**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**



Звіт

до лабораторної роботи № 4

з дисципліни «Комп’ютерні системи»

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

Варіант №5

Виконав:

ст.гр. КІ-38

Возний Т.І.

Прийняв:

Козак Н.Б.

**Львів 2021**

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

**Теоретичні відомості:**

Завданням планувальника є визначення порядку виконання процесів в межах повідомлень, що виникають та в межах проекту , в основі якого лежить чутливість процесів до подій.

Планувальник SystemC підтримує моделювання, орієнтоване на апаратні засоби та підтримує також програмно-орієнтоване моделювання.

Подібно до VHDL та Verilog, планувальник SystemC підтримує дельта цикли. Дельта цикл складається з розділених оцінюючих і оновлюючих стадій ;багатократні дельта цикли можуть бути присутніми в окремих часових інтервалах. Дельта цикли корисні для моделювання повністю дискретних, синхронізованих в часі обчислень, як наприклад, в RTL. У SystemC , використовуючи notify() з нульовим значенням часу призводить до того, що повідомлення про подію зявиться на стадії обчислень наступного дельта циклу, поки виклик request\_update() спричинить виклик update() на стадії поновлення текучого дельта-циклу. Використовуючи такі засоби зв’язку можуть бути побудовані канали, що моделюють поведінку сигналів апаратних засобів ЕОМ.

SystemC підтримує також синхронізовані повідомлення про події. Синхронізовані повідомлення визначаються з використанням notify() з часовим аргументом. Синхронізовані повідомлення примушують конкретні події повідомляти про себе в майбутньому у чітко визначений час. Синхронізовані повідомлення існують в VHDL та Verilog і є корисні при моделюванні програмнимих засобів.

Нарешті, SystemC підтримує негайні повідомлення про події, які визначаються викликом notify() без аргументів. Негайні повідомлення примушують процеси, що чутливі до подій , негайно переходити в стан готовності до виконання (наприклад, готовність до виконання в проміжок виконання обчислень). Негайні повідомлення корисні для моделювання систем програмування та операційних систем, у яких відсутнє поняття дельта-циклу.

Наступні кроки виділяють роботу планувальника SystemC. Детальніший псевдо-код для планувальника-в додатку А.

1. Фаза ініціалізації – виконує всі процеси (крім SC\_CTHREADs) в довільному порядку.
2. Фаза оцінки (обчислення) – вибрати процес що готовий до виконання та продовжити його виконання. Це може спричинити появу негайних повідомлень, що може відбитися на можливості запуску (підготовки до запуску) інших процесів в цей самий час.
3. Якщо все ще присутні процеси, готові до запуску, виконувати крок 2.
4. Фаза оновлення – виконує усі очікувані виклики update(), спричинені request\_update() з пункту 2
5. Якщо є відстроковані повідомлення, визначити, які з процесів можуть запускатися, реагуючи на відстроковані повідомлення. Далі йти до кроку 2
6. Якщо немає синхронізуючих повідомлень моделювання вважати завершеним.
7. Перемістити текучий час моделювання на відмітку найпершого синхронізуючого повідомлення.
8. Визначити, які з процесів готові до виконання внаслідок подій, що залишили необроблені повідомлення в даний момент часу. Далі йти до кроку 2.

Можливість проектувати визначені користувачем зв’язки з довільним інтерфейсом підтримується інтерфейсним стилем дизайну. Цей вид проектування дозволяє легко та гнучко проводити процес з’єднання, що дозволяє реалізувати абстрактні з’єднувальні канали. Наступний приклад це демонструє, зображуючи один з можливих шляхів досягнення.

**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.

Програмна модель складається з 8 модулів:

* **Numgen** – модуль який генерує на своїх виходах 2 числа:

a = № варіанту за списком в журналі.

b = сума ASCII code першої літери прізвища + першої літери імені.

a = 5

b = 86 + 84 = 170

* **Stage1** – модуль обчислює функції o1, o2.

(o1 = X – Y;

o2 = (X + Y / 2.0) && (X >> 2)

Повторює модуль st1 з лабораторної №2.

* **Stage2** – модуль обчислює функції r1, r2.

(r1 = a \* b;

r2 = Номер варіанту)

Повторює модуль st2 з лабораторної №2.

* **Stage3** – модуль обчислює суму та різницю вхідних значень.
* **Stage4** – модуль обчислює добуток та частку.
* **Stage5** – модуль обчислює значення a в степені b (a та b вхідні сигнали).
* **Display** – модуль відображає на екрані кінцевий результат.
* **Observe** – модуль відображає на екрані проміжні значення.

Структурна схема арифметичного конвеєра:

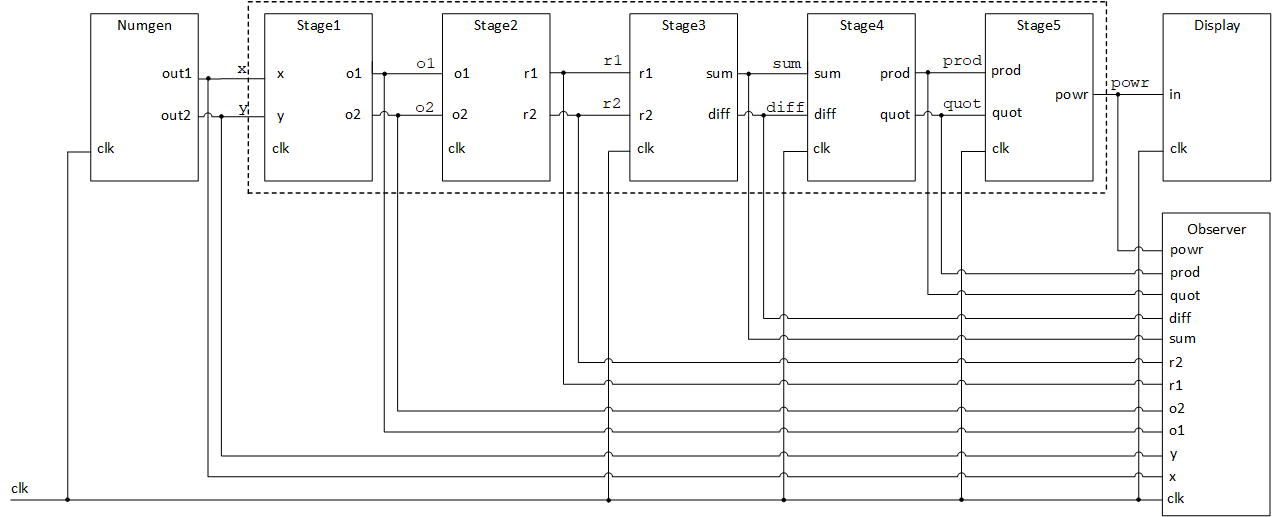


Рис. 1. Структурна схема арифметичного конвеєра

**Внесені зміни:**

До структури арифметичного конвеєра з лабораторної роботи №3 було додано два модулі з лабораторної роботи №2 та один модуль Observe (другий екран), який використувується паралельно з основним екраном і створений для демонстрації правильного функціонування усіх попередніх модулів і розширення функціоналу самого конвеєра.

**Тестування програми:**

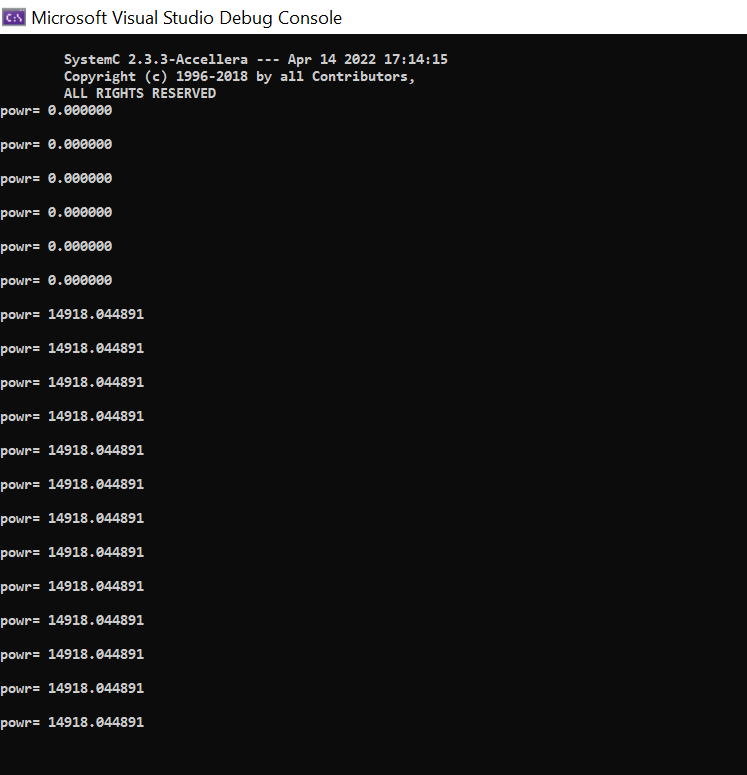
****

Рис. 2. Тестування програми в сукупності з модулем “display”

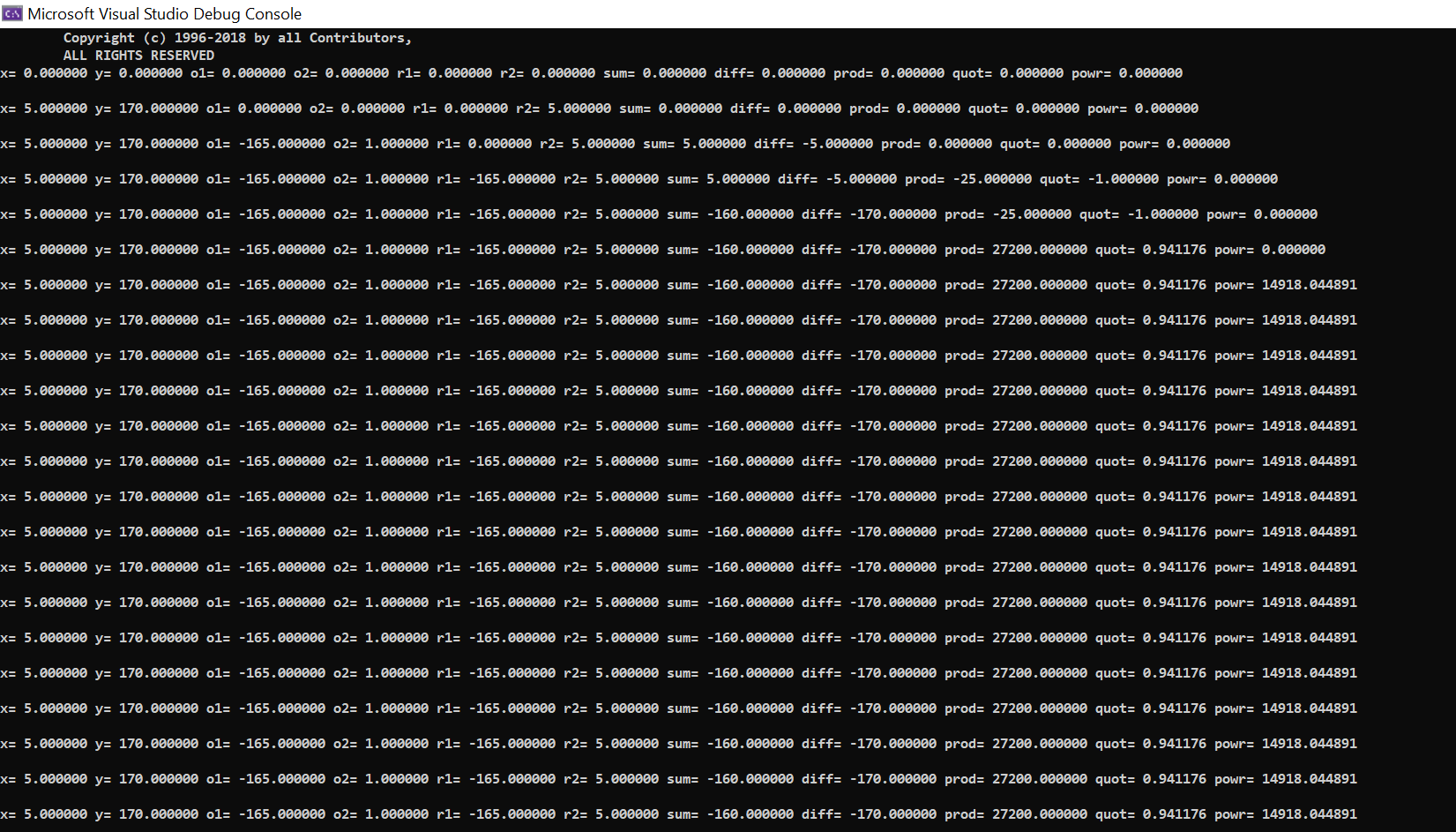


Рис. 3. Тестування програми в сукупності з модулем “observer”

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

**(Дадаток А)**

**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

void generate(); // method to write values to the output ports

//Constructor

SC\_CTOR(numgen) {

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

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

}

};

#endif

**numgen.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "numgen.h"

void numgen::generate()

{

double a = 5;

double b = 170;

out1.write(a);

out2.write(b);

}

**stage1.h**

#ifndef STAGE1\_H

#define STAGE1\_H

struct stage1 : sc\_module {

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

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

sc\_out<double> o1; //output 1

sc\_out<double> o2; //output 2

sc\_in<bool> clk; //clock

void user\_funk1(); //method implementing functionality

//Counstructor

SC\_CTOR(stage1) {

SC\_METHOD(user\_funk1);

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

}

public:

};

#endif

**stage1.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage1.h"

#include <math.h>

void stage1::user\_funk1()

{

double x\_val = x.read();

double y\_val = y.read();

o1.write(x\_val - y\_val);

double temp1 = ((x\_val + y\_val) / 2.0);

double temp2 = ((int)x\_val >> 2);

o2.write(temp1 && temp2);

}

**stage2.h**

#ifndef STAGE2\_H

#define STAGE2\_H

struct stage2 : sc\_module {

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

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

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

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

sc\_in<bool> clk; //clock

void user\_funk2();

//Counstructor

SC\_CTOR(stage2) {

SC\_METHOD(user\_funk2);

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

}

public:

};

#endif

**stage2.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage2.h"

void stage2::user\_funk2()

{

double a\_val = a.read();

double b\_val = b.read();

r1.write(a\_val \* b\_val);

r2.write(5);

}

**stage3.h**

#ifndef STAGE3\_H

#define STAGE3\_H

struct stage3 : 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(stage3) {

SC\_METHOD(addsub);

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

}

public:

};

#endif

**stage3.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage3.h"

void stage3::addsub()

{

double a;

double b;

a = r1.read();

b = r2.read();

sum.write(a + b);

diff.write(a - b);

}

**stage4.h**

#ifndef STAGE4\_H

#define STAGE4\_H

struct stage4 : 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(stage4) {

SC\_METHOD(multdiv);

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

}

};

#endif

**stage4.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage4.h"

void stage4::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);

}

**stage5.h**

#ifndef STAGE5\_H

#define STAGE5\_H

struct stage5 : 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 providing functionality

//Constructor

SC\_CTOR(stage5) {

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

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

}

};

#endif

**stage5.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "stage5.h"

void stage5::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

**display.h**

#ifndef DISPLAY\_H

#define DISPLAY\_H

struct display : sc\_module {

sc\_in<double> powr; // Common test

sc\_in<bool> clk;

void print();

SC\_CTOR(display) {

SC\_METHOD(print);

sensitive\_pos << clk;

}

public:

};

#endif

**display.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "display.h"

#include <stdio.h>

#include <iostream>

#include <iomanip>

using namespace std;

void display::print()

{

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

}

**observer.h**

#ifndef OBSERVER\_H

#define OBSERVER\_H

struct observer : sc\_module {

sc\_in<double> x; // All stages

sc\_in<double> y;

sc\_in<double> o1;

sc\_in<double> o2;

sc\_in<double> r1;

sc\_in<double> r2;

sc\_in<double> sum;

sc\_in<double> diff;

sc\_in<double> prod;

sc\_in<double> quot;

sc\_in<double> powr;

sc\_in<bool> clk;

void observe();

SC\_CTOR(observer) {

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

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

}

public:

};

#endif

**observer.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "observer.h"

#include <stdio.h>

#include <iostream>

#include <iomanip>

using namespace std;

void observer::observe()

{

printf("x= %f ", x.read());

printf("y= %f ", y.read());

printf("o1= %f ", o1.read());

printf("o2= %f ", o2.read());

printf("r1= %f ", r1.read());

printf("r2= %f ", r2.read());

printf("sum= %f ", sum.read());

printf("diff= %f ", diff.read());

printf("prod= %f ", prod.read());

printf("quot= %f ", quot.read());

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

}

**main.cpp**

#include "systemc.h"

#include "stage1.h"

#include "stage2.h"

#include "stage3.h"

#include "stage4.h"

#include "stage5.h"

#include "display.h"

#include "observer.h"

#include "numgen.h"

#define NS \* 1e-9

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

{

sc\_core::sc\_report\_handler::set\_actions("/IEEE\_Std\_1666/deprecated",

sc\_core::SC\_DO\_NOTHING);

//Signals

sc\_signal<double> x;

sc\_signal<double> y;

sc\_signal<double> o1;

sc\_signal<double> o2;

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;

numgen N("numgen");

N(x, y, clk);

stage1 S1("stage1");

S1.x(x);

S1.y(y);

S1.o1(o1);

S1.o2(o2);

S1.clk(clk);

stage2 S2("stage2");

S2.a(o1);

S2.b(o2);

S2.r1(r1);

S2.r2(r2);

S2.clk(clk);

stage3 S3("stage3");

S3(r1, r2, sum, diff, clk);

stage4 S4("stage4");

S4(sum, diff, prod, quot, clk);

stage5 S5("stage5");

S5(prod, quot, powr, clk);

// display D("display");

// D(powr, clk); // Common test

observer Obs("observer");

Obs(x, y, o1, o2, r1, r2, sum, diff, prod, quot, powr, clk); //Test all stages

sc\_start(0, SC\_NS);

for (int i = 0; i < 6; i++)

{

clk.write(0);

sc\_start(10, SC\_NS);

clk.write(1);

sc\_start(10, SC\_NS);

}

return 0;

}