Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Лабораторная №6

Предмет: Проектирование реконфигурируемых гибридных вычислительных систем

Тема: Сравнение протоколов Port-Level I/O

Задание 2

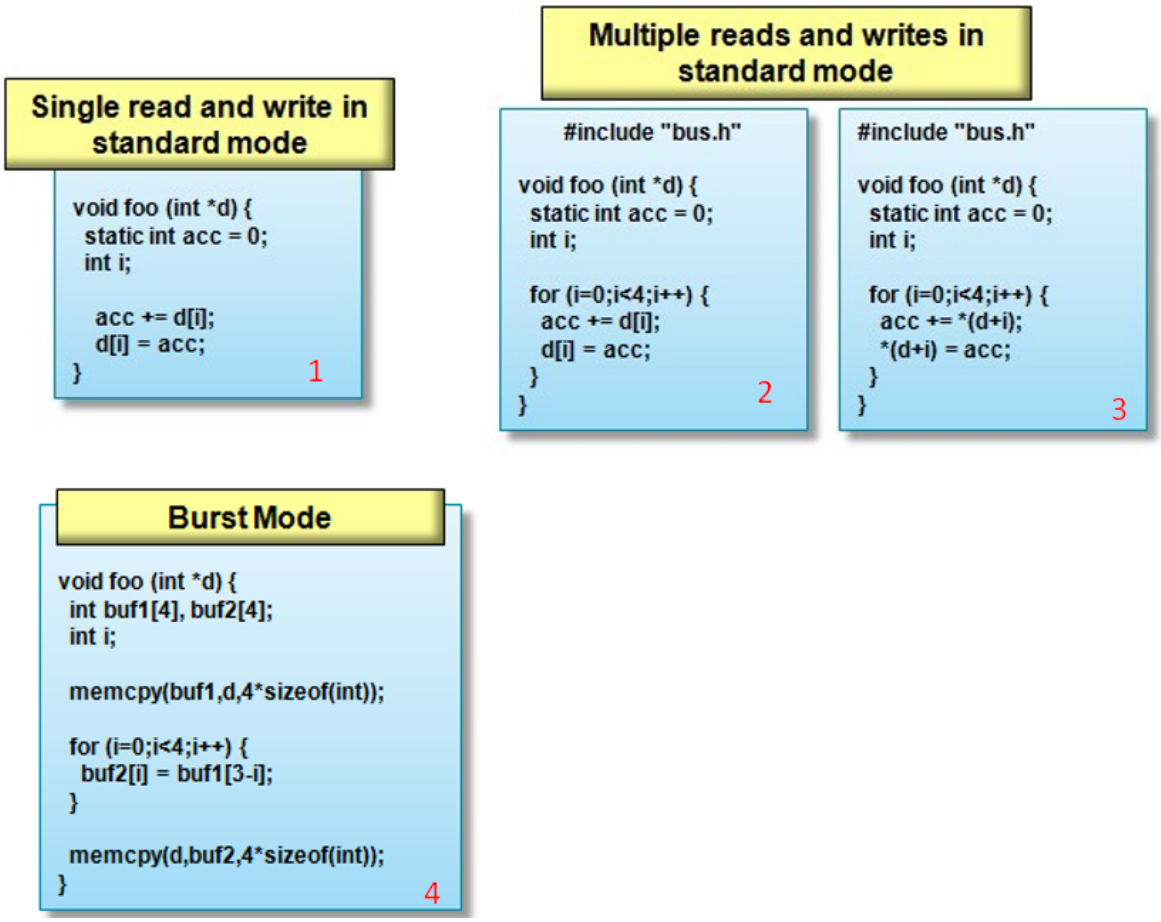
Студенты:
Соболь В.
Темнова А.С.
Группа: 13541/3
Преподаватель:
Антонов А.П.

Санкт-Петербург
2019

Содержание

1. Задание

1. Создать проект lab6_2
2. Микросхема: xa7a12tcsg325-1q
3. Создать четыре функции на основе слайда (функция foo_1, foo_2, foo_3, foo_4). *При желании можно сделать 4 отдельные лабораторные работы lab6_2_1...lab6_2_4 но они все будут очень похожи.*



4. Создать тест lab6_2_test.c для проверки функций выше (это может быть один тест или разные тесты. Функция main д.б. одна, а в ней использовать проверяемый модуль. Д.б. вывод результатов в консоль.).
5. Для каждой функции сделать свой solution
   - задать: clock period 10; clock_uncertainty 0.1
   - Задать протокол
     - a: ap_bus
   - осуществить моделирование (с выводом результатов в консоль)
   - осуществить синтез
     - привести в отчете:
       * performance estimates=>summary
       * utilization estimates=>summary
       * Performance Profile

        * interface estimates=>summary
            · объяснить какой интерфейс использован для блока (и какие сигналы входят) и для портов (и какие сигналы входят).
        * 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
    - Осуществить C|RTL моделирование
        - 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

6. Выводы

    - Объяснить отличие процедур обращения к элементам массива для каждого случая

2. Скрипт

Ниже приводится скрипт, для автоматизации выполнения лабораторной работы.

```
set tasks [list 1 2 3 4]

foreach task $tasks {
  open_project -reset lab6_2_$task
  add_files lab6_2_$task.c
  set_top foo
  add_files -tb lab6_2_test_$task.c

  open_solution solution$task -reset
  set_part {xa7a12tcsg325-1q}
  create_clock -period 10ns
  set_clock_uncertainty 0.1

  set_directive_interface -mode ap_bus foo d

  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 void foo(int *d) {
2     static int acc = 0;
3     int i;
4     acc += d[i];
5     d[i] = acc;
6 }
```

Рис. 3.1. Исходный код устройства

```
1 #include <stdio.h>
2
3 int main() {
4   int pass = 1;
5   int i;
6   int d[5];
7   int expected[5] = {5, 6, 7, 8, 9};
8
9   for (i = 0; i < 5; i++) {
10    d[i] = i + 5;
11  }
12
13  foo(d);
14
15  for (i = 0; i < 5; i++) {
16    fprintf(stdout, "%d: Expeced %d Actual %d\n", i, expected[i], d[i]);
17    if(expected[i] != d[i]) {
18      pass = 0;
19    }
20  }
21
22
23  if (pass)
24  {
25    fprintf(stdout, "----------Pass!----------\n");
26    return 0;
27  }
28  else
29  {
30    fprintf(stderr, "----------Fail!----------\n");
31    return 1;
32  }
33 }
```

Рис. 3.2. Исходный код теста

3.2. Директивы

В данном решения были установлены директивы, приведённые ниже.

Рис. 3.3. Директивы

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	9.900	0.10

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
4	4	4	4	none

Рис. 3.4. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	39
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	48
Register	-	-	101	-
Total	0	0	101	87
Available	40	40	16000	8000
Utilization (%)	0	0	~0	1

Рис. 3.5. Utilization estimates

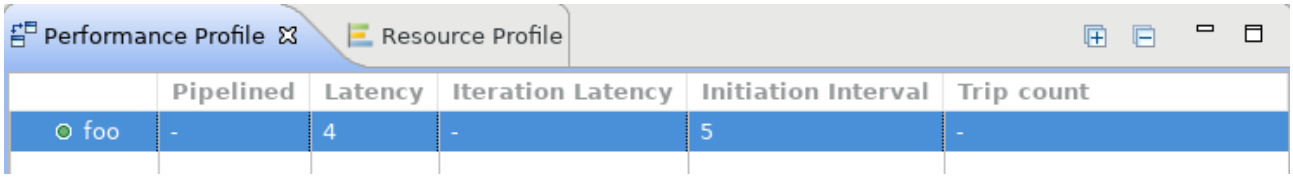


Рис. 3.6. Performance profile

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	foo	return value
ap_rst	in	1	ap_ctrl_hs	foo	return value
ap_start	in	1	ap_ctrl_hs	foo	return value
ap_done	out	1	ap_ctrl_hs	foo	return value
ap_idle	out	1	ap_ctrl_hs	foo	return value
ap_ready	out	1	ap_ctrl_hs	foo	return value
d_req_din	out	1	ap_bus	d	pointer
d_req_full_n	in	1	ap_bus	d	pointer
d_req_write	out	1	ap_bus	d	pointer
d_rsp_empty_n	in	1	ap_bus	d	pointer
d_rsp_read	out	1	ap_bus	d	pointer
d_address	out	32	ap_bus	d	pointer
d_datain	in	32	ap_bus	d	pointer
d_dataout	out	32	ap_bus	d	pointer
d_size	out	32	ap_bus	d	pointer

Рис. 3.7. Interface estimates

По списку сигналов в проекте видно, что для заданного порта установлен протокол ap_bus. Также видно, что для этого протокола требуются дополнительные сигналы.

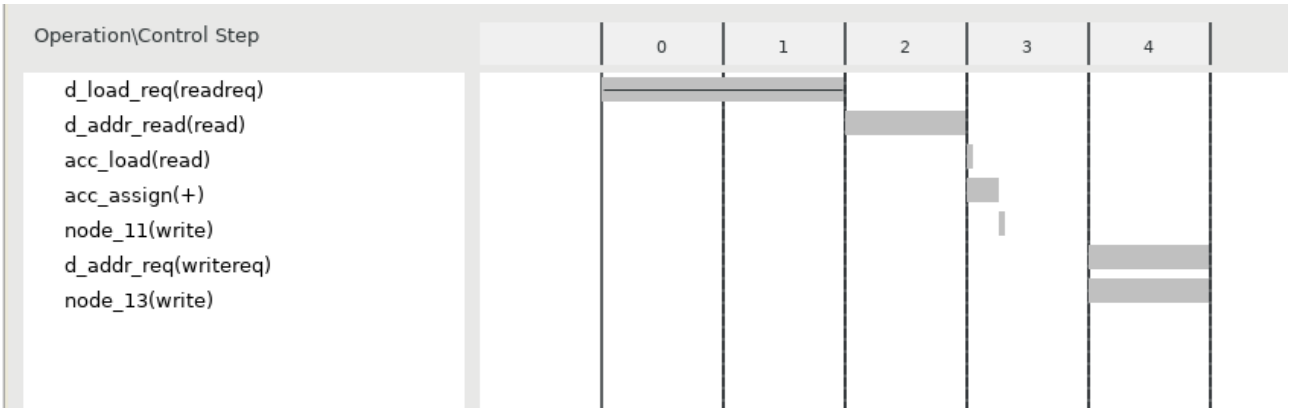
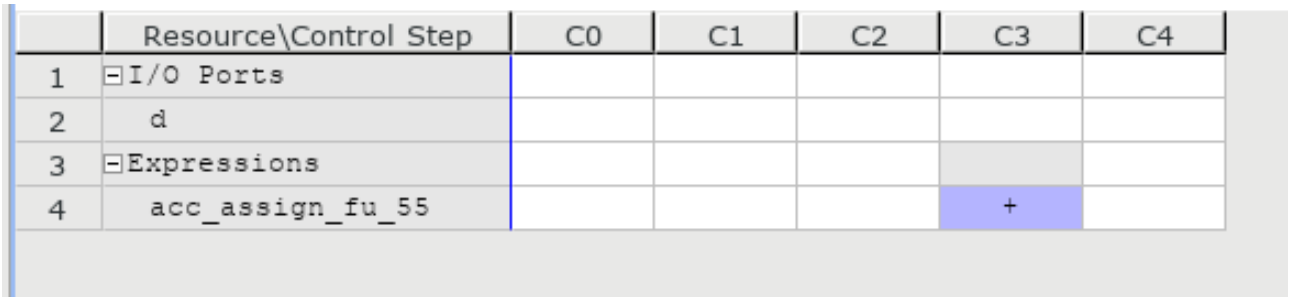


Рис. 3.8. Scheduler viewer



	Resource\Control Step	C0	C1	C2	C3	C4
1	⊟I/O Ports					
2	d					
3	⊟Expressions					
4	acc_assign_fu_55				+	

Рис. 3.9. Resource viewer

4. Решение 2

4.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
1 void foo(int *d) {
2     static int acc = 0;
3     int i;
4     for (i = 0; i < 4; i++) {
5       acc += d[i];
6       d[i] = acc;
7     }
8
9 }
```

Рис. 4.1. Исходный код устройства

```c
#include <stdio.h>

int main() {
  int pass = 1;
  int i;
  int d[5];
  int expected[5] = {5, 11, 18, 26, 9};

  for (i = 0; i < 5; i++) {
    d[i] = i + 5;
  }

  foo(d);

  for (i = 0; i < 5; i++) {
    fprintf(stdout, "%d: Expeced %d Actual %d\n", i, expected[i], d[i]);
    if(expected[i] != d[i]) {
      pass = 0;
    }
  }


  if (pass)
  {
    fprintf(stdout, "----------Pass!----------\n");
    return 0;
  }
  else
  {
    fprintf(stderr, "----------Fail!----------\n");
    return 1;
  }
}
```

Рис. 4.2. Исходный код теста

4.2. Директивы

В данном решения были установлены директивы, приведённые ниже.

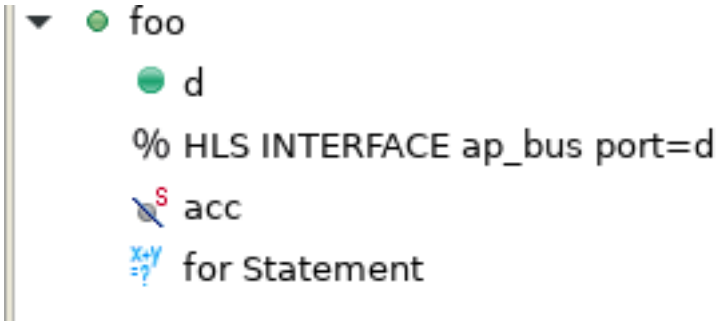


Рис. 4.3. Директивы

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	9.900	0.10

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
14	14	14	14	none

Рис. 4.4. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	77
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	42
Register	-	-	108	-
Total	0	0	108	119
Available	40	40	16000	8000
Utilization (%)	0	0	~0	1

Рис. 4.5. Utilization estimates

Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
foo	-	14	-	15	-
Loop 1	no	12	3	-	4

Рис. 4.6. Performance profile

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	foo	return value
ap_rst	in	1	ap_ctrl_hs	foo	return value
ap_start	in	1	ap_ctrl_hs	foo	return value
ap_done	out	1	ap_ctrl_hs	foo	return value
ap_idle	out	1	ap_ctrl_hs	foo	return value
ap_ready	out	1	ap_ctrl_hs	foo	return value
d_req_din	out	1	ap_bus	d	pointer
d_req_full_n	in	1	ap_bus	d	pointer
d_req_write	out	1	ap_bus	d	pointer
d_rsp_empty_n	in	1	ap_bus	d	pointer
d_rsp_read	out	1	ap_bus	d	pointer
d_address	out	32	ap_bus	d	pointer
d_datain	in	32	ap_bus	d	pointer
d_dataout	out	32	ap_bus	d	pointer
d_size	out	32	ap_bus	d	pointer

Рис. 4.7. Interface estimates

По списку сигналов в проекте видно, что для заданного порта установлен протокол ap_bus. Также видно, что для этого протокола требуются дополнительные сигналы.

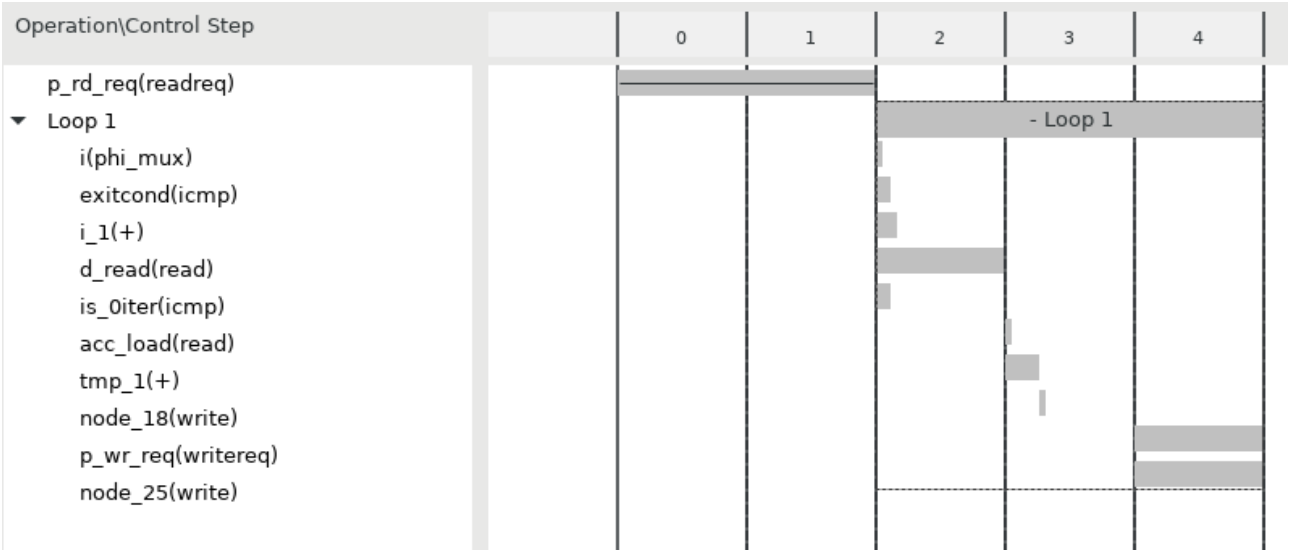


Рис. 4.8. Scheduler viewer

	Resource\Control Step	C0	C1	C2	C3	C4
1	⊟I/O Ports					
2	d					writereq
3	⊟Expressions					
4	i_1_fu_71			+		
5	i_phi_fu_58			phi_mux		
6	is_0iter_fu_77			icmp		
7	exitcond_fu_65			icmp		
8	tmp_1_fu_87				+	

Рис. 4.9. Resource viewer

5. Решение 3

5.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
1 void foo(int *d) {
2     static int acc = 0;
3     int i;
4     for (i = 0; i < 4; i++) {
5       acc += *(d + i);
6       *(d + i) = acc;
7     }
8
9 }
```

Рис. 5.1. Исходный код устройства

```c
#include <stdio.h>

int main() {
  int pass = 1;
  int i;
  int d[5];
  int expected[5] = {5, 11, 18, 26, 9};

  for (i = 0; i < 5; i++) {
    d[i] = i + 5;
  }

  foo(d);

  for (i = 0; i < 5; i++) {
    fprintf(stdout, "%d: Expeced %d Actual %d\n", i, expected[i], d[i]);
    if(expected[i] != d[i]) {
      pass = 0;
    }
  }


  if (pass)
  {
    fprintf(stdout, "----------Pass!----------\n");
    return 0;
  }
  else
  {
    fprintf(stderr, "----------Fail!----------\n");
    return 1;
  }
}
```

Рис. 5.2. Исходный код теста

5.2. Директивы

В данном решения были установлены директивы, приведённые ниже.

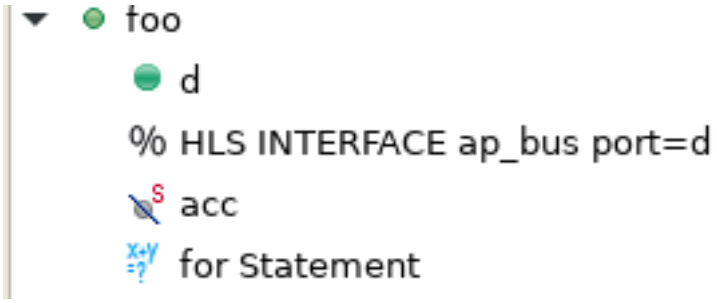


Рис. 5.3. Директивы

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	9.900	0.10

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
14	14	14	14	none

Рис. 5.4. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	77
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	42
Register	-	-	108	-
Total	0	0	108	119
Available	40	40	16000	8000
Utilization (%)	0	0	~0	1

Detail

Рис. 5.5. Utilization estimates

Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
foo	-	14	-	15	-
Loop 1	no	12	3	-	4

Рис. 5.6. Performance profile

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	foo	return value
ap_rst	in	1	ap_ctrl_hs	foo	return value
ap_start	in	1	ap_ctrl_hs	foo	return value
ap_done	out	1	ap_ctrl_hs	foo	return value
ap_idle	out	1	ap_ctrl_hs	foo	return value
ap_ready	out	1	ap_ctrl_hs	foo	return value
d_req_din	out	1	ap_bus	d	pointer
d_req_full_n	in	1	ap_bus	d	pointer
d_req_write	out	1	ap_bus	d	pointer
d_rsp_empty_n	in	1	ap_bus	d	pointer
d_rsp_read	out	1	ap_bus	d	pointer
d_address	out	32	ap_bus	d	pointer
d_datain	in	32	ap_bus	d	pointer
d_dataout	out	32	ap_bus	d	pointer
d_size	out	32	ap_bus	d	pointer

Рис. 5.7. Interface estimates

По списку сигналов в проекте видно, что для заданного порта установлен протокол ap_bus. Также видно, что для этого протокола требуются дополнительные сигналы.

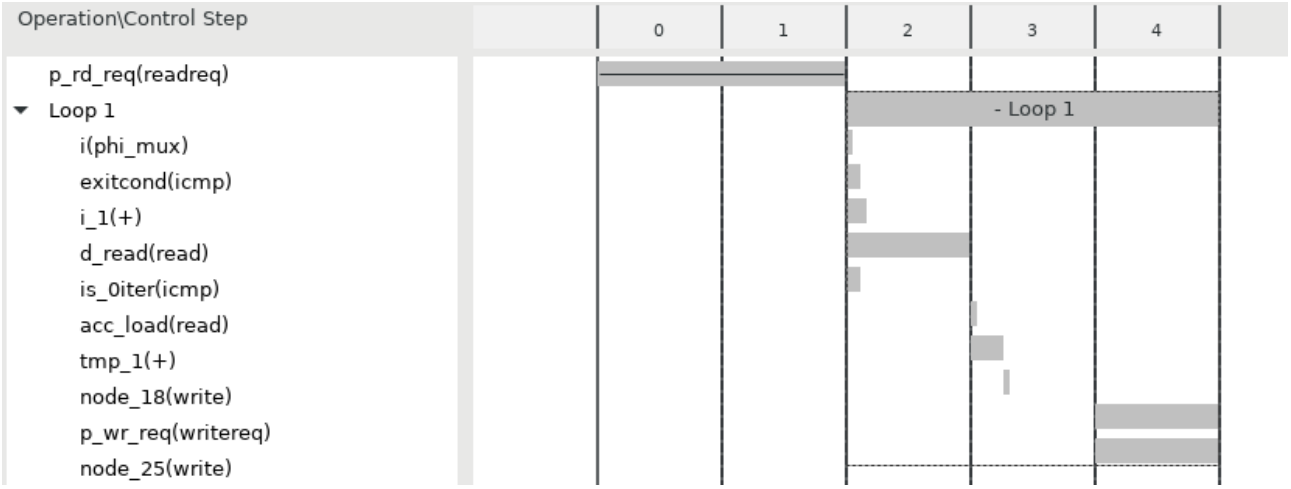


Рис. 5.8. Scheduler viewer

	Resource\Control Step	C0	C1	C2	C3	C4
1	⊟I/O Ports					
2	d					writereq
3	⊟Expressions					
4	i_1_fu_71			+		
5	i_phi_fu_58			phi_mux		
6	is_0iter_fu_77			icmp		
7	exitcond_fu_65			icmp		
8	tmp_1_fu_87				+	

Рис. 5.9. Resource viewer

6. Решение 4

6.1. Исходный код

Ниже приведен исходный код устройства и теста.

```
1  void foo(int *d) {
2      int buf1[4], buf2[4];
3      int i;
4
5      memcpy(buf1, d, 4*sizeof(int));
6
7      for (i = 0; i < 4; i++) {
8        buf2[i] = buf1[3 - i];
9      }
10
11    memcpy(d, buf2, 4*sizeof(int));
12 }
```

Рис. 6.1. Исходный код устройства

```c
#include <stdio.h>

int main() {
  int pass = 1;
  int i;
  int d[5];
  int expected[5] = {8, 7, 6, 5, 9};

  for (i = 0; i < 5; i++) {
    d[i] = i + 5;
  }

  foo(d);

  for (i = 0; i < 5; i++) {
    fprintf(stdout, "%d: Expeced %d Actual %d\n", i, expected[i], d[i]);
    if(expected[i] != d[i]) {
      pass = 0;
    }
  }


  if (pass)
  {
    fprintf(stdout, "----------Pass!----------\n");
    return 0;
  }
  else
  {
    fprintf(stderr, "----------Fail!----------\n");
    return 1;
  }
}
```

Рис. 6.2. Исходный код теста

6.2. Директивы

В данном решения были установлены директивы, приведённые ниже.

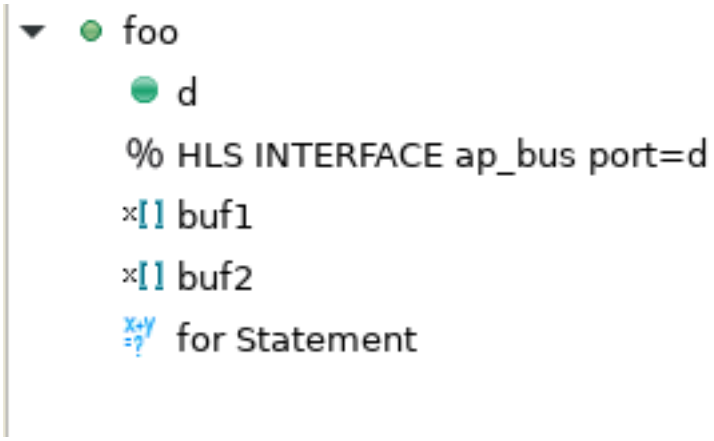


Рис. 6.3. Директивы

6.3. Синтез

По оценке производительности видно, что устройство соответствует заданным критериям.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.900	0.10

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
18	18	18	18	none

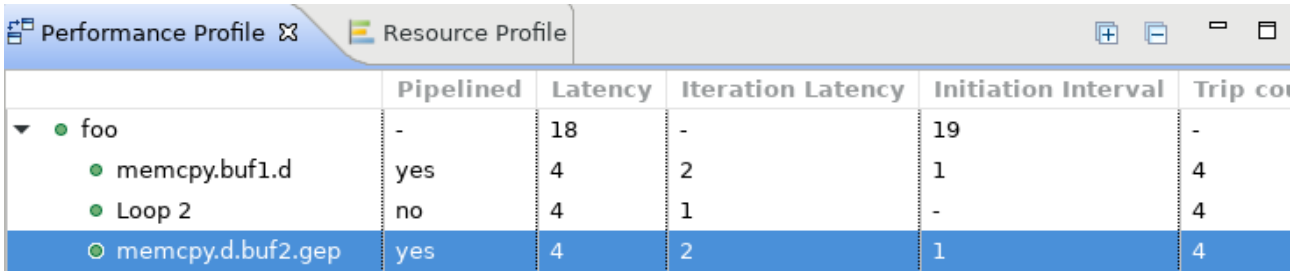
Рис. 6.4. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	136
FIFO	-	-	-	-
Instance	-	-	0	42
Memory	-	-	-	-
Multiplexer	-	-	-	98
Register	-	-	312	-
Total	0	0	312	276
Available	40	40	16000	8000
Utilization (%)	0	0	1	3

Рис. 6.5. Utilization estimates



	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip co
foo	-	18	-	19	-
memcpy.buf1.d	yes	4	2	1	4
Loop 2	no	4	1	-	4
memcpy.d.buf2.gep	yes	4	2	1	4

Рис. 6.6. Performance profile

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	foo	return value
ap_rst	in	1	ap_ctrl_hs	foo	return value
ap_start	in	1	ap_ctrl_hs	foo	return value
ap_done	out	1	ap_ctrl_hs	foo	return value
ap_idle	out	1	ap_ctrl_hs	foo	return value
ap_ready	out	1	ap_ctrl_hs	foo	return value
d_req_din	out	1	ap_bus	d	pointer
d_req_full_n	in	1	ap_bus	d	pointer
d_req_write	out	1	ap_bus	d	pointer
d_rsp_empty_n	in	1	ap_bus	d	pointer
d_rsp_read	out	1	ap_bus	d	pointer
d_address	out	32	ap_bus	d	pointer
d_datain	in	32	ap_bus	d	pointer
d_dataout	out	32	ap_bus	d	pointer
d_size	out	32	ap_bus	d	pointer

Рис. 6.7. Interface estimates

По списку сигналов в проекте видно, что для заданного порта установлен протокол ap_bus. Также видно, что для этого протокола требуются дополнительные сигналы.

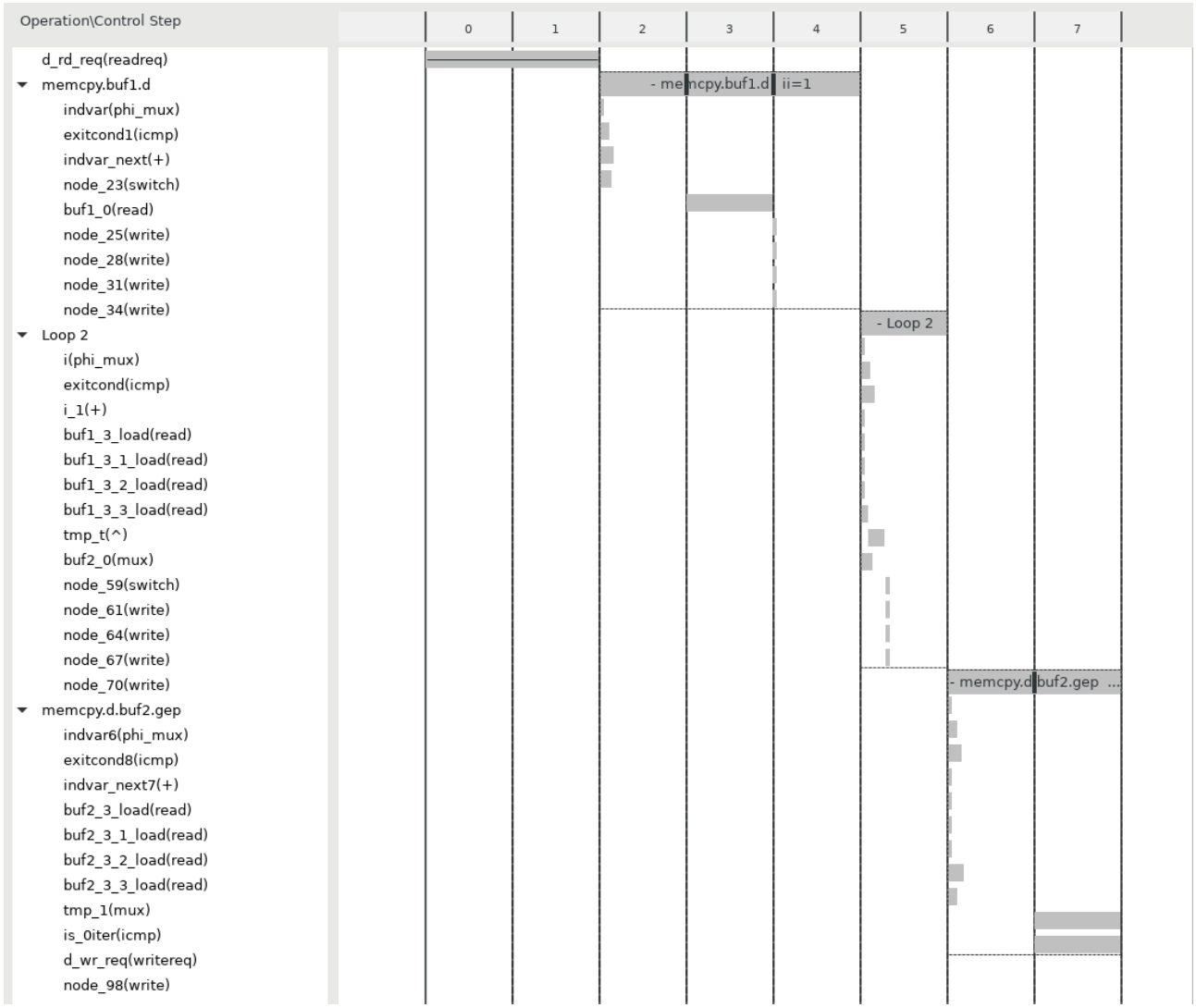


Рис. 6.8. Scheduler viewer

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8
1	⊟I/O Ports									
2	d								writereq	
3	⊟Expressions									
4	indvar_phi_fu_120			phi_mux						
5	indvar_next_fu_155			+						
6	exitcond1_fu_149			icmp						
7	i_phi_fu_131						phi_mux			
8	i_1_fu_191						+			
9	buf2_0_fu_219						mux			
10	tmp_t_fu_213						^			
11	exitcond_fu_185						icmp			
12	indvar6_phi_fu_142							phi_mux		
13	indvar_next7_fu_259							+		
14	tmp_1_fu_281							mux		
15	is_0iter_fu_295							icmp		
16	exitcond8_fu_253							icmp		

Рис. 6.9. Resource viewer

7. Вывод

В данной лабораторной работе были рассмотрены рассмотрены различные процедуры обращения к элементам массива.

В первом решении, используется доступ к одному элементу массива.

Во втором решении, используется последовательный доступ к элементам массива.

Третье решение аналогично второму, так как там используется такой же метод обращения к элементам массива, но записан он иной синтаксической конструкцией.

В последнем решении, основной особенностью является использование Burst mode с использованием функции memcpu. Данная команда после синтеза предоставляет возможность сразу считывать весь размер массива и записывать данные параллельно используя промежуточные буферы. Количество ресурсов увеличивается в сравнении остальными вариантами обращения к памяти, однако увеличивает пропускную способность блока.