Kernelkompilation

Nachfolgend soll die Möglichkeit beschrieben werden, einen anderen Kernel für das Android Custom ROM zu bauen und zu flashen.

Nützliche Webseiten

Auf folgenden Webseiten wurden Informationen dazu gefunden:

- https://wiki.ubuntuusers.de/Archiv/GNU_ARM-Toolchain/
- https://forum.xda-developers.com/showthread.php?t=2073775
- https://forum.xda-developers.com/android/software/guide-easy-kernel-building-tutorial-t3581057
- https://forum.xda-developers.com/chef-central/android/guide-how-to-build-android-kernel-t3654336
- https://forum.xda-developers.com/android/software-hacking/reference-how-to-compile-android-kernel-t3627297

Voraussetzungen

Es wird ein DEBIAN-Linux-System vorausgesetzt, sowie eine Verbindung zum Internet, um Dateien herunterzuladen. Es kann ein Minimalinstallation als Voraussetzung genutzt werden. Nachfolgend werden alle Schritte in einer CHROOT-Umgebung durchgeführt.

Folgende Pakete müssen zusätzlich noch installiert werden:

```
~# apt install python libssl-dev build-essential libgmp3-dev libmpfr-dev
libx11-6 libx11-dev texinfo flex bison libmpc-dev \
libncurses5 libncurses5-dbg libncurses5-dev libncursesw5 libncursesw5-dbg
libncursesw5-dev zlibc git bc
```

Toolchain

Eine „Toolchain“ ist eine Art Werkzeugkiste, welche Programme beinhaltet, die für die Erstellung des Kernels verwendet werden. Das bekannteste Toolchain kommt von Google selbst: NDK. Aber es gibt auch andere, die zum Beispiel auf der Seite elinux.org kurz beschrieben werden.

DEBIAN-Pakete

Das DEBIAN-System bringt selbst auch Toolchains mit, die über die Paketverwaltung installiert werden können:

- `gcc-arm-linux-gnueabi` (für ARM-Architekturen)
- `gcc-aarch64-linux-gnu` (für ARM64-Architekturen)

Folgende Pakete werden bei der Installation von „`gcc-arm-linux-gnueabi`“ mit installiert:

```
binutils-arm-linux-gnueabi cpp-6-arm-linux-gnueabi cpp-arm-linux-gnueabi
gcc-6-arm-linux-gnueabi \
gcc-6-arm-linux-gnueabi-base gcc-6-cross-base gcc-arm-linux-gnueabi
libasan3-armel-cross libatomic1-armel-cross \
libc6-armel-cross libc6-dev-armel-cross libgcc-6-dev-armel-cross libgcc1-
armel-cross libgomp1-armel-cross \
libstdc++6-armel-cross libubsan0-armel-cross linux-libc-dev-armel-cross
```

Die Umgebungsvariable für den Cross-Kompiler wird dann wie folgt gesetzt:

```
CROSS_COMPILE="/usr/bin/arm-linux-gnueabi-"
```

Google NDK

Das Toolchain kann hier heruntergeladen werden.

Das heruntergeladene Archiv wird dann entpackt:

```
~$ cd toolchain
~$ unzip android-ndk-r17-linux-x86_64.zip
Archive:  android-ndk-r17-linux-x86_64.zip
   creating: android-ndk-r17/
   creating: android-ndk-r17/toolchains/
  inflating: android-ndk-r17/toolchains/NOTICE-MIPS64
   creating: android-ndk-r17/toolchains/x86-4.9/
...
  inflating: android-ndk-r17/python-packages/fastboot/setup.py
  inflating: android-ndk-r17/CHANGELOG.md
  inflating: android-ndk-r17/ndk-build
```

Zur Kompilierung des Kernels wird aber nicht das komplette NDK benötigt. Deswegen kann das Toolchain extra als „`Standalone`“-Variante installiert werden:

```
~$ cd android-ndk-r17
~$ build/tools/make-standalone-toolchain.sh  --install-dir=../google-ndk-r17
HOST_OS=linux
HOST_EXE=
HOST_ARCH=x86_64
HOST_TAG=linux-x86_64
HOST_NUM_CPUS=8
BUILD_NUM_CPUS=16
Auto-config: --arch=arm
Toolchain installed to ../google-ndk-r17.
~$ cd ..
```

Das Verzeichnis „`android-ndk-r17`“ kann jetzt bei Bedarf auch wieder entfernt werden, da es nicht mehr benötigt wird.

Das Toolchain kann wie folgt getestet werden:

```
~$ cd google-ndk-r17/bin
~$ echo "main(){}" | ./arm-linux-androideabi-gcc -x c -
~$ file a.out
a.out: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically
linked, interpreter /system/bin/linker, not stripped
~$ rm a.out
```

Die Umgebungsvariable für den Cross-Kompiler wird dann wie folgt gesetzt:

```
CROSS_COMPILE="$(pwd)/toolchain/google-ndk/bin/arm-linux-androideabi-"
```

Google Prebuilts

Das Toolchain kann hier heruntergeladen werden. Nachfolgend wird das Toolchain „`arm-linux-androideabi-4.9`“ heruntergeladen.

Der Download erfolgt mit „`git`“:

```
~$ git clone
https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/arm/arm-li
nux-androideabi-4.9
```

```
Klone nach 'arm-linux-androideabi-4.9' ...
remote: Sending approximately 262.98 MiB ...
remote: Counting objects: 396, done
remote: Total 2263 (delta 1060), reused 2263 (delta 1060)
Empfange Objekte: 100% (2263/2263), 262.98 MiB | 706.00 KiB/s, Fertig.
Löse Unterschiede auf: 100% (1060/1060), Fertig.
```

Die Umgebungsvariable für den Cross-Kompiler wird dann wie folgt gesetzt:

```
CROSS_COMPILE="$(pwd)/toolchain/arm-linux-androideabi-4.9/bin/arm-linux-
androideabi-"
```

GNU Arm Embedded Toolchain

Das Toolchain kann hier heruntergeladen werden.

Das heruntergeladene Archiv wird dann entpackt:

```
~$ tar xvfj gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2
gcc-arm-none-eabi-7-2017-q4-major/
gcc-arm-none-eabi-7-2017-q4-major/lib/
gcc-arm-none-eabi-7-2017-q4-major/lib/libcc1.so.0.0.0
...
gcc-arm-none-eabi-7-2017-q4-major/bin/arm-none-eabi-cpp
gcc-arm-none-eabi-7-2017-q4-major/bin/arm-none-eabi-gcc-7.2.1
gcc-arm-none-eabi-7-2017-q4-major/bin/arm-none-eabi-nm
```

Zur Vereinfachung wird das neu erstellte Verzeichnis noch umbenannt:

```
~$ mv -v gcc-arm-none-eabi-7-2017-q4-major gnu-arm-7-2017-q4
'gcc-arm-none-eabi-7-2017-q4-major' -> 'gnu-arm-7-2017-q4'
```

Die Umgebungsvariable für den Cross-Kompiler wird dann wie folgt gesetzt:

```
CROSS_COMPILE="$(pwd)/toolchain/gnu-arm-7-2017-q4/bin/arm-none-eabi-"
```

Nathanchance Prebuilt ARM

Dieses Toolchain kommt von xda-developers.com.

Der Download erfolgt über das Klonen:

```
git clone -b arm-gnu-7.x --depth=1
https://github.com/nathanchance/gcc-prebuilts nathanchance-prebuilt-arm
```

Die Umgebungsvariable für den Cross-Kompiler wird dann wie folgt gesetzt:

```
CROSS_COMPILE="$(pwd)/toolchain/nathanchance-prebuilt-arm/bin/arm-gnu-linux-
androideabi-"
```

Linaro ARM

Das Toolchain kann hier heruntergeladen werden.

Das heruntergeladene Archiv wird dann entpackt:

```
~$ tar xvfJ gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf.tar.xz
gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf/
gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf/include/
gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf/include/gdb/
...
gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf/libexec/gcc/arm-linux-
gnueabihf/7.2.1/install-tools/fixincl
gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf/libexec/gcc/arm-linux-
gnueabihf/7.2.1/install-tools/mkheaders
gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf/gcc-
linaro-7.2.1-2017.11-linux-manifest.txt
```

Zur Vereinfachung wird das neu erstellte Verzeichnis noch umbenannt:

```
~$ mv -v gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf linaro-arm
'gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabihf' -> 'linaro-arm'
```

Das Toolchain kann wie folgt getestet werden:

```
~$ cd linaro-arm/bin
~$ echo "main(){}" | ./arm-linux-gnueabihf-gcc -x c -
<stdin>:1:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
~$ file a.out
a.out: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically
linked, interpreter \
/lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0,
BuildID[sha1]=995639f16ef402f3cf87b3b2c59fedb3b0ba8db0, not stripped
~$ rm a.out
```

Die Umgebungsvariable für den Cross-Kompiler wird dann wie folgt gesetzt:

```
CROSS_COMPILE="$(pwd)/toolchain/linaro-arm/bin/arm-linux-gnueabihf-"
```

UBER Toolchain

Dieses Toolchain kommt von xda-developers.com. Die fertigen Archive liegen auf bitbucket.org.

Das gewünschte Archiv wird heruntergeladen:

```
~$ wget
https://bitbucket.org/matthewdalex/arm-linux-androideabi-4.9/get/0ed6f4e24a6
2.zip
```

Im nächsten Schritt muss entpackt werden:

```
~$ unzip 0ed6f4e24a62.zip && rm 0ed6f4e24a62.zip
```

Zur Vereinfachung wird das neu erstellte Verzeichnis noch umbenannt:

```
~$ mv -v matthewdalex-arm-linux-androideabi-4.9-0ed6f4e24a62 ubertc-arm
'matthewdalex-arm-linux-androideabi-4.9-0ed6f4e24a62' -> 'ubertc-arm'
```

Die Umgebungsvariable für den Cross-Kompiler wird dann wie folgt gesetzt:

```
CROSS_COMPILE="$(pwd)/toolchain/ubertc-arm/bin/arm-linux-androideabi-"
```

Linaro AARCH64

Das Toolchain kann hier heruntergeladen werden. Es kann ausschließlich für 64-Bit-Architekturen verwendet werden.

Das heruntergeladene Archiv wird dann entpackt:

```
~$ tar xvfJ gcc-linaro-5.5.0-2017.10-x86_64_aarch64-linux-gnu.tar.xz
gcc-linaro-5.5.0-2017.10-x86_64_aarch64-linux-gnu/
gcc-linaro-5.5.0-2017.10-x86_64_aarch64-linux-gnu/include/
gcc-linaro-5.5.0-2017.10-x86_64_aarch64-linux-gnu/include/gdb/
...
gcc-linaro-5.5.0-2017.10-x86_64_aarch64-linux-gnu/aarch64-linux-
gnu/lib64/libstdc++.so.6.0.21
gcc-linaro-5.5.0-2017.10-x86_64_aarch64-linux-gnu/aarch64-linux-
gnu/lib64/libstdc++.a
gcc-linaro-5.5.0-2017.10-x86_64_aarch64-linux-gnu/aarch64-linux-
gnu/lib64/libasan_preinit.o
```

Zur Vereinfachung wird das neu erstellte Verzeichnis noch umbenannt:

```
~$ mv -v gcc-linaro-5.5.0-2017.10-x86_64_aarch64-linux-gnu linaro-aarch64
'gcc-linaro-5.5.0-2017.10-x86_64_aarch64-linux-gnu' -> 'linaro-aarch64'
```

Das Toolchain kann wie folgt getestet werden:

```
~$ cd linaro-aarch64/bin
~$ echo "main(){}" | ./aarch64-linux-gnu-gcc -x c -
<stdin>:1:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
~$ file a.out
a.out: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically
linked, interpreter \
/lib/ld-linux-aarch64.so.1, for GNU/Linux 3.7.0,
BuildID[sha1]=e3733727fdcae630f517bc32fe62aaf1fa074380, not stripped
~$ rm a.out
```

Die Umgebungsvariable für den Cross-Kompiler wird dann wie folgt gesetzt:

```
CROSS_COMPILE="$(pwd)/toolchain/linaro-aarch64/bin/aarch64-linux-gnu-"
```

Architektur

Jetzt müüsen noch die Variablen für die Architektur gesetzt werden:

```
~$ export ARCH=arm
~$ export SUBARCH=arm
```

Hier muss bei Bedarf eine andere Architektur (zum Beispiel „arm64“) verwendet werden.

Kernelquelle

Der Kernel für ein bereits angepasstes Gerät kann zum Beispiel von Github heruntergeladen werden. Wichtig ist es hier, auf den richtigen Chipsatz des Gerätes zu achten. Nachfolgend wird ein anderer Kernel für das Gerät Samsung Galaxy S5 (SM-G900F) gebaut, welches einen Qualcomm-Chipsatz besitzt.

Im ersten Schritt wird das Repository auf den lokalen Rechner geklont:

```
~$ git clone -b cm-14.1
https://github.com/LineageOS/android_kernel_samsung_msm8974.git lineageos-
samsung-msm8974
```

Bauvorgang

Die Kernelkonfiguration ist im neu erstellten Verzeichnis „lineageos-samsung-msm8974/“ unter „arch/arm/configs/“ zu finden, nachfolgend wird die Datei „lineage_klte_bcm2079x_defconfig“ genutzt. Auch diese muss zum eigenen Gerät und zu Architektur passen.

Jetzt erfolgt der Wechsel in das neu erstellte Verzeichnis und das Bauen kann durchgeführt werden:

```
~$ cd lineageos-samsung-msm8974
~$ make clean
#### make completed successfully (2 seconds) ####
```

```
~$ make mrproper
#### make completed successfully (4 seconds) ####

~$ make lineage_klte_bcm2079x_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
sound/soc/codecs/audience/Kconfig:40:warning: type of 'SND_SOC_ES_SLIM'
redefined from 'boolean' to 'tristate'
sound/soc/codecs/audience/Kconfig:43:warning: type of 'SND_SOC_ES_I2C'
redefined from 'boolean' to 'tristate'
boolean symbol SND_SOC_MAX98506 tested for 'm'? test forced to 'n'
sound/soc/codecs/audience/Kconfig:44:warning: choice value used outside its
choice group
sound/soc/codecs/audience/Kconfig:41:warning: choice value used outside its
choice group
#
# configuration written to .config
#

#### make completed successfully (2 seconds) ####

~$ make -s -j$(nproc --all)
sound/soc/codecs/audience/Kconfig:40:warning: type of 'SND_SOC_ES_SLIM'
redefined from 'boolean' to 'tristate'
sound/soc/codecs/audience/Kconfig:43:warning: type of 'SND_SOC_ES_I2C'
redefined from 'boolean' to 'tristate'
boolean symbol SND_SOC_MAX98506 tested for 'm'? test forced to 'n'
...
HMAC-SHA256(builtime_bytes.bin)=
183c03ebecd2d7d347b1eacccfa290e85af99879e9f93349e96131e2810e8196
  Kernel: arch/arm/boot/Image is ready
  Kernel: arch/arm/boot/zImage is ready
  Kernel: arch/arm/boot/zImage-dtb is ready

#### make completed successfully (19 seconds) ####
```

Fehlerbehandlung

Es kann vorkommen, dass die Kompilierung mit einer Fehlermeldung abbricht. Hier muss dann von Fall zu Fall geprüft werden, welche Kerneleinstellungen geändert werden müssen. Nachfolgend werden ein paar Beispiele gezeigt.

net/netfilter/xt_TCPMSS.o

Die Kompilierung bricht mit folgender Meldung ab:

```
  Generating include/generated/mach-types.h
make[2]: *** Keine Regel vorhanden, um das Ziel „net/netfilter/xt_TCPMSS.o“,
  benötigt von „net/netfilter/built-in.o“, zu erstellen.  Schluss.
make[2]: *** Es wird auf noch nicht beendete Prozesse gewartet...
scripts/Makefile.build:443: die Regel für Ziel „net/netfilter“ scheiterte
make[1]: *** [net/netfilter] Fehler 2
make[1]: *** Es wird auf noch nicht beendete Prozesse gewartet...
Makefile:961: die Regel für Ziel „net“ scheiterte
make: *** [net] Fehler 2
```

Aufruf der Kernelkonfiguration:

```
~$ make menuconfig
```

Hier dann „Load an Alternate Configuration File“ auswählen und folgendes eingeben:

```
arch/arm/configs/lineage_klte_bcm2079x_defconfig
```

Hier muss dann unter Umständen die selbst gewählte Kernelkonfiguration eingetragen werden.

Dann zu folgenden Konfigurationspunkt wechseln:

- [*] Networking support –>
  - Networking options –>
    - [*] Network packet filtering framework (Netfilter) –>
      - Core Netfilter Configuration –>
        - [ ] „TCPMSS“ target support **(den Stern hier bitte rausnehmen)**

Beim Beenden des Konfigurationsmenüs muss die Datei gespeichert werden.

Flashen des Kernels

Nach einem erfolgreichen Bauvorgang entsteht die Kerneldatei namens „zImage“.

```
Kernel: arch/arm/boot/zImage is ready
```

Damit diese Datei auf dem Telefon funktioniert, muss sie in das Boot-Image integriert werden („`boot.img`“). Die Boot-Image-Datei entsteht unter anderem als Nebenprodukt, wenn ein angepasstes Image (zum Beispiel LineageOS) gebaut wird. Dem Author sind jetzt zwei Möglichkeiten bekannt, das Kernel-Image in das Boot-Image zu integrieren, wobei auch nur eine funktioniert hat: Android-Image-Kitchen.

Die Dateien können mit Hilfe von „`git`“ auf den lokalen Rechner geklont werden:

```
~$ git clone -b AIK-Linux
https://github.com/osm0sis/Android-Image-Kitchen.git
```

Die enthaltenen Skripte sind so erstellt wurden, dass alles innerhalb des neu erstellten Verzeichnisses „`Android-Image-Kitchen`“ abläuft:

```
~$ cd Android-Image-Kitchen
```

Im ersten Schritt muss das Boot-Image entpackt werden:

```
~$  ./unpackimg.sh boot.img

Android Image Kitchen - UnpackImg Script
by osm0sis @ xda-developers

Supplied image: boot.img

Setting up work folders...

Image type: AOSP

Footer with "SEAndroid" type detected.

Splitting image to "split_img/"...
BOARD_KERNEL_CMDLINE console=null androidboot.hardware=qcom user_debug=31
msm_rtb.filter=0x37 ehci-hcd.park=3 \
zcache.enabled=1 zcache.compressor=lz4 androidboot.bootdevice=msm_sdcc.1
buildvariant=userdebug
BOARD_KERNEL_BASE 00000000
BOARD_NAME
BOARD_PAGE_SIZE 2048
BOARD_HASH_TYPE sha1
BOARD_KERNEL_OFFSET 00008000
BOARD_RAMDISK_OFFSET 02000000
BOARD_SECOND_OFFSET 00f00000
BOARD_TAGS_OFFSET 01e00000
BOARD_OS_VERSION 7.1.2
BOARD_OS_PATCH_LEVEL 2018-04
```

```
BOARD_DT_SIZE 1843200

Unpacking ramdisk (as root) to "ramdisk/"...

Compression used: gzip
5851 blocks

Done!
```

Es wurden zwei neue Verzeichnisse „`ramdisk`“ und „`split_img`“ angelegt. Im zweiten befindet sich unser Kernel-Image, welches ausgetauscht werden soll:

```
~$ cp -v zImage split_img/boot.img-zImage
'zImage' -> 'split_img/boot.img-zImage'
```

Im letzten Schritt muss das „neue“ Boot-Image wieder zusammengebaut werden:

```
~$ ./repackimg.sh

Android Image Kitchen - RepackImg Script
by osm0sis @ xda-developers

Packing ramdisk (as root)...

Using compression: gzip

Getting build information...
kernel = boot.img-zImage
cmdline = console=null androidboot.hardware=qcom user_debug=31
msm_rtb.filter=0x37 ehci-hcd.park=3 \
zcache.enabled=1 zcache.compressor=lz4 androidboot.bootdevice=msm_sdcc.1
buildvariant=userdebug
board =
base = 00000000
pagesize = 2048
kernel_offset = 00008000
ramdisk_offset = 02000000
second_offset = 00f00000
tags_offset = 01e00000
os_version = 7.1.2
os_patch_level = 2018-04
hash = sha1
dtb = boot.img-dtb

Building image...

Using format: AOSP
```

```
Appending footer...

Using type: SEAndroid

Done!
```

Als Ergebnis wird die Datei „`image-new.img`“ erstellt, welche auf das Telefon kopiert und dort (zum Beispiel mit Hilfe vom TWRP) installiert werden kann.

Bauversuche SM-G900F

Nachfolgend sollen Bauversuche von Kompilierungen des Kernels für das Samsung Galaxy S5 (SM-G900F) beschrieben werden.

OK: CyanogenMod

Der Kernel kann von der Seite github.com heruntergeladen werden und kann für ein angepasstes Image verwendet werden.

Download

Der Download der Quellen mit „`git`“:

```
~$ git clone -b cm-14.1
https://github.com/CyanogenMod/android_kernel_samsung_klte.git
github_cyanogenmod_android-kernel-samsung-klte
```

Toolchain

Verlinken der Toolchain:

```
~$ cd github_cyanogenmod_android-kernel-samsung-klte
~$ ln -sf ../toolchain toolchain
```

Bauskript

Bevor das Bauen des Kernel mit Hilfe des mitgelieferten Skriptes „`build_kernel.sh`“ gestartet werden kann, muss dieses noch angepasst werden:

```
export CROSS_COMPILE=$(pwd)/toolchain/arm-linux-androideabi-4.9/bin/arm-
linux-androideabi-
mkdir -p output
...
if [ -e output/arch/arm/boot/Image ]; then
    cp output/arch/arm/boot/Image $(pwd)/arch/arm/boot/zImage
fi;
```

Jetzt kann das Skript ausgeführt werden:

```
~$ bash ./build_kernel.sh
make: Verzeichnis „/data/AndroidKernelBuild/github_cyanogenmod_android-
kernel-samsung-klte“ wird betreten
  HOSTCC  scripts/basic/fixdep
  GEN     /data/AndroidKernelBuild/github_cyanogenmod_android-kernel-
samsung-klte/output/Makefile
  HOSTCC  scripts/kconfig/conf.o
...
make: Verzeichnis „/data/AndroidKernelBuild/github_cyanogenmod_android-
kernel-samsung-klte“ wird verlassen
make: Verzeichnis „/data/AndroidKernelBuild/github_cyanogenmod_android-
kernel-samsung-klte“ wird betreten
  GEN     /data/AndroidKernelBuild/github_cyanogenmod_android-kernel-
samsung-klte/output/Makefile
scripts/kconfig/conf --silentoldconfig Kconfig
...
HMAC-SHA256(builtime_bytes.bin)=
6f415adf8ca4294d36522ca6d1db222bf8e0ec9db6006c60244cad843bed45cc
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  AS      arch/arm/boot/compressed/head.o
  XZKERN  arch/arm/boot/compressed/piggy.xzkern
  CC      arch/arm/boot/compressed/misc.o
  CC      arch/arm/boot/compressed/decompress.o
  CC      arch/arm/boot/compressed/string.o
  SHIPPED arch/arm/boot/compressed/lib1funcs.S
  SHIPPED arch/arm/boot/compressed/ashldi3.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  AS      arch/arm/boot/compressed/ashldi3.o
  AS      arch/arm/boot/compressed/piggy.xzkern.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
```

```
  CAT     arch/arm/boot/zImage-dtb
  Kernel: arch/arm/boot/zImage-dtb is ready
make: Verzeichnis „/data/AndroidKernelBuild/github_cyanogenmod_android-
kernel-samsung-klte“ wird verlassen
```

Manueller Bauversuch

Vor dem erneuten Bauen sollten die Altlasten aufgeräumt werden:

```
~$ make clean && make mrproper
```

Jetzt der manuelle Bauversuch:

```
~$ make msm8974_sec_defconfig
VARIANT_DEFCONFIG=msm8974pro_sec_klte_eur_defconfig
SELINUX_DEFCONFIG=selinux_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
...
arch/arm/configs/msm8974pro_sec_klte_eur_defconfig:213:warning: override:
reassigning to symbol NET_SCHED
arch/arm/configs/msm8974pro_sec_klte_eur_defconfig:216:warning: override:
reassigning to symbol NET_CLS_ACT
arch/arm/configs/msm8974pro_sec_klte_eur_defconfig:218:warning: override:
reassigning to symbol NET_EMATCH
KCONFIG_DEBUG((null))
#
# configuration written to .config
#

~$ make -j$(nproc --all)
```

Fehler "smd_init_dt.c"

```
  CC      kernel/sysctl_binary.o
  CC      mm/fadvise.o
arch/arm/mach-msm/smd_init_dt.c:24:25: fatal error: smd_private.h: No such
file or directory
 #include <smd_private.h>
                         ^
compilation terminated.
scripts/Makefile.build:307: die Regel für Ziel „arch/arm/mach-
```

```
msm/smd_init_dt.o“ scheiterte
make[1]: *** [arch/arm/mach-msm/smd_init_dt.o] Fehler 1
make[1]: *** Es wird auf noch nicht beendete Prozesse gewartet...
  CC      mm/maccess.o
  CC      ipc/msgutil.o
```

Eine Recherche im Internet ergibt, dass in der Datei „`arch/arm/mach-msm/smd_init_dt.c`“ die Zeile 24: „`#include <smd_private.h>`“ in „`#include "smd_private.h"`“ geändert werden muss.

Fehler "rtac.c"

```
  CC      fs/f2fs/node.o
  LD      drivers/clk/built-in.o
sound/soc/msm/qdsp6v2/rtac.c:29:21: fatal error: q6voice.h: No such file or
directory
 #include <q6voice.h>
                     ^
compilation terminated.
scripts/Makefile.build:307: die Regel für Ziel
„sound/soc/msm/qdsp6v2/rtac.o“ scheiterte
make[4]: *** [sound/soc/msm/qdsp6v2/rtac.o] Fehler 1
make[4]: *** Es wird auf noch nicht beendete Prozesse gewartet...
  LD      drivers/clocksource/built-in.o
  CC      drivers/cpufreq/cpufreq.o
```

Die Reparatur ist ähnlich des vorherigen Fehlers. In der Datei „`sound/soc/msm/qdsp6v2/rtac.c`“ die Zeile 29: „`#include <q6voice.h>`“ in „`#include "q6voice.h"`“ ändern.

Fehler "mdss_mdp_trace.h"

```
  LD      drivers/usb/core/built-in.o
  CC      drivers/usb/gadget/udc-core.o
In file included from drivers/video/msm/mdss/mdss_mdp_trace.h:255:0,
                 from drivers/video/msm/mdss/mdss_mdp.c:61:
include/trace/define_trace.h:79:43: fatal error: ./mdss_mdp_trace.h: No such
file or directory
 #include TRACE_INCLUDE(TRACE_INCLUDE_FILE)
                                           ^
compilation terminated.
scripts/Makefile.build:307: die Regel für Ziel
```

```
„drivers/video/msm/mdss/mdss_mdp.o“ scheiterte
make[4]: *** [drivers/video/msm/mdss/mdss_mdp.o] Fehler 1
scripts/Makefile.build:443: die Regel für Ziel „drivers/video/msm/mdss“
scheiterte
make[3]: *** [drivers/video/msm/mdss] Fehler 2
scripts/Makefile.build:443: die Regel für Ziel „drivers/video/msm“
scheiterte
make[2]: *** [drivers/video/msm] Fehler 2
scripts/Makefile.build:443: die Regel für Ziel „drivers/video“ scheiterte
make[1]: *** [drivers/video] Fehler 2
make[1]: *** Es wird auf noch nicht beendete Prozesse gewartet...
  CC      drivers/usb/gadget/android.o
  CC      net/ipv4/netfilter/nf_nat_ftp.o
```

Hier wird die Datei „drivers/video/msm/mdss/mdss_mdp_trace.h“ im Verzeichnis „include/trace/“ von der Datei „define_trace.h“ gesucht. Damit dies funktioniert, kann eine symbolische Verknüpfung erstellt werden:

```
~$ cd include/trace/
~$ ln -s ../../drivers/video/msm/mdss/mdss_mdp_trace.h
~$ cd -
```

Fehler "mdss_mdp.h"

```
  CC      drivers/usb/dwc3/debugfs.o
  CC      drivers/usb/dwc3/dwc3-msm.o
In file included from include/trace/define_trace.h:79:0,
                 from drivers/video/msm/mdss/mdss_mdp_trace.h:255,
                 from drivers/video/msm/mdss/mdss_mdp.c:61:
include/trace/./mdss_mdp_trace.h:25:22: fatal error: mdss_mdp.h: No such
file or directory
 #include "mdss_mdp.h"
                      ^
compilation terminated.
scripts/Makefile.build:307: die Regel für Ziel
„drivers/video/msm/mdss/mdss_mdp.o“ scheiterte
make[4]: *** [drivers/video/msm/mdss/mdss_mdp.o] Fehler 1
make[4]: *** Es wird auf noch nicht beendete Prozesse gewartet...
  CC      drivers/usb/gadget/udc-core.o
  CC      drivers/usb/gadget/android.o
```

Dieser Fehler ist wieder ähnlich dem vorherigen Fehler. eine symbolische Verknüpfung der Datei „./drivers/video/msm/mdss/mdss_mdp.h“ schafft Abhilfe:

```
~$ cd include/trace/
~$ ln -s ../../drivers/video/msm/mdss/mdss_mdp.h
~$ cd -
```

Jetzt kann der Kernel erfolgreich gebaut werden:

```
HMAC-SHA256(builtime_bytes.bin)=
98871c7ee747591b987d4b11ec74fdfc316e1a0fe159a2dabe85c6ae958a3c87
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  AS      arch/arm/boot/compressed/head.o
  XZKERN  arch/arm/boot/compressed/piggy.xzkern
  CC      arch/arm/boot/compressed/misc.o
  CC      arch/arm/boot/compressed/decompress.o
  CC      arch/arm/boot/compressed/string.o
  SHIPPED arch/arm/boot/compressed/lib1funcs.S
  SHIPPED arch/arm/boot/compressed/ashldi3.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  AS      arch/arm/boot/compressed/ashldi3.o
  AS      arch/arm/boot/compressed/piggy.xzkern.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  CAT     arch/arm/boot/zImage-dtb
  Kernel: arch/arm/boot/zImage-dtb is ready
```

FEHLER: Boeffla-Venom-Kernel

Der Kernel wurde auf der Seite von xda-developers.com gefunden. Die Quellen befinden sich auf der Seite von github.com.

Download

Der Download der Quellen mit „`git`“:

```
~$ git clone -b cm-14.1
https://github.com/TheSkater187/android_kernel_samsung_msm8974
github_theskater187_android-kernel-samsung-klte
```

Toolchain

Verlinken der Toolchain:

```
~$ cd github_theskater187_android-kernel-samsung-klte
~$ ln -sf ../toolchain toolchain
```

Bauskript

Damit die Toolchain gefunden wird, muss das Skript „`build_kernel.sh`“ angepasst werden:

```
export CROSS_COMPILE=$(pwd)/toolchain/arm-linux-androideabi-4.9/bin/arm-
linux-androideabi-
mkdir -p output
...
if [ -e output/arch/arm/boot/Image ]; then
    cp output/arch/arm/boot/Image $(pwd)/arch/arm/boot/zImage
fi;
```

Jetzt kann das Skript ausgeführt werden:

```
~$ bash ./build_kernel.sh
make: Verzeichnis „/data/AndroidKernelBuild/github_theskater187_android-
kernel-samsung-klte“ wird betreten
  HOSTCC  scripts/basic/fixdep
  GEN     /data/AndroidKernelBuild/github_theskater187_android-kernel-
samsung-klte/output/Makefile
  HOSTCC  scripts/kconfig/conf.o
...
arch/arm/mach-msm/built-in.o:msm8974-thermistor.c:function msm8974_init:
error: undefined reference to 'msm_8974_init_gpiomux'
arch/arm/mach-msm/built-in.o:msm8974-thermistor.c:function pil_boot: error:
undefined reference to 'poweroff_charging'
arch/arm/mach-msm/built-in.o:msm8974-thermistor.c:function msm_restart:
error: undefined reference to 'poweroff_charging'
drivers/built-in.o:sii8240.c:function of_sii8240_probe_dt: error: undefined
reference to 'acc_register_notifier'
/data/AndroidKernelBuild/github_theskater187_android-kernel-samsung-
klte/Makefile:889: die Regel für Ziel „.tmp_vmlinux1“ scheiterte
make[1]: *** [.tmp_vmlinux1] Fehler 1
Makefile:130: die Regel für Ziel „sub-make“ scheiterte
make: *** [sub-make] Fehler 2
make: Verzeichnis „/data/AndroidKernelBuild/github_theskater187_android-
kernel-samsung-klte“ wird verlassen
```

Bisher wurde für diesen Fehler noch keine Lösung gefunden.

Manueller Bauversuch

Vor dem erneuten Bauen sollten die Altlasten aufgeräumt werden:

```
~$ make clean && make mrproper
```

Jetzt der manuelle Bauversuch:

```
make msm8974_sec_defconfig
VARIANT_DEFCONFIG=msm8974pro_sec_klte_eur_defconfig
SELINUX_DEFCONFIG=selinux_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
...
arch/arm/configs/msm8974_sec_defconfig:434:warning: override: reassigning to
symbol HID_ELECOM
arch/arm/configs/msm8974_sec_defconfig:612:warning: override: reassigning to
symbol KEYS
#
# configuration written to .config
#

~$ make -j$(nproc --all)
```

Es tritt der gleiche Fehler auf.

OK: CrazySuperKernel

Der Kernel wurde auf der Seite von github.com gefunden.

Download

Der Download der Quellen mit „git“:

```
~$ git clone
https://github.com/TheSkater187/CrazySuperKernel-CM14.1-KLTE-New-rebase.git
github_theskater187_crazysuperkernel
```

Toolchain

Verlinken der Toolchain:

```
~$ cd github_theskater187_crazysuperkernel
~$ ln -sf ../toolchain toolchain
```

Bauskript

Damit die Toolchain gefunden wird, muss das Skript „build_kernel.sh“ angepasst werden:

```
export CROSS_COMPILE=$(pwd)/toolchain/arm-linux-androideabi-4.9/bin/arm-
linux-androideabi-
mkdir -p output
...
if [ -e output/arch/arm/boot/Image ]; then
    cp output/arch/arm/boot/Image $(pwd)/arch/arm/boot/zImage
fi;
```

Jetzt kann das Skript ausgeführt werden:

```
make: Verzeichnis
„/data/AndroidKernelBuild/github_theskater187_crazysuperkernel“ wird
betreten
  HOSTCC  scripts/basic/fixdep
  GEN
/data/AndroidKernelBuild/github_theskater187_crazysuperkernel/output/Makefil
e
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
...
make: Verzeichnis
„/data/AndroidKernelBuild/github_theskater187_crazysuperkernel“ wird
verlassen
```

```
make: Verzeichnis
„/data/AndroidKernelBuild/github_theskater187_crazysuperkernel“ wird
betreten
  GEN
/data/AndroidKernelBuild/github_theskater187_crazysuperkernel/output/Makefil
e
scripts/kconfig/conf --silentoldconfig Kconfig
...
  DTC     arch/arm/boot/msm8974pro-ac-sec-k-r14.dtb
  CAT     arch/arm/boot/zImage-dtb
  Kernel: arch/arm/boot/zImage-dtb is ready
make[2]: Für das Ziel „arch/arm/boot/dtbs“ ist nichts zu tun.
make: Verzeichnis
„/data/AndroidKernelBuild/github_theskater187_crazysuperkernel“ wird
verlassen
```

OK: Boeffla-Kernel

Der Kernel wurde auf der Seite von xda-developers.com gefunden. Die Quellen befinden sich auf github.com.

Download

Der Download der Quellen mit „`git`“:

```
~$ git clone -b boeffla_cm14
https://github.com/andip71/boeffla-kernel-cm-s5.git github_andip71_boeffla-
cm14-s5
```

Toolchain

Verlinken der Toolchain:

```
~$ cd github_andip71_boeffla-cm14-s5
~$ ln -sf ../toolchain toolchain
```

bbuild-anykernel.sh

Anpassen des Skriptes „bbuild-anykernel.sh“:

```
TOOLCHAIN="$(pwd)/toolchain/google-ndk/bin/arm-linux-androideabi-"
```

Erstellen des Bauverzeichnisses:

```
~$ ./bbuild-anykernel.sh 0
0 - copy code
```

Bereinigen:

```
~$ ./bbuild-anykernel.sh 1
1 - make clean
```

Konfiguration erstellen:

```
~$ ./bbuild-anykernel.sh 2
2 - make config

Makestring: O=output arch=arm boeffla_defconfig
VARIANT_DEFCONFIG=boeffla_defconfig_variant
  HOSTCC  scripts/basic/fixdep
  GEN     /data/AndroidKernelBuild/build/output/Makefile
...
sound/soc/codecs/audience/Kconfig:41:warning: choice value used outside its
choice group
#
# configuration written to .config
#
```

Kompilieren des Kernels:

```
~$ ./bbuild-anykernel.sh 3
3 - compile

  GEN     /data/AndroidKernelBuild/build/output/Makefile
scripts/kconfig/conf --silentoldconfig Kconfig
sound/soc/codecs/audience/Kconfig:40:warning: type of 'SND_SOC_ES_SLIM'
redefined from 'boolean' to 'tristate'
sound/soc/codecs/audience/Kconfig:43:warning: type of 'SND_SOC_ES_I2C'
redefined from 'boolean' to 'tristate'
...
chipset: 3255369473, rev: 65536, platform: 5, subtype: 0
=> Found 7 unique DTB(s)
```

```
Generating master DTB... completed
compile time: 168 seconds
Kernel image size (bytes):
6486856
```

Kernel vorbereiten:

```
~$ ./bbuild-anykernel.sh 4
4 - prepare anykernel
>>> cleanup repack folder

'/data/AndroidKernelBuild/build/output/net/sunrpc/sunrpc.ko' ->
'/data/AndroidKernelBuild/repack/modules/sunrpc.ko'
'/data/AndroidKernelBuild/build/output/net/sunrpc/auth_gss/auth_rpcgss.ko'
-> '/data/AndroidKernelBuild/repack/modules/auth_rpcgss.ko'
'/data/AndroidKernelBuild/build/output/drivers/input/joystick/xpad.ko' ->
'/data/AndroidKernelBuild/repack/modules/xpad.ko'
...
'/data/AndroidKernelBuild/build/output/fs/nfs/nfs.ko' ->
'/data/AndroidKernelBuild/repack/modules/nfs.ko'
'/data/AndroidKernelBuild/build/output/fs/ntfs/ntfs.ko' ->
'/data/AndroidKernelBuild/repack/modules/ntfs.ko'>>> strip modules
```

Kernel erstellen:

```
~$ ./bbuild-anykernel.sh 5
5 - create anykernel zip
>>> create flashable zip

  adding: anykernel.sh (deflated 54%)
  adding: dtb (deflated 81%)
  adding: META-INF/ (stored 0%)
...
  adding: tools/busybox (deflated 37%)
  adding: zImage (deflated 0%)>>> create load&flash files
```

Die fertigen Dateien liegen eine Verzeichnisebene höher in „`build/`“ und „`repack/`“.

build_kernel.sh

Anpassen des Skriptes „`build_kernel.sh`“:

```
export CROSS_COMPILE="$(pwd)/toolchain/arm-linux-androideabi-4.9/bin/arm-linux-androideabi-"
```

Als nächstes wurden alle Kernelparameter aus der Datei „./arch/arm/configs/boeffla_defconfig“, die nicht in der Datei „./arch/arm/configs/msm8974_sec_defconfig“ enthalten sind, in dieser hinzugefügt. Welche Parameter das sind, kann in dieser Datei nachgelesen werden.

Kernel bauen:

```
~$ bash ./build_kernel.sh
make: Verzeichnis „/data/AndroidKernelBuild/github_andip71_boeffla-cm14-s5“ wird betreten
  HOSTCC  scripts/basic/fixdep
  GEN     /data/AndroidKernelBuild/github_andip71_boeffla-cm14-s5/output/Makefile
  HOSTCC  scripts/kconfig/conf.o
...
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  CAT     arch/arm/boot/zImage-dtb
  Kernel: arch/arm/boot/zImage-dtb is ready
make: Verzeichnis „/data/AndroidKernelBuild/github_andip71_boeffla-cm14-s5“ wird verlassen
'output/arch/arm/boot/Image' -> '/data/AndroidKernelBuild/github_andip71_boeffla-cm14-s5/arch/arm/boot/zImage'
```

Manuelles Bauen

Kopieren der Datei „./arch/arm/configs/msm8974_sec_defconfig“ zu einer eigenen Datei „./arch/arm/configs/meine-samsung-konfiguration_defconfig“:

```
~$ cp -v ./arch/arm/configs/msm8974_sec_defconfig ./arch/arm/configs/meine-samsung-konfiguration_defconfig
'./arch/arm/configs/msm8974_sec_defconfig' -> './arch/arm/configs/meine-samsung-konfiguration_defconfig'
```

Ergänzen aller fehlenden Kernelparameter aus der Datei „./arch/arm/configs/boeffla_defconfig“, die nicht in der Datei „./arch/arm/configs/meine-samsung-konfiguration_defconfig“ vorhanden sind. Welche Parameter das sind, kann in dieser Datei nachgelesen werden. Die verwendete Konfigurationsdatei

„meine-samsung-konfiguration_defconfig“ ist hier zu finden.

Bereinigen des Baumes:

```
~$ make clean && make mrproper
```

Erstellen der Konfiguration:

```
~$ make meine-samsung-konfiguration_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
sound/soc/codecs/audience/Kconfig:40:warning: type of 'SND_SOC_ES_SLIM'
redefined from 'boolean' to 'tristate'
sound/soc/codecs/audience/Kconfig:43:warning: type of 'SND_SOC_ES_I2C'
redefined from 'boolean' to 'tristate'
boolean symbol SND_SOC_MAX98506 tested for 'm'? test forced to 'n'
sound/soc/codecs/audience/Kconfig:44:warning: choice value used outside its
choice group
sound/soc/codecs/audience/Kconfig:41:warning: choice value used outside its
choice group
arch/arm/configs/bornis_defconfig:866:warning: override: reassigning to
symbol RCU_FAST_NO_HZ
arch/arm/configs/bornis_defconfig:867:warning: override: reassigning to
symbol IKCONFIG
arch/arm/configs/bornis_defconfig:868:warning: override: reassigning to
symbol IKCONFIG_PROC
arch/arm/configs/bornis_defconfig:899:warning: override: reassigning to
symbol MODULES
arch/arm/configs/bornis_defconfig:900:warning: override: reassigning to
symbol MODULE_UNLOAD
arch/arm/configs/bornis_defconfig:901:warning: override: reassigning to
symbol MODULE_FORCE_UNLOAD
arch/arm/configs/bornis_defconfig:1175:warning: override: reassigning to
symbol JOYSTICK_XPAD
arch/arm/configs/bornis_defconfig:1295:warning: override: reassigning to
symbol HID_ELECOM
arch/arm/configs/bornis_defconfig:1473:warning: override: reassigning to
symbol KEYS
#
# configuration written to .config
#
```

Erstellen des Kernels:

```
~$ make -j$(nproc --all)
scripts/kconfig/conf --silentoldconfig Kconfig
sound/soc/codecs/audience/Kconfig:40:warning: type of 'SND_SOC_ES_SLIM'
redefined from 'boolean' to 'tristate'
sound/soc/codecs/audience/Kconfig:43:warning: type of 'SND_SOC_ES_I2C'
redefined from 'boolean' to 'tristate'
boolean symbol SND_SOC_MAX98506 tested for 'm'? test forced to 'n'
sound/soc/codecs/audience/Kconfig:44:warning: choice value used outside its
choice group
sound/soc/codecs/audience/Kconfig:41:warning: choice value used outside its
choice group
  WRAP    arch/arm/include/generated/asm/auxvec.h
  WRAP    arch/arm/include/generated/asm/bitsperlong.h
  WRAP    arch/arm/include/generated/asm/cputime.h
...
 AS      arch/arm/boot/compressed/ashldi3.o
  AS      arch/arm/boot/compressed/piggy.xzkern.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  CAT     arch/arm/boot/zImage-dtb
  Kernel: arch/arm/boot/zImage-dtb is ready
```

Weitere Funde

Es gibt im Internet noch verschiedene andere Kernel zu finden, die bisher nicht weiter getestet wurden:

- https://www.android-hilfe.de/forum/aosp-aokp-basierende-custom-roms-fuer-samsung-galaxy-s5.2028/
- https://android-hubo.de/thread/5730-smartpack-kernel-g900f-lineageos-cm14-1-samsung-galaxy-s5/
- https://github.com/gsstudios/LOS-plus-kernel
- https://github.com/bemerguy/tuned-kernel-LOS-s5
- https://sunilpaulmathew.github.io/downloads/
- https://github.com/friedrich420/S5-G900F-AEL-Kernel-LOLLIPOP
- xda-developers.com (Stock-Kernel)

— *Steffen Bornemann 15.06.2018*

CHROOT, DEBIAN, Google, NDK, Toolchain, ARM, Kernel, Flash, SM-G900F, CyanogenMod