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 №27

Виконав:

ст.гр. КІ-38

Швець А.Ю.

Прийняв:

Козак Н. Б.

Львів 2022 р.

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.

Хід виконання роботи:

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 = 27

  b = 83+65=148

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

  (o1 = a + b!;

  o2 = a / b; (+ check if b!= 0)

  Повторює модуль st1 з лабораторної №2.

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

  (r1 = AND (a, b);
  r2 = NOT(AND (a, b)))
  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, що відповідає програмній моделі:

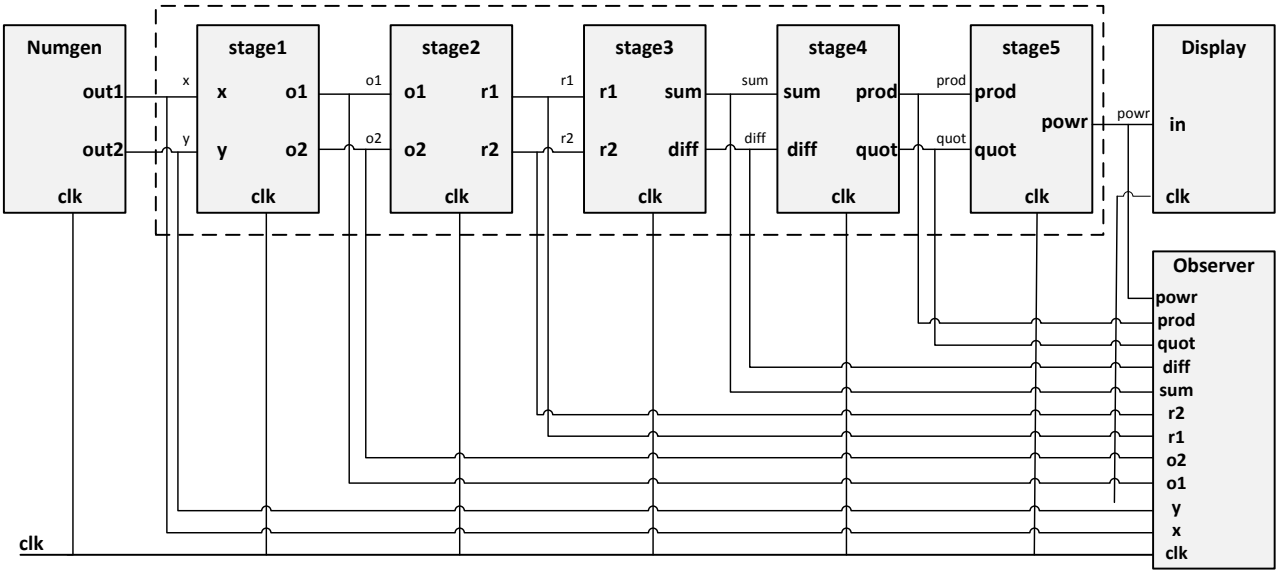

Рис.1. Структурна схема

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

Лістинг програми

Source.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_act
ions("/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;
}
```

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());
	}
```

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
```

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);
	if (y_val != 0) {
		o2.write(x_val + y_val);
	}

	}
```

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
```

Stage2.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include "systemc.h"
#include "stage2.h"

void stage2::user_funk2()
{
	int a_val = a.read();
	int b_val = b.read();
	r1.write(a_val ^ b_val);
	r2.write(!(a_val ^ b_val));
}
```

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
```

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);
	}
```

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
```

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);

	}
```

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
```

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
```

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
```

Numgen.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include "systemc.h"
#include "numgen.h"

void numgen::generate()
{
       double a = 27;//варіант
       double b = 148;//A+S
       out1.write(a);
       out2.write(b);
       }
```

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
```

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());

}
```

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
```

Результат виконання програми

```
        SystemC 2.3.3-Accellera --- Feb 21 2022 19:14:31
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
x= 0.000000 y= 0.000000 o1= 0.000000 o2= 0.000000 r1= 0.000000 r2= 0.000000 sum= 0.000000 di
ff= 0.000000 prod= 0.000000 quot= 0.000000 powr= 0.000000

x= 27.000000 y= 148.000000 o1= 1.000000 o2= 0.000000 r1= 0.000000 r2= 1.000000 sum= 0.000000
 diff= 0.000000 prod= 0.000000 quot= 0.000000 powr= 0.000000

x= 27.000000 y= 148.000000 o1= 27.000000 o2= 175.000000 r1= 1.000000 r2= 0.000000 sum= 1.000
000 diff= -1.000000 prod= 0.000000 quot= 0.000000 powr= 0.000000

x= 27.000000 y= 148.000000 o1= 27.000000 o2= 175.000000 r1= 180.000000 r2= 0.000000 sum= 1.0
00000 diff= 1.000000 prod= -1.000000 quot= -1.000000 powr= 0.000000

x= 27.000000 y= 148.000000 o1= 27.000000 o2= 175.000000 r1= 180.000000 r2= 0.000000 sum= 180
.000000 diff= 180.000000 prod= 1.000000 quot= 1.000000 powr= 0.000000

x= 27.000000 y= 148.000000 o1= 27.000000 o2= 175.000000 r1= 180.000000 r2= 0.000000 sum= 180
.000000 diff= 180.000000 prod= 32400.000000 quot= 1.000000 powr= 1.000000

x= 27.000000 y= 148.000000 o1= 27.000000 o2= 175.000000 r1= 180.000000 r2= 0.000000 sum= 180
.000000 diff= 180.000000 prod= 32400.000000 quot= 1.000000 powr= 32400.000000


D:\anrii\KI38\3.1\ks\lab4\Debug\pipe.exe (процесс 5204) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"П
араметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно…
```

Рис.2. Виконання програми

Моделювання часової діаграми в програмі gtk wave

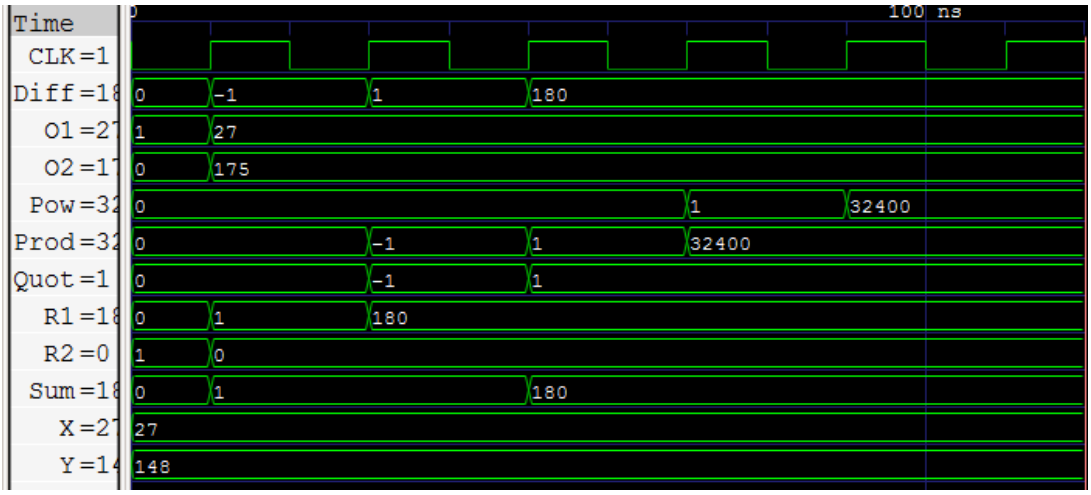

Рис.3. Часова діаграма роботи

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