
AtomVM

About me (Davide Bettio)

https://github.com/bettio/ | davide@uninstall.it | https://uninstall.it/

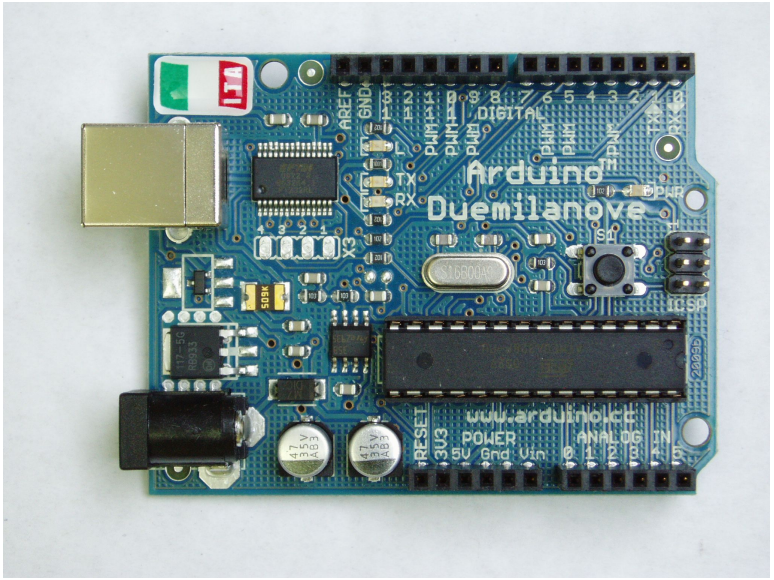
- Tinker with hardware and embedded systems since 2004.
- Long-time open-source dev (since ~2005 contributed to KDE Plasma and others).
- Fell in love with Elixir in 2017, while creating Astarte Platform.
- Started the AtomVM project in 2017
- I love hiking!

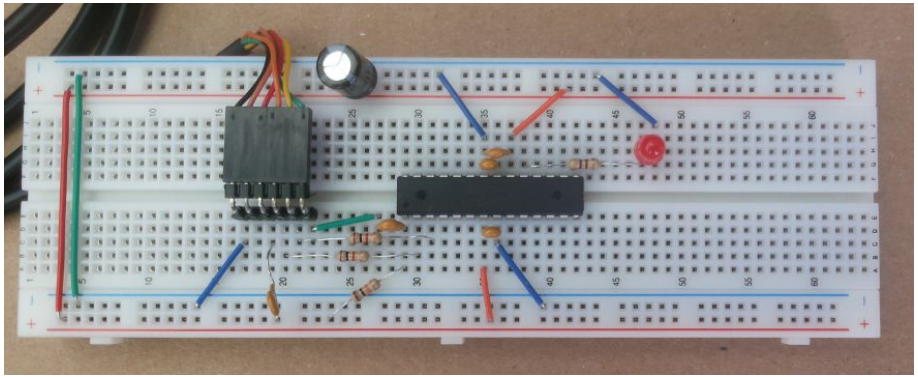
AtomVM

AtomVM

Once Upon a Time, the Arduino

- The pioneer of physical computing devices
- Simple to assemble and develop
- Cheap (arduino board ~20 €, IC: < 2 €)
- Very limited, yet so powerful

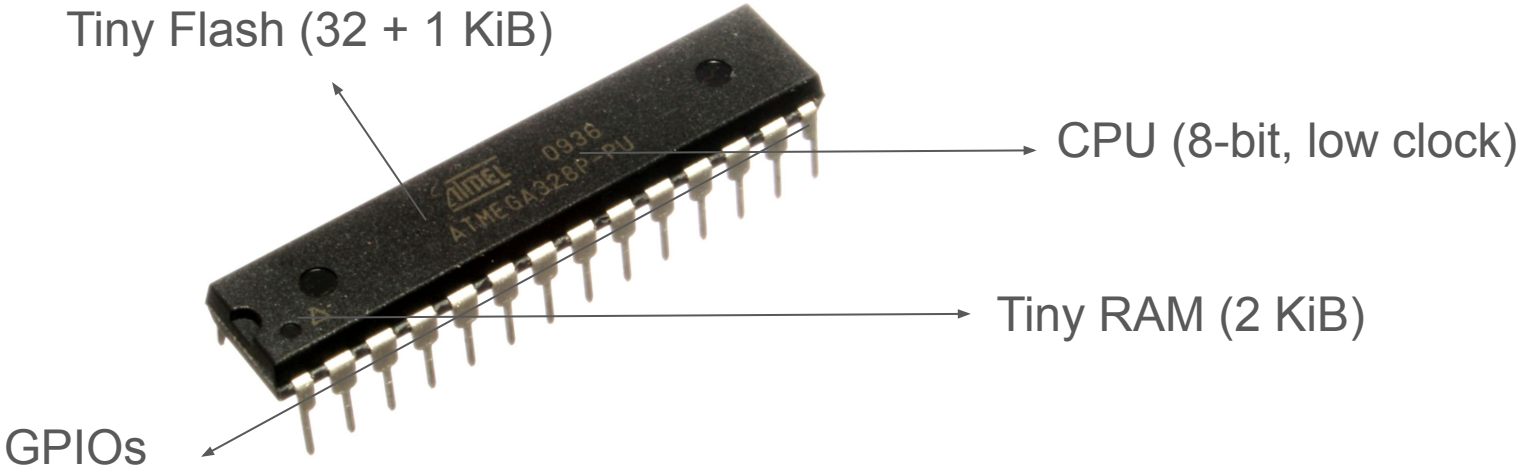




AtomVM

Classic MCU Anatomy (e.g., ATMega328P)

A small computer on a chip:

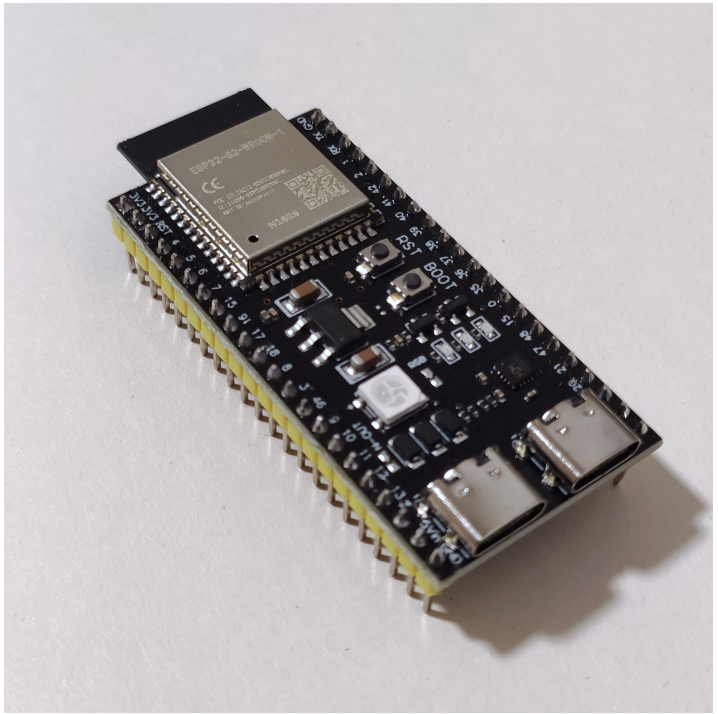


AtomVM

Modern MCU: ESP32 Example

ESP32:

- Cost < 5 €
- Dual Core @ 240MHz
- RAM: ~500KB - 8MB
- Flash: 4MB - 16MB
- Connectivity: WiFi, Bluetooth, etc.
- Lots of GPIOs & integrated peripherals
- Low Power / Battery-friendly



AtomVM

Powerful, But Still...Different

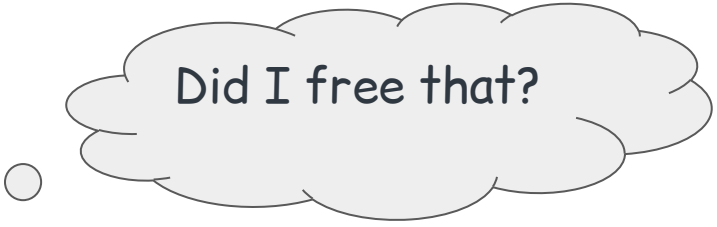
- Massive leap from classic MCUs
- Still resource-constrained vs. PCs/Servers
  - KB/MB RAM, not GB
  - No OS (or RTOS)
  - Development: Mostly C / C++

https://www.reddit.com/r/PallasCats/comments/1d8j3jd/bol/

AtomVM

The C/C++ Experience on MCUs

- Concurrency? Manual, tricky.
- Binary parsing? Boring & dangerous.
- Async? Callback hell, anyone?
- Memory?



AtomVM

Elixir would make our life easier, *but*

the standard BEAM VM wasn't designed for environments with only ~500 KiB of available RAM.

What if we could bring *somehow* the safety, concurrency, and productivity of the BEAM ecosystem to these tiny devices?

AtomVM

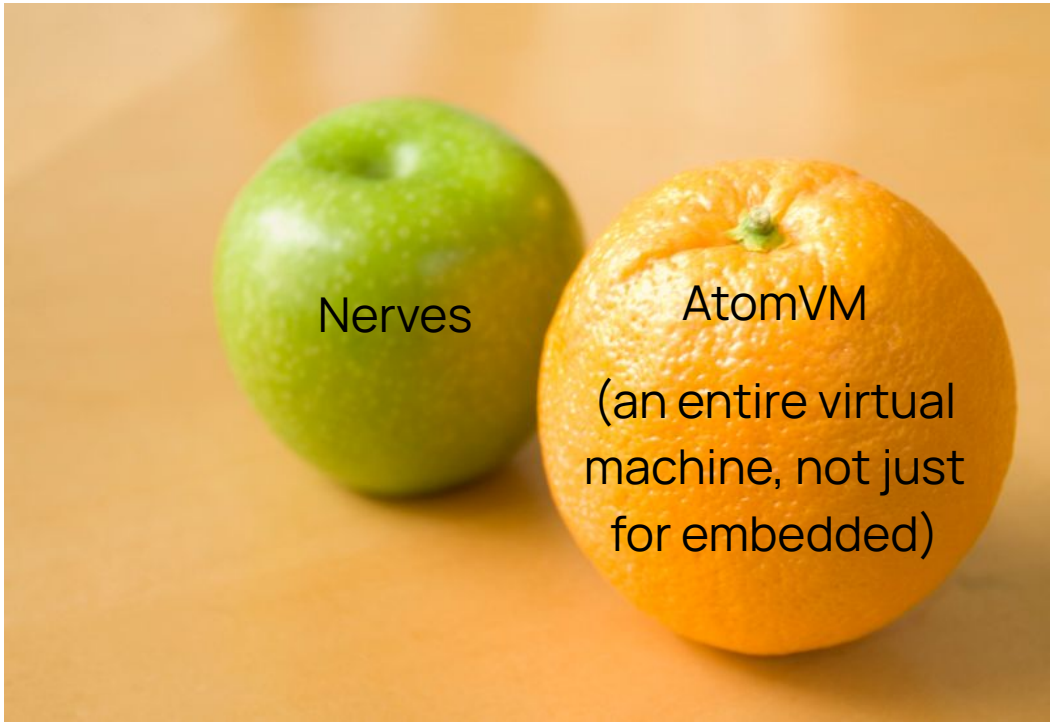
To the Rescue

AtomVM, A lightweight virtual machine designed to run compiled Erlang, Elixir and Gleam code on microcontrollers with limited resources.

- Key Trade-offs:
  - Memory First: RAM & Flash are precious
  - Portability: New targets in hours, not days
  - Flexible Requirements: Adaptable core

AtomVM

Comparison with Nerves



https://en.wikipedia.org/wiki/Apples_and_oranges#/media/File:Apple_and_Orange_-_they_do_not_compare.jpg

Let's compare apples and oranges anyway.

AtomVM on MCUs, TL;DR:

- Hello world footprint: 512 KiB of flash, 32 KiB of RAM
- Perfect for cheap low power devices
- …

AtomVM

Let’s do our simple Hello World Project

What you Need / Suggested Hardware

Option 1: **Espressif**

- **ESP32 / ESP32-S2, ESP32-S3 DevKit C** → **Wifi, Bluetooth**, up to **8 MiB** of PSRAM
- **ESP32-C2/C3** → Wifi, Bluetooth, up to ~512 KiB or RAM, RISC-V CPU
- **ESP32-C6** → Wifi, Bluetooth, **Thread, ZigBee**, RISC-V CPU
- And many more... (ESP32-H2, -C5, -P4) with different features

Disclaimer: Do not buy ESP8266 and other ancient devices pre-ESP32

Idea: search for fancy devboards, or cheap ones, like the “cheap yellow display”

What you Need / Suggested Hardware

Option 2: **Raspberry Pi Pico**

- Pico 1/1W (RP2040) → **264 KiB of RAM,** dual core @ 133 MHz+, 2 MiB+ flash
- Pico 2/W (RP2350) → **512 KiB of RAM,** dual core @ 150 Mhz, 4 MiB+ flash
- **The 'W' models include Wi-Fi + Bluetooth**
- What's cool? Peripherals & **Programmable I/O (PIO)**

Note: Raspberry Pi Pico ≠ "classic" Raspberry Pi or Raspberry Pi Zero

AtomVM

What you Need / Setup

- Minimal hardware setup: just a USB cable (that's it)
- A serial terminal app (like `minicom` on Linux/macOS or PuTTY on Windows)
  - This is how you'll see all the debug, error, and info messages from your device
- Your favorite Elixir setup

AtomVM

Big Disclaimer

Source: https://manulization.com/manuls/magellan.html

- Heads up: AtomVM is still pre-v1.0, which means APIs are not yet stable
- We will break APIs, but the core concepts will remain the same
- The code here might not work forever, but we keep the official documentation and examples up-to-date

See also: https://doc.atomvm.org/latest/UPDATING.html

AtomVM

Next Step: `exatomvm`

- Add `{:exatomvm, github: "AtomVM/exatomvm", runtime: false}` to `mix.exs`
- It provides you a number of mix tasks to build & flash your project
- Optional: add `{:pythonx, "~> 0.4.0", runtime: false}`, so you can just flash your app without any additional toolchain or SDK
- On ESP32: **`mix atomvm.esp32.install`** (downloads a pre-built binary)

AtomVM

Configuring mix.exs

Just add an `atomvm` section to mix.exs project function:

```
def project do
    [...]
    atomvm: [
      start: Blink, # the module with our start/0 entry point function
      flash_offset: 0x210000
    ]
end
```

AtomVM

The Physical Computing Hello World

```
defmodule Blink do
  @pin 2
  def start() do
      GPIO.set_pin_mode(@pin, :output)
      loop(:high)
  end
  defp loop(level) do
      GPIO.digital_write(@pin, level)
      Process.sleep(200)
      loop(toggle(level))
  end
  defp toggle(:high), do: :low
  defp toggle(:low), do: :high
end
```

See also:
https://github.com/atomvm/AtomVM/tree/main/examples

AtomVM

What's a GPIO?

- **GPIO** stands for **G**eneral **P**urpose **I**nput/**O**utput.
- Think of them as simple digital pins that can be either an input or an output
- They can be set to high (e.g., `3.3V`) or low (`0V` / Ground).

For our LED, this means it's either fully on or fully off. No fading.

AtomVM

The AtomVM Workflow

- Add `{:exatomvm, github: "AtomVM/exatomvm", runtime: false}` to `mix.exs` ✓
- Write Elixir code (like always!) ✓
- (Behind the scenes: compile, like always!)
- (Behind the scenes: pack, `mix.atomvm.packbeam → myapp.avm`)
- **Flash, run one command: (e.g. `mix atomvm.esp32.flash`)**

Remember: AtomVM runs unmodified BEAM file, so any language that runs on the BEAM, will run on AtomVM.

AtomVM

Demo

HONESTLY I WILL NOT DO A BLINKING LED DEMO. TRUST ME IT WORKS. I WILL NOT EVEN TRY SHOWING A MICROSCOPIC LED TO THE AUDIENCE. IF I BRING BOARDS AND ELECTRONICS ON A AIRPLANE I MIGHT BE MISIDENTIFIED AS A TERRORIST. DEMOS ALWAYS FAIL ANYWAY.

AtomVM

Time for `minicom -D /dev/ttyACM0`

- After flashing, use a serial monitor (like minicom) to read `IO.puts` and `IO.inspect` output
- Don't run the serial monitor and the flasher at the same time
- It may require some configuration

AtomVM

Circuits Pro-tips

Don’ts

1. **Never mess with “GND”** (the ground): if you connect GND pin to something that is not ground/0V you are likely going to fry your device
2. **Respect polarity:** components like LEDs and some capacitors have positive and negative sides: connecting them backward = ☠️
3. **Don’t mix voltage levels**: sending 5V into a 3.3V pin can permanently damage the chip unless the pin is explicitly *'5V tolerant'*
4. **Always connect an antenna**: before powering on a radio. Without it, the transmitter can be damaged

Do: **Double-check all your connections before powering up your device!**

AtomVM

Let’s Build an ePaper Dashboard in Elixir

Let’s walk through a simple project:

we’ll build an ePaper frame that shows the weather.

For hardware, we’ll use a Seeed Studio ePaper display based on **ESP32**.

AtomVM

Handling a Button Press

Goal: We want to know when a button is pressed

- The naive way is "polling": constantly looping to check the button's GPIO pin
  - The problem? This keeps the CPU busy doing nothing and drains the battery. This is called "busy-waiting"

- A better way: Interrupts
  - A hardware interrupt tells the CPU to pause its current task and handle something important *right now*
  - In AtomVM, we translate these hardware interrupts into standard Elixir messages

AtomVM

Interrupts in Elixir

First, we configure the GPIO as an input and tell it to trigger an interrupt on a 'falling edge' (when the button is pressed):

```
:gpio.set_direction(gpio, gpio_num, :input)
:gpio.set_int(gpio, gpio_num, :falling)
```

The **hardware event is safely delivered to your process's mailbox as a message**. No callbacks, no polling—just the actor model you already know and love. e.g., let's add to our GenServer:

```
def handle_info({:gpio_interrupt, gpio_num}, state) do
    IO.puts "Button pressed"
    {:noreply, state}
end
```

AtomVM

Source: https://www.flickr.com/photos/tambako/31556104335/in/photostream/

Great, I'm blinking an LED and reading a button. Now what?

AtomVM

Peripherals!

- Connecting to the outside world: peripherals!
- Usually, a *bus* lets you connect multiple devices to the same set of GPIO pins
- Many flavors
  - **I2C** → 2 wires, synchronous, slow-to-medium speeds
  - **SPI** → 4-wire bus (or more), synchronous, faster than I²C, good for displays
  - **UART** → 2-wire bus, point-to-point connection. TL;DR: a serial port
- Most sensors, displays, and other modules use one of these standard communication protocols (or "buses")

AtomVM

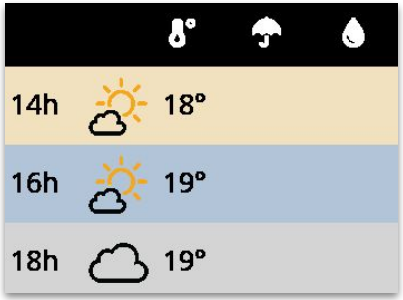
Native vs Elixir Components

- AtomVM APIs (`spi`, `i2c`, `gpio`) let you build libraries for **complex peripherals in pure Elixir/Erlang (no C required).**

- High-performance or very low-level hardware access?
  - Use **Native components (NIFs and Ports)**, just like the regular BEAM
  - The catch: using native components requires a custom AtomVM build using the device's SDK (like the ESP-IDF)

AtomVM

Displaying stuff: AtomGL



- A native component for rendering to a display
- Displays any list of items:

```
{:text, 16, 16, :default16px, 0xFFFFFF, 0x404040, title},
{:image, div(320 - 64, 2), div(240 - 64, 2), 0x404040, error_image}
```

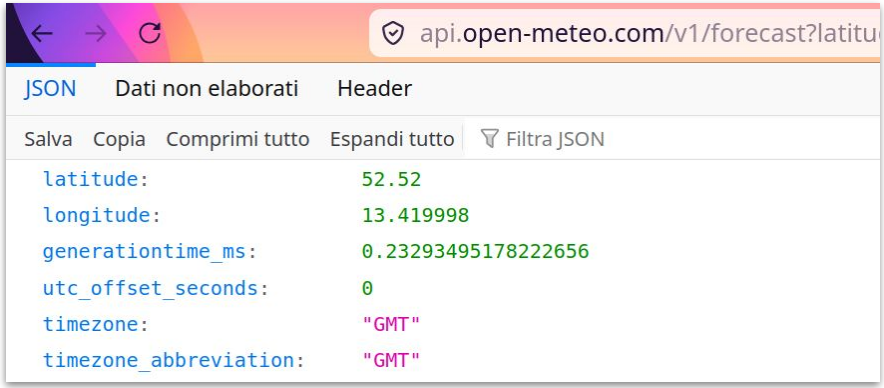
- There is an additional `avm_scene` library that provides some scaffolding for managing displayed scene, using a `gen_server` like approach:

```
def handle_info(:show_foo, %{width: width, height: height} = state)
do
    {:noreply, state, [{:push, items}]}
end
```

See also: https://github.com/atomvm/atomgl

AtomVM

What about I in IoT?



- There is a handy `network` module
- AtomVM has support for `gen_tcp`, `gen_udp` and `socket`
- `http_server` and `ahttp_client` modules
- `mdns` for finding your device on the network

See also: https://doc.atomvm.org/latest/network-programming-guide.html

AtomVM

Setting up Wi-Fi (& Handling HTTP Requests *)

```
config = [sta: [
  ssid: "myssid",
  psk: "mypsk",
  connected: fn() -> …,
  disconnected: fn() -> …,
]]

:network.start(config)
```

```
router = [
  {"*", __MODULE__, []}
]
:http_server.start_server(8080, router)
[...]
def handle_req("GET", [], conn) do
  body =
    "<html>\n" <> [...]
  :http_server.reply(200, body, conn)
end
```

* http_server is not yet used in the weather display project.

AtomVM

AtomVM has some additions/differences

Compared to the regular BEAM, AtomVM has some extensions to the standard BEAM (usually prefixed with atomvm or avm), e.g.:

- `:atomvm.read_priv/2` →reads a binary file stored in a loaded .avm file
- `:atomvm.posix_*` → posix functions, they mimic unistd.h ones

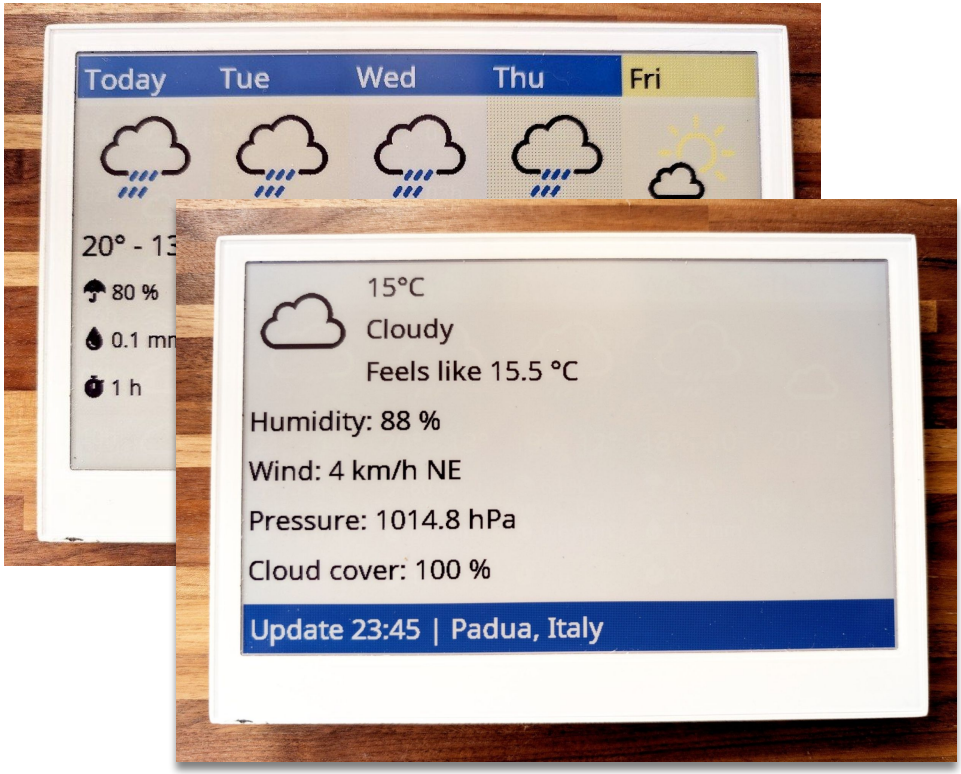
.beam files are packed into .avm files, that are designed to be written directly to flash memory (no filesystem needed).

(BTW: they all start with `#!/usr/bin/env AtomV` →

next idea: making startup-time-free CLI tools with AtomVM ;))

AtomVM

It Works!



Building apps like this weather dashboard for an ePaper display with Elixir and AtomVM isn't just possible, it's surprisingly **easy and a lot of fun**.

Now it's your turn.

Try it, hack on it, build your own frame:

https://github.com/**bettio/frameOS**

AtomVM

Closing Words on AtomVM

Releases

We're getting close to the AtomVM v0.7 release.

- It includes a lot of cool features: JIT, big integers, distributed Erlang, OTP 29 support, and more.
- For the best up-to-date experience, **use the release-0.7 branch** or download a v0.7 pre-release from GitHub.

(Just a heads-up: we're planning to merge a lot of breaking changes into the main branch.)

AtomVM

Support AtomVM

AtomVM has grown a lot, and maintaining it has become a significant effort.

New issues, PRs, and ideas come in every day, and supporting the project now takes full-time work.

Starting this year, I'm working on AtomVM full-time,
and **GitHub Sponsors helps make that sustainable**.
If AtomVM is useful to you or your company,
please consider sponsoring the project.

… And of course, contributions on GitHub are very welcome too!

AtomVM

Join Us

https://atomvm.org/

Discord: https://discord.gg/QA7fNjm9Nw

Telegram: https://t.me/atomvm

Documentation: https://doc.atomvm.org/

AtomVM

Thanks