Faculty of Engineering & Technology Electrical & Computer Engineering Department

Real-Time system and interfacing Techniques laboratory - ENCS5140

Arduino Modules Experiments

(EXP#1, EXP#2, EXP#3)

Student Name: Raghad Afghani

Student Number: 1192423

Group Members:

Student Name: Hala Ziq

Student Number :1191637

Student Name: Zakiya Abu Murra

Student Number: 1191636

Instructor: Dr.Hanna Bullata & Ahed Mafarjeh

Section: #1 **Date: 26/3/2024**

Abstract

Arduino is a popular and affordable microcontroller due to its open-source nature. This report presents three experiments that explore the versatile applications of Arduino microprocessors. The first experiment demonstrates how Arduino can convert analog input from a photocell into digital outputs, control LED brightness, and introduce the concepts of interrupts, hardware-triggered LED blinking, and software-triggered interrupts via timer1 overflow. The second experiment focuses on serial communication and provides an introduction to one of the most commonly used Arduino communication protocols, the UART (universal asynchronous receiver/transmitter). The third experiment showcases practical applications of serial communication and the I2C protocol, covering address scanning, LCD display functions, and seamless device connectivity.

Table Of Contact

Table of Contents

Table Of Figures

1.0 Theory

1.1. Arduino Uno

The Arduino Uno is a microcontroller board that operates on an open-source platform. It is built based on the ATmega328P processor and features several components, including 14 digital I/O pins, 6 analog inputs, a USB connection, a power jack, an ICSP header, and a reset button. These modules provide essential support for the microcontroller's functionality. The Arduino Uno's simple design makes it easy to use, and in the unlikely event of a worst-case scenario, it can be replaced with a cost-effective option compared to other boards such as Raspberry Pi or STM. [1]

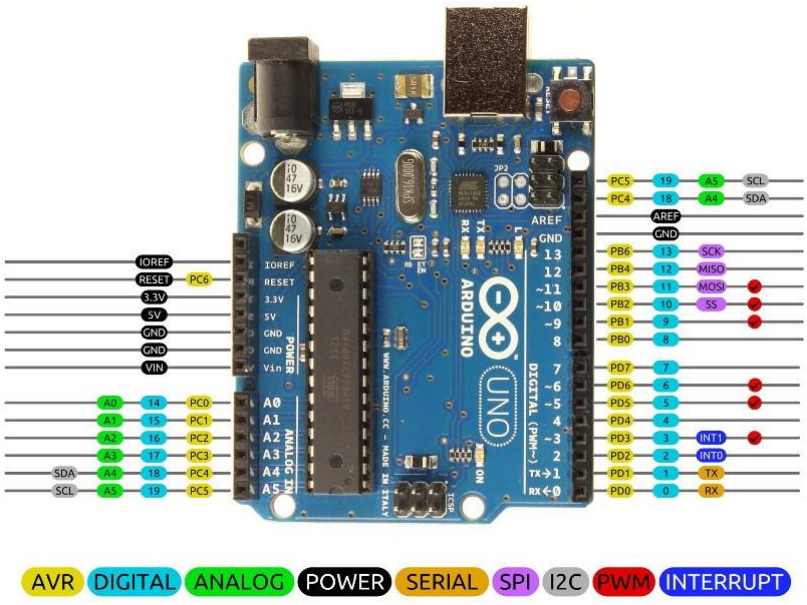

Figure 1 Arduino Uno Map

The hardware structure of Arduino Uno:

- **Microcontroller:** The Arduino Uno is equipped with a microcontroller that acts as its central processing unit, providing the computational power for the board's operations.

- **Digital Pins:** There are 14 digital pins on the Arduino Uno, which can be used to connect components like LEDs, LCDs, and other digital devices.
- **Analog Pins:** The Uno features 6 analog pins, typically used for connecting sensors that generate analog values. These pins serve as inputs for various sensors and analog components.
- **Power Supply:** The Arduino Uno has several power supply pins, including IOREF, GND, 3.3V, 5V, and Vin. These pins are used to connect sensors and other components that operate with analog values.
- **Power Jack:** The Uno board can be powered using either an external power supply or by connecting it to a computer via a USB cable.
- **USB Port:** The USB port on the board allows for programming and uploading programs to the Arduino Uno using the Arduino IDE and a USB cable.
- **Reset Button:** The Uno features a reset button that allows for restarting the uploaded program when necessary.

1.2. Photocells

A photocell is a small, inexpensive, low-power, and easy-to-use sensor that detects light. They are also known as CdS cells, light-dependent resistors (LDRs), or photoresistors. They change their resistance value in response to light exposure. Light measurement involves monitoring resistance changes. In the absence of light, the sensor has high resistance, up to 10MΩ. As light levels increase, resistance decreases. To measure a photocell, a passive setup is used, with one end connected to a power source and the other to a pull-down resistor connected to ground. An analog input of a microcontroller is used to interpret the varying resistance values.[2]

1.3. Interrupts

Interrupts are signals from hardware or software that alert the processor to a high-priority condition, causing it to suspend activities and execute an interrupt handler. [3]

1.3.1 Software Interrupts

A software interrupt is triggered by an unusual event within the processor or the execution of a specific command designed to initiate an interrupt. Attention is given to the interrupt generated by the overflow of timer1. Within the ATmega168/328 chip, timer1 stands out as one of three special hardware timers, each capable of being tailored for a variety of tasks through the adjustment of specific settings. The configuration of these timers, including the adjustment of their operation mode and speed, is facilitated by special registers. The determination of timer1's speed is influenced by the presale setting, which can be adjusted to values such as 1, 8, 64, 256, or 1024, thereby enabling control over the timer's counting speed.

1.3.2 Hardware Interrupts

Hardware interrupts are signals that prompt an Arduino's processor to pause its current task to execute a specified function, critical for reacting to external events on designated pins. For example, the Arduino Uno supports interrupts on only pins 2 and 3, although other models may offer more. Setting up hardware interrupts involves creating an Interrupt Service Routine (ISR) and linking it with the attach Interrupt function, which requires specifying the interrupt vector, ISR, and trigger condition (RISING, FALLING, LOW, CHANGE). This mechanism is essential for projects requiring immediate and responsive actions to changes in the external environment.

1.4. Serial Communication

A method utilized for data exchange between computers and other devices, ensuring security and reliability through strict protocols. [4] It encompasses two main types: asynchronous and synchronous. Asynchronous communication operates without an external clock signal, relying on parameters like baud rate, framing, synchronization, and error control for stable data transfer, where synchronous communication, on the other hand, involves both master and slave devices transmitting data and clock signals on a single bus, enabling faster transmission rates without the need for start, stop, or parity bits. Popular protocols within these types include RS-232, RS-422, RS-485, UART, USART (asynchronous), and SPI, I2C (synchronous). [5]

1.4.1 UART (Universal Asynchronous Receiver/Transmitter)

UART, short for Universal Asynchronous Receiver/Transmitter, is a straightforward protocol used for serial communication between two devices. It's simple because it only requires two wires for transmitting and receiving data, plus a ground connection. Data transfer over UART can be one-way (simplex), alternating (half-duplex), or two-way at the same time (full-duplex). The information is sent in frames, which are structured bits of data that follow a specific format. [6]

Components of UART Communication:

- **Start Bit:** In UART communication, when there's no data being sent, the line stays at a high voltage. To begin sending data, the sender drops the line from high to low for one clock cycle. This signals the receiver to start reading the incoming data at the set baud rate.

- **Data Frame:** This part carries the actual information, ranging from 5 to 8 bits in size if a parity bit is included, or up to 9 bits without a parity bit. Typically, data is sent starting with the least significant bit.

- **Parity:** The parity bit helps check if data was altered during transmission, which could happen due to interference, mismatched baud rates, or long-distance transfers. After reading the data, if the number of bits with a value of 1 is odd or even, it's compared with the parity bit (0 for even, 1 for odd). A match indicates no errors, but a mismatch suggests some bits were changed.

- **Stop Bits:** To mark the end of a data packet, the sender raises the line's voltage for at least two-bit lengths, transitioning from low back to high.

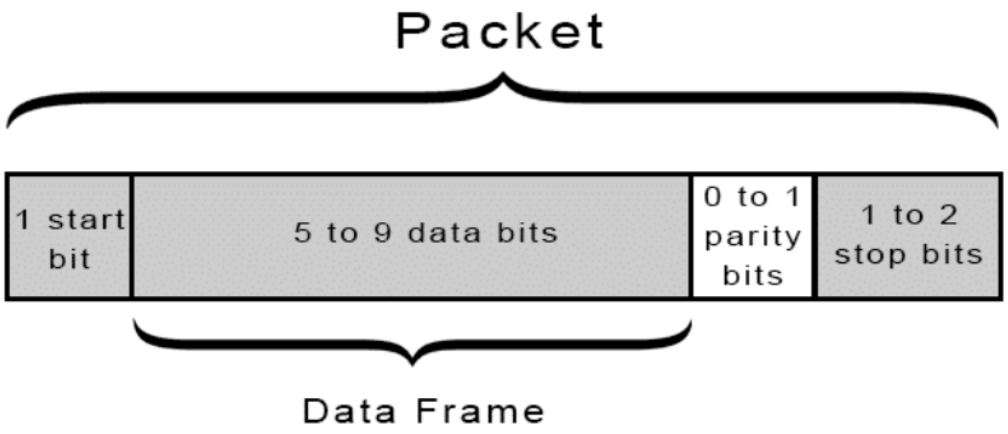

Figure 2 UART Communication Data Packet Structure

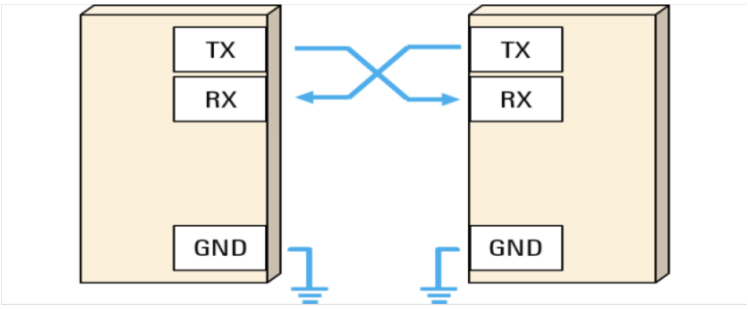

Figure 3 UART Communication [7]

1.4.2 Serial Communication with Arduino [Tx, Rx]

Arduino boards use low-voltage TTL (Transistor-Transistor Logic) signals for serial data exchange through digital pins 0 (RX for receiving) and 1 (TX for transmitting). Unlike traditional RS232 serial ports that operate at higher voltages of +/- 12V, connecting directly to these higher voltages can damage the Arduino. Each Arduino features a built-in serial port known as 'Serial', which interfaces with computers or other devices like a second Arduino, utilizing these two pins and a USB connection. This port is crucial for tasks such as uploading sketches to the board and facilitating the flow of data, such as sensor values or for troubleshooting purposes.[8]

`Serial.available()`: Checks if there's data waiting in the serial receive buffer.
`Serial.begin(baud_rate)`: Initializes serial communication with a baud rate and optionally sets the data format.
`Serial.read()`: Fetches the first byte of data from the serial buffer or returns -1 if there's no data.
`Serial.readBytes(buffer, length)`: Fills a buffer with characters read from the serial port.
`Serial.readString()`: Gathers characters from the serial buffer into a string until it times out.
`Serial.parseInt()`: Scans the incoming serial data for the next whole number.
`Serial.parseFloat()`: Searches for the next floating-point number in the serial data.
`Serial.print()`: Sends data to the serial port as human-readable text.
`Serial.write()`: Outputs binary data to the serial port, either as a byte or in a sequence.[9]

1.4.3 I2C Protocol

Is a two-wire communication protocol commonly used in low-cost, low-power applications like sensors and displays. It requires a clock and data line, enabling synchronous communication for high-speed data transfer. Multiple slaves and masters can be connected to these lines. The Serial Data (SDA) line sends data to the slave device, while the Serial Clock (SCL) line provides synchronization, also need to specify the number of bits that we want to send and also the clock frequency for SCL wire. [10]

1.4.4 I2C Message Packet Format

The data transferred bidirectionally along the Serial Data Line (SDA) by either the master or slave devices, only the master can initiate data transfer, with slaves responding accordingly, where multiple masters can be connected, but only one can be active at a time. The Serial Clock Line (SCL) is always controlled by the master, the communication follows a specific protocol ensuring proper signal handling, the messages are divided into two frames an **address frame** and **data frames**, the address frame identifies the intended slave, while data frames carry 8-bit messages between master and slave.
Each I2C command starts with a START condition and ends with a STOP condition, signaling the beginning and end of a transmission, the master initiates the START condition by pulling SDA low while leaving SCL high, so the bus remains busy until a STOP condition is detected. The address frame precedes data transmission, specifying the destination slave's address, this is crucial for directing data to the correct device, for sure each slave device has a unique address, allowing the master to address individual slaves on the bus, the Figure below shows the I2C Message packet frame.

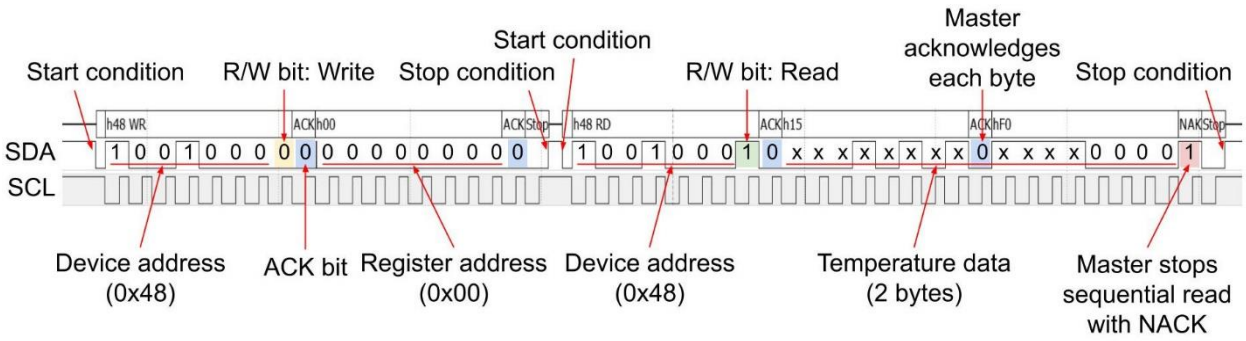

Figure 4 the I2C Message packet frame

- Read/Write Bit: One bit that indicates whether data is being sent from the master to the slave (low voltage level) or received from it (high voltage level).

- ACK/NACK Bit: A message acknowledge/no-acknowledge bit comes after every frame. An ACK bit is sent back to the sender by the receiving device if an address frame or data frame was successfully received.

1.5. LM75B temperature sensor

The LM75B is a digital temperature sensor and thermal watchdog with key features including temperature conversion to digital format using an on-chip sensor, ±2°C accuracy over -25°C to 100°C, and communication via I2C-bus serial interface. It includes data registers for device settings and temperature readings, making it suitable for applications requiring temperature monitoring and control. [11]

2.0 Procedure Decision

2.1 Exp#1: Introduction to Arduino

2.1.1 Photocells

In this part, an LED and a photoresistor are connected to the Arduino using resistors, as depicted in the figure below.

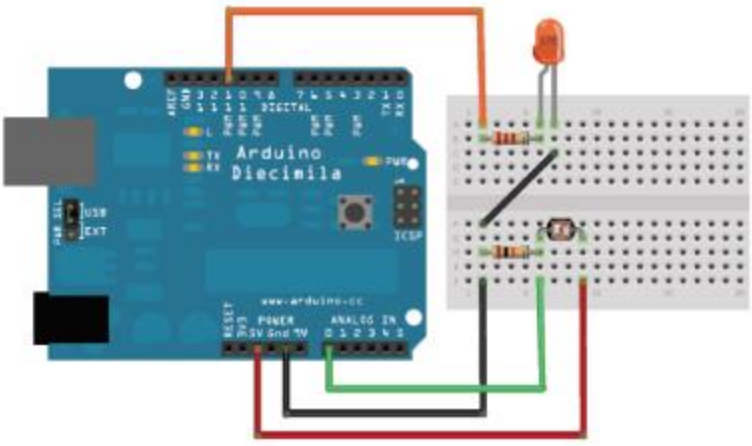

Figure 5 Photocell Arduino Connection

The code was uploaded to the Arduino board, allowing the analog voltage reading from the photocell to determine the brightness of the red LED. The LED is connected to a PWM pin (pin 11) and the photocell is connected between 5V and Analog 0, with a 10K resistor between Analog 0 and ground. During execution, the analog reading from the photocell is obtained using analogRead(), and based on this reading, the code determines whether it is dark, light, or bright. The LED brightness is adjusted by inverting the photocell reading and mapping it from the range of 0-1023 to 0-255 using map(). The calculated LED brightness value is then applied to the LED pin using analogWrite(). The Serial Monitor displays the analog reading and the corresponding brightness level. It can be accessed through Tools -> Serial Monitor.

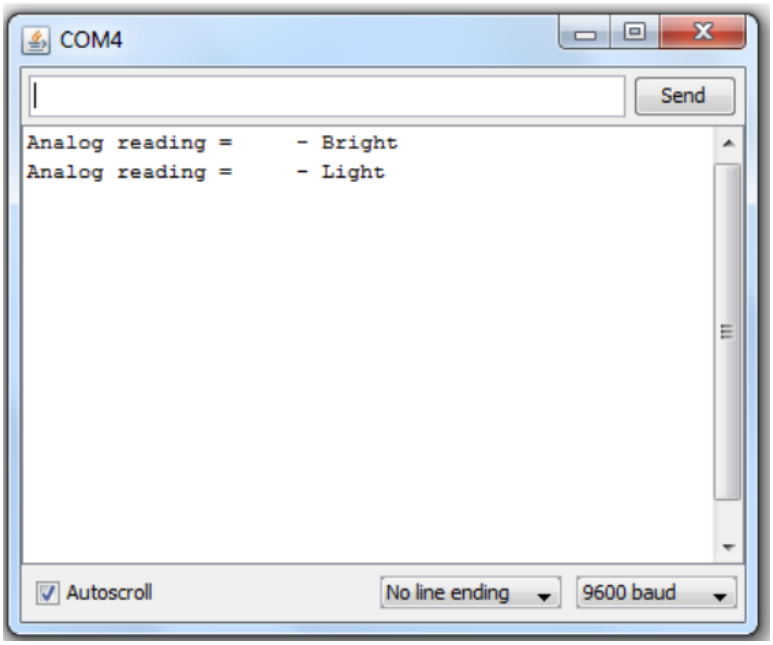

Figure 6 the Photocell status

2.1.2 Hardware Interrupts

Code Appendix A in appendices demonstrates the use of a hardware interrupt in an Arduino program to control an LED's status. It sets up an interrupt on pin 2 (interrupt 0) to detect changes in the signal. When a change occurs, the interrupt triggers the blink() function, which toggles the state of the LED. The main loop constantly writes the current state of the LED to pin 13, resulting in asynchronous blinking. This allows the LED to respond to the external interrupt, such as a push button, by changing its status and creating a blinking effect independent of the program flow.

2.1.3 Software Interrupts

No hardware connections were required for this part. The code provided in Appendix A uploaded to the Arduino board. The code configures Timer1 with a prescaler value of 1024 and preloads the timer with a value of '57723'. This setup ensures that the timer will overflow every half second. The code then enables the timer overflow interrupts, so that when the timer overflows, an interrupt service routine (ISR) is triggered. The ISR is responsible for inverting the state of the Arduino's internal LED, causing the LED to switch its state every half second.

2.2 Exp#2: Serial Communication with Arduino

2.2.1 Basic communication between Arduino & PC

In this part of the experiment, we focused on setting up a simple conversation between a computer and an Arduino UNO using a simple "Hello World!" message. This process involved connecting the two devices with a USB cable and programming the Arduino to communicate at a specific speed, known as the baud rate, set at 9600. The code was writing and uploading to the Arduino that would allow it to receive a message from the computer and then send that same message back, essentially printing what was sent. Through the Arduino's serial monitor, we were able to send our message and see the response from the Arduino, helping us understand the basics of serial communication. the Arduino keeps checking if there is available data, if there is, it reads it byte by byte and prints it to the Serial monitor.

2.2.2 Basic communication between 2 Arduinos

In this Part of our experiment, we made two Arduino UNO boards talk to each other by sending a "Hello World!" message back and forth. We used wires to connect them. One wire went from the Tx (send) pin on the first Arduino to the Rx (receive) pin on the second Arduino, and vice versa, to establish a bidirectional communication pathway. The ground was also shared between the two devices to ensure a common reference point for the electrical signals.

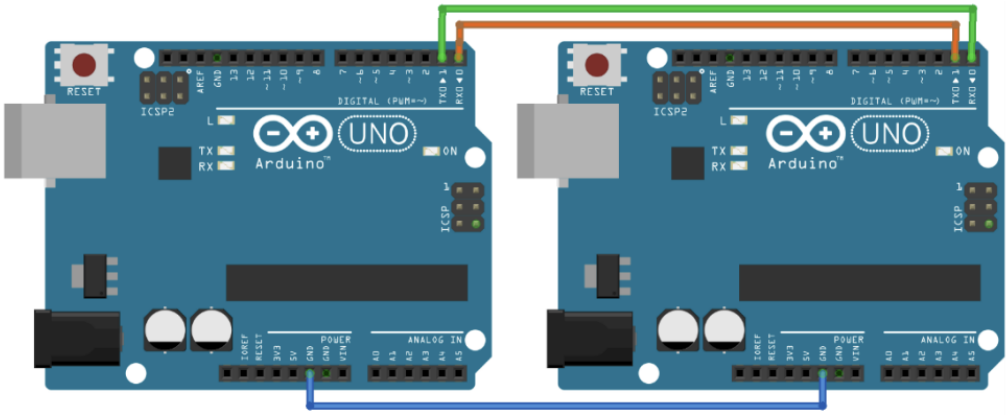

Figure 7Connecting 2 Arduinos using TX & RX

We uploaded the code to each Arduino: one acted as the sender, tasked with transmitting the "Hello World!" message, and the other as the receiver which displays the received message. This setup was carefully prepared to operate at a synchronized baud rate of 9600, ensuring that both Arduino could send and receive data without any miscommunication, the sender Arduino was programmed to continuously transmit the "Hello World!" message at consistent intervals. the code of sender on the Appendix B initializes serial communication at a baud rate of 9600 and, within an infinite loop, sends out the "Hello World!" message followed by a one-second pause (delay(1000);).On the receiving end, the Arduino was set up to be always on the lookout for incoming data. When data arrives, the Arduino captures it and displays it on the Serial Monitor. In setup of receiver code , Serial.available() checks for any incoming messages. When a message is detected, Serial.readString() reads the entire string and Serial.println() then prints this message to the Serial Monitor. This process ensures that any message sent by the sender is displayed by the receiver, allowing for real-time monitoring of the successful data transmission between the two Arduinos.

2.2.3 Push Button & Ultrasonic with LED using 2 Arduinos

In this Part we established a communication link between two Arduinos where one Arduino (Arduino1) sends a signal to the other (Arduino2) upon the press of a push button. When receiving the signal, Arduino2 activates an LED and triggers an ultrasonic sensor to measure distance, which shows how devices can interact and respond to physical inputs through serial communication.

In the code at APPENDIX B, we handled the ultrasonic sensor by first sending a short pulse, we used (digitalWrite(trigPin, HIGH);) followed by a short delay with(delayMicroseconds(10);) and then (digitalWrite(trigPin, LOW);) to send the pulse from the Arduino to the sensor's trigger pin. After this pulse, we listen on the sensor's echo pin for the sound wave to bounce back by (pulseIn(echoPin, HIGH);). The time it takes for the echo to return is measured. This duration is then used to calculate the distance to an object, based on the speed of sound via this formula (distance = duration * 0.0343 / 2;). The maximum range of ultrasonic sensors is typically 4 to 7 meters, while the minimum range is around 2 to 3 centimeters.

2.2.4 Visualization of serial communication using Serial Plotter

In Part 4 of our experiment, we focused on visually understanding serial communication by using the Arduino's Serial Plotter to display the transmission of data bits, including start, data, parity, and stop bits. By programming an Arduino to send the character 'A' at a low baud rate, we were able to slow down the transmission enough to observe each bit's transition on the Serial Plotter.

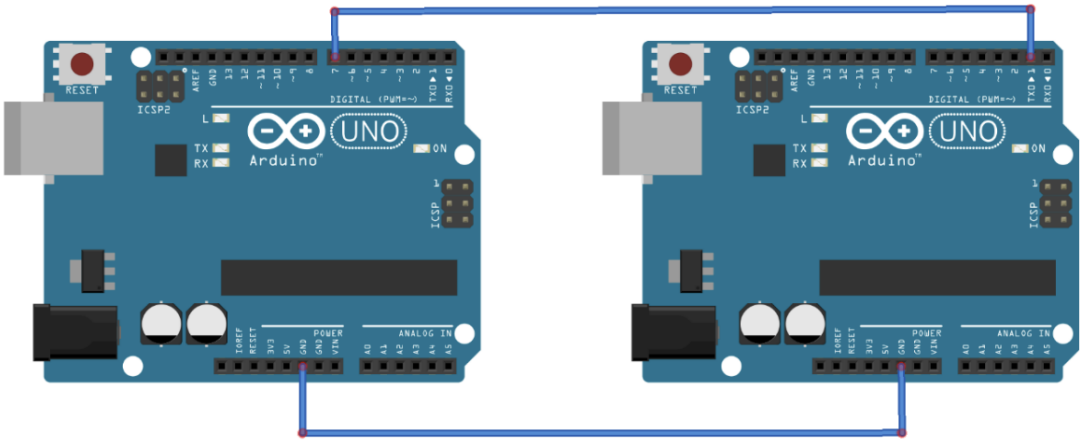

Figure 8 Arduino 1tx connected with Arduino2 PIN_7 with common GND

We configured the transmitter Arduino to send a character ('A') at a low baud rate (e.g., 300 baud) to facilitate clear visualization of individual bits on the plotter. Meanwhile, the receiver Arduino's serial monitor was initialized at a higher baud rate (19200 baud) to ensure accurate reception of the transmitted data. and the data sent from the sender is received as regular input at pin 7, then plotted on the serial monitor. This difference in baud rates between the sender and receiver allowed for a detailed visualization of the waveform.

The expected output of the visualization in Part 4 of the experiment would typically show a waveform representing the transmission of the character 'A' at a low baud rate, as configured in the experiment setup. Initially, the waveform would be a high state, indicating idle line conditions. as the byte transmission begins, the waveform would transition to a low state, signaling the start bit of the data packet. Following this, the individual bits corresponding to the character 'A' (01000001) would be transmitted sequentially, starting from the least significant bit (1) to the most significant bit (0). Once all the byte bits are transmitted, the waveform would return to a high state, indicating the end of the transmission. It's important to note that in this particular setup, we do not expect to see a parity bit at the end of the transmission, as it was not included in the configuration.

2.3. EXP#3: Two wires interface I2C

2.3.1 Writing the LCD to Arduino and I2C scanning

in this part, the LCD configured with Arduino at unique address using I2C communication protocol, the LCD function as a slave, and the Arduino function as the master, The GND, VCC, SDA, and SCL of the LCD were connected to the GND, 5V, A4, and A5 of the Arduino, after the connection done, the I2C Scanner sketch code uploaded to scan the address of the LCD.

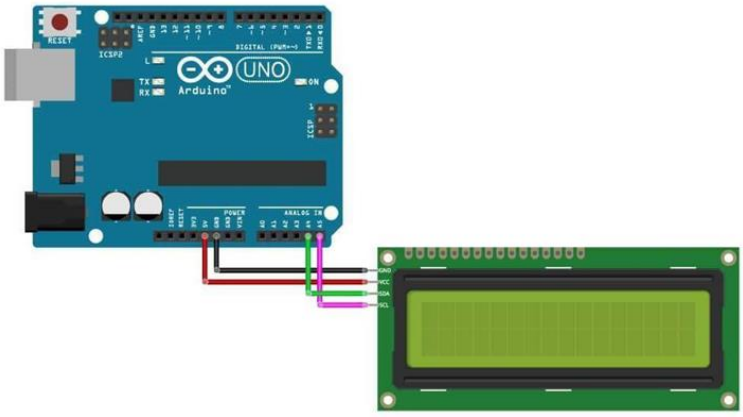

Figure 9 I2c module connection between Arduino and LCD

the code at Appendix C performs the actual scanning of I2C addresses, where it iterates through addresses from 1 to 127 and attempts to communicate with each address using Wire.beginTransmission() and Wire.endTransmission(). If no error occurs (error == 0), it indicates that a device is connected at that address and prints into the serial monitor.

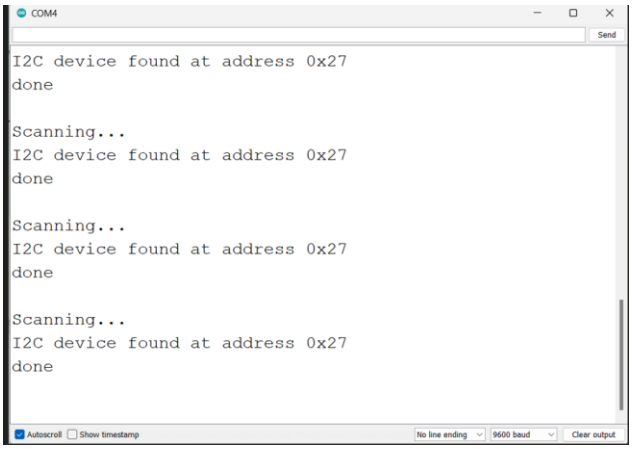

Figure 10 LCD Address

2.3.2 Display static text on the LCD

In this part the connection remaining the same as the previous part, as shown at the code at Appendix C that we define the LCD columns to 16 and the rows to 2, then we Initialize an LCD Object by using LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows), where 0x27 is the I2C address of the LCD determined on the previous part , then we set the cursor to the first column of the first row by using lcd.setCursor(0, 0), to print "Welcome TO", and set the cursor to the first column of the second row by using lcd.setCursor(0, 1), to print "BZU", and we turn the LCD backlight on by using lcd.backlight(). The result of the code shown on the image below.

Figure 11 Display welcome to BZU on the LCD

2.3.3 I2C communication between 2 Arduinos

In this part we will make a communication between two Arduino devices one as a master and the other one as a slave, the slave will be sending a data from 0 to z to the master. Then the connections done by linking A4(SDA), A5(SCL), and GND of the master Arduino to A4, A5, and GND of the slave Arduino, the code at Appendix C shows the communication between a master and a slave Arduino using the I2C protocol by using only two wires which one used for data (SDA) and the other one for a clock (SCL),we initialize the communication by using Wire.begin() for I2C communication, and Serial.begin() for serial communication for both master and slave Arduinos, in the master side has an optional address, while the slave has a written address, then the master sends a request to the slave, at address 2 by using Wire.requestFrom() method, which taking two variables the slave address and the number of bytes read each time. On the slave side it handles the request event by using Wire.onRequest(requestEvent), then the slave sends the first byte of data, followed by the acknowledgement bit, then the master checks if there any data is available and prints it on the serial monitor as shown on the Figure below.

Figure 12 The results of the communication between the 2 Arduinos

2.3.4 I2C communication between 2 Arduinos and LCD

In this part we want to read analog value (potentiometer) from slave Arduino and write the value on LCD connected to Arduino master device, so we connect the LCD to the master Arduino on the SDA, and SCL pins, and the potentiometer connect at A0 on the slave Arduino, the code at Appendix C show that the master Arduino establish I2C communication with a slave device and display data on an LCD screen requests data from the slave device using Wire.requestFrom(), reads the MSB and LSB bytes, combines them to obtain the analog value, and prints this value to the LCD, where The slave Arduino, initialized the I2C communication with a specific address, handle the data which requested by the master by using Wire.onRequest(requestEvent), and the Arduino

slave read the analog value from A0, then split it into MSB and LSB bytes to send them to the master Arduino when requested, the figures below shows the reading of potentiometer value from the slave and print it on the LCD connected with the master.

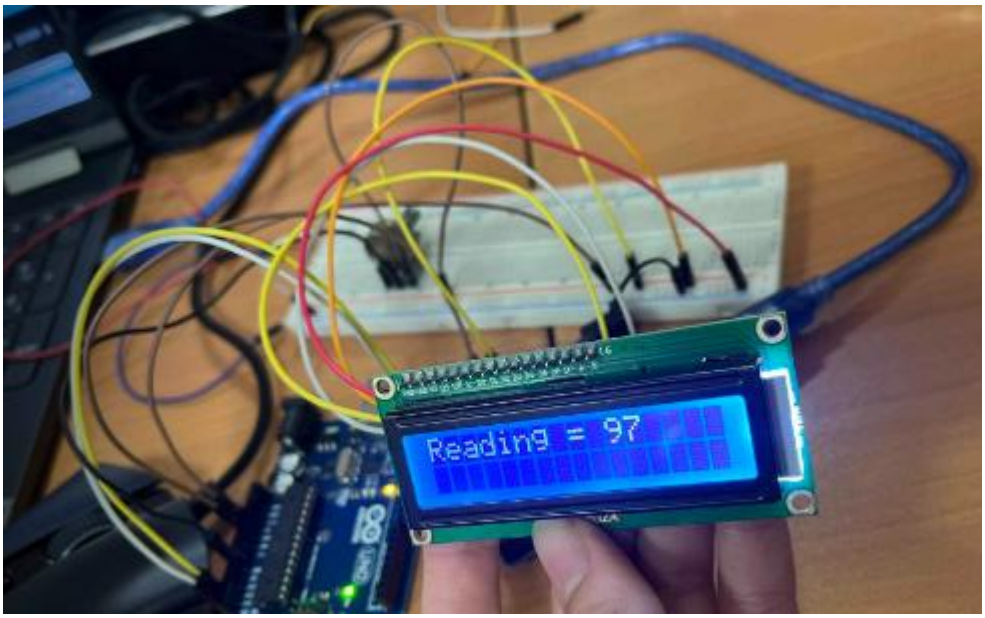

Figure 13 I2C communication between 2 Arduinos and LCD I

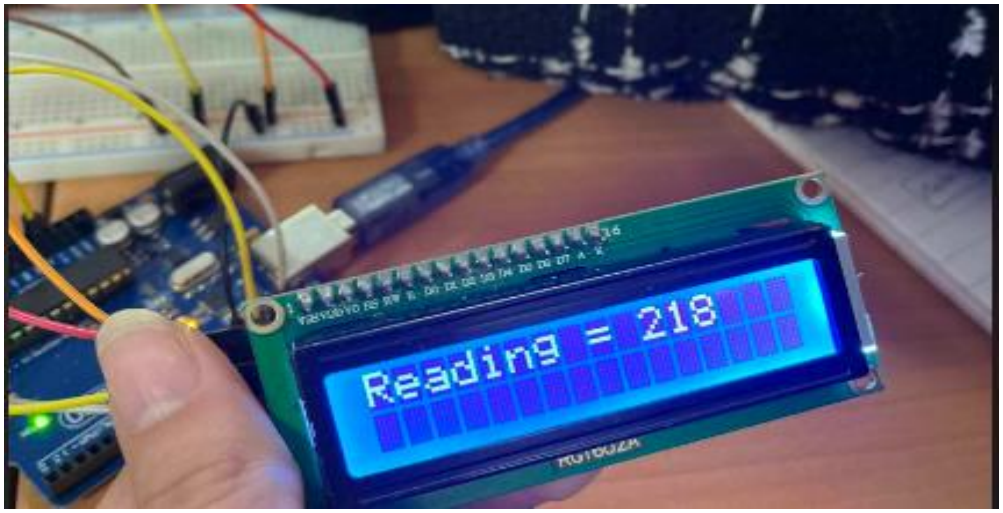

Figure 14 I2C communication between 2 Arduinos and LCD II

Conclusion

In Experiment #1, focused on constructing a circuit for controlling the brightness of an LED. This circuit allowed for the adjustment of brightness based on the detected light levels using an LDR (Light Dependent Resistor). Subsequently, we developed basic programs for switching the LED on and off. These programs were designed to respond to both software and hardware interrupts, providing flexibility in controlling the LED's operation. In Experiment #2, we covered the basics of serial communication using Arduino. We explored sending and receiving data between Arduinos and a computer, enabling direct communication between two Arduinos, integrating additional components for interactive systems, and visualizing serial data transmission using the Serial Plotter. Moving forward, this knowledge and experience will be invaluable in designing and implementing a wide range of Arduino-based projects and applications. Experiment #3 focused on the I2C communication protocol, connecting an LCD to an Arduino as a master-slave setup, utilize an I2C Scanner sketch to identify the LCD's address and control it using the LiquidCrystal_I2C library. Exploring bidirectional communication, we implemented a master-slave scenario where the master Arduino requested data from the slave, Additionally, we integrated a potentiometer with the slave Arduino to read analog values, displaying them on the LCD on the master Arduino.

References

[1] Arduino Uno Pins - A Complete Practical Guide - The Robotics Back-End (roboticsbackend.com)

[2] Photocell : Types, Circuit, Working and Its Applications (watelectrical.com)

[3] Using Arduino Interrupts - Hardware, Pin Change and Timer (dronebotworkshop.com)

[4] Serial Communication. How serial communication works? (serial-port-monitor.org)

[5] Serial Communication: How It Works, Types, & Pros Cons of Each Model (fullyinstrumented.com)

[6] Basics of UART Communication (circuitbasics.com)

[7] Understanding UART | Rohde & Schwarz (rohde-schwarz.com)

[8] docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-monitor

[10] I2C Communication Protocol Basics Working and Applications (microcontrollerslab.com)

[11] LM75B.pdf (nxp.com)

APPENDIX A

Introduction to Arduino

Part 1 Photocell

```
int photocellPin = A1;
int photocellReading;
int LEDpin = 11;
int LEDbrightness;
void setup(void) {
Serial.begin(9600);
 pinMode(photocellPin, INPUT);
 pinMode(LEDpin, OUTPUT);
}
void loop(void) {
 photocellReading = analogRead(photocellPin);
Serial.print("Analog reading = ");
 Serial.print(photocellReading);
if (photocellReading > 300) {
 Serial.println(" - Dark");
 }
 else if (photocellReading < 800) {
 Serial.println(" - Light");
 }
 //else {
 // Serial.println(" - bright");
// }
 photocellReading = 1023 - photocellReading;
 LEDbrightness = map(photocellReading, 0, 1023, 0, 255);
 analogWrite(LEDpin, LEDbrightness);
```

```
 delay(1000);
}
```

Part 2 Hardware interrupt

```
// Hardware interrupt example
int pin = 11;
int pin_button  = 2;
volatile int state = LOW;
void setup()
{
pinMode(pin, OUTPUT);
pinMode(pin_button, INPUT);
attachInterrupt(0int pin = 11;
int pin_button  = 2;
volatile int state = LOW;
void setup()
{
pinMode(pin, OUTPUT);
pinMode(pin_button, INPUT);
attachInterrupt(0, blink, RISING);
}
void loop()
{
digitalWrite(pin, state);
}
void blink()
{
state = !state;
}
, blink, RISING);
```

```
}
void loop()
{
digitalWrite(pin, state);
}
void blink()
{
state = !state;
}
```

Part 3 Software Interrupt

```
// Timer1 overflow interrupt example

#define ledPin 13

void setup() {

 pinMode(ledPin, OUTPUT);

 // initialize timer1

 noInterrupts(); // disable all interrupts

TCCR1A = 0;

 TCCR1B = 0;

 TCNT1 = 57724;

TCCR1B |= (1 << CS10);

 TCCR1B |= (1 << CS12);

 TIMSK1 |= (1 << TOIE1);

interrupts(); // enable all interrupts

}

ISR(TIMER1_OVF_vect)

{

 TCNT1 = 57724; // preload timer

 digitalWrite(ledPin, digitalRead(ledPin) ^ 1);

}

void loop()

{

 // your program here...

}
```

Appendix B

Part 1 Basic communication between Arduino & PC

```
/* Use a variable called byteRead to temporarily store
the data coming from the computer */
byte byteRead;
void setup() {
// Turn the Serial Protocol ON
Serial.begin(9600);
}
void loop() {
/* check if data has been sent from the computer: */
if (Serial.available()) {
/* read the most recent byte */
byteRead = Serial.read();
/*ECHO the value that was read, back to the serial port. */
Serial.println(byteRead,DEC);
}
}
```

Part 2 Basic communication between 2 Arduinos

Receiver

```
void setup() {
// Begin the Serial at 9600 Baud
Serial.begin(9600);
Serial.setTimeout(10000);
}
void loop() {
if (Serial.available())
{
String data = Serial.readString();//Read the serial data and store in var
Serial.println(data);//Print data on Serial Monitor
}
}
```

sender

```
void setup() {
// Setup the Serial at 9600 Baud
Serial.begin(9600);
}
void loop() {
Serial.println("Hello World!"); //Write the serial data
delay(500);
}
```

Part 4 Visualization of serial communication using Serial Plotter

Receiver

```
int readPin = 7;
void setup() {
// Begin the Serial at 19200 Baud
Serial.begin(19200);
pinMode(readPin, INPUT);
}
void loop() {
Serial.println(digitalRead(readPin));
}
```

Part 4 Visualization of serial communication using Serial Plotter

sender

```
void setup() {
// Setup the Serial at 300 Baud
Serial.begin(300);
}
void loop() {
Serial.write('A'); //Write the character A => will be transmitted as Byte [41H]
delay(500);
}
```

APPENDIX C

Part 1 Writing the LCD to Arduino and I2C scanning

```
#include <Wire.h>
void setup() {
Wire.begin();
Serial.begin(9600);
Serial.println("\nI2C Scanner");
}
void loop() {
byte error, address;
int nDevices;
Serial.println("Scanning...");
nDevices = 0;
for(address = 1; address < 127; address++ ) {
Wire.beginTransmission(address);
error = Wire.endTransmission();
if (error == 0) {
Serial.print("I2C device found at address 0x");
if (address<16) {
Serial.print("0");
}
Serial.println(address,HEX);
nDevices++;
}
else if (error==4) {
Serial.print("Unknow error at address 0x");
if (address<16) {

Serial.print("0");
}
Serial.println(address,HEX);
}
}
if (nDevices == 0) {
Serial.println("No I2C devices found\n");
}
else {
Serial.println("done\n");
}
delay(5000);
}
```

Part 2 Display static text on the LCD

```
#include <LiquidCrystal_I2C.h>

// set the LCD number of columns and rows
int lcdColumns = 16;
int lcdRows = 2;
// set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
void setup(){
// initialize LCD
lcd.init();
// turn on LCD backlight
lcd.backlight();
}
void loop(){
// set cursor to first column, first row
lcd.setCursor(0, 0);
// print message
lcd.print("Welcome TO");
delay(1000);
// clears the display to print new message

// set cursor to first column, second row
lcd.setCursor(0,1);
lcd.print("BZU");
delay(2000);
lcd.clear();
}
```

Part 3 I2C communication between 2 Arduinos

Master

```
// master side code
#include <Wire.h>
void setup()
{
Wire.begin(); // join i2c bus (address optional for master)
Serial.begin(9600); // start serial for output
Serial.print("master sleeping...");
delay(2000);
Serial.println("go");
}
void loop(){
Wire.requestFrom(2, 1); // request data from slave device #2
while(Wire.available()){
char c = Wire.read(); // receive a byte as character
Serial.print(c); // print the character]
}
delay(100);
}
```

Slave

```
// slave side code

#include <Wire.h>
void setup(){
Wire.begin(2); // join i2c bus with address #2
Wire.onRequest(requestEvent); // register event
Serial.begin(9600); // start serial for output
}
void loop(){
delay(100);}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void requestEvent(){
  static char c = '0';
Wire.write(c++);
if (c > 'z')
c = '0';
}
```

Part 4 I2C communication between 2 Arduinos and LCD

master

```
// master side code
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
// set the LCD number of columns and rows
int lcdColumns = 16;
int lcdRows = 2;
// set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
void setup()
{
Wire.begin(); // join i2c bus (address optional formaster)
Serial.begin(9600); // start serial for output
Serial.print("master sleeping...");
delay(2000);
Serial.println("go");
// initialize LCD
lcd.init();
// turn on LCD backlight
lcd.backlight();
}
void loop()
{
Wire.requestFrom(50, 2);
#2
// request data from slave device
int res;//result from I2c reading
byte MSB = Wire.read(); /* receive a byte MSB(NOTE: the
function is blocking, so it would not continue the code until a
byte reach)*/
byte LSB = Wire.read(); /* receive a byte LSB(NOTE: the
function is blocking, so it would not continue the code until a
byte reach)*/
res = ((MSB << 8) | LSB);
Serial.print("MSB = ");
Serial.print(MSB);
Serial.print(" , LSB = ");
Serial.print(LSB);
Serial.print(" , analog value = ");
Serial.println(res); // print the analog value
// set cursor to first column, first row
lcd.setCursor(0, 0);
```

```
// print message
lcd.print("Reading = ");
lcd.print(res);
delay(1000);
lcd.clear();
}
```

slave

```
#include <Wire.h>
void setup()
{
Wire.begin(50); // join i2c bus with address #2
Wire.onRequest(requestEvent); // register event
Serial.begin(9600); // start serial for output
}
void loop()
{
delay(100);
}
// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void requestEvent()
{
int value = analogRead(A0);
Wire.write(value >> 8); // send the MSB
Wire.write(value & 0x00ff); // send the LSB
}
```