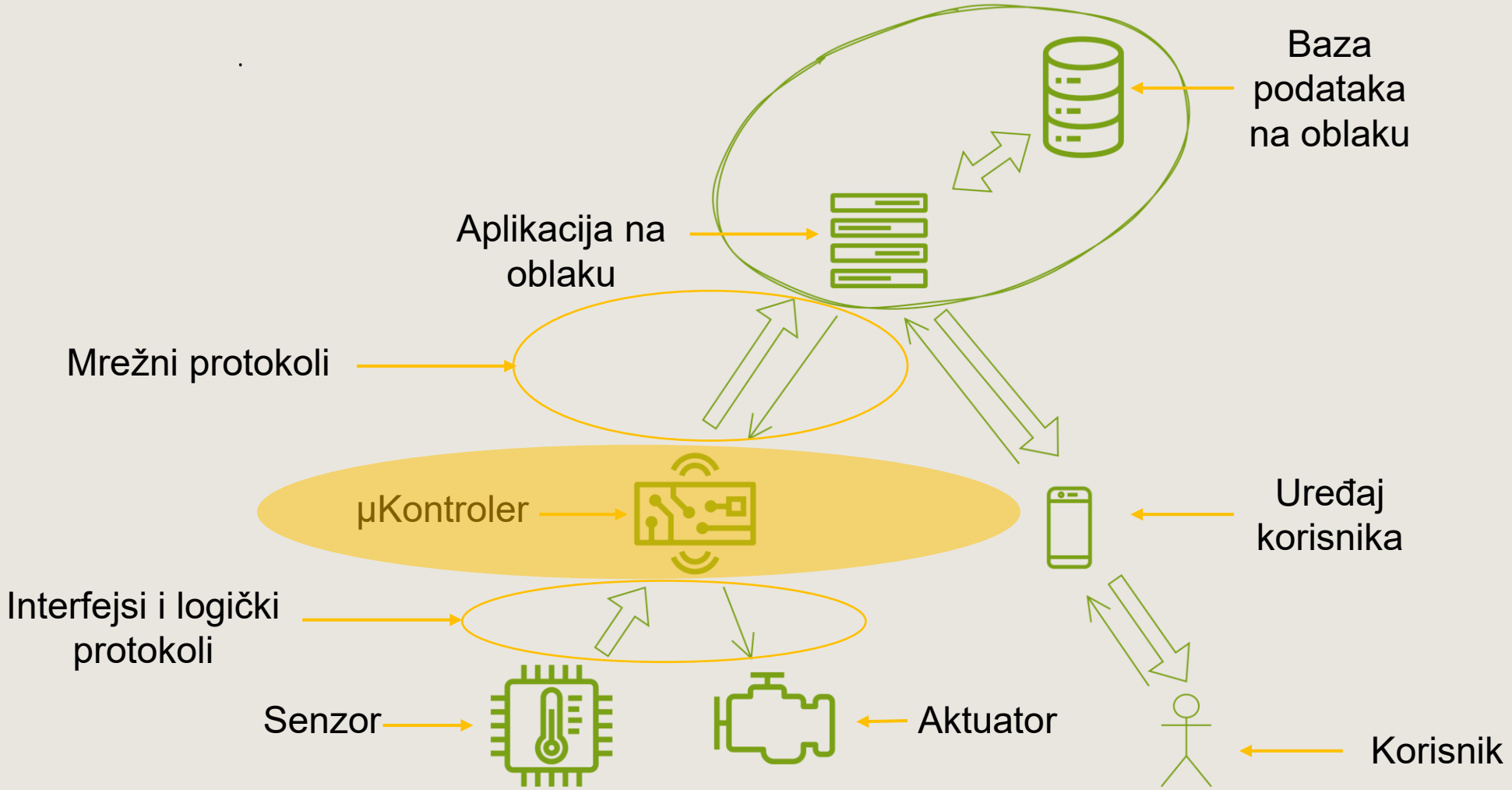
INTERNET PAMETINIH UREĐAJA

prof. dr Dejan S. Aleksić

Prirodno-matematički fakultet, Niš

03. IOT UREĐAJI

IoT UREĐAJI


Baza podataka na oblaku
Aplikacija na oblaku
Mrežni protokoli
µKontroler
Uređaj korisnika
Interfejsi i logički protokoli
Senzor
Aktuator
Korisnik

ŠTA JE "SRCE" IOT UREĐAJA?

- Najpre da definišemo šta jedan "pametni" uređaj čini pametnim.

- Od svakog IoT uređaja, od pametnog sata do industrijskog senzora, generalno se zahtevaj da ima:
  - **Senzore:** Da "oseća" svet oko sebe (temperatura, svetlost, pokret).
  - **Aktuatore:** Da "deluje" na svet (upali LED, pokrene motor, pošalje zvuk).
  - **Procesiranje:** Da obrađuje podatke sa senzora i odlučuje šta da radi.
  - **Povezivost:** Da komunicira sa drugim uređajima, serverima ili korisnikom (Internet).

- Tokom ovog kursa za praktične vežbe koristićemo IoT uređaje tj. razvojne platforme na bazi ESP32 SoC

- Ali pre toga da se podsetimo malo istorije ...

EVOLUCIJA "MOZGA" UREĐAJA: OD KALKULATORA DO IOT-A

- Da bismo razumeli zašto je ESP32 revolucionaran, moramo videti kako smo do njega došli.

- Putanja razvoja išla je ka sve većoj integraciji i možemo izdvojiti tri glavne faze razvoja:

  - **Faza 1:** Era Mikroprocesora (μP) (1970-e / 1980-e)

  - **Faza 2:** *Era Mikrokontrolera (MCU) (1980-e / 1990-e)*

  - **Faza 3:** *Era Sistema na čipu (SoC) (2000-e - Danas)*

EVOLUCIJA "MOZGA" UREĐAJA: OD KALKULATORA DO IOT-A

- **Faza 1: Era Mikroprocesora (µP) (1970-e / 1980-e)**
  - **Primeri:** Intel 8080 (IBM PC), Zilog Z80 (ZX spectrum), MOS 6502 (Commodore 64),
  - **Šta je to?** Samo centralna procesorska jedinica (CPU) – "mozak".
  - **Problem:** Da bi radio, µP je zahtevao mnogo dodatnih, eksternih čipova na štampanoj ploči:
    - Poseban čip za RAM (memoriju)
    - Poseban čip za ROM (skladištenje programa)
    - Poseban čip za serijsku komunikaciju (UART)
    - Poseban čip za digitalne ulaze/izlaze (I/O)
    - Poseban čip za A/D konvertore...
  - **Rezultat:** Velike, skupe i kompleksne ploče.

EVOLUCIJA "MOZGA" UREĐAJA: OD KALKULATORA DO IOT-A

- **Faza 2:** ***Era Mikrokontrolera (MCU) (1980-e / 1990-e)***
  - **Primeri:** Intel 8051, Microchip PIC, Atmel AVR (Srce Arduina).
  - **Ključna ideja:** Integracija!
  - **Šta je to znači?** Jedan čip koji objedinjuje:
    - CPU (mozak)
    - RAM (memoriju)
    - Flash/ROM (skladištenje)
    - Digitalne I/O pinove
    - A/D konvertore, tajmere, UART...
  - **Rezultat:** Idealan za upravljanje (kontrolu) uređajima (veš mašina, mikrotalasna, robot).
  - Ovo je bio "mozak" za većinu **embedded** (ugrađeni) sistema.

EVOLUCIJA "MOZGA" UREĐAJA: OD KALKULATORA DO IOT-A

- **Faza 3:** ***Era Sistema na čipu (SoC) (2000-e - Danas)***
  - ***Primeri****: ESP32, Qualcomm Snapdragon (telefoni), Raspberry Pi (delimično).*
  - ***Ključna ideja:*** *Totalna Integracija.*
  - ***Šta je to?*** *Imaju sve što ima MCU i dodaju se još kompleksniji, specijalizovani blokovi:*
    - *Mnogo moćniji (često dvojezgarni) CPU*
    - *Radio-frekventne module (Wi-Fi, Bluetooth)*
    - *Grafičke procesore (GPU) (kod mobilnih SoC)*
    - *Kompletne sisteme za upravljanje napajanjem.*
  - ***Zaključak:*** *ESP32 nije samo mikrokontroler; on je SoC jer integriše "mozak" (MCU), senzore i "povezivost" (Wi-Fi/BT) na jednom mestu.*

ŠTA SU "EMBEDDED" (UGRAĐENI) SISTEMI?

- Embedded (ugrađeni) sistem je specijalizovani računarski sistem (kombinacija hardvera i softvera) dizajniran da izvršava tačno određeni zadatak unutar nekog većeg uređaja.

- Za razliku od klasičnih računara, embedded sistemi:
  - imaju **jasno definisanu funkciju**
  - često rade **u realnom vremenu**
  - optimizovani su za **pouzdanost**, **stabilnost** i **energetsku efikasnost**
  - najčešće korisnik ne vidi da uopšte postoji računar u pozadini

ŠTA SU "EMBEDDED" (UGRAĐENI) SISTEMI?

- **Karakteristike embedded sistema**
  - Ograničeni resursi (CPU, RAM, Flash)
  - Niska potrošnja energije
  - Visoka pouzdanost i stabilnost
  - Radi 24/7 bez prekida
  - Povezanost sa senzorima i aktuatorima
  - Često integriše komunikacione module (Wi-Fi, BLE, ZigBee, LoRa)
- **Primeri embedded sistema**
  - Mrežni uređaji (modemi, routeri, ...)
  - Automobilski sistemi (ABS, ESC, airbag kontroler)
  - Medicinska oprema (EKG, infuzione pumpe)
  - IoT uređaji (senzori, aktuatori, pametni prekidači)
  - Industrijska automatizacija (PLC moduli, robotika)
  - Kućni aparati (veš mašina, klima, televizor)

ŠTA SU "EMBEDDED" (UGRAĐENI) SISTEMI?

Kriterijum	**Embedded**	**Opšti računar (PC)**
CPU	8–240 MHz	3–5 GHz
RAM	1 KB – 1 MB	8–64 GB
OS	RTOS / Bare-metal	Windows / Linux
Ciklus života	5–20 godina	2–5 godina
Cena	0.5–10€	500–2000€

- Može se reći da je: IoT = mreža embedded uređaja koji komuniciraju međusobno i sa računarima u oblaku (cloud),
- ESP32 je upravo tipičan primer moćnog i jeftinog embedded sistema sa integrisanom komunikacijom koji se odlikuje malom potrošnjom energije.

ŠTA JE ESP32?

- ESP32 nije samo mikrokontroler. To je SoC – System on a Chip.
- **Definicija:** System on a Chip (SoC) je integrisano kolo koje u jednom čipu objedinjuje sve ili većinu komponenti računara ili drugog elektronskog sistema.
- Proizvođač je kompanija Espressif Systems iz Šangaja. ESP32 je naslednik neverovatno popularnog čipa ESP8266.
- ESP32 u jednom, izuzetno jeftinom pakovanju, integriše:
  - Snažan procesor (mozak)
  - Wi-Fi modul (povezivost)
  - Bluetooth modul (povezivost)
  - ADC, DAC
  - GPIO pinove
  - Brojače, PWM, RTC…
- Ovo ga čini savršenim "srcem" za gotovo svaki IoT projekat

KLJUČNE MOGUĆNOSTI (ZAŠTO JE TAKO POSEBAN?)

- Povezivost (Connectivity) - Ovo je njegova glavna prednost.
  - Wi-Fi (802.11 b/g/n): Omogućava ESP32 da se direktno poveže na vašu kućnu ili poslovnu Wi-Fi mrežu i komunicira sa internetom.
    Može da radi kao:
    - Klijent (Station): Povezuje se na postojeći ruter.
    - Pristupna Tačka (Access Point): Sam kreira Wi-Fi mrežu na koju se drugi uređaji (npr. vaš telefon) mogu povezati.
  - Bluetooth (v4.2 BR/EDR i BLE):
    - Klasičan Bluetooth: Za strimovanje muzike ili prenos podataka (npr. povezivanje sa starim telefonom).
    - Bluetooth Low Energy (BLE): Apsolutno ključan za moderni IoT. Omogućava uređajima da mesecima, pa i godinama, rade sa jednom baterijom. Koristi se za komunikaciju sa pametnim telefonima, "beaconima", prenosivim uređajima itd.

KLJUČNE MOGUĆNOSTI (ZAŠTO JE TAKO POSEBAN?)

- Procesorska Snaga (Performance)
  - CPU: Dvojezgarni (Dual-Core) Tensilica Xtensa LX6 procesor, koji radi na taktu do 240 MHz.
  - Zašto je Dual-Core važan? U IoT-u, upravljanje Wi-Fi i Bluetooth protokolima (tzv. TCP "stack") je procesorski veoma zahtevno. ESP32 elegantno rešava ovaj problem:
    - Jezgro 0 (Core 0): Često rezervisano za "radio" (Wi-Fi/BT) operacije.
    - Jezgro 1 (Core 1): Potpuno slobodno za vaš programski kod (aplikaciju).
  - Ovo znači da vaša aplikacija (npr. očitavanje senzora) radi glatko, bez obzira na to šta se dešava sa mrežnom konekcijom.

PORODICA ESP32: NIJE SVE ISTO

- "ESP32" je postao generički termin, ali u praksi, to je **čitava familija (serija) čipova** koju Espressif konstantno razvija.

- Različiti projekti imaju različite zahteve (cena, performanse, potrošnja, specifične konekcije), pa tako postoje i različiti ESP32 čipovi.

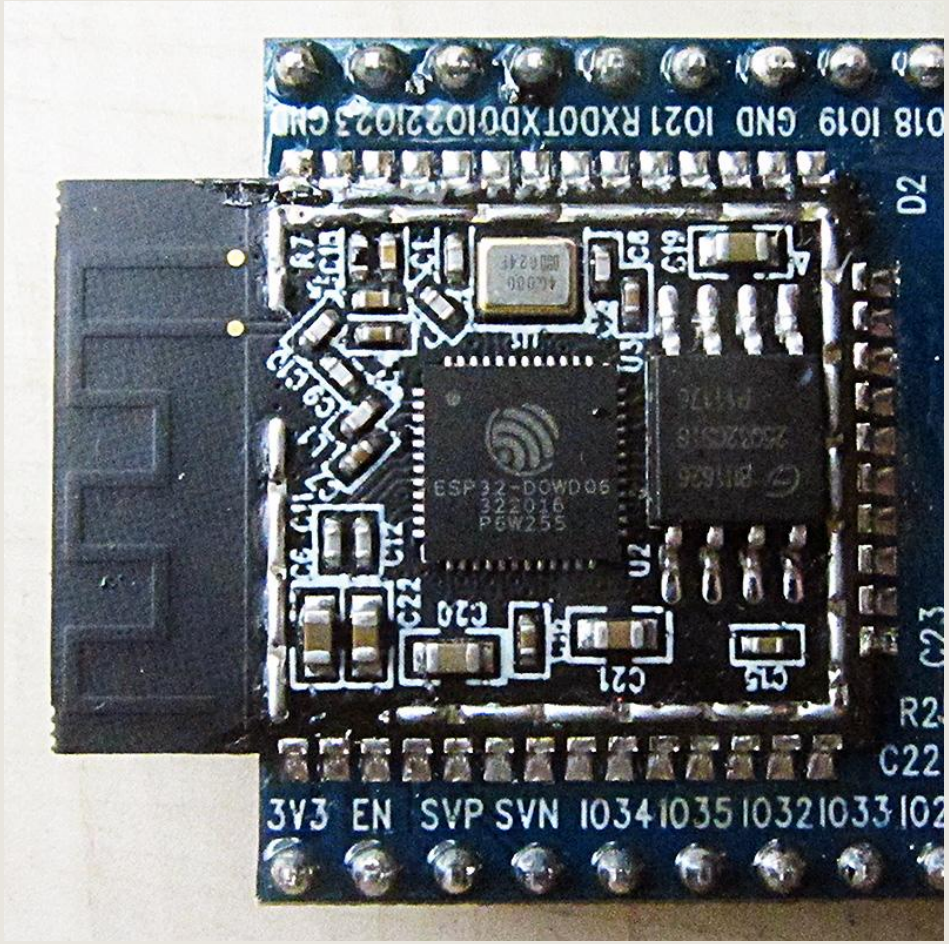
- Najvažnija podela je na serije bazirane na:
  - **Tensilica** (starija, proverena arhitektura)
  - **RISC-V** (novija, otvorenija arhitektura).

PORODICA ESP32: NIJE SVE ISTO

Serija / Model	CPU (Jezgro)	Wi-Fi	Bluetooth	Ključne Karakteristike i Namena
ESP32 (Classic)	Dual-Core Tensilica **LX6**	802.11 b/g/n	**Classic + BLE**	**"Original"** i najsvestraniji čip. Onaj o kojem smo pričali. Odličan za učenje i većinu standardnih projekata.
ESP32-S2	Single-Core Tensilica **LX7**	802.11 b/g/n	**NEMA BLUETOOTH**	Optimizovan za nisku potrošnju i bezbednost. Poseduje **USB OTG** (može da se ponaša kao USB uređaj – npr. tastatura).
ESP32-S3	Dual-Core Tensilica **LX7**	802.11 b/g/n	BLE 5.0 (Nema Classic)	Fokus na performansama. Poseduje **AI/Vektorske instrukcije** za mašinsko učenje (TinyML). Takođe ima USB OTG.
ESP32-C3	Single-Core **RISC-V**	802.11 b/g/n	BLE 5.0	**Niska cena.** Dizajniran kao direktan naslednik popularnog ESP8266, ali sa dodatom BLE podrškom.
ESP32-C6	Single-Core **RISC-V**	**Wi-Fi 6**	BLE 5.0 + **Thread / Zigbee**	**Budućnost IoT-a.** Pored Wi-Fi 6, ima radio za **Matter** protokol (Thread/Zigbee). Omogućava direktnu komunikaciju sa drugim uređajima.

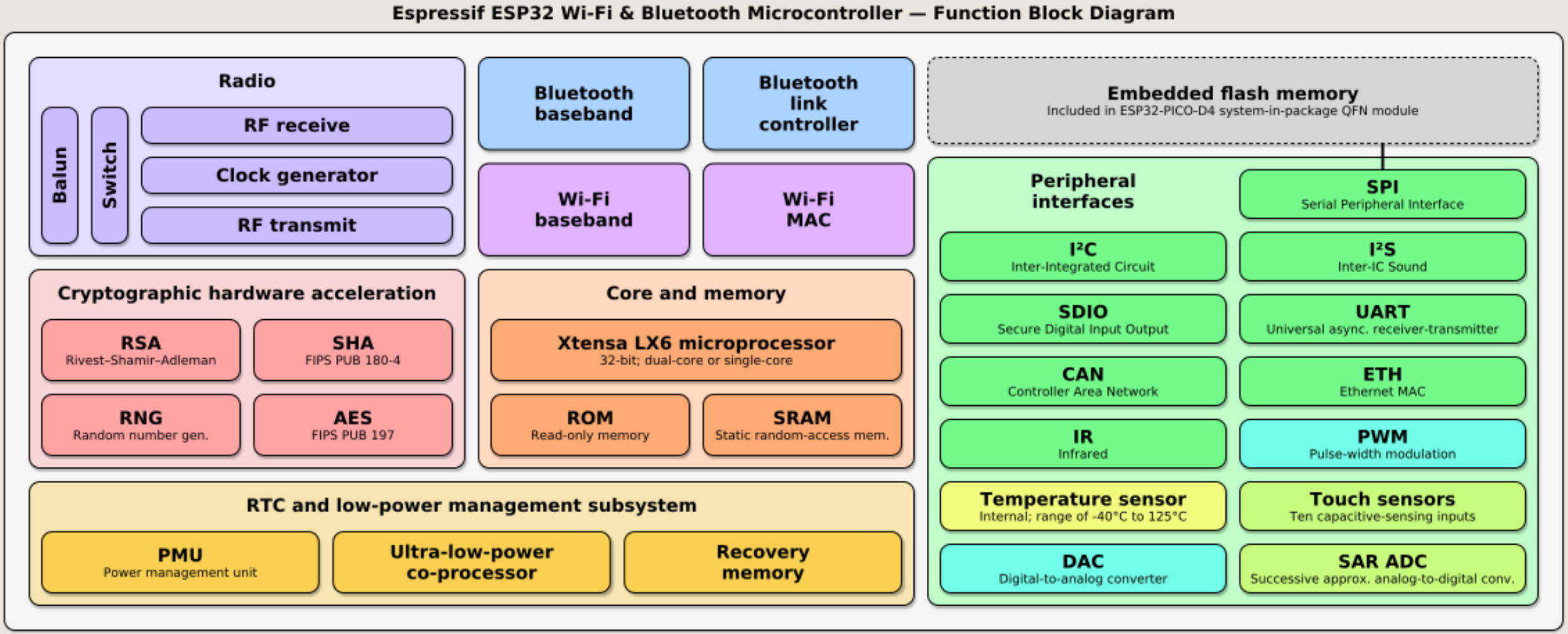
INTERNA STRUKTURA (ARHITEKTURA)

Ne morate znati svaki tranzistor, ali morate razumeti osnovne blokove:



By Brian Krent - Own work, CC BY-SA 4.0,

INTERNA STRUKTURA (ARHITEKTURA)



By Brian Krent (talk · contribs) - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=72304119

INTERNA STRUKTURA (ARHITEKTURA)

- CPU (Jezgra): Dva Xtensa LX6 jezgra.
- Memorija:
  - ROM (Read-Only Memory): Sadrži osnovni bootloader (program koji se pokreće prvi) i neke osnovne funkcije.
  - SRAM (Static RAM): Memorija za podatke i izvršavanje programa. Brza, ali gubi podatke kada se isključi napajanje.
  - Flash Memorija: Ovo je mesto gde je uskladišten vaš program. Ovo je eksterna memorija (obično 4MB, 8MB ili 16MB) koja se nalazi na modulu, odmah pored ESP32 čipa.
- RF (Radio Frekvencija): Kompletan radio-primopredajnik sa antenom (bilo na pločici ili konektorom za eksternu antenu).
- Periferije: Ovo je skup "alata" koje ESP32 nudi za komunikaciju sa spoljnim svetom.

SLEEP MODOVI ESP32 I ULP (ULTRA-LOW-POWER) JEZGRO

- Jedan od najvećih izazova za IoT uređaje je trajanje baterije.
- Wi-Fi i moćni procesori su veoma gladni energije (često troše 150-250mA dok rade).
- ESP32 ovaj problem rešava tako što omogućava uređaju da spava većinu svog vremena.
- **Ključna IoT strategija:**

Probudi se →
Očitaj senzor →
Poveži se na Wi-Fi →
Pošalji podatke →
Idi na spavanje (Deep Sleep)

GLAVNI MODOVI POTROŠNJE

Mod (Stanje)	Šta Radi?	Tipična Potrošnja	Buđenje (Wake-up)
Active	CPU i Radio (Wi-Fi/BT) aktivni. Obavlja se obrada i komunikacija.	50 - 260 mA	-
Modem Sleep	CPU je aktivan, ali je Radio (Wi-Fi/BT) privremeno ugašen između slanja paketa (DTIM).	~ 20 - 70 mA	Automatsko, veoma brzo (µs).
Light Sleep	CPU pauziran. Radio ugašen ali je u modu Čekaj Wi-Fi paket. **RAM i stanje CPU se čuvaju.**	~ 0.8 mA (800 µA)	Veoma brzo (µs). Nastavlja rad tačno gde je stao.
Deep Sleep	**Gotovo sve ugašeno.** CPU, Radio i **glavni RAM su ugašeni.** Samo RTC (sat) i ULP rade.	~ 10 - 150 **µA** (mikroampera)	Sporije (ms). Uređaj se praktično **resetuje** i kreće iz početka.
Hibernation	**Sve ugašeno**, uključujući i brzi RTC tajmer. Samo spori 32kHz sat radi (ako je podešen).	~ 2.5 - 5 **µA**	Najsporije buđenje (reset). Moguće samo preko RTC tajmera ili određenih pinova.

ŠTA ZNAČI "ČEKA WI-FI PAKET" U LIGHT-SLEEP MODU NA ESP32?

Kada ESP32 uđe u Light-sleep režim, glavna jezgra (Xtensa CPU) se isključuju, ali Wi-Fi/BT modem ostaje aktivan u posebnom low-power stanju koje Espressif zove **Modem-sleep + Light-sleep** kombinacija.

Komponenta	Stanje u Light-sleepu	Potrošnja
Glavni CPU (240 MHz)	Isključen	0 mA
RAM	Zadržava sadržaj (refresh)	~0.5 mA
RTC + periferije	Rade (timer, GPIO, touch, ULP…)	
Wi-Fi modem	**Ne radi kontinuirano**, ali se **povremeno budi** da proveri ima li paketa sa Access Point-a	20–60 mA kada je budan

KAKO WI-FI "ČEKA PAKET"?

ESP32 koristi standardni 802.11 power-save mehanizam:

1. Pre ulaska u Light-sleep, ESP32 kaže ruteru (AP-u):
   „Idem na spavanje, slušam samo svakih N beacon-a“ (obično svakih 100–300 ms).
2. Ruter čuva sve pakete koji su namenjeni ESP32 (buffering).
3. Svakih X milisekundi (Delivery Traffic Indication Map – DTIM period):
   - Wi-Fi radio se brzo probudi (za par milisekundi)
   - Pročita beacon od rutera
   - Ako beacon kaže „Imam paket za tebe“ → ESP32 ostaje budan i preuzima podatke
   - Ako nema paketa → odmah se vraća u Light-sleep

Zato se potrošnja kreće 20–60 mA prosečno (a ne 150–200 mA kao kad je Wi-Fi stalno uključen).

PRAKTIČNI PRIMERI KADA SE KORISTI

Situacija	Zašto Light-sleep + Wi-Fi čekanje?
MQTT klijent koji čeka komande	Ne želiš da izgubiš konekciju, a želiš nisku potrošnju
OTA ažuriranje	Mora da ostane dostupan na mreži
NTP sinhronizacija vremena	Čeka odgovor od servera
Web server na baterije	Klijent može da pošalje zahtev u bilo kom trenutku

ULP (ULTRA-LOW POWER) KOPROCESOR

- **Deep Sleep** je sjajan, ali šta ako treba da pratimo senzor dok glavni procesor spava?

- Tu nastupa **ULP Koprocesor**!
  - To je treći, izuzetno mali i efikasan procesor unutar ESP32.
  - On nije poseban "mod", već opcija koja može da radi tokom Deep Sleep moda.
  - Može da radi dok su glavna DVA jezgra (LX6) u DUBOKOM SNU.
  - Njegov posao je da obavlja jednostavne zadatke:
    - Prati vrednosti sa senzora (preko ADC-a).
    - Prati I2C senzore.
    - Čeka na dodir (Touch senzori).

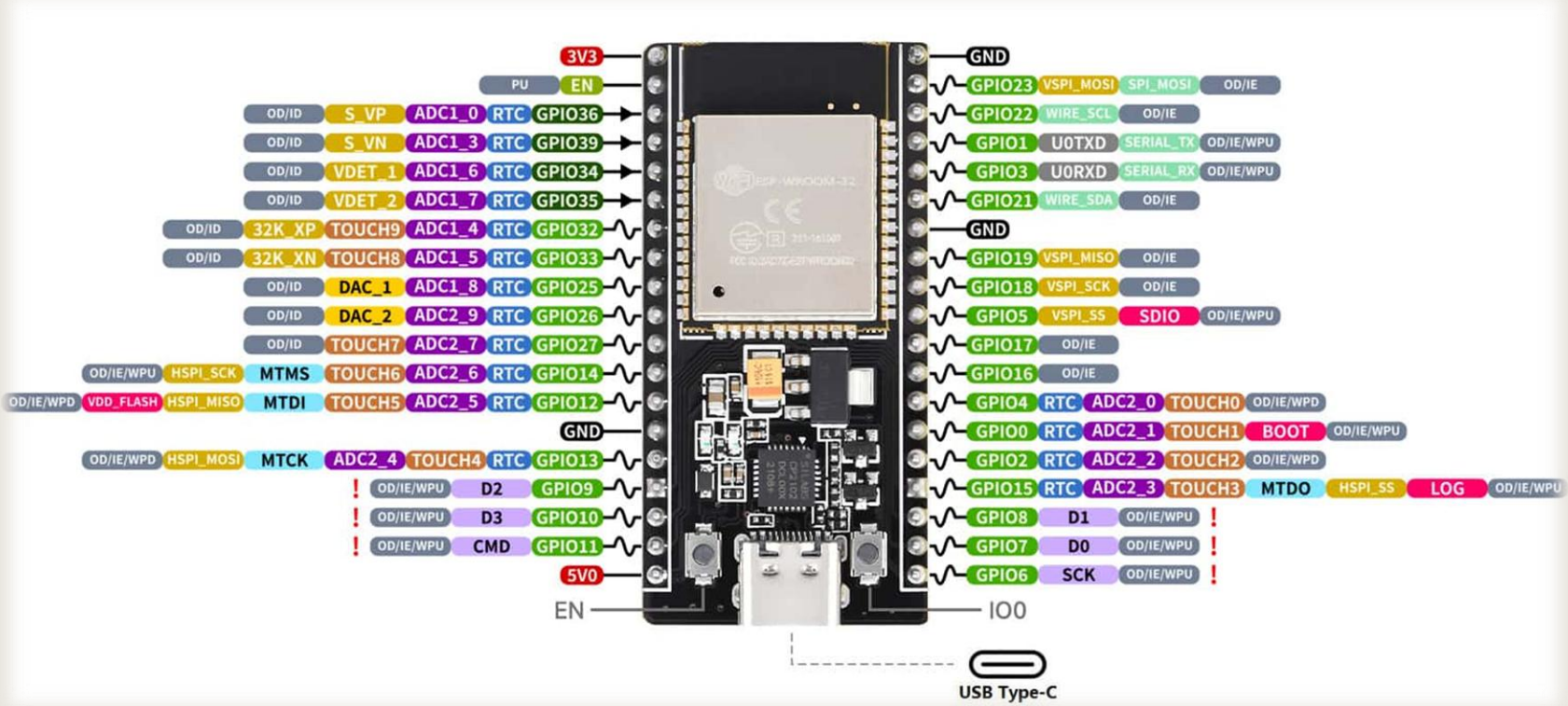
ULP (ULTRA-LOW POWER) KOPROCESOR

- **Primer pametnog scenarija (Deep Sleep + ULP):**
  1. Glavni ESP32 procesori su u Deep Sleep modu (troše ~10μA).
  2. ULP koprocesor se budi svakih 10 sekundi (troši veoma malo).
  3. ULP očita analognu vrednost sa senzora vlažnosti zemljišta.
  4. Sve dok je vrednost iznad 50%, ULP se vraća na spavanje.
  5. Čim vrednost padne ispod 50%, ULP šalje signal za buđenje glavnim procesorima.
  6. Glavni CPU (LX6) se budi, pali Wi-Fi, šalje notifikaciju "Potrebno je zalivanje!" i vraća se u Deep Sleep.

- ULP nam omogućava da ostanemo u najštedljivijem Deep Sleep modu, a da istovremeno budemo "svesni" okruženja – što je suština efikasnog IoT uređaja na baterije.

PINOVI I PERIFERIJE (KAKO GA POVEZUJEMO?)

Kada kupite ESP32, obično ga dobijete na "razvojnoj ploči" (development board) kao što su **NodeMCU-32S** ili **WEMOS D1 Mini32**.
Ove ploče imaju USB port (za programiranje i napajanje) i izvode pinove čipa.
Govorićemo o **GPIO** pinovima – **General Purpose Input/Output**.

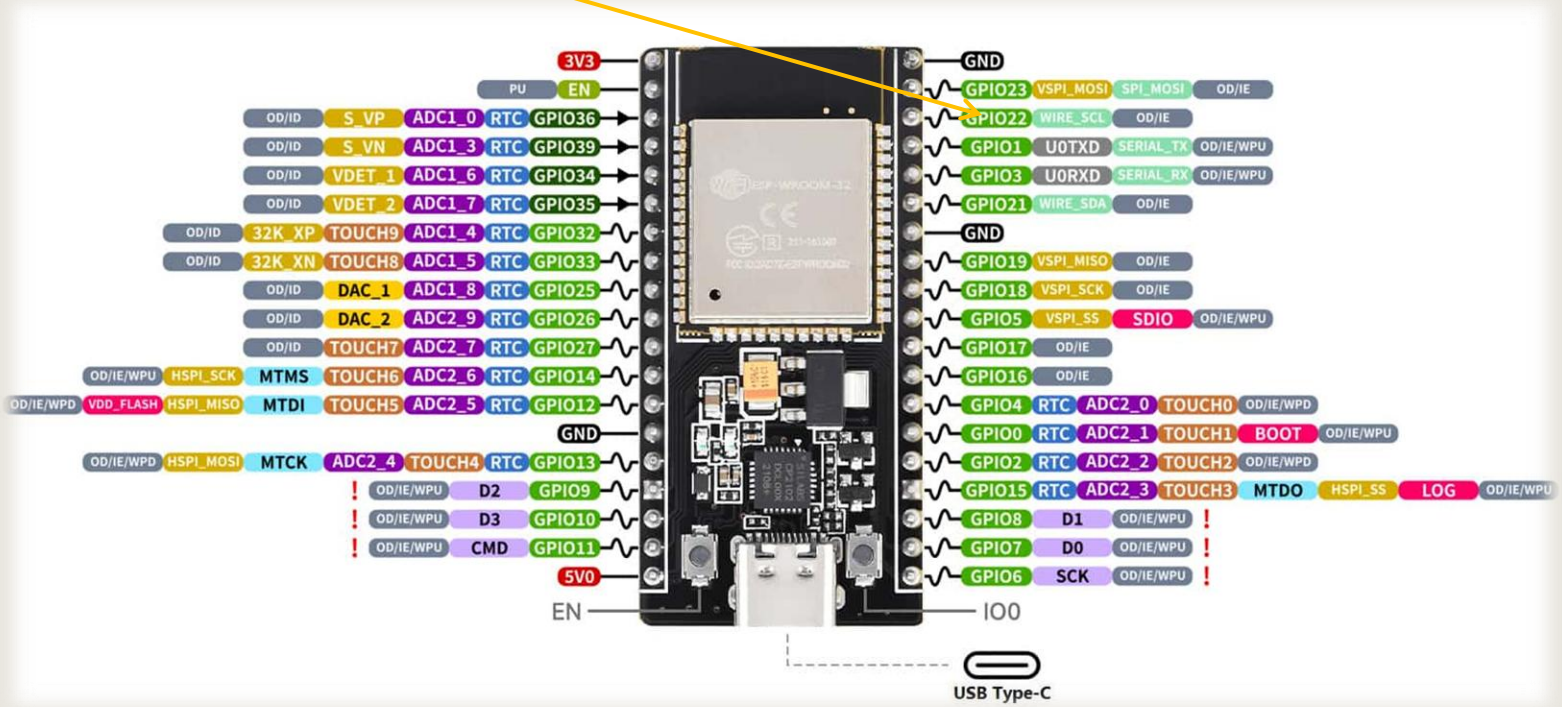


Preuzeto sa: https://malina314.com/proizvod/nodemcu-32s-esp32-wifibluetooth-development-board/

PINOVI I PERIFERIJE (KAKO GA POVEZUJEMO?)

Multiplexing (Najvažniji koncept)
Gotovo svaki pin na ESP32 je **multifunkcionalan**.
To znači da jedan isti fizički pin (npr. GPIO 22) može, kroz softver, postati:

- Digitalni izlaz (za paljenje LED diode)
- Digitalni ulaz (za čitanje tastera)
- Deo I2C magistrale (kao SCL)
- Deo SPI magistrale (kao MOSI)
- Ovo se zove **Pin Multiplexing.**
- Daje nam ogromnu fleksibilnost.



Preuzeto sa: https://malina314.com/proizvod/nodemcu-32s-esp32-wifibluetooth-development-board/

PINOVI I PERIFERIJE (KAKO GA POVEZUJEMO?)

Ključne Grupe Pinova - Evo šta sve ESP32 nudi (ne morate pamtiti brojeve pinova, već tipove funkcija):

- Digitalni I/O: Standardni ulazi i izlazi. Većina pinova.
- Analogni Ulazi (ADC - Analog-to-Digital Converter): Omogućavaju čitanje analognih vrednosti (npr. napon sa potenciometra ili analognog senzora), a ne samo 0 i 1. ESP32 ima dva ADC-a sa do 18 kanala.
- Analogni Izlazi (DAC - Digital-to-Analog Converter): Omogućava generisanje analognog naponskog signala. Ima 2 DAC-a.
- Senzori Dodira (Touch Sensors): 10 pinova mogu da se koriste kao kapacitivni senzori dodira. Možete napraviti dugme na dodir bez ikakvih dodatnih komponenti.
- Hall Senzor: Integrisan senzor koji detektuje magnetno polje.

PINOVI I PERIFERIJE (KAKO GA POVEZUJEMO?)

Komunikacioni Interfejsi (Magistrale) - Ovo je način na koji ESP32 komunicira sa drugim čipovima i senzorima:

- UART (x3): Serijska komunikacija. Koristi se za programiranje, slanje poruka na računar (Serial Monitor) i komunikaciju sa npr. GPS modulom.
- I2C (x2): (Izgovara se "I-squared-C"). Popularna magistrala koja koristi samo dve žice (SDA i SCL) za komunikaciju sa stotinama senzora (senzori temperature, pritiska, žiroskopi...).
- SPI (x4): Veoma brza magistrala, često se koristi za SD kartice, ekrane (TFT, OLED) i brze ADC-ove.

PINOVI I PERIFERIJE (KAKO GA POVEZUJEMO?)

1. RADNI NAPON JE 3.3V!

ESP32 radi isključivo na 3.3V ali većina razvojnih ploča obezbeđuje rad sa 5V
Kada povezujete senzore morate strogo voditi računa sa kojim naponom oni rade !!

2. Nisu svi pinovi isti!

Neki pinovi se koriste interno (npr. za komunikaciju sa Flash memorijom) i ne bi trebalo da ih koristite (GPIO 6-11).
Neki pinovi su "samo ulazni" (Input-Only), npr. GPIO 34, 35, 36, 39. Na njih ne možete slati HIGH signal.

ALATI I IDE OKRUŽENJA ZA PROGRAMIRANJE ESP32

Alat / Okruženje	Glavni Jezik	Nivo Kompleksnosti	Za Koga/Šta je Najbolji?
Arduino IDE	C++ (sa Arduino API-jem)	**Početni / Hobi**	+ Najlakši ulazak u svet hardvera. + Ogromna zajednica i hiljade biblioteka. - Ograničena kontrola nad funkcijama čipa.
PlatformIO (u VS Code)	C++ (Arduino ili ESP-IDF)	**Srednji / Profesionalni**	**"Arduino na steroidima."** + Moderan IDE (Visual Studio Code). + Odlično upravljanje bibliotekama i projektima. + Podržava unit testing i Git integraciju.
ESP-IDF (Espressif IoT Framework)	C / C++ (sa FreeRTOS)	**Profesionalni / Ekspert**	**Zvanični alat proizvođača.** + Maksimalne performanse i efikasnost. + Potpuna kontrola nad oba jezgra, ULP-om i radiom. - Najkompleksniji, zahteva poznavanje RTOS-a.
MicroPython / CircuitPython	Python 3	**Početni / Brzi Prototip**	**Za one koji dolaze iz sveta softvera.** + Laka sintaksa i brzo učenje. + Interaktivan rad (REPL) za testiranje. - Sporije performanse i veća potrošnja memorije.

ŠTA JE PLATFORMIO ?

- PlatformIO je moderno okruženje za razvoj (IDE) i build sistem za embedded uređaje.

- Radi kao proširenje u VS Code i omogućava:
  - ✓ rad sa stotinama mikrokontrolera (ESP32, STM32, AVR…)
  - ✓ automatsko preuzimanje toolchain-ova
  - ✓ upravljanje bibliotekama
  - ✓ više konfiguracija projekata
  - ✓ integrisan serijski monitor, OTA upload, debugging

- PlatformIO je zapravo profesionalna alternativa Arduino IDE-u, pogodna za veće projekte.

ŠTA JE “TOOLCHAIN”?

Toolchain je skup alata koji omogućava da naš izvorni kod (C/C++ za ESP32) bude preveden u firmware koji mikrokontroler može da izvršava.

U embedded svetu, to je komplet koji obično sadrži:

1. **Prevodilac** - Prevodi C/C++ kod u mašinski kod za određeni procesor.
   Za ESP32 to je najčešće:
   - xtensa-esp32-elf-gcc (za klasični ESP32)
   - riscv32-esp-elf-gcc (za C3/S3/C6)
2. **Linker** - Spaja sve delove koda i biblioteka u jedan finalni firmware.bin.
3. **Assembler** - Pretvara niskonivonske instrukcije u binarne podatke.
4. **Build sistem** - Automatski pokreće sve ove alate, kontroliše zavisnosti i kreira finalni firmware. U PlatformIO-u to radi PIO Core.
5. **“Uploader” alat** - šalje firmware u mikrokontroler preko USB-a ili OTA.
   Za ESP32 je to esptool.py.

ZAŠTO NAM TREBA TOOLCHAIN?

- Mikrokontroleri koriste drugačiju arhitekturu od naših računara (Xtensa ili RISC-V), pa običan kompajler sa PC-ja ne može da napravi kod za ESP32.

- Toolchain je zato “specijalizovana fabrika” koja pravi firmware tačno prilagođen tom procesoru.

- PlatformIO **automatski preuzima i instalira** odgovarajući toolchain za vašu razvojnu platformu.

ZAŠTO KORISTIMO PLATFORMIO U IOT PROJEKTIMA?

- ❑ Jedinstveni workflow za različite mikrokontrolere
  - Programira se uvek na isti način
  - Build dugme radi za ESP32 isto kao i za STM32
  - Upload je uvek jedan klik
  - Sve biblioteke se instaliraju istim postupkom
  - Struktura projekta je ista za svaki MCU
  - I najbitnije: Nema ručnog podešavanja – PlatformIO sve uradi automatski - samo se promeni vrsta SoC-a u platformio.ini.
- ❑ Brže kompajliranje i bolje upravljanje zavisnostima
- ❑ Podrška za FreeRTOS, asinhrone biblioteke, napredne firmware structure
- ❑ Logična organizacija projekta (src/, include/, lib/, platformio.ini)
- ❑ Ugrađena podrška za Git
- ❑ Automatske USB/COM detekcije i konfiguracije

KAKO SE INSTALIRA PLATFORMIO U VS CODE?

- **Korak 1:** Instalirati Visual Studio Code
  - Preuzeti sa: https://code.visualstudio.com
  - Instalirati standardno (Next → Next → Finish)
- **Korak 2:** Instalirati PlatformIO Extension
  - Pokrenuti VS Code
  - Klik na Extensions (Ctrl+Shift+X)
  - Ukucati “PlatformIO IDE”
  - Klik na Install
  - Nakon instalacije, VS Code se restartuje
- **Korak 3:** Prvo pokretanje
  - Sa leve strane se pojavljuje PlatformIO Home ikona
  - U meniju postoje opcije:
    - ✓ New Project
    - ✓ Open Project
    - ✓ Libraries
    - ✓ Devices
    - ✓ Projects Overview

KREIRANJE PRVOG ESP32 PROJEKTA

1. PlatformIO Home → **New Project**
2. Uneti naziv projekta (npr. *ESP32_IoT_Lab1*)
3. Board: **ESP32 Dev Module** (ili ESP32-C3/Esp32-S3 po potrebi)
4. Framework: **Arduino**
5. Lokacija projekta – automatski OK
6. Klik **Finish**

PlatformIO sam:

- preuzima toolchain i platformu za ESP32
- kreira folder strukturu
- generiše osnovni platformio.ini

STRUKTURA PLATFORMIO PROJEKTA

```
project/
│
├── src/              ← ovde ide glavni .cpp fajl (main.cpp)
├── include/          ← header fajlovi
├── lib/              ← dodatne biblioteke
├── platformio.ini    ← konfiguracija projekta
└── .pio/             ← build fajlovi (automatski)
```

Prevođenje i Upload

- Build: ✓ (Ctrl+Alt+B)
- Upload: → (Ctrl+Alt+U)
- Serial Monitor: Plug-in ikonica ili pio device monitor

Sve radi direktno iz VS Code.

ŠTA JE PLATFORMIO.INI I ZAŠTO JE VAŽAN?

platformio.ini je glavni konfiguracioni fajl svakog PlatformIO projekta.

Njime opisujemo naše radno okruženje u PlatformIO:

- 👉 koju ploču koristimo
- 👉 koji framework koristimo (Arduino, ESP-IDF…)
- 👉 kako treba da se kompajlira firmware
- 👉 koje biblioteke treba automatski da preuzme
- 👉 kako da se izvrši upload (USB, OTA, baudrate…)

JEDAN PRIMER PLATFORMIO.INI FAJLA

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino

monitor_speed = 115200

lib_deps =
    bblanchon/ArduinoJson@^6.19.4
    adafruit/Adafruit BME280 Library@^2.2.2
```

[env:esp32dev]

- Ovo definiše jedno "okruženje" (environment). Možete imati više okruženja u istom projektu!
- Na primer, jedno za vašu ESP32 ploču ([env:esp32]), a drugo za Arduino Uno ([env:uno]) radi testa.
- Naziv esp32dev je proizvoljan, vi ga birate.

JEDAN PRIMER PLATFORMIO.INI FAJLA

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino

monitor_speed = 115200

lib_deps =
    bblanchon/ArduinoJson@^6.19.4
    adafruit/Adafruit BME280 Library@^2.2.2
```

platform = espressif32

- **Obavezno**. Govori PlatformIO koji "paket" alata da koristi.
- Ovde kažemo da kompajliramo za Espressif 32-bitnu familiju (ESP32, S2, S3, C3...).
- Druge vrednosti bi bile npr. atmelavr (za Arduino Uno) ili ststm32.

JEDAN PRIMER PLATFORMIO.INI FAJLA

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino

monitor_speed = 115200

lib_deps =
    bblanchon/ArduinoJson@^6.19.4
    adafruit/Adafruit BME280 Library@^2.2.2
```

board = esp32dev

- **Obavezno**. Definiše koju tačno ploču koju koristite.
- Ovo je ključno! PIO sada zna tačan raspored pinova, količinu memorije, brzinu procesora i kako da "otpremi" kod.
- Uobičajene vrednosti su nodemcu-32s, esp32dev, lolin_d32 itd.

JEDAN PRIMER PLATFORMIO.INI FAJLA

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino

monitor_speed = 115200

lib_deps =
    bblanchon/ArduinoJson@^6.19.4
    adafruit/Adafruit BME280 Library@^2.2.2
```

framework = arduino

- **Obavezno**. Govori PIO koji "jezik" ili API želite da koristite.
- arduino znači da želimo da koristimo setup(), loop(), digitalWrite() i sve Arduino funkcije.
- Alternativa bi bila espidf (profesionalni Espressif framework).

JEDAN PRIMER PLATFORMIO.INI FAJLA

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino

monitor_speed = 115200

lib_deps =
    bblanchon/ArduinoJson@^6.19.4
    adafruit/Adafruit BME280 Library@^2.2.2
```

monitor_speed = 115200

- **Opciono**. Postavlja podrazumevanu brzinu (Baud rate) za Serial Monitor. Štedi vam jedan klik svaki put kada otvorite monitor.

JEDAN PRIMER PLATFORMIO.INI FAJLA

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino

monitor_speed = 115200

lib_deps =
    bblanchon/ArduinoJson@^6.19.4
    adafruit/Adafruit BME280 Library@^2.2.2
```

lib_deps = (Library Dependencies)

- Najmoćnija funkcija PlatformIO!
- Ovde navodite SVE biblioteke koje su potrebne vašem projektu.
- PlatformIO će ih automatski pronaći i instalirati u lokalni folder projekta (.pio).
- *bblanchon/ArduinoJson* je ime biblioteke (autor/ime).
- @^6.19.4 je verzija.
- Ovo garantuje da će projekat uvek koristiti tu verziju (ili kompatibilnu), čak i ako autor biblioteke napravi drastične izmene u budućnosti.

```cpp
#include <Arduino.h>
#include <WiFi.h>

void setup() {
  Serial.begin(115200);
  pinMode(2, OUTPUT); // LED na većini ESP32 ploča

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  int n = WiFi.scanNetworks();
  Serial.printf("Pronađeno %d mreža:\n", n);
  for (int i = 0; i < n; i++) {
    Serial.printf("%d: %s (%d dBm)\n", i+1, WiFi.SSID(i).c_str(), WiFi.RSSI(i));
  }
}

void loop() {
  digitalWrite(2, HIGH);
  delay(500);
  digitalWrite(2, LOW);
  delay(500);
}
```

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0
mode:DIO, clock div:2
load:0x3fff0030,len:1184
load:0x40078000,len:12776
load:0x40080400,len:3032
entry 0x400805e4
Pronađeno 4 mreža:
1: TP-Link_7DE7 2.4_EXT (-58 dBm)
2: ALEKSIC (-80 dBm)
3: CGA2121_xsQJuVn (-83 dBm)
4: dlink-4227 (-93 dBm)
```

Akcija	Prečica	Ikona
Build	Ctrl+Alt+B	✓
Upload	Ctrl+Alt+U	→
Monitor	Ctrl+Alt+S	
Clean	—	

File Edit Selection View Go Run Terminal Help

Blink_WiFi_Scan

PIO Home | main.cpp | platformio.ini

src > main.cpp > setup()

```cpp
#include <Arduino.h>
#include <WiFi.h>

void setup() {
  Serial.begin(115200);
  pinMode(2, OUTPUT); // LED na većini ESP32 ploča

  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  int n = WiFi.scanNetworks();
  Serial.printf("Pronađeno %d mreža:\n", n);
  for (int i = 0; i < n; i++) {
    Serial.printf("%d: %s (%d dBm)\n", i+1, WiFi.SSID(i).c_str(), WiFi.RSSI(i));
  }
}

void loop() {
  digitalWrite(2, HIGH);
  delay(500);
  digitalWrite(2, LOW);
  delay(500);
}
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1184
load:0x40078000,len:12776
load:0x40080400,len:3032
entry 0x400805e4
Pronađeno 4 mreža:
1: TP-Link_7DE7 2.4_EXT (-58 dBm)
2: ALEKSIC (-80 dBm)
3: CGA2121_xsQJuVn (-83 dBm)
4: dlink-4227 (-93 dBm)
```

DEBUG CONSOLE

CHAT

Build with agent mode

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

main.cpp:5

Add context (#), extensions (@), con

Agent Auto

Default (Blink_WiFi_Scan) Auto Java: Ready Ln 5, Col 22 (6 selected) Spaces: 2 UTF-8 CRLF C++ PlatformIO