

ENEL 453 Lab 3: Voltage and Distance Measurement

1. Revision History

Version	Comments
v1	Initial document
v2	Included how to add ADC_Data and files to project (page 9). Updated submission requirements (page 10). All changes are in red font to highlight the changes in this document version.

Path: C:\Users\donen\Dropbox\U of C\Teaching\ENEL 453\ENEL 453 F2020\Labs\Lab 3\ENEL 453 Lab 3 v2.docx

2. Introduction

This is the third lab project of the course and its focus for the team is to interface a Distance Sensor that provides an analog output voltage and to read this analog voltage with the FPGA's Analog-to-Digital Convertor (ADC). You will extend the design of Lab 2 to add the ADC module and present its data to the 7-segment displays. The requirements will be given at a high-level, to encourage you to complete more conceptual design.

The prerequisite knowledge for Lab 3 is a solid understanding of your Lab 1 and Lab 2 designs and a solid understanding of using the Quartus and ModelSim tools and debugging.

The basic elements of this project are to:

- Review all lab and lecture videos to date.
- Develop a high-level design approach to meet the project requirements. This is most likely a block diagram of the entire design. The instructor and TAs will not be able to help you effectively without you communicating your design intentions with a block diagram.
- Incorporate the given VHDL codes for the ADC module. Design and write the new modules as required for the project and write their testbenches, including an updated version for the top level. Use both Quartus and ModelSim effectively to complete your design. Build up your design with verified modules (i.e. use unit testing), before integrating them into the top level (for system-level testing), otherwise your debugging will be painful.
- Characterize the Distance Sensor and create a look up table with values to convert voltage to distance.
- Modify the .QSF file as needed. Download your FPGA configuration to the DE10-Lite to verify the design in physical hardware.
- Be prepared to do extensive debugging, work effectively as a team, and start your work in a timely manner so that you can get help from the office hours and lab sessions.
- Upload the required videos and documents to the D2L Dropbox before the deadline.

3. Technical Requirements

The project technical requirements are as follows to be eligible to earn up to an A- grade for Lab 3.

3.1. Modify the functionality of Lab 2 by providing four modes of operation for the 7-segment displays. The modes of operation will include:

1. Hexadecimal output of SW(7 downto 0) (existing Lab 1 functionality)
2. Decimal output of distance in centimeters. This means that you must provide the decimal point in the correct location.
3. Decimal output of voltage in volts. This means that you must provide the decimal point in the correct location.
4. Hexadecimal output value of the moving average of the 12-bit ADC value.

You are expected to use testbenches for the modules you add/modify, to aid in your development.

3.2. The pushbutton that was previously used to store an output value will be modified to hold (or “freeze”) the current output value when the pushbutton is pushed down. For example, if the 7-segment displays are fluctuating due to noise in the system, pushing down the pushbutton to freeze the display will enable the user to view a static value. When the pushbutton is released the system will again work normally. This hold function will apply to all four modes of operation.

3.3. The top level must be only structural code, there must be no logic or RTL code in it. This is good design practice and will allow you to complete unit testing on the lower level components to verify that they are working before they are brought into the top level for system integration.

3.4. The system reset must be an asynchronous active-low `reset_n` as it was for Lab 1. This applies to any new modules you add to the design.

3.5. Synchronous Design: The design must be synchronous, meaning that whenever you use a clocked process, it must always be `rising_edge(clk)`. Not `falling_edge(clk)` nor `rising_edge(some_other_signal)`, etc. Further to synchronous design, all asynchronous inputs to the design must be synchronized (already covered by the synchronizer and debounce circuits). No Latches are permitted to be used in the design.

Additional requirements to be eligible to earn up to an A for Lab 3.

3.6. Blank the leading zeros from the distance display and the ADC value (i.e. modes of operation 3 and 4).

3.7. Write your own `averager256` to place a register after each adder. The existing `averager256` creates a binary adder tree to efficiently calculate the sum of all 256 values. Placing a register after each adder would pipeline the computation and increase the potential data rate through this module. Note that the existing `averager256` was written by ENEL 453 students in Fall 2019, and they did a superb job. You are encouraged to modify this code to achieve the required functionality, rather than starting from scratch. But you are free to write your own completely custom code for this module.

4. Resources Provided

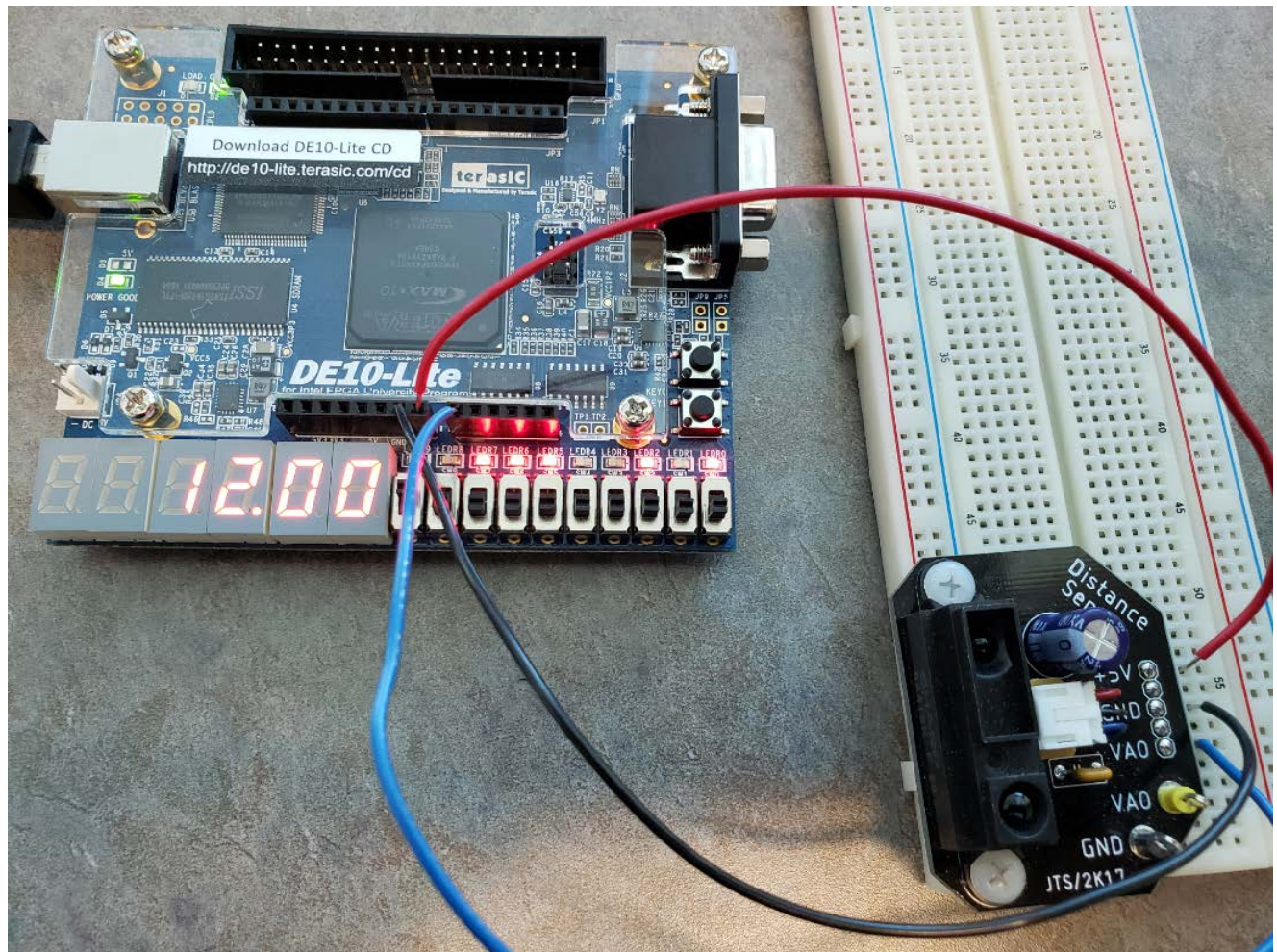
- DE10-Lite kit, Distance Sensor, prototyping kit with jumper wires and breadboard, and multimeter.
- Reference code (ADC_Data.vhd, plus lower-level modules) that demonstrates:
 - ADC interface;
 - averaging ADC readings to produce a more stable output;
 - converting ADC readings to voltage values;
 - converting voltage values to distance values;
 - a selection of the ADC module for acquiring data (for synthesis to download to the DE10-Lite board) or a simulation model for the ADC (to use in ModelSim);
 - a testbench to demonstrate the usage of ADC_Data.vhd.
- Lab and lecture videos.

5. Supporting Information

5.1. Connecting the Distance Sensor to the DE10-Lite

You will interface a Distance Sensor module to your FPGA board as shown in the pictures below and more detailed pictures follow in the next page. The Distance Sensor module has 3 connections to the Arduino header on the DE10-Lite board. These are the Distance Sensor connections:

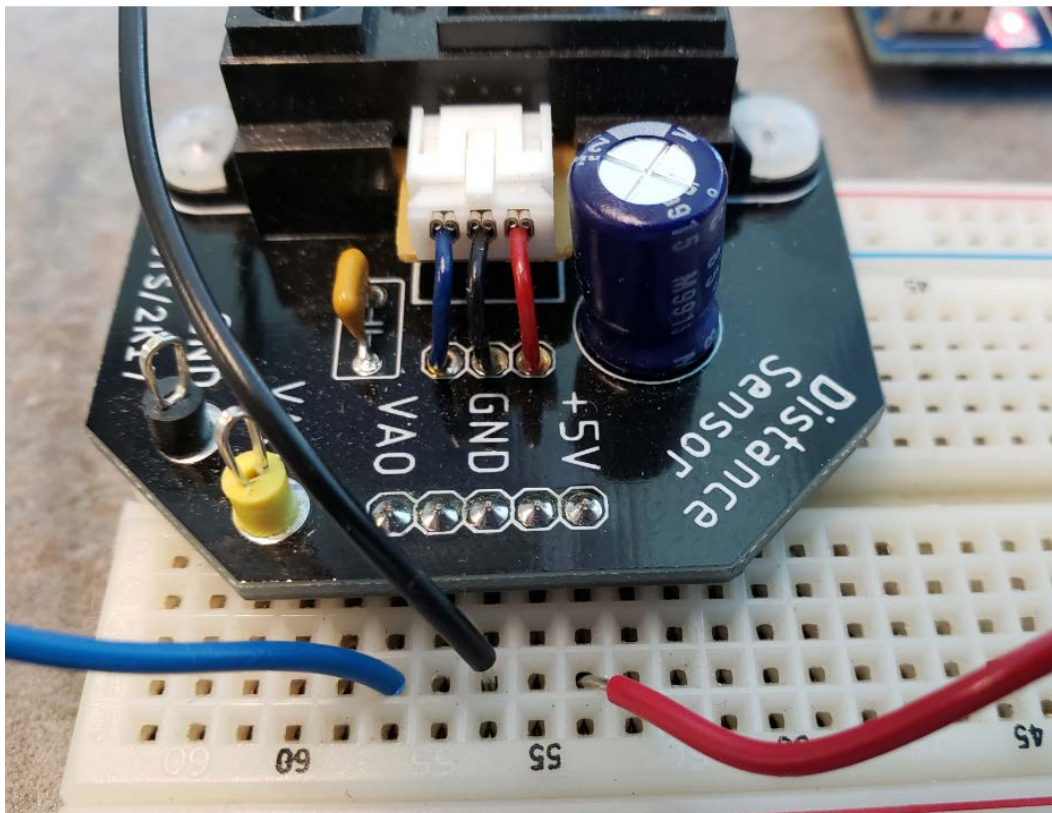
- 5 VDC supply input, labelled as **+5V** (red wire in the picture)
- Ground reference, labelled as **GND** (black wire in the picture)
- Analog voltage output, labelled as **VA0** (blue wire in the picture). This analog voltage is related to the distance measured by the sensor, as described in the next section.
- **NOTE: The distance sensor does not have circuit protection, so you must be very careful to connect it correctly or else the DE10-Lite or the Distance Sensor may be damaged. In addition, only make or modify the wired connections when the power is turned off.**



Below is a close-up picture of the connections to the Arduino header on the DE10-Lite board. The Black wire is ground (labelled **GND** on the board); the Red wire is power (labelled **5V**); and the Blue wire is the ADC input (unlabeled on the board). Note that **GND** on the DE10-Lite board header has 2 adjacent positions, both are ground so you can pick either one to make it easier to plug in.



Below is a close-up picture of the connections to the Distance Sensor. Note the space between connections.



5.2. Sharp GP2Y0A41SK0F Distance Sensor Data Sheet

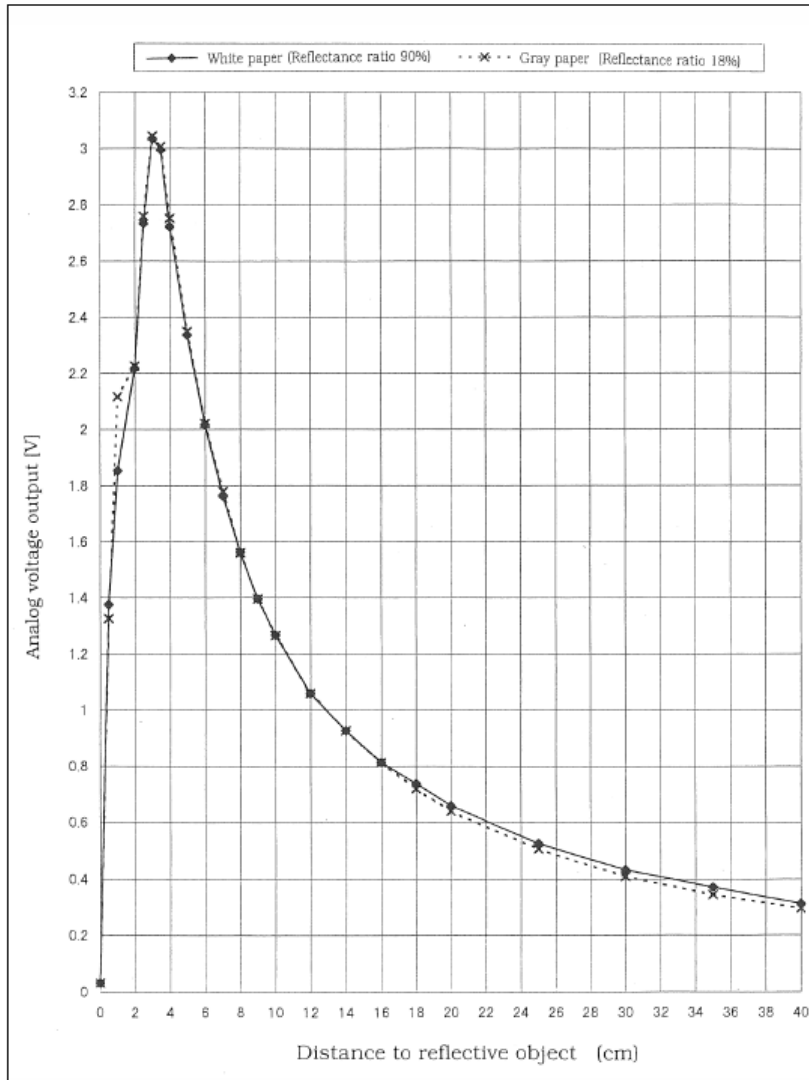
The distance measurer circuit module schematic diagram and PCB layout are provided in D2L. The module uses the Sharp GP2Y0A41SK0F sensor and its data sheet is also provided in D2L. You should review the data sheet and page 4 is the most relevant, as it shows the relationship of the analog voltage output of the sensor, to the measured distance, as shown below.

SHARP

GP2Y0A41SK0F

■Supplements

●Example of output distance characteristics



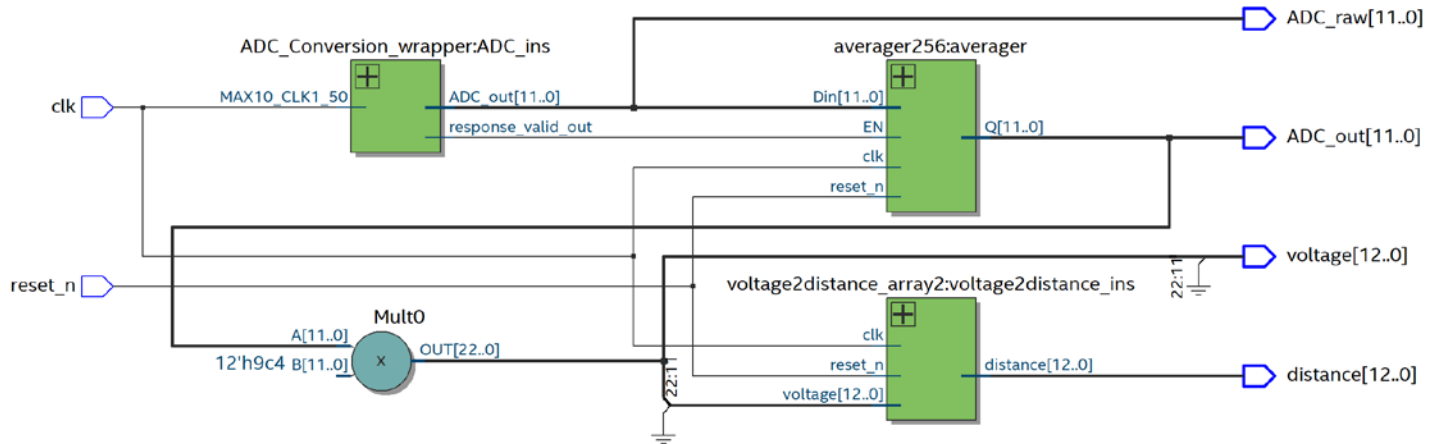
Sheet No.: OP13008EN

4

You will note that the voltage-distance relationship is non-linear, so conversion is not straightforward. You will calibrate your Distance Sensor by making experimental measurements and completing a look up table in the provided file LUT_pkg.vhd, and you may use the provided Excel file to help you with the calculations. This process will be explained in the Lab 3 video.

5.3. Provided Code – ADC_Data.vhd and its Subordinate Files

You will be provided a very useful module, ADC_Data.vhd, and its internal modules are shown below in its RTL Schematic. You will also be provided the testbench for this module, tb_ADC_Data.vhd, which will help you understand the module and its operation.



ADC_Data provides the following functionality and understanding it is aided by viewing its [Port](#) signals below:

- Instantiates the ADC primitive, which allows the FPGA to use its internal ADC.
- Provides a 12-bit ADC output for every ADC conversion (**ADC_raw** signal) of the analog input signal. This is a 12-bit unsigned number that represents the ratio of in analog input signal to the reference voltage. In simple terms, the ADC accepts an input voltage between 0V (i.e. "000000000000" or 12'h0 or 12'd0) to 5V (i.e. "111111111111" or 12'hFFF or 12'd4095). So if the input signal was 2V, the ADC value would be $\frac{2V}{5V} \times 4095 = 1638$, so this would be "011001100110" or 12'h666 or 12'd1638.
- Provides a 12-bit moving average of 256 samples of the raw ADC value (**ADC_out** signal). This smooths out the reading and reduces fluctuations due to noise, by using the averager256.vhd module to compute the moving average.
- Provides a calculation of the voltage in milli-volts (**voltage** signal). The calculation is:

$$12\text{bit ADC value} \times \frac{5000 \text{ mV}}{4096} = \text{voltage (mV)}.$$
- Converts the voltage signal to distance (**distance** signal) using the module voltage2distance_array2.vhd and its VHDL package, LUT_pkg.vhd which stores the values to convert voltage to distance.

```
entity ADC_Data is
  Port(
    clk      : in STD_LOGIC;
    reset_n  : in STD_LOGIC; -- active-low
    voltage  : out STD_LOGIC_VECTOR (12 downto 0); -- voltage in milli-volts
    distance : out STD_LOGIC_VECTOR (12 downto 0); -- distance in 10^-4 cm (e.g. if distance = 33 cm, then 3300)
    ADC_raw  : out STD_LOGIC_VECTOR (11 downto 0); -- the latest 12-bit ADC value
    ADC_out  : out STD_LOGIC_VECTOR (11 downto 0); -- moving average of ADC value, over 256 samples,
                                                    -- number of samples defined by the averager module
  );
end ADC_Data;
```

The design challenge is to integrate ADC_Data into your design and meet the design requirements. To meet the basic requirements, the only thing you need to change within ADC_Data is the values in the look up table in LUT_pkg.vhd, shown below. These values must correspond with your experimentally derived values based on your sensor.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  package LUT_pkg is
5
6  -- This array has been pasted in from the Excel spreadsheet.
7  -- In this array, the values are distances, in units 10^-4 m.
8  -- To get cm, move the decimal point 2 places to the left.
9
10 type array_1d is array (0 to 4095) of integer;
11 constant v2d_LUT : array_1d := (
12
13     ( 3794 ) , -- array index 0 (voltage = "000000000000" or 0 mV), distance output 3794 (37.94 cm)
14     ( 3792 ) , -- array index 1 (voltage = "000000000001" or 1 mV), distance output 3792 (37.92 cm)
15     ( 3791 ) ,
16     ( 3790 ) ,
17     ( 3789 ) ,
18     ( 3787 ) , -- array index 5 (voltage = "000000000101" or 5 mV), distance output 3787 (37.87 cm)
19     ( 3786 ) ,
20     ( 3785 ) ,
21     ( 3784 ) ,
22     ( 3783 ) ,
23     ( 3781 ) ,
24     ( 3780 ) ,
25     ( 3779 ) ,
26     ( 3778 ) ,
27     ( 3776 )

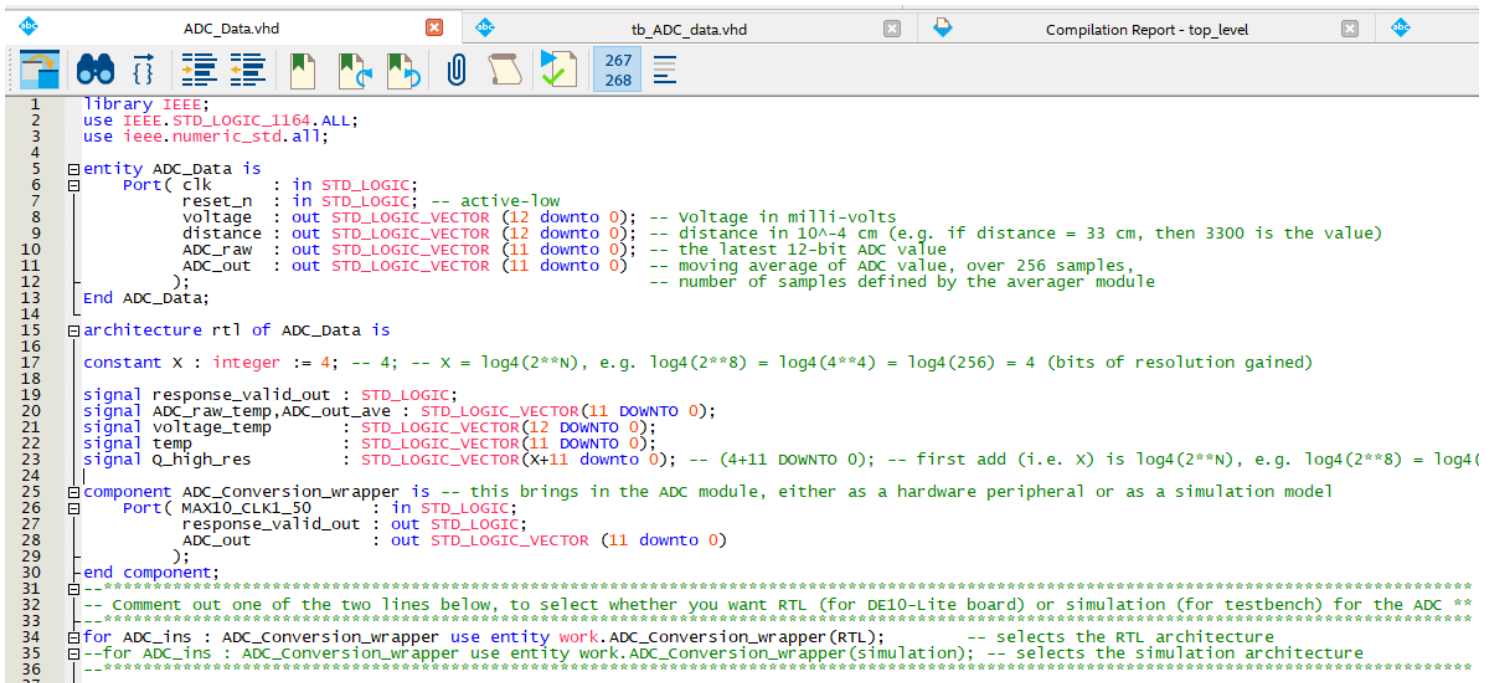
```

The comments in the look up table show how to interpret its values. The Lab 3 video will provide an overview of the ADC_Data and will explain how to obtain the experimental values for the above look up table. The basic procedure will be as follows:

1. Incorporate ADC_Data into your top_level and get your system to report the measured voltage on the 7-segment displays.
2. Create an experimental setup where you can place a target (e.g. the DE10-Lite cardboard box) at a known distance from the Distance Sensor. You may wish to use a ruler or some printed template, to find the distance between the target and the distance sensor.
3. Obtain several measurements of voltage and distance. If you use the provided Excel spreadsheet, you'll need 6 data points.
4. Compute a function that creates a polynomial fit for your data points. The provided Excel spreadsheet uses a 4th order polynomial and it is shown on the graph.
5. Use the function you obtained to compute all the values for the LUT_pkg.vhd look up table (index 0 to 4095). Note that the index of the look up table corresponds to mV and the elements of the look up table correspond to distance in 10⁻⁴ m (tenths of millimeters). Also note that the elements of the look up table (i.e. the distance values) must be positive and cannot exceed 4095 (they represent 12-bit unsigned numbers in the FPGA), so you may need to manually adjust the distance values in the look up table.
6. Copy your look up table values into LUT_pkg.vhd.

Note: Fluorescent lights, like those in ENG 130, disrupt the Distance Sensor readings. Be careful where you point the Distance Sensor when making measurements.

Note also that the module ADC_Data contains configuration statements that allow you to select one of two architectures for the ADC module (ADC_Conversion_wrapper). So, ADC_Conversion_wrapper has something you haven't yet seen, two architectures. One architecture is for synthesis and the other architecture is for simulation. By commenting out either line 34 or line 35 in the code snippet of ADC_Data below, you can select which architecture you will use for the required purpose. *This will likely be a source of lots of debugging, so be careful with it.*



```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.all;
4
5  entity ADC_Data is
6  Port( clk      : in STD_LOGIC;
7        reset_n  : in STD_LOGIC; -- active-low
8        voltage  : out STD_LOGIC_VECTOR (12 downto 0); -- voltage in milli-volts
9        distance : out STD_LOGIC_VECTOR (12 downto 0); -- distance in 10A-4 cm (e.g. if distance = 33 cm, then 3300 is the value)
10       ADC_raw   : out STD_LOGIC_VECTOR (11 downto 0); -- the latest 12-bit ADC value
11       ADC_out   : out STD_LOGIC_VECTOR (11 downto 0); -- moving average of ADC value, over 256 samples,
12               -- number of samples defined by the averager module
13   );
14   End ADC_Data;
15
16   architecture rtl of ADC_Data is
17       constant x : integer := 4; -- 4; -- x = log4(2**N), e.g. log4(2**8) = log4(4**4) = log4(256) = 4 (bits of resolution gained)
18
19       signal response_valid_out : STD_LOGIC;
20       signal ADC_raw_temp, ADC_out_ave : STD_LOGIC_VECTOR(11 DOWNTO 0);
21       signal voltage_temp          : STD_LOGIC_VECTOR(12 DOWNTO 0);
22       signal temp                  : STD_LOGIC_VECTOR(11 DOWNTO 0);
23       signal Q_high_res            : STD_LOGIC_VECTOR(x+11 downto 0); -- (4+11 DOWNTO 0); -- first add (i.e. x) is log4(2**N), e.g. log4(2**8) = log4(
24
25   component ADC_Conversion_wrapper is -- this brings in the ADC module, either as a hardware peripheral or as a simulation model
26   Port( MAX10_CLK1_50 : in STD_LOGIC;
27         response_valid_out : out STD_LOGIC;
28         ADC_out            : out STD_LOGIC_VECTOR (11 downto 0)
29   );
30   end component;
31
32   -- Comment out one of the two lines below, to select whether you want RTL (for DE10-Lite board) or simulation (for testbench) for the ADC **
33   -- ****
34   for ADC_ins : ADC_Conversion_wrapper use entity work.ADC_Conversion_wrapper(RTL); -- selects the RTL architecture
35   --for ADC_ins : ADC_Conversion_wrapper use entity work.ADC_Conversion_wrapper(simulation); -- selects the simulation architecture
36   -- ****
37

```

Working with the ADC module in Quartus.

1. Place the adc_qsys folder into the same folder where all your project's .vhd files are for Quartus.
2. Add the adc_qsys.qip file to your Quartus project, just like you would your .vhd files. The location is: adc_qsys > synthesis > adc_qsys.qip
3. Whether you are synthesizing in Quartus or simulating in ModelSim, you must comment out either line 34 or line 35 in ADC_Data.vhd, as appropriate. Please check the comments in ADC_Data.vhd for which line to use.

Note: I have added a video the shows how to make a Quartus and a ModelSim project for the ADC_Data module. You'll have to follow the basic steps described in the video, to include and use ADC_Data and its associated files, to build upon your Lab 2 project to fulfill Lab 3.

Also note that you should double-check your Chip Planner vs your .qsf, to ensure the pins assignments are correct.

6. Deliverables

The team will upload the following to the D2L Dropbox by **11:59 pm Sunday November 15**. **You must follow the submission requirements specified in D2L: Content > Labs > Lab Submission Procedure to D2L Dropbox.**

1. VHDL code for their project:
 - a. RTL code, i.e. the design code, for the complete project.
 - b. The testbenches for the following modules
 - i. your module for controlling the decimal point for the 7-segment displays
 - ii. your testbench for the module used for blanking the leading zeros (if you completed this part)
 - c. A testbench for the top_level, demonstrating the system operating in all 4 modes: Hexadecimal, Distance, Voltage, ADC, and showing the hold (i.e. freeze) and Reset behavior.
2. The .sdc and .qsf files for their project.
3. A Design Record which is a PDF document that contains the following screenshots. Please refer to the Lab 1 Design Record Example document to see the required format and instructions on how to obtain the screenshots.
 - a. Your RTL schematic.
 - b. Your Slow 1200mV 85C Model Fmax Summary – we want to see the maximum frequency of your design.
 - c. Your Messages Window.
 - d. ModelSim simulation waveforms for top_level and any other testbenches listed above.
 - e. Note, you do not have to show the useful instructions on how to obtain the screenshots, as shown in the example document. You just have to show the screenshots, the document title, and the student names.
4. Three videos of **no more than 3 minutes each**, delivered by the respective leads. All students in a team must present at least one video. You must feature your face in the video, as you are speaking. The videos must be recorded in “one-take.” No video editing, splicing, cutting, speeding up or slowing down, etc. The videos must be uploaded to the D2L Dropbox in MP4 format. You must follow the Lab Submission Procedure to D2L Dropbox, in the Labs folder in D2L. This will allow the video to be played within the D2L Dropbox player and will make it more convenient for the marker and for you. *Keep in mind that the marker is someone who knows VHDL and FPGA design, is familiar with the requirements of the project, and is familiar with a reference solution. This should help you be efficient with your presentation because you won't have to explain simple things like what is a signal assignment etc.*
 - a. Design Lead Video:

Be sure to uncomment line 34 in ADC_Data, and comment line 35, to be able to run synthesis.

 - i. Use Quartus and your face must be present throughout the presentation. You can start a Zoom session and share the screen with yourself and this should make your face visible in a small picture.

- ii. Introduce yourself and your role.
 - iii. Using Quartus: Explain the design and how it works, starting with an overview of the RTL schematic and dive into the design modules as needed, especially how you achieved the additional functionality required by this lab project. Then explain the RTL code, starting with top_level and explain the lower level modules as needed.
 - iv. Explain how the values for the LUT-pkg.vhd were calculated from the experimental data.
- b. Simulation Lead Video:
- Be sure to uncomment line 35 in ADC_Data, and comment line 34, to be able to run simulations.**
- i. Use ModelSim and your face must be present throughout the presentation. You can start a Zoom session and share the screen with yourself and this should make your face visible in a small picture.
 - ii. Introduce yourself and your role.
 - iii. Using ModelSim: Explain the testbench code, starting with tb_top_level and show and explain its simulation waveforms in ModelSim. Then repeat this explanation for any other lower level testbenches, if not adequately covered by the top_level testbench simulation discussion. Be sure to explain how the testbench exercises the design and how the simulation displays the correct functionality of the design (this is the purpose of the testbench). You may use screenshots of the other testbenches, besides tb_top_level. But your video must show you interacting with your tb_top_level simulation to explain it
- c. Implementation Lead Video:
- i. Introduce yourself and your role, while looking into the camera, so that your face is visible.
 - ii. Focus the camera on your DE10-Lite board and demonstrate the required functionality of the system, while explaining what you are doing and the results you are seeing and how it confirms the required behavior: the system operating in all the required modes:
 - 1. Hexadecimal SW(7 downto 0) outputs;
 - 2. Distance;
 - 3. Voltage;
 - 4. ADC; and
 - Hold behavior; and
 - Reset behavior.
 - iii. Show and explain your experimental setup to obtain the experimental data for the look up table for LUT_pkg.vhd.

NOTE: if you completed the additional requirements to be eligible to earn an A, i.e. blanking leading zeros in the 7-segment displays and pipelining the adders of averager256, then you may have an **additional 1-minute** per lead to explain the additional work.

7. Grading Rubric

Lab 3 will be weighted at 30% of the final course grade and it represents a significant design challenge. The grades of the 3 presentations will be averaged with equal weighting for all the students in the team and round to two decimal places.

The Design Record will be used primarily as a reference but a poor-quality Design Record will detract from the presentation grades.

7.1. Presentation Rubric

The video presentations will be assessed on a grade-point scale for a single grade out of 4.00. The grade will be interpreted from the description below.

Grade	Description
4.00 (A)	Excellent: superior performance, showing comprehensive understanding of subject matter. <i>To earn an "A" the submitted work must fully complete and fully meets the project requirements. The work is of very high quality from both the technical and communication perspectives. The work also includes the additional requirements of blanking the leading zeros in the 7-segment displays and pipelining the adders of averager256, otherwise an A will not be awarded. Completion of the additional requirements does not automatically guarantee an A, the entire submission must be of very high quality.</i>
3.70 (A-)	Excellent: superior performance, showing comprehensive understanding of subject matter. <i>The submitted work is fully complete and meets the project requirements. The work is very high quality from both the technical and communication perspectives.</i>
2.70 to 3.30 (B- to B+)	Good: clearly above average performance with knowledge of subject matter generally complete. <i>The submitted work is fully complete and meets the project requirements. The work is very high quality but has minor weakness or weaknesses either technically and/or in communications.</i>
1.70 to 2.30 (C- to C+)	Satisfactory: basic understanding of the subject matter. <i>Submitted work is complete but has significant weakness or weaknesses either technically and/or in communications. Generally indicates insufficient effort or accomplishment is moderate.</i>
1.00 to 1.30 (D to D+)	Minimal pass: marginal performance; generally insufficient preparation for subsequent labs or courses in the same subject. <i>Weak effort demonstrated by missing elements or superficially completed elements either technically and/or in communications.</i>
0.00 (F)	Fail: unsatisfactory performance or failure to meet course requirements. <i>Clear lack of effort or ability to accomplish the project requirements.</i>

Notes: Grade can be reduced down to 0.0 for not complying with requirements or for unprofessional behavior. Excessive video lengths will be penalized one letter grade (e.g. B+ to B) for every 15 second interval overlength of the stated limit.

Late penalty: one letter grade per day late (e.g. A- to B+, not A- to B-), according to the D2L Dropbox timestamp, based on the latest submission for the project. Students are responsible for retaining the D2L Dropbox submission confirmation email and for double-checking their Dropbox submission. Submissions must be fully complete to be eligible for grading of the lab project. This includes code and design files, videos, and Design Record, otherwise the late penalty will apply to the entire lab project.