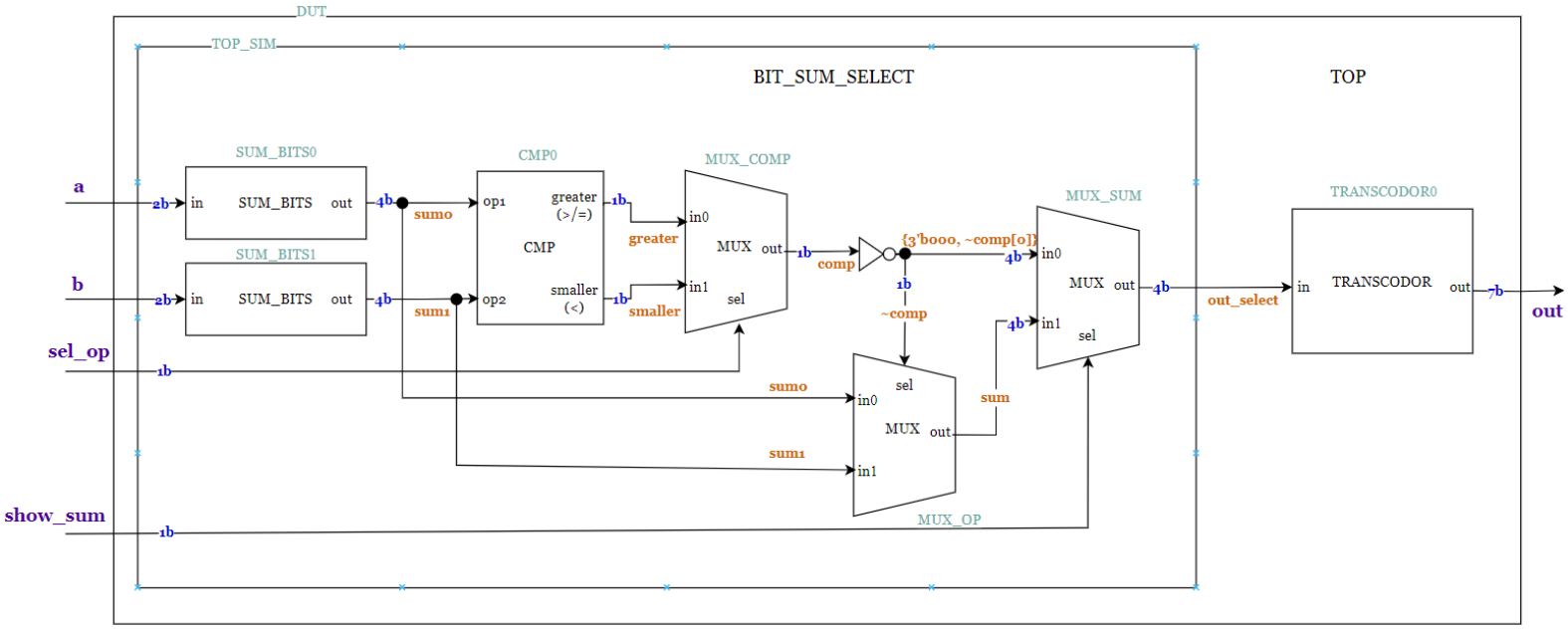
Test 1 CID

Se cere implementarea schemei următoare în Verilog. Efectuați simulări pentru verificarea funcționalității acesteia și sintetizați circuitul pe placa FPGA – Boolean Board.



Circuitul primește două numere pe 2 biți, **a** și **b**, și doi biți independenți de selecție, **sel_op** și **show_sum**.

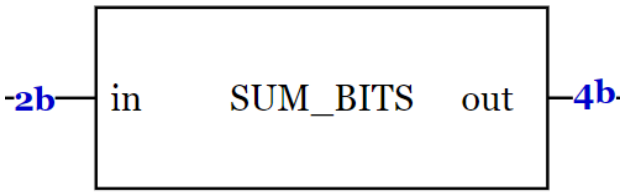
Dacă sel_op este 0, se alege cea mai mare sumă a biților, deci ieșirea lui MUX_COMP va furniza 1 în cazul în care suma biților lui a este mai mare decât suma biților lui b și viceversa. Dacă sel_op este 0, se alege cea mai mică sumă a biților, deci ieșirea lui MUX_COMP va furniza 1 în cazul în care suma biților lui a este mai mică decât suma biților lui b și viceversa.

Dacă show_sum este 0, TOP va furniza la ieșirea sa intrarea, codată pe 7 biți, care are suma biților mai mare/mai mică, în funcție de bitul de selecție sel_op ales. Dacă show_sum este 1, TOP va furniza la ieșirea sa suma biților a intrării corespunzătoare (care are suma biților mai mică/mai mare, în funcție de sel_op).

Module

• SUM_BITS

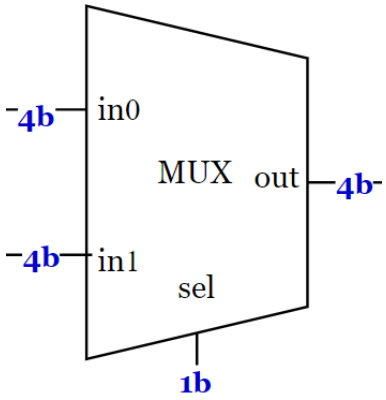
- o intrare pe 2 biți, **in**
- o ieșire pe 4 biți, **out**



SUM_BITS oferă la ieșire suma biților intrării.

Bonus 1p: parametrizați numărul de biți al intrării și folosiți un for pentru calcularea sumei.

• MUX



- trei intrări: - două intrări pe câte 1 **sau** 4 biți, **in0** și **in1**

  - o intrare pe un singur bit, **sel**

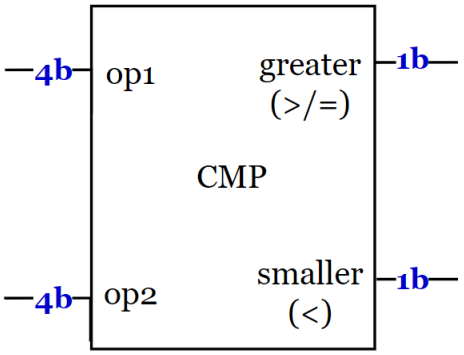
- o singură ieșire pe 1 **sau** 4 biți, out

Multiplexorul furnizează la ieșire una dintre cele două intrări, în funcție de valoarea semnalului de selecție.

Se dorește calcularea output-ului folosind *assign*.

Observație: în BIT_SUM_SELECT, regăsiți atât MUX-uri cu intrări și ieșiri pe 4 biți, cât și pe 1 bit. Puteți parametriza modulu sau puteți crea două module, MUX4 și MUX1 (sau puteți folosi doar un MUX4 și prelucra intrarea în MUX_SUM după cum se vede pe schemă).

• CMP

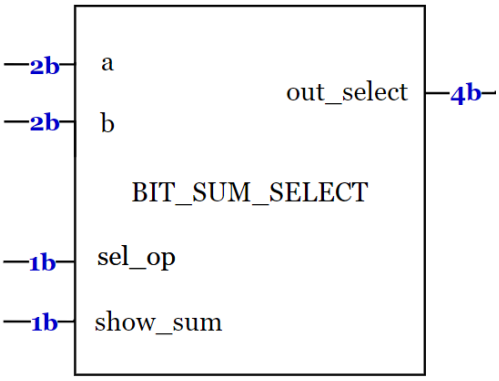


- două intrări pe câte 4 biți, **op1** și **op2**
- două ieșiri pe câte 1 bit, **greater** și **smaller**

Comparatorul oferă, pe ieșirea greater, valoarea de adevăr a expresiei (op1>=op2), iar pe ieșirea smaller, valoarea de adevăr a expresiei (op1<op2).

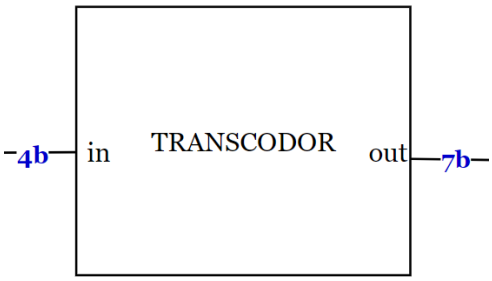
Se dorește implementarea acestuia folosind un bloc de tip *always*.

• BIT_SUM_SELECT



- 4 intrări: - două intrări pe câte 2 biți, **a** și **b**
  - două întrări pe câte un bit, **sel_op** și **show_sum**
- o ieșire pe 4 biți, **out_select**, care frunizează la ieșire fie indexul intrării care are suma biților “câștigătoare”, unde a are indexul 0 și b are indexul 1, fie suma biților acelei intrări, în funcție de sel_op și show_sum

• TRANSCODOR



- o intrare pe 4 biți, **in**
- o ieșire pe 7 biți, **out**

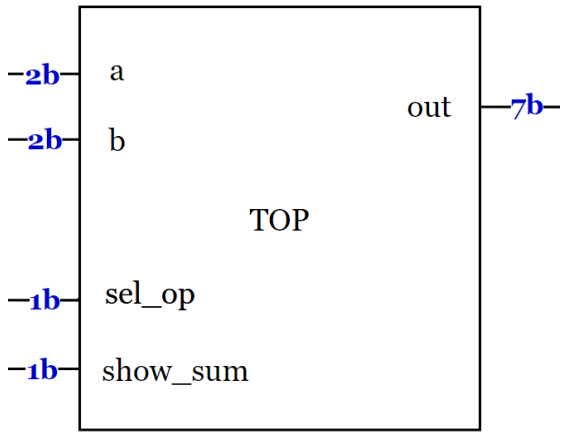
Transcodorul va genera la ieșire codul corespunzător intrării pentru afișajul pe 7 biți. **Default value: E**. Se va consulta anexa pentru afișaj.

Bonus 1p: modificați trancodorul astfel încât, dacă show_sum este 0, deci vom vedea la ieșire una dintre intrări (0 pentru a și 1 pentru b), ieșirea transcodorului reprezintă codurile specifice lui A și b în loc de 1 sau 0. Așadar, TRANSCODOR va avea o intrare în plus, show_sum, care vine de la intrarea modulului TOP.

• TOP

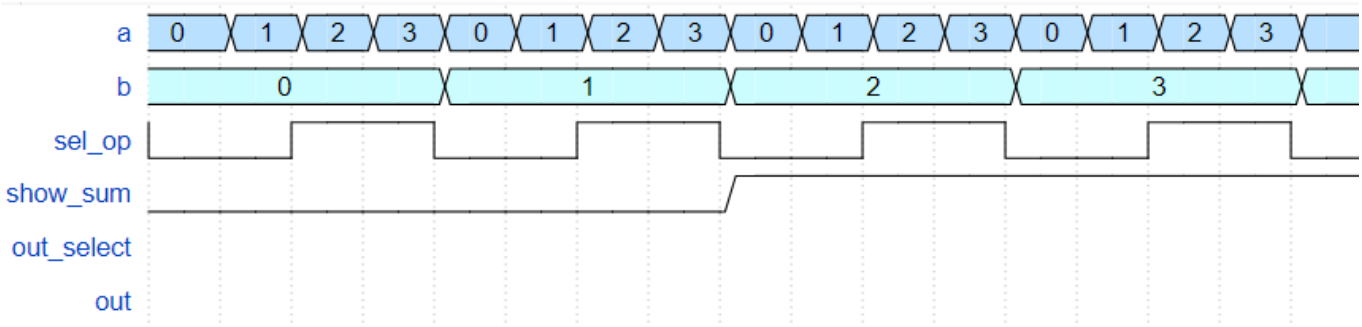
- 4 intrări: - două intrări pe câte 2 biți, **a** și **b**
  - două întrări pe câte un bit, **sel_op** și **show_sum**
- o ieșire pe 7 biți, **out**

TOP cuprinde BIT_SUM_SELECT și TRANSCODOR, furnizând la ieșire codul pentru afișajul cu 7 segmente corespunzător ieșirii modulului BIT_SUM_SELECT.



Testbench

În fișierul testbench.v (sau tb.v), instanțiați modulul TOP și generați următoarele forme de undă pentru intrările circuitului:



Fiecare linie punctată verticală semnifică trecerea a 4 nanosecunde.

Precum este figurat și mai sus, se dorește vizualizarea și vizualizarea firului intern *out_select*.

Pentru programarea plăcii FPGA:

Să se asocieze:

• bitul de selecție *sel_op* cu switch-ul cu numărul 15 (SW15 - ultimul de la dreapta la stânga de pe placă)

• bitul *show_sum* cu switch-ul cu numărul 14

• biții lui *a* cu switch-urile [1:0]

• biții lui *b* cu switch-urile [3:2]

• biții lui out_7seg cu cele 7 segmente din afișaj

Pentru a realiza aceste conexiuni, puteți crea un fișier de constrângeri sau puteți lega pinii direct în meniul I/O output.

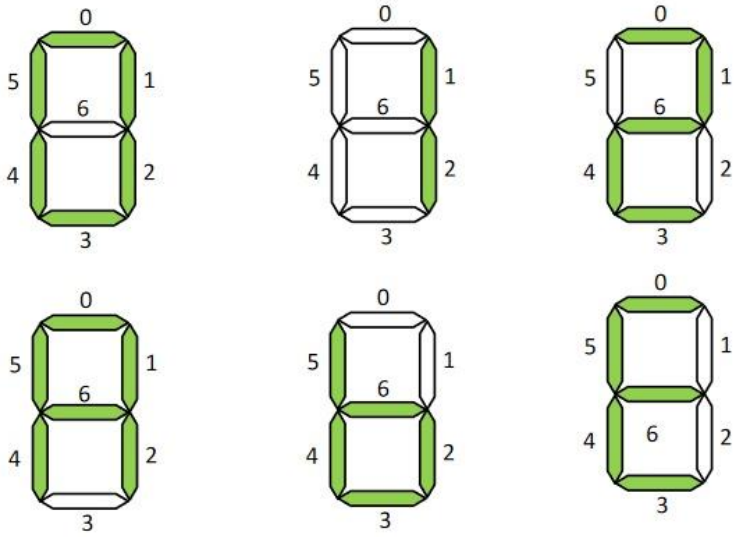
Pentru fișierul de constrângeri, folosiți sintaxa:

set_property -dict { PACKAGE_PIN PIN_NAME IOSTANDARD LVCMOS33 } [get_ports { BIT_TO_ASSIGN }];

Unde PIN_NAME este pinul de pe Boolean Board, iar BIT_TO_ASSIGN este bitul pe care vreți să îl atribuiți unui pin sau LED.

Pentru asocierea directă a biților cu pinii corespunzători, se va selecta "LVCMOS33" ca standard de I/O, unde "33" vine de la 3.3V.

Regăsiți în continuare și în anexă pinii asociați componentelor de care aveți nevoie.



Componenta	Pin
SW0	V2
SW1	U2
SW2	U1
SW3	T2
SW14	K2
SW15	K1

Componenta	Pin
Segmentul 0	D7
Segmentul 1	C5
Segmentul 2	A5
Segmentul 3	B7
Segmentul 4	A7
Segmentul 5	D6
Segmentul 6	B5

Arhivă

Arhiva pe care o încărcați pe moodle trebuie să conțină:

- fișierele .v care conțin descrierile tuturor modulelor (SUM_BITS MUX, CMP, BIT_SUM_SELECT, TRANSCODOR, TOP)

- fișierul .v cu numele testbench.v sau tb.v

- screenshot **complet și sugestiv** al formei de undă indicată la paragraful ***Testbench***

- fișierul cu constrângeri .xdc **sau** screenshot cu panoul I/O Pinout în care ați asociat pinii

Timp de lucru: 1h 40m

Mult succes!

Barem

Design: **13p**

- SUM_BITS – 1.5p

- MUX – 1.5 p

- CMP – 1.5p

- BIT_SUM_SELECT – 3.5p, dintre care 1p pentru interfață, 1.5p pentru insanțieri, 1p pentru fire și conexiuni

- TRANSCODOR – 2p

- TOP – 3p

Simulare: **4p**

- testbench – 2.5p

- simulare funcțională și poză clară a formelor de undă – 1.5p

Programare placă: **3p**

- fișierul de constrângeri/screenshot cu alocarea pinilor – 1p

- demonstrare funcționalitate – 2p