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!]

Answer:

The default setting for the result is to the hold the lower 8-bits in ADCL and the upper 2-bits in ADCH. This is called the Right-Justified mode. The ADLAR bit in the ADMUX register can be used to change it to a Left-Justified mode.

Name: ADMUX
Offset: 0x7C
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

The TWO modes of data presentation are shown below:

Right Justified

Bit	15	14	13	12	11	10	9	8
ADCH	-	-	-	-	-	-	10th bit	9th bit
Bit	7	6	5	4	3	2	1	0
ADCL	8th bit	7th bit	6th bit	5th bit	4th bit	3rd bit	2nd bit	1st bit

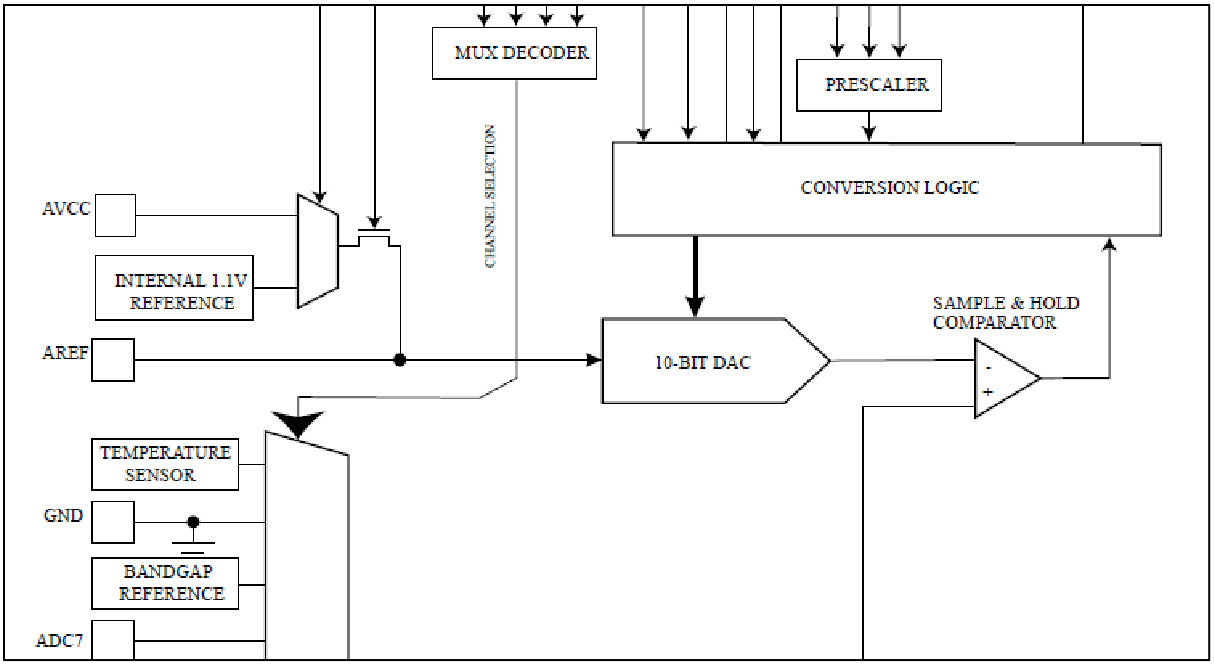
Left Justified

Bit	15	14	13	12	11	10	9	8
ADCH	10th bit	9th bit	8th bit	7th bit	6th bit	5th bit	4th bit	3rd bit
Bit	7	6	5	4	3	2	1	0
ADCL	2nd bit	1st bit	-	-	-	-	-	-

By choosing Left-Justified mode and reading only ADCH, we can immediately get the 8-bit value (“scaled”) value of the ADC result.

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.

Answer:

When the controller initiates and sends an ADC START signal to ADC unit, the Sample & Hold acquires and holds the sample analog voltage from the input. The conversion logic generates a binary code with all bits cleared and the MSB set to '1'. This digital code is fed into the DAC which outputs an analog equivalent of the digital code. In this case, it will be 2.5V. If the Analog Value is greater than the current DAC output, the bit is retained. If not, the bit is cleared. We then proceed to the next bit and repeat the whole process till we reach the LSB.

(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 ?			

Answer:

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	No	0	Discard -> 1.25
Is 3.6 >= 2.5+0.625 = 3.125	Yes	1	Retain -> 0.625
Is 3.6>= 3.125 + 0.3125 = 3.4375	Yes	1	Retain -> 0.3125
Is 3.6>= 3.4375 + 0.15625 = 3.59375	Yes	1	Retain -> 0.15625
Is 3.6>= 3.59375 + 0.078125 = 3.671875	No	0	Discard -> 0.078125
Is 3.6>= 3.59375 + 0.0390625 = 3.6328125	No	0	Discard -> 0.0390625
Is 3.6>= 3.59375 + 0.01953125 = 3.61328125	No	0	Discard -> 0.01953125
Is 3.6>= 3.59375 + 0.009765625 = 3.603515625	No	0	Discard -> 0.009765625
Is 3.6>= 3.59375 + 0.0048828125 = 3.5986328125	Yes	1	Retain -> 3.5986328125

The digital equivalent of analog input from the ADC module is 0b1011100001.

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

Answer:

(3.6 / 5) * 1024 = 737.28 -> 737 -> 0b1011100001

Question 3.

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?

Answer:

There are several factors, but most important would be the mode of communication. With 16-bit data, it would be impractical if the module provided the data in parallel format. That would take up most of the pins on the Uno and leave little for other functions. We have explored the USART block in an earlier studio. There are other Serial Interface options like SPI and I2C that are not taught here, but you will get a chance to explore them in other modules. Other factors include things like, Power Consumption, Form-Factor, Cost, etc.

Question 4.

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

0A:	0000 1010
1C:	0001 1100
XOR:	0001 0110
42:	0100 0010
XOR:	0101 0100
3A:	0011 1010
XOR:	0110 1110

   Checksum is 6E

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

   The checksum 6E is attached to the byte stream to get:

   0A 1C 42 3A 6E

The receiver will repeat the computation over the received bytes and compare the result with the checksum. If the computed result does not match the attached checksum, an error has occurred.

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.

09: 0000 1001
1C: 0001 1100
XOR: 0001 0101
41: 0100 0001
XOR: 0101 0100
3A: 0011 1010
XOR: 0110 1110

Checksum is 6E

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

Checksums can fail to detect errors if there is an even number of errors in a column.

Question 5.

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.

Method getDeviceInfo() at line 179, rplidar_driver.cpp

- Send to RPlidar a command message (_rplidar_cmd_packet_t, line 55, rplidar_protocol.h) with the following fields:

syncByte (1 byte)	cmd_flag (1 byte)	Size (1 byte)	Data (x bytes)
0xA5 (line 39, rplidar_protocol.h)	0x50 (line 50, rplidar_cmd.h)	Not used in this command	Not used in this command

- Receive a response message header (rplidar_ans_header_t, line 63, rplidar_protocol.h)

syncByte1 (1 byte)	syncByte2 (1 byte)	Size_q30_subtype (4 bytes)	Type (1 byte)
0xA5	0x5A	30 bits to represent size of additional response data 5 bytes for device info	0x4 (line 88, rplidar_cmd.h)

- Receive the device info message (rplidar_response_device_info_t, line 149, rplidar_cmd.h)

Model (1 byte)	Firmware_version (2 bytes)	Hardware_version (1 byte)	SerialNumber (1 byte)

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

Mostly similar to (a).

- Send to RPlidar a command message (_rplidar_cmd_packet_t, line 55, rplidar_protocol.h) with the following fields:

syncByte (1 byte)	cmd_flag (1 byte)	Size (1 byte)	Data (x bytes)
0xA5 (line 39, rplidar_protocol.h)	0x52 (line 51, rplidar_cmd.h)	Not used in this command	Not used in this command

- Receive a response message header (rplidar_ans_header_t, line 63, rplidar_protocol.h)

syncByte1 (1 byte)	syncByte2 (1 byte)	Size_q30_subtype (4 bytes)	Type (1 byte)
0xA5	0x5A	30 bits to represent size of additional response data 3 bytes for health info	0x6 (line 89, rplidar_cmd.h)

- Receive the device info message (rplidar_response_device_health_t, line 156, rplidar_cmd.h)

Status (1 byte)	Error_code (2 bytes)

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