CS 223 Digital Design Laboratory Assignment 4
Multifunction Register and 7-Segment Display

Preliminary Report Due: April 15, 2024 13:00

Lab Dates and Times

Section 1: April 18, 2024 Thur. 13:30-17:20 in EA-Z04
Section 2: April 15, 2024 Mon. 13:30-17:20 in EA-Z04

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

Preliminary Work [30]

Note: This part must be completed before coming to the lab. Prepare a neatly organized report of your work and submit on Moodle before the start of the lab. Include your code as **plain text, not image!**

In this lab, you are going to design a 4-bit shift register with parallel load by using synchronously resettable D flip-flops. This register will perform a variety of operations and the desired operation will be achieved by setting the control inputs of the register (You will design a multifunction register capable of Parallel Load, Shift Right, and Shift Left operations). Then, you will create a synchronous 7-segment display driver to control the onboard display digits.

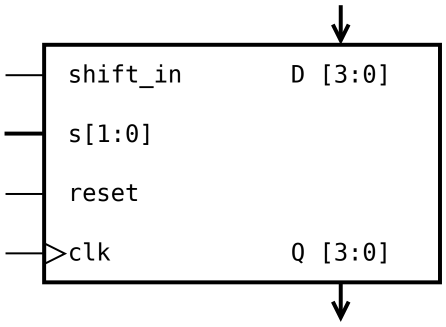


Figure 1: Multifunction register block diagram.

Table 1: Multifunction register operation table.

reset	s_1	s_0	Operation
1	0	0	Clear (load all 0's)
0	0	0	Maintain present value
0	0	1	Parallel load
0	1	0	Shift left
0	1	1	Shift right

Multifunction Register

In this experiment, four resettable D flip-flops with inputs and outputs will be used. All flip-flops will be loaded on every clock cycle. The block symbol showing the multifunction register's inputs and outputs is shown in Fig. 1. The operation table that defines the desired operation for each combination of control inputs is also given in Table 1.

- Reset operation will clear the outputs of all flip-flops on the rising edge of the clock ($Q_{[3:0]} = 0000$). Reset input has priority over the other control inputs ($s_1 s_0$).
- When $s_1 s_0 = 00$ and the clock signal rises, each flip-flop stores its own value and hence the register will retain its present content ($Q_3 \leftarrow Q_3, Q_2 \leftarrow Q_2, Q_1 \leftarrow Q_1, Q_0 \leftarrow Q_0$).
- When $s_1 s_0 = 01$ and the clock signal rises, the register will perform a parallel load operation. Parallel load operation causes the register to get loaded with the external data inputs on the rising edge of the clock ($Q_3 \leftarrow D_3, Q_2 \leftarrow D_2, Q_1 \leftarrow D_1, Q_0 \leftarrow D_0$).

- When $s_1s_0 = 10$ and the clock signal rises, the register will perform a shift left operation. Shift left operation causes a left shift on the rising edge of the clock and `shift_in` input is shifted into the rightmost bit (LSB) of the register ($Q_3 \leftarrow Q_2, Q_2 \leftarrow Q_1, Q_1 \leftarrow Q_0, Q_0 \leftarrow \texttt{shift\_in}$).
- When $s_1s_0 = 11$ and the clock signal rises, the register will perform a shift right operation. Shift right operation causes a right shift on the rising edge of the clock and `shift_in` input is shifted into the leftmost bit (msb) of the register ($Q_3 \leftarrow \texttt{shift\_in}, Q_2 \leftarrow Q_3, Q_1 \leftarrow Q_2, Q_0 \leftarrow Q_1$).

Prepare circuit schematics, SystemVerilog models and testbenches to be included in a Lab Report that has a cover page and pages for the schematics and SystemVerilog codes. The content of the report will be as follows:

A cover page including: course code, course name and section, the number of the lab, your name-surname, student ID, date.

[5] Write a SystemVerilog module for synchronously resettable D flip-flop.

[5] Draw a circuit schematic (block diagram) for the internal design of the multifunction register by using 4:1 multiplexers and synchronously resettable D flip-flops.

[10] Write a Structural SystemVerilog module for the multifunction register you designed and prepare a testbench for it.

Note that structural modeling refers to using and combining simpler components to create more sophisticated building blocks (it is an application of hierarchy). You can refer to the slides of Chapter 4 of your textbook while preparing your SystemVerilog modules and testbenches.

Binary-Coded Decimal to 7-Segment Decoder

Binary-coded decimal (BCD) is a form of representing numbers that is suitable for driving digit-oriented displays. The concept can be best understood through an example: Consider the decimal number "`23`". You need at least five bits to store it, but assume you store your numbers in groups of eight bits. Eight-bit representation of this number is "`0001_0111`". However, when you need to drive a digit-oriented display like the seven-segment display on the Basys3 board, you need to calculate each digit of this number, "`2`" and "`3`" whose binary representations are "`0010`" and "`0011`" respectively. These two binary numbers look nothing like the original two-digit decimal number at the first sight.

Mathematically, you need to perform $\mod 10$ and $\div 10$ operation to get the first and second digits respectively. The division operation is not trivial to realize in FPGA logic unless you are using DSP slices. In this case, it might be preferable to store the number the way you are going to use it: in binary-coded decimal form. If you store the number as "`0010_0011`" where the MSB four-bits are "`2`", and the LSB four-bits are "`3`", you can directly drive the seven-segment display from these bits.

Next, let us look into how the digit-oriented seven-segment display works. Fig. 2b shows a single digit on the display and their segments named with signals $a - g$. The name is obviously coming from the fact that it has seven segments per digit. You can uniquely represent decimal numbers in the $0 - 9$ range, and also all the hexadecimal numbers $0 - F$ with these seven segments. Convince yourself that this is the case. Each segment is controlled by its own signal and depending on how you drive each one, you can alter what is being displayed on the digit. For example, you need to light up all outer segments $a - f$ and only turn off segment g to display the number "`0`". You can already see that each number has a set of bits called the "code" to drive the seven-segment display.

In this part, you are going to implement BCD-to-7-segment display code converter logic with i/o shown in Fig. 2a to drive a single seven-segment digit in Fig. 2b according to truth table in Table 2. For hexadecimal numbers out of the $0 - 9$ range, you may display a "`dash`" character.

[3] Write a SystemVerilog module for BCD-to-7-segment code converter.

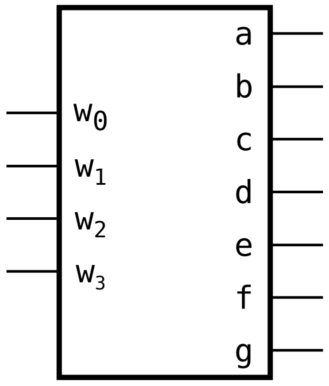
[3] Write a testbench for it to verify functionality.

[2] What are the disadvantages of using BCD to represent numbers? List two or three points and explain them with a few short sentences.

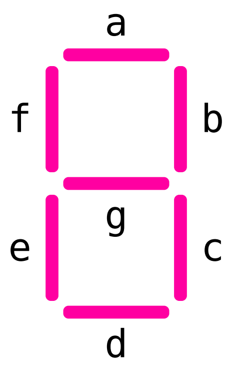
[2] Research how the full four-digit seven-segment display on the Basys3 board works. Think about which sequential components you need to drive it. Explain with a few short sentences.

Hint: At first sight, something might seem "off" in Table 2. This is one of the points you should explain.

Note that this last step is mostly intended for you to prepare for the lab assignment in advance. You should do some detailed research for your own purposes instead of just answering this question.



(a) BCD-to-7-segment display code converter.



(b) Single digit 7-segment display.

Figure 2: BCD-to-7-segment display code converter and 7-segment digit.

Table 2: BCD-to-7-segment display code converter truth table.

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0
ELSE				1	1	1	1	1	1	0

Part 1: Multifunction Register Implementation on FPGA [40]

In this step, you will implement your multifunction register module on the FPGA board according to the operation table previously given to you in Table 1. You don't need to connect your Basys3 board to the Beti board. Working with standalone Basys3 and having it connected to your computer is enough for this lab. There are some switches and LEDs available on Basys3 which you can use them to test your designs.

Create a new Xilinx Vivado Project. Use appropriate names for files and folders, keeping the project in a directory where you can find and upload it on Moodle at the end of lab.

[3 × 5] **Simulation:** Using the SystemVerilog module for the multifunction register and testbench code you wrote in the preliminary work, verify in simulation that your circuit works correctly.

[5 × 5] **Test:** Now, follow the Xilinx Vivado design flow to synthesize, implement, generate bitstream file, and download your multifunction register to Basys3 FPGA board. Using the switches and LEDs (on Basys 3) that you have assigned in constraint file (`.xdc`), test your multifunction register. 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.

Note: You are **NOT** allowed to use a button or a switch for your clock signal and your clear, load, and shift operations should be **clearly observable** on LEDs. There are no excuses for unexpected behavior of LEDs especially during shift operations. Research and plan ahead.

Part 2: 7-segment Display Driver on FPGA [30]

In this part, you will create a seven-segment display driver and utilize all four digits by accepting inputs from all sixteen switches on the Basys3 board. You already have a code converter to convert a BCD to its seven-segment code. First, you will display a single-digit and turn off the other three digits. Then, you will expand this and utilize the full four digits.

Note: You can directly implement the multi-digit driver and get full points from this section. The single-digit implementation is for incremental work and for you to get partial points.

Single-Digit

Design: Create a single digit seven-segment display driver that has an instance of your BCD-to-7-segment code converter, four-bit input, and other necessary logic to drive and display a single-digit.

[3] Simulation: Using the SystemVerilog module for the single-digit driver, prepare a testbench and verify in simulation that your circuit works correctly.

[7] Test: Synthesize, implement, generate bitstream file, and download your single-digit driver to Basys3 FPGA board. Assign four consecutive switches as your input and follow the convention where the leftmost bit is the MSB and rightmost bit is the LSB. You can choose any one of the four of the digits on the display.

Using the switches that you have assigned in the constraint file (`.xdc`), test your single-digit driver. The binary number you input using four switches should appear as a decimal number on the display. If you wish, you may extend the code converter to accept and generate code for all hexadecimal numbers in the range $0 - F$, but this is not required and displaying decimal numbers in the range $0 - 9$ is sufficient.

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.

Multi-Digit

Design: Expand the single-digit seven-segment display driver you created in the last section to utilize the whole display unit. Depending on your initial design, you might need to introduce new logic to handle driving all four digits.

Hint: If you have already done prior research on how a seven-segment display is controlled with sequential logic, this implementation should be rather easy for you.

[6] Simulation: Using the SystemVerilog module for the multi-digit driver, prepare a testbench and verify in simulation that your circuit works correctly. Focus on the logic that handles different digits.

[14] Test: Synthesize, implement, generate bitstream file, and download your multi-digit driver to Basys3 FPGA board. Assign four groups of four consecutive switches as your inputs and follow the convention where the leftmost bit is the MSB and rightmost bit is the LSB. The leftmost group of four switches should control the leftmost digit on the seven-segment display and the next group on the right should control the next digit on the right and so on.

Using the switches that you have assigned in the constraint file (`.xdc`), test your multi-digit driver. The binary numbers you input using groups of four switches should appear as decimal numbers on the display in the corresponding positions. If you wish, you may extend the code converter to accept and generate code for all hexadecimal numbers in the range $0 - F$, but this is not required and displaying decimal numbers in the range $0 - 9$ is sufficient.

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.

Clean Up

1. Clean up your lab station, and return all the parts, wires, the Beti trainer board, etc. Leave your lab workstation for others the way you would like to find it.
2. CONGRATULATIONS! You are finished with Lab #4 and are one step closer to becoming a computer engineer.

Notes

- Advance work on this lab, and all labs, is strongly suggested.
- Be sure to read and follow the Policies and other related material for CS223 labs, posted in Moodle.

Lab Policies

1. There are three computers in each row in the lab. Don't use middle computers, unless you are allowed by lab coordinator.
2. You borrow a lab-board containing the development board, connectors, etc. in the beginning. The lab coordinator takes your signature. When you are done, return it to his/her, otherwise you will be responsible and lose points.
3. Each lab-board has a number. You must always use the same board throughout the semester.
4. You must be in the lab, working on the lab, from the time lab starts until you finish and leave. (bathroom and snack breaks are the exception to this rule). Absence from the lab, at any time, is counted as absence from the whole lab that day.
5. No cell phone usage during lab. Tell friends not to call during the lab hours–you are busy learning how digital circuits work!
6. Internet usage is permitted only to lab-related technical sites. No Facebook, Twitter, email, news, video games, etc–you are busy learning how digital circuits work!
7. If you come to lab later than 20 minutes, you will lose that session completely.
8. When you are done, DO NOT return IC parts into the IC boxes where you've taken them first. Just put them inside your Lab-board box. Lab coordinator will check and return them later.

Recommendations

When building circuits using IC's and FPGAs in CS223 labs, it is important to follow some simple guidelines to prevent damage to electronic parts or confusion during debugging. By following these rules, you can ensure that your circuit functions correctly and avoid potential problems. Here are some key points to keep in mind:

- Avoid touching IC or FPGA pins directly by your hand. Static electricity from your body can permanently damage them. If you have to touch the pins, first ground yourself by touching a nearby grounded surface.
- The white board which you setup your circuit on it, is called "breadboard". Research online and find out how its pins are connected internally and use this knowledge when building your circuit.
- Postpone connecting power pins (V_{cc} and ground) until the last step. Check all other connections first, then connect the power pins if everything seems correct.
- Use a consistent wire color convention for easier debugging of circuits. For example, always use black or white wires for ground and red wires for V_{cc}. This will help you quickly identify any issues that may arise during testing.
- If an LED's light is weak or the IC's package feels hot to touch (you can safely touch the plastic part), there might be a problem with power pin connections, such as a short circuit or connecting V_{cc} wire to ground pin.

Updated by: Barış Bilgin Şenol, March 2024.