Interruptfähige Erweiterung für flink

Patrick Good

Vertiefungsarbeit 2 in MSE Computer Science
Referent: Prof. Dr. Urs Graf

OST – Ostschweizer Fachhochschule

23. November 2023

Aufgabenstellung

Beschreibung und Aufgaben:
Unser flink System soll um Komponenten wie Schrittmotortreiber und Ansteuerung für Reflexlichtsensoren erweitert werden. Flink soll zudem interruptfähig gemacht werden und so den Linuxkernel signalisieren können. Passende Interruptserviceroutinen sollen auf externe Ereignisse an den flink-Komponenten reagieren können.

Zusammenfassung

Diese Arbeit behandelt die Erweiterung des flink Projektes[3]. Dabei soll dieses Projekt um drei Komponenten erweitert werden:

Schrittmotor:
Diese Komponente bietet eine flexible und zuverlässige Möglichkeit mehrere Schrittmotoren in einer Anwendung zu steuern, präzise Bewegungen auszuführen und auf bestimmte Ereignisse zu reagieren. Es ermöglicht die nahtlose Integration von Schrittmotoren in verschiedene Systeme und Anwendungen.

Optischer Reflektionssensor:
Diese Komponente bietet eine flexible und zuverlässige Möglichkeit mehrere Reflektionslichtschranken in einer Anwendung zu steuern, Daten zu sammeln und auf erfasste Ereignisse zu reagieren. Es ermöglicht die nahtlose Integration von Lichtschranken in verschiedene Systeme und Anwendungen.

Interrupts:
Dadurch kann das Polling auf eine flink Komponente vermieden werden. Dies reduziert die CPU-Belastung, da die Komponente nun in der Lage ist, die Anwendung zu benachrichtigen, wenn eine Bedingung oder ein Ereignis eintritt.

Um diese Erweiterungen einzubinden, wurden alle vier Bereiche angepasst:

VHDL Part:
Anpassungen und Erweiterungen in der Hardwarebeschreibungssprache VHDL für die Integration neuer Hardwarefunktionen.

Linux Kernel:
Anpassungen im Linux Kernel, um die Kompatibilität und Interaktion mit neuen Erweiterungen sicherzustellen, einschließlich Treiberentwicklung.

Userspace Bibliothek:
Aktualisierung und Erweiterung von Bibliotheken für die Benutzerumgebung von Linux, um den Zugriff auf neue Funktionen zu ermöglichen.

Python Bibliothek:
Anpassung und Erweiterung der Python-Bibliothek, um neue Features in Python-Anwendungen zu integrieren.

Der Code für die einzelnen Bereiche ist in den Git-Repositories zu finden unter:

VHDL:	https://github.com/flink-project/flinkvhdl
Kernel:	https://github.com/flink-project/flinklinux
Userspace Bibliothek:	https://github.com/flink-project/flinklib
Python Bibliothek:	https://github.com/flink-project/flinkpython

Inhaltsverzeichnis

Stand der Technik

1.1 Ausgangslage

Als Grundlage dieses Projektes dient die Vertiefungsarbeit 1[1], die vor einem halben Jahr abgeschlossen wurde. Daraus wurde die nachfolgende Zusammenfassung kopiert, um einen genauen Überblick über dieses Projekt zu haben.

> Mit dieser Arbeit soll die Grundlage geschaffen werden, um das Systemtechnikprojekt von Java Deep auf Linux mit Python umstellen zu können. Als Grundlage wird hierzu der Xilinx Zynq 7000 System on a Chip verwendet, welcher 1Gigabyte Ram und eine ARM Cortex A9 Dual-Core als CPU verwendet. Auf dem integriertem FPGA wird flink verwendet, um die Hardwareschnittstelle zu erstellen. Dieses stellt Hardwarepins mit diversen unterschiedlichen Funktionen zur Verfügung, um Aktoren und Sensoren an das Zynq 7000 System anzuschliessen.
>
> Um dieses Ziel zu erreichen, wurde die Umsetzung in mehrere Schritte aufgeteilt:
>
> **Im ersten Schritt** wurde ein Linux-Image mittels PetaLinux für den Zynq 7000 erstellt. Petalinux wird direkt von Xilinx angeboten. Dazu bietet der Hersteller des Entwicklungsboards, auf welchem der SoC sitzt, ein File mit den Grundkonfigurationen an.
>
> **Im zweiten Schritt** wurde das Konfigurations-File für das FPGA in das Projekt hinzugefügt. Dies sorgt dafür, dass der FPGA vor dem Aufstarten von Linux richtig konfiguriert wird. Als Grundlage wurde hierfür das flink Projekt, welches an der OST-Buchs entwickelt wurde, verwendet.
>
> **Im dritten Schritt** wurden die Kernel- und die Userspace-Bibliothek zum Linux hinzugefügt. Dabei wurde der Funktionsumfang von der Kernel-Bibliothek erweitert. Die erweiterte Funktion ist die Kommunikation vom Kernel mit dem FPGA über den AXI-Bus. Dann wurde analysiert, ob die Bibliotheken «Thread-Save» sind.
>
> **Im letzten Schritt** wurden eine C-Bibliothek und ein C-Python Wrapper hinzugefügt. Die C-Bibliothek beinhaltet einen Regler für Motoren, wobei der Regler als Java-Code schon existierte. Die C-Python Wrapper sind Abstraktionen, damit C-Bibliotheken im Python Code ohne Probleme ausgeführt werden können.

Technische Umsetzung

2.1 Schrittmotoren

2.1.1 Funktionsbeschrieb

Das Modul ermöglicht die präzise Steuerung eines Schrittmotors über die Signale A, A', B und B'. Es bietet drei Betriebsmodi, den Freilauf, den Halbschrittbetrieb und den Vollschrittbetrieb mit ein oder zwei Phasen. Das Modul löst einen Interrupt aus, sobald der Motor zum Stillstand kommt.

2.1.2 Rampensteuerung

Um hohe Geschwindigkeiten zu erreichen und einem Verlust von Schritten vorzubeugen, wird der Motor immer mittels einer Rampe hochgefahren und auch wieder mittels einer Rampe gestoppt. Die Rampe ist mittels drei Parameter konfigurierbar: Startgeschwindigkeit, Sollgeschwindigkeit und Beschleunigung.

Die Rampe wird durch einen Zustandsautomaten gesteuert, welchen den Motor hoch- und herunterfahren lässt. Der Benutzer hat nur indirekt Einfluss auf diesen Zustandsautomat. Stattdessen kann der Benutzer durch einen Kontrollprozess den Zustandsautomat beeinflussen. Der Zustandsautomat entscheidet selbstständig, ob die Anfragen, welche vom Kontrollprozess gesendet werden, umsetzbar sind.

Zusätzlich wird dieser Zustandsautomat von einem übergeordneten Prozess geschützt. Der Prozess sorgt dafür, dass diejenigen Register, welche über den AXI-Bus vom Benutzer verändert werden können, keinen unerwünschten Effekt hervorrufen. Dieser speichert den Wert des Benutzers und leitet ihn in genau definierten Zuständen an den Zustandsautomaten weiter.

Die Abbildung des Zustandsdiagramms ist im Anhang auf der Seite 41 zu finden.

2.1.3 Betriebsmodi

Deaktivierter Modus:

Im deaktivierten Modus befindet sich der Motor im freien Drehmodus. Wobei keine Wicklung magnetisiert ist, was eine freie Bewegung ermöglicht.

Schrittbetrieb:

- Der Motor führt eine definierte Anzahl von Schritten aus.
- Eine Rampensteuerung sorgt für sanftes Beschleunigen und Abbremsen.
- Rampenparameter wie Startgeschwindigkeit, Sollgeschwindigkeit und Beschleunigung sind vor dem Start des Motors konfigurierbar.

Fixierte Geschwindigkeit:

- Der Motor fährt die Rampe hinauf und erreicht seine Sollgeschwindigkeit.
- Der Motor verbleibt in dieser Geschwindigkeit, bis ein Stoppsignal empfangen wird. Dann bremst der Motor über die definierte Rampe ab.

- Anpassbare Sollgeschwindigkeit und Beschleunigung während des Betriebs ist möglich.

2.1.4 VHDL-Design

Das Design wurde hierfür in vier Module aufgeteilt:

Driver:
Ist zuständig für die Steuerung der Signale A, A', B und B'. In Abhängigkeit von Motorenbetrieb Halbschritt, Vollschritt (ein oder zwei Phasen) sowie vorwärts oder rückwärts. Dabei macht dieses Modul bei jedem Trigger, den es erhält, einen Schritt im konfigurierten Modus.

Speed Controller:
Dieses Modul ist dafür verantwortlich, das Trigger Signal zu generieren, das an den Treiber gesendet wird. Die Generierung dieses Signals erfolgt in Abhängigkeit von den Rampenparametern. Des Weiteren ist dieses Modul zuständig für den Betrieb des ausgewählten Betriebsmodus.

Prescale clock and register control:
Es stellt sicher, dass die Register, einschließlich der Rampenregister, zur richtigen Zeit an den Speed Controller übertragen werden, um einen reibungslosen Betrieb zu gewährleisten. Der Prescaler ist erforderlich, um die maximale Schrittweite des Schrittmotors einzustellen. Es ist jedoch zu beachten, dass eine längere Zeitspanne zwischen zwei Schritten die Auflösung der einzelnen Schritte beeinträchtigen kann. Dieses Modul generiert auch den Interrupt, welcher auslöst wird, sobald der Motor zum Stillstand kommt.

AXI-Bus:
Die Hauptaufgabe dieses Moduls ist die Steuerung des AXI-Bus. Es war nicht notwendig, das gesamte Modul neu zu schreiben, sondern nur die Bereiche, die Schreib- oder Lesevorgänge auf dem AXI-Bus ausführen. Dieses Modul verwendet das Sub-Modul "prescale clock and register control". Die Register, welche über den AXI-Bus erreichbar sind, sind auf der flink Webseite[3] beschrieben.

2.1.5 IP-Design (Block Design)

In der Abbildung 2.1 ist das Frontend mit den zu konfigurierenden Parametern zu sehen, die nach der Implementierung nicht mehr angepasst werden können. Auf der linken Seite der Abbildung sind die Signale zu sehen, die das Modul benötigt.

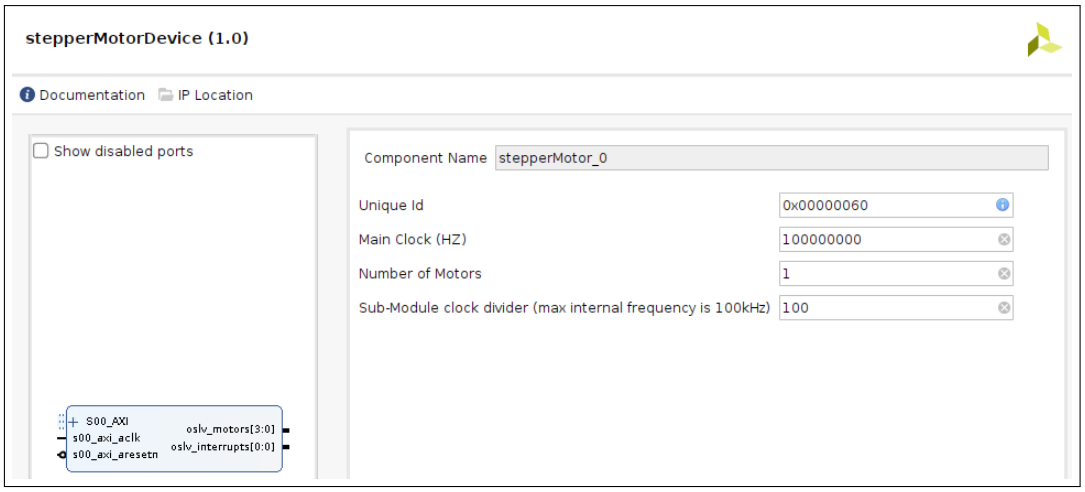

Abbildung 2.1: IP des Schrittmotors

Parameter

- Unique Id: Eindeutige Identifikationsnummer des Moduls.
- Main Clock: Die Grundfrequenz, mit der dieses Modul arbeitet, in Hz.
- Number of Motors: Beschreibt die Anzahl Motoren, welche betrieben werden können.
- Sub-Modul clock divider: Beschreibt den Teiler, welcher auf den Main clock angewendet wird, um den Speedcontroller anzutreiben. Wird im Modul "prescale clock and register control" verwendet.

Eingangssignale

- S00_AXI: Die Leitungen des AXI-Bus. Dies beinhaltet z.B. die Daten- und Adressleitungen.
- s00_axi_aclk: Der Grundtakt, mit dem das Modul betrieben werden soll. Muss mit dem Takt des AXI-Bus übereinstimmen.
- s00_axi_aresetn: Signal, um das Modul in den Ausgangszustand zurückzusetzen. Dieser Signaleingang ist "Aktiv Low".

Ausgangssignale

- oslv_motors: Sind die Signale, welche auf die Motoren geführt werden. Wobei die Anordnung der Signale wie folgt ist: LSB -> MSB (Motor1(A,A',B,B'), Motor2(A,A',B,B'), ...)
- osl_interrupts: Sind die Interrupt-Leitungen, welche zur Weiterverarbeitung auf den Interrupt-Controller (Kapitel: 2.5) geführt werden müssen.

2.1.6 Userspace Bibliothek und Bash Programm

Um dieses Modul, welches das flink Projekt erweitert, auch von einer Userspace Applikation bedienen zu können, wurde der Funktionsumfang der bestehenden "libflink"

erweitert.
Diese Funktionen werden von dem Bash Programm verwendet, um das Modul zu testen und auf einfache Weise zu betreiben.
Der Code dieser beiden Erweiterungen befindet sich im Git Repository des flink Projektes[3].

2.2 Optischer Reflektionssensor

Es wird der Sensor TCRT1000 verwendet. Für diesen Sensor existiert eine Platine (Abbildung 2.2) für das Experimentiersystem [2] des OST Campus Buchs.

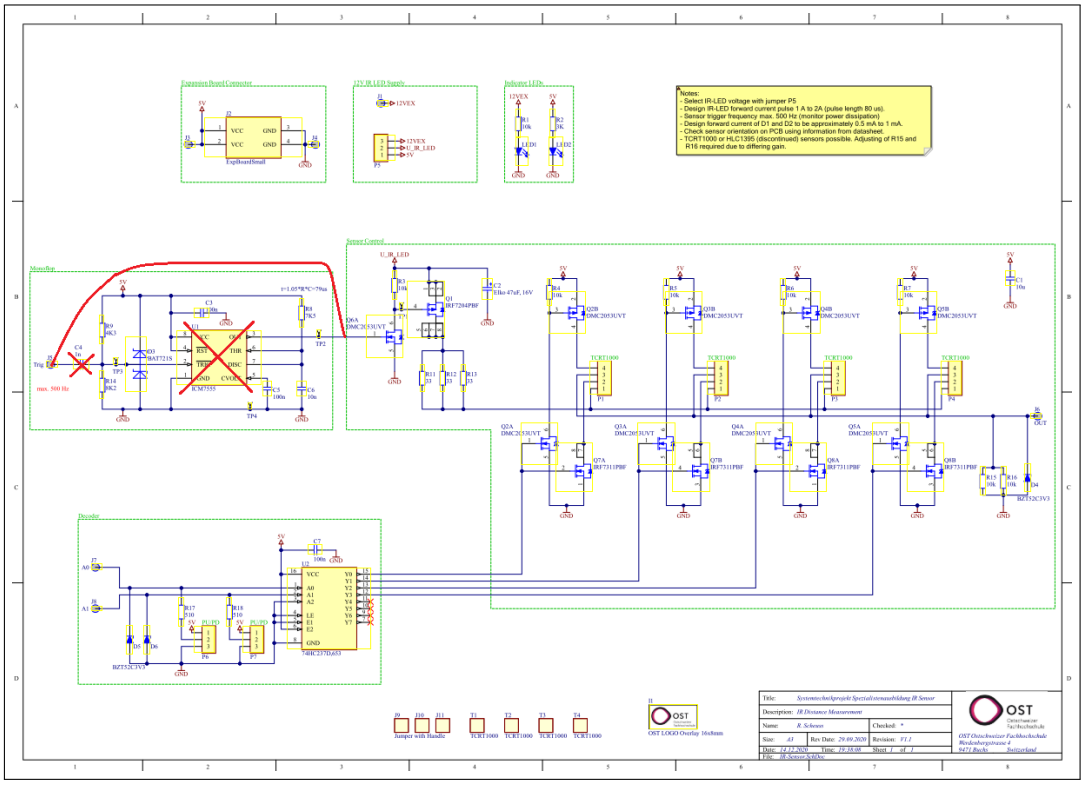

Abbildung 2.2: Schaltplan für Optischer Reflektionssensor

Das Schema musste überarbeitet werden, damit es mit dem flink Modul kompatibel ist. Dazu wurden die Bauteile C4 und U1 ausgelötet und stattdessen wurde eine Brücke von Trig(J5) zum TP2 eingelötet. Um das analoge Signal zu digitalisieren, wurde der ADC7476 verwendet.

2.2.1 Funktionsbeschrieb

Das Modul wurde so ausgelegt, um den Betrieb mehrerer Reflektionslichtschranken zu ermöglichen. Der ausgewählte Sensor wird für eine Zeitdauer aktiviert. Wenn diese Zeit abläuft, wird der analoge Wert des Sensors eingelesen. Die erfassten Werte des ADC sind über den AXI-Bus abrufbar. Darüber hinaus ermöglicht das Modul die Konfiguration von Interrupts für jeden Sensor mithilfe einer Hysterese.

2.2.2 Funktionsweise

Sensorauswahl mittels Encoder:
Das Modul verwendet einen Encoder zur Auswahl des aktiven Sensors. Es erfolgt ein kontinuierlicher Rundlauf, bei dem der Decoder die Sensoren nacheinander auswählt.

Aktivierung der Lichtschranke:
Nach Auswahl des Sensors wird die entsprechende Reflektionslichtschranke für eine vordefinierte Zeitdauer (in der Regel 80 Mikrosekunden) aktiviert. Dies ermöglicht die Erfassung von reflektiertem Licht oder anderen physikalischen Parametern durch den Sensor.

Analogwert-Erfassung:
Kurz vor Ablauf der Aktivierungszeit wird der analoge Wert des Sensors eingelesen. Dieser Wert wird erfasst und in einem speziellen Register gespeichert, um später über den AXI-Bus abgerufen zu werden.

Interrupt-Konfiguration mit Hysterese:
Das Modul ermöglicht die Konfiguration von Interrupts für jeden Sensor. Eine Hysterese kann eingestellt werden, um festzulegen, wann der Interrupt ausgelöst wird. Wenn der analoge Wert des Sensors die Hysterese unterschreitet oder überschreitet, wird der Interrupt ausgelöst.

Wertverarbeitung und Weitergabe:
Die erfassten analogen Werte der Sensoren können über den AXI-Bus an andere Komponenten oder Verarbeitungseinheiten weitergegeben werden. Dies ermöglicht eine Echtzeitüberwachung der Lichtschranken und die Verarbeitung der erfassten Daten gemäß den Anwendungsanforderungen.

2.2.3 VHDL-Design

Die Hardware wurde hierfür in vier Module aufgeteilt:

ADC7476:
Dies Modul existiert bereits im flink Projekt [3]. Dises Modul wird genutzt, um mit dem ADC7476 zu kommunizieren. Es bietet ein Register, in dem der aktuelle digitale Wert des Analogsignals angezeigt wird.

Puls Generator:
Dieses Modul erzeugt einen Impuls sowie einen Trigger. Der Trigger wird kurz vor dem Abfallen der Flanke des Impulses aktiv. Der Impuls wird hierfür mit einem Pin verbunden, welcher wiederum direkt mit dem Trig(J5) auf dem Sensorboard verbunden wird. Der Trigger wird benötigt, um den digitalisierten analogen Wert des ADC7476 im richtigen Moment auszulesen.

Interrupt Generator:
Dieses Modul wird benötigt, um die Interrupts in Abhängigkeit von der eingestellten Hysterese (einstellbar über den AXI-Bus) zu generieren. Die Interrupts werden als Pulse ausgegeben. Diese Pulse werden vom Interrupt Controller (Kapitel: 2.5) weiterverarbeitet.

AXI-Bus:

Die Hauptaufgabe dieses Moduls ist die Steuerung des AXI-Bus. Es war nicht notwendig, das gesamte Modul neu zu schreiben, sondern nur die Bereiche, die Schreib- oder Lesevorgänge auf dem AXI-Bus ausführen. Um die Daten dem Bus zur Verfügung zu stellen, wird das Modul Reflektionssensor verwendet. Die Register, welche über den AXI-Bus erreichbar sind, sind auf der flink Webseite[3] beschrieben.
Sowie nutzt dieses Modul die bereits beschriebenen Module: ADC7476, Pulsgenerator und Interrupt Generator. Jeder Trigger des Puls Generatormoduls löst die Erfassung des ADC7476-Werts aus, der dann im Register des entsprechenden Sensors gespeichert wird. Die Decoder Signale bestimmen, welcher Sensor jeweils betrieben wird. Bei jedem Trigger wird jeweils zum nächsten Sensor gewechselt.

2.2.4 IP-Design (Block Design)

In der Abbildung 2.3 ist das Frontend mit den zu konfigurierenden Parametern zu sehen. Die Parameter können nach der Implementierung nicht mehr angepasst werde. Auf der linken Seite der Abbildung sind die Signale zu sehen, die das Modul benötigt.

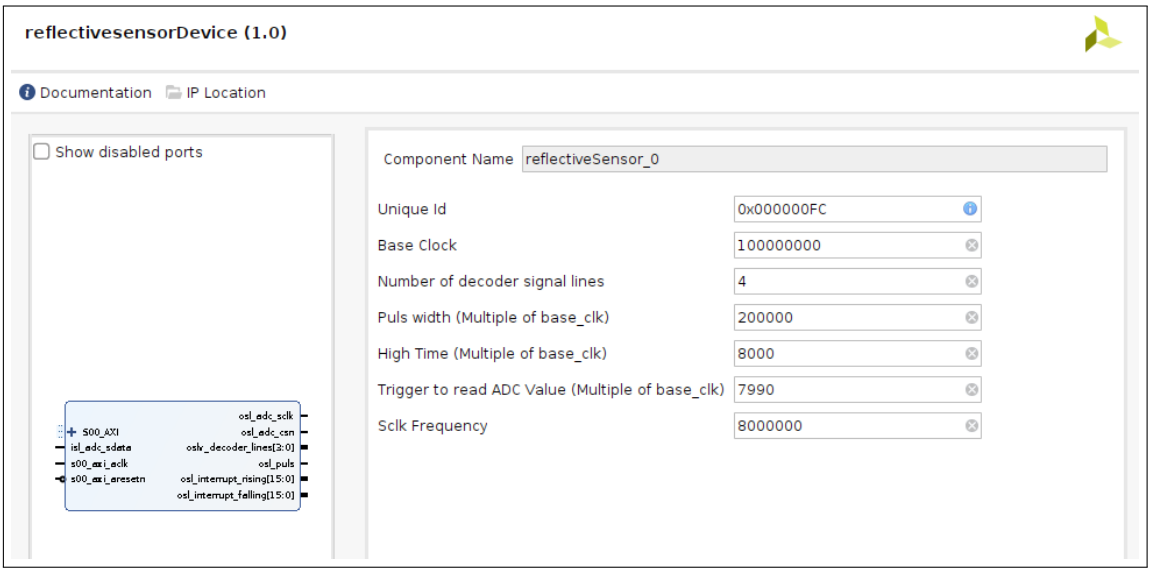

Abbildung 2.3: IP des Reflektionssensors

Parameter:

- Unique Id: Eindeutige Identifikationsnummer des Moduls.
- Base Clock: Die Grundfrequenz, mit der dieses Modul arbeitet, in Hz.
- Number of decoder signal lines: Legt fest, wie viele Signalleitungen (oslv_decoder_lines) das Modul verlassen. Dies steht in direktem Zusammenhang mit der Anzahl der betreibbaren Sensoren (Sensoren = $2^{\text{Number of decoder signal lines}}$). Es ist zu beachten, dass der Decoder diese Anzahl unterstützen muss.
- Puls width (Multiple of base_clk): Zeit zwischen zwei Impulsen (in Vielfachen des Grundtaktes).
- High Time (Multiple of base_clk): Gibt an, wie lange die Pulsdauer sein soll.
- Trigger to read ADC Value (Multiple of base_clk): Gibt den Zeitpunkt an, an dem der Trigger ausgelöst werden soll, um den ADC-Wert kurz vor dem Ende des oben genannten Pulses einzulesen.
- Sclk Frequency: Beschreibt die Geschwindigkeit, in welcher der serielle Bus arbeiten soll, wird auch benötigt für die Verbindung zum ADC.

Eingangssignale:

- S00_AXI: Die Leitungen des AXI-Bus. Dies beinhaltet z.B. die Daten- und Adressleitungen.
- isl_adc_sdata: Die Datenleitung welche von dem ADC7476 kommt.
- s00_axi_aclk: Der Grundtakt, mit dem das Modul betrieben werden soll. Muss mit dem Takt des AXI-Bus übereinstimmen.
- s00_axi_aresetn: Signal, um das Modul in den Ausgangszustand zurückzusetzen. Dieser Signaleingang ist "Aktiv Low".

Ausgangssignale:

- osl_adc_sclk: Ist der serielle Takt, welcher zum Lesen des ADC benötigt wird.
- osl_adc_csn: Der "Chip Select" für den ADC.
- oslv_decoder_lines: Die Verbindungen zu dem Decoder auf der elektrischen Schaltung. Die Anzahl der Verbindungen sind abhängig von dem Parameter "Number of decoder signal lines".
- osl_puls: Das Pulssignal wird benötigt, um die Sensoren anzusteuern. Hängt mit den beiden Parametern "Puls width" und "High time" zusammen.
- osl_interrupt_rising und osl_interrupt_falling: Sind die Interrupt Leitungen, welche zur Weiterverarbeitung auf den Interrupt Controller (Kapitel: 2.5) geführt werden müssen.

2.2.5 Userspace Bibliothek und Bash Programm

Um dieses Modul, welches das flink Projekt erweitert, auch von einer Userspace Applikation bedienen zu können, wurde der Funktionsumfang der bestehenden "libflink" erweitert.
Diese Funktionen werden von dem Bash Programm verwendet, um das Modul zu testen und auf einfache Weise zu betreiben.
Der Code dieser beiden Erweiterungen befindet sich im Git Repository des flink Projektes[3].

2.3 Interrupt Multiplexer

2.3.1 Funktionsbeschrieb

Ein Multiplexer ist eine Schaltung oder ein Gerät, das in der Elektronik verwendet wird, um eine Vielzahl von Eingangssignalen auf eine begrenzte Anzahl von Ausgangssignalen umzuleiten.

Da die flink Geräte eine grosse Anzahl an flink IRQ Signale generieren, wird ein Multiplexer verwendet. Somit können die flink IRQ Signale, mit einer kleineren Anzahl von IRQ Leitungen verbunden werden.
Jede IRQ Leitung kann so, durch die Konfiguration über den AXI-Bus, einem flink IRQ zugeordnet werden.

2.3.2 VHDL-Design

Die Hardware wurde hierfür in zwei Module aufgeteilt:

Multiplexer:
Dieses Modul ist verantwortlich für die Verbindung des ausgewählten flink IRQ Signals mit der IRQ Leitung. Ein einzelnes Modul kann nur eine IRQ Leitung verwalten. Um mehrere IRQ Leitungen verwalten zu können, werden mehrere dieser Module benötigt. Dies wird im übergeordneten Modul (AXI-Bus) realisiert.

AXI-Bus:
Die Hauptaufgabe dieses Moduls ist die Steuerung des AXI-Bus. Es war nicht notwendig, das gesamte Modul neu zu schreiben, sondern nur die Bereiche, die Schreib- oder Lesevorgänge auf dem AXI-Bus ausführen. Ausserdem implementiert dieses Modul die notwendige Anzahl des Modules "Multiplexer", um die IRQ Linien zu verwalten. Zur Konfiguration der IRQ Linien stellt dieses Modul für jede IRQ Linie ein Register für den AXI-Bus zur Verfügung. Die Register, welche über den AXI-Bus erreichbar sind, sind auf der flink Webseite[3] beschrieben.

2.3.3 IP-Design (Block Design)

In der Abbildung 2.4 ist das Frontend mit den zu konfigurierenden Parametern zu sehen. Die Parameter können nach der Implementierung nicht mehr angepasst werde. Auf der linken Seite der Abbildung sind die Signale zu sehen, die das Modul benötigt.

Parameter:

- Unique Id: Eindeutige Identifikationsnummer des Moduls.
- Base Clock: Die Grundfrequenz, mit der dieses Modul arbeitet, in Hz.
- Number of inputs: Grösse des Eingangsvektors (Anzahl flink IRQ Signale).
- Number of outputs: Grösse des Ausgangsvektors (Anzahl IRQ Linien).
- Value of disabeld interrupt: Wert, welcher ein deaktivierter IRQ Ausgang besitzt (0 oder 1).

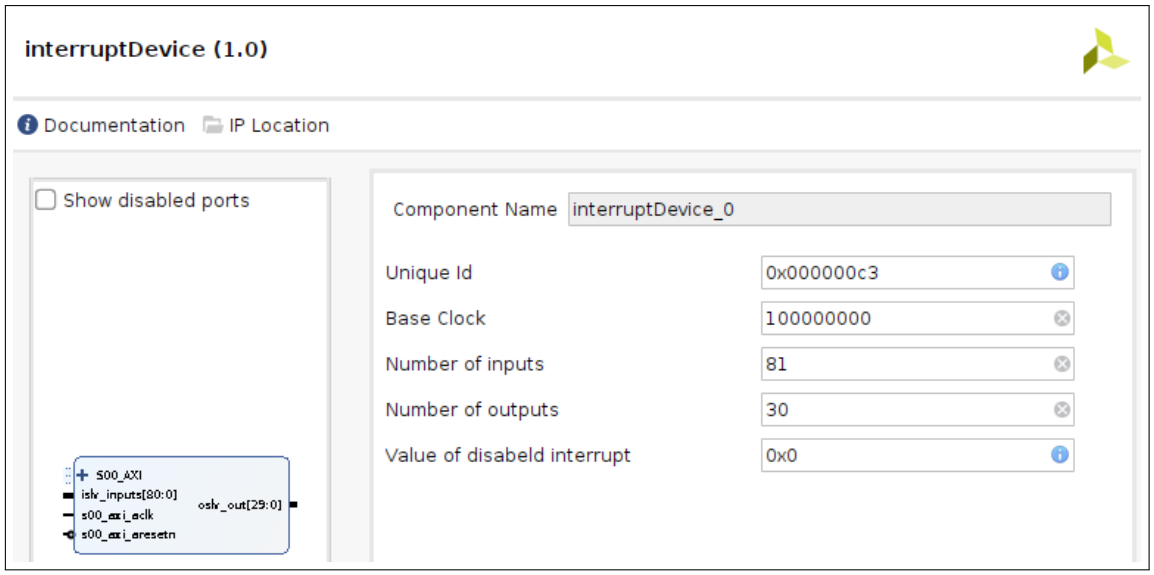

Abbildung 2.4: IP des Interrupt Multiplexers

Eingangssignale:

- S00_AXI: Die Leitungen des AXI-Bus. Dies beinhaltet z.B. die Daten- und Adressleitungen.
- islv_inputs: Der Eingangsvektor für die flink IRQ Signale.
- s00_axi_aclk: Der Grundtakt, mit dem das Modul betrieben werden soll. Muss mit dem Takt des AXI-Bus übereinstimmen.
- s00_axi_aresetn: Signal, um das Modul in den Ausgangszustand zurückzusetzen. Dieser Signaleingang ist "Aktiv Low".

Ausgangssignale:

- oslv_out: Der Ausgangsvektor für die IRQ Leitungen.

2.3.4 Userspace Bibliothek und Bash Programm

Um dieses Modul, welches das flink Projekt erweitert, auch von einer Userspace Applikation bedienen zu können, wurde der Funktionsumfang der bestehenden "libflink" erweitert.
Diese Funktionen werden von dem Bash Programm verwendet, um das Modul zu testen und auf einfache Weise zu betreiben.
Der Code dieser beiden Erweiterungen befindet sich im Git Repository des flink Projektes[3].

2.4 GPIO Interrupts

Um die Interrupt Fähigkeit weiter auszubauen, wurde das bestehende GPIO-Modul des flink[3] Projektes erweitert.
Diese Erweiterung beinhaltet die Implementierung der Interrupt-Generierung für jeden GPIO-Pin, für steigende und für fallende Flanken. Die Interrupt-Generierung wird aktiviert, wenn der Pin als Eingang deklariert wird.
Zusätzlich zur Interrupt-Generierung wurde eine Entprellungsfunktion implementiert, die über den AXI-Bus für jeden Pin individuell eingestellt werden kann. Die Entprellung ist als Vielfaches des Basistaktes realisiert. Das zeitliche Ablaufdiagramm befindet sich im Anhang auf der Seite 40.
Die Register, welche über den AXI-Bus erreichbar sind, sind auf der flink Webseite[3] beschrieben.

2.4.1 IP-Design (Block Design)

Das angepasste Frontend ist in der Abbildung 2.5 zu sehen. Die Änderung ist mit einem roten Kreis markiert.

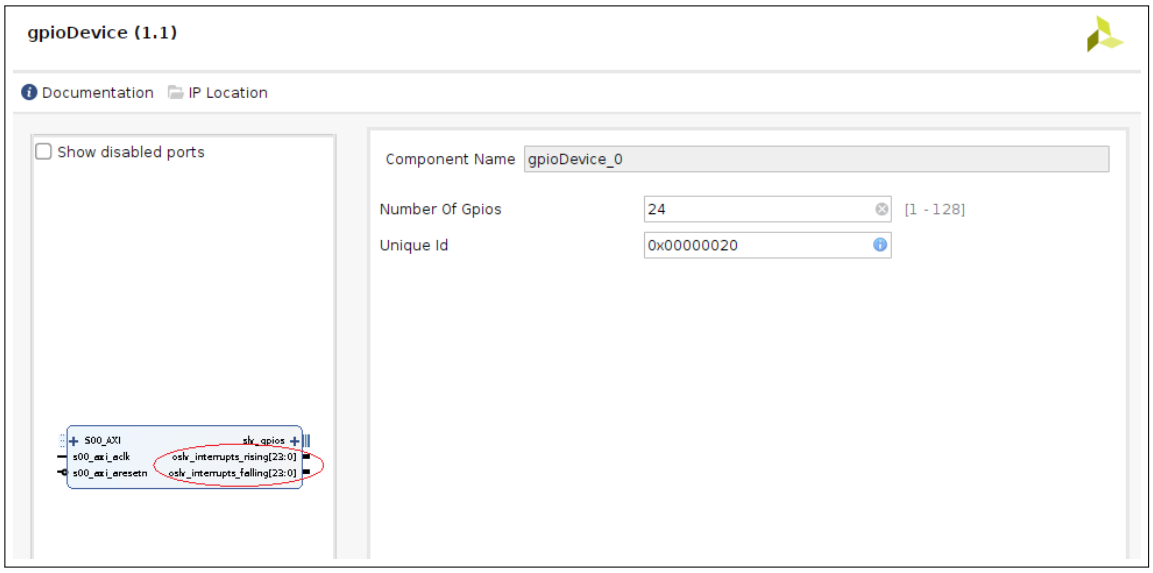

Abbildung 2.5: Angepasste IP des GPIO Device

Neue Ausgangssignale:

- oslv_interrupts_rising und oslv_interrupts_falling: Sind die Interrupt-Leitungen, welche zur Weiterverarbeitung auf den Interrupt-Controller (Kapitel: 2.5) geführt werden müssen.

2.5 Interrupts

Um die Notwendigkeit des ständigen Pollings eines Status zu vermeiden, werden die flink Komponenten um die Interrupt Funktionalität erweitert. Diese Erweiterung zielt darauf ab, den Prozessor zu entlasten.

Der Fluss der Interrupts ist in der Grafik im Anhang auf der Seite 42 dargestellt. Auf der linken Seite befinden sich die Interrupt Quellen, während auf der rechten Seite die Ziele der Interrupts zu finden sind. In diesem Kontext handelt es sich um eine Python Funktion, die vom Benutzer selbst verfasst werden muss.

2.5.1 FPGA Part

Ein Interrupt Signal wird im FPGA durch drei Module geleitet:

Kombinations Modul "Concat":
Dieses Modul wird von der Vivado Standardbibliothek zur Verfügung gestellt. Es kombiniert alle eingehenden Signale und Vektoren, die von den flink Komponenten kommen, zu einem einzigen Vektor. Dies ist wichtig, da die nachfolgenden Module nur einen Vektor zur Verfügung stellen.

flink Interrupt Multiplexer:
Dieses Modul befindet sich in der flink Bibliothek und reduziert einen großen Vektor auf einen Vektor mit maximal 32 Signalen.
Durch die Anbindung an den AXI-Bus, kann dieses Modul zur Laufzeit konfiguriert werden. Je nach Konfiguration wird ein Signal des Eingangsvektors mit einem Signal des Ausgangsvektors verknüpft.
Dabei ist es möglich, ein Signal des Eingangsvektors mit mehreren Signalen des Ausgangsvektors zu verknüpfen. Umgekehrt ist dies nicht möglich.

Xillinx Interrupt Controller "intc":
Dieses Modul ist ebenfalls in der Vivado Standardbibliothek enthalten. Es ist für die Interrupt Abschlusslogik zuständig. Dabei speichert es die Zustände der eingehenden Interrupts in einem Register, das vom "intc Treiber" im Linux Kernel verwaltet wird. Der Ausgang ist ein einzelnes Signal, das an den "GIC" weitergeleitet wird.

2.5.2 CPU Komponente

Der GIC ist für die Verwaltung der verschiedenen Interrupt Quellen zuständig, die von allen möglichen Peripheriegeräten kommen. Die Hauptfunktion dieser Komponente besteht darin, die eingehenden IRQs zu verwalten und korrekt an die entsprechende CPU weiterzuleiten.

2.5.3 Linux Kernel

Die Verarbeitung im Kernel wurde in zwei Module aufgeteilt. Es ist zu beachten, dass im Folgenden zwei verschiedene Interrupts beschrieben werden. Unter "HW-IRQ" wird der Interrupt beschrieben, der vom "GIC" kommt. Bei "Kernel IRQ" handelt es sich um einen Kernel-internen Interrupt, der keinen direkten Hardwareauslöser hat.

Intc Treiber:
Ist für die Verwaltung des "intc" auf dem FPGA zuständig. Einfach ausgedrückt: Dieser Treiber liest beim Empfang des "HW-IRQ" die Register des "intc" auf dem FPGA aus und sendet je nach den ausgelesenen Daten bis zu 32 "Kernel-IRQs", von denen jeder eine eigene Nummer hat.

flink Kern Modul:
Dieses Modul ist für die Umwandlung der "Kernel-IRQs" Signale verantwortlich, die von einem Userspace empfangen werden können. Damit dieses Modul das Signal an den richtigen Userspace Prozess senden kann, muss sich der Userspce Prozess zunächst für jeden "Kernel-IRQ", den er empfangen möchte, beim Modul registrieren.
Dazu führt das Modul für jeden "Kernel-IRQ" eine Liste der Userspace Prozesse, an die das dem "Kernel-IRQ" entsprechende Signal gesendet werden soll.

2.5.4 Userspace C-Library

Diese Bibliothek ist in der Interrupt Funktionalität sehr schlank und bietet nur Konfigurationsfunktionen für den "flink Interrupt Multiplexer" und die Interrupt Verwaltung im Kernel an.
Die Funktionen für die Interrupt Verwaltung im Kernel umfassen dabei die Registrierung, die Deregistrierung des IRQ und die Abfrage der Signalnummer.
Die Funktionen für den "flink Interrupt Multiplexer" umfassen das Lesen und Schreiben der Konfigurationsregister, um die Eingänge mit den Ausgängen zu verbinden.

2.5.5 Zusammenfassung für den Anwender

Diese Anleitung ist eine Hilfestellung für den Programmierer, welcher die Interrupt Funktionalität nützen möchte.

Vorgehensweise:

1. Schreiben der Funktion, die ausgeführt wird, wenn der IRQ ausgelöst wird.
2. Erstellen der Interrupt-Klasse, die sich in der flink Python-Bibliothek befindet.
3. Konfigurieren des "flink Interrupt Multiplexers".
4. Registrieren der in Schritt 1 erstellten Funktion zusammen mit der IRQ-Nummer mithilfe der Python-Funktion "registerIRQ" in der flink Python-Bibliothek.

Erklärungen zu den einzelnen Schritten:

1. Schreiben Sie eine Funktion, die nur das Notwendigste enthält, um so schnell wie möglich ausgeführt zu werden. Verwenden Sie Synchronisierungstechniken wie Mutexe oder Semaphoren für gemeinsam genutzte Daten, da diese Funktion parallel zum Rest des Codes ausgeführt wird, wenn ein Interrupt eintrifft. Berücksichtigen Sie, dass diese Funktion mehrmals gleichzeitig aufgerufen werden kann, wenn mehrere Interrupts schnell hintereinander ausgelöst werden.

2. Verwenden Sie die in der Python-Bibliothek vorhandene Interrupt-Klasse, um die notwendigen Funktionen für Konfiguration und Registrierung zu nutzen. Legen Sie die Klasse gemäß den Python-Standards an.

3. Nur, wenn ein Multiplexer vorhanden ist, entscheiden Sie, welche Interrupt-Quelle auf welches IRQ-Signal gelegt werden soll. Ordnen Sie jedem Ausgang einen Kanal zu, der über eine Nummer mit einer Interrupt-Quelle verbunden werden kann. Diese Nummer sollte aus einer Tabelle stammen, die vom Systemadministrator bereitgestellt wird. Konfigurieren Sie dies mithilfe der Python-Funktion "setIRQmultiplexerValue" in der Bibliothek, wie in Schritt 2 erwähnt.

4. Verknüpfen Sie die im Schritt 1 erstellte Funktion mit der entsprechenden IRQ-Nummer. Führen Sie dies mit Hilfe der Python-Funktion "registerIRQ" in der Bibliothek aus, wie in Schritt 2 beschrieben.

2.6 Device Tree

Ein "Device Tree Node" ist eine strukturierte Beschreibungseinheit im Linux Kernel, die verwendet wird, um die Hardwarekonfiguration und die Hardwareressourcen eines Systems zu definieren. Dies ermöglicht es dem Kernel, die Geräte im System zu erkennen, zu initialisieren und zu verwalten.

In Petalinux erfolgt die Erweiterung des Device Trees im Dateipfad:
`project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`

Der Node im Code Ausschnitt 2.1 beschreibt die flink Komponente im FPGA.

```
/include/ "system-conf.dtsi"
/  {
// this node is tested with kernel 5.15.19-rt29-xilinx-v2022.1
flink_axi_0: flink_axi_registers@7aa00000 {
    compatible = "ost,flink-axi-1.0";
    interrupt-parent = <&axi_intc_0>;
    interrupts = <0 0>;
    reg = <0x7aa00000 0x9000>;
    ost,flink-signal-offset = <34>;
    ost,flink-nof-irq = <30>;
};
};
```

Code-Snippet 2.1: Flink device tree node (AXI Anbindung)

2.6.1 Beschreibung der Variablen

- **compatible:** Gibt die Treiberversion an, die mit diesem Node kompatibel ist.
- **interrupt-parent:** Zeigt auf den übergeordneten Interrupt-Controller-Knoten, in diesem Fall "<&axi_intc_0>" von Xilinx[5], der als Interrupt-Elternteil für diesen Node dient.
- **interrupts:** Definiert die Interrupt-Quellen.
- **reg:** Spezifiziert den Adressbereich und die Registergröße der Komponente.
- **ost,flink-signal-offset:** Beschreibt den Offset, mit dem Signale an den User Space gesendet werden.
- **ost,flink-nof-irq:** Gibt an, wie viele Interrupts empfangen werden können.

2.7 Python Klassen

Um die oben beschriebenen Elemente: Schrittmotoren, Reflektionssensor und Interrupts für Anwender zugänglich zu machen, wurde das Projekt "flink Python" um drei Klassen erweitert.

Diese Klassen vereinfachen den Zugriff über die C-Bibliothek auf die VHDL flink Komponenten. Der Anwender wird so von der Komplexität, der Sprache C, abgekoppelt. Somit muss sich der Anwender nicht mit der Komplexität der C-Bibliothek wie Pointer und Datentypen auseinandersetzen.

Der Code befindet sich im Git-Repository des flink Projektes[3].

2.7.1 Schrittmotoren

Diese Klasse umschliesst für den Schrittmotor Funktionen, die in der C-Bibliothek definiert sind. Desweitern stellt diese Bibliothek weitere Funktionen zur Verfügung für:
Diese Klasse umschliesst für den Schrittmotor Funktionen, die in der C-Bibliothek definiert sind.

Konfiguration:
Es werden Funktionen bereitgestellt, mit denen die einzelnen Bits des Konfigurationsregisters gesetzt werden können. Dadurch wird vermieden, dass sich der Programmierer mit den einzelnen Bits und deren Bedeutung auseinandersetzen muss. Der Programmierer kann einfach die Funktion z.B. "start(...)" aufrufen und das Startbit im Konfigurationsregister wird gesetzt. Für jedes Bit in diesem Register gibt es eigene Funktionen, mit denen die Konfiguration durchgeführt werden kann.

Geschwindigkeiten und Beschleunigung:
Da diese Werte sehr abstrakt in den Registern der VHDL-Komponente abgelegt werden müssen, werden sie für den Anwender in ISO-Einheiten umgerechnet. So wird die Geschwindigkeit in [Schritte/s] angegeben. Für die Beschleunigung wird die Anzahl der Schritte von der Startgeschwindigkeit bis zur Sollgeschwindigkeit eingegeben.

Initialisierung und Geschwindigkeitsänderungen:
Da eine Änderung der Geschwindigkeits- und Beschleunigungsregister nicht jederzeit möglich ist und bei laufendem Motor die Reihenfolge der Registerbeschreibungen eingehalten werden muss, sind diese Parameter nicht direkt zugänglich. Aus diesem Grund werden Funktionen zur Verfügung gestellt, die eine einfache Initialisierung des Motors und die Einstellung der Drehzahlen ermöglichen.

2.7.2 Optischer Reflektionssensor

Diese Klasse bietet, neben den üblichen Funktionen wie dem direkten Zugriff auf die Register, auch Funktionen zur direkten Umrechnung des eingelesenen Sensorwertes in die Spannung an.

2.7.3 Interrupt

Diese Klasse enthält die Funktionen zur Konfiguration des Multiplexers und zur Registrierung der IRQ's im Kernel.

Ausblick

3.1 Weiterführende Ideen

Erweiterung des flink Projektes um weitere Module:
Eine Idee besteht darin, das flink Projekt um weitere Module zu erweitern, darunter beispielsweise einen Treiber für einen DC-Motor. Ein solches Modul würde es ermöglichen, DC-Motoren nahtlos in das flink Framework zu integrieren und ihre Steuerung in flink Anwendungen zu erleichtern. Ein weiteres denkbares Modul wären die Time-of-Flight-Sensoren zu integrieren, um deren Nutzung und Datenverarbeitung in flink Anwendungen zu optimieren. Diese Erweiterungen würden das flink Projekt noch vielseitiger und leistungsfähiger machen.

Ausbau der Interrupt-Fähigkeit:
Eine weitere Möglichkeit bietet die Erweiterung der Interrupt-Fähigkeiten im flink Projekt. Aktuell ist die Anzahl der verfügbaren Interrupts auf 30 begrenzt, da dies die Anzahl der Signale ist, die zwischen dem Linux Kernel und dem Userspace verwendet werden können. Durch den Ausbau dieser Kapazität könnten mehr Interrupts gleichzeitig verarbeitet werden, was die Möglichkeiten zur Integration von Interrupt-basierten Ereignissen in flink Anwendungen erweitern würde.

Erweiterung der Userspace Bibliothek auf andere Sprachen:
Schließlich besteht die Idee, die Userspace-Bibliothek des flink Projektes auf andere Programmiersprachen wie C++, C#, und Java zu erweitern. Dies würde es Entwicklern ermöglichen, flink-Anwendungen in verschiedenen Sprachen zu schreiben und die Flexibilität und Zugänglichkeit des Projektes zusätzlich steigern. Durch die Integration von flink in verschiedene Ökosysteme, könnten mehr Entwickler von den Funktionen und Möglichkeiten des Projektes profitieren.

Erweiterung der Plattformkompatibilität:
Ein weiterer vielversprechender Ansatz besteht darin, die Plattformkompatibilität des flink Projektes zu erweitern, um eine breitere Palette von Betriebssystemen und Plattformen zu unterstützen. Dies könnte die Integration von flink in Betriebssysteme wie Windows, iOS oder Echtzeitbetriebssysteme (RTOS) umfassen. Durch diese Erweiterung könnte das flink Projekt seine Reichweite auf eine größere Entwicklergemeinschaft ausdehnen und die Anwendbarkeit in verschiedenen Anwendungsbereichen verbessern, einschliesslich IoT-Geräten, mobilen Anwendungen und Echtzeit-Steuerungssystemen.

3.2 Schlussfolgerungen

In diesem Projekt haben wir Erweiterung des flink Projektes betrachtet. Während dieser Arbeit wurden verschiedene Aspekte des Projektes beleuchtet, angefangen von der Implementierung auf der VHDL-Ebene bis zur Anbindung an die Benutzeranwendungsebene mit Python.

Die, in dieser Arbeit entwickelten Erweiterungen, zeigten eine sehr gute Integrationsfähigkeit hinsichtlich Modularität und Flexibilität. Es ist jedoch zu beachten, dass die Anzahl der verfügbaren Interrupts aufgrund Kernel-interner Signalmechanismen auf 30 begrenzt ist. Diese Anzahl ist aber in den meisten Fällen ausreichend.

Danksagungen

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Erstellung dieser Vertiefungsarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Herr Prof. Dr. Urs Graf, der meine Vertiefungsarbeit betreut und begutachtet hat. Seine Anregungen und die konstruktive Kritik waren bei der Erstellung sehr hilfreich.

Zweitens ein Dankeschön an Herr Prof. Lazlo Arato für die Unterstützung bei den VHDL-Komponenten.

Ebenfalls bedanken möchte ich mich bei allen Mitarbeitern des Institutes "Ingenieurs Informatik", die mir mit viel Geduld, Interesse und Hilfsbereitschaft zur Seite standen.

Außerdem möchte ich Christine Good für das Korrekturlesen meiner Vertiefungsarbeit danken.

Patrick Good, 23. November 2023

Glossar

Aktiv high: Eine Logiksignal-Bezeichnung, bei der eine hohe Spannung als "aktiv" oder "eins" interpretiert wird.

Aktiv low: Eine Logiksignal-Bezeichnung, bei der eine niedrige Spannung als "aktiv" oder "eins" interpretiert wird.

High time: Die Dauer, während ein Signal auf einem hohen Pegel ist.

Puls width: Die Dauer eines einzelnen Impulses in einem periodischen Signal.

libflink: Die Userspace Bibliothek für Linux des flink Projektes.

Prescaler: Ein Schaltungselement, das die Frequenz eines Signals reduziert.

Frontend: Eine Schnittstelle oder ein Teil eines Systems, das direkt mit Benutzern oder anderen Systemen interagiert.

Git: Ein weit verbreitetes Versionskontrollsystem für die Zusammenarbeit an Softwareprojekten.

Git Repository: Ein Speicherort für Git-Projekte, in dem Versionskontrollinformationen gespeichert werden.

ADC7476: Ein Analog-Digital-Wandler (ADC).

Python: Eine weit verbreitete Programmiersprache (keine Schlange).

C: Eine Programmiersprache, die häufig für Systemsoftware und Embedded-Programmierung verwendet wird.

Interrupt: Ein Signal, das den normalen Programmablauf unterbricht, um auf bestimmte Ereignisse zu reagieren.

Userspace: Der Bereich eines Computerspeichers, in dem Benutzerprogramme ausgeführt werden.

Prozess: Ein laufendes Programm oder eine laufende Anwendung auf einem Computer.

Akronyme

ADC: Analog-Digital-Umsetzer (Analog-to-Digital Converter)

MSB: Höchstwertiges Bit (Most Significant Bit)

LSB: Niedrigstwertiges Bit (Least Significant Bit)

IRQ: Interrupt-Anforderung (Interrupt Request)

FPGA: Feldprogrammierbare (Logik-)Gatter Anordung (Field Programmable Gate Array)

SoC: System-on-a-Chip (System on a Chip)

AXI: Erweiterte Schnittstelle (Advanced eXtensible Interface)

VHDL: Hardware-Beschreibungssprache für integrierte hochgeschwindigkeits Schaltkreise (Very High-Speed Integrated Circuit Hardware Description Language)

Hz: Hertz (Frequenz)

GPIO: Allzweckeingabe/-ausgabe (General Purpose Input/Output)

GIC: Generischer Interrupt-Controller (Generic Interrupt Controller)

MMU: Speicherverwaltungseinheit (Memory Management Unit)

IRQ: Interrupt-Anfrage (Interrupt Request)

Abbildungsverzeichnis

Codeverzeichnis

Literaturverzeichnis

[1] Good, Patrick. Python auf dem Microzed-Board. Vertiefungsarbeit 1 des Masterstudienganges. Fachhochschule OST Campus Buchs(SG)

[2] Taranto, Fabian di. Ost Experimentiersystem. https://wiki.bu.ost.ch/infoportal/embedded_systems/experimentiersystem/start/. (abgerufen 28.04.2023)

[3] Graf, Urs. Flink-Project. https://flink-project.ch/start (abgerufen 22.10.2023)

[4] Johannes4Linux. Sending Signals. https://github.com/Johannes4Linux/Linux_Driver_Tutorial/tree/main/15_Sending_Signals (abgerufen am 16.10.23)

[5] Xillinx. Vivado IP. https://www.xilinx.com/products/intellectual-property/axi_intc.html#documentation (abgerufen 31.10.23)

Anhang

A GPIO Interrupt Zeitdiagramm

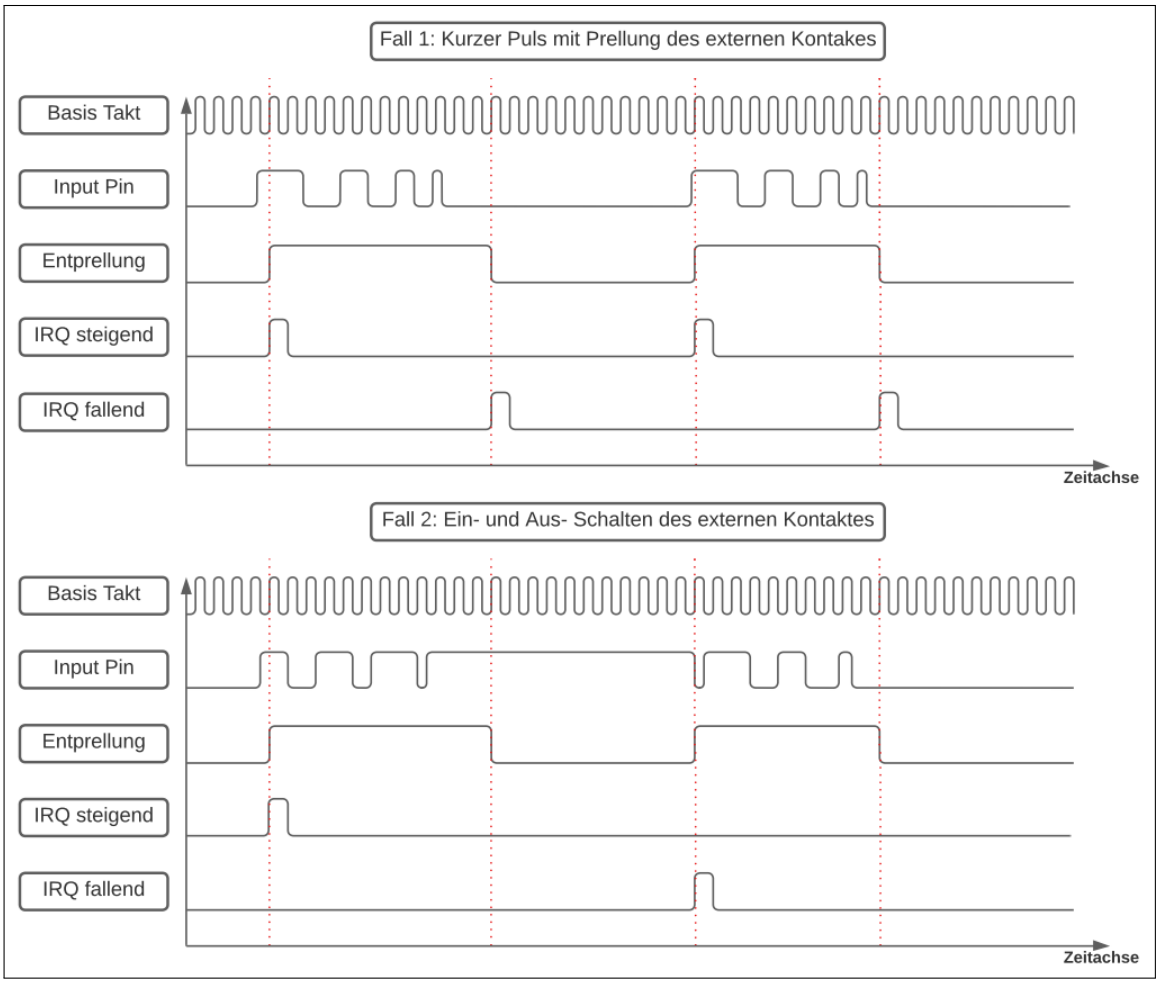

Abbildung 3.1: GPIO Interrupt Zeitdiagramm

B Zustandsautomat der Rampensteuerung

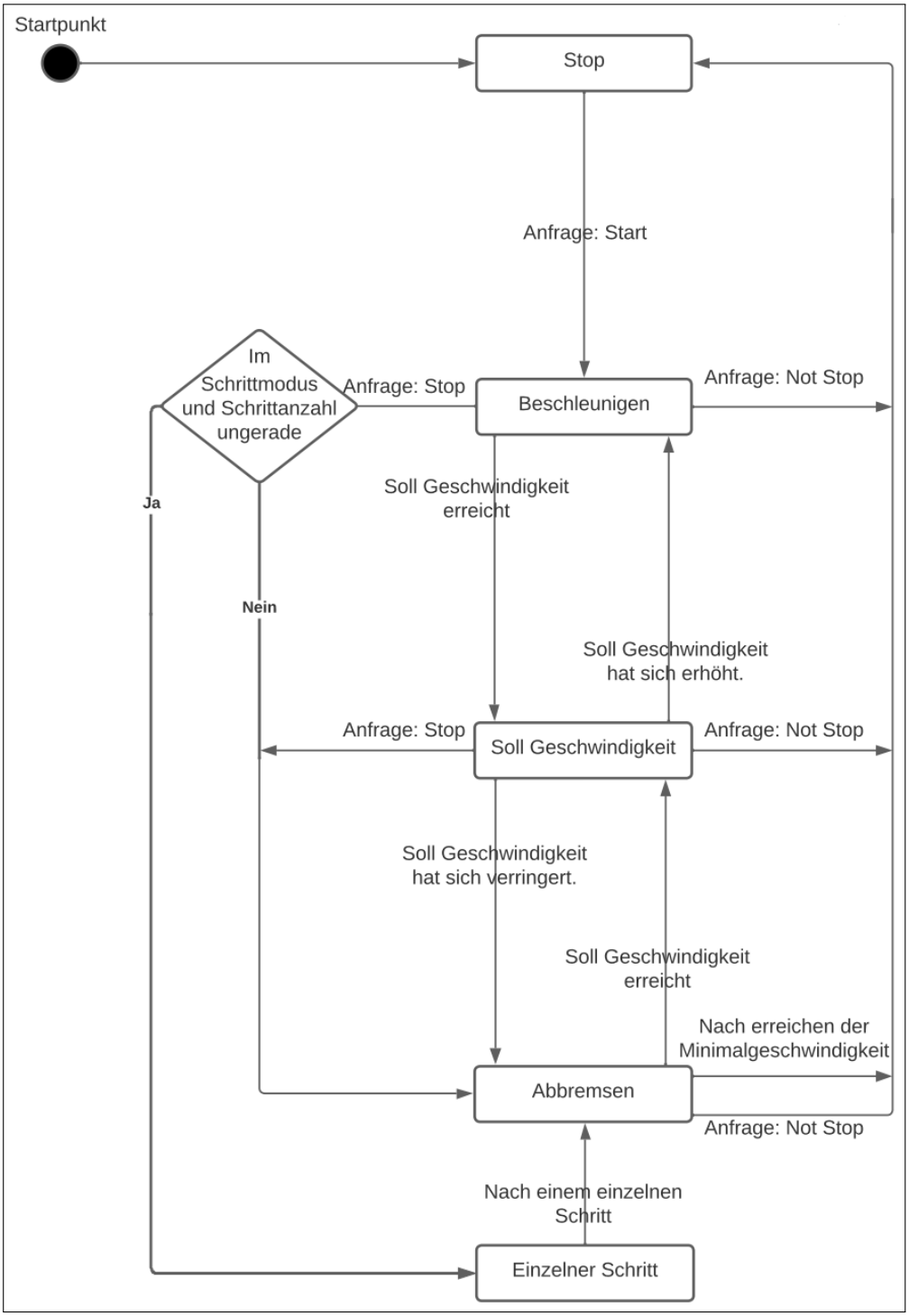

Abbildung 3.2: Zustandsautomat der Rampensteuerung

C Interrupt Signalflussdiagramm

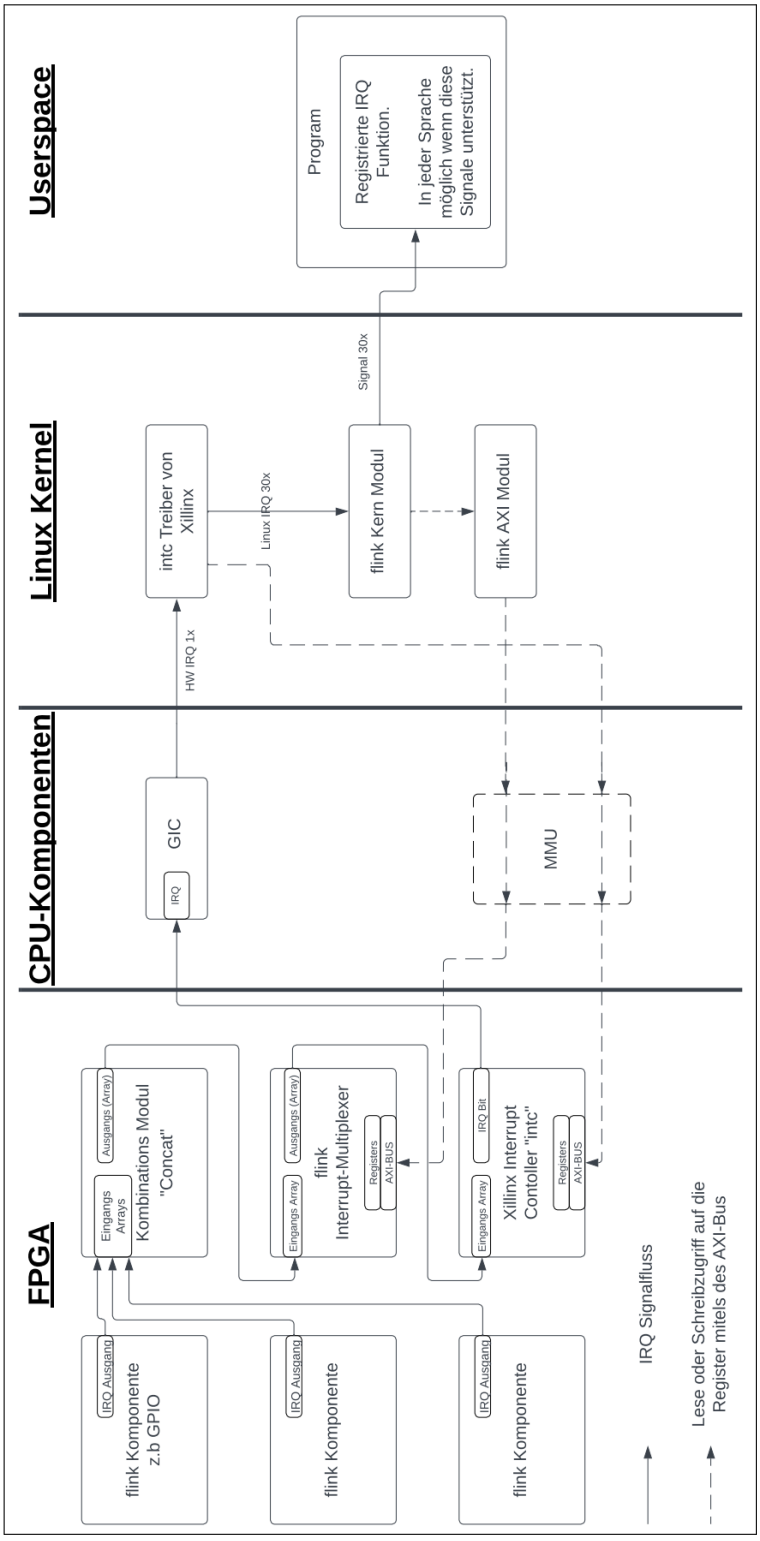

Abbildung 3.3: Interrupt Signalfluss von der Quelle bis zum Ziel

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die Vertiefungsarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Flums, 23. November 2023

_____

Patrick Good