Slackware ARM current gcc-10.3.x armv8 arm64 aarch64 cross-compiler for the Raspberry Pi 4

Preface

With the recent updates on Slackware ARM (December 2020) to gcc-10.x this is an ARMv8 aarch64 cross-compiler bash script that's intended for building binaries from source code and turning them into Slackware packages. Or whatever use you may find for this script.

Previous work in this area had been done for gcc-9.x and the build script and intructions are mostly the same. Some of the code in the script has changed from the old version, with updated versions and especially the patching.

Notes

Slackware ARM current was used on a Raspberry Pi 4 [4GB RAM] to build and install the gcc-10.3.0 aarch64-linux cross-compiler.

The Linux kernel source downloaded is from the Raspberry Pi Github repository: https://github.com/raspberrypi/linux

For usage see the commented section at the top of the script itself. It explains pretty much everything you need to know in order to get started and be successful. If you need to you can refer to the instructions on this page as they are exactly the same, apart from the version numbers.

NB: 'libsanitizer asan' will need patching before building glibc. The issue itself is that no PATH_MAX has been defined in the source and there needs to be a value set in order for it to compile successfully. The patching is all done by the script code. No user input is required after running it.

Requirements

As a pre-requisite, you should have;

- a Raspberry Pi 3 or 4 (i.e. a 64bit CPU) running Slackware ARM current with as much unused [>= 5GB] storage space on your system as possible.
- gawk, git, bison and flex, already installed on your system.

Aarch64 cross-compiler script code

Download the 'SARPi64.SlackBuild-gcc-10.3-aarch64-cc.sh' file by clicking the link at the top of the

script code (or copy and paste the code if you prefer) below.

Make the file executable using 'chmod +x SARPi64.SlackBuild-gcc-10.3-aarch64-cc.sh' or 'chmod 755 SARPi64.SlackBuild-gcc-10.3-aarch64-cc.sh' command.

Refer to the commented top section of the script code for usage.

SARPi64.SlackBuild-gcc-10.3-aarch64-cc.sh

```
#! /bin/bash

#######################################################################
# Slackware ARM gcc-10.3.0 aarch64 cross-compiler for Raspberry Pi
#
# SARPi64.SlackBuild-gcc-10.3.0-aarch64-cc [v3.0] - 2021-04-22
#
# 2020-12-29 by Exaga   -   v3.0   -  gcc-10.x
# 2019-07-10 by Exaga   -   v2.0   -  gcc-9.x
# 2016-12-17 by Exaga   -   v1.0   -  gcc-5.x
# 2016-12-12 by Exaga   -   v0.2b  -  beta
# 2016-12-05 by Exaga   -   v0.1a  -  alpha
#
#######################################################################
#
# This script downloads RPi Linux kernel source and the required
binaries,
# and configures, builds, patches, and installs a gcc 10.3.x aarch64-
linux
# cross-compiler on Slackware ARM current running on a Raspberry Pi
3/4.
#
### Usage & Installation ###
# You should create a 'build-dir' folder and copy this script to it
# (e.g. /tmp/build-dir) and run it from there as a 'root' user.
#
# ~# chmod +x SARPi64.SlackBuild-gcc-10.3-aarch64-cc.sh
# ~# ./SARPi64.SlackBuild-gcc-10.3-aarch64-cc.sh
#
# You may install the cross-compiler anywhere you like, as long as it
can be
# accessed by a normal user (i.e. not 'root'). The default is
/tmp/.gcc-cross
# but if this is not suitable then set your own installation directory
with
# INSTALL_PATH variable, in the settings below.
#
# Ensure 'bison', 'flex', 'gawk', and 'git' are installed on your
```

```
system
# before running this script! Use these commands to check:
#
# ~# whereis gawk
# ~# whereis git
# ~# whereis bison
# ~# whereis flex
#
# If you need to install any of the packages above [* check for
updates!]:
#
http://slackware.uk/slackwarearm/slackwarearm-current/slackware/a/gawk*
.txz
#
http://slackware.uk/slackwarearm/slackwarearm-current/slackware/d/git*.
txz
#
http://slackware.uk/slackwarearm/slackwarearm-current/slackware/d/bison
*.txz
#
http://slackware.uk/slackwarearm/slackwarearm-current/slackware/d/flex*
.txz
#
# NB: The gcc package you compile should match your currently installed
gcc
# version. Use this command to check your current gcc version:
#
# ~# gcc --version
#
# More recent gcc packages-versions may exist. You may wish to install
them.
# NB: if you use newer packages - glibc version _MUST_ suit gcc
version! The
# thing to make sure is the release dates of gcc and glibc versions
being as
# close as possible.
#
# binutils - https://ftp.gnu.org/gnu/binutils/
# cloog - ftp://gcc.gnu.org/pub/gcc/infrastructure/
# gcc - https://ftp.gnu.org/gnu/gcc/
# glibc - https://ftp.gnu.org/gnu/glibc/
# gmp - https://ftp.gnu.org/gnu/gmp/
# isl - ftp://gcc.gnu.org/pub/gcc/infrastructure/
# mpfr - https://ftp.gnu.org/gnu/mpfr/
# mpc - https://ftp.gnu.org/gnu/mpc/
#
### IMPORTANT! ###
# This script will export the INSTALL_PATH variable into the $PATH.
# The PATH of the cross-compiler should always be the first item in
# the $PATH. PATH command:
#
```

```
# ~# export PATH=/tmp/.gcc-cross/bin:$PATH
#
# To check that the INSTALL_PATH is in your $PATH use this command:
#
# ~# echo $PATH
#
### Usage ###
# This script was created on Slackware ARM and intended for research
and
# development towards a Slackware AArch64 port. This script may work on
# other Linux distributions and hardware but it has not been tested and
# therefore cannot be verified. It may be freely distributed, copied,
# modified, or plagiarised in the hope that it will be of some use
towards
# the goal of Slackware AArch64.
#
### Resource(s) ###
# http://www.slackware.com
# http://slackware.uk
# http://arm.slackware.com/FAQs
# http://wiki.osdev.org/GCC_Cross-Compiler
# https://www.raspberrypi.org/documentation/linux/kernel
# https://www.github.com/raspberrypi
# https://ftp.gnu.org/gnu
# ftp://gcc.gnu.org/pub/gcc/infrastructure
#
#######################################################################
#######


# Installation directory - edit INSTALL_PATH as required
INSTALL_PATH=/tmp/.gcc-cross

# Required build packages-versions [* newer versions may exist]
BINUTILS_VERSION=binutils-2.36
CLOOG_VERSION=cloog-0.18.1
GCC_VERSION=gcc-10.3.0
GLIBC_VERSION=glibc-2.33
GMP_VERSION=gmp-6.2.1
ISL_VERSION=isl-0.18
MPFR_VERSION=mpfr-4.1.0
MPC_VERSION=mpc-1.2.1

# RPi GitHub Linux source - working branch [e.g. rpi-4.14.y |
rpi-4.19.y | rpi-5.2.y ]
DEV_BRANCH=rpi-5.10.y


#######################################################################
```

```
######
## DO NOT EDIT ANYTHING BELOW THIS LINE UNLESS YOU KNOW WHAT YOU'RE
DOING! ##
#################################################################
######

# Halt build process on error [with output]
set -euo pipefail
IFS=$'\n\t'
# Uncomment for additional error output when testing/debugging
#trap 'previous_command=$this_command; this_command=$BASH_COMMAND'
DEBUG
#trap 'echo "exit $? due to $previous_command"' EXIT

# Build variables
PRGNAM=SARPi64.SlackBuild-aarch64-cc
ARCH_TARGET=aarch64-linux
LINUX_ARCH=arm64
QUADLET=aarch64-unknown-linux-gnu # aarch64-arm-none-eabi
LINUX_FLAVOUR=linux-rpi
RPI_GITURL_LINUX=https://github.com/raspberrypi
BUILD_LANGUAGES="--enable-languages=c,c++" # --enable-
languages=all,ada,c,c++,fortran,go,jit,lto,objc,obj-c++
ALT_CONFIG_OPTIONS="--disable-multilib" # --disable-threads --disable-
shared --disable-multiarch
TEST_CONFIG_OPTIONS="--with-arch=armv8-a --with-tune=cortex-a72 --with-
fpu=vfpv3-d16 --with-float=hard" #
RPI4_CONFIG_OPTIONS="--prefix=$INSTALL_PATH --target=arm-linux-
gnueabihf --enable-languages=c,c++ --with-arch=armv8-a --with-fpu=vfp -
-with-float=hard --disable-multilib" #
PARALLEL_JOBS=-j4 #
https://www.gnu.org/software/make/manual/html_node/Parallel.html
CWD=$(pwd)

# Define CONFIG_OPTIONS for build
CONFIG_OPTIONS=$ALT_CONFIG_OPTIONS

# Uncomment to log EVERYTHING during build process [** WARNING! HUGE
log filesize! **]
#LOGFLE=${PRGNAM}_build_$(date +"%F").log
#exec 1> >(logger -s -t $(basename $0)) 2>&1 > $LOGFLE

# Output aesthetics
sarpiSP64 () {
echo
echo " ##############################################"
echo " ## $PRGNAM "
echo " ## Build: $GCC_VERSION  Kernel ${DEV_BRANCH} "
echo " ## Timestamp: $(date '+%F %T') "
echo " ## SARPi64 Project [sarpi64.penthux.net] "
echo " ##############################################"
```

```
echo
}

sarpiSP64
echo "Starting $PRGNAM build ..."

# INSTALL_PATH needs to be at the front of $PATH
# Command: export PATH=/tmp/.gcc-cross/bin:$PATH
echo "Checking $ARCH_TARGET $INSTALL_PATH/bin \$PATH ..."
if [[ ! "$PATH" =~ $INSTALL_PATH ]]; then
    export PATH=/"${INSTALL_PATH}"/bin:$PATH
# echo -e $INSTALL_PATH/bin:$(cat $PATH) > $PATH || exit 1
else
    echo "Found $INSTALL_PATH\/bin in \$PATH : OK! ... "
fi

# Prerequisite packages
BISON_REQ=$(which bison)
FLEX_REQ=$(which flex)
GAWK_REQ=$(which gawk)
GIT_REQ=$(which git)

# Check prerequisites are installed, or exit
if [ ! -e "$BISON_REQ" ]; then
  echo "ERROR: bison not found!"
  echo "Install bison before you run this script!"
  exit 1
elif [ ! -e "$FLEX_REQ" ]; then
  echo "ERROR: flex not found!"
  echo "Install flex before you run this script!"
  exit 1
elif [ ! -e "$GAWK_REQ" ]; then
  echo "ERROR: gawk not found!"
  echo "Install gawk before you run this script!"
  exit 1
elif [ ! -e "$GIT_REQ" ]; then
  echo "ERROR: git not found!"
  echo "Install git before you run this script!"
  exit 1
else
  echo "Prerequisite packages are installed ..."
fi

# Download RPi kernel source ** this may take a while **
cd "$CWD"
echo "Checking kernel $DEV_BRANCH source ..."
if [ ! -e $LINUX_FLAVOUR/Makefile ]; then
  echo "Downloading kernel $DEV_BRANCH source ..."
  git clone --depth=1 $RPI_GITURL_LINUX/linux.git --branch $DEV_BRANCH
```

```
$LINUX_FLAVOUR
fi
cd $LINUX_FLAVOUR
echo "Checking kernel $DEV_BRANCH branch for updates ..."
git checkout -f $DEV_BRANCH
cd "$CWD"

# Download gcc and related packages to build cross-compiler
echo "Downloading packages ..."
if [ ! -d "$CWD"/$BINUTILS_VERSION ]; then
  wget -nc --progress=bar
https://ftp.gnu.org/gnu/binutils/$BINUTILS_VERSION.tar.gz
  tar -xvf $BINUTILS_VERSION.tar.gz
fi
if [ ! -d "$CWD"/$GCC_VERSION ]; then
  wget -nc --progress=bar
https://ftp.gnu.org/gnu/gcc/$GCC_VERSION/$GCC_VERSION.tar.gz
  tar -xvf $GCC_VERSION.tar.gz
fi
if [ ! -d "$CWD"/$GLIBC_VERSION ]; then
  wget -nc --progress=bar
https://ftp.gnu.org/gnu/glibc/$GLIBC_VERSION.tar.xz
  tar -xvf $GLIBC_VERSION.tar.xz
fi
if [ ! -d "$CWD"/$GMP_VERSION ]; then
  wget -nc --progress=bar
https://ftp.gnu.org/gnu/gmp/$GMP_VERSION.tar.xz
  tar -xvf $GMP_VERSION.tar.xz
fi
if [ ! -d "$CWD"/$MPFR_VERSION ]; then
  wget -nc --progress=bar
https://ftp.gnu.org/gnu/mpfr/$MPFR_VERSION.tar.xz
  tar -xvf $MPFR_VERSION.tar.xz
fi
if [ ! -d "$CWD"/$MPC_VERSION ]; then
  wget -nc --progress=bar
https://ftp.gnu.org/gnu/mpc/$MPC_VERSION.tar.gz
  tar -xvf $MPC_VERSION.tar.gz
fi
if [ ! -d "$CWD"/$ISL_VERSION ]; then
  wget -nc --progress=bar
ftp://gcc.gnu.org/pub/gcc/infrastructure/$ISL_VERSION.tar.bz2
  tar -xvf $ISL_VERSION.tar.bz2
fi
if [ ! -d "$CWD"/$CLOOG_VERSION ]; then
  wget -nc --progress=bar
ftp://gcc.gnu.org/pub/gcc/infrastructure/$CLOOG_VERSION.tar.gz
  tar -xvf $CLOOG_VERSION.tar.gz
fi

# Create symbolic links so gcc builds these dependencies automatically
```

```
echo "Creating symbolic links in gcc ..."
cd "$CWD"/$GCC_VERSION
ln -sf ../$MPFR_VERSION mpfr
ln -sf ../$GMP_VERSION gmp
ln -sf ../$MPC_VERSION mpc
ln -sf ../$ISL_VERSION isl
ln -sf ../$CLOOG_VERSION cloog

# Create aarch64 cross-compiler install directory
echo "Creating $INSTALL_PATH directory ..."
rm -rf $INSTALL_PATH
mkdir -p $INSTALL_PATH
chown "$(whoami)" $INSTALL_PATH
cd "$CWD"

# Build binutils
echo "Building binutils ..."
rm -rf build-binutils
mkdir build-binutils
cd build-binutils
../$BINUTILS_VERSION/configure --prefix=$INSTALL_PATH --
target=$ARCH_TARGET $CONFIG_OPTIONS
make $PARALLEL_JOBS
echo "Installing binutils ..."
make install

# Install Linux kernel headers
echo "Installing kernel headers ..."
cd "$CWD"/$LINUX_FLAVOUR
make ARCH=$LINUX_ARCH INSTALL_HDR_PATH=$INSTALL_PATH/$ARCH_TARGET
headers_install
cd "$CWD"

# Build gcc C and C++ cross-compilers
echo "Building gcc $ARCH_TARGET C,C++ cross-compiler ..."
mkdir -p build-gcc
cd build-gcc
../$GCC_VERSION/configure --prefix=$INSTALL_PATH --target=$ARCH_TARGET
$BUILD_LANGUAGES $CONFIG_OPTIONS
make $PARALLEL_JOBS all-gcc
echo "Installing gcc $ARCH_TARGET cross-compiler to $INSTALL_PATH ..."
make $PARALLEL_JOBS install-gcc

# create gcc-10.3.0 libsanitizer asan_linux-cpp.patch file
cd "$CWD"
touch asan_linux-cpp.patch
cat << EOF > asan_linux-cpp.patch
--- gcc-10.3.0/libsanitizer/asan/asan_linux.cpp 2021-04-08
12:56:30.229766760 +0100
```

```
+++ asan_linux.cpp.new  2021-04-22 13:26:26.000000000 +0100
@@ -76,6 +76,10 @@
 asan_rt_version_t  __asan_rt_version;
 }

+#ifndef PATH_MAX
+#define PATH_MAX 4096
+#endif
+
 namespace __asan {

 void InitializePlatformInterceptors() {}

EOF

# Patch gcc-10.3.x/libsanitizerasan/asan_linux.cpp [or the build will
fail]
ASANLINUXCC=$CWD/$GCC_VERSION/libsanitizer/asan/asan_linux.cpp
if [ ! -f "$ASANLINUXCC".orig ]; then
  echo "Patching $ASANLINUXCC ..."
  patch -b "$ASANLINUXCC" asan_linux-cpp.patch || exit 1
  sarpiSP64
  echo "$ASANLINUXCC has been PATCHED! ..."
  echo "Backup of original: $ASANLINUXCC.orig ..."
  sleep 10
fi

# Build and install glibc's standard C library headers and startup
files
echo "Building glibc library headers ..."
mkdir -p build-glibc
cd build-glibc
../$GLIBC_VERSION/configure --prefix=$INSTALL_PATH/$ARCH_TARGET --
build="$MACHTYPE" --host=$ARCH_TARGET --target=$ARCH_TARGET --with-
headers=$INSTALL_PATH/$ARCH_TARGET/include $CONFIG_OPTIONS
libc_cv_forced_unwind=yes
make $PARALLEL_JOBS install-bootstrap-headers=yes install-headers
make $PARALLEL_JOBS csu/subdir_lib
echo "Installing glibc library headers ..."
install csu/crt1.o csu/crti.o csu/crtn.o $INSTALL_PATH/$ARCH_TARGET/lib
$ARCH_TARGET-gcc -nostdlib -nostartfiles -shared -x c /dev/null -o
$INSTALL_PATH/$ARCH_TARGET/lib/libc.so
touch $INSTALL_PATH/$ARCH_TARGET/include/gnu/stubs.h

# Build glibc support library
echo "Building glibc support library ..."
cd "$CWD"/build-gcc
make $PARALLEL_JOBS all-target-libgcc
echo "Installing glibc support library ..."
make install-target-libgcc
```

```
# Finish building glibc's standard C library and install it
echo "Completing glibc C library ..."
cd "$CWD"/build-glibc
make $PARALLEL_JOBS
echo "Installing glibc C library ..."
make install

# Finish building gcc's C++ library and install it
echo "Completing glibc C++ library ..."
cd "$CWD"/build-gcc
make $PARALLEL_JOBS
echo "Installing glibc C++ library ..."
make install
cd "$CWD"

# Check status of aarch64-linux-gcc cross-compiler
echo "Checking status of $ARCH_TARGET-gcc cross-compiler ..."
ARCH_TARGET_STATUS=$(which $ARCH_TARGET-gcc)
$ARCH_TARGET-gcc -v
if [ ! -e "$ARCH_TARGET_STATUS" ]; then
  # ERROR!
  echo "ERROR: $ARCH_TARGET-gcc not responding!"
  sarpiSP64
  echo "$(date +"%F %T") : $PRGNAM FAILED! ..."
  exit 1
else
  # Done!
  echo "Verifying $ARCH_TARGET-gcc \$PATH ..."
  echo "PATH: $PATH"
  echo "Status: $ARCH_TARGET-gcc : READY!"
  sarpiSP64
  echo "$(date +"%F %T") : $PRGNAM build complete ..."
fi

#
exit 0

#EOF<*>
```

Configuring the system to use the Aarch64 cross-compiler

Once the cross-compiler has been built and you intend to use it to compile Aarch64 [64bit] binaries, the location of the tool-chains must be exported to your system's $PATH variable. It's also prudent to have this entry appear before all other PATHs. So, for example, to check what your existing $PATH includes, use the 'echo $PATH' command like this:

```
root@slackware:~# echo $PATH
/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/usr/games:/usr
/lib/qt/bin
```

This is the output of the $PATH on our Slackware ARM system before adding the '/path/to/cross-compiler/bin' tool-chain location. Your own $PATH might be very different.

If you didn't change the 'INSTALL_PATH' variable in the build script then the command the script uses to export the PATH will be like this:

```
root@slackware:~# export PATH=/tmp/.gcc-cross/bin:$PATH
```

This is the command to use if it doesn't already appear in your system $PATH. Obviously, if you've modified the INSTALL_PATH variable then that location will be used instead of the default. You can check the $PATH variable to ensure that it has been added correctly:

```
root@slackware:~# echo $PATH
/tmp/.gcc-
cross/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/usr/
games:/usr/lib/qt/bin
```

The example output above all looks good because the cross compile PATH appears first before everything else.

Aarch64 cross-compiling 'HOWTO' example

So, when the Aarch64 tool-chains feature in the $PATH on the system, they can be utilised to build software for the ARMv8 architecture instead of your Slackware ARM system's [ARMv7] existing infrastructure. It's all done via the 'make' process.

As an example to show how this can be achieved, the bespoke RPi kernel source 'master' branch will be used. This kernel source will be downloaded and located in '/tmp/linux-rpi' directory. [**NB: note that this is NOT '/usr/src/linux' - it should never reside in that standard location for cross-compiling purposes!**]. After downloading the Linux source, use 'cd' to the directory where it's located:

```
root@slackware:~# git clone --depth=1 git://github.com/raspberrypi/linux
/tmp/linux-rpi
Cloning into '/tmp/linux-rpi'...
~ Enumerating, Counting, Compressing, yadda yadda yadda ~
Updating files: 100% (66403/66403), done.
root@slackware:~# cd /tmp/linux-rpi/
root@slackware:/tmp/linux-rpi#
```

We use '–depth=1' option to only download the latest commit and not the entire branch content, which may currently be +10GB in size. The actual disk space amount used for the source that's just been downloaded is:

```
root@slackware:/tmp/linux-rpi# du -sh /tmp/linux-rpi/
```

```
1.2G    /tmp/linux-rpi/
```

To verify which major/minor revision of the Linux kernel source you're working with, use the 'head' command on the 'Makefile', like this:

```
root@slackware:/tmp/linux-rpi# head -5 Makefile
# SPDX-License-Identifier: GPL-2.0
VERSION = 5
PATCHLEVEL = 4
SUBLEVEL = 83
EXTRAVERSION =
root@slackware:/tmp/linux-rpi#
```

At the time of writing, this is kernel 5.4.83 source from the Raspberry Pi Github repository 'master' branch. You can download and work with other branches by simply maniplulating the 'git' command options. Right now the downloaded kernel source is in a “raw state” (i.e. it includes no settings or configurations) and it's not going to build anything until some form of instructions exist regarding specifically how/what/which to compile.

What's missing now is a kernel '.config' - a file that includes instructions (i.e. settings) that's an essential prerequisite in order to compile the Linux source into a resulting kernel image file - and you won't be doing much without creating one first. This is much easier said than done if you're starting from scratch, because it requires intimate knowledge about the the computer/device hardware that you're building the kernel to support. However, with the Raspberry Pis (and many other ARM devices) there is such a thing as a 'default config', or 'defconfig', that can be called to build the kernel '.config' file for you and save a lot of time and effort (and unfortunately a huge amount of education and experience is also wasted in this process). So in the interest of saving time, let's do it the easy and lazy way for this example…

For the Raspberry Pi 3 to build a default kernel '.config' file for Aarch64 architecture:

```
root@slackware:/tmp/linux-rpi# make ARCH=arm64 CROSS_COMPILE=aarch64-linux-
bcm2709_defconfig
```

For the Raspberry Pi 4 to build a default kernel '.config' file for Aarch64 architecture:

```
root@slackware:/tmp/linux-rpi# make ARCH=arm64 CROSS_COMPILE=aarch64-linux-
bcm2711_defconfig
```

OK! So, normally you would just use 'make <deviceID>_defconfig' to achieve this but here it's the 'ARCH=arm64 CROSS_COMPILE=aarch64-linux-' that makes ALL the difference. You should now have a '.config' file in your Linux source directory which reflects that. Check it out with the 'head' command:

```
root@slackware:/tmp/linux-rpi# head -10 .config
#
# Automatically generated file; DO NOT EDIT.
# Linux/arm64 5.4.83 Kernel Configuration
```

```
#

#
# Compiler: aarch64-linux-gcc (GCC) 10.3.0
#
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=100300
root@slackware:/tmp/linux-rpi#
```

If this is what you see then it's perfect and ready to compile your kernel. In exactly the same way, now 'make' the kernel image, BUT... invoke 'multijobs' as well. Multijobs is the '-j4' option. The '-j<number>' will use more than just a single core of the CPU to build and dramatically cut down on the compile time(s) involved. The RPi3 and RPi4 have quad-core CPUs so it's quite safe to use four cores here, like this:

```
root@slackware:/tmp/linux-rpi# make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-
linux- Image
```

Incidentally, and very logically, it's the same structure and options to compile the DTBs, kernel-headers, modules, etc. and install them:

```
root@slackware:/tmp/linux-rpi# make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-
linux- dtbs
```

```
root@slackware:/tmp/linux-rpi# make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-
linux- headers_install
```

```
root@slackware:/tmp/linux-rpi# make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-
linux- modules
```

```
root@slackware:/tmp/linux-rpi# make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-
linux- modules_install
```

NB: Under normal circumstances you wouldn't compile and install the kernel-headers (as they are already included in the kernel) unless you're intending to build external modules or “outside of source-tree” software. If you don't understand what this means or what it entails, then skip the kernel-headers completely.

BEFORE CONTINUING - Make a back-up FIRST of your '/boot/' directory before coping any new files to it, in case things don't go exactly as planned. Only reboot once everything is correctly in place and you have your backup(s) to fall-back on.

So with everything compiled and built, you just need to copy the 'kernel' image, 'DTBs' and 'System.map' to your '/boot' directory.

NB: By default, on Raspberry Pi ARMv7 [32-bit] systems, the kernel image is named 'kernel7.img' on the RPi3, and for the RPi4 the kernel image is named 'kernel7l.img' -

where the “L” is actually an indicator for “Large Physical Address Extension (LPAE)”. However, on Raspberry Pi ARMv8 [64-bit] systems the kernel is named 'kernel8.img'. This filename MUST be correct in order for the boot-loader to find and run the ARMv8 [64-bit] kernel! If you have specified a custom kernel image filename in the '/boot/config.txt' file then you already know what you're doing here.

```
root@slackware:/tmp/linux-rpi# cp -avr arch/arm64/boot/Image
/boot/kernel8.img"
root@slackware:/tmp/linux-rpi# cp -avr
arch/arm64/boot/dts/broadcom/bcm27*.dtb /boot/
root@slackware:/tmp/linux-rpi# cp -avr System.map /boot/System.map
```

That's it! When you reboot your system it should now be using a 64bit kernel and modules. One should be mindful that Slackware Aarch64 is in development and building kernels using a cross-compiler will soon become superfluous on that system architecture, but for 32bit systems and users a cross-compiler may still be relevant.

I hope this Aarch64 cross-compiler build script and/or information will be useful for Slackware [ARM] users in whatever ways are possible. Any questions or assistance can be addressed via the Linux Questions forum(s)

Thanks very much for your interest in this Aarch64 cross-compiler on the Raspberry Pi.

Sources

If you need to install any of the pre-requisite software here are the links [**NB: check for updates!**]:

ftp://ftp.arm.slackware.com/slackwarearm/slackwarearm-current/slackware/a/ # Slackware ARM current - gawk package.
ftp://ftp.arm.slackware.com/slackwarearm/slackwarearm-current/slackware/d/ # Slackware ARM current - git package.
ftp://ftp.arm.slackware.com/slackwarearm/slackwarearm-current/slackware/d/ # Slackware ARM current - bison package.
ftp://ftp.arm.slackware.com/slackwarearm/slackwarearm-current/slackware/d/ # Slackware ARM current - flex package.
https://www.github.com/raspberrypi/ # Raspberry Pi Foundation GitHub repository Linux kernel and boot-firmware source.
https://ftp.gnu.org/gnu/ # gcc, binutils, glibc, gmp, mpc, mpfr package source.
ftp://gcc.gnu.org/pub/gcc/infrastructure # cloog, isl package source.

Documentation which assisted in this guide:

http://arm.slackware.com/FAQs # Slackware ARM Linux Project Frequently Asked Questions.
http://wiki.osdev.org/GCC_Cross-Compiler # gcc cross-compiler documentation.

Slackware ARM GCC aarch64-linux cross-compiler for the Raspberry Pi.
https://www.raspberrypi.org/documentation/linux/kernel # Raspberry Pi Linux kernel documentation.

* Originally written by Exaga - 2020-12-31 18:02:17 [GMT]

howtos, slackware, raspberry, pi, arm, aarch64, arm64, armv8, cross-compile, author exaga

From:
https://docs.slackware.com/ - **SlackDocs**

Permanent link:
https://docs.slackware.com/howtos:hardware:arm:gcc-10.x_aarch64_cross-compiler

Last update: **2021/10/01 09:26 (UTC)**