Laborator 12

Automate cu stări finite / Comunicație serială

1. Obiective

Studiul, proiectarea, implementarea și testarea:

- **Automate (mașini) cu stări finite**
- **Comunicație serială**

2. Fundamente teoretice

2.1. Mecanismul de (supra)eșantionare pentru recepția serială UART

La transmiterea unui octet, UART transmite mai întâi bitul de START, urmat de biți de date (în mod uzual 8 biți, dar e posibil și cu 5, 6 sau 7 biți), urmați de bitul de STOP. Protocolul e repetat pentru fiecare octet din secvența care trebuie trimisă.

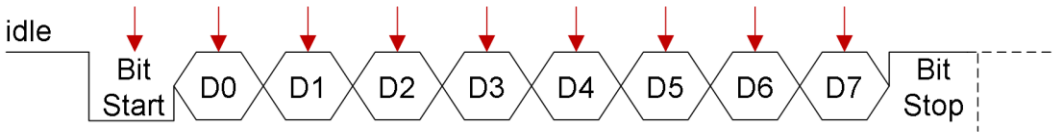

Figura 1: Diagrama de timp pentru transmisia serială, exemplu pe 8 biți, săgețile roșii indicând momentul ideal când linia ar trebui citită la receptor.

Transmisia serială nu implică existența unui semnal de ceas. În schimb se folosește o rată de eșantionare, numită *baud rate* (**număr de biți transmiși pe secundă**). Valorile uzuale pentru baud rate sunt 2400, 4800, 9600 și 19200. Practic, un bit este valid pe linia de transmisie pentru un interval dat de timp, egal cu inversul baud rate. Mai multe detalii legate de transmisie se regăsesc în laboratorul anterior.

În cazul recepției, trebuie citit (eșantionat) semnalul de pe linia serială, și extras, bit cu bit, semnalul transmis de la sursă.

La prima vedere, rata de eșantionare la recepție ar trebui sa coincidă cu rata de eșantionare (baud rate) cu care s-a transmis șirul de biți. Greșit! Dacă s-ar citi la rata de transmisie, din cauza nesincronizării perfecte (baud rate este generat independent la sursa, respectiv destinație, este comunicație asincronă = fără un semnal de ceas comun), există riscul unor decalaje care cauzează citirea dublă a aceluiași bit, sărirea peste un bit, ratarea bitului de start, etc. Frecvența de apariție pentru aceste erori este proporțională cu diferența dintre ratele de eșantionare la transmițător și receptor. Chiar dacă ar fi diferențe foarte mici, la un număr suficient de eșantionări consecutive (pentru biți consecutivi) eroare va apărea. De exemplu, la o diferență de 0.1% între cele două rate, la 1000 de biți consecutivi eroarea apare o dată. În acest caz, dacă avem 10 biți / caracter, rezultă că un caracter din 100 va fi recepționat greșit.

Problema se rezolvă prin supra-eșantionare, adică semnalul de pe linia serială este citit la o rată mai mare decât rata de transmisie. Acest lucru permite detecția mijlocului bitului de start, biții de date fiind apoi citiți aproximativ în jurul mijlocului de interval de bit, eliminând practic riscul de decalaj (la fiecare nou caracter se reia din mijlocul bitului de start = resincronizare). Cea mai uzuală schema de supra-eșantionare este folosirea unei rate la recepție de 16 ori mai mare decât baud rate folosit la transmisie. Concret, fiecare bit care vine pe serial este eșantionat (citit) de 16 ori, însă doar una dintre citiri este salvată (cea din mijloc). Întârzierea maximă de detecție a bitului se start este cel mult 1/16 din intervalul de bit.

Orice circuit UART conține un registru de deplasare (shift register), acesta fiind folosit în mod clasic pentru conversia datelor din forma paralelă în forma serială, și invers.

3. Activități practice

3.1. FSM pentru recepția serială

La fel ca la transmisie, mai întâi trebuie să descrieți în test_env un generator pentru semnalul de eșantionare (baud rate), pentru o valoare de 9600 (biți pe secundă). Diferența față de transmisie este ca acest semnal trebuie să aibă o rată de 16 ori mai mare. Folosiți un numărător pentru a genera semnalul BAUD_ENable (’0’ valoare normală, este pus pe ‘1’ la intervalul de bit, interval care se măsoară în tacți de ceas, după care se reîncepe numărarea). Durata cât stă BAUD_ENable ‘1’ pe este egală cu perioada de ceas.

Pentru generarea baud rate (Basys 3 – 100 Mhz):

- La ceas de 25 MHz, se generează ‘1’ la fiecare 25Mhz/(9600*16)=163 tacți.
- La ceas de 50 MHz, se generează ‘1’, la fiecare 326 tacți.
- La ceas de 100 MHz, se generează ‘1’, la fiecare 651 tacți.

Definiți o nouă entitate pentru FSM-ul de recepție, cu porturile conform figurii următoare.

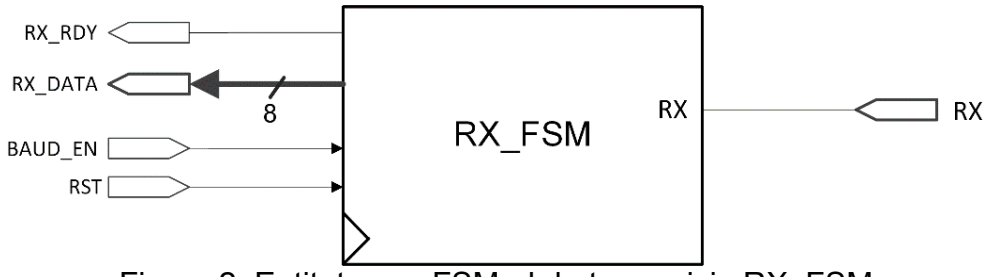

Figura 2: Entitatea cu FSM-ul de transmisie RX_FSM

Diagrama în detaliu a RX_FSM este prezentată în figura de mai jos. O tranziție între stări poate avea loc pe frontul crescător de ceas doar dacă BAUD_ENable este ‘1’. Astfel se asigură că eșantionarea liniei seriale (prin acțiunile de verificare a lui RX din stările idle, start și bit) se face la o rată de 16 ori mai mare ca rata de transmisie.

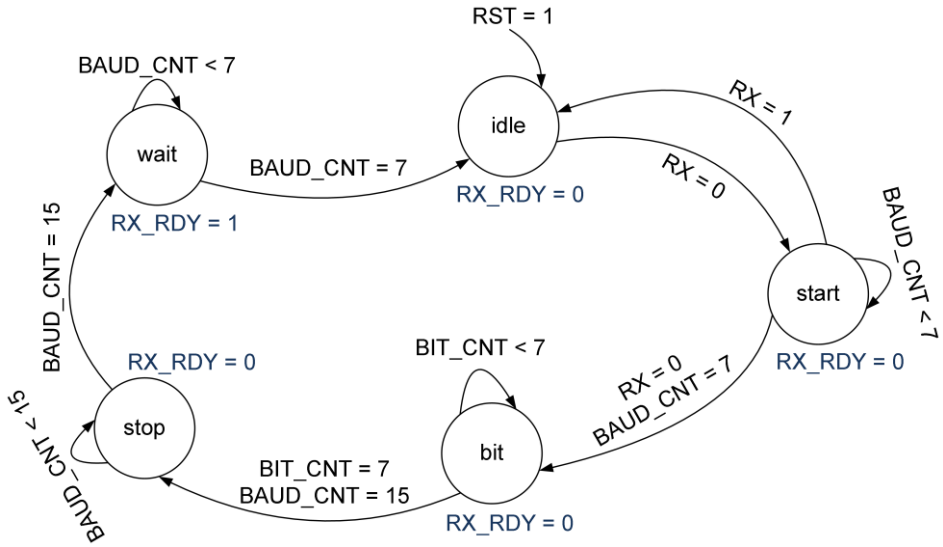

Figura 3: Diagrama TX_FSM

Cei 8 biți citiți se vor stoca în RX_DATA (în RX_FSM, pe starea *bit*). RX_RDY este setat pe 1 după ce s-a terminat recepția unui caracter (acest semnal este util doar la activitatea 3.2).

În RX_FSM sunt necesare două semnale auxiliare cu funcționalitate de numărător: BIT_CNT și BAUD_CNT.

Semnalul BIT_CNT are o funcționalitate la fel ca în TX_FSM, de numărător în interiorul FSM-ului, el reprezentând poziția din RX_DATA unde se stochează bitul curent. El trebuie incrementat în starea *bit*, de fiecare dată când se ajunge la mijlocul bitului curent, și trebuie resetat după fiecare caracter transmis (se poate reseta în starea *idle* sau în toate stările exceptând *bit*).

Semnalul BAUD_CNT contorizează câte activări (valori de 1) are semnalul BAUD_ENable, el fiind utilizat pentru a verifica dacă s-a ajuns la mijlocul intervalului de bit, sau la sfârșit de interval.

În mare, funcționarea RX_FSM este următoarea:

1. În starea *idle*, BAUD_CNT este setat pe 0, se trece în *start* dacă pe linia serială apare 0 (RX=0).
2. În starea *start*, BAUD_CNT este incrementat și folosit pentru a eșantiona linia serială (RX). Dacă s-a ajuns la mijlocul bitului de start (BAUD_CNT=7 și RX=0) atunci BAUD_CNT este resetat și se trece la starea *bit*.
3. În starea *bit*, BAUD_CNT este implicit incrementat, până ajunge la valoarea 15 (mijlocul bitului curent). Dacă a ajuns la 15, se salvează bitul curent (RX_DATA[BIT_CNT]=RX), se resetează BAUD_CNT (se reîncepe numărarea până la 15) și se incrementează BIT_CNT. Dacă BIT_CNT=7 și BAUD_CNT=15 atunci sunt resetate și trece în starea *stop.*

4. În *stop* se așteaptă să treacă încă un interval de bit pe linia serială, interval care acoperă practic jumătate din ultimul bit de date și jumătate din bitul de stop (amintiți-vă că s-a început de la jumătatea bitului de start, deci în *stop* s-a sărit din *bit* la jumătatea ultimului bit de date).
5. În *wait* se semnalează că s-a recepționat un caracter (RX_RDY=1), se mai așteaptă o jumătate de interval de bit (să se termine bitul de stop) și se trece în *idle*.

Descrieți în VHDL comportamentul entității RX_FSM, din test_env. Folosiți o descriere FSM cu 2 sau cu 3 procese (vezi anexa 7 din laboratorul 11).

Conectați ieșirea RX_DATA a lui RX_FSM la afișorul SSD. Se va afișa pe SSD (2 cifre) codul ASCII al caracterelor transmise de pe calculator către placa FPGA.

Acum se va testa comunicația între placa de dezvoltare și calculator. Parametrii pentru comunicația serială sunt: 1 bit de START, 1 bit de STOP, 8 biți de date, fără bit de paritate, baud rate de 9600. Asigurați-vă că aceste setări apar în aplicația de pe calculator HTERM / hyper terminal, și alegeți corect portul serial în aplicație (COM1, 2, 3 etc). Pentru a identifica exact care port serial este cel cu placa, vizualizați în aplicație lista de porturi seriale înainte și după legarea modulului prin cablul USB la calculator.

3.2. Comunicare I/O cu procesorul MIPS - OPȚIONAL

Conectați RX_FSM cu procesorul MIPS pe care l-ați implementat (puteți folosi MIPS cu ciclu unic sau versiunea pipeline, o versiune funcțională). Trebuie să trimiteți 16 biți de date de pe calculator (HTERM) către procesorul vostru. Acești biți ar trebui să reprezinte unul dintre operanzii de intrare ai programului vostru. În funcție de unde este localizat acest operand (în blocul de regiștrii, în memoria de date), veți defini ce câmp din procesor se va scrie cu datele recepționate pe linia serială.

Țineți cont că pe linia serială se recepționează câte 8 biți, care reprezintă codul ASCII al unei cifre. Pentru a transmite 16 biți către procesor de pe calculator, trebuie trimise 4 cifre hexazecimale, fiecare fiind codificată pe linia serială prin codul ei ASCII. Folosiți un decodificator/ROM pentru a genera cifra de 4 biți asociată cu codul ASCII pe 8 biți primit pe linia serială. Concatenați cele 4 cifre primite prin 4 transferuri pentru a obține data pe 16 biți pe care o veți scrie în câmpul dorit din procesor.

Bibliografie

- XST User Guide
- Digilent Basys Board – Reference Manual
- Digilent Pmod USB-UART – Reference Manual
- http://www.asciitable.com/
- http://www.der-hammer.info/terminal/