Documentatie

Restoring division

Proiect 1 OC:

Membrii echipei:

Spatacean Oana

Stoichescu Iulia

Grupa: 2.C.06.1

Cuprins:

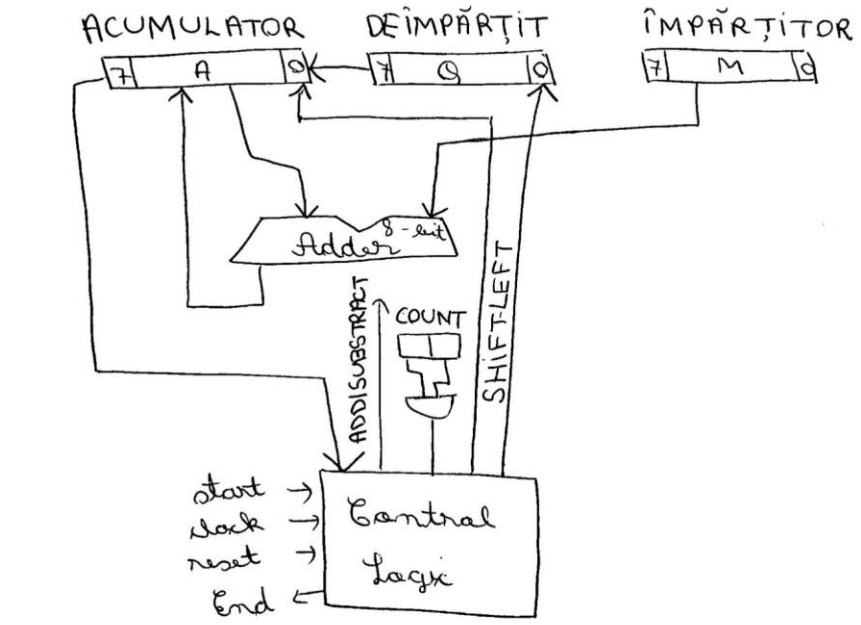
1.Introducere

Restoring Division este un algoritm de împărțire între două numere binare, deimpartit și impartitor. În acest algoritm, împărțirea este realizată prin substracții succesive, asemănătoare împărțirii cu rest.

Această împărțire este numită "restoring division" deoarece, după fiecare substracție, trebuie restaurat bitul cel mai semnificativ al deimpartitului pentru a se asigura că se obține un rezultat corect.

În acest document, vom prezenta o implementare a algoritmului Restoring Division în limbajul Verilog.

2.Design



Design-ul hardware al algoritmului de împărțire prin restaurare (restoring division) constă într-un modul Verilog care implementează operațiile aritmetice de împărțire a două numere întregi fără semn.

Modulul Verilog include un bloc de definire a registrilor și un bloc de definiție a semnalelor de intrare și ieșire. Semnalele de intrare includ semnalele de ceas, reset și start, precum și valorile deimpartitului și împărțitorului. Semnalele de ieșire includ valoarea catului și a restului, precum și semnalul valid.

Algoritmul de împărțire prin restaurare funcționează în modul următor:

1. Se așteaptă semnalul de start.
2. Se încarcă deimpartitul în Q și se incarcă 0 în registrul A.
3. Se execută operația de împărțire a lui A cu M până când toți biții sunt procesați.
4. Dacă semnalul de reset este activat, toate valorile semnalelor de ieșire și de starea curentă sunt resetate.
5. Se utilizează o serie de registre pentru a stoca valorile anterioare și a le utiliza la următoarea operație.

Modulul Verilog implementează această logică de împărțire prin restaurare, permițând realizarea operațiilor aritmetice de împărțire a două numere întregi fără semn.

În implementarea hardware a algoritmului de împărțire prin restaurare, se utilizează o serie de registre pentru a stoca valorile anterioare și a le utiliza la următoarea operație. Această abordare minimizează numărul de resurse utilizate, iar modulul Verilog poate fi implementat pe un circuit integrat cu putere scăzută, cu un consum redus de energie.

3.Implementare

Algoritmul Restoring Division are două intrări: deimpartit și impartitor. Acestea sunt reprezentate de semnalele de intrare Q și M. Algoritmul Restoring Division are, de asemenea, trei semnale de ieșire: restul, câtul și semnalul de validare. Este un algoritm iterativ care are nevoie de un registru (A) și de un număr întreg (M) (impartitorul), care sunt ambele inițializate cu valorile de intrare și care sunt actualizate la fiecare iterație până când toți biții de ieșire sunt obținuți.

Pentru a implementa algoritmul Restoring Division, vom utiliza un set de registre care reprezintă starea curentă a algoritmului, precum și variabile auxiliare pentru a efectua calculele necesare pentru împărțire. Modulul Verilog pentru Restoring Division utilizează patru registre (A, A_urm, A_aux, A_aux2) și patru stări (idle, start, wait, finish) pentru a executa împărțirea. Registrele sunt actualizate pe baza stării curente, a semnalului de ceas și a semnalului de reset, precum și pe baza intrărilor de împărțit și de împărțitor.

Modulul conține două blocuri "always". Primul bloc "always" este activat la orice front crescător pe semnalul de ceas sau orice front descrescător pe semnalul de reset și setează valorile

de ieșire în funcție de valorile următoare ale registrilor interni. Al doilea bloc always efectuează efectiv calculul de împărțire. Algoritmul este efectuat în 8 de iterații, una pentru fiecare bit al deimpartitului și câtului.

Modulul începe în starea idle, așteptând semnalul de start. După ce semnalul de start este primit, modulul trece în starea start și încarcă valorile corespunzătoare în registrul A și în registrele auxiliare A_aux și A_aux2.

În starea start, modulul execută operația de împărțire prin shifarea la stânga a valorii din A și scăderea acesteia cu M. Rezultatul este apoi încărcat în registrul A_urm.

Dacă bitul cel mai semnificativ din A_aux2 este 1, atunci numărul deimpartit este prea mare și trebuie să se scadă din acesta valoarea impartitorului. În acest caz, bitul din Q este setat la 0 și se reține aceeași valoare a registrului A ca în iterația anterioară.

În caz contrar, bitul din Q este setat la 1 și registrul A primește valoarea lui A_aux2.

După ce toți biții au fost procesați, modulul trece în starea finish și semnalul de validare este setat la 0.

În orice moment, semnalul de reset poate fi activat pentru a reseta modulul la starea idle.

Semnale de intrare și ieșire

Modulul Restoring Division:

primește două semnale de intrare, deimpartit și impartitor, și returnează două semnale de ieșire, cat și rest. Toate semnalele sunt de tip reg și sunt de lungime 8 de biți.

Q: Semnalul de intrare care reprezintă deimpartitul de 8 de biți.

M: Semnalul de intrare care reprezintă impartitorul de 8 de biți.

cat: Semnalul de ieșire care reprezintă câtul de 8 de biți.

rest: Semnalul de ieșire care reprezintă restul de 8 de biți.

Algoritm pe scurt:

A[7:0],Q[7:0],M[7:0]

SUBSTRACT: A.Q[7:1] = A.Q;

A = A-M

if(A[7] == 0) than Q[0] = 1

if(A[7] == 1) than A = A+M, Q[0] = 0;

if(count == 0) than goto OUTPUT

else goto SUBSTRACT

Flowchart:

START

A=0

M=deimpartit

Q=impartitor

SUBSTRACT

Shifteaza stanga AQ;

A = A – M;

Verifica semnul lui A: daca e 0, atunci Q[0] = 1;

daca e 1, atunci Q[0] = 0 si A = A +M, adica e restaurat; (restored)

Decrementeaza count;

Daca count e 0, atunci mergi la **STOP;**

Daca nu, atunci revino la SUBSTRACT si repeta-l.

Rezolvarea algortmica exemplificata cu o problema din laborator:

count	A	Q	M
0	0 0001 0110 0 0010 1101+ 1 0111 1001 = 1 1010 0110+ 0 1000 0111 = 0 0010 1101	1000 1011 0001 011_ 0001 0110	0 1000 0111
1	0 0 101 1010+ 1 0111 1001 = 1 1101 0011+ 0 1000 0111 = 0 0101 1010	0010 110_ 0010 1100	
2	0 1011 0100+ 1 011 1001 = 0 0010 1101	0101 100_ 0101 1001	
3	0 0101 1010+ 1 0111 1011 = 1 1100 1011+ 0 1000 0111 =0 0101 1010	1011 001_ 1011 0010	
4	0 1011 0101+ 1 0111 1001 = 0 0010 1110	0110 010_ 0110 0101	
5	0 0101 1100+ 1 0111 1001 = 1 1101 0101+ 0 1000 0111 = 0 0101 1100	1100 101_ 1100 1010	
6	0 1011 1001+ 1 0111 0010 = 0 0011 0010	1001 010_ 1001 0101	
7	0 0110 0101+ 1 0111 1001 = 1 1101 1110+ 0 1000 0111 =0 0110 0101	001 0101_ 0010 1010	

5771:135 Restul= 0 0110 0101= 101 Catul= 0010 1010=42

5771= 0001 0110 1000 1011

135= 0000 0000 1000 0111

M= 0 1000 0111

-M= 1 0111 1001

Cod Verilog:

```
module RestoringDivision(
_clock,
_reset,
_start,
_Q, //deimpartit, A[7:4]
_M, //impartitor
```

```
_valid,
_cat,
_rest
);

input clock;
input reset;
input start;
input [3:0]Q,M;

output [3:0]cat,rest;
output valid;

//definirea registrilor
reg [7:0] A,A_urm,A_aux,A_aux2;
reg valid, valid_urm;




reg stare, stare_urm;
reg [1:0] count,count_urm;
 //extrag date din registrul A si le atribui altor variabile
assign rest = A[7:4];
assign cat = A[3:0];

always @(posedge clock or negedge reset) //executa blocul ori de câte ori se produce un front crescător pe semnalul de ceas sau un front descrescător pe semnalul de reset
begin
if(!reset) //verifică dacă semnalul de reset este inactiv
```

```
begin//inactiv
A <= 0;//setez valorile la iesire
valid <= 0;
stare <= 0;
count <= 0;
end
else
begin//activ
A <= A_urm;
valid <= valid_urm;
stare <= stare_urm;
count <= count_urm;
end
end

always @(*) //executa blocul de fiecare dată când oricare din intrări se schimbă
_begin



if(stare == 0) //idle, așteaptă să primească o intrare
begin //seteaza valorile de pornire ale semnalelor de intrare
count_urm = 0;
valid_urm = 0;
//așteaptă să primească un semnal start pentru a începe operația de împărțire
if(start == 1) //Dacă semnalul start este primit
begin
stare_urm = 1; //modulul trece in starea de start
A_urm[3:0] = Q; //in Q(din tabel) se incarca deimpartitul
A_urm[7:4] = 0; //in A(din tabel) se incarca 0
```

```
end
else ////Dacă semnalul start nu este primit
begin
stare_urm = stare; //se ramane in starea curenta
A_urm[7:0] = 0; //se incarca 0 si in A si si in Q(din tabel)
end
end

if(stare == 1) //start, executa impărțirea, împărțire a lui A cu M
begin
count_urm = count + 1; //incrrementeaza count
A_aux = A<<1; //shifteaza la stanga AQ
A_aux2[7:4] = A_aux[7:4]-M; //A=A-M

//se verifica semnul lui A
if(A_aux2[7] == 1) //daca e 1, pozitiv



begin
A_urm[0] = 0; //Q[0] = 0
//restul biților din A_urm sunt setați la aceeași valoare ca și cei din A_aux = A<<1
A_urm[3:1] = A_aux[3:1];
A_urm[7:4] = A_aux[7:4];
end
else
begin
A_urm[0] = 1; // //Q[0] = 1
//restul biților din A_urm sunt setați la aceeași valoare ca și cei din A_aux2 = A_aux - M
        A_urm[3:1] = A_aux[3:1];
```

```
A_urm[7:4] = A_aux2[7:4];
end

//verifică dacă toți biții au fost procesați
if(&count == 0) //o operație logică SI pe toți biții variabilei count
begin
valid_urm = 0;
stare_urm = stare; // modulul se întoarce la starea idle
end
else
begin
valid_urm = 1;
stare_urm = 0; //modulul trece în starea de așteptare
end
 end


end
endmodule
```

4.Testare

Pentru a testa algoritmul Restoring Division, putem simula modulul într-un simulator Verilog. În această simulare, putem aplica semnale de testare pentru deimpartit și impartitor și putem verifica semnalele de ieșire pentru a confirma că modulul funcționează așa cum este prevăzut. De asemenea, putem verifica rezultatele împărțirii pentru diferite combinații de deimpartit și impartitor pentru a asigura că algoritmul produce întotdeauna un rezultat corect.

Testbench-ul generează semnalele de intrare Q și M, semnalul de reset, semnalul de start și semnalul de ceas pentru modulul RestoringDivision, apoi monitorizează semnalele de ieșire cat, rest și valid pentru a verifica dacă modulul RestoringDivision funcționează corect. Se utilizează o instanțiere a modulului RestoringDivision pentru a testa design-ul hardware al algoritmului de împărțire prin restaurare.

Testbench-ul a fost implementat utilizând Verilog și conține un bloc always pentru a genera semnalul de ceas și un bloc initial pentru a genera semnalele de intrare Q și M, semnalul de reset și semnalul de start. Semnalele de intrare Q și M sunt setate inițial la valorile 14 și 5, respectiv, iar semnalul valid este utilizat pentru a indica faptul că rezultatul este disponibil. Testbench-ul așteaptă apoi până când semnalul valid este activat, după care afișează rezultatul obținut și așteaptă 10 unități de timp înainte de a reîncepe testul cu valori diferite pentru Q și M.

Testbench:

```
module RestoringDivision_tb;


reg clock,reset,start;
reg [3:0]Q,M;


wire [3:0]cat,rest;//pentru a obține ieșirile cat și rest
wire valid; //semnalul valid care indică dacă rezultatul este disponibil


always #5 clock = ~clock; //"clock" e alternativ 0 și 1 la fiecare perioadă a semnalului de ceas



initial
  begin
    $dumpfile("dump.vcd");
    $dumpvars(1);


    $monitor("%d : %d = %d rest %d    valid=%d",Q,M,cat,rest,valid);


    Q=14;
    M=5;
    //așteaptă până când semnalul valid este activat
    clock=1;
    reset=0;
```

```
    start=0;

    #10
    reset = 1;

    #20
    start = 1;

    #10
    start = 0;

    @valid

    #10
    $display("\n");

    //reia testul
    Q=10;
    M=3;

    start = 1;

    #10
    start = 0;

  end

RestoringDivision instantiere(
  .clock(clock),
  .reset(reset),
```

```
  .start(start),
  .Q(Q),
  .M(M),
  .valid(valid),
  .cat(cat),
  .rest(rest)
);

Endmodule
```

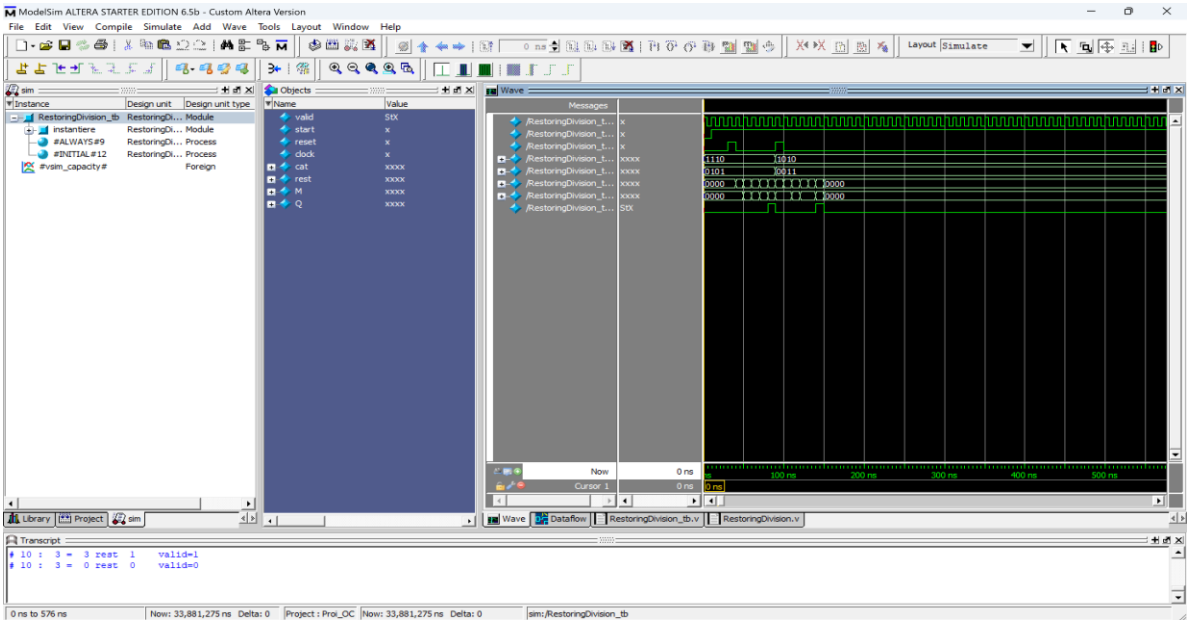
Simulare:

Modul de simulare:

-simulate->start simulation

-la objects add wave->signals in region

-simulate->run->run all



În timpul simulării, putem observa valorile semnalelor la fiecare perioadă a semnalului de ceas. Putem analiza semnalele și putem verifica dacă rezultatele sunt cele așteptate. În cazul în

care avem semnale de ieșire neașteptate, putem utiliza informațiile obținute din simulare pentru a identifica și corecta problemele din designul nostru.

5.Concluzie

În concluzie, implementarea algoritmului Restoring Division în Verilog este un proces complex, dar bine definit, care implică mai mulți pași. De la proiectarea arhitecturii hardware și a modulului, la scrierea testbencherului și simularea în Verilog, toate acestea sunt necesare pentru a valida funcționarea corectă a algoritmului. Testarea este crucială pentru a verifica dacă modulul Restoring Division generează rezultatele corecte pentru toate cazurile de test.

Implementarea algoritmului Restoring Division în Verilog poate fi utilizată în aplicații hardware și software care necesită divizarea a două numere întregi, cum ar fi calculatoare, procesoare și alte aplicații de procesare a datelor. Acesta poate fi, de asemenea, îmbunătățit și optimizat pentru a reduce timpul de calcul și resursele hardware necesare.

Documentația de mai sus a descris implementarea modulului și semnalele de intrare și ieșire.

6.Resurse

- **https://www.geeksforgeeks.org/restoring-division-algorithm-unsigned-integer/**

https://atozmath.com/example/RestoringDivision.aspx?he=e

https://www.gyaanibuddy.com/assignments/assignment-detail/restoring-method-of-division/

https://www.javatpoint.com/restoring-division-algorithm-for-unsigned-integer