**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**



Звіт

до лабораторної роботи № 3

з дисципліни «Комп’ютерні системи»

на тему: Аналіз програмної моделі процесу роботи арифметичного конвеєра, ч.1.

Варіант №6

Виконав:

ст.гр. КІ-38

Гетьман Р.Б.

Прийняв:

Козак Н.Б.

**Львів 2021**

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

**Теоретичні відомості:**

Удосконалення елементної бази вже не дає кардинального росту продуктивності обчислювальної системи. Більш перспективними у цьому плані розглядаються архітектурні рішення, серед яких одне із найбільш значимих – конвеєризація.

Для пояснення ідеї конвеєра розглянемо малюнок 1, де показаний окремий функціональний блок (ФБ). Вихідні дані завантажуються у у вхідний регістр Ргвх , обробляється у функціональному блоці, а результат обробки фіксується у вихідному регістрі Ргвих. Якщо максимальний час обробки у ФБ дорівнює *Тmax ,* то нові дані можуть бути занесені у вхідний регістр Ргвх не раніше, ніж через *Тmax.*

Рг *вх* -------→ ФБ -------------------→ Рг *вих*

Рис 1. Обробка інформації у одиночному блоці.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ргвх | → | ФБ1 | → | Рг1 | → | ФБ2 | → | Рг2 | → | ФБ3 | → | Ргвих |

Рис. 2. Обробка інформації у конвеєрі з регістрами.

Тепер розподілемо функції, що виконуються у функціональному блоці ФБ на малюнку 1 між трьома послідовними незалежними блоками: ФБ1 , ФБ2 і ФБ3, причому так, що б максимальний час обробки у кожному ФБi був однаковий і дорівнював *Tmax/3*. Між блоками розмістимо буферні регістри Ргі, що призначені для збереження результату обробки у ФБі, на випадок, якщо наступний за ним функціональний блок ще не готовий використовувати цей результат.

У розглянутій схемі дані на вхід конвеєра можуть подаватися х інтервалом *Tmax/3* (тобто, втричі частіше), і хоча затримка від моменту поступлення першої одиниці даних у Ргвх до моменту появи результату її обробки на виході Ргвих як і раніше складає *Tmax,* наступні результати з’являються на виході Ргвих вже з інтервалом *Tmax/3.*

На практиці рідко вдається добитися того, щоб затримки у кожному ФБі були однаковими. Як наслідок, продуктивність конвеєра знижується, оскільки період поступлення вхідних даних визначається максимальним часом їх обробки у кожному функціональному блоці. Для усунення цього недоліку або, в решті решт, часткової його компенсації кожний буферний регістр Ргі потрібно замінити буферною пам’яттю БПі , здатною зберігати множину даних і організований за принципом FIFO – “перший зайшов – перший вийшов”.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| БПвх | → | ФБ1 | → | БП1 | → | ФБ2 | → | БП2 | → | ФБ3 | → | БПвих |

Рис. 3. Обробка інформації у конвеєрі з буферною пам’ятю.

Обробивши елемент даних, ФБі заносить результат у БПі, витягає з БПі-1 новий елемент даних і починає черговий цикл обробки, причому ця послідовність здійснюється кожним функціональним блоком незалежно від інших блоків. Обробка у кожному блоці може продовжуватися до тих пір, доки не беде ліквідована попередня черга або доки не буде переповнена наступна черга. Якщо емність буферної пам’яті достатньо велика, різниця у часі обробки не відбивається на продуктивності, тим не менш бажано, щоб середня тривалість обробки у всіх ФБі була однаковою.

По способу синхронізації роботи ланок конвеєри можуть бути синхронні та асинхронні. Для традиційних комп’ютерів характерні *синхронні* *конвеєри.* Пов’язане це, наперед всього, із синхронним характером роботи процесорів. Ланки конвеєрів у процесорі звичайно розміщені близько одна від одної, завдяки чому тракти розповсюдження сигналів синхронізації виходять достатньо короткими і фактор “перекосу” сигналів стає не таким суттєвим.

*Асинхронні конвеєри* виявляються корисними, якщо зв’язок між ланками не така сильна, а довжина сигнальних трактів між різними ланками сильно відрізняється. Прикладом асинхронних конвеєрів можуть слугувати систолічні масиви.

**Завдання:**

1. Проаналізувати склад програмної моделі арифметичного конвеєра, (програма PIPE), яка виконана на мові System C.
2. Визначити інформаційні потоки у моделі арифметичного конвеєра.
3. Визначити зв’язки керування.
4. Накреслити блоки, з яких складається арифметичний конвеєр згідно поданої моделі.
5. Оформити звіт.

1) Програмна модель складається з 5 модулів:

* **Numgen** – модуль який генерує на свої 2 виходи числа.
* **Stage1** – модуль обчислює суму та різницю вхідних значень.
* **Stage2** – модуль обчислює добуток та частку вхідних значень.
* **Stage3** – модуль обчислює значення a в степені b (a та b вхідні сигнали).
* **Display** – модуль відображає на екрані значення з вхідного порту.

2) Потік даних починається з Numgen далі йде до Stage1, потім з Stage1 йдуть до Stage2, потім Stage2 з йдуть до Stage3, потім дані з Stage3 йдуть до Display.

3) Зв’язки керування виглядають наступним чином: Модуль Numgen впливає на Stage1. Модуль Stage1 залежить від Numgen, та впливає на Stage2. Модуль Stage2 залежить від Stage1, та впливає на Stage3. Модуль Stage3 залежить від Stage2, та впливає на Display. Модуль Display залежить від Stage3.

4) Структурна схема арифметичного конвеєра:

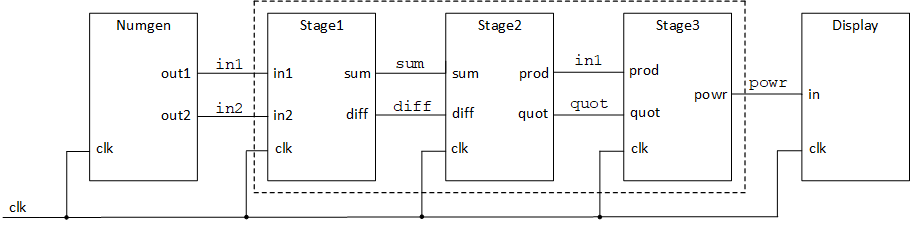


Рис. 1. Структурна схема арифметичного конвеєра

Тестування програми:

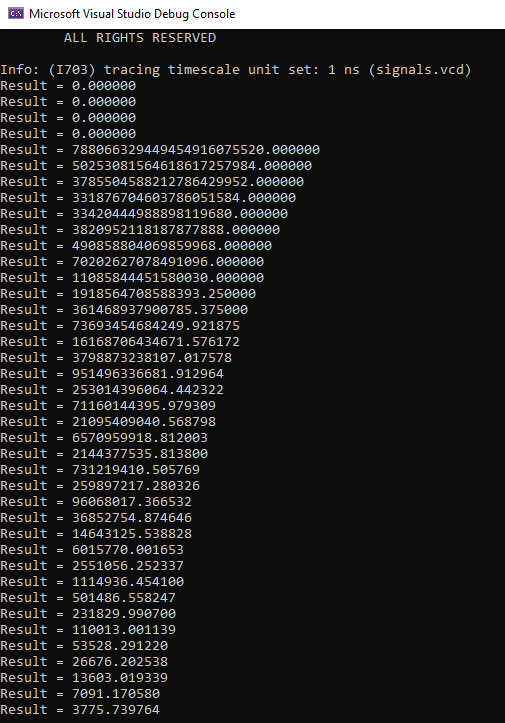
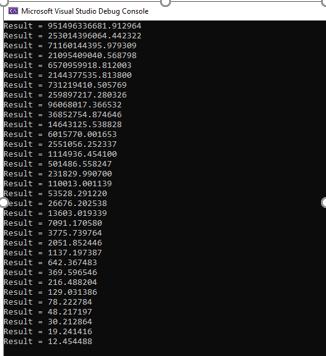
 

Рис. 2. Результат виконання програми

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

(Додаток А)

**display.h**

#include "systemc.h"

#ifndef DISPLAY\_H

#define DISPLAY\_H

//#include "stage1.h" // Added by ClassView

struct display : sc\_module {

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

sc\_in<bool> clk; // clock

sc\_in<double> diff;

void print\_result(); // method to display input port values

//Constructor

SC\_CTOR(display) {

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

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

}

public:

};

#endif

**display.cpp**

#include "systemc.h"

#include "display.h"

#include <stdio.h>

//Definition of print\_result method

void display::print\_result()

{

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

} // end of print method

**numgen.h**

#include "systemc.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

**numgen.cpp**

#include "systemc.h"

#include "numgen.h"

// definition of the `generate' method

void numgen::generate()

{

static double a = 134.56;

static double b = 98.24;

a -= 1.5;

b -= 2.8;

out1.write(a);

out2.write(b);

} // end of `generate' method

**stage1.h**

#ifndef STAGE1\_H

#define STAGE1\_H

struct stage1 : sc\_module {

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

sc\_in<double> in2; //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(stage1) {

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

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

}

public:

};

#endif

**stage1.cpp**

#include "systemc.h"

#include "stage1.h"

//Definition of addsub method

void stage1::addsub()

{

double a;

double b;

a = in1.read();

b = in2.read();

sum.write(a + b);

diff.write(a - b);

} // end of addsub method

**stage2.h**

#include "systemc.h"

#ifndef STAGE2\_H

#define STAGE2\_H

struct stage2 : sc\_module {

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

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

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

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

sc\_in<bool> clk; //clock

void multdiv(); //method providing functionality

//Constructor

SC\_CTOR(stage2) {

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

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

}

};

#endif

**stage2.cpp**

#include "systemc.h"

#include "stage2.h"

//definition of multdiv method

void stage2::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

**stage3.h**

#include "systemc.h"

#ifndef STAGE3\_H

#define STAGE3\_H

struct stage3 : 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(stage3) {

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

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

}

};

#endif

**stage3.cpp**

#include "systemc.h"

#include "stage3.h"

void stage3::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 "stage1.h"

#include "stage2.h"

#include "stage3.h"

#include "display.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> in1;

sc\_signal<double> in2;

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

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

//Named port binding

S1.in1(in1);

S1.in2(in2);

S1.sum(sum);

S1.diff(diff);

S1.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, sum, "Sum");

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

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

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

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

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

S2(sum, diff, prod, quot, clk); //Positional port binding

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

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

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

D(powr, clk, diff); //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;

}