Глава 9. Работа с BRAM

Интерфейс BRAM, однопортовый и двухпортовый режимы работы.

Введение

Внутри ПЛИС Xilinx есть специальный аппаратный ресурс – блочная память или BRAM (Block RAM). Это набор из специальных аппаратных блоков статической памяти, выполненных аппаратно, т.е. часть логики кристалла зафиксирована под память, и не может быть использована по другому назначению. Количество таких блоков зависит от конкретного типа кристалла ПЛИС и может быть до нескольких сотен. Каждый блок имеет два аппаратных порта, которые могут работать на разных тактовых частотах. У блочной памяти есть свои преимущества и недостатки. С одной стороны, это быстрые аппаратные блоки, способные работать на частотах до 500 МГц. При использовании блочной памяти ее разные порты могут работать на разных частотах. Это удобно использовать для синхронизации между разными клоковыми доменами. Но с другой стороны, блочная память – дорогостоящий ресурс. Количество ее ограничено. Если требуется объем памяти меньший чем объем одного блока (18 Кбит), то все равно будет использован весть блок, и неиспользованная его часть просто пропадет. Для создания оперативных и постоянных запоминающих устройств, реализуемых на основе блочной памяти ПЛИС, в составе генератора параметризованных модулей LogiCORE предусмотрено такое ядро, как Block Memory Generator. В данной главе будет подробно рассмотрено его использование, настройка и реализация с помощью средств языка VHDL.

Создание и настройка ядра BRAM

Заходим в Block Memory Generator

Выберите Block Memory Generator из каталога IP (рис. 1-2).

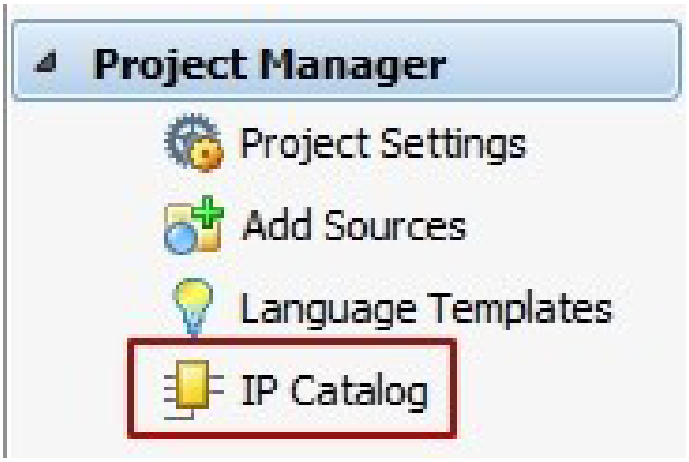

Рис. 1: Рис.1

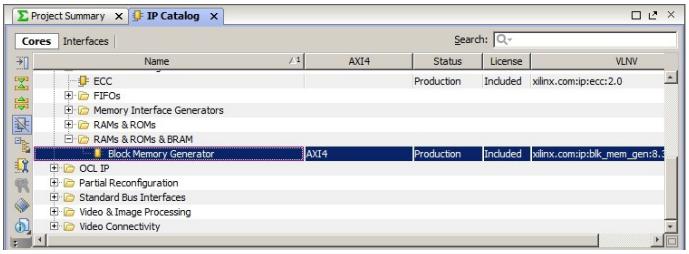

Рис. 2: Рис.2

Работа с интерфейсом BMG

Выбор типа и организации запоминающего устройства, формируемого на основе рассматриваемого параметризированного модуля, осуществляется с помощью соответствующего *мастера* настройки параметров, который содержит от четырёх до шести (в зависимости от конфигурируемых параметров) диалоговых панелей.

Basic

В стартовой диалоговой панели *мастера*, вид которой показан на рисунке, указывается тип создаваемого элемента памяти, его название, а также алгоритм его реализации.

- Название формируемого вида запоминающего устройства вводится с помощью клавиатуры в поле редактирования Component Name.

На выбор доступны два типа интерфейса – Native и AXI4.

Память блоков интерфейса AXI4 построена на памяти блоков нативного (Native) типа. Доступны два стиля интерфейса AXI4: AXI4 и AXI4-Lite. Также на выбор конфигурируется ядро. Далее на выбор предполагается настройка устройства памяти как ведомое или периферийно. В дополнение к приложениям, поддерживаемым нативным интерфейсом, AXI4 может также использоваться в приложениях AXI4 System Bus и приложениях типа Point-to-Point.

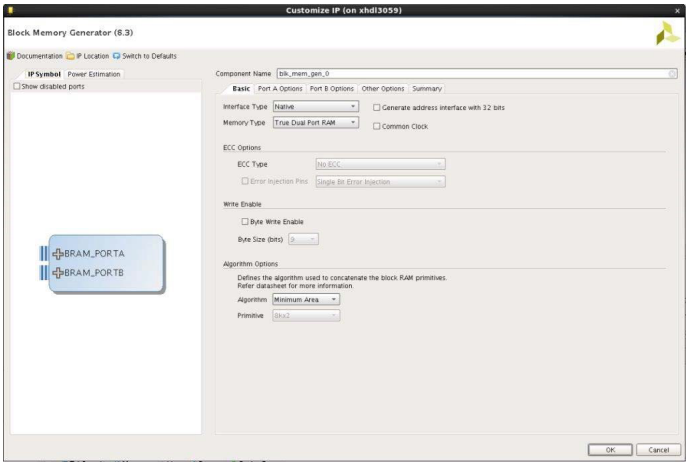

Рис. 3: Рис.3

Для нативного типа интерфейса предполагаются следующие параметры настройки:

- Для определения типа генерируемого элемента памяти следует воспользоваться группой кнопок с зависимой фиксацией Memory Type. Если в нажатом состоянии зафиксирована кнопка Single Port RAM, то будет сформировано ОЗУ с одним портом записи и одним портом чтения данных. При нажатой кнопке Simple Dual Port RAM создается элемент двухпортовой оперативной памяти с одним портом чтения данных.Кроме того, при выборе данного типа памяти становится доступной опция ECC (Error Correction Checking) Type, созданная для регистрации и исправления ошибок с помощью кода Хэмминга и поиска необходимых совместимостей. Для генерации ОЗУ с двумя портами чтения и двумя портами записи данных нужно переключить в нажатое состояние кнопку True Dual Port RAM. Формирование однопортового постоянного запоминающего устройства осуществляется при нажатой кнопке Single Port ROM. Чтобы создать элемент двухпортовой постоянной памяти, следует перевести во включенное состояние кнопку Dual Port ROM. Для двухпортовых элементов доступна опция Common Clock для синхронизации часов ввода (управление одним буфером)

- Также может быть установлен режим побайтной записи. Для этого нужно перевести в состояние *включено* индикатор Use Byte Write Enable, который расположен во встроенной панели Write Enable.

- Побайтная запись может осуществляться с контролем и без контроля четности. При этом размер записываемого байта (количество бит в байте) зависит от использования контроля четности. В случае отсутствия контроля четности длина байта составляет восемь бит. Чтобы побайтная запись выполнялась с контролем четности, в состав байта данных должно входить девять бит (восемь бит данных и один контрольный).

- Размер байта (и, соответственно, использование контроля четности) указывается с помощью поля выбора Byte Size.

- Запоминающее устройство, генерируемое с помощью параметризированного модуля Block Memory Generator, строится в виде совокупности примитивов блочной памяти Block RAM primitive. В общем случае каждый из используемых примитивов может иметь различную организацию. Количество и состав возможных вариантов организации примитивов блочной памяти Block RAM primitive определяется выбранным семейством ПЛИС. Тип организации примитивов блочной памяти и алгоритм их соединения для получения запоминающего устройства с требуемой емкостью и организацией указывается с помощью двух кнопок с зависимой фиксацией, которые расположены во встроенной панели Algorithm. Когда в нажатом состоянии находится кнопка Minimum Area, создаваемый элемент памяти составляется из примитивов с различной организацией с целью минимизации используемого количества модулей блочной памяти Block RAM кристалла ПЛИС. При нажатой кнопке Low Power создаваемый элемент памяти будет состоять из тех же, модулей, что и в Minimum Area ядре, но включёнными только при записи или чтении. При нажатой кнопке Fixed Primitive для построения требуемого запоминающего устройства будут использоваться примитивы блочной памяти одного типа, организация которого указывается пользователем с помощью поля выбора Primitive. Содержимое выпадающего списка этого поля выбора зависит от семейства ПЛИС, для которого создается элемент памяти.

Port A|B options

Вторая диалоговая панель *мастера* настройки параметров генератора оперативных и постоянных запоминающих устройств, реализуемых на основе блочной памяти ПЛИС, предназначена для определения емкости формируемого элемента памяти, а также описания организации и режима работы первого (или единственного) порта запоминающего устройства.

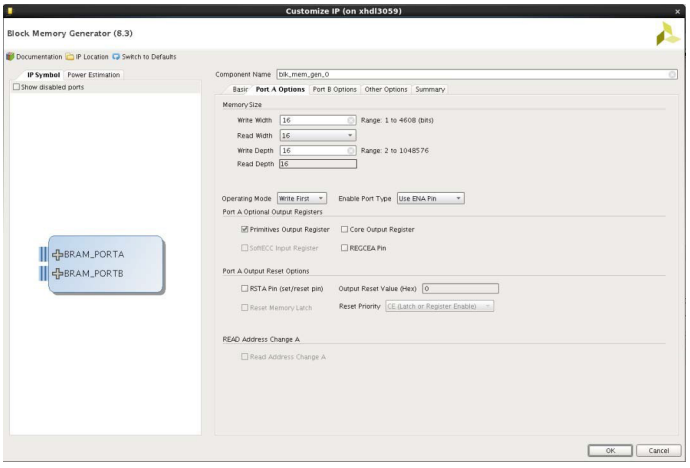

Рис. 4: Рис.4

- Чтобы указать информационную емкость формируемого элемента памяти и разрядность первого (или единственного) порта записи и чтения данных, нужно воспользоваться полями редактирования и выбора, расположенными во встроенной панели Memory Size. Прежде всего, рекомендуется определить разрядность первого входного порта Port A, предназначенного для записи данных, с помощью поля редактирования Write Width. Требуемое число разрядов этого порта (в диапазоне от 1 до 1152) указывается с помощью клавиатуры, после активизации данного поля редактирования. Количество ячеек памяти в формируемом запоминающем устройстве задается в поле редактирования Write Depth. Максимальное значение этого параметра ограничено объемом физических ресурсов блочной памяти Block RAM используемого кристалла ПЛИС. Таким образом, информационная емкость создаваемого элемента памяти, выраженная в битах, равна произведению значений параметров Write Width и Write Depth. Разрядность первого (или единственного) выходного порта, используемого для чтения данных из памяти, определяется с помощью поля выбора Read Width. Выпадающий список этого поля выбора содержит допустимые варианты разрядности первого выходного порта, которые соответствуют установленному значению разрядности первого входного порта записи данных. Количество разрядов адреса (адресных входов) в формируемом запоминающем устройстве вычисляется автоматически, исходя из установленных значений параметров, рассмотренных выше. Значения параметров Write Width и Read Width должны быть кратны размеру байта данных, указанному в поле выбора Byte Size. В этом случае вместо обычного (одиночного) входа разрешения записи данных используется шина, количество разрядов которой вычисляется автоматически в соответствии с указанным значением разрядности порта записи.

- Для определения режима работы первого (или единственного) выходного порта элемента оперативной памяти, предназначенного для чтения данных, при выполнении операции записи данных в ОЗУ нужно воспользоваться группой кнопок с зависимой фиксацией Operating Mode. Если в нажатом состоянии находится кнопка Write First, то данные, поступающие во входной порт, записываются в соответствующую ячейку памяти (адрес которой задается комбинацией сигналов на адресных входах), после чего сразу передаются на выходы запоминающего устройства. При нажатии кнопки Read First в генерируемом элементе оперативной памяти будет установлен режим предварительного чтения данных из указанной ячейки перед записью новых данных в эту ячейку. Таким образом, при осуществлении операции записи данных, поступающих во входной порт формируемого элемента ОЗУ, на его выходах будет отображаться информация, которая содержалась в соответствующей ячейке перед этим (на предыдущем такте). Когда в нажатое состояние переводится кнопка No Change, в формируемом элементе ОЗУ будет установлен режим блокировки выходов при выполнении операции записи данных. В этом режиме на протяжении всего цикла записи выходы находятся в зафиксированном

состоянии, которое соответствует последним считанным данным, присутствовавшим в момент переключения сигнала разрешения записи в активное состояние.

- Чтобы сформировать элемент запоминающего устройства с входом разрешения операций для первого порта, нужно переключить в нажатое состояние кнопку Use ENA Pin. При этом в созданном элементе памяти выполнение операций записи и чтения данных для первого порта будет возможно только при активном уровне сигнала на входе разрешения ENA.

- Для определения состояния выходного регистра или защелки формируемого запоминающего устройства в режиме сброса (или установки) следует воспользоваться полем редактирования Output Reset Value (Hex), которое расположено во встроенной панели Output Reset. Значение, указываемое в этом поле редактирования, должно быть представлено в шестнадцатеричном формате. При этом количество шестнадцатеричных символов должно соответствовать разрядности первого порта чтения данных. Чтобы задействовать в формируемом элементе памяти вход синхронного сброса (или установки) для первого порта (Port A), следует установить индикатор Use SSRA Pin, находящийся в этой же встроенной панели, в состояние *включено*.

- Если в стартовой диалоговой панели *мастера* настройки параметров генератора элементов памяти был выбран двухпортовый тип запоминающего устройства, то третья диалоговая панель будет позволять определить основные параметры второго порта (Port B) создаваемого элемента памяти. Все параметры этого порта, за исключением разрядности, указываются так же, как и для первого порта (Port A). Количество разрядов второго порта записи данных не может быть установлено произвольно. Значение этого параметра должно быть кратным числу разрядов первого порта записи данных (с фиксированным набором коэффициентов кратности), которое было указано в диалоговой панели, представленной на рисунке. Поэтому разрядность второго порта записи данных задается с помощью поля выбора Write Width, выпадающий список которого содержит только допустимые значения данного параметра. Содержимое этого списка автоматически корректируется при изменении значения разрядности первого порта записи данных.

Other options

Предпоследняя диалоговая панель *мастера* настройки параметров генератора оперативных и постоянных запоминающих устройств, реализуемых на основе блочной памяти ПЛИС, позволяет выбрать тип и конфигурацию выходных регистров, а также указать параметры инициализации формируемого элемента памяти.

- Стадии конвейера в пределах Мультиплексора: доступна только, когда опция Register Output of Memory Core выбрана и для порта A и для порта

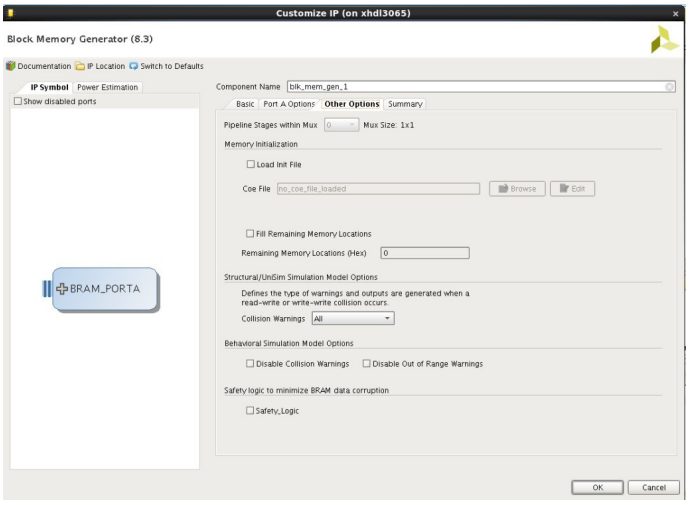

Рис. 5: Рис.5

В и когда у созданной памяти есть больше чем один примитив (так, чтобы MUX был необходим при выводе). Выберите значение 0, 1, 2, или 3 из выпадающего списка.

- Для автоматической инициализации содержимого генерируемых запоминающих устройств, реализуемых на основе блочной памяти ПЛИС, следует определить значения соответствующих параметров с помощью элементов управления, которые расположены во встроенной панели Memory Initialization. Информацию, которую нужно записать в соответствующие ячейки формируемого элемента памяти, можно указать в виде файла формата COE. Для этого нужно, прежде всего, переключить индикатор Load Init File в состояние «включено», после чего станут доступными поле редактирования COE File и кнопка Browse. При нажатии на данную кнопку на экран выводится стандартная панель диалога открытия файла, с помощью которой нужно найти на одном из дисков компьютера требуемый файл, описывающий содержимое создаваемого запоминающего устройства. После выбора соответствующего файла и закрытия стандартной диалоговой панели его название автоматически отображается в поле редактирования COE File. Можно также с помощью клавиатуры сразу указать в этом поле редактирования название требуемого файла инициализации, не выполняя процедуру его поиска. Для быстрого просмотра содержимого выбранного файла нужно воспользоваться кнопкой Show, которая находится во встроенной панели Memory Initialization. Если указываемый файл инициализации описывает содержимое только части генерируемого запоминающего устройства, то все оставшиеся неинициализированными ячейки памяти могут быть заполнены по умолчанию значением, определяемым пользователем. С этой целью нужно установить индикатор Fill Remaining Memory Locations в состояние *включено*. При этом становится доступным поле редактирования Remaining Memory Locations (Hex), в котором нужно с помощью клавиатуры указать шестнадцатеричное значение, записываемое по умолчанию в ячейки памяти, оставшиеся неопреде-

ленными. Количество шестнадцатеричных символов, указываемых в этом поле редактирования, должно соответствовать разрядности первого порта записи генерируемого запоминающего устройства.

- Structural/UNISIM Simulation Model Options: Выберите тип предупреждающих сообщений и выводов, сгенерированных структурной моделью моделирования в случае коллизий. Для опций ALL, WARNING ONLY и GENERATE X ONLY, обнаружения коллизий, опция активирована в моделях UNISIM, чтобы обработать коллизию при любом условии.

- Behavioral Simulation Model Options: Выберите тип предупреждающих сообщений, сгенерированных моделью моделирования на поведенческом уровне. Выберите, должна ли модель принять синхронные часы (Общие Часы) для предупреждений коллизии.

- Dynamic Power Saving: экономия электроэнергии включена в положении, когда память активно не используется в течение длительного периода времени. Если матрица элементов памяти входит в режим ожидания, данные сохраняются. Чтобы использовать память, установите контакт сна в 0.

Summary

Заключительная диалоговая панель *мастера* настройки генератора оперативных и постоянных запоминающих устройств, реализуемых на основе блочной памяти ПЛИС, отражает параметры создаваемого устройства в виде списка.

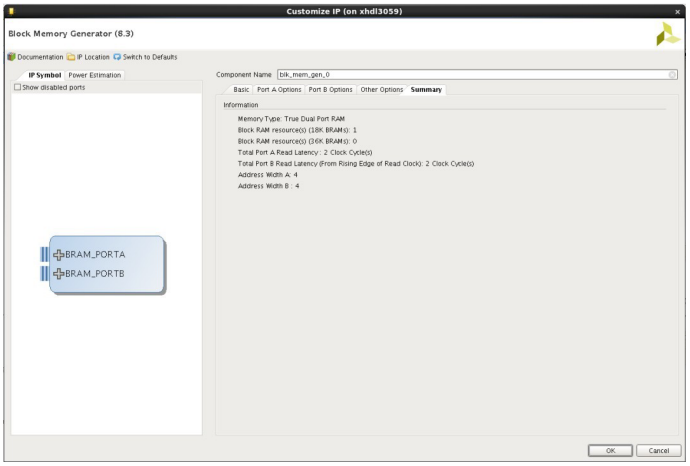

Рис. 6: Рис.6

- Memory Type: Сообщает о выбранном типе памяти.

- Block RAM Resources: Сообщает точный номер 18 K и 36 K блочных примитивов RAM, которые используются, чтобы создать ядро.

- Total Port A Read Latency: номер тактов для Операции чтения для порта A. Этим значением управляют дополнительные выходные опции регистров для порта на предыдущей вкладке.

- Total Port B Read Latency: номер тактов для Операции чтения для порта B. Этим значением управляют дополнительные выходные опции регистров для порта B на предыдущей вкладке.

- Address Width: фактическая ширина адресной шины к каждому порту.

Power Estimation

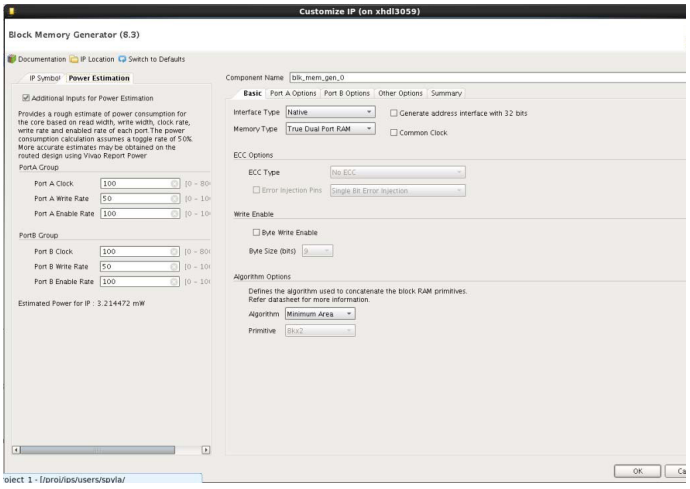

Рис. 7: Рис.7

Вкладка Power Estimation на левой стороне IDE Vivado, показанная на рисунке, обеспечивает грубую оценку потребляемой мощности для ядра, основанного на сконфигурированных параметрах Read width, Write width, clock rate, Write rate and enable rate для каждого порта.

У этой вкладки есть опция *Additional Inputs for Power Estimation*, созданная для ввода дополнительных оценок потребляемой мощности. Можно ввести следующие параметры для расчета питания: Clock Frequency [A|B] (тактовая частота портов A и B), Write Rate [A|B] (скорость записи для портов A и B), Enable Rate [A|B] (средняя скорость доступа к портам A и B)

Включение созданного ядра в код VHDL

После выбора необходимых параметров для модуля BRAM следует нажать OK, а затем Generate во всплывающем окне. После того, как завершится синтез нашего модуля, переходим на вкладку IP Sources и открываем Файл в формате .vho в папке Instantiation Template(см. Рис. 8)

В этом файле уже созданы вставки для определения компонента в основном коде и инстанциация для тестбенча (рис. 9-10)

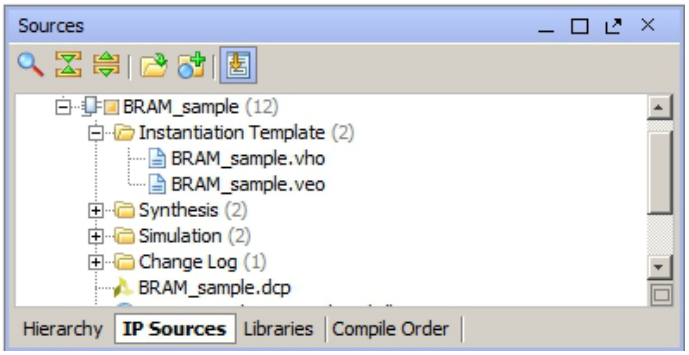

Рис. 8: Рис. 8

```
54 ------------ Begin Cut here for COMPONENT Declaration ------ COMP_TAG
55 COMPONENT BRAM_sample
56   PORT (
57     clka : IN STD_LOGIC;
58     ena : IN STD_LOGIC;
59     wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
60     addra : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
61     dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
62     douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
63   );
64 END COMPONENT;
65 -- COMP_TAG_END ------ End COMPONENT Declaration ------------
```

Рис. 9: Рис. 9

Симуляция

В качестве примера рассмотрим BRAM со следующими параметрами (всё, что не указано – по умолчанию):

1. Algorithm: Low Power
2. Write width:8, Read width:8, Write depth:16
3. Галочка Fill Remaining memory locations

Напишем тестбенч для данного модуля памяти:

Листинг 1: Тестбенч

```
  library IEEE;
1 use ieee.std_logic_1164.all;
2 use ieee.std_logic_arith.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity tb is
6 end tb;
7
8 architecture arch of tb is
9
10 --Объявление временных сигналов
11 signal ena : std_logic := '0';
12 signal wea : std_logic_VECTOR(0 downto 0):="0";
```

```
70 ------------ Begin Cut here for INSTANTIATION Template ----- INST_TAG
71 your_instance_name : BRAM_sample
72   PORT MAP (
73     clka => clka,
74     ena => ena,
75     wea => wea,
76     addra => addra,
77     dina => dina,
78     douta => douta
79   );
80 -- INST_TAG_END ------ End INSTANTIATION Template ---------
```

Рис. 10: Рис. 10

```
13 signal addra : std_logic_vector (3 downto 0) := (others=>
       '0');
14 signal dina,douta : std_logic_VECTOR(7 downto 0) := (others
       => '0');
15 signal clk : std_logic := '0';
16
17 begin
18
19 --Инстанциация подключение() модуля BRAM из( файла .vho)
20 BRAM : entity work.BRAM_sample
21     port map(
22     clka => clk,  --clock for writing data to RAM.
23     ena => ena,   --Enable signal.
24     wea => wea,   --Write enable signal for Port A.
25     addra => addra, --8 bit address for the RAM.
26     dina => dina,   --8 bit data input to the RAM.
27     douta => douta);  --8 bit data output from the RAM.
28 clk <= not clk after 10 ns;
29 --Процесс симуляции
30 process
31 begin
32     wait for 10 ns;
33     --Заполняем ячейки BRAM данными. Для этого устанавливаем wea
       в "1".
34     for i in 0 to 15 loop
35         ena <= '1';  --RAM Работает всегда.
36         wea <= "1";
37         wait for 20 ns;
38         addra <= addra + "1";
39         dina <= dina + "1";
40     end loop;
41     addra <= "0000";  --сбрасываем параметр адреса в 0
42     for i in 0 to 15 loop
43         ena <= '1';  --RAM Работает всегда.
44         wea <= "0";
45         wait for 20 ns;
46         addra <= addra + "1";
```

```
47      end loop;
48      wait;
49  end process;
50  end arch;
51  \end{lstlisting}
52  \end{Code}
```

Результаты симуляции:

Рис. 11: Рис. 11

Лабораторная работа: Накопитель памяти на основе BRAM

Задача: написать VHDL-код накопителя, использующего BRAM. Накопитель имеет вход $data(din)$ и вход $valid(dv_in)$, в накопитель поступают данные с паузами; затем, когда поступило N (16) отсчетов данных, буфер выдает последовательно без пауз все пришедшие ранее отсчеты данных.

Создаём модуль BRAM

Реализация на VHDL

Листинг 2: Реализация кода

```
    library ieee;
1   use ieee.std_logic_1164.all;
2   use ieee.numeric_std.all;
3   entity bram is
4   port(
5       clk, reset: in std_logic;
6       din: in std_logic_vector (15 downto 0);
7       dv_in: in std_logic;
8       dv_out: out std_logic;
9       dout : out STD_LOGIC_VECTOR ( 15 downto 0 )
10  );
11  end bram;
12  architecture arch of bram is
13      component blk_mem_gen_0 --Объявление компонента BRAM
14          Port (
15              clka : in STD_LOGIC;
16              wea : in STD_LOGIC_VECTOR ( 0 to 0 );
```

```
17              addra : in STD_LOGIC_VECTOR ( 3 downto 0 );
18              dina : in STD_LOGIC_VECTOR ( 15 downto 0 );
19              douta : out STD_LOGIC_VECTOR ( 15 downto 0 ));
20      end component;
21
22      type state_type is (state_wr, state_rd);
23      signal state_reg, state_next: state_type;
24      signal cnt_reg, cnt_next, cnt_next1:unsigned (3 downto
        0);
25      signal dv_reg, dv_reg1, dv_next: std_logic;
26      signal din_reg, dout_reg: std_logic_vector(15 downto 0);
27      signal addr: std_logic_vector(3 downto 0);
28      signal wr_en:  std_logic_vector ( 0 to 0 );
29
30  begin
31  blk_mem_inst : blk_mem_gen_0 --Инстаниация BRAM
32  PORT MAP (clka => clk,
33            wea => wr_en,
34            addra => addr,
35            dina => din,
36            douta => dout);
37
38  process(clk,reset) -- Настройка смены состояний по фронту clk
39  begin
40      if (reset='1') then
41          state_reg <= state_wr;
42          cnt_reg <= (others => '0');
43          dv_reg <= '0';
44          dv_reg1<='0';
45
46      elsif (rising_edge(clk)) then
47          state_reg <= state_next;
48          cnt_reg <= cnt_next;
49          dv_reg <= dv_next;
50          dv_reg1 <= dv_reg;
51      end if;
52  end process;
53
54  process(state_reg, cnt_reg, dv_in, dv_reg, dv_reg1) --
        Смена состояний
55  begin
56  state_next <= state_reg;
57  cnt_next <= cnt_reg;
58  dv_out <= dv_reg1;
59  addr <= std_logic_vector(cnt_reg);
60  case state_reg is
```

```
61      when state_rd =>
62          dv_next <= '1';
63          wr_en <= "0";
64          if (cnt_reg /= 15) then
65              cnt_next <= cnt_reg + 1;
66
67          else
68              state_next <= state_wr;
69              cnt_next <= (others => '0');
70          end if;
71      when state_wr =>
72           dv_next <= '0';
73           wr_en <= "1";
74           if (dv_in = '1') then
75                cnt_next <= cnt_reg + 1;
76
77           end if;
78           if (cnt_reg /= 15) then
79               state_next <= state_wr;
80
81           elsif (dv_in = '1') then
82               state_next <= state_rd;
83               cnt_next <= (others => '0');
84           end if;
85  end case;
86  end process;
87  end arch;
88      if (reset='1') then
89          state_reg <= state_wr;
90          cnt_reg <= (others => '0');
91          dv_reg <= '0';
92          dv_reg1<='0';
93
94      elsif (rising_edge(clk)) then
95          state_reg <= state_next;
96          cnt_reg <= cnt_next;
97          dv_reg <= dv_next;
98          dv_reg1 <= dv_reg;
99      end if;
100 end process;
101
102 process(state_reg, cnt_reg, dv_in, dv_reg, dv_reg1)
103 begin
104 state_next <= state_reg;
105 cnt_next <= cnt_reg;
106 dv_out <= dv_reg1;
```

```
107 addr <= std_logic_vector(cnt_reg);
108 case state_reg is
109     when state_rd =>
110         dv_next <= '1';
111         wr_en <= "0";
112         if (cnt_reg /= 15) then
113             cnt_next <= cnt_reg + 1;
114 
115         else
116             state_next <= state_wr;
117             cnt_next <= (others => '0');
118         end if;
119     when state_wr =>
120          dv_next <= '0';
121          wr_en <= "1";
122          if (dv_in = '1') then
123               cnt_next <= cnt_reg + 1;
124 
125          end if;
126          if (cnt_reg /= 15) then
127              state_next <= state_wr;
128 
129          elsif (dv_in = '1') then
130              state_next <= state_rd;
131              cnt_next <= (others => '0');
132          end if;
133 end case;
134 end process;
135 end arch;
```

Самопроверяющийся тестбенч

Листинг 3: Тестбенч

```
    library ieee;
 1  use ieee.std_logic_1164.all;
 2  use ieee.std_logic_textio.all;
 3 
 4  library std;
 5  use std.textio.all;
 6 
 7 
 8  entity bram_tb is
 9  end bram_tb;
10 
11  architecture arch of bram_tb is
12  type num_input1 is array (integer range 0 to 15) of
```

```
    std_logic_vector(15 downto 0);
13
14
15     signal dout : std_logic_vector (15 downto 0);
16     signal num_input: num_input1;
17     signal din: std_logic_vector (15 downto 0);
18     signal dv_in,dv_out:std_logic;
19     signal clk : std_logic := '0';
20     signal reset: std_logic;
21     signal err: boolean := false;
22
23
24 begin
25  dut: entity work.bram (arch)
26
27 port map (
28 clk=>clk,
29 dout=>dout,
30 dv_in=>dv_in,
31 dv_out=>dv_out,
32 din=>din,
33 reset=>reset
34 );
35     clk <= not clk after 10 ns;
36
37     read_file_process: process
38        file   fd    : text is in  "../ip_header.txt";
39        variable word: std_logic_vector(15 downto 0);--line;
40        variable j : integer;
41        variable Message : line;
42        variable inline : line;
43     begin
44
45        Write ( Message, string'("- Reading data from file:  "
    ));
46        writeline(output, Message);
47        j := 0;
48        for i in 0 to 15 loop
49
50           assert (j < 16)
51              report "File bigger than IP Header"
52              severity failure;
53
54           readline(fd, inline);
55           read(inline, word);
56           num_input(j)<=word;
```

```
57          j := j + 1;
58          Write ( Message, word);
59          writeline(output, Message);
60
61       end loop;
62 --       valid <='1';
63 --      wait for 30 ns;
64 --      valid <='0';
65       wait;
66    end process;
67
68    -- input process
69    process
70    begin
71    reset <='1';
72    wait for 100 ns;
73    reset <='0';
74    wait for 20 ns;
75    for i in 0 to 15 loop
76          din<=num_input(i);
77          dv_in<='1';
78          wait for 20 ns;
79
80     end loop;
81     dv_in<='0';
82     wait;
83     end process;
84
85
86    -- compare process
87    process
88       variable Message : line;
89
90    begin
91       Write ( Message, string'("- Waiting for RTL Output:  "
      ));
92       writeline(output, Message);
93
94       for i in 0 to 2 loop
95          wait until rising_edge(clk);
96       end loop;
97       wait until (dv_out='1');
98       wait for 20 ns;
99       for k in 0 to 15 loop
100
101      if (dout /= num_input(k)) then
```

```
102             err <= true;
103             Write ( Message, num_input(k));
104             writeline(output, Message);
105             Write ( Message, dout);
106             writeline(output, Message);
107         else
108             err <= false;
109         end if;
110         wait until rising_edge(clk);
111         end loop;
112 
113         wait until rising_edge(clk);
114 
115         if (err) then
116             Write ( Message, string'("- Compare failed:  "));
117         else
118             Write ( Message, string'("- Compare OK!"));
119         end if;
120         writeline(output, Message);
121         wait;
122     end process;
123 end;
```