Laboratorul 5: Convertorul analog-digital. I2C. LCD grafic color

Capitole utile din Datasheet ATmega324

- Pin Configurations
  - secțiunea 1.1 - pag. 2
- ADC - Analog-to-digital converter
  - secțiunile 23.1-23.2 - pg. 243
  - secțiunea 23.4 - pg. 245
  - secțiunea 23.5 - pg. 246
  - secțiunea 23.8 - pg. 256
  - secțiunile 23.9.1-23.9.3 - pg. 258

Capitolele sunt din Datasheet ATmega324, document care îl aveți și pe desktop-ul calculatorului din laborator. Nu garantăm aceeași ordine a capitolelor în cazul utilizării altui document!

Acest laborator are ca scop familiarizarea voastră cu lucrul cu convertorul analog-digital prezent în microcontroller-ul Atmega324. De asemenea, veți vedea cum se poate lucra cu un modul de LCD grafic color (ST7735R).

Măsurarea semnalelor analogice

Pentru a putea măsura semnalele analogice într-un sistem de calcul digital, acestea trebuie convertite în valori numerice discrete. Un *convertor analog – digital* (ADC) este un circuit electronic care convertește o tensiune analogică de la intrare într-o valoare digitală.

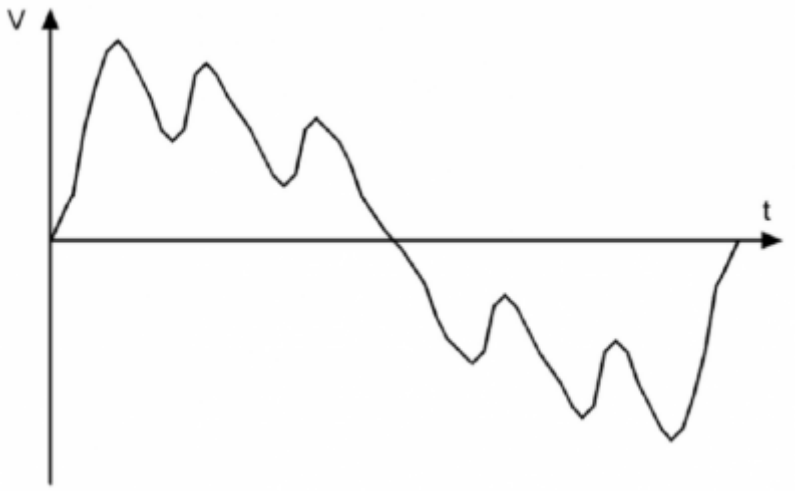


Fig. 1: Reprezentarea unui semnal analogic

O caracteristică importantă a unui ADC o constituie **rezoluția** acestuia. Rezoluția indică numărul de valori discrete pe care convertorul poate să le furnizeze la ieșirea sa în intervalul de măsură. Deoarece rezultatele conversiei sunt stocate intern sub formă binară, rezoluția unui convertor analog-digital este exprimată în biți.

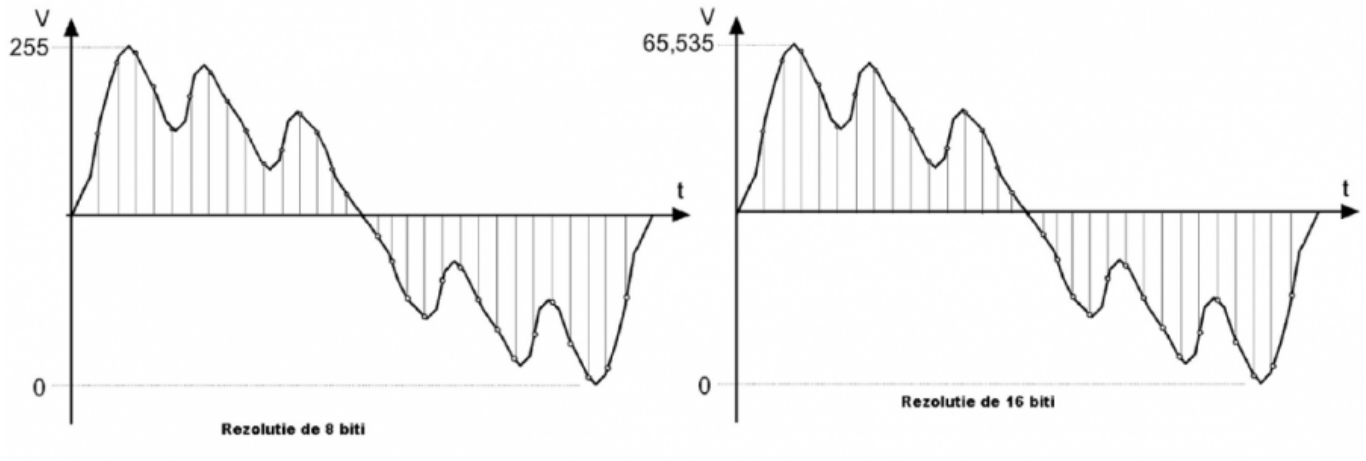


Fig. 2: Același semnal analogic eșantionat la rezoluții diferite

De exemplu, dacă rezoluția unui convertor este de 10 biți atunci el poate furniza $2^{10} = 1024$ valori diferite la ieșire. Dacă gama de măsurare este de 0-5V, rezoluția de măsurare va fi: $\frac{5V-0V}{1024} = 0.0048V = 4.8mV$.

O altă caracteristică importantă a unui convertor analog-digital o constituie **rata de eșantionare**. Aceasta depinde de timpul dintre două conversii succesive și afectează modul în care forma de undă originală va fi redată după procesarea digitală. figure 2 arată modul în care semnalul eșantionat în figure 1 va fi reconstituit în urma trecerii printr-un convertor digital – analog (DAC). După cum se poate observa, semnalul reprodus nu este identic cu cel original. **Dacă rata de eșantionare ar crește semnalul reprodus aproximează din ce în ce mai bine originalul**.

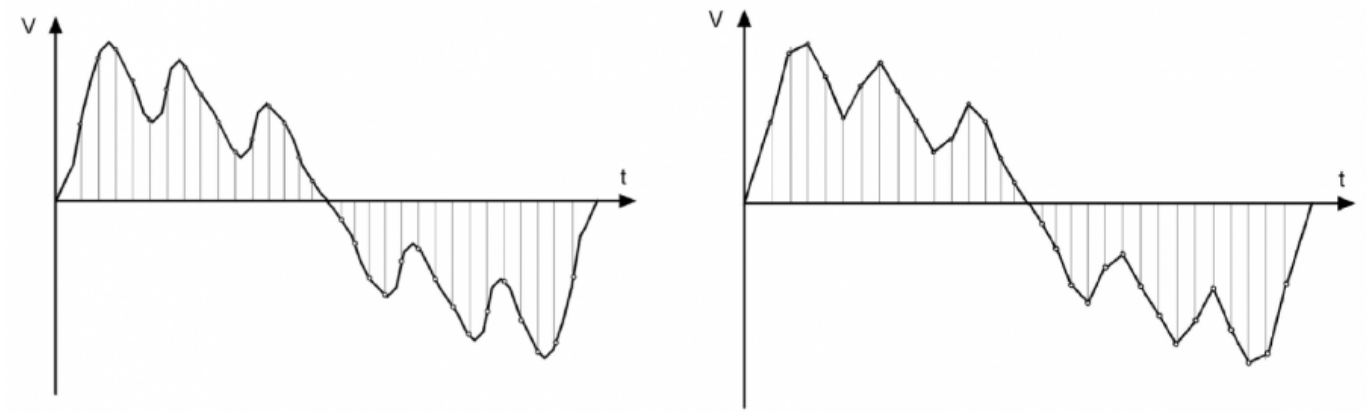


Fig. 3: Semnalul analogic refăcut în urma conversiei inverse (DAC)

Care este însa rata minimă de eșantionare pentru a reproduce fără pierderi un semnal de o frecvența data? Teorema lui Nyquist spune că o rată de eșantionare de minim două ori mai mare decât frecvența semnalului măsurat este necesară pentru acest lucru, teorema aplicându-se și pentru un semnal compus dintr-un intreg spectru de frecvențe, cum ar fi vocea umană sau o melodie. Limitele maxime ale auzului uman sunt 20Hz – 20kHz dar frecvențele obișnuite pentru voce sunt în gama 20-4000Hz, de aceea centralele telefonice folosesc o rată de eșantionare a semnalului de 8000Hz. Rezultatul este o reproducere inteligibilă a vocii umane, suficientă pentru transmiterea de informații

într-o convorbire obișnuită. Pentru reproducerea fidelă a spectrului audibil se recurge la rate mai mari de eșantionare. De exemplu, înregistrarea pe un CD are o rată de eșantionare de 44100Hz ceea ce este mai mult decât suficient pntru reproducerea fidelă a tuturor frecvențelor audibile.

In funcție de modul în care se execută conversia, convertoarele analog-digitale pot fi de mai multe **tipuri** :

- ADC paralel (Flash)
- ADC cu aproximare succesivă
- ADC cu integrare (single-slope, dual-slope);
- ADC Sigma-delta (delta-sigma, 1-bit ADC sau ADC cu oversampling).

Convertorul ADC al Atmega324

Convertorul analog-digital inclus în microcontroller-ul Atmega324 este un ADC cu aproximări succesive. Are o rezoluție de pana la **10 biți** și poate măsura orice tensiune din gama 0-5V de pe **opt intrări analogice** multiplexate. Dacă semnalul de la intrare este prea mic în amplitudine, convertorul are facilitatea de preamplificare a acestuia in două setări, de 10x sau de 200x.

Acest convertor poate fi controlat printr-un registru de stare și control (ADCSRA) și un registru cu biți de selecție pentru multiplexoare (ADMUX). În primul dintre ele, putem stabili când să se efectueze conversia, dacă să se genereze întrerupere la finalul unei conversii etc. Folosind registrul pentru multiplexoare se alege ce canal va genera input pentru convertor si tensiunea de referință. De asemenea, în afară de aceste două registre mai avem registrul ADC în care este scris rezultatul conversiei (ADC).

Registrele necesare pentru configurarea convertorului sunt următoarele:

ADMUX - ADC Multiplexer Selection Register

Mai multe detalii puteți să găsiți la secțiunea 23.9 - Register description din datasheet.

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 4: Registrul ADMUX

- REFS1 și REFS0 selectează referința de tensiune, adică valoarea maximă care poate fi convertită. Utilizatorul poate alege dintre două referințe interne, de 1.1V și 2.56V, sau două referințe externe furnizate de pinii AVCC (egală cu tensiunea de alimentare) sau AREF (orice tensiune între 0V și 5V).
- ADLAR selectează modul de aliniere a celor 10 biți ai rezultatului în registrul ADC de 16 biți (left-aligned sau right-aligned).
- MUX4..0 selectează intrarea de pe care se face conversia. Se poate alege și un mod de functionare

diferențial, caz în care se va măsura diferența de tensiune între cei doi pini aleși.

ADCSRA - ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 5: Registrul ADCSRA

- ADEN este bitul de Enable și activează convertorul.
- ADSC: scrierea acestui bit pe 1 va genera automat o conversie de pe intrarea selectată.
- ADATE este bitul de Enable pentru trigger-mode în care o conversie este generată pe frontul pozitiv al unui semnal furnizat extern de către utilizator.
- ADIF este bitul de întrerupere și va fi setat automat la terminarea unei conversii.
- ADIE este bitul de Enable pentru întrerupere. Aceasta va fi generată automat la fiecare conversie dacă bitul respectiv este 1.
- ADPS2..0 selectează rata de eșantionare pentru convertor. Aceasta poate fi de maxim fclk/2.

ADC - ADC Data Register

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Fig. 6: Registrul de date ADC

În funcție de valoarea bitului ADLAR, rezultatul conversiei va fi pus în registrul ADC aliniat la stânga sau dreapta.

Relația dintre valoarea din registrul ADC și tensiunea măsurată este următoarea:

$ADC = \frac{V_{in} \cdot 1024}{V_{ref}}$, unde *Vin* este tensiunea măsurată iar *Vref* este tensiunea aleasă ca referință.

Exemple de utilizare

- Inițializare ADC: canal 1 (PA1):

```
ADMUX = 0;
ADMUX |= (1 << MUX0);           // ADC1 - channel 1
ADMUX |= (1 << REFS0);          // AVCC with external capacitor at AREF pin
ADCSRA = 0;
ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); //set clock prescaler
at 128
ADCSRA |= (1 << ADEN);          // enable ADC
```

- Citire ADC prin polling:

```
ADCSRA |= (1 << ADSC);          //start conversion
loop_until_bit_is_set(ADCSRA, ADIF);  //wait until conversion is done: until
ADIF from ADCSRA is set to 1
unsigned int value = ADC;
```

Senzori Analogici: Temperatura si Lumina

Pe placa de laborator avem conectați la ADC-ul microcontroller-ului 2 senzori analogici, unul de temperatură și unul de lumină. Aceștia sunt conectați la canale ADC diferite:

- ADC0 (temperatură)
- ADC1 (lumină)

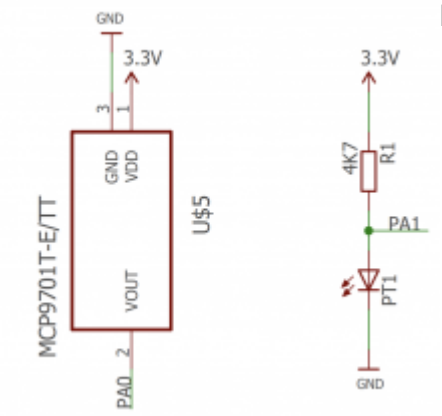


Fig. 7: Senzori de temperatură și lumină

Pe lângă acești doi senzori analogici mai avem conectate la canalul ADC5 6 butoane (BTN1 - BTN6) prin intermediul unei rețele rezistive, ca în figura de mai jos.

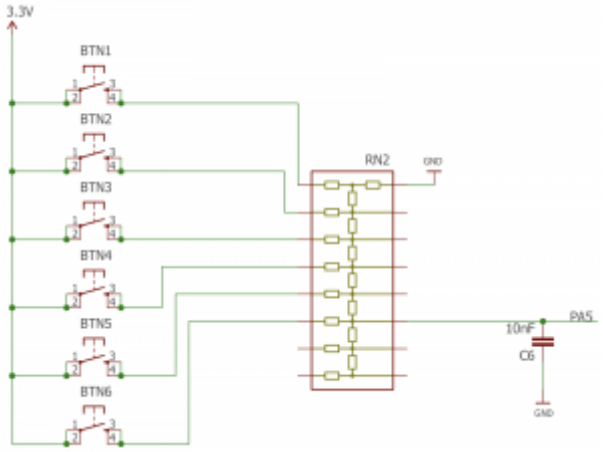


Fig. 8: Butoane/Rețea rezistivă

I2C

Protocolul I2C (sau IIC - Inter-Integrated Circuit) este un protocol de comunicație serială sincron, multi-master - multi-slave, dezvoltat de către Phillips în anul 1982. O magistrală I2C este formată din următoarele semnale:

- SDA = linia de date
- SCL = semnalul de clock

Semnalul de ceas este generat de către master iar linia de date este controlată atât de master cât și de către slave. La un moment dat un singur dispozitiv de pe magistrală poate controla linia de date. Din acestă cauză protocolul I2C este half-duplex.

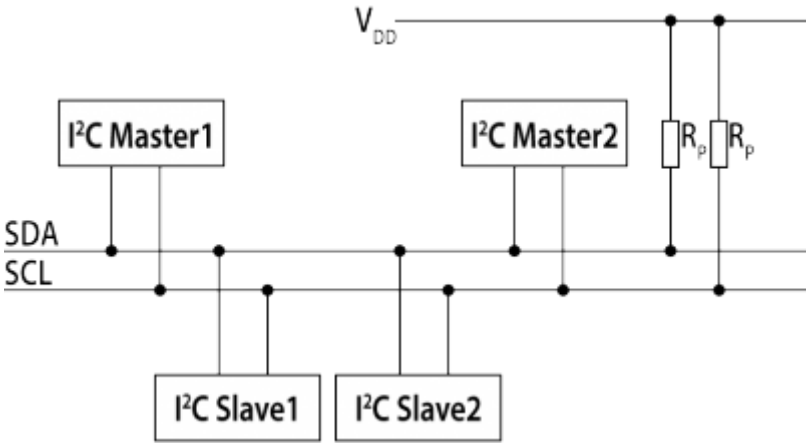


Fig. 9: Magistrală I2C multi-master - multi-slave

Modul de funcționare

Față de SPI unde master-ul activa, prin intermediul semnalului de *Slave Select*, dispozitivul cu care dorea să comunice, I2C nu necesită un asemenea semnal adițional. Protocolul I2C introduce noțiunea de *Slave Address*. Adresa unui dispozitiv de tip slave este un număr pe 7 biți (cel mai comun), pe 8 biți sau 10 biți. Comunicația dintre un master și un slave se face prin mesaje și este tot timpul inițiată de către master. Aceste mesaje pot fi sparte în două tipuri de cadre:

- un cadru de adresă
- unul sau mai multe cadre de date

Aceste cadre sunt interschimbate numai după ce master-ul a trimis *condiția de start*. Sfârșitul unui mesaj este identificat prin *condiția de stop*.

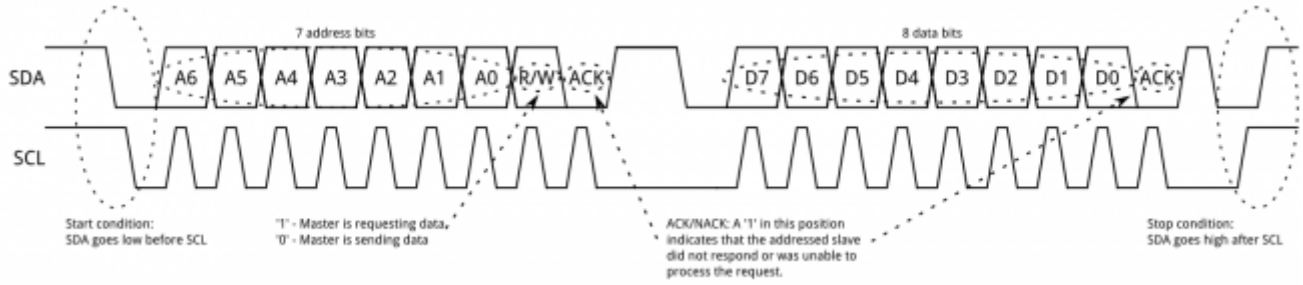


Fig. 10: Exemplu tranzacție I2C

1. Condiția de start

Înainte ca master-ul să trimită pe linia de date adresa slave-ului cu care dorește să comunice, acesta trebui sa genereze o condiție de start. Condiția de start determină toate dispozitivele slave să ”asculte” linia de date pentru că va urma o adresă. Pentru a genera această condiție, master-ul lasă linia SCL in HIGH și pune linia SDA pe LOW.

2. Cadrul de adresă

După ce masterul a generat condiția de start, acesta trimite pe linia de date (SDA) adresa dispozitivului slave cu care dorește să comunice. Adresa este (de cele mai multe ori) un număr pe 7 biți (biții A6-A0 din frame-ul din figura figure 10). Bitul 0 menționează dacă master-ul inițiază o operație de Citire (bitul 0 este 1) sau o operație de Scriere (bitul 0 este 0), așa cum se poate observa în figura figure 10.

Slave-ul care își recunoaște adresa trimite un ACK master-ului prin punerea liniei SDA pe LOW în al nouălea ciclu de ceas. Starea default a liniilor SDA/SCL este HIGH datorită rezistențelor de pull-up. Master-ul/Slave-ul doar ”trag” liniile pe LOW.

Master-ul identifică dacă a primit ACK (SDA pus pe LOW) sau NACK (SDA a rămas HIGH pe durata celui de-al nouălea ciclu de ceas).

3. Cadrele de date

Dacă master-ul a primit ACK (dacă există un slave pe magistrală cu adresa respectivă), el poate continua cu transmiterea datelor (operație de scriere), sau cu recepția datelor (operație de citire). Numărul de cadre de date este arbitrar, pot fi interschimbate oricâte. Fiecare cadru trimis/recepționat este ACK'd sau NACK'd. În funcție de operație (citire sau scriere), ACK-ul/NACK-ul este trimis fie de master fie de slave.

- Daca master-ul a inițiat o operație de scriere, fiecare cadru trimis este confirmat (ACK'd) de către slave.
- Daca master-ul a inițiat o operatie de citire, fiecare cadru recepționat este confirmat de (ACK'd) de master. Când master-ul dorește să oprească tranzacția după ce un anumit număr de cadre a fost recepționat, în loc să trimtă ACK trimite NACK. Astfel slave-ul se va opri din transmitere.

4. Condiția de stop

După ce toate cadrele de date au fost interschimbate, master-ul generează condiția de stop. Aceasta este realizată prin eliberarea linie SDA (trecere din LOW în HIGH) **după** eliberarea liniei SCL (trecere din LOW în HIGH).

Regiștrii configurare I2C

Atmega324p poate funcționa atât în modul I2C Master cât și I2C Slave.

I2C mai este cunoscut și cu numele de TWI (Two-Wire Interface).

- TWBR - TWI Bit Rate Register (datasheet 21.9.1. - pagina 235)
- TWCR - TWI Control Register (datasheet 21.9.2. - pagina 235)
- TWSR - TWI Status Register (datasheet 21.9.3. - pagina 237)
- TWDR - TWI Data Register (datasheet 21.9.4. - pagina 237)

Bibliotecă I2C Master - API

În acest laborator vom utiliza o bibliotecă pentru a ușura lucrul cu dispozitivele I2C. API-ul acesteia poate fi văzut în headerul ”i2c_master.h” din scheletul de laborator.

Click pentru API I2C Master

```
/* Initializeaza I2C. */
void    I2C_init(void);

/* Trimite conditia de start (adresa slave + bit read/write. */
uint8_t I2C_start(uint8_t address);

/* Trimite un cadru de date. */
uint8_t I2C_write(uint8_t data);

/* Returneaza cadrul de date primit de la slave si trimite ACK acestuia. */
uint8_t I2C_read_ack(void);

/* Returneaza cadrul de date primit de la slave si trimite NACK acestuia.
*/
```

```
uint8_t I2C_read_nack(void);

/* Functia trimite conditia de start (adresa slave-ului 'address'),
 * apoi trimite 'length' cadre de date din bufferul 'data'.
 */
uint8_t I2C_transmit(uint8_t address, uint8_t* data, uint16_t length);

/* Functia trimite conditia de start (+adresa slave-ului 'address'),
 * apoi citeste 'length' cadre de date si le salveaza in bufferul 'data'.
 */
uint8_t I2C_receive(uint8_t address, uint8_t* data, uint16_t length);

/* Functia trimite conditia de start (+adresa slave-ului 'address'),
 * apoi realizeaza o operatie de scriere:
 *    - trimite adresa registrului
 *    - trimite 'length' cadre de date din bufferul 'data'.
 */
uint8_t I2C_writeReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data,
uint16_t length);

/* Functia trimite conditia de start (+adresa slave-ului 'address'),
 * apoi realizeaza o operatie de scriere:
 *    - trimite adresa registrului
 * apoi trimite inca o data adresa slave-ului (de data aceasta impreuna cu
 * operatia de citire)
 *    - citeste 'length' cadre de date si le salveaza in bufferul 'data'
 */
uint8_t I2C_readReg(uint8_t devaddr, uint8_t regaddr, uint8_t* data,
uint16_t length);

/* Trimite conditia de stop. */
void    I2C_stop(void);
```

Accelerometrul (+Giroscop si Magnetometru)

Accelerometrul măsoară accelerația pe cele 3 axe ale unui sistem de referința fix în raport cu acesta (figure ##). Atunci când accelerometrul este ținut fix, singura accelerație măsurată este cea gravitațională. În funcție de valoarea componentelor accelerației gravitaționale pe cele 3 axe, putem calcula unghiul de înclinare al accelerometrului și implicit al PCB-ului pe care acesta este montat.

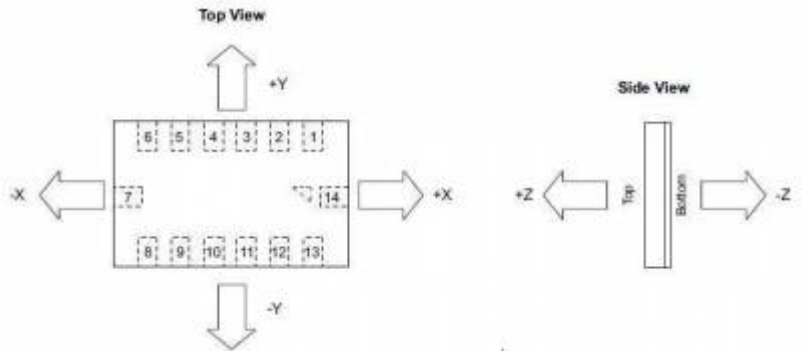


Fig. 11: Sistemul de coordonate al accelerometrului

Mod de funcționare

Accelerometrul folosește un senzor capacitiv diferențial. Acesta este format din două *condensatoare* cu o *armatură* comună *mobilă*. Ca efect al accelerației, armătura mobilă se deplasează și astfel apar modificări în capacitățile celor două condesatoare (capacitatea depinde de distanța dintre armături după formula $C = A\ \epsilon/d$). Accelerația este masurată pe baza diferenței dintre cele două capacități, iar apoi semnalul rezultat este filtrat cu ajutorul unui filtru low-pass intern.

Interfațare

Accelerometrul utilizat la laborator este unul digital (datasheet LSM9DS0) și ne oferă o serie de regiștrii de configurare/stare/date prin intermediul interfeței I2C.

Acest lucru înseamnă că, în capsula circuitului integrat, avem atât un accelerometru analogic cât și un mic microcontroller cu un ADC integrat. Acest microcontroller ne oferă valorile accelerației prin I2C.

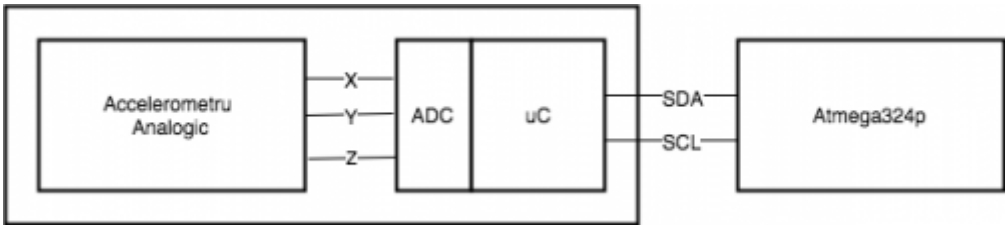


Fig. 12: Accelerometru digital - schema bloc simplificata

API Biblioteca Accelerometru LSM9DS0

În scheletul de laborator există deja o bibliotecă pentru acest accelerometru. Aceasta utilizează API-ul expus de biblioteca de I2C descrisă mai sus. Deschideți header-ul ”LSM9DS0.h” pentru API-ul complet al accelerometrului.

```
/*
 * Functia initializeaza senzorii (accelerometru, gyroscop, magentometru)
 *
```

```
 * @return - true on success, false otherwise
 */
bool LSM9DS0_init(void);

/*
 * Functia citeste si salveaza in 'accelData' valorile acceleratiei curente
 * de pe fiecare axa X, Y, Z.
 *
 * Unitatea de masura este: mG.
 *   1 G ~= 9.8m/s^2
 *   1 G = 1000 mG
 */
void LSM9DS0_readAccel(vector3f_t *accelData);

/*
 * Functia citeste si salveaza in 'magData' valorile campului magnetic
curente
 * de pe fiecare axa X, Y, Z.
 *
 * Unitatea de masura este: mGauss.
 *   1 Gauss = 1000 mGauss
 */
void LSM9DS0_readMag(vector3f_t *magData);

/*
 * Functia citeste si salveaza in 'gyroData' valorile vitezei unghiulare
 * de pe fiecare axa X, Y, Z.
 *
 * Unitatea de masura este: DPS - Degrees-Per-Second.
 */
void LSM9DS0_readGyro(vector3f_t *gyroData);
```

Senzor Presiune si Temperatura

Pe aceeași magistrală I2C avem conectat atât accelerometrul cât și un senzor digital de presiune. Aceștia sunt I2C Slaves și au adrese diferite (specificate in datasheet-ul fiecăruia). Senzorul disponibil pe placa de laborator se numește MPL3115A2.

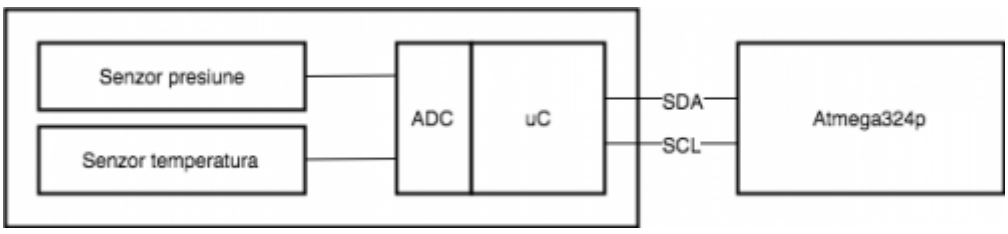


Fig. 13

API Biblioteca Senzor Presiune MPL3115A2

La fel ca în cazul accelerometrului, în scheletul de laborator există deja implementată o bibliotecă pentru citirea presiunii, temperaturii și a altitudinii. Aceasta utilizează tot biblioteca de I2C pentru comunicație. API-ul complet al acesteia este disponibil în fișierul ”MPL3115A2.h”.

```
/*
 * Functia initializeaza senzorul de presiune.
 *
 * @return - true on success, false otherwise
 */
bool MPL3115A2_init();

/*
 * Functia citeste valoarea presiunii atmosferice.
 *
 * @return - valoarea presiunii atmosferice (in pascali)
 */
double MPL3115A2_getPressure(void);

/*
 * Functia calculeaza altitudinea pe baza presiunii atmosferice curente.
 *
 * @return - valoarea altitudinii (in metri)
 */
double MPL3115A2_getAltitude(void);

/*
 * Functia citeste temperatura curenta.
 *
 * @return - valoarea temperaturii (in grade celsius)
 */
double MPL3115A2_getTemperature(void);
```

LCD grafic

LCD-ul folosit are rezoluția 128×160 și este controlat prin controller-ul ST7735R. Interfațarea cu microcontroller-ul se face prin SPI (MOSI/MISO/SCK - la fel ca la cardul SD). Comunicația între microcontroller și controller-ul LCD-ului grafic se face printr-un set de comenzi specificate în datasheet-ul controller-ului LCD. Scheletul laboratorului conține API-ul pentru lucrul cu acest controller în fișierul `ST7735R.h`. Mai jos sunt câteva dintre funcțiile oferite:

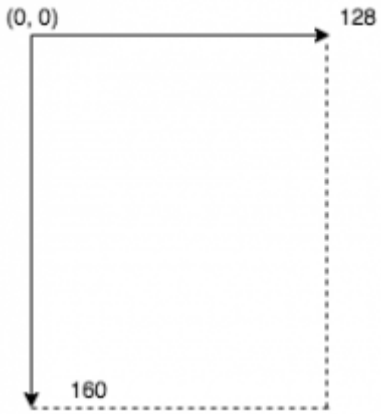
```
/* Initializeaza Display-ul grafic. */
void ST7735R_Begin();

/* Deseneaza o linie de la (x0, y0) la (x1, y1), avand culoarea data de
parametrii r, g, b. */
```

```
void ST7735R_Line(int x0, int y0, int x1, int y1, uint8_t r, uint8_t g,
uint8_t b);

/* Afiseaza un text, incepand de la pozitia (x, y).
 * Textul are culoarea specificata de pereche (r, g, b).
 * Background-ul pe care este afisat textul are culoarea (bgR, bgG, bgB).
 */
void ST7735R_DrawText(int x, int y, const char *text, uint8_t r, uint8_t g,
uint8_t b,
                      uint8_t bgR, uint8_t bgG, uint8_t bgB);

/* Desenează un cerc cu centrul la coordonatele (x, y), de raza 'radius', cu
culoarea specificată. */
void ST7735R_Circle(int x, int y, uint8_t radius, uint8_t red, uint8_t green,
uint8_t blue);

/* Desenează un cerc cu centrul la coordonatele (x, y) de raza 'radius' și
îl umple cu culoarea dată. */
void ST7735R_FilledCircle(int x, int y, uint8_t radius, uint8_t red, uint8_t
green, uint8_t blue);

/* Este un macro: deseneaza un dreptunghi cu colțul stânga sus dat de (x0,
y0)
 * și colțul dreapta jos dat de (x1, y1).
 * Dreptunghiul este umplut cu culoarea (r, g, b).
 */
ST7735R_FillRect(x0, y0, x1, y1, r, g, b);
```

Alte macro-uri utile disponibile in ”ST7735R.h”:

```
#define ST7735R_FONT_HEIGHT 15
#define ST7735R_WIDTH       128
#define ST7735R_HEIGHT      160
```

Poziția pixelului (0,0) este în colțul din stânga sus.



Date stocate în memoria de program

Programele care folosesc șiruri de caractere sau alte date constante le pot stoca în memoria de program a microcontroller-ului. Un avantaj al acestei abordări este acela că se economisește memorie de date în detrimentul memoriei de program care este de obicei mai mare (32k față de 2k). Pentru a stoca constanta în memoria de program, în declarația acesteia trebuie folosit macro-ul PROGMEM sau PSTR:

Pentru citirea unui byte stocat în memoria de program se folosesc funcții precum: `pgm_read_byte` , `pgm_read_word`. Există și corespondențe ale funcțiilor de lucru pe stringuri care lucrează pe date stocate în memoria de program : `memcpy_P`, `strlen_P` și chiar o funcție `printf_P`. Mai multe detalii puteți găsi aici.

Această abordare este folositoare doar pentru constante pentru că memoria program, implementată ca memorie flash, se uzează cu timpul - suportă un număr limitat de scrieri.

Aplicație

Pe placa folosită în laborator aveți conectate la pinii ADC următoarele componente:

- senzor de temperatură analogic (*ADC0*)
- senzor de lumină (*ADC1*)
- 6 butoane+rezistențe la pinul *ADC5* (divizor de tensiune)
- accelerometru - conectat la I2C
- senzori digital presiune și temperatură - conectat la I2C

1. **(2p)** Implementați funcția `ADC_show()`. Folosind API-ul LCD-ului grafic, afișați pe acesta mesaje pentru fiecare componentă conectată la ADC. În dreptul fiecarei componente afișați valoarea sa obținută cu `ADC_get` (care va returna 0 în acest moment).
   - Hint: folosiți funcțiile recomandate in snippet-ul de API din secțiunea LCD Grafic
   - Hint: folosiți `sprintf` pentru conversii în șiruri de caractere
2. **(2p)** Inițializați ADC-ul și faceți conversia pentru fiecare dintre canalele sale. Afișați valorile pe LCD în dreptul fiecărei componente.
   - Implementați funcția `ADC_init()` cu configurările menționate în cod.
   - Implementați funcția `ADC_get(uint8_t channel)` care primește index-ul unui canal și întoarce rezultatul conversiei pentru acel canal.
   - Implementați funcția `ADC_voltage(int raw)` care converteste valorile primite de la ADC in tensiunile corespunzatoare. Modificați funcția `ADC_show()` astfel incat sa afiseze noile valori. Țineți cont că AVCC este legat la 3.3V.
3. **(2p)** Ne dorim să afișăm în continuare pe display valorile citite de la senzorii digitali. Implementați funcția `I2C_show()`. Folosind API-ul LCD-ului grafic, afișați valorile cerute mai jos. Folosiți API-ul accelerometrului și al senzorului de presiune.
   - accelerația de pe fiecare axa X, Y și Z
   - temperatura ambientală
   - presiunea atmosferică

4. **(1p)** Folosind butonul PB2 puteți comuta între afișarea datelor achiziționate de la senzori și un joc cu bilă controlat prin accelerometru. Implementați mișcarea pad-ului (stanga-dreapta) folosind valoarea accelerației de pe axa OY.
   - Poziția pad-ului se schimbă incremental în funcție de înclinația plăcii.
5. **(3p)** La exercițiile anterioare convertorul realizează conversii în continuu, luând pe rând canelele. Acum va trebui să îl reconfigurați să facă aceste conversii doar o dată la 100ms. Folosiți întreruperi atât pentru ADC cât și pentru Timer.
   - Execuția va fi similara modelului: TRIGGER_TIMER → conversia 1 → conversia 2 → ... → ultima conversie → STOP_ADC. Nu se va reporni ADC-ul dupa ultima conversie. ADC-ul este repornit de către Timer.
   - Cand o conversie ADC este finalizată, salvați valoarea citită de la canalul respectiv în `ADC_value[]`.
   - Definiți macro-ul ADC_USE_IRQ pentru a schimba pe modul întrerupere.
     - În funcția `ADC_init` configurați ADC-ul astfel încât să genereze întreruperi la finalul conversiilor
     - În funcția `ADC_get` doar veți întoarce rezultatul ultimei conversii pentru canalul primit ca parametru.
6. **(3p - bragging rights)** Până acum ne-am folosit de o bibliotecă I2C precompilată, complet funcțională (`i2c_master.o`). Ștergeți fișierul `i2c_master.o` și modificați `Makefile`-ul să folosească `i2c_master.c`.
   - Implementați funcțiile `I2C_start()`, `I2C_stop()`, `I2C_write()`, `I2C_read_ack()`, `I2C_read_nack()`
   - Folosiți-vă de capitolul 21.9. din datasheet.
   - I2C-ul trebuie configurat în modul Master

Vedeți exemplul din datasheet - capitolul 21.6.

Resurse

- Schelet laborator
- Soluție laborator
- Datasheet ATmega324
- Datasheet accelerometru LSM9DS0
- Datasheet senzor de presiune MPL3115A2
- Datasheet controller ST7735R
- PDF laborator
- Responsabili: Adriana Drăghici Iulian-Răzvan Mateșică