LAB 6

DISTANCE MEASUREMENT USING THE IR SENSOR AND ADC

This week in lab, you will be working with the infrared (IR) sensor, programming the Analog-to-Digital Converter (ADC) module, and demonstrating data acquisition and distance measurement using the ADC and IR sensor.

BACKGROUND

The IR sensor is an electro-optical device that emits an infrared beam from an LED and has a position sensitive detector (PSD) that receives a reflected IR beam; see Figure 1. A lens is positioned in front of the PSD in order to focus the reflection before it reaches the sensor. The PSD sensor is an array of IR detectors, and the distance of an object can be determined through optical triangulation (see Figures 2-4 below). The location of the focused reflection on the PSD is translated to a voltage that corresponds with the measured distance. An IR distance sensor is designed to measure distances in a specific range.

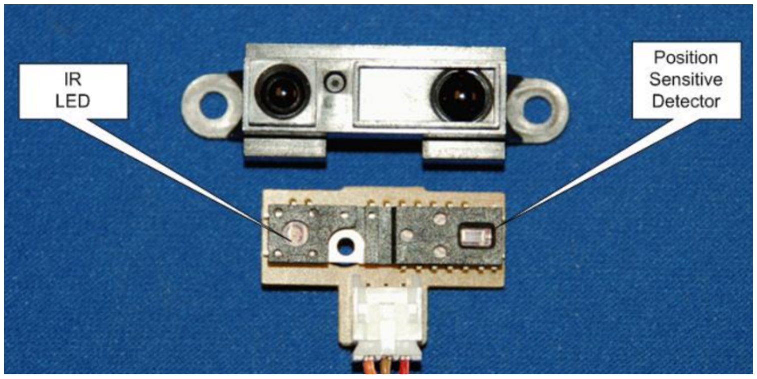


FIGURE 1: INTERNALS OF AN IR ELECTRO-OPTICAL DISTANCE SENSOR

The IR sensor used in the lab is designed for 9 – 80+ cm. However, the IR sensor can only accurately display a distance value for an object 9 – 50 cm away. As the distance increases, the voltage decreases. See the IR Sensor Datasheet to learn more about the operation of the IR sensor.

ADC (analog to digital conversion) and DAC (digital to analog conversion) allow an embedded system to interact with real-world physical systems. Physical quantities such as temperature, pressure, distance, or light are analog and represented using continuously valued signals with infinite possible values in between. In contrast, a digital signal is a discretely valued signal having a fixed precision. Since analog is a continuous value, we need a way to convert a physical analog signal into an n-bit digital signal.

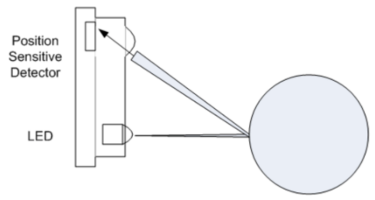


FIGURE 3: OPTICAL TRIANGULATION FOR DISTANCE OF A NEAR OBJECT.

The TM4C123 microcontroller has two 12-bit ADC modules on the chip. The ADC reads a voltage on an analog input pin, such as the voltage corresponding to the distance measured by the IR sensor. The ADC then converts this voltage into a 12-bit digital value

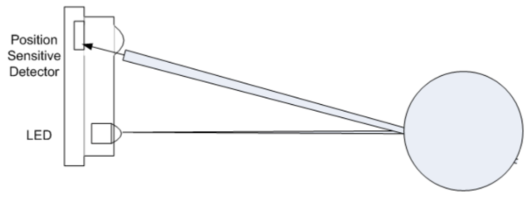


FIGURE 3: OPTICAL TRIANGULATION FOR A DISTANT OBJECT.

between 0 and 4095 (12 bits → 2^{12} values) and stores the result in an ADC register. This digital value is called a conversion result or quantized value. You may need to think about the relationship between the physical input signal and the digital results. Once you have set up the ADC to generate a digital result, you will need to calculate the distances corresponding to the digital results and improve the accuracy of the distance calculations.

Note: The following figure illustrates the use of triangulation to determine the distance to an object based on similar triangles. The IR sensor does this triangulation for you, such that its output voltage represents the distance as shown in the sensor datasheet.

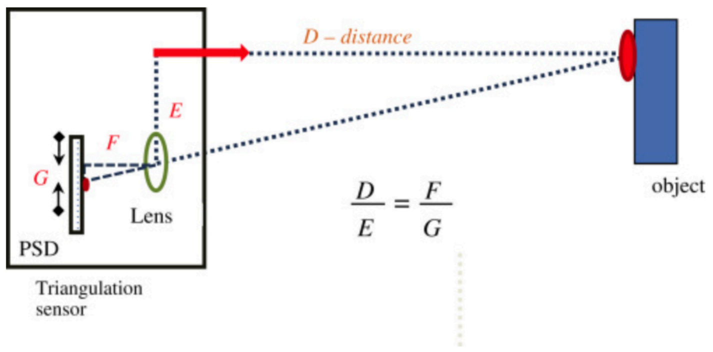


Figure 8 Principle of optical triangulation sensor. The unknown distance, *D*, is determined from the known distances *E*, *F* and the measured value of *G*—the distance to the pixel in the position sensitive detector (PSD) recording the image of the laser spot on the measured object.

FIGURE 4: PRINCIPLE OF OPTICAL TRIANGULATION.

Citation: Garry Berkovic, Ehud Shafir, "Optical methods for distance and displacement measurements," Adv. Opt. Photon. 4, 441-471 (2012); https://www.osapublishing.org/aop/abstract.cfm?uri=aop-4-4-441

REFERENCE FILES

The following reference files will be used in this lab:

- lcd.c, program file containing various LCD functions
- lcd.h, header file for lcd.c
- timer.c, program file containing various wait commands
- timer.h, header file for timer.c
- Infrared (IR) sensor datasheet: datasheet-IR-sensor-GP2D12J0000F_SS_20060207.pdf
- TI Tiva TM4C123G Microcontroller Datasheet
- TI TM4C123G Register Definitions C header file: REF_tm4c123gh6pm.h
- Cybot baseboard and LCD schematics: Cybot-Baseboard-LCD-Schematic.pdf
- GPIO and ADC register lists and tables: GPIO-ADC-registers-tables.pdf

The code files are available to download from a Google Drive folder:
https://drive.google.com/drive/folders/1LPneqbbOkbS_1rPpEeJsEhUi4gdsLAk5?usp=sharing

In addition to the files that have already been provided for you, you will need to write your own **adc.c** file and **adc.h** file and the associated functions for setting up and using the ADC module. Separate functionalities should be in separate functions for good code quality and reusability. This means that in your adc.c file you should write separate functions for initializing/configuring the ADC and taking an ADC sample. Remember to use good naming conventions for function names and variables. For example, you may want to name your ADC initialization function **adc_init** and name your function to take samples **adc_read**. Minimally, we recommend defining the following functions:

```
void adc_init (void);
int adc_read (void);
```

Note: No code has been provided for these functions. Examples are provided in course resources.

Prelab

See the prelab assignment in Canvas and submit it prior to the start of lab.

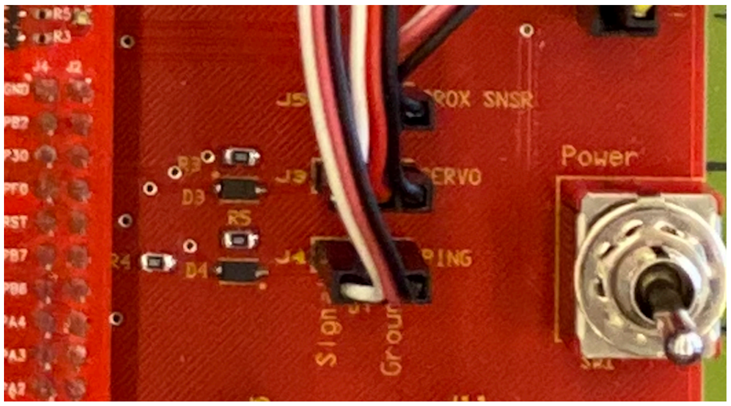
Structured Pairing

You are expected to continue to use structured pairing in this lab and in future labs. It was introduced in Lab 2.

Part 1: Initializing the ADC and Displaying Quantized Value

Write a program for the ADC that will initialize and configure the registers needed to sample the analog voltage from the IR distance sensor. Print the digital result (quantized value) to the LCD screen.

Double-check that the IR sensor is connected on the CyBot board, as shown below. The connector labeled PROX SNSR is for the IR sensor. The white wire connects to the IR sensor signal, the black wire connects to Ground, and the red voltage supply wire is in the center.



The ADC has 12 analog input channels, which use pins from Ports B, D, and E. In lab, we will use AIN10 on PB4. See Table 13-1 in the Tiva datasheet for all channels. There are two ADC modules, and you can choose to use either one (ADC0 or ADC1). Each ADC module also has four Sample Sequencer units to use (SS0, SS1, SS2, SS3). Each Sample Sequencer unit (SSn) has its own set of registers to initialize and control a conversion. An ADC module, Sample Sequencer and several registers are shown in the figure to the right.

The ADC Module Block Diagram from the datasheet is shown below and depicts all ADC registers for SS0 to SS3. As you are setting up the registers for the ADC, you may find these diagrams helpful.

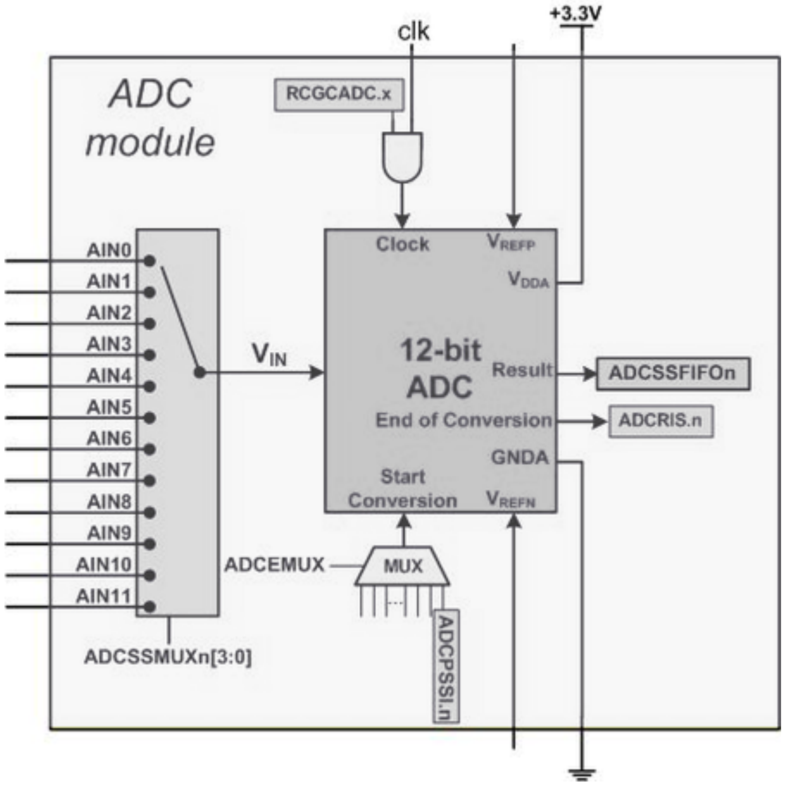
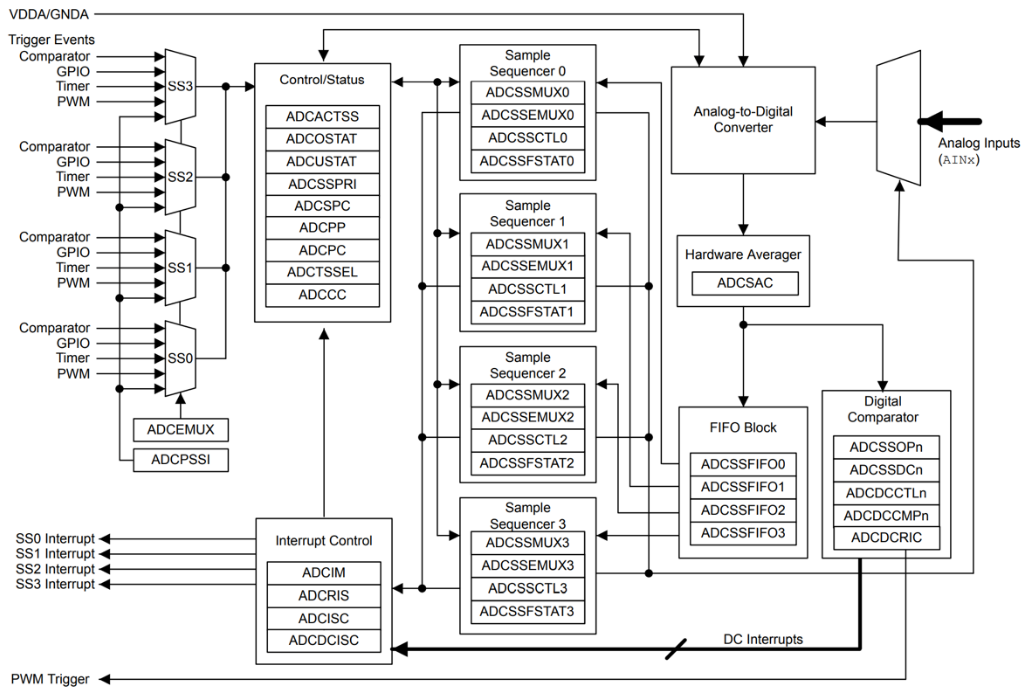


Figure 13-2. ADC Module Block Diagram



Below is the list of registers for the ADC module with their datasheet and C macro names. For ADC#, # is replaced with either 0 or 1. For SS…#, # is replaced with 0, 1, 2, or 3. You choose the number to use.

ADC Active Sample Sequencer (ADCACTSS): ADC#_ACTSS_R
ADC Raw Interrupt Status (ADCRIS): ADC#_RIS_R
ADC Interrupt Mask (ADCIM): ADC#_IM_R
ADC Interrupt Status and Clear (ADCISC): ADC#_ISC_R
ADC Event Multiplexer Select (ADCEMUX): ADC#_EMUX_R
ADC Processor Sample Sequence Initiate (ADCPSSI): ADC#_PSSI_R
ADC Sample Sequence Input Multiplexer Select (0-3) (ADCSSMUX0-3): ADC#_SSMUX#_R
ADC Sample Sequence Control (0-3) (ADCSSCTL0-3): ADC#_SSCTL#_R
ADC Sample Sequence Result FIFO (0-3) (ADCSSFIFO0-3): ADC#_SSFIFO#_R
ADC Clock Configuration (ADCCC): ADC#_CC_R
ADC Run Mode Clock Gating Control (RCGCADC): SYSCTL_RCGCADC_R
ADC Peripheral Ready (PRADC): SYSCTL_PRADC_R

Your program should be set up to repeatedly take samples of the analog input from the IR sensor. The displayed quantization values will be dynamic based on conversion results produced by the ADC. You will notice quite a bit of variability in the quantized value even when the CyBot and object are stationary. This is because of noise inherent with ADC operation. You may want to slow down how fast samples are taken as well as how frequently the display is updated.

Chapter 13 in the Tiva Datasheet contains sections on signal descriptions, functional descriptions, and initialization and configuration steps, as well as the register descriptions. The ADC is a complex subsystem with many options. You do not need to read everything in these sections. Look for basic concepts and overviews first, ignore things that seem outside the scope of the lab (whether or not it may be needed later, ignore it for now, such as digital comparators or differential mode). Browse the steps given for initialization, sample sequencer configuration, etc. Various relevant information is also provided in the Valvano and Yerraballi book, Chapter 14: Analog to Digital Conversion.

CHECKPOINT:

Display quantized values from the ADC (raw digital conversion results). Be ready to explain whether the quantized values appear to be valid. Do not calculate the distance for this part. You will calculate and calibrate the distance measured in the next part.

PART 2: CALCULATING AND CALIBRATING DISTANCE MEASUREMENT

In Part 1, you configured the ADC to generate a quantization value based on the distance of an object. Due to the nonlinear sensor operation, there is not a simple linear transfer function between the distance being measured and the analog voltage output of the sensor. Thus, you will need to implement a technique to map quantization values to distance values. To do this, you will need to take some measurements. Your task is to accurately display a distance value for an object 9 – 50 cm away. In order to calibrate the sensor, you will need to collect several data points consisting of the known distance and the quantized value at that distance. Based on these data points, you can create a lookup table or find an equation for a best fit line/curve. Your method will return an estimated distance value for a given quantization value. A requirement is to be accurate to 1 cm: the estimated distance calculated by your program should be within 1 cm of the actual distance. Print to the LCD both the quantization value and the estimated distance in cm.

If you are using an equation for the estimated distance value, depending on the equation, you may want to include the standard C math library (math.h) to get access to floating point math functions.

You should also reduce the variability in values seen in Part 1 by averaging multiple samples. You can use either hardware averaging or software averaging to collect and average 16 samples to get a more stable sensor value. There is a good description of the hardware sample averaging circuit in the ADC in Tiva Datasheet section 13.3.3. With hardware averaging, the averaging is built into the hardware of the ADC module; the programmer configures the hardware, and it returns the average in the FIFO result register. With software averaging, the programmer writes a loop in their program to compute the average over consecutive samples. The variability observed in Part 1 can be reduced by averaging multiple samples and treating the average value as the estimated distance value. Use an averaging mechanism in your program.

CHECKPOINT:

Print to the LCD both the quantization value and the estimated distance in cm. Estimated distance values should be within 1 cm of actual values. Additionally, implement an averaging mechanism and be ready to explain your approach and its effect.

PART 3: VISUALIZING DATA

Now that you have calibrated and smoothed the distance measurements, send the quantized values and distance values to the PC using PuTTY similar to Lab 4. You can output this data to a file and graph it. The output data should be formatted in PuTTY.

Optional for this lab but useful for the lab project, you may want to think of a way to more efficiently perform a recalibration of the IR sensor measurement. IR sensors are bot dependent, and readings may also be affected by the operating environment, meaning that sensor operation and data may vary. Thus measurements may need to be recalibrated, even if you are using the same bot that you previously worked with. Additionally, sometimes the sensors need to be replaced due to wear and tear or failure, thus it is important to verify that distance measurements you are receiving are accurate.

One recommended way to more efficiently recalibrate is to build off of the knowledge you have from data analysis and the Open Interface API, and write a simple function to move the CyBot back from the wall (a known distance) and send back information about the known distance and quantized value as the CyBot moves. You can graph this data, and more quickly update a distance calculation without having to manually move the CyBot and measure each point. This is not the only way you can perform an efficient calibration. You are encouraged to come up with additional ideas for bonus points in your project.

CHECKPOINT:

Output data using PuTTY and display a graph. Explain any new calibration methods.

DEMONSTRATIONS:

1. **Debug demo using debugging tools to explain something about the internal workings of your system –** The TA will announce any specific debugging requirements at the start of lab; otherwise you will create your own debug demo based on your needs and interests in the lab.
2. **Q&A demo showing the ability to formulate and respond to questions –** This can be done in concert with the other demos.
3. **Functional demo of a lab milestone –** The TA will announce the specific milestone at the start of lab. The milestone is based on the checkpoints.