Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологии
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ Lab4_Z1

Дисциплина: Проектирование реконфигурируемых гибридных вычислительных систем

Тема: Введение в Pipeline for Perfomance

Выполнил студент гр. 01502 С.С. Гаспарян

Руководитель, доцент Антонов А.П.

«26» октября 2021

Санкт-Петербург
2021

1. Задание

Текст задания находится в файле «Задание lab4_z1.docx»

2. Исходный код фукнции

Исходный код синтезируемой функции lab4_z1 представлен на рисунке 1.

```
1 #include "lab4_z1.h"
2
3
4 data_sc lab4_z1(data_sc inA[N], data_sc inB[N])
5 {
6     data_sc res = 0;
7     L1: for(int i = 0; i < N; ++i){
8         res += inA[i] * inB[i];
9     }
10    return res;
11 }
```

Рис. 1 Исходный код функции lab4_z1

Функция принимает два аргумента массива типа int — вычисляет скалярное произведение входных массивов и возвращает результат.

3. Исходный код теста

Исходный код теста проверки функции lab4_z1 приведен на рисунке 2.

Тест обеспечивает проверку корректной работы функции.

4. Командный файл

На рисунке 3 представлен текст команд для автоматизированного создания следующих вариантов аппаратной реализаций:

a. Для sol1 задается clock period 6: clock uncertainty 0.1

b. Для sol2 задается clock period 8. clock uncertainty 0.1

c. Для sol3 задается clock period 10. clock uncertainty 0.1

d. Для sol4 задается clock period 12. clock uncertainty 0.1

```
 5 int cmp_arr(const data_sc res, data_sc inA[N], data_sc inB[N]){
 6
 7     data_sc tmp_res = 0;
 8
 9     for (int i = 0; i < N; ++i){
10         tmp_res += inA[i] * inB[i];
11     }
12     if (res == tmp_res)
13         return 1;
14     else
15         return 0;
16 }
17
18 int main()
19 {
20     int pass=0;
21
22     // Call the function for 2 transactions
23     data_sc inA[N];
24     data_sc inB[N];
25
26     for (int i = 0; i < 3; ++i){
27         for(int j = 0; j < N; j++){
28             inA[j] = rand() % (N - 1);
29             inB[j] = rand() % (N - 1);
30         }
31
32         int res = lab4_z1(inA, inB);
33         pass = cmp_arr(res, inA, inB);
34     }
35
36
37     if (pass == 1)  {
38         fprintf(stdout, "----------Pass!-----------\n");
39         return 0;}
40     else    {
```

Рис. 2 Исходный код lab4_z1_test.c тестирования функции

```
##############################################
#                   Lab                      #
##############################################
open_project -reset lab4_z1_prj
set_top lab4_z1
add_files ./source/lab4_z1.c
add_files -tb ./source/lab4_z1_test.c

open_solution -reset sol1
create_clock -period 6 -name clk
set_clock_uncertainty 0.1
set_part {xa7a12tcsg325-1Q}

csim_design
csynth_design
cosim_design -trace_level all
##############################################
#                 Solutions                  #
##############################################
set all_solutions {sol2 sol3 sol4}
set all_periods   {{8} {10} {12}}

foreach the_solution $all_solutions the_period $all_periods {
open_solution -reset $the_solution
create_clock -period $the_period -name clk
set_clock_uncertainty 0.1
set_part {xa7a12tcsg325-1Q}

csim_design
csynth_design
cosim_design -trace_level all
}

exit
```

Рис. 3 Текст команд для создания решений

5. Сравнение решений после синтезирования функции

На рисунке 4 представлен результат сравнения отчетов для всех решений. На рисунке 5 представлена таблица с данными, где рассчитывается Latency в нс. На рисунке 6 представлен график для решений от заданных параметров. Как видно из рисунков наилучшим решением является решение sol3 с clock period 10 нс.

Он имеет наилучший результат по времени и почти самый низкое потребление аппаратных ресурсов

Performance Estimates

Timing

Clock		sol1	sol2	sol3	sol4
clk	Target	6.00 ns	8.00 ns	10.00 ns	12.00 ns
	Estimated	5.690 ns	6.860 ns	8.470 ns	11.727 ns

Latency

		sol1	sol2	sol3	sol4
Latency (cycles)	min	49153	40961	32769	24577
	max	49153	40961	32769	24577
Latency (absolute)	min	0.295 ms	0.328 ms	0.328 ms	0.295 ms
	max	0.295 ms	0.328 ms	0.328 ms	0.295 ms
Interval (cycles)	min	49153	40961	32769	24577
	max	49153	40961	32769	24577

Utilization Estimates

	sol1	sol2	sol3	sol4
BRAM_18K	0	0	0	0
DSP48E	3	3	3	3
FF	329	327	161	96
LUT	181	179	145	139
URAM	0	0	0	0

Рис. 4 Сравнение отчетов решений

		sol1	sol2	sol3	sol4
Clock	Target (ns)	6	8	10	12
	Estimated (ns)	5,69	6,86	8,47	11,73
Latency	(cycles)	49153	40961	32769	24577
	(ns)	279681	280992	277553	288214
Resources	BRAM_18K	0	0	0	0
	DSP48E	3	3	3	3
	FF	329	327	161	96
	LUT	181	179	145	139
	URAM	0	0	0	0

Рис. 5 Сравнение результатов решений в виде таблицы

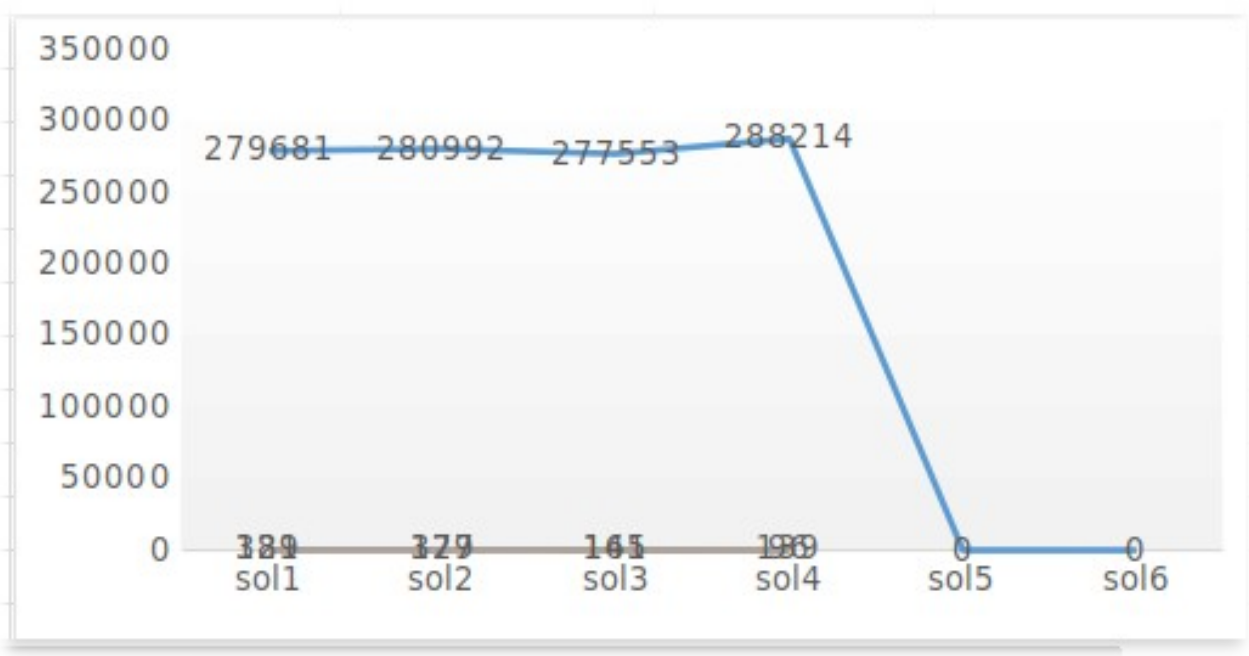

Рис. 6 График сравнения результатов решений

6. Добавление программной конвейеризации для решений

Так как наилучшее решение было для sol3, то заданный период для решений с использованием конвейеризации будет равен 10нс. На рисунке 7 и 8 представлены директивы для добавления pipeline в решения.

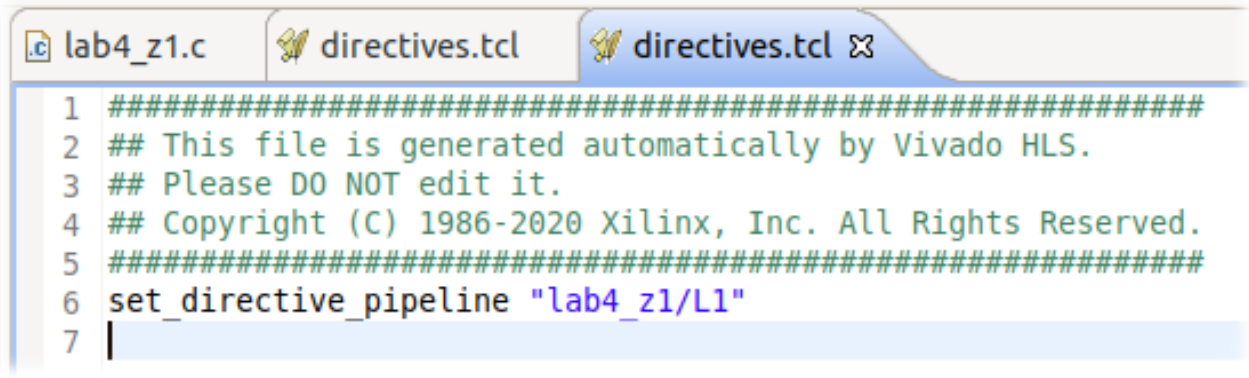

Рис. 7 Директива pipeline для sol5

lab4_z1.c directives.tcl

```
1  ############################################################
2  ## This file is generated automatically by Vivado HLS.
3  ## Please DO NOT edit it.
4  ## Copyright (C) 1986-2020 Xilinx, Inc. All Rights Reserved.
5  ############################################################
6  set_directive_pipeline -rewind "lab4_z1/L1"
7
```

Рис. 8 Директива pipeline для sol6

На рисунке 9 представлено сравнение отчетов двух решений после добавление pipeline.

Performance Estimates

Timing

Clock		sol5	sol6
clk	Target	10.00 ns	10.00 ns
	Estimated	8.470 ns	8.470 ns

Latency

		sol5	sol6
Latency (cycles)	min	8196	8194
	max	8196	8195
Latency (absolute)	min	81.960 us	81.940 us
	max	81.960 us	81.950 us
Interval (cycles)	min	8196	8192
	max	8196	8192

Utilization Estimates

	sol5	sol6
BRAM_18K	0	0
DSP48E	3	3
FF	214	224
LUT	187	195
URAM	0	0

Рис. 9 Сравнение отчетов для pipeline решений

Как видно рисунка 9 разница между этими двумя решениями не велика по все параметрам. На рисунке 10 представлена таблица сравнения всех решений и на рисунке 11 представлен график с временными и аппаратными затратами. Как видно из рисунков добавление pipeline значительно добавляет к производительности системы, но при этом увеличивает потребление аппаратных ресурсов.

		sol1	sol2	sol3	sol4	sol5	sol6
Clock	Target (ns)	6	8	10	12	10	10
	Estimated (ns)	5,69	6,86	8,47	11,73	8,47	8,47
Latency	(cycles)	49153	40961	32769	24577	8196	8194
	(ns)	279681	280992	277553	288214	69420	69403
Resources	BRAM_18K	0	0	0	0	0	0
	DSP48E	3	3	3	3	3	3
	FF	329	327	161	96	214	224
	LUT	181	179	145	139	187	195
	URAM	0	0	0	0	0	0

Рис. 10 Сравнение результатов всех решений в виде таблицы

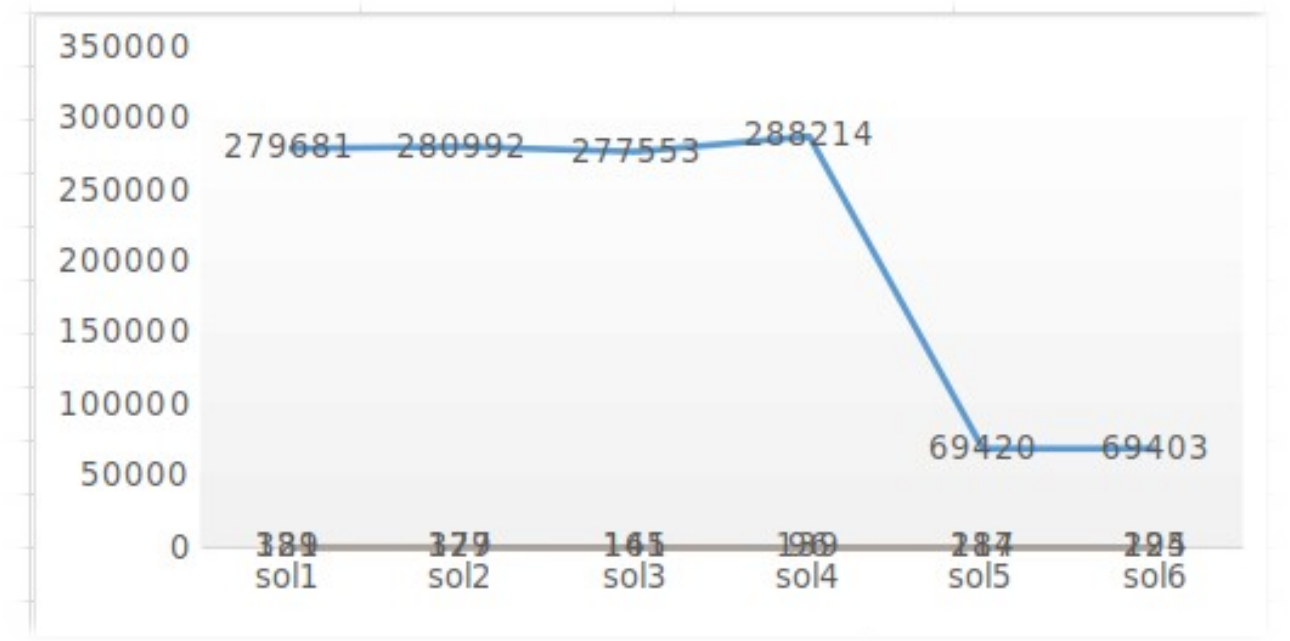

Рис. 11 График сравнения результатов всех решений

7. Исходный код модифицированного теста

Исходный код модифицированного теста для проверки функции lab4_z1 приведен на рисунке 12. Тест обеспечивает проверку производительности функции на ПК. Функция была скомпилирована компилятором gcc-9.3.0. В

таблице 1 представлены характеристики ПК:

Таблица 1

CPU	Intel Core i5-6200U 2.3 GHz
Core	2
Threads	4
RAM	8 Gb

```
19 int main()
20 {
21     int pass=0;
22
23     // Call the function for 32 transactions
24     data_sc inA[N];
25     data_sc inB[N];
26     struct timespec t0, t1;
27     double acc_time = 0.0;
28
29     for (int i = 0; i < 32; ++i){
30         for(int j = 0; j < N; j++){
31             inA[j] = rand() % (N - 1);
32             inB[j] = rand() % (N - 1);
33         }
34
35         if(clock_gettime(CLOCK_REALTIME, &t0) != 0) {
36             perror("Error in calling clock_gettime\n");
37             exit(EXIT_FAILURE);
38         }
39         int res = lab4_z1(inA, inB);
40         if(clock_gettime(CLOCK_REALTIME, &t1) != 0) {
41             perror("Error in calling clock_gettime\n");
42             exit(EXIT_FAILURE);
43         }
44         double diff_time = (((double)(t1.tv_sec - t0.tv_sec))*1000000000.0) + (double)(t1.tv_nsec - t0.tv_nsec);
45         acc_time += diff_time;
46         double temp_avg_time = acc_time / (i + 1); // take average time
47         printf("Elapsed time: %.4lf nanoseconds\n", temp_avg_time);
48
49         pass = cmp_arr(res, inA, inB);
50     }
51
52
53     if (pass == 1)  {
```

Рис. 12 Исходный код тестирования функции для исполнения на ПК

На рисунке 13 представлены результаты запуска функции на ПК. Как видно из рисунка среднее время выполнения функции после 32 итерации равно 29128,59 нс, что почти в 3 раза быстрее, чем решение полученное при синтезировании функции.

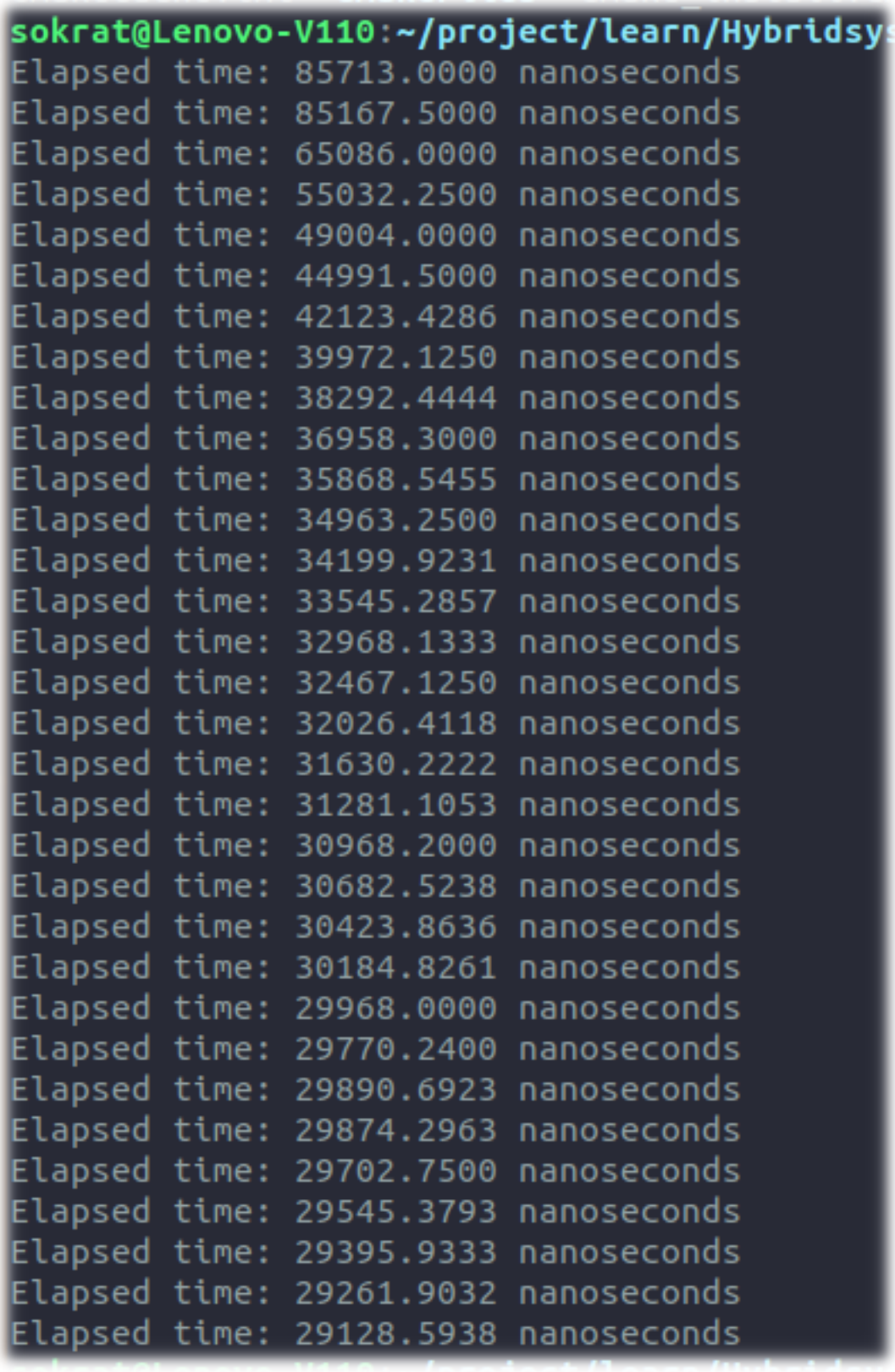

Рис. 13 Временные показатели для модифицированного теста

Вывод

В данной работе была изучена возможность добавления pipeline директивы для синтезируемой функции. Был произведен сравнительный анализ между решением без добавлением и с добавлением pipeline. Также было произведено сравнение временных показателей между решением полученным Vivado HLS и тестированием решения на ПК. Как видно из результатов решением полученное на ПК быстрее, чем решением после синтезирования в Vivado HLS.