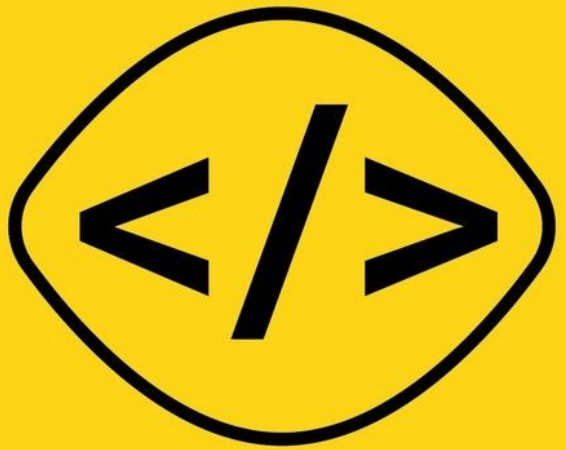


murcia.dev

Una chispica sobre Sistemas Embebidos

Sobre mí

- Me llamo **Eduardo Martínez Martínez**
- Oriundo de **Murcia** 🇪🇸🍋
- Me gusta la naturaleza, la cultura local y los ordenadores 🐢💻
- Graduado en Ingeniería Informática por la Universidad de Murcia
- He trabajado como ingeniero informático en:
  - AED Vantage
  - Qualcomm

¿Qué son los **sistemas embebidos**?

¿Habéis desarrollado algo o trabajáis en alguna empresa que los apliquen?

Los sistemas embebidos son sistemas informáticos **especializados** e íntimamente ligados a la **electrónica**, y pueden o no **formar parte dentro** de un sistema mecánico o electrónico más grande.

Es difícil imaginar qué son solo con una definición…

Por ello, vamos a ilustrar con un enfoque en el **producto** y con **ejemplos** las necesidades que se cubren con dichos sistemas.

Ejemplos de sistemas embebidos

Dispositivos móviles

Periféricos

https://www.newaltte
ch.com/

Dispositivos ponibles

Audio


com
shift
rnd
XY


Hi-Res
AUDIO
FiiO
BAL.
BASS
GAIN


WiiM Amp Pro

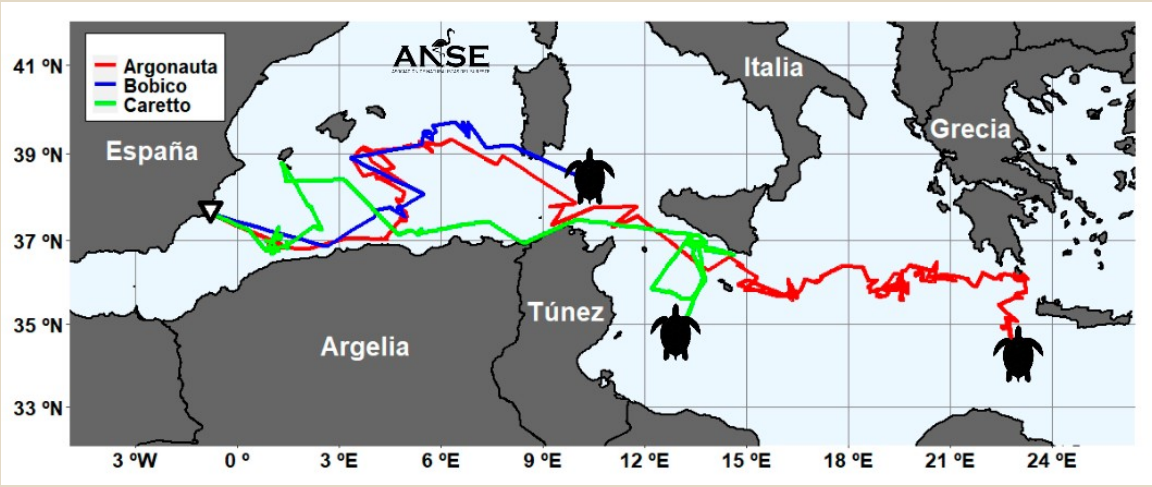

DDJ-FLX4
PLAY / PAUSE
TEMPO
BEAT FX
ON / OFF

Domótica

Internet de las cosas

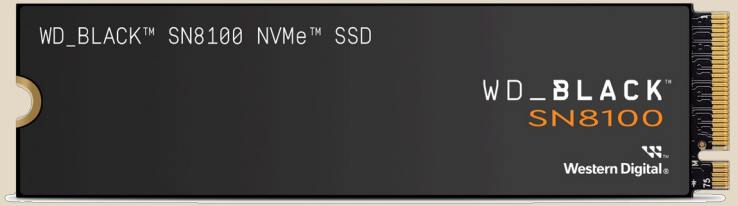
https://odins.es

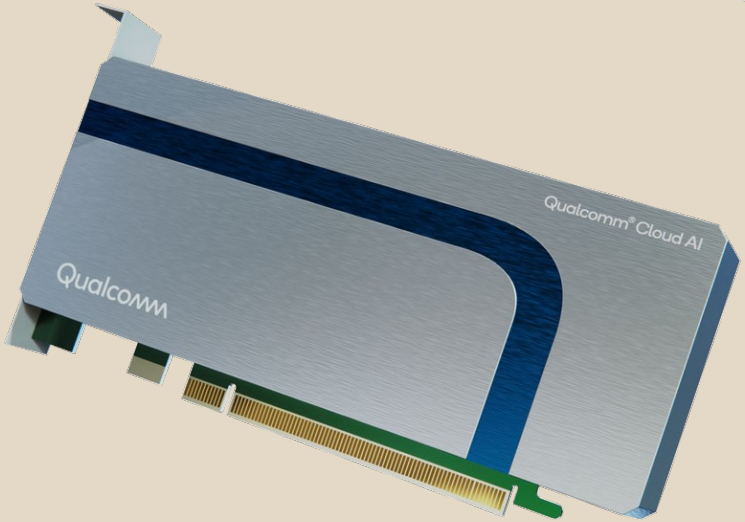
Internet de las cosas



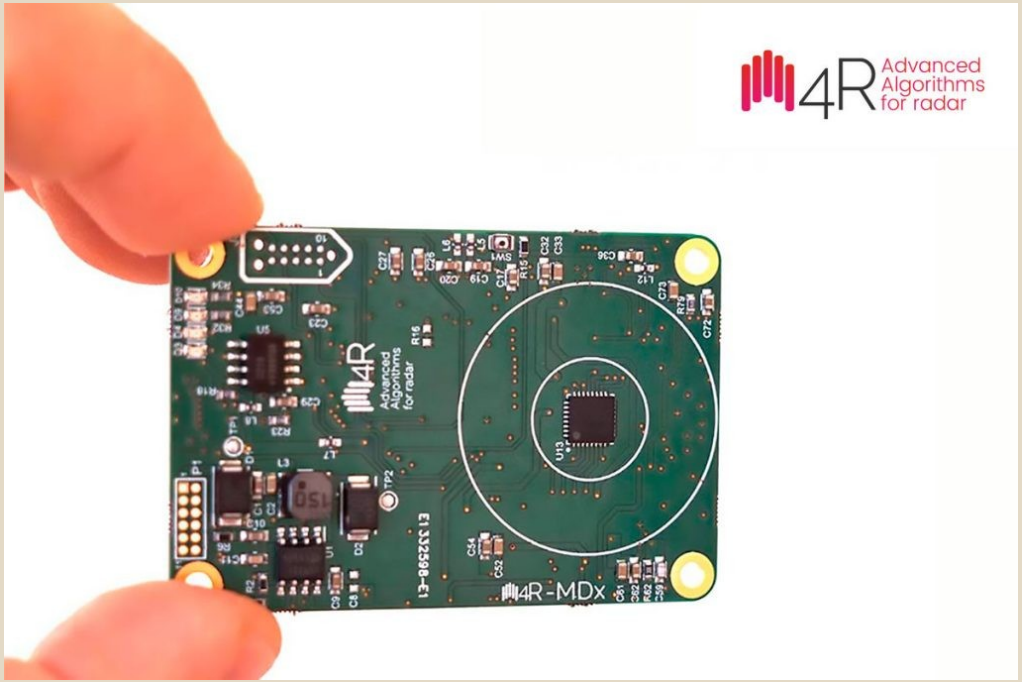
https://www.asociacionanse.org/seis-meses-en-la-vida-de-tres-tortugas-marinas-entre-calblanque-y-grecia/20210424/

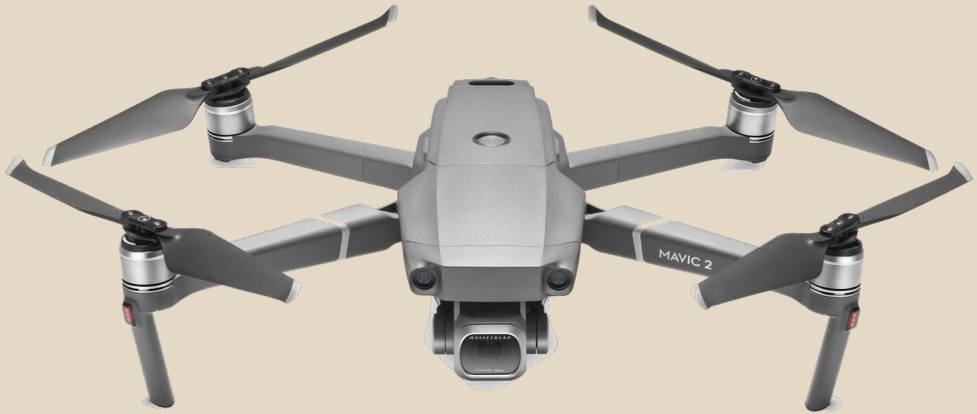
Arquitectura de ordenadores





Industria





https://a4radar.com/

Automoción


ECM Inputs
Camshaft Position Sensor
Mass Air Flow Sensor
Knock Sensor
Heated Oxygen Sensor
Vehicle Speed
VSS Sensor
Manifold Absolute Pressure
MAP Sensor
Crankshaft Position Sensor
Exhaust Gas Recirculation
EGR Sensor
Throttle Position
Coolant Temperature

Automoción

EL PAÍS 50

Tecnología

TU TECNOLOGÍA · CIBERSEGURIDAD · PRIVACIDAD · INTELIGENCIA ARTIFICIAL · INTERNET · GRANDES TECNOLÓGICAS · ÚLTIMAS NOTICIAS

AVANCE Consulte la portada de EL PAÍS, Edición Nacional, del 15 de febrero

TECNOLOGÍA PERSONAL >

Tu coche ya es un ordenador con ruedas: qué datos recoge y para qué se utilizan

Cómo y cuándo conduces, a dónde vas, con quién... Tu coche lo sabe todo de ti, y utiliza esa información para mejorar la conducción, calcular la póliza del seguro o venderlos a terceros

Automoción

https://www.aed-vantage.com/

Medicina


RA
RV
IS-1
IS-1
ACCOLADE™ MRI
Boston Scientific
ESSENTIO™ MRI
Boston Scientific

Medicina

Mano protésica controlada por análisis de los movimientos de la extremidad.

https://youtu.be/pnIUi9Jgbew

Aeroespacial

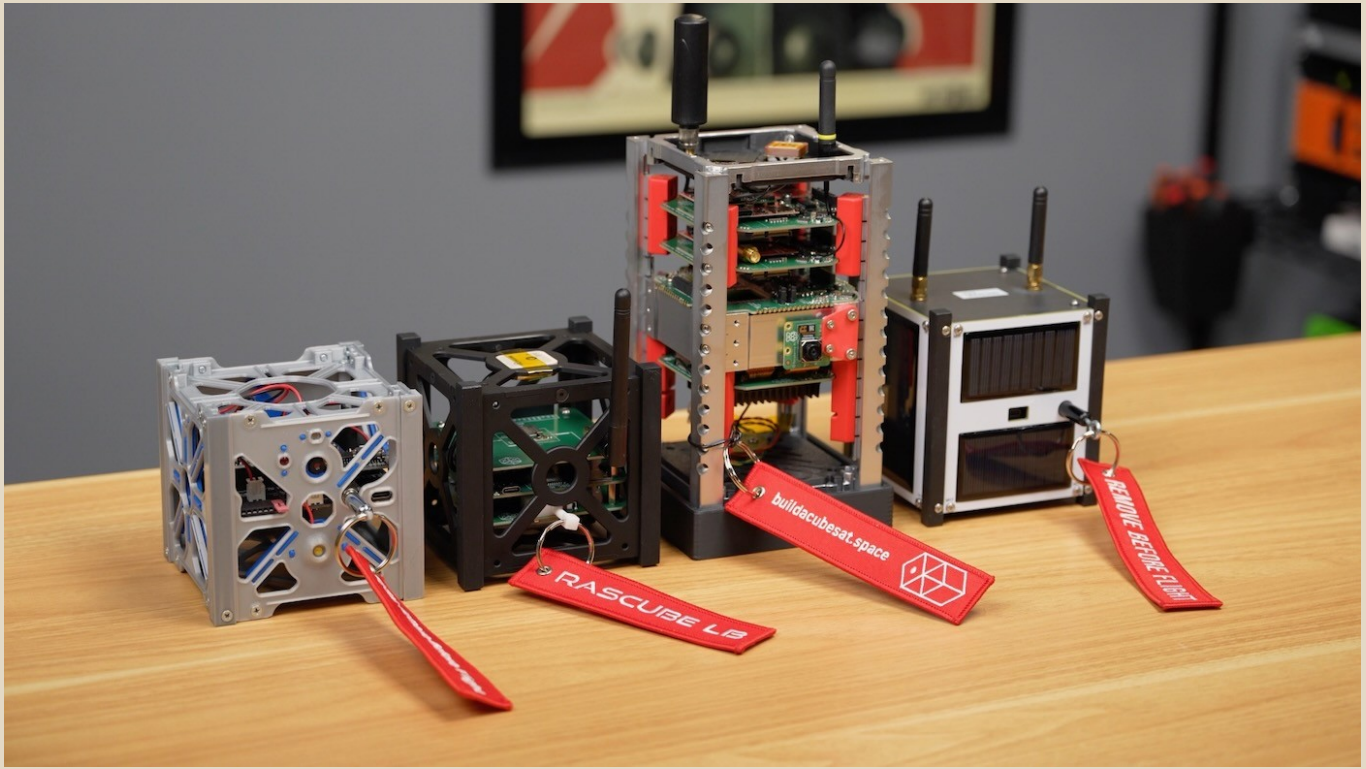
PLD SPACE™

Cohete MIURA 1, desarrollado por PLD Space (Elche, Alicante)

Aeroespacial

Cubesats

Satélites pequeños y económicos que se pueden poner en órbita reservando previamente un hueco en el lanzamiento de un cohete.



https://www.jeffgeerling.com/blog/2025/cubesats-are-fascinating-learning-tools-space/

Es **muy difícil** enumerar todas las aplicaciones posibles.

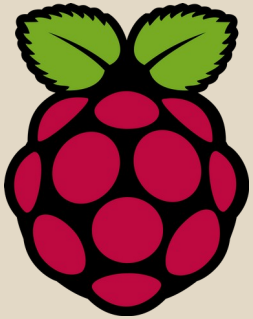
¡Incluso hay **discrepancias** sobre cuál es el límite de lo que es un sistema embebido!

¿Se os ocurre algún dispositivo **importante** que no haya mencionado?

En estos productos, ¿qué parte desarrolla alguien dedicado al **software**?

Normalmente, el **programa** que se carga en el procesador

Hay **muchos fabricantes** de procesadores para embebidos

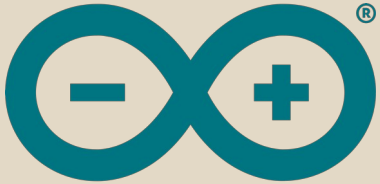


ESPRESSIF

NORDIC®
SEMICONDUCTOR

Rockchip

NXP

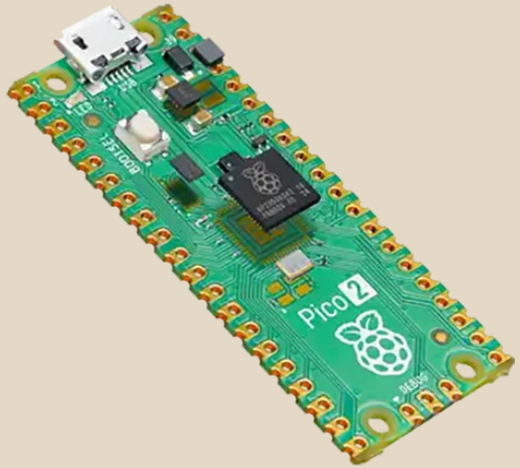


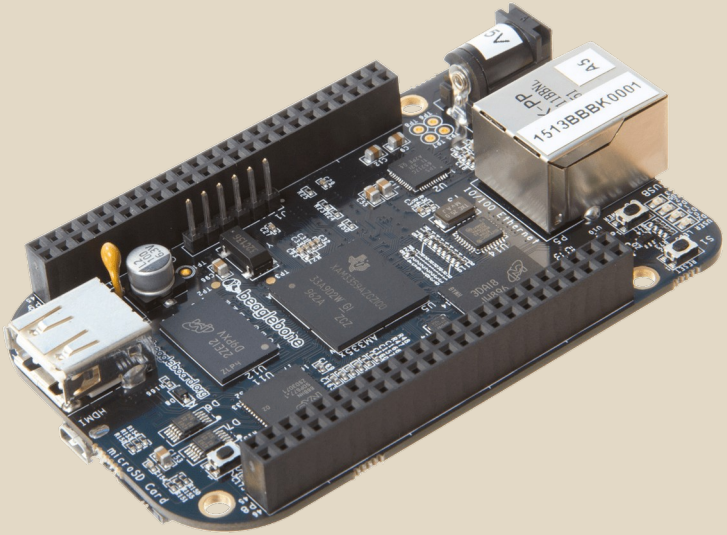
MICROCHIP

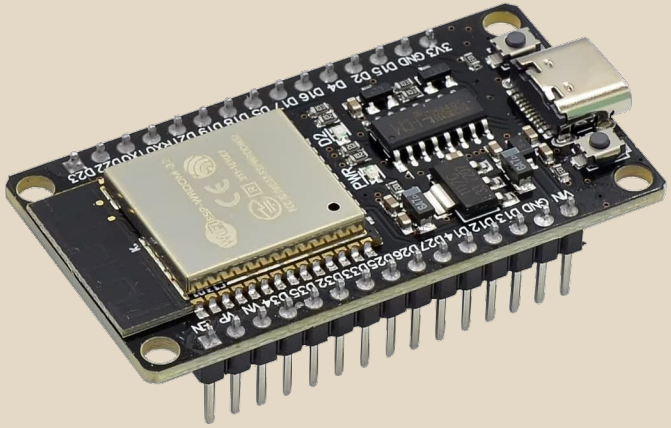
TEXAS
INSTRUMENTS

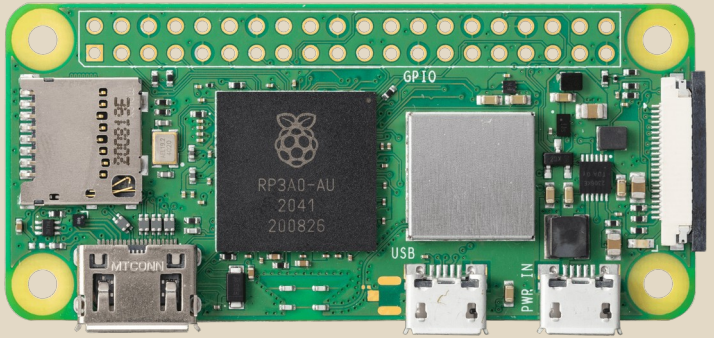
Qualcomm

Estos fabricantes ofrecen una placa llamada **Placa de evaluación** (EVB) para poder probar el procesador fácilmente

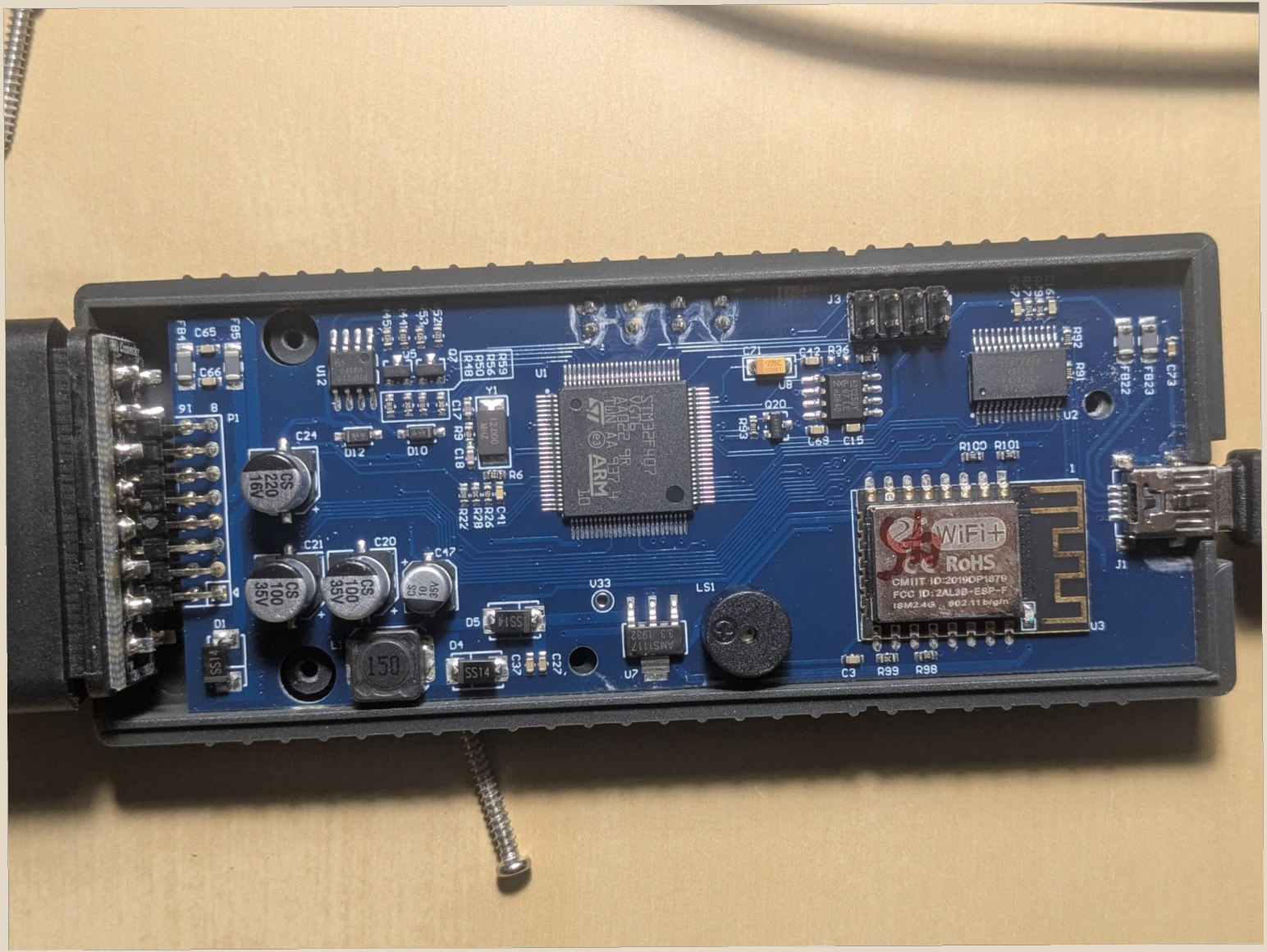

Pico 2


beagleboard




GPIO
RP3A0-AU
2041
200826
USB
PWR IN

Aunque se lanzan productos que directamente usan las EVB, normalmente se desarrolla una **Placa del Circuito Impreso** (*PCB*) personalizada

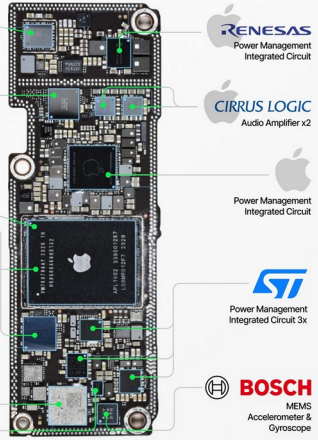

WiFi
RoHS
CMIIT ID:2019DP1879
FCC ID:2AL3B-ESP-F
ISM2.4G 802.11b/g/n
STM32F407
ARM
SS14
150
GS 220 16V
GS 100 35V
AMS1117 3.3 1932

Key chip suppliers for Apple's iPhone 15

This infographic was created by **Quartr** based on **TechInsights'** teardown of iPhone 15 Pro.

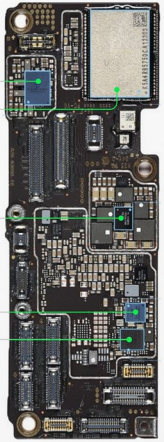
Quartr → www.quartr.com

Logic Board



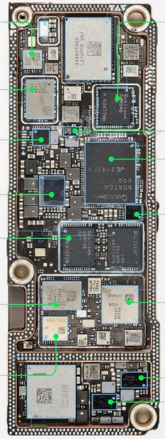
- BROADCOM — Wireless Charging Receiver
- TEXAS INSTRUMENTS — USB Interface or PMIC
- SK hynix — 8GB DRAM
- CIRRUS LOGIC — Audio Codec
- A17 Pro Processor, UWB U2
- RENESAS — Power Management Integrated Circuit
- RENESAS — Power Management Integrated Circuit
- CIRRUS LOGIC — Audio Amplifier x2
- Power Management Integrated Circuit
- Power Management Integrated Circuit 3x
- BOSCH — MEMS Accelerometer & Gyroscope

Memory Board



- CIRRUS LOGIC — Audio Amplifier, Logic Power Conversion
- KIOXIA — 256GB NAND Flash Memory
- TEXAS INSTRUMENTS — AMOLED Display Power Supply
- NXP — NFC Controller & Secure Element

Radio Frequency Board



- BROADCOM — FEM AFEM-8245
- SKYWORKS — FEM SKY503113
- NXP — NFC Controller
- Qualcomm — FR1 RF Transceiver x2, Power Management Integrated Circuit
- SKYWORKS — FEM SKY58440-11
- QORVO — FEM QM76305
- Qualcomm — FR2 RF Transceiver
- QORVO — APT (likely)
- Qualcomm — Snapdragon X70 Modem
- Secure MCU/eSIM
- BROADCOM — FEM AFEM-8234
- Qualcomm — Envelope Tracker x2

Infografía sobre componentes del iPhone 15

https://quartr.com/insights/company-research/the-suppliers-making-the-iphone-possible

¿Y cómo se **desarrollan** estos productos?

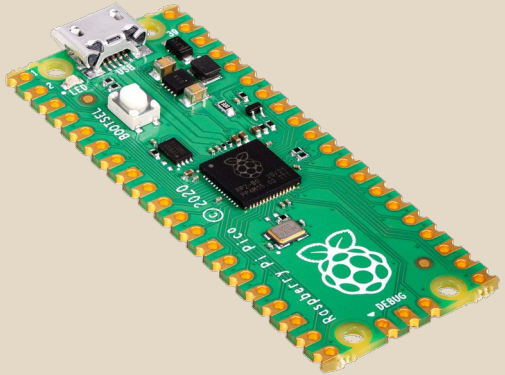
Todos los dispositivos anteriores comparten tecnologías **en común**.

Luego, según el **dominio** en el que se apliquen, hay algunas **diferencias**.

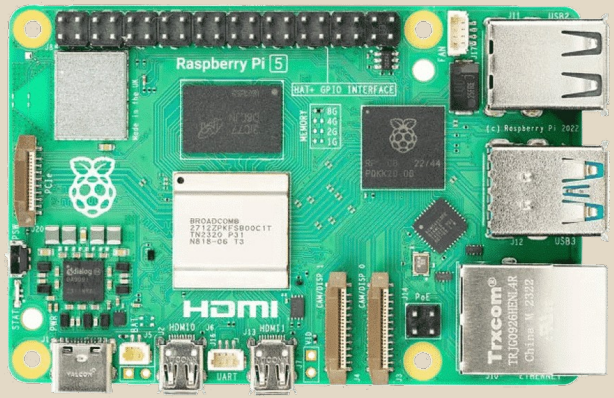
Diferencias entre **procesadores**

Según las **capacidades** del procesador principal:

- MCU (microcontrolador)
- MPU (microprocesador)



VS

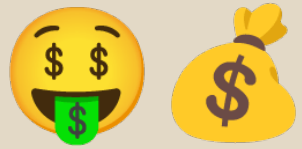


¿Qué es un **MCU**?

MCU

Un MCU es un procesador con capacidades muy **limitadas**.

Esto permite tener un **precio reducido** y fabricar el producto en masa a menor coste.



MCU

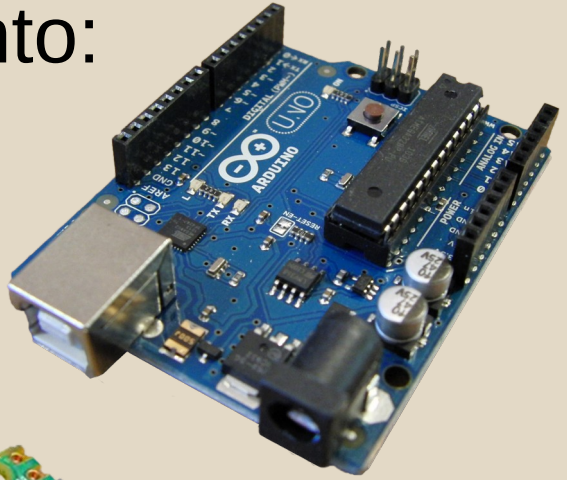
¿Qué tiene de limitado?

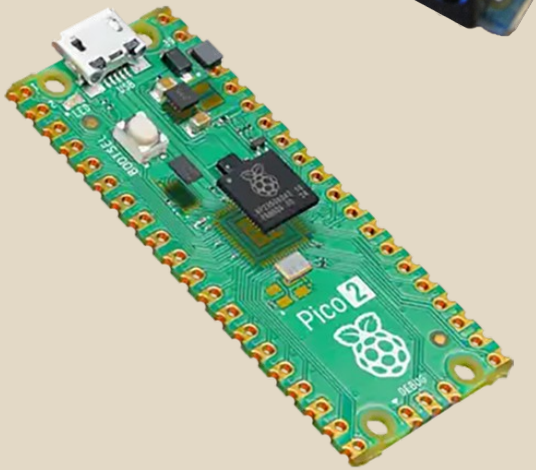
- Memoria y almacenamiento **reducidos**
- Ausencia de **MMU** (Unidad de gestión de memoria)

MCU

Ejemplos de memoria y almacenamiento:

- Arduino UNO (Rev. 3)
  - 2 KB de RAM
  - 32 KB flash, 1 KB EEPROM
- RP2350
  - 520 KB de RAM
  - 2 MB flash

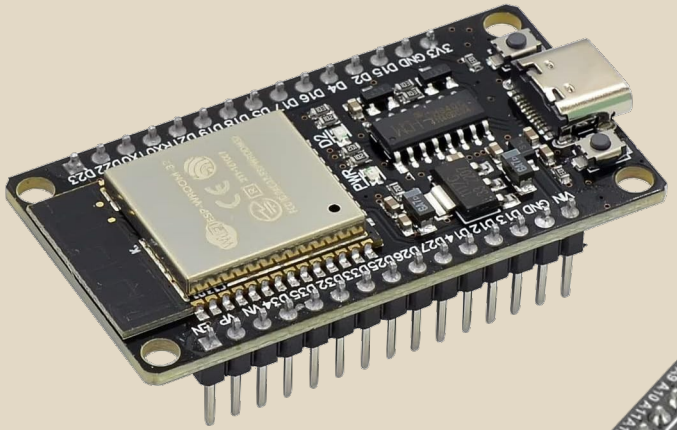


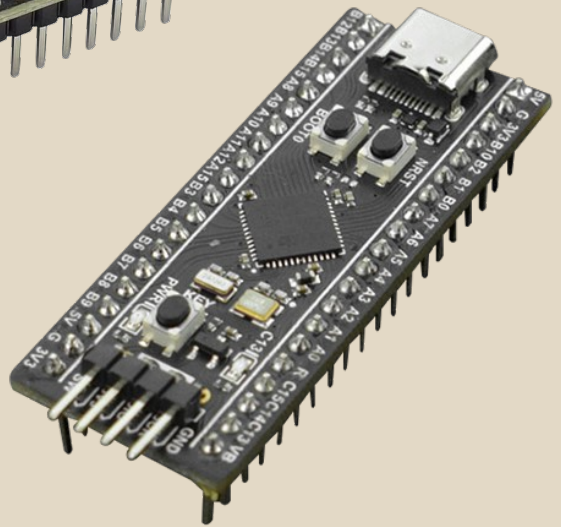


MCU

Ejemplos de memoria y almacenamiento:

- ESP32-S2
  - 320 KB de RAM
  - 4 MB flash
- STM32F411
  - 128 KB de RAM
  - 512 KB flash





MCU

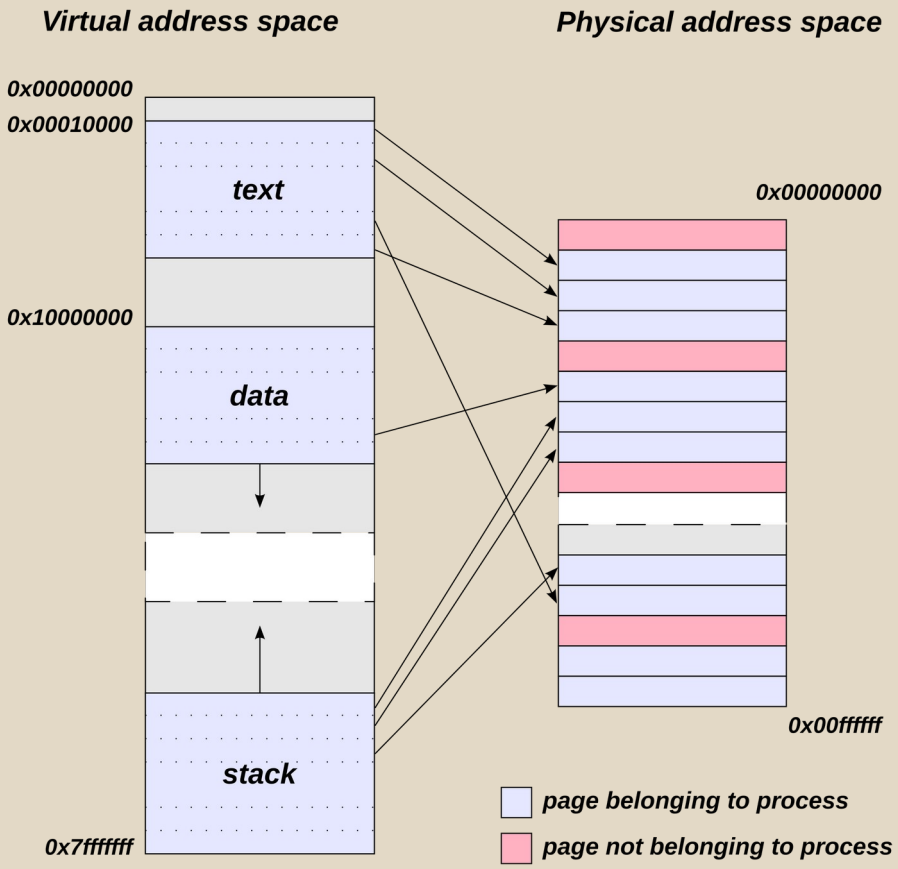
El otro factor, la **ausencia de MMU**, es más complicado de entender pero tiene **mucho impacto**.

MCU

La MMU, **traduce** las direcciones **virtuales** a direcciones **físicas**.

🤨❓

MCU



Fuente: Wikipedia

En un Sistema Operativo (SO) moderno, cada proceso en ejecución tiene su **espacio de memoria virtual.**

Este espacio es **homogéneo en forma** para todos los procesos, pero **difieren en su contenido**, ya que este se rellena dependiendo del programa.

La memoria virtual permite **aislar el espacio de memoria entre procesos**.

Además permite **diferir la carga en memoria de las diferentes partes del programa**. No todo el programa debe estar en memoria al mismo tiempo. 🤓

MCU

Sin MMU, no tenemos memoria virtual. **No podemos usar sistemas operativos modernos**. Por eso el desarrollo en MCU se conoce como **programación *bare-metal***.

MCU

Sin sistemas operativos, **ciertas utilidades se limitan bastante**:

- Memoria dinámica en el montón (*heap*)
- Sistemas de ficheros
- Acceso a una pila de red, como TCP/IP
- Etcétera

MCU

```cpp
#include <iostream>

int main() {
    // Lee un valor del usuario
    size_t arr_size{};
    std::cin >> arr_size;

    // Crea un array del tamaño del usuario
    auto arr = new int[arr_size];
    delete[] arr;

    return 0;
}
```

- Sin memoria montón

Sin un SO que proporcione un ***allocator*** para reservar montón, no podemos asignar memoria **en tiempo de ejecución**.

A no ser que proporcionemos **un *allocator* adaptado a *baremetal***, solo podemos usar **la pila** (*stack*), que permite únicamente memoria **cuyo tamaño se conoce en tiempo de compilación**.

MCU

- Sin sistemas de ficheros

Linux soporta muchísimos sistemas de ficheros, como Ext4, Btrfs, ZFS, etc.

De nuevo tenemos que depender de **código diseñado específicamente para *baremetal***. En la práctica, esto supone una **preferencia** por sistemas basados **en FAT**.

MCU

- Sin pila TCP/IP

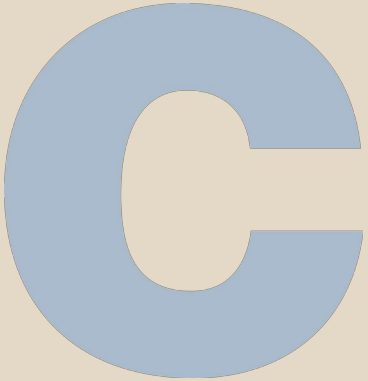
Ídem de lo mismo, hay ciertas funcionalidades limitadas pero algunos MCU permiten conectarse mediante Wi-Fi o Bluetooth **mediante sus propias implementaciones**.

MCU

Para acabar, el **código** suele volverse **dependiente del fabricante del MCU**, en vez de ser agnóstico para cualquier SO y procesador.

MCU

Tecnologías:



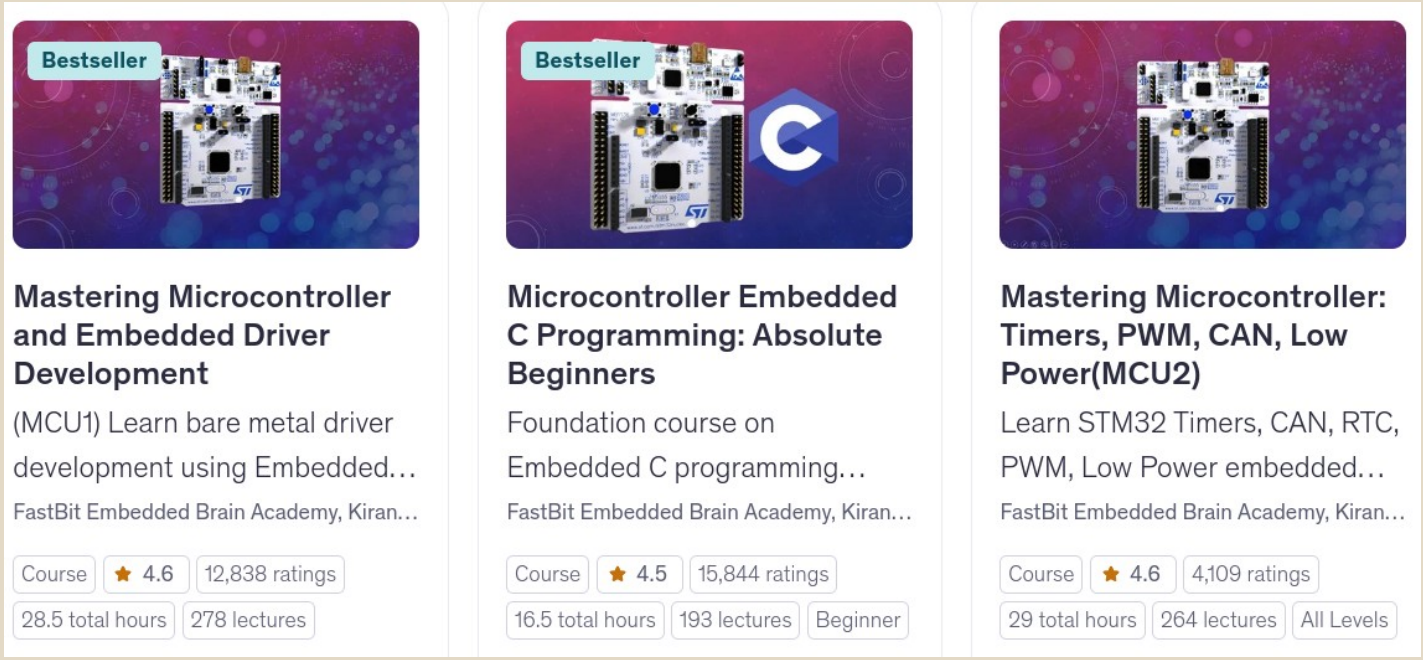
Zephyr™

+ SDK del fabricante 🥇

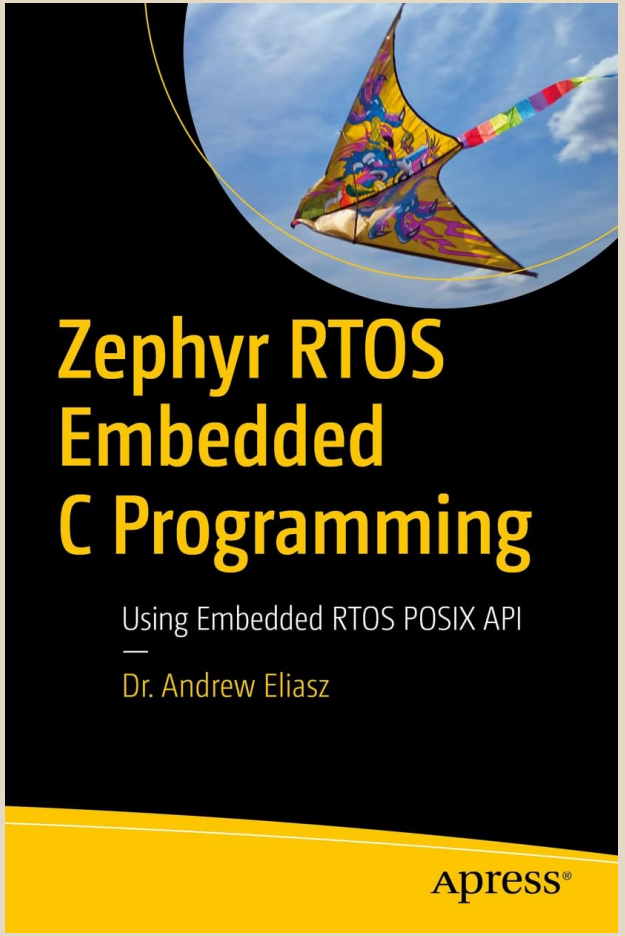
Experimental

MCU

Bibliografía:

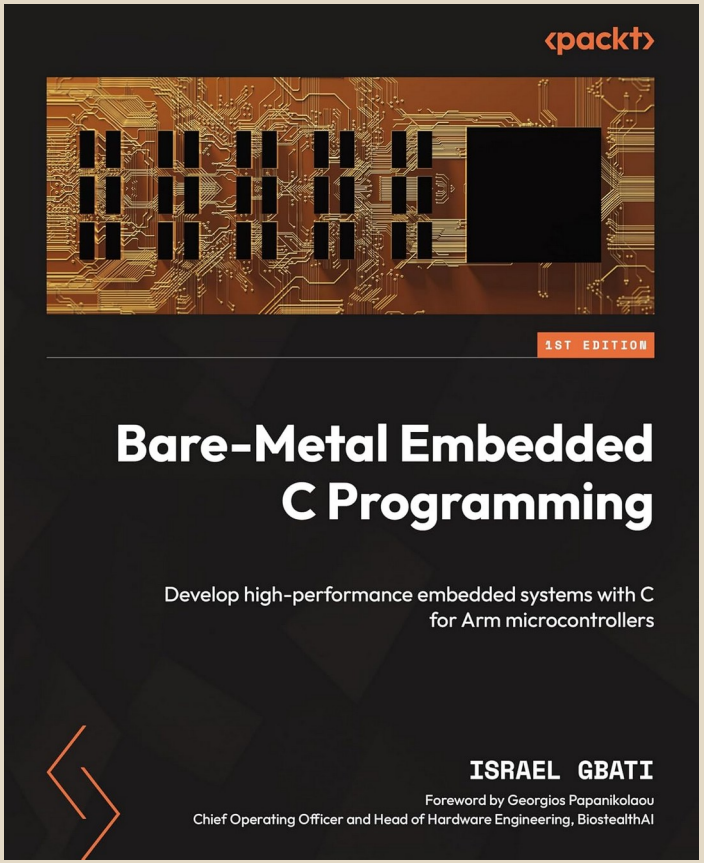


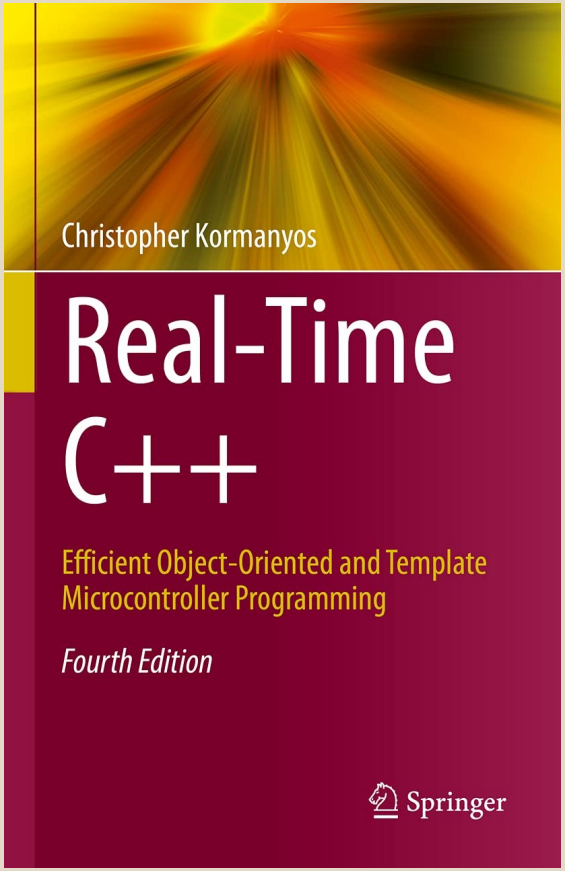
https://www.udemy.com/user/kiran-nayak-2/



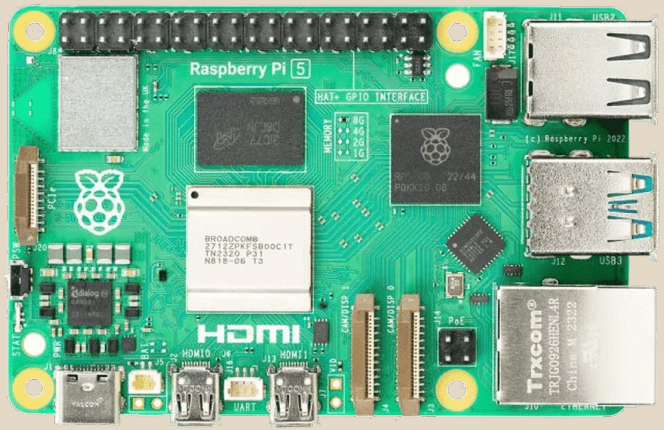
MCU

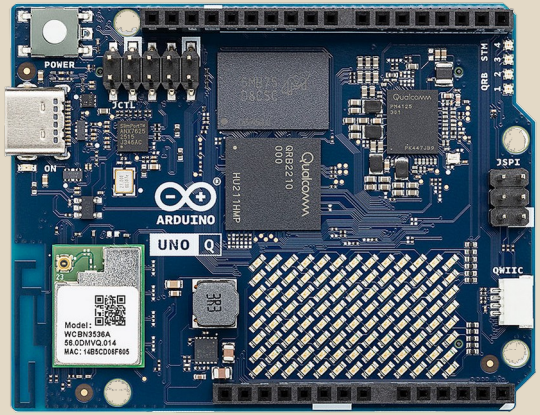
Bibliografía:

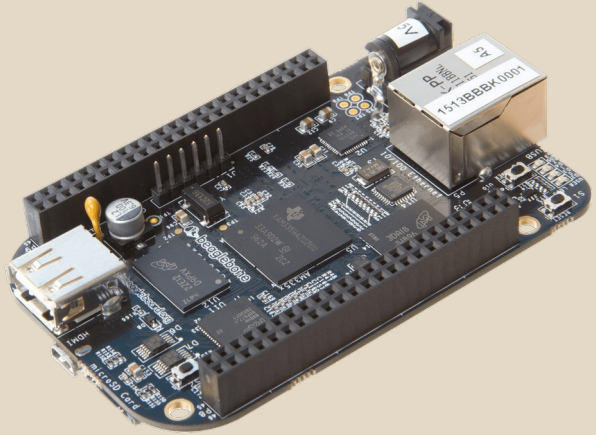


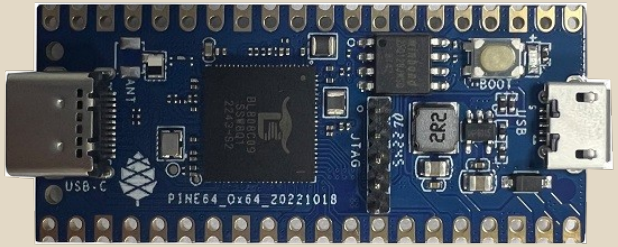


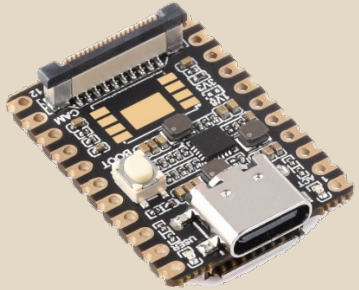
¿Y qué hay sobre **la alternativa**, los MPU?

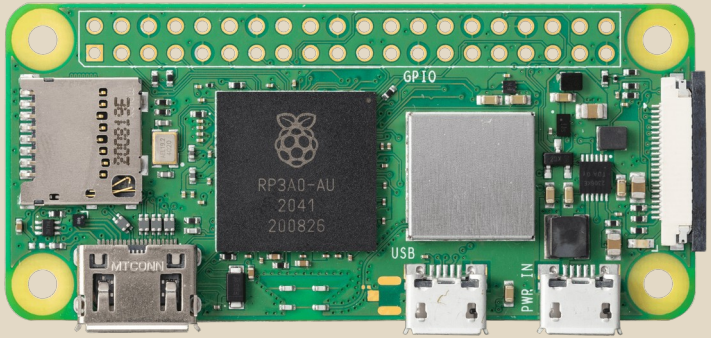










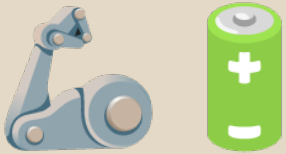


MPU

Todo **lo que no teníamos** con un MCU, **lo tenemos ahora** con un MPU

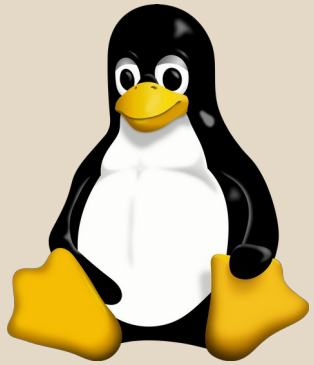
MPU

Tienen **más memoria y almacenamiento**, y como disponen de MMU, pueden **ejecutar un SO**.



MPU

El desarrollo de embebidos con una MPU usando SO se conoce como ***hosted***, como contraparte a ***bare-metal***.



MPU

Se puede programar en **espacio de usuario (no *drivers*)** con cualquier lenguaje de programación, siempre que no se excedan las capacidades de memoria o almacenamiento.

MPU

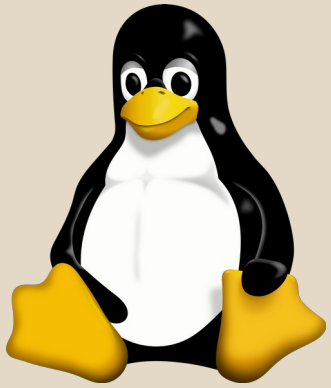
Es decir, el **código** vuelve a ser **independiente** del Sistema Operativo o del fabricante del procesador.

Además de **disponer** de todas las **comodidades** de un SO, como memoria *heap*, sistemas de ficheros avanzados, pila de red avanzada, etc.

MPU

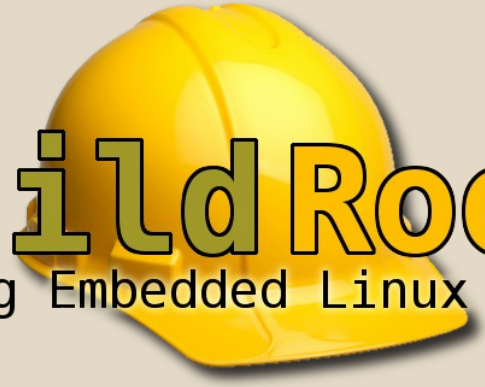
Tecnologías:

yocto PROJECT



U-Boot

Real-Time LINUX

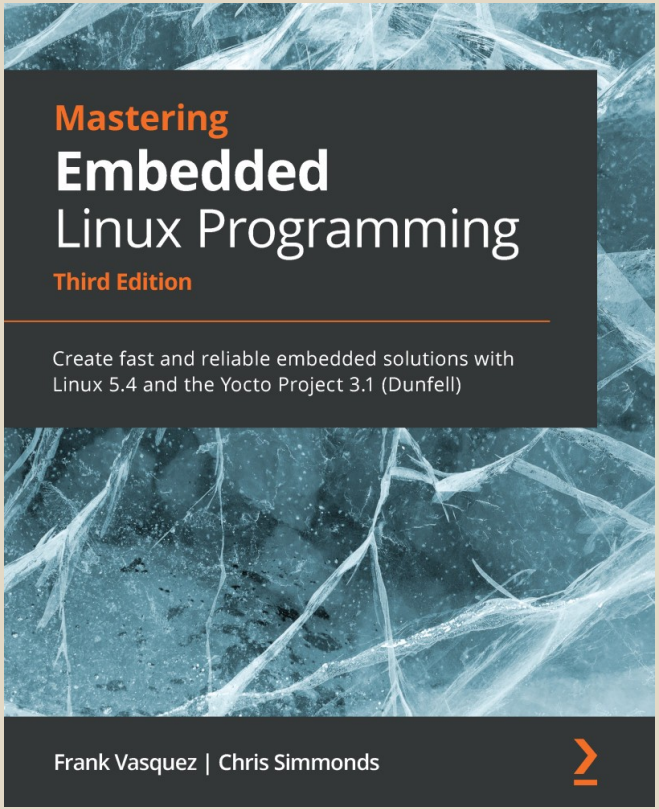


BuildRoot

Making Embedded Linux Easy

MPU

Bibliografía:



Requisitos de **Tiempo Real**

Tiempo Real

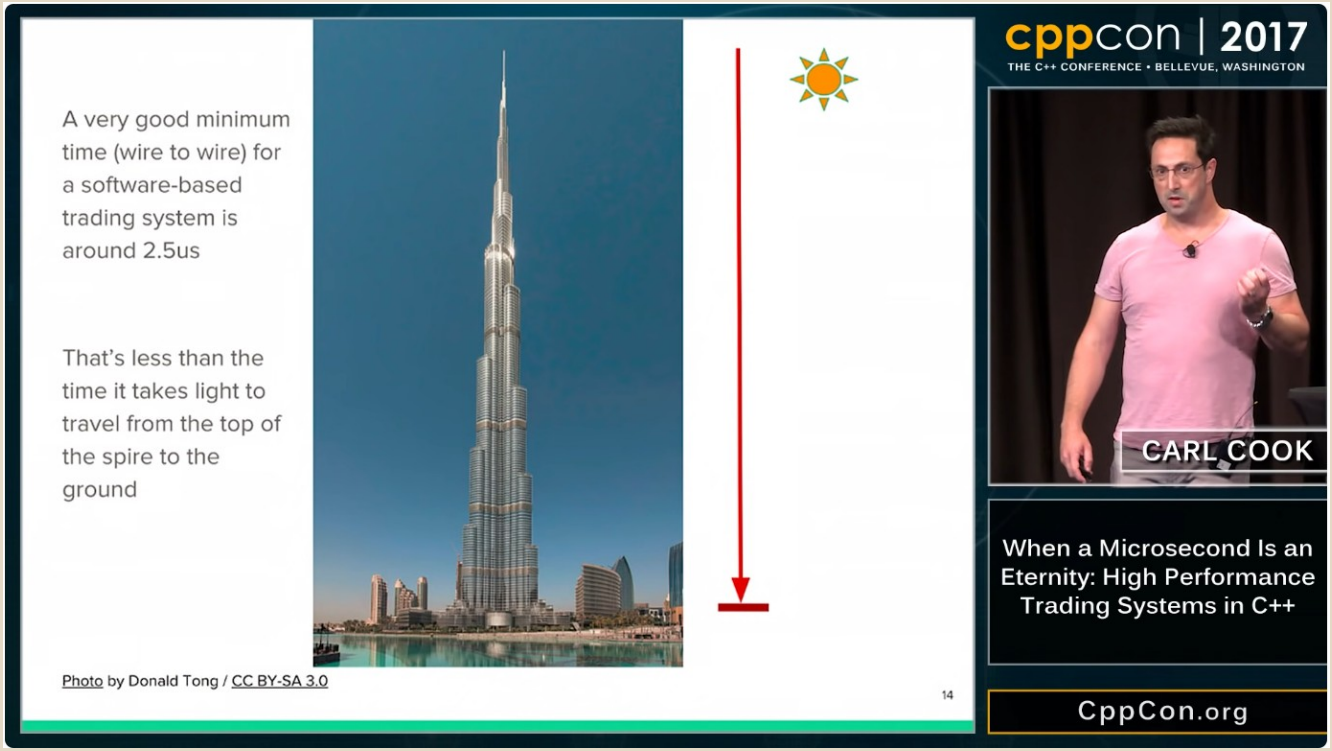
Primero de todo, hay que entender la **diferencia** entre **latencia** y **rendimiento**.

(*Latency vs Throughput*)

Tiempo Real

- **Latencia**: el tiempo desde que se solicita una tarea hasta que esta es atendida.
- **Rendimiento**: el número de tareas que se atienden en un periodo determinado de tiempo.

Tiempo Real



https://youtu.be/NH1Tta7purM

Sistemas de High Frequency Trading, ejemplo de priorizar latencia.

Tiempo Real

Los requisitos de Tiempo Real son **acciones** que si no se realizan dentro de **un límite de tiempo** cuando son necesarios, pueden llevar a **consecuencias negativas**.

Es una cuestión de **latencia**.

Tiempo Real

Según las **consecuencias de fallar** el tiempo límite, existen dos tipos de requisitos:

- Tiempo real **duro** (*hard*): peligro de muerte, destrucción de propiedad, pérdida de dinero...
- Tiempo real **blando** (*soft*): servicios degradados, molestias...

Tiempo Real

Ejemplos de tiempo real **duro**:

- Sistema ABS coches
- Emisor rayos X de radioterapia
- Marcapasos

Tiempo Real

Ejemplos de tiempo real **blando**:

- Retransmisión de vídeo cámara
- Tarjeta gráfica
- Tarjeta de sonido

Tiempo Real

Sectores con **requisitos** de Tiempo Real **Duro** que sean **muy delicados**, como medicina, automoción, aeroespacial, etc., están **estrictamente regulados**, en ocasiones hasta se necesita ***software* homologado**.

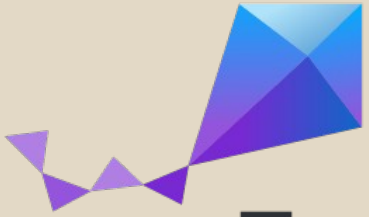
AUTOSAR

QNX

Tiempo Real

Para facilitar **modelar** estos **requisitos**, se usan Sistemas Operativos de Tiempo Real (***RTOS***).

Ofrecen un **planificador apropiativo**, y permiten definir tareas, prioridades, periodos, etc.

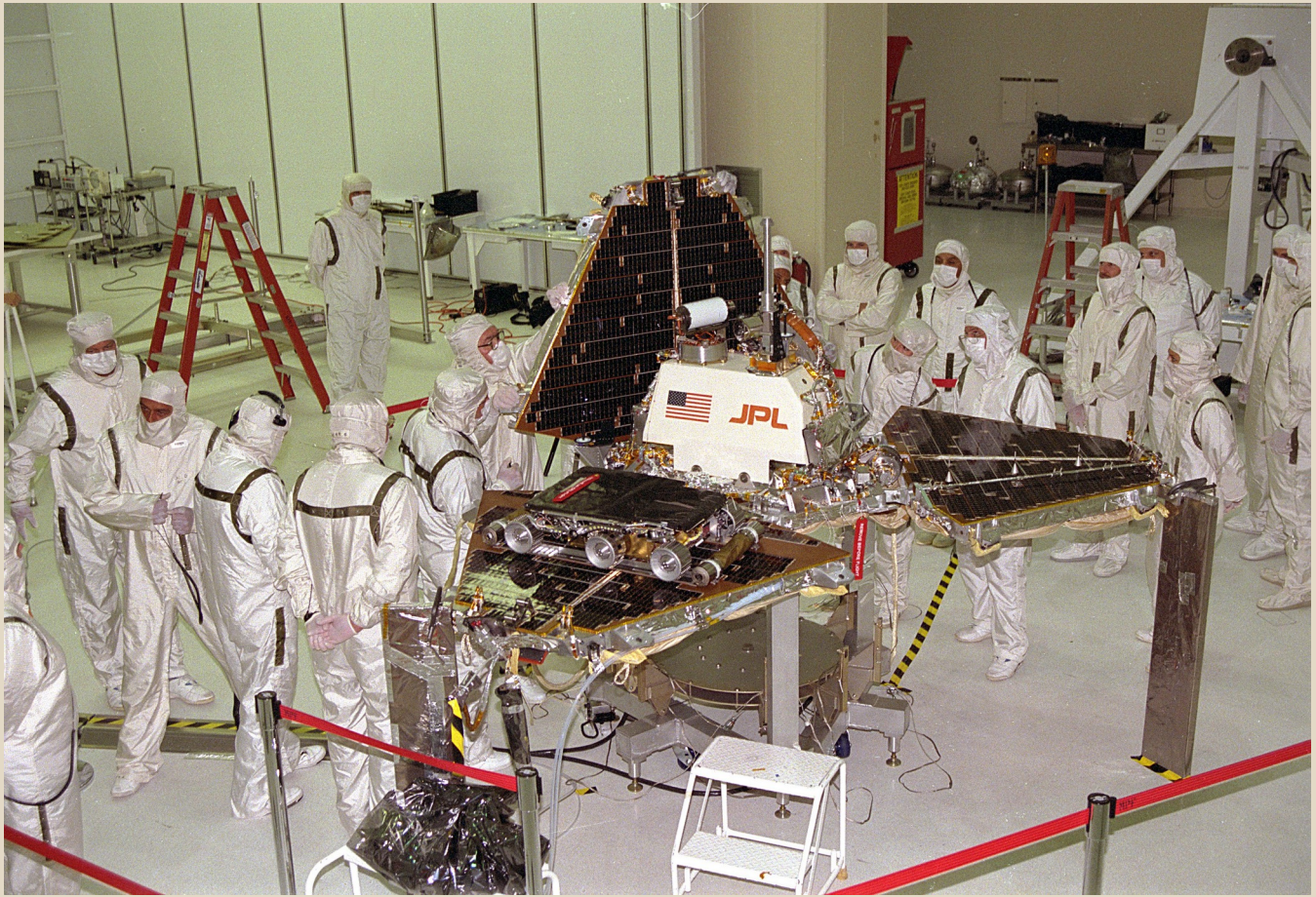


Zephyr™

NuttX RTOS

Real-Time LINUX

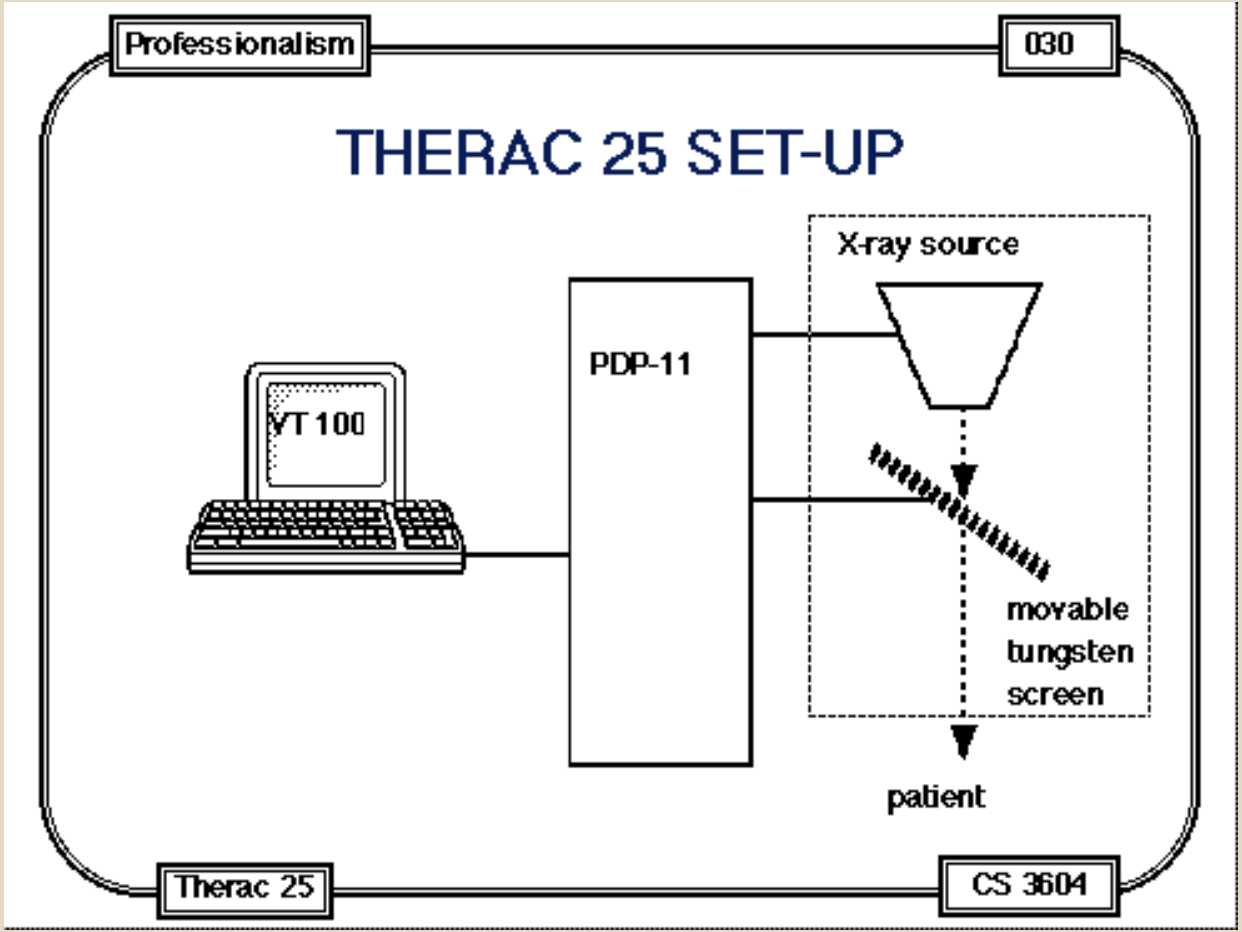
Tiempo Real



Mars Pathfinder, nave no tripulada, tuvo errores de prioridad de inversión durante la misión.

https://en.wikipedia.org/wiki/Mars_Pathfinder

Tiempo Real



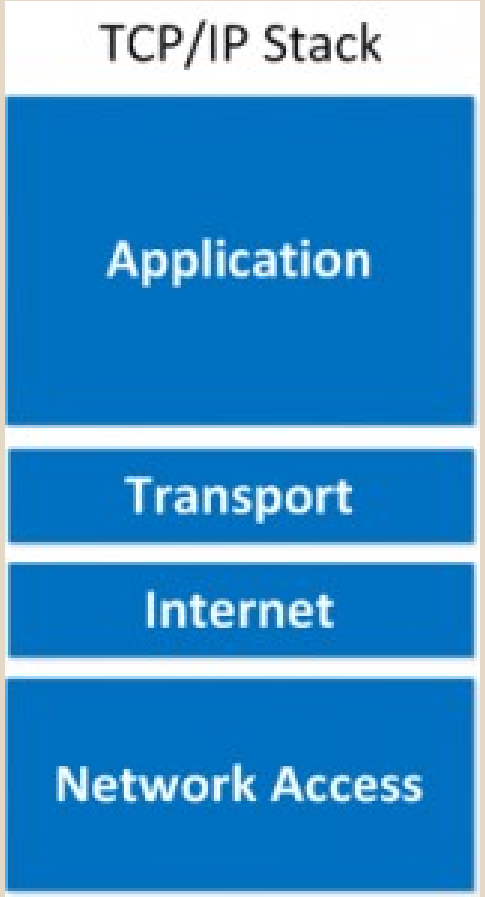
Therac-25, máquina de rayos X conocida por un error que exponía a los pacientes a cantidades de radiación letales.

https://en.wikipedia.org/wiki/Therac-25

Protocolos de **comunicación**

Protocolos de comunicación

En otros campos, no hay mucha diversidad, se suele utilizar exclusivamente **TCP/IP sobre Ethernet**.



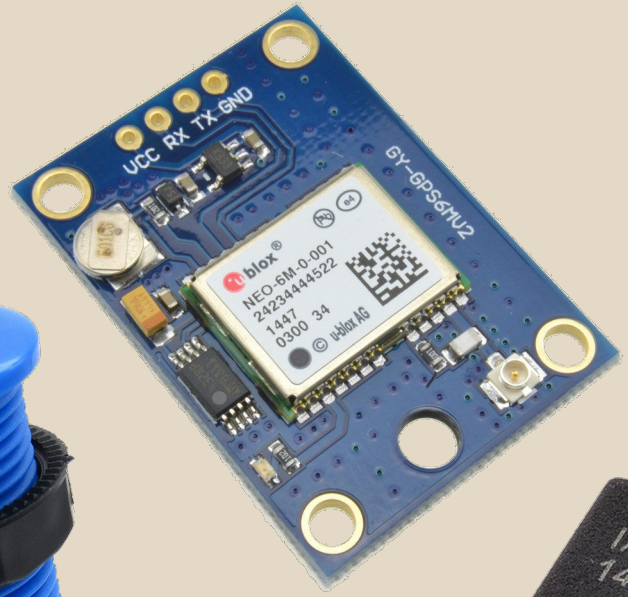
Protocolos de comunicación

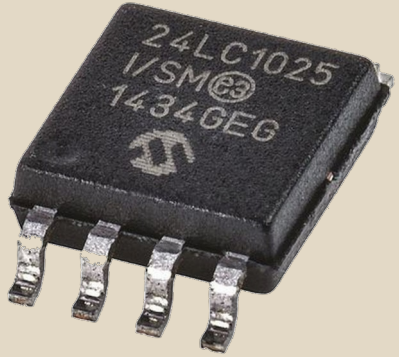
En Sistemas **Embebidos**, sin embargo, hay **muchos protocolos** con diferentes características.

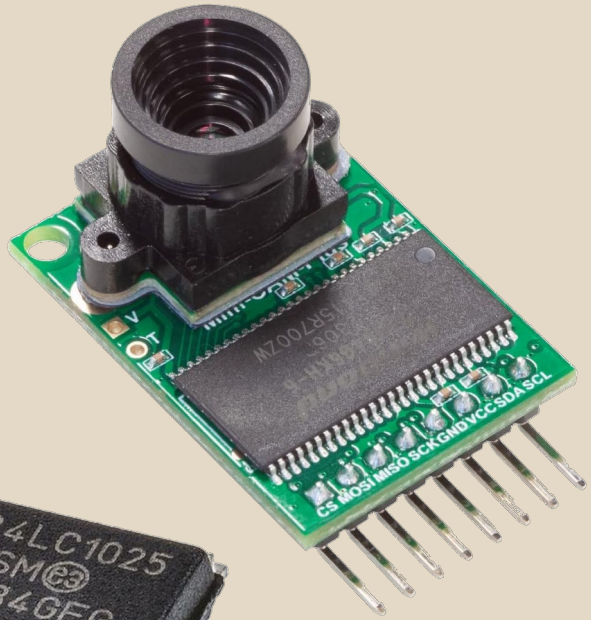
Protocolos de comunicación

Esto es porque cada sensor, pantalla, almacenamiento físico, emisor con antena, etc., usa **un protocolo especializado para su caso de uso**.

Protocolos de comunicación







Protocolos de comunicación

¿Cómo son estos protocolos?

Por simplicar, los clasifico en **cableados** o **inalámbricos**

Protocolos de comunicación

Protocolos

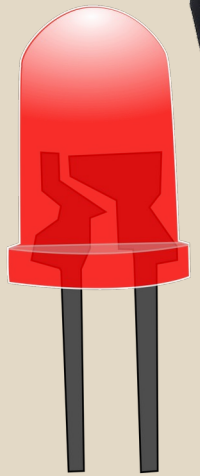
cableados

Comunicación cableada

GPIO

Instancia **más básica** de comunicación.

Es un patilla de **un *bit*** que se puede configurar como **entrada** (v.g. botón) o **salida** (v.g. LED).

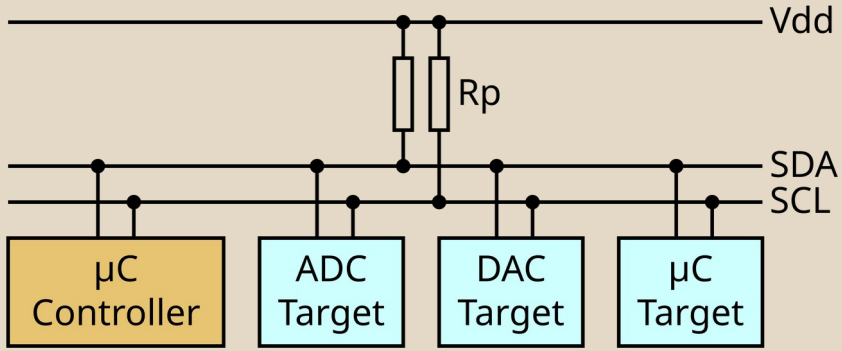


Comunicación cableada

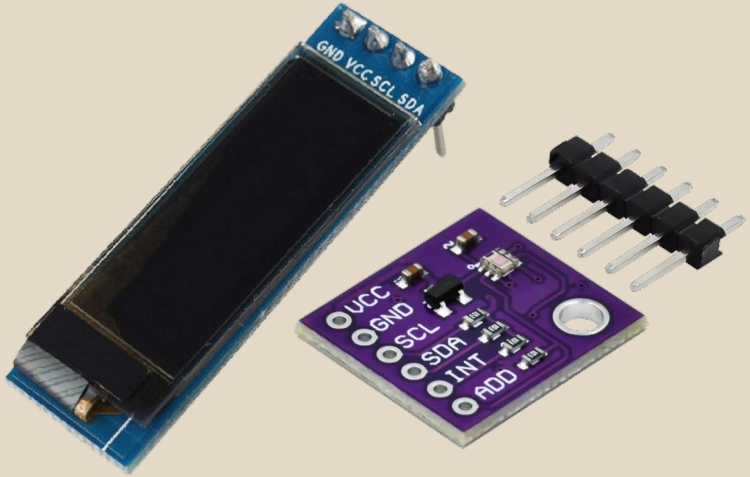
I2C

Baja velocidad de transmisión pero permite **múltiples** emisores y receptores con **únicamente** dos cables.

Los módulos **menos exigentes** suelen usar este protocolo por su **simpleza**.



Fuente: Wikipedia

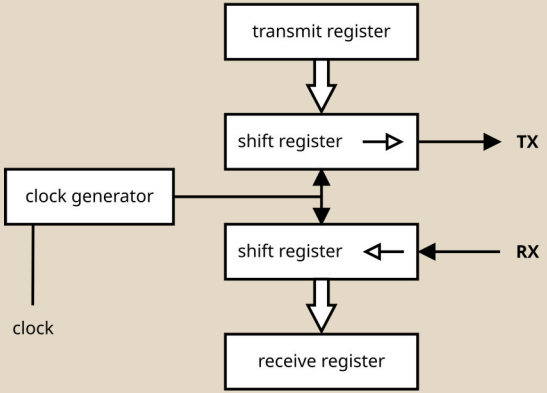


Comunicación cableada

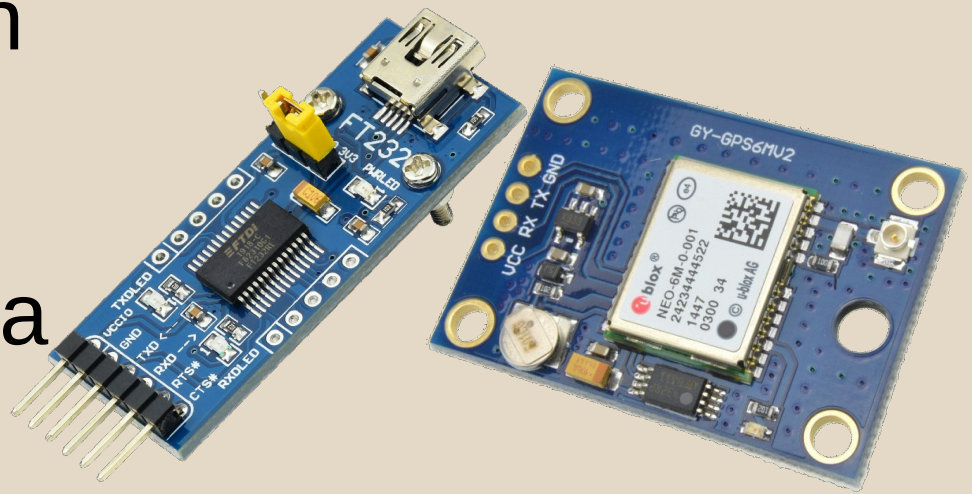
UART

Mayor velocidad de transmisión pero **comunicación de 1:1**.

Para módulos que requieren algo **más transmisión de datos**, o para el mismo procesador para mostrar una terminal.



Fuente: Wikipedia

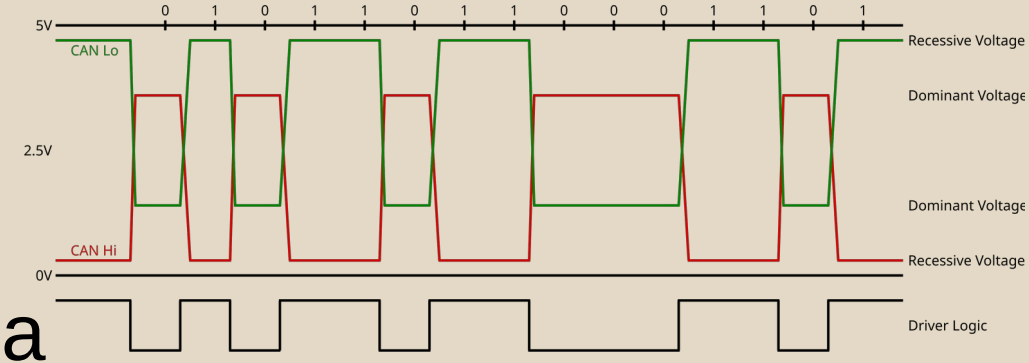


Comunicación cableada

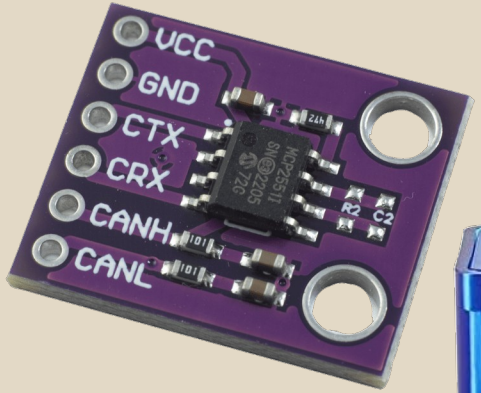
CAN

Alta transmisión de datos y **resistente a interferencias**, a costa de una circuitería más compleja, por el uso de señales complementarias.

Ideal para ambientes con **ruido electromagnético**, como automoción, aeroespacial, etc.



Fuente: Wikipedia



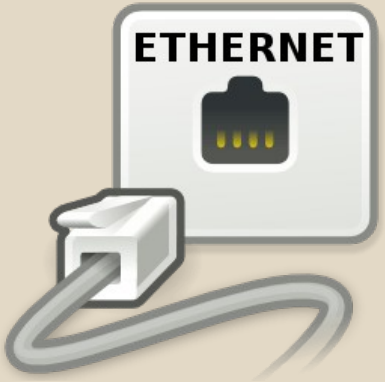
Comunicación cableada

Varios protocolos se han omitido por simplificar la presentación

SPI

lin

LOCAL INTERCONNECT NETWORK



Protocolos de comunicación

Protocolos

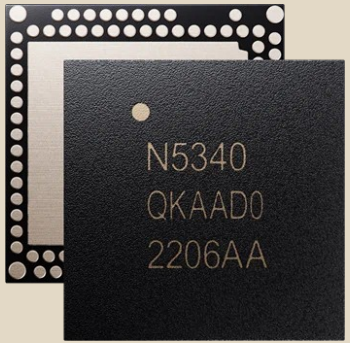
inalámbricos

Comunicación inalámbrica

Bluetooth

Baja energía y alcance medio, diseñado para conexiones inalámbricas estables entre dispositivos cercanos.

Ideal para aplicaciones móviles y de IoT.

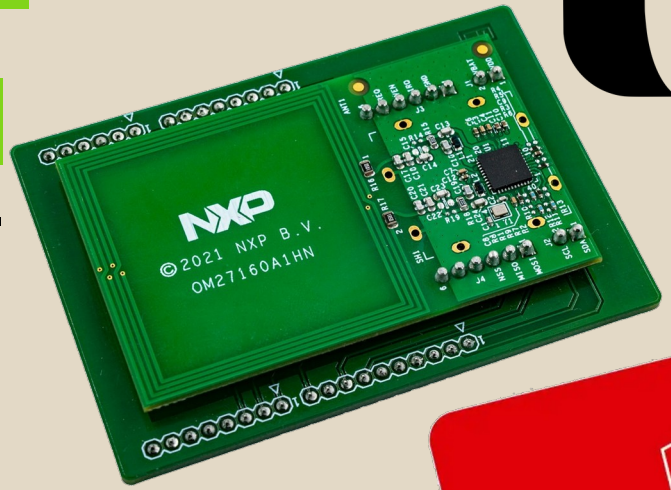


Comunicación inalámbrica

NFC

Comunicación de **corto alcance** y baja velocidad mediante inducción electromagnética, **sin** necesidad de **emparejamiento**.

Para pagos sin contacto, etiquetas inteligentes y intercambio rápido de datos en móviles y accesos seguros.



Comunicación inalámbrica

Zigbee

Red de malla de bajo consumo con **alcance medio** y alta fiabilidad, usando la banda de 2.4 GHz.

Usado domótica, sensores inalámbricos y redes IoT extensas como iluminación y control industrial.

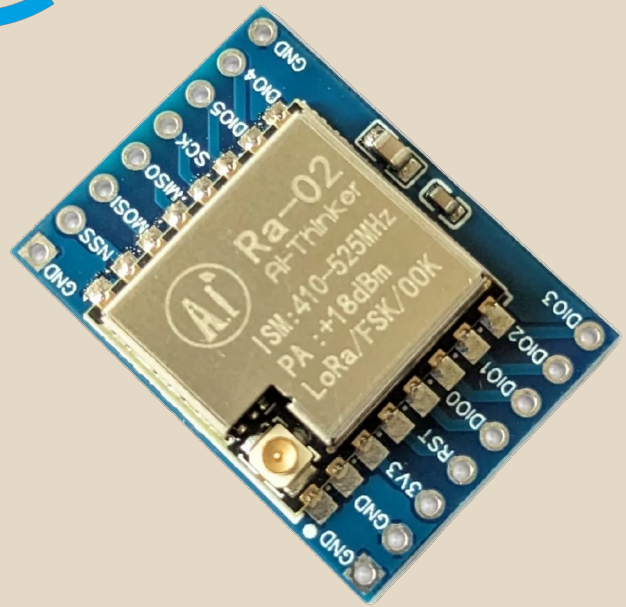
Comunicación inalámbrica

LoRa

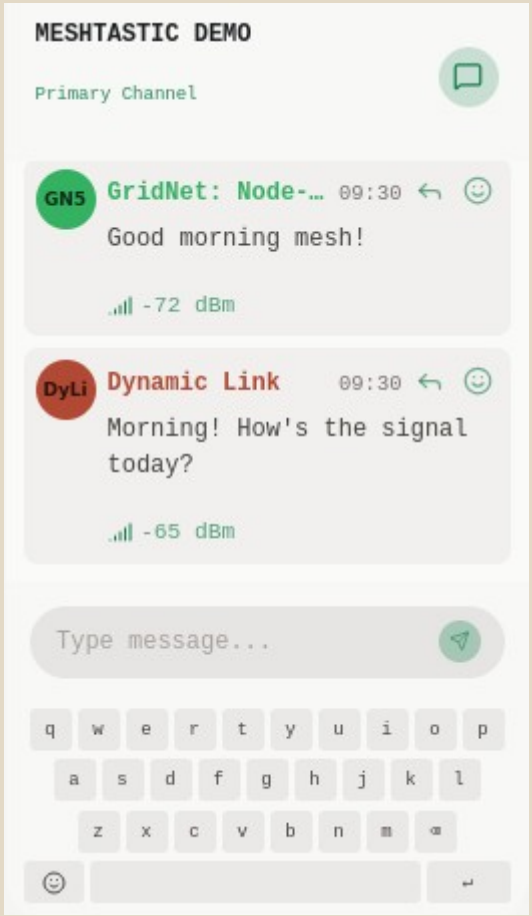
Capaz de alcanzar **largas distancias**, y **extensible mediante nodos** intermediarios. **Poco resistente a obstáculos**.

Para dispositivos que transmiten poca información en lugares remotos.

LoRaWAN®



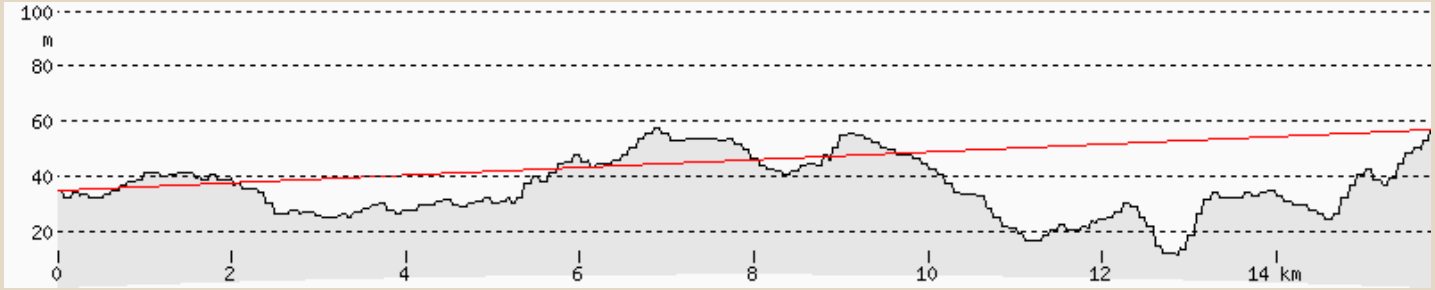
Comunicación inalámbrica



Meshtastic: chat P2P mediante LoRa

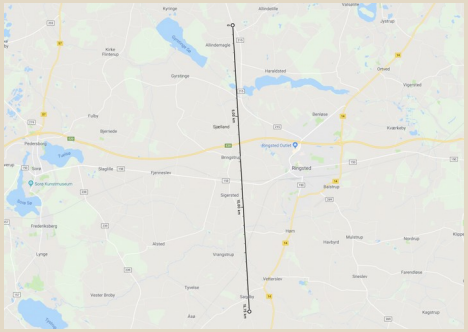
https://meshtastic.org/

Comunicación inalámbrica



SSH sobre LoRa a 15 km de distancia

https://unsigned.io/15-kilometre-ssh-link-with-rnode/



```
debug1: Local version string SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.1
debug1: Remote protocol version 2.0, remote software version OpenSSH_7.5p1 Ubuntu-10ubuntu0.1
debug1: match: OpenSSH_7.5p1 Ubuntu-10ubuntu0.1 pat OpenSSH* compat 0x04000000
debug1: Authenticating to 10.189.77.12:22 as 'markqvist'
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: curve25519-sha256@libssh.org
debug1: kex: host key algorithm: ecdsa-sha2-nistp256
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ecdsa-sha2-nistp256 SHA256:3z/BX63P/B9NAnMuUax0xlvq3DrcT+8gZqu/0HcEdx8
debug1: Host '10.189.77.12' is known and matches the ECDSA host key.
debug1: Found key in /home/markqvist/.ssh/known_hosts:576
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<ssh-ed25519,ssh-rsa,rsa-sha2-256,rsa-sha2-512,ssh-dss,ecdsa-sha2-nistp256,e
cdsa-sha2-nistp384,ecdsa-sha2-nistp521>
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/markqvist/.ssh/id_rsa
debug1: Authentications that can continue: publickey,password
debug1: Trying private key: /home/markqvist/.ssh/id_dsa
debug1: Trying private key: /home/markqvist/.ssh/id_ecdsa
debug1: Trying private key: /home/markqvist/.ssh/id_ed25519
debug1: Next authentication method: password
markqvist@10.189.77.12's password:
```

Comunicación inalámbrica

Aún más protocolos se han omitido por simplificar la presentación

Protocolos de comunicación

Hasta ahora se han mostrado protocolos al **nivel físico**, de los cuales muchos a nivel de **transporte** y **aplicación** suelen usar soluciones ***ad hoc***

Protocolos de comunicación

Para acabar esta sección, se va a mencionar un protocolo **a nivel de aplicación** ampliamente utilizado

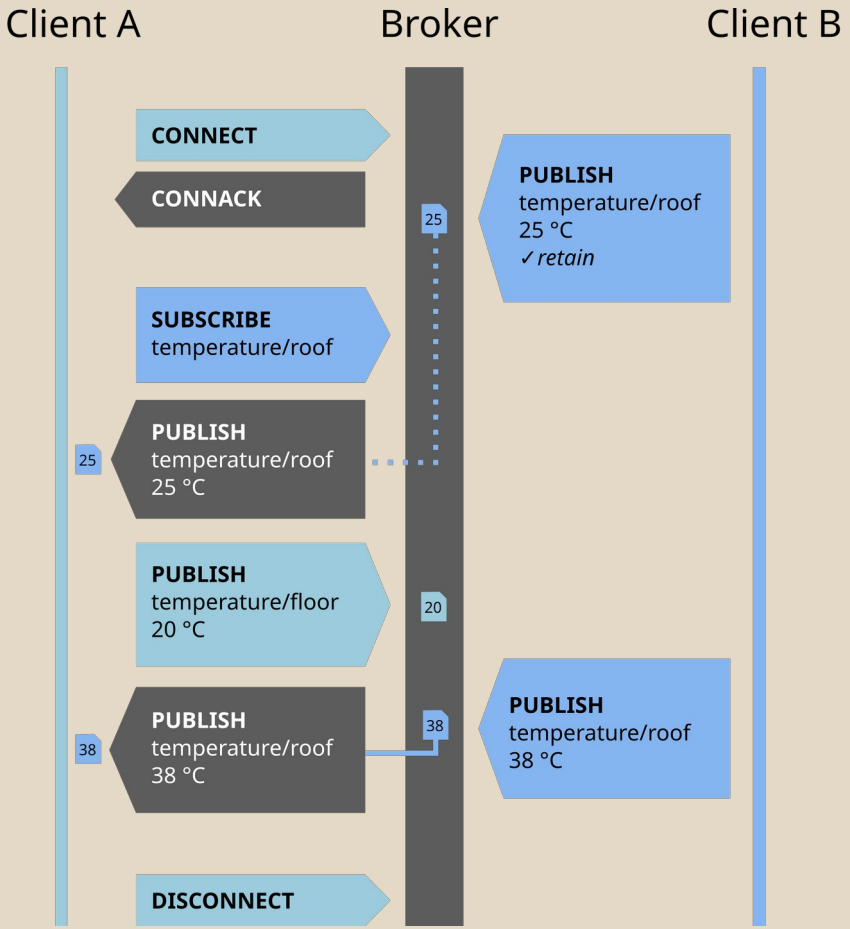
Protocolos de comunicación

MQTT

Basado en **TCP/IP**.

Dispositivos pueden **publicar** información a **tópicos**, o **subscribirse** a tópicos

El encargado de recibir o notificar a los dispositivos la información de los tópicos es el **bróker**.



Fuente: Wikipedia

Seguridad

Se **ha tratado** aquí anteriormente 🤩

Seguridad

Se aplican las **mismas prácticas** que en un sistema informático **general**…

- Disponibilidad
- Integridad
- Confidencialidad
- Autentificación
- Etcétera

Seguridad

Pero además ahora se considera **el factor físico**

Los peligros pueden afectar a la misma placa electrónica

Seguridad

Problemas de **disponibilidad**

Seguridad: disponibilidad

Los sistemas embebidos suelen tener muchas **dificultades** para **actualizar** el *firmware*, porque suelen estar **aislados**, en ubicaciones **remotas** o en general con capacidades de **comunicación** externa **limitada**.

Seguridad: disponibilidad

Por eso los ***bugs*** de programación pueden ser **especialmente problemáticos**.

Requieren un **diseño concienzudo** para poder aplicar actualizaciones.

Seguridad

Soluciones de **disponibilidad**

Seguridad: disponibilidad

Testeo intensivo de las capacidades del sistema, tanto a nivel ***software*** (pruebas unitarias, *fuzzing*, *sanitizers*, etc.) como a nivel ***hardware*** (pruebas de sistema e integración).

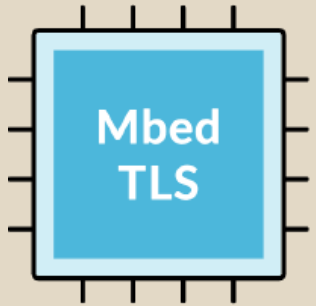
AFL++

Seguridad: disponibilidad

Usar un ***bootloader*** con capacidades de **actualización** segura **remota** y disponer de alguna **interfaz** para recibir dicha actualización.

Seguridad: disponibilidad

Uso de **herramientas** que hayan sido puestas a **prueba en industria** o desarrollar nuevo código con **lenguajes** más **seguros**, como Rust.



wolfSSL

Seguridad: disponibilidad

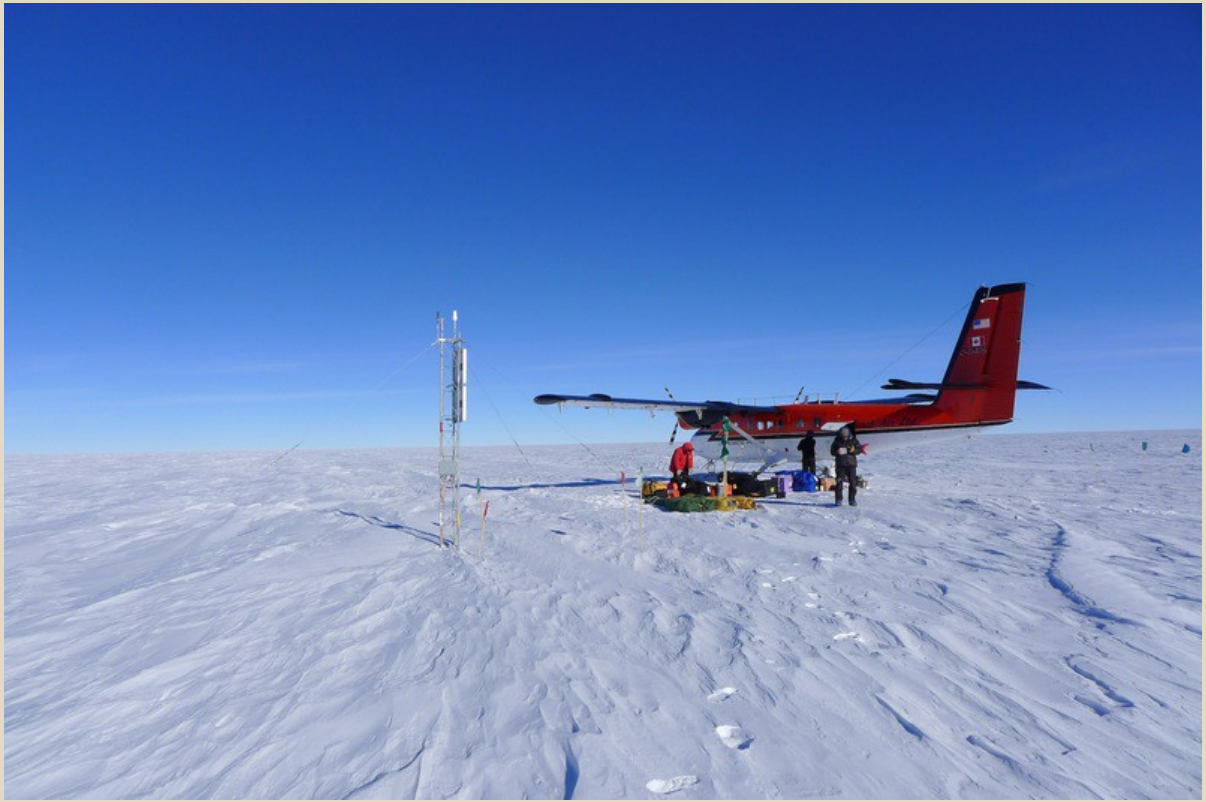
Diseñar sistemas con **redundancia**: copias de ficheros esparcidas por el almacenamiento, sensores duplicados, etc., en general **mecanismos** de **salvaguarda**.

Aplicar la mentalidad de la **Ley de Murphy** a la disponibilidad de cada **recurso crítico**.

Seguridad

Ejemplos de **disponibilidad**

Seguridad: disponibilidad



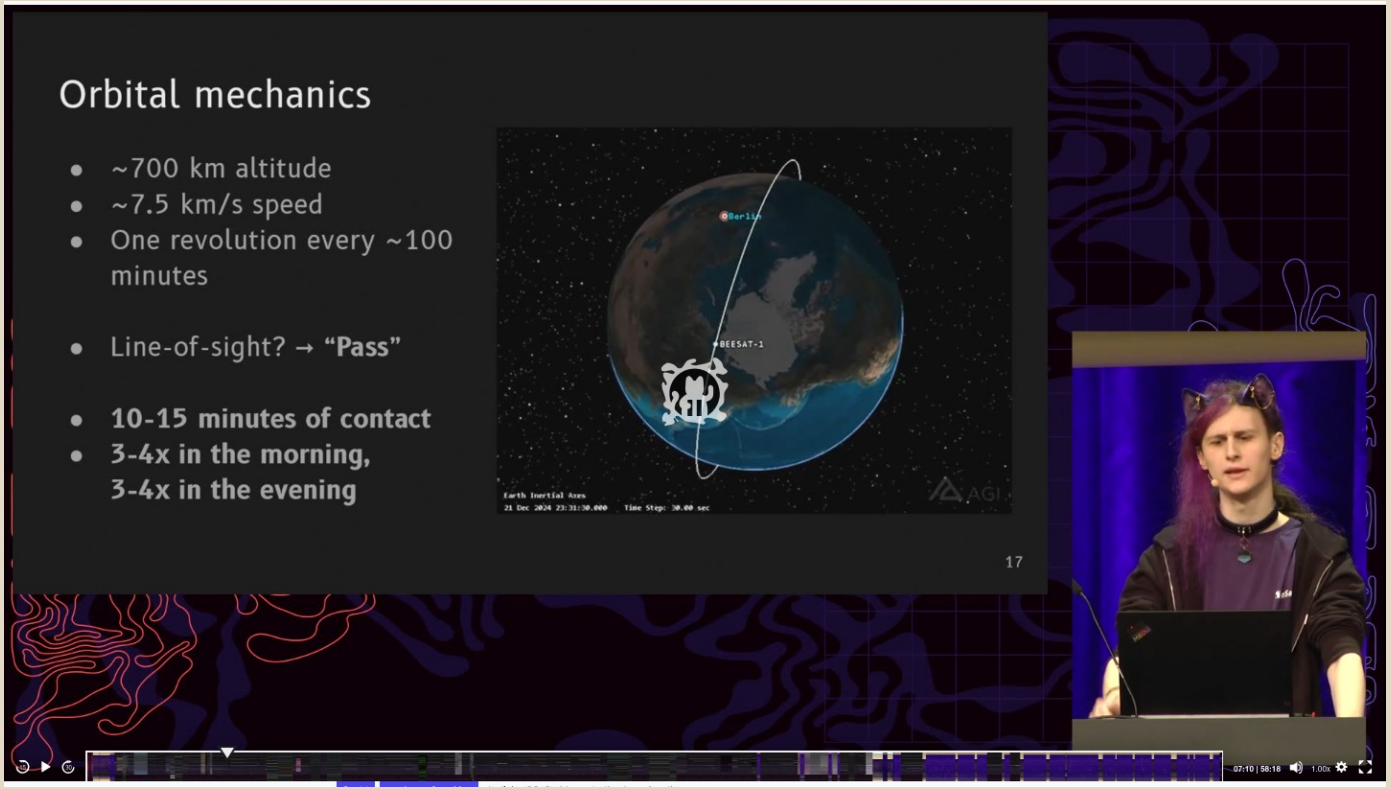
doi.org/10.5194/gi-10-161-2021

Sensores para investigación del clima espacial (la interacción entre el Sol y la Tierra) situados en la Antártida.

Totalmente innaccesibles físicamente excepto en ocasiones del verano.

Hubo que ir presencialmente para actualizar el firmware por un bug en la hora obtenida por GPS...

Seguridad: disponibilidad



Hacking yourself a satellite - recovering BEESAT-1

¿Cómo arreglar un fallo software en un satélite orbitando a 700 km de altitud?

https://media.ccc.de/v/38c3-hacking-yourself-a-satellite-recovering-beesat-1

Seguridad

Problemas de **confidencialidad**

Seguridad: confidencialidad

Tanto la **memoria volátil** (RAM) como **no volátil** (flash, EEPROM) se pueden **leer** o incluso **escribir sin autorización** accediendo **físicamente** a la placa electrónica.

Seguridad

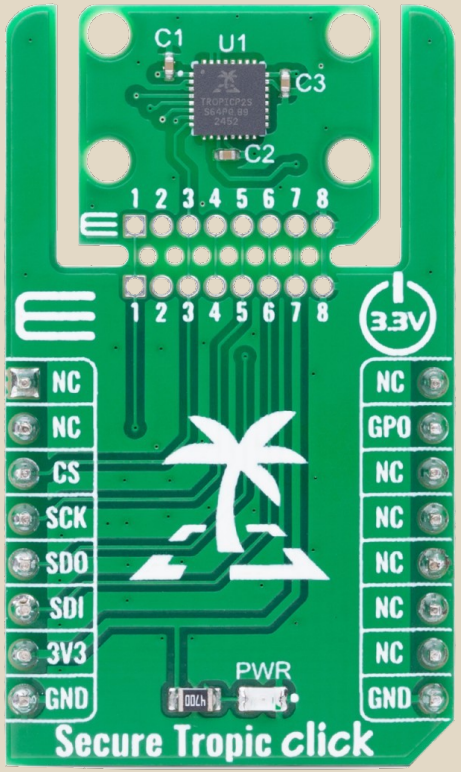
Soluciones de **confidencialidad**

Seguridad: confidencialidad

- **No** llevar información **muy confidencial** a RAM.
- **Encriptar** sistemas de ficheros.

Seguridad: confidencialidad

- Guardar la **información** más **crítica**, como claves criptográficas, en **Elementos Seguros**.
- **No exponer pines de depuración** en producto final, como UART con *logs* o con terminal, o el JTAG.



Seguridad

Ejemplos de **confidencialidad**

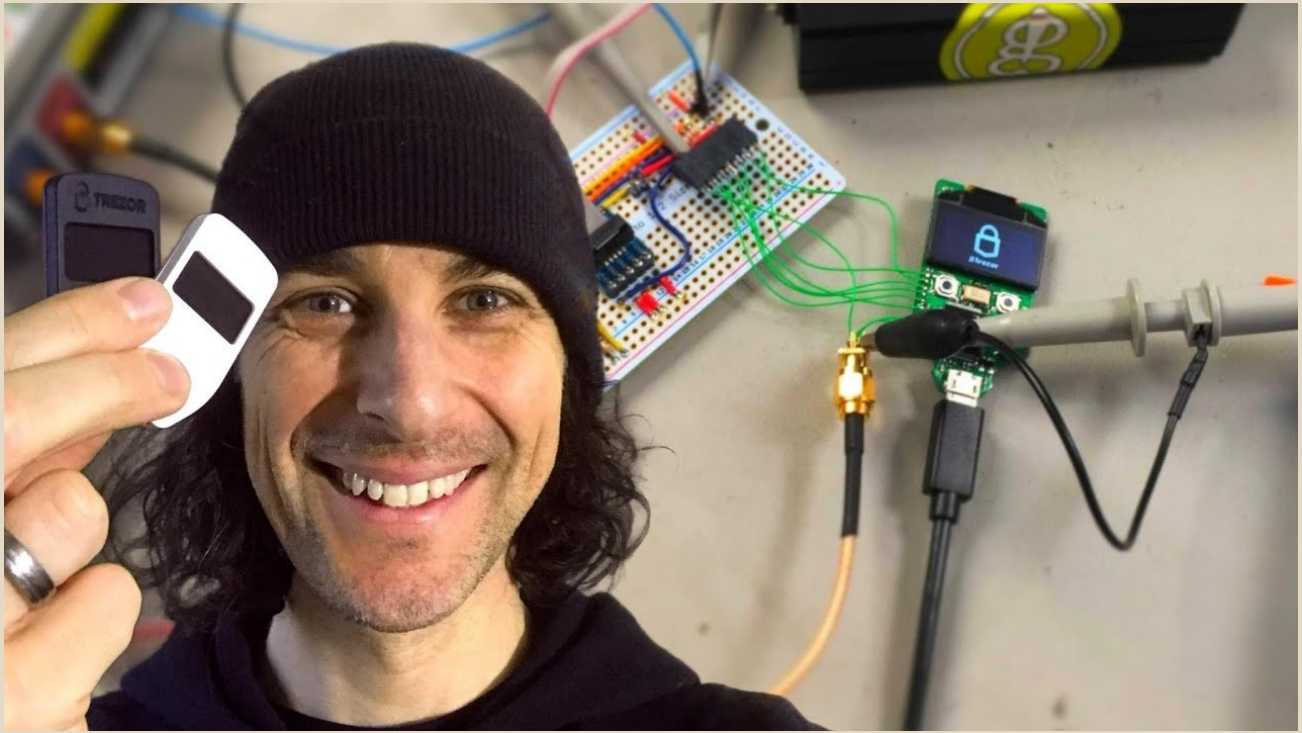
Seguridad: confidencialidad

Intromisión a configuración interna de una cámara.

Motivo: pines expuestos para terminal por UART y *bootloader* sin contraseña.

https://www.synacktiv.com/publications/i-hack-u-boot

Seguridad: confidencialidad



Recuperación de 2 M$ en criptomonedas de una billetera hardware Trezor aprovechando error de *firmware* desactualizado donde carga información sensible en RAM.

https://youtu.be/dT9y-KQbqi4

Seguridad: confidencialidad

https://youtu.be/clESYc9BDvc

Acceso a sistema interno de router TP-Link, mediante pines UART con salida de terminal y extracción de contenido de memoria *flash.*

Seguridad

Problemas de **integridad**

Seguridad: integridad

Hay riesgo de **grabar variantes** del *firmware* con funcionalidad diferente **a la intencionada**.

O directamente **inutilizar** el dispositivo por **corrupción** durante una actualización o por el ambiente.

Seguridad

Soluciones de **integridad**

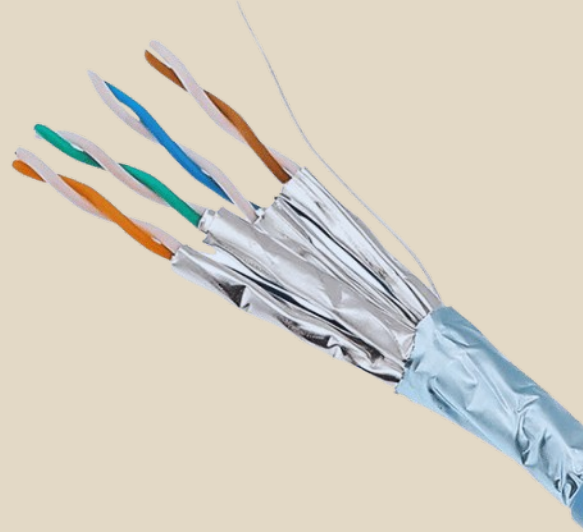
Seguridad: integridad

Usar un **procesador con** capacidades de **arranque seguro**, para poder ejecutar ***firmware*** solo si está **firmado** criptográficamente con nuestra llave.

Seguridad: integridad

Si algún **canal** de comunicación es **crítico**, usar **protocolos resistentes** a interferencias, como CAN, o **proteger** el **medio**, como con cable Ethernet STP (cable acorazado).

CAN



Seguridad: integridad

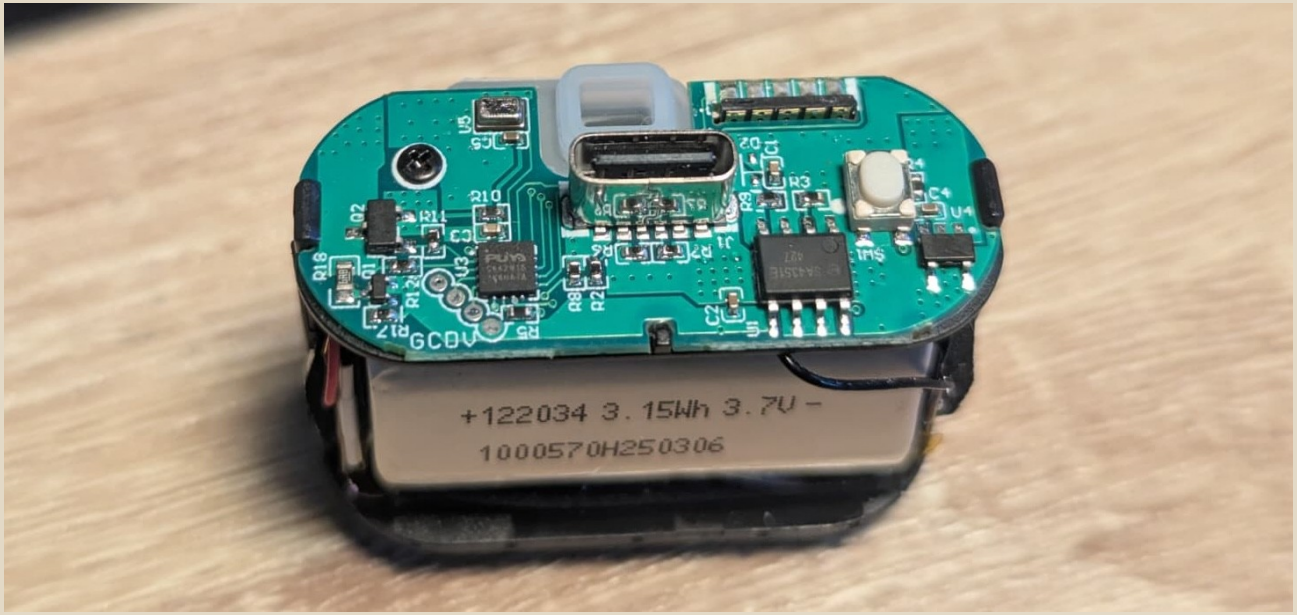
Usar funciones de **encriptación** y ***checksum*** **sin vulnerabilidades**:

- *Checksum*: SHA256, BLAKE3
- Contraseñas: Argon2
- Encriptación simétrica: AES-GCM, ChaCha20-Poly1305

Seguridad

Ejemplos de **integridad**

Seguridad: integridad



Servidor web desplegado con un vapeador reutilizado.

Ejemplo de uso no intencionado de la placa.

https://bogdanthegeek.github.io/blog/projects/vapeserver/

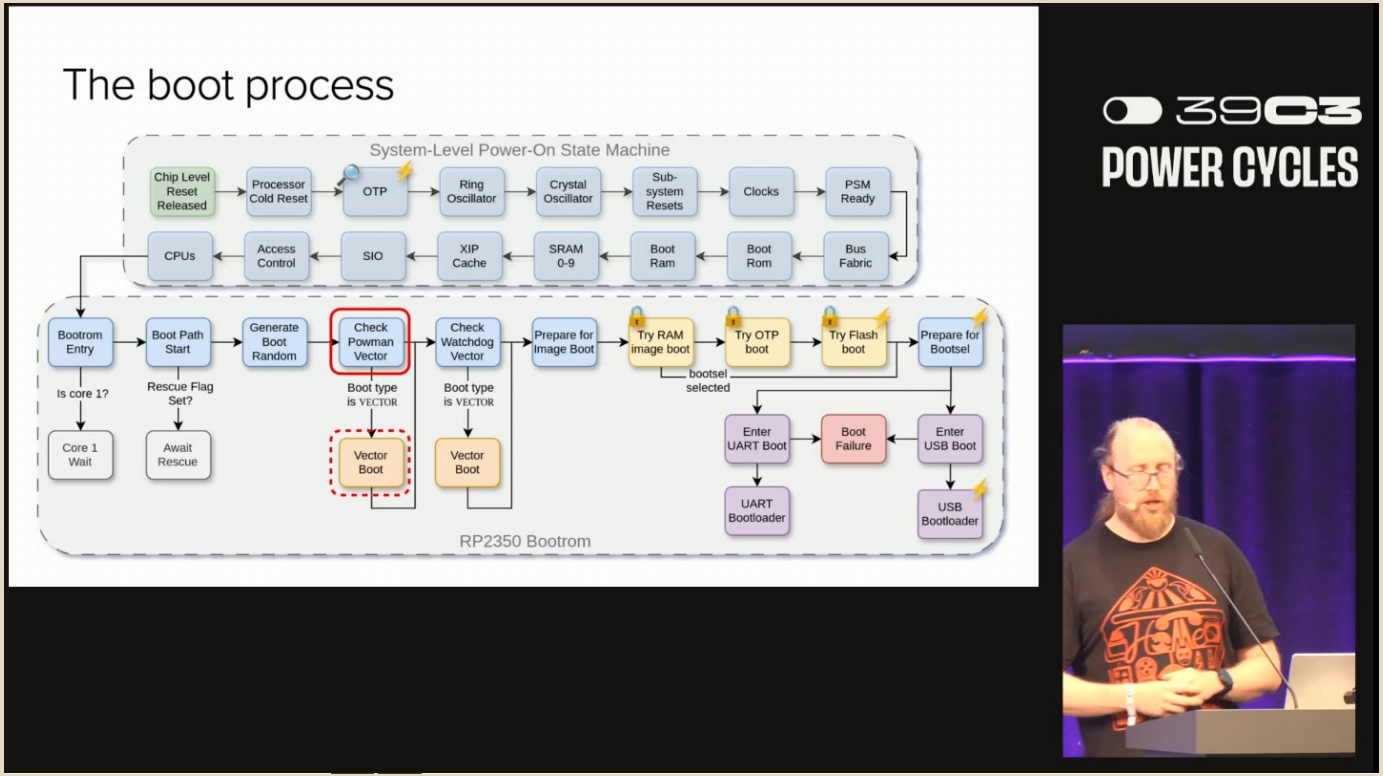
Seguridad: autentificación

Suplantación de mensaje de apertura de coches a UCE mediante intervención física de bus CAN, ya que no hay verificación del remitente.

Fotograma: robo de Toyota RAV4 2021 mediante este método.

https://kentindell.github.io/2023/04/03/can-injection/

Seguridad: integridad

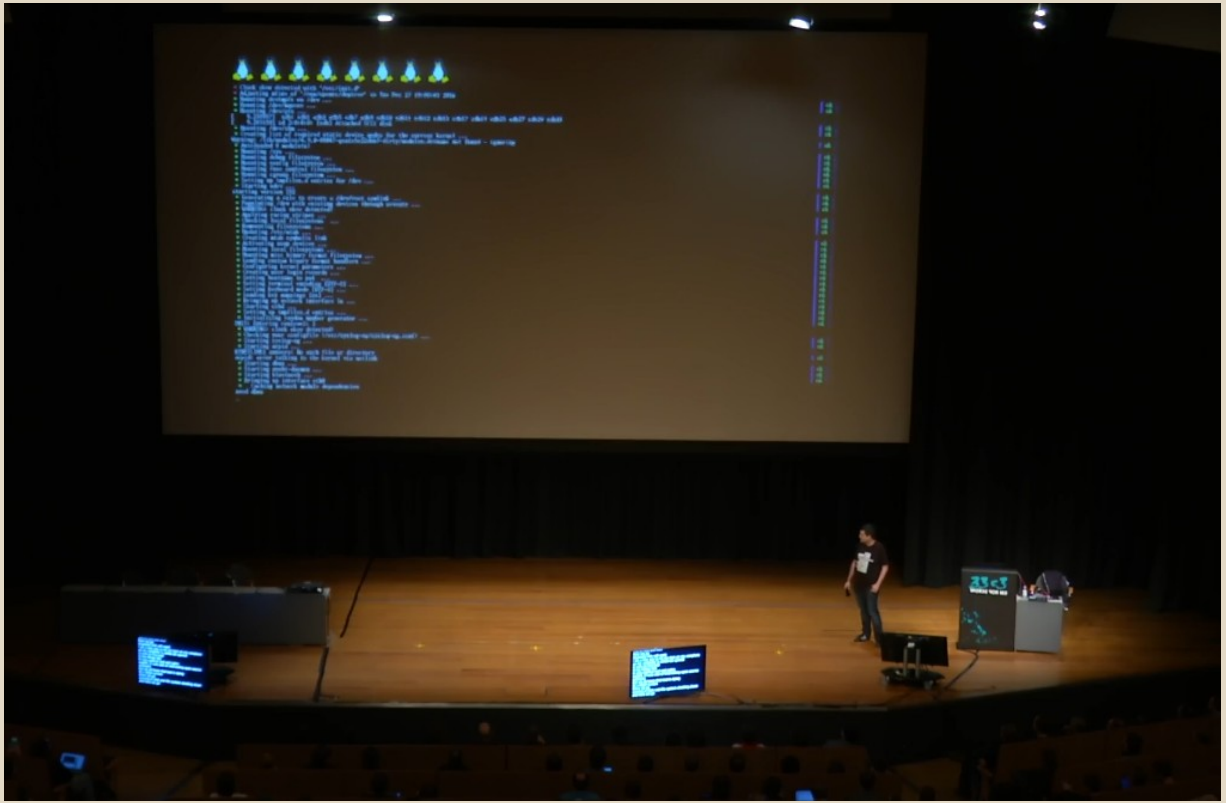


Bypassing RP2350's Secure Boot

Este microcontrolador ofrece mecanismos para subir *firmware* firmado criptográficamente.

https://media.ccc.de/v/39c3-of-boot-vectors-and-double-glitches-bypassing-rp2350-s-secure-boot

Seguridad: integridad



Pirateo de PS4 para correr Linux desde la misma consola.

La presentación la da desde PS4 ya modificada.

https://media.ccc.de/v/33c3-7946-console_hacking_2016

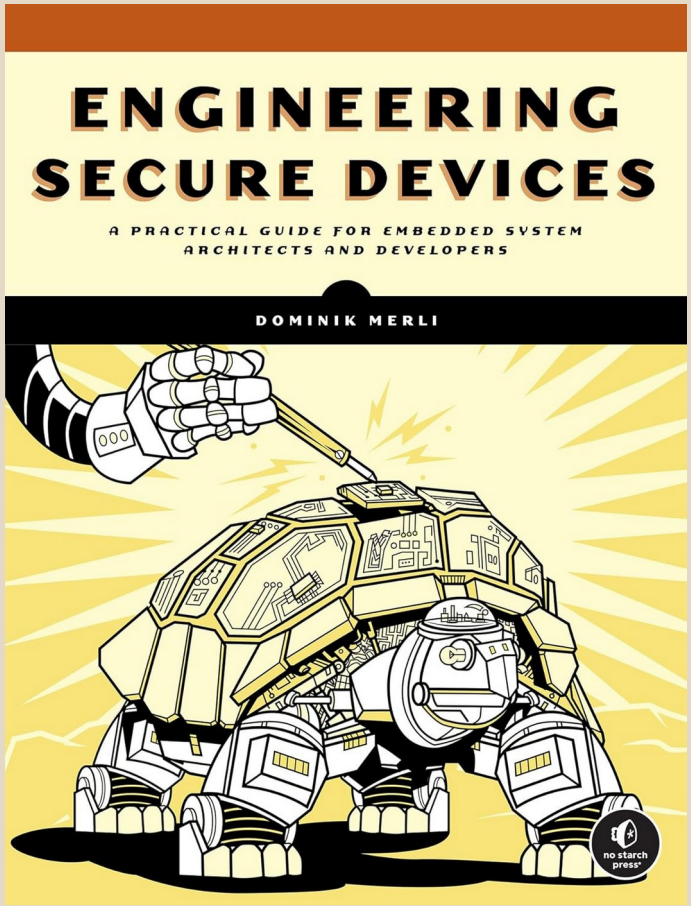
Seguridad: integridad

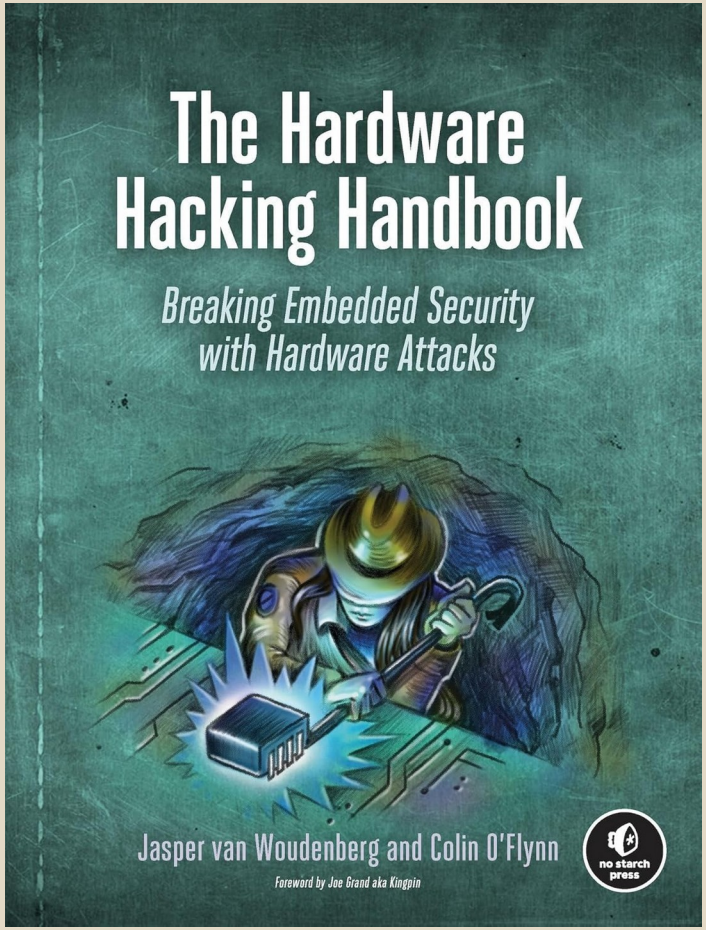
Attacking the Nintendo 3DS Boot ROMs

https://arxiv.org/abs/1802.00359

Seguridad

Bibliografía:

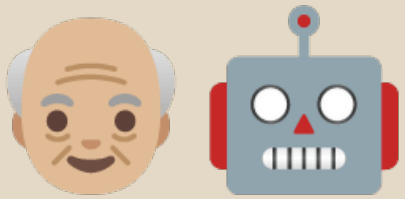




Inteligencia Artificial

Inteligencia Artificial

Alias: *Loh pogramah inteligenteh*



Inteligencia Artificial

Dos escenarios según **dónde se ejecuta** el modelo de IA:

- En un ordenador remoto.
- En el mismo dispositivo.

Inteligencia Artificial

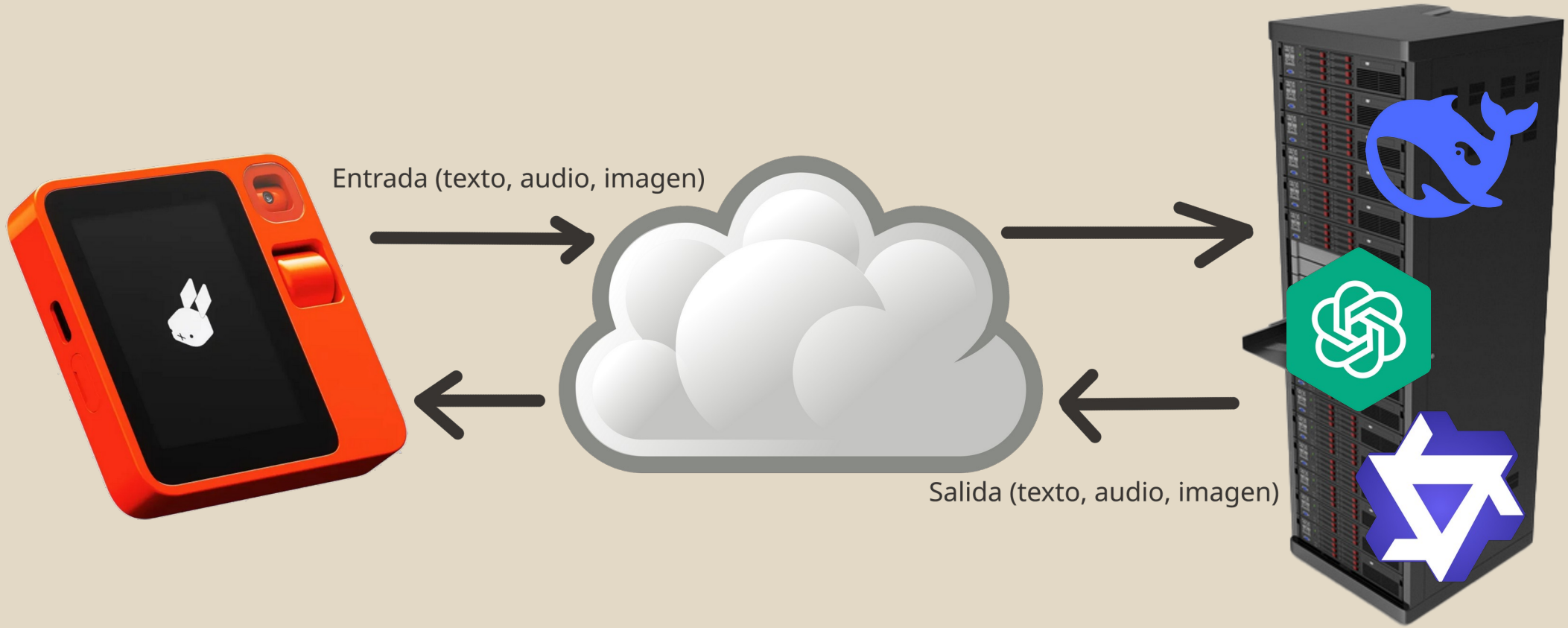
En un ordenador **remoto**

Inteligencia Artificial remota

El dispositivo **comunica a un equipo remoto** las entradas que el modelo requiere para trabajar (texto, imagen, audio…).

El equipo remoto **devuelve los resultados** tras ejecutar el modelo.

Inteligencia Artificial remota



Inteligencia Artificial remota

Auriculares *con IA*: manda audio y órdenes a servidor remoto.

En el artículo: robando *token* de OpenAI y de paso instala Doom.

https://blog.mgdproductions.com/ikko-activebuds/

Inteligencia Artificial remota

MCP en dispositivos IoT

Golioth TinyMCP

https://blog.golioth.io/tinymcp-unlocking-the-physical-world-for-llms-with-mcp-and-microcontrollers/

https://youtube.com/shorts/GJxzDtrK_vQ

Inteligencia Artificial remota

Hay clientes compatibles con OpenClaw para MPU y MCU

Ejemplo: Mimiclaw para ESP32-S3

https://www.cnx-software.com/2026/02/13/mimiclaw-is-an-openclaw-like-ai-assistant-for-esp32-s3-boards/

Inteligencia Artificial

Localmente en el dispositivo

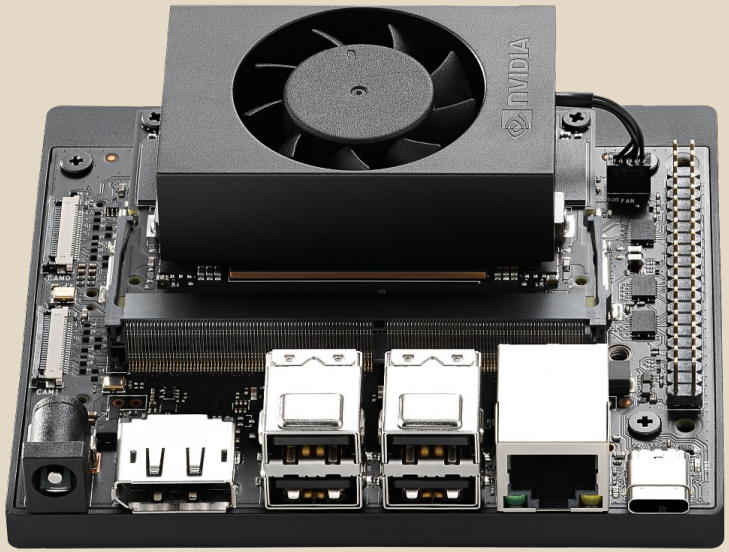
Inteligencia Artificial local

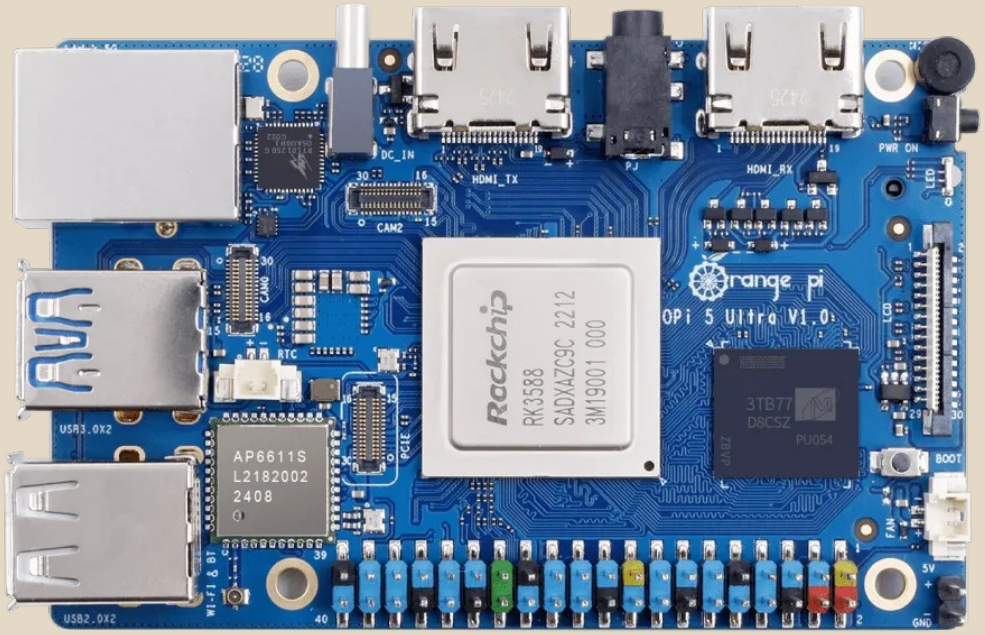
Correr localmente modelos requiere **adaptarlos** a *hardware* con capacidades limitadas:

- **Cuantización** de los pesos a enteros (8, 16 ó 32 bits) por eficiencia en la CPU.
- Escoger modelos de tamaño reducido para poder **caber en la escasa RAM**.
- Limitado a **inferencia.**
- **Requieren** memoria montón (i.e. no funcionan normalmente en MCU).

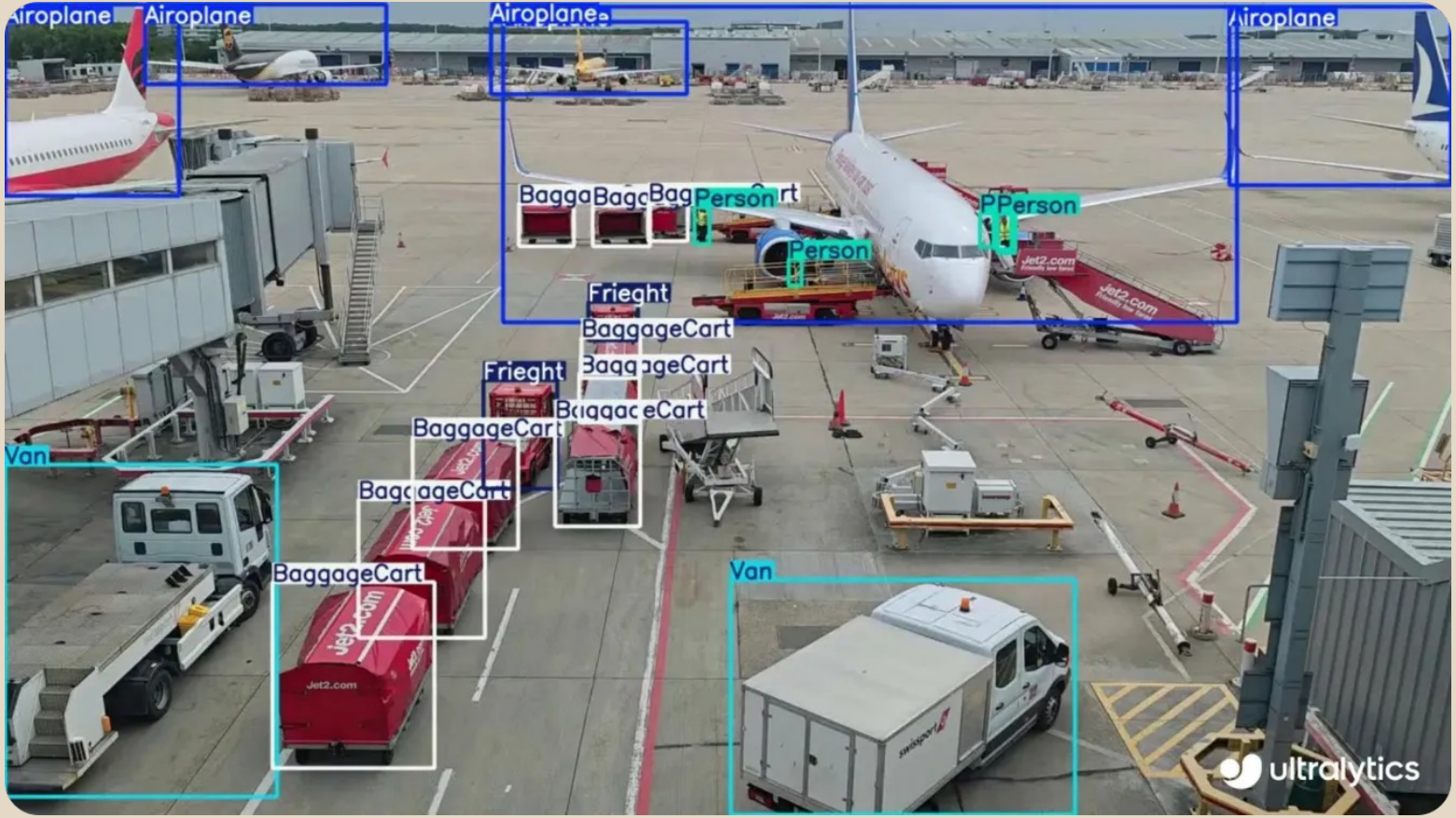
Inteligencia Artificial local

Se suelen ejecutar en CPU, pero empiezan a haber **aceleradores** para Sistemas Embebidos:





Inteligencia Artificial local



Ejecutando YOLO11 localmente en un Rockchip

https://www.ultralytics.com/blog/deploy-ultralytics-yolo11-on-rockship-for-efficient-edge-ai

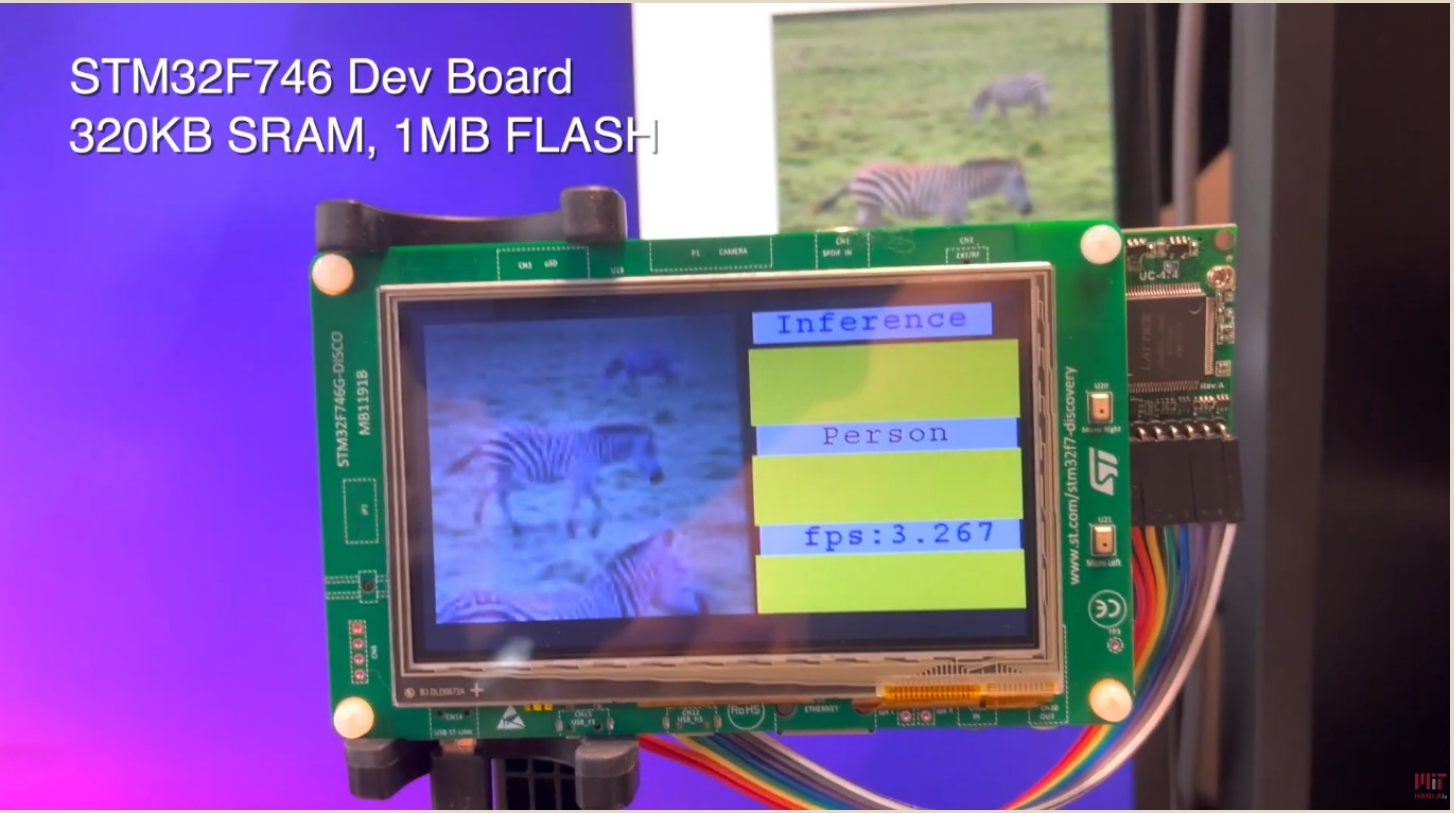
Inteligencia Artificial local

Conducción autónoma, tiempo real duro.

Ejemplo: openpilot de comma.ai

https://blog.comma.ai/gettingstarted/

Inteligencia Artificial local



On-Device Training Under 256KB Memory

https://arxiv.org/abs/2206.15472

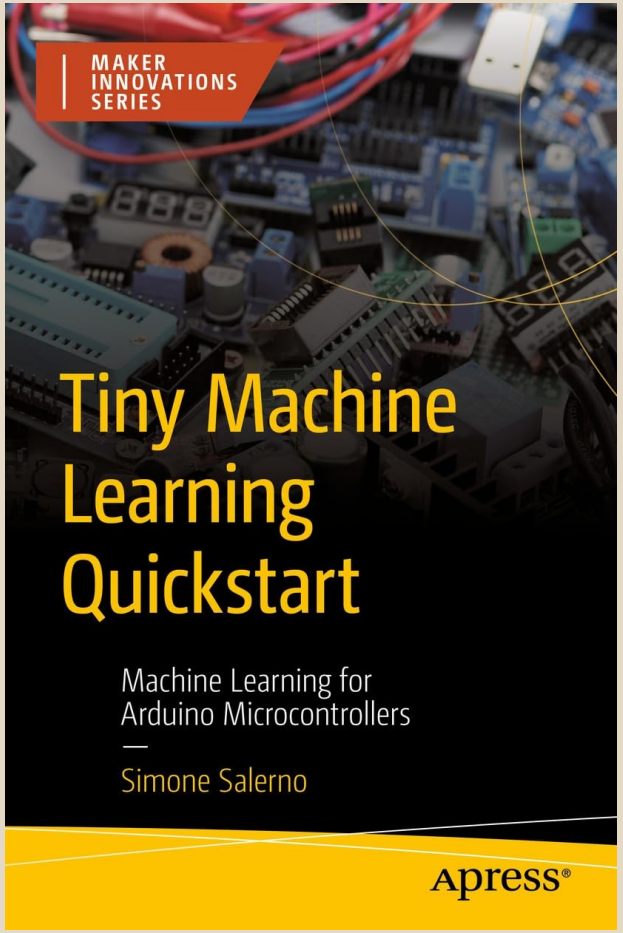
Inteligencia Artificial

Bibliografía:

EDGEAI FOUNDATION

CONNECTING AI TO THE REAL WORLD

https://www.edgeaifoundation.org/learn

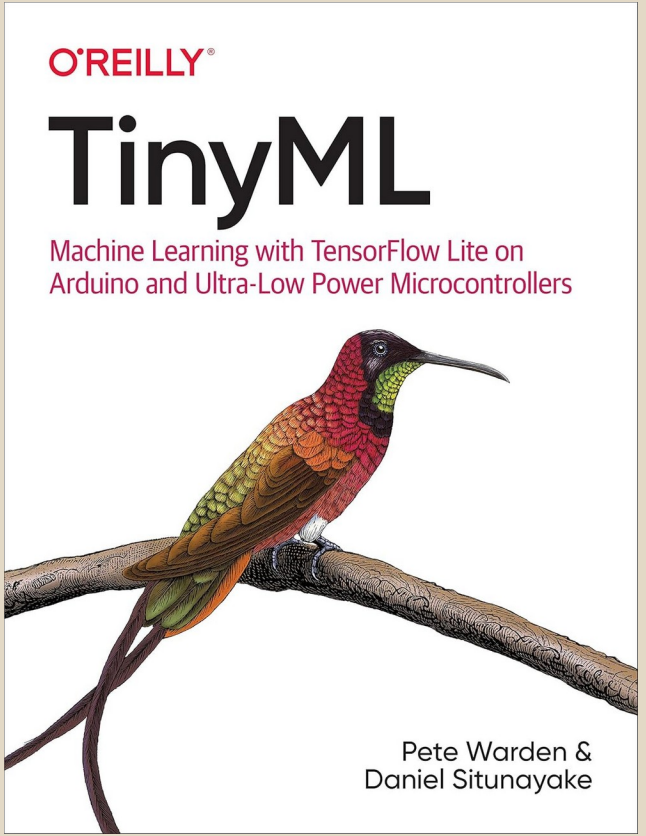


Inteligencia Artificial

Bibliografía:

TINY ML TINY MACHINE LEARNING AT MIT

https://tinyml.mit.edu/



Inteligencia Artificial

Bibliografía:

Efficient Neural Network Deployment for Microcontroller
https://arxiv.org/abs/2007.01348

Algunos **temas** se quedan **en el tintero**…

Se van a mencionar **brevemente**

Diseño PCB

El **diseño** de la placa del **circuito impreso** que contiene todos los componentes electrónicos también se debe realizar.

Con un **EDA** podemos hacer el dibujo de los esquemáticos, las conexiones y la disposición física en la placa, entre otras cosas.

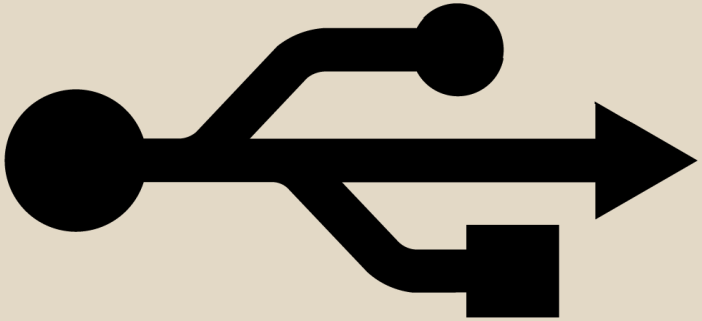
Diseño envoltura

La placa del circuito impreso casi siempre está **integrada dentro** de una carcasa o un sistema mecánico, y esta envoltura debe tener las características físicas para sostener y proteger la placa.

Se usan programas de **diseño 3D**.

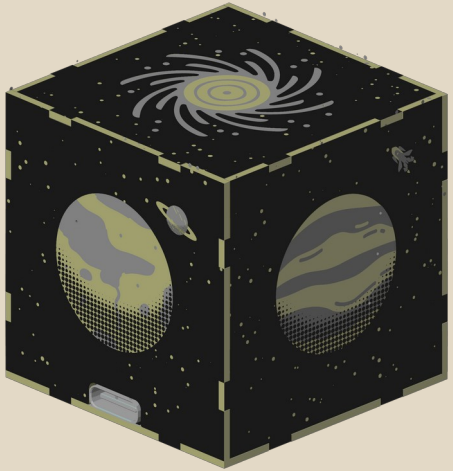
Periféricos USB

Si nuestro producto se conecta **a través de USB**, podemos utilizar las interfaces estándares proporcioanadas por los Sistemas Operativos para poder **interactuar con el equipo**.



Estándares con **Human Interface Device** (HID) especifican las opciones posibles.

Aunque si nuestro producto **no coincide con las capacidades predeterminadas**, es necesario programar un ***driver***.

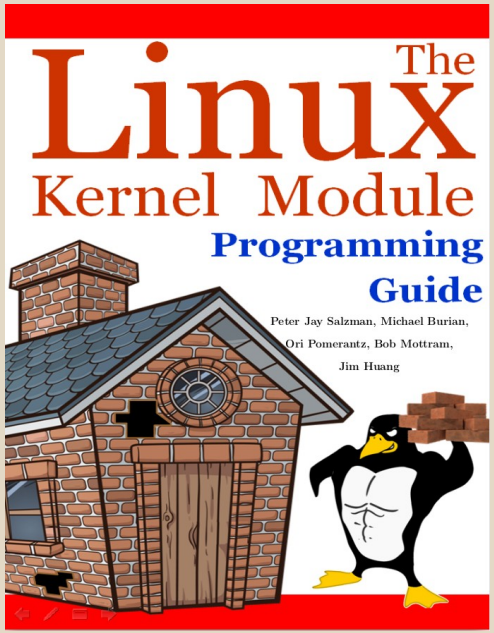


https://cubetouch.noteolvid.es/

Programación *drivers*

Un **Sistema Operativo** suele permitir **expandir las capacidades internas** mediante la programación de ***drivers*** (llamados módulos en Linux).

Por ejemplo, para implementar un planificador de procesos, un sistema de ficheros, un nuevo tipo de *socket*, algún tipo de periférico nuevo, monitorización de rendimiento, etc.

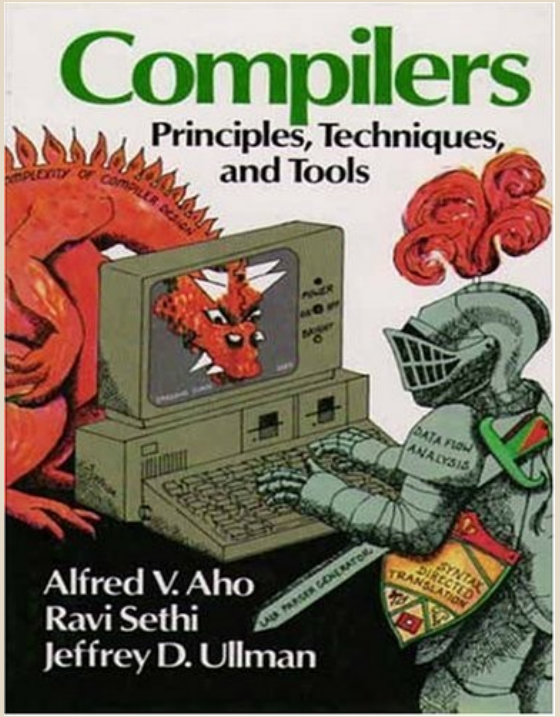


Bootloaders

El **cargador de arranque** (*bootloader*) son los programas que se ejecutan **previamente al programa principal** (en *baremetal* o *hosted*) para inicializar el *hardware*, gestionar actualizaciones del *firmware*, etc.

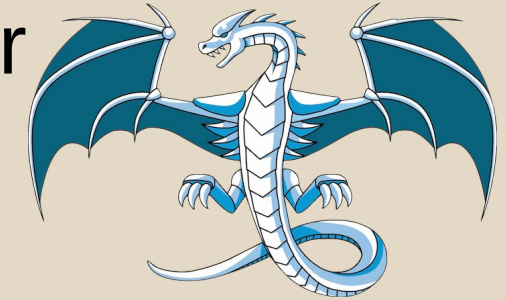
U-Boot

Compiladores



Si se está diseñando un **procesador nuevo**, queremos que **los lenguajes de programación puedan compilar** a nuestro sistema.

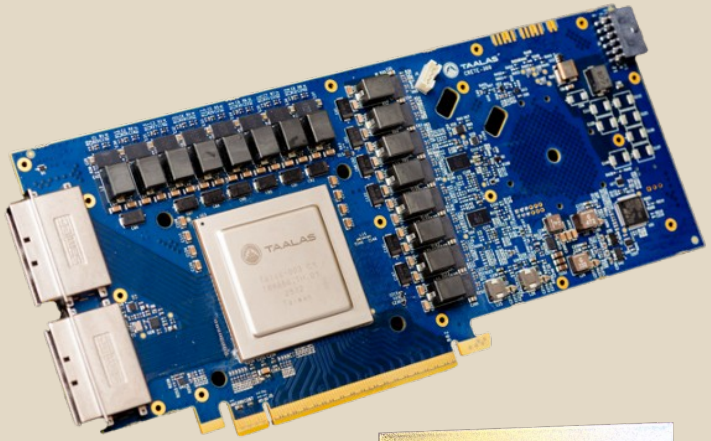
Las infraestructuras como **LLVM** o **GCC** son utilizadas por la gran mayoría de lenguajes.



ASIC

¿Y si en vez de crear un programa para **un procesador generalista**, creásemos un circuito para **adaptarse específicamente** a la finalidad del programa?

Es lo que se conoce como **ASIC**. Con un procesador para un único fin, conseguimos **rendimientos excepcionales** con **costes de producción en masa reducidos**, pero por contra tenemos un procesador **inalterable una vez fabricado y caro si se fabrican pocas unidades**.



FPGA

¿Y si en vez de ser inalterable como un ASIC, queremos que **se pueda actualizar o solo se fabrican pocos productos**?

Las FPGA permiten exactamente esa **flexibilidad**, a costa de ser **más caros de producir en masa y tener rendimientos menos eficientes**.

Estos chips se programan con **lenguajes de descripción de hardware** (*HDL*), como son VHDL y Verilog, aunque hoy día existen opciones de más alto nivel como SystemC o Chisel, entre otros.

Espero que esta charla haya satisfecho vuestra **curiosidad**…

¡O incluso que sea el **comienzo** de una nueva **carrera profesional**!

murcia.dev

¡Muchas gracias por vuestra atención!

Contacto

https://monte.blue