**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**

**Кафедра ЕОМ**



Комп’ютерні системи

«Аналіз програмної моделі процесу роботи арифметичного конвеєра, ч.1.»

Лабораторна робота №3

Варіант №15

**Виконав:**

ст.гр. КІ-33

Радух М.М.

**Прийняв:**

Козак Н.Б.

**Львів - 2020**

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

**Хід роботи**: Проаналізувати склад програмної моделі арифметичного конвеєра, (програма PIPE), яка виконана на мові System C.

**Теорія:**

pipe () – низькорівнева функція, надає засоби передачі даних між двома програмами без накладних витрат на виклик командної оболонки для інтерпретації запитуваної команди. Ця функція також дозволить вам краще керувати читанням і записом даних.

pipe () створює односпрямований канал даних, який можна використовувати для взаємодії між процесами. Масив pipefd використовується для повернення двох файлових описателей, що вказують на кінці каналу. pipefd вказує на кінець каналу для читання. pipefd вказує на кінець каналу для запису. Дані, записані в кінець каналу, буферизується ядром до тих пір, поки не будуть прочитані з кінця каналу для читання.

Дані обробляються за принципом FIFO. Функція повертає значення 0, якщо виконання було успішним, -1 у випадку помилки.

Ініціалізація функції:

**#include <unistd.h>**  
**int pipe(int file\_descriptor[2]);**

**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.

**Інформаційні потоки у моделі арифметичного конвеєра**.

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.

**Зв’язки керування.**

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.

**Блоки, з яких складається арифметичний конвеєр згідно поданої моделі.**

\* модуль numgen:

static double a = 134.56;

static double b = 98.24;

a -= 1.5;

b -= 2.8;

записує на вихід в порти out1 - a i out2 - b.

\* модуль stage1: Зчитує значення з портів in1 та in2. Результат додавання записує в sum а відніманя в diff.

\* модуль stage2: Зчитує значення з портів sum та diff. Результат множення записує в prod а ділення в quot. Якщо значення зчитане з diff рівне 0, йому присвоюється значення 5.0.

\* модуль stage3: Зчитує значення з портів quot та prod. Результат піднесення prod до степеня quot записує в powr. Якшо якийсь з параметрів не є більшим за 0 то на вихід записується 0.

\* модуль display: Зчитує значення з порту in. Виводить його на екран.

**Структурна схему арифметичного конвеєра, яка відповідає програмній моделі**

gennum

stage1

stage2

stage3

display

**Листинг програмної моделі арифметичного конвеєра на мові System C**

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 method

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