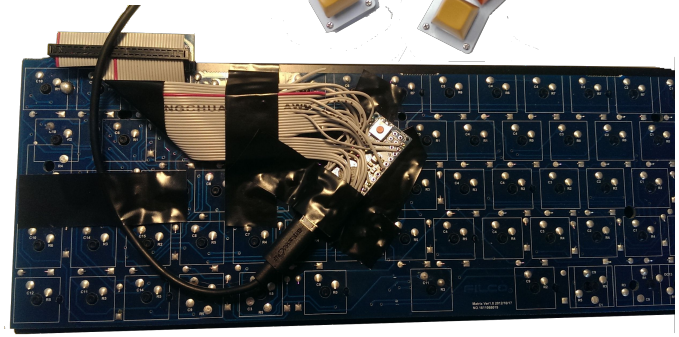
elixir + NERVES +  =

Elixir keyboard!

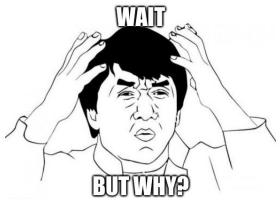
ElixirConf EU 2020

Speaker

- André Albuquerque
  - @amalbuquerque
- Software Engineer @ Hopin
- (Mild) obsession with keyboards

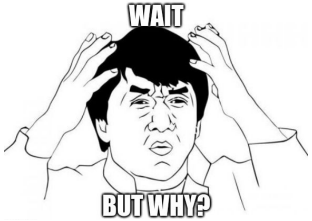


Outline

- Wait, but why? 

- Nerves 101 NERVES
- Baby steps 
- A keyboard on top of the BEAM 
- Pros & Cons 
- Future work BACK TO THE FUTURE
- D E M O time 

Wait, but why?



- Keyboards <3
- Custom firmwares based on C
  - TMK (and its notable QMK fork)

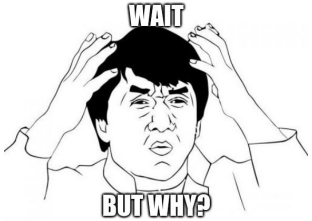
```c
enum minila_layers {
 _QWERTY,   // Qwerty
 _BASIC,    // Basic qwerty
 _F1_LAYER, // Symbols
 _F4_LAYER  // Multimedia
};


enum custom_keycodes {
 CTL_TAB = CTL_T(KC_TAB),
 CTL_BSP = CTL_T(KC_BSPC),
 TTO_FN1 = TT(_F1_LAYER),
 TTO_FN4 = TT(_F4_LAYER),
 SPC_FN1 = LT(_F1_LAYER, KC_SPC),
 APP_FN4 = LT(_F4_LAYER, KC_APP),
 SPC_LGUI = MT(MOD_LGUI, KC_SPC),
 NKROTG = MAGIC_TOGGLE_NKRO
};
```

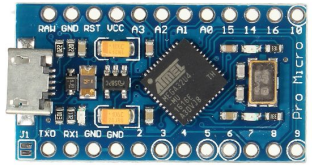
```c
const uint16_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS] = {
 [_QWERTY] = LAYOUT(
   KC_GRV,  KC_1,  KC_2, KC_3, KC_4, KC_5, KC_6, KC_7, KC_8, KC_9, KC_0, KC_MINS, KC_EQL, KC_GRV, KC_BSPC,    \
   KC_ESC,   KC_Q, KC_W, KC_E, KC_R, KC_T, KC_Y, KC_U, KC_I, KC_O, KC_P,    KC_LBRC, KC_RBRC,      KC_BSLS,   \
   CTL_TAB,   KC_A, KC_S, KC_D, KC_F, KC_G, KC_H, KC_J, KC_K, KC_L,   KC_SCLN, KC_QUOT,            KC_ENT,    \
   KC_LSPO,    KC_Z, KC_X, KC_C, KC_V, KC_B, KC_N, KC_M, KC_COMM, KC_DOT, KC_SLSH, KC_RSPC, KC_UP, KC_DELETE,\
   TTO_FN1, KC_LGUI,  KC_LALT, SPC_LGUI,   SPC_FN1,   CTL_BSP, KC_RALT, APP_FN4, KC_LEFT, KC_DOWN, TTO_FN4   \
 ),

 [_F1_LAYER] = LAYOUT(
   // ...
 ),
```

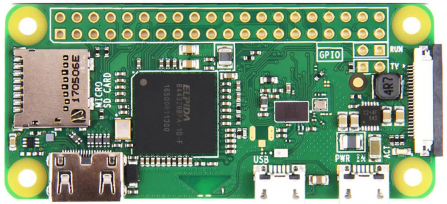
Wait, but why?



- Building the keyboard firmware with C made tweaks more difficult
  1. Update the keymap
  2. Compile the C code with the right toolchain
  3. Burn the firmware to the controller



- Nerves seemed so much fun
  - Actively under development
  - Bringing Elixir to the IoT world
  - Ran on the Raspberry Pi Zero
    - 28 IO pins (!!!)




NERVES
elixir

Nerves 101

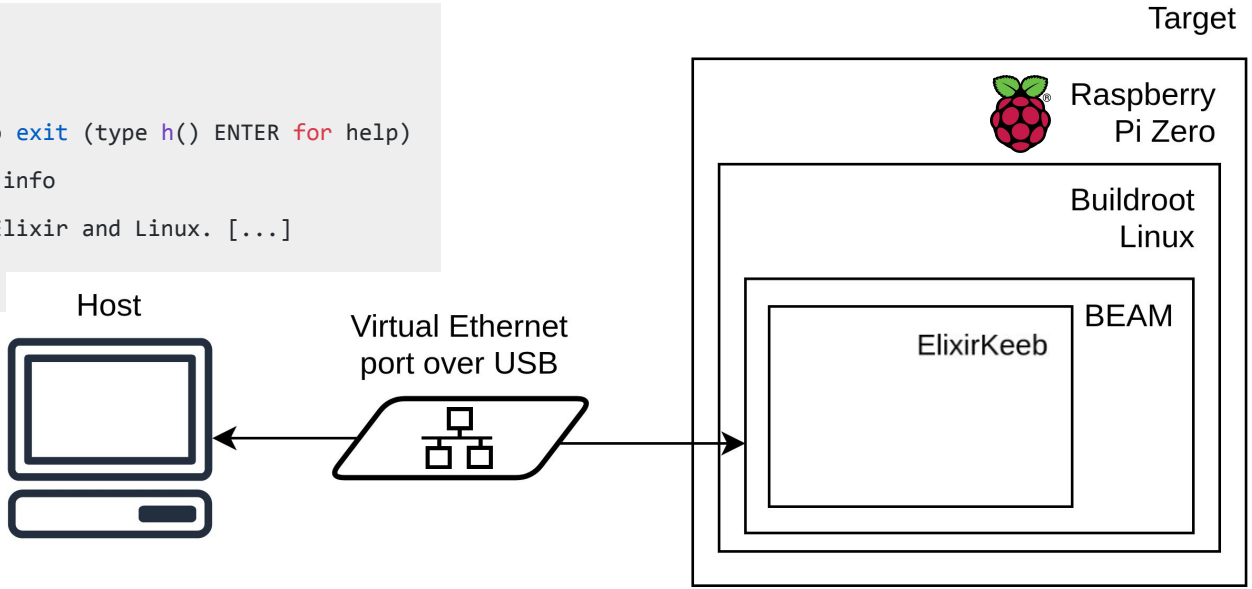
- The Raspberry Pi Zero has an ARM architecture - **target**
  - My laptop has a x86 architecture - **host**
- The right **toolchain** lets me compile code on the **host** that runs on the **target**
  - Our Elixir application needs to be compiled to run on the **target** architecture
- The BEAM needs an Operating System on the **target** to run - **system**
  - Nerves provides a streamlined Linux (Buildroot-based) that will run on the **target**
- The **firmware image** contains:
  - The Buildroot-based Linux
  - Our application compiled for the **target**
  - Everything needed for the **target** system to boot and start our application

Baby steps

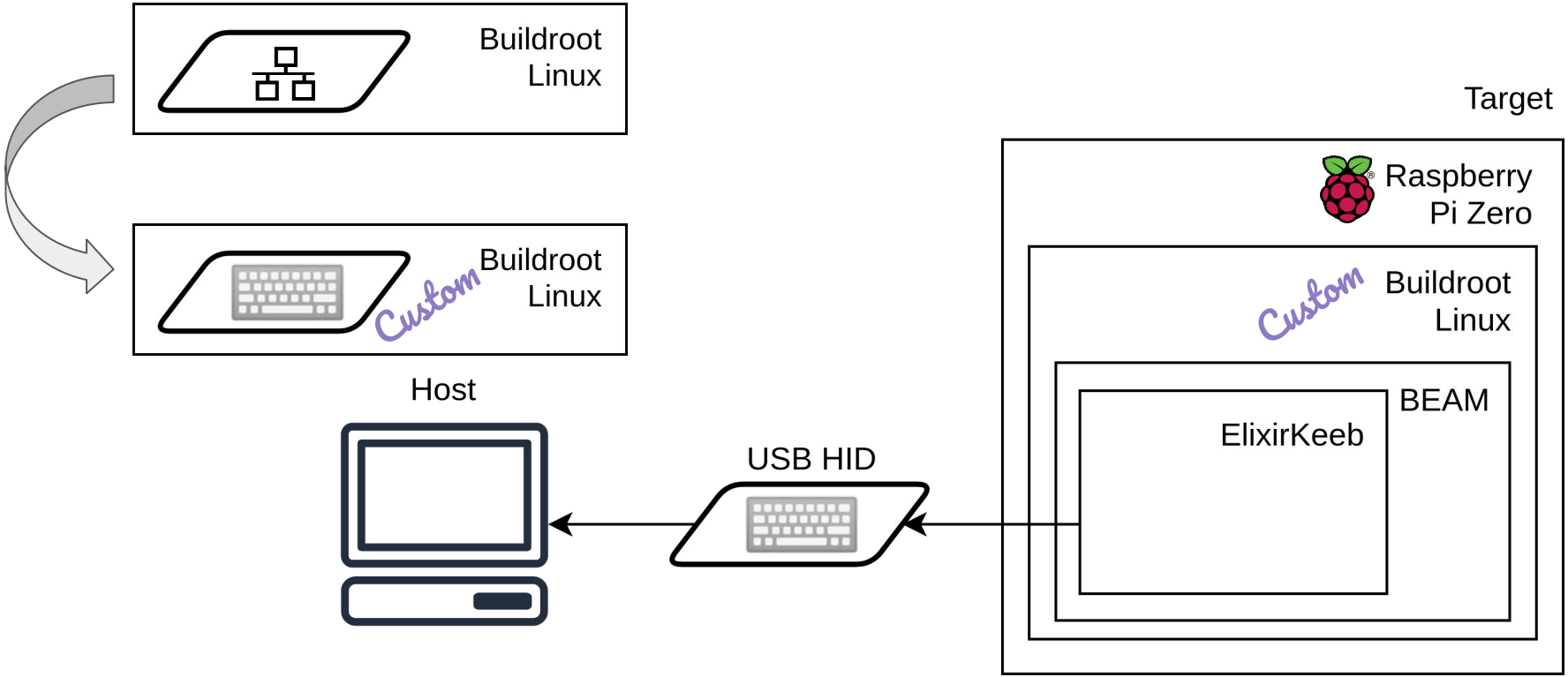
Baby steps (1/4)

- `mix nerves.new elixir_keeb`
- `mix deps.get`
- `MIX_TARGET=rpi0 mix firmware && mix firmware.burn`

```
~/projs/personal/elixir_keeb
❯ ssh nerves.local
Interactive Elixir (1.8.1) - press Ctrl+C to exit (type h() ENTER for help)
Toolshed imported. Run h(Toolshed) for more info
RingLogger is collecting log messages from Elixir and Linux. [...]
iex(elixir_keeb@nerves.local)1> _
```

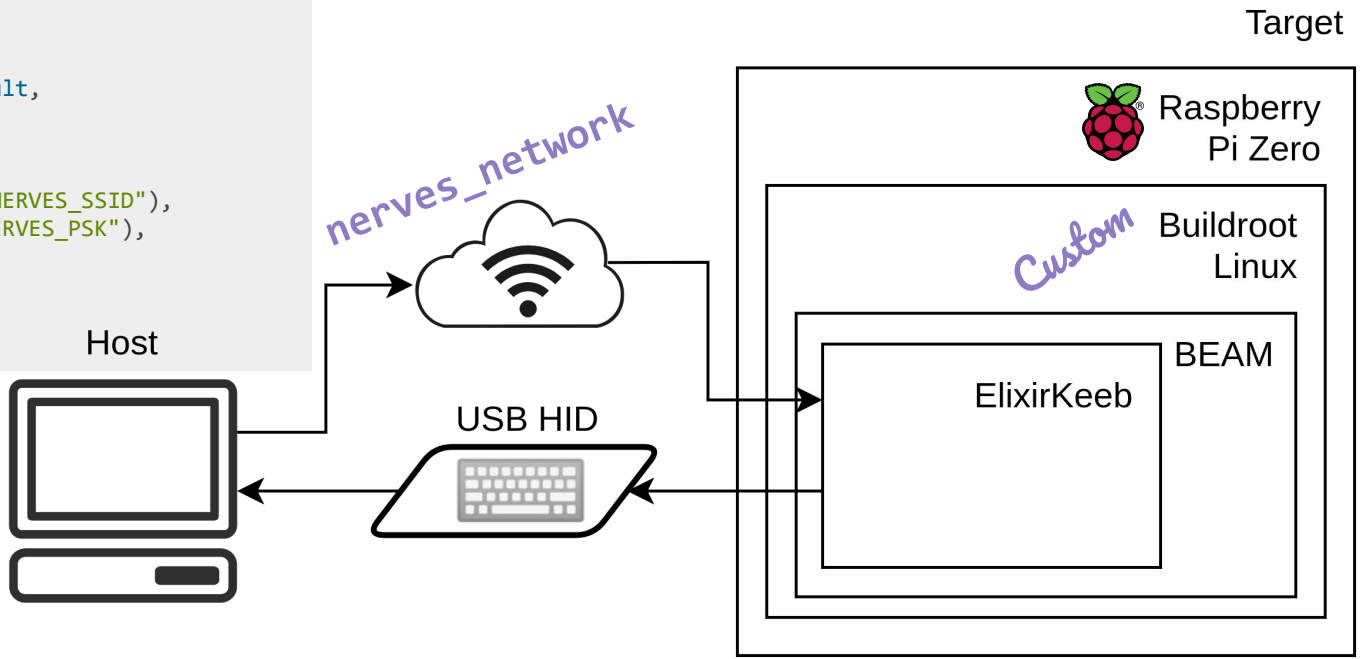


Baby steps (2/4)



Baby steps (3/4)

```
3  config :nerves_network,
4   regulatory_domain: "EU"
5
6  config :nerves_network, :default,
7   wlan0: [
8     networks: [
9       [
10        ssid: System.get_env("NERVES_SSID"),
11        psk: System.get_env("NERVES_PSK"),
12        key_mgmt: :"WPA-PSK"
13      ]
14    ]
15  ]
```




```
NERVES_SSID=... NERVES_PSK=... MIX_TARGET=rpi0 mix firmware
```

Baby steps (4/4)

circuits_gpio

Target

Raspberry Pi Zero

Custom

Buildroot Linux

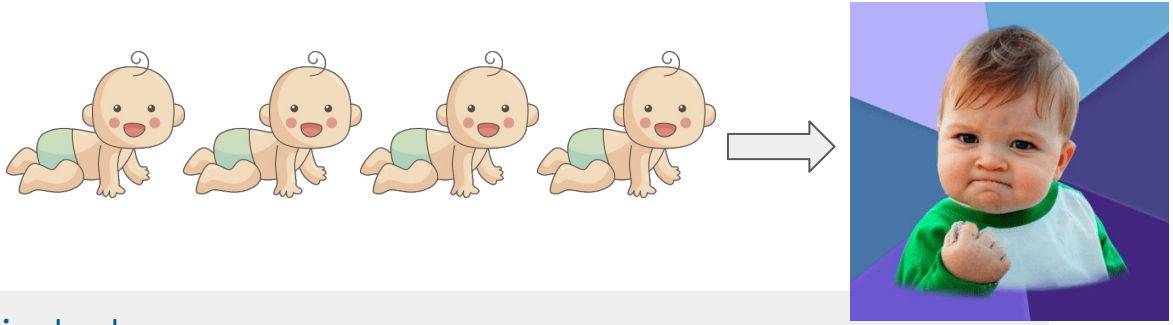
BEAM

ElixirKeeb

Host

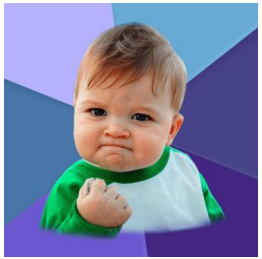
USB HID

Baby steps



```
:elixir_keeb
|> Nerves.mix_new()                          # baby step 1/4
|> custom_build_root_system(fn system -> # baby step 2/4
   system
   |> disable_ethernet_port_over_usb()
   |> enable_usb_hid()
end)
|> configure_nerves_network()                # baby step 3/4
|> use_circuits_gpio()                       # baby step 4/4
|> tada!()
```

Baby steps



A keyboard on top of the BEAM

GenServer

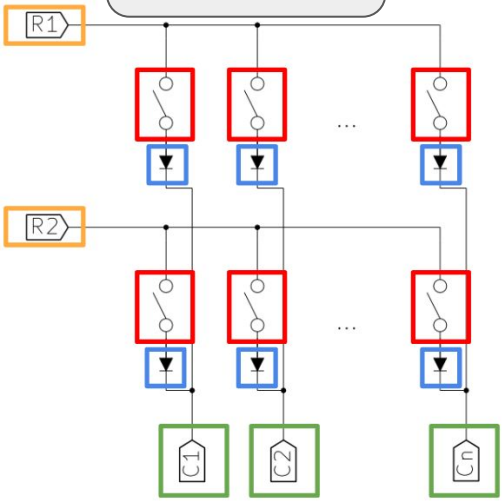
GPIO.Listener

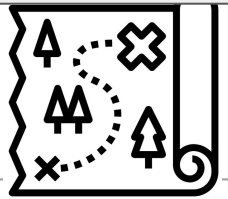
[{R2,C2,:pressed}]

GenServer

USB.Reporter

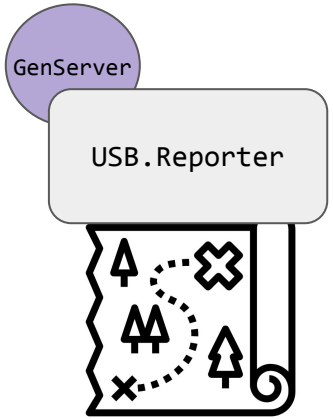
USB Report:
KEY_A





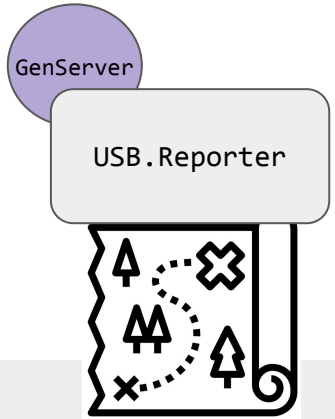
Row	Col	Key
R2	C2	KEY_A
R2	C3	KEY_B
Rm	Cn	KEY_x

A keyboard on top of the BEAM



- DSL to define the keymap
- Multiple layers
  - Toggle or lock
- Macros
  - Pre-defined macros
  - Record macros on-the-fly

A keyboard on top of the BEAM

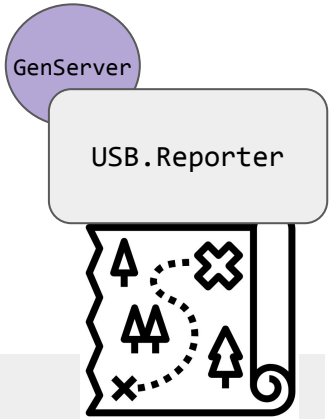


- DSL to define the keymap
- Multiple layers

```elixir
defmodule ElixirKeeb.CanonTypewriter.Layout do
 use ElixirKeeb.Layout, matrix: ElixirKeeb.CanonTypewriter.Matrix

 @layouts [
   [ # layer 0
     [:kc_escape, :kc_1, :kc_2, :kc_3, :kc_4, :kc_5, :kc_6, :kc_7, :kc_8, :kc_9, :kc_0, :kc_equal, :kc_slash, :kc_delete, :kc_bspace],
     [toggle_layer(1), :kc_q, :kc_w, :kc_e, :kc_r, :kc_t, :kc_y, :kc_u, :kc_i, :kc_o, :kc_p, :kc_lbracket, :kc_rbracket, lock_layer(1)],
     [:kc_lctrl, :kc_a, :kc_s, :kc_d, :kc_f, :kc_g, :kc_h, :kc_j, :kc_k, :kc_l, :kc_scolon, :kc_quote, :kc_lgui, :kc_enter],
     [:kc_lshift, :kc_z, :kc_x, :kc_c, :kc_v, :kc_b, :kc_n, :kc_m, :kc_comma, :kc_dot, :kc_grave, :kc_lshift],
     [:kc_lalt, :kc_space, :kc_ralt]
   ],
   [ # layer 1
     [:____, :____, :____, :____, :____, :____, :____, :____, :____, :____, :____, :____, :____, :____, :____],
     [:____, :kc_1, :kc_2, :kc_3, :kc_4, :kc_5, :kc_6, :kc_7, :kc_8, :kc_9, :kc_0, :kc_tab, :kc_bslash, :____],
     [:____, m(0), m(1), m(2), m(3), m(4), record(0), replay(0), :____, :____, :____, :____, :____, :____],
     [:____, :____, :____, :____, :____, :____, :____, :____, :____, :____, :____, :____, :____],
     [:____, :kc_x, :____]
   ]
 ]
end
```

A keyboard on top of the BEAM



- Macros
  - Pre-defined macros
  - Record macros on-the-fly

```elixir
23 defmodule ElixirKeeb.CanonTypewriter.Layout do
24  use ElixirKeeb.Layout, matrix: ElixirKeeb.CanonTypewriter.Matrix
25
26  @macros [
28    [:kc_a, :kc_b, :kc_c], # macro 0
30    "xyz" |> String.graphemes(), # macro 1
32    [
33      {"lshift", :pressed}, "e", {"lshift", :released}, "l", "i", "x", "i", "r", {"lshift", :pressed}, "1", {"lshift", :released},
34    ], # macro 2
36    "Hello, world!", # macro 3
38    &ElixirKeeb.CanonTypewriter.Macros.my_macro/1, # macro 4
39  ]
40
41  @layouts [
42    # layer 0 ...
43    [ # layer 1
44      # ...
46      [:____, m(0), m(1), m(2), m(3), m(4), record(0), replay(0), :____, :____, :____, :____, :____, :____],
47      # ...
49    ]
```

```elixir
17 defmodule ElixirKeeb.CanonTypewriter.Macros do
18  def my_macro(state) do
19    {"Hello from macro function!", state}
20  end
21 end
```

Pros & cons 

- To iterate, I can **redefine the modules on-the-fly** inside an IEx shell on the Pi Zero
- Easily update the keyboard firmware
  - Burn firmware to microSD card only **once**
  - Further firmware changes can be **done via SSH**
- Being able to use a **highly expressive** language like Elixir to do embedded projects is pure bliss
- Listener and Reporter as **supervised** processes

- Raspberry Pi Zero is 10x **more expensive** than the Pro Micro controller
  - ~10EUR vs ~1EUR
- **Slow** boot time
- More **power hungry**
  - I don’t think the ConfigFS device acts on the host suspend/resume instructions
- **Bigger** footprint
  - Trickier to use in smaller keyboards

Future work

- Implement a tap or toggle mechanism
  - Keep Control pressed ➡️ send Control
  - Tap Control ➡️ send Escape
- Proper power management
  - Honor suspend/resume instructions from the host
- Improve boot time
  - Look at streamlining even more the Buildroot system
- Support for other input devices
  - Trackball, mouse

D E M O time 🔥

- Simple keyboard
  - Multiple layers
- Matrix scan latency and Matrix to USB latency
- Macros
  - Pre-defined macros
  - Recording macros
- Tweaking the layout on-the-fly

Thank you!

Questions?

amalbuquerque/elixir-keeb

amalbuquerque/elixir_keeb_ui

Resources

- amalbuquerque/elixir-keeb
- amalbuquerque/elixir_keeb_ui
- elixir-circuits/circuits_gpio: Use GPIOs from Elixir
- nerves-project/usb_gadget: Configure USB Gadget devices
- mindok/contex: Charting and graphing library for Elixir
- Buildroot - Making Embedded Linux Easy
- Nerves Platform
- tmk/tmk_keyboard: Keyboard firmwares for Atmel AVR and Cortex-M
- qmk/qmk_firmware: Open-source keyboard firmware for Atmel AVR and Arm USB families