Achizitii de Date
Laboratorul 1: Pulse Width Modulation

Dan Novischi

1. Introducere

Scopul acestui laborator îl reprezintă utilizarea tehnicii de modularea a semnalelor in latime (en. Pulse Width Modulation) pentru a varia independent viteza unor motoare de curent continuu folosind Arduino si TinkerCAD.

2. Modularea Semnalelor in Latime / Pulse Width Modulation

PWM (Pulse Width Modulation) este o tehnică folosită pentru a varia în mod controlat tensiunea dată unui dispozitiv electronic. Această metodă schimbă foarte rapid tensiunea oferită dispozitivului respectiv din ON în OFF și invers (treceri rapide din HIGH în LOW, de exemplu 5V - 0V). Raportul dintre perioada de timp corespunzătoare valorii ON și perioada totală dintr-un ciclu ON-OFF se numește factor de umplere (duty cycle) și reprezintă, în medie, tensiunea pe care o va primi dispozitivul electronic. Astfel, se pot controla circuite analogice din domeniul digital. În cazul unui motor, căruia i se aplică un semnal PWM cu factor de umplere de 0%, viteza de rotație a acestuia va fi egală cu 0 rpm. Un factor de umplere de 100% va duce la o turație maximă a acestuia.

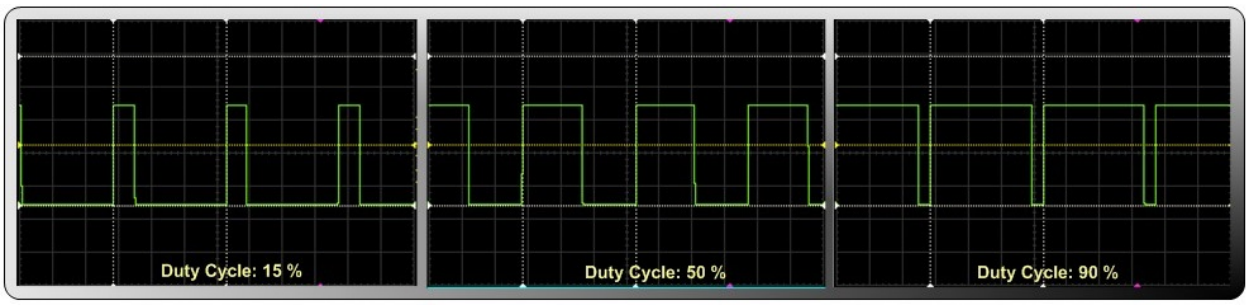
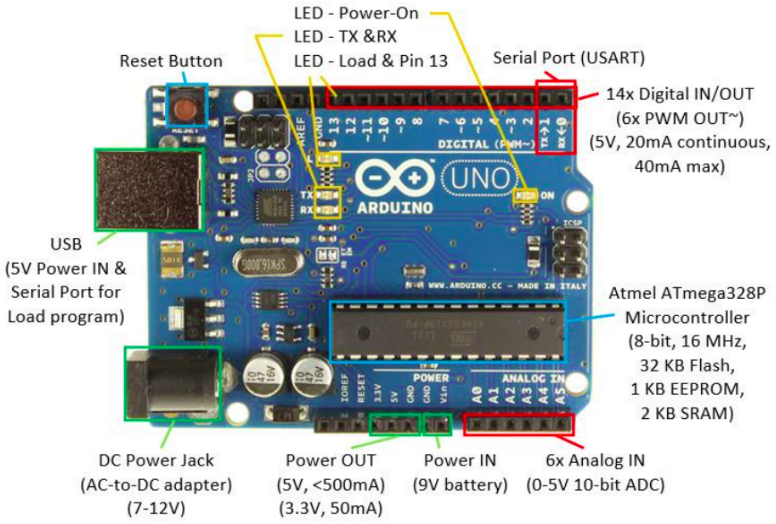


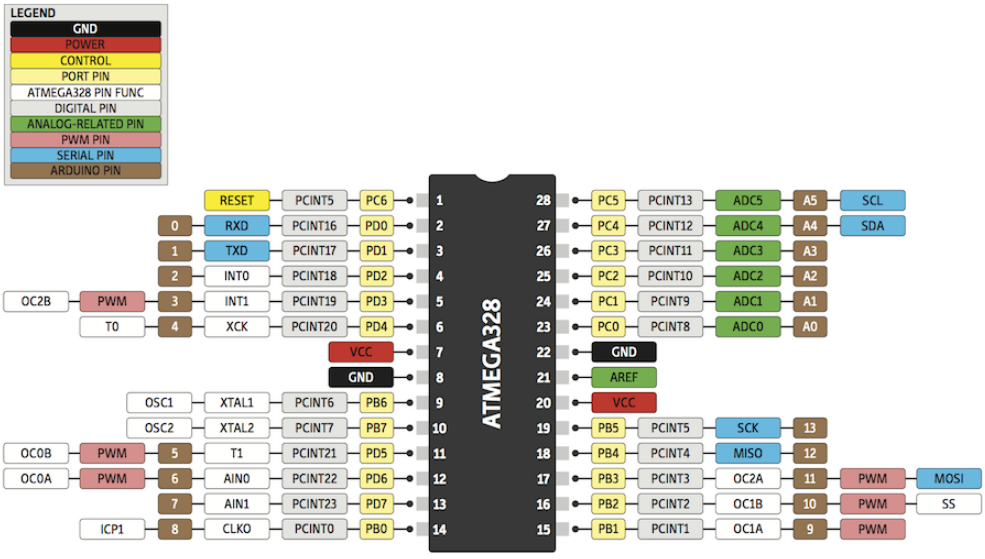
Figura 1: Forme de unda PWM la diferiti factori de umplere (duty cycle)

3. Arduino si Interfata PWM

Placa Arduino UNO, prezentata in Figura 2a, este o placă de dezvoltare open-source realizată pe baza microcontrolerului ATmega328P.



(a) Structura Placii.



(b) Harta Functiilor pentru Pini.

Figura 2: Prezentare Generala Arduino Uno.

Acesta ofera o seriei de module, prezentate in Figura 2b, pentru iterfatarea cu circuite digitale si analoage (ex: GPIO, ADC, PWM, UART, etc.) care permit utilizatorului dezvoltarea rapida a unei plaje largi de aplicatii integrate.

Din punct de vedere al software-ului structura generala a unei aplicatii integrate pentru Arduino se bazeaza pe functiile `void setup(void);` si `void loop(void);` avand urmatoarea forma:

```
int main(void){
  setup();
  while(1){
    loop();
  }
}
```

Astfel, functia `void setup(void);` va contine intotdeauna codul pentru setarea modului de functionare a diferitelor periferice si/sau functii ale pinilor, in timp ce functia `void loop(void);` va contine intotdeauna logica aplicatiei.

Pentru utlizarea facilitatiilor de PWM ale placii Arduino, API-ul ne pune la dispozitie trei functii, anume:

- `pinMode()` – care controleaza functia unui anumit pin (ex: INPUT, OUTPUT)
- `analogWrite()` – care contrleaza factorul de umplere al modulului de PWM relativ la forma de unda generata pe pinii specifici acestuia de pe placa Arduino.
- `map()` – scaleaza un numar de la un interval la alt interval.

4. Cerinte

Cerinta 1 Utilizand TinckerCAD familirizati-va cu circuitul si scheletul de cod pentru acest laborator: https://www.tinkercad.com/things/bBao5PXOQXR

Cerinta 2 Pornind de la circuitul dat legati un al doilea motor la integratul L293D.

Cerinta 3 In scheletul de cod asociat simularii, completati setarile necesare utilizarii pinilor 10, 11, 9, 3 in modul de `OUTPUT` pentru PWM.

Cerinta 4 Implementati functia `void setPWM(Motor m, Direction d, int duty_cycle);` pentru a misca un anumit motor, intr-o anumita directie cu factorul de umplere `duty_cycle` primit ca parametru.

Cerinta 5 In functia `loop();` implementati o simulare pentru accelerarea celor doua motoare, in directia inainte in 10 pasi pe durata unei secunde.

Cerinta 6 In functia `loop();` adaugati o simulare pentru decelerarea celor doua motoare, in directia inainte in 10 pasi pe durata unei secunde.

Cerinta 7 In functia `loop();` adaugati o simulare similara cerintelelor 5 si 6 dar pentru directia inapoi.

Achizitii de Date
Laboratorul 2: Roboti cu *Drive* Diferential

Dan Novischi

1. Introducere

Scopul acestui laborator este de a implemnta modelul cinematic al unui robot diferential utilizand datele provinite de la senzori de tip *quadrature encoder* si regulatorul *PID* in vederea determinarii starii si controlului acestuia la orice moment de timp.

2. Model Diferential

Robotul cu *drive* diferential, prezentat schematic in Figura 1, este un robot mobil a carui miscare este bazata pe actuarea independenta a doua roti montate simetric de o parte si de alta a unui sasiu. Astfel, prin variatia independenta vitezelor de rotatie (la ax) a celor doua roti se poate genera miscarea dorita a robotului, nefiind necesara utilizarea unui mecanism aditional pentru controlul directiei. Uzual, implementarea fizica a acestui robot se realizeaza prin utlizarea a doua motoare electrice (de curent continuu) ce pot fi controlate independent prin tehnica de PWM studiata anterior.

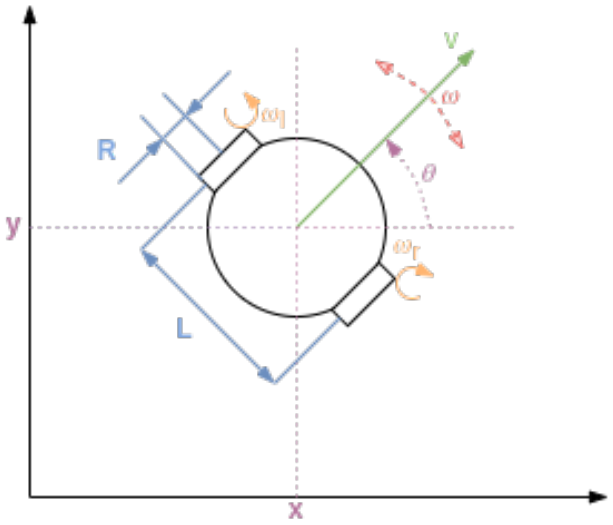


Figura 1: Modelul Diferential al Robotului

La nivelul inalt, miscarea robotului este descrisa de perechea de parametrii (v, ω) unde:

- v – reprezinta viteza liniara de deplasare a robotului.

- ω – reprezinta viteza angulara de rotatie a robotului.

in timp ce starea robotului este descrisa de perechea de parametrii (x, y, θ) denumita pose sau odometrie, unde:

- (x, y) – reprezinta coordonatele robotului fata de o referinta globala in 2D si sunt exprimate (de obicei) fata de punctul care reprezinta centrul de greutate al robotului.
- θ – reprezinta orientarea robotului fata de acelasi referinta globala. Prin conventie, acest unghi creste in sens trigonometric fata de axa X si scade in sensul acelor de cesornic fata de aceeasi axa.

Astfel, ecuatiile de miscare la acest nivel exprima felul in care se modifica starea robotului in timpul deplasarii si poarta numele de **model uniciclu**:

$$\hat{x} = v\cos(\theta)$$

$$\hat{y} = v\sin(\theta)$$

$$\hat{\theta} = \omega$$

Transfelul de la modelul unicilu la cel diferential (fizic) depinde de parametrii constructivi ai robotului, anume:

- R – raza rotiilor cuplate la axurile celor doua motoare (stang si drept).
- L – lungimea axei (imaginare) dintre roti a sasiului.

Prin intermediul acestor parametrii viteza de rotatie la ax a motorului drept ω_r si respectiv viteza la ax a celui stang ω_l se exrpima in functie de perechea (v, ω) astfel:

$$\omega_r = \frac{2v + \omega L}{2R} \ [rad/s]_{SI}$$

$$\omega_l = \frac{2v - \omega L}{2R} \ [rad/s]_{SI}$$

Tot aici, starea robotului la un anumit moment de timp se poate calcula pe baza deplasamentelor realizate de roata dreapta D_r si respectiv cea stanga D_l, anume:

$$x = x + \frac{D_r + D_l}{2}\cos(\theta) \ [m]_{SI}$$

$$y = y + \frac{D_r + D_l}{2}\sin(\theta) \ [m]_{SI}$$

$$\theta = \theta + \frac{D_r - D_l}{L} \ [rad]_{SI}$$

unde deplasamentul unei roti se determina pe baza rotatiei efectuate de acesta intre doua momente succesive de timp:

$$D = 2\pi R \cdot \Delta\texttt{rotatie}\,[m]_{SI}$$

3. Encoder in Cadratura

Un encoder in cadratura (en. quadrature encoder) este un senzor incremental cu ajutorul caruia se poate masura rotatia efectuata de axul unui motor intr-o anumita directie. Acesta are doua canale de iesire (A si B) care furnizeaza un numar de pulsuri egal distantate pe o rotatie a axului – vezi Figura 2. Pulsurile generate de cele doua canale sunt (constructiv) defazate intre ele la 90°. In functie de ordinea de detectie a pulsurilor – (A, B) sau (B, A) – se determina directia de rotatie a axului (in sensul acelor de ceas – clocwise CW sau in sens trigonometric – counter clock wise CCW).

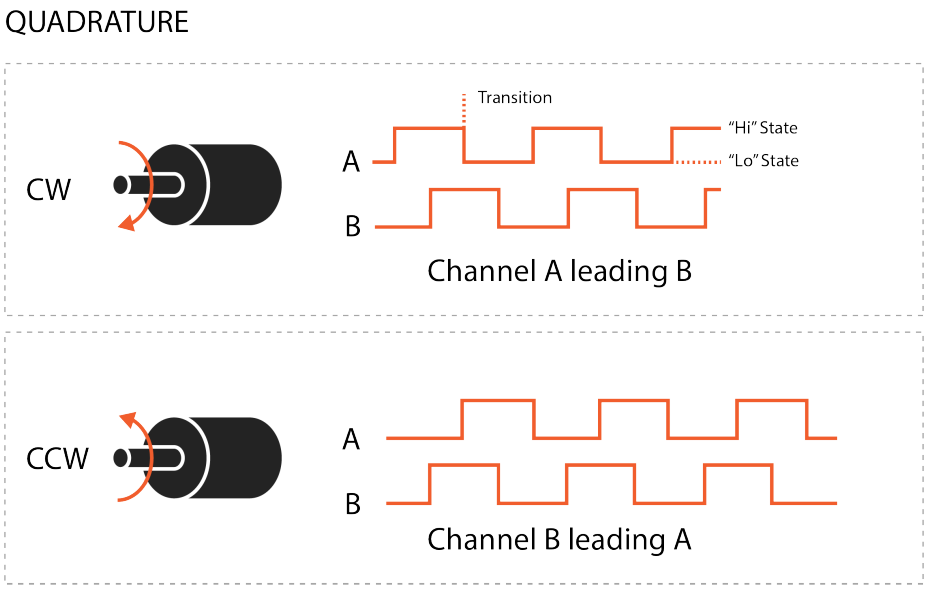


Figura 2: Encoderul in Cadratura.

Astfel, rotatia efectuata de axul unui motor, la care este cuplat un encoder in cadratura, se masoara prin incrementarea sau decrementarea numarului de pulsuri in functie de directia de rotatie. Mai exact, se masoara numarul de treceri aferente pulsurilor de la pozitiv la negativ si de la negativ la pozitiv aferent celor doua canale in functie de directia de rotatie. Acest numar de treceri poarta denumirea generica de *counts*, iar prin diferenta intre doua astfel de numere la momente succesive de timp obtinem deplasemntul (in counts) a axului intre aceste momente:

$$\Delta\texttt{counts} = \texttt{count}_k - \texttt{count}_{k-1}\ \ [\texttt{counts}]$$

Raportand aceasta diferenta la numarul total de *counts* N pe care encoderul il poate inregistra intr-o rotatie completa, obtinem numarul de rotatii (sau rotatia partiala) a axului intre cele doua momente succesive de timp:

$$\Delta\texttt{rotatie} = \frac{\Delta\texttt{counts}}{N} \ [\texttt{rotatii}]$$

Stiind numarul de rotatii la ax putem pe de-o parte calcula starea robotului, iar pe de alta putem raporta acest numar la unitatea de timp pentru a estima viteza de rotatie a motorului in $[rad/s]_{SI}$ sau in `rpm` (rotatii pe minut).

4. Cerinte

Sarciniile din cadrul acestui laborator umaresc cuplarea modelului uniciclu cu bucle de control aferente celor doua motoare astfel incat (vezi Figura 3): pe de-o parte robotul sa fie capabil sa execute orice comanda (v, ω), iar pe de alta parte starea (x, y, θ) acestuia sa fie determinata la orice moment de timp.

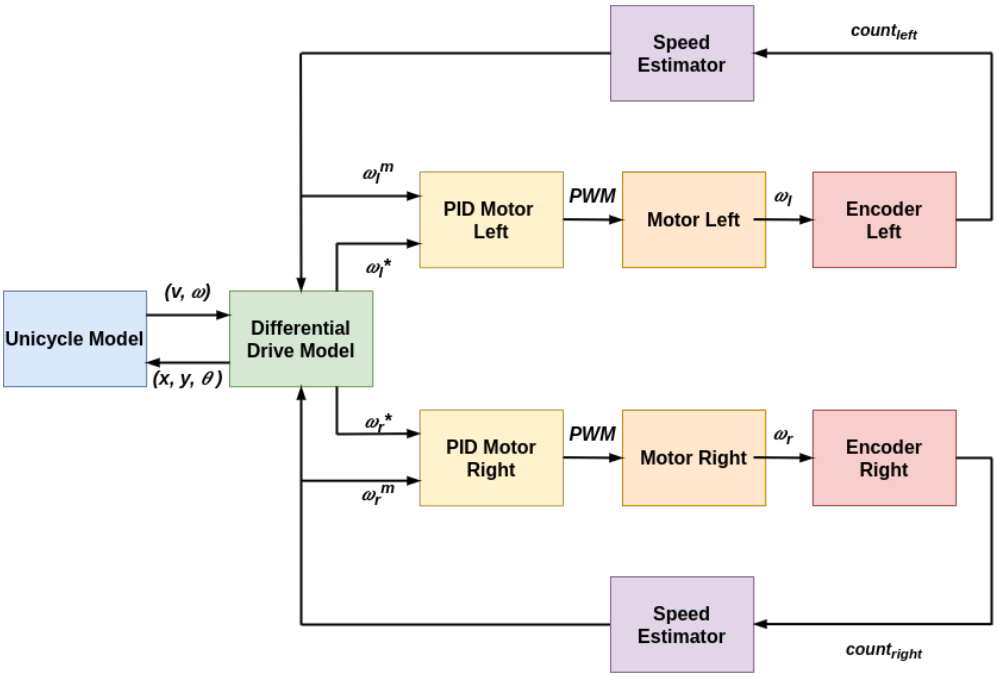


Figura 3: Bucla de control a unui Robot Diferential.

Aveti la dispozitie diagrama din TinkerCAD a carui circuit este deja legat pentru cele doua motoare. Scheletul de cod asociat este divizat in mai multe sectiuni (prin comentarii multi-line) dupa cum urmeaza:

- *Constant & Macro Definitions* – aici gasiti definite constantele/macro-urile utilizate in program; (ex: `#define ENC_MAX_COUNT 2064` care reprezinta numarul total N de counts furnizati de encoder intr-o rotatie).

- *Application Type Definitions* – aici gasiti toate definitiile de tip utilizate in aplicatie; (ex: `typedef struct Robot{...};`).
- *Application Function Definitions* – aici veti gasi toate definitiile de functii pe care le aveti de implementat si cele care au fost deja implementate.

  (ex: `void initRobot(struct Robot* robot, float r, float l){...}`)
- *Application Global Variables* – aici veti gasi toate declaratiile de variabile globale utilizate in program. (ex: `Robot robot;`)
- *Main application* – aplicatia principala (impartita intre `setup();` si `loop();`) care contine initializarea si aplicatia propriu-zisa ce trebuie implementata.

Cerinta 1 Utilizand TinckerCAD familirizati-va cu circuitul si scheletul de cod pentru acest laborator: https://www.tinkercad.com/things/7HMsEsmSYHw. Unde este cazul solicitati clarificari laborantului.

Cerinta 2 Implementati functiile asociate tipului `Robot` dupa cum urmeaza:

a) `void initRobot(struct Robot* robot, float r, float l)` intializeaza un obiect de tip `Robot`.

b) `void setRobotCommand(struct Robot* robot, float v, float omega)` calculeaza vitezele de rotatie pentru motorul stang si drept al robotului pe baza unei comenzi (v, ω) primite ca parametru.

c) `void updateRobotPose(struct Robot* robot, ...)` actualizeaza starea robotului pe baza rotatiilor $\Delta\texttt{rotatie}_{\texttt{left}}$ si $\Delta\texttt{rotatie}_{\texttt{right}}$ efectuate de cele doua motoare intre doua momente de timp succesive.

Cerinta 3 Implementati functiile asociate tipului `SpeedEstimator` dupa cum urmeaza:

a) `void initSpeedEstimator(struct SpeedEstimator* est, struct Encoder *enc)` intializeaza un obiect de tip `SpeedEstimator`.

b) `float estimateSpeed(struct SpeedEstimator* est, int direction)` estimeaza numarul de rotatii (Δ`rotatii`) efectuat intre doua monete succesive de timp.

c) `float get_rpm(float delta_rotation, long update_freq)` calculeaza viteza la axul unui motor in rotatii pe minut – RPM.

d) `float get_rad_per_second(float delta_rotation, long update_freq)` calculeaza viteza la axul unui motor in rad/s.

Cerinta 4 Implementatii aplicatia principala in functia `loop` urmarind TODO-urile din schelet si cerintele de mai jos:

a) Setati comanda ($v = 0.5\ m/s,\ \omega = 0\ rad/s$) si determinati directia de rotatie a fiecarui motor pe baza unei comenzi.

b) In bucla de control a motoarelor initializati referintele aferente celor doua regulatoare PID.

c) Calculati Δ`rotatii` pentru motorul stang si drept aferent unei iteratii a buclei de control.

d) Setati intrarea pentru regulatoarele PID stang si drept, apoi obtineti iesirele celor doua regulatoare.

e) Utilizati iesirile regulatoarelor PID pentru a genera semnalele PWM aferente celor doua motoare.

f) Actualizati poese-ul robotului.

Nota: Ce comanda se poate furniza robotului astfel incat traiectoria acestuia sa descrie un cerc cu raza de 1 m ?

Achizitii de Date

Laboratorul 3: Interfete de Masurare

Dan Novischi

1. Introducere

Scopul acestui laborator este de a utiliza interfetele de conversie anlogica-digitala si cea de masurare a duratelor pentru a prelua informatii de la senzori precum temperatura sau distanta (si nu numai).

2. Conversia Analog-Digitală

Mărimile care caracterizează un mediu fizic reprezintă semnale analogice a caror valori evoluează în domeniu continuu (ex: temperatura, presiunea, umidiatatea, lumina, etc.). Pentru a preleva aceste mărimi in scopul implementării unor aplicații smart (roboti, IOT, sensor networks sau orice fel de sistem autonom), este necesar ca aceste marimi sa fie convertite in echivalentul lor digital. Astfel, lantul tipic de transformare (masurare) a celor mai multe marimi fizice este constituit din (vezi Figura 1):

- Senzor – reprezintă traductorul care transformă o marime fizică într-un echivalent al acesteia de tensiune electică.
- Etajul de Amplificare / Filtrare – reprezintă elementul care conditionează semnalul de tensiune furnizat de sensor prin eliminarea diverselor zgomote si amplificarea corespunzatoare a acestuia.
- Ciruit esantionare/reținere – preleveaza un eșantion al semenalului de tensiune conditionat în scopul retinerii acestuia pe durata conversie din analog in digital.
- Covertor analog/digital – converteste eșantionul furnizat de circuritul de eșantionare/retinere în valoarea lui digitală.

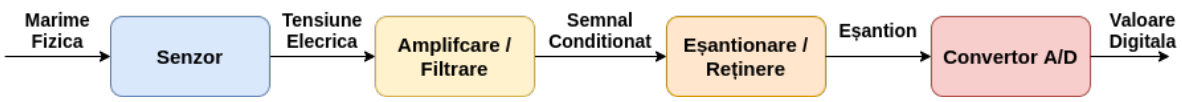


Figura 1: Lantul de măsurare tipic al unei marimi fizice.

Fiecare dintre aceste componente au caracteristici care definesc felul in care obtinem forma digitala a valorii masurate. Spre exemplu **senzorul de temperatura** TMP36 este un sezor

linear cu senzitivitatea de 0.01 $V/°C$. Cea ce inseamna ca o valoare digitala a tensiunii masurate trebuie inmultita cu inversul sezitivitatii (100 $°C/V$) pentru a obtine echivalentul digital in grade. Similar **in cazul etajului de amplifcare/filtrare**, pentru a obtine valoarea reala a tensiunii electrice trebuie sa impartim valoarea digitala a semnalului conditionat la constanta de amplificare. În timp ce, valoarea digitala a unui eșantion si implicit a semnalului coditionat depinde de caracteristicile convertorului analog/digital. Astfel, distingem două **caracteristici principale ale convertorului analog digital** relativ la obtinerea valorilor digitale:

- **Rezoluția** – reprezintă numărul de valori discrete pe care convertorul poate să le furnizeze la ieșirea sa în intervalul de măsură. Intern rezultatele unei conversii sunt stocate sub forma binara, astfel rezoluția unui convertor analog-digital este exprimată în biți. Spre exemplu, rezoluția perifericului de conversie analog/digitală integrat in Arduino este de 10-biti, cea ce inseamna ca numarul de valori discrete (sau nivele) de iesire este egal cu: $2^{10} = 1024$, intre $0 \dots 1023$.

- **Gama de măsurare** – reprezinta intervalul de tensiune al conversiei analog/digitale si este dat de tensiunea de alimentare a convertorului. Spre exemplu, convertorul analog/digital integrat in Arduino este alimentat intre $0 \dots 5V$. Astfel, tinand cont de rezolutia convertorului inseamnă ca avem o rezolutie de masurare de: $\frac{5-0}{1024} \left[\frac{V}{bit}\right]$, adică o rezolutie de masurare de 4.8 $\left[\frac{mV}{bit}\right]$.

3. Masurarea Duratei Pulsurilor

Unii dintre cei mai importanți senzori sunt cei bazați pe emisia si recepția unor pulsuri de frecvență înaltă. In acesta categorie intră senzori de distantă cu ultrasunete (ultrasonici) sau raze luminoase infra roșii și cele laser precum cele LiDAR (en. Light Detection and Ranging). Spre exemplu, senzorii ultrasonici emit pulsatii acustice scurte, de înalta frecvența, la intervale de timp regulate (vezi Figura 2). Acestea se propaga prin aer cu viteza sunetului. Dacă lovesc un obiect, acestea sunt reflectate înapoi ca semnale ecou la senzor. Masurarea distanței in acest caz se realizeaza prin masurarea intervalului de timp Δt scurs dintre emiterea semnalului si receptarea ecoului.

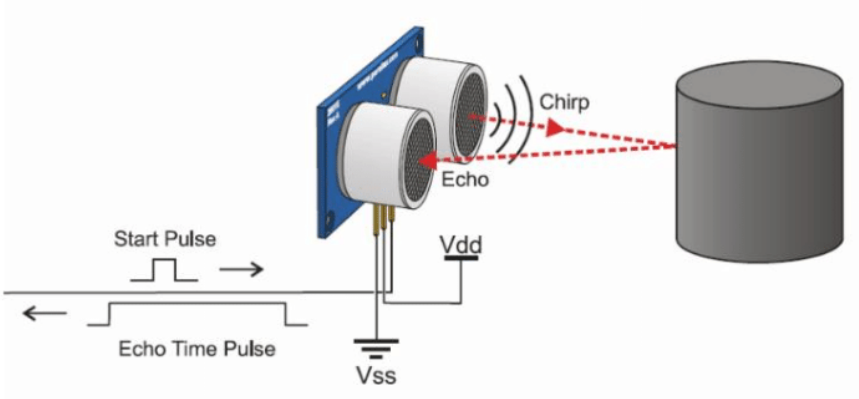


Figura 2: Principiul de functionare a unui senzor de distanta cu ultrasunete.

Astfel, știind ca viteza de propagare a sunetului in aer este de $v_s = 340\ [m/s]$ putem calcula dinstanta față de obiect prin:

$$d_{\texttt{obiect}} = \frac{v_s \ \cdot \ \Delta t}{2}$$

Aici '2' din formula se datoreaza faptului ca unda acustica parcurge drumul pana la obiect si inapoi. Analog, se poate determina distanta si in cazul senzoriilor care folosesc raze luminoase, numai ca in acest caz viteza de propagare a undei prin mediu este cea a luminii.

4. Cerinte

Sarciniile din cadrul acestui laborator au ca obiectiv utilizarea interfetei de conversie analog/digitala pentru masurarea temperaturii si a celei pentru masurarea duratei pulsurilor in vederea obtinerii distantei de la un senzor ultrasonic. Ambele interfete sunt disponibile pe placa Arduino si se pot utiliza prin API-ul asociat. Astfel, in realizarea cerintelor va trebui sa consultati documentatia pentru urmatoarele functii:

- `pinMode()` – functie care configureaza modul de functionare a unui pin.
- `digitalWrite()` – functie care controleaza starea unui pin in modul de `OUTPUT`.
- `delayMicroseconds()` – functie care introduce un delay in microsescude.
- `analogRead()` – functie care citeste valoarea data de convertorul analog digital prin conversia unui semnal analog disponibil pe un pin de input analog.
- `pulseIn()` – functie care masoara durata unui puls in microsecunde.

Aveti la dispozitie diagrama din TinkerCAD a carui circuit este deja legat. Scheletul de cod asociat este divizat in mai multe sectiuni (prin comentarii multiline) dupa cum urmeaza:

- *Constant & Macro Definitions* – aici gasiti definite constantele/macro-urile utilizate in program.
- *Application Function Definitions* – aici veti gasi toate definitiile de functii pe care le aveti de implementat si cele care au fost deja implementate.
- *Application Global Variables* – aici veti gasi toate declaratiile de variabile globale utilizate in program.
- *Main application* – aplicatia principala (impartita intre `setup();` si `loop();`) care contine initializarea si aplicatia propriu-zisa ce trebuie implementata.

Cerinta 1 Utilizand TinckerCAD familirizati-va cu circuitul si scheletul de cod pentru acest laborator: `https://www.tinkercad.com/things/fldstRodykr`. Unde este cazul solicitati clarificari laborantului.

Cerinta 2 Implementati functiile asociate masurarii temperaturii:

a) `float adcReadingToVoltage(float adcRawReadingValue)` primeste ca input valoarea masurata de ADC si intotoarce nivelul de tensiune asociat in V.

b) `float computeTemperature(float voltage)` primeste valoarea unei tensiuni masurate si intoarce temperatura in $°C$.

Notă: Calculul temperaturii din valoarea tensiunii masurate trebuie sa aibă în vedere decalajul acesteia (dat in schelet prin constanta `TEMPERATURE_VOLTAGE_OFFSET`).

Cerinta 3 Implementati functiile asociate masurarii distantei:

a) `void pulseOutUltrasonic(int ultrasoundPin)` primeste ca input pin-ul de semnal al senzorului ultrasonic si emite pe acesta un puls cu durata de 5 us.

b) `float computeDistance(float duration)` primeste durata unui puls in microsecunde si intoarce dinstanta in centimetri.

Cerinta 4 Implementatii aplicatia principala in functia `loop` urmarind cerintele de mai jos si TODO-urile din schelet:

a) Citiți valoarea rezultată in urma unei conversii analog digitale utilizand constanta `ADC_SENSOR_INPUT_PIN`.

b) Convertiti valoarea rezultată in urma conversiei in valoarea de tensiune echivalentă.

c) Calculati valoarea temperaturii pe baza tensiunii obtinute la pasul anterior.

d) Emiteti un puls cu ajutorul senzorului de ultrasunete.

e) Masurati durata pulsului receptat de senzorul de ultrasunete in microsecunde.

f) Calculati distanta in centimetri pe baza duratei obtinute la pasul anterior.

Nota: O implementarea corecta va avea un output al simulararii similar cu cel din imaginile de mai jos:

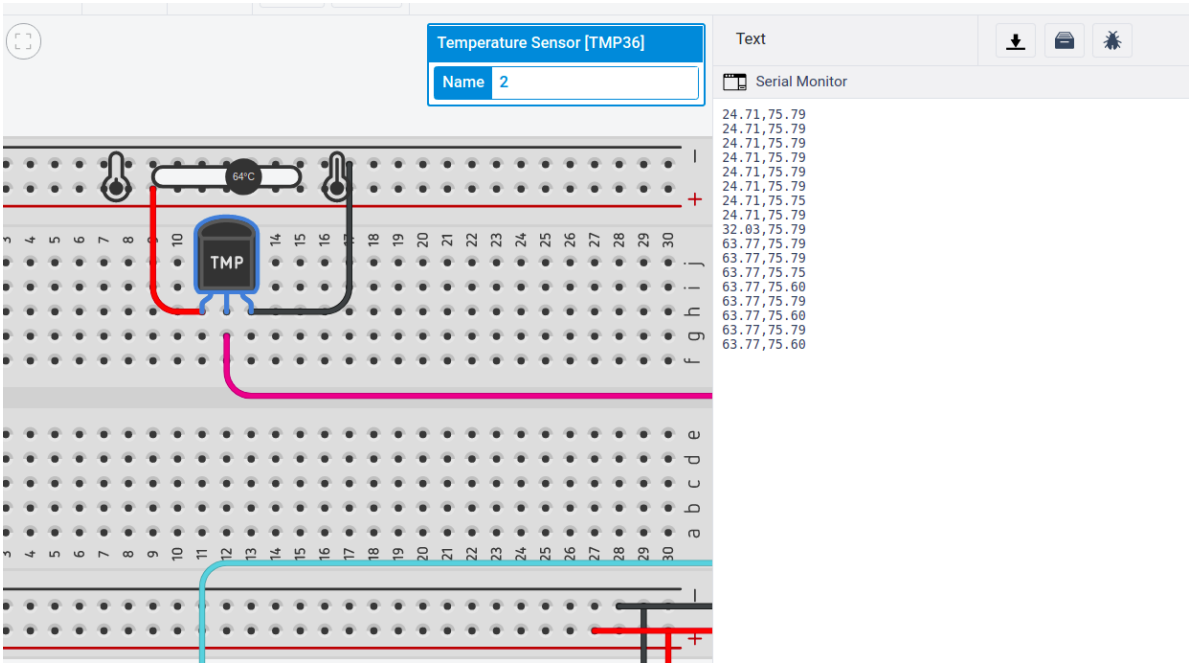


Figura 3: Rezultatul masurarii temperaturii.

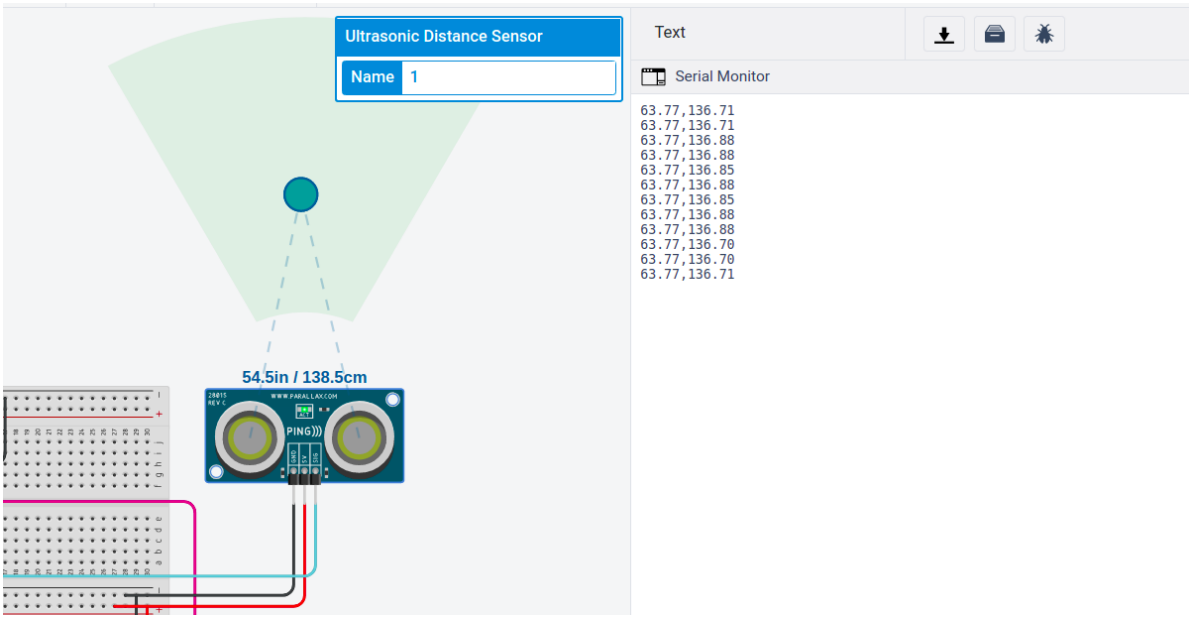


Figura 4: Rezultatul masurarii distantei.

Achizitii de Date

Laboratorul 4: Interfete de Comunicatie Inter-Dispozitive

Dan Novischi

1. Introducere

Scopul acestui laborator este de a utiliza interfetele de comunicatie seriala in vedere inter-conectarii dispozitivelor (ex: SOCs, senzori inteligenti, etc...).

2. Interfete de comunicație

Aplicațiile integrate au inveitabil in componență circuite inter-conectate (procesoare sau alte circuite integrate) cu scopul de a crea un sistem dedicat. Pentru ca aceste circuite să-și poată transfera informații trebuie sa conțină o modalitate de comunicare comună. Deși exista sute de modalități de comunicare, aceste modalități se împart în doua categorii: seriale si paralele.

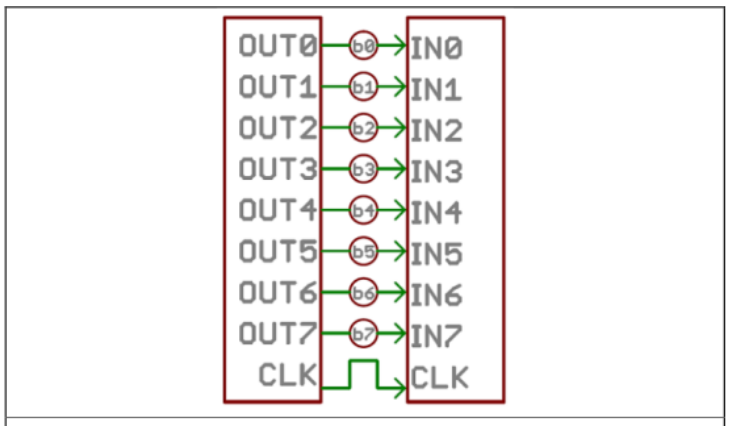


Figura 1: Transmisie paralelă.

Interfețele paralele transferă mai mulți biți în același timp. De obicei au nevoie de magistrale de date (bus) care transmit pe 8, 16 sau mai multe linii. Datele transmise si recepționate sunt fluxuri masive de 1 si 0. În Figura 1, observăm o magistrală de date cu lățimea de 8 biți, controlată de un semnal de ceas și transmite câte un octet la fiecare puls al ceasului.

Interfețele seriale trimit informația bit cu bit. Aceste interfețe pot opera doar pe un singur fir si de obicei nu necesita mai mult de 4 fire (minim 1, maxim 4). In Figura 2, se poate

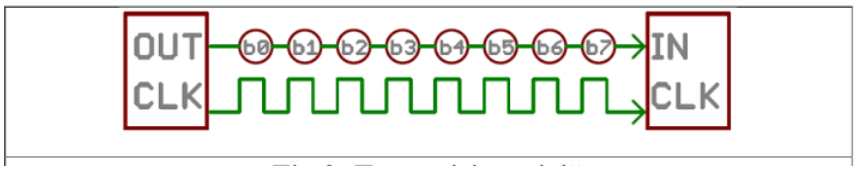


Figura 2: Transmisie serială.

observa un exemplu de interfața care transmite cate un bit la fiecare impuls de ceas (aici doar 2 fire sunt folosite). Deși protocoalele de comunicație paralelă au beneficiile lor, acestea necesita un număr mare de pini din partea platformei de dezvoltare pe care o folosesc. Astfel, având în vedere ca numărul de pini de pe Arduino UNO/ Mega e redus ne vom concentra pe interfețe de comunicație serială.

Tipuri de transfer serial

Din punctul de vedere al direcției de transfer, se pot distinge următoarele tipuri de comunicație serială:

- *Simplex* – datele sunt transferate întotdeauna în aceeași direcție, de la echipamentul transmițător la cel receptor.
- *Half-Duplex* – fiecare echipament terminal funcționează alternativ ca transmițător, iar apoi ca receptor. Pentru acest tip de conexiune, este suficientă o singură linie de transmisie.
- *Full-Duplex* – datele se transferă simultan în ambele direcții, necesetand două linii de date.

Din punctul de vedere al sincronizării dintre transmițător și receptor, există două tipuri de comunicație serială:

- *Sincronă* – folosește un semnal de ceas unic la ambele capete ale comunicației (emițător și receptor). Aceasta modalitate de comunicație este de multe ori mai rapidă, cu toate astea are nevoie de cel puțin un fir in plus intre dispozitivele care comunica (pentru transmiterea semnalului de ceas). Exemple de astfel de comunicații sunt SPI și I2C.
- *Asincronă* – datele sunt transferate fără suportul unui semnal de ceas extern. In acest fel se elimina firul de ceas, dar o atenție sporita trebuie acordata sincronizării datelor transferate.

3. Comunicatie seriala UART

Comunicatia UART (Universal Asynchronous Receiver Transmitter) se referă la un tip de comunicatie full-duplex, asincron care conține un număr de mecanisme ce asigură transferul de date robust şi fără erori, fiind caracterizat de:

- *Rata de transfer (baud rate)* – ne spune cât de rapid sunt transmise datele pe linia serială. Aceasta mărime este exprimată in stări pe secunda (de obicei o stare este 1 sau 0, deci un bit, dar exista interfețe care pot avea mai mult de doua stări, si atunci baud rate nu este același lucru cu biți pe secunda). Exista mai multe baud rate-uri standard precum 1200, 2400, 4800, 19200, 38400, 57600, sau 115200.

- *Pachetul de date* – Fiecare bloc (de obicei un octet) de date, care urmează sa fie transmis este trimis într-un pachet (frame) de biți. Pachetele sunt create adăugând biți de sincronizare sau paritate datelor care urmează sa fie transmise.

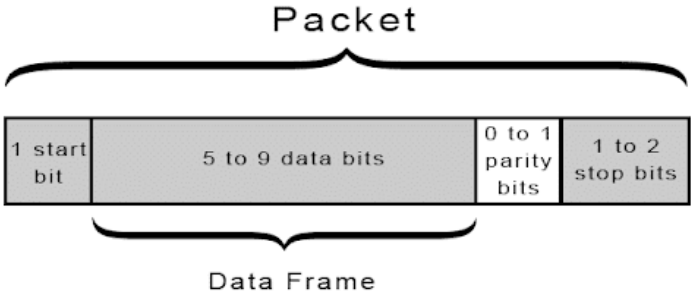


Figura 3: Pachetul de date.

- *Biții de date* – Cantitatea de informație din fiecare pachet poate fi intre 5 si 9 biți (datele standard sunt pe 8 biți).

- *Biții de sincronizare (synchronization bits)* – sunt biți speciali care sunt transferați cu fiecare caracter de date. Aceștia sunt biții de start si de stop; ei marchează începutul si finalul unui pachet.

- *Biții de paritate (parity bits)* – asigură un tip rudimentar de control al erorii. Paritatea poate fi „impară" (odd) sau „pară"(even). Pentru a produce bitul de paritate toți biții din caracterul de date sunt compuși cu operatorul „sau exclusiv" și paritatea rezultatului ne spune daca bitul este setat sau nu

- Linia de date (care in stare inactiva este la nivel logic „1")

4. Transmisia Inter-Integrated Circuit (I2C)

Inter Integrated Circuit (I2C) e un protocol care a fost creat pentru a permite mai multe circuite integrate "slave" sa comunice cu unul sau mai mute cipuri "master". Acest tip de comunicare a fost intenționat pentru a fi folosit doar pe distante mici de comunicare si asemenea protocolului UART are nevoie doar de 2 fire de semnal pentru a trimite/primii informații.

Fiecare bus I2C este compus din 2 semnale: SCL si SDA. **SCL** reprezintă semnalul de ceas iar **SDA** semnalul de date. Semnalul de ceas este întotdeauna generat de bus masterul curent. Spre deosebire de alte metode de comunicație serială, magistrala I2C este de tip "open

drain", ceea ce înseamnă ca pot trage o anumita linie de semnal în 0 logic dar nu o pot conduce spre 1 logic. Așadar, se elimina problema de „bus contention", unde un dispozitiv încearcă să tragă una dintre linii in starea „high" în timp ce altul o aduce în „low", eliminând posibilitatea de a distruge componente. Fiecare linie de semnal are un pull-up rezistor pe ea, pentru a putea restaura semnalul pe „high", când nici un alt dispozitiv nu cere „low".

Protocolul I2C

Mesajele sunt sparte în 2 tipuri de cadre (frames): cadre de adresa, unde masterul indică slave-ul la care mesajul va fi trimis și unul sau mai multe cadre de date care conțin mesaje pe 8 biți pasate de la master la slave sau viceversa. Datele sunt puse pe linia SDA după ce SCL ajunge la nivel low, si sunt eșantionate când SCL ajunge HIGH. Timpul intre nivelul de ceas și operațiile de citire/scriere este definit de dispozitivele conectate pe magistrală şi va varia de la cip la cip.

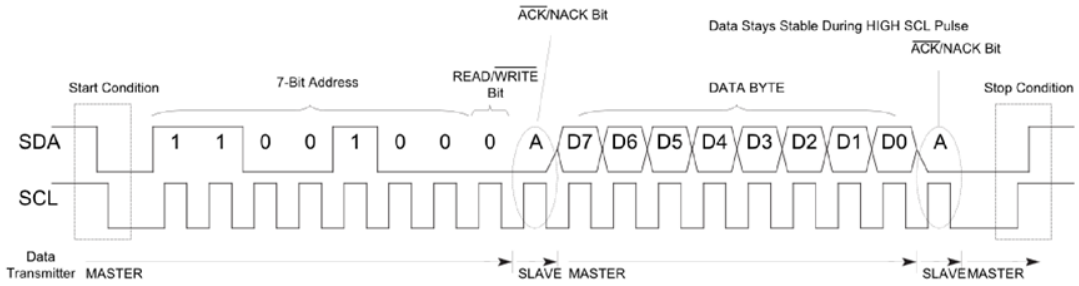


Figura 4: Protocolul de transmisie I2C.

În Figura 4 sunt prezentate schematic etapele de schimb ale informatiilor utlizand interfața I2C:

- *Coditia de start / Start Condition* – Pentru a iniția cadrul de adresa, dispozitivul master lasă SCL high și trage SDA low. Acest lucru pregătește toate dispozitivele slave întrucât o transmisie este pe cale să înceapă. Dacă două dispozitive master doresc să își asume busul la un moment dat, dispozitivul care trage la nivel low SDA primul câștiga arbitrajul și implicit controlul busului.

- *Cadrul de adresa / Address Frame* – este întotdeauna primul în noua comunicație. Mai întâi se trimit sincron biții adresei, primul bit fiind cel mai semnificativ, urmat de un semnal de R/W pe biți, indicând dacă aceasta este o operație de citire (1) sau de scriere (0). Bitul 9 al cadrului este bitul NACK / ACK. Acesta este cazul pentru toate cadrele (date sau adresa). După ce primii 8 biți ale cadrului sunt trimiși, dispozitivului receptor îi este dat controlul asupra SDA. Dacă dispozitivul de recepție nu trage linia SDA în 0 logic înainte de al 9-lea puls de ceas, se poate deduce că dispozitivul receptor fie nu au primit datele, fie nu a știut cum sa interpretezemesajul. În acest caz, schimbul se oprește, și tine de master să decidă cum să procedeze mai departe.

- *Cadrele de date / Data Frames* – După ce cadrul adresă a fost trimis, datele pot începe sa fie transmise. Masterul va continua sa genereze impulsuri de ceas, la un interval

regulat, iar datele vor fi plasate pe SDA, fie de master fie de slave, în funcție de starea biNilor R/W (care indică dacă o operație este citire sau scriere). Numărul de cadre de date este arbitrar.

- *Condiția de oprire / Stop condition* – De îndată ce toate cadrele au fost trimise, masterul va genera o condiție de stop. Condițiile de stop sunt definite de tranziții low/high ($0 \longrightarrow 1$) pe SDA, după o tranziție $0 \longrightarrow 1$ pe SCL, SCL rămânând pe high. În timpul operațiilor de scriere valoarea din SDA nu ar trebui să se schimbe când SCL e high pentru a evita condițiile de stop false.

5. Cerinte

Sarciniile din cadrul acestui laborator au ca obiectiv utilizarea celor doua interfete de comunicatie, UART si I2C, pentru a implementa o comunicatie bidirectionala pe de-o parte intre PC si o placuta Arduino, iar pe de alta intre doua placute Arduino. Ambele interfete sunt disponibile pe placa Arduino si se pot utiliza prin API-ul asociat obiectlui Serial si cel al obiectului Wire – exmplele de aici si aici pot fi utile. Astfel, in realizarea cerintelor va trebui sa consultati documentatia pentru urmatoarele functii/metode:

- `Serial.begin()` – metoda care configurareaza baudrate-ul interfetei UART.
- `Serial.print()` – metoda care transmite un caracter / stream de caractere.
- `Serial.println()` – metoda care transmite un caracter / stream de caractere urmate de un ENTER.
- `Serial.available()` – metoda care intoarce numarul de caractere primite pe interfata UART.
- `Serial.read()` – metoda care citeste un caracter primit pe interfata UART.
- `Wire.begin()` – metoda care initializeaza un master sau un slave pentru interfata I2C.
- `Wire.beginTransmission(address)` – metoda care initiaza trasmisia de la un master catre un slave a datelor.
- `Wire.endTransmission()` – metoda care inchide trasmisia de la un master catre un slave a datelor.
- `Wire.write(data)` – metoda care transmite datele pe bus-ul I2C.
- `Wire.onReceive(handler)` – metoda care ataseaza o functie care se va executa ori de cate ori un eveniment de receptie pe bus I2C este detectat.

Aveti la dispozitie diagrama din TinkerCAD a carui circuit este deja legat. Scheletul de cod asociat este divizat in mai multe sectiuni (prin comentarii multiline) dupa cum urmeaza:

- *Constant & Macro Definitions* – aici gasiti definite constantele/macro-urile utilizate in program.

- *Application Function Definitions* – aici veti gasi toate definitiile de functii pe care le aveti de implementat si cele care au fost deja implementate.
- *Application Global Variables* – aici veti gasi toate declaratiile de variabile globale utilizate in program.
- *Main application* – aplicatia principala (impartita intre `setup();` si `loop();`) care contine initializarea si aplicatia propriu-zisa ce trebuie implementata.

Cerinta 1 Utilizand TinckerCAD familirizati-va cu circuitul si scheletul de cod pentru acest laborator: https://www.tinkercad.com/things/6S90xuKqmLu. Unde este cazul solicitati clarificari laborantului.

Cerinta 2 In functia `setup()` si `loop()` implementati setarile interfelor UART / I2C si aplicatia principala pentru placuta Arduino Master:

a) Setati baudrate intefetei UART folosind constantele din program.

b) Setati master-ul pentru comunicatia pe bus-ul I2C.

c) Implementati aplicatia principala master care citeste succesiv caractere primite de la monitorul Serial (de la PC) catre placuta Arduino Master si le transmite pe bus-ul I2C catre placuta Arduino Slave.

Cerinta 3 Implementati functia `void receiveEvent(int numBytes)` pe placuta Arduino Slave, care citeste un carcter de pe bus-ul I2C in variabila `i2c_read_character`.

Cerinta 4 In functia `setup()` si `loop()` implementati setarile interfelor UART / I2C si aplicatia principala pentru placuta Arduino Slave:

a) Setati baudrate intefetei UART folosind constantele din program.

b) Setati adresa pentru bus-ul I2C a placutei Slave folosind constantele din program.

c) Setati functia `void receiveEvent(int numBytes)` ca si call-back pentru un eveniment de receptie pe bus-ul I2C.

d) Implementati aplicatia principala slave care citeste succesiv caractere primite de la placuta Arduino Master pe bus-ul I2C si le transmite pe bus-ul UART catre PC.