Python auf dem Microzed-Board

Patrick Good

Vertiefungsarbeit 1 in MSE Computer Science
Referent: Prof. Dr. Urs Graf

OST – Ostschweizer Fachhochschule

17. Juli 2023

Aufgabenstellung

Beschreibung:
Auf einem Zynq7000 setzen wir ein Standard-Linuxsystem ein (ohne GUI). Dieses läuft auf beiden Kernen und greift über den AXI-Bus auf das interne FPGA zu. Dort drin sind mit Flink einzigartige Hardwareerweiterungen realisiert.

Aufgaben:
Unter Linux soll eine Applikation in Python auf flink-Devices zugreifen können. Grundlegende Dienste, die wir beispielsweise im Systemtechnikprojekt benötigen, wie Motorenregelungen oder der Zugriff auf ein externes WLAN-Modul, sollen in einer geeigneten Bibliothek oder in Kernelmodulen realisiert werden.

Zusammenfassung

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.

Inhaltsverzeichnis

Ausgangslage

Der Systemtechnik Studiengang am OST Campus Buchs führt in den ersten beiden Semestern des Lehrgangs mit den Studierenden ein Robotik Projekt durch. Dabei werden Roboter entwickelt, welche diverse Aufgaben erledigen und auch Hindernisse überwinden müssen. Um diese Roboter zu programmieren, wurde (vor 3 Jahren) das Systemtechnikprojekt mit Java Deep auf den MPC555 realisiert. Letztes Jahr (2022) wurde ein anderes Board für das Systemtechnikprojekt gewählt: Das Board MicroZed mit dem SoC Zynq-7000 von Xilinx. Die Programmierung wurde als Bare Metal Applikation auf diesem Prozessor mittels Java Deep entwickelt.
Dieses Board hat die Fähigkeit mit Linux betrieben zu werden. Somit haben zukünftige Systemtechnikprojekte eine viel grössere Software-Basis mit Linux zur Verfügung.

Es kann auf ein Labor kit[2] zurückgegriffen werden, welches bereits entwickelt ist. Dieses stellt diverse Peripherien zur Verfügung wie Motoren, LED, Schalter, W-Lan Modul, Spannungsversorgung, Distanzsensoren und vieles mehr. Die Peripherie kann mittels Verbindungsleitungen einfach mit dem MicroZed-Board verbunden werden.

Stand der Technik

2.1 Microzed 7010

Die Grundlage dieser Arbeit bildet das MicroZed 7010 Board von Avnet[1]. Als Herzstück dieses Bord wurde ein Zynq-7000 SoC verwendet. Zusätzlich ist auf dem Board ein 1GB grosser Arbeitsspeicher vorhanden (DDR3).

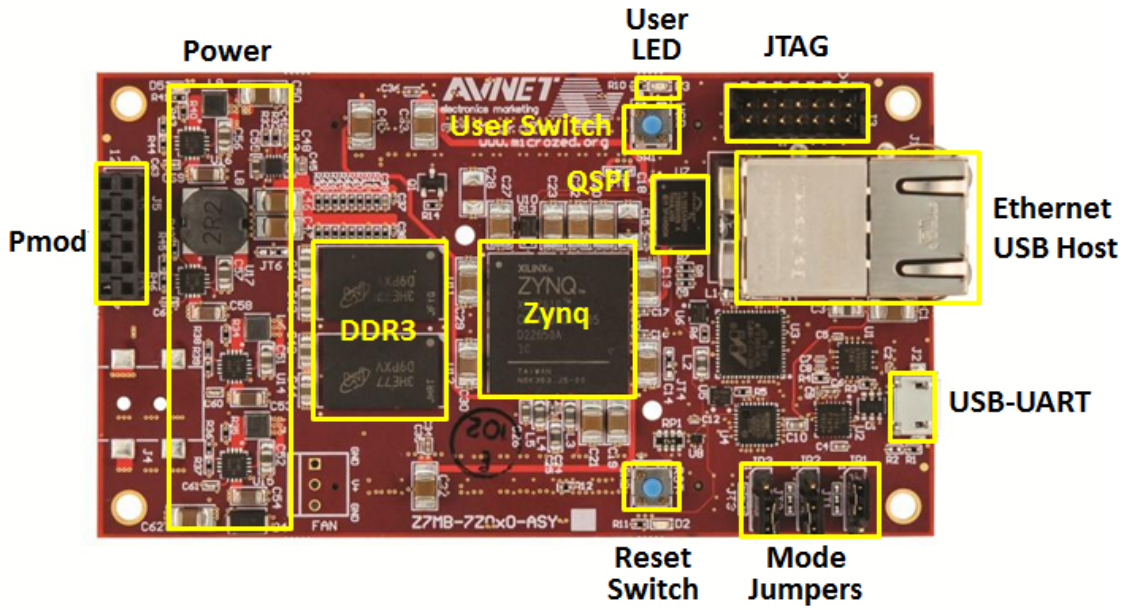

Abbildung 2.1: MicroZed Board mit Zynq7000 (https://linuxgizmos.com/tiny-arm-plus-fpga-sbc-gains-beefier-fpga/)

Ein weiterer Bestandteil sind die Jumpers. Mittels diesen wird die Bootquelle festgelegt, von welcher gestartet werden soll. Es stehen insgesamt drei Bootvarianten zur Verfügung. Für diese Arbeit benötigen wir die Option, welche von der SD-Karte bootet.

2.2 Xilinx Zynq-7000

Der Zynq-7000 ist ein SoC von der Firma Xilinx. Die zwei grössten Komponenten sind der FPGA und die beiden ARM Cortex-A9 Prozessoren. In der Abbildung 2.2 sind noch diverse Peripherien ersichtlich wie USB, Ethernet, GPIO, UART, usw.

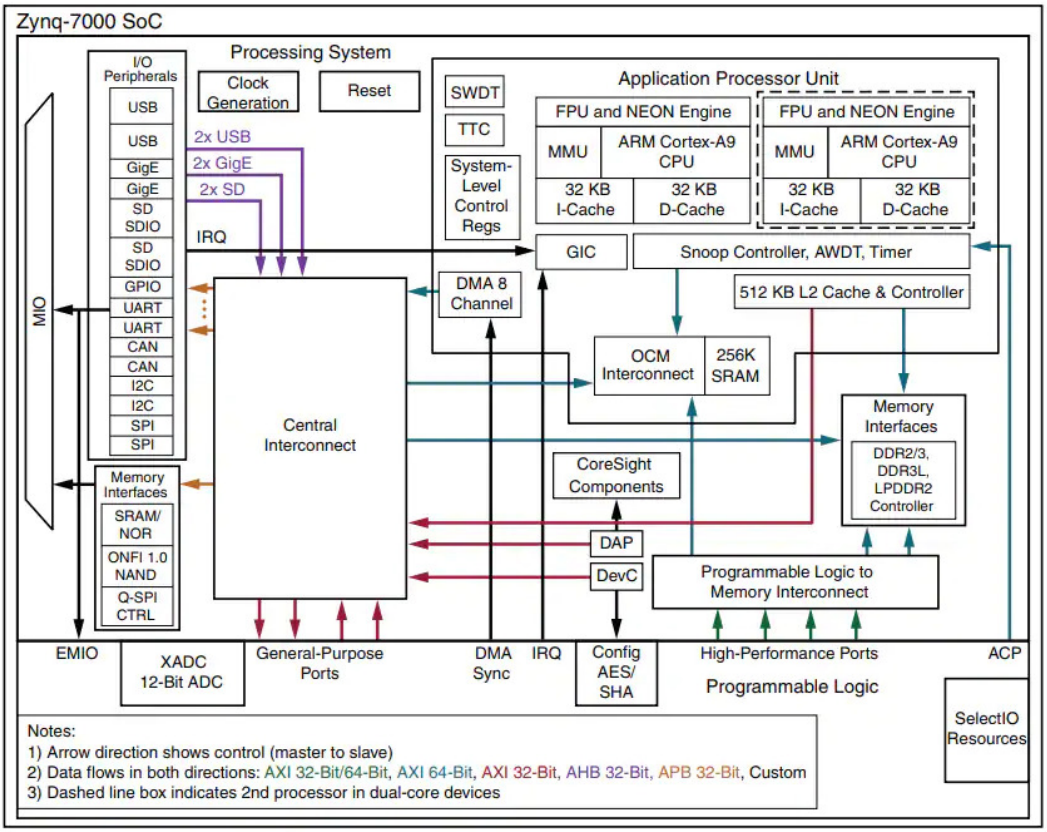

Abbildung 2.2: Zynq-7000 Architektur (Xilinx, Zynq-7000 SoC Technical Reference Manual, S.28)

Der PL ist ein FPGA, auf welchem später das «Programm» Flink betrieben wird. Flink wird nachfolgend erklärt. Die Kommunikation mittels dem PS erfolgt über den AXI-Bus. Der PL hat diverse Ausgänge zur Verfügung, welche auf weitere externe Hardware wie LED, Motorenansteuerung, Schalter usw. geführt werden können.

2.3 PetaLinux

"The PetaLinux Tools offers everything necessary to customize, build and deploy Embedded Linux solutions on AMD processing systems. Tailored to accelerate design productivity, the solution works with the AMD hardware design tools to ease the development of Linux systems for Versal, Zynq™ UltraScale+™ MPSoC, Zynq™ 7000 SoCs, and MicroBlaze™."[4].

PetaLinux ist auf dem Tool Yocto aufgebaut. Yocto wird ebenfalls benützt, um eine eigene Linux Distribution zu erstellen. PetaLinux erweitert so Yocto um Funktionalitäten. Des Weiteren bringt PetaLinux noch XSCT und die CLI-Tools mit.

PetaLinux kann ein Projekt mittels eines .bsp Files generieren. Ein solches File ist auf die jeweilige Hardware angepasst. Daher beinhaltet das .bsp File die Grundkonfiguration für die jeweilige Hardware, auf welchem Linux ausgeführt werden soll. Im Weiteren sind vorgebaute und vorkonfigurierte Komponenten in diesem File deponiert, um die Kompile-Zeit zu verringern.

Um das PetaLinux Projekt zu verwalten hat PetaLinux diverse Befehle. Diese beginnen immer mit den Prefix «petalinux-». Die wichtigsten davon sind:

- **create**: Wird benötigt, um ein Projekt oder eine eigene Applikation zu erzeugen.
- **config**: Wird benötigt, um den Kernel oder das Root-FS zu konfigurieren.
- **build**: Mittels dieses Befehls kann die komplette Distribution gebaut werden oder auch nur eine einzelne Komponente.
- **package**: Wird benötigt, um das BOOT.bin File zu erstellen. Dazu werden diverse Komponenten, welche der build Prozess generiert hat, eingebunden.

2.4 Flink

"flink can be read as fast link or as the German "flink" meaning fast"[3].

Flink ist ein Tool, welches an der Fachhochschule Ost Campus Buchs entwickelt wurde. Dabei kann Flink auf zwei Hauptteile aufgeteilt werden: Auf die FPGA- und Prozessor-Komponente. Wobei der Prozessor Teil diverse Komponenten enthält: Kernel, Userspace bei einem Linux System, und im Falle ohne Betriebssystem, eine Bare Metal Bibliothek (für C/C++ und Java Deep[7])Für diese Arbeit benötigen wir das FPGA, Linux Kernel und die Userspace-Bibliothek.

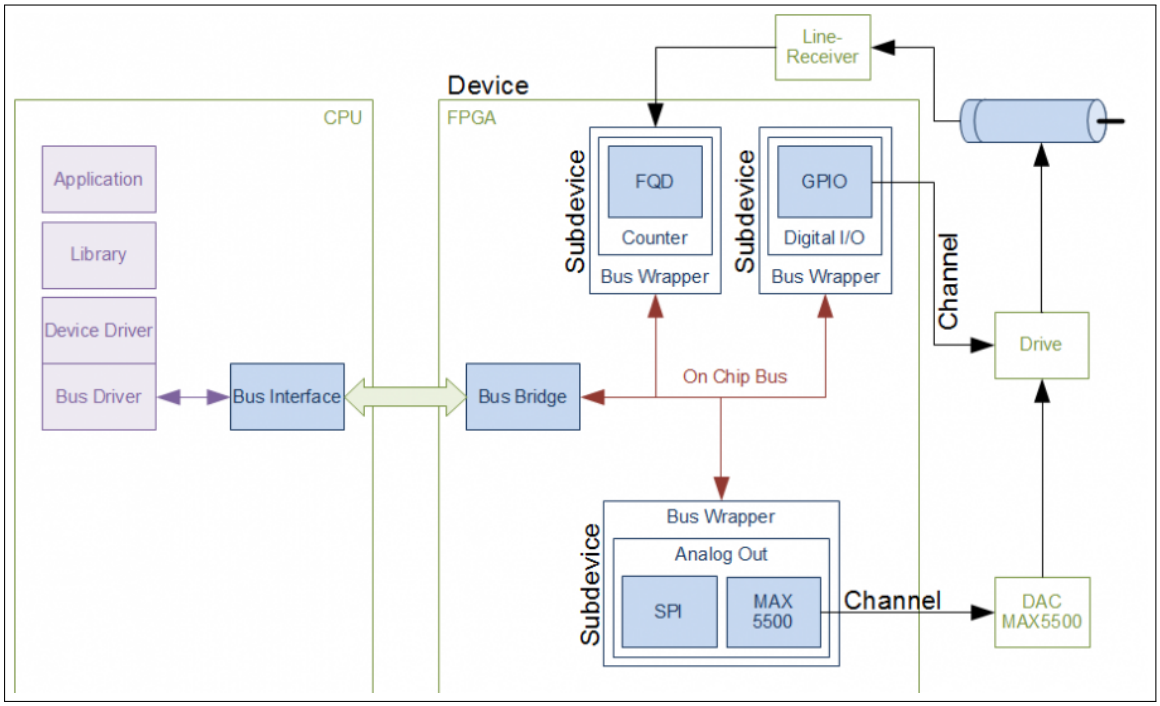

Abbildung 2.3: Architekturbeispiel mit Flink (https://flink-project.ch/_detail /systemarchitectureexample.png?id=start)

Die Abbildung 2.3 zeigt die Architektur mit Linux als Betriebssystem auf der CPU-Seite. Die Kommunikation mit dem Flink Modul wird durch einen Bus ermöglicht. Es sind verschiedene Bussysteme möglich: AXI, SPI, PCI usw. Auf der FPGA-Seite ist ein Architektur Beispiel abgebildet, das zeigt, wie ein Motor betrieben wird.

2.4.1 FPGA

Die FPGA-Programmierung wurde in VHDL vorgenommen und ist unterteilt in verschiedene Subdevices, welche Funktionen nach aussen ausführen. Neben den normalen Subdevices gibt es ein spezielles Subdevice «info». Dieses stellt nur Informationen über das Flink Device für die CPU zur Verfügung. Weitere Subdevices mit Funktionen gegen aussen sind:

- **AnalogIn**: Stellt analoge Eingänge zur Verfügung (ADC).
- **AnalogOut**: Stellt analoge Ausgänge zur Verfügung (DAC).
- **DigitalIO**: Stellt digitale Ein- und Ausgänge zur Verfügung (GPIO).
- **Counter**: Zählereingang (um z.B. Motoren-Encoder anzuschliessen).
- **PWM**: Generiert PWM-Signale.
- **PPWA**: Messeinrichtung für Pulsweiten und Perioden.
- **UART**: Stellt einen UART-Bus an den jeweiligen Pins zur Verfügung.
- **Watchdog**: Stellt einen Ausgang für einen Systemneustart zur Verfügung.
- **Sensor**: Beinhaltet Anschlüsse für diverse Sensoren.

2.4.2 Kernel

Der Kernel selbst kann wieder in zwei Komponenten aufgeteilt werden, dem Kern-Modul und den Bus-Modulen.

Kern-Modul

Das Kern Modul ist zuständig für das Erkennen der Flink Subdevices sowie für die Kommunikation mit dem Userspace. Dabei kommuniziert dieses über das Bus-Modul mit dem FPGA und kann so auch die Subdevices modifizieren. Für die Kommunikation mit dem Userspace erstellt das Modul ein Device im Ordner «/dev/».

Bus-Module

Die Bus-Module sind die Verbindungsstücke zwischen dem Kern-Modul und dem FPGA. In diesem ist die Bus-Logik programmiert, um Daten an das FPGA zu senden und auszulesen. Diese Module gibt es aktuell in diversen Varianten. Zum Beispiel in: PCI, SPI, EIM und LPB.

2.4.3 Userspace Bibliothek

Diese Bibliothek stellt diverse Funktionen zur Verfügung, um einfacher mit dem Kernel-Modul interagieren zu können. Dieses kann so direkt mit eigenen C Projekten verwendet werden. Des Weiteren werden gleich noch diverse Shell Befehle zur Verfügung gestellt, um mittels der Konsole Ein- und Ausgänge auf dem FPGA zu konfigurieren.

2.5 C/C++-Python Interface

Es gibt verschiedene Interfaces für eine Einbindung von C/C++ geteilten Bibliotheken in Python. Für das Interface wurden zwei verschiedene Entwicklungs-Tools angeschaut, die nachfolgend beschrieben werden.

Durch das Verwenden von solchen Interfaces können so erprobte, sichere und schnelle Algorithmen, welche sich in geteilten Bibliotheken befinden, in diversen anderen Sprachen benützt werden. Ohne ein solches Interface müssten diese Algorithmen, für die gewünschten Sprachen, erneut implementiert und getestet werden. Dies würde einen erheblichen Mehraufwand bedeuten.

2.5.1 Swig

Swig[11] ist ein sehr umfangreiches Tool. Es unterstützt diverse Programmiersprachen wie JavaScript, Java, Perl, Python, PHP, usw. So kann eine geteilte Bibliothek, welche in C oder C++ geschrieben wurde, in den unterstützten Sprachen verwendet werden.

Dazu generiert SWIG mithilfe des Headers von der Bibliothek einen zusätzlichen C/C++-Code und einen Python-Wrapper. Dieser C/C++-Code ist auch die Schwachstelle von Swig. Der generierte C/C++-Code muss nämlich mit dem Quellcode der geteilten Bibliothek zusammen gelinked werden. Dafür muss der Entwickler den Quellcode der Bibliothek besitzen. Wenn die geteilte Bibliothek so erstellt wurde, kann anschliessend mittels des Python-Wrappers auf die geteilte Bibliothek zugegriffen werden.

Da Swig trotz diversen Versuchen nie erfolgreich ausgeführt werden konnte, wurde diese Methode verworfen und auf die nachfolgende Methode umgestiegen.

2.5.2 ctypes

ctypes[12] ist eine Bibliothek innerhalb von Python, welche es ermöglicht, C geteilte Bibliotheken zu verwenden. Jedoch ist es mit ctypes nicht möglich C++ geteilte Bibliotheken direkt zu benützen. Ctypes ist ein simples und einfach zu handhabendes Tool. Das führt jedoch zu Mehraufwand für den Programmierer. Es hat jedoch auch einen grossen Vorteil im Gegensatz zu Swig.

Der grösste Vorteil ist, dass man keinen Quellcode von der geteilten Bibliothek benötigt, sondern nur das Headerfile. So kann man die Funktionen in Python einfach aufrufen, welche innerhalb der geteilten Bibliothek zur Verfügung gestellt werden.

Ein Nachteil ist, dass der Wrapper, welche die geteilten Bibliothek umschliesst, von dem Python Entwickler selbst erstellt werden muss. Dies nimmt sehr viel Zeit in Anspruch, vor allem, um den Wrapper zu entwickeln und zu testen.

Technische Umsetzung

3.1 Petalinux Project

3.1.1 Erstellen des Projektes

Beim Erstellen des Projektes mit dem PetaLinux wurde auf die Version 2022.1[6] zurückgegriffen. Für die aktuellste PetaLinux Version existiert das .bsp File[8] für das MicroZed Bord noch nicht.

Anschliessend an die Installation von PetaLinux konnte mit dem .bsp Files das Projekt initialisiert werden. Diese Installation legte einen Ordner an, in welchem alles, für den Bau des Linux Notwendige, vorhanden ist. In diesem Ordner können die entsprechenden manuellen Konfigurationen direkt in den Files vorgenommen werden. PetaLinux bringt eine «angenehmere» Option für diese Konfigurationen mit.

```
petalinux-config -c {option}
```

Code-Snippet 3.1: Bash PetaLinux Konfiguration

Dieses Kommando öffnet eine «Grafische Oberfläche» innerhalb der Bash, welche einfacher zu bedienen ist, als die Änderungen direkt in den Files vorzunehmen.

Der Parameter {option} hat drei verschiedene Optionen:

- **rootfs**: Für die Konfiguration des Root Filesystems. In diesem können z.B. Packages hinzugefügt werden, (Python Umgebung).
- **kernel**: Für die Konfiguration des Kernels. Hier kann z.B. eingestellt werden, wie die Unterbrechbarkeit (Preemption) des Kernels sein soll.
- **keine Option**: Mit dieser Möglichkeit wird der Parameter -c komplett weggelassen. Diese Option ermöglicht es, das PetaLinux-Projekt zu konfigurieren.

3.1.2 Preempt-RT-Patch

Da der Linux-Kernel innerhalb dieses Projektes nicht den Fully Preemptible Kernel standartmässig unterstützt, muss dieser mittels eines Patches manuell hinzugefügt werden. Um dies zu bewerkstelligen, wurde der Patch, welcher zu der Kernelversion 5.15.19 gehört, verwendet. Dieser kann von der Kernelseite «https://cdn.kernel.org/pub/linux/kernel/projects/rt/5.15/older/» heruntergeladen werden. Dabei wurde in dieser Arbeit der Patch mit der Version «patch-5.15.19-rt29.patch.gz» verwendet. 5.15.19 steht hierbei für die unterstützte Kernelversion.

Das heruntergeladene File muss anschliessend entpackt und in das Verzeichnis «project-spec\meta-user\recipes-kernel\linux\linux-xlnx» kopiert werden. Anschliessend, damit dieser Patch beim Bauen des Linux auch berücksichtigt wird, muss das Rezept «project-spec\meta-user\recipes-kernel\linux\linux-xlnx_%.bbappend» angepasst werden. Dabei reicht es aus, wenn folgende Zeile in das File eingefügt wird:

```
SRC_URI += "file://patch-5.15.19-rt29.patch"
```

Code-Snippet 3.2: Hinzufügen des Preempt-RT-Patch

Anschliessend muss dieses Modell noch aktiviert werden mittels der Einstellung «Sheduler», welches zu finden ist unter:

```
petalinux-config -c kernel
```

Code-Snippet 3.3: Aktivierung des Preempt-RT-Patch

3.1.3 Bauen des Linux

Um das Linux mit allen Anpassungen und Änderungen zu erstellen, wird eine vorkonfigurierte SD-Karte benötigt, siehe Kapitel 3.2. Der Bau des Linux erfolgt mit dem PetaLinux Kommando:

```
petalinux-build -c avnet-image-minimal
```

Code-Snippet 3.4: Bash PetaLinux Bau Prozess

Anstatt die Option «avnet-image-minimal» zu verwenden, kann alternativ die Option «avnet-image-full» verwendet werden. Diese baut zusätzlich diverse Packages in das root Filesystem mit ein. So müssen diese Packages nicht von Hand aktiviert werden. Da in diesem Projekt das System so klein wie möglich gehalten werden sollte, wird die «minimale» Version verwendet.

Um schlussendlich ein bootfähiges Image zu bekommen, muss das Boot File noch gepackt werden. Dieses muss mindestens den FSBL, den u-boot und das fpga.bin File beinhalten.

Um die ganzen Schritte nicht jedes Mal einzeln ausführen zu müssen, wurde ein Bash Skript erstellt. In diesem Skript wird zuerst das Linux gebildet und anschliessend die Files auf die jeweilige Partition der SD-Karte kopiert. Das Skript befindet sich im Anhang auf der Seite 46.

3.2 SD-Karte

Das Microzed besitzt einen microSD Karten Steckplatz, von welchem man Linux starten kann. Damit Linux jedoch von der SD-Karte booten kann, muss die Karte speziell vorbereitet werden.

Standardmässig ist auf der SD-Karte nur eine Partition vorhanden, welche den ganzen Speicherplatz von der SD-Karte einnimmt. Diese Partition ist ebenfalls als nicht bootfähige Partition gekennzeichnet. Somit muss die SD-Karte komplett neu partitioniert werden.

Die SD-Karte ist nun auf zwei Partitionen aufzuteilen, welche folgendermassen eingerichtet sind.
Partition 1 (boot) hat eine Grösse von 500MB und das «bootable flag» für diese Partition ist gesetzt. Diese Partition wird anschliessend formatiert mit Filesystem «FAT32».
Partition 2 (root) nimmt den Rest der SD-Karte ein. Diese Partition wird anschliessend mit dem Filesystem «ext4» formatiert.

3.3 Erstellen des FPGA Files

Als Grundlage diente das Vivado-Projekt für das Systemtechnikprojekt[9] 2021/2022 des Bachelorstudienganges Systemtechnik an der Fachhochschule OST Campus Buchs. Dieses wurde für das MicroZed entwickelt, welches mit Java Depp als Bare Metal Anwendung betrieben wurde.
In dieser Arbeit wird das MicroZed Bord mit Linux als Betriebssystem betrieben. Linux erwartet beim Starten einen konfigurierten Interrupt vom FPGA (PL-PS Interrupt). Interrupts sind im Flink Projekt noch nicht implementiert. Da mehrere Versuche fehlgeschlagen haben, das Interrupt im Linux Kernel zu deaktivieren, ist ein «Dummy» Interrupt mittels Vivado in das Flink Projekt implementiert worden. So erkennt Linux beim Aufstarten den FPGA Interrupt und kann mit dem Bootprozess fortfahren.
Eine Abbildung von dem Dummy Interrupt befindet sich auf der Seite 48.

3.4 Einbindung des Flink Kernels & Userspace Bibliothek

3.4.1 Einbindung Flink Modul

Um ein eigenes Kernel Modul dem PetaLinux Projekt hinzuzufügen, ist der einfachste Weg, die Erstellung eines Beispielprojekts. Dazu liefert PetaLinux den Befehl:

```
1 petalinux-create -t modules --name <modulname> --enable
```

Code-Snippet 3.5: Erstellen eines Beispiel Kernel Modules

Dadurch wird unter «project-spec \meta-user \recipes-modules» ein Ordner angelegt, in dem das Projekt abgelegt ist. Dieses kann nun einfach modifiziert werden, indem das Rezept angepasst wird (das .bb file). Das angepasste Rezept für dieses Modul befindet sich im Anhang auf der Seite 40. Im Ordner «Files», welcher am gleichen Ort abgelegt ist wie das Rezept File, wird nun das Gitrepo[10] für das Flink-Kernel-Modul eingefügt.

AXI-Kommunikations-Modul

In dem Git Repository für den Flink Kernel sind bereits diverse Bussysteme für die Kommunikation zwischen dem Linux und dem FPGA implementiert. Das Kommunikations-Modul für den AXI-Bus fehlt jedoch noch. Deshalb wurde das Projekt, welches zuvor von dem Git Repository kopiert wurde, um ein C-Code-File erweitert. Dieses reserviert und mappt das physikalische Memory von Flink in das Linux. Im Weiteren stellt es dem zentralen Modul diverse Funktionen zur Verfügung, um auf dieses Memory zugreifen zu können. Dieses File befindet sich im Anhang auf der Seite 41.

Damit dieses C-Code-File bei der Kompilierung auch berücksichtigt wird, muss noch das Makefile angepasst werden. Dazu reicht es aus die Zeile, `obj-m += flink_axi.o`, hinzuzufügen.

! Anmerkung:

Das File, in welchem der Code von dem Kommunikationsmodul steht, muss unbedingt gleich benannt sein wie der Wert, welchem dem Parameter «obj-m» mitgegeben wird. Einzig die Endung ist beim File «.c»

3.4.2 Einbindung Flink Userspace Bibliothek

Um ein Userspace Applikations Projekt oder eine Bibliothek zu erstellen, liefert PetaLinux ebenfalls ein Kommando, welches ähnlich ist, wie jenes zum Erstellen eines Modules. Dieses legt einen neuen Ordner mit dem Projekt unter «project-spec\meta-user\recipes-apps» an.
Da der Code von der Userspace Bibliothek nicht abgeändert werden muss, kann das Rezept den Code direkt von dem Git Repository holen. So entfällt der ganze Ordner «files», in welchem normalerweise der ganze Code abgelegt ist. Jedoch muss das

Rezept File stärker angepasst werden. So müssen die Funktionen do_configure(), do_compile(), do_install() hinzugefügt oder abgeändert werden, um den Bauprozess zu modifizieren. Diese Funktionen werden von PetaLinux aufgerufen, wenn das Projekt gebaut wird. So kann PetaLinux genau mitgeteilt werden, wie das Projekt konfiguriert, gebaut und wo die ausführbaren Dateien hingelegt werden sollen. Das Rezept befindet sich im Anhang auf der Seite 42.

3.4.3 Python-Wrapper

Die Systemtechnikprojekte werden mit der Programmiersprache Python durchgeführt. Deshalb muss der C-Code von Python erreichbar sein.

Eine Möglichkeit wäre es gewesen, die komplette, oben genannte Bibliothek, in Python umzuschreiben, um Flink auch innerhalb von Python zu benutzen. Jedoch würde so derselbe Code doppelt existieren, was wiederum Mehraufwand und Fehleranfälligkeit bei späteren Änderungen bedeuten würde.

Aus diesem Grund wurde ein Python Wrapper erstellt, welcher die in C geschriebene Bibliothek umfasst und sie in Python einfach zugänglich macht. Diese Bibliothek wurde mithilfe der Python-Bibliothek «ctypes» erstellt. Ausschnitte aus diesem Wrapper befinden sich im Anhang auf den Seiten 44 und 45. Um nicht mehrere Seiten mit dem Code zu füllen, wurden bei «...» diverse Zeilen herausgelöscht.

3.5 Motorentreiber

In den Systemtechnikprojekten werden Roboter entwickelt. Dabei wird die Fortbewegung der Roboter mit Motoren realisiert. Jedoch ist die Ansteuerung der Motoren mit Geschwindigkeitsregelung sehr anspruchsvoll. Daher wird der Geschwindigkeitsregler für die Motoren in die Entwicklungsumgebung integriert. Als Vorlage hierfür dient der Regler der vorhergehenden Systemtechnikprojekte.

3.5.1 C-Bibliothek

Der Regler[13], welcher für die bisherigen Systemtechnikprojekten in Java realisiert wurde, ist nun in eine C-Bibliothek migriert. Der Motorenregler muss periodisch in kurzen Zeitabständen aufgerufen werden. Mit der Programmiersprache Python würde die schnelle Regelung viel Ressourcen, z.b. CPU-Zeit, in Anspruch nehmen. Deshalb wurde dafür die performantere Sprache C gewählt. Ein Teil des Codes der Bibliothek befindet sich im Anhang auf der Seite 49.

3.5.2 Python-Wrapper

Da für das Systemtechnikprojekt Python verwendet wird, muss die C-Bibliothek aus Python erreichbar sein. Dafür wurde ein Wrapper in Python entwickelt. Dieser soll den Zugriff auf die Bibliothek vereinfachen. Dies wurde mittels der Python Bibliothek «ctypes» realisiert.

3.5.3 Motorentest

Ein manueller Test, der bisher durchgeführt wurde, ist nicht mehr möglich. Daher sind die Funktionen des Wrappers und der Bibliothek mittels eines kleinen Skriptes getestet worden. Das Skript befindet sich in Anhang auf der Seite 51. Dieses Skript kann den Studenten des Systemtechnikprojektes als Vorlage dienen, um einen Motor anzusteuern.

3.6 SDK Python

Da Python eine interpretierte Sprache ist, muss der Quellcode nicht erst kompiliert und gelinkt werden, bevor das Programm auf das Ziel-System übertragen wird. Auch sind die meisten Python-Bibliotheken systemübergreifend gleich. Somit kann ein Python-Skript auf einer Windows- und einer Linux- Plattform ohne Anpassungen ausgeführt werden. Jedoch gibt es ein paar Ausnahmen wie z.B. die Bibliothek für das Betriebssystem `import os`. Dies betrifft jedoch nicht die ganze Bibliothek, sondern nur einige Funktionen, welche zwischen Linux und Windows nicht kompatibel sind.

Ebenso benötigen wir spezielle Bibliotheken, z. B. für das Flink-Interface. Um diese der Entwicklungsumgebung zur Verfügung zu stellen, können diese Bibliotheken in einen Ordner auf den Entwicklungscomputer kopiert werden. Jedoch ist zu beachten, dass die Ordnerstruktur des Zielsystems beibehalten wird, in welchem die Python-Bibliotheken abgelegt sind. Der direkte Austausch von Textfiles über einen USB-Stick eines Linux Systems zu einem Windows-System funktioniert nicht (Sonderzeichen können Probleme verursachen). Um diese Fehler zu vermeiden, muss ein Tool wie GIT verwendet werden, um Dateien zwischen den Systemen auszutauschen.

Nun geht es um die Einrichtung der Entwicklungsumgebung. Dabei wird «Eclipse» mit der Erweiterung «PyDev» verwendet. Es spielt keine Rolle, welche Version von Eclipse installiert wird. Für den Test wurde «Eclipse for Java Developers» verwendet. Nach der Installation von Eclipse muss noch die Erweiterung «PyDev» installiert werden. PyDev wird mittels «Marketplace» von Eclipse installiert. Dieser ist unter dem Reiter «Help» innerhalb von Eclipse zu finden. Nach der Installation von «PyDev» können nun Python Projekte erstellt werden. Als nächstes muss der Order, in welchem die speziellen Bibliotheken kopiert wurden, noch zu dem Projekt hinzugefügt werden. Dies ist zu erreichen mittels Rechtsklick auf das Projekt und dann «Properties». Anschliessend wird das Untermenu «PyDev – PYTHONPATH» ausgewählt, in welchem noch der Reiter «External Libraries» ausgewählt werden muss. Als letzter Schritt muss man den Ordner, in welchem die Bibliotheken liegen, mittels des Knopfes «Add source folder» hinzufügen.

Somit ist die Entwicklungsumgebung eingerichtet und es kann mit der Programmierung gestartet werden. Um jedoch das Programm starten zu können, müssen die Python-File(s) noch auf das Zielsystem übertragen werden. Dazu wird das Kommando `scp` verwenden. Um dies zu vereinfachen, wurde ein kurzes Skript für Windows geschrieben, welches im Anhang auf der Seite 47 zu finden ist.

3.7 Flink-Lib und Threadsafety

Während der Analyse der Kernelfunktionen (nur Zugriff aus dem Userspace) und der Userspace-Bibliothek wurden mit dem gleichzeitigen Zugriff von mehreren Threads folgende Aspekte analysiert:

- **Gemeinsame Daten**: Analyse, ob es bei Daten oder Datenstrukturen zu Inkonsistenzen kommt, wenn mehrere Threads gleichzeitig darauf zugreifen.
- **Globale Variablen**: Kontrolle, ob globale Variablen im Code verwendet werden. Es wurde geprüft, ob diese ein Problem aufweisen könnten.
- **Bibliotheken**: Kontrolle, ob die verwendeten Bibliotheken oder Bibliotheksfunktionen für multi-threading ausgelegt sind. Wenn nicht, wurde geprüft, ob diese ein Problem verursachen können.
- **Deadlocks**: Dieses Problem kann ausgeschlossen werden, da keine Synchronisierungs Mechanismen implementiert wurden wie Mutexe oder Semaphoren.

Jedoch muss hierzu noch erwähnt werden, dass nur eine theoretische Analyse durchgeführt wurde. Praktische Tests gehören immer dazu! Diese sollten jedoch noch ausgearbeitet und durchgeführt werden, so dass sichergestellt werden kann, dass der Zugriff durch Multithreading möglich ist.

3.7.1 Kernel

Anschliessend sind die Punkte aufgeführt, welche bei der Analyse der Funktionen als Probleme identifiziert wurden:

- **Funktion: Flink_open**: Das mehrmalige Öffnen des Devices, welches für die Kommunikation zwischen Kernel und Userspace zuständig ist, ist nicht abgesichert worden. Zusätzlich werden beim gleichzeitigen Öffnen das gemeinsame Datenfeld `f->private_data` beschrieben. Dies kann zu einem ungültigen Zeiger führen.
- **Datenfeld: current_subdevice**: Um Daten vom FPGA einzulesen, können die Funktionen flink_read und flink_write verwendet werden. Damit diese Funktionen wissen, welches Subdevice sie verwenden sollen, wird dieses innerhalb der Funktion flink_ioctl ausgewählt. Hier taucht ein Problem auf, da die Auswahl in dem gemeinsamen Datenfeld current_subdevice abgespeichert wird. Ein Beispiel dazu: Der erste Thread wählt mittels flink_ioctl das Subdevice «eins», um anschliessend Daten mittels flink_read einzulesen. Wenn nun ein weiterer Thread das Subdevice «zwei» auswählt, gleich nachdem der erste Thread das Subdevice eins ausgewählt hat, wird der erste Thread beim Einlesen der Daten das falsche Subdevice «zwei» verwenden.

3.7.2 Userspace

Bei der Analyse von der Userspace Bibliothek wurden keine Probleme gefunden, welche durch Multithreading verursacht werden könnten.

Allgemein sollte vermieden werden, gleichzeitig auf denselben Channel in Subdevice zu schreiben. Dies, da der Channel eine Hardwarekomponente darstellt und es so zu Datenfehler kommen kann.

3.7.3 Schlussfolgerung

Beim Zusammenführen der Analyse der Kernel und der Userpace Bibliothek kann man folgende Schlüsse ziehen:

- Das erste Problem `thread_open` vom Kernel kann umgangen werden, wenn ein File innerhalb einer Applikation nur einmal geöffnet wird.
- Das zweite Problem mit dem Datenfeld «current_subdevice» wurde bereits durch eine zweite Threadsichere Implementierung behoben, welche von der Userspace Bibliothek verwendet wird.

Abschliessend kann gesagt werden, dass die Bibliotheken problemlos mit Multithreading verwendet werden können.

Ausblick

4.1 Interrupt PL zu PS

Der aktuelle Entwicklungsstand von Flink hat keine Interrupt Fähigkeit. Das bedeutet, um das Drücken eines Tasters zu registrieren, muss aktuell der Eingang, an welchem der Taster angeschlossen ist, dauernd aktiv überprüft werden. Dies ist das sogenannte Polling. Polling ist im Vergleich zu einem Interrupt sehr rechenintensiv. Dies, da zyklisch ein Code ausgeführt werden muss, nur um zu kontrollieren, ob sich am Signaleingang etwas geändert hat.

Daher ist es vorteilhaft, wenn Flink die Interrupt Fähigkeit hinzugefügt wird.

4.2 Anwendungsmöglichkeiten

Wie bereits in der Ausgangslage auf der Seite I erläutert, diente diese Arbeit dem Robotik Projekt des ersten und zweiten Semesters vom Lehrgang Systemtechnik als Grundlage. So sind die Studierenden nun in der Lage diverse Sensoren und Aktoren anzuschliessen. Dabei ist der ganze Signalfluss so weit abstrahiert, dass die Studierenden sich nicht mit Flink beschäftigen müssen. Sie können einfach die Python Wrapper verwenden, welche vorbereitet wurden.

Jedoch umschliesst diese Arbeit nicht alle Funktionalitäten, welche die Studierenden auf dem alten System zur Verfügung hatten. Daher müssen noch weitere Python Wrapper und/oder Python Bibliotheken hinzugefügt werden.

4.3 Schlussfolgerungen

Zusammenfassend kann gesagt werden, dass diese Vertiefungsarbeit eine gute Grundlage bietet für die Umstellung von Deep Java auf Python unter Linux. So können die nächsten Systemtechnikprojekte einfach auf Linux und Python umstellen und vom grossen Linux Funktionsumfang profitieren.

Dadurch, dass nun unter Linux entwickelt werden kann, können Bibliotheken hinzugefügt werden, um speziellere Hardware zu verwenden, wie zum Beispiel eine USB-Bilderkennungskamera. Zusätzlich kann man auch von bereits entwickelten Algorithmen profitieren, da die Linux- und Python Communities sehr gross sind.

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.

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, 17. Juli 2023

Glossar

U-Boot: Ein Bootloader für eingebettete Systeme.

Bare Metal: Eine Programmiermethode, in welcher das Programm direkt auf den Controller geladen wird, ohne das Betriebssystem als Zwischenschicht.

Python: Eine interpretierte Programmiersprache, welche weit verbreitet ist und sich durch ihre Lesbarkeit und Einfachheit auszeichnet.

Linux: Ein Betriebssystemkern, welcher Open-Source ist. Dabei gibt es diverse Distributionen wie Ubuntu und Mint. Wird auf PCs, Server, Embedded Systems, Mobilgeräten und vielen weiteren Geräten eingesetzt.

MPC555: Ein Microcontroller, welcher in früherer OST-Systemtechnikprojekten eingesetzt wurde.

Jumpers: Ein modularer, elektrischer Verbinder, um Pins auf elektrischen Printplatten zu verbinden.

Yocto: Ein Framework, um das Linux-Betriebssystem anzupassen und zu erstellen. Meistens verwendet für Embedded Systeme.

Kernel: Der Kern eines Betriebssystems. Verwaltet die Ressourcen, Prozesse und Speicher eines Systems.

Gitrepo: Ein online Speicherort, um Dateien zu speichern und zu versionieren.

Akronyme

ADC: Analog-to-Digital Converter

ARM: Advanced RISC Machines (Prozessorarchitektur)

AXI-Bus: Advanced eXtensible Interface Bus

Bash: Bourne Again SHell

CLI: Command Line Interface

CPU: Central Processing Unit

DAC: Digital-to-Analog Converter

DDR3: Double Data Rate 3

EIM: External Interface Module

ext4: Fourth Extended Filesystem

FAT32: File Allocation Table 32

FPGA: Field Programmable Gate Array

FSBL: First Stage Boot Loader

GB: Gigabyte

GPIO: General Purpose Input/Output

LPB: Low Pin Count Bus

MB: Megabyte

PCI: Peripheral Component Interconnect

PL: Programmable Logic

PS: Processing System

SCP: Secure Copy Protocol

SDK: Software Development Kit

SoC: System on a Chip

SPI: Serial Peripheral Interface

UART: Universal Asynchronous Receiver Transmitter

VHDL: Very High-Speed Integrated Circuit Hardware Description Language

XSCT: Xilinx Software Command-Line Tool

Abbildungsverzeichnis

Codeverzeichnis

Literaturverzeichnis

[1] Avnet Boards. Microzed. https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/microzed/. (abgerufen 28.04.2023)

[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 28.04.2023)

[4] Xilinx. Petalinux Tools. https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html (abgerufen 29.04.2023)

[5] Xilinx. PetaLinux Tools Documentation Reference Guide. Version 2021.1

[6] Xilinx. PetaLinux Installer. https://www.xilinx.com/support/download/index.html/content/design-tools/2022-1.html (abgerufen 30.04.2023)

[7] Graf, Urs. Java Deep. https://www.deepjava.org/start (abgerufen 29.04.2023)

[8] Avnet. MicroZed. https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/microzed/ (abgerufen 30.04.2023)

[9] Graf, Urs. MicroZed. Ost Wiki. https://wiki.bu.ost.ch/infoportal/embedded_systems/zynq7000/microzed (abgerufen 30.04.23)

[10] Graf, Urs. flinklinux. https://github.com/flink-project/flinklinux (abgerufen 30.04.2023)

[11] Beazly, David M. swig. https://www.swig.org/ (abgerufen 16.06.2023)

[12] Python, Standart Lib. ctypes. https://docs.python.org/3/library/ctypes.html (abgerufen 16.06.2023)

[13] Graf, Urs. SpeedController4DCMotor. https://github.com/deepjava/runtime-library/blob/master/src/org/deepjava/runtime/zynq7000/sysp/SpeedController4DCMotor.j (abgerufen 13.07.2023)

Anhang

A Flink Modul Rezept

```
SUMMARY = "Recipe for  build an external flinkkernel Linux kernel module"
SECTION = "PETALINUX/modules"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=12f884d2ae1ff87c09e5b7ccc2c4ca7e"

inherit module

INHIBIT_PACKAGE_STRIP = "1"

SRC_URI = " file://./ "

S = "${WORKDIR}"

# The inherit of module.bbclass will automatically name module packages with
# "kernel-module-" prefix as required by the oe-core build environment.
```

Code-Snippet 4.1: Flink Modul Rezept

B Flink AXI Code

[TODO: Code hier noch einfügen]

```
1   hier noch der code
```

Code-Snippet 4.2: Flink AXI Kommunikations Code

C Flink Userspace Rezept

```
    #
    # This file is the flinklib recipe.
    #

    SUMMARY = "Simple flinklib application"
    SECTION = "libs"
    LICENSE = "MIT"
    LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835
ade698e0bcf8506ecda2f7b4f302"

    SRC_URI = "git://github.com/flink-project/flinklib.git;protocol=
https"
    SRC_URI += "file://Python3/Flink.py"
    SRCREV = "master"

    S = "${WORKDIR}"

    PROVIDES = "flink"

    inherit pkgconfig cmake

    do_configure() {
        git config --global http.sslverify false
        cd ${S}/git
        git submodule init
        git submodule update
        cmake ${S}/git
    }

    do_compile() {
        cd ${S}/git
        oe_runmake
    }

    do_install() {
        # install binary shared object and register it in petalinux
for other projects
        install -d ${D}${libdir}
        cp ${S}/git/lib/libflink.so.1.0.* .
        oe_libinstall -so libflink ${D}${libdir}

        # install headers
        install -d -m 0655 ${D}${includedir}/FLINK
            install -m 0644 ${S}/git/include/*.h ${D}${includedir}/
FLINK

        # install packages for commandline
        install -d ${D}${bindir}
        install -m 0755 ${D}/../git/utils/flinkanaloginput ${D}${
bindir}
        install -m 0755 ${D}/../git/utils/flinkanalogoutput ${D}${
bindir}
        install -m 0755 ${D}/../git/utils/flinkcounter ${D}${bindir}
        install -m 0755 ${D}/../git/utils/flinkdio ${D}${bindir}
```

```
49          install -m 0755 ${D}/../git/utils/flinkinfo ${D}${bindir}
50          install -m 0755 ${D}/../git/utils/flinkppwa ${D}${bindir}
51          install -m 0755 ${D}/../git/utils/flinkpwm ${D}${bindir}
52          install -m 0755 ${D}/../git/utils/flinkwd ${D}${bindir}
53          install -m 0755 ${D}/../git/utils/lsflink ${D}${bindir}
54
55      # replace invalied RPATH with a arbitrary one (if i replace it
    with /usr/lib then occurs
56      # a error: flinklib: /usr/bin/flinkanaloginput contains probably
    -redundant RPATH /usr/lib [useless-rpaths])
57      # but if you leave it how it is a error occour: flinklib-1.0-r0
    do_package_qa: QA Issue: package flinklib contains bad RPATH
58          chrpath -r ${bindir} ${D}${bindir}/flinkanaloginput
59          chrpath -r ${bindir} ${D}${bindir}/flinkanalogoutput
60          chrpath -r ${bindir} ${D}${bindir}/flinkcounter
61          chrpath -r ${bindir} ${D}${bindir}/flinkdio
62          chrpath -r ${bindir} ${D}${bindir}/flinkinfo
63          chrpath -r ${bindir} ${D}${bindir}/flinkppwa
64          chrpath -r ${bindir} ${D}${bindir}/flinkpwm
65          chrpath -r ${bindir} ${D}${bindir}/flinkwd
66          chrpath -r ${bindir} ${D}${bindir}/lsflink
67
68          #install python wrapper for libflink.so
69          # install python helpers
70          install -d ${D}${libdir}/python3.9/Systemtechnik
71          install -m 0644 ${D}/../Python3/Flink.py ${D}${libdir}/
    python3.9/Systemtechnik
72      }
73
74      FILES_${PN} = "${libdir}/*.so.* ${includedir}/flink/*"
75      FILES_${PN}-dev = "${libdir}/*.so"
76      FILES:${PN} += "${libdir}/python3.9/Systemtechnik/*.py"
77
78
```

Code-Snippet 4.3: Flink Userspace Rezept

D C-Python Wrapper

```
def __init__(self, lib_path="/usr/lib/libflink.so.1.0.2.8"):

    self.fileOpen = False
    self.initialized = False

    # point to a class variable
    self._private_static_openedFiles = Flink._private_static_openedFiles

    # load the shared lib
    self.lib_path = lib_path
    self.lib = ctypes.CDLL(self.lib_path)

    # ================================ Base operations ================================
    self.lib.flink_open.argtypes = [ctypes.c_char_p]
    self.lib.flink_open.restype = ctypes.c_void_p

    self.lib.flink_close.argtypes = [ctypes.c_void_p]
    self.lib.flink_close.restype = ctypes.c_int

    # ================================ General ================================

    ...

    self.lib.flink_subdevice_get_unique_id.argtypes = [ctypes.c_void_p]
    self.lib.flink_subdevice_get_unique_id.restype = ctypes.c_uint32

    ...

    # ================================ Info ================================
    self.lib.flink_info_get_description.argtypes = [ctypes.c_void_p, ctypes.c_char_p]
    self.lib.flink_info_get_description.restype = ctypes.c_int

    ...

    self.initialized = True
```

Code-Snippet 4.4: C-Python Wrapper Init Section

```
def flink_info_get_description(self, flink_subdevice):
    """
    Reads the description field of an info subdevice

    :param flink_subdevice: Subdevice
    :type flink_subdevice: pointer to struct flink_subdev
    :return: The description is a string representing the
   description field of the info subdevice.
    :rtype: String

    :raises Exception: If initialization isn't complete
    :raises FlinkException: Appears If an error occurs in c-lib
    :raises FileException: Appears if the flink file has not been
   opened for the time being
    """
    self._private_check_init_file()
    p_string = ctypes.create_string_buffer(Flink.INFO_DESC_SIZE)
    error = self.lib.flink_info_get_description(flink_subdevice,
   p_string)
    if error < 0:
        raise FlinkException("Faild to read info from Info-subdevice
   ", error, flink_subdevice)
    return p_string.value.decode('utf-8')
```

Code-Snippet 4.5: C-Python Wrapper Funktion

E Script um Linux zu bauen

```
#!/bin/bash

Red='\033[1;31m'
Color_Off='\033[0m'

cd /home/patrick/Documents/vt1/mz7010_som_base_2022_1/
echo "change to dir: $(pwd)"

source /home/patrick/Documents/vt1/petalinux/settings.sh

# claiming superuser rights for later
sudo echo ""

echo "Create linux"
petalinux-build -c avnet-image-minimal || { echo -e "${Red}Errors in Build${Color_Off}" ; exit 1; }

echo ""
echo "Create BOOT.BIN"
cd ./images/linux/
echo "change to dir: $(pwd)"
petalinux-package --boot --fsbl zynq_fsbl.elf --fpga ../../../Vivado-Project/MicrozedFlink2.runs/impl_1/flink_wrapper.bit --u-boot --force

echo ""
echo "copy files in boot dir if sd card is plugged in"
DIR="/media/patrick/boot"
if [ -d "$DIR" ]; then
  rm -f $DIR/BOOT.BIN
  rm -f $DIR/image.ub
  rm -f $DIR/boot.scr
  cp BOOT.BIN $DIR
  cp image.ub $DIR
  cp boot.scr $DIR
  echo "copied files: BOOT.BIN image.ub boot.scr if no errors occoured"
else
  echo -e "${Red}please insert sd card (formatted) and ensure it is mounted in folder: $DIR ${Color_Off}"
fi

echo ""
echo "extract rootfs and copy it into rootfs partition"
DIR="/media/patrick/root"
if [ -d "$DIR" ]; then
  sudo rm -rf $DIR/*
  sudo tar -xf rootfs.tar.gz -C $DIR
else
  echo -e "${Red}please insert sd card (formatted) and ensure it is mounted in folder: $DIR ${Color_Off}"
fi
```

Code-Snippet 4.6: Linux Build

F Kopieren der Python-Files

```
@echo off

set targetUser=root
set targetIP=192.168.0.85

set sourceFolder=C:\Users\patri\Desktop\Test_python\Source

set targetFolder=/home/root

REM Do not change this!!!
set Red=\033[1;31m
set Color_Off=\033[0m

REM Copy the folder
scp -r "%sourceFolder%" %targetUser%@%targetIP%:%targetFolder%

REM Check the error code
if %errorlevel% equ 0 (
    echo Folder copied successfully.
) else (
    echo %Red%Failed to copy the folder.%Color_Off%
)

exit /b 0
```

Code-Snippet 4.7: Kopieren der Python-Files

G FPGA Dummy Interrupt

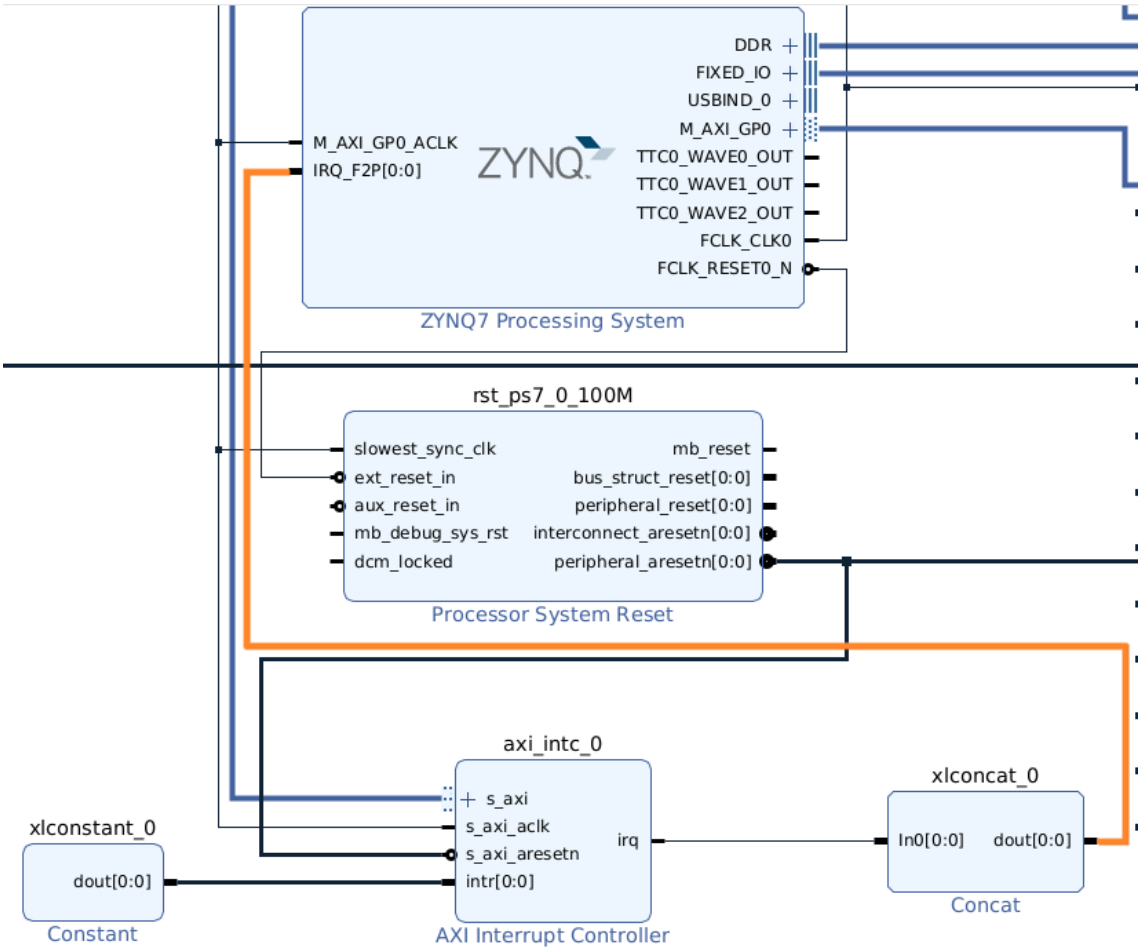

Abbildung 4.1: Screenshot vom Dummy Interrupt

H Motoren Regler

```

struct dataController* SpeedController4DCMotorSign(
          flink_subdev* pwm,
          flink_subdev* enc,
          float ts,
          uint8_t pwmChannel1,
          uint8_t pwmChannel2,
          uint8_t encChannel,
          int encTPR,
          float umax,
          float i,
          float kp,
          float tn) {
    struct dataController* controller = createController();
    controller->pwm = pwm;
    controller->enc = enc;

    controller->scale = (2.0 * M_PI) / (encTPR * FQD * i);
    controller->b0 = kp * (1 + ts / (2 * tn));
    controller->b1 = kp * (ts / (2 * tn) - 1);
    controller->umax = umax;
    controller->pwmChannel0 = pwmChannel1;
    controller->pwmChannel1 = pwmChannel2;
    controller->encChannel = encChannel;

    uint32_t temp = 0;
    flink_pwm_get_baseclock(pwm, &temp);
    controller->period = (temp)/controller->pwmFreq;
    flink_pwm_set_period(pwm, pwmChannel1, controller->period);
    flink_pwm_set_period(pwm, pwmChannel2, controller->period);
    flink_pwm_set_hightime(pwm, pwmChannel1, 0);
    flink_pwm_set_hightime(pwm, pwmChannel2, 0);

    flink_counter_get_count(enc, encChannel, &controller->prevPos
);

    return controller;
}

void setSpeed(struct dataController* controller, float v) {
    controller->desiredSpeed = v;
}

void run(struct dataController* controller) {
    struct timespec currentTime;
    clock_gettime(CLOCK_REALTIME, &currentTime);
    int64_t time = currentTime.tv_sec * 1000000000LL +
currentTime.tv_nsec;
    controller->dt = (time - controller->prevTime) * 1e-9f;
    controller->prevTime = time;

    int32_t actualPos;
    flink_counter_get_count(controller->enc, controller->
encChannel, &actualPos);
```

```
52        int16_t deltaPos = (short)(actualPos - controller->prevPos);
53        controller->prevPos = actualPos;
54        controller->absPos += deltaPos;
55        controller->speed = (deltaPos * controller->scale) /
    controller->dt;
56
57        float e = controller->desiredSpeed - controller->speed;
58        float controlValue = controller->prevControlValue +
    controller->b0 * e + controller->b1 * controller->e_1;
59
60        // anti windup
61        if (controlValue > controller->umax)
62            controlValue = controller->umax;
63        if (controlValue < -controller->umax)
64            controlValue = -controller->umax;
65
66        setPWM(controller, controlValue / controller->umax); //
    update PWM
67        controller->e_1 = e;
68        controller->prevControlValue = controlValue;
69    }
```

Code-Snippet 4.8: Motoren Regler

I Motorentest

```
    from Systemtechnik import Flink
    from Systemtechnik import Motorentreiber
    import threading
    import time

    ts=0.01
    pwmChannel1=2
    pwmChannel2=3
    encChannel=3
    encTPR=4250
    umax=5
    i=1
    kp=1
    tn=1
    run=True

    def periodic_call():
        while run:
            motor.run()  # Aufruf der Funktion motor.run() in jeder
   Iteration
            time.sleep(0.01)  # Wartezeit von 0.01 Sekunden

    speed = float(input("Geben Sie die gewuenschte Geschwindigkeit
   ein in radian pro sekunde [1/s]: " ))

    motor = Motorentreiber.Motor()
    flink = Flink.Flink()
    flink.flink_open()
    pwmDev = flink.flink_get_subdevice_by_unique_id(48)
    counterDev = flink.flink_get_subdevice_by_unique_id(64)
    motor.SpeedController4DCMotorSign(pwmDev, counterDev, ts,
   pwmChannel1, pwmChannel2, encChannel, encTPR, umax, i, kp, tn)
    motor.setSpeed(speed)


    thread = threading.Thread(target=periodic_call)
    thread.start()

    while True:
        try:
            new_speed = float(input("Geben Sie die neue
   Geschwindigkeit ein (oder 'exit' zum Beenden): "))
            motor.setSpeed(new_speed)
            print("Gesetzte Geschwindogkeit", new_speed)
            print("geschwindigkeit Motor", motor.getSpeed())
        except ValueError:
            run=False
            break
```

Code-Snippet 4.9: Motorentest

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, 17. Juli 2023

Patrick Good