
AtomVM

About me (Davide Bettio)

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

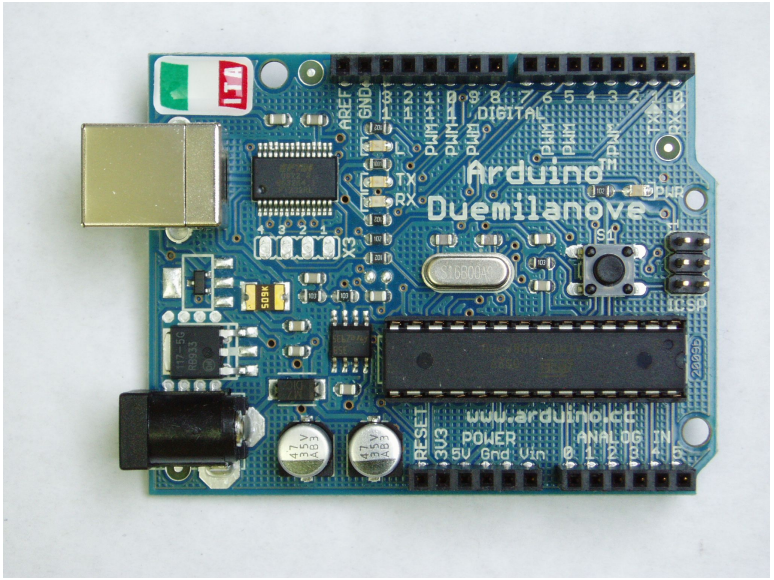
- Tinker with hardware and embedded systems since 2004.
- Long-time open-source dev (since ~2005 contributed to KDE Plasma and others).
- Fell in love with Elixir in 2017, while creating Astarte Platform.
- Started the AtomVM project in 2017
- Currently work at SECO Mind (formerly Ispirata)
- I love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

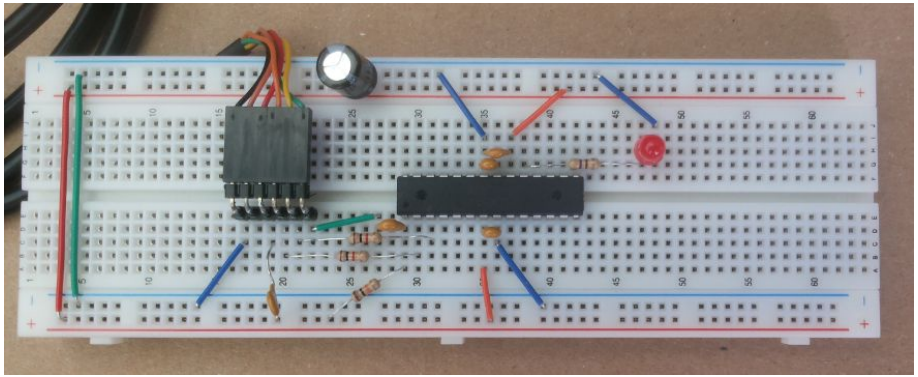
AtomVM

AtomVM

Once Upon a Time, the Arduino

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

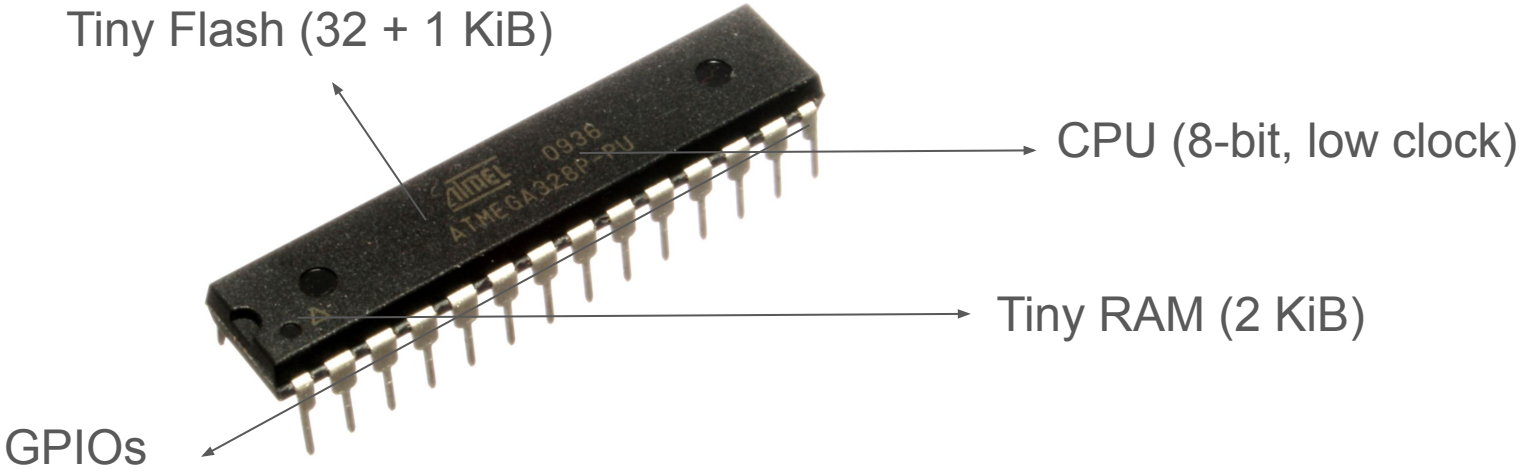




AtomVM

Classic MCU Anatomy (e.g., ATMega328P)

A small Computer on a Chip:

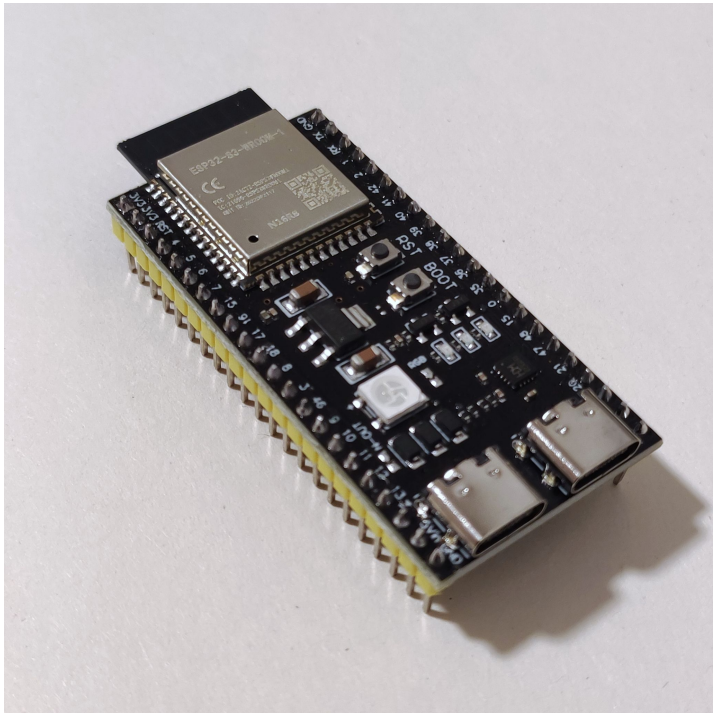


AtomVM

Modern MCU: ESP32 Example

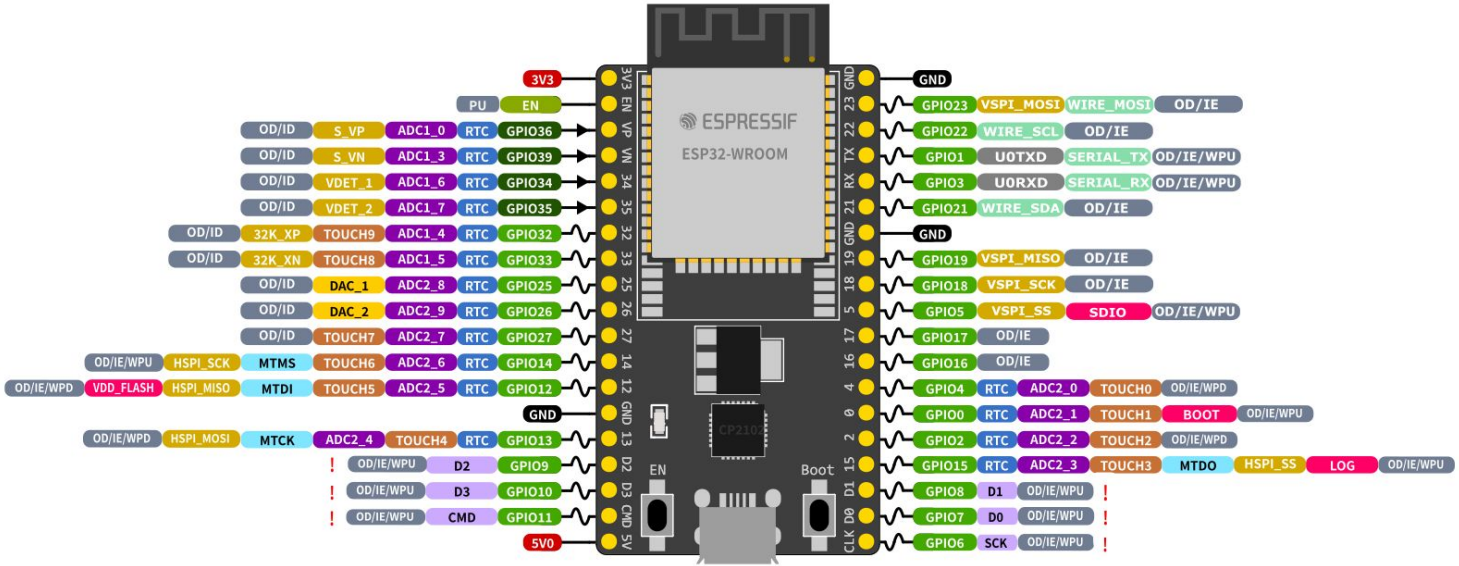
ESP32:

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



AtomVM

ESP32-DevKitC



ESP32 Specs

32-bit Xtensa® dual-core @240 MHz
Wi-Fi IEEE 802.11b/g/n 2.4 GHz
Bluetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

PWM Capable Pin
GPIOX — GPIO Input Only
GPIOX — GPIO Input and Output
DAC_X — Digital-to-Analog Converter
DEBUG — JTAG for Debugging
FLASH — External Flash Memory (SPI)
ADCX_CH — Analog-to-Digital Converter
TOUCHX — Touch Sensor Input Channel
OTHER — Other Related Functions
SERIAL — Serial for Debug/Programming
ARDUINO — Arduino Related Functions
STRAP — Strapping Pin Functions

RTC — RTC Power Domain (VDD3P3_RTC)
GND — Ground
PWD — Power Rails (3V3 and 5V)
! — Pin Shared with the Flash Memory Can't be used as regular GPIO

GPIO STATE

WPU: Weak Pull-up (Internal)
WPD: Weak Pull-down (Internal)
PU: Pull-up (External)
IE: Input Enable (After Reset)
ID: Input Disabled (After Reset)
OE: Output Enable (After Reset)
OD: Output Disabled (After Reset)

Modern MCU: RP2040 (Pi Pico) Example

Raspberry Pi Pico (RP2040):

- Cost < 5 €
- Dual Core @ 133MHz+
- RAM: 264KiB+
- Flash: 2MB+ (via QSPI)
- GPIOs, Periph. & Programmable I/O (PIO)
- WiFi option
- Low power

AtomVM

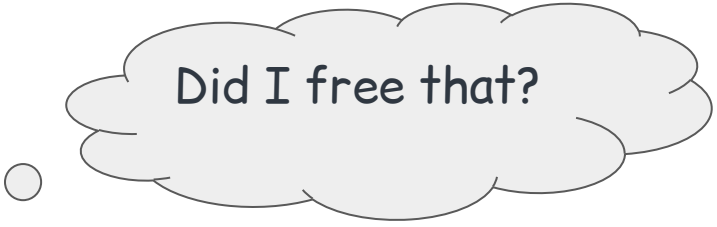
Powerful, But Still...Different

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

AtomVM

The C/C++ Experience on MCUs

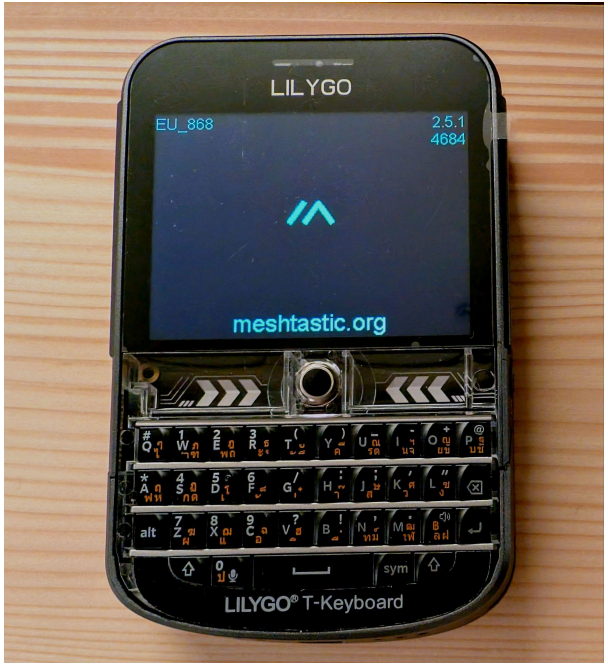
- Concurrency? Manual, tricky.
- Binary parsing? Boring & dangerous.
- Async? Callback hell, anyone?
- Memory?



AtomVM

The Intricacies of Embedded Communication: LoRa

- LoRa: Long-Range radio, raw bytes to CPU
- Need to implement: routing, security, mesh
- Meshtastic parses them in C++
    - C++: One wrong move... 💣



AtomVM

Clarity in Complexity for LoRa Packets

```
def parse(
    <<dest::little-unsigned-32, src::little-unsigned-32, pkt_id::little-unsigned-32,
      hop_start::size(3), via_mqtt::size(1), want_ack::size(1),
      hop_limit::size(3), channel_hash::8, _padding::16, encrypted_data::binary>>) do
  {:ok, %{dest: dest, src: src, packet_id: pkt_id,
          hop_start: hop_start, via_mqtt: int_to_bool(via_mqtt), want_ack: int_to_bool(want_ack),
          hop_limit: hop_limit, channel_hash: channel_hash, encrypted_data: encrypted_data}}
end

def parse(_), do: {:error, :failed_meshtastic_parse}

def decrypt(%{src: src, packet_id: pkt_id, encrypted_data: enc_data} = packet, key) do
    iv = <<pkt_id::little-unsigned-64, src::little-unsigned-32, 0::32>>

    decrypted = :crypto.crypto_one_time(:aes_128_ctr, key, iv, enc_data, false)
    packet
    |> Map.put(:data, decrypted)
    |> Map.delete(:encrypted_data)
end

defp int_to_bool(0), do: false
defp int_to_bool(1), do: true
```

AtomVM

Projects like Meshtastic couldn't leverage these advantages on such microcontrollers. **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 and Elixir 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

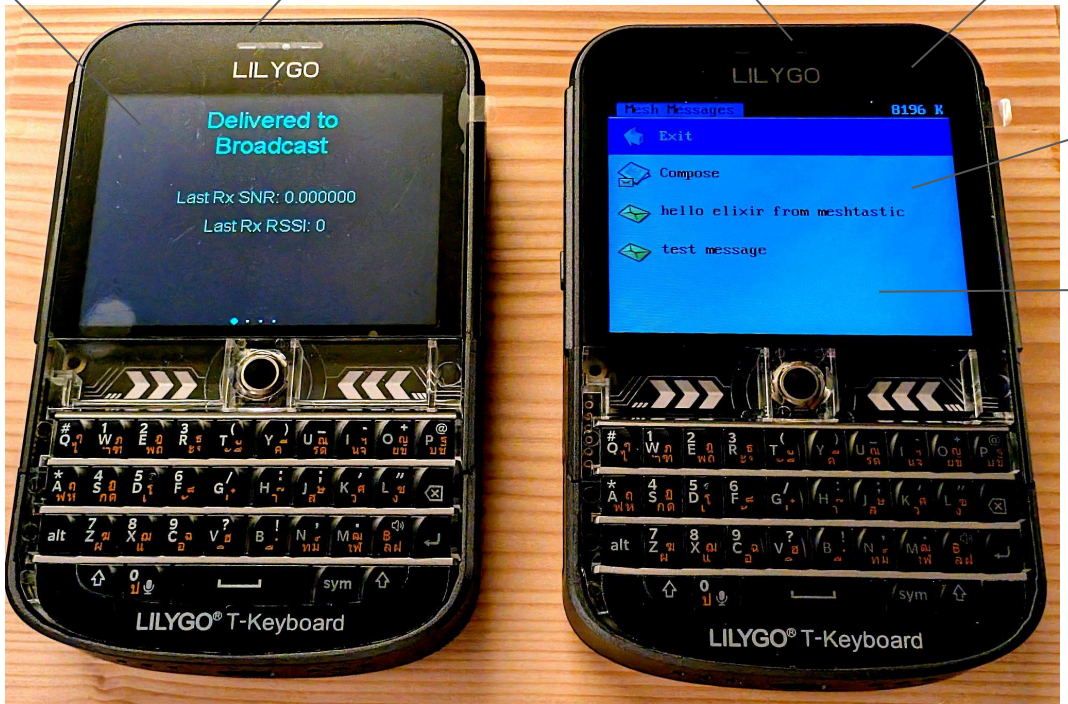
AtomVM

To the Rescue

Original firmware in C++

Battery powered
CPU: ESP32-S3
Radio: LoRa

Runs AtomVM



Display managed using https://github.com/atomvm/**AtomGL** component.
UI is made with Elixir

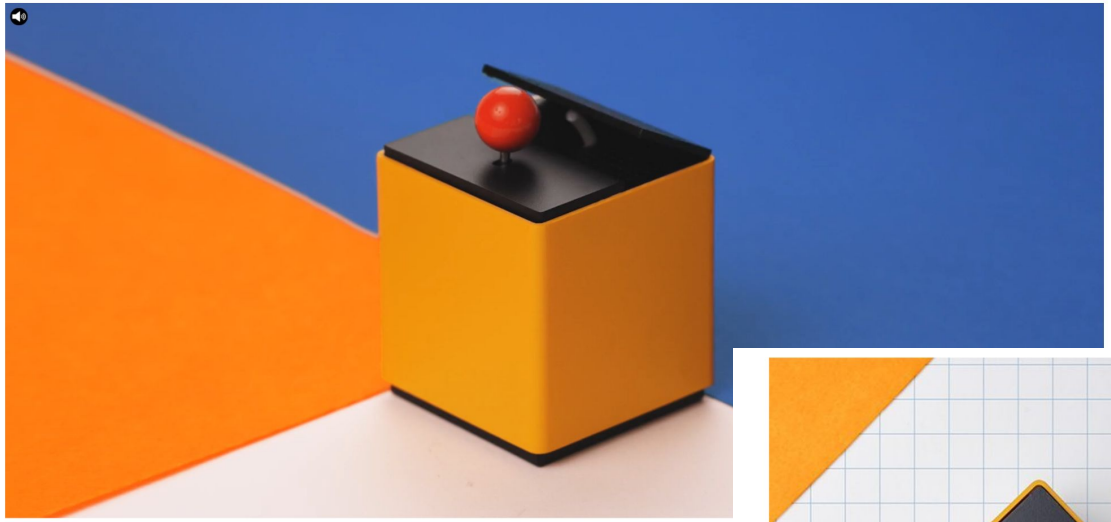
Runs an Elixir app, no native parts are used (github.com/bettio/pocketOS/)

AtomVM

la machine

by multiplié

The Useless Box : Reloaded



- AtomVM powered
- ESP32-C3
- 32-bit RISC V single core @ 160 MHz 400 KB of SRAM
- 5µA in deep sleep !

- la machine code is in Erlang
- uses atomvm_esp_adf component for playing audio from Erlang code (thanks Paul)

KEY FEATURES

- Over 500 unique sound effects & reactions
- Unlimited choreography combinations - never the same twice
- Fully modular design for easy repairs & customization
- Powered by ESP32 architecture
- Completely open source software - hack it, modify it, make it yours
- Eco-friendly construction from 100% recycled materials
- Exceptional battery life: three months on a single charge

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
```

AtomVM

The AtomVM Workflow

- Add `{:exatomvm, github: "AtomVM/exatomvm", runtime: false}` to `mix.exs`
- Write Elixir/Erlang (like always!)
- Compile (like always!)
- Pack (`mix.atomvm.packbeam` → myapp.avm)
- Flash (e.g. `mix atomvm.esp32.flash`)

TL;DR: just `mix atomvm.esp32.flash`

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

AtomVM

Quick Stats & Nerves Comparison

AtomVM:

- AtomVM, hello world footprint: 512 KiB of flash, 32 KiB of RAM
- Targets smaller MCUs (no Linux / no OS at all)

Nerves:

- Awesome on capable devices (RPi, etc.), such as those running Linux

AtomVM

Big Caveat

- Some features, standard modules or functions are missing (e.g. digraph module)
- But `exatomvm` will do its best to tell you if you are using any missing feature, so you can quickly iterate before flashing your application

AtomVM

What's Next

New: Erlang Distribution (thanks Paul) and other 40+ additions

Soon:

- Big Integers (WIP, limited to 256-bit)
- Bitstrings (next release!)

Future:

- More devices & peripherals (Zephyr devices, Bluetooth, Zigbee/Thread, etc…)
- Even better tooling & DevX
- Stable APIs (path to 1.0)

AtomVM

So, we've seen how AtomVM brings the power of the BEAM to the microcontrollers

But the core idea – a portable, lightweight BEAM-like runtime – opens up fascinating possibilities...

What if we could take this capability beyond just hardware? What if we could run this same Elixir/Erlang logic in **other constrained environments?**