Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчет по лабораторной работе №8\_2**

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

**Тема: Dataflow.** **Bypassing Tasks.**

Выполнил студент гр. 3540901/81501 Ерниязов Т.Е.

(подпись)

Руководитель Антонов А.П.

(подпись)

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

Санкт – Петербург

2019

ОГЛАВЛЕНИЕ

[**1. Задание** 3](#_Toc29058818)

[**2. Скрипт** 5](#_Toc29058819)

[**3.Решение №1.** 6](#_Toc29058820)

[3.1. Исходный код программы и теста 6](#_Toc29058821)

[3.2.Моделирование 7](#_Toc29058822)

[3.3.Синтез 7](#_Toc29058823)

[**4. Решение №2.** 10](#_Toc29058824)

[4.1. Исходный код программы и теста 10](#_Toc29058825)

[4.2. Моделирование 11](#_Toc29058826)

[4.3. Синтез 11](#_Toc29058827)

[4.4. С/RTL моделирование 13](#_Toc29058828)

[**5. Решение №3.** 14](#_Toc29058829)

[5.1. Исходный код программы и теста 14](#_Toc29058830)

[5.2. Моделирование 15](#_Toc29058831)

[5.3. Синтез 15](#_Toc29058832)

[**6. Вывод** 17](#_Toc29058833)

# **1. Задание**

* Создать проект lab8\_2
* Микросхема: xa7a12tcsg325-1q
* 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];

}

}

* Создать тест lab8\_2\_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\_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 программы и теста

|  |
| --- |
| #include "lab8\_2 .h"  void foo\_b (int data\_in[N], int scale, int data\_out[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 ] ;  }  } |

Рис. 3.1. Исходный код устройства

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

Рис. 3.2. Заголовочный файл

|  |
| --- |
| #include <stdio.h>  #include "lab8\_2.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.

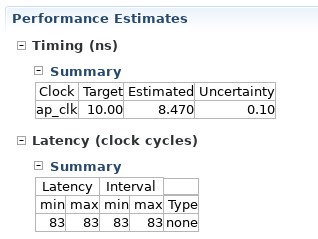


Рис. 3.5. Performance estimates

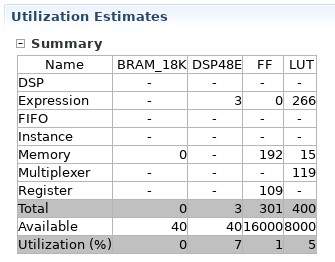


Рис. 3.6. Utilization estimates

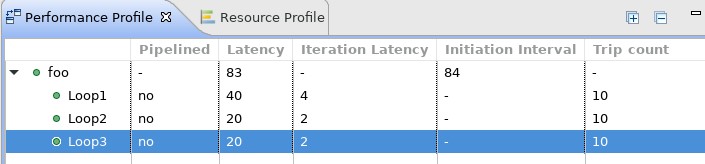


Рис. 3.7. Performance profile

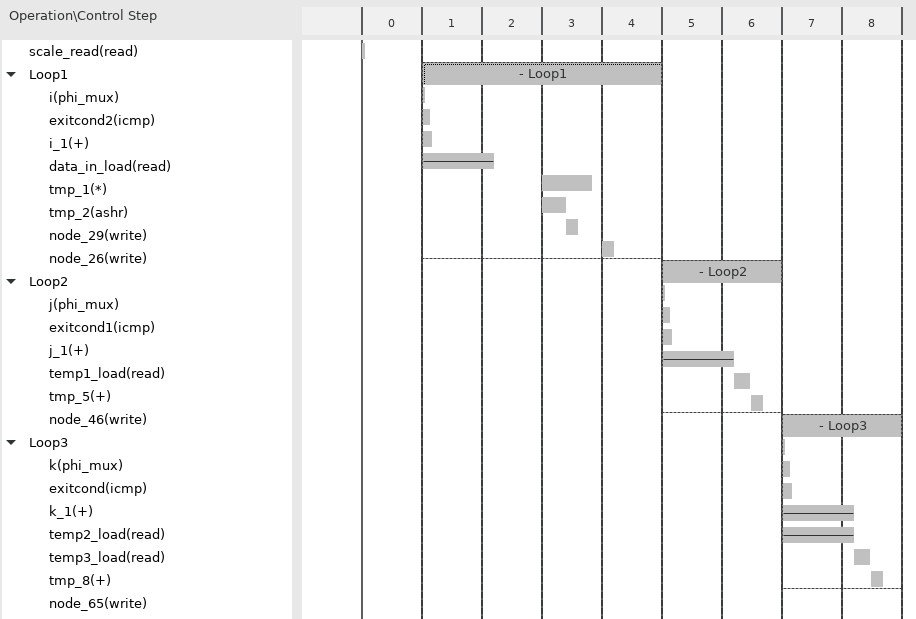


Рис. 3.8. Scheduler viewer

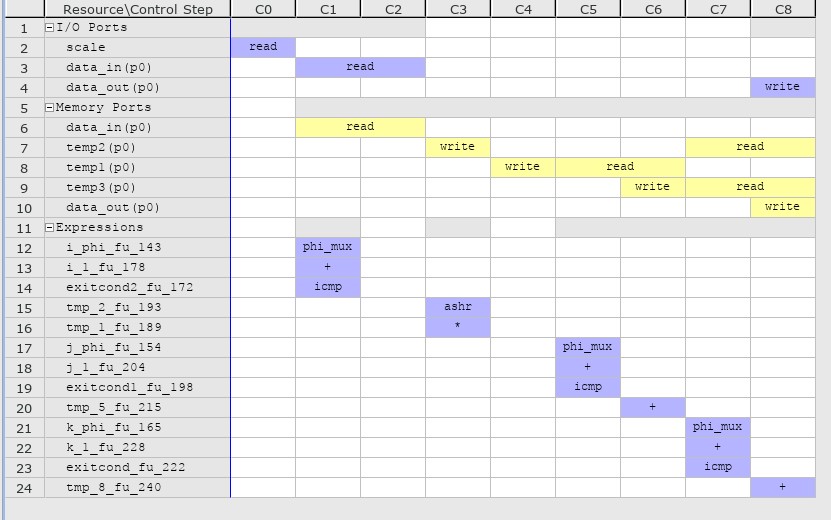


Рис. 3.9. Resource viewer

# **4. Решение №2.**

## 4.1. Исходный код программы и теста

|  |
| --- |
| #include "lab8\_2 .h"  void foo\_b ( int data\_in [N] , int scale , int data\_out [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 ] ;  }  } |

Рис. 4.1. Исходный код устройства

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

Рис. 4.2. Заголовочный файл

|  |
| --- |
| #include <stdio.h>  #include "lab8\_2.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. 4.3. Исходный код теста

## 4.2. Моделирование

Ниже приведены результаты моделирования.

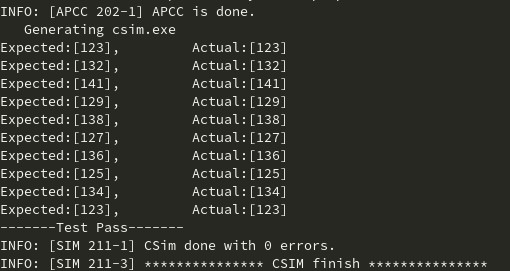


Рис. 4.4. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

## 4.3. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

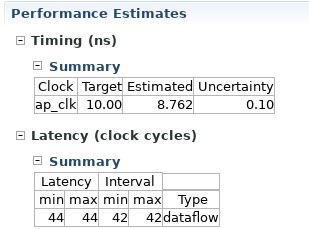


Рис. 4.5. Performance estimates

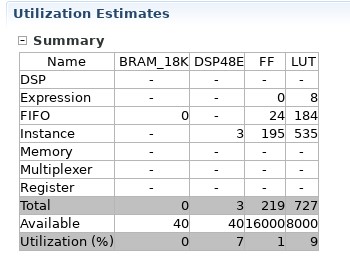


Рис. 4.6. Utilization estimates

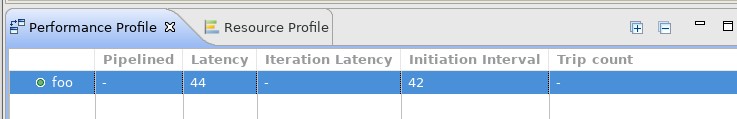


Рис. 4.7. Performance profile

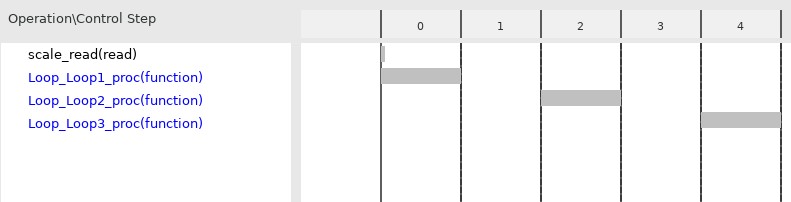


Рис. 4.8. Scheduler viewer

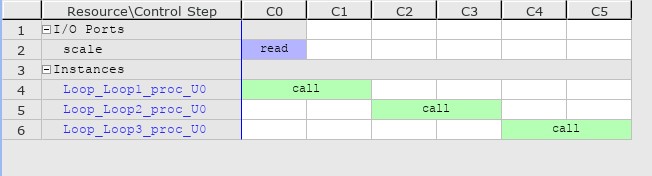


Рис. 4.9. Resource viewer

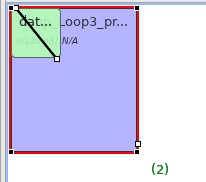


Рис. 4.10. Dataflow viewer

## 4.4. С/RTL моделирование

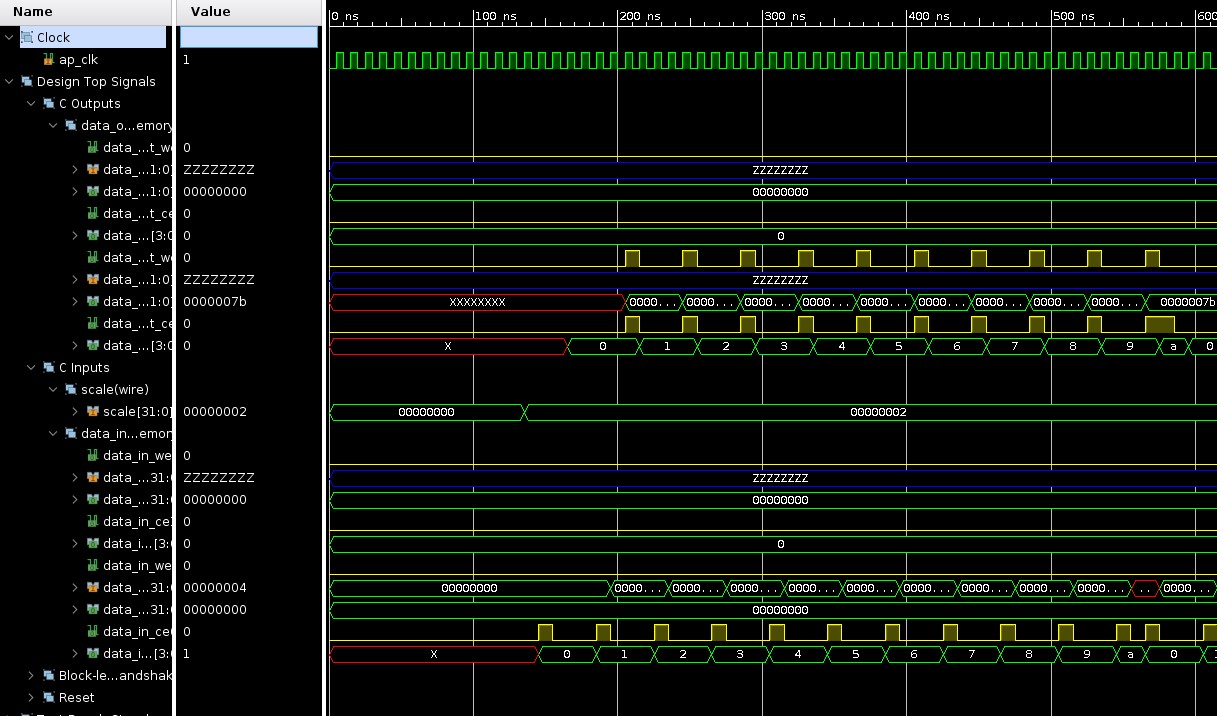


Рис. 4.11. Временная диаграмма

# **5. Решение №3.**

## 5.1. Исходный код программы и теста

|  |
| --- |
| #include "lab8\_2 .h"  void foo ( int data\_in [N] , int scale , int data\_out [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 ] ;  }  } |

Рис. 5.1. Исходный код устройства

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

Рис. 5.2. Заголовочный файл

|  |
| --- |
| #include <stdio.h>  #include "lab8\_2.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. 5.3. Исходный код теста

## 5.2. Моделирование

Ниже приведены результаты моделирования.

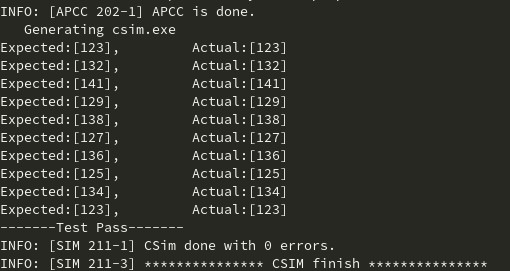


Рис. 5.4. Результаты моделирования

По результатам моделирования видно, что устройство работает корректно.

## 5.3. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

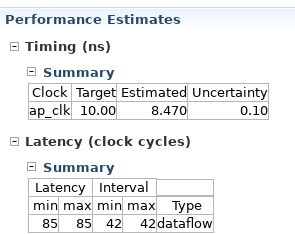


Рис. 5.5. Performance estimates

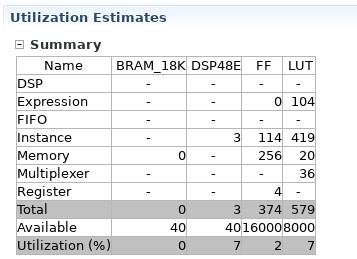


Рис. 5.6. Utilization estimates

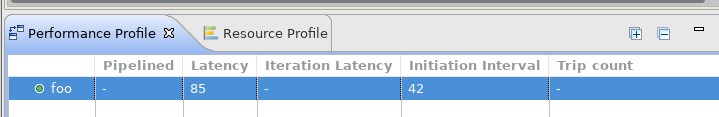


Рис. 5.7. Performance profile

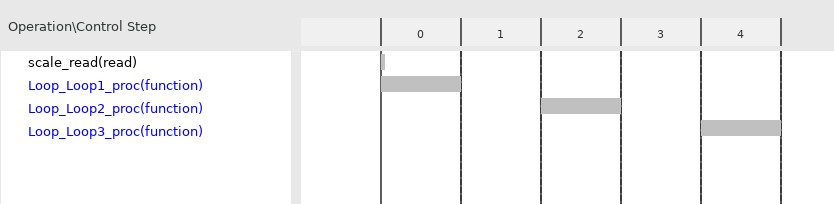


Рис. 5.8. Scheduler viewer

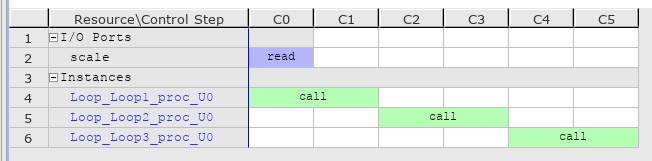


Рис. 5.9. Resource viewer

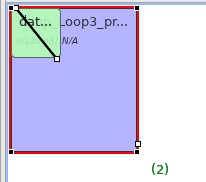


Рис. 5.10. Dataflow viewer

# **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 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.