Міністерство освіти і науки України

Національний університет "Львівська політехніка"

Кафедра ЕОМ

****

**Звіт**

з лабораторної роботи № 4

з дисципліни: “ Комп’ютерні системи”

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

Варіант 6

Виконав: студент .гр. КІ-33

Котик В.В.

Прийняв: асистент каф. ЕОМ

Козак Н.Б.

Львів 2020

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

**ПОРЯДОК ВИКОНАННЯ РОБОТИ**

**Завдання до лабораторної**

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.

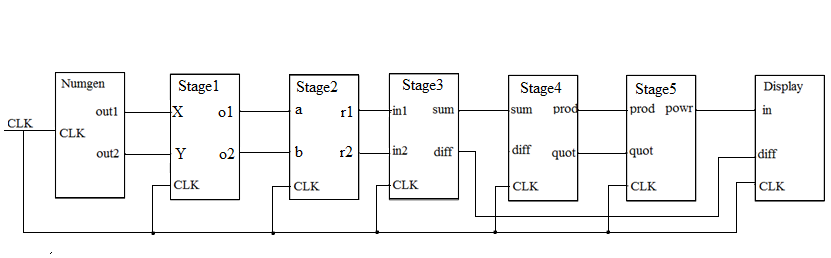


Рис.1 Схема конвеєра

**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 sub();

//Counstructor

SC\_CTOR( stage1 ) {

SC\_METHOD( sub );

sensitive\_pos << clk;

}

};

#endif

**Stage1.cpp**

#include "systemc.h"

#include "stage1.h"

//Definition of sub method

void stage1::sub()

{

double a;

double b;

a = x.read();

b = y.read();

o1.write(a-b);

o2.write((int)(a+b/2.0)&((int)a>>2));

} // end of sub method

**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 mult();

SC\_CTOR( stage2 ) {

SC\_METHOD( mult );

sensitive<<a<<b;

sensitive\_pos << clk; }

};

#endif

**Stage3.h**

#ifndef STAGE3\_H

#define STAGE3\_H

struct stage3 : 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();

//Counstructor

SC\_CTOR( stage3 ) {

SC\_METHOD( addsub );

sensitive\_pos << clk;

}

};

#endif

**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 portik 1

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

sc\_in<bool> clk; //clock

void multdiv();

//Constructor

SC\_CTOR( stage4 ) {

SC\_METHOD( multdiv );

sensitive\_pos << clk; }

};

#endif

**Stage2.cpp**

#include "systemc.h"

#include "stage2.h"

//Definition of addsub method

void stage2::mult()

{

double a;

double b;

\_a = a.read();

\_b = b.read();

r1.write(\_a\*\_b);

r2.write(6);

}

**Stage3.cpp**

#include "systemc.h"

#include "stage3.h"

//Definition of addsub method

void stage3::addsub()

{

double a;

double b;

a = in1.read();

b = in2.read();

sum.write(a+b);

diff.write(a-b);

} // end of addsub method

**Stage4.cpp**

#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();

SC\_CTOR( stage5 ){

SC\_METHOD( power

sensitive\_pos << clk; }

};

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

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

#endif

#endif

**Stage5.cpp**

#include <math.h>

#include "systemc.h"

#include "stage5.h"

void stage5::power()

{

double a;

double b;

double c;

a = prod.read();

b = quot.read();

c = pow(a, b);

powr.write(c);

} // end of power method

**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();

//Constructor

SC\_CTOR( numgen ) {

SC\_METHOD( generate );

sensitive\_pos << clk;

}

};

#endif

**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; // clock

sc\_in<double> diff;

void print();

//Constructor

SC\_CTOR( display ) {

SC\_METHOD( print );

sensitive\_pos << clk;

}

};

#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

**Display.cpp**

#include "systemc.h"

#include "display.h"

#include <stdio.h>

//Definition of print method

void display::print()

{

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

printf("prom rezultati stage1 %d ",diff.read());

} // end of print method

**main.cpp**

#include "systemc.h"

#include "stage1.h"

#include "stage2.h"

#include "stage3.h"

#include "stage4.h"

#include "stage5.h"

#include "display.h"

#include "numgen.h"

#define NS \* 1e-9

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

{

//Signals

sc\_signal<double> x;

sc\_signal<double> y;

sc\_signal<double> o1;

sc\_signal<double> o2;

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;

//Clock

sc\_clock clk("Mod", 2.0, 0.5, 0.2, false);

numgen N("numgen"); //instance of `numgen' module

N(x, y, clk ); //Positional port binding

stage1 S1("stage1");

S1(x,y,o1,o2,clk);

stage2 S2("stage2");

S2(o1,o2,in1,in2,clk);

stage3 S3("stage3"); //instance of `stage1' module

//Named port binding

S3.in1(in1);

S3.in2(in2);

S3.sum(sum);

S3.diff(diff);

S3.clk(clk);

stage4 S4("stage4"); //instance of `stage2' module

S4(sum, diff, prod, quot, clk ); //Positional port binding

stage5 S5("stage5"); //instance of `stage3' module

S5( prod, quot, powr, clk); //Positional port binding

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

D(powr,clk,diff); //Positional port binding

// Open VCD file

sc\_trace\_file\* wf = sc\_create\_vcd\_trace\_file("MainMod");

// Dump the desired signals

sc\_trace(wf, x, "x");

sc\_trace(wf, y, "y");

sc\_trace(wf, o1, "o1");

sc\_trace(wf, o2, "o2");

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, "powr");

sc\_trace(wf, clk, "clk");

sc\_initialize(); //Initialize simulation

sc\_start(clk, 20);

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

return 0;

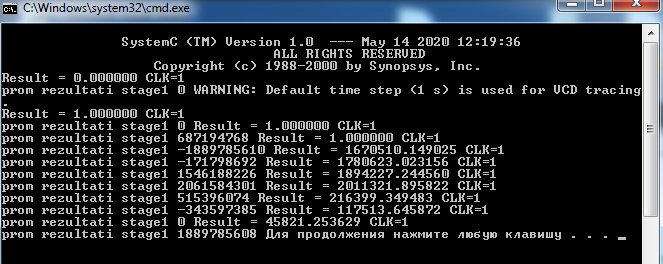


Рис.2 Результат виконання програми

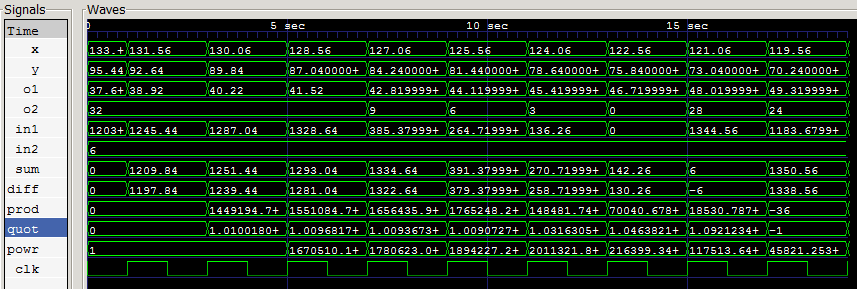


Рис.3 Графік сигналів

**Висновок:** На цій лабораторній роботі я ознайомився з основними конструкціями мови моделювання System C.