Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчет по лабораторной работе №8\_1**

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

**Тема: Dataflow. Single-producer-consumer Violations**

Выполнил студент гр. 3540901/81501 Селиверстов Я.А.

(подпись)

Руководитель Антонов А.П.

(подпись)

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Санкт – Петербург

2019

ОГЛАВЛЕНИЕ

[**1. Задание** 3](#_Toc29051438)

[**2. Скрипт** 5](#_Toc29051439)

[**3.Решение №1.** 6](#_Toc29051440)

[3.1. Исходный код программы и теста 6](#_Toc29051441)

[3.2.Моделирование 8](#_Toc29051442)

[3.3.Синтез 8](#_Toc29051443)

[**4. Решение №2.** 10](#_Toc29051444)

[4.1. Исходный код программы и теста 10](#_Toc29051445)

[4.2. Директивы 12](#_Toc29051446)

[4.3. Моделирование 12](#_Toc29051447)

[4.4. Синтез 12](#_Toc29051448)

[4.5. С/RTL моделирование 14](#_Toc29051449)

[**5. Решение №3.** 15](#_Toc29051450)

[5.1. Исходный код программы и теста 15](#_Toc29051451)

[5.2. Директивы 16](#_Toc29051452)

[5.3. Моделирование 17](#_Toc29051453)

[5.4. Синтез 17](#_Toc29051454)

[**6. Вывод** 19](#_Toc29051455)

# **1. Задание**

* Создать проект lab8\_1
* Микросхема: xa7a12tcsg325-1q
* 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;**

**}**

**}**

* Создать тест lab8\_1\_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\_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 f i f o 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\_1 .h"  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;  }  } |

Рис. 3.1. Исходный код устройства

|  |
| --- |
| #define N 20 |

Рис. 3.2. Заголовочный файл

|  |
| --- |
| #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 data\_in [N];  int scale;  int data\_out1[N];  int data\_out2[N];  int expected\_out1[N];  int expected\_out2[N];      for (int i = 1; i < 4; ++i) {  scale = i;  generate\_test\_data( scale, data\_in, expected\_out1, expected\_out2);    foo\_b(data\_out1, data\_out2, data\_in, scale);    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(stdout, "------Fail!------\n");  return 1;  }  } |

Рис. 3.3. Исходный код теста

## 3.2.Моделирование

Ниже приведены результаты успешного моделирования.

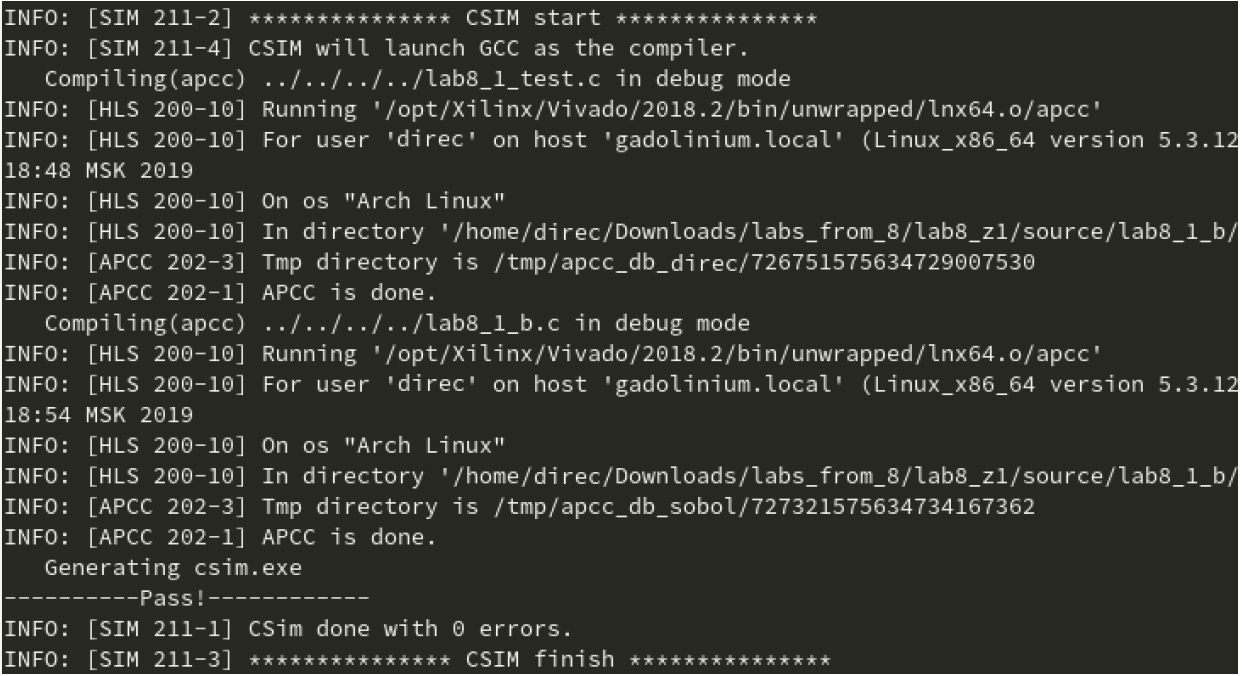


Рис. 3.4. Результаты моделирования

## 3.3.Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

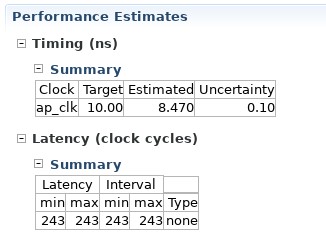


Рис. 3.5. Performance estimates

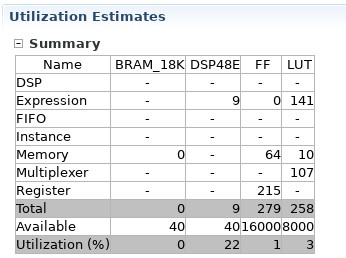


Рис. 3.6. Utilization estimates

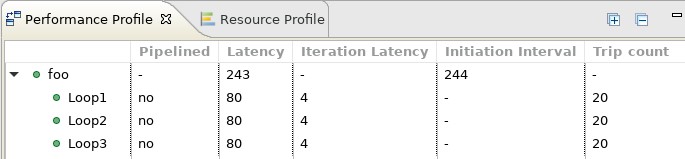


Рис. 3.7. Performance profile

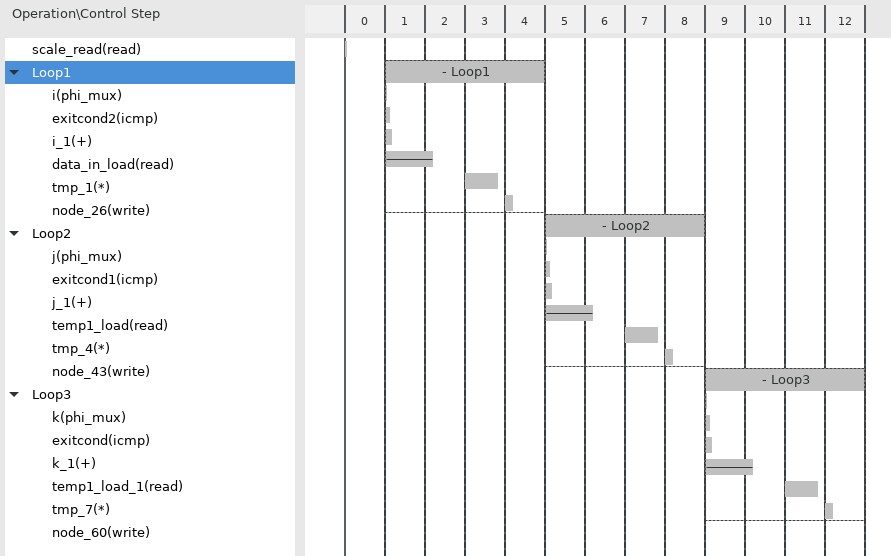


Рис. 3.8. Scheduler viewer

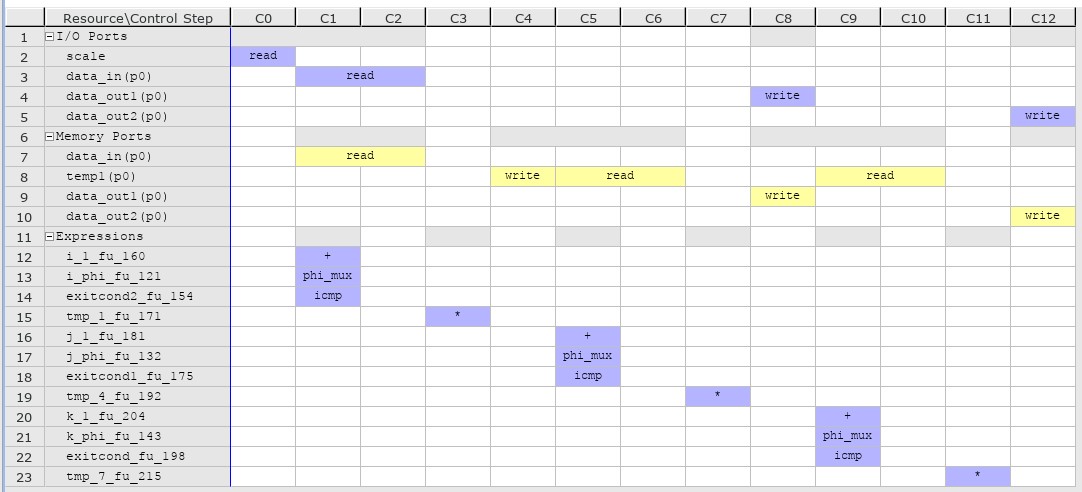


Рис. 3.9. Resource viewer

# **4. Решение №2.**

## 4.1. Исходный код программы и теста

|  |
| --- |
| #include "lab8\_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

|  |
| --- |
| #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 data\_in [N];  int scale;  int data\_out1[N], int data\_out2[N];  int expected\_out1[N], int expected\_out2[N];  for (int i = 1; i < 4; ++i) {  scale = i;  generate\_test\_data( scale, data\_in, expected\_out1, expected\_out2);  foo\_b(data\_out1, data\_out2, data\_in, scale);    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(stdout, "------Fail!------\n");  return 1;  }  } |

Рис. 4.3. Исходный код теста

## 4.2. Директивы

В данном решения были установлены директивы, приведённые ниже.

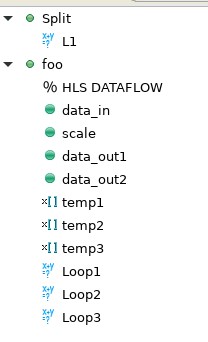


Рис. 4.4. Исходный код теста

## 4.3. Моделирование

Ниже приведены результаты моделирования.

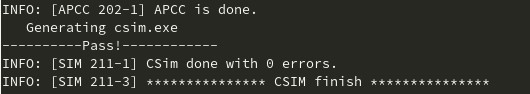


Рис. 4.5. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

## 4.4. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

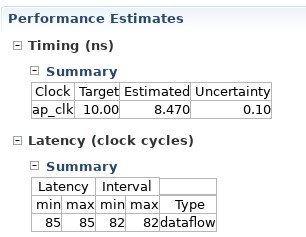


Рис. 4.6. Performance estimates

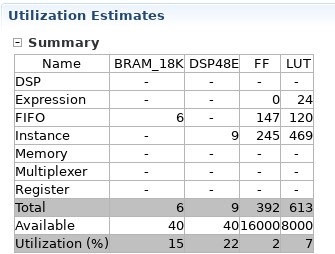


Рис. 4.7. Utilization estimates

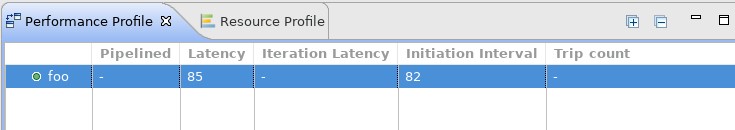


Рис. 4.8. Performance profile

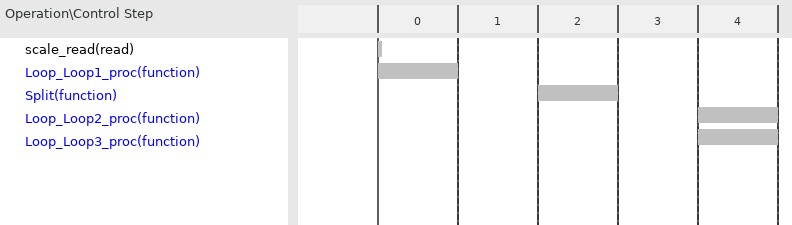


Рис. 4.9. Scheduler viewer

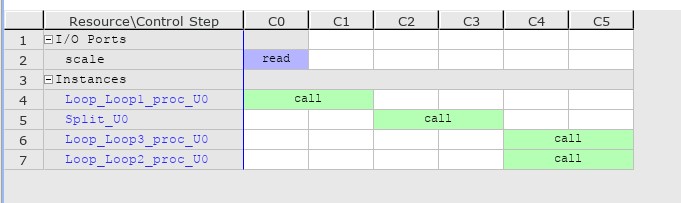


Рис. 4.10. Resource viewer

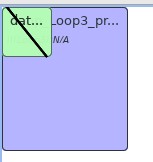


Рис. 4.11. Dataflow viewer

## 4.5. С/RTL моделирование

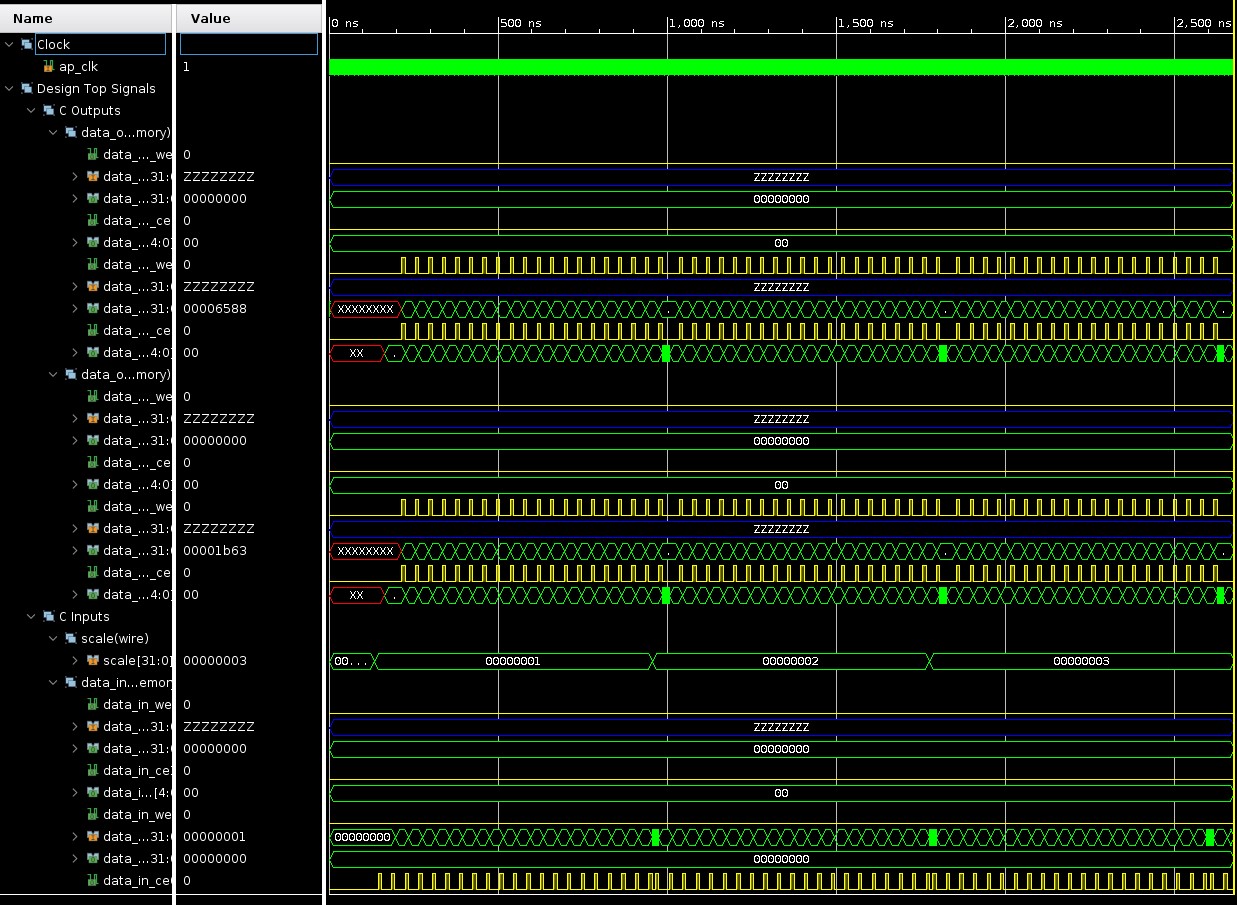


Рис. 4.12. Временная диаграмма

# **5. Решение №3.**

## 5.1. Исходный код программы 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. 5.1. Исходный код устройства

|  |
| --- |
| #define N 20 |

Рис. 5.2. Заголовочный файл

|  |
| --- |
| #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 data\_in [N];  int scale;  int data\_out1[N], int data\_out2[N];  int expected\_out1[N], int expected\_out2[N];  for (int i = 1; i < 4; ++i) {  scale = i;  generate\_test\_data( scale, data\_in, expected\_out1, expected\_out2);  foo\_b(data\_out1, data\_out2, data\_in, scale);    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(stdout, "------Fail!------\n");  return 1;  }  } |

Рис. 5.3. Исходный код теста

## 5.2. Директивы

В данном решения были установлены директивы, приведённые ниже.

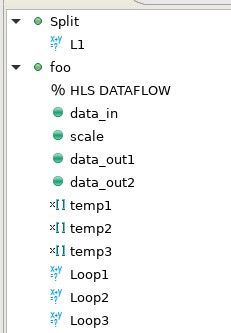


Рис. 5.4. Исходный код теста

## 5.3. Моделирование

Ниже приведены результаты моделирования.

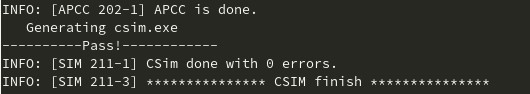


Рис. 5.5. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

## 5.4. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

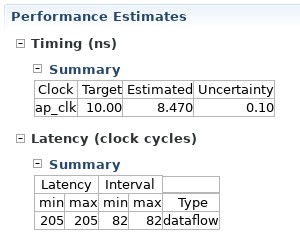


Рис. 5.6. Performance estimates

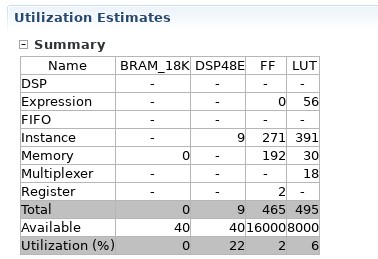


Рис. 5.7. Utilization estimates

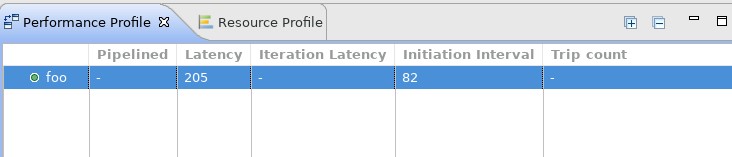


Рис. 5.8. Performance profile

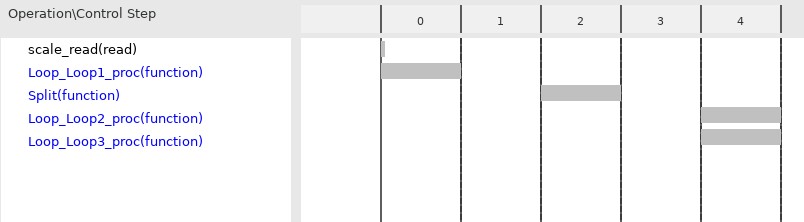


Рис. 5.9. Scheduler viewer

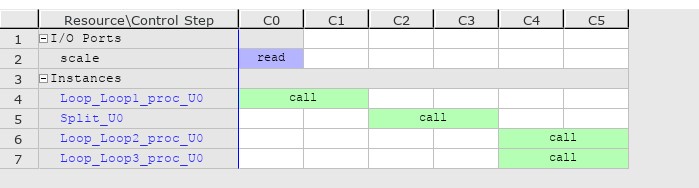


Рис. 5.10. Resource viewer

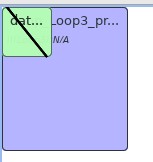


Рис. 5.11. Dataflow viewer

# **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.