NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

SEMESTER II AY2019/20

FINAL ASSESSMENT FOR

CG1112: ENGINEERING PRINCIPLES AND PRACTICE II

April 2020 Time Allowed: 40 MINUTES

INSTRUCTION TO CANDIDATES

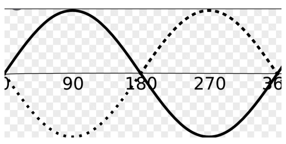
1. There is a quiz created for you on LumiNUS called "CG1112 Final Assessment". Answer all your questions here.
2. **As a backup,** please also fill in the given excel file and email to cg1112.group2@gmail.com
3. You can seek for clarifications (a Google form) here. Please open this form in your browser.
4. Answers to clarifications will be published at this document (a Google Doc) here. Please open this document in your browser.
5. This assessment paper consists of ELEVEN (11) questions.
6. This assessment paper comprises THIRTEEN (13) printed pages including this front page.
7. Question 1 is mandatory but carries no marks. If you do not answer this question your assessment will not be graded.
8. Questions 2 to 11 are MCQ and carry 2 marks each. There is no penalty for a wrong answer.
9. There are no MRQ questions. This paper is worth 20 marks.

1. Choose option b for this question. Do not choose anything else.

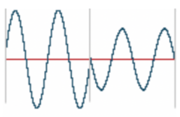
a. Do not choose this. You will get 0 for the assessment.
b. Choose this.
c. Do not choose this. You will get 0 for the assessment.
d. Do not choose this. You will get 0 for the assessment.
e. Do not choose this. You will get 0 for the assessment.

2. We have a communication system with a baud rate of 9600, communicated as a sine wave that can be transmitted at four voltage levels (1.25v, 2.5v, 3.75v, 5v) and tw0 phase shift angles (0, 180 degrees). What is the maximum bit rate (in bits per second or bps) achievable on this system? The diagram below shows examples of signals at a 0 degree and 90 degree phase shift, and one example of some data being transmitted at different voltage levels and phase shift angles:

0 degrees and 90 degree phase shift examples:



Example data transmission



a. 1200 bps
b. 9600 bps
c. 38400 bps
d. 19200 bps
e. 28800 bps

Commented [CT1]: Four voltage levels and two phase angles = 8 possibilities, representing 3 bits. 9600 x 3 = 28,800 bps.

3. The following code fragments are used to test the endianness of a 32-bit machine. Which ONE of these fragments is correct?

a. Solution 1:

```
int test = 0x01000000;
if(*((char *) &test) == 1) {
    // Machine is little endian
}
else {
    // Machine is big endian

}
```

b. Solution 2:

```
int test = 0x01000000;
if(*((char *) &test) == 0) {
   // Machine is big endian
}
else {
   // Machine is little endian

}
```

c. Solution 3:

```
int test = 1;
if(*((char *) &test) == 1) {
   // Machine is big endian
}
else {
   // Machine is little endian

}
```

d. Solution 4:

```
int test = 0x01000000;
if(*((char *) &test) == 1) {
   // Machine is big endian
}
else {
   // Machine is little endian

}
```

e. Solution 5:

```
char test = 1;
if(*((char *) &test) == 1) {
   // Machine is big endian
}
else {
   // Machine is little endian

}
```

Commented [CT2]: In a big-endian machine, 0x01000000 would be stored as 01 00 00 00 (smallest address on the left). Hence taking the first byte will give us a 1.

4. There is only one UART port on the ATMega328P, and one way around this limitation is to use a “soft serial” solution – software that simulates a UART connection. In the following code we use the Arduino timers to simulate a serial connection. The global variable *byteToSend* contain the byte to be sent, and *byteReceived* contains the byte received from the sender. We assume that this code is used to send a single byte between two 16 MHz Arduino UNOs. We further assume that all C statements take negligible time to execute.

```
// Global variable containing the byte to send. This byte is set elsewhere
// in the code
unsigned char byteToSend;

ISR(TIMER1_COMPA_vect) {
       static int bitsSent = 0;

       bitsSent = (bitsSent + 1) % 9;

       if(bitsSent) {
              if(byteToSend & 0x1)
                     digitalWrite(4, HIGH);
              else
                     digitalWrite(4, LOW);

              byteToSend = byteToSend >> 1;
       }
       else {
                     // Pull line high
                     digitalWrite(4, HIGH);

                     // Turn off Timer 1
                     TCCR1B = 0x0;
              }
}

char byteReceived;

ISR(TIMER2_COMPA_vect) {
       static int bitsReceived = 0;
       static char bitToWrite = 0x1;

       bitsReceived = (bitsReceived + 1) % 9;

       if(bitsReceived) {
              if(digitalRead(3) == 0)
                     byteReceived &= ~bitToWrite;
              else
                     byteReceived |= bitToWrite;

              bitToWrite = bitToWrite << 1;
       }
       else
              TCCR2B = 0x0; // Turn off Timer 2.
}
```

```
// Send a byte
void sendByte(unsigned char byte) {

       byteToSend = byte;

       // Pull line low
       digitalWrite(4, LOW);
       delayMicroseconds(500); // Wait 0.5 ms

       /*
          Code to set up and start Timer 1 to trigger COMPA_vect every millisecond
          omitted for brevity
       */
}

// Receive a byte. Called by code expecting a byte
// to come in. Assume that there is some suitable mechanism
// that prevents byteReceived from being read until all
// bits are read in.
void receiveByte() {

       while(digitalRead(3));
       delayMicroseconds(1500); // Wait 1.5 ms

       /*
          Code to set up and start Timer 2 to trigger COMPA_vect every millisecond
          omitted for brevity
       */
}
```

We make the following statements about the code. We assume that all C statements take negligible time to execute.

i. The bits sent are sampled at the end of each bit period.

ii. The bit rate of this soft serial solution is 1000 bps.

iii. The bits sent are received in reverse order.

iv. Pin 3 is used for TX and 4 for RX lines.

v. The GND line between two Arduinos must be connected.

Which states is/are true?

a. i., iii., and iv. only are true.
b. ii., iii. and v. only are true.
c. i., ii. and v. only are true.
d. i., ii. and iii. only are true.
e. +ii., iii and iv. only are true.

Commented [CT3]: True. The sender starts sending 0.5 ms after pulling line 3 low and sends 1 bit every ms, and the receiver starts sample 1.5 ms later, and then 1 ms thereafter. So it at the end of the bit

Commented [CT4]: True because the timers are set to trigger the ISRs every ms.

Commented [CT5]: False. Sender sends bits from right to left, and receiver reassembles them from right to left also.

Commented [CT6]: False, sender is toggling on line 4 and receiver is reading on line 3.

Commented [CT7]: True they need to have a common ground.

Commented [CT8]: Correct answer

5. Given the following program of a data structure that is serialized and sent to a destination computer. Both source and destination computers are 16 bit machines.

Sender and receiver side:

```
typedef struct t {
      char c;
      int y;
      int z;
} TData;
```

Sender side:

```
TData s;
s.c = 15; // A char is just an 8-bit integer, so we can do this.
s.y = 0x1F2D
x.z = 0x22FC

// Buffer is an array that is sufficiently large to accommodate x.
memcpy(buffer, &s, sizeof(s));

// Send out the contents of the buffer
sendData(buffer);
```

Receiver side:

```
TData r;

recvData(buffer); // Get data from sender
memcpy(&r, buffer, sizeof(r))
```

When run, the receiver gets the following:

r.c	15
r.y	002D22
r.z	0xFC2F

Commented [CT9]: We get this because of the char datatype.

Commented [CT10]: The compiler on the receive end assumes that there is padding for r.c, and therefore ignores the 1F from 1F 2D. It reads 2D from r.y then the 22 from r.z, then in r.z it reads the remaining FC, and a random adjacent byte (here it happens to be 2F)

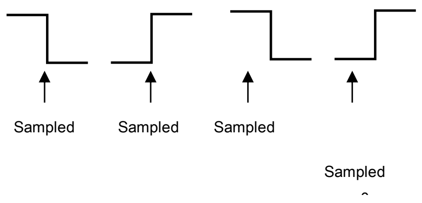
Which ONE of the following statements is true?

a. Sender compiler automatically pads structure elements to 16 bits, receiver compiler does not. Sender and receiver have the same endianness.
b. Receiver compiler automatically pads structure elements to 16 bits, sender compiler does not. Sender and receiver have the same endianness.
c. Sender compiler automatically pads structure elements to 16 bits, receiver compiler does not. Sender and receiver have the different endianness.

Commented [CT11]: Correct answer.

d. Receiver compiler automatically pads structure elements to 16 bits, sender compiler does not. Sender and receiver have the different endianness.
e. None of the above statements a. to d. are true.

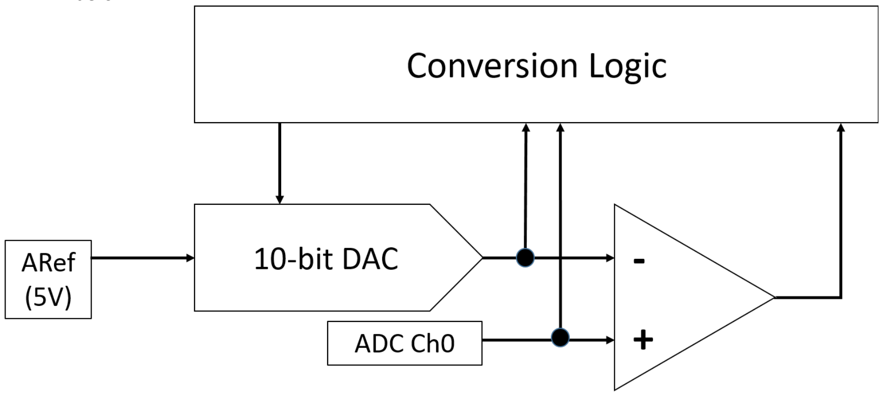
6. We send the bit pattern 0b11011 over a 9600 bps connection. If the receiving side was set at 19200 bps, what would it receive? (Ignore the presence of start, stop and parity bits. Assume that the receiver always samples in the middle of a bit). When the receiver samples, it takes the value at the very instance of sampling, as shown here (the vertical arrow indicates receiver's sampling point):



a. 0b00100
b. 0b0100111010
c. 0b1011000101
d. 0b1111001111
e. 0b10110

Commented [CT12]: Correct answer. Important point is that bit rate corresponds to time, and you can use a 2 cm scale to represent time in 9600 bps and 1 cm scale to represent time in 19200 bps. Sketch the waveform for 0b11011 on the 2 cm scale, then take samples using 0.5 cm marks in the 1 cm scale (i.e. at 0.5, 1.5, 2.5, 3.5 cm marks, etc)

7. The 10-bit ADC Module of the AT328P is upgraded slightly as shown in the figure below.



In the modified circuit, the DAC output and the ADC signal are directly fed into the Conversion Logic. The Conversion Logic continuously monitors the difference between these values. Every-time a bit is updated in the 10-bit result, and the difference between the DAC value and ADC input is less than the ADC resolution, the conversion is immediately stopped regardless of the Op-Amp output. Every bit change in the output and its corresponding comparison with the input requires 0.25 μs.

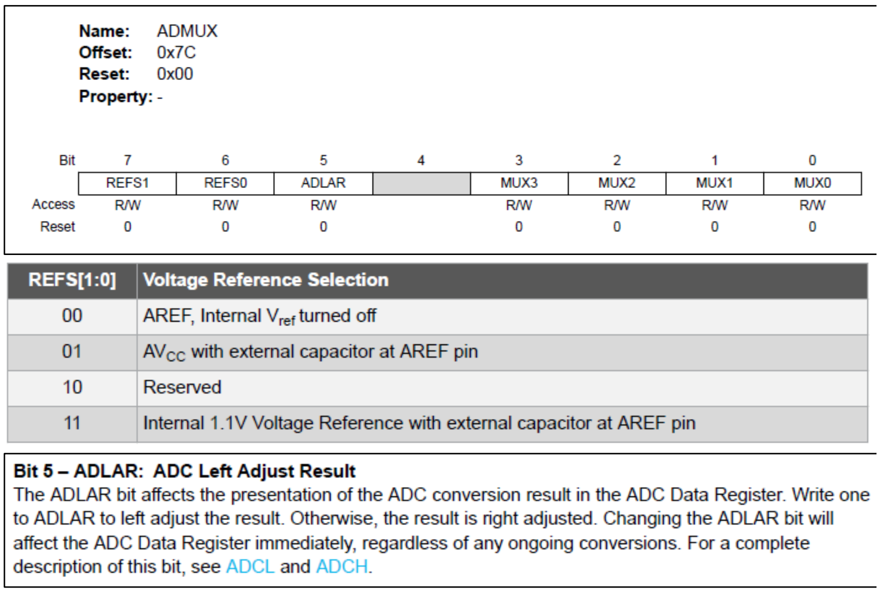
A fixed analog voltage of 1.76V is applied to ADC Ch0. How much time does it take for the upgraded ADC to complete its conversion?

a. 2.25 μs
b. 1.75 μs
c. 1.05 μs
d. 0.5 μs
e. 1.25 μs

Commented [RS13]: Resolution = 5 / 1024 = 0.004882813
(1.76 / 5) * 1024 = 360.448 (round down to 360)
360 = 0b0101101000
Afte the 7th bit (from left) is set, the voltage seen from the DAC will be (1.25 + 0.3125 + 0.15625 + 0.0390625) = 1.7578125
Difference = 1.7578125 - 1.75
= 0.00021875 < Resolution
So the timing is 7 * 0.25 = 1.75us

8. The ADC module in our AT328P is configured to use a Pre-Scale value of 128. The power to the ADC module is enabled and we are using ADC Interrupts. The AVcc pin is connected to 5V. The ADMUX register is configured as below.

```
ADMUX |= ((1 << REFS0) | (1 << ADLAR));
```



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

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V_{ref} turned off
01	AV_{CC} with external capacitor at AREF pin
10	Reserved
11	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Bit 5 – ADLAR: ADC Left Adjust Result
The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see ADCL and ADCH.

The ADC ISR is given below. The variable 'adcvalue' is declared as a global unsigned integer.

```
ISR(ADC_vect)
{
	unsigned char loval, hival;

	loval = ADCL;
	hival = ADCH;
	adcvalue = (unsigned char)(hival * 256) + (unsigned char)(loval);

	// ADSC = 1 (Start Conversion)
	ADCSRA |= (1 << ADSC);
}
```

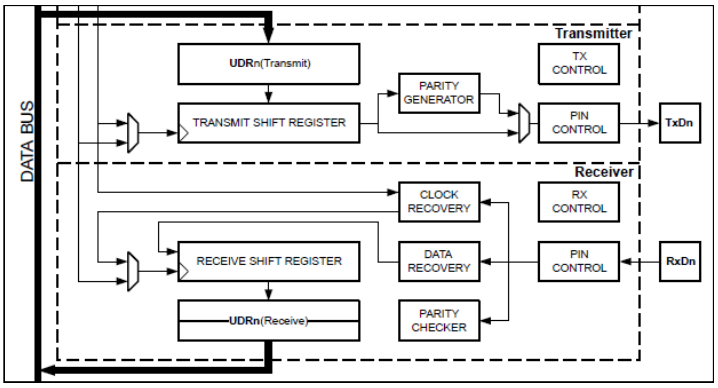
Given an analog voltage of 4.399 V at Channel 0, what would be the value stored in the variable adcvalue after the conversion is complete?

a. 900
b. 901
c. 432
d. 340
e. None of the above

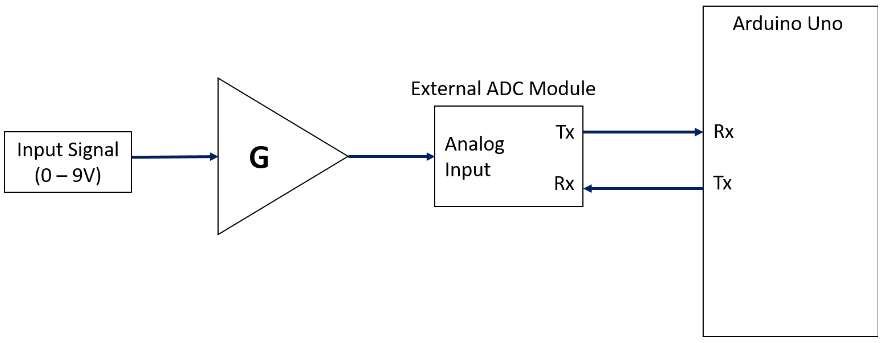
Commented [RS14]: (4.399 / 5) * 1024 = 900 -> 11100001 00
After the multiplication, the upper 8 bits are cast to (unsigned char) so the the output will be zero. The lower 2 bits in loval will also be 0.

9. You decide to use an external ADC that has an input range of 0V – 5V with 7-bit resolution. An input signal in the range of 0V – 9V is to be converted by this ADC module as shown in the figure below. After conversion, an extra bit with a value of '0' is padded to the MSB. The final data packet is then transmitted to the Arduino Uno through the UART Channel which has been configured for 8-bit data transmission. The ADC module transmits MSB first.

The USART Transmit and Receive modules are shown here:



What is the Gain (G) required for the signal (correct to 2 decimal places) to make full use of the External ADC Module's range? What data would the Arduino Uno receive in its **UDR0** register if the Input Signal has a value of 8V?



a. Gain = 1.8, UDR0 = 0b01110011

b. Gain = 0.56, UDR0 = 0b01001110

c. Gain = 1.8, UDR0 = 0b11000010

d. Gain = 0.56, UDR0 = 0b01110010

e. Gain = 0.56, UDR0 = 0b10110011

Commented [RS15]: G = 5 / 9 = 0.56
Analog Input = 8 * 0.56 -> 4.48V
(4.48 / 5) * 128 = 114.688 -> 114
114 = 0b01110010
Since Tx is MSB first, Uno which shifts in from the MSB will have the revers order in UDR registers
UDR0 = 0b01001110

10. Suppose you are working on an intelligent system that requires a reference voltage **VREF** of **3 V** for decision making, and you decided to use a potential divider to obtain this reference voltage (see figure below). You looked up the datasheet of your voltage comparator and found that its input resistance **RIN** has some tolerance and ranges between **3 MΩ** and **5 MΩ**. Suppose you are concerned about the power consumed by the potential divider circuit, and decided to try **R1 = 220 KΩ ± 1%,** and **R2 = 330 KΩ ± 1%** (i.e., 1% tolerance resistors). Which of the following correctly shows the **possible VREF range** considering all the possible tolerances given above?

a. 2.95 V to 3.05 V

b. 2.97 V to 3.03 V

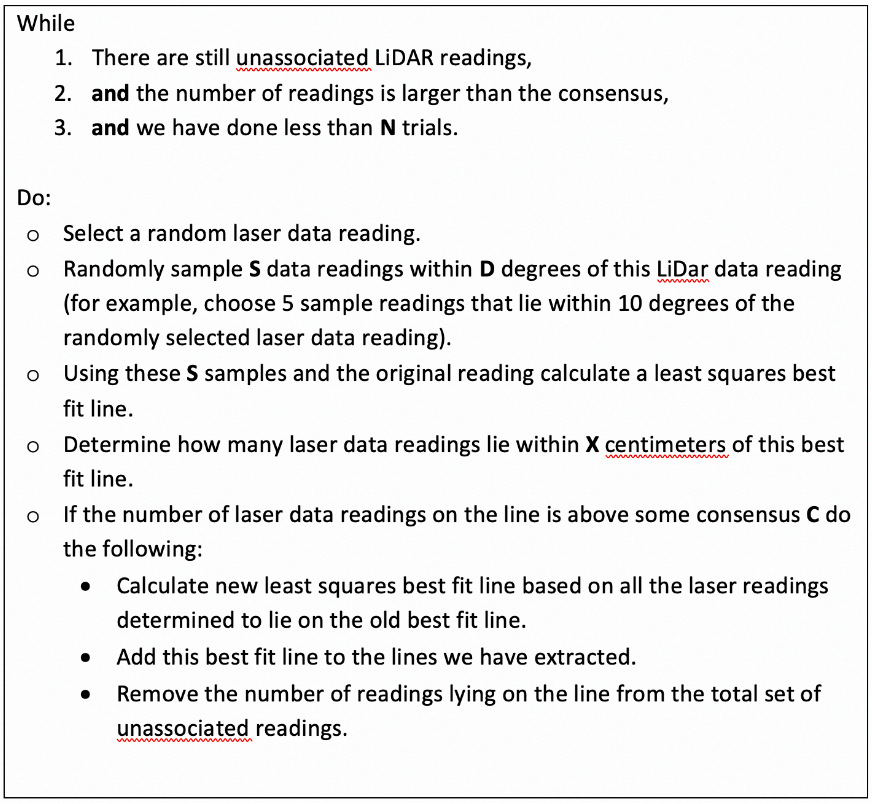
c. 2.93 V to 2.98 V

d. 2.85 V to 2.95 V

e. 2.87 V to 2.93 V

Commented [WS16]: Answer. Min VREF happens when R2 and RIN are min, while R1 is max. Max VREF happens when R2 and RIN are max, while R1 is min.

The following question make use of the RANSAC algorithm discussed in tutorial, reproduced below for your reference:



```
While
    1. There are still unassociated LiDAR readings,
    2. and the number of readings is larger than the consensus,
    3. and we have done less than N trials.

Do:
  o  Select a random laser data reading.
  o  Randomly sample S data readings within D degrees of this LiDar data reading
     (for example, choose 5 sample readings that lie within 10 degrees of the
     randomly selected laser data reading).
  o  Using these S samples and the original reading calculate a least squares best
     fit line.
  o  Determine how many laser data readings lie within X centimeters of this best
     fit line.
  o  If the number of laser data readings on the line is above some consensus C do
     the following:
       •  Calculate new least squares best fit line based on all the laser readings
          determined to lie on the old best fit line.
       •  Add this best fit line to the lines we have extracted.
       •  Remove the number of readings lying on the line from the total set of
          unassociated readings.
```

11. Suppose we know the following for a specific Lidar:

a. Each data point, (x, y) coordinate is **16 bytes in total.**
b. The Lidar scanning rate is remarkably uniform.
c. The bandwidth used to communicate **data points alone** is **46.08 kbps** (kilo-bytes per second).

The Lidar unit is placed at the center of a perfectly square and empty room **facing random direction**. What is the **smallest C consensus value below** that will cause the RANSAC fail to detect **any of the four walls?**

a. C = 530
b. C = 630
c. C = 730
d. C = 830
e. None of the above.

Commented [SYJ17]: 46080 bytes / 16 bytes = 2880 point samples
2880 / 360 degree = 8 samples per degree
8 * 90 degree (each wall) = 720 samples lie on the straight line.