EEE3095/6S Class Test 1 Solutions

18 September 2023

Duration: 90 minutes Total: 80 marks

Empl ID: ______________________

Instructions:

- This is a closed-book test.
- There are four questions to this test, each of which contains sub-questions. You must answer **all** questions as each one corresponds to a Graduate Attribute on which EEE3096S students will be assessed.
- Use a **pen** to complete the test — pencil will only be marked for drawings. Values indicated on drawings must be written in pen. Please show all of your working and reasoning clearly — your method and approach matter.
- Keep all your answers **within** the allocated block(s) for each question; anything outside of these blocks may not be marked.
- This test assesses your understanding of the "Light-of-Things" (LoT) design problem. The LoT assignment and preparatory tasks were meant to clarify the system design being considered, and this test will focus on assessing approaches, methods and solutions that you would apply to address the assigned tasks. A summary of the LoT concept is provided on the next page.

Question	Available Marks	Grade
1	20	
2	20	
3	20	
4	20	
Total:	80	

Light-of-Things System Design

This test relates the "Light-of-Things" design concept that was proposed in the GA Assignment. As mentioned, the assignment is more of a conceptual assignment to explain a design topic of focus and to familiarise students with the system. Accordingly, the same system design description as provided previously is summarised here for this test.

The assignment concerns development considerations for a LoT sensor data transfer network. The baseline version, and the version that students are mainly going to be considering, utilises simplex communication in which a transmitter node sends data to a receiver node.

Figure 1 gives an example scenario of how such a network might be set up. In this scenario, there is a Central Receiver, the CR, to which data messages are sent via light beams. The sensor nodes, labelled SN-1 to SN-3 in the diagram, sample one or more physically attached sensors and transmit this sensor data by light signals back to the CR.

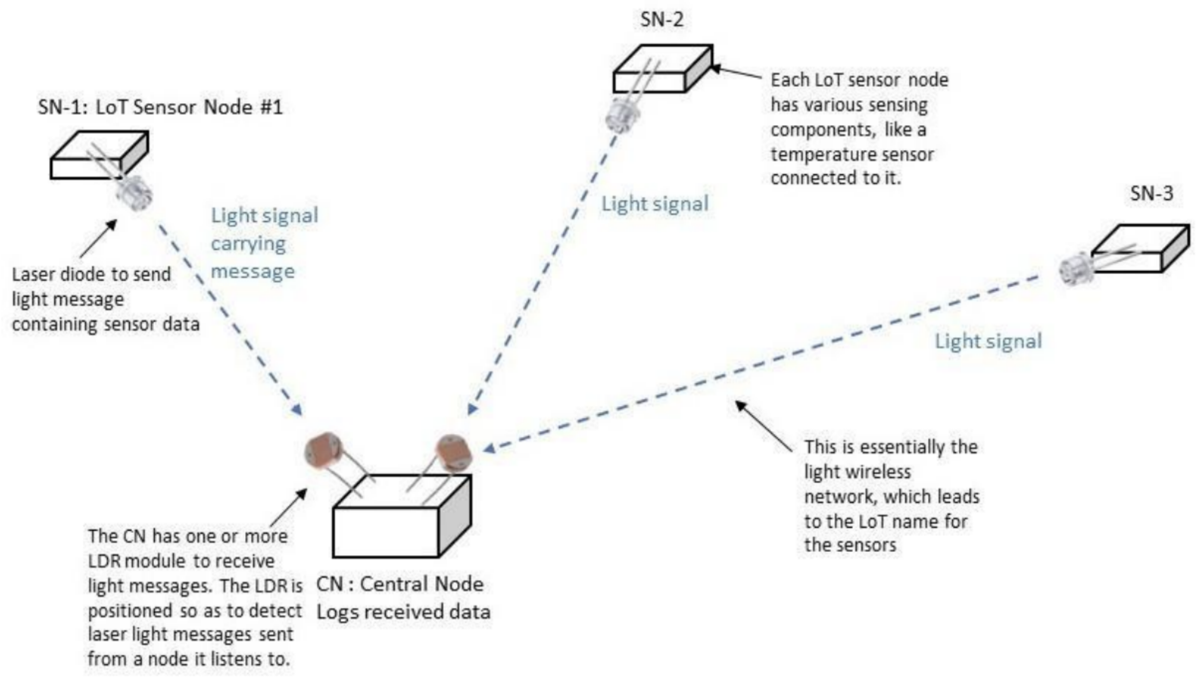
SN-1: LoT Sensor Node #1
Laser diode to send light message containing sensor data
Light signal carrying message
SN-2
Each LoT sensor node has various sensing components, like a temperature sensor connected to it.
Light signal
SN-3
Light signal
This is essentially the light wireless network, which leads to the LoT name for the sensors
The CN has one or more LDR module to receive light messages. The LDR is positioned so as to detect laser light messages sent from a node it listens to.
CN : Central Node
Logs received data

Figure 1: LoT laser light data transfer network

The data to be sent by a node is first packaged into a message package, and this package is appropriately encoded and transmitted by the sensor node's Light-Emitting Diode (LED) transmitter. In the standard configuration, each sensor node needs its LED to point directly at a Light Dependent Resistor (LDR) on the receiver side.

Based on the above (and the full GA Assignment that was handed out in the previous term), answer all of the following questions.

Question 1: Embedded Systems Design [20 marks]

GA aspect: "B.1 Embedded System design as a complex process"

a. Consider that you are the team leader for the development team constructing this LoT system, and you have been asked (by the client) to make a decision as to whether an Application Programming Interface (API) should be developed for the project.

(i) Write down what your response would be, providing motivation for whether or not an API would be advisable. (4)

Solution: An API *would* be advisable for the following reasons:

- Facilitates and clearly defines the compatibility requirements for reliable communication between different systems.
- Maximises system modularity as peripherals/applications that aid system operation can be embedded easily.
- Ensures consistent data formatting conventions in data communicated between different hosts.
- Promotes code reuse as repetitive functions need only be written once into the API module.

(ii) Provide a brief description of the functions that you recommend for this API for the LoT system. (8)

Solution: Transmitter:

- read_msg_pkg(msg_pkg) = reads packaged message/data from sensor(s)
- encode(msg) = encodes message
- transmit(enc_msg) = initiates transmission of encoded message

Receiver:

- detect() = listens for input message and receives message
- decode() = decodes message
- store() = stores message for further use

To get full marks, every reason in your motivation needs to be related to the LoT use case either by example or by explanation. Marks are also awarded if you provide any additional useful functions (with rationale) to the API.

b. The LoT receiver, on the central node, is connected to a microcontroller that controls it. But this controller needs to send received data on to another embedded platform (another STM board, for example) that will make use of the received data.

(i) If you are allowed to use either the I2C or SPI communication protocol for this, explain to the client two advantages of I2C over SPI and two advantages of SPI over I2C. (4)

Solution: Student can choose SPI or I2C. Any of the following advantages/disadvantages can be mentioned, or anything equally appropriate.

Advantages of I2C:

- Uses less hardware; 2 lines instead of 4 for SPI
- The message protocol includes acknowledge commands for verification purposes; more reliable
- Supports multiple masters
- Better for long distances

Advantages of SPI:

- No overhead from start/stop/ACK bits
- Full duplex; devices can transmit and receive at the same time
- Uses Slave Select line instead of 7-bit addressing
- Uses push-pull drivers for superior speed and signal integrity; better suited for high-speed, low-power applications

(ii) If using SPI, is it necessary to have a common clock line between the devices for the purpose of receiving data? Why? And where would this signal be generated? (4)

Solution: SPI is a *synchronous* protocol and thus requires a clock line (SCLK) that connects all the devices; all slave devices should be connected to the master via the common clock line, and this signal is generated directly by the master device. The clock thus ensures that the transmitting and receiving devices are synchronised and no start/stop bits are required.

Question 2: Design Diagrams [20 marks]

GA aspect: "B.2 Use of design diagrams"

The LoT transmitter needs to send out signals of light which need to follow a regular timing period, i.e., with each pulse lasting for a predetermined time period. For the data feed, the most basic item that the LoT transmitter needs to send is a byte (perhaps obtained from an ADC reading). But to improve reliability and sequencing, we could use a start bit (e.g., a logic-high bit, which turns on the LED for a certain period), then the 8 bits of data, then a parity bit, and then finally a stop bit (e.g., a logic-low bit, turning the LED off for a certain period).

a. Provide a labelled diagram (e.g., a flowchart or Finite State Machine) showing how you could go about taking a byte of data (input variable "X") and transform it into the final transmitted data that will be written to the LED, including the surrounding start, parity and stop bits. Assume even parity and show your logic in implementing the parity bit. (16)

Solution: Open-ended for different diagrams and types, but the mark breakdown is generally:

Overall diagram and logic [10]
Start bit [1]
Stop bit [1]
Data byte "X" [1]
Parity bit [1]
Descriptions/labels [2]

The diagram should illustrate some kind of logic to create the desired 11-bit message structure, i.e., starting off with a Start bit, then appending the Data "X", then the Parity bit, and finally the Stop bit. The diagram should also demonstrate the logic whereby the parity bit is decided for even parity, which is where most of the "logic" should be illustrated in the student's answer; something like:

- Take the 8-bit data "X" and count the number of "1" bits in it (without the parity bit)
- If the sum is odd (i.e., a modulus division by 2 yields a non-zero result) then the parity bit becomes 1
- If the sum is even (i.e., a modulus division by 2 yields a zero result) then the parity bit becomes 0

b. Briefly explain how an odd-parity system would differ from the previous implementation. (4)

Solution: With even parity, the sum of the "1" bits in the data needs to be an *even* number *after* adding the parity bit; so with odd parity, the opposite is true and the sum of "1" bits would need to be an *odd* number. Thus, using the same logic as before, the odd-parity implementation would work as follows:

- Take the 8-bit data "X" and count the number of "1" bits in it (without the parity bit)
- If the sum is odd (i.e., a modulus division by 2 yields a non-zero result) then the parity bit becomes 0
- If the sum is even (i.e., a modulus division by 2 yields a zero result) then the parity bit becomes 1

Question 3: Embedded Communications [20 marks]

GA aspect: "B.6 Use of standard embedded systems communication protocols"

a. Consider that your LoT sensor device can be connected as a master to some slave device via SPI or I2C. For this question you can choose to answer either one of the choices below depending on whether you are more comfortable with I2C or SPI. (10)

- **Choice 1 (SPI)**: Draw a labelled diagram showing the physical SPI interface (with a brief description of how it works) as well as the timing diagram/waveforms for sending the data byte 0xC2 from master to slave. Assume that SPI Mode 0 is being used and the most significant bit is transmitted first.

- **Choice 2 (I2C)**: Draw a labelled diagram showing the physical I2C interface (with a brief description of how it works) as well as the complete message structure for sending the data byte 0xC2 to a slave device (with a proposed ID/address for the slave).

Solution: Choice 1 (SPI):

Student needs to give a short explanation of how the protocol works to get full marks. Something like:

1. Master pulls slave select/chip select line Low, turns on the clock signal
2. Master transmits data to slave device over MOSI line; slave can also transmit data to master on the MISO line at the same time
3. After transmission, master pulls SS High again to stop the movement of data

Figures 2 and 3 show the physical interface and timing/signal diagram respectively. The student can show anything on the MISO line (in the latter figure) as there will be only garbage data on that line. The data on MOSI should be 0b11000010 (i.e., 0xC2). The Slave Select line can also be pulled High or left Low at the end; either way, we only need to see the correct bits being transmitted, the clock alternating after SS is pulled Low, and the data being sampled in the middle of the MOSI pulses and on the rising edge of SCLK (which should be Low when idle) due to SPI Mode 0.

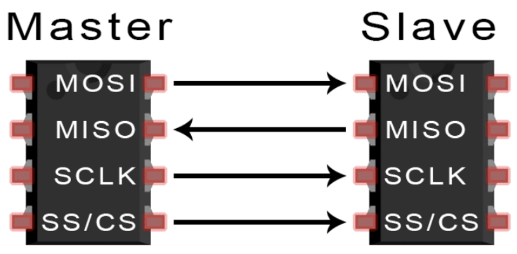

Figure 2: SPI protocol's physical interface

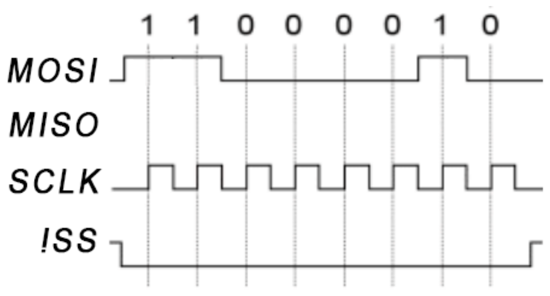

Figure 3: SPI protocol for sending 11000010 from master to slave

Solution: Choice 2 (I2C):
Student needs to give a short explanation of how the protocol works to get full marks. Something like:

1. Master transmits Start bit by pulling SDA Low while SCL is High
2. Master transmits 7-bit Slave address
3. Master transmits Write bit: SDA pulled Low for 1 bit and Receiving Slave transmits ACK bit to acknowledge (pulling SDA Low for 1 bit) or NACK bit to not acknowledge (pulling SDA High for 1 bit)
4. Master transmits 8-bits of data and Receiving Slave transmits ACK or NACK in response
5. Master transmits Stop bit to indicate end of communication, pulling SDA go High while SCL is High

Figures 4 and 5 show the physical interface and message structure respectively; student needs to show the pull-up resistors in the former figure. The student can also use any 7-bit value as the slave address in the latter figure, and this diagram could also have been drawn using pulses or a timing diagram instead of the block message structure shown below.

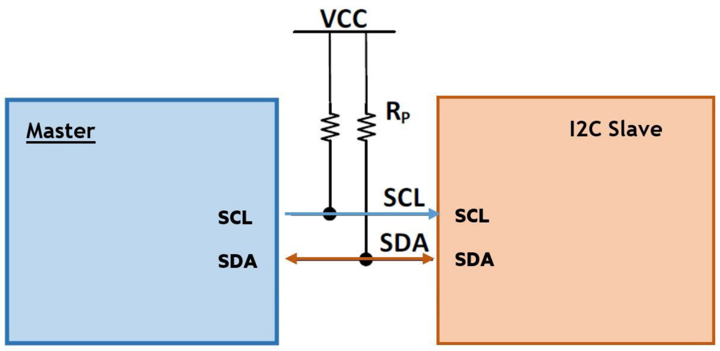

Figure 4: I2C protocol's physical interface

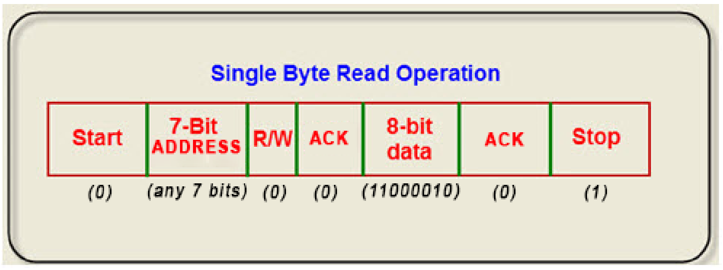

Figure 5: I2C message structure (SDA line) for sending 11000010 from master to slave

b. Consider that you are tasked to construct a communication protocol for the LoT system. You are to allow up to 16 sensor nodes to send data packets to the central node. Assume each sensor is given a 4-bit ID that ranges from 0 to 15 (base-10). Each sensor is able to send up to 20 bytes of data, but not all the sensors need this much space and some sensors produce more sampled data than others.

(i) Decide and motivate for whether or not you think parity checks and/or any other error detection schedule would be advisable. (2)

Solution: To improve communication reliability and detect some degree of error, a parity bit (or some other error checking method) could be used. A parity check system, in particular, is simple to implement and is thus recommended for this system design; ultimately it does not add much overhead but can improve reliability.

(ii) Develop and describe a potential message structure for a sensor node to be able to transmit a block of data via the LoT light link, assuming that there is no clock line and that each bit would take 1 ms to transmit. Do this by proposing a possible schedule by which the 16 sensor nodes could be set up so that they can each transmit to the central receiver/LDR on the central node without doing so simultaneously; then, using your proposed schedule, compute how long it would take for all 16 sensors to transmit 20 bytes each (including all bits surrounding the data in your message). (8)

Note: This is an open-ended question and the marking is based on the logic and clarity of your explanation; think about synchronisation between devices, start/stop conditions, and reliability.

Solution: Ultimately the student should propose some reliable means of transmitting data from the sensor nodes – where each node is given a unique ID from 0 to 15. A generic schedule can thus be developed, e.g., dictating how each node would transmit at a specific time (and for a specified period) after some synchronised $t = 0$ instance. For example, an effective packet structure would be something like this:

- Send a "start of transmission" indication, such as having the transmit LED on for a few milliseconds, e.g., 2 ms. A start bit could thus be ignored since this "start of transmission" indication would serve as the overall "start bit" for the full data transmission (up to 20 bytes).
- Send the transmitter ID as a 4-bit quantity, and this could have its own parity bit and stop bit appended.
- The subsequent 20 bytes of data could then be transmitted, and these could simply be sent as timed pulses with e.g., 1 ms of the LED being on if sending a 1, or 1 ms the LED being off when sending a 0.
- To improve communication and detect errors, a parity bit could be sent after each byte of data.

- Send the "stop transmission" after each byte of data, which could just be having the LED off for e.g., 1 ms.

Accordingly, each message would take: 2 ms "start of transmission" + (4 ms ID + 1 ms parity + 1 ms stop bit) + (8 ms data + 1 ms parity + 1 ms stop)* 20 bytes = 2 + 6 + (10 * 20) ms = 208 ms per sensor node. The maximum period of transmission would thus be 208 ms * 16 = 3328 ms for all the nodes to transmit one after the other, starting at some time $t = 0$.

For the LoT application, this would likely be sufficient in most cases – particularly if sampling conditions (such as environmental temperature or lighting conditions) do not change rapidly.

Question 4: ARM Tools [20 marks]

GA aspect: "B.5 Use (cross-)compiler for developing ES software"

Consider that you are involved in the development and debugging of the program code running on the central node receiver for the LoT system.

a. The LoT receiver is likely going to be detecting and processing signal pulses at a rather fast rate – perhaps at a few hundred kHz. Would there be any point in using a debugger for its code development, such as GDB? Motivate your answer with at least two reasons and include an example scenario where you may/may not want to use such a tool. (6)

Solution: Motivation for why using a debugger would be helpful:

- Breakpoints help when data is processed at high speeds as they allow us to step through the code in certain chosen areas step-by-step which helps us find errors in code no matter how fast the program is running and speeds up the process of finding errors in our code.
- We can change and view variables, call stack and register values at runtime – saving us time when finding and fixing errors.
- We can trace function calls and find errors in our logic or the program flow.
- Helps to reduce debugging in hardware and helps to narrow down the area of the source of the issue by eliminating or confirming whether the error is in software.

To get full marks, you also need to give an example/scenario of how you might use the debugger for the receiver code in the LoT system, e.g., for debugging the implementation of the detection/decoding/storage of data at the receiver.

b. The terms "runtime environment", "development environment" and "execution environment" do not refer to the same thing. Briefly explain the difference between these three terms in the context of embedded systems development. (9)

Solution: The runtime environment addresses various issues, especially the aspects of:

- How the program starts
- How procedures/functions are called and results returned
- Methods for passing parameters between procedures
- Management of application memory
- and more...

The development environment is the environment where development occurs, comprising the equipment and tools used for development – such as compilers and linkers – but also other resources such as relevant manuals, datasheets, etc. The development environment for embedded systems is generally not as well defined as for a PC, as the tools depend on the software and hardware vendors as well as the peripherals used. The toolchain is also often built from scratch, making the development process more tedious and less approachable compared to PC development.

The execution environment refers to those components that are used together with the application's code to make a complete computing system, such as the processors, networks, operating systems and so on. The *runtime* environment can thus be considered a large part of the *execution* environment, and an execution environment could comprise multiple runtime environments.

c. Provide a formal definition for a "cross-compiler toolchain", and give two examples of typical functions of a toolchain. (5)

Solution: A cross-compiler toolchain is a set of tools used to compile and analyse an application that runs on a platform that is not same as the platform on which the tools are executed. Common functions of a toolchain include:

- compiling
- linking
- profiling
- converting formats
- disassembling binaries
- debugging support
- providing tools to program the platform

END OF TEST