Documentatie ALU

Contributie: Paun David Andrei

ALU (Arithmetic-logic unit) reprezinta un dispozitiv hardware capabil sa execute operatii aritmetice (adunare, scadere, inmultire, impartire, ridicare la putere, etc.) si logice (XOR, AND, OR, etc) in functie de anumite intrari. De regula, acest dispozitiv primeste 2 operanzi, un cod al operatiei si returneaza rezultatul operatiei corespunzatoare.

Obiectivele lucrarii:

Realizarea unui dispozitiv secvential binar capabil sa execute operatii de adunare, scadere, inmultire si impartire pe 8 biti, folosind diverse componente logice (porti logice, bistabile, multiplexoare, etc.)

Faza de design:

Acest dispozitiv este construit sa primeasca la intrare doua numere binare pe 8 biti, o codificare a operatiei (00 – adunare, 01 – scadere, 10 – inmultire, 11 – impartire) si returneaza la iesire rezultatul pe 16 biti.

Pentru implementarea acestui dispozitiv intr-un mod cat mai optim, am implementat toate operatiile folosind aceiasi registri in care am stocat valorile intermediare. Pentru operatia de inmultire am folosit algoritmul Booth radix-2, iar pentru impartire am folosit SRT radix-2. Am ales aceasta combinatie de algoritmi deoarece ambii folosesc registri care fac shiftari cu o singura pozitie, acest lucru fiind usor de implementat cu bistabile si multiplexoare in interiorul registrilor. De asemenea, ambii algoritmi se bazeaza pe codificarea booth radix-2, adica un grup de cifre poate fi codificat in elemente din multimea {-1, 0, 1}.

Componente ALU:

* **Data path**, format din toti registrii, numaratoarele, multiplexoarele care contribuie la calcule
* **Control Unit**, care produce semnalele ce dirijeaza ordinea in care se efectueaza pasii operatiilor

Componente Data Path:

* Registrul A
  + Pe 9 biti
  + Are facilitati de incarcare paralela, shiftare la dreapta si shiftare la stanga cu o pozitie
  + Are rol de acumulator, stocheaza rezultate intermediare
  + Semnale de intrare:
    - Load: incarca registrul
    - A7\_mem: folosit DOAR la inmultire, deoarece facem shiftare aritmetica la dreapta, in acest caz se memoreaza in A[7] valoarea de dinainte. Cand facem inmultire registrul A se comporta intotdeauna ca un registru pe 8 biti.
    - Left shift A: shifteaza registrul A la stanga cu o pozitie
    - Left shift entry wire: semnalul care intra in registrul A in cadrul shiftarii la stanga, acest semnal primeste cel mai semnificativ bit al registrului Q
    - Right shift A: shifteaza registrul A la dreapta cu o pozitie
    - Bus de intrare la load
  + Semnale de iesire:
    - Q, iesirea bistabilelor
* Registrul Q
  + Pe 9 biti
  + Are facilitati de incarcare paralela, shiftare la dreapta si shiftare la stanga cu o pozitie.
  + Stocheaza primul operand, suma (in cazul adunarii), diferenta (in cazul scaderii), cei mai putini semnificativi 8 biti ai produsului (in cazul inmultirii) sau catul (in cazul impartirii)
  + Atentie: Registrul Q este mai special, deoarece bistabilul q[0] este folosit doar in cazul operatiei de inmultire cu booth (se comporta ca si q[-1] ), iar q[1] este de fapt cel mai putin semnificativ bit. Deci tot timpul vom verifica q[1:0].
  + Semnale de intrare:
    - Load: incarca registrul, q[8:1] va primi un bus pe 8 biti, iar q[0] intotdeauna 0 la incarcare
    - Left shift Q: Folosit DOAR la impartire, shifteaza registrul Q la stanga cu o pozitie .Se shifteaza doar cei mai semnificativi 8 biti la stanga
    - Left shift entry wire: semnalul care intra in registrul Q in cadrul shiftarii la stanga. Left shift entry wire se va conecta la q[1]
    - Right shift Q: Folosit DOAR la inmultire. shifteaza registrul Q la dreapta cu o pozitie. Se shifteaza toti bitii cu o pozitie la dreapta
    - Right shift entry wire: semnalul care intra in registrul A in cadrul shiftarii la dreapta.
    - Bus de intrare la load
  + Semnale de iesire:
    - Q, iesirea bistabilelor, la operatii vom folosii semnalele q[8:1]. Cum am precizat mai devreme, q[0] este un semnal folosit doar pentru a face codificarea booth la inmultire
* Registrul M
  + Pe 8 biti
  + Are facilitati de incarcare paralela si shiftare la stanga cu o pozitie.
  + Este folosit pentru a retine al doilea operand de la inbus
  + Semnale de intrare:
    - Load: incarca registrul
    - Left shift M: Folosit DOAR la impartire, shifteaza registrul M la stanga cu o pozitie
    - Bus de intrare la load
  + Semnale de iesire:
    - M, iesirea bistabilelor
* Registrul QP
  + Pe 8 biti
  + Are facilitati de incarcare paralela, shiftare la stanga cu o pozitie si incrementare cu 1
  + Este folosit DOAR la impartire, are rol de a stoca rezultate ale codificarii Booth ale cifrelor A[8:6]
  + Semnale de intrare:
    - Load: incarca registrul
    - Left shift QP: Folosit DOAR la impartire, shifteaza registrul M la stanga cu o pozitie
    - Bus de intrare la load, se va incarca cu 0
    - Count up QP: semnal folosit pentru a incrementa registrul in cazul in care facem corectie la impartire, pentru aceasta incrementare am folosit half adder cell-uri, se poate vedea in fig 1.1
    - Left shift entry wire
  + Semnale de iesire:
    - iesirea bistabilelor

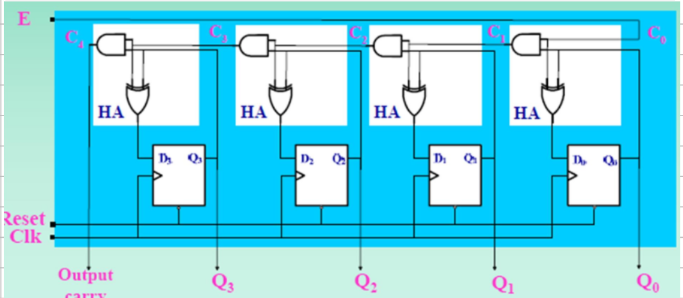


Fig. 1.1 Implementarea unui numarator crescator cu HAC-uri

* Parallel Adder
  + Folosit de catre toate cele 4 operatii
  + Pe 9 biti
  + Implementat sub forma de ripple carry adder, cu celule FAC
* EXOR Wordgate
  + Folosit atunci cand se doreste ca sumatorul sa se comporte ca un scazator
  + Ia ca intrari un bus pe 9 biti si un semnal de control si face XOR intre fiecare dintre semnalele bus-ului respectiv si semnalul de control
* COUNT 1
  + Pe 3 biti
  + Folosit DOAR in cadrul operatiei de impartire, retine cu cate pozitii am shiftat la stanga registrii A, Q si M pentru a scapa de leading zeroes
  + Stie sa numere atat in sus, cat si in jos
  + Are ca output valoarea cnt0 care indica daca numaratorul are valoarea 0
  + Pentru implementarea numaratorului am plecat de la urmatorul tabel:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Normal | | | C\_UP | | | C\_DOWN | | |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |

Dupa minimizari am obtinut urmatoarele formule:

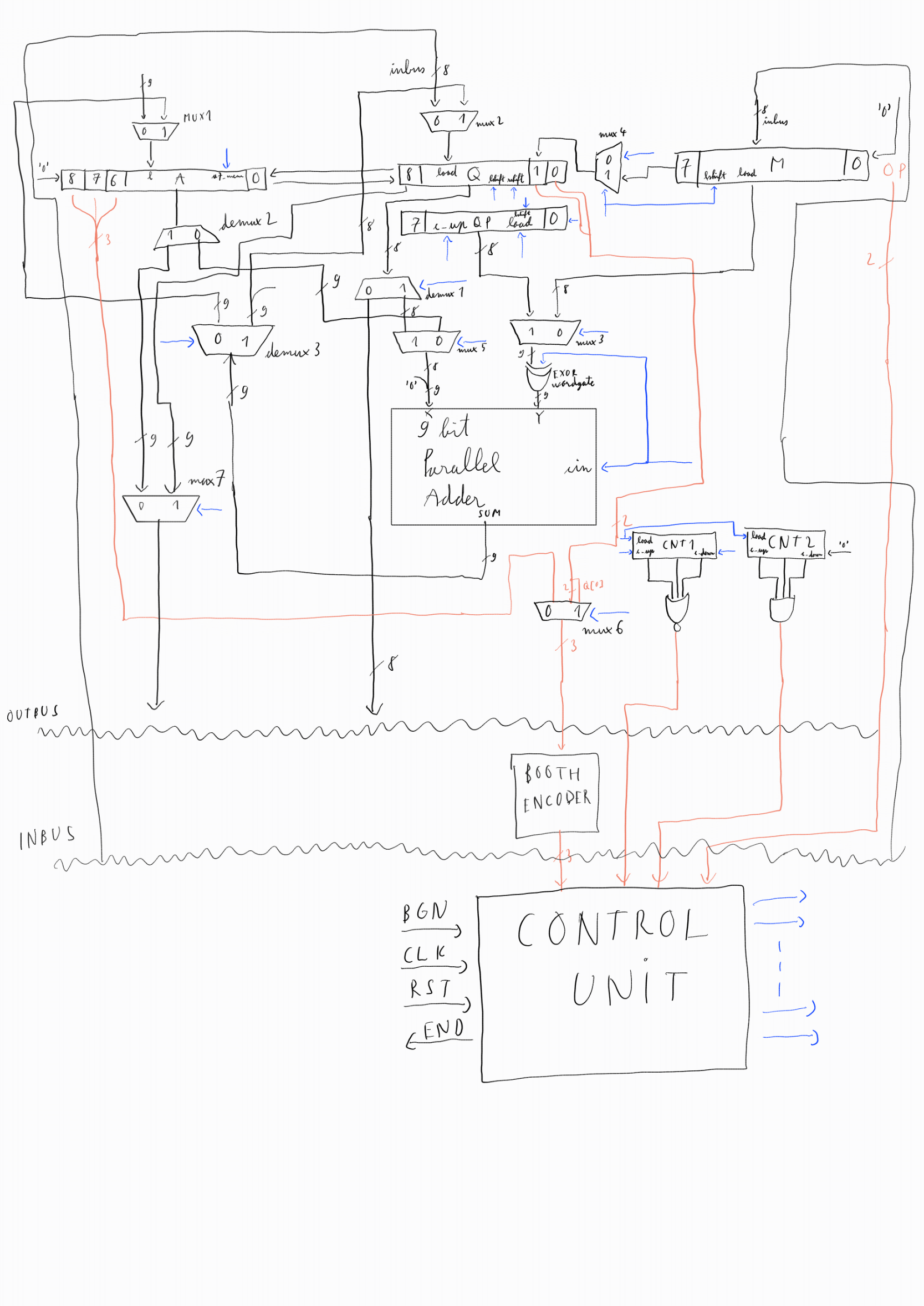
q[0]\_next = ~q[0] & (c\_up ^ c\_down)

q[1]\_next = (q[0]^q[1]) & c\_up & ~(c\_down)

q[2]\_next= c\_down & ~q[2] & ~q[1] & ~q[0]) | (c\_down & q[2] & q[0]) | (c\_down & q[2] & q[1]) | (c\_up & ~q[2] & q[1] & q[0]) | (c\_up & q[2] & ~q[1]) | (c\_up & q[2] & ~q[0])

* COUNT 2
  + Pe 3 biti
  + Folosit in cadrul operatiilor de inmultire si impartire, numara cate shiftari la stanga/dreapta am efectuat
  + Stie sa numere doar in sus
  + Are ca output semnalul cnt7 care ne spune daca numaratorul are valoarea 7
* Multiplexoare si demultiplexoare
  + Au rolul de a asigura ca semnalele ajung unde trebuie la fiecare pas al algoritmului astfel:
    - Mux1: Selecteaza ce sa primeasca registrul A la intrare (0 sau rezultatul sumer Parallel-adder-ului)
    - Mux2: Selecteaza ce sa primeasca registrul Q la intrare (primul operand sau rezultatul sumei parallel adder-ului)
    - Mux3: Selecteaza ce sa primeasca sumatorul ca si al doilea operand (iesirea registrului M sau a registrului Q prim), in special folosit pentru impartire
    - Mux4: Folosit DOAR la impartire, specifica ce sa primeasca registrul A cand facem shiftare la stangs (cel mai semnificativ bit al registrului M sau un rezultat al codificarii booth)
    - Mux5: Specifica ce sa primeasca sumatorul ca si prim operand (A sau Q)
    - Mux6: Folosit la inmultire si impartire, conceput sa selecteze care grup de biti se va codifica Booth (Q[1:0] sau A[8:6]))
    - Mux7: Folosit doar la adunare si scadere, rezolva problema semnului la iesire (iesirea este pe 16 biti, iar cand rezultatul, stocat in registrul Q, este negativ, cu alte cuvinte Q[8]=1, se copiaza Q[8] in cei mai semnificativi 8 biti ai rezultatului).
    - Demux1: Specifica daca iesirea registrului Q se va duce in sumator sau la outbus
    - Demux2: Specifica daca iesirea registrului A se va duce in sumator sau la outbus
    - Demux3: Selecteaza daca iesirea sumatorului se va duce in registrul A sau in registrul Q
* Booth encoder
  + Folosind in cadrul operatiilor de inmultire si impartire
  + Ia un grup de 3 biti si il transforma intr-un alt grup de 3 biti care vor reprezenta semnale de control de intrare pentru control unit astfel:
    - Daca configuratia reprezinta un 1 (001, 010, 011), asigneaza la iesire 100
    - Daca configuratia reprezinta un 0 (000 sau 111), asigneaza la iesire 010
    - Daca configuratia reperezinta –1 (100, 101, 110), asigneaza la iesire 001
  + Daca il folosim la inmultire, unde stim ca va trebui sa transformam numai 2 cifre dupa codificarea booth, va trebui sa extindem bus-ul de intrare la 3 cifre, copiind in cel mai putin semnificativ bit bitul q[0] de la iesirea registrului q (de exemplu 10 devine 100)

Schema hardware a circuitului este prezentata in diagrama urmatoare:



Control Unit

Am implementat Control Unit-ul pentru ALU folosind un automat de tip Moore folosind codificare One – Hot pentru stari , unde iesirile depind numai de starea curenta. Automatul are 15 stari in total, fiecare stare corespunzand unui bistabil de tip D.

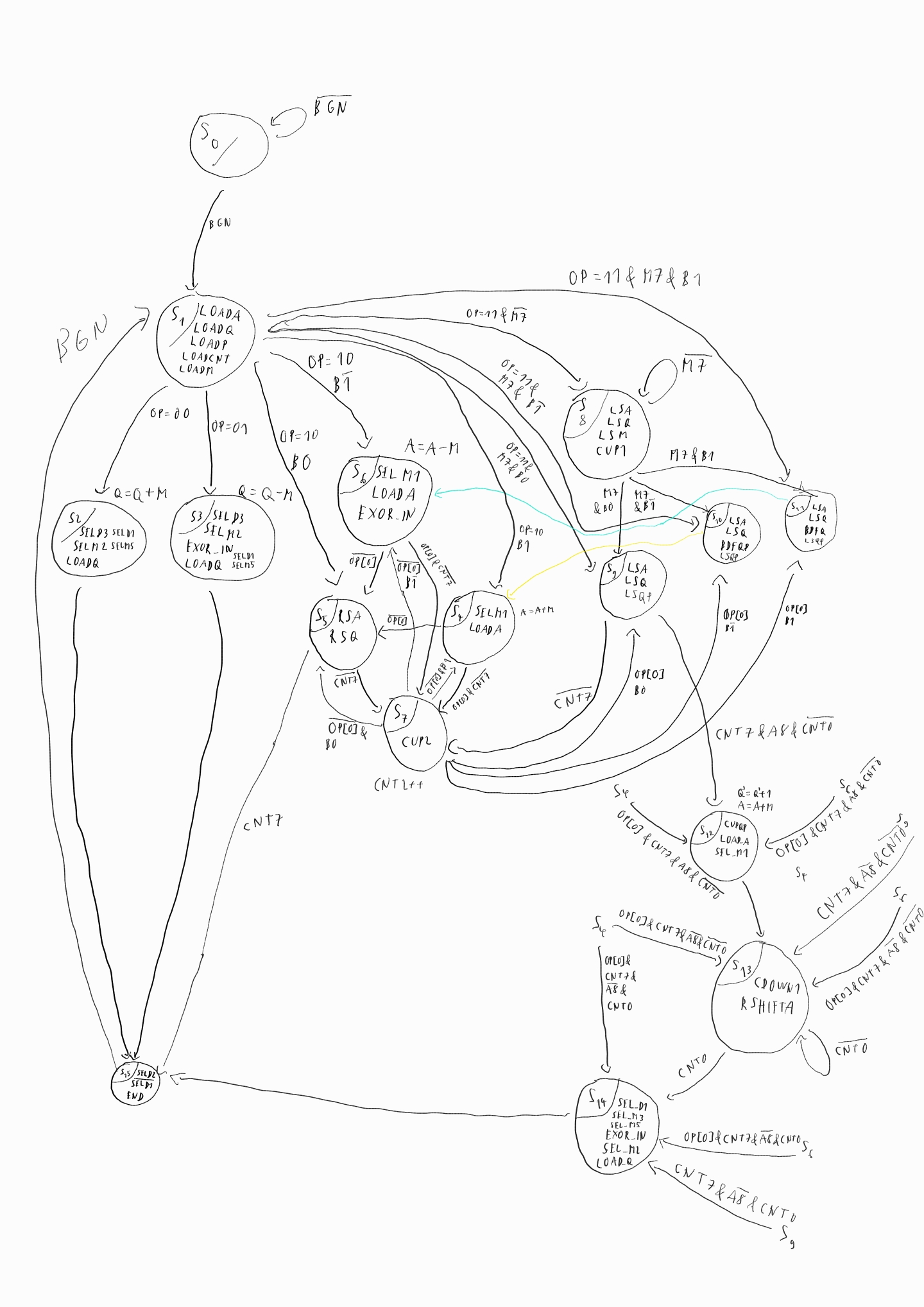
Semnificatie intrari:

* Clk – semnalul de clock al automatului
* Rst – semnalul de reset, aduce automatul in starea initiala (0)
* OP – semnal care codifica operatia dorita (00 - adunare, 01 – scadere, 10 – inmultire, 11 – impartire
* B1 – semnal care ne spune daca codificarea booth reprezinta un 1
* B0 – semnal care ne spune daca codificarea booth reprezinta un 0
* Bminus1 – semnal care ne spune daca codificarea booth reprezinta –1
* M7 – Semnal care ne spune daca m[7] este 1 sau 0, folosit la impartire
* A8 – Semnal care ne spune daca a[8] este 1 sau 0,folosit la impartire

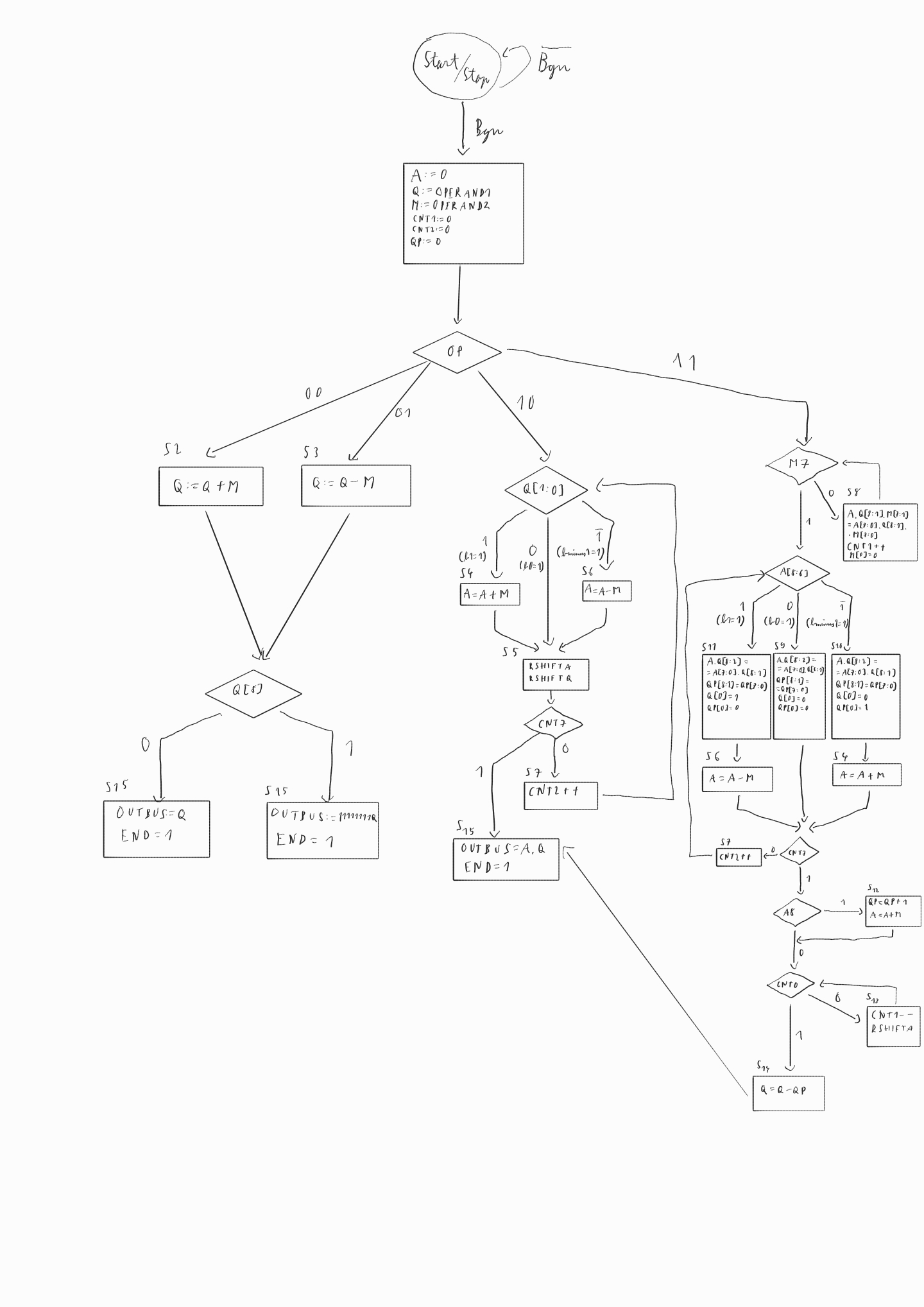
Semnificatie iesiri:

* Load\_a, incarca registrul A
* Load\_q, incarca registrul Q
* Load\_m, incarca registrul M
* Load\_cnt, incarca numaratoarele cu 0
* Sel\_mx, semnal de selectie pentru multiplexorul x din data path, unde x este un numar de la 1 la 7
* Sel\_demux\_x, semnal de selectie pentru demultiplexorul x, unde x este un numar de la 1 la 3
* Lshift A (LSA) - Shifteaza la stanga registrul A
* Rshift A (RSA) - Shifteaza la dreapta registrul A
* Lshift Q (LSQ) - Shifteaza la stanga registrul Q
* Rshift Q (RSQ) - Shifteaza la dreapta registrul Q
* Lshift M – shifteaza la stanga registrul M
* C\_up\_1 – incrementeaza primul numarator
* C\_up\_2 – incrementeaza al doilea numarator
* A7\_mem – folosit la inmultire
* Exor\_in – folosit cand vrem sa facem scadere
* Booth\_digit\_for\_Q – folosit la impartire, este 1 atunci cand codificarea booth a cifrelor A[8:6] este 1
* Booth\_digit\_for\_QP – folosit la impartire, este 1 atunci cand codificarea booth a cifrelor A[8:6] este –1

Reprezentarea diagramei automatului care implementeaza control unit-ul este atasata mai jos:



Pentru o intelegere mai clara a starilor si a tranzitiilor, ordinograma algoritmului impreuna cu starile corespunzatoare este prezentata mai jos:



Semnificatia starilor:

* S0: Starea initiala, se asteapta semnalul de begin (BGN)
* S1: Se incarca registrii Q si M cu operand1, respectiv operand2, si numaratoarele cu valoarea 0. Registrii A si QP se incarca, de asemenea, cu valoarea 0
* S2: Se executa operatia de adunare, se stocheaza in registrul Q Q+M, valabil numai cand op=00
* S3: Se executa operatia de scadere, se stocheaza in registrul Q Q-M, valabil numai cand op=01
* S4: Se executa A=A+M, ajungem in aceasta stare numai cand facem inmultire si impartire
* S5: Se shifteaza la dreapta registrii A si Q, doar la inmultire
* S6: Se executa A=A-M, ajungem in aceasta stare numai cand facem inmultire si impartire
* S7: Se incrementeaza al doilea numarator cand facem shiftarile la stanga/dreapta ale registrilor A si Q
* S8: Shiftam toti registrii cu o pozitie la stanga si incrementam primul numarator, cand facem impartire
* S9: shiftam registrii A, Q si QP cu o pozitie la stanga, iar Q[0] si QP[0] primesc valoarea 0, atunci cand A[8:6] este 0
* S10: shiftam registrii A, Q si QP cu o pozitie la stanga, Q[0]=1, QP[0]=0, atunci cand A[8:6] este 1
* S11: shiftam registrii A, Q si QP cu o pozitie la stanga, Q[0]=0, QP[0]=1, atunci cand A[8:6] este –1
* S12: Pasul de corectie de la SRT-2, incrementam QP cu 1 si A=A+M
* S13: Pasul de shiftare de la SRT-2, Se decrementeaza CNT1 si se shifteaza registrul A cu o pozitie la dreapta
* S14: Se face scaderea finala, Q=Q-QP
* S15: Stocam in outbus rezultatul final si semnalul de end se activeaza, ramanem in aceasta stare pana cand se comanda reinceperea algoritmului (BGN=1)

Conexiunea Data path – Control unit

Pentru a conecta cele 2 module am folosit un semnal de clock care intra normal in control unit si inversat in data path pentru a realiza un delay de jumatate de ciclu de tact cu scopul de a realiza handshaking-ul dintre cele doua.

Implementare:

Pentru implementare si testare am folosit platforma Modelsim. Cele mai importante module din implementare sunt data\_path.v si control\_unit.v. Ele sunt conectate intr-un testbench numit ALU\_tb.v pentru a forma un ALU complet functional pe 8 biti.

Input-urile ALU sunt operand1, pe 8 biti, operand2, pe 8 biti, op, pe 2 biti, specifica operatia pe care dorim sa o efectuam.

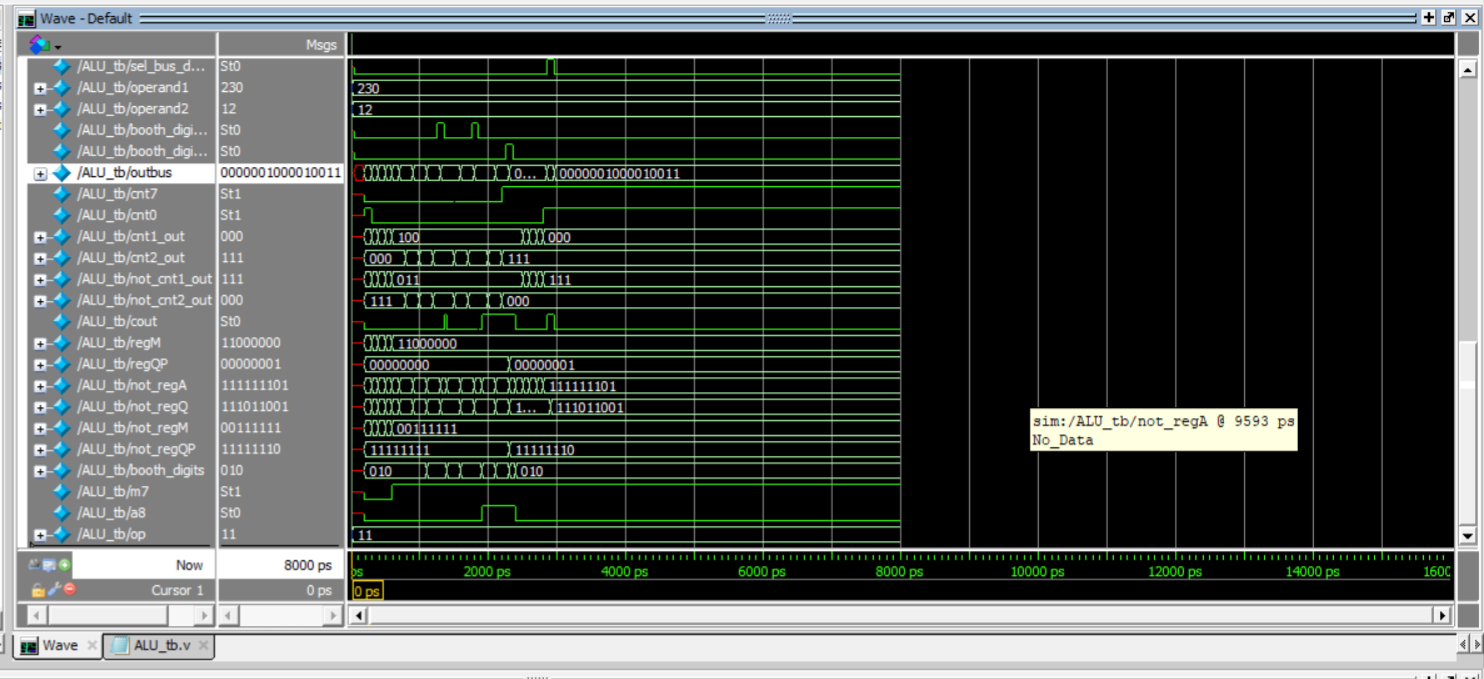
Output-ul este outbus, pe 16 biti, unde stocam rezultatul final

Testare:

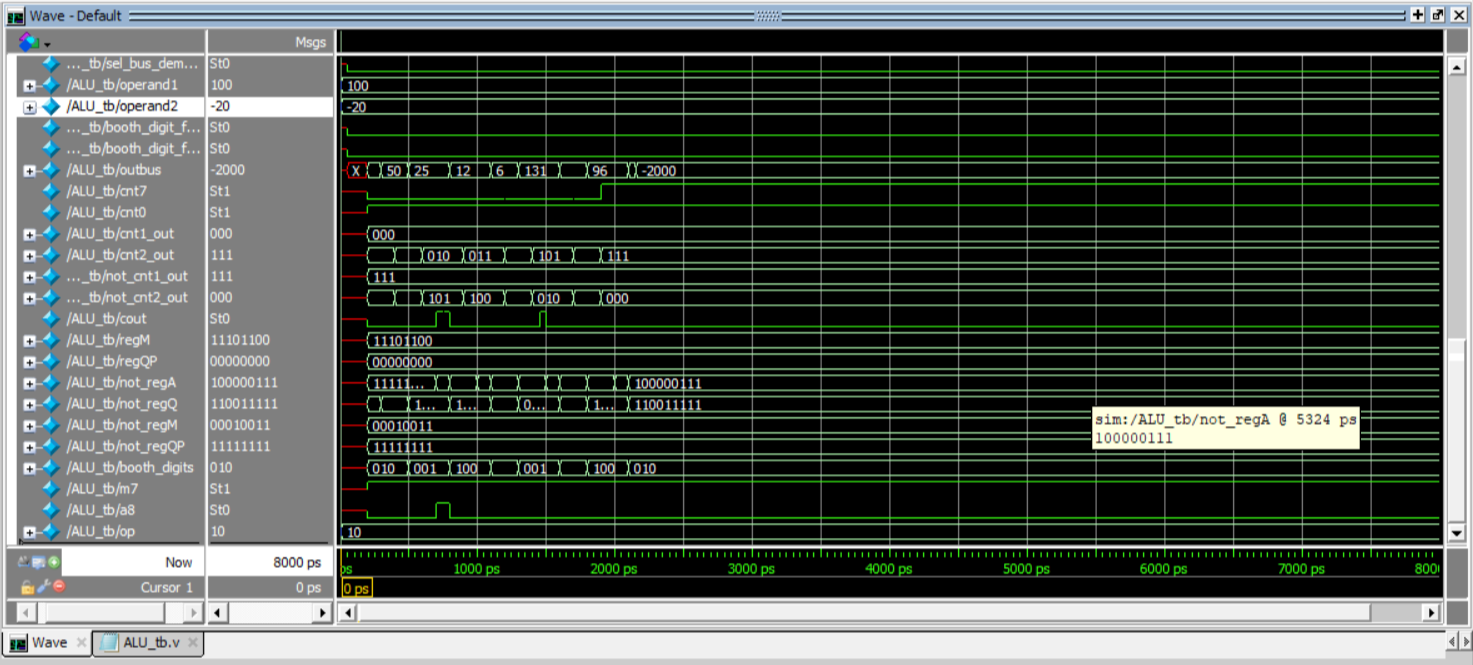
Pentru testare am luat cateva cazuri de adunare, scadere, inmultire si impartire.

Primul exemplu:

* In acest exemplu am calculat 230/12 astfel:
  + In operand 1 am stocat 230
  + In operand 2 am stocat 12
  + In op am stocat 11, deoarece facem impartire
  + Observam ca la linia de **outbus** ne afiseaza 0000001000010011 deoarece restul este 2 (00000010) si catul este 19 (00010011)

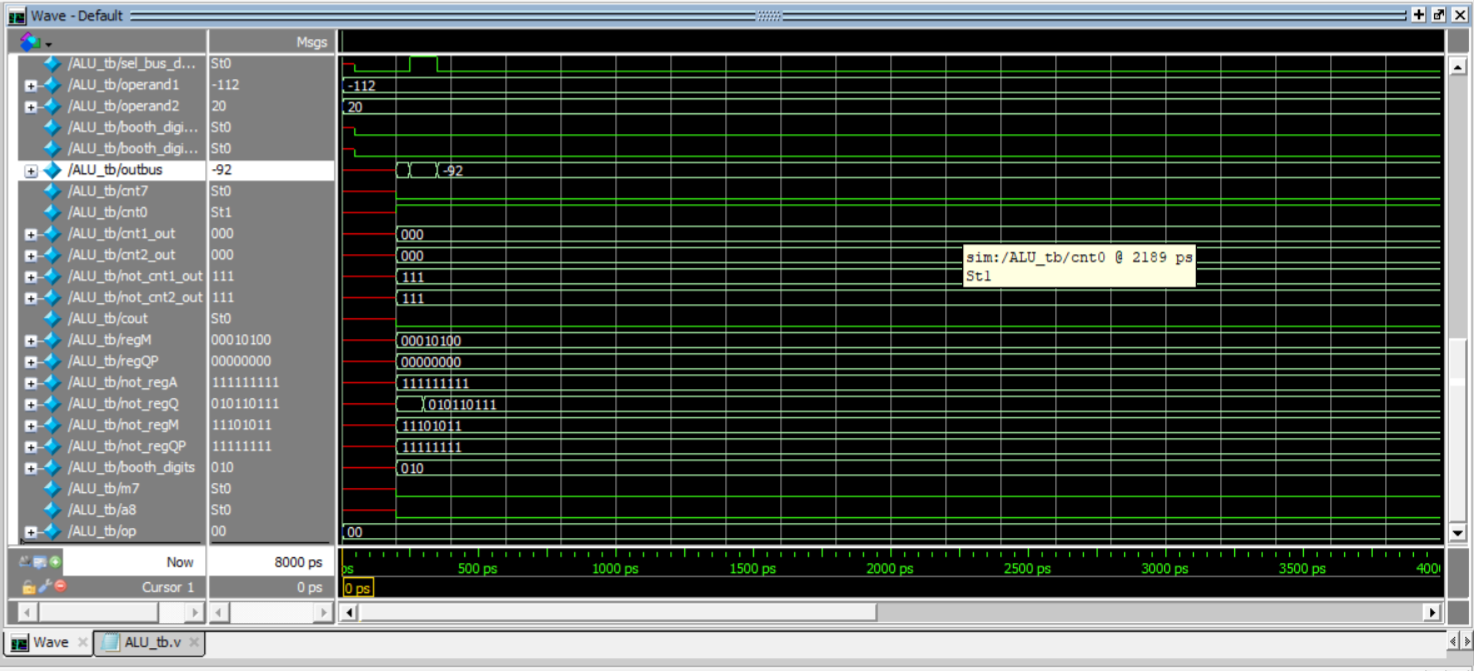


Al doilea exemplu:



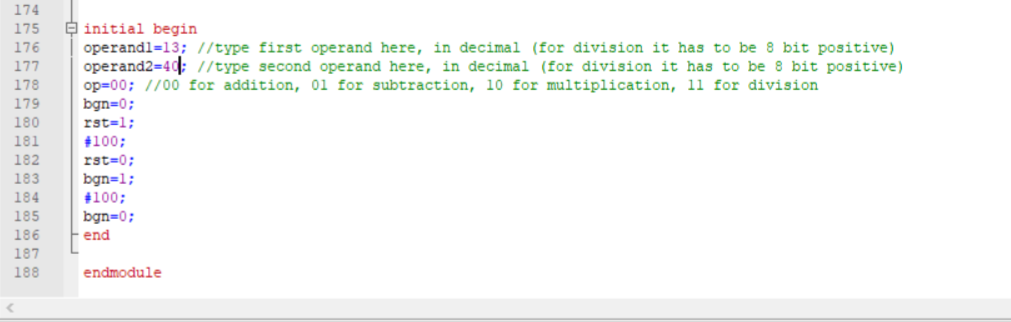
* Aici am calculat 100\*-20 astfel:
  + Am stocat in operand1 100
  + Am stocat in operand2 -20
  + In op am pus 10
  + Observam ca la linia de **outbus** ne afiseaza –2000

Exemplul 3:



* Aici am calculat –112+20 astfel:
  + In operand 1 am stocat –112 (10010000)
  + In operand 2 am stocat 5 (00000101)
  + In op am pus 00
  + Observam ca la linia de **outbus** ne afiseaza –92

In testbench se pot introduce operanzi si codificare a operatiei astfel:



Diverse provocari

Cea mai mare provocare in realizarea proiectului a fost conexiunea unei iesiri pe un numar de biti la o intrare pe un numar diferit de biti. Pentru a rezolva aceasta problema, am pus multiplexoarele si demultiplexoarele pe numarul maxim necesar de biti (9 biti) deoarece in implementare am folosit atat registrii pe 9 biti, cat si registrii pe 8 biti

Concluzie

Acest proiect ajuta la intelegerea modului in care un procesor face operatii matematice simple, folosind doar logica binara. De asemenea, ajuta la intelegerea modului in care implementam un automat capabil sa emita semnale de control corespunzatoare instructiunilor pe care vrem sa le executam la un moment dat. Cu toate ca acesta nu este tocmai cel mai eficient mod de a efectua inmultiri si impartiri, aceasta implementare asigura un echilibru intre optimizarea spatiului ocupat si al timpului de executie.