Catedra de Calculatoare

Implementarea Protocoalelor de Comunicare I2C, SPI și Ethernet

Muresan Davide-Andrei
Grupa:8
Data: 18 decembrie 2024

Cuprins

Rezumat

Proiectul prezintă implementarea a trei protocoale de comunicare: I2C, SPI şi Ethernet. Obiectivul principal a fost dezvoltarea şi testarea acestora pe platforme embedded. Rezultatele obţinute demonstrează eficienţa implementării şi validarea funcţionalităţii.

1. Introducere

Contextul tehnologic al protocoalelor de comunicare joacă un rol crucial în implementările embedded moderne. În acest proiect, sunt analizate şi implementate trei protocoale: I2C, SPI şi Ethernet. Acestea permit comunicarea eficientă între module hardware şi software în sisteme embedded.

2. Fundamentare Teoretică

Această secţiune prezintă fundamentele teoretice ale protocoalelor de comunicare utilizate în proiect: I2C, SPI şi Ethernet. Sunt descrise modelele, metodele şi tehnologiile relevante, precum şi avantaje si dejavantaje.

2.1. I2C - Inter-Integrated Circuit

I2C este un protocol de comunicaţie serială sincronă, dezvoltat de Philips Semiconductor. Este utilizat pentru conectarea mai multor dispozitive pe o singură magistrală cu doar două fire: SDA (Serial Data Line) şi SCL (Serial Clock Line).

- Magistrală multi-master, multi-slave.
- **Viteză de transfer**: 100 kbps (Standard), 400 kbps (Fast), şi până la 3.4 Mbps (High-Speed).
- **Structură**: Fiecare dispozitiv are o adresă unică de 7 sau 10 biţi.
- **Utilizări**: Dispozitive periferice precum senzori, EEPROM-uri, convertoare ADC/DAC.

Avantaje: Utilizarea a doar două linii, uşor de implementat. *Dezavantaje*: Viteză redusă comparativ cu alte protocoale.

2.2. SPI - Serial Peripheral Interface

SPI este un protocol de comunicaţie serială sincronă dezvoltat de Motorola, folosit în comunicaţiile de mare viteză între un master şi unul sau mai mulţi slave.

- Master-Slave.
- **Linii de comunicaţie**: MISO (Master In Slave Out), MOSI (Master Out Slave In), SCLK (Serial Clock), SS (Slave Select).
- **Viteză de transfer**: Până la zeci de MHz.
- **Utilizări**: EEPROM, memorie Flash, afisaje LCD.

Avantaje: Viteză mare, implementare simplă. *Dezavantaje*: Necesită mai multe linii de comunicaţie.

2.3. Ethernet

Ethernet este un protocol standardizat pentru reţele locale (LAN), utilizând topologia star sau bus.

- **Viteză de transfer**: 10 Mbps (Ethernet), 100 Mbps (Fast Ethernet), 1 Gbps (Gigabit Ethernet).
- **Structură**: Comunicaţie bazată pe pachete de date şi adrese MAC.
- **Utilizări**: Conectarea dispozitivelor în reţelele locale.

Avantaje: Viteză mare, fiabilitate. *Dezavantaje*: Necesită echipamente suplimentare pentru implementare (switch, router).

3. Proiectare și Implementare

3.1. Implementare I2C

Protocolul I2C (Inter-Integrated Circuit) permite comunicarea serială între un Master și unul sau mai mulți Slave-i folosind două linii principale: `SCL` (Clock) și `SDA` (Data). Implementarea include componentele Master și Slave, fiecare având funcționalități dedicate pentru a gestiona operațiile I2C conform standardului.

3.1.1 Master I2C

Master-ul I2C controlează întregul proces de comunicație, inclusiv generarea semnalelor `SCL` și `SDA`. Funcționalitatea principală a componentei este gestionată printr-o mașină de stări finite (FSM) care urmează următorii pași principali:

- **Generare Start**: - Master-ul inițiază comunicarea setând linia `SDA` la nivel `LOW` în timp ce `SCL` este `HIGH`.
- **Transmiterea Adresei**: - Adresa Slave-ului este transmisă pe `SDA` bit cu bit, sincronizată cu `SCL`. - Ultimul bit indică operația: `0` pentru scriere, `1` pentru citire.
- **Transmiterea/Recepția Datelor**: - În cazul unei operații de scriere, datele sunt trimise pe `SDA`, sincronizate cu `SCL`. - În cazul unei operații de citire, Master-ul așteaptă date de la Slave și le stochează în `data_out`.
- **Generare Stop**: - Comunicarea se încheie prin setarea `SDA` la `HIGH` în timp ce `SCL` este `HIGH`.

Semnale importante utilizate:

- `scl_internal` și `sda_internal`: semnale interne pentru generarea liniei `SCL` și `SDA`.
- `clk_divider`: utilizat pentru a reduce frecvența sistemului la frecvența I2C dorită.
- `bit_index`: contor pentru urmărirea bit-urilor transmise/recepționate.

3.1.2 Slave I2C

Slave-ul I2C este responsabil pentru răspunsul la comenzi și transmiterea/recepția datelor de la Master. Funcționalitatea este implementată utilizând o FSM care parcurge următorii pași principali:

- **Detecția Start**: - Slave-ul detectează condiția `START` atunci când `SDA` trece de la `HIGH` la `LOW` în timp ce `SCL` este `HIGH`.
- **Recepția Adresei**: - Slave-ul primește adresa pe `SDA`, bit cu bit, sincronizată cu `SCL`. - Adresa este comparată cu adresa proprie pentru a determina dacă răspunde la comandă.
- **Transmiterea/Recepția Datelor**: - În cazul unei operații de scriere, datele de la Master sunt stocate în `data_out`. - În cazul unei operații de citire, datele din `data_in` sunt trimise pe `SDA` bit cu bit.

- **Finalizarea**: - FSM-ul revine în starea inițială după o condiție `STOP`, pregătind Slave-ul pentru o nouă transmisie.

Semnale importante utilizate:

- `sda_internal`: semnal intern pentru controlul `SDA`.
- `state`: reprezentarea stării curente a FSM-ului.
- `bit_index`: contor pentru gestionarea bit-urilor transmise/recepționate.

3.2. Implementare SPI

Protocolul SPI (Serial Peripheral Interface) a fost implementat folosind două componente: Master și Slave. Acestea colaborează pentru a realiza o comunicare serială sincronă bidirecțională între două dispozitive. Codul este structurat logic, utilizând semnale și procese pentru a respecta specificațiile protocolului SPI.

3.2.1 Master SPI

Implementarea componentei Master SPI constă în gestionarea comunicării și controlul liniei de ceas (SCLK) și a selecției slave-ului (SS). În cadrul acestei arhitecturi, sunt gestionate două stări principale: **init** și **execute**, utilizând o mașină de stări finite (FSM). Etapele principale sunt:

- **Inițializare**: La resetare, toate semnalele sunt setate la valori implicite. Linia MOSI este setată la impedanță înaltă (`Z`), iar buffer-ul de recepție (`rx`) este golit.
- **Transmiterea Datelor**: - La activarea semnalului `enable`, Master-ul selectează Slave-ul prin semnalul `SS` și configurează `SCLK` în funcție de polaritatea (CPOL) și faza clock-ului (CPHA). - Datele din buffer-ul `tx` sunt trimise bit cu bit prin linia `MOSI`, sincronizate cu `SCLK`. - Datele recepționate de la Slave sunt stocate în buffer-ul `rx`, utilizând linia `MISO`.
- **Finalizarea**: După ce toate datele sunt transmise și recepționate, Master-ul setează `SS` la `HIGH`, indicând finalizarea comunicării.

Semnale importante utilizate:

- `INT_ss` și `INT_sclk`: semnale interne pentru controlul `SS` și `SCLK`.
- `txBuffer` și `rxBuffer`: buffere utilizate pentru transmiterea și recepționarea datelor.
- `alternanta_clk`: contor pentru alternanțele de clock, utilizat în sincronizarea bit-urilor.

3.2.2 Slave SPI

Componentea Slave SPI implementează logica necesară pentru a răspunde comenzilor Master-ului și a comunica bidirecțional. Etapele principale ale implementării sunt:

- **Determinarea Modului SPI**: În funcție de `CPOL` și `CPHA`, Slave-ul configurează local ceasul `SCLK` pentru a respecta specificațiile protocolului.
- **Recepția Datelor**: - Slave-ul monitorizează linia `MOSI` și, sincronizat cu `SCLK`, preia bit-urile transmise de Master. - Datele primite sunt stocate în buffer-ul `rxBuffer`, iar atunci când `SS` este dezactivat (`HIGH`), acestea sunt transferate în `rx`.
- **Transmiterea Datelor**: - La activarea `SS`, datele din buffer-ul `tx` sunt transmise către Master, bit cu bit, prin `MISO`. - Sincronizarea este asigurată prin `SCLK`, iar fiecare bit transmis este deplasat în buffer-ul `txBuffer`.
- **Finalizarea**: La dezactivarea `SS`, buffer-ele sunt resetate, pregătind Slave-ul pentru o nouă transmisie.

Semnale importante utilizate:

- `rxBuffer` și `txBuffer`: gestionarea datelor recepționate și transmise.
- `clk`: ceas local generat în funcție de `CPOL` și `CPHA`.
- `bit_counter`: contor pentru a urmări progresul prin datele transmise/receptionate.

3.3. Implementare Ethernet

Protocolul Ethernet este implementat pentru a permite transmiterea și recepția de pachete de date între două dispozitive. Cele două componente principale sunt Transmiterea (EthernetTx) și Recepția (EthernetRx).

3.3.1 Transmiterea Ethernet (EthernetTx)

Componenta EthernetTx gestionează procesul de creare și transmitere a unui pachet Ethernet. Etapele principale :

- **Inițializare**: - La reset, toate semnalele sunt setate la valorile implicite. Semnalele `Tx` și `busy` sunt dezactivate.
- **Crearea Pachetului**: - Adresele MAC de destinație și sursă, tipul datelor (`type_data`), și payload-ul sunt concatenate pentru a forma structura completă a pachetului Ethernet.
- **Transmiterea Bit cu Bit**: - La activarea semnalului `start`, componenta transmite pachetul bit cu bit prin `Tx`, sincronizându-se cu `clk`. - Contorul `bit_counter` urmărește progresul transmisiei. La finalizarea transmisiei, `Tx` este dezactivat, iar `busy` revine la `LOW`.

Semnale importante utilizate:

- `tx_data`: buffer intern care stochează pachetul complet.
- `bit_counter`: contor pentru urmărirea progresului transmisiei.
- `Tx`: semnal de ieșire pentru transmiterea datelor.

3.3.2 Recepția Ethernet (EthernetRx)

Componenta EthernetRx gestionează procesul de recepție și interpretare a unui pachet Ethernet. Etapele principale :

- **Recepția Bit cu Bit**: - Componenta recepționează pachetul bit cu bit pe linia `Rx`, stocând datele în buffer-ul `rx_buffer`.
- **Descompunerea Pachetului**: - După recepția completă, pachetul este descompus în adresa MAC de destinație, adresa MAC sursă, tipul datelor, și payload. - Datele sunt mapate pe ieșirile `dest_mac`, `src_mac`, `type_data`, și `payload`.
- **Validarea Pachetului**: - Semnalul `valid` este activat pentru a indica faptul că pachetul recepționat este complet și valid.

Semnale importante utilizate:

- `rx_buffer`: buffer intern care stochează pachetul recepționat.
- `bit_counter`: contor pentru urmărirea progresului recepției.
- `valid`: semnal de ieșire pentru validarea pachetului recepționat.

4. Rezultate TestBeanch

4.1. Rezultate I2C

4.1.1 Etapele Transmiterii Datelor

Protocolul I2C urmează următoarele etape principale pentru transmiterea și recepționarea datelor:

1. **Inițializare**: Master-ul configurează semnalele *SCL* (Clock) și *SDA* (Data), pregătind linia pentru comunicare. Slave-ul intră în stare de așteptare până când detectează o condiție *START* generată de Master.

2. **Start Transmisie**: Master-ul setează *SDA* la nivel *LOW* în timp ce *SCL* este *HIGH*, indicând începutul comunicării. Adresa Slave-ului este transmisă bit cu bit prin linia *SDA*, sincronizată cu *SCL*.

3. **Transmiterea/Recepția Datelor**: - În cazul unei operații de *scriere*, Master-ul trimite datele bit cu bit prin linia *SDA*, iar Slave-ul le recepționează în buffer-ul *data_out*. - În cazul unei operații de *citire*, Slave-ul trimite datele către Master, bit cu bit, pe linia *SDA*, iar Master-ul le stochează în *data_out*.

4. **Finalizarea Transmisiei**: La încheierea comunicării, Master-ul generează o condiție *STOP* prin setarea *SDA* la *HIGH* în timp ce *SCL* este *HIGH*. Datele recepționate sunt disponibile în buffer-ele *data_out* atât pentru Master, cât și pentru Slave.

4.1.2 Rezultate Obținute

În urma simulării testbench-ului, s-au obținut următoarele rezultate:

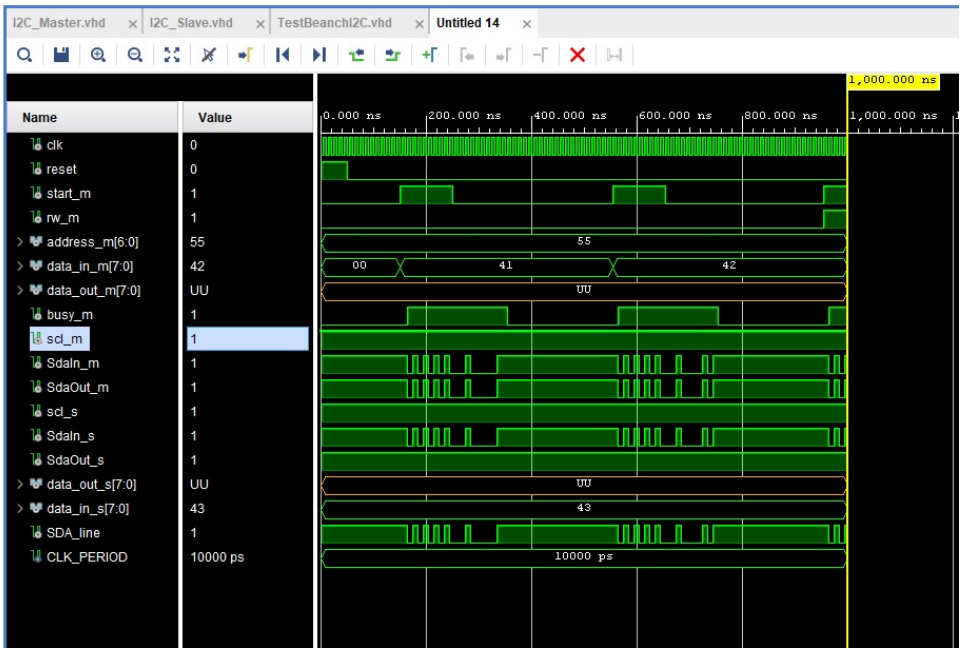
- **Date transmise de Master**: Caracter `'A'` (`01000001`) și `'B'` (`01000010`).
- **Date recepționate de Slave**: Caracter `'A'` (`01000001`) și `'B'` (`01000010`).
- **Date transmise de Slave**: Caracter `'C'` (`01000011`).
- **Date recepționate de Master**: Caracter `'C'` (`01000011`).

Aceste rezultate demonstrează că implementarea protocolului I2C este corectă, iar datele sunt transmise și recepționate fără erori.

4.1.3 Diagrama Temporală a Simulării

În figura de mai jos este prezentată diagrama temporală a semnalelor principale din timpul simulării protocolului I2C. Se pot observa :

- Generarea condițiilor *START* și *STOP* de către Master.
- Transmiterea bit cu bit a adreselor și datelor pe linia *SDA*, sincronizată cu *SCL*.
- Confirmarea recepției datelor prin semnalul *ACK* implicit al Slave-ului.



4.2. Rezultate SPI

Date experimentale pentru implementarea SPI au fost obținute prin rularea testbench-ului dezvoltat. Rezultatele demonstrează transmiterea corectă a datelor între componentele Master și Slave.

4.2.1 Etapele Transmiterii Datelor

Protocolul SPI urmează următoarele etape principale pentru transmiterea și recepționarea datelor:

1. **Inițializare**: Master-ul configurează semnalele *SCLK* (clock-ul SPI), *SS* (Slave Select), și pregătește buffer-ul de date *TX* pentru transmitere. Slave-ul intră în stare de așteptare până când *SS* devine activ (nivel logic *LOW*).

2. **Start Transmisie**: Master-ul setează *SS* la *LOW*, inițiind comunicarea cu Slave-ul. În funcție de configurările *CPOL* (Clock Polarity) și *CPHA* (Clock Phase), datele sunt transmise bit cu bit pe linia *MOSI* (Master Out Slave In) și sunt recepționate simultan de Master prin *MISO* (Master In Slave Out).

3. **Transmiterea Datelor**: - La fiecare alternanță a clock-ului SPI, Master-ul trimite un bit către Slave prin *MOSI* și primește simultan un bit de la Slave prin *MISO*. - Slave-ul își actualizează buffer-ul *RX* cu datele primite și transmite datele din buffer-ul *TX* către Master.

4. **Finalizarea Transmisiei**: După ce toate bit-urile au fost transmise, Master-ul setează *SS* la *HIGH*, încheind comunicarea cu Slave-ul. Datele recepționate sunt stocate în buffer-ele *RX* atât pentru Master, cât și pentru Slave.

4.2.2 Rezultate Obținute

În urma simulării testbench-ului, s-au obținut următoarele rezultate:

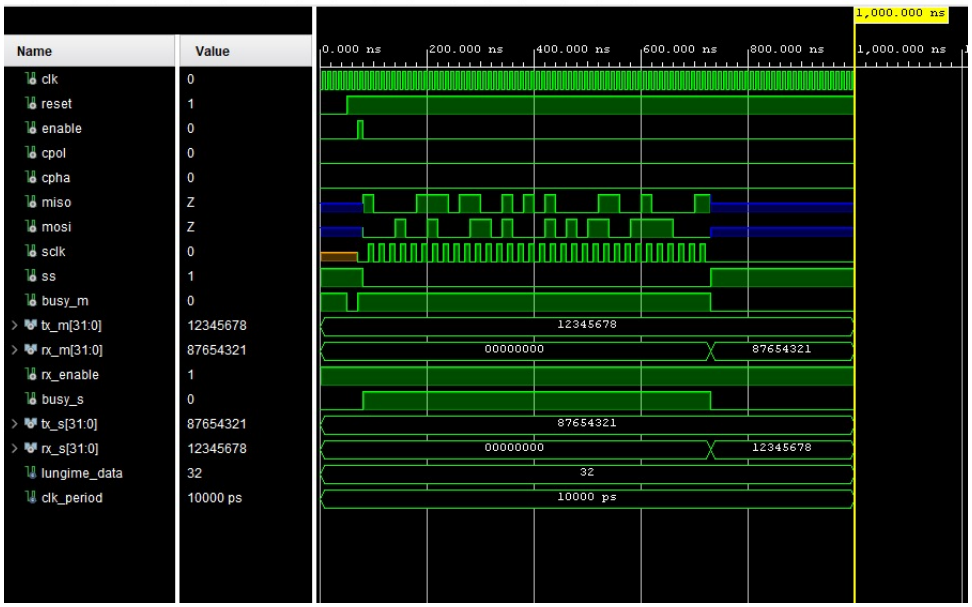
- **Date transmise de Master**: `0x12345678`
- **Date recepționate de Slave**: `0x12345678`
- **Date transmise de Slave**: `0x87654321`
- **Date recepționate de Master**: `0x87654321`

Aceste rezultate confirmă că implementarea protocolului SPI este corectă, iar datele sunt transmise și recepționate în mod bidirecțional, fără erori.

4.2.3 Diagrama Temporală a Simulării

În figura de mai jos este prezentată diagrama temporală a semnalelor principale din timpul simulării protocolului SPI. Se pot observa :

- Alternanța semnalului *SCLK* în funcție de configurațiile *CPOL* și *CPHA*.
- Schimbul de date bit cu bit între *MOSI* și *MISO*.
- Activarea și dezactivarea semnalului *SS* pentru controlul Slave-ului.



4.3. Rezultate Ethernet

Date experimentale pentru implementarea protocolului Ethernet au fost obținute prin rularea testbench-ului dezvoltat. Rezultatele confirmă transmiterea și recepționarea completă și corectă a pachetelor Ethernet între componentele *EthernetTx* și *EthernetRx*.

4.3.1 Etapele Transmiterii Datelor

Protocolul Ethernet urmează următoarele etape principale pentru transmiterea și recepționarea datelor:

1. **Inițializare**: - La resetare, semnalele sunt inițializate, iar *Tx* este setat în stare inactivă.
2. **Configurarea Pachetului**: - Pachetul Ethernet este format prin concatenarea adreselor MAC (destinație și sursă), tipului de date *type_data*, și payload-ului.
3. **Transmiterea Datelor**: - La activarea semnalului *start*, *EthernetTx* începe transmiterea pachetului bit cu bit. - Contorul *bit_counter* urmărește progresul transmisiei până la finalizarea pachetului.
4. **Recepția Datelor**: - Componenta *EthernetRx* recepționează datele bit cu bit și le stochează în buffer-ul intern. - După recepția completă, pachetul este descompus în adresa MAC de destinație, adresa MAC sursă, tipul de date, și payload.
5. **Validarea Pachetului**: - Semnalul *valid* este activat pentru a indica o recepție completă și corectă a datelor.

4.3.2 Rezultate Obținute

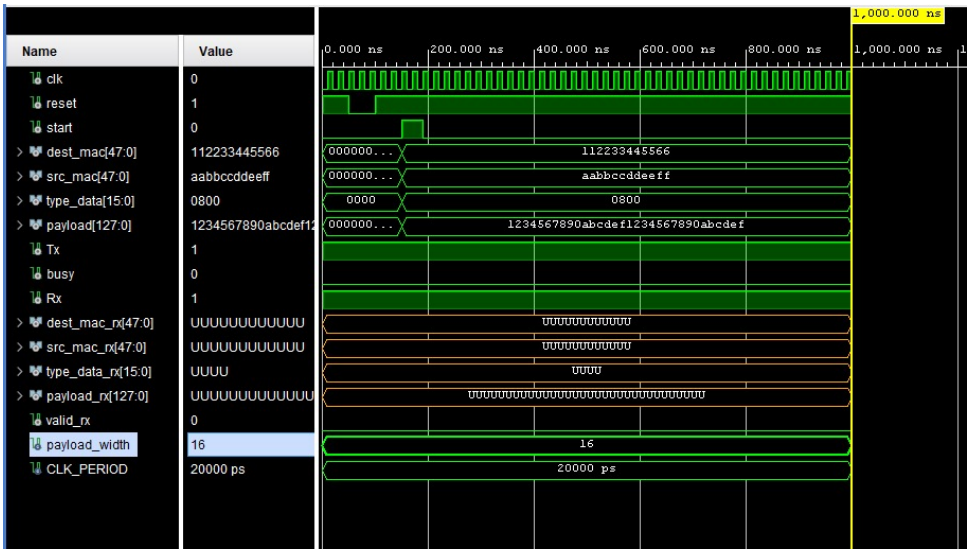
În urma simulării testbench-ului, s-au obținut următoarele rezultate:

- **Adresă MAC destinație recepționată**: `112233445566`.
- **Adresă MAC sursă recepționată**: `AABBCCDDEEFF`.
- **Tip de date**: `0800`.
- **Payload recepționat**: `1234567890ABCDEF1234567890ABCDEF`.
- **Semnal de validare**: `1`.

4.3.3 Diagrama Temporală a Simulării

În figura de mai jos este prezentată diagrama temporală a semnalelor principale din timpul simulării protocolului Ethernet. Se pot observa următoarele:

- Transmiterea bit cu bit a pachetului Ethernet prin *Tx*.
- Recepția completă a datelor în buffer-ul *EthernetRx*.
- Activarea semnalului *valid* la finalizarea recepției.



Bibliografie

[1] Hennessy, J. L., Patterson, D. A., *Organizarea și proiectarea calculatoarelor*, Editura All, 2002.

[2] Chapman, K., *PicoBlaze 8-Bit Microcontroller*, Xilinx Application Note XAPP213.

[3] Satish M. Ghuse, Surendra K. Waghmare, *FPGA Implementation of I2C and SPI Protocols using VHDL*, IJSART, Volume 2 Issue 10, October 2016. Disponibil la: `https://surf-vhdl.com/spi-slave-vhdl-design/` și `https://forum.digikey.com/t/spi-master-vhdl/12717`.

[4] Trupti D. Shingare, R. T. Patil, *SPI Implementation on FPGA*, IJITEE, Volume 2 Issue 2, January 2013. Disponibil la: `https://www.ijitee.org/`.

[5] Richa Malhotra, Swati Juneja, *Design and Implementation of Multiplexed SPI using FPGA*, IJLTEMAS, Volume VI Issue VII, July 2017. Disponibil la: `https://satoms.com/ethernet-cable-pinouts/`.

[6] Document PDF: *Design and Implementation of I2C and SPI Protocols*, pagini 16-19, disponibil pentru detalii.

[7] Document PDF: *SPI Implementation on FPGA*, pagini 7-9, disponibil pentru detalii.

[8] Document PDF: *Design and Implementation of Multiplexed SPI using FPGA*, pagini 76-79, disponibil pentru detalii.

[9] *Ethernet Cable Pinouts*, Satoms. Disponibil la: `https://cot-cn.cougarnet.uh.edu/docs/compnet/012-ethernet.html`.