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 2

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

Санкт-Петербург
2019

Содержание

1. Задание

1. Создать проект lab8_2
2. Микросхема: xa7a12tcsg325-1q
3. Создать две функции (см. Текст ниже) – исходную и модифицированную - и провести их анализ.

   Bypassing Tasks

   Data should generally flow from one task to another. If you bypass tasks, this reduces

   the performance of the DATAFLOW optimization. In the following example, Loop1 generates

   the values for temp1and temp2. However, the next task, Loop2, only uses the value of temp1.

   The value of temp2 is not consumed until after Loop2. Therefore, temp2 bypasses the next task

   in the sequence, which limits the performance of the DATAFLOW optimization

   void foo_b(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;

   temp2[i] = data_in[i] » scale;

   }

   Loop2: for(int j = 0; j < N; j++) {

   temp3[j] = temp1[j] + 123;

   }

   Loop3: for(int k = 0; k < N; k++) {

   data_out[k] = temp2[k] + temp3[k];

   }

   }

   Because the loop iteration limits are all the same in this example, you can modify the code so

   that Loop2 consumes temp2 and produces temp4 as follows. This ensures that the data flow

   from one task to the next.

   void foo_m(int data_in[N], int scale, int data_out1[N], int data_out2[N]) {

   int temp1[N], temp2[N]. temp3[N], temp4[N];

   Loop1: for(int i = 0; i < N; i++) {

   temp1[i] = data_in[i] * scale;

   temp2[i] = data_in[i] » scale;

}
Loop2: for(int j = 0; j < N; j++) {
temp3[j] = temp1[j] + 123;
temp4[j] = temp2[j];
}
Loop3: for(int k = 0; k < N; k++) {
data_out[k] = temp4[k] + temp3[k];
}
}

4. Создать тест lab8_2_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_2_b
add_files lab8_2_b.c
set_top foo
add_files -tb lab8_2_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_2_m
add_files lab8_2_m.c
set_top foo
add_files -tb lab8_2_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_2.h"
2
3  void foo(int data_in[N], int scale, int data_out[N]) {
4    int temp1[N], temp2[N], temp3[N];
5    Loop1: for(int i = 0; i < N; i++) {
6      temp1[i] = data_in[i] * scale;
7      temp2[i] = data_in[i] >> scale;
8    }
9    Loop2: for(int j = 0; j < N; j++) {
10     temp3[j] = temp1[j] + 123;
11   }
12   Loop3: for(int k = 0; k < N; k++) {
13     data_out[k] = temp2[k] + temp3[k];
14   }
15 }
```

Рис. 3.1. Исходный код устройства

```
1  #define N 10
```

Рис. 3.2. Заголовочный файл

```c
#include <stdio.h>
#include "lab8_2.h"

int main() {
  int data_in[N];
  int data_out[N];
  int data_out_expected[N];
  int scale = 2;
  int pass = 1;

  int i, j;

  for (i = 0; i < N; i++) {
    data_in[i] = 211*i % 9;
    int temp1 = data_in[i] * scale + 123;
    int temp2 = data_in[i] >> scale;
    data_out_expected[i] = temp1 + temp2;
  }

  foo(data_in,scale,data_out);

  for (i = 0; i < N; i++) {
    printf("Expected:[%d], \tActual:[%d]\n",data_out_expected[i],data_out[i] );
    if (data_out_expected[i] != data_out[i] ) {
      pass = 0;

    }
  }
  if(pass){
    printf("-------Test Pass-------\n");
    return 0;
  } else {
    printf("-------ERROR-------\n");
    return -1;
  }
}
```

Рис. 3.3. Исходный код теста

3.2. Моделирование

Ниже приведены результаты моделирования.

```
INFO: [APCC 202-1] APCC is done.
   Generating csim.exe
Expected:[123],         Actual:[123]
Expected:[132],         Actual:[132]
Expected:[141],         Actual:[141]
Expected:[129],         Actual:[129]
Expected:[138],         Actual:[138]
Expected:[127],         Actual:[127]
Expected:[136],         Actual:[136]
Expected:[125],         Actual:[125]
Expected:[134],         Actual:[134]
Expected:[123],         Actual:[123]
-------Test 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
83	83	83	83	none

Рис. 3.5. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	3	0	266
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	192	15
Multiplexer	-	-	-	119
Register	-	-	109	-
Total	0	3	301	400
Available	40	40	16000	8000
Utilization (%)	0	7	1	5

Рис. 3.6. Utilization estimates

Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
foo	-	83	-	84	-
Loop1	no	40	4	-	10
Loop2	no	20	2	-	10
Loop3	no	20	2	-	10

Рис. 3.7. Performance profile

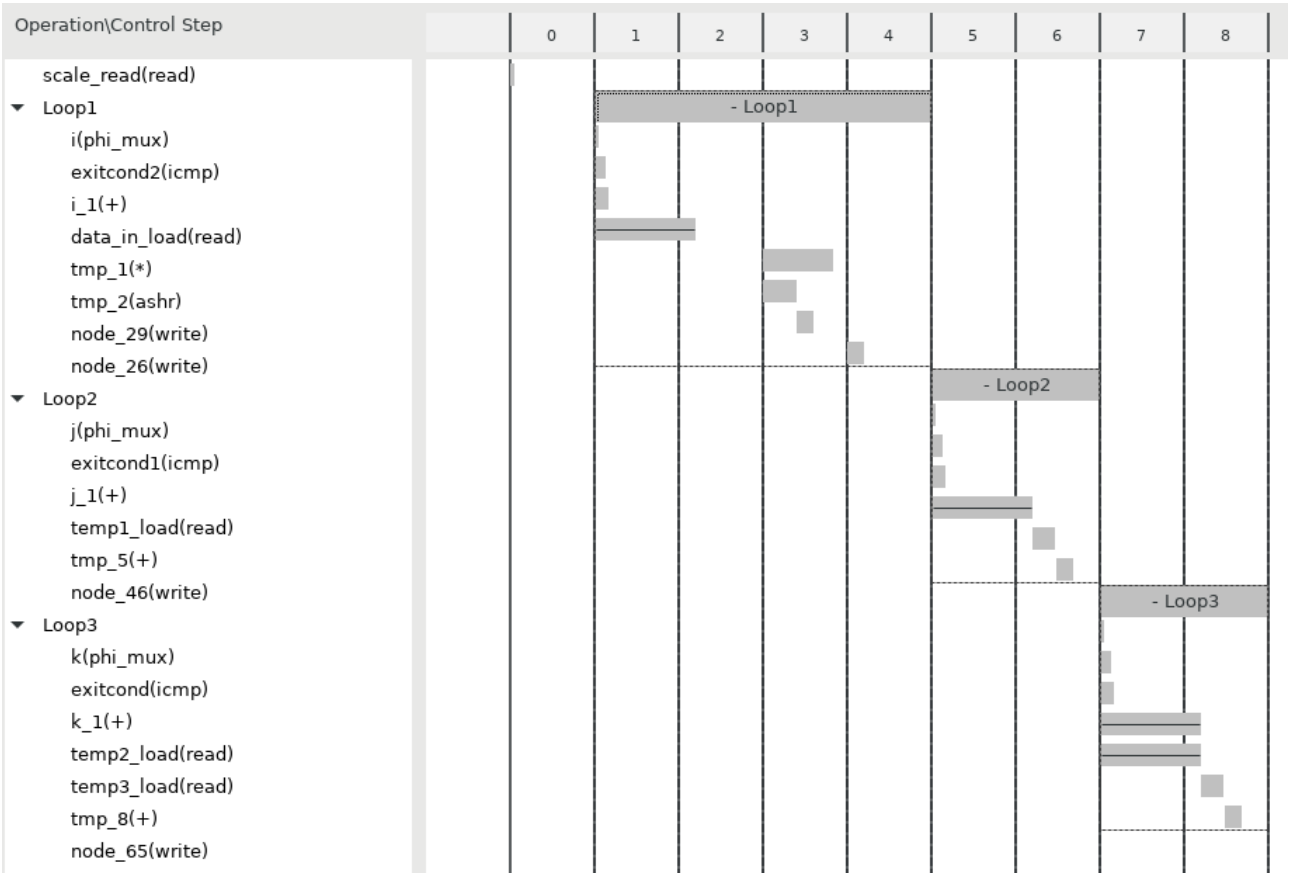

Рис. 3.8. Scheduler viewer

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8
1	I/O Ports									
2	scale	read								
3	data_in(p0)		read							
4	data_out(p0)									write
5	Memory Ports									
6	data_in(p0)		read							
7	temp2(p0)				write				read	
8	temp1(p0)					write	read			
9	temp3(p0)							write	read	
10	data_out(p0)									write
11	Expressions									
12	i_phi_fu_143		phi_mux							
13	i_1_fu_178		+							
14	exitcond2_fu_172		icmp							
15	tmp_2_fu_193				ashr					
16	tmp_1_fu_189				*					
17	j_phi_fu_154						phi_mux			
18	j_1_fu_204						+			
19	exitcond1_fu_198						icmp			
20	tmp_5_fu_215							+		
21	k_phi_fu_165								phi_mux	
22	k_1_fu_228								+	
23	exitcond_fu_222								icmp	
24	tmp_8_fu_240									+

Рис. 3.9. Resource viewer

4. Решение 2

4.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
1  #include "lab8_2.h"
2
3  void foo(int data_in[N], int scale, int data_out[N]) {
4    int temp1[N], temp2[N], temp3[N], temp4[N];
5    Loop1: for(int i = 0; i < N; i++) {
6      temp1[i] = data_in[i] * scale;
7      temp2[i] = data_in[i] >> scale;
8    }
9    Loop2: for(int j = 0; j < N; j++) {
10     temp3[j] = temp1[j] + 123;
11     temp4[j] = temp2[j];
12   }
13   Loop3: for(int k = 0; k < N; k++) {
14     data_out[k] = temp4[k] + temp3[k];
15   }
16 }
```

Рис. 4.1. Исходный код устройства

```
1  #define N 10
```

Рис. 4.2. Заголовочный файл

```c
#include <stdio.h>
#include "lab8_2.h"

int main() {
  int data_in[N];
  int data_out[N];
  int data_out_expected[N];
  int scale = 2;
  int pass = 1;

  int i, j;

  for (i = 0; i < N; i++) {
    data_in[i] = 211*i % 9;
    int temp1 = data_in[i] * scale + 123;
    int temp2 = data_in[i] >> scale;
    data_out_expected[i] = temp1 + temp2;
  }

  foo(data_in,scale,data_out);

  for (i = 0; i < N; i++) {
    printf("Expected:[%d], \tActual:[%d]\n",data_out_expected[i],data_out[i] );
    if (data_out_expected[i] != data_out[i] ) {
      pass = 0;

    }
  }
  if(pass){
    printf("-------Test Pass-------\n");
    return 0;
  } else {
    printf("-------ERROR-------\n");
    return -1;
  }
}
```

Рис. 4.3. Исходный код теста

4.2. Моделирование

Ниже приведены результаты моделирования.

```
INFO: [APCC 202-1] APCC is done.
   Generating csim.exe
Expected:[123],         Actual:[123]
Expected:[132],         Actual:[132]
Expected:[141],         Actual:[141]
Expected:[129],         Actual:[129]
Expected:[138],         Actual:[138]
Expected:[127],         Actual:[127]
Expected:[136],         Actual:[136]
Expected:[125],         Actual:[125]
Expected:[134],         Actual:[134]
Expected:[123],         Actual:[123]
-------Test Pass-------
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
```

Рис. 4.4. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

4.3. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.762	0.10

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
44	44	42	42	dataflow

Рис. 4.5. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	8
FIFO	0	-	24	184
Instance	-	3	195	535
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	3	219	727
Available	40	40	16000	8000
Utilization (%)	0	7	1	9

Рис. 4.6. Utilization estimates

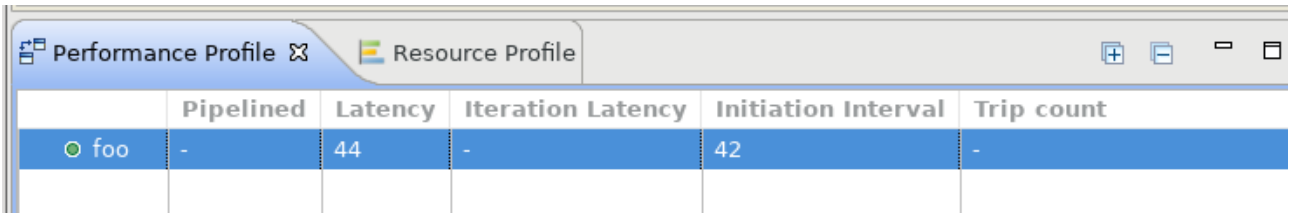

Рис. 4.7. Performance profile

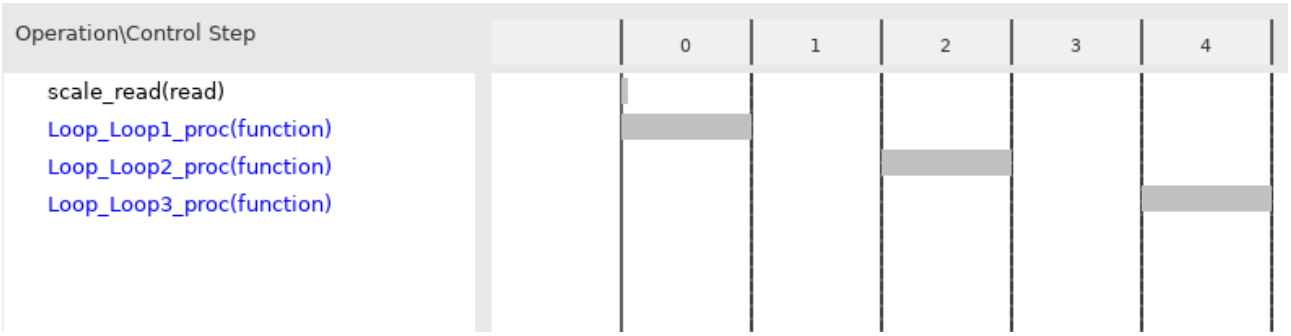

Рис. 4.8. Scheduler viewer

	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	Loop_Loop2_proc_U0			call			
6	Loop_Loop3_proc_U0					call	

Рис. 4.9. Resource viewer

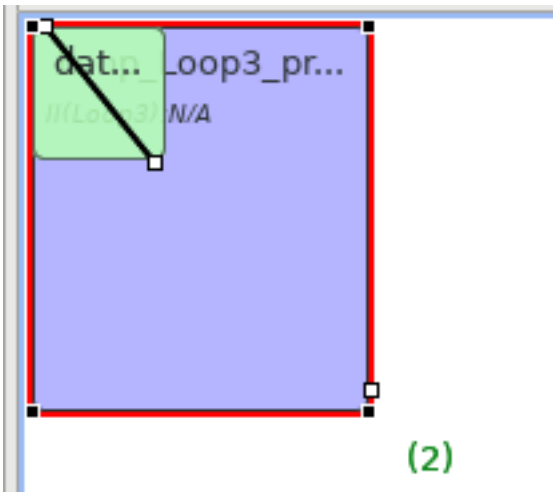

Рис. 4.10. Dataflow viewer

4.4. C/RTL моделирование

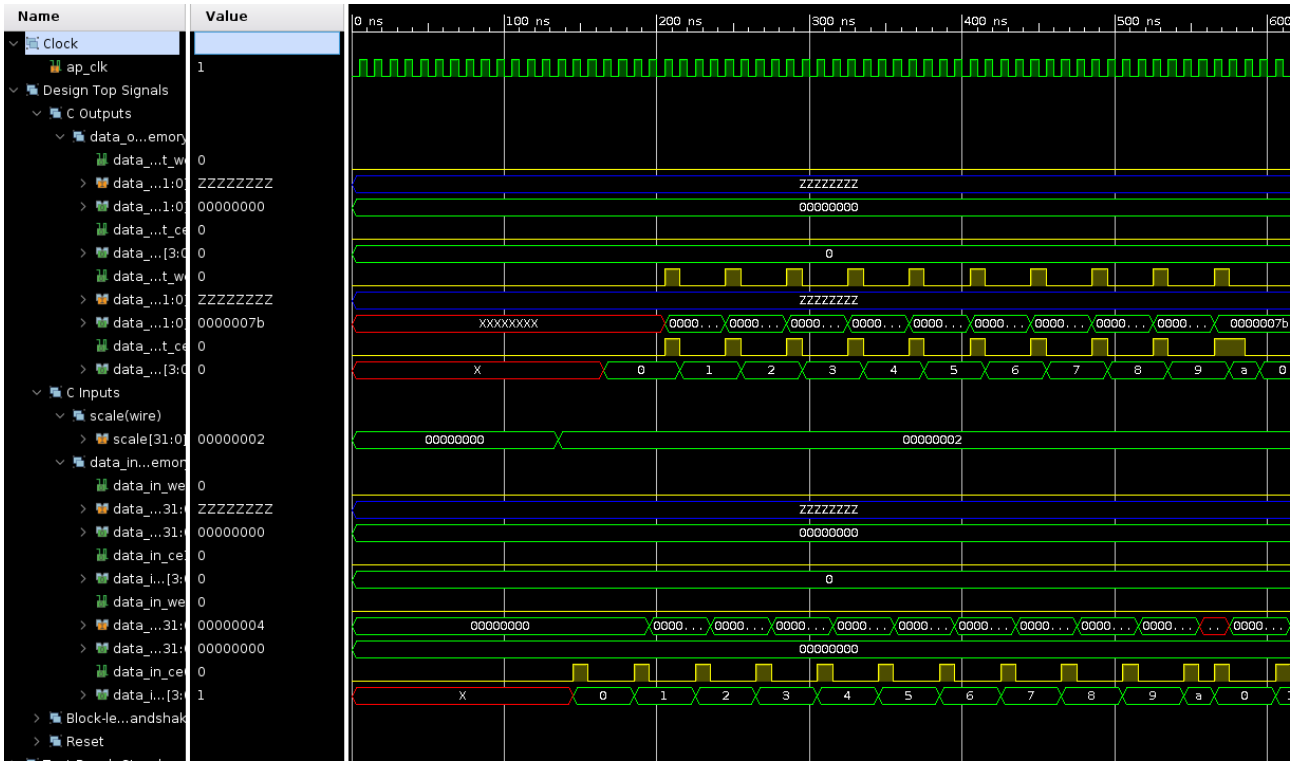

Рис. 4.11. Временная диаграмма

5. Решение 3

5.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
1  #include "lab8_2.h"
2
3  void foo(int data_in[N], int scale, int data_out[N]) {
4    int temp1[N], temp2[N], temp3[N], temp4[N];
5    Loop1: for(int i = 0; i < N; i++) {
6      temp1[i] = data_in[i] * scale;
7      temp2[i] = data_in[i] >> scale;
8    }
9    Loop2: for(int j = 0; j < N; j++) {
10     temp3[j] = temp1[j] + 123;
11     temp4[j] = temp2[j];
12   }
13   Loop3: for(int k = 0; k < N; k++) {
14     data_out[k] = temp4[k] + temp3[k];
15   }
16 }
```

Рис. 5.1. Исходный код устройства

```
1 #define N 10
```

Рис. 5.2. Заголовочный файл

```
1  #include <stdio.h>
2  #include "lab8_2.h"
3
4  int main() {
5    int data_in[N];
6    int data_out[N];
7    int data_out_expected[N];
8    int scale = 2;
9    int pass = 1;
10
11   int i, j;
12
13   for (i = 0; i < N; i++) {
14     data_in[i] = 211*i % 9;
15     int temp1 = data_in[i] * scale + 123;
16     int temp2 = data_in[i] >> scale;
17     data_out_expected[i] = temp1 + temp2;
18   }
19
20   foo(data_in,scale,data_out);
21
22   for (i = 0; i < N; i++) {
23     printf("Expected:[%d], \tActual:[%d]\n",data_out_expected[i],data_out[i] );
24     if (data_out_expected[i] != data_out[i] ) {
25       pass = 0;
26
27     }
28   }
29   if(pass){
30     printf("--------Test Pass--------\n");
31     return 0;
32   } else {
33     printf("--------ERROR--------\n");
34     return -1;
35   }
36 }
```

Рис. 5.3. Исходный код теста

5.2. Моделирование

Ниже приведены результаты моделирования.

```
INFO: [APCC 202-1] APCC is done.
   Generating csim.exe
Expected:[123],         Actual:[123]
Expected:[132],         Actual:[132]
Expected:[141],         Actual:[141]
Expected:[129],         Actual:[129]
Expected:[138],         Actual:[138]
Expected:[127],         Actual:[127]
Expected:[136],         Actual:[136]
Expected:[125],         Actual:[125]
Expected:[134],         Actual:[134]
Expected:[123],         Actual:[123]
-------Test Pass-------
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
```

Рис. 5.4. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

5.3. 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
85	85	42	42	dataflow

Рис. 5.5. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	104
FIFO	-	-	-	-
Instance	-	3	114	419
Memory	0	-	256	20
Multiplexer	-	-	-	36
Register	-	-	4	-
Total	0	3	374	579
Available	40	40	16000	8000
Utilization (%)	0	7	2	7

Рис. 5.6. Utilization estimates

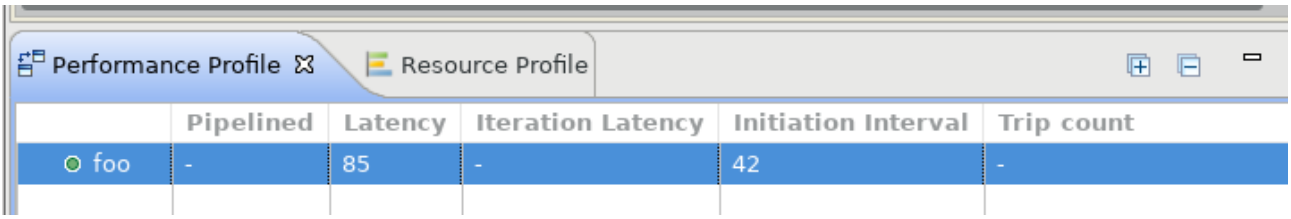

Рис. 5.7. Performance profile

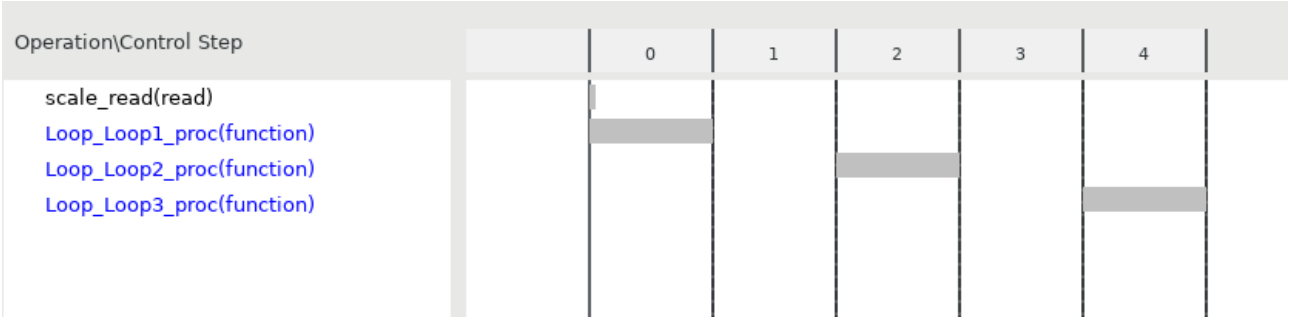

Рис. 5.8. Scheduler viewer

	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	Loop_Loop2_proc_U0			call			
6	Loop_Loop3_proc_U0					call	

Рис. 5.9. Resource viewer

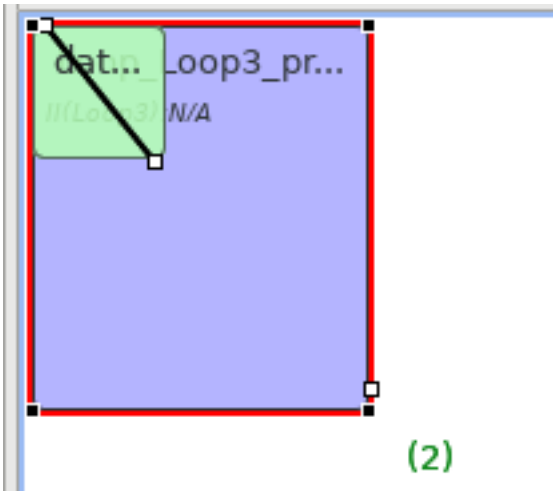

Рис. 5.10. Dataflow viewer

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.