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

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ Lab4_Z4

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.

«8» ноября 2021

Санкт-Петербург
2021

1. Задание

Текст задания находится в файле «Задание lab4_z4.docx»

2. Исходный код фукнции

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

```
1 #include "lab4_z4.h"
2
3
4 void foo_b(int data_in[N], int scale[3], int data_out[N]) {
5
6     int temp1[N], temp2[N], temp3[N];
7     Loop1: for(int i = 0; i < N; i++) {
8         temp1[i] = data_in[i] * scale[0];
9         temp2[i] = data_in[i] * scale[1];
10    }
11    Loop2: for(int j = 0; j < N; j++) {
12        temp3[j] = temp1[j] * scale[2];
13    }
14    Loop3: for(int k = 0; k < N; k++) {
15        data_out[k] = temp2[k] + temp3[k];
16    }
17 }
18
19 void foo_m(int data_in[N], int scale[3], int data_out[N])
20 {
21    int temp1[N], temp2[N], temp3[N], temp4[N];
22    int tmp1 = scale[0], tmp2 = scale[1], tmp3 = scale[2];
23    Loop1: for(int i = 0; i < N; i++) {
24        temp1[i] = data_in[i] * tmp1;
25        temp2[i] = data_in[i] * tmp2;
26    }
27    Loop2: for(int j = 0; j < N; j++) {
28        temp3[j] = temp1[j] * tmp3;
29        temp4[j] = temp2[j];
30    }
31    Loop3: for(int k = 0; k < N; k++) {
32        data_out[k] = temp4[k] + temp3[k];
33    }
34 }
```

Рис. 1 Исходный код функции foo_b и foo_m

Функции принимает три аргумента массива типа int — вычисляет для

входного массива произведение на элементы второго массива и записывает результаты в выходной массив.

3. Исходный код теста

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

```
4  int cmp_arr(int data_in[N], int scale[3], int data_cmp[N]) {
5
6      int temp1[N], temp2[N], temp3[N];
7      for(int i = 0; i < N; i++) {
8          temp1[i] = data_in[i] * scale[0];
9          temp2[i] = data_in[i] * scale[1];
10     }
11     for(int j = 0; j < N; j++) {
12         temp3[j] = temp1[j] * scale[2];
13     }
14     for(int k = 0; k < N; k++) {
15         int tmp = temp2[k] + temp3[k];
16         if (tmp != data_cmp[k]) {
17             return 0;
18         }
19     }
20     return 1;
21 }
22
23 int main()
24 {
25     int pass=0;
26
27     // Call the function for 3 transactions
28     int scale[3];
29     int data_in[N];
30     int data_out[N];
31
32     for (int i = 0; i < 3; ++i){
33         for(int j = 0; j < N; j++){
34             data_in[j] = rand() % (N - 1);
35         }
36         for(int k = 0; k < 3; ++k){
37             scale[k] = rand() % ((N >> 1) - 1);
38         }
39
40         foo_b(data_in, scale, data_out);
41         pass = cmp_arr(data_in, scale, data_out);
42         if (pass == 0) {
43             fprintf(stderr, "----------Fail!-----------\n");
44             return 1;
```

Рис. 2 Исходный код lab4_z4_test.c тестирования функции

4. Командный файл

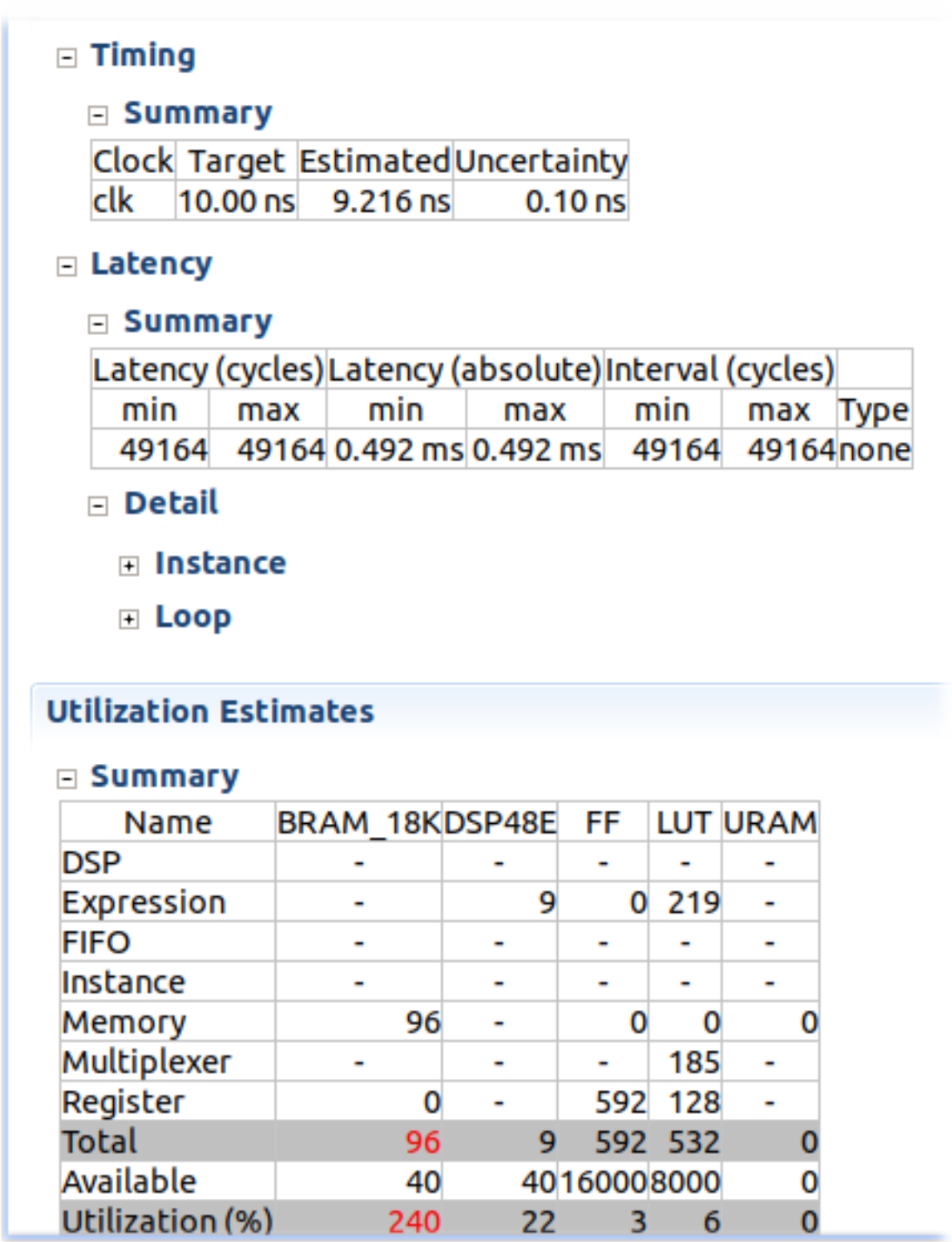
На рисунке 3 представлен текст команд для автоматизированного создания следующего варианта аппаратной реализаций: sol1 задается clock period 10. clock uncertainty 0.1

```
##################################################
#                      Lab                       #
##################################################
open_project -reset lab4_z4b_prj
set_top foo_b
add_files ./source/lab4_z4.c
add_files -tb ./source/lab4_z4_test.c

open_solution -reset sol1
create_clock -period 10 -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                    #
##################################################


exit
```

Рис. 3 Текст команд для создания решений

5. Результаты синтезирования функции

На рисунке 4 представлен perfomance и utilization estimates для данного решения. Как видно из результатов время Latency составляет 455258.64 нс.



Timing

Summary

Clock	Target	Estimated	Uncertainty
clk	10.00 ns	9.216 ns	0.10 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
49164	49164	0.492 ms	0.492 ms	49164	49164	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	9	0	219	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	96	-	0	0	0
Multiplexer	-	-	-	185	-
Register	0	-	592	128	-
Total	96	9	592	532	0
Available	40	40	16000	8000	0
Utilization (%)	240	22	3	6	0

Рис. 4 Результат синтезирования функции

Далее на рисунке 5 представлен schedule viewer с указанным на нем Latency и II(Initiation Interval).

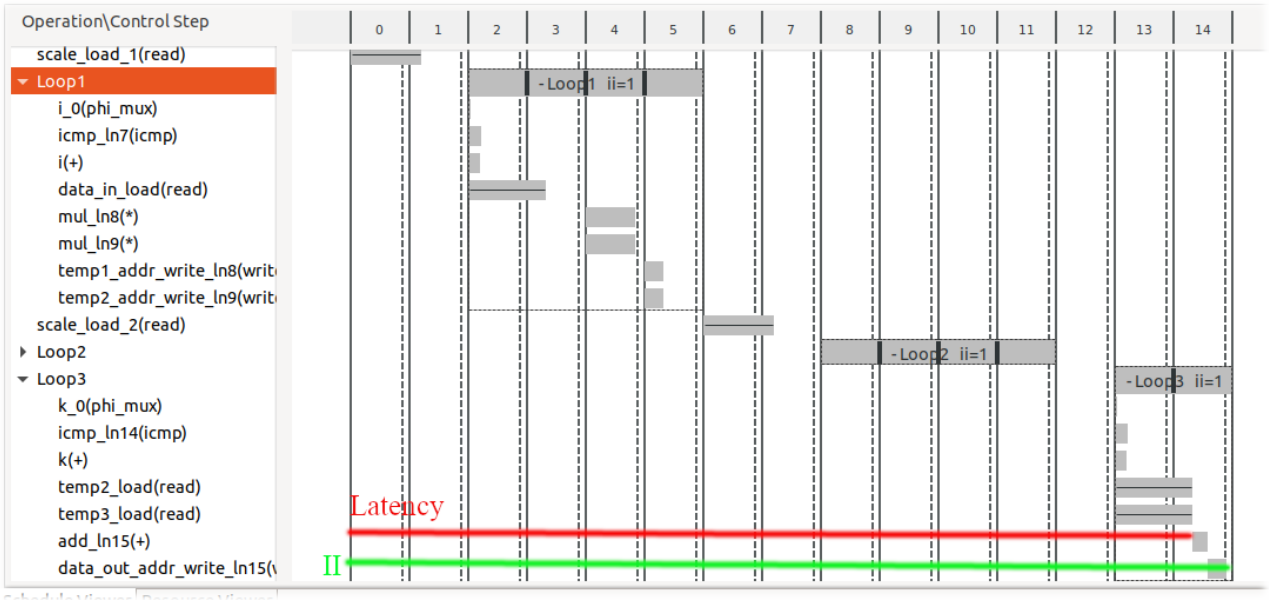
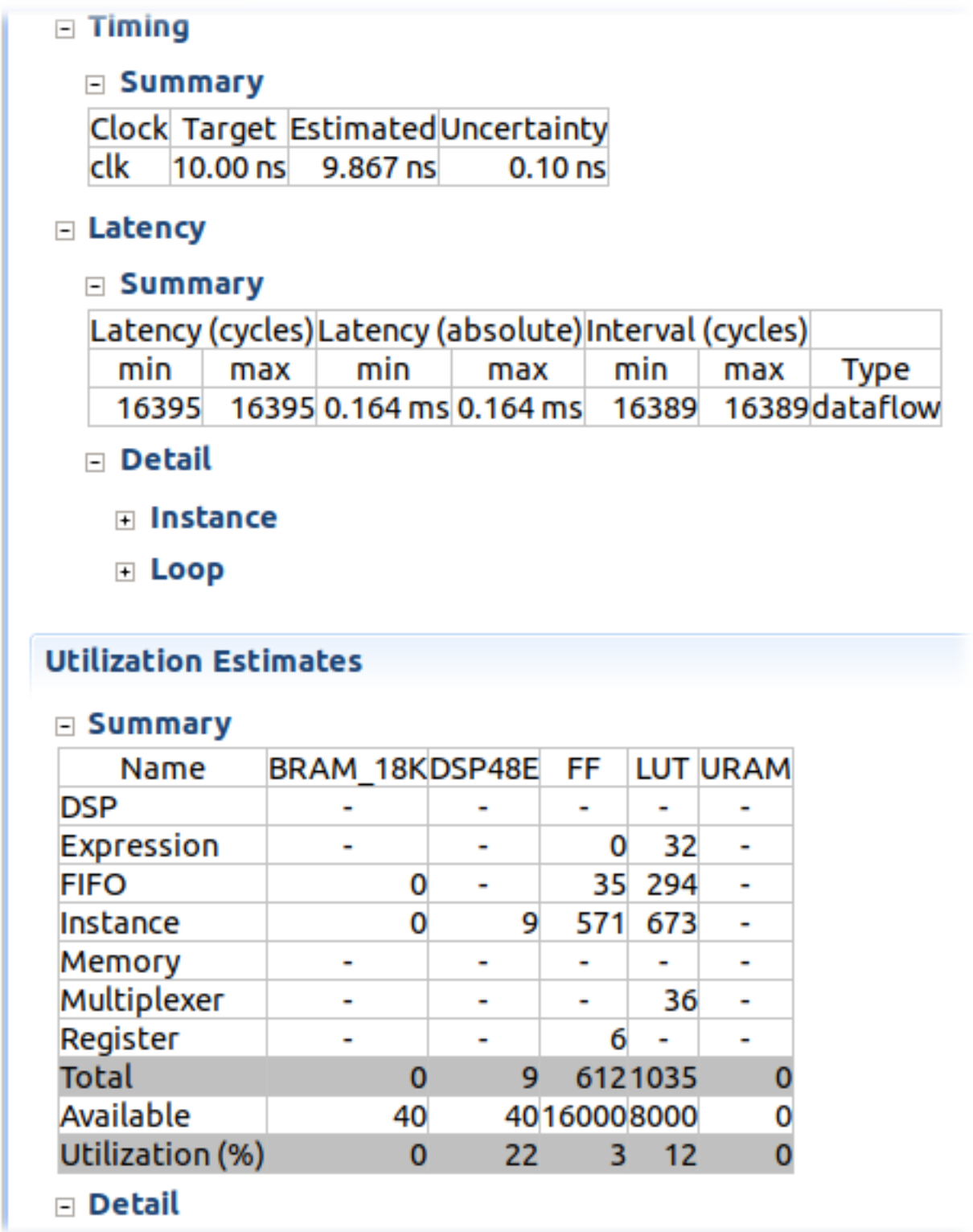


Рис. 5 Schedule viewer для функции foo_b

6. Добавление dataflow конвейеризации для решения

6.1 Dataflow с FIFO memory buffer

На рисунке 6 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	9.867 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	32	-
FIFO	0	-	35	294	-
Instance	0	9	571	673	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	0	9	612	1035	0
Available	40	40	16000	8000	0
Utilization (%)	0	22	3	12	0

Detail

Рис. 6 Результат синтезирования функции после pipeline dataflow

Далее на рисунке 7 и 8 представлены schedule и resource viewer соотвственно.

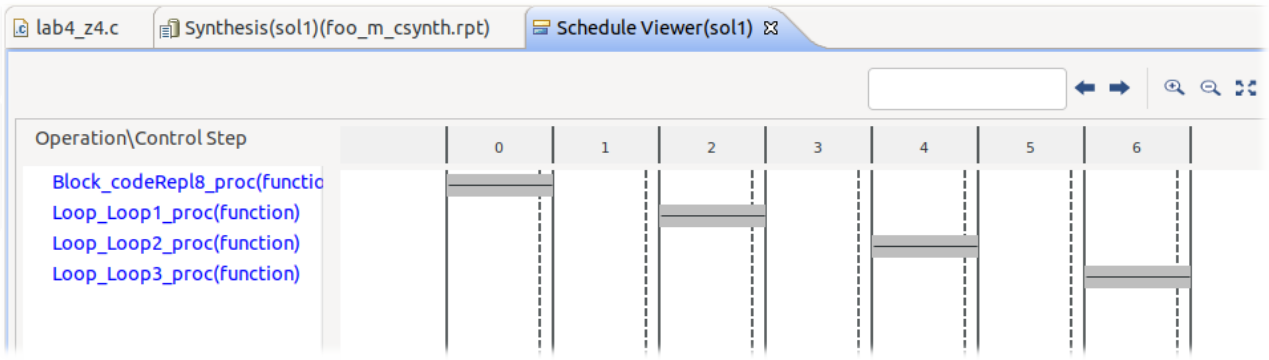


Рис. 7 Schedule viewer для функции foo_m

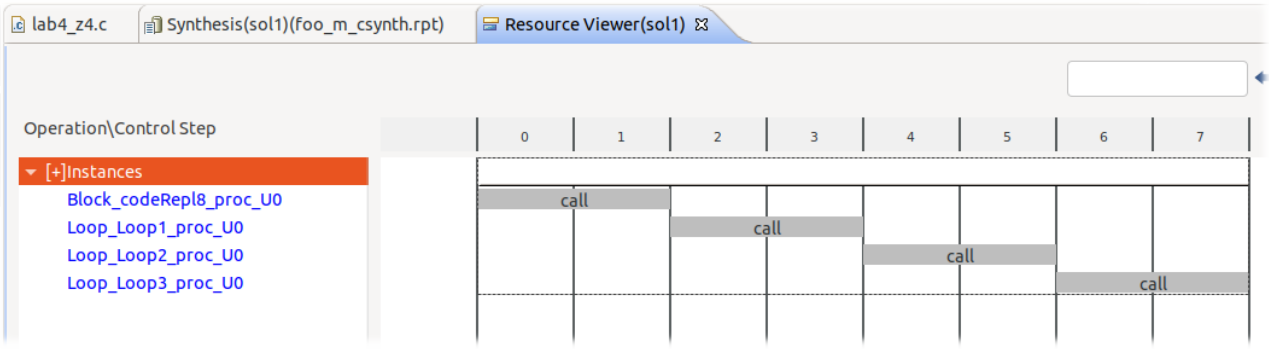


Рис. 8 Resource viewer для функции foo_m

На рисунке 9 и 10 представлен dataflow viewer для решения FIFO memory buffer.

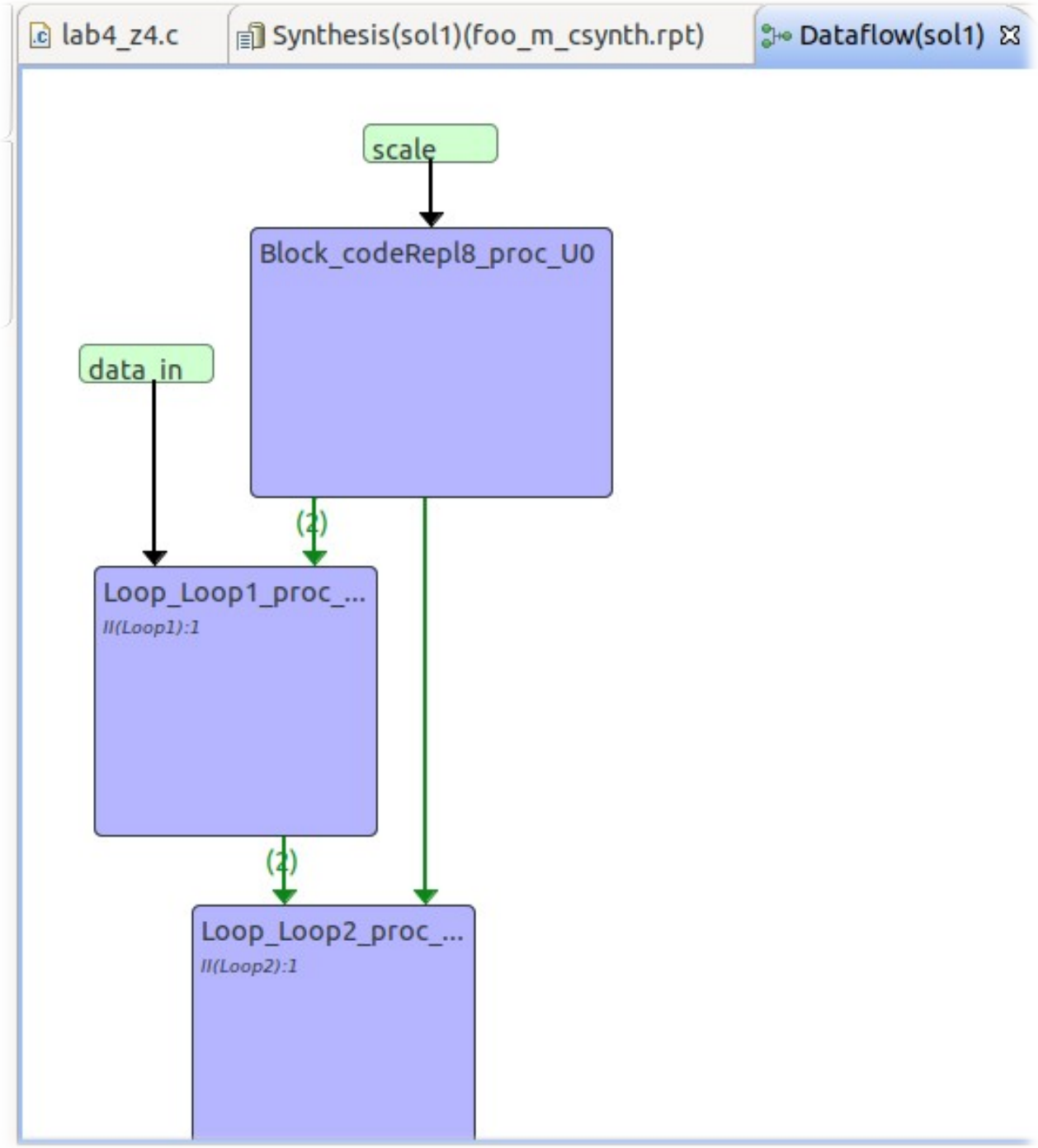


Рис. 9 Dataflow viewer FIFO для функции foo_m часть 1

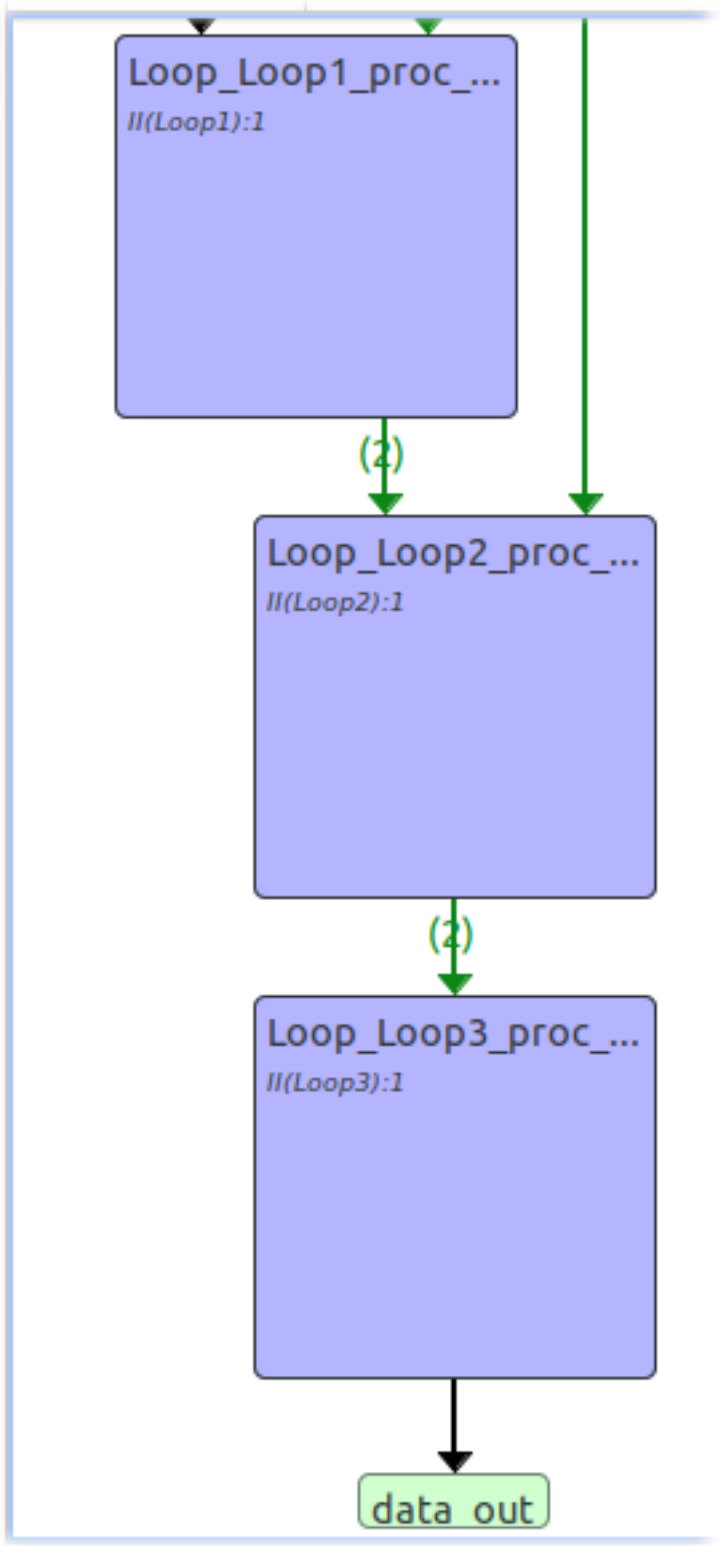


Рис. 10 Dataflow viewer FIFO для функции foo_m часть 2

На рисунке 11 представлена временная диаграмма для этого решения с несколькими тактами работы функции после выполнения C/RTL cosimulation.

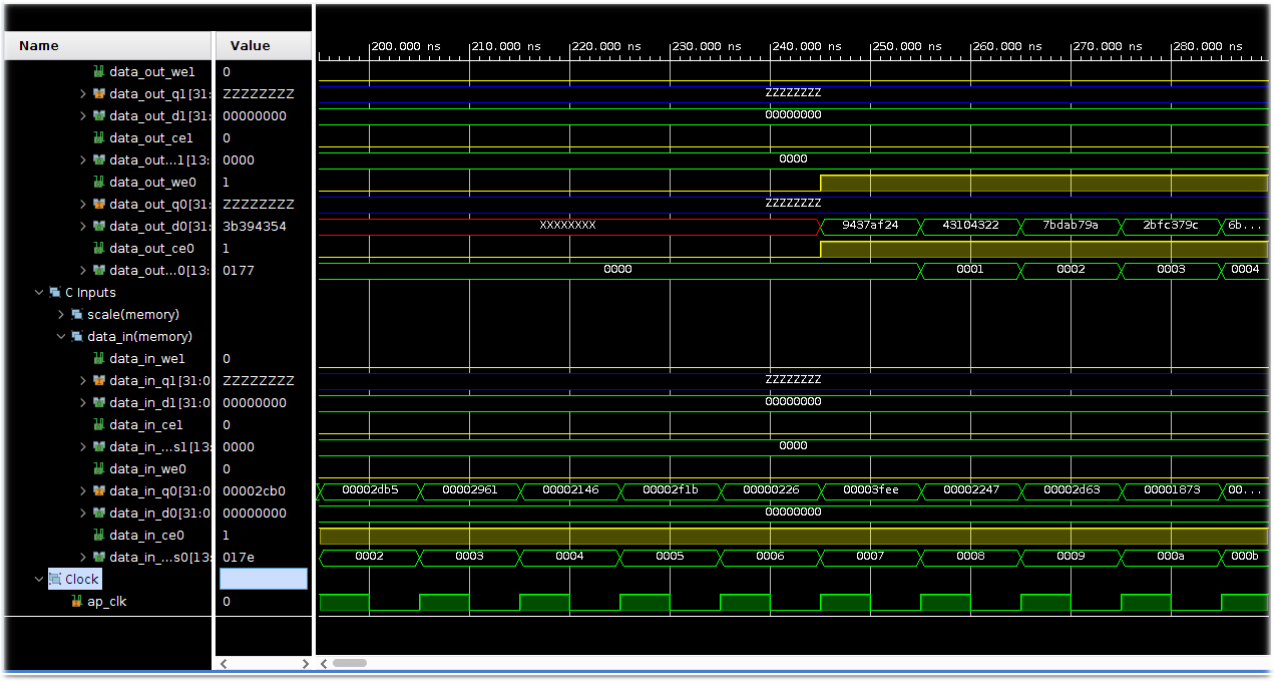


Рис. 11 Временная диаграмма для функции foo_m

6.2 Dataflow с ping-pong memory buffer

На рисунке 12 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	9.216 ns	0.10 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
49167	49167	0.492 ms	0.492 ms	16389	16389	dataflow

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	66	-
FIFO	0	-	15	126	-
Instance	0	9	599	586	-
Memory	128	-	0	0	0
Multiplexer	-	-	-	90	-
Register	-	-	12	-	-
Total	128	9	626	868	0
Available	40	40	16000	8000	0
Utilization (%)	320	22	3	10	0

Рис. 12 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 13 и 14 представлены schedule и resource viewer соотвственно.

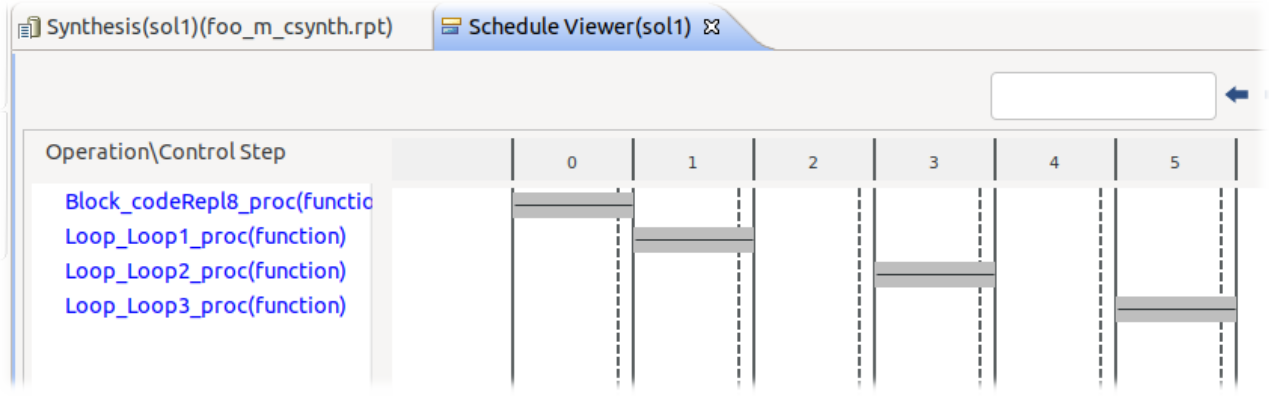


Рис. 13 Schedule viewer для функции foo_m

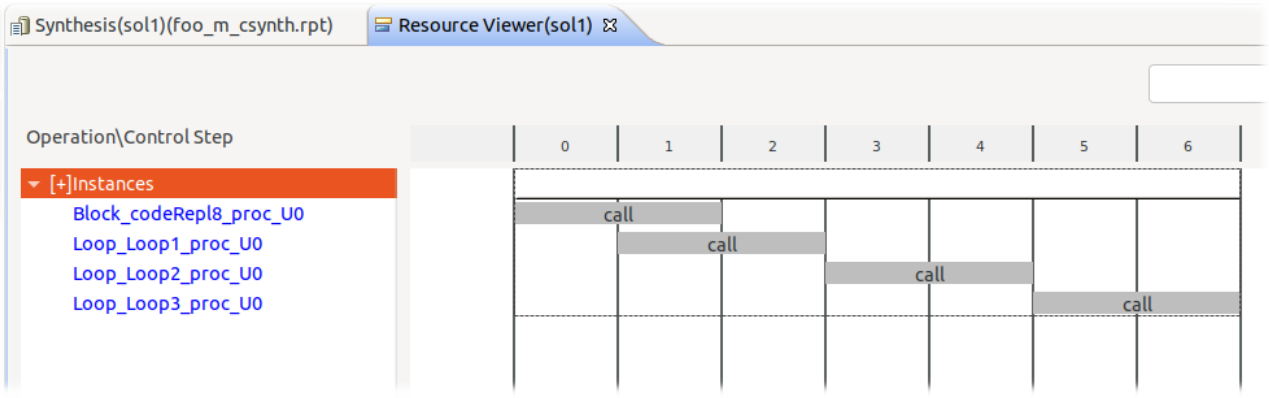


Рис. 14 Resource viewer для функции foo_m

На рисунке 15 и 16 представлен dataflow viewer для решения ping-pong memory buffer.

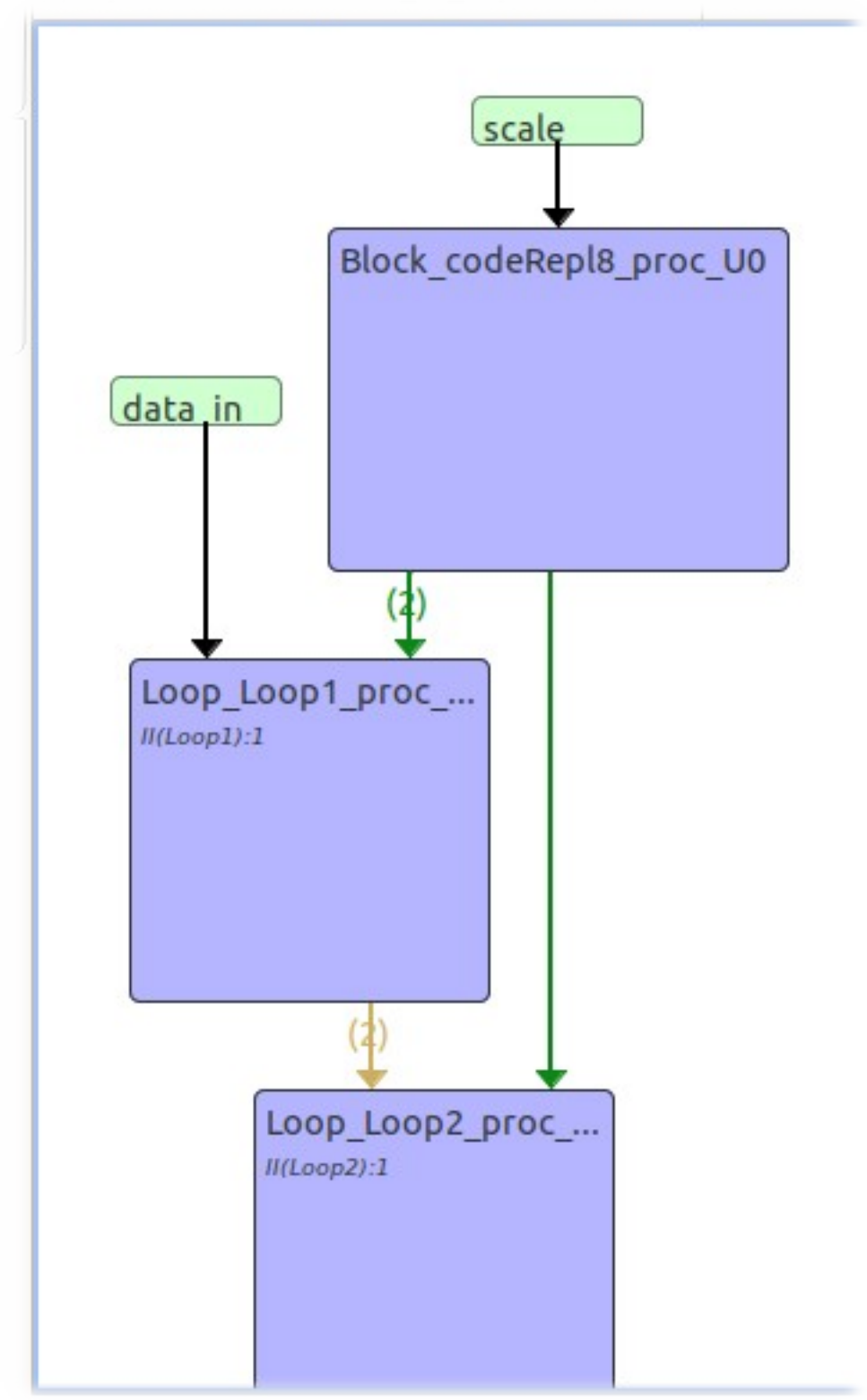


Рис. 15 Dataflow viewer ping-pong для функции foo_m часть 1

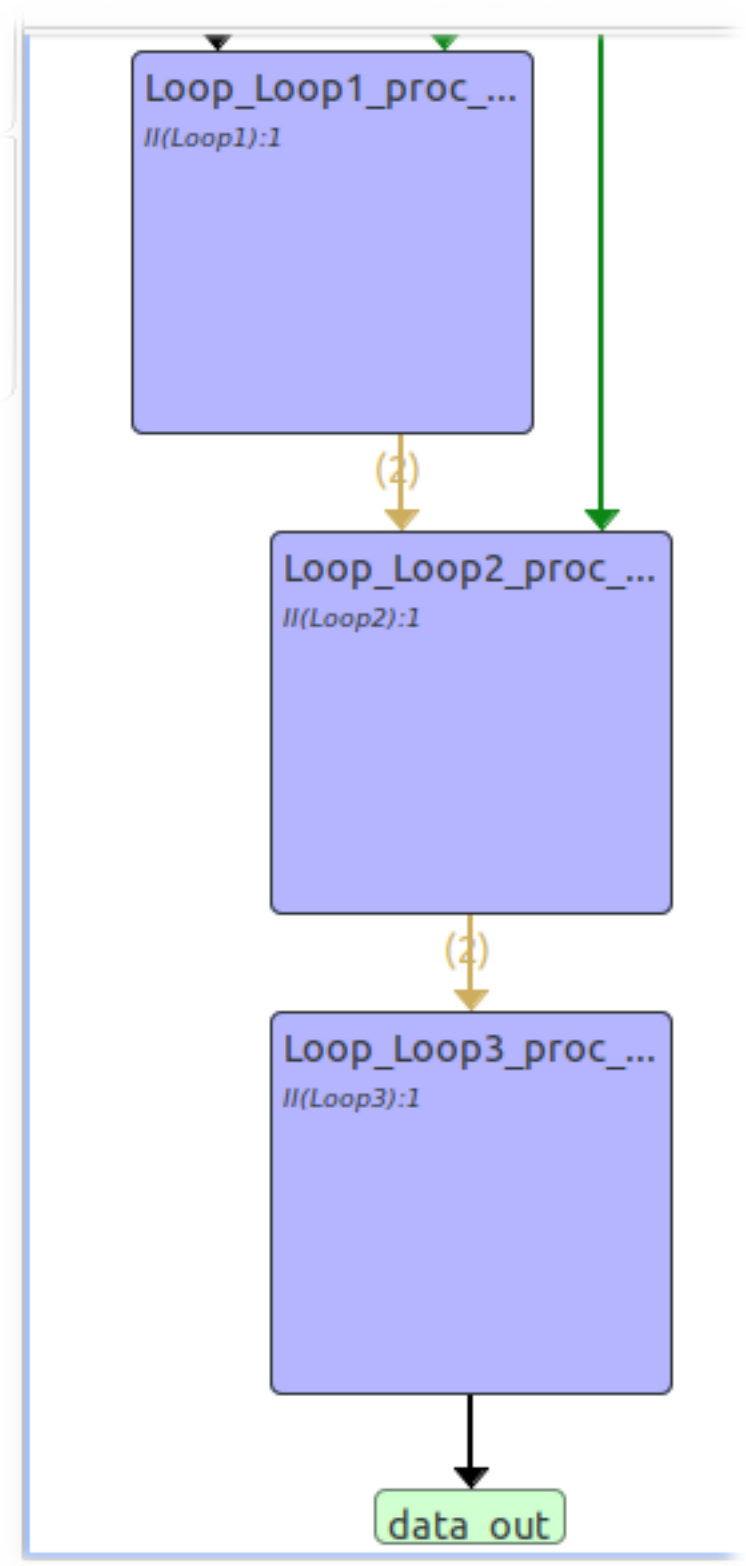


Рис. 16 Dataflow viewer ping-pong для функции foo_m часть 2

7. Исходный код модифицированного теста

Исходный код модифицированного теста для проверки функции foo_m представлен на рисунке 17. Тест обеспечивает проверку производительности функции на ПК. На рисунке 18 представлены результаты запуска функции на ПК. Как видно из рисунка среднее время выполнения функции после 32 итерации равно 191294.00 нс, что почти такой же результат показало решение полученное при синтезировании функции для FIFO memory buffer и почти в 2 раза быстрее для ping-pong memory buffer решения.

Таблица 1

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

```
24 int main()
25 {
26     int pass=0;
27
28     // Call the function for 32 transactions
29     int scale[3];
30     int data_in[N];
31     int data_out[N];
32     struct timespec t0, t1;
33     double acc_time = 0.0;
34
35     for (int i = 0; i < 32; ++i){
36         for(int j = 0; j < N; j++){
37             data_in[j] = rand() % (N - 1);
38         }
39         for(int k = 0; k < 3; ++k){
40             scale[k] = rand() % ((N >> 1) - 1);
41         }
42
43
44         if(clock_gettime(CLOCK_REALTIME, &t0) != 0) {
45             perror("Error in calling clock_gettime\n");
46             exit(EXIT_FAILURE);
47         }
48         foo_m(data_in, scale, data_out);
49         if(clock_gettime(CLOCK_REALTIME, &t1) != 0) {
50             perror("Error in calling clock_gettime\n");
51             exit(EXIT_FAILURE);
52         }
53         double diff_time = (((double)(t1.tv_sec - t0.tv_sec))*1000000000.0) + (double)(t1.tv_nsec - t0.tv_nsec);
54         acc_time += diff_time;
55         double temp_avg_time = acc_time / (i + 1); // take average time
56         printf("Elapsed time: %.4lf nanoseconds\n", temp_avg_time);
57
58         pass = cmp_arr(data_in, scale, data_out);
59         if (pass == 0) {
60             fprintf(stderr, "----------Fail!-----------\n");
61             return 1;
62         }
63     }
64
65
```

Рис. 17 Исходный код тестирования функции для исполнения на ПК

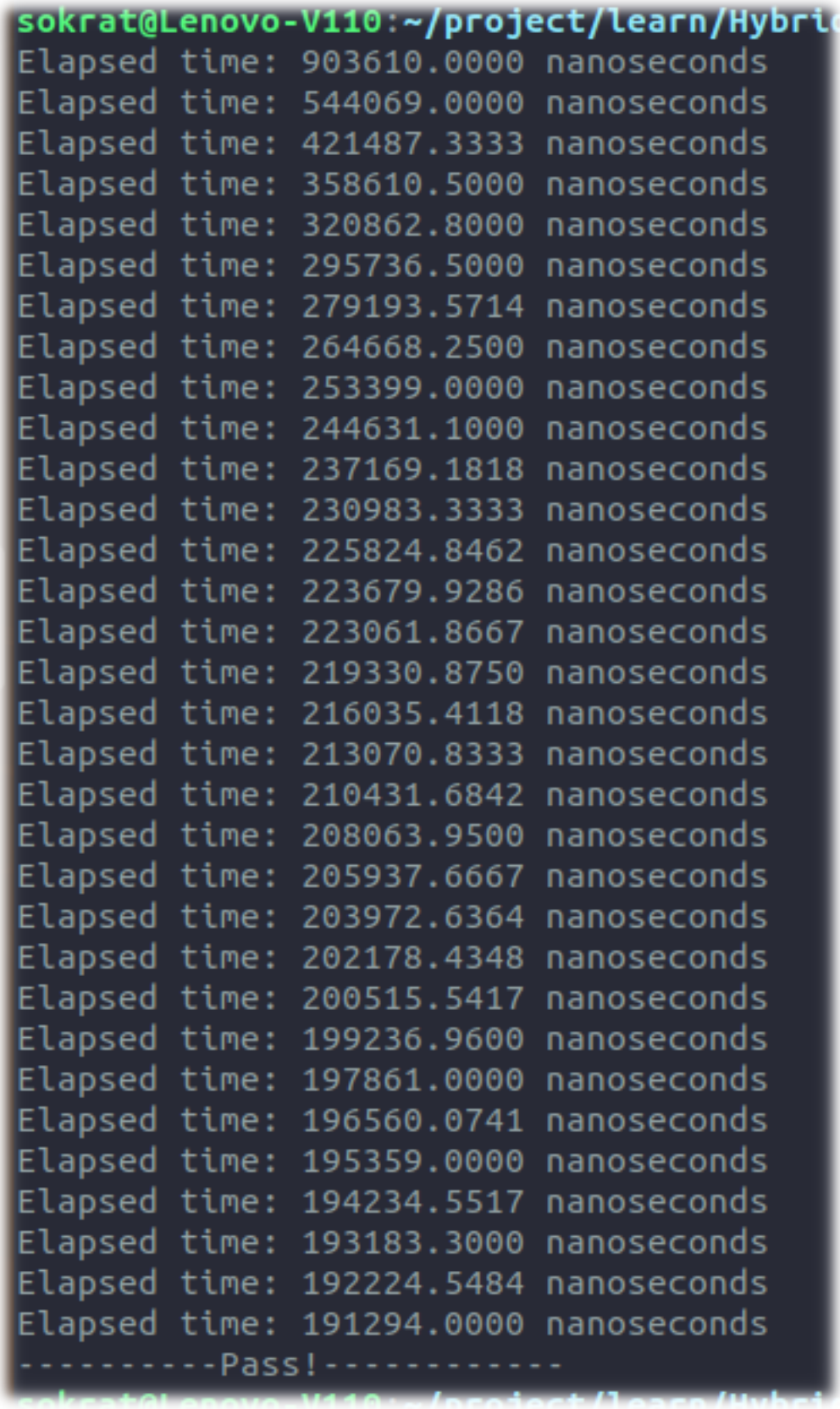


Рис. 18 Временные показатели для модифицированного теста для foo_m

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 такое же, как у решения с FIFO memory buffer.