Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра ЕОМ



Лабораторна робота №4

з дисципліни “ Комп’ютері системи ”

Варіант №15

Виконав: ст.гр. КІ-33 Радух М.М.

Прийняв:

Старший викладач:

Козак Н. Б.

Львів

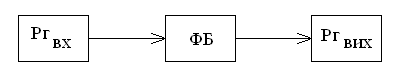
2020

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

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

# Теоретична частина

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.



#### Рис 1. Обробка інформації у одиночному блоці

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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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.

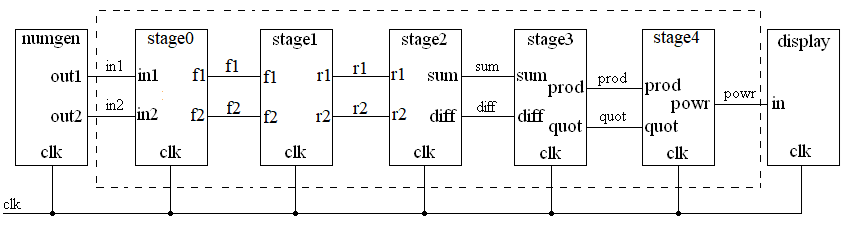
По способу синхронізації роботи ланок конвеєри можуть бути синхронні та асинхронні. Для традиційних комп’ютерів характерні синхронні конвеєри. Пов’язане це, наперед всього, із синхронним характером роботи процесорів. Ланки конвеєрів у процесорі звичайно розміщені близько одна від одної, завдяки чому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

Асинхронні конвеєри виявляються корисними, якщо зв’язок між ланками не такий сильний, а довжина сигнальних тактів між різними ланками сильно відрізняється. Прикладом асинхронних конвеєрів можуть слугувати систолічні масиви.

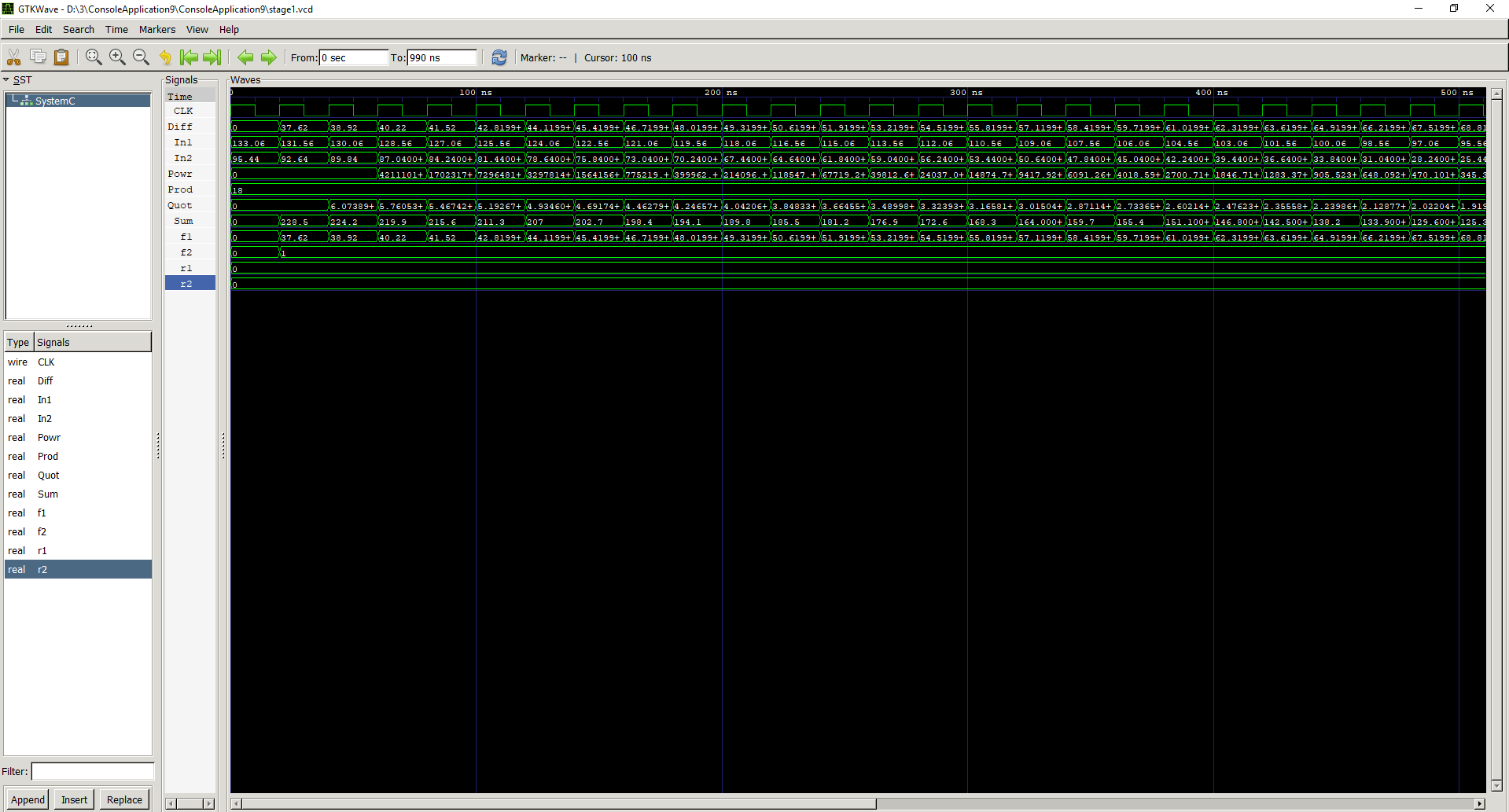
Завдання:

1. Здійснити модернізацію функцій або параметрів арифметичного конвеєра.
2. Накреслити структурну схему арифметичного конвеєра, яка відповідає програмній моделі, що аналізується.

Хід роботи:

Структурна схема арифметичного конвеєра, що відповідає програмній моделі:

#### Рис. 1 Структурна схема арифметичного конвеєра



#### Рис.2 Результат виконання програми

***Код програми:***

***display.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;

sc\_in<double> in1;

sc\_in<double> in2;

sc\_in<double> sum;

sc\_in<double> diff;// clock

sc\_in<double> prod;

sc\_in<double> quot;

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

//Constructor

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

void generate();

SC\_CTOR(numgen) {

SC\_METHOD(generate);

sensitive\_pos << clk;

}

};

#endif

***stage0.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> f1; //output 1

sc\_out<double> f2; //output 2

sc\_in<bool> clk; //clock

void adddif();

SC\_CTOR(stage0) {

SC\_METHOD(adddif);

sensitive\_pos << clk;

}

public:

};

#endif

***stage1.h***

#ifndef STAGE1\_H

#define STAGE1\_H

struct stage1 : sc\_module {

sc\_in<double> f1; //input 1

sc\_in<double> f2; //input 2

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

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

sc\_in<bool> clk; //clock

void factadd

SC\_CTOR(stage1) {

SC\_METHOD(factadd);

sensitive\_pos << clk;

}

public:

};

#endif

***stage2.h***

#ifndef STAGE2\_H

#define STAGE2\_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(); //Counstructor

SC\_CTOR(stage2) {

SC\_METHOD(addsub);

sensitive\_pos << clk; }

public:

};

#endif

***stage3.h***

#ifndef STAGE3\_H

#define STAGE3\_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();

SC\_CTOR(stage3) {

SC\_METHOD(multdiv);

sensitive\_pos << clk; }

};

#endif

***stage4.h***

#ifndef STAGE4\_H

#define STAGE4\_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();

SC\_CTOR(stage4) {

SC\_METHOD(power);

sensitive\_pos << clk; }

};

#if !defined(\_\_BCPLUSPLUS\_\_)

//what the heck is that, Amit? BCB complains. remove it

extern "C" double pow(double, double);

#endif

#endif

***display.cpp***

#include "systemc.h"

#include "display.h"

#include <stdio.h>

//Definition of print method

void display::print()

{

printf("Result = %f, clk =%d, in1 =%f, in2 =%f, sum =%f, diff =%f, prod =%f, quot =%f\n", in.read(), clk.read(), in1.read(), in2.read(), sum.read(), diff.read(), prod.read(), quot.read());

***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);

}

***stage0.cpp***

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "systemc.h"

#include "Stage0.h"

#include <math.h>

void stage0::adddif()

{

double a;

double b;

a = in1.read();

b = in2.read();

int sum = 0;

for (int i = 1; i <= b; i++)

sum += a\*i;

f1.write(pow(a-b);

f2.write((a+b/2)&&(a>>2));

}

***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);

}

***stage2.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

***stage3.cpp***

#include <math.h>

#include "systemc.h"

#include "stage3.h"

//Definition of power method

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);

***stage4.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 "stage1.h"

#include "stage2.h"

#include "stage3.h"

#include "stage4.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> f1;

sc\_signal<double> f2;

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"); //instance of `numgen' module

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

stage0 S0("stage0");

S0.in1(in1);

S0.in2(in2);

S0.f1(f1);

S0.f2(f2);

S0.clk(clk);

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\* tf;

tf = sc\_create\_vcd\_trace\_file("stage1");

((vcd\_trace\_file\*)tf)->sc\_set\_vcd\_time\_unit(-9);

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

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

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

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

sc\_trace(tf, f2, "f2");

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

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

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

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

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

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

sc\_trace(tf, powr, "Powr");

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

stage4 S4("stage4");

S4(prod, quot, powr, clk);

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

D(powr,clk,in1,in2,sum,diff,prod,quot);

//Positional port binding

sc\_initialize(); //Initialize simulation

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

clk.write(1);

sc\_cycle( 10 NS );

clk.write(0);

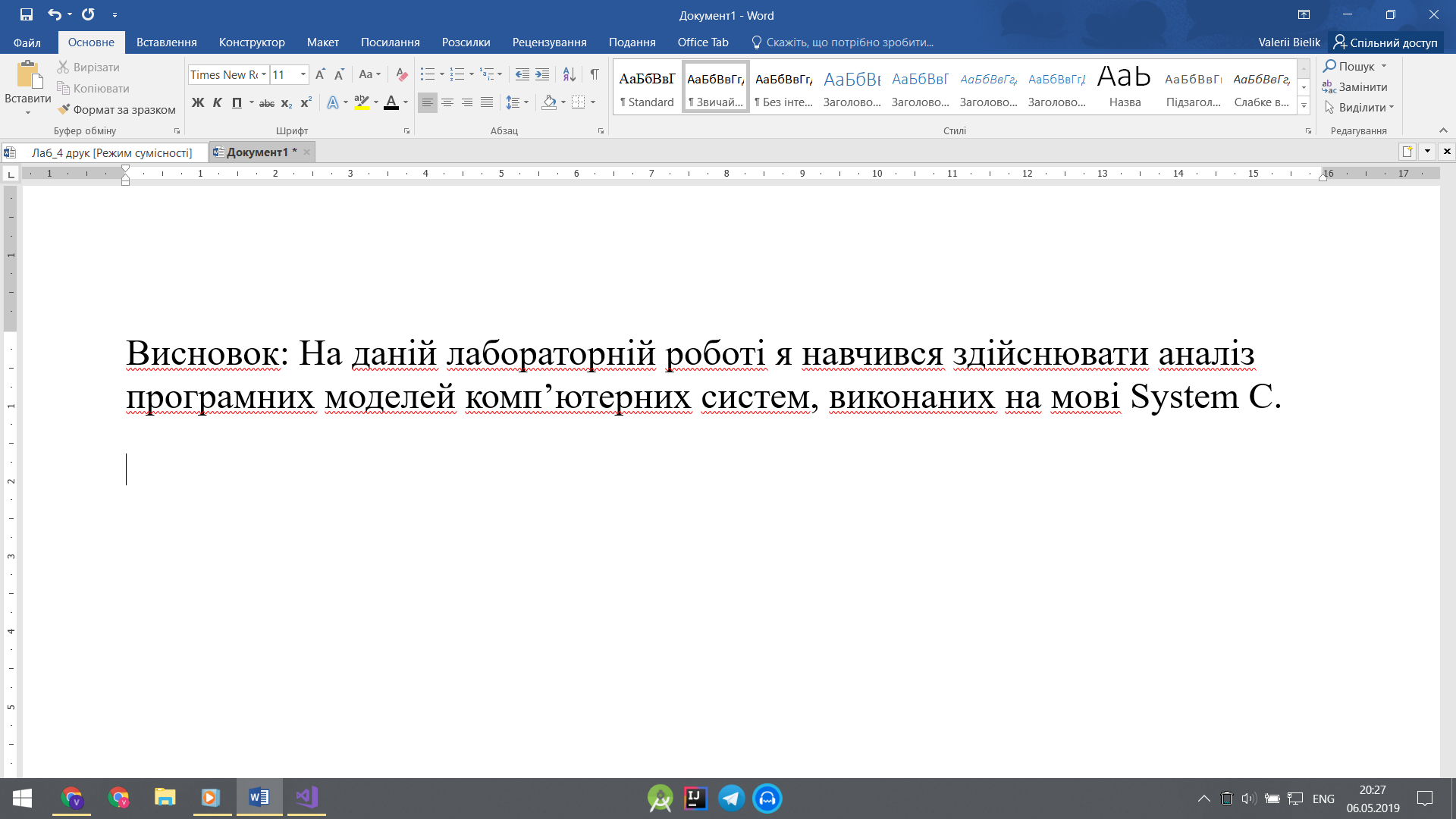
sc\_cycle( 10 NS );

}

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

return 0;

}

**