

BYOC course infrastructure

For assignments #2 - #6

1. The BYOC course infrastructure description

All our assignments will use the same infrastructure. We are using the Nexys2 board for the BYOC course. This board has some HW on it that is connected to the FPGA.

HW circuitry connected to the FPGA:

- Switches – There are 8 switches that can be connected to known pins of the FPGA.
- Pushbuttons – There are 8 pushbuttons connected to other pins of the FPGA. We will use 4 of them.
- 50 MHz oscillator signal – we will use this as our clock source.
- P/S-2 connector – to which we can connect a keyboard and so, we can get information from the keyboard and send that towards the FPGA.
- VGA output connector – We can drive these signals from the FPGA. If we connect a VGA screen to this connector, we can display a picture on the screen. We will discuss later what is displayed.
- Flash memory – This memory keeps its data even when we switch off the power to the board. Thus, we can keep a program in that memory and somehow load it into the Instruction Memory of our CPU when the power is switched on. We will not use that option in this course. However, the circuitry allowing writing to the Flash Memory and reading from that memory is included in the infrastructure.
- RS232 connector – This connector has 2 lines, Rx and Tx allowing receiving serial data from another device (e.g., a PC computer) and also sending serial data back to that device.

It is not enough that the FPGA pins are connected to that circuitry. For example, although the RS232 Rx signal receives a signal that represents a stream of '1'-s and '0'-s, we need to have some design inside the FPGA that can "understand" that stream of bits and convert it to commands or data. For that, and for other functions we need to use during the course, we design a special component called the **BYOC_Host_Intf** – i.e., Host Interface. This interface includes the following functionality:

- It has the MIPS Instruction memory – That is a 1Kx32 memory residing at addresses 00400000h - 00400FFCh.
- It has the MIPS Data Memory – This part has 3 parts in it
 - In addresses 10000000h - 10000FFCh we have a 1Kx32 memory
 - In addresses 20000000h - 20003FFCh we have a 4Kx32 memory that also serves as the VGA Screen memory (to be explained later)
 - In addresses 30000000h - 300000FCh we have a Memory mapped I/O part that allows us to access the KBD, switches, LEDs, 7Seg LEDs, Host Monitor and the Flash Interface.
- It has 16 read back inputs of 32 bit each (denoted rdbk0-rdbk15).
- It has the ability to get commands via the RS232 channel. These commands allow us to do the following:
 - Issue RESET and HOLD signals to your design
 - Using the HOLD line, we can also work with your design in a single clock mode in which, after each single-clock, we will read the 16 rdbk inputs and display them on the screen of the PC driving the RS232 channel. This will be our main debugging tool during the implementation phase.

- We can load the Instruction Memory (IMem) with a program read from a text file. This means that in our PC computer we will ask to load a specific text file to the IMem part of the BYOC_Host_Intf component.
-
- It has a special circuitry that reads constantly from the 2nd part of the Data Memory (DMem) and displays it on the VGA screen. That circuitry is therefore called the VGA Controller.
- It has a circuitry that allows us to copy data from the IMem into the on-board Flash memory. It also allows copying data in the other direction.
- It has special small Boot ROM that resides in addresses 00000000h - 000000FCh, a 256x32 memory that has a small program copying from the Flash Memory to the IMem. If the Flash Memory has a program in it, we can set up the switches so that then a power up reset will start to run from address 0000000h and the Boot ROM will load the program and “launch” it.

In Fig. 1 we show the **BYOC_Host_Intf**. The **black** signals are the signals the students use in their design throughout the course. The **blue** signals are the ones you will not use but should still be connected to the FPGA pins. These will be “connected” for you in advance in the top.vhd files you’ll get with the Homework assignments.

In FPGA design, we first run a simulation of the design. For that we “wrap” our design with a “Test Bench”, which is another design that “injects” signals into the tested (our) design and reads the outputs created by our design. We will also supply a **BYOC_Host_Intf_4sim** which will be a component with the exact same i/o pins as the **BYOC_Host_Intf** but will be made for simulation.

In Fig.2 below you can see the scheme describing a simulation project. Everything is “wrapped” by the Test Bench (TB). The top.vhd file (e.g., HW2_top.vhd, or HW4_MIPS.vhd etc.) will include a **Clock_Driver** component which divides the 50 MHz input clock to a 25 MHz clock signal that will be used by our design. It will also include the **BYOC_Host_Intf** component (actually the **BYOC_Host_Intf_4sim**). And, of course, it will include your design. Your design will be activated by the CK, RESET and HOLD signals coming out of the **Clock_Driver** and the **BYOC_Host_Intf** components. Your design will “output” signals to the TB. The TB will “read” these signals during the simulation and will compare them to a pre-prepared dat file that has the expected values of these signals. The output signals to the TB are drawn in **red**. These are:

- 1) CK_out_to_TB - a signal identical to the CK signal coming out of the **Clock_Driver**
- 2) RESET_out_to_TB - a signal identical to the RESET signal driven by the **BYOC_Host_Intf**
- 3) HOLD_out_to_TB - a signal identical to the HOLD signal driven by the **BYOC_Host_Intf**
- 4) rdbk0_out_to_TB to rdbk15_out_to_TB – 16 vector signals, 32 bit each that will have the data we want to check. We will tell you what should be tested during simulation. We will also connect that data to the 16 rdbk vectors going into the **BYOC_Host_Intf** so that we can also test the same signals in the actual circuit in the implementation phase.

Fig. 3 is a similar scheme of the implementation scheme. Here we see also the Nexys2 board and the ucf file that tells the Xilinx ISE SW which signal goes to which i/o pin of the FPGA.

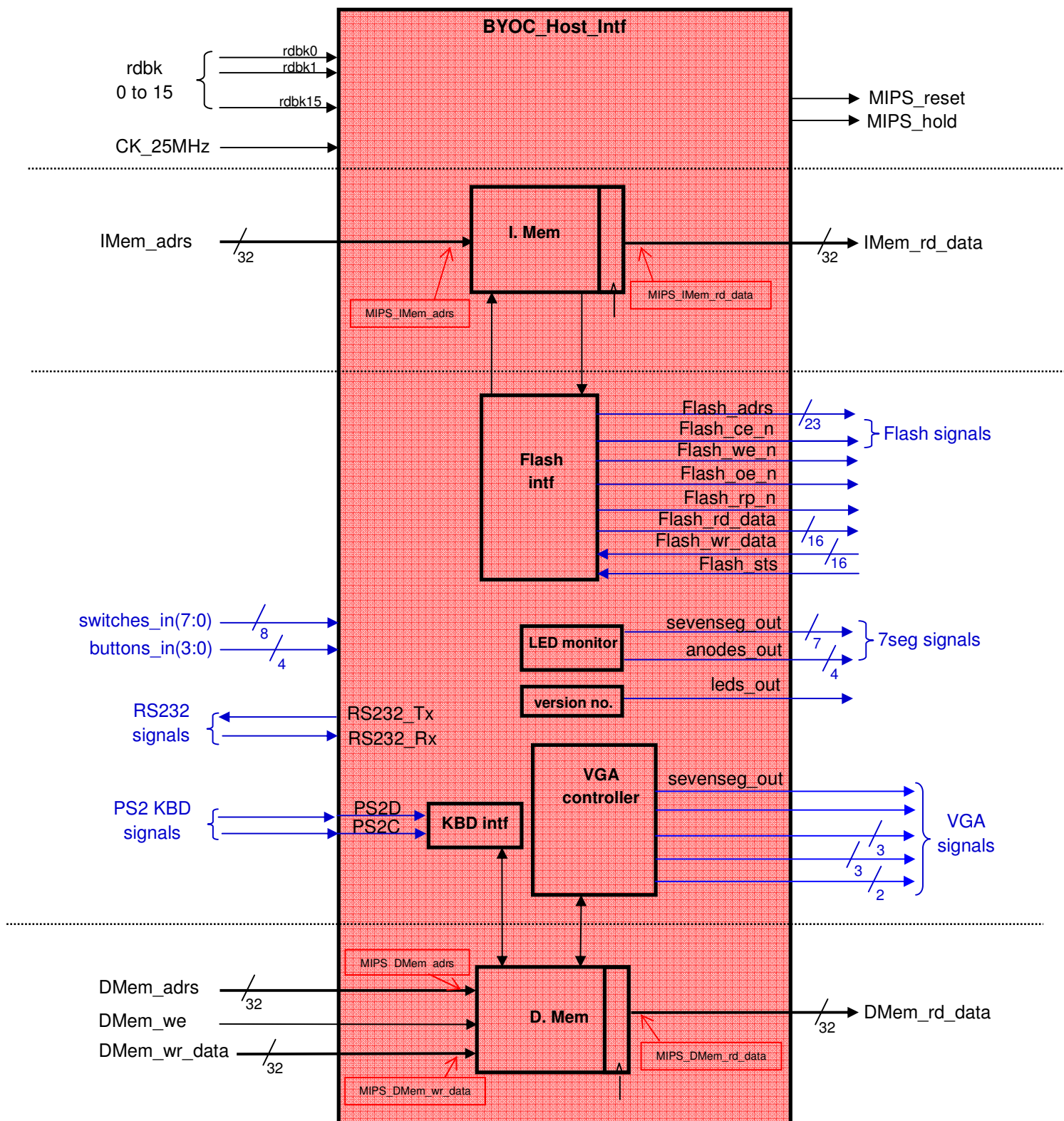


Fig. 1 – BYOC_Host_Intf.ngc

the_Test_Bench.vhd

Your_design.vhd

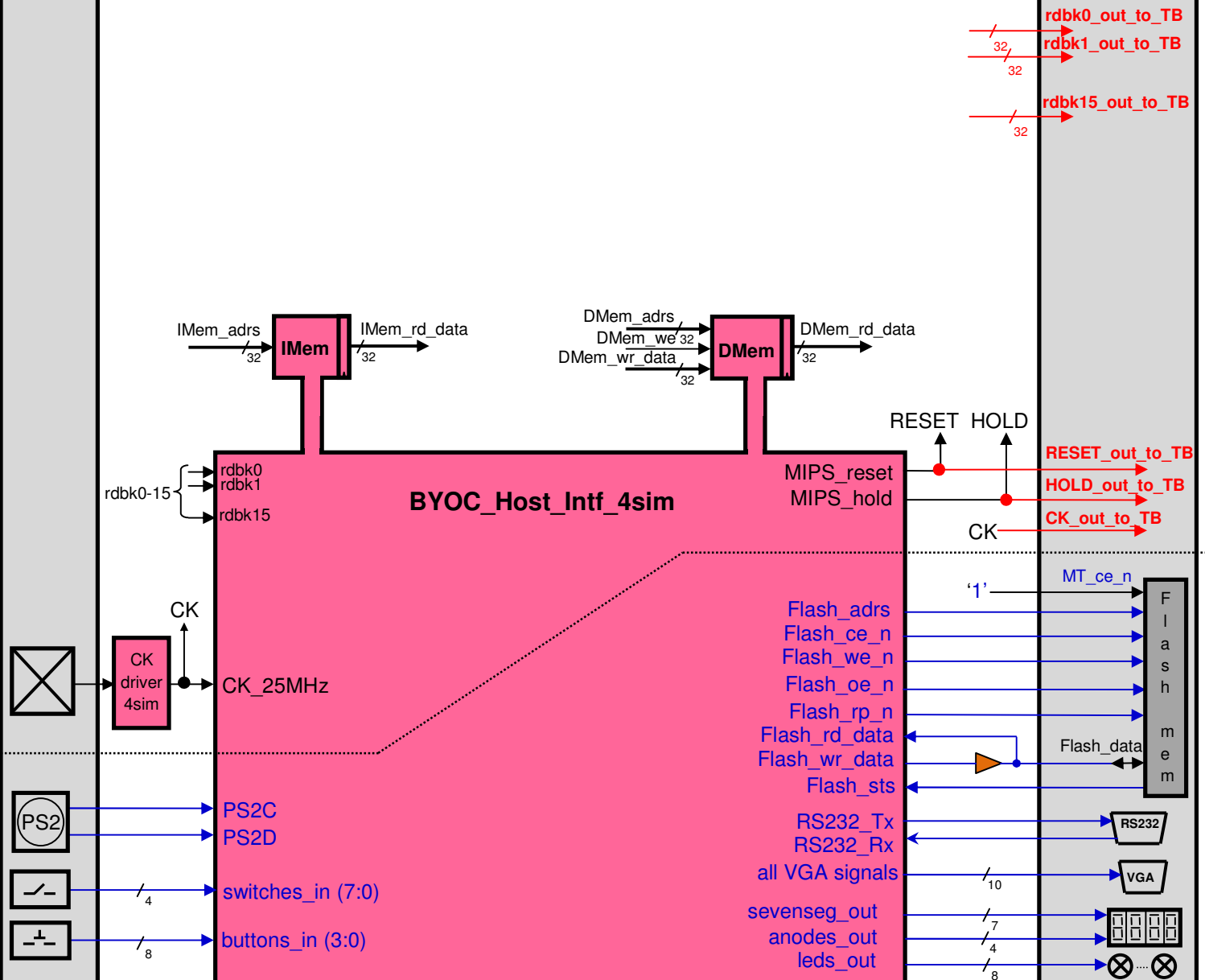


Fig. 2 – The infrastructure for simulating a design

Nexys2 board

BYOC.ucf

Your_design.vhd

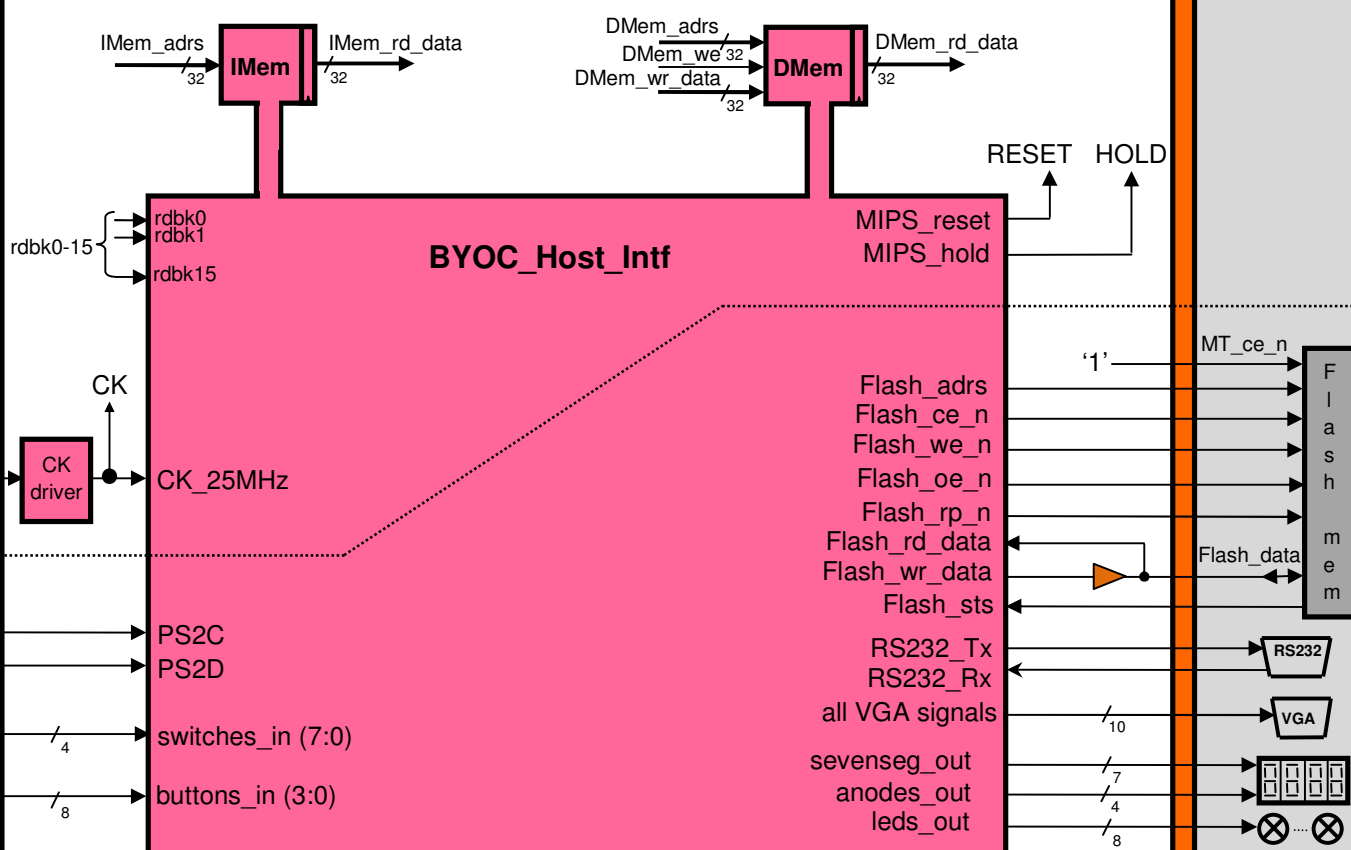


Fig. 3 – Implementing a design


the_Test_Bench.vhd

rdbk0_out_to_TB
rdbk1_out_to_TB
rdbk15_out_to_TB

RESET_out_to_TB

HOLD_out_to_TB

CK_out_to_TB



Nexys2 board

BYOC.ucf

Your_design.vhd

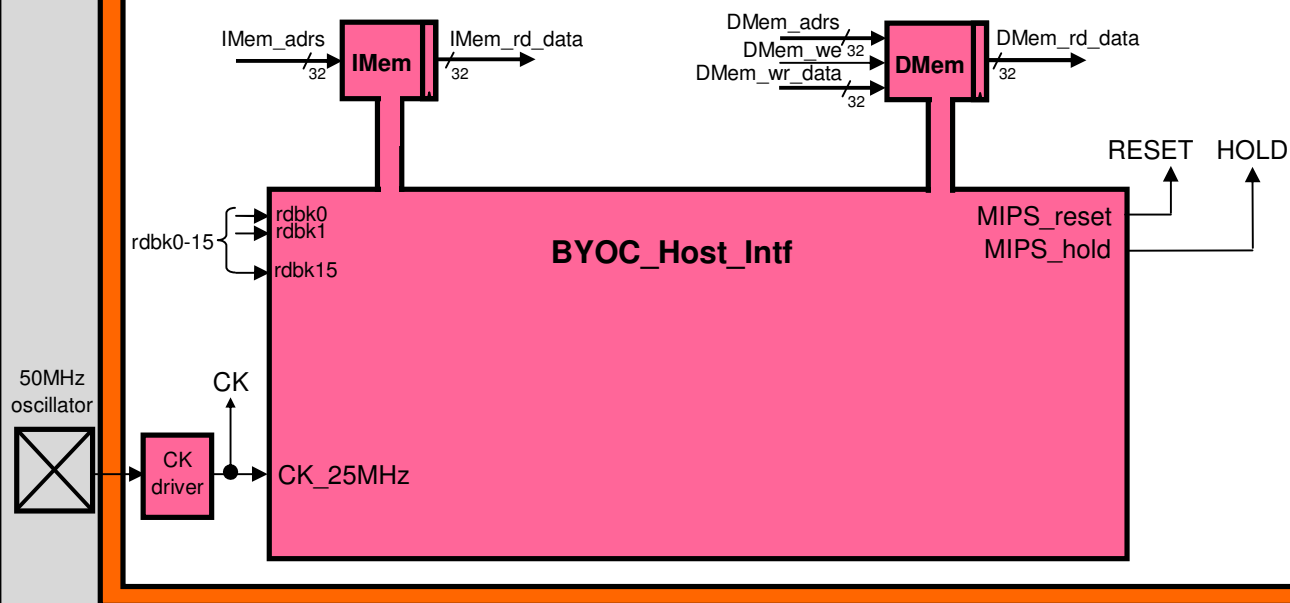


Fig. 5 – Implementing a design – a simplified drawing

When you want to convert your simulation design to its implementation version, you should:

- 1) Remove all TB signals (the red ones in Fig.4) and the **SIM_TB.vhd** file
- 2) Replace **Clock_Driver_4sim.vhd** with **Clock_Driver.vhd**
- 3) Replace **BYOC_Host_Intf_4sim.vhd** with **BYOC_Host_Intf.ngc**
- 4) Add the **BYOC.ucf** file

2. The Data Memory details

The Data Memory infrastructure has 3 parts. The first is the regular DMem part, in addresses 0x10000000-0x10000FFC, i.e. a 1Kx32bit memory. We can read from that memory when executing lw instructions and write to that memory when we execute sw instructions.

The second is another bulk of memory in addresses 0x20000000h – 0x20003FFC. This part of the memory is accessible from the MIPS CPU exactly like the first regular part. This part is also read constantly by a hidden part of the BYOC_Host_intf and displayed on a VGA monitor that can be connected to the Nexys2 board. That hidden infrastructure is therefore called the VGA controller.

The mapping of the memory data to the VGA screen is depicted in Fig. 6 below.

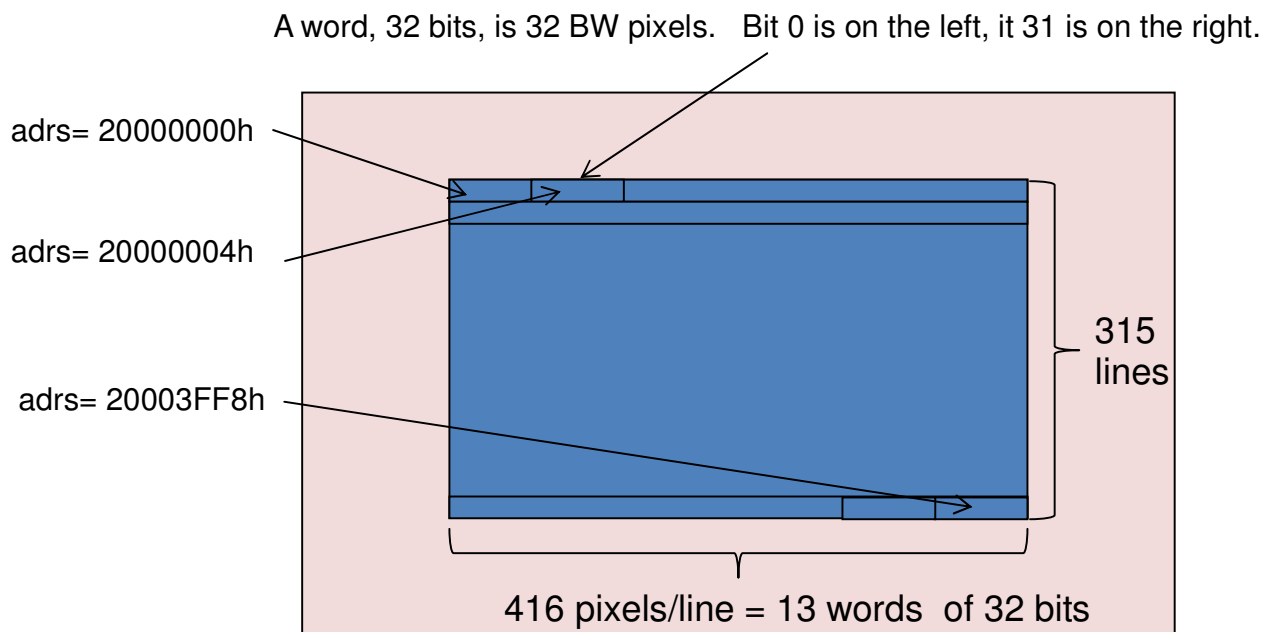


Fig. 6 – The mapping of DMem 2nd part to VGA screen

If we write 1 to address 0x20000000, we get a single white pixel at the top left of the 315x416 pixels area displayed by the BYOC VGA. If we write -1, which is 0xFFFFFFFF, to the same address we get a white 32 pixel horizontal line beginning at the top left of the display area.

The third part of the DMem is made of a few read and write addresses (0x30000000h - 0x300000FC) which control the KBD interface (allowing reading from the PS2 keyboard input of the Nexys2 board), and Flash Interface (allowing loading a program from a Flash memory residing on the Nexys2 board. For this the IMem includes also a “boot ROM” that knows how to copy from the flash into the IMem). In the course we only use the VGA and the KBD.

3. The top entity

Your top.vhd will have the top entity that, for simulation, will be always like this:

```
entity your_design_top is
port (
-- Host intf signals - Infrastructure signals [To be used by PC via RS232 or to/from Nexys2]
RS232_Rx      : in STD_LOGIC;
RS232_Tx      : out STD_LOGIC;
-- VGA signals
VGA_h_sync    : out  STD_LOGIC;
VGA_v_sync    : out  STD_LOGIC;
VGA_red0      : out  STD_LOGIC;
VGA_red1      : out  STD_LOGIC;
VGA_red2      : out  STD_LOGIC;
VGA_grn0      : out  STD_LOGIC;
VGA_grn1      : out  STD_LOGIC;
VGA_grn2      : out  STD_LOGIC;
VGA_blu1      : out  STD_LOGIC;
VGA_blu2      : out  STD_LOGIC;
--Flash Mem signals
MT_ce_n       : out   STD_LOGIC;-- '0' when accessing Nexys2 SDRAM --not used
Flash_adrs    : out   STD_LOGIC_VECTOR(23 downto 1);--Flash read/write address
Flash_ce_n    : out   STD_LOGIC;-- '0' when accessing Flash mem
Flash_we_n    : out   STD_LOGIC;-- '0' when writing to Flash mem
Flash_oe_n    : out   STD_LOGIC;-- '0' when reding from Flash mem
Flash_rp_n    : out   STD_LOGIC;-- '0' when reseting Flash mem
Flash_sts     : in    STD_LOGIC;-- '1' when Flash mem FSM is done
Flash_data    : inout STD_LOGIC_VECTOR(15 downto 0);--Data from/to Flash to/from IMem/DMem
--KBD signals
PS2C          : in    STD_LOGIC;-- PS2 keyboard clock
PS2D          : in    STD_LOGIC;-- PS2 keyboard data
--general signals
leds_out      : out   STD_LOGIC_VECTOR(7 downto 0);-- 7=Flash_stts,6-0=Host_Intf vesion
CK_50MHz      : in    STD_LOGIC;
buttons_in    : in    STD_LOGIC_vector(3 downto 0);--btn0=single clock ,btn3=manual reset
switches_in   : in    STD_LOGIC_VECTOR(7 downto 0);-- to be described later
sevensseg_out : out   STD_LOGIC_VECTOR (6 downto 0);-- to the 7 seg LEDs
anodes_out    : out   STD_LOGIC_VECTOR (3 downto 0); -- to the 7 seg LEDs
-- signals to be tested by the TB
CK_out_to_TB  : out STD_LOGIC;
RESET_out_to_TB : out STD_LOGIC;
HOLD_out_to_TB : out STD_LOGIC;
rdbk0_out_to_TB : out STD_LOGIC_VECTOR (31 downto 0);
rdbk1_out_to_TB : out STD_LOGIC_VECTOR (31 downto 0);
...
rdbk15_out_to_TB : out STD_LOGIC_VECTOR (31 downto 0) );
end your_design_top;
```

In simulation you will use the RED i/o-s. For Implementation you will remove these pins.

The structure of all top.vhd files will be as follows:

- HWx_top Entity declaration – with all Nexys2 used i/o pins. (x=2,4,5,6)
If this is a sim version then we will also have all signals to be sent to TB
- The Architecture of the entity:
 - Constants
 - List of the components used inside this entity:
 - Clock_Driver (HW2, HW4-6)
 - BYOC_Host_Intf (HW2, HW4-6)
 - Fetch_Unit (HW2, HW4-6)
 - GPR (HW4-6)
 - MIPS_ALU (HW4-6)
 -
 - Definition of signals
 - General signals (CK, RESET, HOLD, etc.)
 - Flash Memory signals
 - MIPS design signals:
 - IF phase
 - ID phase (regs, FFs and control)
 - EX phase (regs, FFs and control)
 - MEM phase (regs, FFs and control)
 - WB phase (regs, FFs and control)
 - Special rdbk vector signals
 - begin (beginning of the logical equations – processes)
 - Connecting the components used inside this entity:
 - Clock_Driver
 - BYOC_Host_Intf (including all rdbk signal)
 - Fetch_Unit
 - GPR
 - MIPS_ALU
 - All processes
 - Flash Memory connections
 - General signals creation (RESET only)
 - Your code:
 - IF equations
 - ID phase equations (regs, FFs and control)
 - EX phase equations (regs, FFs and control)
 - MEM phase equations (regs, FFs and control)
 - WB phase equations (regs, FFs and control)
 - Special rdbk phase equations
 - TB rdbk equations (in sim versions only)
- End of the Architecture

4. The expected reports

In most of the assignments you will be asked to submit a single zip file for the Simulation and implementation phases. It should have two directories/folders. The first is called **Simulation**, the 2nd is called **Implementation**.

In the **Simulation** folder you will have 3 sub-folder of:

- **Src_4sim** – here you put all of the *.vhd sources and the *.dat file (to be used by the TB)
- **Sim** – here you should have the HW4_4sim project created by the simulator you used
- **Docs** – Here you put your simulation report. The first few lines in the report will have your ID numbers (names are optional). In some cases we will ask to attach screen shots. In some cases you will have to answer questions. Follow the instructions that will be attached to the assignment document.

In the Implementation phase you will add 2 sub-folders to the **Implementation** folder.

These will be:

- **Src_4ISE** – here you put all of the *.vhd sources and the *.ucf file (and no TB file)
- **ISE** – here you should have the implementation project created by the Xilinx ISE SW.

This is the situation in HW2, HW4, HW5. In HW6 you have 3 separate simulation folders and 1 implementation folder, and in HW3 you have 2 simulation folders only and some extra docs.