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 3

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

Санкт-Петербург
2019

Содержание

1. Задание

1. Создать проект lab8_3
2. Микросхема: xa7a12tcsg325-1q
3. Создать две функции (см. Текст ниже) – исходную и модифицированную - и провести их анализ.

   Conditional Execution of Tasks

   The DATAFLOW optimization does not optimize tasks that are conditionally executed. The

   following example highlights this limitation. In this example, the conditional execution of Loop1

   and Loop2 prevents Vivado HLS from optimization the data flow between these loops, because

   the data does not flow from one loop into the next.

   void foo_b(int data_in1[N], int data_out[N], int sel) {

   int temp1[N], temp2[N];

   if (sel) {

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

   temp1[i] = data_in[i] * 123;

   temp2[i] = data_in[i];

   }

   } else {

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

   temp1[j] = data_in[j] * 321;

   temp2[j] = data_in[j];

   }

   }

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

   data_out[k] = temp1[k] * temp2[k];

   }

   }

   To ensure each loop is executed in all cases, you must transform the code as shown in the

   following example. In this example, the conditional statement is moved into the first loop. Both

   loops are always executed, and data always flows from one loop to the next.

   void foo_m(int data_in[N], int data_out[N], int sel) {

   int temp1[N], temp2[N];

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

if (sel) {
temp1[i] = data_in[i] * 123;
} else {
temp1[i] = data_in[i] * 321;
}
Loop2: for(int j = 0; j < N; j++) {
temp2[j] = data_in[j];
}
Loop3: for(int k = 0; k < N; k++) {
data_out[k] = temp1[k] * temp2[k];
}
}

4. Создать тест lab8_3_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_3_b
add_files lab8_3_b.c
set_top foo
add_files -tb lab8_3_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_3_m
add_files lab8_3_m.c
set_top foo
add_files -tb lab8_3_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_3.h"
2
3 void foo(int data_in[N], int sel, int data_out[N]) {
4   int temp1[N], temp2[N];
5   if (sel) {
6     Loop1: for(int i = 0; i < N; i++) {
7       temp1[i] = data_in[i] * 123;
8       temp2[i] = data_in[i];
9     }
10  } else {
11    Loop2: for(int j = 0; j < N; j++) {
12      temp1[j] = data_in[j] * 321;
13      temp2[j] = data_in[j];
14    }
15  }
16  Loop3: for(int k = 0; k < N; k++) {
17    data_out[k] = temp1[k] * temp2[k];
18  }
19 }
```

Рис. 3.1. Исходный код устройства

```
1 #define N 10
```

Рис. 3.2. Заголовочный файл

```c
#include <stdio.h>
#include "lab8_3.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] * 123;
    int temp2 = data_in[i];
    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) {
    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.

```
   Generating csim.exe
Expected:[0],   Actual:[0]
Expected:[1968],        Actual:[1968]
Expected:[7872],        Actual:[7872]
Expected:[1107],        Actual:[1107]
Expected:[6027],        Actual:[6027]
Expected:[492],         Actual:[492]
Expected:[4428],        Actual:[4428]
Expected:[123],         Actual:[123]
Expected:[3075],        Actual:[3075]
Expected:[0],   Actual:[0]
----------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
82	82	82	82	none

Рис. 3.5. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	9	0	147
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	128	10
Multiplexer	-	-	-	155
Register	-	-	241	-
Total	0	9	369	312
Available	40	40	16000	8000
Utilization (%)	0	22	2	3

Рис. 3.6. Utilization estimates

Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
foo	-	82	-	83	-
Loop1	no	40	4	-	10
Loop2	no	40	4	-	10
Loop3	no	40	4	-	10

Рис. 3.7. Performance profile

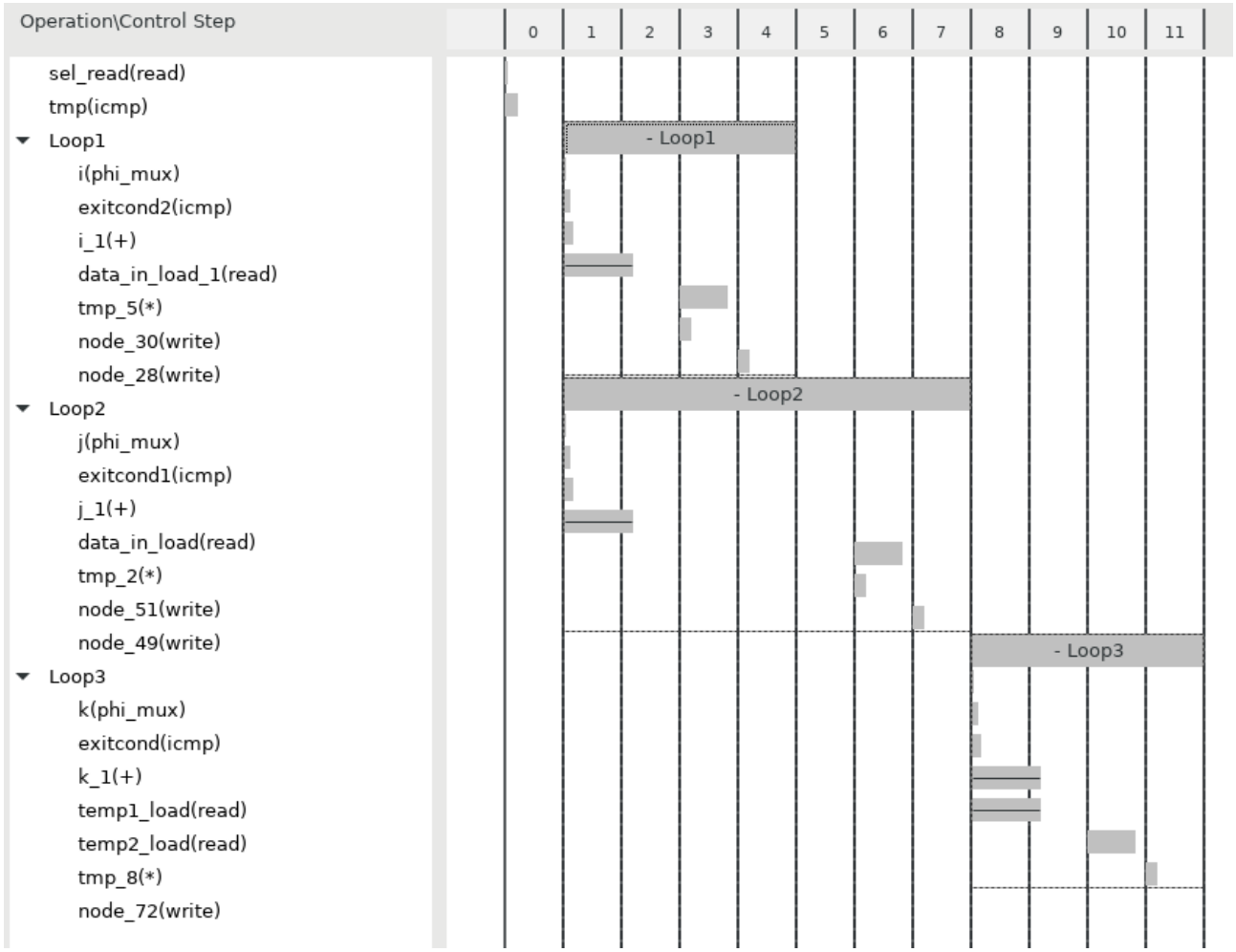

Рис. 3.8. Scheduler viewer

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
1	I/O Ports												
2	sel	read											
3	data_in(p0)		read										
4	data_out(p0)												write
5	Memory Ports												
6	data_in(p0)		read										
7	temp2(p0)				write			write		read			
8	temp1(p0)					write			write	read			
9	data_out(p0)												write
10	Expressions												
11	tmp_fu_180	icmp											
12	j_1_fu_209		+										
13	i_1_fu_192		+										
14	j_phi_fu_157		phi_mux										
15	i_phi_fu_146		phi_mux										
16	exitcond1_fu_203		icmp										
17	exitcond2_fu_186		icmp										
18	tmp_5_fu_220				*								
19	tmp_2_fu_226							*					
20	k_1_fu_238									+			
21	k_phi_fu_168									phi_mux			
22	exitcond_fu_232									icmp			
23	tmp_8_fu_250											*	

Рис. 3.9. Resource viewer

4. Решение 2

4.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
1 #include "lab8_3.h"
2
3 void foo(int data_in[N], int sel, int data_out[N]) {
4   int temp1[N], temp2[N];
5   Loop1: for(int i = 0; i < N; i++) {
6     if (sel) {
7       temp1[i] = data_in[i] * 123;
8     } else {
9       temp1[i] = data_in[i] * 321;
10    }
11    Loop2: for(int j = 0; j < N; j++) {
12      temp2[j] = data_in[j];
13    }
14    Loop3: for(int k = 0; k < N; k++) {
15      data_out[k] = temp1[k] * temp2[k];
16    }
17  }
18 }
```

Рис. 4.1. Исходный код устройства

```
1 #define N 10
```

Рис. 4.2. Заголовочный файл

```c
#include <stdio.h>
#include "lab8_3.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] * 123;
    int temp2 = data_in[i];
    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) {
    fprintf(stdout, "----------Pass!----------\n");
    return 0;
  } else  {
    fprintf(stderr, "----------Fail!----------\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:[0],   Actual:[0]
Expected:[1968],        Actual:[1968]
Expected:[7872],        Actual:[7872]
Expected:[1107],        Actual:[1107]
Expected:[6027],        Actual:[6027]
Expected:[492],         Actual:[492]
Expected:[4428],        Actual:[4428]
Expected:[123],         Actual:[123]
Expected:[3075],        Actual:[3075]
Expected:[0],   Actual:[0]
----------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	8.470	0.10

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
661	661	662	662	dataflow

Рис. 4.5. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	0	9	370	323
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	9	370	323
Available	40	40	16000	8000
Utilization (%)	0	22	2	4

Рис. 4.6. Utilization estimates

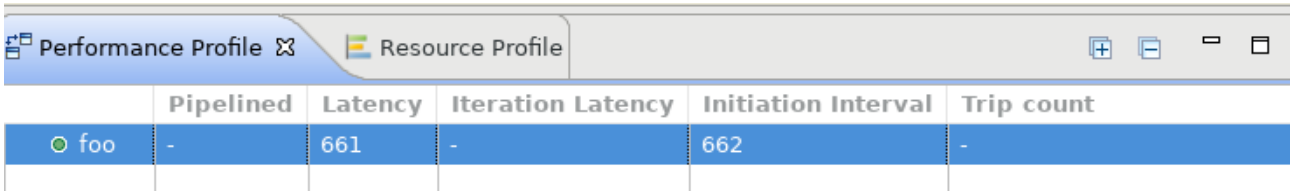

Рис. 4.7. Performance profile

Рис. 4.8. Scheduler viewer

	Resource\Control Step	C0	C1
1	⊟ I/O Ports		
2	sel	read	
3	⊟ Instances		
4	Loop_Loop1_proc4_U0	call	

Рис. 4.9. Resource viewer

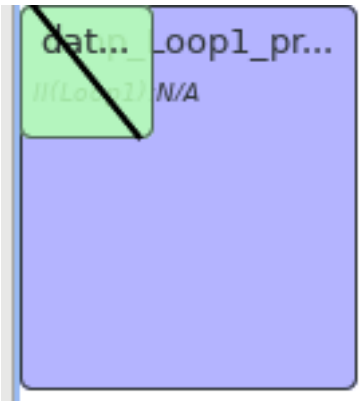

Рис. 4.10. Dataflow viewer

4.4. C/RTL моделирование

Рис. 4.11. Временная диаграмма

5. Решение 3

5.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
#include "lab8_3.h"

void foo(int data_in[N], int sel, int data_out[N]) {
  int temp1[N], temp2[N];
  Loop1: for(int i = 0; i < N; i++) {
    if (sel) {
      temp1[i] = data_in[i] * 123;
    } else {
      temp1[i] = data_in[i] * 321;
    }
    Loop2: for(int j = 0; j < N; j++) {
      temp2[j] = data_in[j];
    }
    Loop3: for(int k = 0; k < N; k++) {
      data_out[k] = temp1[k] * temp2[k];
    }
  }
}
```

Рис. 5.1. Исходный код устройства

```
1 #define N 10
```

Рис. 5.2. Заголовочный файл

```
1  #include <stdio.h>
2  #include "lab8_3.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   int i, j;
11
12   for (i = 0; i < N; i++) {
13     data_in[i] = 211*i % 9;
14     int temp1 = data_in[i] * 123;
15     int temp2 = data_in[i];
16     data_out_expected[i] = temp1 * temp2;
17   }
18
19   foo(data_in,scale,data_out);
20
21   for (i = 0; i < N; i++) {
22     printf("Expected:[%d], \tActual:[%d]\n",data_out_expected[i],data_out[i] );
23     if (data_out_expected[i] != data_out[i] ) {
24       pass = 0;
25     }
26   }
27   if (pass) {
28     fprintf(stdout, "----------Pass!----------\n");
29     return 0;
30   } else {
31     fprintf(stderr, "----------Fail!----------\n");
32     return 1;
33   }
34 }
```

Рис. 5.3. Исходный код теста

5.2. Моделирование

Ниже приведены результаты моделирования.

```
INFO: [APCC 202-1] APCC is done.
   Generating csim.exe
Expected:[0],   Actual:[0]
Expected:[1968],        Actual:[1968]
Expected:[7872],        Actual:[7872]
Expected:[1107],        Actual:[1107]
Expected:[6027],        Actual:[6027]
Expected:[492],         Actual:[492]
Expected:[4428],        Actual:[4428]
Expected:[123],         Actual:[123]
Expected:[3075],        Actual:[3075]
Expected:[0],   Actual:[0]
----------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
661	661	662	662	dataflow

Рис. 5.5. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	0	9	370	323
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	9	370	323
Available	40	40	16000	8000
Utilization (%)	0	22	2	4

Рис. 5.6. Utilization estimates

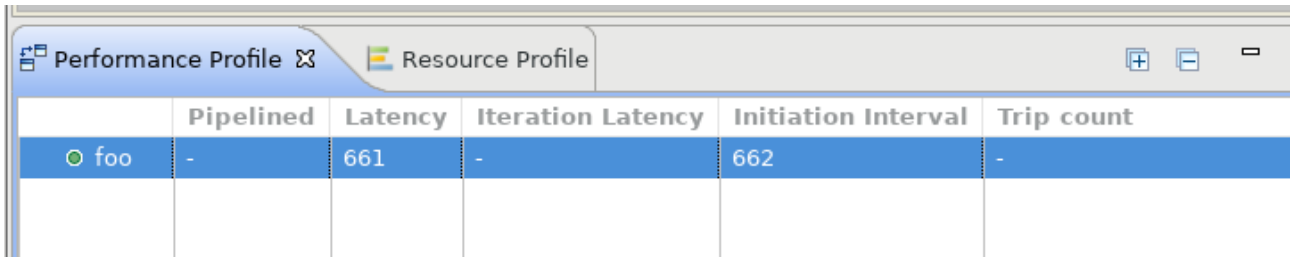

Рис. 5.7. Performance profile

Рис. 5.8. Scheduler viewer

	Resource\Control Step	C0	C1
1	⊟ I/O Ports		
2	sel	read	
3	⊟ Instances		
4	Loop_Loop1_proc4_U0	call	

Рис. 5.9. Resource viewer

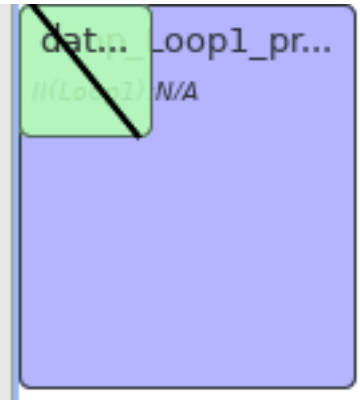

Рис. 5.10. Dataflow viewer

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 остальных решениях, добавление директивы ухудшают ситуацию, так как к данным функциям не применима оптимизация потока данных.