VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

– – – – – – – – – –

LOGIC DESIGN PROJECT

EXPANSION SHIELD FOR STM32 NUCLEO INTERFACING WITH LCD1602 AND DHT20 SENSOR VIA I^2C COMMUNICATION PROTOCOL

MAJOR: COMPUTER ENGINEERING

COUNCIL:	COMPUTER ENGINEERING 2
INSTRUCTOR:	Dr. LÊ TRỌNG NHÂN
COUNCIL SECRETARY:	Assoc. Prof. PHẠM QUỐC CƯỜNG
REVIEWER:	Assoc. Prof. Dr. TRẦN NGỌC THỊNH

Student 1: Cao Minh Quang
Student 2: Lê Tuấn Hưng
Student 3: Trần Cao Duy Trường

Ho Chi Minh City, December 2022

COMMITMENT

We pledge that this project is based on our supervisors' ideas and knowledge. All studies and references are reliable and honest. The group completed the project requirements set by faculty of computer science and engineering - department of Computer Engineering.

Sincerely,

Cao Minh Quang
Lê Tuấn Hưng
Trần Cao Duy Trường

ACKNOWLEDGEMENT

First and foremost, We would express our most profound appreciation to our thesis supervisors, Dr. Lê Trọng Nhân. He has been there, providing his heartfelt support and guidance at all times. He has given us invaluable guidance, inspiration, and suggestions in our quests for knowledge during our university time. Without his assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished.

We sincerely thank the teachers who occupy the Faculty of Computer Science and Engineering in particular and the Ho Chi Minh City University of Technology in general, who have constantly been imparting knowledge in the past four years. Their support, encouragement, and credible ideas have been great contributors to the completion of the thesis.

Last but not least, It would be inappropriate if we omit to thank our friends and family. Our late parents' unconditional love and blessings, the care of friends and acquaintances who never let things get dull have all made a tremendous contribution in helping us reach this stage in our life. We thank them for putting up with us in difficult moments where we felt stumped and for goading us on to reach for our passions.

Finally, we would like to wish you good health and success in your noble life.

ABSTRACT

The project consists of eight chapters to explain our work obviously, and The last chapter is dedicated to the conclusion from our perspective. Our final result allows the system to measure the temperature and humidity of the environment where the system is placed. The information of this examination includes the temperature in degree Celsius and the humidity in percentage . Those values then will be displayed on the LCD 1602. Additionally, we create the UART interface for user to control the behaviour of the system.

Keyword for our work: I^2C & UART connection protocol, LCD 1602 & DHT20 Sensor, STM Nucleo Development Board, STM32F103RBT6 MCU.

Table of contents

List of figures

Tables

CHAPTER 1

INTRODUCTION

In this project, we aim to design and implement an extension shield for the STM Nucleo Development Board. The main functionality of this shield is to sequentially read the real-time temperature and humidity using the DHT-20 sensor. Meanwhile, the result will be showed on a LCD 16×2.

Initially, the design stage requires us to work with the Altium Designer application. With the assistance from the instructor, we are able to print out the circuit board and solder all necessary components such as the DHT-20 sensor, LCD 16×2, as well as resistors, capacitors,etc. to ensure the appropriate operation of the shield.

Our main components use the I2C communication protocol and the microcontroller STM32-F103RBT6 on the STM Nucleo board does support this type of communication. Besides, the UART protocol is also needed to transmit control command from PC to the hardware.

Overall, the design step is quite simple since we have the support from the component library and the design patter provided by the instructor. However, the programming work is much more complicated as we have to deal with a new communication protocol and interact with different devices.

CHAPTER 2

THEORETICAL BASIS

This chapter will present knowledge that contributes to the implementation of our system, including some well-known communication protocol for interacting with peripheral devices including LCD and DHT-20 Sensor as well as control the operation of the system from the PC.

2.1 Universal Asynchronous Receiver - Transmitter (UART)

2.1.1 Introduction to UART Communication Protocol

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:

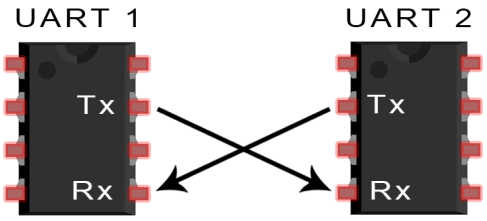

Figure 2.1: UART Communication

UARTs transmit data asynchronously, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate. Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must operate at about the same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off.

Both UARTs must also must be configured to transmit and receive the same data packet structure.

Wires Used	2
Maximum Speed	Any speed up to 115200 baud, usually 9600 baud
Synchronous or Asynchronous ?	Asynchronous
Serial or Parallel ?	Serial
Maximum number of Masters	1
Maximum number of Slaves	1

Table 2.1: UART Communication Properties

2.1.2 UART Protocol Working Principle

The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller.

Data is transferred from the data bus to the transmitting UART in parallel form. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet.

Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end.

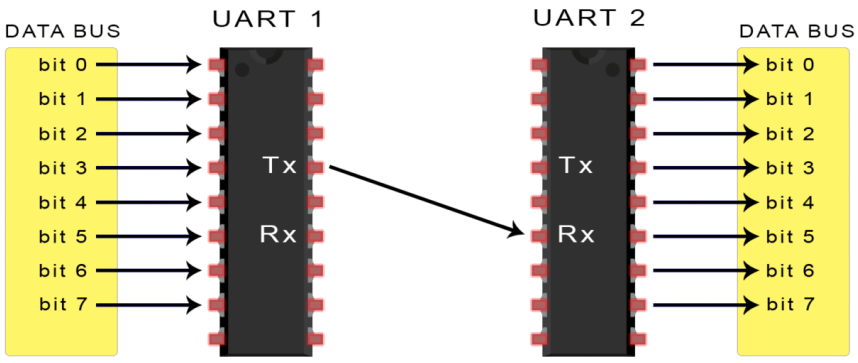

Figure 2.2: Working Principle

UART transmitted data is organized into packets. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional parity bit, and 1 or 2 stop bits

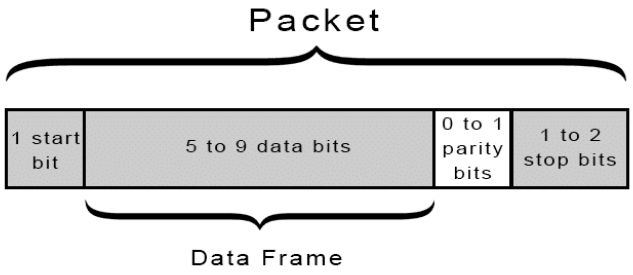

Figure 2.3: Structure of the transmitted data

Start Bit:

The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one clock cycle. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.

Data Frame:

The data frame contains the actual data being transferred. It can be 5 bits up to 8 bits long if a parity bit is used. If no parity bit is used, the data frame can be 9 bits long. In most cases, the data is sent with the least significant bit first.

Parity:

Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long distance data transfers.

After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1 bits in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bits in the data frame should total to an odd number.

When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd; or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

Stop Bits:

To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for at least two bit durations.

2.1.3 Steps of UART Data Transmission

1. The transmitting UART receives data in parallel from the data bus:

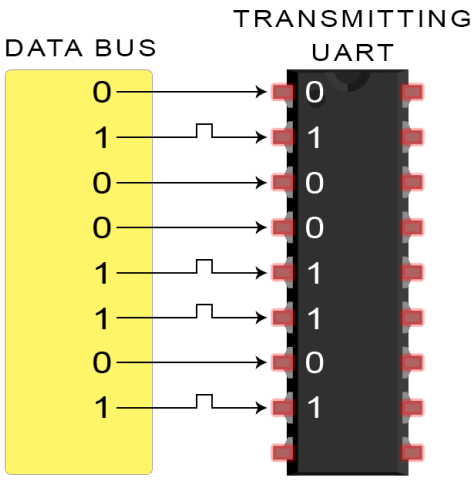

Figure 2.4: Step 1 - UART Data Transmission

2. The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame:

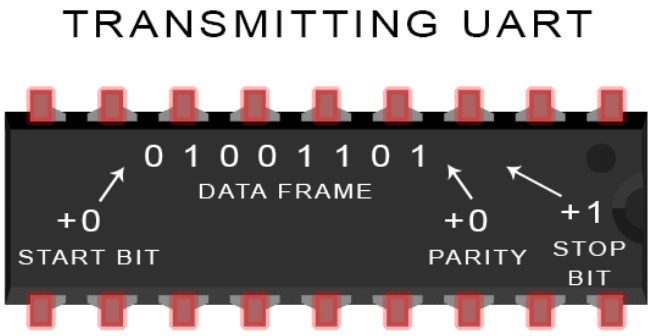

Figure 2.5: Step 2 - UART Data Transmission

3. The transmitting UART to the receiving UART. The receiving UART samples the data line at the pre-configured baud rate:

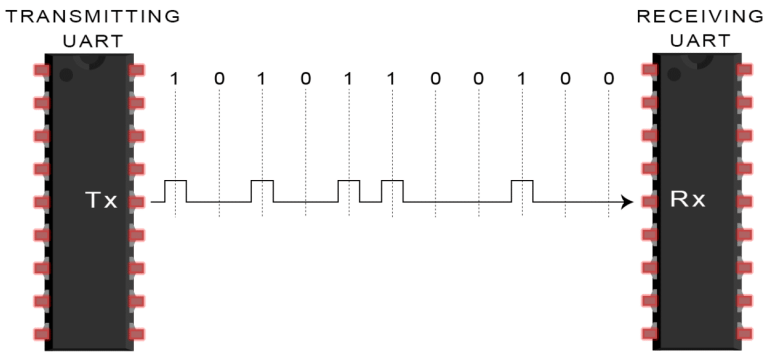

Figure 2.6: Step 3 - UART Data Transmission

4. The receiving UART discards the start bit, parity bit, and stop bit from the data frame:

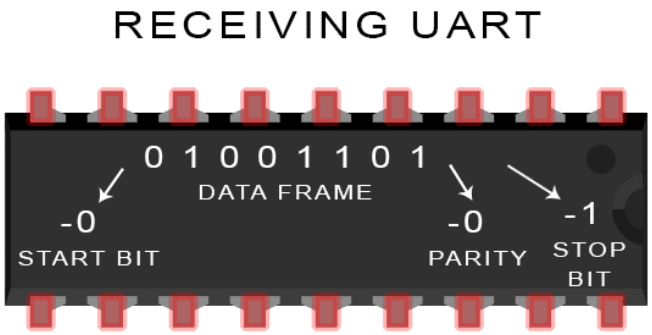

Figure 2.7: Step 4 - UART Data Transmission

5. The receiving UART converts the serial data back into parallel and transfers it to the data bus on the receiving end:

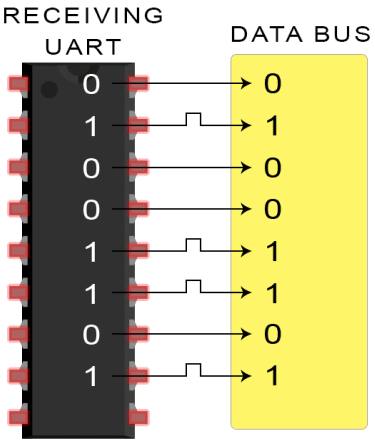

Figure 2.8: Step 5 - UART Data Transmission

2.1.4 Advantages and Disadvantages of UART

Advantages:

- Only uses two wires, no clock signal is required
- A parity bit is available to allow for error checking
- The structure of the data packet can be changed as long as both sides are set up for it
- Well documented and widely used method

Disadvantages:

- The size of the data frame is limited to a maximum of 9 bits
- Doesn't support multiple slave or multiple master systems
- The baud rates of each UART must be within 10% of each other

2.2 I²C - Inter Interconnect Communication

2.2.1 Introduction of I²C Communication Protocol

I²C is termed as the abbreviated form of Inter-Integrated Circuit. This is a type of communication bus which is mainly designed and developed to establish inter-chip communication. This protocol is a bus interface connection that is embedded into multiple devices to set up serial connections.

I²C is considered to be the most prominent chip-to-chip communication protocol as it holds the features of both UART and SPI. With I²C, it is feasible to connect multiple slaves to a single master and we can have multiple masters controlling single, or multiple slaves. This is really useful when we want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

I²C only uses two wires to transmit data between devices and the transmit is synchronous so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.

Wires Used	2
Maximum Speed	Standard mode: 100 kbps Fast mode: 400 kbps High-speed mode: 3.4 Mbps Ultra-fast mode: 5Mbps
Synchronous or Asynchronous ?	Synchronous
Serial or Parallel ?	Serial
Maximum number of Masters	Unlimited
Maximum number of Slaves	1008

Table 2.2: I²C Communication Properties

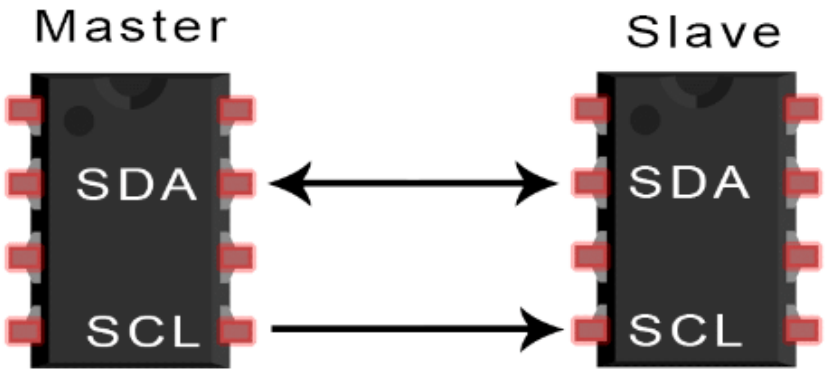

Figure 2.9: Master-Slave Communication in I²C

2.2.2 I²C Protocol Working Principle

The working of the I²C communication protocol happens through open drain lines which are Serial Data (SDA) and SCL (Serial Clock). Initially, both the SDA and SCL lines are pulled high and the bus mainly functions in two modes which are Master and Slave. With I²C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:

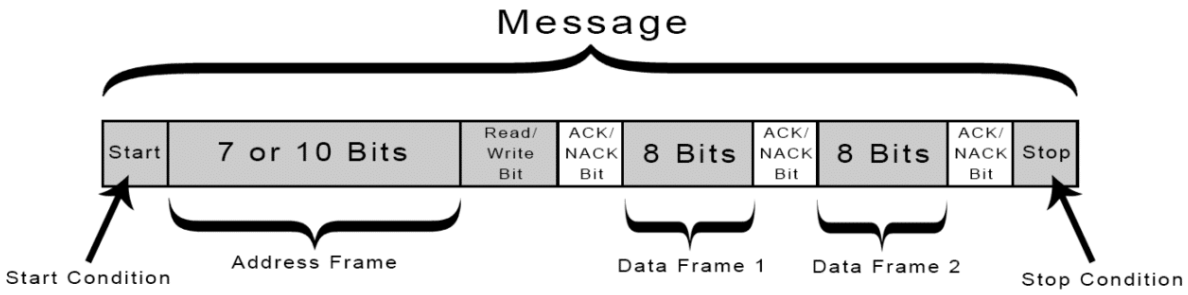

Figure 2.10: The data packet

Every bit which is transferred on the serial data line gets synchronized by HIGH to LOW pulse for every clock signal that is received on the serial clock. The SDA line will not change when the SCL is in a HIGH state and it is changed only when SCL is in a LOW state. As discussed SDA and SCL are open-drain lines, they require of pull-up resistor to make them high as because the devices on the bus are in the active LOW state.

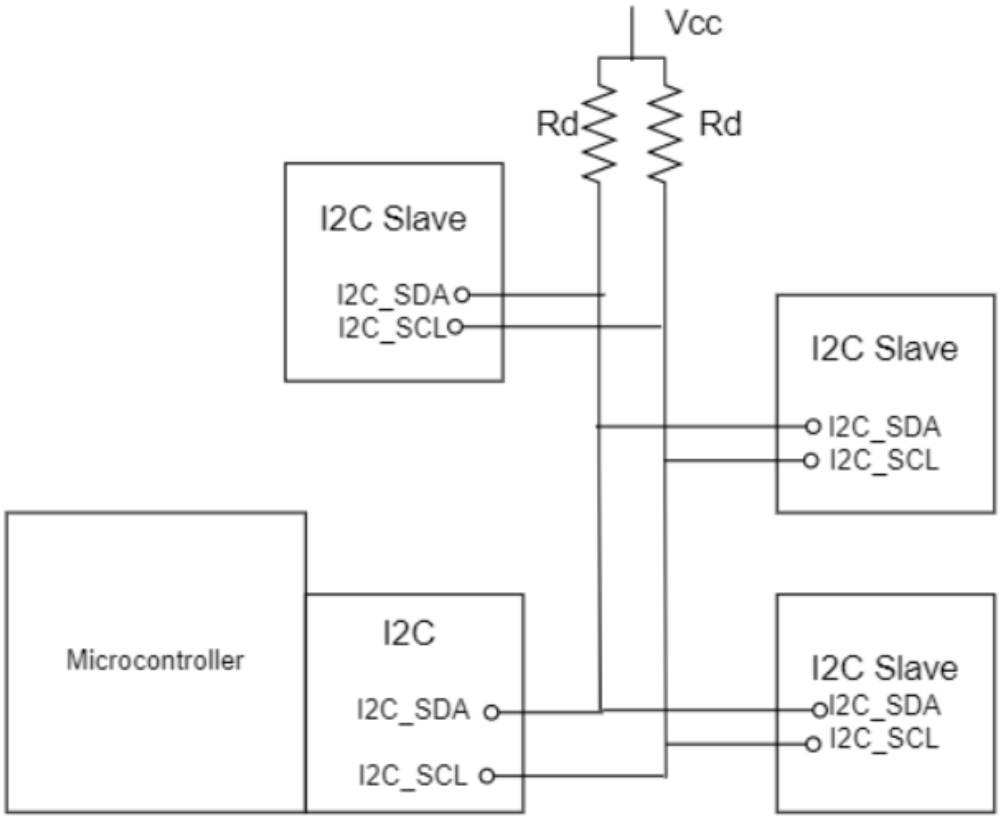

Figure 2.11: I²C Protocol Architecture

Here, data transfer takes place in the form of packets and each packet consists of 9 bits. The sequence of bits is shown below:

Start & Stop Bits:

For Start Condition, the SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low. Meanwhile, The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high for Stop Condition.

Address:

This is the immediate frame after the start bit. The master bit transfers the slave's address with which it has to communicate to each slave that was connected to it. Then the slave bit compares its own address with the address of the slave that was sent by the master. When both the addresses match, it transmits a low voltage ACK signal to the master signal. Whereas when both the addresses do not match, the slave remains idle and the serial data line stays at HIGH.

Read/Write Bit:

When the Read/Write indicates '1', then the master is transmitting data to the slave whereas when Read/Write indicates '0', then the master is receiving data from the slave signal.

ACK/NACK:

The acknowledge/no-acknowledge bit is the subsequent bit of every frame in a message. When data/address was successfully transmitted, then the ACK signal gets back to the sender from the receiver device.

Data Frame:

When the ACK bit is detected by master from slave it indicates that the first data frame can be transferred. This data frame is of 8-bit length. After the data frame, ACK/NACK bit detects the successful transmission of the frame. Once the transmission was successful, then the next data frame will be ready to transmit.

Once all the data frames are received, the master sends the STOP signal indicating to stop the transmission. The below I²C protocol timing diagram gives a clear idea of the working of the protocol.

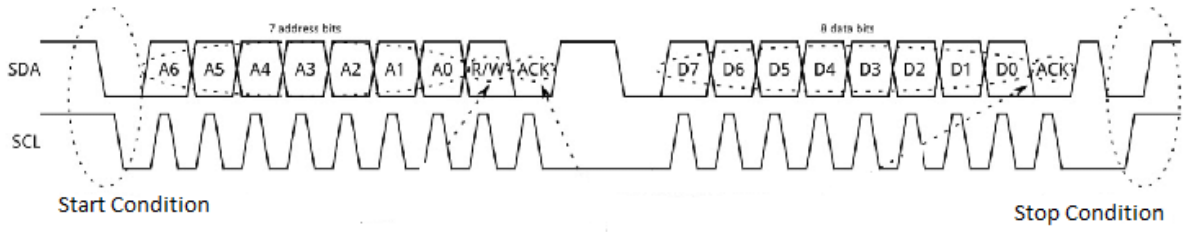

Figure 2.12: Timing Diagram

2.2.3 Steps of I²C Data Transmission

1. The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level before switching the SCL line from high to low:

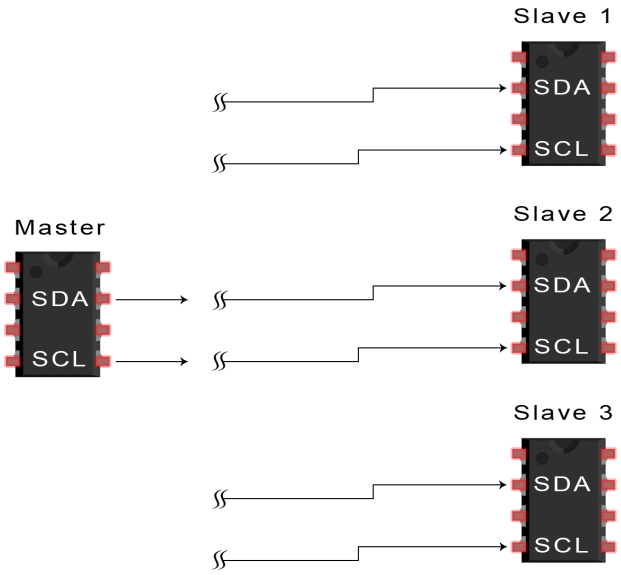

Figure 2.13: Step 1 - I²C Data Transmission

2. The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit:

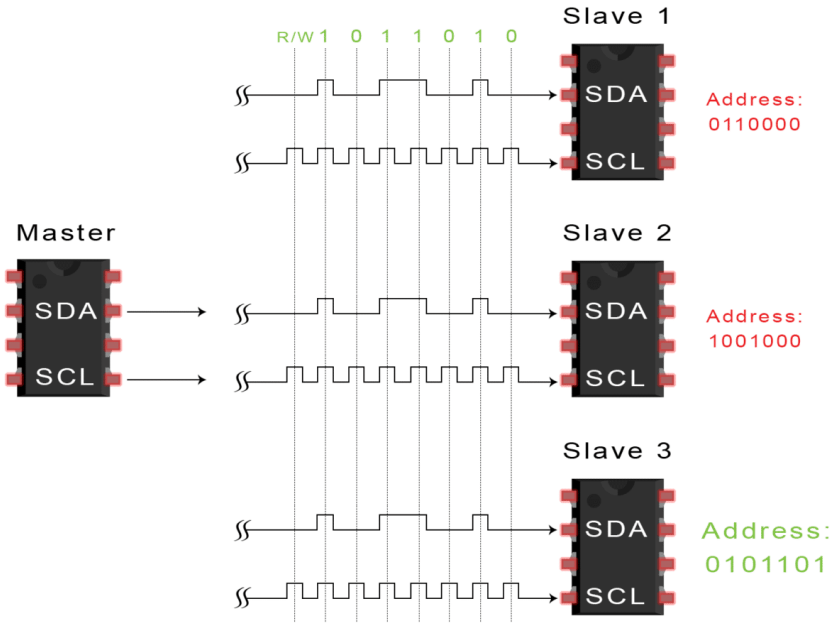

Figure 2.14: Step 2 - I²C Data Transmission

3. Each slave compares the address sent from the master to its own address. If the address matches, the slave returns an ACK bit by pulling the SDA line low for one bit. If the address from the master does not match the slave's own address, the slave leaves the SDA line high.

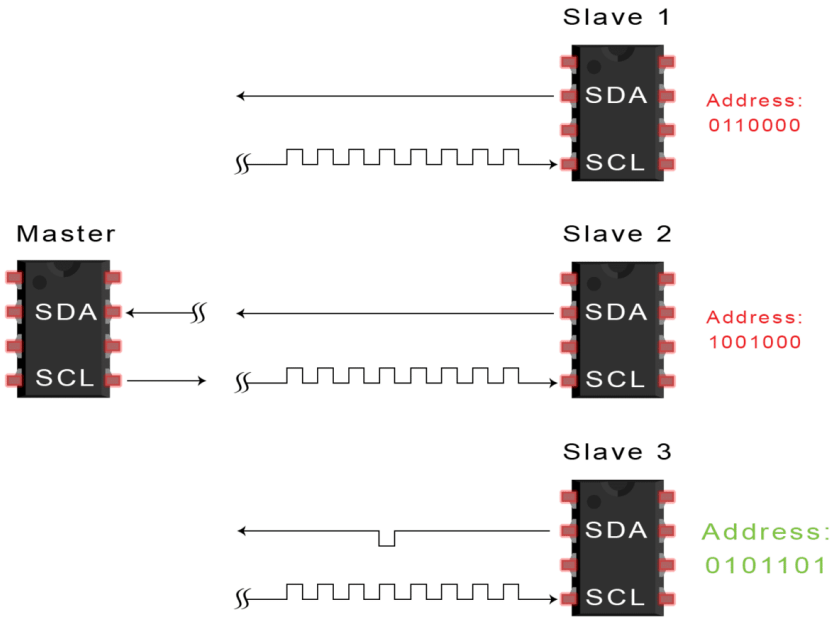

Figure 2.15: Step 3 - I²C Data Transmission

4. The master sends or receives the data frame:

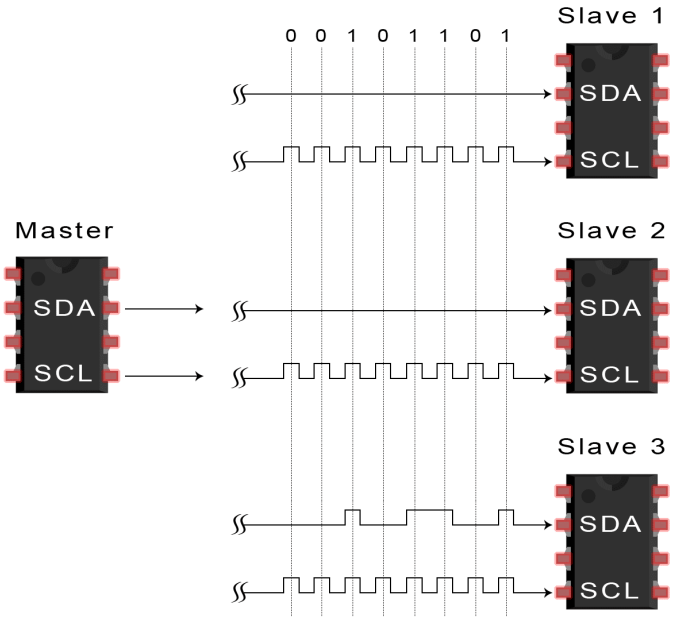

Figure 2.16: Step 4 - I²C Data Transmission

5. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame:

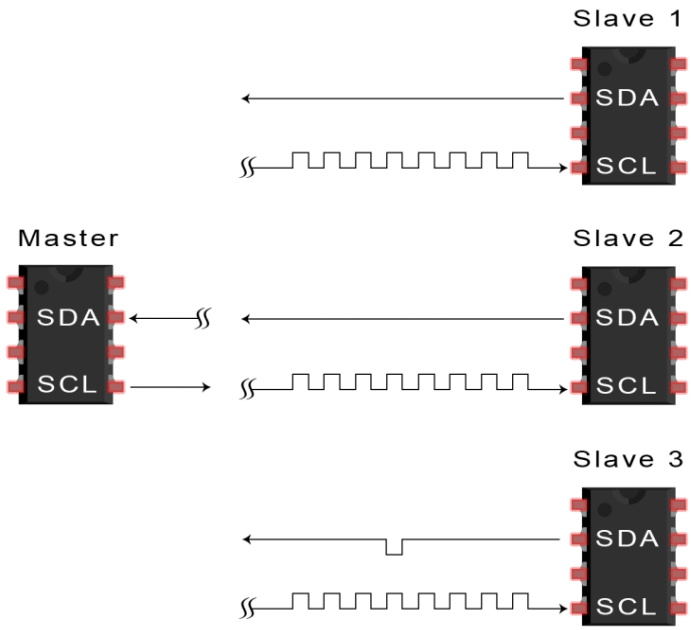

Figure 2.17: Step 5 - I²C Data Transmission

6. To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high:

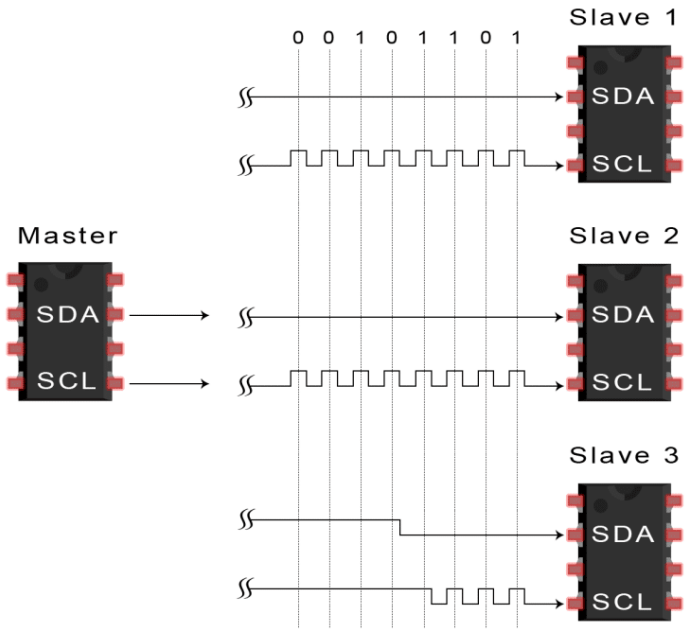

Figure 2.18: Step 6 - I²C Data Transmission

2.2.4 Communication Schemes

Single Master with Multiple Slaves

Because I²C uses addressing, multiple slaves can be controlled from a single master. With a 7 bit address, 128 (2^7) unique address are available. Using 10 bit addresses is uncommon, but provides 1,024 (2^{10}) unique addresses. To connect multiple slaves to a single master, wire them like this, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to VCC:

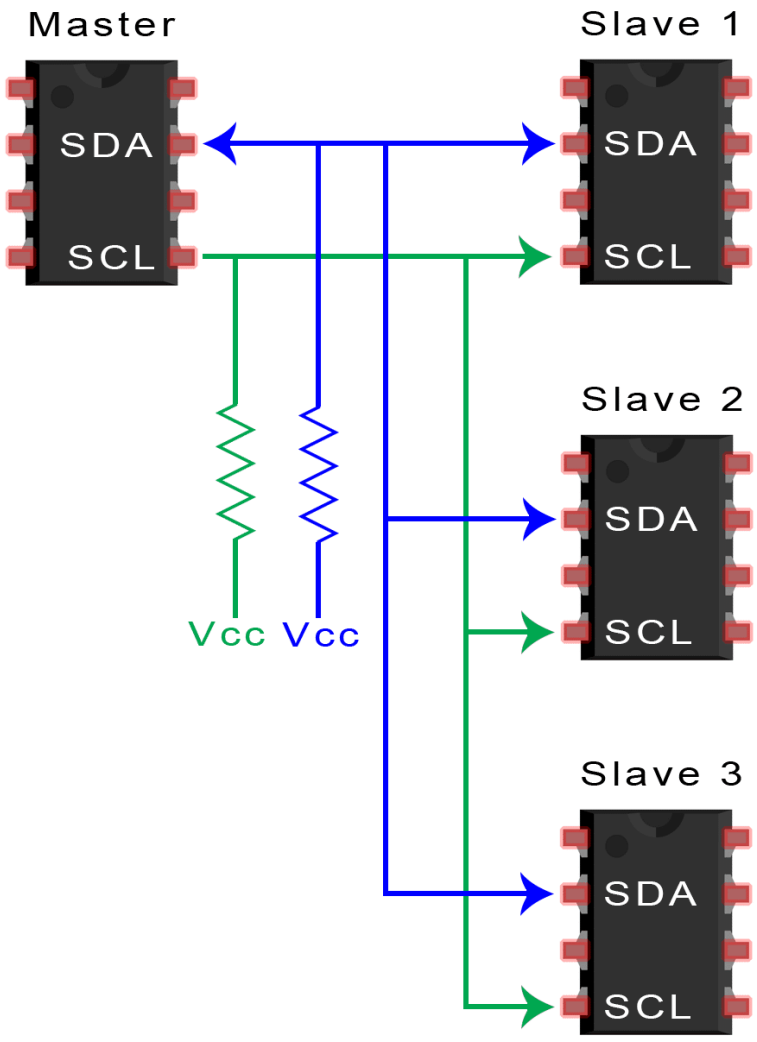

Figure 2.19: Single Master - Multiple Slaves

Multiple Masters with Multiple Slaves

Multiple masters can be connected to a single slave or multiple slaves. The problem with multiple masters in the same system comes when two masters try to send or receive data at the same time over the SDA line.

To solve this problem, each master needs to detect if the SDA line is low or high before transmitting a message. If the SDA line is low, this means that another master has control of the bus, and the master should wait to send the message. If the SDA line is high, then it's safe to transmit the message. To connect multiple masters to multiple slaves, use the following diagram, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to VCC:

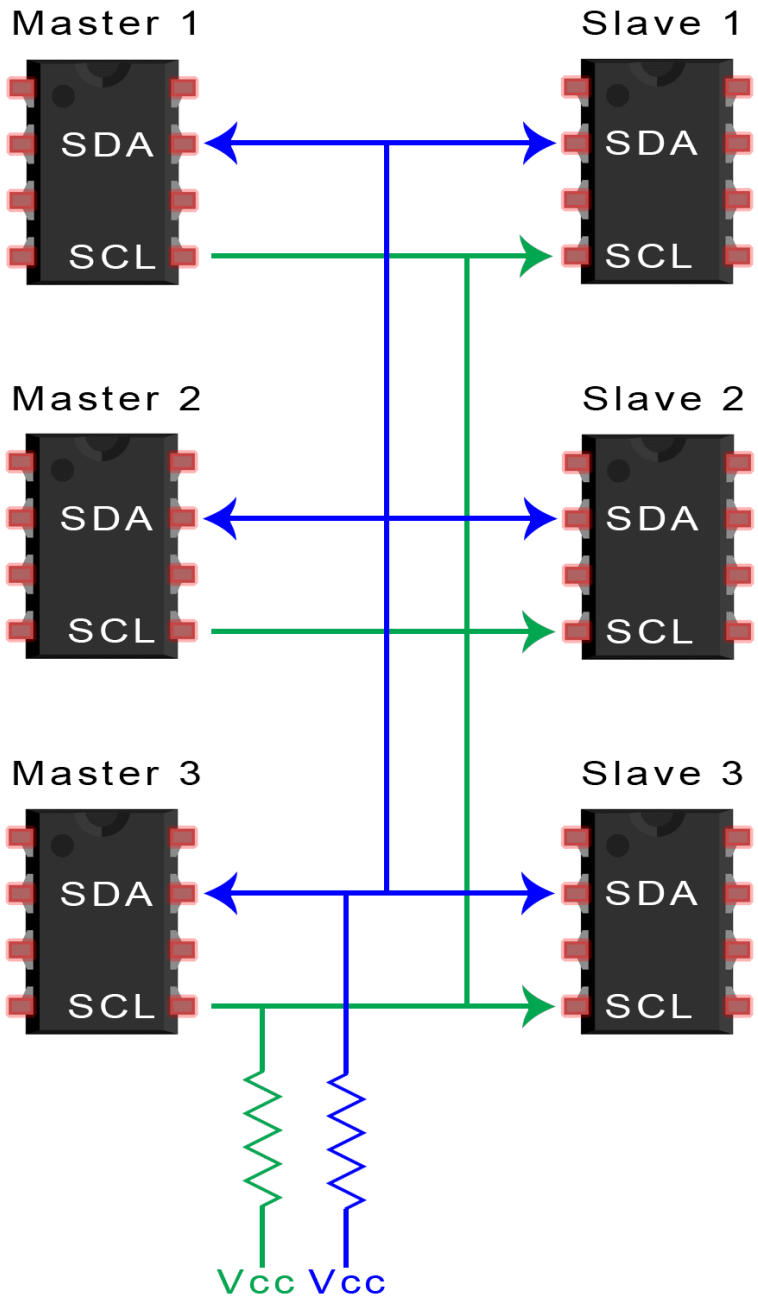

Figure 2.20: Multiple Masters - Multiple Slaves

2.2.5 Advantages and Disadvantages of I²C

Advantages:

- Even though there are multiple devices on the bus, the signals required are less
- It has the ability to support many masters
- It has enhanced adaptability where the requirements of various slave devices can be achieved
- It has multiple-master and multiple-slave flexibility

Disadvantages:

- I²C protocol is designed with an open-drain configuration where it limits the speed and also pull-up resistors are used which further lessens the communication speed
- The size of the data frame is limited to 8 bits

CHAPTER 3

DEVICES AND COMPONENTS

The main content of the chapter is to provide a brief overview of the STM32 Nucleo-64 development board. Beside, the working mechanism and functionality of each component used in the project are also presented.

3.1 Introduction of STM32 Nucleo-64 development board with STM32-F103RB MCU

The STM32 Nucleo-64 board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features, provided by the STM32 microcontroller. For the compatible boards, the external SMPS significantly reduces power consumption in Run mode.

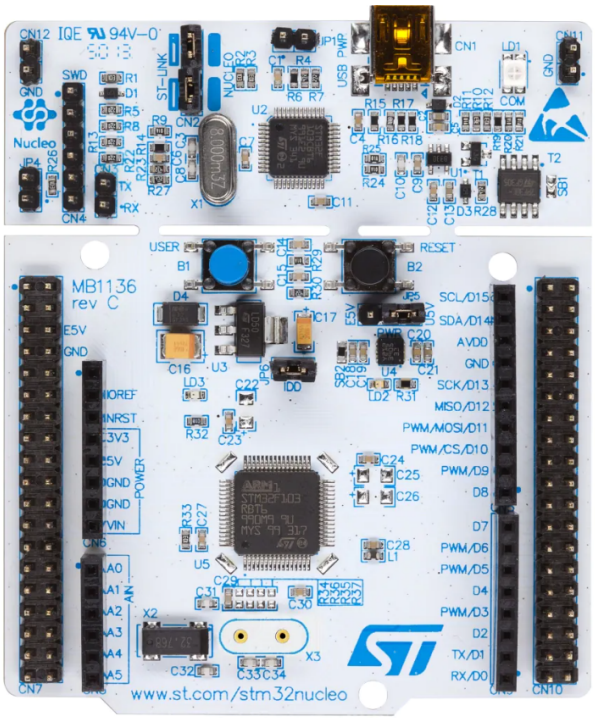

Figure 3.1: STM32 Necleo

The ARDUINO Uno V3 connectivity support and the ST morpho headers allow the easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields.

The STM32 Nucleo-64 board does not require any separate probe as it integrates the ST-LINK debugger/programmer.

The STM32 Nucleo-64 board comes with the STM32 comprehensive free software libraries and examples available with the STM32Cube MCU Package.

Here are some common features of the board:

- STM32 microcontroller in LQFP64 or LQFP48 package

- 1 user LED shared with ARDUINO
- 1 user and 1 reset push-buttons
- 32.768 kHz crystal oscillator
- Board connectors: ARDUINO Uno V3 expansion connector and ST morpho extension pin headers for full access to all STM32 I/Os
- Flexible power-supply options: ST-LINK USB VBUS or external sources
- On-board ST-LINK debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port, and debug port
- Comprehensive free software libraries and examples available with the STM32Cube MCU Package
- Support of a wide choice of Integrated Development Environments (IDEs) including IAR Embedded Workbench, MDK-ARM, and STM32CubeIDE

and also board-specific features:

- External SMPS to generate V_{core} logic supply
- 24MHz or 48MHz HSE
- Board connectors:
  - External SMPS experimentation dedicated connector
  - Micro-B or Mini-B USB connector for the ST-LINK
  - MIPI debug connector

3.2 PCF8574/74A

3.2.1 Description

The PCF8574/74A provides general-purpose remote I/O expansion via the two-wire bidirectional I^2C-bus (serial clock (SCL), serial data (SDA)), with operating supply voltage 2.5 V to 6 V with non-overvoltage tolerant I/O held to VDD with 100 A current source.

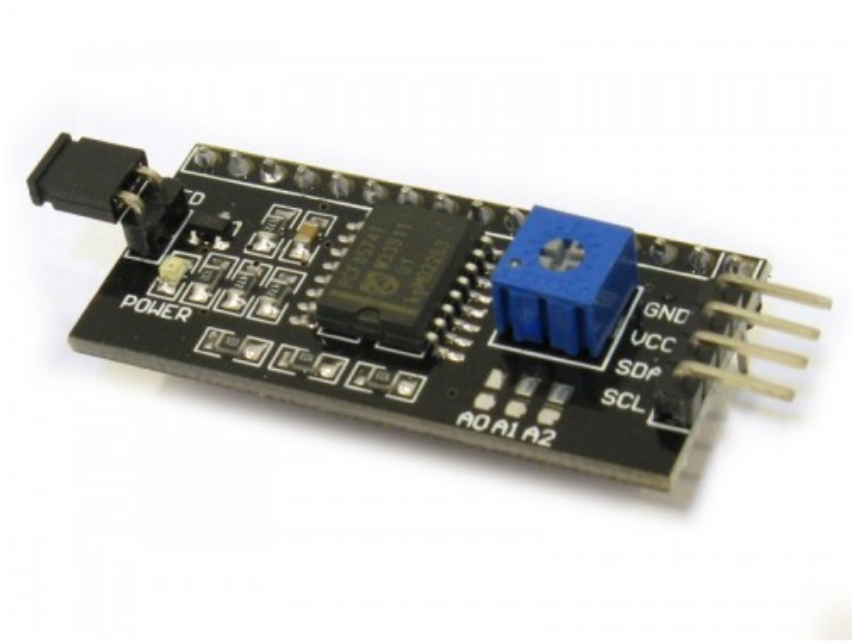

Figure 3.2: PCF8574/74A

The PCF8574 and PCF8574A are identical, except for the different fixed portion of the slave address. The three hardware address pins allow eight of each device to be on the same I2C-bus, so there can be up to 16 of these I/O expanders PCF8574/74A together on the same I2C-bus, supporting up to 128 I/Os (for example, 128 LEDs).

3.2.2 Block Diagram

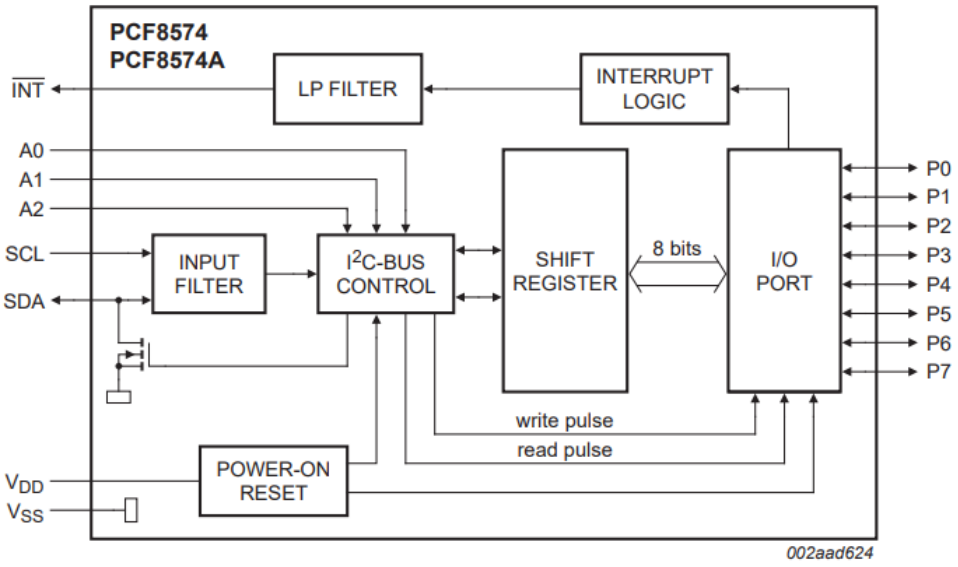

Figure 3.3: PCF8574/74A Block Diagram

3.2.3 Usage

Write to the port (Output Mode)

The master (microcontroller) sends the START condition and slave address setting the last bit of the address byte to logic 0 for the write mode. The PCF8574/74A acknowledges and the master then sends the data byte for P7 to P0 to the port register.

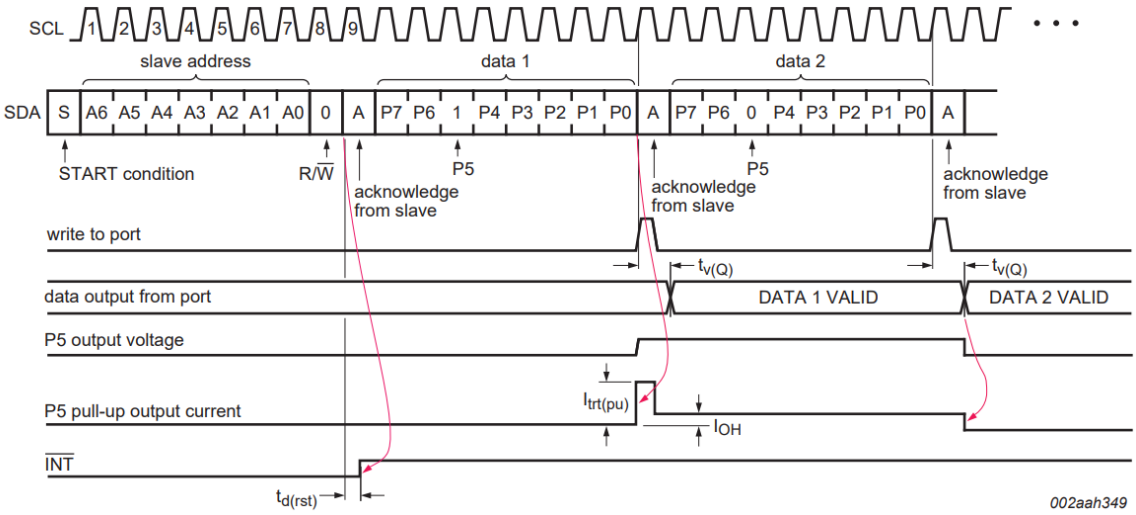

Figure 3.4: PCF8574/74A Write Mode

Simple code **WRITE** mode:

<S> <slave address + write> <ACK> <data out> <ACK> <data out> <ACK> ... <data out> <ACK> <P>

Reading from a port (Input mode)

The port must have been previously written to logic 1, which is the condition after power-on reset. To enter the Read mode the master (microcontroller) addresses the slave device and sets the last bit of the address byte to logic 1 (address byte read). The slave will acknowledge and then send the data byte to the master. The master will NACK and then send the STOP condition or ACK and read the input register again.

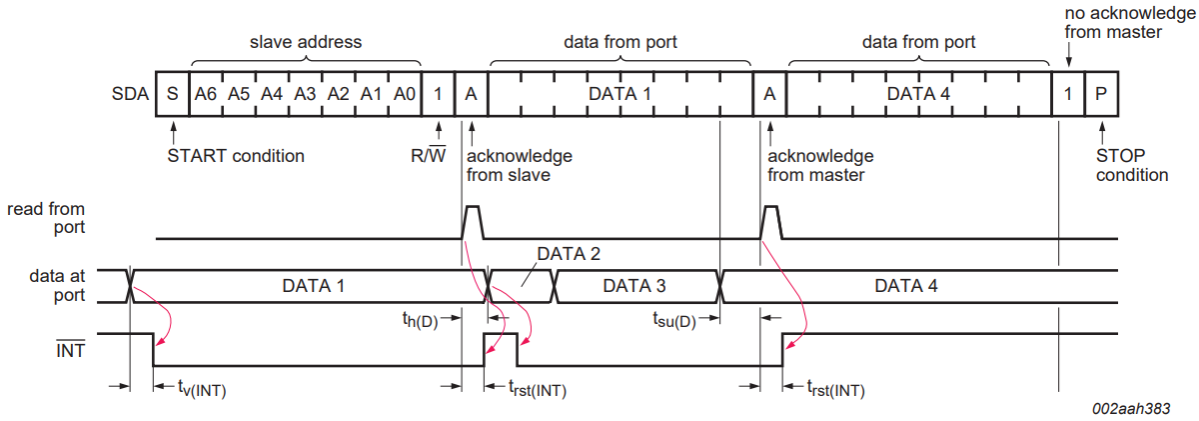

Figure 3.5: PCF8574/74A Read mode

Simple code for **READ** mode:
<S> <slave address + read> <ACK> <data in> <ACK> ... <data in> <ACK> <data in> <NACK> <P>

I²C Address

The PCF8574 and PCF8574A are functionally the same, but have a different fixed portion (A6 to A3) of the slave address. This allows 8 of the PCF8574 and 8 of the PCF8574A to be on the same I2C-bus without address conflict. In normally, we can interface with 16 LCDs at the same time.
(We can also modify the fixed portion (A6 to A3) to extend communication up to 128 LCDs).

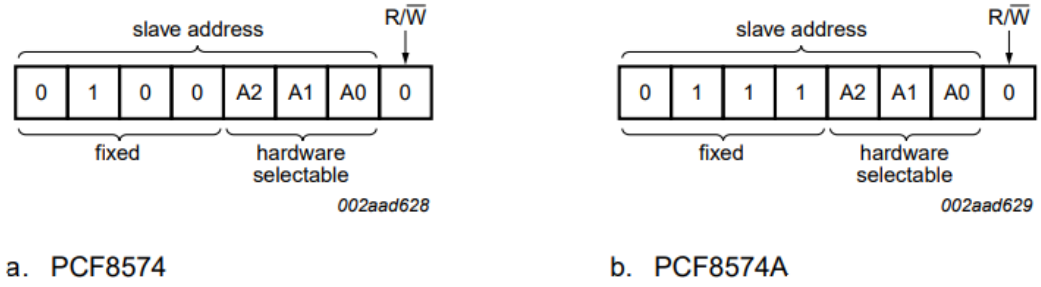

Figure 3.6: PCF8574/74A address

I²C Address Map

The I²C address range for PCF8574 are 0x20 to 0x27, the default value in the retail is 0x27 [A2,A1,A0] = [1,1,1.]
Because there have a R\W bit, so we need to shift left the address by 1 when communicating.

Pin connectivity			Address of PCF8574								Address byte value		7-bit hexadecimal address without R/W
A2	A1	A0	A6	A5	A4	A3	A2	A1	A0	R/W	Write	Read	
V_{SS}	V_{SS}	V_{SS}	0	1	0	0	0	0	0	-	40h	41h	20h
V_{SS}	V_{SS}	V_{DD}	0	1	0	0	0	0	1	-	42h	43h	21h
V_{SS}	V_{DD}	V_{SS}	0	1	0	0	0	1	0	-	44h	45h	22h
V_{SS}	V_{DD}	V_{DD}	0	1	0	0	0	1	1	-	46h	47h	23h
V_{DD}	V_{SS}	V_{SS}	0	1	0	0	1	0	0	-	48h	49h	24h
V_{DD}	V_{SS}	V_{DD}	0	1	0	0	1	0	1	-	4Ah	4Bh	25h
V_{DD}	V_{DD}	V_{SS}	0	1	0	0	1	1	0	-	4Ch	4Dh	26h
V_{DD}	V_{DD}	V_{DD}	0	1	0	0	1	1	1	-	4Eh	4Fh	27h

Figure 3.7: PCF8574 Address Range

The I²C address range for PCF8574A are 0x38 to 0x3F, the default value in the retail is 0x3F [A2,A1,A0] = [1,1,1].

Pin connectivity			Address of PCF8574A								Address byte value		7-bit hexadecimal address without R/W
A2	A1	A0	A6	A5	A4	A3	A2	A1	A0	R/W	Write	Read	
V_{SS}	V_{SS}	V_{SS}	0	1	1	1	0	0	0	-	70h	71h	38h
V_{SS}	V_{SS}	V_{DD}	0	1	1	1	0	0	1	-	72h	73h	39h
V_{SS}	V_{DD}	V_{SS}	0	1	1	1	0	1	0	-	74h	75h	3Ah
V_{SS}	V_{DD}	V_{DD}	0	1	1	1	0	1	1	-	76h	77h	3Bh
V_{DD}	V_{SS}	V_{SS}	0	1	1	1	1	0	0	-	78h	79h	3Ch
V_{DD}	V_{SS}	V_{DD}	0	1	1	1	1	0	1	-	7Ah	7Bh	3Dh
V_{DD}	V_{DD}	V_{SS}	0	1	1	1	1	1	0	-	7Ch	7Dh	3Eh
V_{DD}	V_{DD}	V_{DD}	0	1	1	1	1	1	1	-	7Eh	7Fh	3Fh

Figure 3.8: PCF8574A Address Range

3.3 LCD 1602

3.3.1 Description

LCD1602 or (LCD16×2) is a device which can show up to 16 characters each row and each character can be built with a 5×8 pixel box. To perform the operation of the LCD we need to use the voltage from 0.3V-7.0V and 1mA current. There are 2 display modes which are 4-bit and 8-bit. Beside, these display can be obtained in Blue or Green Backlight and also it can display some customized characters.

3.3.2 Usage

PIN NO.	SYMBOL	DESCRIPTION	FUNCTION
1	VSS	GROUND	0V (GND)
2	VCC	POWER SUPPLY FOR LOGIC CIRCUIT	+5V
3	VEE	LCD CONTRAST ADJUSTMENT	
4	RS	INSTRUCTION/DATA REGISTER SELECTION	RS = 0 : INSTRUCTION REGISTER RS = 1 : DATA REGISTER
5	R/W	READ/WRITE SELECTION	R/W = 0 : REGISTER WRITE R/W = 1 : REGISTER READ
6	E	ENABLE SIGNAL	
7	DB0	DATA INPUT/OUTPUT LINES	8 BIT: DB0-DB7
8	DB1		
9	DB2		
10	DB3		
11	DB4		
12	DB5		
13	DB6		
14	DB7		
15	LED+	SUPPLY VOLTAGE FOR LED+	+5V
16	LED-	SUPPLY VOLTAGE FOR LED-	0V

Figure 3.9: Pin function of LCD 16×2

Clearer specification of each PINS:

- Pin 1 (Ground/Source Pin): This is a GND pin of display, used to connect the GND terminal of the microcontroller unit or power source.
- Pin 2 (VCC/Source Pin): This is the voltage supply pin of the display, used to connect the supply pin of the power source.

- Pin 3 (V0/VEE/Control Pin): This pin regulates the difference of the display, used to connect a changeable POT that can supply 0 to 5V.
- Pin 4 (Register Select/Control Pin): This pin toggles among command or data register, used to connect a microcontroller unit pin and obtains either 0 or 1(0 = data mode, and 1 = command mode).
- Pin 5 (Read/Write/Control Pin): This pin toggles the display among the read or writes operation, and it is connected to a microcontroller unit pin to get either 0 or 1 (0 = Write Operation, and 1 = Read Operation).
- Pin 6 (Enable/Control Pin): This pin should be held high to execute Read/Write process, and it is connected to the microcontroller unit & constantly held high.
- Pins 7-14 (Data Pins): These pins are used to send data to the display. These pins are connected in two-wire modes like 4-wire mode and 8-wire mode. In 4-wire mode, only four pins are connected to the microcontroller unit like 0 to 3, whereas in 8-wire mode, 8-pins are connected to microcontroller unit like 0 to 7.
- Pin 15 (LED+ pin of the LED): This pin is connected to +5V
- Pin 16 (LED- pin of the LED): This pin is connected to GND.

Figure 3.10: LCD-16×2 pins diagram

Instead of passing direct signal through the PINS of LCD 16×2 , we will now take the advantage of using PCF8574 to make LCD a I2C communication device

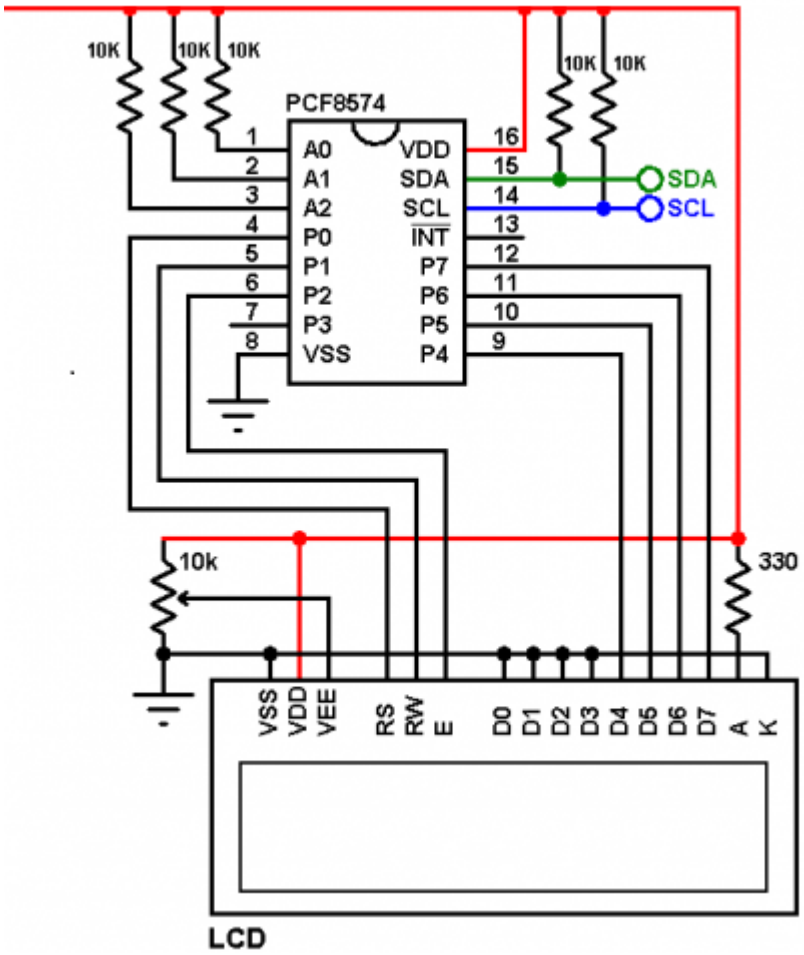

Figure 3.11: LCD-16×2 connecting to PCF8574

3.4 DHT20 Sensor

3.4.1 Description

The DHT20 is a kind of sensor of humidity and temperature with digital I^2C output. It can be applied to HVAC, dehumidifier, testing and inspection equipment, consumer products, automobiles, automatic control, data loggers, weather stations, home appliances, humidity control, medical and other application fields which need to detect and control temperature and humidity.

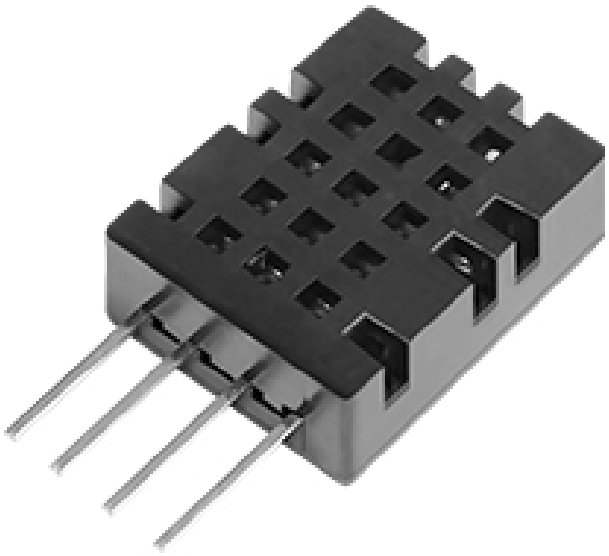

Figure 3.12: DHT20

3.4.2 Usage

Start the Sensor

The first step is to power up the sensor with the selected VDD supply voltage (range between 2.2V and 5.5V). After power-on, the sensor needs less than 100ms stabilization time (SCL is high at this time) to reach the idle state and it is ready to receive commands sent by the host (MCU).

Start/Stop Sequence

Each transmission sequence starts with the Start state and ends with the Stop state, as shown in **Fig.2.10** and **Fig. 2.12**

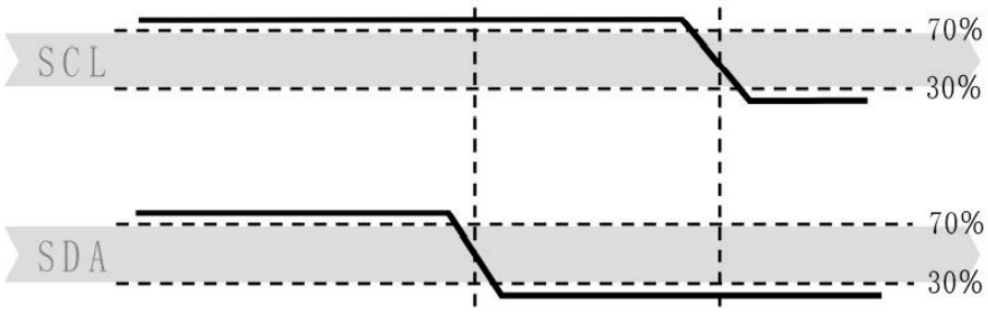

Figure 3.13: Start transmission status

When SCL is high, SDA is converted from high to low. The start state is a special bus state controlled by the master, indicating the start of the slave transfer (after Start, the BUS bus is generally considered to be in a busy state)

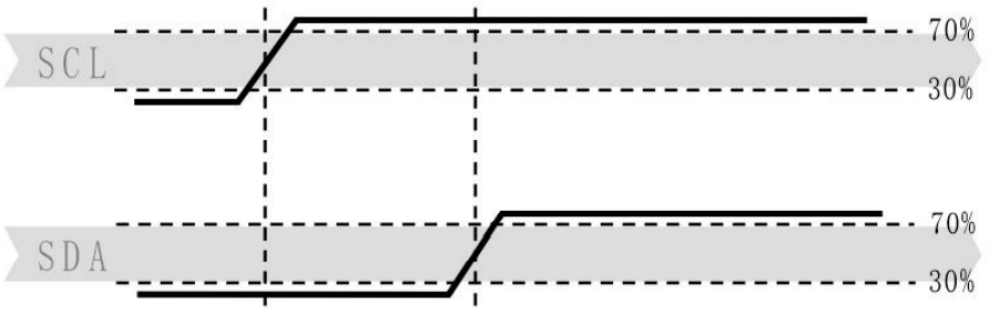

Figure 3.14: Stop transmission state

When SCL is high, the SDA line changes from low to high. The stop state is a special bus state controlled by the master, indicating the end of the slave transmission (after Stop, the BUS bus is generally considered to be in an idle state).

Send Command

After the transmission is started, the first byte of I^2C that is subsequently transmitted includes the 7-bit I^2C device address 0x38 and a SDA direction bit x (read R: '1', write W: '0'). After the 8^{th} falling edge of the SCL clock, pull down the SDA pin (ACK bit) to indicate that the sensor data is received normally. After sending the measurement command 0xAC, the MCU must wait until the measurement is completed.

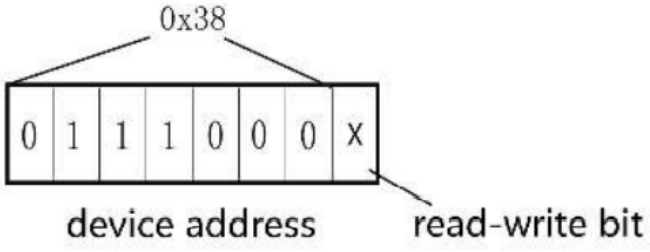

Figure 3.15: DHT20 address interface

Sensor Communication Process

1. After power-on, wait no less than 100ms. Before reading the temperature and humidity value, get a byte of status word by sending 0x71, if the status word and 0x18 are not equal to 0x18, initialize the 0x1B, 0x1C, 0x1E registers, by sending 3 bytes, ["Register", 0x00, 0x00] to DHT20; Then, receive reply 3 bytes from DHT20, Then, send the reply again but with (0xB0 or with "first reply byte").
2. Wait 10ms to send the 0xAC command (trigger measurement). This command parameter has two bytes, the first byte is 0x33, and the second byte is 0x00.

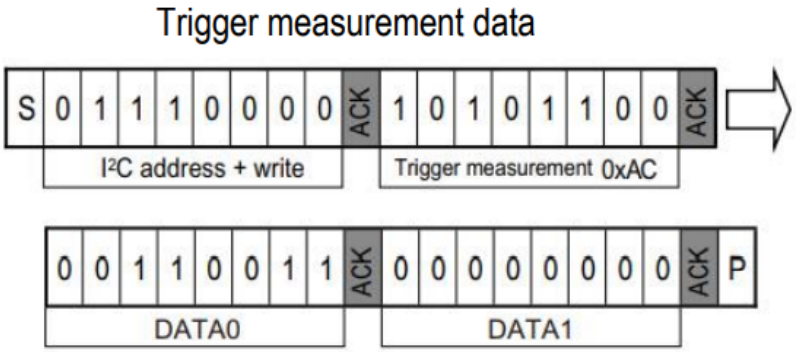

Figure 3.16: DHT20 trigger measurement interface

3. Wait 80ms for the measurement to be completed, if the read status word Bit [7] is 0, it means the measurement is completed, and then six bytes can be read continuously; otherwise, continue to wait.

Bits	Significance	Description
Bit[7]	Busy indication	1-Equipment is busy, in measurement mode 0- Equipment is idle, in hibernation state
Bit[6:5]	Retain	Retain
Bit[4]	Retain	Retain
Bit[3]	CAL Enable	1 - Calibrated 0 - Uncalibrated
Bit[2:0]	Retain	Retain

Figure 3.17: DHT20 status byte

4. First byte read indicate for the status of DHT20, the next 5 bytes regard to Humidity data and Temperature data, and the last byte use for error-detecting code CRC

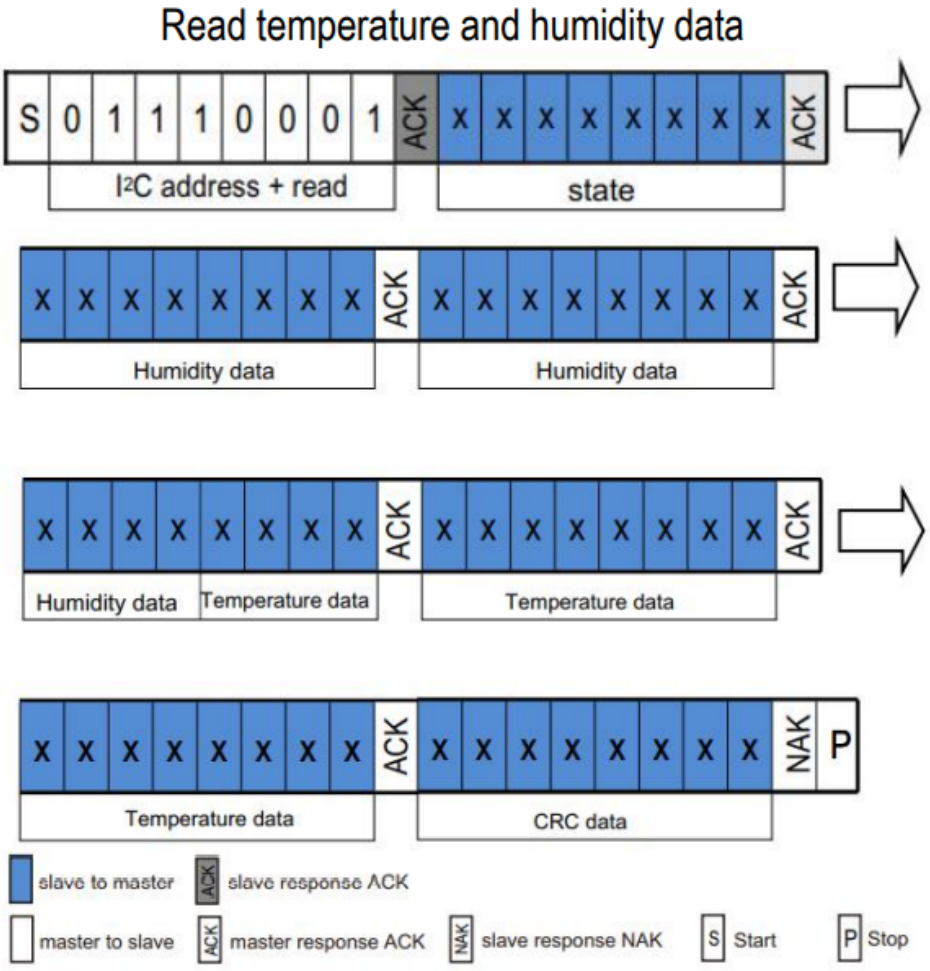

Figure 3.18: DHT20 reply value interface

5. After receiving six bytes, the next byte is the CRC (abbr. Cyclic redundancy check) check data. The user can read it out as needed. If the receiving end needs CRC check,an ACK will be sent after the sixth byte is received. Reply, otherwise send NACK to end,the initial value of CRC is 0XFF, and the CRC8 check polynomial is:

$$\mathrm{CRC}[7{:}0] = 1{+}\mathrm{X}^4{+}\mathrm{X}^5{+}\mathrm{X}^8$$

Signal conversion

Relative Humidity Conversion: The relative humidity RH can be calculated according to the relative humidity signal SRH output by SDA through the following formula (the result is expressed in %RH):

$$\mathrm{RH}[\%] = \frac{\mathrm{S_{RH}}}{2^{20}} \times 100\%$$

Temperature Conversion: The temperature T can be calculated by substituting the temperature output signal ST into the following formula: (The result is expressed in temperature ℃)

$$\mathrm{T}[^{\circ}\mathrm{C}] = \left(\frac{\mathrm{S_T}}{2^{20}} \times 200\right) - 50$$

CHAPTER 4

SYSTEM DESIGN

In this chapter, we propose the architecture of our system, introducing the hardware components used to implement it. Furthermore, we also provide an overview of the system with the block diagram. Finally, there will be a fully description for the connection protocol between each component in our system.

4.1 Proposal Architecture

In this project, we aim to build an extension shield to measure the real-time temperature ad humidity of the surrounding environment with the participation of DHT20 Sensor and LCD1602 for output display. Furthermore, the system also offer control of its operation by typing in command through a simulation tool from the host computer.

In detail, the system consists of three modes which are Waiting, Reading (Every 3 seconds) and Capturing. However, Waiting is just a temporary intermediate state between system boot and Reading mode. When power is plugged in, it will print out a greeting message and then immediately change to Reading mode.

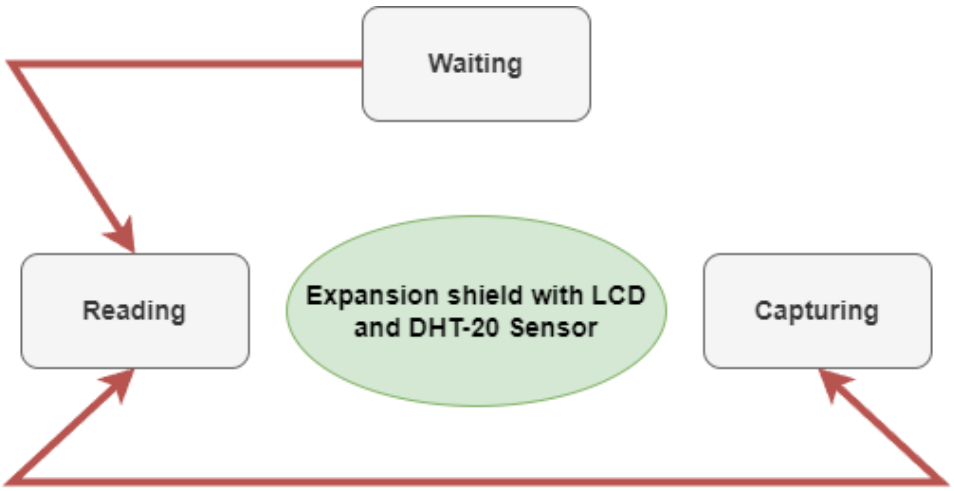

Figure 4.1: Proposal Architecture

In Reading mode, the LCD will display the value read from DHT-20 sensor every 3 seconds. Whenever a UART command in the form **!C#** (stands for Capture) is received, the LCD will show the value at that time and stop fetching new data.

Similarly, if the command **!R#** (stands for Reset) is encountered, the system continues reading again. Generally, the system can change between Reading and Capturing mode frequently.

4.2 Hardware Usage

As we have introduced above, the system will exploit the STM Nucleo board as the main hub for our expansion shield, which attachs the DHT-20 sensor, the LCD 16×2 and some supported electrical components to ensure the correct operation.

4.3 Detailed Description

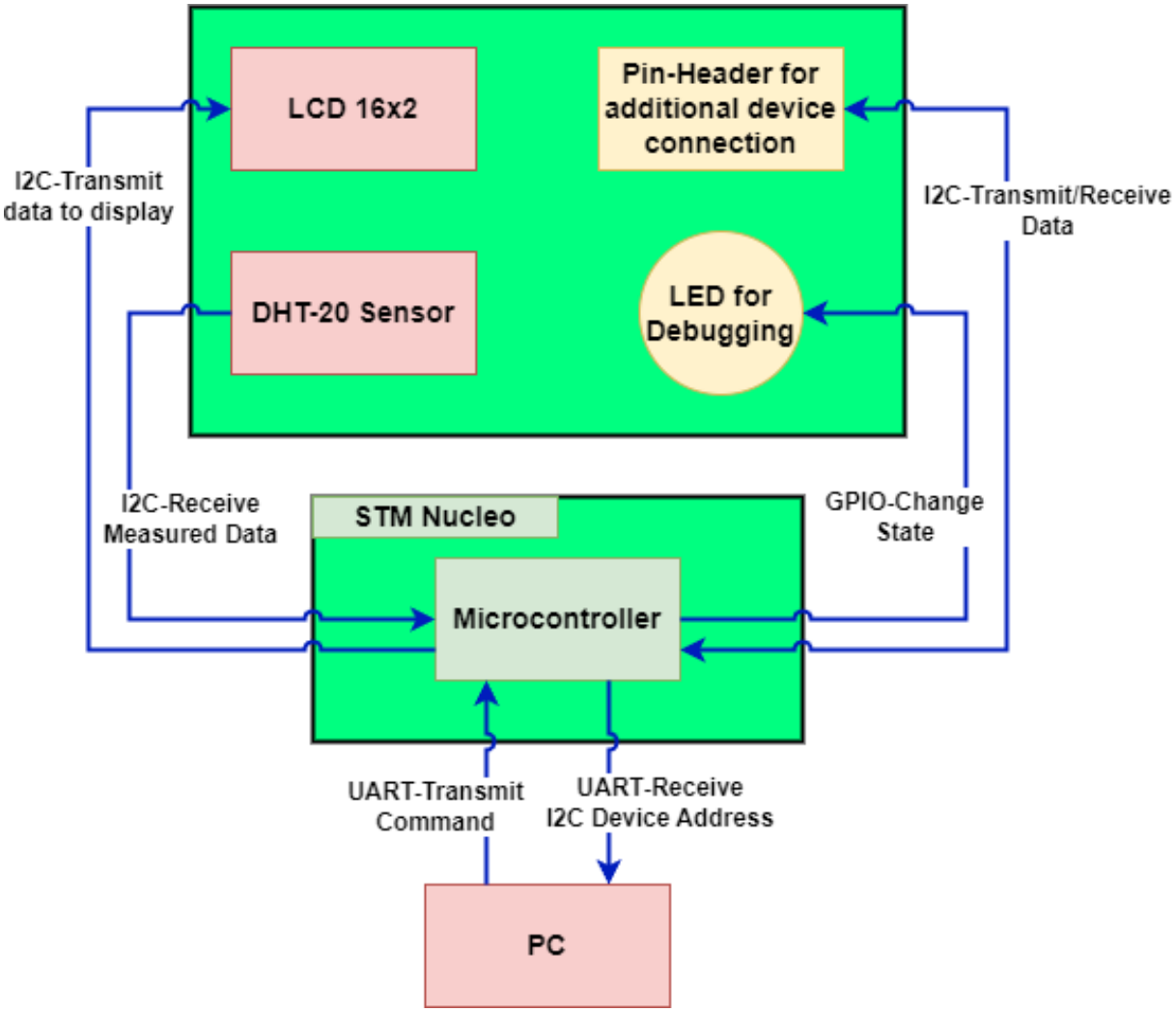

Figure 4.2: Block Diagram for the sytem

Firs of all, our two main components which are LCD 16x2 and DHT-20 sensor will connect with the MCU through I^2C protocol, but in different state Transmit and Receive, respectively. Specifically, the DHT20 acts as a slave device that will successively transmit the measured data back to the MCU. Then, MCU will intercept the data, do calculation to convert data into

comprehensible form which is decimal value in degree Celsius and percentage for temperature and humidity,respectively and make those value ready for output.

The other additional component such as the LED for debugging uses standard GPIO connection. Meanwhile, the Pin-Header links with the MCU by I^2C and provide either Transmit or Receive mode. This part is the extension connector that can attach a different I^2C device.

The bridge between PC and MCU is constructed under UART protocol with the assistance from Hercules or Hterm simulation tool. Then we are able to receive the address of I^2C device by from the scan function as well as sending command to the MCU.

CHAPTER 5

ALTIUM DESIGNER

Following up, we will demonstrate our work on design stage of the expansion shield in which we mainly interact with the Altium Designer application.

5.1 Schematic Design

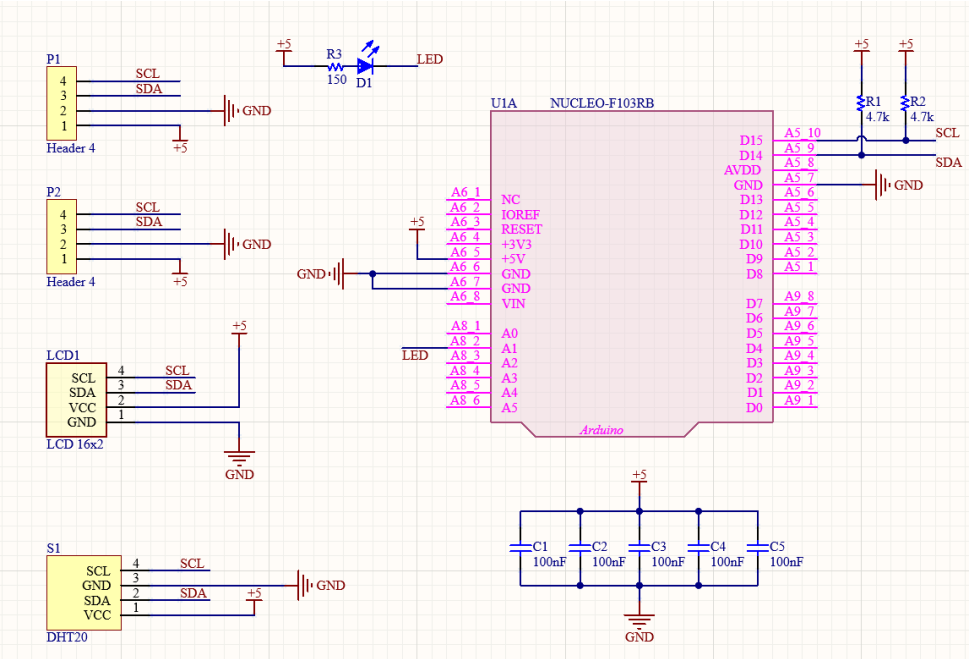

Figure 5.1: The Schematic Design

5.2 PCB Layout

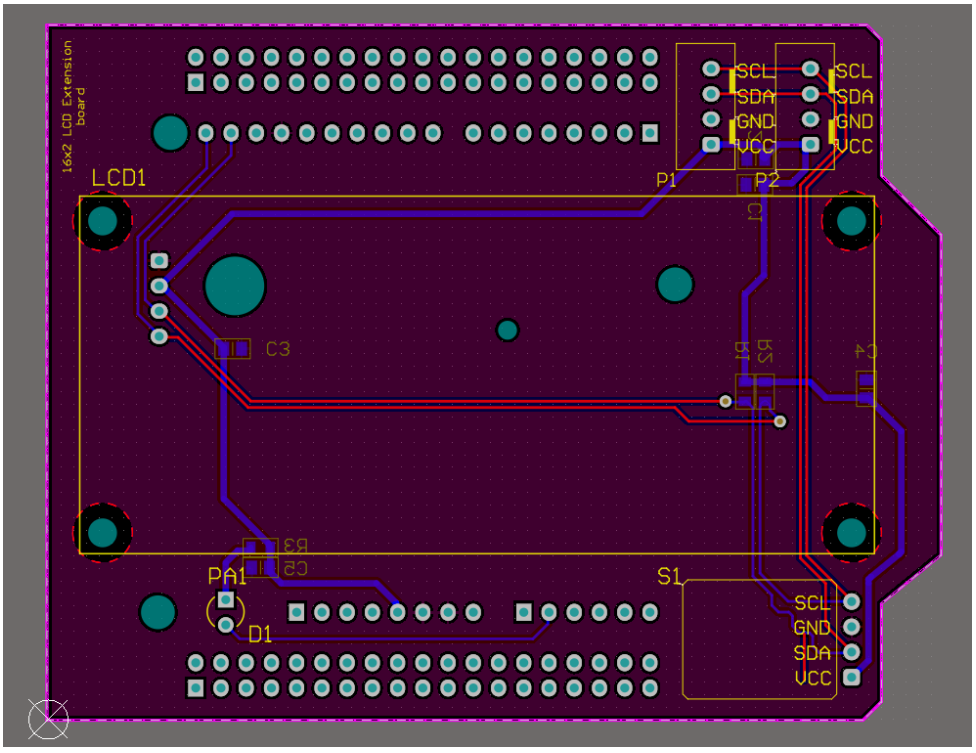

Figure 5.2: The PCB Layout

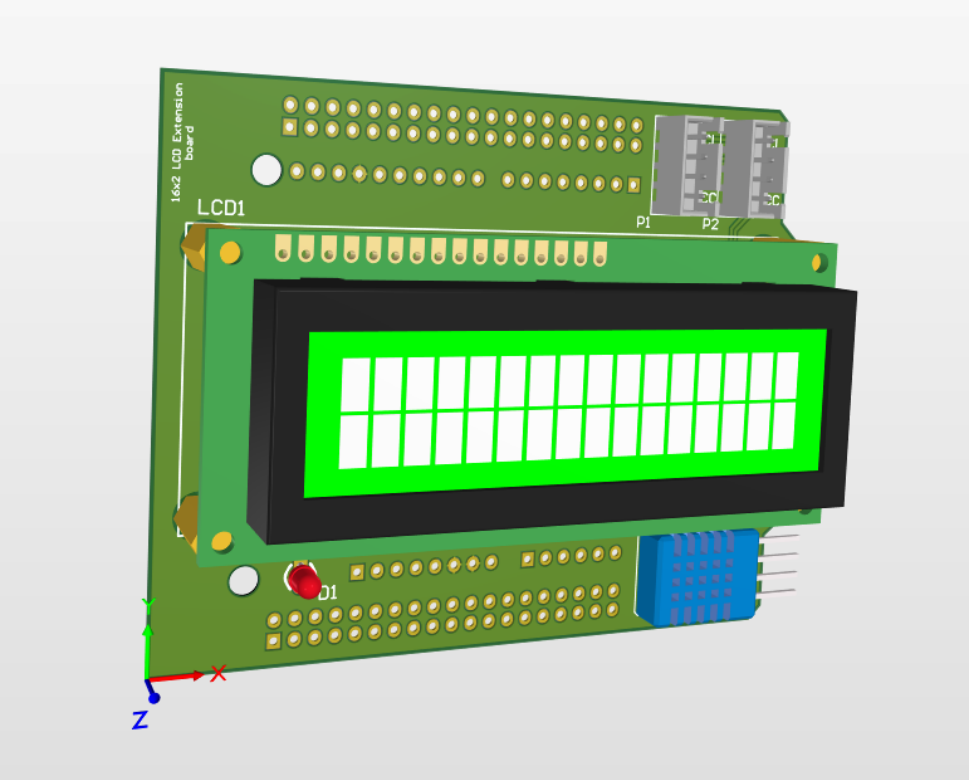

Figure 5.3: The module in 3D view

5.3 Actual View of Extension Board

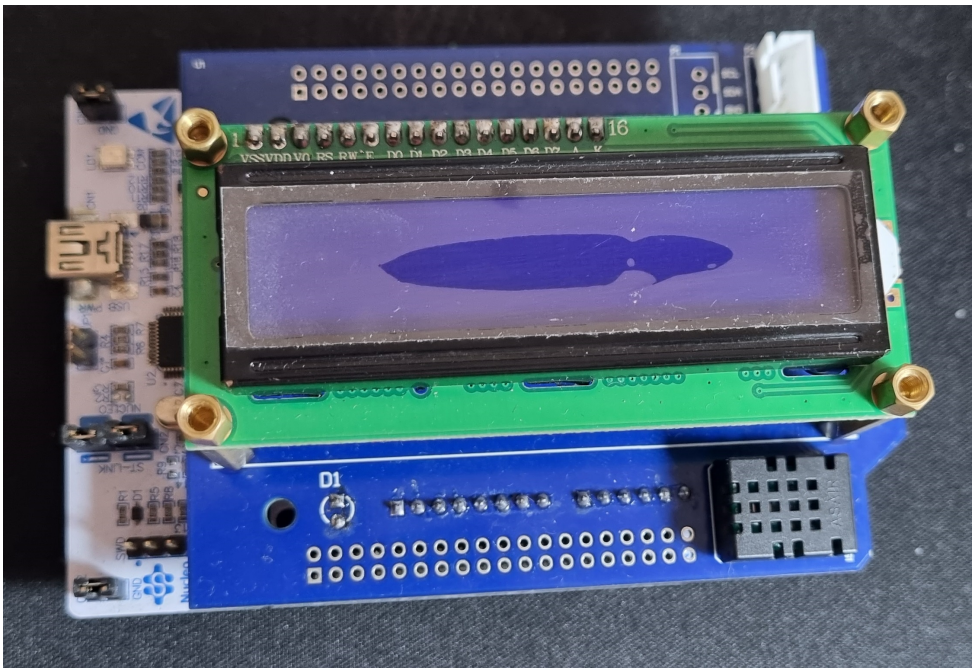

Figure 5.4: The module in actual top view

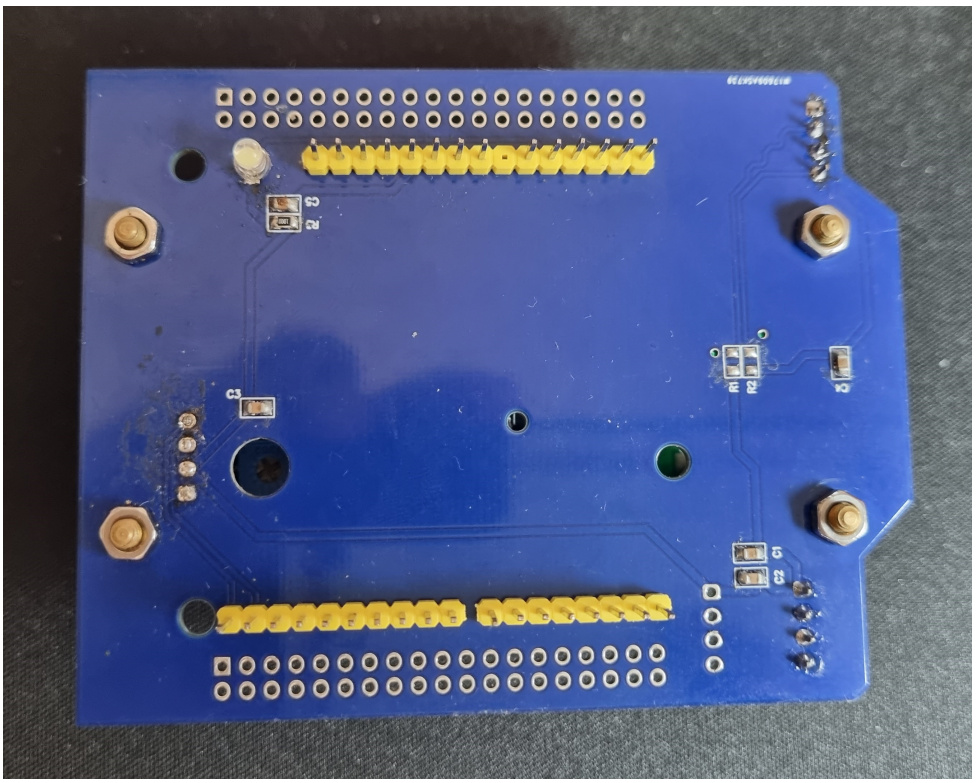

Figure 5.5: The module in actual bottom view

CHAPTER 6

PROGRAM DESIGN

6.1 Program Layout

The program can be divided into three group of function, which are Displaying information on the LCD, Scanning the device address & Interpreting the command and lastly, Reading value from the DHT-20 sensor & Combining to a complete system.

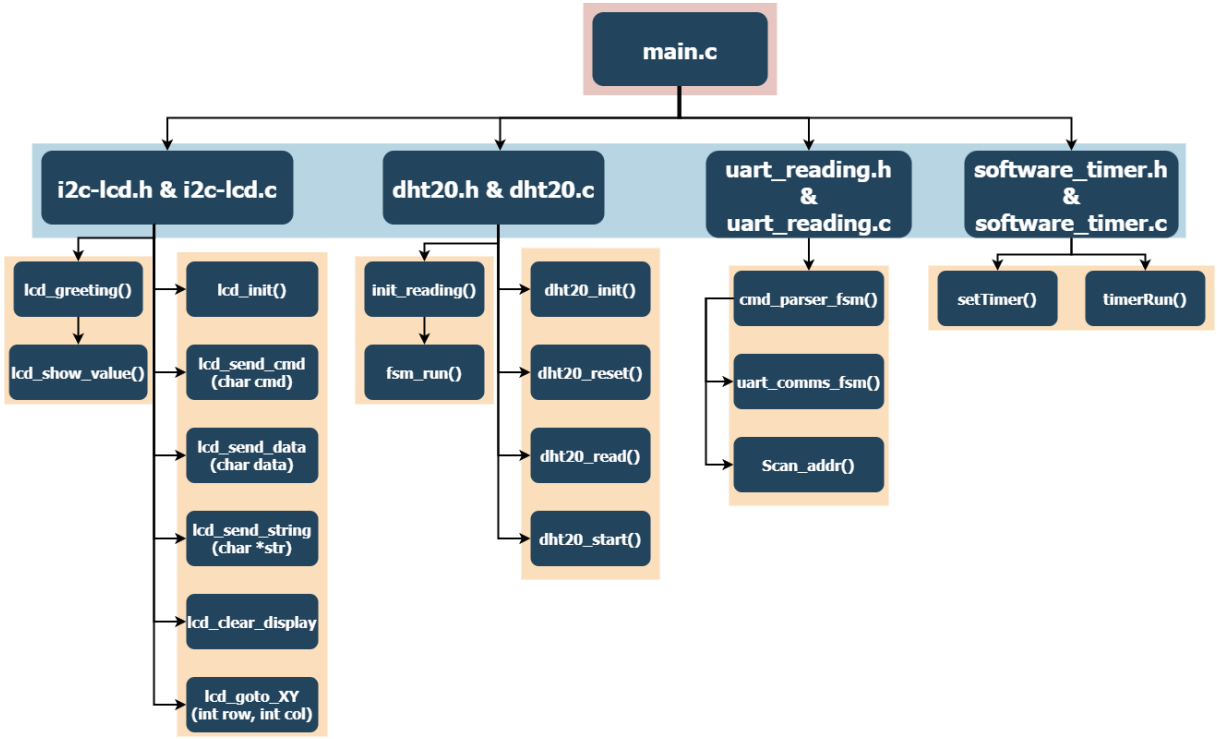

Figure 6.1: The program's layout tree

6.1.1 Display information on the LCD

The contents of this group can be found in the files **i2c-lcd.h** for prototype declaration and **i2c-lcd.h** for the implemntaion. These files including a set of function that are capable of:

- Initialize the LCD to 4-bit interface
- Send data, string and command to the LCD
- Clear the LCD
- Move cursor to any location
- Send greeting for Waiting Mode
- Display measured values

6.1.2 Scanning the device address and interpreting the command

The files **uart_reading.h** and **uart_reading.c** are now in charged for this group of operation. Generally, they will peform the following tasks:

- Parse the input and determine the corresponding command (Here finite state machines are used)
- Do the associated action with each predefined command
- Scanning the address of I^2C devices including the DHT-20 sensor and LCD-1602A

6.1.3 Reading value from the DHT-20 Sensor and Combining to a complete system

To service this group, **dht20.h** and **dht20.c** are the needed files. They will execute the following jobs:

- Initialize, reset, and start reading operation of the DHT-20 sensor.
- Provide a finite state machine that allow the 3-second measuring routine.

6.1.4 Software Timer

This group of function is used to provide a software timer for synchronizing the operation of our system. Since the interrupt is invoked every 10ms, calling the function setTimer(100) will give us a 1s timer interrupt.

6.2 Finite State Machine

6.2.1 For Command Parser

This FSM is built through the function **cmd_parser_fsm()** in the file **uart_reading.h** for prototype declaration & **uart_reading.c** implementation. The main feature of this FSM is to parse the input sequence from user and determine which kind of command is it.

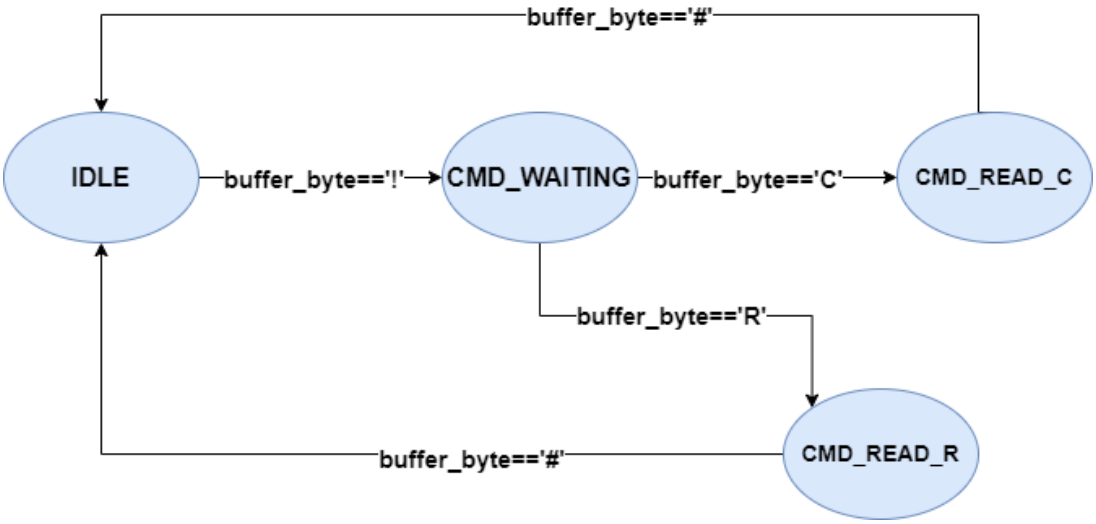

Figure 6.2: FSM For Command Parser

6.2.2 For fully operated system

This FSM is built through two function which are **uart_control_fsm()** and **reading_fsm_run()**. They are located in the set **uart_reading** and **dht20**, respectively. The function **uart_control_fsm()** is in charged of intercepting the corresponding flag to change from **Capture** mode to **Reset** mode and vice versa. Meanwhile, the **reading_fsm_run()** will successively fetch data from the DHT20 sensor every 3 seconds.

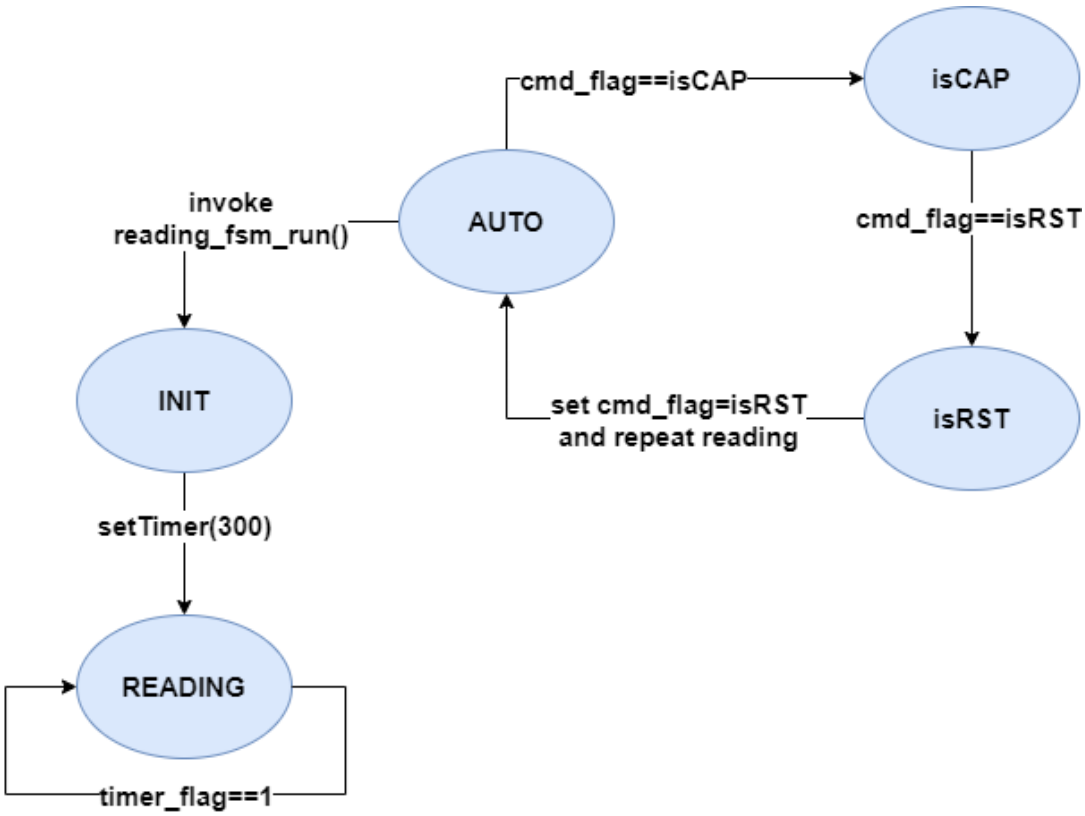

Figure 6.3: FSM For fully operated system

6.2.3 Usage of FSMs

The Command Parser FSM runs first, it will successively get the input and compare whether the receiving message is matched with predefined commands. If so, the remaining task is to set the corresponding **cmd_flag**. This will act as a signal for the Full Operated System FSM to process.

At the beginning, if the user do not type in anything, the Command Parser FSM stays still at IDLE state. However, since the **cmd_flag** is set AUTO by default, the Full Operated System FSM will then calls **reading_fsm_run()** to start the measurement.

6.3 Implementation

For version control and synchronization, we use Git and GitHub to manage the project. In order to gain the full access including materials and code, please follow this link:

https://github.com/cmq2002/Expansion_shield_STMNucleo

CHAPTER 7

EXPERIMENTS

This chapter mainly includes images of the system in different operation mode. It is used to ensure the correctness of our design and implementation.

Firtsly, system sits in Waiting mode for a short duration of time before move to Reading mode.

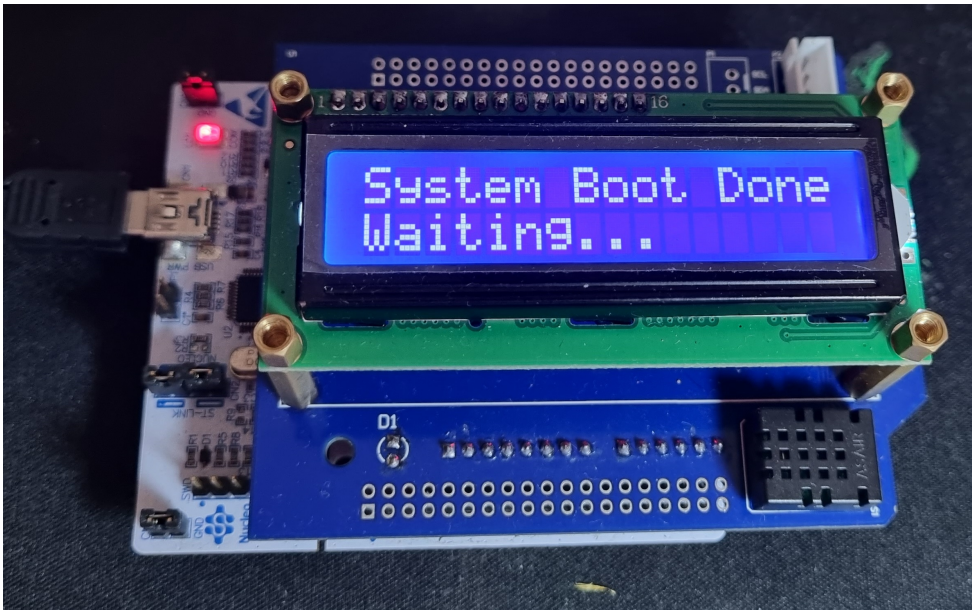

Figure 7.1: The System in temporary intermediate state

Now, it is successful in measure the temperature and humidity of the surrounding environment.

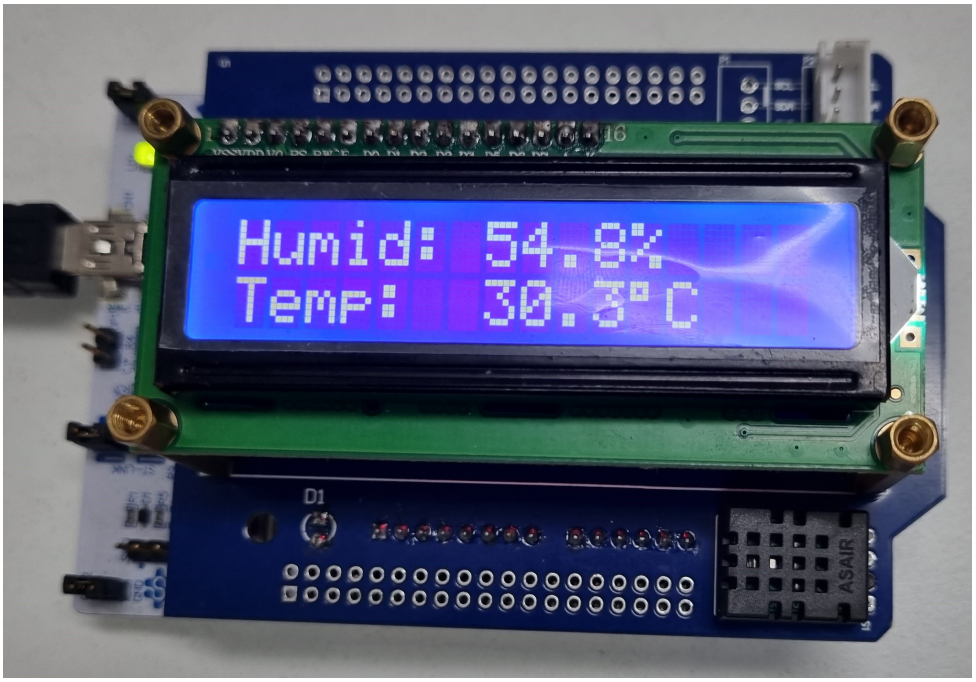

Figure 7.2: The System in fully operation

CHAPTER 8

CONCLUSION & PERSPECTIVE

To sum up, we have fulfilled all goals that set at the beginning of the project's research. With the persistence, well-organization and cooperation between team members, self-study capability and the assistance from the instructors, we have overcome all problems that arise during the research's process.

On the other side, there are lots of difficulties we have encountered while doing the research. For instance, we have spent weeks on figuring out the address of the I^2C devices since it is a core factor to ensure the correct operation. So that, all of the other tasks need to be delayed until the address has been found. Furthermore, in order to complete the project we need to read and understand many user manuals, device data sheets as well as academic references to proceed constructing the system, program designs appropriately and optimize the code for better performance.

In the near future, we would like to extend our work to the IoT's direction. Specifically, our module will act as a machine to collect information about the quality of the air including the temperature, humidity, the presence of harmful gases (for instance, Ammonia - NH_3, Sulfur - S, Benzene - C_6H_6, Carbon dioxide - CO_2, etc.) using MQ-135 sensor and also the concentration of PM 2.5 dust particles with Optical Dust Sensor PM 2.5 GP2Y1010AU0F.

All the relevant information collected from the sensors system will be stored on the Ada Fruit Server and being exploited by the mobile application. The

app will then be in charged of showing the information and pushing notification in case the air quality indexes exceeds the safety boundary. Another feature that we have thought of, is to apply statistical and AI models in attempting to establish prediction on the quality of the air in the near future. Consequently, we are able to give an early warning so that people have time to protect their own health by putting on appropriate mask that can prevent fine dust or turning on the air purifier.

Moreover, the app also allow users to turn on or turn off the machine. This aspect will replace the old fashion UART communication protocol in sending command to the system through a host computer and it would be much more convenient in compare with the UART way as not everyone is familiar with using the simulation tools.

REFERENCES

[1] Robert C.Martin, *"Clean Code - A Handbook of Agile Software Craftsmanship"*, Pearson Education, 2008

[2] Steve McConnell, *"Code Complete - A practical handbook of software construction"*, Microsoft Press, 2004

[3] Michael J.Pont, *"Patterns for Time-Triggerd Embedded Systems"*, ACM Press & Pearson Education Limited, 2010.

[4] Jeri R.Hanly & Elliot B.Koffman, *"Problem Solving and Program Design in C"*, 7^{th} Edition, Pearson, 2012.

[5] STMicroelectronics, *"Reference manual for STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs"*, 2018.

[6] STMicroelectronics, *"User Manual - Description of STM32F1 HAL and low-layer drivers"*, 2020.

[7] ASAIR, *"Data Sheet DHT20 - Humidity and Temperature Module"*, 2021.

[8] Crystalfontz America, Inc., *"Data Sheet CFAH1602A-AGB-JP"*.

[9] NXP Semiconductors, *"Product data sheet PCF8574; PCF8574A - Remote 8-bit I/O epxander for I^2C-bus with interrupt"*, 2013.

[10] STMicroelectronics, *"STM32F103x6, STM32F103x8, STM32F103xB - Performance line, ARM-based 32-bit MCU with Flash, USB, CAN, seven 16-bit timers, two ADCs and nine communication interfaces - Preliminary Data"*, 2007.

[11] STMicroelectronics, *"User Manual - STM32 Nucleo-64 boards (MB1136)"*, 2020.

[12] Scott Campbell, *"Basics of UART Communication"*, https://www.circuitbasics.com/basics-uart-communication/

[13] Scott Campbell, *"Basics of I2C Communication Protocol"*, https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

[14] WatElectronics, *"What is I2C Protocol : Architecture, Timing Diagram Its Applications"*, 9^{th} February 2022, https://www.watelectronics.com/i2c-protocol/

[15] Matt Mielke, *Using the STM32Cube HAL I2C Driver in Master Mode*, https://forum.digikey.com/t/using-the-stm32cube-hal-i2c-driver-in-master-mode/15122

STUDENT INFORMATION

List of project authors:

1. **Cao Minh Quang** - ID: 2052221
    - Phone number: +84 857 333 159
    - Email: quang.caoktmtk2020@hcmut.edu.vn
2. **Lê Tuấn Hưng** - ID: 2052508
    - Phone number: +84 938 173 868
    - Email: hung.lechpro@hcmut.edu.vn
3. **Trần Cao Duy Trường** - ID: 2052299
    - Phone number: +84 367 243 065
    - Email:truong.tranmirai@hcmut.edu.vn