BYOC course

Assignment #5

<u>HW5 MIPS CPU</u>

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 inside the BYOC_Host_Intf component that includes infrastructure allowing loading data into the IMem and DMem memories.

Below we see a simplified drawing of the HW5 MIPS CPU we are going to build in this assignment.

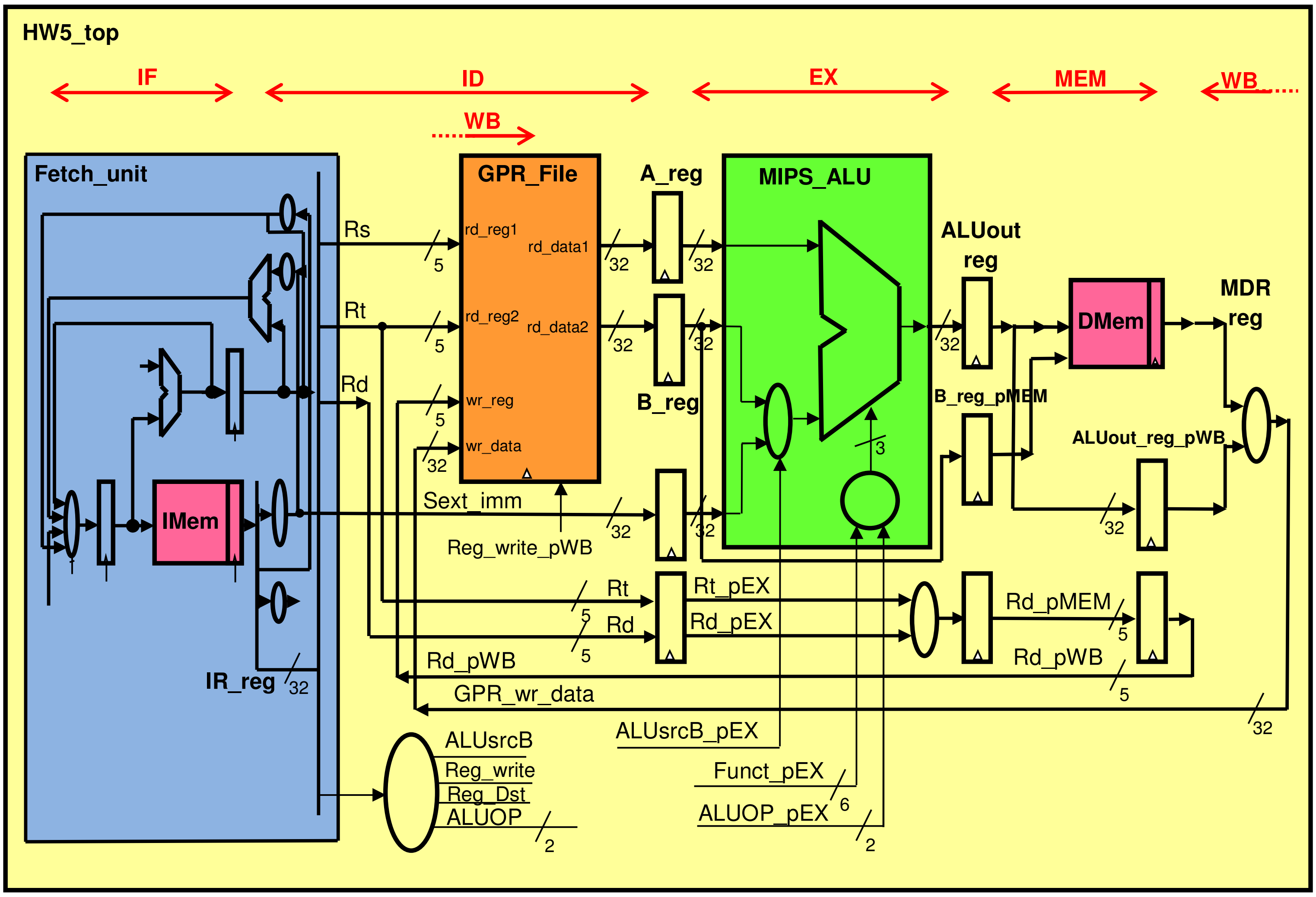

Fig. 1 – The HW5 MIPS CPU

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

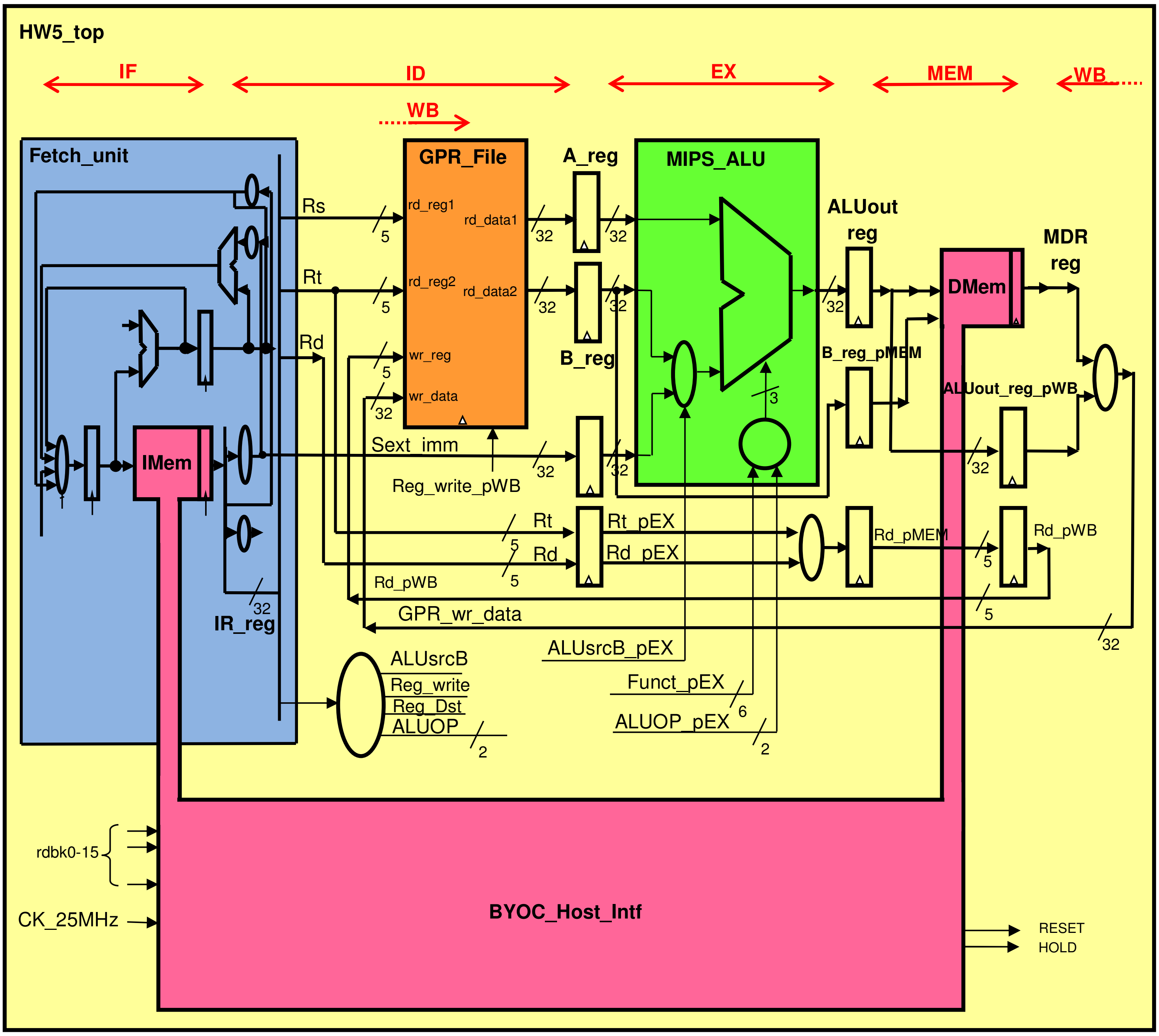

Fig. 2 – The HW5_MIPS CPU with the Host_Intf infrastructure

To your **HW5_top** design, the only required connections to the **BYOC_Host_Intf** are the RESET & HOLD signals, the IMem connections and the DMem connections and the rdbk0-15 coming from the signals we want to check during implementation. These and the rest of the **BYOC_Host_Intf** connections are already given in the **HW5_top_4sim.empty** file.

You should rename that file to **HW5_top_4sim.vhd** and add the necessary equations – all based on HW4 design). Below we describe the actual work required.

a. Connecting the DMem

The DMem signals are already connected in the **HW5_top_4sim.empty** file. There is no need to add any more DMem connections, but you need to understand these connections:

1. MIPS_DMem_adrs – a 32 bit address signal of DMem is connected to ALUout_reg signal.
2. MIPS_DMem_rd_data – the 32 bit data read from the DMem (we read from the address specified by MIPS_DMem_adrs). This is after a register. I.e., it is actually the MDR data. It is directly connected to the HW5_top signal called MDR_reg.
3. MIPS_DMem_wr_data – 32 bit data to be written into the DMem to the address specified by MIPS_DMem_adrs at the rising edge of the CK if MIPS_DMem_we is '1'. It is connected to (i.e., driven by) the B_reg_pMEM signal of HW5_top.
4. MIPS_DMem_we – a '1' means data will be written into the DMem at the rising edge of the CK. This is driven by the MemWrite_pMEM signal of HW5_top.

b. Names & definition of signals inside the HW5_top 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 **BYOC_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_pWB – MemToReg_pMEM delayed by 1 clock cycle
19. RegWrite_pWB – RegWrite_pMEM delayed by 1 clock cycle.

Add more signals according to your needs.

Fig. 3 below shows the control signals scheme of CPU we build in HW5.

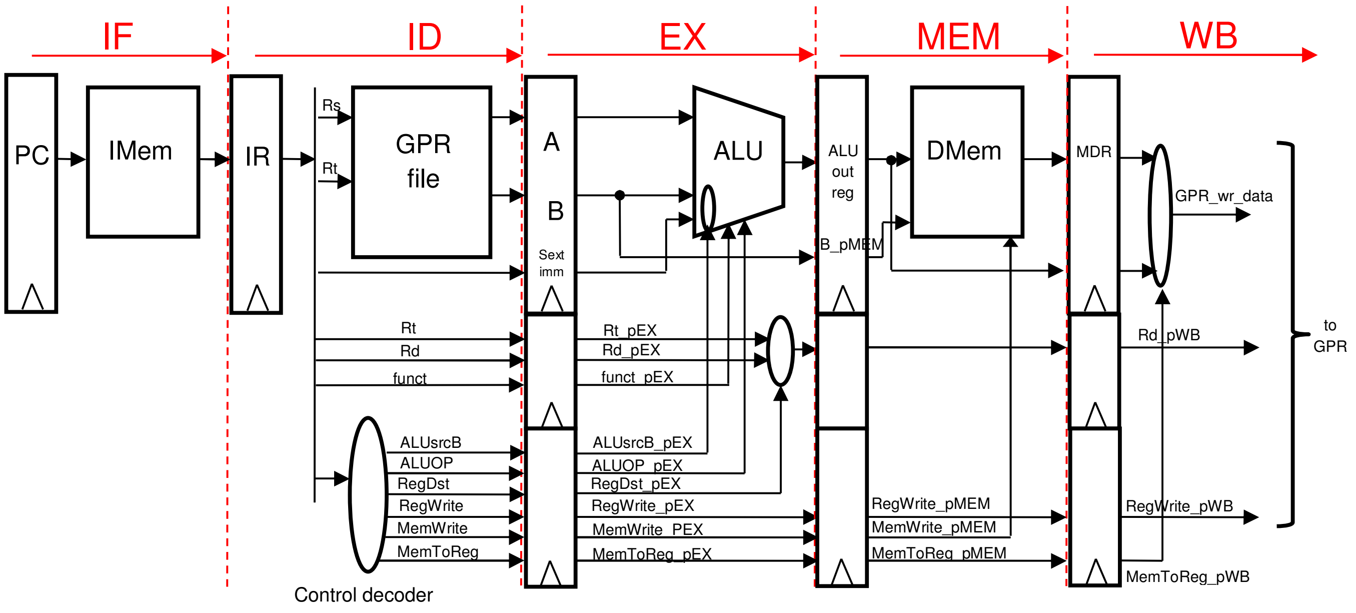

Fig. 3 – The HW5_MIPS CPU control scheme

c. Names and definitions of output signals from HW5_top MIPS CPU towards the TB

We need to define all output signals coming out of the **HW5_top** entity to be tested by the HW5_TB. These signals are the same as we used in **HW4_top** and they already defined as output pins of the **HW5_top** entity. These signals are:

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_top** design you should connect the rdbk signals as follows:

<u>ID signals:</u>		
rdbk0	=>	PC_reg
rdbk1	=>	IR_reg,
rdbk2	=>	sext_imm (in ID phase)
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 **SIM_HW5_TB_data.dat** containing the expected results. Since we will use these signals also in the implementation phase, we should connect them to the Host interface as well.

In Fig.4 below we describe the simulation project.

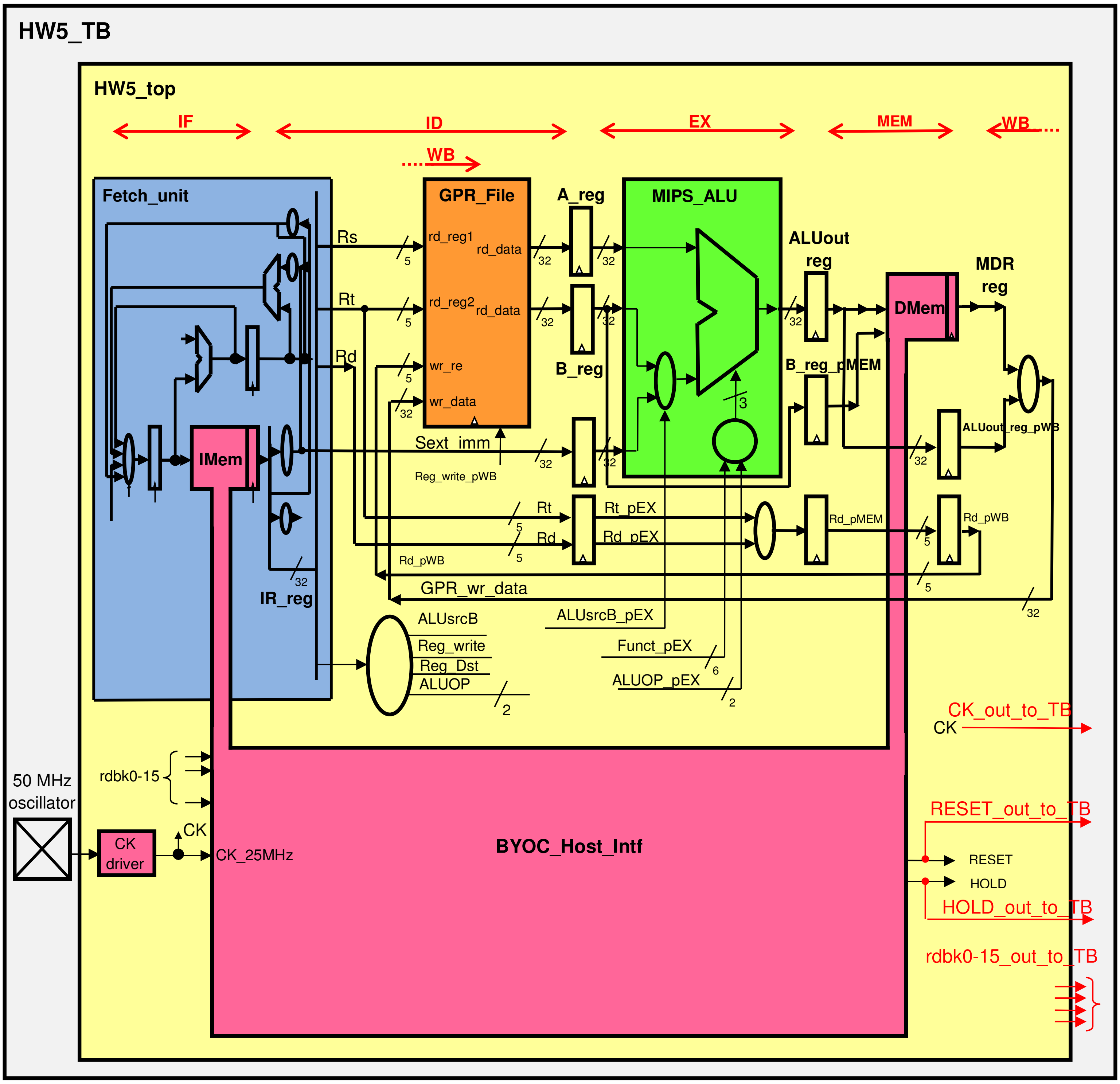

Fig. 4 – The entire simulation system

D. Description of the HW5_MIPS_4sim project

In a regular design you need to define all signals in your design. Then, you should write the equations of all signals and registers and connect all components used. Then you should run the simulation.

In our case, in the **HW5_top_4sim.empty** file we already defined all of the HW5_top signals. We also connected all of the components (Fetch_Unit, GPR, MIPA_ALU, BYOC_Host_Intf – those are the blue, orange, green and pink parts of Fig. 2). So after renaming it to **HW5_top_4sim.vhd** you need to build the logic inside that file, which is the yellow part that includes all of the registers, FFs and combinational logic elements forming the HW5 MIPS CPU.

The files we will use to run the simulation are:

<u>The design files:</u>

1) **HW5_top_4sim.vhd** – This is your design of HW5. It uses your designs of the Fetch_Unit, GPR, MIPS_ALU and the pre-prepared BYOC_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.

<u>BYOC infrastructure files:</u>

6) **BYOC_Clock_driver_4sim.vhd** – the CK divider & driver we use for simulation (also good for the Modelsim simulator)
7) **BYOC_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.

<u>Simulation files:</u>

8) **SIM_HW5_TB.vhd** - The TB vhd file prepared in advance.
9) **SIM_HW5_TB_data.dat** – this is a data file prepared in advance that has the expected TB values. It is read by the SIM_HW5_TB and used to compare the actual simulation results to the expected ones.
10) **SIM_HW5_program.dat** - The program file for simulation
11) **SIM_HW5_filenames.vhd** - The actual path information of the two dat files
    <u>NOTE</u>: You should update that file according to your simulation project actual path.

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 and the *.dat file (to be used by the 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). See the instructions below for the rest of the simulation report. ***Note that this should be a WORD file and not a PDF - to allow addition of remarks while grading the 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 **SIM_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 196-224 (following the end of the reset pulse, find i=196-224) and make the values of all signals readable. For this you will probably need to show clocks 196-210 and 210-224 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. PC_reg
6. IR_reg
7. MIPS_DMem_adrs
8. MIPS_DMem_wr_data
9. MIPS_DMem_we
10. MIPS_DMem_rd_data
11. DMem_reg0
12. DMem_reg1
13. DMem_reg2
14. DMem_reg3
15. DMem_reg4

I=196-224 means CK cycles from 8440 ns to 9600 ns.

3.2) Explain in detail what happens, i.e., what do we see here. Note that it is essential to the success of your future design that you will verify that the design does what we wanted it to do in these CK cycles.

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

3.3) What is the latency of an Rtype instruction? That is: 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 resides in register $3.

3.5) Are there any other limitations due to the pipeline structure in the instructions we implemented (Rtype, addi, beq, bne, lw, sw, j)? How can we overcome these limitations (e.g., by adding nop-s)? Try to list all of the **SW** & **HW** based solutions you can think of.

Later, in the Implementation phase you will add 3 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.
- **DOC** – here you should have the report described later in section 4 below.

3) HW5 MIPS CPU – implementation

After a successful simulation we want to implement the design on the Nexys2 board. Minor changes are required. We can rename the **HW5_top_4sim.vhd** we built in the simulation phase to **HW5_MIPS.vhd** and then, remove all the signals that were outputted to the TB (see 1c above). These are not required anymore. Or instead, we could use the pre-prepared **HW5_top.empty** file in which we removed all of the TB signals, rename it to **HW5_top.vhd** and copy our VHDL code from **HW5_top_4sim.vhd** into the appropriate location inside that file.

Another change required is to use the **BYOC_Host_Intf.ngc** instead of the **BYOC_Host_Intf_4sim.vhd**. The **BYOC_Host_Intf_4sim.vhd** has a pre-loaded IMem (actually, it is loaded during simulation) and a very limited DMem – made especially for simulation. The **BYOC_Host_Intf.ngc** has the infra-structure allowing loading of real programs (via RS232) displaying part of the DMem on the VGA screen and running our design in a single-clock mode for debugging.

Note that the Data Memory we added to our design 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.

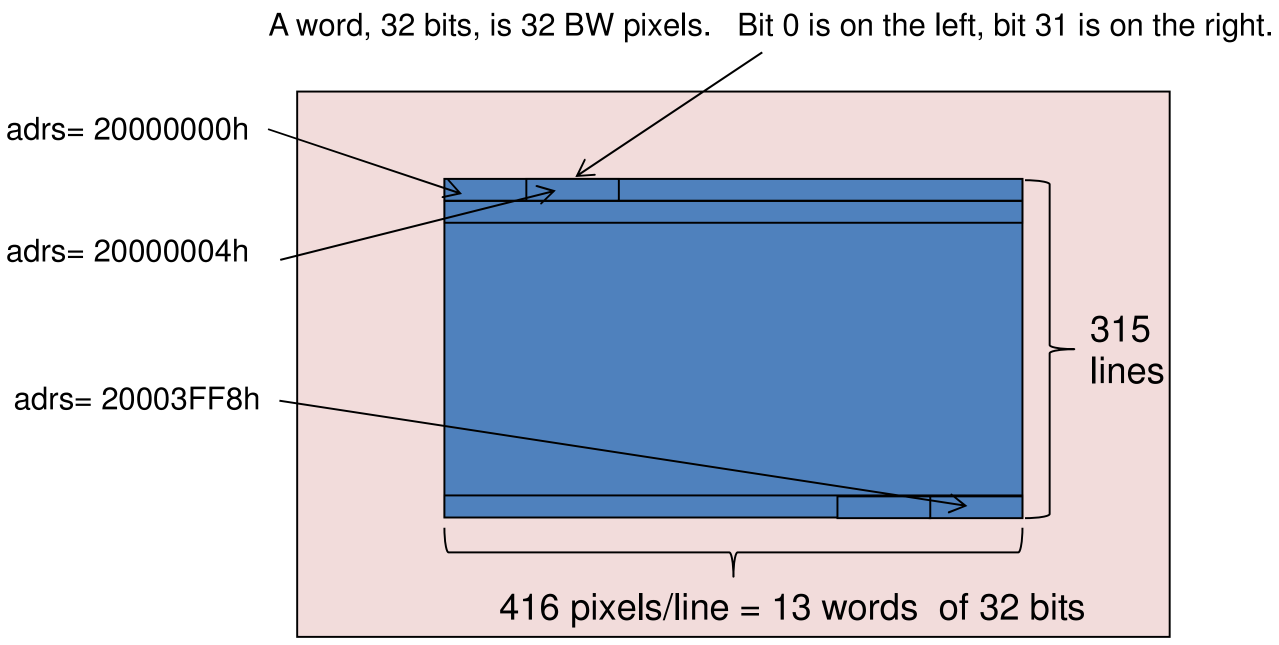
A word, 32 bits, is 32 BW pixels. Bit 0 is on the left, bit 31 is on the right.
adrs= 20000000h
adrs= 20000004h
adrs= 20003FF8h
315 lines
416 pixels/line = 13 words of 32 bits

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 copy from the flash into the IMem). In HW5 we will only use the VGA part.

In Fig. 6 below we see also the connections to the final connectors in the Nexys2 board. We see the PS/2 keyboard connector, the RS232 connector, the VGA D-type connector (and more). 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 need to understand the functionality of 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). There are also 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.

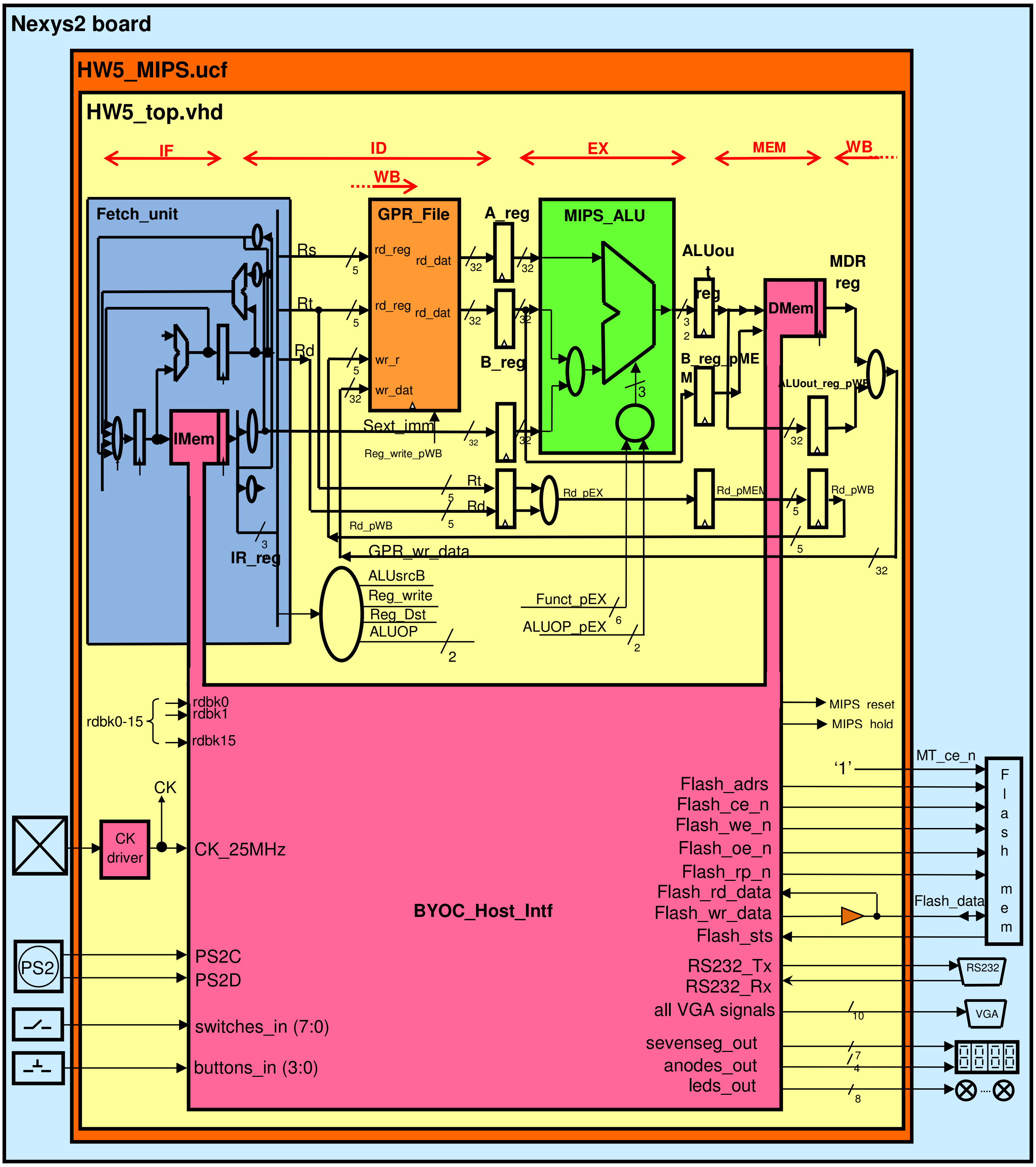

Fig. 6 – The entire system

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

1) **BYOC.ucf** - The file listing which signal are connected to which FPGA pins in the Nexys2 board.
2) **HW5_top.vhd** – This is your design of HW5. It uses the Fetch_Unit, GPR, MIPS_ALU, Clock_Driver 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_Clock_driver.vhd** – the CK divider & driver we also used in HW2 and HW4.
8) **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, as in HW4.

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_reg
rdbk1	=>	IR_reg,
rdbk2	=>	sext_imm (in ID phase)
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_rect4.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_rect4 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 126 CKs? What happens when you press the RUN button?
Take a look at the questions you need to answer in the implementation report.

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_top project created by the Xilinx ISE SW.
- **Docs** - Here you put your implementation report. The first few lines in the report will have your ID numbers (names are optional).
  ***This should be a WORD file and not a PDF file – so remarks can be added when grading the report*.**

  In this part of the implementation report you should answer the following questions:

  1) What is the value of register $2 after 122 cks?
  2) What happens after 126 CKs?
     (To answer these two questions look with the BYOCSIntf SW what happens from CK no. 120 till CK no.130)
  3) What happens when you press the RUN button?
  4) Explain the **HW5_rect4** program (what is the job of every register used. What is done in each loop, etc.)
  5) How long does it take [in seconds] to draw a 32x32 white square when we use the draw loop of the **HW5_rect4** program?
  6) Can you shorten the loop? If you can, write the code and explain.
  7) Can you think of a faster way to draw the square in the same short loop? If you can, write the code and explain.

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.

Appendix A – 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 – Rect4 - 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		addi	$4	$0		16		20040010
40000C		nop						00000000
400010		nop						00000000
400014	shft_lp	add	$2	$2	$2			00421020
400018		addi	$4	$4		-1		2084FFFF
40001C		nop						00000000
400020		nop						00000000
400024		nop						00000000
400028		bne	$4	$0		shft_lp		1480FFFA
40002C		nop						00000000
400030		addi	$2	$2		18h		20420018
400034		addi	$3	$0		-1		2003FFFF
400038		nop						00000000
40003C		nop						00000000
400040		nop						00000000
400044	drawlp	sw	$3	$2		0		AC430000
400048		addi	$1	$1		-1		2021FFFF
40004C		addi	$2	$2		52		20420034
400050		nop						00000000
400054		nop						00000000
400058		bne	$1	$0		drawlp		1420FFFA
40005C		nop						00000000
400060	end	j				end		08100018
400064		nop						00000000