Dezvoltarea de aplicatii folosind interfata SPI

1 Modulul ESP 8266, interfata SPI

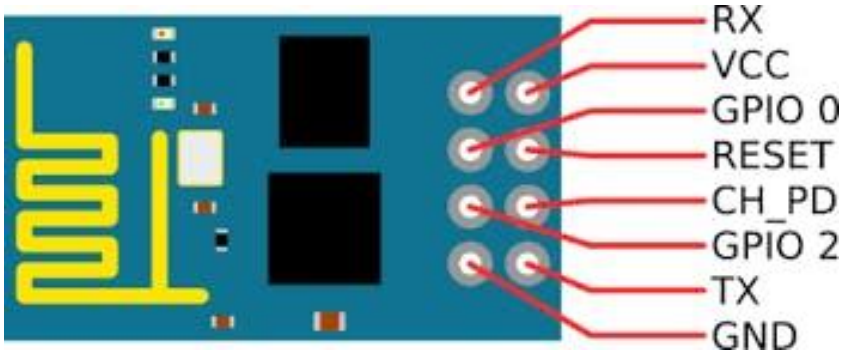
Modulul ESP 8266 este un „Sistem pe un Chip” (System on a Chip – Soc) compus dintr-un micro controller și un modul Wi-Fi de emisie-recepție. Acesta oferă soluții complete pentru comunicații și rețele Wi-Fi, permițând fie să se comporte ca “host” sau “client”. Pune la dispoziția utilizatorului pini de intrare/ieșire de uz general precum și diferiți alți pini cum ar fi pini de intrare analogică, dar configurațiile depinde de generația modulului. Acest modul poate fi programat ca orice alt microcontroler sau placă Arduino și are avantajul de a oferi comunicații Wi-Fi care deschide numeroase posibilități de dezvoltare de aplicații complexe.

Când modulul ESP stochează aplicația, acesta o poate rula direct de pe o memorie flash externă, iar memoria cache integrată îmbunătățește performanța sistemului pentru a minimiza cerințele de memorie. În mod alternativ, când modulul este folosit ca adaptor Wi-Fi, acesta permite adăugarea de conectivitate wireless oricărui microcontroler prin conexiuni simple de tip UART.

Capacitatea modulului ESP 8266 de procesare și stocare permite ca acesta să fie integrat alături de diversi senzori sau alte module prin intermediul pinilor de uz general (GPIO). Există diferite plăci de dezvoltare construite în jurul acestui modul cum ar fi NodeMCU DevKit sau WeMos D1 care pot oferi diferite configurări.

1.1 Specificații

Modulele ESP oferă capacități Wi-Fi, pini de intrare/ieșire de uz general, circuit inter-integrat (I2C), conversie analog-digitală pe 10 biți, interfață seriala SPI (detaliată în cele ce urmeaza), I2S, UART și modulație PWM. Configurația pinilor prezenți pe modulul ESP este următoarea:



Figură 1 Configurația pinilor ESP01

Pe placa ESP 01 găsim 8 pini, printre cei mai principali se enumeră:

- VCC care e folosit pentru alimentarea modulului. Voltajul maxim suportat este de 3.6 V, din acest motiv este nevoie de un regulator sau divizor de tensiune când se conectează la un modul Arduino care oferă tensiune de 5V.

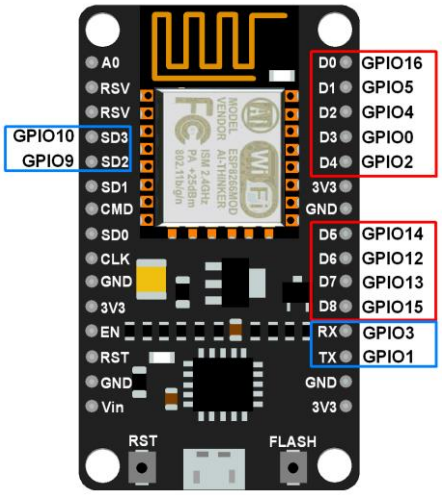
- GND este pinul de masă
- RX, TX sunt pinii folosiți pentru comunicare. Pot fi folosiți ca pini de uz general când nu se folosesc ca pini de comunicare.
- GPIO 0 și GPIO 2 sunt pini de uz general
- CH_PD
- RESET este folosit pentru resetarea modulului

Modulul ESP 8266 integrează în componență un Micro CPU pe 32 biți de consum ultra redus. Acest CPU se poate interfața folosind interfața RAM/ROM pentru a accesa memoria controller-ului, interfața AHB pentru accesul la registrii și cu interfața JTAG pentru diagnosticare (debugging). [1]

1.2 Interfețe disponibile

1.2.1 Pinii de intrare/ieșire de uz general

Pot exista, în funcție de generație, până la 16 pini de uz general. Fiecare pin poate fi configurat în numeroase posibilități. Acești pini sunt multiplexați cu alte funcții cum ar fi interfața host, UART, Bluetooth, etc. În figura 2 avem exemplificată placa de dezvoltare NodeMCU care e construită în jurul modulului ESP 8266 care oferă utilizatorului posibilitatea să folosească pana la 11 pini GPIO.



Figură 2 Exemplificarea pinilor GPIO pentru placa de dezvoltare NodeMCU

1.2.2 Pinii de intrare/ieșire digital

Suportul de intrare/ieșire digital este bidirecțional și ne-inversor. Acesta include un buffer de intrare și ieșire. Pentru operații de care necesită puțină putere, intrarea/ieșirea se poate seta pe hold. Funcționalitatea de hold introduce în suport un feedback pozitiv, din acest motiv driver-ul extern trebuie să fie mai puternic decât feedbackul pozitiv (aproximativ 5uA). Toți pinii de intrare/ieșire digitali sunt protejați de supra-tensiune printr-un circuit snap-back conectat între suport și masă. Acesta oferă protecție pentru supra-tensiune și ESD.

1.2.3 Interfața radio

Interfața radio este compusă din următoarele blocuri principale: Receptor și transmițător de 2.4GHz, generator de ceas, “Bias and regulators”,elemente de gestiune a puterii.

Receptorul de 2.4 GHz convertește semnalul radio în semnale de banda de bază în cuadratură care urmează să fie convertite în domeniul digital folosind două convertoare de rezoluție și viteză mare. Transmițătorul convertește semnalul digital în analog și îl transmite folosind o antenă controlata de un amplificator CMOS.

Frecvențele pe care operează componenta de emisie-recepție sunt următoarele canale, în concordanță cu standardele IEEE802.11bgn. [2]

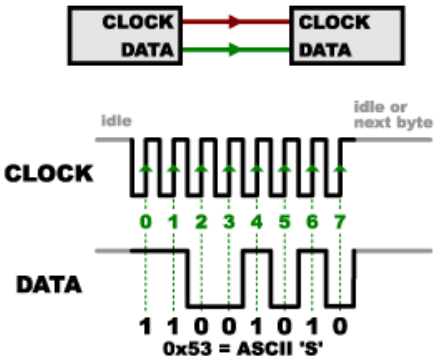
Numărul canalului	Frecvența (MHz)	Numărul canalului	Frecvența (MHz)
1	**2412**	**8**	**2447**
2	**2417**	**9**	**2452**
3	**2422**	**10**	**2457**
4	**2427**	**11**	**2462**
5	**2432**	**12**	**2467**
6	**2437**	**13**	**2472**
7	**2442**	**14**	**2484**

Figură 3 Tabelul cu frecvențele radio folosite de modulul ESP 8266

1.2.4 Interfața SPI

Interfața serială SPI este o interfață sincronă standard de mare viteză care operează în mod full duplex. Este folosită în principal pentru a trimite date între microcontrolere și periferice mici interconectate pe principiul master-slave. Interfața folosește linii separate pentru ceas și date, împreună cu o linie care permite să selectăm componenta cu care se dorește stabilirea comunicării.

SPI este o interfață sincronă care folosește linii de date separate pentru semnalul de ceas care permite sincronizarea celor două părți: transmițător și receptor. Semnalul de ceas este un semnal periodic care îi spune receptorului exact când să eșantioneze biții de pe linia de date. Aceasta se poate face pe palierul ascendent al semnalului de ceas sau palierul descendent. Când receptorul detectează palierul definit pentru care se face sincronizarea, se va uita imediat pe linia de data pentru a citi următorul bit. Pentru că semnalul de ceas este trimis deodată cu semnalul de date, viteza nu este importantă, cu toate că dispozitivele au o viteză maximă.

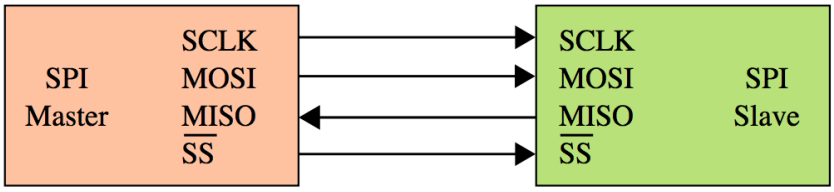


Figură 4 Transmitere date sincronă

Unul dintre motivele pentru care interfața serială SPI este populară, este pentru că receptorul poate fi un dispozitiv foarte simplu ceea ce reduce costurile și complexitatea circuitului.

Interfața SPI are patru semnale logice specifice:

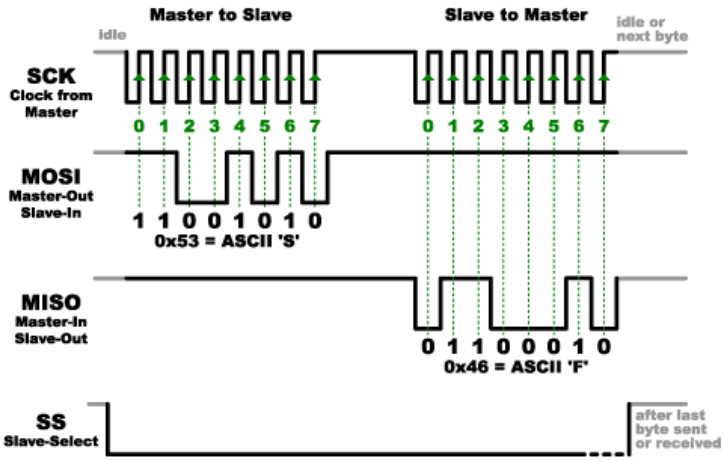
SCLK = semnalul de ceas ieșit din master,
MOSI/SIMO = Master Output, Slave Input,
MISO/SOMI = Master Input, Slave Output,
SS = Slave Select (activ pe low, ieșire din master)



Figură 5 Semnalele logice pentru interfața SPI

Pentru a începe comunicarea, dispozitivul master setează inițial frecvența ceasului ca fiind o frecvență cel mult egală cu maximul frecvenței suportată de dispozitivul slave. Aceste frecvențe sunt in intervalul 1-70 MHz. Semnalul de ceas este generat doar de master, iar în configurație poate exista un singur dispozitiv master (un microcontroler), dar mai multe dispozitive slave. Selectarea dispozitivului slave se face de către master punând low (0) pe linia de Slave Select (SS).

Când se trimit date de la master către slave, se trimite pe linia MOSI = Master Out/Slave In. Daca slave-ul dorește să trimită un răspuns înapoi spre master, masterul va continua sa genereze un număr pre aranjat de cicluri de ceas, iar slave-ul va pune datele de transmis pe linia MISO = Master In / Slave Out.



Figură 6 Exemplificare comunicații pentru interfața SPI

Pentru că masterul este singurul care generează semnal de ceas, trebuie să se știe dinainte când dispozitivul slave trebuie sa returneze date și dimensiunea datelor. Acest mod de transmitere a datelor este foarte diferit de modul asincron, unde puteau fi trimise dimensiuni

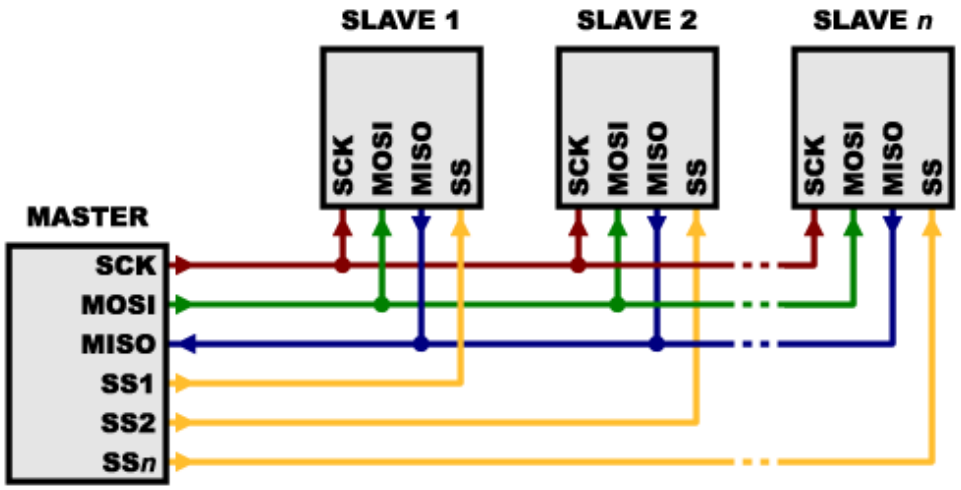
aleatoare a datelor în orice direcție la orice moment de timp. În practică, această metodă de transmitere a datelor nu este o problemă deoarece SPI se folosește, în general, la aplicații cu senzori sau dispozitive care trimit o structură fixă și foarte specifică a datelor.

De notat că interfața SPI este full-duplex, deci în unele situații se pot transmite și recepționa date în același timp (de exemplu, solicitând o nouă citire de la senzor în timp ce se recepționează citirea curentă).

Linia de date Slave Select (SS) controlează selectarea dispozitivelor slave cu care se dorește comunicarea. SS este de obicei ținut HIGH, metodă care deconectează dispozitivele slave de la magistrala SPI. Imediat înainte de trimiterea datelor către slave, linia de date SS este pusă pe LOW, care activează dispozitivul slave selectat. Când nu mai este nevoie de slave, linia SS se va trece HIGH.

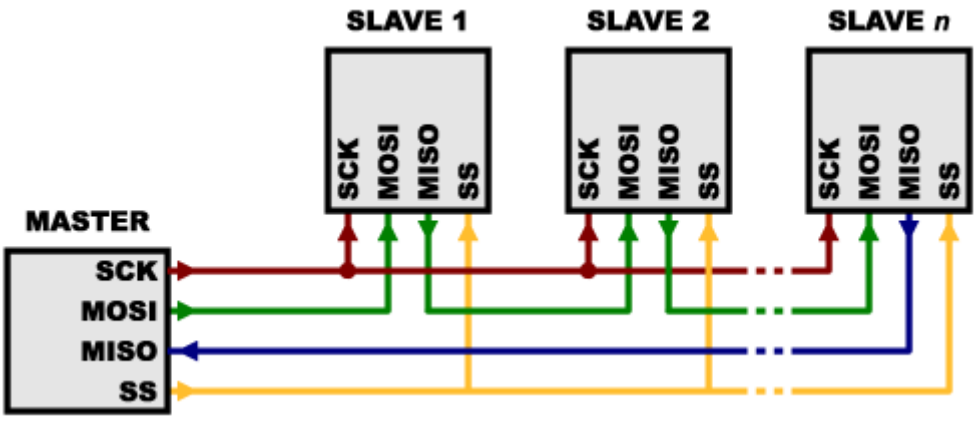
Există două posibilități de conectare a dispozitivelor slave.

1. În general fiecare slave are nevoie de o linie SS separată. Pentru a comunica cu un anumit slave, linia SS corespunzătoare va fi low iar restul high. Un număr mare de slave-uri va necesita un număr mare de linii SS respectiv de ieșiri de la master.



Figură 7 Conectare Slave-uri cu linie SS separată

2. O altă metodă de conectare este cu toate dispozitivele slave pe aceeași linie SS, iar MISO (ieșirea) de la unul este conectată la MOSI (intrarea) de la următorul. În momentul în care se transmit date, linia SS este activată ceea ce duce la activarea tuturor dispozitivelor slave. În această metode de conectare, datele trec dintr-un slave în următorul. Pentru a trimite date pentru un singur slave, trebuie transmise date în așa fel încât toate dispozitivele să le recepționeze. De reținut că primul pachet de date transmis va fi recepționat prima oară de ultimul slave. Acest layout este folosit în principal în aplicațiile în care există doar output din partea dispozitivului master și nu este necesar un răspuns din partea dispozitivelor slave (de exemplu aplicații de control pentru leduri). În cazul în care un răspuns este necesar, datele destinate dispozitivului master vor trece prin toate dispozitivele slave între sursă și destinație. [3]



Figură 8 Conectare Slave-uri pe aceeași linie SS

1.3 Programarea interfeței SPI

Majoritatea microcontroler-uri de pe piață au suport nativ pentru implementarea comunicațiilor folosind SPI. Acest protocol este destul de simplu astfel încât rutinele care manipulează liniile de intrare-ieșire sunt ușor programabile folosind documentația disponibilă de la producătorul microcontrolerului.

Modulul Arduino oferă două posibilități de programare a comunicațiilor SPI după cum urmează:

1. Folosind comenzile *shiftIn()* și *shiftOut()* care deplasează în interiorul unui octet de date câte un bit la un moment dat. Deplasările încep fie de la cel mai semnificativ bit fie de la cel mai puțin semnificativ. Pentru fiecare bit, pinul pentru ceas este high până când bitul este citit de pe linia de date, după care este trecut pe low. Aceste comenzi sunt bazate doar pe software și vor funcționa pentru orice grup de pini cu dezavantajul ca schimbul de date va fi oarecum lent.

Syntax

```
byte incoming = shiftIn(dataPin, clockPin, bitOrder)
```

Parameters

`dataPin`: the pin on which to input each bit (int)

`clockPin`: the pin to toggle to signal a read from **dataPin**

`bitOrder`: which order to shift in the bits; either **MSBFIRST** or **LSBFIRST**. (Most Significant Bit First, or, Least Significant Bit First)

Syntax

```
shiftOut(dataPin, clockPin, bitOrder, value)
```

Parameters

`dataPin`: the pin on which to output each bit (int)

`clockPin`: the pin to toggle once the dataPin has been set to the correct value (int)

`bitOrder`: which order to shift out the bits; either MSBFIRST or LSBFIRST. (Most Significant Bit First, or, Least Significant Bit First)

`value`: the data to shift out. (byte)

Figură 9 Sintaxa metodelor ShiftIn() și ShiftOut()

2. Se poate folosi librăria specifică oferită de Arduino numită *SPI Library*, care are se folosește de hardware-ul SPI integrat în microcontroler. Este mult mai rapidă decât metoda anterioară, dar se poate folosi doar pe anumiți pini specifici. [4]

1.4 Setarea interfeței SPI în Arduino

Înainte să putem folosi protocolul de comunicații SPI trebuie configurate câteva opțiuni esențiale. În funcție de componentele folosite, unele dintre aceste opțiuni s-ar putea să nu fie necesare.

Interfața SPI este capabilă să trimită datele începând cu cel mai semnificativ bit (MSB) sau cel mai puțin semnificativ bit (LSB). Această obține se setează din Arduino folosind metoda *setBitOrder().*

Dispozitivele slave vor citi datele fie pe palierul crescător a semnalului de ceas, fie pe palierul descrescător. În mod adițional semnalul de ceas poate fi considerat inactiv (idle) când este high sau low. Folosind librăria SPI, aceste opțiuni se setează prin intermediul funcției *setDataMode().*

Interfața SPI poate opera la viteze foarte mari (de ordinul milioanelor de octeți pe secundă), viteză care s-ar putea să fie mult prea mare pentru unele dispozitive. Pentru a asigura corectitudinea transmiterii datelor se poate ajusta viteza de transmisie. Prin intermediul funcției *setClockDivider()* se împarte frecvența semnalului de ceas (de obicei 16MHz) la o frecvență cuprinsă în intervalul 8MHz (frecvența ceasului/2) și 125 kHz (frecvența ceasului/128)

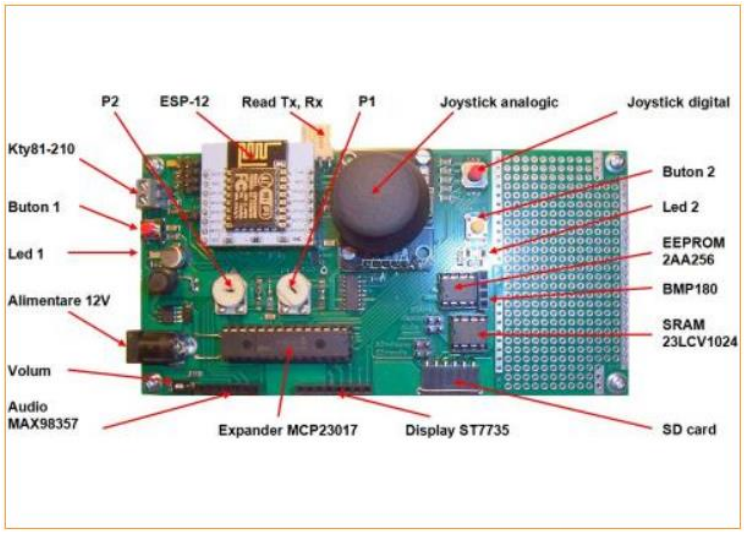
Pentru librăria disponibilă este nevoie de a specifica pinii de ceas (SCK), MOSI și MISO, în felul în care sunt configurați pe plăcuță, conform fișei de date (datasheet). În dispozitivele Arduino mai vechi este nevoie ca pinul SS sa fie controlat manual, setând pinul SS low înaintea unui transfer și high imediat după. Versiunile mai noi fac acest lucru automat facând parte din transferul de date. [5]

1.5 SPI vs I2C vs UART

SPI	I2C	UART
Interfata seriala sincrona	Interfata seriala sincrona	Interfata seriala asincrona
Single-master,multi-slave	Multi-master,multi-slave	Single-master,single-slave
Transferul serial de I/O se face pe 4 fire	Transferul serial de I/O se face pe 2 fire	Transferul serial de I/O se face pe 2 fire
Viteza mare	Viteza mica	Viteza mica
Mod full-duplex	Mod half-duplex	Mod simplex, full/half-duplex
Interfata hardware simpla	Interfata hardware mai simpla decat la UART,dar mai complexa decat laSPI	Interfata hardware complexa

SPI vs I2C vs UART

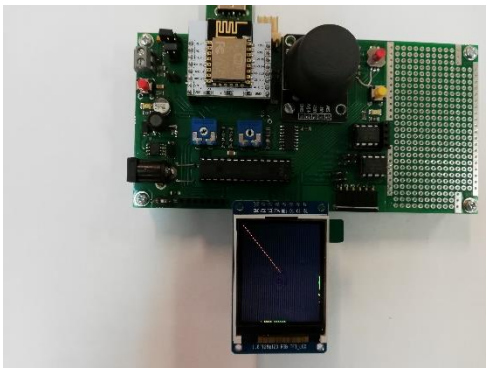
2 Aplicații

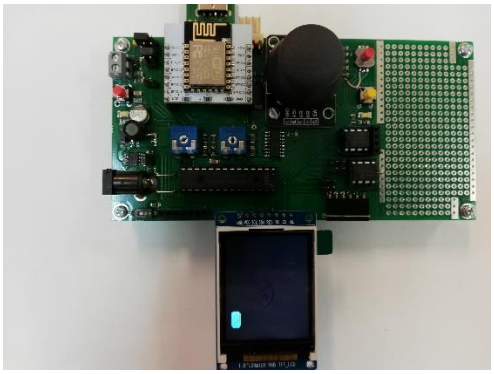


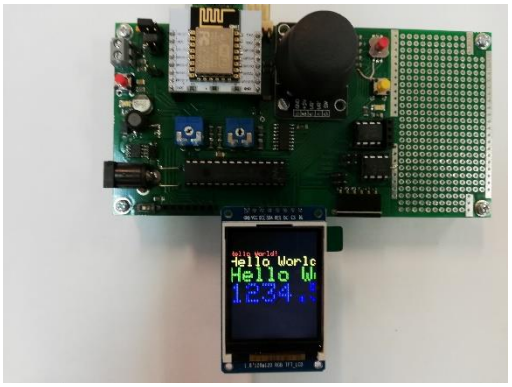
A. Placa de test utilizata in cadrul laboratoarelor, Wifi ESP-12, este echipata cu o memorie SRAM, 23LCV1024, cu o capacitate de 1 Mbiti (128Kocteti * 8 = 1024Kbiti). Adresarea memoriei se face la nivel de octet.

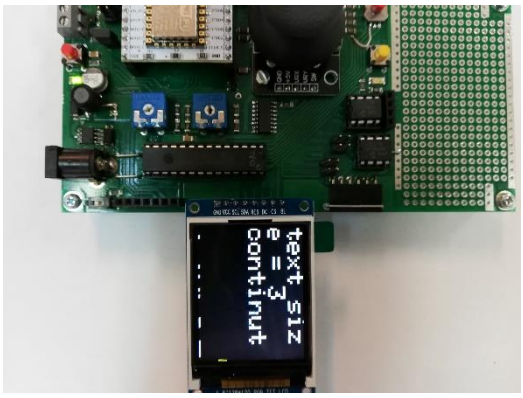
1. Aplicatia *spi_sram_wr_string.ino* scrie in memoria SRAM un sir de caractere ("Testing 0123456789") incepand cu o adresa specificata. Sirul este ulterior citit din memorie si afisat pe Serial Monitor. Adresa octetului in memoria SRAM unde incepe scrierea/citirea este setata ca fiind 138.
2. Aplicatia *spi_sram_test.ino* face testul de scriere / citire pe toata lungimea memoriei SRAM. La apăsarea butonului “BUTON_1” va începe testarea memoriei. Se începe verificarea la adresa zero a octetului de memorie, iar printr-o buclă de program se parcurge toată lungimea memoriei, scrie o valoare apoi verifică prin citire dacă valoarea a fost scrisă sau nu.
3. Aplicatia *spi_sram_wr_functions.ino* scrie respective citeste in/din memoria SRAM valori numerice de tip byte, int, long si float.

B. Pe placuta se conecteaza un modul LCD (ST7735) de 1.8" ce utilizează o comunicație SPI cu 4 pini și are o rezoluție de 128 x 120 pixeli. Ecranul său poate afișa culori pe 18 biți, ușor de utilizat cu orice tip de microprocesor. Studiati aplicatiile din directorul „Exemple LCD ST7735”. Acestea prezinta facilitatile grafice ale afisajului LCD, de rotire a textului, posibilitatea de modificare a fontului, etc.









3 Bibliografie

[1] „A Beginner's Guide to the ESP8266”, *Pieter P, 08-03-2017*
[2] „ESP8266 Technical Reference”, espressif.com
[3] „Serial Peripheral Interface (SPI)”, Sparkfun.com
[4] „ESP8266 Arduino core”, arduino.esp8266.com
[5] „Arduino SPI library”, arduino.cc
[6] „SPI Transactions between ESP8266 and Arduino UNO”, deeplyembedded.org
[7] „WiFi ESP8266 → utilizari noi, aplicatii si tutoriale”, acdcelectronics.ro
[8] „Everything ESP8266”, esp8266.com

Anexa 1

Aplicatia *spi_sram_test.ino* de citire/scriere pe toată lungimea memoriei

```
// Face testul de scriere citire pe toata lungimea memoriei.
// Verificarea se face doar at cand apasati BUTON_1.


// Placa test e echipata cu 23LCV1024, capacitatea = 128Kocteti * 8 = 1024Kbiti
// Pot fi folosite si chipuri 256 respectiv 512 Kbiti
// adresarea memoriei se face la nivel de octet


#include <Espl2TestBoard.h>     //definitii pentru butoane joystick, chip select etc



// ------------- Declara extensie memorie SRAM cu 23LCV1024 --------------------------

// https://github.com/dmason1992/SpiRam_Extended
#include <SpiRAMexp.h>


// Selectati situatia reala !
//#define AUDIO_EXISTS            // Audio MAX9835 exista pe placa !
//#define DISPLAY_EXISTS          // Display ST7735 exista pe placa !


#ifdef AUDIO_EXISTS
  #ifdef DISPLAY_EXISTS
    boolean IsCsSRAMgpio = false;
    const int CS_SRAM = CS_SRAM_EXP;   // -> NU se pune jumper -> SRAM lucreaza cu chip select expandat!
  #else
    boolean IsCsSRAMgpio = true;
    const int CS_SRAM = CS_SRAM_AUDIO;   // -> puneti jumper "SRAM+audio"
  #endif
#else
  #ifdef DISPLAY_EXISTS
    boolean IsCsSRAMgpio = true;
    const int CS_SRAM = CS_SRAM_LCD;     // -> puneti jumper "SRAM+display"
  #else
    boolean IsCsSRAMgpio = true;
    // ambele dispozitive lipsesc, alegeti o varianta:
    const int CS_SRAM = CS_SRAM_AUDIO;   // -> puneti jumper "SRAM+audio"
    //const int CS_SRAM = CS_SRAM_LCD;     // -> puneti jumper "SRAM+display"
  #endif
#endif

// declara marimea memoriei SRAM in Kbiti:
// 256  -> 24K256
// 512  -> 24LCV512
// 1024 -> 24LCV1024

SpiRAMexp SpiRam(CS_SRAM, 1024, IsCsSRAMgpio);
// -------------------------------------------------------------------------------
```

```
unsigned int mem_adr = 0;      // adresa octetului in memoria SRAM unde incepe verificarea

void setup() {
  Serial.begin(115200);
  delay(500);
}

void loop() {
  unsigned int address = 0;
  unsigned int maxRam = 32768;
  byte wValue, rValue;

  for (byte i = 0; i < maxRam; i++) {
    wValue = (byte) (i & 0xFF);

    // cu butonul apasat scrie indexul "i" pe care-l va citi corect
    if (expander.digitalRead(BUTON_1) == LOW) {
      SpiRam.write_byte(i, wValue);
    }
    // cu butonul liber scrie 0
    else {
      SpiRam.write_byte(i, 0);
    }

    rValue = (byte)SpiRam.read_byte(i);
    if (wValue != rValue) {
      Serial.print("RAM error at ");
      Serial.print(i, HEX);
      Serial.print(" should be ");
      Serial.print(wValue, HEX);
      Serial.print(" is ");
      Serial.println(rValue, HEX);
    } else {
      Serial.println(i, HEX);
    }
    delay(300);
  }
}
```