МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №1

з дисципліни «Технології проектування
комп'ютерних систем»
на тему: «Арифметико-логічний пристрій»

Виконав:
студент 4-го курсу
факультету ІОТ
групи ІО-41
Демчик В. В.
НЗК 4111

Перевірив:
проф. Сергієнко А. М.

Київ 2017

Тема: Арифметико-логічний пристрій.

Мета та основні завдання роботи: Оволодіти знаннями та практичними навичками з проектування арифметико-логічних пристроїв (LSM) для сучасних комп'ютерів. Лабораторна також слугує для оволодіння навиками програмування та відлагодження опису логічних схем на мові VHDL.

Завдання на лабораторну роботу: розробити LSM для 12-розрядних операндів, який виконує над ними наступні дії:

1. Якщо F=0 то $Y=A+B+C_0$, де C_0 – біт переносу в молодший розряд.
2. Якщо F=1 то Y=min(A,B)
3. Додатково контролювати наступні біти:

   N-знак результату,

   C3 – ознаку переносу,

   Z – ознаку нульового результату.

Виконати описання поведінкової і структурної (на LUT-елементах) моделей, тестувального стенду. Провести аналіз отриманих графіків роботи схем.

Хід проектування:

Загальний алгоритм містить наступну послідовність дій:

1. Перевіряємо біт F, якщо він = 1 то змінюємо знак операнду B.
2. Перевіряємо знак вхідних операндів A і B, якщо він = 1(“-”), то виконуємо переведення відповідного операнду в доповняльний код.
3. Виконуємо операцію підсумовування.
4. Якщо результат на суматорі має від’ємний знак то виконуємо його переведення в прямий код.
5. Перевіряємо біт F, якщо він = 1 то на мультиплексор, в залежності від знаку результату на суматорі, подаємо початкове значення мінімального вхідного операнду. Якщо F = 0 то на мультиплексор подаємо результат додавання.
6. Перевіряємо побітово результат на рівність нулеві.
7. Додатково з результату виписуємо контрольні біти.

Таблиці істинності розроблених LUT-елементів:

1. Інвертор знаку B (XOR)

```
--if F=1 then reverse sign B
LSM_REVERSE:LUT4 generic map(mask=>X"0006")
port map(a=>B(11), b=>F,c=>'0',d=> '0', Y =>buffer1);
```

F	B(11)	Y
0	0	0
0	1	1
1	0	1
1	1	0

2. Інвертор бітів операндів (XOR)

```
--Two's complement code
LSM_TCC:for i in 0 to 10 generate
 --invertor
 INVERSION_A:LUT4 generic map(mask=>X"0006") --if sign "-"(1) then inv.
 port map(a=>A(i), b=>a_buf(11),c=>'0',d=>'0', Y =>a_buf(i));
 INVERSION_B:LUT4 generic map(mask=>X"0006") --if sign "-"(1) then inv.
 port map(a=>B(i), b=>b_buf(11),c=>'0',d=>'0', Y =>b_buf(i));
end generate;
```

A(11)	A(i)	Ydk
0	0	0
0	1	1
1	0	1
1	1	0

3. Доповнювач до двух (XOR)

```
--add 1
ADD1_A:LUT4 generic map(mask=>X"0006")
port map(a=>a_buf(0), b=>a_buf(11),c=>'0',d=>'0', Y =>adk(0));
ADD1_B:LUT4 generic map(mask=>X"0006")
port map(a=>b_buf(0), b=>b_buf(11),c=>'0',d=>'0', Y =>bdk(0));
```

A(11)	A(0)	A(0)
0	0	0
0	1	1
1	0	1
1	1	0

4. Перевірка переповнення в молодшому біті після доповнення до 2 (OR)

```
--overflow check
OVF_CHECK_A:LUT4 generic map(mask=>X"0008")
port map(a=>a_buf(0), b=>a_buf(11),c=>'0',d=>'0', Y =>a_ovf_c(1));
OVF_CHECK_B:LUT4 generic map(mask=>X"0008")
port map(a=>b_buf(0), b=>b_buf(11),c=>'0',d=>'0', Y =>b_ovf_c(1));
```

A(11)	A(0)	C(1)
0	0	0
0	1	0
1	0	0
1	1	1

5. Врахування переповнення при переведенні в ДК
(спочатку перевіряємо переповнення в наступному розряді, далі при від'ємному значенні числа формуємо остаточний біт ДК з врахуванням переповнення в поточному розряді, при додатному значенні зберігаємо поточне значення біту в ПК)

```
LSM_OVF:for i in 1 to 10 generate
 FULL_OVF_CHECK_A:LUT4 generic map(mask=>X"0080")
```

```
 port map(a=>a_ovf_c(i), b=>a_buf(i),c=>a_buf(11),d=>'0', Y =>a_ovf_c(i+1));
 OVERFLOW_ADD_A:LUT4 generic map(mask=>X"006C")
 port map(a=>a_ovf_c(i), b=>a_buf(i),c=>a_buf(11),d=>'0', Y =>adk(i));
 FULL_OVF_CHECK_B:LUT4 generic map(mask=>X"0080")
 port map(a=>b_ovf_c(i), b=>b_buf(i),c=>b_buf(11),d=>'0', Y =>b_ovf_c(i+1));
 OVERFLOW_ADD_B:LUT4 generic map(mask=>X"006C")
 port map(a=>b_ovf_c(i), b=>b_buf(i),c=>b_buf(11),d=>'0', Y =>bdk(i));
end generate;
```

A(11)	A(i)	C(i)	C(i+1)	A(11)	A(i)	C(i)	Adk(i)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	1
1	1	0	0	1	1	0	1
1	1	1	1	1	1	1	0

6. Суматор (почергово формуємо суму відповідних бітів(XOR), перевіряємо переповнення в наступному розряді(OR), враховуємо переповнення в поточному розряді(XOR))

```
--ADDER
LSM_ADDER:for i in 0 to 11 generate
 ADD:LUT4 generic map(mask=>X"0006")
 port map(a=>bdk(i), b=>adk(i),c=>'0',d=>'0', Y =>x(i));
 OVERFLOW_CHECK:LUT4 generic map(mask=>X"0008")
 port map(a=>bdk(i), b=>adk(i),c=>'0',d=>'0', Y =>c(i+1));
 OVERFLOW_ADD:LUT4 generic map(mask=>X"0006")
 port map(a=>c(i), b=>x(i),c=>'0',d=>'0', Y =>ydk1(i));
end generate;
```

Adk(i)	Bdk(i)	X(i)	Adk(i)	Bdk(i)	C(i+1)	X(i)	C(i)	Ydk1(i)
0	0	0	0	0	0	0	0	0
0	1	1	0	1	0	0	1	1
1	0	1	1	0	0	1	0	1
1	1	0	1	1	1	1	1	0

7. Обробка подвійного переповнення на суматорі (Враховуємо можливість переповнення більше ніж на один розряд, почергово OR-XOR, починаючи з першого біту)

```
LSM_ADDER1:for i in 1 to 11 generate
 OVERFLOW_CHECK2:LUT4 generic map(mask=>X"0008")
 port map(a=>c(i), b=>x(i),c=>'0',d=>'0', Y =>c2(i+1));
 OVERFLOW_ADD2:LUT4 generic map(mask=>X"0006")
 port map(a=>ydk1(i), b=>c2(i),c=>'0',d=>'0', Y =>ydk(i));
end generate;
```

X(i)	C(i)	C2(i+1)	C2(i)	Ydk1(i)	Ydk(i)
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	0

8. Інвертор бітів результату (XOR)

```
--RESULT BIN CODE
LSM_TCC_REZ:for i in 0 to 10 generate
 --invertor
 INVERSION_REZ:LUT4 generic map(mask=>X"0006") --if sign "-"(1) then inv.
 port map(a=>ydk(i), b=>ydk(11), c=>'0', d=>'0', Y =>y_buf(i));
end generate;
y_buf(11)<=ydk(11);
```

Ydk(11)	Ydk(i)	Y_buf(i)
0	0	0
0	1	1
1	0	1
1	1	0

9. Доповнювач до 2 результату (XOR)

```
--add 1
ADD1_REZ:LUT4 generic map(mask=>X"0006")
port map(a=>y_buf(0), b=>y_buf(11),c=>'0',d=>'0', Y =>ypk(0));
```

Y_buf(11)	Y_buf(0)	Ypk(0)
0	0	0
0	1	1
1	0	1
1	1	0

10. Врахування переповнення при переведенні результату в ПК (аналогічно до кроку 5)

```
--overflow check
OVF_CHECK_REZ:LUT4 generic map(mask=>X"0008")
port map(a=>y_buf(0), b=>y_buf(11),c=>'0',d=>'0', Y =>y_ovf_c(1));

LSM_OVF_REZ:for i in 1 to 10 generate
 FULL_OVF_CHECK_REZ:LUT4 generic map(mask=>X"0080")
 port map(a=>y_ovf_c(i), b=>y_buf(i),c=>y_buf(11),d=>'0', Y =>y_ovf_c(i+1));
 OVERFLOW_ADD_REZ:LUT4 generic map(mask=>X"006C")
 port map(a=>y_ovf_c(i), b=>y_buf(i),c=>y_buf(11),d=>'0', Y =>ypk(i));
end generate;
```

11. Визначення мінімуму (мінімум визначається при F = 1, при такому значенні в кроці 1 виконувалось обернення знаку операнду B, і відповідно на виході суматору формувалась сума A + (–B) = A – B.

Шляхом оцінки знаку цієї суми можна визначити більший із двох операндів. Якщо знак від'ємний (1) то мінімумом являється A, якщо знак додатній то мінімумом буде B).

```
--MINIMUM
LSM_MIN:for i in 0 to 11 generate
 MIN:LUT4 generic map(mask=>X"00CA")
 port map(a=>B(i), b=>A(i),c=>ypk(11),d=>'0', Y =>m(i));
end generate;
```

ypk(11)	A(i)	B(i)	m(i)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

12. Мультиплексор (в залежності від F значення результату знімається або з виходу інвертору суматора, або з виходу мікросхеми мінімуму).

```
--MULTIPLEXER
LSM_MULT:for i in 0 to 11 generate
 MULT:LUT4 generic map(mask=>X"00CA")
 port map(a=>ypk(i), b=>m(i),c=>F,d=>'0', Y =>rez(i));
end generate;
```

F	m(i)	ypk(i)	rez(i)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

13.Ознака нульового результату (тричі перевіряємо по 4 біти результату на NAND, формуємо вектор-ознаку, який потім перевіряємо на OR).

```
-- Definition of zero result
 UZ1:LUT4 generic map(mask=>X"0001")
 port map(a=>rez(3),b=>rez(2),c=> rez(1),d =>rez(0), Y =>zer(0));
 UZ2:LUT4 generic map(mask=>X"0001")
 port map(a=>rez(7),b=>rez(6),c=> rez(5),d =>rez(4), Y =>zer(1));
 UZ3:LUT4 generic map(mask=>X"0001")
 port map(a=>rez(11),b=>rez(10),c=> rez(9),d =>rez(8), Y =>zer(2));
 UZ4:LUT4 generic map(mask=>X"0080")
 port map(a=>zer(0),b=>zer(1),c=> zer(2),d =>'0', Y =>Z);
```

rez(i)	rez(i+1)	rez(i+2)	rez(i+3)	zer(i)	zer(2)	zer(1)	zer(0)	Z
0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0	0
0	0	1	1	0	0	1	1	0
0	1	0	0	0	1	0	0	0
0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	1	0	0
0	1	1	1	0	1	1	1	1
1	0	0	0	0				
1	0	0	1	0				
1	0	1	0	0				
1	0	1	1	0				
1	1	0	0	0				
1	1	0	1	0				
1	1	1	0	0				
1	1	1	1	0				

Код програми:

Поведінкова модель:

```
--use CNetwork.all, FUN.all;
library ieee;
use ieee.numeric_bit.all;

entity LSM is
port(F : in BIT;-- function
        A : in BIT_VECTOR(11 downto 0);-- first operand
        B : in BIT_VECTOR(11 downto 0);-- second operand
        C0: in BIT; -- transfer port
        Y : out BIT_VECTOR(11 downto 0);-- result
        C3: out BIT; -- transfer output
        Z: out BIT; -- bit of zero result
        N: out BIT);-- sign bit
end LSM;

architecture BEH of LSM is
signal ai, bi, ci: signed(11 downto 0);
signal yi: signed(12 downto 0);
signal ybi: BIT_VECTOR (12 downto 0);

function MIN(a,b:signed)return signed is
begin
        if a> b then
                return b;
        else
                return a;
        end if;
end function ;

begin
-- Present the input data as signed --
ai <= signed(A);
bi <= signed(B);
-- Adder --
```

```
ADDER: yi <= RESIZE(ai,13) + bi + signed('0'&C0) when F = '0';
-- Results multiplexer --
MUX: with F select
 ybi <= bit_vector(yi) when '0', -- adder
 '0'&bit_vector(MIN(ai,bi)) when others;
 C3 <= ybi (12); -- output transfer
 N <= ybi(11); --sign bit
 Z <= '1' when ybi (11 downto 0) = "000000000000" else '0'; -- zero sign
 Y <= ybi (11 downto 0); -- Result
end BEH;
```

Структурна модель:

```
architecture STR_LUT of LSM is
signal a_ovf_c, b_ovf_c, y_ovf_c, c, c2, x, yi, yr: BIT_VECTOR (13 downto 0);
signal zer:BIT_VECTOR(2 downto 0);
signal a_buf, b_buf, y_buf, adk, bdk, ydk, ydk1, ypk, m, rez: BIT_VECTOR (11 downto 0);
signal buffer1, buffer2, R: BIT;

 component LUT4 is
 generic(mask:BIT_VECTOR(15 downto 0):=X"ffff"; td:time:=1 ns);
 port (a, b, c, d: in BIT;
 Y: out BIT);
 end component;
 component LUT5 is
 generic(mask:BIT_VECTOR(31 downto 0):=X"ffffffff"; td:time:=1 ns);
 port (a, b, c, d, e: in BIT;
 Y: out BIT);
 end component;
 begin
 c(0)<=C0;

 --BUFFERING: for i in 0 to 11 generate
          --a_buf(i)<=A(i);
          --b_buf(i)<=B(i);
 --end generate;
-- Arithmetic-logic circuit diagram ----------------------------

--if F=1 then reverse sign B
LSM_REVERSE:LUT4 generic map(mask=>X"0006")
port map(a=>B(11), b=>F,c=>'0',d=> '0', Y =>buffer1);

--save A and B sing
a_buf(11)<=A(11);
b_buf(11)<=buffer1;

--Two's complement code
LSM_TCC:for i in 0 to 10 generate
 --invertor
 INVERSION_A:LUT4 generic map(mask=>X"0006") --if sign "-"(1) then inv.
 port map(a=>A(i), b=>a_buf(11),c=>'0',d=>'0', Y =>a_buf(i));
 INVERSION_B:LUT4 generic map(mask=>X"0006") --if sign "-"(1) then inv.
 port map(a=>B(i), b=>b_buf(11),c=>'0',d=>'0', Y =>b_buf(i));
end generate;

 --add 1
ADD1_A:LUT4 generic map(mask=>X"0006")
port map(a=>a_buf(0), b=>a_buf(11),c=>'0',d=>'0', Y =>adk(0));
ADD1_B:LUT4 generic map(mask=>X"0006")
port map(a=>b_buf(0), b=>b_buf(11),c=>'0',d=>'0', Y =>bdk(0));
```

```
 --overflow check
OVF_CHECK_A:LUT4 generic map(mask=>X"0008")
port map(a=>a_buf(0), b=>a_buf(11),c=>'0',d=>'0', Y =>a_ovf_c(1));
OVF_CHECK_B:LUT4 generic map(mask=>X"0008")
port map(a=>b_buf(0), b=>b_buf(11),c=>'0',d=>'0', Y =>b_ovf_c(1));

LSM_OVF:for i in 1 to 10 generate
 FULL_OVF_CHECK_A:LUT4 generic map(mask=>X"0080")
 port map(a=>a_ovf_c(i), b=>a_buf(i),c=>a_buf(11),d=>'0', Y =>a_ovf_c(i+1));
 OVERFLOW_ADD_A:LUT4 generic map(mask=>X"006C")
 port map(a=>a_ovf_c(i), b=>a_buf(i),c=>a_buf(11),d=>'0', Y =>adk(i));
 FULL_OVF_CHECK_B:LUT4 generic map(mask=>X"0080")
 port map(a=>b_ovf_c(i), b=>b_buf(i),c=>b_buf(11),d=>'0', Y =>b_ovf_c(i+1));
 OVERFLOW_ADD_B:LUT4 generic map(mask=>X"006C")
 port map(a=>b_ovf_c(i), b=>b_buf(i),c=>b_buf(11),d=>'0', Y =>bdk(i));
end generate;

--sign bit
adk(11)<=a_buf(11);
bdk(11)<=b_buf(11);

--ADDER
LSM_ADDER:for i in 0 to 11 generate
 ADD:LUT4 generic map(mask=>X"0006")
 port map(a=>bdk(i), b=>adk(i),c=>'0',d=>'0', Y =>x(i));
 OVERFLOW_CHECK:LUT4 generic map(mask=>X"0008")
 port map(a=>bdk(i), b=>adk(i),c=>'0',d=>'0', Y =>c(i+1));
 OVERFLOW_ADD:LUT4 generic map(mask=>X"0006")
 port map(a=>c(i), b=>x(i),c=>'0',d=>'0', Y =>ydk1(i));
end generate;

LSM_ADDER1:for i in 1 to 11 generate
 OVERFLOW_CHECK2:LUT4 generic map(mask=>X"0008")
 port map(a=>c(i), b=>x(i),c=>'0',d=>'0', Y =>c2(i+1));
 OVERFLOW_ADD2:LUT4 generic map(mask=>X"0006")
 port map(a=>ydk1(i), b=>c2(i),c=>'0',d=>'0', Y =>ydk(i));
end generate;

--CHECK SIGN OVERFLOW
 --SIGN_OVF_CHECK:LUT4 generic map(mask=>X"0006")
 --port map(a=>c(12), b=>c(11),c=>'0',d=>'0', Y =>R);
 --R<='0';

--RESULT BIN CODE
LSM_TCC_REZ:for i in 0 to 10 generate
 --invertor
 INVERSION_REZ:LUT4 generic map(mask=>X"0006") --if sign "-"(1) then inv.
 port map(a=>ydk(i), b=>ydk(11), c=>'0', d=>'0', Y =>y_buf(i));
end generate;
y_buf(11)<=ydk(11);

 --add 1
ADD1_REZ:LUT4 generic map(mask=>X"0006")
port map(a=>y_buf(0), b=>y_buf(11),c=>'0',d=>'0', Y =>ypk(0));

 --overflow check
OVF_CHECK_REZ:LUT4 generic map(mask=>X"0008")
port map(a=>y_buf(0), b=>y_buf(11),c=>'0',d=>'0', Y =>y_ovf_c(1));
```

```
LSM_OVF_REZ:for i in 1 to 10 generate
 FULL_OVF_CHECK_REZ:LUT4 generic map(mask=>X"0080")
 port map(a=>y_ovf_c(i), b=>y_buf(i),c=>y_buf(11),d=>'0', Y =>y_ovf_c(i+1));
 OVERFLOW_ADD_REZ:LUT4 generic map(mask=>X"006C")
 port map(a=>y_ovf_c(i), b=>y_buf(i),c=>y_buf(11),d=>'0', Y =>ypk(i));
end generate;

--sign bit
ypk(11)<=y_buf(11);

--MINIMUM
LSM_MIN:for i in 0 to 11 generate
 MIN:LUT4 generic map(mask=>X"00CA")
 port map(a=>B(i), b=>A(i),c=>ypk(11),d=>'0', Y =>m(i));
end generate;

--MULTIPLEXER
LSM_MULT:for i in 0 to 11 generate
 MULT:LUT4 generic map(mask=>X"00CA")
 port map(a=>ypk(i), b=>m(i),c=>F,d=>'0', Y =>rez(i));
end generate;

-- Definition of zero result
 UZ1:LUT4 generic map(mask=>X"0001")
 port map(a=>rez(3),b=>rez(2),c=> rez(1),d =>rez(0), Y =>zer(0));
 UZ2:LUT4 generic map(mask=>X"0001")
 port map(a=>rez(7),b=>rez(6),c=> rez(5),d =>rez(4), Y =>zer(1));
 UZ3:LUT4 generic map(mask=>X"0001")
 port map(a=>rez(11),b=>rez(10),c=> rez(9),d =>rez(8), Y =>zer(2));
 UZ4:LUT4 generic map(mask=>X"0080")
 port map(a=>zer(0),b=>zer(1),c=> zer(2),d =>'0', Y =>Z);

 Y<=rez(11 downto 0);
 N<=rez(11);
 C3<=y_ovf_c(12);  --extraction of transfer
end STR_LUT;
```

Стенд для відлагодження:

```
use CNetwork.all;
entity lsm_tb is
end lsm_tb;
architecture TB_ARCHITECTURE of lsm_tb is
 component LSM
 port(F : in BIT;
 A : in BIT_VECTOR(11 downto 0);
 B : in BIT_VECTOR(11 downto 0);
 C0: in BIT;
 Y : out BIT_VECTOR(11 downto 0);
 C3: out BIT;
 N: out BIT;
 Z : out BIT );
end component;
component RANDOM_GEN is
 generic(n:positive:=12;
 tp:time:=100 ns ;
 SEED:positive:=1234);
 port(CLK:out BIT;
 Y : out BIT_VECTOR(n-1 downto 0));
end component;
component RANDOM_BIT is
```

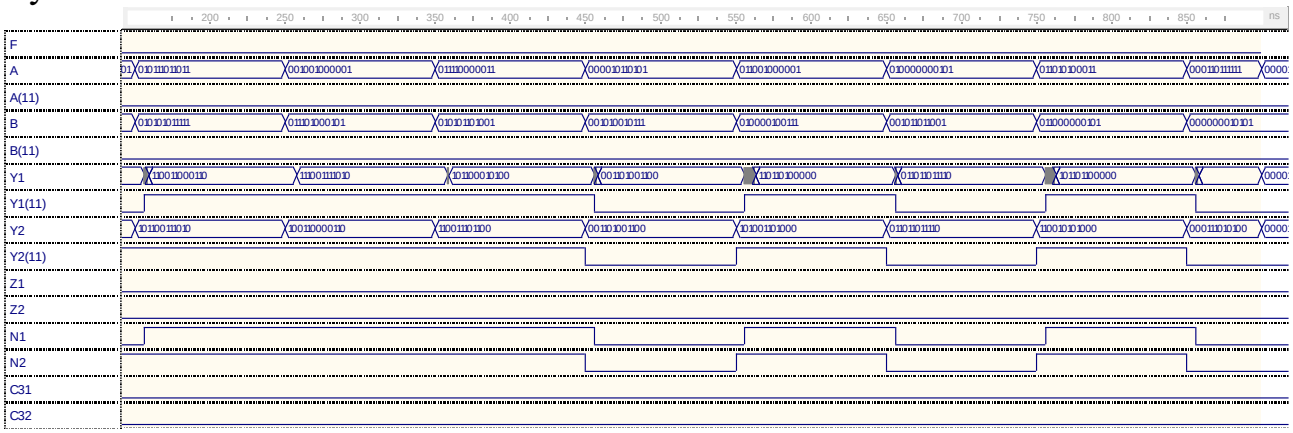
```
 generic(tp:time:=100 ns ;
 SEED:positive:=1234);
 port(CLK:out BIT;
 Y : out BIT);
end component;

signal F : BIT:='0';
signal C0 : BIT:='0';
signal A,B : BIT_VECTOR(11 downto 0);
signal Y1,Y2,Y : BIT_VECTOR(11 downto 0);
signal C31,C32,C, Z1,Z2,Z, N1,N2,N: BIT;
signal A1, B1, C1 : BIT_VECTOR(11 downto 0);
begin
G1: RANDOM_GEN
 generic map(n=>12,SEED=>123)
 port map(CLK=>open,Y =>A);
G2: RANDOM_GEN
generic map(n=>12,SEED=>654)
port map(CLK=>open,Y =>B);
UUT2 :entity LSM(BEH)
 port map (F => F,A => A,B => B,C0 => C0,
 Y => Y2, C3 => C32, N=>N2, Z => Z2);
UUT1 :entity LSM(STR_LUT)
 port map (F => F,A => A,B => B, C0 => C0,
 Y => Y1, C3 => C31, N=>N1, Z => Z1);

 COMP_Y: Y<=Y1 xor Y2;
 COMP_C: C<=C31 xor C32;
 COMP_Z: Z<=Z1 xor Z2;
 COMP_N: N<=N1 xor N2;
end TB_ARCHITECTURE;
```

Результати симуляції:

Сума:



Мінімум: