**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**

**Кафедра ЕОМ**



Звіт

до лабораторної роботи № 4

з дисципліни «Комп’ютерні системи»

на тему: «Аналіз програмної моделі процесу роботи арифметичного конвеєра, ч.2.»

Варіант №17

Виконав:

ст.гр. КІ-38

Онисько О.А.

Прийняв:

Козак Н.Б.

**Львів 2022**

Мета роботи: Навчитись здійснювати аналіз програмних моделей комп’ютерних систем, виконаних на мові System C.

**Завдання**

1. Проаналізувати склад програмної моделі арифметичного конвеєра, (програма PIPE), яка виконана на мові System C.

2. Здійснити модернізацію функцій або параметрів арифметичного конвеєра (див. лабораторну роботу № 3), шляхом під’єднання розроблених модулів S1 та S2 (див. лабораторну роботу № 2). Порядок та тип з’єднання мають бути обгрунтовані, можливо розробка буферних або додаткових модулів з метою надавання нових властивостей тестувальній моделі.

3. Накреслити кінцеву структурну схему отриманої програмної моделі.

4. Навести стисло код та внесені нові зміни.

5. Навести результати тестування та використання програмної моделі.

4. Оформити звіт.

**Хід роботи**

Лістинг програмної моделі арифметичного конвеєра на мові System C:

display.h

#ifndef DISPLAY\_H

#define DISPLAY\_H

#include "stage4.h" // Added by ClassView

struct display : sc\_module {

sc\_in<double> in; // input port 1

sc\_in<bool> clk; // clock

void print(); // method

SC\_CTOR(display) {

SC\_METHOD(print); // declare print as SC\_METHOD and

sensitive\_pos << clk; // make it sensitive to positive clock edge

}

public:

};

#endif

numgen.h

#ifndef NUMGEN\_H

#define NUMGEN\_H

struct numgen : sc\_module {

sc\_out<double> out1; //output 1

sc\_out<double> out2; //output 2

sc\_in<bool> clk; //clock

// method to write values to the output ports

void generate();

//Constructor

SC\_CTOR(numgen) {

SC\_METHOD(generate); //Declare generate as SC\_METHOD and

sensitive\_pos << clk; //make it sensitive to positive clock edge

}

};

#endif

Stage1.h

#ifndef STAGE0\_H

#define STAGE0\_H

struct stage0 : sc\_module {

sc\_in<double> in1; //input 1

sc\_in<double> in2; //input 2

sc\_out<double> power; //output 1

sc\_out<double> f1; //output 2

sc\_in<bool> clk; //clock

void adddif(); //method implementing functionality

//Counstructor

SC\_CTOR(stage1) {

SC\_METHOD(adddif); //Declare addsub as SC\_METHOD and

sensitive\_pos << clk; //make it sensitive to positive clock edge

}

public:

};

#endif

Stage2.h

#ifndef STAGE2\_H

#define STAGE2\_H

struct stage1 : sc\_module {

sc\_in<double> power; //input 1

sc\_in<double> f1; //input 2

sc\_out<double> r1; //output 1

sc\_out<double> r2; //output 2

sc\_in<bool> clk; //clock

void factadd(); //method implementing functionality

//Counstructor

SC\_CTOR(stage1) {

SC\_METHOD(factadd); //Declare addsub as SC\_METHOD and

sensitive\_pos << clk; //make it sensitive to positive clock edge

}

public:

};

#endif

Stage3.h

#ifndef STAGE3\_H

#define STAGE3\_H

struct stage2 : sc\_module {

sc\_in<double> r1; //input 1

sc\_in<double> r2; //input 2

sc\_out<double> sum; //output 1

sc\_out<double> diff; //output 2

sc\_in<bool> clk; //clock

void addsub(); //method implementing functionality

//Counstructor

SC\_CTOR(stage2) {

SC\_METHOD(addsub); //Declare addsub as SC\_METHOD and

sensitive\_pos << clk; //make it sensitive to positive clock edge

}

public:

};

#endif

Stage4.h

#ifndef STAGE4\_H

#define STAGE4\_H

struct stage3 : sc\_module {

sc\_in<double> sum; //input port 1

sc\_in<double> diff; //input port 2

sc\_out<double> prod; //output port 1

sc\_out<double> quot; //output port 2

sc\_in<bool> clk; //clock

void multdiv(); //method providing functionality

//Constructor

SC\_CTOR(stage3) {

SC\_METHOD(multdiv); //Declare multdiv as SC\_METHOD and

sensitive\_pos << clk; //make it sensitive to positive clock edge.

}

};

#endif

Stage5.h

#ifndef STAGE5\_H

#define STAGE5\_H

struct stage4 : sc\_module {

sc\_in<double> prod; //input port 1

sc\_in<double> quot; //input port 2

sc\_out<double> powr; //output port 1

sc\_in<bool> clk; //clock

void power(); //method implementing functionality

//Constructor

SC\_CTOR(stage4) {

SC\_METHOD(power); //declare power as SC\_METHOD and

sensitive\_pos << clk; //make it sensitive to positive clock edge

}

};

#endif

display.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "display.h"

#include <stdio.h>

#include <iostream>

#include <iomanip>

using namespace std;

//Definition of print\_result method

void display::print()

{

printf("Result = %f\n", in.read());

} // end of print method

numgen.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "numgen.h"

// definition of the `generate' method

void numgen::generate()

{

static double a = 5.5;

static double b = 25.8;

a -= 1.5;

b -= 2.8;

out1.write(a);

out2.write(b);

} // end of `generate' method

Stage1.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage1.h"

#include <math.h>

//Definition of addsub method

void stage0::adddif()

{

double a;

double b;

a = in1.read();

b = in2.read();

int temp = a - b;

power.write(temp);

if (b == 0) {

b = 5.0;

int temp = (in1.read() + in2.read() / 2.0);

f1.write(temp);

}

}

Stage2.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage2.h"

//Definition of addsub method

void stage1::factadd()

{

double a;

double b;

a = power.read();

b = f1.read();

r1.write((int)a + (int)b);

r2.write(5);

} // end of addsub method

Stage3.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage3.h"

//Definition of addsub method

void stage2::addsub()

{

double a;

double b;

a = r1.read();

b = r2.read();

sum.write(a + b);

diff.write(a - b);

} // end of addsub method

Stage4.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage3.h"

//definition of multdiv method

void stage3::multdiv()

{

double a;

double b;

a = sum.read();

b = diff.read();

if (b == 0)

b = 5.0;

prod.write(a\*b);

quot.write(a/b);

} // end of multdiv

Stage5.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage4.h"

void stage4::power()

{

double a;

double b;

double c;

a = prod.read();

b = quot.read();

c = (a>0 && b>0) ? pow(a, b) : 0.;

powr.write(c);

} // end of power method

main.cpp

#include "systemc.h"

#include "stage0.h"

#include "stage2.h"

#include "stage3.h"

#include "stage4.h"

#include "stage5.h"

#include "display.h"

#include "numgen.h"

#define NS \* 1e-9

int sc\_main(int ac, char\* av[])

{

//Signals

sc\_signal<double> in1;

sc\_signal<double> in2;

sc\_signal<double> power;

sc\_signal<double> f1;

sc\_signal<double> r1;

sc\_signal<double> r2;

sc\_signal<double> sum;

sc\_signal<double> diff;

sc\_signal<double> prod;

sc\_signal<double> quot;

sc\_signal<double> powr;

sc\_signal<bool> clk;

//Clock

//<TRACE>

//</TRACE>

numgen N("numgen"); //instance of `numgen' module

N(in1, in2, clk); //Positional port binding

stage0 S0("stage0"); //instance of `stage1' module

//Named port binding

S0.in1(in1);

S0.in2(in2);

S0.power(power);

S0.f1(f1);

S0.clk(clk);

sc\_trace\_file\* wf = sc\_create\_vcd\_trace\_file("signals");

wf->set\_time\_unit(1, SC\_NS);

sc\_trace(wf, clk, "CLK");

sc\_trace(wf, in1, "In1");

sc\_trace(wf, in2, "In2");

sc\_trace(wf, power, "power");

sc\_trace(wf, f1, "f1");

sc\_trace(wf, r1, "r1");

sc\_trace(wf, r2, "r2");

sc\_trace(wf, sum, "Sum");

sc\_trace(wf, diff, "Diff");

sc\_trace(wf, prod, "Prod");

sc\_trace(wf, quot, "Quot");

sc\_trace(wf, powr, "Pow");

stage1 S1("stage1"); //instance of `stage2' module

S1(power, f1, r1, r2, clk);

stage2 S2("stage2"); //instance of `stage2' module

S2( r1, r2, sum, diff, clk);

stage3 S3("stage3"); //instance of `stage2' module

S3(sum, diff, prod, quot, clk); //Positional port binding

stage4 S4("stage4"); //instance of `stage3' module

S4(prod, quot, powr, clk); //Positional port binding

display D("display"); //instance of `display' module

D(powr, clk); //Positional port binding

//<TRACE>

//</TRACE>

//<TRACE>

sc\_start(0, SC\_NS);

for (int i = 0; i < 50; i++)

{

clk.write(0);

sc\_start(10, SC\_NS);

clk.write(1);

sc\_start(10, SC\_NS);

}

sc\_close\_vcd\_trace\_file(wf);

//</TRACE>

return 0;

}

**Перелік і призначення блоків арифметичного конвеєра:**

Блок stage0 має два входи і обчислює степінь: обчислюється різниця вхідних даних; а також обчислює заданий вираз.

Блок stage2 має два входи і виконує операцію додавання вхідних даних, а також вивід числа 5.

Блок stage3 має два входи і обчислює суму вхідних даних та їх різницю.

Блок stage4 приймає дані з попереднього блоку та обчислює їх добуток та частку.

Блок stage5 приймає дані з попереднього блоку та обчислює перше вхідне число підняте до степені, що рівна другому вхідному числу.

**Структурна схема арифметичного конвеєра, що відповідає програмній моделі:**

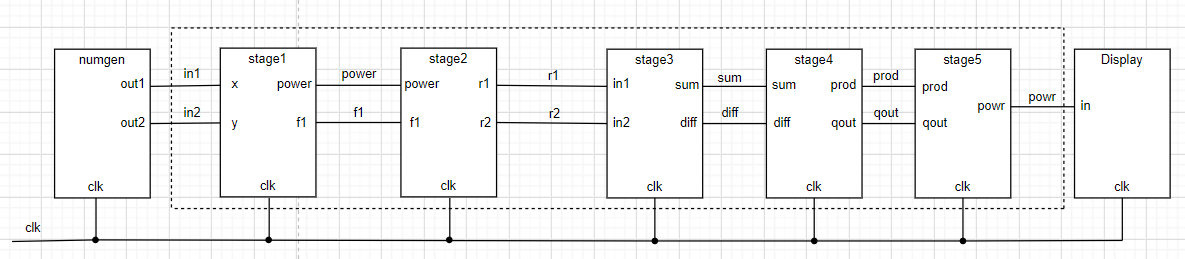


Рис.1. Структурна схема

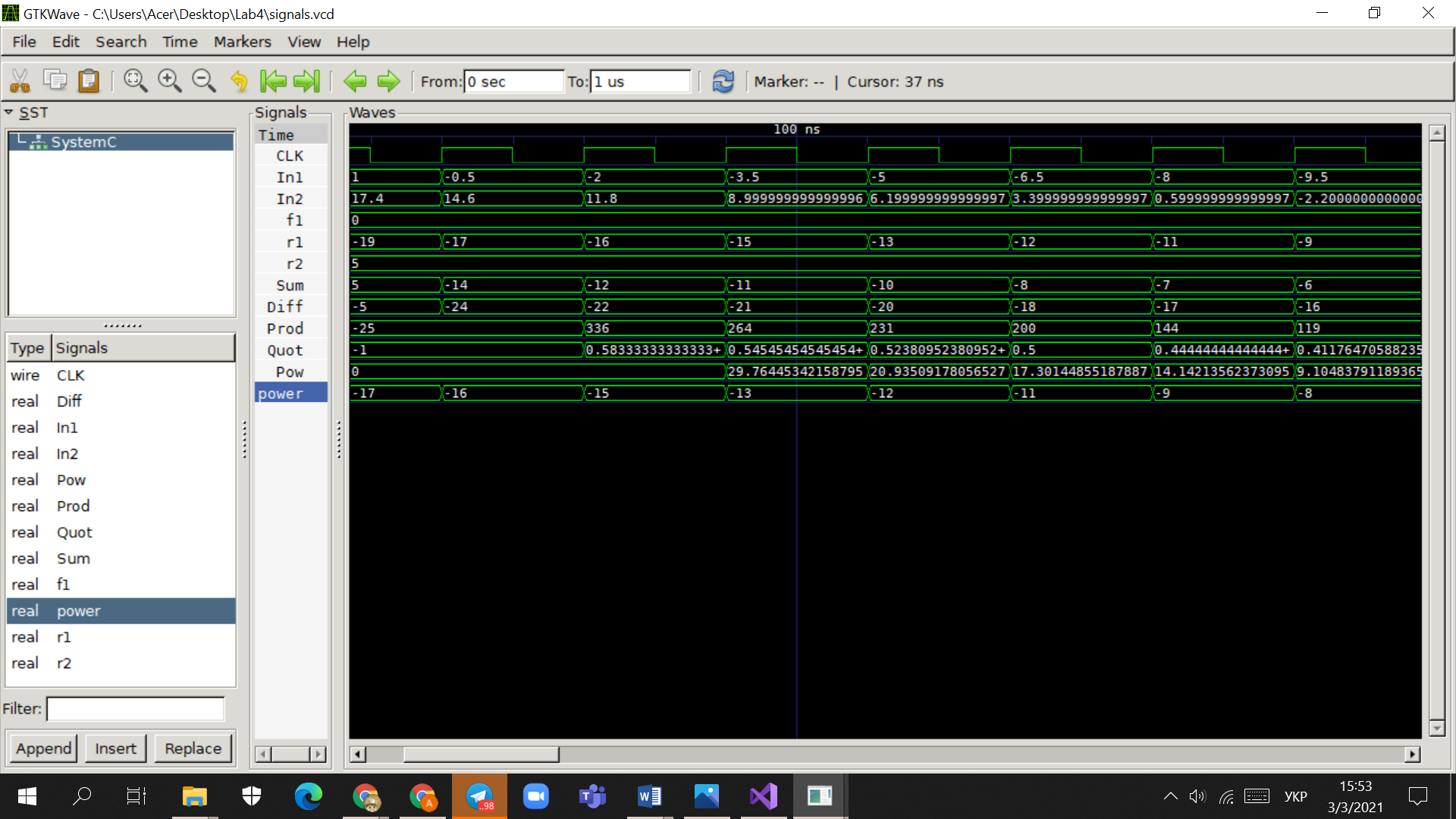
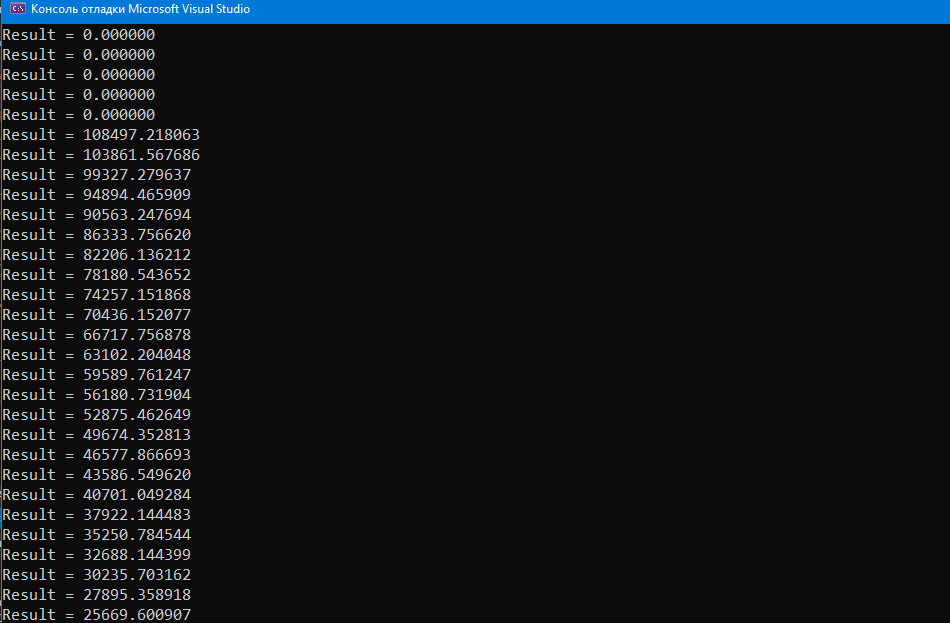


Рис.2. Часова діаграма роботи



**Висновок:** навчився здійснювати аналіз програмних моделей комп’ютерних систем, виконаних на мові System C.