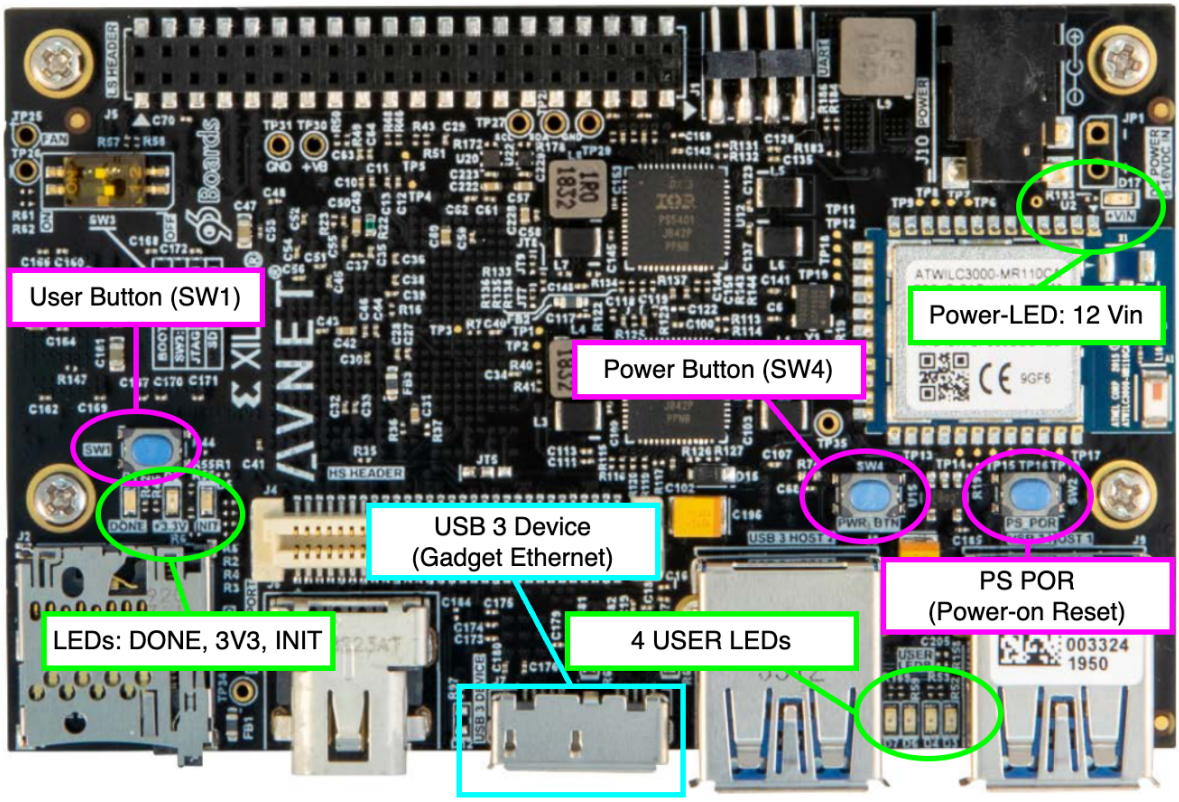
Praktikum 1 – Inbetriebnahme des ultra96 Boards und Installation der Toolchain

1. Lernziele

- Sie können eine Cross-Toolchain aufsetzen.
- Sie können auf ihrem Host-Rechner ein Programm für ihre embedded-Plattform kompilieren und auf die Plattform transferieren.

2. Überblick über die Hardware

Das ultra96-Board verfügt über einen Xilinx Ultrascale+ SoC (System on Chip), der ein FPGA-Fabric und mehrere ARM-Prozessoren vereint.
Die wichtigsten Komponenten für das Praktikum sind in der folgenden Grafik gezeigt:



LEDs: DONE, 3V3, INIT: zeigen an, in welchem Status das FPGA ist.
Power Button: Das Board startet nicht automatisch, wenn die Versorgungsspannung angeschlossen wird, sondern erst nach Druck auf diesen Button.
USER LEDS: LEDs, die vom ARM-Prozessor aus gesteuert werden können.

PS POR: Power-on-Reset, löst einen Reset des ARM-Prozessors aus, wie wenn das Board erst frisch eingeschaltet würde.
Power-LED: Zeigt an, ob die Versorgungsspannung (12 V) anliegt.
USB 3 Device (Gadget Ethernet): Am USB 3 Port kann über ein USB-Kabel eine Netzwerkschnittstelle zu ihrem Host-Rechner aufgebaut werden.

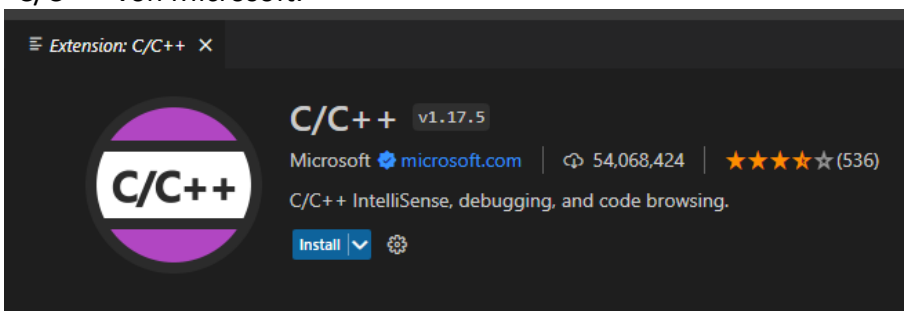
Der eigentliche SoC-Chip befindet sich auf der Unterseite des Boards.

3. Installation Toolchain

Visual Studio Code

Visual Studio Code (VSCode) ist ein Texteditor mit den nötigen Schnittstellen, um eine Entwicklungsumgebung zu erstellen. Hierzu werden auch ein Compiler (GCC) und ein Buildsystem (Make) benötigt.

Installieren Sie Visual Studio Code von https://code.visualstudio.com/. Nach der Installation manövrieren Sie in den Extensions-Store von VS Code und installieren die Erweiterung für "C/C++" von Microsoft.



ARM Cross-Toolchain

Für die Entwicklung auf ihrem Host-Rechner benötigen Sie eine Cross-Toolchain für die ultra96-Plattform, die Executables für die ARM Cortex-A53 CPU builden kann. Wir verwenden dafür die AArch64 GNU/Linux Toolchain, Version 12.3Rel1. Bei einer Cross-Toolchain kommt es darauf an, dass sie sowohl auf ihr Host-System (z.B. Windows, mingw-w64-i686) als auch auf das Target-System (AArch64 GNU/Linux, aarch64-none-linux-gnu) angepasst ist.

Toolchain für Windows:
Laden Sie von https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads die Cross-Toolchain herunter.
Wählen Sie die Version 12.3.Rel1 (ganz nach unten scrollen) und achten Sie darauf, die AArch64 GNU/Linux target (aarch64-none-linux-gnu) zu wählen.

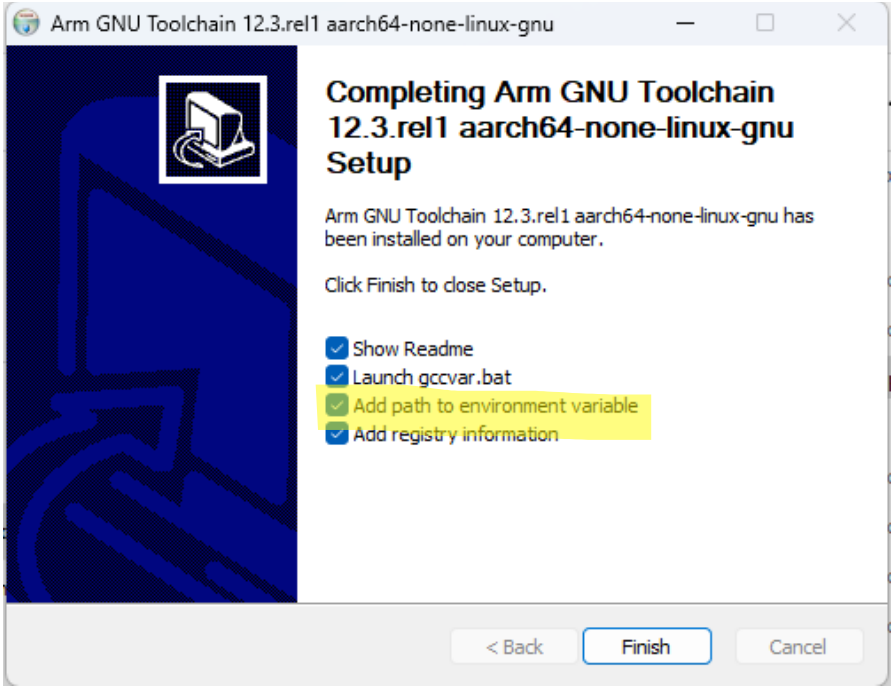
What's new in 12.3.Rel1
This release is based on GCC 12.3

In this release
Windows (mingw-w64-i686) hosted cross toolchains

AArch64 GNU/Linux target (aarch64-none-linux-gnu)

- arm-gnu-toolchain-12.3.rel1-mingw-w64-i686-aarch64-none-linux-gnu.zip
- arm-gnu-toolchain-12.3.rel1-mingw-w64-i686-aarch64-none-linux-gnu.zip.asc
- arm-gnu-toolchain-12.3.rel1-mingw-w64-i686-aarch64-none-linux-gnu.zip.sha256asc
- arm-gnu-toolchain-12.3.rel1-mingw-w64-i686-aarch64-none-linux-gnu.exe
- arm-gnu-toolchain-12.3.rel1-mingw-w64-i686-aarch64-none-linux-gnu.exe.asc
- arm-gnu-toolchain-12.3.rel1-mingw-w64-i686-aarch64-none-linux-gnu.exe.sha256asc

Für die Installation empfiehlt sich der Download des **.exe-Installers,** welcher die Toolchain gleich in der PATH-Variable registrieren. Setzen Sie bei der Installation das entsprechende Häkchen, damit der Cross-Compiler von der Kommandozeile aufgerufen werden kann.



Als Funktionstest öffnen Sie eine Windows Powershell und rufen den Cross-Compiler auf mit dem Befehl ‘aarch64-none-linux-gnu-gcc -v’. Die Toolchain ist korrekt installiert, wenn Sie eine Ausgabe erhalten, die etwa so aussieht:

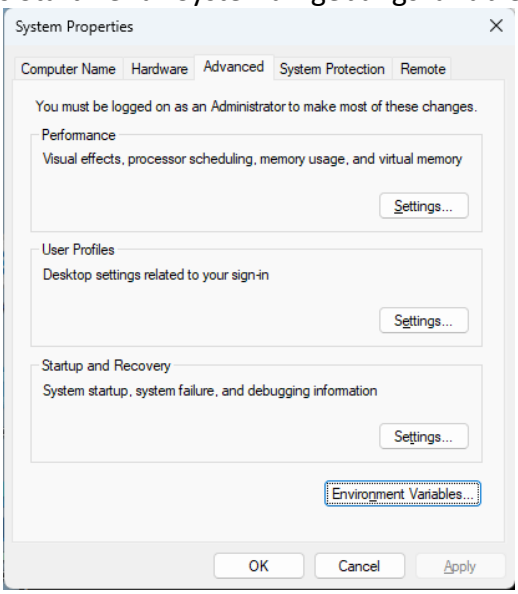
```
PS C:\Users\welo> aarch64-none-linux-gnu-gcc -v
Using built-in specs.
COLLECT_GCC=C:\Program Files (x86)\Arm GNU Toolchain aarch64-none-linux-gnu\12.3 rel1\bin\aarch64-none-linux-gnu-gcc.exe
COLLECT_LTO_WRAPPER=c:/program\ files\ (x86)/arm\ gnu\ toolchain\ aarch64-none-linux-gnu/12.3\ rel1/bin/../libexec/gcc/a
arch64-none-linux-gnu/12.3.1/lto-wrapper.exe
Target: aarch64-none-linux-gnu
Configured with: /data/jenkins/workspace/GNU-toolchain/arm-12-3/src/gcc/configure --target=aarch64-none-linux-gnu --pref
ix= --with-sysroot=/aarch64-none-linux-gnu/libc --with-build-sysroot=/data/jenkins/workspace/GNU-toolchain/arm-12-3/buil
d-mingw-aarch64-none-linux-gnu/install//aarch64-none-linux-gnu/libc --with-bugurl=https://bugs.linaro.org/ --enable-gnu-
indirect-function --enable-shared --disable-libssp --disable-libmudflap --enable-checking=release --enable-languages=c,c
++,fortran --with-gmp=/data/jenkins/workspace/GNU-toolchain/arm-12-3/build-mingw-aarch64-none-linux-gnu/host-tools --wit
h-mpfr=/data/jenkins/workspace/GNU-toolchain/arm-12-3/build-mingw-aarch64-none-linux-gnu/host-tools --with-mpc=/data/jen
kins/workspace/GNU-toolchain/arm-12-3/build-mingw-aarch64-none-linux-gnu/host-tools --with-isl=/data/jenkins/workspace/G
NU-toolchain/arm-12-3/build-mingw-aarch64-none-linux-gnu/host-tools --host=i686-w64-mingw32 --enable-fix-cortex-a53-8434
19 --with-libiconv-prefix=/data/jenkins/workspace/GNU-toolchain/arm-12-3/build-mingw-aarch64-none-linux-gnu/host-tools -
-with-pkgversion='Arm GNU Toolchain 12.3.Rel1 (Build arm-12.35)'
Thread model: posix
Supported LTO compression algorithms: zlib
gcc version 12.3.1 20230626 (Arm GNU Toolchain 12.3.Rel1 (Build arm-12.35))
```

Erhalten Sie jedoch eine Fehlermeldung, gibt es noch Probleme mit der Installation. Fragen Sie den Betreuer…

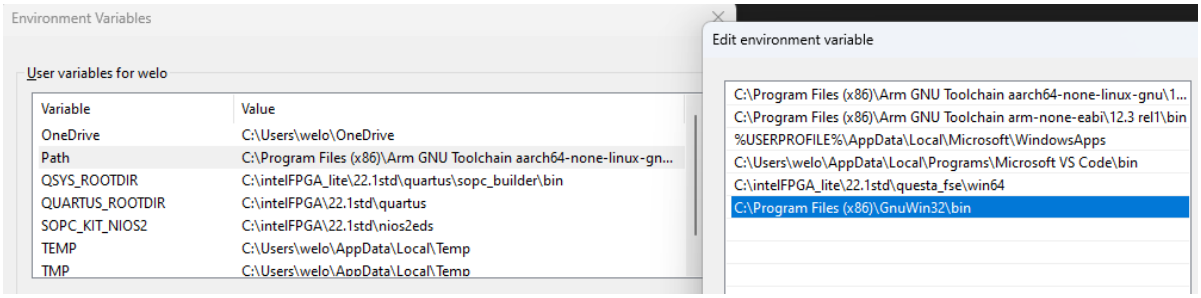
make

Installieren Sie gnu-make: Laden Sie das ***Complete package, except sources*** herunter und installieren sie dieses:
https://gnuwin32.sourceforge.net/packages/make.htm

Damit VS Code das make-Tool findet, müssen Sie seinen Pfad in den Umgebungsvariablen setzen. Öffnen Sie über das Startmenü "Systemumgebungsvariablen bearbeiten":



Fügen Sie den Pfad zu *make* (C:\Program Files (x86)\GnuWin32\bin) bei der Variable "PATH" hinzu:



Damit Visual Studio Code die neue Umgebungsvariable übernimmt, beenden Sie VS Code und starten Sie es erneut.

Toolchain für Linux:
Hier ist die Cross-Toolchain über das Package Management verfügbar:
sudo apt-get install gcc-aarch64-linux-gnu
sudo apt-get install gcc-arm-none-eabi

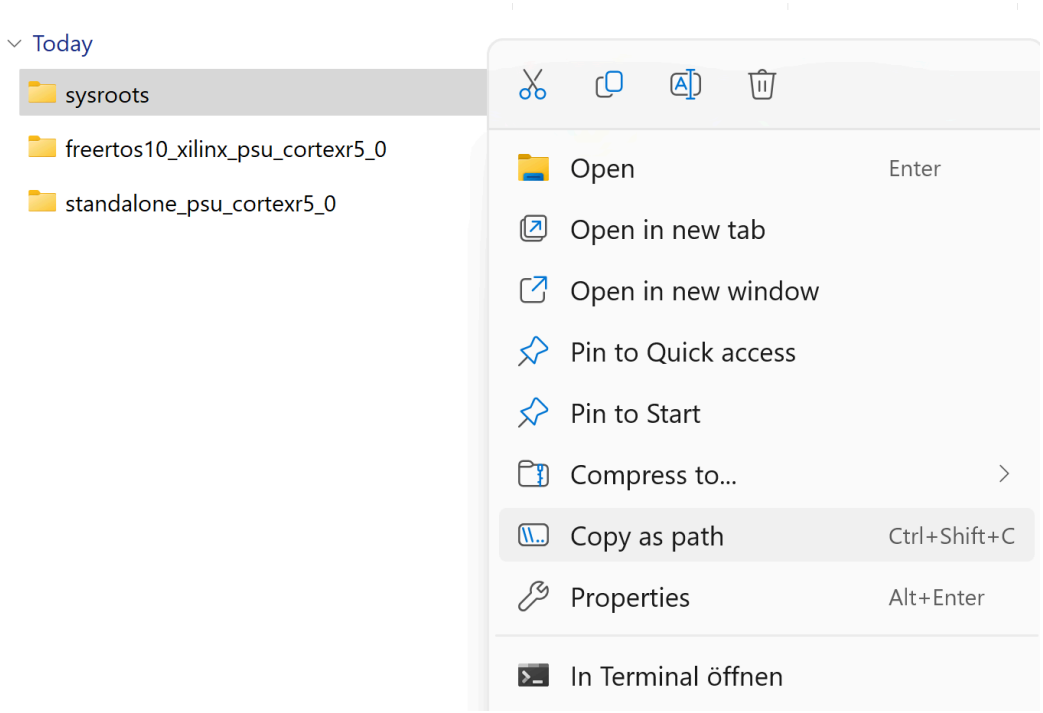
Das Build-Tool make bereits installiert.

Toolchain für Apple:
Für Apple-Rechner wird noch keine Cross-Toolchain für Linux-Targets angeboten, weshalb Sie besser in einer Windows- oder Linux-VM arbeiten.

Ultra96 Linux sysroots

Sie benötigen die plattformspezifischen Linux-Bibliotheken für das ultra96-Board, damit die Cross-Toolchain gegen diese Libraries linken kann. Diese sogenannten sysroot-Files benötigen ca. 4.6 GB Platz. Laden Sie die Datei cross-compile.zip von Moodle herunter und entzippen Sie diese an einem geeigneten Ort, z.B. unter C:\arm\cross-compile. Benutzen Sie dafür wenn möglich ein Tool wie 7zip. Das integrierte zip-Tool von Windows brilliert hier mit rekordverdächtigen Bearbeitungszeiten von bis zu 30 Minuten.

- Damit der Pfad zu den sysroot-Dateien für die Laborübungen zur Verfügung steht, definieren Sie eine weitere **Umgebungsvariable SYS_ROOT** mit dem Pfad zum sysroots-Ordner, also zum Beispiel **C:\arm\cross-compile\sysroots**.



Nun ist ihr Host-Rechner bereit für die Entwicklung von Software für das ultra96-Board.

Checkpoint:

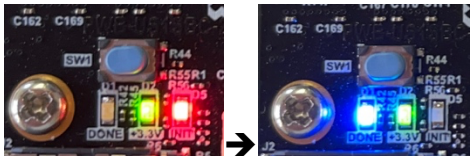
- Existieren die folgenden Systemumgebungsvariablen:
  - SYS_ROOT: Pfad zum Ordner sysroots
  - PATH: Pfad zum Crosscompiler und zu make

4. Inbetriebnahme

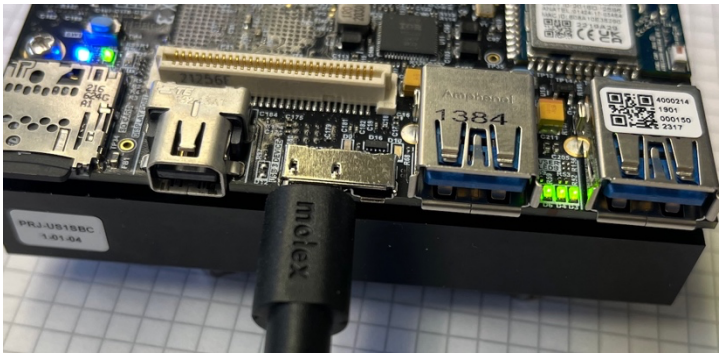
Sie nehmen nun das ultra96-Board in Betrieb. Eine SD-Karte mit embedded Linux ist bereits vorbereitet. Nach dem Boot-Vorgang (ca. 30 Sekunden) steht am USB-Anschluss eine Netzwerkschnittstelle zur Verfügung, über die Sie mit dem Board kommunizieren können.

1. Schliessen Sie das ultra96-Board an die Stromversorgung an. Die **Power-LED** (12 Vin) sollte nun grün leuchten.
2. Stellen Sie sicher, dass die SD-Karte im SD-Slot sitzt.
3. Drücken Sie auf den **Power Button** (SW4), um das Board zu starten.
4. Das Board durchläuft nun den Boot-Prozess. Zuerst wird der FPGA-Teil konfiguriert, danach wird Linux gestartet. Sie erkennen die erfolgreiche FPGA-Konfiguration an den drei LEDs:

DONE	3V3	INIT	Bedeutung
--	grün	rot	INIT zeigt an, dass das FPGA gerade initialisiert wird
blau	grün	--	DONE zeigt an, dass das FPGA konfiguriert ist



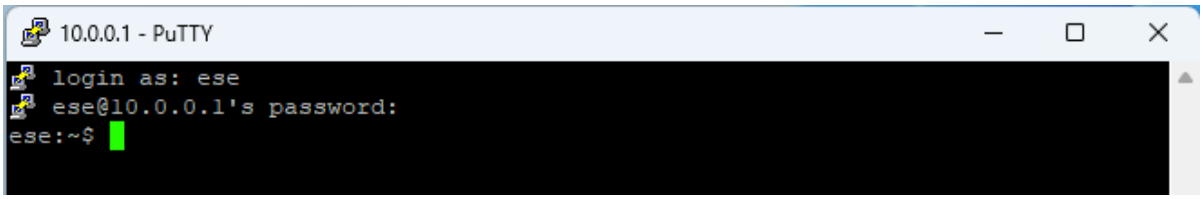
5. Nachdem das FPGA konfiguriert wurde, dauert es noch ca. 25 Sekunden, bis Linux fertig gebootet ist. Warten Sie deshalb so lange, bevor Sie den nächsten Schritt ausführen.
6. Für die Kommunikation mit dem ultra96-Board verwenden wir eine USB-Gadget-Ethernet-Schnittstelle, also eine Ethernet-Schnittstelle über ein USB-Kabel. Schliessen Sie ein USB-micro-Kabel am **USB 3 DEVICE** Stecker des Boards an, wie im folgenden Bild gezeigt. Das andere Ende des Kabels schliessen Sie an Ihrem Rechner an.



   Auf einem Windows- oder Linux-Rechner ist dafür kein Treiber nötig.

   Auf einem Mac benötigen Sie einen inoffiziellen RNDIS-Treiber, z.B. von https://forum.mudita.com/t/tethering-macos-configuration/5230, oder Sie verwenden eine Windows-VM, dann erscheint die Schnittstelle als "Remote NDIS Compatible Device".
7. Nun können Sie über eine SSH-Verbindung unter der IP-Adresse 10.0.0.1 auf das ultra96-Board zugreifen. Verwenden Sie dafür z.B. Putty, oder eine Windows Command

Shell mit dem Befehl `ssh ese@10.0.0.1`. Der Loginname lautet `ese`, das Passwort `ese`.

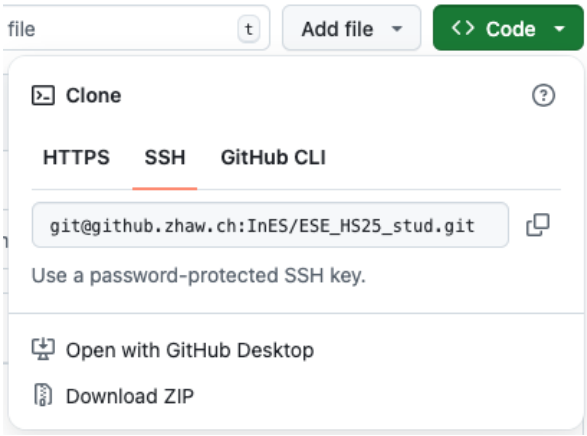


5. Ein erstes Programm…

Fürs Erste benutzen wir nur den Mikroprozessor-Teil des ultra96-Boards und lassen den FPGA-Teil noch aussen vor.

Wir arbeiten mit einem makefile, welches den Buildvorgang steuert, den richtigen Cross-Compiler aufruft und die nötigen Schritte für uns weitgehend automatisiert. Die Dateien für das Praktikum befinden sich auf einem Git-Repository. Laden Sie sich diese Dateien als .zip herunter, oder checken Sie das Repository aus.

Git-Repo: https://github.zhaw.ch/InES/ESE_HS25_stud



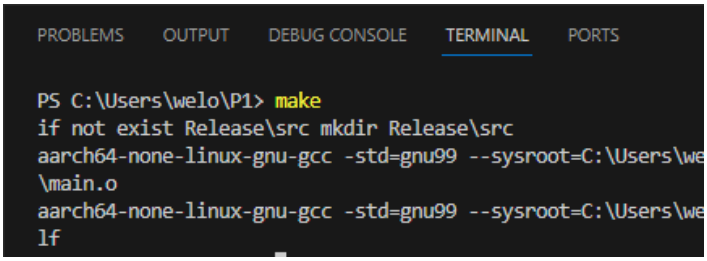
Das Verzeichnis P1 enthält nur ein makefile und ein Verzeichnis mit dem Namen src. Legen Sie sämtliche source-Dateien im Verzeichnis src ab, damit sie von make gefunden werden.

1. Öffnen Sie in VisualStudio Code das Verzeichnis P1.
2. Im makefile sind verschiedene *build targets* definiert. Öffnen Sie mit VS Code das makefile.
   build targets sind Rezepte in der folgenden Form:
   ```
   Name: <Abhängigkeiten>
         Instruktionen für den build
   ```

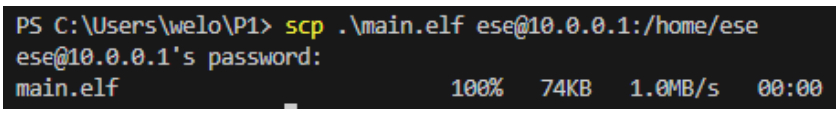
   Welche *build targets* finden Sie?

   _____

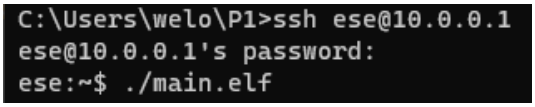
3. Editieren Sie nun im Unterverzeichnis src/ die Datei "main.c" und schreiben Sie ein altbekanntes "Hello World"-Programm. Sie dürfen auch kreativ werden…
4. Builden Sie das Programm mit *make*:
   - Starten Sie in VS Code ein Terminal: Menü Terminal > New Terminal
   - Geben Sie im Terminal *make* ein.



   Mit "*make*" wird das Target all gebuildet. Die anderen Targets können Sie auf der Kommandozeile mitgeben, z.B. *make test* (prüft, ob der Cross-Compiler gefunden wird).
5. Das erstellte Executable main.elf kopieren Sie über die Netzwerkverbindung auf ihr Target (das ultra96-Board). Dafür benutzen Sie *secure copy* mit dem folgenden Befehl:
   `scp -O main.elf ese@10.0.0.1:/home/ese/`



6. Öffnen Sie danach ein Windows Command Prompt und verbinden Sie sich über SSH (Secure Shell) mit dem ultra96-Board. Danach können Sie auf der Kommandozeile des ultra96-Boards ihr Executable ausführen:
   `ssh ese@10.0.0.1`



   Falls Sie eine Fehlermeldung "Permission denied" erhalten, geben Sie main.elf die executable-Rechte:
   `chmod +x main.elf`

6. Bewertungskriterien:

- Die Cross-Toolchain ist installiert und so konfiguriert, dass Sie Software für das ultra96-Board entwickeln können. Make ist funktionsfähig und kann mit der Cross-Toolchain ihre Embedded Software builden.
- Ihr ausführbares Programm "hello world" kann auf das ultra96-Board transferiert und dort ausgeführt werden.