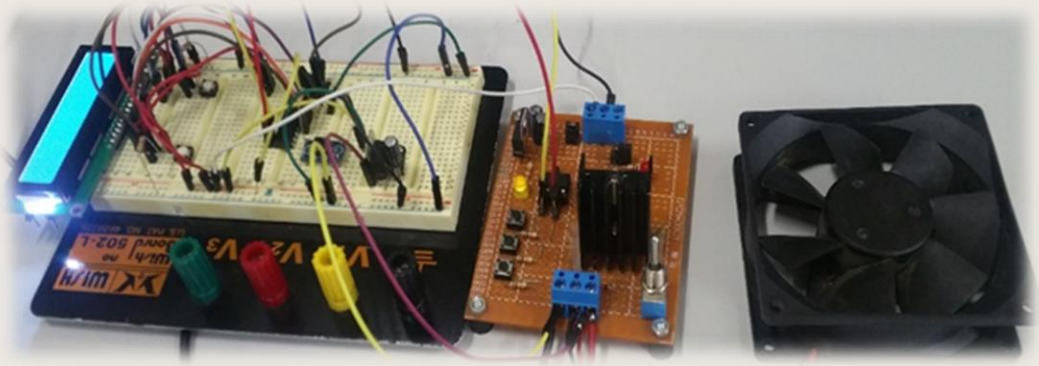
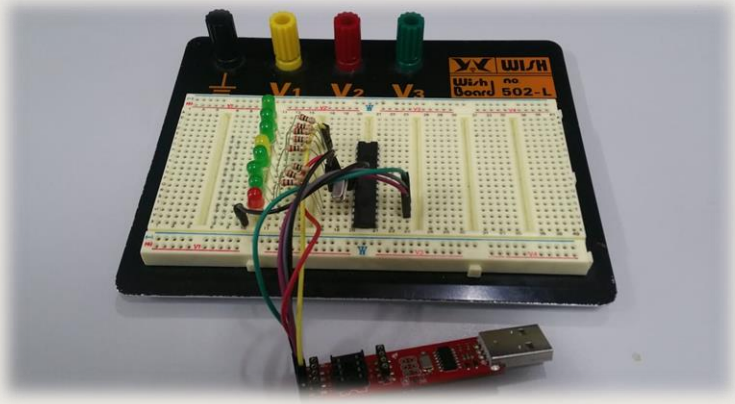
Ioana-Cornelia GROS Lucian Nicolae PINTILIE Teodor Crișan PANĂ

SISTEME EMBEDDED ÎN INGINERIE ELECTRICĂ
GHID DE APLICAȚII





Editura UTPRESS
Cluj-Napoca, 2020
ISBN 978-606-737-431-5

Editura U.T.PRESS
Str. Observatorului nr. 34
C.P. 42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Director: Ing. Călin D. Câmpean

Recenzia: Conf.dr.ing. Ioan Incze
Ş.l.dr.ing. Călin Cenan

ISBN 978-606-737-431-5

CUPRINS

1. INTRODUCERE

1.1 Sisteme embedded

În general, aplicaţiile specifice domeniilor inginereşti, cu precădere cel al ingineriei electrice şi electronice, au nevoie de proiectarea de sisteme de control care să execute anumite procese în timp real; acest lucru a fost facilitat de folosirea, pe diverse nivele de complexitate, corespunzătoare etapelor lor de apariţie şi dezvoltare, a sistemelor încorporate (corespondentul mult mai apreciatului termen "embedded systems"). Denumirea acestor dispozitive face referire, în primul rând, la modularitatea obţinută atât din punct de vedere hardware cât şi software a sistemelor care le cuprind, cu toate elementele şi subsistemele lor. Totuşi, referirea ca sistem încorporat a unui dispozitiv digital de calcul îl deosebeste de un procesor utilizat în sisteme de calcul personale prin faptul că, în cazul celui dintâi, elementele componente: unitatea centrala de procesare, memoria de date şi memoria program, cât şi circuitele periferice, sunt conţinute pe acelasi cip, deci nu sunt componente separate, de sine stătătoare.

Tehnologia sistemelor embedded reprezintă nucleul a miliarde de dispozitive considerate „inteligente" care susţin activităţile de zi cu zi; astfel, un utilizator obişnuit interacţionează zilnic cu sute de microcontrolere încorporate în domenii ca: telecomunicaţii, automobile, aparatură electrocasnică şi electronică, aparatură medicală, sisteme de automatizare şi control, terminale comerciale, electronice etc.

Proiectanţii de ***sisteme embedded*** crează, în acest context, aplicaţii inovatoare şi au ca sarcină îmbunătăţirea permanentă a aplicaţiilor existente pe palierele ce ţin de rapiditatea în funcţionare, consumul energetic redus, interfeţele îmbunătăţite şi posibilitatea de gestionarea de la distanţă a dispozitivelor. Corelate cu conceptele actuale de „conectivitate universală" a dispozitivelor şi de procesarea informaţiilor în "cloud" („cloud computing"), aceste aspecte duc la îmbunătăţirea performanţelor în domeniu.

Interesul major pentru ***platforme de dezvoltare*** cu procesare rapidă şi elemente de „control inteligent", în scopul implementării diferitelor aplicaţii prin intermediul senzorilor şi actuatoarelor, se situează în contextul de avansului tehnologic cuprins în noţiunea „Internet of Things" sau Internetul tuturor lucrurilor. Inovaţiile în acest sens se referă la domenii precum medicina, astronomia, transporturile, tehnologiile portabile etc. Producătorii de platforme de dezvoltare dedicate lansează permanent produse competitive prin care este facilitat accesul cercetătorilor, programatorilor, studenţilor, dar şi amatorilor, în sfera conceperii proiectelor în direcția automatizărilor, acționărilor electrice și a roboticii. Aceste

platforme de dezvoltare se adresează în mod preponderent categoriei de amatori promotori ai conceptului „DiY (eng. Do-it-Yourself)”. Din punct de vedere al tehnologiilor prezente în domeniul microcontrolerelor şi al sistemelor încorporate implicate, gama de dispozitive este variată, pornind chiar de la modelele de baza cu adresare pe 8 biţi (de exemplu Intel 8051, Atmega 328 etc.), până la diverse variante de PIC-uri *(eng. Peripheral Interface Controller)*, AVR (eng. Alf and Vegard's RISC – Reduced Instruction Set Computer), ARM (eng. Advanced RISC Machine), Arduino şi derivatele compatibile cu acesta etc.

Experţii din industrie examinează impactul acestor concepte în noua generaţie de tehnologii pentru microcontrolere în termeni legaţi de: consumul redus de energie, reducerea costurilor de fabricație și comercializare, protocoale de comunicație pentru diferite strategii de implementare. Din punctul de vedere al cerințelor utilizatorului, elementele de importanţă sunt reprezentate de interfețele interactive, conectivitate universală, securitate şi siguranţă în noile sisteme embedded. Aceste cerințe au condus la o concentrare asupra procesului de dezvoltare a unor unelte complexe de vizualizare şi depistare a problemelor legate de programele software asociate (eng. Debugging and Development Environment / Engine).

O **clasificare limitată a sistemelor embedded** (Fig.1.1. şi Fig. 1.2.), urmărind un fir evolutiv al acestora în timp, fără a realiza o altă ierarhizare, cuprinde:

(i) ***microcontrolerele***, cu evoluţie aproape continuă de la apariţia primei configuraţii, urmate de
(ii) ***procesoarele digitale de semnal DSP (eng.* Digital Signal Processors)**,
(iii) ***microprocesoare sau microcomputerele încorporate*** si, combinate în sisteme mai complexe

Tot din catergoria sistemelor de calcul se amintesc şi dispozitivele cu ***FPGA* (eng. Field Programmable Gate Array)**. Acestea sunt sisteme de calcul bazate pe circuite elementare cu porți logice reconfigurabile, având performanțe ridicate, cu utilizare largă în sisteme de control automat; totuşi, această ultimă categorie, deşi urmăreşte acelaşi scop - de dezvoltare a aplicaţiilor în timp real, la frecvenţe ridicate, totuși ele prezintă o structură hardware diferită, bazată pe arii de porți logice programabile, tehnologie care îi permite să fie reconfigurabilă din punct de vedere hardware, în funcţie de sarcina necesară a fi îndeplinită. Respectivele sisteme de calcul, fac parte din categoria *microcrocomputerelor înglobate (embedded)*. FPGA fiind o arie de porți logice reprogramabile, este reconfigurată la fiecare repornire de către o unitate centrală logică de procesare (CPU), de unde si conceptul de **„*System on a Chip” (SoC)*.** Chiar dacă sistemele de calcul conțin FPGA, principiul funcțional nu se bazează existența acestuia, ci pe o unitatea de procesare de sine stătătoare care programează aria de porți (de exemplu platformele DIGILENT Spartan, Zynq, Altera etc). Astfel, sistemele cu FPGA pot fi considerate microcomputere cu structură specială sau arhitectură modificată.

În contextul dezvoltării sistemelor embedded, trebuie mentionate cele două **categorii de tehnologii** care le integrează:

- **Tehnologia on-board** – se referă la integrarea anumitor module de interfațare (cu funcția de intrare / ieșire, care într-o altă arhitectură ar fi fost dedicate sau de sine stătătoare) în structura generală a sistemului de calcul (ex. modulul grafic sau modului audio integrat în placa de bază);

- **Tehnologia System-on-Chip (SoC)** – se referă la integrarea mai multor structuri mai complexe (de exemplu, un microprocesor, o memorie, o arie de porți programabilă), într-un singur cip sau o singură pastilă. Microprocesorul în sine este un sistem complex format din circuite logice simple, la fel și memoria sau o arie de porți programabilă (ex. FPGA);

Un loc semnificativ în această clasificare îl ocupă ***platformele de dezvoltare cu sisteme încorporate***, ca tendinţă determinată atât de industrie şi de progresele făcute în domenii precum electronica şi electrotehnica, telecomunicaţii şi sisteme de calcul, cât şi de interesul utilizatorilor amatori sau în scop didactic, de a crea aplicaţii din sfera „DiY" (eng. do-it-yourself).

În termeni generici, un sistem „embedded" sau încorporat reprezintă un ansamblu sau un grup de subansmable care, din punct de vedere hardware, conţine elemente de calcul specifice unui computer, având partea software dedicată aplicaţiei, nu generalizată (ex. bancomatele, imprimantele cu acces la rețea etc...). Un astfel de dispozitiv embedded este proiectat pentru a funcţiona fără intervenţie umană majoră, pentru a răspunde la evenimente externe în timp real.

Sunt întâlnite ca parte integrantă a oricărui dispozitiv electronic modern şi execută, în general, acţiuni specifice de control, prin intermediul senzorilor şi actuatoarelor, întrunind condiţii specifice aplicaţiilor în timp real. Câteva exemple generale de aplicare a sistemelor embedded sunt listate în continuare:

- Automobile – elementele de control specifice şi partea de comunicare;
- Dispozitive electronice şi aparatură electrocasnică;
- Sisteme automatizate;
- Dispozitive specifice sistemelor informaţionale: telefoane mobile, echipamente în telecomunicaţii, dispozitive pentru comunicaţia wireless etc;
- Reţele de senzori: pentru clădiri, monitorizare trafic şi mediu, domeniul militar

- Dispozitive multimedia portabile;
- Platforme şi aplicaţii din sfera “Internet of Things” etc.

Domeniul sistemelor embedded a cunoscut în ultimele decenii o dezvoltare rapidă la nivel mondial, inovaţiile din domeniu fiind susţinute în industrie, cu impact în domenii precum: al autovehiculelor, al transportului, al automaticii, al echipamentelor medicale, comunicatiilor, sistemelor energetice etc; la nivel mondial, un procent de peste 90% din microprocesoarele produse reprezintă, de fapt, elemente ale sistemelor embedded – încorporate în aplicaţii pe diverse nivele de complexitate (microcontroler, DSP, microprocesor).

1.2 Microcontrolere şi platforme de dezvoltare în contextul actual:

În contextul actual, există o varietate mare de sisteme embedded şi platforme de dezvoltare populare şi utile în aplicaţii de comandă şi control, precum şi o varietate de posibilităţi de programare dedicată a acestora.

Tendinţa generală este una de omogenizare a capabilităţilor hardware, cât şi a modului de programare a acestora; amintim exemplul Matlab - Simulink, care conţine biblioteci de funcţii şi capabilităţi de folosire a numeroase astfel de platforme. Acest subcapitol prezintă câteva sugestii din categoria celor mai uzuale şi mai accesibile microcontrolere / platforme de dezvoltare, iar capitolele următoare propun *aplicaţii de bază*, uşor de implementat, pretabile spre *implementare pe oricare din dispozitivele propuse.*

Fig.1.1. scoate în evidenţă funcţia de bază a unui sistem cu microcontroler şi anume, funcția de interfațare a procesului cu mediul sau factorul uman (eng. Human Machine Interface – HMI sau human interface device – HID).

Fig. 1.2. prezintă **sistemul cu microprocesor (micro – computerul),** care poate realiza funcția de gestiune a sarcinilor de execuție *(‘Multitasking’*). Resursele necesare rulării aplicației (ex. memorie, prioritate de întrerupere etc.) pot fi alocate la cerere de către un **sistem de operare**. Un exemplu concludent este un **microcomputer** care participă la **implementarea algoritmului de gestiune la nivelul întregului sistem** (ex. computer coordonator central SCADA.).

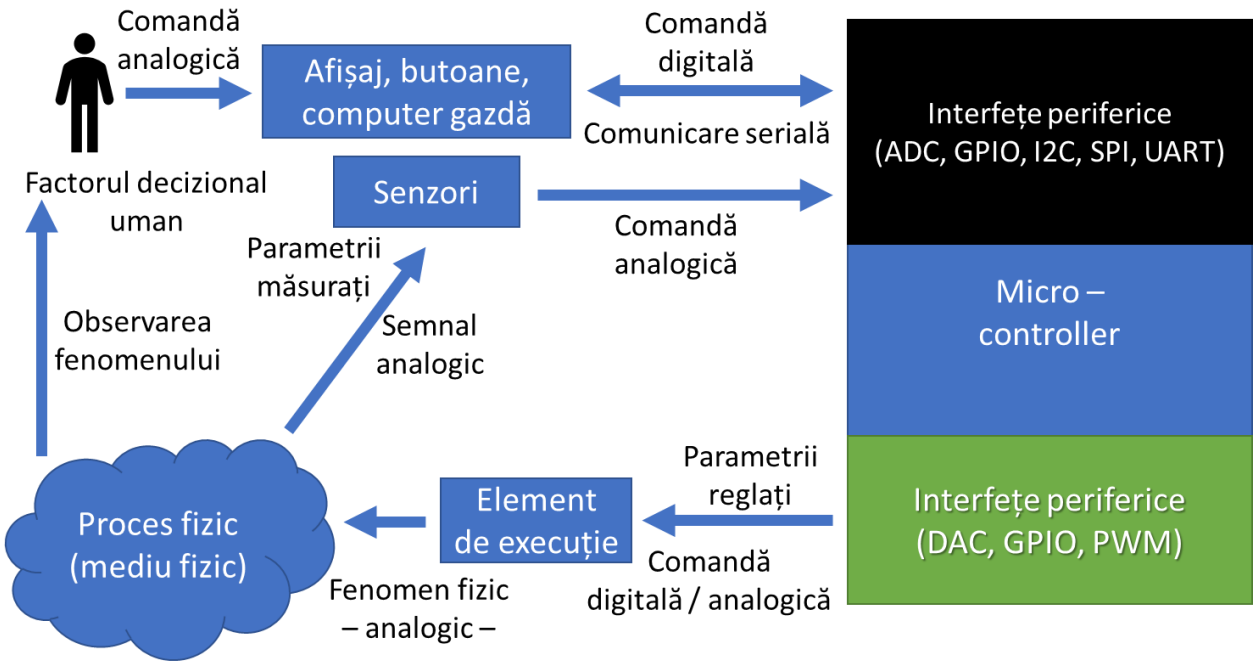


Fig.1.1. Schema unei aplicatii cu microcontroler

În contextul actual, există pe piata o diversitate de dispozitive de tip **platforme de dezvoltare.** Acestea reprezintă mijlocul prin care se folosesc capabilităţile de execuţie ale unui microcontroler, sau procesor (‘mini-computere’ fără periferice) prin integrarea lor pe o placă unică de circuit, cu elementele auxiliare necesare dezvoltării unei *aplicaţii de achiziţii de date sau control.*

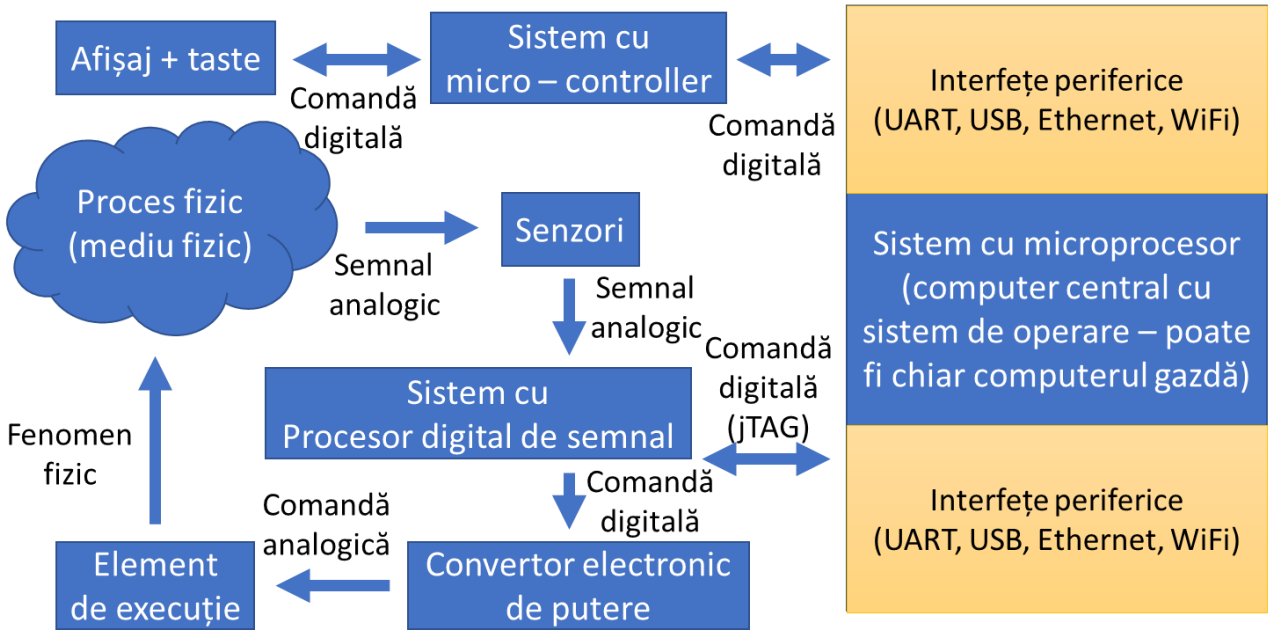


Fig. 1.2. Schema unei aplicatii cu microprocesor incorporat

Pe lângă dimensiunile şi preţurile relativ scăzute ale acestor platforme, nivelul lor de popularitate rezidă din accesibilitatea utilizatorilor, indiferent de nivelul lor de experienţă, pentru a înţelege elemente specifice electronicii şi a construi propriile circuite / dispozitive. Folosind elemente de senzoristică şi actuatoare adecvate, şi realizând programul corespunzător, utilizatorul poate efectua operații specifice de control, în functie de performanţele platformei. Generalizând, funcţiile de bază ale acestor platforme sunt:

- preluarea valorilor de la un set de componente hardware recunoscute ca intrări (butoane, senzori, comutatoare etc);
- generarea unui semnal în vedrea activării unui set de elemente hardware de execuţie (actuatoare), sau furnizarea mărimii de referință pentru un convertor electronic;

Selecţia propusă de sisteme embedded cu microcontrolere şi platforme de dezvoltare cuprinde:

- **Familia AVR: arhitectura ATMEGA 328P si Attiny**
- **Familia Arduino**
- **Intel Galileo**
- **Raspberry Pi 3B**

ATMEGA 328P din **familia AVR** este un reprezentant important în clasele de microcontrolere pe 8 biţi, cu consum redus de putere şi este preponderent utilizat în aplicaţiile de interfaţare. Notorietatea acestui dispozitiv pe piață se datorează faptului că, acesta stă la baza platformei de dezvoltare Arduino, susţinută și promovată de către de comunitătile DiY şi „Open-Source” (ex. MakeZine, LifeHacker, GitHub). Este importantă mențiunea faptului că, AVR a fost o sub-divizie a companiei Atmel, care producea și comercializeaza microcontrollere sub marca ATMega. Începând din anul 2016, tehnologia și sigla AVR, ATMega și Atmel a trecut în posesia companieie MicroChip Technologies, care până în prezent producea și comercializa microcontrollere sub marca PIC, DSPic (varianta procesoarelor de semnal DSP);

Din punct de vedere al structurii hardware, este dotată cu următoarele caracteristici:

- Set de instrucţiuni pe 8 biţi;
- Lucrul cu 32 de regiştri;
- Frecvenţa de operare maxim 16 MHz;
- 32 kB memorie FLASH programabilă;
- 1 kB memorie EEPROM;
- 2 kB memorie SRAM;
- Unitate de temporizare / numărător pe 8 biţi (rezoluţie 0-255 trepte);
- Convertor ADC cu rezoluţie pe 10 biţi (0-1023 trepte);
- Interfeţe seriale: UART, I2C, SPI şi intrări / ieşiri de uz general GPIO;

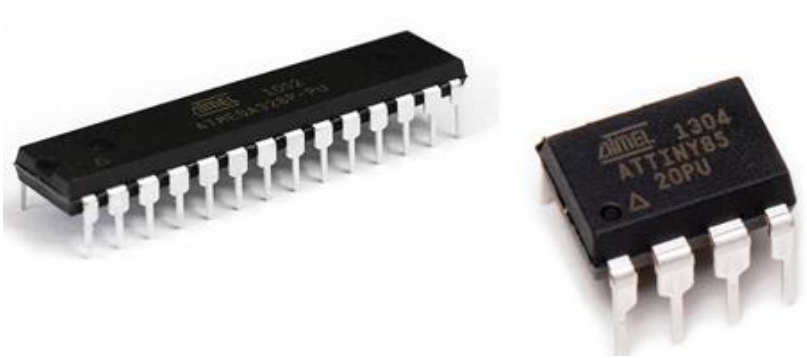


Fig. 1.3. – ATMEGA 328 şi Attiny85 [1], [2]

Microcontrolerului integrat ATTiny 45 reprezintă o variantă utilă mai ales în aplicaţiile cu nivel mare de portabilitate, în care numărul pinilor de intrare/ieşire solicitaţi nu este foarte mare, iar integrarea fizică a acestuia în aplicaţie aduce simplificări circuitului. Are aceleaşi facilităţi legate de compatibilităţi hardware şi software cu platformele de dezvoltare Arduino, iar structura sa hardware conţine:

- Memorie program de tip flash cu capacitatea de 4 KB;
- Memorie SRAM: 256 Byte;
- Memorie de date EEPROM 256 Byte;
- 6 intrări și ieșiri digitale de uz general (GPIO);
- 32 de regiștrii de uz general;
- Un numărător cu reprezentare pe 8 biți;
- Convertor ADC cu rezoluţie pe 10 biți;

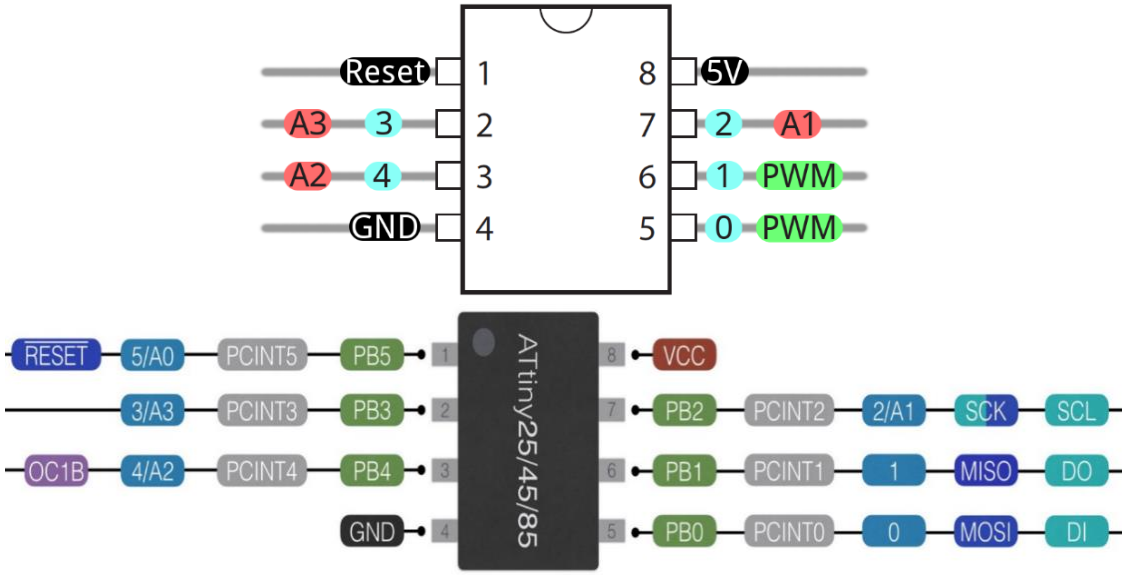


Fig. 1.4. Harta pinilor si terminalele de acces ATTiny45 [3], [4]

Comparativ cu alte microcontrolere din seria ATMEL de la AVR, acest dispozitiv are un set de instrucţiuni reduse ca număr, memorie mai puţină şi mai puţine periferice; totuşi, consumul de energie şi pretul sunt semnificativ reduse, iar avantajul major rezidă din posibilitatea integrării acestuia cu un program prescris, într-o aplicaţie independentă de sine stătătoare (eng. standalone).

Atât ATMEGA328P, cât şi ATTiny pot fi programate prin mai multe metode:

- Prin programator specializat ISP (de ex. AVR ISP mk II – USB, respectiv SparkFun USBTinyISP – TinyAVR);
- Cu adaptor de la interfaţa serială
- Cu ajutorul altui microcontroler (de ex. Arduino)

Scrierea logicii de comandă pentru o aplicaţie dorită cu oricare din aceste platforme se bucură de folosirea unei palete largi de instrumente, pornind de la limbajul de asamblare, la sintaxa standard şi programarea cu blocuri grafice.

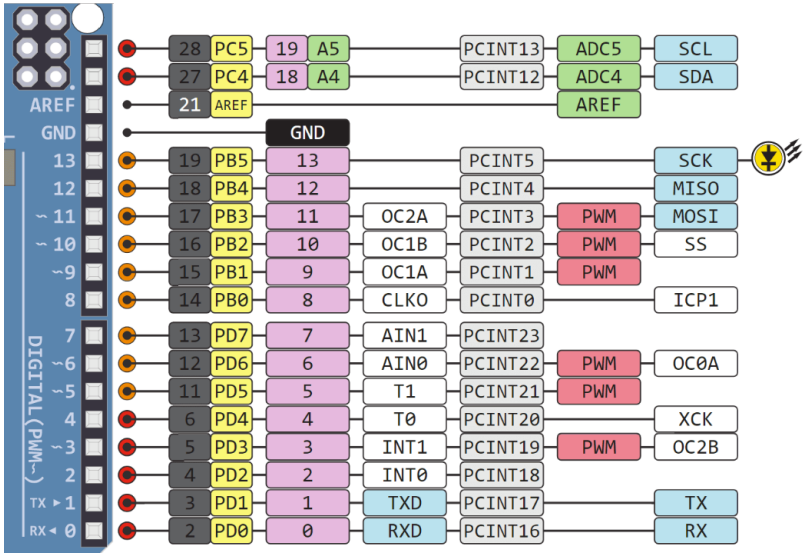


Fig. 1.5. Harta pinilor si terminalele de acces Arduino/AVR [6]

În evoluţia recentă a sistemelor embedded, s-au impus în atenţia utilizatorilor platformele de dezvoltare, încadrate, în general, în categoria dispozitivelor SBC (Single Board Computer), care pot fi utilizate pentru dezvoltarea rapidă, facilă, cu minime resurse auxiliare din punct de vedere al circuitelor, a unor aplicaţii de control şi de achiziţii de date şi cu posibilitatea implementării lor în reţele inteligente, cu acces la Internet. Cea mai populară şi mai vânduta familie de placi de dezvoltare în acest sens o reprezintă **arhitectura Arduino**, cu acces deschis la surse, accesibilă ca preţ, robustă, cu conexiuni standard care permit implementarea de sisteme de măsură şi control autonome, controlate de la distanţă, pe diverse categorii de complexitate.

Familia de platforme de dezvoltare Arduino a fost lansată în anul 2005, cu o structură bazată pe circuite cu microcontroler Atmel, unele variante mai performante, având şi elemente de procesare sau de comunicare adiţionale. O scurtă trecere în revistă a plăcilor din familia Arduino [5]:

- *Arduino Uno, Nano, Mini 05* – cu microcontroler pe 8 biţi ATmega328 şi unele cu acces la comunicaţia Ethernet, posibilitatea folosirii de card Micro SD;
- *Arduino Mega 2560* – cu microcontroler pe 8 biţi ATmega2560 şi posibilitatea folosirii unui kit suplimentar de accesorii pentru folosirea cu Android;
- *Arduino Leonardo, Micro, Robot, Esplora* – cu microcontroler pe 8 biţi ATmega32u4, acces la comunicaţia Ethernet, folosirii de card Micro SD;
- *Arduino Due* – cu microcontroler Atmel SAM3X8E ARM Cortex-M3 pe 32 de biţi, cu capacitate extinsă pe partea de intrări/ieşiri digitale (54), respectiv analogice (12), capacitate mai mare a memoriei RAM (de tip SRAM) şi funcţionare la 3.3 V, spre deosebire de 5 V la celelalte variante.

În aceeaşi categorie, regăsim platforme de dezvoltare precum **Intel Galileo Gen1 şi Intel Galileo Gen2** - variantele lansate de Intel începând cu anul 2013, ca versiuni total compatibile, atât din punct de vedere hardware, cât şi software, cu toate plăcile din familia Arduino. Fiind bazate pe clasa de procesoare de tip SoC ("System on a chip") Intel Quark Soc X10000, pe 32 de biţi, au avantajul folosirii unui sistem de operare, ceea ce aduce un plus de performanţă în sisteme de control avansate.

Această platformă de dezvoltare permite diferitelor categorii de cercetatori, programatori sau dezvoltatori de sisteme embedded să se folosească de caracteristicile unui circuit integrat (*engl.* PCB - *Printed Circuit Board)* pentru a realiza într-un mod accesibil, modular, cu acces la surse deschise, ansambluri de componente electrice şi electronice, în aplicaţii de diferite complexităţi, pentru achiziţii de date şi control.

Programarea platformei poate fi realizată prin intermediul mediului de programare dedicat (eng. Integrated Development Environment – I.D.E.) sau prin intermediul unor medii de simulare și testare precum Matlab Simulink, Altair VisSim etc. De asemenea, se poate folosi şi ca un sistem total independent cu procesor şi sistem de operare (Microsoft Windows IoT, Linux dedicat), în aplicaţii independente (eng. standalone).

Deşi unitatea centrală Quark oferă compatibilitate cu arhitecturile x86, fiind echivalentul unui Pentium original la 400MHz, cu consum energetic redus *("ultra-low-power"*), performanţele sale nu sunt ale unui mini-computer, cum este considerat succesorul sau, **Raspberry Pi** [7], lipsindu-se de orice formă de ieşiri video sau periferice clasice, dar se remarcă în viteza de execuţie a codului complex, comparativ cu variantele simple de microcontrolere; totodată, folosind *sistem propriu de operare*, rulează ca *server web*, fără a necesita un calculator separat, folosind doar *portul Ethernet încorporat*, sau *printr-un adaptor wireless*.

Fiind o variantă complexă a platformelor Arduino, cuprinzătoare din punct de vedere al elementelor interfaţate, chiar dacă cu nucleu de procesare diferit, explicarea principalelor elemente constructive se va face, în această carte, raportat la platforma Intel Galileo.

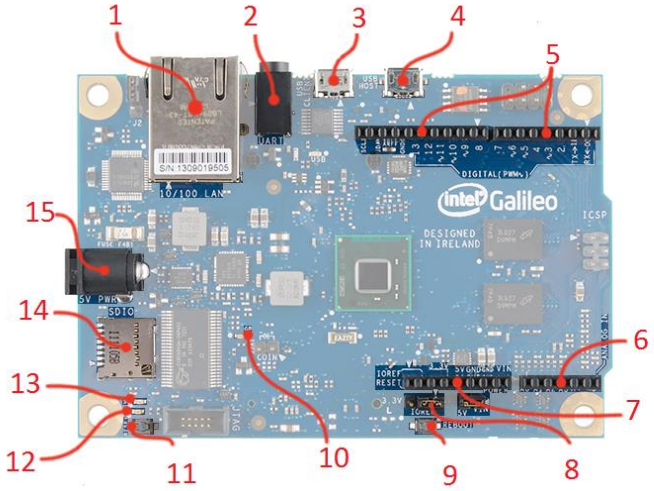


Fig. 1.6. Elementele constructive ale platformei de dezvoltare Intel Galileo

1 *Conectorul Ethernet* - conectează platforma la orice reţea LAN de 10/100 Mb/s, cu ajutorul unui cablu mufat corespunzator;

2 *Conectorul RS-232* –folosesit pentru a accesa, de exemplu terminalul Linux;

3 *USB Client* – conectează platforma la calculator printr-un cablu USB, pentru a incarca executabilele pe platforma sau pentru a alimenta (Arduino);

4 *USB 2.0 Host* – face posibilă conectare a până la 128 de dispozitive, care pot fi conectate prin USB, cum ar fi tastaturi, camere web sau dispozitive de stocare in masa (Intel Galileo);

5 *Intrări/ Ieşiri Digitale* – 8 pini, de la D0 la D7, care includ UART pe pinii 0/1, şi PWM pe pinii 3, 5 şi 6; 10 pini, de la D8 la SCL, care includ pini de tip I^2C şi PWM pe pinii 9, 10 şi 11;

6 *Intrări/ Ieşiri Analogice* – 6 pini de intrare, de la A0 la A6, şi 8 pini pentru alimentare: GND, 5V, 3.3V, reset etc;

7 *GND – pini de alimentare;*

8 *IOREF Select* – este un pin de tensiune, care este utilizat pentru selecţia între 3.3V şi 5V;

9 *Buton pentru reboot* – butonul va şterge intrările Galileo, inclusiv sistemul de operare Linux. Timpul de resetare este de aproximativ 30 de secunde;

10 *Pin 13 LED* –LED incorporat pe placă, conectat prin circuit la pinul 13;

11 *Buton de reset* – reporneste doar schita de program din mediul Arduino IDE care ruleaza;

12 *LED indicator pentru conectarea la o sursă de putere*;

13 *LED indicator pentru utilizare cardului SD*;

14 *Conector memorie µSD* – se utilizează, în cazul în care se utilizează platforma ca un calculator individual, pentru a încărca sistemul de operare Linux sau pentru incarcarea altor elemente (de exemplu o pagina web) necesare unei aplicatii mai complexe. Se poate utiliza un card de memorie cu memoria de până la 32 GB;

15 *Conector pentru alimentarea la +5V* – Cablu de alimentare de 5V, trebuie să fie inclus.

Arhitectura x86 a procesorului (Intel Galileo) este folosită în mod uzual în computerele personale, cu sistem de operare Windows, rapide şi puternice. Microcontrolerele ATMEGA (Arduino), pe de alta parte, sunt folosite la scară mare în elemente electronice portabile, cu eficienţă energetică mai mare. Procesorul Intel Quark este considerat primul procesor compatibil cu dispozitivele portabile din sectorul IoT.

Diferenţa dintre arhitectura ARM şi arhitectura x86 se referă la dimensiunea setului de instrucţiuni. ARM este o arhitectură de tip RISC (*Reduced Instruction Set Computing*, cu set de instrucţiuni mai mic şi mai simplu), iar procesoarele x86 sunt CISC (*Complex Instruction Set Computing*, cu set de instrucţiuni complex şi mai mare.

Platforma este proiectată pentru a funcţiona atât cu alimentare la 3.3 V, cât şi la 5 V, translatarea de la o tensiune la cealaltă a pinilor I/O făcându-se prin folosirea unui element ‘jumper’ pe pinul *IOREF Select*; totodată se poate folosi şi alimentarea prin USB, pe soclul corespunzător, deşi *producatorul recomandă doar alimentarea prin adaptorul specific de 5 V.*

Pini I/O digitali şi analogici (GPIO):

În cazul platformei Intel Galileo se face referire la 14 pini GPIO (*General Purpose Input Output – intrări/ieşiri cu uz general*), **8 pentru intrări/ieşiri digitale** şi **6 pentru intrări/ieşiri analogice**.

În cazul **Intrărilor/Ieşirilor digitale**, pinii 0 şi 1 sunt folosiţi implicit şi pentru conexiunea serială UART şi şase pini (3, 5, 6, 9, 10, 11) pentru aplicaţii cu ieşiri capable sa furnizeze semnal modulat în latime PWM *(Pulse Width Modulation*).

Modul de lucru al terminalelor de tip „pin” se realizează din programul specific Arduino IDE, prin funcţiile *pinMode(), digitalWrite() şi digitalRead()* – adică terminalele periferice se pot reconfigura în terminale de intrare / ieșire. Fiecare pin poate genera un curent pozitiv de maxim 10 [mA], respectiv un curent negativ de maxim 25 [mA] şi are o rezistenţă internă configurabilă pentru inter-punere la sursă sau la masă (*eng. pull-up / pull-down*), deconectată implicit în gama 5.6 -10 kΩ. În modul de ieşiri – ‘OUTPUT’, pinii digitali îşi pot schimba starea (în cazul pinilor 2 si 3 și 9) cu o frecvenţă de aproximativ 230 Hz, cu posibilitatea de a creşte această frecvenţă până la 477 Hz, prin controlarea acestuia cu functia *fastGpioDigitalWrite().*

Intrările Analogice, reprezentate prin 6 pini, sunt notate de la A0 la A5 şi pot fi folosite în acelaşi fel ca şi pinii digitali; convertorul analog-digital AD7298 aferent, are o rezoluţie de maxim 12 biţi, funcţionând în mod implicit la o rezoluţie de 10 biţi, din care rezultă un pas de conversie de aproximativ 4,8 mV (explicat, pe larg in Capitolul 3); acest aspect prezintă importanţă în cadrul calibrării diferiţilor senzori analogici.

O descriere comparativă a variantelor Intel Galileo Gen 1, Intel Galieo Gen 2, Raspberry PI, din punct de vedere al specificatiilor tehnologice si constructive este cuprinsa in Tabelul 1.1.

Tabelul 1.1. Scurtă descriere comparativă între platforme de dezvoltare

	Intel Galileo Gen1	**Intel Galileo Gen2**	**Raspberry PI**
Dimensiuni (Lxl) mm	100x70	123.8x72	85mm x 56mm
Procesor	Intel® Quark™ SoC X1000 (16K Cache, 400 MHz)	Intel® Quark SoC X1000 (16K Cache, 400 MHz)	SoC Broadcom BCM2837
RAM	256 MB DDR3 800	256 MB DDR3	1 GB RAM
Alimentare	5 V	7-15 V	5 V
Pini	14 digitali I/O, 6 intrari analogic	20 digitali I/O, 6 intrari analogic	40 GPIO
Interfete	UART, ICSP, SPI, I^2C, TWI	2 UART – 6 pini, ICSP – 6 pini, SPI, I2C, TWI	UART, SPI, I2C, Ethernet, Bluetooth, Wifi
Sisteme de operare	Arduino IDE, Linux, Windows 8.1 embedded core edition	Arduino IDE, Linux, Windows 8.1 embedded core edition	RaspBian – Debian, Ubuntu, OpenSuse, Windows 10 IoT

Compatibilă cu aceeaşi familie de platforme este şi **Intel Edison** (2014) cu arhitectură bazată pe două procesoare, Intel Atom dual-core 500MHz şi Intel Quark 100MHz, iar pentru cunoscătorii sistemului de operare Linux şi a limbajului de programare Python, o alternativă recentă şi apreciată, datorită arhitecturii sale de tip "single board computer" este resprezentat de Raspberry PI [7].

Câteva caracteristici şi funcţionalităţi specifice acestor plăci sunt enumerate în continuare:

- Folosirea intrărilor/ieşirilor de tip GPIO de două tipuri – analogice şi digitale; intrările analogice sunt folosite pentru măsurarea unui nivel de tensiune, iar ieşirile analogice pot fi implementate prin circuitul convertorului Digital - Analog (eng. Digital to Analog Converter – DAC) sau ca ieşiri PWM (eng. Pulse Width Modulation); pinii digitali pot fi folosiţi atât ca intrări, cât şi ca ieşiri.

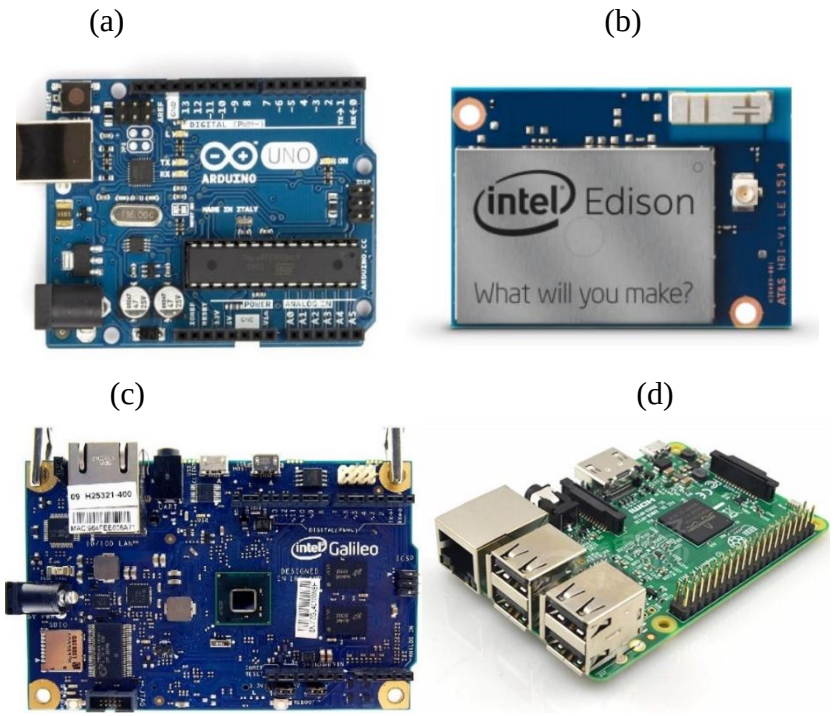


Fig.1.7. Platforme de dezvoltare: Arduino Uno (a), Intel Edison (b), Intel Galileo Gen1 (c), Raspberry PI (d)

- Folosirea diferitelor tipuri de comunicaţie:

  - USB (Universal Serial Bus) – pentru încărcarea programului scris pe calculatorul personal prin intermediul mediului specific de programare Arduino IDE (Integrated Development Environment)
  - UART (Universal Asynchronous Receiver Transmitter) – pentru comunicatia serială, preferată pentru schimbul de date între diferite platforme;
  - I^2C (Internal IC)/TWI (Two Wire Interface)
  - SPI (Serial Peripheral Interface) – pentru comuncaţii rapide, în cazul folosirii de shield-uri suplimentare

- Acces direct la folosirea microcardului SD - pentru încărcarea sistemul de operare şi a altor programe folosite şi a conexiunii Ethernet – doar unele variante constructive

- Tipurile de memorie limitate (fiind vorba despre microcontrolere), organizate ca memorie Flash, SRAM, EEPROM

Un element important în caracterizarea dinamică a platformelor în diferite aplicaţii îl reprezintă viteza maximă de achiziţie a valorilor din mediul extern, care depinde în primul rând de frecvenţa de operare a microcontrolerului; în cazul achiziţiei de date prin pinii digitali, viteza este mai mare decît în cazul pinilor analogici, datorită lipsei convertorului analog-digital ADC specific, care necesită pentru conversie mai multe cicluri maşina de execuţie. O prezentare comparativă a câtorva specificaţii ale platformelor de dezvoltare din familia Arduino, cu caracteristicile lor fizice, sunt prezentate în Tabelul 1.2.

Tabelul 1.2 – Caracteristici ale placilor Arduino Uno, Raspberry PI - B, si Intel Galileo

Denumire platforma	**Arduino Uno**	**Raspberry PI 3 - B**	**Intel Galileo Gen1**
Frecventa procesare	16 [MHz] single core	1.2 [GHz] quad - core	400 [Mhz]
Memorii	2 [kB] SRAM 32 [kB] Flash 1 [kB] EEPROM	1 [GB] sRAM Compatible SD - card	256 [MB] DDR3 8 [MB] Flash 8 [kB] EEPROM
GPIO	22	40	20
Analogic	10-bit ADC, PWM		12-bit ADC, 6 PWM
Ethernet	-	10/100 [Mb / s]	10/100 [Mb / s]
OS	Hard coded bootloader	Linux/Windows 10 IoT Core	Windows 8.1 embedded core edition, Linux
Tensiune de intrare	5 [V] – 12 [V]	3.3 [V] – 5 [V]	5[V]
Consum maxim de putere	0.233 [W]	~2.4 [W] (5V)	15 [W]

Capitolul 2 prezintă moduri de abordare software actuale pentru aceste tipuri de sisteme embedded cu microcontrolere sau platforme de dezvoltare, urmând ca în **Capitolele 3-5** să fie prezentate seturi de aplicaţii specifice *domeniului Inginerie Electrică*, care permit întelegerea sistematizată a problematicii.

2. MODALITĂŢI SOFTWARE DE ABORDARE A PLATFORMELOR DE DEZVOLTARE

Din punct de vedere software, programarea platformelor din familia Arduino se poate realiza în diverse moduri, cel mai agreat de catre utilizatori, datorită accesibilităţii şi existenţei de reţele dedicate de tip comunităţi, este prin **interfaţa dedicată Arduino IDE (Integrated Development Environment);** prin această modalitate, utilizatorii îşi pot crea proiectul pe baza unui schelet de cod realizat în limbaj C/C++ simplificat şi dedicat, cu unele extensii pentru accesarea elementelor hardware. Ulterior, acesta este salvat ca schiţă (eng.'sketch'), verificat din punct de vedere al sintaxei, compilat, iar executabilul încărcat pe memoria flash a platformei prin intermediul conexiunii specifice USB-Serial va permite rularea automată a programului scris şi funcţionarea sistemului independent. Monitorizarea rulării programului şi a datelor trimise şi primite de la senzori şi actuatoare în timpul execuţiei programului se face prin monitorul serial specific IDE-ului. Avantajul acestui mediu specific de programare este reprezentat de existenţa unui număr cuprinzător de exemple de schiţe, pe diferite domenii de aplicaţie şi de dispozitive, pe partea de comunicaţie, compatibile cu toate platformele din gama Arduino. Totodată, utilizatorii se pot folosi de multitudinea de librării incluse, furnizate de unii producători de senzori şi componente electronice, cât şi pe partea de comunicaţie: Ethernet, Wifi, GSM, UART etc.

O altă modalitate de programare a platformelor de dezvoltare prezentată în această carte, cu câteva limitări din punct de vedere al variantelor de plăci acceptate, este prin **pachetul Matlab Simulink dedicat**, care permite configurarea şi accesarea diferiţilor senzori şi actuatoare, accesarea comunicaţiei Ethernet şi Wifi şi implementarea modelelor hardware create, în timp real. Totuşi, bibliotecile Matlab pentru Arduino nu sunt compatibile, în acest moment, cu toate platformele recente din această familie.

Există şi posibilitatea utilizării platformelor Arduino în sistemul de operare Linux, în diferite variante, în funcţie de experienţa utilizatorilor care trebuie să înteleagă aspecte legate de structura internă şi de sistemul de operare. Rularea mediului Arduino IDE sub sistemul de operare Linux necesită instalarea unor kituri de dezvoltare specifice, cum sunt Java Development Kit sau JDK, compilatoare specifice. Programarea platformei Arduino sub sistemul de operare Linux, se consideră a fi nativă, deoarece compilatoarele sunt scrise în limbajele specifice sistemului UNIX/Linux, mai precis CygWin. În Windows, mediul Arduino IDE rulează un sistem de tip „maşină virtuală" în care, sistemul „CygWin Linux shell" este virtualizat pentru a permite compilatoarelor să funcţioneze [8], [9].

Modelele create prin aplicaţile cu platforme de dezvoltare includ controlul unor acţiuni specifice mediului înconjurător. Mărimile fizice colectate sunt valori analogice, care necesită senzori şi convertoare adecvate, cu ieşiri de tip semnal electric, prelucrate prin intrările/ieşirile de pe placă, pentru a controla dispozitive şi actuatoare şi pentru a realiza comunicarea cu alte sisteme. În funcţie de complexitatea sistemului şi de

acţiunile de control prevăzute de aplicaţie, se pot utiliza diferite configuraţii, care sunt dependente de tipul plăcii, modul de programare şi de comunicare cu alte sisteme, precum şi de numărul de platforme utilizate în sistem.

2.1 Viziunea Internet of Things în contextul actual şi perspective

Odată cu dezvoltarea tehnologiilor în domeniul digital, s-a creat şi conceptul, iniţial controversat, ***IoT (Internet of Things)*** **sau Internetul tuturor lucrurilor**, considerat de specialişti ca fiind următoarea revoluţie industriala la nivelul telecomunicaţiilor, menit să schimbe atât implicarea diferitelor industrii în activitatea economică, cât şi modul de interacţiune a dispozitivelor şi a persoanelor. Conceptul IoT face referire la o reţea avansată de obiecte, în diferite domenii de aplicaţie, care conţine tehnologii din categoria sistemelor embedded şi care interacţionează prin mărimile fizice preluate de la alte dispozitive sau din mediul înconjurător, pentru a crea noi aplicaţii; iniţial, elementul comun se definea în jurul comunicării prin reţele clasice sau wireless, susţinut prin implicarea protocoalelor de comunicaţie adecvate, ca apoi să se extindă la un domeniu vast care cuprinde dispozitive, tehnologia Internet şi persoane şi dispozitive care crează împreună un ecosistem inovator, securizat, de interoperabilitate [10].

Există diverse viziuni corelate cu acest concept, care implică *multidisciplinaritate, protocoale de comunicaţie eficiente, senzori performanţi, procesoare accesibile ca preţ, software de aplicaţie corespunzător* şi care reprezintă, în primul rând**, o reţea globală de conectare a persoanelor, datelor şi lucrurilor**. În acest context, majoritatea aplicaţiilor a devenit parte din realitatea de zi cu zi, prin încorporarea tehnologiilor existente într-un circuit permanent al datelor. Aplicaţii precum încorporarea senzorilor în autovehicule, tehnologii moderne de control în domeniul biomedical, aparatura electrocasnică cu elemente de control inteligent (‘smart’) prin conexiuni Internet, sunt doar câteva exemple deja consacrate în domeniu. Fiecare element din reţeaua IoT poate deveni “inteligent” prin fluxul de date achiziţionate şi transmise prin intermediul Internetului. Dispozitivele portabile din noile generaţii transmit în timp real în orice destinaţie de pe glob, date legate de timp, viteze, distanţe, parametri medicali; sistemele de irigaţie, elementele de confort şi siguranţă în spaţiile locative pot fi controlate în timp real, totul într-o hartă reală a tuturor lucrurilor. Astfel, aceste dispozitive realizează o combinaţie între lumea digitală şi cea fizică, furnizând obiectelor („Things”) identitate şi personalitate [11] capacitate de sesizare şi de control. În acest context, eforturile sunt îndreptate înspre acţiunea de uniformizare atât din punct de vedere hardware şi software, cât şi din punct de vedere al comunicaţiilor, implicând elementele de *standardizare*, interoperabilitate, *protocoale de comunicaţie* şi

tinând cont de implicaţiile aferente la nivel de *securitate*, sociale şi legislative (de exemplu, standardele 802.101 de reglementare a radiocomunicaţiilor).

Aplicabilitatea conceptului necesită existenţa cel puţin a unei unităţi de procesare şi control; unităţile clasice de procesare de tip desktop nu fac parte din dispozitivele IoT, fiind maşini cu utilizare generală (General Purpose Machine), în timp ce caracteristicile platformelor IoT sunt legate de dimensiuni reduse, consum redus de energie. Platformele din familia Arduino sunt caracterizate de costuri reduse şi eficienţă din punct de vedere al consumului de energie; cele din familia Raspberry, deşi mai costisitoare, sunt platforme Linux de tip „stand alone", agreate în aplicaţii de control de la distanţă; platformele compatibile Arduino, cum sunt cele Intel Galileo oferă frecvenţe mai ridicate de procesare, posibilitatea folosirii pinilor analogici pentru interfaţare şi existenţa sloturi-lor încorporate pentru realizarea conexiunilor de comunicaţie prin internet (Ethernet, Wifi). Alegerea va fi facută de utilizator, în funcţie de scopul aplicaţiei şi bugetul alocat pentru implementarea acestuia.

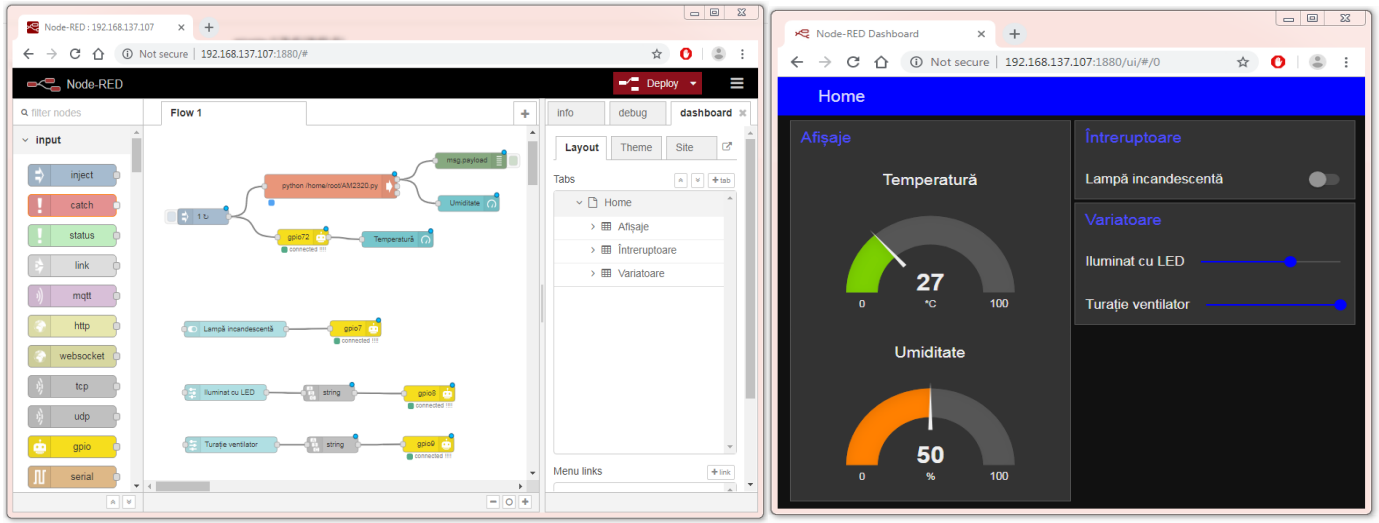


Fig. 2.1. Node-Red – panou de editare şi interfaţă web

Aspectele legate de accesibilitatea acestor resurse din domeniul IoT (preţ, consum redus de energie şi dimensiuni reduse ale hardware-ului) şi cerinţele minime de cunoştinţe legate de hardware şi software, cât şi integrarea acestora în sistemul de acces „open source” - reprezintă premize pentru orice categorie de utilizatori de a dezvolta aplicaţii de control în timp real, pe diverse niveluri de complexitate, deschizând perspective pentru înţelegerea electronicii, programării şi comunicaţiilor.

Un exemplu usor de accesat pentru dezvoltatorii de sisteme de comandă şi control îl reprezintă modulul *Node – Red* - soluție IoT relativ simplă de implementat, pe baza unui micro-computer dedicat în sfera automatizărilor (ex. RaspBerry PI sau Intel Galileo). Această soluție implică mai multe aspecte precum: - *un server web,* un *limbaj de programare universal* valabil (ex. Java script, Node.js, sau Python), o *interfață web de administrare* și un *panou de comandă și control* (eng. DashBoard). Programarea în Node – Red se realizează pe bază de blocuri și conexiuni. Schimbul de date dintre blocuri se realizează secvențial prin mesaje. Platforma Node – Red funcționează în browser atât ca pagină de administrare, cât și ca panou frontal de lucru (un exemplou de panou de editare și o pagină rezultantă – Fig. 2.1.).

2.2 Modalităţi de programare a platformelor de dezvoltare din familia Arduino (Arduino, Intel Galileo)

a) Mediul de programare Arduino IDE

Unul din elementele de noutate aduse odată cu aceste platforme de dezvoltare, începând cu prima varianta Arduino, este integrarea lor, din punct de vedere software, în categoria “open-source”; mediul dedicat IDE conţine o colecţie de elemente ce permit scrierea de programe orientate spre grafică şi este disponibil cu variante actualizate frecvent, conform versiunii de sistem de operare folosit (Windows, Linux, Mac OS X).

Partea de instalare şi configurare a platformei, cu firmware-ul şi driverele specifice, vor fi explicate, pe scurt, în continuare. Schimbul de informaţii cu computerele conectate cu platformele din familia Arduino se realizeaza prin portul USB; astfel, este nevoie de un cablu de conexiune USB la computerul gazdă și instalarea si configurarea corespunzatoare a driver-ului USB, pentru a realiza comunicarea cu computerul.

Descărcarea celei mai recente versiuni ale mediului Arduino IDE se realizează de la adresa http://arduino.cc/en/Main/Software, după care se instalează pe calculatorul personal.

Alimentarea se realizează, fie direct de la computer, prin cablul de date specific, fie prin adaptorul specific de 5 V de la producător; de obicei, alimentarea corectă este semnalizată, de un LED verde încorporat pe platformă. Dacă este cazul, întotdeauna se realizează alimentarea platformei, urmată de conectarea cablului USB la computer, respectiv pe portul USB Client al platformei.

Instalarea fizică a platformei sau recunoaşterea acesteia în mediul Arduino IDE, se poate realiza în cazul în care există conexiune la Internet, din interfaţa Arduino IDE → Tools → Boards → Boards Manager → căutare platformă în fereastra deschisă → Install → descărcarea fişierului corespunzător se va realiza automat, la fel ca şi încărcarea acestuia.

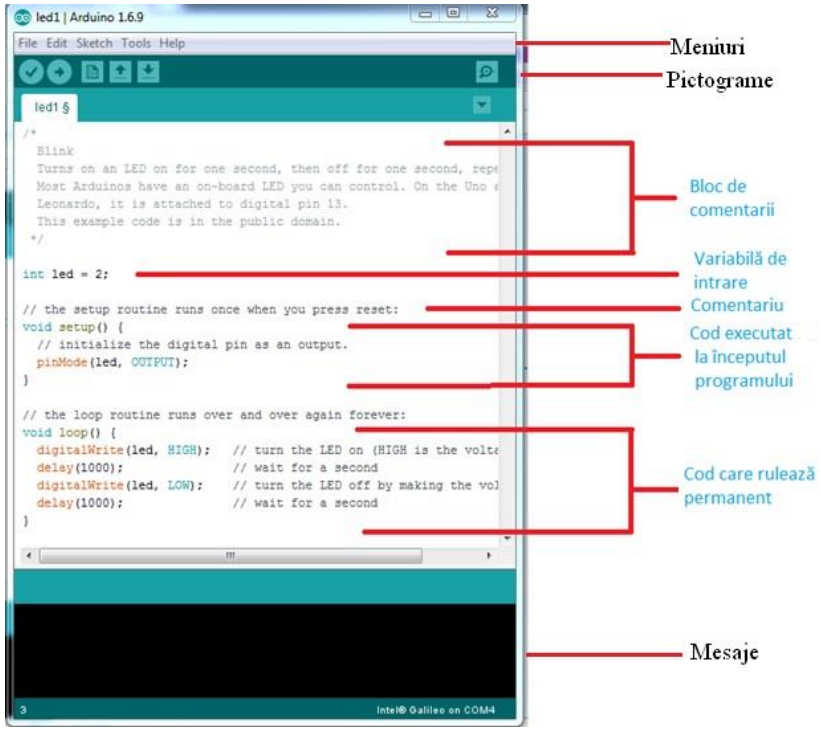


Figura 2.2. Structura de bază a mediului Arduino IDE

Computerul va încerca să găsesca driver-ul USB pentru platformă, dar, pentru a evita neconcordanţele, cea mai accesibilă metodă este instalarea manuală a acestuia. Accesând adresele web corespunzătoare, se va alege varianta de driver pentru sistemul de operare folosit pe calculator. Apoi, fişierul descărcat va fi folosit pentru a realiza actualizarea software-ului pentru driver (Update Driver Software); această acţiune se poate realiza urmând paşii: Control Panel → Device Manager → Gadget Serial v2.4 → Update Driver Software → calea fişierului descărcat.

După instalarea corespunzătoare, existenţa platformei pe unul din porturile computerului se va putea vizualiza fie în Device Manager, fie în Arduino IDE → Tools → Ports → platforma on COM X.

Din punct de vedere software, partea de implementare a programelor prin mediul **Arduino IDE** se realizează conform unei structuri de baza explicate, schematic, în Fig. 2.1.

Structura de bază a mediului Arduino IDE cuprinde:

- *Blocul de comentarii* – util pentru explicarea întregului demers matematic folosit în aplicaţie;
- *Comentariu* – util când se doreşte explicarea punctuală a folosirii unor instrucţiuni;
- *Variabilele de intrare* - la începutul programului, înainte de prima buclă se declară variabilele care urmează să fie utilizate în program
- *Void Setup* - o buclă executată o singură dată şi care include iniţializările necesare pentru execuţia programului; în cadrul ei se pot declara şi variabile şi constante necesare pe parcursul rulării programului;
- *Void loop* este o buclă executată în mod continu, într-o buclă infinită şi efectuează repetitiv operaţii de citire semnale, procesare şi generare de comenzi;

Mediul de programare Arduino conţine *biblioteci de funcţii* (proceduri) prin intermediul cărora programatorul poate să acceseze: semnale digitale de intrare şi de ieşire, semnale analogice de intare şi de ieşire, interfeţe seriale, sau alte interfeţe care pot fi ataşate plăcii Arduino. Utilizatorii cu experienta îşi pot crea propriile librării sau pot interveni în modificarea celor existente.

Acesta mai conţine şi un număr mare de *exemple de programe,* accesibile din meniu: (File → Examples). Aceste exemple au menirea de a exemplifica modul de accesare a diferitelor resurse ale plăcii şi modul de utilizare a funcţiilor din bibliotecă.

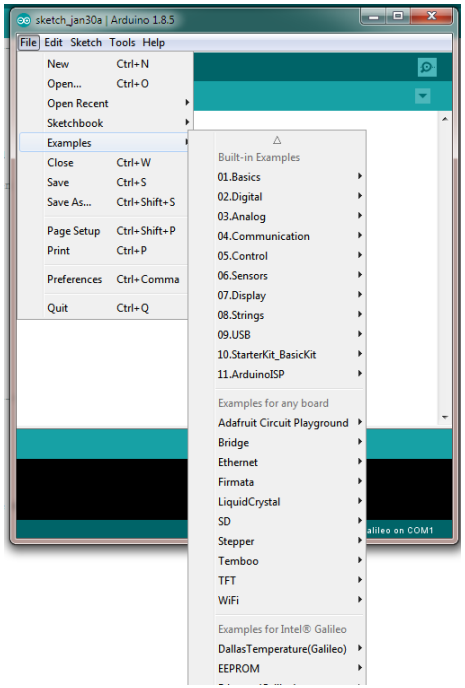


Fig.2.3. Accesarea exemplelor din meniul Arduino IDE

Programele scrise în mediul IDE Arduino, în panelul alb de editare, conform Fig.2.2., se numesc schiţe (*engl. Sketches*), care pot fi şi scrise în editor text şi salvate cu extensia .ino. Scrierea direct în zona de editare este facilă, datorită faptului că se dispune de funcţia de ‚subliniere a sintaxei', adică, se colorează cuvintele cheie din instrucţiunile limbajului de programare

In Fig.2.3. se vizualizeaza si alte zone de manipulare ale mediului IDE Arduino:

- zona de meniuri: **File, Edit, Sketch, Tools si Help**
- zona pentru comenzi rapide sau pictograme
- zona de mesaje de feed-back.

Zona de mesaje de feed-back (panelul negru) afişează eventualele erori apărute la compilare, rezultate din editarea greşită a programului sau la încărcarea executabilului pe procesorul platformei de dezvoltare.

Un element important din zona meniului se referă la configurarea din submeniul Tools, care trebuie verificată înainte de încărcarea programului scris în editor. Întotdeauna trebuie selectate elementele corecte din zona de meniu: **Tools>Board şi Tools>Serial Port.** În primul rând, selectarea dispozitivului conectat în experiment trebuie realizată corespunzător – Arduino, Intel Galileo etc. Iar apoi se selectează portul serial corespunzator, recunoscut după conectarea plăcii.

Structura de bază a unui program editat prin mediul Arduino IDE, se bazează pe scheletul alcătuit din funcţiile de baza **Setup()** şi **Loop().**

Funcţia Setup () se foloseste pentru:

- iniţializarea variabilelor şi a constantelor,
- declararea modurilor de folosire a pinilor
- iniţierea/apelarea librăriilor etc.

Această funcţie va rula o singură dată, la fiecare pornire sau resetare a platformei.

Funcţia loop() execută buclele prevazute în algoritmul implementat prin program în mod continuu, permiţând programului să fie schimbat şi adaptat; definirea de variabile este, de asemenea, posibilă şi în această buclă, în funcţie de modul de abordare propus de utilizator.

Din punct de vedere al limbajului folosit, acesta este un limbaj hibrid (*Wiring*), dedicat în special programării "hard", deoarece implică sintaxe și mnemonici încetățenite în limbajul tehnic din domeniul electric/electronic (ex. pinMode (); DigitalWrite(););

- Limbajul Wiring, stă la baza programării platformelor Arduino, dar și Intel Galileo, și se regăsește atât sub forma **Arduino IDE**, cât și sub forma unui compilator, compatibil cu Microsoft Visual Studio/Visual Basic;
- Acest limbaj a fost dezvoltat de comunitatea Open Source, și funcționează în mod nativ pe platformele de tip Unix/Linux;
- Pentru funcționarea compilatorului din limbaj Wiring în limbaj de asamblare, apoi în cod mașină, în sistemul de operare Windows, este necesară virtualizarea unui sistem de operare de tip Linux numit **CygWig →** rulat într-o așa zisă „mașină virtuală" în fundal, odată cu mediul Arduino IDE (sub sistem de operare Linux, compilarea are loc mai rapid).

Repere complexe legate de sintaxa specifică şi modul de folosire a limbajului C pentru Arduino se regăsesc în multe documente de specialitate, un exemplu fiind [12].

b) Medii de programare grafice

Există şi alternative pentru mediul de programare Arduino IDE, care oferă posibilitatea realizării logicii de comandă prin folosirea de blocuri grafice; unul dintre exemplele amintite aici este *ArduBlock* – o extensie bazată pe programarea grafică, care, în loc de folosirea funcţiilor, variabilelor şi urmărirea unei sintaxe destul de sensibile, oferă celor care nu stăpânesc suficient aceste noţiuni, folosirea blocurilor grafice prin opţiunea "drag and drop" şi construirea algoritmului, prin interconectarea adecvată a acestora. Fig.2.4.

Fig.2.4. Structura de bază a mediului ArduBlock [13]

Acest mediu specific de programare, care rulează în toate sistemele de operare (Windows, MAC, Linux) este unul foarte simplu şi potrivit pentru novicii în programare. Este considerat un 'add-on' al Arduino şi necesită instalarea mediului Arduino IDE, care va rula în spatele aplicaţiei,

oferind un avantaj în plus utilizatorilor: schiţa Arduino este automat generată pentru a fi încărcată pe platformă şi poate fi confruntată cu programul ArduBlock, facilitând, în timp, trecerea la scrierea de cod.

Alte unelte actuale permit, pe lânga realizarea şi verificarea circuitului, alegerea modului de programare – cu blocuri sau cu linii de cod, inclusiv dezvoltarea de aplicaţii în cloud, oferind utilizatorului experienţa dezvoltării unui circuit complet la nivel de simulator. Un astfel de exemplu este *Autodesk TinkerCad.*

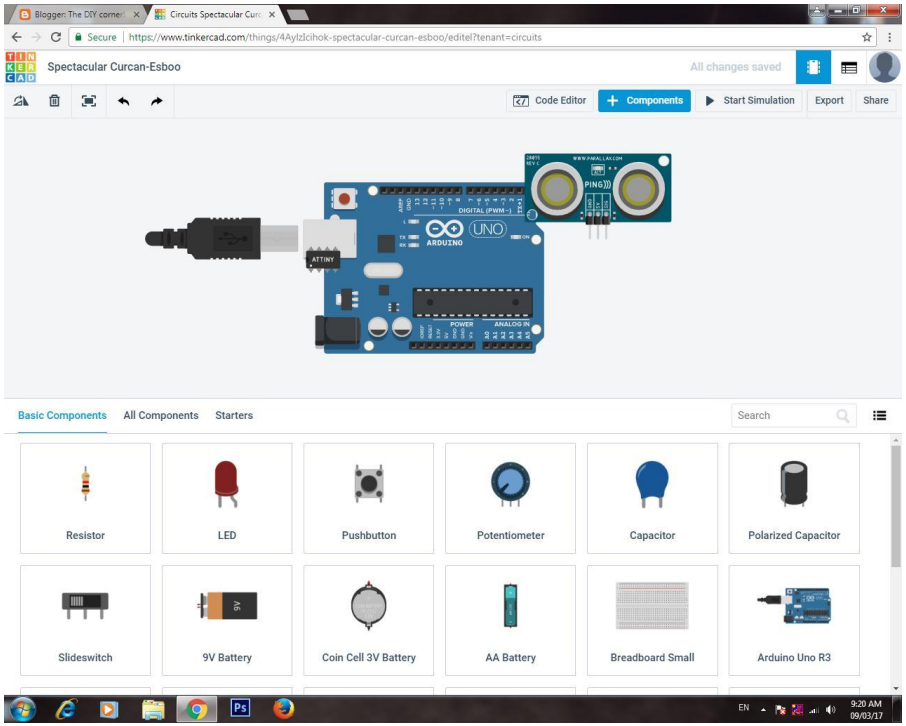


Fig.2.5. Panou de editare Autodesk TinkerCAD [14]

c**) Mediul Matlab/Simulink**

Folosirea Matlab/Simulink pentru programarea oricărei platforme din familia Arduino (Intel Galileo, în cazul aplicaţiilor prezentate în capitolele următoare), va fi detaliată pe larg în acest paragraf, dată fiind popularitatea mai scăzută a acestei metode, dar şi utilităţii folosirii ei, mai ales în domeniu ingineriei electrice şi electronice. Elementul software de bază în acest caz este pachetul ArduinoIO de la MathWorks, care facilitează interfaţarea între elementele hardware şi Simulink şi oferă comunicarea serială, în timp real pentru a realiza activităţile de control propuse. Paşii de implementare expuşi aici vor fi folosiţi, de fiecare dată, pentru oricare din aplicaţiile prezentate în capitolele următoare.

2.2.1.1 *Observaţie:*

2.2.1.2 Aceste instrucţiuni funcţionează fără a fi necesare alte intervenţii, doar pe versiunile R2014 sau mai vechi ale Matlab. Versiunile mai noi necesită unele modificări pachetului de instrumente ArduinoIO. Se recomandă folosirea versiunii Matlab/Simulink R2014.

Setări iniţiale:

- **Încărcarea programului de bază**

Pentru început, se va instala paleta de funcții/instrumente (*engl. toolbox)* aferentă platformei Arduino în mediul Matlab/Simulink. Această paletă de funcții/instrumente este compatibilă cu platforma de dezvoltare Intel Galileo (old. Arduino Galileo – iniţial a fost dezvoltat conceptul de către comunitatea Open-Source). În acest sens, se va utiliza pachetul de soft aferent platformei - Arduino IDE, pentru a încărca programul de bază (*Simulink Firmware*), în vederea realizării unei comunicații seriale permanente.

Programul încărcat pe platforma Intel Galileo are rolul de a prelua instrucțiunile vehiculate prin comunicare serială, dinspre Matlab Simulink și elementele periferice ale platformei. În pachetul arhivat *„ArduinoIO.rar”*, în directorul „*pde”* se găsesc cinci variante de program de bază sub formă de cod sursă, special destinate, pentru comunicarea cu mediul Matlab – Simulink. Se va alege varianta „*adio.pde*” din directorul „*adio*”. Extensia .pde a fost extensia inițială a codului sursă pentru ArduinoIDE (actual .ino).

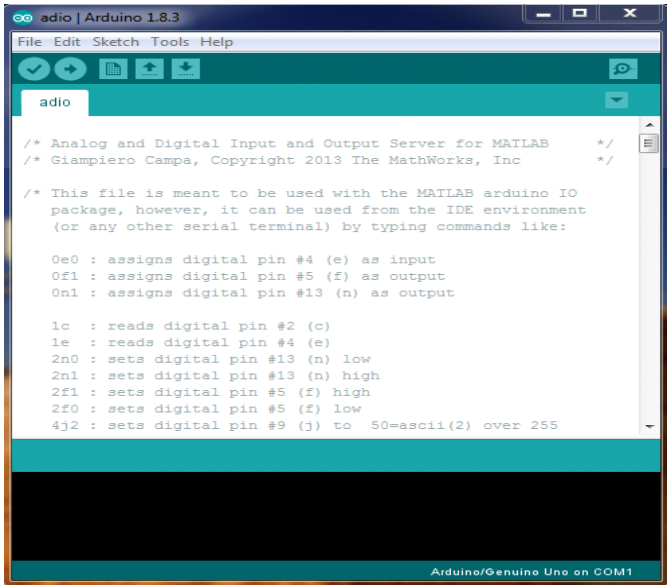


Fig.2.6. Arduino IDE- pachetul soft aferent platformei

- **Instalarea paletei de funcţii/instrumente**

Odată încărcată, această rutină de program va plasa platforma de dezvoltare folosită (ex. Intel Galileo), într-o stare de așteptare (*„ listening state”*) a comenzilor primite prin comunicare serială. Pentru a realiza interacțiunea în timp real cu platforma de dezvoltare, este necesar ca în mediul Matlab/ Simulink să fie instalată paleta de instrumente aferentă rutinei de comunicare anterior încărcată.

Fișierele necesare instalării se găsesc pe internet sub numele „ArduinoIO.rar” (de obicei arhivate). Se vor dezarhiva într-un director cu nume și cale de acces cunoscute.

Mai departe, în mediul Matlab, se va introduce ca și cale directoare (*engl. path*) implicită de lucru (Fig. 2.7.) directorul dezarhivat, în care se găsesc cele trei fișiere principale de instalare: „arduino.m”, „contents.m” și „install_arduino.m”.

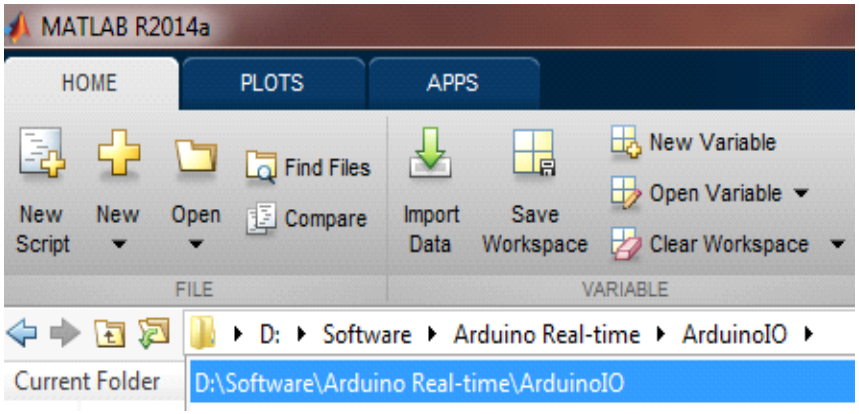


Fig. 2.7. Introducere cale directoare implicită de lucru

Odată aleasă, calea, în lista de fișiere, (conținute în director), vor apărea cele trei fișiere indicate mai sus, după cum se vede în Fig. 2.8.

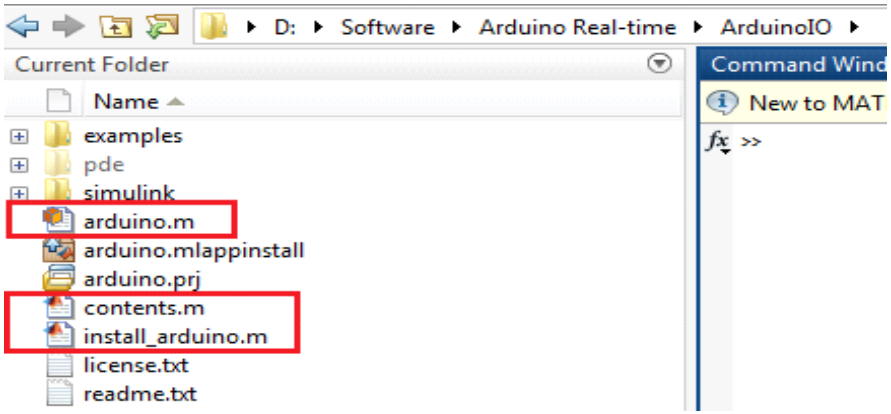


Fig.2.8. Fişiere în director de lucru

Prezența celor trei fișiere indică faptul că se poate apela comanda de instalare, fără a întâmpina alte dificultăți. Deci, în consola de comandă mediului Matlab se introduce următoarea comandă: install_arduino (Fig. 2.9).

Consola de comandă va răspunde cu următoarele mesaje:

„Arduino folders added to the path
Saved updated MATLAB path”

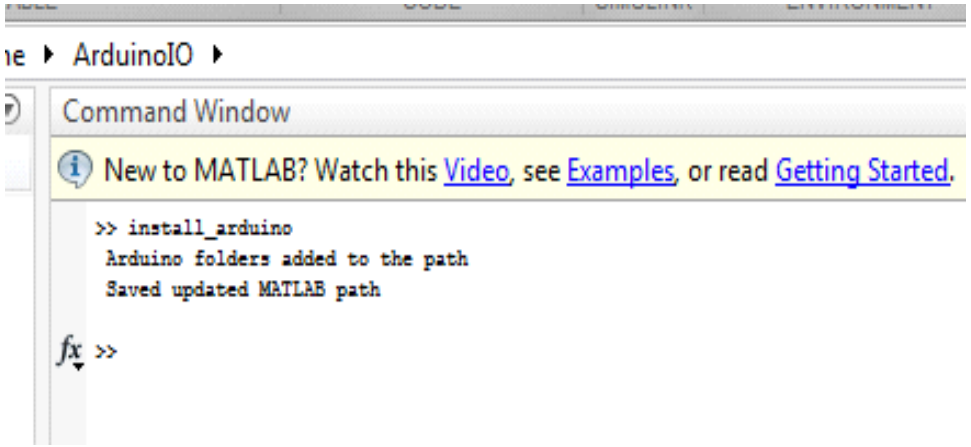


Fig. 2.9. Consola de comanda

Rezultatul aceste acțiuni constă în instalarea paletei de funcții/instrumente necesare dezvoltării de programe, cu numele „Arduino IO Library” în Simulink – Fig.2.10.

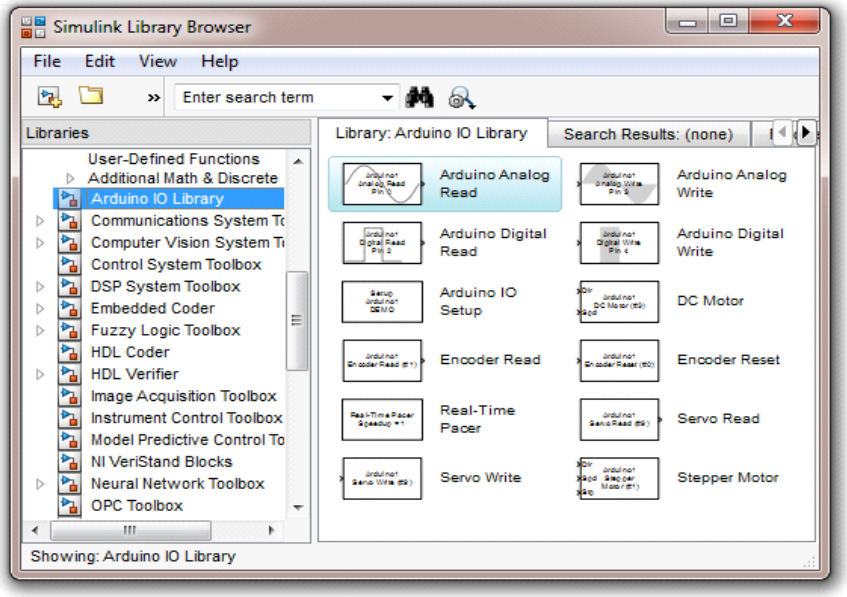


Fig.2.10. Paleta de funcții instrumente „Arduino IO Library”

- **Configurarea parametrilor de simulare**

Pentru a începe un nou model de simulare în timp real cu platforma Arduino, şi modelele compatibile (inclusiv Intel Galileo), este necesară configurarea parametrilor de simulare din mediul Simulink. Deci, se va tasta comanda: „simulink” în consola de comandă Matlab, pentru deschiderea mediului de simulare. Din fereastra nou deschisă „Simulink Library Browser” în meniul „File” se alege comanda „New” – „Model” (se va deschide spațiul de lucru). Din meniul „Simulation” se alege comanda „Model configuration parameters” – Fig. 2.11.

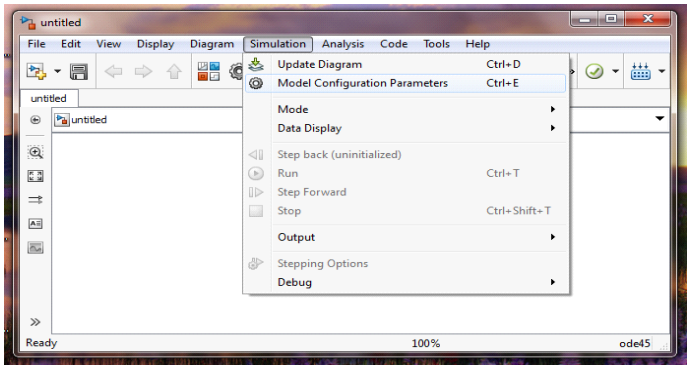


Fig. 2.11. Configurarea parametrilor de modelare

Se va deschide o fereastră în care avem posibilitatea să configurăm parametrii de simulare. În categoria „Solver options” rubrica „Type” se va alege opțiunea „Fixed-step”, iar în rubrica din partea dreaptă „Solver” se va alege opțiunea „discrete”.

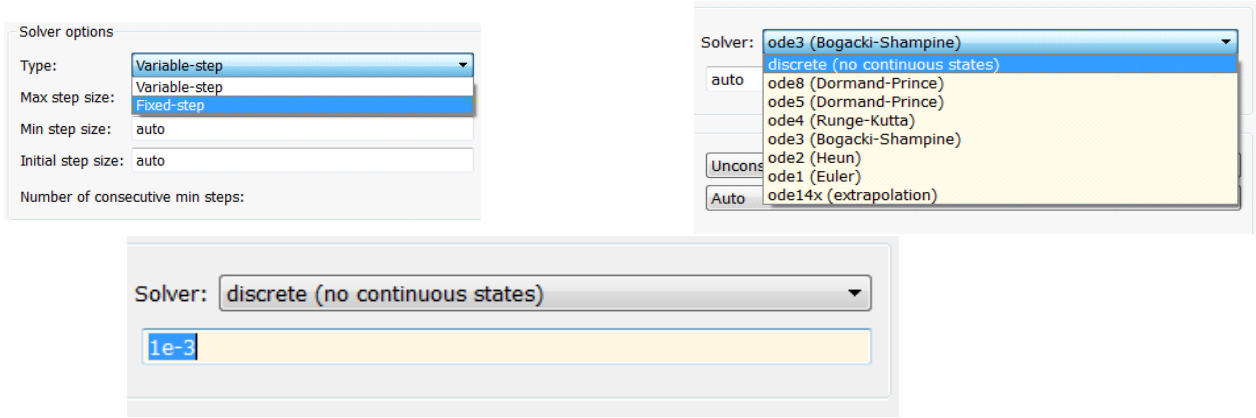


Fig. 2.12. Setările pentru tipul de modelare matematică şi timpul de simulare

Pentru pasul de integrare sau timpul de eșantionare (*sample time*) există rubrica „Fixed-step size”, unde se va înlocui valoarea implicită „auto” cu valoarea „1e-3” (care înseamnă 10^{-3} s sau o frecvenţă de simulare de 1kHz) sau „1e-4”. Pentru aplicarea setărilor se va alege butonul „Apply” apoi „Ok”. Paşii enumeraţi în acest paragraf se pot vizualiza în Fig. 2.12.

- **Setările de conectare la platforma de dezvoltare**

În continuare, se vor introduce blocurile necesare pentru comunicarea cu platforma de dezvoltare, aflate în paleta de funcții/instrumente. Un prim bloc, necesar pentru conectarea la portul serial al platformei, se numește „Arduino IO Setup”, corespunzător buclei Void Setup din Arduino IDE. După adăugarea blocului în spațiul de lucru al modelului configurat anterior, cu o comandă de tip dublu clic se va deschide ferestra de configurare a blocului respective – Fig. 2.13.

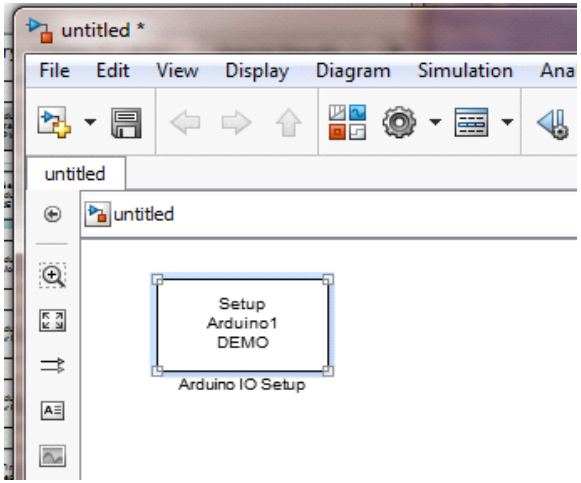


Fig. 2.13.Fereastră de configurare

În meniul de configurare al blocului „Arduino IO Setup”, se va putea schimba portul de comunicare serial. Se va alege portul pe care platforma Intel Galileo (sau Arduino) comunică cu calculatorul.
Observatii:

1. În sistemele de operare de tip Windows, porturile de comunicare serial, sunt notate cu simbolul COMx, unde „x” este indicele de ordine / prioritate. Pentru a afla indicele de ordine al portului sau numărul, se va accesa meniul „ My Computer” , opțiunea „Device manager”; o structură arboriscentă a dispozitivelor se va deschide, iar la categoria „Ports (COM & LPT)” se va regăsi și „platforma COMx”.
2. În sistemele de tip UNIX / Linux porturile serial se vor simboliza cu „ttyX”. Pentru identificarea numărului de ordine „X” există comenzi specifice în consolă, prin care se pot afla datele necesare despre port.

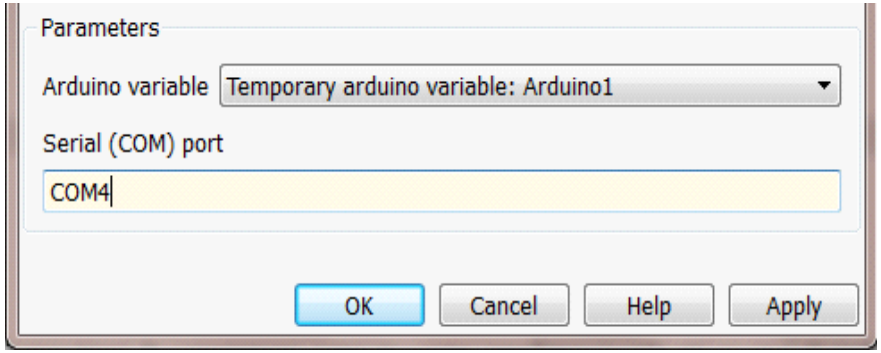


Fig. 2.14.Alegerea portului de comunicare

În cazul de față, portul de comunicare serial va fi „COM4”. Acesta se va completa în câmpul de text „Serial (COM) port” al ferestrei de proprietăți: „Arduino IO Setup”. În cazul în vor exista mai multe platforme Intel Galileo sau Arduino conectate la calculator, se vor alege din categoria „Arduino variable” „Arduino2” la ca se va declara portul dorit. În cazul de față avem doar o singură platformă.

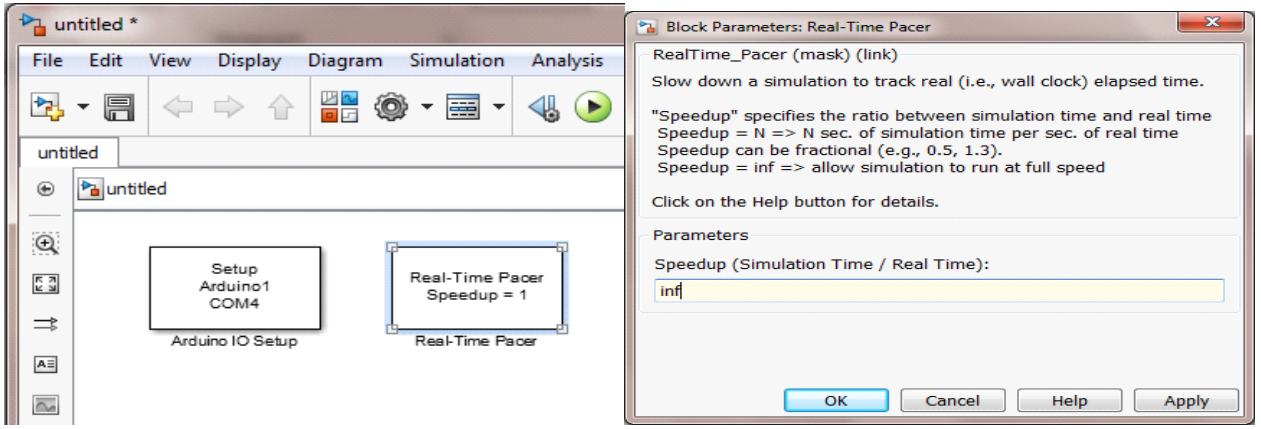


Fig. 2.15. Alegerea vitezei de simulare

Un alt aspect important, este cel al „vitezei” de transmisie/recepție dinspre platformă înspre calculator sau modul de corelare a timpului de simulare cu constanta de timp reală a fenomenului fizic. Simularea poate fi accelerată, sau încetinită în funcție de cererea utilizatorului (ex. Achiziția de date de la un acumulator care se descarcă în timp – fenomen care necesită un timp mai îndelungat, cu măsurători din „x” în „x” secunde). Acest parametru se va configura din proprietățile blocului „Real – Time Pacer” – un echivalent al buclei Void Loop.

Pentru interacțiunea la mod general cu fenomenul fizic (de obicei dinamic, nestaționar în timp), cea mai bună alegere va fi opțiunea „inf.” (infinit), deoarece fixând acest parametru la infinit, viteza de simulare va fi maximă, iar timpul de simulare va fi același cu timpul real – Fig.2.15.

d) Limbajul Python - a fost dezvoltat cu precădere pentru sistemele de operare din familia UNIX (ex. Linux). Spre deosebire de limbajul C++ sau C standard, Python are la bază reguli specifice de formatare a codului program. Câteva exemple de regului de formatare funcțională a codului pot fi considerate „de bază”:

- specificarea căii de acces înspre compilator;

- lipsa caracterelor delimitatoare ale capătului de rând;
- delimitarea secțiunilor de cod prin aliniere la stânga;
- introducerea comentariilor prin caracterul „#”;

Din alt punct de vedere, există și structuri de cod echivalente din C++ în Python:

Wiring C++ (Arduino IDE)	**Python**
`#include <>`	`import <>`
`pinMode ()`	`mraa.DIR_<IN / OUT>`
`loop ()`	`while True`
`digitalWrite()`	`write()`
`digitalRead()`	`read()`
`Serial.print(””)`	`print(””)`

Domeniul de utilizare a limbajului Python este cel al aplicațiilor cu execuție în timp real, cu precădere în domeniile în care resursele fizice sunt relativ reduse (ex. platforme de lucru încorporate). Un alt avantaj major față de celelalte limbaje, îl reprezintă faptul că, prin intermediul unor module de inter-operabilitate cu limbajul HTML (ex. Flask, NodeJS, MQTT etc.), limbajul Python permite executarea aplicației în timp real, de la distanță și controlul acesteia prin intermediul unui program de navigație web (eng. Browser). Acest proces poartă numele de asistare de la distanță și depanare (eng. *Remote Assistance and Debugging*) și constituie una din cele mai importante probleme pe care le abordează sistemele SCADA digitalizate/automatizate actuale.

Există biblioteci prin intermediul cărora se poate realiza interacțiunea în mod direct dintre aplicația program și elementele fizice de care dispune platforma de dezvoltare. De obicei, platformele de dezvoltare pe bază de microprocesor, rulează ca și sistem de operare un nucleu UNIX/Linux. De exemplu, biblioteca pe care platformele cu procesor Intel o pune la dispoziție se numește „libMRAA”. LibMRAA permite gestionarea resurselor fizice (ex. intrări / ieșiri digitale, convertor analog / digital, numărător etc.) prin intermediul limbajului Python.

Interacțiunea cu sistemul de operare care rulează pe platformă se poate realiza fie printr-o conexiune TTy Serial (eng. Tele Type) sau (COM. serial), fie prin intermediul protocolului SSh (eng. Secure Shell). Aceste protocoale de comunicație permit accesarea consolei de comandă aferentă

sistemului de operare care rulează pe platforma de lucru (echivalentul acestor protocoale în sistemele DOS / Windows poartă numele de „TELNET”). PuTTy este un program specializat în a permite conectarea la consola platformei de lucru de pe un calculator gazdă, iar pentru a crea și modifica anumite conținuturi ale fișierelor, se utilizează editorul de text pentru consolă, denumit „nano”. Acest editor poate fi apelat prin introducerea cuvântului „nano” în consola de comandă. Prin intermediul editorului se pot creea fișiere noi cu formatul / extensia dorită de utilizator (ex. nano <nume_fișier_nou>.py). Executarea unei aplicații se poate realiza prin setul de comenzi: „python <nume_fișier>.py”.

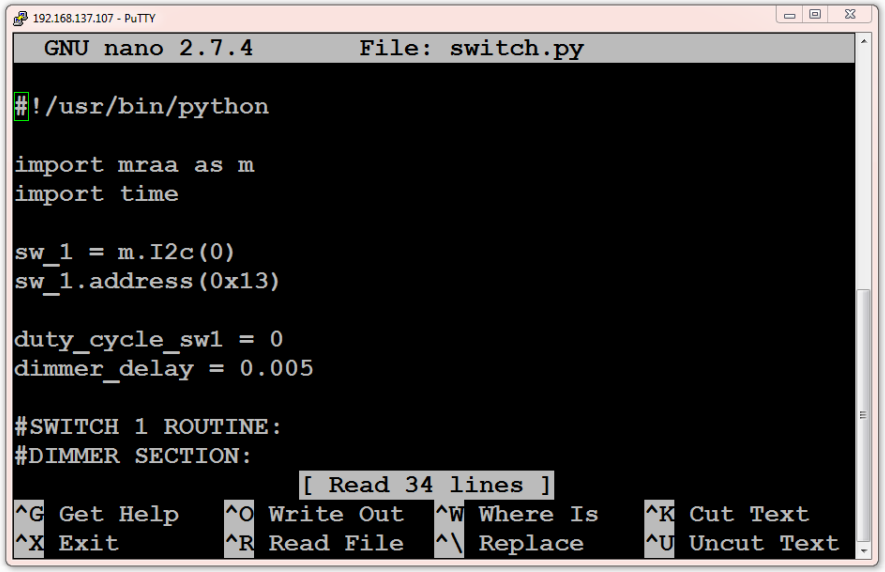


Fig.2.16. Editorul de fișiere text „Nano” pentru consolă

Un *exemplu de aplicație scrisă în limbaj Python* este redat în continuare:

```
#!/usr/bin/python          #Specificare cale absolută #interpretor Python;
import mraa                #Importare bibliotecă “mraa";
import time                #Importare bibliotecă "time";

# Setup                          #Comentariu;
x = mraa.Gpio(9)           #Adresarea ieșirii digitale;
x.dir(mraa.DIR_OUT)        #Configurarea modului de lucru al ieșirii;

# Loop
while True:                #Buclă „WHILE”;
     x.write(1)            #Activarea ieșirii;
     time.sleep(0.5)       #Așteptare 500 milisecunde;
     x.write(0)              #De-activarea ieșirii;
     time.sleep(0.5)       #Așteptare 500 milisecunde;
```

Codul programului realizează comutarea în stare activă a ieșirii digitale GPIO nr. 9 a platformei Intel Galileo, odată la 500 de milisecunde, permițând unei diode electroluminiscente atașate să semnalizeze în mod intermitent.

Limbajul Python reprezintă, aşadar, un limbaj de nivel înalt, care necesită o platformă de dezvoltare capabilă să ruleze un sistem de operare de tip UNIX / Linux, mai precis, un sistem de calcul pe bază de microprocesor.

În concluzie, **mediile actuale de programare şi uneltele software disponibile oferă utilizatorilor opţiuni nelimitate de programare a sistemelor embedded**. În timp ce marea majoritate va alege programarea folosind **IDE**-uri şi profitând de ajutorul implicit al comunităţilor **open-source**, pot fi alese şi variante la **nivel de simulator** (ex. TinkerCAD), folosind **programarea grafică**, cu blocuri. Varianta propusă utilizând mediul de simulare **Matlab/Simulink** oferă studenţilor în domeniul inginerie electrică un control mai apropiat al fenomenului. Totodată, luând în considerare avântul aplicaţiilor din sfera IoT, cu gestiunea evenimentelor controlate/comandate în timp real, cu lucrul în cloud sau prin servere web gazduite direct pe platformele folosite, marea majoritate a mediilor de programare vin însoţite de unelte software de programare web (ex. Node-Red, Wyliodrin etc.).

3. APLICAŢII DE INTERFAŢARE CU ELEMENTE DIGITALE

Utilitatea generală a microcontrolerelor ca şi circuite integrate programabile este recunoscută în aplicaţiile de interfaţare, cu scopul controlului elementelor din lumea fizică, prin adecvarea elementelor hardware şi software. Sistemele de achiziţii de date, ca şi sisteme tipice de control digital, folosesc intrările/ieşirile digitale, cu diferite capabilităţi de interfaţare; în detaliu, abilităţile şi limitările corespunzătoare intrărilor/ieşirilor digitale ale unui anumit dispozitiv de procesare embedded sunt cuprinse în manualele de producător ale acestora.

Aplicaţiile prezentate în acest capitol au scopul de a familiariza utilizatorii de platforme din familia Arduino (cu concentrare pe platforma Intel Galileo Gen 1) cu funcţionalitatea acestora de tip ‚open-source', în contextul folosirii operaţiilor de bază cu intrări/ieşiri (I/O) digitale, atât din punct de vedere hardware, cât şi software şi a modului de comunicare cu un computer personal. Aplicaţiile prezentate sunt dintre cele mai generale, pe diferite niveluri de complexitate, abordate atât prin mediul Arduino IDE şi programare în limbaj wiring, cât şi prin mediul de simulare Simulink, permiţând dezvoltări ulterioare cu acelaşi tip de elemente.

Circuitele de interfaţare cu sisteme embedded se realizează prin pinii disponibili pe fiecare platformă, cu apelarea, în program, a numelor sau numerelor afiliate acestora pe placă. Există două categorii principale de pini: Digitali şi Analogici; cei digitali au asociate doar numere (0-13), respectiv cei analogici se reprezinta prin asocierea numerelor cu litera A (A0-A5).

Pinii digitali pot fi declaraţi atât ca intrări (Input), cât şi ca ieşiri (Output), putând citi sau scrie semnale corespunzătoare valorilor 0 V, respectiv 5 V.

Declararea pinilor folosiţi ca şi intrări sau ieşiri este primul pas în folosirea acestora. Comanda generală prin mediul Arduino IDE este:

`pinMode (pinNumber, OUTPUT)` – declararea ca ieşire, respectiv
`pinMode (pinNumber, INPUT)` – declararea ca intrare.

Ulterior, **comanda digitală** a acestora în aplicaţie se realizează prin instrucţiunile de citire/scriere:

`digitalRead(pinNumber)` – returnează valoarea citită, respectiv

`digitalWrite(pinNumber, HIGH)` – transmite valoarea logică „high”, sau ‚1’ logic, corespunzătoare valorii 5V şi
`digitalWrite(pinNumber, LOW)` – transmite valoarea logica „LOW” sau ‚0’ logic, corespunzătoare valorii 0V.

Câţiva dintre pinii digitali (6 *pini digitali PWM*, în cazul platformei Intel Galileo: 3, 5, 6, 9, 10, 11) pot, de asemenea, livra semnale modulate în amplitudine (PWM), prin intermediul cărora se poate produce acelaşi efect ca şi în cazul unei tensiuni analogice (de exemplu, controlul nivelului de iluminare a unui LED).

Pinii digitali furnizează la ieşire două nivele de tensiune: 0V – LOW, respectiv 5V – HIGH; pinii cu funcţie PWM reprezintă o modalitate „hardware” de a returna tensiune de ieşire similară cu mărimea analogică corespunzătoare, în intervalul 0-5V, tocmai prin capabilitatea lor de a produce un semnal periodic, de tip puls, cu perioade diferite ale nivelelor de tensiune (*duty cycle*), „LOW’, respectiv „HIGH”. Pasul de discretizare corespunzător tensiunii maxime de alimentare a platformei şi valorii maxime digitale corespunzătoare reprezentării pe 8 biţi este calculat: 5 V/ 255. Asupra modului de funcţionare a acestor pini se va reveni în momentul aplicaţiilor specifice, care îi folosesc.

3.1 Aplicaţie de comandă temporizată a unui LED

În general, aplicaţia clasică de demonstrare a simplităţii interfaţării de componente electrice şi electronice într-un circuit cu sisteme embedded (microcontroler, microprocesor, platformă de dezvoltare) este cea de semnalizare temporizată sau ‘*Hello world!*, folosind un LED (engl. *light-emitting diode*).

În cazul de faţă, platforma utilizată - Intel Galileo generează semnalele de comanda ON/OFF pentru un sistem controlat. Ulterior, în capitolele referitoare la interfaţarea cu elemente analogice, prin intermediul platformei de dezvoltare vor fi citite semnale analogice sau nivele de tensiune de la senzori şi sistem.

Conectarea hardware a unui LED în circuitul platformei Intel Galileo

LED-ul sau dioda electro-luminiscentă are polaritate unică, fapt pentru care, se vor avea în vedere conexiunile în circuit. Identificarea terminalelor la diodă – Fig.3.1., se face prin recunoaşterea poziției catodului (adică minusul „-”), care este situat în apropierea uni mici teşituri în partea de jos a diodei.

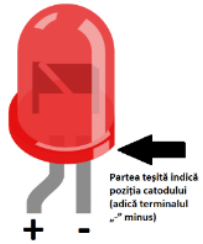


Fig.3.1. Identificarea terminalelor unui LED

Un alt aspect important este faptul că printr-o diodă curentul crește exponențial în timp în raport cu tensiunea, contrar caracterului uzual al unui rezistor la care curentul crește liniar în timp în funcție de tensiunea aplicată la borne, dupa cum se vizualizeaza in Fig.3.2.

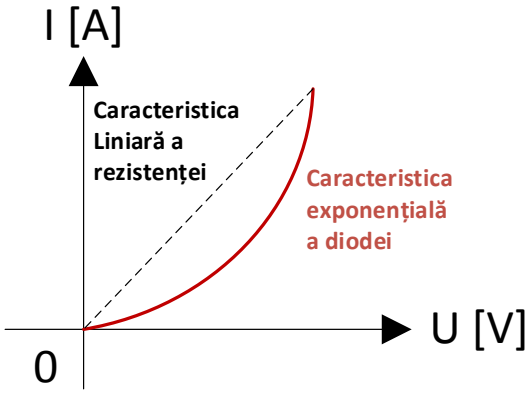


Fig.3.2. Caracteristicele electrice ale LED-ului şi rezistenţei

Din acest motiv, se are în vedere limitarea curentului prin înserierea unui rezistor cu LED-ul, pentru a proteja circuitul intern al ieșirii digitale împotriva supracurentului, care l-ar putea absorbi dioda.

Există două variante de implementare a aplicatiei cu LED-uri, prezentate în abordarea Simulink:

A. Semnalizarea cu un singur LED, cu răspuns (eng. loopback) sau monitorizarea stării;
B. Semnalizare cu două LED-uri în antifază/alternativ

Conectarea şi comanda oricărui tip de LED se poate realiza simplu, prin utilizarea unui pin digital I/O al plăcii şi înserierea unei rezistenţe potrivite, conform schemei din Fig. 3.3.Rezistenţa se foloseşte pentru limitarea curentului şi, teoretic, se alege în funcţie de caracteristicile electrice – curent de aprindere şi tensiune de alimentare - ale LED-ului; în general, valori de 220 Ω – 330 Ω sunt cele mai potrivite pentru aplicaţiile uzuale cu LED.

Pentru realizarea circuitului este nevoie de următoarele **componente**:

- Platforma de dezvoltare
- Cablu USB
- Cablu de alimentare al platformei (în cazul Intel Galileo)
- Breadboard
- LED
- Rezistenţa
- Fire

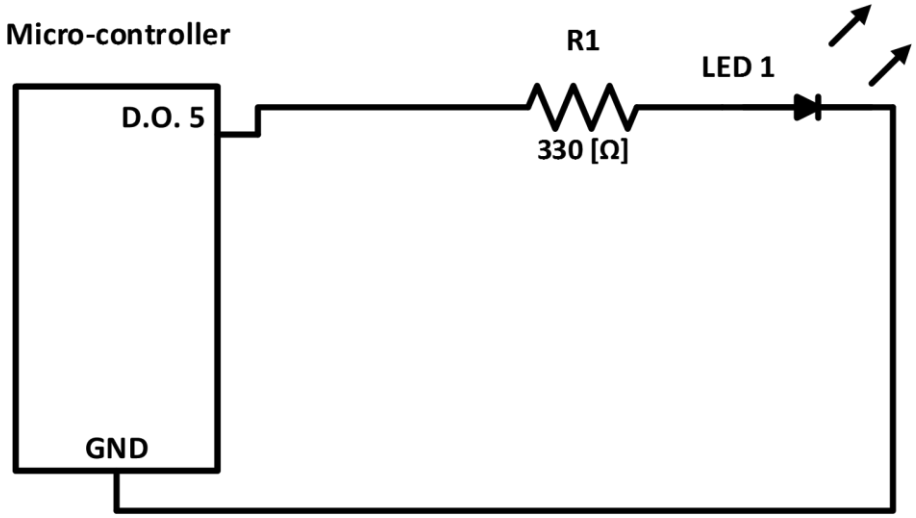


Fig.3.3. Schema electrică a montajului simplu cu LED

Din punct de vedere hardware, o altă modalitate de înţelegere a polarităţii LED-ului este din lungimile diferite ale celor două picioare: catodul corespunde piciorului mai scurt, care este conectat la pinul corespunzător masei platformei (GND). Un capăt al rezistenţei este conectat la unul din cei 8 pini digitali ai plăcii (în aplicatia de faţă, la pinul 8), iar celălalt capăt la anodul LED-ului sau piciorul lung al acestuia, prin coloanele breadboard-ului.

Din punct de vedere al declarării hardware, pinii de intrare/ieşire I/O generici pot fi manipulaţi prin comanda pinMode(pin, mode), argumentul mode indicând intrarea – INPUT, respectiv ieşirea – OUTPUT; ulterior, comanda de ON/OFF a LED-ului se face prin funcţia: digitalWrite(pin, valoare), ieşirile digitale putând genera valorile 0 V sau 5 V, după urmatorul demers:

`digitalWrite(pin, LOW)` – pin-ul respectiv este setat pe starea logica “0”, adică legare la masă, starea LED-ului OFF;
`digitalWrite(pin, HIGH)` - pin-ul respectiv este setat pe starea logica “1”, adică 5V, valoare suficientă pentru a comuta starea LED-ului pe ON;

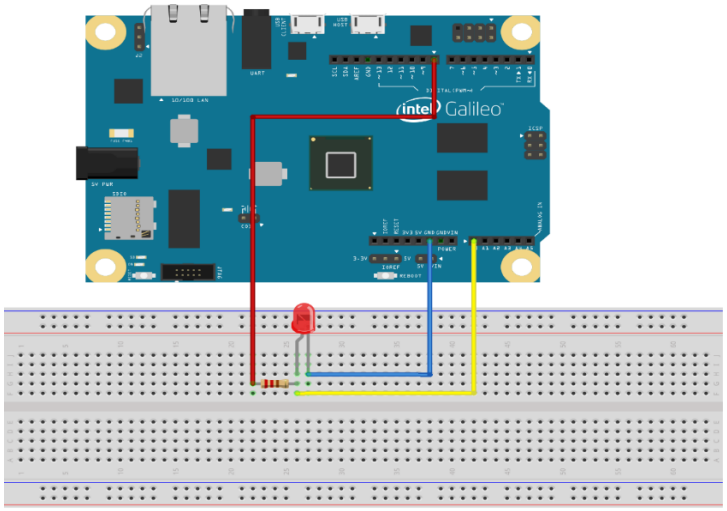


Fig. 3.4. Diagramă de montaj Fritzing pentru circuitul simplu cu LED

Prin intermediul pinilor analogici de intrare (A0-A5) poate fi masurat si un nivel de tensiune; în exemplul de faţă, rezistenţa se conectează între pinul A0 și masă (Fig. 3.4.), doar pentru a exemplifica modalitatea în care se pot primi semnale de feed-back de la diferite sisteme implementate.

Implementarea algoritmului de comanda temporizata a unui LED prin mediul Arduino IDE:

```
void setup()
{
  pinMode(8, OUTPUT); //se declara LED-ul pe pinul digital 8, ca iesire
                      // pinul 13 al placii este asignat automat unui led
                      // built-in pe platforma
Serial.begin(9600);   // deschide comunicarea pe portul serial la 9600 bps:
}
```

```
void loop()
{
  digitalWrite(8, HIGH); // Comanda On a LED-ului
  delay(1000);           // functia delay introduce o pauza, cu argumentul in milisecunde

  digitalWrite(8, LOW); // Comanda Off a LED-ului
  delay(1000);          // Pastreaza starea OFF  o secunda

  // La terminarea functiei loop() aceasta se reia
  // Urmariti cum afecteaza programul schimbarea argumentului functiei delay()

analogRead(A0); //citeste valoarea de pe pinul analogic A0
Serial.print(A0);
}
```

Corelarea circuitului din Fig. 3.3. cu secvenţa de cod ataşată conduce la funcţionarea aplicaţiei de comandă temporizată a unui LED.

Abordarea aplicaţiei folosind pachetul Matlab – Simulink:

Prima aplicaţie abordata prin pachetul MATLAB/Simulink se va prezenta în detaliu, pentru uşurinta înţelegerii modului de implementare, diferit de cel prin mediul Arduino IDE. Aplicaţiile ulterioare vor fi prezentate mai succint, din punctul de vedere al paşilor şi al elementelor folosite.

Observatie!
Pentru implementarea aplicaţiei în Simulink, se vor folosi setările iniţiale şi paşii de implementare explicaţi în capitolul 2.2 c).

A. **Semnalizarea cu un singur LED, cu răspuns** (eng. ‚loopback') sau monitorizarea stării, se realizează prin alimentarea circuitului rezistor–LED în regim de impulsuri cu o anumită frecvență. Platforma Intel Galieo este capabilă să genereze impulsuri de amplitudine 0–5 V, cu o frecvență de până la 30 Hz în modul de comunicare cu calculatorul; prin comunicare serială există limitarea de 9600 bps la viteza de transmisie (*engl. baud rate*) și până la frecvența tactului de ceas al plăcii, în mod autonom.

În vederea obținerii unui regim intermitent de alimentare pentru circuitul în cauză (LED + rezistență), se va genera un semnal dreptunghiular cu factor de umplere constant (50%), amplitudine unitară (adică „1"), iar frecvența va fi impusă de către operator. Acest semnal logic generat în MATLAB/Simulink, va constitui comanda pentru tensiunea vehiculată la ieșirea digitală (logic „1" = 5 V; logic „0" = 0 V). Forma de undă pentru un semnal astfel descris, este reprezentată în Fig. 3.7, unde: „T(s)" este perioada semnalului, iar „δ" este factorul de umplere/lățimea impulsului *(engl. duty cycle*). Deoarece este doar o aplicație simplă de semnalizare, nu se are în vedere modificarea factorului de umplere, acesta este predefinit inițial la 50%.

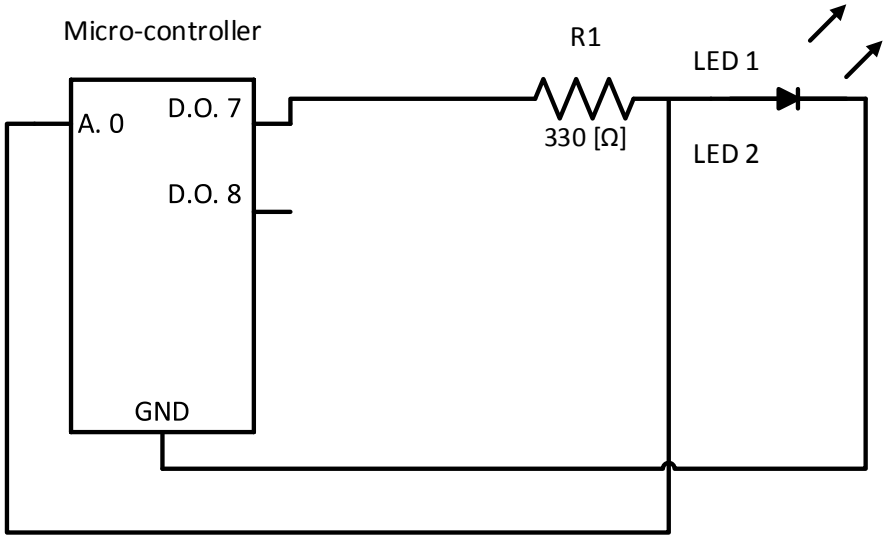


Fig.3.5. Schema electrică a montajului de semnalizare cu un LED si raspuns

În MATLAB/Simulink, pentru a produce un astfel de semnal, se utilizează blocul „generator de semnal” (*Signal Generator*), din categoria „Surse” (*Sources*) – Fig.3.7.

Se vor stabili parametrii specificați mai sus: formă de undă dreptunghiulară (*square*), amplitudine „1”, frecvență „1 - 30” [Hz] - în fereastra aferentă generatorului de semnal. Pentru sincronizare se introduce blocul „*Rate Transition”* din subcategoria „*Signal Attributes*”.

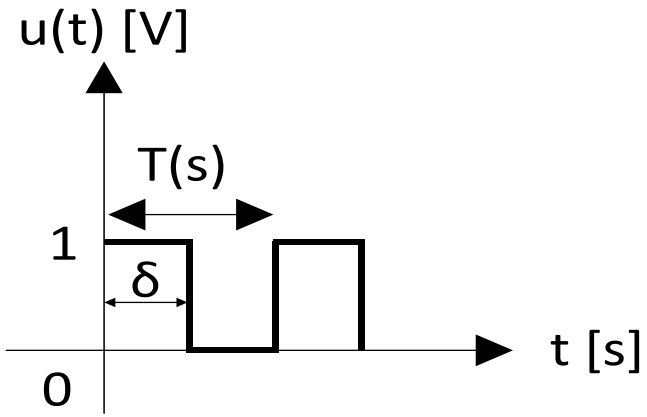


Fig. 3.6. Caracteristicile formei de unda folosita in aplicatie

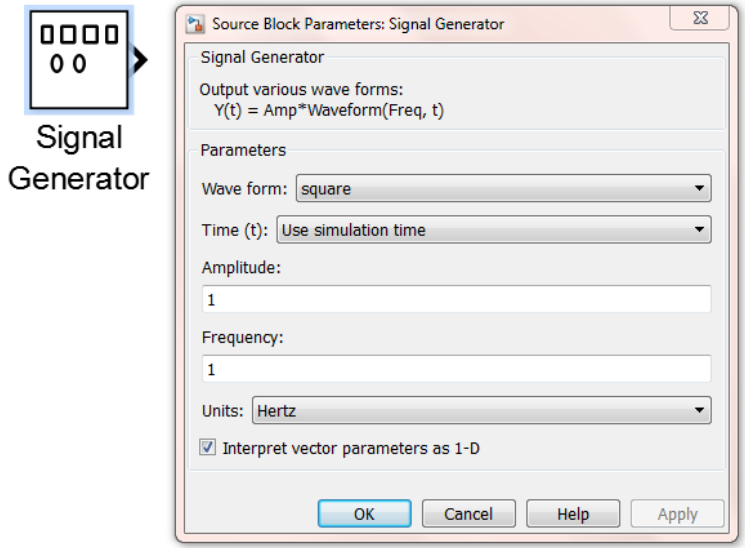


Fig. 3.7. Blocul „generator de semnal” din libraria Simulink

Pentru validarea rezultatelor, se utilizează un bloc osciloscop virtual (Fig.3.8.), din subcategoria „*Sinks*” (receptoare de semnal), pentru a vizualiza forma de undă generată de către blocul generator de semnal.

Dacă semnalul a fost generat corect conform parametrizării de mai sus, vom putea cupla ieșirea generatorului cu intrarea din blocul „scriere în ieșirea digitală” (*Digital Output Write – Arduino Digital Write*). Cu ajutorul blocurilor „Arduino Digital Write” se vor genera semnale logice la ieșirea digitală stabilită prin indice.

Se alege numărul portului digital de pe placă, precum și timpul de eșantionare la care să aibă loc comutare. Acest lucru se realizează din blocurile platformei din categoria „*Arduino IO Library*” – blocurile „*Arduino Digital Write*” (fereastra pentru configurare se obține prin comanda dublu click pe blocul respectiv - pin 9, sample time 0.001).

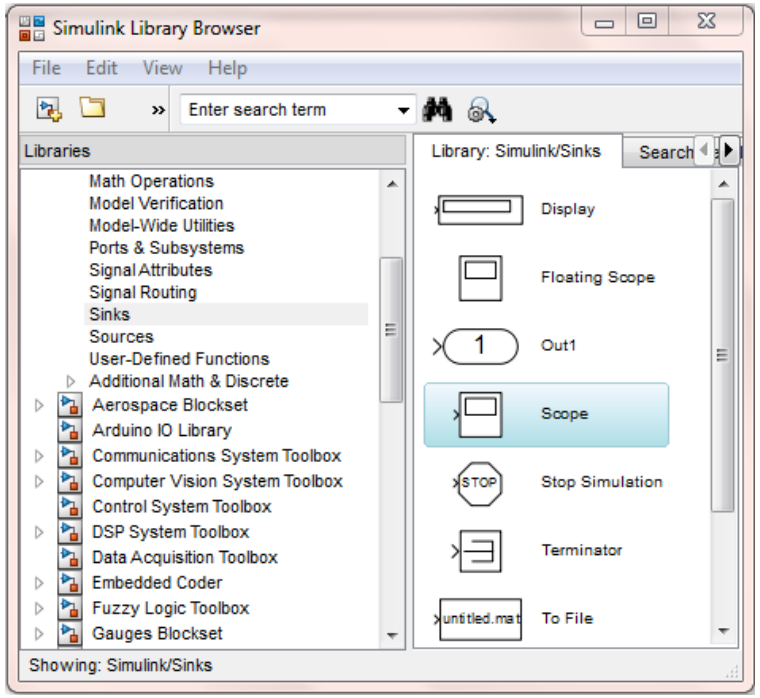


Fig. 3.8. Blocul „Osciloscop” din biblioteca Simulink

De asemenea, pentru obținerea unui semnal de răspuns de la placa de dezvoltare, se va introduce un conductor auxiliar în nodul de intersecție al rezistenței cu LED-ul, care va fi conectat la intrarea analogică, pentru a măsura nivelul de tensiune. Pentru acest lucru este nevoie de un bloc pentru „citirea semnalelor analogice” (*Analog Input Read – Arduino Analog Read)* și un bloc pentru multiplicare (*Gain - factor de amplificare*).

Considerațiile cu privire la numărul pinului corespunzător portului analogic și la timpul de eșantionare rămân la fel ca și la ieșirea digitală, cu excepția faptului că, la intrarea analogică, se va alege ca și indice de ordine/pin „0” – Fig.3.10.

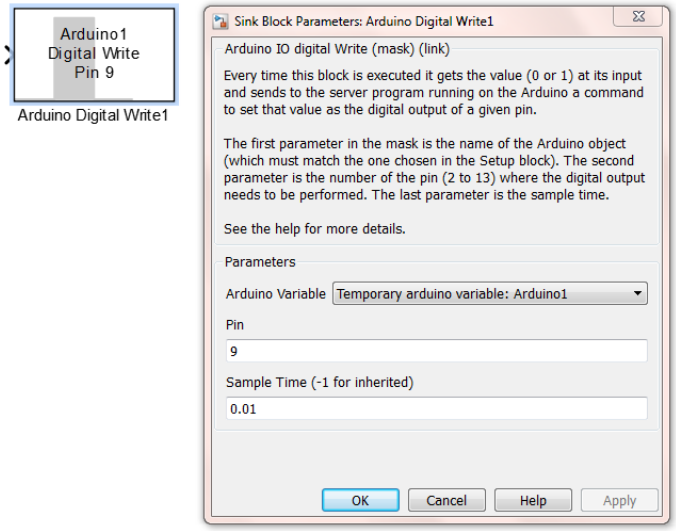


Fig. 3.9. Setarea portului digital și a timpul de eșantionare în Simulink

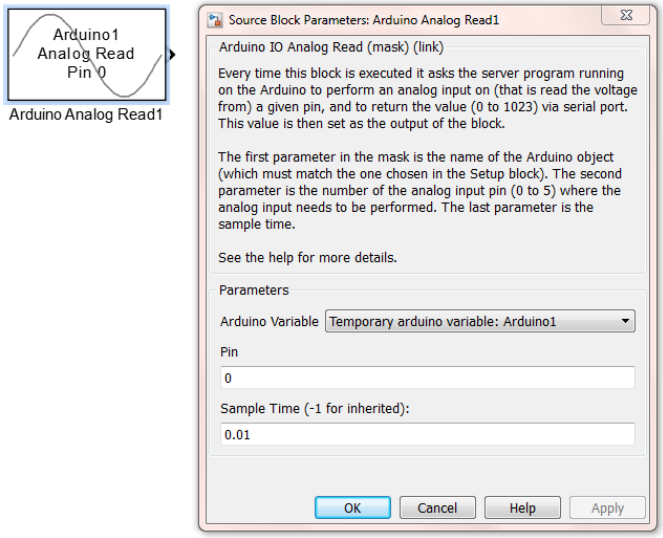


Fig.3.10. Bloc Simulink pentru citirea semnalelor analogice

Rezultatul obținut din blocul de intrare analogică, va fi un număr întreg reprezentat pe 10 biți; in detaliu, acest lucru se intampla pentru ca, pentru citirea unui semnal analogic, placa Intel Galileo, implică un convertor analog–digital cu 1024 de comparatoare electronice, având (2^{10}-1)=1023 stări, exceptând treapta de zero. În vederea obținerii valorii de tensiune corespunzatoare (adică un număr fracționar), demersul matematic apeleaza la un bloc de multiplicare, pentru a realiza conversia corespunzatoare: semnalul de ieșire din **blocul de citire analogică se inmulteste cu unitatea de conversie „5/1023=0.0048 [V/div] pe fiecare treaptă de comparație**. Astfel schema modelului matematic se prezinta in Fig. 3.11.

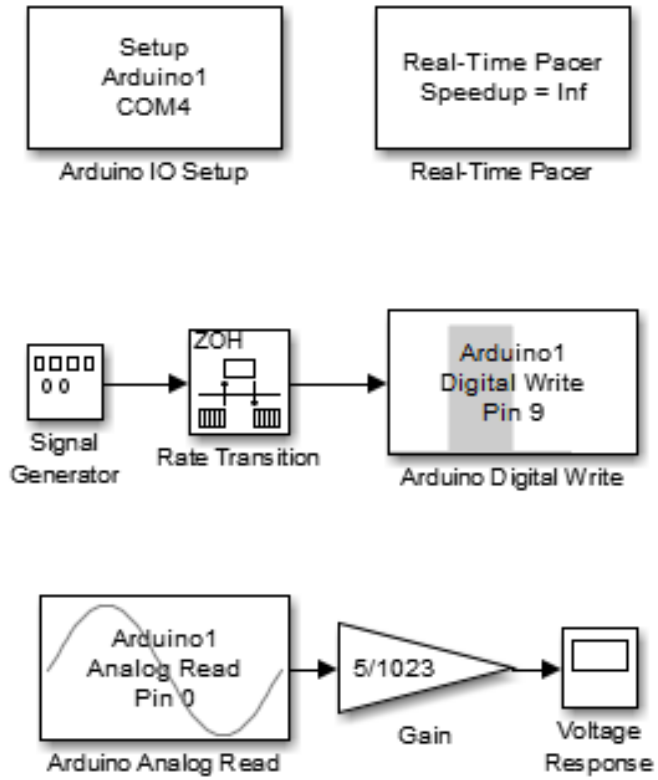


Fig.3.11. Implementarea Simulink a modelului matematic corespunzător aplicației cu LED și răspuns

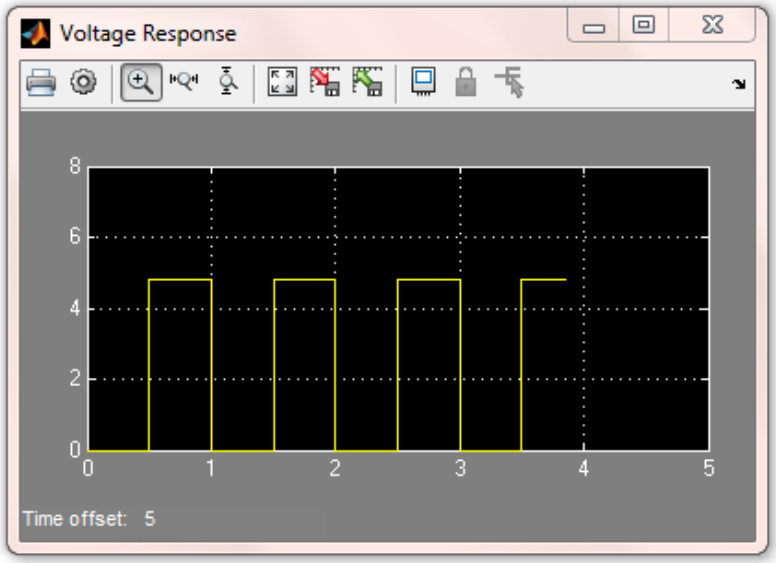


Fig.3.12. Rezultatul simulării – răspunsul platformei

În urma blocului de multiplicare se introduce in model un osciloscop virtual prin intermediul căruia se observă răspunsul plăcii (eng. loopback). Astfel, se poate realiza un sistem de comunicare bidirecțional cu verificarea stării/îndeplinirii sarcinii impuse (*status check*).

B. Semnalizare cu două LED-uri în antifază/alternativ

Această variantă a aplicaţiei aduce în plus faţă de prima variantă încă o ieşire digitală şi încă o intrare analogică. De asemenea, se va elimina bucla de verificare a stării (*check status loop*), astfel, nu mai este nevoie de intrarea analogică în aplicație.

În vederea implementării aplicației, se introduce cel de-al doilea bloc de ieșire digitală, în care se declară portul cu indicele 10 (pin 10), iar ca și timp de eșantionare la fel ca și în cazul ieșirii anterioare , 0.01.

Comutarea ieșirilor digitale în mod alternativ necesită generarea a două semnale logice complementare, unul ca și tact, iar celălalt ca și contra-tact (complementar față de primul). Acest lucru se poate obține prin complementarea semnalului principal furnizat de generatorul de semnal. Acest lucru se realizează prin funcția logică „NOT”/„COMPLEMENT”. Pentru a evita conflictele de tipuri de date (ex. întreg/logic/fracționar), se alege calea cea mai simplă de implementarea a funcție logice prin următoarea expresie matematică:

$$f(u(1)) = (u(1) – 1) * (-1),$$

unde u(1) este variabila funcției.

Se identifică două cazuri:

Pentru: u(1) = 0 => f(0) = (0 - 1) * (-1) = (-1) * (-1) = 1;
Pentru: u(1) = 1 => f(1) = (1 - 1) * (-1) = 0 * (-1) = 0;

Astfel, a fost implementată funcția logică „NOT”. În MATLAB/ Simulink, pentru a introduce o expresie algebrică - ecuație sau funcție - se folosește blocul functie „Fnc” (*Function*) din subcategoria *User-Defined Functions*. Ca și expresie, se introduce (u(1)-1)*(-1) și se realizează modelul prezentat in Fig. 3.13.

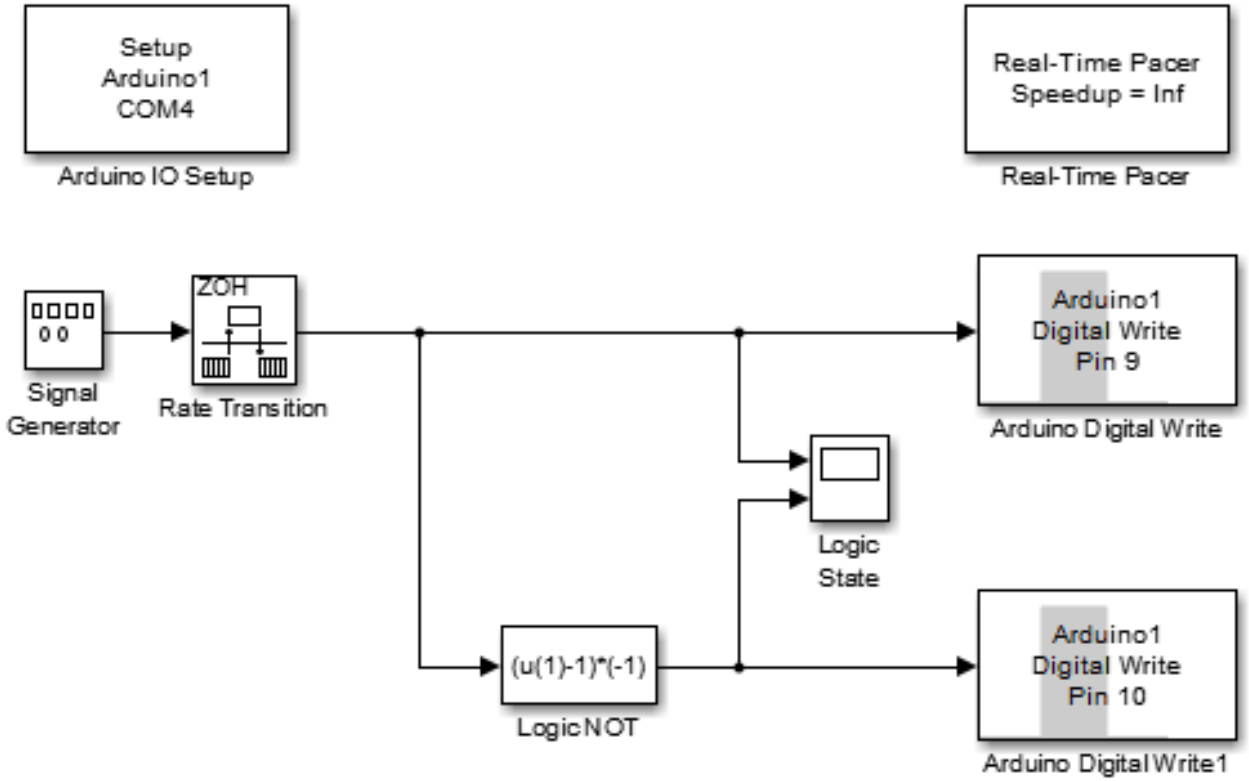


Fig.3.13. Implementarea Simulink a modelului matematic corespunzător aplicaţiei cu LED-uri în antifază

O setare suplimentară trebuie realizată pentru osciloscopul „Logic State” din schema Simulink. În acest caz se foloseşte un osciloscop cu două canale, care se obține prin parametrizarea unui osciloscop clasic, după cum se explică prin Fig.3.14. În interfața de vizualizare, în bara de sus a ferestrei, se accesează „Logic”, iar în meniul care se deschide, la caseta de text cu denumirea „Number of axes” se alege „2”.

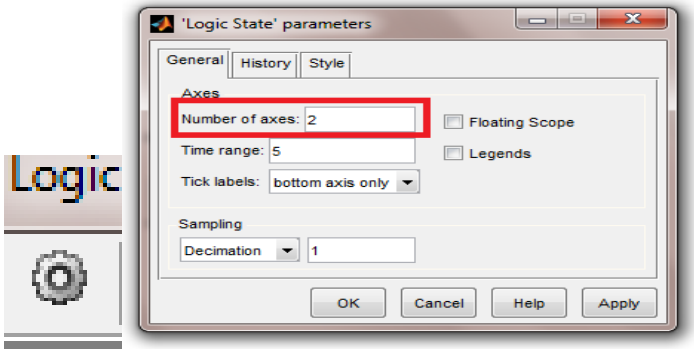


Fig.3.14. Alegerea osciloscopului cu 2 canale în Simulink

În urma implementării modelului matematic, se rulează simularea. Se obține comutarea alternantă a celor LED-uri, iar pe osciloscop, se observă cele două semnale generate complementar în contra-tact sau antifază – Fig.3.15.

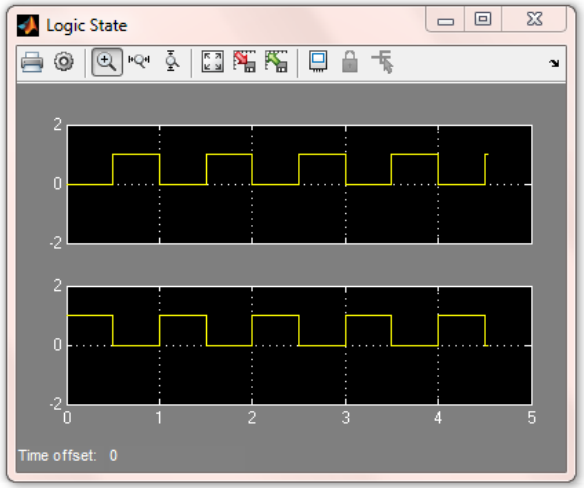


Fig.3.15. Rezultatele de simulare pentru functionarea a 2 LED-uri in antifaza

3.2 Aplicaţie de direcţionare a stării contactului înspre diodă

În acest exemplu, se propune introducerea în circuitul LED-ului a butoanelor, ca elemente uzuale de tip intrare digitală; în mod asemănător se pot folosi diverse tipuri de comutatoare (ex. mecanice, opto-electrice, dinamometric – centrifugale etc) , potrivite pentru interfaţare în aplicaţii pe pinii digitali ai platformei de dezvoltare, comanda facându-se doar prin valorile HIGH şi LOW, corespunzătoare stărilor On şi Off.

Conectarea hardware a unui element de tip buton în circuitul platformei Intel Galileo

Circuitul corespunzător aplicaţiei este prezentat în Fig. 3.16, respectiv Fig.3.17., corespunzătoare celor două cazuri particulare de topologii prezentate în continuare:

- Primul caz: atunci *când butonul este apăsat, tensiunea la bornele de intrare ale circuitului de comparare este egală cu tensiunea furnizată de sursă 5V, iar în caz contrar intrarea circuitului de comparare este pusă la masă prin intermediul unei rezistențe. Starea se numeste activ pe plus (engl. „active high"), adică, prin acționarea butonului sau contactului, se furnizează nivelul maxim de tensiune la bornele circuitul de comparare.* Rezistența are rolul de a pune intrarea comparatorului la masă (*engl. pull down* - adică în starea de înaltă impedanță), dar totodată are și un rol de protecție, deoarece nu permite potențialului 5V să ajungă în conexiune directă la borna de potențial zero, adică la masă, scurt-circuit. Cele patru terminale ale butonului sunt conectate două câte două, pe coloanele vizibile, în general, pe partea din spate a elementului, deci adaugarea în circuit trebuie realizată corespunzător (butonul conține în capsulă o pereche de contacte independente).

- Al doilea caz: dacă se alege schema cu rezistență de punere la sursă (*engl. pull–up*), *când butonul este apăsat, se realizează conectarea la masă, practic, șuntarea (engl. bypass) circuitului de comparare aferent intrării digitale, iar când butonul nu este apăsat, rezistența de „punere la sursă" furnizeaza 5V la intrarea comparatorului în permanență*. Asemănător cazului precedent, rezistența are un dublu rol, anume: pentru a proteja împotriva fenomenului de scurcitcuit la apăsarea butonului, dar și pentru a asigura starea de „înaltă impedanță" la bornele circuitului de comparare.

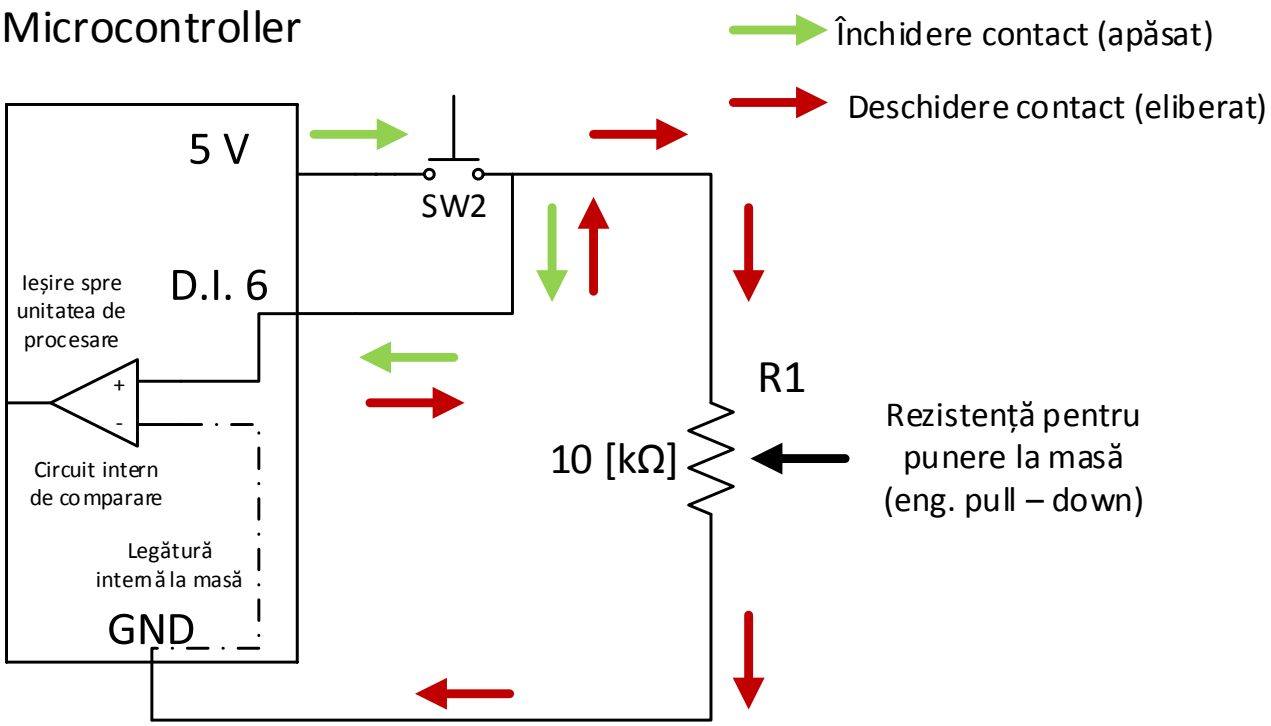


Fig. 3.16. Schemă de principiu – rezistența pentru punere la masă („pull–down resistor”)

În ambele cazuri, în urma „citirii” de către circuitul comparator a stării intrării digitale, se obține un rezultat logic (activ/inactiv; pornit/oprit; logic 1 sau logic 0). Pe baza rezultatului furnizat de către ieșirea circuitului comparator, se vor putea elabora diverse strategii de manipulare ale semnalului și algoritmi de comandă pentru activarea sau dezactivarea unei ieșiri digitale. Pentru a sesiza fenomenul se utilizează un LED, pentru a furniza semnale luminoase la acționarea butonului din circuit, în ambele cazuri (cu rezistență de punere la masă, și cu rezistență de punere la sursă).

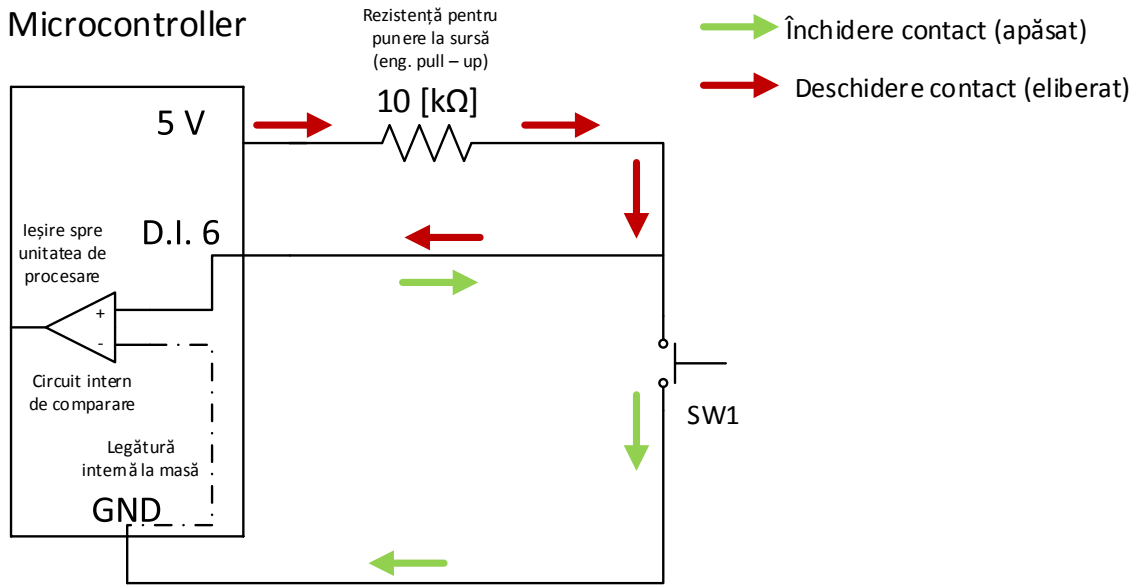


Fig. 3.17. Schemă de principiu – rezistența pentru punere la sursă („ pull–up resistor”)

Pentru realizarea circuitului este nevoie de următoarele **componente**:

- Platforma de dezvoltare
- Cablu USB
- Cablu de alimentare al platformei (în cazul Intel Galileo)
- Breadboard
- 2 LED-uri
- Rezistente: 330 Ω, respectiv 10kΩ
- 2 butoane
- Fire

Aplicaţia de faţă presupune implementarea topologiei din Fig.3.17, cu rezistenţa de tip „pull-down”, folosind două butoane şi două LED-uri; se foloseşte corelarea stării fiecărui buton cu câte un LED, astfel, încât: ‘buton1’ apăsat activează un LED (ON), respectiv ‘buton2’ activează un LED roşu (ON).

Implementarea algoritmului de comandă prin mediul Arduino IDE:

```
// declararea constantelor
const int buton1 = 2;  // butonul 1 pe pinul digital 2
const int buton2 = 3;  // butonul 1 pe pinul digital 2
const int led_rosu =  10;    // LED rosu pe pinul 10
const int led_galben =  8;    // LED rosu pe pinul 8

void setup()
{
  // declararea butoanelor ca intrari
  pinMode(buton1, INPUT);
  pinMode(buton2, INPUT);
  // declararea LED-urilor ca iesiri
  pinMode(led_rosu, OUTPUT);
  pinMode(led_galben, OUTPUT);
  Serial.begin(9600);
    }

void loop()
{
// declarare variabile: starile butoanelor
    int Stare_buton1, Stare_buton2;
  // citirea valorilor celor 2 variabile
```

```
    Stare_buton1 = digitalRead(buton1);
    Stare_buton2 = digitalRead(buton2);

if (Stare_buton1==LOW) {

//activeaza LED-ul rosu
  digitalWrite(led_rosu, HIGH);
  digitalWrite(led_galben, LOW);
  Serial.println("Buton1 este apasat");
  delay(500);
}
  if (Stare_buton2==LOW) {

//activeaza LED-ul galben
  digitalWrite(led_galben, HIGH);
  digitalWrite(led_rosu, LOW);
  Serial.println("Buton2 este apasat");
  Serial.print(Stare_buton1);
  delay(500);
}
  else {

//dezactiveaza oricare din cele doua LED-uri
  digitalWrite(led_galben, LOW);
  digitalWrite(led_rosu, LOW);
     }}
```

Abordarea aplicatiei folosind pachetul MATLAB/ Simulink

În vederea implementării aplicației de mai sus în mediul MATLAB/ Simulink, este necesar să se urmărească pașii de configurare a plăcii - încărcarea programului de comunicare prin intermediul Arduino IDE, stabilirea portului de comunicare serială, configurarea parametrilor de timp și inițializare - stabilirea indicelui de ordine pentru porturile/pinii de intrare/ieșire. În acest sens, se recomandă revederea aplicațiilor anterioare, mai precis a pașilor de configurare și inițializare prezentati in capitolul 2.2.c).

Modelul matematic aferent aplicației, implementat prin mediul Simulink, are structura prezentata Fig.3.18.

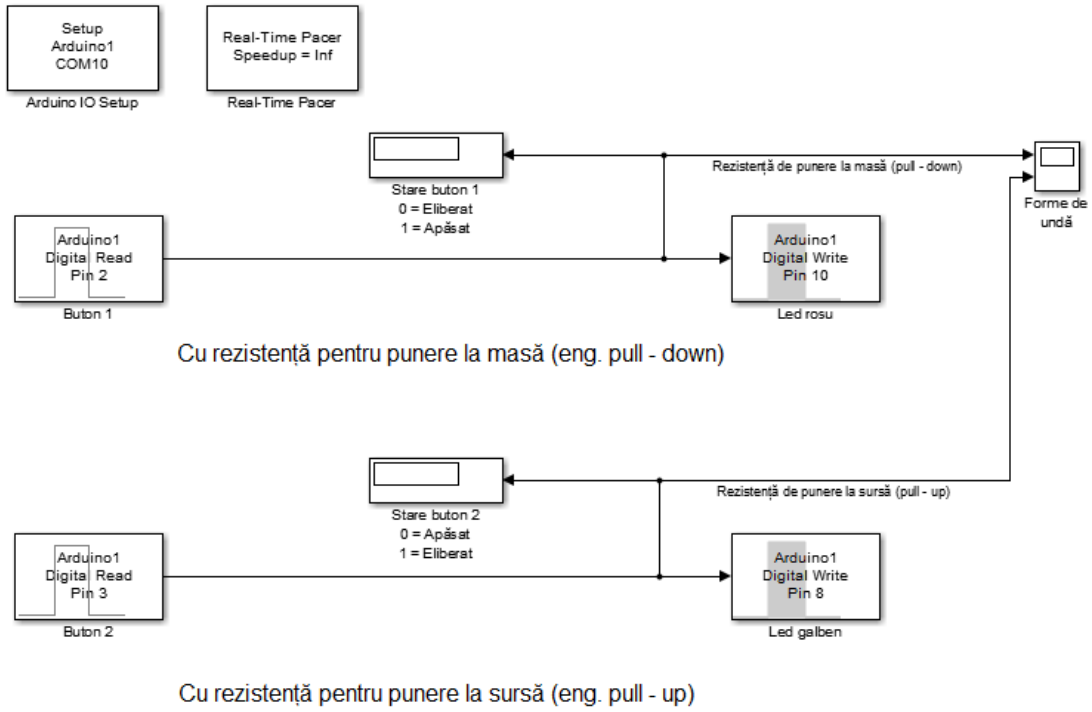


Fig.3.18. Implementarea Simulink a modelului matematic corespunzător aplicaţiei de direcţionare a stării contactului înspre diodă

Acesta conține două blocuri de preluare a valorii numerice de la circuitele de comparare ale intrărilor digitale și două blocuri de ieșire digitală (care au rolul de a furniza o stare logică, electrică sau fizică). Osciloscopul virtual și afișajele numerice au rolul de a prelua în calculator datele de la microcontroler în timpul simulării. Practic, acest model arată modul de funcționare al intrărilor/ieșirilor digitale și modul de manipulare al semnalelor logice furnizate. *Funcționarea acestei aplicații* se rezumă la următoarele aspecte:

- când butonul „1” va fi apăsat, pe afișajul virtual numeric, va apărea „1”, forma de undă de la osciloscop va indica o trecere de la starea de „logic 0” la starea de „logic 1”, iar LED-ul roșu va fi activat. (comportament tipic montajului cu rezistență de punere la masă – „active high” – pull down resistor – redă un caracter de contact normal deschis butonului);

-când butonul „2” va fi apăsat, pe afișajul numeric, va apărea „logic 0” (adică dintr-o stare de continuu activ, va trece în stare de inactiv la apăsarea butonului), forma de undă de pe osciloscopului în acest caz, va indica trecerea din starea de „logic 1” în starea de „logic 0”, iar LED-ul galben va fi inactiv la apăsarea butonului. (comportament tipic montajului cu rezistență de punere la sursă – „active low” – pull up resistor – redă un caracter de contact normal închis butonului).

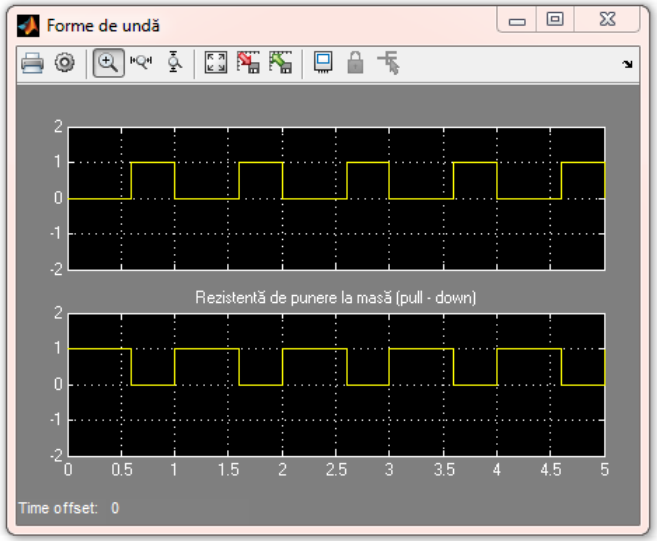


Fig.3.19. Rezultatele de simulare pentru functionarea aplicatiei cu butoane si LED-uri

3.3 Aplicaţie de comandă LED-uri multiple

Aplicaţia de faţă propune diferite secvenţe de comandă a 8 LED-uri, introducând câteva elemente referitoare la partea de programare – *bucle de tip for()* şi folosirea *matricelor de variabile*; ulterior, testarea diferitelor modalităţi de comandă ON/OFF a LED-urilor poate fi experimentată de utilizatori prin modificarea secvenţei de cod, folosind diferite tipuri de condiţionări şi bucle în algoritm. Un exemplu practic al folosirii de LED-uri multiple este cel al afisajelor cu derularea zonelor marcate pentru informaţiile utile.

Aplicaţia este dezvoltată și în scopul înţelegerii bazelor de numeraţie: transformarea, de exemplu a unui număr zecimal în baza de numeraţie binară, marcând, cu ajutorul LED-urilor ON, biţii de 1 corespunzători octetului calculat.

Pentru realizarea circuitului este nevoie de următoarele **componente:**

- Platforma de dezvoltare
- Cablu USB
- Cablu de alimentare al platformei (în cazul Intel Galileo)
- Breadboard
- 8 LED-uri
- 8 Rezistente: 330 Ω, respectiv 10kΩ
- Fire

Conectarea componentelor în circuit se face conform Fig. 3.4, ţinând cont de indicaţiile din Aplicaţia 3.1.

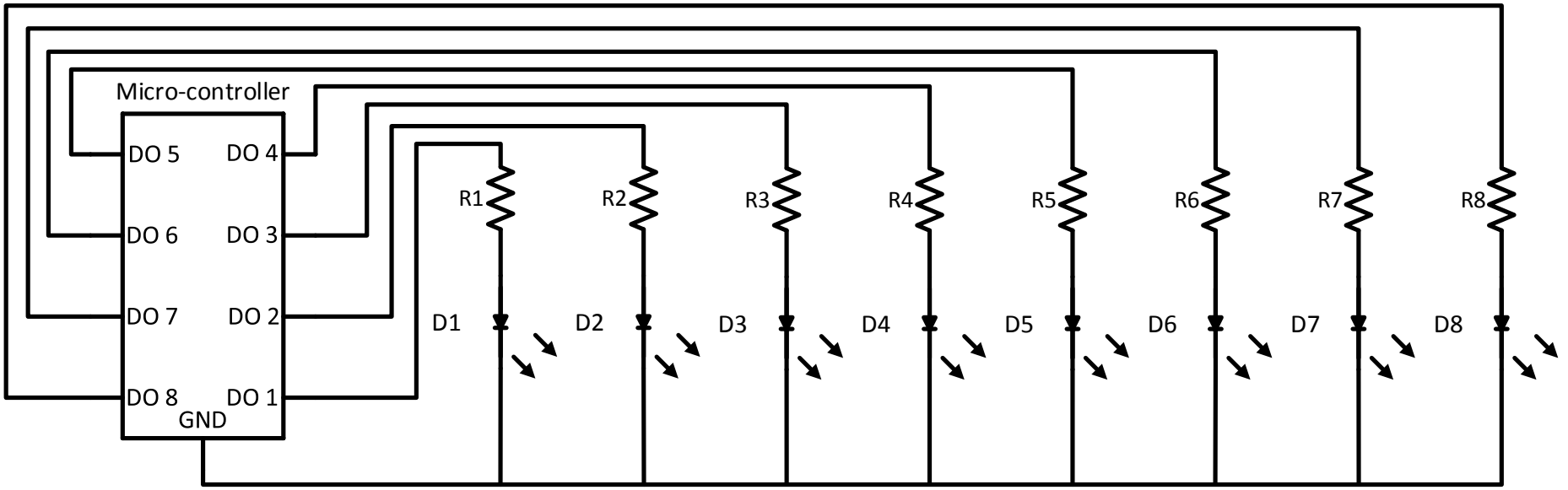


Fig. 3.20. Schema electronică pentru circuitul cu opt LED-uri

Pentru a evita declararea consecutivă a 8 LED-uri, pe 8 pini digitali ai platformei Intel Galileo, de tipul: **`pinMode(pin, mode),`** respectiv asocierea consecutivă a stărilor ON/OFF a acestora, de tipul: **`digitalWrite(pin, valoare`**), se foloseşte declarea variabilelor într-o matrice de valori întregi, cu 8 elemente, de exemplu:

```
int led_pins[] = {5,6,7,8,9,10,11,12};
```

Fiecare element din matrice este accesat în program prin poziţia acestuia în matrice, începând cu poziţia 0, până la poziţia 7. De exemplu, pentru a declara LED-ul de pe pinul 10 ca fiind ON, se comandă:
`digitalWrite(ledPins[5], HIGH);` corespunzătoare poziţiei 5 din matricea declarată anterior.

Aplicaţia foloseşte buclele de tip for(), pentru a realiza cei 8 paşi de declarare, respectiv de comandă corespunzători celor 8 LED-uri; acest tip de bucle este util atunci când numărul de paşi este cunoscut la începutul algoritmului, iar cele trei condiţii necesare realizării ei sunt cuprinse între paranteze, astfel:

```
for(expresia1, expresia2, expresia3);
```

urmată de paranteze {}, în interiorul cărora se execută partea controlată prin buclă.

- expresia 1 – în general, iniţializează variabila care controlează bucla;
- expresia 2 – execută teste logice prin care se determină trecerea la pasul următor
- expresia 3 – schimbarea stării variabilei de control al buclei, care, în mod neconvenţional, poate fi trecută şi în corpul buclei [12] ;

În cazul acestei aplicaţii, se va declara o variabilă 'position', de la 0 la 7, corespunzătoare locului din matrice a fiecarui element. Programul, prezentat în continuare, cuprinde *3 bucle for():* una pentru a iniţializa LED-urile pe pinii plăcii şi două pentru secvenţa de comandă a acestora, şi anume declararea ON consecutivă, de la poziţia 0 la 7 a LED-urilor, respectiv comanda OFF consecutivă, în sens invers a acestora.

Implementarea algoritmului de comandă a 8 LED-uri prin mediul Arduino IDE:

```
//declararea LED-urior pe pinii digitali si a variabilelor folosite
int led_pins[] = {1,2,3,4,5,6,7,8};
int position;
int del = 100;//declarare întârziere
// perioada de comanda, in ms poate fi schimbata pentru comanda // mai rapida sau mai lenta
void setup()
{

  for(position = 0; position <= 7; position++)
    {
      pinMode(led_pins[position],OUTPUT);
      // led_pins[position] corespunde, consecutiv,
```

```
      //fiecarui element din matrice
   }
}

void loop()
{
  // COMANDA ‘ON’ A LED-urilor:

  for( position = 0; position <= 7; position++)
    {
    digitalWrite(led_pins[position], HIGH);
    delay(del);
   }

  // COMANDA ‘OFF’ A LED-urilor, in sens invers:

  for(position = 7; position >= 0; position --)
    {
    digitalWrite(led_pins[position], LOW);
    delay(del);
  }
}
```

Aplicaţia poate fi testată, în funcţie de aplicaţie, pentru diferite secvenţe de comandă, cum ar fi: comanda fiecărui LED în parte, în ambele direcţii, comanda controlată pe diferite poziţii (din 2 în 2, pe poziţia 4 etc), comanda aleatoare a LED-urilor, folosind funcţia **`random(); position=random(8)`** va selecta o valoare de la 0 la 7 din matricea declarată.

3.4 Aplicaţie de comandă LED-uri multiple – afişarea în binar a unui număr zecimal

Folosind acelaşi circuit, cu 8 LED-uri şi prin adaptarea corespunzătoare a programului, se propune implementarea unei aplicaţii privitoare la trecerea dintr-o bază de numeraţie în alta (mai precis din zecimal în binar), folosind 8 LED-uri ca şi mijloace de semnalizare a stării fiecărui tact de numărare, cu semnalizarea stării fiecărui tact de numărareŞ

-Se defineşte un şir discret de numere de la „0” la „255” (adică 256 de combinaţii 2 ^ 8 = 256 - un număr reprezentat cu 8 combinaţii de tact), cu pas de incrementare 0.001. Acest lucru se poate realiza cu ajutorul unui bloc generator de semnal rampă, la care se va defini parametrul „pantă de creştere” *(engl. „rising slope” sau „slope”)* ca având valoarea de „0.001”. Se va realiza conversia/rotunjirea la întreg a valorilor fracţionare produse de către generatorul de rampă, prin intermediul unui bloc de conversie la întreg cu reprezentare pe 8 biţi. Prin această reprezentare a numărului fracţionar în întreg pe 8 biţi se schimba pasul de incrementare de la „1” în „1”, odată la o secundă de calcul în timp real; aceasta înseamnă că, atunci când semnalul ia valori de ordinul 0.001, acestea vor fi ignorate, iar când partea întreagă se modifică, 1,001, se va lua doar partea întreagă în considerare, rezultând o întârziere de o secundă în procesul de numărare, dar şi o selecţie doar din „1” în „1” a valorilor din şir.

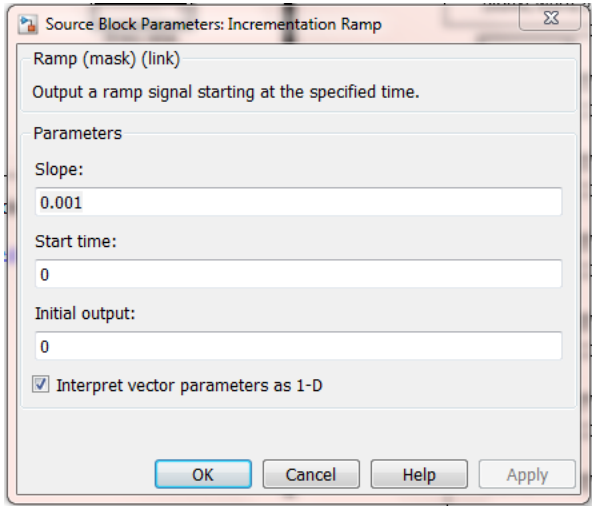


Fig. 3.21. Configurarea parametrilor pentru blocul generator de semnal rampă

-Valorile discrete obținute, variază în intervalul „0 – 255”. Pentru a realiza „numărarea” în binar, se va utiliza blocul *„Integer to bit converter”*, care, generează o matrice unidimensională sau un vector de 8 valori binare. Rezultatele pot fi vizualizate pe un afișaj numeric sau pe un osciloscop virtual.

-Pentru a comanda ieșirile digitale ale unui microcontroler, se vor separa valorile din vectorul astfel generat. Acest lucru se va realiza cu ajutorul unui *bloc de-multiplexor (eng. de-mux*), iar fiecare rezultat se va furniza la ieșirea digitală corespunzătoare – Fig. 3.26.

Configurarea parametrilor corespunzatori algoritmului prezentat, se vizualizeaza in Fig.3.21-3.23., iar diagrama modelului matematic în Fig.3.24.

-când procesul de numărare ajunge la valoarea „255” prin intermediul unui comparator se va genera un semnal logic „ADEVĂRAT” (*eng. TRUE* – condiție satisfăcută), care va ajunge la un bloc de oprire a simulării. Practic, la valoarea de 255 se va opri în mod automat procesul de numărare.

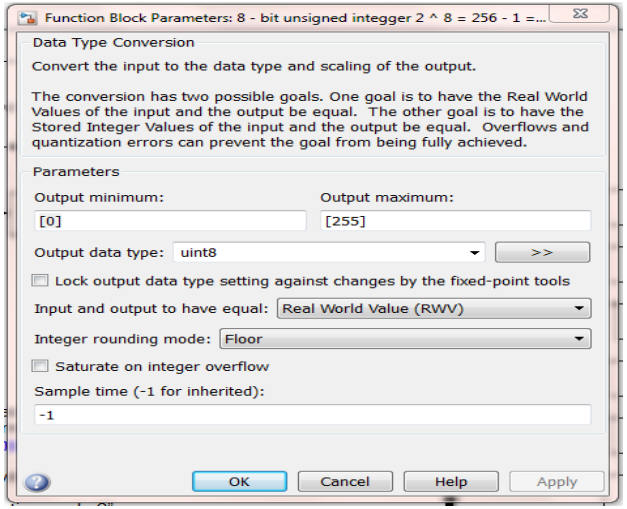


Fig. 3.22. Configurarea parametrilor pentru blocul de conversie fracționar la întreg

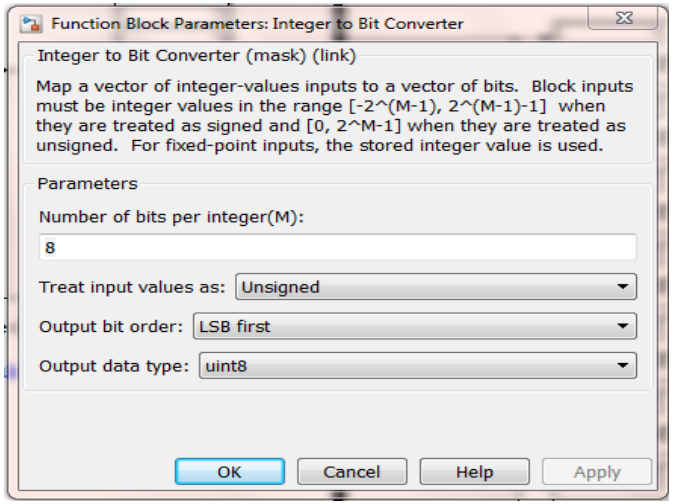


Fig. 3.23. Configurarea parametrilor pentru blocul de conversie întreg la binar

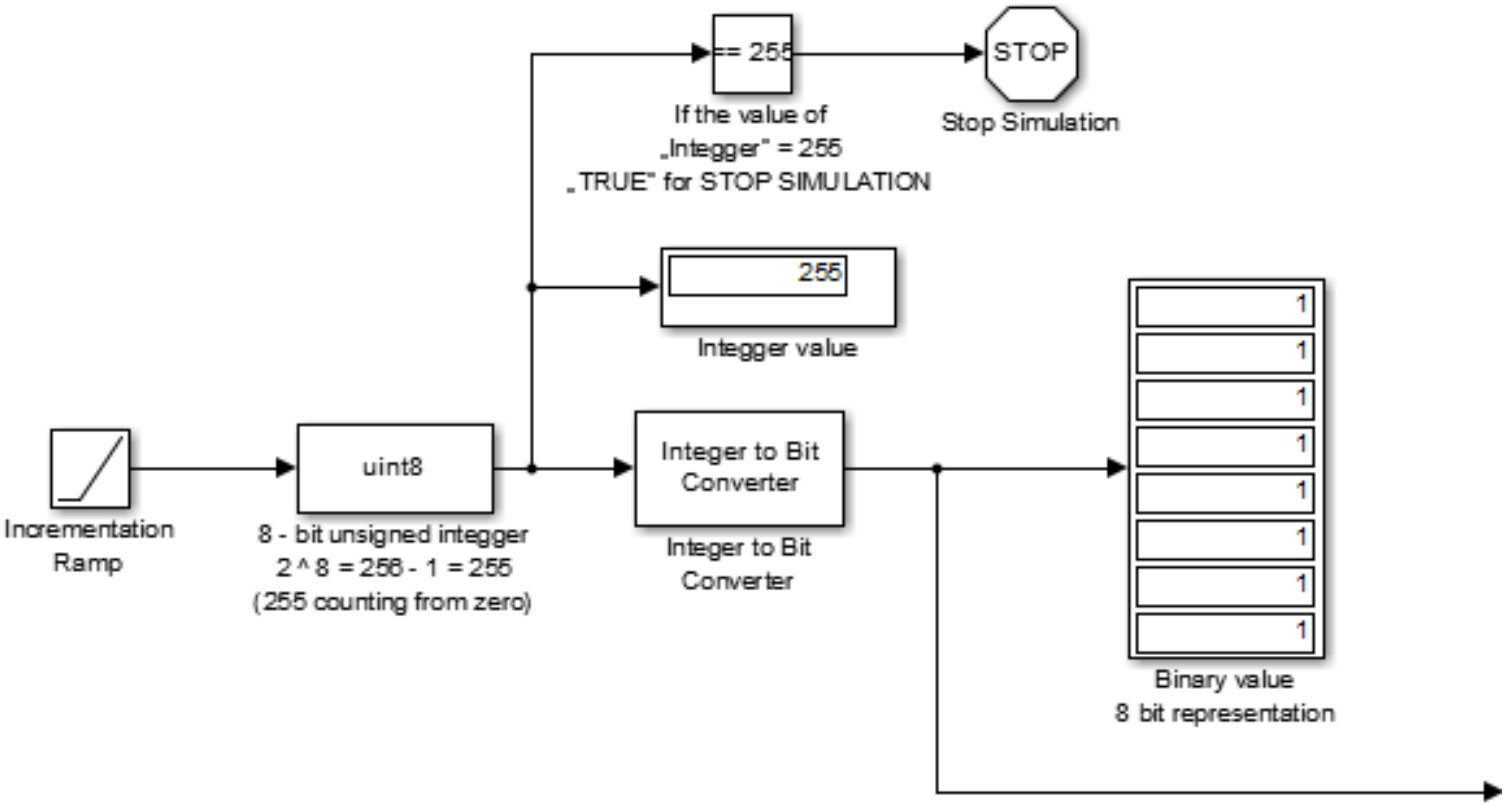


Fig. 3.24. Diagrama modelului matematic al aplicației realizat în Simulink

Rezultatul simulării pentru un exemplu aleator al stărilor de comutare cu cei 8 tacţi de numărare aferenţi se vizualizeaza în Fig.3.26.

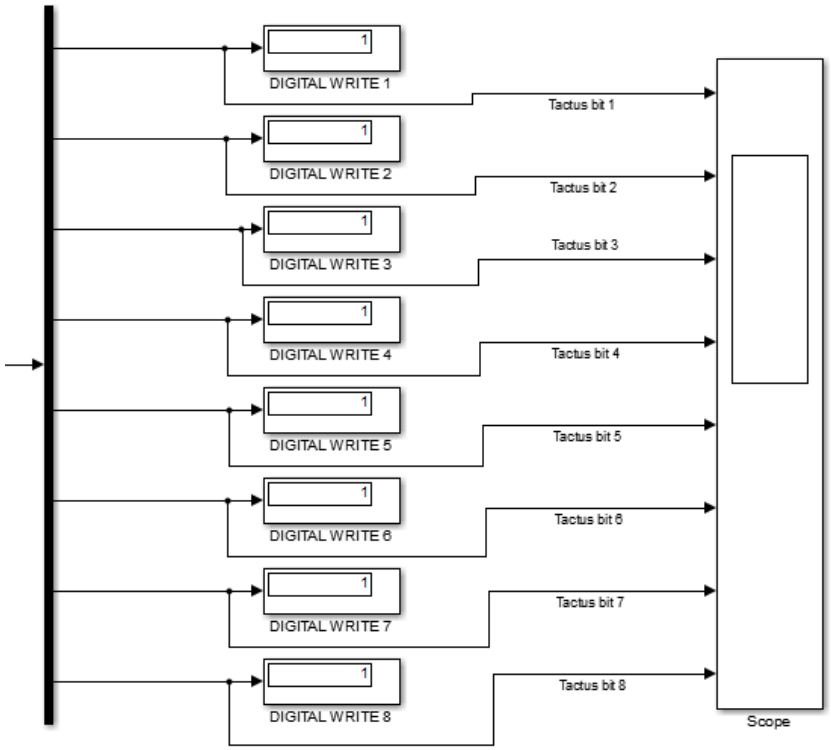


Fig. 3.25. Exemplu de de-multiplexare

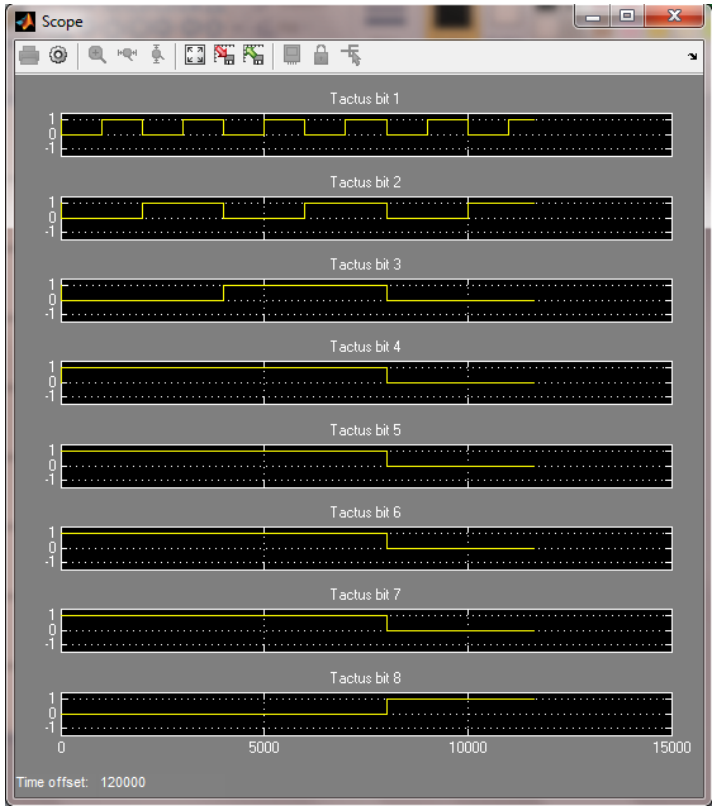
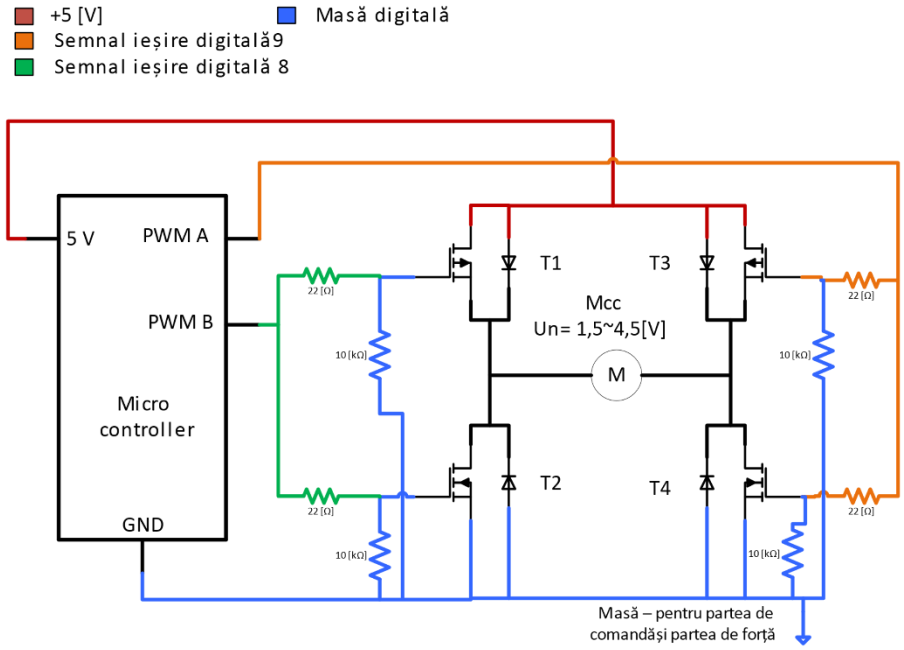


Fig. 3.26. Rezultate simulare - exemplu comutare tacți de numărare

3.5 Aplicaţie - Controlul digital al turației unui motor de curent continuu

În aplicaţiile cu microcontrolere care necesită realizarea unei acţionări mecanice, cele mai folosite componente sunt motoarele electrice, parte din clasa actuatoarelor. Deşi principiile de funcţionare ale acestora sunt general valabile, indiferent de variantele constructive, în cazul topologiilor folosite în circuite cu microcontrolere, cele trei tipuri de motoare des întâlnite (*motoare de curent continuu, motoare pas cu pas şi servomotoare*) pot fi considerate ca facând parte dintr-o aceaşi clasă, şi anume a *servomotoarel*or, datorită faptului că, în acest caz, conversia energiei electrice în energie mecanică se realizează la nivel de semnal şi nu nivelul de putere este cel definitoriu.

Controlul vitezei şi a direcţiei motorului de curent continuu se realizează prin controlul amplitudinii şi polarităţii tensiunii aplicate, dar nu se rezumă doar la aceşti doi parametri electrici, puterea pe care o poate genera un motor fiind determinată de tensiunea şi curenţii de alimentare. Cum alimentarea/comanda directă a motorului prin pinii dedicaţi GPIO ai platformei din familia Arduino (Intel Galileo) este limitată din punct de vedere al valorilor curenţilor, se folosesc circuite suplimentare: circuite cu tranzistoare/diode şi, eventual, a unei surse externe de tensiune, variatoare de tensiune continuă de patru cadrane (punți H, cu patru elemente de comutaţie: tranzistoare, MOSFET-uri), care, acţionate exclusiv pe una dintre diagonale, comandă alimentarea motorului, astfel încât să se realizeze mişcarea de rotaţie a acestuia într-una din cele două direcţii. Fig.3.28. prezintă circuitul electric de alimentare printr-o platformă de dezvoltare şi folosind o punte invertoare H; sunt prezentate, explicit secvenţele de comutaţie ale elementelor punţii pentru a comanda motorul.



T1	T2	T4	T3	Acţiune
0	1	0	1	rotire în sens invers acelor de ceasornic
1	0	1	0	rotire în sensul acelor de ceasornic
0	1	1	0	frânare
1	0	0	1	frânare

Fig. 3.27. Schema de control digital al unui motor de curent continuu, folosind punte H şi comandă digitală a elementelor de comutaţie

În aplicaţia de faţă, se foloşeste un motor de curent continuu cu sistem perie – colector şi magneţi permanenţi în excitație. Există şi alte variante constructive, mai putin folosite în aplicaţiile de interfațare cu microcontrolere.

Circuitul de alimentare poate fi realizat corespunzător, din componentele electronice mentionate sau poate fi folosită o variantă comercială, de punte sub denumirea de “driver”, care include diode sau senzori de curent încorporaţi. O configuraţie comercială capabilă să comande două motoare de curent continuu este prezentata în Fig. 3.28.

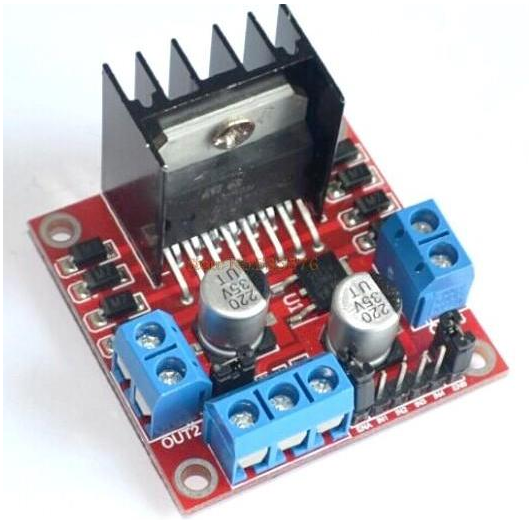


Fig. 3.28. Modul Arduino L298N - punte dubla H, varianta comercială, pentru controlul motoarelor

Câteva din caracteristicile de bază ale motoarelor care trebuiesc luate în considerare în cazul unei aplicaţii cu motor sunt:

- Tensiunea nominala la care funcţionează
- Curentul – care depinde de sarcina acţionată
- Viteza (rpm)
- Cuplul electromagnetic

- Rezistenţa înfăşurărilor

În cazul motorului de curent continuu, viteza fiind proporţională cu tensiunea, cea mai eficientă metodă de control al acesteia este folosirea modulaţiei în lăţime a pulsului (PWM).

În general, semnale analogice sau continue nu sunt furnizate în mod natural în sistemelele de calcul, doar sub forma semnalelor digitale și discrete, adică șiruri de valori constante, finite; drept urmare, pentru acest caz, se vorbeşte despre control „digital" sau „numeric". Există două metode fundamentale de comandă, reglaj și control pentru turația motorului de curent continuu, utilizate de către sistemele numerice de calcul:

- *comanda de pornire / oprire (ex. la depășirea unui anumit prag sau manual);*
- *variația discretă a turației prin semnale modulate în durată (PWM.);*

Pentru a preveni orice *problemă* legată de:

-modul de conexiune al masei sau terminalului de potențial zero;
-zgomote aleatorii atât de mod comun, cât și de mod diferențial;
-diferența de nivel de tensiune (eng. 'level shifting');
-adaptarea la impedanță a circuitului;
se recomanda introducerea elementelor de interfatare sau separare galvanica între partea de forță și partea de comandă (Fig. 3.29). Izolarea sau separarea galvanică s-a realizat prin intermediul optocuploarelor A3120 - „opto-gate driver" (un circuit integrat hibrid, deoarece, pe lângă optocuplor conține și un etaj formator al impulsului de comandă pentru grila tranzistoarelor MOSFET pe care le comandă în punte);

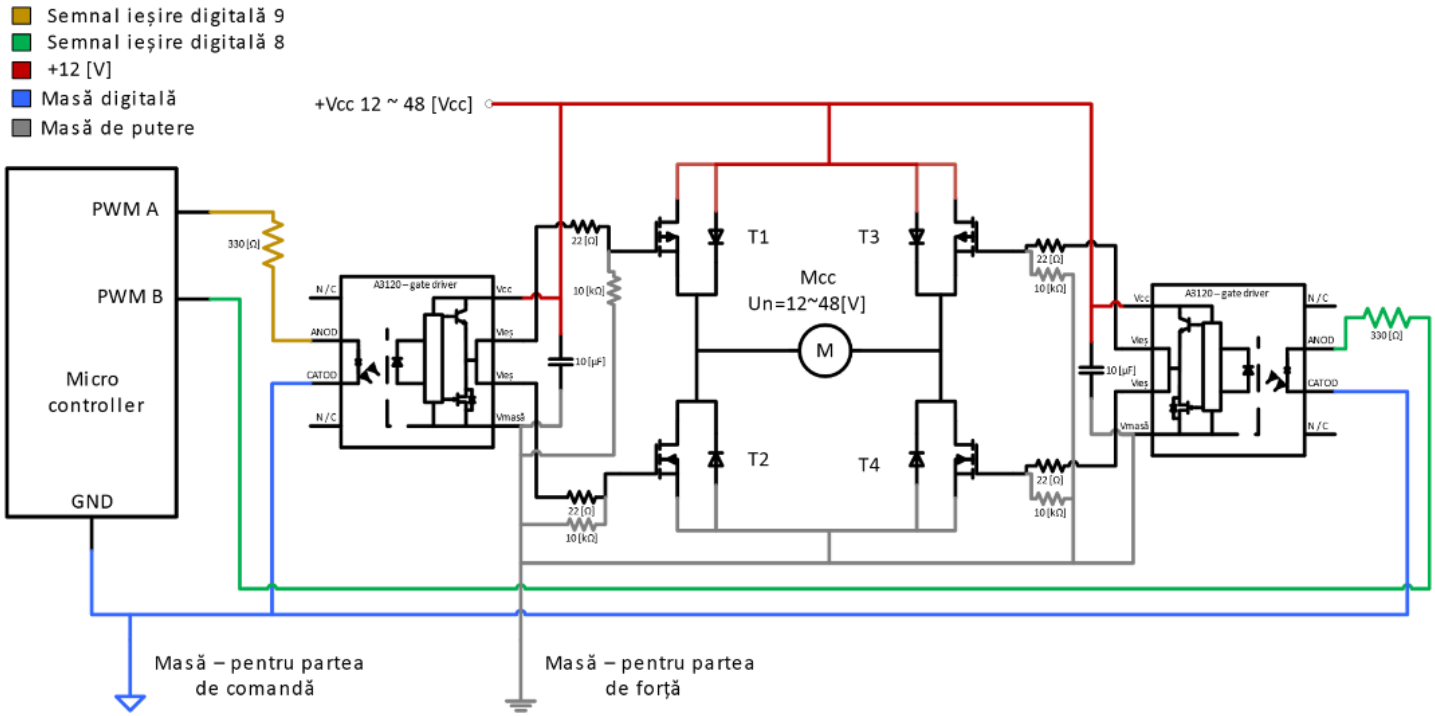


Fig. 3.29. Schema de control digital a unui motor de curent continuu, folosind punte H si optocuploare

Pe partea de forță (ex. motor, tranzistoare, sursă auxiliară de tensiune mult mai mare decât alimentarea platformei) se vehiculează curenți mult mai mari și tensiuni/diferențe de potențial însemnate. Acest fenomen cauzează, de obicei, cele mai importante probleme în instalațiile de acționare, comandă și control sau automatizare; astfel, dacă nu se ia în considerare folosirea strategiei de separare galvanică și se folosește doar conexiunea directă a ieșirii digitale de la platforma Intel Galileo la circuitul de forță, datorită diferenței mari de potențial (5 [V] la 12 – 48 [V]), s-ar vehicula curenți de circulație mari între circuite. Suplimentar, în termeni de compatibilitate electromagnetică, s-ar realiza calea perfectă de cuplaj

direct galvanic pentru permiterea zgomotelor de comutație din circuitul de forță, dar și din circuitul motorului, sistemul perie colector. De asemenea, datorită faptului că motorul de curentu continuu funcționează pe principiul comutației succesive a înfășurărilor, apar perturbații de frecvențe și tensiuni mari. Aceste zgomote se induc, în cazul aplicațiilor industriale sau în timp real, în circuitul de masă sau traseele de alimentare. De obicei, traseele de comandă neprotejate, legate la o masă comună cu partea de forță, sunt ținta acestor zgomote electromagnetice. Din aceste motive, se preferă varianta de circuit cu optocuplare. Această soluție este suficientă și acoperitoare atât pentru problemele de masă (introduce un punct de conectare la masa pentru partea de comandă, de obicei masă digitală eng. Digital Ground – DGND și un punct de conectare la masa de putere sau forță eng. Power Ground – PGND).

În general, principiul de funcţionare a punţii optocuploare A120 se bazează pe structura din Fig.3.30. Prin intermediul unui foto-emițător (de obicei, LED cu aplicare în spectru invizibil infra-roșu) și al unui foto-receptor (de obicei foto-tranzistor cu aplicare în spectru invizibil infra-roșu), încapsulați într-un substrat opac la exterior, dar transparent în interior se realizează transferul de informație sau comandă prin câmp electro-magnetic luminos, de la un circuit la altul.

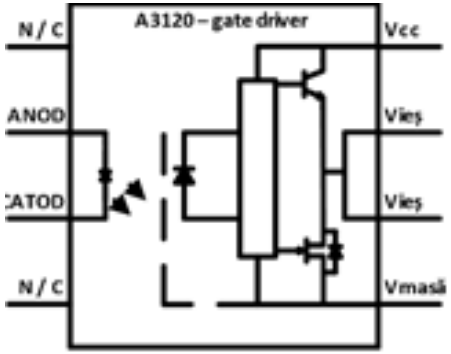


Fig. 3.30. Structura unui optocuplor A3120

De exemplu, platforma Intel Galileo, beneficiază de o sursă de alimentare independentă față de sursa de la portul USB. Acest lucru poate avantaja compatibilitatea circuitelor de mică putere, deoarece extinde gama de puteri vehiculate pe ieşirile digitale față de celelalte modele aflate pe piață. **Platforma nu beneficiaza de izolarea galvanica de a portului USB, ci doar de alimentare separată**. Modelele mai noi de

microcontrolere care înglobează sau conțin controlerul de interfațare USB – FTDI (ex. TI-F2877s Delfino) dispun de interfaţare sau izolare galvanică.

Pinii dedicaţi ai platformelor din familia Arduino pot furniza curenţi până la 40 mA, valoare suficientă pentru a activa LED-uri, de exemplu; dar nu pot furniza curenţi pentru funcţionarea unui motor (50-100 mA). În acest caz, folosirea unui tranzistor în circuitul de alimentare permite furnizarea unui curent mai mare, permiţând deschiderea sau închiderea circuitului motorului.

Pentru realizarea circuitului este nevoie de urmatoarele **componente:**

- Platforma de dezvoltare
- Cablu USB
- Cablu de alimentare al platformei (în cazul Intel Galileo)
- Breadboard
- 1 motor de curent continuu, max.5 V
- 1 rezistenta de 10k Ω
- 1 tranzistor NPN
- 1 dioda 1N4148
- Fire

Aplicaţia de faţă foloseşte schema simplificată din Fig.3.31, în cazul puterilor relativ mici vehiculate şi dacă nu mai există elemente sau dispozitive comandate/citite prin ieşirile digitale capabile PWM, circuitul optocuplor putând fi considerat opţional (se ia în considerare şi faptul că alimentarea circuitului optocuplor are nevoie de alimentare separată, 15V). În circuitul prezentat se foloseşte un tranzistor de tip 2N2222, care permite transmiterea în circuit a unui curent de până la 200 mA; comanda digitală, în ArduinoIDE se realizează cu functia digitalWrite() pentru cazul ON/OFF, respectiv cu analogWrite(), pentru comanda PWM, care permite variaţia lăţimii pulsului între 0% şi 100%; la valori prea mici ale vitezei, reducându-se proporţional cuplul electromagnetic al motorului, pot apărea probleme la pornire, caz în care trebuie mărită puţin viteza.

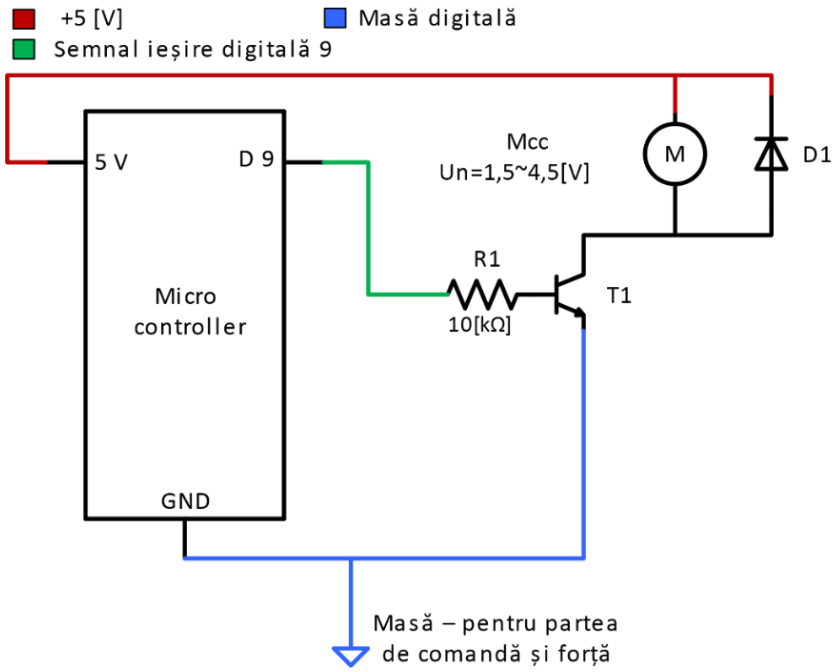


Fig.3.31. Circuitul aplicaţiei de control al unui motor de curent continuu, folosind platforma Intel Galileo

Conectarea tranzistorului în circuitul aplicaţiei (Fig.3.31) se face după identificarea, din documentaţia specifică a tranzistorului, a *colectorului, bazei şi emitorului*; succesiv, colectorul se conectează la firul negru al motorului, emitorul la pinul de masa (GND), iar baza, printr-o rezistenţă de 330 Ω la un pin digital al platformei capabil să realizeze PWM (în exemplu, pinul 10). Cealaltă conexiune a motorului de curent continuu, cablul rosu, se conectează la pinul de 5V.

Rolul diodei în circuit este de a suprima vârfurile de tensiune apărute din cauza variaţiei câmpului magnetic la porniri şi opriri bruşte. Astfel, catodul diodei se introduce în circuit, conectându-se la 5V, iar anodul la cablul negru al motorului.

În cazul folosirii platformelor, care necesită introducerea separării galvanice, se va folosi circuitul corespunzator, din Fig.3.32.

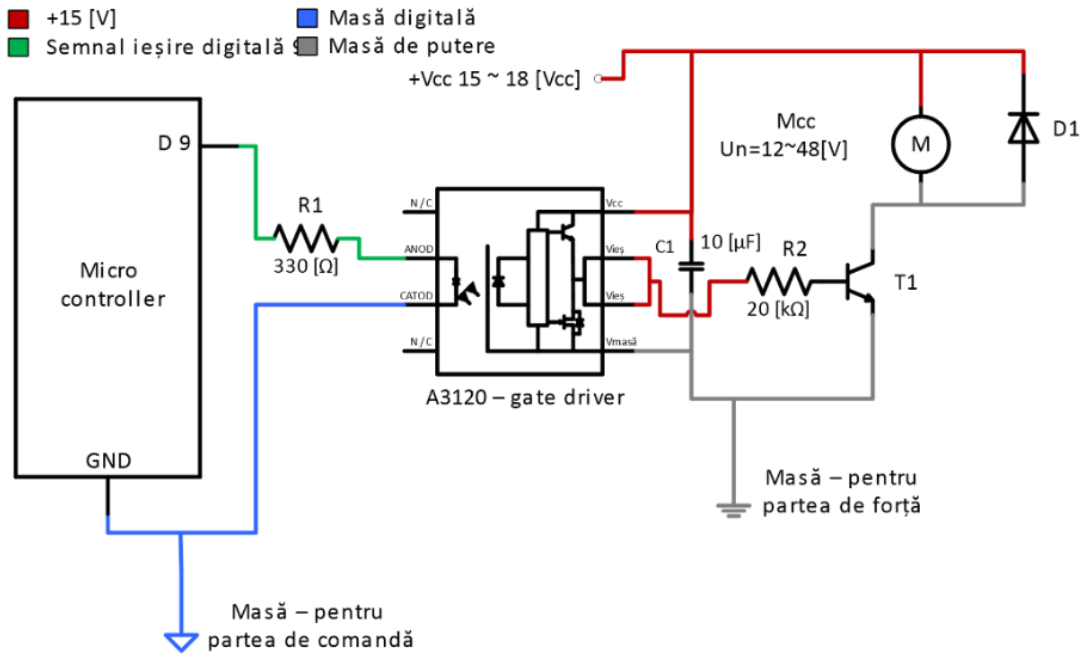


Fig.3.32. Circuitul aplicatiei de control a unui motor de curent continuu, cu separare galvanica

Implementarea algoritmului de comanda prin mediul Arduino IDE:

```
// A. Pornirea/oprirea temporizata a motorului, la intervale de 2 secunde;
const int motor = 10;
void setup()
{
  pinMode(motor, OUTPUT);
```

```
    //setarea motorului ca iesire
  Serial.begin(9600);
    //stabilirea comunicatiei seriale
}
void loop()
{
  digitalWrite(motor, HIGH);
    // pornirea motorului la viteza maxima
  delay(2000);
    // pastreaza starea timp de 2 secunde
  digitalWrite(motor, LOW);
    // oprirea motorului
  delay(2000);
    // pastreaza starea timp de 2 secunde
}
```

// B. Accelerarea motorului pana la viteza maxima si decelerarea pana la viteza 0;

```
const int motor = 10;
void setup()
{
  pinMode(motor, OUTPUT);
    //declararea motorului ca iesire
  Serial.begin(9600);
    //stabilirea comunicatiei seriale
}
void loop()
{
  int viteza;
  int del = 20;
```

```
    // intarziere 20 milisecunde intre pasii de accelerare
    //accelerarea motorului, prin trecerea de la 0 la 255 – //valori digitale corespunzatoare vitezei
    motorului
  for(viteza = 0; viteza <= 255; viteza++)
  {    analogWrite(motor,viteza);
     // se impune viteza
    delay(del);
      // intarzierea intre pasi
  }
       // decelerarea motorului
  for(viteza= 255; viteza >= 0; viteza--)
  {
    analogWrite(motor,viteza);
    delay(del);                }
}
```

// C. Introducerea vitezei de functionare a motorului prin monitorul serial al Arduino IDE

```
// introducerea unei valori valide 0 – 255 (fiind vorba despre comanda pe pini digitali ai platformei Intel
//Galileo)  pentru viteza motorului de curent continuu;
const int motor = 10;
void setup()
{
  pinMode(motor, OUTPUT);
//declararea motorului ca iesire
  Serial.begin(9600);
//stabilirea comunicatiei seriale
}
void loop()
{
```

```
  int viteza;
  Serial.println("introduceti o viteza intre (0-255)");
  Serial.println();  // linie goala
  while(true)  // conditie permanent adevarata, pentru rularea in bucla infinita
  {
    // verificare date disponibile
    while (Serial.available() > 0)
    {
      // parseInt() pentru extragerea datelor de tip ,integer'
      viteza = Serial.parseInt();
 // restrangem gama de lucru la  0-255,pentru ca functia analogWrite() functioneaza doar in aceasta gama
viteza = constrain(viteza, 0, 255);
      Serial.print("Setarea vitezei la valoarea ");
      Serial.println(viteza);
      analogWrite(motor, viteza);// transmitem/scriem viteza motorului
    }
  }
}
```

Abordarea aplicației folosind pachetul Matlab – Simulink

Comanda de pornire/oprire se realizează prin prezența/absența tensiunii la bornele de alimentare ale mașinii de curent continuu cu perii. Acest fenomen fizic poate fi interfațat atât cu ajutoul unui releu cât și cu ajutorul unui tranzistor de putere cu efect de câmp (MOSFET - avantajul major al tranzistoarelor cu efect de câmp constă în posibilitatea de a fi comandat cu ajutorul unei **tensiuni de comandă**, nu cu ajutorul unui curent, ca și în cazul tranzistorului bipolar). De asemenea, prin însăși construcția sa, tranzistoarele MOSFET, au grila izolată față de circuitul de forță/ putere, deoarece comanda se realizează prin câmp electric – se comportă ca un condensator, deci nu există legătură fizică/electrică între partea de forță/putere și partea de comandă). Indiferent de metoda aleasă, este necesară generarea unui semnal de comandă, cu ajutorul căruia se va acționa **circuitul de comandă al elementului comutator**. Acest semnal se poate obține de la sistemul de calcul sub formă de tren de impulsuri sau impuls

constant. Furnizarea fizică a acestui semnal este asigurată de ieşirea digitală a sistemului de calcul, deci, un astfel de semnal furnizează starea de pornire/oprire a mașinii.

În Matlab Simulink, acest lucru se poate realiza utilizând, fie blocurile dedicate ieșirilor digitale, la care se vor furniza valori logice (boolean) „logic 1”/„logic 0” sau se pot utiliza blocurile de furnizare a semnalului modulat în durată, la intrarea căruia se impun valorile 0 - oprit și 255 - pornit.

Variaţia discretă a turaţiei prin semnale modulate în durata (PWM) se realizează prin *furnizarea unui tren de impulsuri de lățime variabilă.* Printr-un astfel de procedeu, se modifică valoarea medie a tensiunii; odată cu modificarea factorului de umplere (sau lățimii impulsului), se va modifica și valoarea medie a tensiunii, dar și curentul absorbit va avea o formă triunghiulară. La frecvențe mari de comutație, forma triunghiulară devine insesizabilă. Asadar, controlul factorului de umplere afectează turația mașinii.

Tensiunea medie se poate exprima ca și aria de sub grafic. Graficul unei tensiuni continue descrie un dreptunghi; dacă acesta a fost împărțit în „n” bucăți discrete, atunci valoarea medie a tensiunii se poate exprima ca și în ec. (1):

$$U_{MED} = \sum_{i=1}^{n} A_i = \sum_{i=1}^{n} \mathrm{u(t)}_i * t_i = \frac{1}{T} \sum_{i=1}^{n} \mathrm{u(t)}_i \qquad (1)$$

Unde: „A_i” – aria fiecărei porțiuni discrete de sub grafic;
„$u(t)_i$” – valoarea instantanee a tensiunii la momentul „t”;
„t_i” – timpul la care s-a măsurat valoarea instantanee a tensiunii;
„T” – perioada semnalului;

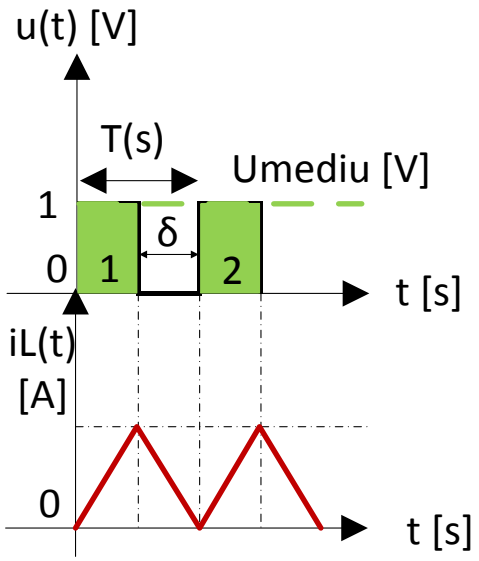


Fig. 3.33. Formarea curentului printr-o sarcină inductivă și variația valorii medii în funcție de factorul de umplere

Valoarea medie este, astfel, o mărime strict dependentă de timp – Fig.3.33.- (conform ec. (1)). Prin modificarea „duratei de acționare” „δ” sau a factorului de umplere, se va putea modifica valoarea medie a tensiunii de alimentare la bornele mașinii, fără diminuarea amplitudinii semnalului; acest lucru este practic imposibil de realizat într-un sistem numeric sau sistem de calcul digital. Prin această metodă, se poate controla în mod discret, turația mașinii.

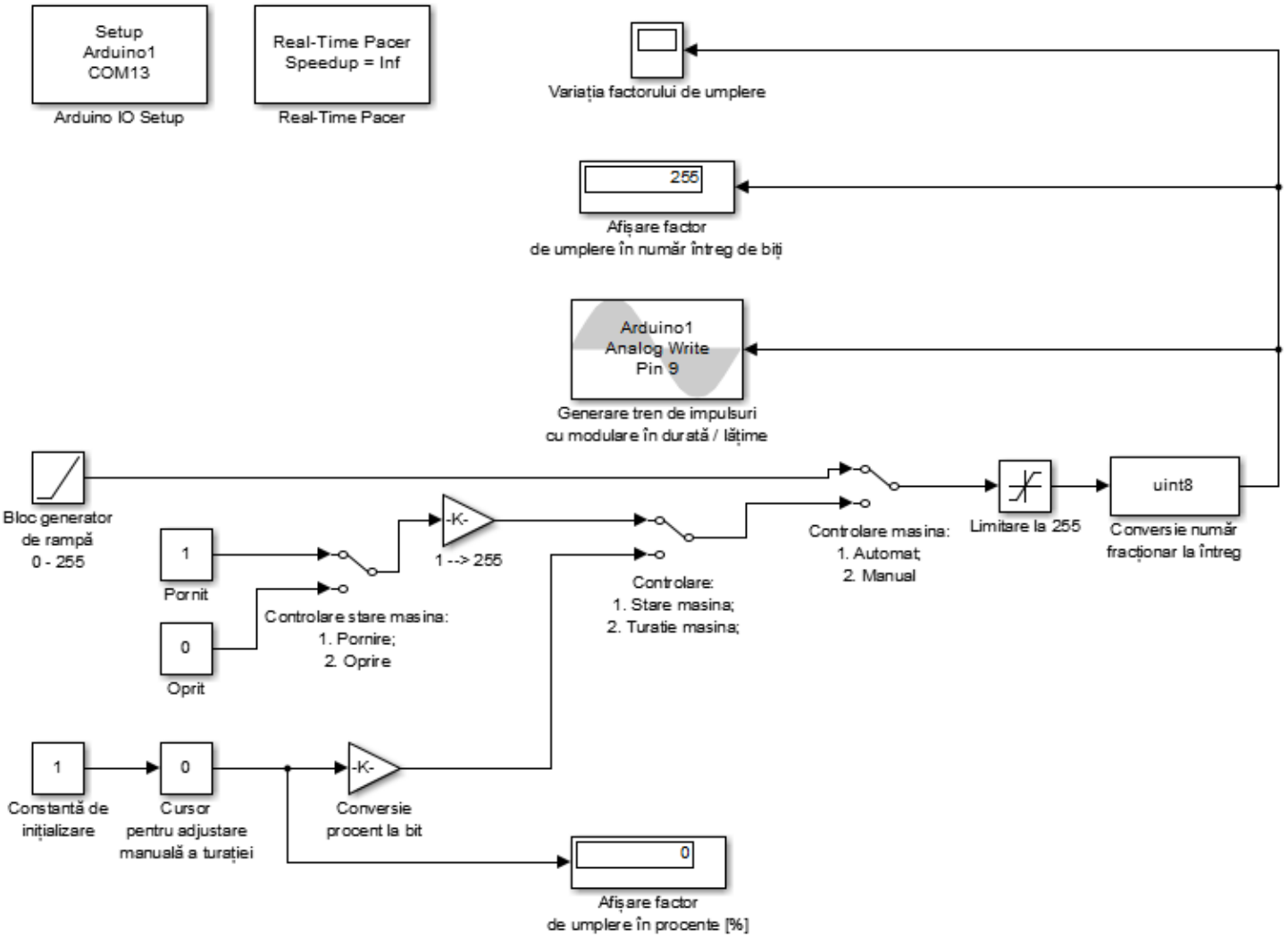


Fig.3.34. Aplicație de control motor de curent continuu

În modelul Matlab/Simulink specificat pentru respectiva aplicație, există posbilitatea reglării factorului de umplere manual sau automat. Pe cale automată o funcție de tip rampă (generator de rampă) va modifica unitar factorul de umplere, până la valoarea 255, crescând turația mașinii. Exista și posibilitatea de a comanda manual în stare pornit/oprit sau, cu ajutorul unui cursor numeric, se poate impune valoarea dorită a turației. Posibilitatea folosirii unor comutatoare oferă şi posibilitatea alegerii unui *mod de lucru (manual sau automat*).

Diagrama modelului matematic implementat in Matlab/Simulink şi rezultatele simulării sunt prezentate în Fig.3.34-3.35.

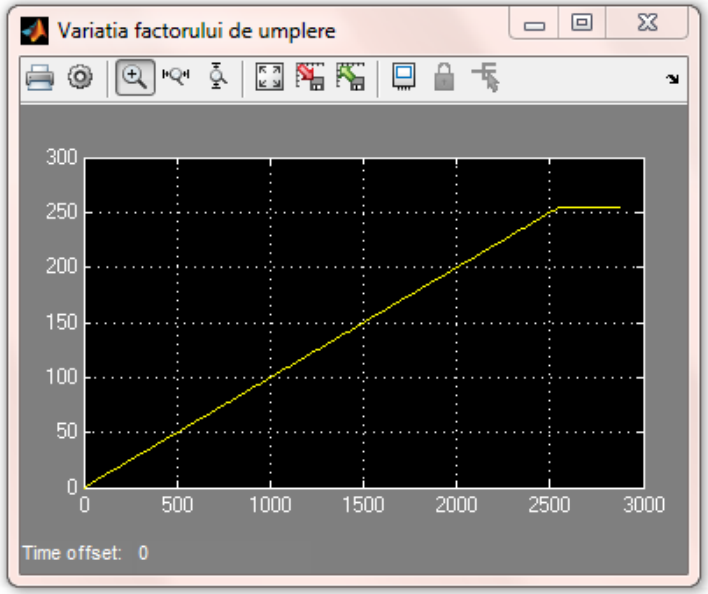


Fig.3.35. Rezultatele simulării - variația turației după o funcție rampă

3.6 Aplicaţie de incrementare-decrementare digitală

O aplicaţie utilă în controlul digital, care foloseşte doar un circuit cu două butoane (pornind de la circuitul din aplicatia 3.2), o reprezintă incrementarea/decrementarea digitală, cu urmărirea valorilor prin monitorul serial specific lui Arduino IDE. Prin adaptarea codului prezentat, elementele de tip comutator cu logica de comandă prezentată pot deveni **butoane** dintr-un meniu software sau **elemente de comandă într-un sistem automatizat.** Logica de comandă implementată este redată prin secvenţa de cod explicită.

Implementarea algoritmului de comanda prin mediul Arduino IDE:

```
const int buton_crescator = 3;
const int buton_descrescator = 4;
int contor_buton = 0;
int stare_buton_crescator = 0;
int ultima_stare_buton_crescator = 0;
int stare_buton_descrescator = 0;
int ultima_stare_buton_descrescator = 0;
void setup() {
  pinMode(buton_crescator, INPUT);
  pinMode(buton_descrescator, INPUT);
  Serial.begin(9600);
}
void loop() {
```

```
 stare_buton_crescator = digitalRead(buton_crescator);
stare_buton_descrescator = digitalRead(buton_descrescator);
  if(stare_buton_crescator != ultima_stare_buton_crescator){
    if(stare_buton_crescator == HIGH){
      contor_buton++;
      if (contor_buton >= 8){
        contor_buton = 8;
      }
      Serial.println("Valoarea impusa digital este: ");
      Serial.println(contor_buton);
    }
    delay(50);
  }
ultima_stare_buton_crescator = stare_buton_crescator;
if(stare_buton_descrescator != ultima_stare_buton_descrescator){
    if(stare_buton_descrescator == HIGH){
      contor_buton--;
      if (contor_buton < 0){  contor_buton = 0; }
      Serial.println("Valoarea impusa digital este: ");
```

```
      Serial.println(contor_buton);
    }
    delay(50);
  }
ultima_stare_buton_descrescator = stare_buton_descrescator;
}
```

În contextul aplicaţiilor prezentate în acest capitol, se sugerează combinarea elementelor descrise, în aplicaţii mai complexe, utilizând şi componente din capitolele referitoare la comunicare şi la comandă analogică. Teme sugerate sunt legate de folosirea butoanelor şi a LED-urilor în aplicaţii precum:

- **Contoare apăsare buton**
- **Coloană de LED-uri comandate prin buton**
- **Autoreţinerea stării unui buton**
- **Aplicaţie de semnalizare (auto)**
- **Comanda unui motor cu meniu realizat din butoane**

4. APLICAŢII DE INTERFAŢARE FOLOSIND INTRĂRI ANALOGICE

Aplicaţiile de control în timp real au la bază sesizarea fenomenelor fizice şi procesarea lor, prin intermediul dispozitivelor de procesare de tip sisteme încorporate/microcontrolere, care conţin, pe lângă elementele specifice de stocare şi procesare, şi pini dedicaţi de intrare/ieşire pentru interacţiunea cu circuitele externe. Aceste intrări/ieşiri specifice se folosesc pentru citirea datelor de la senzori, pentru comunicarea cu alte dispozitive şi pentru controlul altor sisteme din circuit. Prin intermediul acestor dispozitive care au la bază un microcontroler/microprocesor, s-au înlocuit tehnicile de afişare şi de măsurări electronice în aplicaţii simplificate de achiziţii de date. Procesele şi fenomenele legate de natură variază în timp, sunt valori analogice, iar sistemele cu microcontrolere „înteleg" şi procesează doar valori digitale (biţi); astfel, valorile preluate, prin senzori adecvaţi, referitori la achiziţii de temperatură, presiune, poziţie etc. trebuie traduse corespunzător pentru a fi prelucrate în timp real într-un sistem de control ce deserveşte o aplicaţie. Diagrama din Fig.4.1. prezintă funcţionarea unui sistem de conversie a informaţiei preluate din lumea fizică în limbajul dedicat al procesoarelor.

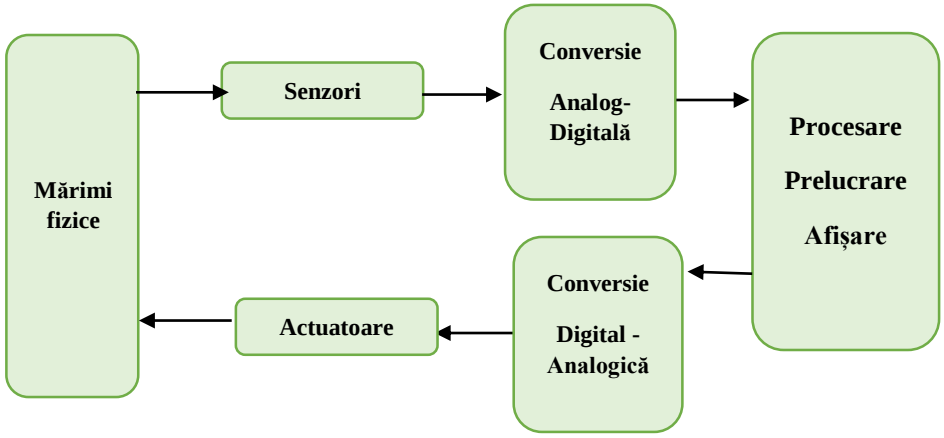


Fig.4.1. Diagrama procesării datelor într-un sistem cu microcontroler

O scurtă explicaţie comparativă cu **intrările/ieşirile digitale**, prezentate în capitolul anterior, trebuie menţionată. Intrările/ieşirile digitale pot primi două valori, reprezentate de două stări electrice: nivelul de tensiune HIGH şi nivelul de tensiune LOW, manipularea lor software făcându-

se prin funcţiile **`digitalWrite()`** - pentru setarea unui pin pe cele două stari (de exemplu, pentru comutarea ON/OFF a unui dispozitiv), respectiv **`digitalRead()`** pentru obţinerea stării unui anumit pin.

În comparaţie, **intrările analogice** reprezintă semnale electrice într-o anumită gamă de tensiune; intrările analogice sunt, în general, folosite ca intrări pentru diferiţi senzori, care furnizează o mărime electrică variabila în timp şi care reprezintă amplitudinea variabilei sesizată de senzor. Această mărime trebuie să fie interpretată de elementul de procesare, deci trebuie convertită într-o valoare discretă.

Preluarea mărimilor fizice de natură neelectrică se realizeaza în circuite prin folosirea senzorilor sau a traductoarelor, care fac conversia mărimilor preluate în mărimi fizice de natură electrică (curent, tensiune). O *clasificare simplificată* a acestor dispozitive electronice conţine:

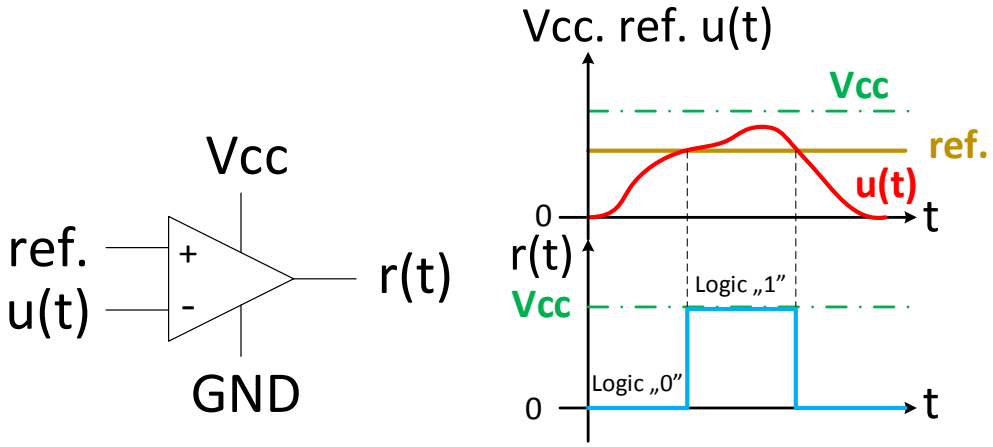


Fig.4.2. Funcţionarea comparatorului electronic

- Traductoare active (fotodiode, tahogeneratoare, termocuple) – nu necesită sursă auxiliară de alimentare, fiind generatoare de curent sau tensiune; se folosesc cu circuite formatoare de semnal pentru a reglementa parametrii semnalului de ieşire în limitele admisibile de prelucrabilitate;
- Traductoare pasive (fotorezistenţe, termistori, mărci tensometrice etc.) – necesită sursă auxiliară de alimentare, pentru că mărimea fizică preluată este transformată într-un parametru fizic de circuit; se folosesc circuite de adaptare pentru măsurarea efectului produs de acest tip de traductoare: divizor de tensiune sau curent, punte de măsură, transformatoare de măsură).

În ambele situații, fie că se dorește măsurarea unui curent, fie că se dorește măsurarea unei tensiuni, semnalul rezultant este tot o tensiune proporțională cu semnalul fizic neelectric, astfel ca înțelegerea funcționării divizorului de tensiune și a convertorului analog digital reprezinta elemente importante în aplicaţiile de interfaţare pe pinii analogici ai oricărei platforme de dezvoltare

Procesul de conversie a semnalului din analogic în digital se realizează prin intermediul unui **Convertor Analog-Digital (ADC Analog-Digital Converter)**, realizat din comparatoare electronice (Fig.4.2.), care furnizează reprezentarea unei valori analogice de intrare pe un număr de biţi corespunzător; numărul de comparatoare necesar este egal cu $2^{\text{numar biti}}$, un exemplu sugestiv fiind prezentat în exemplul din Fig.4.3.

În mod normal, numărul de comparatoare definește rezoluția convertorului, referința pentru fiecare comparator fiind dată de un divizor de tensiune rezistiv multiplu, alcătuit din rezistențe cu valori proporționale astfel încât tensi unea să fie împărțită în paşi egali (ex. 10 mV/pas, 100 mV/pas, 1 V/pas).

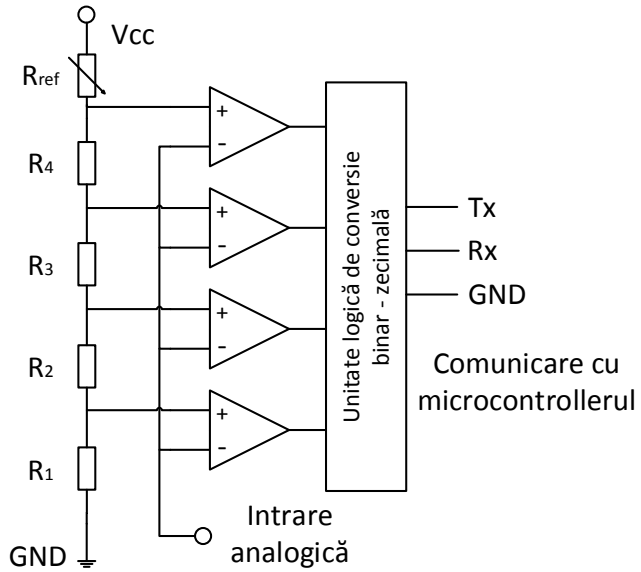


Fig.4.3. Convertor Analog Digital CAD cu patru comparatoare

Rezultatul conversiei analog numerice va fi reprezentat de un semnal discretizat numeric, care să reprezinte semnalul analogic de intrare.

Caracteristicile esenţiale ale unui convertor ADC sunt gama de intrare şi **rezoluţia de afişare**. Astfel, un convertor cu gama 0-5V şi rezoluţia 10 biţi, va afișa valoarea „00000000" pentru 0V, respectiv „11111111" pentru 5V, unitatea de conversie corespunzătoare rezoluţiei de 10 biţi fiind 5V/1023 (2^{10}-1) ~ 0,00488V, care reprezintă pasul de incrementare pentru tensiunile măsurate în gama data. Cu alte cuvinte, un modul ADC cu rezoluţia 10 biţi poate realiza 1024 (1023 cu semn) de măsurători diferite în gama 0-5V.

Acest demers matematic prin care semnalele analogice sunt traduse, prin intermediul convertoarelor analog digitale în valori binare care pot fi procesate de către unitatea de procesare, trebuie înteles şi adaptat paltformei folosite în aplicaţiile utilizate, tocmai pentru a facilita integrarea corectă a senzorilor în circuit. Diferiţi senzori de temperatură, de nivel, de presiune, de poziţie, de viteză etc. produc un semnal electric proporţional cu mărimiea fizică măsurată, corespunzătoare rolului respectivului senzor. Valoarea tensiunii sau curentului furnizate reprezintă valoarea brută, care necesită transformările aferente scalării sau calibrării senzorului în cauză; în cazul folosirii în circuite cu microcontrolere, acest lucru se realizeaza prin metodologia explicată anterior, urmată de implementarea în algoritmul matematic, a formulei de calibrare sau a metodei de scalare corespunzătoare.

Funcțiile de bază caracteristice intrărilor/ieșirilor analogice folosite în Arduino IDE sunt citite cu **analogRead(pin)** – convertește o valoare de tensiune din gama 0-5V la o valoare digitală din gama (0-$2^{\text{biți\_rezolutie-1}}$). În mod reciproc, se poate furniza cu ajutorul funcției **analogWrite (pin, valoare_intre_0_255)** – un semnal modulat în durata impulsului (eng. P.W.M. Pulse Width Modulation) în cazul pinilor digitali capabili să furnizeze aces tip de semnal, la o frecvență de lucru pre-stabilită.

Aplicaţiile prezentate în acest capitol, atât în abordarea generala, prin mediul Arduino IDE, cât şi prin mediul Matlab/Simulink, îşi propun exemplificarea folosirii senzorilor şi dispozivitelor analogice în circuite simple, generale, de la care se pot dezvolta aplicaţii mai complexe.

4.1 Aplicaţie de interfaţare cu intrări analogice – divizorul de tensiune şi potenţiometrul

Pentru înțelegerea procesului de citire și prelucrare ulterioară a unui semnal analogic (tensiune), se prezintă un circuit simplificat de intrare care furnizează tensiune variabilă pe pinii analogici de intrare ai platformei din familia Arduino/Intel Galileo, iar în acest sens, cea mai simplă

metodă este construirea un divizor de tensiune, folosind un potențiometru; la ieșire, se propune folosirea unui circuit cu LED, care poate fi controlat prin tensiunea de pe pinul de intrare analog.

Atât divizorul de tensiune, cât și potențiometrul (echivalente din punct de vedere electric) sunt considerate elemente pasive de circuit. Deci, este nevoie de o sursă separată de alimentare pentru a realiza transformarea unui nivel de tensiune continuă, la alt nivel de tensiune continuă, strict în sensul de scădere a acestora. Din Fig. 4.4. se poate remarca componenta circuitelor asociate unui divizor de tensiune (a), respectiv a unui potențiometru (b).

În cazul divizorului de tensiune, V_{in} se aplică pe rezistențele R_1, R_2, iar tensiunea de ieșire V_{out} reprezintă căderea de tensiune pe rezistența R_2, rezultând o diminuare a tensiunii de ieșire. Potențiometrul poate fi considerat un divizor de tensiune, care conține un cursor.

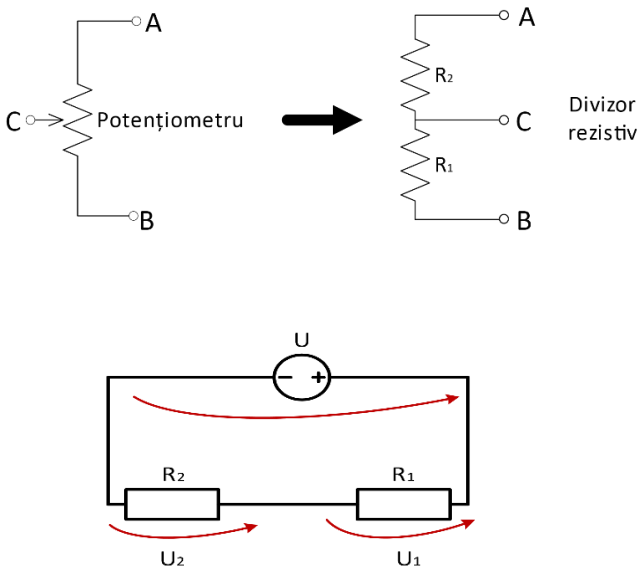


Fig.4.4. Circuitul electric al unui (a) potentiometru și al unui divizor de tensiune (b)

Nivelurile de tensiune la ieșire vor fi calculate cu relaţiile matematice:

$$U_1 = R_1 \cdot \frac{U}{R_1 + R_2} \qquad U_2 = R_2 \cdot \frac{U}{R_1 + R_2} \tag{1}$$

Prima aplicație propune preluarea „brută” a valorii citite de pe pinul analogic prin platforma din familia Arduino, cu ajutorul unui potentiometru conectat în circuit. Citirea simpla a semnalului de la o intrare analogică reprezintă, de fapt, **aplicaţia de funcţionare a Convertorului Analog Digital** al platformei.

Conectarea hardware a potenţiometrului în circuitul platformei de dezvoltare

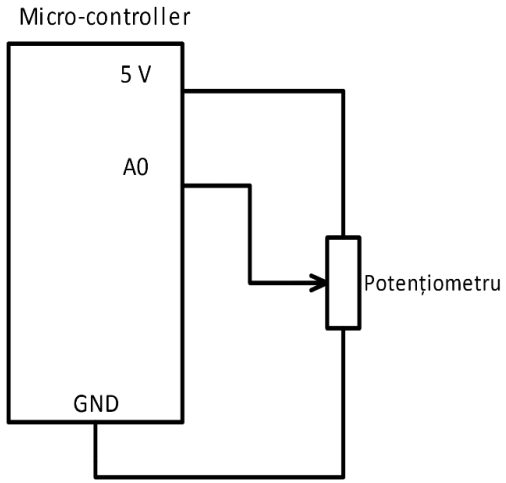


Fig. 4.5. Schema electrică a montajului simplu cu potentiometru

Pentru realizarea circuitului, prezentat în Fig. 4.5. este nevoie de componentele listate mai jos. Cursorul potentiometrului este conectat la pinul analogic prin care se face citirea/conversia A/D, iar celelalte două capete ale potenţiometrului închid circuitul prin pinul de alimentare 5V şi pinul de masă GND.

- Platforma de dezvoltare
- Cablu USB
- Cablu de alimentare al platformei (în cazul Intel Galileo)
- Breadboard
- Potenţiometru 10k
- Fire

Implementarea algoritmului de citire a unui pin analogic, prin mediul Arduino IDE:

(a) Funcţionarea convertorului analog digital

```
const int pin_analogic=A0;
int valoare_CAD=0;
void setup()
{
  Serial.begin(9600);
// deschide comunicarea pe portul serial la 9600 bps:

}
void loop()
{
valoare_CAD=analogRead(pin_analogic);
```

```
// salveaza in constanta declarata, valoarea citita de pe pinul A0;
Serial.println(„Valoarea conversiei D/A”);
// afisarea pe monitorul serial a valorii citite, insotite de textul //explicativ
Serial.println(valoare_CAD);
}
```

(b) Transformarea rezultatului conversiei in valori de tensiune

```
const int pin_analogic=A0;
int valoare_CAD=0;
float tensiune=0;
 //declararea variabilei tensiune, de tip float
void setup()
{
  Serial.begin(9600);
// deschide comunicarea pe portul serial la 9600 bps:
}
void loop()
{
valoare_CAD=analogRead(pin_analogic);
tensiune=((5.0/1023.0)*valoare_CAD);
//tensiunea maxima 5V corespunde valorii maxime a  CAD conform rezolutiei //maxime a acestuia – 10biti
(1023=2^10-1);
Serial.println(„Valoare tensiune [V]:”);
//  afisarea pe monitorul serial a valorii calculate, insotite de textul //explicativ
Serial.println(tensiune);
delay(100);
}
```

Valorile tensiunii, la mutarea cursorului potenţiometrului, vor fi afişate în Monitorul Serial, ca în Fig. 4.6.

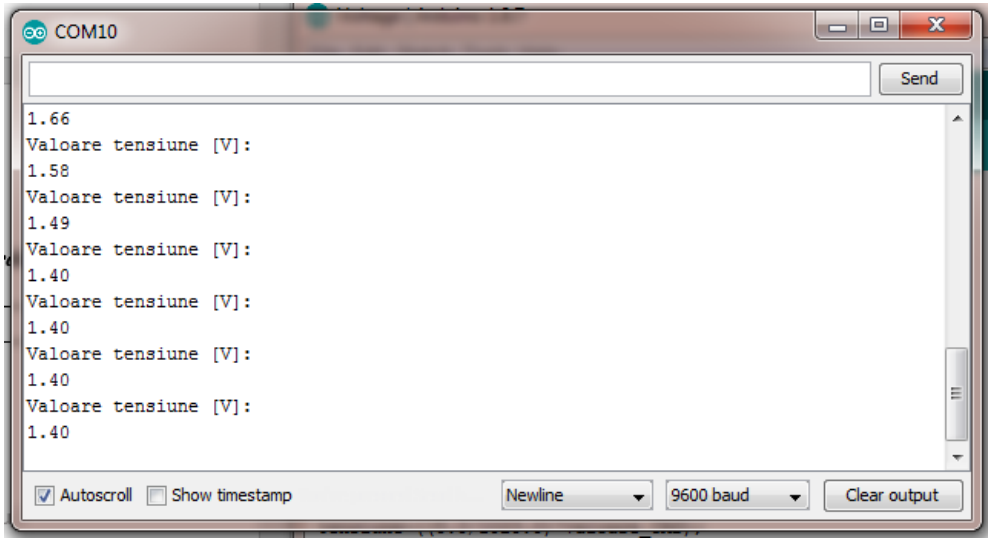


Fig.4.6. Afișare valori de tensiune în monitor serial

4.2 Implementarea unui comparator numeric cu prag reglabil

Aplicaţia propune introducerea în circuitul exemplului anterior a două LED-uri de culori diferite, care să semnalizeze – prin comanda numerică, în funcţie de un prag de tensiune impus prin program - când valoarea de tensiune citită prin potenţiometru depăşeşte pragul creat sau când nu îl atinge. LED-urile se comandă digital, în funcţie de o valoare analogică primită, indiferent de elementul introdus în circuit: traductor, senzor, potenţiometru, astfel încât aplicaţia de semnalizare devine utilă în orice exemplu de automatizare casnică, de detecţie, monitorizare sau control al unei situaţii sau a unei mărimi fizice.

a) Conectarea hardware a aplicaţiei, folosind citirile de la potentiometru şi prag reglabil de tensiune

Circuitului prezentat în Fig. 4.5. îi sunt adaugate 2 LED-uri (de exemplu, verde şi rosu), conform schemei din Fig.4.7. Din punct de vedere software, în cod se impune un prag de tensiune şi se va implementa un algoritm „if/else” pentru cazurile:

- tensiunea citită pe pinul analogi A0 > *prag*, atunci LED roşu este ON şi LED verde OFF
- tensiunea citită pe pinul analogi A0 < *prag*, atunci LED verde este ON şi LED roșu OFF

- Platforma de dezvoltare
- Cablu USB
- Cablu de alimentare al platformei (în cazul Intel Galileo)
- Breadboard
- Potenţiometru 10k
- Fire
- 2 LED-uri
- 2 rezistente 220Ω

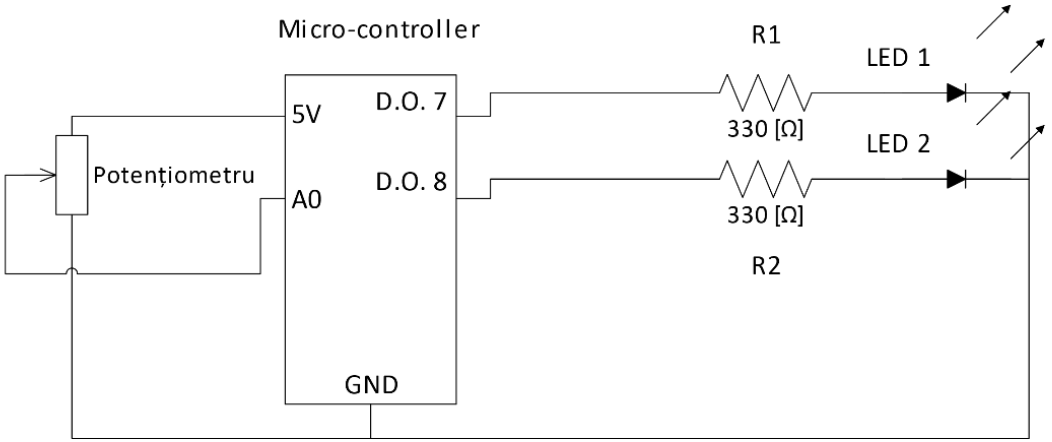


Fig. 4.7. Schema electrică a montajului pentru aplicația 4.2.

Implementarea algoritmului prin mediul Arduino IDE:

```
const int pin_analogic=A0;
int valoare_CAD=0;
float tensiune=0;
//declararea variabilei tensiune, de tip float
const int LED_rosu=10;
const int LED_verde=9;
const int prag=2.5;
//pragul de tensiune este stabilit la jumatatea gamei citite 0 – 5 V

void setup()
```

```
{
  pinMode(LED_rosu, OUTPUT);
  pinMode(LED_verde, OUTPUT);
  Serial.begin(9600);
// deschide comunicarea pe portul serial la 9600 bps:
}
void loop()
{
valoare_CAD=analogRead(pin_analogic);
tensiune=((5.0/1023.0)*valoare_CAD);
//tensiunea maxima 5V corespunde valorii maxime a  CAD conform //rezolutiei maxime a acestuia – 10biti
(1023=2^10-1);
//cazurile algoritmului if/else, explicate anterior
if (tensiune >prag){
digitalWrite(LED_rosu, HIGH);
digitalWrite(LED_verde, LOW);}
else {
digitalWrite(LED_verde, HIGH);
digitalWrite(LED_rosu, LOW);}

//afisarea pe monitorul serial a valorii calculate,insotite de  textul //explicativ
Serial.println(„Valoare instantanee tensiune [V]:”);
Serial.println(tensiune);
delay(100);
}
```

Pragul de tensiune impus este comparat cu nivelul de tensiune citit de la intrarea analogică A0. Semnalul de tensiune variabil în timp în acest caz este furnizat de la un potenţiometru. În cazul în care se foloseşte un senzor, se realizează calibrarea corespunzătoare, iar mărimea fizică citită prin acesta va fi comparată cu un prag numeric ales în gama acestei mărimi, după cum este explicat în aplicaţiile următoare ale acestui capitol.

4.3 Implementarea unei coloane luminoase indicatoare de nivel

Prin această aplicaţie, se propune implementarea unui şir de LED-uri, a căror activare se realizează cu valorile citite de pe pinul analogic, prin potenţiometru. Valorile respective vor fi „aliniate" proporţional, prin algoritmul matematic, pentru a face activarea succesivă a LED-urilor. În scopul adaptării sau alinierii acestor valori citite prin convertorul analog-digital se foloseşte în Arduino IDE, funcţia:

map(value, fromLow, fromHigh, toLow, toHigh);

care rearanjează vaolarea *"value"* de la gama *(fromLow, fromHigh)* la gama *(toLow, toHigh).*

Aceasta funcţie se foloseşte des în aplicaţii în care se fac readaptări ale unor valori numerice, de la o gamă la o alta, fără constrângeri legate de limitele acestor game; de exemplu, se poate folosi când se încearcă reversarea unei game, ca în exemplul:

```
y=map(x, 0, 150, 150, 1);
```

```
// fiecărei valori din gama [0,150] i se asociază o valoare în gama //[150,0].Valorile asociate sunt întotdeauna de tip întreg.
```

Conectarea hardware a aplicaţiei de tip coloană luminoasă indicatoare de nivel

Schema de montaj se realizează în mod asemănător exemplului din capitolul III, Fig. 3.21, căruia i se mai adaugă potențiometrul pe intrarea analogică A0.

- Platforma de dezvoltare
- Cablu USB
- Cablu de alimentare al platformei (în cazul Intel Galileo)
- Breadboard

- 8 LED-uri
- 8 Rezistenţe: 330 Ω, respectiv 10kΩ
- Potenţiometru 10k
- Fire

Implementarea algoritmului prin mediul Arduino IDE:

```
const int pin_analogic=A0;
const int LEDs=8; //se declara numarul de LED-uri
int CAD=0; //valoarea citita prin CAD
int nivel=0; //nivelul sau valoarea cu care se face compararea valorilor citite
int pin_LED[]={3, 4, 5, 6, 7, 8, 9, 10};
//matrice de LED-uri (vezi aplicatia 3.3)

void setup()
{//declararea LED-urilor ca iesiri, folosind matrice si bucla de tip „for” (vezi aplciatia 3.3.)
           for (int LED=0; LED<LEDs; LED++){
             pinMode(pin_LED[LED], OUTPUT); }
                                }
void loop()
{
CAD=analogRead(pin_analogic);
//prin folosirea functiei map se calculeaza o valoare integer „nivel” prin care se asociaza valori din gama
//(0-1023) a convertorului analog digital gamei (0-8) care reprezinta numarul de LED-uri;
nivel=map(CAD, 0, 1023, 0, LEDs);
//se interogheaza fiecare din cele 8 LED-uri, si cand valoarea constantei „nivel” este mai mica
for (int LED=0; LED<LEDs; LED++){
      if (LED<nivel){
```

```
          digitalWrite(pin_LED[LED], HIGH);}
      else{
          digitalWrite(pin_LED[LED], LOW);}
         }
}
```

4.4 Implementarea unei aplicații folosind senzor analogic

Așa cum s-a explicat în paragrafele anterioare, folosirea majorității senzorilor pe pinii analogici ai sistemelor embedded se poate generaliza în doi pași simpli:

- *Convertirea valorii citite prin pinul analogic în valoare de tensiune*
- *Prelucrarea valorii de tensiune, în valoari din gama mărimii fizice preluate de senzor, prin realizarea unei* ***calibrări corespunzătoare***

În aplicația de față, se prezintă cazul simplu și des folosit, al senzorului de temperatură LM3, Texas Instruments. Modul de conectare a senzorului la placă se realizează conform Fig. 4.8. și a datelor din catalogul senzorului. Conectarea pinilor trebuie facută corespunzător, astfel încât să se evite scurtcircuitarea senzorului; astfel, vederea din Fig. 4.8 pentru reperarea pinilor, se realizează cu partea teșită a senzorului înspre utilizator.

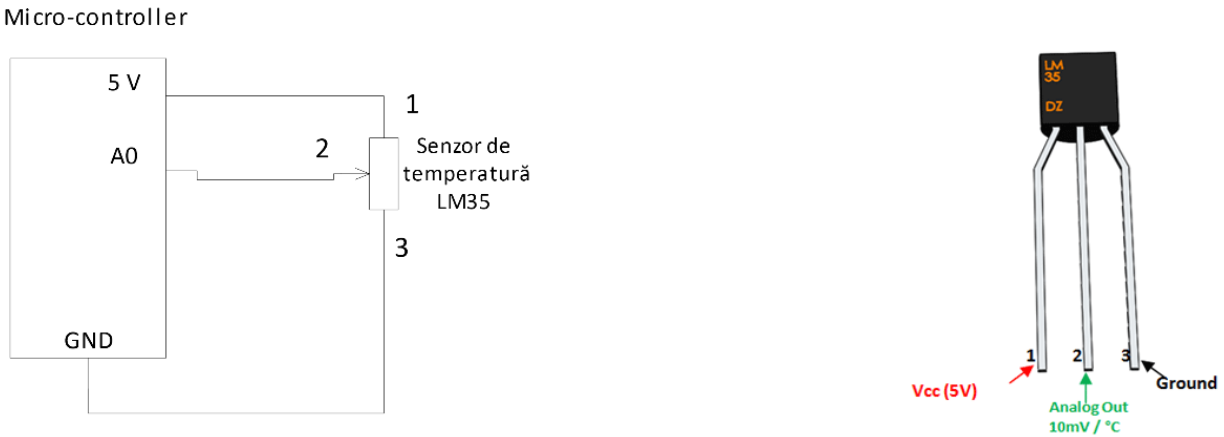


Fig.4.8. Conectarea senzorului de temperatură LM35

Componentele necesare pentru realizarea circuitului sunt următoarele:

- Platforma de dezvoltare
- Cablu USB
- Cablu de alimentare al platformei (în cazul Intel Galileo)
- Breadboard
- Senzor de temperatură

Algoritmul de citire a valorilor senzorului, cu explicațiile aferente sunt prezentate în continuare:

Implementarea algoritmului prin mediul Arduino IDE:

```
const int pin_analogic=A0;
int CAD=0; //valoarea citita prin CAD
```

```
float tensiune=0;
//declararea variabilei tensiune, de tip float, masurata in V
//float tesiune_mV;
float temp=0;
//declararea variabilei temp, de tip float, pentru exactitate
void setup()
{
  Serial.begin(9600);
// deschide comunicarea pe portul serial la 9600 bps:

}
void loop()
{
valoare_CAD=analogRead(pin_analogic);
tensiune=((5.0/1023.0)*valoare_CAD);
//tensiunea maxima 5V corespunde valorii maxime a  CAD conform rezolutiei maxime a acestuia-10
//biti=(1023=2^10-1);
tensiune_mV=tensiune*1000;
//relatia matematica de calcul a temperaturii, sub forma constantei temp se bazeaza pe formula de
//calibrare data de producator; in acest caz: 10mV/1^0C
     temp=tensiune_mv/10;
// afisarea pe monitorul serial a valorii de temperatura masurate, insotita de text explicativ
Serial.println(„Valoare temperatura [^0C]:”);
Serial.println(tensiune);
Delay(100);
}
```

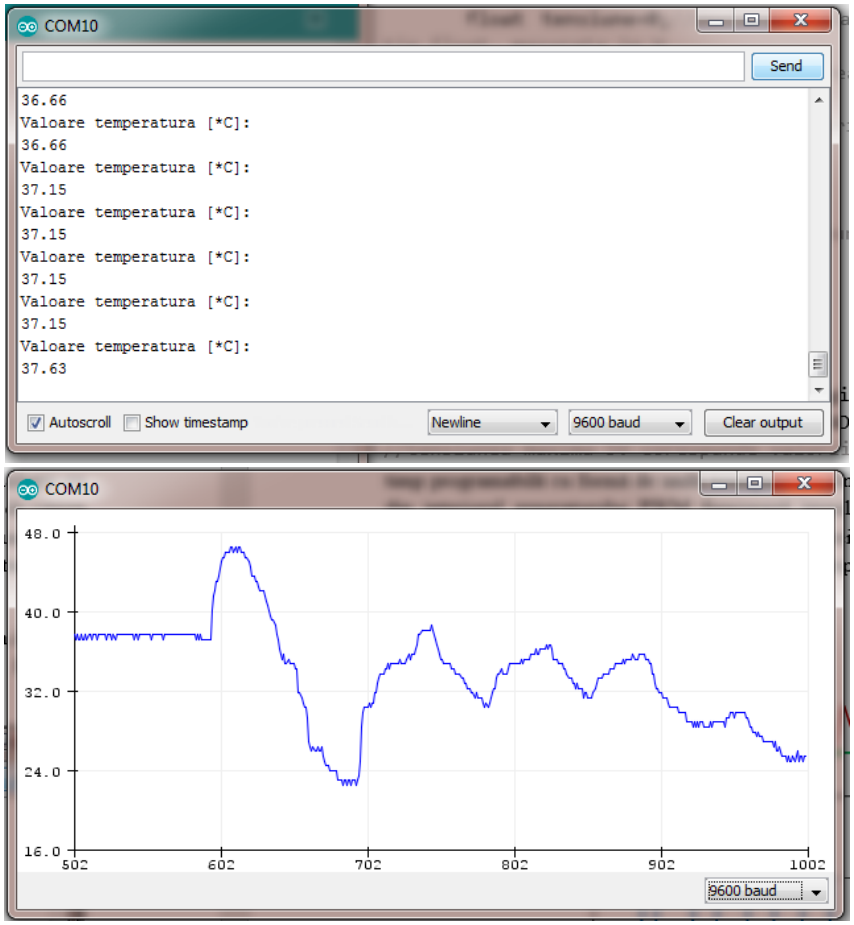


Fig. 4.9. Vizualizarea temperaturii citite cu senzorul LM35

Generalizând, se poate spune că, în cazul folosirii oricărui senzor analogic cu formulă de calibrare liniară, singura implementare matematică diferită față de citirea valorii CAD sau a nivelului de tensiune din aplicația 4.1. este cea corespunzătoare calibrării. Modificări aferente se fac în cazul în care tensiune de alimentare este diferită de 5 V sau convertorul CAD are rezoluția diferită de 10 biți.

Rezultatele citirii, se pot vizualiza în timp real în monitorul serial din Arduino IDE. Totodata, se pot vizualiza și sub forma unui grafic în timp real, folosind elementul “Serial Plotter” din meniu, dupa cum se observa în Fig. 4.9.

Într-o aplicație mai complexă, se pot realiza filtrări hardware, dar mai ales software pentru valorile citite de pe senzorul de temperatură. **De exemplu, se poate introduce o buclă de tip for pe un anumit interval, iar citirilor făcute în acest interval li se aplică o medie aritmetică, calculându-se valoarea finală a temperaturii ca o medie a valorilor citite in interval.**

4.5 Generarea semnalelor dreptunghiulare modulate în durata impulsului cu comandă analogică dintr-un potențiometru de control

Atunci când se realizează operația inversă a convertorului analog/numeric, adică, furnizare unui semnal cvasi–analogic prin cuplarea succesivă a unor nivele de tensiune artificial construite printr-un divizor rezistiv multiplu, se folosește un convertor digital analogic; semnalul furnizat de un astfel de dispozitiv este unul discretizat/numeric. În acest sens, aplicația de față, prezintă implicarea tehnicii PWM pentru controlul „analogic” a unui LED. Generatorul PWM de undă dreptunghiulară modulată în lățime (Pulse Width Modulation) constă într-un comparator, un convertor digital–analogic și o bază de timp programabilă cu formă de undă triunghiulară/dinte de fierăstrău. Comparatorul din interiorul generatorului PWM furnizează impulsuri la intersecția semnalului produs de convertorul digital analog cu semnalul bazei de timp (forma triunghiulară). Frecvența bazei de timp redă frecvența trenului de impulsuri generat (Fig.4.10).

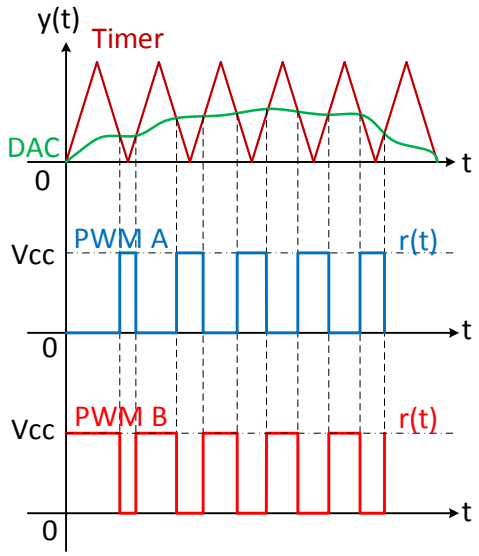


Fig.4.10. Generarea semnalului modulat PWM

În acest sens, circuitului din Fig. 4.6., cu potențiometrul conectat la intrarea analogică A0, i se adauga un LED, pe pinul digital 9 (doar pinii digitali prevăzuți pe platforma cu simbolul "~" sau în catalogul de la producător pot fi folositi pentru PWM).

Implementarea algoritmului prin mediul Arduino IDE:

```
const int pin_analogic=A0;
int valoare_CAD =0;
//valoarea citita prin CAD
int factor_umplere=0;
const int LED=9;
```

```
void setup()
{     pinMode(LED,OUTPUT);
      Serial.begin(9600);
// deschide comunicarea pe portul serial la 9600 bps:
}
void loop()
{
      valoare_CAD=analogRead(pin_analogic);
//folosind functia map se face alinierea asocierea valorilor din gama citirilor analogice (0-1023) la cea
//a digitalului – LED-ul in acest caz – (0-255);
      factor_umplere=map(CAD, 0, 1023, 0, 255);
      Serial.println(factor_umplere);
 //se afiseaza valoarea calculata in monitorul serial
      analogWrite(LED, factor_umplere);
 //scriere de valori analogice pe pin digital →LED -  PWM
}
```

O variantă a acestei aplicații o reprezintă **introducerea valorilor de comandă a elementului digital din consola de comunicare serială, prin tastatură**; potențiometrul nu mai este folosit în acest caz. Algoritmul implementat în Arduino IDE este prezentat în continuare:

```
const int LED = 9;
int valoare_primita = 0;
int factor_de_umplere = 0;
void setup() {
        pinMode(LED, OUTPUT);
        Serial.begin(9600);
        Serial.println("Introduceti factorul de umplere 0 - 255: ");
```

```
}
void loop() {
//transmiterea valorilor de la tastatura, prin consola seriala se realizeaza impunand, prin interogari de
//tip “if”, limitarea valorilor in gama digitala 0-255 si returnare mesaj de eroare in caz contrar.
     if(Serial.available()){
     valoare_primita = Serial.parseInt();
     if(valoare_primita < 0 || valoare_primita > 255){
          Serial.println("Valoarea introdusa nu este in intervalul 0 - 255!");
                    valoare_primita = 0;
Serial.println("Valoarea factorului de umplere introdusa este: ");
Serial.println(valoare_primita);
  }}
//in cazul transmiterii unei valori corecte, se face comanda LED-ului, cu valoarea de tensiune
//corespunzatoare
analogWrite(LED, valoare_primita);
}
```

Pentru exersare, se propun spre experimentare variante ale aplicațiilor prezentate, folosind principiile de bază și diferite componente/senzori/traductori. De exemplu:

- **Controlul factorului de umplere pentru reglarea intensității luminoase în funcție de temperatură**
- **Implementarea unui comparator numeric cu prag reglabil în funcție de semnalul analogic măsurat printr-o fotorezistență**
- **Implementare control PWM pentru motor de curent continuu, în funcție de temperatura din ambient (aplicație de comandă sistem de încălzire/răcire)**

Următoarele două aplicații propun variații ale temelor 4.1.- 4.5. în ***abordarea folosind pachetul Matlab – Simulink;*** se folosesc setările prezentate în Cap.2 și reperele din exemplele de aplicații din Cap.3.

4.6 Aplicație de comandă PWM a unui motor de curent continuu

a. Cu element de tip slider

Schema bloc a aplicației implementată într-un model Simulink este prezentată în Fig. 4.11. Urmărind pașii specifici de implementare, blocurile constructive sunt prezentate în continuare:

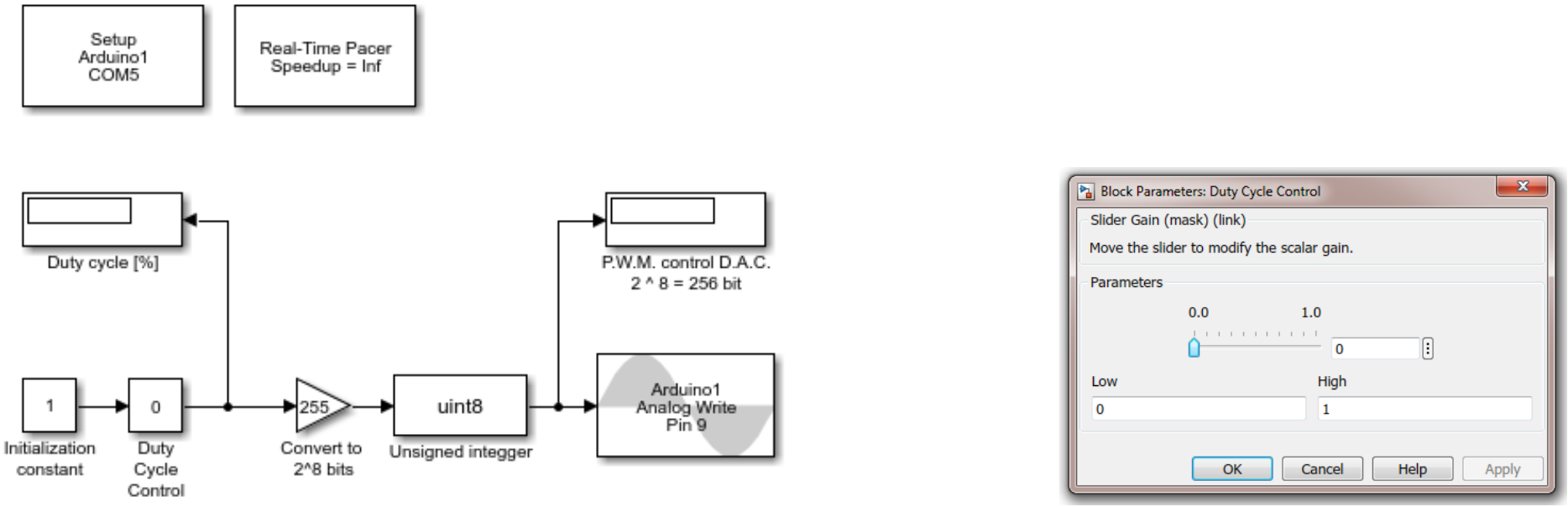


Fig.4.11. Modelul Matlab/Simulink pentru aplicația 4.6. – cu ‚slider'

Blocul pentru înmulțire cu variabilă specificată prin glisarea unui cursor numeric: - „Slider Gain" permite modificarea manuală, în timp real a factorului de umplere prin înmulțire cu variabila impusă prin cursor. Se inițializează cu „1" pentru un raport 1:1.

Blocuri pentru afișare – „Display” (în aplicație, pentru factor de umplere); Prin intermediul acestora se poate vizualiza valoarea numerică instantanee, zecimal - procentuală a mărimii furnizate la intrare (factorului de umplere 1 = 100 %; 0.5 = 50 %);

Blocul de conversie procent în biți „convert” - Se foloseste prin înmulțirea valorii zecimale a procentului cu „255=2^8-1” (ex. 0.5 * 255 = 127.5);

Blocul de conversie a numărului fracționar la număr întreg cu rotunjire: Valorile fracționare obținute în urma înmulțirii procentului cu 255 se rotunjesc la un număr întreg cu reprezentare pe 8 biți fără semn (ex. 127.5 = 128);

Ieșire port digital PWM - Cu ajutorul acestui bloc, se furnizează la ieșirea digitală cu numărul stabilit în acest bloc un **tren de impulsuri cu frecvență constantă**, dar cu **lățime variabilă**, în funcție de valoarea numerică a factorului de umplere, impusă de utilizator (duty cycle).

b. Cu potențiometru sau altă componentă analogică

În acest caz al aplicației, controlul PWM al motorului se realizează indicând factorul de umplere sau duty cycle din citirea unei componente hardware din circuitul platformei de dezvoltare. Această componentă poate fi un potențiometru sau un alt senzor analogic.

Atât pașii de implementare, cât și blocurile folosite au fost detaliate în aplicațiile anterioare.

Modelul matematic implementat prin Matlab/Simulink al aplicației este indicat în schema bloc din Fig. 4.12.

Modul de lucru este similar cu cel al aplicației anterioare doar că, în locul blocului de înmulțire cu cursor reglabil (‘slider gain’) se utilizează un mijloc analogic de furnizare a datelor numerice (ex. un potențiometru). Conversia numerică de la 10 biți ($2^{10} – 1 = 1023$) la 8 biți ($2^8 – 1 = 255$) este realizată prin intermediul înmulțirii cu raportul celor două valori maxime 255: 1023 = 0,249.

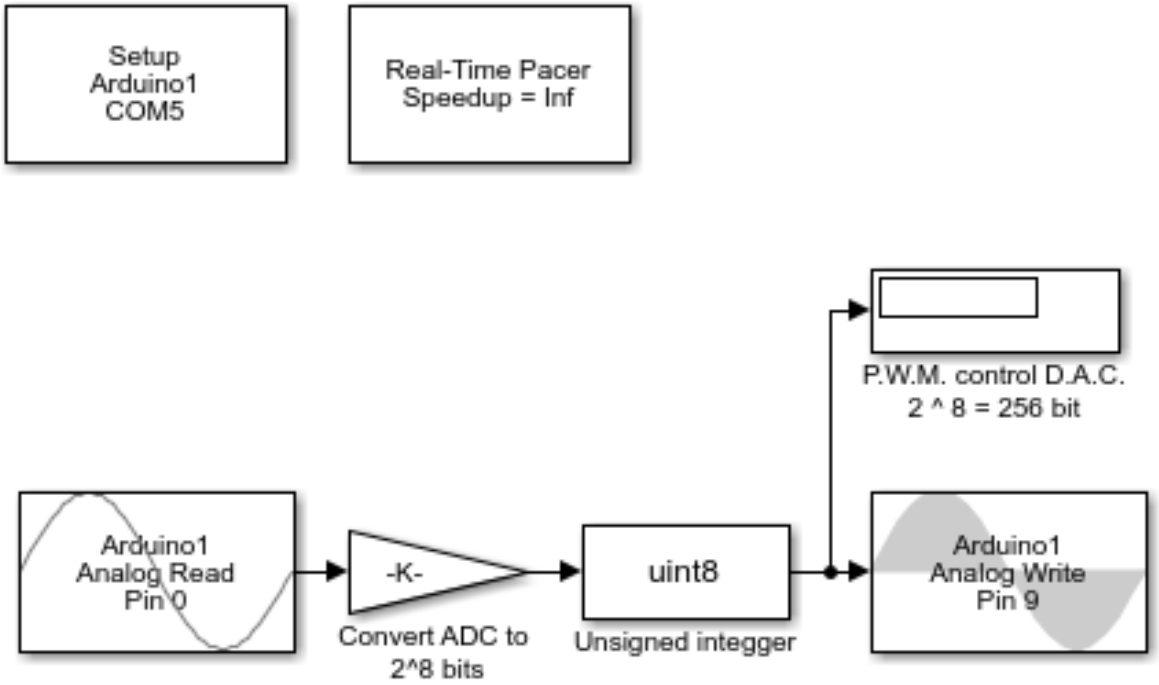


Fig.4.12. Modelul Matlab/Simulink pentru aplicația 4.6. – cu potentiometru

Citirea instantanee a rezultatelor și vizualizarea se realizeaza pe elemente de tip osciloscop se realizează după cum se observă in Fig. 4.12.

Concluzionând, **înțelegerea funcționării convertorului analog digital al unei platforme/sistem cu microcontroler, a folosirii divizorului de tensiune și a funcțiilor principale specifice manipulării** software din Arduino IDE în acest caz, sunt repere necesare și suficiente pentru manipularea unei varietăți largi de senzori analogici și realizarea de aplicații de 'sesizare' a mediului ambient, indiferent de abordarea software folosita.

5. APLICAȚII DIVERSE

Capitolul de față propune prezenarea unui set de aplicații complexe, în contextul folosirii tehnologiilor existente pentru comunicarea datelor, atât la nivel aplicație - computer gazdă, cât și la distanță; în continuă dezvoltare, aplicațiile de tip "remote-control" sau control la distanță, sunt utile în industrie, au ecou în economia modernă, dar cunosc un avânt deosebit mai ales în rândul aplicațiilor personale, cu concentrare pe mărirea confortului și a calității vieții.

5.1 Comunicare cu platforme de dezvoltare Arduino: UART, I2C și SPI

În contextul aplicatiilor de interfațare, metodele de comunicare a datelor între platformele de dezvoltare și dispozitivul gazdă prezintă un interes în dezvoltare. Corelat cu tipul aplicației, comunicarea pe magistrala serială I²C (*Inter-Integrated Circuit)* și SPI (*Serial Peripheric Interface*) sunt considerate protocoale "simple" de comunicație, comparativ cu Ethernet, USB, SATA, PCI-Express, care prezintă un anumit grad mai ridicat de complexitate. Totuși, ca și destinație, Ethernet, USB, SATA sunt folosite pentru "comunicații în afara cutiei" și pentru schimb de date între sisteme. Atunci când se dorește comunicația între circuite integrate (microcontroler) și periferice relativ rapide I2C, UART și SPI au devenit foarte populare printre utilizatorii și dezvoltatorii de sisteme embedded.

UART - Universal Asynchronous Reception and Transmission este un protocol simplu de comunicare care permite schimbul de date dintre platforma de dezvoltare și dispozitive seriale gazdă. În general, toate microcontrolerele prezintă pini digitali special dedicați comunicației UART: ‚RX – pinul de recepție' ‚TX – pinul de transmisie'(X este simbolul pentru orice date care ar putea fi vehiculate pe magistrală). Platforma de dezvoltare Arduino, fiind construită în jurul arhitecturii AVR – ATMega 328p, prezintă doi pini digitali pentru acces la magistrala UART de comunicare serială cu alte dispozitive sau cu calculatorul gazdă. Platforma Arduino, înglobează și un adaptor dedicat USB – serial TTL.

Astfel, orice program poate fi folosit cu platformele Arduino, prin conectarea la portul USB, ca și cum ar fi un port serial. Nu necesită semnal de ceas, comparativ cu SPI și I2C, pentru că utilizatorul dă dispozitivelor informațiile necesare despre temporizarea aplicației.

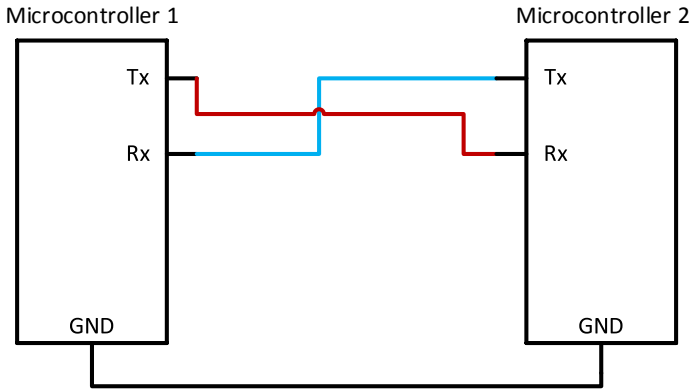


Fig.5.1. Pinii de comunicare serială UART între două microcontrolere

Câțiva termeni specifici pentru comunicarea UART sunt amintiți în continuare:

- Start bit: primul bit din transmisia unui byte prin UART; faciliteaza comunicatia intre RX si TX dar nu transfera date importante.
- Stop bit: ultimul bit din transmisia unui byte prin UART.
- Baud rate: rata aproximativă la care se transferă datele (bits per secunda sau bps) - reprezintă frecvența (în bps) de transmisie a unui bit de date. Un sistem cu baudrate 9600 bps realizează transmisia unui bit în 1/9600 s ≈ 104.2 µs.
- Sistemul nu poate, de fapt, să transmită real 9600 de biți semnificativi de date într-o secundă, din cauza timpilor pentru întârzieri între bytes sau biții de start/stop;

I2C – Inter Integrated Circuit

I2C - Inter-Integrated-Circuit reprezinta un protocol de comunicare serială, special creat pentru microcontrolere, popular în utilizarea senzorilor și modulelor în proiecte ample, cu multe componente, care interacționează. Folosirea acestui tip de comunicație permite utilizarea a până

la 128 de dispozitive într-un circuit. Principiul de interconectare a dispozitivelor este cel a unui dispozitiv central – "master" (de obicei un computer sau un procesor embedded) și a dispozitivelor cu care comunică ("slave" – de obicei, senzori).

I2C face posibilă conectarea de tip *"multiple masters, multiple slaves"*, menținând calea de comunicație și folosind recunoașterea/apelarea dispozitivelor pe bus-ul comun prin adresele alocate manual fiecăruia, în circuit. Avantajul major al folosirii acestui tip de comunicație rezidă din folosirea a doar două fire de conectare în circuit (CLK –Clock si Data), cum se observă în Fig. 5.2., indiferent de numărul de dispozitive

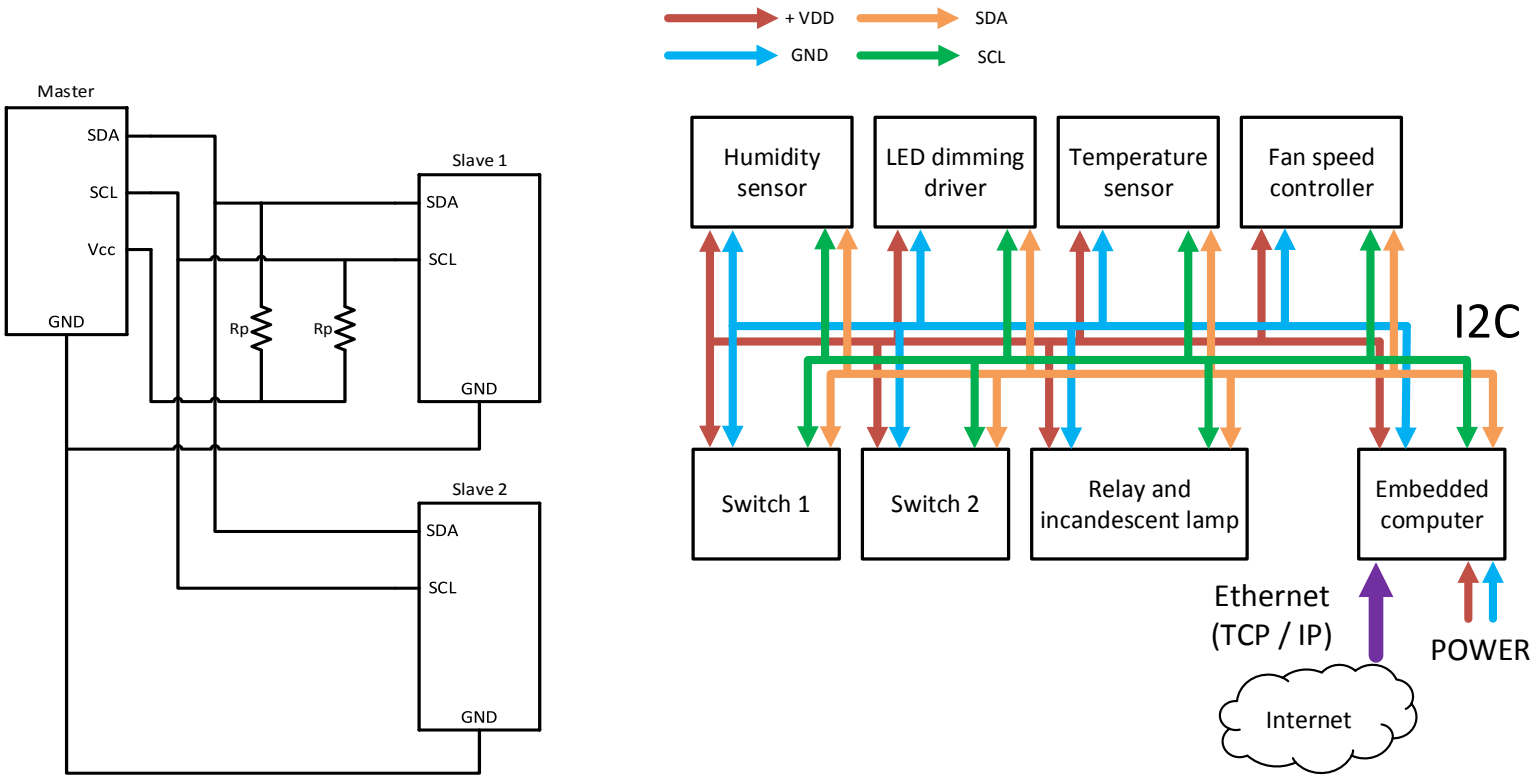


Fig.5.2. Exemplu de aplicație cu dispozitive care comunică pe magistrala I2C

Din demersul de comunicare I2C, sunt prezentate sintetic câteva repere:

- Fiecare dispozitiv slave din circuit se identifică printr-o adresă pe 7 biți;
- master-ul trebuie să cunoască aceste adrese pentru a putea comunica;
- toate transmisiile sunt inițiate și terminate de un master și tot acesta poate scrie datele la unul sau multe dispozitive slave sau să ceară date de la un dispozitiv slave;
- orice dispozitiv poate funcționa ca master sau slave; în practică, însă, referirea la sisteme embedded se face mai ales în configurație: un master care comunică cu mai multe dispozitive slave;

Aplicație – variator de tensiune continuă controlat numeric / digital:

Conform principiului de funcționare al aplicației 3.5 „controlul digital al turației unui motor de curent continuu” descrise în capitolul 3 a fost realizat fizic un modul de tip variator de tensiune continuă cu comandă digitală având un singur element comutator (tranzistor).

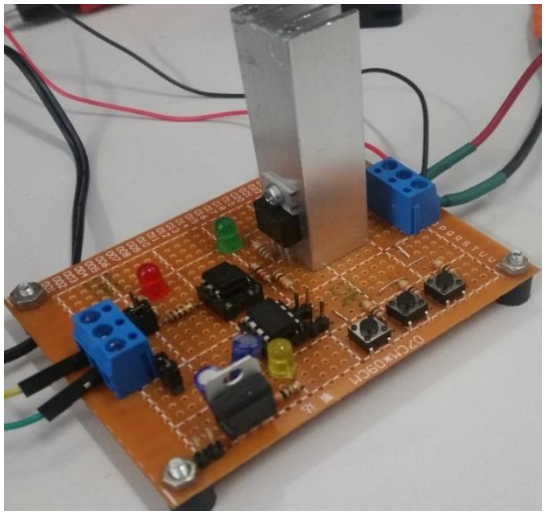


Fig.5.3. Modul variator de tenisune digital

În componența modulului astfel realizat, pe lângă circuitul descris în figura 3.32 din capitolul III a mai fost adăugat un microcontroler ATTiny 45, trei butoane cu acțiune momentană, trei diode LED electroluminiscente, un stabilizator liniar de tensiune (5 [V]), și două mufe cu șurub pentru racord. Modulul are posibilitatea de a fi reconfigurat în funcție de aplicația dorită. Acest lucru se poate realiza prin intermediul unor pini cu punte de conexiune (‘jumper-i’).

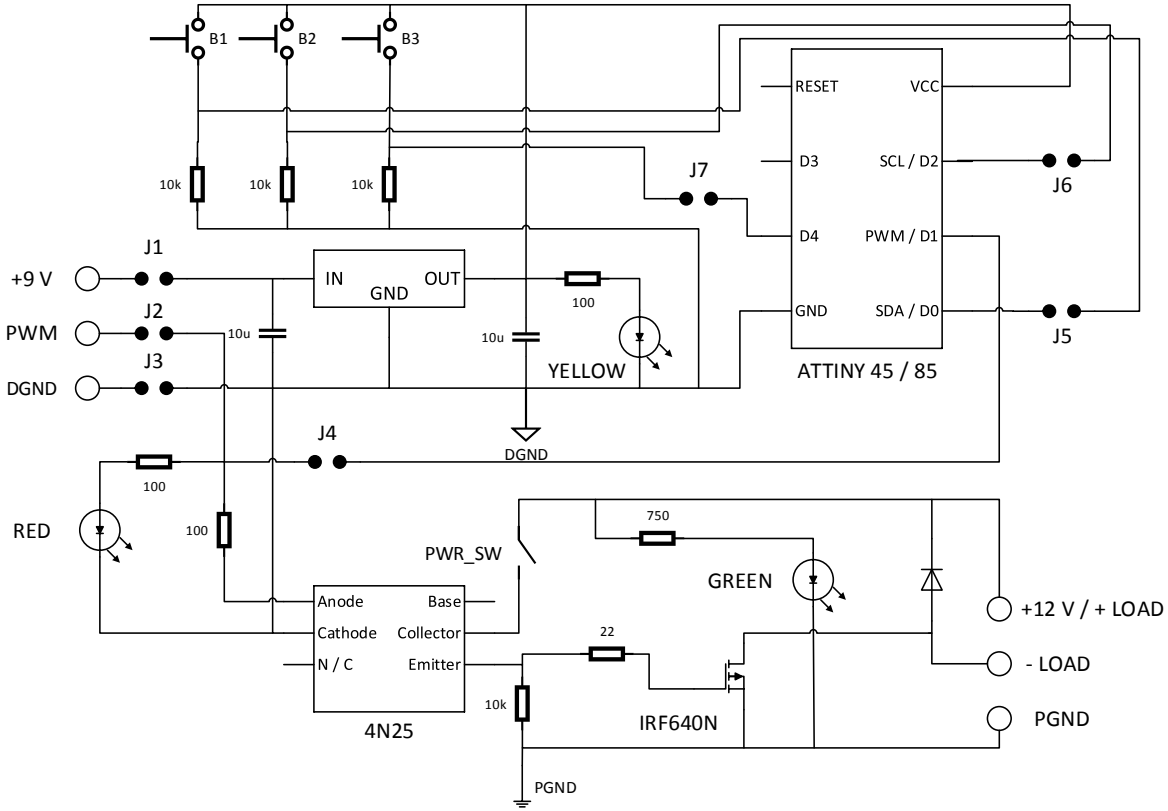
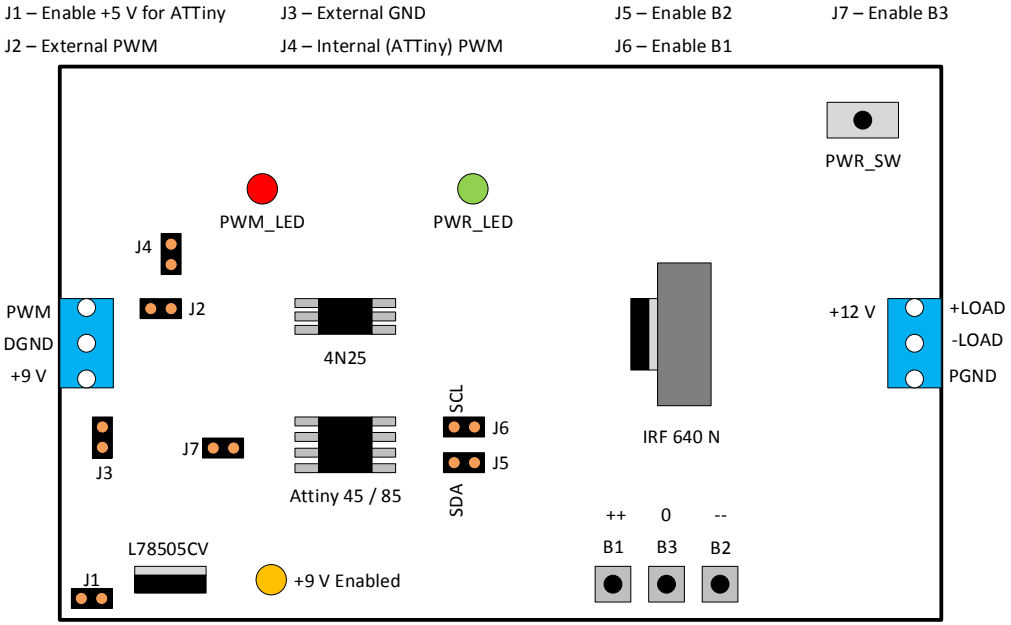


Fig.5.4. Schema electronică a modulului variator de tensiune

Astfel, modulul poate funcționa în trei moduri de lucru: comandă digitală externă PWM, comandă digitală externă I2C, comandă digitală internă PWM cu incrementare sau decrementare de la butoane (ATTiny 84) – funcționare autonomă. Practic, un astfel de modul poate fi considerat un convertor electronic de putere „inteligent" care poate recepționa comanda digitală de la un sistem de calcul sub diferite forme (ex. comandă fizică – 'hard – PWM' sau comandă numerică – ‚software – I2C'). Un aspect important de menționat constă în faptul că, în afara modului de lucru extern cu comandă PWM, în celelalte două moduri este necesară programarea microcontrolerului ATTiny 45. Atât pentru modul de lucru ‚I2C slave' cât și pentru modul de lucru din butoanele din circuit, există două programe realizate în acest sens.



Comandă digitală externă PWM	Comandă digitală internă PWM	Comandă digitală externă I2C
J1 - Absent	J1 - Prezent	J1 - Prezent
J2 - Prezent	J2 - Absent	J2 - Absent
J3 - Prezent	J3 - Prezent	J3 - Prezent
J4 - Absent	J4 - Prezent	J4 - Prezent
J5 - Absent	J5 - Prezent	J5 - Absent
J6 - Absent	J6 - Prezent	J6 - Absent
J7 - Absent	J7 - Prezent	J7 - Absent

Fig.5.5. Modurile de lucru ale variatorului de tensiune continuă

Programul încărcat în memoria microcontrolerului ATTiny 45 pentru modul de lucru „Comandă digitală internă PWM”:

```
const int pwm_pin = 1;
const int buton_crescator = 0;
const int buton_descrescator = 4;
const int buton_avarie = 2;

int factor_de_umplere = 0;
int stare_buton_crescator = 0;
int stare_buton_descrescator = 0;
int stare_buton_avarie = 0;

void setup() {
  pinMode(buton_crescator, INPUT);
  pinMode(buton_descrescator, INPUT);
  pinMode(buton_avarie, INPUT);
  pinMode(pwm_pin, OUTPUT);
}

void loop() {
 stare_buton_crescator = digitalRead(buton_crescator);
 stare_buton_descrescator = digitalRead(buton_descrescator);
 stare_buton_avarie = digitalRead(buton_avarie);

 if(stare_buton_crescator == HIGH){
      factor_de_umplere++;
      if (factor_de_umplere >= 255){
        factor_de_umplere = 255;
      }
```

```
    }

  if(stare_buton_descrescator == HIGH){
      factor_de_umplere--;
      if (factor_de_umplere <= 0){
        factor_de_umplere = 0;
      }
    }

  if(stare_buton_avarie == HIGH){
      factor_de_umplere = 0;
      if (factor_de_umplere == 0){
        factor_de_umplere = 0;
      }
    }
  analogWrite(pwm_pin, factor_de_umplere);
delay(25);
}
```

Programul încărcat în memoria microcontrolerului ATTiny 45 pentru modul de lucru „Comandă digitală externă I2C”:

```
#define I2C_SLAVE_ADDRESS 0x08 // Address of the slave
#include <TinyWireS.h>
int pwm_pin = 1;
int duty_cycle = 0;

void setup() {
    TinyWireS.begin(I2C_SLAVE_ADDRESS); // join i2c network
```

```
    TinyWireS.onReceive(receiveEvent);
    pinMode(pwm_pin, OUTPUT);
}

void loop() {
    analogWrite(pwm_pin, duty_cycle);
    TinyWireS_stop_check();
}

void receiveEvent() {
    if (TinyWireS.available()) {
      duty_cycle = TinyWireS.read();
      if (duty_cycle >= 255) {
        duty_cycle == 255;
      }
    if (duty_cycle <= 0) {
        duty_cycle == 0;
      }
    }
}
```

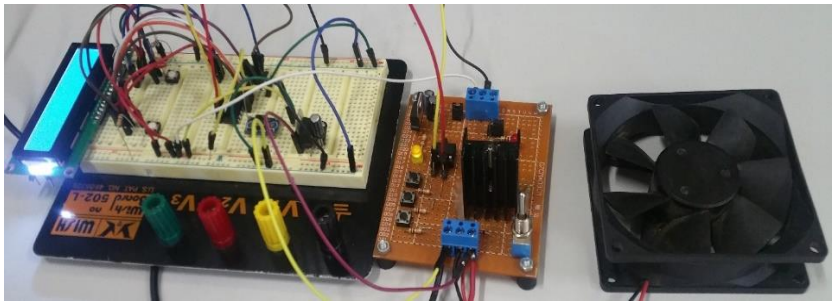


Fig.5.6. Aplicație - Controlul digital al turației unui ventilator prin intermediul microcontrollerelor ATMega 328P și ATTiny 45 – Studiul comunicației I2C

SPI – Serial Peripheral Interface

SPI reprezintă o formă de protocol de comunicare serială proiectat special pentru comunicarea cu dispozitivele periferice.

- permite utilizarea unui singur dispozitiv master cu maxim 4 dispozitive slave;
- în mod fizic protocolul SPI necesită patru terminale de conexiune pentru intrare / ieșire (SCK – Serial ClocK, MOSI – Master Output Slave Input, MISO – Master Input Slave Output, CS – Chip Select);
- este mai rapid decat I2C - liniile de Data / Clock sunt împărțite între dispozivite, fiecare necesitând o adresă unică;
- se folosește în aplicații de comunicații unde viteza este importantă: SD cards, module de afișare sau atunci când actualizarea informațiilor (de ex. la citirea în timp real într-un browser) trebuie să fie rapida;

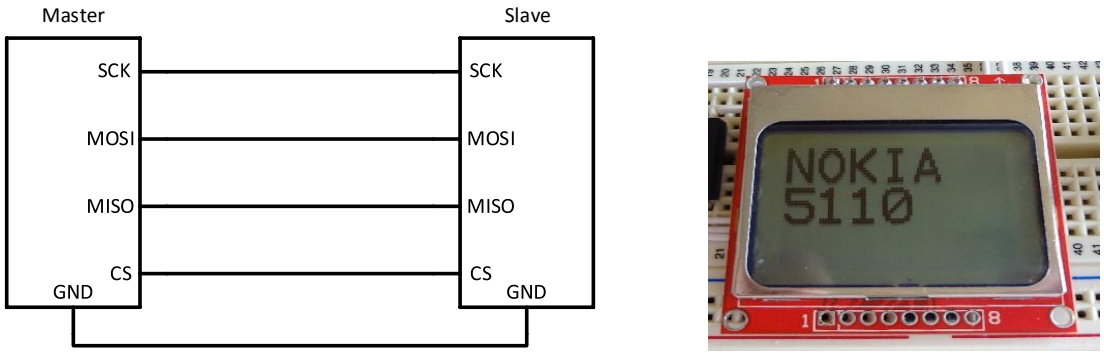


Fig. 5.7. Nokia 5110 dot-matrix SPI LCD - http://pic18fxx.blogspot.com/2015/10/nokia-5110-lcd-library.html]

5.2 Interfața jTAG - Schimbul de date între sisteme de calcul

Într-o aplicație cu *sisteme numerice de calcul cu răspuns în timp real* se are în vedere manipularea datelor de intrare și ieșire pe baza unor mărimi de comandă. În acest scop este necesară introducerea unui protocol sau canal de comunicație între echipamente. În cazul unei aplicații industriale, se are în vedere interacțiunea dintre un sistem de calcul primar de tip *computer gazdă - microprocesor cu sistem de operare* și un alt *echipament numeric secundar dedicat* în a deservi o aplicație fizică: *procesoare digitale de semna*l eng. **Digital Signal Processor – DSP** sau elemente de interfațare pe bază de microcontroler / automate programabile (Programmable Logic Controller – PLC).

În orice situație, aplicația dedicată sau logica de control a acesteia, ce rulează pe un sistem de calcul dedicat (ex. controlul turației unui ventilator industrial) trebuie să permită accesul utilizatorului din exterior asupra parametrilor de comandă și control. Astfel, modul în care este concepută aplicația din punct de vedere al interfațării și interacțiunii cu utilizatorul, are un rol important; de exemplu modul de declarare a parametrilor de control ca variabile globale, implementarea rutinelor de comunicare și preluare de date, implementarea unui protocol de comunicație în aplicație etc.

Din punct de vedere al necesității schimbului de date se pot distinge trei categorii:

1. **Aplicații independente, cu decizie bazata direct pe datele colectate de la senzori** – Fig. 5.8.

În această categorie se încadrează următoarele aplicații, ca exemple: detectoare, sesizoare, elemente de semnalizare sau orice alte dispozitive cu logică de comandă și control prestabilite și impuse de către producător.

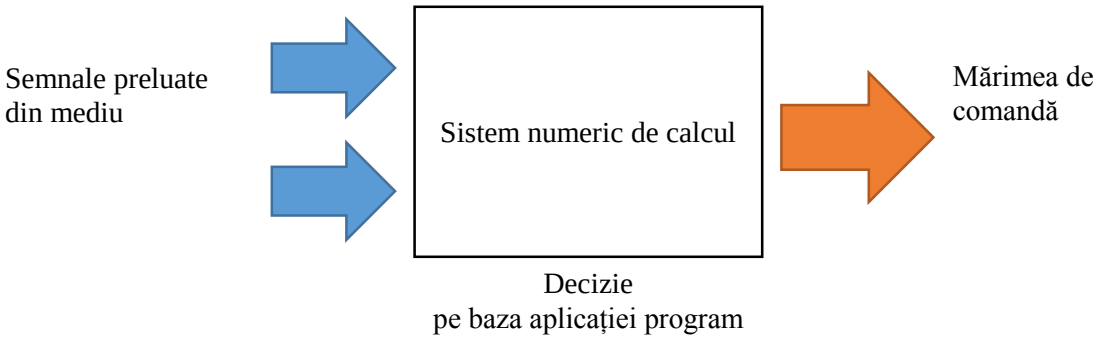


Fig. 5.8. – Aplicații independente, cu decizie proprie

2. **Aplicații dependente de elemente de interfațare** (computer gazdă, panou frontal, LCD etc) – Fig. 5.9.

Câteva exemple de aplicații din această categorie sunt: dispozitivele periferice ale computerului, panouri frontale cu afișaj și butoane, echipamente de interacțiune umană cu procesul fizic (Human Interface Device – HID).

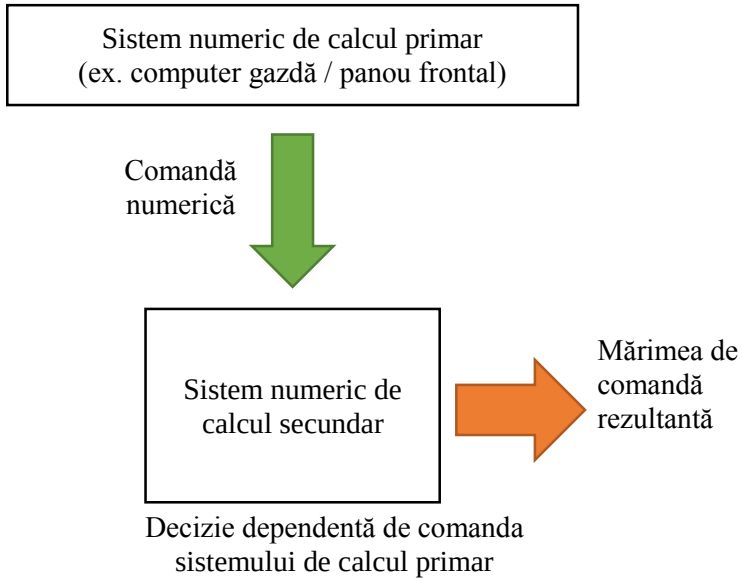


Fig. 5.9. – Aplicații dependente de un sistem numeric de calcul primar

3. Aplicații hibride (comandate de utilizator, dar cu decizie asupra algoritmului propriu)

Exemple din aceasta categorie de aplicatii - dispozitivele complexe care pot fi configurate și asistate de la distanță pentru a deservi un proces fizic: termostate digitale programabile atât de la distanță cât și prin intermediul panoului frontal propriu, ventilatoarele industriale comandate de la distanță dar care au și posibilitatea adjustării în funcție de datele preluate de la senzori, imprimante etc.

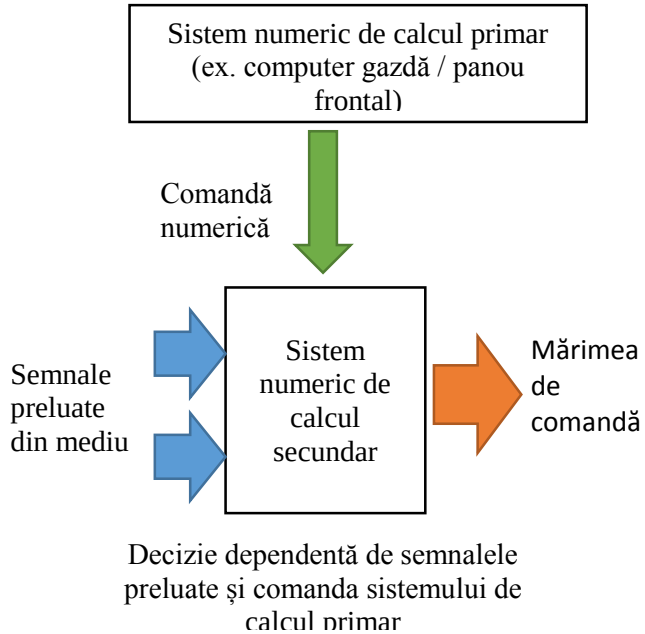


Fig. 5.10. – Aplicații hibride

Există câteva motive pentru care schimbul de date este necesar:

- ***Comanda, controlul și monitorizarea de la distanță a procesului fizic (sisteme SCADA);***
- ***Depanarea, verificarea și validarea funcționării aplicației (diagnoză și depanare în timp real – eng. real-time debugging);***

În practică, verificarea și depanarea joacă un rol important din punct de vedere al platformei de dezvoltare implicate in aplicație, deoarece este necesară urmărirea parcursului de execuție a aplicației program la nivelul memoriei platformei. De exemplu, în cazul unei aplicații independente cu decizie proprie, este necesară interogarea platformei în vederea calibrării senzorilor și a urmăririi funcționării corecte a

algoritmului de decizie. Aceste procedee de *testare, verificare, și validare* vizează atât funcționarea optimă a algoritmului la nivelul procesorului platformei, cât și funcționarea algoritmului în corelație cu datele preluate din teren (ex. *Hardware In the Loop – HIL sau Rapid Control Prototyping – RCP*). Pentru a putea realiza schimbul de date, există două metode folosite în mod frecvent:

A. Comunicația de tip serial (preponderent UART point to point) – este o metodă clasică și cel mai intens folosită. De asemenea este considerată o metoda de schimb de date invazivă, care, depinde de parcursul execuției programului (este necesară introducerea rutinelor de comunicație în codul programului, și este necesară de asemenea, implementarea protocolului în unele situații);

B. jTAG – reprezintă un standard industrial (*Joint Test Action Group*) și este considerata o metodă de schimb de date non – invazivă (deoarece preluarea datelor din program se realizează independent de execuția programului, iar datele preluate sunt vehiculate pe o magistrală independentă de cea principală a arhitecturii sistemului de calcul, astfel, schimbul de date poate fi realizat în timp real). Este important de precizat faptul că, vehicularea datelor prin interfața jTAG se realizează în paralel cu execuția programului, spre deosebire de comunicația serial (UART), la care operațiile necesare vehiculării datelor spre magistrală trebuie să se realizeze în timpul execuției programului.

A. Aplicații folosind comunicația serială

Următoarele caracteristici de bază ale acestei metode de comunicare de date sunt definitorii:

- Se utilizează în cazul în care este nevoie să se citească sau să se înscrie o valoare a unui parametru (variabilă) declarat în codul programului;
- Informația este transmisă bit cu bit pe unități de câte opt biți (byte);
- Aplicația program trebuie concepută în așa fel încât informația serializată să poată fi utilizată în conținutul programului (preluarea, reasamblarea și parcarea informațiilor în variabile);
- Aplicația program trebuie să-și reia ciclul de lucru pentru a-și actualiza informația vehiculată de la interfața seriala (reactualizarea variabilei printr-o buclă repetitivă);
- Informația transmisă prin protocolul și interfața serial afectează modul de funcționare al aplicației. modul de execuție al aplicației afectează calitatea informațiilor vehiculate (ex. executarea comunicației serial la un timp de eșantionare redus);
- Este un mod simplu de implementat și conferă compatibilitate cu protocoalele moderne care au la bază protocolul serial (ex. I^2C, SPI, Ethernet, Bluetooth etc);
- Viteza transferului de date dintre echipamentul utilizat în aplicație și calculatorul gazdă este limitată de timpul de execuție a programului;

- Nu poate fi vehiculat un volum mare de date prin intermediul interfeței seriale de tip UART deoarece dimeniunea pachetului de date se limitează la un octet (adică un byte sau 8 biți) iar informația vehiculată trebuie fragmentată și re-asamblată cât mai rapid;

În această categorie, se detaliază o aplicație specifică, ușor de implementat pe orice platformă din familia Arduino sau, generalizând, cu folosirea limbajului Wiring C - **Variația factorului de umplere.** Următoarea aplicație urmărește varierea intensității luminoase, prin intermediul comenzilor serial, care acționează asupra variabilei de control a lățimii impulsului (vezi principiul semnalelor modulate în lățime). Codul sursă în limbaj Wirinc C este prezentat mai jos.

```
     const unsigned int pin_dioda = 9;
     unsigned int valoare_primita = 0;
     unsigned int factor_de_umplere = 0;
     void setup() {
       pinMode(pin_dioda, OUTPUT);
       Serial.begin(9600);
Serial.println("Introduceti factorul de umplere 0 - 255: ");}
     void loop() {
       if(Serial.available()){
         valoare_primita = Serial.parseInt();
         if(valoare_primita < 0){
      Serial.println("Valoarea introdusa nu este in intervalul   0 - 255!");
           valoare_primita = 0;   }
         if(valoare_primita > 255){
           Serial.println("Valoarea introdusa nu este in intervalul 0 - 255!");
           valoare_primita = 0;
         }
         Serial.println("Valoarea factorului de umplere introdusa este: ");
         Serial.println(valoare_primita);  }
      analogWrite(pin_dioda, valoare_primita);
     }
```

Algoritmul implementat în **limbaj Wiring C** reprezinta de fapt o **aplicație dependentă de un sistem de calcul**, pentru că datele de comanda trebuiesc introduse de la un sistem de calcul gazda sau orice alt dispozitiv portabil. Prin executia acestui cod, se poate genera un tren de impulsuri cu lățime variabilă, în funcție de comanda primită de la sistemul de calcul. În domeniul electronicii de putere, al automatizărilor și al controlului digital, cu ajutorul unui tren de impulsuri cu lățime controlabilă, se pot obține mărimi utile în acționarea unui element de execuție (ex. variația turației unei mașini electrice de curent continuu, variația intensității luminoase, variația curentului absorbit etc.). Schimbul de date cu aplicația se realizează prin **comunicație seriala** clasică.

Pașii de execuție ai secvenței sunt prezentați, în aces caz, detaliat. Astfel:

- *În primul pas de execuție a programului sunt declarate constanta numărului de ordine al pinului de ieșire digitală și variabilele de lucru („valoare_primita” și „factor_de_umplere”);*

- *Ulterior, se stabilește modul de lucru al pinului digital atribuit constantei cu numele „pin_dioda” și se inițializează rutina de comunicație seriala prin funcția „Serial.begin()”. Dacă toate operațiile de inițializare au decurs în mod corect, se va transmite un mesaj prin interfața de comunicație serial cu ajutorul funcției „Serial.println()”. Mesajele de inițializare, apar atunci când se apasă butonul RESET;*

- *În al treilea pas, se va executa conținutul buclei „void loop ()”. În conținutul acestei bucle, se regăsește un algoritm condițional, prin intermediul căruia, se stabilesc criteriile de funcționare a aplicației. Adică, rutina pentru generarea trenului de impulsuri cu lățime variabilă este activă doar dacă valoarea factorului de umplere a fost introdusă corect (în intervalul 0 – 255).*

Ansamblul experimental al aplicației este prezentat în Fig. 5.11., iar în Fig. 5.12. și Fig.5.13. se vizualizează comanda factorului de umplere prin monitorul serial specific mediului Arduino IDE, de unde se introduc valorile dorite, respectiv oscilografia obținută la monitorizarea pe un osciloscop.

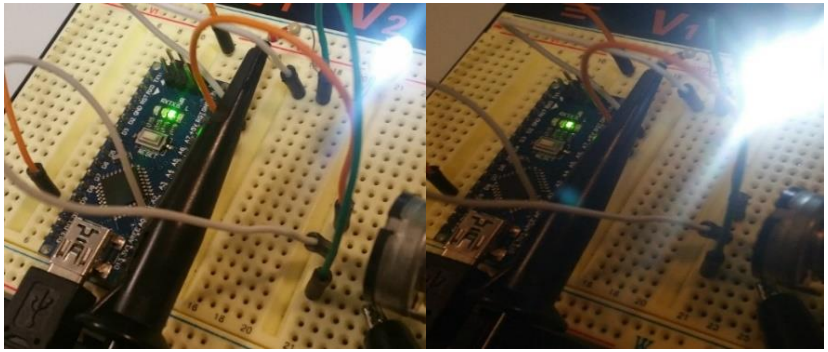


Fig. 5.11. – Ansamblul experimental al aplicației pentru variația fluxului luminos al unui LED

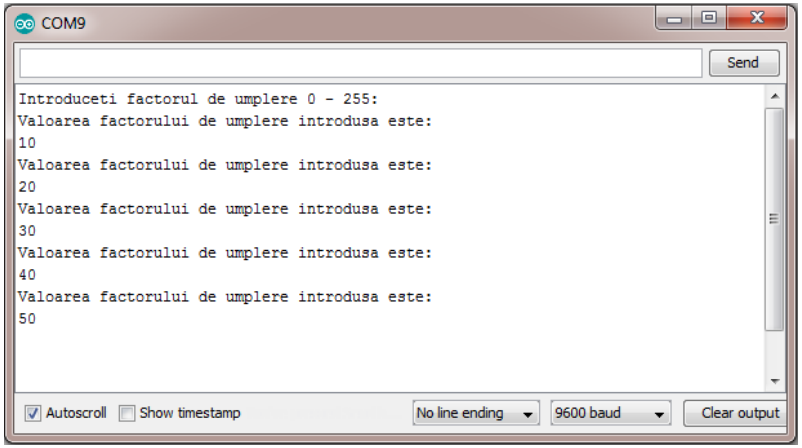


Fig.5.12.– Schimbul de date bidirecțional, prin monitorul serial Arduino IDE

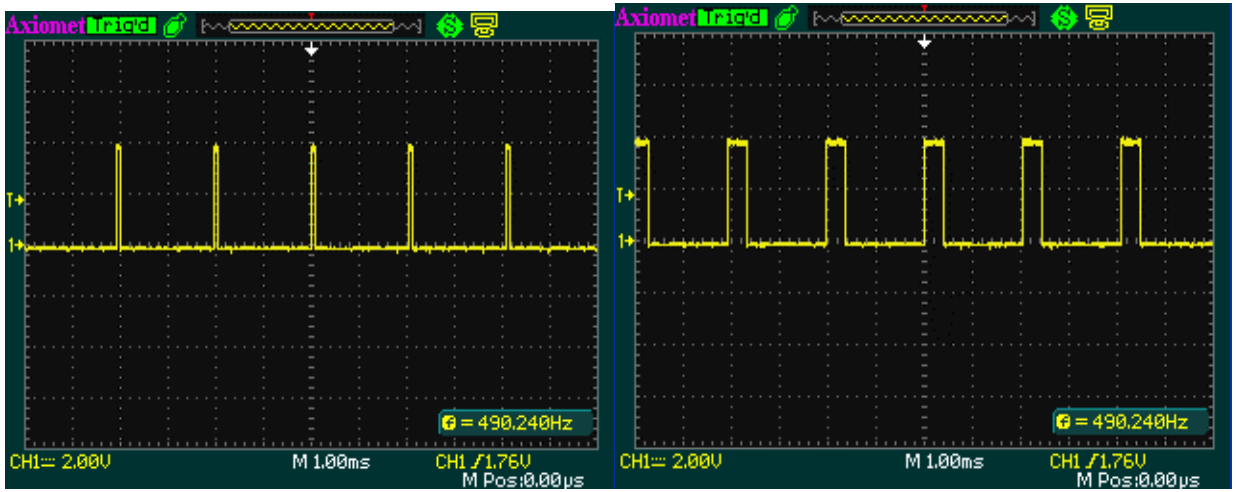


Fig.5.13. - Modificarea factorului de umplere – oscilografie

Funcționarea acestei aplicații se bazează pe schimbul de date bidirecțional prin intermediul interfeței seriale. Preluarea datelor vehiculate prin interfața seriala, se realizează cu ajutorul funcției `Serial.parseInt(),` prin care se implementează următoarele operații:

- *Preluarea / transmiterea informației bit cu bit;*
- *Stocarea informației într-un vector cu opt poziții de tip bit (adică formarea unui octet – byte);*
- *Concatenarea a doi vectori în care au fost stocate datele preluate din serial;*

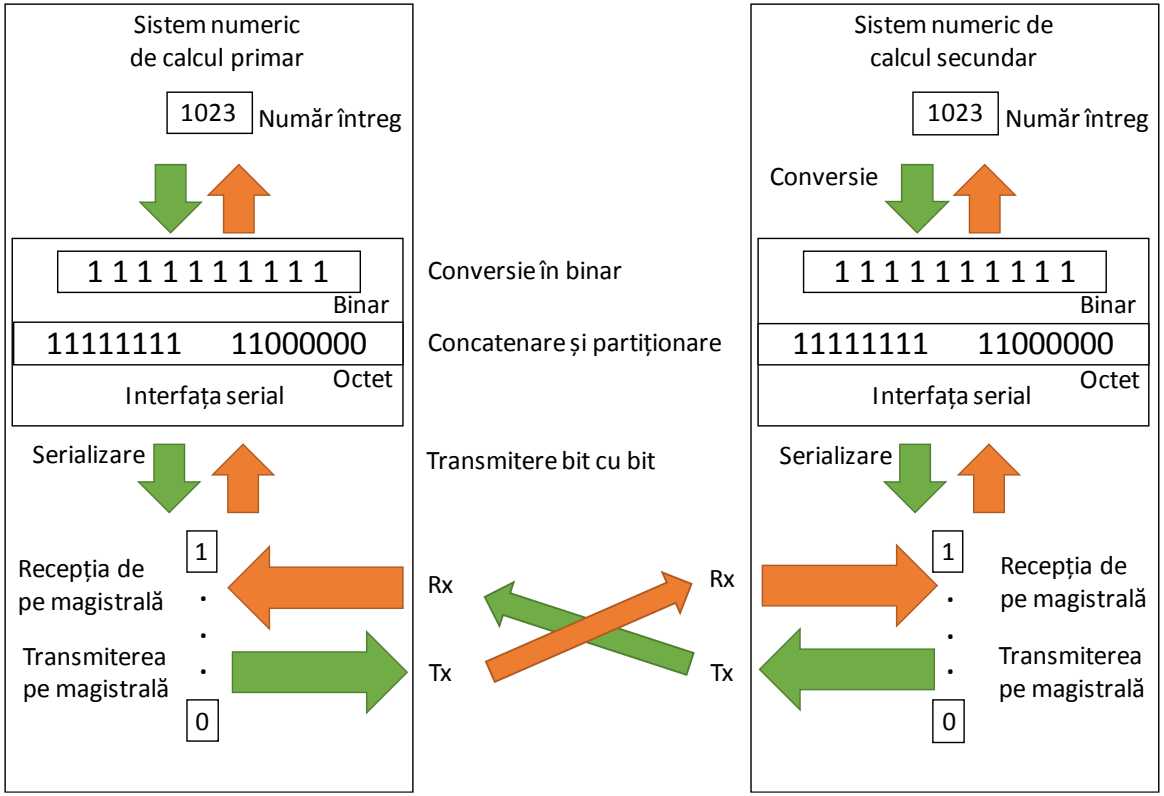


Fig. 5.14. – Diagrama de exemplificare a principiului comunicației seriale asincrone UART

Comunicația serială se realizează [15], fizic, prin cele două fire specifice „Tx” (eng. ‚Transmit data’) și „Rx” (eng. ‚Receive data’). Implementarea interfeței seriale se realizează, de obicei, la nivel fizic pe bază de porți logice.

În cazul platformelor de dezvoltare din familia Arduino, există un circuit integrat *convertor USB – TTL specializat* (eng. Transistor to Transistor Logic), care realizează transferul de date de la interfața serială a platformei înspre computerul gazdă. În sistemele de operare bazate pe nucleu / kernel DOS (ex. Windows), porturile de interfațare seriale se notează cu „COMn” (unde „n” reprezintă numărul de ordine al portului). În sistemele de operare bazate pe nucleul Unix / Linux, porturile de comunicare se notează cu „TTyn”. Tot în cazul platformei Arduino, programarea se realizează pe baza comunicației seriale, explicate în graficul din Fig. 5.14.

B. Aplicație - schimb de date în timp real

Schimbul de date in timp real **(**eng. **Real Time Data eXchange – RTDX**) [16] reprezintă un protocol (specific Texas Instruments) și vizează cu precădere sistemele de calcul pe bază de procesor digital de semnal DSP. Necesitatea de a transfera un volum mai mare de date la viteze mult mai mari decât în cazul comunicației seriale a condus la dezvoltarea acestui protocol. Important de menționat este și faptul că, prin utilizarea acestei metode de schimb de date, nu este afectată execuția aplicației deoarece *nu trebuie impuse constrângeri de execuție la nivelul aplicației care rulează pe sistemul de calcul secundar* (de exemplu pe DSP). În cazul comunicației seriale, nu se poate afirma acest lucru, deoarece, introducerea unui anumit punct de întârziere în execuția buclei de program, afectează procesul de comunicație serial [17], [18].

Schimbul de date în timp real se implementează prin **intermediul interfeței / standard de diagnoză și depanare de tip jTAG (Joint Test Action Group),** implementată la nivelul capsulei circuitului integrat al procesorului. Rolul de bază al interfeței jTAG este de a diagnostica funcționalitatea fiecărui etaj în parte ale circuitelor implementate. Practic, în procesul de fabricație al circuitelor integrate moderne, se are în vedere implementarea unei arhitecturi suplimentare de conductoare și dispozitive de control, cu scopul diagnosticării, verificării și testării funcționalității fiecărui etaj în parte. Prin transmiterea unor anumite impulsuri prin intermediul arhitecturii de conductoare, se va putea obține un semnal de răspuns de stare (eng. Status signal) de la fiecare etaj în parte. Mai mult decât atât, orice componentă electronică la nivel de chip central (ex. tehnologia Sistem On a Chip – SOC), este conectată de magistrala de conductoare pentru diagnosticare (inclusiv regiștrii fizici ai tuturor dispozitivelor periferice (ex. ADC, ePWM, GPIO etc.), magistrala de date și adresele de memorie). Astfel că, printr-un procedeu de introducere a unor semnale de test în arhitectura de conductoare distribuite, și preluarea răspunsului de la fiecare componentă în parte se poate realiza diagnosticarea problemelor (debugging) sau validarea funcționalității (logic validation). Acest procedeu poartă numele de *„Boundary Scan” [19]*.

Orice tip de interfață jTAG introduce patru terminale principale și unul opțional pentru accesul la magistrala de comunicație dintre dispozitive:

- TDI (Test Data Input) – are rolul de a introduce datele de intrare în vederea testării;
- TDO (Test Data Output) – are rolul de a prelua datele în vederea testării;
- TCK. (eng. Test ClocK) – are rolul de a furniza un tact de ceas în vederea testării;
- TMS. (Test – Mode Select) – are rolul de a plasa dispozitivul în modul test - service;
- TRST (Test Re – SeT - opțional) – are rolul de a reinițializa starea circuitului testat;

Magistrala interfeței jTAG este independentă de magistrala de date a sistemului de calcul. Interogarea dispozitivelor aflate pe magistrala jTAG se poate realiza independent de execuția aplicației program (la nivelul procesorului), fără a fi nevoie ca execuția să fie suspendată.

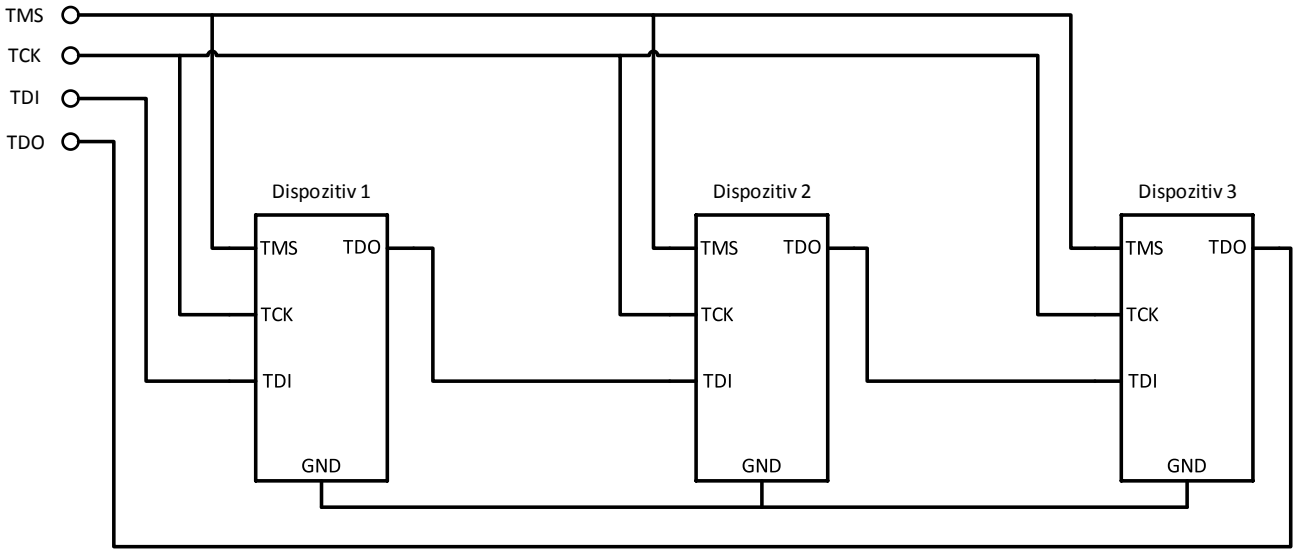


Fig. 5.15. Exemplu de aplicație de comunicare a dispozitivelor pe magistrala jTAG

Starea procesorului poate fi controlată cu ajutorul comenzilor specifice (ex. HALT, RUN, STEP, PAUSE etc.), prin intermediul interfeței jTAG. Astfel, prin utilizarea interfeței jTAG se va putea realiza analiza în detaliu a modului de execuție la nivel de procesor. De asemenea, accesarea datelor din memorie și a regiștrilor componentelor periferice permite controlul în timp real a întregului proces.

Din acest motiv, protocoalele de comunicație implementate la nivelul interfeței jTAG permit realizarea unui schimb de date în timp cvasi real (RTDX). Implementarea acestui protocol, necesită atât un sistem numeric de calcul secundar dotat cu interfață și arhitectură jTAG, cât și un dispozitiv adaptor – emulator jTAG la USB pentru calculatorul gazdă (Fig.5.16.). Dispozitivul adaptor-emulator este dotat cu un sistem similar platformei de lucru (ex. DSP), în care a fost încărcată în prealabil o aplicație specifică protocolului jTAG (ex. un sistem de operare în timp real - DSP / BIOS). Aplicația încărcată în memoria dispozitivului jTAG mai poartă și numele de „Debug Server”.

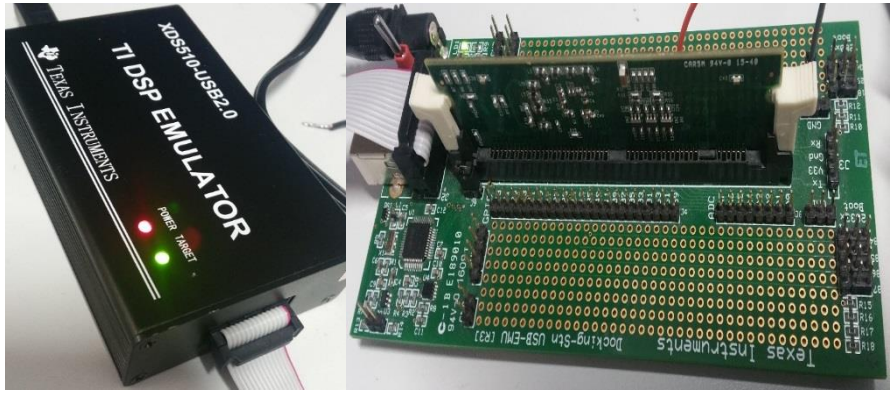


Fig. 5.16. Interfața – emulator jTAG – USB (stânga) și procesor digital de semnal DSP (dreapta)

Pentru a realiza toate operațiile specificate (manipularea datelor din regiștrii, citirea și accesarea datelor direct din memorie, programare etc.) este nevoie de un mediu de lucru special conceput pentru a permite executarea acestor operații. Un exemplu de mediu de programare și testare este *Code Composer Studio,* de la Texas Instruments, care permite gestionarea datelor la nivelul unui procesor de semnal. De asemenea, tot prin intermediul acestui mediu de lucru, se poate deschide sau închide canalul de comunicație pentru schimb de date în timp real.

În general, utilizarea canalelor de comunicație în timp real prin intermediul interfaței jTAG se aplică atunci când este necesară interacțiunea în timp real dintre un mediu de simulare, programare și testare si unul sau mai mulți parametrii din aplicația program inscripționată în memoria sistemului de calcul secundar (DSP). Exemplul 5.2.(A) poate fi considerat o aplicație importantă în contextul controlului digital pentru o mărime

de comandă, generată numeric din calculator, de către un utilizator. Aceeași aplicație va fi implementată diferit, în exemplul următor, ținând cont de reperele teoretice prezentate.

Aplicație - Variația factorului de umplere, folosind platforma de lucru TI - F28335 (Matlab Simulink):

MOD DE LUCRU:

- Pe baza paletei de intrumente „Embedded Coder Support Package for Texas Instruments C2000 MCU and DSP" se va concepe un model Matlab Simulink compus din două blocuri. Blocul de recepționare a datelor vehiculate pe magistrala de comunicație jTAG – „From RTDX ichan1" și blocul „ePWM" (eng. Enhanced Pulse Width Modulation) – pe baza căruia se generează trenul de impulsuri cu lățime variabilă;

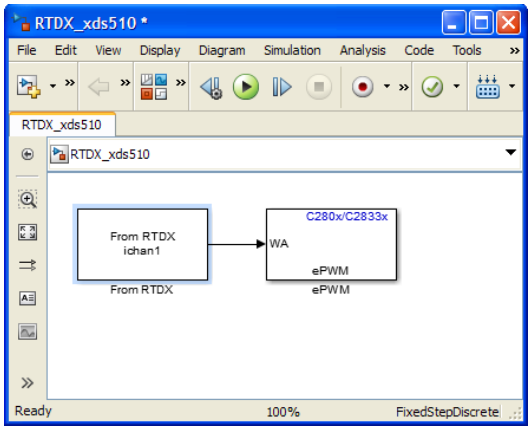


Fig. 5.17. Dezvoltarea aplicației program pentru platforma DSP F28335

- Se continuă cu parametrizarea numelui canalului de comunicație, a modului de generare al semnalului modulat în lățime (ex. blocul ePWM), a timpului de eșantionare pentru recepția de date,a mediului de programare, platformei DSP etc.
- În urma parametrizărilor menționate, se va încărca aplicația program în memoria platformei de lucru cu DSP F2833, prin apăsarea butonului „BUILD";

- Mediul de programare Code Composer Studiu 3.3 se va deschide în mod automat și va executa comenzile necesare pentru stabilirea conexiunii prin intermediul canalului „ichan1”;

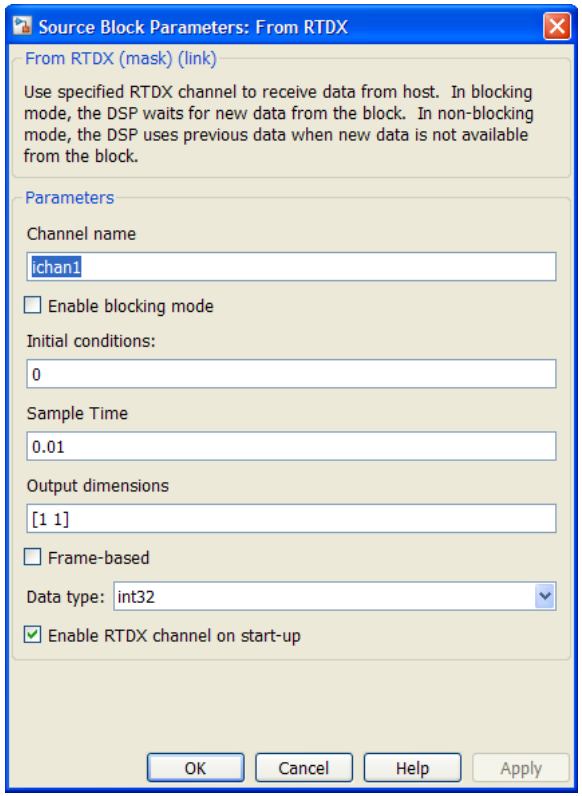
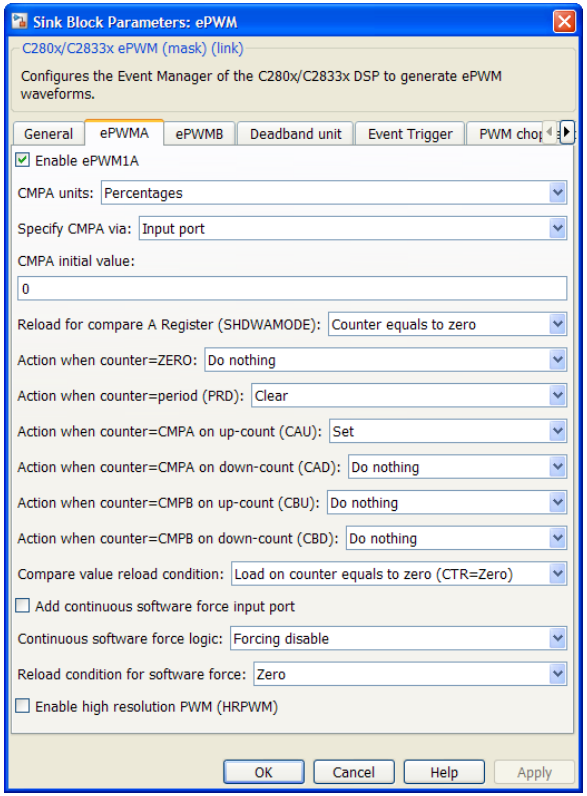


Fig. 5.18. Parametrizarea canalului de comunicație RTDX Fig.



5.19. Parametrizarea blocului de generare al trenului de impulsuri modulate în lățime

- Odată încărcată în memorie, aplicația își va începe execuția și va aștepta un răspuns pe canalul „ichan1” deschis prin intermediul interfeței jTAG. Canalul de comunicație permite accesul direct la o adresa din memoria platformei D.S.P., mai precis adresa la care este stocată informația despre factorul de umplere al semnalului modulat (duty cycle);

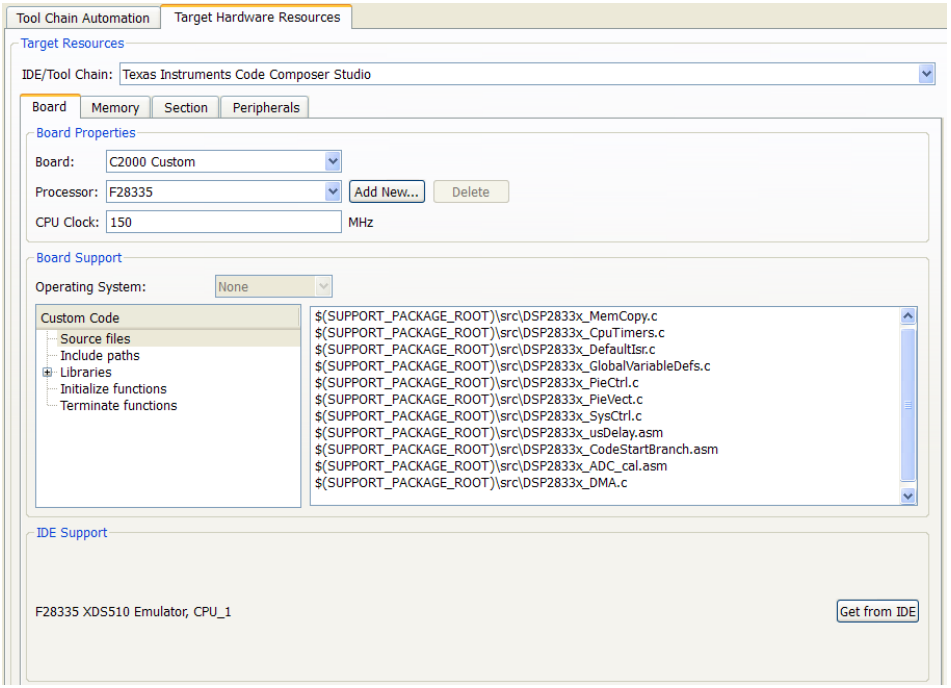


Fig. 5.20. Selectarea platformei de lucru și mediului de programare

- Pentru a putea controla prin intermediul calculatorului gazdă în timp real factorul de umplere al semnalului modulat în lățime trebuie conceput un al doilea model simulink, aplicația program care rulează pe calculatorul gazdă, și comunică cu aplicația din memoria platformei;

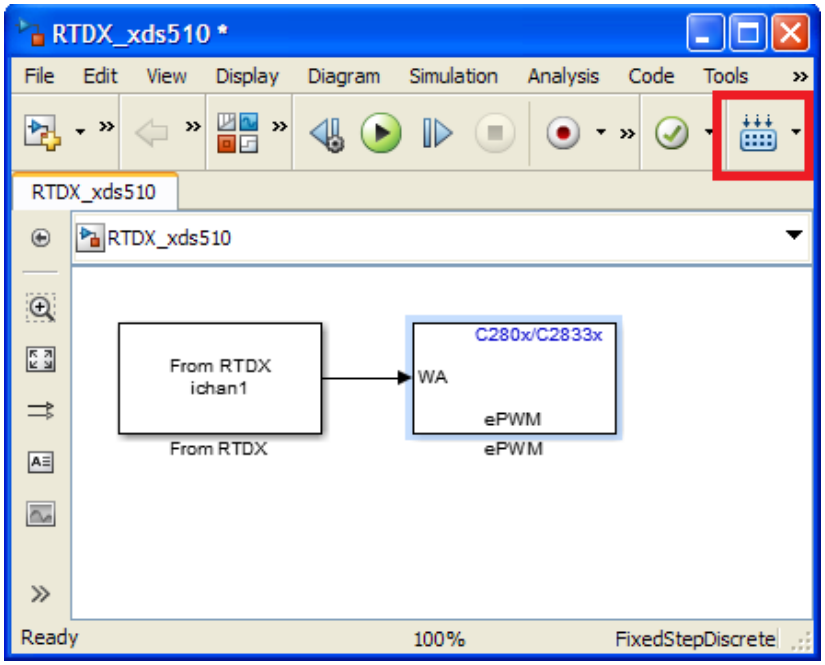


Fig. 5.21. Încărcarea aplicației – program în memoria platformei de lucru F28335 – BUILD

Odată încărcată și executată aplicația binară (cu extensia „.out") în memoria procesorului digital de semnal, se va deschide un canal de comunicație în timp real „ichan1". Prin intermediul acestui canal se vor putea transfera date în timp real între procesorul digital de semnal și calculatorul gazdă la o viteză de transfer de ordinul 2 - 10 MB / s prin intermediul unei interfețe USB – jTAG. Parametrii canalului de comunicație pot fi adjustați din mediul de programare Texas Instruments Code Composer Studio v 3.3.

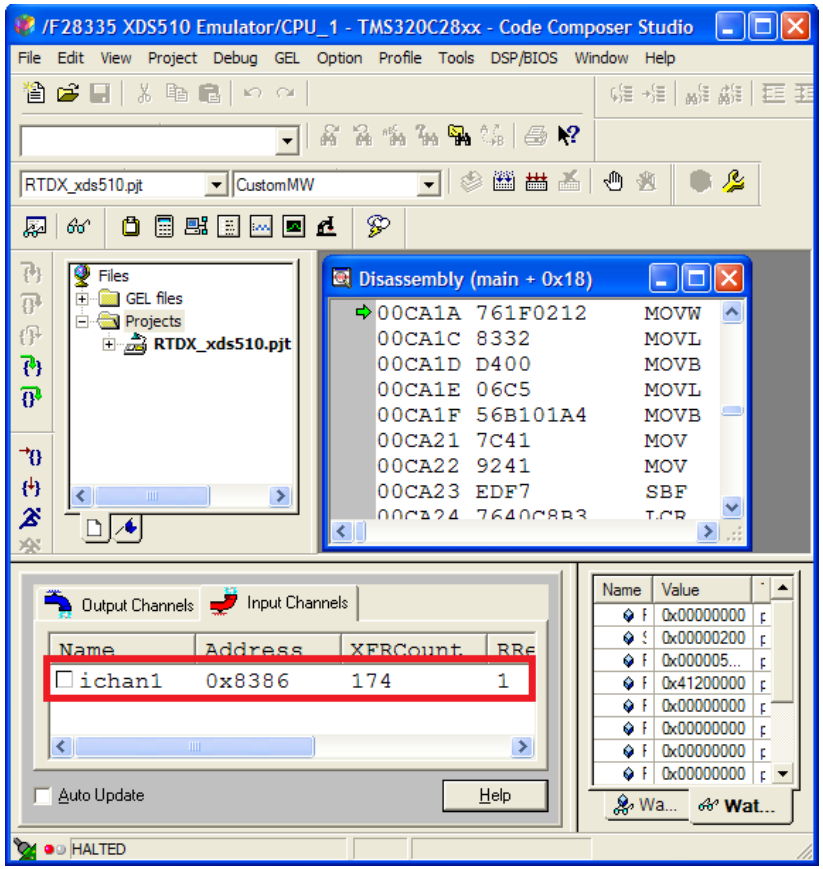


Fig. 5.22. Mediul de dezvoltare și programare Code Composer Studio 3.3 – accesarea canalului pentru schimb de date în timp real „ichan1”

- Pentru a iniţializa comunicaţia dintre calculatorul gazdă şi procesorul digital de semnal, cu ajutorul comenzii *„rtdxsimlib*” introduse în consola de comandă Matlab se va deschide o nouă paletă de instrumente. Aceasta va pune la dispoziţie blocurile de legătură dintre aplicaţia ce rulează pe platforma DSP şi computerul gazdă (blocurile „RTDX HOST”);

- Pe baza acestor blocuri, se va contrui un model de inscripţionare a valorilor factorului de umplere pe magistrala jTAG („RTDX Write” înspre canalul „ichan1”). Pentru a permite variaţia factorului de umplere se va introduce un bloc cu factor de amplificare reglabil (având limitele 0 - 100) şi o constantă cu valoare unitară şi tip de date întreg cu reprezentare pe 32 de biţi („int32”). Factorul de amplificare se stabileşte iniţial la zero.

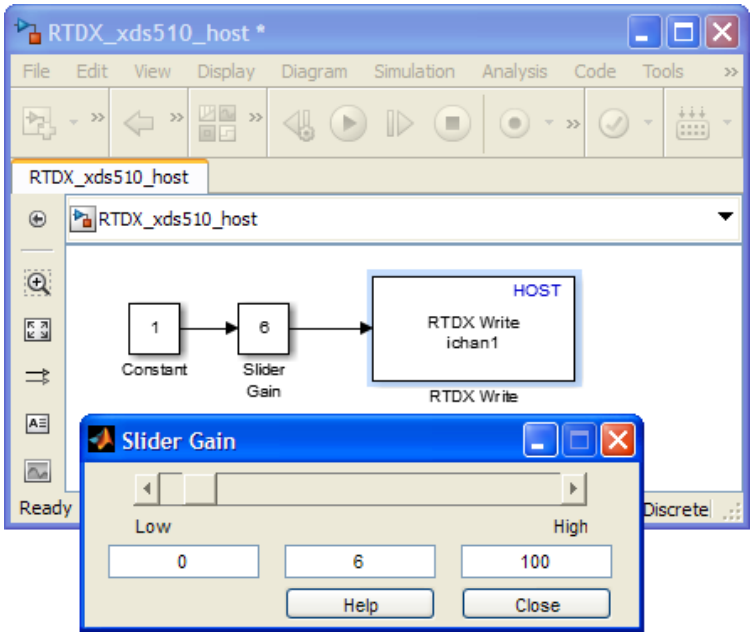


Fig. 5.23 – Conceperea modelului de aplicaţie - program pentru computerul gazdă

- urmează parametrizarea blocului de comunicaţie prin stabilirea numelui canalului;

- în meniul de configurare a parametrilor de simulare, odată cu introducerea blocurilor pentru comunicarea cu calculatorul gazdă apare opțiunea „Host-Target Communication”. În această categorie se vor furniza informațiile necesare pentru a creea legătura între aplicația ce rulează pe platforma de lucru, și aplicația ce rulează pe computerul gazdă. Mai precis, va trebui specificată calea înspre fișierul de proiect „.pjt” (generat de către mediul Code Composer 3.3) și către fișierul executabil „.out”;

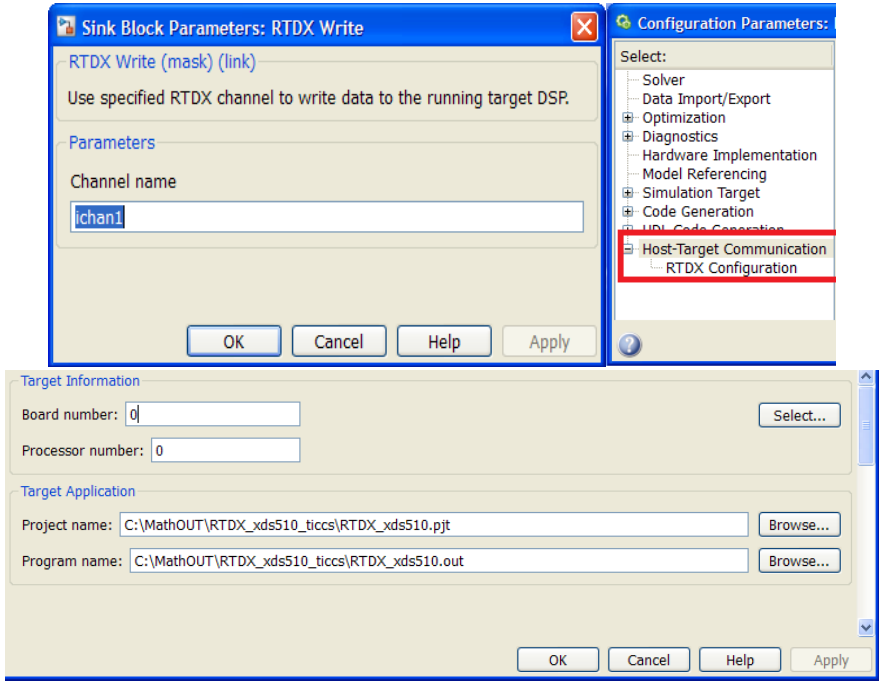


Fig. 5.24 Parametrizarea canalului de comunicație și specificarea căilor de acces înspre fișierele aferente aplicației inscripționate în memoria platformei DSP

- Pentru a pune în funcțiune aplicația propriu–zisă (gazdă–platformă), se trece la pornirea procesului de simulare în modelul Simulink de aplicație gazdă prin apăsarea butonului verde din bara de instrumente (Play);

- Mediul Code Composer Studio 3.3 va încărca fișierul executabil actualizat în memoria platformei, iar la scurt timp, se va stabili legătura între computerul gazdă și platforma DSP;

- Prin acționarea asupra cursorului blocului cu factor de amplificare variabil (eng. Slider Gain), se va putea observa cu ajutorul unui osciloscop și al unei diode electro – luminiscente, variația factorului de umplere (fluxului luminos în cazul diodei electro – luminiscente);

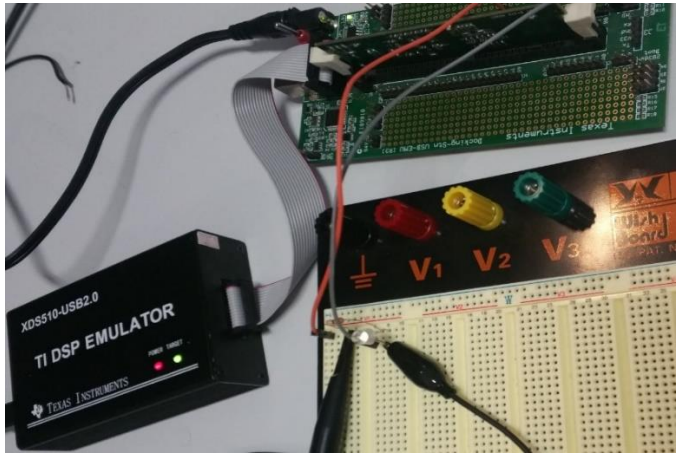


Fig. 5.25. Montaj experimental

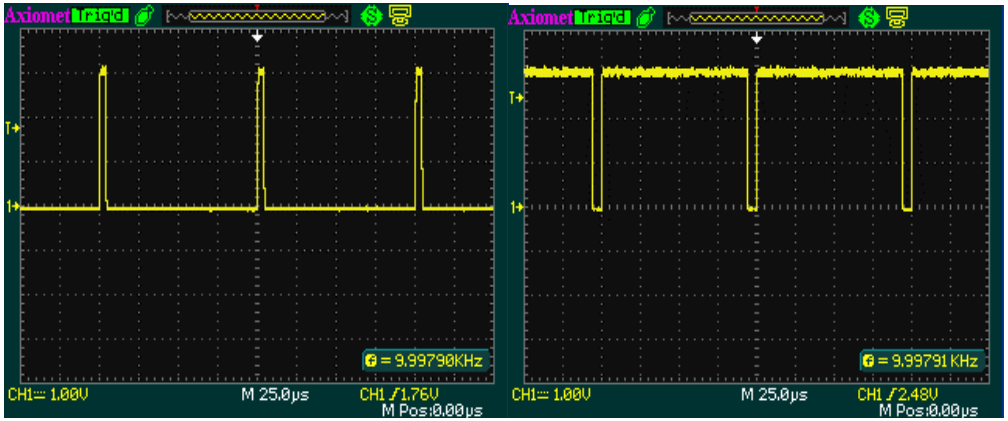


Fig. 5.26. Trenul de impulsuri cu lățime variabilă (oscilografie)

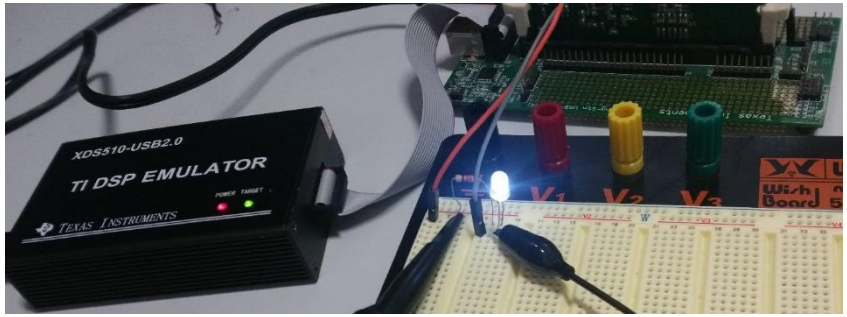


Fig. 5.27. Variația fluxului luminos al unei diode electro – luminiscentă (control digital)

MOD DE FUNCȚIONARE:

- Metoda schimbului de date în timp real prin intermediul interfeței jTAG, poate fi considerată non–invazivă, deoarece nu afectează modul de execuție al aplicației de pe platforma de lucru, execuția aplicației de pe platforma de lucru nu aduce impedimente procesului de comunicație dintre platforma DSP și computerul gazdă.

- Avantajul major față de comunicația serială, constă în faptul că interfața jTAG are acces în mod direct la adresele de memorie, fără ca să fie necesară stocarea unei variabile în memorie, serializarea informației partiționarea în unități mai mici sau concatenarea la primire. Astfel, prin interfața jTAG se pot obține viteze mai mari de răspuns și transfer, iar volumul de date vehiculat poate fi mult mai mare decât în cazul comunicației serial.

- Schimbul de date dintre DSP și emulatorul jTAG se realizează în timp real, deoarece se evită repetarea operațiilor clasice de fragmentare a informației în cadre de date și de preluare a datelor din memorie iar apoi înaintarea acestora înspre zonele tampon (buffer) ale interfației serial sub formă fragmentată. În cazul canalului de comunicație jTAG, informația poate fi transmisă și sub formă brută (nefragmentată) într-un flux continuu (eng. burst mode sau continuous flow). De asemenea, se precizează faptul că, interfața jTAG este conectată în mod fizic la toate componentele din cadrul arhitecturii unui sistem de calcul cu procesor digital de semnal, deoarece scopul primordial al acestei interfețe este diagnoza și depanarea sistemului. Astfel, interfața jTAG are căile sale fizice dedicate de comunicare cu perifericele! Astfel, se evită transferul datelor înspre unitatea jTAG prin intermediul magistralei principală de date sau a memoriei, care la rândul ei, poate fi sau nu disponibilă. Fapt pentru care, ar putea fi întârziat în mod considerabil procesul de comunicație. În cazul interfeței asincrone serial, datele obținute de la perifericele procesorului digital de semnal, sunt stocate în zonele de memorie prin intermediul variabilelor (adică în anumite zone de memorie). Pentru a face acest lucru, perifericele utilizează magistrala comună de date a arhitecturii procesorului digital de semnal.

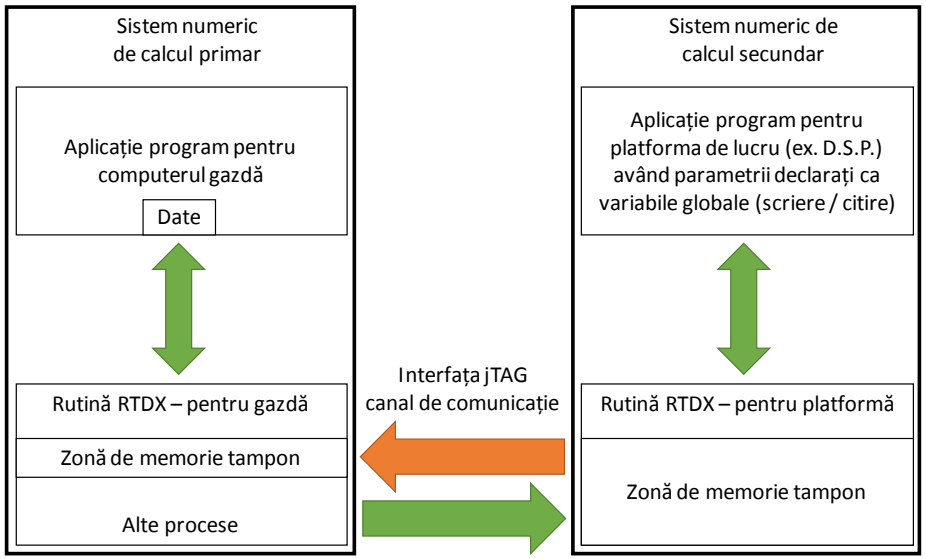


Fig. 5.28. Principiul de funcționare al protocolului Real Time Data eXchange

Datele vehiculate sunt fragmentate în pachete de 8 biți care sunt stocate temporar într-o zonă de memorie tampon, deoarece se va aștepta un moment de disponibilitate al magistralei serial. La momentul în care magistrala este disponibilă pentru transfer, fiecare cadru de date, va fi înaintat înspre magistrală. De asemenea, interfața jTAG, totuși utilizează o zonă de memorie tampon, doar că, pentru a stoca în această zonă, nu se utilizează magistrala de date a sistemului ci se utilizează căile dedicate al interfeței jTAG. Viteza de transfer înspre computerul gazdă depinde de performanța emulatorului jTAG și nivelul de implementare al interfeței jTAG în cadrul procesorului digital de semnal (ex. există emulatoare care dispun de un port Ethernet prin intermediul căruia se poate accesa interfața jTAG, sau sunt emulatoare care dispun de procesor propriu pentru interfața jTAG, în alte situații procesoarele digitale de semnal dispun de un sistem de operare în timp real DSP / BIOS etc.);

Ghidul de aplicații propus realizează introducerea în modul de folosire a sistemelor embedded cu microcontrolere, cu aplicații în ingineria electrică. Pașii de inițiere și aprofundare a cunoștintelor sunt prezentați gradual, cu exemple simple, folosind elemente de procesare din familia Arduino, atât de populară și accesibilă în ultima perioadă. Astfel, rezumând structura propusă, s-au discutat aspecte legate de:

- ✓ Introducerea noțiunii de sistem embedded în ingineria electrică, cu clasificări ale dispozitivelor și cu concentrare pe câteva din cele mai utile și utilizate platforme:
- ✓ Modul de programare și uneltele software disponibile pentru manipularea sistemelor embedded;
- ✓ Realizarea aplicațiilor de interfațare cu elemente digitale, respectiv analogice, încercându-se o generalizare utilă pentru a oferi utilizatorilor perspective de dezvoltare a subiectului;
- ✓ Modalități de comunicare a aplicațiilor cu sisteme embedded, oferindu-se o suită de aplicații mai complexe.

Cum domeniul aplicațiilor cu sisteme embedded în inginerie electrică este în continuă expansiune, cu un ritm greu de urmărit, lucrarea de față reprezintă încercarea autorilor de a oferi studenților în domeniu un ghid sistematizat, generalizat de realizare a aplicațiilor cu sisteme embedded, dar și un imbold în direcția realizării de proiecte mai ample, în acord cu tendințele tehnologice actuale: conectivitate universală, comunicare globală între dispozitive și între utilizatori, cu stocarea datelor în 'cloud', spre îmbunătățirea calității vieții și a cresterii eficienței proceselor industriale.

Bibliografie selectivă

[1] https://robu.in/product/atmega328p-pu-pdip-28-microcontroller/

[2] https://potentiallabs.com/cart/attiny-85-online-india-hyderabad

[3] https://learn.sparkfun.com/tutorials/tiny-avr-programmer-hookup-guide/attiny85-use-hints

[4] https://medium.com/jungletronics/attiny85-easy-flashing-through-arduino-b5f896c48189

[5] V.Cvjetkovic, M. Matijevic, „Overview of architectures with Arduino boards as building blocks for data acquisition and control systems", 13th International Conference on Remote Engineering and Virtual Instrumentation (REV), 2016.

[6] https://www.circuito.io/blog/arduino-uno-pinout/

[7] P. Cocchi, „Analysing and Experimenting the Intel Galileo Board for the Internet-Of-Things", Technical Report, n.12, 2015, Sapienza Universita di Roma, ISSN 2281 – 4299;

[8].http://yehnan.blogspot.com/2012/03/compile-and-upload-arduino-sketches.html

[9]. M.C. Ramon, "Intel Galileo and Intel Galileo Gen 2, API Features and Arduino Projects for Linux Programmers", Apress Open, 2014, ISBN 978-1-4302-6838-3.

[10].P. Friese, F. Ibanez, „Putting the Internet of Things forward to the Next Level" – „Internet of Things – from Research and Innovation to Market Deployment", River Publishers Series in Communications, 2014, ISBN: 9788793102941.

[11] O. Vermesan, P. Friess, „Digitising the Industry Internet of Things Connecting the Physical, Digital and Virtual Worlds, River Publishers Series in Communications, 2016, ISBN: 9788793379817, e-ISBN: 9788793379824.

[12] Purdum J., "Beginning C for Arduino", 2012, ISBN: 978-1-4302-4777-7.

[13] http://blog.ardublock.com/

[14]https://www.tinkercad.com/circuitshttps://www.curiousmotor.com/2017/09/5-effective-online-tools-that-will.html

[15] https://learn.sparkfun.com/tutorials/serial-communication/all

[16]https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwi44rfDu4PgAhUO_aQKHVP_AFUQFjAAegQICRAC&url=https%3A%2F%2Fe2echina.ti.com%2Fcfs-file%2F__key%2Ftelligent-evolution-components-attachments%2F13-106-00-00-00-00-41-26%2FReal_2D00_Time-Data-Exchange.pdf&usg=AOvVaw0-Mgd8ZUnf1el4B3z-gHRD

[17] http://www.unife.it/ing/lm.infoauto/sistemi-elaborazione/dispense/comparing_serial_interfaces_an_01_e.pdf

[18] https://www.embedded.com/debugger-performance-matters-the-importance-of-good-metrics/]

[19] https://www.rose-hulman.edu/class/ee/yoder/ece581/Labs/Lab05/spra895.pdf

[20]http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=2ahUKEwi44rfDu4PgAhUO_aQKHVP_AFUQFjACegQIBxAC&url=http%3A%2F%2Fusers.utcluj.ro%2F~arsinte%2FProcMedia%2FJTAG_Emulators.ppt&usg=AOvVaw0qDWekQUdi-o526WrO5DsS