Salzbrenner Media

Anleitung UART-SPI-CPLD

Verfasser:

Wjatscheslaw Rein

Änderungshistorie

Version	Verfasser	Änderungsbeschreibung	Freigabedatum
1.0	Wjatscheslaw Rein	Initiale Erstellung	31.08.2022
1.1	Wjatscheslaw Rein	Zwischenstand	01.08.2022
1.2	Wjatscheslaw Rein		

Inhaltsverzeichnis

1. Testaufbau

1.1 Concerto-Board-Schematics-VAR-SOM-6UL

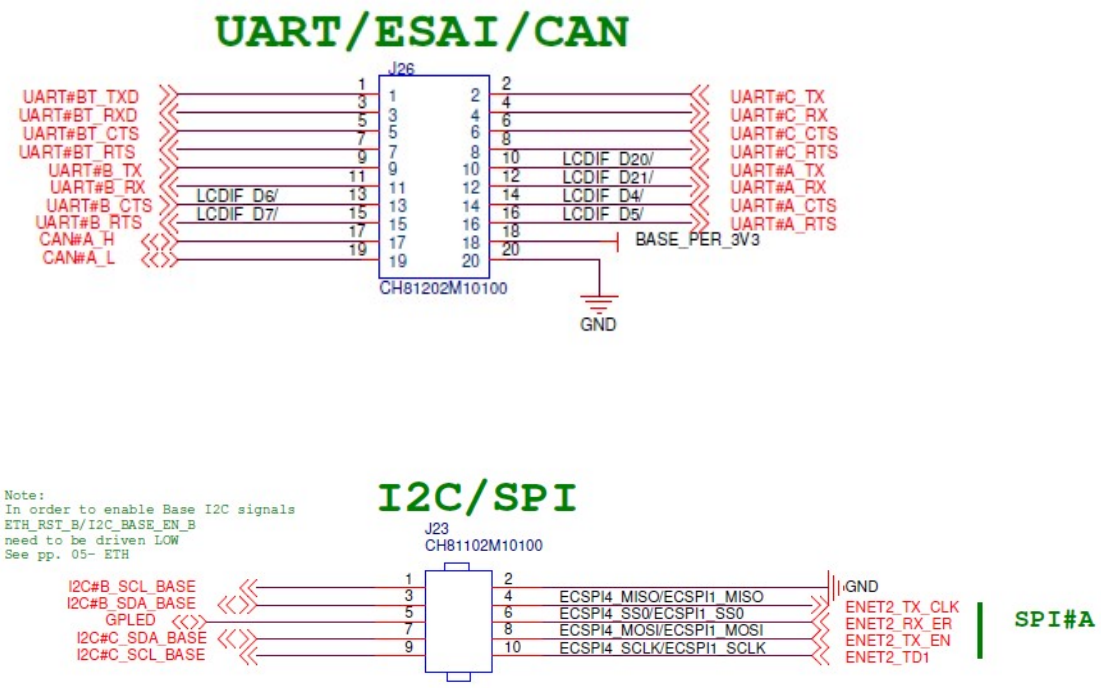


Abbildung 1.1-1: Concerto-Board-Schematics-VAR-SOM-6UL.

1.2 Basys 3™ FPGA Board (DIGILENT)

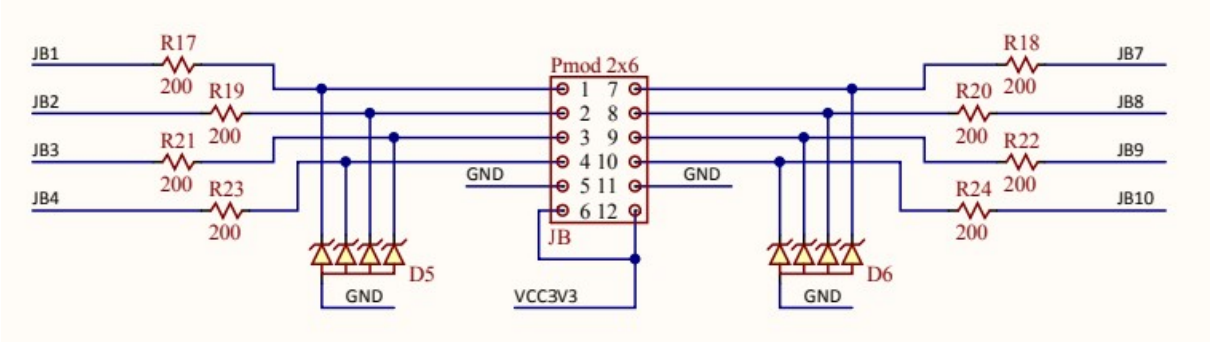


Abbildung 1.2-1: Basys 3™ FPGA Board.

1.3 Verbindung

Concerto-Board		Basys 3™ FPGA Board		
J23	**Pin**	**JB**	**Pin**	
ECSPI SS0	6	JB1	1	SPI CS
ECSPI SCLK	10	JB2	2	SPI Clock
ECSPI MOSI	8	JB3	3	SPI MOSI
ECSPI MISO	4	JB4	4	SPI MISO
GND	2	JB5	5	GND
-	-	JB6	6	VCC3V3 (not connected)
GPLED	5	JB7	7	IRQ (interrupt not used)
J26	20	JB11	11	GND

2. Projektaufbau

Ein UART-SPI "Projekt" lässt sich in zwei eindeutige Bereiche unterteilen. Zu einem gehören Treibern und andere für deren Funktion notwendige Komponenten. Zum anderen die USER-Programme, sogenannte Applikationen, die eine Schnittstelle zwischen Benutzer und Treiber repräsentieren. Ein Bereich ist somit dem Kernel-, und ein anderen dem Userspace zuzuordnen.

2.1 Kernel Space

Ein Kernel-Space beinhaltet vor allem hardwarespezifische Device Tree Files, für den Projekt benötigte Treibern sowie eine Menge Service- bzw. Funktionsdateien, die zu dem Treibersystem gehören. Da in unserem Fall von einem sogenannten build-in-driver die Rede ist, müssen auch entsprechende Scripts angepasst werden (Makefiles, Kconfig, BUILTIN- und ORDER-Dateien). Es existieren mehrere Varianten, wie man ein lauffähiges Debian-Projekt auf eine SD-Karte bekommt. Jede einzelne hat Vor- und Nachteile, deswegen werden diese in nächsten Unterkapiteln kurz erläutert.

2.1.1 SVN Archiv

Die einfachste Variante berührt sich auf die Möglichkeit fertiggestelltes und mit allen notwendigen Änderungen und Anpassungen vorliegendes Projektarchiv zu nehmen und ihn direkt auf SD-Karte zu übertragen. Vorausgesetzt ist dieser in einer Versionsverwaltung (z.B.: TortoiseSVN o. Ä.) gesichert und zum Auschecken bereit ist. Dabei sind noch drei unterschiedliche Szenarien vorgesehen:

Image

In dem Ordner von SVN-Archiv (siehe die Abb. 2.1.1-1) namens „Image“ ist eine Datei „*.img“ gespeichert, die eine komplette, kompilierte SD-Karten Abbildung darstellt. Diese lässt sich direkt auf die Karte schreiben.

Debian-Distribution komprimiert

In dem gleichen Ordner befindet sich auch eine komprimierte Variante von Debian-Distribution, die allerdings nach dem Extrahieren eine vollständige Kompilierung braucht. Wie man ein Image erstellen kann, wird in dem Kapitel 2.1.2 beschrieben (Beachte: nur die tatsächlich notwendige Schritte!).

Komponenten

Zusätzlich sind die einzelnen Komponenten und deren SVN-Versionsgeschichte für den UART-SPI Projekt in den zuständigen Ordnern gespeichert und können bei Bedarf geändert werden. Die Komponenten sind Treiber, Headerdateien, Scripts und Device Tree Source Files. Diese Variante erleichtert zwar die Verwaltung von Projektdateien (nur die, die geändert sind) fördert aber im Vergleich zu den zwei oben genannten ein tiefes Verständnis von dem Aufbau und Struktur sowie dem Zusammenspiel der Systemteilen.

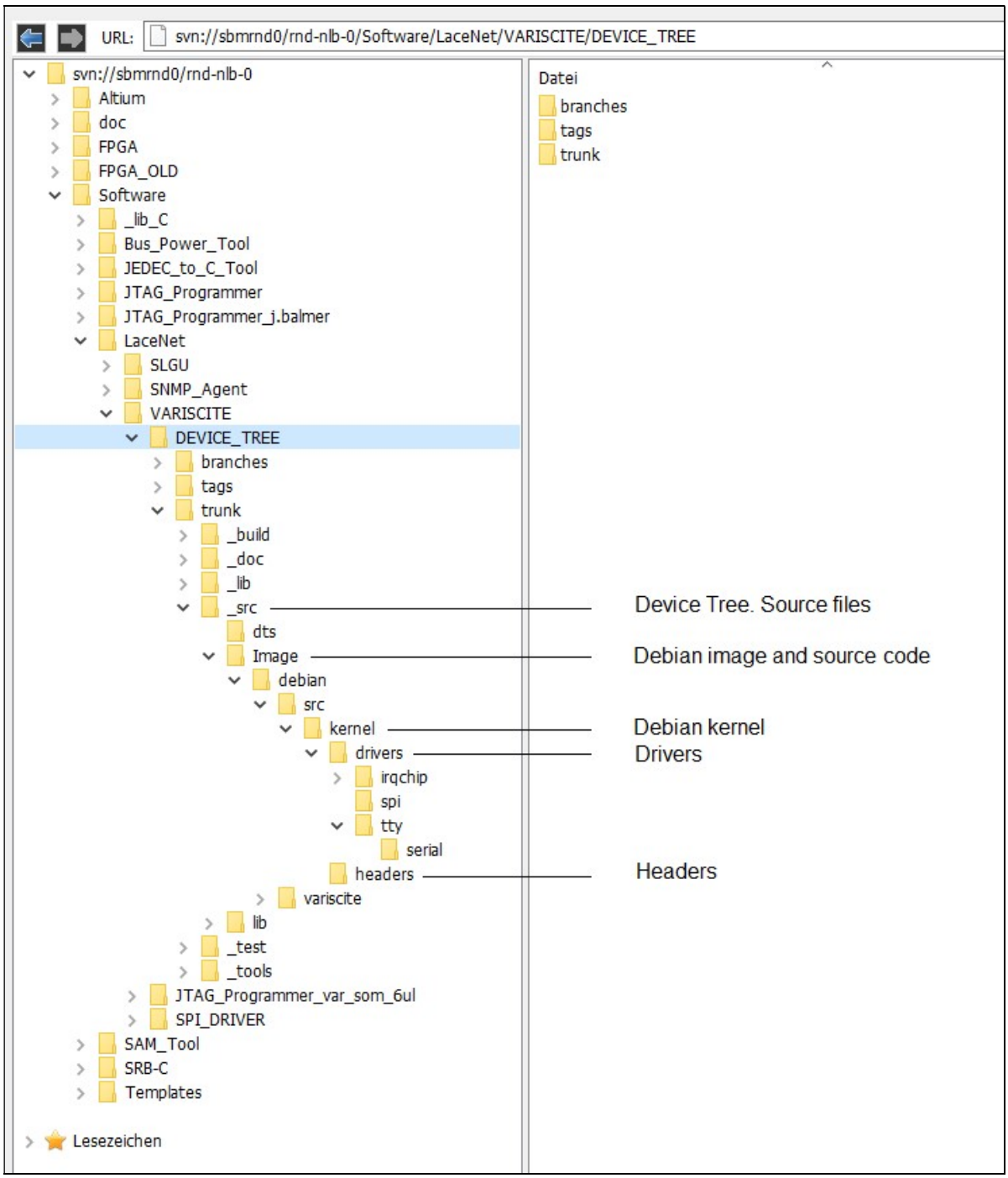


Abbildung 2.1.1-2: TortoiseSVN Projektarchiv.

Aktuelle SVN Revision Nr.: 1515

2.1.2 Debian Distribution

Eine genaue Beschreibung der Schritt-für-Schritt Installation einer Debian-Distribution ist komplett von der Variscite Website kopiert und dient hier als Anleitung für den Fall, wenn kein Debian-Image aus SVN zur Verfügung steht. *Hinweis*: externe, abweichende Nummerierung von Kapiteln beachten.

Zu beachten: Selbstverständlich ist ein Debian Projekt nach der Installation ganz jungfräulich und benötigt entsprechende Änderungen bzw. Anpassungen (siehe dazu Kapitel 2.1.1 „Komponenten“. Device Tree, Header, Treiber usw.), damit UART-SPI Funktionalitäten unterstützt werden. Diese können einzeln aus SVN extrahiert (ausgecheckt) werden.

Aktuell installiert : qemu-user-static/bionic-security 1:2.11+dfsg-1ubuntu7.37 i386

DART-6UL - Debian Bullseye 11 with 5.4-2.1.x-imx_var01 Linux release

Link

DART-6UL in this wiki refers to both the Variscite DART-6UL and VAR-SOM-6UL SOMs.

Contents

1 Overview

This page describes how to build and install Debian distribution (Bullseye) on Variscite boards with DART-6UL.

Please note that the build script is based on *debootstrap*. As described in the following instructions, it's kindly suggested to **create the build folder on the main Ubuntu volume on your host PC** rather than an external media.
Using an external media, although automatically mounted, will cause debootstrap to complain about access rights (even when running the script with sudo).

2 Create build environment

These instructions were tested on **Ubuntu 18.04/20.04 x64 host PCs**. When using other distributions, there may be issues.

Note: Variscite provides Docker containers that can be used for a development environment as an alternative to using a virtual machine or a dedicated computer. To learn more, please see Variscite's Docker Build Environment guide.

2.1 Installing required packages

On Ubuntu building machine:

```
$ sudo apt-get install binfmt-support qemu qemu-user-static debootstrap kpartx \
lvm2 dosfstools gpart binutils bison git lib32ncurses5-dev libssl-dev python-
m2crypto gawk wget \
git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat
libsdl1.2-dev \
autoconf libtool libglib2.0-dev libarchive-dev python-git xterm sed cvs subversion
\
kmod coreutils texi2html bc docbook-utils python-pysqlite2 help2man make gcc g++ \
desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev mercurial automake groff curl
\
lzop asciidoc u-boot-tools mtd-utils device-tree-compiler flex cmake
```

While installing the packages, Ubuntu 20.04 users may receive the warning

```
Package python-git is not available, ...
```

Since the package is only available in up to Ubuntu 18.04, just ignore it and proceed further running

```
$ sudo apt-get install python3-git python3-m2crypto
```

Note: If you are using Ubuntu 20.04 the QEMU package must be updated to latest version (>= 1:4.2-3ubuntu6.19).

To check the currently installed version of the qemu-user-static package on Ubuntu 20.04 LTS, use the below command:

```
$ apt list qemu-user-static
```

2.2 Deploy source

Download archive containing the build script and support files for building Debian Bullseye for this board:

```
$ cd ~
$ git clone https://github.com/varigit/debian-var.git -b debian_bullseye_var01
debian_imx6ul-var-dart
```

Create environment (*Internet connection should be available*):

```
$ cd ~/debian_imx6ul-var-dart
$ MACHINE=imx6ul-var-dart ./var_make_debian.sh -c deploy
```

This environment prepared to build.

3 Make Debian

3.1 Build all

Internet connection should be available

```
$ cd ~/debian_imx6ul-var-dart
$ sudo MACHINE=imx6ul-var-dart ./var_make_debian.sh -c all |& tee build.log
```

3.2 Build by parts

3.2.1 Build bootloader

```
$ cd ~/debian_imx6ul-var-dart
$ sudo MACHINE=imx6ul-var-dart ./var_make_debian.sh -c bootloader
```

3.2.2 Build kernel, dtb files and kernel modules

```
$ cd ~/debian_imx6ul-var-dart
$ sudo MACHINE=imx6ul-var-dart ./var_make_debian.sh -c kernel
$ sudo MACHINE=imx6ul-var-dart ./var_make_debian.sh -c modules
```

3.2.3 Build rootfs

Internet connection should be available

```
$ cd ~/debian_imx6ul-var-dart
$ sudo MACHINE=imx6ul-var-dart ./var_make_debian.sh -c rootfs
```

3.2.4 Pack rootfs

To create the root file system archive (rootfs.tar.gz) and UBI image (rootfs.ubi.img), run the following commands:

```
$ cd ~/debian_imx6ul-var-dart
$ sudo MACHINE=imx6ul-var-dart ./var_make_debian.sh -c rtar
$ sudo MACHINE=imx6ul-var-dart ./var_make_debian.sh -c rubi
```

Note: The NAND filesystem is console only and may not have all the features as eMMC/SD card
To fit the NAND UBIFS filesystem as per your need, optimize the UBIFS by removing the packages.

4 Create boot SD card

1. Follow the above steps for make rootfs, kernel, bootloader;
2. Insert the SD card to card reader connected to a host system;
3. Run the following commands (Caution! All data on the card will be destroyed):

```
$ cd ~/debian_imx6ul-var-dart
$ sudo MACHINE=imx6ul-var-dart ./var_make_debian.sh -c sdcard -d /dev/sdX
```

where '/dev/sdX' path to the block SD device in your system.

4.1 Create a boot SD card image using a loop device

It is also possible to use the "MACHINE=imx6ul-var-dart ./var_make_debian.sh" script to create a boot SD card image, while using a loop device instead of attaching a real SD card.

Create an empty file using the following command:

```
$ dd if=/dev/zero of=imx6ul-var-dart-debian-sd.img bs=1M count=3720
```

The above command creates a 3700MiB file representing the SD card.

Attach the first available loop device to this file:

```
$ sudo losetup -Pf imx6ul-var-dart-debian-sd.img
```

To find the actual loop device being used, run: $ losetup -a | grep imx6ul-var-dart-debian-sd.img

Write the content to the loop device to generate the SD card image:

```
$ sudo MACHINE=imx6ul-var-dart ./var_make_debian.sh -c sdcard -d /dev/loopX
```

(Replace /dev/loopX with your actual loop device, e.g. /dev/loop0)

Detach the loop device from the file:

```
$ sudo losetup -d /dev/loopX
```

To compress the SD card image file use the following command:

```
$ gzip -9 imx6ul-var-dart-debian-sd.img
```

To write the SD card image to a real SD card device use the following command:

```
$ zcat imx6ul-var-dart-debian-sd.img.gz | sudo dd of=/dev/sdX bs=1M && sync
```

(Replace /dev/sdX with your actual SD device, e.g. /dev/sdb)

5 Boot the board with a bootable SD card

Note: The WiFi is not operational when booting from SD card, as the WiFi and SD card are using the same SDIO interface.
A typical use-case is to boot from an SD card, flash the eMMC/NAND flash, and re-boot from the eMMC/NAND flash to have the WiFi operational.

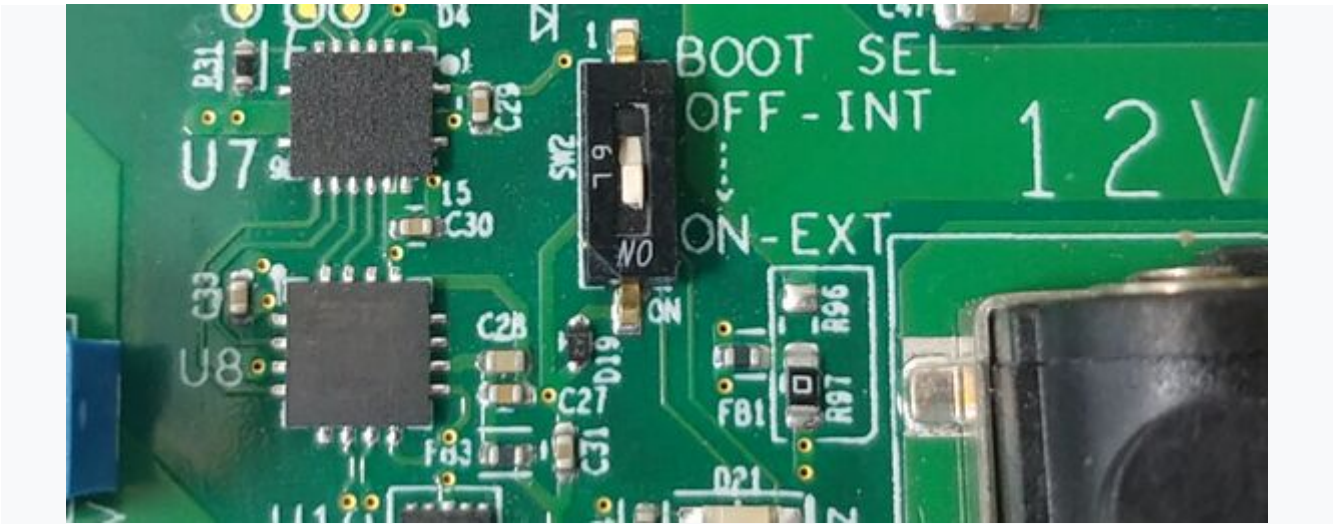
5.1 Setting the Boot Mode

Booting your system from an SD card requires switching the Boot DIP switches. See picture below.
On the VAR-6ULCustomBoard (with a DART-6UL):

- "00" The current position in the picture will set the system to boot from SD card
- "01" Moving the right switch will set the system to boot from eMMC
- "10" Moving the left switch will set the system to boot from NAND flash
- "11" is illegal.

Be aware that your system has eMMC or NAND but never both.

On the Concerto-Board (with a VAR-SOM-6UL):



- ON-EXT: Boot from the external storage (SD card) - the current position in the picture
- OFF-INT: Boot from the SOM's internal storage (eMMC/NAND flash)

5.2 Automatic device tree selection in U-Boot

5.2.1 Enable/Disable Automatic Device Tree selection

To enable the automatic device tree selection in U-Boot (already enabled by default):

```
$ setenv fdt_file undefined
$ saveenv
```

To disable the automatic device tree selection in U-Boot, set the device tree file manually:

```
$ setenv fdt_file YOUR_DTB_FILE
$ saveenv
```

Comment:
Make sure you don't set an inappropriate dtb file, like a dtb with nand on a SOM that has eMMC, or a dtb for mx6ull on a SOM with an mx6ul SOC.

6 Build Results

The resulted images are located in **~/debian_imx6ul-var-dart/output/**.

Image name	How to use
rootfs.tar.gz	Root filesystem tarball used for installation on SD card and eMMC

rootfs.ubi.img	Console only image for installation on NAND flash
zImage	Linux kernel image
SPL.nand	SPL built for NAND. The SPL is pre-U-Boot SW component, required for DDR initialization
SPL.mmc	SPL built for SD card and eMMC boot. The SPL is pre-U-Boot SW component, required for DDR initialization
u-boot.img.nand	U-Boot built for NAND flash
u-boot.img.mmc	U-Boot built for SD card or eMMC
kernel-headers	kernel headers folder for package creation
Device Tree name	**Details**
imx6ull-var-dart-6ulcustomboard-emmc-sd-card.dtb	Device tree blob for DART-6UL with i.MX6ULL SOC, eMMC & SD card enabled. (NAND flash & WiFi disabled)
imx6ull-var-dart-6ulcustomboard-emmc-wifi.dtb	Device tree blob for DART-6UL with i.MX6ULL SOC, eMMC & WiFi enabled. (NAND flash & SD card disabled)
imx6ull-var-dart-6ulcustomboard-nand-sd-card.dtb	Device tree blob for DART-6UL with i.MX6ULL SOC, NAND flash & SD card enabled. (eMMC & WiFi disabled)
imx6ull-var-dart-6ulcustomboard-nand-wifi.dtb	Device tree blob for DART-6UL with i.MX6ULL SOC, NAND flash & WiFi enabled. (eMMC & SD card disabled)
imx6ul-var-dart-6ulcustomboard-emmc-sd-card.dtb	Device tree blob for DART-6UL with i.MX6UL SOC, eMMC & SD card enabled. (NAND flash & WiFi disabled)
imx6ul-var-dart-6ulcustomboard-	Device tree blob for DART-6UL with i.MX6UL SOC, eMMC & WiFi

emmc-wifi.dtb	enabled. (NAND flash & SD card disabled)
imx6ul-var-dart-6ulcustomboard-nand-sd-card.dtb	Device tree blob for DART-6UL with i.MX6UL SOC, NAND flash & SD card enabled. (eMMC & WiFi disabled)
imx6ul-var-dart-6ulcustomboard-nand-wifi.dtb	Device tree blob for DART-6UL with i.MX6UL SOC, NAND flash & WiFi enabled. (eMMC & SD card disabled)
imx6ulz-var-dart-6ulcustomboard-emmc-sd-card.dtb	Device tree blob for DART-6UL with i.MX6ULZ SOC, eMMC & SD card enabled. (NAND flash & WiFi disabled)
imx6ulz-var-dart-6ulcustomboard-emmc-wifi.dtb	Device tree blob for DART-6UL with i.MX6ULZ SOC, eMMC & WiFi enabled. (NAND flash & SD card disabled)
imx6ulz-var-dart-6ulcustomboard-nand-sd-card.dtb	Device tree blob for DART-6UL with i.MX6ULZ SOC, NAND flash & SD card enabled. (eMMC & WiFi disabled)
imx6ulz-var-dart-6ulcustomboard-nand-wifi.dtb	Device tree blob for DART-6UL with i.MX6ULZ SOC, NAND flash & WiFi enabled. (eMMC & SD card disabled)
imx6ull-var-som-concerto-board-emmc-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6ULL SOC, eMMC & SD card enabled. (NAND flash & WiFi disabled)
imx6ull-var-som-concerto-board-emmc-wifi.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6ULL SOC, eMMC & WiFi enabled. (NAND flash & SD card disabled)
imx6ull-var-som-concerto-board-nand-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6ULL SOC, NAND flash & SD card enabled. (eMMC & WiFi disabled)
imx6ull-var-som-concerto-board-nand-wifi.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6ULL SOC, NAND flash & WiFi enabled. (eMMC & SD card disabled)

imx6ul-var-som-concerto-board-emmc-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6UL SOC, eMMC & SD card enabled. (NAND flash & WiFi disabled)
imx6ul-var-som-concerto-board-emmc-wifi.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6UL SOC, eMMC & WiFi enabled. (NAND flash & SD card disabled)
imx6ul-var-som-concerto-board-nand-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6UL SOC, NAND flash & SD card enabled. (eMMC & WiFi disabled)
imx6ul-var-som-concerto-board-nand-wifi.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6UL SOC, NAND flash & WiFi enabled. (eMMC & SD card disabled)
imx6ulz-var-som-concerto-board-emmc-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6ULZ SOC, eMMC & SD card enabled. (NAND flash & WiFi disabled)
imx6ulz-var-som-concerto-board-emmc-wifi.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6ULZ SOC, eMMC & WiFi enabled. (NAND flash & SD card disabled)
imx6ulz-var-som-concerto-board-nand-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6ULZ SOC, NAND flash & SD card enabled. (eMMC & WiFi disabled)
imx6ulz-var-som-concerto-board-nand-wifi.dtb	Device tree blob for VAR-SOM-6UL on Concerto board with i.MX6ULZ SOC, NAND flash & WiFi enabled. (eMMC & SD card disabled)
imx6ull-var-som-symphony-board-emmc-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6ULL SOC, eMMC & SD card enabled. (NAND flash & WiFi disabled)
imx6ull-var-som-symphony-board-emmc-wifi.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6ULL SOC, eMMC & WiFi enabled. (NAND flash & SD card disabled)

imx6ull-var-som-symphony-board-nand-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6ULL SOC, NAND flash & SD card enabled. (eMMC & WiFi disabled)
imx6ull-var-som-symphony-board-nand-wifi.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6ULL SOC, NAND flash & WiFi enabled. (eMMC & SD card disabled)
imx6ul-var-som-symphony-board-emmc-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6UL SOC, eMMC & SD card enabled. (NAND flash & WiFi disabled)
imx6ul-var-som-symphony-board-emmc-wifi.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6UL SOC, eMMC & WiFi enabled. (NAND flash & SD card disabled)
imx6ul-var-som-symphony-board-nand-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6UL SOC, NAND flash & SD card enabled. (eMMC & WiFi disabled)
imx6ul-var-som-symphony-board-nand-wifi.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6UL SOC, NAND flash & WiFi enabled. (eMMC & SD card disabled)
imx6ulz-var-som-symphony-board-emmc-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6ULZ SOC, eMMC & SD card enabled. (NAND flash & WiFi disabled)
imx6ulz-var-som-symphony-board-emmc-wifi.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6ULZ SOC, eMMC & WiFi enabled. (NAND flash & SD card disabled)
imx6ulz-var-som-symphony-board-nand-sd-card.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6ULZ SOC, NAND flash & SD card enabled. (eMMC & WiFi disabled)
imx6ulz-var-som-symphony-board-nand-wifi.dtb	Device tree blob for VAR-SOM-6UL on Symphony board with i.MX6ULZ SOC, NAND flash & WiFi enabled. (eMMC & SD card disabled)

7 Linux console access

User name	User password	User descriptor
root	root	system administrator
user	user	local user
x_user		used for X session access

8 Flash images to NAND / eMMC

In case you are using a SOM with NAND flash, run the following command **as root** to install Debian on it:

```
# install_debian.sh -r nand (Follow instructions)
```

In case you are using a SOM with eMMC, run the following command **as root** to install Debian on it:

```
# install_debian.sh -r emmc (Follow instructions)
```

The above scripts are located in /usr/sbin in the rootfs of the SD card used to boot Debian.

9 How-to: Test and use an interface

Please see this section in the Yocto developer guide page. It is the same for Debian.

10 How-to: Modify the kernel configuration

To modify the kernel configuration (add/remove features and drivers) please follow the steps below:

```
1. $ cd ~/debian_imx6ul-var-dart/src/kernel
2. $ sudo make ARCH=arm mrproper
3. $ sudo make ARCH=arm imx_v7_var_defconfig
4. $ sudo make ARCH=arm menuconfig
5. Navigate the menu and select the desired kernel functionality
6. Exit the menu and answer "Yes" when asked "Do you wish to save your new
configuration?"
7. $ sudo make ARCH=arm savedefconfig
8. $ sudo cp arch/arm/configs/imx_v7_var_defconfig
arch/arm/configs/imx_v7_var_defconfig.orig
9. $ sudo cp defconfig arch/arm/configs/imx_v7_var_defconfig
```

```
10. Follow the instructions above to rebuild kernel and modules, repack rootfs
images and recreate SD card
```

11 How-to: Build and install a custom device tree

To build and install a custom device tree, add the filename to G_LINUX_DTB
in https://github.com/varigit/debian-var/blob/debian_bullseye_var01/variscite/imx6ul-var-dart/imx6ul-var-dart.sh

12 Build a sample C "Hello, world!" program

Create a file called myhello.c with the following content:

```
#include <stdio.h>

int main() {
        printf("Hello, World!\n");
        return 0;
}
```

Export the C (cross-)compiler path:

```
$ export CC=~/debian_imx6ul-var-dart/toolchain/gcc-linaro-6.3.1-2017.05-
x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc
```

Compile:

```
$ $CC myhello.c -o myhello
```

Now you should have an app called myhello, that can be run on your target board.
You can add it to your rootfs image or copy it directly to the rootfs on the board (using scp, for example).

Die Abbildung (2.1.2-1) gibt eine grobe Übersicht von Debian Distribution auf dem Linux Laufwerk. Dabei sind für den UART-SPI Projekt irrelevanten Informationen ausgeblendet. Die projektbezogene Treiber-Systeme (Packege) sind farbig hervorgehoben und die einzelnen Komponenten kurz aufgelistet. Ergänzend sind auch die hardwarespezifische Device Tree Sourcefiles in dem dts-Ordner dargestellt.

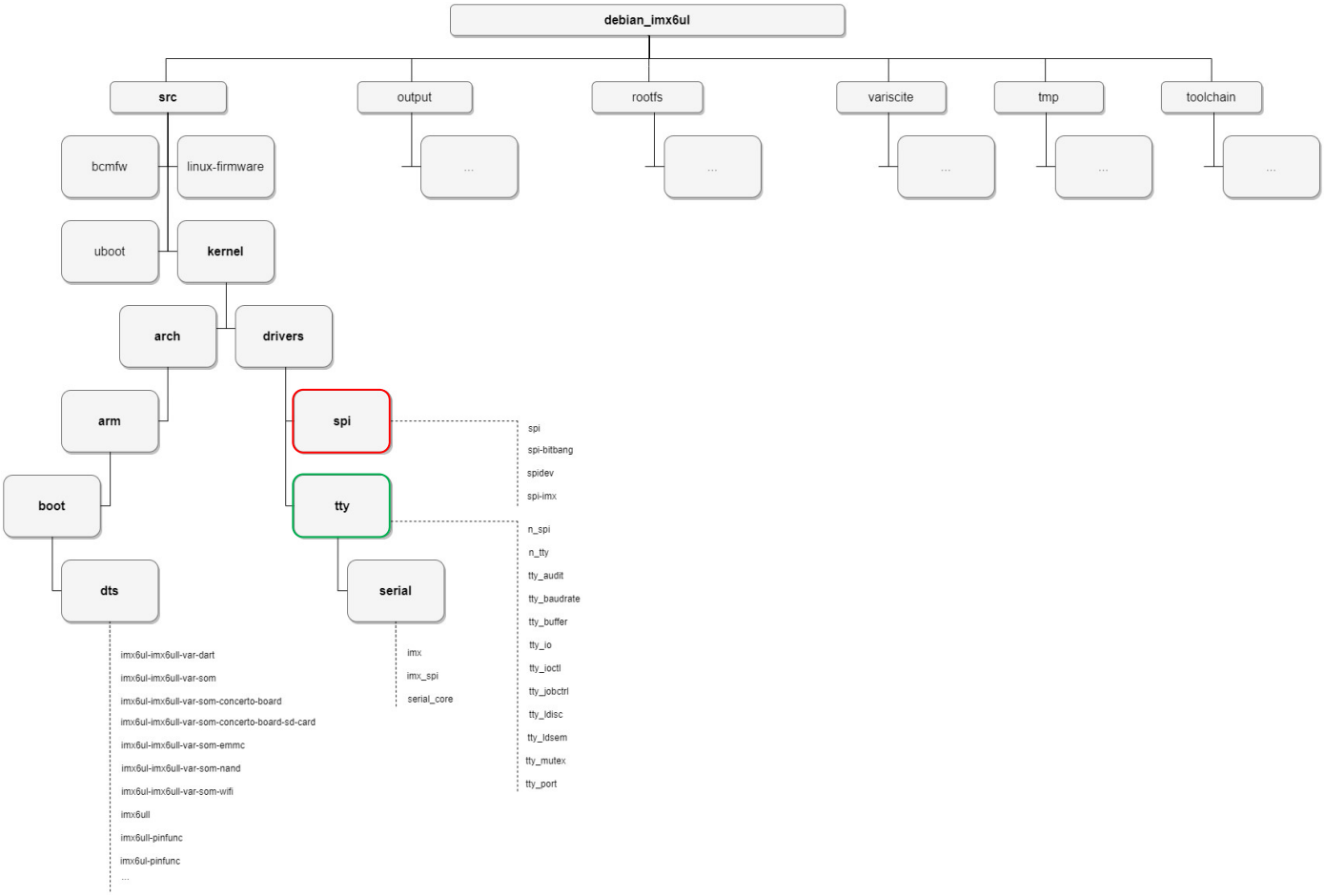


Abbildung 2.1.2-1: Hierarchieebenen der Ordnerstruktur von Debian Distribution.

In dem nächsten Kapitel wird eine detaillierte Einpflanzung vom UART-SPI Projekt in ein bestehendes Treibersystem vorgestellt. Die farbig gekennzeichnete Elemente aus der Abbildung 2.1.2-1 stimmen mit den in der Abbildung 2.1.2-2 präsentierten Komponenten überein.

2.1.3 UART-SPI Integration in Treibersystem. Detaillierte Darstellung.

Die folgende Abbildung (2.1.3-1) gibt die vertiefte Informationen über innere Projektaufbau und erläutert den Zusammenhang zwischen beteiligten Module. Die Pfeile können als Datenfluss zwischen einzelnen Systemteile betrachtet werden.

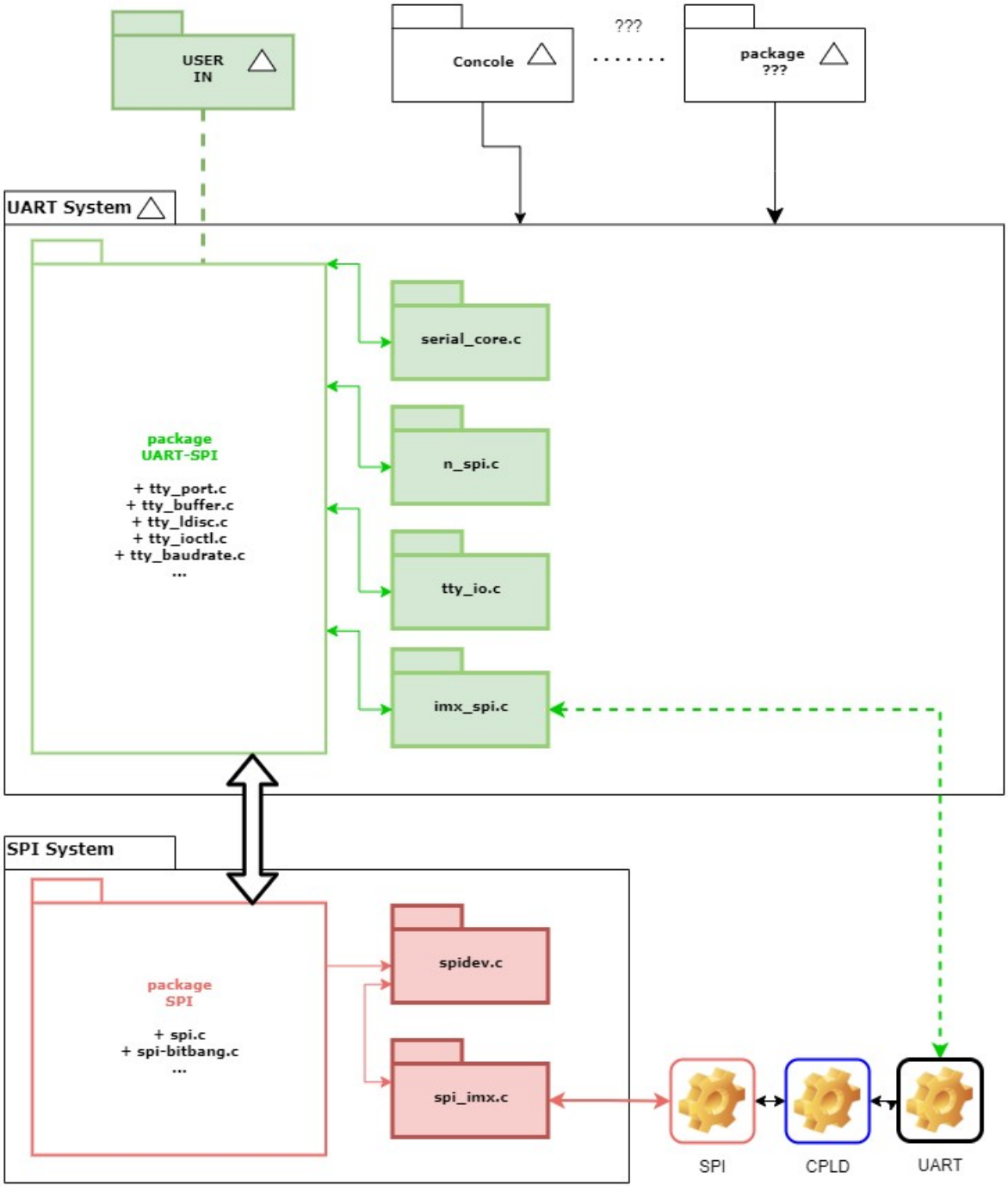


Abbildung 2.1.3-2: UART-SPI Integration in Treibersystem. Detaillierte Darstellung.

2.1.4 UML Sequenzdiagramm für WRITE

Die UML Sequenzdiagramme bieten eine hervorragende Möglichkeit, komplizierte Abläufe zu veranschaulichen und in einer dem Benutzer verständlichen Form zu präsentieren. In dem vorliegenden UART-SPI Projekt sind ein paar grundsätzliche Sequenzen die eine detaillierte Beschreibung verdienen. Das sind zum Beispiel: INIT, OPEN, WRITE und READ. Um den Rahmen der Anleitung nicht unnötig zu vergrößern, werden nur zwei WRITE/READ Vorgänge dargestellt. Die Abbildung 2.1.4-1 zeigt wie ein UART-SPI Treibersystem auf User-Befehl „Write“ reagieren kann. Das erklärt unter anderem, auch wieso mehrere Varianten (siehe dazu „Dokumentation“ Kapitel 2.1.1 bis 2.1.4) bei der Realisierung einer User-Applikation entstehen können. Treiber ist konfigurierbar und flexibel einsatzbereit. Je nach Wunsch/Bedarf/Anforderung kann er synchron bzw. asynchron (siehe dazu „Dokumentation“ Kapitel 2.1) bedient werden.

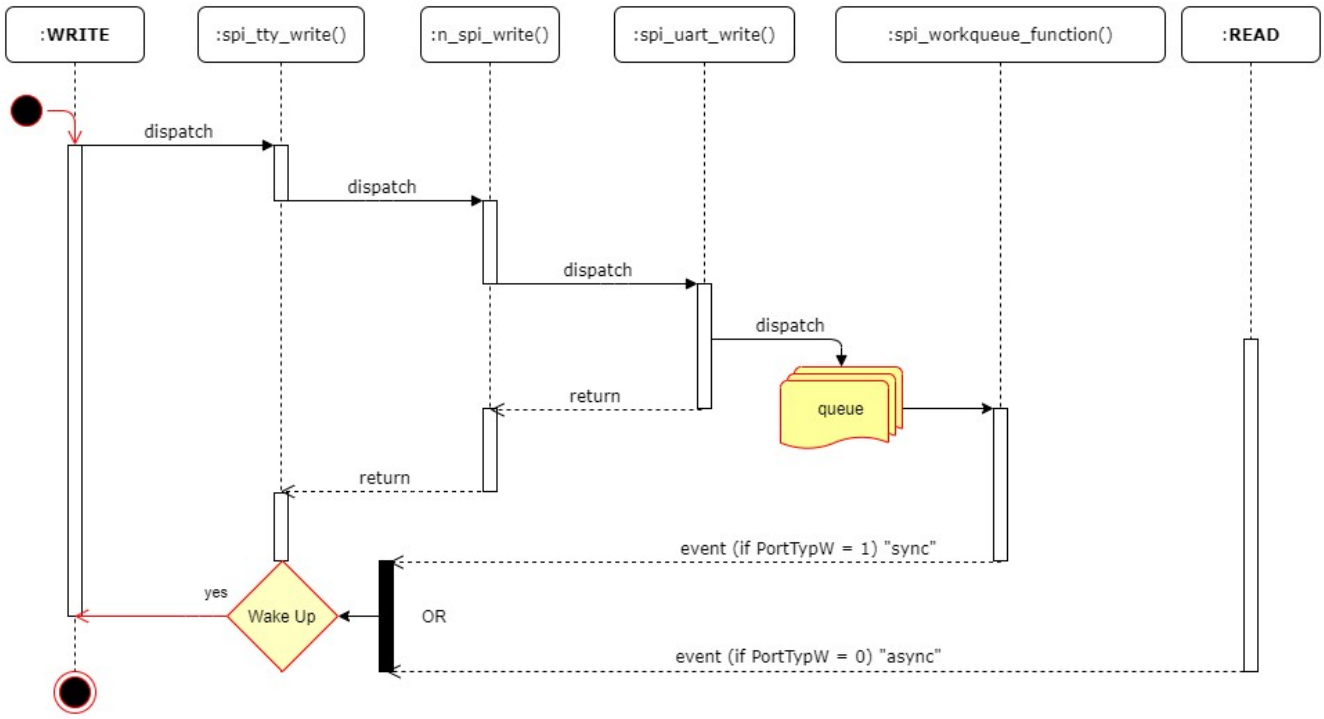


Abbildung 2.1.4-2: UML Sequenzdiagramm für WRITE-Vorgang

2.1.5 UML Sequenzdiagramm für READ

Ein READ-Vorgang ist wesentlich umfangreicher und komplizierter als WRITE-Vorgang, dennoch lässt er sich unter leichter Vereinfachung darstellen. Beachte dabei rot markierte logische Verzweigungen und Auswahlbedingungen.

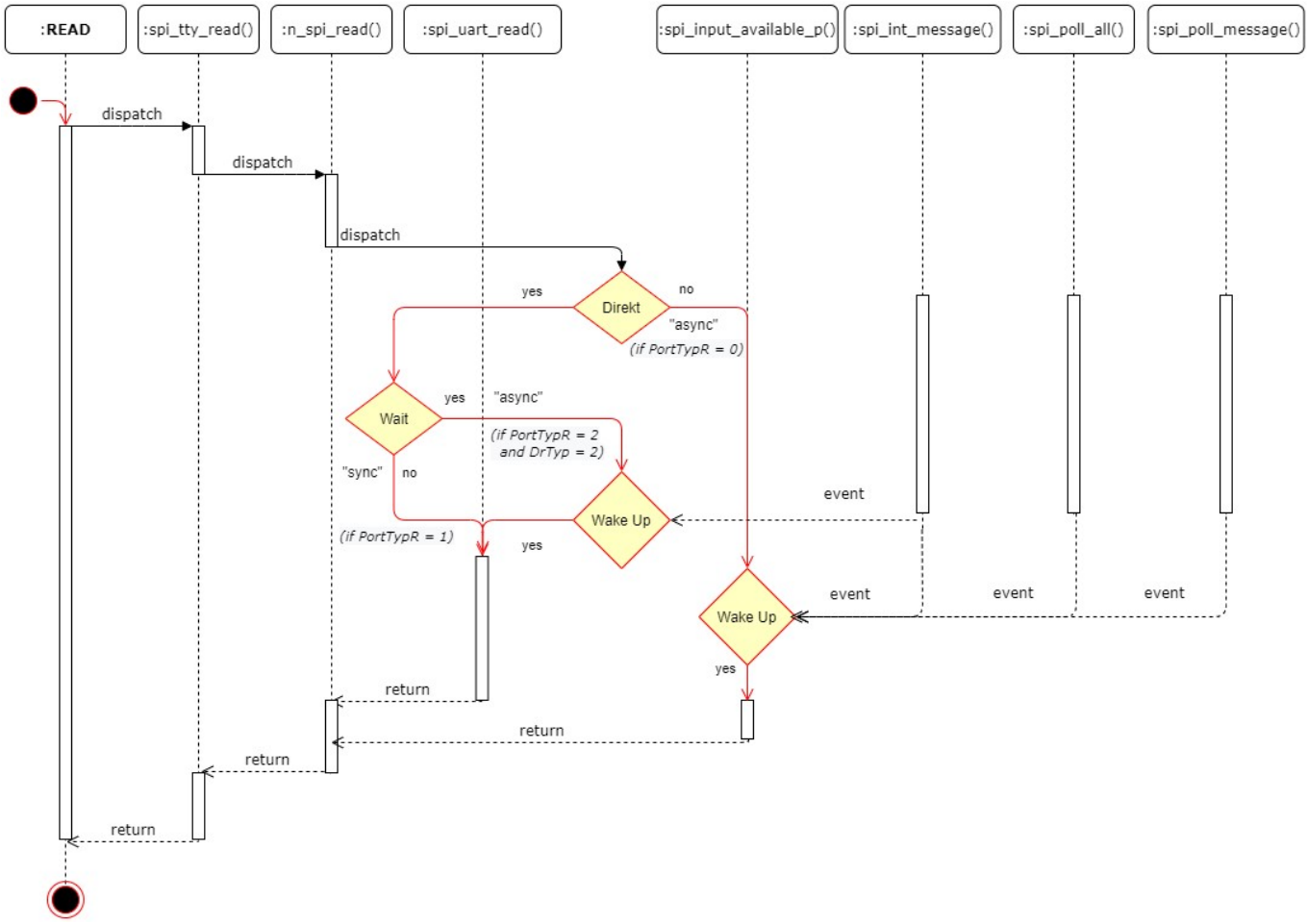


Abbildung 2.1.5-1: UML Sequenzdiagramm für READ-Vorgang

2.2 User Space

Im User Space laufen die Benutzerprogramme, die Treiber nach einem bestimmten Schema ansprechen, konfigurieren und über vordefinierte Schnittstelle bedienen. Diese sogenannte Applikationen richten sich nach den Kundenwünschen aus, sind äußerst flexibel und werden nur durch Treiber-Schnittstelle begrenzt. Die möglichen Varianten sind in der „Dokumentation“ beschrieben. z.B.:

"dezentralisiert" / vollständig asynchron

"Zum Teil dezentralisiert" / asynchron

"zentralisiert“ / synchron

"zentralisiert“ / synchron

"dezentralisiert" / zum Teil asynchron

usw.

Dazu gehören auch die Timing-Test Applikationen.

Ein einfaches Beispiel zum besseren Verständnis:

1-15 Ports WRITE/READ synchrion (Polling in einem File/App). z.B.: Sensoren.

16te Port WRITE/READ asynchron (Interrupt und/oder einzelnen Polling in anderem File/App). z.B.: NOT-Aus Taste

Erklärung: *Erste 15 UART-SPI Ports werden in einem Programm (User-Applikation) geöffnet, konfiguriert, write/read sequenziell (synchron) vom CPU verarbeitet. In einem anderen (asynchron) Programm wird ein einziger Port 16 angesprochen. Die beiden Programme werden von Scheduling zu den unterschiedlichen Zeitpunkten gestartet.*

Treiberkonfiguration

(siehe dazu auch die Bedingungen in den UML Diagrammen)

driver->DrTyp

== 0	Polling Verfahren
== 1	IRQ Verfahren bzw. IRQs sind aktiv
== 2	Polling Verfahren und mit Interrupts gesteuerte Read (SPI_BUS_IDLE_IRQ) wenn port->PortTypR == 2

port->PortTypR

== 0	Port Typ: Normal, SLEEP, Read indirekt/async von asynchronen LinuxBuffer
== 1	Port Typ: Normal, DONT SLEEP, Read direkt/sync
== 2	Port Typ: Gemischt, SLEEP Interrupts gesteuerte Read direkt/async wenn driver->DrTyp == 2

port->PortTypW

== 0	WRITE Wake Up erst nach dem READ.
== 1	WRITE Wake Up gleich nachdem in workqueue abgelegte Nachricht versendet wurde.

Eine Treibers Konfiguration ist generell über IOCTL des Treibers möglich. Ob diese expliziert mit der Verwendung einer speziellen Config-Applikation anhand des Scripts oder direkt am Anfang einer USER-Applikation stattfindet, ist dem Benutzer überlassen. Momentan ist kein User-Interface für benutzerfreundliche Konfiguration des Treibers implementiert. Es können gefertigte Scripts oder Einträge im Quellcode verwendet werden, um eine gewünschte Treibereinstellung zu erreichen.

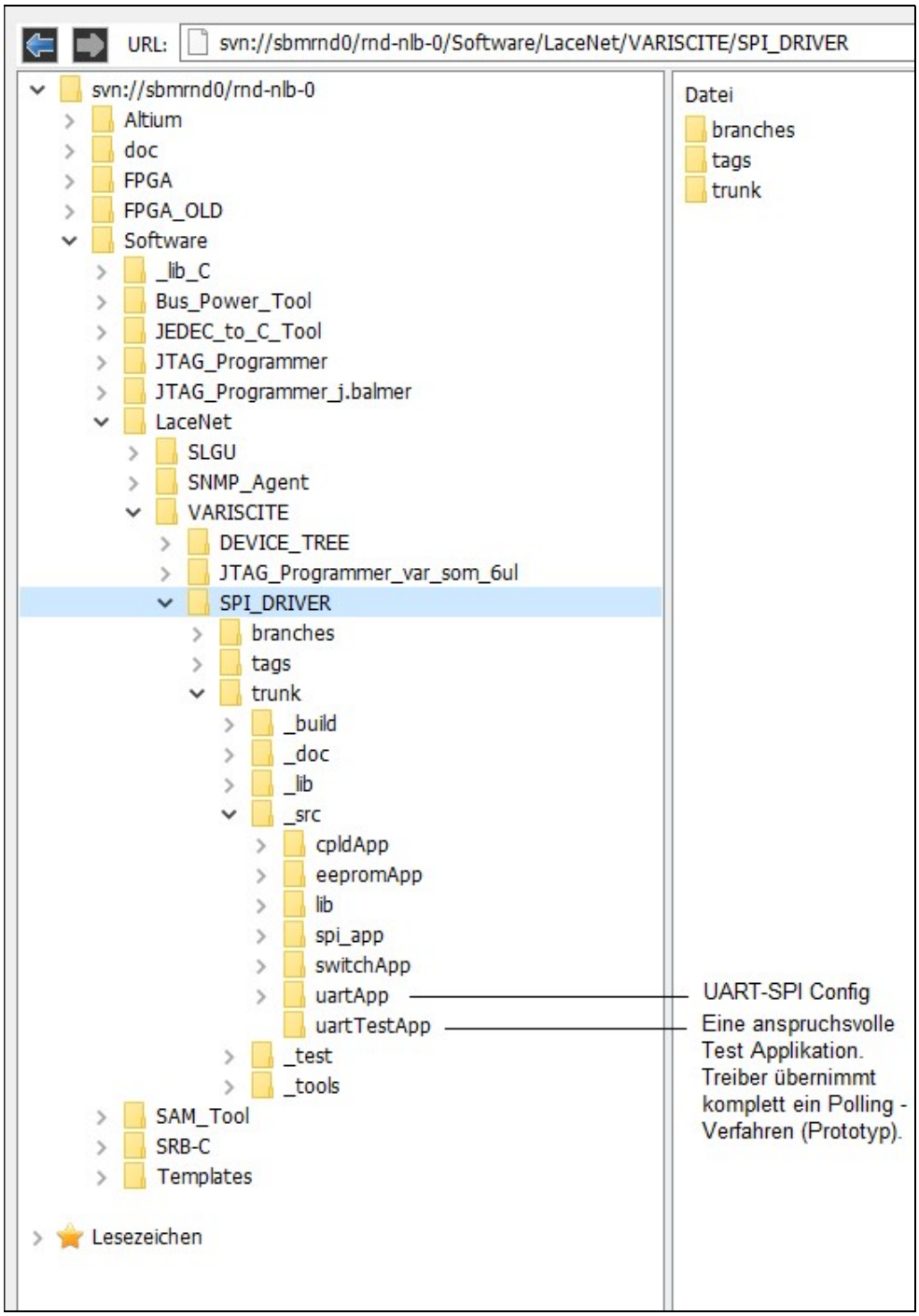


Abbildung 2.2-1: Treiber Konfiguration-Applikation.

Aktuelle SVN Revision Nr.: 1515

3. Timing – Tests Applikationen

In dem Ordner von SVN-Archiv (siehe die Abb. 3-1) namens „user_apps“ sind unterschiedliche Testvarianten gespeichert. In jedem Unterordner sind entsprechende Config – Scripts (start*.c), Quelldateien (*.c) und Binärfiles vorhanden.

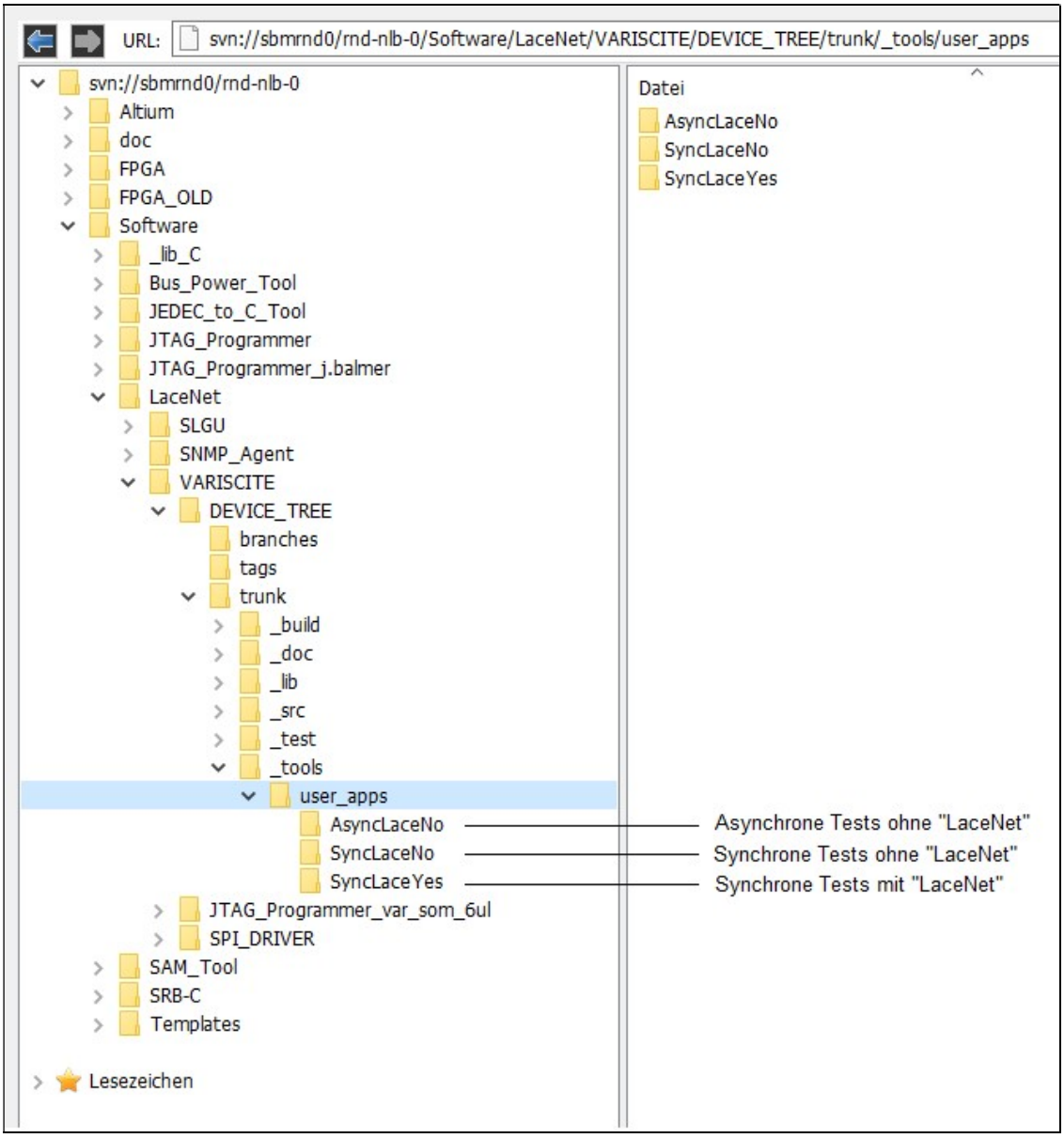


Abbildung 3-1: TortoiseSVN. Applikationen.

Diese Ordner müssen nach dem Auschecken auf den Linux Rechner kopiert werden. Eventuell sollen die Scripts noch angepasst werden, damit die Pfadeingaben dem aktuellen Verzeichnis entsprechen. Die Quellcodedateien können mit „gcc“ oder „cc“ kompiliert werden. Die Binärdateien und Scripts sollen zuerst ausführbar gemacht werden (z.B.: #`chmod a+x datei`). Als erstes soll CPLD konfiguriert werden. Das übernehmen die vordefinierte Scripts (start*.c). Sobald alle Einstellungen getroffen wurden, kann mit den Tests begonnen werden.

Abbildungsverzeichnis