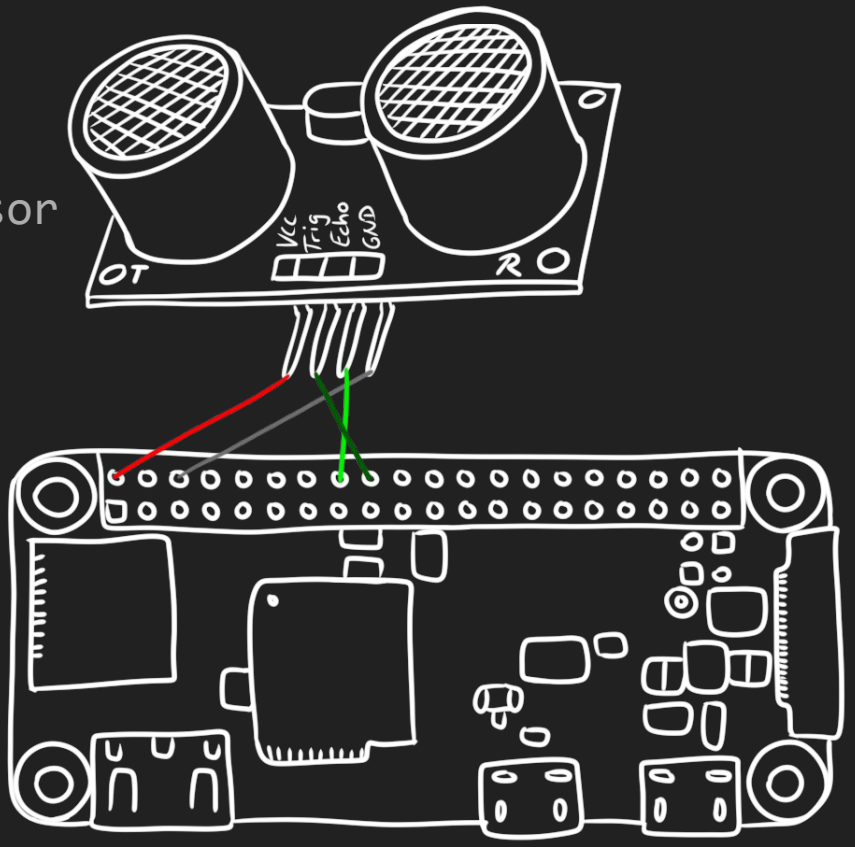
Time dependent sensor readings

using NIFs

Setup

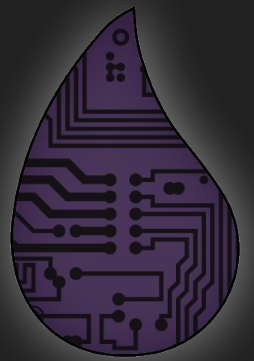
HC-SR04 ultrasonic sensor

Raspberry Pi Zero W



Nerves libraries

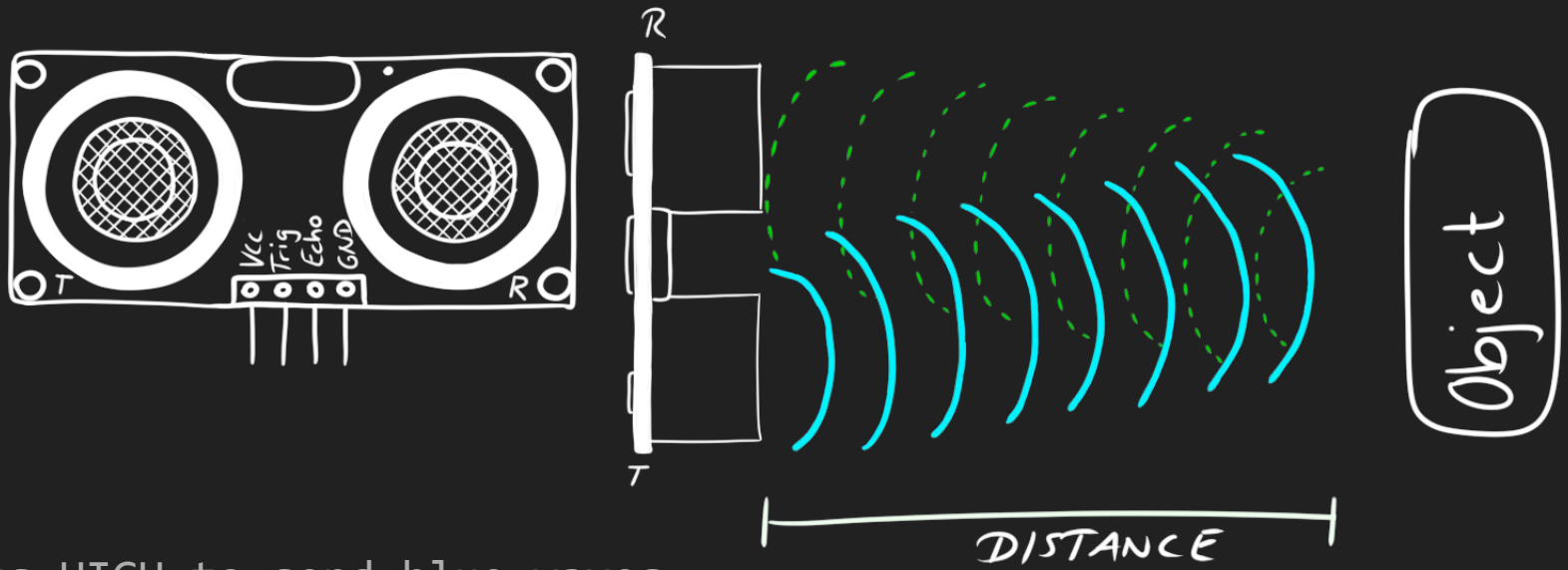
- Circuits GPIO (elixir-circuits/circuits_gpio)
- elixir_ale (fhunleth/elixir_ale)



Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL level

(Circuits.GPIO/elixir_ale) works great with LEDs, buttons, many kinds of sensors, and simple control of motors. In general, if a device requires high speed transactions or has hard real-time constraints in its interactions, this is not the right library. For those devices, see if there's a Linux kernel driver.

HC-SR04 time constraints



10 µs HIGH to send blue waves

Listen for green waves

C code for distance

```
gpio_write(PIN_TRIGGER, 1);
usleep(10);
gpio_write(PIN_TRIGGER, 0);

while(gpio_read(PIN_ECHO) == 0);

startTime = getMicrotime();
while(gpio_read(PIN_ECHO) == 1);
stopTime = getMicrotime();

difference = stopTime - startTime;
rangeCm = difference / 58;
```

Setting up NIFs

- `{:elixir_make, "~> 0.4", runtime: false}`
- Makefile with Erlang linker flags + target toolchain
- `#include <erl_nif.h>`
- `ERL_NIF_INIT(Elixir.Module, nif_funcs, NULL,NULL,NULL,NULL)`
- `:erlang.load_nif/2`

https://embedded-elixir.com/post/2017-12-23-using-nifs-with-nerves/

NIF-ifying distance code

```c
#include <erl_nif.h>
static ERL_NIF_TERM do_sensor_reading(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
  // ..
  ERL_NIF_TERM atom_ok = enif_make_atom(env, "ok");
  ERL_NIF_TERM double_reading = enif_make_double (env, rangeCm);
  return enif_make_tuple(env, 2, atom_ok, double_reading);
}

static ERL_NIF_TERM init_sensor(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
  // ..
  return enif_make_atom(env, "ok");
}

static ErlNifFunc nif_funcs[] =
{
    {"do_sensor_reading", 0, do_sensor_reading, 0},
    {"init_sensor", 0, init_sensor, 0}
};

ERL_NIF_INIT(Elixir.Firmware.Sensor, nif_funcs, NULL,NULL,NULL,NULL)
```

Module using the NIF

```elixir
defmodule Firmware.Sensor do
  require Logger

  def do_sensor_reading, do: :erlang.nif_error(:nif_not_loaded)
  def init_sensor, do: :erlang.nif_error(:nif_not_loaded)

  @on_load :load_nif
  def load_nif do
    nif_file = '#{:code.priv_dir(:firmware)}/sensor'
    case :erlang.load_nif(nif_file, 0) do
      :ok -> :ok
      {:error, {:reload, _}} -> :ok
      {:error, reason} -> Logger.warn("Failed to load NIF: #{inspect(reason)}")
    end
  end
end
```

Making use of the NIF

```elixir
defmodule Firmware.SensorServer do
  use GenServer
  alias Firmware.Sensor

  def init(opts) do
    Sensor.init_sensor()
    schedule_reading()
    {:ok, -1.0}
  end

  def handle_info(:pub_reading, state) do
    {:ok, reading} = Sensor.do_sensor_reading()
    Phoenix.PubSub.broadcast(Nerves.PubSub, "distance", {:reading, reading})
    schedule_reading()
    {:noreply, state}
  end

  defp schedule_reading(), do: Process.send_after(self(), :pub_reading, 100)
end
```

“we suggest to use over 60ms measurement cycle”

↑ PubSub exposed with Phoenix

LiveView!

```elixir
defmodule UiWeb.DistanceLive do
  use Phoenix.LiveView

  @topic "distance"

  def render(assigns), do: UiWeb.PageView.render("distance.html", assigns)

  def mount(_, socket) do
    distance = 0.0
    UiWeb.Endpoint.subscribe(@topic)
    {:ok, assign(socket, :distance, distance)}
  end

  def handle_info({:reading, reading}, socket) when reading < 200 do
    {:noreply, assign(socket, :distance, reading)}
  end

  def handle_info({:reading, _}, socket), do:  {:noreply, socket}

end
```

← All we have to do!

now readings pour in

Demo!

Conclusions

- NIFs weren’t that scary
- erl_nif.h is pretty neat
- Nerves has great example projects and ecosystem

https://github.com/thehabbos007/nerves-hc-sr04-distance/