



UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI” DIN IAȘI
FACULTATEA DE INGINERIE ELECTRICĂ, ENERGETICĂ
ȘI INFORMATICĂ APLICATĂ
PROGRAMUL DE STUDIU: INFORMATICĂ APLICATĂ ÎN INGINERIA
ELECTRICĂ



Lucrare de Licență

Coordonatori:

Șef lucr.dr.ing. Silviu Ionuț Ursache

Student:

**Simion Ștefan-
Cătălin**



UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI” DIN IAȘI
FACULTATEA DE INGINERIE ELECTRICĂ, ENERGETICĂ
ȘI INFORMATICĂ APLICATĂ
PROGRAMUL DE STUDIU: INFORMATICĂ APLICATĂ ÎN INGINERIA
ELECTRICĂ



Lucrare de Licență

Sistem de irigare automată a plantelor de apartament

Coordonatori:

Șef lucr.dr.ing. Silviu Ionuț Ursache

Student:

Simion Ștefan-Cătălin

2024

Cuprins

Introducere	1
Capitolul 1. Instrumente hardware folosite în realizarea sistemului.....	2
1.1. Senzor de umiditate a solului.....	2
1.2. Modul sursă de alimentare externă.....	4
1.3. Pompă submersibilă.....	5
1.4. Modul releu 1 canal, 3.3V.....	7
1.5. Modul buton.....	8
1.6. Modul ecran LCD, 16x2 și protocolul I2C.....	10
1.7. Senzor de nivel al apei.....	12
1.8. Modul buzzer MH-FMD.....	14
1.9. Placa de dezvoltare Raspberry Pi Pico W.....	15
1.10. Placa de dezvoltare Arduino Uno (revizia 3).....	18
1.11. Conversia analogic-digitală prin Raspberry Pi Pico W.....	20
1.12. Protocolul de comunicație UART.....	21
Capitolul 2. Instrumente software folosite în realizarea sistemului.....	24
2.1. Limbajul de programare MicroPython.....	24
2.2. Mediul de dezvoltare a aplicațiilor Thonny IDE.....	26
2.3. Limbajul de programare folosit pentru Arduino Uno.....	28
2.4. Mediul de dezvoltare a aplicațiilor Arduino IDE.....	30
2.5. Diferențe între MicroPython și adaptarea de C++ a Arduino.....	31
Capitolul 3. Modul de funcționare a întregului sistem.....	34
3.1. Diagrama întregului sistem.....	34

3.2.	Librăriile folosite în programarea sistemului.....	35
3.3.	Variabile și funcții construite pentru programarea sistemului.....	36
3.4.	Interfațarea de date pe web cu ajutorul platformei Blynk.....	45
Concluzie.....		48
Anexe.....		49
Bibliografie.....		72

Introducere

Sistem de irigare automată a plantelor de apartament

De ce am ales această temă?

Tema aleasă de mine constituie un subiect ce poate ușura viața celor ce dețin un număr semnificativ de plante, însă nu-și găsesc timpul necesar pentru a se îngriji de toate. Tratatând această temă, am căutat să găsesc o soluție autonomă pentru irigarea plantelor prin realizarea unui sistem ce achiziționează informații cu privire la umiditatea solului plantei și le transmite mai departe către un microcontroller care, pe baza unor algoritmi și automatizări, decide dacă planta trebuie sau nu udată, prin intermediul unei pompe de apă submersibilă.

Scopul lucrării

Acest proiect își propune realizarea unui sistem ce să aibă ca și sarcină principală irigarea plantelor de apartament într-un mod complet autonom, lipsindu-se de necesitatea prezenței operatorului uman. Viața poate fi foarte ocupată în unele momente, astfel că, folosind acest sistem, nu va mai fi nevoie să vă faceți griji nici măcar o clipă în ceea ce privește irigarea plantelor. Lucrarea dorește să demonstreze faptul că sistemul prezentat în capitolele ce urmează este unul capabil de achiziționarea și valorilor obținute din senzorii folosiți și să îi prelucreze în timp util pentru a își dovedi eficiența. După parcurgerea întregii lucrări, cititorul va înțelege și își va putea forma o imagine de ansamblu atât în legătura cu principiile ce stau la baza unui astfel de sistem cât și cu modul în care acestea funcționează și, eventual, va realiza că acesta prezintă o soluție care este utilă și simplă de implementat, cu costuri mult mai mici față de opțiunile existente deja pe piață. Desigur, sistemul prezentat în lucrare nu ar putea fi comparat cu sistemele disponibile pe piață, deoarece lucrarea și implementarea sa au un caracter exclusiv didactic.

Capitolul 1

Instrumente hardware folosite în realizarea sistemului

1.1. Senzor de umiditate a solului

În cadrul sistemului de irigare automată este utilizat senzorul de umiditate a solului cu rolul de a determina nivelul de umiditate din solul plantelor, datorită costului său redus, a dimensiunilor sale mici și a eficienței de care dă dovadă în îndeplinirea sarcinii sale caracteristice: achiziționarea de informații referitoare la umiditatea din solul în care este implantat.

Acesta este un senzor analogic capacitiv ce măsoară concentrația de apă din solul în care este introdus și folosește în acest proces comparatorul de precizie înaltă LM393. Senzorul dispune de două sonde metalice pe suprafața acestuia care au o conductivitate electrică între ele, în funcție de umiditatea din sol.

Senzorul este frecvent utilizat pentru astfel de aplicații și funcționează cu o alimentare în intervalul de 3.3 și 5 volți. Dispozitivul dispune atât de o ieșire digitală, cât și de o ieșire analogică. Pinii și componentele ce constituie acest modul sunt reprezentate în figura 1.1.

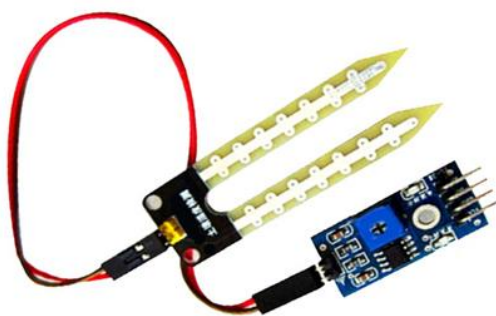


Fig. 1.1 - Modulul senzor de umiditate a solului

Senzorul este conectat alături de un potențiometru ce ajută la reglarea sensibilității ieșirii digitale (pinul D0 al senzorului). Pe plan structural, există un LED ce confirmă faptul că

dispozitivul este alimentat și un LED ce ne informează cu privire la prezența fluxului de informații de pe ieșirea digitală, după cum se poate observa și în figura 1.2.

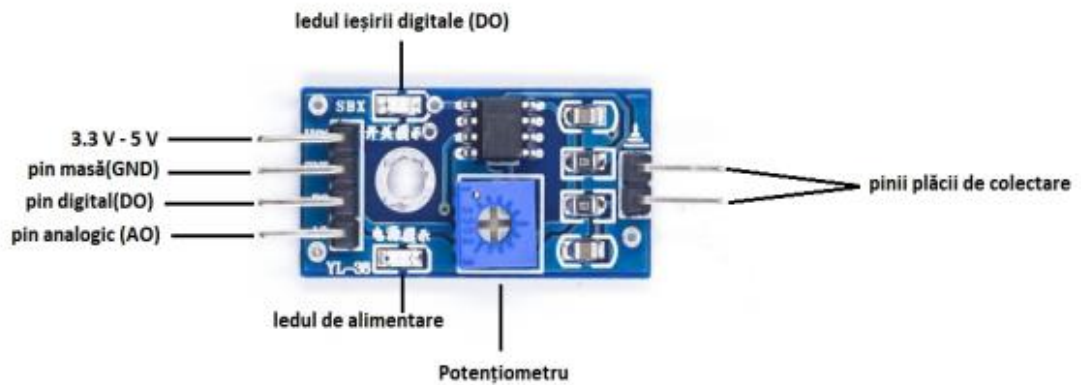


Fig. 1.2 – Modulul comparator cu potențiomtru

Schema electrică a HW-080 alături de comparatorul LM393 și microcontroler este prezentată în figura 1.3.

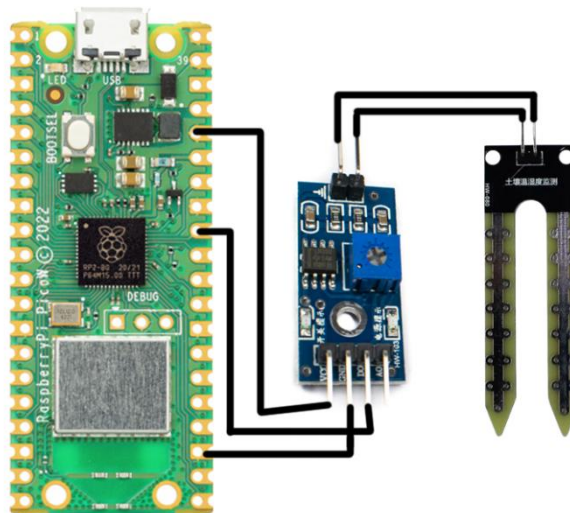


Fig. 1.3 – Schema electrică a ansamblului senzor – microcontroler Raspberry Pico W

La baza principiului de funcționare a senzorului folosit stă tensiunea pe care o emite senzorul, aceasta modificându-și valoarea în funcție de concentrația apei prezentă în solul în care HW-080 este instalat astfel:

- atunci când concentrația apei în sol este în parametrii optimi, tensiunea de la ieșirea senzorului scade;
- atunci când concentrația apei în sol este sub parametrii optimi, tensiunea de la ieșirea senzorului crește.

Senzorul este conectat pe plăcuța de dezvoltare prin patru pini, după cum se observă și în figura 1.3:

- pinul GND ce este conectat la unul din pinii GND ai plăcii;
- pinul VCC ce este conectat la pinul 36 al plăcii (ce are la ieșirea sa 3.3 volți), pinul responsabil de alimentarea dispozitivelor ce intră în componența sistemului;
- pinul AO, ce reprezintă ieșirea analogică a semnalului provenit de la senzor în funcție de umiditatea solului în care acesta este plasat, este conectat la pinul 26 ce are funcția de ADC, adică convertește semnalul analogic transmis pe acel pin în semnal digital.

1.2. Modul sursă de alimentare externă

Acest modul face posibilă funcționarea releului alături de pompă, și ajută la protecția microcontroler-ului, în sensul că nu îl expune la curenții și tensiunile ce trec prin ansamblul modul releu-pompă.

Tensiunea de intrare pentru această sursă de alimentare externă este cuprinsă între 7 și 12 volți, iar alimentarea sa se face fie direct de la priză printr-un port USB, fie dintr-o baterie de 9 volți.

Tensiunile de ieșire pe care acest dispozitiv este capabil să le ofere sunt de 3.3 și de 5 volți, tensiuni potrivite pentru alimentarea unui număr semnificativ de microcontrolere și dispozitive electronice. În componența sa intră și un buton care are ca și funcție pornirea/oprirea modulului. De asemenea, acest modul este prevăzut și cu pini cu funcție de GND.

În figura 1.4 este prezentată schema circuitului pe care acest modul o are la baza funcționării sale, iar în figura 1.5 este prezentat modulul.

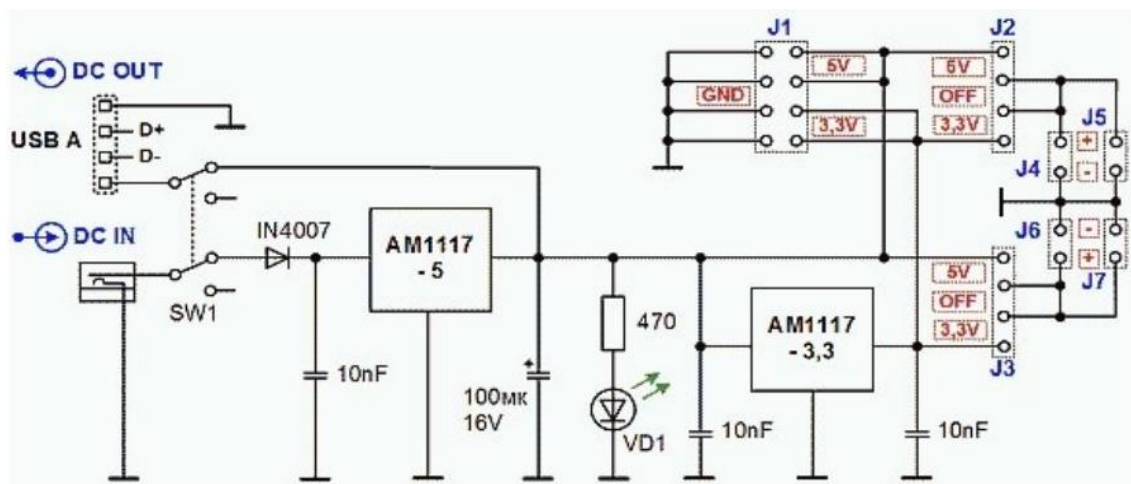


Fig. 1.4 – Schema circuitului modului sursă de alimentare externă

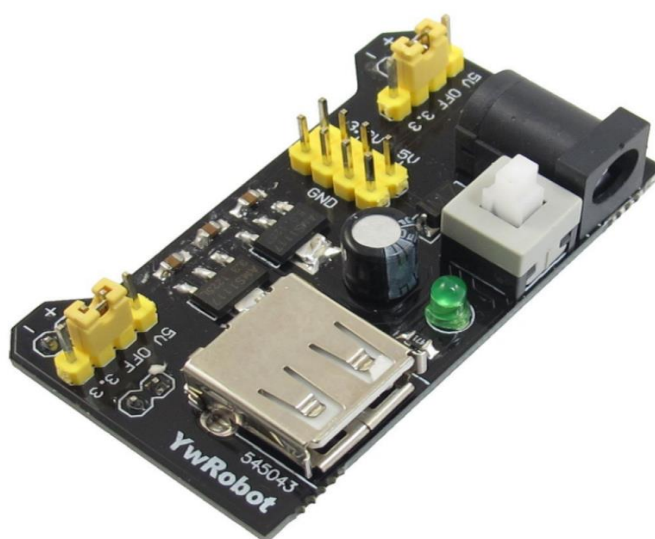


Fig. 1.5 – Modulul sursă de alimentare externă

1.3. Pompă submersibilă

Această pompă de apă submersibilă este folosită în cadrul aplicației ca și o soluție pentru irigarea automatizată efektivă și eficientă a solului plantei în care este conectat senzorul de umiditate a solului prezentat în subcapitolul anterior. Pompa este comandată de către un modul releu cu un singur canal și comanda se efectuează în funcție de valoarea achiziționată de senzorul de umiditate a solului.

Tensiunea nominală de funcționare a pompei folosite este de 3-5 volți, curent continuu, și are un consum de aproximativ 230 de miliamperi, alături de un debit de 1.2 litri pe minut, cu aproximație. Ideal este ca regimul de punere în funcțiune al acestei componente să lase loc și de pauze pentru a o proteja de evenimente nedorite, precum deteriorarea motorașului din interiorul acesteia. Încă o măsură de protecție a acesteia constă în evitarea funcționării pompei în gol, deoarece această acțiune poate duce la supraîncalzire și, ulterior, la deteriorarea motorului. Pentru comandarea acesteia, se folosește și un modul releu ce operează la tensiunea de 3.3 volți (tensiunea pe care o poate furniza microcontroler-ul Raspberry Pi Pico W), pentru a putea avea un curent suficient de puternic cât să acționeze pompa. Pompa poate fi observată în figura 1.6, iar în figura 1.7 se regăsește schema electrică a conexiunii pompă–modul releu–modul de alimentare externă–microcontroler.



Fig. 1.6 – Pompă de apă submersibilă

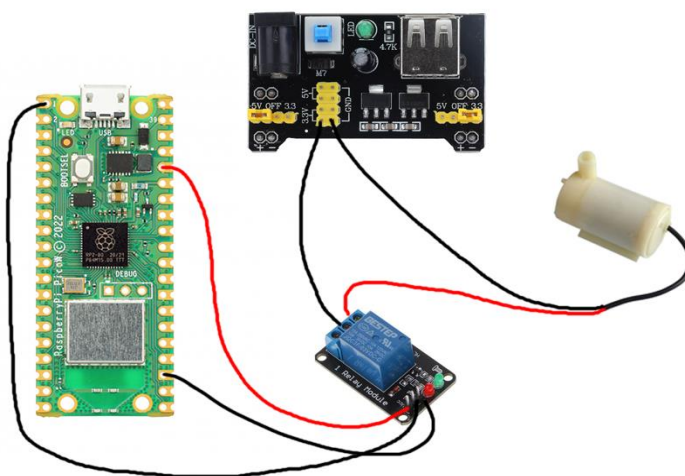


Fig. 1.7 – Schema electrică a ansamblului pompă–modul releu–modul de alimentare externă–microcontroler

Ca și conexiuni, pompa este conectată în felul următor:

- firul de alimentare este conectat la terminalul NC (Normal Închis) al modulului releu;
- firul de potențial 0 este conectat la unul din pinii cu funcție GND de pe modulul de alimentare externă.

1.4. Modul releu 1 canal, 3.3V

Un modul releu constă în două contacte metalice interne ce nu sunt conectate unul cu celălalt. Pe un releu se pot regăsi trei terminale: COM (Common, sau, Comun tradus din limba engleză în română), ce reprezintă terminalul comun atât pentru circuitul de intrare, cât și pentru circuitul de ieșire, NO (Normally Open, sau, Normal Deschis tradus din engleză în română), ce este conectat la terminalul COM atunci când releul este activat și NC (Normally Closed, sau, Normal Închis tradus din engleză în română) ce este conectat la COM atunci când releul nu este activat.

Un switch intern face posibilă conectarea acestor două contacte pentru a forma un circuit electric complet, permițând curentului să treacă. Pentru a activa un releu, un curent este aplicat asupra unei bobine. Acest fapt aduce cele două contacte metalice împreună și permite curentului să “curgă” pe cealaltă parte a releului. Dispozitivul poate fi folosit într-o gamă largă de aplicații, precum automatizarea dispozitivelor electrice (în cazul lucrării prezentate, pompa de apă submersibilă), transmiterea izolată de putere, etc., și sunt deseori integrate în anumite microcontrolere. Acestea pot avea unul, două, patru sau 8 canale. Fiecare canal al unui releu poate controla câte un alt dispozitiv, într-un mod independent față de celelalte canale prezente pe modul.

Am selectat acest modul releu cu 1 singur canal deoarece este comandabil cu o tensiune de 3.3 volți, ceea ce îl face perfect din punctul de vedere al compatibilității și funcționării cu microcontroler-ul Raspberry Pi Pico W, ce nu poate scoate la nicio ieșire de pe placă o tensiune mai mare de 3.3 volți. De asemenea, acest modul releu ne ajută la comandarea pompei în situațiile în care parametrul de umiditate a solului se află sub limita nivelului de umiditate adecvat și este nevoie să pompeze apă în solul plantei.

Aceasta este componenta ce realizează switch-ul pompei și ajută la alimentarea și punerea ei în funcțiune. Modulul este conectat la unul dintre pinii cu funcție GPIO (pinul 1 -

GPIO 0) de pe microcontroler, configurat în modul de ieșire digitală, iar releul este pus în funcțiune și devine activ atunci când pinul nu îi trimite niciun semnal, ceea ce se consideră 0 logic, realizându-se comandarea și punerea în funcțiune a pompei, ceea ce îl face să fie un releu de tip Low Level. În momentul în care acesta primește un semnal cu o amplitudine suficient de mare încât să fie considerat High Level din partea pinului de pe microcontroler, modulul releu este inactiv, respectiv nici pompa nu este comandată. Modul de conectare a modulului releu cu pompa submersibilă, placa de dezvoltare și modulul de alimentare externă poate fi observat în figura 1.7.

Acest modul prezintă patru pini:

- pinul VCC, ce reprezintă alimentarea modulului și este legat la pinul 36 de pe microcontroler, cu funcție de alimentare;
- pinul GND, ce reprezintă ground-ul modulului, care este legat la unul din pinii cu funcție GND a microcontroler-ului;
- pinul IN, ce reprezintă semnalul primit de la unul dintre pinii cu funcție de ieșire digitală de pe microcontroler, ce are puterea de a îl activa (în cazul unui semnal 0 logic) și de a îl dezactiva (în cazul unui semnal 1 logic), și este legat la pinul 1 al microcontroler-ului.

Terminalele modulului releu sunt conectate după cum urmează:

- terminalul NO (Normal Deschis) rămâne liber
- terminalul NC (Normal Închis) este conectat la firul de alimentare al pompei de apă
- terminalul COM (Comun) este conectat la alimentarea cu 3.3 volți de pe modulul sursă de alimentare externă.

Prin această conexiune se face posibilă legarea pompei la sursa externă atunci când modulul releu este pus în funcțiune de către microcontroler.

1.5. Modul buton

Modulul buton reprezintă un buton fizic care, atunci când este apăsat, transmite un impuls către pinul digital la care este conectat. Acesta poate fi utilizat pentru schimbarea stării unui pin digital din cardul microcontrolerului din nivelul 0 logic în 1 logic, astfel se pot utiliza subrutini de întrerupere, care să se efectueze la apăsarea butonului.

Ca și construcție, acest modul are 4 pini, care sunt interconectați între ei în număr de câte 2, ca și în figura 1.8. Motivul pentru care acest modul este construit în formatul de 4 pini este pentru o montare mai ușoară pe placa de realizare a circuitelor.

Ca și principiu de funcționare, la apăsarea butonului se închide circuitul său, ceea ce duce la transmiterea semnalului către pinul la care este conectat.

Conexiunea lor cu microcontroler-ul este prezentată în figura 1.9 și este una dintre cele mai simple de realizat:

- un set de pini se conectează la pinul cu funcția de intrare digitală de pe microcontroler;
- un set de pini se conectează la cel mai apropiat pin GND față de pinul de intrare digitală.

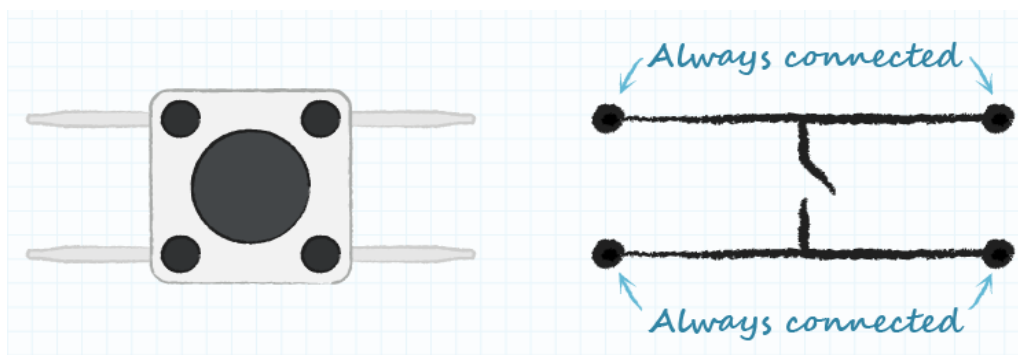


Fig. 1.8 – Construcția modulului buton

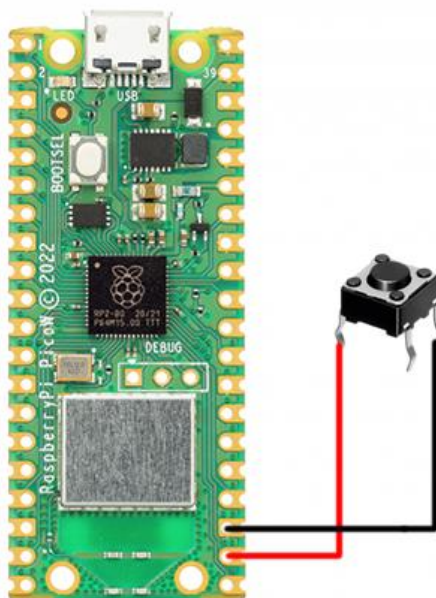


Fig 1.9 – Conexiunea butonului cu microcontroler-ul

1.6. Modul ecran LCD, 16x2 și protocolul I2C

Acest modul ecran LCD are la baza funcționării sale protocolul IIC, însă este posibilă comunicarea cu acesta și prin protocolul I2C, pentru primirea și afișarea informațiilor transmise prin microcontroler. Folosindu-ne de el, avem posibilitatea de a afișa câte 16 caractere pe cele 2 linii de care acesta dispune. Pentru aplicația pe care am elaborat-o, am preferat să folosesc acest dispozitiv în protocolul I2C, datorită gradului de flexibilitate pe care acesta îl oferă în dezvoltare și a numărului redus de pini ce îl folosește în comparație cu alte protocoale.

Ecranul este folosit în cadrul proiectului pentru afișarea parametrilor achiziționați de către cei doi senzori utilizați, mai exact umiditatea din solul plantei și nivelul de apă din bazinul în care este introdusă pompa de apă submersibilă, dar și pentru a afișa informații precum starea solului, valoarea digitală citită de la senzor și data și ora curentă. Selecția între afișarea acestor informații pe ecran se face prin apăsarea unui buton.

Acesta este conectat la o placă de dezvoltare Arduino Uno, ce comunică cu Raspberry Pi Pico W prin intermediul protocolului de comunicare UART, în vederea primirii de date și afișarea acestora pe ecranul LCD.

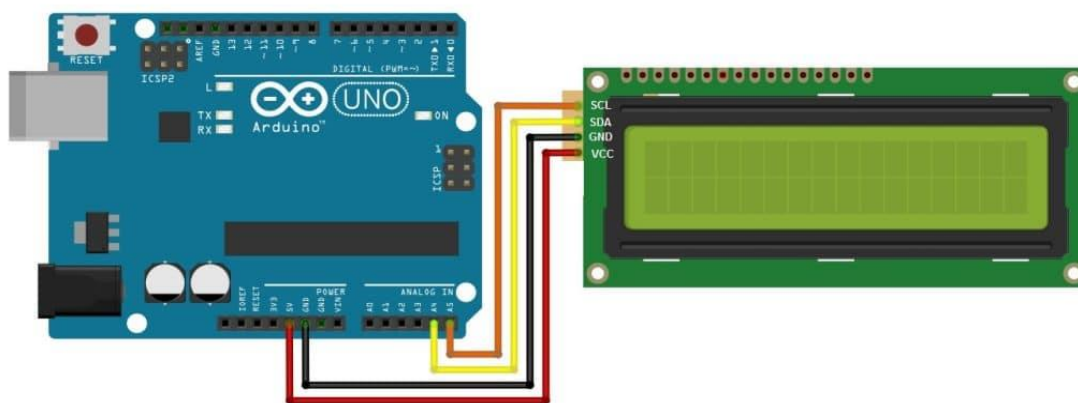


Fig. 1.10 – Schema electrică dintre microcontroler și modulul ecran LCD

Pentru funcționarea ecranului în protocolul I2C, este necesar să se realizeze următoarele conexiuni între dispozitiv și placă, după cum se prezintă și în figura 1.10:

- pinul SCL al modului, care este folosit pentru sincronizarea clock-ului dintre modul și placa de dezvoltare în scopul sincronizării transmiterii și primirii de informații, este

conectat la pinul A5 al plăcuței Arduino Uno, ce are funcția prestabilită de SCL pentru I2C;

- pinul SDA al modulului, ce este folosit pentru transmisia și recepționarea de date pe baza clock-ului de pe pinul SCL, este conectat la pinul A4 de pe Arduino, ce are funcția prestabilită de SDA în scopul funcționării în modul I2C;
- pinul GND, care reprezintă practic pinul de potențial 0, este conectat la unul dintre pinii cu funcția GND de pe placă;
- pinul VCC, ce reprezintă pinul de alimentare, este conectat la pinul Arduino-ului de alimentare cu o tensiune de 5 volți.

Acest modul operează la o tensiune cuprinsă între 4.5 și 5.5 volți, are 2.6 inchi iar textul afișat pe acesta este de culoare neagră, cu un fundal verde. Curentul recomandat la care acest modul operează este de 80 de miliamperi.

Pentru a transmite un mesaj către ecran, microcontroler-ul trimite întâi un set de instrucțiuni prin conexiunea I2C, instrucțiuni ce conțin o adresă a dispozitivului, adresă ce face posibilă asigurarea faptului că doar dispozitivul corect ascultă și primește transmisia. De asemenea, este trimis și un bit ce informează dacă microcontroler-ul dorește să citească sau să scrie o informație.

Fiecare mesaj trimis trebuie să aibă o confirmare de primire, pentru a nu exista situații neașteptate, astfel, când ecranul primește informația ce trebuie afișată, acesta înștiințează microcontroler-ul, pentru a putea trece la următorul set de biți.

Formatul unui mesaj în protocolul I2C este structurat în felul următor, după cum se poate observa și în figura 1.11:

- startul transmisiei
- 7 sau 10 biți ce specifică adresa dispozitivului cu care se comunică
- un bit care informează dacă dispozitivul vrea să primească sau să transmită date
- un bit ce confirmă primirea informației
- un pachet de 8 biți cu date;
- încă un bit de înștiințare a primirii datelor;
- încă un pachet de 8 biți cu date;
- un ultim bit de înștiințare a primirii datelor;
- stopul transmisiei.

Cunoscând formatul prin care se comunică în protocolul I2C, putem înțelege cu mai multă ușurință rolul pinului SCL, care este responsabil de sincronizarea clock-urilor celor două dispozitive, asigurându-se astfel că acestea sunt înștiințate cum sunt delimitate începutul secvenței, pachetul ce conține adresa, bitul de scriere/citire, biții de înștiințare, pachetele ce conțin informații și finalul secvenței. Prin pinul SDA trece fluxul de date.

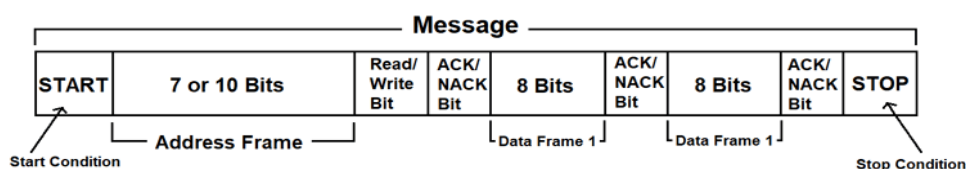


Fig. 1.11 – Formatul unui mesaj transmis prin protocolul I2C

1.7. Senzor de nivel al apei

Senzorul de nivel al apei este un dispozitiv analogic ce monitorizează nivelul de lichid dintr-un bazin, container sau rezervor. Acesta este constituit din zece trasee de cupru expuse, dintre care cinci sunt destinate ca fiind trasee de alimentare, iar celelalte cinci reprezintă trasee de măsurare. Aceste trasee sunt intercalate între ele astfel încât există câte un traseu de măsurare între fiecare două trasee de alimentare.

În mod normal, aceste două trasee, de alimentare și de măsurare, nu sunt conectate. Însă, când sunt imersate într-un lichid, ele formează o conexiune.

Modulul are un LED integrat ce se aprinde atunci când senzorul este alimentat, pentru a confirm alimentarea acestuia, după cum se poate observa în figura 1.12.

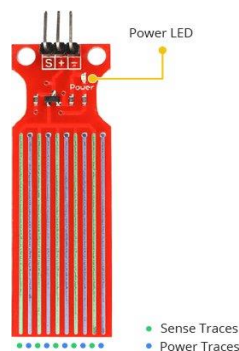


Fig. 1.12 – Senzorul de nivel al lichidului

Principiul funcționării acestui dispozitiv este unul relativ simplu: traseele de măsurare și de alimentare formează un rezistor variabil, a cărei rezistență variază în funcție de cât de adânc sunt expuse la lichid. Rezistența variază invers proporțional cu adâncimea imersării în apă a senzorului.

Cu cât senzorul este imersat mai mult în apă, cu atât conductivitatea va fi mai mare și rezistența mai mică, iar cu cât senzorul este imersat mai puțin în apă, cu atât conductivitatea va fi mai slabă și rezistența va fi mai mare.

În consecință, senzorul generează la ieșire o tensiune care variază în funcție de valoarea rezistenței și, măsurând această tensiune, putem determina nivelul unui lichid aflat într-un anumit volum.

Senzorul în cauză are în construcția sa trei pini:

- pinul S, care reprezintă semnalul analogic de la ieșirea senzorului ce poartă informații despre nivelul apei din bazin, este conectat la pinul 32 al plăcii de dezvoltare, pin ce are funcție de ADC, adică convertește semnalul obținut din domeniul analogic în domeniul digital, făcând posibilă afișarea și prelucrarea sa de către dispozitive digitale;
- pinul +, care reprezintă pinul de alimentare al modulului, și este conectat la pinul 36 al plăcii, pinul de alimentare cu 3.3 volți;
- pinul -, ce reprezintă ground-ul acestui dispozitiv, și este conectat la unul din pinii de pe placă cu funcția de GND.

În cadrul proiectului, acest senzor este folosit pentru a ne înștiința atunci când trebuie adăugată apă în bazinul în care se află submersată pompa ce contribuie la irigarea automată.

Senzorul de nivel al apei este un dispozitiv ce operează la o tensiune de 3-5 volți, curent continuu, și un curent de minim 20 de miliamperi. Modul de conectare al acestuia la microcontroler este prezentat în figura 1.13.

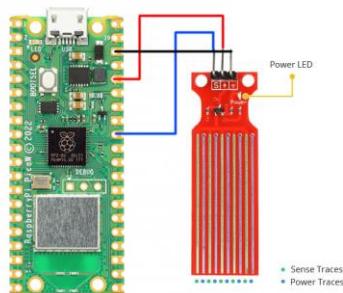


Fig. 1.13 – Conectarea senzorului de nivel al lichidelor la microcontroler

Motivele pentru care am ales acest senzor în dezvoltarea sistemului prezentat sunt costul redus al acestuia și eficiența relativ acceptabilă cu un grad de toleranță, calități ce îl fac recomandat pentru acest tip de aplicație.

1.8. Modul buzzer MH-FMD

Modulul buzzer MH-FMD este un dispozitiv ce este constituit dintr-un buzzer piezoelectric și un oscilator incorporat ce este capabil să genereze un sunet cu o frecvență de rezonanță de aproximativ 2.5 de kHz (cu o toleranță de $\pm 300\text{Hz}$), la un nivel de 85 de decibeli până în distanța de 10 centimetri, însă, nu are capacitatea de a modifica tonul sunetului emis, deoarece este un buzzer activ. Acesta operează la o tensiune între 3 și 5 volți și un curent de 30 de miliamperi. MH-FMD funcționează pe o logică inversă, adică acesta este pus în funcțiune și poate emite sunet doar atunci când pinul de output de pe microcontroler la care este modulul conectat nu trimite către acesta niciun semnal, adică se află în starea 0 logic. În cazul în care acesta primește la intrare un semnal cu o amplitudine suficient de mare cât să îi treacă starea în nivelul 1 logic, sunetul încetează.

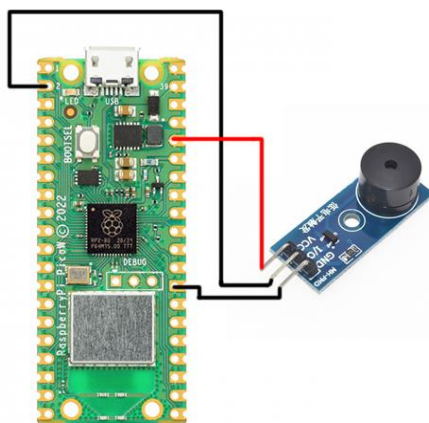


Fig 1.14 – Conectarea modulului buzzer MH-FMD la microcontroler

Modulul MH-FMD are la bază trei pini folosiți pentru configurarea acestuia, conform figurii 1.14:

- pinul GND, ce reprezintă ground-ul dispozitivului și se leagă la unul dintre pinii cu funcția GND de pe microcontroler;

- pinul I/O, ce reprezintă pinul ce primește semnal de la unul din pinii microcontrolerului cu funcție de GPIO, și este conectat la pinul 2 (GPIO 1) ce este configurat în modul output;
- pinul VCC, ce reprezintă alimentarea modulului, și este legat la pinul 36 de pe plăcuță, pin ce are funcție de alimentare la o tensiune de 3.3 de volți.

Scopul acestui modul este de a anunța operatorul uman în situația în care trebuie reumplut bazinul din care se pompează apă către solul plantei atunci când umiditatea acestuia nu se află în parametrii impuși de către aplicație.

1.9. Placa de dezvoltare Raspberry Pi Pico W

Raspberry Pi Pico W este un o placă de dezvoltare a sistemelor embedded din familia de microcontrolere Raspberry, cu un grad ridicat de performanță și cu un preț care este cu mult sub alte opțiuni valabile pe piață, fără a sacrifica din calitatea și eficiența pe care o oferă în dezvoltarea sistemelor embedded. La baza plăcii de dezvoltare stă cipul RP2040, ce constituie microcontroler-ul în adevăratul sens al cuvântului. Limbajele de programare în care se poate realiza configurarea sistemului sunt Assembly, MicroPython, CircuitPython, C/C++ și Rust. Dintre caracteristicile sale fundamentale, cele mai importante de menționat sunt:

- cipul RP2040, care stă la baza plăcii de dezvoltare și constituie microcontroler-ul în adevăratul sens al cuvântului;
- procesorul Arm Cortex M0+ pe 32 de biți ce dispune de două nuclee de tip M0+ (ceea ce face posibilă configurarea unei aplicații de timp real dezvoltat pe mai multe nuclee și fire de execuție) și are un clock flexibil, ce se poate duce până în valoarea de 133 MHz;
- un pin capabil să scoată o tensiune de alimentare de 3.3 volți la ieșirea sa;
- 2MB de memorie flash pentru memorarea programelor și 264kB de memorie RAM statică, ce duce la o performanță mai mare cu un consum de putere mai mic;
- 2 pini ce pot fi configurați să ia parte la comunicații de tip UART;
- 2 pini cu controllere ce facilitează transmisiile în protocolul SPI;
- 2 pini cu controllere ce fac posibilă trimiterea de date și informații prin protocolul I2C;

- 16 canale PWM cu posibilitatea de configurare a parametrilor specifice tipului de semnal, precum factorul de umplere și frecvența semnalului;
- modul ceas și modul timer de o acuratețe fină;
- 26 de pini ce au funcția de GPIO ce pot fi configurați să deservească roluri de intrări sau de ieșiri, în funcție de specificațiile aplicației;
- modul pentru conexiune la rețele de internet în mod Wireless de 2.4GHz, cu o singură bandă;
- modul pentru conexiune Bluetooth 5.2;
- 8 pini de tip GND pentru conectarea pinilor de potențial 0 a diferitelor dispozitive electronice folosite în construcția diverselor aplicații;
- 3 pini cu funcția de ADC, adică de conversie a semnalelor din domeniul analogic în domeniul digital, ce se dovedesc a fi extrem de utili în aplicațiile ce au în construcția lor un set de senzori a căror valoare trebuie afișată și/sau prelucrată.

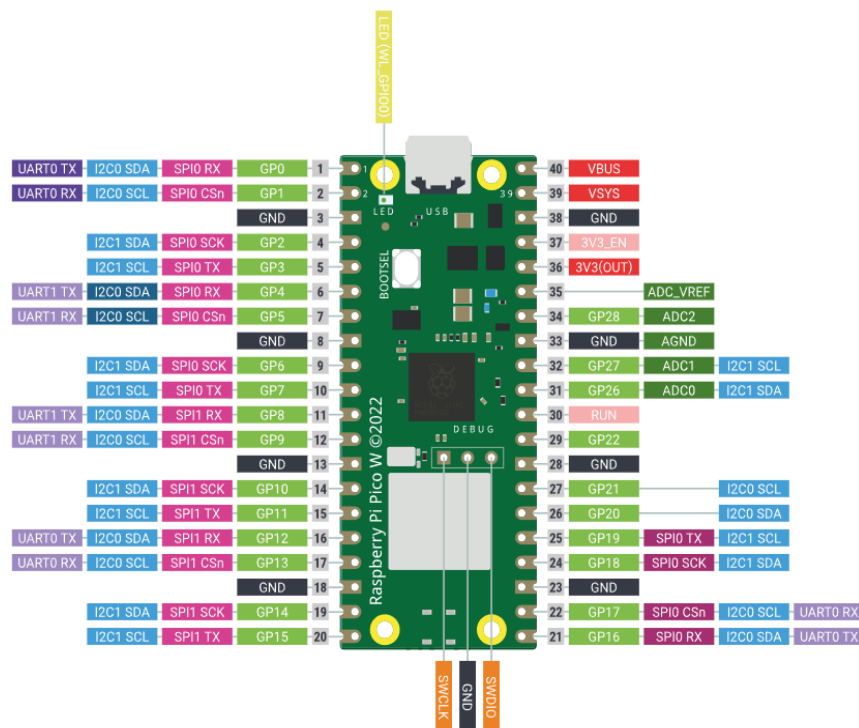


Fig. 1.15 – Diagrama pinilor de pe placa de dezvoltare Raspberry Pi Pico W

Pentru o prezentare mai detaliată a pinilor de pe această placă și a funcțiilor pe care aceștia le pot îndeplini, vizualizați figura 1.15.

Există mai multe avantaje în folosirea acestei plăcuțe de dezvoltare decât cele enumerate mai sus, însă, cele descrise sunt cel mai frecvente specificații ce îl recomandă și de care depinde cel mai mult funcționarea întregului sistem.

La baza denumirii cipului RP2040 stau mai mulți factori ce îl caracterizează, prezentați în figura 1.16. În figura 1.17 este prezentată arhitectura hardware a cipului RP2040.

Fiecare pin ce poate îndeplini una din funcțiile descrise în figura 1.18 este conectat la unul din multitudinea de module ce intră în arhitectura întregului microcontroler.

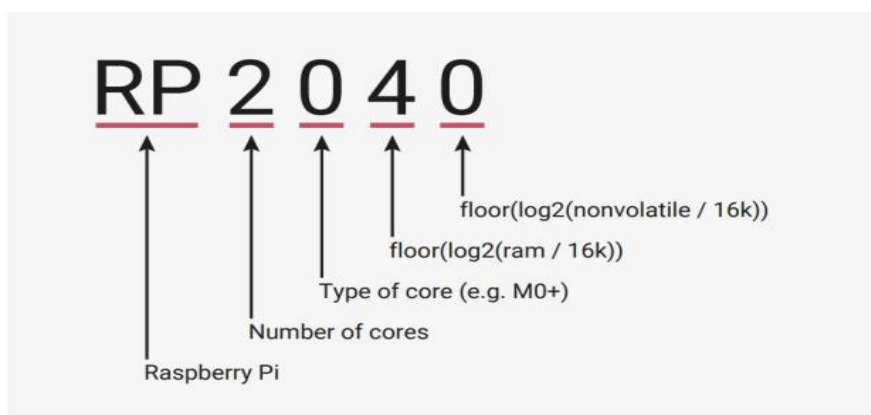


Fig. 1.16 – Semnificația acronimului RP2040

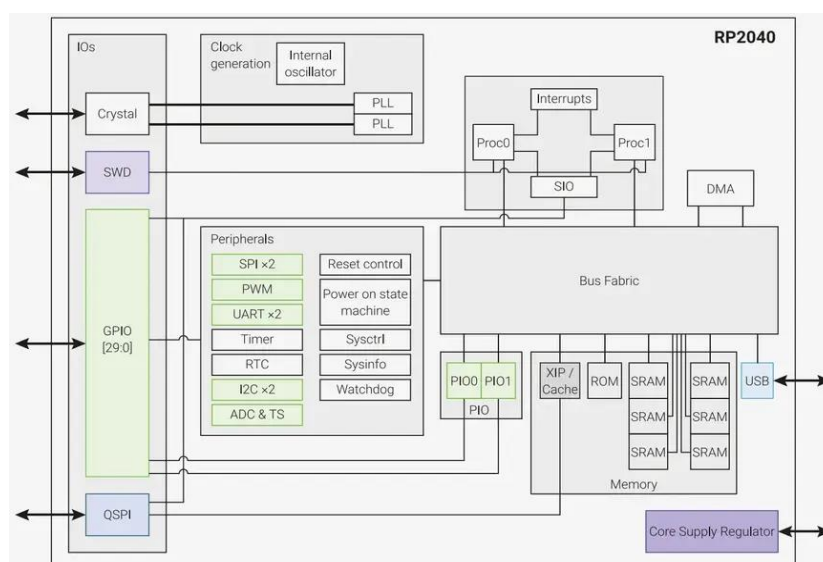


Fig. 1.17 – Arhitectura hardware a cipului RP2040

1.10. Placa de dezvoltare Arduino Uno (revizia 3)

Arduino Uno Rev. 3 este o placă de dezvoltare populară în lumea sistemelor de tip embedded, datorită ușurinței și nenumăratelor opțiuni în cadrul programării sale. Aceasta este bazată pe microcontroler-ul ATmega328P. Din punctul de vedere a programării plăcii, aceasta nu este foarte flexibilă, având ca și principale opțiuni doar limbajele de programare C și C++.

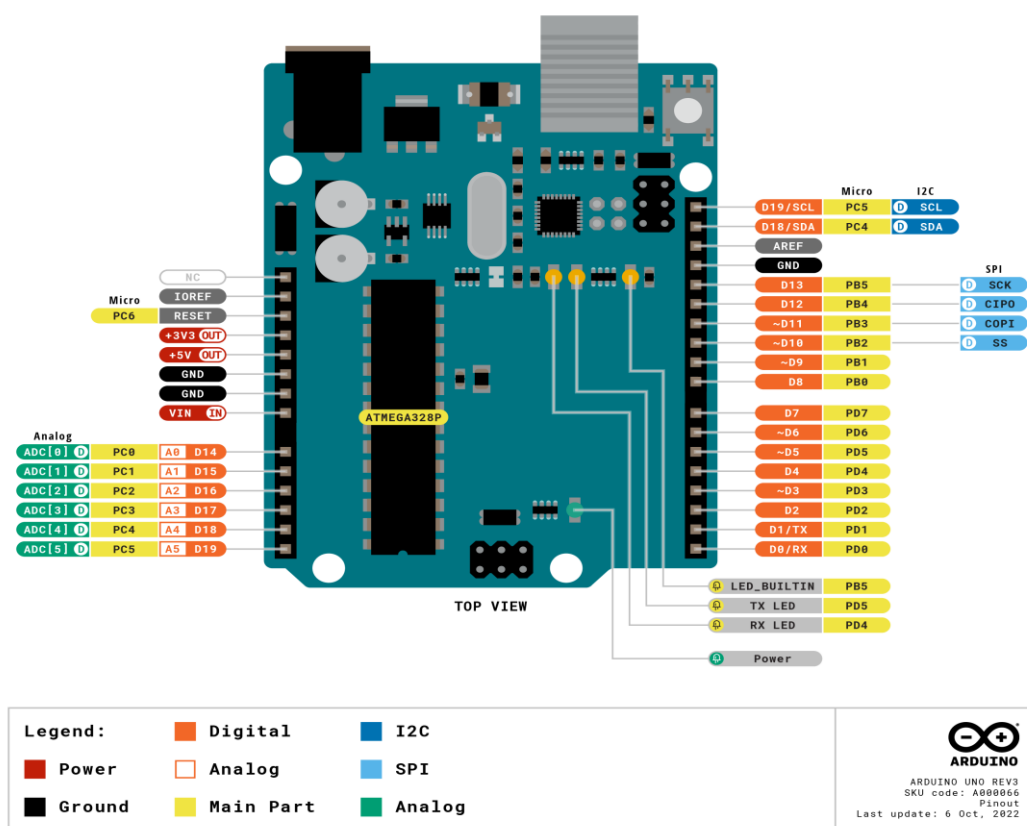


Fig 1.18 - Diagrama pinilor de pe placa de dezvoltare Arduino Uno (Rev. 3)

Caracteristicile care recomandă folosirea Arduino Uno în dezvoltarea sistemelor sunt:

- arhitectura microcontroler-ului pe 8 biți AVR;
- viteza clock-ului, ce operează în general la frecvența de 16MHz, însă poate fi configurat și pentru frecvențe mai mici;
- 32KB de memorie flash pentru memorarea programelor;
- 2KB de memorie RAM statică pentru memorarea temporară a datelor;

- 23 de pini digitali, configurabili pentru a fi intrări sau ieșiri;
- intrări analogice, capabile de o conversie analog-digitală cu o rezoluție de 10 biți;
- temporizatoare și numărătoare ce includ o gamă de astfel de module pentru generarea semnalelor de tip PWM, operații în funcție de timp, etc.;
- controlere pentru diverse protocoale de comunicație, precum UART, SPI, I2C, ce permit interfațarea microcontroler-ului cu alte dispozitive;
- tensiune de operare de 5 volți, însă poate opera și la tensiuni mai mici, până la 1.8 volți, în funcție de condiții specifice de operare;

Pinii de pe acest dispozitiv pot îndeplini anumite funcții, ce sunt vizibile în figura 1.18, în funcție de tipul de aplicație care se dorește a fi dezvoltat. Arhitectura cipului ATmega328P folosit în Arduino Uno este atașată figurii 1.19.

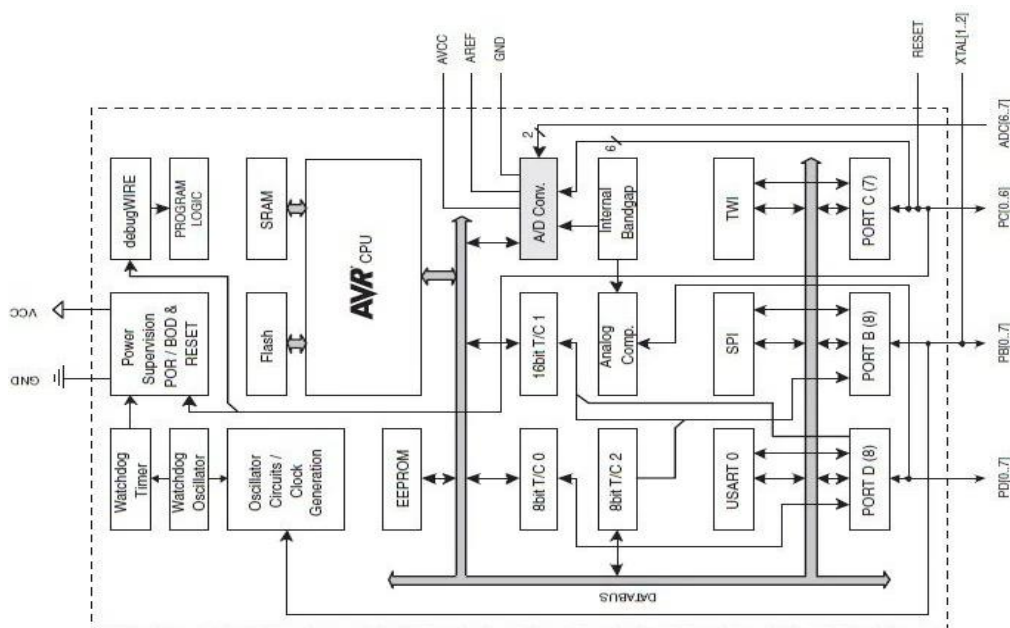


Fig. 1.19 – Arhitectura hardware a cipului ATmega328P

1.11. Conversia analogic-digitală prin Raspberry Pi Pico W

Raspberry Pi Pico W dispune de un modul canale ADC cu 3 canale ce reprezintă, practic, un convertor analogic-digital. Acesta este conectat la pinii 34 (GPIO 28), 32 (GPIO27) și 31 (GPIO 26), ce deservește drept canale ce pot fi utilizate pentru prelucrarea într-o valoare numerică digitală a unui semnal de natură analogică primită de la ieșirea unui dispozitiv.

Acest convertor este capabil de o conversie pe 12 biți și are o rată de eșantionare de 500000 de eșantioane pe secundă, folosindu-se de un clock de 48MHz. În figura 1.20 este prezentată schema bloc a modului ADC.

Acest modul este semnificativ pentru o aplicație precum cea prezentată în lucrare și este esențial datorită capacității sale de a achiziționa diferite date ce reprezintă parametri fizici, oferiți sub formă de semnal analogic din partea senzorilor, și convertirea lor în domeniul digital, făcându-le imediat disponibile pentru a fi afișate și/sau prelucrate pe un dispozitiv digital precum un computer.

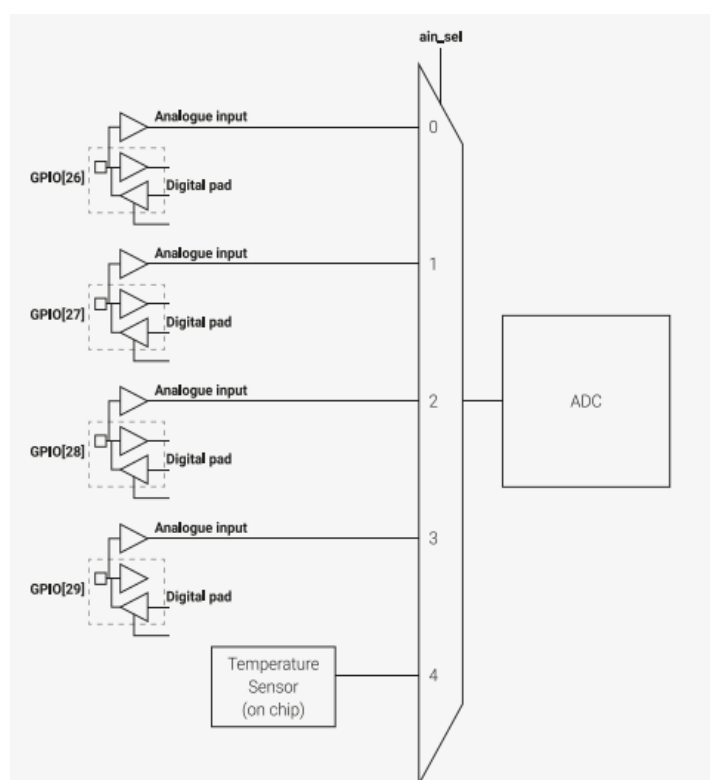


Fig 1.20 – Schema bloc a modului ADC

În afară de cei trei pini ce funcționează în regim de canale ale convertorului, mai există și un al patrulea pin cu funcția de canal ADC. Acel pin este, însă, rezervat în regim exclusiv doar pentru citirea și convertirea valorii primite de la senzorul de temperatură integrat în cip.

Acest modul convertor analogic-digital este unul semnificativ pentru sistemul elaborat deoarece reprezintă principala modalitate de a achiziționa datele transmise de către cei doi senzori ce fac parte din alcătuirea și funcționarea sistemului.

1.12. Protocolul de comunicație UART

Protocolul UART (Universal Asynchronous Receiver-Transmitter în engleză, tradus Receptor-Transmițător Asincron Universal în limba română) este un protocol de comunicație serială ce este folosit pentru a transmite informații între două dispozitive (în cazul acestei lucrări, între două microcontrolere). Este folosit la un nivel larg în domeniul sistemelor de tip embedded și în comunicațiile între computere datorită simplității și eficienței pe care o are la baza funcționării sale.

Ambele microcontrolere implicate în dezvoltarea acestui sistem dispun și se folosesc de acest protocol pentru îndeplinirea cu succes a scopului aplicației.

Cele mai frecvente cazuri în care întâlnim utilizat acest protocol este în comunicația cu dispozitive periferice (precum modulele GPS, modulele Bluetooth), sistemele embedded sau în situații de depanare a sistemelor embedded.

Unele caracteristici cheie ale acestui protocol de comunicație sunt:

- comunicația asincronă, ceea ce înseamnă că nu necesită un semnal de tact pentru sincronizare, asemenea celorlalte protocoale de comunicație sincronă. În schimb, acesta folosește biți de start și de stop pentru a captura datele și pentru a asigura o durată eficientă a acestui proces;
- suportă comunicație full-duplex, adică datele pot fi transmise și primite în mod simultan;
- necesită un hardware minimal, se rezumă doar la necesitatea unui controler UART în dispozitivele ce comunică și o conexiune printr-un cablu serial.

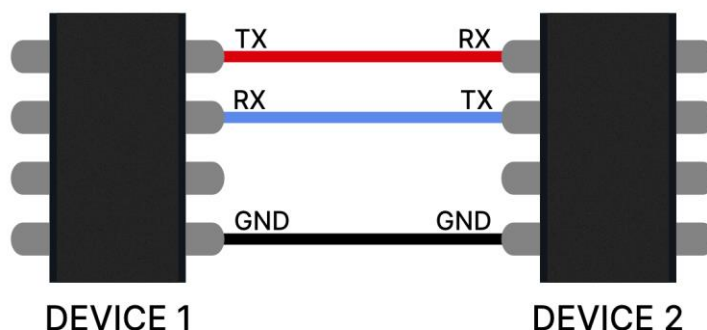


Fig. 1.21 – Comunicația UART, conexiunile necesare între dispozitive

Pentru realizarea comunicației, există conexiuni specifice ce trebuie realizate între cele două dispozitive, prezentate și în figura 1.21:

- pinul TX al dispozitivului 1 se conectează la pinul RX al dispozitivului 2;
- pinul RX al dispozitivului 1 se conectează la pinul TX al dispozitivului 2;
- se conectează GND-urile ambelor dispozitive între ele, pentru a stabili un punct comun de referință între acestea, asigurându-se astfel interpretarea corectă a semnalelor UART și fiabilitatea comunicării acestora.

UART are anumiți parametri ce trebuie configurați înainte de a începe comunicația, iar acești parametri sunt:

- baud rate-ul, ce definește viteza comunicației în biți pe secundă. Baud rate-uri comune folosite în cadrul microcontrolerelor sunt: 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600 biți pe secundă;
- biții de informație, reprezentați și în figura 1.22, ce reprezintă numărul de biți aferenți informației pentru fiecare transmisie realizată și, de obicei, aceștia sunt în număr de 7 sau 8;
- paritatea, ce este configurată pentru a detecta orice eroare posibilă în cadrul transmisiei sau recepției;
- bitul de stop, ce reprezintă bitul care indică sfârșitul cadrului de informații dintr-o transmisie, și poate fi configurat ca fiind unul sau doi biți de stop prezenți într-un cadru întreg UART.

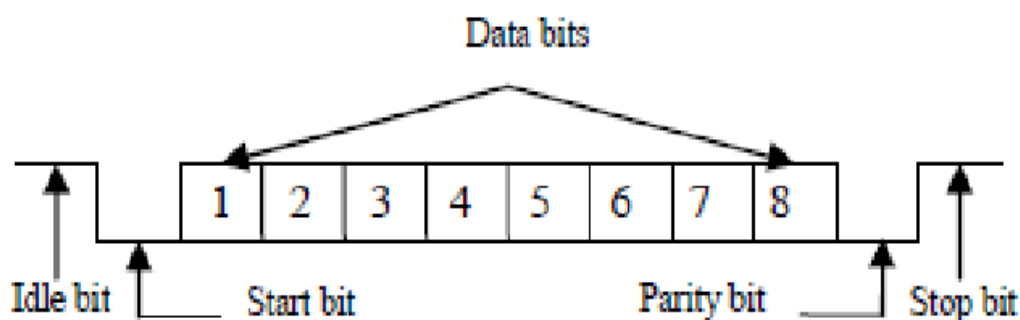


Fig. 1.22 – Formatul unui mesaj transmis în protocolul UART, cu 8 biți pentru informație

Dezavantajele în folosirea acestui protocol sunt reprezentate de:

- limitarea de distanță între dispozitivele ce comunică, acestea trebuie să fie aproape unul de celălalt;
- limitarea de viteză în ceea ce privește transmisia de date, fiind relativ mai puțin rapid decât protocoale precum SPI sau I2C;
- capacitate limitată în situații de depanare, față de alte protocoale de comunicație.

Capitolul 2

Instrumente software folosite în realizarea sistemului

2.1. Limbajul de programare MicroPython

MicroPython este un limbaj de programare ce reprezintă o implementare a limbajului Python 3. Ceea ce îl face diferit de cel din urmă este faptul ca MicroPython a fost dezvoltat special pentru execuția de instrucțiuni de către dispozitive de tip microcontroler, fiind cel mai îndrăgit limbaj în lumea pasionaților de microcontrolere din familia Raspberry.

Fiind o implementare a Python 3, MicroPython beneficiază de aceeași simplitate, prin prezența unor sintaxe intuitive și a librăriilor standard sau adăugate de utilizatori, ceea ce constituie un avantaj prin ușurința de a scrie și executa linii de cod direct de pe un microcontroler.

Având în vedere faptul că MicroPython a fost creat special pentru microcontrolere, acesta este optimizat în ceea ce privește consumul de resurse specifice și oferă acces direct la hardware-ul disponibil, prin posibilitatea de a controla pinii de intrare și de ieșire digitală, pinii cu diverse funcții de I2C, ADC, UART, SPI, etc..

Acesta are o amprentă digitală relativ mică în comparație cu alte limbaje, ceea ce înseamnă că nu necesită decât o porțiune relativ mică din spațiul de stocare, făcându-l potrivit pentru dispozitivele cu memorie de tip flash și memorie de tip RAM limitate. În prim planul acestui limbaj de programare stă performanța și eficiența în utilizarea resurselor de memorie.

```
from machine import Pin
import time

# Initializam pinul 2 ca si iesire digitala
led = Pin(2, Pin.OUT)

while True:
    led.high() # Porneste ledul
    time.sleep(1) # Asteapta 1 secunda
    led.low() # Opreste ledul
    time.sleep(1) # Asteapta 1 secunda
```

Fig. 2.1 – Exemplu de cod scris în MicroPython

Un exemplu simplu de cod scris în acest limbaj este prezentat în figura 2.1, iar scopul secvenței scrise este de a repeta bucla while la nesfârșit. Instrucțiunile specificate în buclă nu fac nimic altceva decât să pornească și să stingă la nesfârșit un led conectat pe pinul de index 2 al dispozitivului pe care se încarcă codul. Codul poate fi explicat astfel:

- `'from machine import Pin'` realizează importarea sintaxelor și funcțiilor specifice configurării pinilor;
- `'import time'` realizează importarea sintaxelor și funcțiilor specifice execuției programului cu funcții de timp;
- `'led = Pin(2, Pin.OUT)'` realizează declararea și inițializarea pinului la care este conectat ledul, unde parametrul `'2'` reprezintă indexul pinului folosit, iar `'Pin.OUT'` indică faptul că pinul respectiv este inițializat în modul de ieșire digitală;
- bucla `'while True'` realizează execuția continuă a codului, unde parametrii `'led.high()'` și `'led.low()'` controlează starea de pe pinul de ieșire digitală a plăcii, adică duc acel pin în stări de 0 logic și 1 logic la nesfârșit;
- `'time.sleep(1)'` este o funcție a librăriei `'time'` și încetează execuția codului pentru o valoare în secunde, în cazul acesta 1 secundă.

Din acest exemplu putem observa și similaritățile dintre MicroPython și Python ca și structură, declarația variabilelor, declarația librăriilor folosite, forma de apelare a funcțiilor specifice și sintaxele folosite.

Acest limbaj este folosit cel mai adesea în aplicații precum:

- controlul senzorilor și citirea datelor de pe aceștia;
- controlul actuatorilor;
- construirea de dispozitive IoT (Internet of Things – internetul lucrurilor);
- prototiparea rapidă a obiectelor hardware;
- învățarea programării pe dispozitive electronice pentru începători;

Deși MicroPython este o implementare a Python-ului, croită în mod specific cu atenție pentru microcontrolere, este adesea numit un limbaj de programare diferit față de restul datorită accentului pus pe sistemele embedded și implementările prin care oferă acces la în cod la nivel hardware.



Fig 2.2 – Logo-ul MicroPython

Pentru a utiliza MicroPython pe un microcontroler din familia Raspberry, trebuie întâi descărcat fișierul cu extensia “.uf2” specific versiunii dorite, ce conține definiții ale limbajului. Ulterior, acesta trebuie încărcat în memoria flash a plăcii de dezvoltare. Pentru a accesa această memorie, trebuie să fie ținut apăsat butonul BOOTSEL de pe dispozitiv și apoi trebuie să se conecteze placa la computer.

După ce se urmează pașii anterior menționați, pentru a începe lucrul cu MicroPython-ul va fi nevoie de un mediu de dezvoltare a aplicațiilor care să fie compatibil cu acesta. Mediul de dezvoltare folosit în principal de către utilizatorii de MicroPython este Thonny IDE.

2.2. Mediul de dezvoltare a aplicațiilor Thonny IDE

Thonny IDE este un mediu de dezvoltare în programare suficient de intuitiv pentru a fi folosit de către începători, însă, suficient de puternic pentru dezvoltarea unor aplicații complexe de către utilizatorii ceva mai experimentați.

Motivul pentru care acest mediu de programare este numit ca fiind ‘prietenos cu începătorii’ datorită design-ului programului, ce prezintă o interfață simplă și curată, lipsită de elemente ce nu își au rostul și care doar încarcă mult prea mult aspectul interfeței, lăsând

utilizatorul să se concentreze doar pe ceea ce contează cu adevărat: funcționalitatea și principiile de funcționare ale codului scris.

Acesta vine cu la pachet cu limbajul Python deja instalat, așa că nu necesită instalarea separată a acestuia, însă, pentru scrierea de cod în MicroPython, trebuie urmați pașii menționați înainte.

Programul are integrat și un depanator ce permite utilizatorului să execute codul linie cu linie, făcând mai ușoară înțelegerea modului de funcționare a acestuia și detecția de erori și motivul pentru care acestea apar în codul scris. Mai mult decât atât, valorile variabilor sunt afișate concomitent cu rularea liniilor, pentru a ajuta utilizatorul să țină cont de evoluția lor în timp real.

Datorită suportului pentru MicroPython al acestui mediu de programare, Thonny IDE este adesea folosit în construirea de aplicații de tip IoT sau în sisteme de tip embedded.

Thonny IDE dă dovadă de ușurință și simplitate chiar și atunci când vine vorba de instalarea de librării, simplificând procesul de extindere a funcționalității prin librării externe. Librăriile sunt pachete cu ajutorul cărora putem introduce în program diferite funcționalități care nu se află în programul de bază. Acestea trebuie mereu să fie declarate în primele linii de cod, înainte de declararea vreunei variabile sau de definirea vreunei funcții. Un exemplu de astfel de librărie ar fi librăria ‘machine’ ce permite declararea de pini digitali, a funcțiilor pe care aceștia le îndeplinesc și o serie de astfel de operațiuni concentrate asupra pinilor. În figura 2.1 din subcapitolul anterior poate fi consultat un exemplu cu privire la cum se declară o librărie.

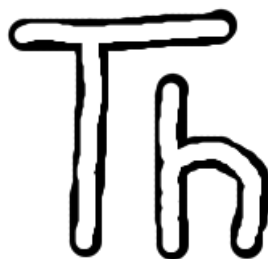


Fig 2.3 – Logo-ul Thonny IDE

În concluzie, Thonny IDE oferă o soluție simplă și puternică pentru învățarea limbajelor de programare Python și MicroPython, în timp ce prezintă și o balanță între nivelul de simplitate al programului și funcționalitatea acestuia. Caracteristicile cheie ale acestuia sunt ajutarea începătorilor în înțelegerea conceptelor utilizate în programare și în diversele sintaxe utilizate, făcându-l o alegere excelentă pentru medii educaționale și programatori aflați la început de drum.



Fig 2.3 – Logo-ul Python

2.3. Limbajul de programare folosit pentru Arduino Uno

Limbajul de programare folosit în cadrul dezvoltării de aplicații pe plăcile de dezvoltare Arduino Uno sunt bazate pe limbajul C++ și reprezintă o implementare a acestuia, cu abstracționări adiționale și librării specifice hardware-ului de care dispune dispozitivul. Acesta include caracteristici clasice ale C++, cum ar fi tipurile de date, structurile de control, funcțiile și clasele.

```
const int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Fig 2.4 – Exemplu de cod pentru Arduino Uno

Codul scris în acest limbaj are o structură specifică cu două funcții principale, prezentate și în figura 2.4 ce are aceeași funcționalitate ca și exemplul din prezentarea limbajului MicroPython:

- funcția ‘setup()’, ce rulează doar o dată la execuția programului și este de obicei folosită pentru a declara și inițializa diferite variabile, configurări ale pinilor, începerea folosirii unor librării, etc.;
- funcția ‘loop()’, ce rulează în mod continuu după execuția funcției ‘setup()’ și are scopul de a controla în mod activ microcontroler-ul;
- ‘const int ledPin = 13’ este o constantă cu valoare întreagă ce definește indexul pinului conectat la led;
- ‘pinMode(ledPin, OUTPUT)’ inițializează pinul folosit, parametrul ‘ledPin’ reprezintă indexul pinului iar parametrul ‘OUTPUT’ semnifică rolul de ieșire digitală a celui pin;
- ‘digitalWrite(ledPin, HIGH)’ este funcția ce duce pinul în starea 1 logic și aprinde ledul;
- ‘digitalWrite(ledPin, LOW)’ este funcția ce duce pinul în starea 0 logic și stinge ledul;
- ‘delay(1000)’ este o funcție care oprește execuția programului pentru o valoare în milisecunde, 1000 de milisecunde (1 secundă) în exemplul dat;

Acest limbaj reprezintă un nivel de dificultate redus în utilizare și are scopul de a face programarea asupra elementelor hardware accesibilă începătorilor și celor ce fac din asta un hobby.

În concluzie, acest limbaj nu reprezintă decât o versiune simplificată de C++ și adaptată la cerințele plăcii de dezvoltare, abstractizând o mulțime de elemente ale programării în sisteme embedded și permițând utilizatorilor să creeze proiecte și să învețe printr-o experiență directă.

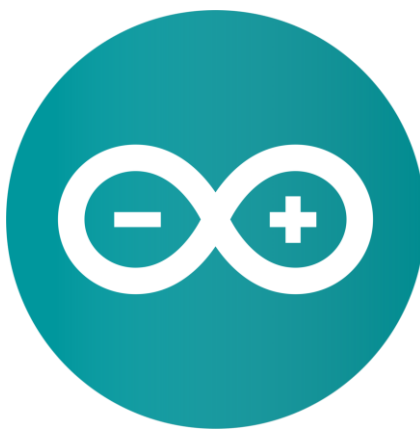


Fig 2.5 – Logo-ul Arduino

2.4. Mediul de dezvoltare a aplicațiilor Arduino IDE

Arduino IDE este un program ce servește drept mediu de dezvoltare pentru programare și a fost construit în mod direct pentru microcontrolerele din familia Arduino. Este o platformă prietenoasă cu utilizatorul, în sensul că interfața sa nu este încărcată cu elemente inutile ce tind să încurce utilizatorii, mai ales pe cei începători în programare.

Programul prezentat este destinat scrierii, compilării și încărcării de cod în memoria microcontrolerelor Arduino, fiind construit într-un mod special ce simplifică tot procesul de programare, fapt ce îl face utilizabil pentru începători și eficient pentru utilizatorii ce dispun de cunoștințe avansate.

O serie de caracteristici cheie ale acestui program software constau în:

- compatibilitatea între platforme diferite, având valabilitate pentru Windows, macOS și Linux;
- după cum a fost menționat anterior, acesta dispune de o interfață curată și intuitivă;
- include un editor de cod, o zonă de mesaje, o consolă de text, o bară de instrumente cu butoane pentru funcții comune și o consolă în care se pot citi mesajele din transmisiile seriale;
- ușurința în instalarea de librării adiționale, iar acesta vine la pachet cu o mulțime de librării preinstalate ce ajută utilizatorul în programarea diverselor dispozitive, precum senzori, ecrane, protocoale de comunicație, etc.;
- editorul de cod oferă funcționalități de bază precum evidențierea sintaxelor, indentare automata și potrivirea acoladelor ce se pun la începutul și sfârșitul unei funcții;
- încărcarea programelor pe plăcuță se face într-un mod simplificat, printr-un simplu click, deoarece Arduino IDE este capabil să transforme codul într-un format pe care microcontroler-ul îl poate înțelege și executa;
- monitorul serial permite utilizatorilor să trimită și să primească informații de la portul serial al plăcuței, lucru ce este vital pentru depanarea și monitorizarea comportamentului codului executat în timp real;
- Arduino IDE are în spate o comunitate imensă și activă ce contribuie cu ghiduri, proiecte, răspunsuri la întrebări și librării.

În concluzie, Arduino IDE prezintă o soluție puternică și ușor de utilizat pentru scrierea, execuția și încărcarea codului pe microcontrolerele Arduino și dă dovadă de un nivel înalt de simplitate în realizarea acestor procese.

2.5. Diferențe între MicroPython și adaptarea de C++ a Arduino

MicroPython și adaptarea de C++ a Arduino-ului servesc scopuri similare, însă diferă semnificativ în ceea ce privește abordarea, capabilitățile și audiențele lor țintă.

Unele dintre diferențele cheie dintre acestea sunt:

- MicroPython este o adaptare a Python 3, făcută să ruleze pe microcontrolere și sisteme embedded și folosește sintaxe specifice Python, mult mai ușoare de înțeles și mai intuitive decât limbajul specific Arduino ce este o adaptare a C++ și folosește alte sintaxe, folosindu-se de acolade pentru definirea funcțiilor și semicolane, sintaxe specifice limbajului din care derivă;
- MicroPython necesită mai multă memorie flash și memorie RAM pentru execuția programelor, fiind un descendent direct al Python-ului care este un limbaj interpretat, pe când adaptarea C++ a Arduino are o manageriere mai eficientă a memoriei, având la bază C++, un limbaj de programare compilat;
- MicroPython este un tip de limbaj de programare ‘high-level’, adică gradul de acces asupra tuturor dispozitivelor din componența microcontroler-ului este limitat, pe când adaptarea C++ a Arduino este un limbaj de programare ‘low-level’ și oferă utilizatorului un nivel de libertate mai mare asupra configurării diferitelor componente integrate în microcontroler;
- MicroPython beneficiază de o gamă largă de librării specifice și poate fi compatibil uneori cu anumite librării specifice Python, însă nu are disponibilitate la un număr de librării la fel de mare ca și C++-ul adaptat de Arduino, ce reprezintă practic un ecosistem în propriul sens și este în mod continuu reînnoit de comunitatea Arduino;
- MicroPython atrage dezvoltatori Python, mentori și începători ce preferă simplitatea și versatilitatea specifică acestui limbaj, pe când C++ al Arduino este mai atractiv pentru o audiență mai largă, precum entuziaștii de dispozitive electronice, cu un focus pe învățarea bazelor electronicii programabile;

- MicroPython este potrivit pentru prototipare rapidă a sistemelor și aplicații IoT, unde posibilitățile programării ‘high-level’ își pot arăta strălucirea, iar C++ al Arduino este ideal pentru aplicații de timp real, în care sunt obligatorii respectarea unor timpi preciși și scenarii în care controlul direct al hardware-ului pot fi critice.

În figurile de mai jos se pot observa diferențe în ceea ce privește modul de declarare a funcțiilor și a variabilelor scrise în cele două limbaje prezentate.

```
from machine import Pin
import time

# Initializam pinul 2 ca si iesire digitala
led = Pin(2, Pin.OUT) declararea, inițializarea și configurarea unui pin
aprins = 1;
stins = 0; } declararea și inițializarea unor variabile

numele funcției
def stingeSiAprindeLedul():
    led.value(aprins) # Porneste ledul
    time.sleep(1) # Asteapta 1 secunda
    led.value(stins) # Opreste ledul
    time.sleep(1) # Asteapta 1 secunda } corpul funcției

while True:
    stingeSiAprindeLedul() apelarea funcției
```

Fig 2.6 – Exemplu de declarare a funcțiilor, a pinilor și a variabilelor în MicroPython

```
const int ledPin = 13;
const int ledStins = 0;
const int ledAprins = 1; } declararea de variabile

void setup() {
    pinMode(ledPin, OUTPUT); declararea, inițializarea și configurarea unui pin
}

numele funcției
void aprindeSiStingeLed() {
    digitalWrite(ledPin, ledAprins);
    delay(1000);
    digitalWrite(ledPin, ledStins);
    delay(1000); } corpul funcției
}

void loop() {
    aprindeSiStingeLed(); apelarea funcției
}
```

Fig 2.6 – Exemplu de declarare a funcțiilor, a pinilor și a variabilelor în MicroPython

Astfel, putem face următoarele sesizări:

- în MicroPython se folosește simbolul ‘:’ înainte de scrierea corpului funcției, pe când în C++ este necesară utilizarea acoladelor înainte și după corpul funcției;
- în MicroPython nu este necesar să se pună punct și virgulă după fiecare linie de cod, pe când în C++ acest lucru este obligatoriu;
- în MicroPython poți declara o variabilă în orice parte din cod, atât timp cât ea nu este utilizată într-o funcție înainte de a fi declarată, pe când în C++ este necesar ca toate variabilele să fie declarate în începutul codului;
- sintaxele pentru funcții asupra hardware-ului și parametrii care pot fi incluși în aceste funcții diferă între cele două limbaje de programare, însă au același scop comun: configurarea sistemului în modul dorit de utilizator.

În concluzie, ambele limbaje de programare au punctele lor forte și sunt potrivite pentru dezvoltarea a diferite tipuri de aplicații și sisteme. Alegerea unuia dintre cele două depinde de factori precum: consumul de memorie al aplicației, familiaritatea cu limbajul de programare, cerințele proiectului de îndeplinit, librării valabile și mediul de programare preferat.

Capitolul 3

Modul de funcționare a întregului sistem

3.1. Diagrama întregului sistem

După cum se poate observa în figura 3.1, toate elementele prezentate până acum sunt incluse în construcția sistemului și fiecare dintre acestea joacă un rol important în funcționarea acestuia.

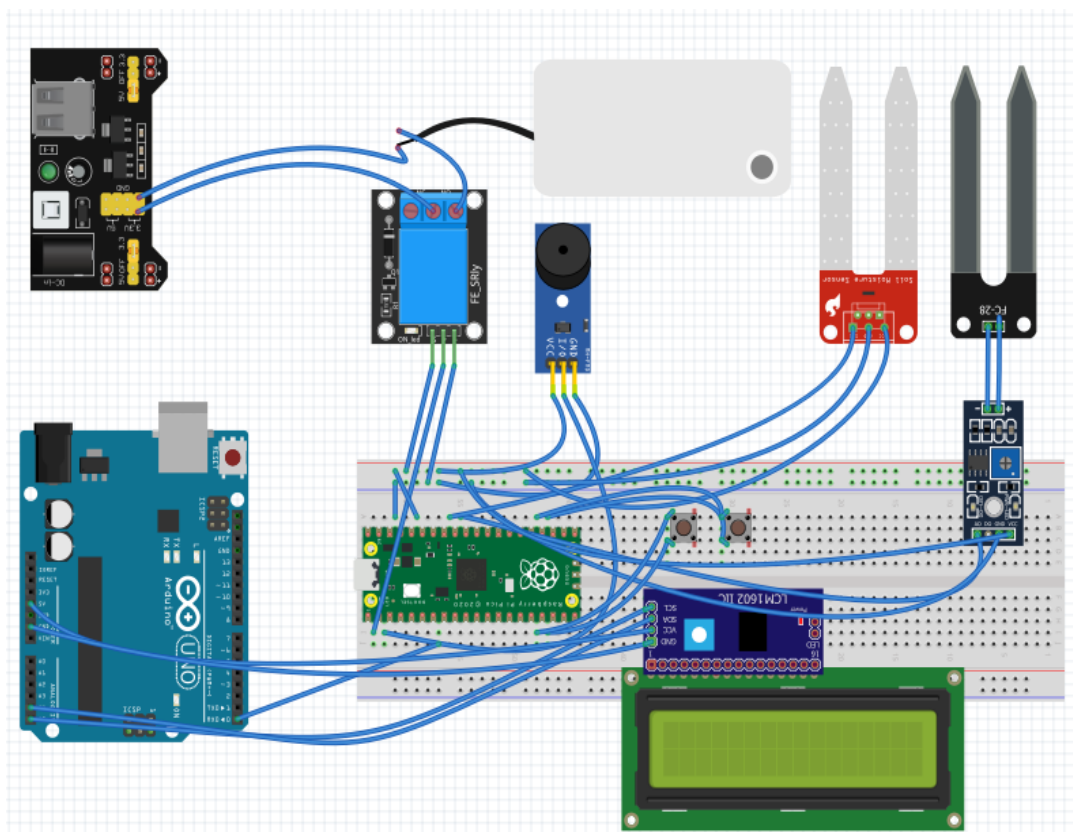


Fig. 3.1 – Diagrama de conectare a sistemului

Pe baza fiecărei informații preluate de către senzori, sistemul execută seturi de instrucțiuni și ia decizii cu privire la execuția funcțiilor precum cea de acționare a pompei de

apă în cazul umidității scăzute a solului sau cea de activare a buzzer-ului în cazul în care bazinul din care pompa își trage apa a fost golit.

3.2. Librăriile folosite în programarea sistemului

Pe partea de programare a Raspberry Pi Pico W în limbajul MicroPython s-au folosit următoarele librării:

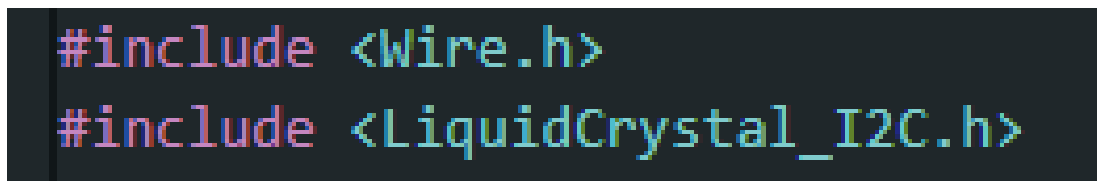
- din librăria machine s-au importat Pin, ADC, UART, componente ale librăriei machine ce permit inițializarea și configurarea pinilor, a modulului ADC și a protocolului de comunicație UART;
- din librăria picozero s-a importat Button, parte a librăriei picozero care face posibilă tratarea întreruperilor generate de apăsări, prin execuția subrutinelor specifice acestora;
- librăria time, ce dispune de funcții cu ajutorul căreia se generează întreruperi de timp în cadrul sistemului;
- librăria network, cu ajutorul căreia se poate realiza conectarea microcontroler-ului la o rețea de internet wireless;
- librăria ntptime, care stă la baza calculării și ulterior a afișării orei și datei locale pe ecranul LCD;
- din librăria blynklib se regăsește importat Blynk, ce permite transmisia de date și interfațarea web pe computere și telefoane a anumitor parametri din sistem, doar prin simpla conectare la internet a microcontroler-ului;

```
from machine import Pin, ADC, I2C
from picozero import Button
import time
import network
import ntptime
from blynklib import Blynk
```

Fig. 3.2 – Librării folosite în MicroPython

Dintre librăriile disponibile pentru ușurarea scrierii de cod pentru diverse dispozitive electronice în adaptarea de C++ a Arduino se utilizează următoarele:

- librăria LiquidCrystal_I2C care, prin funcțiile pe care le pune la dispoziție, face mult mai ușoară interfațarea și programarea dispozitivelor de tip LCD care funcționează în protocolul I2C;
- librăria Wire, ce se prezintă a fi extrem de utilă în cazul comunicării și programării cu orice dispozitiv ce funcționează în protocolul I2C.

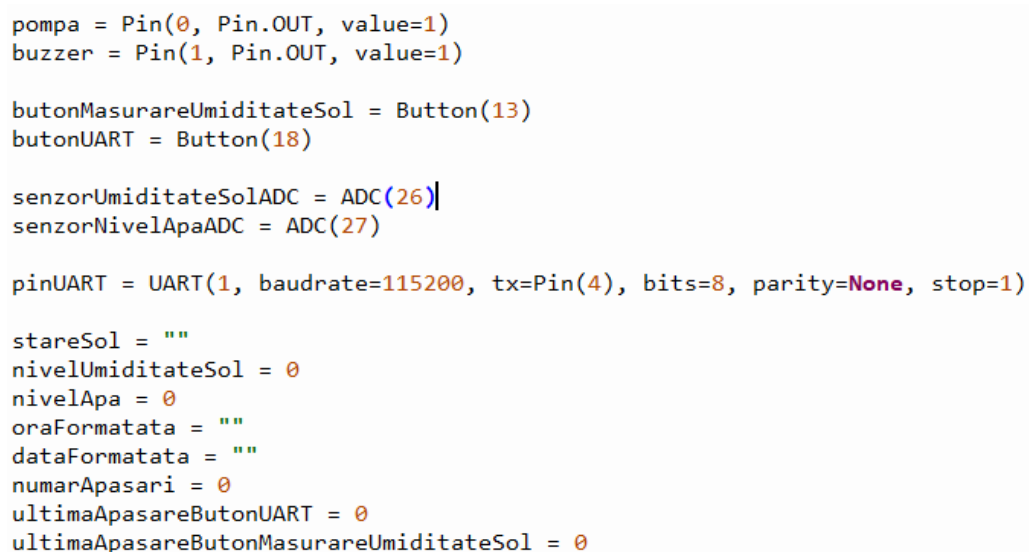


```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Fig 3.3 – Librării folosite în C++ adaptat Arduino

Utilizarea librăriilor poate ușura munca utilizatorului în ceea ce privește memorarea anumitor sintaxe sau a anumitor tipuri de configurări ale sistemului. De asemenea, acestea pot și să adauge noi caracteristici și posibilități de programare iar numărul de astfel de librării este unul semnificativ.

3.3. Variabile și funcții construite pentru programarea sistemului



```
pompa = Pin(0, Pin.OUT, value=1)
buzzer = Pin(1, Pin.OUT, value=1)

butonMasurareUmiditateSol = Button(13)
butonUART = Button(18)

senzorUmiditateSolADC = ADC(26)
senzorNivelApaADC = ADC(27)

pinUART = UART(1, baudrate=115200, tx=Pin(4), bits=8, parity=None, stop=1)

stareSol = ""
nivelUmiditateSol = 0
nivelApa = 0
oraFormatata = ""
dataFormatata = ""
numarApasari = 0
ultimaApasareButonUART = 0
ultimaApasareButonMasurareUmiditateSol = 0
```

Fig 3.4 – Inițializarea pinilor și variabilelor sistemului pe partea de MicroPython

- Secvența de cod prezentată în figura 3.4 constituie partea programului de declarare, inițializare și configurare a variabilelor și pinilor ce s-au folosit în realizarea sistemului. Astfel, putem observa că:
- pinul 0 reprezintă ieșirea digitală a releului ce comandă pompa de apă și este inițializată cu valoarea 1 logic pentru a nu fi pornită la execuția programului, acesta fiind pusă în funcțiune prin setarea sa în starea de 0 logic;
- pinul 1 reprezintă ieșirea digitală a modului buzzer și este inițializat cu valoarea 1 din același motiv ca și în cazul modului releu;
- pe pinii 13 și 18 sunt declarate butoane ce îndeplinesc anumite funcții în sistem;
- este inițializat canalul 1 de comunicație în protocolul UART, cu un baud rate de 115200, utilizând pinul 4 ca și pin de transmisie a informațiilor, informații ce conțin 8 biți și au bitul de stop în valoarea 1 logic;
- restul variabilelor reprezintă valori de care codul se folosește și care trebuie inițializate pentru buna execuție a programului;

```
def conectareInternet():
    wifi = network.WLAN(network.STA_IF)
    wifi.active(True)
    wifi.connect('StefanSimion', '12345678')
    pinUART.write("    Conexiune        in asteptare  ")
    time.sleep(2)
    while not wifi.isconnected():
        print("Se asteapta conexiunea...")
        time.sleep(2)
    else:
        print("Conectat la WiFi!")
        pinUART.write("    Conexiune        realizata    ")
        ntptime.settime()
        time.sleep(5)

conectareInternet()

BLYNK_AUTH_TOKEN = '15tuLdIAxL7G01Z3hXDbhoK3EZaURL1R'
BLYNK = Blynk(BLYNK_AUTH_TOKEN)
```

Fig. 3.5 – Funcția conectareInternet()

Funcția conectareInternet() este principala metodă folosită în dezvoltarea aplicației pentru a realiza conexiunea cu o rețea de internet wireless. Acesta rulează până ce reușește să

se conecteze la rețea, după care efectuează conectarea la aplicația Blynk, ce ne permite să interfașăm diverse date din sistem direct pe o pagină web și într-o aplicație mobilă.

```
def verificareUmiditateSol():
    global stareSol, nivelUmiditateSol
    nivelUmiditateSol = senzorUmiditateSolADC.read_u16()
    nivelUmiditateSol = nivelUmiditateSol >> 4
    nivelUmiditateSol = round(120 - (nivelUmiditateSol / 4095) * 100, 2)
    BLYNK.virtual_write(30, nivelUmiditateSol)
    print(f"Procentajul de umiditate al solului: {nivelUmiditateSol}%")
```

Fig. 3.6 – Funcția verificareUmiditateSol()

Această funcție realizează citirea de date de pe modulul de conversie analogic-digitală pentru senzorul de umiditate a solului, după care, pe baza unei formule de conversie obținută prin calibrarea senzorului în diferite tipuri de soluri în ceea ce privește umiditatea, o convertește într-un procentaj. Valoarea convertită este trimisă mai departe către Blynk pentru interfașare.

```
def stareUmiditateSol():
    global stareSol, nivelUmiditateSol
    if nivelUmiditateSol >= 85:
        stareSol = "UMIDITATE OPTIMA"
    elif nivelUmiditateSol >= 70:
        stareSol = "UMED"
    elif nivelUmiditateSol >= 40:
        stareSol = "USCAT"
    else:
        stareSol = "FOARTE USCAT"

    BLYNK.virtual_write(0, stareSol)
    print(f"Starea solului: {stareSol}")
```

Fig 3.7 – Funcția stareUmiditateSol()

Funcția stareUmiditateSol() este responsabilă de potrivirea procentajului de umiditate a solului într-un parametru de tip text care este și el trimis la rândul său către Blynk pentru interfașarea în aplicație.

Funcția din figura 3.8 este funcția ce se ocupă cu citirea și prelucrarea informațiilor cu privire la ora și data exactă. Funcția prezentată anterior, conectareInternet(), ce asigură conexiunea la internet, face posibilă și citirea unor parametri de oră și dată prin funcția

ntp.settime(), însă acestea returnează o oră ce este cu 3 ore înaintea orei locale a României, astfel că, pe baza unor algoritmi de decizie, se realizează conversia în ora locală, și se ține cont și de numărul de cifre al minutelor, secundelor, zilelor, orelor și lunilor și, în cazul în care acestea sunt formate dintr-o singură cifră, le adaugă un zero în față pentru a se asigura de faptul că acestea sunt afișate de fiecare dată într-un număr similar de caractere pentru o interfațare eficientă pe ecranul LCD. De asemenea, această funcție verifică și dacă se depășește formatul orar de 24 de ore și revine cu corecții asupra orei citite, astfel încât, atunci când ntp.settime() citește ora 22, programul să nu returneze 25, ci 1.

```
def afisareDataSiOra():
    global timpFormatat, dataFormatata
    timpLocal = time.localtime()
    an = str(timpLocal[0])
    luna = str(timpLocal[1])
    if len(str(luna)) < 2:
        luna = "0" + str(luna)
    zi = str(timpLocal[2])
    if len(str(zi)) < 2:
        zi = "0" + str(zi)
    if timpLocal[3] >= 21:
        ora = timpLocal[3]-21
        if len(str(ora)) < 2:
            ora = "0" + str(ora)
    else:
        ora = timpLocal[3] + 3
        if len(str(ora)) < 2:
            ora = "0" + str(ora)
    if timpLocal[4]<10:
        minut = "0" + str(timpLocal[4])
    else:
        minut = str(timpLocal[4])
    if timpLocal[5]<10:
        secunda = "0" + str(timpLocal[5])
    else:
        secunda = str(timpLocal[5])
    timpFormatat = str(ora) + ":" + str(minut) + ":" + str(secunda)
    dataFormatata = str(zi) + "." + str(luna) + "." + str(an)
    print(f"{timpFormatat} {dataFormatata}")
```

Fig. 3.8 – Funcția afisareDataSiOra()

```

def masurareVolumApa():
    global nivelApa
    nivelApa = senzorNivelApaADC.read_u16()
    nivelApa = nivelApa >> 4
    nivelApa = nivelApa / 4095
    if nivelApa < 0.05:
        buzzer.low()
        time.sleep(0.25)
        buzzer.high()
        time.sleep(0.25)
        buzzer.low()
        time.sleep(0.25)
        buzzer.high()
        time.sleep(0.25)
        buzzer.low()
        time.sleep(0.25)
        buzzer.high()
        time.sleep(0.25)
        buzzer.low()
        time.sleep(0.25)
        buzzer.high()
        time.sleep(0.25)
        print("Nivel apa: insuficient!!!")
        masurareVolumApa()
    print("Nivel apa: suficient")

```

Fig 3.9 – Funcția masurareNivelApa()

Funcția din figura 3.9 este responsabilă de convertirea valorii citite pe convertorul analogic-digital ce aparține senzorului de nivel al apei. Se realizează o mutare de 4 biți deoarece software-ul citește valoarea de la convertor ca fiind pe 16 biți, pe când aceasta dispune de o rezoluție de doar 12 biți. De asemenea, în cazul în care nivelul de apă scade în mod critic, funcția activează și dezactivează modulul buzzer odata la 250 de milisecunde, până ce intervine operatorul uman în reumplerea bazinului cu apă.

Atât timp cât variabila nivelApa se află sub valoarea de 0.05, funcția este executată la nesfârșit, până ce se depășește acest prag.

Funcția starePompa() este partea de cod ce decide dacă trebuie să fie pusă sau nu în funcțiune pompa de apă, considerându-se că aceasta nu este necesară până ce umiditatea solului

nu scade sub 70 de procente. De asemenea, această funcție transmite date către Blynk cu privire la starea pompei.

```
def starePompa():
    global nivelUmiditateSol, nivelApa
    if nivelUmiditateSol < 70:
        print("Pompa: PORNITA")
        BLYNK.virtual_write(2,"PORNITA")
        pompa.low()
        time.sleep(5)
        print("Pompa: OPRITA")
        BLYNK.virtual_write(2,"OPRITA")
        pompa.high()
        time.sleep(5)
    elif nivelUmiditateSol > 70:
        print("Pompa: OPRITA")
        BLYNK.virtual_write(2,"OPRITA")
        pompa.high()
```

Fig 3.10 – Funcția starePompa()

```
def transmisieUART():
    global timpFormatat, dataFormatata, nivelUmiditateSol, stareSol, numarApasari
    if numarApasari == 0:
        pinUART.write(f" {timpFormatat} {dataFormatata} ")
        print(f"{timpFormatat} - {dataFormatata}")
    elif numarApasari == 1:
        nivelUmiditateSol=int(nivelUmiditateSol)
        pinUART.write(f"Concentratia de apa din sol: {nivelUmiditateSol}%")
    elif numarApasari == 2:
        if nivelUmiditateSol >= 85:
            stareSol = "UMIDITATE OPTIMA"
            pinUART.write("Stare sol:      umiditate optima")
        elif nivelUmiditateSol >= 70:
            stareSol = "UMED"
            pinUART.write("Stare sol:      umed      ")
        elif nivelUmiditateSol >= 40:
            stareSol = "USCAT"
            pinUART.write("Stare sol:      uscat      ")
        else:
            stareSol = "FOARTE USCAT"
            pinUART.write("Stare sol:      foarte uscat  ")
```

Fig 3.11 – Funcția transmisieUART()

Funcția `transmisieUART()` este responsabilă de transmiterea datelor prin protocolul UART către microcontroler-ul Arduino Uno și este bazată pe valoarea variabilei `numarApasari`, variabilă ce este incrementată prin apăsarea butonului 18 care, atunci când este apăsat, execută subrutina de întrerupere prezentată în figura 3.12.

În figura 3.13 este prezentată subrutina de tratare a întreruperilor de la butonul 13 care, atunci când este apăsat, efectuează o funcție care realizează un set de măsurări a parametrilor achiziționați de către sistem prin intermediul senzorilor folosiți.

În figura 3.14 se prezintă modul de declarare a funcțiilor respective pentru întreruperea de pe pinii corespunzători.

```
def incrementareNumarApasari():
    global numarApasari, ultimaApasareButonUART
    apasareActualaButonUART = time.ticks_ms()
    numarApasari=numarApasari+1
    if numarApasari > 2:
        numarApasari = 0
    ultimaApasareButonUART = apasareActualaButonUART
```

Fig 3.12 – Funcția executată la întreruperea butonului 18

```
def masurareUmiditateSol():
    global ultimaApasareButonMasurareUmiditateSol
    apasareActualaButonMasurareUmiditateSol = time.ticks_ms()
    if time.ticks_diff(apasareActualaButonMasurareUmiditateSol, ultimaApasareButonMasurareUmiditateSol) > 300:
        afisareDataSiOra()
        stareUmiditateSol()
        verificareUmiditateSol()
        starePompa()
        ultimaApasareButonMasurareUmiditateSol = apasareActualaButonMasurareUmiditateSol
```

Fig 3.13 – Funcția executată la întreruperea butonului 13

Figura 3.15 prezintă bucla ‘while True’ ce are rolul de a executa funcțiile date la nesfârșit atât timp cât microcontroler-ul este alimentat și dispune de o conexiune la internet.

```
butonUART.when_pressed = incrementareNumarApasari
butonMasurareUmiditateSol.when_pressed = masurareUmiditateSol
```

Fig. 3.14 – Modul de declarare a funcțiilor subrutine ale întreruperilor

```

while True:
    afisareDataSiOra()
    verificareUmiditateSol()
    stareUmiditateSol()
    masurareNivelApa()
    starePompa()
    transmisieUART()
    BLYNK.run()
    print("\n")
    time.sleep(1)

```

Fig 3.15 – Bucla 'while True'

Pe partea de C++ adaptat la Arduino avem prezentată configurația sistemului în figura 3.16, în prima linie și în cadrul funcției setup().

```

LiquidCrystal_I2C ecranLCD(0x27, 16, 2);

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

    ecranLCD.init();
    ecranLCD.backlight();

    ecranLCD.setCursor(0, 0);
    ecranLCD.print("LCD");
    ecranLCD.setCursor(0, 1);
    ecranLCD.print("Pregatit");

    while (Serial.available()) {
        Serial.read();
    }
}

```

Fig 3.16 – Inițializarea variabilelor și a obiectului ecranLCD, funcția setup()

Practic, această secvență de cod inițializează modulul LCD, îi dă parametrul de adresă specific acestuia, 0x27, și îi declară rezoluția de 16 pe 2. În corpul funcției setup() se inițializează comunicația serială cu un baud rate de 115200 și se face și inițializarea ecranului și pornirea luminării sale. Liniile de cod ce urmează pregătesc un prim mesaj pentru a fi afișat pe ecranul LCD, urmând ca mai apoi să facă citire pe serial în cazul în care se detectează o recepție.

```
void loop() {
  if (Serial.available() > 0) {
    ecranLCD.clear();
    String mesaj = "";
    while (Serial.available()) {
      mesaj += char(Serial.read());
      delay(2);
    }
    ecranLCD.setCursor(0, 0);
    ecranLCD.print(mesaj.substring(0, 16));
    if (mesaj.length() > 16) {
      ecranLCD.setCursor(0, 1);
      ecranLCD.print(mesaj.substring(16, 32));
    }
    while (Serial.available()) {
      Serial.read();
    }
    delay(1000);
  }
}
```

Fig 3.17 – Funcția loop()

În figura 3.17 este prezentată funcția loop() a aplicației Arduino ce are un rol repetitiv. Scopul acestei funcții este de a curăța ecranul de informații la fiecare execuție a acesteia, de a citi date de pe serial și de a împărți fluxul de date pentru o afișare pe două linii cât mai eficientă.

La finalul funcției avem funcția ‘delay’ ce trece sistemul într-o stare de pauză timp de o secunda, pentru a oferi LCD-ului timp necesar să primească informația și să o afișeze.

În concluzie, modul de funcționare al întregului ansamblu este următorul:

- sistemul realizează întâi conexiunea la internet și la aplicația de interfațare web, Blynk;
- după ce sunt realizate conexiunile, sistemul preia un flux de date cu privire la data și ora curentă și o convertește în ora locală, ținând cont de toate situațiile posibile;
- se verifică nivelul procentajului de umiditate din solul plantei în care este instalat senzorul de umiditate a solului;
- pe baza valorii de umiditate în procente, solului îi este atribuită o stare: foarte uscat, uscat, umed sau umiditate optimă;
- se măsoară nivelul de apă din bazinul în care este instalată pompa de apă pentru a ne asigura că există resurse lichide în cazul acționării releului. În cazul în care nu există suficientă apă, funcția este repetată la nesfârșit până la intervenția operatorului uman, pentru a nu permite pompei să meargă în gol, fapt ce poate duce la defectarea acesteia;
- se ia o decizie dacă planta trebuie sau nu udată, pe baza procentajului de umiditate al solului pe care sistemul încearcă să îl mențină în jurul valorii de 70-80 de procente;
- se realizează transmisia UART către microcontroler-ul Arduino Uno iar dispozitivul Raspberry Pi Pico W este pus în starea de ‘somn’ pentru o secundă;
- Arduino Uno primește datele recepționate de la Raspberry Pi Pico W prin protocolul UART și le afișează pe ecranul LCD, după care intră și acesta în starea de ‘somn’ pentru o secundă;
- se reia toată execuția la nesfârșit, într-o buclă infinită.

3.4. Interfațarea de date pe web cu ajutorul platformei Blynk

Blynk reprezintă o platformă construită pentru a facilita dezvoltarea și administrarea aplicațiilor de tip IoT într-un mod suficient de simplu și de eficient. Aceasta oferă unelte și servicii pentru crearea, administrarea și monitorizarea dispozitivelor IoT, permițând utilizatorilor să construiască proiecte conectate între ele cu ușurință.

Cu ajutorul acestei platforme putem interfața:

- valori ale senzorilor;
- grafice ale evoluției valorilor în timp;

- căsuțe de text în care se poate scrie starea unor anumite dispozitive din componența sistemelor;

În cadrul sistemului prezentat, Blynk este utilizat pentru a interfața web o serie de parametri din program, astfel că putem monitoriza de la distanță:

- procentajul de umiditate a solului și variația sa în timp pentru a afla dacă sistemul își indeplinește sau nu sarcina;
- nivelul apei din bazin pentru a ști dacă acesta necesită reumplere;
- starea pompei, pentru a confirma faptul că sistemul este funcțional și pompa nu pornește decât după modul în care este programată.

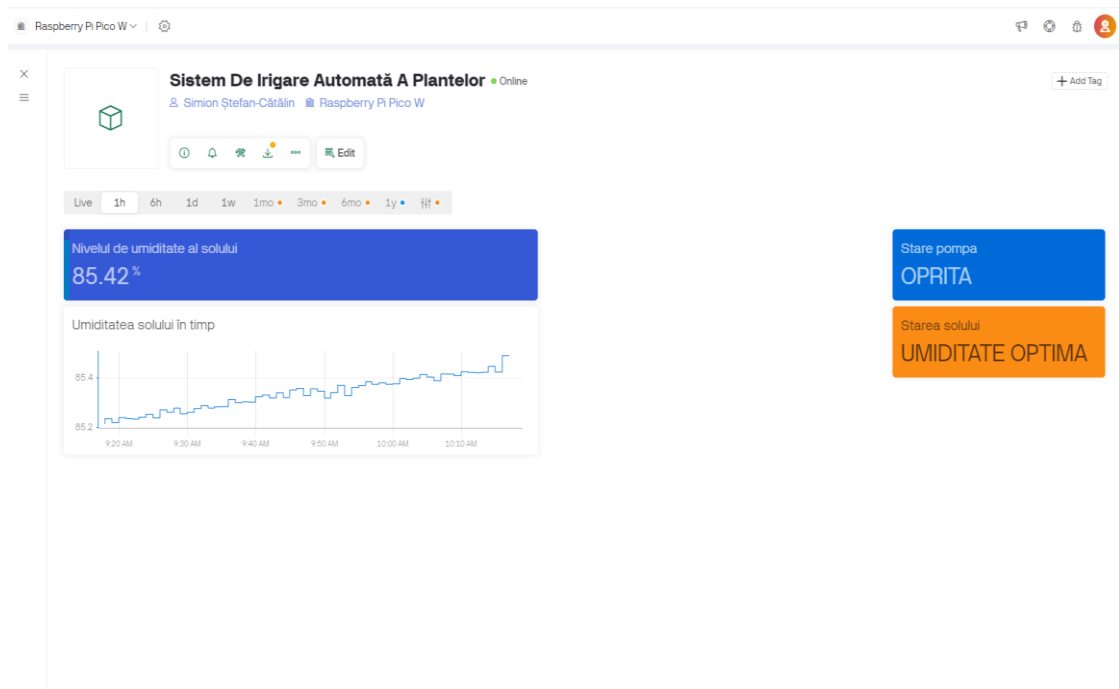


Fig. 3.18 – Interfața web Blynk a aplicației vizualizată de pe un dispozitiv tip desktop

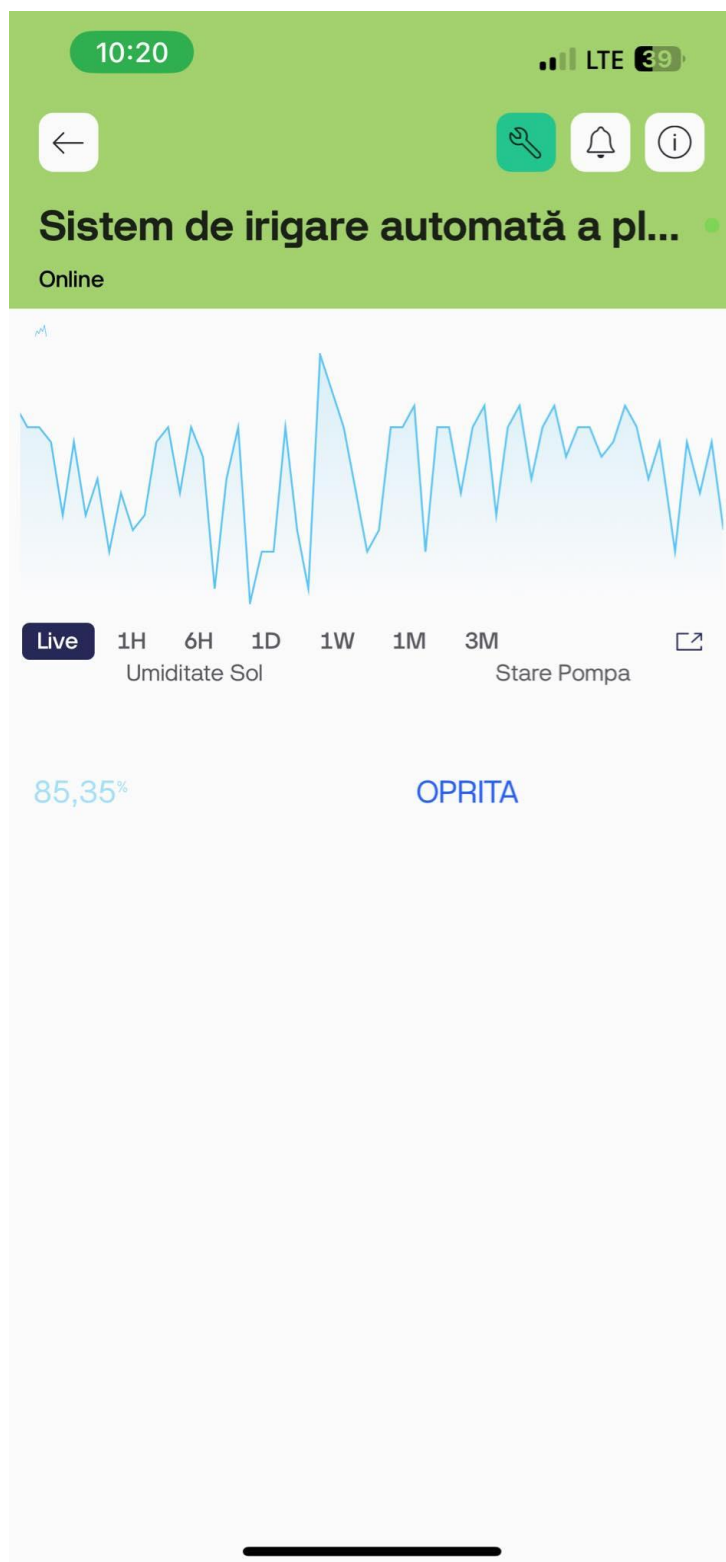


Fig. 3.19 – Interfața web Blynk a aplicației vizualizată de pe un dispozitiv mobil

Concluzii

În urma parcurgerii acestei lucrări, cititorul ar trebui să fie suficient de familiarizat cu termenii specifici atât ai domeniului microcontrolerelor și al programării cât și al electronicii și să aibă un punct de referință și o bază de cunoștințe pentru realizarea unui astfel de sistem.

Sistemul prezentat prezintă o soluție eficientă ce are un cost de producție destul de redus pentru funcția pe care o îndeplinește, iar posibilitatea de monitorizare a acestuia prin internet, fie pe computer sau pe telefon, la doar un click distanță, îl recomandă ca a fi unul avansat din punct de vedere tehnologic în ceea ce reprezintă achiziția de date în sistemele de tip IoT.

Realizarea unui astfel de sistem nu vine la pachet cu un cost redus, însă necesită un nivel de răbdare pentru a înțelege tot ceea ce se întâmplă în spatele procesului, mai ales în situații de depanare a unor erori ce par imposibil de remediat.

Însă, ceea ce stă la baza acestui proiect este tocmai principiul de încercare și eșec, sute de compilări de cod și de schimbări de variabile, până ce a ajuns în punctul în care a fost prezentat în aceste rânduri.

O astfel de aplicație vine la pachet și cu un set de dezavantaje:

- costurile reduse își spun cuvântul, astfel că se recomandă ca senzorii folosiți să fie schimbați odată la o perioadă de timp, deoarece stau în medii umede unde se degradează;
- datorită degradării acestora, senzorii își pierd din acuratețea de conversie analogică-digitală iar datele pe care aceștia le transmit nu vor mai fi la fel de actuale;
- programarea și construirea acestuia necesită un anumit nivel de abilități tehnice în electronice și programare;
- asigurarea faptului că toate componentele funcționează și se comportă cum trebuie poate reprezenta un proces dificil;

Înafara acestor dezavantaje, un astfel de sistem realizat piesă cu piesă și linie cu linie reprezintă o experiență excelentă de învățare, însă, planificarea și luarea în considerare în primă fază a acestor dezavantaje este cel mai important lucru de făcut. Dintr-un punct de vedere comercial, acest sistem nu ar putea să se compare vreodata cu un sistem deja disponibil pe piață.

Anexe

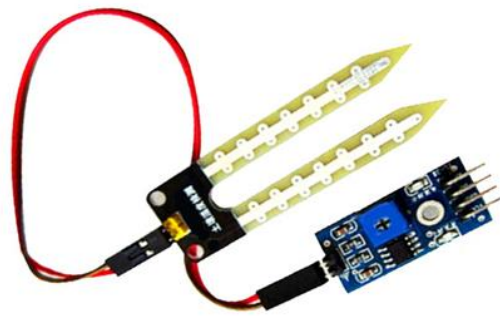


Fig. 1.1 - Modulul senzor de umiditate a solului

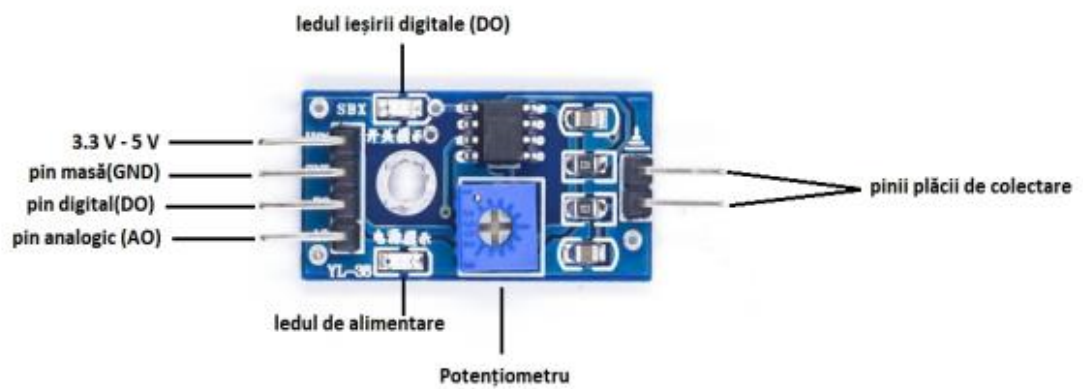


Fig. 1.2 – Modulul comparator cu potențiomtru

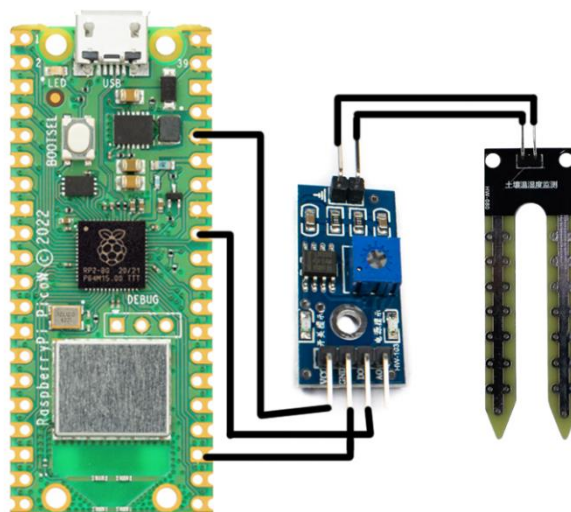


Fig. 1.3 – Schema electrică a ansamblului senzor – microcontroler Raspberry Pico W

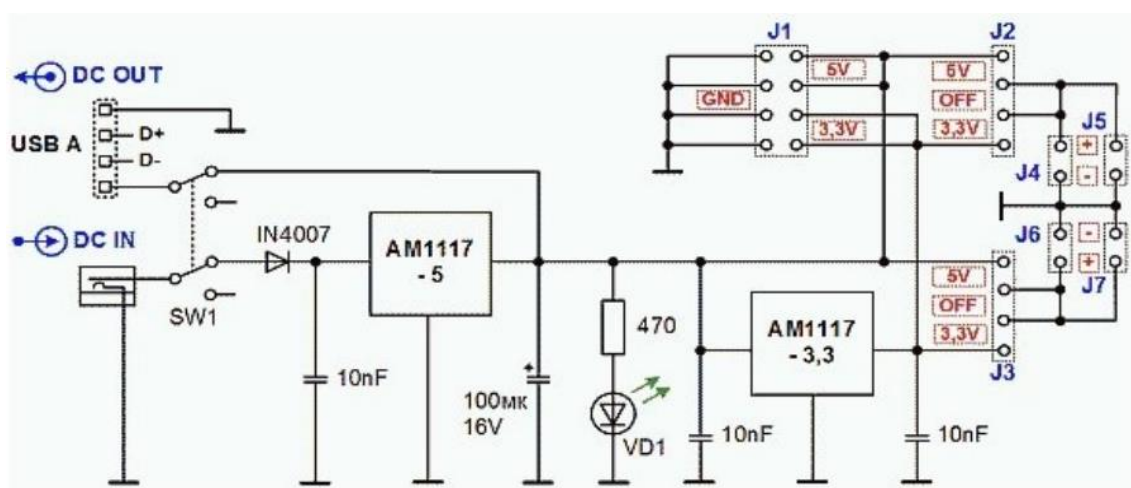


Fig. 1.4 – Schema circuitului modului sursă de alimentare externă

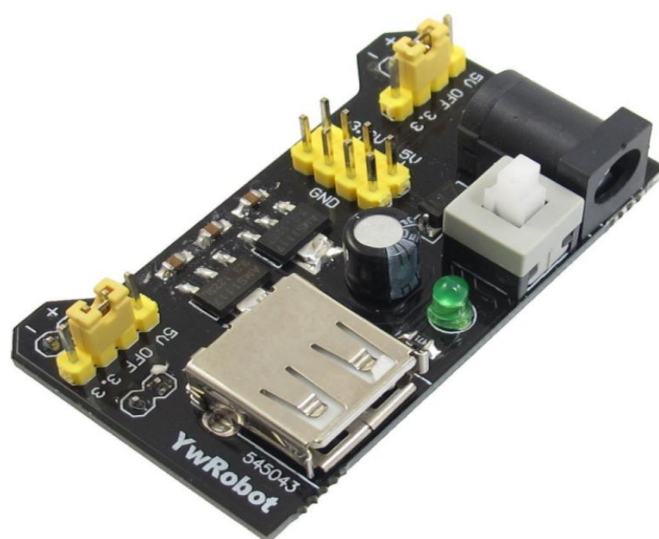


Fig. 1.5 – Modulul sursă de alimentare externă



Fig. 1.6 – Pompă de apă submersibilă

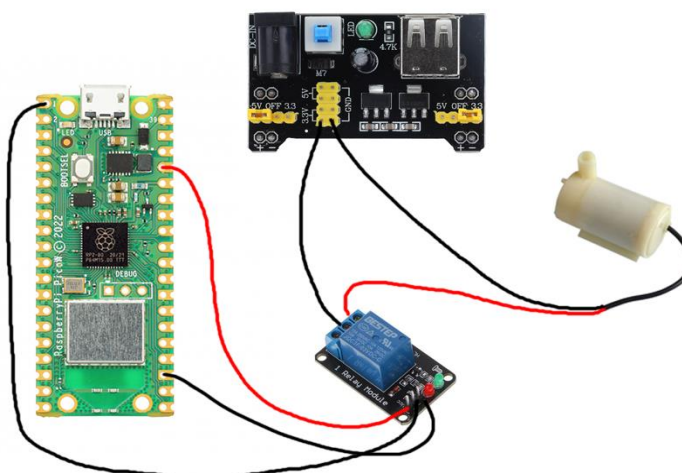


Fig. 1.7 – Schema electrică a ansamblului pompă–modul releu–modul de alimentare externă–microcontroler

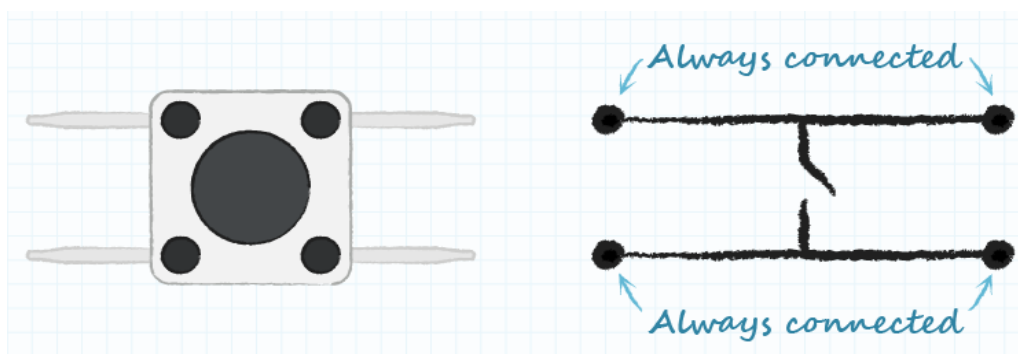


Fig. 1.8 – Construcția modului buton

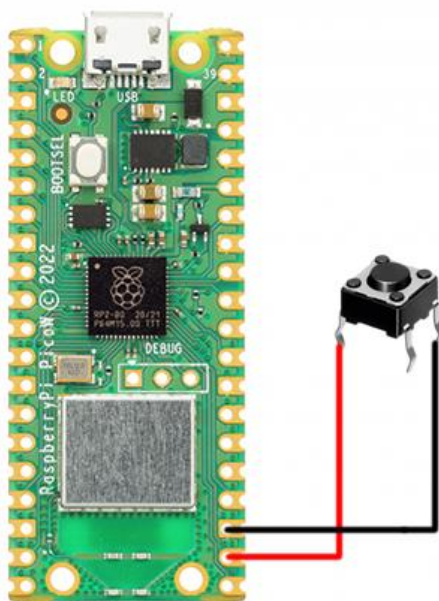


Fig 1.9 – Conexiunea butonului cu microcontroler-ul

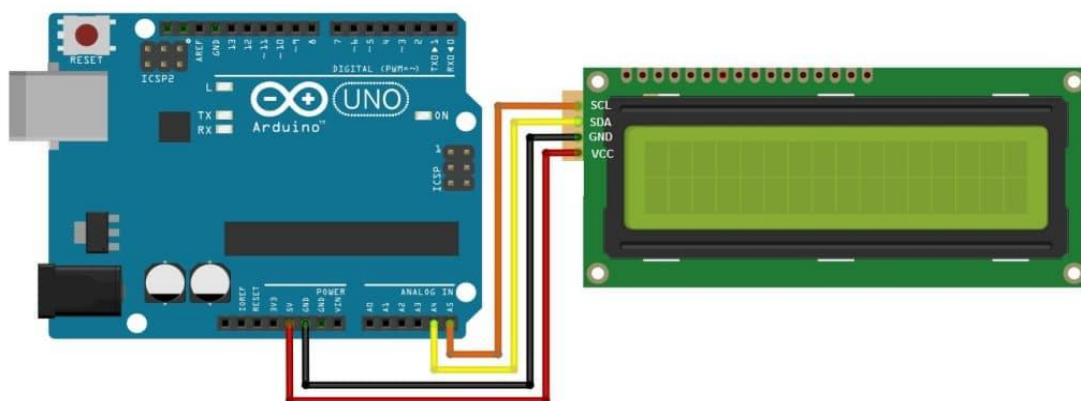


Fig. 1.10 – Schemba electrică dintre microcontroler și modulul ecran LCD

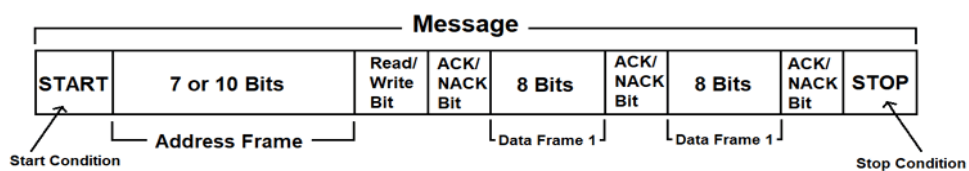


Fig. 1.11 – Formatul unui mesaj transmis prin protocolul I2C



Fig. 1.12 – Senzorul de nivel al lichidului

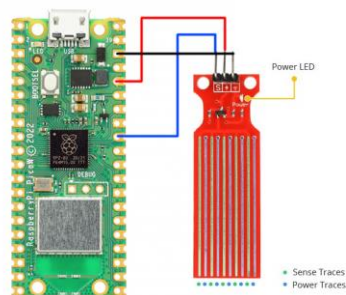


Fig. 1.13 – Conectarea senzorului de nivel al lichidelor la microcontroler

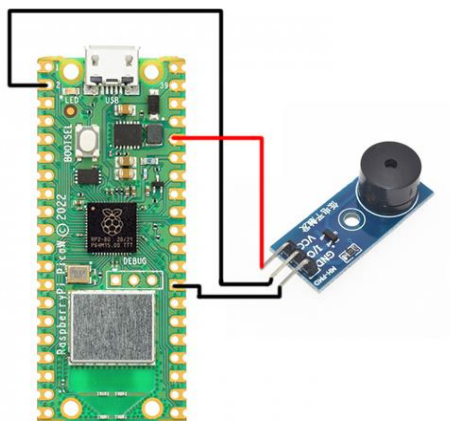


Fig 1.14 – Conectarea modulului buzzer MH-FMD la microcontroler

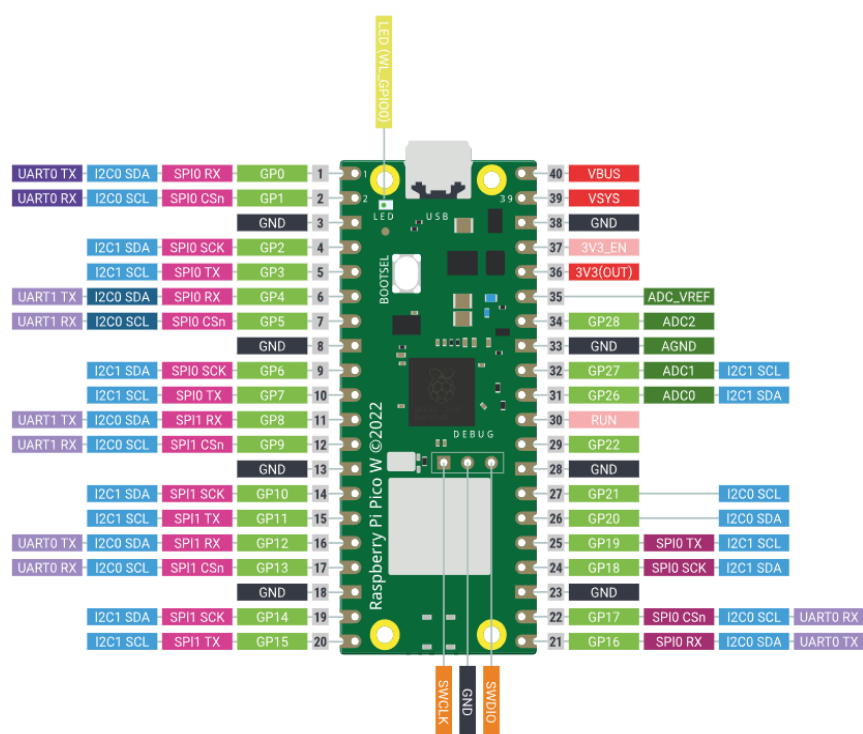


Fig. 1.15 – Diagrama pinilor de pe placa de dezvoltare Raspberry Pi Pico W

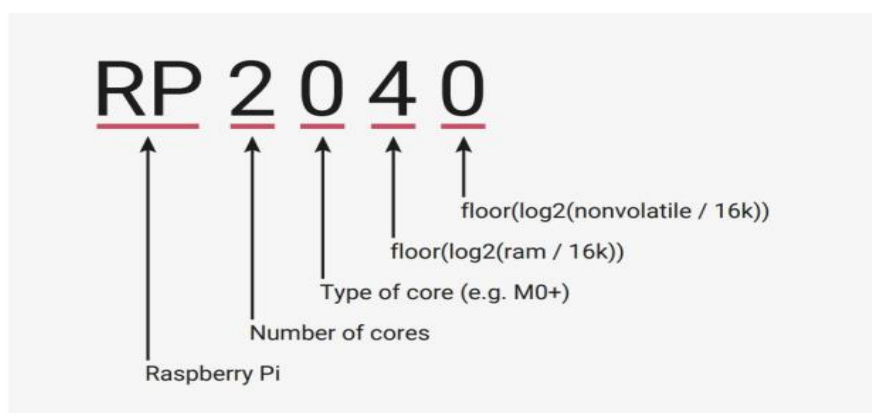


Fig. 1.16 – Semnificația acronimului RP2040

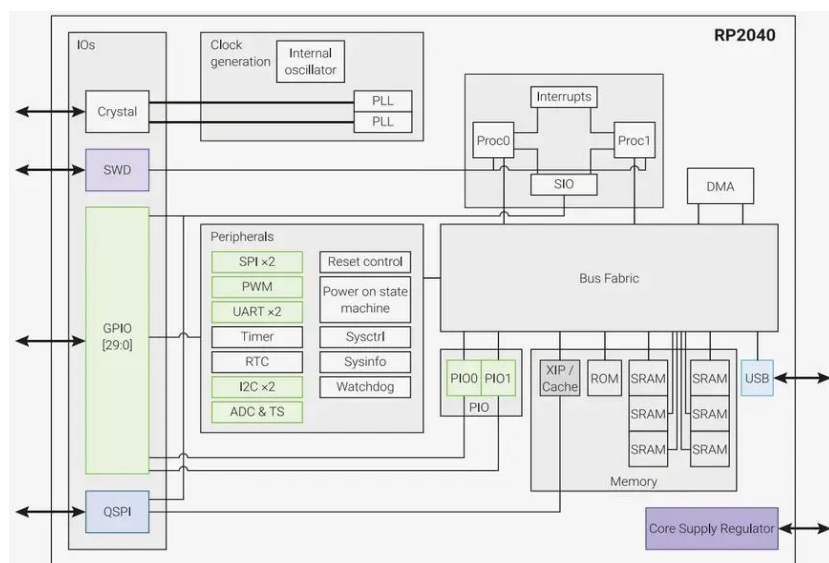


Fig. 1.17 – Arhitectura hardware a cipului RP2040

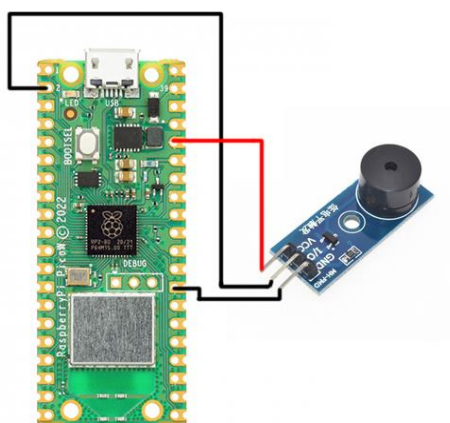


Fig 1.14 – Conectarea modulului buzzer MH-FMD la microcontroler

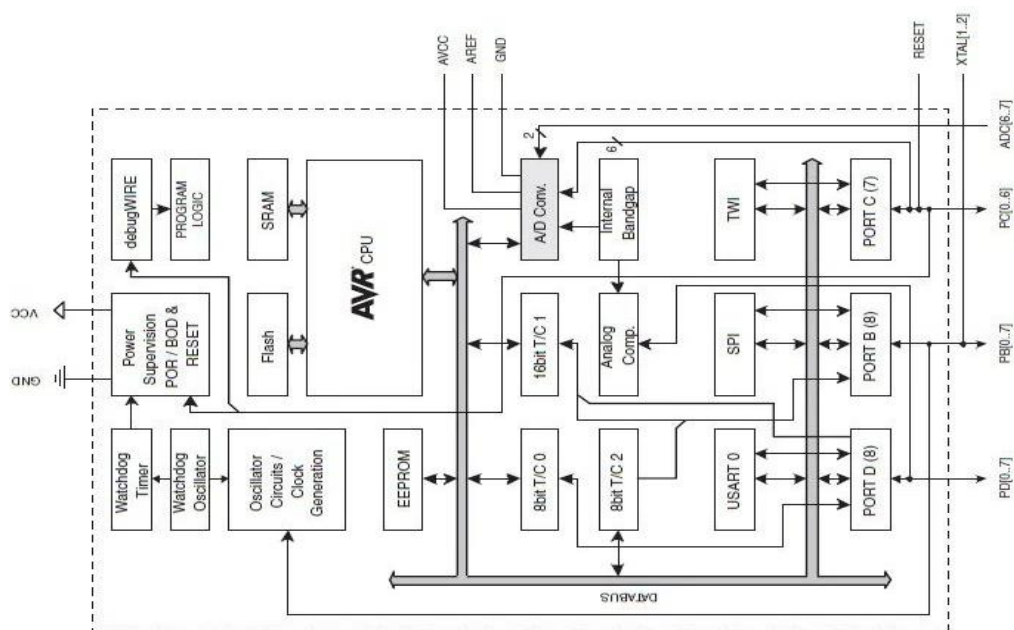


Fig. 1.19 – Arhitectura hardware a cipului ATmega328P

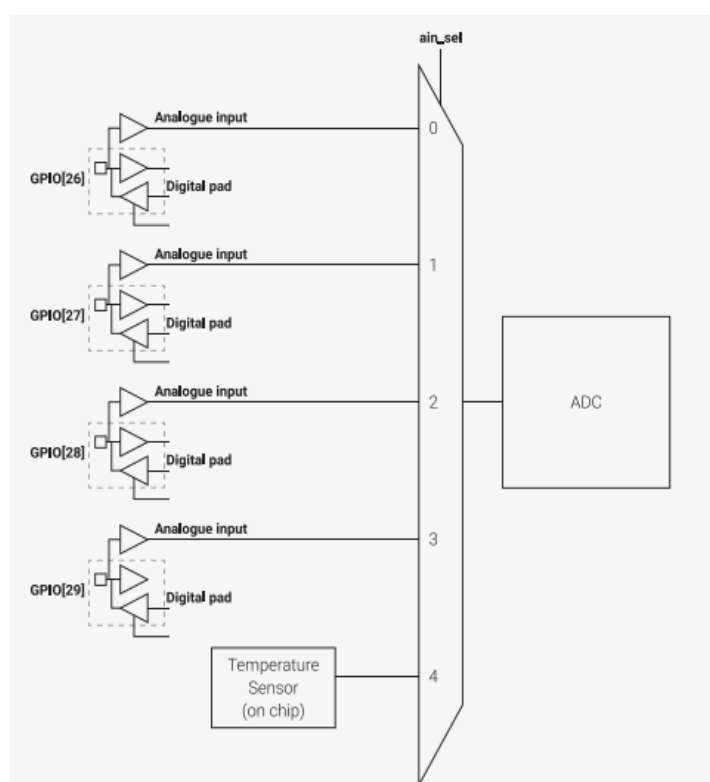


Fig 1.20 – Schema bloc a modului ADC

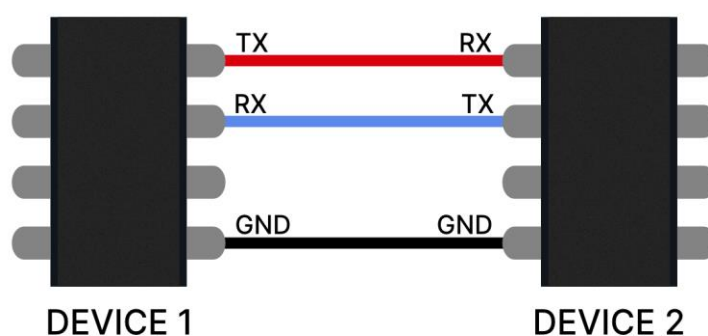


Fig. 1.21 – Comunicația UART, conexiunile necesare între dispozitive

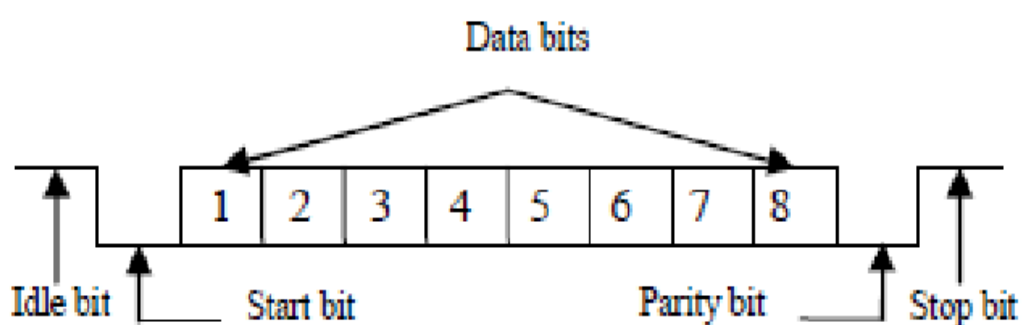


Fig. 1.22 – Formatul unui mesaj transmis în protocolul UART, cu 8 biți pentru informație

```
from machine import Pin
import time

# Initializam pinul 2 ca si iesire digitala
led = Pin(2, Pin.OUT)

while True:
    led.high() # Porneste ledul
    time.sleep(1) # Asteapta 1 secunda
    led.low() # Opreste ledul
    time.sleep(1) # Asteapta 1 secunda
```

Fig. 2.1 – Exemplu de cod scris în MicroPython



Fig 2.2 – Logo-ul MicroPython

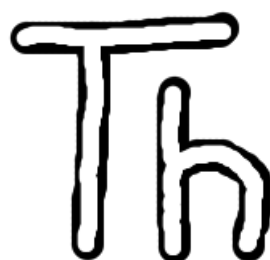


Fig 2.3 – Logo-ul Thonny IDE



Fig 2.3 – Logo-ul Python

```
const int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Fig 2.4 – Exemplu de cod pentru Arduino Uno



Fig 2.5 – Logo-ul Arduino

```
from machine import Pin
import time

# Initializam pinul 2 ca si iesire digitala
led = Pin(2, Pin.OUT) declararea, inițializarea și configurarea unui pin
aprins = 1;
stins = 0; } declararea și inițializarea unor variabile

numele funcției
def stingeSiAprindeLedul():
    led.value(aprins) # Porneste ledul
    time.sleep(1) # Asteapta 1 secunda
    led.value(stins) # Opreste ledul
    time.sleep(1) # Asteapta 1 secunda } corpul funcției

while True:
    stingeSiAprindeLedul() apelarea funcției
```

Fig 2.6 – Exemplu de declarare a funcțiilor, a pinilor și a variabilelor în MicroPython


```

const int ledPin = 13;
const int ledStins = 0;
const int ledAprins = 1;
} declararea de variabile

void setup() {
    pinMode(ledPin, OUTPUT); declararea, inițializarea și configurarea unui pin
}

    numele funcției
void aprindeSiStingeLed() {
    digitalWrite(ledPin, ledAprins);
    delay(1000);
    digitalWrite(ledPin, ledStins);
    delay(1000);
} } corpul funcției

void loop() {
    aprindeSiStingeLed(); apelarea funcției
}

```

Fig 2.6 – Exemplu de declarare a funcțiilor, a pinilor și a variabilelor în MicroPython

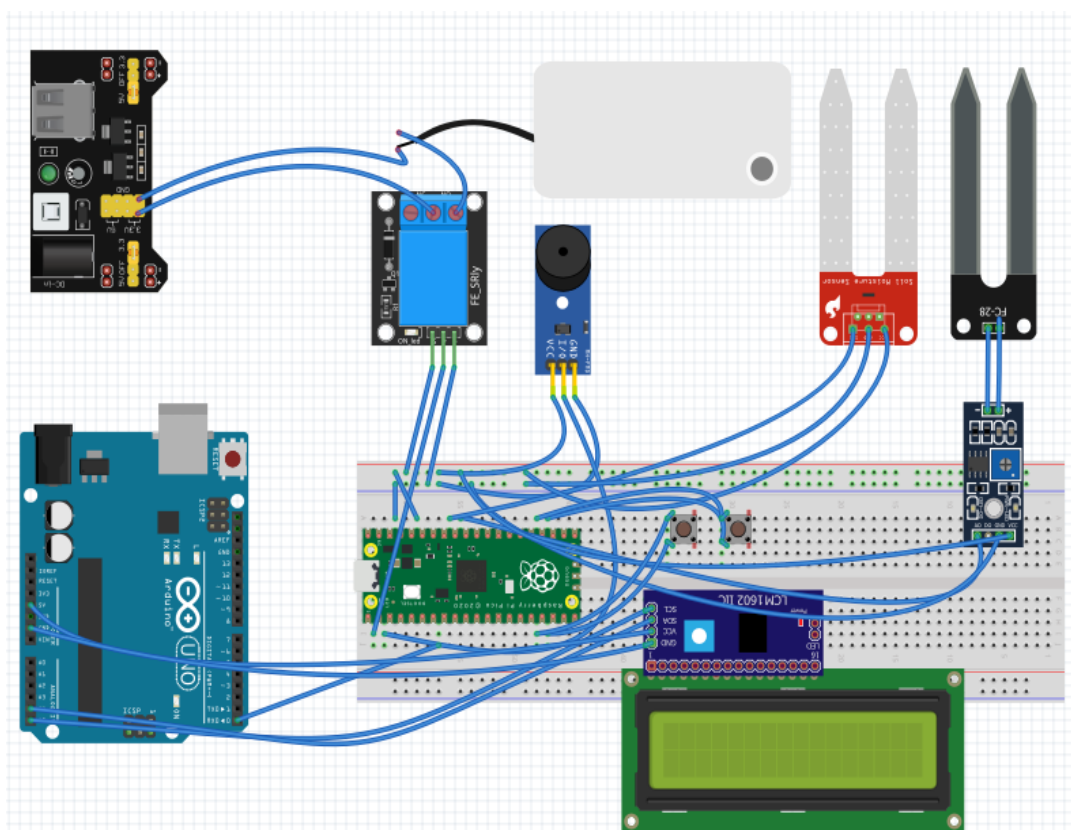


Fig. 3.1 – Diagrama de conectare a sistemului

```

from machine import Pin, ADC, I2C
from picozero import Button
import time
import network
import ntptime
from blynklib import Blynk

```

Fig. 3.2 – Librării folosite în MicroPython

```

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

```

Fig 3.3 – Librării folosite în C++ adaptat Arduino

```

pompa = Pin(0, Pin.OUT, value=1)
buzzer = Pin(1, Pin.OUT, value=1)

butonMasurareUmiditateSol = Button(13)
butonUART = Button(18)

senzorUmiditateSolADC = ADC(26)
senzorNivelApaADC = ADC(27)

pinUART = UART(1, baudrate=115200, tx=Pin(4), bits=8, parity=None, stop=1)

stareSol = ""
nivelUmiditateSol = 0
nivelApa = 0
oraFormatata = ""
dataFormatata = ""
numarApasari = 0
ultimaApasareButonUART = 0
ultimaApasareButonMasurareUmiditateSol = 0

```

Fig 3.4 – Inițializarea pinilor și variabilelor sistemului pe partea de MicroPython

```

def conectareInternet():
    wifi = network.WLAN(network.STA_IF)
    wifi.active(True)
    wifi.connect('StefanSimion', '12345678')
    pinUART.write("    Conexiune        in asteptare ")
    time.sleep(2)
    while not wifi.isconnected():
        print("Se asteapta conexiunea...")
        time.sleep(2)
    else:
        print("Conectat la WiFi!")
        pinUART.write("    Conexiune        realizata ")
        ntptime.settime()
        time.sleep(5)

conectareInternet()

BLYNK_AUTH_TOKEN = '15tuLdIAxL7G01Z3hXDbhoK3EZaURl1R'
BLYNK = Blynk(BLYNK_AUTH_TOKEN)

```

Fig. 3.5 – Funcția conectareInternet()

```

def verificareUmiditateSol():
    global stareSol, nivelUmiditateSol
    nivelUmiditateSol = senzoruUmiditateSolADC.read_u16()
    nivelUmiditateSol = nivelUmiditateSol >> 4
    nivelUmiditateSol = round(120 - (nivelUmiditateSol / 4095) * 100, 2)
    BLYNK.virtual_write(30, nivelUmiditateSol)
    print(f"Procentajul de umiditate al solului: {nivelUmiditateSol}%")

```

Fig. 3.6 – Funcția verificareUmiditateSol()

```

def stareUmiditateSol():
    global stareSol, nivelUmiditateSol
    if nivelUmiditateSol >= 85:
        stareSol = "UMIDITATE OPTIMA"
    elif nivelUmiditateSol >= 70:
        stareSol = "UMED"
    elif nivelUmiditateSol >= 40:
        stareSol = "USCAT"
    else:
        stareSol = "FOARTE USCAT"

    BLYNK.virtual_write(0, stareSol)
    print(f"Starea solului: {stareSol}")

```

Fig 3.7 – Funcția stareUmiditateSol()

```

def afisareDataSiOra():
    global timpFormatat, dataFormatata
    timpLocal = time.localtime()
    an = str(timpLocal[0])
    luna = str(timpLocal[1])
    if len(str(luna)) < 2:
        luna = "0" + str(luna)
    zi = str(timpLocal[2])
    if len(str(zi)) < 2:
        zi = "0" + str(zi)
    if timpLocal[3] >= 21:
        ora = timpLocal[3]-21
        if len(str(ora)) < 2:
            ora = "0" + str(ora)
    else:
        ora = timpLocal[3] + 3
        if len(str(ora)) < 2:
            ora = "0" + str(ora)
    if timpLocal[4]<10:
        minut = "0" + str(timpLocal[4])
    else:
        minut = str(timpLocal[4])
    if timpLocal[5]<10:
        secunda = "0" + str(timpLocal[5])
    else:
        secunda = str(timpLocal[5])
    timpFormatat = str(ora) + ":" + str(minut) + ":" + str(secunda)
    dataFormatata = str(zi) + "." + str(luna) + "." + str(an)
    print(f"{timpFormatat} {dataFormatata}")

```

Fig. 3.8 – Funcția afisareDataSiOra()

```

def masurareVolumApa():
    global nivelApa
    nivelApa = senzorNivelApaADC.read_u16()
    nivelApa = nivelApa >> 4
    nivelApa = nivelApa / 4095
    if nivelApa < 0.05:
        buzzer.low()
        time.sleep(0.25)
        buzzer.high()
        time.sleep(0.25)
        buzzer.low()
        time.sleep(0.25)
        buzzer.high()
        time.sleep(0.25)
        buzzer.low()
        time.sleep(0.25)
        buzzer.high()
        time.sleep(0.25)
        buzzer.low()
        time.sleep(0.25)
        buzzer.high()
        time.sleep(0.25)
        print("Nivel apa: insuficient!!!")
        masurareVolumApa()
    print("Nivel apa: suficient")

```

Fig 3.9 – Funcția masurareNivelApa()

```

def starePompa():
    global nivelUmiditateSol, nivelApa
    if nivelUmiditateSol < 70:
        print("Pompa: PORNITA")
        BLYNK.virtual_write(2,"PORNITA")
        pompa.low()
        time.sleep(5)
        print("Pompa: OPRITA")
        BLYNK.virtual_write(2,"OPRITA")
        pompa.high()
        time.sleep(5)
    elif nivelUmiditateSol > 70:
        print("Pompa: OPRITA")
        BLYNK.virtual_write(2,"OPRITA")
        pompa.high()

```

Fig 3.10 – Funcția starePompa()

```

def transmisieUART():
    global timpFormatat, dataFormatata, nivelUmiditateSol, stareSol, numarApasari
    if numarApasari == 0:
        pinUART.write(f"      {timpFormatat}      {dataFormatata}  ")
        print(f"{timpFormatat} - {dataFormatata}")
    elif numarApasari == 1:
        nivelUmiditateSol=int(nivelUmiditateSol)
        pinUART.write(f"Concentratia de apa din sol: {nivelUmiditateSol}%")
    elif numarApasari == 2:
        if nivelUmiditateSol >= 85:
            stareSol = "UMIDITATE OPTIMA"
            pinUART.write("Stare sol:      umiditate optima")
        elif nivelUmiditateSol >= 70:
            stareSol = "UMED"
            pinUART.write("Stare sol:      umed      ")
        elif nivelUmiditateSol >= 40:
            stareSol = "USCAT"
            pinUART.write("Stare sol:      uscat      ")
        else:
            stareSol = "FOARTE USCAT"
            pinUART.write("Stare sol:      foarte uscat  ")

```

Fig 3.11 – Funcția transmisieUART()

```
def incrementareNumarApasari():
    global numarApasari, ultimaApasareButonUART
    apasareActualaButonUART = time.ticks_ms()
    numarApasari=numarApasari+1
    if numarApasari > 2:
        numarApasari = 0
    ultimaApasareButonUART = apasareActualaButonUART
```

Fig 3.12 – Funcția executată la întreruperea butonului 18

```
def masurareUmiditateSol():
    global ultimaApasareButonMasurareUmiditateSol
    apasareActualaButonMasurareUmiditateSol = time.ticks_ms()
    if time.ticks_diff(apasareActualaButonMasurareUmiditateSol, ultimaApasareButonMasurareUmiditateSol) > 300:
        afisareDataSiOra()
        stareUmiditateSol()
        verificareUmiditateSol()
        starePompa()
        ultimaApasareButonMasurareUmiditateSol = apasareActualaButonMasurareUmiditateSol
```

Fig 3.13 – Funcția executată la întreruperea butonului 13

```
butonUART.when_pressed = incrementareNumarApasari
butonMasurareUmiditateSol.when_pressed = masurareUmiditateSol
```

Fig. 3.14 – Modul de declarare a funcțiilor subrutine ale întreruperilor

```
while True:
    afisareDataSiOra()
    verificareUmiditateSol()
    stareUmiditateSol()
    masurareNivelApa()
    starePompa()
    transmisieUART()
    BLYNK.run()
    print("\n")
    time.sleep(1)
```

Fig 3.15 – Bucla 'while True'

```
LiquidCrystal_I2C ecranLCD(0x27, 16, 2);

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

  ecranLCD.init();
  ecranLCD.backlight();

  ecranLCD.setCursor(0, 0);
  ecranLCD.print("LCD");
  ecranLCD.setCursor(0, 1);
  ecranLCD.print("Pregatit");

  while (Serial.available()) {
    Serial.read();
  }
}
```

Fig 3.16 – Inițializarea variabilelor și a obiectului ecranLCD, funcția setup()


```
void loop() {  
  if (Serial.available() > 0) {  
    ecranLCD.clear();  
    String mesaj = "";  
    while (Serial.available()) {  
      mesaj += char(Serial.read());  
      delay(2);  
    }  
    ecranLCD.setCursor(0, 0);  
    ecranLCD.print(mesaj.substring(0, 16));  
    if (mesaj.length() > 16) {  
      ecranLCD.setCursor(0, 1);  
      ecranLCD.print(mesaj.substring(16, 32));  
    }  
    while (Serial.available()) {  
      Serial.read();  
    }  
    delay(1000);  
  }  
}
```

Fig 3.17 – Funcția loop()

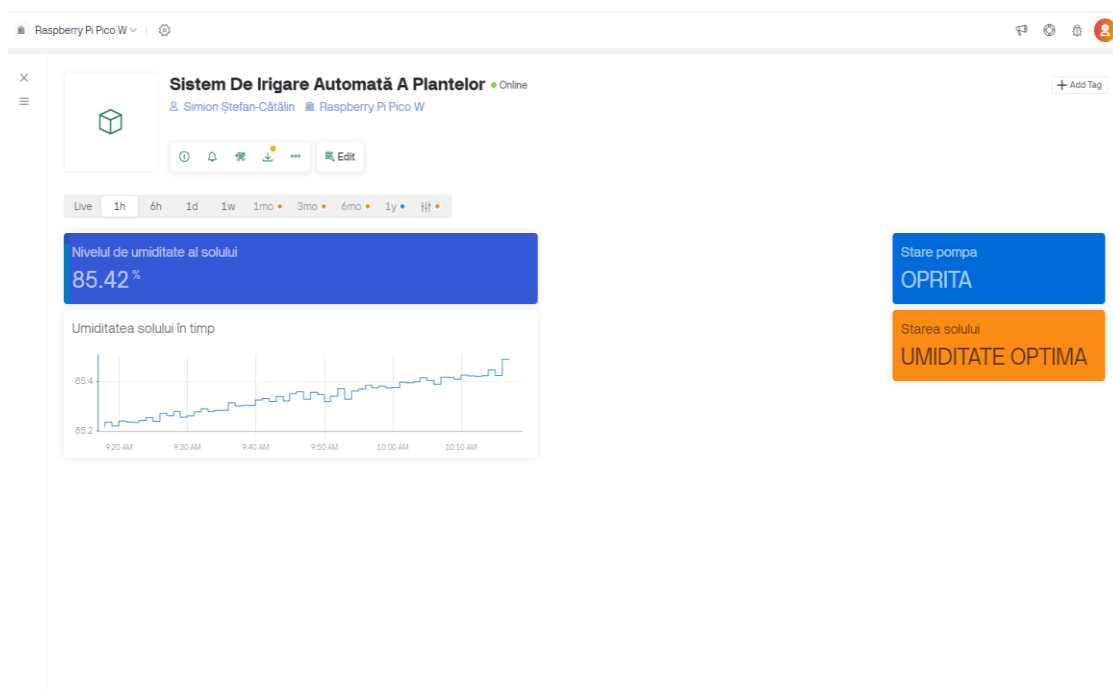


Fig. 3.18 – Interfața web Blynk a aplicației vizualizată de pe un dispozitiv tip desktop

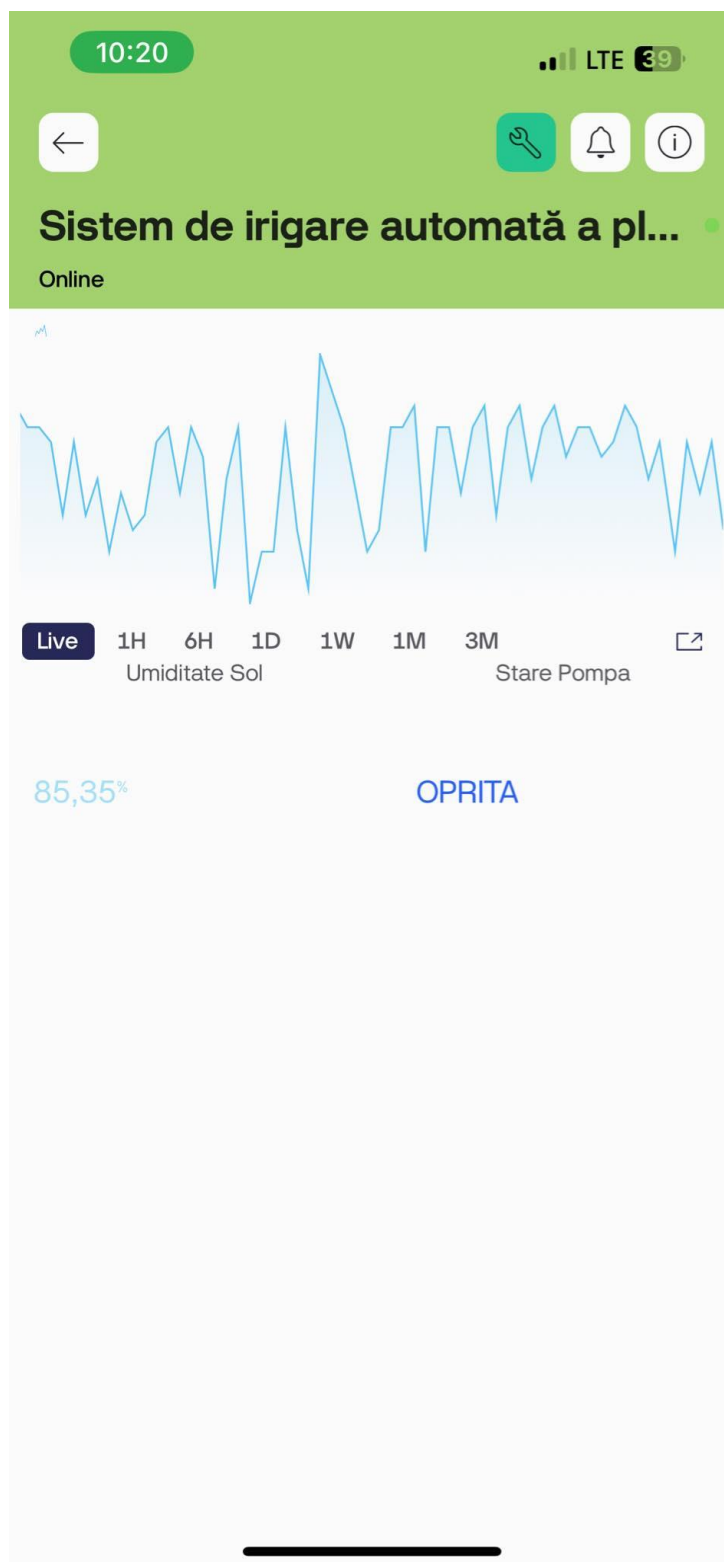


Fig. 3.19 – Interfața web Blynk a aplicației vizualizată de pe un dispozitiv mobil

Bibliografie

1. <https://kunkune.co.uk/blog/understanding-the-basics-what-is-a-relay-module/>
2. <https://lastminuteengineers.com/i2c-lcd-arduino-tutorial/>
3. <https://docs.arduino.cc/learn/communication/wire/>
4. <https://lastminuteengineers.com/water-level-sensor-arduino-tutorial/>
5. <https://handsontec.com/index.php/product/active-buzzer-module-low-level-trigger/>
6. <https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf>
7. https://mytoolbox.tech/tec/en/2021/12/27/rspi_pico_adc_mpython_e/
8. <https://electrocredible.com/raspberry-pi-pico-serial-uart-micropython/>
9. <https://dev.to/shilleh/easily-create-an-iot-app-with-blynk-and-raspberry-pi-pico-w-3o72>
10. <https://projects.raspberrypi.org/en/projects/get-started-pico-w/2>