Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчет по лабораторной работе №8\_3**

**Курс: «Проектирование реконфигурируемых гибридных вычислительных систем»**

**Тема: Dataflow.** **Conditional Execution of Tasks**

Студент:\_\_\_\_\_\_Волкова М.Д

Гр. № \_\_\_\_\_\_\_ [3540901/91501](https://vk.com/im?sel=c136)

Преподаватель: Антонов А.П.

Санкт-Петербург

2020

ОГЛАВЛЕНИЕ

[**1. Задание** 3](#_Toc29070801)

[**2. Скрипт** 5](#_Toc29070802)

[**3.Решение №1.** 6](#_Toc29070803)

[3.1. Исходный код программы и теста 6](#_Toc29070804)

[3.2.Моделирование 7](#_Toc29070805)

[3.3.Синтез 7](#_Toc29070806)

[**4. Решение №2.** 10](#_Toc29070807)

[4.1. Исходный код программы и теста 10](#_Toc29070808)

[4.3. Моделирование 11](#_Toc29070809)

[4.4. Синтез 11](#_Toc29070810)

[4.5. С/RTL моделирование 13](#_Toc29070811)

[**5. Решение №3.** 14](#_Toc29070812)

[5.1. Исходный код программы и теста 14](#_Toc29070813)

[5.3. Моделирование 15](#_Toc29070814)

[5.4. Синтез 15](#_Toc29070815)

[**6. Вывод** 17](#_Toc29070816)

# **1. Задание**

* Создать проект lab8\_3
* Микросхема: xa7a12tcsg325-1q
* 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];

}

}

* Создать тест lab8\_3\_test.c для проверки функций выше.
* 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
* 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
* Выводы
  + Объяснить отличия в синтезе foo\_b и двух вариантов foo\_m между собой

# **2. Скрипт**

Представим на рис. 2.1 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 программы и теста

|  |
| --- |
| #include "lab8\_3 .h"  void foo ( int data\_in [N] , int sel , int data\_out [N]) {  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 ] ;  }  } |

Рис. 3.1. Исходный код устройства

|  |
| --- |
| #define N 10 |

Рис. 3.2. Заголовочный файл

|  |
| --- |
| #include <stdio.h>  #include "lab8\_3.h"  int main() {  int pass = 1;  int data\_in [N];  int scale = 2;  int data\_out[N];  int data\_out\_expected [N] ;  int i, j;    for (int 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\_b(data\_out, data\_in, scale);  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 ("------Pass!------\n");  return 0;  } else {  fprintf ( "------Fail!------\n");  return 1;  }  } |

Рис. 3.3. Исходный код теста

## 3.2.Моделирование

Ниже приведены результаты успешного моделирования.

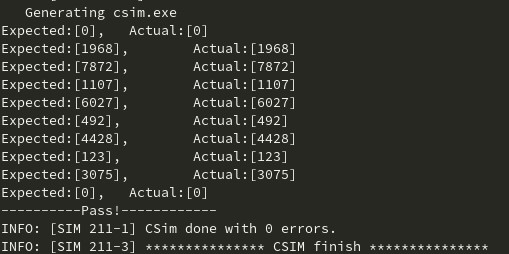


Рис. 3.4. Результаты моделирования

## 3.3.Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

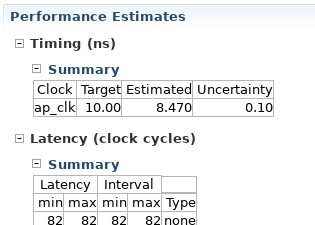


Рис. 3.5. Performance estimates

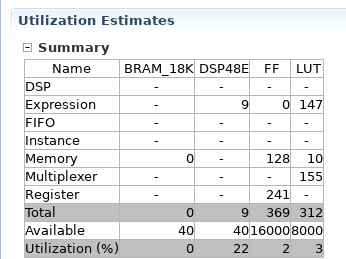


Рис. 3.6. Utilization estimates

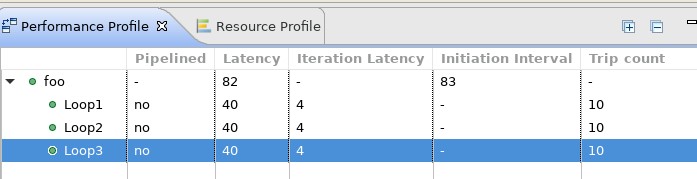


Рис. 3.7. Performance profile

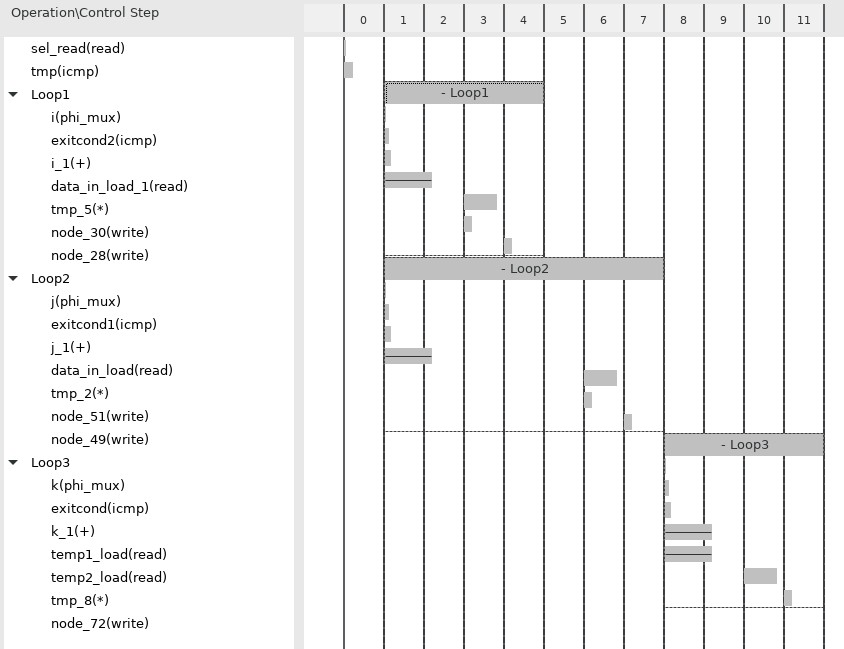


Рис. 3.8. Scheduler viewer



Рис. 3.9. Resource viewer

# **4. Решение №2.**

## 4.1. Исходный код программы 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. 4.1. Исходный код устройства

|  |
| --- |
| #define N 10 |

Рис. 4.2. Заголовочный файл

|  |
| --- |
| #include <stdio.h>  #include "lab8\_3.h"  int main() {  int pass = 1;  int data\_in [N];  int scale = 2;  int data\_out[N];  int data\_out\_expected [N] ;  int i, j;    for (int 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\_b(data\_out, data\_in, scale);  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 ("------Pass!------\n");  return 0;  } else {  fprintf ( "------Fail!------\n");  return 1;  }  } |

Рис. 4.3. Исходный код теста

## 4.3. Моделирование

Ниже приведены результаты моделирования.

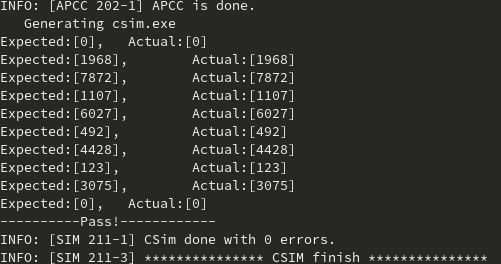


Рис. 4.4. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

## 4.4. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

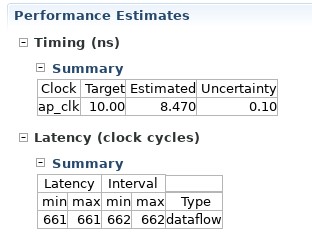


Рис. 4.5. Performance estimates

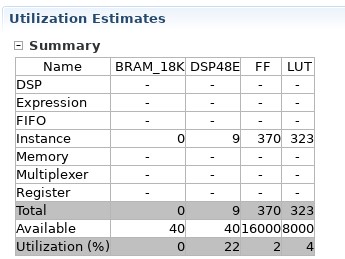


Рис. 4.6. Utilization estimates

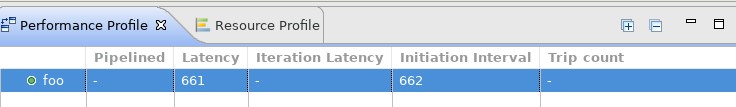


Рис. 4.7. Performance profile



Рис. 4.8. Scheduler viewer

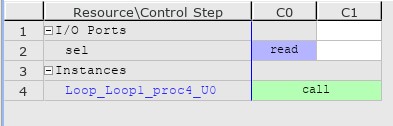


Рис. 4.9. Resource viewer

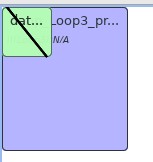


Рис. 4.10. Dataflow viewer

## 4.5. С/RTL моделирование



Рис. 4.11. Временная диаграмма

# **5. Решение №3.**

## 5.1. Исходный код программы 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

|  |
| --- |
| #define N 10 |

Рис. 5.2. Заголовочный файл

|  |
| --- |
| #include <stdio.h>  #include "lab8\_3.h"  int main() {  int pass = 1;  int data\_in [N];  int scale = 2;  int data\_out[N];  int data\_out\_expected [N] ;  int i, j;    for (int 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\_b(data\_out, data\_in, scale);  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 ("------Pass!------\n");  return 0;  } else {  fprintf ( "------Fail!------\n");  return 1;  }  } |

Рис. 5.3. Исходный код теста

## 5.3. Моделирование

Ниже приведены результаты моделирования.

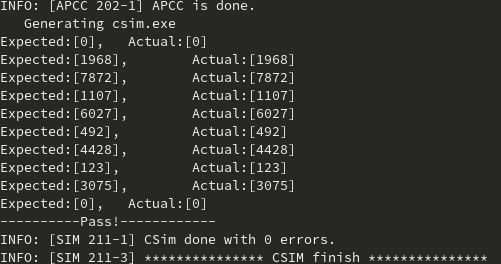


Рис. 5.4. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

## 5.4. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

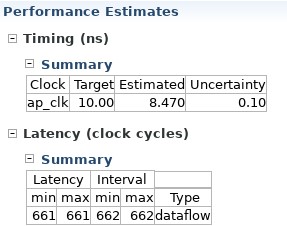


Рис. 5.5. Performance estimates

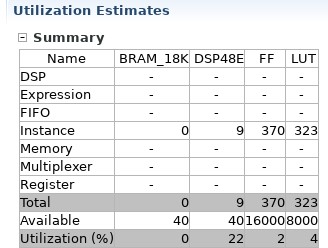


Рис. 5.6. Utilization estimates

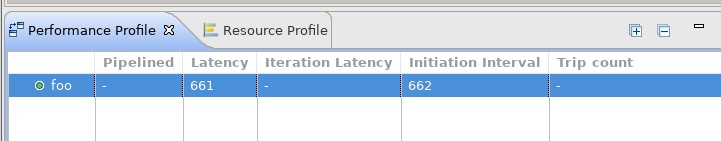


Рис. 5.7. Performance profile



Рис. 5.8. Scheduler viewer

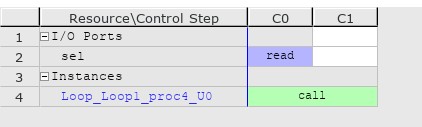


Рис. 5.9. Resource viewer

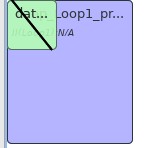


Рис. 5.10. Dataflow viewer

# **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.