CSEE W4840 Embedded System Design Lab 6

Stephen A. Edwards

Due March 9, 2006

Abstract

Implement an OPB peripheral that maps the off-chip SRAM into the Microblaze's memory space. Demonstrate that it works by writing a simple memory-test program that writes data into the memory space, reads it back, and verifies the values are right.

1 Introduction

This is the trickiest lab of the course, but closely reflects the sort of thing you will have to do for the project.

Your task is to create a sort of protocol converter that speaks the OPB protocol on one side so your C programs running on the Microblaze can communicate with it, and the off-chip, on-board 256K × 16 SRAM (Toshiba TC55V16256J). To do this, you need to understand both protocols, how the XESS XSB-300E board is wired, and how to implement such a protocol converter in VHDL.

In lab6.tar.gz, I've supplied a project that implements half of the lab. Specifically, it contains a working OPB peripheral that maps one of the on-chip BRAMS into memory and a small C program that performs a memory test that verifies its operation. Look it over carefully (I discussed it at length in class) to understand how it works. The main component is a finite-state machine that controls the various signals for the BRAM and the OPB.

For this lab, remove the BRAM block from the supplied VHDL file and replace it with a connection to the off-chip SRAM.

You will need to understand the operation of the Toshiba 256K × 16 SRAM. Read the data sheet (it's posted on the class website) carefully.

2 I/O Connections

You will also need to tell the Xilinx tools to connect your VHDL module to the SRAM chip. It needs to know where the pins are and what they're named, that these pins are top-level ports, that your peripheral connects to them, and how your peripheral connects to them. This involves a number of steps:

- Figure out the number and names of the pins on the FPGA that connect to the SRAM chip by reading the XSB-300E documentation (it's on the class website).
- Add information about these pins to the data/system.ucf file. The format is fairly simple. For example, the first few data bits should be described as follows:

```
net PB_D<0> loc="p153";
net PB_D<1> loc="p145";
net PB_D<2> loc="p141";
```

- Modify the pcores/opb_bram_v1_00_a/data/opb_bram_v2_1_0.mpd file to include connections to these pins. A line for the sixteen-bit data bus would look like

```
PORT PB_D = "", DIR=INOUT, VEC=[15:0],
    3STATE=FALSE, IOB_STATE=BUF
```

- Add the relevant ports to the system.mhs file, e.g., by adding

```
PORT PB_D = PB_D, DIR = INOUT, VEC = [15:0]
```

  to the list of global ports (at the beginning of the file) and trivial mappings for the ports on your peripheral to the instance of your peripheral. These names have to match those in the .mpd file.

```
 PORT PB_D = PB_D
```

- Add these ports to the VHDL entity for your peripheral. Their names and widths must match those in the .mpd file.

The LED display controller from Lab 1 is a simple reference for how to connect to pins on the FPGA.

3 Pad Drivers

The data pins on the SRAM are bidirectional, that is, they can be used as both receivers and transmitters to save pins. Unfortunately, this complicates your design slightly because you need to tell the FPGA to connect to these pins using so-called tri-state drivers that can either receive or transmit depending on a control signal.

The way to ask for such drivers is to instantiate special components that the Xilinx tools know about that contain the desired functionality. The two you should use for this assignment are IOBUF_F_24, a bidirectional, TTL-level driver with 24 mA drive, and OBUF_F_24, an output buffer with 24 mA drive. Use the bidirectional driver for the data lines, since they can convey data both from the SRAM and to the SRAM, and the output buffers for the address lines because they are connected to many other peripherals in addition to the SRAM.

Using these blocks is fairly straightforward, just instantiate them where you need them, connect one of their terminals to the port that will become the pin, and connect the rest to internal signals.

```
entity my_peripheral is
  port (
    mypin1 : out std_logic;
    mypin2 : inout std_logic);
end my_peripheral;

architecture Behavioral of my_peripheral is

  component OBUF_F_24
    port (
      O : out STD_ULOGIC; -- the pin
      I : in STD_ULOGIC); -- signal to pin
  end component;

  component IOBUF_F_24
    port (
      O : out STD_ULOGIC; -- signal from pin
      IO : inout STD_ULOGIC; -- the pin
      I : in STD_ULOGIC;  -- signal to pin
      T : in STD_ULOGIC); -- 1=drive IO with I
  end component;

  signal mysignal1, mysignal2,
      mysignal3, mytristate : std_logic;

  mypin1pad : OBUF_F_24 port map (
    O => mypin1,
    I => mysignal1);

  mypin2pad : IOBUF_F_24 port map (
    O => mysignal2,
    IO => mypin2,
    I => mysignal3,
    T => mytristate);
end Behavioral;
```

4 The Assignment

Implement a 16-bit peripheral that maps the 256K $\times$ 16 SRAM into memory space somewhere. Write a memory test procedure (e.g., by adapting the one provided) that verifies that you can successfully read and write 16-bit quantities from this memory.

Map this in memory such that bytes numbered 0 and 1 hold data and bytes numbered 2 and 3 do not, i.e., make this a series of 16-bit words hiding in 32-bit words.

In the end, turn in all the code you have written or modified for this lab. As usual, coding style, clarity, and succinctness will be taken into account during grading.