Lab-exercise

Lab6:
OPB-interface

Introduction

During this lab, the student is going to generate an interface between the dual port RAM and the “software system”. This means that a VHDL entity has to be written that can communicate with the OPB bus. The software will eventually use this bus to write data to the dual port RAM or read from it.

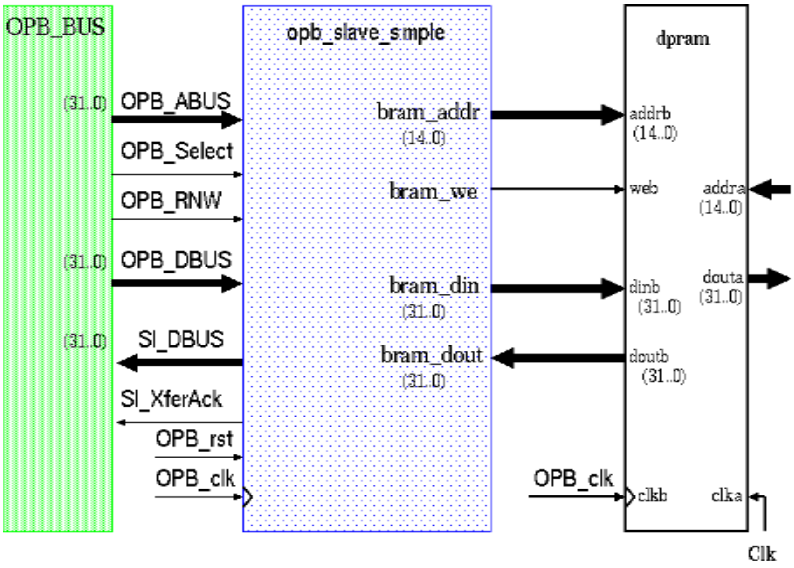

Figure 1

Goals

- Understand the OPB-bus

- Understand and implement the basic functionality of an OPB bus interface.

Knowledge background

Basic VHDL
Module 4a: generation of the dual port RAM

Classification

Difficulty level (from 1 to 5): 3.5
Time needed (without support): 6 hours

Input

The following folders and files are given for module5a.
Hardware
opb_slave_simple.vhd: VHDL template
Simulation
TB_opb_slave_simple.vhd: complete testbench
General_TB_pack.vhd: package used in the testbench
dpram.vhd: dual port RAM, instantiated in the testbench
dpram.mif: initial values for the dual port RAM

The Lab

1. Study the OPB-bus.

Figure 2 shows the relevant signals of an OPB bus connecting 2 master units to 2 slave units.

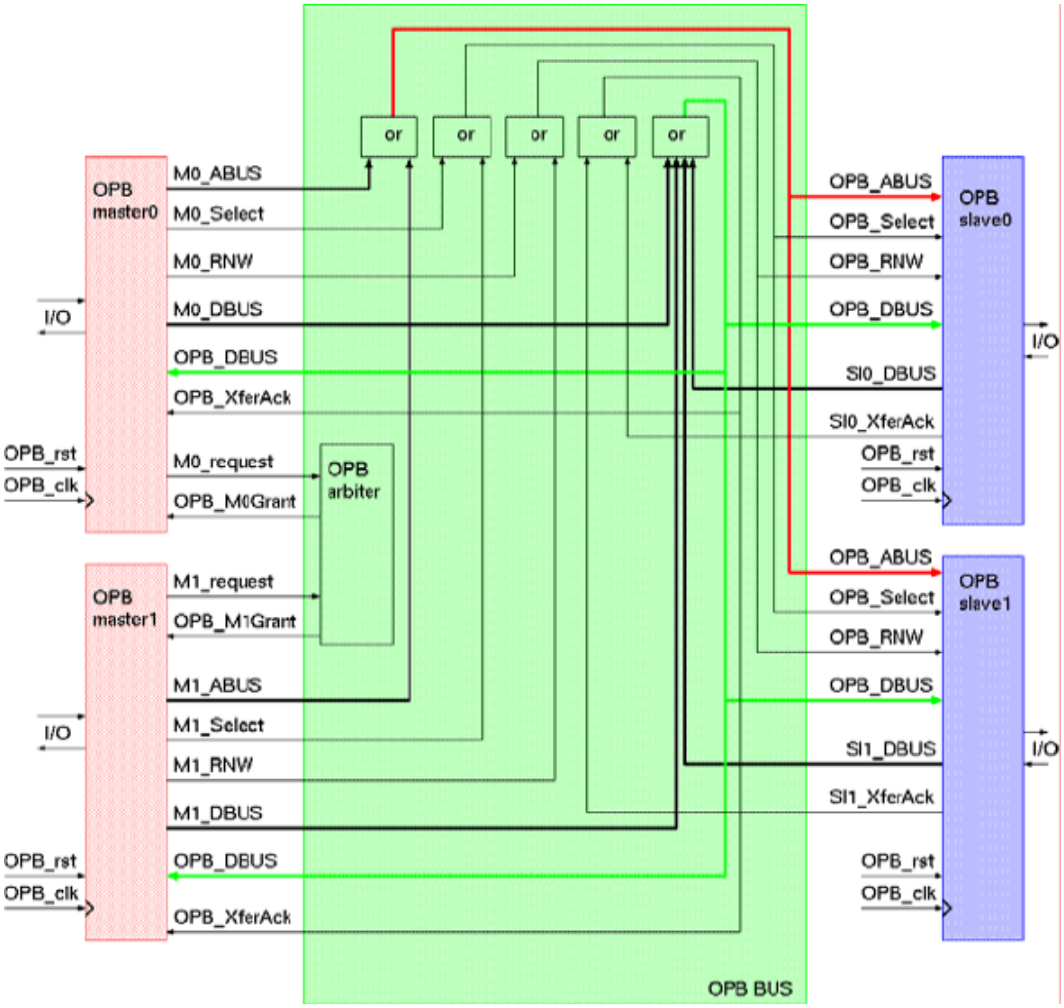

Figure 2

In this lab there is only one master and one slave. OPBmaster0 would be a processor, while OPBslave0 would be the OPB_slave_simple we have to implement here.

As you can see on figure 2 the OPB bus in fact is nothing more than an arbiter deciding which master gets access to the bus and a number of “or” gates. This

means that masters as well as slaves that are not active have to drive zeros on their outputs to make the bus function correctly. Address and data busses are both 32 bits wide. The signals in figure 2 are the only ones important in this lab, but those interested can find the functionality of sl_errack, sl_toutsup, sl_retry, opb_be and opb_seqaddr in the OPB-bus documentation.
Below are the timing diagrams for reading from and writing to the OPB_slave_simple. The OPB master, slave units and the bus itself are synchronous to the rising edge of the OPB_clk. That means that the bus is fully synchronous. In this lab a write operation is done as follows.
1. After receiving a grant, and on the rising edge of OPB_Clk, the master
- puts the address Awrite on the OPB_ABUS,
- puts the data Dwrite on the OPB_DBUS,
- makes the OPB_Select signal high
- makes the OPB_RNW signal low.
2. On the next rising edge of OPB_clk the slave detects being selected (this depends on the address OPB_ABUS and the value of OPB_Select) and writes the data, which in our case means passing it to the DPRAM.
3. In the final cycle the slave makes acknowledge high such that the master knows the write action is finished in the next cycle.

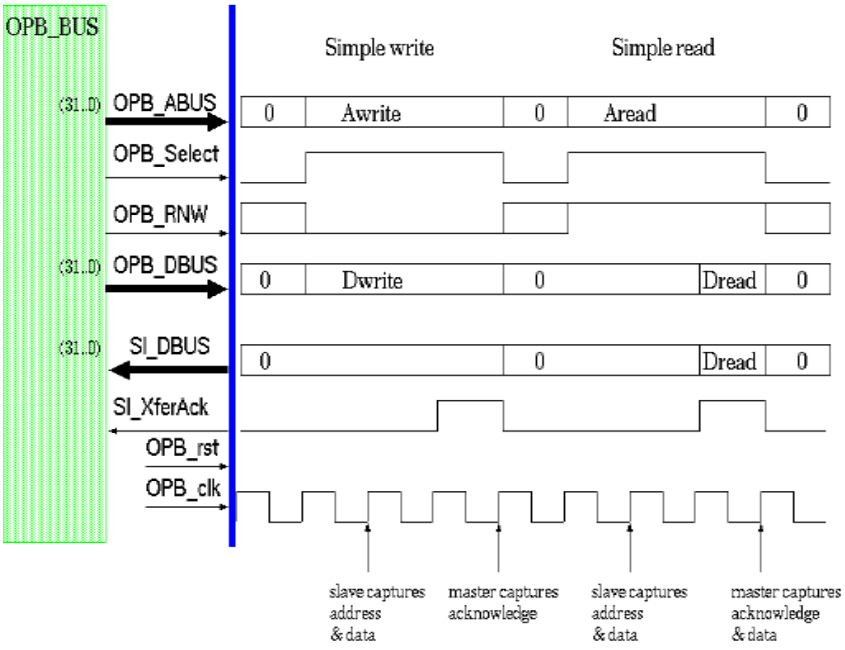

Figure 3

A read operation is analogous. In this case the OPB_RNW signal is put high and the slave sends the data asked for on the SL_DBUS at the same time it makes the acknowledge high.
Question: Why is "Dread" also available on OPB_DBUS?
2. Write the VHDL code for OPB_slave_simple.
Complete opb_slave_simple.vhd such that it follows the protocol of figure 3.
The opb_slave_simple entity has four generics:
- C_BASEADDR,

- C_HIGHADDR,
- VideoRamStartAddress and
- VideoRamLastAddress.

Specific values for these generics are defined on the hierarchically higher level where opb_slave is instantiated.
Generics C_BASEADDR and C_HIGHADDR define the address area within which the opb_slave_simple is active. This means the opb_slave_simple has to give an acknowledge when the read or write address is within the range from C_BASEADDR till C_HIGHADDR.

Figure 4

The other two generics (VideoRamStartAddress and VideoRamLastAddress) are the begin- and the end-address of the video RAM we are going to connect to the opb_slave_simple. The table below shows when an ack pulse has to be generated and how the SI_DBUS and bram_din have to be driven as a function of the address used.

	A and E		B and D		C	
	read	write	read	write	read	write
ack pulse	no	no	yes	yes	yes	yes
SL_DBUS	0	0	0	0	data	0
bram_din	0	0	0	0	0	data

Area B and D area can later possibly be used to add extra registers.
Use extra internal registers to generate the acknowledge pulse at the right moment. Also make sure the SI_DBUS is always zero, except during the acknowledge phase of a read operation in area C. In that case the data coming from the DPRAM have to be passed to the OPB bus.

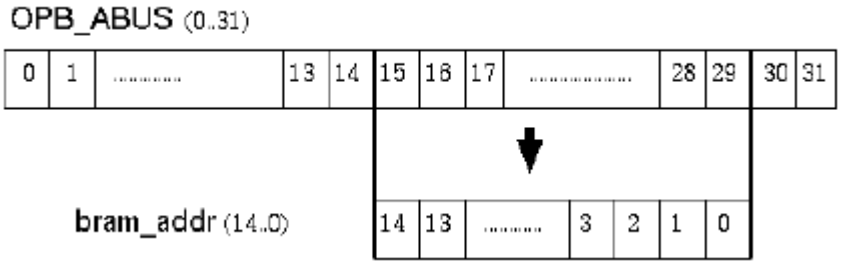

Figure 5

When driving the bram_addr signals you have to keep in mind that the addresses of the processor are byte oriented. This means that the two LSB bits of OPB-ABUS will always be zero when the processor wants to read or write 32 bit data.

Take care that when driving the RAM the write enable signal (we) only comes high for one clock period.

3. Verifiy the opb_slave_simple using the given testbench TB_opb_slave_simple.

Question: When you analyse TB_opb_slave in library WORK, which other entities also have to be analyzed in this library such that the testbench would run?

The simulation of this testbench (run –all) should generate no errors. Don't forget to look at the waveforms of all relevant signals and to compare them to figure 3.