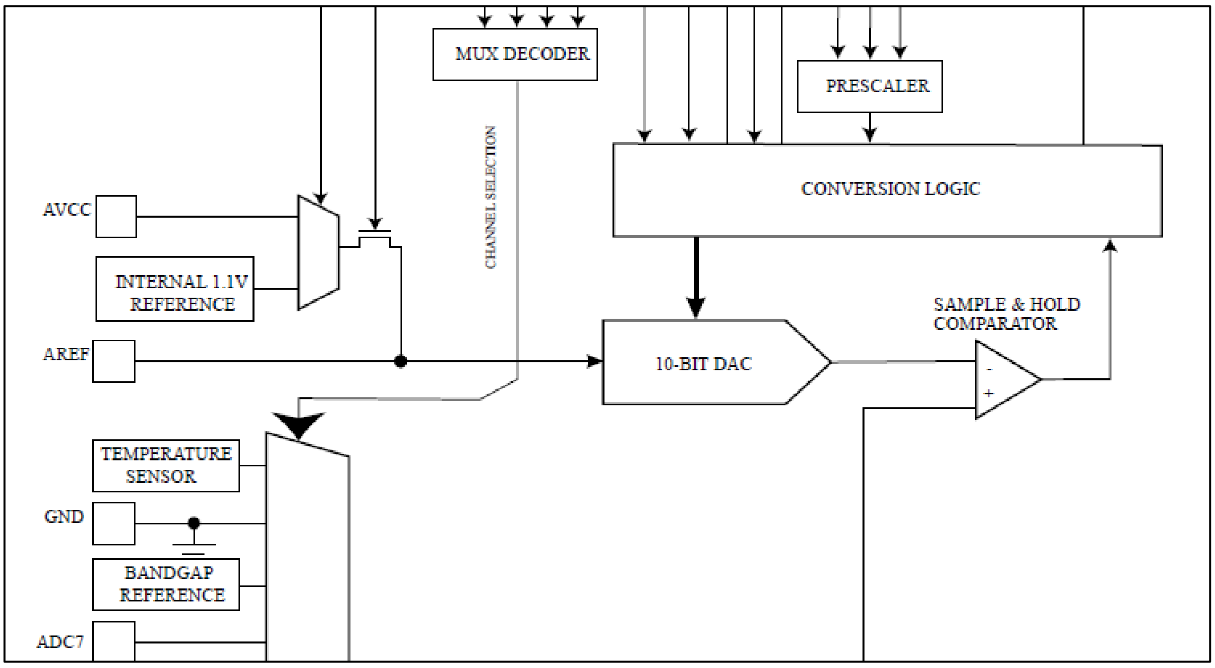
CG1112 Engineering Principle and Practice II
Tutorial 4 Suggested Solutions
ADC

Question 1.

In the studio, we saw the need to scale the 10-bit result to 8-bits in order to update the OCR register value. One approach commonly used was to divide by 4, which is to effectively discard the last 2-bits of the result. Describe how we could have obtained this result directly from the ADC operation without having to do any other computation. [Hint: Data sheet is your friend!]

Question 2.

The ADC module in the AT328 is a Successive-Approximation ADC. You might recall this name from EPP1. The block diagram below shows the part of the ADC module that does this.



(a) Describe the operation of the Successive Approximation ADC module.

(b) Suppose that the input analog voltage is 3.6V. Describe how the output will be generated based on the operation of the ADC module. You can use a table similar to below for your working:

Condition	**Comparator O/P**	**Bit Status**	**Result**
Is 3.6 >= 2.5 ?	**Yes**	**1**	**Retain -> 2.5**
Is 3.6 >= 2.5+2.5/2 = 3.75 ?			

(c) How would you have calculated that value directly without going through bit-by-bit?

<u>Question 3.</u>

You decide to use an external ADC module (16-bit) for your project. What are the important factors to consider before choosing the ADC module?

<u>Question 4.</u>

In this question we will explore the idea of checksums (see Week 2 studio 2 lecture slide 11).

a. Derive the checksum for the following sequence of bytes:

   0A 1C 42 3A

b. Explain how to use this checksum to check for errors.

c. The sequence in part a. was sent out by the transmitter but the receiver instead received:

   09 1C 41 3A

   Derive the checksum for this new sequence.

d. From your answer in c., what is the main weakness of checksums?

<u>Question 5.</u>

We learned quite a bit of serial communication and communication protocol. For this question, we are going to look at another source to reinforce our understanding. As you know (hopefully), the RPLidar unit uses serial communication too! Let us “dig around” in the source code to learn more.

Please refer to the sdk source code given in week 7 studio 2, i.e. **rplidar_sdk_v1.5.7.zip**. Pay attention to the following two subfolders once you unzipped it:

* **sdk/sdk/include** : Important defines and data type declarations
* **sdk/sdk/src**: Implementation of the rplider driver

Although rplidar driver is written in C++, the code is still largely understandable to a C programmer. Try to glean the key logical steps instead of worrying too much about unfamiliar syntax.
Answer the following questions:

Hints:

* Start from **src/rplidar_driver.cpp**, look for the relevant “functions”, e.g. **getDeviceInfo()** and **getHealth()**

* Find out the relevant definitions from include**/rplidar_protocol.h**, **inlucde/rplidar_cmd.h** and other header files in that folder.

a. Describe the steps required to get the device information from the rplidar unit. Focus on the message format and meaning of the fields of the messages exchanged.

b. Similarly, describe the steps required to get the health information from the rplidar unit.

c. [Optional – not discussed] Find out how the scan data are retrieved from the rplidar unit.