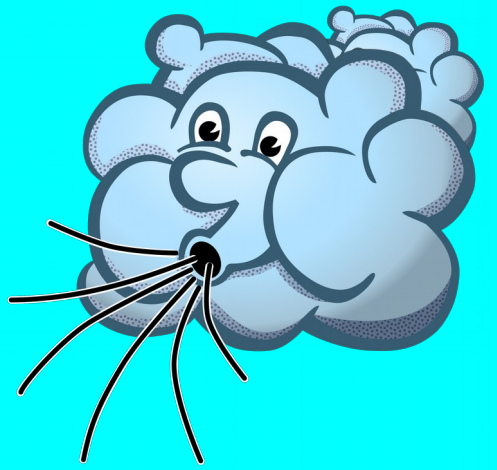


Did You Hear That Wind?

ElixirConf 2018

Frank Kumro

@frigidcode

ElixirConf 2018

A long long time ago*

*7 years

Phoenix
Framework

ElixirConf 2018

Source Code

Lake Effect

Nerves project

https://gitlab.com/fkumro/lake_effect

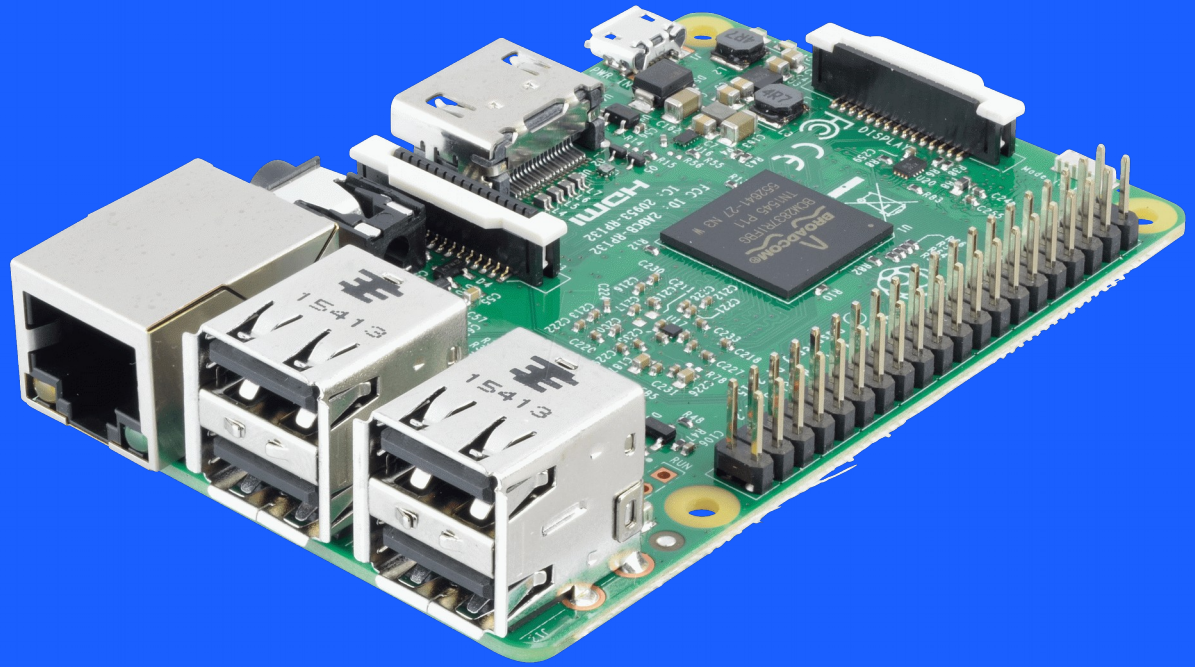
Thunder Snow

Phoenix project

https://gitlab.com/fkumro/thunder_snow

Parts List

Raspberry Pi 3
Model B



ElixirConf 2018

Parts List

Wind Speed
Sensor w/***Analog***
Voltage Output

Parts List

High Temp Waterproof DS18B20 Digital temperature sensor + extras

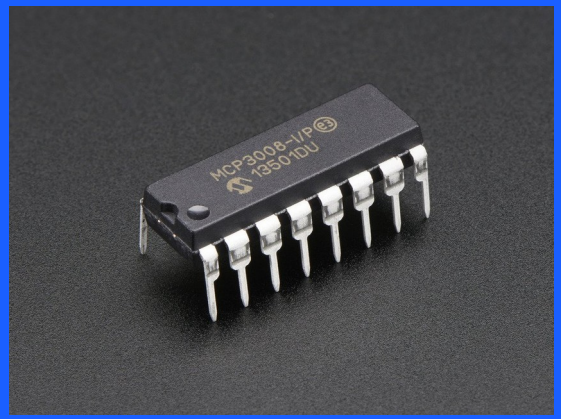
Parts List

Pi Cobbler Plus

```
# Tip:
# Get the “T” cobbler for more room
```

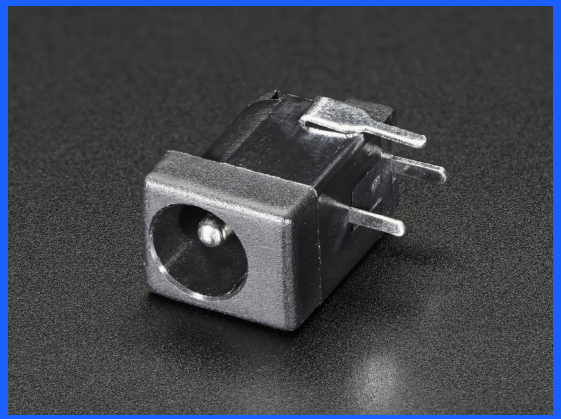
Parts List

MCP3008



Power Supply

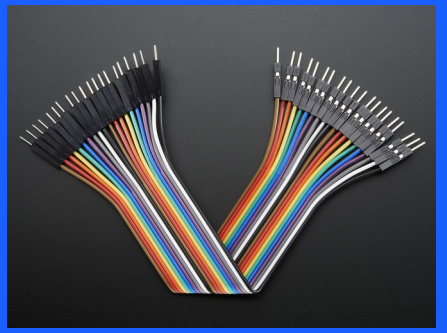
Barrel Jack

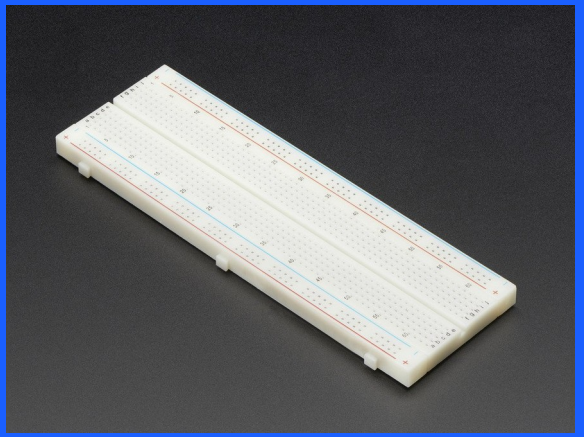


Parts List

Various supplies:

```
  [
    “wires”,
    “solder”,
    “soldering iron”,
    “mounting hardware”,
    “card reader/writer”,
    “breadboard”,
    “patience”
  ]
```





ElixirConf 2018

ElixirConf 2018

Terminology

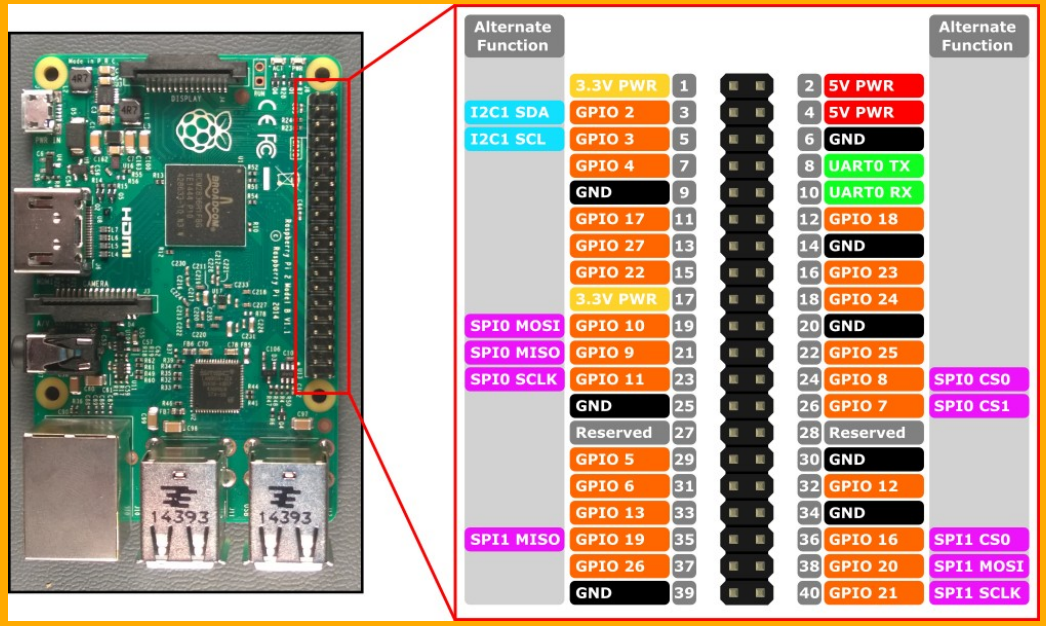
Mix Target - The platform for which your firmware is built. Other commands utilize the target as well (mix deps.get).

```
$ export MIX_TARGET=rpi3

$ MIX_TARGET=rpi3 mix firmware
```

Terminology

GPIO - General purpose input/output.



ElixirConf 2018

Terminology

SPI -Serial Peripheral Interface bus is a common multi-wire bus.

Elixir ALE - Library for interacting with hardware over GPIO/I2C/SPI.

Creating a Nerves Project

Installing Nerves

https://hexdocs.pm/nerves/getting-started.html

Creating a Nerves Project

Generate project

```
$ mix nerves.new hello_nerves
```

Burning firmware

```
$ export MIX_TARGET=rpi3
$ mix compile
$ mix firmware
$ mix firmware.burn
```

Reading Temperature

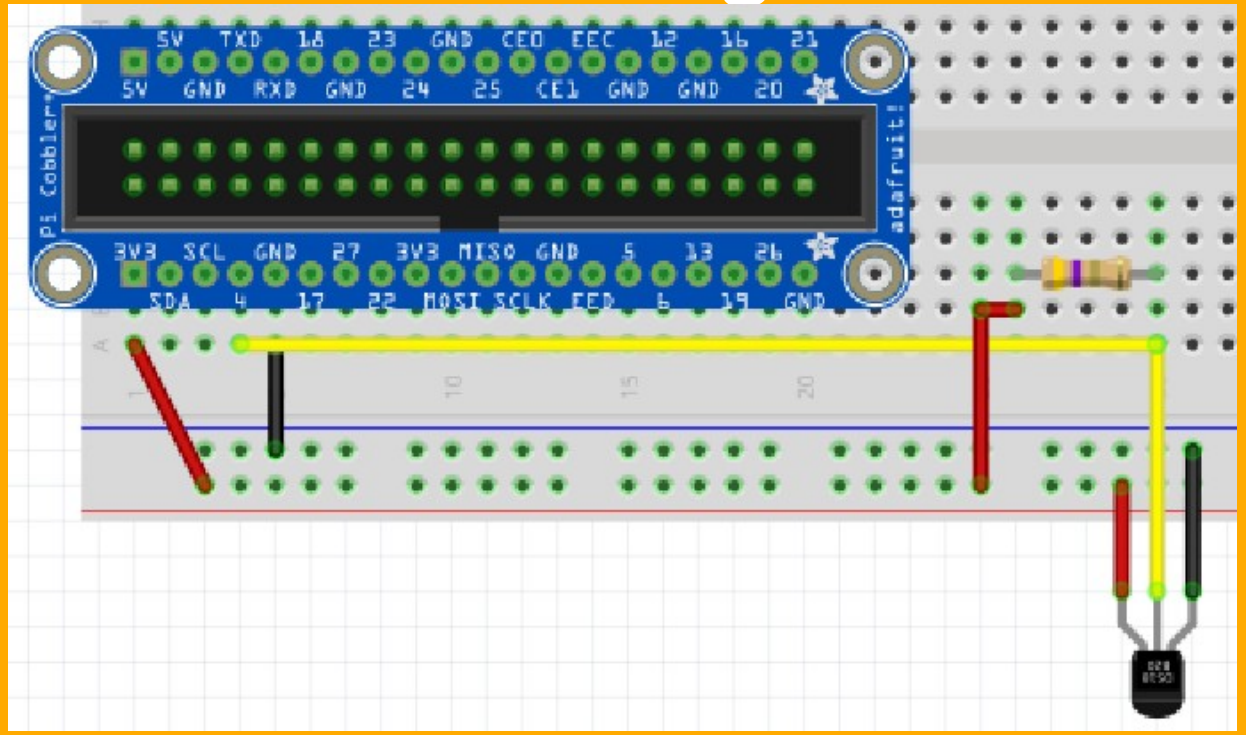
1-Wire Protocol

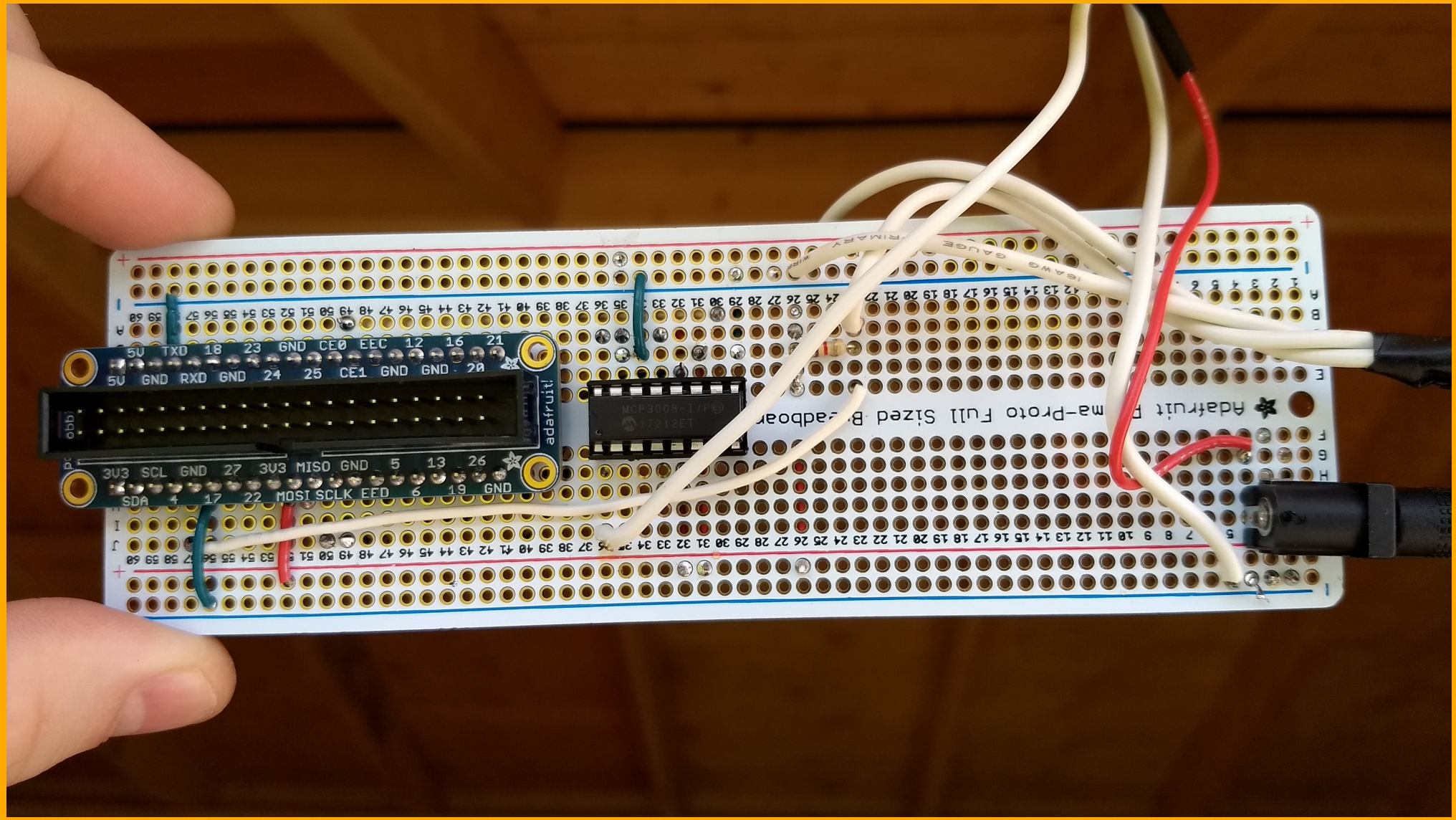
Supported by Nerves w/ configuration

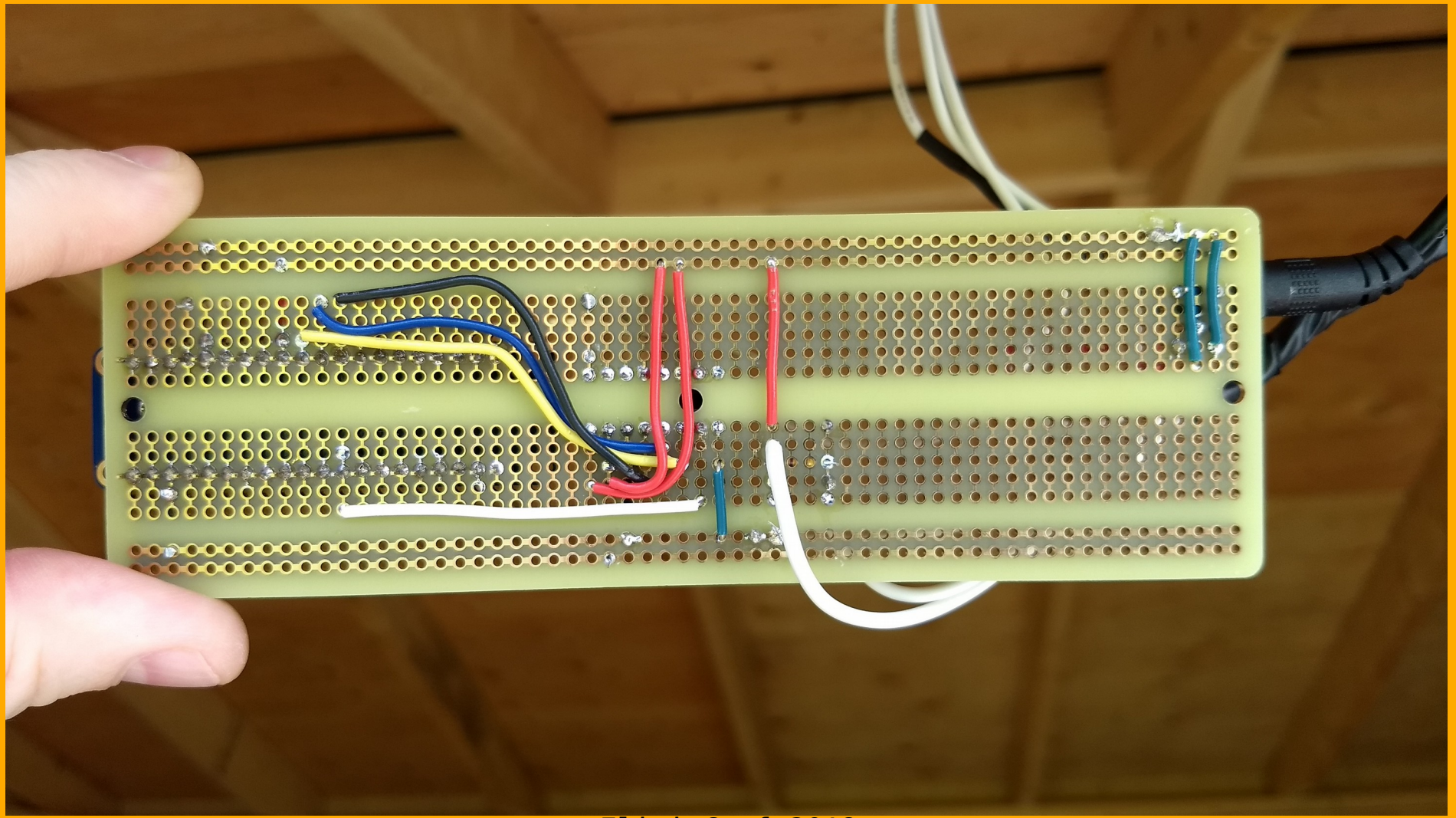
Connects to the GPIO

Reading Temperature

Wiring




5V TXD 18 23 GND CE0 EEC 12 16 21
5V GND RXD GND 24 25 CE1 GND GND 20
adafruit
3V3 SCL GND 27 3V3 MISO GND 5 13 26
SDA 4 17 22 MOSI SCLK EED 6 19 GND
MCP3008-I/P
17212ET



Reading Temperature

Configuration

```
# Create a folder to store the configuration file

mkdir config/rpi3

# Copy config files to our config folder for editing

cp deps/rpi3/nerves_system_rpi3/fwup.conf config/rpi3/

cp deps/rpi3/nerves_system_rpi3/config.txt config/rpi3/
```

Reading Temperature

Configuration

Configure the system to use our fwup.conf file

```
# config/rpi3/config.txt
config :nerves, :firmware,
   fwup_conf: "config/rpi3/fwup.conf"
```

Reading Temperature

Configuration

Specify location of our config.txt file and enable 1-Wire on pin 4

```
# config/rpi3/fwup.conf
file-resource config.txt {
    host-path = "${NERVES_APP}/config/rpi3/config.txt"
}

# uncomment the following lines
dtoverlay=w1-gpio-pullup,gpiopin=4
```

Reading Temperature

Configuration

Set the path to the temperature sensor

```
# config/rpi3.exs
config :lake_effect,
       LakeEffect.Sensors.Temperature.Comm,
       sensor_path:
       "/sys/bus/w1/devices/28-0000081bfd1c/w1_slave"
```

Device id

Reading Temperature

Reading from sensor

Data from sensor

```
xx xx xx xx xx xx xx xx xx : crc=2c YES
xx xx xx xx xx xx xx xx xx t=22187
```

Reading Temperature

Reading from sensor

Set the path to the temperature sensor

```
# Step 1 - Read file contents
file_contents =
  Application.get_env(:lake_effect, __MODULE__)[:sensor_path]
  |> File.read!()

# Step 2 - Parse contents (single row)
result = Regex.run(~r/t=(\d+)/, file_contents)
  |> List.last()
  |> Float.parse()

result / 1000
```

Reading Temperature

Reading from sensor

ElixirConf 2018

Reading Temperature

OTP

```
defmodule LakeEffect.Sensors.Temperature.Server do
  alias LakeEffect.Sensors.Temperature.Comm

  def read() do
    GenServer.call(:temperature_sensor, :read)
  end

  # read temperature - could have been a task
  @impl true
  def handle_call(:read, _from, _state) do
    {:reply, Comm.read(), %{}}
  end
end
```

Reading Temperature

OTP

```
defmodule LakeEffect.Sensors.Temperature.Comm do
  @spec read() :: float
  def read() do
    as_binary()
    |> parse_temp()
  end

  def as_binary() do
    Application.get_env(:lake_effect, __MODULE__)[:sensor_path]
    |> File.read!()
  end
end
```

Reading Temperature

OTP

```
defmodule LakeEffect.Sensors.Temperature.Comm do
  ...
  def parse_temp(data) do
    {temp, _} = Regex.run(~r/t=(\d+)/, data)
      |> List.last()
      |> Float.parse()

    temp / 1000
  end
end
```

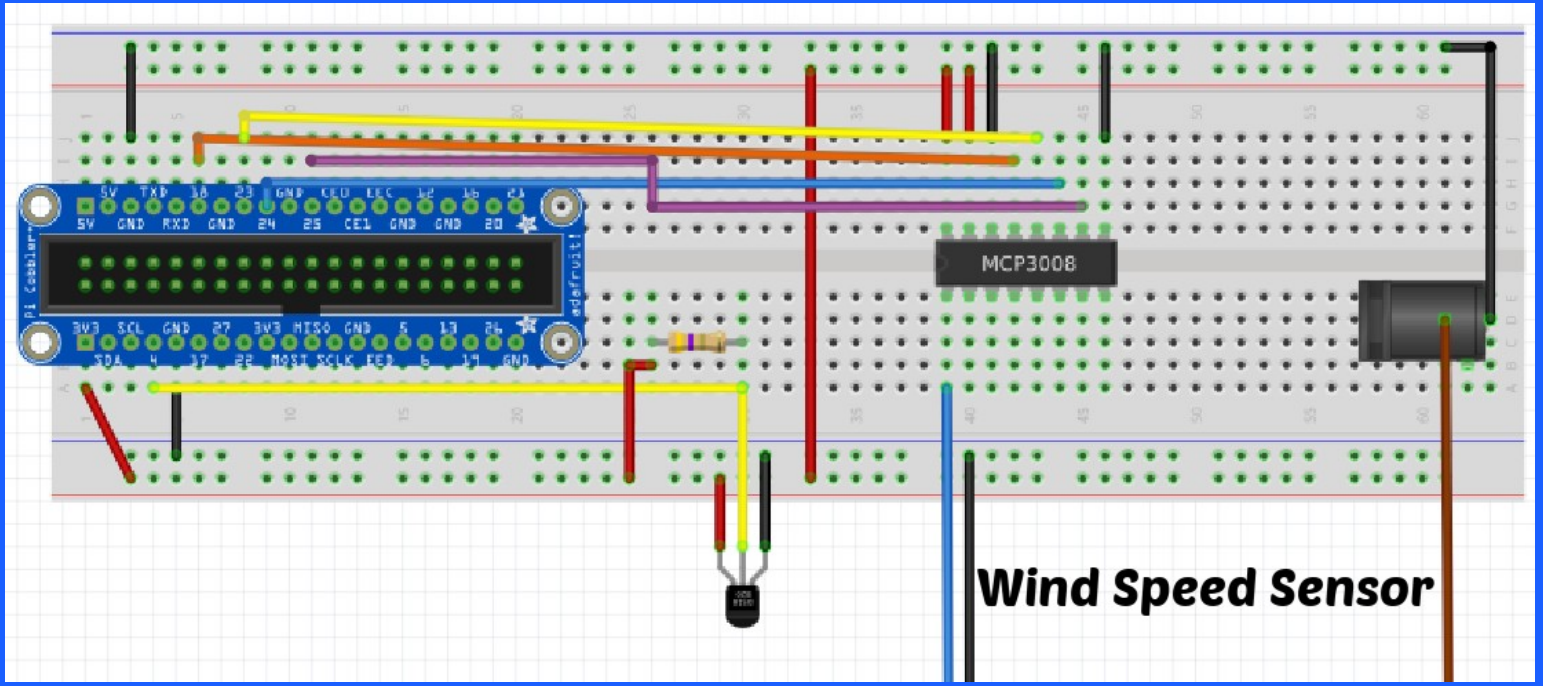
Reading Temperature

OTP

```
LakeEffect.Sensors.Temperature.Server.read()
```

Reading Wind Speed

Wiring



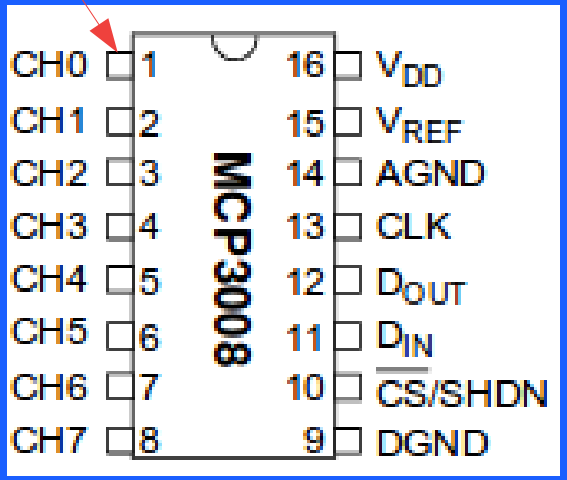
ElixirConf 2018

Reading Wind Speed

Wiring

Reading Wind Speed

Reading from MCP3008



VDD -> 3.3V

VREF-> 3.3V

AGND-> GND

CLK -> #18

DOUT-> #23

DIN -> #24

CS -> #25

DGND-> GND

Reading Wind Speed

Reading from MCP3008

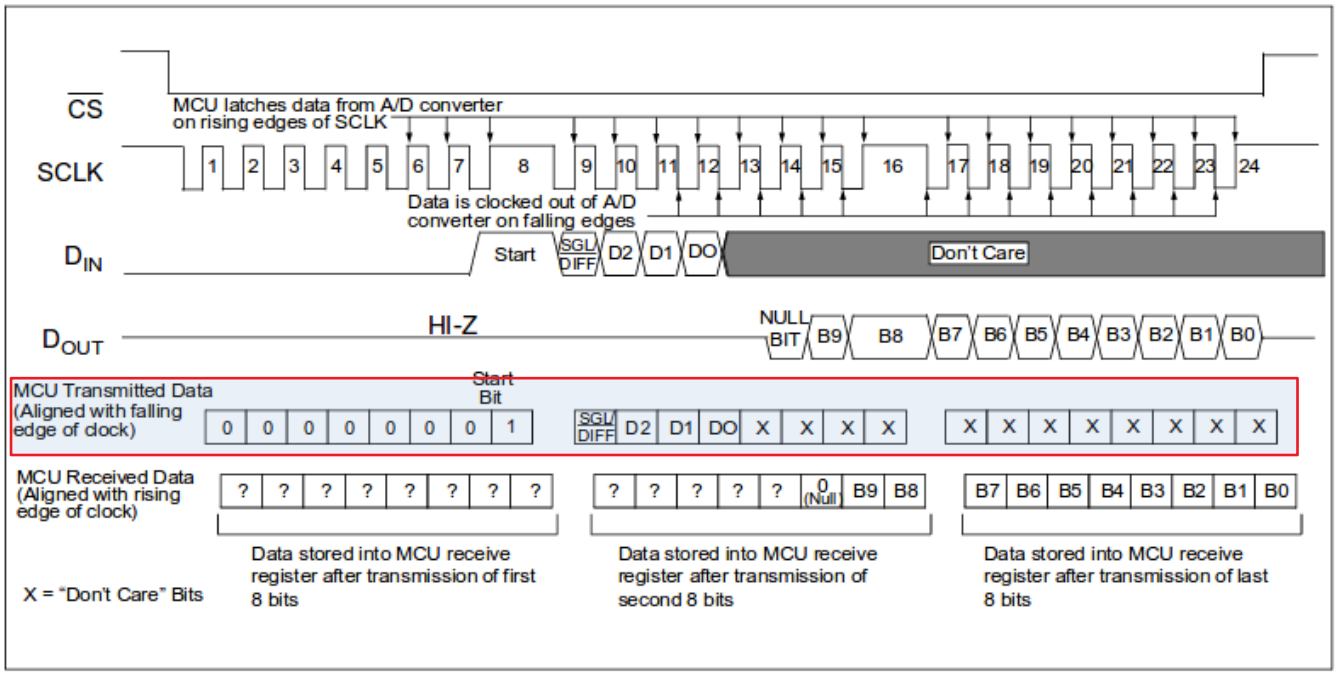


FIGURE 6-2: *SPI Communication with the MCP3004/3008 using 8-bit segments (Mode 1,1: SCLK idles high).*

Reading Wind Speed

Reading from MCP3008

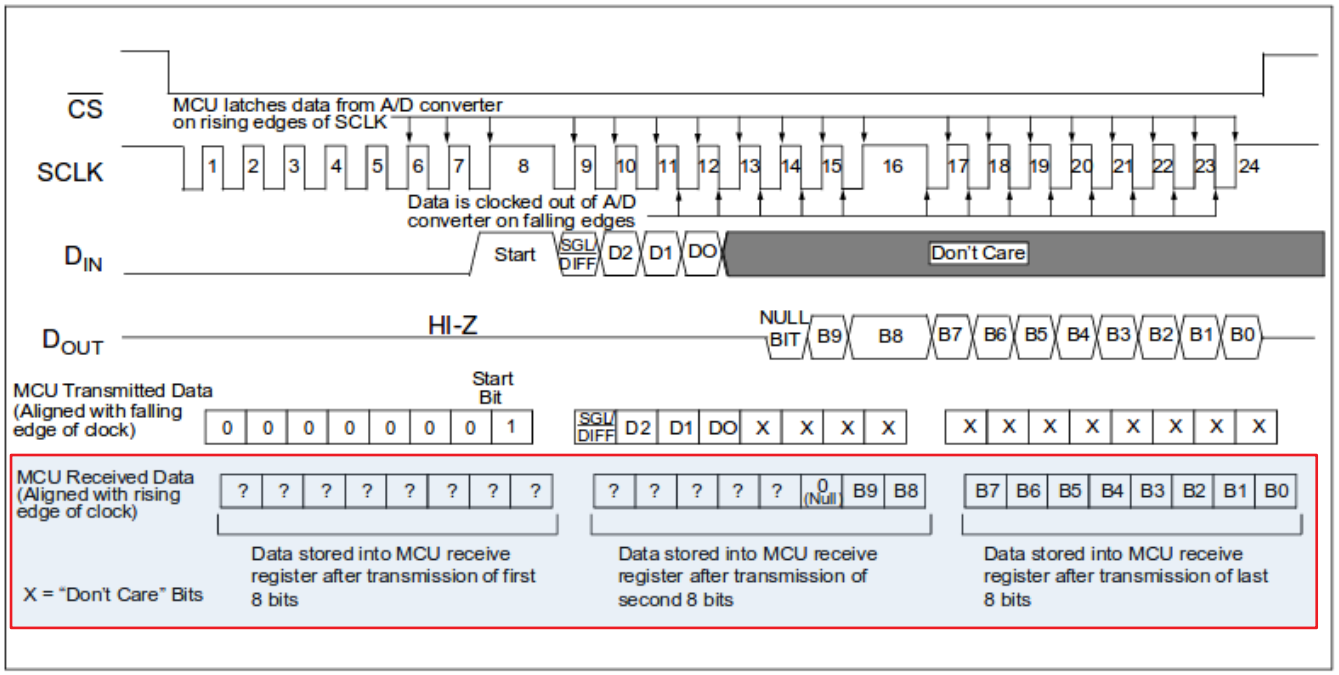


FIGURE 6-2: *SPI Communication with the MCP3004/3008 using 8-bit segments (Mode 1,1: SCLK idles high).*

Reading Wind Speed

Reading from MCP3008

TABLE 5-2: CONFIGURE BITS FOR THE MCP3008

Control Bit Selections				Input Configuration	Channel Selection
Single /Diff	D2	D1	D0		
1	0	0	0	single-ended	CH0
1	0	0	1	single-ended	CH1
1	0	1	0	single-ended	CH2
1	0	1	1	single-ended	CH3
1	1	0	0	single-ended	CH4
1	1	0	1	single-ended	CH5
1	1	1	0	single-ended	CH6
1	1	1	1	single-ended	CH7
0	0	0	0	differential	CH0 = IN+ CH1 = IN-
0	0	0	1	differential	CH0 = IN- CH1 = IN+
0	0	1	0	differential	CH2 = IN+ CH3 = IN-
0	0	1	1	differential	CH2 = IN- CH3 = IN+
0	1	0	0	differential	CH4 = IN+ CH5 = IN-
0	1	0	1	differential	CH4 = IN- CH5 = IN+
0	1	1	0	differential	CH6 = IN+ CH7 = IN-
0	1	1	1	differential	CH6 = IN- CH7 = IN+

```
Single Diff = 1

[D2,D1,D0]  = 0

# payload to send - 24 bits
00000001 10000000 00000000

# hex
0x01 0x80 0x00
```

Reading Wind Speed

Reading from MCP3008

```
# add ElixirALE to mix.exs
{:elixir_ale, "~> 1.1"}

$ MIX_TARGET=rpi3 mix deps.get
```

Reading Wind Speed

Reading from MCP3008

```
defmodule LakeEffect.Sensors.WindSpeed.Server do
  alias ElixirALE.SPI

  def read() do
    GenServer.call(:wind_speed_sensor, :read)
  End

  # start connection with the wind speed sensor via SPI
  def init(_) do
    {:ok, pid} = SPI.start_link(“spidev0.0”)
    {:ok, %{ale: pid}}
  end
end
```

Reading Wind Speed

Reading from MCP3008

```
defmodule LakeEffect.Sensors.WindSpeed.Server do
  alias LakeEffect.Sensors.WindSpeed.Comm
  ...

  def handle_call(:read, _from, state) do
    {:reply, Comm.read(state[:ale]), state}
  end
end
```

Reading Wind Speed

Reading from MCP3008

```
defmodule LakeEffect.Sensors.WindSpeed.Comm do
  def read(pid) do
    pid
    |> spi_transfer(transmission_payload())
    |> spi_read
    |> Convert.from_counts_to_volts()
    |> Convert.from_voltage_to_ms()
    |> Convert.from_ms_to_mph()
  end
end
```

Reading Wind Speed

Reading from MCP3008

```
defmodule LakeEffect.Sensors.WindSpeed.Comm do
  ...
  defp transmission_payload(), do: <<0x01, 0x80, 0x00>>

  defp spi_transfer(pid, payload) do
    pid |> SPI.transfer(payload)
  end

  defp spi_read(<<_::size(14), counts::size(10)>>) do
    counts
  end
end
```

Reading Wind Speed

Reading from MCP3008

```
# convert counts into voltage
voltage = counts / 1023 * 3.3

# specific for the Anemometer Wind Speed Sensor w/Analog
# Voltage Output
wind_speed_in_ms = (voltage - 0.4) / 1.6 * 32.4

# close enough for government work
wind_speed_in_mph = wind_speed_in_ms * 2.2369
```

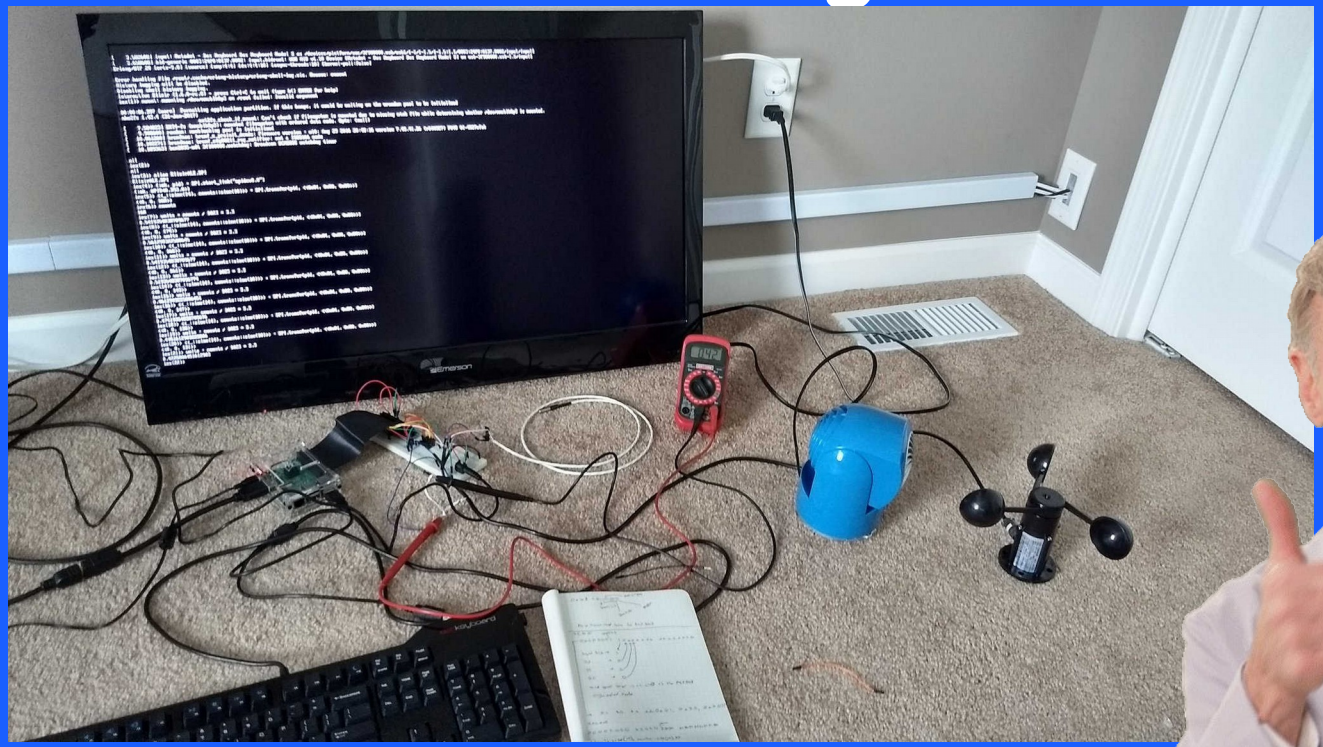
Reading Wind Speed

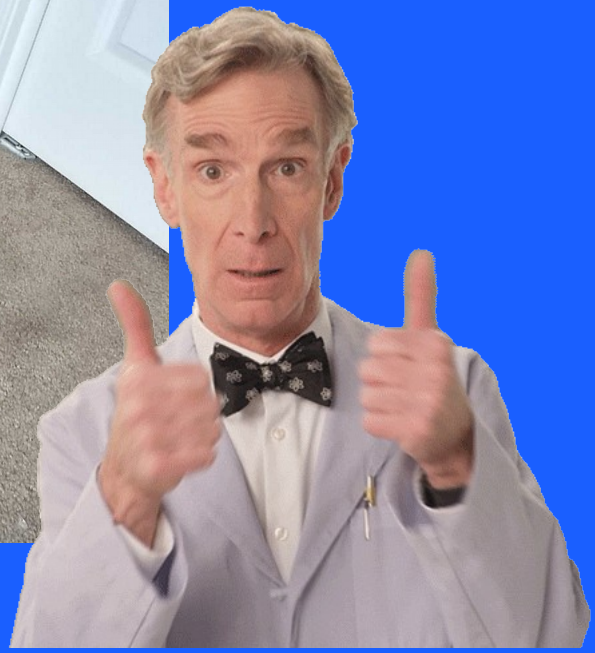
Reading from MCP3008

```
LakeEffect.Sensors.WindSpeed.Server.read()
```

Reading Wind Speed

Testing





ElixirConf 2018

Sending data to Thunder snow

```elixir
defmodule LakeEffect.Jobs.Server do
  def start_link() do
    GenServer.start_link(__MODULE__, [], name: :jobs_server)
  end

  def init(_) do
    if enabled?() do
      schedule_initial_job()
      {:ok, nil}
    else
      :ignore
    end
  end
end
```

Sending data to Thunder snow

```elixir
defmodule LakeEffect.Jobs.Server do
  defp schedule_initial_job() do
    # wait 60 seconds to allow networking to connect / ntpd
    Process.send_after(self(), :perform, 60_000)
  end

  defp schedule_next_job() do
    Process.send_after(self(), :perform, 30_000)
  end

  defp enabled?() do
    Application.get_env(:lake_effect, __MODULE__)[:enabled]
  end
end
```

Sending data to Thunder snow

```
defmodule LakeEffect.Jobs.Server do
  ...
  def handle_info(:perform, state) do
    WeatherReport.collect()
    |> WeatherReport.save()

    schedule_next_job()

    {:noreply, state}
  end
end
```

Wireless Setup

Using nerves_network

```
# add nerves_network to mix.exs
{:nerves_network, "~> 0.3"}

# fetch nerves_network hex package
$ MIX_TARGET=rpi3 mix deps.get
```

Wireless Setup

Using nerves_network

```
# config/rpi3.exs
config :nerves_network, regulatory_domain: "US"
key_mgmt = System.get_env("NERVES_NETWORK_KEY_MGMT") || "WPA-PSK"

config :nerves_network, :default,
  wlan0: [
    ssid: System.get_env("NERVES_NETWORK_SSID"),
    psk: System.get_env("NERVES_NETWORK_PSK"),
    key_mgmt: String.to_atom(key_mgmt)
  ],
  eth0: [
    ipv4_address_method: :dhcp
  ]
```

Wireless Setup

Using nerves_network

ENV vars must be present before compiling.

ElixirConf 2018

Remote firmware flashing

Using nerves_firmware_ssh

```
# add nerves_firmware_ssh to mix.exs
{:nerves_firmware_ssh, "~> 0.3", except: :test}

# fetch nerves_firmware_ssh hex package
$ MIX_TARGET=rpi3 mix deps.get
```

Remote firmware flashing

Using nerves_firmware_ssh

```
# config/rpi3.exs
config :nerves_firmware_ssh,
  authorized_keys: [
    File.read!(
      Path.join(
                  System.user_home!(),
                  System.get_env("NERVES_FIRMWARE_SSH_KEY")
                )
    )
  ]
```

Remote firmware flashing

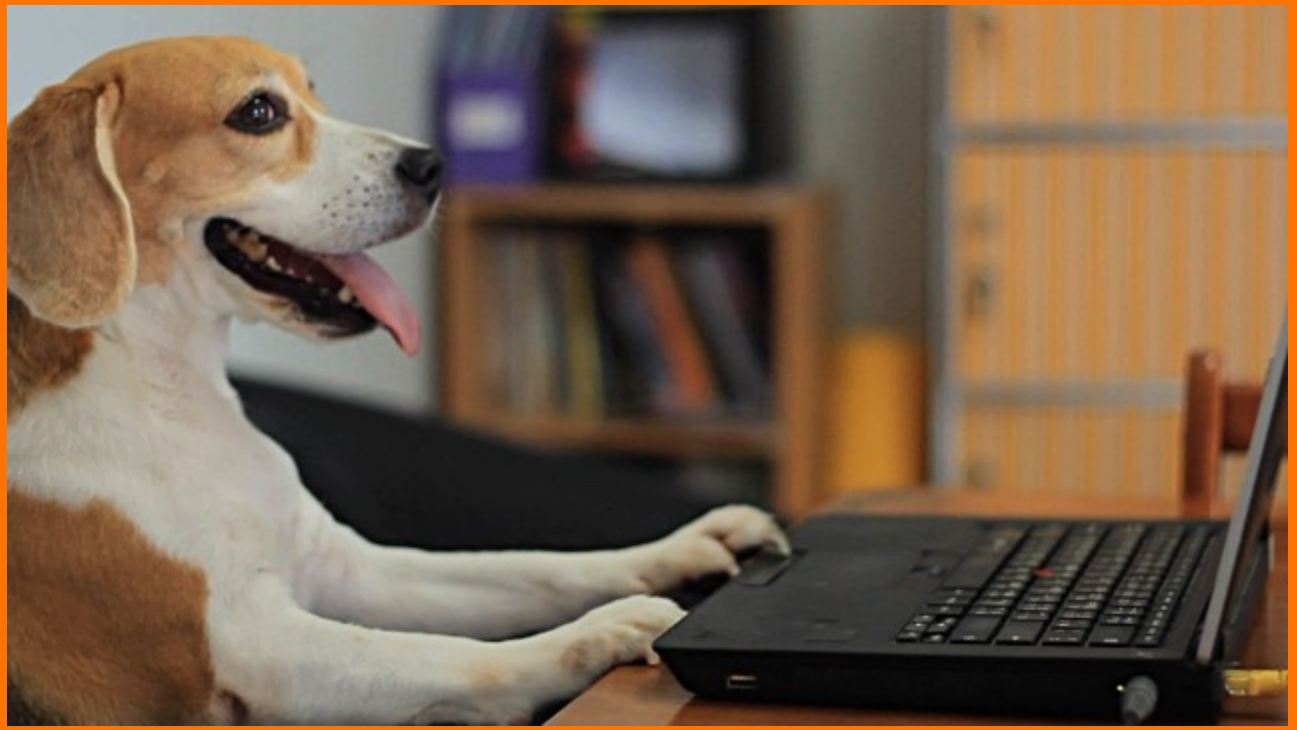
Using nerves_firmware_ssh

Tip - Create a directory to store your ssh key in to avoid issues with keys that have passphrases vs no passphrases.

Remote firmware flashing

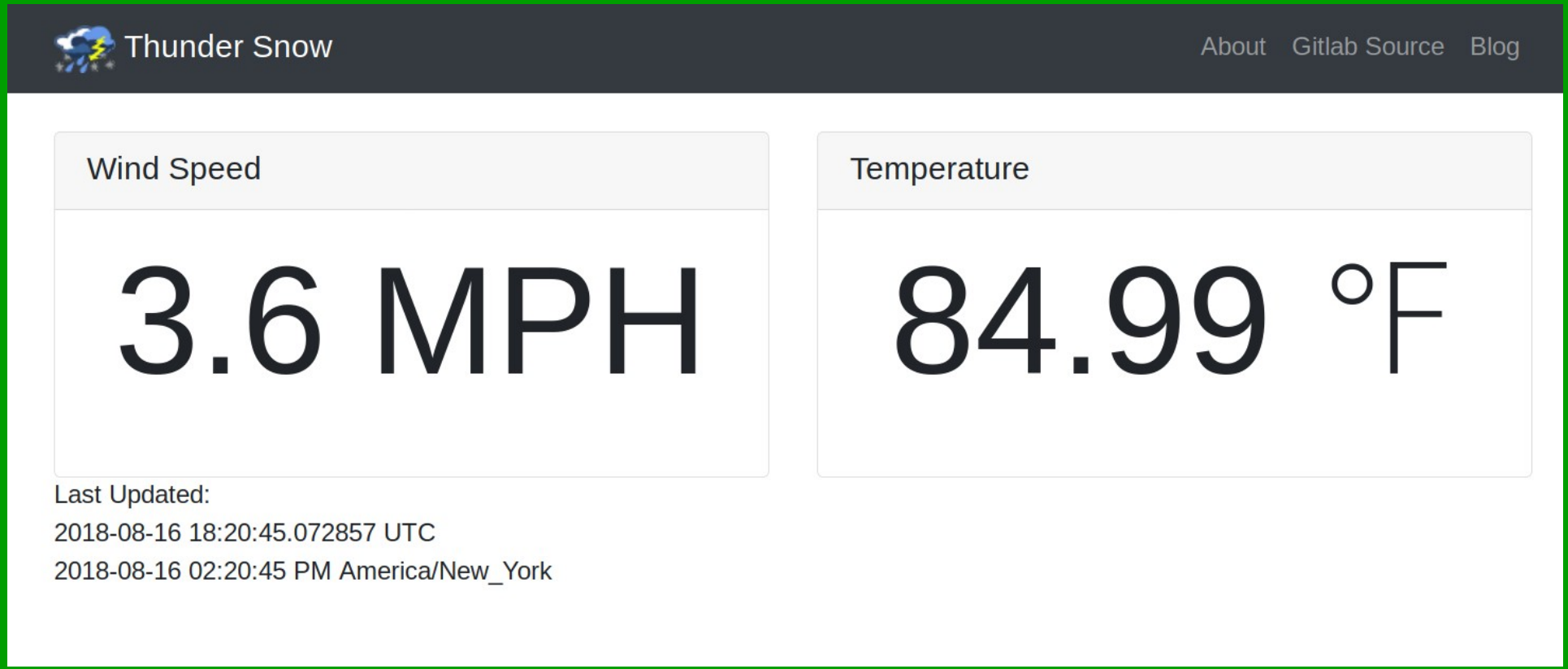
Using nerves_firmware_ssh

```
$ export MIX_TARGET=rpi3
$ mix compile
$ mix firmware
$ mix firmware.push -user-dir=PATH_TO_SSH_KEY_FOLDER
  192.168.xxx.xxx
```



ElixirConf 2018

Thunder Snow



ElixirConf 2018

Thunder Snow

Auth

Bearer Auth – Shared token between weather station and Phoenix application.

```
Authorization: Bearer <shared token>
```

Thunder Snow

Auth

```
# thunder_snow/plugs/auth.ex
def init(default), do: default

def call(conn, _default) do
  case is_weather_station?(get_auth_header(conn)) do
    true -> conn
    _ -> conn |> redirect_to_403()
  end
end
```

Thunder Snow

Auth

```
defp is_weather_station?([head | _]) do
    String.equivalent?(
      head,
      "Bearer #{Application.get_env(:thunder_snow, __MODULE__)
[:api_key]}"
    )
end

defp is_weather_station?(_), do: false

defp get_auth_header(conn) do
  Plug.Conn.get_req_header(conn, "authorization")
end
```

Thunder Snow

Auth

```
defp redirect_to_403(conn) do
    conn
    |> put_resp_content_type("application/json")
    |> send_resp(403, build_403_response())
    |> halt()
end

defp build_403_response do
  """
  {"status_code": 403, "message": "forbidden"}
  """
end
```

Thunder Snow

Auth

```
# thundersnow_web/router.ex

pipeline :api do
  plug(:accepts, ["json"])
  plug(ThunderSnow.Plugs.Auth)
end
```

Thunder Snow

API Routing

```
# thundersnow_web/router.ex

scope "/api", ThunderSnowWeb do
  pipe_through(:api)
  resources("/reports", ReportController)
end
```

Thunder Snow

API to receive data from Lake Effect

```
POST /api/reports

{
  "report":
  {
    "temperature":111.0,"wind_speed":10.2
  }
}
```

Thunder Snow

Database

```
# priv/migrations/*_create_reports.exs
create table(:reports) do
  add(:wind_speed, :float)
  add(:temperature, :float)

  timestamps()
end

create(index(:reports, ["inserted_at DESC NULLS LAST"]))
```

Thunder Snow

Controller

```
# thundersnow_web/controllers/report_controller.ex
def create(conn, %{"report" => report_params}) do
    with {:ok, %Report{} = report} <-
           Weather.create_report(report_params) do
      conn
      |> put_status(:created)
      |> put_resp_header("location", report_path(conn, :show,
report))
      |> render("show.json", report: report)
    end
  end
```

Thunder Snow

Controller

```
# thundersnow_web/controllers/page_controller.ex
def index(conn, _params) do
  weather_report =
    case ThunderSnow.Weather.get_latest_report() do
      report when is_map(report) -> report
      nil -> %{wind_speed: "NA", temperature: "NA", inserted_at: "NA"}
    end

  render(conn, "index.html", weather_report: weather_report)
end
```

Thunder Snow

Controller

```
# thundersnow/weather.ex
def get_latest_report do
  Report
  |> order_by(desc: :inserted_at)
  |> limit(1)
  |> Repo.one()
end
```

What's Next

ElixirConf 2018

What's Next

Graphing of historical data (12/24 hours, monthly)

Refactoring to utilize hex packages for sensors

Support more sensors (humidity, barometric pressure, rainfall)

Better UI

Real Feel-ish

Your suggestions welcome!

Follow Along

ElixirConf 2018