EE105: Introduction to Electrical Engineering

Module 5: Optical Communication with LiFi

Objective:

In this lab, we'll learn about LiFi Module and how to transfer data wirelessly over this module.

- **Li-Fi (Light Fidelity) Technology:** Li-Fi technology is a form of free-space communication that transmits data using visible light; typically through LEDs as transmitters and photodetectors as receivers.
- **Free Space Communication:** This means the communication happens without any physical medium like cables or waveguides; instead, it relies on light traveling through the air.

Apparatus

Arduino Nano 33 BLE

- Acts as the microcontroller to process the transmitting and received data from the LiFi system.

LED (Light Emitting Diode):

- Functions as the transmitter (T_x module).
- It emits light that is modulated (turned on and off rapidly) with the Arduino to encode the data.
- The modulation happens so quickly that the human eye cannot detect the blinking

PD (Photodiode):

- Functions as the receiver (R_x module).
- The Photodiode (PD) is used to detect changes in light intensity caused by the transmitting LED.
- The PD converts these light changes back into electrical signals, which are then received as data by the Arduino.

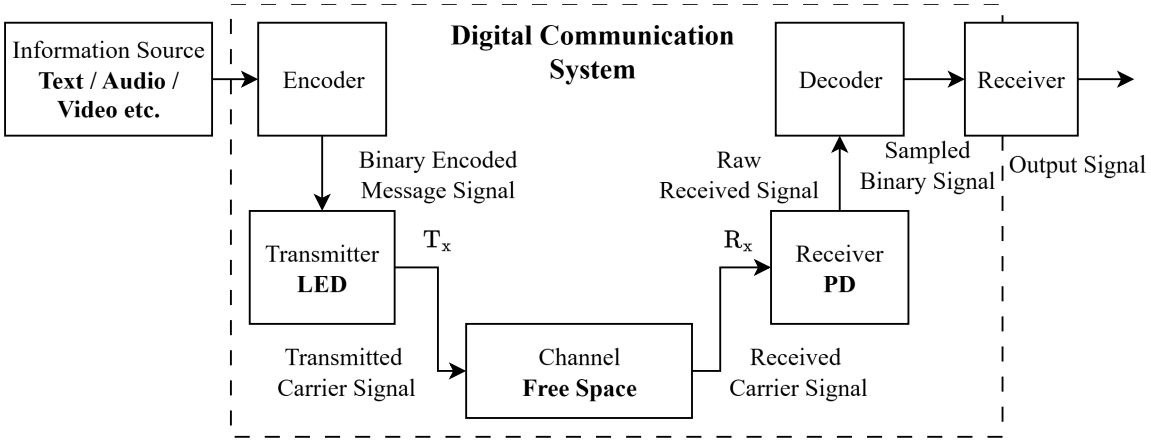


Figure 1: Block diagram of the free-space optical digital communication system used in this lab. An information source (text, audio, video, etc.) is converted by the encoder into a binary message signal, which drives the LED transmitter (T_x). The modulated light propagates through free space to the photodiode receiver (R_x), whose raw analog output is sampled and quantized by the decoder to recover the binary signal and reconstruct the original output data.

Board Setup

Set up **both** boards as below before you start Task 1. The same setup is used for Task 2.

- **Selector:** On both boards turn **S10** to **Module 3**. This also powers LED pair 2 (printed **TX**) and its photodiode (printed **RX**).
- **Jumper:** On both boards put the cap of **J5** on **A-P2** (pins 2 and 3), so that pin A6 reads the RX photodiode.
- **Placement:** TX and RX sit at the right edge of each board and face out over the edge. Lay the two boards flat with their right edges facing each other, about 3 cm apart. One board is turned half a turn. Slide one board along the gap until the sender's TX points straight at the receiver's RX.
- **Light:** Shade the receiver from sunlight and bright lamps. Their light also reaches the photodiode.
- **USB:** Plug each board into a laptop with a USB data cable. Work in pairs: one laptop sends and one receives.
- **Afterwards:** Put J5 back on **P1-A**, the position that the pulse-oximeter lab uses.



Figure 2: Board setup. Turn the selector S10 to Module 3 and put the cap of jumper J5 on A-P2 on both boards. TX and RX are the LED and the photodiode of pair 2 at the right edge of the board.

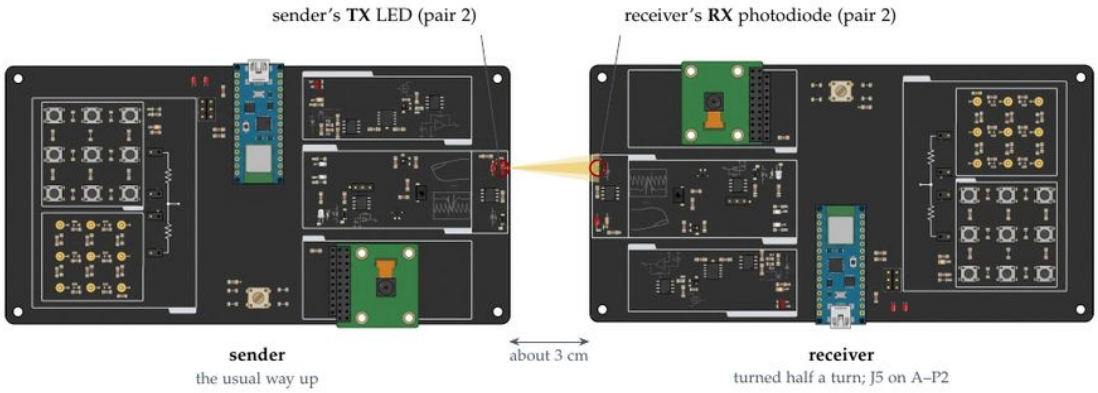


Figure 3: Placement of the two boards. The right edges face each other about 3 cm apart, and the receiver is turned half a turn so that the sender's TX LED points at the receiver's RX photodiode.

Task 1

1. Form pairs and position your boards next to each other so that the photodiode (PD) and LED face each other.
2. Run the `devboard_sketch` code from the `Task 1` folder. Open **Serial Monitor** from `Tools > Serial Monitor` or by pressing the  icon.
    a. Sender: Type `lifi_tx` to transmit data
    b. Receiver: Type `lifi_rx` to receive data.

    Adjust the value of `THRESHOLD` if needed. Use the `show_pd` function at the receiver if needed.
3. If you align the two boards correctly, you should see the message: ***"This is a test transmission."***

Transmission Process (`lifi_tx` and `send_byte`)

- The `lifi_tx` function transmits data (e.g., a text string) by modulating the light emitted from an LED.
- The input string (e.g., *"This is a test transmission!"*) is split into individual characters.
- Each character is converted into its 8-bit binary representation.
- The `send_byte` function modulates the LED's brightness to represent each bit:
    - **HIGH (LED ON):** represents 1.
    - **LOW (LED OFF):** represents 0.
- A short delay is inserted after each byte so the receiver can distinguish between consecutive bytes.
- Each byte is transmitted bit by bit, with the LED switching ON or OFF according to the bit value.

Reception Process (`lifi_rx`, `get_byte`, `read_pd`)

- The `lifi_rx` function receives the modulated light signal via the photodiode (PD) and decodes the transmitted data.
- The photodiode detects changes in light intensity produced by the transmitting LED.
- The `read_pd` function reads the PD value and classifies it as a HIGH (1) or LOW (0) bit based on a threshold (THRESHOLD).
- Each reconstructed byte is converted back into its character form and displayed using the `print_byte` function.

Example — LiFi in Action: Sending "Hi"

The string *"Hi"* is encoded as:

- 'H' → 72 → 01001000
- 'i' → 105 → 01101001

At the receiver:

- The photodiode detects the corresponding light changes.
- The Arduino decodes the binary stream back into the characters 'H' and 'i'.

Conversion Process:

- Text characters are converted into numeric ASCII codes.
- ASCII codes are transmitted as binary sequences.

Theory:

Transmission:

- Data from the information source (Figure 1) is converted to a binary bit sequence by the encoder block.
- This bit sequence is sent from the Arduino to the LED as the **message signal**, which modulates the emitted light (**the carrier signal**). The carrier's amplitude, phase or frequency is controlled by the message signal via modulation.
- Modulation is performed using two schemes in this lab:
    - ***1-bit Amplitude Shift Keying (ASK)*** – the message is transmitted as 1-bit symbols, modulating the carrier between **two intensity levels**.
    - ***2-bit ASK*** – the message is transmitted as 2-bit symbols, modulating the carrier between **four intensity levels**.

Receiver:

- The photodiode (PD) detects variations in the received light intensity and sends the raw analog signal to the Arduino.
- The decoder block "**samples"** and "**quantizes"** the analog signal:
    - 2 quantization levels for 1-bit ASK.
    - 4 quantization levels for 2-bit ASK.
- The quantized signal is converted back into a binary bit sequence and then into the received output data.

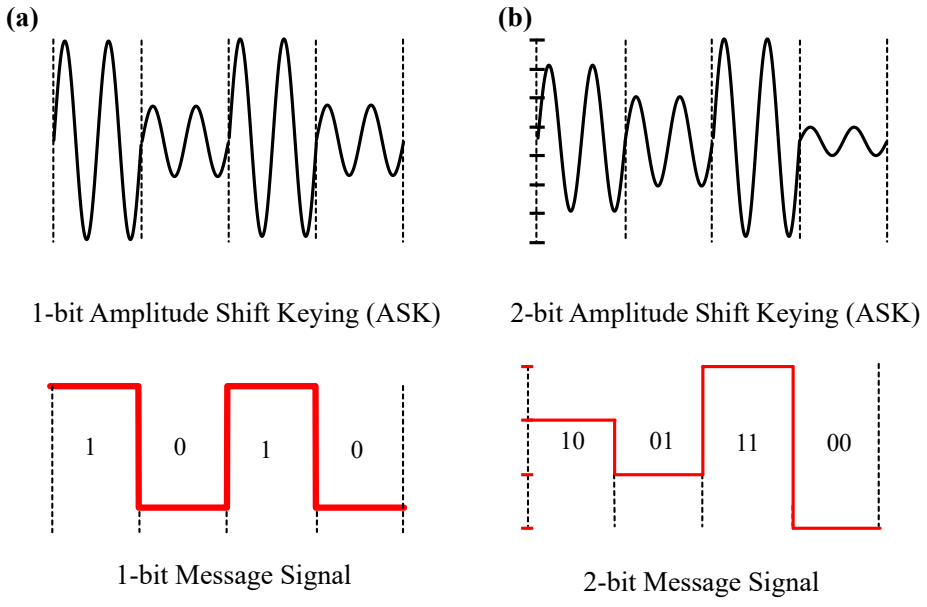


Figure 4: Amplitude Shift Keying (ASK) modulation, showing the message signal (red) modulating the amplitude of the carrier signal (black). (a) 1-bit ASK encodes one bit per symbol using two amplitude levels. (b) 2-bit ASK encodes two bits per symbol using four amplitude levels, doubling data throughput at the cost of reduced noise margin between levels.

Some Common Digital Communication Metrics

- **Bit Error Rate (BER)**

  The **bit error rate (BER)** is the ratio of the number of bits received in error to the total number of bits transmitted over a communication channel:

$$\text{BER} = \frac{\text{Number of erroneous bits}}{\text{Total number of transmitted bits}}$$

  It is a dimensionless figure of merit (often expressed as a probability or in scientific notation, e.g., 10^{-6}) that quantifies the reliability of a digital communication link. A lower BER indicates better signal integrity, and it is influenced by factors such as noise, interference, attenuation, and timing jitter in the channel.

- **Eye Diagrams**

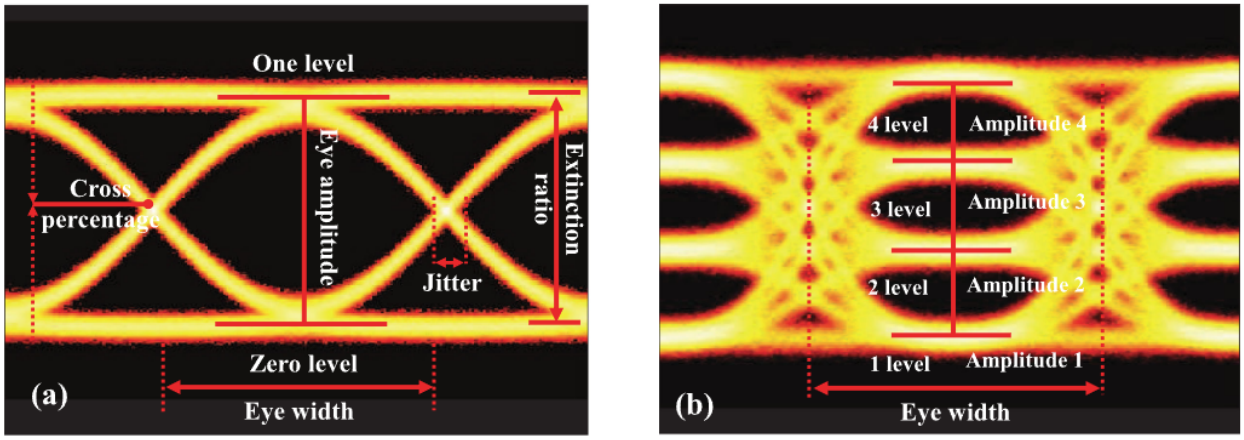


Figure 5: Eye diagrams illustrating signal quality in amplitude-shift-keyed optical communication. (a) A 1-bit (two-level) ASK eye diagram, annotated with key performance metrics: the *one* and *zero* levels, *eye amplitude*, *eye width*, *extinction ratio*, *cross percentage*, and *jitter*. A wide, open eye indicates a high signal-to-noise ratio and low BER. (b) A 2-bit (four-level) ASK eye diagram showing the four distinct amplitude levels required to encode two bits per symbol. The reduced vertical spacing between adjacent levels lowers the noise margin, making higher-order ASK more susceptible to bit errors than its 1-bit counterpart.

Task-2:

In the uploaded **Task2_v2** folder, you will find a file named **message_small_v2.txt**. Your task is to transmit this message from one development board to another.

Folder Structure:

- `/devboard_sketch_receiver`: contains the Arduino code for the receiver. The Receiver notebook uploads it to the board for you.
- `/devboard_sketch_sender`: contains the Arduino code for the sender. The Sender notebook uploads it to the board for you.
- `Sender.ipynb`: Python notebook for communicating with the sender Arduino
- `Receiver.ipynb`: Python notebook for communicating with the receiver Arduino
- `message_small_v2.txt`: Message file for transmitting over Arduino

Steps:

1. Open `Receiver.ipynb` on the receiver's laptop and `Sender.ipynb` on the sender's laptop. **You do not need the Arduino IDE for Task 2.** Close the Serial Monitor of Task 1 first: an open Serial Monitor blocks the port.
2. In each notebook run cell `1` and cell `2`. Cell `1` lists the serial ports. Type the port of your board in cell `2`, then run the **Upload** cell below it. It compiles the sketch and uploads it to your board. Wait for `[ok] Uploaded.`
   - Note the value of `interval` in each sketch (receiver: 2 ms, sender: 12 ms). The receiver interval should be a factor of the sender interval for proper sampling of the received carrier signal.
   - To change `interval`, edit the `.ino` file in the sketch folder, save it, and run the **Upload** cell again.
3. On the `Sender.ipynb` notebook, run all the cells **UNTIL** the **"transmit"** cell — `"3a. Transmit (1-bit)"`. Do not run the transmit cell yet.
4. **The receiver goes first.** Run the `"3. Capture raw ADC data from serial port"` cell in `Receiver.ipynb`. **This cell must be running before the sender transmits.** As soon as it is running, run the `"3a. Transmit (1-bit)"` cell in `Sender.ipynb` (within about 2 seconds). If the sender starts first, the beginning of the message is lost.
5. Run cell `4` in the receiver notebook to plot the raw signal. You should see the idle level before and after the message and **two** signal levels, as in Figure 6. Adjust these values in the receiver notebook if needed:
   - `CAPTURE_SECONDS` (cell `3`): how long the receiver records, 10 s by default. Raise it if the message is cut off at the end or if the sender needs more time to start. Then capture and send again.
   - `threshold` (cell `4`): only zooms the plot to the message. Set it just above the idle level.
   - `lvl_0` and `lvl_1` (cell `1a`): the thresholds that turn the ADC values into bits. Set `lvl_0` between the idle level and the lower signal level. Set `lvl_1` halfway between the two signal levels.

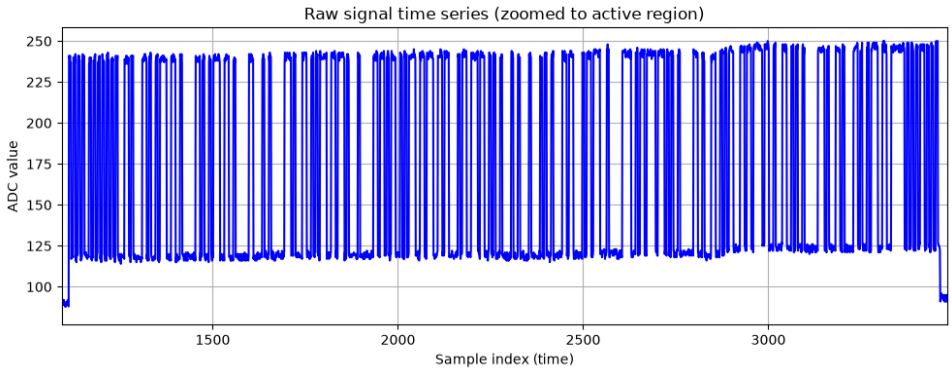


Figure 6: Raw signal at the receiver for the 1-bit scheme (cell `4`). The idle level is near 90 counts and the two signal levels are near 120 and 240 counts. Here `lvl_0` = 105 and `lvl_1` = 180 would work. Your levels depend on the distance and the room light.

6. Note the value of `T_symbol_ms` from the sender notebook. Use this value as the value of `T_send` in cell `2a` in the receiver notebook.

7. Run the blocks from `1a` to `6a` in the receiver notebook. Check the decoded text, the eye diagram and the BER.

8. Repeat for the 2-bit scheme (Part B). Run the `"3b. Capture raw ADC data"` cell in `Receiver.ipynb` **first**, then the `"3b. Transmit (2-bit)"` cell in `Sender.ipynb`. The raw signal now has **four** signal levels, as in Figure 7. Set the four thresholds `lvl_00`, `lvl_01`, `lvl_10` and `lvl_11` in cell `1b`: `lvl_00` between the idle level and the lowest signal level, and each of the others halfway between two neighboring signal levels. Set `T_send` in cell `2b`, then run the blocks from `1b` to `6b`.

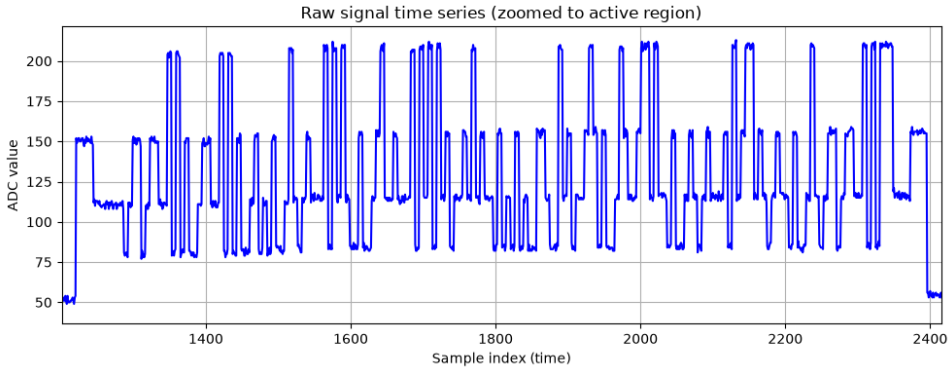


Figure 7: Raw signal at the receiver for the 2-bit scheme (four levels). The idle level is near 50 counts and the four signal levels are near 80, 115, 155 and 210 counts. Here `lvl_00` = 65, `lvl_01` = 98, `lvl_10` = 135 and `lvl_11` = 183 would work.

9. Copy the text that the receiver's Serial Monitor printed in **Task 1** and paste it into `task1_output` in the **Task 1 output** cell at the end of `Receiver.ipynb`. Run that cell.

10. Run the **Save** cell at the end of **both** notebooks. The receiver notebook writes `lab5.1_receiver_submission.json` and the sender notebook writes `lab5.1_sender_submission.json`, in the `Task2_v2` folder. Upload these files on the course portal (`ee105.org/portal`) under **Lab 5.1**. They are graded there.

Goal:

In the task, you will observe the two metrics (**BER** and **Eye Diagrams**) discussed above and identify how they are influenced by the following

1. Sampling interval of the receiver
2. Data modulation scheme.