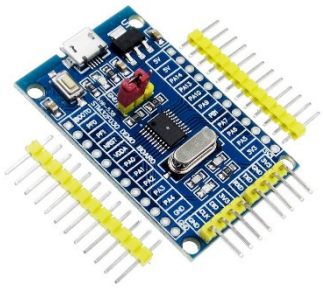
01 Aplikace mikroprocesorů 2

Obsah

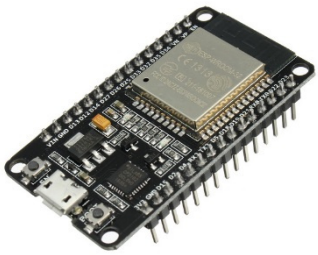
Aplikace mikroprocesorů 2

- Cílem je seznámit posluchače s aplikací ARM Cortex mikroprocesorů, zejména ve vestavěných aplikacích.
- Předmět logicky navazuje na znalosti získané v předmětu Aplikace mikroprocesorů 1 a posouvá znalosti k nasazení výkonnějších procesorů.
- Cílem je studenty seznámit s architekturou 32 bitových procesorů s využitím vyspělých periferií (čítače, AD převodníky) dále připojení a obsluhu náročnějších periferií jako USB.



Technické informace

- Vzhledem k současné situaci nedodržíme základní strukturu Přednáška – Cvičení.
  - Budeme organizovat formou navazujících bloků.
- Výuka proběhne pomocí MS-Teams.
- Účast na jednotlivých blocích je povinná.
- Bude hojně využíván systém domácích úkolů – samostudium.
- Kurz zakončíme zpracováním praktické semestrální práce.



První část kurzu založíme na vývojovém kitu Espressif ESP32.

Později přistoupíme k procesorům STM32.

Zkouška a zápočet

- Kurz je organizován do 14 týdnů. Polední týden je zápočtový.
- V průběhu kurzu je student povinen účastnit se všech cvičení (MS-teams)
- Během semestru jsou zadávány úkoly pro samostudium - pro získání zápočtu je nutné odevzdat všechny.
- V závěru semestru je zadána semestrální práce.
    - Student je povinen demonstrovat autorství a natočit krátké (cca 3 min) prezentační video v EN.
- Zkouška má dialogickou formu. V rámci zkoušky musí student předvést aktivní znalosti témat přednesených na přednáškách i znalosti získané při cvičeních.

Osnova kurzu

1. 32 bitová architektura ARM Cortex HW zapojení systémů s rychlými mikroprocesory.
2. Vývojové prostředí, překladače, programátory, příprava a konfigurace vývojového řetězce. Instalace a nastavení křížové kompilace. Programovací rozhraní JTAG, SWD
3. Minimální zapojení CPU. Typy výstupních bran MCU a jejich vlastnosti. Porty náhradní schéma, charakteristiky.
4. Jádro ARM, rozdělení a vlastnosti jader ARM, architektura a instrukční sada procesorů s jádrem ARM.
5. Paměťový model ARM Cortex.
6. Zdroj systémových hodin, externí zdroj hodinového signálu, fázový závěs.
7. Obvod Reset, Watchdog, Selfreset CPU, BOD.
8. Systémové rozhraní, Napájecí systém, podpůrné obvody (reset, BOD, watchog, oscilátory).
9. I/O operace, porty, sériová rozhraní, přerušení, Alternativní funkce GPIO. Komunikační protokoly.
10. AD-DA převody, generování signálů, DMA.
11. Specifická rozhraní, LCD port, Camera port, grafické výstupy, připojení a práce s LCD displejem.
12. USB rozhraní, VCP komunikace pomocí integrovaného USB rozhraní
13. Frameworky pro ARM, přizpůsobení vybrané architektuře.

Doporučená literatura

1. Herout P.: „Učebnice jazyka C (nebo jiná vhodná učebnice jazyka C)“, KOPP, ISBN 978-80-7232-351-7
2. Jonathan W. Valvano: „Embedded systems - introduction to ARM Cortex-M microcontrollers“
3. Neil Matthew a Richard Stones: „Linux - Začínáme programovat“
4. Darryl Gove, [překlad Lukáš Krejčí]: „Programování aplikací pro vícejádrové procesory“

Mikroprocesory ARM

Prostudovat:

- https://cs.wikipedia.org/wiki/Von_Neumannova_architektura
- https://cs.wikipedia.org/wiki/Harvardsk%C3%A1_architektura
- https://cs.wikipedia.org/wiki/CISC
- https://cs.wikipedia.org/wiki/RISC
- https://cs.wikipedia.org/wiki/ARM

Vývojové prostředí

Možnosti volby programového prostředí

1. Arduino IDE
2. Eclipse + PlugIn
3. VisualStudio + VisualMicro ($12/rok/student; 45$/rok komerční – 2021)

Nastavení Arduino IDE

1) Instalace ovladačů COM portu CP210x
2) Soubor->vlastnosti->Správce dalších desek - přidat:
   - https://dl.espressif.com/dl/package_esp32_index.json
3) Nástroje - Vývojová deska -> manažer desek
   - nainstalovat ESP32
4) Zvolit desku ESP32-WROOM
5) Příklad Wifi->SimpleWifiServer
   - Serial monitor 115200
   - Reset při nahrávání zmáčknout BOOT

Tutoriál:

https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

Popis vývojového kitu

ESP32 vývojový kit je velmi primitivní. Základní skladba:

- Espressif ESP32-Wroom-32
- LD1117 s 1N5819
- CP210x
- uUSB konektor
- Tlačítko BOOT bootloader mode
- Tlačítko RST/EN reset
- LED (aktivita)

Hold BOOT, then press EN briefly to enter the bootloader, then release BOOT. This way you can flash a new firmware on it.

Usually, the USB-to-UART adapter on the board can control these pins, so you don't have to do it manually.

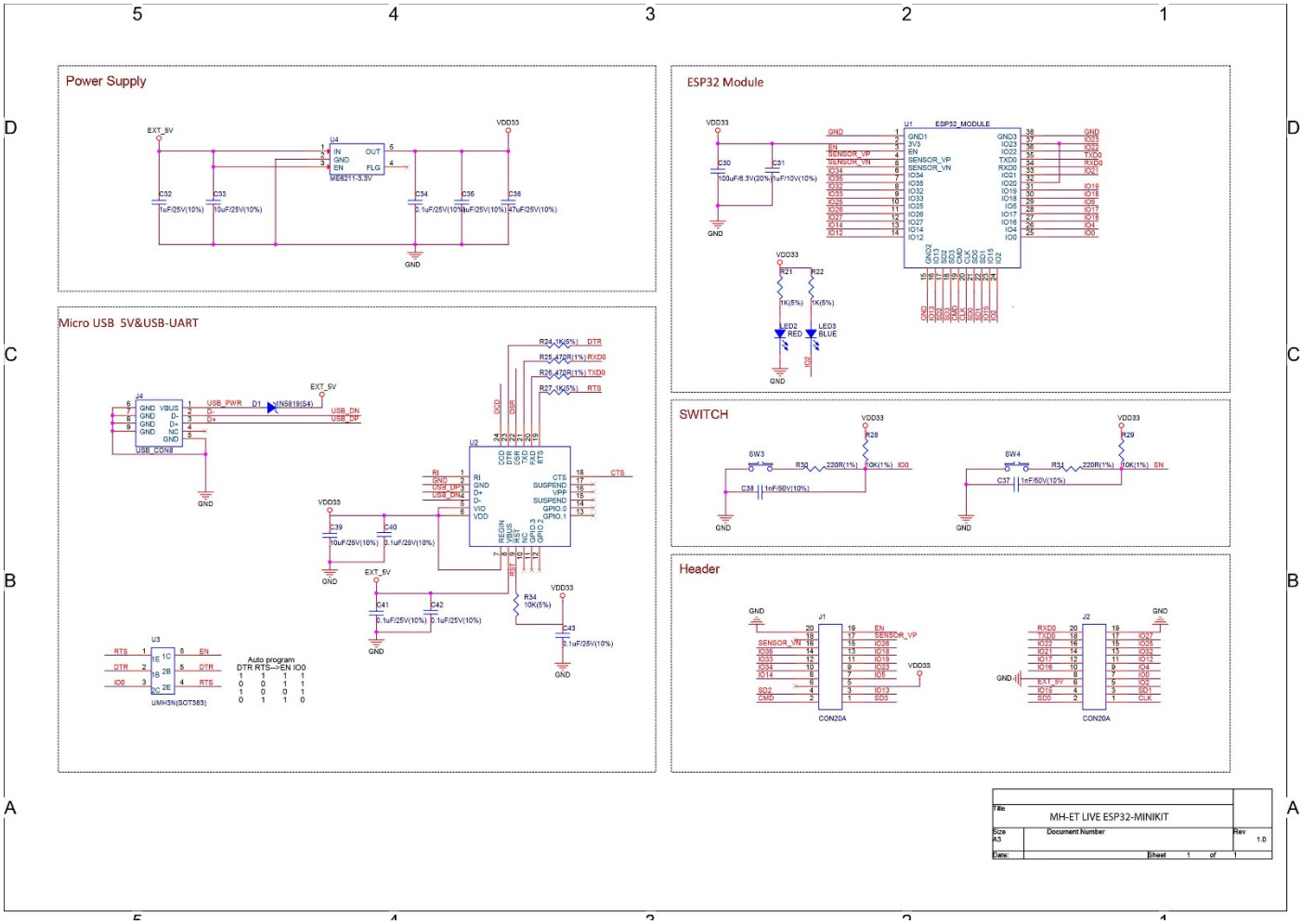
Blok napájení

Pozor! – ESD zařízení.

Základní napájení – uUSB konektor -> max. 500 mA (bez USB enumerace)

LDO stabilizátor LD1117 spolu se Schottkyho diodou determinují základní vlastnosti napájeni

- Z 5V větve odebírejte max. 500 mA
- Z 3.3V větve max. 100 mA

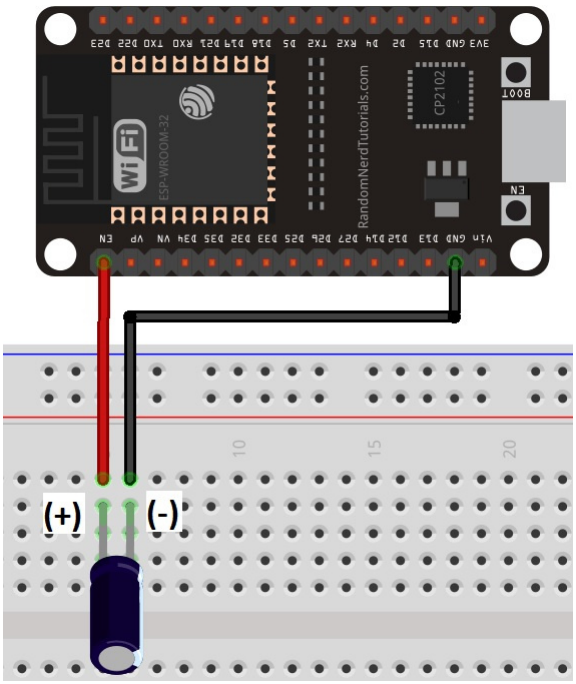


Symbol	Parameter		Min	Typ	Max	Unit
C_{IN}	Pin capacitance		-	2	-	pF
V_{IH}	High-level input voltage		0.75×VDD[1]	-	VDD[1]+0.3	V
V_{IL}	Low-level input voltage		–0.3	-	0.25×VDD[1]	V
I_{IH}	High-level input current		-	-	50	nA
I_{IL}	Low-level input current		-	-	50	nA
V_{OH}	High-level output voltage		0.8×VDD[1]	-	-	V
V_{OL}	Low-level output voltage		-	-	0.1×VDD[1]	V
I_{OH}	High-level source current (VDD[1] = 3.3 V, V_{OH} >= 2.64 V, output drive strength set to the maximum)	VDD3P3_CPU power domain [1, 2]	-	40	-	mA
		VDD3P3_RTC power domain [1, 2]	-	40	-	mA
		VDD_SDIO power domain [1, 3]	-	20	-	mA
I_{OL}	Low-level sink current (VDD[1] = 3.3 V, V_{OL} = 0.495 V, output drive strength set to the maximum)		-	28	-	mA
R_{PU}	Resistance of internal pull-up resistor		-	45	-	kΩ
R_{PD}	Resistance of internal pull-down resistor		-	45	-	kΩ
V_{IL_nRST}	Low-level input voltage of CHIP_PU to power off the chip		-	-	0.6	V

Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	240	-	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	-	190	-	mA
Transmit 802.11n, OFDM MCS7, POUT = +14 dBm	-	180	-	mA
Receive 802.11b/g/n	-	95 ~ 100	-	mA
Transmit BT/BLE, POUT = 0 dBm	-	130	-	mA
Receive BT/BLE	-	95 ~ 100	-	mA

Tlačítko BOOT

- Pro automatický přechod do BOOT módu při použití Arduino IDE přidejte C cca 10uF/10V mezi EN a GND:


(+)
(-)
RandomNerdTutorials.com
ESP-WROOM-32
CP2102
BOOT
EN

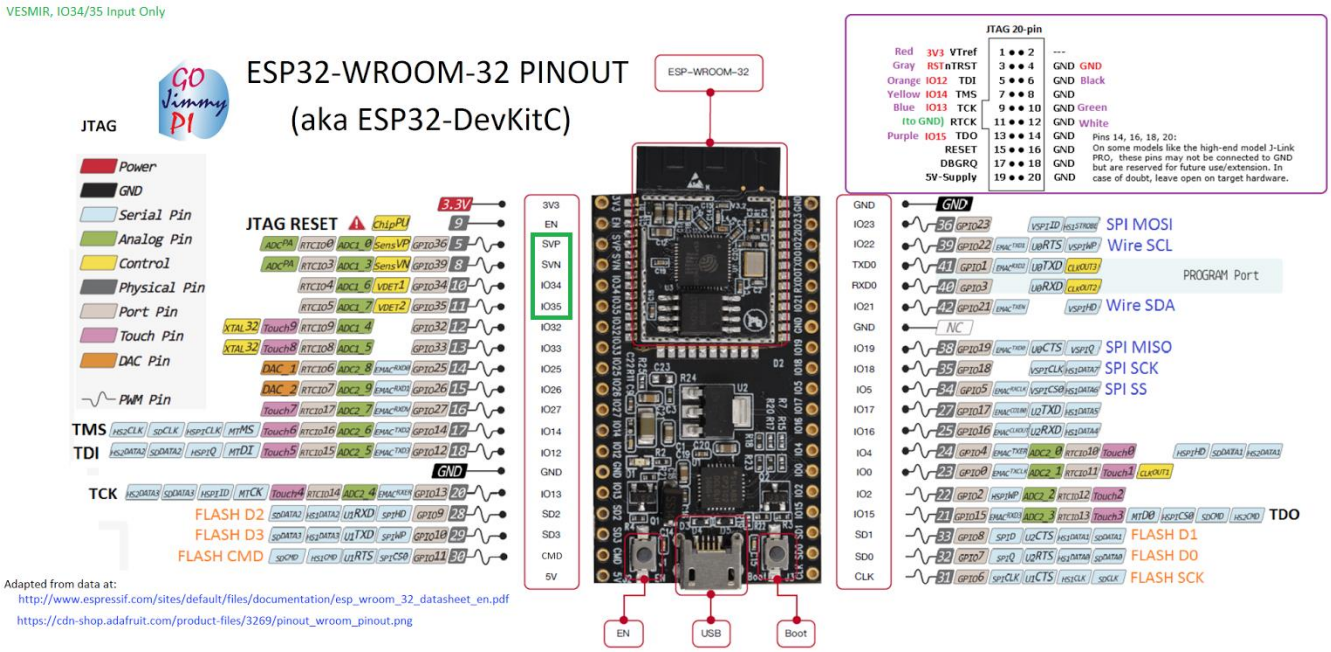
03 Aplikace mikroprocesorů 2

Obsah

GPIO - pokračování

Připomenutí: dostupné piny

Odkaz: http://martindobrovolny.cz/vyuka/bami2/download/materialy/ESP32%20Piny%20popis.pdf



Základní nastavení pinů:

- Pro definici maker prostudujte **esp32-hal-gpio.h**

- Pozor piny 34, 35, 36 a 39 jsou pouze vstupní!

```
#define MUJ_pin  22
pinMode(MUJ_pin, OUTPUT);
//INPUT, OUTPUT, INPUT_PULLUP, INPUT_PULLDOWN, OPEN_DRAIN, OUTPUT_OPEN_DRAIN, ANALOG
```

Zápis na pin:

```
digitalWrite(MUJ_pin, HIGH)      // LOW
```

Vyčtení pinu:

```
int digitalRead(MUJ_pin);
```

Externí přerušení

- GPIO přerušení – je třeba zvolit typ a registrovat obslužnou funkci.
- Pro vstup přerušení je možné využít všechny GPIO ESP32 (pozor na připojené periferie FLASH atd…)

Registrace obslužné rutiny: `attachInterrupt(GPIOPin, Moje_ISR_funkce, Mode);`

`Moje_ISR_funkce` – obslužná rutina, viz dále.

`Mode` – způsob triggeru:

`LOW`	Spuštění vyvolá hladina LOW
`HIGH`	Spuštění vyvolá hladina HIGH
`CHANGE`	Spuštění vyvolá změna
`FALLING`	Spuštění vyvolá sestupná hrana
`RISING`	Spuštění vyvolá vzestupná hrana

Odregistrování obslužné rutiny: `detachInterrupt(GPIOPin);`

Syntaxe obslužné funkce:

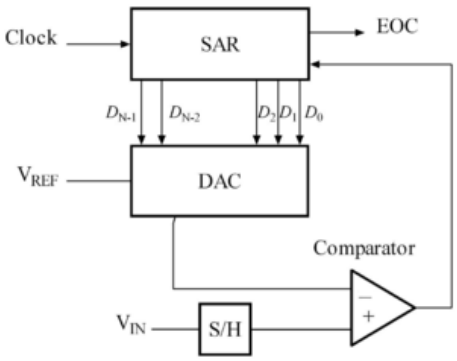
```
void IRAM_ATTR    Moje_ISR_funkce () {
      …tělo obslužné funkce
      !!!Trávit zde minimum času!
}
```

Podrobnější popis obsluhy přerušení a vstupu do kritické sekce najdete v článku: „ESP32 Arduino: External interrupts“

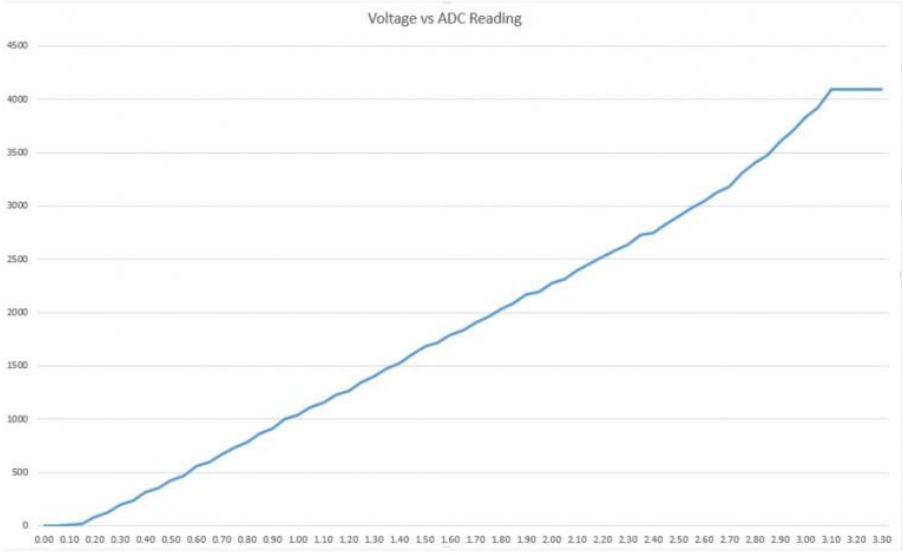
https://techtutorialsx.com/2017/09/30/esp32-arduino-external-interrupts/

ADC převodník

- ESP32 obsahuje dva 12-bit SAR (Successive Approximation Register) AD převodníky.
- K dispozici je u ADC1 8 a ADC2 10 vstupních kanálů (viz [3]).
  - Pozor!!! ADC2 nelze využít při zapnuté WiFi.
- 150 mV – 3000 mV poměrně lineární (neověřoval jsem)
- Deafult 12b (0 - 4095), ukládat do min uin16_t!
- Vref defaultně nastavena na 3,3 V



Podrobnější inf. o využití ADC naleznete v dokumentu: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html



Vyčtení hodnot v ATC-RTC módu:

1. Funkce `analogRead` – Neinvertované vyčítání, nap. reference VDD

```
pinMode(AD1_pin, INPUT); // PullUp…

int AD1=analogRead(MUJ_pin);
```

2. Alternativní přístup – pomocí HAL knihovny – umožňuje nastavit další parametry.

*kanály ADC - viz obr. zapojení pinů!

- Vyčítá invertovaně (0V = max)
- Lze nastavit napěťovou referenci 1100 mV

```
#include <driver/adc.h>

  ...

adc1_config_width(ADC_WIDTH_BIT_12);
adc1_config_channel_atten(ADC1_CHANNEL_0,ADC_ATTEN_DB_0); // Vref 1100mV
  ...

int val = adc1_get_raw(ADC1_CHANNEL_0);
```

S ADC souvisí ještě integrovaný HAL senzor: (popravdě není nijak zázračně citlivý…)

```
#include <driver/adc.h>

  ...

adc1_config_width(ADC_WIDTH_BIT_12);
int val = hall_sensor_read();
```

- ***analogReadResolution(resolution)***: set the sample bits and resolution. It can be a value between 9 (0 – 511) and 12 bits (0 – 4095). Default is 12-bit resolution.
- ***analogSetWidth(width)***: set the sample bits and resolution. It can be a value between 9 (0 – 511) and 12 bits (0 – 4095). Default is 12-bit resolution.
- ***analogSetCycles(cycles)***: set the number of cycles per sample. Default is 8. Range: 1 to 255.
- ***analogSetSamples(samples)***: set the number of samples in the range. Default is 1 sample. It has an effect of increasing sensitivity.
- ***analogSetClockDiv(attenuation)***: set the divider for the ADC clock. Default is 1. Range: 1 to 255.
- ***analogSetAttenuation(attenuation)***: sets the input attenuation for all ADC pins. Default is ADC_11db. Accepted values:
  - ***ADC_0db***: sets no attenuation. ADC can measure up to approximately 800 mV (1V input = ADC reading of 1088).
  - ***ADC_2_5db***: The input voltage of ADC will be attenuated, extending the range of measurement to up to approx. 1100 mV. (1V input = ADC reading of 3722).
  - ***ADC_6db***: The input voltage of ADC will be attenuated, extending the range of measurement to up to approx. 1350 mV. (1V input = ADC reading of 3033).
  - ***ADC_11db***: The input voltage of ADC will be attenuated, extending the range of measurement to up to approx. 2600 mV. (1V input = ADC reading of 1575).
- ***analogSetPinAttenuation(pin, attenuation)***: sets the input attenuation for the specified pin. The default is ADC_11db. Attenuation values are the same from previous function.
- ***adcAttachPin(pin)***: Attach a pin to ADC (also clears any other analog mode that could be on). Returns TRUE or FALSE result.
- ***adcStart(pin)***, ***adcBusy(pin)*** and ***resultadcEnd(pin)***: starts an ADC convertion on attached pin's bus. Check if conversion on the pin's ADC bus is currently running (returns TRUE or FALSE). Get the result of the conversion: returns 16-bit integer.

DA převodník

- Na ESP32 jsou k dispozici dva 8 bit. DAC (0V..3.3V)
  - DAC1 (GPIO25)
  - DAC2 (GPIO26)

Příklad:

```
#define DAC1 25
int Value = 255; //255= 3.3V 128=1.65V
dacWrite(DAC1, Value);
```

Poznámka:

- Max. frekvence pily (0..255) v loop() bez dalších činností cca 700kHz….

PWM

ESP32 je vybaveno 16 nezávislými PWM kanály – neplést s GPIO piny.

- Kanály je možné přivést na každý GPIO.
- Každý kanál může být nastaven s jinými parametry!
-

Obecný postup nastavení:

1. Zvolte GPIO na kterém chcete používat PWM
   ```
   #define mujPWM_pin 5
   ```
2. Zvolte volný PWM kanál (0 až 15)
   ```
   const int PWMchannel = 0;
   ```
3. Následně je třeba nastavit frekvenci PWM v Hz
   Max. frekvence ESP32 pro 8b je cca 312kHz
   ```
   const int freq = 5000;
   ```
4. Nastavení rozlišení PWM (1 až 16b)
   ```
   const int resolution = 8;
   ```
5. Nastavte PWM
   ```
   ledcSetup(PWMchannel, freq, resolution);
   ```
6. Přiřaďte PWM kanál na zvolený GPIO
   ```
   ledcAttachPin(mujPWM_pin, PWMchannel);
   ```

7. Uvědomte si rozsah stříd pro dané rozlišení, např. pro 8b to budou hodnoty 0 až 255.
8. V algoritmu zapište aktuální střídu PWM.
   ```
   ledcWrite(PWMChannel, strida);
   ```

Odkazy:

- GPIO output: https://microcontrollerslab.com/gpio-pins-esp32-led-blinking-example/
- GPIO input: https://microcontrollerslab.com/push-button-esp32-gpio-digital-input/
- Ext. přerušení: https://lastminuteengineers.com/handling-esp32-gpio-interrupts-tutorial/
- ADC: https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/
- SAR teorie: https://en.wikipedia.org/wiki/Successive-approximation_ADC
- PWM: https://randomnerdtutorials.com/esp32-pwm-arduino-ide/

04 Aplikace mikroprocesorů 2 – EEPROM a WatchDog

Obsah

EEPROM

- Non-volatilní paměť - umožňuje uložit data (iniciační parametry, konstanty, poslední stav atd…) do trvalé (přepisovatelné paměti).
- Obsah zůstává zachován i po odpojení napájení.
- Může obsahovat např. linearizační konstanty AD převodníku.

K tématu prostudovat:

1. https://en.wikipedia.org/wiki/EPROM
2. https://en.wikipedia.org/wiki/EEPROM
3. https://en.wikipedia.org/wiki/Flash_memory

NOR-FLASH – umožňuje náhodný přístup

NAND-FLASH – přístup po stránkách

- EEPROM u ESP32 chybí!
- Emulována v NOR-FLASH paměti
  - Využívány instrukce pro self programming.
  - Omezený počet zápisů: > 10tis. zápisových cyklů (standard ke 100 tis.), klesá s narůstající tepl.!
    - To zase není tak mnoho – pozor na chyby v programu!
    - Pro zájemce – hledejte klíčová slova „Flash Endurance Testing“ https://hypnocube.com/2014/11/flash-endurance-testing/

Praktické použití

- Nutná knihovna `#include <EEPROM.h>`
- Emulována max. velikost 512B
- Inicializace EEPROM: `EEPROM.begin(EEPROM_SIZE);`

- Zápis: `EEPROM.write(address, value);//adresa 0 .. 511, hodnota 0 .. 255`
  - Lze zapsat více hodnot v cyklu…
  - Následně nutno skutečně zapsat pomocí: `EEPROM.commit();`
- Vyčtení: `int EEPROM.read(address);`

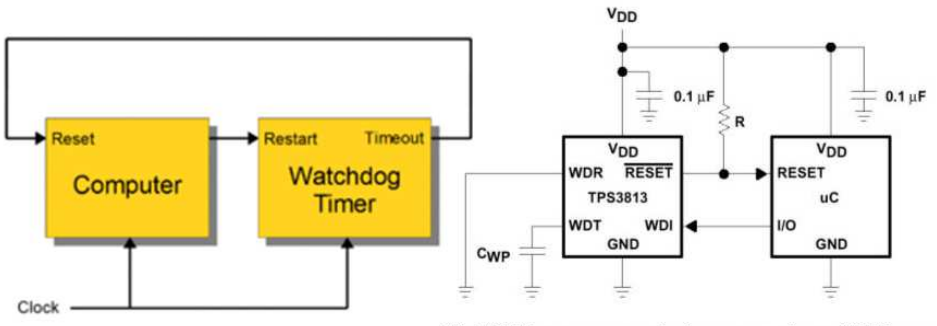
Příklad:

```
// include library to read and write from flash memory
      #include <EEPROM.h>


// define the number of bytes you want to access
      //ESP32 eeprom size = max 512B
      #define EEPROM_SIZE 1



void setup() {
  // put your setup code here, to run once:
```

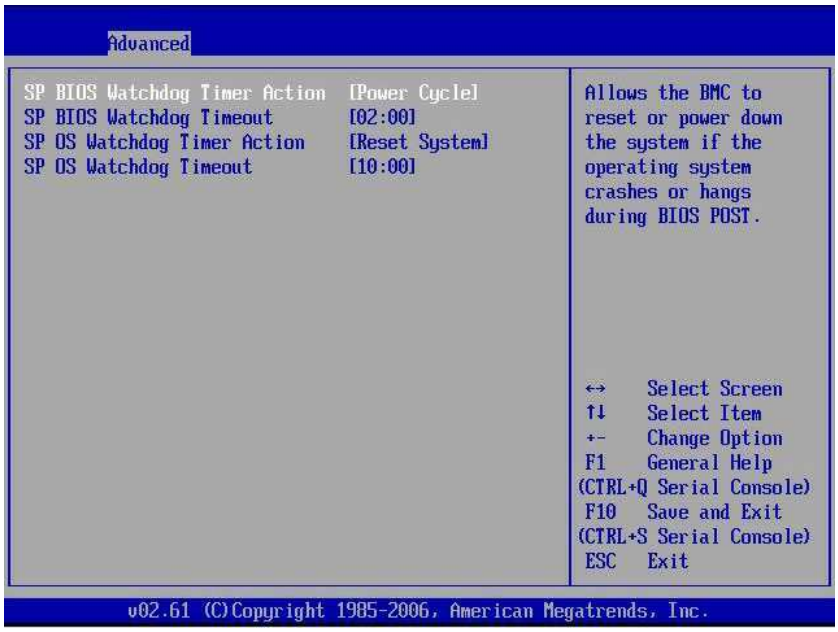
```
        Serial.begin(115200);
        Serial.println("Zapis/cteni EEPROM");

  // initialize EEPROM with predefined size
        EEPROM.begin(EEPROM_SIZE);
                /*
                  EEPROM.write(0, 123); //B on address 0
                  EEPROM.commit();      //nutno zavolat po zapsani
                  Serial.println("zapsano");
                */
  int a = EEPROM.read(0);
  Serial.println("Precteno");
  Serial.println(a);
}

void loop() {
 }
```

Watchdog

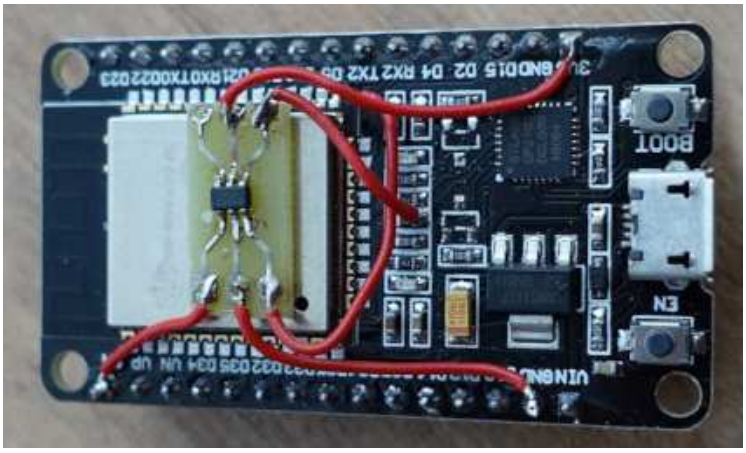
- Speciální čítač s možností resetu ESP32
- V mnoha případech - nutno hlídat správnou funkci systému.
- Obecná idea – pokud vše v pořádku – průběžně asynchronně nuluji volně běžící čítač.
- Když se řídící SW zasekne – čítač přeteče a vyvolá reset.
- Obvykle realizován hardwarově jako samostatný čítač s přístupem k reset obvodu.
- Nespoléhejte na WatchDog a ověřujte a analyzujte svůj kód!



IT's TPS3813 is easy to use and offers a nice windowing WDT feature.



Obrázek 1 – Watchdog není jen záležitost embedded zařízení – Watchdog u serveru



Obrázek 2 - Tak takhle ne!

Praktické použití

- Nutná knihovna `#include <esp_task_wdt.h>`
- Inicializace WatchDogu: `esp_err_t esp_task_wdt_init(WDT_TIMEOUT, true);`
  - `WDT_TIMEOUT` v sekundách (uint32_t .. cca 126 let max ;).
  - Panic mode – když WDT přeteče, vyvolá hardware panic and reboot
- Přidání WDT do aktuálního vlákna (viz dále): `esp_task_wdt_add(NULL);`
- Periodicky nulujeme WDT pomocí: `esp_task_wdt_reset();`
-

Příklad:

```
#include <esp_task_wdt.h>

#define WDT_TIMEOUT 30      //30 seconds WDT

void setup() {
```

```
…
    esp_task_wdt_init(WDT_TIMEOUT, true); //inicializace WDT
    esp_task_wdt_add(NULL); //aktivni pro toto vlakno
      //v pripade zalozeni jiného vlakna pridat do vlakna
}

void loop() {
  …
  delay(5000);
  esp_task_wdt_reset();      Serial.println("Resetting WDT...");
}
```

Odkazy

1. ESP32 EEPROM: https://randomnerdtutorials.com/esp32-flash-memory/
2. WatchDog: https://iotassistant.io/esp32/enable-hardware-watchdog-timer-esp32-arduino-ide/
3. WatchDog popis espressif: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/wdts.html

04 Aplikace mikroprocesorů 2 – Úsporné režimy

Obsah

Úsporné režimy

ESP32 má při použití všech periferií spotřebu až cca 250 mA, viz následující tabulka.

Table 10: RF Power-Consumption Specifications

Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	240	-	mA
Transmit 802.11b, OFDM 54 Mbps, POUT = +16 dBm	-	190	-	mA
Transmit 802.11g, OFDM MCS7, POUT = +14 dBm	-	180	-	mA
Receive 802.11b/g/n	-	95 ~ 100	-	mA
Transmit BT/BLE, POUT = 0 dBm	-	130	-	mA
Receive BT/BLE	-	95 ~ 100	-	mA

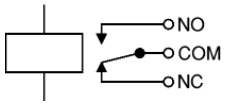
To může být značný problém při bateriovém napájení. Příklad:

1. **CR2032** 235mAh (2,8V) DC/DC (85%),
   - 3/3 x 0,85 x 235 / 25 (250mA trvale vůbec nelze!!!) = cca 8 h
   - 3/3 x 0,85 x 235 / 250 (pouze teoreticky, např. pulzně) = cca 45 min
2. **NiMh** článek AA 1000 mAh (1.25 V – 1.0 V ≈ 1.1 V), DC/DC měnič 85%, Vnap = 3,3 V
   - 1,1 / 3.3 x 0,85 x 1000 / 250 = cca 68 min
3. **Li-Ion** článek AA, 3400 mAh 3.6 V (4.2 V – 3.3 V ≈ 3.6 V) DC/DC měnič 85%, Vnap = 3,3 V
   - 3,6 / 3.3 x 0,85 x 3400 / 250 = cca 12h

Jak se dostat na provoz v řádu týdnů, měsíců…?

Spotřeba jádra je velmi nízká, nezapomínejte ale na běžné periferie, např.:

- jedna nízkopříkonová LED cca 2 – 5 mA
- jedna běžná LED cca 20 mA
- spínací cívka relé v sepnutém stavu cca 50 mA (vyplatí se používat rozpínací kontakty…)



Power mode	Description	Power consumption
Active (RF working)	Wi-Fi Tx packet 14 dBm ~ 19.5 dBm	Please refer to Table 10 for details.
	Wi-Fi / BT Tx packet 0 dBm	
	Wi-Fi / BT Rx and listening	
Modem-sleep	The CPU is powered on.	Max speed 240 MHz: 30 mA ~ 50 mA
		Normal speed 80 MHz: 20 mA ~ 25 mA
		Slow speed 2 MHz: 2 mA ~ 4 mA
Light-sleep	-	0.8 mA
Deep-sleep	The ULP co-processor is powered on.	150 μA
	ULP sensor-monitored pattern	100 μA @1% duty
	RTC timer + RTC memory	10 μA
Hibernation	RTC timer only	5 μA
Power off	CHIP_PU is set to low level, the chip is powered off	0.1 μA

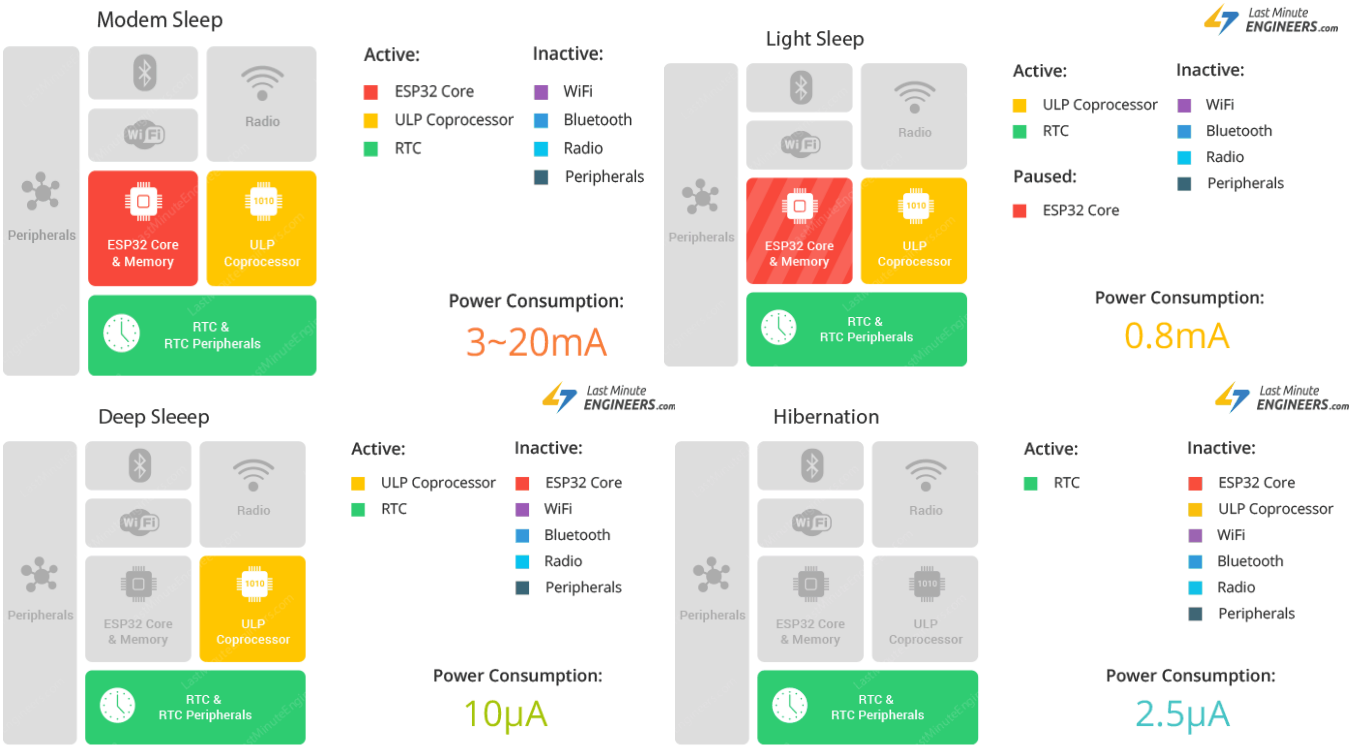
ESP32 umožňuje snížit provozní spotřebu několika způsoby:

1. Rozumným využitím – potřebujete v návrhu snímat teplotu skutečně 100 krát za 1s?!
2. Burst režim odesílání dat – navazování spojení BT/WiFi trvá dlouho – odešlete dávkově více dat najednou!
3. Deaktivací nepoužívaných periferií.
4. Přechodem do některého z **úsporných režimů**:
5. Využitím ULP[1] koprocesoru k řízení jednodušších činností.

Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	ON	PAUSE	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC memory and RTC peripherals	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

Úsporné režimy - využití

- Procesor a periferie jsou uspány – minimalizuje se spotřeba.
- Je však možné CPU probudit (RESET) jedním ze zdrojů:
  a. Ext. přerušení GPIO / Touch senzor
  b. RTC časovač
  c. ULP
  d. Sériový port (asi pouze Light-Sleep)



[1] ULP - *Ultra Low* Power coprocessor

Obecný postup přechodu ESP32 do úsporného režimu:

1. Detekce zdroje probuzení, ale není nutné.
2. Nastavení zdroje probuzení.
3. Výběr periferií, které budou v úsporném režimu pracovat. Ve výchozím nastavení jsou deaktivovány všechny periferie, které nejsou potřebné pro probuzení ESP32.
4. Uložení důležitých dat.
5. Zavolání funkce `esp_light_sleep_start();` nebo `esp_deep_sleep_start()`

Light sleep

V režimu Light sleep jsou vypnuty komunikační periferie a CPU je **zastaven hodinový signál**.

- Obsah paměti a registrů se zachovává.
- Přechod do Light Sleep pomocí volání funkce `esp_light_sleep_start();`
- Probuzení (pokračování práce CPU v přerušeném stavu) shodné jako v režimu Deep Sleep, ale s rozdílnými následky – viz dále!

Deep Sleep

Důležité – **probuzení** (přechod z úsporných režimů) **je provedeno RESETem**!!

- Obsah RAM a registrů je ztracen!

Toto je třeba v kódu zohlednit (uložit data, odeslat, využít EPROM apod.)

RTC SRAM

Pro uložení kritických dat je možné použít RTC SRAM.

- ESP32 má 8 KiB SRAM (tzv. RTC fast memory), kterou je možné použít pro uchování dat.
- Paměť není smazána během DeepSleep.
- Je ale smazána při HW resetu!!!

Uložení dat do RTC fast sram pomocí makra:

```
RTC_DATA_ATTR int PocetRestartu = 0;
```

Detekce zdroje RESETu

V kódu je možné implementovat detekci důvodu (zdroje RESET) :

```
#include <rom/rtc.h>
void print_reset_reason(RESET_REASON reason)
{
  switch ( reason)
  {
    case 1 : Serial.println ("POWERON_RESET");break;          /**<1, Vbat power on reset*/
    case 3 : Serial.println ("SW_RESET");break;               /**<3, Software reset digital core*/
    case 4 : Serial.println ("OWDT_RESET");break;             /**<4, Legacy watch dog reset digital core*/
    case 5 : Serial.println ("DEEPSLEEP_RESET");break;        /**<5, Deep Sleep reset digital core*/
    case 6 : Serial.println ("SDIO_RESET");break;             /**<6, Reset by SLC module, reset digital core*/
    case 7 : Serial.println ("TG0WDT_SYS_RESET");break;       /**<7, Timer Group0 Watch dog reset digital core*/
```

```
    case 8 : Serial.println ("TG1WDT_SYS_RESET");break;       /**<8, Timer Group1 Watch dog reset digital core*/
    case 9 : Serial.println ("RTCWDT_SYS_RESET");break;       /**<9, RTC Watch dog Reset digital core*/
    case 10 : Serial.println ("INTRUSION_RESET");break;       /**<10, Instrusion tested to reset CPU*/
    case 11 : Serial.println ("TGWDT_CPU_RESET");break;       /**<11, Time Group reset CPU*/
    case 12 : Serial.println ("SW_CPU_RESET");break;          /**<12, Software reset CPU*/
    case 13 : Serial.println ("RTCWDT_CPU_RESET");break;      /**<13, RTC Watch dog Reset CPU*/
    case 14 : Serial.println ("EXT_CPU_RESET");break;         /**<14, for APP CPU, reseted by PRO CPU*/
    case 15 : Serial.println ("RTCWDT_BROWN_OUT_RESET");break;/**<15, Reset when the vdd voltage is not stable*/
    case 16 : Serial.println ("RTCWDT_RTC_RESET");break;      /**<16, RTC Watch dog reset digi. core and rtc module*/
    default : Serial.println ("NO_MEAN");
  }
}
...
print_reset_reason(rtc_get_reset_reason(0));     //for CPU 0
```

Či pouze rozlišit zdroj probuzení ze sleep režimu:

```
// Method to print the reason by which ESP32 has been awaken from sleep
void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;
          wakeup_reason = esp_sleep_get_wakeup_cause();
  switch(wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason); break;
  }
}
```

Zdroj probuzení – čítač

- Vhodný, pokud potřebujete periodicky např. několikrát za den odebrat vzorek.
- Nastavení: `esp_sleep_enable_timer_wakeup(time_in_us) // cas  v us`

Ukázka kódu:

```
#include <rom/rtc.h>

#define uS_TO_S_FACTOR 1000000  /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP  5        /* Time ESP32 will go to sleep (in 5 seconds) */
...
print_reset_reason(rtc_get_reset_reason(0));     //for CPU 0
print_wakeup_reason();
...
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
...
Serial.println("Going to sleep now");
```

```
delay(1000);
Serial.flush();
esp_deep_sleep_start();
Serial.println("This will never be printed");
```

Zdroj probuzení – Touch sensor

ESP32 má 10 interních kapacitních dotykových senzorů. Tyto sensory mají široké využití a mohou např. reagovat na vodivost lidské pokožky. Tyto sensory mohou být snadno využity místo mechanických spínačů:

- T0 (GPIO 4)
- T1 (GPIO 0)
- T2 (GPIO 2)
- T3 (GPIO 15)
- T4 (GPIO 13)
- T5 (GPIO 12)
- T6 (GPIO 14)
- T7 (GPIO 27)
- T8 (GPIO 33)
- T9 (GPIO 32)

Detekce zdroje:

```
void print_wakeup_touchpad(){
  touch_pad_t touchPin = esp_sleep_get_touchpad_wakeup_status();
  switch(touchPin) {
    case 0  : Serial.println("Touch detected on GPIO 4"); break;
    case 1  : Serial.println("Touch detected on GPIO 0"); break;
    case 2  : Serial.println("Touch detected on GPIO 2"); break;
    case 3  : Serial.println("Touch detected on GPIO 15"); break;
    case 4  : Serial.println("Touch detected on GPIO 13"); break;
    case 5  : Serial.println("Touch detected on GPIO 12"); break;
    case 6  : Serial.println("Touch detected on GPIO 14"); break;
    case 7  : Serial.println("Touch detected on GPIO 27"); break;
    case 8  : Serial.println("Touch detected on GPIO 33"); break;
    case 9  : Serial.println("Touch detected on GPIO 32"); break;
    default : Serial.println("Wakeup not by touchpad"); break;
  }
}
```

Nastavení zdroje probuzení např. na T3:

```
#define Threshold 40 /* Lower readed value lead to wakeup */
//Setup interrupt on Touch Pad 3 (GPIO15)
  touchAttachInterrupt(T3, Mycallback_function, Threshold);
//Configure Touchpad as wakeup source
  esp_sleep_enable_touchpad_wakeup();

...
Serial.println("Going to sleep now");
delay(1000);
Serial.flush();
esp_deep_sleep_start();
Serial.println("This will never be printed");
```

Zdroj probuzení – Externí přerušení

- Dva zdroje ext. přerušení:
  1. EXT0 – Je možné použít pouze jeden?? RTC GPIO
     - RTC periferie, RTC paměť a RTC Controller musí být On i v DeepSleepu.
  2. EXT1 – Může být využito několik RTC GPIO najednou.
     - RTC periferie a RTC paměť může být vypnuta. RTC Controller musí být On i v DeepSleepu.
     - Pokud ale použijeme PullUp, nebo PullDown, tak i RTC periferie On!

EXT0

registrace zdroje přerušení: `esp_sleep_enable_ext0_wakeup(GPIO_NUM_X, level);`

příklad: `esp_sleep_enable_ext0_wakeup(GPIO_NUM_33, Low);`

EXT1

registrace zdroje přerušení: `esp_sleep_enable_ext1_wakeup(bitmask, mode);`

mode: `ESP_EXT1_WAKEUP_ALL_LOW` nebo `ESP_EXT1_WAKEUP_ANY_HIGH`

příklad:

```
#define BUTTON_PIN_BITMASK 0x200000000 // 2^33 in hex, = GPIO33
esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK, ESP_EXT1_WAKEUP_ANY_HIGH);
```

Identifikace zdroje přerušení: `uint64_t esp_sleep_get_ext1_wakeup_status(void);`

- Funkce vrací číslo o základu 2, tedy 2^(GPIO_NUMBER). Pro získání dekadického vyjádření můžeme např. provést: GPIO = log(GPIO_NUMBER)/log(2)…¨
- I v případě specifikace skupiny pinů, např. 2^15 + 2^33 nám bude log2 stačit – probudí stejně pouze první z nich!

Zdroj probuzení – Sériový port (pouze pro zájemce)

Thought, there's also another very interesting and useful way of waking up our ESP32, and it's on receiving serial data. Serial interrupt seems to be working only on light sleep, according to the API doc for the ESP32, but… what if we wanted to wake it up from deep sleep?

A solution I came up, which works for both deep and light sleeps, is:

1. First of all, the Rx serial pin is not an RTC pin (none of the other serials, either!) so, attaching directly to it an interrupt won't work…
2. so, wire up (directly) from SERIAL_RX_PIN (=GPIO_NUM_3) to any free RTC GPIO, for example, GPIO_27.
3. We cannot do "esp_sleep_enable_ext1_wakeup(MULTIPLE_INT_BITMASK, ESP_EXT1_WAKEUP_ANY_HIGH);" because SERIAL_RX_PIN pin is always HIGH when idle (and its signal is passed to our GPIO_27 because wired), so it will awake the ESP32 continuously.

4. We cannot use “esp_sleep_enable_ext1_wakeup(MULTIPLE_INT_BITMASK, ESP_EXT1_WAKEUP_ALL_LOW);” because we will probably want to awake the ESP32 from different independent sources, and not when having ALL of them LOW.
5. Solution: we can do “esp_sleep_enable_ext0_wakeup(GPIO_NUM_27, LOW);” and also having it combined with “esp_sleep_enable_ext1_wakeup(MULTIPLE_INT_BITMASK, ESP_EXT1_WAKEUP_ANY_HIGH);” in case we have also other interrupt sources. This will awake our ESP32 from deep_sleep on receiving any serial input.
6. Careful:
    - Only RTC IO can be used as a source for external wake source: they are pins: 0,2,4,12-15,25-27,32-39
    - ext0 uses RTC_IO to wake up thus requires RTC peripherals; to be on while ext1 uses RTC Controller so doesn’t need peripherals to be powered on.
    - Note that using internal pullups/pulldowns also requires RTC peripherals to be turned on: this makes “esp_sleep_enable_touchpad_wakeup()” to NOT work! (docs.espressif.com/projects/esp-idf/en/latest/api-reference/system/sleep_modes.html#_CPPv332esp_sleep_enable_touchpad_wakeupv), throwing an exception: “E (151) sleep: Conflicting wake-up trigger: ext0”
    - So, if there are to use capacitive-pin interrupts in addtion to ext0 or ext1 interrupts, don’t use this. Instead, put all interrupts as ext1 masked.
    - If we need to use touch_interrupts, ext0 must NOT to be used.
    - We can NOT use instead ext1 again, defining a second macro ONLY for our copied RTC pin for SERIAL_RX_PIN, like: “esp_sleep_enable_ext1_wakeup(SERIAL_INT_BITMASK, ESP_EXT1_WAKEUP_ALL_LOW);”, because further definitions of ext1 will overwrite prior ones… same applies to ext0.

Odkazy

1. Úsporné režimy: https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/

06 Aplikace mikroprocesorů 2 - Vlákna

Obsah

Vlákna

K čemu používat vlákna?

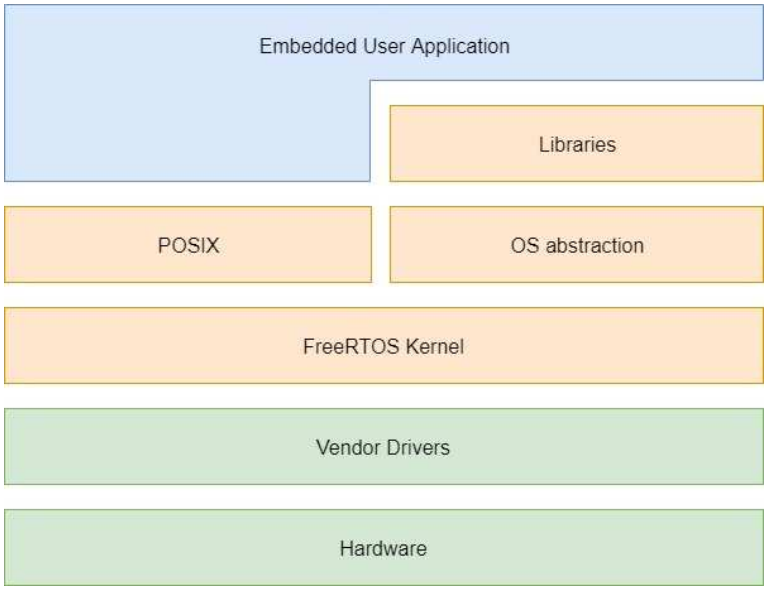
- Díky použití vláknového programování dochází ke zpřehlednění kódu.
- Usnadní se řešení některých asynchronních typů úloh.
- v případě vícejádrových CPU jsme schopni využít plný výkonový potenciál.
- Mnoho úloh lze efektivně paralelizovat (šifrování, FT, komprese obrazu…).
- Lze snadno oddělit např. část kódu starající se o interakci s uživatelem (GUI) a vlastní výpočet.
- SW projekt se snadněji dekomponuje a rozdělí mezi více spolupracujících tvůrců.

Na modulech ESP32 je předpřipraven operační systém FreeRTOS s podporou vláken.

- Je možné použít i další OS.
- FreeRTOS – realtime OS s podporou preemptivního multitaskingu.
- MIT open source license (https://opensource.org/licenses/MIT).

FreeRTOS vlastnosti:

- Minimalistické řešení (snadno portovatelný) – **v podstatě pouze správa procesů!**
- Umožňuje použít vlákna (i na jednojádrových CPU).
- Podporuje semafory, mutexy a SW časovače.
- U vláken podporuje priority.
- Nepodporuje ochranu paměti!
- V základu nepodporuje další periferie (Síť atd..) – vše ostatní pomocí dalších SW modulů.



ESP32 je postaven na **dvoujádrovém** Xtensa 32-bitovém LX6 mikroprocesoru.

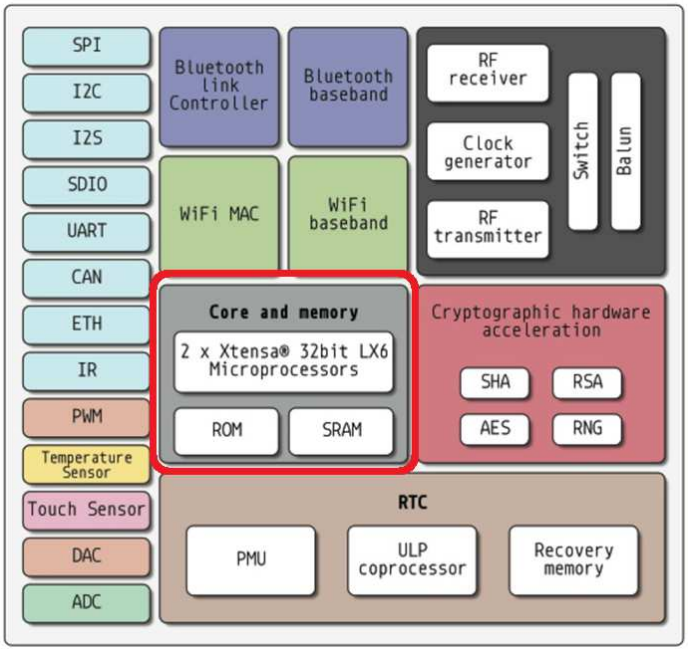
- Pomocí OS s plánovačem úloh (Taskmanager) je možné využívat obě jádra.
- Využití obou jader programem v čistém C (bez plánovače) není možné!

Máme tedy k dispozici:

- *Core-0*
- *Core-1*

FreeRTOS je na ESP32 předpřipraven – pokud nahrajeme kód pomocí ArduinoIDE, je po spuštění vytvořena smyčka *loop()*, která se automaticky stane jedním z již existujících vláken (a je spuštěna na Core-1).

Důležité – **vlákna mají sdílenou paměť!!!** (nejedná se o procesy!)



Nemusíme se starat o to, na kterém jádru je spuštěna, ale pokud by nás to zajímalo, je možné to zjistit voláním funkce *xPortGetCoreID()*.

Úkol (01)

- Vytvořte program, který bude na sériový port vypisovat, na kterém jádru právě běží smyčka *loop();*

Vytváření vláken

FreeRTOS umožňuje jednoduše vytvořit nové vlákno pomocí funkce *xTaskCreatePinnedToCore()*:

Nejprve je třeba deklarovat ukazatel vlákna TaskHandle_t Vlakynko1_hndl;

Následně je možné zavolat funkci *xTaskCreatePinnedToCore()* pro vytvoření vlákna:

```
xTaskCreatePinnedToCore(
    Vlakynko1_fce, /* Function to implement the task */
    "Vlakno1", /* Handel - Name of the task */
    10000,  /* Stack size in words */
    NULL,  /* Task input parameter */
    0,  /* Priority of the task */
```

```
        &Vlakynko1_hndl,  /* Task handle. */
        tskNO_AFFINITY); /* Core where the task should run */
```

- Funkci můžeme volat opakovaně a vytvářet další vlákna.
    - Přípustné, ale nikoliv vhodné je volání funkce z nově vytvořeného vlákna.

Vlastní vlákno následně specifikujeme v těle funkce:

```
void Vlakynko1_fce( void * parameter) {
 for(;;) {
  Serial.print("Vlakno1 on core:");
  Serial.println(xPortGetCoreID());
  delay(500);
 }
}
```

- Důležité – nezapomeňte, že po opuštění funkce *Vlakynko1_fce()* vlákno skončí!!
    - Chcete-li tedy nekonečný běh vlákna, musíte zajistit např. pomocí *while(1)*

Úkol (02)

- Vytvořte program, který vytvoří a spustí 2 vlákna a v každém bude vypisovat jádro, na kterém je vlákno spuštěno.

Správa vláken

- Každé vlákno vytvořené funkcí *xTaskCreatePinnedToCore()* je ihned po vytvoření (…chvíli trva!) ihned spuštěno.
- Vlákno je možné kdykoliv dočasně uspat funkcí *vTaskSuspend (Vlakynko1_hndl);*
- Vlákno je možné kdykoliv dočasně probudit funkcí *vTaskResume(Vlakynko1_hndl);*
- Vlákno je možné odstranit funkcí *vTaskDelete(Vlakynko1_hndl);*

Úkol (03)

- Doplňte předchozí program tak, že první vlákno spustíte ihned po vytvoření, ale 2. vlákno spustíte s časovým odstupem 3s.
- Následně obě vlákna po 10s mandatorně ukončete funkcí *vTaskDelete().*

Afinita vláken – spřažení vlákna s jádrem CPU

V současné době má ESP32 dvě jádra. Předchůdce ESP8266 obsahoval pouze jedno jádro. Na trhu je však i modul ESP32-S0WD, který také obsahuje pouze jedno jádro.

Navíc v budoucnu lze předpokládat, že se objeví vícejádroví nástupci.

- Z těchto důvodů není vhodné ve funkci *xTaskCreatePinnedToCore()* specifikovat konkrétní jádro, na kterém naše vlákno poběží – o přiřazení (afinitu) se stará taskmanager FreeRTOS.

```
xTaskCreatePinnedToCore(
    …
    tskNO_AFFINITY); /* Core where the task should run */
```

- Je však možné si afinitu programově vynutit specifikací jádra 0, nebo 1.

```
xTaskCreatePinnedToCore(
    …
    0); /* Core where the task should run */
```

Úkol (04)

- Pozměňte příklad 02 tak, aby vlákno 1 bylo spuštěno na jádru 0 a vlákno 2 na jádru 1.

Priorita vláken

Za normálních okolností jsou všechna vlákna vytvářena s prioritou 0 – Přiřazení na aktivní CPU čas je plánovačem řízeno rovnoměrně, tak aby se všechna vlákna vystřídala (u velkých CPU a graf. karet se tím zrovnoměrňuje i tepelné zatížení).

- Prioritu vláken je však možné změnit při vytváření vlákna ve funkci *xTaskCreatePinnedToCore()* případně za běhu vlákna voláním funkce *vTaskPrioritySet()* / *vTaskPriorityGet()*.
- Pro priority platí:
  - Rozsah 0 až (configMAX_PRIORITIES – 1)
    - configMAX_PRIORITIES je definován v FreeRTOSConfig.h[1]
  - Čím vyšší číslo, tím vyšší priorita.

Úkol (04a,b) - dodelat

- Z předchozího kódu vymažte všechna volání funkce *delay()* a vypisujte nějaké hlášení identifikující vlákno 1, vlákno 2 a hlavní smyčku *loop()*. Spusťte a pozorujte střídání vláken ve výpisu.
- Založte globální proměnnou *int A = 0;*
- Ve vlákně 1 ji inkrementujte a ve vlákně 2 ji dekrementujte.
- V hlavní smyčce vypisujte hodnotu.
- Změňte prioritu vlákna 1 na např. 2 a opakujte experiment.

[1] c:\Users\martin\AppData\Local\Arduino15\packages\esp32\hardware\esp32\1.0.4\tools\sdk\include\freertos\freertos\FreeRTOSConfig.h

Odkazy

1. ESPRESSIF popis funkcí FreeRTOS: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html
2. FreeRTOS Wiki: https://en.wikipedia.org/wiki/FreeRTOS
3. ESP32 tasks tutorial: https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/
4. Priorita vláken: https://exploreembedded.com/wiki/Changing_Priority_of_Tasks

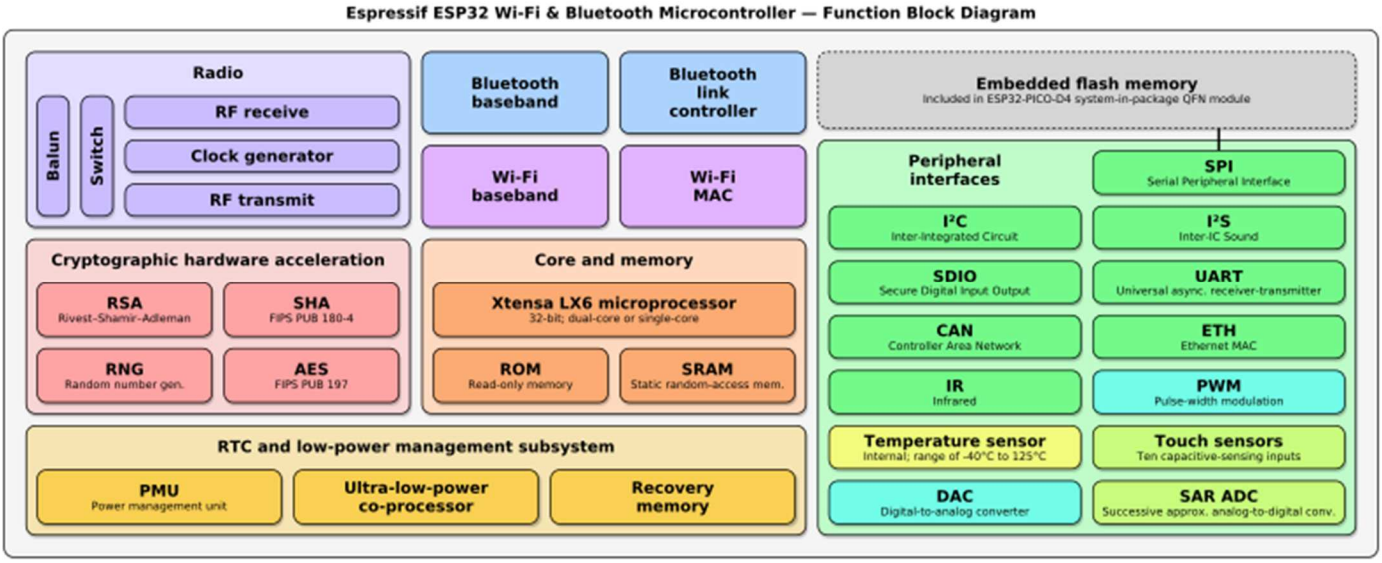
07 Aplikace mikroprocesorů 2 - WiFi

Obsah

Wifi a Bluetooth

Pro bezdrátovou komunikaci moduly ESP32 obsahuji implementaci Wifi b/g/n v pásmu 2.4 Ghz a Bluetooth.

- Verze Bluetooth (event. nepřítomnost) se liší dle verze modulu (např. ESP32-S2 BL neobsahují, ESP32-C3 mají v5.0 atd.)
- Na trhu je celá řada modifikací dalších výrobců s podporou dalších technologií (např. moduly Pycom s podporou LoRa, Sigfox).



Categories	Items	Specifications
Wi-Fi	Standards	FCC/CE/IC/TELEC/KCC/SRRC/NCC
	Protocols	802.11 b/g/n/d/e/i/k/r (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Frequency range	2.4 ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with -98 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC

Použití WiFi

- Pro použití bezdrátové komunikace je třeba vložit:

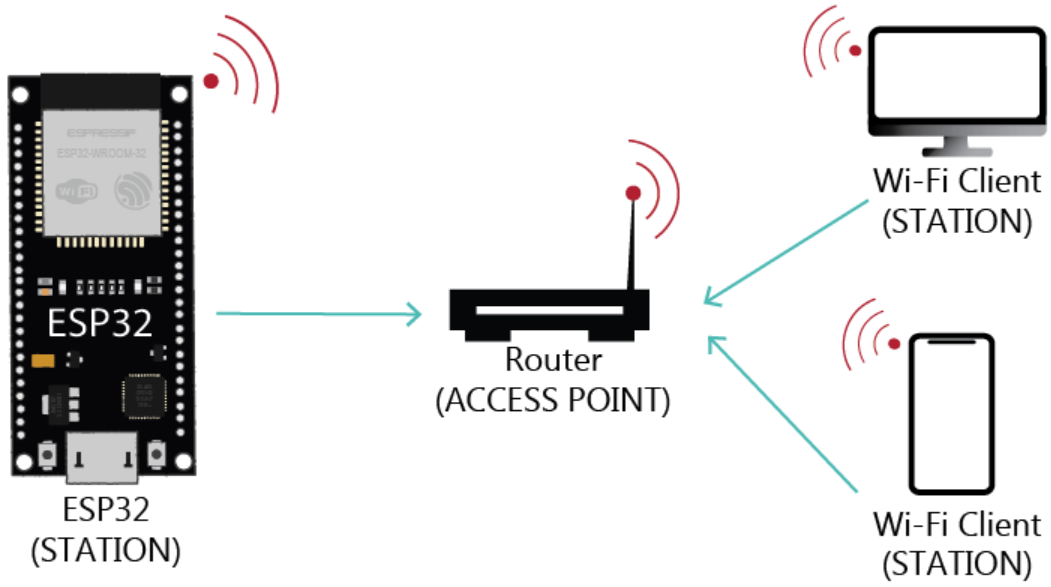
```
#include <WiFi.h>

#include "time.h"
```

- ESP32 může být v roli:

`WiFi.mode(WIFI_STA)`	station mode: the ESP32 connects to an access point
`WiFi.mode(WIFI_AP)`	access point mode: stations can connect to the ESP32
`WiFi.mode(WIFI_STA_AP)`	access point and a station connected to another access point

ESP32 v roli klienta již existující sítě

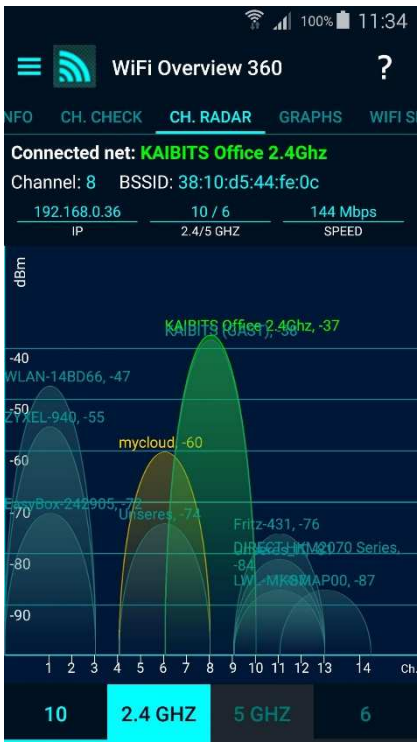
- Výchozí role je WIFI_STA – ESP32 se připojuje do již existující sítě WiFi na zvoleném kanále.



- Ve výchozím nastavení bude po připojení použit klient DHCPv4
- Po připojení bude v ESP32 plně funkční TCP/UDP/IPv4 stack.
- Autentizační mechanizmy a nastavení linkové vrstvy je provedeno ve většině případů automaticky, ale je možné jej změnit.

Scan okolních sítí: (01scan.ino)

Pokud potřebujeme zjistit viditelné sítě v bezprostředním okolí, je možné je proskenovat v celém rozsahu ISM 2,402 – *2,483* GHz.



- Pásmo 5 GHz zatím není podporováno.
- Tip: Nainstalujte na mobilu aplikaci Wifi Overview 360 a porovnejte výsledek následujícího algoritmu s výstupem aplikace.

```
// Set WiFi to station mode and disconnect from an
// AP if it was previously connected
WiFi.mode(WIFI_STA);
WiFi.disconnect();


int n = WiFi.scanNetworks();
  Serial.println("scan done");
```

```
  if (n == 0) {
      Serial.println("no networks found");
  } else {
    Serial.print(n);
    Serial.println(" networks found");
    for (int i = 0; i < n; ++i) {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.print(")");
      Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
      delay(10);
    }
  }
```

Připojení do existující WiFi – režim WIFI_STA (O2wificonnect.ino)

```
#include <WiFi.h>
#include "time.h"

const char* ssid       = "XXXXX";
const char* password   = "xxxxx";


void setup()
{
  Serial.begin(115200);

  //connect to WiFi
  Serial.printf("Connecting to %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {delay(500);Serial.print(".");}
  Serial.println(" CONNECTED");


  //disconnect WiFi as it's no longer needed
  /*
  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);
  */
}

void loop()
{
  delay(1000);
  Serial.println(WiFi.localIP());
}
```

- Otestujte dosažitelnost ESP32 ze svého PC/mobilu

Řešení problému s inicializací – navázáním spojení

- Chybka v ESP32 – u některých modulů se připojení povede pouze např. po následujícím restartu.

Nejprve vynutit ukončení:

```
WiFi.disconnect(true);
  while (WiFi.status() == WL_CONNECTED) {Serial.println("cekam na Wifi.disconnect");
delay(500);}
```

Poté znovu pokus o připojení s restartem:

```
int count =0;
while (WiFi.status() != WL_CONNECTED) {
  Serial.print("count=");Serial.println(count);
  delay(500);
  Serial.println(".");
  count++;
    if (count==3) ESP.restart();
}
```

NTP klient (03NTPclient.ino)

NTP (Network Time Protocol) je protokol pro synchronizaci vnitřních hodin počítačů po paketové síti s proměnným zpožděním.

Pro získání času přidáme do zdrojového kódu:

```
const char* ntpServer = "tik.cesnet.cz";
const long  gmtOffset_sec = 3600;
const int   daylightOffset_sec = 3600;


//init and get the time
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
printLocalTime();
```

a obsah funkce:

```
void printLocalTime()
{
  struct tm timeinfo;
  if(!getLocalTime(&timeinfo)){
    Serial.println("Failed to obtain time");
    return;
  }
  Serial.println(&timeinfo, "%d.%m.%Y %H:%M:%S"); //Friday, February 12 2021 10:53:03
              /*
                %a Abbreviated weekday name
                %A Full weekday name
```

```
                %b Abbreviated month name
                %B Full month name
                %c Date and time representation for your locale
                %d Day of month as a decimal number (01-31)
                %H Hour in 24-hour format (00-23)
                %I Hour in 12-hour format (01-12)
                %j Day of year as decimal number (001-366)
                %m Month as decimal number (01-12)
                %M Minute as decimal number (00-59)
                %p Current locale's A.M./P.M. indicator for 12-hour clock
                %S Second as decimal number (00-59)
                %U Week of year as decimal number,  Sunday as first day of week (00-51)
                %w Weekday as decimal number (0-6; Sunday is 0)
                %W Week of year as decimal number, Monday as first day of week (00-51)
                %x Date representation for current locale
                %X Time representation for current locale
                %y Year without century, as decimal number (00-99)
                %Y Year with century, as decimal number
                %z %Z Time-
zone name or abbreviation, (no characters if time zone is unknown)
                %% Percent sign
                You can include text literals (such as spaces and colons) to make a neater
 display or for padding between adjoining columns.
                You can suppress the display of leading zeroes  by using the "#" character
  (%#d, %#H, %#I, %#j, %#m, %#M, %#S, %#U, %#w, %#W, %#y, %#Y)
              */
}
```

Ping

```
      #include <ESP32Ping.h>


       Serial.print("Ping to google...");
          for(int i=0;i<3;i++){
            if(  Ping.ping("www.google.com", 1) ){ Serial.print("ok.."); } else Serial.pri
            ntln("!..");
          }
            float avg_time_ms = Ping.averageTime();

            Serial.println(avg_time_ms);
```

Eduroam

Pro připojení do sítě s WPA/WPA2 Enterprise je třeba upravit způsob asociace zařízení:

```
#include "esp_wpa2.h"
#include <WiFi.h>


const char* ssid = "eduroam"; // Eduroam SSID
const char* host = "cokoliv.cz"; //external server domain
```

```
#define EAP_IDENTITY "ramze0452@upce.cz" //identity@youruniversity.domain
#define EAP_PASSWORD "XXXXXXX" //your Eduroam password


WiFi.mode(WIFI_STA);
    esp_wifi_sta_wpa2_ent_set_identity((uint8_t *)EAP_IDENTITY, strlen(EAP_IDENTITY));
    esp_wifi_sta_wpa2_ent_set_username((uint8_t *)EAP_IDENTITY, strlen(EAP_IDENTITY));
    esp_wifi_sta_wpa2_ent_set_password((uint8_t *)EAP_PASSWORD, strlen(EAP_PASSWORD));
      WiFi.enableSTA(true);

  esp_wpa2_config_t config = WPA2_CONFIG_INIT_DEFAULT();
  if (esp_wifi_sta_wpa2_ent_enable(&config) != ESP_OK) {Serial.println("WPA2  Not OK");}

  WiFi.begin(ssid); //connect to Eduroam function
```

Odkazy

1. ESP32 Wifi: https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/
2. Popis NTP: https://cs.wikipedia.org/wiki/Network_Time_Protocol

07 Aplikace mikroprocesorů 2 - WiFi

Obsah

Wifi 2 - Nastavení statické IP (01staticIP.ino)

Pokud v síti není přítomen DHCP server, je možné použít tzv. statickou konfiguraci IP stacku:

1. Připojte ESP32 do své, již existující sítě
2. Pomocí aplikace IP tools zjistěte rozsah přidělovaných adres / volnou IP adresu.
3. Zjistěte výchozí bránu, masku sítě.
4. Nastavte staticky IP dle následujícího schématu:

```
IPAddress local_IP(192, 168, 1, 184);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 0, 0);
IPAddress primaryDNS(8, 8, 8, 8);
IPAddress secondaryDNS(8, 8, 4, 4);

if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
    Serial.println("Chyba konfigurace IP");
  }
```

5. Otestujte dosažitelnost ESP32 pomocí aplikace IP tools.

UDP odeslání zprávy (02UDPsend.ino)

ESP32 implementuje plný TCP/IP stack. Je tak možné pohodlně vysílat, nebo přijímat síťová data.

Pro odeslání a příjem zpráv však potřebujeme vhodnou protistranu. Vhodným nástrojem je například utilita NETCAT dostupná pro Linux i Windows, která umožňuje přijímat a odesílat UDP/TCP zprávy a vypisovat je na obrazovku.

Pro Windows stáhněte utilitu z: https://eternallybored.org/misc/netcat/

V OS Linux je většinou běžně k dispozici.

Je samozřejmě možné propojit i např. 2 ESP32…

Pro síťovou komunikaci existuje větší množství knihoven. Využijeme:

```
#include <WiFiUdp.h>
```

Postup odeslání dat:

```
WiFiUDP Udp;
      const char * udpAddress = "martindobrovolny.cz";
      //const char * udpAddress = "192.168.0.121";
      const int udpPort = 5678;
      uint8_t  zprava[] = "Nazdarek\r\n";
```

Inicializace Udp:

```
Udp.begin(2345);
```

Odeslání zprávy:

```
Udp.beginPacket(udpAddress, udpPort);
Udp.write(zprava,sizeof(zprava));
Udp.endPacket();
```

Na přijímacím PC spusťte naslouchání:

- Windows nc -u -l –p 5678
- Linux: nc -u -l 5678

UDP příjem a odeslání zprávy (03UDPrecieve.ino)

V následujícím příkladu obrátíme role. Zprávy budeme generovat na PC. ESP32 zprávu příjme a odpoví odesilateli.

Do předchozího příkladu přidejte:

```
unsigned int localPort = 2390;
char PrijataZprava_Buffer[255];
```

Změňte `Udp.begin(5555);` za `Udp.begin(localPort);`

Smyčku loop upravíme – budeme se cyklicky dotazovat, zda v příchozím buffru je validní zpráva a pokud ano, tak jí zpracujeme a odesilateli odešleme obsah naší zprávy:

```
int packetSize = Udp.parsePacket();
   if (packetSize) {
    Serial.print("Received packet of size ");   Serial.println(packetSize);
    Serial.print("From ");  IPAddress remoteIp = Udp.remoteIP();Serial.print(remoteIp);
    Serial.print(", port ");Serial.println(Udp.remotePort());


    //prevod zpravy na retezec zakonceny 0
    int len = Udp.read(PrijataZprava_Buffer, 255);
      if (len > 0) { PrijataZprava_Buffer[len] = 0;}
      Serial.println("Contents:");Serial.println(PrijataZprava_Buffer);


    //odesilateli odesleme odpoved
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
      Udp.write(zprava,sizeof(zprava));
    Udp.endPacket();

  }
```

Otestujte chování programu odesláním zprávy:

- Windows echo hello | nc -u 192.168.0.184 2390
- Linux: echo -n "hello" | nc -4u -q1 192.168.0.184 2390

Úkol (03UDPrecieve_b.ino)

- Modifikujte předchozí příklad tak, aby podle přijaté zprávy odpověděl různými hlášeními (vykonal různé činnosti)
- Pro vyhledání podřetězce v řetězci použijte funkci `strfind("AAA",PrijataZprava_Buffer)`

```
//nalezni retezec v podretezci, nuly na rozdil od strstr opravdu neporovnava
// size(retezce) musi byt >= size(sablony), size(sablony) musi byt nenulova
uint8_t strfind(char const S[30], char R[999])
{
    uint8_t n_r=strlen(R);
    uint8_t n_s=strlen(S);

    uint8_t n1=0;
    uint8_t n2=0;
    uint8_t shoda=0;

    if((n_r>=n_s) & (n_s>0)){
        for(n1=0;n1<n_r;n1++){
            shoda=1;
            for(n2=0;n2<n_s;n2++){
                if(S[n2]!=R[n2+n1]) shoda=0;
                if(shoda==0) break;        //sem urychlit zkrácení
            }
            if(shoda==1)
            break;
        }
    }
    return(shoda);
}
```

ESP32 v roli přístupového bodu (04AP.ino)

- V režimu *WiFi.mode(WIFI_AP)* vytvoří vlastní BSSID se zvolenými parametry.
- Ostatní zařízení se následně mohou asociovat a autentizovat.
    - Dále platí všechny postupy uvedené výše…

- Aby byl příklad zajímavější – spojíme jej s implementací jednoduchého WWW serveru.
    - Webový server by však samozřejmě fungoval i v režimu WIFI_STA

AP je na ESP32 vytvořen voláním funkce:

```
WiFi.softAP(ssid, password);
```

Případně

```
WiFi.softAP(const char* ssid, const char* password, int channel, int ssid_hidden, int max_
connection)
```

`ssid`: name for the access point – maximum of 63 characters;
`password`: **minimum of 8 characters**; set to `NULL` if you want the access point to be open;
`channel`: Wi-Fi channel number (1-13)
`ssid_hidden`: (0 = broadcast SSID, 1 = hide SSID)
`max_connection`: maximum simultaneous connected clients (1-4)

Přidejte do kódu:

```
const char* ssid     = "MojeESP32";
const char* password = "123456789";
```

AP vytvoříme voláním funkce:

```
WiFi.softAP(ssid, password);
```

AP můžeme nastavit statickou adresu[1]:

```
IPAddress IP = {10, 10, 1, 1};
IPAddress gateway = {10, 10, 1, 1};
IPAddress NMask = {255, 255, 255, 0};
WiFi.softAPConfig(IP, IP, NMask);
```

Aktuální adresu AP můžeme vypsat:

```
IPAddress  IP = WiFi.softAPIP();     //vypis pro kontrolu…
Serial.print("AP IP address: ");Serial.println(IP);
```

- Implementujte AP do ESP32, nahrajte kód a pomocí aplikace WiFi Overview 360 prozkoumejte parametry sítě.
- Pokuste se k síti připojit z mob. telefonu a zjistěte přidělenou IP adresu.
- Odešlete ICMP zprávu na ESP32 a ověřte komunikaci.

Implementace webserveru (05APwww.ino)

- Pro demonstraci chování ESP32 v režimu AP použijeme minimalistickou implementaci webového serveru se staticky zapsanou HTML stránkou.
- Webový server bude periodicky zjišťovat nové uživatele a pokud se nový klient připojí a pošle požadavek GET, tak mu zpátky odešle řetězec s obsahem HTML stránky.

Založte v deklaraci proměnných:

```
String header;
```

Deklarujte port, na kterém bude server naslouchat:

```
WiFiServer server(80);// Set web server port number to 80
```

Na konci funkce *setup()* server spusťte:

```
server.begin();
```

Přidejte do smyčky *loop()* následující kód:

```
WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                             // If a new client connects,
    Serial.println("New Client.");          // print a message out in the serial port
    String currentLine = "";                // make a String to hold incoming data from the client
```

[1] Nastavení DHCP pool viz: c:\Users\uzivatel\AppData\Local\Arduino15\packages\esp32\hardware\esp32\1.0.5\libraries\WiFi\src\WiFiAP.cpp

```
    while (client.connected()) {            // loop while the client's connected
      if (client.available()) {             // if there's bytes to read from the client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
        header += c;
        if (c == '\n') {                    // if the byte is a newline character
          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            client.println("<!DOCTYPE html><html>");

            client.println("<body><h1>ESP32 Web Server</h1>");

            client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
```

- Otestujte – připojte se z mobilního telefonu na síť ESP32 a webovým prohlížečem na IP adresu AP
- Pokud je vám známa syntaxe HTML, zkuste kód upravit

Implementace webserveru ovládajícího LED (06APwww_LEDblink.ino)

- Pomocí zaslání požadavku z HTML prohlížeče je možné ESP32 ovládat.
- V následující části rozšíříme výše implementovaný webserver o možnost blikání LED

Připojte ke GPIO 26 LED a ke GPIO 27 motorek s vrtulkou

Do kódu přidejte deklaraci potřebných proměnných:

```
String output26State = "off";
```

```
String output27State = "off";
const int output26 = 26;
const int output27 = 27;
```

Ve funkci *Setup()* proveďte základní nastavení GPIO:

```
pinMode(output26, OUTPUT);
pinMode(output27, OUTPUT);
digitalWrite(output26, LOW);
digitalWrite(output27, LOW);
```

Nahraďte obsah smyčky *loop()* následujícím kódem:

```
WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                             // If a new client connects,
    Serial.println("New Client.");          // print a message out in the serial port
    String currentLine = "";                // make a String to hold incoming data from the client
    while (client.connected()) {            // loop while the client's connected
      if (client.available()) {             // if there's bytes to read from the client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
        header += c;
        if (c == '\n') {                    // if the byte is a newline character
          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // turns the GPIOs on and off
            if (header.indexOf("GET /26/on") >= 0) {
              Serial.println("GPIO 26 on");
              output26State = "on";
              digitalWrite(output26, HIGH);
            } else if (header.indexOf("GET /26/off") >= 0) {
              Serial.println("GPIO 26 off");
              output26State = "off";
              digitalWrite(output26, LOW);
            } else if (header.indexOf("GET /27/on") >= 0) {
              Serial.println("GPIO 27 on");
              output27State = "on";
              digitalWrite(output27, HIGH);
            } else if (header.indexOf("GET /27/off") >= 0) {
              Serial.println("GPIO 27 off");
              output27State = "off";
              digitalWrite(output27, LOW);
            }
```

```
            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-size attributes to fit your preferences
            client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");
            client.println(".button { background-color: #4CAF50; border: none; color: white; padding: 16px 40px;");
            client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
            client.println(".button2 {background-color: #555555;}</style></head>");

            // Web Page Heading
            client.println("<body><h1>ESP32 Web Server</h1>");

            // Display current state, and ON/OFF buttons for GPIO 26
            client.println("<p>GPIO 26 - State " + output26State + "</p>");
            // If the output26State is off, it displays the ON button
            if (output26State=="off") {
              client.println("<p><a href=\"/26/on\"><button class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/26/off\"><button class=\"button button2\">OFF</button></a></p>");
            }

            // Display current state, and ON/OFF buttons for GPIO 27
            client.println("<p>GPIO 27 - State " + output27State + "</p>");
            // If the output27State is off, it displays the ON button
            if (output27State=="off") {
              client.println("<p><a href=\"/27/on\"><button class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/27/off\"><button class=\"button button2\">OFF</button></a></p>");
            }
            client.println("</body></html>");

            // The HTTP response ends with another blank line
            client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
```

```
}
```

- Znovu otestujte – připojte se z mobilního telefonu na síť ESP32 a webovým prohlížečem na IP adresu AP

Téma ESP32 & Wifi je takřka nevyčerpatelné. Hledejte další zajímavá témata:

- ESP32 Disconnect from Wifi
- ESP32 Automatic reconnect
- ESP32 TCP protocol
- Esp32 Get Wi-Fi Connection Status
- ESP32 Over The Air (OTA) Programming

Odkazy

1. Statická IP: https://randomnerdtutorials.com/esp32-static-fixed-ip-address-arduino-ide/
2. Arduino UDP: https://www.arduino.cc/en/Tutorial/WiFiSendReceiveUDPString/
3. Webserver: https://randomnerdtutorials.com/esp32-web-server-arduino-ide/