UNIVERSITY of York

Department of Electronic Engineering

Assessments 2019/20

ELE00067M

Digital Design

This assessment (**Final Project**) contributes **30%** of the assessment for this module.

Clearly indicate your **Exam Number** on every separate piece of work submitted.

Unless the assessment specifies a group submission, you should assume all submissions are individual and therefore should be your own work.

All assessment submissions are subject to the Department's policy on plagiarism and, wherever possible, will be checked by the Department using Turnitin software.

Submission is via VLE and is due by **12:00** on **20 January 2020 (Spring Term, Week 3, Monday)**. Please try and submit early as any late submissions will be penalised.

Please remember that if this is your first year of study, you need to complete the mandatory Academic Integrity Tutorial http://www.york.ac.uk/integrity/

ELE00067M Digital Design

Final project: Part V

Memory subsystem and full integration

THE PROJECT SHOULD BE DONE IN GROUPS OF TWO STUDENTS

Report formatting:

There is no formal report structure – you will be marked on the items listed within the script. The report will include code printouts and simulation screenshots – see Lab 1 appendices for guidelines.

Only one report for each group for the entire project has to be submitted, in a single pdf file, via the VLE by the deadline indicated on the front page of the script. The PDF file should be named **Yxxxxxxx-Yxxxxxxx_DDMSc_Project.pdf**, where the Yxxxxxxx are the exam numbers of the group members. The PDF file should be compressed and submitted as a .zip archive.

The report should include all the material listed in each of the scripts that make up the complete project. The material for each script should start on a new page, containing all the items required in the order specified. The exam number should be printed on the first page of each script report.

In all cases, read carefully the instructions on the VLE submission page. Failure to follow the instructions could lead to your assignment not being marked and in any case to a mark penalty.

Submission weight on module mark: 30%

Marking criteria:

In general, the marking will take into account:

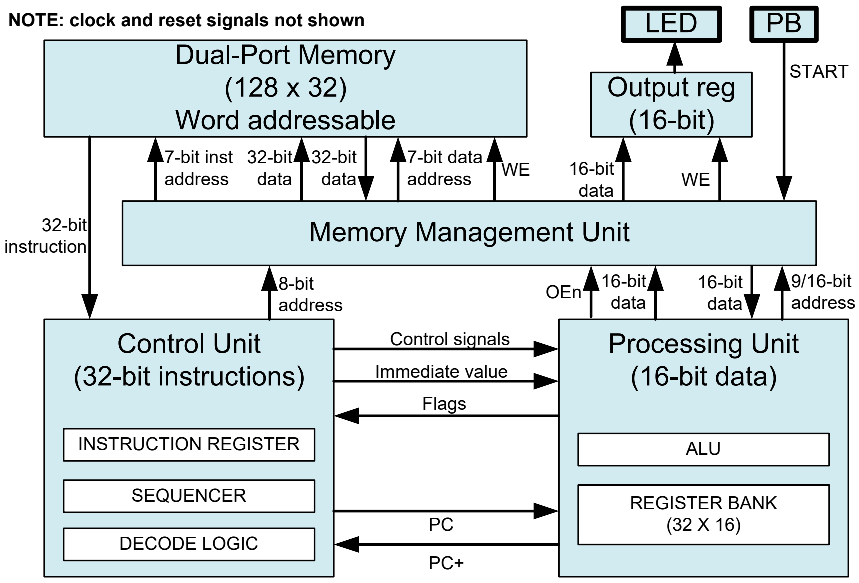
- Design quality, i.e. does the circuit generated by your VHDL operate correctly and efficiently.
- VHDL quality, i.e., is it written in accordance with the guidelines presented in the lectures.
- Documentation, i.e. are all required documents present? Are they legible? Is the code commented? (Refer to VHDL Style Guide for acceptable standards)

Memory subsystem, integration and verification **[20 marks]**

The processor sees the memory subsystem as two separate elements: a 256x32 read-only memory for the instructions and a 512x16 read/write memory for the data (both word-addressable). Your task in this step is to design a (very simple) **memory management unit (MMU)** that will allow the processor to work with a single 128x32 dual-port memory.

Specifically, the single-write/dual-read memory dual port memory is a RAM (identical to the ones seen in the VHDL portion of the module) with the addition of a secondary, read only port (address and data out). The memory will be word-addressable (i.e. each address matches a full 32-bit word).

The MMU will also allow the processor to interface with the external world using two *memory-mapped I/O devices* (a pushbutton and an array of LEDs).



Within the memory, addresses 0 to 63 will be used for the program (instruction memory), while addresses 64 to 127 will be used to store the data (data memory). Outputs should not be registered.

A few implementation details:

- The I/O peripherals for the processor will be *memory-mapped*, that is, they can be accessed as if they were memory locations. The address range x'0100 to x'01FF are reserved for peripherals. In particular, the START button (which should NOT be debounced in this particular case – unless you feel like modifying the standard debouncer to produce a pulse longer than a single clock cycle) will be accessed at address x'01F0 and the register that will be connected to the LEDs at address x'01F8 in the program memory space. As only 8 LEDs are available, only the 8 most significant bits should be displayed.
- You can assume that the program will not access memory locations outside the memory capacity (except the peripherals). In other words, not all the program memory space will be accessible (no page management).
- Note that the data is 16 bits wide, whereas the memory is 32 bits wide. You should nevertheless make full use of the memory and take into account that the memory will return *two* data words (4 bytes) for each access. This should not cause any undue problems for reads (you will simply have to select either the most significant or the least significant half of the word depending on the least significant bit of the address), but will have to be handled when writing. To simplify this, keep in mind that when the location to be written to is addressed, the memory will automatically read the *current* value and present it at its output.
- You can safely remove any logic (tri-state buffers or other) controlled by OEn in your datapath, if any. The OEn signal will become the write enable signal for your memories.
- The pushbutton value will obviously be coded as a single STD_LOGIC line. Note that this will have to be converted to a 16-bit word before being sent to the processing unit.
- The MMU is a *purely combinational* entity. The only sequential component involved is the output register (besides, of course, the memory).

Once the memory subsystem has been designed and integrated to the other components, the processor will be complete! To fully test the operation of the processor, the following program should be translated to machine language and loaded into the Instruction Memory (lower half of the dual-port memory) file. A <u>very</u> simple TB (which just provides a clock, a reset, and a start input) will be sufficient to test the processor.

MEM	**Label**	**Instruction**
00		nop
01		loadi r15, h01F0
02		br r15,=0,-1
03		addi r31, r0, h001F
04		add r1, r0, r0
05		inc r1, r1
06	L1	br r31, <0, +5 (L2)
07		storr r1, r31
08		inc r1, r1
09		dec r31, r31
10		jmp -4 (L1)
11	L2	loadi r2, h0010
12		ori r30, r0, h0004
13		loadr r3, r30
14		loado r4, r30, 3
15		xori r29, r0, hFFFF
16		move r7, r0
17		andi r5, r29, h0010
18	L3	andi r6, r4, h0001
19		br r6, =0, +2 (L4)
20		add r7, r3, r7
21	L4	shr r4, r4, 1
22		shl r3, r3, 1
23		dec r5, r5
24		br r5, ≠0, -6 (L3)
25		stori r7, 0
26		rol r7, r7, 9
27		ror r7, r7, 3
28		not r8, r7
29		xor r8, r29, r8
30		sub r9, r8, r7
31		subi r10, r9, h0001
32		br r10, >0, +9 (Lx)
33		br r10, ≥0, +8 (Lx)
34		addi r11, r10, h0002
35		br r11, ≤0, +6 (Lx)
36		or r12, r7, r11
37		storo r12, r0, 1
38		and r12, r12, r11
39		br r12, =1, +3 (Lok)
40		jmp +0
41	Lx	jmp +0
42	Lok	stori r7,h01F8
43		jmp +0

This first instruction does nothing, but allows the same program to work for both single and multi-cycle CPUs.

Do nothing until the user presses START

When the program gets to instruction 11, DMEM positions 0 to 31 should contain decimal values 32 to 1, r1 decimal value 33, and r31 decimal value -1 (65535).

When the program gets to instruction 25, r7 should contain decimal value 700 (result of the multiplication 25 x 28).

If the program has run correctly, execution should finish at instruction 43 and never move from there. Register and memory contents should be the following (decimal):
r1 = 33; r2 = 16; r3 = r4 = r5 = r6 = r9 = 0;
r7 = r8 = 44800; r10 = -1 (65535); r11 = r12 = 1;
DMEM(0) = 700; DMEM(1) = 44801
The LEDs should display decimal 44800.

Note that finishing execution at instruction 43 is *not* by itself sufficient to verify the operation of the system: the contents of registers and memories are fundamental.

A RTF version of this code is provided in a separate file on the module website.

Once the simulation produces the expected results, the design should be implemented on the Xilinx boards to verify that it synthesizes correctly. The decimal value 44800 (1010111100000000 in binary) should be output and therefore bits 10101111 should be displayed on the board LEDs.

REPORT: The report for this exercise should include:

- The machine language equivalent of the test program.
- The commented VHDL code for the memory subsystem.
- The commented VHDL testbench.
- Screenshots of the simulation of the test program, showing, in readable size, the following values for instruction sequences 6-11-12, 24-25-26, and 42-43:
    - inputs and outputs of your top module (pushbutton, LEDs, reset, clock)
    - the contents of IR in hexadecimal and PC in unsigned decimal notation
    - for the multi-cycle implementation, the state values for the complete instruction
    - the contents, in signed decimal notation, of the following registers: r1-r3, r7, r30 and r31
    - all inputs and outputs of the memory management unit
- The "RTL Component Statistics" part of the synthesis report
- The XDC file