Setting up Rust cross-compilation environment for BeagleBone

Table of contents:

Guide:

This guide will walk you through setting up an embedded Rust development environment for Beaglebone and compiling “Hello World” audio program. It assumes that NixOs 19.09 is installed. Alternatively one can use one of the virtual images from the following link [1]. Also, audio should be enabled on a target device. I found that if Rust project needs a library I was able to get it to work on this operating system.

Let’s get started. Enter shell and execute the following commands.

1. Create a directory of your choice and place *default.nix* into it.
2. Enter Rust environment:

   nix-shell

3. Follow these steps only when entering the environment for the first time.

   a. Configure Cargo:

   mkdir -p ~/.cargo

   $ cat >>~/.cargo/config <<EOF
   > [target.armv7-unknown-linux-gnueabihf]
   > linker = "arm-linux-gnueabihf-gcc"
   > EOF

   b. Install Beaglebone target:

   rustup target add armv7-unknown-linux-gnueabihf

   c. Set up Rust:

   rustup default stable

4. Follow these steps if Rust project requires an additional library. For example our project will require *alsalibasound-dev:*

   a. Check to see if the library is available in the repository [3]. If not, you will have to write nix expression yourself. Most common libraries, however, should be available.
   b. Add library to the nativeBuildInputs in default.nix file. For example *beaglebone_arm.alsaLib*

c. Add the lines to the following file:

/home/<your name>/.config/nixpkgs/config.nix :

```
{
        allowUnfree = true;
        allowUnsupportedSystem = true;
}
```

5. Create new hello project:

   cargo new --bin hello && cd hello

6. Add the line *alsa = "0.4.1"* to *cargo.toml* under *[dependencies]* section
7. Replace the contents of main.rs with the one below or copy the lines from the following link [4].
8. Run Cargo build:

   cargo build --target=armv7-unknown-linux-gnueabihf

9. Binary *hello* is produced in the following directory:

   target/armv7-unknown-linux-gnueabihf/debug/

10. Because compilation was done on NixOS, the binary has to be patched using the following command:

    patchelf --set-interpreter /lib/ld-linux-armhf.so.3 <path to binary>/hello

11. Copy binary to Beaglebone and run it. The output should generate sine audio wave.

References:

1. https://nixos.org/nixos/download.html
2. https://github.com/japaric/rust-cross
3. https://nixos.org/nixos/packages.html?channel=nixos-19.09
4. https://docs.rs/alsa/0.4.1/alsa/pcm/index.html

default.nix

```
with import <nixpkgs> {};
let
beaglebone_arm = import <nixpkgs> {
        crossSystem = {
        system = "beaglebone";
        config = "arm-linux-gnueabihf";
        platform = (import <nixpkgs/lib>).systems.platforms.beaglebone;
};
};
in
mkShell {

  name = "rust-env";

  nativeBuildInputs = with buildPackages; [
        vscode # visual studio code
        rustup
        beaglebone_arm.stdenv.cc
        beaglebone_arm.alsaLib # libasound2-dev.so
        pkg-config
        git
  ];
  buildInputs = [];

  PKG_CONFIG_ALLOW_CROSS=1;

  shellHook = ''
        export RUSTFLAGS="-C linker=arm-linux-gnueabihf-gcc"
        export RUST_BACKTRACE="1"
        '';
}
```

main.rs

```
use alsa::{Direction, ValueOr};
use alsa::pcm::{PCM, HwParams, Format, Access, State};

fn main() {
	// Open default playback device
let pcm = PCM::new("default", Direction::Playback, false).unwrap();

// Set hardware parameters: 44100 Hz / Mono / 16 bit
let hwp = HwParams::any(&pcm).unwrap();
hwp.set_channels(1).unwrap();
hwp.set_rate(44100, ValueOr::Nearest).unwrap();
hwp.set_format(Format::s16()).unwrap();
hwp.set_access(Access::RWInterleaved).unwrap();
pcm.hw_params(&hwp).unwrap();
let io = pcm.io_i16().unwrap();

// Make sure we don't start the stream too early
let hwp = pcm.hw_params_current().unwrap();
let swp = pcm.sw_params_current().unwrap();
swp.set_start_threshold(hwp.get_buffer_size().unwrap() -
hwp.get_period_size().unwrap()).unwrap();
pcm.sw_params(&swp).unwrap();

// Make a sine wave
let mut buf = [0i16; 1024];
for (i, a) in buf.iter_mut().enumerate() {
	*a = ((i as f32 * 2.0 * ::std::f32::consts::PI / 128.0).sin() * 8192.0) as i16
}

// Play it back for 2 seconds.
for _ in 0..2*44100/1024 {
	assert_eq!(io.writei(&buf[..]).unwrap(), 1024);
}

// In case the buffer was larger than 2 seconds, start the stream manually.
if pcm.state() != State::Running { pcm.start().unwrap() };
// Wait for the stream to finish playback.
pcm.drain().unwrap();
}
```