

Міністерство освіти і науки України
Національний університет «Львівська політехніка»

Кафедра ЕОМ



Звіт
до лабораторної роботи № 3
з дисципліни «Комп'ютерні системи»
на тему: «Аналіз програмної моделі процесу роботи арифметичного
конвеєра, ч.1.»
Варіант №27

Виконав:
ст.гр. КІ-38
Швець А.Ю.
Прийняв:
Козак Н. Б.

Львів 2022 р.

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

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.

Хід виконання роботи:

Перелік і призначення блоків арифметичного конвеєра:

Блок stage1 має два входи і обчислює суму вхідних даних та їх різницю.

Блок stage2 приймає дані з попереднього блоку та обчислює їх добуток та частку.

Блок stage3 приймає дані з попереднього блоку та обчислює перше вхідне число підняте до степені, що рівна другому вхідному числу

Структурна схема арифметичного конвеєра, що відповідає програмній моделі:

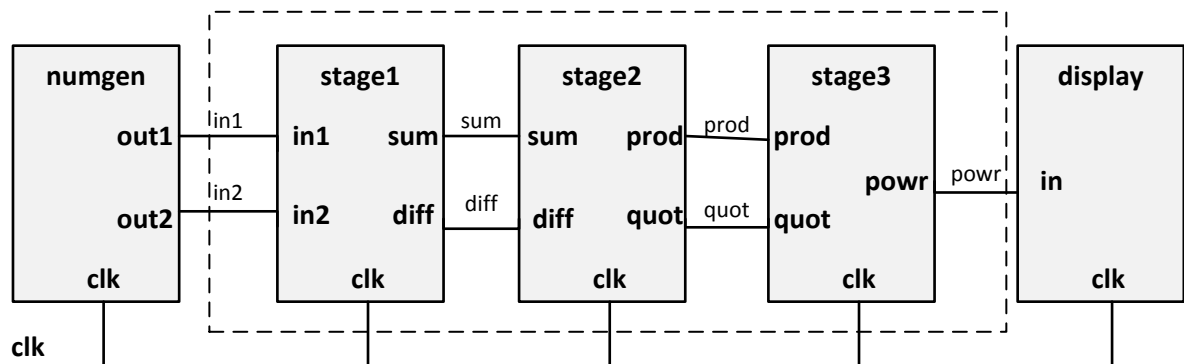


Рис.1. Структурна схема

Перелік і призначення блоків арифметичного конвеєра:

Блок stage1 має два входи і обчислює суму вхідних даних та їх різницю.

Блок stage2 приймає дані з попереднього блоку та обчислює їх добуток та частку.

Блок stage3 приймає дані з попереднього блоку та обчислює перше вхідне число підняте до степені, що рівна другому вхідному числу

Лістинг програми

Source.cpp

```
#include "systemc.h"
#include "stage1.h"
#include "stage2.h"
#include "stage3.h"
#include "display.h"
#include "numgen.h"
#define NS * 1e-5
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;
    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;
}
```

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
```

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
```

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
```

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

    //Constructor
    SC_CTOR(stage1) {
        SC_METHOD(addsub);
    //Declare addsub as SC_METHOD and
        sensitive_pos << clk; //make
it sensitive to positive clock edge
    }

public:

};

#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

```

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
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
        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

```

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

```

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

    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

```

Результат виконання програми

```

Info: (I703) tracing timescale unit set: 1 ns (signals.vcd)
Result = 0.000000
Result = 0.000000
Result = 0.000000
Result = 0.000000
Result = 788066329449454916075520.000000
Result = 50253081564618617257984.000000
Result = 3785504588212786429952.000000
Result = 331876704603786051584.000000
Result = 33420444988898119680.000000
Result = 3820952118187877888.000000
Result = 490858804069859968.000000
Result = 70202627078491096.000000
Result = 11085844451580030.000000
Result = 1918564708588393.250000
Result = 361468937900785.375000
Result = 73693454684249.921875
Result = 16168706434671.576172
Result = 3798873238107.017578
Result = 951496336681.912964
Result = 253014396064.442322
Result = 71160144395.979309
Result = 21095409040.568798
Result = 6570959918.812003
Result = 2144377535.813800
Result = 731219410.505769
Result = 259897217.280326
Result = 96068017.366532
Result = 36852754.874646
Result = 14643125.538828
Result = 6015770.001653
Result = 2551056.252337
Result = 1114936.454100
Result = 501486.558247
Result = 231829.990700
Result = 110013.001139
Result = 53528.291220
Result = 26676.202538

```

Рис.2. Виконання програми

Моделювання часової діаграми в програмі gtk wave

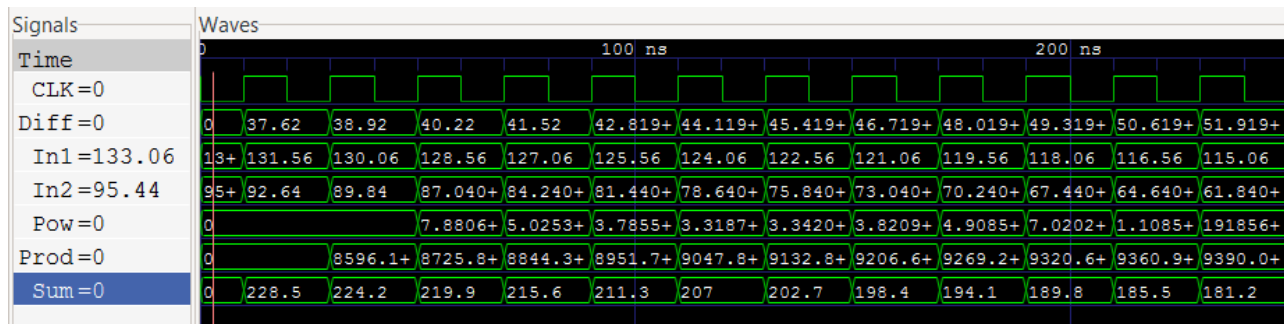


Рис.3. Часова діаграма роботи

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