EEP3010 : COMMUNICATION SYSTEMS LAB PROJECT FINAL REPORT

Divyam Patel (B20EE082), Rishabh Jain (B20EE083)
and Harshit Mathur (B20EE085)

May 15, 2023

1 Project Title : Data Transmission Using LiFi

Abstract:
LiFi (Light Fidelity) is a wireless communication technology that uses visible light to transmit data. It is a promising alternative to traditional radio frequency (RF) communication, offering high-speed data transfer rates, increased security, and low electromagnetic interference. In this project, we will use an Arduino UNO board to implement a simple LiFi system to transmit data wirelessly over a light beam. LiFi data is transmitted in several bit streams and the receiver side consisting an LDR decodes the message. The transmission happens in the form of binary data where 0 means LED in 'OFF' state and 1 means that the LED is in the 'ON' state. Transmitter and receiver sections contain Arduino which is programmed using Arduino IDE. High power intensity led's are used in the LiFi transmitter. In receiver section LDR is used to detect the light signal generated by the LiFi transmitter.

2 Equipment Used

Hardware Components Used:

- Arduino Uno x 2
- USB Cable x 2
- LED x 1
- LDR x 1
- Resistor (220 Ω) x 2
- Breadboard x 1

- Connecting Wires

Software Components Used:

- Arduino IDE
- Visual Studio Code

3 Methodology

In order to incorporate LiFi technology in the current project, we opted to employ an LED as the transmitting device and an LDR as the receiving device. The mode of operation involves the transmission of digital signals by modulating the LED to turn on and off at a particular frequency. Subsequently, the receiver perceives these signals by gauging the variations in light intensity through the use of the LDR. The binary-coded data is transmitted using the LED, and the decoding process takes place at the receiver end by monitoring the changes in the light intensity at predetermined time intervals.

3.1 Circuit Diagram

The following circuit diagram shows the connections between the transmitter and receiver. The circuit was set up using an Arduino Uno board. The transmitter code was uploaded to the Arduino Uno board, while the receiver code was uploaded to the second Arduino Uno board.

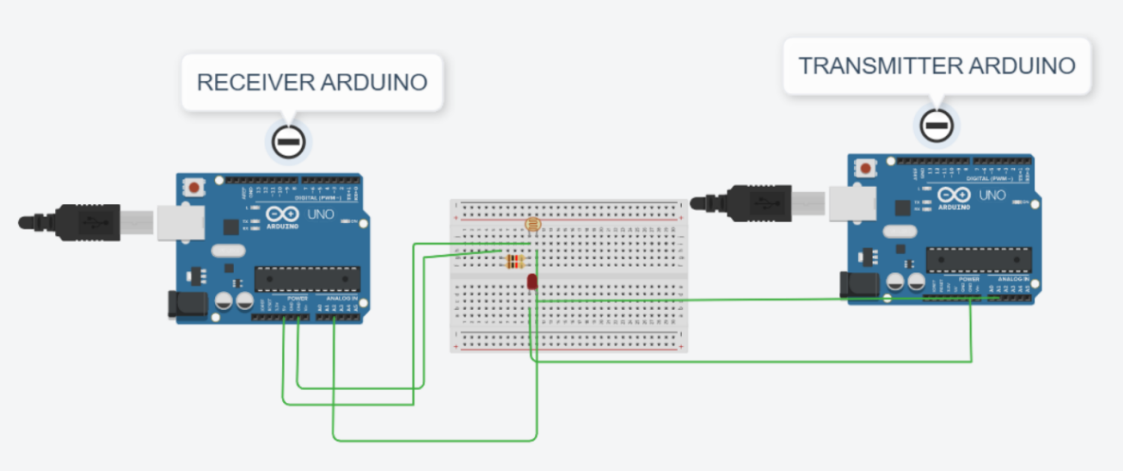

Figure 1: Circuit Diagram using Tinkercad

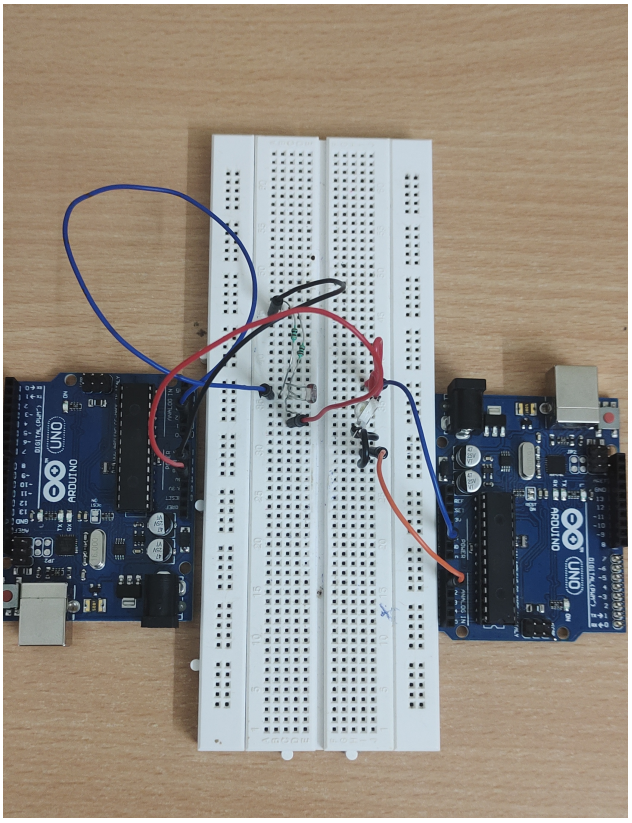

Figure 2: Hardware Circuit Diagram

3.2 Procedure:

1. Connect an LDR (Light Dependent Resistor) to pin A2 of the Arduino Uno and connect a LED to pin A1 of the Arduino Uno.

2. Connect the transmitter and receiver to separate computers, and open the Arduino IDE on both computers.

3. Write the transmitter and receiver code into the IDE on the transmitter and receiver computer and upload it to the Arduino Uno.

4. Open the serial monitor on the receiver computer and set the baud rate to 9600.

5. Ensure that the LDR is positioned in a way that it can receive the light emitted by the LED.

3.3 Codes:

3.3.1 Transmitter Code:

```
const int LED_PIN = A1;
const int PERIOD = 30;
const char* message = "This was text transmitted using LiFi ";

int message_length;

void setup()
{
  pinMode(LED_PIN, OUTPUT);

  message_length = strlen(message);
}

void loop()
{
  for (int i = 0; i < message_length; i++)
  {
    send_byte(message[i]);
  }

  delay(1500);
}

void send_byte(char byte_to_send)
{
  digitalWrite(LED_PIN, LOW);
  delay(PERIOD);

  for (int i = 0; i < 8; i++)
  {
    digitalWrite(LED_PIN, (byte_to_send & (0x01 << i)) != 0);
    delay(PERIOD);
  }

  digitalWrite(LED_PIN, HIGH);
  delay(PERIOD);
}
```

The character string comprises of 8 bytes, and to send a single character, a loop is executed 8 times. During each iteration, a bitwise AND operation is performed using bit 1 shifted to

the i^{th} position. The resultant value obtained is used to determine the value of the bit at the i^{th} position. If it is 0, we write LOW on the LED pin, and if it is 1, we write HIGH on the LED pin. To maintain the sequence of transmission, each bit is transmitted with a delay of 1 period. To ensure consistency, we add a delay period both before and after transmitting 1 character. This delay aids in ensuring the effective transmission of data and facilitates seamless communication between the transmitting and receiving devices.

3.3.2 Preprocessing of image before transmission:

```python
from PIL import Image

# Open the image file
image = Image.open("image.png")

# Convert the image to grayscale
gray_image = image.convert('L')

# Resize the image to reduce the size
resized_image = gray_image.resize((32, 32))

# Get the pixels of the resized grayscale image
pixels = resized_image.load()

# Create a string to hold the binary data
binary_data = ""

# Iterate through the pixels and append 0 or 1 depending on the pixel value
for i in range(resized_image.size[0]):
    for j in range(resized_image.size[1]):
        if pixels[i, j] < 128:
            binary_data += "0"
        else:
            binary_data += "1"

# Save the binary data to a text file
with open('image_binary_new.txt', 'w') as file:
    file.write(binary_data)
```

To transmit an image, it is first necessary to convert it into binary format. To achieve this, we utilize Python to perform the necessary image processing. Firstly, the image is converted to gray-scale, and then it is resized to 32x32 resolution to meet the memory limitations of the Arduino. Following this, the pixel values are encoded, whereby a value greater than 128 is encoded as '1' and a value less than or equal to 128 is encoded as '0'. The resulting binary pattern is then flattened to enable it to be transmitted serially. The transmission process for images follows the same steps as that of text.

3.3.3 Receiver Code:

```
const int LED_PIN = A3; const int LDR_PIN = A2;
const int THRESHOLD = 60; const int PERIOD = 50;

bool get_ldr(); char get_byte();

bool previous_state; bool current_state;

void setup()
{
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
}

void loop()
{
  current_state = get_ldr();
  if (!current_state && previous_state)
  {
    char received_byte = get_byte();
    Serial.write(received_byte);
  }
  previous_state = current_state;
}

bool get_ldr()
{
  int voltage = analogRead(LDR_PIN);
  return voltage > THRESHOLD ? true : false;
}

char get_byte()
{
  char ret = 0;
  delay(PERIOD * 1.5);
  for (int i = 0; i < 8; i++)
  {
    ret = ret | get_ldr() << i;
    delay(PERIOD);
  }
  return ret;
}
```

When setting up the receiver, it is important to note that the LDR is connected in a pull-down configuration. This means that when light falls on the LDR, the resistance of the LDR decreases, causing an increase in the value at the pin to which it is connected. Therefore, it is the change in resistance of the LDR due to the presence or absence of light that ultimately affects the voltage at the pin.

In the code for the receiver, several elements are defined including the LED pin, LDR pin,

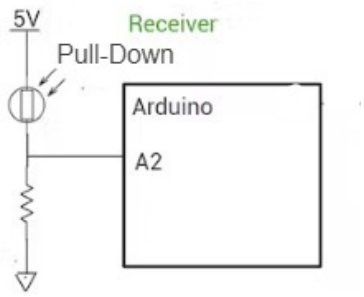

Figure 3: Receiver Circuit

threshold, and period. During the setup function, the LED pin is initialized as an output and the baud rate of serial communication is set to *9600*. Meanwhile, the loop function operates by constantly checking the state of the LDR with the *get_ldr* function. If the LDR moves from a high state to a low state, which signifies that the light beam has been interrupted, the *get_byte* function is called upon to read the transmitted byte and then send it over the serial port. The *get_ldr* function itself operates by reading the analog value of the LDR pin and returning a value of true if the value is above the threshold, and false if it is not. Lastly, the *get_byte* function works by delaying briefly to ensure that the light beam has been interrupted, before reading each bit of the transmitted byte using bit-shifting and bitwise OR operations in order to recover the data, ultimately returning the received byte.

Once the binary sequence is received for an image, a process begins where each binary value is transformed into a corresponding pixel value. If the binary value is 1, the pixel value is set to 255, while if it is 0, the pixel value is set to 0. This conversion step allows for the representation of the image in a format that can be easily understood by a computer. Subsequently, the pixel values are utilized to reconstruct the original image. This process involves mapping the pixel values to their corresponding positions in the image and rendering each pixel with its associated value. Ultimately, the image is recovered and can be viewed or processed as necessary.

The following code is used to reconstruct the image from the binary data received by the receiver code of Arduino:

```python
from PIL import Image

# Open the binary data text file and read the contents
with open('image_binary_new.txt', 'r') as file:
    binary_data = file.read()

# Calculate the image dimensions based on the binary data length
image_size = int(len(binary_data) ** 0.5)

# Create a new PIL Image object with the calculated dimensions
image = Image.new('L', (image_size, image_size))

# Load the pixel data of the image
pixels = image.load()

# Iterate through the binary data and set the pixel value based on the ...
    binary value
for i in range(image_size):
    for j in range(image_size):
        pixel_value = 255 if binary_data[i * image_size + j] == "1" ...
            else 0
        pixels[i, j] = pixel_value

# Save the reconstructed image
image.save('reconstructed_image_new.png')
```

4 Results & Discussion

Firstly, we sent a sample text message *"This text was transmitted using LiFi"* using the transmitter code and received the message successfully on the receiver end. The received text message was displayed on the serial monitor of the Arduino IDE, as shown in the figure below:

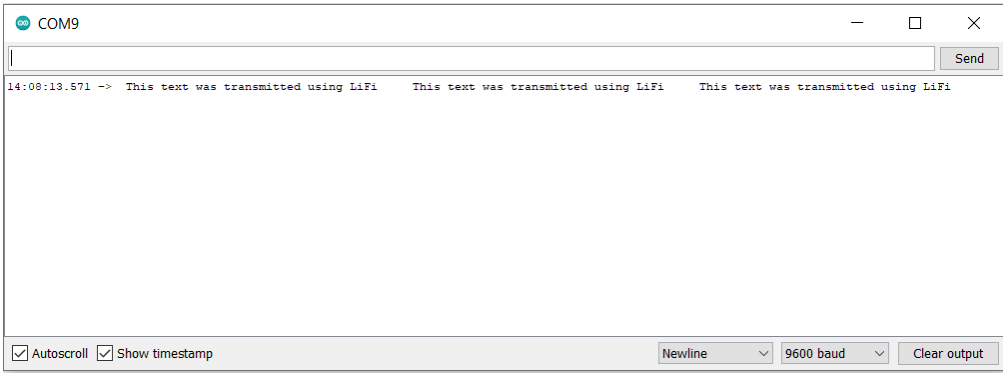

Figure 4: Serial Monitor Output of Received Text Data

After successfully transmitting the text message, we proceeded to send an image using

the same LiFi system. The image was first resized to a 32x32 pixel format and converted to binary data. The binary data was then transmitted using the transmitter code, and the receiver code converted the received data back to an image.

The transmitted image and the received image are shown below:

Figure 5: Transmitted Image (left) and Received Image (right)

As can be seen, the received image is slightly distorted due to the limitations of the LiFi system. In summary, we were able to successfully transmit both text data and images using a LiFi system consisting of an LED and an LDR.

5 Challenges Associated with the Project

During the implementation of this project, several challenges were encountered:

- **Arduino Uno Size:** The limited processing power and memory capacity of the Arduino Uno made it challenging to transmit large amounts of data. In particular, the size of the image data had to be reduced to 32x32 pixels in order to fit within the memory constraints of the Arduino Uno.
- **Transmission Speed:** The use of a single LED as the transmitter resulted in relatively slow data transmission rates. One potential solution to this problem would be to use multiple LEDs to increase the data transmission speed.
- **Alignment of the Transmitter and Receiver:** The alignment of the transmitter and receiver was critical for successful data transmission. Even slight misalignments could result in loss of data.
- **Limitations of Arduino Uno:** In order to transfer images directly using libraries like SPIFFS file system, more memory capacity is required than the Arduino Uno can

provide. One potential solution to this problem would be to use a board with more memory capacity, such as the FPGA Board.

Despite these challenges, we were able to successfully transmit both text data and images using a LiFi system consisting of an LED and an LDR.

6 Conclusion

In this project, we have demonstrated how to implement a simple LiFi system using an Arduino Uno. The transmitter sends a binary signal over a light beam, and the receiver detects the signal using an LDR and decodes it to receive the transmitted data. This project can be extended by increasing the transmission speed, adding error correction codes, and implementing a more sophisticated modulation scheme.

7 Work Distribution

- Divyam Patel [B20EE082] -
  He wrote the code for converting image to text file and received data bak to image, transmitter and receiver code. Report work was done by him.
- Rishabh Jain [B20EE083] -
  He worked extensively on researching the theoretical aspects of LiFi and planning the project's goals and requirements. Hardware Circuit Designing was done by him.
- Harshit Mathur [B20EE085] -
  Worked on writing the code for Transmitter and Receiver Part on Arduino IDE. He also helped in writing the final report of the project.