armv7-unknown-linux-uclibceabihf

Tier: 3

This tier supports the ARMv7 processor running a Linux kernel and uClibc-ng standard library. It provides full support for rust and the rust standard library.

Designated Developers

- @skrap

Requirements

This target is cross compiled, and requires a cross toolchain. You can find suitable pre-built toolchains at bootlin or build one yourself via buildroot.

Building

Get a C toolchain

Compiling rust for this target has been tested on `x86_64` linux hosts. Other host types have not been tested, but may work, if you can find a suitable cross compilation toolchain for them.

If you don't already have a suitable toolchain, download one here, and unpack it into a directory.

Configure rust

The target can be built by enabling it for a `rustc` build, by placing the following in `config.toml`:

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

[target.armv7-unknown-linux-uclibceabihf]
# ADJUST THIS PATH TO POINT AT YOUR TOOLCHAIN
cc = "/TOOLCHAIN_PATH/bin/arm-buildroot-linux-uclibcgnueabihf-gcc"
```

Build

```
# in rust dir
./x.py build --stage 2
```

Building and Running Rust Programs

To test cross-compiled binaries on a `x86_64` system, you can use the `qemu-arm` userspace emulation program. This avoids having a full emulated ARM system by doing dynamic binary translation and dynamic system call translation. It lets you run ARM programs directly on your `x86_64` kernel. It's very convenient!

To use:

- Install `qemu-arm` according to your distro.

- Link your built toolchain via:
  - `rustup toolchain link stage2 ${RUST}/build/x86_64-unknown-linux-gnu/stage2`
- Create a test program

```
cargo new hello_world
cd hello_world
```

- Build and run

```
CARGO_TARGET_ARMV7_UNKNOWN_LINUX_UCLIBCEABIHF_RUNNER="qemu-arm -L ${TOOLCHAIN}/arm-buildroot-linux-uclibcgnueabihf/sysroot/" \
CARGO_TARGET_ARMV7_UNKNOWN_LINUX_UCLIBCEABIHF_LINKER=${TOOLCHAIN}/bin/arm-buildroot-linux-uclibcgnueabihf-gcc \
cargo +stage2 run --target armv7-unknown-linux-uclibceabihf
```