Transmiterea unor comenzi de la un dispozitiv mobil la o placă de dezvoltare

Pereanu Alexandru

Cuprins

1.Rezumat

Scopul proiectului este de a realiza o legătură între placa FPGA Nexys 4 DDR și o aplicație Java/Android. Această legătură se va face cu un modul Bluetooth PMOD BT2. Prin aplicație se vor da comenzi plăcii reprezentând operații fundamentale matematice care sunt implementate în VHDL prin intermediul unei unități aritmetice și logice.

Obiectivele principale ale proiectului sunt de a vedea cum se comportă placa FPGA Nexys 4 DDR dacă are legată la ea un modul Bluetooth PMOD BT2 prin care se va comunica cu o aplicație Java și de a învăța mai multe despre FPGA-uri si structura sistemelor de calcul. Noi prin acest proiect vom efectua câteva operații aritmetice cu ajutorul unei ALU care va fi implementată în VHDL, se vor transmite date plăcii printr-un modul Bluetooth și printr-o aplicație Java. Ca și utilitare se va lucra în Vivado și în Intellij IDEA.

Pe placă se vor afișa rezultatele următoarelor operații: adunare, scădere, înmulțire, împărțire, deplasare dreapta-stânga și într-un final se vor aprinde și câteva leduri.

2. Introducere

2.1 Contextul temei proiectului

Tema proiectului este una de interes deoarece tehnologia ne pune la dispoziție diferite metode prin care să efectuăm operații aritmetice mult mai ușor și accesibil. Plăcile FPGA au ajuns să fie destul de cunoscute și folosite de foarte mulți dezvoltatori de aplicații iar limbajul Java este un limbaj de programare foarte utilizat în domeniul IT. Combinând cele două putem obține un sistem care să ne rezolve operațiile aritmetice fundamentale: adunarea, scăderea, înmulțirea, împărțirea și eventual deplasările stânga-dreapta.

2.2 Generalități despre FPGA

Circuitul FPGA (Field Programmable Gate Array) este un circuit generic ce poate fi programat pentru a implementa orice funcție definită de utilizator. Un FPGA este constituit dintr-o matrice de blocuri programabile, o rețea de interconectare care leagă aceste blocuri între ele, circuite de I/O și o memorie SRAM care configurează aceste structuri. Figura alăturată prezinta o schema simplificata de FPGA.

Blocurile programabile pot fi generatoare de funcții logice, memorii RAM, sau circuite de înmulțire. În figură este prezentată structura unui bloc generator de funcții logice. Acesta este format dintr-un LUT (Look-Up Table) și un registru pentru a sincroniza ieșirea acestuia, dacă se dorește acest lucru. Un LUT poate implementa orice funcție logică de 4 variabile cu ajutorul unui selector. Fiecare combinație de valori ale variabilelor de intrare selectează rezultatul corespunzător, calculat în prealabil de programul de sinteză și stocat în memoria de configurație la programarea FPGA-ului. Ieșirea generatorului de funcții logice poate fi legată fie la registru, fie direct la ieșirea LUT-ului.

Fiecare bloc programabil este legat la rețeaua de interconectare. Aceasta este formată din segmente de fire și comutatoare programabile. Fiecare comutator poate lega două sau mai multe segmente între ele, asa cum se poate observa în figură. Starea comutatoarelor (on/off) este deasemenea determinată în prealabil în etapa de Place and Route, și stocată în memoria de configurație.

2.3 Soluția propusă

Soluția propusă pentru rezolvarea proiectului este de a implementa o unitate aritmetică logică care să efectueze următoarele operații: adunare, scădere, înmulțire, împărțire și deplasare stânga-dreapta. Operația de adunare se face face printr-un sumator cu anticiparea transportului, cea de scădere realizându-se la fel, adică ne folosim de sumatorul de la adunare dar pe locul celui de-al doilea termen vom pune complementul față de doi al celui de-al doilea număr din operație. Înmulțirea se va realiza printr-un circuit de înmulțire prin metoda Booth, iar împărțirea printr-un circuit de împărțire cu refacerea restului parțial. Pentru deplasări stânga-dreapta ne vom folosi de circuitele de deplasare învățate în semestrele trecute la materiile asemănătoare. Aplicația Java se va face în IDEA-ul Intellij iar legătura dintre placă și aplicația Java va fi făcută printr-un modul Bluetooth PMOD BT2. Mai multe detalii despre toate acestea în sectiunile din capitolele următoare.

2.4 Descrierea proiectului

Realizarea proiectului constă în proiectarea și implementarea unui sistem format dintr-o aplicație Java care va transmite comenzi la o placă de dezvoltare. Această legătură dintre aplicația Java și placa FPGA va fi făcută prin utilizarea unui modul Bluetooth Pmod BT2.

Implementarea acestui proiect se realizează cu ajutorul unei plăci de dezvoltarea FPGA(Nexys 4/Basys 3) utilizând limbajul de descriere hardware VHDL, folosind programul software Vivado.

Pentru a crește complexitatea sistemului am ales de a implementa o Unitate Aritmetică Logică (ALU) ce va primi intrări date de utilizator în aplicația Java.

În continuarea acestei documentații se vor prezenta câteva noțiuni teoretice despre plăcile FPGA pe care le vom folosi, utilitarele în care se va lucra, metodele folosite pentru realizarea proiectului, câteva rezultate experimentale și partea de bibliografie.

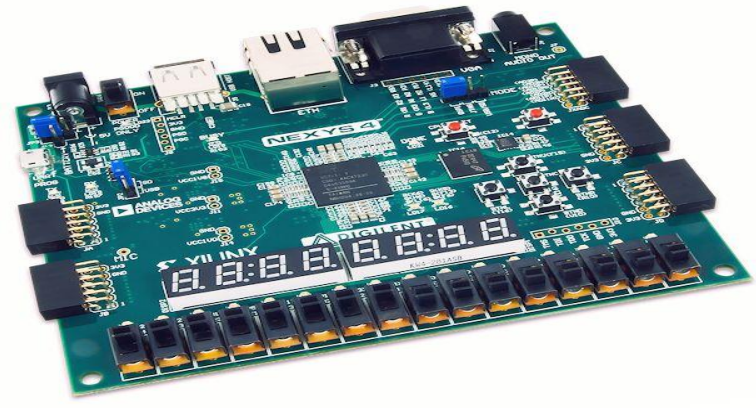
3. Fundamentare teoretică

3.1 Nexys 4

Nexys 4 este o plăcuță folosită pentru dezvoltarea circuitelor digitale.Aceasta poate găzdui proiecte variind de la circuite introductive combiationale la procesoare puternice integrate. Această incorporează două memorii externe: una de 1Gb(128Mb) DDR2 ȘRAM și alta de 128Mb(16Mb) nevolatilă și serială Flash. Modulele DDR2 sunt integrate pe placă și se conectează la FPGA utilizând interfața standard primită la momentul fabricației.

Nexys 4 este folosită în special în industrie și în mediul academic.

Se folosește de procesorul Artix 7 FPGA de la Xilinx și este optimizată pentru logica de inaltă performantă. Oferă capacitate, performanțe mult mai mari și mai multe resurse decât modelele anterior lansate pe piață.



[Fig. 1]

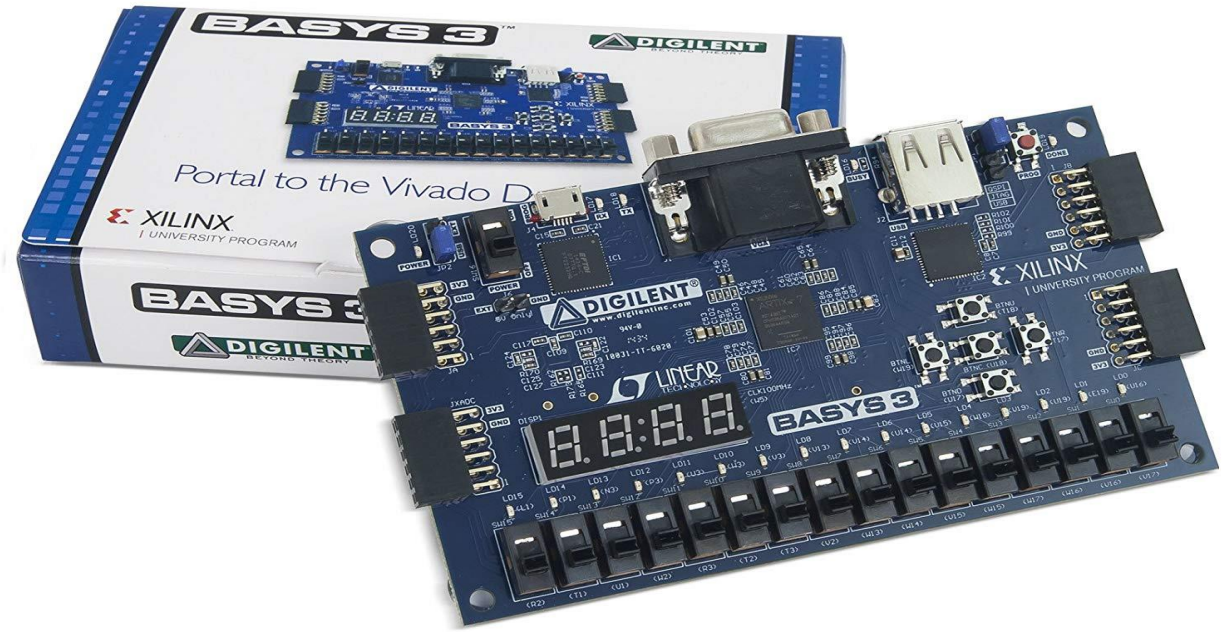
Plăca Nexys 4 DDR

În componența sa există mai multe tipuri de periferice incorporate. Printre acestea se numără: accelerometrul, senzorul de temperatură, Fig.1 Nexys 4 DDR
microfonul digital, amplificatorul de difuzor, etc.

Nexys 4 poate fi folosit în implementarea proiectelor împreună cu programe software, cum ar fi Vivado Design Suite sau ISE.

3.2 Basys 3

Basys 3 este o placă de dezvoltare FPGA entry-level concepută exclusiv pentru Vivado Design Suite, cu arhitectura Xilinx Artix-7 FPGA. Basys 3 este cea mai nouă adăugare la linia populară Basys a panourilor de dezvoltare FPGA și este perfect potrivită pentru studenți sau începători care încep doar cu tehnologia FPGA. Bazele 3 includ caracteristicile standard găsite pe toate plăcile Basys: hardware complet gata de utilizare, o colecție mare de dispozitive de I / O la bord, toate circuitele de suport FPGA necesare, o versiune gratuită a instrumentelor de dezvoltare și la un student.



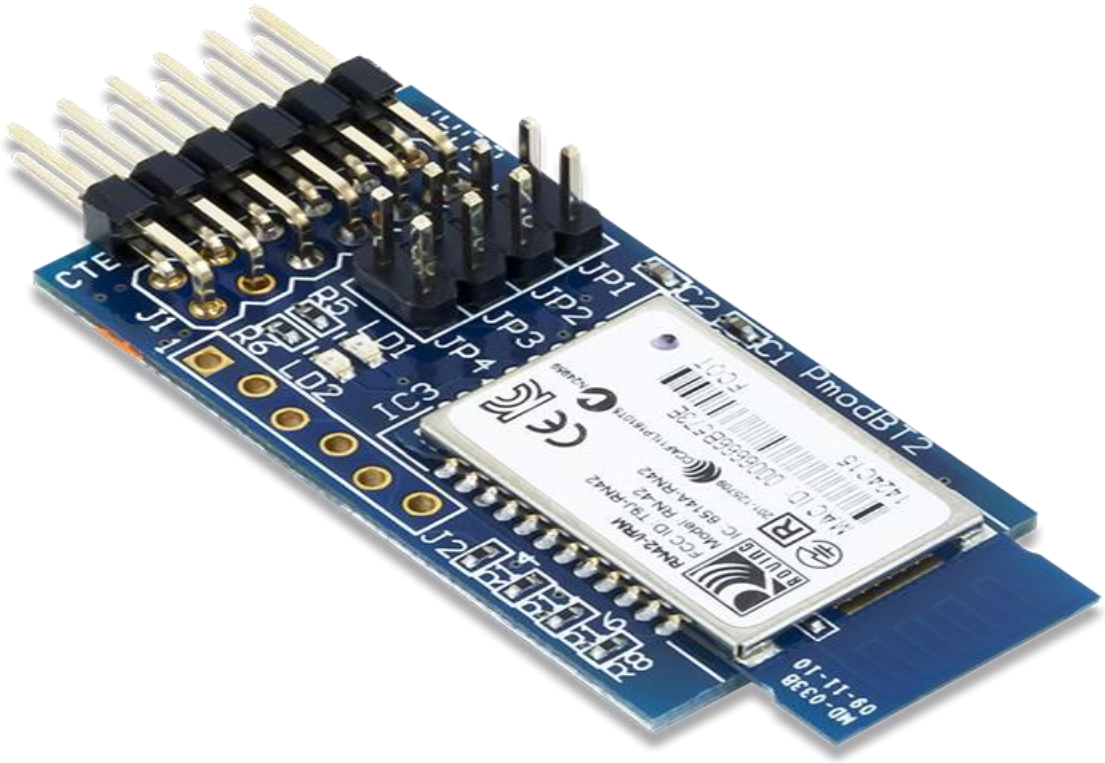
[Fig. 2]

Placă Basys 3

3.3 Modulul Bluetooth Pmod BT2

Pmod BT2 este un puternic modul periferic care utilizează Roving Networks RN-42 pentru a crea o interfață Bluetooth complet integrată. Utilizatorii pot comunica cu cipul prin UART și pot utiliza, de asemenea, antetul SPI secundar de pe placă pentru actualizarea firmware-ului RN-42.

Modulele Pmod comunică cu plăcile de sistem cu conectori cu 6, 8 sau 12 pini care pot transporta mai multe semnale de control digitale, cum ar fi SPI și alte protocoale seriale. Modulele Pmod permit proiectări mai eficiente prin dirijarea semnalelor analogice și a surselor de alimentare numai acolo unde sunt necesare și departe de plăcile de control digitale.



[Fig. 3]

Modul PMOD BT2

3.3.1 Descriere funcțională

Pmod BT2 utilizează un port cu 12 pini și comunică prin UART. Există un antet SPI secundar pe placă pentru actualizarea firmware-ului RN-42, dacă este necesar, însă acest port nu este utilizat în funcționare normală.

Aplicația tipică pentru Pmod BT2 este înlocuirea unei conexiuni UART cu fir între două dispozitive Bluetooth. Când este asociat cu un computer Android, Linux, Mac OS X sau Windows, Pmod BT2 arată ca un port COM serial, similar cu modul în care se comportă un port USB-UART sau un port serial RS-232. Pmod BT2 poate fi configurat cu ușurință de pe computerul conectat prin Bluetooth, prin introducerea unui „Mod de comandă” care permite programarea setărilor precum rata de transfer UART în registrele de configurare non-volatile.

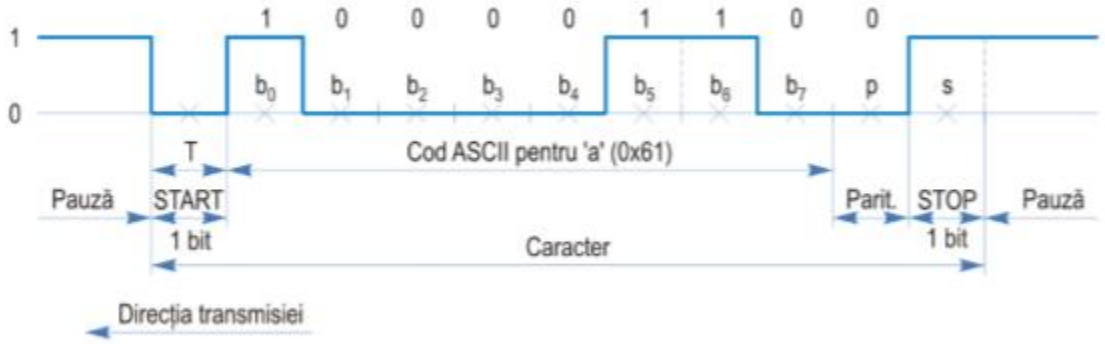
Rețineți că, din cauza restricțiilor de securitate încorporate în iOS, Pmod BT2 nu poate fi utilizat cu iPhone, iPad sau alte dispozitive iOS.

3.3.2 Interfațarea cu Pmod BT2

Comunicația serială asincronă

În cazul comunicației seriale asincrone, fiecare caracter transmis este precedat de un bit de START, cu valoarea logică 0, și este urmat de cel puțin un bit de STOP, cu valoarea logică 1. Deci, biții de START și de STOP încadrează fiecare caracter transmis. Intervalul de timp între transmisia a două caractere succesive este variabil, pe durata acestui interval linia de comunicație fiind în starea 1 logic.

Atunci când receptorul detectează bitul de START care indică începutul unui caracter, pornește un oscilator de ceas local, care permite măsurarea intervalului de timp corespunzător unui bit, interval care depinde de debitul binar. Acest oscilator permite eșantionarea corectă a biților individuali ai caracterului. Eșantionarea biților se realizează aproximativ la mijlocul intervalului corespunzător fiecărui bit.

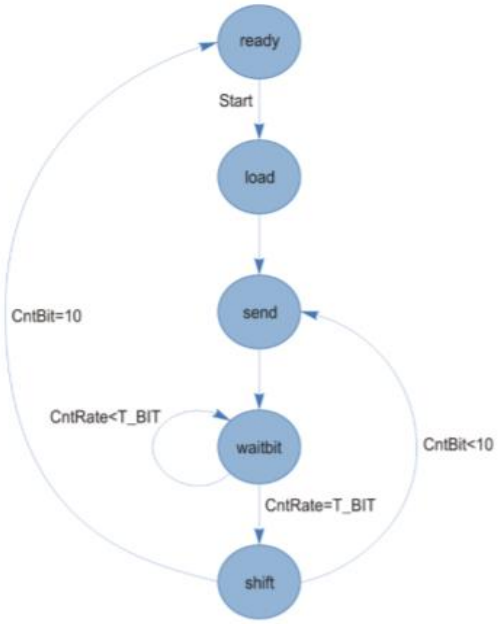


[Fig. 4]
Transmisia serială asincronă a caracterului cu codul ASCII 0x61

Interfață UART / Pmod

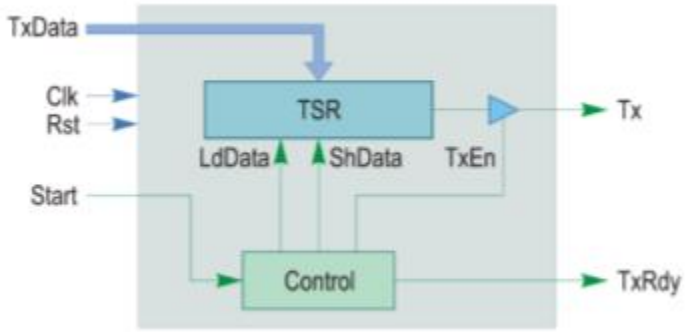
Pmod BT2 comunică cu placa gazdă prin protocolul UART. În mod implicit, interfața UART folosește o rată de transfer de 115,2 kbps, 8 biți de date, fără paritate și un singur bit de oprire. Rata de transfer de pornire poate fi personalizată la rate predefinite sau setată la un anumit debit de transfer personalizat, de la 1200 bps la 921 kbps.

Pinul de resetare (RST) de pe J1 este activ scăzut. Dacă pinul RST este comutat, dispozitivul va fi resetat. A doua interfață în afară de semnalul UART standard este pinul STATUS de asemenea pe J1. Pinul STATUS reflectă direct starea de conectare a dispozitivului. STATUS este condus înalt de dispozitiv atunci când este conectat și este condus în alt mod scăzut.



[Fig. 5]
Diagrama de stare a unității de control pentru transmițătorul serial UART

Modulul transmițătorului serial care se va realiza va permite specificarea debitului binar (a vitezei de comunicație) printr-un generic. Caracterele transmise vor fi de 8 biți, fără bit de paritate, și vor fi urmate de un bit de STOP. Acest modul se poate implementa cu ajutorul unui registru de deplasare cu încărcare paralelă și ieșire serială, și a unei unități de control care generează semnalul de stare TxRdy și semnalele de comandă necesare registrului de deplasare. Unitatea de control va include și numărătoare pentru contorizarea biților transmiși și pentru măsurarea duratei unui bit, în funcție de debitul binar specificat. Schema bloc a modulului transmițător este prezentată în figura 4. Intrările modulului sunt semnalul de ceas Clk, semnalul de resetare sincronă Rst, vectorul de 8 biți TxData, reprezentând octetul care trebuie transmis, și semnalul Start, reprezentând comanda de începere a transmisiei octetului de la intrarea TxData. Ieșirile modulului sunt linia Tx pe care sunt transmise datele în mod serial și semnalul TxRdy, care indică prin starea sa activă faptul că transmisia unui octet a fost terminată.



[Fig. 6]
Schema bloc a transmițătorului serial UART

Registrul de deplasare TSR este încărcat cu octetul care trebuie transmis, completat cu bitul de START ('0') și cu bitul de STOP ('1'); deci, registrul va avea 10 biți și se va deplasa la dreapta. Modulul transmițătorului conține și un buffer comandat de semnalul TxEn. Atunci când semnalul TxEn este activat, ieșirea serială a registrului de deplasare va fi transmisă pe linia Tx, iar în caz contrar linia Tx va trece în starea 1 logic, corespunzătoare intervalului de pauză. Unitatea de control se poate implementa prin automatul de stare cu diagrama ilustrată în figura 5. Se utilizează un semnal CntBit pentru contorizarea biților transmiși pe linia serială și un semnal CntRate pentru contorizarea ciclurilor de ceas în scopul măsurării intervalului de timp în care trebuie menținut fiecare bit pe linia serială. Atunci când semnalul Start devine activ, se trece în starea load, în care se încarcă registrul de deplasare cu octetul care trebuie transmis, completat cu biții de START și de STOP. În continuare, se trece în starea send, în care se va reveni după transmisia unui bit. În starea waitbit se așteaptă trecerea intervalului de timp egal cu durata unui bit, incrementând contorul CntRate. Dacă acest contor ajunge la valoarea T_BIT (numărul ciclurilor de ceas corespunzător duratei unui bit), se trece în starea shift, în care se deplasează la dreapta registrul de deplasare și se incrementează contorul CntBit. Dacă nu s-au transmis toți cei 10 biți, se revine în starea send, iar în caz contrar se revine în starea ready în care se activează semnalul TxRdy, indicând terminarea transmiterii unui octet.

Jumpers

Pmod BT2 are mai multe setări care pot fi configurate prin intermediul blocurilor jumper JP1 până la JP4. Acesti jumperi sunt toți eșantionați în primii 500 ms de funcționare și configurează comportamentul modulului RN-42.

3.4 Interfata SPI

Interfața seriala SPI *(Serial Peripheral Interface)* este o interfața sincronă standard de mare viteză, ce operează în mod full duplex. Numele ei a fost dat de Motorola. Ea e folosită ca sistem de magistrală serială sincronă pentru transmiterea de date, unde circuitele digitale pot să fie interconectate pe principiul master-slave. Aici, modul master/slave înseamnă că dispozitivul (circuitul) digital master inițiază cuvântul de date. Mai multe dispozitive (circuite) digitale slave sunt permise cu *slave select individual*, adică cu selectare individuală.

SPI-ul are patru semnale logice specifice:

- **SCLK** - *Ceas serial* (ieșire din master).
- **MOSI/SIMO** - *Master Output, Slave Input* (ieșire master, intrare slave).
- **MISO/SOMI** - *Master Input, Slave Output* (intrare master, iesire slave).
- **SS** - *Slave Select* (active low, ieșire din master).

Transmisia de date

Pentru a începe comunicarea, master-ul mai întâi configurează ceasul, folosind o frecvență mai mică sau egală cu maximul frecvenței suportata de slave. Aceste frecvențe sunt de obicei în intervalul 1-70 MHz. Atunci master-ul setează slave select-ul pe nivelul 'jos' (en. low) pentru chip-ul dorit. Dacă este necesară o perioada de așteptare (ca la conversia analog-digitală) atunci master-ul așteaptă cel puțin acea perioadă de timp înainte de a începe ciclurile de ceas.

În timpul fiecărui ciclu de ceas SPI, apare o transmisie full duplex:

- master-ul trimite un bit pe linia MOSI; slave-ul îl citește de pe aceeași linie;
- slave-ul trimite un bit pe linia MISO; master-ul îl citește de pe aceeași linie.

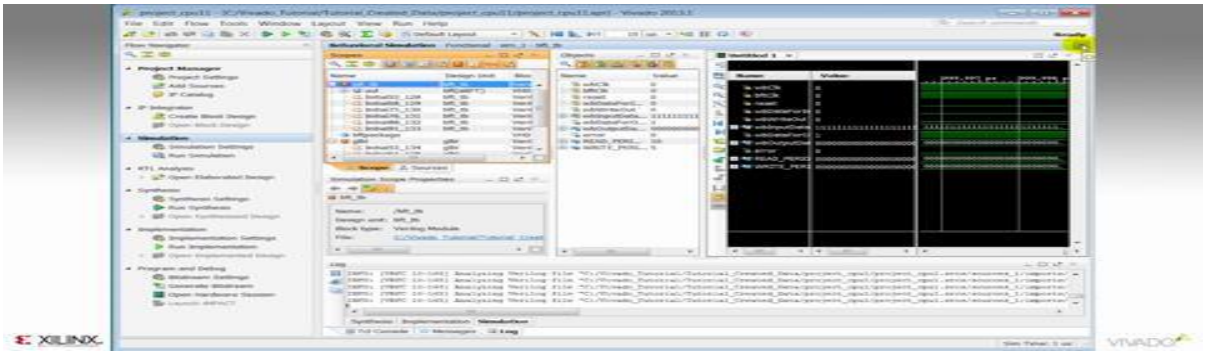
Nu toate transmisiile de date necesita toate aceste operații (de ex. transmisia unidirecționala) deși acestea se petrec.

În mod normal, transmisia implică existența a doi *regiștri de date* de o lungime oarecare a cuvântului, cum ar fi opt biți, unul situat în dispozitivul master și celalalt în dispozitivul slave; ei sunt conectați într-o configurație de tip inel. Informația este de obicei transferată începând cu cel mai semnificativ bit (eng: *Most Significant Bit - MSB*), și continuând bit cu bit până se transferă și cel mai nesemnificativ bit (eng: *Least Significant Bit - LSB*) pentru același registru. În această fază putem afirma că cele două dispozitive master/slave și-au schimbat valorile din regiștri. Imediat după, fiecare dispozitiv citește valoarea stocată în registrul de date și o prelucrează, cum ar fi scrierea într-o locație de memorie. Dacă mai sunt date de schimbat, regiștrii de schimb sunt încărcați cu noi date și procesul se repetă.

3.5 Vivado

Vivado Design Suite este un produs software ce aparține de Xilinx, folosit pentru sinteza și analiza design-urilor HDL. Vivado permite dezvoltatorilor să sintetizeze desene, să efectueze analize de sincronizare, să examineze diagrame RTL, să simuleze reacția unui design la diferiți stimuli, etc. Acesta este compatibil doar cu produsele FPGA realizate de Xilinx.

Vivado le permite dezvoltatorilor să sintetizeze design-urile, să efectueze analize de sincronizare, să examineze diagrame RTL, să simuleze reacția unui design la diferiți stimuli și să configureze dispozitivul țintă cu programatorul. Vivado este un mediu de design pentru produsele FPGA de la Xilinx și este strâns cuplat la arhitectura unor astfel de cipuri și nu poate fi utilizat cu produsele FPGA de la alți furnizori.



[Fig. 7]

Imagini Vivado

3.6 Unitate aritmetică și logică

Unitatea aritmetică logică este un circuit electronic digital complex care poate efectua operații aritmetice și logice. În diagramele-bloc de computere, unitatea aritmetică logică este reprezentată ca un modul funcțional, componentă a schemei de principiu a unui calculator electronic. Constructiv, în calculator, UAL este un bloc fundamental al unității centrale de procesare (prelucrare) UCP, în engleză CPU.

UAL asigură funcții de prelucrare a datelor, respectiv:

- efectuare de operații aritmetice;
- efectuare de operații logice;
- efectuarea altor operații specifice, la nivel de bit asupra operanzilor.

Întrucât efectuarea acestor operații se reprezintă considerând operanzii reprezentați în baza 2 (reprezentare binară), granița între tipurile de operații este greu de stabilit.

Într-un sistem de calcul modern, granița fizică a unității aritmetice-logice este greu de precizat, funcțiile respective fiind regăsite în cadrul microprocesorului, care, începând cu calculatoarele de generația a IV-a, este un circuit integrat.

3.6.1 Adunarea

Adunarea este o operație aritmetică elementară care totalizează două sau mai multe numere, numite „termenii adunării" într-o singură valoare, numită suma sau „totalul" numerelor respective.

Mecanismele calculatoarelor mecanice trebuiau să fie tot timpul pornite, rotite și oprite. Greutatea părților mobile limita viteza cu care puteau fi efectuate asemenea operații și, deci, și rata de execuție a calculatoarelor. însă aceste probleme au fost eliminate o dată cu introducerea calculatoarelor electronice, în anii 1940. Folosind numere binare - un sistem de numărare bazat pe numere luate două câte două - electronii care se deplasau în circuite îndeplineau sarcina efectuată anterior de părți mecanice. Viteza cu care funcționau calculatoarelor a arătat că începuse o mare revoluție în tehnologie. Cu mulți ani mai târziu, această tehnologie de bază, dar în formă miniaturizată, ne-a oferit încă o unealtă pentru lucrul cu numerele - calculatorul de buzunar.

3.6.2 Scăderea

Scăderea este o operație aritmetică care reprezintă operația de scoatere a obiectelor dintr-o colecție. Rezultatul unei scăderi se numește diferență. Scăderea este descrisă prin semnul minus (-).Scăderea reprezintă eliminarea sau scăderea cantităților fizice și abstracte folosind diferite tipuri de obiecte, inclusiv numere negative, fracții, numere iraționale, vectori, zecimale, funcții și matrici.

Scăderea urmează mai multe tipare importante. Este anticommutativă, ceea ce înseamnă că schimbarea ordinii schimbă semnul răspunsului. De asemenea, nu este asociativă, ceea ce înseamnă că atunci când se scad mai mult de două numere, ordinea în care se efectuează scăderea contează. Deoarece 0 este identitatea aditivă, scăderea acesteia nu schimbă un număr. Scăderea

respectă, de asemenea, reguli previzibile referitoare la operațiunile conexe, cum ar fi adăugarea și înmulțirea. Toate aceste reguli pot fi dovedite, începând cu scăderea numerelor întregi și generalizând prin numerele reale și nu numai. Operațiile binare generale care continuă aceste tipare sunt studiate în algebră abstractă.

3.6.3 Înmulțirea

Înmulțirea este una dintre cele patru operații matematice elementare ale aritmeticii, celelalte fiind adunarea , scăderea și divizarea.

Înmulțirea numerelor întregi poate fi gândită ca o adăugare repetată; adică înmulțirea a două numere este echivalentă cu adăugarea a cât mai multe copii ale unuia dintre ele, multiplicând, ca valoare a celuilalt, multiplicatorul. Multiplicatorul poate fi scris primul și multiplicandul al doilea; ambii pot fi numiți factori.

3.6.4 Împărțirea

Împărțirea este una dintre cele patru operații de bază ale aritmeticii, modalități prin care numerele sunt combinate pentru a face numere noi. Celelalte operații sunt adunarea, scăderea și înmulțirea (care poate fi privită ca inversul împărțirii).

Împărțirea cu rest sau împărțirea euclidiană a două numere naturale oferă un coeficient, care este numărul de câte ori cel de-al doilea este conținut în primul și un rest, care este parte a primului număr care rămâne.

3.6.5 Deplasare stânga. Deplasare dreapta

În programarea computerului, o deplasare aritmetică este un operator de deplasare, uneori denumit deplasare cu semn (deși nu este restricționat la operanzi deplasați). Cele două tipuri de bază sunt deplasarea aritmetică la stânga și deplasarea aritmetică la dreapta. În cazul numerelor binare, este o operație în formă de biți care schimbă toate pozițiile biților operandului său; fiecare bit din operand este pur și simplu mutat un anumit număr de poziții de biți, iar pozițiile de bit vacante sunt completate. În loc de a fi umplute toate cu 0, ca în deplasarea logică, când treceți la dreapta, bitul din stânga (de obicei bitul de semn în reprezentări întregi cu semn) este reprodus pentru a completa toate pozițiile vacante (este un fel de extensie de semn).

În știința computerelor, o deplasare logică este o operație pe biți care schimbă toți biții din operandul său. Cele două variante de bază sunt deplasarea logică la stânga și deplasarea la dreapta logică. Aceasta este modulată în continuare de numărul de poziții de biți, o valoare dată trebuie schimbată, cum ar fi deplasarea stânga cu 1 sau deplasarea spre dreapta cu n. Spre deosebire de o schimbare aritmetică, o schimbare logică nu păstrează bitul unui semn și nu distinge exponentul unui număr de semnificația sa (mantissa); fiecare bit din operand este pur și simplu mutat un anumit număr de poziții de biți, iar pozițiile de bit vacante sunt umplute, de obicei cu zerouri și, eventual cu 1.

3.7 Metode folosite

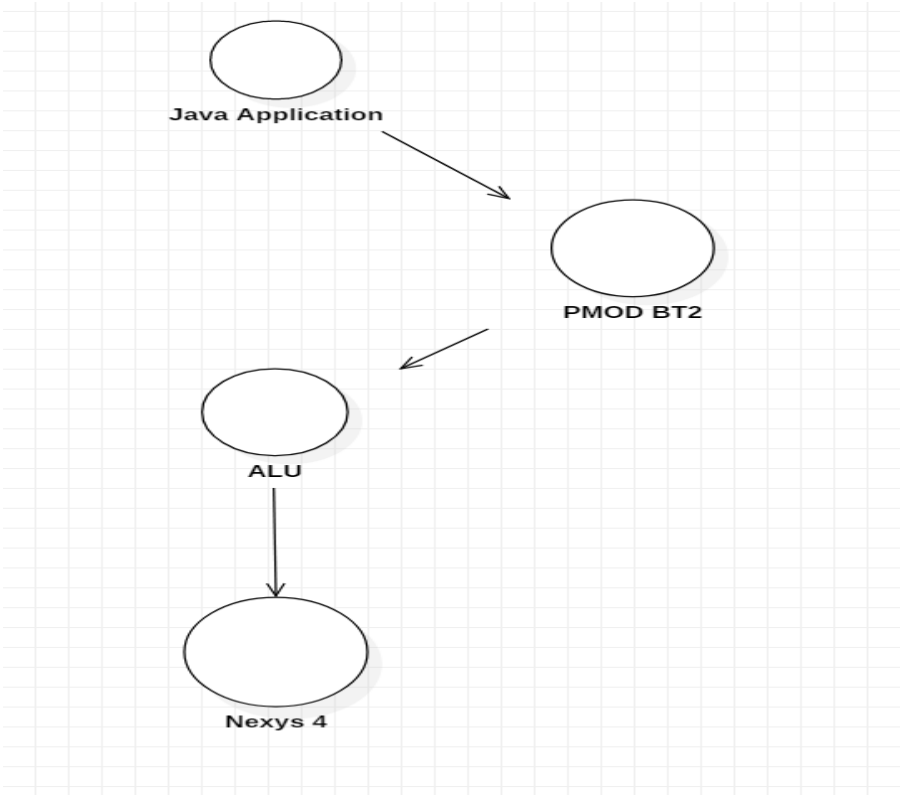
Pentru realizarea operațiilor din unitatea aritmetică logică am folosit ca și metode următoarele: pentru adunare un sumator cu anticiparea transportului, pentru scădere ne-am folosit de sumatorul de la adunare dar pe locul celui de-al doilea termen am pus complementul față de 2 al celui de-al doilea număr, pentru înmulțire am folosit un înmulțitor Booth, iar pentru împărțire un circuit de împărțire cu refacerea restului parțial. Pentru operația de deplasare stânga-dreapta am folosit un registru de deplasare învățat la materiile de hardware din semestrele trecute (ASDN). Aplicația Java va consta în principal dintr-un GUI în care se vor tasta diferite valori pentru numerele pe care se vrea a fi făcute operațiile aritmetice. Aplicația Java va comunica cu placa Nexys 4 DDR printr-un modul Bluetooth PMOD BT2.

4. Proiectare și implementare

4.1 Soluția aleasă

Soluția aleasă pentru rezolvarea proiectului este de a implementa o unitate aritmetică logică care să efectueze următoarele operații: adunare, scădere, înmulțire, împărțire și deplasare stânga-dreapta. Operația de adunare se face face printr-un sumator cu anticiparea transportului, cea de scădere realizându-se la fel, adică ne folosim de sumatorul de la adunare dar pe locul celui de-al doilea termen vom pune complementul față de doi al celui de-al doilea număr din operație. Înmulțirea se va realiza printr-un circuit de înmulțire prin metoda Booth, iar împărțirea printr-un circuit de împărțire cu refacerea restului parțial. Pentru deplasări stânga-dreapta ne vom folosi de circuitele de deplasare învățate în semestrele trecute la materiile asemănătoare. Aplicația Java se va face în IDEA-ul Intellij iar legătura dintre placă și aplicația Java va fi făcută printr-un modul Bluetooth PMOD BT2. Mai multe detalii despre toate acestea în sectiunile din capitolele următoare.

4.2 Schemă proiect



[Fig. 8]

Schema generală a proiectului

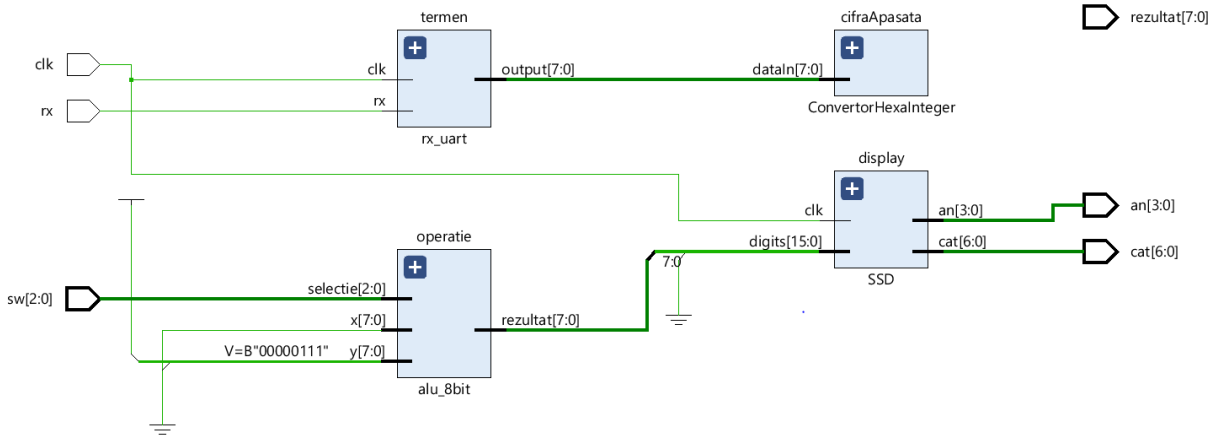
Java application se referă la aplicația Java prin care utilizatorul va da valorile pe care se vor efectua operațiile aritmetice. Ea va fi făcută în Intellij IDEA și va reprezenta în principal un GUI format din text box-uri în care se vor tasta numere întregi.

PMOD BT2 este modulului Bluetooth care ne va ajuta să comunicăm cu placa FPGA Nexys 4 DDR. Datele din aplicația Java se vor transmite plăcii prin acest modul.

ALU este unitatea aritmetică-logică ce va fi implementată în limbajul de descriere hardware VHDL. Ea reprezintă partea din proiect care se va ocupa cu efectuarea următoarelor operații aritmetice și logice: adunare, scădere, înmulțire, împărțire și deplasare stânga-dreapta.

Nexys 4 DDR este placa FPGA pe care noi vom verifica rezultatele și pe care noi vom implementa unitatea aritmetică logică.

4.2.1 Schemă detaliată VHDL



[Fig. 9]

Schema detaliată a proiectului

În schema de mai sus sunt prezentate principale entități ale părții de proiect scrisă în VHDL. În ea putem observa 4 componente mari și anume: rx_uart, alu_8bit, SSD, convertorHexaInteger.

1. **rx_uart** este folosit pentru a realiza conexiunea dintre aplicația Java și placa de dezvoltare FPGA.
2. **alu_8bit** conține metodele prin care se realizează operațiile pe numerele transmise plăcii, aceste metode fiind: adunarea, scăderea, înmulțirea și împărțirea.
3. **SSD** este un circuit prin care afișăm rezultatul operațiilor pe Seven Segment Display-ul plăcii FPGA.
4. **ConvertorHexaInteger** este o componentă ce face conversia din codul ASCII al cifrelor primite prin modulul Bluetooth de către placă în codul binar al numerelor întregi aferente codurilor ASCII.

4.3 Algoritmii implementați

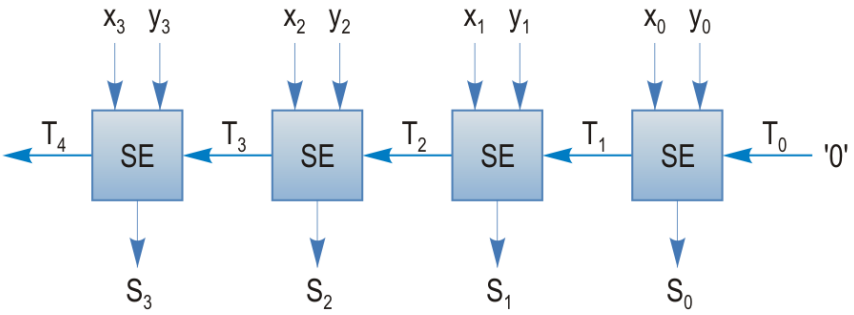
4.3.1 Sumatorul cu anticiparea transportului

Adunarea este operația utilizată cel mai frecvent într-un sistem de calcul. Operațiile aritmetice mai complexe ale ALU se reduc până la urmă la o serie de adunări. Prin creșterea vitezei operației de adunare se poate crește și viteza ALU. Cu alte cuvinte, viteza și costul circuitelor de adunare sunt proporționale cu complexitatea acestora. Pentru acest proiect am ales sumatorul cu propagarea succesivă a transportului.

Sumatorul cu propagarea succesivă a transportului sau "Ripple Carry Adder" este un sumator paralel. În acest tip de sumator transportul trebuie să se propage succesiv prin toate sumatoarele înainte de a se cunoaște rezultatul final. Algoritmul de adunare cu propagarea succesivă a transportului este:

```
  X3 X2 X1 X0 +
  Y3 Y2 Y1 Y0
-----------------
S4 S3 S2 S1 S0
```

O posibilitate de implementare ar fi conectarea mai multor sumatoare elementare în serie, exact cum arată schema de mai jos a unui sumator pe 4 biți:



[Fig. 10]

Sumator elementar pe 4 biți

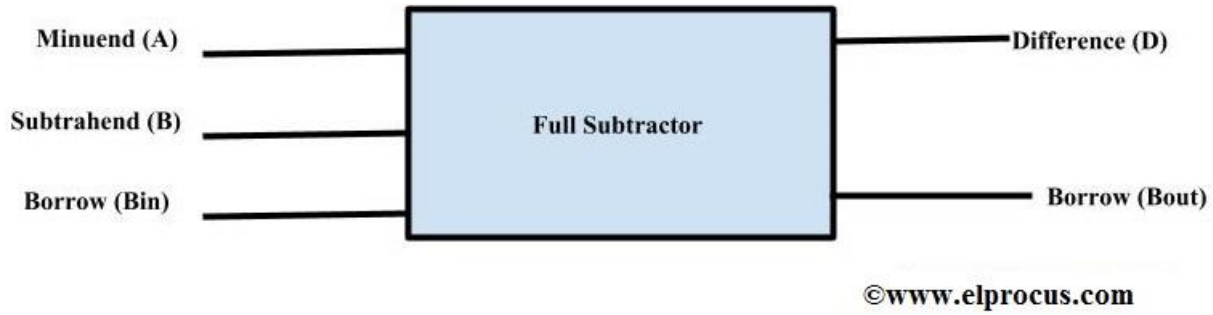
Avantajele acestui tip de sumator sunt simplitatea schemei circuitului și costul relativ redus de implementare. Ca și dezavantaj principal ar fi viteza redusă a sumatorului.

4.3.2 Adunarea cu complementul față de 2 a unui număr

La fel ca și adunarea, scăderea este o operație foarte des utilizată în sistemele de calcul, ea fiind inversul adunării. Scăderea în binar este tot o adunare: A - B = A + (-B)

Pentru realizarea completă a ALU am folosit un scăzător complet. Acest tip de scăzător este una dintre cele mai folosite și esențiale circuite din logica combinațională. Circuitul acesta de scădere poate fi făcut cu porți logice cum ar fi: OR, XOR sau NAND.

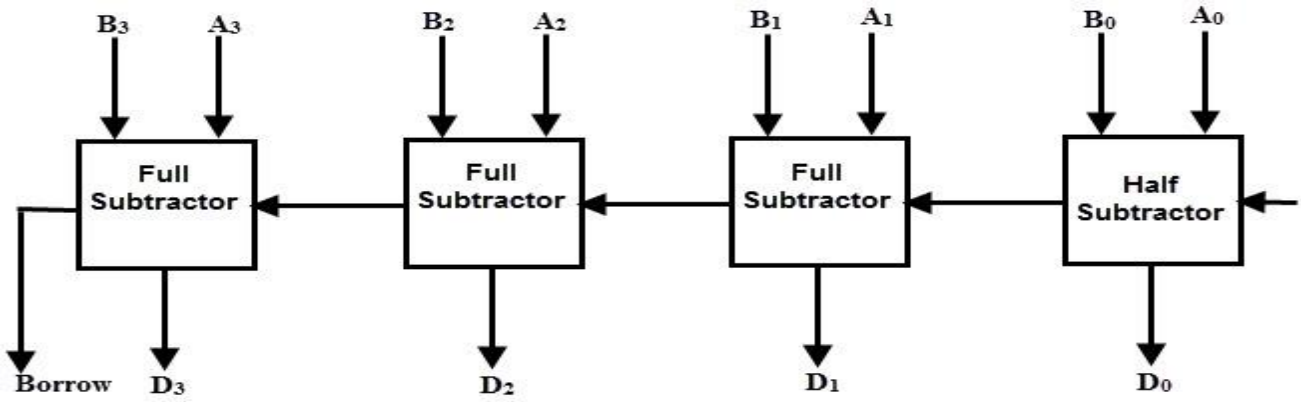
Scăzătorul complet are 3 intrări: descăzutul A, scăzătorul B și împrumutul Bin. Tot el, generează două ieșiri: diferența D și împrumutul următor Bout. Schema generală a unui scăzător complet este:



[Fig. 11]

Scăzător complet

Un scăzător complet binar pe 4 biți ar arăta în felul următor:



[Fig. 12]

Schema internă a unui scăzător pe 4 biți

4.3.3 Tehnica înmulțirii prin deplasare și adunare

Înmulțirea este una din cele 4 operații aritmetice de bază. Înmulțirea numerelor binare este similară cu cea a numerelor zecimale. Avem doi operanzi: deînmulțit, înmulțitor și un rezultat, produsul. Pentru realizarea ALU am folosit tehnica înmulțirii prin deplasare și adunare. Aceasta adună deînmulțitul X cu el însuși de Y ori. Algoritmul acestei metode este următorul:

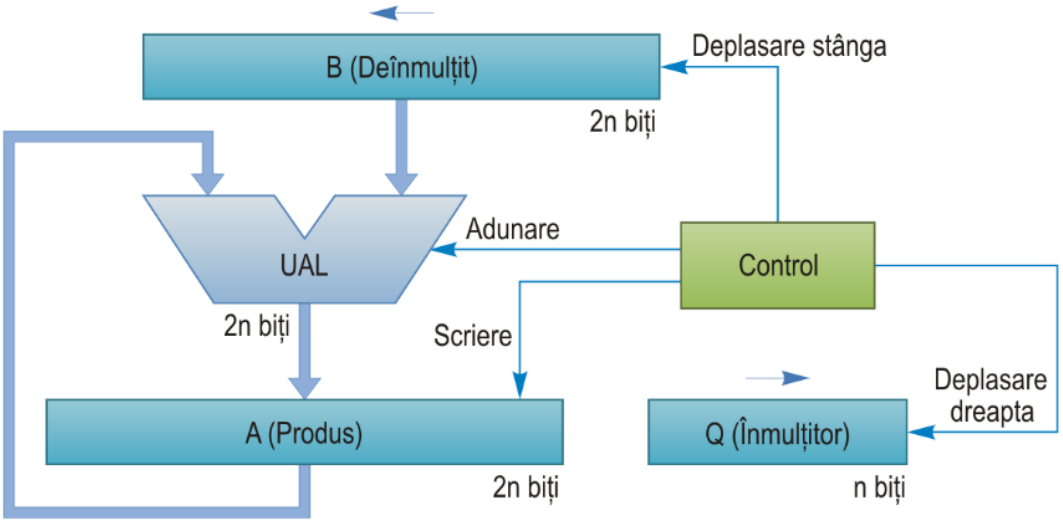
1. Se selectează cifrele înmulțitorului una câte una de la dreapta la stânga.
2. Se înmulțește deînmulțitul cu cifra selectată a înmulțitorului.
3. Se plasează produsul intermediar la stânga rezultatelor precedente.

În cazul înmulțirii binare, cifrele sunt 0 sau 1. Avem următorul exemplu:

X=9 (1001), Y=10 (1010)

```
   1001 *
   1010
  ------
   0000
  1001
 0000
1001
-------
1011010
```

Schema unui astfel de înmulțitor ar fi următoarea:



[Fig. 13]

Schema logică a înmulțirii prin deplasare și adunare

4.3.4 Împărțirea cu refacerea restului partial

Împărțirea este operația matematică de forma:

X / Y = Q rest R

Aceasta mai poate fi scrisă și sub forma:

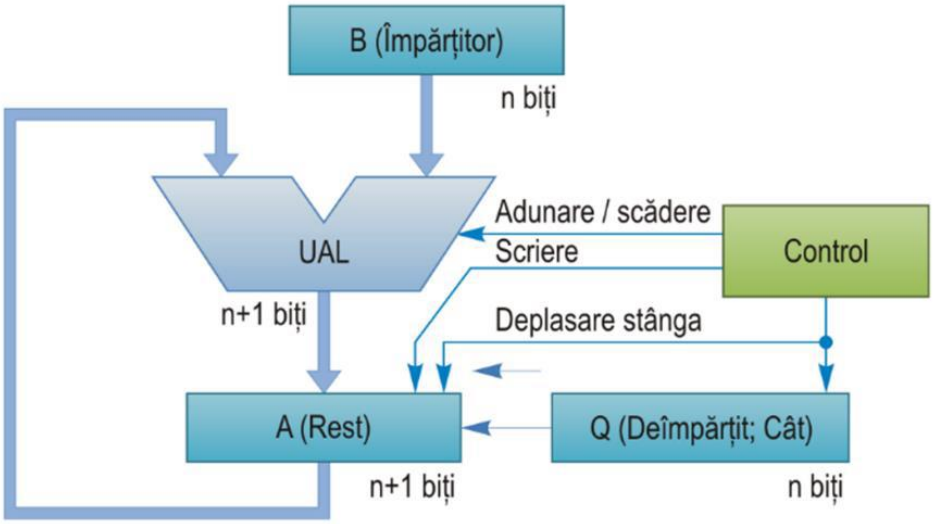
X = Q * Y + R unde:

Împărțirea cu refacerea restului partial

Algoritmul presupune deplasarea restului partial la stânga cu o poziție și a câtului la stânga cu o poziție. Se încarcă deîmpărțitul și împărțitorul în câte un registru. Pentru a determina dacă

împărțitorul e mai mic ca și restul parțial, scădem din registrul împărțitorului, registrul deîmpărțitului. Dacă rezultatul e negativ, trebuie refăcută valoarea precedentă, adunând împărțitorul și restul.

Pentru metodele de împărțire cu refacerea restului parțial și fără refacerea restului parțial, folosim circuitul:



[Fig. 14]

Schema logică a împărțirii cu refacerea restului parțial

4.3.5 Deplasări folosind regiștri de deplasare

În circuitele digitale, un registru de deplasare este o cascadare de bistabile, care împărtășesc același semnal de ceas și în care ieșirea fiecărui bistabil este conectată la intrarea de date a următorului bistabil. Acestea sunt registre care, pe lângă funcția de stocare a informației pe durata unui impuls de ceas, pot realiza și deplasarea și circularea (rotirea) acesteia, de asemenea sincron cu tactul.

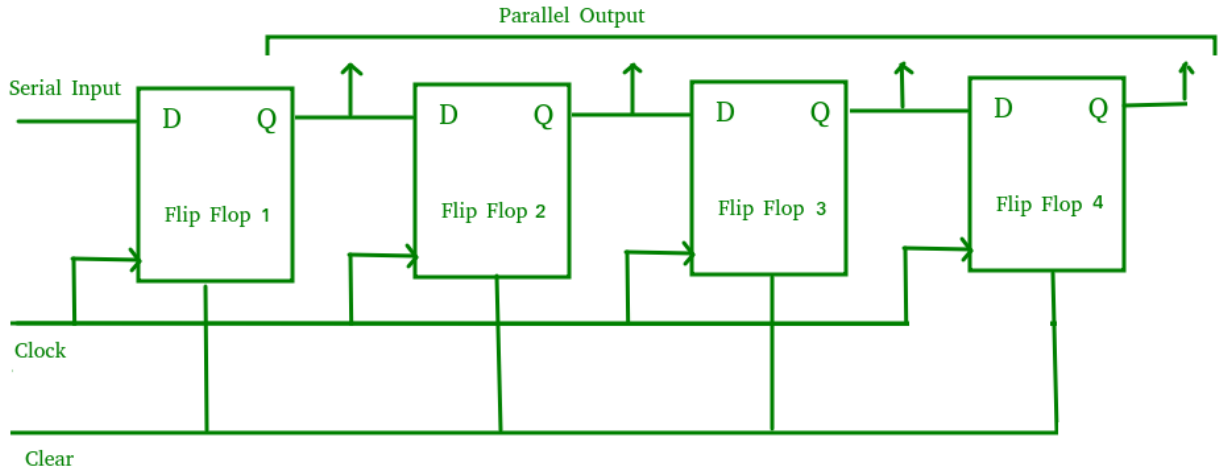
Registrele de deplasare pot avea atât intrări cât și ieșiri paralele și seriale. Acestea sunt adesea configurate ca "serial-in, parallel-out" (SIPO) sau ca "parallel-in, serial-out" (PISO).

Deplasarea se poate face:

- De la stânga la dreapta: Q0 → Q1 → Q2 → Q3
- De la dreapta la stânga: Q3 → Q2 → Q1 → Q0

Ținând cont de codificarea binară a informației, deplasarea spre stânga va avea semnificația unei înmulțiri cu 2 a numărului în binar, iar deplasarea spre dreapta va avea semnificația unei împărțiri la 2 a numărului în binar.

Schema unui circuit de deplasare pe 4 biți este următoarea:



[Fig. 15]

Schema unui circuit de deplasare

4.3.6 Aprindere leduri

Pentru a mări complexitatea proiectului am ales în a adăuga butoane în aplicația Java pentru aprinderea ledurilor de pe placa Nexys 4. Putem alege ce leduri de pe placă să se aprindă sau să se stingă cu un simplu click în aplicație. Putem chiar face un joc de lumini aprinzând alternativ ledurile.

4.4 Detalii de implementare

4.4.1 Conexiunea cu PMOD BT2

Atat placa Nexys4 DDR cat si PMOD BT2 au fiecare cate doua module: de transmis date respectiv de primit date. Cel de transmis date se numeste UART TX, iar cel de primit date UART RX. Pentru a conecta placa cu PMOD-ul a trebuit sa conectam UART TX de la PMOD BT2 cu UART RX de la placa, in acest mod placa primind date prin intermediul Bluetooth de la dispozitivul mobil. Tot acest lucru s-a realizat cu ajutorul aplicatiei Serial Bluetooth Terminal, o aplicatie conceputa pentru Android si descarcata din Magazin Play.

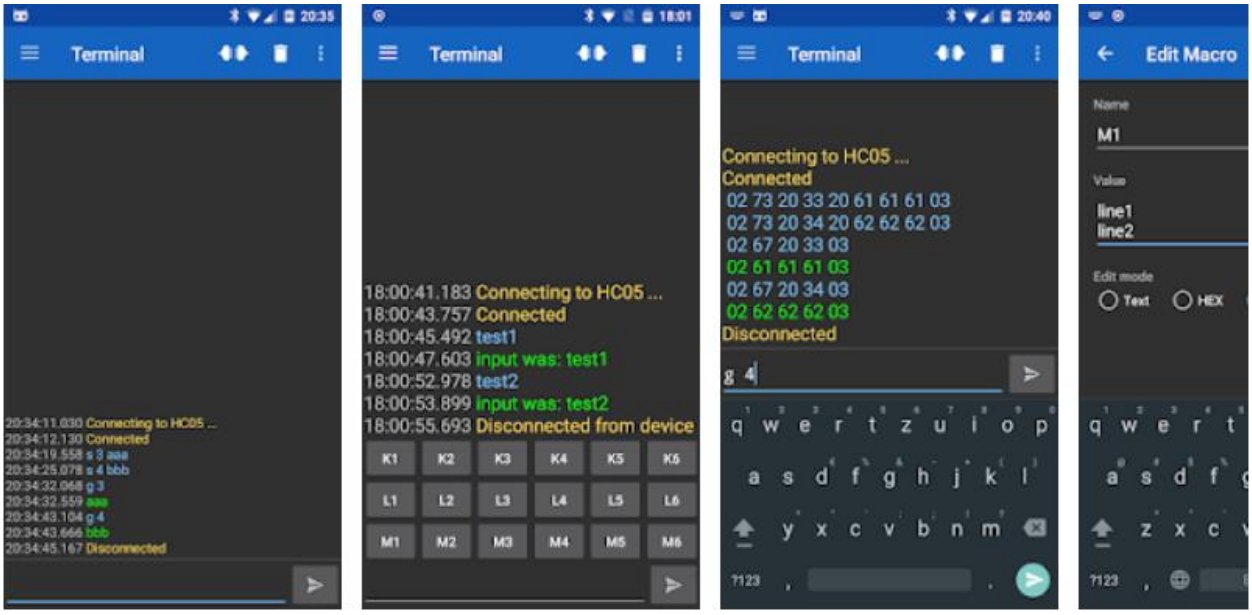
4.4.2 Serial Bluetooth Terminal

Serial Bluetooth Terminal is a line-oriented terminal / console app for microcontrollers, arduinos and others devices with a serial / UART interface connected with a bluetooth to serial converter to your android device.

This app supports different bluetooth versions:

-Bluetooth Classic

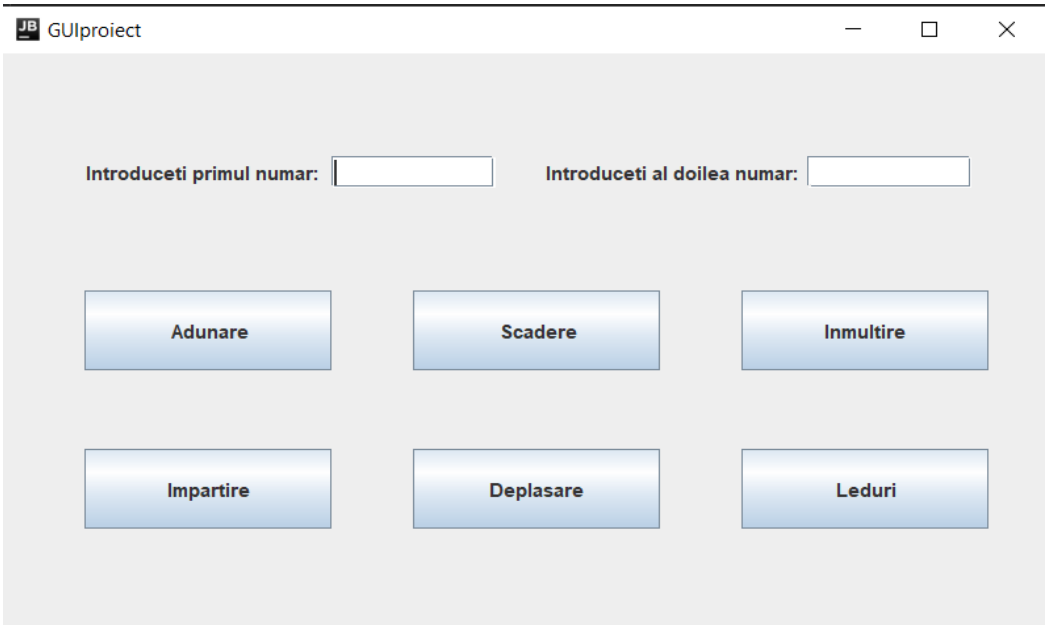
-Bluetooth LE / Bluetooth Low Energy / BLE / Bluetooth Smart



[Fig. 16]

Imagini cu aplicația Serial Bluetooth Terminal furnizate de producător

4.4.3 Aplicația Java



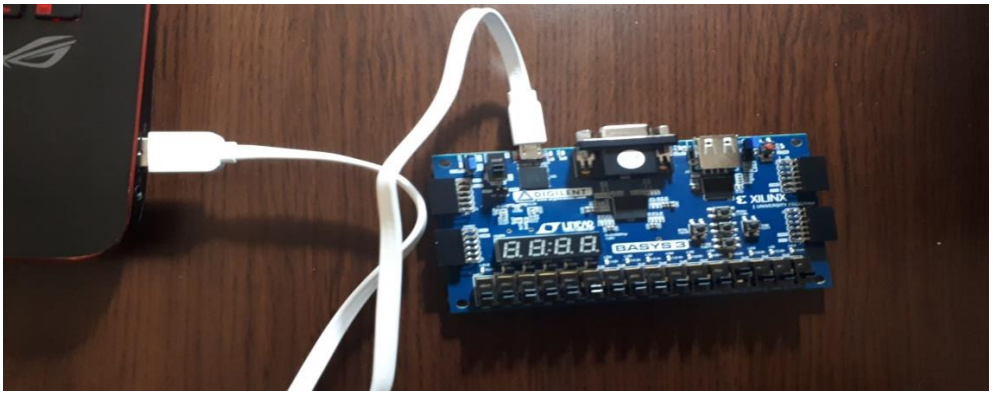
[Fig. 16]

Imagine cu aplicația Java

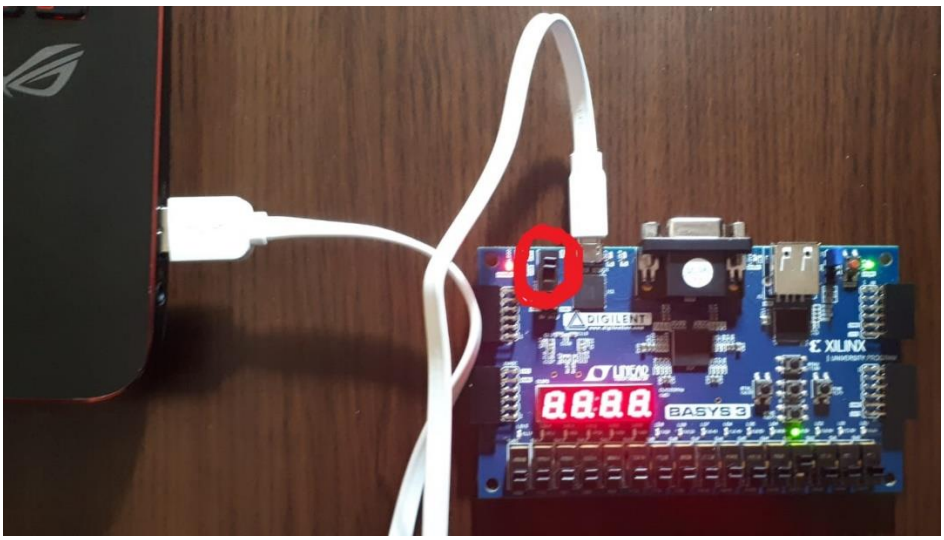
4.5 Manual de utilizare

Pentru a testa aplicația avem nevoie de următoarele: placă de dezvoltare FPGA(Nexys4 DDR sau Basys 3), modul Bluetooth PMOD BT2, cablu de date USB și un mediu de dezvoltare pentru rularea unui cod JAVA și a codului scris în VHDL. Pașii prin care putem verifica funcționalitatea proiectului sunt:

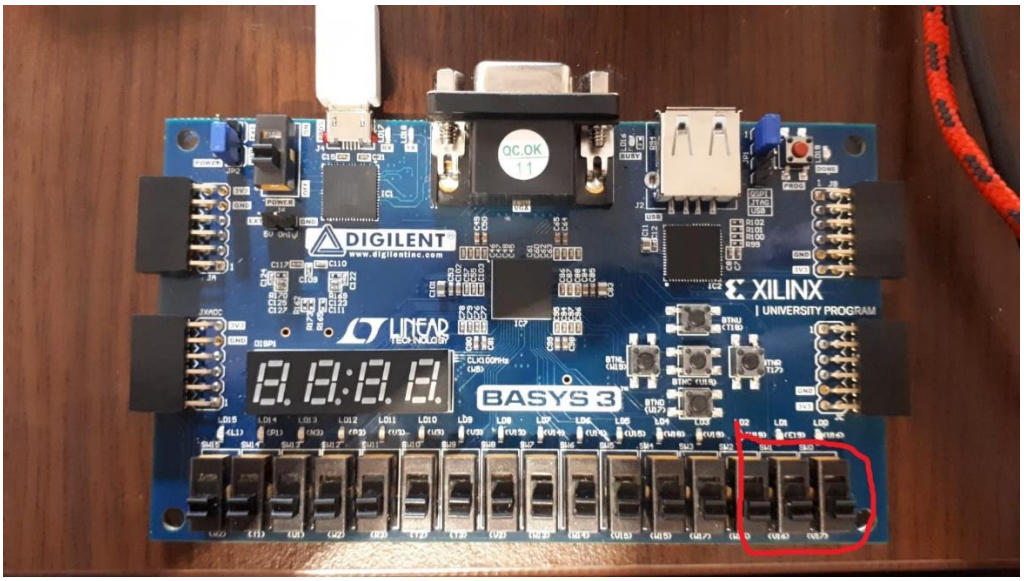
1. Conectarea plăcii FPGA la stație prin intermediul cablului de date.



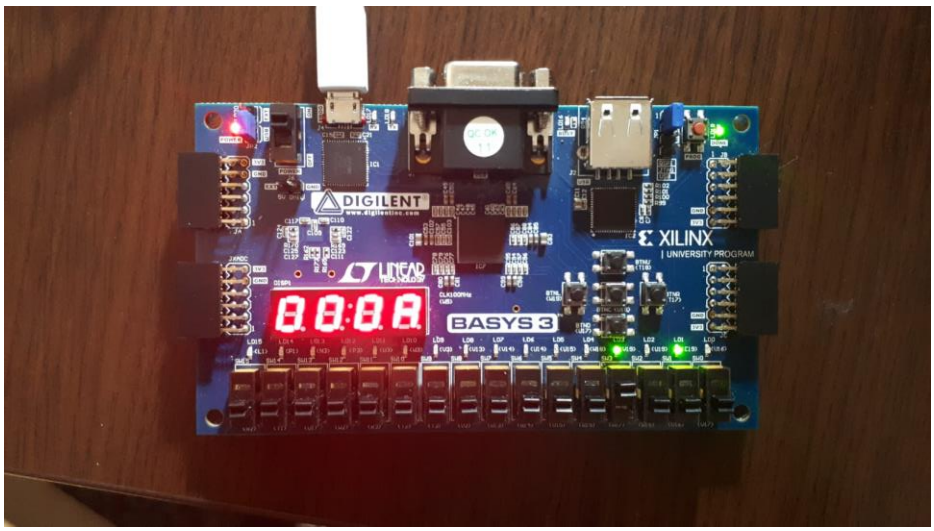
2. Pornirea plăcii FPGA prin comutatorul de pornire, switch-ul aflat în partea stângă sus.



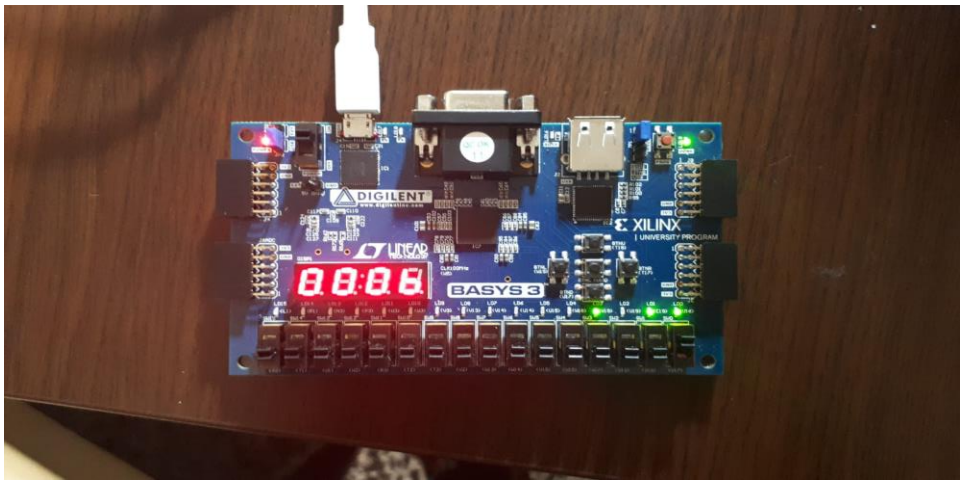
3. Generarea bitstream-ului și încărcarea codului pe placa de dezvoltare.
4. Introducerea modului PMOD BT2 în portul 1 al plăcii din partea dreaptă jos.
5. Pornire aplicație Java.
6. Introducerea numerelor și trimiterea acestora plăcii de dezvoltare prin modului FPGA.
7. Vizualizarea rezultatelor operațiilor ALU în funcție schimbarea lor prin comutarea primelor 3 switch-uri de pe placă din partea dreapta jos.



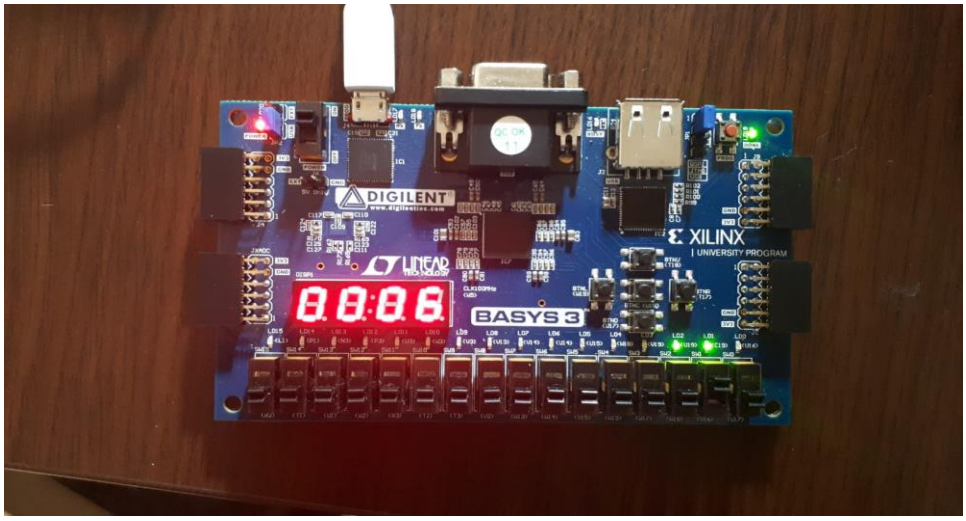
8. În funcție de operația dorită trebuie activate următoarea combinație a switch-urilor:
   - **000 → adunare**



   - **001 → adunare cu transport**



   - **010 → scăderea**

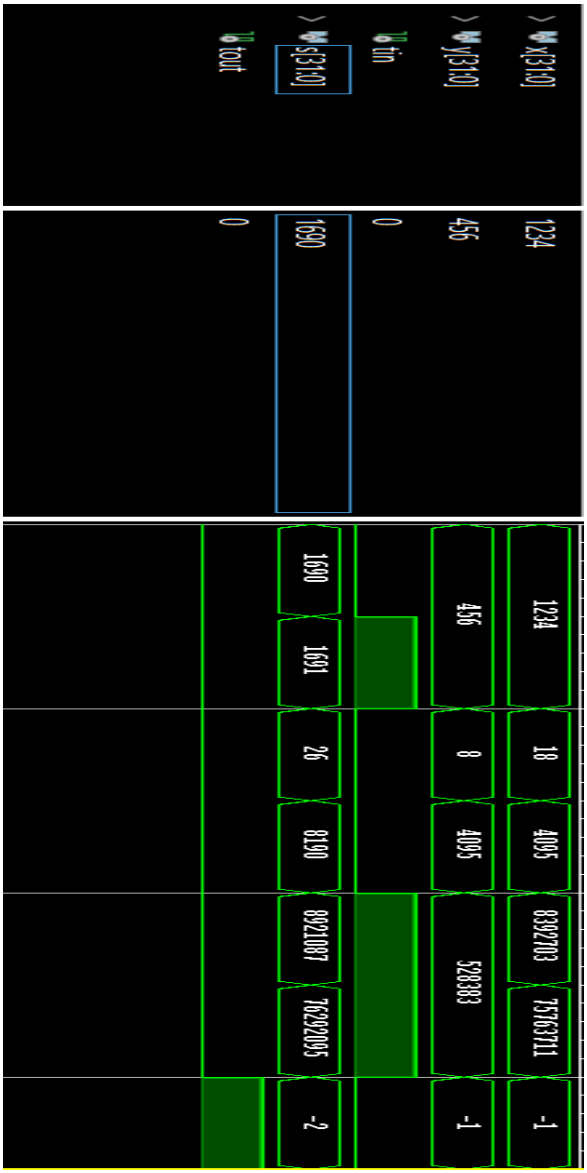


- **011 → scăderea în complement față de 2**
- **100 → înmulțirea**
- **101 → împărțirea**

9. Vizualizarea rezultatelor pe Seven Segment Display-ul plăcii.

5. Rezultate experimentale

5.1 Simulări componente



[Fig. 18]

Simulare sumator pe 32 de biți

Simularea sumatorului pe 32 de biți

La această simulare avem un x reprezentând primul număr (4d2 îi reprezentarea în hexazecimal al numărului 1234 din baza 10), un y (1c8 echivalentul lui 456 în baza 10), un Tin care îi Carry in-ul, o sumă (69a îi 1690 în baza 10) și un Carry out.

4d2 + 1c8 = 69a ⇔

1234+456=1690

[Fig. 19]

Simulare scăzător pe 32 de biți

Simularea scăzătorului pe 32 de biți

În această simulare putem vedea funcționalitatea scăzătorului pe 32 de biți. Avem un x reprezentând descăzutul, respectiv y care îi scăzătorul. Rezultatul diferenței se poate vedea în semnalul diferența.

639265-1869839=-1230574

[Fig. 20]

Simulare circuit de deplasare pe 32 de biți

Simularea circuitului de deplasare pe 32 de biți

Mai jos putem observa simularea circuitului de deplasare pe 32 de biți. Avem de analizat 3 intrări și o ieșire. dataIn reprezintă semnalul de intrare care se va deplasa la stânga sau la dreapta. selPoziţiiDeplasare ne spune cu câte poziții se va deplasare dataIn iar selDeplasare direcția de deplasare. Rezultatul deplasat se va stoca în dataOut.

04000007 deplasat cu două poziții

[Fig. 21]

Simulare circuit de înmulțire pe 32 de biți

Simularea circuitului de înmulțire prin metoda Booth pe 32 biți

Mai jos putem observa simularea circuitului de deplasare prin metoda Booth pe 32 de biți. Avem ca intrări un semnal de clock, semnale pentru reset și start. Numerele ce vor fi înmulțite sunt reprezentate de x și y, iar rezultatul va fi stocat în semnalele de ieșire a și q.

789*986=777954

6. Concluzii

Proiectul realizat poate fi folosit ca un sistem ce realizează operațiile elementare cu numere transmise prin Bluetooth. Pentru realizarea lui s-au folosit plăci FPGA (Basys 3 și Nexys 4 DDR), modul PMOD BT2 și ca limbaje s-a folosit Java pentru aplicația de desktop și VHDL pentru descrierea hardware a componentelor ce au fost implementate pe placă. Pe placa FPGA s-au implementat operațiile de bază din ALU și legătura cu modulul Bluetooth.

6.1 Contribuții originale

Au existat mai multe versiuni de sisteme gen acesta. Contribuțiile originale, acel plus valoare adus sistemului de către echipa ce a implementat proiectul a fost aplicația Java făcută într-un mod destul de original și modul în care s-au implementat operațiile din ALU.

6.2 Dezvoltări ulterioare

Pe viitor proiectul ar putea fi îmbunătățit pentru început adăugând și alte operații mai complexe pe unitatea aritmetică și logică, cum ar fi ridicarea la putere și extragerea rădăcinii pătrate. O altă noutate ce ar putea veni la sistem ar fi să facem o legătură pentru transmiterea datelor și prin Wifi, nu doar prin Bluetooth cum există acum deja.

7. Bibliografie

7.1 Link-uri la referințe legate de implementări

1. **Am luat de la link-ul acesta referințe legate de modulul PMOD BT2 ce erau disponibile la data de 19.10.2019**
   https://reference.digilentinc.com/reference/pmod/pmodbt2/reference-manual
2. **Am luat de la link-ul acesta referințe legate de placa Nexys 4 DDR ce erau disponibile la data de 23.10.2019**
   https://store.digilentinc.com/nexys-4-ddr-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/
3. **Am luat de la link-ul acesta referințe legate de scăderea ca operație matematică ce erau disponibile la data de 14.10.2019**
   https://en.wikipedia.org/wiki/Subtraction
4. **Am luat de la link-ul acesta referințe legate de împărțirea ca operație matematică ce erau disponibile la data de 16.11.2019**
   https://en.wikipedia.org/wiki/Division_(mathematics)
5. **Am luat de la link-ul acesta informații legate de deplasarea logică ce erau disponibile la data de 15.11.2019**

https://en.wikipedia.org/wiki/Logical_shift

6. **Am luat de la link-ul site-ului domnului profesor Baruch informații legate de implementarea circuitelor de adunare, scădere, înmulțire și împărțire ce erau disponibile la data de 03.11.2019**
   http://users.utcluj.ro/~baruch/ro/pages/cursuri/structura-sistemelor-de-calcul.php
7. **Am luat de la link-ul acesta informații legate de aplicația Android numită Serial Bluetooth Terminal, link care era disponibil la data de 21.12.2019**
   https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=ro
8. **Am luat de pe site-ul domnului profesor Baruch pozele cu numerele 4,5 și 6 și informații legate de transmisia serială UART, link care era disponibil la data de 10.01.2020.**
   http://users.utcluj.ro/~baruch/ssc/labor/Testare-Depanare.pdf

7.2 Link-uri poze

1. **Am luat de la link-ul acesta poza generală cu placa FPGA Nexys 4 disponibilă la data de 15.11.2019**
   https://store.digilentinc.com/nexys-4-ddr-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/
2. **Am luat de la link-ul acesta poza generală cu placa FPGA Basys 3 disponibilă la data de 15.11.2019**
   https://www.google.ro/search?q=basys+3&client=opera&hs=RBJ&sxsrf=ACYBGNSJdp8SSe0cjq1XLzBwVjBsaSYlPQ:1572787794635&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjqprmOk87lAhVIQkEAHeUfAtQQ_AUIESgB&biw=1496&bih=754#imgrc=YS3cS8Tbv1rH5M
3. **Am luat de la link-ul acesta poza cu modulul Bluetooth PMOD BT2 disponibilă la data de 18.11.2019**
   https://store.digilentinc.com/pmod-bt2-bluetooth-interface/
4. **Am luat de la link-ul acesta poza cu modulul de dezvoltare Vivado disponibilă la data de 11.11.2019**
   https://www.google.ro/search?q=vivado&client=opera&hs=JEJ&sxsrf=ACYBGNSbRk1PzT4ds9rDEqHUzur0HNFJQg:1572787972065&source=lnms&tbm=isch&sa=X&ved=0ahUKEwi3xYbjk87lAhXBy6QKHYUiBUgQ_AUIESgB&biw=1496&bih=754&dpr=1.25#imgrc=WFCfYqzMpcXPOM
5. **Am luat de la link-ul acesta poza cu schema generală a circuitului de adunare cu anticiparea transportului disponibilă la data de 11.11.2019**
   http://users.utcluj.ro/~baruch/media/ssc/curs/SSC-Adunare.pdf
6. **Am luat de la link-ul acesta poza cu schema generală a circuitului de scădere disponibilă la data de 11.11.2019**
   https://www.elprocus.com/full-subtractor-circuit-using-logic-gates/

7. **Am luat de la link-ul acesta poza cu o schema de circuit ce efectuează adunarea și scăderea disponibilă la data de 20.11.2019 https://www.electronicshub.org/binary-adder-and-subtractor/**
8. **Am luat de la link-ul acesta poza cu schema generală a circuitului de înmulțire prin metoda Booth disponibilă la data de 13.11.2019 http://users.utcluj.ro/~baruch/media/ssc/curs/SSC-Inmultire-1.pdf**
9. **Am luat de la link-ul acesta poza cu schema generală a circuitului de împărțire prin metoda refacerii restului parțial disponibilă la data de 13.11.2019 http://users.utcluj.ro/~baruch/media/ssc/curs/SSC-Impartire.pdf**
10. **Am luat de la link-ul acesta poza cu schema generală a circuitului de deplasare stânga-dreapta disponibilă la data de 13.11.2019 https://www.geeksforgeeks.org/digital-logic-shift-registers/**
11. **Am luat de la link-ul acesta poza cu diferite imagini din aplicatia Serial Bluetooth Terminal furnizate de producator disponibilă la data de 21.12.2019 https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=ro**

7.3 Anexă

1.1 Cod sumator cu anticiparea transportului pe 32 biți:

1.1.1 Sursă sumator pe 32 biți

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity sumator32biti is
 Port ( x: in std_logic_vector (31 downto 0);
        y: in std_logic_vector (31 downto 0);
        Tin: in std_logic;
        s: out std_logic_Vector(31 downto 0);
        Tout: out std_logic );


end sumator32biti;
```

```
architecture Behavioral of sumator32biti is

signal P0, G0, P1, G1, P2, G2, P3, G3, P4, G4, P5, G5, P6, G6, P7, G7, P8, G8, P9, G9, P10, G10,
P11, G11, P12, G12, P13, G13, P14, G14, P15,G15: std_logic;
signal T1, T2,T3, T4,T5,T6,T7,T8,T9, T10, T11, T12, T13, T14,T15: std_logic;

component sumator8biti is
  Port (x: in std_logic_vector (7 downto 0);
        y: in std_logic_vector (7 downto 0);
        Tin: in std_logic;
        s: out std_logic_Vector(7 downto 0);
        G,P: out std_logic);
end component;

component sumator2biti is
  Port (x,y:in std_logic_vector(1 downto 0);
        tin:in std_logic;
        s:out std_logic_vector(1 downto 0);
        G,P:out std_logic);
end component;
begin

sum1: sumator2biti port map(x(1 downto 0), y(1 downto 0), Tin, s(1 downto 0), G0, P0);
sum2: sumator2biti port map(x(3 downto 2), y(3 downto 2), T1, s(3 downto 2), G1, P1);
sum3: sumator2biti port map(x(5 downto 4), y(5 downto 4), T2, s(5 downto 4), G2, P2);
sum4: sumator2biti port map(x(7 downto 6), y(7 downto 6), T3, s(7 downto 6), G3, P3);

sum5: sumator2biti port map(x(9 downto 8), y(9 downto 8), T4, s(9 downto 8), G4, P4);
sum6: sumator2biti port map(x(11 downto 10), y(11 downto 10), T5, s(11 downto 10), G5, P5);
```

```
sum7: sumator2biti port map(x(13 downto 12), y(13 downto 12), T6, s(13 downto 12), G6, P6);
sum8: sumator2biti port map(x(15 downto 14), y(15 downto 14), T7, s(15 downto 14), G7, P7);

sum9: sumator2biti port map(x(17 downto 16), y(17 downto 16), T8, s(17 downto 16), G8, P8);
sum10: sumator2biti port map(x(19 downto 18), y(19 downto 18), T9, s(19 downto 18), G9, P9);
sum11: sumator2biti port map(x(21 downto 20), y(21 downto 20), T10, s(21 downto 20), G10, P10);
sum12: sumator2biti port map(x(23 downto 22), y(23 downto 22), T11, s(23 downto 22), G11, P11);

sum13: sumator2biti port map(x(25 downto 24), y(25 downto 24), T12, s(25 downto 24), G12, P12);
sum14: sumator2biti port map(x(27 downto 26), y(27 downto 26), T13, s(27 downto 26), G13, P13);
sum15: sumator2biti port map(x(29 downto 28), y(29 downto 28), T14, s(29 downto 28), G14, P14);
sum16: sumator2biti port map(x(31 downto 30), y(31 downto 30), T15, s(31 downto 30), G15, P15);

T1<= G0 OR  (P0 and Tin);
T2<= G1 OR (P1 and T1);
T3<= G2 OR (P2 and T2);
T4<= G3 OR (P3 and T3);
T5<= G4 OR (P4 and T4);
T6<= G5 OR (P5 and T5);
T7<= G6 OR (P6 and T6);
T8<= G7 OR (P7 and T7);
T9<= G8 OR (P8 and T8);
T10<= G9 OR (P9 and T9);
T11<= G10 OR (P10 and T10);
```

```
T12<= G11 OR (P11 and T11);
T13<= G12 OR (P12 and T12);
T14<= G13 OR (P13 and T13);
T15<= G14 OR (P14 and T14);

Tout <= T15;

end Behavioral;
```

1.2 Cod scăzător pe 32 biți:

1.2.1 Sursă scăzător pe 32 biți

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity scadere32biti is
    Port(x: in std_logic_vector(31 downto 0);
        y: in std_logic_vector(31 downto 0);
        diferenta: out std_logic_vector(31 downto 0));
end scadere32biti;

architecture Behavioral of scadere32biti is

component sumator32biti is
   Port ( x: in std_logic_vector (31 downto 0);
        y: in std_logic_vector (31 downto 0);
        Tin: in std_logic;
        s: out std_logic_Vector(31 downto 0);
        Tout: out std_logic );
```

```
end component;

signal Tin: std_logic:='0';
signal Tout: std_logic;
signal suma: std_logic_vector(31 downto 0);
signal y_complement: std_logic_vector(31 downto 0);

begin
    y_complement<=not(y) + "00000000000000000000000000000001";
    sumator: sumator32biti port map(x,y_complement,Tin,suma,Tout);
    diferenta<=suma;
end Behavioral;
```

1.3 Cod circuit deplasare pe 32 biți:

1.3.1 Sursă circuit deplasare

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity deplasariStangaDreapta32biti is
    Port(dataIn: in std_logic_vector(31 downto 0);
        selDeplasare: in std_logic;
        selPozitiiDeplasare: in std_logic_vector(1 downto 0);
        dataOut: out std_logic_vector(31 downto 0));
end deplasariStangaDreapta32biti;

architecture Behavioral of deplasariStangaDreapta32biti is
```

```
begin

    process(selDeplasare)
    begin
        case selDeplasare is
        when '0' => if(selPozitiiDeplasare = "00") then
                        dataOut<= dataIn;
                    elsif (selPozitiiDeplasare="01") then
                        dataOut<= dataIn(30 downto 0) & '0';
                    elsif (selPozitiiDeplasare="10") then
                        dataOut<=dataIn(29 downto 0) & "00";
                    elsif (selPozitiiDeplasare="11") then
                        dataOut<=dataIn(28 downto 0) & "000";
                    end if;
        when '1' => if(selPozitiiDeplasare = "00") then
                        dataOut<= dataIn;
                    elsif (selPozitiiDeplasare="01") then
                        dataOut<= '0' & dataIn(31 downto 1);
                    elsif (selPozitiiDeplasare="10") then
                        dataOut<= "00" & dataIn(31 downto 2);
                    elsif (selPozitiiDeplasare="11") then
                        dataOut<= "000" & dataIn(31 downto 3);
                    end if;
         when others => dataOut<= dataIn;
       end case;
    end process;

end Behavioral;
```

1.4 Cod ALU pe 32 biți:

1.4.1 Cod sursă ALU

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity alu_32 is
    generic (n : natural := 32);
    port(x: in std_logic_vector(31 downto 0);
        y: in std_logic_vector(31 downto 0);
        selectie: in std_logic_vector(2 downto 0);
        rezultat: out std_logic_vector(31 downto 0));
end alu_32;

architecture Behavioral of alu_32 is

signal clk: std_logic:='0';
signal rst: std_logic:='0';
signal start: std_logic:='0';
signal a: std_logic_vector(n-1 downto 0):=x"00000000";
signal q: std_logic_vector(n-1 downto 0):=x"00000000";
signal term: std_logic:='0';
signal Tin: std_logic:='0';
signal Tout: std_logic;
signal selDeplasare: std_logic;
signal selPozitiiDeplasare: std_logic_vector(1 downto 0);
signal rezultatAdunare: std_logic_vector(31 downto 0):=x"00000000";
signal rezultatScadere: std_logic_vector(31 downto 0):=x"00000000";
```

```
signal rezultatDeplasare: std_logic_vector(31 downto 0):=x"00000000";
signal rezultatInmultireA: std_logic_vector(n-1 downto 0):=x"00000000";
signal rezultatInmultireQ: std_logic_vector(n-1 downto 0):=x"00000000";

component sumator32biti is
   Port ( x: in std_logic_vector (31 downto 0);
        y: in std_logic_vector (31 downto 0);
        Tin: in std_logic;
        s: out std_logic_Vector(31 downto 0);
        Tout: out std_logic );
end component;

component scadere32biti is
    Port(x: in std_logic_vector(31 downto 0);
        y: in std_logic_vector(31 downto 0);
        diferenta: out std_logic_vector(31 downto 0));
end component;

component deplasariStangaDreapta32biti is
    Port(dataIn: in std_logic_vector(31 downto 0);
        selDeplasare: in std_logic;
        selPozitiiDeplasare: in std_logic_vector(1 downto 0);
        dataOut: out std_logic_vector(31 downto 0));
end component;

component mult_booth is
    Port (clk : in std_logic;
          rst : in std_logic;
```

```
        start : in std_logic;
        x : in std_logic_vector(n-1 downto 0);
        y : in std_logic_vector(n-1 downto 0);
        a : out std_logic_vector(n-1 downto 0);
        q : out std_logic_vector(n-1 downto 0);
        term : out std_logic);
end component;

begin

    adunare: sumator32biti port map(x, y, Tin, rezultatAdunare, Tout);
    scadere: scadere32biti port map(x,y, rezultatScadere);
    deplasare: deplasariStangaDreapta32biti port map(x,selDeplasare,selPozitiiDeplasare,rezultatDeplasare);
    inmultire: mult_booth port map(clk,rst,start,x,y,rezultatInmultireA,rezultatInmultireQ,term);

    process(selectie,rezultatAdunare, rezultatScadere, rezultatDeplasare, rezultatInmultireA,rezultatInmultireQ)

    variable rez: std_logic_vector(31 downto 0);

    begin
        case selectie is
            when "000" => Tin <= '0';
                        rez := rezultatAdunare;

            when "001" => Tin <= '1';
                        rezultat <= rezultatAdunare;
```

```
            when "010" => rezultat <= rezultatScadere;

            when "011" => rezultat <= rezultatDeplasare;

            when "100" => a<=rezultatInmultireA;
                          q<=rezultatInmultireQ;

            when others => rezultat <= "00000000000000000000000000000000";
        end case;

        --rezultat<=rez;
    end process;
end Behavioral;
```

1.5 Cod înmulțitor pe 32 biți:

1.5.1 Cod sursă main înmulțitor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mult_booth is
    generic (n : natural := 32);
    Port (clk : in std_logic;
          rst : in std_logic;
          start : in std_logic;
          x : in std_logic_vector(n-1 downto 0);
```

```
        y : in std_logic_vector(n-1 downto 0);
        a : out std_logic_vector(n-1 downto 0);
        q : out std_logic_vector(n-1 downto 0);
        term : out std_logic);
end mult_booth;

architecture Behavioral of mult_booth is
signal loadB : std_logic;
signal outB : std_logic_vector(n-1 downto 0);
signal subB : std_logic;
signal newB : std_logic_vector(n-1 downto 0);

signal inA : std_logic_vector(n-1 downto 0);
signal outA : std_logic_vector(n-1 downto 0);
signal loadA : std_logic;
signal sarAQ : std_logic;

signal loadQ : std_logic;
signal outQ : std_logic_vector(n-1 downto 0);
signal qm1 : std_logic;
signal q0qm1 : std_logic_vector(1 downto 0);

signal tout : std_logic;
signal ovf : std_logic;

signal rstA : std_logic;
signal rstqm1 : std_logic;
```

```
begin
   fdn_B: entity WORK.fdn generic map (n => 32) port map (clk => clk,
                                                          rst => rst,
                                                          ce => loadB,
                                                          d => x,
                                                          q => outB);

   xorgates: for i in n-1 downto 0 generate
      newB(i) <= subB xor outB(i);
   end generate;

   srrn_A: entity WORK.srrn generic map (n => 32) port map (clk => clk,
                                                            rst => rstA,
                                                            ce => sarAQ,
                                                            load => loadA,
                                                            sri => outA(n-1),
                                                            d => inA,
                                                            q => outA);

   srrn_q: entity WORK.srrn generic map (n => 32) port map (clk => clk,
                                                            rst => rst,
                                                            ce => sarAQ,
                                                            load => loadQ,
                                                            sri => outA(0),
                                                            d => y,
                                                            q => outQ);

   fd_qm1: entity WORK.fd port map (clk => clk,
```

```
                    rst => rstqm1,
                    ce => sarAQ,
                    d => outQ(0),
                    q => qm1);

    addn: entity WORK.addn generic map (n => 32) port map (x => outA,
                                                           y => newB,
                                                           tin => subB,
                                                           s => inA,
                                                           tout => tout,
                                                           ovf => ovf);
    q0qm1 <= outQ(0) & qm1;
    com_unit: entity WORK.comand_unit generic map (n => 32) port map (clk => clk,
                                                                      rst => rst,
                                                                      start => start,
                                                                      q0qm1 => q0qm1,
                                                                      term => term,
                                                                      loadB => loadB,
                                                                      subB => subB,
                                                                      rstA => rstA,
                                                                      loadA => loadA,
                                                                      sarAQ => sarAQ,
                                                                      loadQ => loadQ,
                                                                      rstqm1 => rstqm1);
    a <= outa;
    q <= outq;
end Behavioral;
```

1.5.2 Cod sursă unitatea de comandă

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comand_unit is
    generic (n : natural);
    Port (clk : in std_logic;
          rst : in std_logic;
          start : in std_logic;
          q0qm1 : in std_logic_vector(1 downto 0);
          term : out std_logic;
          loadB : out std_logic;
          subB : out std_logic;
          rstA : out std_logic;
          loadA : out std_logic;
          sarAQ : out std_logic;
          loadQ : out std_logic;
          rstQm1 : out std_logic);
end comand_unit;

architecture Behavioral of comand_unit is
signal c : natural;

type states is (idle, init, test, sub, add, shift, stop);
signal state : states;
begin
    process(clk)
```

```
begin
   if rst = '1' then
      state <= idle;
   elsif rising_edge(clk) then
      case state is
         when idle => if start = '1' then
                        state <= init;
                      end if;
         when init => state <= test;
                      c <= n;
         when test =>if c = 0 then
                        state <= stop;
                     elsif c > 0 then
                        if q0qm1 = "10" then
                           state <= sub;
                        elsif q0qm1 = "01" then
                           state <= add;
                        elsif q0qm1 = "00" or q0qm1 = "11" then
                           state <= shift;
                        end if;
                     end if;
         when sub => state <= shift;
         when add => state <= shift;
         when shift => state <= test;
                       c <= c - 1;
         when stop => null;
      end case;
   end if;
```

```
    end process;


    --outputs
    loadB <= '1' when state = init else '0';
    term <= '1' when state = stop else '0';
    subB <= '1' when state = sub else '0';
    rstA <= '1' when state = init else '0';
    loadA <= '1' when state = sub or state = add else '0';
    sarAQ <= '1' when state = shift else '0';
    loadQ <= '1' when state = init else '0';
    rstQm1 <= '1' when state = init else '0';

end Behavioral;
```

1.5.3 Cod sursă fd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fd is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           ce : in STD_LOGIC;
           d : in STD_LOGIC;
           q : out STD_LOGIC);
```

```
end fd;

architecture Behavioral of fd is

begin
    process(clk)
    begin
        if rising_edge(clk) then
            if rst = '1' then
                q <= '0';
            elsif ce = '1' then
                q <= d;
            end if;
        end if;
    end process;
end Behavioral;
```

1.5.4 Cod sursă fdn

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fdn is
    generic (n : natural);
    Port (clk : in std_logic;
          rst : in std_logic;
          ce : in std_logic;
          d : in std_logic_vector(n-1 downto 0);
```

```
        q : out std_logic_vector(n-1 downto 0));
end fdn;

architecture Behavioral of fdn is

begin
    process(clk)
    begin
        if rising_edge(clk) then
            if rst = '1' then
                for i in n-1 downto 0 loop
                    q(i) <= '0';
                end loop;
            elsif ce = '1' then
                q <= d;
            end if;
        end if;
    end process;
end Behavioral;
```

1.5.5 Cod sursă srrn

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity srrn is
    generic (n : natural);
    Port (clk : in std_logic;
```

```
        rst : in std_logic;
        ce : in std_logic;
        load : in std_logic;
        sri : in std_logic;
        d : in std_logic_vector(n-1 downto 0);
        q : out std_logic_vector(n-1 downto 0));
end srrn;

architecture Behavioral of srrn is
signal temp : std_logic_vector(n-1 downto 0);
begin
   process(clk)
   begin
      if rising_edge(clk) then
         if rst = '1' then
            for i in n-1 downto 0 loop
               temp(i) <= '0';
            end loop;
         elsif load = '1' then
            temp <= d;
         elsif ce = '1' then
            temp(n-2 downto 0) <= temp(n-1 downto 1);
            temp(n-1) <= sri;
         end if;
      end if;
   end process;
   q <= temp;
end Behavioral;
```

1.5.6 Cod sursă addn

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity addn is
    generic (n : natural);
    Port (x : in std_logic_vector(n-1 downto 0);
          y : in std_logic_vector(n-1 downto 0);
          tin : in std_logic;
          s : out std_logic_vector(n-1 downto 0);
          tout : out std_logic;
          ovf : out std_logic);--se seteaza daca carry out de la ultimul e diferit de carry out de la penultimul
end addn;

architecture Behavioral of addn is
signal total_sum : std_logic_vector(n downto 0);
signal zeros : std_logic_vector(n-1 downto 0):= (others => '0');
begin

    total_sum <= x + y + (zeros & tin);
    s <= total_sum(n-1 downto 0);
    tout <= total_sum(n);
    ovf <= total_sum(n) xor (x(n-1) or y(n-1));
```

```
end Behavioral;
```

1.6 Cod sursă main împărțitor

1.6.1 Cod sursă main împărțitor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity REFACERE_REST is

generic (n : integer := 8);
Port ( clk : in STD_LOGIC;
rst : in STD_LOGIC;
start : in STD_LOGIC;
x : in STD_LOGIC_VECTOR (n-1 downto 0);
y : in STD_LOGIC_VECTOR (n-1 downto 0);
rest : out STD_LOGIC_VECTOR (n-1 downto 0);
cat : out STD_LOGIC_VECTOR (n-2 downto 0);
term : out STD_LOGIC);

end REFACERE_REST;

architecture Behavioral of REFACERE_REST is
```

```
signal registrulA_in: STD_LOGIC_VECTOR(n downto 0):=(others=>'0');
signal registrulA: STD_LOGIC_VECTOR(n downto 0):=(others=>'0');
signal registrulB: STD_LOGIC_VECTOR(n-1 downto 0):=(others=>'0');
signal registrulQ: STD_LOGIC_VECTOR(n-2 downto 0):=(others=>'0');
signal inputNow: STD_LOGIC_VECTOR(n-1 downto 0):=(others=>'0');
signal loadA: STD_LOGIC:='0';
signal loadB: STD_LOGIC:='0';
signal loadQ: STD_LOGIC:='0';
signal rezultatSuma: STD_LOGIC_VECTOR(n-1 downto 0):=(others=>'0');
signal rstA,minusB,shlAQ : STD_LOGIC:='0';
signal intrareSum: STD_LOGIC_VECTOR(n-1 downto 0):=(others=>'0');
signal tout, shif: STD_LOGIC:='0';
signal ovf: STD_LOGIC:='0';
signal q0: STD_LOGIC:='0';
signal valQ0:std_logic:='0';
signal loadQ0:std_logic:='0';
signal regQfin:STD_LOGIC_VECTOR(n-3 downto 0):=(others=>'0');
signal rezultatQ: STD_LOGIC_VECTOR(n-2 downto 0):=(others=>'0');
signal selectInitValue:STD_LOGIC:='0';

begin
    rest <= registrulA(n-1 downto 0);
    cat <= registrulQ;
    registrulA_in <= tout & rezultatSuma;
    shif <= not(registrulA(n)) when minusB='1' else '0';

unitateControl:entity work.CONTROL generic map(n => n)
    port map(
```

```
    clk => clk,
    rst => rst,
    start => start,
    an => registrulA(n-1),
    selectInitValue => selectInitValue,
    term => term,
    rstA => rstA,
    loadA => loadA,
    loadB => loadB,
    loadQ => loadQ,
    loadQ0 => loadQ0,
    valQ0 => valQ0,
    shlAQ => shlAQ,
    minusB => minusB);

regA: entity work.SLEFTNA generic map (n => n + 1)
port map (
    clk => clk,
    shif => registrulQ(n-2),
    rst => rstA,
    ce => shlAQ,
    q => registrulA,
    d => registrulA_in,
    load => loadA);

regQ: entity work.SLEFTNQ generic map (n => n-1)
port map (
    clk => clk,
```

```
    shif => shif,
    rst => rst,
    ce => shlAQ,
    loadQ0=>loadQ0,
    valQ0=>valQ0,
    q => registrulQ,
    d => x(n-2 downto 0),
    load => loadQ);

xorG: entity work.sauExclusiv generic map (n => n)
port map (
    minusB => minusB,
    x => registrulB,
    q => intrareSum);

sumator: entity work.SUMATOR generic map(n => n)
port map (
    x => registrulA(n-1 downto 0),
    y => intrareSum,
    tin => minusB,
    tout => tout,
    ovf => ovf,
    sum => rezultatSuma);

regB:entity work.FDN generic map(n => n)
port map (
    clk => clk,
    d => y,
```

```
    q => registrulB,
    ce => loadB,
    rst => rst);
end Behavioral;
```

1.6.2 Cod sursă unitate de control

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CONTROL is
generic(n:integer);
Port ( clk : in STD_LOGIC;
rst : in STD_LOGIC;
start : in STD_LOGIC;
an: in STD_LOGIC;
selectInitValue:out std_logic;
term : out STD_LOGIC;
rstA : out STD_LOGIC;
loadA : out STD_LOGIC;
loadB : out STD_LOGIC;
loadQ : out STD_LOGIC;
loadQ0: out std_logic;
valQ0: out std_logic;
shlAQ : out STD_LOGIC;
minusB : out STD_LOGIC );
end CONTROL;
architecture Behavioral of CONTROL is
type state_type is (empty, init, choose, substract, add, subC,shift, stop, setQ0 );
signal state: state_type;
```

```
signal c:integer;
signal termAux:std_logic:='0';
begin
chooseState: process(clk)
begin
if rising_edge(clk) then
if (rst='1') then
termAux<='0';
state<= empty;
else
case state is
when empty =>
if (start = '1') then
state <= init;
end if;
c <= n-2;
when init =>
state<=shift;
when shift =>
state<=substract;
when substract =>
state <= choose;
when choose =>
if(an='1') then
state<=add;
else
state<=setQ0;
end if;
```

```
when add=>
valQ0<='0';
state<=subC;
when setQ0 =>
valQ0<='1';
state <= subC;
when subC=>
c<=c-1;
if(c=0) then
state<=stop;
else
state<=shift;
end if;
when stop =>
termAux<='1';
when others =>
termAux<='1';
end case;
end if;
end if;
end process chooseState;
term <= termAux;
loadA <= '1' when ( (state=add)or(state=init) or (state=substract) ) else '0';
loadB <= '1' when state=init else '0';
loadQ <= '1' when state=init else '0';
loadQ0<= '1' when state=setQ0 else '0';
minusB <= '1' when state=substract else '0';
shlAQ <= '1' when state=shift else '0';
```

```
rstA <= '1' when ( (state=init) or (state=empty) ) else '0';
selectInitValue<='1' when state=shift else '0';
end Behavioral;
```

1.6.3 Cod sursă fn

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fdn is
    generic (n : integer);
      Port ( clk : in STD_LOGIC;
        ce : in STD_LOGIC;
        rst : in STD_LOGIC;
        d : in STD_LOGIC_VECTOR (n-1 downto 0);
        q : out STD_LOGIC_VECTOR (n-1 downto 0));
end fdn;

architecture Behavioral of fdn is
signal aux: std_logic_vector (n-1 downto 0);
begin
process(clk)
    begin
        if(clk'event and clk='1') then
            if(rst='1') then
                aux<=(others=>'0');
            elsif(ce='1') then
                aux<=d;
end if;
end if;
```

```
end process;
q<=aux;

end Behavioral;
```

1.6.4 Cod sursă shift left A

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity SLEFTNA is
generic (n : integer);
Port (
      clk : in STD_LOGIC;
      shif : in STD_LOGIC;
      rst : in STD_LOGIC;
      ce : in STD_LOGIC;
      q : out STD_LOGIC_VECTOR (n-1 downto 0);
      d : in STD_LOGIC_VECTOR (n-1 downto 0);
      load : in STD_LOGIC);
end SLEFTNA;
architecture Behavioral of SLEFTNA is
signal aux: std_logic_vector (n-1 downto 0);
begin
process(clk)
begin
   if clk='1' and clk'event then
      if rst='1' then
         aux <= (others=>'0');
      elsif load='1' then
         aux <= d ;
```

```
        elsif ce='1' then
            aux <= aux(n-2 downto 0) & shif;
end if;
end if;
end process;
q <= aux;
end Behavioral;
```

1.6.5 Cod sursă shift left Q

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SLEFTNQ is
generic (n : integer);
Port (
clk : in STD_LOGIC;
shif : in STD_LOGIC;
rst : in STD_LOGIC;
ce : in STD_LOGIC;
loadQ0:in std_logic;
valQ0:in std_logic;
q : out STD_LOGIC_VECTOR (n-1 downto 0);
d : in STD_LOGIC_VECTOR (n-1 downto 0);
load : in STD_LOGIC);
end SLEFTNQ;
architecture Behavioral of SLEFTNQ is
signal aux: std_logic_vector (n-1 downto 0);
begin
```

```
process(clk)
begin
if clk='1' and clk'event then
if rst='1' then
aux <= (others=>'0');
elsif load='1' then
aux <= d ;
elsif ce='1' then
aux <= aux(n-2 downto 0) & shif;
elsif loadQ0 ='1' then
aux <= aux(n-1 downto 1) & '1';
end if;
end if;
end process;
q <= aux;
end Behavioral;
```

1.7 Cod sursă UART

1.7.1 Cod sursă main UART

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity rx_uart is
  Port (clk : in STD_LOGIC;
        rx : in std_logic;
        output: out std_logic_vector(7 downto 0));
        --an: out STD_LOGIC_VECTOR(3 downto 0);
        --cat: out STD_LOGIC_VECTOR(6 downto 0)
```

```
end rx_uart;

architecture Behavioral of rx_uart is

component SSD is
    Port ( clk: in STD_LOGIC;
           digits: in STD_LOGIC_VECTOR(15 downto 0);
           an: out STD_LOGIC_VECTOR(3 downto 0);
           cat: out STD_LOGIC_VECTOR(6 downto 0));
end component;

signal counterUart: integer := 0;
signal baudEn : std_logic := '0';
signal rx_rdy : std_logic;
signal rx_data: std_logic_vector(7 downto 0):="00000000";
signal rezultatDisplay: std_logic_vector(15 downto 0):="0000000000000000";

begin

baud_rate_generator:process(clk)
    begin
        if rising_edge(clk) then
            if counterUart < 651 then
                counterUart <= counterUart + 1;
                baudEn <= '0';
            elsif counterUart = 651 then
                counterUart <= 0;
                baudEn <= '1';
```

```
            end if;
        end if;
    end process;

fsm: entity work.rx_fsm port map(rx_rdy, rx_data, baudEn, '0', rx, clk);
--display: SSD port map(clk,rezultatDisplay,an,cat);
output<=rx_data;

--rezultatDisplay<="00000000" & rx_data;

end Behavioral;
```

1.7.2 Cod sursă RX_FSM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RX_FSM is
  Port ( RX_RDY: out STD_LOGIC;
         RX_DATA: out STD_LOGIC_VECTOR(7 downto 0);
         BAUD_EN: in STD_LOGIC;
         RST: in STD_LOGIC;
         RX: in STD_LOGIC;
         clk: in STD_LOGIC);
```

```
end RX_FSM;

architecture Behavioral of RX_FSM is
type state_type is(s1,s2,s3,s4,s5);
signal state:state_type;
signal BAUD_CNT: STD_LOGIC_VECTOR(3 downto 0);
signal BIT_CNT: STD_LOGIC_VECTOR(2 downto 0);

begin
process(clk)
begin
   if rst = '1' then
      state<=s1;
      BAUD_CNT<="0000";
   else
      if rising_edge(clk) then
         if BAUD_EN = '1' then
            case state is
               when s1 => if RX = '0' then
                          BAUD_CNT<="0000";
                          BIT_CNT<="000";
                          state<=s2;
                         end if;
               when s2 => BAUD_CNT<=BAUD_CNT+1;
                          if BAUD_CNT = "0111" then
                            if RX = '1' then
                               state<=s1;
                               BAUD_CNT<="0000";
```

```
            else
              BAUD_CNT<="0000";
              BIT_CNT<="000";
              state<=s3;
            end if;
          end if;
when s3 =>  BAUD_CNT<=BAUD_CNT+1;
          if BAUD_CNT = "1111" then
            RX_DATA(conv_integer(BIT_CNT))<=RX;
            BIT_CNT<=BIT_CNT+1;
            if BIT_CNT = "111" then
                state<=s4;
                BIT_CNT<="000";
                BAUD_CNT<="0000";
            else
                state<=s3;
                BAUD_CNT<="0000";
            end if;
          end if;
when s4 => BAUD_CNT<=BAUD_CNT+1;
          if BAUD_CNT = "1111" then
              state<=s5;
              BAUD_CNT<="0000";
          end if;
when s5 => BAUD_CNT<=BAUD_CNT+1;
          if BAUD_CNT = "0111" then
              BAUD_CNT<="0000";
              state<=s1;
```

```
                    end if;
                end case;
            end if;
        end if;
    end if;
end process;

process(clk)
begin
    if rising_edge(clk) then
        if BAUD_EN = '1' then
            case state is
                when s1 => RX_RDY<='0';
                when s2 => RX_RDY<='0';

                when s3 => RX_RDY<='0';

                when s4 => RX_RDY<='0';

                when s5 => RX_RDY<='1';

            end case;
        end if;
     end if;
end process;

end Behavioral;
```

1.8 Cod sursă aplicație JAVA

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;

public class HC05 {

    boolean scanFinished = false;
    RemoteDevice hc05device;
    String hc05Url;

    public static void main(String[] args) {
        try {
            new HC05().go();
        } catch (Exception ex) {
            Logger.getLogger(HC05.class.getName()).log(Level.SEVERE, null, ex);
        }
```

```
    }

    private void go() throws Exception {
        //scan for all devices:
        scanFinished = false;
        LocalDevice.getLocalDevice().getDiscoveryAgent().startInquiry(DiscoveryAgent.GIAC,
new DiscoveryListener() {
            @Override
            public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
                try {
                    String name = btDevice.getFriendlyName(false);
                    System.out.format("%s (%s)\n", name, btDevice.getBluetoothAddress());
                    System.out.println(name);
                    if (name.matches("HC.*")) {
                        hc05device = btDevice;
                        System.out.println("got it!");
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void inquiryCompleted(int discType) {
                scanFinished = true;
            }

            @Override
            public void serviceSearchCompleted(int transID, int respCode) {
```

```
    }

    @Override
    public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
    }
});
while (!scanFinished) {
    //this is easier to understand (for me) as the thread stuff examples from bluecove
    Thread.sleep(100);
}

//search for services:
UUID uuid = new UUID(0x1101); //scan for btspp://... services (as HC-05 offers it)
UUID[] searchUuidSet = new UUID[]{uuid};
int[] attrIDs = new int[]{
    0x0100 // service name
};
scanFinished = false;
LocalDevice.getLocalDevice().getDiscoveryAgent().searchServices(attrIDs, searchUuidSet,
        hc05device, new DiscoveryListener() {
            @Override
            public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
            }

            @Override
            public void inquiryCompleted(int discType) {
            }
```

```
            @Override
            public void serviceSearchCompleted(int transID, int respCode) {
               scanFinished = true;
            }

            @Override
            public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
               for (int i = 0; i < servRecord.length; i++) {
                  hc05Url = servRecord[i].getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
                  if (hc05Url != null) {
                     break; //take the first one
                  }
               }
            }
         });

      while (!scanFinished) {
         Thread.sleep(100);
      }

      System.out.println(hc05device.getBluetoothAddress());
      System.out.println(hc05Url);

      //if you know your hc05Url this is all you need:
      StreamConnection streamConnection = (StreamConnection) Connector.open(hc05Url);
      OutputStream os = streamConnection.openOutputStream();
      InputStream is = streamConnection.openInputStream();
```

```
        //os.write("1".getBytes()); //just send '1' to the device
        os.close();
        is.close();
        streamConnection.close();
    }
}
```