project structure.

5 Course project 1 (*Grade 4*) - A calculator with memory

In this project, the ALU implemented in lab assignment 3 and PS/2 keyboard controller designed in lab assignment 2 will be reused. A new IP will be generated using the Xilinx IP generator tool.

5.1 Task 1

Start by first understanding how the given VGA controller works. Try assigning your own rgb colors to the display instead of ROM data. Figure out how the vertical and horizontal counters can be used in order to emulate the seven segment display on the LCD. An illustration of the LCD display required is shown in a screen capture, placed under:

"*S:\course_projects\rtl_ref_designs\project_1.jpg*".

Integrate the VGA controller to the keyboard and the ALU. Use a top level file to instantiate these three IPs as components in order to keep them functionally in separate files. Reuse as much code as possible from your previous designs.

5.2 Task 2

In this step, you will generate your own memory module. The basic steps of generating an IP core are listed below.

* Click on the IP Catalog in the Project Manager.
* Search for RAM. In Memories & Storage elements, choose *RAMs & ROMs*. Then choose Block memory generator.
* In the new window that opens up, examine the memory block that will be generated.
* Choose a Single port RAM with the Algorithm set to Minimum Area.
* Set Memory write width to 8 bits and write depth to 8 kB.
* Leave all other options unchanged. Generate memory.

Once this is done, a new IP will appear in your design hierarchy window. Examine the HDL files generated by clicking on + beside the IP and choosing the HDL files. The component instantiation that needs to be used in your calculator design can be found in this HDL instantiation file.

5.3 Task 3

Integrate the memory module into your design by instantiating it as a component and verify that integration has succeeded. Refer to Fig 4 and the following steps for some suggested ways to verify the memory controller. We will perform read and write operations to the memory using the basic pins and switches available on the board.

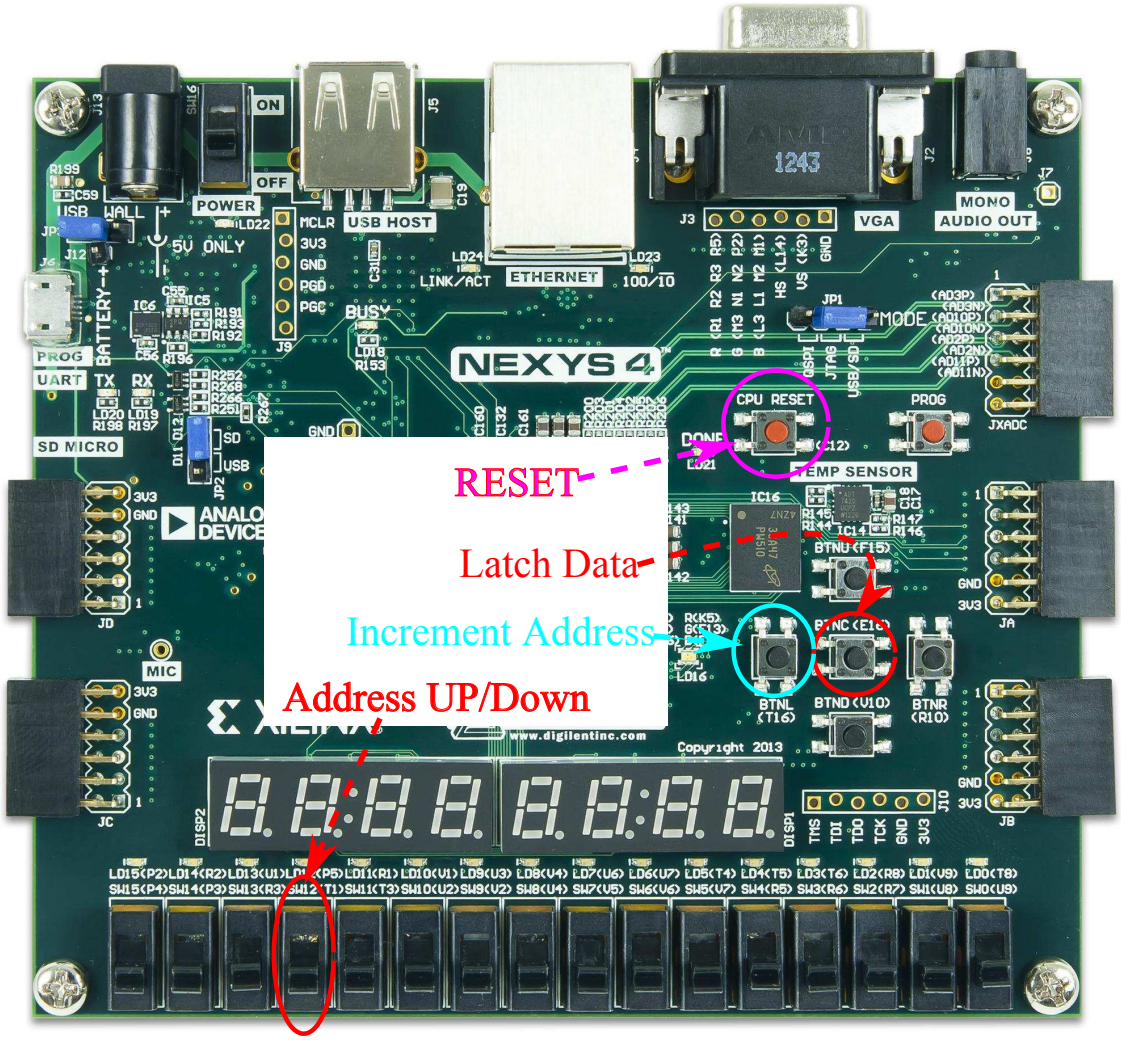

Figure 4: FPGA with memory controls

* Assign CPU RESET to reset your system. Check whether the reset is active high or low and make appropriate changes to your code. If you have a different polarity, instead of changing the whole code, you can create a local reset signal with the required polarity in your top level and use this as the reset for the rest of your system.

* Even though the mem_data bus generated from the IP will be 8 bits, for testing we will now use only 4 bits. Assign the mem_data[3 downto 0] bits to your keyboard out data. Assign the upper 4 bits to zero.

* Design a counter and connect the mem_address to this counter. When BTNL is pressed and if SWITCH[12] is set to 0, the address should increment. The address

should decrement if BTNL is pressed when SWITCH[12] is set to 1. Remember to use debouncing logic on the BTN, if not the memory address might increment by more than one at each press of BTNL. It would be a good idea to also connect the mem_write_enable to this button.

* Try and use the LED0-7 present on the board for debugging. Check whether after adding debouncing logic the address increments by the required steps.

* Assign BTNC to enable data latching. The keyboard data should be registered to the memory input only when BTNC is pressed.

* Connect the memory_out data to the seven segment display, either on the FPGA or on the LCD screen for debugging.

It is always a good idea to look at the warnings tab when synthesizing the design. Understand the warnings shown and see if they are OK for your design. It may happen that the memory block is not connected properly and your system does not work.

5.4 Task 4

The next step is to write code to enable data storage in the memory along with the operators. Use the same BTNC-BTNL logic described above to store a string of data into the memory along with the operands. The input data range is from 0 to 255 and the inputs are considered to be unsigned. At the end of entering data values along with operands, the ALU should be started. This can be done by pressing the <Enter> key on your keyboard.

A) At the press of every <Enter> key the memory controller should be able to pop the top three memory locations (the two data operands and the operator), compute the result and display it on the VGA screen.

B) On the next <Enter> key the next two data operands and the operator have to popped out from the memory and result should be displayed on the VGA screen. Do not forget to take into account that for the mod 3 operator we need to enter only one data and the operand. Remember also that the result could be either a positive or a negative number. Therefore it is required to display the sign of the result before the result as shown in the example in Fig. 5.

C) Since the data RAM created will be 8 bits wide and we need to store some operands along with data, some of the bit patterns can be assigned for these operators(e.g. “+”, “-”, “=” “mod”). Choose the range of 130 to 135 for operators. This also means that input data in the range of 130 to 135 shall not be considered as operands.

D) The values should be stored into the RAM only when proper operands and operators have been entered. There should be an option to use the back space key to input a different set of operators and operands. For example, if a mistake is done while entering the operands, one could use the backspace key to delete the already entered numbers and start over. A detailed description of operators and operands is as follows:

E) Both data operands must be displayed in at least 3 digits (hundreds, tens, units) on the VGA screen and the computation results must also be represented in three digits (hundreds, tens, units) along with the **sign**. The operands, computation result along with the operator must be shown on the emulated 7-segments on a VGA monitor. For example if one has to compute the sum of 98 and 99 the VGA display should look like the ouput in Fig. 5. The inputs will be entered in the 3 digits format, meaning if one wants to use 9 as an operand, the input from the keyboard shall be 009. If the data entered is above the limit, then the number shall be stored as 255. For example if the user enters 1234 as the first input operator, the calculator shall store this number as 255 when the data latch button is pressed. Note that the backspace key should be operational to fix the data before the data latch button is pressed.

F) If you plan on performing project 2, consider that the maximum width of the result displayed on the monitor should be 5 digits, which corresponds to the square root of 255 = 15.968.

Figure 5: Example VGA Output

Remember that the result is signed and the operands are unsigned. This will enable one to design a simple state machine to accept the right amount of inputs before storing them in the memory.

G) The design must be able to perform the following different computation operations: addition, subtraction, multiplication and modulo 3. An indication of overflow/underflow should also be displayed when it happens.

H) The emulated 7-segments have to be shown in a visible size. It is allowed to load digits and operators from data ROMs, however, you have to consider the available memory capacity in the FPGA. It is recommended to design a display engine for one 7-segment, and use it to generate digits at all locations during system run-time. Using either logics or data memories is always a design trade off, where a common practice is to use a mixed design approach to find a balanced point between them. You may, for instance, store all data operators (e.g. “+”, “-”, “=”) in ROMs, and generate all digits by using one 7-segment display engine.

An example output of the memory operation is shown in Fig 6. Refer to the presentations uploaded along with this manual for details on how to fill in the memory and reading the memory. To begin with, the write address is “0". When data and operands are entered, the memory will fill up in a way similar to a stack. Once the user decides to compute the results, the <Enter> key will be pressed. This should result in popping data from the top of the stack and displaying results on the VGA screen. Further <Enter> keys should pop data correctly in order to display the corresponding result. The calculator should also be

able to accept data inputs in the middle of displays. For example in Fig. 6 after the third <Enter> key has been pressed resulting in a display of +049 on the VGA monitor, the user should be able to input more operands and operators. This should result in the memory being filled up again until the user decides to evaluate the results using the <Enter> key. **Contact the teaching assistant if you have any questions on the operation of the memory.**

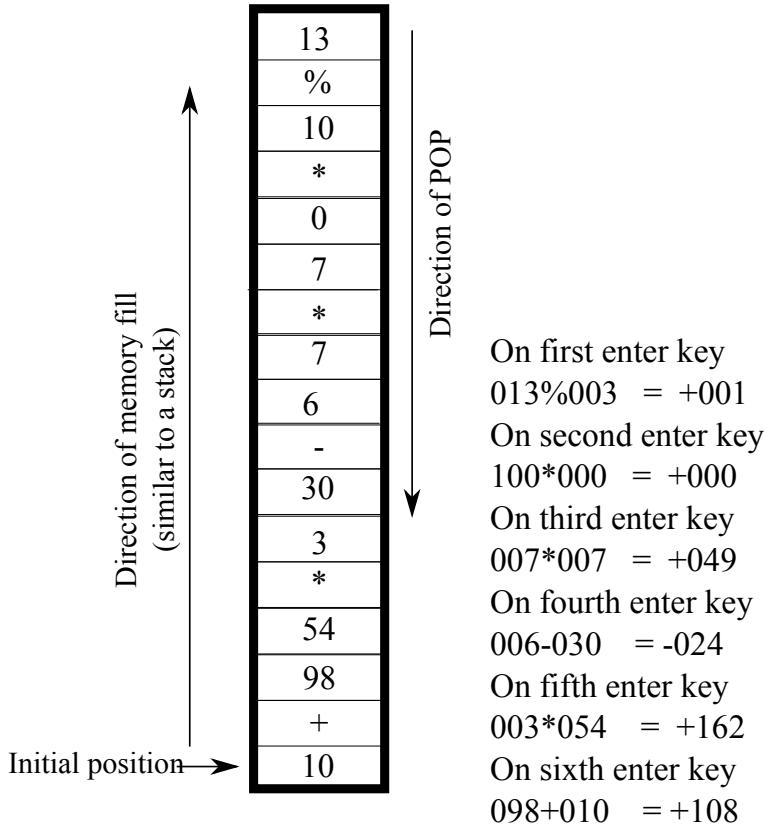

Figure 6: Example output