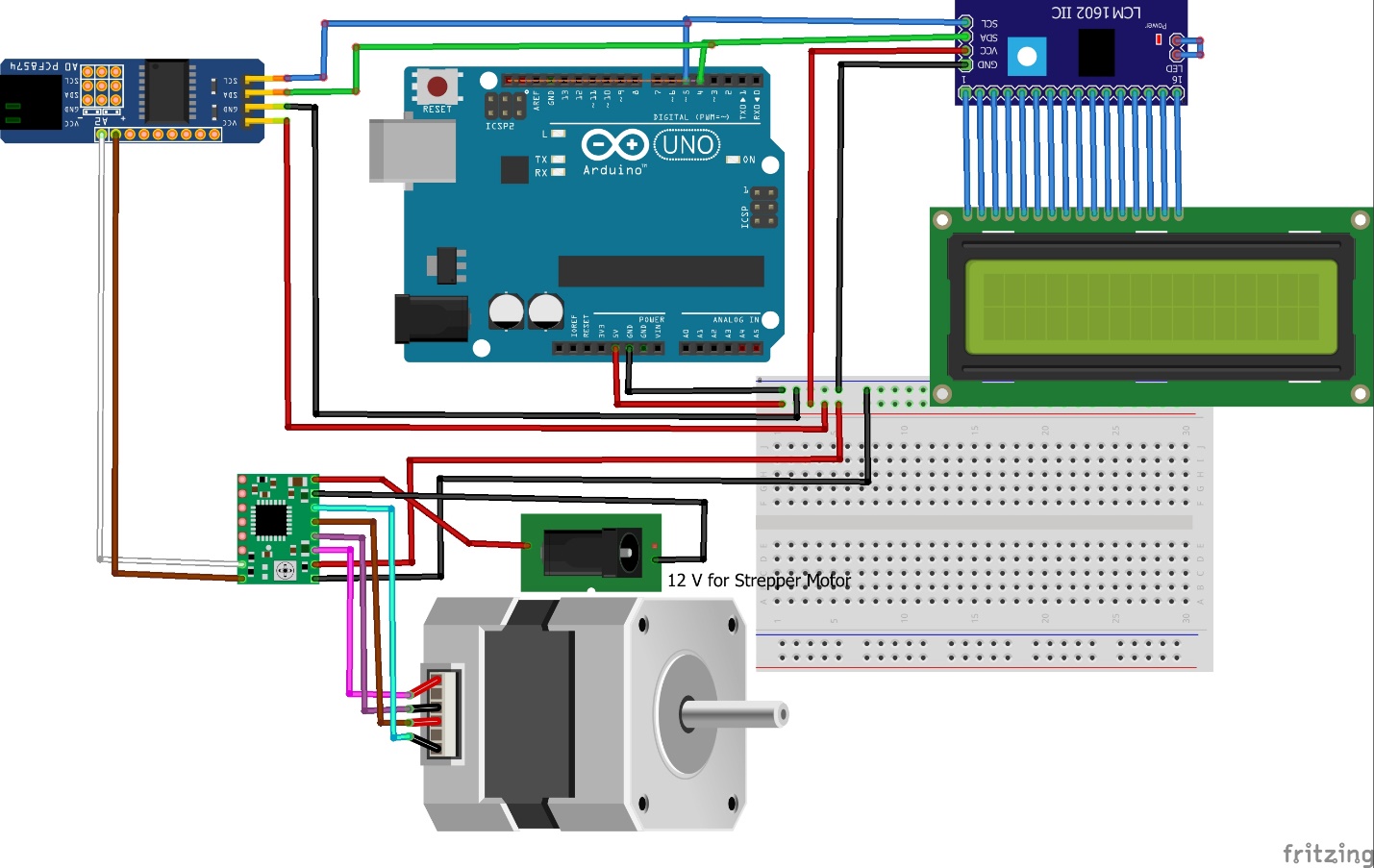
**I2c implementation without any library for LCD1602A and Stepper Motor**

**Circuit Diagram:**

****

**List Of Components :**

1. **Arduino Uno**
2. **Lcd 1602A**
3. **Pcf8574 Backpack for lcd**
4. **Pcf8574 extender**
5. **Stepper Motor**
6. **A4988 Stepper Driver**

**Explanation of Code :**

**I2clcd\_final.ino**

**Header Files:**

**"i2c.h":** this file contains functions for handling I2C communication, like i2cStart() and i2cSendByte().

"lcd.h": this file contains functions to manage the LCD display, such as initLCD() and sendStringToLCD().

"stepper.h": this file holds definitions and functions related to the stepper motor control, like moveAndDisplayStepperDirection().

**Global Variables:**

inputString: A variable to collect characters received from the serial port until a newline character is found.

stringComplete: A flag indicating when a complete string has been received from the serial port.

**setup() Function:**

This function is executed once when the program starts:

* Initializes the I2C pins (SDA and SCL) as outputs.
* Initializes serial communication at 9600 baud for communication with the computer.
* Sends an initial message to the serial monitor.
* Initializes the LCD.
* Calls moveAndDisplayStepperDirection(1) to move the stepper motor one step.
* Displays a message on the LCD and the serial monitor indicating that the stepper has moved.

**Loop() Function:**

This function is called repeatedly:

* Checks if a complete string has been received from the serial monitor.
* If a complete string is received, it converts the string to an integer and moves the stepper motor accordingly using moveAndDisplayStepperDirection(steps).
* Resets the inputString and stringComplete flag for the next input.
* **serialEvent()** Function: Reads characters from the serial port as they become available.
* Appends them to inputString until a newline character ('\n') is found, then sets stringComplete to true.

**I2c.h File:**

The file i2c.h is a header file defining the interface for I2C communication on an Arduino. Here's an explanation of its contents:

**Preprocessor Directives:**

#ifndef I2C\_H and #endif: These are used to prevent double inclusion of the file. If the file is included more than once in a compilation unit, the preprocessor will ignore everything between these directives after the first inclusion.

**Inclusions:**

<stdint.h>: This includes definitions for standard integer types, such as uint8\_t.

<Arduino.h>: This includes the basic definitions and functions of the Arduino platform, enabling the code to access standard Arduino functionality.

**Definitions:**

**SDA\_PIN 4 and SCL\_PIN 5:** These definitions set the pin numbers for the Serial Data Line (SDA) and Serial Clock Line (SCL) for I2C communication. You can change these values to the appropriate pins depending on your hardware configuration.

**Function Prototypes:**

**void i2cStart();**: This function is to initiate the start condition for I2C communication, signaling to connected devices that communication is beginning.

**void i2cStop();**: This function is to send the stop condition for I2C communication, signaling that communication is ending.

**void i2cSendByte(uint8\_t data);**: This function is used to send a byte of data (data) over the I2C bus to a connected device.

**I2c.cpp File:**

The i2c.cpp file contains the implementations for the functions declared in the i2c.h header file. These functions control the I2C communication protocol, specifically dealing with starting and stopping communication and sending bytes. Here's a detailed explanation of each function.

**i2cStart():**

* This function initiates the start condition for the I2C communication protocol.
* The SDA and SCL pins are set high, and then the SDA pin is pulled low followed by the SCL pin being pulled low.
* This specific sequence of events indicates the start of a transmission to all devices connected to the I2C bus.

**i2cStop():**

* This function sends the stop condition for I2C communication.
* The SDA pin is pulled low, then the SCL pin is pulled high, followed by the SDA pin being pulled high.
* This sequence of events signals the end of the transmission and releases the bus.

**i2cSendByte(uint8\_t data):**

* This function sends a byte of data over the I2C bus.
* The data is sent bit-by-bit over the SDA line, with the most significant bit first. The SCL line is used to synchronize the sending of these bits.
* If a particular bit is 1, the SDA line is set high; otherwise, it is set low.
* After each bit is set on the SDA line, the SCL line is pulsed high then low to signal to the receiving device to read the bit.
* The data is shifted left by one bit after each bit is sent, so the next bit in line becomes the most significant bit.
* After all 8 bits are sent, an acknowledgment is expected from the receiving device. The SDA line is switched to input mode, and the code waits for the SDA line to go low, which signals the acknowledgment.
* After receiving the acknowledgment, the SDA pin is reverted to output mode to continue the transmission.

**Lcd.h File**

The lcd.h file contains declarations for the functions and definitions for the macros that are used to control an LCD display connected via the I2C bus. Here's an explanation of what each part does

**Macro Definitions:**

**LCD\_ADDR:** The I2C address for the LCD, set to 0x3F. This is the specific address the code will use to communicate with the LCD.

**BACKLIGHT:** Controls the LCD's backlight, defined as bit 3 of the byte (0x08).

**EN:** Enable bit for controlling the LCD, defined as bit 2 of the byte (0x04).

**Function Declarations:**

**initLCD():**

This function is responsible for initializing the LCD. It will contain all the necessary setup instructions to prepare the LCD for display.

**sendToLCD(uint8\_t data, bool isChar = false):**

This function sends a byte of data to the LCD.

The data parameter is the byte to be sent.

The isChar parameter is a boolean that defaults to false. If isChar is true, the data is treated as a character to be displayed; otherwise, it is treated as a command for the LCD.

**sendStringToLCD(const char \*str):**

This function takes a string (a pointer to a sequence of characters) and sends each character to the LCD to be displayed.

The str parameter is a pointer to the string of characters to be displayed.

**Lcd.cpp File**

This code is part of the implementation for controlling an LCD display using the I2C protocol, corresponding to the declarations found in lcd.h.

**sendToLCD(uint8\_t data, bool isChar = false):**

* This function sends a single byte of data to the LCD, taking care to split the byte into two nibbles (4-bit groups) and send them separately, as is typical for many LCDs.
* The data parameter is the byte to be sent to the LCD.
* The isChar parameter is a flag to determine whether the data should be treated as a character to display (true) or a command (false).
* The upper and lower nibbles of the byte are split and sent to the LCD with the appropriate control signals. The BACKLIGHT bit is used to keep the backlight on, and the EN bit is used to enable writing the data to the LCD.
* If isChar is true, bit 0 is set to tell the LCD that the data is a character to display, not a command.

**initLCD():**

This function initializes the LCD, preparing it for display.

* The series of calls to sendToLCD with specific hexadecimal values are commands that configure the LCD to a common initialization sequence. This includes setting the LCD to a 2-line display with 5x8 dot characters (0x28), turning on the display and cursor (0x0C), setting the cursor move direction (0x06), and clearing the display (0x01).
* A delay(5) is included to give the LCD time to process these commands.

**sendStringToLCD(const char \*str):**

* This function takes a pointer to a string of characters and sends each one to the LCD using the sendToLCD function.
* The loop continues sending characters to the LCD until it reaches the null terminator (\0) at the end of the string.

**Stepper.h File**

The stepper.h file defines the interface for controlling a stepper motor using the A4988 driver. It includes the following components:

**Macro Definition:**

#define STEPPER\_ADDR 0x20: This line defines the address of the stepper motor driver as 0x20. This is likely used to communicate with the driver over I2C.

**Function Declarations:**

**void moveAndDisplayStepperDirection(int steps)**: This function is declared to take an integer steps that represents the number of steps the stepper motor should move. If the value is positive, the stepper motor will move in one direction (e.g., clockwise), and if it's negative, it will move in the other direction. The movement and direction will also be displayed.

**void controlA4988(bool enable, bool direction):** This function is declared to control the A4988 stepper motor driver. The enable parameter is used to either enable or disable the motor driver, and the direction parameter is used to set the movement direction of the stepper motor.

**Stepper.cpp File**

The stepper.cpp file contains the implementation of two functions related to controlling a stepper motor using an A4988 stepper motor driver and a PCF8574 I2C expander. Here's the explanation of the functions:

**controlA4988(bool enable, bool direction):**

* Parameters: enable (to turn on/off the driver), direction (to set the movement direction).
* Operation: This function configures the control pins for the A4988 driver.
* enable is used to set or unset the Enable pin (p1).
* direction is used to set the Direction pin (p0).
* I2C Communication: The control pins' status is sent to the PCF8574 I2C expander at the stepper motor's address.
* Use Case: This function is called to enable/disable the stepper motor driver and set its direction.

**moveAndDisplayStepperDirection(int steps):**

* Parameters: steps (the number of steps to move the stepper motor).
* Operation: This function moves the stepper motor and displays its direction.
* Determines the direction based on whether steps is positive or negative.
* Calls controlA4988(true, direction) to enable the stepper and set the direction.
* Sends pulses to the STEP pin (controlled by P1) the specified number of times, thus causing the stepper to move.
* Calls controlA4988(false, direction) to disable the stepper after movement.
* Display: The function sends messages to an LCD displaying the movement and direction, and prints the same to the Serial Monitor.
* Use Case: This function is used to move the stepper motor by a specified number of steps and inform the user about the movement and direction.