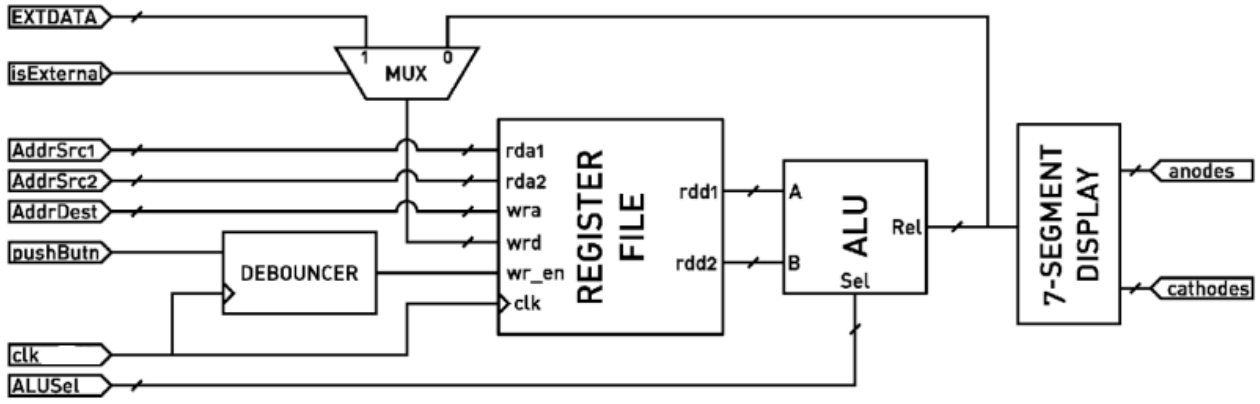
CS223 Project Assignment
Simple Processor Datapath

Lab dates and times:

Section 1:	15.04.2021 Thursday	13:30-17:20
Section 2:	13.04.2021 Tuesday	08:30-12:20
Section 3:	14.04.2021 Wednesday	08:30-12:20
Section 4:	15.04.2021 Thursday	08:30-12:20

Location: EA Z04 (in the EA building, straight ahead past the elevators)
Groups: Each student will do the lab individually. Group size = 1

In this lab we are going to design and implement a simple processor datapath, which includes a register file and an arithmetic logic unit, along with other peripheral modules. The peripheral modules include a 4-bit 2-to-1 multiplexer, a debouncer and a simple seven-segment display module. You will be using the switches on the BASYS3 board. In addition, you will use the leftmost push button as the trigger of the circuit; that is, the instruction will be done when pushButn is pressed. You will also use this design with minor extensions in the final project.



From left to right, the switches on your BASYS3 board will be used as follows:

2-bit select input of the operation which ALU is to perform (ALUSel), 3-bit Address selection buses,AddrSrc1, AddrSrc2, AddrDest, 1-bit isExternal, remaining 4 switches for EXTDATA bits, for external data input.

Register File

A 4-bit register file serves as a memory block, and the values stored in it can be accessed by specifying the addresses. Your register file should have three address buses, namely **rda1**, **rda2** and **wra** (abbreviations for "read address 1, 2" and "write address"); and three corresponding data ports, namely **rdd1**, **rdd2** and **wrd** ("read data 1, 2" and "write data"). For example, rdd1 should output the data stored in address rda1. Note that having two distinct read ports allows us to read two values at once.

rda1, rda2 and wrd are 3-bit buses; therefore your register file will have $2^3 = 8$ addresses. Each of these addresses should be capable of storing a 4-bit value. That is; rdd1, rdd2 and wrd should be 4-bit buses. Read ports are always enabled; however, writing should not always happen. Therefore you need another input,

namely **wr_en**, that enables the writing process (When wr_en is 1, your register file should write the data given by wrd into the address wra).

ALU

You are asked to implement an arithmetic logic unit. The inputs of the ALU are A and B, both of which are 4-bit inputs. The 2-bit control input Sel determines the operation the ALU is to do. The output is Res, which is also 4-bit (Flags that a real ALU should have, such as 'overflow' or 'carry' are neglected for simplicity).

The ALU should perform the following operations depending on Sel input:

Sel	Operation
00	Increment by 1 (A + 1)
01	Addition (A + B)
10	Subtraction (A – B)
11	Or bitwise (A or B)

Debouncer

There is a physical phenomenon that is observed when using pushbuttons, which is called bouncing. Simply put, when a person presses the pushbutton, the button bounces from the conductive surface below, causing more than one rising edges. If you will implement a design with an edge-driven pushbutton input, it is important to overcome this problem with a debouncer. The debouncer module is a finite state machine that creates a one-clock-cycle-long pulse synchronized with a high-frequency clock (in our case, 100 MHz for Basys3) whenever the push-button is pressed; and discards any other rising edges in the push-button signal for a while (100 ms should work fine). This will effectively eliminate any bouncing. You will be provided the debouncer code in Moodle as well as with 7-Segment Driver. You should be able to use it as is, you don't have to change anything.

Preliminary Work (40 points)

a. Write the Systemverilog code for RegisterFile, ALU and MUX modules (Basically, all modules).
b. Write test benches for your ALU and RegisterFile modules.
c. Write a top module for the datapath and an overall test bench. Instead of the 7-segment display output, show the ALU output in your simulation.

Lab Work (60 points)

a. Verify in simulation that your datapath module works.
b. Now, follow the Xilinx design flow to synthesize, create programming file, and download your top module to your BASYS-3 FPGA board. When you are convinced that it works correctly, show the physical implementation results to the TA. Be prepared to answer questions that you may be asked.