Cross compiling with rust

Patrick J. Pereira

patrick@bluerobotics.com

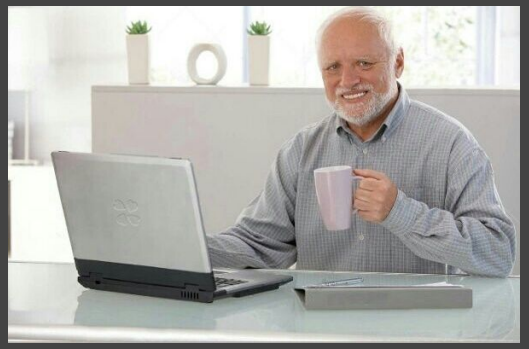
patrickjp@kde.org

- Definitions
- Cross compiling ?
- Targets
- glibc vs musl
- The main machine
- The other machine
- Cross compiling with rust
- Examples

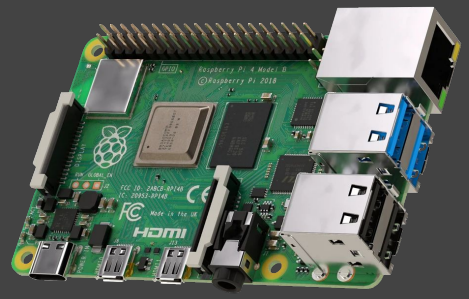
Everything in this talk will be a TLDR version.

Definitions

- **Host**: Your desktop or working environment.



- **Target:** Your embedded device or the machine that will run the code that does **not** uses the same **architecture**.



Definitions

- **Compiling**: Create binaries from the source code.

```
fn main() {
    println!("Hello, world!");
}
```

```
Contents of section .text.main:
 0000 4883ec18 8a050000 00004863 cf488d3d  H.........Hc.H.=
 0010 00000000 48897424 104889ce 488b5424  ....H.t$.H..H.T$
 0020 10884424 0fe80000 00004883 c418c3    ..D$......H....
Contents of section .rodata..L__unnamed_3:
 0000 48656c6c 6f2c2077 6f726c64 210a      Hello, world!.
Contents of section .data.rel.ro..L__unnamed_1:
 0000 00000000 00000000 0e000000 00000000  ................
Contents of section .debug_gdb_scripts:
 0000 01676462 5f6c6f61 645f7275 73745f70  .gdb_load_rust_p
 0010 72657474 795f7072 696e7465 72732e70  retty_printers.p
 0020 7900                                 y.
Contents of section .debug_str:
 0000 636c616e 67204c4c 564d2028 72757374  clang LLVM (rust
```

- Opcode: Operation code or human friendly representation of the CPU instruction.

```
Disassembly of section .interp:

00000000000002e0 <.interp>:
 2e0:   2f                      (bad)
 2e1:   6c                      insb   (%dx),%es:(%rdi)
 2e2:   69 62 36 34 2f 6c 64    imul   $0x646c2f34,0x36(%rdx),%esp
 2e9:   2d 6c 69 6e 75          sub    $0x756e696c,%eax
 2ee:   78 2d                   js     31d <_ZN3std9panicking18update
 2f0:   78 38                   js     32a <_ZN3std9panicking18update
 2f2:   36 2d 36 34 2e 73       ss sub $0x732e3436,%eax
 2f8:   6f                      outsl  %ds:(%rsi),(%dx)
 2f9:   2e 32 00                xor    %cs:(%rax),%al
```

```
Disassembly of section .text:

00010100 <elf_try_debugfile>:
   10100:   e92d 4ff0   stmdb   sp!, {r4, r5, r6, r7, r
   10104:   b08b        sub sp, #44 ; 0x2c
   10106:   4c29        ldr r4, [pc, #164]  ; (101ac <e
   10108:   4680        mov r8, r0
   1010a:   4617        mov r7, r2
   1010c:   4828        ldr r0, [pc, #160]  ; (101b0 <e
   1010e:   9a15        ldr r2, [sp, #84]   ; 0x54
   10110:   447c        add r4, pc
   10112:   9106        str r1, [sp, #24]
   10114:   9307        str r3, [sp, #28]
   10116:   4623        mov r3, r4
   10118:   9204        str r2, [sp, #16]
   1011a:   5822        ldr r2, [r4, r0]
   1011c:   9b14        ldr r3, [sp, #80]   ; 0x50
   1011e:   9804        ldr r0, [sp, #16]
   10120:   18fb        adds    r3, r7, r3
   10122:   9303        str r3, [sp, #12]
```

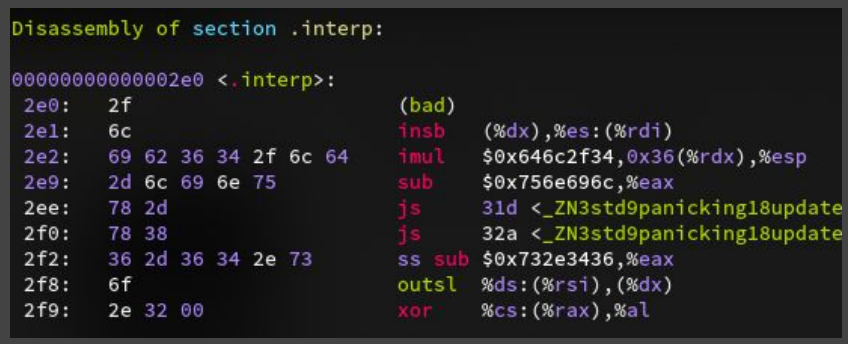
- **objdump**: Display information from object files.

- **strings**: Search printable characters from object files.

- Cross compiling ?

x86_64

GCC



```
Disassembly of section .interp:

00000000000002e0 <.interp>:
 2e0:   2f                      (bad)
 2e1:   6c                      insb   (%dx),%es:(%rdi)
 2e2:   69 62 36 34 2f 6c 64    imul   $0x646c2f34,0x36(%rdx),%esp
 2e9:   2d 6c 69 6e 75          sub    $0x756e696c,%eax
 2ee:   78 2d                   js     31d <_ZN3std9panicking18update
 2f0:   78 38                   js     32a <_ZN3std9panicking18update
 2f2:   36 2d 36 34 2e 73       ss sub $0x732e3436,%eax
 2f8:   6f                      outsl  %ds:(%rsi),(%dx)
 2f9:   2e 32 00                xor    %cs:(%rax),%al
```

```
fn main() {
    println!("Hello, world!");
}
```

- Cross compiling ?

x86_64

```
Disassembly of section .interp:

00000000000002e0 <.interp>:
 2e0:   2f                      (bad)
 2e1:   6c                      insb   (%dx),%es:(%rdi)
 2e2:   69 62 36 34 2f 6c 64    imul   $0x646c2f34,0x36(%rdx),%esp
 2e9:   2d 6c 69 6e 75          sub    $0x756e696c,%eax
 2ee:   78 2d                   js     31d <_ZN3std9panicking18update
 2f0:   78 38                   js     32a <_ZN3std9panicking18update
 2f2:   36 2d 36 34 2e 73       ss sub $0x732e3436,%eax
 2f8:   6f                      outsl  %ds:(%rsi),(%dx)
 2f9:   2e 32 00                xor    %cs:(%rax),%al
```

GCC

ARMv7

```
fn main() {
    println!("Hello, world!");
}
```

ARM-GCC

```
Disassembly of section .text:

00010100 <elf_try_debugfile>:
   10100:   e92d 4ff0   stmdb   sp!, {r4, r5, r6, r7, r
   10104:   b08b        sub sp, #44 ; 0x2c
   10106:   4c29        ldr r4, [pc, #164]  ; (101ac <e
   10108:   4680        mov r8, r0
   1010a:   4617        mov r7, r2
   1010c:   4828        ldr r0, [pc, #160]  ; (101b0 <e
   1010e:   9a15        ldr r2, [sp, #84]   ; 0x54
   10110:   447c        add r4, pc
   10112:   9106        str r1, [sp, #24]
   10114:   9307        str r3, [sp, #28]
   10116:   4623        mov r3, r4
   10118:   9204        str r2, [sp, #16]
   1011a:   5822        ldr r2, [r4, r0]
   1011c:   9b14        ldr r3, [sp, #80]   ; 0x50
   1011e:   9804        ldr r0, [sp, #16]
   10120:   18fb        adds    r3, r7, r3
   10122:   9303        str r3, [sp, #12]
```

Cross compiling ?

- Why can’t you compile in your target computer ?


```
# AMD Ryzen 7 2700 Eight-Core Processor
# 16 threads, 16GB Ram

cargo build
Finished dev [unoptimized + debuginfo] target(s) in  1m 48s


# ARMv7 Processor rev 4 (v7l)
# 4 threads, 1GB Ram
Finished dev [unoptimized + debuginfo] target(s) in  33m 56s
```

- Targets
  - List of available targets

rustup target list

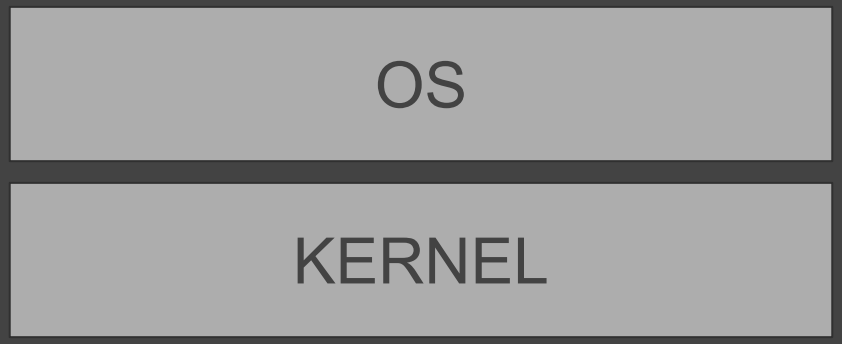
!! | wc -l # 81

glibc vs musl

- Both are low-level libraries

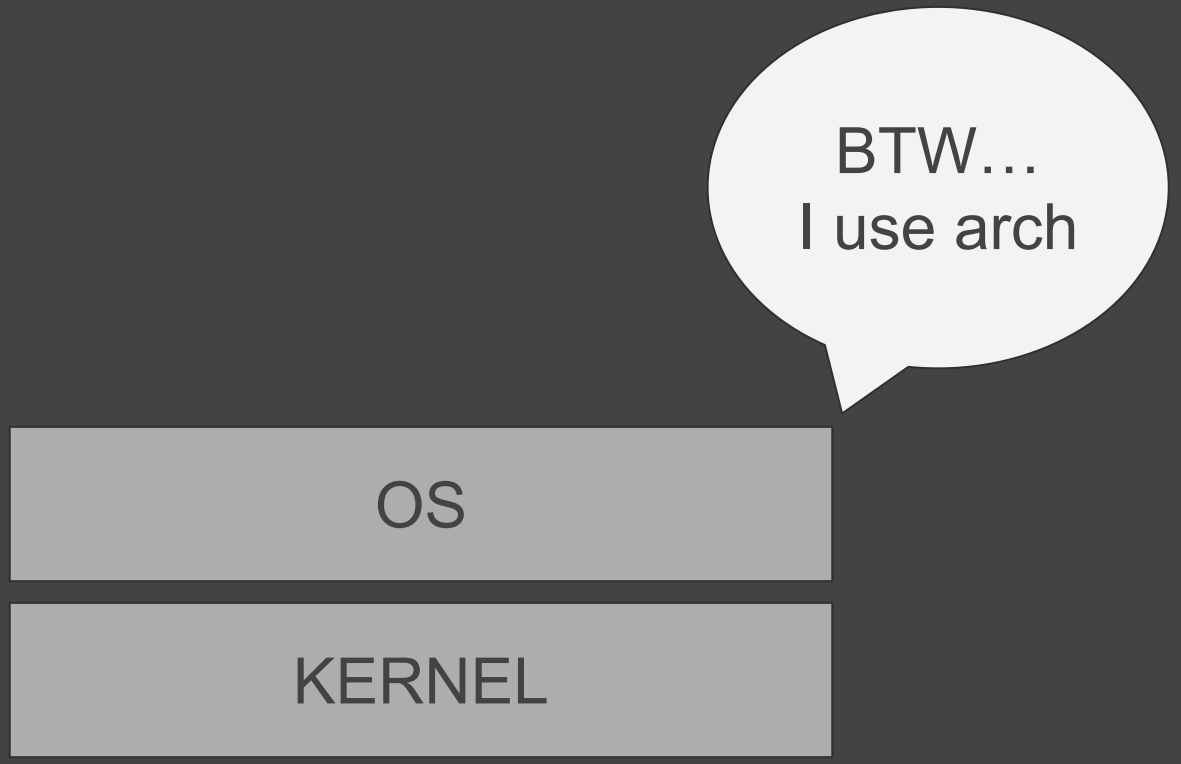
glibc vs musl

- Both are low-level libraries



glibc vs musl

- Both are low-level libraries



glibc vs musl

- Both are low-level libraries

glibc vs musl

- Both are low-level libraries

glibc vs musl

- Both are low-level libraries

- glibc is backward compatible but not forward compatible
  - If you compile with glibc2 it'll run in a system with glibc6
  - If you compile with glibc6 it'll **not** run in a system with glib2
  - Linux Appimages should be compatible with the standard deployment OS [CentOS 6 (2011)]

glibc vs musl

- Both are low-level libraries

- glibc is backward compatible but not forward compatible
  - If you compile with glibc2 it'll run in a system with glibc6
  - If you compile with glibc6 it'll **not** run in a system with glib2
  - Linux Appimages should be compatible with the standard deployment OS [CentOS 6 (2011)]

- musl is a new standard library
- It's backward and forward compatible
- There is more advantages that you can check here:
  - https://www.musl-libc.org/

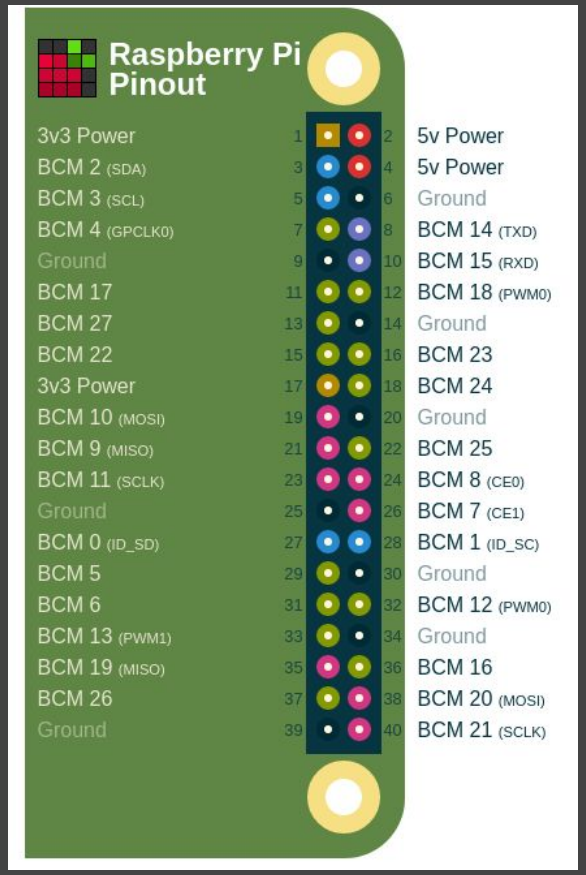
- The main machine

macbook pro 2017

The other machine

- Raspberry Pi 3B

```
✘ alarm@alarmpi ~ uname -m
armv7l
```



Cross compiling with rust

1. Install rust!
   a. https://rustup.rs/
2. Create a new project
   a. cargo new cross
3. Build the project locally
   a. cargo build
4. Run it!
   a. cargo run

HOST MACHINE

Cross compiling with rust

1. Install rust!
   a. https://rustup.rs/
2. Create a new project
   a. cargo new cross
3. Build the project locally
   a. cargo build
4. Run it!
   a. cargo run

HOST MACHINE

Note: linker not found = no compiler in your machine
Try to install GCC

Cross compiling with rust

1. Install rust-std for our target
   a. rustup target add armv7-unknown-linux-gnueabihf
2. Build the project locally
   a. cargo build --target armv7-unknown-linux-gnueabihf

TARGET MACHINE

• Cross compiling with rust

1. Install rust-std for our target
   a. rustup target add armv7-unknown-linux-gnueabihf
2. Build the project locally
   a. cargo build --target armv7-unknown-linux-gnueabihf

TARGET MACHINE

```
error: linking with `cc` failed: exit code: 1
```

Cross compiling with rust

1. Install rust-std for our target
   a. rustup target add armv7-unknown-linux-gnueabihf
2. Build the project locally
   a. cargo build --target armv7-unknown-linux-gnueabihf
3. Download the compiler
   a. https://www.linaro.org/downloads/-arm-linux-gnueabihf[bin]
4. Make it visible
   a. export PATH=$PATH:BIN_FOLDER
5. Edit cargo configuration file
   a. $EDITOR ~/.cargo/config
      [target.armv7-unknown-linux-gnueabihf]
      linker = "arm-linux-gnueabihf-gcc"
6. Build it 🎶 *one more time* 🎶
   a. cargo build --target armv7-unknown-linux-gnueabihf

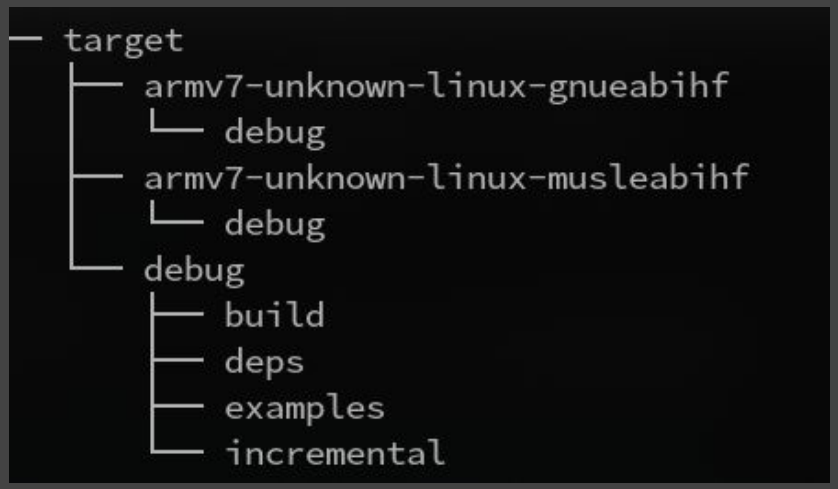
TARGET MACHINE

Cross compiling with rust

1. Install rust-std for our target
   a. rustup target add armv7-unknown-linux-**musleabihf**
2. Download the compiler
   a. https://musl.cc/ - armv7l-linux-musleabihf-cross.tgz
3. Make it visible
   a. export PATH=$PATH:BIN_FOLDER
4. Edit cargo configuration file
   a. $EDITOR ~/.cargo/config
      [target.armv7-unknown-linux-**musleabihf**]
      linker = "armv7l-linux-**musleabihf**-gcc"
5. Build the project locally
   a. cargo build --target armv7-unknown-linux-**musleabihf**

TARGET MACHINE

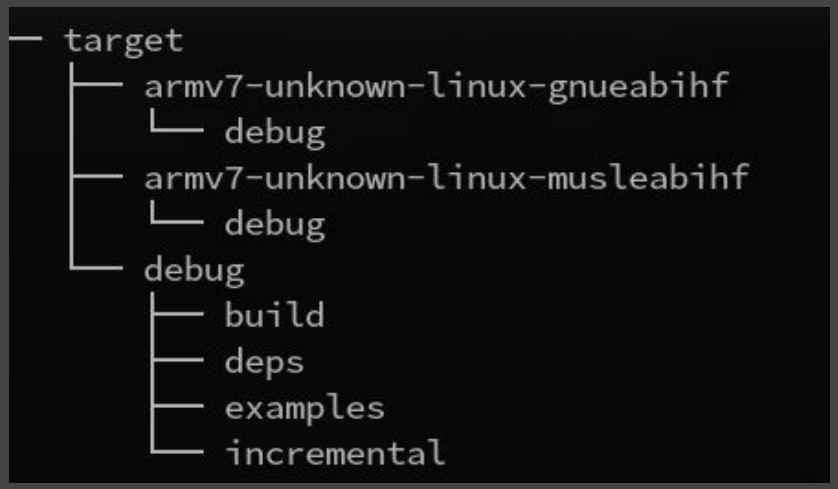
Cross compiling with rust



1. cargo build
   a. file executable:
      i. cross: **ELF 64-bit LSB** pie executable, **x86-64**, version 1 (SYSV), dynamically linked, interpreter…
   b. ldd executable:

```
linux-vdso.so.1 (0x00007ffca0df7000)
libdl.so.2 => /usr/lib/libdl.so.2 (0x00007fe61751b000)
libpthread.so.0 => /usr/lib/libpthread.so.0 (0x00007fe6174f9000)
libgcc_s.so.1 => /usr/lib/libgcc_s.so.1 (0x00007fe6174df000)
libc.so.6 => /usr/lib/libc.so.6 (0x00007fe617318000)
/lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-linux-x86-64.so.2
(0x00007fe61755d000)
```

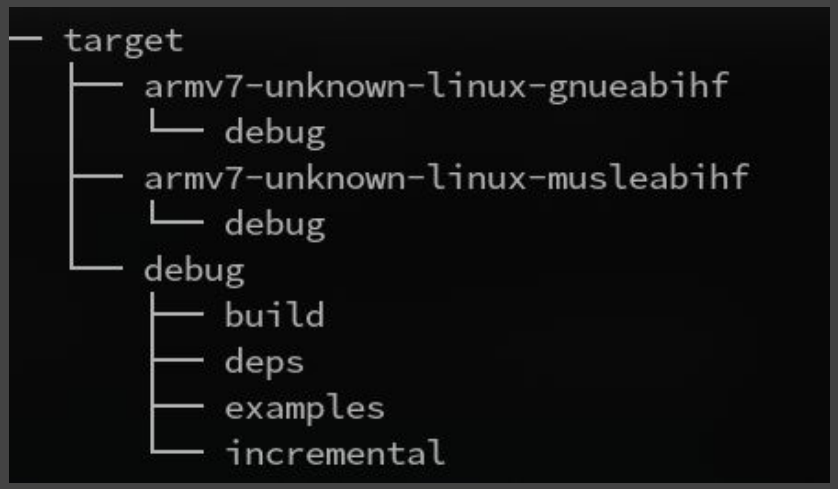
Cross compiling with rust



1. cargo build --target armv7-unknown-linux-gnueabihf
   a. file executable:
      i. cross: **ELF 32-bit LSB** pie executable, **ARM, EABI5** version 1 (SYSV), **dynamically linked**, interpreter ...
   b. ldd executable:

```
linux-vdso.so.1 (0x7eff8000)
libc.so.6 => Not found
/lib/ld-linux-armhf.so.3 => /usr/lib/ld-linux-armhf.so.3 (0x76f49000)
libdl.so.2 => /usr/lib/libdl.so.2 (0x76a66000)
libpthread.so.0 => /usr/lib/libpthread.so.0 (0x76a3c000)
libgcc_s.so.1 => /usr/lib/libgcc_s.so.1 (0x76a0f000)
libm.so.6 => Not found
```

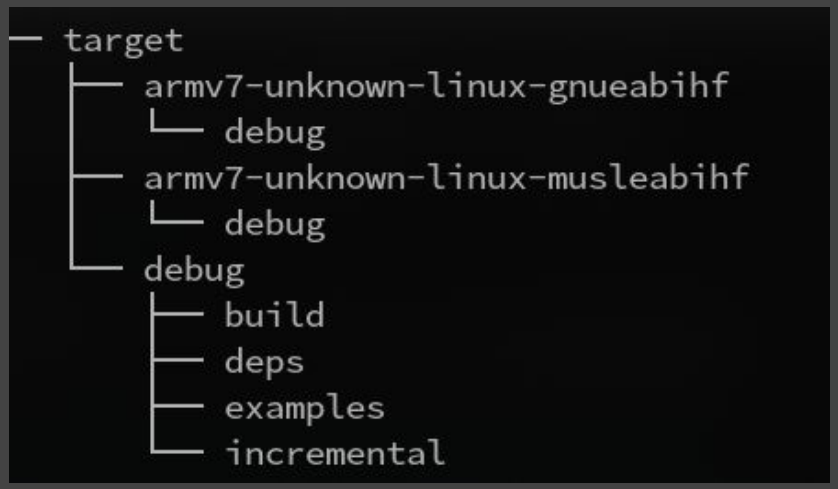
Cross compiling with rust



1. cargo build --target armv7-unknown-linux-gnueabihf
   a. file executable:
      i. cross: **ELF 32-bit LSB** pie executable, **ARM, EABI5** version 1 (SYSV), **dynamically linked**, interpreter ...
   b. ldd executable:

```
linux-vdso.so.1 (0x7eff8000)
libc.so.6 => /usr/lib/libc.so.6 (0x76a79000)
/lib/ld-linux-armhf.so.3 => /usr/lib/ld-linux-armhf.so.3 (0x76f49000)
libdl.so.2 => /usr/lib/libdl.so.2 (0x76a66000)
libpthread.so.0 => /usr/lib/libpthread.so.0 (0x76a3c000)
libgcc_s.so.1 => /usr/lib/libgcc_s.so.1 (0x76a0f000)
libm.so.6 => /usr/lib/libm.so.6 (0x769a1000)
```

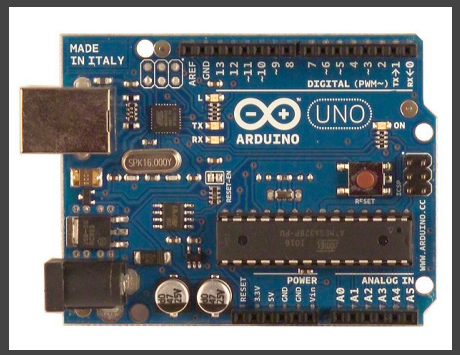
Cross compiling with rust



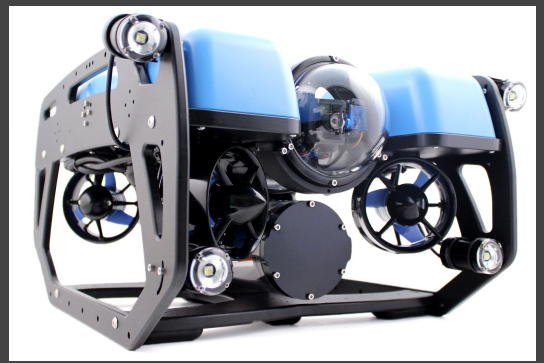
1. cargo build --target armv7-unknown-linux-**musleabihf**
   a. file executable:
      i. cross: **ELF 32-bit LSB** executable, **ARM, EABI5** version 1 (SYSV), **statically linked**, with ...
   b. ldd executable:
      not a dynamic executable

Examples

1. https://github.com/patrickelectric/bridges
   a. A bidirectional Serial-UDP bridge!



2. https://github.com/patrickelectric/mavlink2rest
   a. REST server that provides mavlink information from a mavlink source



- Demo

@patrickelectric

@patrickelectric

@patrickelectric

https://patrickelectric.work

Patrick J. Pereira

patrick@bluerobotics.com

patrickjp@kde.org