**lowRISC**

**Installation der benötigten Tools**

Um die RISC-V Tools erstellen zu können, werden verschiedene Pakete benötigt, welche vorher zu installieren sind.

-autoconf: Produziert shell scripts zur automatischen Konfiguration von Software

-automake: Tool zur Generierung von Makefile Dateien

-autotools-dev: Erneuert die Infrastruktur für config-Dateien

-libmpfr-dev: Mehrfach genaue Arithmetikbibliothek – Entwicklungswerkzeuge

-libmps-dev: siehe libmpfr-dev

-libgmp-dev: siehe libmpfr-dev

-gawk: GNU-Projekt Implementierung der AWK Programmiersprache

-build-essentials: Benötigte Pakete zur Kompilierung eines Pakets

-bison: YACC- kompatibeler Parser-Generator

-flex: schneller lexical- Analysator Generator

-texinfo : hypertextfähiges Dokumentationssystem

-gperf: Hash-Funktionsgenerator

-libncurses5-dev: Ncurses-Bibliotheken für Entwickler

-libusb-1.0.-0: Programmbibliothek zum Schreiben und Lesen von USB-Geräten

-libboost-dev: Boost C++ Bibliotheken

-Git: Software zur Versionsverwaltung von Dateien

CODE:

Sudo apt-get install autoconf automake autotools-dev curl \

Libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison \

Flex texinfo gperf libncurses5-dev libusb-1.0-0 libboost-dev \

Git

**Download des lowRISC-Git-Repository**

Hierbei ist es empfehlenswert, dass gesamte Repository zu klonen, statt einzelne Untermodule.

Code:

Cd ˜/lowRISC/DIR

#download des untether-v0.2-Branch

git clone –b untether-v0.2 –recursive <https://github.com/lowrisc/lowrisc-chip.git>

cd lowrisc-chip

**Setzen der RISCV-Variablen**

Um die passende Umgebung zu schaffen, müssen die Variablen gesetzt werden. Hierfür ist zu erst der \emph{TOP}-Pfad anzugeben:

CODE:

export TOP=[Gesamter Pfad zu dem lowRISC-Verzeichnis]

Bevor nun das Setupskript \emph{lowrisc-chip/set\_riscv\_env.sh} ausgeführt wird, um die restlichen Variablen anzupassen, muss in diesem Skript das passende Board angegeben werden. Im Falle des verwendeten Boards Nexys4DDR wie folgt:

CODE:

[...]

#choose the FPGA board (KC705 in default)

export FPGA\_BOARD=nexys4

Die angepasste Datei wird im Terminal mit:

CODE:

Source set\_riscv\_env.sh

Ausgeführt.

Es gilt zu beachten, dass das Einrichten dieser Umgebung jeweils nur für das genutzte Terminal gilt. Wird ein neues beziehungsweise weiteres Terminal geöffnet, müssen vorherigen Schritte erneut ausgeführt werden.

**Bauen des RISC-V Cross-Kompilierung-Tools**

In dem Repository wurde ein Skript mitgeliefert, welches die meisten Tools für die Cross-Kompilierung und Spike vorbereitet.

Zum Ausführen gilt es in den \emph{$TOP/riscv-tools}-Ordner zu wechseln und dass Skript mit dem Befehl :

CODE:

./build.sh

auszuführen.

Nach der erfolgreichen Kompilierung sollten der Cross-Compiler, sowie Spike zur Verfügung stehen. Geprüft werden kann dies mit:

CODE:

Which riscv64-unknown-elf-gcc

**Erstellen des Linux-Cross-Compilers**

Neben dem RISCV-Cross-Compiler, welcher für das System an sich benutzt wird, wird ebenfalls ein Cross-Compiler für Programme, welche unter Linux ausgeführt werden sollen, benötigt.

Hierfür werden lediglich die Variablen innerhalb der RISCV-Umgebung angepasst.

CODE:

cd $TOP/riscv-tools/riscv-gnu-toolchain

#wenn der Ordner „build“ bereits existiert, kann der nächste Schritt ignoriert werden

mkdir build

cd build

../configure –prefix=$RISCV

make –j(Anzahl Prozessorkerne) linux

Auch hier kann die Verfügbarkeit mit

CODE:

which riscv64-unkown-linux-gnu-gcc

Überprüft werden.

**Kompilieren des Linux Kernel**

Für den Linux Kernel, welcher mit Hilfe von Spike simuliert oder auf dem FPGA-Board gebootet werden kann, muss ebenfalls die Umgebung angepasst werden.

Es werden verschiedene Dateien benötigt, die unter den folgenden Verlinkungen heruntergeladen werden können.

CODE:

# Setzen der RISCV Umgebungsvariablen

cd $TOP/riscv-tools

curl [https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.14.41.tar.xz \](https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.14.41.tar.xz%20\)

| tar -xJ

cd linux-3.14.41

git init

git remote add origin <https://github.con/lowrisc/riscv-linux.git>

git fetch

git checkout –f –t origin/untether-v0.2

make ARCH=riscv defconfig

make ARCH=riscv –j vmlinux

Nach der Kompilierung sollte das Linux Kernel Image mit Hilfe von

CODE:

ls –l vmlinux

zu finden sein.

**BusyBox**

BusyBox wird im Root-Image genutzt um die grundlegende Shell-Umgebung bereitzustellen. Um ein eigenes Image zu erstellen, muss die BusyBox-Binärdatei vorher generiert werden:

CODE:

# Setzen der RISCV Umgebungsvariablen

cd $TOP/riscv-tools

curl –L <http://busybox-net/downloads/busybox-1.21.1.tar.bz2> | tar -xJ

cd busybox-1.21.1

cp $TOP/risscv-tools/busybox\_config .config

make –j(Anzahl Prozessorkerne)

Wurde die Kompilierung erfolgreich abgeschlossen, wurde die Binärdatei im selben Verzeichnis generiert:

CODE:

ls –l busybox

Mit Hilfe des \emph{make\_root}-Skripts kann das Root-Image (root.bin) erstellt werden:

$TOP/riscv-tools/make\_root.sh

**FPGA-Demo**

Um nun das FPGA-Projekt zu kompilieren wird die Bitstream-Datei mit Hilfe eines Skripts erstellt:

CODE:

cd $TOP/fpga/board/nexys4

make bitstream

Die Bitstream-Datei ist anschließend unter dem Pfad \emph{lowrisc-chip-imp/lowrisc-chip-imp.runs/impl\_1/chip\_top.bit} zu finden.

**Kompilieren und Ausführen von Bare-Metal-Beispielen**

In dem \emph{lowrisc}-Verzeichnis sind bereits vier Bare-Metal Beispiele hinterlegt. Mit Hilfe einfacher Skripte können diese kompiliert werden und die neue Bitstream-Datei generiert werden.

Für das klassische Hello-World Programm ist folgender Aufruf notwendig:

CODE:

Cd $TOP/fpga/board/nexys4

Make hello

Anschließend befindet sich im Verzeichnis \emph{lowrisc-chip-imp/lowrisc-chip-imp.runs/impl\_1} eine \emph{chip\_top.new.bit}-Datei, welche auch zum konfigurieren des FPGA genutzt werden sollte.

Mit Hilfe von Vivado kann der FPGA mit der Bitstrea-Datei programmiert werden.

Um die UART-Schnittstelle zu testen wird der \emph{Minicom} benutzt, welcher mit:

CODE:

Minicom –D /dev/(passende USB-Schnittstelle)

Aufgerufen wird.

Nun sollte als Ausgabe Hello World erscheinen.

**Vorbereiten des eigenen Images**

Um RISC-V Linux zu starten, werden drei Dateien auf der SD-Karte benötigt:

1. boot: Der Bootloader(Berkeley bootloader) um den Linux Kernel zu laden
2. vmlinux: Der RISC-V Linux Kernel
3. root.bin: Das ramdisk-Image

Zum Kopieren des Kernels und des Images:

CODE:

Cd $TOP/riscv-tools

Cp linux-3.14.41/vmlinux /(Pfad der SD-Karte)/vmlinux

Cp busybox-1.21.1/root.bin /(Pfad der SD-Karte/root.bin

Der fehlende Bootloader kann wie folgt generiert werden:

CODE:

Cd $TOP/fpga/board/nexys4

Make bbl

Cp bbl/bbl /(Pfad der SD-Karte)/boot