Lab 1: Fixed-Point Temperature Display

`ECE 212: Digital Circuits II – Fall 2024`

Abstract. In the lab, students will work to read a fixed-point value from the ADT7420 temperature sensor on the Nexys A7 board, convert that value to Fahrenheit, and round it to the nearest tenth, before displaying it on the board's seven-segment display. A provided using a VHDL IP block by Digilent is used to read the ADT7420 value. The design shall allow hardware testing of the full range of temperature values by substituting switches `SW[12:0]` for the 13-bit value from the sensor module.

Objectives.

- Gain experience working with fixed-point numbers consisting of an integer part and a fractional part.
- Implement a logic circuit using combinational logic in SystemVerilog
- Test the operation of a circuit using the input and output components provided by the Nexys A7

Materials and Equipment.

- Xilinx Vivado
- Nexys A7-100T FPGA board with USB cable

Deliverables.

- In-person lab demonstration (per team)
- Code submission via GitHub repository
- Lab report (per team)

Part 0: Set Up a Vivado Project and a Shared GitHub Repository

To begin the lab, set up a new Vivado project for Lab 1 and import the SystemVerilog files provided with this assignment on Moodle. You can reference Lab 0, Part 1 for a more detailed description of these steps.

For this and future labs, source code will be managed and submitted via private Git repositories (*e.g.,* GitHub). In this part, you will configure a Git repository to use with your lab partner and the instructor for the entire semester. Each Git repository will contain individual sub-folders for each lab assignment.

1. Create a Git repository by following the directions in the *Introduction to the Git Version Control System* handout. The repository should be named `ece212_names` or where `names` are the last names of your team members.

2. Under "Settings/Collaborators," share the Git repository with the instructor (`LBiernac`).

3. Open a “Git CMD” command prompt window from the “start” menu by navigating to start->All apps->Git->Git CMD (you may also want to right-click on this menu item and select “Pin to Start” for faster access in the future). This command prompt window accepts commands for working with files and directories as well as invocations of the Git command.

4. Using the “`cd`” command, navigate to the folder where you wish to place your lab files.

5. Set up your username and email with the following commands:

```
git config --global user.name lbiernac
git config --global user.email biernacl@lafayette.edu
```

6. With the newest version of GitHub, you will also have to generate ssh keys and add them to your GitHub account in order to connect with GitHub from the lab machine. Follow the instructions on how to do this here:
   https://docs.github.com/en/authentication/connecting-to-github-with-ssh

   Under the subtitles “*Generating a new SSH key and adding it to the ssh-agent*” and “*Adding a new SSH key to your GitHub account*”

7. Once this is configured, clone the repository onto your computer.

```
git clone git@github.com:yourUsername/RepositoryName.git
```

8. Add a folder named “Lab01” inside of this repository. This folder will contain all source files for this lab, including SystemVerilog and Constraints files downloaded from Moodle.

9. Create a Xilinx Vivado project inside of the Lab01 folder made in the previous step. Import the provided constraints file (`lab01_constraintsxdc`) and SystemVerilog/VHDL modules (`*.sv`, `*.vhdl`).

10. Connect your FPGA board, generate a bitstream, and program the FPGA.

Part 1: Converting Celsius to Fahrenheit

In this lab, you will create an FPGA application that displays the current temperature in either Celsius or Fahrenheit on the seven-segment display, as shown in Figure 1.

The Analog Devices ADT7420 temperature sensor chip on board the Nexys A7 provides a temperature reading in Celsius formatted as a 13-bit, fixed-point number with nine integer bits and four fractional bits. Two source files are provided along with this lab to interface with the ADT7420 sensor and output the resultant 13-bit reading. These files are written in VHDL. Together, VHDL and SystemVerilog are the two mainstream hardware description languages. Vivado supports mixing SystemVerilog and VHDL modules in hardware designs! Since we cannot manufacture ADT7420 readings, your final test setup allows you to also convert values entered manually via the switches.

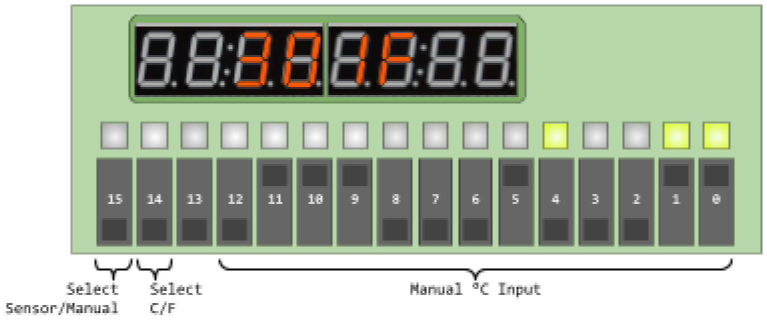

Figure 1. Nexys A7-100T Input/Output Configuration for Lab 1

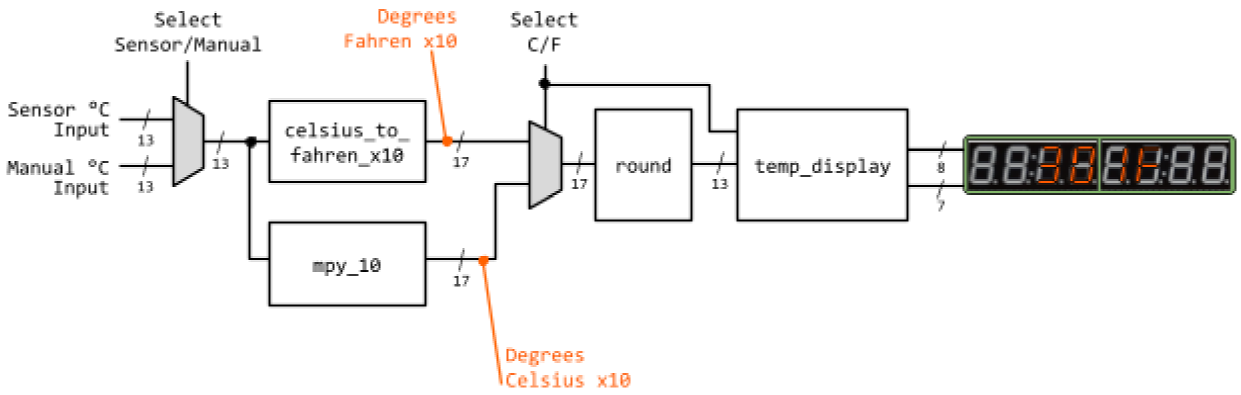

Figure 2. Module Configuration for Lab 1

The circuit should output a value in either Celsius or Fahrenheit, rounded to the nearest tenth. The provided `temperature_display` module uses the double-dabble algorithm to convert this binary value into a binary-coded decimal (BCD) to display on the seven-segment display.

Converting Celsius to Fahrenheit. The initial modules in this lab multiply the temperature reading by ten. The purpose of this is twofold: *1)* Division is costly in digital logic; Scaling by ten allows us to avoid division in the module that converts Celsius to Fahrenheit, and *2)* The double-dabble algorithm does not support fixed-point numbers; Multiplying the computation by places the "tenths" place in the integer portion of the fixed-point number, allowing us to use the double-dabble algorithm.

The formula for converting temperature to Fahrenheit is:

$$T_F = \frac{9}{5} \times T_C + 32$$

By multiplying both sides by ten, we can remove the division operator:

$$T_F \times 10 = (\frac{9}{5} \times T_C + 32) \times 10$$
$$T_F \times 10 = 18 \times T_C + 320$$

In your implementation, you will create a multiply-by-ten module (**`mpy_10`**) as well as Celsius to Fahrenheit module (**`celsius_to_fahren_x10`**) that implements the above equation. The **`mpy_10`** is used to scale the input Celsius value before rounding, whereas the **`celsius_to_fahren_x10`** scales the Fahrenheit value.

Fast Multiplication. While multiplication is not as costly as division, it is still expensive compared to other arithmetic operations. Luckily, multiplication by a constant value (such as by 10 or 18), can be implemented rather efficiently. This is because multiplication by a power of two is equivalent to shifting that value to the left by a certain number of bits. Thus, if we can express the constant as a sum of power's of two, we can implement the multiplication using bit-shifts.

For example, we can express multiplication by ten as:

$$\begin{aligned} A \times 10 &= A \times (8 + 2) \\ &= (A \times 8) + (A \times 2) \\ &= (A << 3) + (A << 1) \end{aligned}$$

The following SystemVerilog module implements this fast multiplication by ten using the SystemVerilog binary shift-left operator (`<<`):

```
module mpy_10 ( input logic  [12:0] a,
                output logic [16:0] y );

    assign y = (a << 3) + (a << 1);

endmodule
```

Temperature Format and Rounding. The format of temperature signals is initially a 13-bit fixed-point value where the lower 4-bits specify the fractional component of the number. After multiplying by 10, the entire number is expanded to 17-bits with a 4-bit fractional component, to accommodate the larger range of integer values. We can refer to this as our x10 value.

After converting the temperature value, we can use the fractional bits to perform rounding and accurately display the temperature value to the tenth of a degree. In this lab, we will use traditional rounding, where we "round up" numbers with a fractional component that is ≥ 0.5 to the next highest integer while "rounding down" numbers with a fractional component that is < 0.5 to the integer value. Note that in a fixed-point representation, the most significant digit of the fractional component represents 0.5, so we can implement traditional rounding by adding this bit to the integer component and then dropping the fractional bits, as this either will or will not increase the integer component by 1! When applying this approach to our x10 value, we get a number rounded to the nearest tenths place.

1. Create a new SystemVerilog module, **`mpy_10`**, that has a 13-bit input and a 17-bit output, and implements fast multiplication according to the above description.

2. Create a new SystemVerilog module, **`celsius_to_fahren_x10`**, that has a 13-bit input and a 17-bit output, and implements Celsius to Fahrenheit conversion (scaled by 10) according to the above description.
   a. Remember that this module is processing a *fixed-point* number! You have to consider this when specifying constant values. For example, the fixed-point decimal value for the integer 320 is `101000000.0000`. In SystemVerilog, you could write this using the concatenation operator: `{13'd320, 4'b0000}`. Alternatively, you could write this as the decimal constant `17'd5120`. However, it is clearer to use the concatenation since it makes combining the integer and fractional parts more explicit!
   b. In this module, implement multiplication ($18 \times T_C$) using the fast multiplication approach described above!

3. Create a new SystemVerilog module, **`round`**, that has a 17-bit input and a 13-bit output, and implements rounding according to the above description.
   a. Your module should round the temperature value to the nearest tenth of the degree by checking the bit in the ½'s place. Depending on the value of this bit, the next highest digit should either be incremented (rounded up) or left unchanged.
   b. After rounding to the nearest tenth place, the circuit should return the most-significant 13-bits of the number (dropping the lower four fractional bits). Since all values are multiplied by ten (x10), dropping the lower four bits still retains the tenths place in the final result!

4. Create a testbench to test each of your three modules. You may choose to test each module individually or together, as shown in Figure 2. When interpreting your module's output, remember that they are processing *fixed-point* values. This may impact how you interpret the results.

5. Run your testbench and ensure that the module's output correctly meets the specification. More information on testbenches can be found in Lab 0.

Part 2: Synthesizing the Circuit for Hardware

In Part 2 of this lab, you will connect your modules to implement the circuit shown in Figures 1 and 2. Specifically, your circuit will use **`SW[15]`** to select between reading the temperature from the sensor or the manual switch input. **`SW[14]`** will toggle between displaying the temperature in Celsius or Fahrenheit on the seven-segment display.

The provided top-level file, **`lab01_top.sv`**, instantiates the provided temperature sensor module. This module produces a 13-bit fixed point result in the signal named **`temp_sensor`**: This serves as the "Sensor °C Input" from Figure 2. The "Manual °C Input" should be taken from the switches.

```
// Instantiate the VHDL temperature sensor controller
logic tmp_rdy, temp_err;
TempSensorCtl U_TSCTL (.TMP_SCL(TMP_SCL), .TMP_SDA(TMP_SDA),
                        .TEMP_O(temp_sensor), .RDY_O(tmp_rdy),
                        .ERR_O(tmp_err), .CLK_I(clk), .SRST_I(rst));
```

This file also instantates and connects a module that displays the resultant temperature on the seven-segment display.

1. In the top-level module, `lab01_top.sv`, instantiate your temperature circuit according to the previous descriptions.
    a. The logic for the top-level module is shown in Figure 2.
    b. Your implementation should select either the wire **`temp_sensor`** or **`SW[12:0]`** as the initial temperature value in Celsius.
    c. The final, rounded result should be connected to the provided **`temperature_display`** module via the signal **`temp_final`**.

2. Generate a bitstream and program the FPGA. More information on how to connect and program the FPGA board can be found in Lab 0.

3. Check that the circuit meets the specification. Demonstrate your working implementation to the instructor.

Part 3: Push your Final Code to GitHub

Complete the lab by pushing your changes to your new Git repository.

1. Check that your code is properly indented and meets coding guidelines.

2. Add your new (or modified) files to your Git repository using the `git add` command, such as:

   ```
   git add hdl/sevenseg_ext_n.sv
   git add hdl/sevenseg_ext_top.sv
   git add constraints/sevenseg_ext_top.xdc
   ```

   Alternatively, you can choose to add *everything* in the current directory:

   ```
   git add *
   ```

   Or just add the files in your Vivado project's sources folder (ECE_212_Lab0.srcs):

   ```
   git add ECE_212_Lab0.srcs/*
   ```

3. Push this version to the cloud (origin) Git repository using the following command:

   ```
   git commit -m "Some commit/version message like Lab01 finished"
   ```

4. Push your repository back to the Git server using the command:

   ```
   git push
   ```

Lab Report *(Due before the next lab session)*

Lab Report. In collaboration with your lab partner, submit one lab report on Moodle that describes the lab activity and your team's circuit design. Your written report is worth 60% of your grade for each lab assignment. Your lab report should include the following sections:

- **Heading/Title:** Your lab report should begin with a title block that includes the course number, the title of the lab assignment, the names of each team member, and the date of the lab assignment.

- **Distribution of Work:** Describe the contributions of each team member to the lab assignment, including writing of the lab report. What is specified in this section will not affect your grade, however it is expected that you periodically rotate roles within your lab group if you do divide work (especially with regards to writing the report).

- **Introduction:** Include a brief paragraph summarizing the objectives of the lab and what your team achieved. This section should be written in your own words; it is not acceptable to copy text for this from the lab handout.

- **Procedure:** Briefly enumerate the steps you followed to complete the lab assignment. If designing a circuit, include steps involved in making the design, such as writing a truth table. Include any relevant circuit schematics in this section. This section should also contain a description of how you tested your circuit, including rationale for why you chose specific test cases if the circuit is not tested exhaustively.

- **Results:** Your results section can take many forms. This section should include any recorded measurements, tests, diagrams, or images specified by the lab document. You can include additional observation data in your lab report beyond what is specified. Each item should be numbered and labeled with a caption (e.g., "Figure 1" or "Table 1"). When including code experts in your lab report, it is recommended that these are placed in an appendix at the end of the report.

- **Discussion/Conclusion:** Your discussion/conclusion section should comment on any observations about the technique you employed to get your results. Briefly describe any difficulties that you encountered and how you resolved them. Then, summarize what you concluded from your work. (It may also be helpful to think negatively, *i.e.* what your results did not show. Remember that a negative result is often as valuable as a positive result!)

- **Questions:** A number of technical questions may be posed in the assignment sheet. If questions are given, include your answers to them in this section. If no questions are given, you do not need to include this section.

Your lab report should be typed and submitted as a .pdf document. Only one member per team needs to submit the lab report.