# armv7-unknown-linux-uclibceabi

**Tier: 3**

This target supports ARMv7 softfloat CPUs and uses the uclibc-ng standard library. This is a common configuration on many consumer routers (e.g., Netgear R7000, Asus RT-AC68U).

## Target maintainers

- @lancethepants

## Requirements

This target is cross compiled, and requires a cross toolchain.

This target supports host tools and std.

## Building the target

You will need to download or build a `'C'` cross toolchain that targets ARMv7 softfloat and that uses the uclibc-ng standard library. If your target hardware is something like a router or an embedded device, keep in mind that manufacturer supplied SDKs for this class of CPU could be outdated and potentially unsuitable for bootstrapping rust.

Here is a sample toolchain that is built using buildroot. It uses modern toolchain components, older thus universal kernel headers (2.6.36.4), and is used for a project called Tomatoware. This toolchain is patched so that its sysroot is located at /mmc (e.g., /mmc/bin, /mmc/lib, /mmc/include). This is useful in scenarios where the root filesystem is read-only but you are able attach external storage loaded with user applications. Tomatoware is an example of this that even allows you to run various compilers and developer tools natively on the target device.

Utilizing the Tomatoware toolchain this target can be built for cross compilation and native compilation (host tools) with project

rust-bootstrap-armv7-unknown-linux-uclibceabi.

Here is a sample config if using your own toolchain.

```
[build]
build-stage = 2
target = ["armv7-unknown-linux-uclibceabi"]

[target.armv7-unknown-linux-uclibceabi]
cc = "/path/to/arm-unknown-linux-uclibcgnueabi-gcc"
cxx = "/path/to/arm-unknown-linux-uclibcgnueabi-g++"
ar = "path/to/arm-unknown-linux-uclibcgnueabi-ar"
ranlib = "path/to/arm-unknown-linux-uclibcgnueabi-ranlib"
linker = "/path/to/arm-unknown-linux-uclibcgnueabi-gcc"
```

## Building Rust programs

The following assumes you are using the Tomatoware toolchain and environment. Adapt if you are using your own toolchain.

### Native compilation

Since this target supports host tools, you can natively build rust applications directly on your target device. This can be convenient because it removes the complexities of cross compiling and you can immediately test and deploy your binaries. One downside is that compiling on your ARMv7 CPU will probably be much slower than cross compilation on your x86 machine.

To setup native compilation:

- Download Tomatoware to your device using the latest nightly release found here.
- Extract `tar zxvf arm-soft-mmc.tgz -C /mmc`
- Add `/mmc/bin:/mmc:sbin/` to your PATH, or `source /mmc/etc/profile`
- `apt update && apt install rust`

If you bootstrap rust on your own using the project above, it will create a .deb file that you then can install with

```
dpkg -i rust_1.xx.x-x_arm.deb
```

After completing these steps you can use rust normally in a native environment.

### Cross Compilation

To cross compile, you'll need to:

- Build the rust cross toochain using rust-bootstrap-armv7-unknown-linux-uclibceabi or your own built toolchain.
- Link your built toolchain with

  ```
  rustup toolchain link stage2 \
  ${HOME}/rust-bootstrap-armv7-unknown-linux-
  uclibceabi/src/rust/rust/build/x86_64-unknown-linux-gnu/stage2
  ```

- Build with:

  ```
  CC_armv7_unknown_linux_uclibceabi=/opt/tomatoware/arm-soft-mmc/bin/arm-linux-
  gcc \
  CXX_armv7_unknown_linux_uclibceabi=/opt/tomatoware/arm-soft-mmc/bin/arm-
  linux-g++ \
  AR_armv7_unknown_linux_uclibceabi=/opt/tomatoware/arm-soft-mmc/bin/arm-linux-
  ar \
  CFLAGS_armv7_unknown_linux_uclibceabi="-march=armv7-a -mtune=cortex-a9" \
  CXXFLAGS_armv7_unknown_linux_uclibceabi="-march=armv7-a -mtune=cortex-a9" \
  CARGO_TARGET_ARMV7_UNKNOWN_LINUX_UCLIBCEABI_LINKER=/opt/tomatoware/arm-soft-
  mmc/bin/arm-linux-gcc \
  CARGO_TARGET_ARMV7_UNKNOWN_LINUX_UCLIBCEABI_RUSTFLAGS='-Clink-arg=-s -Clink-
  arg=-Wl,--dynamic-linker=/mmc/lib/ld-uClibc.so.1 -Clink-arg=-Wl,-
  rpath,/mmc/lib' \
  cargo +stage2 \
  build \
  --target armv7-unknown-linux-uclibceabi \
  --release
  ```

- Copy the binary to your target device and run.

We specify `CC`, `CXX`, `AR`, `CFLAGS`, and `CXXFLAGS` environment variables because sometimes a project or a subproject requires the use of your `'C'` cross toolchain. Since Tomatoware has a modified sysroot we also pass via RUSTFLAGS the location of the dynamic-linker and rpath.

### Test with QEMU

To test a cross-compiled binary on your build system follow the instructions for `Cross Compilation`, install `qemu-arm-static`, and run with the following.

```
CC_armv7_unknown_linux_uclibceabi=/opt/tomatoware/arm-soft-mmc/bin/arm-linux-gcc \
CXX_armv7_unknown_linux_uclibceabi=/opt/tomatoware/arm-soft-mmc/bin/arm-linux-g++ \
AR_armv7_unknown_linux_uclibceabi=/opt/tomatoware/arm-soft-mmc/bin/arm-linux-ar \
CFLAGS_armv7_unknown_linux_uclibceabi="-march=armv7-a -mtune=cortex-a9" \
CXXFLAGS_armv7_unknown_linux_uclibceabi="-march=armv7-a -mtune=cortex-a9" \
CARGO_TARGET_ARMV7_UNKNOWN_LINUX_UCLIBCEABI_LINKER=/opt/tomatoware/arm-soft-
mmc/bin/arm-linux-gcc \
CARGO_TARGET_ARMV7_UNKNOWN_LINUX_UCLIBCEABI_RUNNER="qemu-arm-static -L
/opt/tomatoware/arm-soft-mmc/arm-tomatoware-linux-uclibcgnueabi/sysroot/" \
cargo +stage2 \
run \
--target armv7-unknown-linux-uclibceabi \
--release
```

### Run in a chroot

It's also possible to build in a chroot environment. This is a convenient way to work without needing to access the target hardware.

To build the chroot:

- `sudo debootstrap --arch armel bullseye $HOME/debian`
- `sudo chroot $HOME/debian/ /bin/bash`
- `mount proc /proc -t proc`
- `mount -t sysfs /sys sys/`
- `export PATH=/mmc/bin:/mmc/sbin:$PATH`

From here you can setup your environment (e.g., add user, install wget).

- Download Tomatoware to the chroot environment using the latest nightly release found here.
- Extract `tar zxvf arm-soft-mmc.tgz -C /mmc`
- Add `/mmc/bin:/mmc:sbin/` to your PATH, or `source /mmc/etc/profile`
- `sudo /mmc/bin/apt update && sudo /mmc/bin/apt install rust`

After completing these steps you can use rust normally in a chroot environment.

Remember when using `sudo` the root user's PATH could differ from your user's PATH.