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

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ Lab4_Z3

Дисциплина: Проектирование реконфигурируемых гибридных вычислительных систем

Тема: Введение в Pipeline of Perfomance Dataflow

Выполнил студент гр. 01502 С.С. Гаспарян

Руководитель, доцент Антонов А.П.

«7» ноября 2021

Санкт-Петербург
2021

1. Задание

Текст задания находится в файле «Задание lab4_z3.docx»

2. Исходный код фукнции

Исходный код синтезируемых функции foo_b и foo_m представлен на рисунке 1.

```
4  void foo_b(int data_in[N], int scale[3], int data_out1[N], int data_out2[N])
5  {
6      int temp1[N];
7      Loop1: for(int i = 0; i < N; i++) {
8                  temp1[i] = data_in[i] * scale[0];
9              }
10     Loop2: for(int j = 0; j < N; j++) {
11                 data_out1[j] = temp1[j] * scale[1];
12             }
13     Loop3: for(int k = 0; k < N; k++) {
14                 data_out2[k] = temp1[k] * scale[2];
15             }
16
17 }
18
19 void Split(int in[N], int out1[N], int out2[N]) {
20 // Duplicated data
21     L1: for(int i=1;i<N;i++) {
22         out1[i] = in[i];
23         out2[i] = in[i];
24     }
25 }
26
27
28 void foo_m(int data_in[N], int scale[3], int data_out1[N], int data_out2[N])
29 {
30     int temp1[N], temp2[N], temp3[N];
31     Loop1: for(int i = 0; i < N; i++) {
32         temp1[i] = data_in[i] * scale[0];
33     }
34     Split(temp1, temp2, temp3);
35     Loop2: for(int j = 0; j < N; j++) {
36         data_out1[j] = temp2[j] * scale[1];
37     }
38     Loop3: for(int k = 0; k < N; k++) {
39         data_out2[k] = temp3[k] * scale[2];
40     }
41 }
```

Рис. 1 Исходный код функции foo_b и foo_m

Функции принимает четыре аргумента массива типа int — вычисляет для входного массива произведение на элементы второго массива и записывает результаты в выходные массивы результат.

3. Исходный код теста

Исходный код теста проверки функции foo_b и foo_m приведен на рисунке 2. Тест обеспечивает проверку корректной работы функции.

```
int cmp_arr(int data_in[N], int scale[3], int data_cmp1[N], int data_cmp2[N]) {
    int temp1[N];
    for(int i = 0; i < N; i++) {
        temp1[i] = data_in[i] * scale[0];
    }
    for(int j = 0; j < N; j++) {
        int tmp = temp1[j] * scale[1];
        if (data_cmp1[j] != tmp) {
            return 0;
        }
    }
    for(int k = 0; k < N; k++) {
        int tmp = temp1[k] * scale[2];
        if (data_cmp2[k] != tmp) {
            return 0;
        }
    }
    return 1;
}

int main()
{
    int pass=0;

    // Call the function for 3 transactions
    int scale[3];
    int data_in[N];
    int data_out1[N];
    int data_out2[N];

    for (int i = 0; i < 3; ++i){
        for(int j = 0; j < N; j++){
            data_in[j] = rand() % (N - 1);
        }
        for(int k = 0; k < 3; ++k){
            scale[k] = rand() % ((N >> 1) - 1);
        }

        foo_b(data_in, scale, data_out1, data_out2);
        pass = cmp_arr(data_in, scale, data_out1, data_out2);
        if (pass == 0) {
            fprintf(stderr, "----------Fail!-----------\n");
```

Рис. 2 Исходный код lab4_z3_test.c тестирования 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

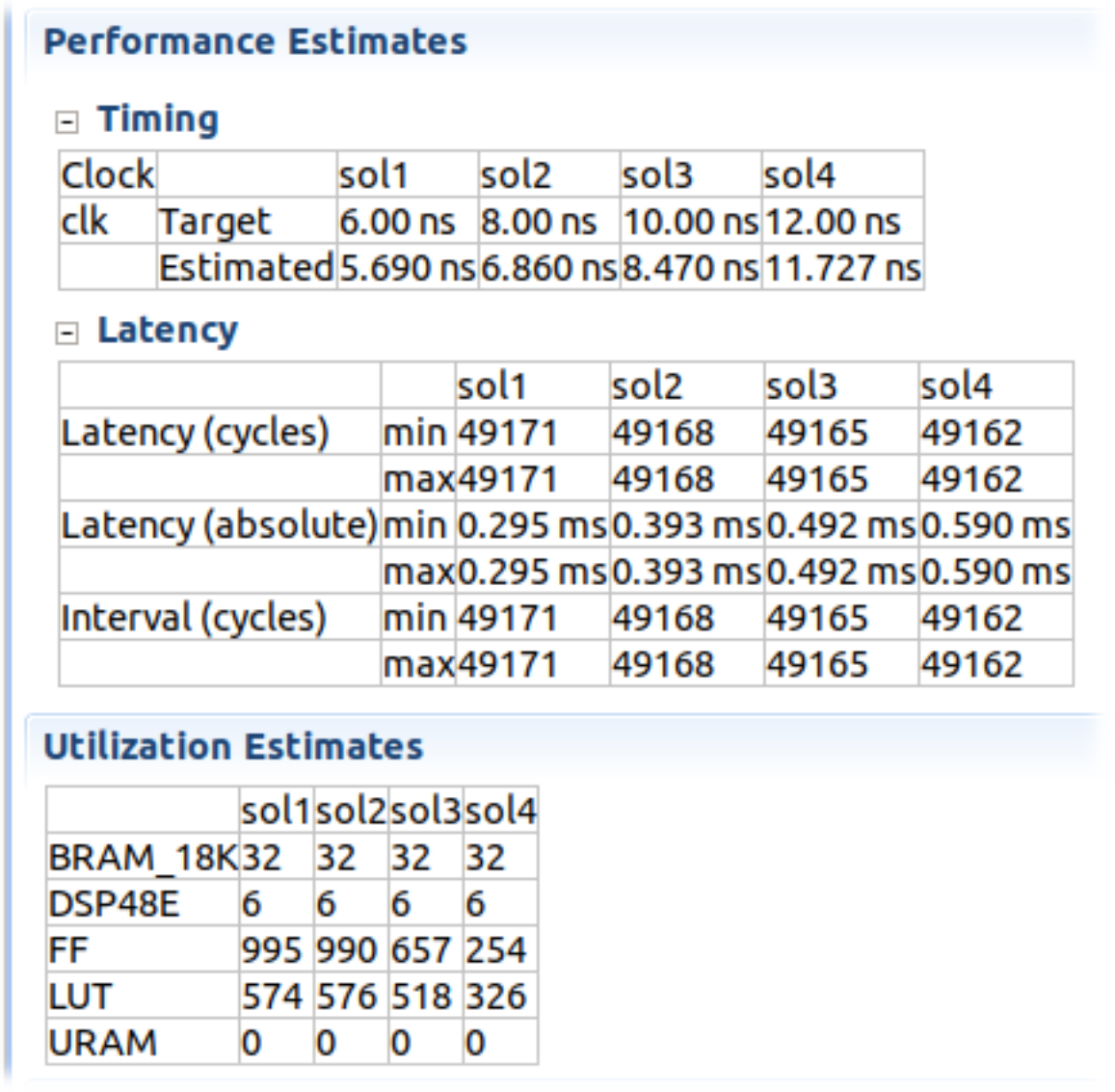
d. Для sol4 задается clock period 12. clock uncertainty 0.1

```
############################################
#                   Lab                    #
############################################
open_project -reset lab4_z3b_prj
set_top foo_b
add_files ./source/lab4_z3.c
add_files -tb ./source/lab4_z3_test.c

open_solution -reset sol1
create_clock -period 6 -name clk
set_clock_uncertainty 0.1
set_part {xa7a12tcsg325-1Q}
source ./directives_foob.tcl

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}
source ./directives_foob.tcl

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 и график для решений от заданных параметров. Как видно из рисунков наилучшим решением является решение sol3 с clock period 10 нс. Он имеет приемлемый 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	49171	49168	49165	49162
	max	49171	49168	49165	49162
Latency (absolute)	min	0.295 ms	0.393 ms	0.492 ms	0.590 ms
	max	0.295 ms	0.393 ms	0.492 ms	0.590 ms
Interval (cycles)	min	49171	49168	49165	49162
	max	49171	49168	49165	49162

Utilization Estimates

	sol1	sol2	sol3	sol4
BRAM_18K	32	32	32	32
DSP48E	6	6	6	6
FF	995	990	657	254
LUT	574	576	518	326
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)	49171	49168	49165	49162
	(ns)	279783	337292	416428	576523
Resources	BRAM_18K	32	32	32	32
	DSP48E	6	6	6	6
	FF	995	990	657	254
	LUT	574	576	518	326
	URAM	0	0	0	0

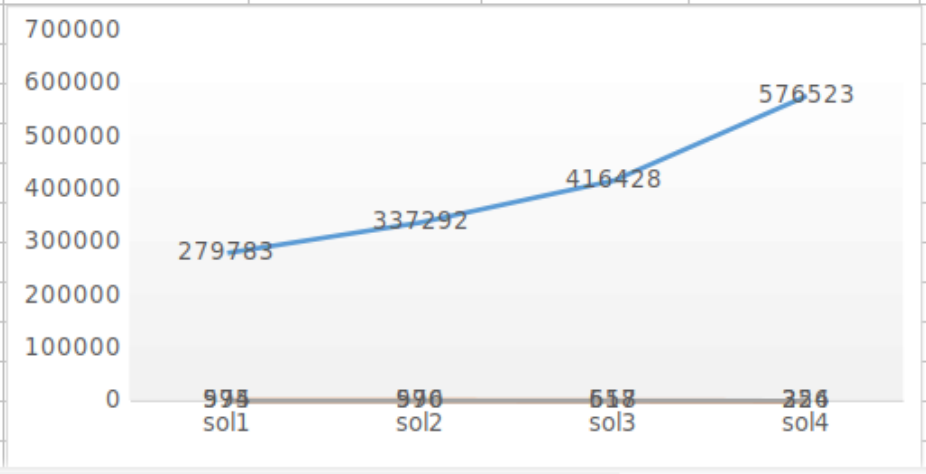


Рис. 5 Сравнение результатов решений в виде таблицы и графика

Далее на рисунке 6 и 7 представлены schedule и resource viewer соотвественно

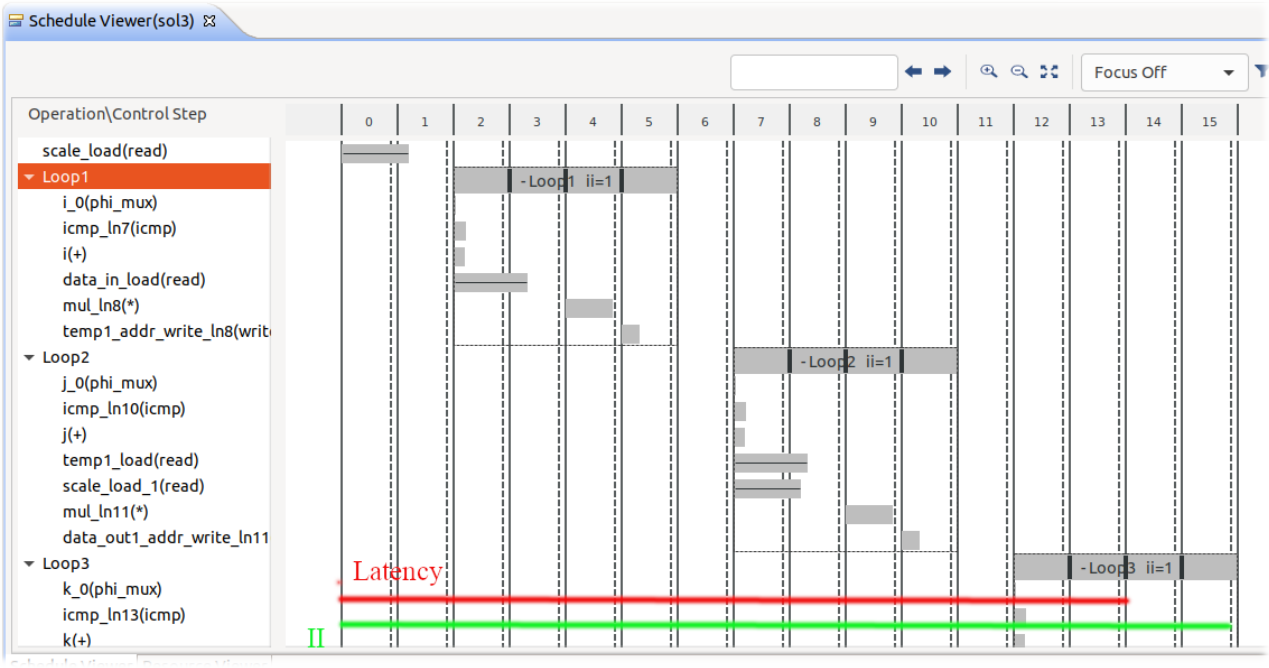


Рис. 6 Schedule viewer для функции foo_b

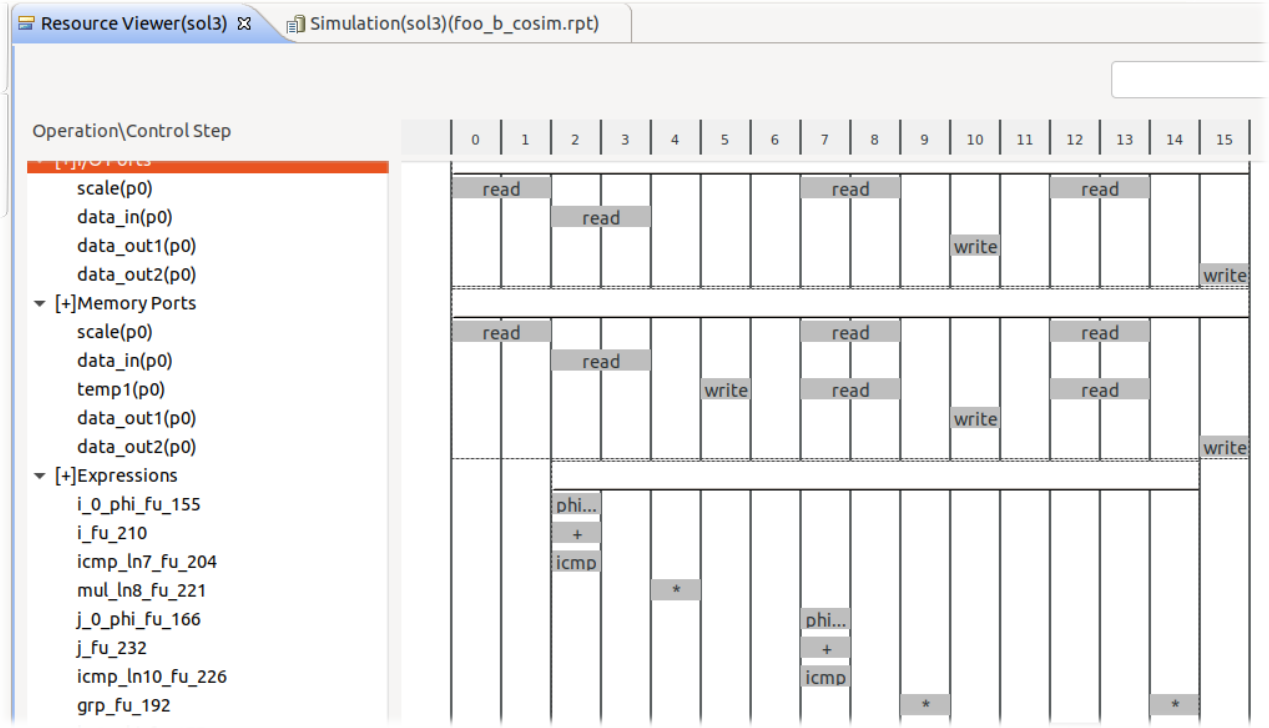


Рис. 7 Resource viewer для функции foo_b

6. Исходный код модифицированного теста

Исходный код модифицированного теста для проверки функции foo_b приведен на рисунке 8. 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

```
31    // Call the function for 32 transactions
32    int scale[3];
33    int data_in[N];
34    int data_out1[N];
35    int data_out2[N];
36    struct timespec t0, t1;
37    double acc_time = 0.0;
38
39    for (int i = 0; i < 32; ++i){
40        for(int j = 0; j < N; j++){
41            data_in[j] = rand() % (N - 1);
42        }
43        for(int k = 0; k < 3; ++k){
44            scale[k] = rand() % ((N >> 1) - 1);
45        }
46
47
48        if(clock_gettime(CLOCK_REALTIME, &t0) != 0) {
49            perror("Error in calling clock_gettime\n");
50            exit(EXIT_FAILURE);
51        }
52        foo_m(data_in, scale, data_out1, data_out2);
53        if(clock_gettime(CLOCK_REALTIME, &t1) != 0) {
54            perror("Error in calling clock_gettime\n");
55            exit(EXIT_FAILURE);
56        }
57        double diff_time = (((double)(t1.tv_sec - t0.tv_sec))*1000000000.0) + (double)(t1.tv_nsec - t0.tv_nsec);
58        acc_time += diff_time;
59        double temp_avg_time = acc_time / (i + 1); // take average time
60        printf("Elapsed time: %.4lf nanoseconds\n", temp_avg_time);
61
62        pass = cmp_arr(data_in, scale, data_out1, data_out2);
63        if (pass == 0) {
64            fprintf(stderr, "----------Fail!-----------\n");
65            return 1;
66        }
```

Рис. 8 Исходный код тестирования функции для исполнения на ПК

На рисунке 9 представлены результаты запуска функции на ПК. Как видно из рисунка среднее время выполнения функции после 32 итерации равно 160787.93 нс, что почти в 2.5 раза быстрее, чем решение полученное при синтезировании функции.

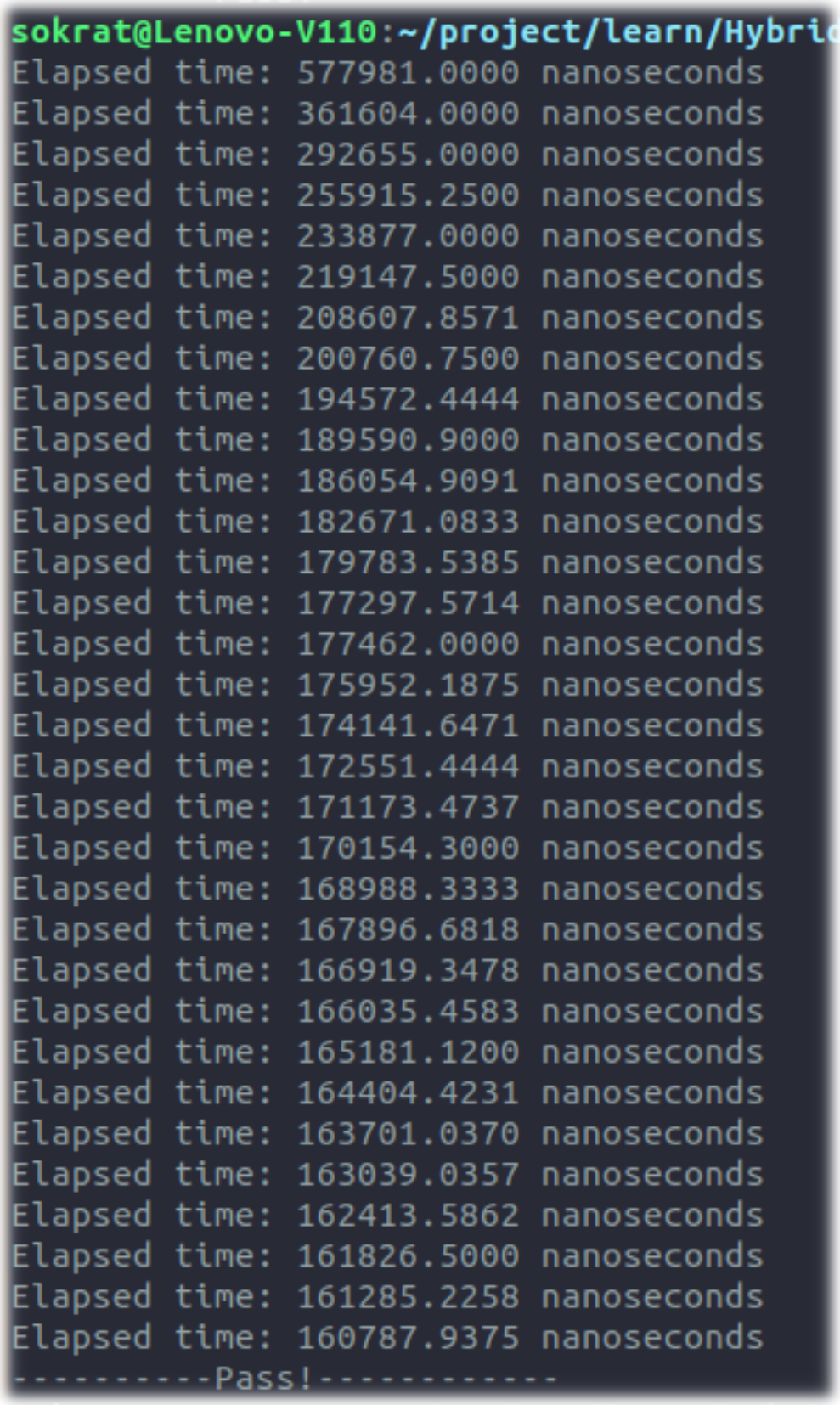


Рис. 9 Временные показатели для модифицированного теста для foo_b

7. Добавление dataflow конвейеризации для решения

7.1 Dataflow с FIFO memory buffer

На рисунке 10 представлен результат синтезирования функции foo_m для FIFO memory buffers в виде perfomance и utilization estimates соответственно.

Timing

Summary

Clock	Target	Estimated	Uncertainty
clk	10.00 ns	8.470 ns	0.10 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
16395	16395	0.164 ms	0.164 ms	16389	16389	dataflow

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	36	-
FIFO	0	-	30	252	-
Instance	0	9	766	794	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	0	9	802	1118	0
Available	40	40	16000	8000	0
Utilization (%)	0	22	5	13	0

Рис. 10 Результат синтезирования функции после pipeline dataflow

Далее на рисунке 11 и 12 представлены schedule и resource viewer соотвственно.

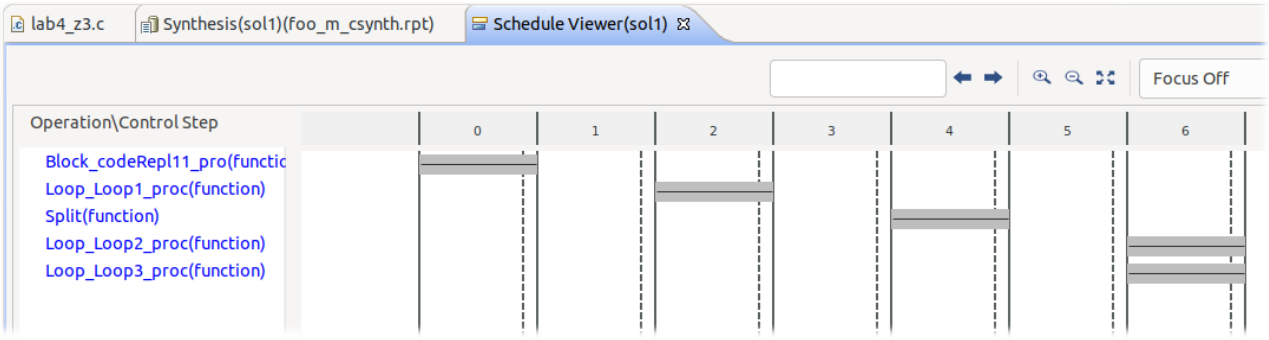


Рис. 11 Schedule viewer для функции foo_m

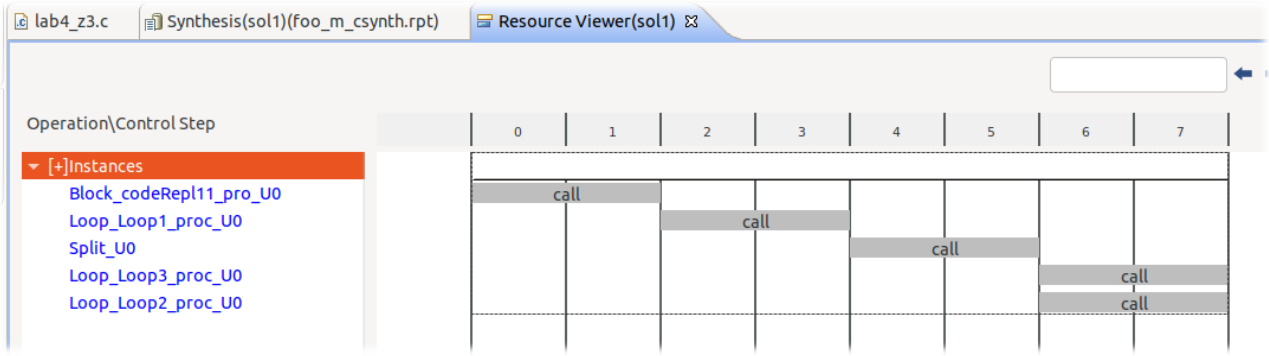


Рис. 12 Resource viewer для функции foo_m

На рисунке 13 представлена временная диаграмма для этого решения с несколькими тактами работы функции после выполнения C/RTL cosimulation.

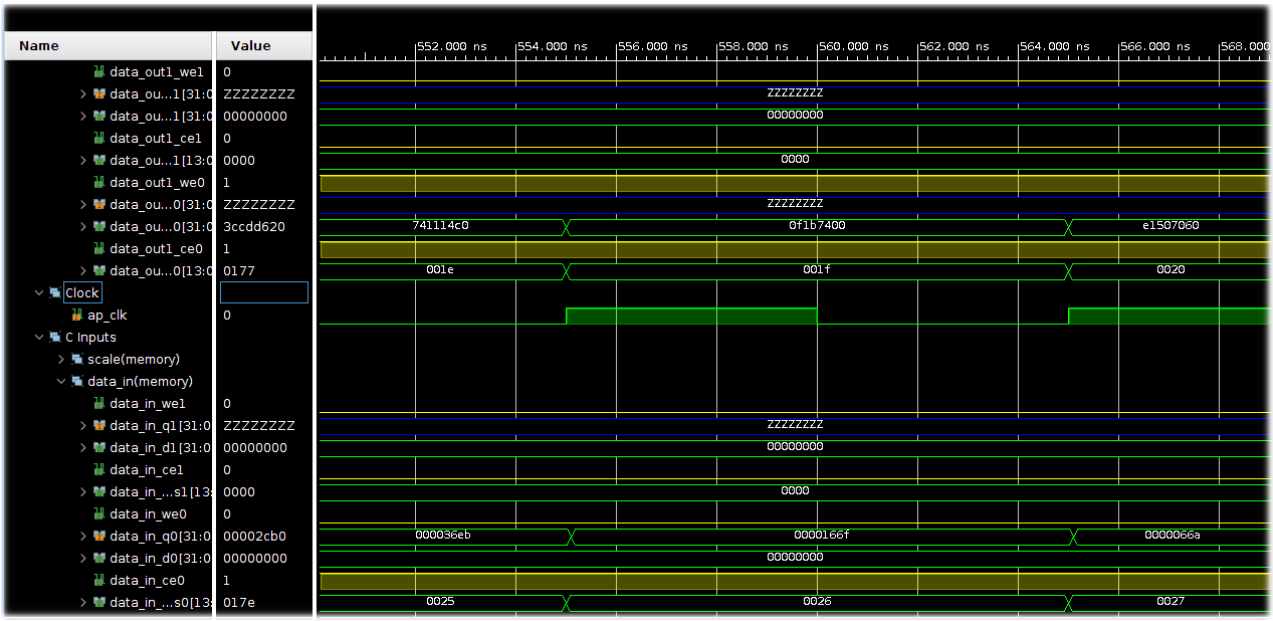


Рис. 13 Временная диаграмма для функции foo_m

На рисунке 14 и 15 представлен dataflow viewer для решения FIFO memory buffer.

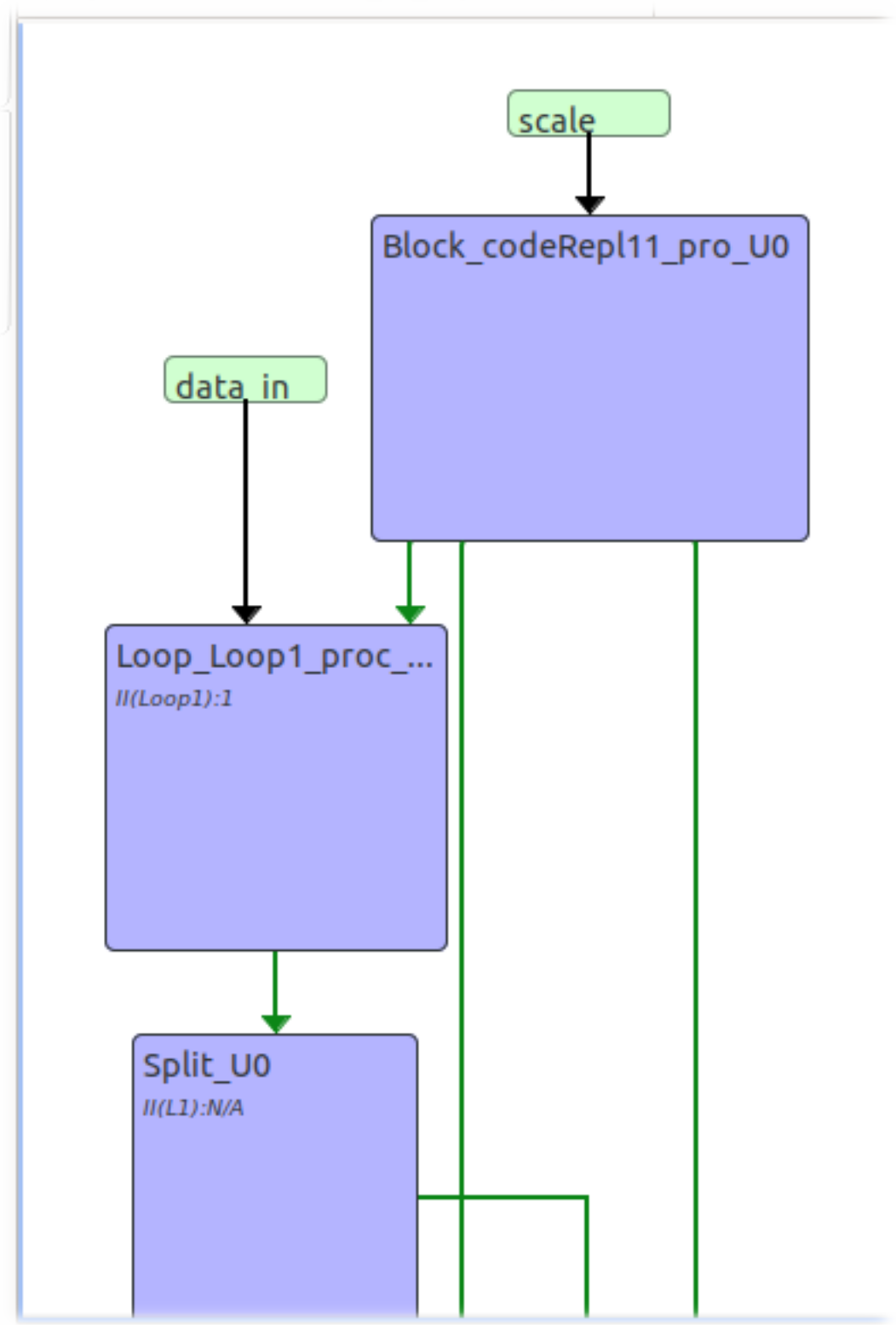


Рис. 14 Dataflow viewer FIFO для функции foo_m часть 1

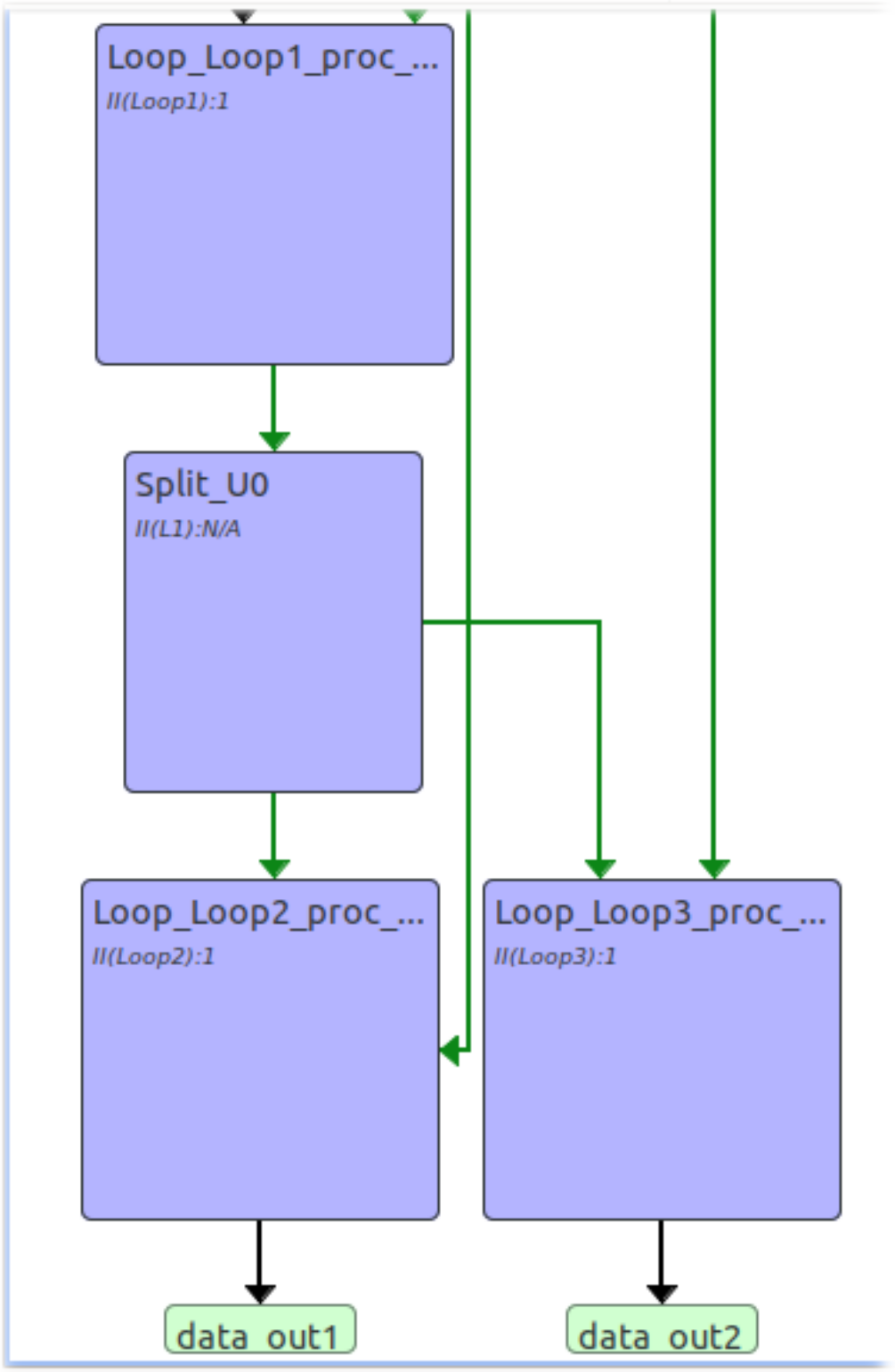


Рис. 15 Dataflow viewer FIFO для функции foo_m часть 2

7.2 Dataflow с ping-pong memory buffer

На рисунке 16 представлен результат синтезирования функции foo_m для ping-pong memory buffers в виде perfomance и utilization estimates соответственно.

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
clk	10.00 ns	8.470 ns	0.10 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
65547	65547	0.655 ms	0.655 ms	32770	32770	dataflow

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	44	-
FIFO	0	-	15	126	-
Instance	0	9	842	710	-
Memory	96	-	0	0	0
Multiplexer	-	-	-	54	-
Register	-	-	8	-	-
Total	96	9	865	934	0
Available	40	40	16000	8000	0

Рис. 16 Результат синтезирования функции после pipeline dataflow

Как видно из рисунка для ping-pong memory buffer результат стал хуже по времени и по аппаратным ресурсам в сравнений с другим решением.

Далее на рисунке 17 и 18 представлены schedule и resource viewer соотвственно.

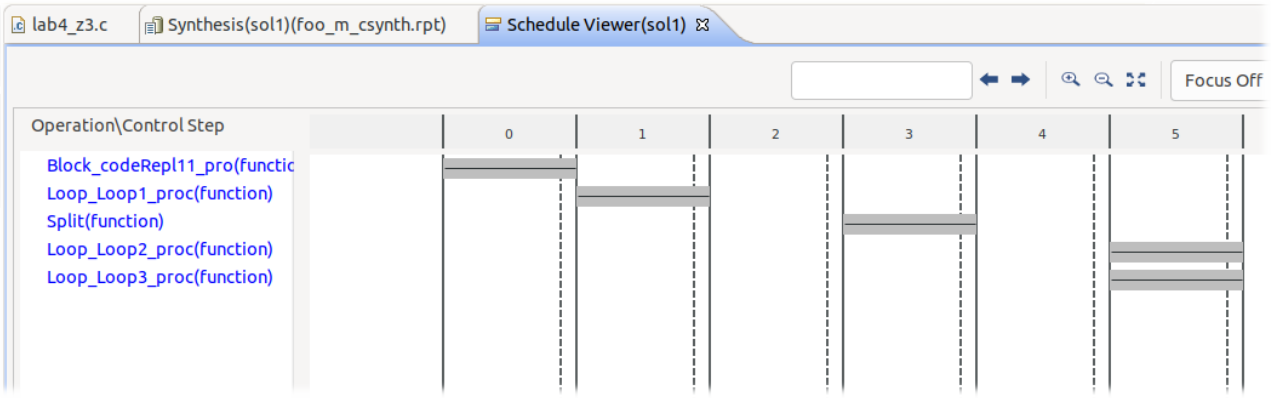


Рис. 17 Schedule viewer для функции foo_m

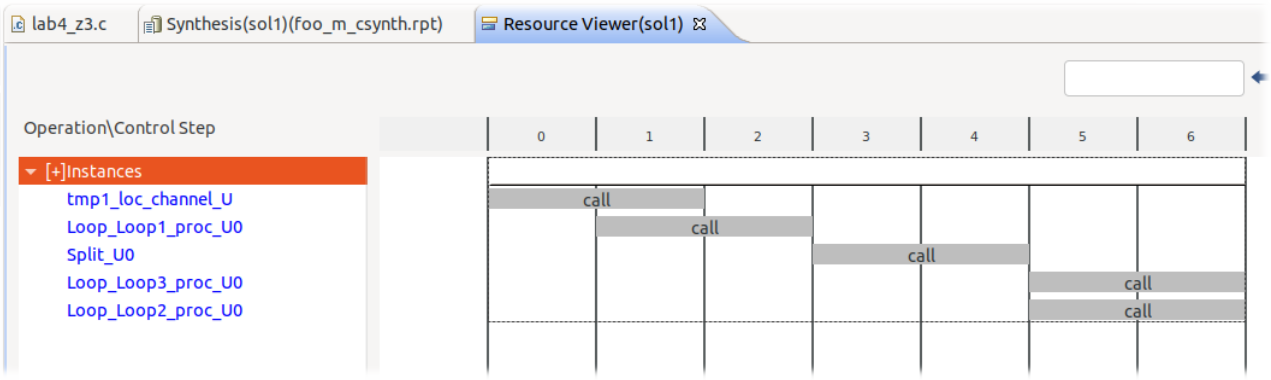


Рис. 18 Resource viewer для функции foo_m

На рисунке 19 и 20 представлен dataflow viewer для решения ping-pong memory buffer.

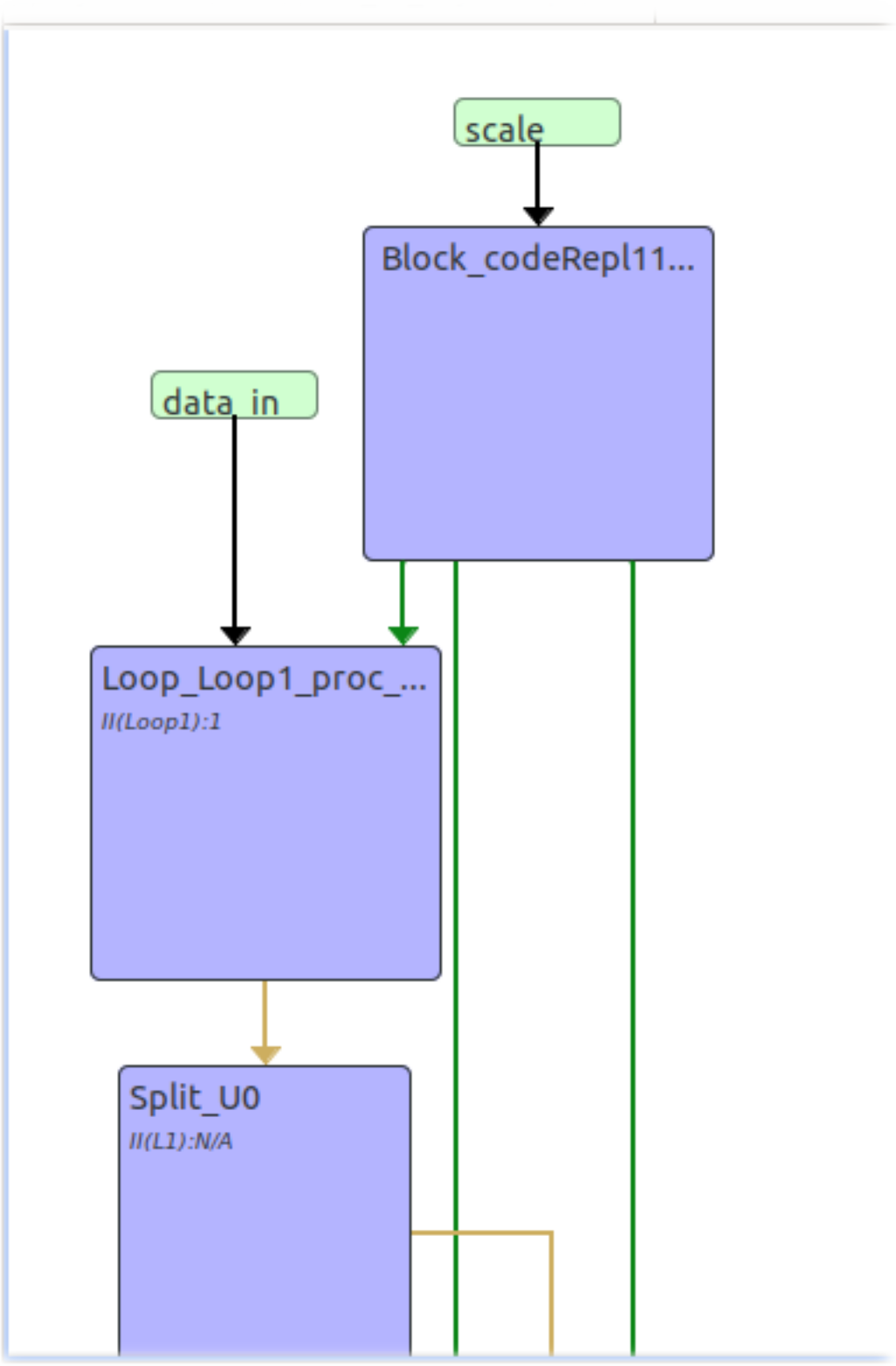


Рис. 19 Dataflow viewer ping-pong для функции foo_m часть 1

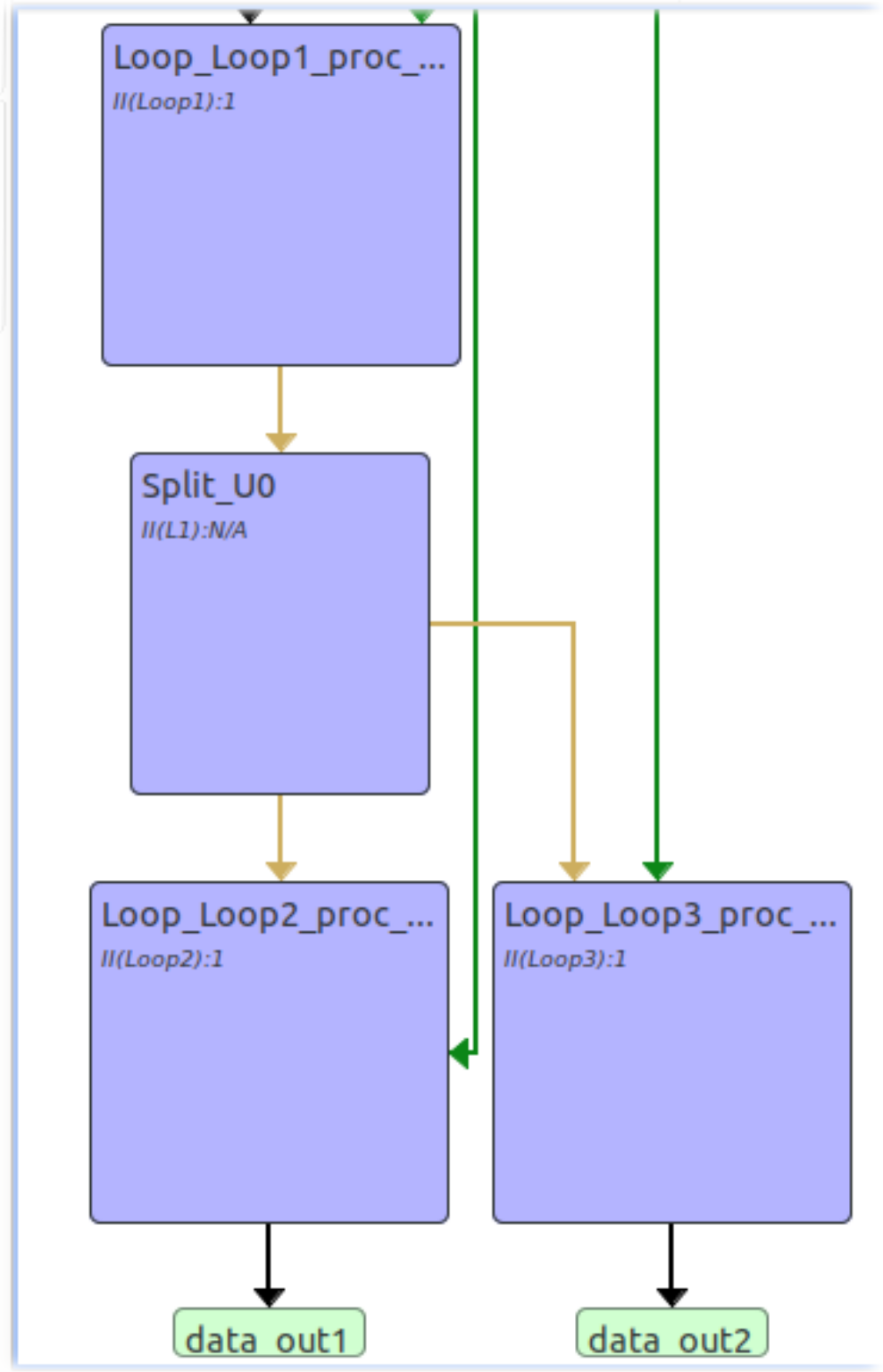


Рис. 20 Dataflow viewer ping-pong для функции foo_m часть 2

8. Исходный код модифицированного теста

Исходный код модифицированного теста для проверки функции foo_m используется такой же, как и при проверки функции foo_b. Тест обеспечивает проверку производительности функции на ПК. На рисунке 21 представлены результаты запуска функции на ПК. Как видно из рисунка среднее время выполнения функции после 32 итерации равно 225133.09 нс, что почти в 2 раза медленнее, чем решение полученное при синтезировании функции для FIFO memory buffer и почти в 2 раза быстрее для ping-pong memory buffer решения.

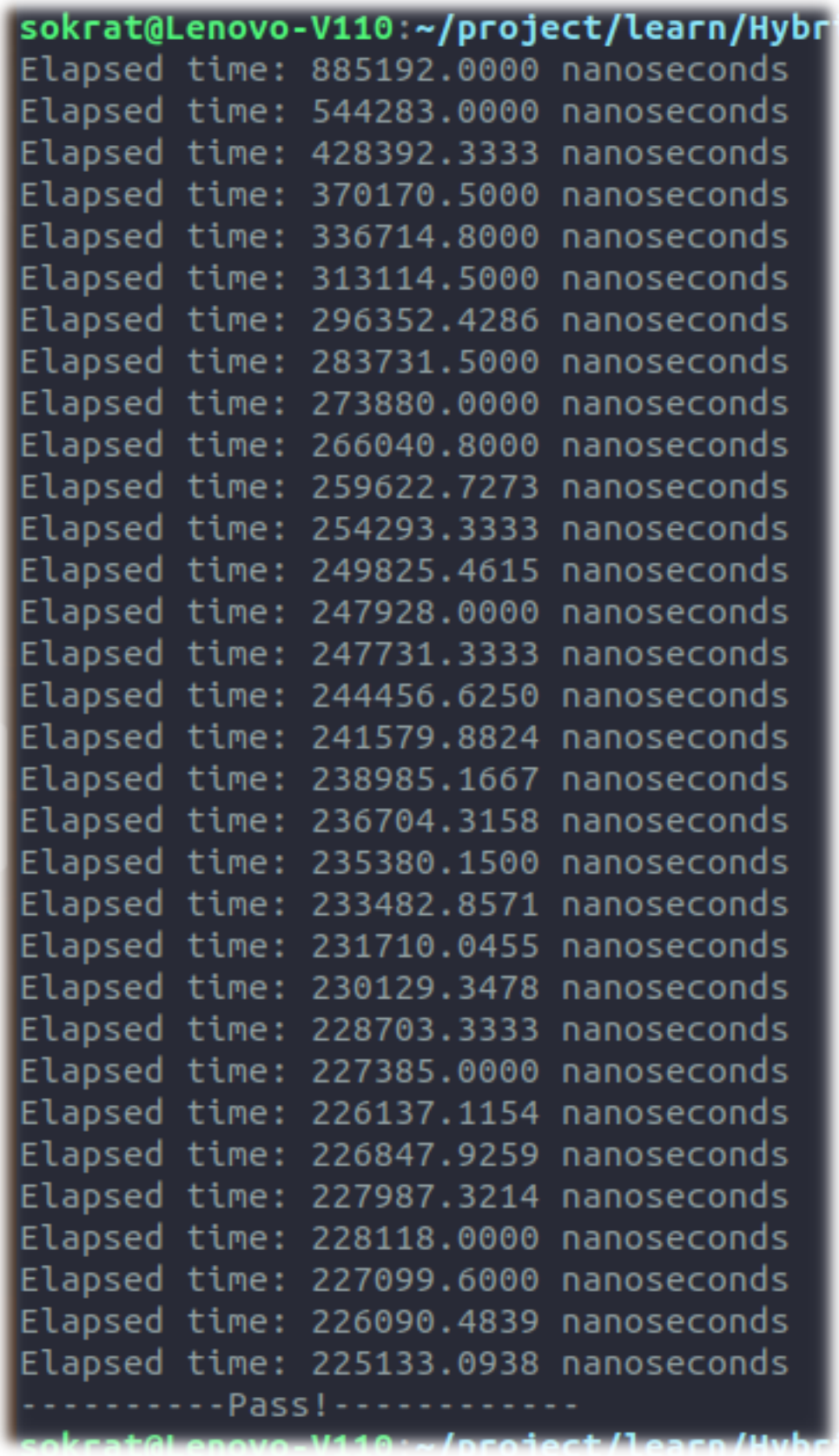


Рис. 21 Временные показатели для модифицированного теста для foo_m

Вывод

В данной работе была изучена возможность добавления pipeline dataflow директивы для синтезируемой функции. Был произведен сравнительный анализ между решением без добавлением и с добавлением pipeline dataflow. 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 для ping-pong memory buffer и медленнее, чем для FIFO memory buffer.