Bundesamt
für die Sicherheit
der nuklearen Entsorgung

RESSORTFORSCHUNGSBERICHTE ZUR
SICHERHEIT DER NUKLEAREN ENTSORGUNG

Analyse der Fehlermodi von programmierbaren logischen Schaltungen in der Sicherheitsleittechnik von Kernkraftwerken

Vorhaben FKZ 7420R01310

AUFTRAGNEHMER:INNEN:
TÜV Rheinland Industrie Service GmbH, Hallbergmoos

R. Heigl
H. Miedl
G. Savchyn
G. Schnürer
D. Wölbing

Analyse der Fehlermodi von programmierbaren logischen Schaltungen in der Sicherheitsleittechnik von Kernkraftwerken

BASE-RESFOR-028/24

Bitte beziehen Sie sich beim Zitieren dieses Dokumentes immer auf folgende URN:
urn:nbn:de:0221-2024103048058

Berlin, November 2023

Impressum

Bundesamt
für die Sicherheit
der nuklearen Entsorgung
(BASE)

RESSORTFORSCHUNGSBERICHTE ZUR
SICHERHEIT DER NUKLEAREN ENTSORGUNG

Auftragnehmer:innen:
TÜV Rheinland Industrie Service GmbH, Hallbergmoos

R. Heigl
H. Miedl
G. Savchyn
G. Schnürer
D. Wölbing

030 184321–0
www.base.bund.de

Stand: November 2023

GZ: F 4 – BASE – BASE62110/4720R01310#0021

TÜVRheinland®
Genau. Richtig.

Analyse der Fehlermodi von programmierbaren logischen Schaltungen in der Sicherheitsleittechnik von Kernkraftwerken

Abschlussbericht

R. Heigl

H. Miedl

G. Savchyn

G. Schnürer

D. Wölbing

ISTec - A - 4013

Rev. 5.0

November 2023

Das diesem Bericht zugrundeliegende F&E-Vorhaben wurde im Auftrag des Bundesministeriums für Umwelt, Naturschutz und nukleare Sicherheit unter dem Kennzeichen 7420R01310 durchgeführt. Der Bericht gibt die Auffassung und Meinung des Auftragnehmers wieder und muss nicht mit der Meinung der Auftraggeberin übereinstimmen.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

VERZEICHNIS DER TABELLEN

REVISIONSBLATT

Datum	Rev.	Änderungen	Bearbeiter
23.02.2023	1	Entwurf	Heigl Savchyn Woelbing
04.05.2023	1.1	Aktualisierung: Komplexitätsbasiertes Testen	Heigl Savchyn
25.07.2023	2	Einarbeitung aller Kommentare von JDE. Kapitel zu Komplexitätsanalyse eingefügt. Kapitel zu Qualifizierungskriterien eingefügt. Zusammenfassung aktualisiert. Publikation und Präsentationsfolien eingefügt.	Heigl
24.10.2023	3	Überarbeitung nach internem Review	Miedl Heigl
17.11.2023	4	Überarbeitung gemäß Kommentaren der Auftraggeberin	Miedl Heigl Woelbing
28.11.2023	5	Redaktionelle Korrekturen	Heigl

KURZFASSUNG

Der Einsatz digitaler Leittechnik in Kernkraftwerken (KKW) führte zu einer zunehmenden Diskussion darüber, wie Methoden für eine Bewertung der Zuverlässigkeit solcher Systeme geschaffen werden können. Die Probleme ergeben sich zum einen aus dem Mangel an allgemein akzeptierten Modellen zur quantitativen Bewertung von Verfügbarkeit und Zuverlässigkeit, und zum anderen aus der Problematik der Integration solcher Modelle in die Bewertung der Zuverlässigkeit eines Gesamtsystems. Dennoch werden immer komplexere Systeme implementiert, um den wachsenden Anforderungen an die Funktionalität gerecht zu werden. Daher wurde die Frage nach einer Methode zur Bewertung der Zuverlässigkeit von Systemen, die auf Software oder programmierbarer Logik basieren, immer dringlicher.

Für die sicherheitsrelevante Leittechnik von KKW sind zunehmend Geräte auf der Basis programmierbarer logischer Schaltungen wie Field Programmable Gate Array (FPGA) auf dem Markt verfügbar. Dementsprechend besteht die Notwendigkeit, Methoden zur Bewertung ihrer Sicherheit und Zuverlässigkeit zu ermitteln oder zu entwickeln. Es ist zu erwarten, dass programmierbare logische Schaltungen zunehmend während der Restlebensdauer von KKWs und darüber hinaus in Nach- und Umrüstungsmaßnahmen sowie zur Deckung des Ersatzbedarfs (Redesign von Komponenten) eingesetzt werden.

In vorangegangenen Forschungsvorhaben (FKZ-3614R01310 [9]) wurde ein Ansatz zur Bestimmung der Komplexität auf Basis eines Komplexitätsvektors entwickelt, und die zugrundeliegende Methode auf programmierbare logische Schaltungen erweitert. Der Komplexitätsvektor definiert eine Metrik, welche die spezifischen Eigenschaften und Merkmale der programmierbaren Logik digitaler Leittechniksysteme widerspiegelt. Das Komplexitätsmessverfahren und seine praktische Anwendbarkeit wurden durch Untersuchung und Anwendung auf mehrere beispielhafte Geräte demonstriert, wie sie in kerntechnischen Leittechnikanwendungen eingesetzt werden.

In dem vorliegenden Bericht werden Fehlermodi von FPGA-basierten Leittechnik-Komponenten identifiziert. Ferner werden Fehlerursachen und Auswirkungen sowie verschiedene Testverfahren zur Sicherheits- und Zuverlässigkeitsbewertung analysiert. Ziel ist es, zu untersuchen, welche Fehlermodi die Testverfahren zu erkennen vermögen. Die Tests werden an eigens im Rahmen des Forschungsvorhabens entwickelten FPGA-Designs durchgeführt, die repräsentativ für sicherheitsrelevante Anwendungen in der Leittechnik von Kernkraftwerken sind. Die Komplexität der entwickelten FPGA-Designs wurde bei der Generierung der Testfälle besonders berücksichtigt. Der Grundgedanke war, komplexe Abschnitte des Designs intensiv zu testen, um eine optimierte Testabdeckung zu erreichen. Die Testverfahren wurden durch Anwendung von Fehlerinjektionen auf mögliche Schwachstellen untersucht und die erreichte Testabdeckung bewertet.

ABSTRACT

Utilization of digital I&C for application in nuclear power plants (NPPs) led to a growing discussion on how methods can be developed to assess the reliability of such systems. The issues involved arise, on the one hand, from a lack of commonly accepted models for quantitative evaluation of availability and reliability, and on the other hand from the problem of integrating such models into the assessment of the reliability of an overall system. Nevertheless, increasingly complex systems are being implemented to satisfy growing requirements on functionality. Therefore, the need for a method to evaluate the reliability of systems based on software or on programmable logic became even more urgent.

For safety-related I&C of NPPs, devices based on programmable logic circuits such as Field Programmable Gate Array (FPGA) are increasingly available on the market. Accordingly, there is a need to identify or develop methods for assessing their safety and reliability. It is to be expected that programmable logic circuits will increasingly be used during the remaining service life of NPPs and beyond in retrofitting and refurbishment measures as well as to cover replacement requirements (redesign components).

In previous research projects (FKZ-3614R01310 [9]) an approach for determining complexity based on a complexity vector was developed, and the underlying method was extended to programmable logic devices. The complexity vector defines a metric that reflects the specific properties and features of programmable logic used in digital I&C systems. The complexity measuring procedure and its practical applicability has been demonstrated by investigation and application to several exemplary devices which are actually used in nuclear I&C applications.

Failure modes of FPGA based I&C components are identified in this report. Moreover, their causes and effects as well as various test procedures for safety and reliability assessment are analyzed. The aim is to investigate which failure modes the test procedures are able to detect. The tests are performed on FPGA designs developed specifically as part of the research, which are representative of safety-relevant I&C applications in NPP. The complexity of the developed FPGA designs was particularly taken into account when the test cases were generated. The basic idea was to test complex sections of the design more intensively in order to achieve an optimized test coverage. Test procedures were examined for potential weaknesses using fault injection and the achieved test coverage was assessed.

1 EINLEITUNG

In den folgenden Unterkapiteln wird auf die Beauftragung, Zielsetzung und relevante Technologien des Vorhabens eingegangen.

1.1 Beauftragung

Mit Schreiben vom 08.09.2020 beauftragte das Bundesamt für die Sicherheit der nuklearen Entsorgung (BASE) im Auftrag des Bundesministeriums für Umwelt, Naturschutz, nukleare Sicherheit und Verbraucherschutz (BMUV) die TÜV Rheinland Industrie Service GmbH mit der Analyse von Fehlermodi von programmierbaren logischen Schaltungen in der Sicherheitsleittechnik von Kernkraftwerken. Diese Arbeiten sollen auch zur Aktualisierung und Fortschreibung des deutschen und internationalen Normenwerks und des kerntechnischen Regelwerks beitragen. Die im Vorhaben angewandte Methodik zur Komplexitätsmessung von programmierbarer Logik setzt auf den Ergebnissen von Vorläuferprojekten [5] und [9] auf.

1.2 Zielsetzung

In der Sicherheitsleittechnik von Kernkraftwerken kommen zunehmend Geräte auf den Markt, die auf programmierbaren logischen Schaltungen wie FPGA basieren. Entsprechend besteht der Bedarf, Methoden zur Bewertung ihrer Sicherheit und Zuverlässigkeit zu identifizieren bzw. zu entwickeln. Auch in Deutschland ist damit zu rechnen, dass während der Restlaufzeit von KKWs und darüber hinaus bei Nach- und Umrüstmaßnahmen sowie zur Deckung des Ersatzbedarfs (Redesign-Komponenten) zunehmend entsprechende Geräte eingesetzt werden.

Da die Einsatzzeit von FPGA-basierte Leittechnik im Vergleich zu CPU-basierter Leittechnik in KKWs kürzer ist, gibt es eine geringere Datenbasis, um belastbare Aussagen zur Zuverlässigkeit oder dem Fehlerpotential von FPGA-Anwendungen zu treffen. FPGAs stellen grundsätzlich eine neuere Technologie dar, die zudem einen hohen Innovationsgrad aufweist, weshalb weitere Forschung einen wertvollen Beitrag zur Entwicklung und Erprobung von Bewertungsmethoden leisten kann. In diesem Vorhaben werden Fehlermodi FPGA-basierter Leittechnikkomponenten ermittelt, ihre Ursachen und Auswirkungen analysiert, und verschiedene Testverfahren zur Sicherheits- und Zuverlässigkeitsbewertung erprobt. Dabei wird untersucht, welche Fehlermodi die Testverfahren aufzudecken vermögen. Die Testverfahren werden an eigens im Rahmen des Vorhabens entwickelten repräsentativen FPGA-Designs erprobt.

Im Rahmen des vom BMU geförderten Projektes „Komplexität und Fehlerpotential bei softwarebasierter digitaler Sicherheitsleittechnik“ [9] wurde eine Methodik zur Ermittlung von Komplexitätsmetriken für softwarebasierte digitale Sicherheitsleittechnik entwickelt und für die Anwendung auf programmierbare logische Schaltungen erweitert. Ziel dieses Forschungsprojekts ist, auf der Grundlage von FPGA-Designs, die eigens mit einem entsprechenden Designwerkzeug erstellt werden, eingehende Analysen durchzuführen. Dabei wird die in früheren Forschungsvorhaben ausgearbeitete Methodik für die Ermittlung der Komplexität auf die verschiedenen FPGA-Designs angewendet. Bei der Generierung der Testfälle wird der Komplexitätsaspekt besonders berücksichtigt. Ziel dabei ist, komplexere Bereiche der FPGA-Designs verstärkt zu testeten, um so eine möglichst optimale Testabdeckung zu erreichen. Die Untersuchung eigens entwickelter Designs ermöglicht es

außerdem, die angewendeten Testverfahren durch Fehlerinjektion auf Schwachstellen hin zu untersuchen und die erreichte Testabdeckung zu bewerten.

Anhand der gewonnenen Erkenntnisse über die Fehlermodi FPGA-basierter Leittechnik und die Leistungsfähigkeit verschiedener Testverfahren werden Kriterien für die Qualifizierung von FPGA-Designs abgeleitet. Ebenso werden bestehende Qualifizierungsanforderungen in verschiedenen Regelwerken vor dem Hintergrund der Ergebnisse kritisch hinterfragt.

1.3 Technologien

Beginnend in den 1970er und 80er Jahren fanden Entwicklungen der Mikroelektronik statt, die für die Leittechnik von Kernkraftwerken von Bedeutung sind. Es wurde eine zunehmende Integration von Funktionalität bei integrierten Schaltkreisen (wie z.B. Mikroprozessoren und programmierbaren logischen Schaltungen) möglich. Die Entwicklung verlief dabei von zentralen Rechenanlagen zur Steuerung und Regelung von Prozessen zu dezentralen Industrie-Computern und zur Anwendung von Mikrocontrollern bzw. Speicherprogrammierbaren Steuerungen. Diese Entwicklung ermöglichte einen zunehmenden Einsatz softwarebasierter Systeme für leittechnische Funktionen.

Außerdem führten die Verbesserungen der Herstellungstechnologien dazu, dass integrierte Schaltkreise, anwendungs- bzw. kundenspezifische Schaltkreise (ASIC) und programmierbare Schaltkreise (PLD) kostengünstig zu produzieren waren. Über FPLAs ab Mitte der 1980er Jahre führte die Entwicklung zu den hochintegrierten CPLDs und FPGAs der heutigen Zeit.

Tabelle 1: Technologien programmierbarer logischer Schaltungen

Bezeichnung	Beschreibung
PLD	Programmable Logic Device (z.B. PLA, PROMs) programmierbare integrierte Schaltkreise FPGAs und CPLDs sind spezielle PLDs ca. ab 1980 verfügbar
PLA, FPLA	Programmable Logic Array, Field Programmable Logic Array integrierte Schaltkreise mit programmierbaren Logik-Feldern, die logische Grundverknüpfungen wie UND/ODER- Gatter, Addierer, Zähler, Flipflops usw. ermöglichen geringer bis mittlerer Integrationsgrad ca. ab 1980 verfügbar

Bezeichnung	Beschreibung
PAL	Programmable Array Logic, sind spezielle PLAs integrierte Schaltkreise mit programmierbaren Logik-Feldern (nur UND-Verknüpfungen) geringer Integrationsgrad ca. ab 1980 verfügbar
CPLD / SPLD	Complex Programmable Logic Device bestehen aus zusammengeschalteten Simple Programmable Logic Devices CPLDs sind integrierte Schaltkreise mit: programmierbarer UND/ODER-Matrix und logischen Funktionen programmierbaren Verknüpfungslogiken und Rückkopplungen mit bestimmbarem Zeitverhalten programmierbaren Eingabe- und Ausgabeblöcken mit hohem bis sehr hohem Integrationsgrad (VLSI-Technologie, bis 10.000 Gatter) ca. ab Anfang der 1990er Jahre verfügbar
FPGA	Field Programmable Gate Array integrierte Schaltkreise mit: programmierbaren Logik-, Speicher-, Ein- und Ausgabe- und Routingfeldern verschieden programmierbarer Verknüpfungslogik mit Einfluss auf das Zeitverhalten der Logikpfade sehr hohem Integrationsgrad (VLSI-Technologie, bis 100.000 Gatter-Äquivalente) ca. ab erster Hälfte der 1990er Jahre verfügbar

Der Entwicklungsgang der unterschiedlichen Technologien, bis hin zum System on a Chip (SOC), ist in Abbildung 1 dargestellt.

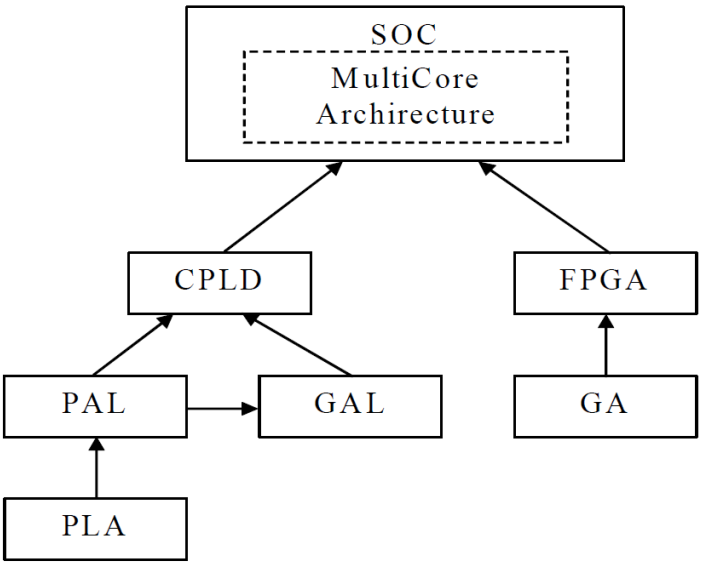


Abbildung 1: Architekturtypen von programmierbarer Logik [1]

In [1] werden FPGAs und CPLDs wie folgt gegenübergestellt: CPLDs haben ihren Ursprung in PALs, die wiederum aus den PLAs hervorgegangen sind, sowie der Generic Array Logic (GAL). Die funktionelle Einheit der CPLD besteht aus Mikrozellen, von denen jede eine kombinatorische UND/ODER-Registerfunktion ausführt. Die funktionelle Logik in einem Block ist eine Matrix logischer Terme. Über ein Term-Verteilungs-Diagramm kann jede Mikrozelle auf eine Teilmenge dieser Terme zugreifen. Umschaltmatrizen ordnen die Signale der Ausgänge der funktionalen Einheit und der Eingabe/Ausgabe-Einheit zu. Im Gegensatz zu FPGAs, bei denen die Verbindungen segmentiert sind (parallele Abarbeitung), weisen CPLDs ein kontinuierliches System von Verbindungen (serielle Abarbeitung) auf. Die FPGA-Architektur hat ihren Ursprung in verbundenen Gate Arrays. Die interne Architektur besteht aus einem Satz konfigurierbarer logischer Einheiten, die regulär angeordnet und mit Leitkanälen verbunden sind. Basisblöcke wie z.B. logische Zellen und Look-Up Tables werden als konfigurierbare Logikblöcke verwendet. Diese Module weisen wiederum eine gegliederte Architektur von internen Verbindungen auf.

2 BEGRIFFE UND ABKÜRZUNGEN

Im Folgenden sind die Begriffe und Abkürzungen definiert, die in diesem Bericht verwendet werden.

2.1 Begriffe

Im Folgenden sind Begriffe definiert, die in diesem Bericht verwendet werden.

2.1.1 Basisblock

Wesentliche Grundstruktur eines FPGA, bestehend aus einer vielfach wiederholten konfigurierbaren logischen Schaltung (z.B. Look-up Tables und Register). Die Struktur der Basisblöcke ist herstellerspezifisch.

Anmerkung: Die leittechnischen Funktionen einer FPGA-Anwendung ergeben sich aus der Verschaltung der Basisblöcke, die aus einer Hierarche von Bibliotheken in der Netzliste des FPGAs hervorgehen.

2.1.2 BIST

Ein „Built-In-Self-Test“ (BIST) beschreibt die eingebaute Eigendiagnose eines programmierbaren logischen Schaltkreises. Dabei verfügen ausgewählte elektronische Bausteine über eine integrierte Testschaltung, welche Testsignale erzeugt, die in der Regel mit vorgegebenen richtigen Antwort-Signalen verglichen werden.

2.1.3 CPLD und FPGA

Programmierbare logische Schaltungen, die in der Sicherheitsleittechnik von Kernkraftwerken zum Einsatz kommen, beruhen maßgeblich auf Technologien wie programmierbaren logischen Bausteinen (Programmable Logic Device, PLD), feldprogrammierbare logische Gatter (Field Programmable Gate Array, FPGA) und komplexen programmierbaren logischen Bausteinen (Complex Programmable Logic Device, CPLD).

Während bei herkömmlichen anwendungsspezifischen integrierten Schaltkreisen (ASIC) die Funktionen in der Regel bereits festgelegt sind, können die internen Ressourcen (z.B. programmierbare Basisblöcke) des FPGA vom Anwender im Rahmen des Entwicklungsprozesses so konfiguriert werden, dass die gewünschte Systemfunktion ausgeführt werden kann. Die programmierbaren Basisblöcke ermöglichen logische Grundverknüpfungen wie UND/ODER-Gatter, Addierer, Zähler, Flipflops usw. Die hierbei verwendeten Basisblöcke weisen eine gegliederte Architektur von internen Verbindungen auf.

2.1.4 Einschränkungen (engl.: constraints)

Vorgaben, die bei der Entwicklung eingehalten werden müssen, damit der Entwurf erforderliche Eigenschaften aufweist.

2.1.5 Fehlermodi

Art des Auftretens eines Ausfalls.

(online IEC vocabulary: “manner in which failure occurs”).

2.1.6 Funktionsbaustein

Repräsentation eines Elements mit begrenzter Funktionalität in einer graphisch orientierten Programmiersprache.

Anmerkung: Mit Hilfe eines Funktionsbausteins können neben Elementarfunktionen durch deren Verschaltung nahezu beliebige Funktionen implementiert werden, womit sich leittechnische Funktionen realisieren lassen. Aus der Verschaltung von Funktionsbausteinen, dem Funktionsplan, ergeben sich die leittechnischen Funktionen einer speicherprogrammierbaren Steuerung.

2.1.7 FPGA-Anwendung

Konfiguration eines FPGA zur Umsetzung einer gewünschten Funktion. Die Entwicklung des FPGA-Programms basiert auf dem FPGA-Design.

2.1.8 FPGA-Design

Softwarebasierter Entwurf einer FPGA-Anwendung.

Anmerkung: In diesem Vorhaben wird der Entwurf basierend auf einer Spezifikation in C/C++ Code entwickelt und mittels Synthese in HDL erzeugt. Nach weiteren Simulationen, Optimierungen und Synthesen wird eine Electronic Design Interchange Format (EDIF) Netzliste des Entwurfs generiert. Der Begriff FPGA-Design beschreibt den softwarebasierten Entwurf dabei unabhängig von dem jeweiligen Entwicklungsschritt bzw. der verwendeten Sprache.

2.1.9 Hardware-Beschreibungssprache (HDL)

Sprache, die dazu verwendet wird, die Funktionen und/oder die Struktur einer elektronischen Komponente für Dokumentation, Simulation oder Synthese formal zu beschreiben [4].

Anmerkung: Die verbreitetsten HDLs sind VHDL (IEEE 1076) und Verilog (IEEE 1364).

2.1.10 HDL-programmierter Baustein (HPD)

Integrierter Schaltkreis, der (für leittechnische Systeme in Kernkraftwerken) unter Verwendung von Hardware-Beschreibungssprachen und zugehörigen Softwarewerkzeugen konfiguriert wird [4].

Anmerkung 1: HDLs und zugehörige Werkzeuge (z. B. Simulatoren, Synthesewerkzeuge) werden verwendet, um die Anforderungen in eine geeignete Konfiguration von vorentwickelten mikroelektronischen Ressourcen umzusetzen.

Anmerkung 2: Bei der Entwicklung von HPDs können vorentwickelte Blöcke verwendet werden.

Anmerkung 3: HPDs basieren typischerweise auf FPGAs, PLDs oder ähnlichen mikroelektronischen Technologien.

2.1.11 Komplexität

Die Definition eines allgemeinen Begriffs „Komplexität“ ist ein außerordentlich schwieriges Unterfangen. Deshalb wird dieser Begriff in unterschiedlichen Gebieten unterschiedlich definiert. Zusätzlich ist er vom Wissen, das ein Betrachter in einem speziellen Gebiet besitzt, bestimmt. Deshalb wird in diesem Bericht der Begriff „Komplexität“ als Kombination messbarer Größen angesehen. Sowohl die einzelnen Messgrößen als auch deren Kombination werden als Komplexitätsmaß (bzw. synonym als Komplexitätsmetrik) bezeichnet.

2.1.12 Komplexitätsbewertung

Der Prozess der Beurteilung einer Komplexitätsmessung.

2.1.13 Komplexitätsmetrik, Komplexitätsmaß

Eine Komplexitätsmetrik ist eine Funktion, die eine Software-Einheit in Bezug auf die Qualitätseigenschaft „Komplexität“ auf einen Zahlenwert abbildet.

Anmerkung 1: Wichtige Komplexitätsmetriken sind das McCabe-Maß (zyklomatische Komplexität), die Knotenmetrik und das Verknüpfungsmaß [5].

Anmerkung 2: Die zyklomatische Komplexität und die Knotenmetrik sind Kontrollflussmaße. Das Verknüpfungsmaß nach [5] ist ein Datenflussmaß.

2.1.14 Komplexitätswert, Komplexitätsvektor und Komplexitätsmessung

Der Komplexitätswert stellt eine messbare oder berechenbare Maßzahl dar.

Die Zusammenfassung von Komplexitätswerten erfolgt z.B. in einem Komplexitätsvektor.

Unter Komplexitätsmessung wird der Prozess der Bestimmung der Werte des Komplexitätsvektors für ein bestimmtes Objekt einer Objektklasse zusammengefasst.

2.1.15 Programmierbarer logische Baustein (engl.: programmable logic device PLD)

Integrierter Schaltkreis, der aus logischen Elementen mit einer Verbindungsstruktur besteht, von denen ein Teil vom Anwender programmierbar ist [4].

Anmerkung: Programmierbarer logischer Baustein wird in diesem Bericht als programmierbare logische Schaltung bezeichnet.

2.1.16 Zustandsmaschine (engl.: finite state machine)

Eine Zustandsmaschine ist gemäß [21] das abstrakte Modell eines Automaten mit einfach strukturiertem Speichervermögen. Das Verhalten dieser Zustandsmaschine ist dadurch gekennzeichnet, dass für eine begrenzte Anzahl von Eingangssignalkombinationen (Eingangszuständen) ausgangsseitig jeweils definierte Ausgangszustände angenommen werden. Das Verhalten einer Zustandsmaschine kann in einem Diagramm dargestellt werden. Hierbei wird für jeweils einen Eingangsstatus ein Ausgangsstatus erzeugt.

2.1.17 Testbank (Simulation)

Eine Simulation, die erstellt wird, um die Funktionalität des FPGAs zu überprüfen. Bei der Erstellung der Testbank wird ein funktionaler Zusammenhang zum prüfenden FPGA hergestellt.

Anmerkung: Die Testbank wird in VHDL oder Verilog geschrieben, ist jedoch selbst nicht zwingend synthesefähig.

2.1.18 Wellenfunktion

Die Wellenfunktion stellt den zeitlichen Verlauf der angesteuerten Funktion für den Bereich der Ein- und Ausgangssignale eines FPGA dar. Wellenfunktionen werden auf Grundlage diskreter Signale, mit den Werten 0 und 1, dargestellt. Ein Wert von 1 bedeutet, dass die jeweilige Funktion zum Zeitpunkt $\boldsymbol{t_1}$ angesteuert und über das Zeitintervall $\boldsymbol{t_{d1}}$ ausgeführt wird. Ein Wert von 0 bedeutet, dass die Funktion zum Zeitpunkt $\boldsymbol{t_2}$ und über das Zeitintervall $\boldsymbol{t_{d2}}$ nicht angesteuert wird.

2.1.19 Platzieren

Das Platzieren (engl.: „place-and-route“) ist der Prozess der Abbildung des erstellten FPGA-Designs auf den FPGA-Baustein sowie das logische Verbinden der elektronischen Komponenten zur Realisierung der FPGA-Funktionen.

2.1.20 Synthese

Die Synthese ist ein Prozess, bei dem basierend auf vorgegebenen Eigenschaften ein Programm, beispielsweise eine Netzliste für ein FPGA, erzeugt wird. Die Eigenschaften sind dabei durch Modelle, Code oder Einschränkungen vorgegeben.

Anmerkung: Im Rahmen des Vorhabens wird bei der Synthese C/C++ Code als Quelle verwendet und in HDL Code umgewandelt, dessen funktionales Verhalten äquivalent zum C/C++ Code sein soll.

2.2 Abkürzungen

Im Folgenden sind die Abkürzungen definiert, die in diesem Bericht verwendet werden.

ASIC	Application-Specific Integrated Circuit
BIST	Built-In-Self-Test
CPU	Central Processing Unit
CPLD	Complex Programmable Logic Device
EDIF	Electronic Data Interchange Format
EN	Europäische Norm
EU	Europäische Union
FB	Funktionsblock
FP	Funktionsplan
FPGA	Field Programmable Gate Array
FMEA	Failure Mode and Effect Analysis
FPLA	Field Programmable Logic Array
HDL	Hardware Description Language
HPD	(HDL) Programmed Device
HW	Hardware
I&C	Instrumentation and Control
IAEA	International Atomic Energy Agency
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronic Engineers
ISO	International Organization for Standardization
ISTec	Institut für Sicherheitstechnologie
IDE	Integrated Development Environment
KKW	Kernkraftwerk
LOC	Lines of Code
PAL	Programmable Array Logic

PLA	Programmable Logic Array
PLD	Programmable Logic Device
PROM	Programmable Read-Only Memory
SOC	System on a Chip
SPLD	Simple Programmable Logic Device
SSG	Specific Safety Guide
SW	Software
TR	TÜV Rheinland
V&V	Verification and Validation

3 STAND VON WISSENSCHAFT UND TECHNIK ZUM FEHLERPOTENTIAL UND ZU FEHLERMODI BEI FPGA-DESIGNS

Der Teilbericht [8] (siehe Anhang A) fasst unter anderem den Stand von Wissenschaft und Technik zur Analyse der Fehlermodi von programmierbaren logischen Schaltungen in der Sicherheitsleittechnik von Kernkraftwerken zusammen. Dazu wurden bisherige Arbeitsergebnisse bezüglich einer Methode zur Messung der Komplexität der Anwendungssoftware leittechnischer Automatisierungssysteme herangezogen. Die Ergebnisse des Vorläuferprojekts zeigen, dass die erarbeitete Methode der Komplexitätsmessung zusammen mit dem entwickelten Instrumentarium der Analyse-Werkzeuge auf verschiedene programmierbare logische Schaltungen in der Sicherheitsleittechnik von Kernkraftwerken anwendbar ist [9].

Zu den Fehlermodi digitaler Leittechnik im Allgemeinen, aber auch zu programmierbarer Logik wie FPGAs, sowie den Ursachen der Fehlermodi, den Auswirkungen und Möglichkeiten zur Behandlung, gibt es bereits eine Vielzahl an Literatur. Allerding beschäftigen sich die Studien mehrheitlich mit hardwarebasierten Fehlermodi. Dies ist insofern nicht verwunderlich, als dass der FPGA-Chip selbst keine Software oder ein Betriebssystem beinhaltet [19]. Es wurde im Rahmen der Literaturrecherche deutlich, dass auch softwarebasierte Fehlermodi betrachtet werden müssen. Insbesondere durch den vermehrten Einsatz vorgefertigter Software und der immer stärker vertretenen werkzeuggestützten Umsetzung von FPGA-Designs, rücken softwarebasierte Fehlermodi in den Vordergrund. Es zeigte sich, dass die Ursachen softwarebasierter Fehlermodi hierbei im Entwicklungsprozess und vor allem beim Entwurf des FPGA-Designs [2], [15] liegen.

Für das Testen von FPGA-Designs spielen neben den normativ geforderten Verifizierungs- und Validierungsverfahren auch die verwendeten Entwicklungswerkzeuge eine große Rolle. Es handelt sich hierbei in der Regel um Werkzeug-Sets, die außer dem reinen Entwurf des FPGA-Designs noch weitere Testwerkzeuge anbieten. Sofern innerhalb eines Werkzeug-Sets sowohl der Entwurf als auch gewisse Tests des FPGA-Designs erfolgen, muss neben der Korrektheit des Testwerkzeugs ggf. auch die Unabhängigkeit der Testmethode vom verwendeten Entwicklungswerkzeug nachgewiesen werden. Obwohl Werkzeug-Sets einige Verifizierungsschritte unterstützen können, ist es nicht ausreichend die V&V-Ergebnisse allein von diesen Werkzeugen abzuleiten. Es bedarf in jedem Fall einer unabhängigen Verifizierung und Validierung. Die Korrektheit der implementierten Funktionalität einer FPGA-Anwendung lässt sich mittels Funktionstests und statischen Analysen nicht vollständig nachweisen [18], [19]. Im Vergleich zu CPU-basierten Lösungen sind andere Teststrategien erforderlich, die das zeitliche Verhalten bzw. die Dynamik der Logik analysieren.

Der in Kapitel 4.1 ermittelte Sachstand von Wissenschaft und Technik zu programmierbaren logischen Schaltungen dient als Grundlage für die Auswahl geeigneter Entwicklungswerkzeuge zur Erstellung von FPGA-Designs für die Untersuchungen innerhalb des Vorhabens.

3.1 Fehlermodi bei integrierter digitaler Leittechnik

FPGAs sind eine Form programmierbarer digitaler Hardwarebausteine, die für die Ausführung digitaler Logikfunktionen konfiguriert werden können. Die Programmierung wird mit einer HDL durchgeführt, d.h. FPGAs sind eine Form von HPD. In der Leittechnik von KKW werden FPGAs u.a. im Rahmen von Nachrüstmaßnahmen und zum Ersatz veralteter Systeme eingesetzt.

Fehlermodi lassen sich durch eine Vielzahl von Merkmalen klassifizieren. Sie werden in der Literatur zumeist in drei große Gruppen unterteilt:

- Entwicklungs- oder Entwurfsfehler,
- physikalische Fehler,
- Interaktionsfehler.

Die erste Gruppe ist typisch für Software. Entsprechende Ausfälle treten unter bestimmten Bedingungen (z.B. Eingabedaten) auf. Die zweite Gruppe betrifft zumeist Hardware und wird durch natürliche Ursachen (wie Alterung) verursacht. Die dritte Gruppe ist eine Folge externer Effekte (wie nicht autorisierte Handlungen und Bedienfehler) [2], [3] und [15].

Abgebildet auf den Lebenszyklus digitaler Leittechnik lassen sich die Fehlermodi von FPGAs auf Grundlage deren Ursache in den beiden Bereichen Entwicklung und Betrieb aufteilen [15].

Innerhalb dieses Forschungsvorhabens liegt der Fokus auf softwarebasierten Fehlermodi, daher wird im Weiteren nicht genauer auf physikalische Fehler, deren Ursache in der Hardware (z.B. im Herstellungsprozess) liegt, eingegangen. Die Ursachen der Fehlermodi während des Betriebs umfassen ebenfalls zum Großteil Hardwarefehler, wie z.B. durch Strahlung, Umgebungseinflüsse und Alterungsprozesse. Die Fehlermodi, deren Ursache im Bereich Betrieb auch innerhalb der Software liegt, beziehen sich auf Wartung (bzw. Modifikation) und Cybersecurity.

Die Fehlermodi, deren Ursache im Bereich Entwicklung liegt, sind hingegen mehrheitlich auf fehlerhafte Software zurückzuführen. Hierzu gehören vor allem Fehlermodi mit Bezug auf:

- Zeitliche Aspekte (Signallaufzeiten)
- Logische Fehler (HDL)
- Zustandsautomat (engl.: state machine)
- Eingangssignale und Datentypen
- Soft Processor (d.h. ein vollständig mittels Logiksynthese implementierter Mikroprozessor)
- Vorgefertigte Software
- Wartbarkeit der Software
- Cybersecurity (im Entwicklungsprozess)

Cybersecurity umfasst bei digitaler Leittechnik neben dem Entwurf (Design) auch die Phasen Systemintegration (Engineering) und Betrieb (Operation) [16]. Da für digitale Leittechnik gemäß IEC 62645 Ed. 2 [17] bezüglich der Cybersecurity vor allem böswillige Handlungen berücksichtigt werden, wird auf Fehlermodi, deren Ursachen im Bezug zur Cybersecurity stehen, im Weiteren nicht näher eingegangen.

Es wird deutlich, dass mit Ausnahme von Modifikation und Cybersecurity die Ursache aller softwarebasierten Fehlermodi im Entwicklungsprozess liegt. Dies ist insbesondere deshalb bemerkenswert, da der FPGA-Chip selbst keine Software oder ein Betriebssystem beinhaltet [19]. Die Ursachen softwarebasierter Fehlermodi liegen daher bei der Konfiguration der FPGA-Logik mittels HDL-Code. Die Fehlermodi von FPGAs aus dem Bereich Entwicklung lassen sich anhand ihrer Ursache innerhalb des Entwicklungslebenszyklus aufteilen [15]. Das Auftreten bestimmter Fehler kann an bestimmte Phasen im Lebenszyklus von Leittechnik gebunden sein. Die Ursachen dieser Fehler können dabei bereits in früheren Lebenszyklus-Phasen begründet sein.

Im Umgang mit Fehlermodi bei FPGA-basierter Leittechnik können verschiedene Strategien angewendet werden [18].

- Fehlervermeidung: Durch geeignete Qualitätssicherungsmaßnahmen soll das Auftreten von Fehlern verhindert werden.
- Fehlerbeherrschung: Durch geeignete administrative und technische Maßnahmen (spezielle Handlungsanweisungen im Fehlerfall, Maßnahmen zur Erreichung eines sicheren Zustands) sollen die Auswirkungen von auftretenden Fehlern begrenzt werden.
- Fehlertoleranz: Durch geeignete technische Maßnahmen (z.B. Redundanz, Diversität) sollen negative Auswirkungen auftretender Fehler eingedämmt und die Funktion betroffener Systeme gewährleistet werden.

Eine Besonderheit von FPGA-basierter Logik ist das vergleichsweise komplexe Zeitverhalten. Daher sind statische Analysen und Prüfmethoden für die meisten FPGA-Anwendungen nicht geeignet, um das dynamische Verhalten abzubilden und die Korrektheit des FPGA-Designs zu verifizieren [19]. Für die Behandlung von FPGA-Fehlermodi sind daher Testmethoden, die das zeitliche Verhalten bzw. die dynamischen Eigenschaften der Logik analysieren, erforderlich. Die entsprechenden Testmethoden sind in Kapitel 3.3 zusammengefasst.

3.2 Fehlermodi im Zusammenhang mit modellbasierten FPGA-Entwicklungswerkzeugen

In der Regel kommen bei der Erstellung von FPGA-Designs Entwicklungswerkzeuge (Hardware Design Tools) zum Einsatz. Die Entwicklungswerkzeuge für FPGA-Designs sind dabei häufig Bestandteil eines Werkzeug-Sets, das zusätzlich zu dem eigentlichen Entwurf des FPGA-Designs noch Validierungs- und Testwerkzeuge beinhaltet. Diese sollen die Robustheit bzw. Fehlertoleranz der FPGA-Anwendungen und damit die Zuverlässigkeit der Leittechniksysteme erhöhen [18]. Die folgende Betrachtung fokussiert sich hierbei auf modelbasierte („model-based design“) Entwicklungswerkzeuge. Bei diesen modelbasierten Werkzeugen bilden virtuelle Modelle den Mittelpunkt Ihres Entwicklungsprozesses und sollen somit die Art und Weise, komplexe Anwendungen erfolgreich umzusetzen verbessern und die Entwicklungsdauer verkürzen. Bei der Betrachtung von modellbasierten Entwicklungswerkzeugen für das Erstellen von FPGA-Designs sind mehrere Faktoren bei der Bewertung und Auswahl dieser Werkzeuge zu berücksichtigen:

- Lesbarkeit, Länge und Qualität des generierten Codes
- Bedienungsfreundlichkeit und Lernkurve. Dank der höheren Abstraktionsebene, die in der modellbasierten Hardware-Entwurf-Methodik durch die Werkzeuge verwendet wird, ist es möglich, komplexe Algorithmen viel schneller zu erstellen, zu testen und zu implementieren.
- Verfügbarkeit vorhandener Hardware-Bibliotheken.
- Nachvollziehbarkeit (Übersetzung der angewendeten Programmiersprache zur HDL-Beschreibung).
- Kontrollierbarkeit und Rückverfolgbarkeit der generierten Anwendung.
- Feinabstimmung des generierten Codes auf modellbasierter Abstraktionsebene z.B. zur Beschleunigung der Methodik.

Die durchgeführten Recherchen ergaben, dass bereits mehrere modellbasierte Entwicklungswerkzeuge für FPGA-Designs auf dem Markt sind. Die folgende Tabelle gibt einen Überblick zu gängigen Entwicklungswerkzeugen. Für jedes Werkzeug ist zudem die jeweils verwendete Sprache für den generierten Code aufgeführt. Obwohl die meisten Werkzeuge HDL-Code erzeugen, kann sich die Codeausgabe erheblich unterscheiden.

Tabelle 3: Entwicklungswerkzeuge zur Erstellung von FPGA-Designs

Hersteller	Produkt	Sprache
Mathworks	Simulink HDL Coder	HDL
	Real-Time Workshop	ANSI C, C++
Xilinx	System Generator	HDL
	Accel DSP	HDL
Silicon Software	VisualApplets	HDL
Synplicity	Synplify DSP	HDL
Altera	DSP Builder	HDL
	SOPC Builder	HDL
Universität Leiden	Compaan/Laura	HDL
Bluespec	Bluespec System Verilog	Verilog
Synfora	Pico Express	RTL/SystemC
Agility	Agility MCS	C
	DK Design Suite	HDL
Scilab	Scicos-HDL	HDL

Die Fehlermodi der Entwicklungswerkzeuge zur Erstellung von FPGA-Designs entsprechen im Allgemeinen den Fehlermodi von Software-Entwicklungswerkzeugen. Dazu gehören:

- Fehlerhafte Editoren zur Generierung der FPGA-Netzlisten.
- Fehlerhafte Umsetzung des erzeugten logischen Funktionsplans in Code, durch z.B. Compiler und Codegeneratoren.
- Fehlerhafte Bibliotheken-Inhalte.

Daneben können spezifische Fehlermodi der Werkzeuge beim Hardware-Entwurf auftreten:

- Fehlerhafte Umsetzung des generierten Codes in ein Chip-Layout.
- Fehlerhafte Übertragung (Programmierung) des erstellten Layouts in den Chip.

Ebenfalls ist der Faktor Mensch, als Entwickler oder Bediener, bei der Fehlerbetrachtung zu berücksichtigen, z.B. durch:

- Fehlerhafte Konzeption und Spezifikation vor dem Einsatz der Werkzeuge,
- Fehlerhafte Bedienung der Werkzeuge

Da im Lieferumfang der gängigen Entwicklungswerkzeuge in der Regel auch Prüf- bzw. Verifizierungswerkzeuge enthalten sind, sind auch Fehlermodi, die beim Testen des FPGA-Designs nicht erkannt werden, zu berücksichtigen.

- Fehlerhaftes Prüfwerkzeug
- Falsche Anwendung der Prüfwerkzeuge (Testprofil, Testfallgenerierung)
- Mangelnde Unabhängigkeit zwischen Entwicklungs- und Prüfwerkzeug

3.3 Testen von FPGA-Designs

Für FPGA-Anwendungen, die Leittechnik-Funktionen der Kategorie A ausführen [11], sind die grundlegenden Qualifizierungsanforderungen und somit auch Testanforderungen in IEC 62566 Ed. 1 [4] festgelegt. Daneben sind Testanforderungen auch in IAEA-Publikationen [20], [21], [22] beschrieben und begründet. Abbildung 2 zeigt die Lebenszyklusphasen programmierbarer logischer Schaltungen gemäß IEC 62566 Ed. 1 [4]. Der Standard verwendet hierbei den Ausdruck HPD für Bausteine, die mittels einer Hardware-Beschreibungssprache (HDL) programmiert sind und in diesem Bericht als programmierbare logische Schaltungen bezeichnet werden.

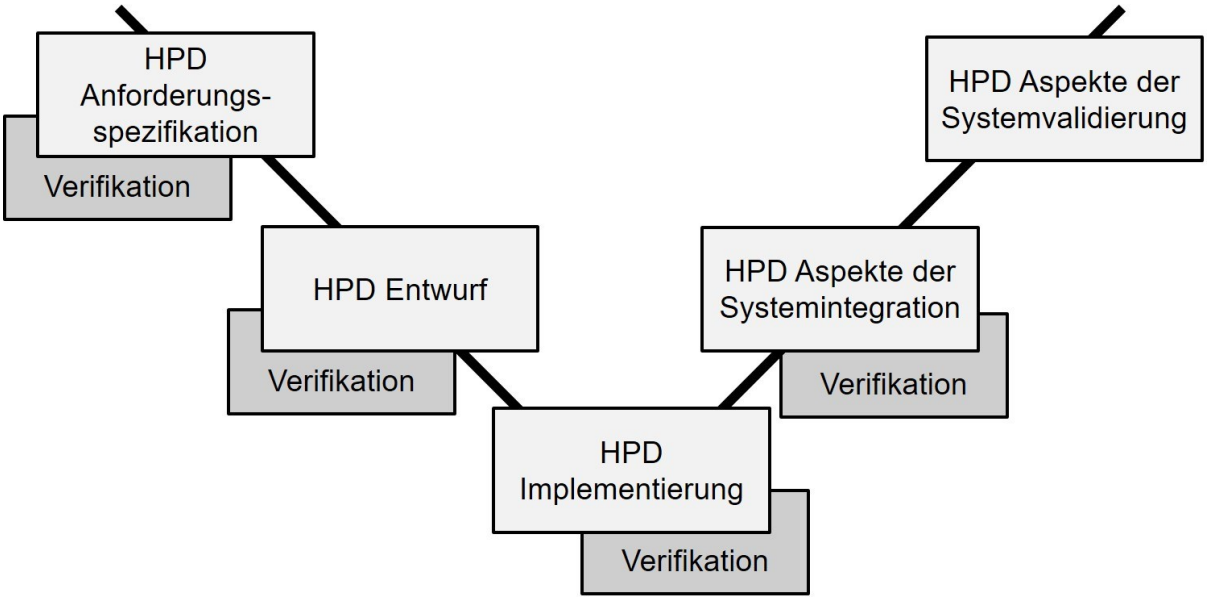


Abbildung 2: Entwicklungslebenszyklus programmierbarer logischer Schaltungen (HPD) [4]

Für alle Entwicklungsphasen sind zur Gewährleistung sowie zum Nachweis der notwendigen Qualität eine unabhängige Verifizierung und Validierung durchzuführen und zu dokumentieren. In diesem Zusammenhang kommen Test-Werkzeuge, Test-Strategien und Test-Methoden zum Einsatz, die eine

- unabhängige Verifizierung der jeweiligen Lebenszyklusphase ermöglichen (korrekte Durchführung der Phasen),
- korrekte Ausführung des Entwicklungslebenszyklus validieren (gegen FPGA Anforderungen prüftechnisch bestätigen), und
- korrekte Ausführung phasenübergreifend und auf Systemebene validieren (gegen Systemanforderungen prüftechnisch bestätigen).

Die Anforderungsspezifikation bildet unter anderem Anforderungen an die Qualitätssicherung, relevante Entwicklungsstandards sowie lebenszyklusrelevante Anforderungen ab. Zudem gibt es dezidierte Anforderungen zu den internen und externen Schnittstellen, also den Bereichen, an denen die FPGA-Anwendung in ein bestehendes System eingebunden ist. Ein Test der Korrektheit dieser Anforderungen erfolgt in der Regel mittels Konfigurationswerkzeugen. Die Anforderungsspezifikation wird gemäß den kerntechnischen Regelwerksanforderungen mittels entwicklungsunabhängiger Testverfahren verifiziert. Insbesondere bei komplexeren FPGA-Designs kommen hierbei auch Prüfwerkzeuge, z.B. bei der Regelwerksumsetzung, Tests zur Konfiguration, Anforderungsverfolgung und Dokumentation zum Einsatz. Fehler oder

Mängel in der Anforderungsspezifikation wirken sich unmittelbar auf den kompletten Lebenszyklus der FPGA-Entwicklung aus. Ein besonderes Augenmerk gilt daher den Entwicklungswerkzeugen, die innerhalb eines Werkzeug-Sets schon in der frühen Phase der Entwicklung in die Prüfung der Spezifikation eingebunden sind.

Beim Entwurf wird die Konfiguration eines FPGA-Designs grafisch mittels eines Schaltplans oder textuell mit einer Hardwarebeschreibungssprache erstellt, welche die gesamte Funktion der Schaltung in Form von Strukturen und Abläufen beschreibt. Als Grundlage der Entwicklungswerkzeuge für FPGA-Designs haben sich Sprachen wie VHDL [23], Verilog [24] und SystemC [25] etabliert. Verilog gilt dabei neben VHDL als die meistgenutzte Hardwarebeschreibungssprache. Für die Entwurfserstellung kommen Werkzeuge wie z.B. LabVIEW [26] oder Matlab/Simulink [27] zum Einsatz, die über eine nutzerfreundliche graphische Oberfläche verfügen. Mit einem Synthesewerkzeug erstellt die verwendete Hardware-Beschreibung dann in mehreren Schritten für einen gewünschten Baustein eine spezifische Netzliste unter Nutzung der in diesem FPGA-Baustein verfügbaren Ressourcen. Die im Anschluss an die Hardware-Beschreibung durchzuführende Verifizierungsprüfung erfolgt in der Regel auch werkzeugbasiert (z.B. mittles ModelSim [28]).

Für das Testen von FPGA-Designs spielen neben den angewendeten Testmethoden vor allem auch die verwendeten Entwicklungswerkzeuge eine große Rolle. Hersteller und Lieferanten von FPGA-Technologien vermarkten in der Regel nicht nur die FPGA-Hardware, sondern zusätzlich auch Entwicklungs-, Test-, Simulations- und Verifizierungswerkzeuge. Diese Werkzeug-Sets werden jeweils in einem Entwicklungswerkzeug zusammengefasst. Als Beispiel werden für drei der auf dem kerntechnischen Markt etablierten FPGA Hersteller bzw. Lieferanten (Actel[1], Altera[2], Xilinx[3]) die jeweiligen Entwicklungswerkzeuge betrachtet.

Tabelle 4: Hersteller und Typen von FPGAs

FPGA-Typ	Hersteller/Lieferant	Design-Werkzeug
ACTEL [29]	Actel GmbH; Dachorganisation: Microchip Technology Inc.	IDE: Libero Integrated Design Environment (IDE) Microsemi [29]
ALTERA [30]	ALTERA Cooperation; Dachorganisation INTEL	Quartus: Intel „Quartus“ FPGA Design Software [30]
XILINX [24]	Xilinx Cooperation	Vivado: Vivado Design Suite [24]

[1] Der Bezug zum internationalen kerntechnischen Markt erfolgt über den FPGA-Hersteller Actel. Durch eine Übernahme ist Actel heute Teil der Microchip Technology Inc.

[2] Der Bezug zum internationalen kerntechnischen Markt erfolgt über den FPGA-Hersteller Altera. Durch eine Übernahme ist Altera heute Teil von INTEL.

[3] Der Bezug zum internationalen kerntechnischen Markt erfolgt über den FPGA-Hersteller XILINX. Durch eine Übernahme ist XILINX heute Teil von AMD.

- Libero ist laut Hersteller [29] ein speziell für Microsemi-FPGAs konzipiertes „Softwarewerkzeug-Set“, welches die erforderlichen Werkzeuge für die Entwicklung, Evaluierung und Verifizierung der FPGA-Designs bereitstellt.
- Quartus [30] ist ein speziell für Intel® FPGAs konzipiertes „Softwarewerkzeug-Set”, welches auch für CPLD eingesetzt wird und den gesamten FPGA Entwicklungsprozess einschließlich Synthese, Verifizierung und Simulation unterstützt.
- Vivado [24] wird von Xilinx als Input/Output Werkzeug-Set beschrieben, welches zur Entwicklung, Simulation, Platzierung („place-and-route“) und Synthese der FPGA-Designs eingesetzt wird. Darüber hinaus können spezielle Peripherien auf dem FPGA implementiert werden.

Gemäß IEC 62566 Ed. 1 [4] und IAEA NP-T-3.17 [21] sind Verifizierung und Validierung für HPD unabhängig von der Entwicklung durchzuführen. Dabei können z.B. folgende Testmethoden angewendet werden [15], [31], [32]:

- Statische Verifizierung: In IEC 62566 Ed. 1 [4] Kap 9.8 erfolgt hier eine Prüfung des Quellcodes nach vorher festgelegten Kriterien (typischerweise werkzeugbasiert).
- Zeitverhalten: Das Zeitverhalten (wie Signallaufzeiten und Verzögerungen) ist präzise bei den Simulationen nachzubilden, um eine Verifizierung auf Chipebene zu ermöglichen.
- Testbänke: In IEC 62566 Ed. 1 [4], Kap 9.5 sind Anforderungen an Testbänke für FPGA-Anwendungen festgelegt. Demnach sind unter anderem Simulation und Testprogramm (sog. Testbank) zu entwickeln und zu dokumentieren.
- Fehlerinjektion: In IAEA NP-T-3.17 [21] wird die Fehlerinjektion als Technik empfohlen, um die Testabdeckung weiter zu verbessern. Durch gezieltes Einbringen von Fehlern kann das Fehlerverhalten von FPGA-Anwendungen untersucht werden. Weitere Techniken sind Stress-Tests zur Prüfung von Anforderungen an die Robustheit sowie Untersuchungen im Zusammenhang mit Common Mode Analysen.
- Emulation: Durch Emulatoren werden, anders als bei einer Simulation, Teile von FPGA-Designs für Verifizierungszwecke ausschließlich funktional nachgebildet.
- Post-Route-Analysen: Diese Prüfmethoden werden in der Regel in Testbänken umgesetzt. Vor Implementierung des Design-Flows auf dem Chip wird in einer Simulationsumgebung das FPGA-Design detailgetreu nachgebildet, getestet und analysiert. Die so ermittelten Parameter, wie z.B. das Zeitverhalten, werden dann unter anderem zur Verifizierung der Korrektheit der tatsächlichen FPGA-Anwendung genutzt.
- Selbstüberwachung und Diagnosen: Diese sog. „online“ Prüfmethoden, die sich auf die Robustheit der FPGA-Anwendung während ihres Betriebs auswirken, werden in der Regel über BIST realisiert. Sofern FPGA-Anwendungen in bestehende verteilte Leittechniksysteme integriert sind, werden zudem in der Regel die Selbstüberwachungsfunktionen des Leittechniksystems übergreifend auch für die FPGA-Anwendung genutzt. Anforderungen an die Nutzung von Selbstüberwachungen einschließlich BIST sind in IEC 62566 Ed. 1 [4], Kap. 6.4.9 festgelegt.

- Endlicher Zustandsautomat (engl.: finite state machine): Dieses Verifizierungsverfahren basiert auf einem Verhaltensmodell, bestehend aus Zuständen, Zustandsübergängen und Aktionen. Bei FPGA-Designs existieren dann je nach Komplexität Zustandsmatrizen, welche für eine endliche Anzahl an Zuständen die Kombinationen der jeweiligen Eingangs- und zugehörigen Ausgangsparameter abbilden. Dabei ist zu verifizieren, ob bei einer endlichen Zahl von Zuständen für sämtliche Kombinationen von Eingangsparametern der erwartete Zustand erreicht wird.
- Statische Laufzeit-Analyse (engl.: static timing analysis): Bei der statischen Laufzeit-Analyse wird eine Bewertung der tatsächlichen Signal-Laufzeit, ohne Nutzung einer Simulation des Schaltungsentwurfes, basierend auf Laufzeitberechnungen mittels kombinatorischer Logik, durchgeführt.
- Fehlerbeseitigungsprozess (engl.: fault resolution procedures): Gemäß IEC 62566 Ed. 1 [4] sind, übergeordnet zur Fehlerbeseitigung, die Prozesse von IEC 61513 Ed. 2 [12], Kap. 6.3.2.4 anzuwenden. Die Anwendung dieser Prozesse soll sicherstellen, dass bei Modifikationen die Anforderungen von Kapitel 12 der IEC 62566 Ed. 1 [4] eingehalten werden.
- Software-FMEA (Failure Modes and Effects Analysis) für FPGAs [15]: Diese Analyse beruht auf der systematischen Betrachtung der Fehlermodi und ihrer möglichen Auswirkungen zur gezielten Ermittlung möglicher fehlervermeidenden bzw. fehlerbeherrschenden Maßnahmen.

Die nach IEC 62566 Ed. 1 [4] und IAEA NP-T-3.17 [21] vorgestellten Testmethoden haben sich bewährt. Unterstützt durch wiederkehrende Prüfungen sowie Selbstüberwachungs-Routinen kann für FPGA-Anwendungen eine hohe Verlässlichkeit erzielt werden [33], [34] und [35]. Die für die Umsetzung der Testmethoden eingesetzten Werkzeuge sind oftmals Bestandteil des vom Hersteller bereitgestellten Werkzeug-Sets. Gemäß den Regelwerksanforderungen sind die Tests unabhängig von den verwendeten Entwicklungswerkzeugen auszuführen. Somit muss beim Einsatz vorgefertigter Prüfwerkzeuge neben der Korrektheit des Prüfwerkzeugs ggf. auch die Unabhängigkeit der Prüfmethode vom verwendeten Entwicklungswerkzeug nachgewiesen werden.

3.4 Vorangegangene eigene Projekte

Die Einführung und Weiterentwicklung immer komplexerer verfahrenstechnischer Systeme in den verschiedensten Bereichen führte auch in der Kerntechnik zu einer erhöhten Verwendung programmierbarer digitaler Leittechniksysteme mit sicherheitstechnischer Bedeutung. Die Frage nach einer Quantifizierung der Zuverlässigkeit dieser Systeme mithilfe eines allgemein anerkannten Verfahrens ist dabei noch nicht abschließend geklärt. Es gibt eine Reihe von Ansätzen, die meist Teilprobleme behandeln [6]. Die Ergebnisse der bisherigen Forschungsvorhaben [5], [7], [13], [14] und [9] leisten ebenfalls einen Beitrag zur Quantifizierung von Zuverlässigkeitseigenschaften auf der Basis von Komplexitätsberechnungen. Die Komplexitätsbewertung ist umso gebotener, da das deutsche kerntechnische Regelwerk neben Anforderungen zur Zuverlässigkeit auch Anforderungen an die Begrenzung der Komplexität von Sicherheitsleittechnik enthält [10].

Wie die praktische Erfahrung zeigt, ist es die Komplexität eines programmierbaren digitalen Systems in Verbindung mit der Vielfalt der Anwendungsprofile, wovon das Risiko einer Fehlfunktion des Systems entscheidend abhängt. Deshalb stellt sowohl die Bestimmung der Komplexität der Software bzw. der programmierbaren Logik als auch der Ansatz, die Zuverlässigkeit programmierbarer digitaler Leittechniksysteme auf der Grundlage der Komplexität der Software bzw. der programmierbaren Logik zu bewerten, eine wichtige Zielsetzung dar. Eine Methode zur Messung der Komplexität, die sich auf ein strukturiertes Vorgehen stützt, wurde in im Rahmen eines vorangegangenen Forschungsvorhabens [5] für eine spezifische CPU-basierte Leittechnikplattform entwickelt. Um das entwickelte Konzept für verschiedene digitale Leittechniksysteme verwendbar zu gestalten, wurde die Methodik im Rahmen des letzten Forschungsvorhabens [9] sowohl für CPU-basierte Anwendungen weiterentwickelt als auch für FPGA-basierte Anwendungen nutzbar gemacht.

Für die Komplexitätsbewertung wurde in [5] ein Verflechtungsmaß definiert, welches auf der Verschaltung der Logik eines Programms basiert. Neben dem Verflechtungsmaß werden weitere Komplexitätsaspekte herangezogen und zu einem Komplexitätsvektor zusammengefasst, der als Ausgangspunkt für die Bewertung und Ableitung von Zuverlässigkeitsaussagen dient. Die Berechnung des Verflechtungsmaßes basiert in [5] auf der Verschaltung von Funktionsbausteinen und Eingabe-Signalen eines Funktionsplans. Dabei können einzelne Funktionsbausteine oder auch ganze Sätze von Funktionsbausteinen zur Berechnung von mehr als nur einem Ausgangssignal verwendet werden. Diese Verflechtung V(FP) hinsichtlich der Berechnung verschiedener Ausgangssignale ist kennzeichnend für die Komplexität eines Funktionsplans. Eine weitere wesentliche Rolle spielt die Verflechtung, die durch die Verwendung gemeinsamer Eingangssignale gegeben ist.

Die Menge der Funktionsbausteine, die zur Berechnung eines Ausgangssignals S beitragen, bilden den Vorbereich VB(S) des Ausgabesignals S.

Die Menge der Eingangssignale, die zur Bildung des Ausgangssignals S verwendet werden, bilden den Input IN(S) des Ausgangssignals S.

Die Verflechtung V (FP) eines Funktionsplans ist definiert durch

$$V(FP) = \sum_i \frac{|VB_{FB}(S_{Ai})|+|IN(S_{Ai})|}{|BFP|+|SIN|} \quad (1)$$

Diese Größe hängt von der Verflechtung der Vorbereiche VB(S) und der Inputs IN(S) aller Ausgangssignale eines Funktionsplans ab.

Dabei bezeichnet:

S_{Ai}	die einzelnen Ausgangssignale des Funktionsplans FP
$VB_{FB}(S_{Ai})$	die Menge der Funktionsbausteine, die zur Berechnung des Ausgangsignals S_{Ai} beitragen
$IN(S_{Ai})$	die Menge der Eingang-Signale, die zur Berechnung des Ausgangssignals S_{Ai} beitragen
BFP	die Menge der Funktionsbausteine, aus denen der Funktionsplan FP besteht
SIN	die Menge der Eingangssignale des Funktionsplans FP
I I	die Mächtigkeit (Anzahl der Elemente) einer Menge

Mit den erarbeiteten Komplexitätsmaßen und dem daraus resultierenden Komplexitätsvektor wurde Software betrachtet, die z.B. bei CPU-basierten Anwendungen zum Einsatz kommt. Die Erweiterung der Methodik auf FPGA-basierte Anwendungen erfolgte in [9]. Dafür wurden zusätzliche Komplexitätsmaße zur Bewertung hinzugezogen und der Komplexitätsvektor entsprechend erweitert und anschließend für die Anwendung auf FPGA-basierte Anwendungen modifiziert. Der aktuelle Stand dieser Bewertungsgrößen, die als Grundlage für die Komplexitätsbewertung in dem laufenden Vorhaben herangezogen werden, ist in Tabelle 5 dargestellt.

Das zur Komplexitätsbewertung definierte Verflechtungsmaß (1) der Funktionspläne wurde um ein modifiziertes (2) und internes Verflechtungsmaß (3) erweitert, um eine höhere Aussagekraft der Messung der Komplexität zu erzielen. Das modifizierte Verflechtungsmaß $V_{mod}(FP)$ spiegelt die interne Verknüpfung einer Schaltung stärker wider.

$$V_{mod}(FP) = \sum_i \frac{|VB_{FB}(S_i)|+|IN(S_i)|}{|BFP|+|SIN|} \qquad (2)$$

Die Verflechtung wird bei diesem Maß über die Vorbereiche und die zur Berechnung herangezogenen Eingangssignale der internen Signale S_i berechnet. Hierbei steht S_i für alle Signale (Ausgabesignale und internen Signale) des Funktionsplans. Somit stellt $VB_{FB}(S_i)$ die Menge der Funktionsbausteine, die zur Berechnung der Ausgabesignale bzw. der internen Signale beitragen, dar.

Bildet man die Differenz

$$V_{intern}(FP) = V_{mod}(FP) - V(FP) \geq 0 \qquad (3)$$

ergibt sich eine Maßzahl $V_{intern}(FP)$, die unabhängig von der Anzahl der Ausgangssignale ist. Diese Zahl kann als eine Charakteristik der inneren Komplexität des Funktionsplans verstanden werden. Diese Differenz ist immer größer oder gleich Null, da $V_{mod}(FP)$ immer größer oder gleich V(FP) ist.

Die Berechnung des Verflechtungsmaßes basiert auf der Verschaltung der Funktionsbausteine bzw. Basisblöcke und der Eingangssignale des Funktionsplans bzw. der Netzliste, und ist sowohl auf Funktionspläne CPU-basierter Anwendungen als auch auf Netzlisten von FPGA-basierten Anwendungen anwendbar. Neben den Verflechtungsmaßen werden weitere Komplexitätscharakteristiken wie Anzahl der Ein- und Ausgangssignale, Anzahl der Bausteine bzw. Blöcke eines Funktionsplans bzw. einer Netzliste, Anzahl der Verbindungen sowie vor- und nachgelagerte Funktionspläne bzw. Netzlisten berücksichtigt. Aus den verschiedenen Komplexitätscharakteristiken wurde ein modifizierter Komplexitätsvektor gebildet, der auf FPGA-basierte Anwendungen anwendbar ist. Der Komplexitätsvektor ist Ausgangspunkt für die Ableitung von Kriterien zur Bewertung der Zuverlässigkeit digitaler Leittechniksysteme.

Tabelle 5: Komponenten des Komplexitätsvektors für FPGA-basierte Anwendungen

ID	Bedeutung
K_f1:	Anzahl der Basisblöcke
K_f2:	Anzahl der Eingangssignale
K_f3:	Anzahl der Ausgabesignale
K_f4:	Anzahl der Verbindungen
K_f5:	Anzahl der vorgelagerten Netzlisten
K_f6:	Anzahl der nachgelagerten Netzlisten
K_f7:	Komplexität der Verschaltung V(FP)
K_f8:	Komplexität der Verschaltung $V_{mod}(FP)$
K_f9:	Komplexität der Verschaltung $V_{intern}(FP)$

Für die automatisierte Ermittlung der Komplexitätsmerkmale und somit der Komponenten des Komplexitätsvektors wurden die prototypischen Skriptprogramme weiterentwickelt. Mithilfe der Werkzeuge lassen sich sowohl Funktionspläne (CPU-basierte Anwendungen) als auch Netzlisten (FPGA-basierte Anwendungen) auswerten. Die gewonnenen Ergebnisse fließen direkt in die Messung der Komplexität ein und können anschließend zur Ableitung von Kriterien zur Bewertung der Zuverlässigkeit des jeweiligen digitalen Leittechniksystems herangezogen werden. Für programmierbare logische Schaltungen lassen sich aus dem standardisierten EDIF-Netzlistenformat alle notwendigen Komplexitätscharakteristika für die Ermittlung der

Verflechtungsmaße direkt gewinnen, so dass ein Zwischenschritt über eine graphische Darstellung der Verschaltung nicht mehr notwendig ist. Netzlisten im EDIF-Dateiformat können unabhängig von Hersteller und Typ des jeweiligen FPGA-Chips ausgewertet werden.

Die praktische Anwendbarkeit des Messverfahrens auf Software bzw. FPGA-Strukturen, die mit graphischen Spezifikationen generiert wurden, konnte durch die Anwendung auf Leittechniksysteme verschiedener Hersteller nachgewiesen werden. Hierbei wurde auch die Aussagekraft des Verfahrens anhand der Gegenüberstellung der Komplexitätsvektoren ähnlicher Anwendungen aus der Kerntechnik dargelegt. Das erzielte Ergebnis umfasst eine automatisierte Methodik zur Ermittlung der Komplexität von FPGA-basierter Sicherheitsleittechnik und bietet damit die Möglichkeit verschiedene FPGA-Designs anhand ihrer Komplexität vergleichend zu bewerten.

4 ERSTELLUNG VON FPGA-DESIGNS ZUR UNTERSUCHUNG DER FEHLERMODI

Im Rahmen des Vorhabens wurden eigene FPGA-Designs für Leittechnikkomponenten erstellt. Hierfür wurde zunächst eine Entwicklungsumgebung für die Erstellung repräsentativer FPGA-Designs ausgewählt. Die Auswahl einer geeigneten Entwicklungsumgebung orientierte sich an marktüblichen Entwicklungswerkzeugen, welche zur Erstellung FPGA-basierter Leittechnikkomponenten in der Sicherheitsleittechnik von Kernkraftwerken typischerweise zum Einsatz kommen.

Für die Realisierung wurden geeignete repräsentative FPGA-Designvarianten ausgewählt. Die FPGA-Designs wurden auf Grundlage einer Spezifikation mithilfe der Entwicklungsumgebung realisiert. Art und Umfang der erstellten Designs bzw. Designvarianten wurden mit Blick auf weitere Untersuchungen hinsichtlich der Validierung von Testverfahren gewählt. Die FPGA-Designs weisen dabei eine hinreichende Komplexität auf, so dass in den nachfolgenden Untersuchungen praxisrelevante Aussagen gewonnen werden können. Das Design der Leittechnikkomponenten umfasst ausschließlich deren FPGA-Kern. Weitere zu einer realen Leittechnikkomponente gehörende Teile, wie Schaltregler für Betriebsspannungen, Schnittstellen, Schutzbeschaltungen von Ein- und Ausgängen und Selbstüberwachung, waren nicht Teil der Entwicklung.

Basierend auf der Entwicklung wurden geeignete Testumgebungen und Schnittstellen für die Testautomation erstellt. Es wurden hierfür verschiedene Werkzeuge zur Verifizierung genutzt. Für die Tests der FPGA-Designs wurden Testbänke geschaffen. Das Testen erfolgt softwarebasiert, die FPGA-Designs wurden nicht in Hardware konfiguriert. Nach Durchführung von Voruntersuchungen wurden Art und Umfang der ausgewählten FPGA-Designs, die Entwicklungsumgebung und der gewählte Ansatz zur Realisierung mit dem Auftraggeber in einem Projektgespräch am 17.03.2021 abgestimmt [51]. Die Art der Realisierung der Testumgebung wurde mit dem Auftraggeber im Projektgespräch am 18.10.2022 abgestimmt [52].

4.1 Werkzeugauswahl zur Erstellung und Prüfung von FPGA-Designs

Im Folgenden wird die Funktionalität von drei gängigen FPGA-Entwicklungswerkzeugen dargestellt, die den gesamten Design-Flow unterstützen: „Libero Integrated Design Environment (IDE)“ von Microchip Technology Inc. [37], „Intel Quartus” [38] von Intel und „Vivado Design Suite ” [39] von Xilinx. Die genannten Werkzeuge werden von den drei gängigsten Herstellern für den Entwurf von FPGA-Anwendungen in kerntechnischer Leittechnik angewendet. In den folgenden Unterkapiteln wird ein Überblick über deren Funktionalitäten hinsichtlich des Entwurfs und dem Testen von FPGA-Designs gegeben. Dabei werden Gemeinsamkeiten und Unterschiede der drei Werkzeuge dargestellt und die Auswahl eines geeigneten Entwicklungswerkzeugs für das Vorhaben erörtert.

Die Auswahl orientierte sich an FPGA-Entwicklungsumgebungen, welche in der Kerntechnik zum Einsatz kamen. Im laufenden Vorhaben wurde nach Auswahl der Entwicklungsumgebung eine aktuelle Version des Werkzeugs herangezogen, da sich sonst Probleme mit der Verfügbarkeit und Kompatibilität ergeben hätten.

4.1.1 Gemeinsamkeiten der gängigsten Entwicklungswerkzeuge

Eine Vielzahl an Funktionalitäten sind bei allen drei bisher genannten FPGA-Entwicklungswerkzeugen enthalten. Dazu gehören Werkzeuge zur zeitlichen Analyse, Energiebedarfsanalyse und Prüfung von Entwurfsregeln.

Werkzeuge zur zeitlichen Analyse schätzen das Zeitverhalten der entwickelten Anwendung ab. Abhängig vom jeweiligen Werkzeug beinhaltet dies Charakteristiken, wie das allgemeine Zeitverhalten, Verzögerungen, Verstöße gegen Einschränkungen und potenzielle Engpässe hinsichtlich zeitlicher Einschränkungen.

Werkzeuge zur Analyse des Energiebedarfs ermitteln den Energieverbrauch der entwickelten Anwendung. Damit lassen sich Informationen zu Aspekten wie dem Energieverbrauch einzelner Blöcke bis hin zu dem Gesamtentwurf sowie statischem, dynamischem und durchschnittlichem Energieverbrauch einholen.

Die Werkzeuge sind trotz der unterschiedlichen Hersteller mit diversen weiteren Werkzeugen und Anwendungen kompatibel. Zum Beispiel kann das Simulations-Werkzeug „ModelSim“ mit allen drei angegebenen FPGA Entwicklungswerkzeugen verwendet werden. Auch andere Drittanbieter-Werkzeuge und Software zur Synthese (beispielsweise „Synplify“ und „Precision“) werden ebenfalls von den betrachteten FPGA Entwicklungswerkzeugen unterstützt. Weiterhin verfügen die drei Entwicklungswerkzeuge über sogenannte „Viewer“, mit denen Einblick in die einzelnen Phasen des Lebenszyklus, wie z.B. die Zuweisung der Pins beim Platzieren auf die FPGA Zielarchitektur („place-and-route“), ermöglicht wird.

4.1.2 Actel Libero IDE

Das Entwicklungswerkzeug „Libero IDE“ wurde ursprünglich von dem FPGA Hersteller Actel angewendet, durch den der Bezug zum internationalen kerntechnischen Markt gegeben ist. Mittlerweile wurde Actel von der Microchip Technology Inc. übernommen, was sich in den Namen der aktuelleren Werkzeuge widerspiegelt.

Eine der besonderen Funktionen von „Libero IDE“ ist „SmartDesign“. Es erlaubt dem Entwickler die Erstellung eines sogenannten Top-Level Designs durch das Zusammenfügen verschiedenartiger Entwurfsblöcke. Damit können Entwurfsblöcke aus vorherigen Projekten wiederverwendet werden. Weiterhin lassen sich die „SmartDesign“-Komponenten mithilfe von „ModelSim“ auf für diese Werkzeuge übliche Weise simulieren.

Mithilfe des enthaltenen „Synopsys Identify Debuggers“ ist es zudem möglich, direkt beim Erstellen des Quellcodes erste Prüfungen und „Debugging“ durchzuführen. Die Art der damit erfassten Daten kann präzisiert werden, wodurch der Debugging-Prozess optimiert wird. Die Verwendung der eingebauten Funktion „Incremental Compile“ führt zu einer zusätzlichen Zeitersparnis bei der Iteration.

„Libero IDE“ kann zudem einschätzen, wie sich ionisierende Strahlung potenziell auf das die Wahrscheinlichkeit von Singe Event Upsets (SEU) auswirkt. Allerdings sind in dieser Hinsicht große Unterschiede zwischen verschiedenen Bausteinen zu vermerken. Um realistische Werte zu den potenziellen Effekten von ionisierender Strahlung zu ermitteln, muss in den jeweiligen bausteinspezifischen Leistungsberichten nachgeschlagen werden [37].

4.1.3 Altera Quartus

Das Entwicklungswerkzeug „Altera Quartus“ wurde ursprünglich von dem FPGA-Hersteller Altera zur Verfügung gestellt. Mittlerweile wurde Altera von der Firma INTEL übernommen, was sich auch im Namen der aktuelleren Werkzeuge widerspiegelt.

Eine spezifische Funktionalität von „INTEL Quartus“ sind die sogenannten „Megafunctions“. Dabei handelt es sich um komplexe Schaltungsblöcke, welche Aufgaben wie arithmetischen Funktionen, Gates, Eingabe/Ausgabe-Komponenten und Speicher erfüllen können. Der Entwickler kann entweder selbst Blöcke erstellen oder vorentwickelte Blöcke verwenden. Die Verwendung dieser speziellen Blöcke ist notwendig für den Zugriff auf bestimmte INTEL (bzw. vormals Altera) bausteinspezifische Funktionen.

„Altera Quartus“ verfügt zudem über das Werkzeug „Design Assistant“, womit die Zuverlässigkeit eines Entwurfes verifiziert werden soll. Inwieweit sich durch dessen Verwendung Aussagen zu einer gesteigerten Zuverlässigkeit bzw. verringerten Ausfallwahrscheinlichkeit oder Fehlerrate belasten lassen, ist jedoch fraglich. Die Entwurfsregeln, auf deren Basis diese Zuverlässigkeit geprüft wird, lassen sich vom Entwickler einstellen [38].

4.1.4 Vivado Design Suite (Neuimplementierung der ISE Design Suite)

Das Entwicklungswerkzeug „ISE Design Suite“ wurde vom FPGA Hersteller Xilinx angewendet. Mittlerweile werden keine Aktualisierungen mehr für „ISE Design Suite“ angeboten. Xilinx stellt mittlerweile die „Vivado Design Suite“ [40] als Nachfolger von „ISE Design Suite“ zur Verfügung.

„ISE“ besitzt sein eigenes Simulations-Tool namens „ISim“. Damit lässt sich sowohl eine Verhaltensanalyse zur Verifizierung funktionaler Anforderungen, als auch eine zeitliche Analyse zur Verifizierung von zeitlichen Anforderungen, durchführen. Da „ISim“ bereits in „ISE“ enthalten ist, sind gesonderte Simulationswerkzeuge wie z.B. „ModelSim“ nicht erforderlich. Allerdings verfügt „ISim“ nicht über bestimmte Funktionalitäten von „ModelSim“, wie zum Beispiel die Fähigkeit, Signallisten für spätere Verwendung zu speichern [39].

4.1.5 Auswahl der Vivado Design Suite

Unter den oben genannten Werkzeugen gibt es keine entscheidenden Unterschiede hinsichtlich ihrer Funktionalität. Alle der hier betrachteten Entwicklungswerkzeuge erfüllen die Voraussetzungen, welche für das Forschungsvorhaben als notwendig erachtet werden, und erscheinen daher gleichermaßen geeignet.

Für die Erstellung der FPGA-Designs im Rahmen des Forschungsvorhabens wird das Entwicklungswerkzeug „Vivado Design Suite“ des FPGA-Herstellers Xilinx genutzt. Bei der Auswahl eines geeigneten Entwicklungswerkzeugs hat vor allem der Aspekt den Ausschlag gegeben, dass die Produkte des Herstellers Xilinx die größte Verbreitung im kerntechnischen Sektor haben [43]. Aus Gründen der Verfügbarkeit und Kompatibilität wurde statt ISE die Neuimplementierung Vivado für die Entwicklung herangezogen.

Ein Vorteil des von Xilinx eingesetzten Werkzeugs Vivado sind die verfügbaren herstellerspezifischen Hilfswerkzeuge. Die Beschaffung erforderlicher zusätzlicher Hilfswerkzeuge für die Entwicklung entfällt dadurch. So erübrigt der in Vivado inkludierte

Vivado Simulator die Notwendigkeit der Beschaffung eines zusätzlichen Simulationswerkzeugs wie zum Beispiel das eingangs beschriebene ModelSim [28]. Zusätzlich ermöglicht das Werkzeug „Vitis HLS“ („High-Level Synthesis“) den Entwurf in Form von C/C++ Code zu erstellen und anschließend mittels Synthese in einen VHDL- oder Verilog-Entwurf umzusetzen. Dieser Entwurf kann dann in Vivado eingefügt und weiterbearbeitet werden. “Vitis HLS“ ermöglicht es somit, die bestehenden C/C++ Vorkenntnisse für die Erstellung von FPGA-Designs im Rahmen des Forschungsvorhabens nutzbar zu machen. Dies erspart die Verwendung von VHDL oder Verilog zur Entwurfserstellung und dementsprechend Zeit zur Einarbeitung in diese Sprachen zum Hardwareentwurf [42]. Es ist denkbar, dass die im Forschungsvorhaben angewendete indirekte Entwicklung in C/C++ Code inklusive HDL-Synthese eine zusätzliche Komplexität generiert, die bei einer direkten Implementierung in VHDL oder Verilog möglicherweise vermeidbar wäre. Bei der Bestimmung der Entwicklungsmethodik wurden die Vorteile hinsichtlich der Vorkenntnisse der Entwickler gegen die Risiken in Bezug auf eine potenzielle zusätzliche Komplexität abgewogen.

Die für die Erstellung der FPGA-Designs verwendete Entwicklungsumgebung sowie der gewählte Ansatz zur Realisierung wurden mit dem Auftraggeber in einer frühen Phase des Vorhabens in einem Projektgespräch am 17.03.2021 abgestimmt [51].

In dem Vorhaben kommen die Werkzeuge:

- Vitis HLS [42] zur HDL Synthese von C/C++ Code,
- Viviado Design Suite [40] zur Optimierung und Verifizierung, und
- ModelSim [28] zur unabhängigen Source-Code Verifizierung

zum Einsatz.

4.2 Spezifikation und Realisierung von FPGA-Designs

Im Rahmen des Forschungsvorhabens wurden drei repräsentative FPGA-Designs zur werkzeugbasierten Realisierung und weitergehenden Untersuchung ausgewählt. Hierbei handelt es sich um eine

1. Messsignalerfassung mit Plausibilitätsprüfung (Eingangsprüfung),
2. 2-von-3-Auswahlschaltung von drei Messsignalen mit Auslösung (Auswahlschaltung), und eine
3. Zusammenschaltung von drei (2-von-3) Auswahlschaltungen (Redundanzschaltung).

Diese drei FPGA-Designs repräsentieren typische kerntechnische Anwendungen und weisen jeweils eine unterschiedliche Komplexität für die weitergehenden Untersuchungen auf.

Zusammenfassend wurden die ausgewählten repräsentativen FPGA-Designs mit unterschiedlicher funktionaler Komplexität mit folgenden Zielsetzungen werkzeugbasiert realisiert und untersucht:

- Schaffung von Transparenz und Nachvollziehbarkeit bei der FPGA-Entwicklung durch Anwendung des in Kapitel 3.3 beschriebenen Phasenmodels.
- Nachweis der Wirksamkeit der in Kapitel 3.3 genannten Testmethoden.
- Umsetzung von kerntechnischen Regelwerksanforderungen.
- Diskussion und Anwendung weiterer Ansätze zum Testen von FPGA-Designs.

- Analyse der Komplexität von FPGA-Designs.

Mit der beschriebenen Zielsetzung wurde die Entwicklung der FPGA-Designs, beginnend mit der Anforderungsspezifikation über den Lebenszyklus bis hin zur (simulierten) Platzierung auf dem FPGA-Chip durchgeführt. Da die Implementierung selbst nicht weiter analysiert wurde, sind Umgebungseinflüsse nicht berücksichtigt; insofern hat die Realisierung des FPGA-Designs einen „Modellcharakter“.

Basierend auf dem Modellcharakter wurden für die Anforderungsspezifikationen der FPGA-Designs folgende Aspekte berücksichtigt:

- Es wurden ausschließlich regelbasierte Anforderungen an die Spezifikation berücksichtigt, die im Rahmen der Durchführung des Forschungsprojekts anwendbar sind.
- Der in IEC 62566 Ed. 1 [4] geforderte, und im Kapitel 3.2.1 erläuterte, Entwicklungslebenszyklus erfolgte gemäß dem in Abbildung 2 dargestellten Phasenmodell.
- Die Nicht-Anwendung einzelner Regelwerksanforderungen wurde begründet.
- Die einzelnen Entwicklungsschritte der FPGA-Designs sind in diesem Bericht nicht gesondert dokumentiert. Als Beispiel ist der funktionsrelevante Entwurf der 2-von-3-Auswahlschaltung in Abbildung 3 skizziert.

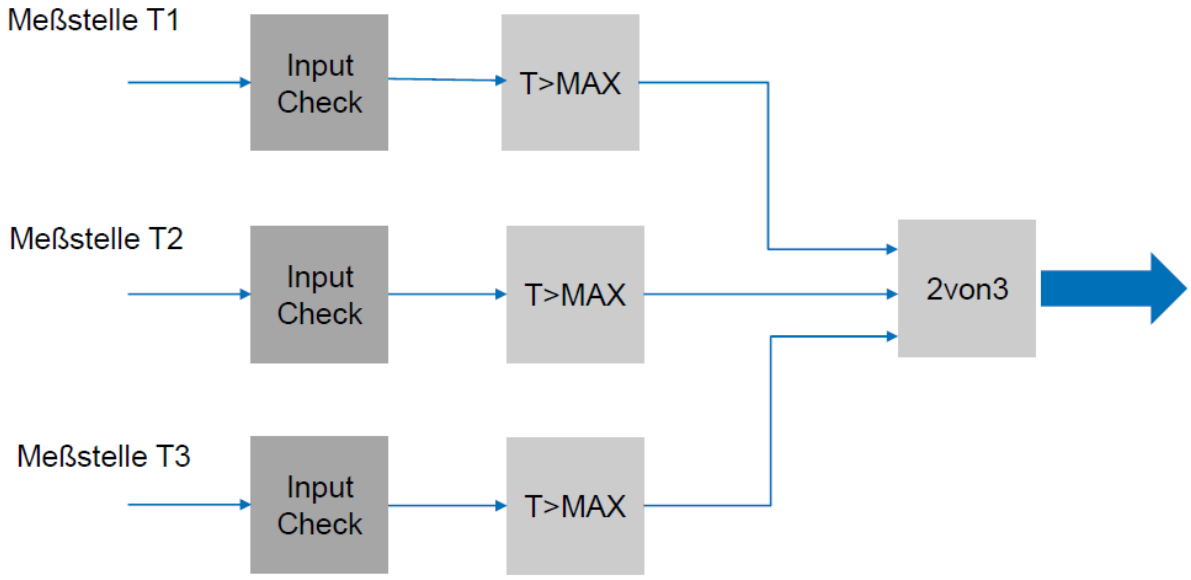


Abbildung 3: Funktionsrelevante 2-von-3 Temperatur-Auswahlschaltung

4.3 FPGA-Entwicklung mittels Vitis HLS

In diesem Kapitel wird die Entwicklungsmethodik mithilfe des Xilinx-Werkzeugs Vitis HLS [44]. In der üblichen Entwicklung wird der FPGA-Entwurf für einen spezifischen Zielbaustein (Hardware) programmiert und anschließend „debugged“. Auf diesen Entwicklungsschritt wird aufgrund fehlender Anwendung im Forschungsvorhaben nicht eingegangen.

Innerhalb eines typischen industriellen FPGA-Entwicklungslebenszyklus basiert die Erstellung des Codes üblicherweise auf einer Entwicklungsspezifikation. Es wird zunächst eine Systemspezifikation erstellt, also die Anforderungen an das System, in der ein FPGA (bzw. HPD) eingesetzt werden soll. Darauf basierend wird in der FPGA-Spezifikation festgelegt, was das FPGA leisten muss, um die Erfüllung der Systemspezifikation zu gewährleisten. Die konkrete FPGA-Spezifikation umfasst dabei sowohl Hardwareanforderungen als auch Anforderungen an die Software. Üblicherweise wird nach Ermittlung des geeigneten FPGA-Hardware-Typs der FPGA-Entwurf erstellt, welcher vor allem die Softwareaspekte des FPGA festlegt. Im nächsten Entwicklungsschritt stellt eine Modulspezifikation die Anforderungen an das zu entwickelnde Modul (die HDL-Funktion) dar, welcher der Entwickler bei der Erstellung des Entwurfes folgt. Die Erstellung des C/C++ Codes sollte bei der Entwicklung außerdem Programmierrichtlinien einhalten, welche innerhalb des Entwicklungsprozesses vorgegeben werden, oder z.B. von dem verwendeten Entwicklungswerkzeug (hier Vitis HLS) vordefiniert sein können.

Nachdem der Entwickler einen Arbeitsschritt bzw. die Arbeiten in einer Phase des Entwicklungslebenszyklus beendet hat, führt das Verifizierungsteam eine Verifizierung dieses Arbeitsschrittes gemäß den innerhalb der Entwicklung geplanten und vordefinierten Prüfmethoden durch. Zum Abschluss der Entwicklung wird eine Validierung (z.B. der Software) durchgeführt, d.h. die Einhaltung der Anforderungen der Spezifikation werden formal bestätigt.

Da es sich bei diesem Vorhaben um ein Forschungsprojekt handelt, enthält die entsprechende Modulspezifikation jeweils ausschließlich die zur Durchführung des Arbeitsplanes notwendigen Anforderungen. Auf Grundlage der Modulspezifikation wird in diesem Vorhaben C/C++ Quellcode erstellt. Die Einhaltung der funktionalen Anforderungen der Spezifikation wird mittels einer Testbank verifiziert (siehe Kapitel 4.3.1).

Mit dem Werkzeug Vitis HLS wird der erstellte C/C++ Code in HDL-Code umgesetzt. Der erstellte HDL-Code wird anschließend innerhalb von Vitis HLS per Analyse hinsichtlich Ressourcennutzung und zeitlichem Verhalten und mittels Simulation im Hinblick auf funktionale Korrektheit verifiziert. Im Anschluss wird der HDL-Code zur weiteren Bearbeitung in die Vivado Design Suite exportiert.

Bei der Entwicklung mit Vitis HLS erfolgt die Umsetzung der Modulspezifikationen in C/C++ Code. Der Vorteil gegenüber der Umsetzung in HDL ist bei C/C++ die schnellere Validierung und Iteration des Entwurfes, womit Entwicklungszeit erspart wird. Vitis HLS ermöglicht mit den bereits im Werkzeug vorhandenen Verifizierungsmethoden die Richtigkeit des Entwurfes ohne die Verwendung zusätzlicher Werkzeuge zu prüfen.

4.3.1 C/C++ Code

In diesem Vorhaben wird der Entwurf für die in der Spezifikation benannten funktionalen Anforderungen als C/C++ Code direkt in Vitis HLS geschrieben. Der Entwurf wird hierbei durch C/C++ Funktionen (einschließlich Hilfsfunktionen) realisiert.

Zum Schreiben des Codes stellt Vitis HLS einen Code-Editor bereit. Dieser überprüft die Syntax des Codes, womit beispielsweise Fehler wie inkorrekte Klammerung oder fehlende Semikolone gefunden werden. Es werden jedoch keine logischen Fehler, wie z.B. eine Zuweisung inkorrekter Datentypen, automatisch erkannt. Weiterhin bietet der in Vitis HLS

integrierte Code-Editor einige unterstützende Funktionen zur Bearbeitung des Codes, wie das Isolieren von Code-Abschnitten zu einer Funktion.

Um die Korrektheit des erstellten Codes zu prüfen, kann in Vitis HLS eine Testbank manuell erstellt und eingefügt werden, mit welcher das Design simuliert wird. Diese Testbank lässt das in C/C++ geschriebene Design mehrmals mit festgelegten Eingabewerten durchlaufen und vergleicht die Ergebnisse mit erwarteten Resultaten. Stimmt eine Ausgabe nicht mit dem erwarteten Wert überein, so gibt die C/C++ Simulation eine Fehlermeldung aus.

Zur Ermittlung der fehlerhaften Stellen im Code bietet Vitis HLS eine Debugging-Umgebung. Diese erlaubt es, den Code bei der Ausführung an bestimmten Stellen zu stoppen und die Daten zu diesem Zeitpunkt zu untersuchen. Somit lässt sich die fehlerhafte Stelle im Code präzise identifizieren, um sie anschließend manuell korrigieren zu können.

4.3.2 HDL Synthese

Die Synthese ist ein Prozess, bei dem auf Basis von bestimmten vorgegebenen Eigenschaften ein Design, beispielsweise eine Netzliste für ein FPGA, erzeugt wird. Die Eigenschaften des Designs können beispielsweise durch Modelle, Code oder Einschränkungen (siehe Kapitel 4.4.2) angegeben werden. Die in Vitis HLS ausgeführte Synthese verwendet C/C++ Code als Quelle und synthetisiert dazu HDL-Code, dessen funktionales Verhalten äquivalent zum C/C++ Code sein soll.

Mithilfe der HDL-Synthese wird bei der Entwicklung der C/C++ Entwurf zu einem HDL-Entwurf synthetisiert. Da innerhalb eines C/C++ Codes meistens mehrere Funktionen existieren, aber nur eine den angedachten Entwurf insgesamt beschreibt, muss die zu synthetisierende Funktion konkret in den Syntheseeinstellungen festgelegt werden.

Weil verschiedene FPGA-Typen unterschiedliche Charakteristiken aufweisen, welche die Synthese beeinflussen, muss der FPGA-Typ, also der Baustein auf den der Entwurf angewendet werden soll, festgelegt werden.

Nachdem die zu synthetisierende Funktion und der FPGA-Typ festgelegt sind, werden mittels Vitis HLS, auf Basis des C/C++ Codes, HDL-Dateien (sowohl in Verilog als auch in VHDL) erstellt. Bei der Synthese wird zudem ein Synthesebericht erzeugt, welcher erste Einblicke zum zeitlichen Verhalten und der Ressourcennutzung des Entwurfes verschafft. An dieser Stelle wird auch verifiziert, ob die synthetisierte HDL-Datei funktional äquivalent zum C/C++ Quellcode ist. Diese Verifizierung erfolgt mithilfe der C/RTL Co-Simulation (siehe Kapitel 4.3.5).

4.3.3 Analyse

Die Verifizierung des Entwurfes erfolgt über eine Analyse, die sich auf mehrere Eigenschaften stützt. So kann in einer Ansicht der zeitlichen Eigenschaften (dem „Schedule Viewer“) betrachtet werden, wie Operationen bzw. Kontrollschritte in der HDL-Datei zeitlich und in Abhängigkeit zueinander ausgeführt werden. In dieser Ansicht können weitere allgemeine Eigenschaften der Operationen bzw. Kontrollschritte, wie die Bit-Weite der Ausgabe angezeigt und somit geprüft werden (siehe Abbildung 4).

In dem in Abbildung 4 dargestellten Fenster „Module Hierarchy“ wird die Hierarchie der im Entwurf verwendeten HDL Funktionen angezeigt. Damit können die Eigenschaften über- und

untergeordneter Module analysiert werden, um beispielsweise die Anzahl der benutzten „Look-up Tables“ zu ermitteln. Leistung und Ressourcennutzung des jeweiligen Moduls können in den Fenstern „Performance Profile“ und „Resource Profile“ angezeigt werden. Im „Performance Profile“ werden die Schleifen angezeigt, die von den Modulen verwendet werden. Das „Resource Profile“ zeigt dagegen physische Aspekte an, wie die von den Modulen genutzten Ports und wie viele Bits diese Ports nutzen.

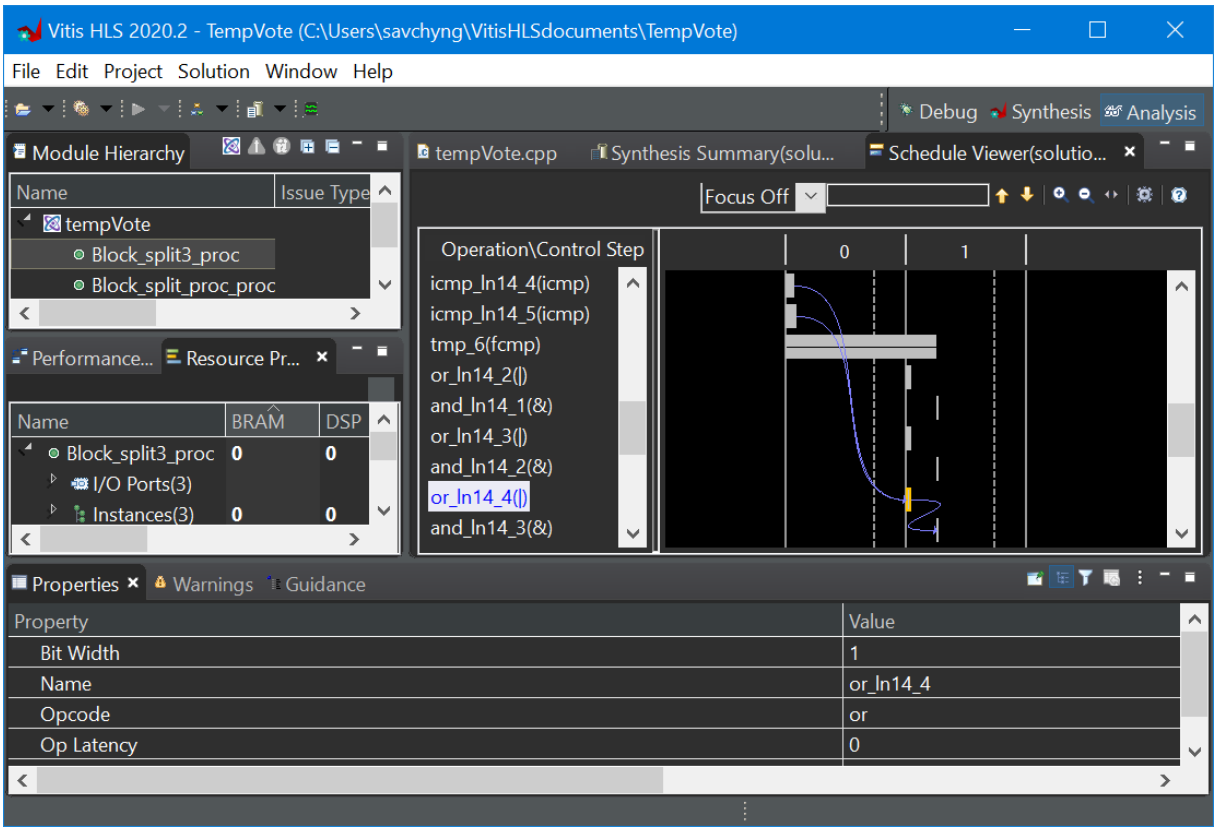


Abbildung 4: Schedule Viewer in Vitis HLS

4.3.4 Optimierung

Die manuelle Optimierung des Entwurfs findet iterativ gemäß den Vorgaben des Anwenders statt. Ergibt die Verifizierung, dass der erstellte Entwurf nicht den Modul-Anforderungen entspricht, müssen Änderungen in den entsprechenden Entwicklungsphasen durchgeführt und die zugehörigen Verifizierungsschritte wiederholt werden. Ebenso werden die darauffolgenden Entwicklungs- und Verifizierungsschritte erneut durchgeführt.

Sofern beispielsweise eine Anpassung der Spezifikation oder des Entwurfs (-Entwicklungs-Dokuments) von der Änderung betroffen ist, müssen auch hierfür die Verifizierungsschritte erneut durchlaufen werden. Der Änderungsprozess kann sowohl werkzeuggestützt als auch manuell, wie z.B. mittels Code Review, erfolgen. Hierbei ist die Unabhängigkeit zwischen den Entwicklungs- und Verifizierungsmethoden zu beachten.

4.3.5 Simulation und Export

Für eine noch präzisere Verifizierung der HDL-Datei kann die Wellenfunktion des Entwurfes angezeigt werden. Mit Wellenfunktion ist die graphische Darstellung der zeitlichen Änderung der Ein- und Ausgabewerte und der internen FPGA Signale (beispielsweise das „clock“-Signal)

gemeint. Die Verifizierung mittels der Wellenfunktion ermöglicht es, Diskrepanzen zwischen den simulierten Eingabe- und Ausgabewerten und dem erwarteten bzw. spezifizierten Verhalten festzustellen. Die ermittelbaren Diskrepanzen beziehen Abweichungen der jeweiligen Werte und Definitionsfehler mit ein.

Sollte sich der Entwurf bei diesem Schritt als fehlerhaft erweisen, kann entweder die HDL-Datei direkt umgeschrieben oder der C/C++ Code nachgebessert werden. Im letzteren Fall werden für Modifikationen, den gängigen kerntechnischen Standards entsprechend, die Schritte, beginnend mit 4.3.1, erneut ausgeführt.

Die Wellenfunktion wird im Zuge der C/RTL Co-Simulation erstellt, bei welcher der C/C++ Quellcode und die HDL-Datei einzeln simuliert und miteinander verglichen werden. Für die Generierung und Anzeige der Wellenfunktion (siehe Abbildung 5) wird das Xilinx Design Tool Vivado verwendet, welches partiell ebenfalls im Vitis HLS hinterlegt ist.

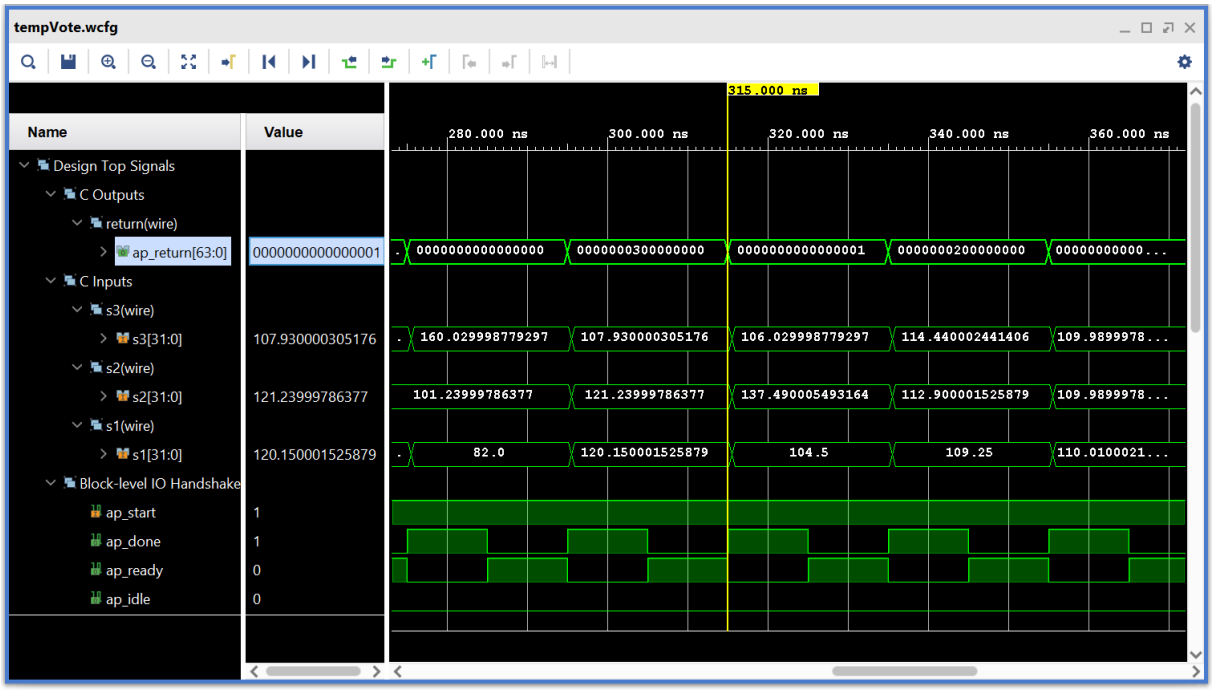
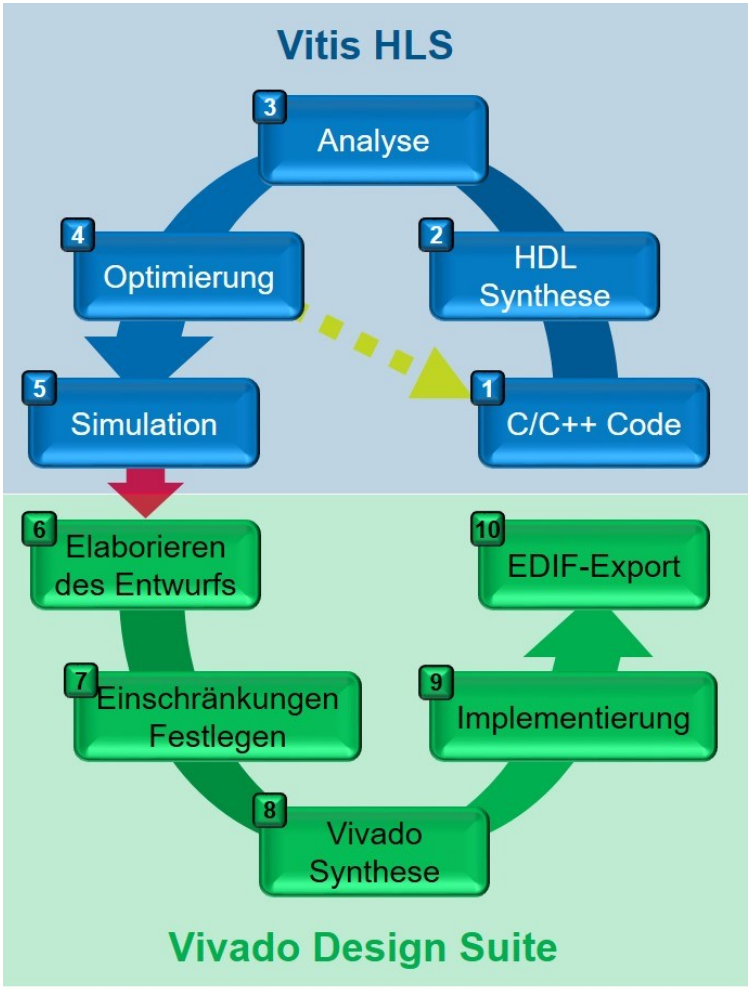


Abbildung 5: Wellenfunktion der synthetisierten HDL

Nach der Fertigstellung des gewünschten HDL-Entwurfes wird dieser für die weitere Bearbeitung in der Vivado Design Suite zu einer Vivado IP (Vivado Dateiformat) exportiert.

4.4 Entwicklung mittels Vivado Design Suite



In diesem Kapitel wird die FPGA-Entwicklungsmethodik mithilfe des Xilinx-Werkzeugs Vivado Design Suite [40] dargestellt. Der in Vitis HLS erstellte Entwurf wird in Vivado eingefügt und weiterbearbeitet. Mit dem FPGA Entwicklungswerkzeug Vivado von Xilinx, werden die Entwicklungsschritte von der Erstellung des Entwurfs bis hin zum Programmieren des FPGAs (einschließlich der Verifizierungsschritte) durchgeführt. Die Entwicklungsschritte, welche im Rahmen dieses Vorhabens ausgeführt werden (einschließlich der in Vitis HLS), sind in Abbildung 6 dargestellt. Die Optimierung in Schritt 4 wirkt sich direkt auf den C/C++ Code aus (gelber Pfeil). Von Schritt 5 nach Schritt 6 findet ein Wechsel der Werkzeugumgebung von Vitis HLS zur Vivado Design Suite statt (roter Pfeil).

Abbildung 6: Entwicklungsschritte in Vitis HLS und Vivado

Die Entwicklungsschritte, die mit der Vivado Design Suite durchgeführt werden, bauen auf dem in Vitis HLS generierten HDL-Code auf. In diesem Vorhaben wird bei der Erstellung der FPGA-Designs die Kompatibilität der Vivado Design Suite mit Vitis HLS genutzt. Die Kompatibilität der Entwicklungswerkzeuge erstreckt sich auch auf ausgewählte Drittanbieter-Produkte, wie z.B. ModelSim, welches für eine unabhängige Verifizierung genutzt wird.

In die Vivado Design Suite werden zunächst die vorher erzeugten Quelldateien importiert. Vivado akzeptiert hierfür HDL-Dateien, EDIF-Netzlisten und Dateien einiger weiterer Formate. Für dieses Vorhaben wird der in C/C++ erstellte und über Vitis HLS synthetisierte HDL-Entwurf als Quelldatei verwendet. Nach Verifizierung der funktionalen Korrektheit und Festlegung der

Einschränkungen (siehe Kapitel 4.4.2) lässt sich der Entwurf mit der Vivado eigenen Synthese weiterbearbeiten. Abschließend wird die Umsetzung des Entwurfs auf den Baustein ermittelt ohne diese tatsächlich auf eine Hardware zu schreiben. Daraufhin wird diese Umsetzung in Form einer Netzliste für die folgende Testung in eine EDIF-Datei exportiert.

4.4.1 Elaborieren des Entwurfs

Der aus Vitis HLS exportierte HDL-Entwurf kann in Vivado als Schaltplan grafisch dargestellt werden. Aus der grafischen Darstellung geht die Logik des Entwurfs hervor, womit potenzielle Fehler oder Anomalien im Entwurf frühzeitig erkannt werden können. Die manuelle Prüfung der Logik des Entwurfs dient als erste frühe Verifizierungsmethode.

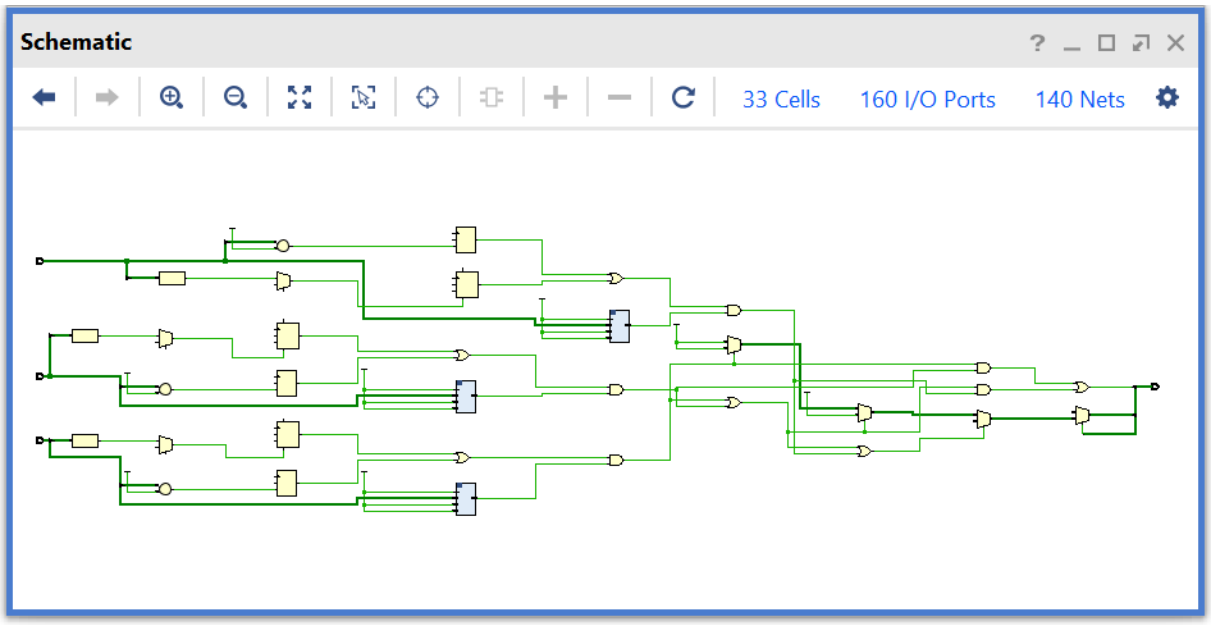


Abbildung 7: Beispiel eines Schaltplans mit 2-von-3-Auswahl in Vivado

Mit der Vivado Design Suite wird überdies geprüft (mittels eines automatisch generierten Berichts), ob im Entwurf bestimmte vom Werkzeug vorgegebene Entwurfsregeln eingehalten werden. In Vivado ist eine Vielzahl vorgefertigter Entwurfsregeln enthalten, die nach Bedarf eingesetzt oder ergänzt werden können. Diese Entwurfsregeln prüfen beispielweise, dass Ein- und Ausgabe-Ports miteinander kompatibel sind oder dass keine ungültigen Zuweisungen stattfinden.

Für eine noch detailliertere Verifizierung besteht die Möglichkeit, den Entwurf zu simulieren. Die Simulation vor der Vivado-Synthese (siehe Kapitel 4.4.3) wird dabei als Verhaltenssimulation bezeichnet. Ziel der Simulation ist es zu verifizieren, dass das Verhalten des Entwurfs (die von ihm ausgegebenen Werte) mit den Erwartungen übereinstimmt. Die erwarteten Ausgabewerte der Simulation müssen bei der Verifizierung manuell festgelegt werden. Hierfür (und in allen folgenden Simulationen) wird eine eigens in HDL geschriebene Testbank verwendet.

4.4.2 Festlegung von Einschränkungen

Die in der Vivado Design Suite verwendeten Einschränkungen sind Vorgaben, die bei den verschiedenen Syntheseschritten eingehalten werden müssen, damit der Entwurf geforderte Eigenschaften aufweist. Dabei wird zwischen zeitlichen und physischen Einschränkungen unterschieden. Die Einschränkungen können direkt zur Umsetzung der Modulspezifikation angewendet werden. Zeitliche Einschränkungen legen das Zeitverhalten des Entwurfes fest, wie z.B. die Verzögerung von ein- und ausgehenden Signalen. Physische Einschränkungen bestimmen hingegen, wie der Entwurf auf dem FPGA platziert werden soll, sowie die Platzierung der E/A Pins (siehe Abbildung 8).

Die Einschränkungen werden mithilfe der in Vivado vorhandenen Werkzeuge festgelegt und in XDC („Xilinx Design Constraints“) -Dateien gespeichert. Alternativ können schon vorhandene XDC-Dateien zum FPGA-Design hinzugefügt werden.

Um zu prüfen, ob der Entwurf die geforderten zeitlichen Einschränkungen tatsächlich einhält, wird der in der Vivado Design Suite automatisch generierte „Timing summary report“ betrachtet. Das Zeitverhalten wird auch in den folgenden Arbeitsschritten analysiert.

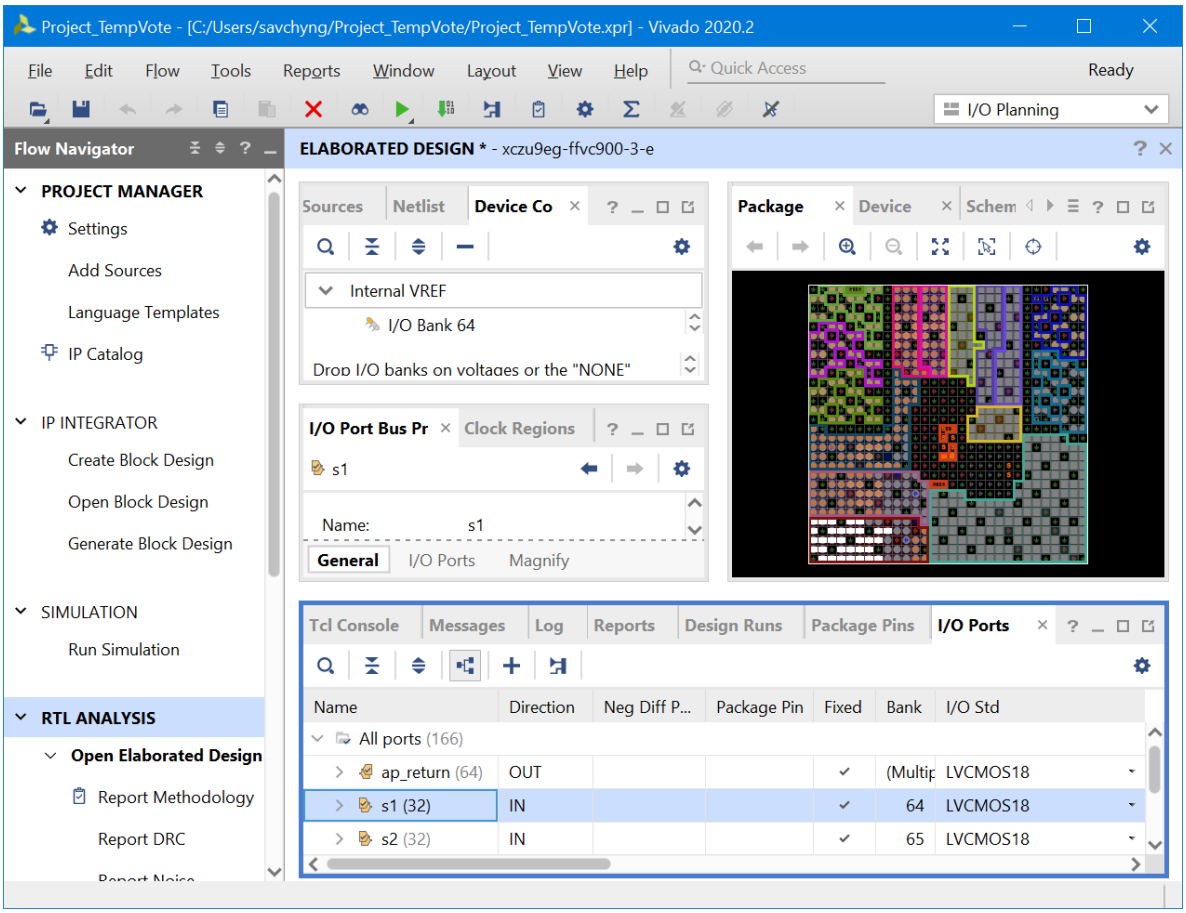


Abbildung 8: Zuweisung von Eingabe-/Ausgabe-Ports auf Pins in des Ziel-FPGA

4.4.3 Synthese in Vivado

Bei der Vivado-Synthese schreibt Vivado den Entwurf in eine Netzliste auf Chip-Ebene um, wobei die zuvor festgelegten zeitlichen Einschränkungen berücksichtigt werden. Die Vivado Design Suite bietet hierfür verschiedene Umsetzungsstrategien, die gegebenenfalls während der Entwicklung ergänzt werden können. Die Strategie zur Umsetzung des Entwurfs kann beispielsweise auf die Erhöhung der Leistung, oder auf Kosten der Leistung und der Flächennutzung zur Verbesserung des Zeitverhaltens abzielen.

Das Ergebnis der Vivado-Synthese wird erneut als Schaltplan ausgegeben und auf Zeitverhalten und Ressourcennutzung analysiert. Dabei wird auch ermittelt, wie viel Energie der Entwurf nach der Umsetzung voraussichtlich verbraucht. Damit wird verifiziert, ob der Entwurf nach seiner Umsetzung die Spezifikation hinsichtlich des Energieverbrauch einhält. Darüber hinaus wird die Simulation für weitere Verifizierungsschritte genutzt. In der sogenannten „funktionalen Simulation“ wird die funktionale Korrektheit des Entwurfes verifiziert, um sicherzustellen, dass sich das Verhalten des Entwurfes bei der Vivado-Synthese nicht verändert hat. Für die Entwürfe in HDL erfolgt zudem eine zeitliche Simulation, welche die von der Vivado Design Suite berechneten zeitlichen Verzögerungen des Entwurfes miteinbezieht. Damit wird der Entwurf erneut auf die Erfüllung von zeitlichen Anforderungen geprüft.

4.4.4 Platzierung

Bei der Platzierung wird der Entwurf unter Verwendung der vorher definierten physischen Einschränkungen (siehe Kapitel 4.4.2) optimiert und auf den Baustein platziert, ohne diesen tatsächlich auf einer Hardware umzusetzen. Innerhalb der Vivado Design Suite sind die zu platzierenden Bausteine vordefiniert.

Für die Platzierung bietet die Vivado Design Suite unterschiedliche Strategien. Ziel der Platzierung ist es, dass der platzierte Entwurf der Modulspezifikation entspricht. Es gibt beispielsweise Strategien, bei denen der Entwurf hinsichtlich der Leistung optimiert wird, oder bei denen der Entwurf so auf dem FPGA verteilt wird, dass eine Überlastung bestimmter FPGA-Regionen verhindert wird. Es können auch eigens erstellte Strategien angewendet werden. Zur Verifizierung der Platzierung wird anschließend das Zeitverhalten, die Ressourcennutzung und die Energienutzung analysiert. Die Ergebnisse dieser Analyse werden von der vorher gewählten Strategie beeinflusst.

Zur manuellen Verifizierung der Sinnhaftigkeit der Platzierung wird eine von der Vivado Design Suite erzeugte Grafik verwendet, die anzeigt, wie der Entwurf auf dem FPGA platziert wurde.

Wie nach der Vivado-Synthese (siehe Kapitel 4.4.3) lässt sich nach der Implementierung die funktionale und (bei Verwendung von HDL) zeitliche Simulation durchführen, um die funktionalen und zeitlichen Anforderungen des platzierten Entwurfs zu verifizieren. Anders als nach der Vivado-Synthese wird bei dieser zeitlichen Simulation die Verzögerung nicht geschätzt, sondern anhand der physischen Eigenschaften der verwendeten Ressourcen des Chips berechnet. In diesem Entwicklungsschritt lassen sich die zeitlichen Anforderungen strikter verifizieren.

4.4.5 EDIF

EDIF ist ein standardisiertes Format für Netzlisten und Schaltpläne. Dieses einheitliche Entwurfsformat dient der Kompatibilität verschiedenster Werkzeuge, die bei der Entwicklung von FPGA-Designs zur Anwendung kommen. Das EDIF-Format wird ebenfalls von der Vivado Design Suite unterstützt. In Vivado können EDIF-Dateien als Quellen für Entwürfe verwendet werden und vorhandene Entwürfe können mittels Vivado als EDIF-Dateien exportiert werden. Das Exportieren als EDIF-Datei kann alternativ auch zwischen den beschriebenen Arbeitsschritten ausgeführt werden.

Für dieses Vorhaben wird der Entwurf nach der Implementierung als EDIF-Netzliste exportiert. Die so erstellte EDIF-Netzliste kann dann mithilfe eines im Vorläufervorhaben entwickelten Werkzeugs auf dessen Komplexität analysiert werden [9].

4.5 Realisierung der FPGA-Designs

4.5.1 Eingangsprüfung

Bei diesem FPGA-Design soll ein einzelnes Messsignal von einem Sensor erfasst und auf Plausibilität überprüft werden. In der gewählten Realisierung muss der repräsentative Signalwert innerhalb eines vordefinierten Intervalls (in diesem Beispiel von größer/gleich 50 und unter 100) liegen. Ist der Wert innerhalb des Intervalls, wird das Messsignal ausgegeben, um von anderen FPGA-Designs weiterverarbeitet zu werden. Ist der Signalwert hingegen außerhalb des Intervalls (in diesem Beispiel etwa kleiner als 50 oder größer/gleich 100), wird ein Fehlersignal (Wert 0) ausgegeben. Dieses Fehlersignal wird auch dann ausgegeben, falls das Design kein Messsignal erhält (z.B. Kabelbruch). Abbildung 9 zeigt einen Auszug des C++ Codes des FPGA-Designs zur Eingangsprüfung innerhalb von Vitis HLS.

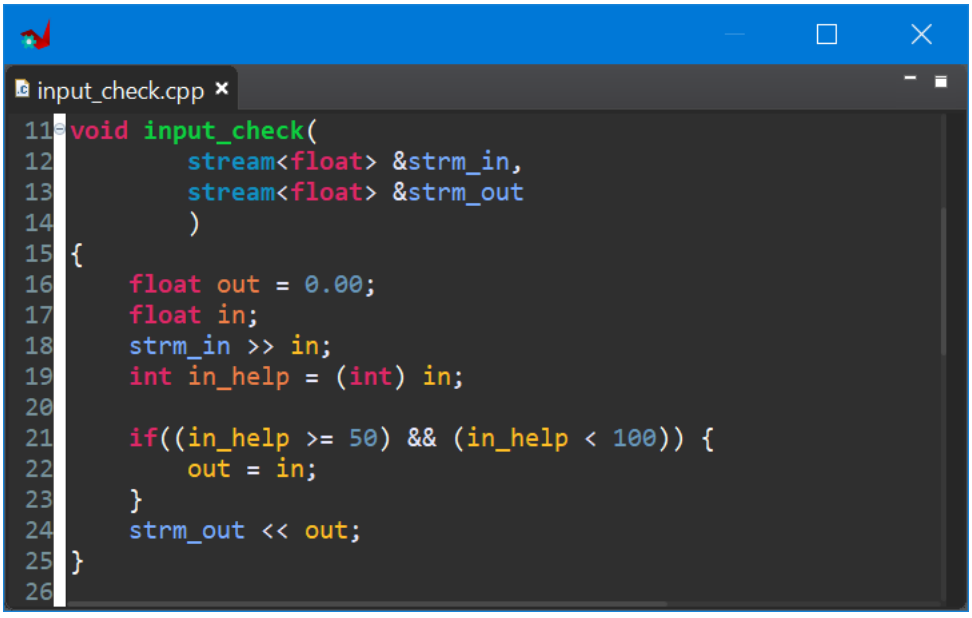


Abbildung 9: C++ Codeabschnitt der Eingangsprüfung (Vitis HLS 2021.1)

In Abbildung 10 ist der Schaltplan des FPGA-Designs zur Eingangsprüfung zu sehen, wie er sich in der Vivado Design Suite anzeigen lässt. Dabei lassen sich die einzelnen Signale und die Verschaltung der Logikblöcke nachvollziehen.

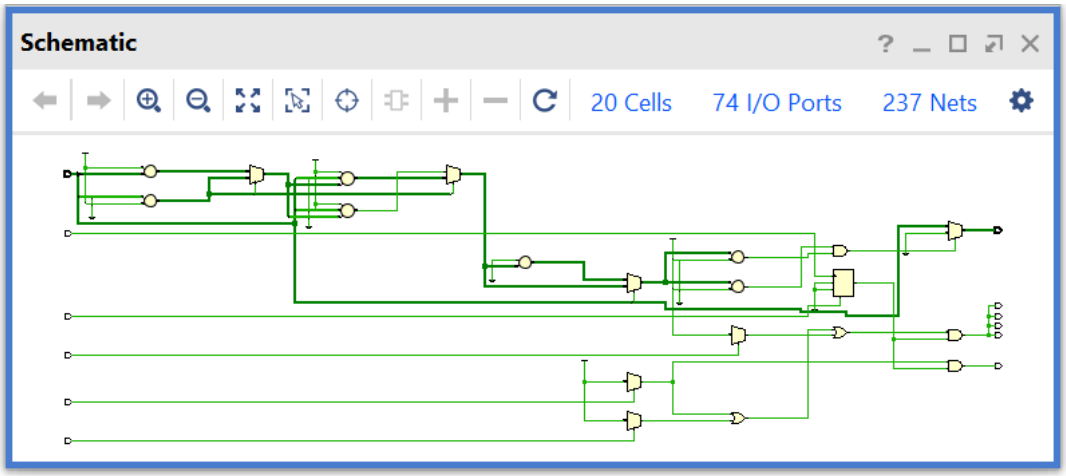


Abbildung 10: Schaltplan des FPGA-Designs zur Eingangsprüfung (Vivado 2021.1)

Abbildung 11 zeigt an, wie die Implementierung der Eingangsprüfung auf eine ausgewählte FPGA-Architektur aussehen würde. Hierbei wird vom Entwickler eine FPGA-Hardwarearchitektur aus einer vom Tool vordefinierten Liste verfügbarer FPGA-Chips des Herstellers ausgewählt. Dabei lässt sich z.B. die Zuweisung von Pins in der endgültigen Hardwarerealisierung prüfen und in welcher Weise das FPGA-Design auf einen gewählten Chip platziert wird.

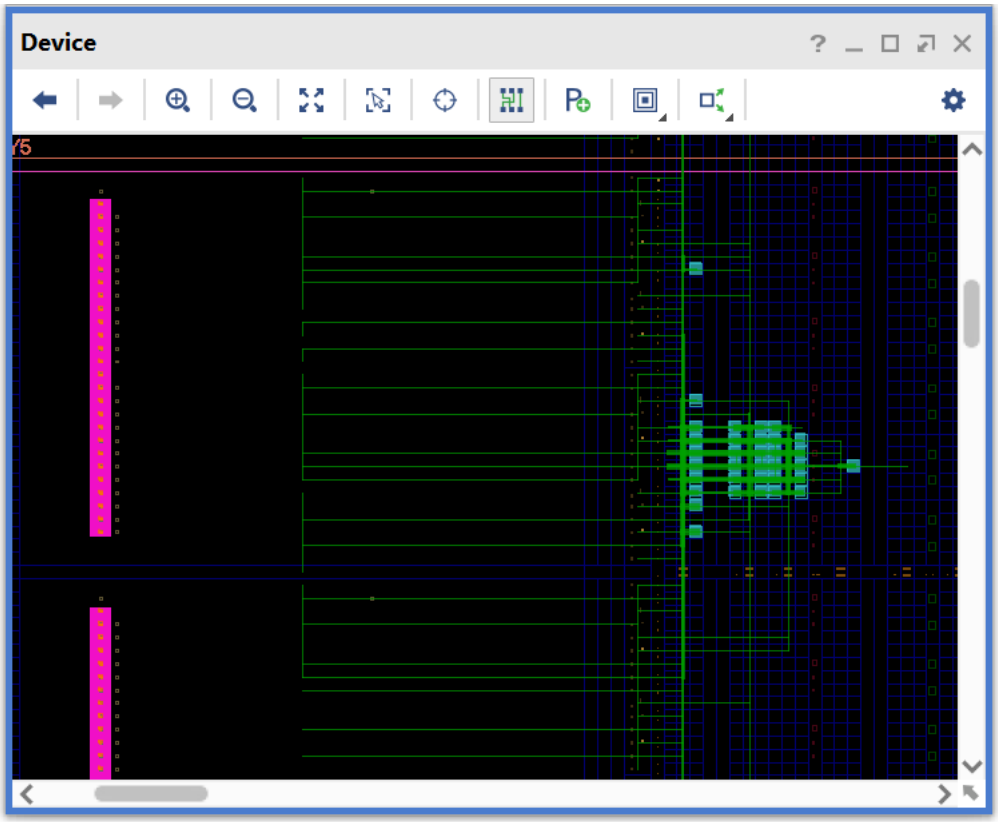


Abbildung 11: FPGA-Architektur der Eingangsprüfung (Vivado 2021.1)

4.5.2 2-von-3-Auswahlschaltung

Dieses FPGA-Design liest drei Signale (z.B. von drei Temperatursensoren) ein und überprüft, ob sie einen vorgegebenen Grenzwert überschreiten. Liegt eine Grenzwertüberschreitung vor, so gibt das Design ein Warnsignal aus. Liegen zwei oder mehr Grenzwertüberschreitungen vor, so gibt es zusätzlich ein Fehlersignal aus (kann z.B. für das Auslösen weiterer Maßnahmen wie einer Reaktorschnellabschaltung verwendet werden). Das Design wurde so umgesetzt, dass es bei einem fehlenden Signal eine 1-von-2-Auswahl mit den restlichen beiden Signalen ausführt. Fehlen zwei oder alle Signale, wird ein Fehlersignal ausgegeben. Abbildung 12 zeigt einen C++ Codeabschnitt des FPGA-Designs der 2-von-3-Auswahlschaltung innerhalb von Vitis HLS.

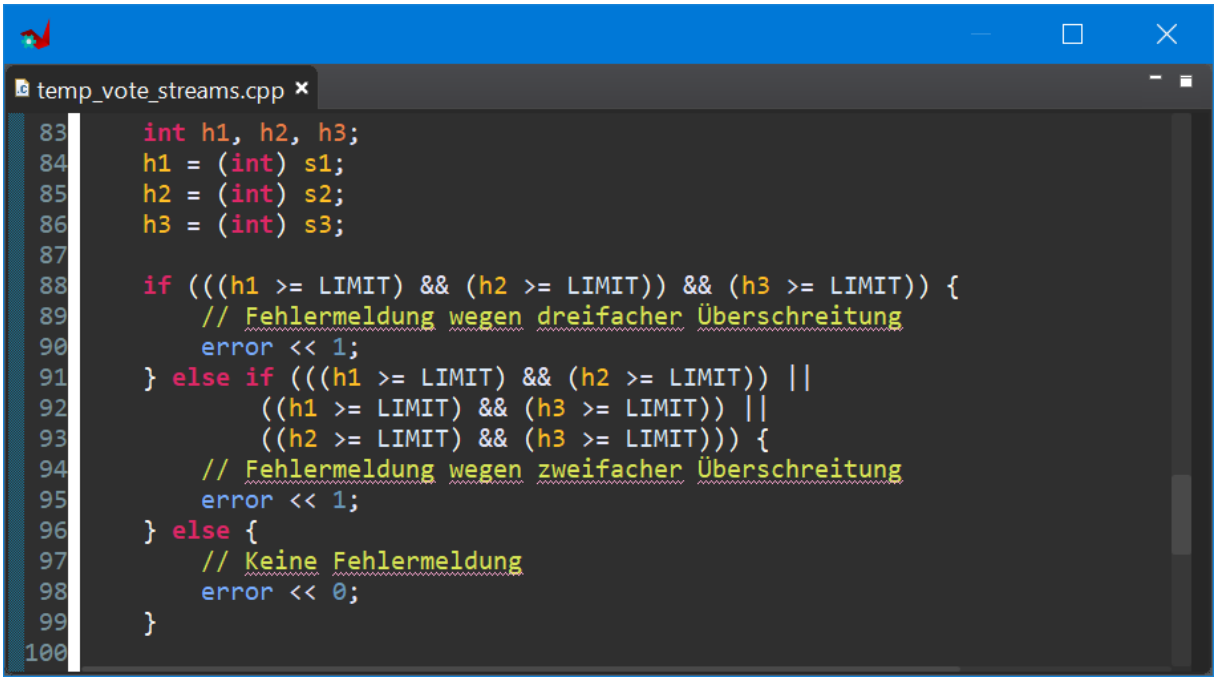


Abbildung 12: C++ Codeabschnitt der 2-von-3-Auswahlschaltung (Vitis HLS 2021.1)

In Abbildung 13 ist der Schaltplan des FPGA-Designs der 2-von-3-Auswahlschaltung zu sehen, wie er sich in der Vivado Design Suite anzeigen lässt, um Signale und Verschaltung der Logikblöcke nachzuvollziehen. In der Schaltplanansicht lassen sich gezielt Funktionen des FPGA-Designs anzeigen und nachvollziehen.

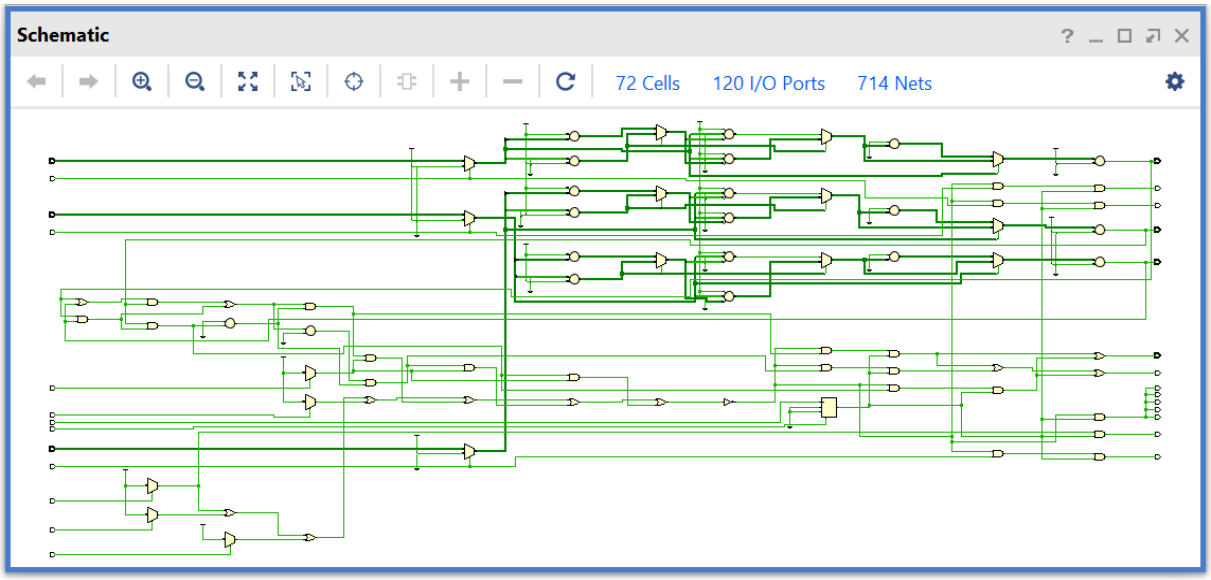


Abbildung 13: Schaltplan der 2-von-3-Auswahlschaltung (Vivado 2021.1)

Abbildung 14 zeigt einen vereinfachten Schaltplan, bei dem nur die Messsignale und Warn- bzw. Fehlermeldungen als Ein- und Ausgaben beachtet werden.

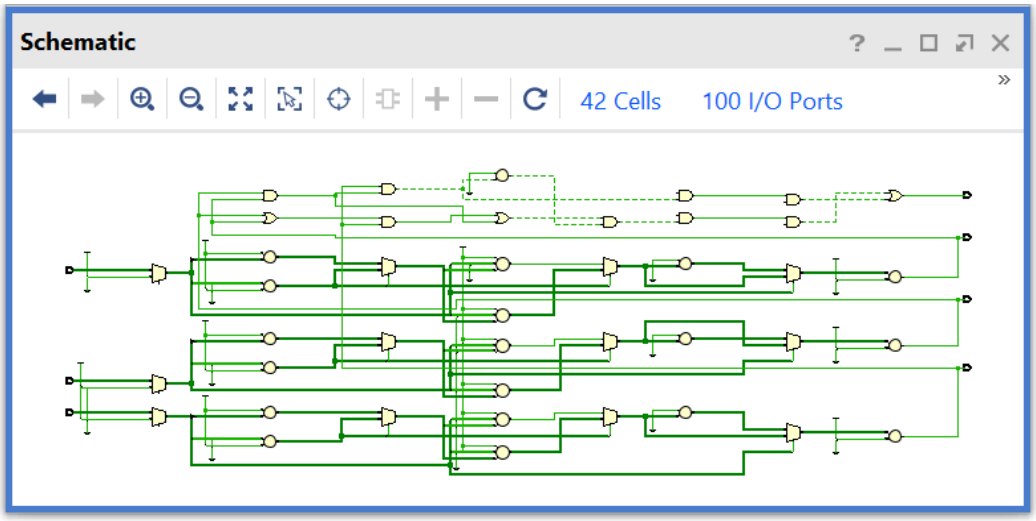


Abbildung 14: Vereinfachter Schaltplan der 2-von-3-Auswahlschaltung (Vivado 2021.1)

4.5.3 Redundanzschaltung

Bei diesem FPGA-Design werden drei verschiedene Messgrößen (z.B. Temperatur, Druck und Neutronenfluss) in eine 2-von-3-Auswahlschaltung mit vorgeschalteter Eingangsprüfung eingelesen. Anschließend wird auf die Ausgaben der 2-von-3-Auswahlen eine weitere 2-von-3-Auswahl angewendet und deren Ergebnis ausgegeben. Wird bei mindestens zwei Messgrößen ein Fehler ausgegeben, so gibt die Redundanzschaltung ein Fehlersignal aus (das z.B. für weitere Maßnahmen wie eine Reaktorschnellabschaltung verwendet werden kann). Die Behandlung von ausfallenden Eingangssignalen erfolgt wie in den beiden vorher beschriebenen Designs.

Für weitere Diversifizierung wurden verschiedene Eingabetypen der Messgrößen ausgewählt. In unserem Beispiel werden Temperatur und Druck als Gleitkommazahlen (C++ Datentyp „float“) und der Neutronenfluss als ganze Zahl (C++ Datentyp „int“) angegeben.

Bei dem Design wird die Eingangsprüfung sechs Mal für die „float“-Eingaben und zweimal die 2-von-3-Auswahl für „float“-Messgrößen verwendet. An dieser Stelle sei erwähnt, dass die Bereiche im Design, welche mit „float“ arbeiten, komplexer (in Bezug auf Anzahl der Untermodule) sind als die, bei denen mit „int“ gearbeitet wird. Als Resultat realisiert Vitis HLS diese Bereiche als konkrete Untermodule. Der Grund dafür sind die zusätzlichen Rechenoperationen für das Umwandeln der Signale vom Datentyp „float“ zu „int“. Entsprechend müssen in späteren Arbeitsschritten die Bereiche zur Temperatur und zum Druck stärker getestet werden als der Neutronenfluss.

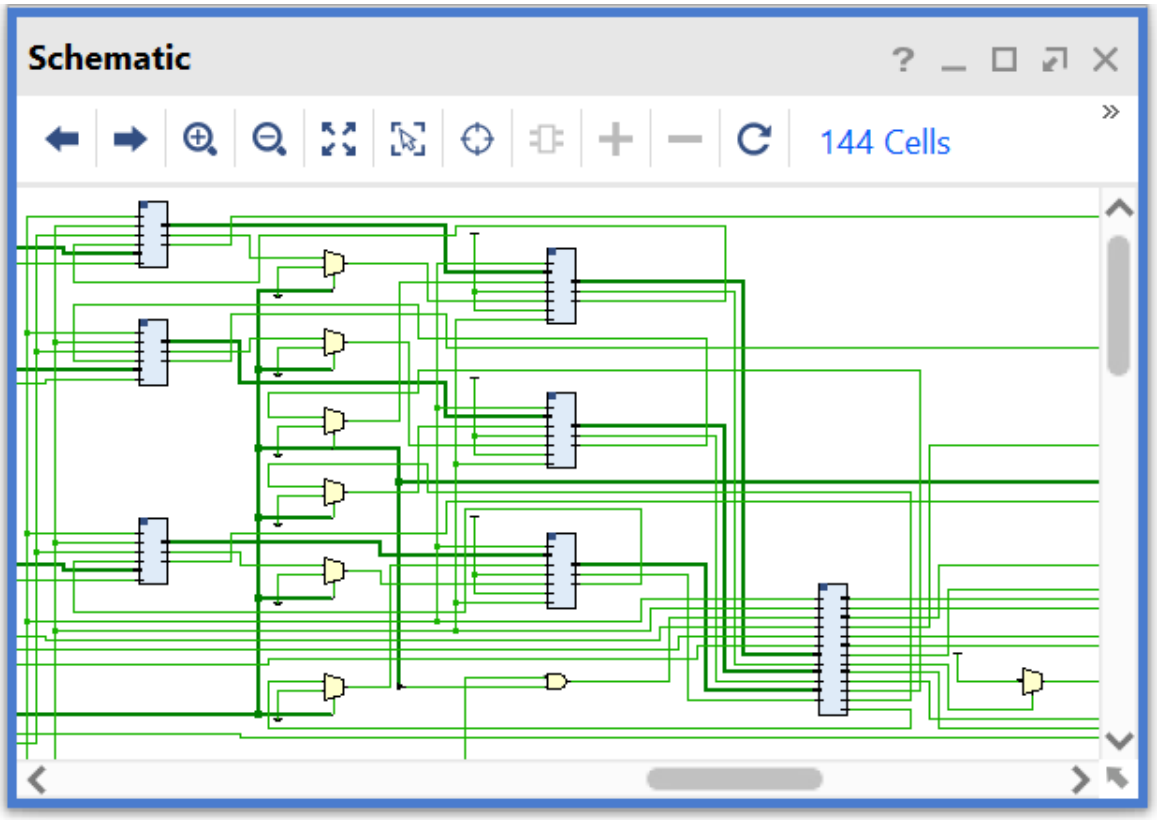


Abbildung 15: Schaltplanabschnitt der Redundanzschaltung (Vivado 2021.1)

In Abbildung 15 ist der Schaltplanausschnitt des FPGA-Designs der Redundanzschaltung zu sehen. Die Anzeige in der Vivado Design Suite ermöglicht es, die einzelnen Signale und die

Verschaltung der Logikblöcke nachzuvollziehen, die für die Verarbeitung von Drucksignalen verantwortlich sind.

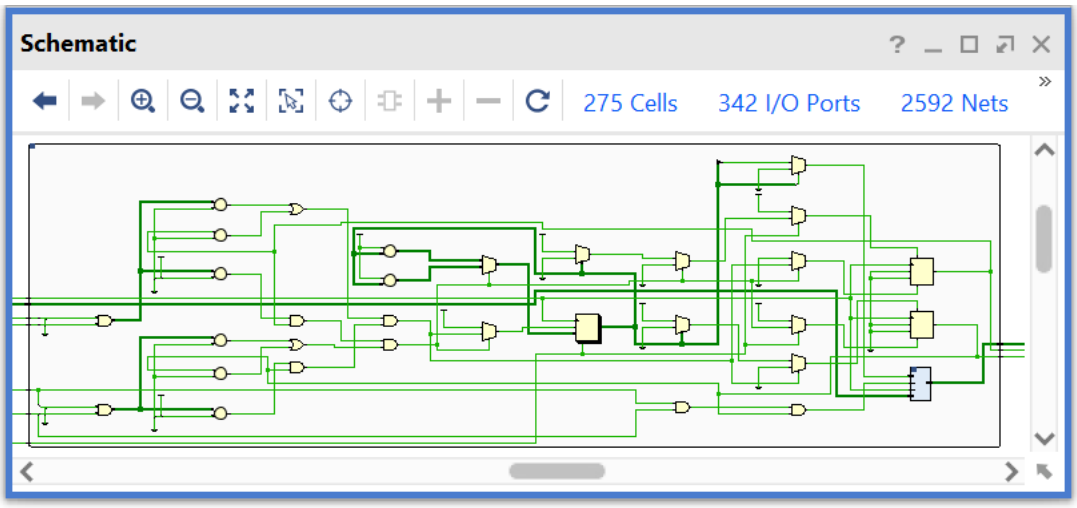


Abbildung 16: Eingangsprüfung der Redundanzschaltung (Vivado 2021.1)

In Abbildung 16 ist der Schaltplanabschnitt des FPGA-Designs der Redundanzschaltung zu sehen, der die vorgelagerte Eingangsprüfung der Anwendung repräsentiert. Die Signale laufen nach der Eingangsprüfung zu den 2-von-3-Auswahlelementen des Designs. Ein Schaltplanabschnitt des 2-von-3-Auswahlelements des FPGA-Designs der Redundanzschaltung ist in Abbildung 17 dargestellt.

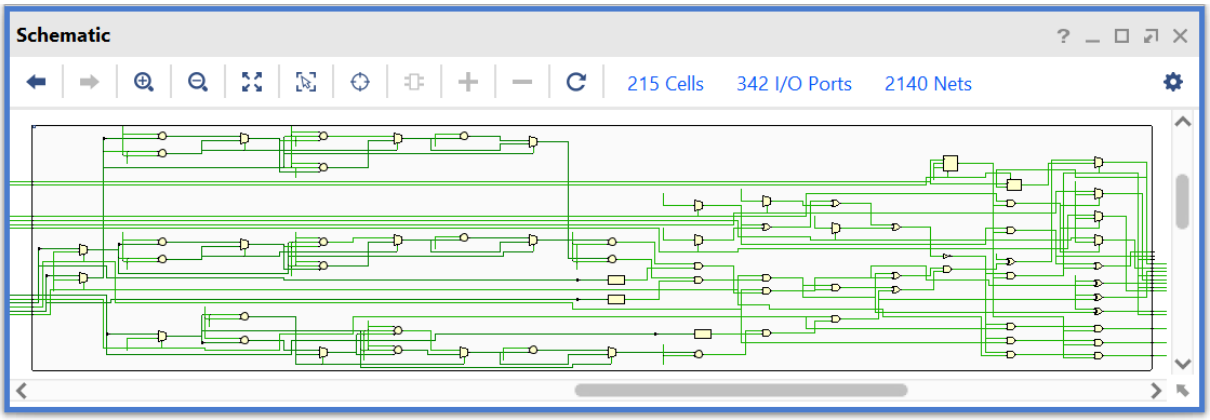


Abbildung 17: 2-von-3-Auswahlelement der Redundanzschaltung (Vivado 2021.1)

Abbildung 18 zeigt, wie das FPGA-Design der Redundanzschaltung nach Platzierung für eine ausgewählte FPGA-Architektur aussehen würde.

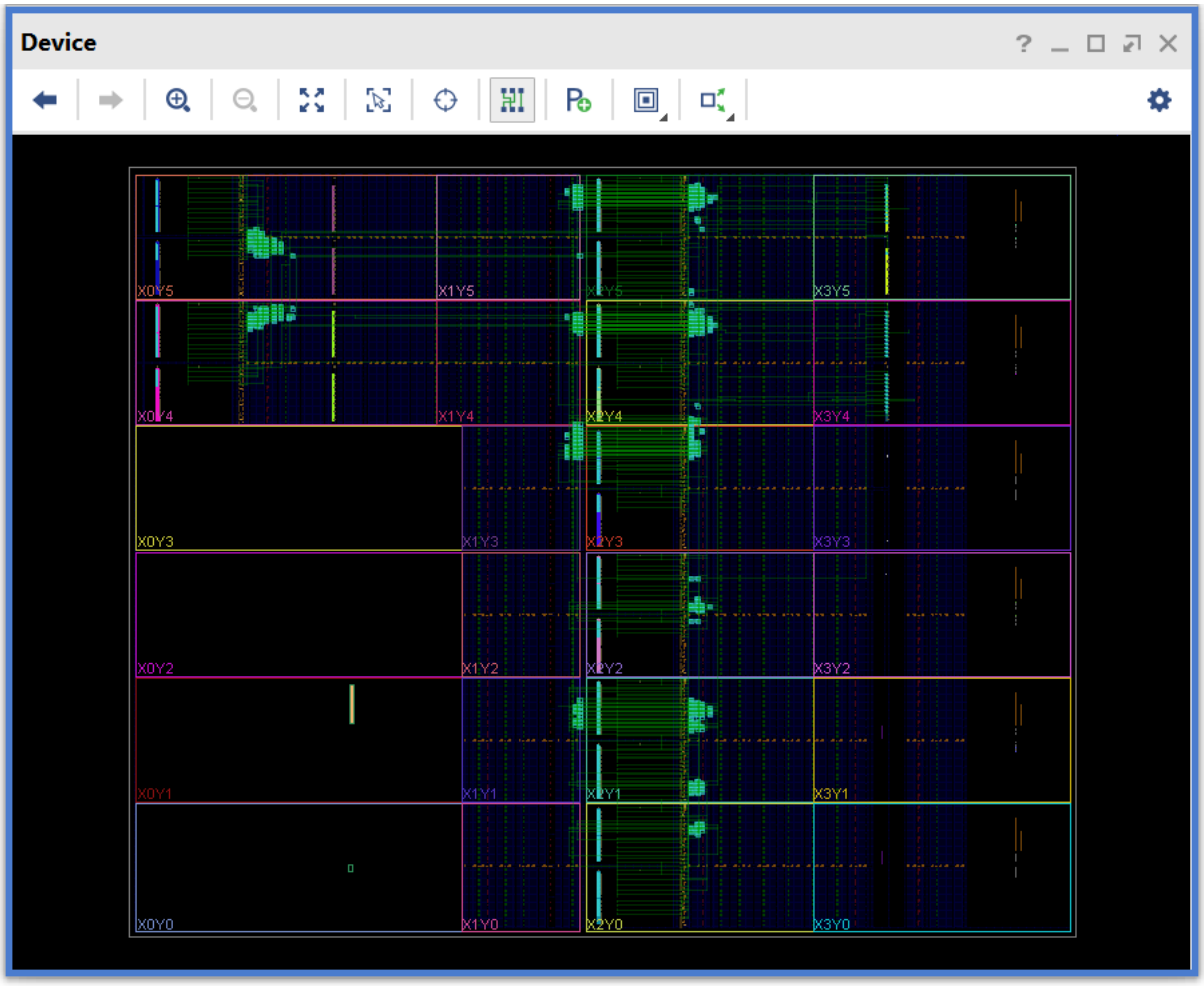


Abbildung 18: FPGA-Architektur der Redundanzschaltung (Vivado 2021.1)

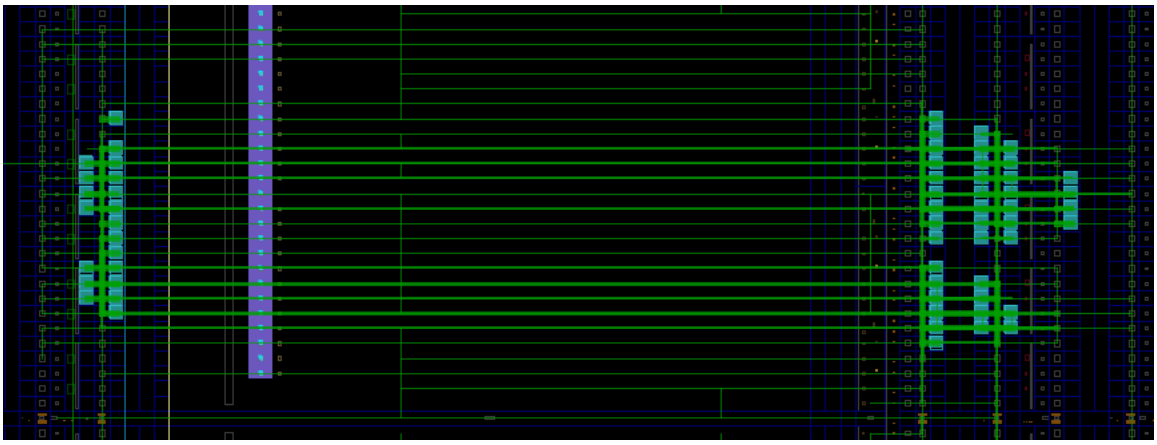


Abbildung 19: Ausschnitt der FPGA-Architektur der Redundanzschaltung (Vivado 2021.1)

Abbildung 19 zeigt eine vergrößerte Ansicht auf den mittleren unteren Abschnitt aus Abbildung 18. Das eingelesene Signal wird durch die Eingangsprüfung (linker Bildbereich) auf Korrektheit und im Anschluss auf Grenzwertüberschreitung (rechter Bildbereich) geprüft.

4.6 Erstellung einer geeigneten Testumgebung

Für die Entwicklung der FPGA-Designs wird innerhalb der Entwicklungswerkzeuge Vitis HLS und Vivado Design Suite bereits die dort vorhandene Testumgebung eingesetzt. Als Teil der Testumgebung der Entwicklungswerkzeuge werden für die einzelnen Entwicklungsschritte eigene Testbänke (in C++, RTL und VHDL) erstellt und für die entwicklungsbegleitenden Tests genutzt (siehe Kapitel 5.2).

Neben der Korrektheit der FPGA-Designs wird jedoch auch die Korrektheit der verwendeten Werkzeuge als wichtiger Bestandteil der Verifizierung angesehen. Häufig ist bei der FPGA-Entwicklung die Testmethode nicht unabhängig vom verwendeten Entwicklungswerkzeug. Die Werkzeuge für die FPGA-Entwicklung bieten in der Regel umfangreiche Verifizierungs- und Testmöglichkeiten, die jedoch nicht die im kerntechnischen Regelwerk geforderte Unabhängigkeit aufweisen. Werden die Tests mit den gleichen Werkzeugen wie für die Entwicklung durchgeführt, können keine verlässlichen Aussagen z.B. über die Korrektheit des verwendeten Synthesetools. Daher wurde eine Testumgebung geschaffen, um die FPGA-Designs unabhängig von den Entwicklungswerkzeugen zu testen. Im Folgenden sind die Testmethoden der Testumgebung beschrieben, welche auf zur Entwicklung diversitären Testwerkzeugen basieren. Die erstellte Testumgebung eignet sich zur Verifizierung des Entwicklungsprozesses, insbesondere der Korrektheit der Entwicklungswerkzeuge, sowie zum eigenständigen Test der funktionalen Korrektheit der FPGA-Designs.

4.6.1 Testplan

Die Planung und Durchführung des Testprogramms erfolgt unabhängig von der Entwicklung des FPGA-Designs. Die Unabhängigkeit ist personell (Tester, Entwickler) sowie durch eine räumliche Trennung der Standorte (Berlin, Hallbergmoos) gegeben. Die Grundlage der Tests bilden die internationalen Standards IEC 61508-3 Ed. 2 [47], IEC 62566 Ed. 1 [4] sowie IAEA NR-T-3.31 [20].

In Abschnitt 3.3 wurde das prinzipielle Vorgehen der Verifizierung sowie ausgewählte Verifizierungsmethoden vorgestellt. Diese werden bei der Testplanung und Durchführung berücksichtigt. Das Testprogramm setzt sich aus den folgenden Testschritten zusammen:

- Unabhängige Verifizierung der Spezifikation und Review des Quellcodes;
- Dynamische Quellcodeanalyse;
- Unabhängiger Test mit diversitärem Rechner und Betriebssystem;
- Unabhängiges Erstellen und Vergleichen der Wellenfunktion;
- Testen des Quellcodes mit einem diversitären (Intel HLS) Werkzeug;
- Testen des FPGA-Designs mit diversitärem Synthese-Werkzeug (Quartus Prime)
- Testen des FPGA-Designs mit diversitärem Testwerkzeug (ModelSim).

Die Tests mit der eigens entwickelten Testumgebung werden am Beispiel des FPGA-Designs zur 2-von-3-Auswahlschaltung dargestellt. Es handelt sich um eine exemplarische Leittechnikanwendung, bei der drei Sensoren kontinuierlich einen Eingangswert überwachen (z.B. Kühlmitteltemperatur am Reaktordruckgefäß). Wenn mindestens zwei der Sensoren einen erhöhten Wert, d.h. eine Grenzwertüberschreitung, (z.B. im Kühlmittel eines

Primärkreislaufs) messen, wird ein Warnsignal ausgegeben. In dem exemplarischen Anwendungsfall soll zusätzlich eine Schnellabschaltung eingeleitet werden. Sollte lediglich einer der drei Sensoren einen Wert über dem Grenzwert registrieren, wird ein Warnsignal ausgegeben, aber keine Schnellabschaltung eingeleitet. Das FPGA-Design ist in C++ implementiert. Der C++ Code wird mit Hilfe des Vitis HLS Compiler in Verilog und VHDL übersetzt. Für die Testbanksimulation wird der generierte Verilog-Code genutzt.

4.6.2 Dynamische Codeanalyse

Durch die Verwendung des Prüfwerkzeugs ModelSim soll eine unabhängige Verifizierung des in Vitis HLS erstellten Codes erfolgen. Die mittels ModelSim durchgeführten Testschritte umfassen die strukturabhängigen Tests zur Anweisungsabdeckung, Zweigabdeckung und MC/DC Abdeckung („Modified Condition/Decision Coverage“).

Für die genannten Tests wird die Struktur des Codes als ein Kontrollflussgraph dargestellt. Dieser Graph wird durch Knoten und Zweige definiert. Ein Knoten stellt eine Funktion (Ausführanweisungen) dar, während ein Zweig die Flussrichtung des Programms visualisiert. Der Test der Anweisungsabdeckung prüft die Abdeckung aller ausführbaren Anweisungen eines Kontrollflussgraphen. Ziel des Zweigabdeckungstests ist es sämtliche Zweige des Kontrollflussgraphen theoretisch zu durchlaufen bzw. alle nicht durchlaufenen Zweige aufzudecken. Der MC/DC Test überprüft die Ausführung aller Zweige durch Kombinationen von Bedingungen und Entscheidungen. Somit überprüft dieser Test, vergleichbar zum Zweigabdeckungstest, die Logik des Programms durch eine tabellarische Darstellungsform. Alle drei Tests können für die Verifizierung des Quellcodes genutzt werden.

Die Abdeckungstests unterliegen folgender Hierarchie. Die Prüfung der Anweisungsabdeckung stellt eine Untermenge der Zweigabdeckung dar, weshalb die Zweigabdeckung im Vergleich als höherwertig angesehen wird. Der MC/DC Abdeckungstest stellt gemäß IEC 61508-3 [47] die umfangreichste dynamische Analyse dar und deckt beide vorher genannten Abdeckungstests implizit mit ab. Während die dynamische Codeanalyse im Zuge der Modultests („unit test“) von der IEC 61508-3 [47] gefordert wird und für höhere Safety Integrity Level jeweils ein höherwertiger Abdeckungstest empfohlen wird, gibt es im kerntechnischen Regelwerk keine explizite Vorgabe zu der Art der dynamischen Analysen bzw. Abdeckungstests.

Anweisungsabdeckungstest

Der Anweisungsabdeckungstest prüft die Abdeckung aller ausführbaren Anweisungen des Kontrollflussgraphen eines Programms auf Vollständigkeit. Dabei werden alle Knoten durchlaufen, sofern die dafür erforderlichen Bedingungen erfüllt sind.

Die Effektivität dieser Verifizierungsmethode kann bei der Durchführung der Anweisungsabdeckung für das FPGA-Design wie folgt veranschaulicht werden: Im ersten Ergebnis wurde festgestellt, dass die Erfassung der Grenzwertüberschreitung aller drei Sensoren nicht berücksichtigt wurde. Diese Abweichung wurde entsprechend durch den Entwickler korrigiert. Die erneute Verifizierung des modifizierten Programms war befundfrei. Die Durchführung dieses Testschritts hatte somit unmittelbar eine Auswirkung auf die Korrektheit der Entwicklung. In einer realen Entwicklung können mithilfe der

strukturabhängigen Tests in einer frühen Phase des Sicherheitslebenszyklus systematische Fehler minimiert und Korrekturen effizient in den Entwicklungsablauf eingebracht werden.

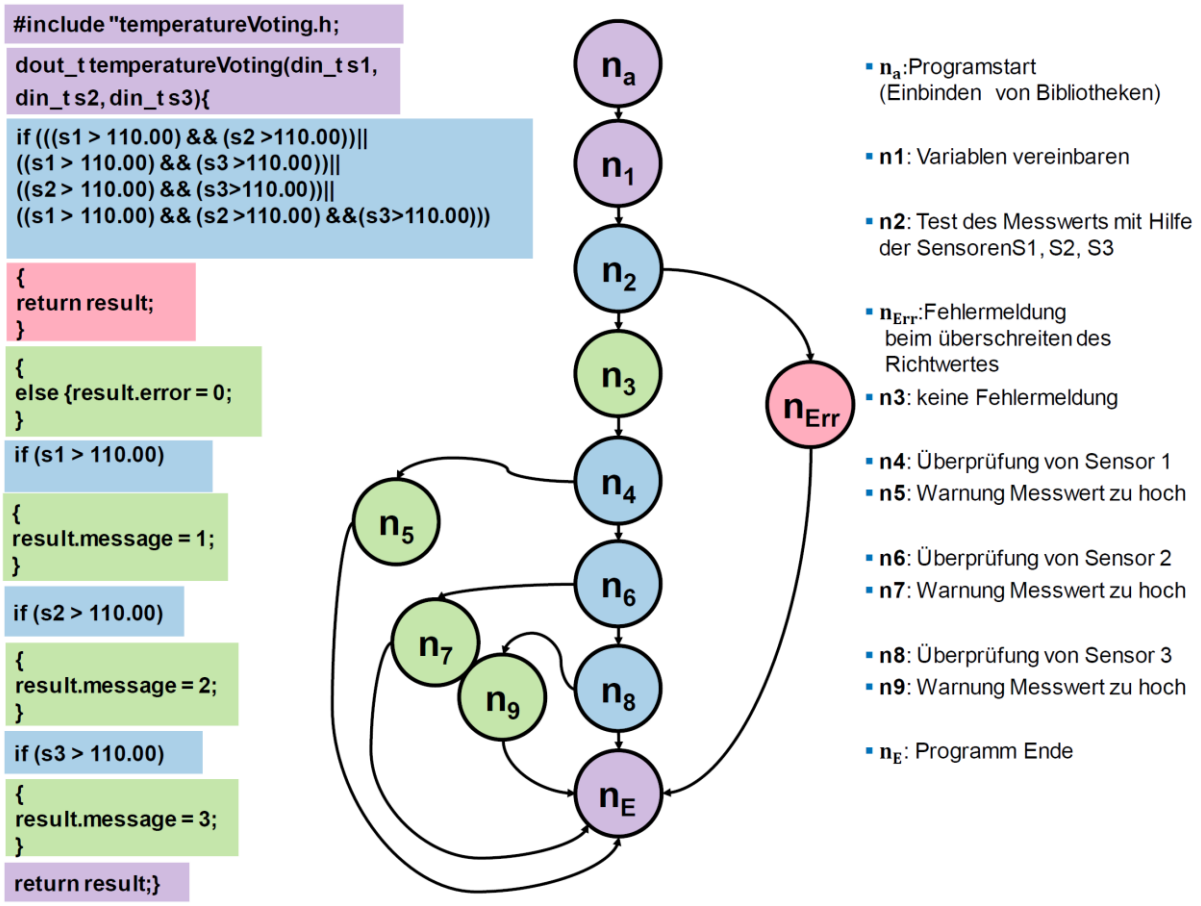


Abbildung 20: Kontrollflussgraph des Programms der Auswahlschaltung.

Zweigabdeckungstest

Der Wechsel der Abstraktionsebene von einem Quellcode in eine strukturierte Darstellungsform (z.B. Kontrollflussgraph oder Struktogramm) fördert die Betrachtung des Codes, wodurch sich Anomalien und Abweichungen leichter erkennen lassen. In Abbildung 20 wird die Struktur des Codes als Kontrollflussgraph dargestellt. Anhand des Kontrollflussgraphen lässt sich die Funktionsweise folgendermaßen veranschaulichen:

- Im Knoten n_2 sind die Sensoren S_1, S_2, S_3 mittels der 2-von-3-Auswahl verknüpft. Wird von zweien der vorgegebene Grenzwert überschritten, wird ein Auslösesignal erzeugt. Anschließend wird das Programm beendet. Andernfalls läuft das Programm weiter.
- Im Knoten n4 wird geprüft, ob Sensor S_1 den Grenzwert anzeigt. Sollte dies der Fall sein wird das Programm beendet, andernfalls durchläuft das Programm die nächste Funktion.
- Im Knoten n_6 und n_8 sind Funktionen programmiert, welche die Werte von den Sensoren S_2 und S_3, entsprechend der Funktionsweise von n_4, überprüfen.
- Es liegt ein Fehler vor: Der Fall, dass alle drei Sensoren über dem Grenzwert liegen, ist im Knoten n_2 zunächst nicht bedacht worden. Dieser Fehler wurde in einer nachfolgenden Version entsprechend korrigiert. Es sei angemerkt, dass dieser Fehler nicht systematisch durch den Zweigabdeckungstest, sondern durch eine manuelle

Prüfung der Verzweigungen aufgedeckt wurde. Der Test auf Bedingungsüberdeckung (MC/DC-Test) würde diesen „Fehler“ systematisch aufdecken.

Modified Condition/Decision Coverage (MC/DC)

Beim MC/DC-Test wird der Code systematisch (hier tabellarisch) durch die Ausführung aller Zweige mittels Kombinationen von Bedingungen und Entscheidungen (siehe Tabelle 6) überprüft. Vergleichbar zum Zweigabdeckungstest wird somit unter Zuhilfenahme einer Entscheidungsmatrix (tabellarische Darstellungsform) die Logik des Programms überprüft.

Tabelle 6: Beispiel des MC/DC Test

Nr.	$S_1 \geq 110$ a.u.	$S_2 \geq 110$ a.u.	$S_3 \geq 110$ a.u.	$(S_1 \cap S_2 \geq 110) \cup (S_2 \cap S_3 \geq 110) \cup (S_1 \cap S_3 \geq 110)$
1.	erfüllt	nicht erfüllt	nicht erfüllt	nicht erfüllt
2.	nicht erfüllt	erfüllt	nicht erfüllt	nicht erfüllt
3.	nicht erfüllt	nicht erfüllt	erfüllt	nicht erfüllt
4.	erfüllt	erfüllt	nicht erfüllt	erfüllt
5.	nicht erfüllt	erfüllt	erfüllt	erfüllt
6.	erfüllt	nicht erfüllt	erfüllt	erfüllt
7.	nicht erfüllt	nicht erfüllt	nicht erfüllt	nicht erfüllt
8.	**erfüllt**	**erfüllt**	**erfüllt**	**fälschlicherweise unberücksichtigt**

Bei dieser Testmethode trat im ersten Ergebnis der gleiche Fehler auf wie beim Anweisungsabdeckungstest, was die Effektivität des MC/DC Tests aufzeigt (siehe Tabelle 6).

4.6.3 Bestimmung der zyklomatischen Komplexität des Codes

Um Fehler zu erkennen, die unbeabsichtigt und für den Entwickler nicht direkt ersichtlich beim Entwurf des Programms eingebaut werden, kann eine Analyse der zyklomatischen Komplexität herangezogen werden. Hierbei wird angenommen, dass die Fehlerdichte abhängig von der zyklomatischen Komplexität des Programms ist. Auf dieser Grundlage wird empfohlen, die zyklomatische Komplexität des Programms gering zu halten, um die Anzahl versteckter Fehler zu minimieren. Um dies zu erreichen, wird das Programm in der Entwicklung in mehrere Unterprogramme (Modularisierung) aufgeteilt und auf den verschiedenen Bereichen eines FPGA oder verschiedenen FPGA-Bausteinen platziert. Dadurch soll das

Fehlerrisiko minimiert werden, dass bei der Erstellung eines (vergleichsweise) großen und damit komplexen Programms besteht, und dessen Korrektheit schwer zu überprüfen ist.

Die Komplexität des C++ Codes wird bei dieser Methode durch die Ermittlung der zyklomatischen Komplexität bestimmt. Dabei wird mithilfe des Durchflussgraphen, der Ermittlung der Zweiganzahl (n_{Zweige}) und Knotenanzahl (n_{Knoten}) der Wert der zyklomatischen Komplexität (Z_K) am Beispiel der Auswahlschaltung ermittelt:

$$Z_K = n_{Zweige} - n_{Knoten} + 2 = 5.$$

Gemäß IEC 61508 [47], dem Standard zur funktionalen Sicherheit, wird eine zyklomatische Komplexität des Programms $Z_K \leq 10$ empfohlen, um das Fehlerrisiko gering zu halten. Für Werte im Bereich $10 < Z_K \leq 20$ ist ein moderates Fehlerrisiko zu erwarten. Bei einer zyklomatischen Komplexität $Z_K > 20$ wird empfohlen, den Quellcode in kleinere Module zu unterteilen, da ein erhöhtes Fehlerrisiko vermutet wird.

Bei dem Beispiel der Auswahlschaltung liegt der ermittelte Wert der zyklomatischen Komplexität des C++ Quellcodes bei Z_K = 5.

4.6.4 Diversitätstest

Die Abhängigkeit der Ergebnisse des Vitis HLS Compilers vom verwendeten Rechner und Betriebssystem soll ermittelt werden. Der C++ Quellcode des Programms der Auswahlschaltung wird mit dem Vitis HLS Compiler in Verilog übersetzt. Als primärer Testschritt wird die Übersetzung des Quellcodes auf einem (von der Entwicklung) unabhängigen Rechner durchgeführt. Die Übersetzung erfolgt in erster Instanz auf verschiedenen Rechnern. Beide Rechner sind mit dem Betriebssystem Windows 10 ausgestattet. Das Ergebnis dieses Testschritts zeigt, dass in beiden Fällen das gleiche Übersetzungsresultat vom Compiler erarbeitet wurde.

Im nächsten Testschritt wird die Übersetzungsabhängigkeit des Compilers vom Betriebssystem geprüft. Hierfür wird der Vitis HLS Compiler auf einen Windows 7 Rechner installiert und der Quellcode der Auswahlschaltung in Verilog übersetzt. Beim Test der Übersetzungsabhängigkeit des Compilers wurden keine Abweichungen festgestellt.

Das Testergebnis legt nahe, dass der Vitis HLS Compiler unabhängig vom verwendeten Windows Betriebssystem und von den verwendeten Rechnern die gleichen Ergebnisse lieferte.

4.6.5 Erstellung und Vergleich der Wellenfunktionen auf verschiedenen Betriebssystemen

Um die Korrektheit des Programms zu überprüfen, wurde mit Hilfe der Vivado Tool Suite eine Testbanksimulation durchgeführt. In dieser Simulation wurden für das Beispiel der Auswahlschaltung für alle drei Sensoren Testwerte von einer (*.dat) Datei eingelesen. Die vom Testprogramm erstellten Wellenfunktionen wurden anschließend miteinander verglichen. Die Testergebnisse zeigen, dass die zeitlichen Verläufe und Formen (Rechteckfunktion) der Wellenfunktionen für alle drei Sensoren übereinstimmen. Auf einem weiteren Rechner (mit

einem anderen Windows-basierten Betriebssystem) wurden zur ersten Testdurchführung identische Wellenfunktionen ermittelt.

Das Testergebnis zeigt, dass die Wellenfunktionen der FPGA-Designs auf unterschiedlichen (Windows-basierten) Betriebssystemen ohne erforderliche Anpassungen genutzt werden können.

4.6.6 Tests mit einem diversitären Compiler

Mithilfe von Intel Altera HLS, eines zur Entwicklung diversitären Compilers, wird die Korrektheit der Funktion des Vitis HLS Compilers überprüft. Bei der Testdurchführung hat sich zunächst herausgestellt, dass die Syntax des Programms angepasst werden musste, da der Intel Altera HLS Compiler den für Vitis HLS erstellten Quellcode nicht fehlerfrei übersetzen konnte. Die Ursache hierfür liegt in der Nutzung verschiedener C++ Programmiersprachversionen der beiden Compiler. Vitis HLS ist im Gegensatz zu Intel Altera HLS in der Lage, auch ältere Sprachversionen zu verarbeiten. Das Vermischen von verschiedenen Sprachversionen führte bei der Testdurchführung zunächst zu Problemen, da die Syntax von beiden Compilern unterschiedlich interpretiert wurde. Durch die Entwicklung eines einheitlichen Quellcodes, der sowohl für Vitis HLS als auch Intel Altera HLS genutzt werden kann, wurde die Übersetzungsfähigkeit unabhängig vom verwendeten Compiler getestet.

In beiden Testfällen wurde das FPGA-Design als Ergebnis erfolgreich auf einem FPGA-Baustein platziert.

4.6.7 Diversitäres Synthese-Werkzeug

Um die Korrektheit der Arbeitsschritte innerhalb der Vivado Tool Suite (Xilinx) zu testen, wurde das dazu diversitäre Programm Quartus Prime (Intel Altera) verwendet. Bei diesem Testschritt wurde der Cyclone 5-Baustein verwendet. Der synthetisierte Verilog-Code wurde bei der Testdurchführung erfolgreich von Quartus Prime kompiliert und analysiert. Die anschließende Platzierung wurde ohne Auffälligkeiten durchgeführt. Die Vorrausetzung hierfür war die Synthese des C++ Codes mit dem Intel Altera HLS-Compiler (siehe Kapitel 4.3.2). Die Platzierung ermöglicht die Schaltdarstellung als Diagramm in Quartus Prime. Eine Schaltdarstellung als Diagramm in Vivado setzt voraus, dass im Vorfeld der eigens von Xilinx entwickelte HLS benutzt wurde.

Ein Vergleich der beiden diversitär generierten Schaltdarstellungen ist aufgrund ihrer sehr unterschiedlichen Struktur nicht zielführend. Als Testergebnis lässt sich jedoch festhalten, dass beide Programme trotz der Anwendung zweier unterschiedlichen Synthese-Werkzeuge funktional identisch sind. Die funktionale Korrektheit des Programms kann somit über eine Synthese des C++ Codes sowohl in Vivado, als auch in Quartus Prime nach Integration in das jeweils zugehörige Synthese-Werkzeug erzielt werden. Die Verwendung eines HLS-Compilers mit einem nicht zugehörigen Synthese-Werkzeug hingegen führt zu gravierenden Fehlern beim Kompilieren, Analysieren und dem Platzieren. Die Testergebnisse legen den Schluss nahe, dass die beiden Hersteller (Xilinx und Altera) unterschiedliche Algorithmen verwenden, die bei korrekter Verwendung der Entwicklungswerkzeuge zu gleichen Ergebnissen führen.

4.6.8 Diversitärer Testbanksimulator (Altera ModelSim)

Um die C++ basierte Testbanksimulation der FPGA-Designs auf Korrektheit zu prüfen, wurde der zur Vivado Toolsuite diversitäre Simulator ModelSim benutzt. Für die Durchführung des Tests ist die Synthese der C++ basierten Testbanksimulation mit dem jeweiligen HLS Compiler (Altera bzw. Xilinx) erforderlich. Es ist die Voraussetzung für eine störungsfreie und unkomplizierte Einbindung in den jeweiligen Testbanksimulator.

Die als Testergebnis generierten Wellenfunktionen weisen beim Anwenden der verschiedenen Testbanksimulatoren keine Auffälligkeiten auf.

5 ANALYSE DER FEHLERMODI MITTELS TESTUMGEBUNG

Die eigens erstellten FPGA-Designs sollen hinsichtlich des Fehlerpotentials und der hierfür relevanten Einflussfaktoren untersucht sowie zur Validierung von Testverfahren genutzt werden.

5.1 Mögliche Designschwächen bei der Entwicklung

Das Auftreten von Fehlern für ein FPGA-Design kann weder ausgeschlossen noch vollumfänglich vermieden werden. Fehler treten in der Regel dann auf, wenn Ereignisse oder Verknüpfungen von Ereignissen eintreten, die in der FPGA-Entwicklung nicht bedacht wurden. So können für ein FPGA-Design Fehler auftreten, wenn Schwächen im Design wirksam werden, die bei der Entwicklung im Vorfeld nicht bekannt waren oder entsprechend berücksichtigt wurden. Das Fehlerpotential lässt sich durch die Erkennung und Berücksichtigung von möglichen Designschwachstellen reduzieren. Designschwächen können beispielsweise durch Erfahrungswerte und durch Anwendung von Test- und Verifizierungsverfahren bei der Entwicklung von FPGA-Designs aufgedeckt werden.

Im Allgemeinen hängt die Identifikation potenzieller Designschwachstellen von der Erfahrung und dem Wissensstand der Entwickler sowie des Umfangs und der Qualität der angewendeten Test- und Verifizierungsmaßnahmen ab. Die beiden Kriterien, Herkunft der Designschwäche (menschlich, zufällig und betrieblich) sowie Auswirkung auf die Korrektheit des Programms (unkritisch, potenziell kritisch und kritisch) wurden im Rahmen des Forschungsvorhabens betrachtet, um potenzielle Designschwächen zu untersuchen. Nach der Identifikation einer Designschwäche wurde jeweils mindestens ein direkt zugeordnetes Testszenario (siehe Kapitel 5.2 für Testmethoden bzw. Kapitel 5.4 zur Fehlerinjektion) angewendet. Damit soll ermittelt werden, ob aus einer Designschwachstelle resultierende Fehler aufgedeckt werden können. Im Folgenden werden identifizierte potenzielle Designschwächen und daraus resultierende Fehler untersucht.

5.1.1 Fehler bei der Berechnung mit Konstanten und Variablen

Es besteht die Möglichkeit, dass benutzte Konstanten und Variablen falsch miteinander in Formeln kombiniert werden und somit ein falsches Ergebnis in einem Zwischenschritt weiterverarbeitet wird. Daher ist es notwendig, jedes Zwischenergebnis manuell mit geeigneten Hilfsmitteln zu berechnen, um Logikfehler in den Formeln auszuschließen. Dies sollte im Rahmen der Tests mit beispielhaften Werten berücksichtigt werden. Bei dem im Vorhaben generierten Code war dies nicht erforderlich, da unveränderte Einlesewerte

verwendet wurden. Für das Auftreten eines derartigen Fehlers ist in unserem Fall somit der Entwickler verantwortlich (menschlicher Fehler). Sofern Test- und Verifizierungsmethoden nicht korrekt angewendet werden, besteht das Risiko, dass ein Programm falsche Werte verarbeitet und somit nicht die gewünschte Funktion ausführt. Daher wird dieser Fehler als potenziell kritisch eingestuft.

5.1.2 Variablendoppelbelegung

Eine Doppelbelegung von Variablen führt in der Regel zum Aufrufen von falschen Werten, was eine Beeinträchtigung der Funktionalität des entwickelten Programms zur Folge hat. Würde zum Beispiel bei dem generierten Code der Auswahlschaltung einer der Sensoren doppelt eingelesen werden, so würde das Programm sich unbeabsichtigt von einer 2-von-3 zu einer 1-von-2-Auswahlschaltung verändern. Solche Programmierfehler (ugs. „Bugs“) sind zumeist schwer zu verhindern und zu erkennen. Das Auftreten dieser Fehler kann im Vorfeld beispielsweise durch eine defensive Programmierweise und eine geeignete Strukturierung des Programms minimiert werden. Beim Code der Auswahlschaltung wurde z.B. explizit jedem einzelnen Sensor ein eigener Wert zugewiesen, der beim Aufrufen des jeweiligen Sensors immer zusätzlich abgefragt wird. Sollte versehentlich ein Wert doppelt angefragt werden, würde der Compiler das Programm in unserem Beispiel nicht übersetzen. Solche Fehler werden in der Regel erst beim fehlerhaften Ausführen der jeweiligen programmierten Sicherheitsfunktion erkannt. Daher wird die korrekte Anwendung von Test- und Verifizierungsmethoden als essentiell für die Vermeidung dieser Fehler angesehen. Sofern keine defensive Programmierweise angewendet wird, ist der Fehler als kritisch einzustufen, da seine Vermeidung ausschließlich in der Verantwortung des Entwicklers liegt, und er somit als menschlicher Fehler erachtet wird. Die Doppelbelegung von Variablen wurde im Rahmen der Fehlerinjektion als postulierter Fehler (siehe Kapitel 5.4) untersucht.

5.1.3 Logikfehler einer bedingten Anweisung - Vergleichszeichen Fehler

Eine bedingte Anweisung ist unter definierten Bedingungen erfüllt. Die Bedingung muss hierbei eindeutig mathematisch formuliert werden, damit das Programm die erwünschte Funktionalität erfüllt. Zum Beispiel muss bei der 2-von-3-Auswahlschaltung eine bedingte Anweisung zwei der drei Sensoren alternierend vergleichen und dabei auch alle drei Sensoren miteinander vergleichen. Bei dem im Vorhaben entwickelten FPGA-Design wurde die zuletzt genannte Abfrage zunächst bei der Entwicklung vergessen. Im Allgemeinen sollten Logikabfragen möglichst vollständig abgebildet sein, um die Wahrscheinlichkeit des Auftretens von Anomalien und Fehlern eines komplexeren Programms, welches die Logikabfragen weiterverarbeitet, zu reduzieren. Des Weiteren hätten beispielsweise das versehentliche Vertauschen eines Vergleichszeichens oder das Weglassen eines Gleichheitszeichens zur Folge, dass die Funktion nicht wie spezifiziert ausgeführt wird. Solche Fehler treten häufig bei der Software-Implementierung auf. Durch geeignete Test- und Verifizierungsmethoden können menschliche Fehler dieser Art erkannt werden. Da die Möglichkeit der Fehlererkennung von der Wirksamkeit der angewendeten Test- und Verifizierungsmethoden abhängt, werden die beschriebenen Logikfehler als potenziell kritisch eingestuft.

5.1.4 Falsche Festlegung von Grenzwerten

Eine falsche Festlegung von Grenzwerten und anderen Konstanten, entweder in der Spezifikation oder hervorgerufen durch Fehler in der Entwicklung, z.B. aufgrund der Verwendung unterschiedlicher Einheiten (wie Grad Celsius und Grad Fahrenheit), führt potenziell zur fehlerhaften Ausführung der Sicherheitsfunktion, da diese nicht unter den vorgesehenen Bedingungen ausgelöst wird. Sofern die Grenzwerte und Konstanten richtig spezifiziert, jedoch falsch umgesetzt wurden, können resultierende Fehler in der Testbanksimulation erkannt werden. Eine falsche Festlegung von Grenzwerten wurde im Rahmen der Fehlerinjektion postulierter Fehler (siehe Kapitel 5.4) untersucht. Solche Fehler werden durch den Menschen hervorgerufen, sind durch gängige Test- und Verifizierungsmethoden identifizierbar und werden daher als eher unkritisch eingestuft.

Fehler in der Spezifikation wiederum lassen sich zumeist nicht automatisiert (z.B. durch Tests) erkennen, da die Testmethoden auf Grundlage der Spezifikation erarbeitet werden, um die Übereinstimmung der Realisierung mit der Spezifikation zu prüfen. Solche Fehler lassen sich während der Entwicklung oftmals nur durch manuelle Verifizierungsmethoden (z.B. durch Inspektion) erkennen und korrigieren, weshalb Fehler in der Spezifikation als kritisch eingestuft werden.

5.1.5 Syntaxfehler im C bzw. C++ Code

Diese Designschwachstelle ist bei der Entwicklung der FPGA-Designs im Laufe des Vorhabens aufgefallen. Generell sollten Compiler stets eine Programmiersprache mit einer zugehörigen Syntaxversion verwenden. Einige Anwendungsprogramme, wie der im Vorhaben verwendete HLS-Compiler von Xilinx, sind potenziell in der Lage, C++ Syntax-Versionen mit C Syntax-Versionen zu vermengen, was zu einer inkorrekten Übersetzung des Codes führen kann. Da die korrekte Übersetzung durch den Compiler insbesondere in diesem Fall schwer zu überprüfen ist, kann beispielsweise ein diversitärer Compiler zur Übersetzung herangezogen werden. Die Designschwachstelle wurde mittels Tests untersucht, bei denen das Übersetzen zusätzlich mit einem diversitären Compiler durchgeführt wurde, um den Einfluss der beschriebenen Syntaxvermengungen verschiedener Programmiersprachenversionen auf den Code zu untersuchen (siehe Kapitel 4.6.6). Ein Übersetzungsfehler des Compilers wird als kritisch angesehen, da diese als schwer nachweisbar erachtet werden. Im Allgemeinen besteht bei der Softwareentwicklung die Gefahr, Werkzeugen (z.B. Compilern, Linkern oder Codegeneratoren) ein höheres Maß an Vertrauen in Bezug auf deren Korrektheit entgegenzubringen als durch Nachweise belegt wurde.

5.1.6 Vermengung der Dezimalstellenkonvention

Für die Anzeige von Dezimalstellen werden (international gesehen) sowohl Punkt als auch Komma gleichermaßen verwendet. Eine Vermengung von beiden Konventionen innerhalb einer Entwicklung führt zu Fehlinterpretationen im Programm, da die jeweils andere Dezimalstellentrennung vom Programm als buchstabenähnliches Symbol interpretiert wird, jedoch nicht als Bestandteil einer Zahl. Die Vermengung der Dezimalstellenkonvention wurde im Rahmen der Fehlerinjektion postulierter Fehler (siehe Kapitel 5.4) untersucht. Dieser menschliche Fehler wird als unkritisch eingestuft, da er sich mittels Testbanksimulation vergleichsweise leicht erkennen lässt.

5.1.7 Fehler bei Definition der Variablen und Rundungen

Ein Sensor liefert in jedem Fall Werte, welche einer Teilmenge der rationalen Zahlen entsprechen. Das heißt, die Ausgabewerte der Sensoren können sowohl ganze Zahlen als auch gebrochen rationale Zahlen sein. Das Programm muss somit beide Zahlentypen akzeptieren und verarbeiten können. Darüber hinaus müssen vom Programm auch gerundete Werte verarbeitet werden können. Potenzielle menschliche Fehler hierbei werden durch die Testbanksimulation während der Fehlerinjektion abgefangen und werden als eher unkritisch angesehen. Fehler bei Definition der Variablen und Rundungen wurden im Rahmen der Fehlerinjektion postulierter Fehler (siehe Kapitel 5.4) untersucht.

5.1.8 Werkzeugbasierte Fehler

Neben den Compilern gibt es eine Reihe weiterer Werkzeuge, die in einem Entwicklungsprozess genutzt werden. Beispiele aus der im Vorhaben durchgeführten Entwicklung sind der High Level Synthesizer (HLS) und ein spezieller Werkzeug-Set (z.B. Vivado Toolsuite), welche für das Synthetisieren, das Platzieren und auch Kompilieren verantwortlich sind. Um möglichen werkzeugbasierten Fehlern vorzugreifen, wurden diversitäre Werkzeuge herangezogen und damit die Funktionsweise der im Vorhaben verwendeten Werkzeuge verifiziert. Im Laufe des Vorhabens wurde daher, neben den Xilinx HLS und der Vivado Toolsuite, der Intel Altera HLS und das Werkzeug Quartus Prime benutzt (siehe Kapitel 4.6), um die Synthese, das Kompilieren und Platzieren unabhängig von der Entwicklung durchführen zu können. Ein werkzeugbasierter Fehler wird als kritisch eingestuft, da insbesondere die Korrektheit der verwendeten Werkzeuge häufig nicht ausreichend hinterfragt wird.

5.1.9 Auswirkungen von Strahlung auf FPGAs

An dieser Stelle werden die Auswirkungen von Strahlung auf FPGAs und mögliche Fehlermodi aufgezeigt. Einige Fehlermodi wie sogenannte Soft-Errors wirken sich flüchtig auf das FPGA aus, d.h. durch den Einfluss von Strahlung können einzelne Bits „kippen“, was zu einer veränderten und auch falschen Funktionsweise des FPGA-Designs führen kann. Soft-Errors treten zufällig auf und hängen stark von der verwendeten Technologie ab. Die Häufigkeit von Soft-Errors nimmt mit steigernder Integrationsdichte und geringerer Betriebsspannung exponentiell zu (siehe

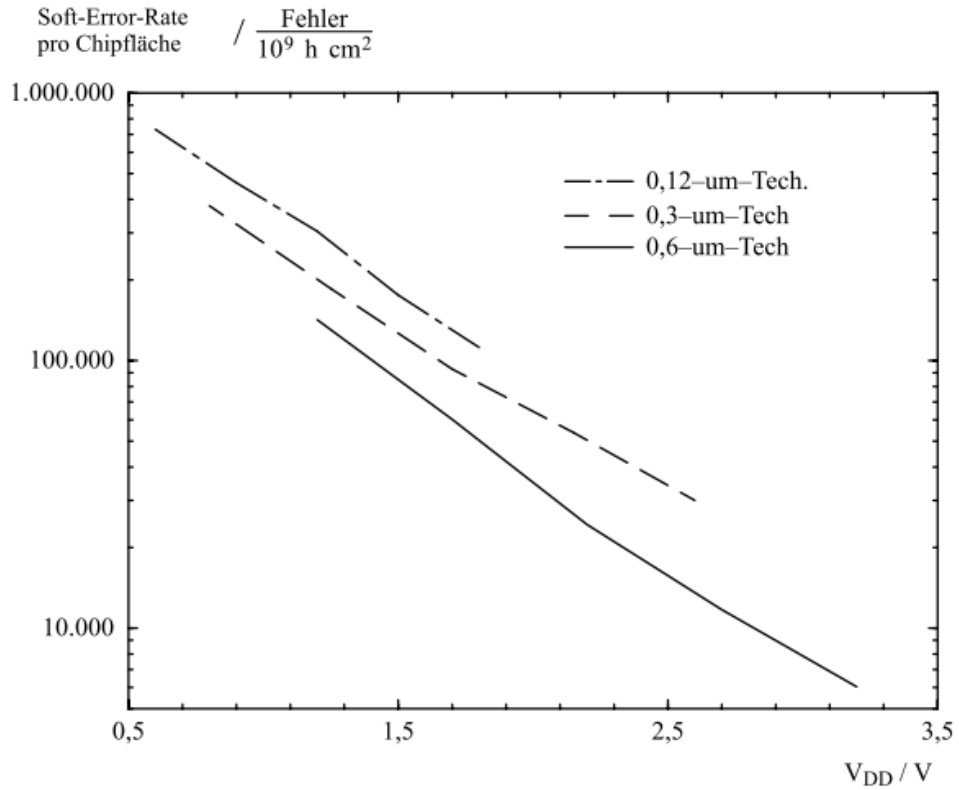


Abbildung 21). Soft-Errors stellen keinen permanenten Fehler der Hardware dar und sind somit nicht direkt reproduzierbar. IEC 61508 [47] beschreibt die Anwendung von Redundanz als effektivste Maßnahme zur Fehlervermeidung für Soft-Errors. Dabei kann eine physische Redundanz, also die Ausführung der Funktion in zwei Redundanzen mit anschließendem Vergleich, oder eine logische Redundanz durch mehrfache Ausführung der Funktion nacheinander mit anschließendem Vergleich, erfolgen. Als weitere Maßnahmen für die Kontrolle von Soft-Errors sind die Verwendung von „Error-Detection-Codes“ (EDC), „Memory-Management-Unit“ MMU und diverse Speichertests empfohlen.

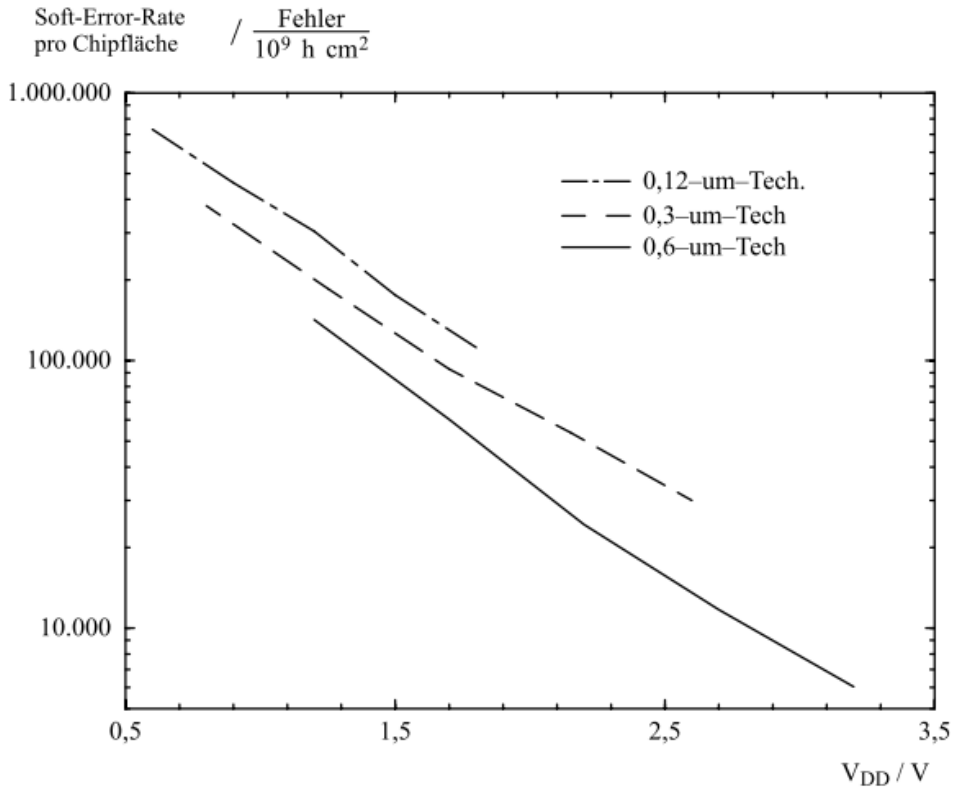


Abbildung 21: Soft-Error-Rate pro Chipfläche für das Gehäuse als Alphapartikelquelle (typische Technologieparameter) [48]

Radioaktive Strahlung kann zu sogenannten „Single Event Upsets“ führen. Die Häufigkeit dieser Ereignisse ist abhängig von der Energie des auslösenden Teilchens (z.B. Proton, Neutron) und der Strahlungsdichte [49]. Ein solches Ereignis beeinträchtigt die Arbeitsweise des FPGA. Auftretende Fehler können erkannt und korrigiert werden. Xilinx hat hierfür z.B. einen Soft Error Mitigation Controller (SEMC) entwickelt [50], welcher in Echtzeit Fehler im FPGA erkennt und korrigiert. Bei großer Strahlenbelastung wird der FPGA neu gestartet, um sämtliche Bit-Flips zu bereinigen.

Die Vernachlässigung von Strahlungseinflüssen auf FPGAs kann im Betrieb zu kritischen Fehlern führen. So sind gemäß IEC 61508 ab einem vom Hersteller reklamierten Diagnosedeckungsgrad von 90 Prozent oder mehr („medium diagnostic coverage“) die Berücksichtigung von Strahlungseinflüssen wie Soft-Errors gefordert [47]. Die Untersuchungen der FPGA-Fehlermodi in diesem Vorhaben erfolgen softwarebasiert und die FPGA-Designs werden nicht in Hardware konfiguriert, weshalb die hier diskutierten hardwarebasierten Fehlermodi im Vorhaben nicht untersucht wurden.

5.2 Testmethoden

Einige Testmethoden werden bereits innerhalb der Entwicklungswerkzeuge Vitis HLS und Vivado Design Suite durchgeführt. Die Testmethoden, die bei der Erstellung der FPGA-Designs unterstützen und somit direkt zur Entwicklung beitragen, sind im Folgenden kurz zusammengefasst.

5.2.1 C-Simulation

Die Korrektheit des anfänglich in C/C++ geschriebenen Programms wird durch eine eigens erstellte C/C++ Testbank geprüft. Bei dieser Testmethode, welche in Vitis HLS als „C-Simulation“ bezeichnet wird, ruft die Testbank das Programm mehrfach unter Verwendung von vorab definierten Testdaten auf und testet, ob die Ergebnisse des Programms mit den erwarteten Resultaten übereinstimmen. Damit werden funktionale Fehler im Programm identifiziert und so die Korrektheit des C/C++ Codes geprüft. Beim Start der C-Simulation kann zudem ausgewählt werden, ob eine Debugging-Umgebung hinzugezogen werden soll, womit etwaige Fehler im Programm direkt erkannt und korrigiert werden können.

Die Qualität der Testung des C/C++ Codes hängt unmittelbar von der Qualität der verwendeten Testbank ab. Falls die Testbank bestimmte Testfälle nicht abdeckt oder inkorrekt behandelt, können entsprechende Fehler bei der C-Simulation unentdeckt bleiben. Weiterhin kann die C-Simulation keine Aussagen über das Timing oder die physische Umsetzung des Programms treffen, da dieses noch nicht in HDL umgesetzt ist.

Mithilfe der C-Simulation konnten während der Entwicklung mehrere unbeabsichtigte Fehler in den FPGA-Designs aufdeckt und korrigiert werden, wie z.B. ein fehlerhafter Grenzwert oder die fehlerhafte mehrfache Verwendung von Variablen.

5.2.2 Analyse des HDL-Designs

Bei der Synthese des C/C++ Programms zu HDL erstellt Vitis HLS zugleich einen Synthesebericht, in dem Daten zum Zeitverhalten und zur Ressourcennutzung des HDL-Designs angegeben sind. Damit lassen sich unter anderem die geschätzte Ausführungsdauer des Designs, die zeitliche Verzögerung, die Schnittstellen und ihre jeweils benötigte Zahl an Bits erkennen. Der Synthesebericht beinhaltet zudem eine Auswertung der geplanten zeitlichen Eigenschaften des Designs. Abbildung 22 zeigt als Beispiel einen zeitlichen Verstoß im Synthesebericht. Die vom Werkzeug geschätzte Ausführungsdauer des HDL-Designs plus der zeitlichen Unsicherheit überschreitet hier die kalkulierte Zeitperiode von 20 ns. Die zeitliche Abweichung zur angestrebten Zielzeit des Designs von -0.12 ns ist hier rot hervorgehoben. Der Fehler wurde in einer folgenden Version behoben.

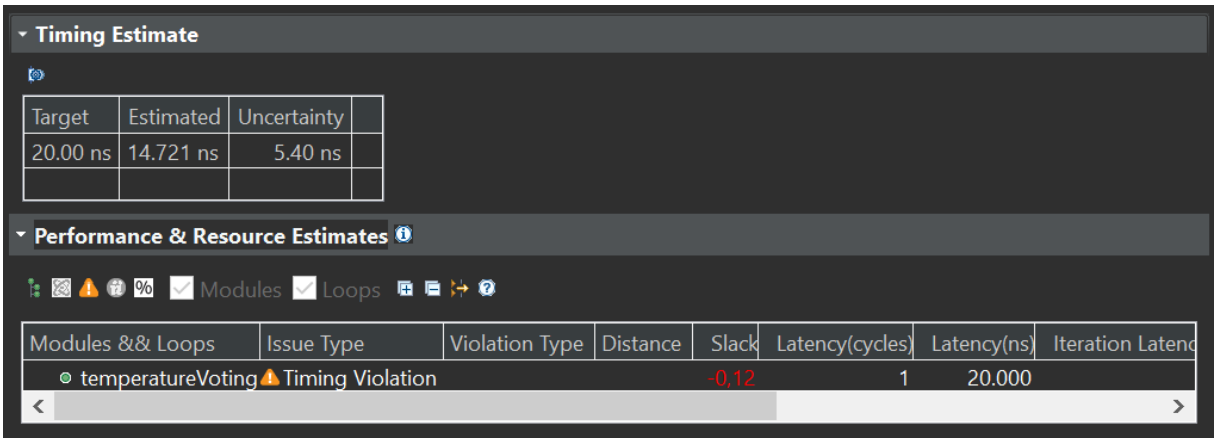


Abbildung 22: Zeitlicher Verstoß im Synthesebericht der 2-von-3-Auswahlschaltung.

Weiterhin wird im sogenannten „Schedule Viewer“ angezeigt, wie sich die einzelnen Operationen zeitlich und in Abhängigkeit voneinander verhalten. Daran lässt sich ablesen, welche Operationen für eine Überschreitung der geplanten Laufzeit verantwortlich sind. Anhand dessen kann das C/C++ Programm korrigiert und abermals synthetisiert werden, um etwaige Fehler zu beheben.

Anhand dieser Analyse können Fehler wie z.B. Timing-Fehler des HDL-Designs, falsche physische Anforderungen und unpassende Umsetzung in Bezug auf Operationen und Arbeitsschritte entdeckt werden. An dieser Stelle ist aber noch nicht zu erkennen, ob das HDL-Design tatsächlich die korrekte Funktionalität aufweist.

Während der Entwicklung hat die Analyse des HDL-Designs (insbesondere mithilfe des „Schedule Viewer“) Abweichungen der Umsetzung zu der Spezifikation bzw. den Erwartungen des Designteams ermittelt. Durch entsprechende Anpassungen des C/C++ Codes und erneuter Synthese wurden die Designs gemäß den spezifizierten Eigenschaften korrigiert.

5.2.3 C/RTL Co-Simulation

Eine weitere Methode in Vitis HLS, die als „C/RTL Co-Simulation“ bezeichnet wird, testet das HDL-Design auf funktionale Äquivalenz zu dem C/C++ Programm. Dafür simuliert Vitis HLS beide Versionen und vergleicht die Ergebnisse der Simulationen miteinander. Die Co-Simulation gilt als bestanden, wenn die Ergebnisse jeweils gleich sind. Für die Simulation des C/C++ Programms wird die zuvor für die C-Simulation erstellte Testbank eingesetzt. Für die RTL-Simulation hingegen übersetzt Vitis HLS die C-Testbank zu einer RTL-Testbank und verwendet diese zur Ermittlung der Testergebnisse.

Mithilfe der C/RTL Co-Simulation werden funktionale Fehler im HDL-Design festgestellt. Wie bei der C-Simulation stellt es aber keine Fehler fest, die aufgrund einer inkorrekten Testbank auftreten. Der Co-Simulationsbericht gibt zudem an, für wie viele Zeitintervalle das Design bzw. dessen Unterfunktionen arbeiten. Es werden hier jedoch keine Fehler zur physischen Umsetzung erkannt.

Alle entwickelten FPGA-Designs haben jeweils die C/RTL Co-Simulation bestanden und werden somit als funktional äquivalent zu ihrem Quellcode erachtet.

5.2.4 Wellenfunktion

Zur genaueren Untersuchung des Verhaltens des HDL-Designs kann bei bestandener Co-Simulation die zugehörige Wellenfunktion erstellt und in Vivado angezeigt werden. Mit ihr lässt sich das von Vitis HLS erwartete Zeitverhalten des Entwurfes ermitteln. Die durch die Wellenfunktion gelieferten Testdaten umfassen dabei, welche Werte die “Variablen“ (Ports) zu bestimmten Zeiten annehmen, Ausgabedaten und interne FPGA Signale (beispielsweise das Taktsignal). Anhand der Wellenfunktion kann auch abgelesen werden, ob bei einem bestimmten Testfall ein oder mehrere Signale fehlen. Somit können Testfälle, die einen Signalverlust simulieren (beispielsweise durch Kabelbruch), unmittelbar erkannt und analysiert werden.

Anhand der Wellenfunktion kann abgelesen werden, ob zeitliche Fehler in Bezug auf das Timing des Designs existieren. Da die Wellenfunktion weiterhin die Ein- und Ausgabewerte konkret anzeigt, können die genauen Werte abermals auf Sinnhaftigkeit begutachtet werden.

Dadurch können auch funktionale Fehler entdeckt werden, die eine inkorrekte Testbank fehlerhaft behandelt.

Die Wellenfunktion hat bei ersten Iterationen der Designs ungewünschte Eigenschaften aufgedeckt. So gab es beispielsweise eine Version der 2-von-3-Auswahlschaltung, bei der abhängig von den Eingabewerten die Ausführungsdauer der FPGA-Anwendung unterschiedlich war. Damit konnten geeignete Umsetzungsstrategien für die spezifizierten Anforderungen ermittelt werden. Bei den finalen Umsetzungen der FPGA-Designs zeigt die Wellenfunktion das erwünschte Verhalten an.

5.2.5 Vivado-Simulation

In Vivado lässt sich der Entwurf mittels einer eigens erstellten HDL-Testbank simulieren. Die Simulation des aus Vitis HLS exportierten Designs wird als „Behavioral Simulation“ (Verhaltenssimulation) bezeichnet. Nach Synthese und Implementierung besteht jeweils die Möglichkeit zur funktionalen Simulation. Damit wird getestet, ob das Design auch nach Synthese und Implementierung das erwartete Verhalten aufweist oder ob es sich geändert hat und entsprechend korrigiert werden muss. Die Simulation wird vom Nutzer gesteuert und kann je nach Bedarf für das komplette Design, für ein festgelegtes Zeitintervall oder iterativ für einen Arbeitsschritt nach dem anderen durchlaufen werden. Parallel dazu wird eine Wellenfunktion erstellt, anhand derer die Ein- und Ausgaben der Simulation abgelesen werden können. In Verilog kann weiterhin mittels „Timing Simulation“ getestet werden, ob das Zeitverhalten des Designs die zeitlichen Spezifikationen erfüllt. Ähnlich wie zur C-Simulation in Vitis HLS lässt sich bei der Vivado-Simulation ein Debugging des HDL-Designs durchführen (allerdings nur bei synthetisierten Designs). Damit lassen sich fehlerhafte Stellen im Design identifizieren und korrigieren.

Mittels der Vivado Simulation [41] können funktionale und zeitliche Fehler im HDL-Design gefunden werden. Ebenso wie bei den vorherigen Simulationen sollte die Qualität bzw. Korrektheit der Testbank gesichert werden. Bei einer unzureichenden Testabdeckung können funktionale Fehler unbemerkt bleiben. Fehler in der Testbank lassen sich mittels der Wellenfunktion (wie in Kapitel 5.2.4 beschrieben) feststellen.

5.2.6 Vivado-Analysen

Nach Import des Designs in Vivado können direkt erste Analysemethoden eingesetzt werden. Eine dieser Methoden ist die Möglichkeit, das Design in Form eines Schaltplans anzuzeigen. Damit kann abgelesen werden, ob das HDL-Design die erwartete Struktur aufweist und das gewünschte Verhalten sinnhaft ausführt. Zudem kann in späteren Entwicklungsphasen auch das synthetisierte / implementierte Design als Schaltplan angezeigt werden.

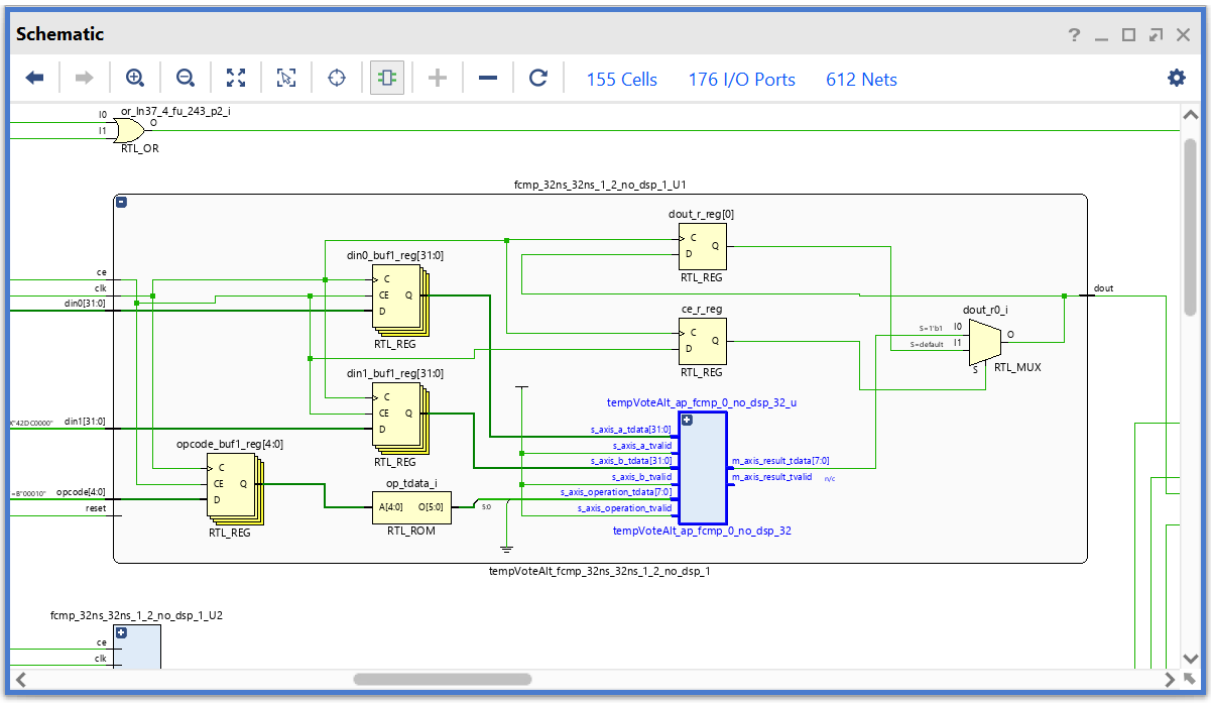


Abbildung 23: Untermodul mit Vivado-IP im Schaltplan der 2-von-3-Auswahlschaltung

Während der FPGA-Entwicklung wurde durch die Schaltplansimulation erkannt, dass zunächst Teile des Designs unter Verwendung geschützter „intellectual property“ (IP) von Vivado umgesetzt sind, was zur Generierung von verschlüsselten EDIF-Netzlisten geführt hat. In Abbildung 23 ist ein Schaltplanabschnitt einer Version der 2-von-3-Auswahlschaltung zu sehen, bei dem geschützte Vivado-IPs verwendet wurden. Im blau umrandeten Untermodul wird Vivado-IP verwendet, wodurch die Netzliste verschlüsselt wird. Die Designs wurden entsprechend angepasst, so dass die in Vivado importierten HDL-Dateien keine geschützte Vivado-IP einsetzen. Als Resultat konnten für die Komplexitätsanalyse verwendbare EDIF-Netzlisten aus den Designs generiert werden.

Eine weitere früh anwendbare Testmethode in Vivado ist der sogenannte „Design Rule Check“ (kurz: „DRC“). Dieser testet, ob das exportierte (oder synthetisierte bzw. implementierte) Design die in Vivado vorgefertigten Entwurfsregeln erfüllt. Mithilfe dieser Entwurfsregeln wird sichergestellt, dass das Design den von Xilinx empfohlenen Verfahren folgt und korrekt funktioniert. Es kann beim Start des „DRC“ in einem Fenster konkret ausgewählt werden, welche dieser Regeln beim Test beachtet werden sollen. Je nach gewählten Regeln können Fehler im Timing, in der physischen Umsetzung und in der formalen Korrektheit der HDL-Dateien erkannt werden. Bei Verstößen gegen die Entwurfsregeln erstellt der „DRC“ einen Bericht, in dem die einzelnen Verstöße und deren Ausmaß aufgelistet sind. Der „DRC“ hat bei den Designs in keinem der Entwicklungsschritte wesentliche Abweichungen diagnostiziert. Somit sind bei der Realisierung der FPGA-Designs, die von Xilinx vorgegebenen Entwurfsregeln eingehalten.

Vivado verfügt außerdem über mehrere Testmethoden, die zur Post-Route Analyse nach dem Platzieren eingesetzt werden können. Bei diesen Testmethoden ermittelt Vivado die Charakteristiken des implementierten Designs, wie dessen Timing, Energienutzung und Ressourcennutzung, und fasst diese in jeweiligen Berichten zusammen. Die in den Ergebnisberichten angegebenen Daten basieren auf der konkreten Realisierung des Designs und sind somit die realistischsten Angaben zum FPGA-Design vor der physischen Umsetzung.

Diese können anschließend genau analysiert werden. Jede dieser Analysen kann zudem auf dem synthetisierten Design angewendet werden und so vorgreifende Resultate liefern. Die daraus gewonnenen Daten basieren dabei auf Schätzungen von Vivado.

Der Bericht zum Timing gibt Daten zu den Eigenschaften der einzelnen Pfade, wie deren maximale und minimale Verzögerung oder Pufferzeit und ob das Design Abweichungen zu den zeitlichen Vorgaben aufweist. Welche konkreten Pfade im Bericht angezeigt werden sollen, lässt sich zu Beginn der Timing Analyse einstellen. Anhand des Berichts kann getestet werden, ob das Design die zeitlichen Spezifikationen einhält. Damit lassen sich Fehler im Timing des Designs identifizieren und entsprechend ausbessern.

Der Bericht zur Ressourcennutzung zeigt an, wie viele bzw. welcher Anteil an Hardwareelementen des Ziel-FPGA vom Design verwendet wird. Es wird sowohl die Anzahl als auch der prozentuale Anteil davon angegeben, wie viele Ein- und Ausgabepins und Look-Up Tables des FPGA verwendet werden. Damit lässt sich ermitteln, ob die Spezifikation hinsichtlich der Auslastung des FPGA erfüllt ist. Die Auswertung hat gezeigt, dass sich die entwickelten FPGA-Designs hinsichtlich der Auslastung im Rahmen des ausgewählten Ziel-FPGA befinden.

Bei dem Bericht der Energienutzung wird angezeigt, wie viel Energie das FPGA mit dem entwickelten Design verbraucht und wie viel Wärme es produziert. Es wird angegeben, welcher Anteil des Energieverbrauchs aus statischem und dynamischem Verhalten des FPGA hervorgeht. Beim dynamischen Anteil wird in Energieverbrauch aus internen Signalen, Ein- und Ausgangssignalen, und logischen Operationen unterschieden. Anhand des Berichts kann das Einhalten von Spezifikationen zur Energienutzung getestet werden. Bei einer sehr hohen Temperatur oder einem übermäßigen Energieverbrauch sind Fehler des FPGAs in Abhängigkeit der Einsatzumgebung zu erwarten. Da für dieses Projekt die physische Implementierung der Designs entfällt, sind entsprechend keine Umgebungsspezifikationen (wie Umgebungstemperatur und Luftzufuhr) gegeben. Es wurde daher auf typische Umgebungskennwerte zurückgegriffen, die im Werkzeug für die jeweils ausgewählte Ziel-Hardware als Voreinstellung des Herstellers hinterlegt sind. Unter Berücksichtigung der voreingestellten Kennwerte zeigt die Analyse zur Energienutzung keinerlei Auffälligkeiten. Dies lässt die Vermutung zu, dass die entwickelten FPGA-Designs realistisch umsetzbar sind.

In Vivado lässt sich außerdem graphisch darstellen, wie das Design auf die FPGA-Architektur platziert und verbunden wird. Daraus lässt sich ableiten, wie die Ein- und Ausgaben des Designs den Pins des Ziel-FPGA zugeteilt werden. Für ein implementiertes Design lässt sich somit anzeigen, wie es konkret auf dem FPGA verbunden wird. Potenzielle Fehler, wie eine falsche Zuweisung der Pins oder eine fehlerhafte Verbindung, können daran abgelesen werden. Damit lässt sich testen, ob das Design den korrekten Pins zugeteilt wurde (und damit die physischen Spezifikationen erfüllt werden) und ob die genaue Verbindung sinnhaft ist. Bei den finalen Umsetzungen der FPGA-Designs zeigt die Vivado Analyse das erwünschte Ergebnis an.

5.3 Ermittlung von Schwachstellen und potenziellen Fehlerquellen

Zur Ermittlung des Fehlerpotentials und der Fehlermodi wurden die entwickelten Designs bzw. Design-Varianten auf FPGA-Basis (siehe Kapitel 4.5) und die zugehörige Testumgebung (siehe Kapitel 4.6) herangezogen. Anhand der Erfahrungen aus der Entwicklung wurden mögliche Schwachstellen im Design und daraus resultierende potenzielle Fehlerquellen

ermittelt (siehe Kapitel 5.1). Diese wurden für die Generierung von Testfällen und zur Postulierung möglicher Fehler zur Fehlerinjektion genutzt.

Die identifizierten potenziellen oder tatsächlichen Fehler sind in Kapitel 5.1 dokumentiert und unter Berücksichtigung ihrer sicherheitstechnischen Auswirkungen klassifiziert. In Tabelle 7 sind die potenziellen Fehlerquellen aus der Betrachtung der Designschwachstellen sowie die Möglichkeit ihrer Erkennung mit den erarbeiteten Testmethoden und verbleibende potenzielle Testlücken zusammengefasst. Die verbleibenden Testlücken wurden bei der Fehlerinjektion (siehe Kapitel 5.4) entsprechend berücksichtigt.

Tabelle 7: Potenzielle Fehlerquellen bei der FPGA-Entwicklung

Fehler	Beispiel	Testmethode	Potenzielle Testlücken
Fehler im C++ Programm	Falsche Grenzwerte mehrfache Variablennutzung Logikfehler	C-Simulation Dynamische Analysen Ermittlung der zyklomatischen Komplexität	Fehler in der Testbank z.B. hinsichtlich Grenz- und Eingabewerte, Syntax (Fehlerinjektion erforderlich)
Fehlerhafte Ressourcen-nutzung	Timing-Fehler (Signallaufzeiten)	Analyse des HDL-Designs	Korrekte Funktionalität des HDL-Designs (C/RTL Co–Simulation erforderlich)
Fehler im HDL-Design	Funktionale Unterschiede zwischen C++ und HDL	C/RTL Co-Simulation Vivado Simulation	Fehler in der Testbank (Fehlerinjektion erforderlich) Fehler in der physischen Umsetzung (Vivado Analysen erforderlich)
Zeitliche Fehler	Falsches Timing des Designs Signalverlust	Wellenfunktion	Fehler in der Testbank sind nur teilweise aufdeckbar (Fehlerinjektion erforderlich)
Fehler in der Platzierung	Falsche Pin-Zuweisung	Vivado Analysen	Fehlerhafte Vorgaben (Spezifikation)
Werkzeug-bedingte Fehler	Fehler im Design werden erzeugt oder bleiben unerkannt	Diversität der Testmethoden (Compiler, Betriebssystem, Synthese, Simulator)	-

5.4 Fehlerinjektion postulierter Fehler

Die Untersuchung eigens entwickelter FPGA-Designs ermöglicht es, die generierte Testumgebung durch Fehlerinjektion auf Schwachstellen zu untersuchen und die erreichte Testabdeckung einzuschätzen. Zur Postulierung möglicher Fehler zur Fehlerinjektion wurden Erfahrungen aus der Entwicklung der FPGA-Designs und der Testumgebung, mögliche Schwachstellen im Design und daraus resultierende potenzielle Fehlerquellen berücksichtigt. Durch die Injektion realistischer postulierter Fehler sollen die Werkzeuge zur Testfallgenerierung erprobt und ihre Effektivität ermittelt werden. Potenzielle Testlücken wurden durch die Fehlerinjektion identifiziert und entsprechend dokumentiert. Durch die Fehlerinjektion wurden insbesondere die potenziellen Fehler untersucht, die mit den bisherigen Testmethoden (siehe Kapitel 5.2) auf Grundlage der Testumgebung (siehe Kapitel 4.6) nicht oder nicht ausreichend untersucht werden konnten.

Technisch wird die Fehlerinjektion durch eine Manipulation des C++ Quellcodes vorgenommen. Mithilfe der eigens erstellten Testbänke wird die Testabdeckung der Testumgebung untersucht und die möglichen Auswirkungen der injizierten Fehler simuliert.

Die hierfür untersuchten Testschritte sind das Kompilieren, das Debugging (C-Simulation, siehe Kapitel 5.2.1), die Testbanksimulation (C/RTL Co-Simulation, siehe Kapitel 5.2.3) und die Wellenfunktion (siehe Kapitel 5.2.4). Es wird überprüft, ob der jeweilige Testschritt erfolgreich ausgeführt wird oder der injizierte Fehler in einem Testschritt erkannt werden kann. Der Fehler gilt in einem Testschritt als erkannt, wenn die Kompilierung scheitert, das Debugging einen Fehler meldet, die Testbanksimulation abbricht oder einen Fehler meldet, oder die Generierung der Wellenfunktion scheitert.

5.4.1 Sensorausfall

Zunächst wird mittels Fehlerinjektion der Ausfall eines Sensors simuliert. Durch Manipulation der (.dat) Datei (siehe Kapitel 4.6.5) wurden die für die Fehlerinjektion notwendigen Eingabewerte in der Testbank simuliert. In der (.dat) Datei ist jedem Sensor eine Spalte zugeordnet. Im Zuge der Fehlerinjektion wurde die Zelle einer Spalte frei gelassen, um einen Sensorausfall zu simulieren.

Ergebnis:

- Das Programm wird trotz des injizierten Fehlers erfolgreich (d.h. ohne Auffälligkeiten oder Abbruch) kompiliert.
- Beim Debugging des Programms in der Testbanksimulation wird ein Fehler erkannt. Der Fehler tritt bei der frei gelassenen Zelle auf.
- Die Testbanksimulation läuft im Vergleich zu einem fehlerfreien Programm nicht vollständig durch.
- Die Wellenfunktion kann nicht ausgegeben und zu weiteren Tests genutzt werden.

Testschritt	Kompilieren	Debugging	Testbanksimulation	Wellenfunktion
Erfolgreich ausgeführt	☒	☐	☐	☐

Die Analyse der Fehlerinjektion zeigt, dass die Struktur der Tabelle in der (.dat) Datei von fünf auf vier Zellen in der manipulierten Zeile reduziert ist. Somit liest die Testbank anstelle des Wertes der freigewordenen Zelle den Wert der darauffolgenden Zelle ein. Als Folge ergibt sich eine fehlerhafte Reihenfolge der eingelesenen Sensorwerte innerhalb der Testbank. Die Testbank simuliert hierbei den Betriebsfall. Das heißt, im Betrieb würde der Fehler in gleicher Weise auftreten. Der potenzielle Fehler ließe sich durch die vorhandene Testumgebung indirekt aufdecken, da die Wellenfunktion nicht generiert werden kann.

Um den Fehler zu vermeiden, muss das Programm entsprechend umgeschrieben werden. Im Programm kann beispielsweise ein defekter Sensor definiert werden. Somit würde bei einem Sensorausfall der Wert für einen defekten Sensor vom Programm verwendet werden. Damit kann das Programm im Falle eines fehlenden oder defekten Sensors als 1-von-2-Auswahl weiterarbeiten. Der defekte Sensor könnte den Anwender im Realbetrieb z.B. durch die Statusänderung einer Kontrolllampe signalisiert werden. Die Veränderung bzw. Korrektur des Programms hat zur Folge, dass das Kompilieren des Programms sowie das Debugging fehlerfrei durchgeführt werden kann. Die Testbanksimulation verläuft vollständig und es kann eine Wellenfunktion generiert werden.

Der Fehler wird nicht als kritisch erachtet, da er sich im Zuge der Entwicklung aufdecken und durch eine Korrektur des Programms vermeiden lässt.

5.4.2 Eingabe von unzulässigen Werten

Mittels Fehlerinjektion werden der (.dat) Datei in diesem Szenario Werte außerhalb der Spezifikation vergeben, die bei einem realen Messspektrum nicht zu erwarten sind. Dabei handelt es sich beispielsweise um negative Werte, um Werte außerhalb des zu erfassenden Messbereiches sowie unerwartete Gradienten.

Ergebnis:

- Das Programm wird trotz des injizierten Fehlers erfolgreich (d.h. ohne Auffälligkeiten oder Abbruch) kompiliert.
- Beim Debugging des Programms in der Testbanksimulation tritt kein Fehler auf.
- Die Testbanksimulation läuft wie bei einem fehlerfreien Programm vollständig durch.
- Die Wellenfunktion kann wie gewohnt ausgegeben und zu weiteren Tests genutzt werden.

Testschritt	Kompilieren	Debugging	Testbanksimulation	Wellenfunktion
Erfolgreich ausgeführt	☒	☒	☒	☒

Die Analyse der Fehlerinjektion zeigt, dass die Testbank nicht unterscheiden kann, ob es sich um einen realistischen Wert handelt oder ein potenzielles Fehlverhalten eines Sensors vorliegt. Der potenzielle Fehler ließe sich durch die vorhandene Testumgebung nicht aufdecken, da alle Testschritte der Entwicklungsumgebung wie bei einem fehlerfreien Programm ohne Auffälligkeiten mit positivem Testergebnis durchlaufen werden.

Um den Fehler zu beherrschen, muss das Programm modifiziert werden. Durch das präventive Einfügen einer unteren Intervallgrenze können induzierte Fehler vom Programm beherrscht bzw. zugeordnet werden. Man spricht hierbei von defensiver Programmierung. Das Fehlen eines Grenzwerts kann in der Praxis nur manuell z.B. durch Verifizierung, oder durch Auftreten eines daraus resultierenden Fehlers und anschließender Analyse ermittelt werden. Die bei der Entwicklung angewendete defensive Programmierung deckte diese Fehler zunächst nicht auf.

Der Fehler wird als potenziell kritisch erachtet, da er sich im Zuge der gewöhnlichen automatisierten Testumgebung der Entwicklungswerkzeuge nicht aufdecken lässt. Mittels defensiver Programmierung, Verifizierung und ggf. Korrektur des Programms lässt sich der Fehler vermeiden.

5.4.3 Variablen Doppelbelegung zur Identifikation von Konflikten zwischen Quellcode und Hardware (Codemanipulation)

Mittels Fehlerinjektion wird ein Missverständnis zwischen dem Entwicklerteam des Programms und den Entwicklerteams der Hardware simuliert. Dabei wurde der Code vom FPGA-Design zur Auswahlschaltung (siehe Kapitel 4.5.2), bei dem die Sensorwerte eingelesen und gespeichert werden, bewusst so manipuliert, dass gewisse Variablen doppelt belegt werden.

Ergebnis:

- Kompilieren des Programms funktioniert nicht.
- Beim Debugging des Programms in der Testbanksimulation werden Fehler erkannt.
- Die Testbanksimulation läuft im Vergleich zu einem fehlerfreien Programm nicht vollständig durch.
- Die Wellenfunktion kann nicht ausgegeben und zu weiteren Tests genutzt werden.

Testschritt	Kompilieren	Debugging	Testbanksimulation	Wellenfunktion
Erfolgreich ausgeführt	☐	☐	☐	☐

Die Analyse der Fehlerinjektion zeigt, dass die Struktur der Programmierung des veränderten Moduls logisch inkonsistent ist, da die Doppelbelegung von Variablen zu Logikfehlern führt, welche durch den Compiler aufgedeckt werden. Bei einer Programmierung von mehreren Modulen und deren Verknüpfung könnte der Compiler unter Umständen diesen Konflikt nicht erkennen, sofern die Module eigenständig arbeiten. Der injizierte Fehler wird als unkritisch angesehen, da er sich im Zuge der Entwicklung durch den Compiler aufdecken lässt und eine Korrektur des Programms vollzogen werden muss. Diese Korrektur muss notwendigerweise im Code vorgenommen werden, um das Programm in einen funktionsfähigen Zustand zu

bringen, andernfalls könnte das Programm nicht erstellt werden und somit nicht in die Vivado Tool Suite eingebunden werden.

5.4.4 Syntax und Semantik Fehler (Codemanipulation)

Durch Fehlerinjektion im Code wird die Funktion des Programms, der 2-von-3-Auswahlschaltung, beeinträchtigt. Um dies zu erzeugen, wurden im Code gezielt Sensorvariablen in unzulässiger Weise in den bedingten Anweisungen miteinander verglichen. Durch die Injektion des Fehlers werden z.B. die Werte eines Sensors mit sich selbst anstelle mit den Werten der anderen Sensoren verglichen. Das heißt, die Fehlerinjektion hat das Programm in eine 1-von-1-Auswahl verändert.

Ergebnis:

- Kompilieren des Programms funktioniert nicht.
- Beim Debugging des Programms in der Testbanksimulation werden Fehler erkannt.
- Die Testbanksimulation läuft im Vergleich zu einem fehlerfreien Programm nicht vollständig durch.
- Die Wellenfunktion kann nicht ausgegeben und zu weiteren Tests genutzt werden.

Testschritt	Kompilieren	Debugging	Testbanksimulation	Wellenfunktion
Erfolgreich ausgeführt	☐	☐	☐	☐

Die Analyse der Fehlerinjektion führt auch hier zum Ergebnis, dass die Struktur des Programms logisch inkonsistent ist. Diese Unzulässigkeit wurde im Vorfeld bei der Entwicklung des Codes spezifiziert und umgesetzt. Durch die Programmierweise kann der mögliche Fehler durch den Compiler beherrscht und aufgedeckt werden. Ohne die genannte Spezifikation und deren Umsetzung hätte der Compiler diesen Fehler gegebenenfalls nicht erkennen können und die weiteren Testschritte würden möglicherweise trotz dem vorhandenen Fehler ohne Abweichungen durchlaufen werden. Der Fehler wird als potenziell kritisch eingeordnet, da in diesem Fall die während der Entwicklung angewendete Programmierweise dazu geführt hat, dass dieser Fehler durch den Compiler aufgedeckt wird. Im Speziellen hat das Anwenden des V-Modells und das damit verbundene Einbeziehen von möglichen Designschwächen des Programms in der Entwicklung dazu geführt, den Code im Vorfeld anzupassen und somit die Eigenschaften des Compilers entsprechend zu nutzen.

5.4.5 Manipulation bei der Grenzwertanpassung (Codemanipulation)

Durch Fehlerinjektion wurde die Grenzwertinterpretation programmtechnisch verändert, indem die nicht vorkonfigurierte Temperatureinheit Grad Fahrenheit (°F) anstelle von Grad Celsius (°C) benutzt wurde, um eine Inkonsistenz zwischen Programm und Hardware zu simulieren. Dafür wurden die Grenzwerte des Programms in der Einheit °F hinterlegt, während die hardwarebasierten simulierten Einlesewerte der Testbank die Einheit°C aufweisen.

Ergebnis:

- Das Programm wird trotz des injizierten Fehlers erfolgreich (d.h. ohne Auffälligkeiten oder Abbruch) kompiliert.
- Debugging des Programms verweist auf Grenzwertfehler.
- Die Testbanksimulation läuft im Vergleich zu einem fehlerfreien Programm nicht vollständig durch.
- Die Wellenfunktion kann nicht ausgegeben und zu weiteren Tests genutzt werden.

Testschritt	Kompilieren	Debugging	Testbanksimulation	Wellenfunktion
Erfolgreich ausgeführt	☒	☐	☐	☐

Sobald es zu einer Grenzwertüberschreitung kommt, wird ein Signal weitergeleitet, welches das Überschreiten des Grenzwertes anzeigt. In der Testbank wird dieses Signal mit Referenzwerten des jeweiligen Sensors verglichen. Tritt bei diesem Vergleich ein Konflikt auf (in diesem Fall durch falsche Einheit verursachte Abweichung), wird dieser Fehler bereits während des Debugging-Prozesses angezeigt. Gleiches gilt für eine Unterschreitung des Grenzwerts. Der Fehler wird als potenziell kritisch eingestuft, da auch in diesem Fall im Vorfeld die Eigenschaften des Debugging-Prozesses bei der Entwicklung analysiert wurden und die Testbank an diese Eigenschaften angepasst wurde, um das Debugging-Werkzeug optimal für den Code zu nutzen. In der Realität muss eine Anpassung an die jeweilige Hardware durchgeführt werden. Verifizierung und Validierung werden auch für dieses Szenario als notwendig angesehen, um den Fehler entsprechend aufdecken bzw. vermeiden zu können.

5.4.6 Manipulation der Präzision der Eingabewerte der Sensoren

Mittels Fehlerinjektion wurde die Robustheit des Programms im Sinne der Präzision im Bereich der Dezimalstellen geprüft. Dieser Fehler sollte den Fall simulieren, dass der Sensor plötzlich eine ganze Zahl anzeigt, aber das Programm eine gebrochen rationale Zahl (Dezimalzahl) erwartet. Daher wurden für diese Simulation einige Einlesewerte der Testbank gezielt zu ganzen Zahlen verändert.

Ergebnis:

- Das Programm wird trotz des injizierten Fehlers erfolgreich (d.h. ohne Auffälligkeiten oder Abbruch) kompiliert.
- Beim Debugging des Programms in der Testbanksimulation tritt kein Fehler auf.
- Die Testbanksimulation wie bei einem fehlerfreien Programm vollständig durch.
- Die Wellenfunktion kann wie gewohnt ausgegeben und zu weiteren Tests genutzt werden.

Testschritt	Kompilieren	Debugging	Testbanksimulation	Wellenfunktion
Erfolgreich ausgeführt	☒	☒	☒	☒

Die Analyse der Fehlerinjektion zeigt, dass das Programm mit Rundungen der Temperatureingabe umgehen kann und diese auch richtig interpretiert. Dieser Fehler wird als eher unkritisch angesehen, da eine Diskussion über die möglichen genutzten Eingabewerte bzw. Verarbeitungswerte in der Regel im Vorfeld bei der Entwicklung eines Programms durchgeführt wird. Sollte hier ein Verarbeitungsfehler der zu nutzenden Zahlen auftauchen, verweist in der Regel der Compiler und/oder der Debugger auf das vorliegende Problem. Um diese Eigenschaft der beiden Tools auszunutzen, kann die defensive Programmierweise vernachlässigt werden, da sie in der Regel standardmäßig integriert ist. Bei der im Vorhaben angewendeten Programmierweise spielte die Präzision der Testbankwerte im Dezimalbereich ebenfalls eine untergeordnete Rolle.

5.4.7 Syntaxfehler bei der Eingabe - Ersetzen von „Punkt“ durch „Komma“

Hintergrund dieser Fehlerinjektion ist die Simulation von Kommunikationsproblemen zwischen den Entwicklern der Software und der Hardware, welche unterschiedliche Dezimalschreibweisen benutzen. In diesem Szenario wird das unbeabsichtigte Vermengen von Dezimalstellenkonventionen (Punkt und Komma) betrachtet. Hierfür wurde in der Datendatei der Punkt, welcher die Dezimalstelle repräsentiert, durch ein Komma ersetzt.

Ergebnis:

- Das Programm wird trotz des injizierten Fehlers erfolgreich (d.h. ohne Auffälligkeiten oder Abbruch) kompiliert.
- Beim Debugging des Programms in der Testbanksimulation tritt kein Fehler auf.
- Die Testbanksimulation läuft im Vergleich zu einem fehlerfreien Programm nicht vollständig durch.
- Die Wellenfunktion kann nicht ausgegeben und zu weiteren Tests genutzt werden.

Testschritt	Kompilieren	Debugging	Testbanksimulation	Wellenfunktion
Erfolgreich ausgeführt	☒	☒	☐	☐

Die Analyse der Fehlerinjektion zeigt, dass das Kompilieren störungsfrei durchgeführt wird, da das Komma als Buchstabe (Sonderzeichen) und nicht als fehlerhafter Zahlenwert interpretiert wird. Die Testbanksimulation ordnet diesen Temperaturwert richtigerweise als unzulässig ein (Sonderzeichen statt Zahl). Dieser Fehler wird als potenziell kritisch eingestuft, da in diesem Fall wiederum die Art und Weise der Testbank-Erstellung diesen Fehler aufdeckt. Die Produktion eines Sensors kann mit einer anderen Dezimalstellenkonvention erfolgen als die Entwicklung der zughörigen Software. Das unbedachte Einbeziehen vorgegebener Werte in den Entwicklungsprozess kann hierbei zu gravierenden Fehlern, wie zum Beispiel

Programmabstürzen, führen. Es ist daher empfehlenswert, die Dezimalstellenkonvention im Entwicklungsprozess gesondert zu prüfen.

5.4.8 Auswertung der Testschritte zur Fehlerinjektion

Es wurden insgesamt sieben verschiedene Szenarien für die Fehlerinjektion auf die entwickelten FPGA-Designs angewendet. Fünf der sieben Fehlerszenarien wurden von dem verwendeten Tool-Set aufgedeckt, somit konnten zwei der Szenarien zur Fehlerinjektion von der im Vorhaben angewendeten Testumgebung nicht erkannt werden. Die Analyse der Szenarien Nr. 2 und Nr. 6 zeigt, dass das Programm trotz des injizierten Fehlers erfolgreich (d.h. ohne Anomalien oder Abbruch) kompiliert wurde. Beim Debuggen des Programms in der Testbank tritt in beiden Szenarien kein Fehler auf. In beiden Fällen läuft die Testbanksimulation erfolgreich und die Wellenfunktion kann wie gewohnt erzeugt und für weitere Tests verwendet werden. Somit konnten für diese zwei Szenarien die potenziellen Fehlermodi von der Testumgebung nicht erkannt werden. Um diese beiden Fehlermodi zu beherrschen, musste jeweils das Programm geändert werden. Tabelle 8 fasst die durchgeführten Fehlerinjektionsszenarien zusammen und zeigt, welche der Fehlermodi durch die verwendeten Testwerkzeuge aufgedeckt werden konnten.

Tabelle 8: Fehlerinjektion von postulierten FPGA-Fehlermodi

Nr.	**Fehler-injektion**	Kompilieren	Debugging	Testbanksimulation	Wellenfunktion
		erfolgreich ausgeführt			
1	Simulation Sensorausfall	☒	☐	☐	☐
2	Unzulässige Werte	☒	☒	☒	☒
3	Variablen Doppelbelegung	☐	☐	☐	☐
4	Syntax- und Semantikfehler	☐	☐	☐	☐
5	Grenzwert-anpassung	☒	☐	☐	☐
6	Präzision der Eingabewerte	☒	☒	☒	☒
7	Eingabefehler Punkt & Komma	☒	☒	☐	☐

Die Analyse der Fehlerinjektion zeigt, dass einige der injizierten Fehler nur durch die Anwendung von Verifizierung und Validierung (V&V) erkannt werden konnten. Vier Szenarien wurden als potenziell kritisch eingestuft, da zusätzliche V&V-Aktivitäten erforderlich waren, um den injizierten Fehler zu identifizieren. So musste beispielsweise bei dem zweiten Szenario („Unzulässige Werte") das Programm geändert werden, um den injizierten Fehler zu kontrollieren. Bei Syntax- und Semantikfehlern oder der Anpassung von Grenzwerten wurden die injizierten Fehler durch den bei der Entwicklung angewendeten defensiven

Programmieransatz bei der Testbankerstellung aufgedeckt. Dies wird jedoch nicht als Voraussetzung für eine übliche Entwicklung angesehen. Die Fehlerinjektion kommt zu dem Ergebnis, dass bei der FPGA-Entwicklung neben dem automatisierten Testen die Anwendung von V&V erforderlich ist, um möglichst viele potenzielle Fehlermodi zu abzudecken.

5.4.9 Fehlerinjektion in der Literatur (VTT Modell)

Neben der Codemanipulation gibt es diverse andere Möglichkeiten, logische Schaltungen und Software im Allgemeinen zu erproben und unentdeckte Fehler durch Injektion realistischer postulierter Fehler zu ermitteln. Im Folgenden wird das VTT Modell beschrieben, dass für die Fehlerinjektion genutzt werden kann. Hierbei handelt es sich um eine Testbank, die eine Vielzahl an Möglichkeiten zur Fehlerinjektion anbietet, die speziell für kerntechnische Anwendungen entwickelt wurden [45], [46].

Bei diesem Modell werden zunächst allgemein alle unentdeckten Fehler als Anomalien bezeichnet. Diese Anomalien können den Zustand des Programms kurzfristig durch einen Durchlauffehler („transient fault“) oder einen permanenten Fehler („permanent fault“) ändern. Diese theoretischen Fehler werden als implizit vorhanden angenommen und werden durch eine bestimmte Bedingung hervorgerufen. Ziel des Modells ist, immer die Anomalien aufzudecken. Zu den Aufdeckungsmethoden zählt eine gezielte Fehlerinjektion. Die Fehler können dabei in den Code (Variablen ändern, Eingangsgrößen ändern, Dimensionen ändern), in den Eingangskomponenten (Eingabegeräte wechseln, Ströme ändern), durch Testbänke oder durch eine bewusste Manipulation der Datenstruktur injiziert werden.

Zur Veranschaulichung der Vorgehensweise werden zwei der Methoden des VTT Modells erläutert.

1. Extended Propagation Analysis: Bei dieser Vorgehensweise werden Fehler direkt in den Code injiziert und das Verhalten des Programms mithilfe der VTT-Testbank analysiert.
2. Adaptive Vulnerability Analysis: Für diese Vorgehensweise wird ein kommerzielles Werkzeug „Fault injection security tool“ zur Fehlerinjektion genutzt. Dabei sollen Sicherheitsschwachstellen mithilfe der VTT-Testbänke, die für spezielle Anwendungen bzw. Industrievorhaben vorprogrammiert sind, aufgedeckt werden.

Bei beiden Vorgehensweisen steht die Analyse des Verhaltens des FPGA im Vordergrund. Dabei wird z.B. das Zeitverhalten oder die Kompilationsfähigkeit untersucht. Diese Methoden der Fehlerinjektion sind jedoch nicht in der Lage, die Anforderungsspezifikation auf Korrektheit zu prüfen. Das VTT-Modell wird als ergänzende Methode betrachtet, um Schwachstellen (sog. „weak spots“) im Programm aufzudecken. Die Methode der Fehlerinjektion mittels VTT-Modell wird als „Debugging“ bezeichnet und beschreibt, dass bekannte Fehler in einer Datenbank systematisch eingearbeitet und abgearbeitet werden. Es fällt auf, dass hierbei eine Vielzahl von kommerziellen Werkzeugen vorhanden sind, welche wiederum jeweils eigene Datenbanken heranziehen und somit eine unterschiedliche Art der Fehlerinjektion betreiben. Jedes dieser kommerziellen Werkzeuge ist eigens für einen bestimmten Prozess entwickelt worden.

Im Rahmen des Forschungsvorhabens wurde von einem Erwerb dieser Werkzeuge zur Fehlerinjektion abgesehen.

6 ANALYSE DER KOMPLEXITÄT VON FPGA DESIGNS

Die Methodik zur Komplexitätsmessung (siehe Kapitel 3.4) wird im Folgenden auf die Netzlisten der in Kapitel 4.5 beschriebenen FPGA-Designs angewandt. Die Komplexitätsmessung wird dabei anhand des modifizierten Komplexitätsvektors (siehe Tabelle 5) durchgeführt. Es wird aufgezeigt, wie die Komplexität in die Testfallgenerierung einfließt. Da die FPGA-Designs eigens entwickelt wurden, können Auswirkungen von Änderungen (z.B. funktional) untersucht werden.

6.1 Bestimmung der Komplexität

Die FPGA-Designs zur Prüfung der Plausibilität von Eingangswerten (siehe Kapitel 4.5.1) und zur 2-von-3-Auswahlschaltung (siehe Kapitel 4.5.2) wurden mit der automatisierten Komplexitätsmessung durch das Einlesen der jeweiligen EDIF-Netzlisten ausgewertet. Die ermittelten Komplexitätsvektoren sind in Tabelle 9 dargestellt. Für das FPGA-Design zur Redundanzschaltung konnte die EDIF-Netzliste nicht erfolgreich generiert und in das Werkzeug eingelesen werden, da die Berechnungszeit des Werkzeugs trotz umfangreicher Anpassungen mehr als zwei Wochen in Anspruch nahm. Die Ursache hierfür liegt in der Größe der Netzliste, die mit mehr als 53.000 Zeilen deutlich umfangreicher als übliche Netzlisten (bis zu 1.000 Zeilen) ist. Der Grund für die enorm umfangreiche Netzliste ist jedoch nicht ausschließlich auf die Komplexität der Redundanzschaltung zurückzuführen, sondern auch die zusätzlichen Rechenoperationen für das Umwandeln der Signale vom Datentyp „float“ zu „int“ (siehe Kapitel 4.5.3). Dies führt bei der Generierung der EDIF-Netzliste zu einer starken Verschachtelung (bzw. Verzweigung) der Signale (bzw. Verbindungspunkte). Neben der Größe der Netzliste wirkt sich dieser Umstand entscheidend auf die Berechnungszeit aus. Während bei der Generierung der Netzliste lediglich die einzelnen Knoten und ihre Quellen aufgelistet werden, muss bei der Auswertung durch das Programm zur Komplexitätsmessung jede mögliche Verbindung (zwischen zwei Knoten) der Netzliste über die vorhandenen Zweige einzeln ermittelt werden, um daraus eine auswertbare Verbindungsliste zu generieren. Trotz umfassender Arbeiten zur Optimierung des Werkzeugs sowie vereinfachende Anpassungen des Designs und der Netzliste ließ sich die Berechnungszeit für die Auswertung der Komplexität der Redundanzschaltung nicht entscheidend verkürzen. Für eine praxistaugliche Auswertung solcher (zugegebenermaßen künstlich erzeugten) übergroßen Netzlisten wäre eine komplett andere Herangehensweise bei der Auswertung der Komplexität erforderlich.

Tabelle 9: Komplexitätsvektoren der FPGA-Designs

ID	Eingangsprüfung	Auswahlschaltung
K_f1:	299	705
K_f2:	35	104
K_f3:	37	14
K_f4:	1260	3373

ID	Eingangsprüfung	Auswahlschaltung
K_f5:	0	0
K_f6:	0	0
K_f7:	24,9	10,8
K_f8:	124,6	86,2
K_f9:	99,7	75,4

Beide FPGA-Designs kommen auf demselben Baustein zum Einsatz. Bei der Gegenüberstellung der Komplexitätsvektoren fällt jedoch auf, dass bei der Auswahlschaltung trotz der höheren Anzahl von verwendeten Blöcken und Verbindungen, verglichen mit der Eingangswertprüfung, niedrigere Werte für die Verflechtungsmaße ermittelt wurden. Dies lässt sich nur zum Teil dadurch erklären, dass bei der Berechnung der Verflechtung V(FP) die Anzahl der Basisblöcke im Nenner des Verflechtungsmaßes im Vergleich etwas größer ist, und nicht in demselben Maße durch die höhere Anzahl von Ein- und Ausgängen im Zähler des Verflechtungsmaßes kompensiert wird. Die Gegenüberstellung zeigt, dass sich trotz der geringen Unterschiede bezüglich der Anzahl der Blöcke, der Ein- und Ausgangssignale und der Verbindungen für die Eingangswertprüfung größere Werte der Verflechtungsmaße ergeben als für die Auswahlschaltung. Dies ist insofern unerwartet, da die Auswahlschaltung im Vergleich zur Eingangswertprüfung eine scheinbar komplexere Funktionalität aufweist. Bei genauerer Betrachtung zeigt sich, dass die Mächtigkeit der Vorbereiche der jeweiligen Blöcke aufgrund der unterschiedlichen Verschaltung für die Eingangswertprüfung im Vergleich zur Auswahlschaltung im Mittel höher ist und sich somit für die Eingangswertprüfung eine komplexere Verflechtung ergibt.

Die Auswertung der Komplexitätsvektoren erlaubt drei mögliche Schlussfolgerungen:

1. Bei der Eingangswertprüfung wurde eine vergleichsweise einfache Funktion trotz der erfolgten werkzeugbasierten Optimierungen (in der Vivado Tool Suite) mit einer scheinbar unnötig hohen Verschachtelung realisiert.

2. Bei der Umsetzung der Auswahlschaltung auf das FPGA konnte eine vergleichsweise effiziente Verschaltung erzielt werden.

3. Die beiden erstgenannten Folgerungen sind gleichermaßen zutreffend.

Die Auswertung der Komplexitätsvektoren unter Berücksichtigung der Erfahrungen aus vorherigen Arbeiten zeigt, dass die Werte für die Verflechtungsmaße nicht stark voneinander abweichen. Dass sich für das FPGA-Design mit der scheinbar komplexeren Funktion der Auswahlschaltung im Vergleich zur Eingangswertprüfung höhere Verflechtungsmaße ergeben, ist dennoch unerwartet. Aufgrund der etwas komplexeren Funktionsweise der Auswahlschaltung wurde bei der Entwicklung, dem Testen und auch der Optimierung im Vergleich zur Eingangswertprüfung ein größerer Aufwand betrieben. In jedem Fall zeigt die

Auswertung der Komplexität, dass für die Entwicklung des FPGA-Designs zur Eingangswertprüfung weiteres Potential zur Optimierung besteht.

6.2 Berücksichtigung der Komplexität zur Generierung von Testfällen

Der Zusammenhang zwischen der Komplexität und der Zuverlässigkeit eines FPGA-Designs lässt sich folgendermaßen darstellen. Entscheidend ist zunächst einmal, dass ein Fehler im FPGA-Design vorhanden ist, der durch versagensauslösende Eingangsdaten zur Wirkung gebracht wird (der Fehler wird „getriggert“). Dabei steigt die Wahrscheinlichkeit für ein Versagen (bzw. es sinkt die Zuverlässigkeit) mit der Anzahl der Fehler im FPGA-Design. Um die Menge der Fehler zu minimieren, werden Qualitätssicherungsmaßnahmen, wie z.B. Verifizierungs- und Validierungsverfahren angewendet. Dabei beinhaltet die Validierung umfangreiche Funktionstests. Während für einfache Schaltungen nahezu alle möglichen Ausführungspfade getestet werden können, ist dies für komplexere Strukturen jedoch kaum bzw. gar nicht möglich. Je komplexer das FPGA-Design, desto geringer ist die erreichbare Testabdeckung [6]. Somit steigt mit der Komplexität des FPGA-Designs die Möglichkeit, dass Fehler, die eventuell enthalten sind durch die angewandten Qualitätssicherungsmaßnahmen und Tests unentdeckt bleiben und damit die Wahrscheinlichkeit eines potenziellen Versagens. Das heißt, mit steigender Komplexität sinkt potenziell die Zuverlässigkeit. Unter der Prämisse, dass für komplexere Bereiche der FPGA-Netzliste eine niedrigere Testabdeckung erreicht wird, werden diese komplexeren Bereiche des Designs mit den Werkzeugen zur Komplexitätsmessung [9] gezielt ermittelt und verstärkt getestet. Die Zielsetzung ist so eine möglichst optimale Testabdeckung zu erreichen.

Die durch die Komplexitätsmessung ermittelte Maßzahl $V_{intern}(FP)$ der inneren Verflechtung kann als eine Charakteristik der inneren Komplexität der FPGA-Netzliste verstanden werden [9]. Ausschlaggebend für den Beitrag der internen Verflechtung $V_{intern}(FP)$ ist die Mächtigkeit der Vorbereiche einer FPGA-Netzliste. Somit ergibt sich vereinfacht ausgedrückt für Vorbereiche innerhalb der FPGA-Netzliste, die eine größere Mächtigkeit aufweisen, eine größere Komplexität (bzw. innere Verflechtung), weshalb für diese Bereiche gezielt Testfälle generiert werden können, um die Testabdeckung zu optimieren. Im Folgenden werden die Ergebnisse des komplexitätsbasierten Testens exemplarisch am FPGA-Design der Auswahlschaltung dargestellt.

Bei der Analyse der Vorbereiche wurde festgestellt, dass die sogenannten „CARRY8“-Zellen des FPGA-Designs zur Auswahlschaltung verglichen mit den vor- und parallelgeschalteten Zellen eine beträchtlich höhere Mächtigkeit der Vorbereiche aufweisen. Diese „CARRY8“-Zellen beinhalten zudem eine große Anzahl an Ein- und Ausgangssignalen, womit eine vergleichsweise breite Testabdeckung erzielt werden kann. Für das Testen werden die sechs „CARRY8“-Zellen untersucht, welche zur Bestimmung einer Grenzwertüberschreitung des ersten Sensorsignales verantwortlich sind. Für die anderen beiden Sensorsignale liegt dieselbe Implementierung mit jeweils sechs identischen „CARRY8“-Zellen vor, womit die Resultate aus dem ersten Signal auch auf die anderen beiden Signale übertragbar sind. Die sechs „CARRY8“-Zellen werden zum besseren Verständnis von 1 bis 6 durchnummeriert, wobei in diesem Design kleiner nummerierte Zellen näher am Ende des Designabschnittes liegen und somit mehr Vorbereiche aufweisen (siehe Abbildung 24).

Zum Testen der sechs FPGA-Zellen wird die Vivado-Simulation als Testumgebung verwendet. Zusätzlich zur üblichen Simulation inklusive der zugehörigen Testfälle werden bei den sechs

untersuchten Zellen die Bits ihrer jeweiligen Ein- und Ausgabesignale einzeln geändert. Der Grundgedanke dahinter ist die Simulation von Ereignissen, bei denen während des Betriebs radioaktive Strahlung zu einem Wechsel von einem einzelnen Bit führt. Die Auswirkungen dieser Fehler konnten direkt mit der Testumgebung untersucht werden.

Zunächst werden die Zellen 1 und 2 betrachtet, die sich am Ende des Designabschnittes befinden. Das Ausgabesignal der Zelle 1 gibt an, ob eine Grenzwertüberschreitung vorliegt oder nicht. Bei normalem Betrieb übernimmt sie dafür das Ausgabesignal von Zelle 2, welche direkt vor Zelle 1 geschaltet ist. Beim Testen hat die Änderung des Ausgabesignals von Zelle 1 als auch die Änderung vom ersten Eingabesignal (also Ausgabesignal von Zelle 2) erwartungsgemäß dazu geführt, dass Unterschreitungen des Grenzwerts fälschlich als Überschreitungen angegeben werden und umgekehrt. Die übrigen Eingangssignale haben im Falle einer Unterschreitung keine Veränderung des Resultates bewirkt. Hingegen wurde bei einer Überschreitung des Grenzwerts bei der Hälfte der Eingangssignale beobachtet, dass als Resultat fälschlich eine Unterschreitung ausgegeben wird. Auch bei Veränderungen der Eingabesignale von Zelle 2 konnten potenzielle Fehlerquellen und ihre Auswirkungen aufgedeckt werden. Bei Unterschreitung der Grenzwerte führen einige Signale zu einer fälschlichen Überschreitung, während bei einer Überschreitung einige Signale fälschlicherweise eine Unterschreitung bei den durchgeführten Testfällen zur Folge haben. Es wurde nicht explizit untersucht, welche Sensorwerte beziehungsweise deren Bits oder Bitkombinationen zu den jeweiligen Fehlerpotentialen führen. Die fälschlich verursachten Unter- und Überschreitungen scheinen aber nur bei Sensorsignalen nahe dem Grenzwert aufzutreten.

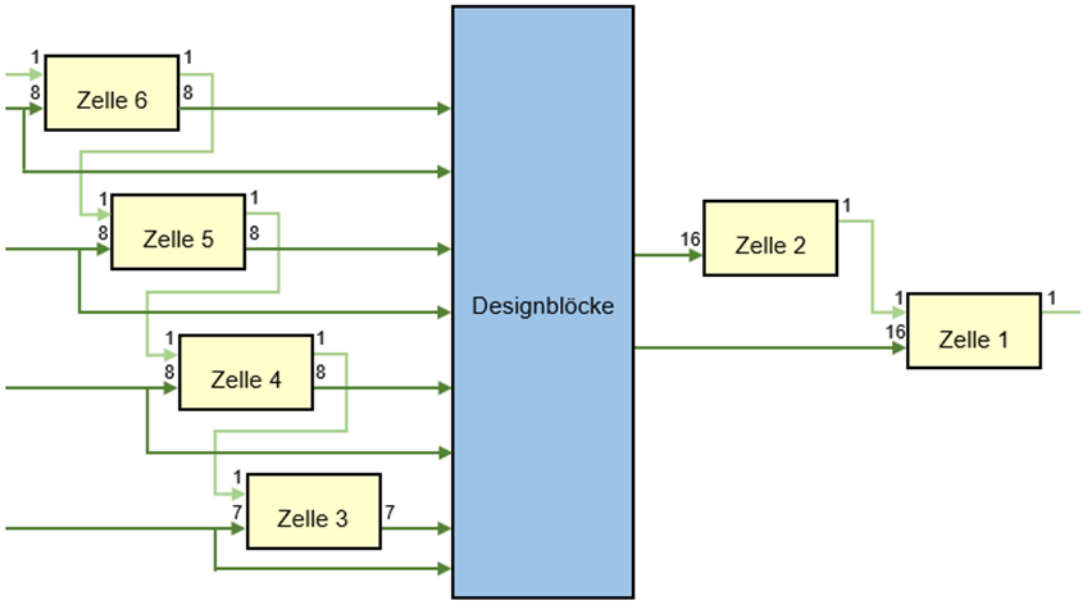


Abbildung 24: Untersuchte Zellen der Komplexitätsanalyse.

Nun werden die Zellen 3 bis 6 betrachtet, die etwas weiter vor Ende des Designabschnittes platziert sind. Die Ausgaben dieser vier Zellen werden von Blöcken des Designs eingelesen, welche im Anschluss Signale an die Zellen 1 und 2 ausgeben. Beim Testen dieser vier Zellen

hat kein einziges ihrer Ausgabesignale eine Auswirkung auf das Grenzwertsignal gehabt. Die Auswirkung von mehreren veränderten Ausgabesignalen wurde nicht untersucht, da die Wahrscheinlichkeit eines solchen durch Strahlung verursachten Fehlers als vernachlässigbar erachtet wird. Es wurden hingegeben unerwartete Auswirkungen bei Veränderungen der Eingangssignale beobachtet. Die Eingangssignale können sowohl die Ausgangssignale ihrer jeweiligen Zellen als auch das Grenzwertsignal ändern. Es stellt sich heraus, dass jedes der Eingangssignale zusätzlich von zwei der oben genannten Basisblöcke eingelesen wird. Ein Fehler in diesen Eingangssignalen führt zu Fehlern bei den Basisblöcken, welche im Anschluss von Zelle 1 oder 2 eingelesen werden und beeinflussen somit die Korrektheit des Grenzwertsignals. Wie bereits bei den ersten beiden Zellen beobachtet, führen einige Eingangssignale bei Unterschreitung der Grenzwerte zu einer fälschlichen Überschreitung des Grenzwertsignals im FPGA-Design. Abweichungen bei einer Überschreitung der Eingangssignale konnten nur für die Zellen 3 und 6 beobachtet werden.

Zusammengefasst existieren unter den 100 analysierten Signalen bei Unterschreitung des Grenzwerts zwischen 27 bis 31 und bei dessen Überschreitung zwischen 20 bis 24 potenzielle Fehlerquellen (je nach postuliertem Fehlerszenario) für das erste Grenzwertsignal des FPGA-Designs. Trifft ein solcher Fehler auf, so ändert er immer das zugehörige Warnsignal. Für den Fall, dass das erste Sensorsignal den Grenzwert unterschreitet und genau ein anderes Sensorsignal diesen überschreitet, kann dies zu einem fälschlich ausgelösten Tripsignal führen. Überschreitet das erste und genau ein anderes Sensorsignal den Grenzwert, kann umgekehrt ein notwendiges Tripsignal ausbleiben.

Durch komplexitätsbasiertes Testen wurde aufgezeigt, dass sich Einzelsignalfehler auf Logikzellen mit größeren Vorbereichen stärker auswirken. Somit konnte nachgewiesen werden, dass das Fehlerpotential von FPGA-Logikzellen mit der Mächtigkeit ihrer Vorbereiche zunimmt. Insbesondere sollte hier das Risiko hervorgehoben werden, dass das Fehlerpotential bei Eingangswerten nahe an den Grenzwerten steigt. Das hier untersuchte Fehlerrisiko sollte für FPGA-Designs allgemein betrachtet und mit gezielten komplexitätsbasierten Testfällen berücksichtigt werden.

6.3 Untersuchung der Auswirkung von Änderungen auf die Komplexität von FPGA-Anwendungen

Um die Auswirkung der Funktionalität einer FPGA-Anwendung auf deren Komplexität zu untersuchen, werden an zwei der entwickelten FPGA-Designs Änderungen vorgenommen und anschließend die Komplexität der somit veränderten EDIF-Netzliste ausgewertet.

Es werden im C++ Quellcode der jeweiligen Designvarianten Änderungen an den Grenzwerten vorgenommen. Auf Grundlage des veränderten Codes werden die darauffolgenden Arbeitsschritte in Vitis HLS und Vivado Design Suite ausgeführt, um im Anschluss die EDIF-Dateien der Designs zu generieren. Aus den EDIF-Dateien werden mithilfe des Werkzeugs zur Komplexitätsmessung die jeweiligen Komplexitätsvektoren berechnet, welche auf Unterschiede zu den ursprünglich generierten Komplexitätsvektoren untersucht werden. Hierfür wurden die FPGA-Designs zur Eingangsprüfung und Auswahlschaltung herangezogen. Die Redundanzschaltung wurde nicht in diese Untersuchung einbezogen.

Für das FPGA-Design zur Auswahlschaltung wurden zusätzliche Designvarianten erstellt, bei denen zu dem bereits existierenden oberen Grenzwert zusätzlich ein unterer Grenzwert hinzugefügt wurde. Es wurden hierbei drei Designvarianten mit den Grenzwerten 20, Null („0“)

und -20 betrachtet. Die Werte wurden so gewählt, damit positive und negative Grenzwerte als auch der Sonderfall des Grenzwerts Null abgedeckt werden. Die hierzu ermittelten Komplexitätsvektoren sind in der Tabelle 10 zusammengefasst.

Tabelle 10: Komplexitätsvektoren von Designvarianten der Auswahlschaltung

Designvariante	Anzahl FB	Verbindungen	V (FP)	$V(FP)_{mod}$	$V(FP)_{intern}$
Original	705	3373	10,8	86,2	75,4
Grenzwert 20	768	3718	10,8	97,2	86,4
Grenzwert 0	740	3241	2,1	52,4	50,4
Grenzwert -20	816	4003	10,8	103,3	92,5

Die Werte der Komplexitätsvektoren zeigen, dass die Varianten mit unteren Grenzwerten 20 und -20 eine höhere Ressourcennutzung und somit höhere Komplexitätswerte im Vergleich zum ursprünglichen FPGA-Design aufweisen. Die Ursache dafür ist, dass der hinzugefügte Grenzwertvergleich zusätzliche logische Operationen im Design erfordert und sich somit die Komplexität erhöht. Auf Grundlage der Ergebnisse wird außerdem gefolgert, dass logische Operationen (oder zumindest Vergleiche) im negativen Wertebereich im Vergleich zum positiven Wertbereich zusätzliche Ressourcen erfordern, was mit einer Erhöhung der Komplexität einhergeht.

Bei der Designvariante mit Grenzwert Null haben sich im Vergleich zu den anderen Designs die größten Unterschiede ergeben. Während bei den übrigen beiden veränderten Designvarianten die Werte des Komplexitätsvektors im Vergleich zum Original angestiegen sind, sind hier die Werte des Verflechtungsmaßes V(FP) deutlich gesunken. Eine nachfolgend durchgeführte Analyse der Vorbereiche und Schaltpläne in Vivado hat ergeben, dass zur Berechnung von bestimmten Ausgangssignalen für das ursprüngliche FPGA-Design nicht erforderliche logische Verbindungen zu existieren scheinen. Bei den FPGA-Designs werden die Signale für die Überschreitung der oberen und unteren Grenzwerte in die Berechnung der Ausgabewerte einbezogen. Über die Vivado-Simulation zeigte sich jedoch, dass die Signale zur Grenzwertüberschreitung keinen Einfluss auf die betrachteten Ausgangssignale haben. Entsprechend weisen die Ausgabesignale bei dem ursprünglichen Design und bei den Designvarianten mit den Grenzwerten 20 und -20 Vorbereiche in der Größenordnung von 800 oder 900 auf, während bei der Designvariante mit Grenzwert Null dieselben Ausgangssignale Vorbereiche von 12 bzw. 14 aufweisen. Bemerkenswerterweise enthalten alle Designvarianten bei der Elaboration diese nicht erforderlichen Verbindungen. Bei der Designvariante mit Grenzwert Null werden im Gegensatz zu den anderen Varianten die nicht benötigten Verbindungen bei der Synthese herausoptimiert. Anscheinend führt der hinzugefügte Grenzwert Null in Vivado zu einer verbesserten Optimierung des Designs.

Als weiteren Unterschied zu den anderen Varianten werden im FPGA-Design in dem Bereich, in dem die Grenzwertvergleiche erfolgen, keine logischen Operationen für den

Grenzwertvergleich mit Null hinzugefügt. Vitis HLS ist in diesem Fall offenbar in der Lage, das Programm vorab zu optimieren.

Für das FPGA-Design zur Eingangsprüfung wurden für die zusätzlichen Designvarianten die unteren und oberen Grenzwerte verändert. Hierbei wurden für die untere Grenze (ursprünglicher Grenzwert 50) die Grenzwerte -20, Null („0“), 20 und 60 verwendet. Für die obere Grenze (ursprünglicher Grenzwert 100) wurden 80 und 120 ausgewählt. Die zugehörigen Komplexitätsvektoren sind in Tabelle 11 zusammengefasst.

Tabelle 11: Komplexitätsvektoren von Designvarianten der Eingangsprüfung

Designvariante	Anzahl FB	Verbindungen	V (FP)	$V(FP)_{mod}$	$V(FP)_{intern}$
Original	299	1260	24,9	124,7	99,7
Grenzwert -20	300	1285	24,9	120,4	95,5
Grenzwert 0	281	1228	23,9	109,2	85,3
Grenzwert 20	307	1314	25,1	125,6	100,5
Grenzwert 60	311	1347	25,2	125,9	100,7
Grenzwert 80	297	1251	24,9	123,9	99,0
Grenzwert 120	303	1286	25,0	125,0	100,0

Wie schon bei der Betrachtung der veränderten Designvarianten zur Auswahlschaltung sind die Werte des Komplexitätsvektors bei der Designvariante mit Grenzwert Null geringer als bei allen anderen Designvarianten zur Eingangsprüfung. Allerdings fallen für das Design zur Eingangsprüfung die Unterschiede der Komplexitätsvektoren deutlich geringer aus. Die nachfolgend durchgeführte Analyse zeigte, dass für die Designvariante mit Grenzwert Null wieder logische Operationen entfallen. Hier wirkt sich offenbar nur aus, dass der Vergleich mit Null schaltungstechnisch einfacher ist, da bei Zweierkomplementdarstellung nur ein Bit verglichen werden muss. Das bei den Designvarianten der Eingangsprüfung für den Grenzwert Null eine bessere Optimierung gelingt, ist aufgrund der dargestellten Ergebnisse nicht erkennbar.

Bei Betrachtung der Ergebnisse der Designvarianten mit positiven Grenzwerten sind nur geringfügige Unterschiede zwischen den Komplexitätsvektoren zu vermerken. Während der Elaboration sind die HDL-Programme der Varianten bis auf den genauen Zahlenwert beim Grenzwertvergleich und den Namen der Variablen identisch zueinander (mit Ausnahme der Variante mit Grenzwert Null). Die verschiedenen Zahlenwerte führen anscheinend zu keiner

wesentlichen Änderung der Logik. Es gibt lediglich bei der Synthese und Implementierung in Vivado leichte Unterschiede bei der Anzahl von verwendeten Funktionsbausteinen.

7 POTENZIELLE KRITERIEN FÜR DIE QUALIFIZIERUNG VON PROGRAMMIERBARER LOGIK

Zur Ableitung von potenziellen Kriterien für die Qualifizierung programmierbarer Logik werden die im Vorhaben erzielten Erkenntnisse im Kontext bestehender Regelwerksanforderungen betrachtet. Hierfür sind die Anforderungen der IEC 62566 [4] für die Entwicklung HDL-programmierter integrierter Schaltkreise für Systeme, die Funktionen der Kategorie A ausführen, sowie die Anforderungen der IEC 60880 [36] für Softwareaspekte für rechnerbasierte Systeme zur Realisierung von Funktionen der Kategorie A maßgeblich.

Die Ergebnisse der Untersuchung der Auswirkung von Änderungen auf die Komplexität von FPGA-Anwendungen hat gezeigt, dass die werkzeugbasierten Optimierungen hinsichtlich des Entfernens nicht erforderlicher Verbindungen auf Netzlistenebene keine verlässliche Reduzierung der Komplexität erzielt. Es scheint, dass bei der werkzeugbasierten Optimierung der Fokus auf der Umsetzung des Entwurfs auf die Zielhardware unter Einhaltung gewisser Ressourcen- und Zeitvorgaben liegt. Die Optimierung scheint jedoch nicht dafür vorgesehen zu sein eine Reduktion auf notwendige Signalverbindungen bzw. eine Verringerung der Komplexität des Entwurfs zu bewirken. Die eingangs erwähnten Regelwerke beinhalten hinsichtlich der Optimierung bereits Anforderungen. Diese betreffen in IEC 62566, 5.3.1.2 einen Verweis auf IEC 60880, 5.4 zur Dokumentation automatisierter Vorgänge, welche auch die Optimierung von Softwarewerkzeugen umfassen. Gemäß IEC 62566, 5.5.2 müssen die Parameter für diese Vorgänge dem Konfigurationsmanagement unterliegen. Weiterhin dürfen gemäß IEC 62566, 8.4.5.4 durch die Optimierung keine Fehlerentdeckungs- und Toleranzmechanismen entfernt werden. Gemäß IEC 60880, 8.2.3.1.1.3 dürfen Softwaremodule keine ungeplanten Funktionen ausführen. Eine gleichwertige Anforderung für den Funktionsumfang von programmierbarer Logik ist in IEC 62566 nicht formuliert. Eine solche Anforderung für programmierbare Logik wäre schwer umsetzbar, weil laut IEC 62566, B.4.4 die komplexen und unterschiedlichen Strukturen von FPGAs es erschweren vorherzusagen, wie viele Blöcke für eine gegebene Funktion benötigt werden. Auf Grundlage der im Vorhaben erzielten Erkenntnisse könnte für die Qualifizierung programmierbarer Logik zumindest eine Empfehlung formuliert werden, welche die werkzeugbasierte Optimierung hinsichtlich notwendiger Signalverbindungen auf Netzlistenebene nahelegt.

Um für das Testen der FPGA-Designs eine möglichst optimierte Testabdeckung zu erreichen, wurden im Vorhaben komplexere Bereiche auf den FPGAs auf Netzlistenebene mittels eines eigens erstellten Werkzeugs zur Bestimmung der Komplexität ermittelt und anschließend gezielt getestet. IEC 62566 formuliert Empfehlungen hinsichtlich der Komplexität. Diese umfassen eine Vermeidung von unnötiger Komplexität bei der RTL-Beschreibung programmierbarer Logik (IEC 62566, 8.5.1), sowie ein ausgeglichenes Verhältnis zwischen defensiver Auslegung und der damit verbundenen zusätzlichen Komplexität des Entwurfs (IEC 62566, 6.4.2). Im Vorhaben konnte durch komplexitätsbasiertes Testen nachgewiesen werden, dass das Fehlerpotential von FPGA-Logikzellen mit der Mächtigkeit ihrer Vorbereiche zunimmt und das Fehlerpotential bei Eingangswerten nahe an den Grenzwerten steigt. Die Ermittlung der komplexeren Bereiche auf Netzlistenebene wurden im Vorhaben mittels eines eigenen, nicht frei verfügbaren Werkzeugs ermittelt. Diese komplexeren Bereiche können jedoch alternativ auch bei der Vivado-Analyse (siehe Kapitel 5.2.6) identifiziert werden. Es ist

davon auszugehen, dass andere Entwicklungswerkzeuge ähnliche Möglichkeiten der Analyse auf Netzlistenebene anbieten. Für die Qualifizierung programmierbarer Logik kann potenziell eine Empfehlung zu gezielten komplexitätsbasierten Tests des Logikentwurfs formuliert werden, um das beschriebene Fehlerrisiko zu minimieren. Da die Mächtigkeit der Vorbereiche bzw. die Komplexität der Verschaltung auf Netzlistenebene immer relativ zum jeweiligen FPGA-Design ist, könnte eine solche Empfehlung nur qualitativ formuliert werden.

Die im Vorhaben erzielten Erkenntnisse zeigen, dass mittels Fehlerinjektion postulierter Fehler Testlücken erkannt werden konnten, die sich mitunter nur durch manuelle Verifizierung und Validierung (V&V) kontrollieren lassen. Postulierte Fehler wie die Verwendung unzulässiger Werte oder Fehler bei der Präzision der Eingabewerte können durch automatisierte Test- und Verifizierungsmethoden möglicherweise nicht vollständig erkannt werden. Bei Syntax- und Semantikfehlern oder der Anpassung von Grenzwerten wurden die injizierten Fehler durch den bei der Entwicklung angewendeten defensiven Programmieransatz bei der Testbankerstellung aufgedeckt. Die defensive Auslegung ist in IEC 62566, 6.4 zu Fehlerentdeckung und Fehlertoleranz sowie in IEC 62566, 8.3.2 direkt berücksichtigt. Die Wirksamkeit der angewendeten defensiven Auslegung ist allerdings nur schwer prüfbar. Es sei an dieser Stelle erwähnt, dass der TÜV Rheinland Industrie Service bei Zertifizierungen nach IEC 61508 [47] üblicherweise Fehlerinjektionstests als zusätzliche vertrauensbildende Maßnahme durchführt, obwohl dies normativ nicht ausdrücklich gefordert ist. Für die Qualifizierung programmierbarer Logik kann potenziell eine Empfehlung zur Durchführung von codebasierter Fehlerinjektion formuliert werden, um die Wirksamkeit der durchgeführten Test- und Verifizierungsmaßnahmen festzustellen und möglichst viele potenzielle Fehler aufzudecken.

In Bezug auf die Unabhängigkeit der Verifizierung programmierbarer Logik fokussiert sich IEC 62566, 9 auf personelle Aspekte und nicht explizit auf die Unabhängigkeit der Werkzeuge, die für Entwicklung und Testen zum Einsatz kommen. Eine Anforderung hinsichtlich der Unabhängigkeit zwischen Entwicklungs- und Testwerkzeugen ist hingegen nicht ausdrücklich formuliert. Da die Entwicklung von programmierbarer Logik zumeist werkzeugbasiert erfolgt und im Lieferumfang der Entwicklungswerkzeuge umfassende Test- und Verifizierungsmöglichkeiten enthalten sind, scheint der Einsatz zusätzlicher, von der Entwicklung unabhängiger Test-Werkzeuge oder Verfahren im Hinblick auf die bestehenden Regelwerke nicht unerheblich zu sein. Es besteht das Risiko, dass bei den automatisierten werkzeugbasierten Entwicklungsschritten wie z.B. den Synthese- und Optimierungsprozessen systematische Fehler auftreten, die von den Werkzeugen selbst in den anschließend durchgeführten Verifizierungen unerkannt bleiben. Dieses Risiko lässt sich durch eine unabhängige Verifizierung in Bezug auf die verwendeten Testwerkzeuge minimieren. Dies ist indirekt in IEC 60880 zur Vermeidung von Fehlern gemeinsamer Ursache für Software beschrieben (IEC 60880, 13.1.1). Die Forderung der unabhängigen Verifizierung (insbesondere hinsichtlich der automatisierten Testdurchführung) in Bezug auf die verwendeten Entwicklungswerkzeuge stellt das augenscheinlichste potenzielle Kriterium für die Qualifizierung programmierbarer Logik dar.

8 ZUSAMMENFASSUNG

Moderne digitale Leittechniksysteme, die durch eine grafikbasierte Spezifikation programmiert werden, weisen eine ähnliche Struktur wie FPGA-basierte Logik auf, die durch Verschaltung von Logikzellen konfiguriert wird. Für die Beschreibung und Quantifizierung ihrer Komplexität kann daher ein einheitliches Konzept verwendet werden, das in früheren Forschungsvorhaben [5] ausgearbeitet und für die Bewertung von FPGAs modifiziert wurde [9].

Im Vorhaben wurden eigene FPGA-Designs erstellt, die repräsentativ für sicherheitsrelevante Anwendungen der Leittechnik in KKW sind. Durch die Eigenentwicklung von FPGA-Designs wurden Einblicke in gängige Entwicklungswerkzeuge, darin enthaltene Test- und Simulationsumgebungen und deren Funktionsweise gewonnen. Bei der Eigenentwicklung ist aufgefallen, dass die FPGA-Entwicklungswerkzeuge auch für eine Vielzahl von Test-, Simulations- und Analysefunktionen verwendet werden können. Diese umfangreichen Verifizierungs- und Testfunktionen weisen dabei nicht den von kerntechnischen Normen geforderten Grad der Unabhängigkeit von der Entwicklung auf. Werden die Tests mit den Entwicklungswerkzeugen durchgeführt, können keine verlässlichen Aussagen über die Korrektheit des verwendeten Compilers, der Funktionen zur Synthese, Analysieren oder Simulieren getroffen werden. Daher wurden in diesem Vorhaben die Tests der FPGA-Designs zusätzlich mit einer von der Entwicklung unabhängigen Testumgebung durchgeführt. Die Testergebnisse zeigten jedoch keine signifikanten Unterschiede zwischen den beiden verwendeten Testumgebungen. Allerdings führt die Verwendung eines HLS-Compilers mit einem nicht zugehörigen Synthese-Werkzeug zu gravierenden Fehlern beim Kompilieren, Analysieren und dem Platzieren. Die Testergebnisse legen den Schluss nahe, dass der mit dem HLS-Compiler generierte Code Verilog-Sprachfeatures nutzt oder Konstrukte generiert, die eventuell nicht von jedem Werkzeug synthetisierbar sind. Dies wirft die Frage auf, ob der HLS-Compiler in sicherheitskritischen Anwendungen überhaupt empfehlenswert ist. Für sicherheitskritische Anwendungen sollte der Compiler VHDL/Verilog-Code ausgeben, der von allen (bzw. zumindest von den meisten) Werkzeugen synthetisierbar ist. Das Problem hierbei ist, dass manche herstellerspezifischen Features auf dem FPGA-Chip nur durch spezielle werkzeugspezifische Sprachkonstrukte nutzbar sind. Um die Kompatibilität zu erhöhen, wäre es wünschenswert, wenn solche Sprachkonstrukte eingekapselt sind (z.B. in ein Verilog-Modul mit definierter Schnittstelle nach außen).

Bei der Generierung der Testfälle wurde insbesondere die Komplexität der entwickelten FPGA-Designs berücksichtigt. Je komplexer das FPGA-Design, desto geringer ist die erreichbare Testabdeckung und potenziell dessen Zuverlässigkeit [6]. Unter dieser Prämisse wurden komplexere Bereiche der FPGA-Netzlisten mit den Werkzeugen zur Komplexitätsmessung [9] gezielt ermittelt und verstärkt getestet. Damit wurde eine optimierte Testabdeckung erzielt. Durch komplexitätsbasiertes Testen wurde überdies nachgewiesen, dass das Fehlerpotential von FPGA-Logikzellen mit der Mächtigkeit ihrer Vorbereiche zunimmt. Das Fehlerrisiko sollte für FPGA-Designs allgemein betrachtet und mit gezielten komplexitätsbasierten Testfällen berücksichtigt werden.

Es wurde eine Untersuchung der Auswirkungen von Änderungen auf die Komplexität von FPGA-Anwendungen durchgeführt. Die vorgenommenen Änderungen an den Grenzwerten der jeweiligen Designvarianten zeigten Auswirkungen auf deren Logik und Komplexität. Eine Änderung des Grenzwertes auf Null („0“) hat bei dieser Untersuchung die deutlichsten Unterschiede aufgezeigt. Wie sich durch Analysen der Designvarianten zeigte, wurde in

diesem Fall die FPGA-Logik während der C-Simulation durch einen entsprechenden Grenzwertvergleich optimiert. Es zeigte sich, dass dies in der Folge zu einer weiteren Optimierung der Designlogik während der Vivado-Synthese beiträgt. Insbesondere bei dem FPGA-Design zur Auswahlschaltung konnte durch das Hinzufügen eines Grenzwerts bei Null eine deutliche Verringerung der Verflechtungsmaße festgestellt werden, was auf eine Verringerung der Komplexität des FPGA-Designs schließen lässt. Die deutliche Veränderung der Komplexität des FPGA-Designs ist insofern bemerkenswert, da hierbei die Abfrage eines Wertes hinzugefügt wurde. Eine detaillierte Analyse zeigte, dass die FPGA-Designs nicht erforderliche Verbindungen enthalten können, obwohl diese innerhalb der verwendeten Entwicklungsumgebung mehrfach optimiert, getestet und funktional validiert wurden. Diese Beobachtung legt den Schluss nahe, dass die FPGA-Entwicklungswerkzeuge nicht darauf ausgelegt sind die Komplexität der generierten FPGA-Anwendungen zu optimieren. Bei der Generierung von FPGA-Designs könnte das Einfügen oder Verändern von logischen Operationen mit dem Wert Null zu einer verbesserten Optimierung des FPGA-Designs beitragen und somit als zusätzliche Testmethode dienen.

Die Verwendung von Grenzwerten im negativen und positiven Wertebereich führte bei der Untersuchung der Auswirkungen von Änderungen auf die Komplexität von FPGA-Anwendungen zu weniger deutlichen, aber erkennbaren Veränderungen der Komplexität. Die Untersuchungsergebnisse legen nahe, dass FPGA-Designs bei Änderungen im negativen Wertebereich oder bei Vorzeichenwechsel auf ungewünschte Änderung der Logik bzw. Komplexität hin getestet werden sollten.

Basierend auf den Erfahrungen aus der Entwicklung von FPGA-Designs wurden mögliche Schwachstellen im Design und daraus resultierende potenzielle Quellen für Fehlermodi identifiziert. Diese wurden zur Generierung von Testfällen und zur Postulierung von potenziellen Fehlern für die Fehlerinjektion verwendet. Ziel war es zu untersuchen, welche Fehler die Testverfahren erkennen können. Die identifizierten potenziellen oder tatsächlichen Fehlermodi wurden hinsichtlich ihrer sicherheitstechnischen Auswirkungen klassifiziert.

Als Ergebnis der Analyse zu FPGA-Fehlermodi und insbesondere der Bewertung der Fehlerinjektionsszenarien wird die Anwendung von Verifizierung und Validierung (V&V) als entscheidend erachtet. Die Analysen zeigen, dass ein V&V-Prozess nicht vollständig automatisiert oder bedingungslos den mit einer FPGA-Entwicklungsumgebung gelieferten Testwerkzeugen überlassen werden kann. Wie sich bei der Analyse der Injektion postulierter Fehler zeigte, können bestimmte injizierte Fehler nur durch einen dedizierten V&V-Prozess erkannt und kontrolliert werden, da ausschließlich durch V&V einige der bewusst injizierten Fehler aufgedeckt werden konnten.

Für die Ableitung von Kriterien für die Qualifizierung von programmierbarer Logik wurden die gewonnenen Erkenntnisse des Vorhabens genutzt. Dabei wurden bestehende Testverfahren und aktuelle nationale und internationaler Regelwerke im Hinblick auf die Ergebnisse analysiert und Vorschläge zur möglichen Optimierung von Qualifizierungsanforderungen und -methoden anhand der Vorhabensergebnisse dargelegt. Die so abgeleiteten potenziellen Qualifizierungskriterien umfassen Empfehlungen zur werkzeugbasierten Optimierung des Logikentwurfs hinsichtlich einer Reduktion auf notwendige Signalverbindungen, zu komplexitätsbasierten Tests des Logikentwurfs und zur Durchführung von codebasierter Fehlerinjektion. Als weiteres Qualifizierungskriterium wurde eine Anforderung zu

unabhängigem Verifizieren und Testen der programmierbaren Logik in Bezug auf die zur Entwicklung der Logik verwendeten Werkzeuge abgeleitet.

Das in diesem Vorhaben erzielte Ergebnis zur Analyse der Fehlermodi von programmierbarer Logik umfasst mögliche Schwachstellen bei FPGA-Designs und daraus resultierende potenzielle Fehlerquellen, die bei der FPGA-Entwicklung zu berücksichtigen sind. Die Ergebnisse der Fehlerinjektion realistischer postulierter Fehler zeigen potenzielle Testlücken bei der Anwendung von FPGA-Testwerkzeugen auf. Mittels einer automatisierten Methodik zur Ermittlung der Komplexität von programmierbarer Logik wurden Einflussfaktoren wie Komplexität oder Funktionalität des FPGA-Designs auf das Fehlerpotential und die Testbarkeit untersucht. Die gewonnenen Erkenntnisse ermöglichen sowohl Herstellern programmierbarer Logik, ihre Test- und Verifizierungsverfahren von Sicherheitsleittechnik zu verbessern als auch Prüfern, diese objektiver zu bewerten.

9 REFERENZEN

[1] Kharchenko, V., S., Sklyar, V., V., (Ed.) FPGA-based NPP Instrumentation and Control Systems: Development and Safety Assessment, Kirovograd – Kharkiv – 2008.

[2] E.S. Bakhmach, A.D. Herasimanko, V.A. Golovyr, V.S. Kharchenko, Yu.V. Rozen, A.A: Siora, V.V. Sklyar, V.I. Tokarev, S.V. Vinogradskaya, M.A. Yastrebenetsky, FPGA-based NPP Instrumentation and Control Systems: Development and Safety Assessment, Kirovograd Kharkiv 2018

[3] Failure Modes Taxonomy for Reliability Assessment of Digital Instrumentation and Control Systems for Probabilistic Risk Analysis, Nuclear Safety, NEA/CSNI/R(2014)16, Februar 2015

[4] IEC 62566 Ed.1, Nuclear power plants - Instrumentation and control important to safety - Development of HDL-programmed integrated circuits for systems performing category A functions, Januar 2012

[5] Märtz, J., Miedl, H., Lindner, A., Gerst, C., Komplexitätsmessung der Software digitaler Leittechniksysteme
ISTec-A-1569
ISTec Garching, 2010

[6] Dependability Assessment of Software for Safety Instrumentation and Control Systems at Nuclear Power Plants, International Atomic Energy Agency, IAEA Nuclear Energy Series No. NP-T-3.27, 2018

[7] Märtz, J., Fachberatung zur Um- und Nachrüstung der Sicherheitsleittechnik in deutschen Kernkraftwerken Sicherheitstechnische Bewertung zum Einsatz und Betrieb rechnergestützter Sicherheitsleittechnik in deutschen Kernkraftwerken AP 3: Analyse und Bewertung der Komplexität der Software und Software-basierter Sicherheitsleittechnik
ISTec-A-1311
ISTec Garching, 2010

[8] Heigl, R., Miedl, H., Hellie, F., Schnürer, G., Analyse der Fehlermodi von programmierbaren logischen Schaltungen in der Sicherheitsleittechnik von Kernkraftwerken, Teilbericht
ISTec-A-2891, Garching, 2017

[9] Heigl, R., Hellie, F., Linder, A., Mölleken, A., Komplexität und Fehlerpotential bei softwarebasierter digitaler Sicherheitsleittechnik
ISTec-A-4010, Rev. 2, April 2021

[10] KTA 3501 Reaktorschutzsystem und Überwachungseinrichtungen des Sicherheitssystems, Fassung November 2015

[11] IEC 61226 Ed. 4, Nuclear power plants - Instrumentation, control and electrical power systems important to safety - Categorization of functions and classification of systems, April 2020

[12] IEC 61513 Ed. 2, Nuclear power plants - Instrumentation and control important to safety - General requirements for systems, August 2011

[13] Märtz, J., et.al., Complexity Measurement and Reliability of Software in Digital I&C-Systems. (Proceedings of the 4th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technology) Columbus, Ohio 2004

[14] Märtz, J., Gerst, C., Complexity and Structural Analysis
HARMONICS Deliverable (D-N°: 2.4), 2015

[15] P. McNelles, Z. Chang Zeng, G. Renganathan, M. Chirila, L. Lu, Failure mode taxonomy for assessing the reliability of Field Programmable Gate Array based Instrumentation and Control systems, April 2017

[16] IEC 63096 Ed. 1, Nuclear power plants - Instrumentation, control and electrical power systems - Security controls, Oktober 2020

[17] IEC 62645 Ed. 2, Nuclear power plants - Instrumentation, control and electrical power systems - Cybersecurity requirements, November 2019

[18] EPRI, IAEA Workshop on FPGAs, Oktober 2014

[19] P. McNelles, Z. Chang Zeng, G. Renganathan, G. Lamarre, Y. Akl, L. Lu, A Comparison of Fault Trees and the Dynamic Flowgraph Methodology for the Analysis of FPGA-based Safety Systems Part 1: Reactor Trip Logic Loop Reliability Analysis

[20] IAEA NUCLEAR ENERGY SERIES No. NR-T-3.31: Challenges and Approaches for Selecting, Assessing and Qualifying Commercial Industrial Digital Instrumentation and Control Equipment for Use in Nuclear Power Plant Applications, VIENNA, 2020

[21] IAEA NUCLEAR ENERGY SERIES No. NP-T-3.17: Application of Field Programmable Gate Arrays in Instrumentation and Control Systems of Nuclear Power Plants, VIENNA, 2016

[22] IAEA SAFETY STANDARDS SERIES No. SSG-39: Design of Instrumentation and Control Systems for Nuclear Power Plants, VIENNA, 2016

[23] VHDL, www.de.wikibooks.org, 26.08.2014

[24] Prof. Dr.-Ing. Michael Karagounis, Dipl.-Ing. Rolf Paulus, FH Dortmund, Tutorial Vivado/Verilog Teil1, Erste Schritte mit Verilog auf dem FPGA, Juni 2018

[25] John Moondanos, Strategic CAD Labs, INTEL Corp.&GSRC Visiting Fellow, UC Berkeley, SystemC Tutorial, Januar 2009

[26] National Instruments Corporation, Introduction to LabVIEW, September 2003 Edition

[27] Dipl.-Ing. U. Wohlfarth, Vorlesung: SIMULINK Grundlagen, Hochschule Augsburg, April 2007

[28] Mentor Graphics, ModelSim® Tutorial, Software Version 10.1a, 2012

[29] Actel Corporation, Mountain View, CA 94043, Libero IDE v9.1User's Guide,2010

[30] ALTERA, My First FPGA Design Tutorial, www.altera.com/literature/, Juli 2008

[31] Panel Discussion. Development Tools and Processes, 9th International Workshop on the Application of FPGA's in NPP's, Lyon, October 2016

[32] Mark Burzynski, NRC Topical Report Certification Process, Suggestions for Success, 9th International Workshop on the Application of FPGAs in NPPs, Lyon, October 2016

[33] P. McNelles, Z.C. Zeng, G. Renganathan, M. Chirila and L. Lu, Canadian Nuclear Safety Commission, Failure Modes Taxonomy: Assessing the Reliability of FPGA-Based I&C Systems, 9th International Workshop on the Application of FPGA's in NPP's, Lyon, October 2016

[34] Sergio Russomanno, COO, www.sunport.ch, A case for the adoption of FPGA technology in the implementation and replacement of equipment and systems in NPPs, 9th International Workshop on the Application of FPGA's in NPP's, Lyon, October 2016

[35] Alexander Wigg (alexander-john.wigg@edf.fr), Ludovic Pietre-Cambacedes (ludovic.pietre-cambacedes@edf.fr), EDF Nuclear Engineering Division, Basic Design (SEPTEN),FPGA-Based I&C Systems, Unraveling Myths from Reality, NPIC HMIT 2015

[36] IEC 60880 Ed. 2, Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions, Mai 2006

[37] Libero IDE v9.1 User's Guide
https://www.microsemi.com/document-portal/doc_view/130848-libero-ide-v9-1-user-s-guide?bcsi-ac-

947de7b86f0184a0=26532D3500000004ioNodBKyao4z0ATlC/8SDAXvqgpsAQAABAAAAPJEVQBAOAAAAAAAAEG0AAA=, [Online; Aufruf: 27.01.2021]

[38] Introduction to the Quartus® II Software https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/intro_to_quartus2.pdf, [Online; Aufruf: 27.01.2021]

[39] ISE In-Depth Tutorial https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ise_tutorial_ug695.pdf, [Online; Aufruf: 27.01.2021]

[40] Produktseite zu Vivado Design Suite https://www.xilinx.com/products/design-tools/vivado.html, [Online; Aufruf: 21.06.2021]

[41] Vivado Simulation Flow https://www.xilinx.com/products/design-tools/vivado/simulation.html, [Online; Aufruf: 21.06.2021]

[42] Introduction to Vitis HLS https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/introductionvitishls.html#dvj1582158621847, [Online; Aufruf: 21.06.2021]

[43] 7th International Workshop on Application of Field Programmable Gate Arrays in Nuclear Power Plants, Charlotte, North Carolina, October 2014

[44] Getting Started with Vitis HLS https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/yxj1602022623766.html [Online, Aufruf: 07.07.2021]

[45] https://sunport.ch/wp-content/uploads/2017/06/D2-4-VTT.pdf

[46] https://www.vttresearch.com/sites/default/files/pdf/publications/2001/P448.pdf

[47] IEC 61508 Ed. 2, Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements. May 2010

[48] Die Soft-Error-Rate von Submikrometer CMOS-Logikschaltungen, Dr.–Ing. Thomas Juhnke; Berlin 2003

[49] P. E. Dodd, M. R. Shaneyfelt, and F. W. Sexton, “Charge Collection and SEU from Angled Ion Strikes,” IEEE Trans. Nucl. Sci., vol. 44, no. 6, pp. 2256–2265, Dec. 1997

[50] Xilinx, “Soft Error Mitigation Controller v4.1, PG036,” 2018, [Online] Available: https://www.xilinx.com/support/documentation/ip_documentation/sem/v4_1/pg036_sem.pdf [30 April 2018]

[51] Präsentationsfolien zum Projektgespräch wischen BASE und TIS 17.03.2021

[52] Präsentationsfolien zum Projektgespräch wischen BASE und TIS 18.10.2022

Anhang A: ISTec-A-4010 Teilbericht

Das Dokument ist in der PDF-Datei verlinkt.

TÜV Rheinland Industrie Service GmbH (TIS)

TÜVRheinland®
Genau. Richtig.

Analyse der Fehlermodi von programmierbaren logischen Schaltungen in der Sicherheitsleittechnik von Kernkraftwerken

Teilbericht

R. Heigl
H. Miedl
F. Hellie
G. Schnürer

ISTec - A - 4010

Rev. 2

April 2021

Das diesem Bericht zugrundeliegende F&E-Vorhaben wurde im Auftrag des Bundesministeriums für Umwelt, Naturschutz und nukleare Sicherheit unter dem Kennzeichen 7420R01310 durchgeführt. Der Bericht gibt die Auffassung und Meinung des Auftragnehmers wieder und muss nicht mit der Meinung der Auftraggeberin übereinstimmen.

TÜV Rheinland Industrie Service GmbH – Zeppelinstraße 1, 85399 Hallbergmoos – Tel. +49 811 95945-0 - Fax +49 811 95945-41

Anhang B: Publikation NPIC&HMIT

FAILURE MODES OF PROGRAMMABLE LOGIC CIRCUITS

Raimund J. Heigl (Raimund.Heigl@de.tuv.com)[1], Horst Miedl (Horst.Miedl@de.tuv.com)[2]

[1]TÜV Rheinland Industrie Service GmbH (TIS), 85399 Hallbergmoos, Germany

[2]TÜV Rheinland Industrie Service GmbH (TIS), 85399 Hallbergmoos, Germany

ABSTRACT

Utilization of digital I&C for application in nuclear power plants led to a growing discussion on how to create methods for a probabilistic evaluation of such systems. The issues arise on the one hand due to a lack of commonly accepted models for evaluation of safety and reliability and on the other hand due to the problems of integrating such models into the evaluation of reliability of an overall system. Nevertheless even more complex systems are implemented to satisfy growing reliability requirements. Therefore, the question for a method to evaluate the reliability of systems based on software or on programmable logic became even more urgent.
For safety-related I&C of NPPs, devices based on programmable logic circuits such as FPGA (Field Programmable Gate Array) are increasingly available on the market. Accordingly, there is a need to identify or develop methods for evaluating their safety and reliability. It is expected that corresponding devices will be used during the remaining service life and beyond in retrofitting and modernization measures as well as to cover replacement requirements (redesign components).

Keywords: failure mode[4], programmable logic, FPGA design, complexity, fault injection

1. INTRODUCTION

The average age of NPPs currently operated in Europe exceeds 25 years. Thus the need for backfitting and modernization will inescapably grow in the next future. Most of the installed I&C systems are based on analogue technologies such as analogue electronic modules. The technical solutions available on the market mainly rely on digital technologies, applying microcontrollers or Field Programmable Gate Arrays (FPGA) [8]. These technologies still raise

[4] The term "fault mode" would be more appropriate but it is deprecated and failure mode is used instead [13].

many technical and procedural issues. One of them is the evaluation of software reliability. Since the time that computer-based systems have been used in other safety-relevant applications (e.g. Aviation, Telecommunication) there is an ongoing discussion concerning the safety assessment of software. The problems are due to the lack of widely accepted models for evaluating the reliability of software systems.

ISO/IEC/IEEE 24765:2017:3.3387-2 [12] defines reliability as degree to which a system, product or component performs specified functions under specified conditions for a specified period of time. In software engineered for a high-criticality system, if the conditions in its environment remain the same, its behavior should not change over time. Thus, its reliability should be evaluated in terms of the extent to which the system, product or component satisfies its requirements.

As known from practical experience it is the complexity of a software-system together with the high variety of its operational profile, which determines the risk of the software to fail, i.e. its dependability. Therefore, prediction of software dependability, based on complexity measurement is one of the promising approaches for evaluating the reliability of software [7]. In order to facilitate this approach in the nuclear field, research effort was spent to identify and quantify complexity of digital I&C systems and its correlation to reliability [1-3]. ISO/IEC/IEEE 24765:2017:3.694-3 [12] defines complexity as degree to which a system or component has a design or implementation that is difficult to understand and verify. The approach introduced in previous research was modified in order to enhance significance and also to apply the complexity measurement to a wide range of programmable I&C systems including Programmable Logic Controller (PLC) and FPGA technology.

In previous research, an approach for determining complexity based on a complexity vector was developed and the method was extended to programmable logic [9]. The complexity vector defines a metric that reflects the specific properties and features of programmable logic of digital I&C systems. The measuring procedure and its practical applicability has been demonstrated by investigation and application on several exemplary I&C devices, which are actually operating in nuclear applications.

Failure modes of FPGA-based I&C components were identified, their causes and effects, and various test procedures for safety and reliability assessment were analyzed. The aim was to investigate which faults the test procedures are capable of detecting. The tests are performed on FPGA designs developed as part of the research, which are representative for safety-relevant applications of I&C in NPP. Complexity of the developed FPGA designs was particularly taken into account for generation of test cases. The basic idea was to test complex sections of the design intensively in order to achieve an optimized test coverage. Test procedures were investigated for potential weaknesses by application of fault injection and the achieved test coverage was assessed.

2. CREATION OF FPGA DESIGNS FOR INVESTIGATION OF FAILURE MODES

Within the scope of the research project, a development environment was first selected for the creation of representative FPGA designs for I&C components. The selection of a suitable development environment is based on commercially available development tools, which are typically used for the creation of FPGA-based components in the safety I&C of nuclear power plants.

The FPGA designs were realized on the basis of a specification using the development environment. The type and scope of the created designs (or design variants) was chosen in view of further investigations with regard to validation of test procedures. The FPGA designs are sufficiently complex to allow practical conclusions to be drawn in the subsequent investigations. The design of the I&C components only includes their FPGA core. Other parts belonging to a practical I&C component, such as switching regulators for operating voltages, interfaces, protective circuits of inputs and outputs and self-monitoring were not considered during this development.

Based on the development, suitable test environments and interfaces for test automation were created. Various tools for verification were used for this purpose. A test bench was created for testing the FPGA designs. Testing was performed software-based, the FPGA designs were not configured in hardware.

Within the scope of the research project, three representative FPGA designs were selected for tool-based realization and investigation. These are

1. measurement signal acquisition with plausibility check (input check)
2. selection circuit of three measurement signal acquisitions with 2 of 3 triggering (T AmAra, see figure 1)
3. interconnection of three (2 of 3) selection circuits (redundant circuit)

These three representative FPGA designs replicate typical signal acquisitions (Case 1) and trip signal formations within a redundancy (Case 2) and across redundancies (Case 3) within nuclear facilities. In addition, these three FPGA designs exhibit different complexities, which in turn will be further analyzed.

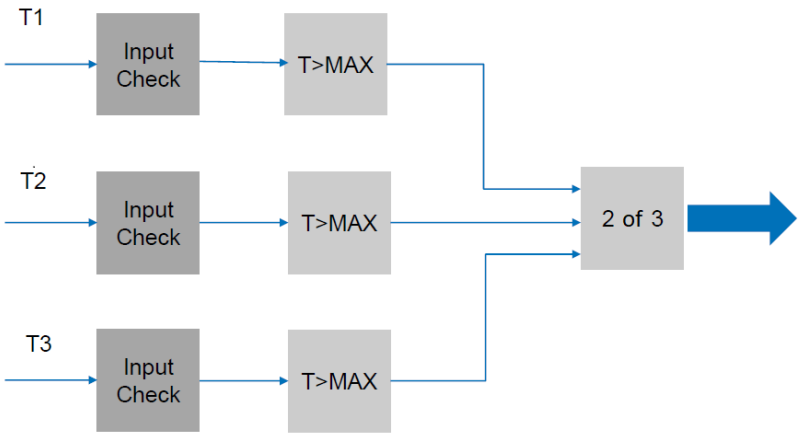


Figure 1. FPGA Design TAmAra – 2 of 3 temperature selection circuit.

In summary, the selected representative FPGA designs with different functional complexities were realized and investigated in a tool-based manner with the following objectives:

- Create transparency and traceability in FPGA development by applying a lifecylce phase model described in IEC 60880 to specific examples.
- Demonstrating the effectiveness of predefined test methods.
- Consistent implementation of nuclear requirements.
- Discussion and application of other approaches to testing from non-nuclear intern ational standards.
- Complexity analyses

3. FPGA DEVELOPMENT USING VITIS HLS AND VIVADO DESIGN SUITE

The chosen FPGA development methodology utilizes the Xilinx tools Vitis HLS ("High-Level Synthesis") [10] and Vivado Design Suite [11], and illustrated in figure 3. First, a system specification is created, i.e. the requirements of the system in which the FPGA (or HPD) is to be used. Based on this, the FPGA specification defines what the FPGA must do to ensure that the system specification is met. The FPGA specification includes both hardware requirements and software requirements. Usually, after determining the appropriate FPGA hardware type, the FPGA design is created, which primarily defines the software aspects of the FPGA. In the next development step, a module specification represents the requirements for the module to be developed (the HDL function), which the developer follows when creating the design.

After the development has completed a work step or the work in a phase of the development lifecycle, a verification of this work step is performed according to the test methods planned and predefined within the development. At the conclusion of development, validation (e.g., of the software) is performed, i.e., formal confirmation of compliance with the requirements of the specification. Compliance with the functional requirements of the specification is verified by means of the test bench.

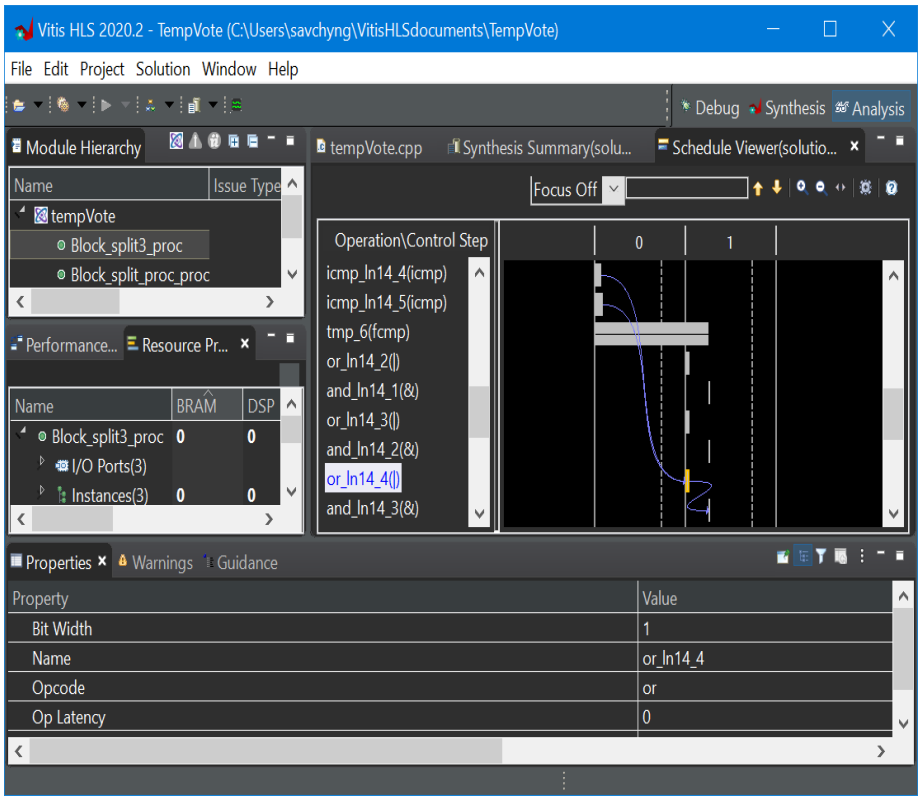


Figure 2. Schedule viewer in Vitis HLS.

Using Vitis HLS, the created C/C++ code is converted to HDL code. The created HDL code is then verified within Vitis HLS by analysis with respect to resource utilization and temporal behavior and by simulation with respect to functional correctness. The HDL code is then exported to Vivado Design Suite for further processing.

The development steps in Vitis HLS comprise several development and verification steps:

- Generation of C/C++ code and testing by “test bench”,
- HDL synthesis where a netlist for the FPGA is generated based on certain predefi ned properties,
- Analysis, for example of the temporal properties (see figure 2 "schedule viewer"),
- Optimization, which takes place iteratively according to the user's specifications,
- Simulation, e.g. by means of the wave function and export to Vivado Design Suite.

The development steps performed using Vivado Design Suite build on the HDL code generated in Vitis HLS. In this project, the compatibility of Vivado Design Suite with Vitis HLS is used in the creation of the FPGA designs. The compatibility of the development tools also extends to selected third-party products (e.g. ModelSim), which is used for independent verification.

With Vivado Design Suite, the previously generated source files are added first. Vivado accepts HDL files, EDIF netlists and files of some other formats for this purpose. For this project, the HDL design created in C/C++ and synthesized via Vitis HLS is used as the source file. After verifying the functional correctness and defining the constraints, the design can be further processed using Vivado's own synthesis. Finally, the conversion of the design to the target device is determined without actually writing it to any hardware. Then, this conversion is exported to an EDIF file in the form of a netlist for subsequent complexity analyses.

The development steps in Vivado Design Suite comprise several development and verification steps:

- Elaboration of the design by graphical representation used as an initial manual verification method,
- Definition of temporal constraints (timing) and physical constraints (place and routing),
- Vivado Synthesis rewriting the design into a chip-level netlist to improve effectiveness
- Implementation and optimization of the design without actual hardware implementation,
- EDIF Export in the form of a netlist for subsequent complexity analyses.

For this project, the design is exported as an EDIF netlist after implementation. The EDIF netlist created in this way can then be analyzed for its complexity using a pre-built tool [9].

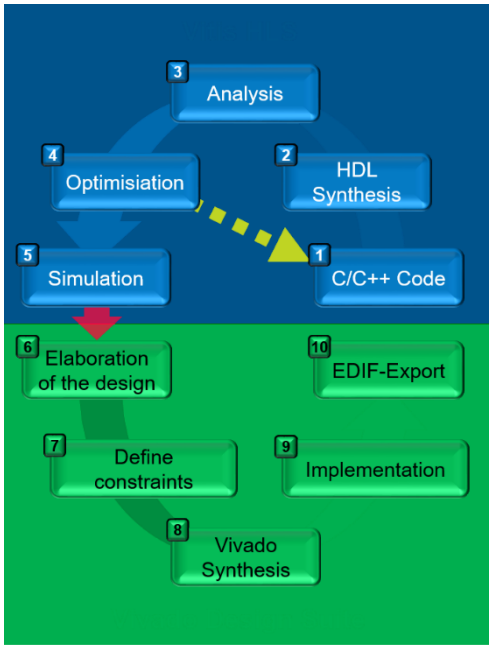


Figure 3. Development steps in Vitis HLS and Vivado.

4. FPGA TEST ENVIRONMENT

As a first approach the existing test environment within the FPGA development tools Vitis HLS and Vivado Design Suite was used (see chapter 3). As part of this environment separate test benches (in C++, RTL and VHDL) were created for the individual development steps and used for the tests accompanying the development.

Besides the correctness of the developed FPGA design, the correctness of the tools used for testing and development is also considered as an integral part of verification and validation. In FPGA development the test method is often not independent from the used development tool. The tools for FPGA development usually provide extensive verification and test features, which do not, however, exhibit the independence required by nuclear regulation. If the tests are carried out with the same tools as the development, no reliable statements can be made, for example, about the correctness of the compiler used or the functions for reading in and analysis. It is therefore advisable to carry out tests independently of the development tools.

Therefore test methods of a specially created independent test environment were applied, which are based on test tools that are diverse from development. This second test environment is suitable for the verification of the development process and the test environment of the development tools as well as for the independent test of the functional correctness of the FPGA designs. The second test program consists of the following test steps.

- Independent verification of the FPGA specification and review of the source code;
- Dynamic source code analysis
- Independent test with diverse computer and operating system;
- Independent creation and comparison of the generated wave functions;
- Testing of the source code with a diverse (Intel HLS) compiler;
- Testing of the FPGA designs with a diverse read-in tool (Quartus Prime)
- Testing of the FPGA designs with diverse test tool (Modelsim).

By independent verification of the specification, review of the source code and dynamic analyses minor errors were identified and corrected after reporting to the development. Independent tests with diverse computer and operating system were performed. The result of the first diversity test steps suggests that the compiler can be used regardless of the operating system and computers used. To verify the correctness of the FPGA designs, a test bench simulation was performed using two different test environments. The wave functions generated by the test program were then compared. The test results show that the time histories and shapes of the wave functions match. The conclusion of the first test steps is that FPGA applications can be used on different operating systems without any adjustments being necessary.

The correctness of the operation of the Vitis HLS compiler is checked with the help of Intel Altera HLS, a diverse compiler for development. Initially mixing different language versions caused problems during testing, as the syntax was interpreted differently by the two compilers. By developing a uniform source code that can be used for both Vitis HLS and Intel Altera HLS, the translation capability was tested regardless of the compiler used. In both test cases, the design was successfully placed on a virtually generated FPGA chip or board.

To test the correctness of the steps within the Vivado Tool Suite (Xilinx) used for development, a diverse program called Quartus Prime (Intel Altera) was used. The synthesized Verilog codes were successfully compiled and analyzed by Quartus Prime during the test execution. The subsequent placement and connection (place-and-route) process was performed without any abnormalities. A comparison of the two diversely generated switch representations is not meaningful due to their very different structure. As a test result, however, it can be stated that both programs are functionally identical, despite the use of two different tools for reading. The functional correctness of the program can thus be achieved by synthesizing the C++ code of the FPGA designs in Vivado as well as in Quartus Prime after integration in the respective tool for read in. In contrast, using an HLS compiler with a non-associated read-in tool leads to serious errors in compilation, parsing, and the place-and-route process of placement and connection. The test results suggest that the vendors use different algorithms that produce the same results when the development tools are used correctly.

To verify the correctness of the C++ based test bench simulation used for the development of the FPGA designs, a test bench simulator called ModelSim, which is diverse to the Vivado toolsuit, was used. To perform the test, the synthesis of the C++ based test bench simulation with the respective HLS compiler (Altera or Xilinx) was required. As a result the wave functions generated do not show any discrepancies when applying the different test bench simulators.

5. FPGA FAILURE MODES ANALYSIS

The developed FPGA design variants (see chapter 2) and the associated test environment (see chapter 4) were used to determine the potential for errors and the failure modes. Based on the experiences from the development of FPGA designs, possible vulnerabilities in the design and resulting potential sources for failure modes were identified. These were used to generate test cases and postulate potential faults for fault injection.

The identified potential or actual failure modes were classified with respect to their safety implications. Table I summarizes the potential sources of FPGA failure modes from the consideration of design vulnerabilities, as well as the possibility of their detection with the applied test methods and remaining potential test gaps. The remaining test gaps were taken into account accordingly for fault injection (see chapter 7).

Table I. Potential sources of FPGA failure modes in the development

Failure sources	Example	Test method	Potential test gaps
Error in C++ program	Wrong limits Multiple variable usage Logic error	C-Simulation Dynamic analyses Determination of cyclomatic complexity	Errors in the test bench e.g. regarding limit and input values, syntax (fault injection required)
Incorrect resource usage	Too long operation time	Analysis of the HDL program	Correct functionality of the HDL program (C/RTL co-simulation required)
Error in HDL design	Functional differences between C++ and HDL	C/RTL co-simulation Vivado simulation	Error in test bench (fault injection required) Errors in the physical implementation (Vivado analyses required)
Timing errors	Incorrect timing of the program Signal loss	Wave function	Errors in the test bench are only partially detectable (fault injection required)
Placing errors	Incorrect pin assignment	Vivado analyses	Incorrect specifications (no measure identified)
Tool-related errors	Errors in design are generated or remain undetected	Diversity of test methods (compiler, operating system, read-in program, simulator)	-

6. FAULT INJECTION OF POSTULATED FAILURE MODES

Analysis of the FPGA designs developed in the research project enables the generated test environment to be analyzed for weaknesses by fault injection and for the achieved test coverage. To postulate possible failure modes for fault injection, experience from the development of the FPGA designs and the used test environment, possible vulnerabilities in the design and resulting potential sources for failure modes were considered. Injection of realistic postulated faults will be used to analyze the test case generation tools and determine their effectiveness. Potential test gaps are identified through fault injection and documented accordingly. In particular, fault injection was used to investigate the potential failure modes that could not or not sufficiently investigated using the available development test methods (see chapter 3) or the independent test environment (see chapter 4).

Technically the fault injection is realized by a manipulation of the C++ source code. With the help of the generated test benches, the test coverage of the test environment is examined and the possible effects of the injected faults are simulated.

The test steps examined for this purpose are compilation, debugging (C simulation), test bench simulation (C/RTL co-simulation), and wave function. It is analyzed whether the test step is executed successfully or whether the injected fault is intercepted by the test steps. Intercepted is to be understood here as compilation fails, debugging reports errors, the test bench simulation aborts or reports errors, or the generation of the wave function fails.

In course of the research project, seven different fault injection scenarios were developed and executed. Five of the seven fault scenarios were uncovered by the used tool set, which means that two of the fault injection scenarios could not be detected by the used FPGA tools. Certain injected faults can only be detected or controlled by a V&V process. Four of the test scenarios were classified as potentially critical. The classification of potentially critical means that additional V&V activities were necessary to detect or control the injected fault. For example, for error injection 2 ("invalid values"), the program had to be modified to control the injected fault. Table 2 summarizes the executed fault injection scenarios and which of the failure modes could be uncovered by the used test tools.

Table II. Fault injection of postulated FPGA failure modes

No.	Fault injection scenario	Compile	Debugging	Test bench simulation	Wave function
		Executed and passed successfully			
1	Simulation of sensor failure	☒	☐	☐	☐
2	Usage of invalid values	☒	☒	☒	☒
3	Double assignment of variables	☐	☐	☐	☐
4	Syntax and semantic errors	☐	☐	☐	☐
5	Limit value adjustment	☒	☐	☒	☐
6	Precision of input values	☒	☒	☒	☒
7	Decimal point and comma	☒	☒	☐	☐

The fault injection analysis for scenario No. 2 and No. 6 shows, that the program was compiled successfully (i.e. without abnormalities or abort) despite the injected fault. No error occurs when debugging the program in the test bench simulation. The test bench simulation runs successfully and the wave function can be generated as usual and be used for further testing. The potential failure mode could not be detected by the test environment. To control the failure mode the program had to be modified.

Only two of the scenarios were not detected by the tool set in the applied test environment. However, the fault injection analysis revealed that four scenarios were classified as potentially critical. For syntax and semantic errors or the adjustment of limit values, the injected faults were uncovered by a defensive programming approach during test bench creation. This is not considered to be a prerequisite for any realization. This shows that for FPGA development in addition to automated testing, the implementation of a verification and validation (V&V) lifecycle is required to cover as many potential failure modes as possible.

7. CONCLUSIONS

Most digital I&C systems, programmed by graphic-based specification of its functionality, such as FPGA based logic, have a common structure. For the description and quantification of their complexity, a unified concept can be used, which was elaborated in previous research projects and modified for the evaluation of FPGAs.

Dedicated FPGA designs, which are representative for safety-relevant applications of I&C in NPP, were created. Development of FPGA designs revealed insights into commonly used development tools, test and simulation environments contained therein and their operating principle. It was recognized that the tools for FPGA development offer a variety of testing, simulation and analysis features in addition to the development steps. These extensive verification and test features do not exhibit the level of independence required by nuclear regulation. For tests that are carried out with the same tools as the development, no reliable statements can be made, for example, about the correctness of the compiler used or the functions for read in, analysis or simulation. Therefore, the tests of the FPGA designs were additionally carried out with a test environment independent of the development. The test results showed no significant differences between the two test environments used.

Complexity of the developed FPGA designs was particularly taken into account for generation of test cases. The basic idea was to test complex sections of the design intensively in order to achieve an optimized test coverage. Based on the experiences from development of FPGA designs, possible vulnerabilities in the design and resulting potential sources for failure modes were identified. These were used to generate test cases and postulate potential faults for fault injection. The aim was to investigate which faults the test procedures are capable of detecting. The identified potential or actual failure modes were classified with respect to their safety implications.

As a result of the FPGA failure mode analysis and the evaluation of the fault injection scenarios in particular, the application of a V&V process is found to be crucial. The analysis shows, that a V&V process cannot be completely automated or left unconditionally to the test tools supplied with an FPGA development environment. As fault injection analysis showed certain injected faults could only be detected and controlled by a dedicated V&V process, since exclusively due to application of a V&V lifecycle process some of the consciously injected faults could remain uncovered.

8. ACKNOWLEDGMENTS

The work on analysis of failure modes of programmable logic circuits in safety-relevant IEC of nuclear power plants is funded by the German Federal Office for the Safety of Nuclear Waste Management (BASE) under the project number 47207R01310 within its scope of research in reactor safety. Special thanks go to Dr. Mario Hellmich for his competent and profound support during the elaboration of the project results.

9. REFERENCES

1. J. Märtz, A. Lindner, H. Miedl, M. Baleanu “COMPLEXITY MEASUREMENT AND RELIABILITY OF SOFTWARE IN DIGITAL I&C-SYSTEMS”, NPIC&HMIT 2004, Columbus, Ohio, September, 2004.
2. J. Märtz, A. Lindner, H. Miedl, “COMPLEXITY MEASUREMENT OF SOFTWARE IN DIGITAL I&C-SYSTEMS”, NPIC&HMIT 2009, Knoxville, Tennessee, April 5-9, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009).
3. J. Holmberg, S. Guerra, N. Thuy, J. Märtz, B. Liwång, “HARMONICS-EU FP7 PROJECT ON THE RELIABILITY ASSESSMENT OF MODERN NUCLEAR I&C SOFTWARE”, NPIC&HMIT 2012, San Diego, CA, July 22-26, 2012.
4. J. Märtz, H. Miedl, A. Lindner, C. Gerst, „KOMPLEXITÄTSMESSUNG DER SOFTWARE DIGITALER LEITTECHNIKSYSTEME“, ISTec-A-1569, February 2010.
5. F. Hellie, A. Lindner, A. Mölleken, “KOMPLEXITÄT UND FEHLERPOTENTIAL BEI SOFTWAREBASIERTER DIGITALER SICHERHEITSLEITTECHNIK”, Interim Report, ISTec-A-2891, February 2016.
6. R. Heigl, F. Hellie, A. Lindner, A. Mölleken, „KOMPLEXITÄT UND FEHLERPOTENTIAL BEI SOFTWAREBASIERTER DIGITALER SICHERHEITSLEITTECHNIK“, Final Report, September 2017.
7. Smidts, C., “FROM MEASURES TO RELIABILITY”, NPIC&HMIT 2000, Washington D.C., November 13-16, 2000.
8. IAEA Nuclear Energy Series No. NP-T-3.17, APPLICATION OF FIELD PROGRAMMABLE GATE ARRAYS IN INSTRUMENTATION AND CONTROL SYSTEMS OF NUCLEAR POWER PLANTS, International Atomic Energy Agency, Vienna (2016).
9. R.J. Heigl, A. Lindner, “COMPLEXITY AND ERROR POTENTIAL OF DIGITAL I&C”, NPIC&HMIT 2017, San Francisco, California, June, 2017.
10. Getting Started with Vitis HLS, https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/yxj1602022623766.html [Online, Call up: 07.07.2021].
11. Product page to Vivado Design Suite, https://www.xilinx.com/products/design-tools/vivado.html, [Online; Call up: 21.06.2021].
12. ISO/IEC/IEEE 24765:2017, ISO/IEC/IEEE International Standard - Systems and software engineering - Vocabulary.
13. IEC 60050, International Electrotechnical Vocabulary

Anhang C: PowerPoint Folien zum Vortrag der TIS zur NPIC

Failure modes of programmable logic circuits

NPIC & HMIT 2023

July 15-20, Knoxville Tennessee

Raimund Heigl / Horst Miedl
TÜV Rheinland Industrie Service GmbH
Zeppelinstraße 1, 85399 Hallbergmoos
Germany

TÜVRheinland®
Precisely Right.

Introduction

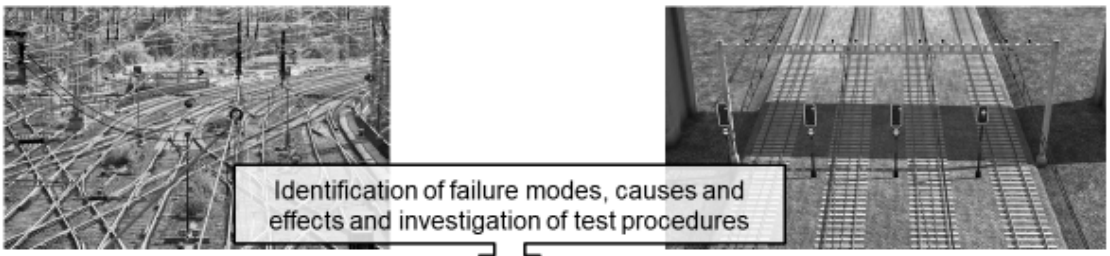
"The assignment of the application functions to the systems shall attempt to minimize the complexity of class 1 systems."

IEC 61513:2011

source: Licensing experience Reconstruction of NPP Unit Computer Systems, Károly Hamar, HAEA

TÜVRheinland®
Precisely Right.

Introduction



Identification of failure modes, causes and effects and investigation of test procedures

Development of FPGA designs representative for safety-relevant applications of I&C in NPP

Analysis of FPGA failure modes considering the impact of complexity

3 7/24/2023 Failure modes of programmable logic circuits NPIC & HMIT 2023

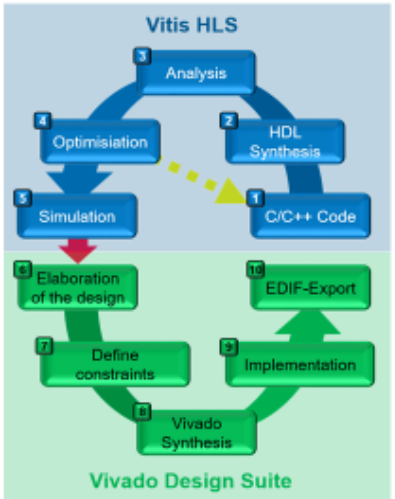
TÜVRheinland® Precisely Right.

Creation of FPGA designs for investigation of failure modes

Development steps in Vitis HLS and Vivado

FPGA development tools used (**Xilinx**):

- Vitis „High-Level Synthesis" HLS
- Vivado Design Suite



The EDIF netlists are analyzed for complexity using a pre-built tool

4 7/24/2023 Failure modes of programmable logic circuits NPIC & HMIT 2023

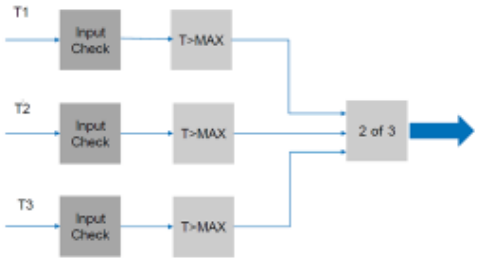
TÜVRheinland® Precisely Right.

Creation of FPGA designs for investigation of failure modes

Representative FPGA designs for I&C components

3 representative FPGA designs were selected for tool-based realization and investigation:

1. **Typical signal acquisition:**
   Measurement signal acquisition with plausibility check
2. **Trip signal formation:**
   Selection circuit of three measurement signal acquisitions (with 2 of 3 triggering)
3. **Redundant circuit:**
   Interconnection of three (2 of 3) selection circuits



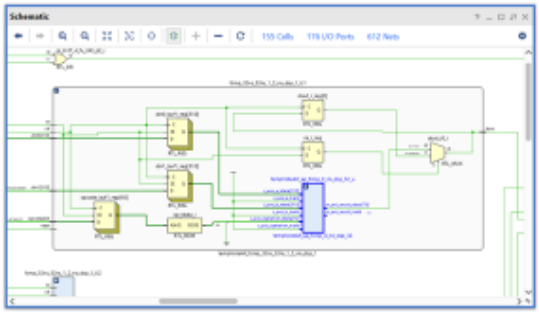
6 7/24/2023 Failure modes of programmable logic circuits NPIC & HMIT 2023

TÜVRheinland® Precisely Right.

FPGA test environment

Correctness of design vs correctness of the used tools

1. **Use of "existing" test environment as part of the development tools**
   - Simulation, optimization and test benches (in C++, RTL and VHDL)



2. **Creation of an independent test environment**
   - Independent verification of the FPGA specification and review of the source code;
   - Dynamic source code analysis
   - Independent test with diverse computer and OS;
   - Independent creation and comparison of the generated wave functions;
   - Testing of the source code with a diverse (Intel HLS) compiler;
   - Testing of the FPGA designs with a diverse read-in tool (Quartus Prime)
   - Testing of the FPGA designs with diverse test tool (Modelsim).

6 7/24/2023 Failure modes of programmable logic circuits NPIC & HMIT 2023

TÜVRheinland® Precisely Right.

FPGA failure modes analysis

Potential sources of FPGA failure modes during development

Failure sources	Example	Test method	Potential test gaps
Error in C++ program	Wrong limits Multiple variable usage Logic error	C-Simulation Dynamic analyses Determination of cyclomatic complexity	Errors in the test bench e.g. regarding limit and input values, syntax (fault injection required)
Incorrect resource usage	Too long operation time	Analysis of the HDL program	Correct functionality of the HDL program (C/RTL co-simulation required)
Error in HDL design	Functional differences between C++ and HDL	C/RTL co-simulation Vivado simulation	Error in test bench (fault injection required) Errors in the physical implementation (Vivado analyses required)
Timing errors	Incorrect timing of the program Signal loss	Wave function	Errors in the test bench are only partially detectable (fault injection required)
Placing errors	Incorrect pin assignment	Vivado analyses	Incorrect specifications (no measure identified)
Tool-related errors	Errors in design are generated or remain undetected	Diversity of test methods (compiler, operating system, read-in program, simulator)	-

7 7/24/2023 Failure modes of programmable logic circuits NRC & HMIT 2023

TÜVRheinland® Precisely Right.

Fault injection of postulated FPGA failure modes

Fault injection is realized by C++ source code manipulation.
Test coverage is examined by test benches. The possible effects of the injected faults are simulated.

No.	Fault injection scenario	Compile	Debugging	Test bench simulation	Wave function
		Executed and passed successfully			
1	Simulation of sensor failure	☒	☐	☐	☐
2	Usage of invalid values	☒	☒	☒	☒
3	Double assignment of variables	☐	☐	☐	☐
4	Syntax and semantic errors	☐	☐	☐	☐
5	Limit value adjustment	☒	☒	☒	☐
6	Precision of input values	☒	☒	☒	☒
7	Decimal point and comma	☒	☒	☐	☐

8 7/24/2023 Failure modes of programmable logic circuits NRC & HMIT 2023

TÜVRheinland® Precisely Right.

Investigation of the effects of changes on FPGA designs

Changes are made to the boundary values in the C++ source code of the respective design

Design variant	# blocks	connections	Complexity metrics		
			V (FP)	$V(FP)_{mod}$	$V(FP)_{internal}$
Original	705	3373	10,8	86,2	75,4
Limit value 20	768	3718	10,8	97,2	86,4
Limit value 0	740	3241	2,1	52,4	50,4
Limit value -20	816	4003	10,8	103,3	92,5

*V(FP) is a measure of the interconnection of logical blocks.
A higher number means more signal connections between the logical blocks which indicates a higher complexity

- For the design with limit value zero, "unnecessary" connections are "optimized out" during synthesis
- The added limit zero leads to an improved optimization of the design

9 7/24/2023 Failure modes of programmable logic circuits NPIC & HMIT 2023

TÜVRheinland® Precisely Right.

Conclusions

1. Unified approach to describe and **quantify complexity of programmable logic**
   - more complex areas of the FPGA netlists are tested more intensively to achieve a better **test coverage**
   - Improve FPGA design **optimization** (logical operation & complexity)
2. Insights into tools for development, test and simulation environments
   - Identification of possible **weaknesses in the design**
   - Classification of **potential sources of failure modes** investigated e.g. by fault injection
   - Identification of potential **test gaps**
   - Tools offer extensive built-in verification and test capabilities, which do not exhibit the degree of **independence** from development required by nuclear standards
3. Results will be used to derive criteria for qualification of programmable logic

V&V process cannot be fully automated or unconditionally left to the test tools provided with an FPGA development environment

10 7/24/2023 Failure modes of programmable logic circuits NPIC & HMIT 2023

TÜVRheinland® Precisely Right.


Thank you
for your attention!
Contact: raimund.heigl@de.tuv.com
horst.miedl@de.tuv.com
11 7/24/2023 Failure modes of programmable logic circuits NPIC & HMIT 2023
TÜVRheinland®
Precisely Right.