
CS 223 Digital Design Laboratory Assignment 4

Multifunction Register and 7-Segment Display

Preliminary Report Due: November 25, 2024 08:00

Lab Dates and Times

Section 1: November 25, 2024 Mon. 08:30-12:20 in EA-Z04
 Section 2: November 26, 2024 Tue. 08:30-12:20 in EA-Z04
 Section 3: November 27, 2024 Wed. 08:30-12:20 in EA-Z04
 Section 4: November 25, 2024 Mon. 13:30-17:20 in EA-Z04
 Section 5: November 29, 2024 Fri. 08:30-12:20 in EA-Z04
 Section 6: November 26, 2024 Tue. 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 it 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, which you will then use to observe a key press.

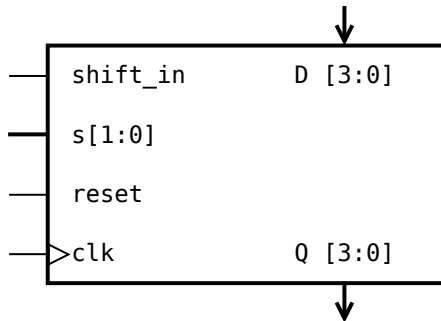


Table 1: Multifunction register operation table.

<i>reset</i>	<i>s</i> ₁	<i>s</i> ₀	OPERATION
1	x	x	CLEAR (LOAD ALL 0's)
0	0	0	MAINTAIN PRESENT VALUE
0	0	1	SHIFT LEFT
0	1	0	SHIFT RIGHT
0	1	1	PARALLEL LOAD

Figure 1: Multifunction register block diagram.

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_1s_0).
- When $s_1s_0 = 00$ 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 $s_1s_0 = 01$ 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 ($Q_3 \leftarrow Q_2, Q_2 \leftarrow Q_1, Q_1 \leftarrow Q_0, Q_0 \leftarrow \text{shift_in}$).

- When $s_1s_0 = 10$ 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 ($Q_3 \leftarrow \text{shift_in}, Q_2 \leftarrow Q_3, Q_1 \leftarrow Q_2, Q_0 \leftarrow Q_1$).
- When $s_1s_0 = 11$ 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 ($Q_3 \leftarrow D_3, Q_2 \leftarrow D_2, Q_1 \leftarrow D_1, Q_0 \leftarrow D_0$).

Prepare circuit schematics, SystemVerilog models and test benches 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, and 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 test-bench for it.

Note that structural modelling 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.

Hexadecimal to 7-Segment Decoder

Hexadecimal numbers are comprised of 4-bit values that can be represented using characters $0 - F$ which are suitable for driving digit-oriented displays. Seven-segment display built-in on the Basys3 board is one such digit-oriented display. Fig. 2b shows a single digit on the display and their segments named with signals $a - g$. The name comes from the fact that it has seven segments per digit. You can uniquely represent 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 Hex-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.

- [3] Write a SystemVerilog module for Hex-to-7-segment code converter.
- [3] Write a testbench for it to verify functionality.
- [2] What are the advantages/disadvantages of using hex values to represent numbers? List two or three points and explain them with a few short sentences. Provide at least one disadvantage.
- [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.

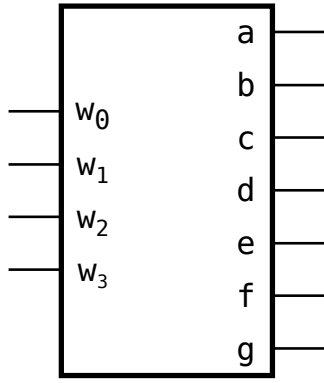
Hint: At first sight, something might seem "off" in Table 2. This is one of the points you should explain.

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.

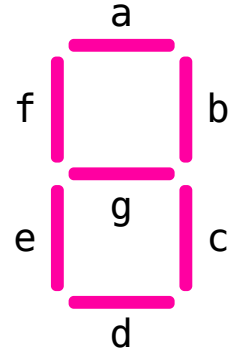
Part 1: Multifunction Register Implementation on FPGA [20]

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 a 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 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.



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



(b) Single digit 7-segment display.

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

Table 2: Hex-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
1	0	1	0	0	0	0	1	0	0	0
1	0	1	1	1	1	0	0	0	0	0
1	1	0	0	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0	0	1	0
1	1	1	0	0	1	1	0	0	0	0
1	1	1	1	0	1	1	1	0	0	0

[1 × 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.

[3 × 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 the 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 the 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 hex value 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 hex-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 a 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 the 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 hexadecimal number on the display.

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 a bitstream file, and download your multi-digit driver to the 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 the 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 hexadecimal numbers on the display in the corresponding positions.

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.

Part 3: Lab Counter with Alarm [20]

In this section, you are going to implement a lab counter using your multifunction register and 7-segment display driver design. The operation of the lab counter is described as follows:

- The lab counter will double or halve its count at every rising edge of the clock depending on the operation switches that were previously defined in Table 1. The counter should roll over when the count cannot fit into 16-bits. Designate the leftmost 4 switches on Basys3 for mode selection purposes.
- The lab counter will store 16-bits. Rightmost 12 switches will be used to load a value to your counter. As you have only 12 bits to do a parallel load, set the MSB 4 bits to zero whenever you do a load operation. For example, if you were to set the 12 switches to hex value ADF, you should load 0ADF to your counter instead.
- Display the current count of the lab counter on the full 4 digits of the 7-segment display using the driver you wrote in Part 2.
- Convert the least significant 4 digits of your ID number to hex and use this value to set an "alarm" for your counter. When the lab counter goes over this value, flash (constantly turn on and off) some onboard LEDs. The alarm should stop when the value goes below the count.

For example: ID = 21703020 → 3020 → 0x0BCC.

- Example mode of operation when you first load 0x14 in double mode and alarm is set to 0x0BCC:
0x0014 → 0x0028 → 0x0050 → 0x00A0 → 0x0140 → 0x0280 → 0x0500 → 0x0A00 → 0x1400 (alarm goes off)
→ 0x2800 → 0x5000 → 0xA000 → 0x4000 (rolls over) → 0x0000 (alarm turns off)

Note: You can directly implement the lab counter and get full points from this section. The simulation is for you to get partial points.

Design: Combine the multifunction register and the 7-segment display driver to create the lab counter described above. Add new logic to set an alarm and to flash LEDs depending on the state of the counter.

[5] **Simulation:** Using the SystemVerilog module for the lab counter, prepare a testbench and verify in simulation that your circuit works correctly.

[15] **Test:** Synthesize, implement, generate a bitstream file, and download your lab counter to the Basys3 FPGA board. Assign the leftmost four switches for states and the rightmost twelve switches as your inputs and follow the convention where the leftmost bit is the MSB and the rightmost bit is the LSB.

Using the switches that you have assigned in the constraint file (.xdc), test your multi-digit driver.

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 exceptions 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 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 the lab later than 30 minutes, you will lose that session completely.
8. When you are done, DO NOT return IC parts to the IC boxes where you've taken them first. Just put them inside your Lab-board box. The lab coordinator will check and return them later.

Recommendations

When building circuits using ICs 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 with 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 ground surface.
- The whiteboard on which you set up your circuit on, is called the “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 colour 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 ICs 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 the ground pin.