**Tema1**

Movileanu Raluca 331CB

1. Codul sursa C++ (\*.h și \*.cpp)

cl\_adder\_16bit.cpp:

**#include** "cl\_adder\_16bit.h"

// Functia implementeaza un adder carry look-ahead de 16 biti.

// Intrari: a, b - numere de 16 biti pentru adunare

// Iesiri: sum - suma calculata de 16 biti, carry\_out - bitul de carry out

**void** **cl\_adder\_16bit**(**ap\_uint**<16> a, **ap\_uint**<16> b, **ap\_uint**<16> &sum, **ap\_uint**<1> &carry\_out) {

// Array-uri pentru stocarea valorilor de generate si propagate

**ap\_uint**<1> generate[16];

**ap\_uint**<1> propagate[16];

// Initializarea array-ului de carry cu toate valorile la 0

**ap\_uint**<1> carry[17] = {0};

// Calculul generate si propagate pentru fiecare bit

**for** (**int** i = 0; i < 16; i++) {

generate[i] = a[i] & b[i]; // Genereaza (AND) pentru fiecare bit

propagate[i] = a[i] ^ b[i]; // Propaga (XOR) pentru fiecare bit

}

// Initializare carry-in ca 0

carry[0] = 0;

// Calculul carry pentru fiecare bit, folosind valorile de generate si propagate

**for** (**int** i = 0; i < 16; i++) {

carry[i+1] = generate[i] | (propagate[i] & carry[i]);

}

// Calculul sumei pentru fiecare bit

**for** (**int** i = 0; i < 16; i++) {

sum[i] = propagate[i] ^ carry[i]; // XOR intre propagate si carry

}

// Seteaza carry-out ca ultimul bit de carry calculat

carry\_out = carry[16];

}

cl\_adder\_16bit.h:

**#ifndef** **CL\_ADDER\_16BIT\_H\_**

**#define** **CL\_ADDER\_16BIT\_H\_**

**#include** <ap\_int.h>

**void** **cl\_adder\_16bit**(**ap\_uint**<16> a, **ap\_uint**<16> b, **ap\_uint**<16> &sum, **ap\_uint**<1> &carry\_out);

**#endif**

1. Codul sursă pentru testbench (\*-tb.h și \*-tb.cpp)

adder-tb.cpp:

**#include** "adder-tb.h"

**#include** "cl\_adder\_16bit.h"

**#include** <iostream>

**#include** <bitset>

**void** test\_cl\_adder\_16bit() {

**ap\_uint**<16> a, b;

**ap\_uint**<16> sum;

**ap\_uint**<1> carry;

// Test cu valori mici

a = 0x0003; // 0000000000000011

b = 0x0002; // 0000000000000010

**cl\_adder\_16bit**(a, b, sum, carry);

std::cout << "SUMA: Test cu valori mici: " << std::**bitset**<16>(sum).to\_string() << ", Carry: " << carry << std::**endl**;

// Test cu overflow

a = 0xFFFF; // 1111111111111111

b = 0xFFFF; // 1111111111111111

**cl\_adder\_16bit**(a, b, sum, carry);

std::cout << "SUMA: Test cu overflow: " << std::**bitset**<16>(sum).to\_string() << ", Carry: " << carry << std::**endl**;

// Test cu 0

a = 0x1234; // 0001001000110100

b = 0x0000; // 0000000000000000

**cl\_adder\_16bit**(a, b, sum, carry);

std::cout << "SUMA: Test cu 0: " << std::**bitset**<16>(sum).to\_string() << ", Carry: " << carry << std::**endl**;

// Test cu valori medii

a = 0x0A5A; // 0000101001011010

b = 0x05A0; // 0000010110100000

**cl\_adder\_16bit**(a, b, sum, carry);

std::cout << "SUMA: Test cu valori medii: " << std::**bitset**<16>(sum).to\_string() << ", Carry: " << carry << std::**endl**;

//Test cu un bit diferit

a = 0x5555; // 0101010101010101

b = 0x5554; // 0101010101010100

**cl\_adder\_16bit**(a, b, sum, carry);

std::cout << "SUMA: Test cu un bit diferit: " << std::**bitset**<16>(sum).to\_string() << ", Carry: " << carry << std::**endl**;

// Test cu Carry Out 1

a = 0xFFFF; // 1111111111111111 (valoarea maxima pe 16 biți)

b = 0x0001; // 0000000000000001 (cel mai mic numar pozitiv)

**cl\_adder\_16bit**(a, b, sum, carry);

std::cout << "SUMA: Test cu Carry out 1: " << std::**bitset**<16>(sum).to\_string() << ", Carry: " << carry << std::**endl**;

}

**int** **main**() {

test\_cl\_adder\_16bit();

**return** 0;

}

adder-tb.h:

**#ifndef** ADDER\_TB\_H\_

**#define** **ADDER\_TB\_H\_**

**#include** <ap\_int.h>

**void** **cl\_adder\_16bit**(**ap\_uint**<16> a, **ap\_uint**<16> b, **ap\_uint**<16> &sum, **ap\_uint**<1> &carry\_out);

**void** **test\_cl\_adder\_16bit**();

**#endif** // ADDER\_TB\_H

3.Rezultatele simularii:

INFO: [SIM 2] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CSIM start \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INFO: [SIM 4] CSIM will launch GCC as the compiler.

Compiling ../../../cl\_adder\_16bit.cpp in debug mode

Generating csim.exe

SUMA: Test cu valori mici: 0000000000000101, Carry: 0

SUMA: Test cu overflow: 1111111111111110, Carry: 1

SUMA: Test cu 0: 0001001000110100, Carry: 0

SUMA: Test cu valori medii: 0000111111111010, Carry: 0

SUMA: Test cu un bit diferit: 1010101010101001, Carry: 0

SUMA: Test cu Carry out 1: 0000000000000000, Carry: 1

INFO: [SIM 1] CSim done with 0 errors.

INFO: [SIM 3] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CSIM finish \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. Raportul generat de sinteza codului

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated5. Formele de undă obținute prin CO-SIMULARE