**Comunicația dintre calculator și o**

**placă de dezvoltare FPGA prin portul serial**

Studenți: **Bagoly Szabolcs-Márton** și **Bánhidi Zoltán**

Îndrumător: **Mocan Cristi**

Grupa: **30234**

Cuprins

[1. Rezumat 3](#_Toc483139539)

[2. Introducere 4](#_Toc483139540)

[2.1. Clasificarea interfețelor de comunicare serială 4](#_Toc483139541)

[2.2. Caracteristicile generale ale interfeței UART 5](#_Toc483139542)

[2.3. Interfața I2C 6](#_Toc483139543)

[2.4. Obiective 9](#_Toc483139544)

[3. Fundamentare teoretică 10](#_Toc483139545)

[3.1. FSM pentru transmisie de date serială 10](#_Toc483139546)

[3.2. FSM pentru recepție de date serială 11](#_Toc483139547)

[3.3. Standardul RS-232 12](#_Toc483139548)

[3.4. FSM pentru interfața I2C 13](#_Toc483139549)

[3.4. Senzorul de temperatură al plăcii FPGA Nexys4DDR 15](#_Toc483139550)

[3.5. Comenzi MS-DOS pentru portul serial 15](#_Toc483139551)

[4. Proiectare și implementare 16](#_Toc483139552)

[4.1. Baud Rate Generator 17](#_Toc483139553)

[4.2. FSM pentru transmisie UART 17](#_Toc483139554)

[4.3. FSM pentru recepția UART 18](#_Toc483139555)

[5. Rezultate experimentale 19](#_Toc483139556)

[5.1. Testarea TX\_FSM UART 19](#_Toc483139557)

[5.2. Simularea modulului RX\_FSM UART 21](#_Toc483139558)

[5.2. Simularea modulului I2C\_Master 23](#_Toc483139559)

[5.4. Testarea aplicației pe calculator 26](#_Toc483139560)

[6. Concluzii 29](#_Toc483139561)

[7. Bibliografie 30](#_Toc483139562)

[Anexe 31](#_Toc483139563)

[Anexa A (Manual de utilizare al aplicației de control al temperaturii) 31](#_Toc483139564)

[Anexa B (Codul sursă al componentelor semnificative) 32](#_Toc483139565)

# 1. Rezumat

Comunicația dintre diferite dispozitive electronice este un lucru indispensabil pentru a realiza transferul de informații. Există mai multe tipuri de comicare, fiecare având avantaje și dezavantaje, de care se ține cont când se decide asupra modalității de comunicare care se încorporează într-un anumit dispozitiv. În cadrul acestui proiect au fost descrise și implementate două protocoale de comunicare frecvent întâlnite: protocolul asincron UART (*Universal Asynchronous Receiver/Transmitter*) și comunicația sincronă prin protocolul I2C (*Inter-Integrated Circuit*). Modulele responsabile pentru realizarea protocoalelor de comunicare au fost descrise în limbajul de descriere hardware VHDL, utilizând IDE-ul Vivado 2016.4 și testate pe placa de dezvoltare Nexys4DDR. Pentru aplicația dezvoltată pe calculator s-a folosit mediul Visual Studio, aplicația fiind scrisă în limbajul C#.NET Windows Forms. Pentru a demonstra comunicarea a diferitelor componente s-a implementat un termostat. Prin protocolul I2C s-a realizat conexiunea cu senzorul de temperatură încorporat pe placa FPGA Nexys4DDR, s-au citit informațiile relevante, iar apoi utilizând protocolul UART s-au transmis mai departe datele care conțin temperatura către un calculator personal. Aplicația pe calculator interpretează aceste informații și transmite înapoi plăcii de dezvoltare operația care trebuie efectuată pentru a păstra temperatura dorită de utilizator.

# 2. Introducere

Tema proiectului de față este realizarea comunicării dintre calculator și o placă de dezvoltare FPGA, mai exact Nexys4DDR. Comunicația serială este foarte des utilizată pentru a realiza comunicarea dintre două dispositive (de exemplu calculator, microcontroller, FPGA, senzor etc). Comunicarea serială are ca sarcină principală transmiterea de date între cele două dispozitive. Acest tip de comunicare este foarte des folosită, deoarece:

* necesită un număr redus de pini pentru a interconecta dispozitivele
* este acceptat pe scară largă
* costul implementării și utilizării este una redusă
* cea mai mare parte din microcontrollere include această interfață
* software-ul pentru implementarea portului serial este ușor de implementat

## 2.1. Clasificarea interfețelor de comunicare serială

O primă metodă de clasificare a interfețelor de comunicare serială este după modul de sincronizare a transmisiei. Din acest punct de vedere, există două categorii:

* **Sincronă**: Folosește un semnal de ceas pentru a sincroniza cele două dispozitive. Nu se folosesc biți de sincronizare (START și STOP), ci fiecare mesaj (poate conține unul sau mai mulți octeți) transmis este precedat de un șir de caractere speciale de sincronizare. Exemple de astfel de sisteme sunt: SPI (Serial Peripheral Interface) și I2C (Inter-Integrated Circuit).
* **Asincronă**: Nu folosește un semnal de ceas. Pentru asigurarea sincronizării dintre emițător și receptor se folosesc biți de sincronizare (un bit de START cu valoarea logică ‘0’ și unul sau doi biți de STOP cu valoarea logică ‘1’) pentru transmiterea unui caracter de lungime variabilă (5-8 biți de date). În această categorie se încadrează interfața UART (Universal Asynchronous Receiver/Transmitter).

Interfața USART (Universal Synchronous/Asynchronous Receiver/Transmitter) implementează ambele categorii, permițând utilizatorului să aleagă modalitatea dorită. O altă modalitate de clasificare este după direcția de transfer al informației și se disting următoarele categorii: [5]

* **Simplex**: Datele se pot transfera într-o singură direcție, de la emițător la receptor. Necesită un singur fir de tranmisie.
* **Half-Duplex**: Fiecare echipament terminal de date funcționează alternativ ca emițător, iar apoi ca receptor. Fiind transmis un singur mesaj la un moment dat, este suficient un singur fir de transmisie.
* **Full-Duplex**: Datele se transferă simultan în ambele direcții, astfel fiind necesare două fire de transmisie.

## 2.2. Caracteristicile generale ale interfeței UART

Adesea, când se face referire la comunicarea serială se utilizează diferite denumiri: Serial Port, RS-232 Interface, COM Port. Însă denumirea unanim acceptată este **UART** (Universal Asynchronous Receiver/Transmitter). Aceasta este o interfață, care de obicei, transmite un octet la un moment dat utilizând un singur fir de transmisie. Fiind o interfață asincronă, nu este dependentă de un semnal de ceas. [1]

Când este vorba despre UART, trebuie definiți următorii termeni, care caracterizează această interfață:

* **Baud Rate**:Este rata de eșantionare, indică cât de rapid se transmit biții. Este un număr care reprezintă *numărul de biți transmiși pe secundă.* Valorile standard (cele mai uzuale) sunt: 2400, 4800, 9600, 19200, 38400.
* **Data Frame** (pachetul de date): Fiecare bloc de date transmis este trimis într-un pachet de biți. Pachetul începe cu un bit de START și se termină cu unul sau doi biți de STOP. Între acești biți sunt situați biții de date propriu ziși (5-8 biți). Opțional se pot include biți de paritate a datelor.
* **Data chunk**: Reprezintă biții de informație efectivă, care urmează să fie transmiși. Dimensiunea acestuia variază între 5-8 biți, iar endianness-ul trebuie prestabilit (LSB first sau MSB first).
* **Biți de sincronizare**: Sunt biții de START și STOP. Ei marchează începutul și sfârșitul unui pachet de date. Semnul de START este un singur bit (trecerea de la ‘1’ logic la ’0’ logic), iar semnalul de STOP poate varia de la unul la doi biți de ’1’ logic.
* **Biți de paritate**: Sunt folosiți pentru validarea datelor transmise. Paritatea poate fi *pară (even* – bitul de paritate are valoarea ‘0’ logic*)* sau *impară (odd* – ‘1’ logic*).* Pentru a calcula paritatea unui șir de biți, se aplică operatorul *sau exclusiv* între toți biții de date.

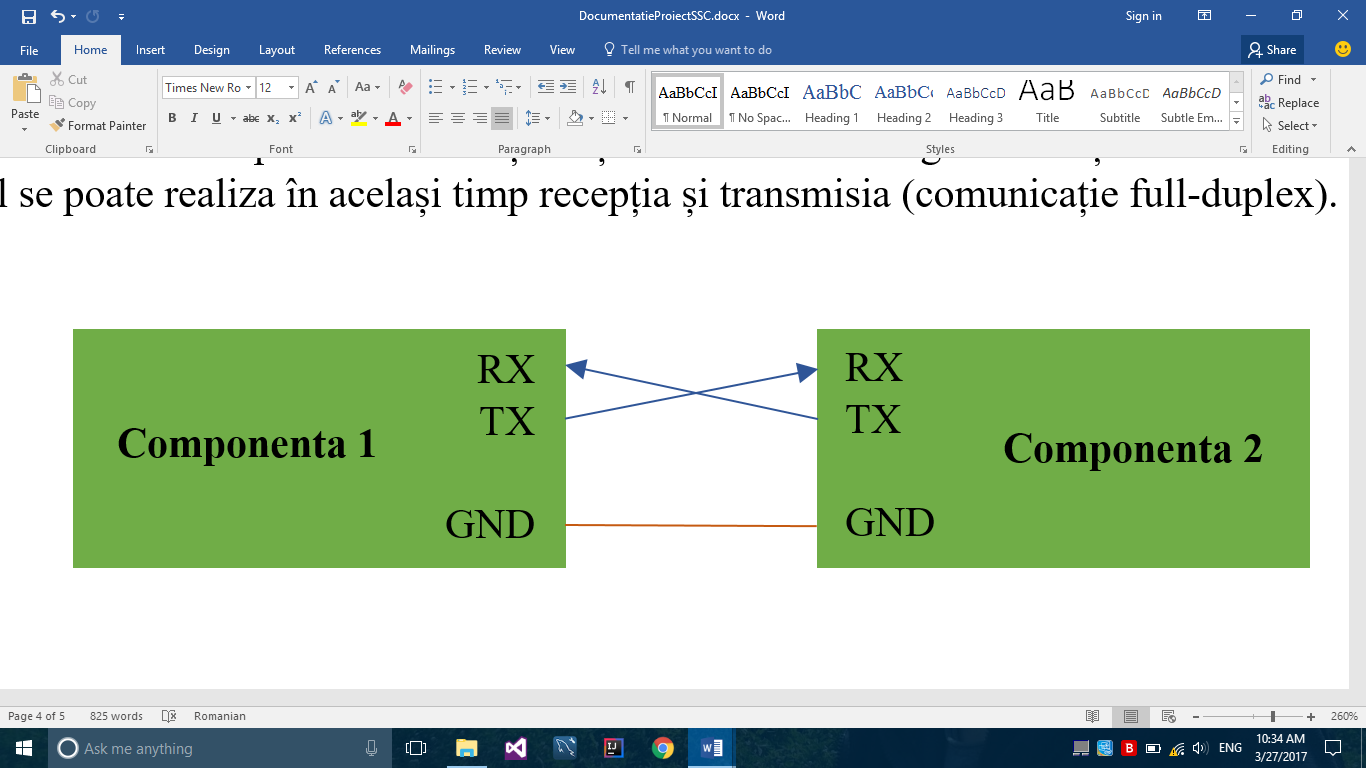


Figura 1. Conexiunile în interfața UART

**Figura 1** prezintă schema generală a conectării a două componente fizice care comunică prin interfața UART. Se conectează semnalul RX (recepție) al primei componente la TX-ul (transmisie) componentei cu numărul 2, astfel realizându-se comunicarea în direcția *Componenta 2 → Componenta 1.* Simetric, prin conectarea semnalului TX al primei componente cu RX-ul componentei cu numărul 2 se realizează comunicarea în direcția *Componenta1 → Componenta2.*  Pentru asigurarea referinței commune al nivelelor logice, se interconectează *ground*-urile celor două componente.

## 2.3. Interfața I2C

Interfața I2C (Inter-Integrated Circuit) este o interfață de comunicație serială, half-duplex dezvoltată de firma Philips în anii 1980. Comunicația se realizează printr-o magistrală bidirecțională cu două linii:

* SCL (Serial Clock) – linia pe care se aplică semnalul de ceas
* SDA (Serial Data) – linia pe care se efectuează transferul de biți

Fiecare dispozitiv conectat la magistrala I2C are o adresă unică de 7 biți. Distingem două tipuri de dispozitive: master și slave. Un schimb de informații poate fi inițiat doar de către un dispozitiv master. Deoarece pot exista mai multe dispozitive master interconectate, în cazul în care acestea doresc începerea unei transmisii în același moment de timp se aplică o procedură de arbitraj. La versiunea inițială a interfeței au existat două moduri de comunicație: **modul normal** realizându-se printr-o frecvență de ceas de 100 kHz și **modul rapid** (fast mode) la frecvența de 400 kHz.

SDA și SCL au valoarea logică ‘1’ atunci când nu se execută nici un transfer (semnale conectate cu colectorul în gol). O transmisie începe printr-un **bit de start** (condiție de start) și se termină printr-un **bit de stop** (condiție de stop) ilustrat în **Figura 2**. Un bit de start este o trecere al lui SDA de la ‘1’ la ‘0’ cât timp SCL este menținut pe valoarea ‘1’. Bitul de stop este o tranziție al lui SDA de la ‘0’ la ‘1’ cât timp SCL este menținut pe valoarea ‘1’.

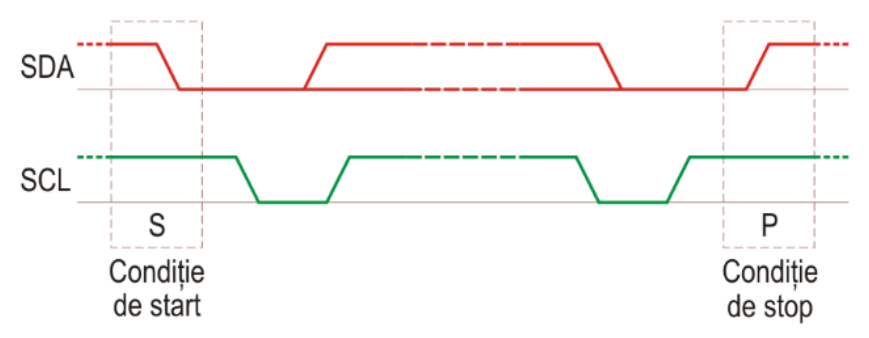


Figura 2. Biții de start și stop pe magistrala I2C [5]

După bitul de start efectuat de către dispozitivul master, acesta trebuie să genereze semnalul de ceas. SDA trebuie să conțină valoarea bitului transmis cât timp SCL este ‘1’ logic. Deci când SCL este ‘0’ se schimbă valoarea pe SDA (corespunzător bitului următor destinat transmisiei) și acesta este menținut cât timp SCL este ‘1’ logic. Astfel la fiecare ciclu de ceas se transmite câte un bit.

În continuare se prezintă modul de citire de la un dispozitiv slave. În primul rând, trebuie să punem pe magistrală adresa slave-ului. Dispozitivele slave conectate la magistrala I2C vor citi adresa, iar slave-ul care își recunoaște adresa proprie va semnala printr-un bit de *Acknowledge* (ACK). Transmisia începe prin bitul de start urmată de generarea semnalului de ceas de către dispozitivul master. Acesta transmite cei 7 biți de adresă și un bit de (‘1’ pentru citire sau ‘0’ pentru scriere). După acest bit, master-ul continuă generarea semnalului de ceas (SCL) dar eliberează semnalul SDA pentru ca slave-ul să transmită bitul de *Acknowledge* (ACK). Dacă acesta este ‘0’, transmisia poate continua. În caz contrar avem o erare și transmisia trebuie întreruptă cu un bit de stop. Transmiterea adresei este reprezentată în **Figura 3**. Linia SCL este controlată de dispozitivul master în întregime, iar culoarea verde indică controlul master-ului asupra liniei SDA, respectiv culoarea portocalie marchează controlul dispozitivului slave.

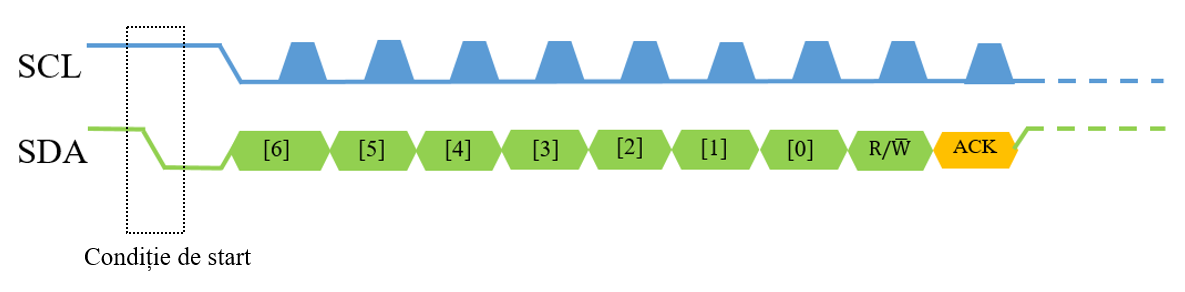


Figura 3. Transmiterea adresei și răspunsul de Acknowledge

În caz concret, la citire după transmiterea adresei, se transmite un bit de R/ cu valoarea ‘0’ logic. Dacă nu apare o erare de *Acknowledge* masterul trebuie să trimită adresa registrului intern de la care dorește să citească (această adresă are 8 biți). După transmiterea adresei urmează un nou bit de start și retrimiterea adesei slave-ului, însă de această dată cu bitul R/setat la ‘1’ logic. În continuare master-ul trebuie să genereze semnalul de ceas, dar fiind vorba de citire slave-ul va scrie pe SDA și master-ul va citi și va pune bitul de ACK. După finalizarea transmisiei de biți de date (în pachete de câte 8 octeți urmați de un bit de ACK), masterul trebuie să încheie transmisia cu ultimul bit de ACK cu valoarea ‘1’ logic (numit deseori NACK) și un bit de stop. Procesul de citire este reprezentat în **Figura 4**. Semnalul SCL este generat doar de către dispozitivul master. Semnalul SDA este controlat atât de master cât și de slave: culoarea verde reprezintă momentele de timp când dispozitivul master controlează linia, iar culoarea portocalie indică controlul slave-ului. În realizarea figurii, s-a presupus că slave-ul are adresa 4B16, se transmite comanda de citire al registrului intern cu adresa 0016, iar răspunsul este pe 8 biți (având valoarea de EA16). Acesta este doar un exemplu, există și dispozitive care dau răspuns pe 16 biți (transmiși prin mesaje de 8 biți urmate de bitul ACK).

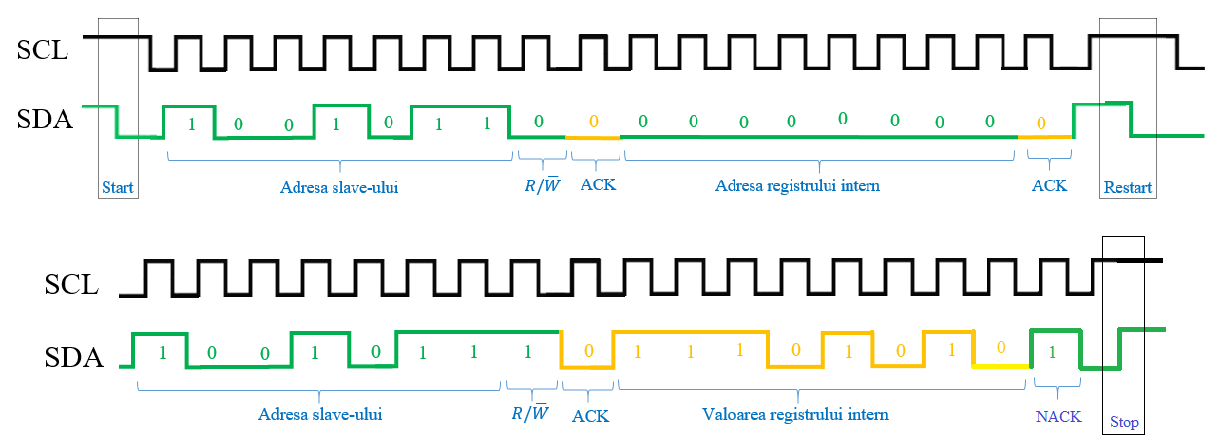


Figura 4. Citirea prin interfața I2C

## 2.4. Obiective

Pentru ilustrarea funcționării și a unui caz de utilizare al circuitului de comunicare UART, acesta va fi folosit pentru trimiterea temperaturii de la placa de dezvoltare FPGA la calculator. Temperatura se măsoară folosind senzorul de temperatură aflat pe placa Nexys4DDR. Temperatura măsurată se trimite folosind circuitul UART la calculator, pe care rulează un program care preia datele trimise și le afișează pe ecran. Prin intermediul programului care rulează pe calculator, utilizatorul poate să aleagă o temperatură dorită. Calculatorul transmite comanda utilizatorului către FPGA, care va efectua acțiunea corespunzătoare (încălzire, răcire sau idle).

Proiectul este realizat folosind mediul de dezvoltare Vivado Design Suite și limbajul de descriere hardware VHDL. Circuitul realizat prin implementarea proiectului VHDL este încărcat pe plăcuța de dezvoltare FPGA Nexys4DDR. Legarea celor două componente se va realiza utilizând un cablu USB. De menționat faptul că, la placa de dezvoltare Nexys4DDR pentru comunicare serială putem utiliza același cablu USB prin care se face programarea plăcii și depanarea programelor.

În **Figura 5** se prezintă portul J6 al plăcii Nexys4DDR, care are rol dublu în funcționarea plăcii de dezvoltare, după cum este specificat în manualul de referință al acesteia: în primul rând tool-urile Xilinx în mod tipic comunică prin placa de dezvoltare prin metoda JTAG. În timpul programării JTAG, fișierele *.bit* se transferă de la calculator către placa FPGA, astfel realizând încărcarea programelor; în același timp, tot acest port include și un *FTDI FT2232HQ USB-UART bridge* care permite comunicarea dintre aplicațiile pe calculator și FPGA, utilizând comenzi standard Windows COM. Funcțiile USB-UART și USB-JTAG, cu toate că utilizează același port al plăcii de dezvoltare, sunt complet independente una față ce cealaltă. Astfel se poate realiza comunicarea serială dintre calculator și FPGA utilizând același cablu prin care se face și programarea plăcii fără să se țină cont că ar exista interferențe între cele 2 controllere.

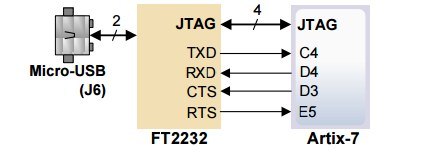


Figura 5. Nexys4 DDR Micro-USB connection [2]

În continuare se prezintă fundamentele teoretice pe care se bazează cele două module de transmisie și recepție a portului serial și se face o introducere în comenzile Windows COM standard și celelalte componente utilizate în cadrul aplicației cu sarcina de a comunica cu placa de dezvoltare FPGA. Apoi, se descriu etapele parcurse în faza de proiectare și implementare, urmând ca prin rezultatele experimentale să ajungem la concluzia proiectului.

# 3. Fundamentare teoretică

Interfața UART are la bază două module asemănătoare pentru recepția, respectiv transmisia de date. Aceste componente pot fi implementate ca și automate/mașini cu stări finite (Finite State Machine). În continuare se vor descrie automatele atât pentru recepție, cât și pentru transmisia de date. În descrierea componentelor considerăm următoarea structură a pachetului de date: 1 bit de START, 8 biți de date, fără bit de paritate și 1 bit de STOP. În cadrul segmentului de date, ordinea de transmitere a biților este de la cel mai puțin semnificativ la cel mai semnificativ (LSB first) ilustrată în **Figura 6**.

Figura 6. Pachetul de date transmis prin UART

Bit de start

Biții transmiși

Bit de stop

70]0]

61]

[2]

[3]

[4]

[5]

[6]

[7]

**‘0’**

**‘1’**

[0] [1] [2] [3] [4] [5] [6] [7]

## 3.1. FSM pentru transmisie de date serială

Automatul se află în starea *idle* cât timp nu apare un semnal, care indică începutul unei transmisii de date. La apariția semnalului de început al transmisiei, se trece în starea *start* și pe linia TX se pune bitul de start. În continuare se trimite octetul cu informațiile utile, repetând transmiterea de un bit de 8 ori, în starea *bit*. După ce datele au fost transmise, mai rămâne să se transmită și bitul de sincronizare de la sfârșitul pachetului, bitul de *stop,* după ce se trece din nou în starea *idle*, și se așteaptă începutul următoarea transmisiuni. Stările și tranzițiile automatului cu stări finite se prezintă în **Figura 7**.

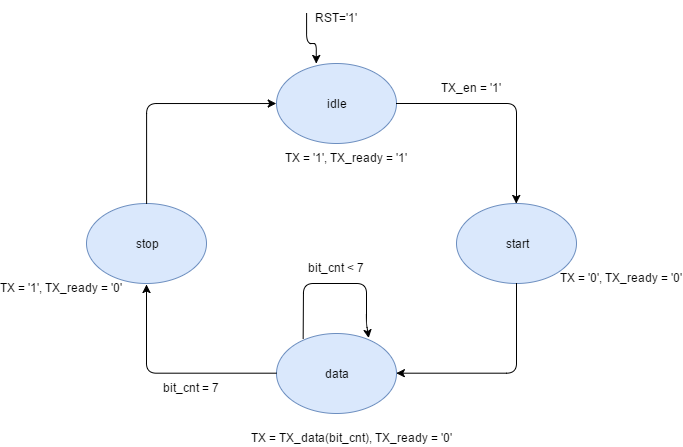


Figura 7. FSM pentru transmisie serială

## 3.2. FSM pentru recepție de date serială

Pentru a recepționa în mod corect datele de pe linia serială, semnalul trebuie citit și eșantionat de pe linia serială, cunoscând rata de eșantionare al dispozitivului sursă. În cazul recepției se folosește supra-eșantionarea, adică cunoscând rata de eșantionare a sursei, semnalul de pe linia serială este citit la o rată mai mare decât baud rate-ul sursei. Astfel, se poate realiza detecția mijlocului bitului de start, și în continuare, eșantionarea se va face aprx. la mijlocul intervalului, unde semnalul este stabil cu certitudine. Cea mai folosită de supra-eșantionare este folosirea unei rate la recepție de 16 ori mai mare decât rata de eșantionare a sursei.

În **Figura 8** se prezintă automatul cu stări finite pentru recepție serială. Dacă pe linia serială apare semnal de recepție se trece din starea *idle* în starea *start*. În această stare se așteaptă până când BAUD\_CNT devine 7, adică se ajunge la mijlocul bitului de start. În continuare, se trece în starea *bit,* unde similar stării precedente, se citește mijlocul fiecărui bit recepționat, până când se înregistrează toți cei 8 biți de date. Se trece în starea *stop*, unde se așteaptă un interval de timp, în care ajunge un bit pe linia serială. Starea *wait* este esențială, pentru că mai este nevoie să așteptăm o jumătate de bit, fiindcă numărarea a fost începută de la mijlocul bitului de START. Apoi se trece în starea *idle*, așteptând să se repete procedura de recepție.

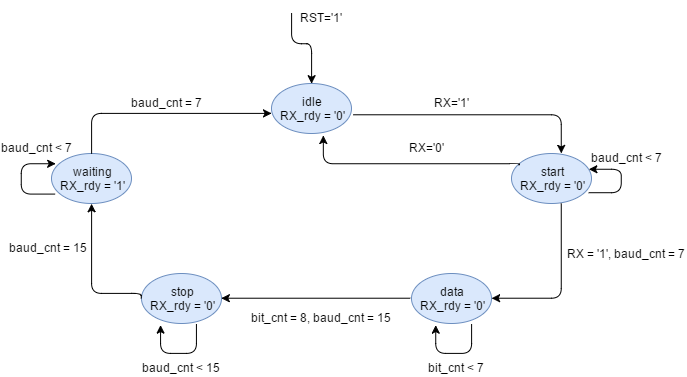


Figura 8. FSM pentru recepție serială

## 3.3. Standardul RS-232

RS-232 (Recommend Standard number 232) este un standard de comunicare serială folosită la calculatoarele IBM PC. Acesta a fost elaborată pentru realizarea comunicației digitale între: [5]

* un calculator și un terminal aflat la distanță
* două terminale fără utilizarea unui calculator

Acest standard definește atât un mod sincron cât și un mod asincron de comunicare. Deși legătura de bază necesită doar 3 fire de conexiune (recepție - RxD, transmisie - TxD și masa electrică - GND), cele mai multe dispozitive conțin și alte semnale pentru controlul fluxului de date. Configurația pinilor unui port care respectă standardul RS-232 este ilustrată în **Figura 9**.

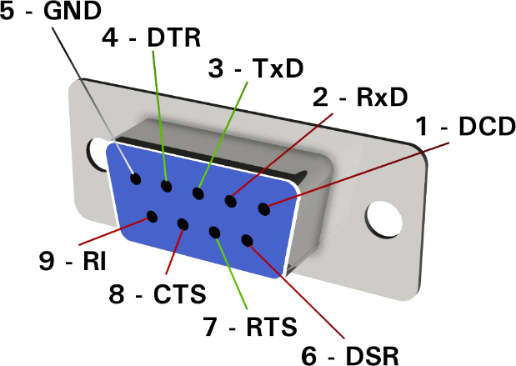


Figura 9. Configurația pinilor RS-232 [6]

## 3.4. FSM pentru interfața I2C

Controlerul de I2C este implementat cu ajutorul unui FSM (Finite State Machine). Diagrama de stări a FSM-ului este vizibilă în **Figura 11**, iar intrările și ieșirile sunt vizibile în **Figura 10**.

I2C\_Master primește la intrare un semnal de ceas (cu frecvența data de un generic), un semnal de reset asincron, un semnal pentru pornirea transmisiei (*Start*), un bit de R/ (‘1’ pentru citire și ‘0’ pentru scriere), adresa slave-ului pe 7 biți și datele care trebuie transmise (*Data\_in* este luat în considerare doar dacă avem o operație de scriere).

I2C\_Master produce la ieșire un bit de *Finish* (se setează atunci când se citește/scrie bitul de *Acknowledge*), *Request* care se activează înainte de citirea bitului de *Acknowledge* (pentru a notifica controllerul pentru a da comandă de restart), *Rdy* se setază în starea *ready*, *Ack\_Error* este setat dacă se ajunge la o eroare de *Acknowledge*, *Data\_out* conține datele citite de la slave (stabil atunci când *Finish* este ‘1’ logic). Se generează și cele două linii ale magistralei I2C: SCL și SDA.

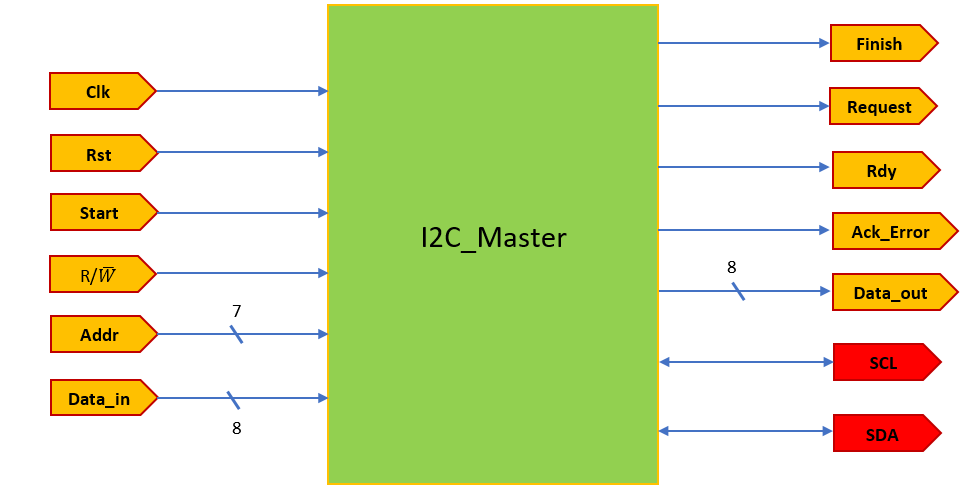


Figura 10. Controllerul I2C\_Master



ready

Figura 11. FSM pentru modulul I2C\_Master

## 3.4. Senzorul de temperatură al plăcii FPGA Nexys4DDR

Placa de dezvoltare FPGA Nexys4 DDR dispune de un dizpozitiv analogic pentru măsurarea temperaturii ambientale ADT7420. Senzorul poate măsura temperatura cu o rezoluție de pâna la 16 biți cu o acuratețe tipică de aproximativ *0,25 grade Celsius*. Transmitirea datelor dintre senzor și FPGA se realizează utilizând comunicație serială sincronă *I2C*. În **Figura 12** sunt prezentate legăturile dintre FPGA și senzor. Odată citită valoarea măsurată de senzor (valoare întreagă cu semn, pe 16 biți reprezentată în complement față de 2) aceasta trebuie deplasată la dreapta cu 3 poziții (deplasare aritmetică – se ia în considerare și semnul) și înmulțită cu 0.0625. Valoarea rezultată este temperature în grade Celsius.

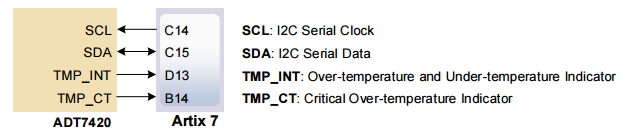


Figura 12. Interfața de comunicare FPGA - Senzor de temperatură [2]

## 3.5. Comenzi MS-DOS pentru portul serial

Pentru a realiza comunicarea dintre un calculator și FPGA, trebuie să fie cunoscută modalitatea prin care calculatorul comunică utilizând portul serial. Putem crea conexiuni seriale prin utilizarea interfeței *command line.* Sub sistemul de operare Windows, avem nevoie de următoarele comenzi[4]:

* *powershell* : pentru a intra în modul PowerShell
* *[System*.*IO.Ports.SerialPort]::getportnames() :* vizualizarea listei cu porturile COM active
* *$port= new-Object System.IO.Ports.SerialPort COM#,Baudrate,None,8,one :* inițializarea unui port nou cu caracteristicile specificate
* *$port.open() / $port.close() :* conectare/deconectare port
* *$port.WriteLine(“exemplu”) / $port.ReadLine() : scriere/citire date.*

# 4. Proiectare și implementare

În acest capitol se prezintă componentele proiectate și implementate pentru a realiza comunicația serială asincronă UART între placa de dezvoltare FGPA și calculator, respectiv comunicația folosint protocolul I2C între placa de dezvoltare FPGA și senzorul de temperatură, care se găsește atașat la placa FGPA Nexys 4 DDR.

În **Figura 13** se prezintă sistemul în ansamblu, cu principalele componente utilizate în cadrul acestui proiect. Sistemul poate fi împărțit în două module mai mari, modulul mai mic fiind inclus în cel mare. Primul modul (modulul principal) este responsabil pentru comunicația prin protocolul UART între placa de dezvoltare și calculator, cât și pentru legarea împreună a tutoror componentelor. Componentele esențiale din cadrul acestui modul sunt mașinile de stări finite pentru transmisia și recepția biților de date (TX\_FSM, respectiv RX\_FSM), controllerul pentru transmisia a doi octeți într-o singură tranzacție (TX\_Controller) generatorul de ratei de eșantionare și perifericele care permit interacția utilizatorului cu placa de dezvoltare. Modulul al doilea (I2C Controller) este responsabil pentru comunicația prin protocolul I2C dintre placa de dezvoltare și senzorul de temperatură incorporat în aceasta. Acest modul constă dintr-un controlor I2C, care controlează automatul I2C Master pentru a realiza conexiunea cu senzorul de temperatură. FPGA-ul ia rolul masterului, care cere informații în mod periodic (din 2 în 2 secunde). Senzorul de temperatură îndeplinește rolul slave-ului care, la cerere, furnizează informații despre temperatura ambientală curentă sub forma a doi octeți de date.

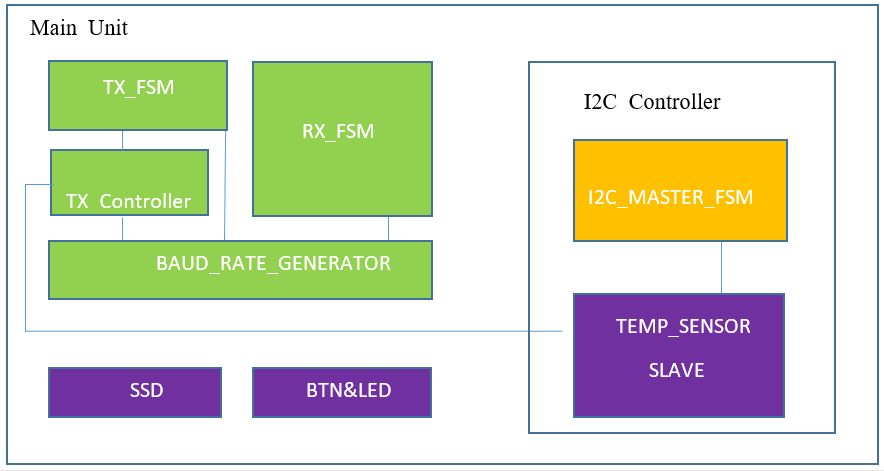


Figura 13. Schema în ansamblu al sistemului

În continuare se va prezenta în mod detaliat, specificând semnalele de intrare/ieșire și modul de implementare a componentelor, care fac parte din sistem prezentat anterior.

## 4.1. Baud Rate Generator

Figura 14. Schema bloc Baud Rate generator

IN: CLK

BAUD

RATE

GENERATOR

OUT: TX\_BAUD\_RATE

OUT: RX\_BAUD\_RATE

Componenta cu schema bloc prezentată în **Figura 14** este responsabilă pentru generarea a semnalelor de eșantionare corespunzătoare. Există o singură intrare în această componentă, semnalul de ceas al plăcii de dezvoltare (în cazul Nexys4DDR, semnalul de ceas este de 100 Mhz) și la ieșire apar două semnale de ceas pentru cele două componente din care este alcătuit sistemul de comunicație UART. Componenta încorporează două divizoare de frecvență pentru cele 2 semnale de ieșire. Pentru transmisie, implementarea în proiectul curent presupune că este nevoie de un baud rate de 9600 bit/s, iar în cazul recepției, acest semnal trebuie multiplicat de 16 ori. Pentru a obține o rată de eșantionare de 16\*9600 bit/s, divizorul de frecvență trebuie să fie format dintr-un numărător, care numără 100.000.000 /16\*9600 = apr. 651 de tacți.

## 4.2. FSM pentru transmisie UART



Figura 15. Schema Bloc TX\_FSM

Componenta TX\_FSM (**Figura 15**) este masina de stări finite responsabilă pentru transmisia biților de date de la placa de dezvoltare către calculator. FSM-ul este contruit pe baza diagramei prezentate la capitolul de Fundamentare teoretică. Semnalul Rst readuce sistemul în starea de repaus (idle). Începe o nouă transmisie în momentul în care TX\_EN are valoarea logică ‘1’. Acesta înseamnă că automatul iese din starea de repaus și începe transmisia biților la rata de eșantionare dată se semnalul de intrare BAUD\_EN. La portul de Clk se leagă semnalul de ceas intern al plăcii și este utilizat pentru sincroniza toate tranzițiile. Semnalul TX\_READY se activează în momentul în care tranziția curentă este terminată și se așteaptă începerea unei noi tranziții, adică automatul se află în starea *idle*. Pe semnalul de ieșire se pun biții corespunzători pachetului de date al protocolului UART, adică bitul de start, biții de date și bitul de stop.

## 4.3. FSM pentru recepția UART

IN: RST

IN: RX

IN: CLK

RX\_FSM

OUT: RX\_DATA

OUT: RX\_READY

IN: BAUD\_EN

Figura 16. Schema bloc RX\_FSM

Componenta RX\_FSM (**Figura 16**) este mașina de stări finite responsabilă pentru recepția biților de date care provin de la dispozitivul cu care este legată placa de dezvoltare. Implementarea componentei are la bază diagrama prezentată în capitolul de Fundamentare Teoretică. Semnalul BAUD\_EN provine de la componenta Baud Rate Generator, reprezintă semnalul de eșantionare la recepție, care este de 16 ori mai mare decât rata de eșantionare la transmisie. Semnalul RST readuce sistemul în starea *idle*. Pe portul semnalului RX sunt plasate biții de date care provin de la transmițător. Acest port este citit la momente corespunzătoare și informațiile citite sunt salvate într-un registru intern. După terminarea unei citiri, conținutul registrului se pune pe portul de ieșire RX\_DATA și semnalul RX\_READY devine ‘1’ logic, indicând sfârșitul tranzacției și posibilitatea începerii unei noi recepții.

# 5. Rezultate experimentale

## 5.1. Testarea TX\_FSM UART

În **Figura 17** se prezintă formele de undă generate prin rularea unui banc de test, pentru a testa funcționalitatea automatului cu stări finite pentru transmisia de date. Se poate observa modalitatea în care se modifică portul de ieșire TX al sistemului, luând pe rând valorile biților din vectorul TX\_DATA, începând cu bitul cel mai puțin semnificativ. Deasemenea, se observă comportamentul semnalului TX\_READY: înaintea începerii transmisiei și după terminarea acesteia are valoare logică 1, în rest este adusă la 0 logic.

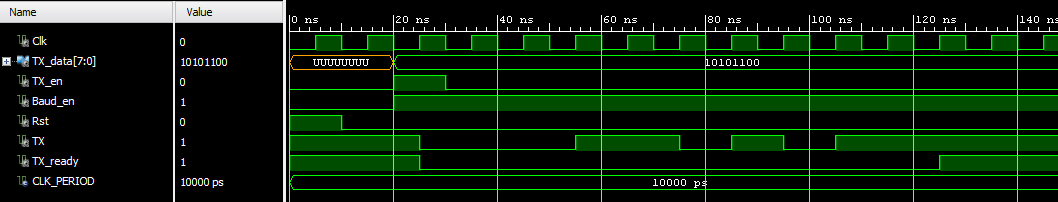


Figura 17. Simulare transmisie date prin TX\_FS

## 5.2. Simularea modulului RX\_FSM UART

În **Figura 18** se prezintă formele de undă generate prin rularea unui banc de test, pentru a testa funcționalitatea automatului cu stări finite pentru recepția de date. Se observă faptul că inițial automatul se află în starea idle, iar la tranziția de la ‘1’ la ‘0’ pe portul RX automatul intră în starea *start*. Din această stare trece în starea în care se preiau datele efectiv, utilizând o rată de eșantionare de 16 ori mai mare decât la transmisie. Datorită așteptării introduse în starea *start*, datele de pe portul RX sunt citite aproximativ la mijlocul intervalului, undele datele sunt stabile cu siguranță. Acest aspect este ilustrat prin liniile albastre verticale. După recepția celor 8 biți de date, mai este recepționat și bitul de STOP, după care automatul intră într-o stare de *waiting* prin care se corectează așteptarea introdusă la începutul recepției, în starea *start.* În această stare se activează și ieșirea RX\_READY semnalând sfârșitul recepției. Pe portul de ieșire RX\_data se poate observa apariția șirului de biți “10101100”, care a fost recepționat. Automatul intră în starea *idle,* așteptând începerea unei noi recepții.

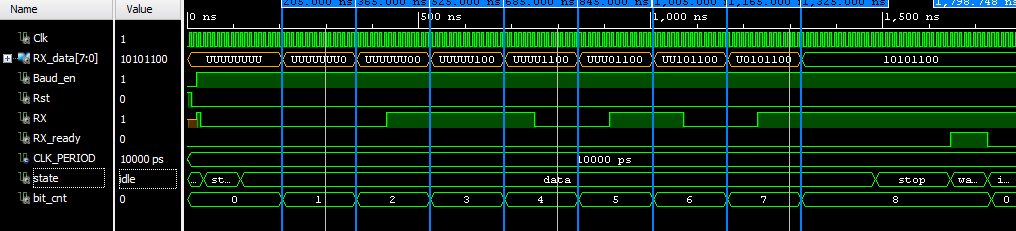


Figura 18. Simularea recepției UART

## 5.2. Simularea modulului I2C\_Master

În **Figura 19** se poate observa inițierea transmisiei I2C prin setarea semnalului *START\_RD*. Astfel modulul de control pentru I2C\_Master începe transmisia printr-un bit de start, urmat de generarea semnalului de ceas *SCL* și transmiterea adresei slave-ului. Datele se modifică atunci când *SCL = ‘0’* și se citesc apriximativ la jumătatea intervalului de timp când *SCL = ‘1’*. După transmiterea adresei (7 biți) se transmite și bitul *R/.* După transmiterea acestor biți se eliberează *SDA* dar se generează în continuare *SCL* și se citește bitul de *Acknowledge*. Dacă bitul citit este ‘0’ atunci transmisia poate continua, dar dacă este ‘1’ atunci transmisia trebuie oprită (slave-ul nu a recepționat datele).

În **Figura 20** este reprezentată repetarea bitului de start, urmată de transmiterea adresei slave-ului a doua oară, dar de această dată cu *R/W = ‘1’*.Se eliberează apoi linia *SDA* pentru ca slave-ul să poată trimite bitul de *Acknowledgement*.

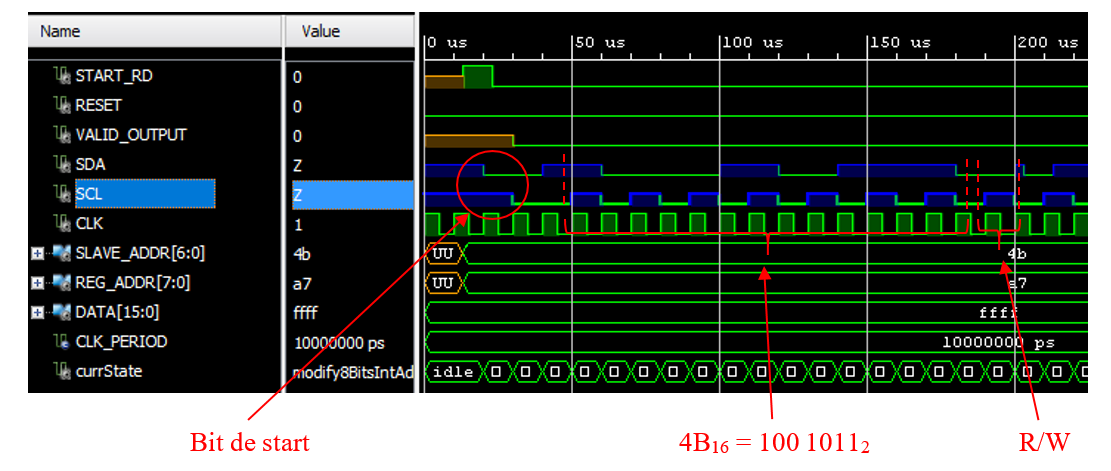


Figura 19. Transmiterea adresei slave-ului pe magistrala I2C

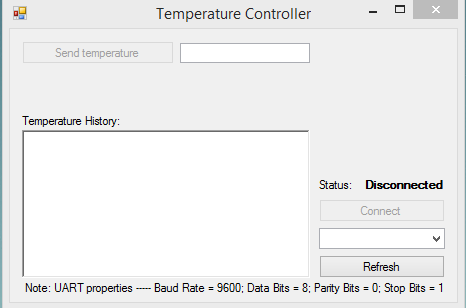


Figura 20. Repetarea bitului de start și retransmiterea adresei slave-ului

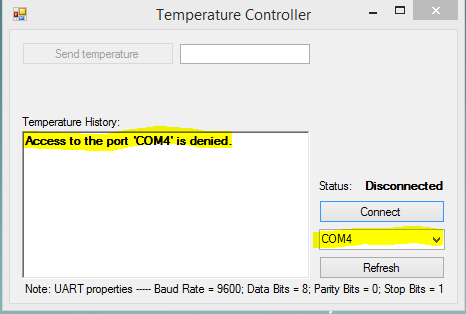
## 5.4. Testarea aplicației pe calculator

În prima fază a testării aplicației pe calculator scris în limbajul C#.NET utilizând Windows Forms, se testează transmisia de date de la placa de dezvoltare către calculator, apoi și recepția datelor.

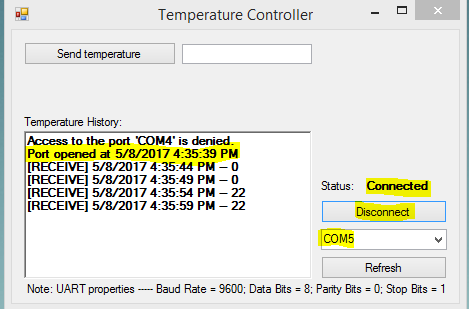
Inițial când se deschide aplicația mai multe opțiuni sunt dezactivate după cum se vede în figura de mai jos, fiindcă pornirea comunicației necesită câțiva pași de setare. Se poate observa că programul nu este conectat cu niciun dispozitiv, starea conexiunii fiind **Disconnected.**



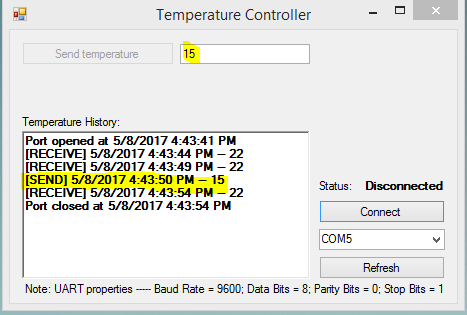
Apăsând butonul *Refresh,* se vor afișa porturile seriale disponibile. Observăm că butonul se vor afișa porturile seriale disponibile. Observăm că butonul *Connect* devine activ. Cu acest buton încercăm conectarea la un port serial. Când se încearcă conectarea la un port nepotrivit se observă apariția unui mesaj de informare, indicând faptul că conexiunea la portul selectat nu este posibilă.



Selectăm alt port disponibil și repetăm operația precedentă. Când operația se efectuează cu succes se observă mai multe modificări care apar în program. În primul rând starea sistemului devine **Connected**, iar butonul cu care mai înainte am efectuat operația de conectare, acum poate efectua operația de întrerupere a conexiunii. Se afișează mesajul care confirmă conectarea cu succes la dispozitivul conectat la portul selectat. Deasemenea, observăm că datele se recepționează periodic. Acesta se datorează faptului că unitatea principală din cadrul proiectului pus pe placa de dezvoltare conține un proces care trimite date prin modulul de transmisie într-un mod regulat (din 5 în 5 secunde).

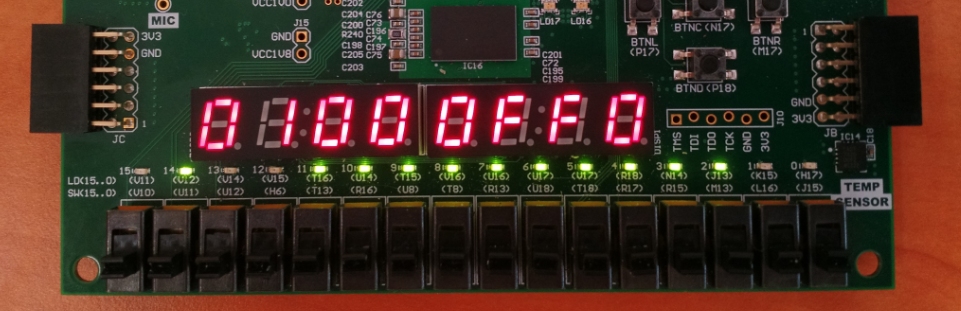


Transmisia de date de la aplicație către placa de dezvoltare se realizează utilizând componentele grafice care apar în partea de sus a aplicației. În *TextBox* introducem o valoare întreagă pozitivă care încape pe 1 byte (0-255) și apăsând butonul *Send temperature* acesta se va trimite către placa de dezvoltare. La transmisie reușită se afișează un mesaj de confirmare în căsuța de istoric.

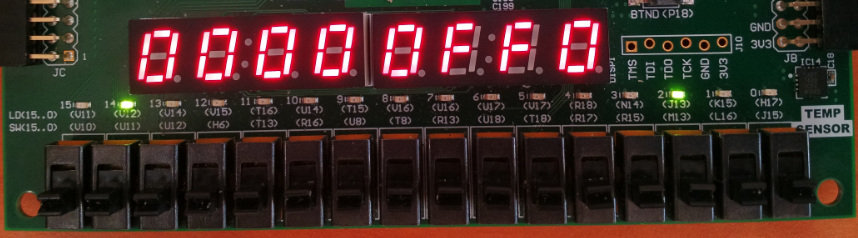


După legarea componentelor implementate în cadrul acestui proiect se testează corectitudinea interpretarea corectă a datelor. Fiind vorba despre un controlor de temperatură, calculatorul trebuie să transmită plăcii de dezvoltare FPGA operația corectă de efectuat: încălzire, răcire, repaus și placa să interprezete în mod corect aceste comenzi. Pentru exemplificarea operațiilor care se pot efectua se utilizează LED-urile plăcii FPGA. Astfel, pentru situația în repaus, se ilumnează un singur LED(2). Pentru operația de încălzire ledurile LED(2-11) se vor ilumina în mod permanent. Pentru operația de răcire ledurile menționate anterior vor pâlpâi.

Operațiile de efectuat se transmit de la calculator către placa de dezvoltare codificat pe 2 biți astfel: 00b – repaus, 01b – încălzire, 10b – răcire. În imaginea de mai jos, primele două cifre din partea stângă de pe afișajul de 7 segmente reprezintă octetul recepționat. Se poate observa că, în cazul recepției a numărului 1 (01b), ledurile LED(2-11) se iluminează.



Când numărul 0 este recepționat rămâne aprins doar LED(2), care indică oprirea sistemului de control al temperaturii.



În partea dreaptă a afișajului de 7 segmente se afișează cei doi octeți primiți de la senzorul de tempatură. Acest număr necesită o prelucrare pentru a indica temperatura în grade Celsius interpretabil de către operatorul uman. Aceată prelucrare se realizează în aplicația pe calculator.

# Concluzii

În prezentul proiect s-au implementat cu succes două protocoale de comunicare des întâlnite: protocolul UART și I2C. Înaintea implementării propriu zise, a fost necesară o înțelegere foarte clară a task-ului de rezolvat și înțelegerea modalității de funcționare a diferitelor protocoale de comunicare. Pe baza diagramelor de tranziție între stări care descriu automatele de stări finite ale componentelor de comunicare, s-au implementat modulele care realizează comunicarea și controlere pentru aceste module, pentru a accesa sau a transmite datele dorite.

După dezvoltarea modulelor pe placa de dezvoltare, s-a implementat și o aplicație pe calculator, prin care se demonstrează funcționarea corectă a componentelor. Pentru a demonstra corectitudinea și aplicabilitatea proiectului, un exemplu de controlor de temperatură a fost dezvoltată. Acest exemplu interconectează toate modulele dezvoltate în cadrul acestui proiect. Manualul de utilizare pentru aplicația atașată proiectului se prezintă în Anexa A.

Pentru a realiza controlul efectiv al temperaturii ambientale și nu numai exemplificarea operațiilor efectuabile, sistemul s-ar putea fi extins cu alte componente periferice: un reșou pentru a încălzi incinta, și un ventilator pentru a răci camera respectivă. Comunicația cu aceste componente periferice s-ar putea realiza cu un alt protocol de comunicare, de exemplu SPI.

# Bibliografie

[1] What is a UART?

<https://www.nandland.com/articles/what-is-a-uart-rs232-serial.html>

[2] Nexys4-DDR Reference Manual <https://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPNexys4DDR/documentation/Nexys4-DDR_rm.pdf>

[3] Arhitectura Calculatoarelor: Lucrări de laborator, Florin Oniga, UTCN, 2016

<http://users.utcluj.ro/~onigaf/files/AC.html>

[4] Serial Terminal Basics

<https://learn.sparkfun.com/tutorials/terminal-basics/command-line-windows-mac-linux>

[5] Sisteme de Intrare-Ieșire: Lucrări de laborator, Baruch Zoltan-Francisc

<http://users.utcluj.ro/~baruch/sie/labor/Port-Serial.pdf>

[6] Ethernet RS-232 Primer

<http://www.ethernut.de/en/documents/rs232primer.html>

[7] Controller I2C Master

<https://eewiki.net/pages/viewpage.action?pageId=10125324>

# Anexe

# Anexa A (Manual de utilizare al aplicației de control al temperaturii)

Pentru a rula programul implementat se parcurg pașii următori:

1. Se introduce CD-ul atașat și se copiază pe calculator proiectul Vivado 2016.4, cât și proiectul soluției Visual Studio.
2. Se deschide cu Vivado 2016.4 proiectul *proiectUARTNexys4DDR.*
3. Se conectează placa de dezvoltare Nexys4DDR la calculator.
4. Se pornește placa FPGA și se încarcă pe ea fișierul *bitstream* generat din proiectul deschis.
5. Se deschide soluția Visual Studio *ProjectUARTControl.sln* și se pornește o instanță a programului, sau se rulează direct executabilul *ProjectUARTControl.exe* din folderul ProjectUARTControl/bin/Debug.
6. După deschiderea aplicației se apasă pe butonul de *Refresh*, se selectează portul serial corespunzător și se conectează la placa FPGA apăsând butonul *Connect.* Dacă operația se efectuează cu succes, programul ar trebui să recepționeze în mod periodic datele citite de la senzorul de temperatură.
7. În *textboxul* de sus, se introduce o temperatură întreagă dorită și urmărește comportamentul ledurilor de pe placa FPGA în funcție de *System Status* în aplicația pe calculator.
8. Se închide programul pe calculator, se oprește placa de dezvoltare și se deconectează de la calculator.

# Anexa B (Codul sursă al componentelor semnificative)

În continuare se prezintă codul sursă VHDL care descriu cele mai semnificative componente din cadrul proiectului, adică mașinile de stări finite ale modulelor I2C\_Master, TX\_FSM, RX\_FSM.

--I2C Master FSM

process(Clk, Rst)

begin

if (Rst = '1') then

state <= ready;

Int\_Ack\_Error <= '0';

elsif (rising\_edge(Clk)) then

case state is

when ready =>

if (CE\_r = '1') then

if (Start = '1') then

state <= start\_bit;

Addr\_RW\_reg <= Addr & RW;

Data\_reg <= Data\_in;

end if;

end if;

when start\_bit =>

if (CE\_f = '1') then

Int\_Ack\_Error <= '0';

state <= command;

Bit\_Cnt <= 7;

Shift\_reg <= Addr\_RW\_reg;

end if;

when command =>

if (CE\_f = '1') then

if (Bit\_Cnt = 0) then

state <= slv\_ack1;

end if;

Bit\_Cnt <= Bit\_Cnt - 1;

-- shift

Shift\_reg(7 downto 1) <= Shift\_reg(6 downto 0);

Shift\_reg(0) <= SDA;

end if;

when slv\_ack1 =>

if (CE\_f = '1') then

if (SDA /= '0') then

Int\_Ack\_Error <= '1';

state <= stop1;

else

if (RW\_reg = '0') then

state <= wr;

Shift\_reg <= Data\_reg;

else

state <= rd;

end if;

Bit\_Cnt <= 7;

end if;

end if;

when rd =>

if (CE\_f = '1') then

if (Bit\_Cnt = 0) then

if (Start = '1') then

if (RW = '1' and Addr = Addr\_RW\_reg(7 downto 1)) then

state <= mstr\_ack\_rd;

else

state <= mstr\_ack\_restart;

Addr\_RW\_reg <= Addr & RW;

Data\_reg <= Data\_in;

end if;

else

state <= mstr\_nack;

end if;

end if;

Bit\_Cnt <= Bit\_Cnt - 1;

-- shift

Shift\_reg(7 downto 1) <= Shift\_reg(6 downto 0);

Shift\_reg(0) <= SDA;

end if;

when wr =>

if (CE\_f = '1') then

if (Bit\_Cnt = 0) then

state <= slv\_ack2;

end if;

Bit\_Cnt <= Bit\_Cnt - 1;

-- shift

Shift\_reg(7 downto 1) <= Shift\_reg(6 downto 0);

Shift\_reg(0) <= SDA;

end if;

when slv\_ack2 =>

if (CE\_f = '1') then

if (SDA /= '0') then

Int\_Ack\_Error <= '1';

state <= stop1;

else

if (Start = '1') then

if (RW = '0' and Addr = Addr\_RW\_reg(7 downto 1)) then

state <= wr;

Bit\_Cnt <= 7;

Data\_reg <= Data\_in;

Shift\_reg <= Data\_in;

else

state <= SDA1\_SCL0;

Addr\_RW\_reg <= Addr & RW;

Data\_reg <= Data\_in;

end if;

else

state <= stop1;

end if;

end if;

end if;

when mstr\_ack\_rd =>

if (CE\_f = '1') then

state <= rd;

Bit\_Cnt <= 7;

end if;

when mstr\_ack\_restart =>

if (CE\_f = '1') then

state <= SDA1\_SCL0;

end if;

when mstr\_nack =>

if (CE\_f = '1') then

state <= stop1;

end if;

when SDA1\_SCL0 =>

if (CE\_r = '1') then

state <= SDA1\_SCL1;

end if;

when SDA1\_SCL1 =>

if (CE\_r = '1') then

state <= start\_bit;

end if;

when stop1 =>

if (CE\_r = '1') then

state <= stop2;

end if;

when stop2 =>

if (CE\_f = '1') then

state <= ready;

end if;

when others =>

state <= ready;

end case;

end if;

end process;

Ack\_Error <= Int\_Ack\_Error;

Finish <= '1' when (state = slv\_ack2 or state = mstr\_ack\_rd or state = mstr\_ack\_restart or state = mstr\_nack) else '0';

Request <= '1' when ((state = rd and Bit\_Cnt = 0) or state = slv\_ack2) else '0';

Data\_out <= Shift\_reg;

Rdy <= '1' when (state = ready) else '0';

SCL\_Sel <= 0 when (state = SDA1\_SCL0 or state = stop1) else

1 when (state = ready or state = start\_bit or state = SDA1\_SCL1 or state = stop2) else 2;

Read\_Write <= '1' when (state = slv\_ack1 or state = rd or state = slv\_ack2) else '0';

SDA\_Sel <= 0 when (state = start\_bit or state = mstr\_ack\_rd or state = stop1 or

state = stop2 or state = mstr\_ack\_restart) else

1 when (state = ready or state = SDA1\_SCL0 or state = SDA1\_SCL1 or state = mstr\_nack) else 2;

--FSM for UART Transmission

process (Clk, Rst, Baud\_en, TX\_en)

begin

if RST ='1' then

state <= idle;

bit\_cnt <= 0;

elsif rising\_edge(Clk) then

if Baud\_en = '1' then

case state is

when idle =>

if TX\_en = '1' then

bit\_cnt <= 0;

state <= start;

else

bit\_cnt <= 0;

state <= idle;

end if;

when start =>

bit\_cnt <= 0;

state <= data;

when data =>

if bit\_cnt = 7 then

state <= stop;

else

bit\_cnt <= bit\_cnt + 1;

state <= data;

end if;

when stop =>

bit\_cnt <= 0;

state <= idle;

end case;

end if;

end if;

end process;

TX\_ready <= '1' when state = idle else '0';

TX <= '1' when state = idle or state = stop else

'0' when state = start else

TX\_data(bit\_cnt);

-- FSM for UART RECEPTION

process (Clk, Rst, Baud\_en)

begin

if Rst = '1' then

state <= idle;

elsif rising\_edge(Clk) then

if Baud\_en = '1' then

case state is

when idle =>

if RX = '0' then

state <= start;

else

state <= idle;

end if;

baud\_cnt <= 0;

bit\_cnt <= 0;

when start =>

if RX = '1' then

state <= idle;

elsif baud\_cnt = 7 then

state <= data;

else

state <= start;

baud\_cnt <= baud\_cnt + 1;

end if;

when data =>

if baud\_cnt = 15 then

if bit\_cnt = 8 then

state <= stop;

else

state <= data;

RX\_data(bit\_cnt) <= RX;

bit\_cnt <= bit\_cnt + 1;

end if;

baud\_cnt <= 0;

else

state <= data;

baud\_cnt <= baud\_cnt + 1;

end if;

when stop =>

if baud\_cnt = 15 then

state <= waiting;

baud\_cnt <= 0;

else

state <= stop;

baud\_cnt <= baud\_cnt + 1;

end if;

when waiting =>

if baud\_cnt = 7 then

state <= idle;

baud\_cnt <= 0;

else

state <= waiting;

baud\_cnt <= baud\_cnt + 1;

end if;

end case;

end if;

end if;

end process;

-- descriere valorilor semnalelor de iesire in functie de stare

RX\_ready <= '1' when state = waiting else '0';