Universitatea Politehnica Timișoara

**Proiect Organizarea Calculatoarelor**

**Booth – Radix 4**

Petrescu Alex

Pistrilă Paula

Calculatoare și Tehnologia Informației

Anul 2, grupa 5.1

Profesor coordonator: David Tudor-Mihai

Cuprins

[Cuprins 2](#_Toc136209737)

[***Funcționalitate*** 3](#_Toc136209738)

[***Implementare*** 4](#_Toc136209739)

[***Concluzie*** 8](#_Toc136209740)

[***Referințe bibliografice*** 8](#_Toc136209741)

# ***Funcționalitate***

Algoritmul Booth Radix 4 este o eficientizare a algoritmului Booth clasic, deoarece acesta are o eficiența de timp scăzuta din cauza operatiei de adunare intre produsele partiale. Prin formarea a grupari de cate 3 biti pentru a fi evaluati la fiecare iteratie numarul produselor partiale se injumatateste. Deci, algoritmul Booth Radix-4 este aproximativ de doua ori mai rapid ca si Booth Radix-2.

|  |  |  |  |
| --- | --- | --- | --- |
| Q[i+1] | Q[i] | Q[i-1] | Operația |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 2 |
| 1 | 0 | 0 |  |
| 1 | 0 | 1 |  |
| 1 | 1 | 0 |  |
| 1 | 1 | 1 | 0 |

0 = shiftare

1 = +M

2 = +2M

= -M

= -2M

-M este complementul de doi al lui M.

2M este calculat prin shiftarea lui M la stanga cu o pozitie, iar LSB al lui M este setat pe zero.

Analog, -2M este complementul de 2 al lui 2M.

Tabelul de codificari

Pasii algoritmului:

1. Registrului A setati toti bitii pe 0
2. Bitul Q[i-1] primeste valoarea 0
3. Sunt calculati M, -M, 2M, -2M
4. Se evalueaza fiecare grupare Q[i+1]Q[i]Q[i-1] si se aduna cu corespondentul din tabel
5. MSB-ul lui A se duplica in noul A, iar ceilalti biti sunt shiftati la dreapta
6. Q se shiftează la dreapta
7. Se incrementeaza counter-ul

Vom ilustra algoritmul cu un exemplu trivial pe 8 biti:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Count | A | Q | Q[-1] | M |
| 00 | 0000 0000  11011 0101  11011 0101  11110 1101 | 0110 0101  0101 1001 | 0  0 | M=11011 0101  -M=00100 1011  2M=10110 1010  -2M=01001 0110 |
| 01 | 11011 0101  11010 0010  11110 1000 | 1001 0110 | 0 |  |
| 10 | 01001 0110  00111 1110  00001 1111 | 1010 0101 | 1 |  |
| 11 | 10110 1010  11000 1001  11110 0010 | 0110 1001 | 0 |  |

Astfel, rezultatul inmultirii este pe 2n-1 biti si se obtine din concatenarea acumulatorului A cu multiplicandul Q, iar bitul Q[-1] este ignorat.

# ***Implementare***

Inmulțitorul lui Booth poate fi implementat ca si un circuit secvential, cand fiecare produs partial este generat si acumulat intr-un singur ciclu de tact sau ca si un circuit combinational, unde produsele partiale sunt generate in paralel. Pentru implementare am folosit software-ul de simulare Modelsim.

Am ales sa implementam algoritmul lui Booth in limbajul Verilog, cu intrări de 8 biți fiecare. Pentru implementare avem nevoie de registrii A,Q,M, o variabilă de numărare Count, un multiplexor, o unitate de adunare paralela si o unitate de control.

Folosind nomenclatura arhitecturii Harvard, se definesc:

*A* - acumulatorul unde se va calcula o parte din rezultat, este initializat cu 0, este stocat pe n+1 biti

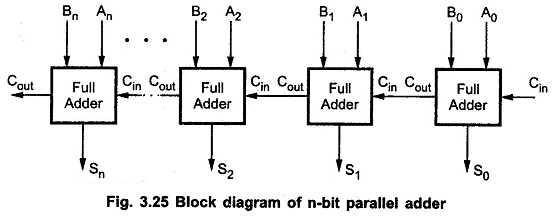
*Q* – multiplicandul, este stocat pe n biti

*M* – multiplicatorul, este stocat pe n biti

Count – este calculat ca log(n), unde n este numarul de biti al M si Q si tine evidenta pasilor efectuati

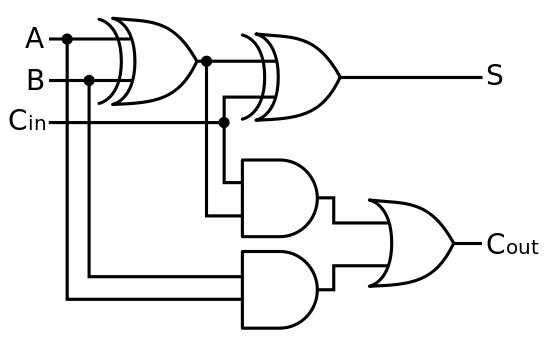
#### **Parallel adder**

Implementarea unui parallel adder pe n biți, în schema hardware, noi am folosit unul pe 9 biți. Mai jos este prezentată schema hardware a parallel adder-ului:



*Full Adder Cell*

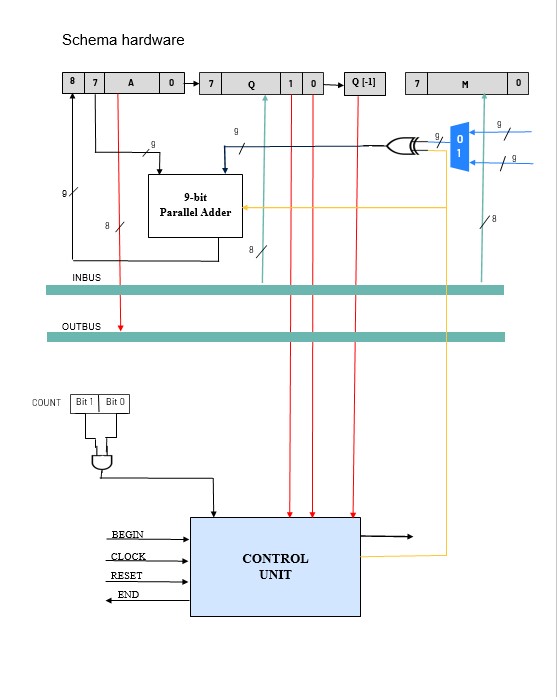
 Mai jos este prezentată o celulă Full Adder construită la nivel de poartă. Intrarile A si B reprezintă numerele ce urmează a fi adunate, iar Cin este carry in-ul. Iesirea S este suma calculată, iar Cout este carry out-ul. Toate variabilele din modulul Full Adder Cell sunt pe un bit.

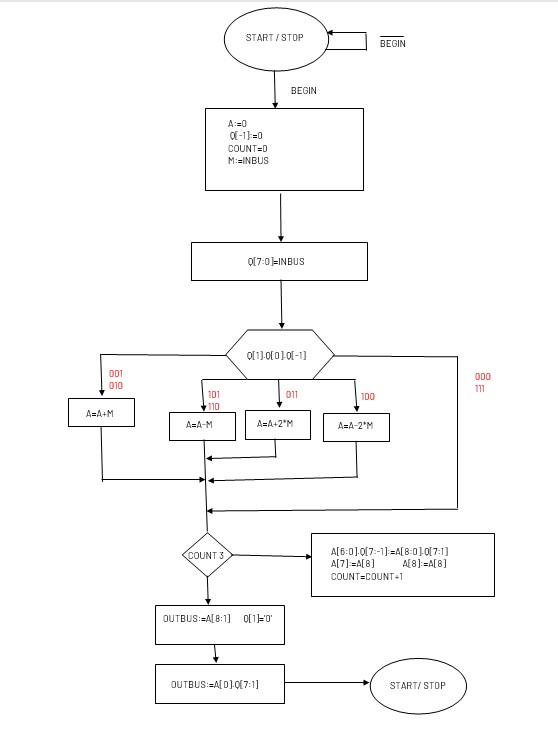


#### **Unitatea de control**

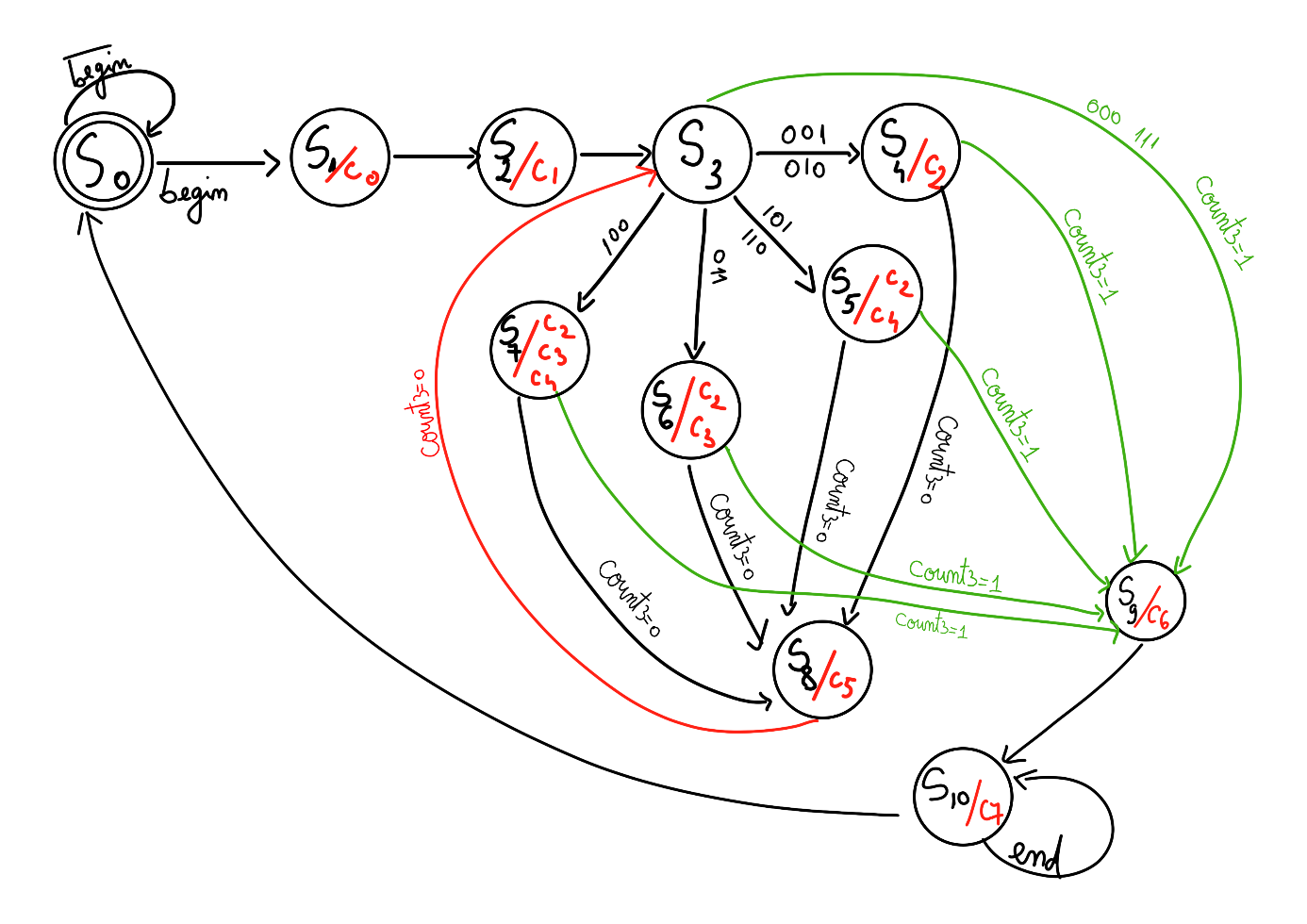
Pentru implementarea unității de control, noi am dezvoltat-o ca un Finite State Machine cu 8 stări, prezentate mai jos:

* *begin* - demareaza algoritmul de înmulțire
* *q[1:0]* - ultimii 2 LSB registrului Q
* *q\_minus1* - Q[-1]
* *count3* - valoarea counter-ului e egala cu 3
* *c0* - sterge registrul A si counter-ul (le seteaza toti bitii pe 0), incarca M in INBUS
* *c1* - incarca Q in INBUS
* *c2* - incarca rezultatul sumatorului inapoi in registrul A
* *c3* - determina operatia de adunare/scadere
* *c4* – determina scaderea cu reigstrul M sau 2M
* *c5* - incrementeaza counter-ul, deplaseaza A concatenat cu Q la dreapta, incarca A[8] la A[7] si A[8]
* *c6* - livreaza registrul A la OUTBUS si il seteaza pe Q[0] pe 0, in functie de caz
* *c7* - livreaza pe registrul A[0] concatenat cu Q[7:1] la OUTBUS
* *end* - marcheaza finalul operatiei, semnalul este activat impreuna cu c6 si c7





# 

***Diagrama de stări***

# ***Concluzie***

Pe parcursul implementarii, am intampinat dificultati in legătură cu actualizarea valorii registrului M auxiliar din modulul parallel adder, ce nu primeste valoarea registrului M conform semnalelor de control.

Se pot observa îmbunătățirile considerabile aduse algoritmului Booth Radix-2, ce duc la înjumătățirea timpului de executie, ceea ce determină popularitatea crescută a algoritmului Booth Radix-4.

# ***Referințe bibliografice***

* <https://forum.digikey.com/t/booth-radix-4-multiplier-for-low-density-pld-applications-verilog/13401>
* <http://vlsigyan.com/booth-multiplier-verilog-code/>
* <https://www.researchgate.net/figure/Parallel-architecture-of-proposed-radix-4-8-bit-Booth-multiplier_fig2_330685391>
* <http://people.ee.duke.edu/~jmorizio/ece261/F08/projects/MULT.pdf>
* <http://vlsiip.com/download/booth.pdf>