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.

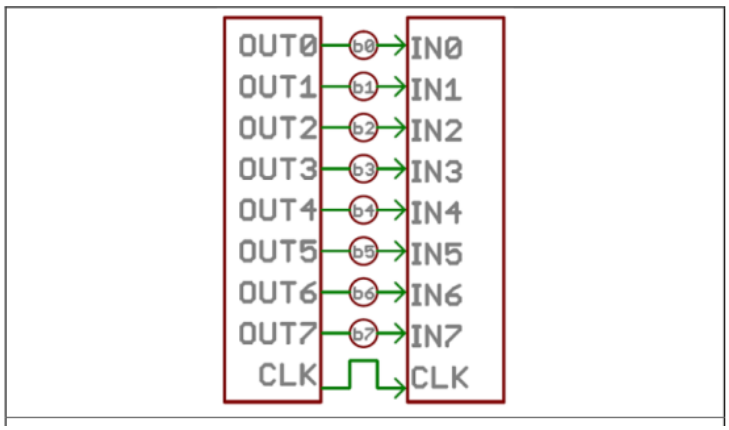

Figura 1: Transmisie paralelă.

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

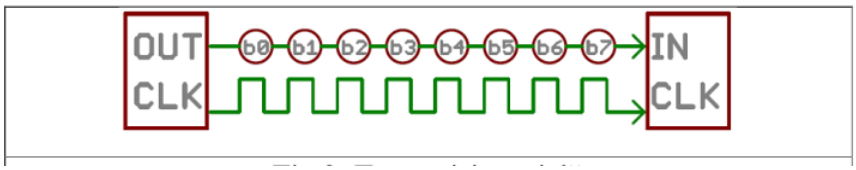

Figura 2: Transmisie serială.

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.

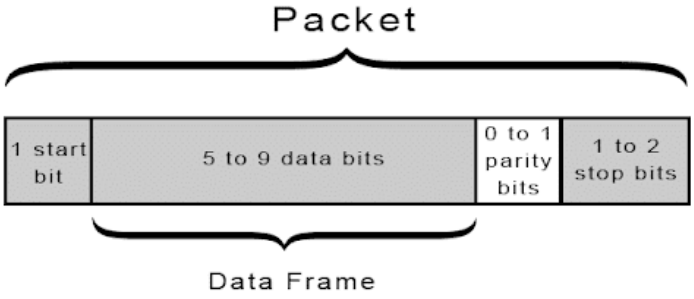

Figura 3: Pachetul de date.

- *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.

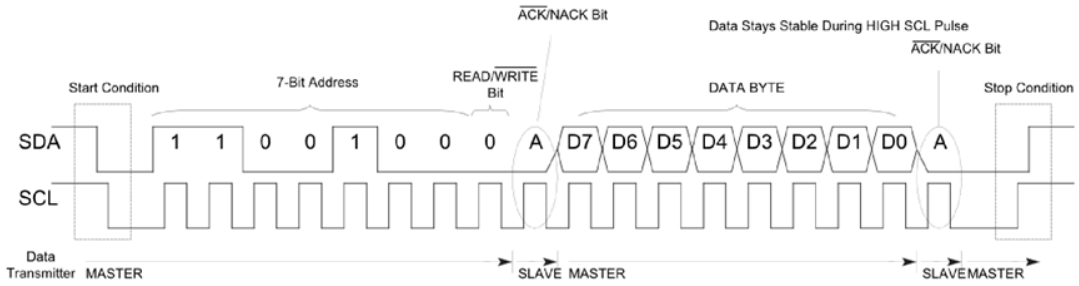

Figura 4: Protocolul de transmisie I2C.

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.