhalten wurde eine *"Modul-ID (MID)"* und eine *"Datatype-ID (DID)"* eingeführt. Die MID ist 8-Bit breit und deckt somit einen Zahlenbereich von 0 bis 255 ab. Dabei ist die 0 und die 255 reserviert und darf nicht verwendet werden. Die DID ist ebenfalls 8-Bit breit und deckt den gleichen Zahlenbereich ab. Hier dürfen zusätzlich die Zahlen 0 und 255 vergeben werden. Damit wird das Datum gekennzeichnet. Diese Kennzeichnung

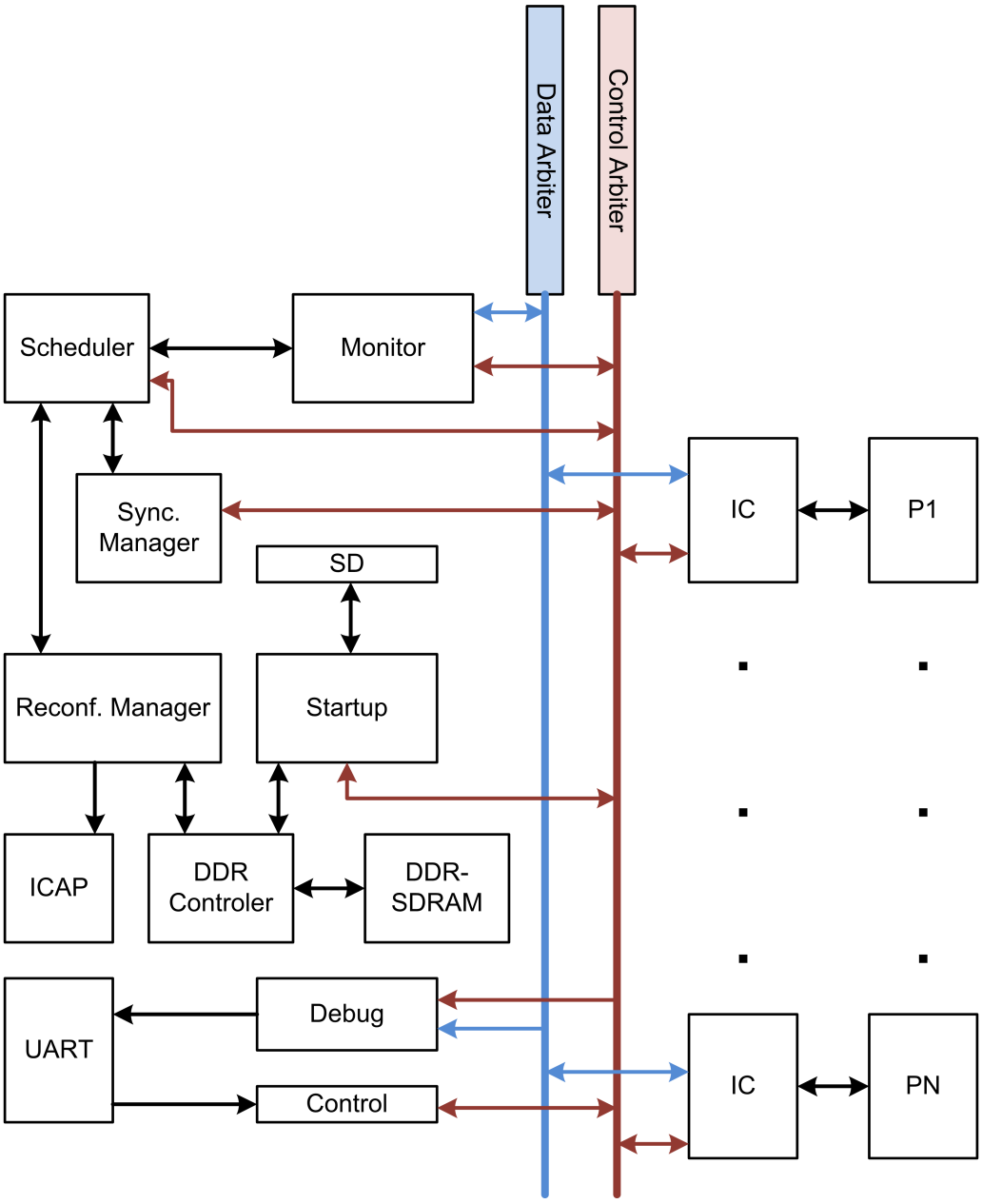

Abbildung 4.2: Darstellung der Infrastruktur mit allen funktionalen Komponenten und dem Bussystem.

ist applikationsabhängig. Beispielsweise kann durch den Wert 128 (0x80) der Rollwert eine Inertial Navigation System (INS) gekennzeichnet werden und darf somit systemintern nur mit diesem Wert belegt werden. Dadurch ist das Filtern für den Voter einfacher. Durch die zusätzliche MID ist immer angegeben, welches konkrete Modul die Nachricht versendet hat. Der Timeslot ist insgesamt 16-Bit breit und besteht aus einer MID und einer Restzeit. Für die Restzeit sind die letzten 8-Bit reserviert. Durch die MID weiß das Modul, ob es sendeberechtigt ist. Die Restzeit gibt dem Modul zusätzlich an, wie lange es noch senden darf. Sobald der Wert 255 erreicht wird, weiß das Modul, dass es nach dem nachfolgenden Takt nicht mehr senden darf.

Datenbus

Der Datenbus besteht aus den folgenden Signalen: *"CLK"*, *"RST_N"*, *"ADDR"*, *"DATA"* und *"TSLOT"*. Die Bitbreite der einzelnen Signale und die Beschreibung jedes Signals kann der Tabelle 4.1 entnommen werden. Die Adresse besteht aus der MID und DID des Senders. Wenn eine Punkt-zu-Punkt-Verbindung gewünscht wird, kann zusätzlich die MID und DID eines Empfängers angegeben werden, ansonsten muss mindesten die MID auf 0 oder 255 gesetzt werden. Das

Name	**Abkürzung**	**Bitbreite**	**Beschreibung**
Clock	CLK	1	Taktsignal
Reset, Low-Active	RST_N	1	Resetsignal, Low-Active
Address	ADDR	32	Adresse
Data	DATA	32	Das zu übertragende Datum.
Time Slot	TSLOT	16	Das Timeslot-Signale

Tabelle 4.1: Signalbeschreibung des Datenbusses mit Angaben zur Signalbreite.

"CLK"-Signal gewährleistet die datensynchrone Übertragung zwischen den einzelnen Modulen. Durch das *"RST_N"*-Signal können alle am Bus hängenden Module zurückgesetzt werden. Da die Anforderungen an den Bus die Multi-Master-Fähigkeit ist, wurde das *"TSLOT"*-Signal hinzugefügt. Dadurch werden Timeslots von einer zentralen Uhr an die Module verteilt und es wird sichergestellt, dass immer nur ein Modul zur Zeit sendet. Dadurch kann auf dem Bus keine Kollision auftreten. Eine Datenübertragung beginnt mit der steigenden Flanke des *"CLK"*-Signals und muss bis zum Ende der nächsten steigenden Flanke abgeschlossen sein. Unter normalen Betriebsbedingungen ist der Bus also ein Broadcast-Netzwerk.

Steuerbus

Der Steuerbus ist ähnlich wie der Datenbus aufgebaut. Dieser besteht aus den folgenden Signalen: *"CLK"*, *"RST_N"*, *"CMD"* und *"TSLOT"*. Die Bitbreite der einzelnen Signale und die Beschreibung jedes Signals kann der Tabelle 4.2 entnommen werden. Das *"CMD"*-Signal ist insgesamt 32-Bit breit. Dabei besteht das Signal aus einem 28-Bit breitem Kommando, welcher der Tabelle 4.3 entnommen werden können. Zusätzlich wird die Quell-MID und die Ziel-MID angegeben. Die restlichen Bits sind für die Nutzlast reserviert. In diesem Bereich können zusätzliche Informationen, je nach Kommando angehangen werden. Das *"CLK"*-Signal gewährleistet

Name	**Abkürzung**	**Bitbreite**	**Beschreibung**
Clock	CLK	1	Taktsignal
Reset, Low-Active	RST_N	1	Resetsignal, Low-Active
Command	CMD	28	Kommando
Time Slot	TSLOT	16	Das Timeslot-Signale

Tabelle 4.2: Signalbeschreibung des Steuerbusses mit Angaben zur Signalbreite.

die datensynchrone Übertragung zwischen den einzelnen Modulen. Durch das *"RST_N"*-Signal können alle am Bus hängenden Module zurückgesetzt werden. Auch der Steuerbus muss Multi-Master-fähig sein. Daher wurde das *"TSLOT"*-Signal hinzugefügt. Das Signal funktioniert identisch zum Datenbus. Durch dieses können Kollisionen auf dem Bus verhindert werden. Der Bus stellt immer Punkt-zu-Punkt-Verbindungen her, die aber von allen Modulen gelesen werden kann.

Durch das Einführen der Kommandos wird ein zusätzliches Protokoll definiert. Alle Kommandos und dessen Erklärung können der Tabelle 4.3 entnommen werden. Gegliedert werden die Kommandos in *"Generic"*, *"Reconfiguration"* und *"Data Synchronization"*. Die Kommandos vom Typ *"Generic"* stellen grundlegende Funktionalitäten zur Verfügung, wie den Systemstart und die Fehlerübertragung. Kommandos vom Typ *"Reconfiguration"* werden nur von der Systemsteuerung ausgeführt und steuern mit dessen Hilfe die Rekonfiguration und publizieren eben diese an den Rest des Systems. Die Kommandos vom Typ *"Data Synchronization"* dienen ausschließlich der Synchronisation nach einer Rekonfiguration. Dabei werden drei Typen unterschieden. Für den ersten Typ ist keine Synchronisation erforderlich. Für den zweiten Typ stellt das Modul die Synchronisation selber sicher und teilt den erfolgreichen Abschluss über den Bus mit. Der dritte Typ benötigt zusätzliche Zeitslots, um entweder schnell synchronisiert zu werden oder eine große Datenmenge zu synchronisieren. Auch dieser teilt den erfolgreichen Abschluss über den Bus mit. Ein Überblick über den Ablauf des Protokolls kann der Abbildung

Kommando	Beschreibung	Hexcode	Typ
NOP	Diese Kommando wird gesendet, wenn nichts gemacht werden soll.	0	Generic
START	Dieses Kommando wird übertragen, wenn das System betriebsbereit und vollständig geladen wurde.	1	Generic
FAILURE	Wird übertragen, sobald ein Fehler im System, Modul oder Voter erkannt wird. Details zum Fehler werden in der Nutzlast angehangen.	2	Generic
TURN_ON	Einschaltsignal an ein einzelnes Modul.	3	Generic
TURN_OFF	Abschaltsignal an ein einzelnes Modul.	4	Generic
RECONF_BEG	Signalisiert den Beginn einer Rekonfiguration.	5	Reconfiguration
RECONF_CMP	Signalisiert das Ende einer Rekonfiguration.	6	Reconfiguration
SYNC_BEG	Signalisiert den Beginn einer Datensynchronisation.	7	Data Synchronization
SYNC_CMP	Signalisiert das Ende einer Datensynchronisation.	8	Data Synchronization
SYNC_REQ	Anfragen, welche Synchronisation erforderlich ist.	9	Data Synchronization
SYNC_NO	Antwort, keine Synchronisation.	10	Data Synchronization
SYNC_INT	Antwort, Synchronisation intern, also durch das Modul selber.	11	Data Synchronization
SYNC_EXT	Antwort, Synchronisation extern also durch ein anderes Modul.	12	Data Synchronization
SYNC_DEST	Synchronisationsziel bekanntgeben.	13	Data Synchronization
SYNC_TSLOT	Zusätzliche Zeitslots übertragen.	14	Data Synchronization
SYNC_TSLOT_CMP	Zusätzliche Zeitslot wurden übertragen.	15	Data Synchronization
SCHEDULE_CMP	Nach Abschluss des Schedulings der Rekonfiguration.	16	Reconfiguration
RESERVED	Dieser Bereich ist für zukünftige Kommandos reserviert.	17-31	-

Tabelle 4.3: Auflistungen und Beschreibung der verschiedenen Kommandos für die Steuerung durch die Infrastruktur.

4.3 entnommen werden. Zunächst muss das System freigegeben werden. Danach verweilt das System in einem IDLE-Zustand und überträgt keine Daten. Von diesem Punkt kann ein Modul deaktiviert werden, Fehler senden oder es wird eine Rekonfiguration begonnen. Die Rekonfiguration schaltet automatisch das Zielmodul aus und wartet auf den erfolgreichen Abschluss. Danach kann eine Synchronisations-Variante gewählt werden. Wenn alles erfolgreich war, wird das Modul aktiviert und das System ist betriebsbereit. Der Automat soll das gesamte Protokoll kurz veranschaulichen. In der konkreten Implementierung sind in jedem Modul nur Teile vorhanden, da es eine Kommunikation untereinander beschreibt.

4.2.2 Systemsteuerung

Die Systemsteuerung besteht aus mehreren einzelnen Komponenten: Monitor, Reconfiguration Manager, Scheduler und Synchronous Manager. Der Monitor dient zum Einen der Überwachung des Steuer- und Datenbus. Das bedeutet, dass dieser prüft, ob der Datenfluss korrekt ist und die Zeitslots eingehalten werden. Ansonsten kann der Monitor Module deaktivieren oder rekonfigieren lassen. Zum Anderen nimmt dieser die Fehlersignale entgegen. Der Monitor entscheidet in Abhängigkeit vom Fehlertyp, welche Gegenmaßnahme eingeleitet werden muss. Gegenmaßnahmen reichen von einem einfachen Abschalten über das Neuladen oder Verschieben eines Moduls bis hin zum vollständigen Entfernen.

Der Scheduler erhält Nachrichten über notwendige Rekonfigurationen vom Monitor. Dabei reiht der Scheduler diese in eine Liste ein. Wenn eine Rekonfiguration an der Reihe ist, sendet der Scheduler eine Nachricht an den Monitor und auf den Steuerbus, wodurch auch andere Module über eine Rekonfiguration informiert werden. Daraufhin überträgt der Scheduler die Daten an den Reconfiguration Controller. Außerdem teilt der Scheduler dem Manager mit, welches Modul geladen werden soll.

Der Reconfiguration Controller steuert ausschließlich die Rekonfiguration. Dazu lädt der Controller das Bit-File, welches für die Rekonfiguration nötig ist, über den DDR Controller aus dem DDR-SDRAM. Anhand einer Allokationtabelle bestehend aus der MID und der Zielpartition findet er die korrekte Datei im DDR-Speicher. Der Datenstrom wird an den ICAP weitergeleitet. Diese Schnittstelle greift direkt auf die FPGA-Fabrik zu und steuert die Rekonfiguration. Nachdem die Partition erfolgreich rekonfiguriert wurde, sendet der Reconfiguration Manager eine Nachricht an den Scheduler. Zudem pflegt der Reconfiguration Manager intern eine Lookup-Table, worüber er referenzieren kann, in welche Partition das Modul geladen werden kann.

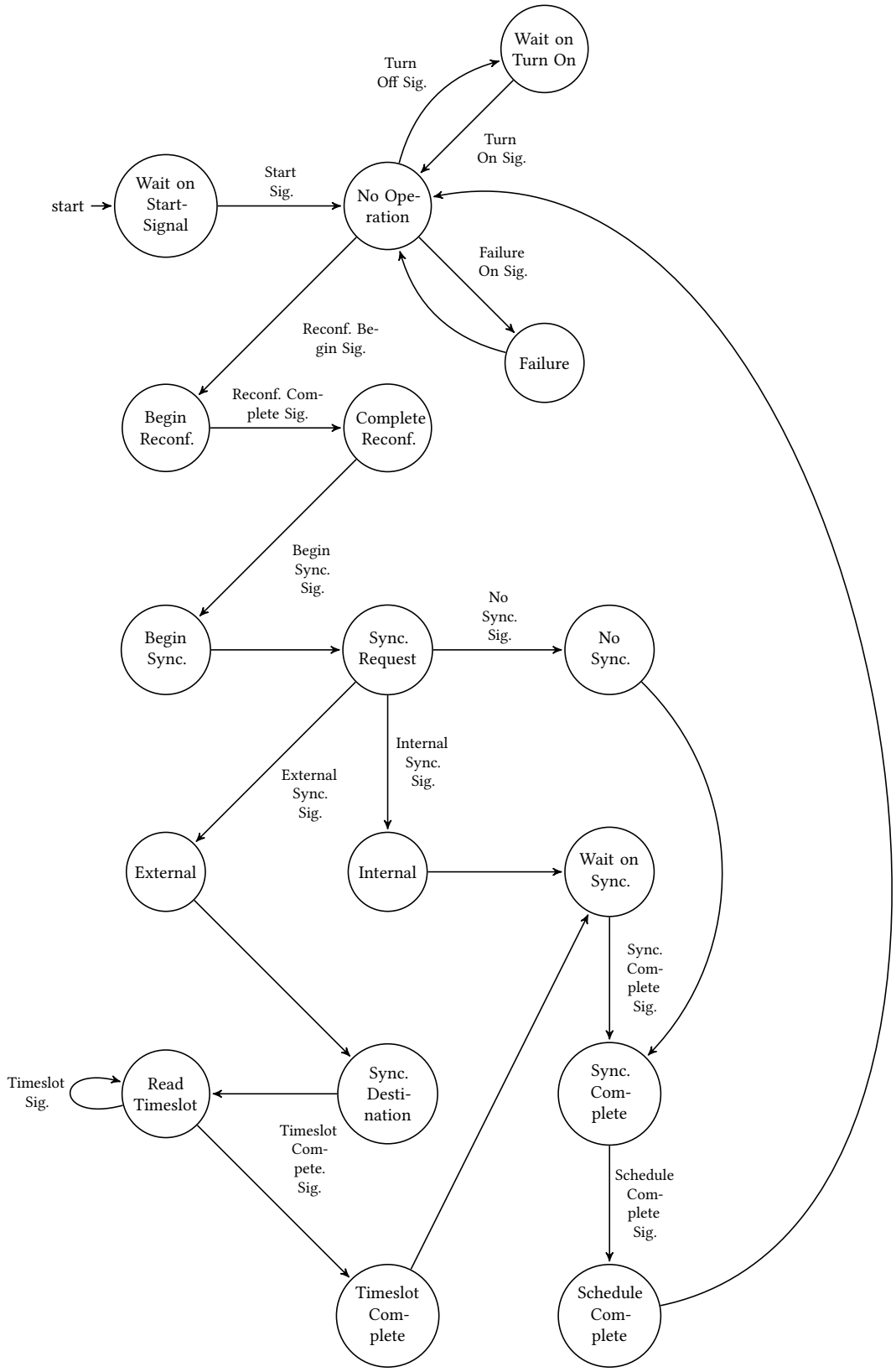

Abbildung 4.3: Schematische Darstellung des Protokollentwurfs für die Kommunikation zwischen den einzelnen Modulen und der Systemsteuerung.

Der Scheduler sendet eine Nachricht an den Synchronous Manager. Der Manager fragt zunächst das neue Modul an, welchem Synchronisationtyp er entspricht. Unabhängig vom Typ wartet der Manager auf eine Rückmeldung. Wenn keine Synchronisation notwendig ist, leitet der Manager direkt eine Nachricht an Scheduler weiter. Im Falle dass eine Synchronisation nötig ist, wartet der Manager auf den Abschluss dieser und leitet erst dann eine Nachricht weiter. Für eine externe Synchronisation verteilt der Manager zusätzliche Zeitslots an ein Modul. Alle Informationen des Synchronous Manager werden auf dem Steuerbus signalisiert, damit auch andere Module über die Vorgänge Bescheid wissen. Wenn der Scheduler seine Rückmeldung erhalten hat, bestätigt er die vollständige Rekonfiguration dem Monitor, der das Modul wieder freigibt.

4.2.3 Interconnect und die Partition

Der Interconnect stellt die Schnittstelle zwischen den beiden Bussen und der Applikationsschicht dar. Die Applikationsschicht entspricht der Partition in der die funktionalen Module und der Voter für die TMR instantiiert werden können. Der Inhalt jeder Partition wird durch die Startkonfiguration festgelegt und kann durch die DPR während der Laufzeit verändert werden. Der IC befindet sich im statischen Bereich des FPGA. Diese Komponente verbindet jede einzelne Partition mit dem Steuer- und Datenbus und implementiert das Protokoll, sodass in der Applikationsschicht dieser Implementierungsaufwand entfällt. Außerdem wird ein zusätzlicher Watchdog instantiiert, auf den die Applikation in regelmäßigen Abständen reagieren muss. Durch diese Schicht ist es möglich, die Applikation vollständig von den Bussen zu trennen. Durch eine Sperrung gelangen keine Daten in die Partition herein, beziehungsweise heraus. Durch diesen Aufbau wird eine höhere Sicherheit garantiert und es werden während einer Rekonfiguration undefinierte Zustände auf dem Bussystem vermieden. Ansonsten werden in dieser Schicht Buffer implementiert, wodurch die Applikationsschicht zu jeder Zeit senden und empfangen kann. Der IC kümmert sich um das Einhalten des Zeitslots.

Debug und Control

Die Debug- und Control-Schnittstellen dienen lediglich der Failure Injection und der externen Systemüberwachung. Beide Komponenten sind über eine UART-Schnittstelle mit einer Workstation verbunden. Die Debug-Komponente sendet alle Daten der Busse an eine Workstation. Dort können die Daten analysiert werden. Dies dient ausschließlich Test- und Überwachungszwecken. Die Control-Komponente kann Daten von einer Workstation empfangen. Mit der

Hilfe dieses Moduls können zum Einen gezielt Fehler in den Bus injiziert werden und zum Anderen gezielt eine Rekonfiguration durchgeführt werden. Dadurch eignet sich diese Komponente besonders für die Failure Injection.

Redundanzkonzept

Das hier vorgeschlagene TMR-Konzept stellt eine Abwandlung von [1] dar. Letztlich soll dem Applikationsentwickler freigestellt sein, wie er innerhalb der Partitionen und der gegebenen Infrastruktur ein Redundanzkonzept implementiert. Dieses Konzept zeigt einen möglichen Weg auf und ist für das Bussystem angepasst. Des weiteren wird eine Möglichkeit geschaffen, mehrere Voter gleichzeitig zu betreiben, wodurch der Voter als SPF nicht mehr im System auftreten kann. Dadurch, dass der Voter über den Steuerbus immer informiert wird, ob sich etwas im System ändert, kann er darauf reagieren und alternativ auch mit weniger als drei Modulen agieren. Jedoch muss bei der Implementierung eine maximale Anzahl prüfbarer Module vorgesehen werden. Dadurch kann jeder Voter unterscheiden, ob nur ein oder mehrere Module vorhanden sind. Das angepasste Konzept kann der Abbildung 4.4 entnommen werden. Der Discrepancy Sensor ermittelt die Unterschiede zwischen den Eingangswerten und erzeugt einen Report an den Redundancy Controller. Der Voter wird nur benötigt, wenn mehr als zwei Module Daten an das System übertragen und kann auf Basis der Ergebnisse des Discrepancy Sensor entscheiden, welches Modul fehlerhaft ist. Der Redundancy Controller überwacht die Schritte des Discrepancy Sensors und des Voters. Von diesen erhält er Reports und entscheidet auf Basis dieser, welche Daten nach außen weitergegeben werden. Außerdem erhält der Redundancy Controller die Steuerinformationen vom Steuerbus, wodurch dieser entscheiden kann, wie viele und welche Module geprüft werden. Da durch den Datenbus und dem Zeitslot-Verfahren die einzelnen Module ihre Daten zu unterschiedlichen Zeiten an den Voter weitergeben, wurde zusätzlich ein Speicher implementiert, welcher die Daten der Reihenfolge nach puffert und erst dann eine Prüfung durchführt, da es ansonsten zu Fehlern kommen würde. Dieses Konzept kann in ihrem Umfang beliebig erweitert werden.

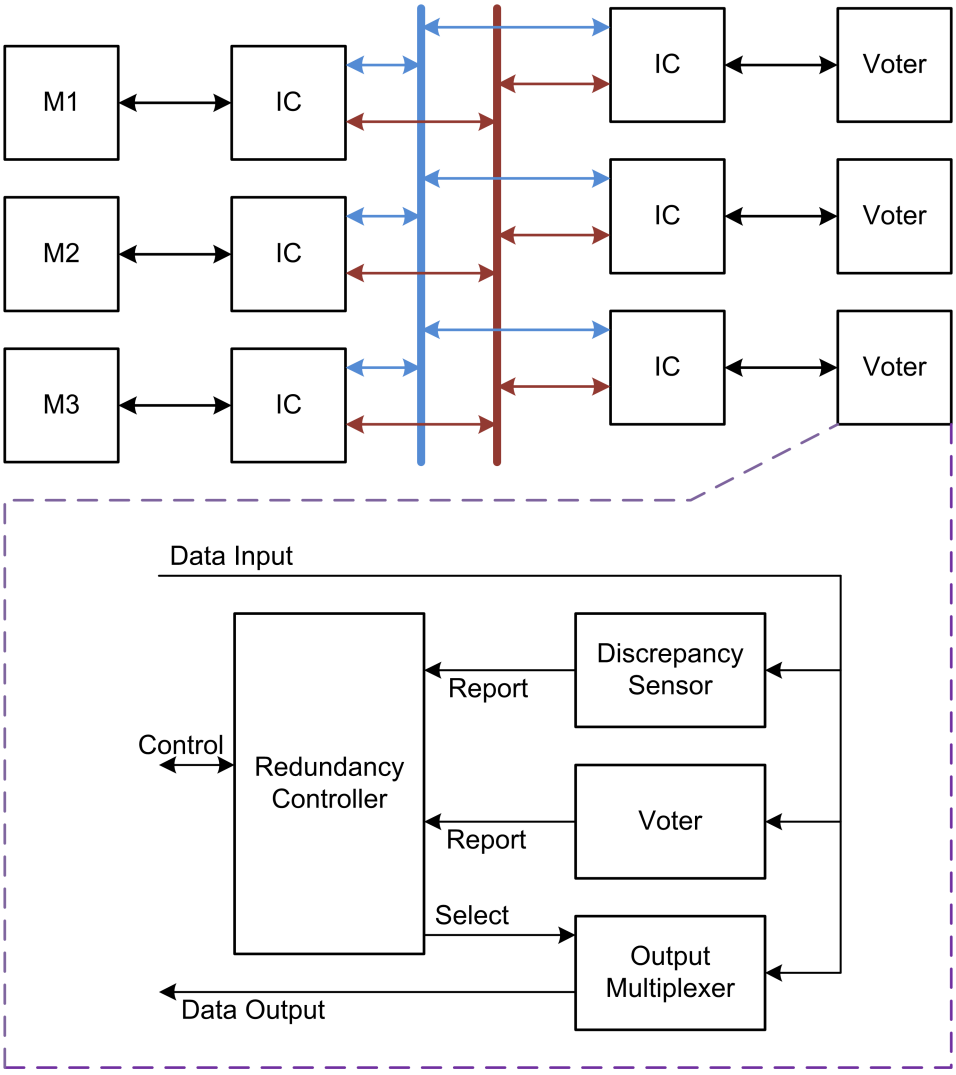

Abbildung 4.4: Darstellung des angepassten TMR-Konzeptes aus [1].

Systemstart

Wie bereits erwähnt, stellt der Systemstart ein gesondertes Ereignis dar. Zu Beginn lädt der FPGA seine Konfigurationsdatei aus einem Flash-Speicher, das sogenannte Golden-Bitfile. In unserem Fall beschreibt diese Datei den Inhalt des statischen Bereichs des FPGA. Dabei werden dynamische Partitionen zunächst nicht geladen. Zunächst lädt die Komponente die Startkonfiguration, die Allokationstabelle sowie alle Bit-Files von der SD-Karte in den DDR-Speicher. Danach wird die Startkonfiguration ausgelesen, wodurch bereits geprüft wird, ob die Daten korrekt im Speicher abgelegt wurden. Durch die Startkonfiguration weiß die Komponente, welche Partition mit welchem Modul geladen werden soll. Nun lädt die Komponente jede Partition die hinterlegt ist. Erst wenn dies abgeschlossen wurde und alle Partitionen korrekt geladen wurden, sendet die Startup-Komponente das Start-Kommando. Nachdem das Kommando auf dem Bus gelegt wurde, beginnen alle Komponenten und Module mit dem nächsten Taktzyklus mit ihrer Arbeit.

5 Systemanalyse

In diesem Kapitel wird die Systemanalyse erläutert und diskutiert. Die Grundlage für die Analyse stellt der Systemvorschlag, welcher im Kapitel 4 präsentiert wurde, dar. Ziel der Analyse ist es Schwachstellen im Vorschlag zu finden, um Ausfällen vorzubeugen. Die Analyse dient dem Systementwurf und der späteren Implementierung als Grundlage. Der erste Abschnitt beschreibt die verwendete Analysemethode und begründet die Auswahl der FMEA. Im zweiten Abschnitt werden die Systemgrenzen definiert und weitere Einschränkungen erläutert. Der letzte Abschnitt zeigt das Analyseergebnis und diskutiert die Ergebnisse.

5.1 Analysemethode

Für die Analyse wurde die System-FMEA gewählt. Das bedeutet, dass speziell das Zusammenwirken von einzelnen Komponenten und Teilsystemen untersucht wird. Dabei zielt die FMEA darauf, mögliche Schwachstellen zu identifizieren. Besonderes Augenmerk liegt auf den Schnittstellen und dessen Kommunikationskanälen. Es werden vor allem die Interaktionen zwischen den Komponenten, Teilsystemen und der Umwelt betrachtet. Durch die Betrachtung sollen zufällige und systematische Fehler identifiziert werden. Als Basis für die Analyse dient der Systemvorschlag aus Kapitel 4. Ziel ist es die Ausfallmodi der Systemkomponenten zu untersuchen, wodurch potentielle Auswirkungen der Fehler auf die Systemleistung ermittelt werden. Dabei setzt die FMEA auf die *"Bottom-Up-Technik"*. Der Vorteil durch die *"Bottom-Up-Analyse"*, beispielsweise gegenüber einer FTA, ist die Identifikation und Priorisierung einer Vielzahl von Ausfallmodi und dessen Ursachen. Eine FTA konzentriert sich ausschließlich auf genau einen einzelnen Ausfallmodi und versucht dafür alle Ausfallmechanismen zu identifizieren. Häufig wird mit einer FMEA begonnen und danach für die Ausfallmodi eine zusätzliche FTA durchgeführt. Eigentlich werden FMEA durch mindestens einen Moderator und mehrere Fachspezialisten der verschiedenen beteiligten Gewerke durchgeführt. In dieser Arbeit wird die FMEA als Basis für einen robusteren Systementwurf eingesetzt. Das wesentliche Ziel ist nicht die vollständige Analyse, sondern eine beispielhafte Analyse, um Probleme aufzuzeigen. Inhalt der Analyse sind folgende Punkte:

- Eingrenzung des Systems
- Analyse potenzieller Fehlerursachen
- Fehlerarten und Fehlerfolgen
- Risikobeurteilung
- Maßnahmen- und/oder Lösungsvorschläge
- Restrisikobeurteilung und/oder -bewertung

Der nächste Abschnitt beschreibt die wesentlichen Randbedingungen und geht auf die Risikobeurteilung ein, da auch diese nur beispielhaft und nach bestem Wissen durchgeführt wird. Dort werden zudem kurz die verschiedenen Fehlerarten diskutiert.

5.2 Systemgrenzen und Randbedingungen

Das Ziel der Analyse ist das Finden verschiedener Schwachstellen. Dabei muss das System und die Analyse zunächst eingegrenzt werden. Folgende Systemkomponenten werden von der FMEA erfasst:

- Monitor
- Reconfiguration Manager
- Datenbus (Arbiter)
- Steuerbus (Arbiter)

Die Betrachtung wird eingeschränkt, da es nicht das Ziel der Arbeit ist, eine komplette Analyse durchzuführen, sondern nur an ausgewählten Beispielen zu zeigen, welche Schwachstellen vorhanden sind, sodass diese im Systementwurf und der Implementierung berücksichtigt werden können. Der Monitor und der Reconfiguration Manager werden anhand ihrer internen Zustände und anhand der Zustände an ihren Schnittstellen untersucht. Dabei werden beide Komponenten nicht vollständig, sondern nur anhand beispielhafter Fehler untersucht. Das gleiche gilt für den Daten- und Steuerbus, beziehungsweise den Arbiter, die die gesamte Businfrastruktur bereitstellen. Das definierte Protokoll aus Kapitel 4 wird nur in Teilen mit einbezogen. Es wird zum Einen angenommen, dass das System bereits betriebsbereit ist und korrekt gestartet wurde. Außerdem wird nur das Senden und Empfangen eines Fehlersignals sowie die Rekonfiguration betrachtet. Die Synchronisation soll in dieser Analyse nicht weiter betrachtet werden, da diese für die erste Umsetzung weniger wichtig ist und zudem die Analyse, um ein vielfaches komplexer werden würde.

Die verwendete FMEA-Tabelle besteht aus folgenden Spalten: Referenznummer, Komponente, Fehlermodi, Fehlerursache, Lokale Effekte, System Effekte, Abstellmaßnahmen und der Resikobewertung. Dabei stellt die Referenznummer eine eindeutige Kennung des Fehlermodi dar. Innerhalb der Spalte Komponente muss die zu untersuchende Komponente eingetragen werden. Die Spalte Fehlermodi beschreibt die unterschiedlichen Fehlermodi. Innerhalb der Spalte Fehlerursachen kann eine Liste möglicher Ursachen je Modi eingetragen werden. Die Spalten Lokale Effekte und System Effekte beschreiben die möglichen Einflüsse auf die Systemteile. Die Abstellmaßnahmen enthalten eine weitere Liste mit möglichen Problemlösungen je Modi. Die Risikobewertung ist hier obligatorisch. Die Auftrittswahrscheinlichkeit kann der Tabelle 5.1 entnommen werden. Das letztliche Risiko wird zudem nur geschätzt und nicht wie sonst berechnet. Dazu wird auf die Tabelle 5.2 verwiesen. Die Risikobewertung wurde nach bestem Wissen und Gewissen durchgeführt.

Auftrittswahrscheinlichkeit	**Wertigkeit**
Unwahrscheinlich	1
Sehr gering	2-3
Gering	4-6
Mäßig	7-8
Hoch	9-10

Tabelle 5.1: Auflistungen der Wahrscheinlichkeiten eines Fehlereintrittes.

Fehlerrisiko	**Wertigkeit**	**Handlungsbedarf**
Hoch	1	Dringender Handlungsbedarf
Mittel	2	Handlungsbedarf
Akzeptabel	3	Kein zwingender Handlungsbedarf
Keines	4	Kein Handlungsbedarf

Tabelle 5.2: Auflistungen des Fehlerrisikos.

5.3 Analyseergebnisse

Dieser Abschnitt präsentiert die Analyseergebnisse. Dazu zeigt der Abschnitt die tabellarische FMEA und beschreibt den dortigen Inhalt und das Vorgehen. Der nächste Bereich beschreibt explizit die ermittelten Fehler und Gegenmaßnahmen und diskutiert die Ergebnisse. Am Ende

dieses Abschnitts folgt eine kurze Auflistung, welche Dinge bei einem Entwurf berücksichtigt werden müssen.

Die Analyse

Die gesamte Analyse kann den Abbildungen 5.1, 5.2 und 5.3 entnommen werden. Dabei zeigt die Abbildung 5.1 den Monitor als zu untersuchende Komponente, die Abbildung 5.2 den Reconfiguration Manager und die Abbildung 5.3 den Daten- und Steuerbus. Es wurde bei allen Komponenten mindestens der Ausfall geprüft. Dabei stellte ein Schwerpunkt der Analyse die Kommunikation zwischen den einzelnen Systemteilen dar. Betrachtet dazu wurden verschiedene Varianten von Deadlocks, welche durch das Ausbleiben von Nachrichten auftreten können. Außerdem stehen die Komponenten teilweise in direkter Abhängigkeit von anderen Komponenten. Daher wurde zusätzlich betrachtet, was passieren würde, wenn eine solche Komponente nicht mehr reagiert. Des Weiteren wurde bei allen Komponenten geprüft, welches Problem ein Speicherüberlauf auf diese Komponente und andere Komponenten hat. Durch das Bussystem, die interne Datenverarbeitung und das Aufteilen der Taktregion werden zwischen einzelnen Komponenten zusätzlich Buffer implementiert, jedoch auch mit dem Wissen, dass diese Überlaufen könnten. Somit wurden auch komponenteninterne Probleme betrachtet. Der nächste Abschnitt präsentiert die Ergebnisse und fasst diese exemplarisch zusammen.

Die Ergebnisse

Die Ergebnisse können in drei verschiedene Kategorien eingeordnet werden: Hardwareänderung, Systemveränderung und Modulanpassung. Die Hardwareänderung ist am schwierigsten und aufwendigsten umzusetzen und stellt beispielsweise die Umstellung auf ein anderes Entwicklerboard dar. Die Systemveränderung stellt eine Anpassung, Erweiterung oder Reduktion am bestehenden Systemvorschlag dar. Eine solche Anpassung ist aufwendig, aber umsetzbar. Die Modulanpassung gilt nur für einzelne Module und muss bei der Implementierung beachtet werden. Da diese noch nicht stattgefunden hat, muss man dazu nur einen geringen Aufwand betreiben und kann die Vorschläge ohne größere Probleme in den Entwicklungsprozess integrieren.

Für alle geprüften Komponenten sind Moduländerungen vorgesehen. Dabei stellt das häufigste Problem ein Deadlock, welcher durch die Kommunikation zwischen den einzelnen Komponenten über den Bus oder durch direkte Kommunikation untereinander erzeugt wird, dar. Durch die verteilte Architektur ist dies ein klassisches Problem. Innerhalb aller Komponenten

FMEA für Systemvorschlag							Risikobewertung	
Rerenz-nummer	Komponente	Fehlermodi	Fehlerursache	Lokale Effekte	System Effekte	Abstellmaßnahmen	Auftreten	Risiko
1	Monitor	Ausfall	- Fehler in der Implementierung - Fehler innerhalb der FPGA-Logik - SEU oder MEU durch externe Einflüsse wie Höhenstrahlung, EMV, usw.	-	- System kann keine Fehler mehr bekanntgeben bzw. diese werden nicht verarbeitet/behoben - Es wird keine Rekonfiguration getriggert - Module können nicht mehr aktiviert/deaktiviert werden	- Redundanz erforderlich	7	1
2		Interner Deadlock	- Interne Prozessantwort bleibt aus - Interne Antwort kommt zu spät	- System stagniert in einem Zustand und reagiert nicht mehr	- Erwartete Befehle oder Antworten an ander Busteilnehmer bleiben aus - Kann nicht mehr von außen angesprochen	- Timeout verwenden um endloses Warten zu verhindern	5	2
3		Externer Deadlock	- Erwartete Antwort bleibt aus - Antwort kommt zu spät	- System stagniert in einem Zustand und reagiert nicht mehr	- Erwartete Befehle oder Antworten an ander Busteilnehmer bleiben aus - Kann nicht mehr von außen angesprochen	- Timeout verwenden um endloses Warten zu verhindern	5	2
4		Falsches Eingangssignal	- Fehlerhaftes Modul am Bus	-	-	-	2	4
5		Unspezifizierte Eingangssignal	- Fehlerhaftes Modul am Bus - Falsche Version des Protokolls oder Bus	-	-	- Versionskennung im Protokoll vorsehen	2	3
6		Speicherüberlauf	- Eingangsbuffer zu klein gewählt - Ausgangsbuffer zu klein gewählt	- Informationsverlust - Monitor beginnt geplante Tätigkeit nicht	- Informationsverlust - Antworten werden nicht übertragen - Andere Komponenten beginnen die gewählte Tätigkeit nicht	- Korrekte Speichergröße wählen - Überlauf durch Speicherüberwachung verhindern	4	2
7		Fehlerhandling reagiert nicht	- Falsche Implementierung - Speicherproblem - Deadlock	- Fehlererkennung funktioniert nicht mehr	- Fehlererkennung funktioniert nicht mehr	- Redundanz erforderlich	8	2

Abbildung 5.1: Auszug der ersten Seite aus der durchgeführten FMEA. Enthalten ist die Darstellung des Monitor.

FMEA für Systemvorschlag							Risikobewertung	
Rerenz-nummer	Komponente	Fehlermodi	Fehlerursache	Lokale Effekte	System Effekte	Abstellmaßnahmen	Auftreten	Risiko
8	Reconfiguration Manager	Ausfall	- Fehler in der Implementierung - Fehler innerhalb der FPGA-Logik - SEU oder MEU durch externe Einflüsse wie Höhenstrahlung, EMV, usw.	-	- Es kann keine Rekonfiguration durchgeführt werden	- Redundanz erforderlich	7	3
9		Interner Deadlock	- Interne Prozessantwort bleibt aus - Interne Antwort kommt zu spät	- System stagniert in einem Zustand und reagiert nicht mehr	- Erwartete Befehle oder Antworten an ander Busteilnehmer bleiben aus - Kann nicht mehr von außen angesprochen	- Timeout verwenden um endloses Warten zu verhindern	4	3
10		ICAP reagiert nicht	- Fehler innerhalb der FPGA-Logik	- Es kann keine Reakonfiguration ausgeführt werden - Kein Zugriff auf die FPGA-Fabrik möglich	- Es kann keine Rekonfiguration durchgeführt werden	- ICAP zurücksetzen und neustarten - Redundanz erforderlich	1	4
11		Memory Controller reagiert nicht	- Interner Deadlock - Arbeitsspeicher defekt	- Es kann nicht die Allokationstabelle ausgelesen werden - Es kann kein Bit-File ausgelesen werden	- Es kann keine Rekonfiguration durchgeführt werden	- Timeout verwenden um endloses Warten zu verhindern	3	3
12		Fehlerhafter Datenstrom	- Arbeitsspeicher defekt	- Auswahl der falschen Quelldaten	- Partition wird falsch rekonfiguriert - Flasche Partition wird rekonfiguriert	- ECC-Speicher verwenden	1	3
13		Speicherüberlauf	- Eingangsbuffer zu klein gewählt - Ausgangsbuffer zu klein gewählt	- Informationsverlust - Geplante Tätigkeit wird nicht ausgeführt	- Informationsverlust - Antworten werden nicht übertragen - Andere Komponenten beginnen die gewählte Tätigkeit nicht	- Korrekte Speichergröße wählen - Überlauf durch Speicherüberwachung verhindern	4	2

Abbildung 5.2: Auszug der zweiten Seite aus der durchgeführten FMEA. Enthalten ist die Darstellung des Reconfiguration Manager.

FMEA für Systemvorschlag							Risikobewertung	
Rerenz-nummer	Komponente	Fehlermodi	Fehlerursache	Lokale Effekte	System Effekte	Abstellmaßnahmen	Auftreten	Risiko
14	Datenbus (Arbiter)	Ausfall	- Fehler in der Implementierung - Fehler innerhalb der FPGA-Logik - SEU oder MEU durch externe Einflüsse wie Höhenstrahlung, EMV, usw.	-	- Das gesamte System ist Arbeitsunfähig	- Redundanz erforderlich	7	1
15		Interner Deadlock	- Interne Prozessantwort bleibt aus - Interne Antwort kommt zu spät	- System stagniert in einem Zustand und reagiert nicht mehr	- Erwartete Befehle oder Antworten an ander Busteilnehmer bleiben aus - Kann nicht mehr von außen angesprochen	- Timeout verwenden um endloses Warten zu verhindern	5	3
16		Timeslot-Signal fehlerhaft	- Timer wurde falsch parametrisiert - Berechnungsfehler in der Timerlogik - Wahl der falschen Registerbreite	-	- Erwartete Befehle oder Antworten an ander Busteilnehmer bleiben aus - Das gesamte System ist Arbeitsunfähig	- Redundanz erforderlich	2	2
17	Steuerbus (Arbiter)	Ausfall	- Fehler in der Implementierung - Fehler innerhalb der FPGA-Logik - SEU oder MEU durch externe Einflüsse wie Höhenstrahlung, EMV, usw.	-	- Das gesamte System ist Arbeitsunfähig	- Redundanz erforderlich	7	1
18		Interner Deadlock	- Interne Prozessantwort bleibt aus - Interne Antwort kommt zu spät	- System stagniert in einem Zustand und reagiert nicht mehr	- Erwartete Befehle oder Antworten an ander Busteilnehmer bleiben aus - Kann nicht mehr von außen angesprochen	- Timeout verwenden um endloses Warten zu verhindern	5	3
19		Timeslot-Signal fehlerhaft	- Timer wurde falsch parametrisiert - Berechnungsfehler in der Timerlogik - Wahl der falschen Registerbreite	-	- Erwartete Befehle oder Antworten an ander Busteilnehmer bleiben aus - Das gesamte System ist Arbeitsunfähig	- Redundanz erforderlich	2	2

Abbildung 5.3: Auszug der dritten Seite aus der durchgeführten FMEA. Enthalten ist die Darstellung des Daten- und Steuerbus.

könnte es dadurch zu einem endlosen Warten auf eine Antwort oder einen bestimmten Befehl kommen.

Einem solchen Fehler kann man durch das Hinzufügen eines Timeouts einfach und unkompliziert entgegenwirken. Daher sollte dieses Problem bei der Implementierung berücksichtigt werden, um dadurch einen Ausfall einzelner Komponenten oder infolgedessen mehrerer Komponenten oder dem ganzen System vorzubeugen. Im Idealfall sollte für einen solchen Fehler eine Art Quittierung vorgesehen werden, damit auch andere Systemteilnehmer über diese Problematik informiert werden. Auch die internen Speichergrößen können durch einen Überlauf dazu beitragen, dass Fehler und unnötige Wartezustände innerhalb des Systems auftreten. Um dies zu verhindern, muss auf der einen Seite die Speicherbreite groß genug gewählt werden, sodass ein Überlauf nicht eintreten sollte. Zusätzlich kann eine Möglichkeit zu Speicherüberwachung vorgesehen werden, welche im Falle eines Überlaufs ebenfalls eine Nachricht versendet. Ein nicht generisches Problem betrifft ausschließlich den Reconfiguration Manager. Dieser besitzt den exklusiven Zugriff auf den ICAP. Es sollte eine Möglichkeit zum Zurücksetzen und Neustarten dieser Schnittstelle vorgesehen werden, falls diese nicht mehr reagiert.

Des weiteren ist für alle Komponenten eine Systemänderung notwendig. Durch die Analyse konnte ermittelt werden, dass der Ausfall der geprüften Komponenten nur durch eine identische Komponente abgefangen werden kann. Dies gilt insbesondere für die Bussysteme. Wenn eines der Bussysteme ausfällt oder ein falschen Zeitslot-Signal generiert, führt dies unweigerlich zum kompletten Systemausfall. Daher ist hier eine Redundanz unabdingbar, auch um bestimmte Fehler überhaupt erkennen zu können. Ähnliches gilt für den Monitor. Ein Ausfall kann auch an diesem Beispiel nur durch eine Redundanz kompensiert werden. Ohne den Monitor wäre eine Rekonfiguration im aktuellen Systemvorschlag nicht mehr möglich, wodurch eines der wichtigsten Ziele der zu entwickelnden Infrastruktur nicht mehr erreicht werden kann. Des weiteren kann der Ausfall des Fehlerhandlings nur durch eine Redundanz kompensiert werden. Ohne dieses Submodul innerhalb des Monitors, wäre ein weiteres wichtiges Ziel nicht mehr zu erreichen. Außerdem würde dieser Ausfall eine Fehlerfolge verursachen, wodurch zum einen keine Fehler mehr quittiert werden und letztendlich keine Rekonfiguration mehr ausgeführt werden würde. Auch der Reconfiguration Manager kann vor einem Ausfall nur durch eine Redundanz geschützt werden. Ein Ausfall dieser Komponente würde ebenfalls dazu führen, dass keine Rekonfiguration mehr durchgeführt werden kann. Das liegt daran, dass der Reconfiguration Manager exklusiven Zugriff auf den ICAP hat. Eine weitere Systemänderung wird durch die Protokolländerung indiziert. Da durch das Hinzufügen einer Versionskennung alle beteiligten Komponenten und die Protokollimplementierung angefasst werden müssen.

Außerdem müssen alle Komponenten um eine Versionserkennung erweitert werden.

Ein weiterer identifizierter Fehlerfall wird durch ein Hardwaremodul hervorgerufen. Um einen fehlerhaften Datenstrom, beziehungsweise allgemein fehlerhafte Zustände im DDR-Speicher zu verhindern, wäre es erforderlich, zum Beispiel ein ECC-Speicher zu verwenden. Dieser kann automatisch SEU und MEU ermitteln und diese beheben. Dadurch kann sichergestellt werden, dass die Konfigurationsdaten und die Allokationstabelle nicht beschädigt werden. Diese Änderung wird nicht weiter in den Systementwurf einfließen, sollte aber in zukünftigen Weiterentwicklungen berücksichtigt werden.

Die Risikokennzahlen wurden bisher nicht weiter erwähnt. Diese sollen eine Sortierung der Wichtigkeiten ermöglichen. Durch das Betrachten der Tabelle können daher Prioritäten vergeben werden. Die folgende Liste zeigt in absteigender Reihenfolge alle notwendigen Änderungen nach Priorität:

- Redundanten Monitor im Systementwurf vorsehen
- Redundantes Bussystem im Systementwurf vorsehen (Steuer- und Datenbus)
- Kommunikations-Timeouts in den verschieden Komponenten vorsehen
- Speichergrößen der Buffer innerhalb der Komponenten richtig wählen
- Speicherüberwachung je Komponente vorsehen, um Überläufe zu vermeiden
- Versionskennung im Steuerprotokoll vorsehen
- ECC-Fähiger DDR-Speicher für das Zwischenspeichern der Konfigurationsdaten und Allokationstabelle

6 Systemdesign

In diesem Kapitel wird die Anpassung des Systemvorschlags aus Kapitel 4 auf Basis der FMEA im Kapitel 5 präsentiert und diskutiert. Im ersten Abschnitt wird die Systemerweiterung erläutert und präsentiert. Dabei geht es im Detail um das Hinzufügen der Redundanzen. Der nächste Abschnitt beschäftigt sich im Detail mit der Erweiterung des Protokolls und die neuen Befehle. Der letzte Abschnitt diskutiert die Vor- und Nachteile die durch den Systementwurf entstehen.

6.1 Systemanpassung

Dieser Abschnitt beschreibt die Systemanpassung die der Abbildung 6.1 entnommen werden kann. Die wichtigsten Änderung stellen die Redundanzen dar. Es wurden beide Bussysteme durch ein identisches verstärkt und dadurch redundant ausgelegt. Zusätzlich wurden die Systemkomponenten Monitor, Scheduler und Synchronous Manager jeweils um eine identische Komponente verstärkt, wodurch auch diese redundant vorhanden sind. Durch die Erweiterung der Bussysteme müssen an allen beteiligten Komponenten zusätzliche Schnittstellen bereit gestellt werden. Dadurch ist es allen Komponenten möglich, auf beiden Steuer- und Datenbussen zu Lesen und zu Schreiben. Zusätzlich muss durch eine Systemkomponente festgelegt werden, welche der Bussysteme verwendet werden soll. Dies soll nach dem Starten zunächst von der Komponente Startup durchgeführt werden. Danach kann der aktive Monitor zwischen den Bussen wechseln. Dadurch wird bereits eine weitere Problematik deutlich. Die Startup Komponente muss zu Beginn festlegen, welcher Monitor aktiv und welcher passiv ist. Zur Laufzeit können die Monitore untereinander wechseln und sich so ablösen.

Die grundsätzliche Idee hinter der Redundanz stellt die Trennung der Steuerbusse und Datenbusse sowie der Monitore, Scheduler und Sychronous Manager in jeweils zwei separate Systemsteuerungen dar. Dabei ist immer eine Systemsteuerung aktiv und die andere passiv, wodurch eine Systemsteuerung immer im Hot-Standby ist. Dabei führt die passive Systemsteuerung immer die identischen Tätigkeiten aus, wie es die aktive Systemsteuerung durchführt, und überträgt die Daten über den passiven Steuerbus. Durch das Vergleichen des aktiven und

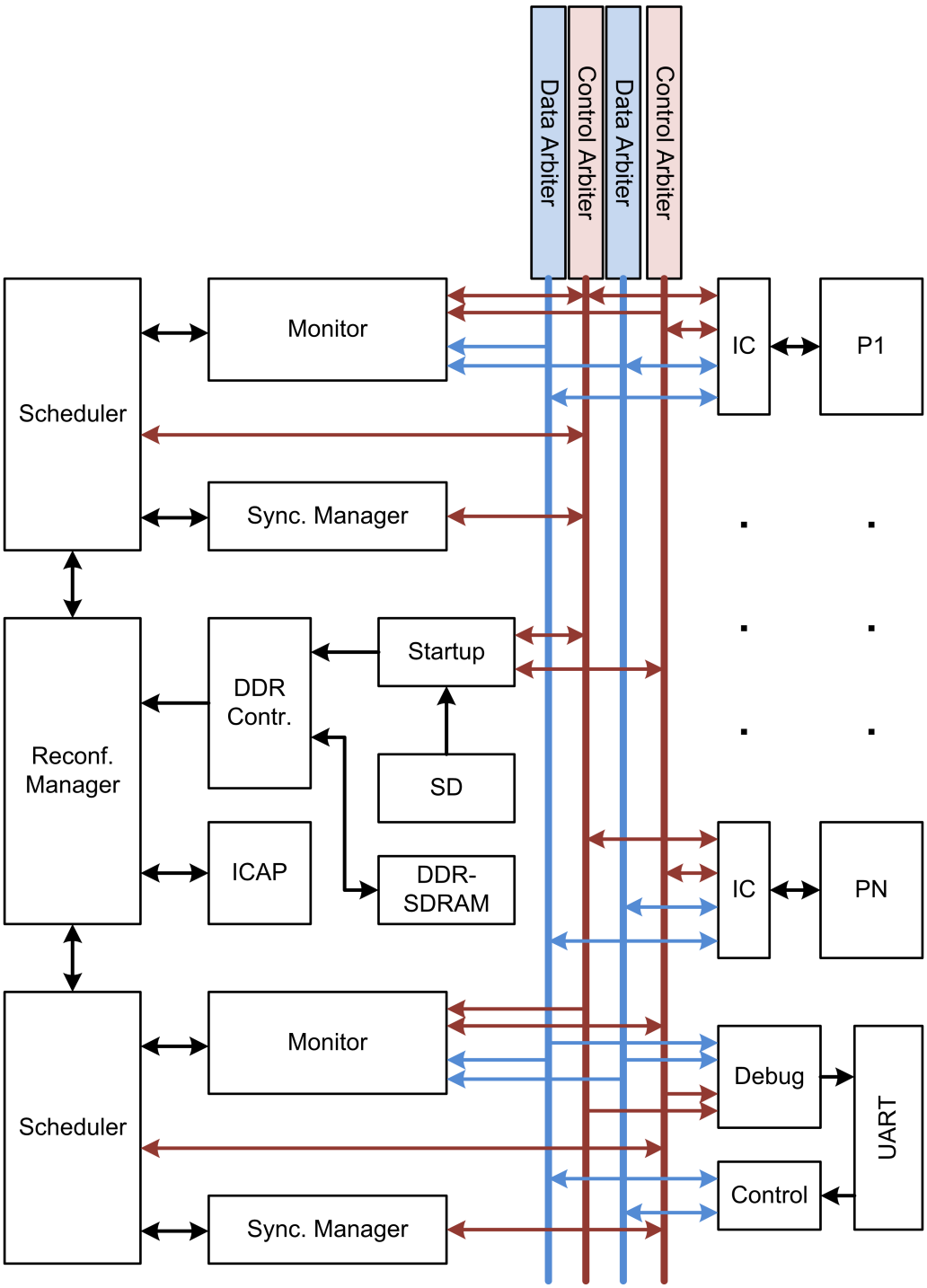

Abbildung 6.1: Darstellung der Infrastruktur mit allen funktionalen Komponenten und dem Bussystem nach der Anpassung durch ein Redundanzkonzept.

passiven Steuerbus ist die Fehlererkennung und Identifizierung erst möglich. Die Erkennung wird jeweils von den Monitoren der Systemsteuerungen durchgeführt. Zusätzlich sind die einzelnen Partitionen dazu angehalten, immer beide Busse zu beschreiben, jedoch nur auf den aktiven zu hören und dessen Befehle umzusetzen. Dadurch ist ein Parallelsystem vorhanden, welches zunächst nur der Überwachung dient. In einem Fehlerfall, welcher nicht anderes behoben werden kann, wird ein Wechsel durchgeführt. Durch Hot-Standby stellt dies kein Problem dar und die Tätigkeit kann direkt übernommen werden, selbst während des Betriebs. Dabei muss jedoch beachtet werden, dass nur der jeweils aktive Scheduler mit dem Reconfiguration Manager kommuniziert. Der Reconfiguration Manager stellt in dem Entwurf einen Flaschenhals dar und ist nicht wie in der Analyse empfohlen redundant. Dabei ist der Grund ein rein technischer. Die Anzahl der ICAP variiert von FPGA zu FPGA. Laut dem Hersteller ist eine Co-Existenz und eine parallele Verwendung nicht empfohlen, auch wenn mehrere ICAP vorhanden sind. Daher ist es momentan nur möglich eine dieser Schnittstellen zu verwenden. Daher wurde in diesem Entwurf entschieden, den Reconfiguration Manager auch nur einmal zu instantiieren. Dieser stellt einen Kommunikationskanal zu beiden Systemsteuerungen bereit, kann aber nur von einer zur Zeit verwendet werden. Da der Manager keinerlei Kenntnis über den Systemzustand hat, ist die Verantwortung an die Scheduler ausgelagert, mitzuteilen welche Systemsteuerung aktiv und welche passiv ist. Durch diesen technischen Aspekt wurde entschieden hierfür keine Redundanz vorzusehen. Für zukünftige Systeme könnte dies einen Erweiterung darstellen, wenn die Probleme des ICAP gelöst werden können. Auf einen ECC-Speicher und die Redundanz des Speichercontrollers wurde ebenfalls verzichtet. Dadurch, dass auf der gewählten Hardware nur ein DDR-Speicher-Modul vorhanden ist, wird es nicht notwendig sein, einen zweiten Speichercontroller zu instantiieren. Dadurch ist es zudem nicht möglich, auf ECC-Speicher zu wechseln.

Die Startup-Komponente bleibt von der Redundanz unberührt. Diese wird nur zum Starten des Systems benötigt und schaltet sich nach einem erfolgreichem Start ab. Zu dieser Zeit befindet sich das System noch in einer sicheren Umgebung und hat noch nicht mit dem Bearbeiten von missionspezifischen Daten begonnen. Das bedeutet, wenn im schlimmsten Fall der Systemstart scheitert und der Grund dafür die Startup-Komponente ist, stellt man dies frühzeitig fest und das System würde unter keinen Umständen mit der Mission beginnen. Daher wäre eine Redundanz an dieser Stelle vollkommen unnötig. Die Implementierung der Timeouts und die Anpassung sowie Überwachung der internen Buffer-Speicher erfolgt im Kapitel 7, da es sich um eine konkrete Implementierungsfrage der einzelnen Komponenten handelt. Dabei findet die Fehlerbehandlung in jeder einzelnen Komponente statt.

6.2 Protokollanpassung

Um die im Abschnitt 6.1 präsentierte Systembeschreibung umzusetzen und die Anforderungen aus der Analyse zu berücksichtigen, wurde das Protokoll um insgesamt drei Befehle erweitert: *"PROTOCOL_VER"*, *"SWITCH_SYS"* und *"HARD_FAILURE"*. Die Tabelle 6.1 zeigt eine Übersicht über die neuen Befehle. Der Befehl *"PROTOCOL_VER"* wird nach dem Start-Signal als erstes von allen am Bus beteiligten Modulen versendet. Dabei prüfen die Monitore, ob veraltete oder bereits neuere Busteilnehmer im Bus vorhanden sind. Ziel dabei ist es, zu prüfen, ob alle Komponenten und Module des Systems zueinander kompatibel sind. Es soll verhindert werden, dass ein schwerwiegender Fehler durch eine veraltete Komponente oder durch neue Befehle einer zu aktuellen Komponenten für Fehler im System sorgen. Durch den Befehl *"HARD_FAILURE"* kann der Monitor das System wieder abschalten. Dieser muss von beiden Monitoren nahezu gleichzeitig auf beiden Steuerbussen präsent sein. Dadurch kann das System direkt und ohne Rückkehrmöglichkeit abgeschaltet werden. Dies stellt eine Art *"Not-Aus"* dar. Mithilfe des Befehls *"SWITCH_SYS"* ist ein gezielter Wechsel zwischen den beiden Systemsteuerungen möglich. Für den *"SWITCH_SYS"*-Befehl muss eine gültige System-ID in der Payload übertragen werden. Danach wechseln alle Busteilnehmer direkt auf den Bus der gewählten Systemsteuerung. Dies gilt für den Daten- und Steuerbus gleichermaßen. Der Abbildung 6.2 kann zusätzlich der schematische Ablauf des Protokolls entnommen werden.

Kommando	**Beschreibung**	**Hexcode**	**Typ**
SYNC_TSLOT_CMP	Zusätzliche Zeitslot wurden übertragen.	15	Data Synchronization
SCHEDULE_CMP	Nach Abschluss des Schedulings der Rekonfiguration.	16	Reconfiguration
PROTOCOL_VER	Übermitteln der Protokoll Version über den Bus. Version ist in der Payload enthalten.	17	Generic
SWITCH_SYS	Signalisiert den Wechsel auf eine andere Systemsteuerung. Systemsteuerung wird explizit über die Payload gewählt.	18	Generic
HARD_FAILURE	Direktes Abschalten aller Systemkomponenten und Module.	20	Generic
RESERVED	Dieser Bereich ist für zukünftige Kommandos reserviert.	21-31	-

Tabelle 6.1: Auflistungen der zusätzlichen Kommandos und dessen Beschreibung nach der Systemerweiterung und Einführung des Redundanzkonzeptes.

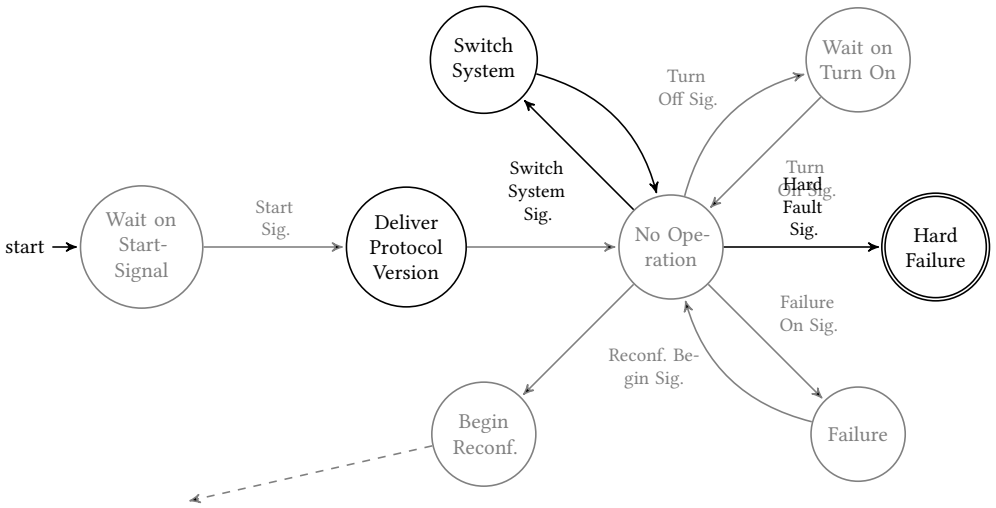

Abbildung 6.2: Schematische Darstellung der Protokollerweiterung für die Kommunikation zwischen den einzelnen Modulen des Systems. Hervorgehobene Elemente wurden aus dem Basisentwurf übernommen.

6.3 Vor- und Nachteile der Anpassungen

Dieser Abschnitt beschreibt die Vor- und Nachteile die durch die Systemanpassungen entstanden sind. Durch die verschiedenen Anpassungen sollte das System sicherer sein. Es kann jetzt auch den Ausfall von Teilen der Systemsteuerung kompensieren, da die Tätigkeiten direkt von der anderen Systemsteuerung übernommen werden können. Mit diesem Systementwurf ist letztlich nur eine Fehlererkennung implementiert, also eine klassische DMR. Daher ist ein wechseln zwischen den redundanten Komponenten möglich, aber es kann nicht beurteilt werden, welche Komponente fehlerhaft arbeitet. Der Fehler kann nur erkannt, aber nicht zugeordnet werden. Einen Ausfall einer oder mehrerer Komponenten wiederum kann durch den Entwurf kompensiert werden. Durch die Fehlererkennung ist es zudem möglich, dass die missionsabhängigen Module bei der Erkennung eines Fehlers innerhalb der Systemsteuerung eine Art Notfallprotokoll starten. Dies wäre beispielsweise das Notlanden eines Flugsystems oder ähnliches. Daher ist aus gesamter Sicht eine Erhöhung der Sicherheit gegeben. Des Weiteren konnten Probleme identifiziert werden, welche vollständig von der Implementierung abhängig sind. Dazu gehören beispielsweise die Timeouts oder auch Speicherlecks. Ein solcher Fehlermodi kann einen Gesamtausfall zur Folge haben. Daher ist es ein positives Ergebnis, das die Problemstellen identifiziert werden konnten und im nächsten Kapitel in der Implementierung berücksichtigt werden. Durch die Anpassungen am System hat sich jedoch auch die

Systemkomplexität stark erhöht. Das bedeutet, dass dadurch neue noch nicht identifizierte Ausfallmodi im System vorhanden sein können. Zudem ist durch die Redundanz niemals eine vollständige Ausfallsicherheit gegeben. Es besteht immer noch ein Restrisiko. Des Weiteren ist der Implementierungsaufwand für die redundanten Objekte und der zusätzlich notwendigen Kommunikation eine neue Fehlerquelle. Da durch die zusätzlichen Implementierungen mehr Chipfläche nötig ist, können auch äußere Einflüsse einen stärkeren Faktor auf die Sicherheit haben.

7 Implementierung und Test

Dieses Kapitel beschreibt die Implementierung des System der Kapitel 4 und 6 unter Berücksichtigung der Ausfallmodi der Analyse aus dem Kapitel 5. Dazu beschreiben die ersten drei Abschnitte die Implementierung der verschiedenen Komponenten, also der Systemsteuerung, des Redundanzkonzeptes und der externen Steuerung sowie der Debugschnitstelle. Die letzten beiden Abschnitte beschreiben die Evaluierung des Systems. Dazu werden im ersten Abschnitt ausgewählte Simulationsergebnisse beschrieben und diskutiert. Der zweite Abschnitt beschreibt eine Teilimplementierung einer Testapplikation auf dem Entwicklungsboard.

7.1 Sytemkomponenten

Dieser Abschnitt beschreibt die Implementierung der verschiedenen Systemkomponenten. Dabei wird jeweils der interne Aufbau angeschnitten und die Schnittstellen nach außen werden definiert und beschrieben. Zusätzlich werden Komponenten-Spezifische Konfigurationen und Abweichungen vom Systemdesign aufgezeigt. Für jede Komponente ist intern eine Memory Management Unit und eine Timeout Managament Unit vorgesehen. Diese wurden aber nur teilweise in wenigen Komponenten implementiert, da diese für das erste Design nicht notwendig waren. Daher werden diese Submodule nicht in den Abbildungen aufgeführt und es wird bei keiner Erläuterung auf dies Submodule eingegangen. Dabei soll der nächste Abschnitt kurz erläutern was deren Aufgabe wäre, damit dies in zukünftigen Entwicklungen berücksichtigt werden kann. Die Memory Management Unit soll die internen Buffer überwachen, um Überlaufe festzustellen und zu verhindern, wodurch einer Fehlfunktionen vorgebeugt wird. Die Timeout Management Unit soll interne Deadlocks verhindern und diese gegebenenfalls wieder auflösen, um dadurch ein mögliches Fehlerverhalten zu verhindern.

Monitor

In diesem Bereich wird die Implementierung des Monitors beschrieben. Der Abbildung 7.1 kann dazu der schematische Aufbau und die Schnittstellen des Monitors entnommen werden. Der Monitor hat Zugriff auf beide Steuer- und Datenbusse. Welchen dieser beiden Busse ver-

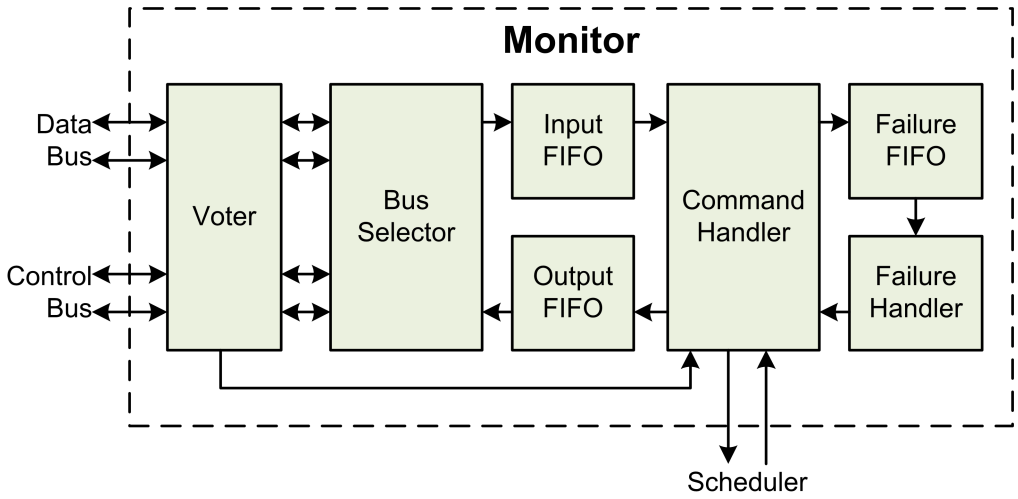

Abbildung 7.1: Schematischer Aufbau des Monitors und Darstellung der Schnittstellen zu anderen Komponenten.

wendet werden, wird durch den Busselector entschieden. Durch den Selector ist im inneren des Monitors logisch nur ein Bus sichtbar. Vor dem Selector wurde ein Voter implementiert. Dieser prüft die Busse auf Datenintegrität und gibt einen erkannten Fehler an den Command Handler weiter. Dabei werden die Busse auf Gleichheit geprüft. Zusätzlich sind bereits Stubs vorgesehen, welche das Einhalten der Zeitslots prüfen und ein weiterer Stub zum Prüfen einer fehlerhaften Komponente auf dem Bus, welche sich beispielsweise durch ständige Datenübertragung auszeichnet. Beim Datenbus wird zudem geprüft, ob auf beiden Leitungen zur gleichen Zeit ein identisches Verhalten vorliegt, da ansonsten ein Fehler des Datenbus vorliegt. Identisch wird der Steuerbus kontrolliert. Für den Steuerbus existiert eine zusätzliche Kontrolle, welche die Kommondos der Systemsteuerung miteinander vergleicht und dadurch prüfen kann, ob eine der Systemsteuerungen fehlerhaft ist. Durch das Timeslotverfahren liegen die Daten zu unterschiedlichen Zeiten vor. Daher wurde innerhalb des Voters ein Buffer erzeugt, welcher die Daten über zwei gesamte Zyklen aller Zeitslots betrachtet und erst dann auf Gleichheit prüft. Dadurch hatte jedes Modul und jede Komponente, welche mit dem System verbunden wurde, mindestens einmal eine Sendeberechtigung und somit die Möglichkeit, die zu erwartenden Kommandos abzusetzen. Wenn innerhalb des Voters ein Fehler gefunden wurde, wird dieser direkt an den Command Handler weitergeleitet. Alle Daten, die den Busselector passieren müssen, werden vorher in einem FIFO gespeichert. Das gilt für ein- und ausgehende Daten. Der Command Handler prüft, ob entweder Daten des Voters, Failure Handler oder Daten im FIFO vorhanden sind und beginnt gegebenenfalls mit der Arbeit. Die Abbildung 7.2 zeigt dazu

schematisch den Ablauf. Der Monitor wartet zunächst auf die Systemfreigabe und überträgt

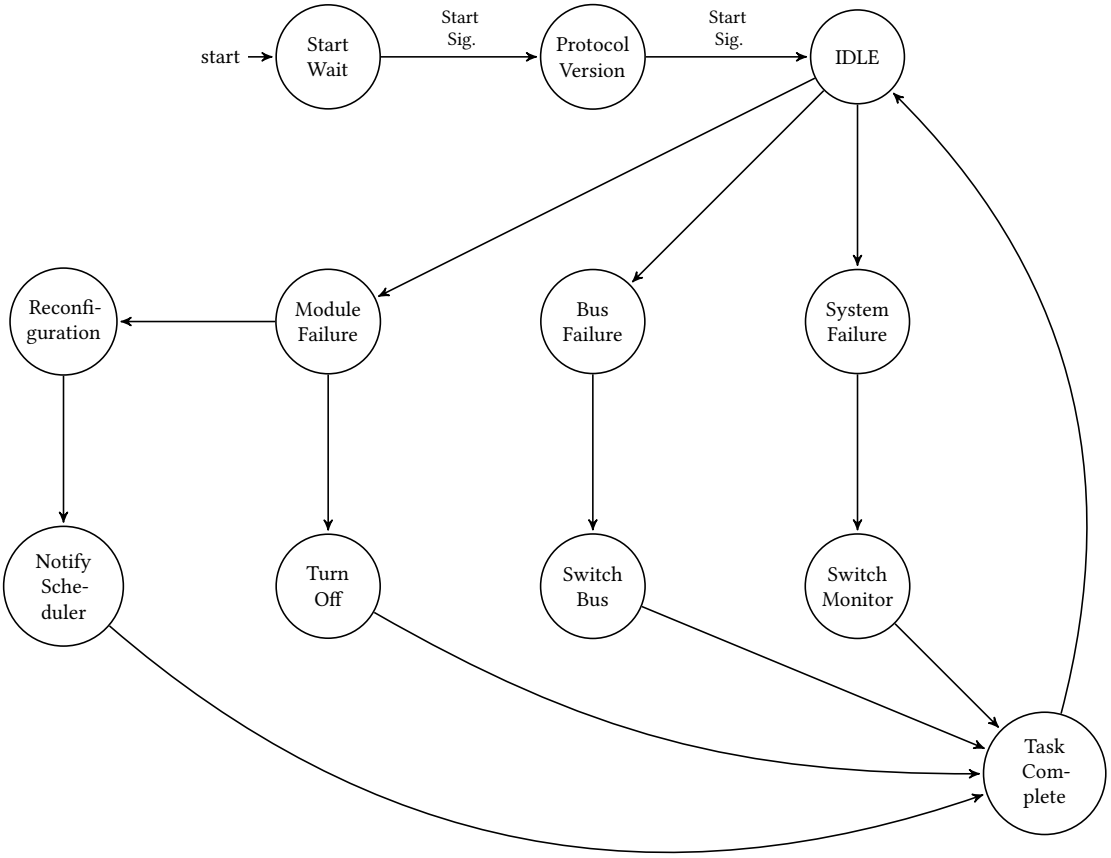

Abbildung 7.2: Schematische Darstellung des Ablaufes innerhalb des Command Controller des Monitors.

dann seine Protokoll-Version. Danach wartet er in einem IDLE-Prozess auf Input-Daten vom Steuerbus, dem Voter oder dem Failure Handler. Wenn ein Fehler vom Voter oder via Kommando den Monitor erreicht, leitet er die Informationen an den Failure FIFO weiter. Der Failure Handler prüft, ob Daten im FIFO enthalten sind und entscheidet, welche Gegenmaßnahme nötig ist. Zur Zeit akkumuliert er die Fehlerhäufigkeit je Modul, um nach dem Überschreiten eines definierten Grenzwertes das Modul zu verschieben. Welche Tätigkeit ausgeführt werden soll, wird dem Command Hanler mitgeteilt und dieser beauftragt, bei einer Rekonfiguration, den Scheduler. Wenn ein Modul nur aktiviert oder deaktiviert werden soll, wird dies direkt vom Monitor durchgeführt. Der Failure Handler kann einfach durch andere Logik ersetzt

werden. Dadurch kann eine deutlich umfangreichere Fehlerbehandlung integriert werden. Das bedeutet, dass der Monitor nur die MID an den Scheduler weitergibt.

Scheduler

Der schematische Aufbau und die Schnittstellen des Schedulers können der Abbildung 7.3 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Wie bereits erwähnt, erhält der Scheduler einen Reconfiguration

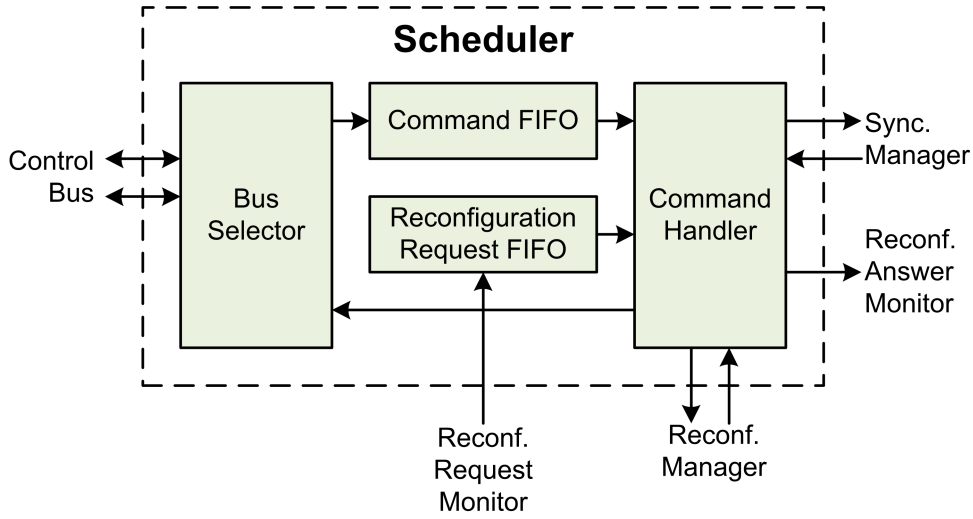

Abbildung 7.3: Schematischer Aufbau des Schedulers und Darstellung der Schnittstellen zu anderen Komponenten.

Komponenten

Request vom Monitor. Diesen Request legt er in seinen Reconfiguration Request FIFO ab. Zunächst wurde hier nur ein einfaches FIFO realisiert. Dieses ist aber austauschbar, beispielsweise durch eine Priority Queue oder ein Round-Robin-Verfahren. Prioritäten sind bereits vorgesehen, werden aber bisher nicht berücksichtigt. Auch der Scheduler besitzt einen Bus Selector und ein Command FIFO. Diese beiden Module funktionieren identisch zu den Modulen des Monitors. Der Command Handler innerhalb des Schedulers wartet auf eingehende Kommandos und Reconfiguration Requests. Dies macht er, indem er prüft, ob Daten im FIFO vorhanden sind.

Wenn eine Rekonfiguration gewünscht ist, beginnt der Command Handler mit seiner Arbeit. Zunächst prüft er, welches Modul betroffen ist und was gewünscht wird: Entfernen, Neuladen, Verschieben oder Hinzufügen. Diese Informationen werden an den Reconfiguration Manager weitergeleitet. Zusätzlich überträgt der Scheduler die MID an den Manager. Der Command Handler wartet auf den Abschluss der Rekonfiguration, durch eine Bestätigung des Reconfigu-

ration Manager. Danach werden die Informationen zum Modul an den Synchronous Manager weitergegeben. An dieser Stelle wartet der Scheduler wieder auf den erfolgreichen Abschluss der Synchronisation, durch eine Bestätigung des Synchronous Manager. Wenn diese Bestätigung eingeht, meldet er die erfolgreiche Rekonfiguration und Synchronisation dem Monitor. Zu den verschiedenen Prozessschritten überträgt der Scheduler jeweils die Informationen über den Steuerbus, damit auch alle anderen Busteilnehmer über die Rekonfiguration und dem Status dieser informiert sind. Zudem ermöglicht dies, den Monitoren Rückschlüsse über den Systemzustand zu ziehen.

Reconfiguration Manager

Der schematische Aufbau und die Schnittstellen des Reconfiguration Managers können der Abbildung 7.4 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Zentrale Element stellt hierbei der Reconfiguration

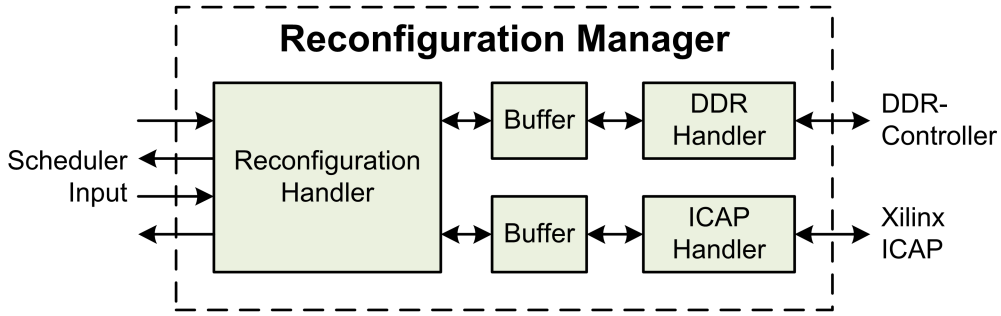

Abbildung 7.4: Schematischer Aufbau des Reconfiguration Manager und Darstellung der Schnittstellen zu anderen Komponenten.

Komponenten

Handler dar. Dieser steuert die internen Abläufe des DDR Handler und des ICAP Handler. Der Reconfiguration Handler erhält seine Informationen von einem der beiden Scheduler. Dabei gibt nur der aktive Scheduler seine Daten an den Manager weiter. Der passive Scheduler wartet nur auf den Rückgabewert, ob die Rekonfiguration erfolgreich war. Nachdem der Reconfiguration Manager die Informationen erhalten hat, welches Modul rekonfiguriert werden soll und was genau getan werden muss, beginnt er mit seiner Arbeit. Der Reconfiguration Manager pflegt eine Partitionstabelle. Diese enthält Informationen zu allen Partition, also welches Modul geladen ist und welche Partition frei ist. Zusätzlich sind in dieser Prioritäten festgelegt, die derzeit nicht berücksichtigt werden, da für die ersten Versuche ein einfachers FIFO als Schedulingverfahren implementiert wurde. Durch diese ist es möglich, in zukünftigen Entwicklungen zu entscheiden, welches Modul entfernt werden muss, um einem wichtigeren

Modul Platz zu schaffen. Der Manger sucht zunächst in der Allokationstabelle nach dem Modul mit einer passenden Zielpartition. Ein Ausschnitt aus der Tabelle kann der Tabelle 7.1 entnommen werden. Beim Entfernen oder Neuladen entspricht die Zielpartition der Quellpartition. Die Allokationtabelle ist im DDR-Speicher enthalten und muss daher über den DDR-Handler aus diesem ausgelesen werden. Dazu wird dem DDR-Handler die Start und Endadresse übergeben und der Reconfiguration Handler prüft jeden Datensatz der Tabelle, bis der passende Eintrag gefunden wurde. Hat er das richtige Modul für die richtige Partition gefunden, beauftragt der Handler erneut den DDR Handler, das Bitfile unter Angabe von Start- und Endadresse, komplett auszulesen und an den Reconfiguration Handler zu übertragen. Der Datenstrom wird an den ICAP Handler weitergeleitet. Dieser steuert die Reconfiguration und kontrolliert direkt den ICAP von Xilinx. Beim Verschieben muss dieser Prozess zusätzlich für das Entfernen des Moduls aus der alten Partition wiederholt werden, jedoch wird hier in der Allokationstabelle nicht nach dem Modul sondern nach der Leerpartition gesucht. Nachdem die Rekonfiguration erfolgreich war, quittiert der Reconfiguration Manager dieses dem Scheduler.

Partition	Modul	Startadresse	Endadresse
0	leer	0x00001001	0x00002000
0	0	0x00002001	0x00003000
0	1	0x00003001	0x00004000
1	leer	0x00004001	0x00005000
1	0	0x00005001	0x00005000
...	...	...	...

Tabelle 7.1: Auszug aus einer Allokationstabelle, welche vom Reconfiguration Manager zum Auffinden der Modul-Partitions-Zuordnung genutzt wird.

Synchronous Manager

Der schematische Aufbau und die Schnittstellen des Synchronous Managers können der Abbildung 7.5 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Im Grunde leitet und überwacht der Manager nur die Synchronisation von neu geladenen Modulen. Da die Synchronisation vollständig von der konkreten Applikation abhängig ist, stellt diese Komponente nur Hilfsmittel für die Durchführung bereit und überwacht dabei die Synchronisationszeiten und publiziert alle nötigen Informationen über den Steuerbus. Die Information, welches Modul synchronisiert werden muss, erhält der Synchronous Manager dabei von dem Scheduler. Nach dem Erhalt dieser Information erfragt der Manager zunächst, welcher Synchronisationstype gewünscht ist. Dies

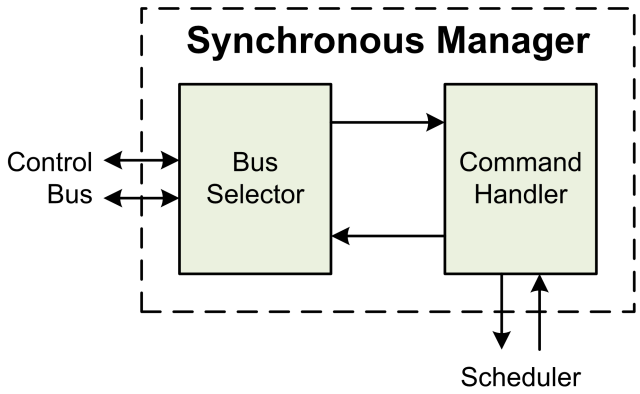

Abbildung 7.5: Schematischer Aufbau des Synchronous Manager und Darstellung der Schnittstellen zu anderen Komponenten.
Komponenten

geschieht mithilfe eines Kommandos auf der Steuerleitung und wird dabei durch eine Antwort durch das Modul dieser quittiert. Dabei gibt es drei Mögliche Synchronisation: keine, intern, extern. Bei keiner Synchronisation, quittiert der Manager direkt den erfolgreichen Abschluss beim Scheduler und das Modul wird freigegeben. Bei einer internen Synchronisation, wartet der Manager auf ein Signal vom Modul über den erfolgreichen Abschluss. Danach quittiert er den Abschluss wieder beim Scheduler. Bei einer externen Synchronisation, stellt der Manager zusätzliche Zeitslots zur Synchronisation bereit. Diese gelten dann für ein bereits laufendes identisches Modul. Dadurch können besonders große oder viele Daten schneller synchronisiert werden. Auch hier muss nach dem Abschluss das zu synchronisierende Modul, dies auf dem Steuerbus dem Manager gegenüber quittieren. Dieser gibt die Freigabe dem Scheduler weiter. Die externe Synchronisation ist bisher nur theoretisch vorgesehen und noch nicht implementiert. Für die Funktionalität sind entsprechende Stubs vorgesehen.

Startup

Der schematische Aufbau und die Schnittstellen der Startup-Komponente können der Abbildung 7.6 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Die Startup-Komponente hat ausschließlich Zugriff auf den Steuerbus. Dabei kann die Komponente beide lesen, schreibt aber nur auf einem. Diese Komponente ist für das Laden der Startkonfiguration und für die Systemfreigabe verantwortlich und wird nach dem erfolgreichen Systemstart beendet. Die Komponente hat danach keinen Einfluss mehr auf den Systemablauf. Zentrales Element stellt hier der Startup-Handler dar. Dieser wartet nach dem FPGA-Start auf Daten von der UART-Schnittstelle. Es werden nachein-

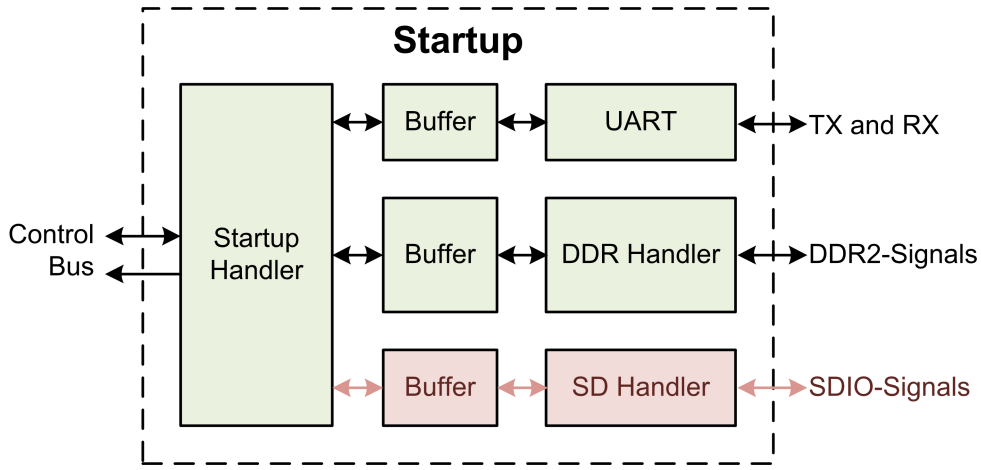

Abbildung 7.6: Schematischer Aufbau der Startup-Komponente und Darstellung der Schnittstellen zu anderen Komponenten.

Komponenten

ander ein Start-Paket und ein Payload-Paket gesendet. Dabei handelt es sich um eine große Datei, welche die Allokationstabelle, Startkonfiguration und alle Bit-Files beinhaltet. Diese Datei wird in den DDR-Speicher abgelegt. Dazu wird der DDR Handler verwendet, welcher direkt mit dem DDR-Controller kommuniziert. Vom Systementwurf ist hierfür eine SD-Karte vorgesehen. Die SDIO-Schnittstelle zu dieser und das notwendige Kommunikationsprotokoll ist noch nicht vollständig implementiert. Daher wird für die ersten Testläufe die deutlich simplere UART-Schnittstelle verwendet und erst in zukünftigen Entwicklungen das SD-Card Interface dazu kommen. Nach dem erfolgreichen Initialisieren des DDR-Speichers liest die Startup-Komponente die Startkonfiguration aus dem Speicher aus. Die Konfiguration beinhaltet alle nötigen Informationen über die initiale Konfiguration aller Partitionen. Nun wird die Startkonfiguration Datensatz für Datensatz an den Reconfiguration Controller weitergegeben, sodass dieser jede Partition zunächst laden kann. Leere Partitionen müssen dabei nicht extra geladen werden, da die Partition standardmäßig leer ist. Nachdem alle Partitionen geladen wurden, erfolgt die Systemfreigabe durch das Senden des Startsignals und das System kann nun seiner Aufgabe nachgehen.

DDR-Controller

Der schematische Aufbau und die Schnittstellen des DDR-Controllers können der Abbildung 7.7 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Der DDR-Controller wird zum einen vom Reconfiguration Manager

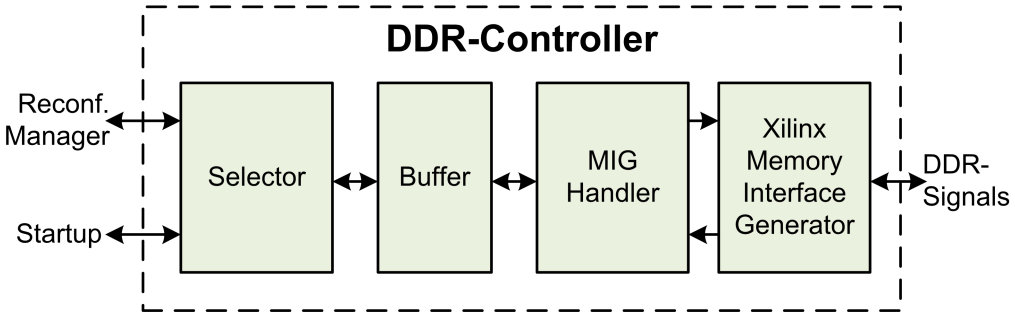

Abbildung 7.7: Schematischer Aufbau des DDR-Controller und Darstellung der Schnittstellen zu anderen Komponenten.

Komponenten

und zum anderen von der Startup-Komponente verwendet, um Daten aus dem DDR-Speicher auszulesen. Der Controller verwendet für den Zugriff auf den Arbeitsspeicher den Xilinx Memory Interface Generator für Serie-7 FPGA von Xilinx. Ausführliche Informationen zu diesem „Framework“ können dem Manual [25], [7] und [5] entnommen werden. Dabei stellt der Controller nur ein vereinfachtes Interface zum MIG bereit. Zunächst einmal wählt der Selector aus, welche Komponente exklusiven Zugriff auf den Speicher erhält. Standardmäßig ist dies die Startup-Komponente, bis diese ihre Arbeit abgeschlossen hat. Danach kann nur noch der Reconfiguration Manager zugreifen, also für die initiale Konfiguration und alle weiteren Rekonfigurationen. Der MIG-Handler implementiert einen Automaten, der über das Applikationsinterface des MIG mit dem DDR-Speicher kommuniziert. Dabei bietet dieser lediglich an, zu Schreiben und zu Lesen, unter Angabe einer Start und Endadresse. Über Statussignale erfährt der Anwender, welchen Zustand das Interface hat und ob die Daten erfolgreich geschrieben, beziehungsweise aus dem Buffer gelesen wurden. Durch die Abstraktion ist ein schneller und effizienter Zugriff möglich. Des weiteren wird innerhalb des DDR-Controller der Mixed-Mode Clock Manager integriert. Diese Komponente wird von Xilinx bereitgestellt und es können dem Manual [26] mehr Informationen entnommen werden. Diese Komponente erzeugt den Basis- und Referenztakt für das MIG-Interface und den DDR-Speicher.

7.2 Redundanzkonzept

Dieser Abschnitt beschreibt die Implementierung des Redundanzkonzeptes des Abschnittes 4.2.3. Außerdem wird der Interconnect beschrieben, welcher die Schnittstelle zwischen der Applikationsschicht und den Systembussen darstellt. Dabei wird jeweils der interne Aufbau angeschnitten und die Schnittstellen nach außen werden definiert und beschrieben.

Interconnect

Der schematische Aufbau und die Schnittstellen des Interconnect können der Abbildung 7.8 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Der Interconnect stellt die Schnittstelle zwischen der dynamischen

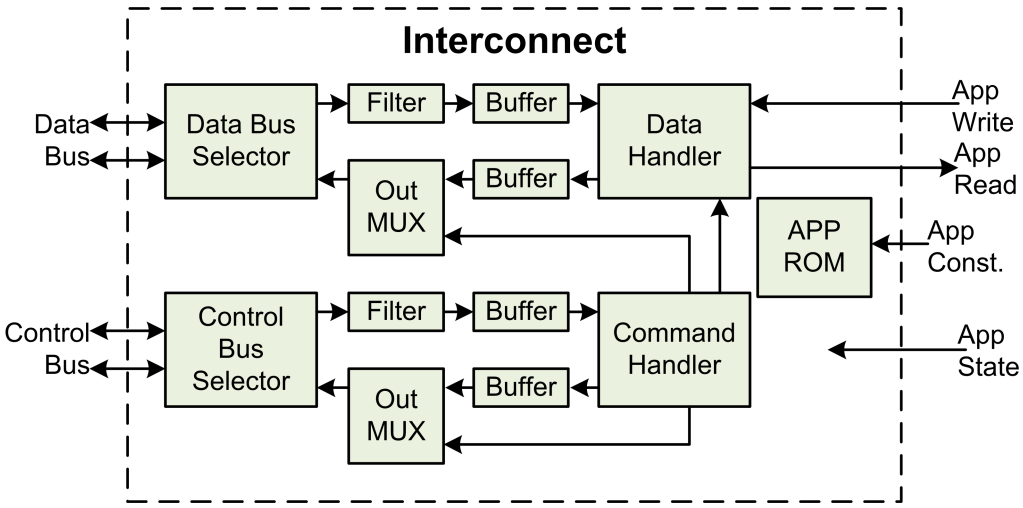

Abbildung 7.8: Schematischer Aufbau des DDR-Controller und Darstellung der Schnittstellen zu anderen Komponenten.

Komponenten

Partition und den Bussystemen dar. Die Schnittstelle für die einzelnen Applikationen ist dabei mit beiden Daten- und Steuerbussen verbunden. Um zu wählen, welcher Bus verwendet wird, ist der Data Bus und Control Bus Selector da. Durch diesen ist die Busredundanz innerhalb des Systems transparent. Nachdem die Daten und die Steuerinformationen die Selektoren passiert haben, werden diese zunächst gefiltert. Der Filter soll zunächst unwichtige Informationen filtern und nur die wesentlichen Informationen zum Data und Control Handler weitergeben. Bevor die Daten an die Handler weitergegeben werden, werden diese wieder in einem FIFO-Speicher gebuffert. Zunächst wird der Data Handler und das Lesen und Schreiben von Daten besprochen. Wenn die Applikation ein Datum versenden möchte, meldet die Applikation den Wunsch über ein Steuersignal beim Data Handler an und stellt zusätzlich die DID sowie das Datum bereit. Der Data Handler quittiert den Erhalt über eine andere Steuerleitung. Der Handler sorgt nun dafür, dass das Datum in ein Datenpaket verpackt wird und über einen weiteren FIFO-Buffer innerhalb seines Zeitslots auf dem Bussen übertragen wird. Somit ist der Datenbus für die Applikation völlig transparent und das Protokoll und der Zeitslot muss nicht weiter beachtet werden. Wenn Daten erhalten werden, sorgt der Handler im Gegenzug dafür,

dass die Applikation eine Nachricht erhält. Dies geschieht ebenfalls mit einem Steuersignal. Außerdem werden gleichzeitig die DID und das Datum bereitgestellt. Die Applikation muss den Erhalt ebenfalls quittieren. Dies wird solange wiederholt bis das Dateneingangs-FIFO geleert ist. Der Command Handler greift auf die Kommandos auf dem Steuerbus und Statusinformationen der Applikation zurück. Wenn des Kommandoeingangs-FIFO nicht leer ist, kopiert der Command Handler das Datum aus dem Speicher und analysiert den Befehl. Dabei setzt er die Kommandos um und verarbeitet diese. Zudem erzeugt er verschiedene Statusinformationen für die Applikationsschicht. Dies sind Statusinformationen, die angeben ob das Modul eingeschaltet oder abgeschaltet ist oder Rekonfigurationsinformationen anderer Module, damit sich eine abhängige Applikation darauf einstellen kann. Zusätzlich werden Informationen zur Datensynchronisation ausgetauscht. Zudem erhält auch der Command Handler Statusinformationen der Applikationen. Beispielsweise Fehlermeldungen oder Rekonfigurationswünsche. Diese werden vom Handler wieder in Kommandos umgesetzt und über ein Ausgangs-FIFO über den Steuerbus übertragen. Zusätzlich besitzt der Command Handler Zugriff auf die OUT Multiplexer. Die Multiplexer ermöglichen es dem Handler, das Modul von den Bussen zu trennen, wenn beispielsweise eine Rekonfiguration durchgeführt wird oder die Applikation deaktiviert werden soll. Dadurch gelangen keine Daten mehr auf den Bus, das Lesen wiederum funktioniert weiterhin. Ein weiterer Bestandteil ist der Application ROM. In diesem schreibt die Applikation konstante Informationen wie die MID, Filterinformationen oder Zeitslot. Diese ändern sich nur nach einer Rekonfiguration einmalig und sind innerhalb des Interconnects von allen Submodulen lesbar. Dadurch ist der Interconnect für jede Applikation generisch.

Voter

Der schematische Aufbau und die Schnittstellen des Voters können der Abbildung 7.9 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Der Voter stellt hier einen Implementations-Vorschlag für eine Applikation dar. Dieser ist über den Interconnect mit dem Bus verbunden und er kann rekonfiguriert werden. Die Applikationsdaten werden in einem Buffer zwischengespeichert. Dies gilt sowohl für die Eingangs- als auch die Ausgangsdaten. Das gleiche gilt für die Applikationszustände. Diese werden vom Redundancy Controller verwaltet. Die Eingangsdaten werden vom Redundancy Controller an einen Zwischenspeicher weitergegeben. Dieser speichert die Werte in einer Art Tabelle mit der MID und DID als Indikator. Dabei werden immer die Werte von dem aktuellen Zyklus zwischengespeichert. Der Discrepapancy Sensor arbeitet am Ende des Zyklus die Liste ab. Dabei verrechnet er die aktuellen Daten aller gleichen DIDs und erzeugt draus einen Report den er an den Controller zurückgibt. Das gleiche gilt für den Voter, nur dass

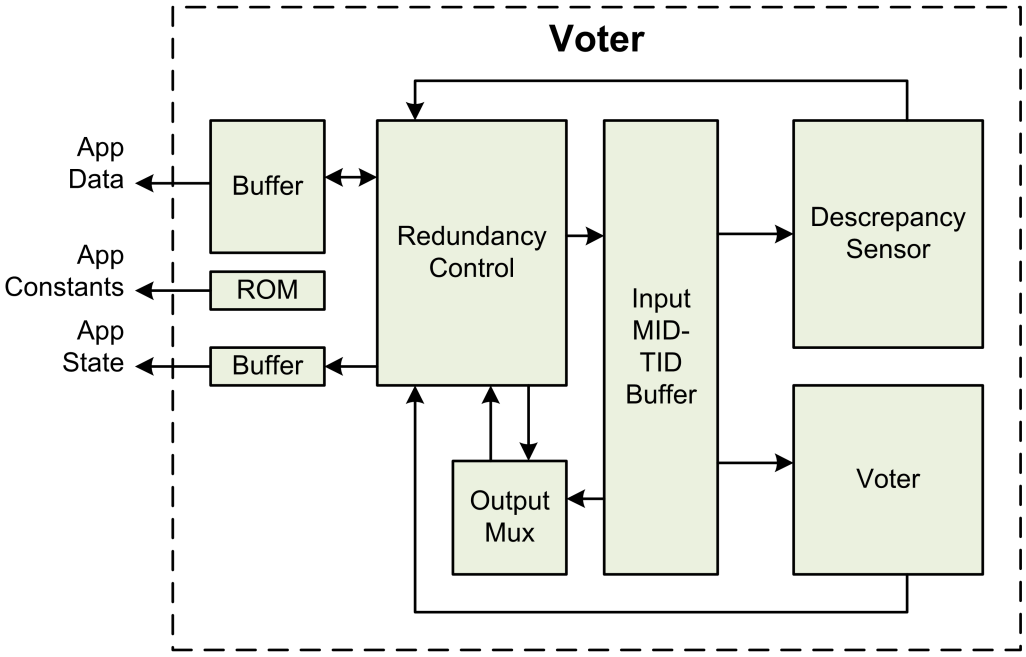

Abbildung 7.9: Schematischer Aufbau des Voters und Darstellung der Schnittstellen zu anderen Komponenten.

Komponenten

dieser ausschließlich die DIDs miteinander vergleicht und ebenfalls einen Report erzeugt. Der Voter und Sensor wiederholen diesen Vorgang für alle identischen DIDs in der Liste. Außerdem können in der Liste einzelne MIDs deaktiviert werden, sodass sie in der Rechnung keine Rolle mehr spielen, wodurch eine Redundanzreduktion berücksichtigt wird. Die maximal mögliche Anzahl an Modulen und die Datentypen müssen einmalig konfiguriert werden, damit genügent Ressourcen akquiriert werden können. Dadurch ist der Voter nach oben begrenzt, aber nach unten flexibel. Wenn nur noch ein Modul zum Prüfen vorhanden ist, werden die Daten einfach weitergeben, ohne dass eine Berechnung oder ein Vergleich stattfindet. Dadurch kann Strom gespart werden. Auf Basis der Reports kann der Redundancy Controller entscheiden, welche Daten nach außen weitergegeben werden. Dazu wird der Output-Multiplexer verwendet. Außerdem kann nach einer bestimmten Fehlerhäufigkeit, ein Kommando an den Interconnect erzeugt werden, wodurch beispielsweise eine Rekonfiguration eingeleitet wird. Die Daten des ROMs werden während der Implementierungszeit eingefügt und beinhalten beispielsweise die MID des Moduls, welche dann im Interconnect zusätzlich abgelegt wird.

Modul

Der schematische Aufbau und die Schnittstellen eines Modules für die NMR können der Abbildung 7.10 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Das Modul stellt eines der simpelsten Bestandteile

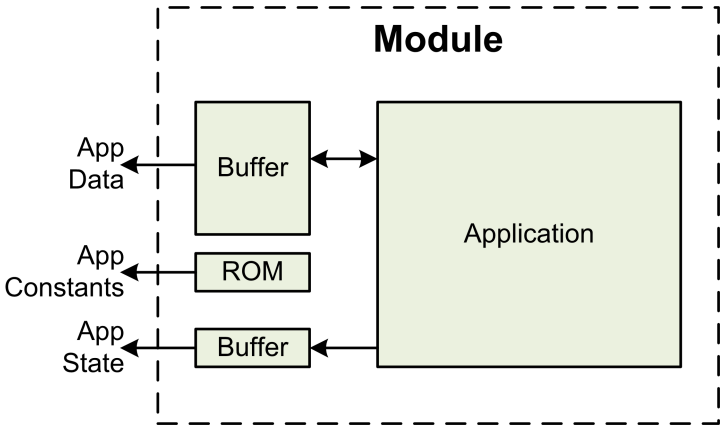

Abbildung 7.10: Schematischer Aufbau eines Modules für die NMR und Darstellung der Schnittstellen zu anderen Komponenten.

Komponenten

des Systems dar. Es verwendet ebenfalls einen Buffer um Daten zwischen der Applikation und Interconnect auszutauschen. Das ROM beinhaltet auch hier die MID und andere konstante

Informationen zum Modul. Ebenfalls existiert der Statusaustausch zwischen Interconnect und der Applikation. Für die hiesige Anwendung schreibt die Applikation zyklisch Daten auf dem Bus, welche durch einen Voter geprüft werden. Der Status wird innerhalb der Module nicht verwendet. Es existieren Abwandlungen dieser Applikation, für die Fehlerindizierung. Ansonsten ist es an dieser Stelle dem Entwickler freigestellt, welcher Inhalt in der Applikation ausgeführt wird. Es stellt hier ein simples Minimalbeispiel dar.

7.3 Steuerungs- und Debugapplikation

Dieser Abschnitt beschreibt die Implementierung der Sterungs- und Debugapplikation. Beide Module teilen sich eine UART-Schnittstelle. Dabei nutzt das Debugmodul nur die TX-Leitung und das Control-Modul die RX-Leitung. Dabei wird jeweils der interne Aufbau angeschnitten und die Schnittstellen nach außen werden definiert und beschrieben.

Debug

Der schematische Aufbau und die Schnittstellen des Debug-Interface können der Abbildung 7.11 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Die Debugschnittstelle sendet alle Daten aller Busse über

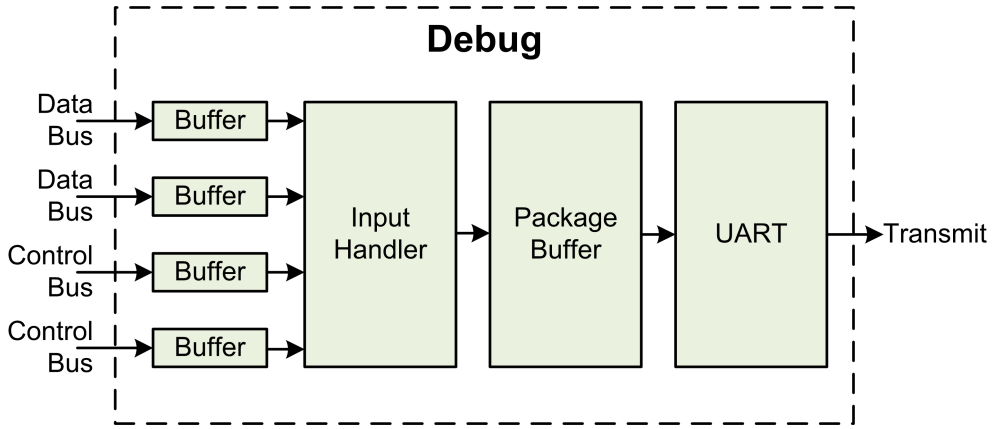

Abbildung 7.11: Schematischer Aufbau des Debug-Interface und Darstellung der Schnittstellen zu anderen Komponenten.

Komponenten

die eine UART-Schnittstelle zu einer Workstation. Zunächst werden dazu alle übertragenen Daten in einem FIFO zwischengespeichert. Für jeden Bus ist jeweils ein FIFO vorgesehen. Der Input Handler prüft durchgehend, ob Daten in den FIFOs vorhanden sind. Ist dies der

Fall, kopiert er den Inhalt und verpackt diesen in ein Paket. Der Aufbau des Pakets kann der Tabelle 7.2 entnommen werden. Das fertige Paket wird in den Package Buffer kopiert. Dieser ist ebenfalls ein FIFO. Die UART-Schnittstelle prüft durchgehend, ob ein Paket im Package Buffer enthalten ist. Sobald dort ein gültiges Paket gespeichert wurde, kopiert die UART-Schnittstelle das Datum und überträgt die Daten byteweise. Das Paket ist insgesamt 11 Byte groß.

	Kanaltyp	**Payload**
Steuerbus	0 (Steuerbus 1) oder 1 (Steuerbus 2)	Kommando, Quellmodul, Zielmodul, Payload, Zeitslot
Datenbus	2 (Datenbus 1) oder 3 (Datenbus 2)	Adresse (Quell- und Zielmodul, Quell- und Ziel-TID), Datum, Zeitslot

Tabelle 7.2: Aufbau des Debugpaketes für den Steuer- und Datenbus.

Control

Der schematische Aufbau und die Schnittstellen des Control-Interface können der Abbildung 7.12 entnommen werden. Dabei sind jeweils die internen Module und die internen sowie externen Schnittstellen angegeben. Die Control-Schnittstelle ermöglicht die Kontrolle des

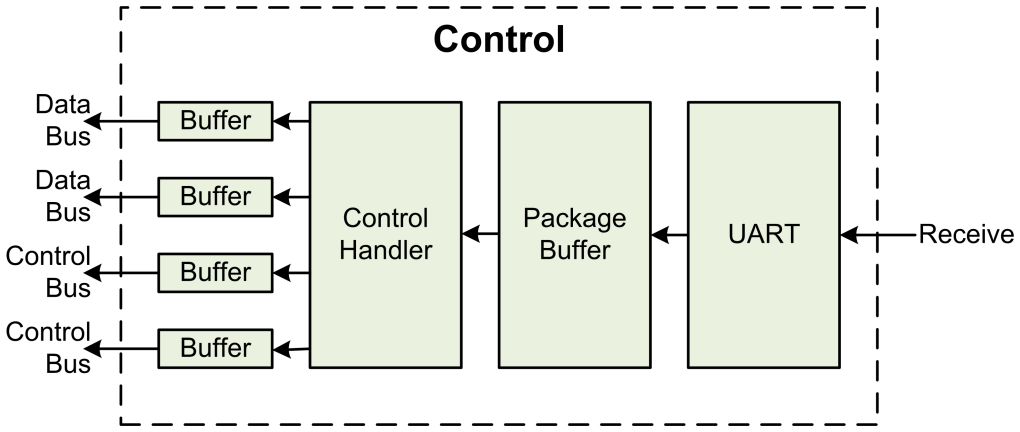

Abbildung 7.12: Schematischer Aufbau des Control-Interface und Darstellung der Schnittstellen zu anderen Komponenten.

Komponenten

Systems von einer Workstation aus. Diese Schnittstelle wird für die Fault Injection oder dem direkten Rekonfigurieren verwendet. Dabei wird jeder Befehl ohne eine Antwort übertragen. Ob der Befehl richtig angekommen ist, kann über die Debug-Schnittstelle herausgefunden werden, da mit jedem Befehl entweder ein Datum auf dem Datenbus übertragen wird oder

ein Kommando auf dem Steuerbus. Die Pakete werden über die UART-Schnittstelle an das Interface übertragen und dort in einem Package Buffer zwischengespeichert. Der Package Buffer entspricht dabei einem FIFO. Der Control Handler prüft durchgehend, ob ein Paket im Package Buffer vorhanden ist. Wenn ein Paket vorhanden ist, wird dieses in den Control Handler kopiert. Der Handler analysiert zunächst das Paket. Dabei kann der Paketaufbau der Tabelle 7.3 entnommen werden. Je nach Paketinhalt werden die Daten auf einem oder beiden Steuerbussen oder Datenbussen versendet. Verwendet wurde diese Schnittstelle um gezielt falsche Daten und Kommandos auf dem Bussen zu injizieren. Des Weiteren wurden gezielt Module gegen fehlerhafte Module getauscht, um Voter zu prüfen. Hierdurch konnten die grundlegenden Funktionen des System getestet werden.

	Kanal	**Payload**
Reconfiguration Package	0 (Steuerbus 1) oder 1 (Steuerbus 2) oder 3 (beide)	MID, Rekonfigurationstypen, Quell- und Zielpartition
Data Package	0 (Datenbus 1) oder 1 (Datenbus 2) oder 3 (beide)	Adresse (Quell- und Zielmodul, Quell- und Ziel-TID), Datum
Command Package	0 (Steuerbus 1) oder 1 (Steuerbus 2) oder 3 (beide)	Kommando, Quellmodul, Zielmodul, Payload

Tabelle 7.3: Tabellarische Darstellung der verschiedenen Control-Pakete und Beschreibung des Paketaufbaus.

7.4 Simulationsergebnisse

Dieser Abschnitt präsentiert verschiedene Simulationsergebnisse, welche mithilfe von ModelSim erzeugt wurden. Dabei soll gezeigt werden, dass der Ablauf funktioniert und die Datenübertragung gewährleistet ist. Hierbei wird zunächst der Systemstart gezeigt. Danach wird eine normale Datenübertragung präsentiert. Zum Schluss wird der Ablauf einer Moduldeaktivierung gezeigt und schematisch das Verschieben eines Moduls in eine neue Partition. Letzteres kann nur schematisch präsentiert werden, da die physikalische Rekonfiguration im Moment nicht simuliert werden kann, da die vollständige Simulation momentan nicht von Xilinx unterstützt wird. Dazu wurden Stubs implementiert, welche eine Wartezeit emulieren, um ein möglichst reales Ergebnis zu zeigen. In der Simulation wurde das Zusammenspiel aller redundanten Komponenten und dessen Kommunikation und Steuerung geprüft. Dieser

Abschnitt präsentiert nur einen Auszug der Simulation. Durch diese konnte die Integration geprüft werden. Mithilfe der Simulation konnte die Funktionalität des Systems nachgewiesen werden.

Systemstart

Als Einstieg dient hier der Systemstart, der der Abbildung 7.13 entnommen werden kann. In der Abbildung ist das Simulationergebnis aus Mentor Graphics ModelSim enthalten. Für die

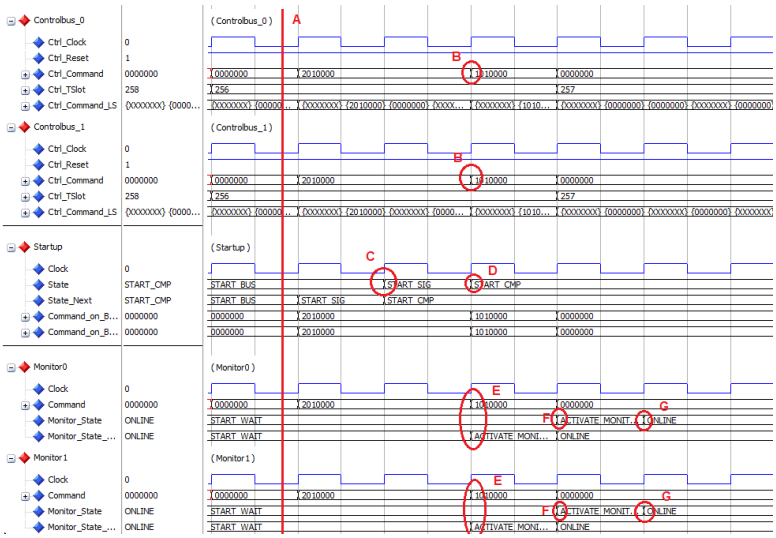

Abbildung 7.13: Simulationsergebnis des Systemstarts. Betrachtung der redundanten Steuerbusse und Monitore.

Veranschaulichung der Simulation wurden die redundanten Steuerbusse, das Startup-Modul sowie die redundanten Monitore herangezogen. Der konkrete Startvorgang beginnt kurz hinter der Markierung *"A"*. Durch das Senden des Start-Signals auf beide Steuerbusse, erhalten alle am Bus hängenden Komponenten eine Nachricht, dass ab jetzt das System betriebsbereit ist. Das Senden des Signals durch die Startup-Komponente wird durch den Kreis *"C"* markiert. Die Daten werden erst etwas später bei *"B"* auf dem Bus anliegen. Die Markierungen *"E"* zeigen, dass das Start-Signal die beiden Monitore erreicht hat. Ab diesem Moment wechseln die beiden Komponenten ihren Zustand von Wartend auf Online. Die Verfügbarkeit kann der Markierung *"G"* entnommen werden. Durch dieses Ergebnis ist in der Simulation das korrekte Startverhalten nachgewiesen.

Datenübertragung

Als nächstes betrachten wir die einfache Datenübertragung im Detail. Diese kann der Abbildung 7.14 entnommen werden. Der Beginn der Datenübertragung kann der Markierung *"A"*

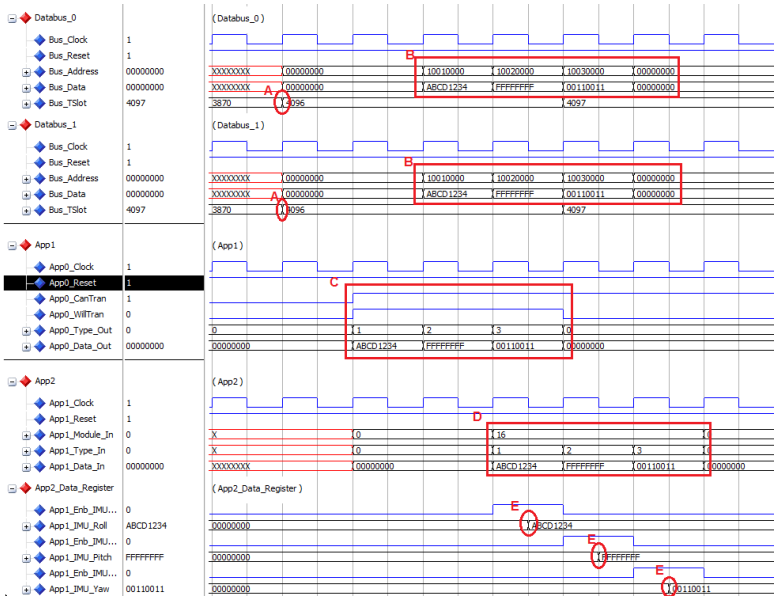

Abbildung 7.14: Simulationsergebnis einer Datenübertragung. Betrachtung einer sendenden und einer empfangenen Applikation.

entnommen werden. Hier sieht man den Übergang auf einen neuen Zeitslot. Die Partition stellt nach dem Übergang fest, dass die unterliegende Applikation sendeberechtigt ist. Dieses Signal kann innerhalb der Markierung *"C"* gefunden werden. Zudem muss die Applikation ein Sendeinteresse angeben, damit die Partition Daten auf dem Bus überträgt. Mit dem Einschalten der Sendeanfrage muss bereits das gültige Datum und eine DID anliegen. Mit jedem weiterem Taktzyklus kann ein neues Datum übertragen werden. Dieser Vorgang kann ebenfalls der Markierung *"C"* entnommen werden. Zeitversetzt kümmert sich der Interconnect darum, die Daten auf beide Datenbusse zu legen. Dies kann der Markierung *"B"* entnommen werden. Zusätzlich wurde eine zweite Applikation intrigiert. Diese veranschaulicht das Empfangen von Daten. Das Empfangen kann der Markierung *"D"* entnommen werden. Mit jeder steigenden Flanke wird ein neues Datum an die Applikation übergeben, welche die Daten in die Register zwischenspeichert, was der Markierung *"E"* jeweils entnommen werden kann. Dadurch kann das ordnungsgemäße Senden und Empfangen durch die Simulation nachgewiesen werden.

Moduldeaktivierung und Sendeverbot

Das nächste Ergebnis wirft einen Blick auf die Deaktivierung eines Moduls und veranschaulicht das Sendeverbot. Die Simulationsergebnisse können der Abbildung 7.15 entnommen werden. Dieses Simulationsergebnis zeigt die Deaktivierung eines Moduls. Ausgangspunkt dafür stellt

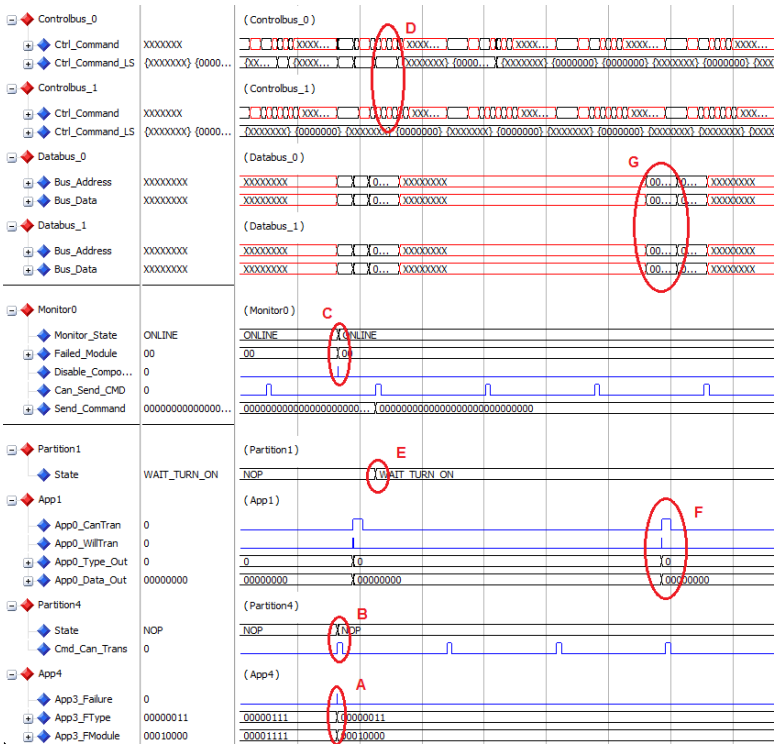

Abbildung 7.15: Simulationsergebnis des Deaktivieren eines Moduls. Detailbetrachtung der Datenübertragung.

die Applikation 4 dar. Die Applikation überträgt, der Markierung *"A"* zu entnehmen, einen Fehler an das Interconnect. Die Schnittstelle kümmert sich letztlich darum ein Kommando auf dem Bus zu versenden. Das Senden wird leicht zeitversetzt ausgeführt, was der Markierung *"B"* entnommen werden kann. Dies liegt am Zeitslot-Verfahren. Nachdem das Kommando auf dem Bus versendet wurde, reagiert der Monitor auf das Fehlersignal. In der Markierung *"C"* kann man die Reaktion sehen. Anhand des Fehlertyps entscheidet der Monitor, dass es sinnvoll ist die Applikation 1 vorläufig zu deaktivieren. Dazu versendet der Monitor ein entsprechendes Kommando über den Steuerbus. Auch dies geschieht zeitversetzt wegen dem Zeitslot-Verfahren, wie der Markierung *"D"* zu entnehmen ist. Nach dem Übertragen des Kommondos reagiert der Interconnect des Moduls auf das Kommando und sperrt daraufhin

umgehend die Kommunikation zwischen Applikation und Bus. Die Markierungen *"F"* und *"G"* dienen dafür als Nachweis. Die Applikation kann Daten übertragen, was der Markierung *"F"* entnommen werden kann. Jedoch wird dies durch den Interconnect verhindert, wie man auf den beiden Datenbussen innerhalb der Markierung *"G"* sehen kann. Dadurch kann die Abschaltvorrichtung zwischen Applikation und den Steuer- und Datenbussen geprüft und die Funktionsfähigkeit durch die Simulation nachgewiesen werden.

Modulrekonfiguration

Das letzte Simulationsergebnis beschreibt den Ablauf einer Rekonfiguration, im speziellen dem Neuladen des Moduls innerhalb seiner alten Partition. Das Ergebnis kann der Abbildung 7.16 entnommen werden. Dieses Simulationsbeispiel zeigt den Ablauf einer Rekonfiguration.

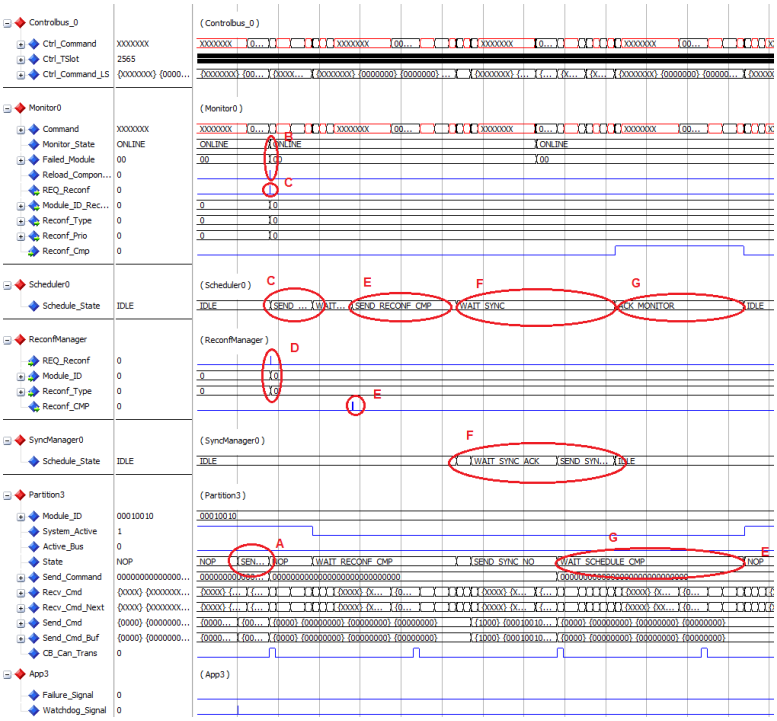

Abbildung 7.16: Simulationsergebnis der Rekonfiguration eines Moduls durch das Neuladen des Moduls. Rein schematische Betrachtung, da die Rekonfiguration als solche nicht simuliert werden kann.

Die Rekonfiguration wird durch einen Fehler eingeleitet. Dabei löst innerhalb der Partition 3 ein Watchdog aus, was daraufhin deutet, dass die Applikation nicht mehr reagiert. Daraufhin versendet der Interconnect ein Fehlerpaket auf dem Steuerbus, damit der Monitor über das

Problem in Kenntnis gesetzt wird. Das Senden des Fehlers kann der Markierung *"A"* entnommen werden. Der Monitor reagiert direkt auf den Fehler, wie in der Markierung *"B"* gezeigt wird. Anhand dieses Fehlertyps versucht der Monitor zunächst ein Neuladen der Applikation. Dazu wird eine Rekonfigurationsanfrage mit allen nötigen Informationen an den Scheduler gestellt. Der Scheduler reagiert direkt auf die Anfrage, da das FIFO bisher leer ist. Dieser Schritt geschieht innerhalb der Markierungen *"C"*. Der Scheduler beauftragt nun den Reconfiguration Manager mit dem Neuladen der Partition. Der Reconfiguration Manager führt nun die Rekonfiguration durch. In der Simulation wurde der Manager durch ein Stub ausgetauscht, da die Rekonfiguration als solche nicht simulierbar ist. Durch die Rückmeldung, der Markierung *"E"* zu entnemmen, bekommt der Scheduler Bescheid, dass die Rekonfiguration abgeschlossen wurde. Dies wird ebenfalls über den Bus versendet, wie der Markierung *"E"* entnommen werden kann. Danach wird der Synchronous Manager mit der Synchronisation beauftragt. Dies kann der Markierung *"F"* entnommen werden. Der Manager fragt die Partition an, welche Synchronisation gewünscht ist. In diesem Beispiel keine, daher kann der Manager direkt den Abschluss der Synchronisation bekanntgeben. Nachdem die Synchronisation abgeschlossen wurde, überträgt der Scheduler die erfolgreiche Synchronisation an den Monitor. Dies kann eine Weile dauern, wie innerhalb der Markierung *"G"* zu sehen ist. Nachdem der Monitor die Rückgabe quittiert hat, wird auf dem Bus publiziert, dass die Rekonfiguration und Synchronisation erfolgreich abgeschlossen wurde. Die Partition gibt die Applikation wieder frei, wie bei *"E"* gesehen werden kann. Dadurch konnte schematisch der Ablauf der Systemsteuerung dargestellt werden und auf Korrektheit geprüft werden.

7.5 Systemtest

Für den Systemtest sollte ein Nexys 4 DDR Artix-7 FPGA von Digilent[7] verwendet werden, da dieses Board der Hochschule zur Verfügung steht. Die Entwicklungsplattform verwendet den Xilinx Artix-7 FPGA XC7A100T-1CSG324C[8]. Auf der Plattform ist bereits ein 128 MB großer DDR2-RAM von Micron vorhanden. Der genaue Speichertyp lautet: Micron MT47H64M16HR-25[9]. Zudem sind bereits alle wichtigen Schnittstellen wie ein MicroSD-Card Connector und ein FTDI für die Umsetzung von UART auf USB vorhanden.

Beim Versuch das gesamte System auf dem FPGA abzubilden gab es viele Probleme. Es stellte sich dabei heraus, dass der gewählte FPGA zu wenig Ressourcen für eine vollständige Implementierung bereitstellt. Das liegt zum Einen an dem Speichercontroller von Xilinx, dem MIG. Dieser benötigt eine Vielzahl an Ressourcen. Zusätzlich werden Hardwaremakros eingesetzt,

um gezielt bestimmte dedizierte Ressourcen des FPGA zu nutzen. Diese werden benötigt um das Timing zwischen FPGA und DDR2-Speicher einhalten zu können. Diese Komponenten finden sich physikalisch verteilt auf dem gesamten FPGA, wodurch der Speichercontroller auf dem gesamten FPGA verteilt ist. Zum Anderen liegt es an den Einschränkungen die für eine partielle Partitionen gelten. Auf einem FPGA der 7-Serie können die Partitionen nicht willkürlich definiert werden. Es ist nötig das eine Partitionen an ihren Rändern die Clock-Regionen des FPGA erreicht. Dadurch kann eine Partition nicht an jeder gewünschten physikalischen Position im FPGA platziert werden. Durch den Speichercontroller und der gesamten Infrastruktur war es nicht möglich, alle Komponenten und Module auf dem FPGA physikalisch zu platzieren, ohne das es zu Ressourcenknappheit oder Laufzeitproblemen kam. Daher wurde die konkrete Implementierung zurückgestellt.

Durch die Simulation wurden bereits Integrationstests durchgeführt, die Zeigen das einer Implementierung auf einem FPGA theoretisch nichts im Wege steht. Durch diese Ergebnisse konnte gezeigt werden, dass die entwickelte und getestet Infrastruktur die Anforderungen erfüllt. Da die Simulation aber weder den Arbeitsspeicher noch die Rekonfiguration abbildet, wurden diese Module mit einer einfachen Steuerung in den FPGA integriert. Mit diesem kompakten System und zwei kleinen Partitionen war es möglich, zu prüfen, ob die beiden Komponenten funktionieren und es konnte gezeigt werden, das der gewählte Ansatz und dessen Implementierung funktioniert. Dabei wurde die Rekonfigurationen explizit über eine UART-Schnittstelle ausgelöst, sodass keine Intelligenz in der Systemsteuerung nötig war. Dazu wurden zu Beginn, die partiellen Bit-Files über eine UART-Schnittstelle via FPGA an den Arbeitsspeicher übertragen. Über einen Befehl über die UART-Schnittstelle, war es dann möglich eine Partition zu rekonfigurieren. Dazu musste die exakte Adresse im Arbeitsspeicher an die Steuerung übermittelt werden. Dieser Test lief zufriedenstellend. Dadurch konnte gezeigt werden, das auch diese Komponenten den Anforderungen entsprechen. Es mussten jedoch alle Adressen, für den Arbeitsspeicher händisch berechnet werden, was einen zusätzlichen Aufwand darstellt.

[7] Nexys 4 DDR Artix-7: http://store.digilentinc.com/nexys-4-ddr-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/
[8] Xilinx Artix-7 FPGA: https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html
[9] Micron DDR2-RAM: https://www.micron.com/parts/dram/ddr2-sdram/mt47h64m16hr-25

8 Schluss

In diesem Kapitel wird ein Fazit über die Ergebnisse der Arbeit gezogen. Im Zuge dessen erfolgt außerdem eine Zusammenfassung aller Arbeitsschritte sowie der im Rahmen dessen gewonnenen Ergebnisse. Abschließend wird im Ausblick dargelegt, welche Weiterentwicklungsmöglichkeiten die vorhandene Infrastruktur bietet und welche Systemteile für ein effizienteres Arbeiten angepasst oder entwickelt werden müssen.

8.1 Fazit

Im Rahmen dieser Arbeit wurde eine Infrastruktur entwickelt, welche eine sichere und robuste Rekonfiguration ermöglicht. Diese bietet dabei ein Minimalprinzip, wodurch diese besonders für kleine, leichte und kompakte Trägersysteme geeignet ist. Die Infrastruktur vereint dabei die Steuerung, Rekonfiguration, Datensynchronisation und Systemüberwachung und nimmt dadurch dem Applikationsentwickler viel Arbeit ab. Dadurch wird es ihm möglich, sich auf das Wesentliche zu konzentrieren: „der Applikationsentwicklung“. Die Simulation liefert sehr gute Ergebnisse und alle bisherigen Tests verliefen erfolgreich.

Der erste Systemvorschlag bot ein gutes Gerüst als Ausgangspunkt. Durch die ausführliche Problemanalyse und das Heranziehen vergleichbarer Arbeiten konnten die Vorschläge bereits in viele Erkenntnisse einfließen und im Entwurf berücksichtigt werden. Der erste Vorschlag stellt bereits einen relativ robusten Systementwurf bereit. Durch die zusätzliche FMEA-Analyse konnten weitere Schwachstellen im Systemvorschlag identifiziert und in den nächsten Schritten behoben werden. Dadurch konnte die Systemzuverlässigkeit deutlich erhöht werden.

Die Umsetzung wurde sehr ausführlich durchgeführt. Dadurch konnten viele Details des Systementwurfs noch einmal aufgegriffen und explizite Designentscheidungen begründet werden. Zudem flossen Ergebnisse der Analyse direkt in die Implementierung ein. Diese wurden ausführlich besprochen. Bestimmte Systemteile wurden zunächst nicht implementiert. Zum Einen wurde bisher keine Speicherüberwachung für den Speicher der einzelnen Komponenten

implementiert, da diese Probleme zunächst durch die Speichergröße gelöst wurden. Die SDIO-Schnittstelle wurde noch nicht umgesetzt, da durch diese, eine sehr komplexe Komponente in das System integriert werden muss, ohne dass ein erster Test ausgeführt werden konnte. Daher wurde dies zurückgestellt. Die Simulationsergebnisse decken bisher den gesamten Systemumfang im einzelnen ab. Bisher wurde kein Belastungstest durchgeführt. Auch die M-zu-N-Kommunikation sowie das parallele Rekonfigurieren wurde noch nicht getestet.

Im Großen und Ganzen ist das Ergebnis ein Schritt in die richtige Richtung. Alle bisher geprüften Bestandteile funktionieren und das System stellt eine generische Infrastruktur bereit. Diese ist leichtgewichtig und kann vielseitig eingesetzt werden. Dadurch konnten bereits eine Vielzahl der Ziele erreicht werden. Jedoch besteht an verschiedenen Stellen verbesserungsbedarf.

8.2 Ausblick

Die größte Problematik stellt im Moment die fehlende Integration in ein Testsystem dar. Um die Integration erfolgreich abschließen zu können, wäre es zum Einen möglich eine anderen FPGA zu wählen, beispielweise einen Kintex-7 oder Virtex-7. Diese FPGA haben bezüglich der Partitionsplatzierung weniger Einschränkungen und stellen eine wesentlich größere Anzahl an Ressourcen bereit. Zum Anderen wäre es möglich, die bestehende Implementierung zu reduzieren. Dadurch kann es aber dazu kommen, dass der Funktionsumfang des Systems reduziert werden muss. Wenn eine Implementierung erfolgreich war, sollte sichergestellt werden das nachvollziehbare Tests entworfen werden. Es wäre nötig ein automatisches Tool zu entwickeln, was gezielt nachvollziehbare Tests ausführt und das Ergebnis auf Basis der Debug-Schnittstelle mit dem Erwartungswert abgleicht. Des Weiteren ist es dringend notwendig Belastungstests aufzuführen, sowohl in der Simulation als auch für eine mögliche Integration auf dem FPGA.

Eine weiteres erwähntes Problem stellt die Synthese dar. Es ist momentan viel Handarbeit gefragt. Da nur ein kleiner Systemteil getestet wurde, war das berechnen der Adressen kein größeres Problem aber mit einer wachsenden Anzahl von Modulen und Partition wird dies immer schwieriger, unübersichtlicher und somit fehlerträchtiger. Daher wäre ein Vorschlag diese direkt mithilfe des TCL-Skriptes zu errechnen. Das hätte den Vorteil, dass man sein System nur an einer Stelle konfigurieren muss und das Ergebnis bereits die fertige Datei erzeugt, was vergleichbar mit einem Image ist. Zudem sollte auf lange Sicht das SD-Karten-Interface implementiert werden, da die UART-Schnittstelle bereits an den Grenzen des Machbaren betrieben

wird, um eine zügige Datenübertragung zu gewährleisten.

Im Idealfall sollte in Zukunft ein gesamtes System die Infrastruktur verwenden. Eine Infrastruktur macht letztlich nur Sinn, wenn sie auch verwendet wird. Daher sollte ein kleines System auf Basis der gebotenen Systemsteuerung implementiert werden. Zudem kann es nötig werden, dass zusätzlich externe IO-Schnittstellen benötigt werden. Dazu muss noch eine Implementation erstellt werden. Bisher gab es lediglich zwei Ideen dazu. Entweder existiert pro Partition eine begrenzte Anzahl an IO-Schnittstellen oder es existieren mehrere Busteilnehmer welche verschieden definierte IO-Bereich auf dem Bus publizieren. Letzteres würde aber auch die Schnittstellenvielfalt deutlich einschränken.

Literaturverzeichnis

[1] Al-Haddad, R. ; Oreifej, R. ; Ashraf, R. A. ; DeMara, R. F.: Sustainable Modular Adaptive Redundancy Technique Emphasizing Partial Reconfiguration for Reduced Power Consumption. In: *International Journal of Reconfigurable Computing vol. 2011*, 2011. – https://www.doi.org/10.1155/2011/430808

[2] Bozzano, Marco ; Villafiorita, Adolfo: *Design and Safety Assessment of Critical Systems*. Auerbach Publications, 2011. – ISBN 978–1–4398–0331–8

[3] Dumitriu, Victor ; Kirischian, Lev ; Kirischian, Valeri: Run-Time Recovery Mechanism for Transient and Permanent Hardware Faults Based on Distributed, Self-organized Dynamic Partially Reconfigurable Systems, 2015. – https://www.doi.org/10.1109/TC.2015.2506558

[4] Eapen, Madhuri E. ; Pradeep, C. ; Varghese, Anila A. ; Nair, Jisha M.: Built-in Self Repair Approach by Module Relocation for FPGA Based Reconfigurable Systems. In: *Procedia Technology* 24 (2016), S. 1587 – 1594. – ISSN 2212–0173. – International Conference on Emerging Trends in Engineering, Science and Technology (ICETEST - 2015). https://dx.doi.org/10.1016/j.protcy.2016.05.146

[5] JEDEC Solid State Technology Association (Hrsg.): *JEDEC Standard - DDR2-SDRAM Specification*. 79-2F. JEDEC Solid State Technology Association, November 2009

[6] Lüdtke, D. ; Westerdorff, K. ; Stohlmann, K. ; Börner, A. ; Maibaum, O. ; Peng, T. ; Weps, B. ; Fey, G. ; Gerndt, A.: OBC-NG: Towards a reconfigurable on-board computing architecture for spacecraft. In: *2014 IEEE Aerospace Conference*, 2014. – ISSN 1095–323X, S. 1–13

[7] Micron Technology, Inc. (Hrsg.): *MICRON - DDR2 SDRAM - 1Gb: x4, x8, x16 DDR2 SDRAM*. Rev. AA. Micron Technology, Inc., Juli 2014

[8] Paulsson, K. ; Hubner, M. ; Jung, M. ; Becker, J.: Methods for run-time failure recognition and recovery in dynamic and partial reconfigurable systems based on Xilinx Virtex-II

Pro FPGAs. In: *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, 2006. – ISSN 2159–3469, S. 6 pp.–. – https://www.doi.org/10.1109/ISVLSI.2006.62

[9] Pilotto, Conrado ; Azambuja, José R. ; Kastensmidt, Fernanda L.: Synchronizing Triple Modular Redundant Designs in Dynamic Partial Reconfiguration Applications. In: *Proceedings of the 21st Annual Symposium on Integrated Circuits and System Design*. New York, NY, USA : ACM, 2008 (SBCCI '08). – ISBN 978–1–60558–231–3, S. 199–204. – https://dx.doi.org/10.1145/1404371.1404426

[10] Russek, Paweł: *FPGA Design Tips*

[11] Sabena, D. ; Sterpone, L. ; Schölzel, M. ; Koal, T. ; Vierhaus, H. T. ; Wong, S. ; Glein, R. ; Rittner, F. ; Stender, C. ; Porrmann, M. ; Hagemeyer, J.: Reconfigurable high performance architectures: How much are they ready for safety-critical applications? In: *2014 19th IEEE European Test Symposium (ETS)*, 2014. – ISSN 1530–1877, S. 1–8. – https://www.doi.org/10.1109/ETS.2014.6847820

[12] Stock, Michael: *CANaerospace - Interface specification for airborne CAN applications*. 1.7. Stock Flight Systems, Januar 2006

[13] Storey, Neil: *Safety-Critical Computer Systems*. Pearson Education Limited, 1996. – ISBN 0–201–42787–7

[14] Straka, M. ; Kastil, J. ; Kotasek, Z.: Modern fault tolerant architectures based on partial dynamic reconfiguration in FPGAs. In: *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010, S. 173–176. – https://www.doi.org/10.1109/DDECS.2010.5491793

[15] Szurman, K. ; Miculka, L. ; Kotasek, Z.: State Synchronization after Partial Reconfiguration of Fault Tolerant CAN Bus Control System. In: *2014 17th Euromicro Conference on Digital System Design*, 2014, S. 704–707. – https://dx.doi.org/10.1109/DSD.2014.103

[16] Vít, P. ; Borecký, J. ; Kohlík, M. ; Kubátová, H.: Fault Tolerant Duplex System with High Availability for Practical Applications. In: *2014 17th Euromicro Conference on Digital System Design*, 2014, S. 320–325. – https://www.doi.org/10.1109/DSD.2014.54

[17] Xilinx Inc. (Hrsg.): *Partial Reconfiguration User Guide*. v14.1. Xilinx Inc., April 2012

[18] Xilinx Inc. (Hrsg.): *Vivado Design Suite Tcl Command Reference Guide*. v2013.4. Xilinx Inc., Dezember 2013

[19] Xilinx Inc. (Hrsg.): *Xilinx 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide for HDL Designs.* v14.7. Xilinx Inc., Oktober 2013

[20] Xilinx Inc. (Hrsg.): *Vivado Design Suite User Guide - Design Flows Overview.* v2014.1. Xilinx Inc., April 2014

[21] Xilinx Inc. (Hrsg.): *Vivado Design Suite User Guide - Partial Reconfiguration.* v2014.4. Xilinx Inc., November 2014

[22] Xilinx Inc. (Hrsg.): *Vivado Design Suite User Guide - Programming and Debugging.* v2014.1. Xilinx Inc., Mai 2014

[23] Xilinx Inc. (Hrsg.): *7 Series FPGAs Configuration - User Guide.* v1.10. Xilinx Inc., Juni 2015

[24] Xilinx Inc. (Hrsg.): *Vivado Design Suite Tutorial - Partial Reconfiguration.* v2016.1. Xilinx Inc., April 2016

[25] Xilinx Inc. (Hrsg.): *Zynq-7000 AP SoC and 7 Series Devices Memory Interface Solutions v4.1 - User Guide.* v4.1. Xilinx Inc., November 2016

[26] Xilinx Inc. (Hrsg.): *7 Series FPGAs Clocking Resourcess - User Guide.* v1.13. Xilinx Inc., März 2017

[27] Zhang, K. ; Bedette, G. ; DeMara, R. F.: Triple Modular Redundancy with Standby (TMRSB) Supporting Dynamic Resource Reconfiguration. In: *2006 IEEE Autotestcon*, 2006. – ISSN 1088–7725, S. 690–696. – https://www.doi.org/10.1109/AUTEST.2006.283750

[28] *MP21283X Triple Redundant UAV Autopilot.* https://www.micropilot.com/pdf/brochures/brochure-MP21283x.pdf, . – [Online; besucht 22. November 2016]

[29] *AR44942 - Virtex-7, Kintex-7, Artix-7 FPGA Configuration - BUSY Pin Removal.* https://www.xilinx.com/support/answers/44942.html, November 2011. – [Online; besucht 22. November 2016]

[30] *Design Methodologies and Advanced Tools - How to use ICAP primitive in our program???* https://www.xilinx.com/support/answers/44942.html, Februar 2011. – [Online; besucht 22. November 2016]

[31] *AR53347 - 14.3 and 14.4 Config Simulation - Functional simulation passed but post-PAR...* https://www.xilinx.com/support/answers/53347.html, Dezember 2012. – [Online; besucht 22. November 2016]

[32] *7 Series FPGAs - ICAPE2 Configuration Register Read Procedure.* https://forums.xilinx.com/t5/7-Series-FPGAs/ICAPE2-Configuration-Register-Read-Procedure/td-p/356861, Dezember 2013. – [Online; besucht 22. November 2016]

[33] *Configuration - ICAPE2 documentation.* https://forums.xilinx.com/xlnx/board/crawl_message?board.id=CNFG&message.id=616, Juni 2014. – [Online; besucht 22. November 2016]

[34] *Configuration - ICAPE2 partial reconfiguration.* https://forums.xilinx.com/xlnx/board/crawl_message?board.id=CNFG&message.id=1741, Maerz 2015. – [Online; besucht 22. November 2016]

[35] *MP21283X Triple Redundant.* https://www.micropilot.com/products-mp21283x.htm, Juni 2017. – [Online; besucht 22. Juni 2017]

[36] *PaparazziUAV.* https://wiki.paparazziuav.org/wiki/Main_Page, Juni 2017. – [Online; besucht 22. Juni 2017]

[37] *Pixhawk Autopilot.* https://pixhawk.org/modules/pixhawk, Juni 2017. – [Online; besucht 22. Juni 2017]

[38] *U-Pilot - Airelectronics.* http://www.airelectronics.es/products/air_segment/air_hw/, Juni 2017. – [Online; besucht 22. Juni 2017]

Inhalt der CD-ROM

Diesem Abschnitt kann eine kurze Übersicht des Inhaltes der DVD entnommen werden:

```
DVD-Root
├── latex
├── library
│   ├── _generation
│   ├── _simulation
│   ├── generic
│   ├── peripheral
│   └── sra
└── Masterarbeit_Buescher_Rene.pdf
```

Der Hauptordner enthält als einzige Datei die Masterarbeit im PDF-Format. Alle anderen Daten können den anderen Ordnern entnommen werden. Zusätzlich zum PDF-Format sind ebenfalls alle LaTeX und BibTeX Quelldaten vorhanden, die dem Ordner *"latex"* entnommen werden können. In diesem Ordner sind zudem alle Bilddaten abgelegt. Die Abbildungen sind jeweils im PNG-Format vorhanden, sowie im Rohformat. Der Ordner *"library"* enthält zum einem die VHDL-Daten und alle weiteren Projektdaten für Vivado oder ModelSim. Innerhalb des Unterordners *"_generation"* können das statische und dynamische Projekt zum Erstellen der Bit-Dateien gefunden werden. Zusätzlich liegen alle TCL-Skripte vor. Im Unterordner *"_simulation"* können die Simulationsdateien und Skripte für ModelSim gefunden werden. Der Unterordner *"generic"* beinhaltet generische VHDL-Bibliotheken. Der Unterordner *"peripheral"* beinhaltet VHDL-Bibliotheken, welche standardisierte Schnittstellen enthält. Der Unterordner *"sra"* beinhaltet alle projektspezifischen VHDL-Bibliotheken.

Verwendete Programme

In diesem Abschnitt werden alle verwendeten Programme in der folgenden Tabelle erfasst. Für jedes Programm ist der genaue Name, die Version sowie eine kurze Beschreibung angegeben, wozu das Programm verwendet wurde.

Programm	**Version**	**Beschreibung**
MikTeX	2.9.6300	Enthält und verwaltet die TeXPakete. Beinhaltet verschiedene benötigte Interpreter.
TeXstudio	2.10.6	Editor für TeXDateien.
Microsoft Visio Professional 2010	14.0	Erstellung von verschiedenen Diagrammen.
Binary Viewer	5.15.5.9	Binäre Volumen- und Dateibetrachtung.
Vivado Design Suite	2016.1	Hardwaresynthese für Xilinx Produkte, FPGAs und CPLDs.
Mentor Graphics ModelSim SE-64	10.5c	Hardwaresimulation.
Notepad++	6.8	Einfacher Editor für die Betrachtung verschiedener Quellcodedateien.
HTerm	-	Textuelle Anzeige von COM-Port Übertragungen.
WaveDrom Editor	1.4.2	Erstellung von grafischen Signalverläufen.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 31. August 2017 René Büscher