



Abschlussprüfung Winter 2023/2024

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

FPGA-Konsolenanwendung

FPGA - Automatische Generierung von Test Bench Vorlagen

Abgabedatum: Dortmund, den 18.10.2023

Prüfungsbewerber:

Burcu Arik
Öztalerstr. 10
44339 Dortmund

Ausbildungsbetrieb:

Quendro GmbH
Gröpperstr. 5
58454 Witten



Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis.....	I
Abbildungsverzeichnis.....	IV
Tabellenverzeichnis.....	V
Glossar.....	VI
Abkürzungsverzeichnis.....	VII
1 Einleitung	1
1.1 Projektumfeld	1
1.2 Projektziel	1
1.3 Projektbegründung.....	2
1.4 Projektschnittstellen	2
1.4.1 Organisationschnittstellen:	2
1.4.2 Persönliche Schnittstellen:	2
2 Projektplanung	3
2.1 Projektphasen	3
2.2 Abweichungen vom Projektantrag.....	3
2.3 Ressourcenplanung	4
2.3.1 Hardwareressourcen	4
2.3.2 Softwareressourcen	4
2.3.3 Personalressourcen	4
2.4 Entwicklungsprozess.....	5
3 Analysephase.....	6
3.1 Ist-Analyse	7
3.2 Soll-Konzept.....	8
3.2.1 Fachliche Anforderungen	8
3.2.2 Technische Anforderungen	9

Inhaltsverzeichnis

3.3	Wirtschaftlichkeitsanalyse	9
3.3.1	Make or Buy-Entscheidung	10
3.3.2	Projektkosten	10
3.3.3	Amortisationsdauer	13
3.4	Nutzwertanalyse.....	14
3.5	Anwendungsfälle	15
3.6	Qualitätsanforderungen	16
3.7	Lastenheft/Fachkonzept	18
4	Entwurfsphase	18
4.1	Zielplattform	18
4.2	Geschäftslogik.....	18
4.3	Maßnahmen zur Qualitätssicherung	25
4.4	Pflichtenheft/Datenverarbeitungskonzept	26
5	Implementierungsphase	26
5.1	Implementierung der Geschäftslogik	26
6	Testphase und Qualitätskontrolle	38
7	Abnahme.....	38
8	Dokumentation	39
9	Fazit	39
9.1	Soll-/Ist-Vergleich	40
9.2	Wirtschaftlichkeitsbetrachtung	40
9.3	Erfahrungen	40
	Literaturverzeichnis	41
	Persönliche Erklärung	42
	Anhang.....	i
A1	Detaillierte Zeitplanung.....	i
A2	Lastenheft (Auszug)	i

Inhaltsverzeichnis

A3 Use Case Diagramm	iii
A4 Pflichtenheft (Auszug)	iii
A5 Code Ausschnitte	vi
A6 Testprotokolle (Auszug).....	viii
A7 Benutzer Dokumentation	ix
A8 Tabelle Soll-/Ist-Vergleich (detailliert)	xii

Abbildungsverzeichnis**Abbildungsverzeichnis**

Abbildung 1:	6
Abbildung 2:	20
Abbildung 3:	21
Abbildung 4:	22
Abbildung 5:	23
Abbildung 6:	24
Abbildung 7:	25
Abbildung 8:	30
Abbildung 9:	31
Abbildung 10:	31
Abbildung 11:	32
Abbildung 12:	34
Abbildung 13:	34
Abbildung 14:	35
Abbildung 15:	36
Abbildung 16:	iii
Abbildung 17:	vi
Abbildung 18:	vi
Abbildung 19:	vii
Abbildung 20:	viii
Abbildung 21:	viii
Abbildung 22:	ix
Abbildung 23:	ix
Abbildung 24:	x
Abbildung 25:	xi
Abbildung 26:	xi
Abbildung 27:	xii

Tabellenverzeichnis

Tabellenverzeichnis

Tabelle 1: grobe Zeitplanung	3
Tabelle 2: Detaillierte Auflistung Hard - Softwareressourcen	5
Tabelle 3: Personalkosten	10
Tabelle 4: Sachmittelkosten	12
Tabelle 5: Nutzwertanalyse	14
Tabelle 6: Softwarequalität	17
Tabelle 7: Soll Ist Vergleich	40
Tabelle 8: Detaillierte Zeitplanung	i
Tabelle 9: Detaillirte Soll-Ist Vergleich	xii

Glossar

Glossar

IntelliJ IDEA :

Das ist eine integrierte Entwicklungsumgebung (IDE) des Softwareunternehmens Jet Brains für die Programmiersprachen Java, Kotlin, Groovy und Scala.

FPGA :

Ein FPGA (Akronym für Field Programmable Gate Array) ist ein integrierter Schaltkreis (IC) der Digitaltechnik, in welchen eine logische Schaltung geladen werden kann.

FPGA Test Bench-Vorlage :

Ein Test Bench Vorlage ist eine vorgefertigte Struktur oder ein vorgefertigter Code-Rahmen, der speziell für die Erstellung von Testbenches in FPGA-Projekten entwickelt wurde. Diese Vorlagen sind hilfreich, um die Entwicklung von Testbenches zu beschleunigen und sicherzustellen, dass die FPGA-Designs ordnungsgemäß funktionieren.

Xilinx Vivado :

Xilinx Vivado ist eine integrierte Entwicklungsumgebung (IDE) und ein Tool-Set, das von Xilinx, einem führenden Hersteller von Field-Programmable Gate Arrays (FPGAs) und programmierbaren Logikbausteinen (PLDs), entwickelt wurde.

Verilog :

Verilog ist eine Hardwarebeschreibungssprache (HDL), die in der Elektronik- und Halbleiterindustrie weit verbreitet ist.

VHDL :

VHDL steht für "VHSIC Hardware Description Language", wobei "VHSIC" für "Very High-Speed Integrated Circuit" steht. Es handelt sich um eine Hardwarebeschreibungssprache (HDL), die in der Elektronik- und Halbleiterindustrie weit verbreitet ist. VHDL wird hauptsächlich zur Beschreibung und zum Entwurf von digitalen Schaltungen und Systemen verwendet, ähnlich wie Verilog.

FPGA-KONSOLENANWENDUNG

FPGA - Automatische Generierung von Test Bench Vorlagen

Abkürzungsverzeichnis

Abkürzungsverzeichnis

FPGA.....Field Programmable Gate Array

VHDL..... (VHSIC: Very High-Speed Integrated Circuit) Hardware Description Language

1 Einleitung

Dieses Dokument wurde erstellt, um den Ablauf des IHK-Abschlussprojekts zu beschreiben, das Teil eines Lehrgangs für Anwendungsentwicklungskompetenz ist. Das Ausbildungsunternehmen ist die Business IT Learning Centre GmbH (BitLC). BitLC organisiert Umschulungs- und Entwicklungsprogramme für IT-Interessierte und Neueinsteiger in der Branche. Zum Ausbildungsprogramm gehört in der Regel ein sechsmonatiges Praktikum in einem anderen IT-Unternehmen. Die Firma, bei der ich mein Praktikum absolviert habe, bietet Softwarelösungen, insbesondere für eingebettete Systeme, hat einen FPGA-Entwicklungsservice und ist auch in anderen Bereichen tätig. Die Quendro GmbH bietet auch Online-Kurse zur Softwareentwicklung an.

1.1 Projektumfeld

Der Auftraggeber dieses Projekts ist die Firma Quendro GmbH, die im Jahr 2022 gegründet wurde und ihren Firmensitz in Witten hat. Aufgrund der Tatsache, dass es sich bei der Quendro GmbH um ein neu gegründetes Start-up-Unternehmen handelt, besteht das Team derzeit aus lediglich 2 Mitarbeitern. Das Unternehmen arbeitet auf Basis von Projekten und Verträgen. Seine Expertise liegt in der Entwicklung von Software (C, C++), FPGA-Programmierung (Verilog, VHDL) sowie elektronischen Geräten (Hardware). Die Firma arbeitet hauptsächlich an Systemen, die in Kombination mit FPGA (Field Programming Gate Arrays).

1.2 Projektziel

Das Ziel dieses Abschlussprojekts ist die Erstellung automatisierter Test Bench Vorlagen für FPGA-Entwickler. Im Rahmen dieses Projektkonzepts plane ich, eine Grundvorlage für eine Test Bench-Datei zu erstellen. Ich werde eine Verilog-Datei als Eingabe verwenden und daraus eine Test Bench-Vorlage generieren. Dieses Testskript ist vergleichbar mit den Unit -Test in der Softwareentwicklung.

FPGA steht für "Field programmierbares Gate-Array". FPGAs sind dynamisch reprogrammierbare Bausteine. Mit diesen Bausteinen können Hardware-Designs und Datenbusse auf die spezifischen Anforderungen des Benutzers zugeschnitten werden. Die Programmiersprache, die für FPGAs verwendet wird, heißt Verilog. Die Erstellung einer ausführbaren Binärdatei für

ein FPGA kann Stunden dauern, und wenn dabei ein logischer Fehler auftritt, müssen Korrekturen vorgenommen und erneut gepflegt werden, was viel Zeit in Anspruch nehmen kann. Um dies zu vermeiden, schreiben FPGA-Entwickler zunächst Test Benches.

Damit können sie ihr Design in Sekundenschnelle testen und entscheiden, ob es funktioniert oder nicht. Als Teil dieses Projektkonzepts plane ich, eine grundlegende Vorlage für eine Test Bench-Datei zu erstellen. Diese Vorlage wird eine Verilog-Datei als Eingabe akzeptieren und daraus eine Test Bench-Vorlage als Ausgabe generieren

Dies ist eine wichtige Anforderung für Vivado (eine FPGA-Entwicklungsumgebung), für die es noch keine allgemein verfügbare Lösung gibt. So können alle FPGA-Entwickler es verwenden.

1.3 Projektbegründung

Die Quendro GmbH hat sich zum Ziel gesetzt, FPGA-Entwicklern eine dynamische und arbeitssparende Umgebung zur Verfügung zu stellen, um in kurzer Zeit effizient arbeiten zu können. Damit spart sie Zeit für Projekte durch die Beschleunigung von Prozessen, die lange Entwicklungszeiten und manchmal Tage zum Testen benötigen. Eines der primären Ziele der Firma Quendro ist es, die Arbeitsbelastung der Entwickler auf eine ausgewogene Produktivität zu lenken. Auf diese Weise können sich FPGA-Entwicklern über längere Zeiträume auf komplexere Teile großer Projekte konzentrieren. Aus diesem Grund werden die FPGA-Test Bench-Vorlagen, die ich in meinem Projekt entwickelt habe, den Entwicklern unter anderem Effizienz, Zeitersparnis, Kostenersparnis und weniger Fehler bringen. Darüber hinaus wird dieses Projekt, das ich der Welt der Technologie vorstellen werde, für andere FPGA-Entwickler in der Welt nützlich sein, und ich als einen Entwicklerin werde mit diesem innovativen und effizienten Projekt zufrieden sein.

1.4 Projektschnittstellen

1.4.1 Organisationschnittstellen:

Auftraggeber ist die Quendro GmbH unter der Leitung von Herrn Kocal.

1.4.2 Persönliche Schnittstellen:

Ich bin als Entwicklerin persönlich für die Umsetzung des Projekts verantwortlich, und Herr Kocal war bei den technischen Diskussionen und der Überprüfung des Codes anwesend.

2 Projektplanung

2.1 Projektphasen

Für die Durchführung des Projekts sind 80 Stunden vorgesehen. Es ist nach den Vorgaben der Industrie- und Handelskammer Dortmund entwickelt worden. Dies entspricht einer Arbeitszeit von 8 Stunden pro Tag und genau 10 Arbeitstagen, d.h. 2 Arbeitswochen. Der Zeitraum für die Umsetzung meines Projekts erstreckt sich vom 29. August 2023 bis zum 11. September 2023. Um sicherzustellen, dass das Projekt innerhalb dieser festgelegten Zeitspanne erfolgreich durchgeführt werden kann, habe ich das Projekt in mehrere separate Arbeitsbereiche oder Aufgaben unterteilt. Tabelle 1 zeigt, wie lange es dauert, diese Arbeitszweige zu realisieren. Ein detaillierter Zeitplan ist unter A1: Detaillierter Zeitplan beigefügt.

Projektplanung	
Projektphase	Soll-Stunden
1 Analyse	4h
2 Konzeption	9h
3 Planung	6h
4 Realisierung	26h
5 Validierung	11h
6 Abschluss	7h
7 Dokumentation	17h
Gesamtsumme	80h

Tabelle 1 grobe Zeitplanung

2.2 Abweichungen vom Projektantrag

Es gab 2 Vorgänge, die ich im Projektantrag nicht angegeben hatte, die aber bei der Implementierung des Projekts durchgeführt werden mussten.

Erstens: Ich musste auch eine Kommentarzeile für die Stellen entfernen, an denen sich Kommentare in den Codezeilen befinden, die ich aus der Verilog-Datei übernommen habe.

Zweitens: Ein weiterer Vorgang, der in der Entwurfsphase nie stattfand, war die Ermittlung der signierten Codes. Wenn die Zeile vorzeichenbehaftet ist, fügte ich sie mit einer If-Bedingung zur Ausgabe hinzu.

2.3 Ressourcenplanung

Die Quendro GmbH stellt verschiedene Ressourcen für die Realisierung des Projekts zur Verfügung. Dazu gehören Personal-, Finanz-, Material-, Software- und Infrastrukturressourcen. Zu den personellen Ressourcen gehören der Geschäftsführer und Softwareentwickler Herr Kocal und mich als Projektleiter der Quendro GmbH. Als finanzielle Mittel Gehälter, Infrastrukturkosten und Verbrauchsmaterial kann ich sprechen. Zu den materiellen Ressourcen gehören Verbrauchsmaterialien wie z.B. Büromaterialien. Bildungsressourcen sind die folgenden;

Schulungsunterlagen, die für die Ausbildung von Mitarbeitern und Auszubildenden benötigt werden, E-Learning-Plattformen.

2.3.1 Hardwareressourcen

Die Projektarbeit werde ich mit einem VivoBook_AsusLaptop mit einem 15,6 Zoll Bildschirm umsetzen, der über die folgenden technischen Merkmale verfügt: einen Intel(R) Core(TM) i5-1035G1 Prozessor mit 1,00 GHz (bis zu 1,20 GHz), 8 GB Arbeitsspeicher und eine zusätzliche Festplatte mit 512 GB Speicherkapazität. Zudem steht mir am Arbeitsplatz ein zusätzlicher Bildschirm zur Verfügung.

2.3.2 Softwareressourcen

Der Laptop hat ein Windows 11 Home-Betriebssystem. Die eingesetzte IDE ist Xilinx Vivado und IntelliJ. Die Diagramme habe ich mit Draw.io erstellt. Die Testskripte wurden in Java erstellt. Um die generierten Testskripte mit den Eingabedateien zu vergleichen, habe ich Beyond Compare 4 verwendet. Beyond Compare macht es einfach, lange Texte zu vergleichen und Unterschiede zwischen ihnen zu finden. Ich habe die Eingabe- und Ausgabedateien mit Notepad++ analysiert, was nützlich ist, weil es die gewünschten Wörter einfärbt. Getestet mit Vivado-Simulator. Die Projektdokumentation wird mit Microsoft Word, in der Cloud Version 365, geschrieben.

2.3.3 Personalressourcen

Personelle Schnittstelle ist Herr Saban Kocal in seiner Rolle als Betreuer und Auftraggeber. Herr Saban stand während der Projektentwicklungsphase stets als professioneller Ansprechpartner zur Verfügung, um Fragen zu beantworten und bei technischen Problemen zu helfen. Er teilte sehr wichtige Ideen bei Code-Reviews. Er hat mich bei der Überprüfung

FPGA-KONSOLENANWENDUNG

FPGA - Automatische Generierung von Test Bench Vorlagen

und Genehmigung meines Projekts stets professionell angeleitet. Als Projektleiter werde ich zuständig sein.

Die im Projekt eingesetzte Hard- und Software ist in der beigefügten Tabelle im Detail dargestellt:

Ressourcenplanung	
Hardware	
Laptop	VivoBook_AsusLaptop
Prozessor	Intel(R) Core(TM) i5-1035G1
Datenträger	512 GB SSD Festplatte
Bildschirm	ASUS VZ279HE-W
RAM (Random-Access Memory)	8,00 GB (7,74 GB verwendbar)
Software	
Betriebssystem	64-Bit-Betriebssystem, x64-basierter Prozessor
Browser	Google Chrome
Entwicklungsumgebung	IntelliJ IDEA
Sprache um Konsolenanwendung aufzubauen	Java
Vergleichumgebung	Beyond Compare 4
Ausgabedateien analyseumgebung	Notepad++
Draw.io	Ein Tool zur Diagrammen Erstellung
Microsoft Word und PowerPoint	Erstellung der Dokumentation und Presentation
Personelle Ressourcen	
Geschäftsführer	Festlegung der Anforderungen und code review
Sonstiges	
Büro,Arbeitsplatz	

Tabelle 2 Detaillierte Auflistung Hard - Softwareressourcen

2.4 Entwicklungsprozess

Bevor mit der Durchführung des Projekts begonnen wurde, war es sehr wichtig, einen geeigneten Entwicklungsprozess zu wählen. Damit sollte sichergestellt werden, dass das Projekt ordnungsgemäß durchgeführt wurde und dass sich das Projekt für die Entwicklung der Test Bench-Vorlage eignete.

Ich habe mich für das erweiterte Wasserfallmodell entschieden. Dieses Modell besteht aus strukturierten Phasen, die es meinem Projekt leicht machen. Das erweiterte Wasserfallmodell ermöglicht es, jederzeit zu einer früheren Phase zurückzukehren, um später Korrekturen oder Verbesserungen vorzunehmen. Das erleichtert mir als Projektleiter die Fehlersuche.

Erweitertes Wasserfallmodell

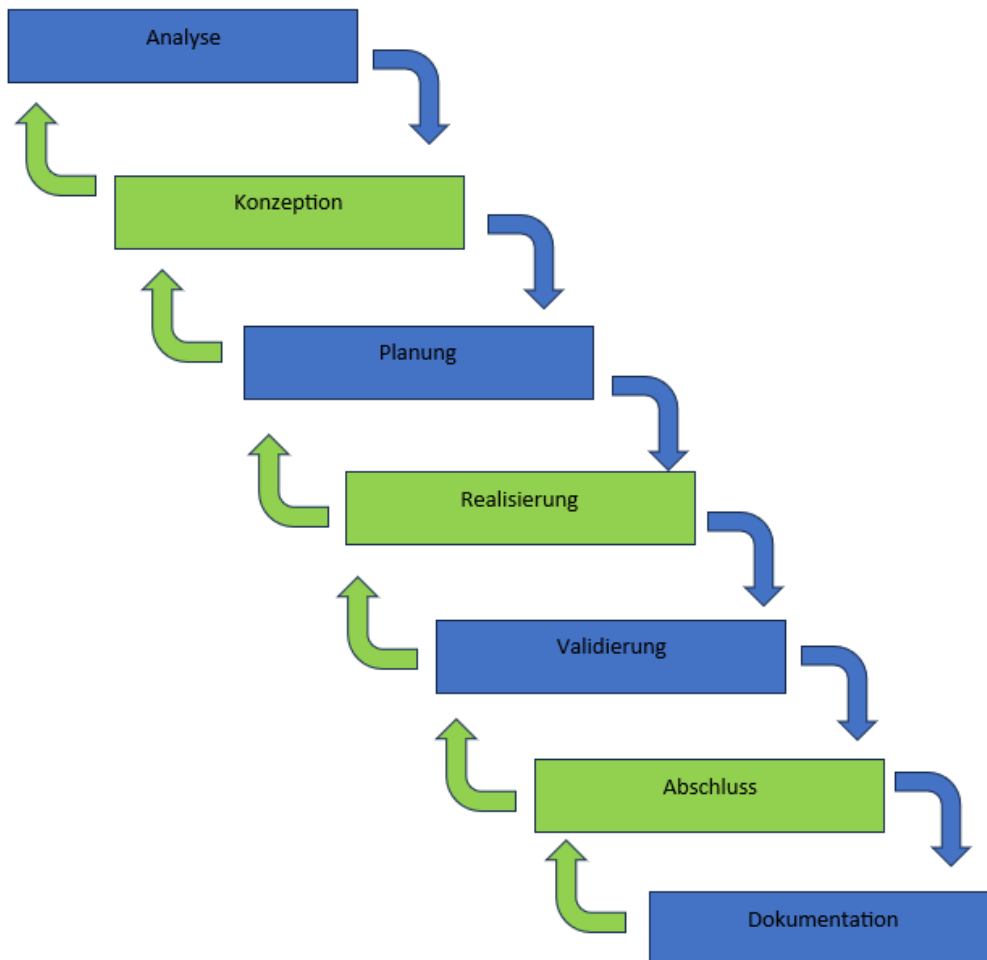


Abbildung 1 erweitertes Wasserfallmodell

3 Analysephase

Nachdem die Projektplanung abgeschlossen ist, stelle ich die Analysephase vor, in der ich den Ist-Zustand (Ist-Analyse) und das Soll-Konzept (Soll-Konzept) erörtere. Im Soll-Konzept finden Sie sowohl fachliche als auch technische Anforderungen im Detail.

Ich werde wirtschaftliche Faktoren berücksichtigen, z.B. ob ich das Projekt ausschließlich in Eigenregie entwickeln werde oder ob externe Unterstützung erforderlich ist. Die Gesamtkosten und der Zeitbedarf für die Investition sind in dieser Phase entscheidend.

Generell wird in diesem Abschnitt analysiert, ob eine Zusammenarbeit mit externen Unternehmen in Betracht gezogen werden sollte, wie die Gesamtkosten des Projekts berechnet werden und wann mit einer Amortisierung der Investitionen zu rechnen ist. Um die Gewichtung dieser verschiedenen Faktoren zu bestimmen, werde ich eine Nutzwertanalyse durchführen.

3.1 Ist-Analyse

Mein Projekt wurde vom Geschäftsführer der Quendro GmbH in Auftrag gegeben und besteht aus insgesamt 80 Stunden Entwicklungs- und Dokumentationszeit.

Es handelt sich um eine Test Bench-Vorlage, die erstellt wurde, um einige der für FPGA-Bausteine entwickelten Anwendungen zu testen. An dieser Stelle möchte ich kurz erklären, was FPGAs sind. FPGA steht für "Field Programmable Gate Array". FPGAs sind Siliziumbausteine, die dynamisch umprogrammiert werden können. Der Hardwareentwurf und der Datenpfad sind genau auf die Arbeitslast des Anwenders abgestimmt. Die sogenannte FPGA-Programmiersprache wird Verilog genannt. Die Erstellung eines ausführbaren Binärprogramms mit FPGA dauert manchmal Stunden. Wenn Sie einen logischen Fehler machen, müssen Sie ihn korrigieren und erneut warten. Das kann Stunden dauern. Um dies zu vermeiden, schreiben FPGA-Entwickler zunächst Test Benches, Das ist vergleichbar mit Unit-Tests in der Softwareentwicklung. So können Sie Ihr Design in Sekundenschnelle testen. Entwerfen Sie es in Sekunden und entscheiden Sie, ob es funktioniert oder nicht.

Ich werde im Projektkonzept eine Grundvorlage für eine Test erstellen. Es wird eine Verilog-Datei als Eingabe haben und eine Test Bench Vorlage als Ausgabe erstellen. Dies ist eine wichtige Anforderung für Vivado (FPGA)-Entwickler. Bisher wurde der Test Prozess von den FPGA-Entwicklern manuell in Excel durchgeführt, was jedoch sehr zeitaufwendig ist. Bei meinem Projekt wird dieser Prozess automatisch durchgeführt. Auf diese Weise wird die Überlastung der Mitarbeiter reduziert und Zeit im FPGA-Entwicklungsteil eingespart. Generell bietet seine Analyse eine strategische Perspektive zur Lösung des Problems und eine klarere Sicht auf die Ziele im Projektverlauf.

3.2 Soll-Konzept

Das Ziel des von mir entwickelten Projekts ist es, eine automatische FPGA-Test Bench-Vorlage zu erstellen, die den komplexen und langwierigen FPGA-Entwicklungsprozess vereinfacht. FPGAs arbeiten normalerweise nicht wie Software. Das Erlangen einer funktionierenden Binärdatei in einem FPGA dauert in der Regel Stunden und kann manchmal noch länger dauern. Daher sollte der erste Schritt in der FPGA-Entwicklung definitiv das Schreiben einer Test Bench (Simulation) sein. Das Schreiben einer Simulation bedeutet, sicherzustellen, dass der von mir geschriebene Code korrekt funktioniert, ähnlich wie ich sicherstellen muss, dass der von mir geschriebene Code auf der Softwareseite durch Unit-Tests korrekt funktioniert. Das Äquivalent zu Unit-Tests in der Softwarewelt ist das Schreiben von Simulationen und Test Benches in der FPGA-Welt.

Die Logik dahinter ist einfach. Das Programm soll Verilog-Code lesen, die Eingänge und Ausgänge erkennen und dann die Vorlage für die entsprechende Test Bench erstellen. FPGA-Entwickler müssen sich nicht mehr manuell mit den Ein- und Ausgängen befassen; sie schreiben lediglich den Codekörper. Den Rest kann ich automatisch generieren.

Zusammenfassend lässt sich sagen, dass das Ziel des Projekts darin besteht, die Module schnell zu testen, um die Simulation zu vervollständigen und zu überprüfen, ob sie innerhalb von Sekunden, vielleicht Minuten, funktionieren oder nicht, ähnlich wie beim Unit-Testing. Sobald sie ihre Tests durchgeführt haben, stellen sie sicher, dass alles richtig funktioniert. Dann kann der eigentliche Prozess, der Stunden dauern kann, um das Binary zu erstellen, gestartet werden. Ein Konzept, wie die in der Ist-Analyse identifizierten Probleme gelöst werden können.

3.2.1 Fachliche Anforderungen

Funktionalen Anforderungen:

FPGA-Entwicklungsunterstützung: Die Lösung sollte speziell die Entwicklung von FPGA-Projekten mit Verilog-Code unterstützen.

Automatische Test Bench-Generierung: Die Lösung sollte die Möglichkeit bieten, automatisch Test Bench-Vorlagen für Verilog-Module zu generieren. Dies erleichtert den FPGA-Entwicklern die Durchführung von Simulationen und Tests.

Flexibilität: Die generierten Test Bench Templates sollten an unterschiedliche Projektanforderungen und Module anpassbar sein.

FPGA-KONSOLENANWENDUNG

FPGA - Automatische Generierung von Test Bench Vorlagen

Integration: Die Lösung sollte sich nahtlos in bestehende FPGA-Entwicklungsumgebungen integrieren lassen, so dass sie nahtlos in den Entwicklungsprozess eingebunden werden kann.

Nicht Funktionalen Anforderungen:

Zeitersparnis: FPGA-Entwickler sparen erhebliche Zeit bei der manuellen Erstellung von Test Benches. Dies hilft ihnen, sich auf andere wichtige Entwicklungsbereiche zu konzentrieren.

Entwicklungsbeschleunigung: Die Möglichkeit, Test Bench schnell zu erstellen und Simulationen durchzuführen, beschleunigt den gesamten Entwicklungsprozess von FPGA-Projekten.

Skalierbarkeit: Die Lösung kann die Produktivität des FPGA-Entwicklungsteams erhöhen und damit die Skalierbarkeit verbessern.

Fehlerreduzierung: Automatisch generierte Test Benches minimieren menschliche Fehler, die bei der manuellen Erstellung auftreten können, und ermöglichen zuverlässigere Tests.

Benutzerfreundlichkeit: Es ist wichtig, dass die Lösung benutzerfreundlich ist und auch von Entwicklern mit wenig Erfahrung leicht verstanden und verwendet werden kann.

Erweiterte Testabdeckung: Mit den Vorteilen der automatischen Test Bench-Generierung können Entwickler umfassendere Tests durchführen, um sicherzustellen, dass die FPGA-Module ordnungsgemäß funktionieren.

3.2.2 Technische Anforderungen

Die Umgebung, in der ich das Projekt entwickle, ist IntelliJ, und es wird als Konsolenanwendung entwickelt. Als Programmiersprache wird Java verwendet. Es wird JDK 17 verwendet. Xilinx Vivado ist die Umgebung für FPGA (Field-Programmable Gate Array) Design. Mein Projekt nimmt den Verilog-Entwurf und verarbeitet ihn so, dass er für Vivado geeignet ist. Der Verilog-Code wird für die Simulation des Entwurfs verwendet. Die Hardwarebeschreibungssprache Verilog ermöglicht es, das Design in einer computergestützten Simulationsumgebung zu testen. Die so erzeugten Testbenches werden im Vivado-Simulator verwendet.

3.3 Wirtschaftlichkeitsanalyse

Die im SOLL-Konzept vorgeschlagenen Lösungen boten nicht nur technische Vorteile, sondern führten auch zu einer Verringerung der Verwaltungsaufgaben für den Geschäftsführer. Die sich daraus ergebenden finanziellen Vorteile werden in den folgenden Abschnitten im Einzelnen erläutert.

3.3.1 Make or Buy-Entscheidung

Da die Quendro GmbH auf Softwareentwicklung und -Design spezialisiert ist, spielt die interne Softwareentwicklung eine wichtige Rolle bei der Entwicklung von Fachwissen durch Erfahrung und ermöglicht es dem Unternehmen außerdem, Verbesserungen entsprechend den Bedürfnissen des Unternehmens in Übereinstimmung mit den Wünschen des Geschäftsführers vorzunehmen. Im Gegensatz dazu kann ein ausgelagertes Projekt komplexer und kostspieliger sein und mehr Zeit erfordern, um die erforderlichen Ergänzungen rechtzeitig vorzunehmen und schnelle Lösungen zu liefern. Eine intern zu entwickelnde Software ist in Bezug auf die Möglichkeit, sie zu verbessern, immer vorzuziehen.

3.3.2 Projektkosten

Personalkosten:

Nachfolgend finden Sie eine Übersicht über die Berechnung der Personalkosten. Da die exakten Stundensätze nicht verfügbar sind, werden pauschale Sätze berücksichtigt. Die Tabelle unten enthält eine detaillierte Auflistung und Erläuterung der einzelnen Kostenpositionen.

Vorgang	Mitarbeiter	Zeitaufwand	Personalkosten
Entwicklungskosten der Projektarbeit	1 Auszubildende	80h	(80 Std * 7,5 €/Std) 600 €
Hilfestellung bei Problemen	Geschäftsführer	2h	(2 Std*120,00 €/Std) 240,00€
Aufsicht bei Projektplanung und Rückfragen	Geschäftsführer	2h	(2 Std*120,00 €/Std) 240,00€
Abnahme des Projekts	Geschäftsführer	2h	(2 Std*120,00 €/Std) 240,00€
Gesamtkosten			1320 €

Tabelle 3 Personalkosten

Besondere Aufmerksamkeit sollte den tatsächlichen Personalkosten gewidmet werden, die dem Projektleiter und den Kontaktpersonen während der Problemlösungs- und Feedback-

Runden entstehen (siehe Tabelle 3). Der Praktikant wird während des Projekts insgesamt 80 Stunden arbeiten. In der vorstehenden Tabelle sind die Gesamtkosten mit 1.320 € angegeben. Das durchschnittliche Monatsgehalt für einen Auszubildenden beträgt € 1.200,00. 240 Arbeitstage in einem Jahr.

Azubi Stundensatz berechnen = 60€/Tag8 h=7,5€.

$$\begin{aligned}
 8 \frac{\text{h}}{\text{Tag}} \cdot 240 \frac{\text{Tage}}{\text{Jahr}} &= 1920 \frac{\text{h}}{\text{Jahr}} \\
 1.200 \frac{\text{€}}{\text{Monat}} \cdot 12 \frac{\text{Monate}}{\text{Jahr}} &= 14.400 \frac{\text{€}}{\text{Jahr}} \\
 \frac{14.400 \frac{\text{€}}{\text{Jahr}}}{1920 \frac{\text{h}}{\text{Jahr}}} &\approx 7,5 \frac{\text{€}}{\text{h}}
 \end{aligned}$$

Der Stundensatz des Geschäftsführers beträgt 120,00 €. Der Geschäftsführer hat 6 Stunden für das Projekt aufgewendet.

Der Praktikant (ich) erhält, nehmen wir einmal an, 7,5 € pro Stunde.

Die Berechnung wird wie folgt durchgeführt:

Formel: Personalkosten = Anzahl Mitarbeiter * Zeit * Stundensätze

Materialkosten:

Während der Projektdurchführung entstanden keine wesentlichen Kosten für den Erwerb spezifischer Hardware, Software oder Lizenzgebühren, da diese größtenteils bereits im Unternehmen vorhanden waren. Die Kosten konzentrierten sich hauptsächlich auf die verwendeten Materialien und die Grundausstattung am Arbeitsplatz, einschließlich des Büros und der speziell für das Projekt angeschafften Geräte wie Maus und Tastatur.

Für Tools wie IntelliJ IDEA, Microsoft Office Word 365 und das Betriebssystem Windows 10 wurden Kosten für bis zu 80 Arbeitsstunden berücksichtigt. In der nachfolgenden Tabelle sind die resultierenden Sachkosten zuerst aufgelistet und anschließend den Personalkosten hinzugefügt, um das Gesamtergebnis zu ermitteln.

Sachmittel	Zeit	Ressourcen
Materialkosten	80 h	15,00 €
Energiekosten	80 h	15,00 €
Bürraum		100,00 €
Arbeitsmaterial		7,00 €
Software		9,00 €
Gesamtkosten		146,00 €

Tabelle 4 Sachmittelkosten

In der obenstehenden [Tabelle 4](#) setzen sich die Sachmittelkosten aus zwei Hauptkomponenten zusammen: den Ressourcenkosten und den Gemeinkosten, wie Miete und Strom. Alle weiteren Kosten setzen sich aus den Materialkosten, den Arbeitsmaterialien und den Softwarekosten zusammen. Um die genauen Projektkosten besser erkennbar zu machen, wurden alle Kosten auf die Projektdauer von 80 Stunden umgerechnet.

Die Lizenzkosten für die Materialien umfassen die Ausgaben für den Laptop und den Bildschirm. Die Energiekosten wurden ebenfalls auf 80 Stunden umgerechnet und belaufen sich auf insgesamt 15,00 €.

Die monatliche Miete für die Büroräume von Quendro beträgt 200,00 €, was zu Kosten von 10 € pro Tag führt. Da das Projekt für 80 Stunden geplant ist, habe ich zunächst die Anzahl der Tage berechnet.

Die Kosten für das Arbeitsmaterial bestehen aus den Lizenzkosten für Microsoft Word (monatlich)(7.00€). Die Gesamtkosten des Projekts ergeben sich aus der Summe der Personalkosten und der Kosten für Ressourcen. Die Gesamtkosten des Projekts belaufen sich auf EUR 1.466,00, die sich aus Personalkosten in Höhe von EUR 1320,00 und Ressourcenkosten in Höhe von EUR 146,00 zusammensetzen.

3.3.3 Amortisationsdauer

Zeiteinsparung bildet die Grundlage für die Berechnung der Amortisationsdauer. Obwohl eine genaue Zeiteinsparung nicht konkret berechnet werden kann, wurde aufgrund von Gesprächen mit den Geschäftsführern eine geschätzte Zeiteinsparung ermittelt, und die Amortisationsdauer kann wie folgt berechnet werden:

Die Gesamtkosten des Projekts betragen 1.466,00 € (Personalkosten + Sachmittel). Durch den Einsatz der Software können sowohl der Mitarbeiter als auch der Geschäftsführer Zeit sparen. Bei der Berechnung der Amortisationszeit gehe ich davon aus, dass mein Unternehmensleiter Herr Kocal den Prüfstand 1 Mal pro Woche manuell erstellt. Ich rechne daher mit 4 Stunden pro Monat. 60 Minuten manuelles Schreiben der Testdatei pro Datei = 120 Euro

4 Stunden Prüfung in einem Monat: $4\text{Std} \times 60\text{ min} = 240\text{ min}$

240 Minuten \times 12 Monate = 2880 Minuten.

$2880\text{ min} / 60\text{ min} = 48\text{ Stunden}$ jährliche manuelle Prüfzeit.

Wenn der Geschäftsführer vor der Automatisierung 48 Stunden pro Jahr für das Testen aufwendet und nach der Automatisierung nur noch 16 Stunden pro Jahr, ergibt sich eine jährliche Einsparung von 32 Stunden. Der automatische Testvorgang dauert etwa 20 Minuten. Das sind $20 \times 4 = 80$ Minuten pro Monat. Die jährliche Zeit beträgt $80\text{ min} \times 12\text{ Monate} = 960\text{ Minuten}$. Das heißt $960\text{ min} / 60 = 16\text{ Stunden}$.

Jährliche Kosteneinsparung:

Einsparung in Jahresstunden = 48 Stunden - 16 Stunden = 32 Stunden.

Kosteneinsparung pro Stunde = 120 € (da die Kosten des Geschäftsführers pro Stunde 120 € betragen).

Jährliche Gesamtkosteneinsparung = 32 Stunden \times 120 €/Stunde = 3.840,00 €.

Hinzu kommen noch Sachmittelkosten in Höhe von 146 €. Die Gesamtkosteneinsparung ergibt sich wie folgt:

Gesamte jährliche Kosteneinsparung = 3.840,00 € + 146,00 € = 3.986,00 €.

Amortisationszeit:

Amortisationszeit = Projektkosten / jährliche Einsparungen.

Amortisationszeit = 1.466,00 € / 3.986,00 € \approx 0,37 Jahre.

Dies entspricht etwa 4,4 Monaten. ($0,37 \text{ Jahre} \times 12 \text{ Monate/Jahr} = 4,44 \text{ Monate}$)

Mit anderen Worten: In etwa 4,4 Monaten werden sich die Projektkosten durch die eingesparten Kosten amortisiert haben.

3.4 Nutzwertanalyse

Die Nutzwertanalyse kann als ein Werkzeug definiert werden, das uns hilft herauszufinden, wie gut etwas funktioniert. In diesem Fall hilft uns die FPGA-Test Bench zu verstehen, wie effektiv meine Template-Lösung den Entwicklungsprozess beschleunigt und Fehler minimiert. Ich kann diese Lösung dann mit anderen Entwicklungsmethoden vergleichen.

Ich habe meine Anwendung so konzipiert, dass sie bei der Entwicklung von FPGA automatisch Test Bench Vorlagen erstellt. Auf diese Weise können Test Bench Vorlagen schneller und mit geringerer Fehlertoleranz erstellt werden als manuell erstellte Skripte, die viel Zeit in Anspruch nehmen. Wie nützlich werden die in meinem Projekt entwickelten automatischen Test Bench Vorlagen im Vergleich zu manuell erstellten Test Bench Vorlagen sein?

Um diese Frage zu beantworten, habe ich eine Nutzenanalyse anhand von vier Kriterien erstellt. Diese vier Kriterien sind: Effektivität, Effizienz, Zuverlässigkeit und Benutzerfreundlichkeit.

Die Leistung der Anwendung habe ich anhand dieser Kriterien bewertet und sie in der folgenden Tabelle dargestellt:

		Test Bench Vorlage		manuelle Test	
		Bewertung		Bewertung	
Kriterium	Gewichtung	ungewichtet	gewichtet	ungewichtet	gewichtet
Effektivität	20%	4	0,8	3	0,6
Effizienz	30%	5	1,5	3	0,9
Zuverlässigkeit	20%	4	0,8	2	0,4
Benutzerfreundlichkeit	30%	5	1,5	2	0,6
Gesamt	100%		4,6		2,5

Tabelle 5 Nutzwertanalyse

Effektivität: Meine FPGA-Test Bench-Vorlagenlösung ist besser als die alten Methoden, weil sie schneller und weniger fehleranfällig ist. Das bedeutet, dass sie die Arbeit effektiver erledigen kann.

Effizienz: Meine Lösung spart Zeit und ist leicht zu verwenden. Im Gegensatz dazu benötigen die traditionellen Methoden mehr Zeit und Schulung, um sie zu nutzen.

Zuverlässigkeit: Meine Lösung ist äußerst zuverlässig, da sie präzise Test Benches generiert, um die Fehlerminimierung zu unterstützen.

Benutzerfreundlichkeit: Die Bedienung meine Lösung ist benutzerfreundlich und erfordert keine spezielle Schulung. Im Gegensatz dazu könnten traditionelle Methoden komplexer sein.

Ergebnis: Meine FPGA-Test Bench-Vorlagenlösung erzielte eine Gesamtwertung von 4,6 Punkten, während traditionelle manuelle Methoden nur eine Gesamtwertung von 2,5 Punkten erhielten. Aufgrund dieser Nutzwertanalyse scheint meine Lösung eine überlegene Option zu sein, die den FPGA-Entwicklungsprozess optimiert und die Effizienz steigert.

3.5 Anwendungsfälle

In der Analysephase habe ich ein Use Case Diagramm erstellt, um zu zeigen, wie die Software arbeiten und funktionieren soll. Dieses Diagramm identifiziert die verschiedenen Aufgaben und Aktionen und bietet eine Grundlage für die Definition der Systemanforderungen. Es zeigt, wie der Benutzer mit dem System interagiert und was die Anwendungsfälle sind.

1. das Starten der Test Bench-Anwendung: Dieser Schritt öffnet die Test Bench-Anwendung und startet den FPGA-Entwicklungsprozess.
2. eine Liste der FPGA-Module und -Komponenten anzeigen: Es wird eine Übersicht über die verfügbaren FPGA-Module und -Komponenten angezeigt.
3. die Auswahl des FPGA-Moduls: ich wähle aus der Liste das spezifische FPGA-Modul oder die Komponente aus, mit der ich arbeiten will.
4. Erstellen eines Prüfstandes für das Modul: Hier kommt die Schlüsselkomponente der Anwendung ins Spiel. Die Anwendung generiert automatisch eine Test Bench-Vorlage auf der Grundlage des ausgewählten Moduls, wodurch die manuelle Erstellung entfällt.

5. Anpassung der Test Bench je nach Bedarf: Die generierte Test Bench kann entsprechend den Anforderungen des Entwicklers angepasst werden. Dies kann das Hinzufügen von benutzerdefinierten Testfällen, das Ändern von Eingabeparametern oder das Hinzufügen von Überwachungspunkten umfassen.

6. Starten der Simulation des FPGA-Moduls: Sobald die Test Bench konfiguriert ist, starte ich die Simulation des FPGA-Moduls. Dies ermöglicht dem Entwickler, die Funktionalität des Moduls zu überprüfen und zu überwachen.

7. Simulationsergebnisse anzeigen und auf Fehler überprüfen: ich zeige Simulationsergebnisse, Ausgangswerte und mögliche Fehlermeldungen. Durch die Überprüfung dieser Ergebnisse stelle ich sicher, dass das Modul wie erwartet funktioniert.

8. Anpassungen und Neustart der Simulation, falls erforderlich: Wenn während der Simulation Fehler oder unerwartetes Verhalten auftreten, kann ich Anpassungen an der Test Bench vornehmen und die Simulation neu starten, um sicherzustellen, dass die Probleme behoben werden.

Bei diesem Verfahren wurde der Entwicklungsprozess beschleunigt, weil ich die Test Bench-Anwendung manuell erstellt und die FPGA-Module effizient gesteuert habe.

Im Anhang A3 finden Sie ein Use Case Diagramm, in dem die Anwendungsfälle dargestellt sind.

3.6 Qualitätsanforderungen

Im Projekt habe ich sehr darauf geachtet, dass die Quendro GmbH mit der Qualität meiner Arbeit zufrieden ist. Ich habe untersucht, wie gut meine Produkt funktioniert und festgelegt, welche Standards es erfüllen muss, damit es gut funktioniert und zuverlässig ist. Ich orientierte mich an international anerkannten Qualitätsstandards wie ISO 9126 (jetzt bekannt als ISO 25000), die Anforderungen an die Softwarequalität und deren Bewertung definieren.

	Sehr gut	gut	Normal	Nicht relevant
Funktionalität				
Angemessenheit		x		
Richtigkeit	x			
Ordnungsmäßigkeit		x		
Sicherheit	x			
Zuverlässigkeit				
Fehlertoleranz	x			
Wiederherstellbarkeit	x			
Benutzerbarkeit				
Verständlichkeit	x			
Erlernbarkeit	x			
Bedienbarkeit	x			
Effizienz				
Zeitverhalten	x			
Änderbarkeit				
Stabilität		x		
Prüfbarkeit	x			
Modifizierbarkeit		x		
Übertragbarkeit				
Anpassbarkeit		x		
Austauschbarkeit		x		

Tabelle 6 Softwarequalität

In diesem Zusammenhang habe ich sechs wichtige Aspekte berücksichtigt:

Funktionalität, Zuverlässigkeit, Benutzerfreundlichkeit, Benutzerfreundlichkeit, Effizienz, Modifizierbarkeit und extreme Portabilität. Die Anwendung wurde in Bezug auf bestimmte Merkmale wie Genauigkeit, Sicherheit, Fehlertoleranz, Wiederherstellbarkeit, Benutzerfreundlichkeit, Effizienz und Testbarkeit mit "sehr gut" bewertet. Diese Eigenschaften wurden auch vom Generaldirektor als besonders wichtig erachtet. Andere Merkmale wie Angemessenheit, Regelmäßigkeit, Stabilität und dergleichen wurden ebenfalls bewertet, aber als weniger kritisch angesehen und daher mit "gut" bewertet. Ändert sich die Meinung des Verwalters, wird der Antrag entsprechend angepasst.

3.7 Lastenheft/Fachkonzept

Nachdem die Analysephase abgeschlossen war, habe ich zusammen mit dem Geschäftsführer eine Vereinbarung getroffen, in der alle Anforderungen und Bedingungen festgehalten sind. In dieser Vereinbarung sind sowohl die verschiedenen Aufgaben, die die Anwendung erfüllen muss, als auch alle anderen Anforderungen und Konditionen enthalten, die von der Anwendung erfüllt werden müssen. Eine Zusammenfassung der Funktionen aus dieser Vereinbarung finden Sie im Anhang A2.

4 Entwurfsphase

4.1 Zielplattform

Die Auswahl der Zielplattform für die Entwicklung einer automatisierten FPGA-Test Bench-Vorlage ist wichtig, um die Effizienz und Effektivität des Projekts zu gewährleisten, und muss sorgfältig geprüft werden:

Programmiersprache: Verilog/VHDL: Da das Ziel des Projekts die Analyse von Verilog-Code und die Erstellung von Prüfständen ist, liegt die Verwendung von Verilog oder VHDL als Hauptprogrammiersprachen auf der Hand. Diese Sprachen sind der Industriestandard für die FPGA-Entwicklung. Die Programmiersprache Java wird in meiner Anwendung in IntelliJ IDEA verwendet.

Datenbank: Dateibasiert: Test Bench-Vorlagen und zugehörige Informationen können in Dateien gespeichert werden, was die Portabilität und einfache Verwaltung erleichtert.

Client/Server-Architektur: Eigenständiges Werkzeug: Die Anwendung kann als eigenständiges Desktop-Tool für den Einsatz durch FPGA-Entwickler im Feld entwickelt werden.

4.2 Geschäftslogik

Die Geschäftslogik der Anwendung FPGA - Automatische Generierung von Test Bench-Vorlagen kann ich wie folgt beschreiben:

Im Projektkonzept werde ich eine Basisvorlage für eine Test Bench-Vorlage erstellen. Es wird eine Verilog-Datei als Eingabe erhalten und eine Test Bench-Vorlage als Ausgabe generieren.

Generell besteht die Anwendung aus 3 Teilen.

1. `mainProgram.java`: Diese Datei ermöglicht den Start des Programms und die Verarbeitung von Benutzereingaben. Sie prüft die Kommandozeilenargumente, erzeugt ein Objekt der Klasse ``FPGAModule`` und gibt die Ergebnisse in ``demo_test.v`` aus. Die Rolle dieser Datei scheint bei den Codebeispielen eher ein Kontrollmechanismus zu sein.

2. `FPGAModule.java`: Diese Klasse repräsentiert ein FPGA-Modul. Sie enthält den Namen eines FPGA-Moduls, die darin enthaltenen Parameter und andere Eigenschaften. Die Methode ``getModuleParameters()`` verwendet die Klasse ``ModuleParameter``, um die Modulparameter zu erhalten. Auch die Methode ``toString()`` generiert die Details des Moduls in einem String.

3. `ModuleParameter.java`: Diese Klasse stellt die Parameter des Moduls dar. Parameter haben Eigenschaften wie Namen, Längen, Typen. Diese Klasse enthält die Details zu einem bestimmten Parameter.

Ich werde mit den Ein- und Ausgängen des Moduls arbeiten, die ich von vielen Modulen im FPGA benötige.

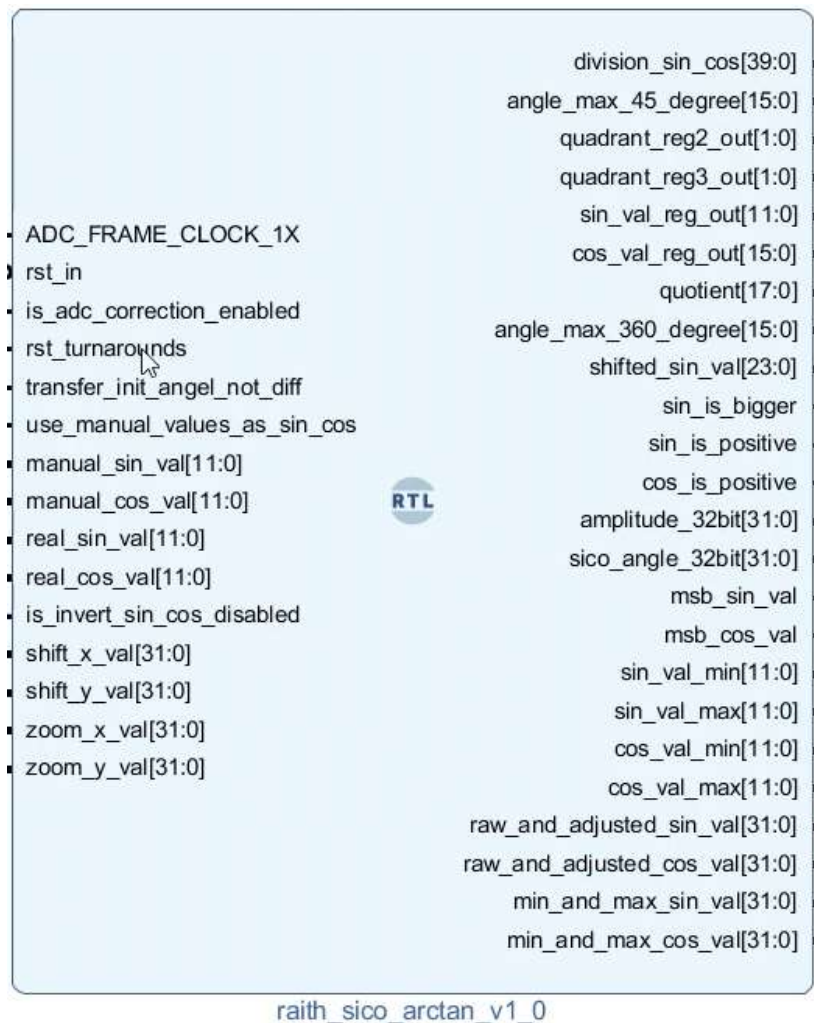


Abbildung 2 Beispiel eines FPGA Module und Inputs-Outputs

Dieses Bild zeigt die Ein- und Ausgänge des FPGA-Modul. Die Aufgabe, die ich in meiner Anwendung ausführen werde, besteht darin, die Eingänge und Ausgänge in die von mir angegebene Datei im gewünschten Format zu schreiben.

Eingabedaten lesen und auf der Konsole ausgeben:

Daher muss ich in meine Anwendung den Code zum Lesen der Verilog-Datei in das Hauptprogramm schreiben. Zunächst ist es möglich, die Verilog-Datei durch einen Befehl von der Kommandozeile zu erhalten. Dazu muss zuerst `cd` (change directory) von der Kommandozeile aus gemacht werden. Dann wird der Pfad der Datei geschrieben. Auf diese Weise wird der Abruf der Datei durchgeführt.

Modulname, Eingänge und Ausgänge identifizieren :

- Modulname identifizieren. Er kommt nach dem Schlüsselwort "module".

Nennen ich meine Modul zum Beispiel agc_module. Ttb_agc_module ist ein Name, den ich gegeben habe, er kommt von Test Bench. Ttb Unterstrich, dann ist der gleiche Name des Moduls agc_module. Daher suche ich zuerst nach dem Schlüsselwort „module“ in der Datei. Wenn ich das Wort module finde, ist das nächste Wort der Name meines Moduls. Ich nehme diesen Namen und schreibe ihn, dann schreibe ich „ttb_“, und füge den Namen des Moduls hier wieder ein. Auf diese Weise gebe ich meinem Test Bench einen Namen. Wenn das Modul also agc_module ist, dann wird der Name agc_module ttb_agc_module sein.

- Identifizieren Sie Inputs und Outputs.

Beschreibung:

- Lesen Sie die Datei zeilenweise bis zum ersten "). Das bedeutet das Ende der Ein-/Ausgabe-Deklarationen.

- Wenn eine Zeile das Schlüsselwort "input" enthält, handelt es sich um eine Eingabe.
- Wenn eine Zeile das Schlüsselwort "output" enthält, handelt es sich um eine Ausgabe.

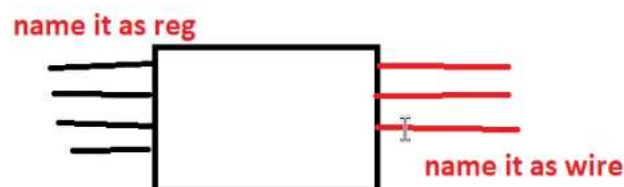


Abbildung 3 Benennung den inputs und outputs

Ich führe diesen Vorgang in der fpga-Modulkasse durch. Ich muss spezielle Methoden erstellen, die diese Aufgabe erledigen kann. Ich muss die Eingänge innerhalb des Moduls als „reg“ ausgeben. Ich muss die Ausgänge ausgeben, indem ich sie in „wire“ konvertiere. In der Funktion, die ich dafür erstellen werde, kann ich die Werte in einer Zeichenkette mit einer Arrayliste aufbewahren. In Fällen, in denen ich nicht weiss, wie viele Werte kommen werden, bietet mir die Arrayliste eine praktische Lösung. Außerdem kann ich in einer for-Schleife nach Eingabe

und Ausgabe suchen. Durch die Verwendung von Enums kann ich den Code klarer und verständlicher gestalten. Entwurf der Klassendefinitionen:

```
enum Direction {
    IN,
    OUT,
    INOUT
};
```

Hier füge ich hier ein Beispielfragment aus dem Quellcode der Verilog-Datei mit Eingängen und Ausgängen.

```
module get_abs_pos_state_machine(
    input wire clk,
    input wire rst,
    input wire init_state_machine,
    input wire hls_done,
    input wire hls_ready,
    input wire [31:0] axis1_hw_counter,
    input wire [31:0] axis1_set_position_part1,
    input wire [31:0] axis1_set_position_part2,
    input wire [31:0] axis1_counts_per_m,
    input wire [31:0] axis2_hw_counter,
    input wire [31:0] axis2_set_position_part1,
    input wire [31:0] axis2_set_position_part2,
    input wire [31:0] axis2_counts_per_m,

    input wire [63:0] selected_axis_hls_calculated_abs_pos,

    output reg start_hls_calculations, //start absolute
    output reg [2:0] state,
    output reg [31:0] selected_axis_hw_counter,
    output reg [31:0] selected_axis_set_position_part1,
    output reg [31:0] selected_axis_set_position_part2,
    output reg [31:0] selected_axis_counts_per_m,

    output reg [63:0] axis1_hls_calculated_abs_pos,
    output reg [63:0] axis2_hls_calculated_abs_pos
);
```

Abbildung 4 Verilogcode

- Wenn diese Zeile auch "[,]" enthält, bedeutet dies, dass es sich um einen Bus handelt und ich seine Länge identifizieren muss.

FPGA-KONSOLEANWENDUNG

FPGA - Automatische Generierung von Test Bench Vorlagen

Zum Beispiel: Wenn es eine Zeile wie "input wire [11:0] sin_val_min" gibt, bedeutet das, dass es ein Bus ist und seine Länge 12 ist, also 11 + 1. Ich kann diesen Vorgang in der Modulparameterklasse mit Codes wie if Bedingung und „contains“ Methode durchführen.

- Identifizieren Sie Takt- und Reset-Pins.

Dieses Mal muss ich die Clock- und Reset-Pins finden. Wenn es clock, clk, _clk enthält, wird dieses Mal isClock true(wahr) sein.

Als Beispiel-Eingangsstifte, die einen von ihnen enthalten "clock, clk, _clk..." - Für Reset-Eingangsstifte, die einen von ihnen "reset, rst..." enthalten, ist unten zu sehen:

```
clk = 0;
rst = 1;
init_state_machine = 0;
hls_done = 0;
hls_ready = 0;
```

Abbildung 5 clock und reset code Beispiel

Im Modul Parameterklasse definiere ich die Eigenschaften der Parameter, ich kann die wichtigen Variablen wie folgt benennen;

```
class ModuleParameter{

    String name;

    int length;

    Direction direction; //it can be one of in, out, inout

    boolean isClock;

    boolean isReset;

}
```

Durch die Erstellung einer speziellen Methode in der FPGA-Modulkasse kann ich die Modulparameter extrahieren und in einer Array-Liste speichern.

```
import java.util.ArrayList;

class FPGAModule{
```



```
String name;  
  
ArrayList<ModuleParameter> moduleParameters;  
  
};
```

3- Erzeugen eines Instanziierungsblocks eines verwandten Moduls

Bei der Instanziierung, sage ich, es gibt 10 Zeilen, wird jede der 10 Zeilen die gleiche Eigenschaft haben; Sie beginnt mit einem Punkt „.“, am Ende jeder Zeile steht ein Komma „ , „ und außer in der letzten Zeile, hier ist es egal, ob es sich um eine Eingabe oder eine Ausgabe handelt, d.h. der Name des Parameters sollte der Name des Parameters in Klammern „ () „ sein. Jede Zeile steht in einer neuen Zeile. Es sollte eine Ausgabe wie im Beispiel hier erfolgen.

```
.clk(clk),  
.rst(rst),  
.init_state_machine(init_state_machine)
```

Abbildung 6 Beispiel eines Instanziierungsblocks

4- Generierung der Ausgabedatei als Basis-Test Bench.

Dies ist der letzte Teil des Test Bench-Skripts, der der Grund für die Erstellung der Anwendung ist. Ich erstelle eine Ausgabedatei, die den Namen ``./demo_test.v`` tragen kann, ich benenne sie selbst und schreiben das Objekt `fpgaModule` in diese Datei.

Das folgende UML-Diagramm zeigt die Interaktion zwischen dem Hauptprogramm, dem fpga-Modul und den Modulparameterklassen des in OOP geschriebenen Codes.

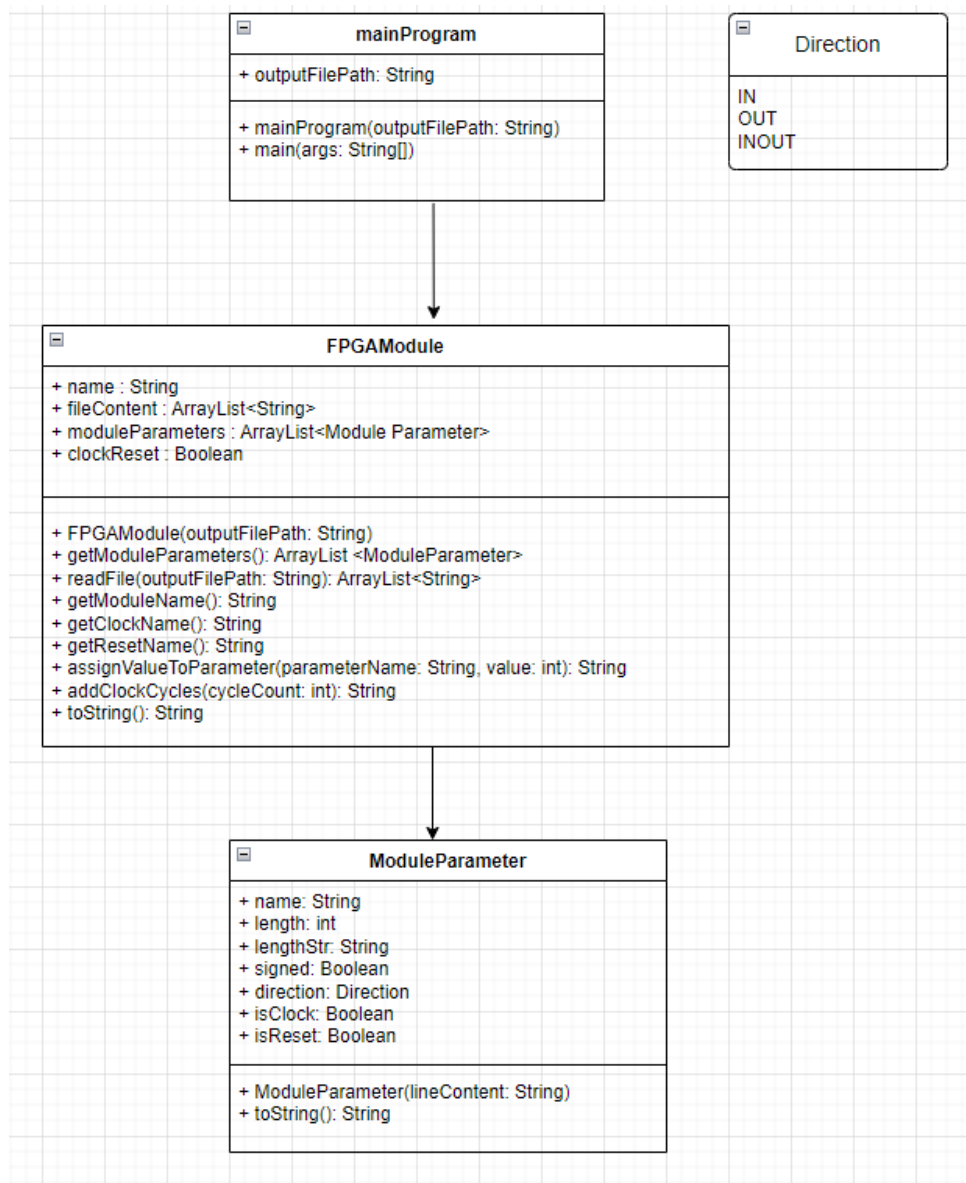


Abbildung 7 UML Klassendiagramm

4.3 Maßnahmen zur Qualitätssicherung

Ich habe mehrere Schritte unternommen, um die hohe Qualität meiner Durchführung zu gewährleisten:

1. Regelmäßige Treffen: Ich hielt regelmäßige Treffen ab, um abgeschlossene Aufgaben zu überprüfen und die Ergebnisse zu diskutieren. So konnte ich Funktionen und Code ständig

überprüfen und verbessern, um sicherzustellen, dass die App immer auf dem neuesten Stand ist.

2. Code-Reviews: Gemeinsam mit meinem Ausbilder habe ich den Code in regelmäßigen Abständen überprüft, um nicht nur die technische Korrektheit, sondern auch die technische Umsetzung des Codes zu beurteilen. Dies half mir, die Qualität des Codes hoch zu halten.

3. Manuelles Testen: Konkret führte ich Bewertungen und kurze Besprechungen durch, um sicherzustellen, dass die Anwendung alle Anforderungen erfüllt.

Durch diese Maßnahmen konnte ich sicherstellen, dass die Qualität meine Anwendung während des gesamten Entwicklungsprozesses hoch blieb. Ich war in der Lage, Fehler frühzeitig zu erkennen und zu beheben, was Zeit und Ressourcen sparte und eine solide Grundlage für die weitere Entwicklung schuf.

4.4 Pflichtenheft/Datenverarbeitungskonzept

Nach Abschluss der Entwurfsphase habe ich ein Pflichtenheft erstellt, das auf den im Lastenheft festgelegten Anforderungen aufbaut. Dieses Dokument enthält detaillierte technische Anforderungen für das Projekt. Es dient dazu sicherzustellen, dass am Ende des Projekts überprüft werden kann, ob alle festgelegten Anforderungen erfolgreich umgesetzt wurden. Ein Auszug aus diesem Pflichtenheft findet sich im Anhang A4 meines Projektdokuments.

5 Implementierungsphase

5.1 Implementierung der Geschäftslogik

Allgemeine Beschreibung des Projekts;

In diesem Projekt habe ich die Prinzipien der objektorientierten Programmierung (OOP) angewandt. Ich habe Klassen und Methoden in dem Projekt erstellt, um das Design modular und verständlich zu gestalten. Die Reihenfolge und die Geschäftslogik dieser Codes in der Anwendung sind im Allgemeinen wie folgt:

1. Das Hauptprogramm; enthält den Einstiegspunkt des Programms. Es nimmt einen Ausgabepfad von der Kommandozeile entgegen, erstellt ein ``FPGAModule``-Objekt und schreibt den Modulentwurf in eine Ausgabedatei.

2. Die Klasse `ModuleParameter`, die die Parameter des FPGA-Moduls darstellt. Jeder Parameter hat Eigenschaften wie Name, Länge, Beschriftung, Richtung (Eingang/Ausgang), Takt

und Reset-Signal. Die Eigenschaften der Parameter können über Methoden eingestellt werden und es können Textstrings erstellt werden, die die Parameter darstellen.

3. Die Klasse `FPGAModule` repräsentiert das Design des FPGA-Moduls. Sie enthält den Modulnamen, den Dateiinhalt, die Modulparameter, das Taktsignal, das Reset-Signal und die Methoden, die zur Erstellung des Moduldesigns verwendet werden.

In diesem Projekt wurde eine modulare Struktur unter Verwendung eines objektorientierten Programmieransatzes geschaffen. Jede Klasse bietet Funktionalität, indem sie sich auf ihre eigenen Aufgaben konzentriert.

Die Klasse `FPGAModule` repräsentiert beispielsweise das FPGA-Modul, während die Klasse `ModuleParameter` die Modulparameter darstellt. Darüber hinaus führt die Klasse `mainProgram` Funktionen wie das Starten des Programms und die Verarbeitung von Benutzereingaben aus.

Modularer Struktur bedeutet, dass ein Programm oder System in Teile unterteilt ist und jeder Teil eine bestimmte Verantwortung oder Funktion übernimmt. Diese Struktur dient dazu, die Komplexität zu beherrschen, die Wartung zu erleichtern, Fehler einzugrenzen und die Teamarbeit zu verbessern. Der modulare Aufbau hat sich in meinem Projekt als großer Vorteil erwiesen, da er die Entwicklung großer und komplexer Projekte erleichtert, indem er sie in kleinere und besser handhabbare Teile aufteilt.

HAUPTPROGRAMM

Dieses Programm führt eine Operation durch, bei der es eine FPGA-Designdatei liest, um die Details und Eigenschaften eines FPGA-Moduls zu verstehen.

1. Zuerst beginnt das Programm mit der `main`-Funktion.

```
if(args.length != 1) {  
    System.out.println("Usage: java mainProgram <outputFilePath>");  
    return;  
}
```

Mit dem obenstehenden Code wird ein Dateipfadargument von der Befehlszeile akzeptiert. In diesem Dateipfad sollte das FPGA-Design gefunden werden. Ich gebe den Dateipfad in der Befehlszeile an, beispielsweise;

```
`cd C:\JavaProjects\testbench 2\TestBenchGenerator\src\main\java`.
```

FPGA-KONSOLEANWENDUNG

FPGA - Automatische Generierung von Test Bench Vorlagen

Wenn kein korrektes Argument (Dateipfad) von der Befehlszeile übergeben wird, gibt das Programm die Verwendungsanweisungen auf der Konsole aus und beendet sich. Wenn der Benutzer mehrere Argumente übergibt oder überhaupt keine Argumente übergibt (d.h., einen Verwendungsfehler macht), gibt die Zeile `System.out.println("Usage : java mainProgram <outputFilePath>");`` dem Benutzer die richtige Verwendungsanweisung aus und das Programm endet mit ``return;``.

2. In der Variable ``outputFilePath`` speichere ich den Dateipfad, der von der Befehlszeile eingegeben wurde.

3. Ein Objekt ``fpgaModule`` der Klasse ``FPGAModule`` wird erstellt: ``FPGAModule fpgaModule = new FPGAModule(outputFilePath);``. Dieses Objekt ist eine Instanz der Klasse ``FPGAModule`` und repräsentiert die Details der FPGA-Design-Datei, indem es die Struktur der Klasse ``FPGAModule`` verwendet. Das erstellte Objekt wird als "fpgaModule" bezeichnet, damit es später im Programm verwendet werden kann. Dieses Objekt enthält Informationen wie den Modulnamen des FPGA-Designs, Ein- und Ausgangsparameter sowie Takt- und Rücksetzungsstifte.

4. Die Zeile `System.out.println(fpgaModule);`` gibt das erstellte ``fpgaModule``-Objekt aus, indem es die Details des Moduls auf der Konsole anzeigt.

5. Zuletzt erstelle ich mithilfe eines ``PrintWriter``-Objekts eine Ausgabedatei mit dem Namen `"/demo_test.v"` und schreibe das ``fpgaModule``-Objekt in diese Datei. Dieser Schritt dient dazu, eine grundlegende Test Bench-Datei für das FPGA-Design zu erstellen.

Dieses Programm wird verwendet, um den Inhalt einer gegebenen FPGA-Design-Datei zu analysieren und den Modulnamen, die Eingangs-/Ausgangsparameter sowie Takt- und Rücksetzungsstifte dieses Designs zu ermitteln. Darüber hinaus erstellt es eine Ausgabedatei, die diese Informationen enthält und als Grundlage für eine Test Bench-Datei verwendet werden kann.

FPGA Module Klass

Die Erklärung der verwendeten Bibliotheken in diesem Code-Abschnitt lautet wie folgt:

1. ``import java.io.FileWriter;``: Diese Bibliothek wird verwendet, um Schreibvorgänge in Dateien durchzuführen. Innerhalb des Codes wird die Klasse ``FileWriter`` verwendet, um Daten in Dateien zu schreiben. Insbesondere wird sie in der Methode ``writeFile`` verwendet, um Daten in eine Datei zu schreiben.

2. ``import java.io.IOException;``: Diese Bibliothek wird verwendet, um mögliche Ausnahmen (Fehler) während Dateioperationen zu behandeln. Sie wird insbesondere verwendet, um Fehler wie das Fehlen von Schreibberechtigungen für eine Datei während des Schreibvorgangs zu behandeln.

3. ``import java.nio.file.Paths;``: Diese Bibliothek wird verwendet, um Dateipfade zu verarbeiten. Innerhalb des Codes wird die Klasse ``Paths`` verwendet, um einen angegebenen Dateipfad zu repräsentieren und zu verarbeiten. Dieser Pfad wird verwendet, um den Speicherort von Dateien anzugeben, beispielsweise durch ``Paths.get(outputFilePath)``.

4. ``import java.util.ArrayList;``: Diese Bibliothek wird verwendet, um die dynamische Listenstruktur namens `ArrayList` zu verwenden. Innerhalb des Codes werden Datenstrukturen wie ``ArrayList<String>`` und ``ArrayList<ModuleParameter>`` verwendet, um Daten zu speichern und zu verarbeiten.

5. ``import java.util.Scanner;``: Diese Bibliothek wird verwendet, um Daten aus Dateien zu lesen. Innerhalb des Codes wird die Klasse ``Scanner`` verwendet, um Daten zeilenweise aus Dateien zu lesen. Insbesondere wird sie in der Methode ``readFile`` verwendet, um Daten aus einer Datei zu lesen.

In diesem Codebeispiel werden einige Elemente der objektorientierten Programmierung (OOP) wie Vererbung und die Verwendung von Klassen und Objekten verwendet, um Datenstrukturen und Funktionen zu organisieren. Dies ermöglicht eine strukturierte und effiziente Verarbeitung von Informationen und erleichtert die Wartbarkeit des Codes.

1. Klassenbeschreibung: Die ``FPGAModule``-Klasse ist eine Klassendefinition. Die Klasse bildet die Hauptstruktur, in der Ihr Code organisiert ist.

2. Eigenschaften (Felder):

- ``String name``: Eine Eigenschaft, die den Namen der Klasse repräsentiert.
- ``ArrayList<String> fileContent``: Ein dynamisches Array (`ArrayList`), das verwendet wird, um den Dateiinhalt zu speichern.

FPGA-KONSOLENANWENDUNG

FPGA - Automatische Generierung von Test Bench Vorlagen

- ``ArrayList<ModuleParameter> moduleParameters``: Ein dynamisches Array, das Objekte der Klasse ``ModuleParameter`` enthält.

- ``Boolean clockReset``: Eine boolesche Variable, die die Uhr- und Rücksetzungseigenschaft repräsentiert.

3. Konstruktor-Methode (Constructor):

- ``FPGAModule(String outputPath)``: Diese Konstruktor-Methode führt die ersten Schritte aus, die beim Erstellen eines Objekts der Klasse erforderlich sind. Sie liest den Dateiinhalt, erhält den Modulnamen und holt sich die Modulparameter.

4. Methoden:

Ich möchte die Ein- und Ausgabeparameter des Moduls testen. Für jede Datei gibt es unterschiedliche Parameter. Als Beispiel gebe ich einen Parameter hier an:

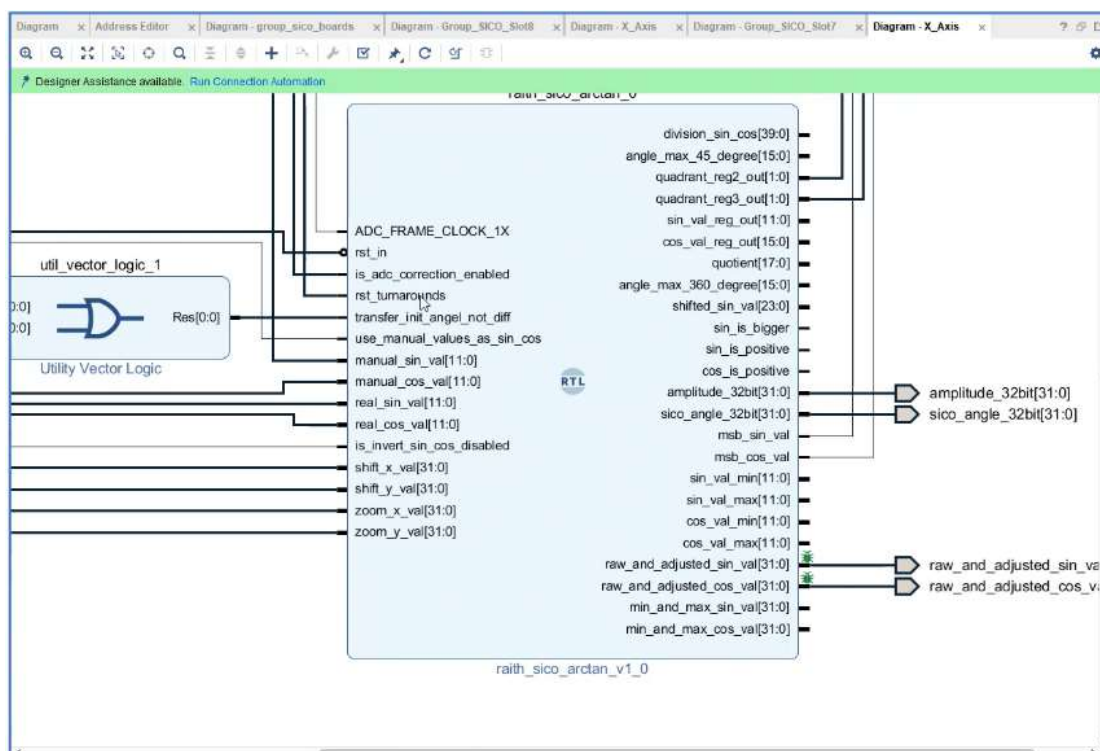


Abbildung 8 Parametern von eine FPGA Modul

Diese Parameter werden aus einer Verilog-Datei extrahiert. Als Beispiel kann ich Ihnen diese Datei geben. Darin sind Eingänge und Ausgänge enthalten. Um diese Eingänge in Registern

FPGA-KONSOLENANWENDUNG

FPGA - Automatische Generierung von Test Bench Vorlagen

und Ausgänge in Drähte umzuwandeln, schreibe ich eine Funktion namens `getModuleParameters()`:



Abbildung 9 Beispiel von eine Dateiname

- `public ArrayList<ModuleParameter> getModuleParameters()`: Diese Methode analysiert den Dateinhalt, um die Modulparameter zu extrahieren, und gibt sie als `ArrayList` zurück. Der Grund, warum ich eine ArrayList verwende, ist, dass ich am Anfang nicht weiß, wie viele Elemente es geben wird. Die ArrayList fügt automatisch Elemente hinzu, unabhängig davon, wie viele es sind.

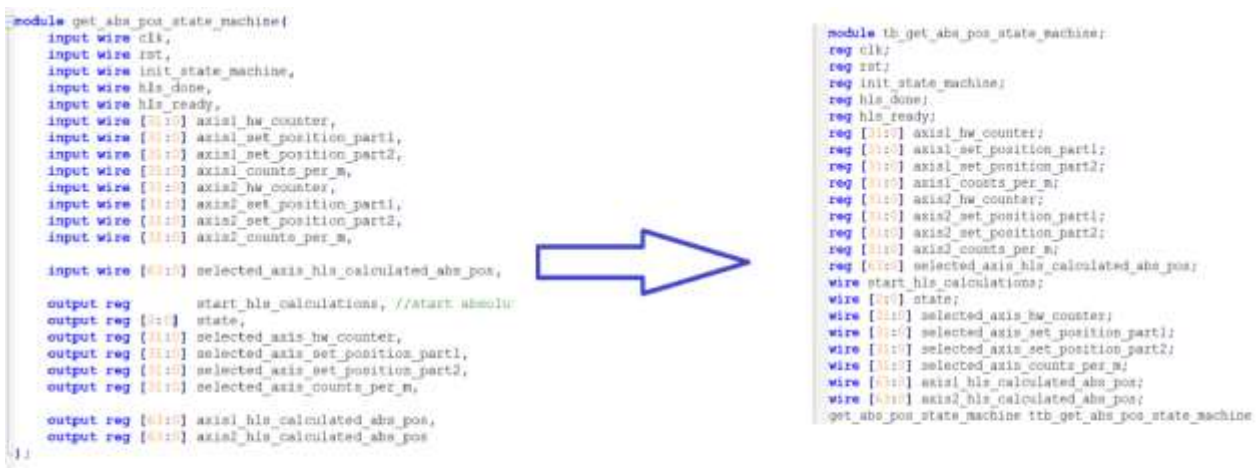


Abbildung 10 Konvertierung von Ein- und Ausgängen

- `public ArrayList<String> readFile(String outputFilePath)`: Diese Methode liest den Inhalt der angegebenen Datei und speichert jede Zeile in einem `ArrayList`.

Nachdem die Eingangs- und Ausgangsparameter organisiert wurden, müssen sie instanziiert werden. Um den Namen für die Instanzierung anzugeben, muss ich zuerst den Namen des Moduls finden. Dafür suche ich nach dem Schlüsselwort "module". Das Wort, das nach "module" kommt, wird der Name meines Moduls sein, und in der Test Bench muss ich dieses Modul instanziiert. Ich nenne den Modulnamen und lasse einen Leerraum. Dann füge ich "ttb_" am Anfang hinzu und füge den Modulnamen hier ein. Der Name des Moduls wird also "get_abs_pos_state_maschine ttb_ get_abs_pos_state_maschine" sein. Diesen Benennungsvorgang führe ich mit der Methode `public String getModuleName()` durch.

Für den Instanziierungsteil muss ich die Ein- und Ausgänge finden. Wenn ich jede Zeile durchgehe und Input lese, wenn ich Eingänge finde, und Output lese, wenn ich Ausgänge finde, wird jede Zeile das gleiche Format haben: Es wird mit einem Punkt "." beginnen, gefolgt vom Namen des Parameters, dann in Klammern der Name noch einmal, und am Ende wird ein Komma stehen. Zusätzlich wird jede Zeile in einer neuen Zeile sein. Ob es sich um Eingabe oder Ausgabe handelt, ist unwichtig; es wird in diesem Format sein, ohne Unterschied. Zum Beispiel wird es so aussehen: ".init_state_machine(init_state_machine),".

Zuvor wurden all diese Aufgaben manuell in Excel durchgeführt. Um diese Aufgabe zu automatisieren, habe ich die Methode `@Override String toString()` erstellt. Diese Methode erzeugt eine Repräsentation der Klasse und gibt sie als Zeichenkette zurück. Im Code-Block habe ich Text zu einem StringBuilder-Objekt hinzugefügt, da StringBuilder eine effizientere Möglichkeit zur Erstellung von Text bietet. Es kann Text in mehreren Schritten zusammenführen und schließlich in einen String umwandeln. Dies war besonders hilfreich, um große Textabschnitte zusammenzuführen. Nach dieser Operation erhielt ich folgende Ausgabe:

```
(
    .clk(clk),
    .rst(rst),
    .init_state_machine(init_state_machine),
    .hls_done(hls_done),
    .hls_ready(hls_ready),
    .axis1_hw_counter(axis1_hw_counter),
    .axis1_set_position_part1(axis1_set_position_part1),
    .axis1_set_position_part2(axis1_set_position_part2),
    .axis1_counts_per_m(axis1_counts_per_m),
    .axis2_hw_counter(axis2_hw_counter),
    .axis2_set_position_part1(axis2_set_position_part1),
    .axis2_set_position_part2(axis2_set_position_part2),
    .axis2_counts_per_m(axis2_counts_per_m),
    .selected_axis_hls_calculated_abs_pos(selected_axis_hls_calculated_abs_pos),
    .start_hls_calculations(start_hls_calculations),
    .state(state),
    .selected_axis_hw_counter(selected_axis_hw_counter),
    .selected_axis_set_position_part1(selected_axis_set_position_part1),
    .selected_axis_set_position_part2(selected_axis_set_position_part2),
    .selected_axis_counts_per_m(selected_axis_counts_per_m),
    .axis1_hls_calculated_abs_pos(axis1_hls_calculated_abs_pos),
    .axis2_hls_calculated_abs_pos(axis2_hls_calculated_abs_pos)
):
```

Abbildung 11 Instanziierungsteil

In meinem Projekt habe ich einen weiteren Schritt implementiert, bei dem der Text gelesen wird, um zu überprüfen, ob die Begriffe "clock" und "reset" vorhanden sind, und diese Informationen in einer ArrayList gespeichert werden. Wenn die gelesene Zeile eines der Wörter "clk", "_clk" oder "clock" enthält, wird "clock" auf "true" gesetzt. Wenn die Zeile "rst" oder "reset" enthält, wird "reset" auf "true" gesetzt. Andernfalls bleiben beide auf "false". Hier sind die verwendeten Methoden:

- `ArrayList<Boolean> getClockReset(String outputPath)`: Diese Methode bestimmt die Takt- und Rücksetzeigenschaften. In dieser Methode habe ich `toLowerCase()` verwendet, um sicherzustellen, dass alle Buchstaben in der gelesenen Zeile klein sind. Der Grund dafür ist, die Groß- und Kleinschreibung zu vernachlässigen und die Erkennung von "clock" oder "reset" unabhängig von Groß- und Kleinschreibung zu ermöglichen. Außerdem habe ich `try` und `catch` verwendet, um die Methode gegen Ausnahmen abzusichern, die während des Dateilesens auftreten können.

- `String getClockName()`: Diese Methode gibt den Namen des Taktparameters zurück und findet einen Parameter, der den Takt repräsentiert.

- `String getResetName()`: Diese Methode gibt den Namen des Rücksetzparameters zurück und findet einen Parameter, der das Rücksetzsignal repräsentiert.

- `String assignValueToParameter(String parameterName, int value)`: Diese Methode wird verwendet, um einem bestimmten Parameter einen Wert zuzuweisen und erstellt eine Zuweisungsaussage.

Die Methode akzeptiert zwei Argumente: `parameterName` (eine Zeichenfolge, die den Namen eines Parameters repräsentiert) und `value` (eine Ganzzahl `int`, die den Wert darstellt, der diesem Parameter zugewiesen wird).

Die Methode erstellt eine Textzeichenfolge, die aufgrund dieser beiden Argumente eine Zuweisungsaussage enthält, die den Wert für den Parameter darstellt.

Dies ermöglicht es uns, Parameterwerte dynamisch zur Laufzeit des Programms zu erstellen, was für die Automatisierung in FPGA-Designs sehr nützlich ist. Ich kann beispielsweise null für alle Inputs schreiben, da ich nicht weiß, welcher Wert in ihnen sein wird. Dies erzeugt eine Ausgabe wie folgt:

```
initial
begin
    clk = 0;
    rst = 1;
    init_state_machine = 0;
    hls_done = 0;
    hls_ready = 0;
    axis1_hw_counter = 0;
    axis1_set_position_part1 = 0;
    axis1_set_position_part2 = 0;
    axis1_counts_per_m = 0;
    axis2_hw_counter = 0;
    axis2_set_position_part1 = 0;
    axis2_set_position_part2 = 0;
    axis2_counts_per_m = 0;
    selected_axis_hls_calculated_abs_pos = 0;
    start_hls_calculations = 0;
    state = 0;
    selected_axis_hw_counter = 0;
    selected_axis_set_position_part1 = 0;
    selected_axis_set_position_part2 = 0;
    selected_axis_counts_per_m = 0;
    axis1_hls_calculated_abs_pos = 0;
    axis2_hls_calculated_abs_pos = 0;
```

Abbildung 12 Assign value to Parameter

`String addClockCycles(int cycleCount)`: Diese Methode wird verwendet, um Taktzyklen zu erstellen und sicherzustellen, dass das Taktsignal für eine bestimmte Anzahl von Zyklen wiederholt wird. Dies modelliert das Verhalten des Signals und simuliert die Funktionsweise des Designs. `result.append("repeat(10)@(posedge)")`: Dies ist ein Ausdruck in der Verilog-Sprache, der den Beginn einer Taktzyklusdefinition kennzeichnet. Die Aussage `repeat(10)` stellt eine Schleife dar, die 10-mal um das Taktsignal herum ausgeführt wird. `@(posedge)` gibt an, dass ich auf die steigende Flanke des Taktsignals warte. Das Ergebnis sieht folgendermaßen aus:

```
repeat (10) @(posedge clk);
rst = 0;
```

Abbildung 13 Ausgabe von Taktzyklen

Module Parameter Klass

Die Klasse `ModuleParameter` definiert eine spezielle Java-Klasse, die die Modulparameter einer FPGA-Designs repräsentiert. Diese Klasse ist nützlich, um Modulparameter in FPGA-Designs oder verwandten Vorgängen darzustellen. Jeder Parameter hat Eigenschaften wie Name, Länge, Signierung, Richtung (Eingang/Ausgang), Takt und Rücksetzsignal. Die Eigenschaften der Parameter können über Methoden festgelegt werden, und Textzeichenfolgen, die die Parameter darstellen, können erstellt werden.

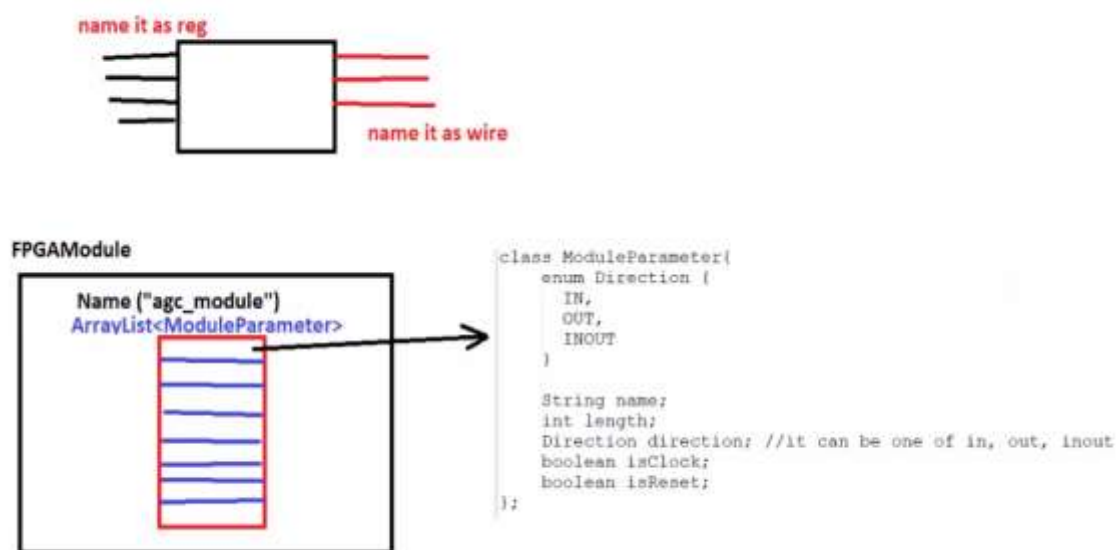


Abbildung 14 Enum Benutzung Beispiel und FPGA Module

ENUMs; werden verwendet, um in Situationen, in denen die Wahrscheinlichkeit zu 100% auf eine von mehreren Möglichkeiten beschränkt ist, eine Auswahl zu treffen. In meinem Fall gibt es nur zwei Möglichkeiten für die Richtung: Eingang (IN) oder Ausgang (OUT), es gibt keine anderen Optionen. Wenn ich "IN" sehe, wähle ich "IN", wenn ich "OUT" sehe, wähle ich "OUT". Wenn ich "INOUT" sehe, wähle ich ebenfalls entsprechend.

Insgesamt gibt es 18 Elemente in der Liste, und ich muss alle von ihnen festlegen. Ich muss Informationen wie den Namen des Ein-/Ausgangsparameters, die Länge (falls vorhanden), die Anzahl der öffnenden und schließenden Klammern (falls vorhanden), die Richtung (INPUT oder OUTPUT), das Vorhandensein von Begriffen wie "clock" oder "clk" (true oder false) und das Vorhandensein von "RESET" oder "RST" (true oder false) festlegen.

Die `Direction`-ENUM wird verwendet, um die Richtung des Parameters festzulegen. Hierbei handelt es sich um Eingang (IN), Ausgang (OUT) oder Eingang/Ausgang (INOUT).

```
enum Direction {  
    IN,  
    OUT,  
    INOUT  
};
```

```
module get_abs_pos_state_machine(  
    input wire clk,  
    input wire rst,  
    input wire init_state_machine,  
    input wire hls_done,  
    input wire hls_ready,  
    input wire [31:0] axis1_hw_counter,  
    input wire [31:0] axis1_set_position_part1,  
    input wire [31:0] axis1_set_position_part2,  
    input wire [31:0] axis1_counts_per_m,  
    input wire [31:0] axis2_hw_counter,  
    input wire [31:0] axis2_set_position_part1,  
    input wire [31:0] axis2_set_position_part2,  
    input wire [31:0] axis2_counts_per_m,  
  
    input wire [63:0] selected_axis_hls_calculated_abs_pos,
```

Abbildung 15 Bestimmung die Länge

Wenn in einer Zeile eckige Klammern `[]` vorhanden sind, bedeutet dies, dass es sich um einen Bus handelt und ich seine Länge identifiziere. Zum Beispiel, `[11:0]`, wenn diese Werte in den eckigen Klammern vorhanden sind, bedeutet das, dass es sich um einen Bus handelt, der insgesamt 12 Werte von 0 bis 11 enthält, was einen 12-Bit-Wert darstellt.

Wenn `[31:0]` vorhanden ist, bedeutet dies, dass es sich um einen 32-Bit-Wert handelt. Indem Sie diese Informationen erkennen und wissen, dass es sich um einen Bus handelt, können Sie die Eingänge mit dem Präfix "reg" in Register umwandeln und die Ausgänge mit dem Präfix "wire" kennzeichnen.

Während der Entwurfsphase des Projekts gab es zwei wichtige Prozesse, die während der Implementierung umgesetzt wurden.

1. Entfernen von Kommentaren: Wenn eine Zeile einen `//` Kommentar enthält, muss dieser Kommentar entfernt werden. Während des Implementierungsprozesses wurde dies mithilfe der `substring`-Methode erreicht. Das bedeutet, dass während der Ausführung des Programms, wenn eine Zeile einen solchen Kommentar enthält, dieser Kommentar entfernt wird.

2. Erkennen von Vorzeichen (Signed): Ein weiterer Schritt, der während der Implementierung hinzugefügt wurde, bestand darin, Vorzeichen (Signed) in den Zeilen zu erkennen. Wenn eine Zeile das Wort "Signed" enthält, wird dies in einer Bedingung erkannt und entsprechend verarbeitet. Dies könnte bedeuten, dass bestimmte Operationen oder Eigenschaften basierend auf dem Vorhandensein von "Signed" in der Zeile durchgeführt werden.

Variablenbeschreibung:

- `String name`: Dies ist eine Zeichenfolge, die den Namen des Modulparameters speichert.
- `int length`: Dies ist eine Ganzzahl, die die Länge des Modulparameters speichert.
- `String lengthStr`: Dies ist eine Zeichenfolge, die die Länge des Modulparameters speichert.
- `boolean signed`: Dies ist ein boolescher Wert, der angibt, ob der Modulparameter signiert (signed) ist oder nicht.
- `Direction direction`: Dies ist eine Variable vom Typ Direction, die die Richtung des Modulparameters festlegt.
- `boolean isClock`: Dies ist ein boolescher Wert, der angibt, ob der Modulparameter ein Takt (clock) ist oder nicht.

- ``boolean isReset``: Dies ist ein boolescher Wert, der angibt, ob der Modulparameter ein Reset (reset) ist oder nicht.

Der Konstruktor ``public ModuleParameter(String lineContent)`` wird verwendet, um einen Modulparameter zu erstellen. Dieser Konstruktor akzeptiert eine Zeichenfolge (`lineContent`) und extrahiert die Eigenschaften des Modulparameters aus dieser Zeichenfolge.

Die ``toString`` Methode ist mit ``@Override`` markiert und wird verwendet, um die Eigenschaften eines `ModuleParameter`-Objekts in Textformat zu formatieren und diesen Text zurückzugeben. Der zurückgegebene Text enthält die Eigenschaften des Modulparameters wie Name, Richtung, Länge, Vorzeichen, Takt und Reset.

6 Testphase und Qualitätskontrolle

Der Zweck des Projektes ist es, automatisch eine FPGA Test Bench zu erstellen. Aus diesem Grund ist das Projekt selbst bereits ein Testprogramm. Darüber hinaus wird kein spezielles Testprogramm verwendet. Die Eingaben und Ausgaben von Verilog-Dateien werden mit der Anwendung „Beyond Compare“ verglichen. Beyond compare wird für den Vergleich langer und komplexer Texte verwendet. Es schafft eine schnelle und effiziente Testumgebung, indem es verschiedene Wörter oder Sätze unterschiedlich einfärbt. Die Tatsache, dass die von mir geschriebene Anwendung läuft, wenn ich einen Befehl auf der Konsole eingebe, und die gewünschte Ausgabe liefert, zeigt, dass die Anwendung korrekt funktioniert. Ich habe auch überprüft, ob mein Projekt funktioniert, indem ich verschiedene Verilog-Dateien in meiner Anwendung verwendet habe. Screenshots dazu finden Sie im Anhang A6.

7 Abnahme

Nach der Umsetzungsphase wurde die Phase der Abnahme der Projektergebnisse erreicht. Alle Tests wurden mit einem positiven Ergebnis abgeschlossen. Zur Qualitätssicherung der Anwendung traf ich mich mit meinem Projektberater Herrn Kocal. Ich stellte die von mir durchgeführten Tests und die Arbeit an der Anwendung vor, die von Herrn Kocal genehmigt wurde. Damit war die Anwendung einsatzbereit und konnte für die FPGA-Tests verwendet werden.

8 Dokumentation

Die Anwendung ist für unterschiedliche Zielgruppen wie Benutzer und Administratoren dokumentiert. Diese Dokumentation wurde manuell erstellt, um sicherzustellen, dass die spezifischen Bedürfnisse und Anforderungen jeder Gruppe berücksichtigt werden konnten.

Für Benutzer wurde ein Benutzerhandbuch erstellt, das in klarer und verständlicher Sprache verfasst ist. Dieses Handbuch enthält detaillierte Anweisungen zur Verwendung der Anwendung, ohne komplizierte IT-Begriffe zu verwenden.

Für Administratoren wurden eigene Dokumente erstellt, die speziell auf deren Bedürfnisse zugeschnitten sind. Dieses Dokument enthält technische Informationen zur Entwicklung und Erstellung der Anwendung.

Um den Benutzern eine einfache Bedienung zu ermöglichen, sind die entsprechenden Bilder im Anhang A7 beigelegt.

Um eine effektive Nutzung der Dokumentation zu gewährleisten, wurde bei der Erstellung der Dokumentation darauf geachtet, die Anforderungen und Bedürfnisse der jeweiligen Zielgruppe zu berücksichtigen.

9 Fazit

Bei der Prüfung des IHK-Abschlussprojekts kann festgestellt werden, dass alle zuvor ermittelten Anforderungen vollständig erfüllt sind. Durch die Einhaltung des zu Beginn des Projekts erstellten Projektplans konnten die Erwartungen gemäß den Vorgaben erfüllt werden. Daher war der Geschäftsführer mit dem Ergebnis zufrieden und es wurde ein Programm erstellt, das in der Zukunft eingesetzt werden kann.

9.1 Soll-/Ist-Vergleich

Die folgende Tabelle 7 : Plan-Ist-Vergleich zeigt die für jede Phase tatsächlich aufgewendete Zeit im Vergleich zur geplanten Zeit. Diese Vergleich zeigt nur geringfügige Unterschiede. Diese Unterschiede wurden so berechnet, dass sie sich gegenseitig ausgleichen, wodurch das Projekt innerhalb des von IHK festgelegten Zeitrahmens von 80 Stunden realisierbar wurde. Mit anderen Worten, kurze Arbeitszeiten waren entscheidend, um bedeutende Ergebnisse zu erzielen.

Phase	Soll	Ist	Differenz
1 Analyse	4h	4h	0h
2 Konzeption	9h	9h	0h
3 Planung	6h	6h	0h
4 Realisierung	26h	26h	0h
5 Validierung	11h	10h	-1h
6 Abschluss	7h	7h	0h
7 Dokumentation	17h	18h	+1h
Gesamtsumme	80h	80h	0h

Tabelle 7 Soll-Ist Vergleich

Für weitere Details können Sie in Anhang A8 eine detaillierte Soll-/Ist-Vergleichung nachlesen.

9.2 Wirtschaftlichkeitsbetrachtung

Für die wirtschaftliche Bewertung des Projekts habe ich die Abschreibungsdauer berechnet. Dieser Zeitraum zeigt, wie lange es dauern wird, bis die Investitionskosten durch die erwarteten Einsparungen gedeckt sind. Als Kosten für die Einführung des neuen Systems wurden 1.466 Euro ermittelt. Durch die Automatisierung des Bewertungsprozesses, der bisher manuell durchgeführt wurde, können Zeit- und Kosteneinsparungen erzielt werden. Basierend auf diesen Einsparungen habe ich berechnet, dass sich die Investitionskosten in nur 3 Monaten amortisieren würden. Diese Berechnung zeigt, dass das Projekt wirtschaftlich rentabel ist, da in kurzer Zeit Investitionskosten eingespart werden.

9.3 Erfahrungen

Ich habe viel Erfahrung bei der Umsetzung des Projekts gesammelt. Der Entwurf einer objekt-orientierten Programmierung mit geeigneten Modulen für die FPGA-Testvorlage war zunächst

recht schwierig. Aber später, als ich daran arbeitete, hat es großen Spaß gemacht, die Codes anzuwenden und zu versuchen, eine Lösung zu finden.

Dabei habe ich vor allem in Technik und technischen Belangen neue Erfahrungen gesammelt. Die Projektplanungs- und Entwurfsphase ist entscheidend für den Erfolg des Projekts. Trotz der anfänglichen Schwierigkeiten habe ich durch dieses Projekt wertvolle Erfahrungen gesammelt und eine funktionierende Anwendung entwickelt, die bald in den täglichen Betrieb des Unternehmens integriert werden soll.

Literaturverzeichnis

FPGA: https://de.wikipedia.org/wiki/Field_Programmable_Gate_Array

Verilog: <https://www.chipverify.com/verilog/verilog-tutorial>

IntelliJ IDEA : <https://www.oracle.com/de/java/>

Java : <https://www.oracle.com/de/java/>

Persönliche Erklärung

Erklärung der Prüfungsteilnehmer/in

Ich versichere durch meine Unterschrift, dass ich das betriebliche Projekt und die dazugehörige Dokumentation selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat in dieser Form keiner anderen Prüfungsinstitution vorgelegen.

Das Projekt wurde ordnungsgemäß vom 29. August 2023 bis zum 11. September 2023 durchgeführt.

Bochum, 17.10.2023



Ort, Datum

Unterschrift des Prüfungsteilnehmers

Erklärung des Ausbildungsbetriebes / Praktikumsbetriebes

Wir versichern, dass das betriebliche Projekt wie in der Dokumentation dargestellt, in unserem Unternehmen realisiert worden ist.

Bochum, 17.10.2023



Ort, Datum

Unterschrift und Firmenstempel

Anhang

A1 Detaillierte Zeitplanung

Zeitplanung detailliert	Zwischensumme	Gesamtstunden
1.Analysephase		4h
Ist-Analyse	2h	
Use-Case Diagramm	1h	
Erstellen des Lastenheftes	1h	
2.Konzeption		9h
Soll-Konzept	1h	
Aufbau der Programmstruktur	3h	
Konzeption der Ausgabeformat	1h	
Geschäftslogik planen	2h	
Pflichtenheft	2h	
3.Planung		6h
Wirtschaftlichkeit prüfen	2h	
Zeitplanung	1h	
Kostenplanung	1h	
Planung der Tests	2h	
4.Realisierung		26h
Lesen der Eingabedatei	4h	
Bestimmen der Eingänge und Ausgänge	4h	
Instantieren des Moduls zum Schreiben der Testbench	3h	
Testbench als Ausgabe in eine Datei schreiben	3h	
Integration der Eingabedaten und erwarteten Ausgabewerte	6h	
Implementieren der Simulationsschleife	6h	
5.Validierung		11h
Durchführen von umfangreichen Tests	7h	
Fehlerbeheben und Verbessern der Anwendung	4h	
6.Abschluss		7h
Soll-/Ist-Vergleich	2h	
Wirtschaftliche Nachbetrachtung	2h	
Abgabe/ Präsentation	3h	
7.Dokumentation		17h
Dokumentation verfassen	15h	
Benutzerdokumentation erstellen	2h	

Tabelle 8 detaillierte Zeitplanung

A2 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die Anforderungen:

Die Anwendung muss folgende Anforderungen erfüllen.

1. Eingabedaten lesen und ausgeben

1.1. Die Anwendung muss in der Lage sein, Eingabedaten aus einer Verilog-Datei zu lesen und sie auf der Konsole auszugeben.

1.2. Muss der Dateipfad über die Befehlszeile eingegeben werden..

2. Identifizieren von Modulnamen, Eingängen und Ausgängen

2.1. Die Anwendung sollte in der Lage sein, den Modulnamen aus der Verilog-Datei gefolgt vom Schlüsselwort „module“ zu identifizieren.

2.2. Wenn das Wort „Module“ gefunden wird, sollte das darauf folgende Wort als Modulname verwendet werden.

2.3. Die Anwendung muss in der Lage sein, Ein- und Ausgänge in der Verilog-Datei zu definieren. Dies geschieht, indem die Eingabe/Ausgabe in der Datei Zeile für Zeile gelesen wird, bis das erste schließende Klammerzeichen „)“ das Ende der Datei darstellt.

2.4. „Input“ im Modul muss als „reg“ geschrieben werden.

2.5. „Output“ im Modul muss als „wire“ geschrieben werden.

2.6. Die Wörter IN, out, inout müssen mit enum abgeschlossen werden.

2.7. Enthält eine Zeile zusätzlich „[]“, muss die Anwendung erkennen, dass es sich um einen Datenpfad handelt und dessen Länge ermitteln. Beispiel: „Eingangsdraht [11:0] sin_val_min“ bedeutet, dass es sich um einen Bus der Länge 12 handelt.

2.8. Die Anwendung sollte auch in der Lage sein, Clock und Reset-Pins zu identifizieren, indem sie nach bestimmten Schlüsselwörtern wie „Clock,clk “ und „Reset,rst“ sucht.

3. Generierung des Instanzierungsblocks für ein verwandtes Modul

3.1. Die Anwendung muss in der Lage sein, einen Instanzierungsblock für ein verwandtes Modul zu generieren.

4. Generierung der Ausgabedatei als Grundlage für die Test Bench

4.1. Die Anwendung muss eine Ausgabedatei erstellen, die als Grundlage für die Test Bench verwendet werden kann.

A3 Use Case Diagramm

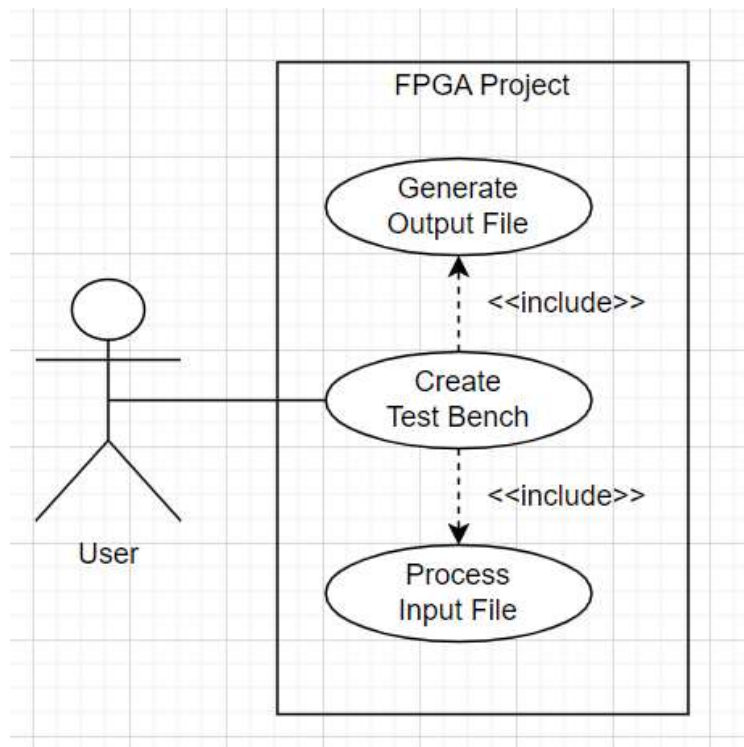


Abbildung 16 UseCase Diagramm

A4 Pflichtenheft (Auszug)

1. Einleitung

Das Pflichtenheft definiert die funktionalen Anforderungen und Spezifikationen für das FPGA-Projekt zur automatischen Generierung von Test Bench-Vorlagen. Das Projekt zielt darauf ab, eine Anwendung zu entwickeln, die Verilog-Dateien analysiert, Modulnamen, Eingänge, Ausgänge, Datenpfade und Takt-/Reset-Pins identifiziert und eine Test Bench-Vorlage erstellt.

2. Funktionale Anforderungen

2.1 Eingabedaten lesen und ausgeben

Die Anwendung muss die Fähigkeit haben, Eingabedaten aus einer Verilog-Datei zu lesen und diese auf der Konsole auszugeben.

2.2 Einlesen des Dateipfads

Die Anwendung muss den Dateipfad der Verilog-Datei über die Befehlszeile akzeptieren. Der Benutzer muss in der Lage sein, den Dateipfad beim Starten der Anwendung anzugeben.

2.3 Identifizieren von Modulnamen, Eingängen und Ausgängen

Die Anwendung sollte in der Lage sein, den Modulnamen aus der Verilog-Datei gefolgt vom Schlüsselwort "module" zu identifizieren.

2.4 Modulnamen identifizieren

Wenn das Schlüsselwort "module" gefunden wird, sollte die Anwendung das darauf folgende Wort als Modulnamen verwenden.

2.5 Identifizieren von Ein- und Ausgängen

Die Anwendung muss in der Lage sein, Ein- und Ausgänge in der Verilog-Datei zu definieren. Dies erfolgt durch das Zeilenweise Lesen der Eingabe-/Ausgabedefinitionen in der Datei bis zum ersten schließenden Klammerzeichen ")".

2.6 Eingänge als "reg" definieren

Eingänge im Modul müssen als "reg" in der Test Bench-Vorlage geschrieben werden.

2.7 Ausgänge als "wire" definieren

Ausgänge im Modul müssen als "wire" in der Test Bench-Vorlage geschrieben werden.

2.8 Verwendung von Enums für IN, OUT, INOUT

Die Anwendung sollte Enums verwenden, um die Richtungen der Ein- und Ausgänge (IN, OUT, INOUT) zu definieren.

2.9 Identifizieren von Datenpfaden

Enthält eine Zeile "[]", muss die Anwendung erkennen, dass es sich um einen Datenpfad handelt, und dessen Länge ermitteln. Zum Beispiel: "Eingangsdraht [11:0] sin_val_min" bedeutet, dass es sich um einen Bus der Länge 12 handelt.

2.10 Identifizieren von Clock und Reset-Pins

Die Anwendung sollte in der Lage sein, Clock- und Reset-Pins zu identifizieren, indem sie nach bestimmten Schlüsselwörtern wie "Clock", "clk" und "Reset", "rst" sucht.

2.11 Entfernen von Kommentaren

Während des Implementierungsprozesses wurde die Anwendung entwickelt, um Kommentare in den Zeilen zu erkennen und zu entfernen, wenn sie das Muster "//" aufweisen.

2.12 Erkennen von Vorzeichen (Signed)

Ein weiterer Prozess, der während der Implementierung hinzugefügt wurde, besteht darin, Vorzeichen (Signed) in den Zeilen zu erkennen und entsprechend zu verarbeiten. Wenn eine Zeile das Wort "Signed" enthält, wird dies in einer Bedingung erkannt und entsprechend verarbeitet.

3. Generierung des Instanzierungsblocks für ein verwandtes Modul

Die Anwendung muss in der Lage sein, einen Instanzierungsblock für ein verwandtes Modul zu generieren.

4. Generierung der Ausgabedatei als Grundlage für die Test Bench

Die Anwendung muss eine Ausgabedatei erstellen, die als Grundlage für die Test Bench verwendet werden kann. Die Ausgabedatei sollte dem spezifizierten Format entsprechen.

5. Nicht-funktionale Anforderungen

- Die Anwendung muss in Java entwickelt werden.
- Sie sollte auf verschiedenen Verilog-Dateien getestet werden, um sicherzustellen, dass sie korrekte Test Bench-Vorlagen generiert.

6. Abnahme- und Testverfahren

Die Abnahme der Anwendung erfolgt, indem sie auf verschiedenen Verilog-Dateien getestet wird, um sicherzustellen, dass sie alle spezifizierten Anforderungen erfüllt.

A5 Code Ausschnitte

```

mainProgram.java x FPGAModule.java ModuleParameter.java ModuleParameter.class
1  import java.io.FileNotFoundException;
2  import java.io.PrintWriter;
3  import java.nio.file.Paths;
4  import java.util.ArrayList;
5  import java.util.Scanner;
6
7  public class mainProgram {
8
9      public static void main(String[] args) throws FileNotFoundException { //
10
11          if (args.length != 1) {
12              System.out.println("Usage: java mainProgram <outputFilePath>");
13              return;
14          }
15
16          String outputFilePath = args[0];
17
18          FPGAModule fpgaModule = new FPGAModule(outputFilePath);
19          System.out.println(fpgaModule);
20          try(PrintWriter out = new PrintWriter("./demo_test.v")){
21              out.println(fpgaModule);
22          }
23      }
24  }
25
  
```

Abbildung 17 mainProgram

```

mainProgram.java FPGAModule.java x ModuleParameter.java ModuleParameter.class
1
2
3  import java.io.FileWriter;
4  import java.io.IOException;
5
6
7  import java.nio.file.Paths;
8  import java.util.ArrayList;
9  import java.util.Scanner;
10
11
12  public class FPGAModule {
13
14      String name;
15      ArrayList<String> fileContent;
16      ArrayList<ModuleParameter> moduleParameters;
17      Boolean clockReset;
18
19      public FPGAModule(String outputFilePath) {
20          fileContent = readFile(outputFilePath);
21          name = getModuleName();
22          //getParameters needs to come here...
23          moduleParameters = getModuleParameters();
24      }
25
26      public ArrayList<ModuleParameter> getModuleParameters() {
27          moduleParameters = new ArrayList<>();
28          boolean insideDeclarations = false;
29
30          for (String lineContent : fileContent) {
31              if (lineContent.contains("(")) {
32                  insideDeclarations = false;
33              }
34
35              if (lineContent.startsWith("input")
36                  || lineContent.startsWith("output")
37                  || lineContent.startsWith("inout")) {
38                  ModuleParameter moduleParameter = new ModuleParameter(lineContent);
39                  moduleParameters.add(moduleParameter);
40              }
41          }
42      }
43  }
  
```

Abbildung 18 FPGAModule


```
mainProgram.java  FPGAModule.java  ModuleParameter.java  x  ModuleParameter.class

14  public class ModuleParameter {
15
16      enum Direction {
17          IN,
18          OUT,
19          INOUT
20      };
21
22      String name;
23
24      int length;
25
26      String lengthStr = "";
27
28      boolean signed = false;
29
30      Direction direction; //it can be one of in, out, inout
31
32      boolean isClock;
33
34      boolean isReset;
35
36
37
38      public ModuleParameter(String lineContent) {
39          //remove comment part of the line
40          if (lineContent.contains("//")) {
41              int beginIndex = lineContent.indexOf("//");
42              lineContent = lineContent.substring(0, beginIndex);
43          }
44
45          //if we split line content, then last part is name of parameter
46          String[] splittedLineContent = lineContent.split(" ");
47          name = splittedLineContent[splittedLineContent.length - 1].replace(",", "");
48
49          if (lineContent.contains("input")) {
```

Abbildung 19 moduleParameter

A6 Testprotokolle (Auszug)

```

module get_abs_pos_state_machine(
    input wire clk,
    input wire rst,
    input wire init_state_machine,
    input wire hls_done,
    input wire hls_ready,
    input wire [31:0] axis1_hw_counter,
    input wire [31:0] axis1_set_position_part1,
    input wire [31:0] axis1_set_position_part2,
    input wire [31:0] axis1_counts_per_m,
    input wire [31:0] axis2_hw_counter,
    input wire [31:0] axis2_set_position_part1,
    input wire [31:0] axis2_set_position_part2,
    input wire [31:0] axis2_counts_per_m,

    input wire [63:0] selected_axis_hls_calculated_abs_pos,

    output reg [2:0] state,
    output reg [31:0] selected_axis_hw_counter,
    output reg [31:0] selected_axis_set_position_part1,
    output reg [31:0] selected_axis_set_position_part2,
    output reg [31:0] selected_axis_counts_per_m,

    output reg [63:0] axis1_hls_calculated_abs_pos,
    output reg [63:0] axis2_hls_calculated_abs_pos
);

    reg [2:0] prev_state;

module tb_get_abs_pos_state_machine;
    reg clk;
    reg rst;
    reg init_state_machine;
    reg hls_done;
    reg hls_ready;
    reg [31:0] axis1_hw_counter;
    reg [31:0] axis1_set_position_part1;
    reg [31:0] axis1_set_position_part2;
    reg [31:0] axis1_counts_per_m;
    reg [31:0] axis2_hw_counter;
    reg [31:0] axis2_set_position_part1;
    reg [31:0] axis2_set_position_part2;
    reg [31:0] axis2_counts_per_m;

    reg [63:0] selected_axis_hls_calculated_abs_pos;
    wire start_hls_calculations;

    wire [2:0] state;
    wire [31:0] selected_axis_hw_counter;
    wire [31:0] selected_axis_set_position_part1;
    wire [31:0] selected_axis_set_position_part2;
    wire [31:0] selected_axis_counts_per_m;

    wire [63:0] axis1_hls_calculated_abs_pos;
    wire [63:0] axis2_hls_calculated_abs_pos;
    get_abs_pos_state_machine ttb_get_abs_pos_state_machine
    (
        .clk(clk),

```

Abbildung 20 Test 1

```

    init_state_machine_reg <= init_state_machine;
    hls_done_reg <= hls_done;
    hls_ready_reg <= hls_ready;
end
end

// get absolute position for X and Y Axis State Machine
always @(posedge clk or negedge rst) begin
    if (!rst) begin
        state <= 0;
        prev_state <= 0;
        start_hls_calculations <= 0;
        selected_axis_hw_counter <= 0;

        selected_axis_set_position_part1 <= 0;

        selected_axis_set_position_part2 <= 0;
        selected_axis_counts_per_m <= 0;

    end
    else begin
        prev_state <= state;
        start_hls_calculations <= 0;

        // state transition
        case (state)

```

```

        .init_state_machine(init_state_machine),
        .hls_done(hls_done),
        .hls_ready(hls_ready),

        .axis1_hw_counter(axis1_hw_counter),
        .axis1_set_position_part1(axis1_set_position_part1),
        .axis1_set_position_part2(axis1_set_position_part2),
        .axis1_counts_per_m(axis1_counts_per_m),
        .axis2_hw_counter(axis2_hw_counter),
        .axis2_set_position_part1(axis2_set_position_part1),
        .axis2_set_position_part2(axis2_set_position_part2),
        .axis2_counts_per_m(axis2_counts_per_m),
        .selected_axis_hls_calculated_abs_pos(selected_axis_hls_calculated_abs_pos),

        .start_hls_calculations(start_hls_calculations),

        .state(state),

```

Abbildung 21 Test 2

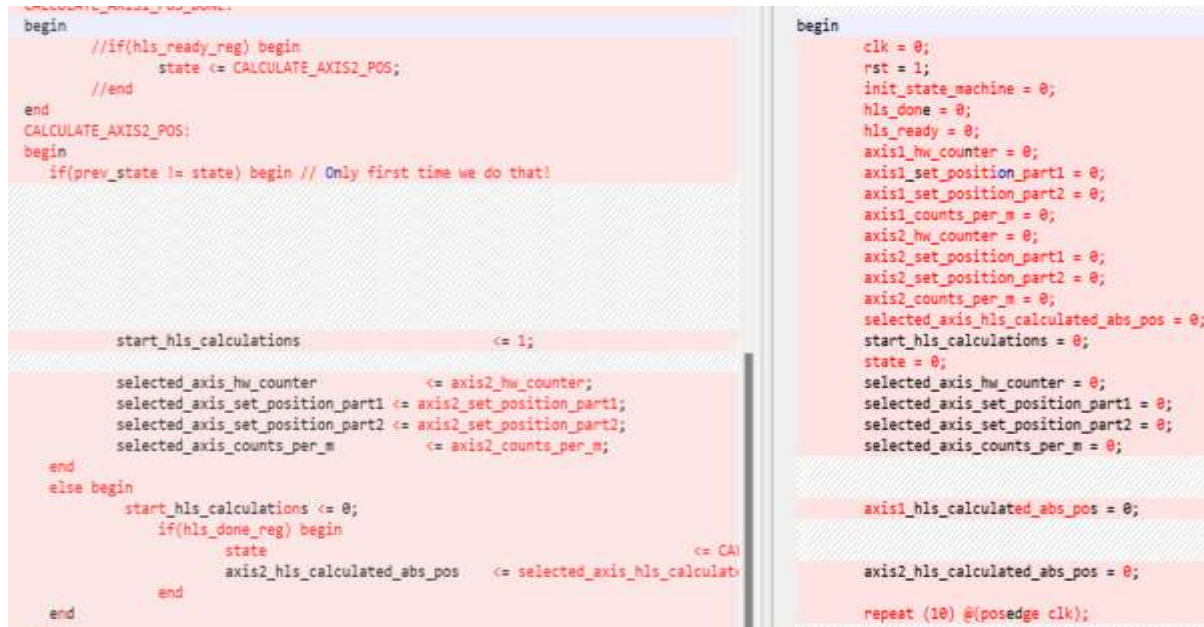


Abbildung 22 Test 3

A7 Benutzer Dokumentation

Einleitung

Diese Dokumentation beschreibt, wie Sie das TestBenchGenerator-Tool verwenden können, um Java-Dateien zu kompilieren und das Test Bench Vorlage auszuführen.

Schritt 1: Den Dateipfad öffnen

1. Starten Sie die Eingabeaufforderung auf Ihrem Computer.
2. Navigieren Sie zum Speicherort Ihrer Java-Projekte mithilfe des Befehls „cd“. Zum Beispiel:

“cd C:\JavaProjects\testbench2\TestBenchGenerator\src\main\java”

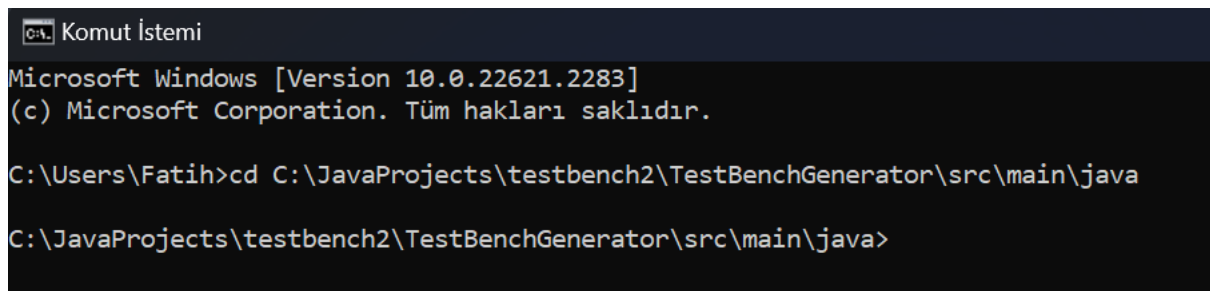


Abbildung 23 Datei Öffnen

FPGA-KONSOLEANWENDUNG

FPGA - Automatische Generierung von Test Bench Vorlagen

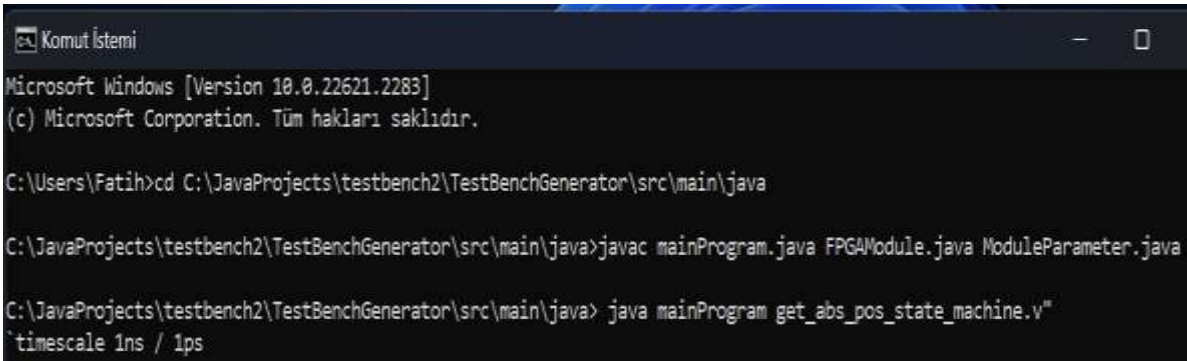
Sie sollten nun im Verzeichnis „C:\JavaProjects\testbench2\TestBenchGenerator\src\main\java sein“, wie in Abbildung 23 dargestellt.

Schritt 2: Java-Quellcode kompilieren

1. Sobald Sie sich im korrekten Verzeichnis befinden, geben Sie den folgenden Befehl ein, um Ihre Java-Dateien zu kompilieren:

```
“javac mainProgram.java FPGAModule.java ModuleParameter.java”
```

Dieser Befehl kompiliert die drei Java-Dateien: **mainProgram.java**, **FPGAModule.java** und **ModuleParameter.java**.



```
Komut İstemi
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\Fatih>cd C:\JavaProjects\testbench2\TestBenchGenerator\src\main\java

C:\JavaProjects\testbench2\TestBenchGenerator\src\main\java>javac mainProgram.java FPGAModule.java ModuleParameter.java

C:\JavaProjects\testbench2\TestBenchGenerator\src\main\java> java mainProgram get_abs_pos_state_machine.v"
'timescale 1ns / 1ps
```

Abbildung 24 Programm Kompilieren

Schritt 3: Test Bench Vorlage ausführen

1. Um die kompilierten Java-Dateien auszuführen und die Test Bench Vorlage zu starten, geben Sie den folgenden Befehl ein:

```
“java mainProgram get_abs_pos_state_machine.v”
```

Dies führt das Hauptprogramm mit der Datei „ get_abs_pos_state_machine.v „ als Argument aus.

2. Sie sollten nun die Ausgabe der Test Bench Vorlage auf Ihrem Bildschirm sehen.

Abschluss

Nachdem Sie die oben genannten Schritte befolgt haben, sollten Sie in der Lage sein, Ihre Java-Quelldateien problemlos zu kompilieren und die Test Bench Vorlage erfolgreich auszuführen.

FPGA-KONSOLENANWENDUNG

FPGA - Automatische Generierung von Test Bench Vorlagen

Die Ausgabe wird die Vorlage sein, von der einige unten aufgeführt sind:

```
Komut İstemi
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\Fatih>cd C:\JavaProjects\testbench2\TestBenchGenerator\src\main\java

C:\JavaProjects\testbench2\TestBenchGenerator\src\main\java>javac mainProgram.java FPGAModule.java ModuleParameter.java

C:\JavaProjects\testbench2\TestBenchGenerator\src\main\java> java mainProgram get_abs_pos_state_machine.v"
\timescale 1ns / 1ps

module tb_get_abs_pos_state_machine;
reg clk;
reg rst;
reg init_state_machine;
reg hls_done;
reg hls_ready;
reg [31:0] axis1_hw_counter;
reg [31:0] axis1_set_position_part1;
reg [31:0] axis1_set_position_part2;
reg [31:0] axis1_counts_per_m;
reg [31:0] axis2_hw_counter;
reg [31:0] axis2_set_position_part1;
reg [31:0] axis2_set_position_part2;
reg [31:0] axis2_counts_per_m;
reg [63:0] selected_axis_hls_calculated_abs_pos;
wire start_hls_calculations;
wire [2:0] state;
wire [31:0] selected_axis_hw_counter;
wire [31:0] selected_axis_set_position_part1;
wire [31:0] selected_axis_set_position_part2;
```

Abbildung 25 Ausgabe

```
Komut İstemi
get_abs_pos_state_machine ttb_get_abs_pos_state_machine
(
    .clk(clk),
    .rst(rst),
    .init_state_machine(init_state_machine),
    .hls_done(hls_done),
    .hls_ready(hls_ready),
    .axis1_hw_counter(axis1_hw_counter),
    .axis1_set_position_part1(axis1_set_position_part1),
    .axis1_set_position_part2(axis1_set_position_part2),
    .axis1_counts_per_m(axis1_counts_per_m),
    .axis2_hw_counter(axis2_hw_counter),
    .axis2_set_position_part1(axis2_set_position_part1),
    .axis2_set_position_part2(axis2_set_position_part2),
    .axis2_counts_per_m(axis2_counts_per_m),
    .selected_axis_hls_calculated_abs_pos(selected_axis_hls_calculated_abs_pos),
    .start_hls_calculations(start_hls_calculations),
    .state(state),
    .selected_axis_hw_counter(selected_axis_hw_counter),
    .selected_axis_set_position_part1(selected_axis_set_position_part1),
    .selected_axis_set_position_part2(selected_axis_set_position_part2),
    .selected_axis_counts_per_m(selected_axis_counts_per_m),
    .axis1_hls_calculated_abs_pos(axis1_hls_calculated_abs_pos),
    .axis2_hls_calculated_abs_pos(axis2_hls_calculated_abs_pos)
);

initial
begin
    clk = 0;
    rst = 1;
```

Abbildung 26 Ausgabe

```

Komut Istermi
axis2_set_position_part2 = 0;
axis2_counts_per_m = 0;
selected_axis_hls_calculated_abs_pos = 0;
start_hls_calculations = 0;
state = 0;
selected_axis_hw_counter = 0;
selected_axis_set_position_part1 = 0;
selected_axis_set_position_part2 = 0;
selected_axis_counts_per_m = 0;
axis1_hls_calculated_abs_pos = 0;
axis2_hls_calculated_abs_pos = 0;

repeat (10) @(posedge clk);
rst = 0;

repeat (10) @(posedge clk);

repeat (10) @(posedge clk);
rst = 1;

end

always #20 clk <= ~clk;

endmodule

C:\JavaProjects\testbench2\TestBenchGenerator\src\main\java>

```

Abbildung 27 Ausgabe

A8 Tabelle Soll-/Ist-Vergleich (detailliert)

Projektphase	Geplant	Tatsächlich	Differenz
1. Analysephase			
Ist-Analyse	2h	2h	0h
Use-Case Diagramm	1h	1h	0h
Erstellen des Lastenheftes	1h	1h	0h
2. Konzeption			
Soll-Konzept	1h	1h	0h
Aufbau der Programmstruktur	3h	3h	0h
Konzeption der Ausgabeformat	1h	1h	0h
Geschäftslogik planen	2h	2h	0h
Pflichtenheft	2h	2h	0h
3. Planung			
Wirtschaftlichkeit prüfen	2h	2h	0h
Zeitplanung	1h	1h	0h
Kostenplanung	1h	1h	0h
Planung der Tests	2h	2h	0h
4. Realisierung			
Lesen der Eingabedatei	4h	4h	0h
Bestimmen der Eingänge und Ausgänge	4h	4h	0h
Instantieren des Moduls zum Schreiben der Testbench	3h	3h	0h
Testbench als Ausgabe in eine Datei schreiben	3h	3h	0h
Integration der Eingabedaten und erwarteten Ausgabewerte	6h	6h	0h
Implementieren der Simulationsschleife	6h	6h	0h
5. Validierung			
Durchführen von umfangreichen Tests	7h	6h	-1h
Fehler beheben und Verbessern der Anwendung	4h	4h	0h
6. Abschluss			
Soll-/Ist-Vergleich	2h	2h	0h
Wirtschaftliche Nachbetrachtung	2h	2h	0h
Abgabe/ Präsentation	3h	3h	0h
7. Dokumentation			
Dokumentation verfassen	15h	16h	+1h
Benutzerdokumentation erstellen	2h	2h	0h

Tabelle 9 Soll - Ist Vergleich detailliert