Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Лабораторная №8

Предмет: Проектирование реконфигурируемых гибридных вычислительных систем

Тема: Анализ потока данных

Задание 1

Студенты:
Соболь В.
Темнова А.С.
Группа: 13541/3
Преподаватель:
Антонов А.П.

Санкт-Петербург
2019

Содержание

1. Задание

1. Создать проект lab8_1
2. Микросхема: xa7a12tcsg325-1q
3. Создать две функции (см. Текст ниже) – исходную и модифицированную - и провести их анализ.

Single-producer-consumer Violations

For Vivado HLS to perform the DATAFLOW optimization, all elements passed between tasks

must follow a single-producer-consumer model. Each variable must be driven from a single task

and only be consumed by a single task. In the following code example, temp1fans out and is

consumed by both Loop2and Loop3. This violates the single-producer-consumer model.

```
void foo_b(int data_in[N], int scale, int data_out1[N], int data_out2[N]) {
int temp1[N];
Loop1: for(int i = 0; i < N; i++) {
temp1[i] = data_in[i] * scale;
}
Loop2: for(int j = 0; j < N; j++) {
data_out1[j] = temp1[j] * 123;
}
Loop3: for(int k = 0; k < N; k++) {
data_out2[k] = temp1[k] * 456;
}
}
```

A modified version of this code uses function Split to create a single-producer-consumer

design. In this case, data flows from Loop1 to Split and then to Loop2 and Loop3.

The data now flows between all four tasks, and Vivado HLS can perform the DATAFLOW

Optimization

```
void Split (in[N], out1[N], out2[N]) {
// Duplicated data
L1:for(int i=1;i<N;i++) {
out1[i] = in[i];
out2[i] = in[i];
}
```

}
void foo_m(int data_in[N], int scale, int data_out1[N], int data_out2[N]) {
int temp1[N], temp2[N]. temp3[N];
Loop1: for(int i = 0; i < N; i++) {
temp1[i] = data_in[i] * scale;
}
Split(temp1, temp2, temp3);
Loop2: for(int j = 0; j < N; j++) {
data_out1[j] = temp2[j] * 123;
}
Loop3: for(int k = 0; k < N; k++) {
data_out2[k] = temp3[k] * 456;
}
}

4. Создать тест lab8_1_test.c для проверки функций выше.
5. Для функции **foo_b**
   - задать: clock period 10; clock_uncertainty 0.1
   - осуществить моделирование (с выводом результатов в консоль)
   - осуществить синтез для:
     - привести в отчете:
       * performance estimates=>summary
       * utilization estimates=>summary
       * scheduler viewer (выполнить Zoom to Fit)
         · На скриншоте показать Latency
         · На скриншоте показать Initiation Interval
       * resource viewer (выполнить Zoom to Fit)
         · На скриншоте показать Latency
         · На скриншоте показать Initiation Interval
6. Для функции **foo_m**
   - задать: clock period 10; clock_uncertainty 0.1
   - осуществить моделирование (с выводом результатов в консоль)
   - осуществить синтез для случая **FIFO for the memory buffers**:
     - привести в отчете:
       * performance estimates=>summary
       * utilization estimates=>summary
       * scheduler viewer (выполнить Zoom to Fit)
         · На скриншоте показать Latency

            - На скриншоте показать Initiation Interval
        * resource viewer (выполнить Zoom to Fit)
            - На скриншоте показать Latency
            - На скриншоте показать Initiation Interval
        * Dataflow viewer
- осуществить синтез для случая **ping-pong buffers**:
    - привести в отчете:
        * performance estimates=>summary
        * utilization estimates=>summary
        * scheduler viewer (выполнить Zoom to Fit)
            - На скриншоте показать Latency
            - На скриншоте показать Initiation Interval
        * resource viewer (выполнить Zoom to Fit)
            - На скриншоте показать Latency
            - На скриншоте показать Initiation Interval
        * Dataflow viewer
- Осуществить C|RTL моделирование для случая **FIFO for the memory buffers**
    - Привести результаты из консоли
    - Открыть временную диаграмму (все сигналы)
        * Отобразить два цикла обработки на одном экране
            - На скриншоте показать Latency
            - На скриншоте показать Initiation Interval

7. Выводы

- Объяснить отличия в синтезе foo_b и двух вариантов foo_m между собой

2. Скрипт

Ниже приводится скрипт, для автоматизации выполнения лабораторной работы.

```
open_project -reset lab8_1_b
add_files lab8_1_b.c
set_top foo
add_files -tb lab8_1_test.c

open_solution solution1 -reset
set_part {xa7a12tcsg325-1q}
create_clock -period 10ns
set_clock_uncertainty 0.1

csim_design
csynth_design

open_project -reset lab8_1_m
add_files lab8_1_m.c
set_top foo
add_files -tb lab8_1_test.c


open_solution solution_ping_pong -reset
set_part {xa7a12tcsg325-1q}
create_clock -period 10ns
set_clock_uncertainty 0.1
config_dataflow  -default_channel pingpong
set_directive_dataflow foo

csim_design
csynth_design




open_solution solution_fifo -reset
set_part {xa7a12tcsg325-1q}
create_clock -period 10ns
set_clock_uncertainty 0.1
config_dataflow  -default_channel fifo
set_directive_dataflow foo

csim_design
csynth_design
cosim_design -trace_level all


exit
```

Рис. 2.1. Скрипт

3. Решение 1

3.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
1  #include "lab8_1.h"
2
3  void foo(int data_in[N], int scale, int data_out1[N], int data_out2[N]) {
4    int temp1[N];
5    Loop1: for(int i = 0; i < N; i++) {
6      temp1[i] = data_in[i] * scale;
7    }
8    Loop2: for(int j = 0; j < N; j++) {
9      data_out1[j] = temp1[j] * 123;
10   }
11   Loop3: for(int k = 0; k < N; k++) {
12     data_out2[k] = temp1[k] * 456;
13   }
14 }
```

Рис. 3.1. Исходный код устройства

```
1  #define N 20
```

Рис. 3.2. Заголовочный файл

```c
#include <stdio.h>
#include "lab8_1.h"


void generate_test_data(int scale, int data_in[N], int data_out1[N], int
    ↪ data_out2[N]) {
  int temp1[N];
  for(int i = 0; i < N; i++) {
    data_in[i] = i;
    temp1[i] = i * scale;
  }
  for(int j = 0; j < N; j++) {
    data_out1[j] = temp1[j] * 123;
  }
  for(int k = 0; k < N; k++) {
    data_out2[k] = temp1[k] * 456;
  }
}

int compare_array_eq(int actual[N], int expected[N]){
  for (int i = 0; i < N; ++i) {
    if (actual[i] != expected[i]) {
      fprintf(stdout, "%d: Expeced %d Actual %d\n", i, expected[i], actual[i]);
    ↪
      return 0;
    }
  }
  return 1;
}

int main() {
  int pass = 1;
  int scale;
  int data_in[N];
  int data_out1[N], data_out2[N];
  int expected_out1[N], expected_out2[N];


  for (int i = 1; i < 4; ++i) {
    scale = i;
    generate_test_data(scale, data_in, expected_out1, expected_out2);


    foo(data_in, scale, data_out1, data_out2);

    if(!compare_array_eq(data_out1, expected_out1) || !compare_array_eq(
    ↪ data_out2, expected_out2)){
      pass = 0;
    }

  }

  if (pass) {
    fprintf(stdout, "----------Pass!----------\n");
    return 0;
  } else  {
    fprintf(stderr, "----------Fail!----------\n");
    return 1;
  }
}
```

Рис. 3.3. Исходный код теста

3.2. Моделирование

Ниже приведены результаты моделирования.

```
INFO: [SIM 211-2] *************** CSIM start ***************
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
   Compiling(apcc) ../../../../lab8_1_test.c in debug mode
INFO: [HLS 200-10] Running '/opt/Xilinx/Vivado/2018.2/bin/unwrapped/lnx64.o/apcc'
INFO: [HLS 200-10] For user 'sobol' on host 'gadolinium.local' (Linux_x86_64 version 5.3.12
18:48 MSK 2019
INFO: [HLS 200-10] On os "Arch Linux"
INFO: [HLS 200-10] In directory '/home/sobol/Downloads/labs_from_8/lab8_z1/source/lab8_1_b/
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_sobol/72675175634729007530
INFO: [APCC 202-1] APCC is done.
   Compiling(apcc) ../../../../lab8_1_b.c in debug mode
INFO: [HLS 200-10] Running '/opt/Xilinx/Vivado/2018.2/bin/unwrapped/lnx64.o/apcc'
INFO: [HLS 200-10] For user 'sobol' on host 'gadolinium.local' (Linux_x86_64 version 5.3.12
18:54 MSK 2019
INFO: [HLS 200-10] On os "Arch Linux"
INFO: [HLS 200-10] In directory '/home/sobol/Downloads/labs_from_8/lab8_z1/source/lab8_1_b/
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_sobol/727321575634734167362
INFO: [APCC 202-1] APCC is done.
   Generating csim.exe
----------Pass!------------
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
```

Рис. 3.4. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

3.3. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.470	0.10

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
243	243	243	243	none

Рис. 3.5. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	9	0	141
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	64	10
Multiplexer	-	-	-	107
Register	-	-	215	-
Total	0	9	279	258
Available	40	40	16000	8000
Utilization (%)	0	22	1	3

Рис. 3.6. Utilization estimates

Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
foo	-	243	-	244	-
Loop1	no	80	4	-	20
Loop2	no	80	4	-	20
Loop3	no	80	4	-	20

Рис. 3.7. Performance profile

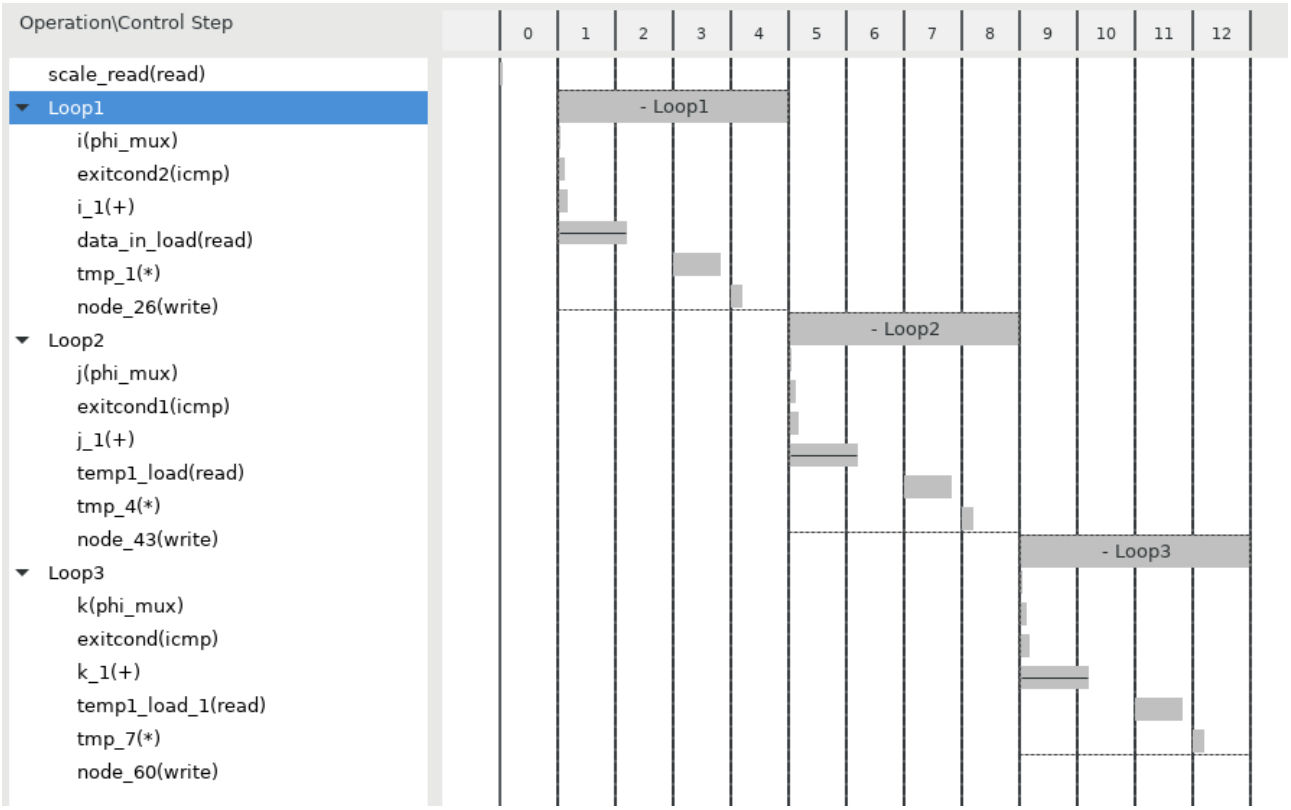

Рис. 3.8. Scheduler viewer

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
1	I/O Ports													
2	scale	read												
3	data_in(p0)		read											
4	data_out1(p0)									write				
5	data_out2(p0)													write
6	Memory Ports													
7	data_in(p0)		read											
8	temp1(p0)					write	read				read			
9	data_out1(p0)									write				
10	data_out2(p0)													write
11	Expressions													
12	i_1_fu_160		+											
13	i_phi_fu_121		phi_mux											
14	exitcond2_fu_154		icmp											
15	tmp_1_fu_171				*									
16	j_1_fu_181						+							
17	j_phi_fu_132						phi_mux							
18	exitcond1_fu_175						icmp							
19	tmp_4_fu_192								*					
20	k_1_fu_204										+			
21	k_phi_fu_143										phi_mux			
22	exitcond_fu_198										icmp			
23	tmp_7_fu_215												*	

Рис. 3.9. Resource viewer

4. Решение 2

4.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
#include "lab8_1.h"

void Split (int in[N], int out1[N], int out2[N]) {
  // Duplicated data
  L1:for(int i=0; i<N; i++) {
    out1[i] = in[i];
    out2[i] = in[i];
  }
}

void foo(int data_in[N], int scale, int data_out1[N], int data_out2[N]) {
  int temp1[N], temp2[N], temp3[N];
  Loop1: for(int i = 0; i < N; i++) {
    temp1[i] = data_in[i] * scale;
  }
  Split(temp1, temp2, temp3);
  Loop2: for(int j = 0; j < N; j++) {
    data_out1[j] = temp2[j] * 123;
  }
  Loop3: for(int k = 0; k < N; k++) {
    data_out2[k] = temp3[k] * 456;
  }
}
```

Рис. 4.1. Исходный код устройства

```
#define N 20
```

Рис. 4.2. Заголовочный файл

```c
#include <stdio.h>
#include "lab8_1.h"


void generate_test_data(int scale, int data_in[N], int data_out1[N], int
    ↪ data_out2[N]) {
  int temp1[N];
  for(int i = 0; i < N; i++) {
    data_in[i] = i;
    temp1[i] = i * scale;
  }
  for(int j = 0; j < N; j++) {
    data_out1[j] = temp1[j] * 123;
  }
  for(int k = 0; k < N; k++) {
    data_out2[k] = temp1[k] * 456;
  }
}

int compare_array_eq(int actual[N], int expected[N]){
  for (int i = 0; i < N; ++i) {
    if (actual[i] != expected[i]) {
      fprintf(stdout, "%d: Expeced %d Actual %d\n", i, expected[i], actual[i]);
    ↪
      return 0;
    }
  }
  return 1;
}

int main() {
  int pass = 1;
  int scale;
  int data_in[N];
  int data_out1[N], data_out2[N];
  int expected_out1[N], expected_out2[N];


  for (int i = 1; i < 4; ++i) {
    scale = i;
    generate_test_data(scale, data_in, expected_out1, expected_out2);


    foo(data_in, scale, data_out1, data_out2);

    if(!compare_array_eq(data_out1, expected_out1) || !compare_array_eq(
    ↪ data_out2, expected_out2)){
      pass = 0;
    }

  }

  if (pass) {
    fprintf(stdout, "----------Pass!----------\n");
    return 0;
  } else  {
    fprintf(stderr, "----------Fail!----------\n");
    return 1;
  }
}
```

Рис. 4.3. Исходный код теста

4.2. Директивы

В данном решения были установлены директивы, приведённые ниже.

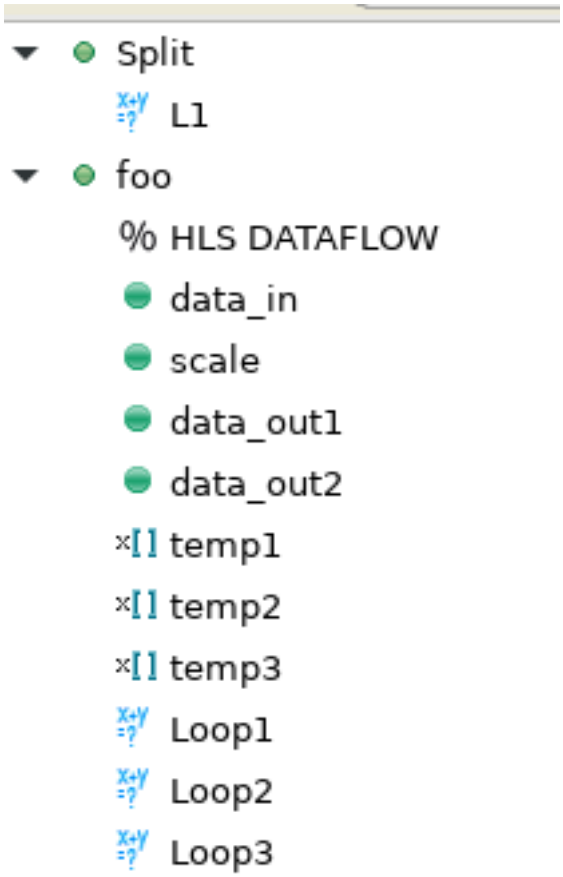

Рис. 4.4. Директивы

4.3. Моделирование

Ниже приведены результаты моделирования.

```
INFO: [APCC 202-1] APCC is done.
   Generating csim.exe
----------Pass!------------
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
```

Рис. 4.5. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

4.4. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.470	0.10

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
85	85	82	82	dataflow

Рис. 4.6. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	24
FIFO	6	-	147	120
Instance	-	9	245	469
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	6	9	392	613
Available	40	40	16000	8000
Utilization (%)	15	22	2	7

Рис. 4.7. Utilization estimates

Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
foo	-	85	-	82	-

Рис. 4.8. Performance profile

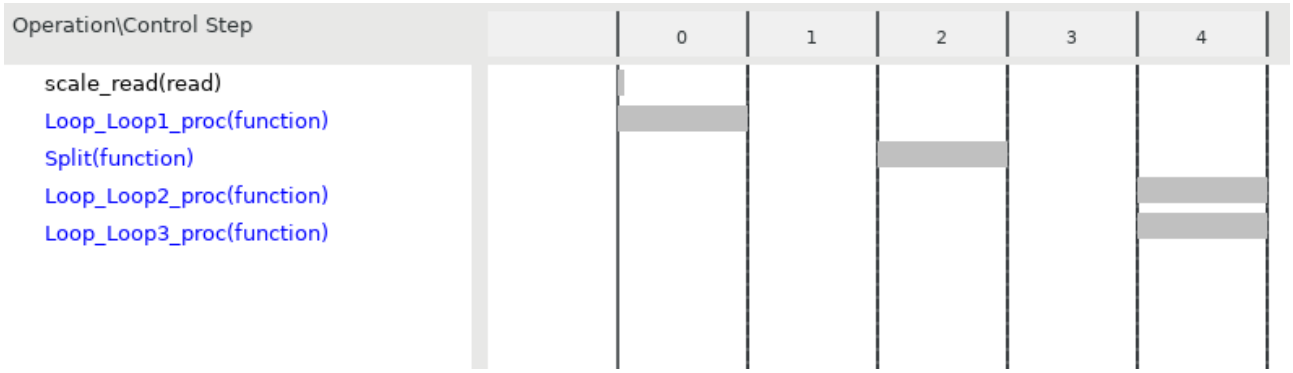

Рис. 4.9. Scheduler viewer

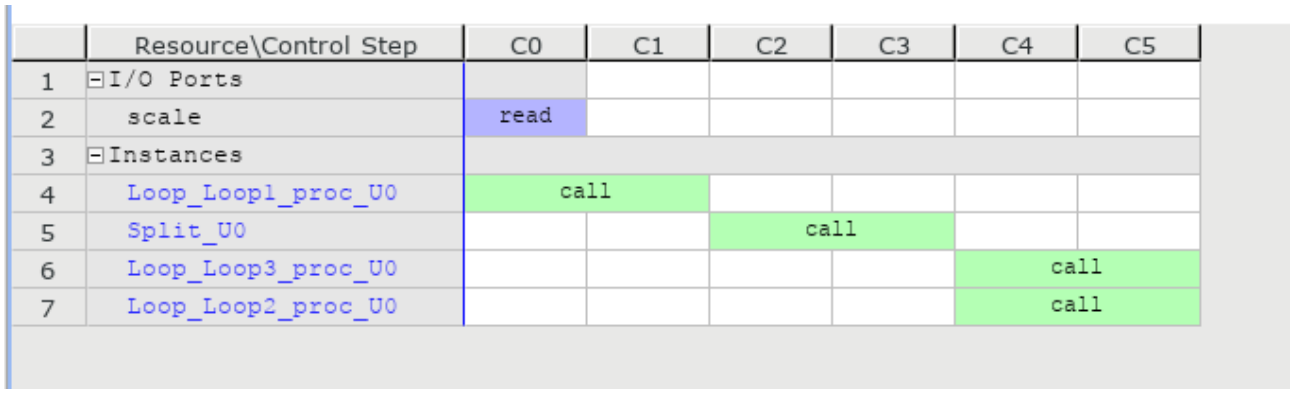

Рис. 4.10. Resource viewer

Рис. 4.11. Dataflow viewer

4.5. C/RTL моделирование

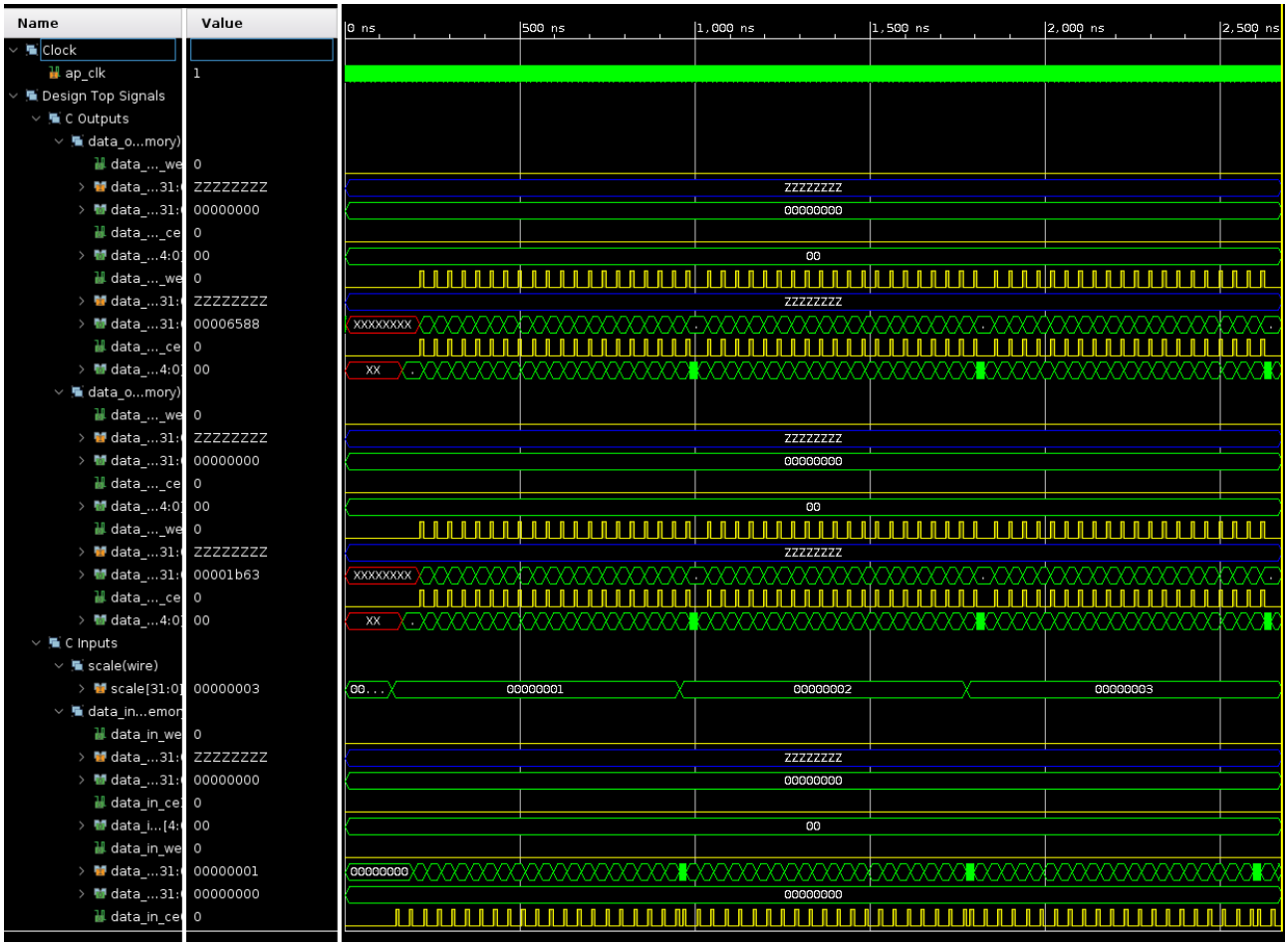

Рис. 4.12. Временная диаграмма

5. Решение 3

5.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
1 #include "lab8_1.h"
2
3 void Split (int in[N], int out1[N], int out2[N]) {
4   // Duplicated data
5   L1:for(int i=0; i<N; i++) {
6     out1[i] = in[i];
7     out2[i] = in[i];
8   }
9 }
10
11 void foo(int data_in[N], int scale, int data_out1[N], int data_out2[N]) {
12   int temp1[N], temp2[N], temp3[N];
13   Loop1: for(int i = 0; i < N; i++) {
14     temp1[i] = data_in[i] * scale;
15   }
16   Split(temp1, temp2, temp3);
17   Loop2: for(int j = 0; j < N; j++) {
18     data_out1[j] = temp2[j] * 123;
19   }
20   Loop3: for(int k = 0; k < N; k++) {
21     data_out2[k] = temp3[k] * 456;
22   }
23 }
```

Рис. 5.1. Исходный код устройства

```
1 #define N 20
```

Рис. 5.2. Заголовочный файл

```c
#include <stdio.h>
#include "lab8_1.h"


void generate_test_data(int scale, int data_in[N], int data_out1[N], int
    ↪ data_out2[N]) {
  int temp1[N];
  for(int i = 0; i < N; i++) {
    data_in[i] = i;
    temp1[i] = i * scale;
  }
  for(int j = 0; j < N; j++) {
    data_out1[j] = temp1[j] * 123;
  }
  for(int k = 0; k < N; k++) {
    data_out2[k] = temp1[k] * 456;
  }
}

int compare_array_eq(int actual[N], int expected[N]){
  for (int i = 0; i < N; ++i) {
    if (actual[i] != expected[i]) {
      fprintf(stdout, "%d: Expeced %d Actual %d\n", i, expected[i], actual[i]);
    ↪
      return 0;
    }
  }
  return 1;
}

int main() {
  int pass = 1;
  int scale;
  int data_in[N];
  int data_out1[N], data_out2[N];
  int expected_out1[N], expected_out2[N];


  for (int i = 1; i < 4; ++i) {
    scale = i;
    generate_test_data(scale, data_in, expected_out1, expected_out2);


    foo(data_in, scale, data_out1, data_out2);

    if(!compare_array_eq(data_out1, expected_out1) || !compare_array_eq(
    ↪ data_out2, expected_out2)){
      pass = 0;
    }

  }

  if (pass) {
    fprintf(stdout, "----------Pass!----------\n");
    return 0;
  } else  {
    fprintf(stderr, "----------Fail!----------\n");
    return 1;
  }
}
```

Рис. 5.3. Исходный код теста

5.2. Директивы

В данном решения были установлены директивы, приведённые ниже.

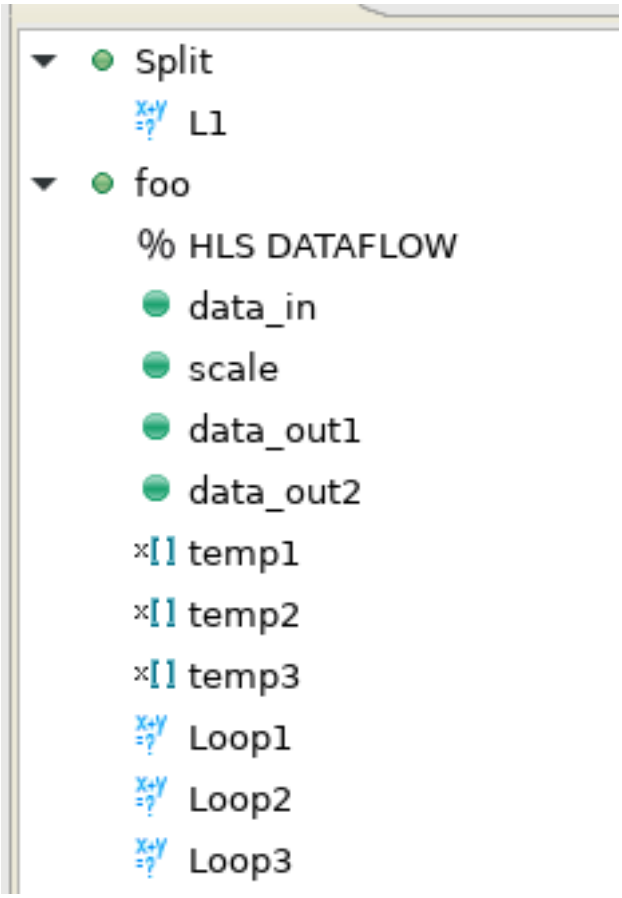

Рис. 5.4. Директивы

5.3. Моделирование

Ниже приведены результаты моделирования.

```
INFO: [APCC 202-1] APCC is done.
   Generating csim.exe
----------Pass!-----------
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
```

Рис. 5.5. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

5.4. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.470	0.10

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
205	205	82	82	dataflow

Рис. 5.6. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	56
FIFO	-	-	-	-
Instance	-	9	271	391
Memory	0	-	192	30
Multiplexer	-	-	-	18
Register	-	-	2	-
Total	0	9	465	495
Available	40	40	16000	8000
Utilization (%)	0	22	2	6

Рис. 5.7. Utilization estimates

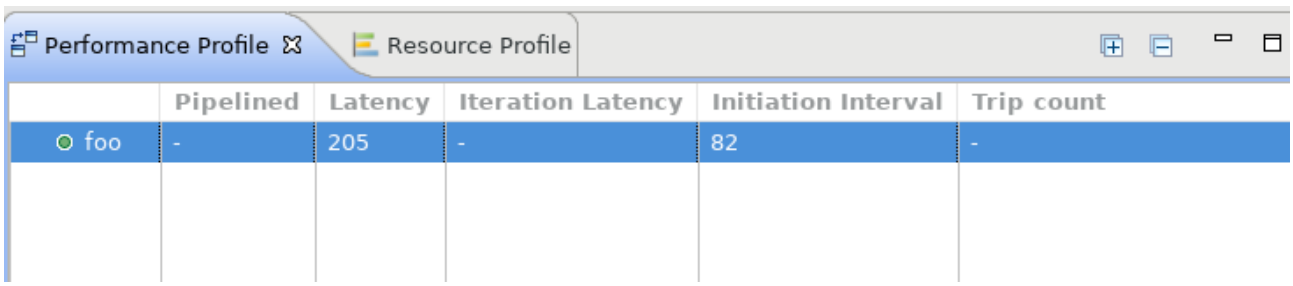

Рис. 5.8. Performance profile

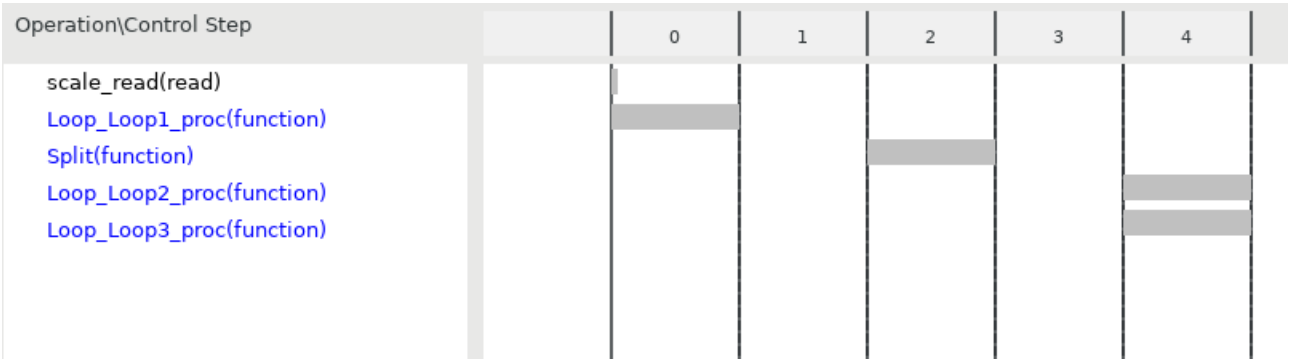

Рис. 5.9. Scheduler viewer

	Resource\Control Step	C0	C1	C2	C3	C4	C5
1	⊟I/O Ports						
2	scale	read					
3	⊟Instances						
4	Loop_Loop1_proc_U0	call					
5	Split_U0			call			
6	Loop_Loop2_proc_U0					call	
7	Loop_Loop3_proc_U0					call	

Рис. 5.10. Resource viewer

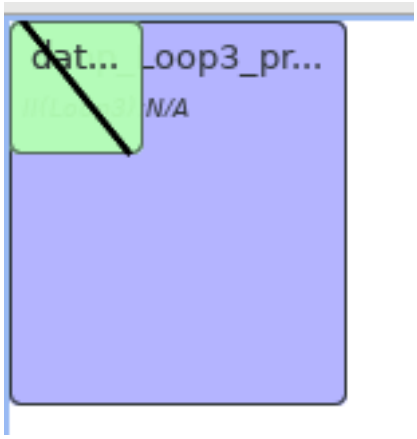

Рис. 5.11. Dataflow viewer

6. Вывод

В данной лабораторной работе были рассмотрены варианты применения директивы DATAFLOW.

В первом решении не используются директивы, выполнение циклов в функции происходит последовательно. В случае, когда добавляется директива DATAFLOW для функции, между функциями добавляются буферы данных, что позволяет циклам работать параллельно. Количество требуемых ресурсов выше чем у первого случая.

В третьем решении, вместо буферов FIFO используются буферы ping-pong,что сказывается негативно на производительности.