ECE 198 Final Document

Section Number: 002

Group Number: 12

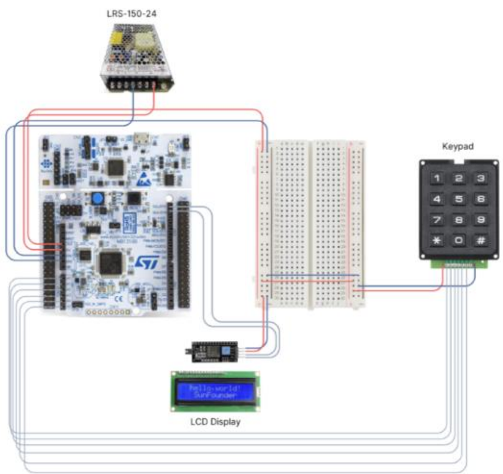
Names of Group Members: Joanne Moon (21117131), Ray Xue (21123470), Daryna Finko (21128112)

Date of Implementation Demo: 27/11/2024

Name of Grading TA: Amir Hossein Akbarian

Project Description: A portable dyslexia-friendly font translator using the STM32 Nucleo microcontroller to enhance text readability

Electrical Schematics



Version 1 (Initial version)

Simplified Design: Utilized only one board of STM32 Nucleo.

Added Components:

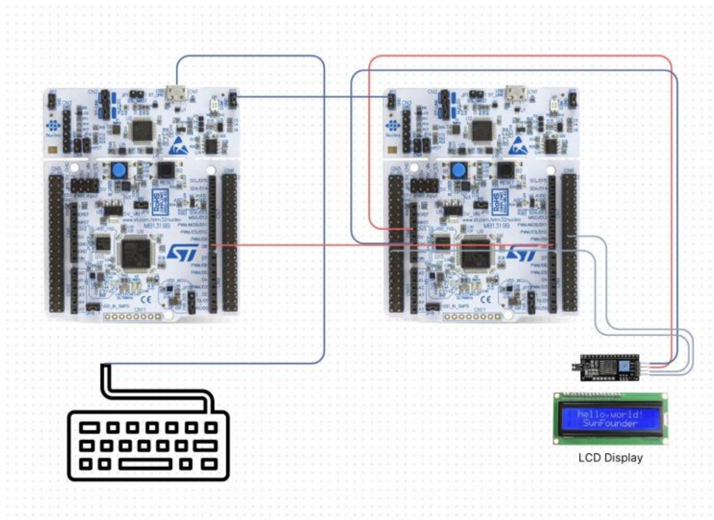
- A keypad instead of a keyboard as an input device.
- Breadboard to allow for further prototyping and organization.
- LRS-150-24 power supply unit for maintaining a steady power supply to the system.

Output: Still uses the same LCD display.

Limitations:

- The keypad limits input flexibility compared to a full keyboard, therefore limiting the range of characters that can be entered.
- This is because it risks overloading one microcontroller with all the tasks combined in case complex operations are carried out or future expansions, reducing performance and reliability.

- The lack of inter-board communication sacrifices modularity and scalability. This makes it hard to add features or components in the future.



Version 2 (Final Version)

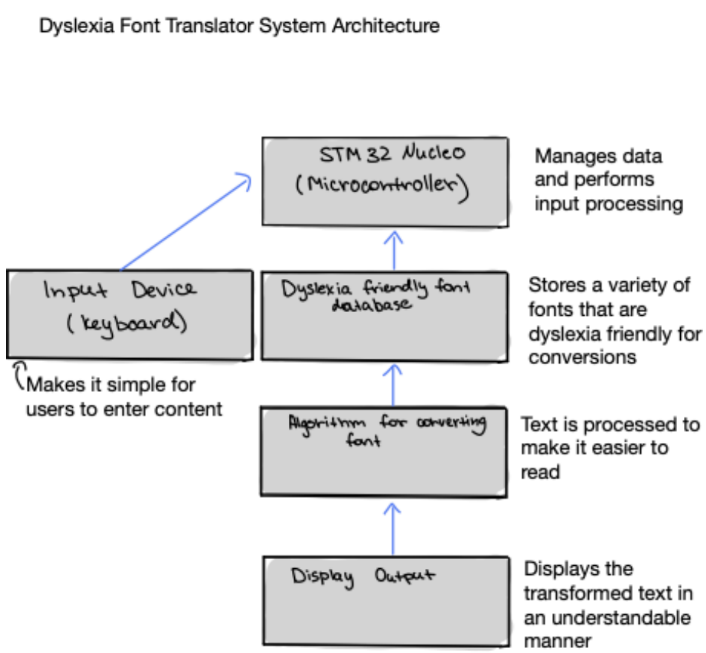
- **Setup:** Includes two STM32 Nucleo boards connected in parallel.
- **Peripheral Input/Output:** It contains a keyboard as input, connected to one board, and an LCD display as output, connected to the second.
- **Complexity**: Since there must be inter-board communication, this increases wiring complexity but does allow distributed processing—in other words, each board will focus on its specific task.
- **Functionality:**

One STM32 board processes the keyboard input and sends it to the second board, dedicated to display output. This modular design enables real-time performance by reducing the processing load for each microcontroller.

- **Efficiency:**

While this setup utilizes two microcontrollers, the system efficiency improves by spreading the tasks across the board. Each microcontroller operates within its optimal range to prevent overloading and maintain reliable, scalable operation.

System Architecture Design



Version 1 (Initial Version)

Overview:

- Consolidates all the tasks on one board: the STM32 Nucleo board.
- Replaces the keyboard with a keypad for input.
- Uses a breadboard and an external power supply to make prototyping and operation easy.

Key Components:

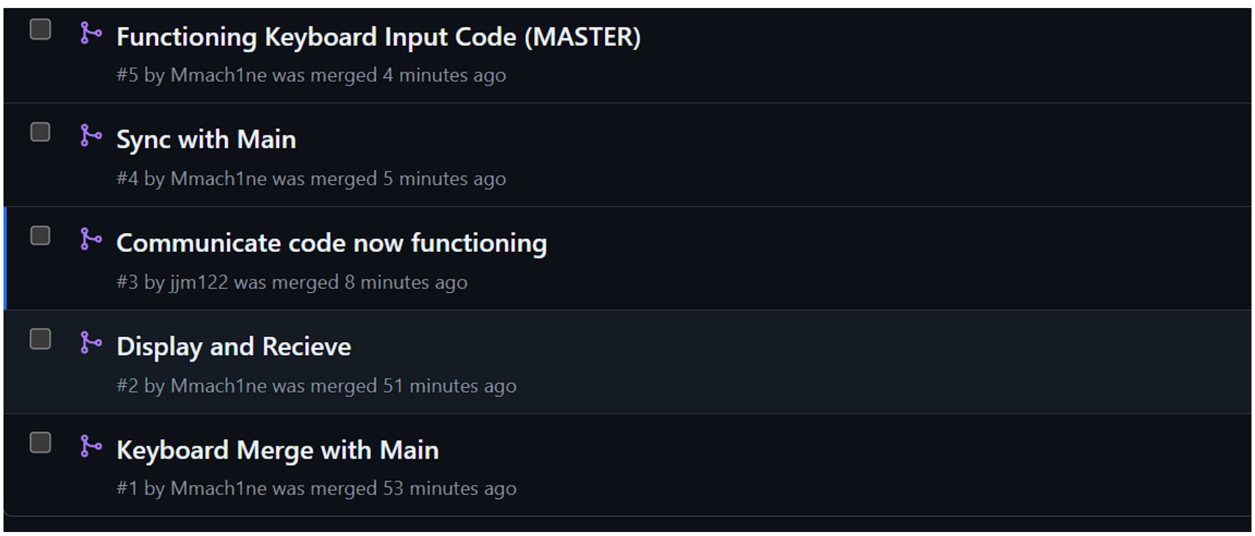
- **Input Device (Keypad):**
  Limited character input compared to a full keyboard.
  Great for prototyping, but less user-friendly for complex inputs.
- **Single STM32 Board:**
  Handles all tasks related to input processing, error detection, font conversion, and output control.

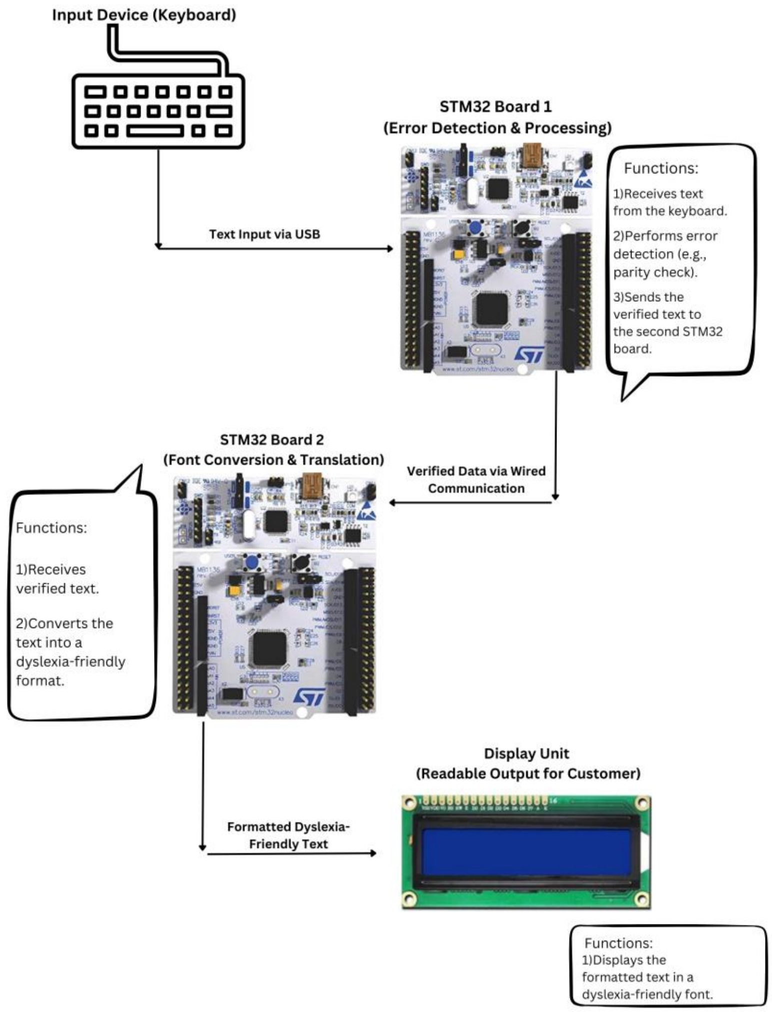
- **Breadboard:**
  Facilitates additional connections and prototyping.
- **Power Supply (LRS-150-24):**

Supplies stable power to the system.

Limitations:

- **Reduced Input Flexibility:**
  The keypad narrows the range of input characters, making it less suitable for a dyslexia translator.
- **Risk of Overloading:**
  Consolidating all tasks onto one board increases the risk of performance bottlenecks during operation.
- **Lack of Modularity:**
  Upgrading the system in the future, such as extending it with new functions, is more complicated with a single-board design.





Version 2 (Final Version)

Overview:

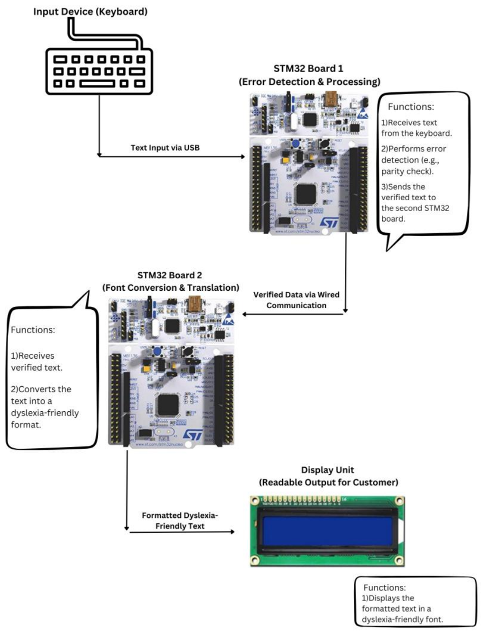
- Uses two STM32 Nucleo boards for parallel processing:
  - **STM32 Board 1:** Detects errors and processes text input from the keyboard.
  - **STM32 Board 2:** Converts text into a dyslexia-friendly format and sends it to the display.
- Clear demarcation of tasks guarantees modularity, scalability, and optimized system performance.

Key Components:

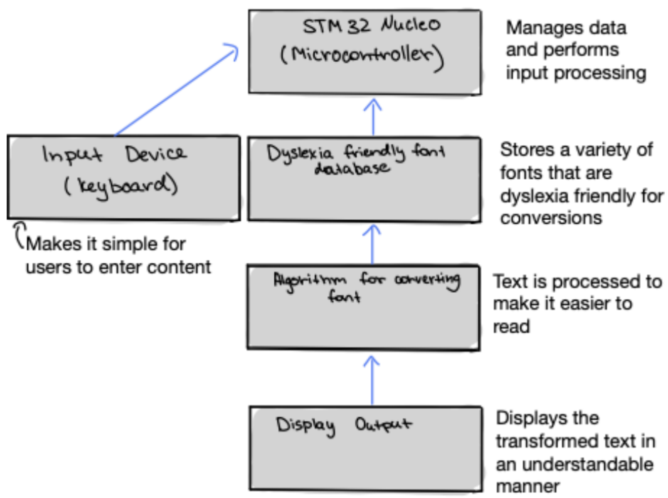
- **Input Device (Keyboard):**
  Provides complete flexibility for the user to enter any text.
  Connected via USB to STM32 Board 1.
  Performs basic error checking, such as parity checks, to verify input accuracy.
- **STM32 Board 1 (Error Detection & Processing):**
  Processes and verifies the input text.
  Sends the validated data to STM32 Board 2 for font conversion.
- **STM32 Board 2 (Font Conversion & Translation):**
  Converts the verified text into a dyslexia-friendly font.
  Communicates with the display unit to provide formatted text output.
- **Display Unit:**
  Outputs the formatted dyslexia-friendly text in a readable manner for users.

Advantages:

- **Efficiency:**
  By delegating specific tasks to each STM32 board, the system avoids overloading a single microcontroller.
- **Modularity:**
  Allows for easy future scalability, extending capabilities to support more fonts or additional error-checking algorithms.
- **Real-Time Performance:**
  Parallel processing ensures quick response time for user input and display output.



Dyslexia Font Translator System Architecture



Change Log (Version 1 to Version 2)

- **Input Device:**
  Replaced the keypad with a keyboard for broader character input and better usability.
- **Microcontroller Configuration:**
  Added a second STM32 board to modularize error detection and processing of font conversion.
- **Power Supply:**
  Removed the external power supply, relying on USB power for simplicity
- **Breadboard:**
  Obviated the use of a breadboard to reduce system complexity and minimize connections.

Pseudocode

Version 1 (Initial version) :

```
// STM32 Pseudo-Code for Main Functions

void receiveMessage() {
    // Receive message and check for errors
    message = readFromLaptop();
    if (checkParity(message) && checkErrorSum(message)) {
        message = correctErrors(message);
        translateMessage(message);
        formatDyslexiaFriendly(message);
        sendMessage(message);
    } else {
        requestRetransmission();
    }
}

bool checkParity(String message) {
    // Logic for parity check
}

bool checkErrorSum(String message) {
    // Logic for error sum check
}

String correctErrors(String message) {
    // Apply error correction algorithm
}

void translateMessage(String message) {
    // Simple English-French translation logic
}

String formatDyslexiaFriendly(String message) {
    // Apply dyslexia-friendly formatting
}

void sendMessage(String message) {
    // Transmit to next STM32 or display device
```

```
}
```

Focus and Simplicity:

The initial pseudocode operates at a high level, mainly outlining the core functionalities of error detection, correction, and formatting. While functions such as checkParity, checkErrorSum, and correctErrors are specified, their implementation details are not included, making the design incomplete.

Translation Logic:

The translateMessage function is presented as a placeholder, with unclear logic focusing on basic English-to-French translation. This approach does not align with the intended goal of translating text into a dyslexia-friendly font, leaving a critical gap in functionality.

Modular yet Limited:

The design adopts a modular structure with functions like receiveMessage and sendMessage. However, it lacks crucial details, such as hardware initialization steps or the mechanisms for inter-board communication. The workflow is overly simplistic, overlooking real-time constraints, resource validation, and the setup required for reliable operation.

Error Handling:

Error handling is minimal, relying heavily on retransmission as the primary method to address failures. This approach does not provide robust mechanisms to ensure data integrity or recover from persistent errors.

Version 2 (Final Version):

// START PROGRAM

// Step 1: Initialization Phase

```
FUNCTION InitializeSystem()
  INITIALIZE communication for input device
    CONFIGURE parameters such as data rate, word length, and stop bits
    ENABLE necessary modes for transmitting and receiving

  INITIALIZE communication between components
    ASSIGN unique identifier for communication
    CONFIGURE data transfer settings
    ENABLE communication-related interrupts

  INITIALIZE output display
    CALL display_setup()       // Prepare the display
    CALL display_clear()       // Clear any existing data
    CALL display_message("System Ready")  // Indicate readiness to the user

  LOAD required resources
    VERIFY that all resources are successfully loaded
    IF any resource fails to load THEN
      DISPLAY "Load Error" and STOP PROGRAM
    END IF
END FUNCTION
```

// Step 2: Main Execution Loop

```
FUNCTION Main()
  CALL InitializeSystem()

  WHILE True DO
```

```
    // Handle input data from the user
    DECLARE inputBuffer AS ARRAY
    CALL CaptureInput(inputBuffer)

    // Validate and correct input data
    IF ValidateInput(inputBuffer) THEN
      CALL FixInputErrors(inputBuffer)
      IF CorrectionFails THEN
        CALL display_clear()
        CALL display_message("Input Error. Retry.")
        CONTINUE // Restart the loop
      END IF
    END IF

    // Modify input data for desired output format
    DECLARE formattedBuffer AS ARRAY
    CALL ReformatData(inputBuffer, formattedBuffer)

    // Transmit processed data to another module
    CALL TransmitData(SourceModule, TargetModule, formattedBuffer)

    // Receive and render output data
    DECLARE outputBuffer AS ARRAY
    CALL ReceiveData(TargetModule, outputBuffer)
    CALL RenderOutput(outputBuffer)
  END WHILE
END FUNCTION
```

// Function: Capture Input

```
FUNCTION CaptureInput(outputBuffer)
  DECLARE tempBuffer AS CHAR
  DECLARE bufferIndex AS INTEGER = 0
  CALL display_clear()
  CALL display_message("Enter text:")

  WHILE True DO
    READ tempBuffer FROM input device
    IF tempBuffer == 'Submit' THEN
      BREAK // End input capture
    ELSE
      outputBuffer[bufferIndex] = tempBuffer
      INCREMENT bufferIndex
    END IF
  END WHILE
END FUNCTION
```

// Function: Validate Input Data

```
FUNCTION ValidateInput(data)
  DECLARE validationCheck AS BOOLEAN
  PERFORM validation logic ON data
  RETURN True IF validation fails, ELSE False
END FUNCTION
```

// Function: Fix Input Errors

```
FUNCTION FixInputErrors(data)
  APPLY corrective algorithm TO data
  RETURN corrected data IF successful, ELSE "Correction Failed"
END FUNCTION
 // Function: Reformat Data
FUNCTION ReformatData(inputData, outputData)
  DECLARE mappingRules AS DICTIONARY
  FOR EACH character IN inputData DO
    IF character IN mappingRules THEN
      APPEND mappingRules[character] TO outputData
    ELSE
      APPEND character TO outputData // Retain unrecognized inputs
    END IF
  END FOR
END FUNCTION
```

// Function: Transmit Data

```
FUNCTION TransmitData(source, target, data)
  ENCODE data FOR transmission
  SEND data TO target
END FUNCTION
```

// Function: Receive Data

```
FUNCTION ReceiveData(target, receivedData)
  WAIT FOR data FROM source
  STORE data IN receivedData
```

```
END FUNCTION

 // Function: Render Output
FUNCTION RenderOutput(data)
   CALL display_clear()
   FOR EACH character IN data DO
       CALL display_render_character(character) // Render formatted characters
   END FOR
END FUNCTION
 // Helper Function: Load Resources
FUNCTION LoadResources()
   LOAD necessary assets INTO memory
   RETURN True IF successful, ELSE False
END FUNCTION
// END PROGRAM
```

Comprehensive System Flow:

The final pseudocode provides a detailed, step-by-step flow covering system initialization, main execution, and helper functions. It integrates hardware-specific details, including UART and I2C communication, setting up the LCD, and displaying messages.

Enhanced Modularization:

- Divides functionalities into clearly defined phases:
  - Initialization Phase
  - Main Execution Loop
  - Specific data handling functions like CaptureInput, ValidateInput, and ReformatData.

Each function operates independently while seamlessly fitting into the overall workflow.

Focus on Dyslexia-Friendly Features:

- Replaces the translateMessage function with a specialized dyslexia-friendly formatting logic in ReformatData.
- Implements a mappingRules dictionary that explicitly converts characters into their dyslexia-friendly equivalents.

Robust Error Handling:

- Incorporates comprehensive input validation (ValidateInput) and error correction (FixInputErrors).
- Ensures graceful failure handling, such as notifying users of errors and restarting the process.
- Provides feedback messages on the LCD to guide users when input errors or other issues occur.

Hardware Integration:

- Introduces dedicated functions for hardware initialization (InitializeSystem), input capture (CaptureInput), and output rendering (RenderOutput).
- Supports inter-board communication with functions for data transmission (TransmitData) and reception (ReceiveData).

Scalability:

- Designed for extensibility, allowing for additional font rules or character mappings through ReformatData.
- Includes resource loading and validation (LoadResources) to ensure the system operates reliably under various conditions.

Version 1 vs Version 2

Feature	Initial Version	Final Version
System Initialization	Minimal setup for basic components like input and output.	Comprehensive setup, including UART for keyboard, I2C for inter-board communication, and LCD initialization
Error Handling	Relies on simple retransmission in case of failure.	Incorporates detailed error detection, correction, and feedback mechanisms to ensure robust handling.
Font Translation	Focused on basic English-to-French translation.	Implements dedicated logic for dyslexia-friendly font formatting using ReformatData.
Workflow	High-level function calls with minimal integration details.	Step-by-step execution phases with modular, well-integrated functions for smooth system operation.
Feedback	No mechanisms for user feedback.	Provides user prompts and error messages via the LCD for better guidance and interaction.
Hardware Integration	Assumes hardware works without any explicit configuration.	Includes configuration and testing of peripherals like communication interfaces and the display.
Scalability	Limited modularity, making it difficult to expand.	Fully modular design, adaptable for adding new features or enhancements in the future.

Project Tracker