Lab 5: Memories: ROMs and BRAMs Internal to the FPGA

EE-459/500 HDL Based Digital Design with Programmable Logic

Electrical Engineering Department, University at Buffalo

Last update: Cristinel Ababei, 2012

1. Objective

The objective of this lab is to illustrate the use of ROM and block RAM memories located inside the FPGA – a Spartan-6 in the case of our Atlys board. We'll learn how to use the ISE's Core Generator tool to create BRAMs. Depending on what your course project will do, you may need to use such memories in your project.

2. Description

In this lab we'll create a project to implement the following design description: The circuit must contain two memories. A ROM created using a case statement and initialized to the desired values (such as the coefficients of a filter). This memory will be inferred as a distributed RAM memory by the Xilinx synthesis tool (XST). The second memory is a block RAM (BRAM) created using the Core Generator tool (part of ISE WebPack). The contents of these memories will be read continuously and displayed on the 7 LEDs. Slide switch SW(0) is used to select between the two outputs of the two memories to drive the LEDs. A simplified representation of this functionality is shown in the block diagram in Fig.1.

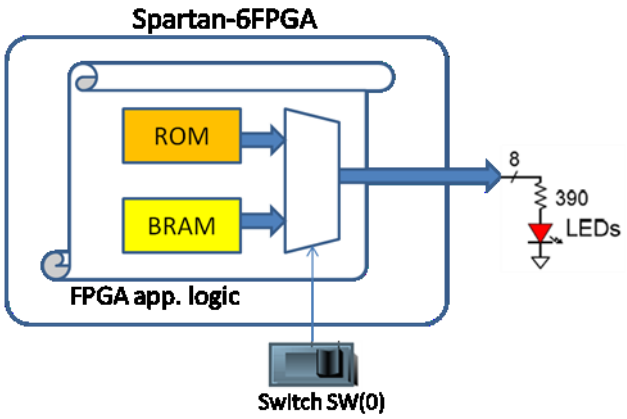

Figure 1 Block diagram of desired circuit

3. ISE WebPack project

Create a new ISE project and add to it the VHDL files listed at the end of this document. These files contain the declaration and description of this lab's entities including clock divider, ROM, and top level design. These files together with other useful files (.ucf and .coe files) are also included in the downloadable archive with all the data for this lab.

To create a custom single-port block RAM using the Core Generator, inside your ISE project, follow these steps:

- First create, using ISE's or any other text editor, a file named **my_bram8x8.coe** and save it in the main directory of your ISE project, with the following contents:

```
memory_initialization_radix=2;
memory_initialization_vector=
00000000,
01000000,
00100000,
00010000,
00001000,
00000100,
00000010,
00000001;
```

- Select New Source->IP (CORE Generator and Architecture Wizard); name it my_bram8x8 and click Next.
- In the new New Source Wizard that pops-up, select Memories and Storage Elements->RAMs & ROMs->Block Memory Generator and click Next then Finish, action which closes the New Source Wizard window and brings-up the Block Memory Generator window.
- Click Next, leave the Memory Type as "Single Port RAM" and click Next.
- On Page 3 of 6, set Write Width to 8 and Write Depth to 8. Click Next.
- On Page 4 of 6, select Load Init File and then browse to locate the file created earlier, **my_bram8x8.coe**, and then click Next. Click Next again. Page 6 of 6 should look like in Figure 2.
- Click Generate. This will generate the memory core and in your ISE project's Console you should get the message:
  `Wrote CGP file for project 'my_bram8x8'.`
  `Core Generator create command completed successfully.`

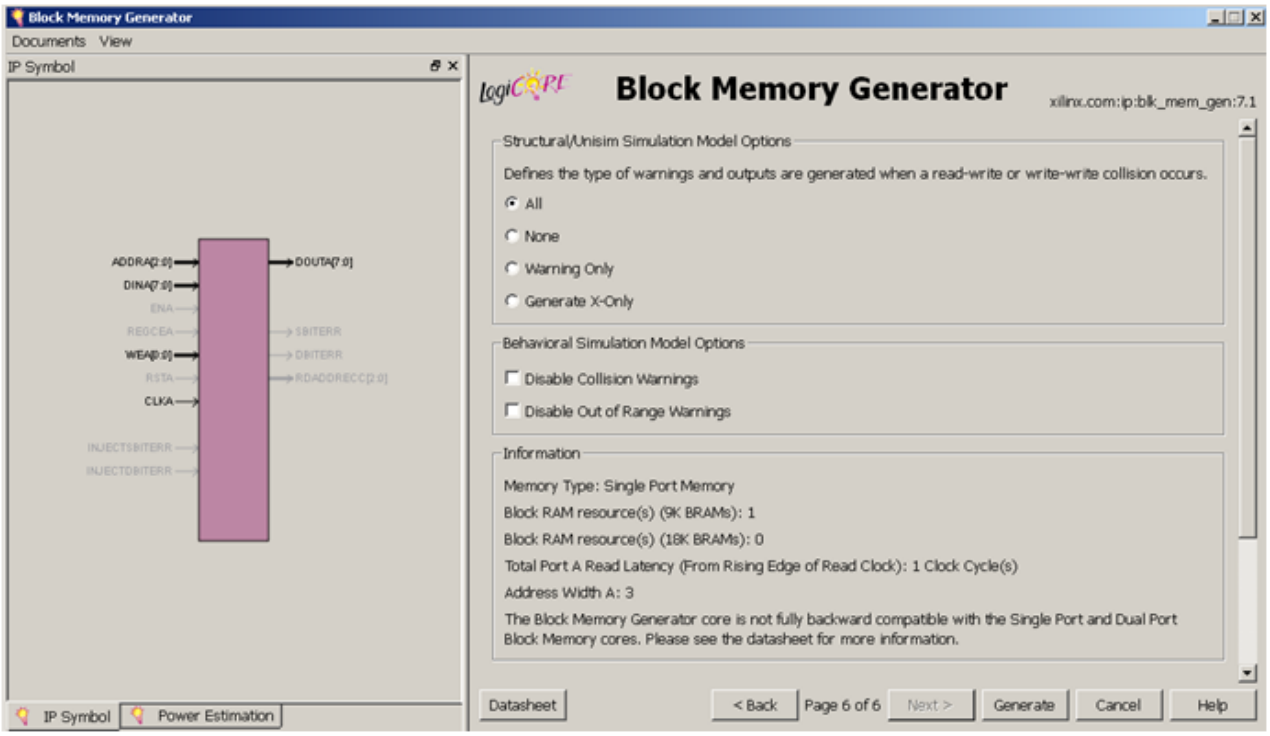

Figure 2

During the above process, several files are created and stored in **ipcore_dir/** folder of your ISE project main folder. Among them, you can find **my_bram8x8.vhd**. Open it and read the VHDL code. Identify the BRAM entity declaration, and use it to instantiate a component in the top level VHDL file of your project.

Do the pin assignment. After this, your UCF file should have the following contents:

```
NET "clk_100MHz" LOC = L15;
NET "switch" LOC = A10;
NET "leds[0]" LOC = U18;
NET "leds[1]" LOC = M14;
NET "leds[2]" LOC = N14;
NET "leds[3]" LOC = L14;
NET "leds[4]" LOC = M13;
NET "leds[5]" LOC = D4;
NET "leds[6]" LOC = P16;
NET "leds[7]" LOC = N12;
```

Run the Implement Design step inside ISE WebPack to perform placement and routing and observe the messages that the tool prints in the Console window. These messages provide useful information about the resource utilization on the FPGA as well as performance estimates.

Generate the programming .bit file and program the FPGA. Verify the operation of your design; turn on/off the first slide-switch. Observe and comment.

4. Lab assignment

Modify the project to be able to also write into the BRAM. The writing process should allow writing into BRAM new words (as dictated by the status of the slide switches) during eight cycles. These cycles should be controlled via one of the push-buttons on the Atlys board (BTND P3). It is up to you how you want to utilize the remaining push-buttons to achieve the desired operation of the whole system.

5. Credits and references

[1] XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices - ROMs and ROM coding examples (page 247): http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/xst_v6s6.pdf
[2] Spartan-6 Libraries Guide for HDL Designs:
http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/spartan6_hdl.pdf
[3] Spartan-6 FPGA Block RAM Resources:
http://www.xilinx.com/support/documentation/user_guides/ug383.pdf

Appendix: Listing of VHDL code

my_modules.vhd

```
-- This is a ROM. XST tool (part of ISE WebPack tools) will
-- infer this and implement this declaration as a distributed
-- memory.
-- The contents of this basically LUT will be utilized to drive
```

```
-- the 8 LED on the Atlys board. This should turn them on one-by-one
-- from right to left.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY rom8x8 IS
 PORT (
       addr: in  std_logic_vector(2 downto 0);
       dout:    out std_logic_vector(7 downto 0));
END rom8x8;

ARCHITECTURE behav OF rom8x8 IS
BEGIN

PROCESS(addr)
BEGIN
  CASE addr IS
    when "000" => dout <= "00000001";
    when "001" => dout <= "00000010";
    when "010" => dout <= "00000100";
    when "011" => dout <= "00001000";
    when "100" => dout <= "00010000";
    when "101" => dout <= "00100000";
    when "110" => dout <= "01000000";
    when "111" => dout <= "10000000";
    when others => NULL;
  END case;
END process;

END behav;

-- This is a clock divider. It takes as input a signal of 100 MHz
-- and generates an output as signal with a frequency of about 1 Hz.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ck_divider is
    Port ( CK_IN : in STD_LOGIC;
           CK_OUT : out STD_LOGIC);
end ck_divider;

architecture Behavioral of ck_divider is

constant TIMECONST : integer := 84;
signal count0, count1, count2, count3 : integer range 0 to 1000;
signal D : std_logic := '0';
begin

process (CK_IN, D)
begin
       if (CK_IN'event and CK_IN = '1') then
              count0 <= count0 + 1;
              if count0 = TIMECONST then
                     count0 <= 0;
                     count1 <= count1 + 1;
              elsif count1 = TIMECONST then
                     count1 <= 0;
                     count2 <= count2 + 1;
              elsif count2 = TIMECONST then
                     count2 <= 0;
                     count3 <= count3 + 1;
```

```
                elsif count3 = TIMECONST then
                        count3 <= 0;
                        D <= not D;
                end if;
        end if;
        CK_OUT <= D;
end process;

end Behavioral;
```

top_level.vhd

```
-- This is a simple design, in which we use two memories:
-- memory1: ROM created using a case statement and initialized to desired values
-- This should be inferred as a distributed RAM memory by the Xilinx tool
-- memory2: block RAM created using the Core Generator and then only instantiated
-- This mmeory is initialized using a .coe file
-- The contents of these memories will be displayed on the 7 LED of Atlys.
-- Slide switch SW(0) is used to select between the two memories.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top_level is
    Port ( clk_100MHz : in  STD_LOGIC; -- FPGA's external oscillator
           switch : in  STD_LOGIC; -- hooked to slide switch SW(0) on Atlys board
           leds : out  STD_LOGIC_VECTOR (7 downto 0)); -- drives all eight LEDs on board
end top_level;

architecture Structural of top_level is

component ck_divider
        Port (CK_IN : in STD_LOGIC;
                        CK_OUT : out STD_LOGIC);
end component;

-- Question: what would be different if instead of using this component we would
-- simply declare an array that would also need to be initialized with desired values?
-- type ram_t is array (0 to 7) of std_logic_vector(7 downto 0);
-- signal ram : ram_t := (others => (others => '0'));
-- Exersize: Currently if mem_selector changes the counter does not reset. Change
-- the code such that each time the mem_selector changes the counter is reset to
-- zero.
component rom8x8
        PORT (addr : in  std_logic_vector(2 downto 0);
                        dout : out std_logic_vector(7 downto 0));
end component;

-- This component is created using the Core Generator. Its VHDL description
-- is inside ipcore_dir/my_bram8x8.vhd, which was created during the use
-- Core Generator as explained in the lab.
component my_bram8x8
  PORT (
```

```
    clka : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
end component;

signal clk_1Hz : STD_LOGIC;
signal my_addr_counter : STD_LOGIC_VECTOR (2 downto 0) := "000";
signal dout_rom8x8, dout_bram8x8 : STD_LOGIC_VECTOR (7 downto 0);
-- for the time being, we'll only read from this block RAM, so
-- let's set all data ins to zero;
signal dina_null : STD_LOGIC_VECTOR (7 downto 0) := "00000000";
signal wea_null : STD_LOGIC_VECTOR(0 DOWNTO 0) := "0"; -- no need for writing in this example

begin

clock_divider : ck_divider port map (clk_100MHz, clk_1Hz); -- poor instantiation
memory1 : rom8x8 port map (addr => my_addr_counter, dout => dout_rom8x8); -- better instantiation
-- Instantiate BRAM.
memory2 : my_bram8x8 port map (
        clka => clk_1Hz, -- clock for writing data to RAM
        wea => wea_null, -- write enable signal for Port A
        addra => my_addr_counter, -- 3 bit address for the RAM
        dina => dina_null, -- 8 bit data input to the RAM
        douta => dout_bram8x8); --8 bit data output to the RAM

multiplex_out : process (clk_1Hz) is
begin
        if (clk_1Hz'event and clk_1Hz = '1') then
                case switch is
                        when '0' =>
                                leds <= dout_rom8x8;
                        when '1' =>
                                leds <= dout_bram8x8;
                        when others => NULL;
                end case;
                my_addr_counter <= std_logic_vector( unsigned(my_addr_counter) + 1);
        end if;
end process;

end Structural;
```