

Developing Linux with Buildroot

MA35D1/MA35D0/MA35H0

江天文

2024/04/15

Joy of innovation
nuvoTon

| Table of contents

Setting up build environment	
Installing Ubuntu 22.04 LTS 64-bit Desktop	
Fetching source code	
Overriding source directories	
Configuring Buildroot	
Specifying where files are downloaded	
Specifying location to save Buildroot configuration	
Specifying boot device	
Building the whole artifact	
Building a specific artifact	
Building Linux artifact	
Building U-Boot artifact	
Building ARM Trusted Firmware artifact	
Building optee OS artifact	
Building rootfs artifact	

Saving configurations	
Saving Buildroot configuration	
Saving Linux configuration	
Saving U-Boot configuration	
Saving Busybox configuration	
Saving uClibc configuration	
Saving all configurations	
Customizing rootfs	
Customizing rootfs through Post-Build script	
Remote debugging with the GNU GDB debugger	
Remote debugging with GDB using Eclipse	
Remote debugging with GDB using Qt Creator	
Generating standalone SDK toolchain	
Building an external (out-of-tree) kernel module	

| Setting up build environment

- Install Ubuntu 22.04.4 LTS 64-bit Desktop

Browse <https://releases.ubuntu.com/jammy> to fetch the latest Ubuntu 22.04, or directly download the ISO image <https://releases.ubuntu.com/jammy/ubuntu-22.04.4-desktop-amd64.iso> (link maybe broken or outdated).

- Update and install the essential APT packages

```
$ sudo apt update
```

```
$ sudo apt install git build-essential libncurses-dev automake
```

| Fetching source code

- Fetch Buildroot for MA35 series

\$ until git clone https://github.com/OpenNuvoton/MA35D1_Buildroot.git ma35xx-buildroot ; do echo "retry" ; done

- Create a workspace in ***the root directory of Buildroot***

\$ cd \${BR2_DIR} ; mkdir -p workspace

Note: \${BR2_DIR} denotes the root directory of Buildroot 'ma35xx-buildroot'.

- Fetch Linux into workspace/linux-5.10

\$ until git clone https://github.com/OpenNuvoton/MA35D1_linux-5.10.y.git workspace/linux-5.10 ; do echo "retry" ; done

- Fetch U-Boot into workspace/uboot-2020.07

\$ until git clone https://github.com/OpenNuvoton/MA35D1_u-boot-v2020.07.git workspace/uboot-2020.07 ; do echo "retry" ; done

- Fetch TF-A into workspace/tfa-2.3

\$ until git clone https://github.com/OpenNuvoton/MA35D1_arm-trusted-firmware-v2.3.git workspace/tfa-2.3 ; do echo "retry" ; done

- Fetch optee-os into workspace/optee-os-3.9.0

\$until git clone https://github.com/OpenNuvoton/MA35D1_optee_os-v3.9.0.git workspace/optee-os-3.9.0 ; do echo "retry" ; done

| Overriding source directories

- Fetch *local.mk* from link <https://raw.githubusercontent.com/symfund/ma35d1-portal/master/mk/local.mk> into the root directory of Buildroot.
- Modify local.mk as shown below. It is not allowed to use absolute path in local.mk, use relative path instead. It is preferable to use ***\$(CONFIG_DIR)*** to denote ***the root directory of Buildroot***.

ARM_TRUSTED_FIRMWARE_OVERRIDE_SRCDIR=*workspace/tfa-2.3*

UBOOT_OVERRIDE_SRCDIR=*\$(CONFIG_DIR)/workspace/uboot-2020.07*

OPTEE_OS_OVERRIDE_SRCDIR=*workspace/optee-os-3.9.0*

LINUX_OVERRIDE_SRCDIR=*workspace/linux-5.10*

| Configuring Buildroot

- Before configuring Buildroot, it is required to load a default configuration for target board.
- List all configurations of ma35
`$ make list-defconfigs | grep ma35`
- Load a configuration for specific board
`$ make numaker_iot_ma35d16f80_defconfig`
- Configure Buildroot
`$ make menuconfig`

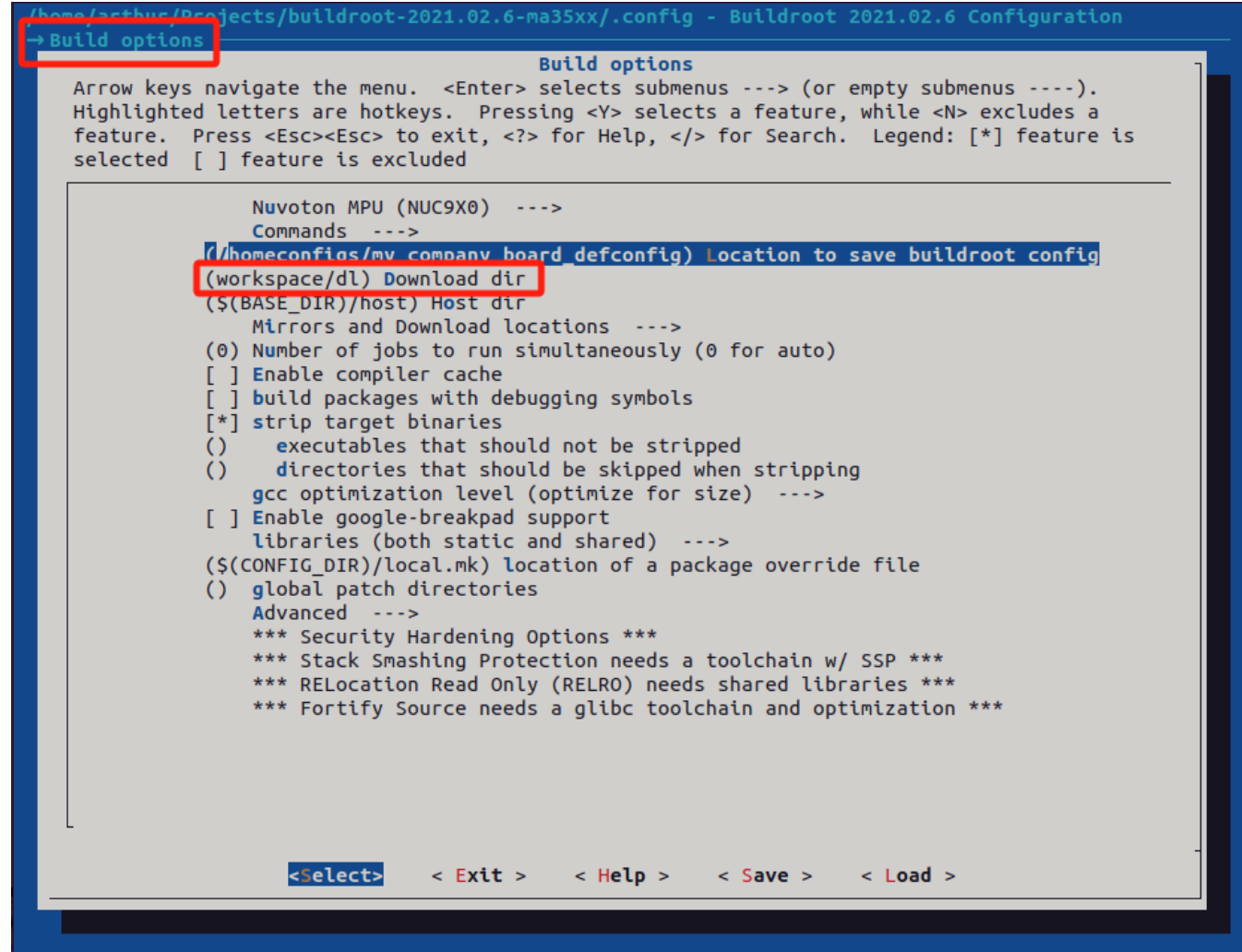
| Configuring Buildroot (Download Directory)

- Specify where files are downloaded

→ Build options

Download dir

workspace/dl



```
thoma@arthu:~/projects/buildroot-2021.02.6-ma35xx/.config - Buildroot 2021.02.6 Configuration
→ Build options
Build options
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys. Pressing <Y> selects a feature, while <N> excludes a
feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is
selected [ ] feature is excluded

Nuvoton MPU (NUC9X0) --->
Commands --->
[*] (/home/configs/my_company_board_defconfig) Location to save buildroot config
(workspace/dl) Download dir
($BASE_DIR)/host) Host dir
Mirrors and Download locations --->
(0) Number of jobs to run simultaneously (0 for auto)
[ ] Enable compiler cache
[ ] build packages with debugging symbols
[*] strip target binaries
() executables that should not be stripped
() directories that should be skipped when stripping
gcc optimization level (optimize for size) --->
[ ] Enable google-breakpad support
libraries (both static and shared) --->
($CONFIG_DIR)/local.mk) location of a package override file
() global patch directories
Advanced --->
*** Security Hardening Options ***
*** Stack Smashing Protection needs a toolchain w/ SSP ***
*** RElocation Read Only (RELRO) needs shared libraries ***
*** Fortify Source needs a glibc toolchain and optimization ***

<Select> < Exit > < Help > < Save > < Load >
```

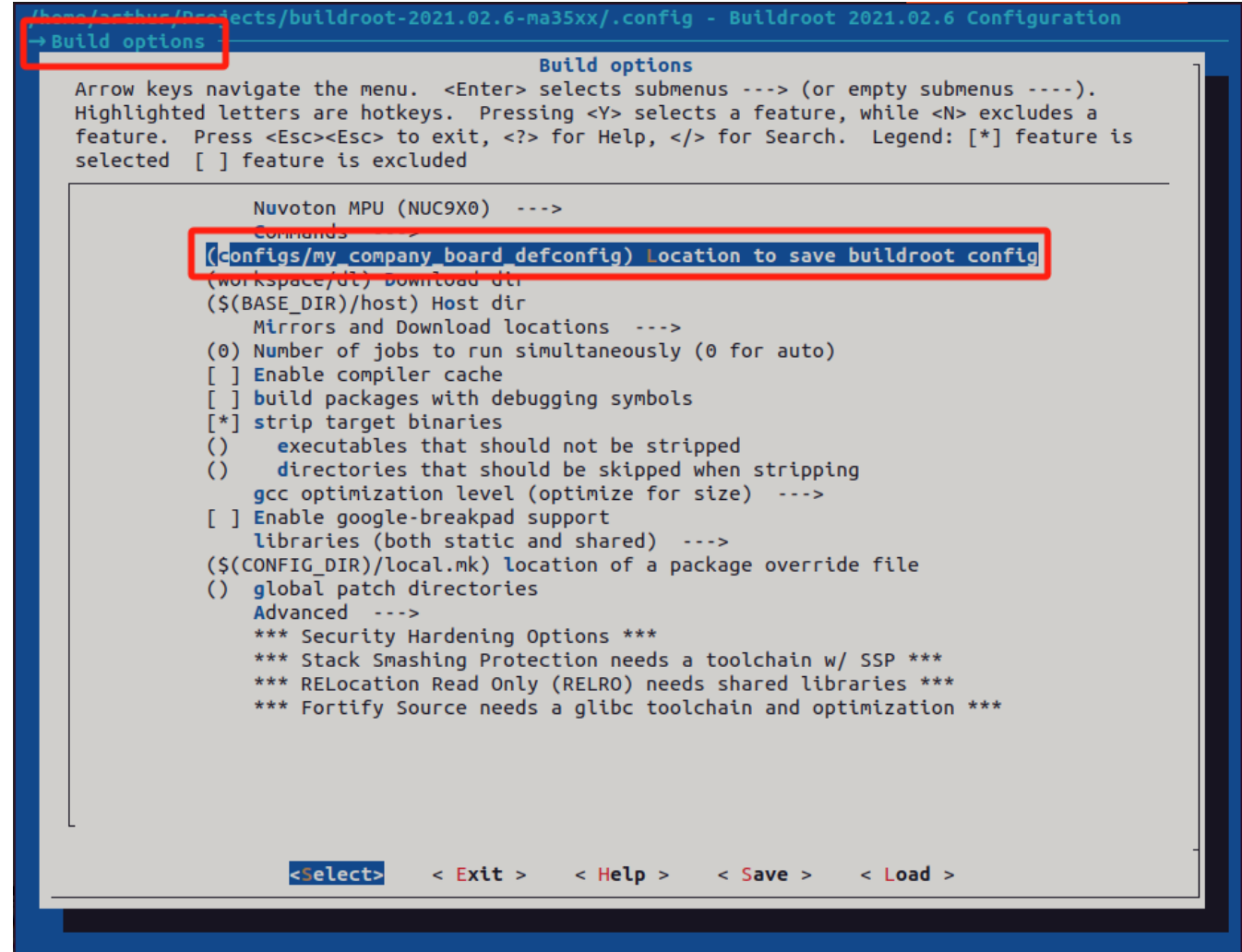

| Configuring Buildroot (Configuration Location)

- Specify location to save Buildroot configuration. The configuration file must be in the form of *_defconfig and should be saved to configs/*_defconfig

→ Build options

Location to save buildroot config

configs/my_company_board_defconfig



```
Buildroot 2021.02.6 Configuration
→ Build options

Build options
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys. Pressing <Y> selects a feature, while <N> excludes a
feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is
selected [ ] feature is excluded

Nuvoton MPU (NUC9X0) --->
Commands
([configs/my_company_board_defconfig]) Location to save buildroot config
(workspace/dt) Download dir
($BASE_DIR)/host) Host dir
Mirrors and Download locations --->
(0) Number of jobs to run simultaneously (0 for auto)
[ ] Enable compiler cache
[ ] build packages with debugging symbols
[*] strip target binaries
() executables that should not be stripped
() directories that should be skipped when stripping
gcc optimization level (optimize for size) --->
[ ] Enable google-breakpad support
libraries (both static and shared) --->
($CONFIG_DIR)/local.mk) location of a package override file
() global patch directories
Advanced --->
*** Security Hardening Options ***
*** Stack Smashing Protection needs a toolchain w/ SSP ***
*** RElocation Read Only (RELRO) needs shared libraries ***
*** Fortify Source needs a glibc toolchain and optimization ***

<Select> < Exit > < Help > < Save > < Load >
```


| Configuring Buildroot (Boot Device)

- MA35D1/MA35D0/MA35H0 series use the default U-Boot board configuration to boot from different devices.

1. ma35d1_sdcard0
2. ma35d1_sdcard1
3. ma35d1_spinand
4. ma35d1_spinor
5. ma35d1_nand

```
thema/enthuse/projects/buildroot-2021.02.6-ma35xx/.config - Buildroot 2021.02.6 Configuration
-> Bootloaders

Bootloaders
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenu ----).
Highlighted letters are hotkeys. Pressing <Y> selects a feature, while <N> excludes a feature.
Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected [ ]
feature is excluded

↑(-)
[*] optee_os
    OP-TEE OS version (MA35D1 tarball) --->
(master) Custom repository version
*- Build core
[*] Build TA devkit
[*] Build service TAs and libs
(nuvoton) Target platform (mandatory)
(MA35D1) Target platform flavor (optional)
() Additional build variables
(tee.bin tee-*_v2.bin) Binary boot images
[ ] shim
[*] U-Boot
    Build system (Kconfig) --->
    U-Boot Version (Custom tarball) --->
    ($ (call github,OpenNuvoton,MA35D1_u-boot-v2020.07, master)/MA35D1_u-boot-v2020.07-m
    () Custom U-Boot patches
    U-Boot configuration (Using an in-tree board defconfig file) --->
    (ma35d1_nand) Board defconfig
    [ ] Auto resize sdcard to Max.
    () Additional configuration fragment files
    [*] U-Boot needs dtc
        U-Boot needs host Python (no) --->
    [ ] U-Boot needs pylibfdt
    [ ] U-Boot needs pyelftools
    [ ] U-Boot needs OpenSSL
    [ ] U-Boot needs lzop
↓(+)

<Select>  <Exit>  <Help>  <Save>  <Load>
```

| Configuring Buildroot (Remote Debugging)

- The options of toolchain and other packages must be tailored to meet the conditions for remote debugging with GDB server.

→Toolchain

[*] Build cross gdb for the host

Python support (Python 3)

→Target packages →Debugng, profiling

[*] gdb

→Target packages →Networking applications

[*] openssh

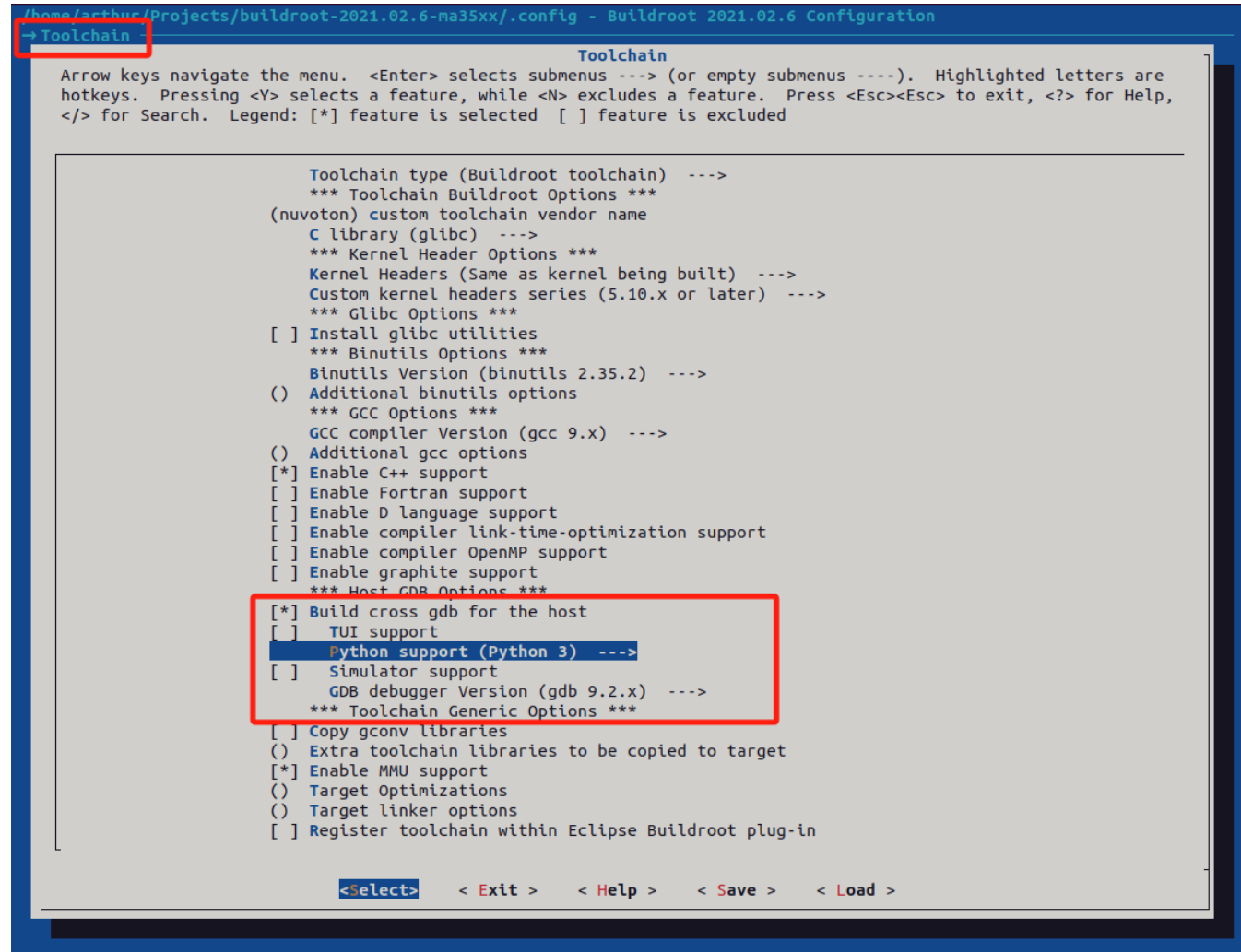
[*] server

[*] key utilities

[*] rsync

→Host utilities

[*] host cmake



```
~/home/athun/Projects/buildroot-2021.02.6-na35xx/.config - Buildroot 2021.02.6 Configuration
→ Toolchain

Toolchain
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenu ----). Highlighted letters are
hotkeys. Pressing <Y> selects a feature, while <N> excludes a feature. Press <Esc><Esc> to exit, <?> for Help,
</> for Search. Legend: [*] feature is selected [ ] feature is excluded

Toolchain type (Buildroot toolchain) --->
*** Toolchain Buildroot Options ***
(nuvoton) custom toolchain vendor name
C library (glibc) --->
*** Kernel Header Options ***
Kernel Headers (Same as kernel being built) --->
Custom kernel headers series (5.10.x or later) --->
*** Glibc Options ***
[ ] Install glibc utilities
*** Binutils Options ***
Binutils Version (binutils 2.35.2) --->
() Additional binutils options
*** GCC Options ***
GCC compiler Version (gcc 9.x) --->
() Additional gcc options
[*] Enable C++ support
[ ] Enable Fortran support
[ ] Enable D language support
[ ] Enable compiler link-time-optimization support
[ ] Enable compiler OpenMP support
[ ] Enable graphite support
*** Host GDB Options ***
[*] Build cross gdb for the host
[ ] TUI support
Python support (Python 3) --->
[ ] Simulator support
GDB debugger Version (gdb 9.2.x) --->
*** Toolchain Generic Options ***
[ ] Copy gconv libraries
() Extra toolchain libraries to be copied to target
[*] Enable MMU support
() Target Optimizations
() Target linker options
[ ] Register toolchain within Eclipse Buildroot plug-in

<Select> <Exit> <Help> <Save> <Load>
```

| Building the whole artifact

- Type **make** to build the whole artifact
\$ make
- Depending on boot device, machine type and memory capacity, the artifact is produced at location **output/images** as shown in below table.

Boot Device	Artifact
SD0/SD1	pack-core-image-buildroot-ma35d1-som-256m-sdcard.bin
SPI NAND Flash	pack-core-image-buildroot-ma35d1-iot-512m-spinand.bin
SPI Nor Flash	pack-core-image-buildroot-ma35d1-iot-512m-spinor.bin
NAND Flash	pack-core-image-buildroot-ma35d1-som-1g-nand.bin

- Use **NuWriter** to program the artifact to memory of boot device

| Building a specific artifact

- Build the artifact Linux when its source is changed
`$ make linux-rebuild ; make`
- Build the artifact U-Boot when its source is changed
`$ make uboot-alter-rebuild`
Note: Nuvoton modified, do not use `uboot-rebuild`.
- Build the artifact Arm Trusted Firmware when its source is changed
`$ make arm-trusted-firmware-rebuild ; make`
- Build the artifact optee OS when its source is changed
`$ make optee-os-rebuild ; make`
- Build the rootfs when in Buildroot a package is selected/unselected or *system configuration* is changed
`$ make rootfs-rebuild`

| Saving configurations

- Save Buildroot configuration
`$ make buildroot-save-config`
- Save Linux configuration
`$ make linux-save-config`
- Save U-Boot configuration
`$ make uboot-save-config`
- Save Busybox configuration
`$ make busybox-save-config`
- Save uClibc configuration
`$ make uclibc-save-config`
- Save all configurations
`$ make all-save-config`

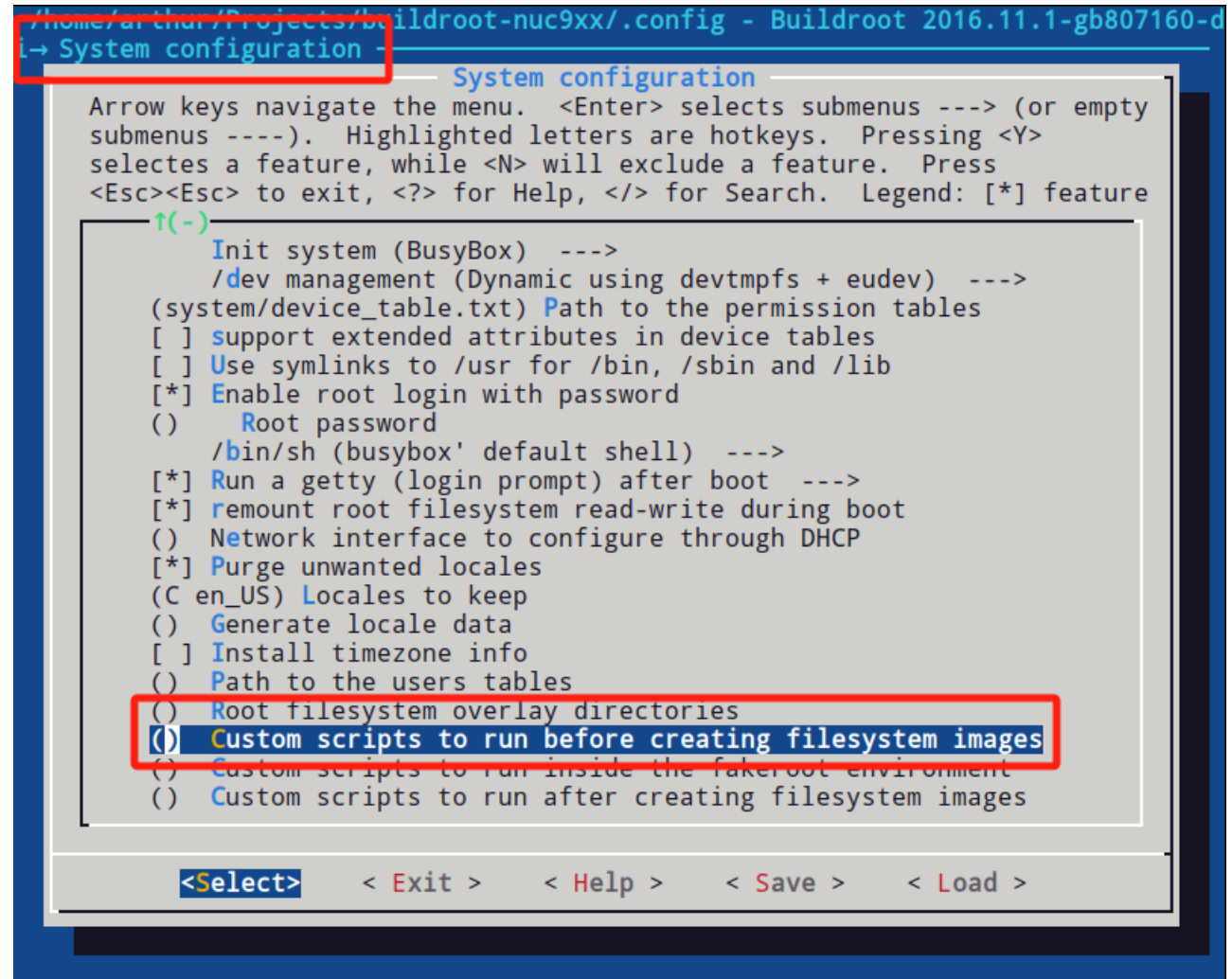
Main	Parts
Buildroot Configuration	U-Boot Configuration
	Linux Configuraiton
	Busybox Configuration
	uClibc Configuration

Configuration	Command (*-save-config)
buildroot	make buildroot -save-config
linux	make linux -save-config
uboot	make uboot -save-config
busybox	make busybox -save-config
uclibc	make uclibc -save-config
all	make all -save-config

Customizing rootfs

- Buildroot provides two recommended methods *root filesystem overlay(s)* and *post build script(s)*, which can co-exist, to customize the generated target root filesystem.

1. Root filesystem overlays
2. Post-build scripts



```
/home/Arthur/Projects/buildroot-nuc9xx/.config - Buildroot 2016.11.1-gb807160-d
-> System configuration -
System configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
selects a feature, while <N> will exclude a feature. Press
<Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature
↑(-)
  Init system (BusyBox) --->
    /dev management (Dynamic using devtmpfs + eudev) --->
    (system/device_table.txt) Path to the permission tables
    [ ] support extended attributes in device tables
    [ ] Use symlinks to /usr for /bin, /sbin and /lib
    [*] Enable root login with password
    () Root password
    /bin/sh (busybox' default shell) --->
    [*] Run a getty (login prompt) after boot --->
    [*] remount root filesystem read-write during boot
    () Network interface to configure through DHCP
    [*] Purge unwanted locales
    (C en_US) Locales to keep
    () Generate locale data
    [ ] Install timezone info
    () Path to the users tables
    () Root filesystem overlay directories
    (0) Custom scripts to run before creating filesystem images
    () Custom scripts to run inside the fakeroot environment
    () Custom scripts to run after creating filesystem images

<Select>  < Exit >  < Help >  < Save >  < Load >
```

| Customizing rootfs through Post-Build script

- Post-Build scripts are shell scripts called after Buildroot builds all the selected software, but before the rootfs images are assembled. The following environment variables can be used in post-build scripts.

Variable	Value
BR2_CONFIG	the path to the Buildroot .config file
CONFIG_DIR	the directory containing the .config file, and therefore the top-level Buildroot Makefile to use
TARGET_DIR	the path to the <i>output/target</i>
BINARIES_DIR	The place where all binary files (aka images) are stored

| Customizing rootfs through Post-Build script

- The below Post-Build script file checks whether **OpenSSH Server** is enabled in *the Buildroot configuration* (**`${BR2_CONFIG}`**). If enabled, it changes the configuration file (`/etc/ssh/sshd_config`) of OpenSSH Server on the generated target root filesystem.

Post-Build Script (`workspace/scripts/post-build.sh`)

```
if grep -Eq "^BR2_PACKAGE_OPENSSH=y$" ${BR2_CONFIG}; then
    sed -i 's/^#PermitRootLogin.*/PermitRootLogin yes/' ${TARGET_DIR}/etc/ssh/sshd_config
    sed -i 's/^#PasswordAuthentication.*/PasswordAuthentication yes/' ${TARGET_DIR}/etc/ssh/sshd_config
    sed -i 's/^#PermitEmptyPasswords.*/PermitEmptyPasswords yes/' ${TARGET_DIR}/etc/ssh/sshd_config
fi
```

| Generating standalone SDK toolchain

- For third-party developers, a standalone SDK toolchain is must be have on hand. By distributing standalone SDK tool to independent software vendors, OEM manufacturers need not to disclose proprietary source code to the public.

Resort to the local Makefile ***local.mk***, SDK toolchain can be generated without user intervention by the extensions of build instruction.

| Generating standalone SDK toolchain

- Before generating the standalone SDK toolchain, Buildroot must be correctly configured. That means toolchain options are tailored to meet the development requirements, some mandatory packages are selected in mind.

Refer to the slide *Configuring Buildroot (Remote Debugging)* for configuring toolchain to meet development requirements.

| Generating standalone SDK toolchain

- Begin building SDK toolchain, the action *make clean* is optional, if want to save the build time.

```
$ make clean ; make sdk-tool
```

- When complete generating SDK tool, the SDK tool is located at *output/images/aarch64-nuvoton-linux-gnu_sdk-buildroot_installer*
- To install the SDK tool on local computer in which the SDK tool is built or another computer, launch the SDK tool installer show below.

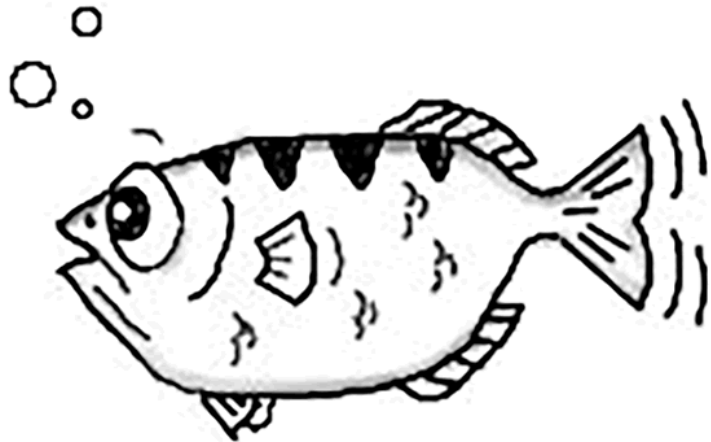
```
$ sudo output/images/aarch64-nuvoton-linux-gnu_sdk-buildroot_installer
```

- By default, the SDK tool is installed in */opt/aarch64-nuvoton-linux-gnu_sdk-buildroot*
- Before using the SDK tool, open a terminal window and set up the build environment for the new terminal.

```
$ source /opt/aarch64-nuvoton-linux-gnu_sdk-buildroot/environment-setup
```

| Remote debugging with the GNU GDB debugger

- MA35D1/MA35D0/MA35H0 series support remote debugging with *the GNU GDB debugger* using *Eclipse* and *Qt Creator*.



GDB
The GNU Project
Debugger

| Remote debugging with GDB using Eclipse

- Refer to developer guide (PDF) *Remote Debugging with the GNU GDB Debugger using Eclipse*.

The Eclipse Installer 2023-12 R now includes a JRE for macOS, Windows and Linux.

Try the Eclipse **Installer** 2023-12 R

The easiest way to install and update your Eclipse Development Environment.

[Find out more](#)

📄 874,864 Installer Downloads

📄 772,712 Package Downloads and Updates

Download

macOS [x86_64](#) | [AArch64](#)

Windows [x86_64](#)

Linux [x86_64](#) | [AArch64](#)

| Remote debugging with GDB using Qt Creator

- Refer to developer guide (PDF) *Remote Debugging with the GNU GDB Debugger using Qt Creator*.



| Building an external (out-of-tree) kernel module

- To build an out-of-tree (external) module, a prebuild kernel with the configuration and header files must exist. Also the kernel must have been configured to enable loadable module support (CONFIG_MODULES). That means the Linux artifact is built by typing “make linux-rebuild” in Buildroot.
- Assuming that the module name is “mod_xyz”, create a Kbuild file as show below

```
# filename: Kbuild
obj-m := mod_xyz.o
mod_xyz-y := main.o
ccflags-y := -I$(src)
```

| Building an external (out-of-tree) kernel module

- Create a makefile as shown below

```
# filename: Makefile
ifneq ($(KERNELRELEASE),)
include Kbuild
else
KERNELDIR ?= /lib/modules/`uname -r`/build

default:
$(MAKE) -C $(KERNELDIR) M=$$PWD

clean:
$(MAKE) -C $(KERNELDIR) M=$$PWD clean

endif
```

| Building an external (out-of-tree) kernel module

- Structure of kernel module source
\$(src)
 - |_ Kbuild
 - |_ Makefile
 - |_ main.c
- Open a new terminal window, and set up build environment for the terminal
\$ source \${BR2_DIR}/output/host/environment-setup
NOTE: \${BR2_DIR} denotes the root directory of Buildroot.
- Build the kernel module
\$ make
- Clean build
\$ make clean

```
/* filename: main.c */
#include <linux/init.h>
#include <linux/module.h>
#include <linux/printk.h>
#include <linux/kernel.h>
#include <linux/utsname.h>

static int __init mod_xyz_init(void)
{
    printk(KERN_INFO "Loading mod_xyz ...");
    pr_alert("%s version %s: %s\n", utsname()->sysname, utsname()->release, utsname()->version);
    return 0;
}

static void __exit mod_xyz_exit(void)
{
    pr_alert("exit mod_xyz driver\n");
}

module_init(mod_xyz_init);
module_exit(mod_xyz_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Linux module mod-xyz");
MODULE_AUTHOR("2024@nuvoton");
```

Joy of innovation
nuvoTon

谢谢

謝謝

Děkuji

Bedankt

Thank you

Kiitos

Merci

Danke

Grazie

ありがとう

감사합니다

Dziękujemy

Obrigado

Спасибо

Gracias

Teşekkür ederim

Cảm ơn