BYOC course

Assignment #5

HW5_MIPS CPU

1. The HW5 MIPS CPU and its main components

In this assignment we add lw and sw instructions to the Rtype MIPS CPU we designed in HW4. This means we have to add the Data Memory (DMem) to our design. Following this we will have an almost complete MIPS CPU capable of performing Rtype, addi, j, beq, bne, lw and sw instructions. In our next & final assignment we will complete the CPU by adding jal, jr, lui and ori instructions and add forwarding to enhance the CPU performance.

The DMen we add is located in the Host_Intf component that includes the IMem and DMem and infrastructure allowing loading data into these memories.

Below we see a simplified drawing of the HW5_MIPS CPU we are going to build in this assignment.

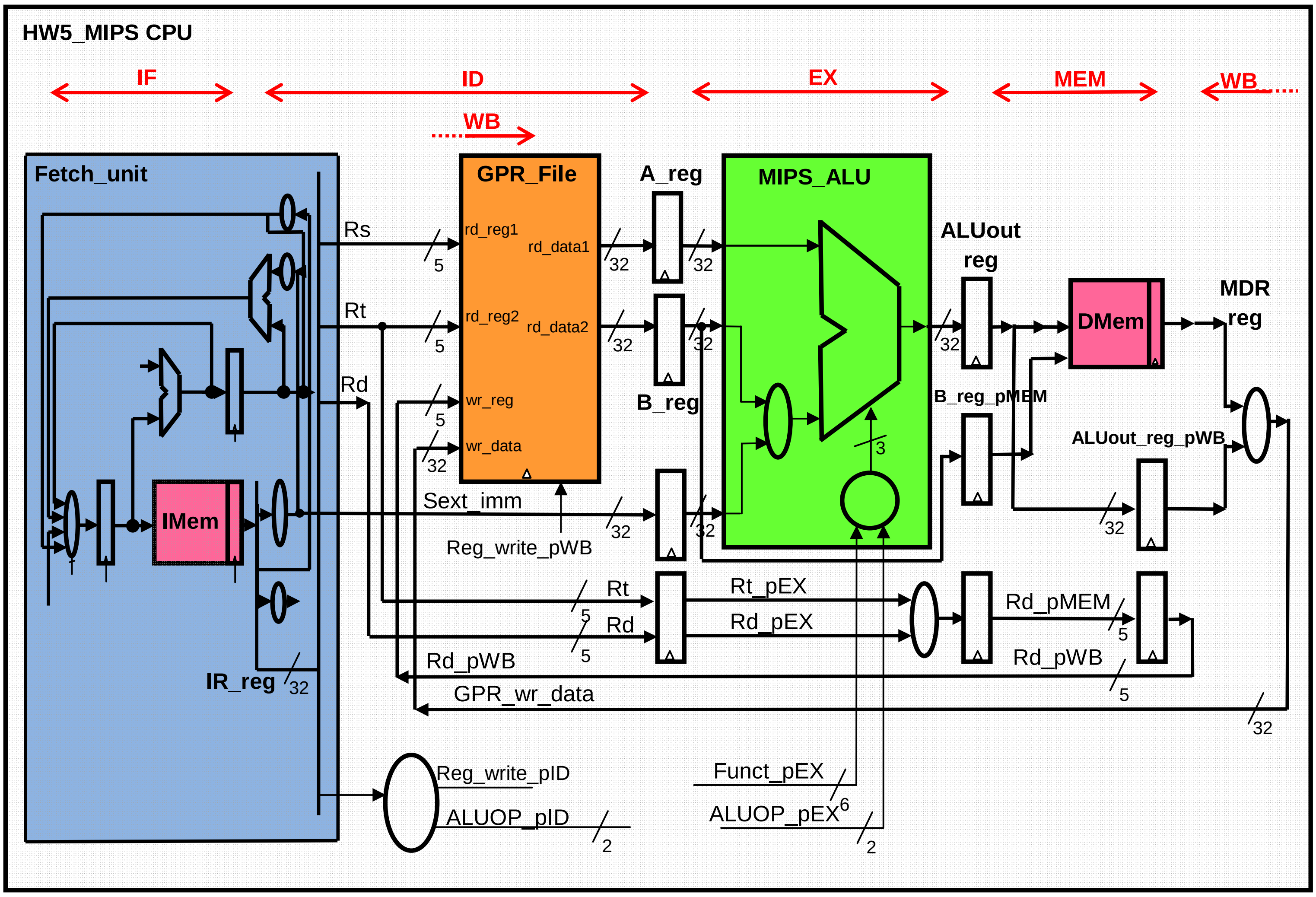

Fig. 1 – The HW5_MIPS CPU

A more accurate drawing includes the Host_Intf part – as depicted in Fig.2 below:

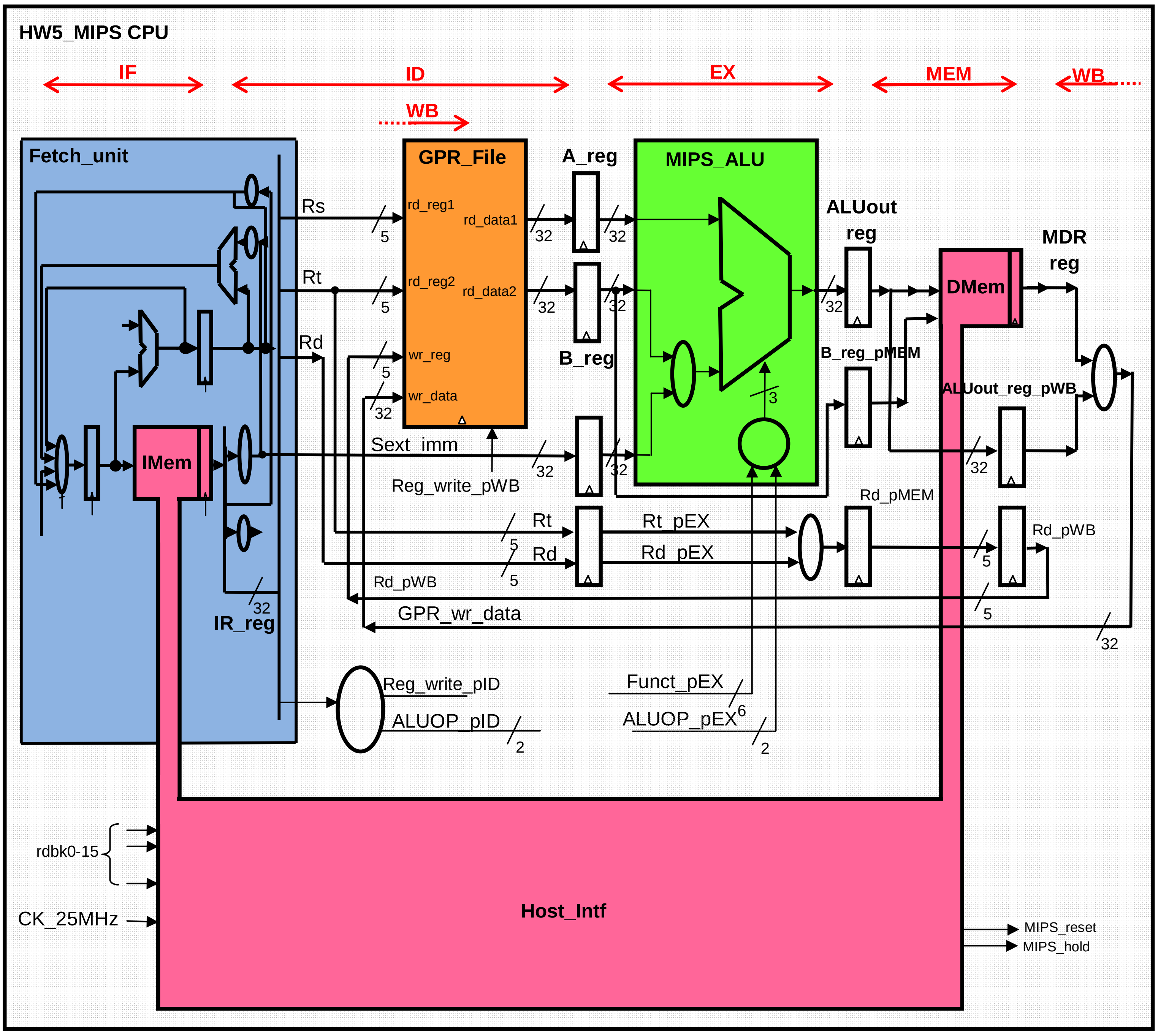

Fig. 2 – The HW5_MIPS CPU with the Host_Intf infrastructure

To your design, the only connections required are the RESET& HOLD signals, the IMem connections and the DMem connections. The rest of the Host_Intf connections will be given in the **HW5_MIPS_4sim-empty.vhd** or the **HW5_MIPD.vhd** files.

The Host_Intf component is similar to the one we used in HW4 in the simulation phase with the addition of the DMem signals. This is depicted in Fig. 3 below where all DMem signals are shown below the dashed line at the lower part of the figure. We divided the Host_Intf to 3 parts. The top one is the IMem, the bottom one is the DMem and the middle one is the infrastructure to be used late in the implementation phase. We left that part as in HW4. Later we will add more signals for the implementation phase of HW5.

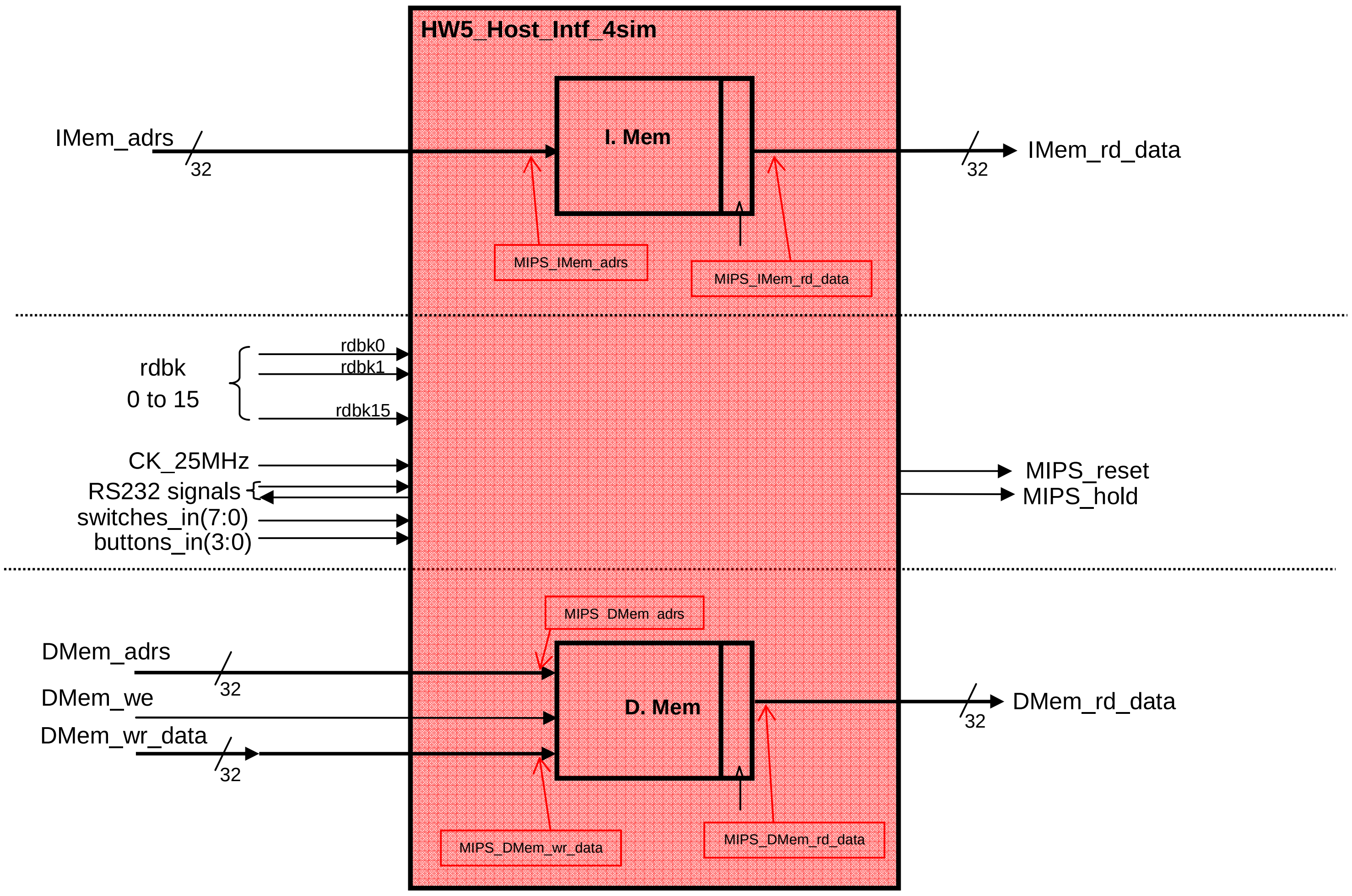

Fig. 3 – HW5_Host_Intf_4sim

In the simulation version we “load” the IMem memory with the desired program for testing the HW5_MIPS_4sim design.

Below we describe the actual work required.

a. DMem signals

The addition of the DMem signals is already done and these signals appear in the **HW5_MIPS_4sim-empty.vhd**. These signals are:

1. DMem_adrs – a 32 bit address signal to the DMem (coming from ALUout_reg).
2. DMem_rd_data – the 32 bit data read from the DMem (read from the address specified by DMem_adrs). This is after a register. I.e., it is actually the MDR data. It will be renamed to MDR_reg.
3. DMem_wr_data – 32 bit data to be written into the DMem to the address specified by DMem_adrs at the rising edge of the MIPS_ck if DMem_we is '1'.
4. DMem_we – a '1' means data will be written into the DMem at the rising edge of the MIPS_ck.

b. Names & definition of signals inside the HW5_MIPS CPU

In your design, you should use the exact signal names as were used in the Rtype MIPS CPU of HW4 and **add** the following signals using the exact signal names shown below:

ID additional signals

5. MemWrite – '1' when this is a sw instruction and we write into the DMem, '0' otherwise.
6. MemToReg – '1' when we read from memory, i.e., in lw instruction.

EX phase signals

7. MemWrite_pEX – MemWrite delayed by 1 clock cycle.
8. MemToReg_pEX – MemToReg delayed by 1 clock cycle.

MEM phase signals.

9. B_reg_pMEM – a 32 bit register receiving the B_reg signal (i.e., B_reg delayed by 1 CK cycle). This register has the data to be written into the DMem in sw instruction.
10. Rd_pMEM – the output of RegDest mux selecting to which register the CPU writes in the WB phase.
11. MemWrite_pMEM - MemWrite_pEX delayed by 1 clock cycle.
12. MemToReg_pMEM – MemToReg_pEX delayed by 1 clock cycle.
13. RegWrite_pMEM – RegWrite_pEX delayed by 1 clock cycle.

WB phase signals

14. MDR_reg- a 32 bit register that has the data read from the memory. This is a rename of the DMem_rd_data signal coming out of the **HW5_Host_Intf_4sim** component.
15. ALUout_reg_pWB - a 32 bit register that has the ALUour_reg data delayed by 1 CK cycle.
16. GPR_wr_data - a 32 bit signal that is the output of the MemToReg mux (selecting between MDR_reg and ALUout_reg_pWB).
17. Rd_pWB – Rd_pMEM delayed by 1 clock cycle.
18. MemToReg_pMEM – MemToReg_pEX delayed by 1 clock cycle
19. RegWrite_pWB – RegWrite_pMEM delayed by 1 clock cycle.

Add more signals according to your needs.

c. Names and definitions of output signals from HW5_MIPS CPU to the TB

You need to define all output signals coming out of the **HW5_MIPS_4sim** entity to be tested by the HW5_TB. These signals are the same as we used in **HW4_MIPS_4sim**:

1) CK_out_to_TB - a signal identical to the MIPS_ck "internal" signal
2) RESET_out_to_TB - a signal identical to the MIPS_reset "internal" signal
3) HOLD_out_to_TB - a signal identical to the MIPS_hold "internal" signal
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 – as detailed below.

In your **HW5_MIPS_4sim** design you should connect the rdbk signals as follows:

<u>ID signals:</u>		
rdbk0	=>	PC_plus_4_pID
rdbk1	=>	IR_pID,
rdbk2	=>	sext_imm_pID
rdbk3	=>	Rs, Rt, Rd, Funct (Rs= bits 28:24, Rt= bits 20:16, Rd= bits 12:8, Funct= bits 5:0)
rdbk4	=>	RegWrite, Rs_eq_Rt, MemWrite (Reg Write= bit 28, MemWrite= bit 24, Rs_eq_Rt=bit0)
<u>EX signals:</u>		
rdbk5	=>	ALUsrcB_pEX, ALUOP, Funct_pEX (ALUsrcB=bit 28, ALUOP= bits 9:8, Funct_pEX = bits5:0)
rdbk6	=>	A_reg,
rdbk7	=>	B_reg,
rdbk8	=>	sext_imm_reg,
rdbk9	=>	ALU_output,
<u>MEM signals:</u>		
rdbk10	=>	ALUOUT_reg
rdbk11	=>	B_reg_pMEM
<u>MEM & WB control signals</u>		
rdbk12	=>	MemWrite_pMEM (bit31), MemToReg_pMEM (bit28), RegWrite_pMEM (bit24), Rd_pMEM (bits 20:16), MemToReg_pWB (bit12), RegWrite_pWB (bit8), Rd_pWB (bits 4:0)
<u>WB signals:</u>		
rdbk13	=>	MDR_reg
rdbk14	=>	ALUOUT_reg_pW
rdbk15	=>	GPR_wr_data

The TB will compare the expected data of these signals to the actual result of the simulation and will tell you where the errors are. It will use a file called **HW5_TB_data.dat** containing the expected results. Since you will use these signals in the implementation phase, you should also connect them to the Host Interface.

In Fig.4 below we describe the simulation project.

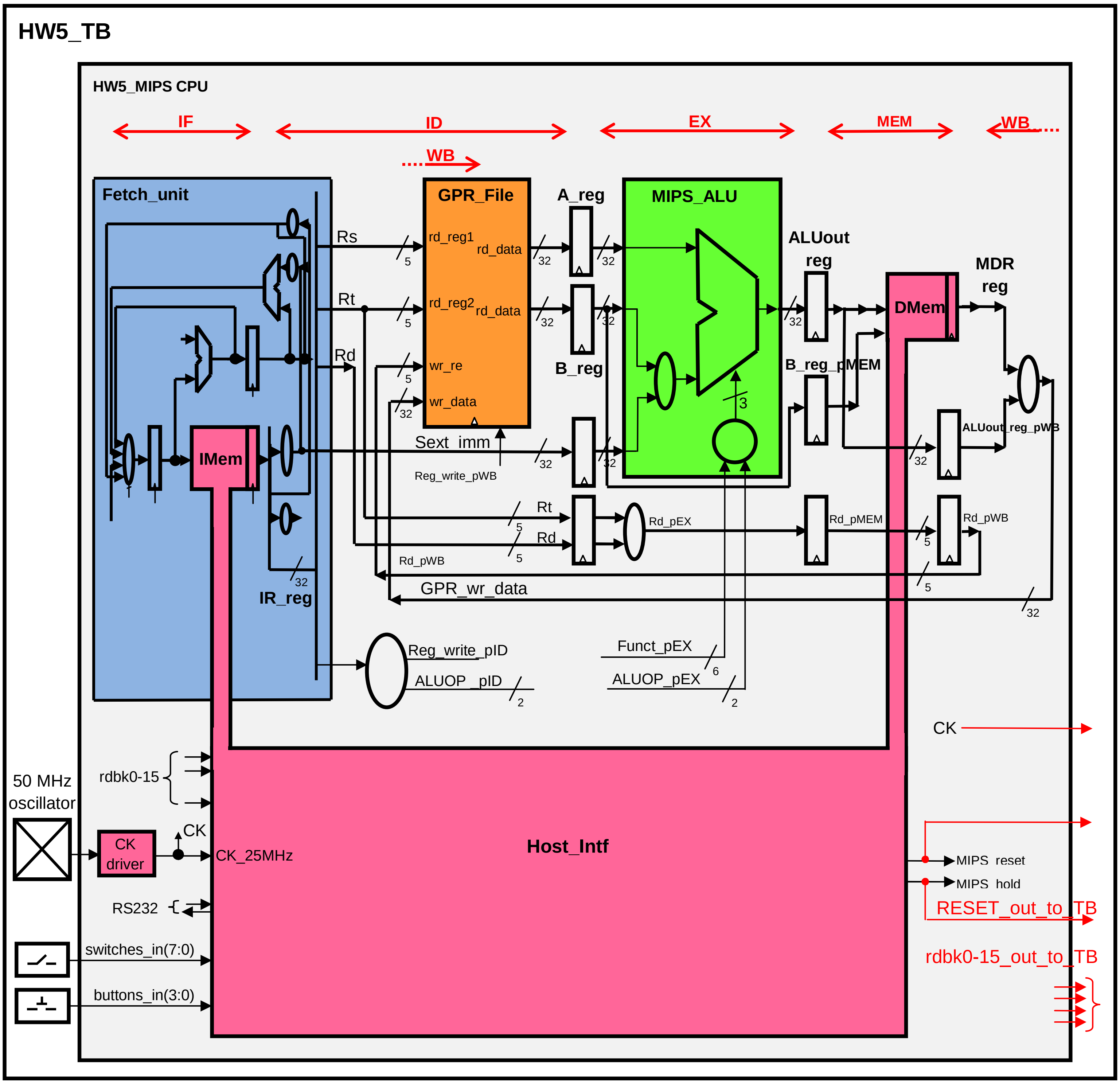

Fig. 4 – The entire simulation system

D. Description of the HW5_MIPS_4sim project

You need to define all signals in your design (actually, we already did that for you in the **HW5_MIPS_4sim-empty.vhd** and **HW5_Host_Intf_4sim.**vhd files). Then, you should write the equations of all signals and registers and connect all of the components. Then you should run the simulation.

The files we will use to run the simulation are:

1) **HW5_MIPS_4sim.vhd** – This is your design of HW5. It uses your designs of the Fetch_Unit, GPR, MIPS_ALU and the pre-prepared HW5_Host_Intf_4sim.
2) **Fetch_Unit.vhd** - The Fetch Unit you designed in HW2 after the modifications of HW4.
3) **GPR.vhd** – your GPR File design you designed in HW3.
4) **dual_port_memory.vhd** – part of the GPR File you designed in HW3.
5) **MIPS_ALU.vhd** – your MIPS_ALU design prepared in HW3.
6) **HW5_Host_Intf_4sim.vhd** – The pre-prepared component including the IMem, DMem and “pre-loaded” program and data. This component also creates the reset & hold signals to the rest of the HW5_MIPS CPU.
7) **HW5_TB._for_students_no_compare.vhd** - The TB vhd file prepared in advance.
8) **HW5_TB**.dat – this is a data file prepared in advance that has the expected TB values. It is read by the HW5_TB and used to compare the actual simulation results to the expected ones.

2) Simulation report

You should 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 for simulation
- **Sim** – here you should have the HW5_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). See the instructions below for the rest of the simulation report.

The first part of the program we have in the IMem in HW5 simulation (addresses 400000h to 4001A4h) is very similar to the one you had in HW4 and is meant to check that you did not mess anything during the changes you did. That part is tested by the TB. The HW5_TB.dat file contains test data only for this part of the IMem program.

The rest of the IMem program, from 4001A8h till the end (400330h), is meant to test the writing and reading from the DMem. The program is given in Appendix A at the end of this document.

In order to test your design you need to look at the simulated waveforms and decide whether it is OK or not. You should run the simulation for 10.5us (10500ns). In the doc file you need to attach screen captures describing this part of the simulation you made, as detailed in 3.1.

3.1) The listed below signals should be presented in the screen capture you need to attach to your report. Show clock cycles 200-220 (following the end of the reset pulse, find i=200-220) and make the values of all signals readable. For this you will probably need to show clocks 200-210 and 210-220 separately. These are the signals that can help you in "testing" the DMem.

Note the some of these signals are inside the Host Intf :

1. CK
2. RESET
3. HOLD
4. i (the serial no. of the clock cycle)
5. MIPS_DMem_adrs
6. MIPS_DMem_wr_data
7. MIPS_DMem_we
8. MIPS_DMem_rd_data
9. DMem_reg0
10. DMem_reg1
11. DMem_reg2
12. DMem_reg3
13. DMem_reg4
14. PC_reg
15. IR_reg

I=200-220 means CK cycles from 8560 ns to 9400 ns.

3.2) Explain what happens, i.e., what do we see here.

In that doc file you need also to answer the following questions:

3.3) What is the latency of Rtype instruction? How many nop-s should be inserted between two consecutive Rtype instructions if the 2nd one uses the result of the 1st one?

3.4) Explain the limitation of beq that tests a register that is calculated by Rtype instruction. As an example, translate the following C if statement: for (i=0;i<10;i++) { … }
where i is register $3.

3.5) Are there any other limitations due to the pipeline structure in the instructions we implemented (Rtype, addi, beq, bne, j, lw, sw)? How can we overcome these limitations (e.g., by adding nop-s)?

Later, 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 HW5_MIPS project created by the Xilinx ISE SW.

3) HW5_MIPS CPU – implementation

After a successful simulation we want to implement the design on the Nexys2 board. A few changes are required. We use the **HW5_MIPS.vhd** file instead of the **HW5_MIPS_4sim.vhd** file. We can rename the **HW5_MIPS_4sim.vhd** to **HW5_MIPS.vhd** and then, remove all the signals that were outputted to the TB (see 1c above). These are not required anymore. Then we should add new signals that will be connected to the Nexys2 board functions. These are described below. [Or, instead that, you could use the pre-prepared **HW5_MIPS-empty.vhd** file which includes all of the new signals, and copy your VHDL code into the appropriate location inside that file].

The new signals are added since we want now to use the full BYOC_Host_intf device that has additional features we could not use so far. Let's discuss the features of the BYOC_Host_intf we use in this assignment.

The Data Memory we add in this assignment has 3 parts. The first is the regular DMem part, in addresses 0x10000000-0x10000FFC, i.e. a 1Kx32bit memory. Very similar to the one we "used" in the simulation part.

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. 5 below.

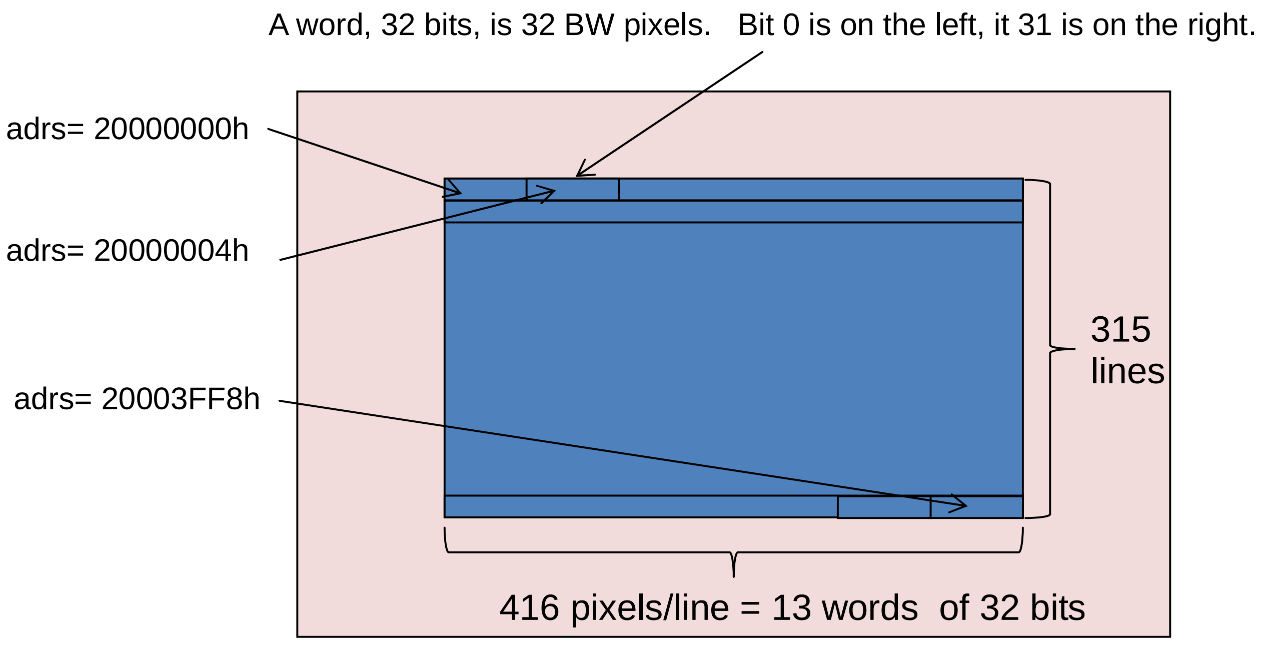
A word, 32 bits, is 32 BW pixels.   Bit 0 is on the left, it 31 is on the right.
adrs= 20000000h
adrs= 20000004h
adrs= 20003FF8h
315 lines
416 pixels/line = 13 words of 32 bits

Fig. 5 – 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 c copy from the flash into the IMem). In HW5 we will use the VGA part and the KBD part only.

The Host_Intf therefore includes the DMem, the VGA controller, the KBD intf and Flash intf (and 7seg LEDs). In Fig. 6 below we specified all new signals to be connected to the KBD, the VGA monitor, the 7 seg LEDS and to the Flash memory on the Nexys2 board. All new signals (compared to Fig. 3) are in blue.

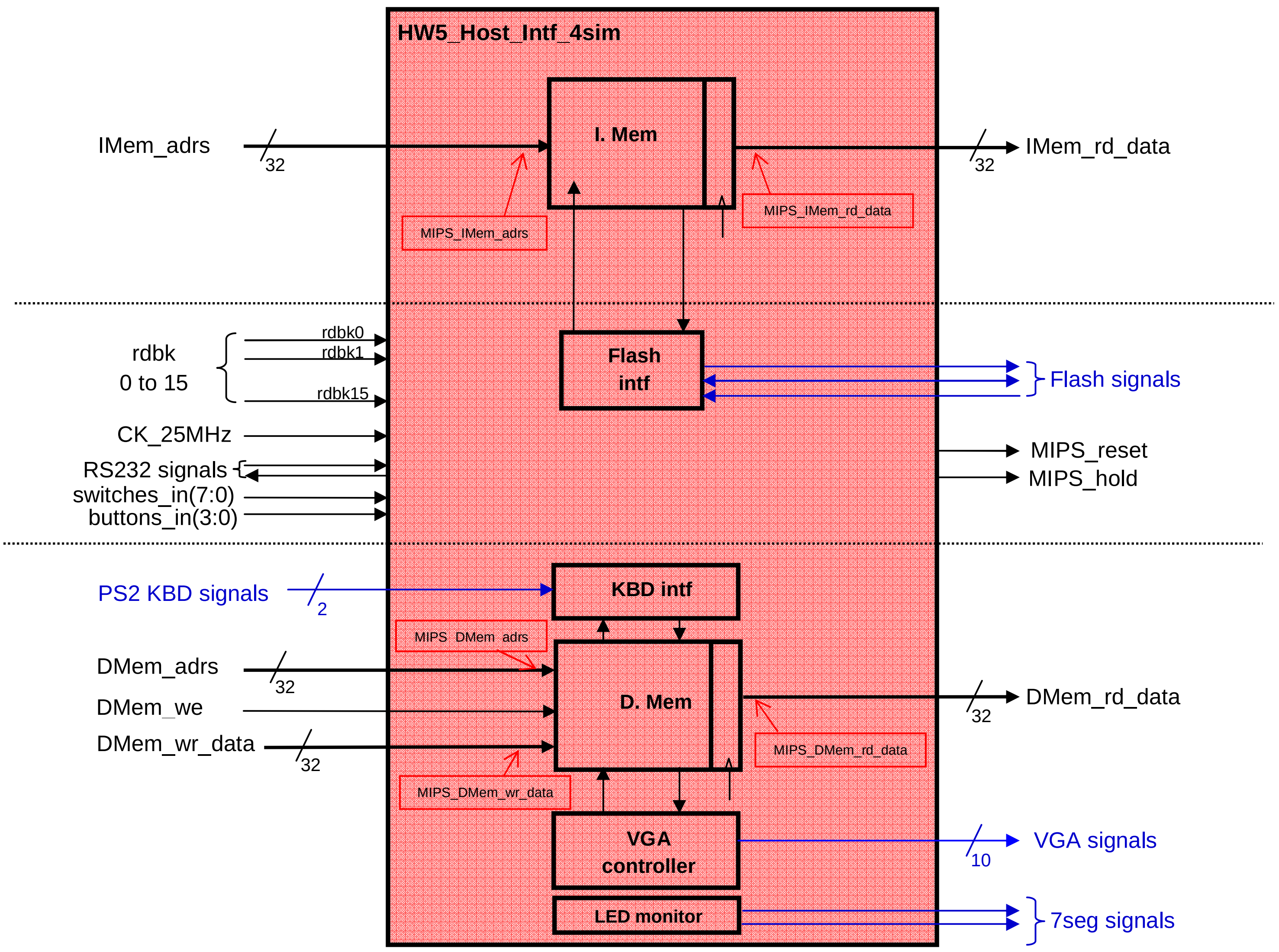

Fig. 6 – BYOC_Host_Intf.ngc

Until now we always “wrapped” the **BYOC_Host_Intf.ngc** with some vhd file (e.g., **HW4_Host_intf.vhd**) in order to hide all these extra signals. Now we get rid of the extra “wrap” and use the **BYOC_Host_Intf.ngc** as is.

We will connect these new signals to i/o pins of **HW5_MIPS.vhd**, the top file of our design, so they could be connected to the Nexys2 board. For that we need to define the **HW5_MIPS** entity as follows (the new signals are shown in blue):

```
entity HW5_MIPS 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=MIPS_ck,5-0=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
sevenseg_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
      );
end HW5_MIPS;
```

Most of these new signals we added to the **HW5_MIPS** entity are coming directly from the **BYOC_Host_intf** entity. Thus these signals should be transferred from **the BYOC_Host_intf** input and output pins to the input and output pins of the **HW5_MIPS** entity.

There are special cases. The case of the Flash_data pins of the **HW5_MIPS** is a special case. These pins are ***inout***. They are used for input of the Flash_rd_data and output of the Flash_wr_data. The equations that connect the Flash_data vector signal to the Flash_wr_data and Flash_wr_data vector signals coming from the **BYOC_Host_intf** entity are already included in the **HW5_MIPS-empty.vhd** file.

Another signal that disconnects a SDRAM device sitting on the Nexys2 board in parallel to the Flash memory is the MT_ce_n signal. We forced it to be ‘1’.

We describe all these connections in Fig.6 below.

To summarize, the change we need to do in the **HW5_MIPS** for implementation is to replace the component called **HW5_Host_intf_4sim** with a “similar” one called **BYOC_Host_intf**, and connect almost all of the new signals to new pins of the **HW5_MIPS**. Most of the new pins, carrying signals to be connected to the Nexys2 board, will be connected to the new pins of the **HW5_MIPS** entity. Some of them will require bidir “drivers”.

In Fig. 7 below we see also the connections to the final connectors in the Nexys2 board. For example, we see the PS/2 keyboard connector, the RS232 connector, the VGA D-type connector. In the actual board there is some electronics circuitry between the FPGA pins and these connectors. This is not really important for us since it does not change the functionality and we understand the reason for these connections.

There are also other devices such as the 4 buttons and 8 switches we use as inputs to control the BYOC_Host_intf (for example sw7 on means single ck only, and sw4 on means reset will start the PC form 0x400000 while sw4 off means reset will start the PC form 0x000000 – the boot ROM). And output devices such as the 8 green LEDs or the 4 Seven-Segment LED digits. We also have as inputs the 50 MHz oscillator residing on the Nexys2 board.

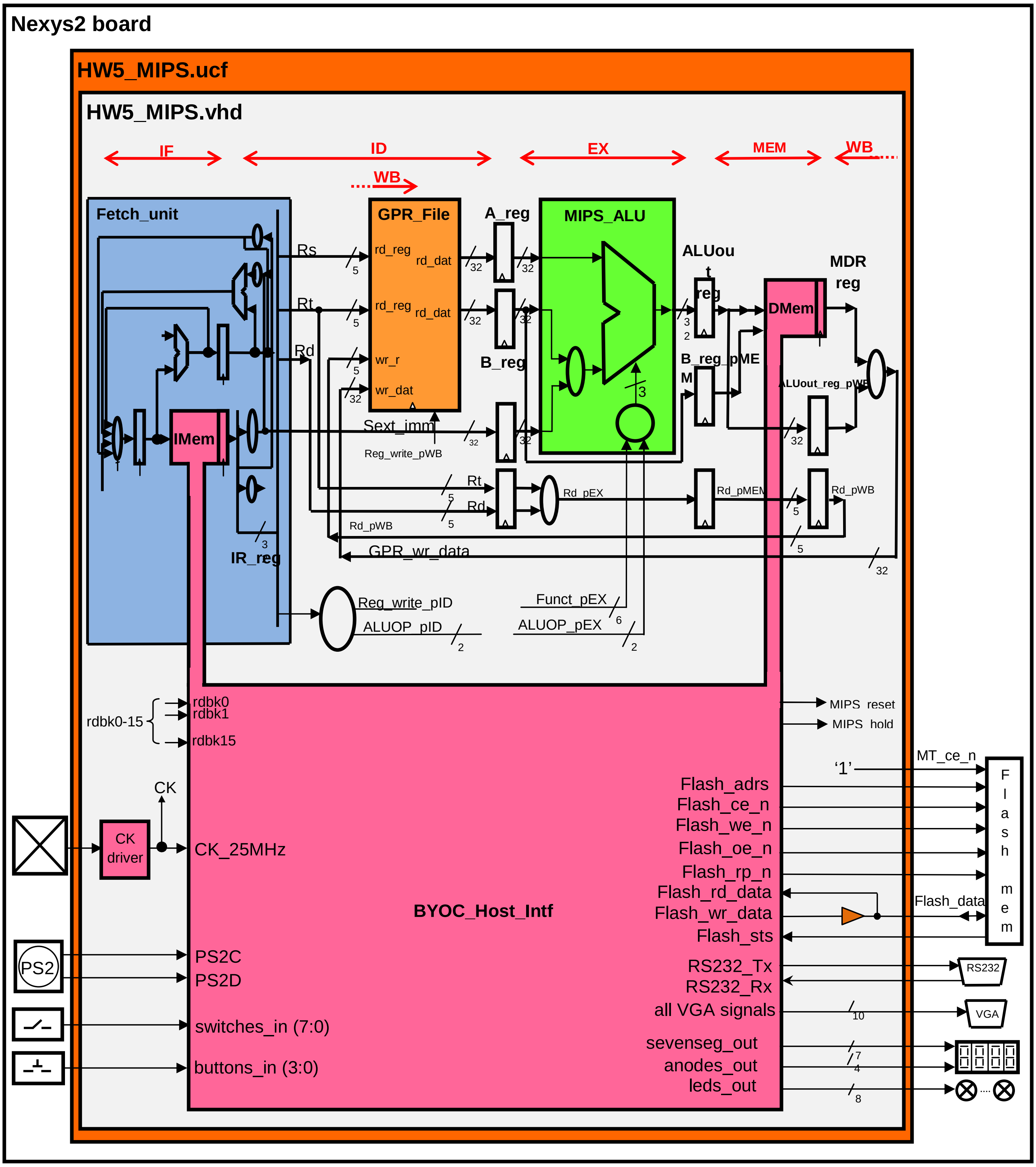

Fig. 7 – The entire system

The files we will use to implement the design on the Nexys2 board are:

1) **HW5_MIPS.ucf** - The file listing which signal are connected to which FPGA pins in the Nexys2 board.
2) **HW5_MIPS.vhd** – This is your design of HW5. It uses the Fetch_Unit, GPR, MIPS_ALU and the BYOC_Host_Intf components and all the signals described in 1b above.
3) **Fetch_Unit.vhd** - The Fetch Unit you prepared in HW2 after the modifications of HW4.
4) **GPR.vhd** – your GPR File design you prepared in HW3.
5) **dual_port_memory.vhd** – part of the GPR File design you prepared in HW3.
6) **MIPS_ALU.vhd** – your MIPS_ALU design you prepared in HW3.
7) **BYOC_Host_Intf.ngc** – This prepared component is the Host Interface. It includes the infrastructure interfacing to the PC allowing us to load programs into IMem and data into DMenm and read feedback signals in single clock mode. It also has the VGA controller and the KBD and Flash interfaces. We give that component in the form of a single netlist file, which is an already compiled version of the vhd files forming the BYOC_Host_interface.

To allow an easy debugging you should connect the rdbk signals to the BYOC_Host_intf as follows [same as in the simulation part]:

ID signals:

rdbk0	=>	PC_plus_4_pID
rdbk1	=>	IR_pID,
rdbk2	=>	sext_imm_pID
rdbk3	=>	Rs, Rt, Rd, Funct (Rs= bits 28:24, Rt= bits 20:16, Rd= bits 12:8, Funct= bits 5:0)
rdbk4	=>	RegWrite, Rs_eq_Rt, MemWrite (Reg Write= bit 28, MemWrite= bit 24, Rs_eq_Rt=bit0)

EX signals:

rdbk5	=>	ALUsrcB_pEX, ALUOP, Funct_pEX (ALUsrcB=bit 28, ALUOP= bits 9:8, Funct_pEX = bits5:0)
rdbk6	=>	A_reg,
rdbk7	=>	B_reg,
rdbk8	=>	sext_imm_reg,
rdbk9	=>	ALU_output,

MEM signals:

rdbk10	=>	ALUOUT_reg
rdbk11	=>	B_reg_pMEM

MEM & WB control signals

rdbk12	=>	MemWrite_pMEM (bit31), MemToReg_pMEM (bit28), RegWrite_pMEM (bit24), Rd_pMEM (bits 20:16), MemToReg_pWB (bit12), RegWrite_pWB (bit8), Rd_pWB (bits 4:0)

WB signals:

rdbk13	=>	MDR_reg
rdbk14	=>	ALUOUT_reg_pW
rdbk15	=>	GPR_wr_data

So we'll run that the BYOCInterface SW and load the IMem. Then run the circuit in a single ck mode and check that the reading we see at the points we "hooked" to the rdbk signals are as what we expect.

The file we want to load into the IMem is called "**HW5_rect1.txt**". The file itself includes all the information required in order to load it into the IMem and switch to a single ck mode. Following the loading, we can run in single ck mode and see the readback values on the PC screen after each clock. The HW5_rect1 program is given in Appendix B at the end of this document.

This time we would like to connect a VGA screen to the Nexys2 board.
What happens after 80 CKs? What happens when you press the RUN button?

4) Implemetation report

You should 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 **Implementation** directory you should have 2 directories:

- **Src_4ISE** – here you put all of the *.vhd sources and the *.ucf file (and no TB file)
- **ISE** – here you should have the HW5 project created by the Xilinx ISE SW.

As part of completing this part of the course you will have to show me how you run the design on the Nexys2 board in the lab. And maybe answer some questions.

Enjoy the assignment !!

At the end of this assignment you will have a real CPU capable of running simple programs that also access the Data Memory. Only a few instructions are missing. We will add them and improve the performance in our next assignment.

<u>Appendix A</u> – IMem program for simulation – 2nd part

Address	label	Inst.	Rd/Rt	Rs	Rt	Imm/label	# remark	Inst. code
4001A8	cont:	addi	$1	$0		1000h	# prep4 sw/lw test	20011000
4001AC		nop						00000000
4001B0		addi	$2	$0		5555h		20025555
4001B4		addi	$3	$0		AAAAh		2003AAAA
4001B8		add	$1	$1	$1		# 1 add once	00210820
4001BC		nop						00000000
4001C0		nop						00000000
4001C4		nop						00000000
4001C8		add	$1	$1	$1		# 2 add for the 2nd time	00210820
4001CC		nop						00000000
4001D0		nop						00000000
4001D4		nop						00000000
4001D8		add	$1	$1	$1		# 3	00210820
4001DC		nop						00000000
4001E0		nop						00000000
4001E4		nop						00000000
4001E8		add	$1	$1	$1		# 4	00210820
4001EC		nop						00000000
4001F0		nop						00000000
4001F4		nop						00000000
4001F8		add	$1	$1	$1		# 5	00210820
4001FC		nop						00000000
400200		nop						00000000
400204		nop						00000000
400208		add	$1	$1	$1		# 6	00210820
40020C		nop						00000000
400210		nop						00000000
400214		nop						00000000
400218		add	$1	$1	$1		# 7	00210820
40021C		nop						00000000
400220		nop						00000000
400224		nop						00000000
400228		add	$1	$1	$1		# 8	00210820
40022C		nop						00000000
400230		nop						00000000
400234		nop						00000000
400238		add	$1	$1	$1		# 9	00210820
40023C		nop						00000000
400240		nop						00000000

400244	nop						00000000
400248	add	$1	$1	$1		# 10	00210820
40024C	nop						00000000
400250	nop						00000000
400254	nop						00000000
400258	add	$1	$1	$1		# 11	00210820
40025C	nop						00000000
400260	nop						00000000
400264	nop						00000000
400268	add	$1	$1	$1		# 12	00210820
40026C	nop						00000000
400270	nop						00000000
400274	nop						00000000
400278	add	$1	$1	$1		# 13	00210820
40027C	nop						00000000
400280	nop						00000000
400284	nop						00000000
400288	add	$1	$1	$1		# 14	00210820
40028C	nop						00000000
400290	nop						00000000
400294	nop						00000000
400298	add	$1	$1	$1		# 15	00210820
40029C	nop						00000000
4002A0	nop						00000000
4002A4	nop						00000000
4002A8	add	$1	$1	$1		# 16 - the 16th addition	00210820
4002AC	nop						00000000
4002B0	nop						00000000
4002B4	nop						00000000
4002B8	sw	$2	$1		0	# now $1=???	AC220000
4002BC	sw	$3	$1		4		AC230004
4002C0	lw	$4	$1		0		8C240000
4002C4	lw	$5	$1		4		8C250004
4002C8	nop						00000000
4002CC	nop						00000000
4002D0	nop						00000000
4002D4	add	$5	$5	$4			00A42820
4002D8	nop						00000000
4002DC	nop						00000000
4002E0	nop						00000000
4002E4	addi	$5	$5		1		20A50001
4002E8	nop						00000000
4002EC	nop						00000000

4002F0		nop					00000000
4002F4		bne	$5	$0	errlp		14A00007
4002F8		nop					00000000
4002FC		nop					00000000
400300		nop					00000000
400304	endlp:	j			endlp		081000C1
400308		nop					00000000
40030C		nop					00000000
400310		nop					00000000
400314	errlp:	j			errlp		081000C5
400318		nop					00000000
40031C		nop					00000000
400320		nop				end of errlop	00000000
400324	endlp2:	j			endlp2		081000C9
400328		nop					00000000
40032C		nop					00000000
400330		nop				end of program	00000000

Appendix B – IMem program for implementation

Address in Hex	label	instruction	Rd/ Rt	Rs/ Rt	Rt	Imm/ label	remark	MIPS Hex ode
400000	main:	addi	$1	$0		64		20010040
400004		addi	$2	$0		2000h		20022000
400008		nop						00000000
40000C		nop						00000000
400010		nop						00000000
400014		add	$2	$2	$2			00421020
400018		nop						00000000
40001C		nop						00000000
400020		nop						00000000
400024		add	$2	$2	$2			00421020
400028		nop						00000000
40002C		nop						00000000
400030		nop						00000000
400034		add	$2	$2	$2			00421020
400038		nop						00000000
40003C		nop						00000000
400040		nop						00000000
400044		add	$2	$2	$2			00421020
400048		nop						00000000
40004C		nop						00000000
400050		nop						00000000
400054		add	$2	$2	$2			00421020
400058		nop						00000000
40005C		nop						00000000
400060		nop						00000000
400064		add	$2	$2	$2			00421020
400068		nop						00000000
40006C		nop						00000000
400070		nop						00000000
400074		add	$2	$2	$2			00421020
400078		nop						00000000
40007C		nop						00000000
400080		nop						00000000
400084		add	$2	$2	$2			00421020
400088		nop						00000000
40008C		nop						00000000
400090		nop						00000000
400094		add	$2	$2	$2			00421020
400098		nop						00000000

40009C		nop						00000000
4000A0		nop						00000000
4000A4		add	$2	$2	$2			00421020
4000A8		nop						00000000
4000AC		nop						00000000
4000B0		nop						00000000
4000B4		add	$2	$2	$2			00421020
4000B8		nop						00000000
4000BC		nop						00000000
4000C0		nop						00000000
4000C4		add	$2	$2	$2			00421020
4000C8		nop						00000000
4000CC		nop						00000000
4000D0		nop						00000000
4000D4		add	$2	$2	$2			00421020
4000D8		nop						00000000
4000DC		nop						00000000
4000E0		nop						00000000
4000E4		add	$2	$2	$2			00421020
4000E8		nop						00000000
4000EC		nop						00000000
4000F0		nop						00000000
4000F4		add	$2	$2	$2			00421020
4000F8		nop						00000000
4000FC		nop						00000000
400100		nop						00000000
400104		add	$2	$2	$2			00421020
400108		nop						00000000
40010C		nop						00000000
400110		nop						00000000
400114		addi	$2	$2		18h		20420018
400118		addi	$3	$0		-1		2003FFFF
40011C		nop						00000000
400120		nop						00000000
400124		nop						00000000
400128	drawlp:	sw	$3	$2		0		AC430000
40012C		addi	$2	$2		52		20420034
400130		addi	$1	$1		-1		2021FFFF
400134		nop						00000000
400138		nop						00000000
40013C		nop						00000000
400140		bne	$1	$0		drawlp		1420FFF9
400144		nop						00000000

400148		nop			00000000
40014C		nop			00000000
400150	end:	j	end		08100054
400154		nop			00000000
400158		nop			00000000
40015C		nop			00000000
400160		nop			00000000