

Organizarea Calculatoarelor

LABORATOR 4 – Implementarea și simularea unui sumator/scăzător pe 4 biți

SCOPUL LUCRĂRII

Pe baza sumatorului pe 4 biți implementat anterior, se cere construirea unui sumator/scăzător cu operanzi reprezentați pe 4 biți. Funcționarea corectă a sumator/scăzătorului se va verifica prin simulare.

Chestiuni teoretice

Dacă nu vă mai amintiți complementul față de 2, citiți Anexa 1 (sau orice alt material!).

Pentru a proiecta un bloc aritmetic care oferă la ieșire fie suma, fie diferența, este nevoie de un semnal care să decidă ce operație se face: adunare sau scădere. Numele acestui semnal va fi **S1A0** (Scădere 1, Adunare 0). Dacă S1A0 este 0L se va face adunare iar dacă este 1L se va face scădere. **Scăderea se va face prin adunarea complementului față de 2.**

Fie două numere întregi α și β . Acestea se reprezintă în complement față de 2. $A=C2(\alpha)$ este reprezentarea în complement față de 2 a lui α iar $B=C2(\beta)$ este reprezentarea în complement față de 2 a lui β . Rezultatul adunării sau scăderii se va numi ρ iar reprezentarea sa în complement față de 2 este $R=C2(\rho)$.

În cazul **adunării** $\rho = \alpha + \beta$ se poate demonstra că

$$C2(\rho) = C2(\alpha) + C2(\beta) \quad \text{echivalent cu} \quad R = A + B$$

„+” este adunare pentru numere fără semn. Adică este suficient să adună reprezentările în complement față de 2 ale lui α și β **ca și când ar fi numere fără semn** și rezultatul este corect. Operația „+” se implementează hardware cu un sumator ca cel proiectat în laboratorul precedent.

În cazul **scăderii** $\rho = \alpha - \beta$ se poate demonstra că

$$C2(\rho) = C2(\alpha) + [C2(\beta)]^* \quad \text{echivalent cu} \quad R = A \text{ ADDU } B^*$$

Adică trebuie să adunăm reprezentarea în complement față de 2 ale lui α cu reprezentarea lui β în complement față de 2 complementată. La fel ca mai sus, adunarea se face **ca și când ar fi numere fără semn** și rezultatul este corect.

În concluzie blocul de adunare/scădere va implementa următoarele prelucrări exprimate în stil C:

```
if (S1A0==0)
    R= A+B;    //adunare
else
    R= A+B*;   // scădere
```

Complementul față de 2 al unui număr x se calculează cu formula:

$$x^* = \text{not}(x) + 1$$

Complementul față de 2 este egal cu complementul față de 1 plus 1. Complementul față de 1 se obține inversând numărul bit cu bit. Înlocuind în prelucrarea stil C de mai sus obținem:

```
if (S1A0==0)
    R=A+B;           //adunare
else
    R=A+not(B)+1;    // scădere
```

Adunam un 0 în cazul adunării pentru ca modul de implementarea sa devină mai clar:

```
if (S1A0==0)
    R=A + B + 0; //adunare
else
    R=A + not(B)+ 1; // scădere
```

Implementarea hardware a sumator/scăzătorului conform specificării de mai sus necesită:

1. Un sumator pentru numere naturale, ca cel proiectat în laboratorul precedent. Pentru fiecare bit $k=0..n-1$ este nevoie de un sumator de un bit.
2. Primul operand al sumatorului este $A=C2(\alpha)$.
3. Al doilea operand al sumatorului este fie B dacă operația este adunare, fie negatul acestuia dacă operația este scădere. Pentru fiecare rang al celui de-al doilea operand vom selecta fie b_k , fie not b_k , cu un MUX2. Pentru a genera pe not b_k este nevoie de un inversor.
4. C_i este 0L pentru adunare și 1L pentru scădere. Se observă cu ușurință că C_i este chiar S1A0.

Desfășurarea lucrării

Pasul 1: Terminarea sumatorului pe 4 biți.

În laboratorul precedent intrarea de transport a sumatorului pe 4 biți a fost legată la masă astfel încât C_i este 0L tot timpul. Implementarea sumator/scăzătorului conform secțiunii „Chestiuni teoretice” folosește C_i pentru a aduna un ,1’ în cazul scăderii.

Procedați după cum urmează:

1. Deschideți proiectul as4 din laboratorul 3 și apoi deschideți sum4.sch
2. Ștergeți simbolul GND conectat la intrarea C_i a rangului zero. Conectați la acest C_i un marker IO cu numele C_i .
3. Creați simbolul lui sum4 după procedura descrisă în laboratorul 3 (**Design Utilities → Create Schematic Symbol**).
4. Adăugați la proiect o nouă sursă de tip schematic cu numele **as4**. Pe această planșă vom desena sumator/ scăzătorul

Pasul 2: Crearea schemei sumator/scăzătorului pe 4 biți folosind magistrale.

Deschideți planșa as4. Plasați pe aceasta magistrala B, cele 4 inversoare și cele 4 MUX2. Încercați să obțineți o schemă cât mai apropiată de cea din figura 1:

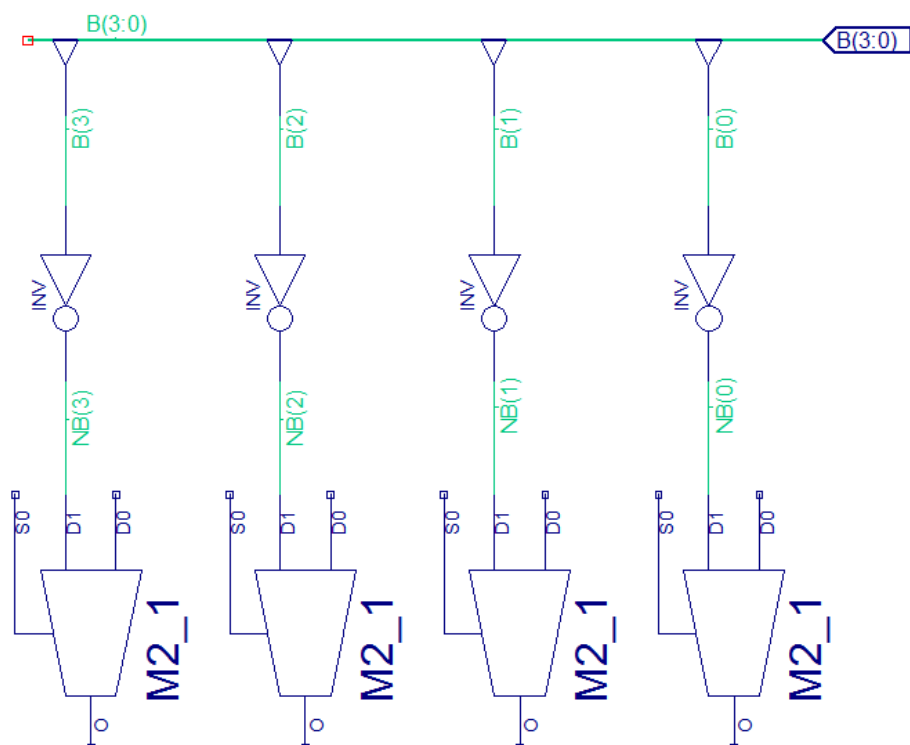


figura 1

Conectați intrările D0 ale celor 4 MUX-uri. Adăugați markerul IO S1A0. Încercați să obțineți o schemă cât mai apropiată de cea din figura 2. În această fază am obținut al doilea operand al sumatorului: **B** dacă S1A0 este 0L (aducă adunare) sau not **B** dacă S1A0 este 1L.

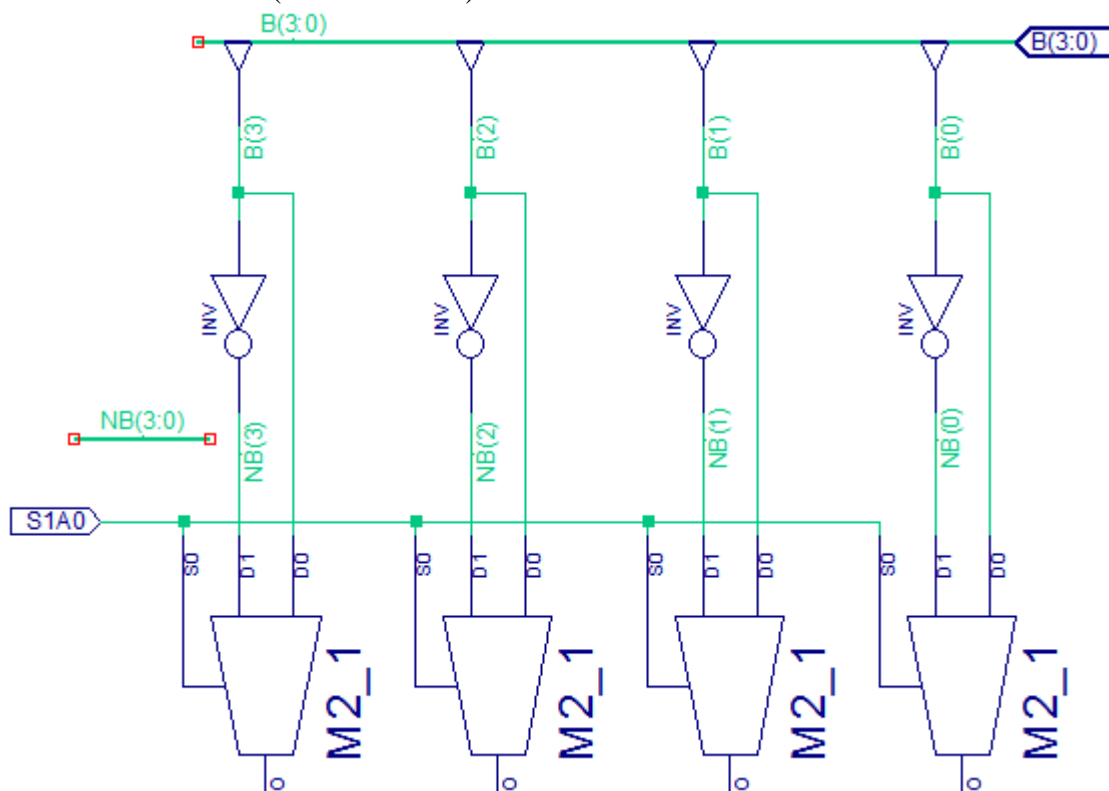


figura 2

Adăugați pe schemă sumatorul **sum4** obținut la pasul 1. La intrarea b(3:0) a lui sum4 adăugați o conexiune în T ca în figura 3. Această conexiune este de tip magistrală și are un nume dat automat de ISE. Dacă plasați cursorul deasupra acestei magistrale va apare numele acesteia, ca în figura 3. În cazul schemei dumneavoastră, numele va fi diferit.

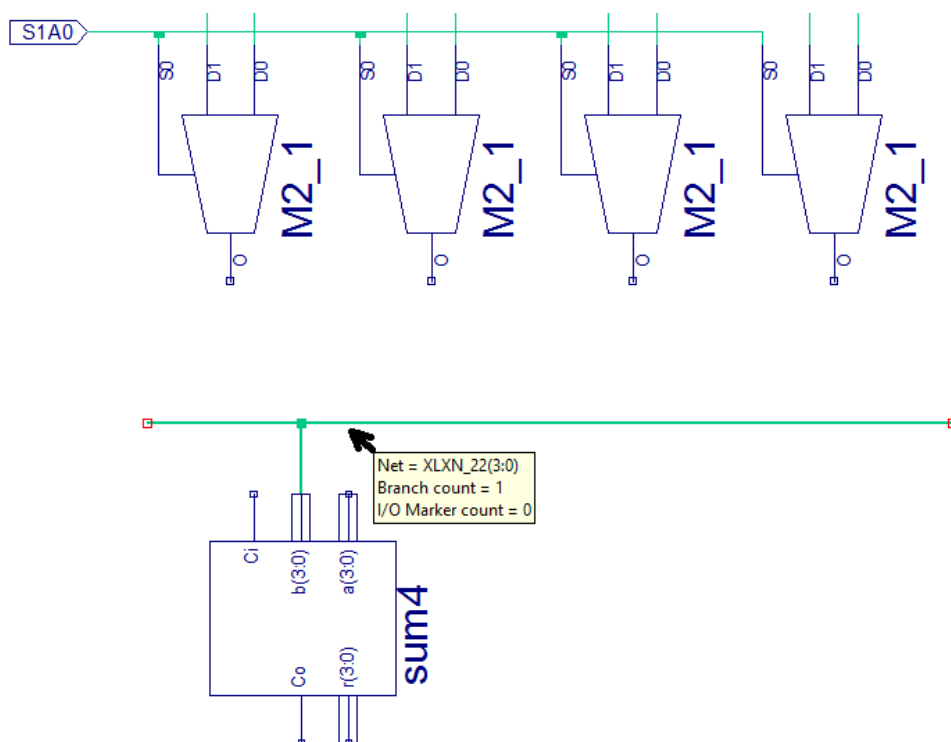


figura 3

Magistrala creată automat de ISE are aceeași dimensiune și aceleași limite stânga –dreapta ca pinul ierarhic b(3:0). Altfel spus numele magistralei se obține din numele pinului ierarhic înlocuind pe b cu XLXN_22. Schimbați numele acestei magistrale în BNB(3:0). BNB înseamnă B sau not B deoarece aici se vor conecta ieșirile celor 4 MUX2. Pentru a schimba numele magistralei adăugați apăsând butonul **Add net name** și în câmpul **Name** introduceți **BNB(3:0)**.

În continuare conectați ieșirile MUX-urilor la magistrala BNB, conectați intrarea Ci la S1A0 și adăugați markerii IO A(3:0), R(3:0) și Co. Încercați să obțineți schema din figura 4.

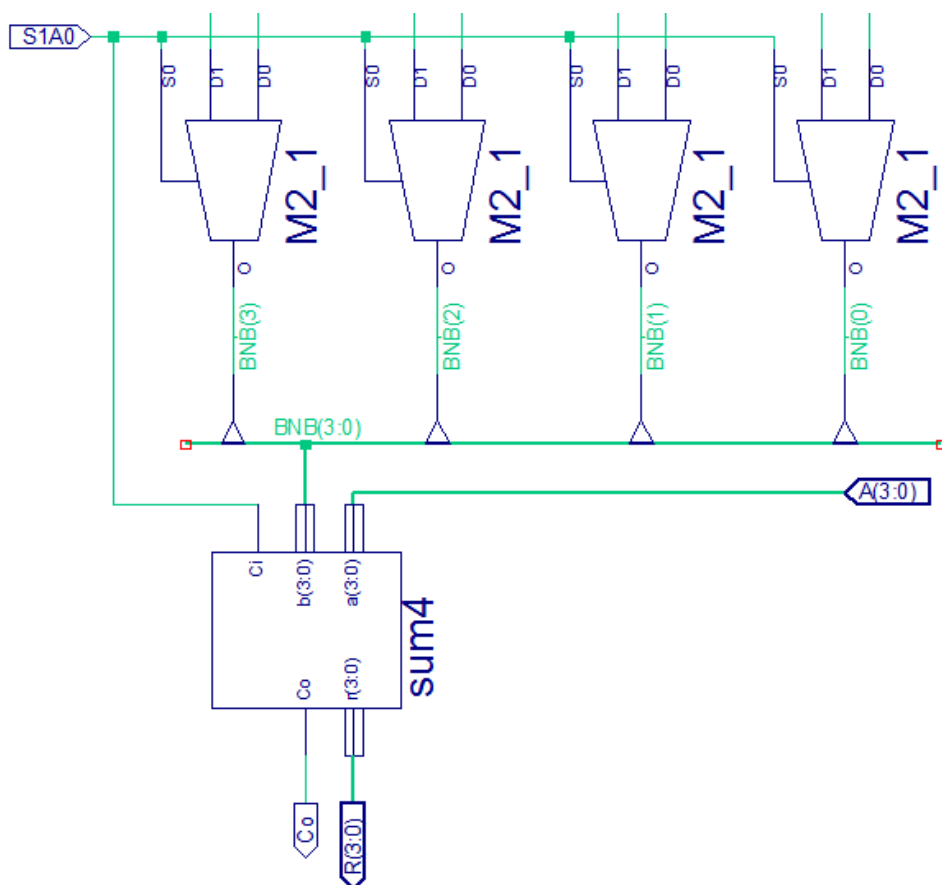


figura 4

Verificați corectitudinea schemei și salvați

Pasul 3: Simularea sumator-scăzătorului.

În cazul simplu al unui sumator pe 4 biți, funcționalitatea se poate verifica prin încărcarea proiectului în placa de dezvoltare. Dacă ceva nu funcționează corect este relativ simplu, dat fiind complexitatea scăzută a proiectului, să ne dăm seama de locul din care provine eroarea. Cum se procedează în cazul proiectelor complexe, care umplu un FPGA din seria 5000, nu din seria 400? Răspunsul este „Prin simulare”!

Pentru simularea sumatorului se vor parcurge următorii pași:

1. Prima etapă constă în crearea unui fișier scris în limbaj de nivel înalt, de regula VHDL sau Verilog, în care să se specifice modul de comportare în timp a intrărilor și în care se verifică dacă ieșirile corespunzătoare intrărilor generate sunt corecte. Acest tip de fișier se numește TEST BENCH.
2. ISE are o componentă grafică care permite generarea interactivă a unei „jumătăți” de test bench, adică numai partea de generare a intrărilor, numite și **stimuli**. Rularea acestei componente duce la crearea unui test bench scris în VHDL.
3. Pentru a crea un test bench adăugați la proiect o nouă sursă de tip **Test Bench Waveform**, cu numele **tbw** (sau altul, dacă acesta nu vă place). Adăugarea unei noi surse se face din meniul **Project → New Source...**. Ghidați-va după figura 5.
4. Apoi suntem întrebați cărui fișier sursă îi va fi asociat test bench-ul, pentru a ști ce stimuli trebuie generați. Vom asocia test bench-ul modulului din vârful ierarhiei, și anume as4, ca în figura 6.

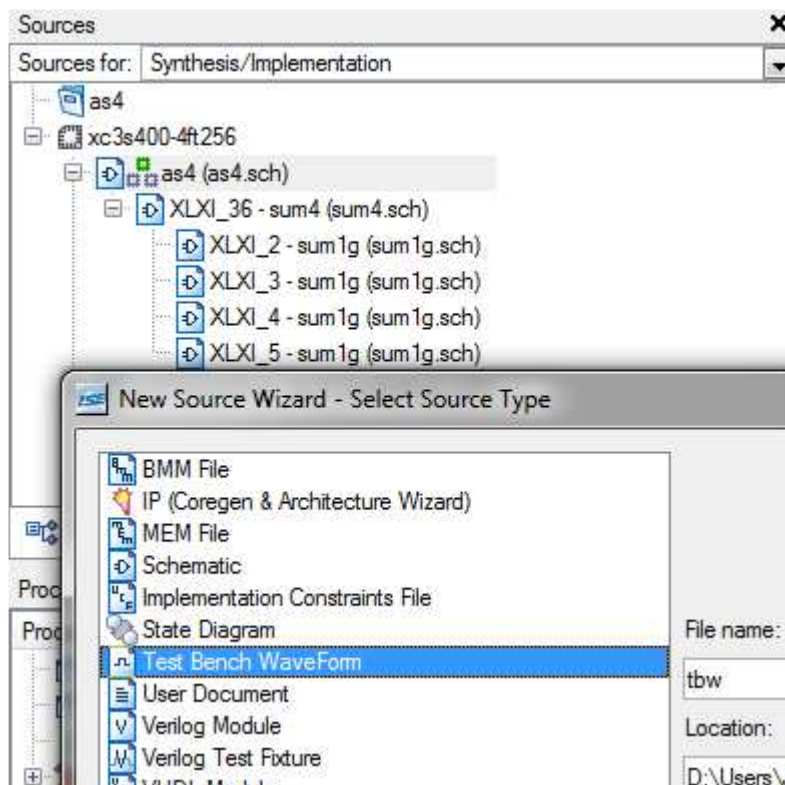


figura 5

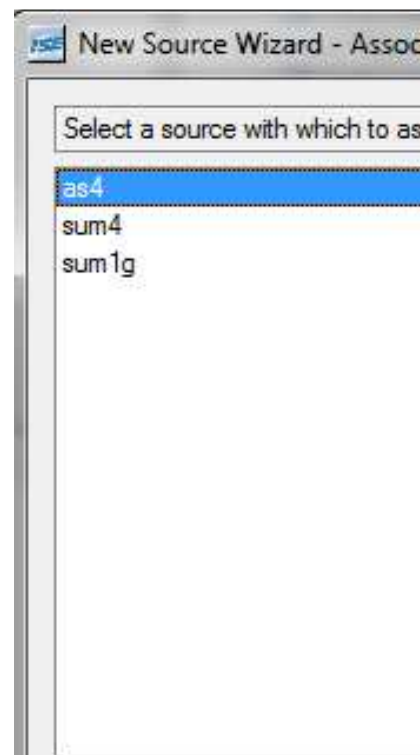


figura 6

5. În continuare ISE propune anumiți parametrii pentru generarea stimulilor. Selectați opțiunea **Combinatorial** dacă aceasta nu este deja selectată. Trebuie să obțineți situația din figura 7.

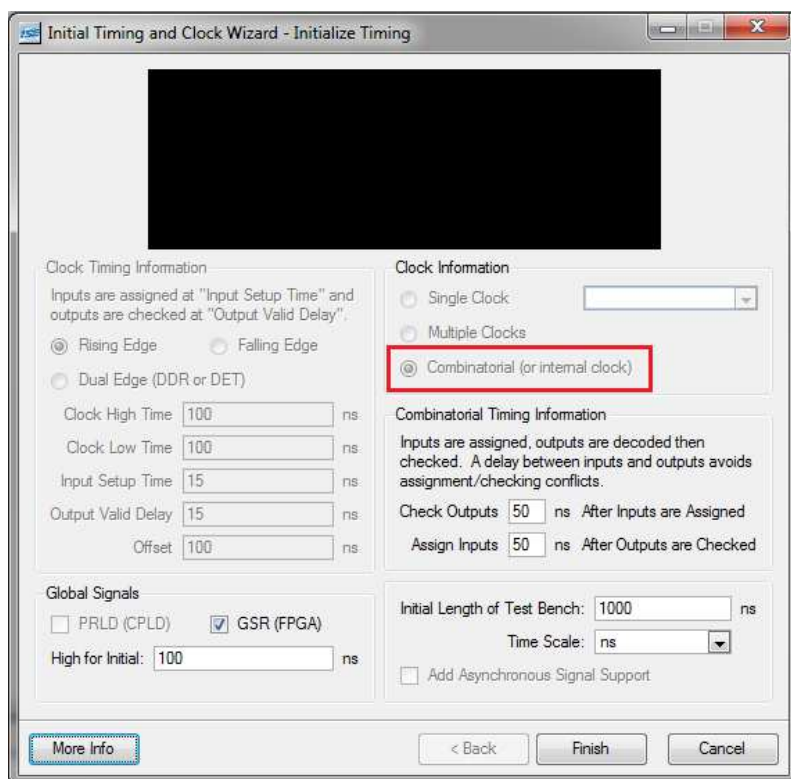


figura 7

6. Acceptați restul stărilor apăsând butonul **Finish**. Veți obține situația din figura 8 sau ceva foarte asemănător

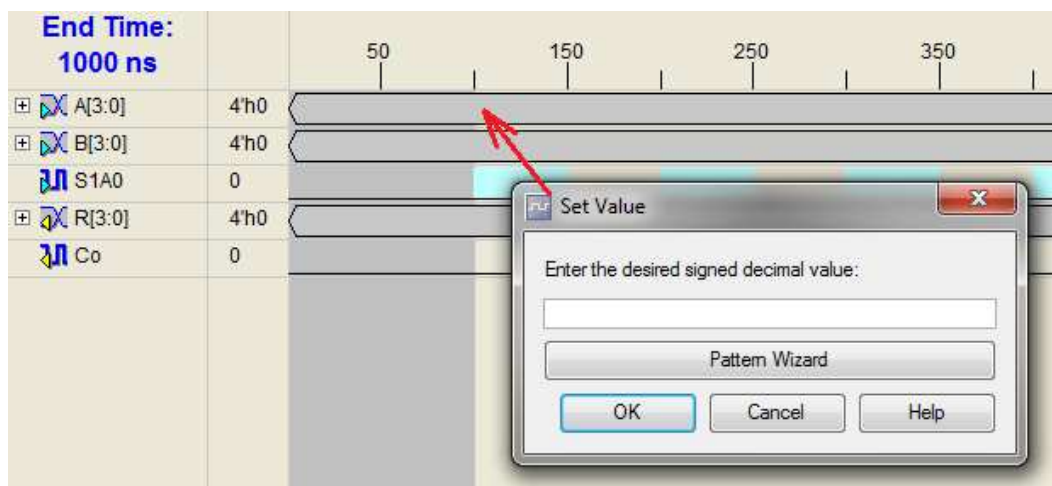


figura 8

Dacă semnalele nu sunt ordonate ca în figura 8 puteți să la reordonați trăgând de ele în poziția dorită. Nu este obligatoriu.

Comportarea în timp a unei magistrale se poate defini per componentă sau global. Pentru definirea per componentă se expandează magistrala prin apăsarea semnului „+” adiacent numelui. Ca urmare a acestei operații se vor afișa membrii magistralei și aceștia pot fi modificați individual. **Nu se va folosi această metodă!!**

Pentru definirea globală a valorii unei magistrale la un moment de timp se face clic în locul în care se dorește schimbarea valorii. De exemplu, pentru a modifica valoarea lui **A(3:0)** la 100 ns faceți clic în zona indicată de săgeata roșie din figura 8. Ca urmare va apare fereastra de dialog **Set Value**.

Atenție: în zona de la 0 la 100ns este perioada de reset. În acest interval valorile semnalelor nu se pot schimba!

În fereastra **Set Value** introduceți valoarea 5.

Pentru a testa complet funcționarea sumator/scăzătorului ar trebui ca **A(3:0)**, **B(3:0)** și S1A0 să ia toate valorile posibile. Deoarece sunt 9 biți în total numărul de combinații posibile este 512.

Operație (0→+, 1→ -)	A(3:0)	B(3:0)
+	+5	+2
+	-5	+2
+	+5	-2
+	-5	-2
-	+5	+2
-	-5	+2
-	+5	-2
-	-5	-2

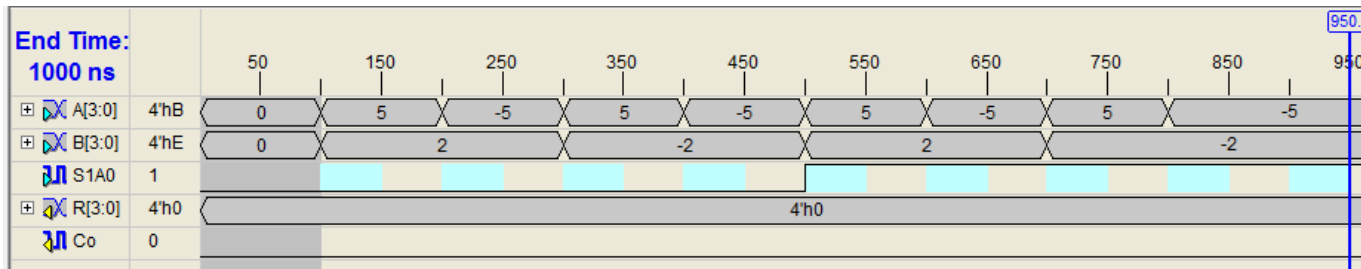
Specificarea tuturor combinațiilor nu este greu de făcut deoarece editorul grafic de stimuli are suficiente capabilități în acest sens.

Partea dificilă constă în verificarea corectitudinii ieșirii **R(3:0)**. Dacă verificarea pentru un set de valori de intrare, de exemplu 5+(-3), ia 3 secunde, ar fi nevoie în total de 512*3 secunde, adică aproximativ 30 de minute. Pe lângă timpul lung necesar verificării există șanse mari ca să trecem cu vederea o eventuală eroare. Motivele expuse mai sus fac ca verificarea funcționalității să se facă automat, în acest sens scriindu-se cod HDL. Deoarece nu aveți suficiente cunoștințe în acest moment, se va face doar o verificare preliminară a funcționării sumator/scăzătorului.

Verificarea se va face pentru 8 valori de intrare, conform tabelului alăturat.

Bug: introduceți echivalentul hexazecimal al combinațiilor din tabelul alăturat conform reprezentării complementului față de 2 din figura 15. Specificarea decimal signed care ar fi fost ideală în acest caz nu funcționează corect!

După introducerea valorilor din tabel **trebuie** să obțineți situația din figura 9:

**figura 9**

7. Există mai multe opțiuni de simulare dintre care două sunt de interes: **Behavioural** și **Post Route**. Simularea de tip Behavioural este cea mai rapidă dar toate întârzierile sunt zero. Simularea Post Route este cea mai exactă, toate întârzierile sunt vizibile, dar durează foarte mult. În laboratoarele următoare vom alege simularea Behavioural pentru a câștiga timp dar în acest laborator **vom alege Post Route** pentru a vedea modul real de funcționarea proiectului. Pentru aceasta în fereastra **Sources**, lista **Sources for**, selectați opțiunea **Post-Route Simulation**, ca în figura 10. Ca urmare conținutul ferestrelor **Sources** și **Processes** se modifică pentru simulare.
8. În fereastra **Processes** **selectați fișierul tbw** și apoi lansați în execuție procesul **Simulate Post-Place&..**, conform figura 11. Aceasta va duce la lansarea simulatorului **Modelsim**. Va dura cam un minut.

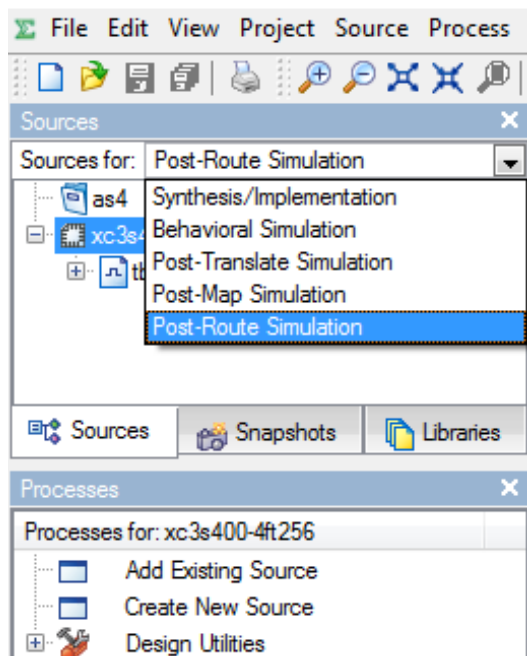


figura 10

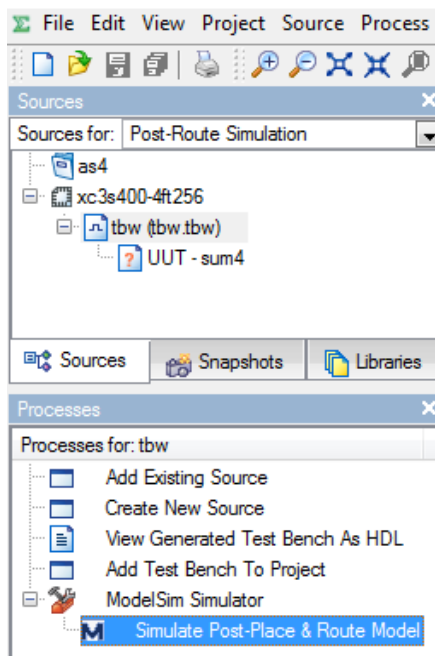


figura 11

9. Fereastra simulatorului este alcătuită din mai multe subferestre. Identificați fereastra **Wave**. În această fereastră se afișează evoluția în timp a tuturor semnalelor din proiect. Ea joacă rolul ferestrei „Watches” din depanarea software. Detașați fereastra Wave (undock) și maximizați-o. Apoi, prin drag and drop, mișcați forma de undă Co în ultima poziție. Ar trebui să obțineți o situație **asemănătoare** cu cea din figura 12.
10. **Pentru a afișarea cu semn** a valorilor semnalelor selectați toate semnalele ce se vizualizează, faceți clic dreapta, din meniul contextual ce va apare selectați **Radix** și din noul meniu contextual selectați **Decimal**, ca în figura 12.

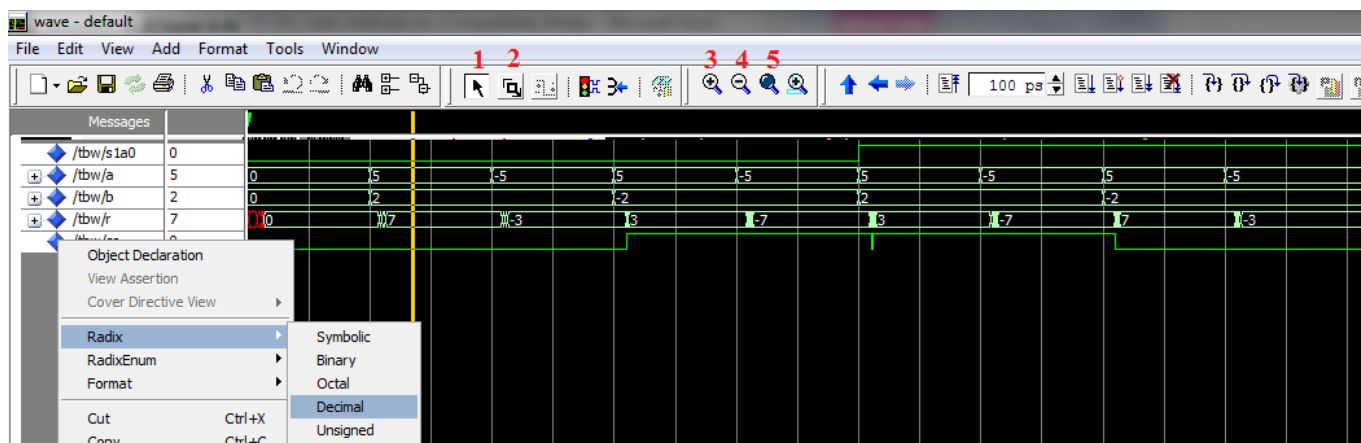



figura 12

Rolul butoanelor marcate cu 1-5 în figura 12 este:

- 1 – **Selecție**. Pentru selectarea unei forme de undă și pentru manevrarea cursorilor.
- 2 – **Zoom mode**. Pentru a mări o zonă selectată.
- 3,4 – Este evident.
- 5 – **Zoom full**. Afișează toată simularea.

Mutați cursorul pe toate valorile de intrare și verificați dacă valoarea ieșirii este corectă.

Pentru a evidenția întârzierile se procedează după cum urmează:

1. Mai întâi apăsați **lupa neagră** (Zoom full) pentru a afișa rezultatul întregii simulări într-un singur ecran. Trebuie să obțineți situația din figura 12. În general **Zoom full** este prima acțiune executată după terminarea simulării.
2. Se expandează magistrala **r**. Pentru a expanda un bus apăsați semnul + din dreptul numelui respectivului bus.
3. Se apasă **Zoom mode** (). Remarcați schimbarea formei cursorului.
4. Se poziționează cursorul înaintea tranziției din 0 în 1 a lui S1A0.
5. Se apasă butonul stânga al mouse-ului și, cu butonul apăsat, se mișcă cursorul pentru a încadra tranzițiile lui **a**, **b** și a lui S1A0, ca în figura 13. După ce se eliberează butonul mouse-ului ar trebui să se obțină situația din figura 14.

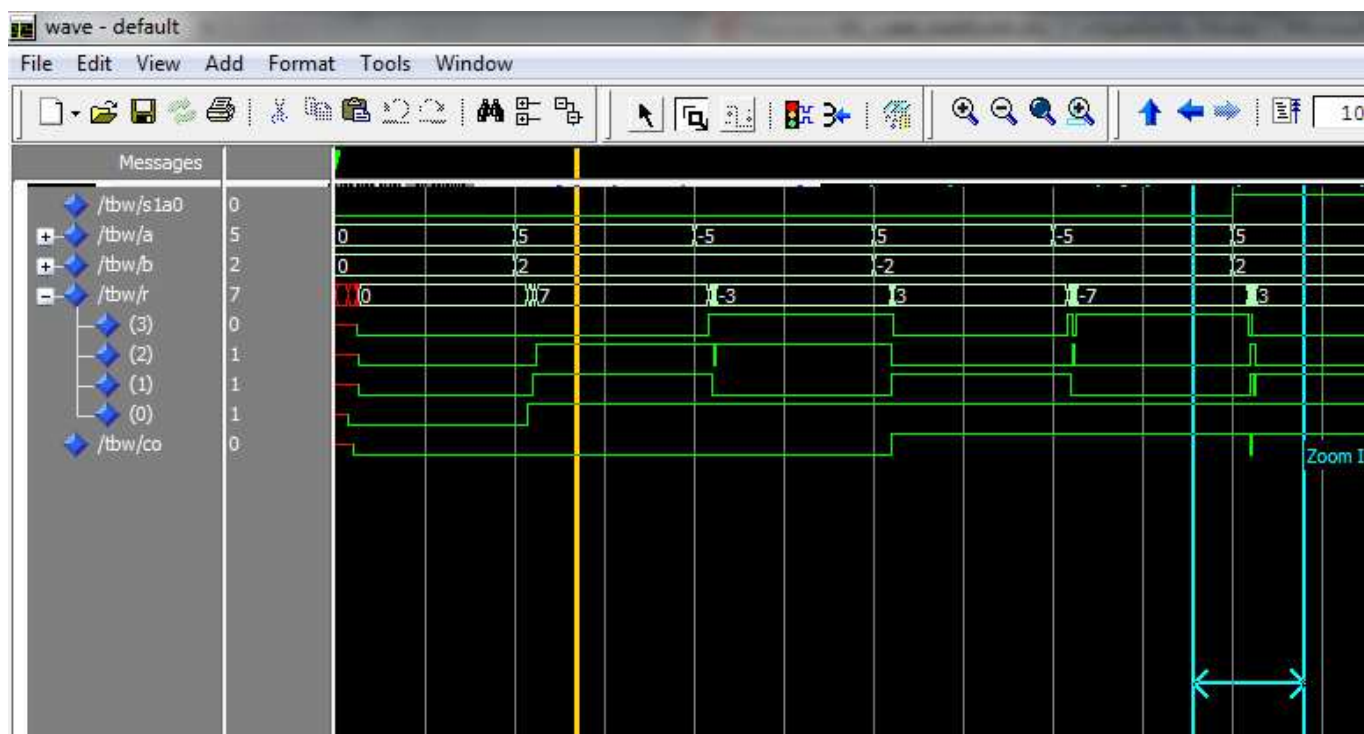


figura 13

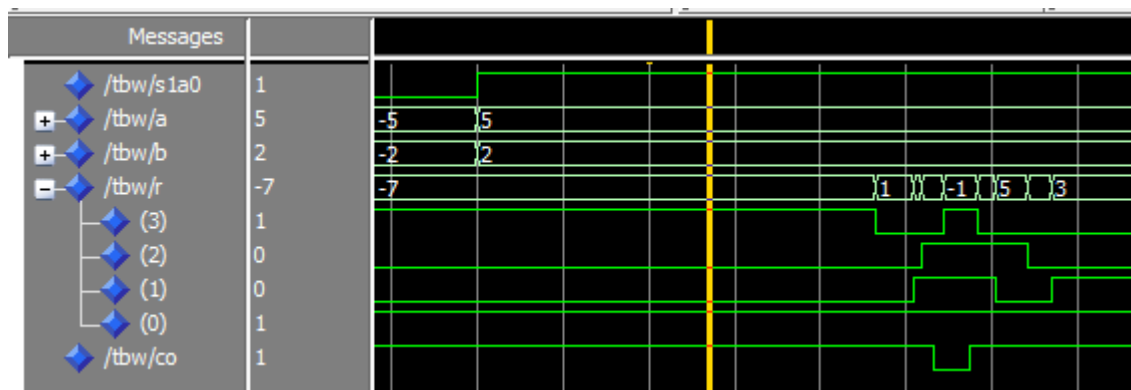


figura 14

Dacă rezultatele obținute de dumneavoastră sunt la fel ca cele din figura 12 și figura 14, chemați profesorul pentru **validare**.

Dacă ați ajuns aici aveți nota 5.

Continuați pentru notă mai mare. (opțional)

Pasul 4: Depășirea formatului - Overflow. (opțional)

În schema sumatorului/scăzător pe 4 biți semnalul Co poate fi folosit pentru a semnaliza depășirea formatului de reprezentare dacă operandii sunt priviți ca numere pozitive în gama 0 - 15. În schimb, dacă operandii sunt priviți ca numere cu semn reprezentate în complement față de 2 în gama (-8) ...(+7), atunci Co nu poate fi folosit pentru a semnaliza depășirea. De exemplu adunăm 5+6 și obținem: 0101+0110=1011. Adunăm două numere pozitive și obținem un număr negativ iar Co este 0!

Pentru a semnaliza corect depășirea în cazul numerelor cu semn se analizează **semnul lui A, semnul lui B, semnul rezultatului și tipul operației** (adunare sau scădere).

Se cere:

1. Să se scrie tabelul de adevăr pentru funcția depășire, funcție ce depinde de variabilele enumerate mai sus.
2. Să se implementeze această funcție.

Operație (0 → +, 1 → -)	A(3:0)	B(3:0)
+	+5	+4
+	-5	+4
+	+5	-2
+	-5	-4
-	+5	+2
-	-5	+4
-	+5	-4
-	-5	-2

Se va testa cu valorile din tabela alăturată.

Veți obține între 0 și 5 puncte.

Anexa 1 – Complementul față de 2

Dacă $X = x_{n-1} \dots x_1 x_0$ este un **număr binar pozitiv** reprezentat pe n biți complementul său față de 2 se notează X^* și se calculează ca $X^* = 2^n - X$. **Operația $2^n - X$ se numește complementare.**

Fie i un număr **întreg** ce aparține intervalului $[-2^{n-1}, 2^{n-1})$. De exemplu dacă $n=4$, $i \in [-8, 8)$. Reprezentarea în cod complementar (complement față de 2) a numărului i se notează cu $C2(i)$ și se calculează după următoarea regulă:

$$C2(i) = \begin{cases} |i|, & i \geq 0 \\ |i|^* = 2^n - |i| & i < 0 \end{cases}$$

Pentru $n=4$ numerele $i \in [-8, 7]$ se reprezintă conform figurii următoare:

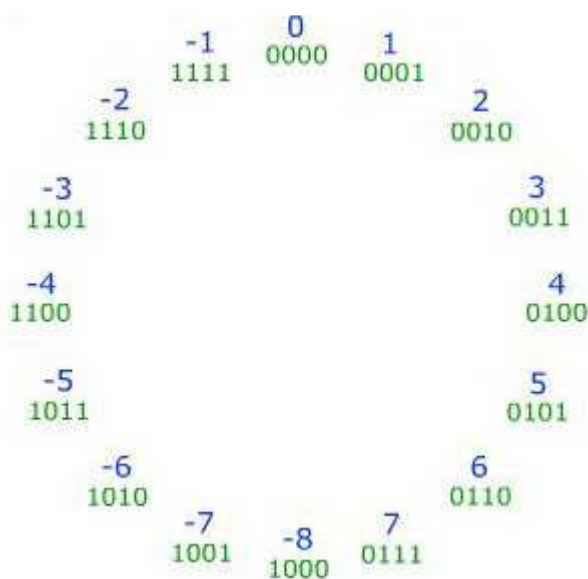


figura 15

Codul complementar are o serie de proprietăți remarcabile:

- Numerele pozitive au bitul cel mai semnificativ 0 iar cele negative, 1 (vezi figura).** Bitul cel mai semnificativ se numește bit de semn.
- Așa cum am construit sumatorul pe 4 biți, să presupunem că am construi un scăzător pe 4 biți prin cascada a 4 scăzătoare de un bit. Dacă un scăzător pe 4 biți are de efectuat $5-2$ rezultatul va fi:

$$\begin{array}{r} 0101- \\ 0010 \\ \hline 0011, \text{ adică } 3. \end{array}$$

Care este rezultatul operației $2-5$? Conform regulilor învățate la clasa a VI-a, mai întâi ar trebui comparate modulele celor două numere, apoi se scade modulul cel mai mic din modulul cel mai mare și se dă semnul din față operandului care modulul mai mare. Adică se face $5-2=3$ și apoi semnul lui 3 se schimbă.

Dacă nu se ține seama de regula de la clasa a VI-a și scădem pe 5 din 2, obținem:

$$\begin{array}{r} 1 \\ 0010- \\ 0101 \\ \hline 1101, \text{ adică } C2(-3) \text{ din figura 15 și un împrumut care se va ignora.} \end{array}$$

Scăderea fără compararea modulelor are nevoie doar de scăzător, dar rezultatul apare în complement față de 2. Dacă am vrea să implementăm scăderea ca la clasa a VI-a, pe lângă scăzător am mai avea nevoie de un comparator și de două multiplexoare vectoriale.

- Nu este nevoie de un sumator special pentru adunarea numerelor reprezentate în cod complementar.** Adunarea numerelor reprezentate în cod complementar se face cu același hardware ca cel folosit pentru adunarea numerelor naturale. Rezultatul va apare însă în cod complementar.

ATENȚIE: Bitul de semn participă la calcule!

Un prim exemplu: $5+(-3)$, $C2(5)=0101$, $C2(-3)=1101$

$$\begin{array}{r} 0101+ \\ 1101 \\ \hline \end{array}$$

10010, adică 2 din figura 15 și un transport care se va ignora.

Un al doilea exemplu: $(-4) + (-3)$, $C2(-4) = 1100$, $C2(-3)=1101$

$$\begin{array}{r} 1100+ \\ 1101 \\ \hline \end{array}$$

11001, adică -7 din figura 15 și un transport care se va ignora.

4. **Nu este nevoie de scăzător!** În loc de scădere se adună scăzătorului cu semn schimbat și rezultatul este întotdeauna corect. Nu contează dacă scăzătorul este pozitiv sau negativ!

Fie α și β două numere întregi. Operația $\alpha - \beta$ se poate efectua ca $\alpha + (-\beta)$. Singura problemă este că nu dispunem de numerele α și β , ci de reprezentarea lor în cod complementar, adică de $C2(\alpha)$ și de $C2(\beta)$.

Operația $\alpha - \beta = \alpha + (-\beta)$ va executată de hardware ca $C2(\alpha)+C2(-\beta)$

Se poate arăta imediat că dacă $C2(\beta)$ este reprezentarea în cod complementar a lui β , atunci:

$$C2(-\beta)=2^n - C2(\beta)$$

Adică $C2(-\beta)$ se obține prin complementarea lui $C2(\beta)$!

Un prim exemplu: $5-(-2)$, $C2(5) = 0101$, $C2(2) = 0010$, $C2(-2) =$

$$=10^4 - 0010=10000-0010=1110$$

$$\begin{array}{r} 0101+ \\ 1110 \\ \hline \end{array}$$

10011, adică $C2(3)$ și un transport care se va ignora.

Un al doilea exemplu:

$$(-4) - (-3), C2(-4) = 1100, C2(-3) = 1101, C2(-(-3)) = 10^4-1101=10000-1101=0011$$

$$\begin{array}{r} 1100+ \\ 0011 \\ \hline \end{array}$$

1111, adică $C2(-1)$.

Chiar dacă în loc de a scădea am adunat pe $C2(-\beta)$, tot nu am scăpat de scădere. Termenul $C2(-\beta)$ se obține efectuând o scădere, și anume $2^n-C2(\beta)$. Dacă nu scăpăm de ea ideea prezentată mai sus nu are nici o valoare practică (din punct de vedere al implementării). Așa cum vom vedea în continuare, nu este nevoie să se efectueze o scădere pentru a obține complementul unui număr.

Complementarea se poate face prin 3 metode. Metoda de interes din punct de vedere al implementării este cea prin intermediul complementului față de 1. Mai exact, cantitatea care trebuie complementată se inversează bit cu bit și în final se adună un 1. De exemplu complementul lui 0010 este $1101+1=1110$. Complementul față de 1 se notează ca operația de inversare : complementul față de 1 al lui B este \overline{B} .

Marele avantaj al acestei metode este că scăderea se transformă în inversarea bit cu bit a lui $C2(B)$, plus adunarea unei unități. Inversarea necesită un LUT per rang pe când scăderea necesită două LUT-uri per rang. Economie de un LUT per rang!

Ultima problemă constă în adunarea unei unități. Această operație se poate face prin intermediul intrării Cin a rangului 0. Această intrare este conectată la masă (0L) la sumatorul pe 4 biți proiectat anterior.

În concluzie, dacă avem de adunat două numere α și β reprezentate în cod complementar ca $C2(\alpha)$ și $C2(\beta)$, suma, reprezentată tot în cod complementar, se obține ca:

$$C2(\sigma) = C2(\alpha) + C2(\beta) \quad (1)$$

Dacă avem de efectuat A-B diferență D se obține ca:

$$C2(\delta) = C2(\alpha) + \overline{C2(\beta)} + 1 \quad (2)$$

Adunarea se face cu un sumator pentru numere naturale. Bitul de semn participă la adunare.