Міністерство освіти і науки

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Звіт**

з лабораторної роботи №3

з дисципліни: “ Комп’ютерні системи ”

на тему: “ Аналіз програмної моделі процесу роботи арифметичного конвеєра ”

Виконала: ст. гр. КІ-32

Селебинка Д.В.

Перевірила: викладач

Козак Н.Б.

Львів – 2020**Мета лабораторної роботи**

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

**Теоретичні відомості**

Планувальник System c

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.

**Послідовність виконання роботи**

1.Програмна модель конвеєра складається із трьох функціональних блоків, результат виконання операцій попереднього подається на вхід наступному як операнд.

2.Перший функціональний блок приймає на вхід операнди a та b. Далі виконує над ними перетворення і їх результат передає на вхід наступному блоку, який виконує над ними операцій і тд. Всього у моделі можна виділити два потоки: потік даних(a та b для першого блоку, результат виконання операцій попереднього блоку як вхідна дані для наступного) та потік керування(сигнал синхронізації clk).

3.Додактові елементи у структурі арифметичного конвеєра: блок взяття квадратного кореня від вхідної величини та взяття оберненого значення від вхідної величини.

Перелік і призначення блоків, з яких складається конвеєр:

1)блок генерування даних:



2)блок виконання сумування та віднімання:



3)блок виконання множення та ділення:



4)блок піднесення до степеня:



5)блок взяття квадратного кореня:



6)блок взяття оберненої величини:



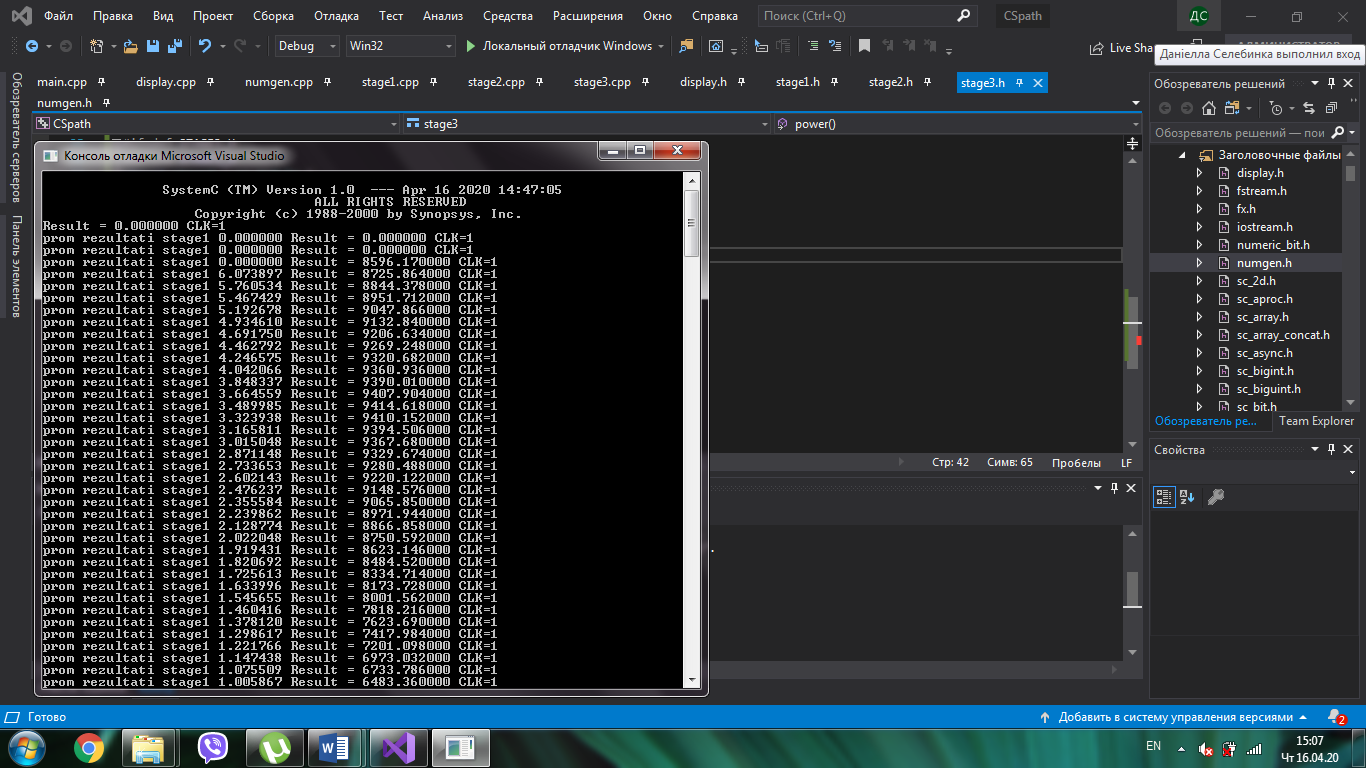
7)блок відображення результату:



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

Скрін виконання :



**Лістинг програми:**

**File main.cpp**

#include "systemc.h"

#include "stage1.h"

#include "stage2.h"

#include "stage3.h"

#include "display.h"

#include "numgen.h"

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

{

//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;

//Clock

sc\_signal<bool> clk;

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);

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); //Positional port binding

sc\_start(0, SC\_NS); //Initialize simulation

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

clk.write(1);

sc\_start( 10, SC\_NS );

clk.write(0);

sc\_start( 10, SC\_NS );

}

return 0;

}

**File display.h**

#ifndef DISPLAY\_H

#define DISPLAY\_H

struct display : sc\_module {

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

sc\_in<bool> clk; // clock

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

//Constructor

SC\_CTOR( display ) {

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

dont\_initialize();

sensitive << clk.pos(); // make it sensitive to positive clock edge

}

};

#endif

**File 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

File 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

dont\_initialize();

sensitive << clk.pos(); //make it sensitive to positive clock edge

}

};

#endif

**File 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

**File 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

dont\_initialize();

sensitive << clk.pos(); //make it sensitive to positive clock edge

}

};**File 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

**File stage2.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 port 1

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

sc\_in<bool> clk; //clock

void multdiv(); //method providing functionality

//Constructor

SC\_CTOR( stage2 ) {

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

dont\_initialize();

sensitive << clk.pos(); //make it sensitive to positive clock edge.

}

};

#endif

**File 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

**File stage3.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

dont\_initialize();

sensitive << clk.pos(); //make it sensitive to positive clock edge

}

};

#endif

**File 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);

} // end of power meth

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