A logo of a computer science

Description automatically generated Custom Text Display on 1602 LCD

**Session**: 2022 – 2026

**Submitted by:**

Shammas Ahmad 2022-CS-83

Usman Kibria 2022-CS-90

Khalil Ahmed 2022-CS-93

**Supervised by:**

Syed Tehseen ul Hasan Shah

Department of Computer Science

**University of Engineering and Technology Lahore Pakistan**

A circuit board with wires and a display

Description automatically generated

Description**:**

This endeavor involves the development of a sophisticated system for rendering customized textual content on a **1602 liquid crystal display (LCD)**. The core of this system is an **ESP32 microcontroller unit (MCU)**, which is responsible for retrieving the designated text via a specifically allocated wireless network connection known as “**espnet839039**”. To facilitate this communication, the system employs **Message Queuing Telemetry Transport (MQTT)** – a lightweight, publish-subscribe network protocol optimized for the transmission of telemetry data between devices. MQTT communication is achieved through the utilization of public brokers such as **test.mosquitto.org** or **broker.hivemq.com**.

Upon receipt of the textual information, the system further processes the data using an Arduino UNO, a microcontroller board based on the AVR microcontroller architecture. After this processing, the Arduino UNO directs the output to an LCD 1602 display, which is thus able to exhibit the received text.

Complementing the hardware components is the inclusion of a real-time database hosted on Firebase, which serves dual purposes. Firstly, it meticulously archives each instance of text displayed, thereby creating a comprehensive historical log. Secondly, the database functions as a contingency storage solution, safeguarding the system's data.

An additional feature of this system is the integration of a push button, which, upon activation, initiates a countdown sequence followed by a reiteration of the most recently displayed text on the screen.

This multifaceted technology boasts a broad spectrum of practical applications: from disseminating advertising content and municipal information, such as traffic regulations and directional guidance on roadways, to providing real-time updates on flight schedules at airports and queue numbers within financial institutions. The utility of this device is evident across various facets of day-to-day operations, making it an invaluable asset in contemporary informational display solutions.

Methodology**:**

In our project, we mixed ready-made code blocks given by our supervisor with our own custom code to create an efficient system for handling data. We used a Firebase Realtime Database to manage and store our data. Most of this was done using C++ code, which also helped us connect with an MQTT broker, an important part of our system.

Whenever users type something, it's immediately sent to the Firebase database, which keeps the data safe and updates everything in real time. This is very important for making sure our system is stable and reliable.

We chose an UNO Arduino microcontroller to manage the user's text and run our data processing code. We wrote this part of the code in assembly language, which is great for fast performance and working closely with the hardware. We organized this code in Microchip Studio to make it easy to manage and maintain.

We carefully created routines to take apart the text strings and send each letter one by one to the LCD screen to show the user. We've made sure the process is smooth so that after text is shown, the cursor is hidden to make the display look clean.

Our project also uses a button that users can push to start a countdown from 9 to 0, taking ten seconds. Before starting the countdown, we need to send the text we want to show afterward to the Arduino. When the countdown ends, the text is immediately shown.

By using both hardware like buttons and versatile programming, our project is user-friendly and can show data and respond to users effectively.

Data flow diagram**:**

Detailed flow chart**:**

Firebase Realtime Database**:**

Circuit diagram**:**

Working of components**:**

ESP32**:**

The ESP32 microcontroller series is an economical and power-efficient range of modules with integrated Wi-Fi and Bluetooth functionality, engineered by Espressif Systems based in China. Its design is particularly suitable for Internet of Things (IoT) implementations because of its robust wireless communication capabilities and conservative energy consumption profile. This makes it a preferred choice for our project. We have established a connection with the ESP32 to an MQTT (Message Queuing Telemetry Transport) broker through the Wi-Fi network named "espnet839039." This connection facilitates communication with commonly utilized free MQTT brokers such as test.mosquitto.org or broker.hivemq.com.

The primary function of the ESP32 within our system is to act as a receiver for textual data transmitted via the MQTT broker. This data is subsequently forwarded to an Arduino unit for additional processing tasks. The LED indicators on the ESP32 are programmed such that the red LED remains illuminated when the device is not connected to the internet. Conversely, upon establishing a network connection, the red LED is deactivated. In addition to this, the ESP32 has a secondary but critical role of transferring the received text messages over the internet to a Firebase Realtime Database. This operation not only maintains a sequential record of the transmitted information but also serves as a means for data redundancy and recovery. The implementation of these functionalities has been achieved through programming in C++, and communication with the MQTT broker occurs over the standard port designated as 1883.

UNO Audino**:**

The Audino microcontroller is deployed as an intermediary data processor. It is tasked with receiving and processing textual information to be visually represented on a digital screen. The ingestion of textual data is carried out through a communication protocol with an ESP32 unit. Following the reception of this data, the Audino microcontroller temporarily stores it in a designated data buffer. This facilitates sequential retrieval of individual characters, which are subsequently transmitted for display.

Within the microcontroller's processing workflow, the textual string is initially transferred from the UDRO register into register R22. After this transfer, the microcontroller executes a subroutine labeled ‘print string’. This routine initiates with the preservation of state by pushing the contents of registers R16 through R21 onto the stack, ensuring that their current values are not lost during the string printing process. Thereafter, the microcontroller clears the display to prepare for the new text output.

1602 LCD**:**

The 1602 liquid crystal display module is utilized to present textual information, featuring two distinct rows capable of rendering such content sequentially. Upon the saturation of the initial row's capacity, the progression of text output naturally transitions to the subsequent row. The text's origination point is the ESP32 microcontroller, which conveys the data to an Arduino board. The Arduino, functioning as an intermediary, sequentially transmits individual characters of the string to the LCD module. Upon completion of this transfer, the LCD module is activated while the onboard cursor is disabled, providing an uninterrupted visual presentation of the text.

The implementation of this textual transmission and display process is orchestrated through code crafted in AVR assembly language, which is known for its low-level hardware control capabilities, offering fine-grained management of the device's operations.

Complementary to the text display feature, a tactile switch, colloquially known as a pushbutton, is integrated into the system. Interaction with this element initiates a countdown sequence on the LCD, decrementing numerically from 9 to 0. Upon the culmination of this countdown, the LCD reverts to showcasing the text that was previously active before the initiation of the countdown sequence. This feature adds an interactive element, allowing users to trigger specific display behaviors on the LCD module.

AVR Module code**:**

.include "m328pdef.inc"

.include "UART\_Macros.inc"

.include "1602\_LCD\_Macros.inc"

.include "delay\_Macro.inc"

.org 0x0000

jmp main

.org PCI2addr

jmp PCINT2\_ISR

.dseg

.org 0x0100

data\_buffer: .BYTE 128 ; Buffer to store string

.cseg

main:

LDI r16, high(RAMEND) ; Set Stack Pointer to end of the RAM

OUT SPH, r16 ; it is necessary when using interrupt vectors

LDI r16, low(RAMEND)

OUT SPL, r16

SBI DDRB, PB5

CBI DDRD, PD2 ; PD2 set as INPUT pin (push button1)

SBI PORTD, PD2 ; Enable internal pull-up resistor

LDI r16, 0b00000100 ; enabling PCIE2 interrupts

STS PCICR, r16

LDI r16, 0b00000100 ;enabling PCINT18 interrupt (PD2 Pin)

STS PCMSK2, r16

SEI

LCD\_init

Serial\_Begin

loop:

LDI XL,LOW(data\_buffer) ; Set string pointer to the beginning of buffer

LDI XH,HIGH(data\_buffer)

rx:

LDS R21, UCSR0A

SBRS R21, RXC0 ; USART Receive Complete

RJMP rx

LDS R22, UDR0

CPI R22, 0x0D ; Check if carrage return character is received

BRNE store\_char ; Store the received character in buffer

LDI R22, 0x00 ; Store null termination character

ST X+, R22

LDI XL, LOW(data\_buffer) ; Set string pointer to the beginning of buffer

LDI XH, HIGH(data\_buffer)

call print\_string

RJMP loop

store\_char:

ST X+, R22

RJMP rx

;Prints string on LCD , starting index of string in X pointer

print\_string:

; save the states of R16 and R17

PUSH R16

PUSH R17

PUSH R20

PUSH R21

LCD\_send\_a\_command 0x01

LDI R21, 16 + 1

LCD\_writeBuffer\_LOOP:

; load the current byte/character pointed to be Z and increment the Z pointer

LD R16, X+

DEC R21

CPI R21, 0

BRNE c0

LCD\_send\_a\_command 0xC0

c0:

delay 15

; check if the remaining size of the string is non-zero and return if it is

CPI r16, 0

BREQ LCD\_writeBuffer\_END

LCD\_writeBuffer\_CHAR:

; Send the string character to the LCD

; Clear the pins

CBI PORTD, PD7 ; Clear PD7

CBI PORTD, PD6 ; Clear PD6

CBI PORTD, PD5 ; Clear PD5

CBI PORTD, PD4 ; Clear PD4

; Send the upper 4 bits of the Register to LCD

SBRC R16, 7 ; Check the MSB (7th bit)

SBI PORTD, PD7 ; Set PD7 according to the data bit

SBRC R16, 6 ; Check the 6th bit

SBI PORTD, PD6 ; Set PD6 according to the data bit

SBRC R16, 5 ; Check the 5th bit

SBI PORTD, PD5 ; Set PD5 according to the data bit

SBRC R16, 4 ; Check the 4th bit

SBI PORTD, PD4 ; Set PD4 according to the data bit

SBI PORTB, PB0 ; Set RS pin to HIGH (set LCD mode to "Data Mode")

SBI PORTB, PB1 ; Set E pin to HIGH (set LCD to receive the data)

delay 10

CBI PORTB, PB1 ; Set E pin to LOW (set LCD to process the data)

; Clear the pins

CBI PORTD, PD7 ; Clear PD7

CBI PORTD, PD6 ; Clear PD6

CBI PORTD, PD5 ; Clear PD5

CBI PORTD, PD4 ; Clear PD4

; Send the lower 4 bits of the Register to LCD

SBRC R16, 3 ; Check the 3rd bit

SBI PORTD, PD7 ; Set PD7 according to the data bit

SBRC R16, 2 ; Check the 2nd bit

SBI PORTD, PD6 ; Set PD6 according to the data bit

SBRC R16, 1 ; Check the 1st bit

SBI PORTD, PD5 ; Set PD5 according to the data bit

SBRC R16, 0 ; Check the 0th bit

SBI PORTD, PD4 ; Set PD4 according to the data bit

SBI PORTB, PB1 ; Set E pin to HIGH (set LCD to receive the data)

delay 10

CBI PORTB, PB1 ; Set E pin to LOW (set LCD to process the data)

DEC R20

RJMP LCD\_writeBuffer\_LOOP

LCD\_writeBuffer\_END:

LCD\_send\_a\_command 0x0C ; screen on, Cursor off

; restore the states of R16 and R17 and return

POP R21

POP R20

POP R17

POP R16

RET

;This procedure starts count from 9 to 0

start\_countdown:

LDI r16,9+48 ; Setting Initial Countdown value

LCD\_send\_a\_command 0x01 ; Clearing Screen

delay 1000

continue\_counter:

LCD\_send\_a\_character

LCD\_send\_a\_command 0x80 ; Moving Cursor to the start

delay 1000

DEC r16

CPI r16,48

BREQ end\_counter ; Checks if countdown reaches zero

RJMP continue\_counter

end\_counter:

;Interrupt Service Routine for Interrupt attached on PD2

PCINT2\_ISR:

SBIS PIND, PD2 ; Checking if button is pressed

RJMP l9

RETI

l9:

CLI ; Disabling Interrupts

call start\_countdown

LDI XL, LOW(data\_buffer) ; Set string pointer to the beginning of buffer

LDI XH, HIGH(data\_buffer)

call print\_string

LDI XL, LOW(data\_buffer) ; Set string pointer to the beginning of buffer

LDI XH, HIGH(data\_buffer)

SEI ; Enabling Interrupts

RETI

IOT module code**:**

// ESP32 Board: DOIT ESP32 DEVKIT V1

#include <WiFi.h>

#include <PubSubClient.h>

#include <Firebase\_ESP\_Client.h>

// Provide the token generation process info.

#include "addons/TokenHelper.h"

// Provide the RTDB payload printing info and other helper functions.

#include "addons/RTDBHelper.h"

// WiFi Credentials

const char \*ssid = "espnet839093";  // Enter your WiFi name

const char \*password = "12345678";        // Enter WiFi password

// MQTT Broker

const char \*mqtt\_broker = "test.mosquitto.org";

const char \*topic = "2022-CS-83-90-93/q3vjw7-recv";

const int mqtt\_port = 1883;

// Firebase objects

FirebaseData fbdo;

FirebaseAuth auth;

FirebaseConfig config;

// Variable to save USER UID

String uid;

// Variables to save database paths

String databasePath;

WiFiClient espClient;

PubSubClient client(espClient);

void setup() {

  // Use ESP32 buit-in LED to indicate the state of WiFi and MQTT

  pinMode(LED\_BUILTIN, OUTPUT);

  digitalWrite(LED\_BUILTIN, LOW);

  // connecting to a WiFi network

  WiFi.mode(WIFI\_STA);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    digitalWrite(LED\_BUILTIN, HIGH);  // LED ON while No WiFi

    delay(500);

  }

  digitalWrite(LED\_BUILTIN, LOW);  // LED OFF when connected to WiFi

  //connecting to a mqtt broker

  client.setServer(mqtt\_broker, mqtt\_port);

  client.setCallback(callback);

  while (!client.connected()) {

    digitalWrite(LED\_BUILTIN, HIGH);  // LED ON while No MQTT Connection

    String client\_id = "esp32-client-";

    client\_id += String(WiFi.macAddress());

    if (client.connect(client\_id.c\_str())) {

      digitalWrite(LED\_BUILTIN, LOW);  // LED OFF when connected to MQTT Server

    } else {

      delay(2000);

    }

  }

  client.subscribe(topic);  // Subscribing to a MQTT topic

  // Assign the api key

  config.api\_key = "AIzaSyD6VLAGIQ54X3hEnepiHbW42m8qR5E-OJA";

  // Assign the user sign in credentials

  auth.user.email = "server@coalproject.com";

  auth.user.password = "12345678";

  // Assign the RTDB URL

  config.database\_url = "https://coal-project-default-rtdb.firebaseio.com/";

  Firebase.reconnectWiFi(true);

  fbdo.setResponseSize(4096);

  // Assign the callback function for the long running token generation task

  config.token\_status\_callback = tokenStatusCallback;

  // Assign the maximum retry of token generation

  config.max\_token\_generation\_retry = 5;

  // Initialize the library with the Firebase authen and config

  Firebase.begin(&config, &auth);

  // Getting the user UID might take a few seconds

  while ((auth.token.uid) == "") {

    delay(1000);

  }

  // Print user UID

  uid = auth.token.uid.c\_str();

  // Update database path

  databasePath = "/UsersData/" + uid;

  Serial.begin(9600);

}

void callback(char \*topic, byte \*payload, unsigned int length) {

  String msg = "";

  for (int i = 0; i < length; i++) {

    Serial.print((char)payload[i]);

    msg += (char)payload[i];

  }

  Serial.write(0xd); // Write Carriage return character

  while (!Firebase.RTDB.pushString(&fbdo, "/UsersData/" + uid + "/msgs", msg)); // Pushing data to Firebase database

}

void loop() {

  client.loop();

}

Code documentation**:**

The IOT code is written in C++. It is used to connect the ESP32 to the internet named “**espnet839039**” to utilize the public MQTT brokers such as **test.mosquitto.org** or **broker.hivemq.com**. The primary function of the ESP32 within our system is to act as a receiver for textual data transmitted via the MQTT broker. This data is subsequently forwarded to an Arduino unit for additional processing tasks. The LED indicators on the ESP32 are programmed such that the red LED remains illuminated when the device is not connected to the internet. Conversely, upon establishing a network connection, the red LED is deactivated. In addition to this, the IOT code controlling the ESP32 has a secondary but critical role of transferring the received text messages over the internet to a Firebase Realtime Database. This operation not only maintains a sequential record of the transmitted information but also serves as a means for data redundancy and recovery. The implementation of these functionalities has been achieved through programming in C++, and communication with the MQTT broker occurs over the standard port designated as 1883.

IOT code**:**

Libraries**:**

As we enter the IOT code we first include the libraries

* “**Wifi.h,**
* **PubSubClient.h**
* **firebase\_ESP\_Client.h**
* addons/Tokenherlper.h
* addons/RTDBhelper.h.

Main**:**

In the main function we define the Wifi, MQTT broker and Firebase credentials.

The Wifi name is “**espnet839039**”.

The MQTT broker URL is **test.mosquitto.org** or **broker.hivemq.com**.

The topic of the connection is "**2022-CS-83-90-93/q3vjw7-recv**"

The MQTT port to which the connection is made for communication has the id “**1883**”.

The firebase credentials include the user email address and the API.

AVR Code**:**

Libraries**:**

The included libraries and macros of our code are

* **"m328pdef.inc"**
* **"UART\_Macros.inc"**
* **"1602\_LCD\_Macros.inc"**
* **"delay\_Macro.inc"**

These macros were defined by our Project supervisor to help with the code, which we edited in accordance with our project needs.

Procedures**:**

This section defines the procedures that are being used in AVR module:

#### main loop:

firstly, Address of buffer to store a string is moved in X index pointer. Then check UCSR0A register if any character is received through USART. If any character received other than Carriage Return character then is stored at address present in X index register and X is incremented, Otherwise a null termination character is stored in X index register and print\_string procedure is called with starting address of buffer in X index register.

print\_string:

The print\_string process starts by taking the location of a chosen string using the X register. It starts working by protecting the Arduino microcontroller registers that are being used now, especially R16 to R22. It saves them onto a storage place called system stack. This protecting step is very important to stop their current conditions from being changed, because the process might change what they have.After the safe storage of registers, the process continues to give a clear screen command to a 1602 LCD device. This action basically starts the screen again, giving a fresh page for when you want to put out words.After the LCD setup is done, it checks if the string's length matches higher than zero. This step is very important to make sure that a good, not empty word has been given for showing. This stops wasting resources on no content at all.  
  
After this double-check step, the process moves onto the next part and goes through a set of labels. These tags act as pointers or turning points in the series of steps, directing how to do things. They mark parts of code meant for particular tasks in the bigger job to show a string on an LCD. Each label is for a specific part of the total job, putting the program in order from start to finish. It ends with showing words on that screen called 1602 LCD panel.

Interrupt Routines**:**

The interrupt routine is called as the PD2 pushbutton is pressed. This interaction calls the start\_countdown procedure on the LCD, decrementing numerically from 9 to 0.Upon completion the print\_string procedure is called to again display the string on the display window.

MQTT dashboard**:**

Project links:

Youtube:

https://youtu.be/k8TBRc9Y2kc

### Linkdin:

https://www.linkedin.com/posts/khalil-ahmad-411270260\_thrilled-to-unveil-the-result-of-our-collaborative-activity-7148620074759753728-eUY2?utm\_source=share&utm\_medium=member\_desktop

GitHub repository Link**:**

The GitHub repository link for the Customizable text display project:

[**https://github.com/sma02/CustomTextDisplay1602LCD.git**](https://github.com/sma02/CustomTextDisplay1602LCD.git)

References

[**Tehseen-ul-Hassan Github Link for AVR Code**](https://github.com/TehseenHasan/AVR_Assembly_Example_Codes_for_Atmega328p)