

TECHNICAL UNIVERSITY OF CRETE

DIPLOMA THESIS

*EMBEDDED SYSTEM FOR MULTIPLE-USER ACCESS TO
MICROPROCESSOR-BASED EDUCATIONAL PLATFORM*

Author:
Panagiotis Anastasiou

Thesis Committee:
Prof. Apostolos DOLLAS, THESIS SUPERVISOR
Assoc. Prof. Sotirios IOANNIDIS
Dr. Evaripidis SOTIRIADIS



A thesis submitted in fulfillment of the requirements
for the Diploma of Electrical and Computer Engineer
in the

School of Electrical and Computer Engineering
Microprocessor and Hardware Laboratory

14-Sep-23

TECHNICAL UNIVERSITY OF CRETE

ABSTRACT

School of Electrical and Computer Engineering

Electrical and Computer Engineer

EMBEDDED SYSTEM FOR MULTIPLE-USER ACCESS TO MICROPROCESSOR- BASED EDUCATIONAL PLATFORM

by Panagiotis Anastasiou

The aim of this thesis is to develop and assess an embedded system, intended for use in an educational environment. Specifically, the platform accommodates a user interface, capable of exchanging information with a host server allowing the remote access of multiple users to an embedded system, supporting upload of files and commands to various microcontrollers through a master microcontroller, while providing real-time visual feedback through a webcam. A server was developed using Python to communicate with a database and was hosted in the laboratories within the Technical University of Crete. For the experiment, an STK500 development board was used to program an ATMEGA32A for the upload of the .bin files and commands to a pair of ATMEGA328P controllers and a pair of ATMEGA168P. Further, various operational tests were performed remotely using a windows computer. Tests deemed the overall operation of the system successful, including the log-in and reservation slot feature of the user interface. Further, a successful upload of .bin files was noted, with a stable real-time video feed from the webcam. Tests also detected some instabilities within the log-in and reservation phase of the system, with unsuccessful login and undesired termination of the application within the user system.

ABSTRACT

School of Electrical and Computer Engineering

Electrical and Computer Engineer

EMBEDDED SYSTEM FOR MULTIPLE-USER ACCESS TO MICROPROCESSOR- BASED EDUCATIONAL PLATFORM

by Panagiotis Anastasiou

Στόχος της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη και αξιολόγηση ενός ενσωματωμένου συστήματος, που προορίζεται για χρήση σε εκπαιδευτικό περιβάλλον. Συγκεκριμένα, η πλατφόρμα φιλοξενεί μια διεπαφή χρήστη, ικανή να ανταλλάσσει πληροφορίες με έναν διακομιστή που επιτρέπει την απομακρυσμένη πρόσβαση πολλών χρηστών σε ένα ενσωματωμένο σύστημα, υποστηρίζοντας μεταφόρτωση αρχείων και εντολών σε διάφορους μικροελεγκτές μέσω ενός κύριου μικροελεγκτή, ενώ παρέχει οπτική ανατροφοδότηση σε πραγματικό χρόνο μέσω κάμερας. Αναπτύχθηκε ένας διακομιστής με χρήση Python για την επικοινωνία με μια βάση δεδομένων και φιλοξενήθηκε στα εργαστήρια του Πολυτεχνείου Κρήτης. Για το πείραμα, χρησιμοποιήθηκε μια πλακέτα ανάπτυξης STK500 για τον προγραμματισμό ενός ATMEGA32A για τη μεταφόρτωση των αρχείων .bin και των εντολών σε ένα ζεύγος ελεγκτών ATMEGA328P και σε ένα ζεύγος ATMEGA168P. Περαιτέρω, διάφορες λειτουργικές δοκιμές πραγματοποιήθηκαν εξ αποστάσεως χρησιμοποιώντας έναν υπολογιστή με Windows. Οι δοκιμές έκριναν ότι η συνολική λειτουργία του συστήματος ήταν επιτυχής, συμπεριλαμβανομένης της δυνατότητας ταυτοποιήσεις και κράτησης μικροελεγκτή από τη διεπαφή χρήστη. Επιπλέον, σημειώθηκε επιτυχής μεταφόρτωση αρχείων .bin, με σταθερή ροή βίντεο σε πραγματικό χρόνο από την κάμερα. Οι δοκιμές εντόπισαν επίσης ορισμένες αστάθειες στη φάση σύνδεσης και κράτησης του συστήματος, με ανεπιτυχή σύνδεση και ανεπιθύμητο τερματισμό της εφαρμογής εντός του συστήματος χρήστη.

Table of Contents

ABSTRACT	3
ABSTRACT	5
Table of Contents	6
Table of Figures	8
List of Tables	10
List of Abbreviations	11
Chapter 1 Introduction	1
1.1 Problem Statement	1
1.2 Objectives	1
1.3 Scope of the Thesis	2
1.4 Thesis Outline	2
Chapter 2 Related Work	4
2.1 Integrated System for Logic Design exercises	4
2.2 FPGA e-Lab	5
2.3 ATMEL AVR ICSP based on Wi-Fi and ZigBee	7
2.4 Development and Application of Remote Laboratory for Embedded Systems Design	8
2.5 Web-based User interface with RPDs interaction	9
2.6 Remote labs based on the ESP32 and NOD-red	10
2.7 CAD Tools	11
2.8 WebLab-PLD	11
2.9 An Embedded System for Remote Access of Microcontroller for University Laboratory Exercises	12
Chapter 3 Methodology	14
3.1 Goal and Objectives	14
3.1.1 User login and reservation slots.....	15
3.1.2 File Transmission	19
3.1.3 Real-Time visual feedback	23
3.1.4 Multiple users	25
3.1.5 Flexible	26
3.1.6 Expansion Possibilities.....	26
3.1.7 Scalable.....	27
3.2 Apparatus	28
3.2.1 Platforms Connection layout	28
3.2.2 Hardware Components	29
3.2.3 Software Components.....	34
3.3 Assumptions and Limitations	39
Chapter 4 User and Institution System Perspective	41

4.1	User Perspective.....	41
4.1.1	Installation instructions for Ubuntu/Windows	41
4.1.2	Using the application	41
4.1.3	Writing programs	43
4.2	Engineer's Perspective.....	47
4.2.1	setupdb.py	47
4.2.2	addUser.py	47
4.2.3	removeUser.py	48
4.2.4	serverv2.py	48
4.2.5	CameraStreamWebSockets.ino	49
4.2.6	WebSocketServer.ino	51
4.2.7	Atmega32Av2.atsln	52
4.2.8	Main.py	58
Chapter 5 System Verification and Evaluation		60
5.1	Testing and validation procedures	60
5.1.1	Initial Development and Component Testing	60
5.1.2	Login System and Database Testing.....	61
5.1.3	File Transmission Testing	64
5.1.4	Microcontroller Programming Testing.....	65
5.1.5	Real-Time Camera Access Testing	68
5.1.6	Multi-Microcontroller Integration Testing	69
5.2	System Evaluation	70
Chapter 6 Conclusion and Future Work.....		72
6.1	Conclusion	72
6.2	Future work	73
Chapter 7 Appendix		74
7.1	Hardware Schematics.....	74
7.2	Example Applications.....	75
7.3	Parts List	78
References.....		79

Table of Figures

Figure 2.1 - Interconnection between the distinct elements of the platform - Source [1]	4
Figure 2.2 - Schematic diagram of a remote interface system – Source [2]	5
Figure 2.3 - The GUI showing the FPGA development board – Source [2]	6
Figure 2.4 - System Block Diagram of Wireless Programmer – Source [3]	7
Figure 2.5 - Overview of RELDES – Source [4]	8
Figure 2.6 - Communication flow of RPD system – Source [5]	9
Figure 2.7 - Software Architecture – Source [6]	10
Figure 2.8 - Last Iteration Software Architecture – Source [7]	11
Figure 2.9 - Schematic diagram of the Platform – Source [8]	12
Figure 2.10 - Main Web Page of Remote Lab – Source [8]	13
Figure 3.1 - Overall Layout	14
Figure 3.2 - Server Communication Diagram	16
Figure 3.3 - Database Fields	17
Figure 3.4 - Login Stage	17
Figure 3.5 - Reservation Stage	18
Figure 3.6 - Unavailable Slot Stage	19
Figure 3.7 - Send File/Command flowchart	20
Figure 3.8 - ESP File Receive Flowchart	21
Figure 3.9 - ATMEGA32A Receive File	22
Figure 3.10 - ESP32-CAM Flowchart	24
Figure 3.11 - Platforms Connection Layout	28
Figure 3.12 - ATmega328P and ATmega168P Diagram	30
Figure 3.13 - ATMEGA32A Diagram	31
Figure 3.14 - ESP32-S Diagram	32
Figure 3.15 - ESP32-CAM Diagram	33
Figure 4.1 - Available file to be Uploaded	42
Figure 4.2 - Control Menu	42
Figure 4.3: New Project	43
Figure 4.4: Device Selection	44
Figure 4.5: Programmer Selection	44
Figure 4.6: HEX File Location	45
Figure 4.7 - Adding / Removing User From DB	47
Figure 4.8 - ESP32-CAM Flowchart	49
Figure 4.9 - Wi-Fi Module Flowchart	51

Figure 4.10 - AVR910 Progaming Flowchart	54
Figure 4.11 – ATMEGA32A File Receive	56
Figure 5.1 - Database Records	60
Figure 5.2 - Login Portal.....	61
Figure 5.3 - Reservation Panel.....	62
Figure 5.4 - Database with Populated Fields.....	62
Figure 5.5 - Next Available Slot Panel	62
Figure 5.6 - Removed Reservation.....	63
Figure 5.7 – Adding & Removing Users.....	63
Figure 5.8 - Control Panel Without A Selected File.....	64
Figure 5.9 - Control Panel With A Selected File.....	64
Figure 5.10 - Blink File LEDs OFF.....	65
Figure 5.11 - Blink File LEDs ON.....	66
Figure 5.12 - Buttons Test File	66
Figure 5.13 - Buttons File Button1 Pressed.....	67
Figure 5.14 - Buttons File Button2 Pressed.....	67
Figure 5.15 - All Users Upload Different Files.....	69
Figure 7.1 - Hardware Schematics	74
Figure 7.2 - Blink.....	75
Figure 7.3 - Buttons	75

List of Tables

Table 1 - Database Fields and Types	47
Table 2 - Utilized Ports	57
Table 3 - Testing Login Entries	61

List of Abbreviations

BIN	Binary
CAD	Computer Aided Design
CPU	Central Processing Unit
DDRC	Data Direction Register for Port C
DDR D	Data Direction Register for Port D
EEPROM	Electrically Erasable Programmable Read-Only Memory
FOTA	Firmware Over-the-Air
FPGA	Field-Programmable Gate Array
FTDI	Future Technology Devices International
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
HEX	Hexadecimal
HTTP	Hypertext Transfer Protocol
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISP	In-System Programming
ISR	Interrupt Service Routine
JSON	JavaScript Object Notation
LED	Light-Emitting Diode
LMS	Learning Management System
MHL	Microprocessor and Hardware Laboratory
MISO	Master In, Slave Out
MOSI	Master Out, Slave In
MQTT	Message Queuing Telemetry Transport
PC	Personal Computer
PLD	Programmable Logic Device
RELDES	Remote Laboratory for Embedded Systems Design
RPD	Remote Programing Device
RS	Recommended Standard
RXC	Receive Complete
SCK	Serial Clock
SPCR	Serial Port Control Register
SPDR	Serial Port Data Register
SPI	Serial Peripheral Interface
SPIF	Serial Peripheral Interface Flag
SPSR	Serial Port Status Register
SRAM	Static Random-Access Memory
SS	Slave Select
SW	Switch
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UCSRA	Universal Serial Control Register A
UDR	Universal Receive Data Register

UDRE	Universal Receive Data Register Empty
UI	User Interface
URL	Uniform Resource Locator
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VCP	Virtual COM Port
VGA	Video Graphics Array
VPN	Virtual Private Network
WPA	Wi-Fi Protected Access

Chapter 1 Introduction

Many institutes aim to increase student engagement with the introduction of new educational platforms providing a better hands-on learning experience. This thesis presents the design and implementation of a remote access Embedded System for Multiple-User Access to a Microprocessor-Based Educational Platform. The platform aims to enable multiple users to remotely access a set of microcontrollers. Enabling the user to program, control, and monitor a microcontroller in real-time. Such a platform could be easily accessible from anyone at any time and can reduce the costs of providing one development board to each student who desires to develop and test its projects.

1.1 Problem Statement

A lot of publicly available tools like Tinkercad circuits that allow users to create and simulate electronic circuits do not meet the requirements of institutes to integrate them into their course's curriculum. Its supported programming languages are Blockly and Python whereas, institutes mostly prefer to use low-level programming languages like C and assembly enabling better utilization of their hardware. This absence of online presence in many cases where students are not able to attend at their institutions limits their interaction with microcontrollers. Resulting in an absence of understanding of embedded systems and their capabilities.

1.2 Objectives

The primary objectives are:

- The implementation of a user-friendly interface that allows users to easily navigate it and interact with their microcontrollers within the educational platform.
- Support real-time file uploading to the system enabling multiple users to remotely program and control their microcontrollers.

- Integrated a camera stream from the target microcontroller into the User Interface to enhance the overall experience for the end user by providing them with a visualization of the results in real-time.
- Ensuring the stability and responsiveness of the application as students will need to be able to depend on the system's proper functionality under various conditions.

1.3 Scope of the Thesis

This thesis focuses on developing a user-friendly User Interface that allows multiple users to access remotely a multi-microcontroller-based embedded system that supports real-time monitoring through an integrated camera. Whilst, enabling users to program their assigned microcontroller and interact with them using software buttons. The scope of the thesis also includes the selection of suitable hardware and software components.

While the platform attempts to offer each of its users a cohesive experience, certain limitations should be recognized. Some of them may include hardware compatibility, potential security risks, the processing capabilities of the selected microcontrollers, and any network stability issues.

1.4 Thesis Outline

This thesis is organized into six chapters:

Chapter 2: Includes related work regarding multi-microcontroller systems that support multiuser access with real-time monitoring and control.

Chapter 3: Details the methodology followed for the development of the proposed platform, and provides the selection criteria for hardware and software components.

Chapter 4: Provides information on how the user can use the proposed platform and expands on the software that fulfills the platform requirements.

Chapter 5: Focuses on describing the testing and verification procedures that were followed to evaluate and ensure the proper functionality of the developed platform.

Chapter 6: Concludes the thesis and expands on future work to improve the proposed platform.

Chapter 2 Related Work

Affected by the pandemic the demand for multi-microcontroller platforms in education with real-time control and monitoring was increased. To accommodate the increased demand new embedded systems are required for a more inclusive and interactive learning experience. These systems need to provide access to different types of equipment that cannot be used or borrowed by the students outside the laboratory. Providing access to that type of equipment could potentially increase active engagement by students in various courses due to the convenience of using the otherwise restricted equipment in their own time. Allowing them to explore the capabilities of the equipment in their free time. Further, students can utilize such systems to recreate previous laboratory experiments and learn from any potential issues they might have encountered.

This chapter researches related work on similar projects and provides insides while establishing a foundation for the proposed platform. By exploring previous implementations various key aspects can be identified and deployed in this research to effectively achieve set goals and objectives.

2.1 Integrated System for Logic Design exercises

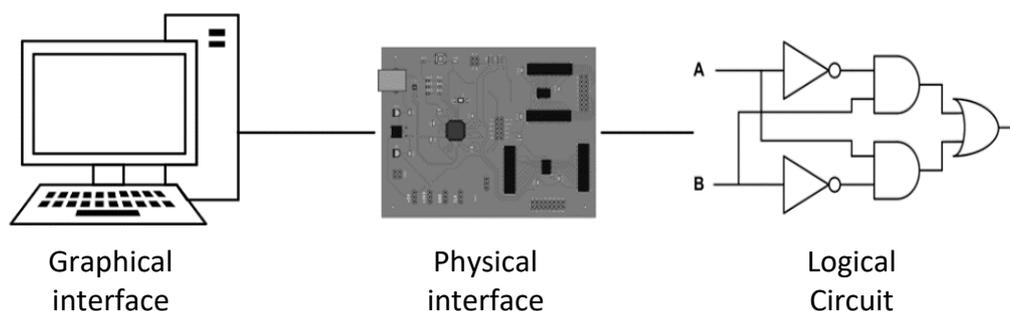


Figure 2.1 - Interconnection between the distinct elements of the platform - Source [1]

In the research “Design and Implementation of an Integrated System for Logic Design exercises” by Lykos Emmanouil [1]. A platform was developed to accommodate the needs of

the "Logic Design" lab-course and it consists of two main components as presented in Figure 2.1. The physical interface "MHL Development Board" is used to control the physical circuits, this system was developed using an STM32F072B-DISCKO platform which was chosen for its low cost, various features, and USB support. The other part is the graphical interface the "MHL Development Board Control" that allows users to control the I/O to verify the systems operations and to send a new clock pulse. The communication between the graphical interface and the physical interface is established through a VCP (Virtual Com Port) and the STM USB Stack. Overall, this implementation offered real-time control and effective support for experimental exercises in the "Logic Design" course.

2.2 FPGA e-Lab

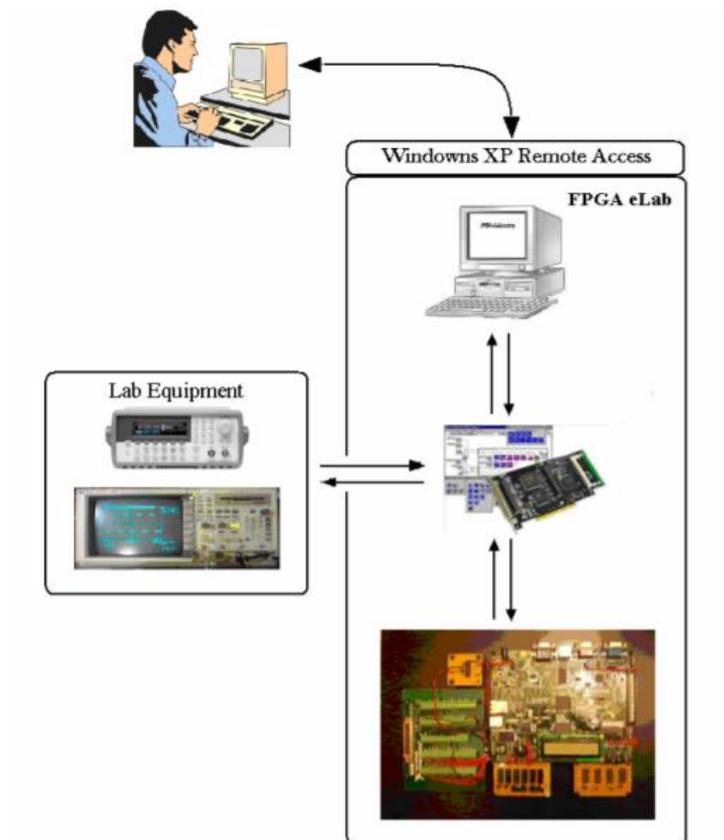


Figure 2.2 - Schematic diagram of a remote interface system – Source [2]

Similarly, another research had an approach to implement a remote access technique for designing and testing FPGA-based experiments in a laboratory setting. The proposed system

by Reza Hashemian and Jason Riddley [2] utilizes Microsoft XP Remote Desktop to provide users with complete control over the laboratory PC, enabling remote access (Figure 2.2).



Figure 2.3 - The GUI showing the FPGA development board – Source [2]

The FPGA development board, specifically the Xilinx Spartan-3E Starter Kit, serves as the target for hardware interfacing. As illustrated in Figure 2.3, the control of the development board is facilitated through the LabView GUI and the NI PCI-6025E/CB-100 data acquisition hardware. The system follows 7 stages for its digital design process, allowing for programming, verification, and modification. Additionally, the system demonstrates its flexibility by supporting additional equipment through the interface. This remote access technique offers numerous benefits and enhances FPGA-based experimentation in educational settings.

2.3 ATMEL AVR ICSP based on Wi-Fi and ZigBee

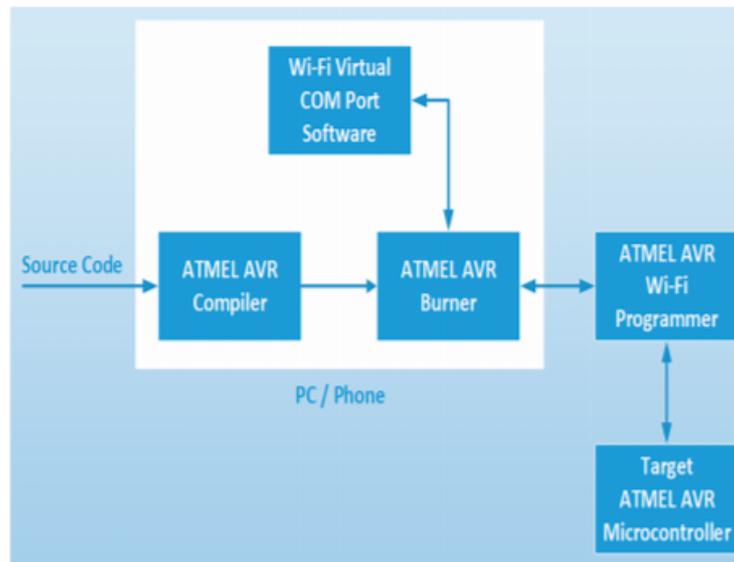


Figure 2.4 - System Block Diagram of Wireless Programmer – Source [3]

In the research of “Wireless ATMELE AVR In-Circuit Serial Programmer based on Wi-Fi and ZigBee” [3] various implementations to allow remote programming in microcontrollers have been reviewed to address the limitations of traditional programming methods. This led to the introduction of FOTA which is a method that is used to remotely update the firmware of embedded devices to keep them up-to-date and bug-free extending their lifetime. The paper presents an ICSP based on an AVR microcontroller, which when combined with a Wi-Fi or ZigBee module, it enables the remote programming of its target microcontroller. The presented wireless programmer can read, write, verify the EEPROM and Flash memories, and modify the fuse bits. The system has three components Figure 2.4 a target microcontroller, a Wi-Fi programmer, and a PC/Phone subsystem which includes compiler software, burner software, and a Wi-Fi Virtual COM Port Converter software.

The system was tested and validated with both the Wi-Fi and ZigBee modules which are used to handle the wireless communication between the PC and the remote programmer.

Overall, the paper focuses on the development of a wireless ATMEEL AVR ICSP based on Wi-Fi and ZigBee. Highlighting its potential in enabling remote programming and debugging of microcontrollers. Concluding that the system can be utilized in educational and academic projects as well as in industrial systems.

2.4 Development and Application of Remote Laboratory for Embedded Systems Design

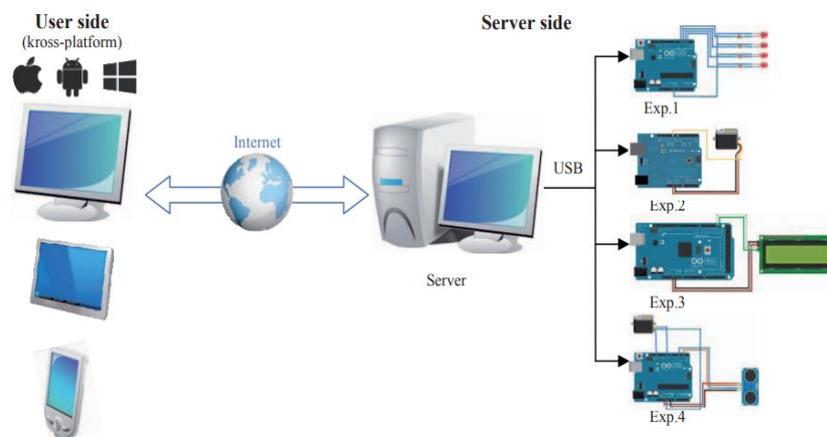


Figure 2.5 - Overview of RELDES – Source [4]

Similarly, another research developed RELDES (Remote Laboratory for Embedded Systems Design) [4]. The proposed approach aims to improve the current state of the art in embedded systems design, allowing to accelerate stages of hardware-software integration and testing. This system uses a server to provide access for testing and programming, while it also enables visual feedback through a camera feed. The server can compile a code that is received from a client and return the compilation results. Consequently, it enables code upload upon microcontroller availability or applies a client queue for the in-use hardware.

2.5 Web-based User interface with RPDs interaction

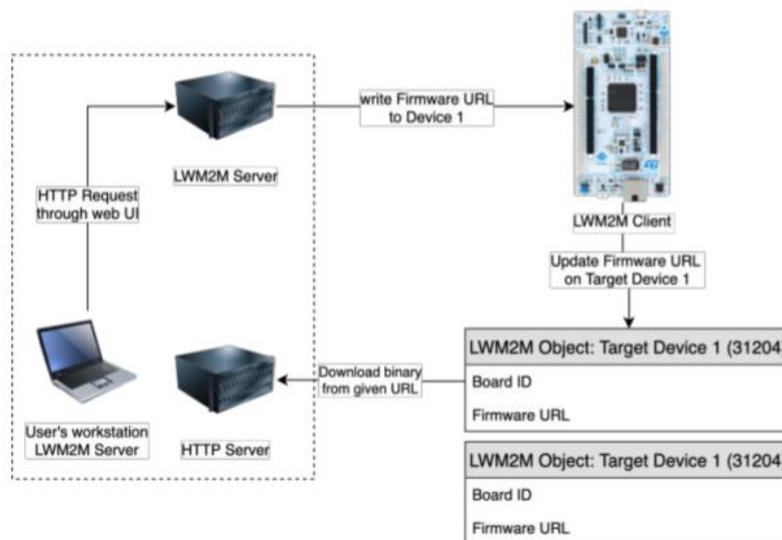


Figure 2.6 - Communication flow of RPD system – Source [5]

In the research of “Remote Programming and Reconfiguration System for Embedded Devices” [5] the approach of a remote management add-on system, specifically resource-constrained wireless sensors and IoT nodes was presented for embedded devices. The system allows for remote management, operation monitoring, and firmware updates and can be easily customized for various microcontrollers or other programmable logic chips (Figure 2.6). Providing a Web-based UI that enables interaction with the connected RPDs, also allows the flexibility for modification to allow remote laboratory access for educational purposes. A proof-of-concept prototype has been successfully developed and evaluated using commonly available hardware. The implementation utilizes standard protocols and interfaces, providing flexibility and versatility. The proposed system addresses the challenge of remote management and firmware updates for embedded devices in IoT systems, offering a practical and adaptable solution.

2.6 Remote labs based on the ESP32 and NOD-red

The research by Abekiri, N. [6] aims to address the challenges faced by educational institutions, during the COVID-19 pandemic, where conventional teaching and laboratory work were restricted or disrupted. To accomplish this, the researchers utilized LMS Chamilion and Node-RED to create an interactive environment that allows students and teachers to access the laboratories remotely via the MQTT protocol. Connecting an ESP-32 to the instruments and I/O devices, the platform enabled students to control the targeted systems and acquire real-time data. Additionally, with the ESP-32 CAM, a live video was provided from the lab.

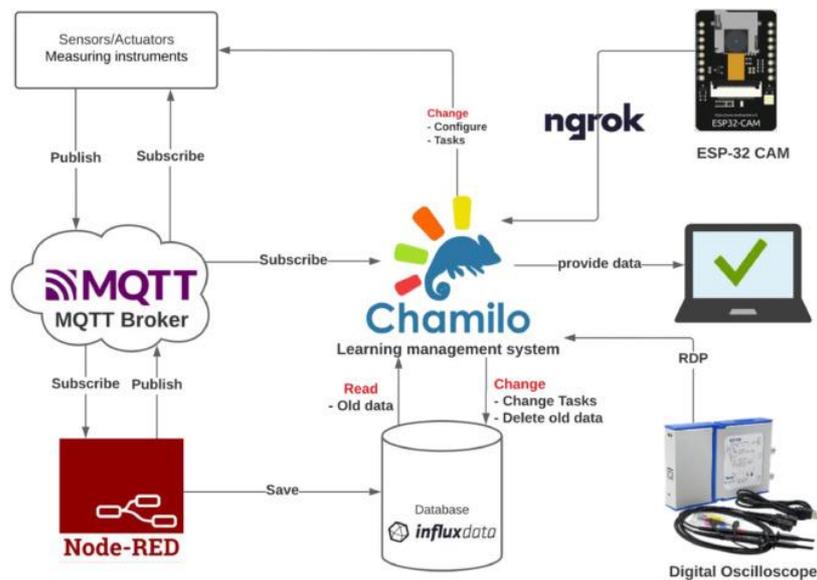


Figure 2.7 - Software Architecture – Source [6]

The software was built on Node.js offering input-output nodes to allow scenarios to be represented graphically (Figure 2.7). Users can further customize these scenarios using the included JavaScript editor. The Node-RED, in conjunction with Mosquitto as a broker, facilitates the transformation of ESP32 messages to JSON format through the MQTT protocol. This ensures seamless communication between the remote access platform and the laboratory instruments.

2.7 CAD Tools

To utilize the previously mentioned platforms, a suite of different CAD tools is needed to aid in the development and compilation of the respective codes. Various publicly available platforms can support the unique demands of logic design, FPGA development, and microcontroller programming. Tools such as Logisim and Digital Works are user-friendly solutions for logical design, providing visual circuit creation and simulation. For FPGA design, Quartus Prime and Vivado offer advanced features for design, simulation, and programming. Microcontroller programming can be accomplished utilizing environments like Arduino IDE, Atmel Studio, and MPLAB X IDE, each designed to accommodate different processor architectures and complexities providing an integrated environment for writing, compiling, and uploading code to microcontrollers.

2.8 WebLab-PLD

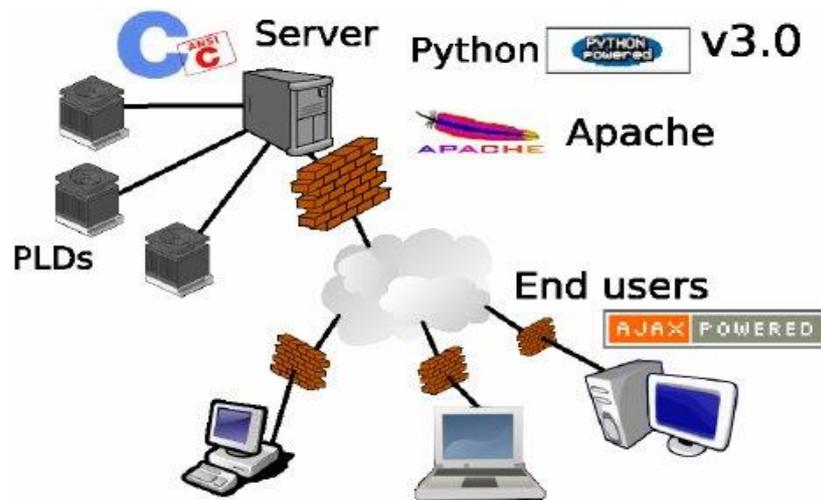


Figure 2.8 - Last Iteration Software Architecture – Source [7]

In the research “Evolving towards better architectures for remote laboratories: a practical case” [7] the system utilizes WebLabs for students to access remotely and control hardware using TCP/IP and monitor the progress through a web camera or any other means. Further, it explores the different implementations of web labs before presenting the WebLab-PLD. This system is made to control PLD programmable devices allowing students to access and control the equipment through a website that allows them to upload a Jedec file, change inputs, and

see the results over a web camera. The research goes through multiple iterations of this system before presenting the last iteration showing the process and evolution from 1 user, 1 server, and 1 programmable device to multiple users, 1 server, and multiple programmable devices Figure 2.8.

The last iteration replaced the RS-232 cable which was used for communication to the server with a micro server to allow communications through HTTP using the XLM encoding standard, transforming each programmable device to a networked node, and enabling communication between them without using the server.

2.9 An Embedded System for Remote Access of Microcontroller for University Laboratory Exercises

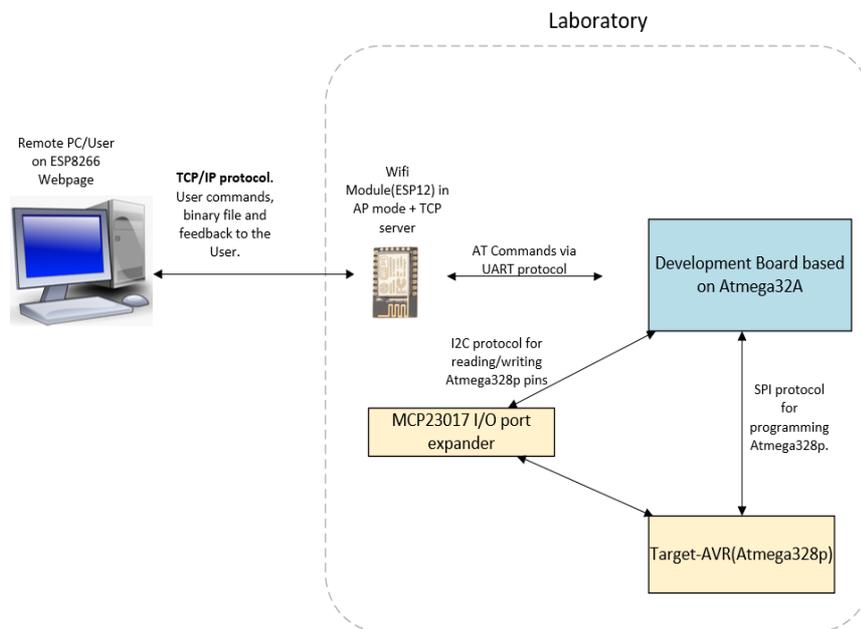


Figure 2.9 - Schematic diagram of the Platform – Source [8]

In this thesis by Athanasios Kalliontzis at the School of ECE, Technical University of Crete [8] an embedded system for remote access was designed and implemented. The remote access was achieved through a website that is stored on the ATmega32A and hosted by the

ESP12-E module. The platform aims to allow students to remotely program a target microcontroller such as ATMEGA328P and receive graphical feedback that is achieved using an I/O port expander to read or set values for specific ports of the ATMEGA328P. The ATMEGA32A-PU uses a binary file and the AVR910-ISP protocol [9] to program the ATMEGA328P through the SPI protocol (Figure 2.9).

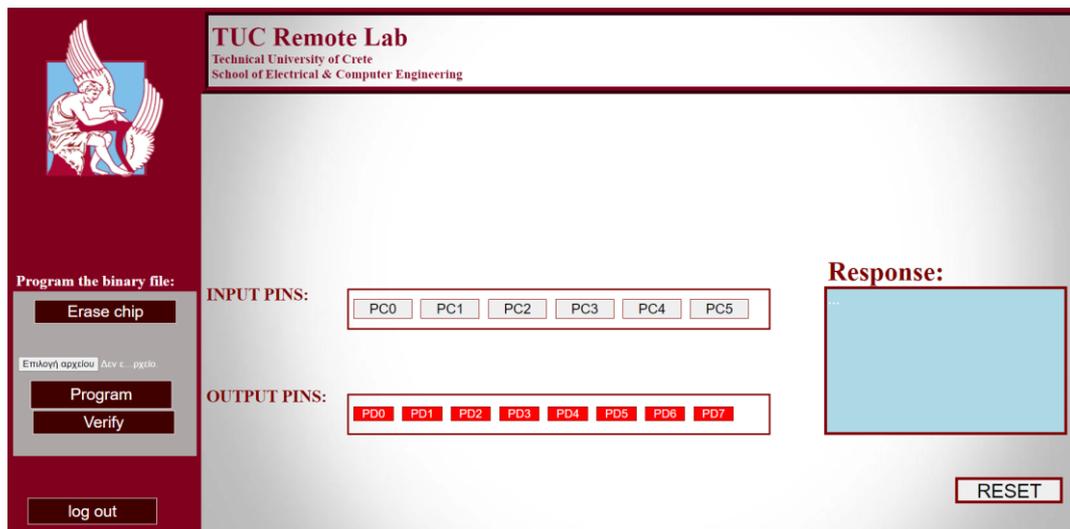


Figure 2.10 - Main Web Page of Remote Lab – Source [8]

To use the system a user must log in to the web application (Figure 2.10). The web application offers various functionalities such as:

- code upload for microcontroller programming
- Chip erase
- program verification
- microcontroller reset
- Adjustment of input values within the application
- Display of output values for ports PD0-PD7
- Activity/Action report

Chapter 3 Methodology

3.1 Goal and Objectives

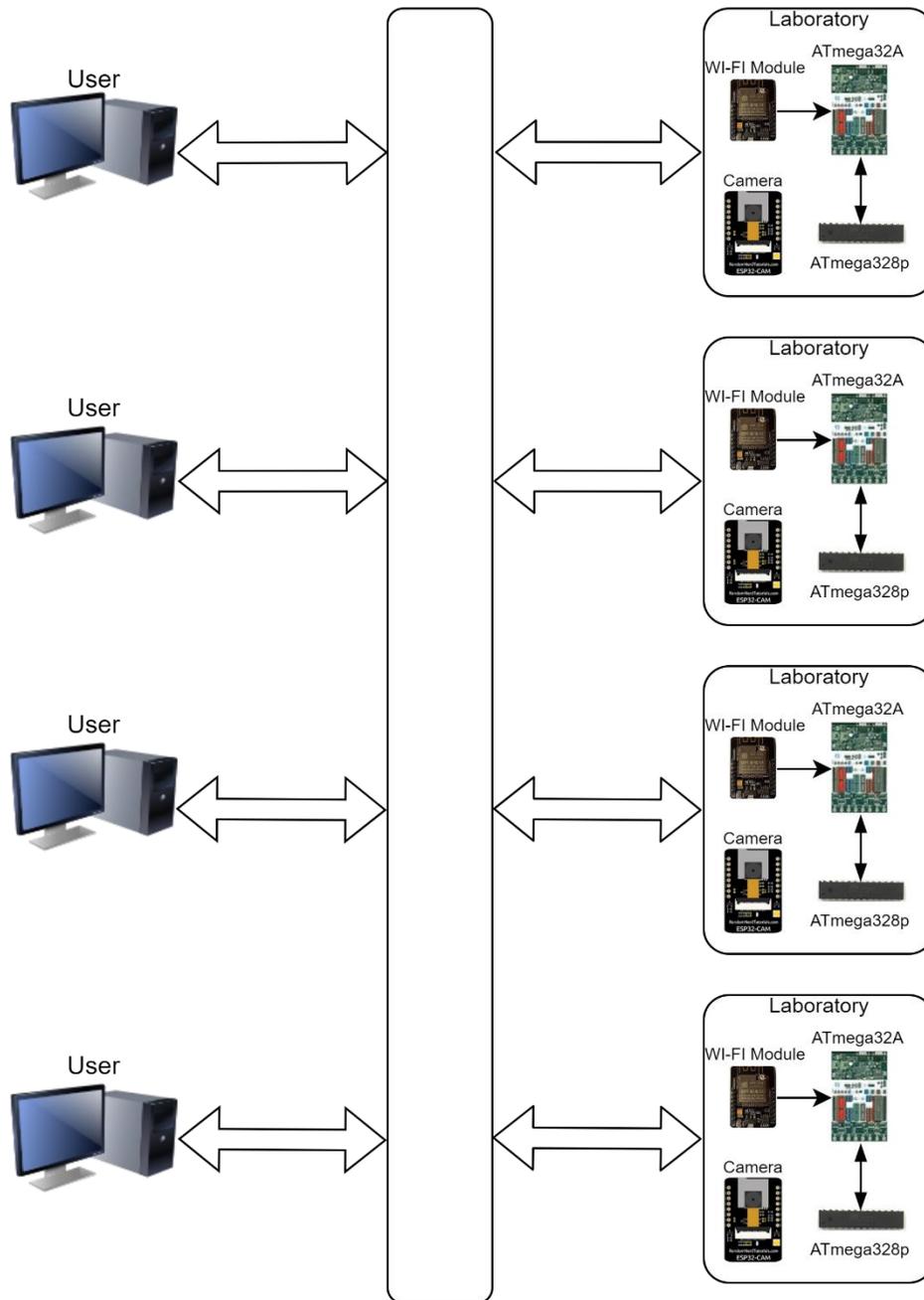


Figure 3.1 - Overall Layout

The goal of this thesis is to expand a previously developed to support multiple users and assess the technology and viability of the proposed system toward its integration into an educational environment. Developing an educational platform requires each aspect to support the desired functionality. To ensure that the platform is capable of handling multiple users facilitates remote access, and enables interaction through uploading files or commands to a microcontroller and providing real-time visual feedback.

The block in Figure 3.1 represents the necessary software utilized in the platform as described in the sections below. To transition the existing single-user system into a multiple-user setup, several changes are required. These include the implementation of a user login system to manage multiple users' access and a reliable file transmission process must be developed to facilitate seamless interaction between users and microcontrollers. Also, ensuring real-time video feed delivery to connected users is a key aspect of the platform. Further, expanding on aspects such as the system's flexibility, expansibility, and scalability is vital for a better understating of the platform's potential applications.

To ensure a common understanding of the terminology used throughout this thesis, the term "system" refers to the hardware and software components. Specifically, the system includes the ESP32-S Wi-Fi module, ATMEGA32A, ATMEGA328P, ATMEGA168P microcontrollers, and the ESP32-CAM camera module. Whilst, the term "platform" represents the complete ecosystem. Including the user interface (UI), the server, and the database infrastructure along, with the "system".

3.1.1 User login and reservation slots

From the perspective of the end user, the student should be able to obtain access to the system by logging in with some form of credentials, which could vary from his personal login information. Further, the access obtained should be limited in time, so that it will allow other users to use the platform afterward.

Necessary for the achievement of these goals, is a creation of a user interface in which the user will be able to log in. To establish that the input credentials are valid, the system should

have a database, which will be used to store and cross-match the input data with the stored valid credentials. Upon verification, it should then allow the user to choose various fixed timeframes in which access will be granted. Further, the system should be able to revoke the access from a user once the set time-period expires. Lastly, to view and edit the database, authorized personnel should have a way to access the database for view and modification. To ensure a basic level of security, the communication between UI and the database should be encrypted.

The communication between the login system and the database can be established through two possible methods: either directly through the application or a server. In the direct approach, the application itself connects to the database to fetch and verify user login credentials. Alternatively, in the server-based approach, the login system communicates with a server that acts as an intermediary between the application and the database. Though the low scale of the project would require fewer procedures and equipment to operate in the application-based approach, the server-based approach is chosen as it is in line with the pre-existing technologies of educational institutions, and their scalability.

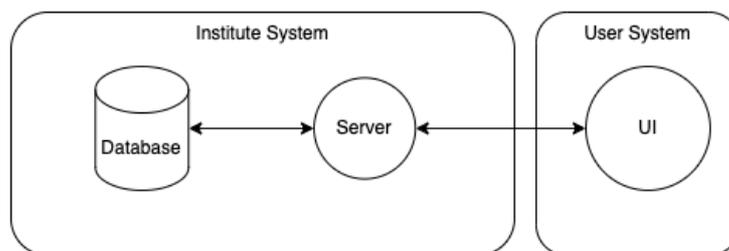


Figure 3.2 - Server Communication Diagram

During the setup procedure (section 4.2.1) a database is created on the institution server, as well as an intermediary portal which will establish the communication between the server database and the UI which will be installed on the user's system (Figure 3.2).

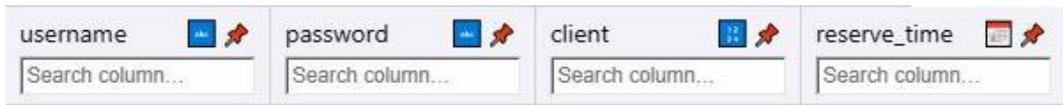


Figure 3.3 - Database Fields

The database consists of a single table (Figure 3.3), which can further be altered to withhold additional information according to the necessities of each institute. For this experiment, it holds the essential operating parameters, such as the username, password, client, and reservation time. The program holds for each available slot the client which reserved the given module and the reservation time. More information about the database's operation can be found in section 4.2.

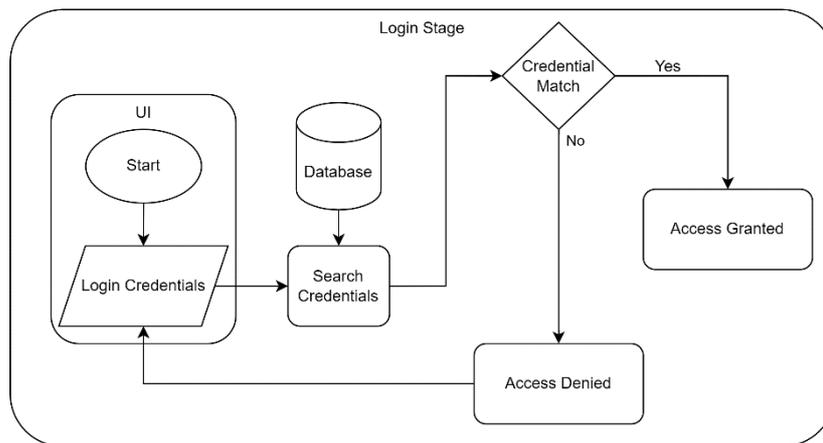


Figure 3.4 - Login Stage

The flow chart in (Figure 3.4) above, illustrates the operating procedure for the login stage. In this stage, the user inputs their credentials to the system. This packet is then sent through the server into a query, which looks for any records containing the same data.

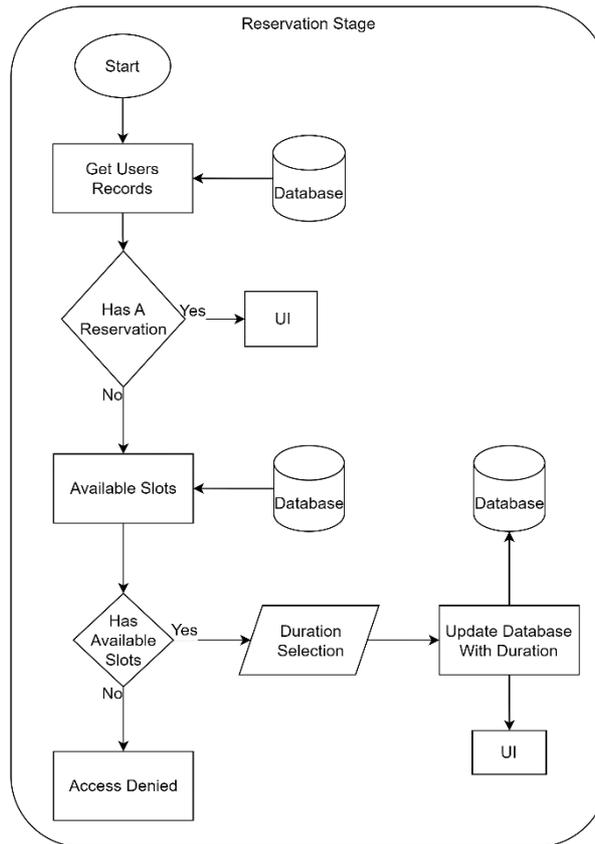


Figure 3.5 - Reservation Stage

Initially, the system checks if the credentials of the user are already found in any of the reservation slots. In this event, the system should allow the user to continue using the system without denying access, or allowing for a second reservation to be made. If no match is found the system, then checks if any of the four slots are available for reservation. If there is an available slot, the user is prompted to the reservation stage, otherwise, access is denied.

Once prompted to the reservation stage, the user can select from a dropdown list a duration to complete their reservation. This information is then used to update the database's first available slot with the user's credentials and the reservation time.

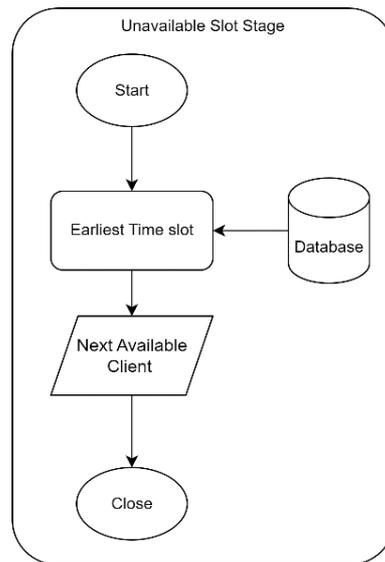


Figure 3.6 - Unavailable Slot Stage

The flow chart in (Figure 3.6) above, illustrates the operating procedure when there are no available time slots. A query looks for the closest available time that a slot is obtainable and notifies the user.

3.1.2 File Transmission

The file transmission must happen reliably each time no matter the network conditions. Files should reach their destination without any data loss to ensure the system's seamless operation. Therefore, when evaluating available options for sending files it, is important to prioritize reliability. Responsiveness is also a consideration to maintain a smooth user experience.

To achieve a fast and reliable file transmission, the chosen method must have the capability to handle binary files, as that is the format used by the system. Additionally, file transmission should happen in chunks for example 256 bytes each time or as large as the system's buffer allows. Considering that storing files in the system may not be feasible due to the limitations of the microcontroller's memory. Further, the system must be limited to receiving one file at a time. Permitting more than one file at a time the system will require more buffers (memory) and processing power. Further, explanation in section 3.2.3.4.

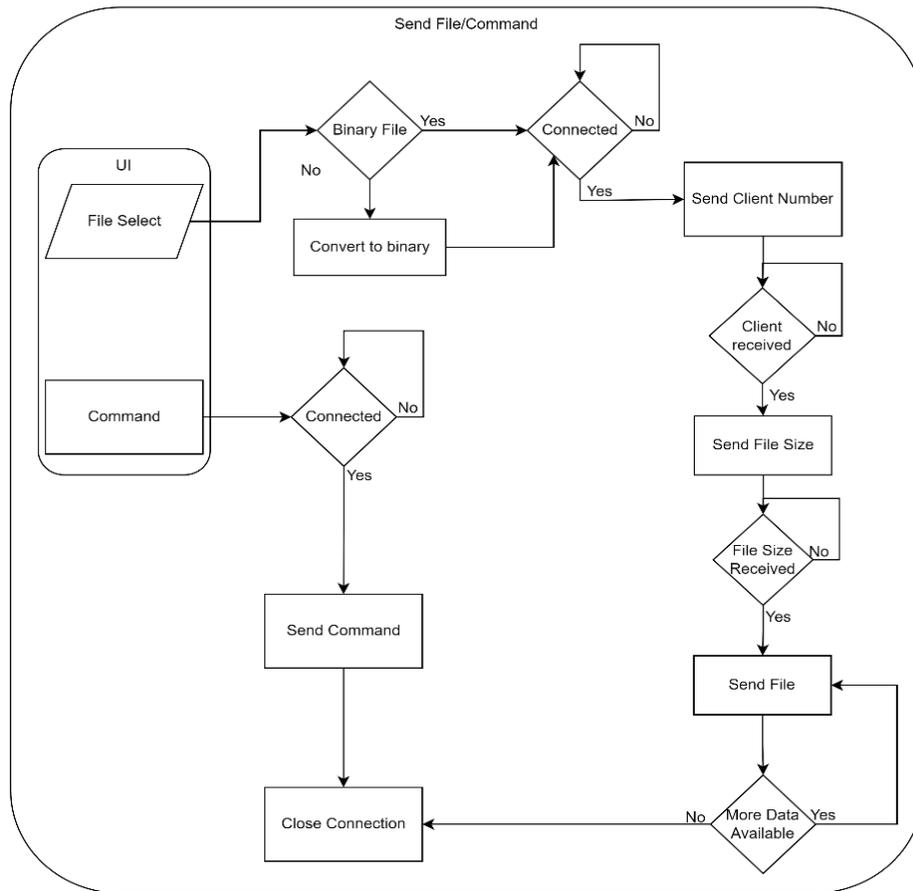


Figure 3.7 - Send File/Command flowchart

The file transition can be broken into 3 stages. In the first stage the end user can choose a file to be Uploaded. The selected file must be in HEX (Hexadecimal) or BIN (binary) format. After the user chooses a file, the application checks whether that file is in binary format. If not, the file gets converted and saved to the user’s PC for future use. For the upload to be successful the application must connect to ESP32-S (Wi-Fi module) which is responsible for receiving and passing data to the ATmega32A (Figure 3.7).

Based on the file transmission requirements, WebSockets proves to be a suitable method. WebSockets offer bidirectional communication ensuring efficient data exchange between the application and the Wi-Fi module. To ensure reliable transmission and avoid conflicts, the ESP32-S should only accept one connection at a time. If a user is currently uploading a file and another user attempts to upload simultaneously, the application will continuously attempt to

connect to the ESP32-S. The connection is established when the ongoing file transmission is completed.

When a connection is established, the application sends essential information to the master microcontroller (ATMEGA32A). This information consists of a client number, which is used to identify the file's destination along, with the file size to enable the microcontroller to identify the end of the file. Afterward, the application reads and sends the file in chunks, this allows the implementation of a simple flow control. Between each transmission, the application waits for a confirmation from the Wi-Fi module which is sent after the data are passed on to the ATMEGA32A.

This method allows the system to efficiently control memory usage but also to ensure that large files can be transmitted without overwhelming the master microcontroller. Once all the data are sent, the connection between the application and the Wi-Fi module is closed allowing other users to connect a interact with their microcontroller.

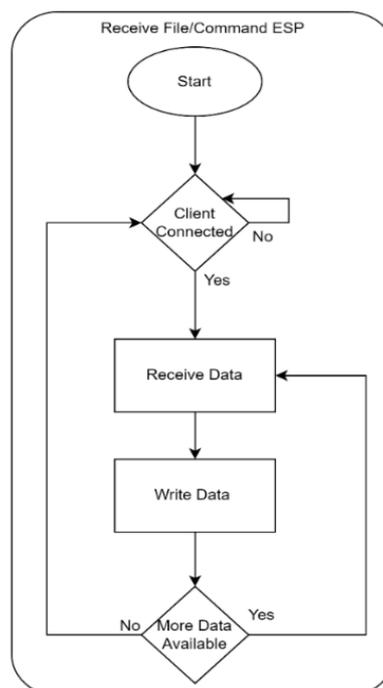


Figure 3.8 - ESP File Receive Flowchart

In the 2nd stage, after a connection is established the Wi-Fi module waits for data to arrive. When the data transmission is completed, a function is called to write that data over

to USART for the microcontroller to read. This process is repeated until the connected user closes the connection. Once a connection is closed the internet module can now establish a connection with other users (Figure 3.8).

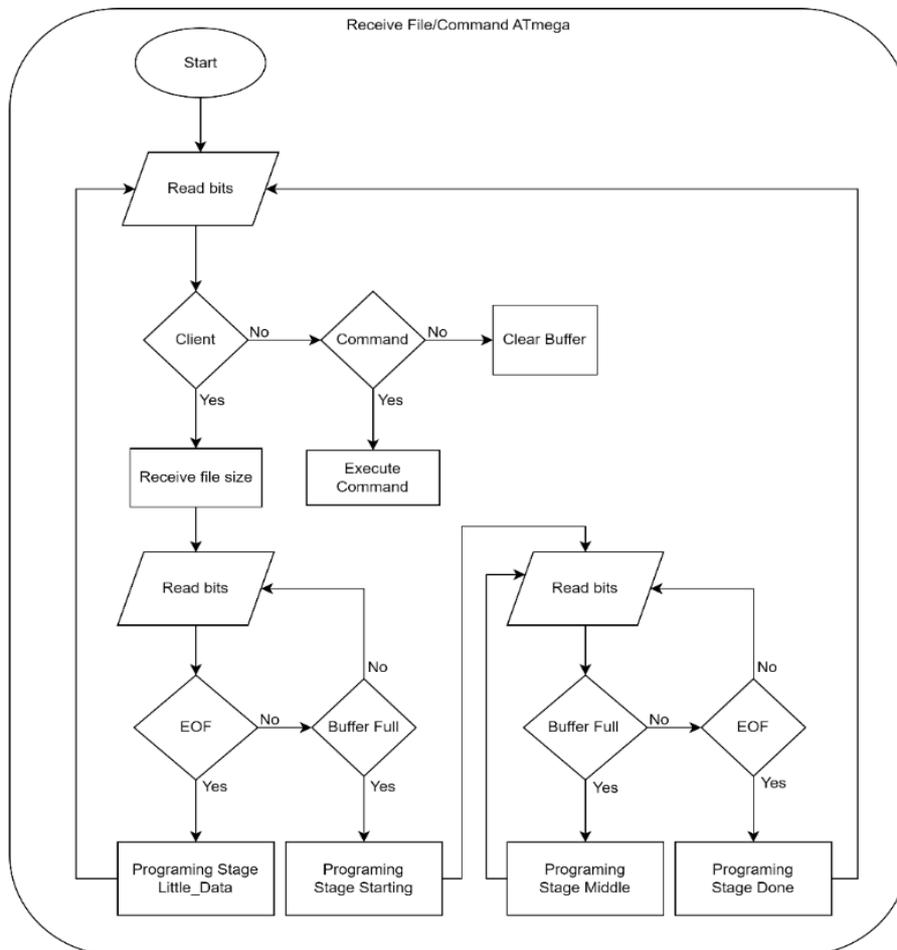


Figure 3.9 - ATMEGA32A Receive File

In the final stage, the master microcontroller (ATmega32A) reads the data serially from the Wi-Fi module and stores them in a buffer. To initiate the programming procedure or to perform any other desired function the received data are compared against some keywords (Figure 3.9). The master microcontroller before initializing the programming procedure expects to receive commands and client identifications only, any other data gets discarded. If a client identification is received then a file size is expected. When the file size is received the programming can start. Since the programming happens in pages the microcontroller waits

for the buffer to fill or reach the end of the file. To successfully, program a slave microcontroller (ATMEGA328P or ATMEGA168P) 4 stages are used. Further, explanation in section 4.2.7.4.

3.1.3 Real-Time visual feedback

Providing users with real-time visual feedback is essential for the feasibility of the proposed platform. Users should experience minimal latency between the camera feed and its display on the user interface and minimal frame loss. The quality of the stream needs to be good enough to differentiate the LEDs. Further, the stream should require a minimal amount of bandwidth to allow the application to be usable in case of an inconsistent network connection.

There are several possible ways to accomplish real-time video feedback. One way is for the ESP32-CAM to host a website where the stream will be posted. Then the application would connect to the website capture the stream and display it in the UI. However, this method requires a web server capable of handling multiple connections, which might require more processing power than what is available on the ESP32-CAM.

Another feasible approach is to broadcast the frames from the Camera module to the application utilizing the WebSockets communication protocol. The frames that way will be broadcast in real-time and the application can process the frame and update the UI with the new frames instantly. The latter approach is preferred due to its simplicity, efficiency, and direct connection between the ESP32-CAM and the application. This ensures that the video stream reaches the user interface with minimal latency.

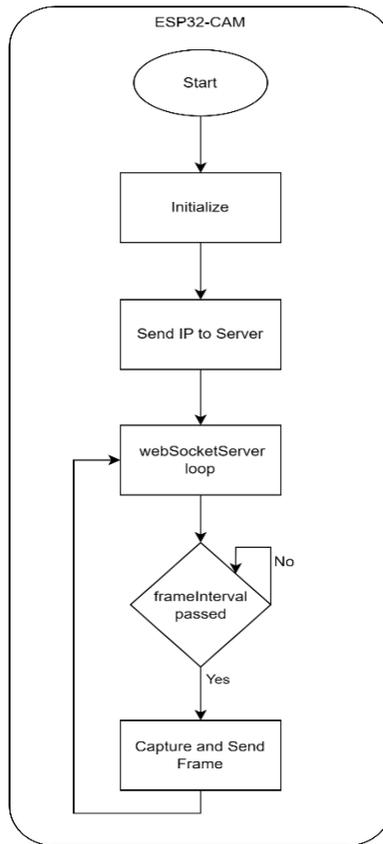


Figure 3.10 - ESP32-CAM Flowchart

For the stream to reach the end user, the camera shares its IP address with the server. The server sends that IP to each instance of the application. The application then, using WebSockets connects with the ESP32-CAM. To reduce the required bandwidth a frame interval is introduced which can be calculated based on the time a frame needs to arrive at the application. This ensured that no or little frames would be lost. To send the frames to the UI a connection is established the moment the log-in process is completed. When the interval passes, a frame is sent to all connected users, and the UI is updated with the new frame (Figure 3.10).

In this approach, a camera is used for 4 users. The stream gets divided into 4 parts by the application and each user gets to see only its allocated microcontroller. Based on availability and the budget for the overall system the university can use one camera for each microcontroller allowing a better stream quality. Further discussed in Future work.

3.1.4 Multiple users

The systems need to allow only a certain number of users at once. This number should be determined by the number of the available slave microcontrollers. While multiple users are using the system, its performance and reliability should not be affected. Further, each microcontroller must be programmable and receive commands only from its assigned user.

To handle multiple users a set of rules needs to be followed to avoid conflicts and provide a fair allocation of resources. First, the number of available microcontrollers needs to be predetermined to avoid non-existing resources being allocated to users. Based on that number a server should be able to allocate all the available resources starting from number 1-4. The number also indicates allocation priority for example if we have multiple systems like A and B each consisting of 4 microcontrollers. The first microcontroller to be allocated is A1 and then B1, not A2. This will reduce the system's load by using all available resources equally. Also, the server needs to free those resources when their reservation expires to allow other users to use the system. To allow an automatic log-out function to work properly a way is needed to inform the application about each user's period of reservation.

When users enter their credentials and the verification is completed, the server checks if the user has a reservation. If so, the server sends the corresponding client number to the application along with the remaining time of the reservation. The client identifier is sent before each file or along with commands to the ATMEGA. The ATMEGA then identifies the destined microcontroller and performs the necessary actions. That helps the system to avoid collisions and provides access only to the user's microcontroller. The application revokes access to the user when its reservation expires and empties a slot to enable other users to use the system. Also, the limit of allowed users is set by the server.

3.1.5 Flexible

In terms of flexibility, the proposed platform should aim to allow the integration of various components that are available and widely used by the institute. It must support multiple microcontrollers, both for the main microcontroller which acts as the master of the system, and for the slave microcontrollers.

To achieve flexibility, the system should be able to identify and distinguish between different slave microcontrollers. The UI should provide clear information about the type of microcontroller a user is assigned and even be allowed to choose which microcontroller they want to reserve. By being able to select a specific microcontroller, users can learn to identify which type of microcontroller is more suitable for them based on their project's needs.

The flexibility also extends to the possibility of running the main code, designed for the master microcontroller (ATMEGA32A), on other microcontrollers with similar pinout and oscillator speeds. As for the slave microcontrollers (ATMEGA328P), the current version can support only ATMEGA microcontrollers which are programmed in pages (not Bytes) and have up to 32kB of memory.

3.1.6 Expansion Possibilities

Considering any expansion possibilities of the platform in the development stage ensures that it will potentially remain relevant over time. An expandable platform can be easily adapted to be used in different courses. Offering the flexibility to expand if its user's needs change over time.

To ensure that the platform could expand in the future the microcontroller's different pin capabilities like support for I2C or USART should be assessed. Having access to those pins allows for future additions to expand the platform's capabilities. For instance, the platform could be used in fields like robotics just by allowing the potential integration of a motor controller and sensors should enable students to program and control robotic systems remotely.

For any new expansion to be integrated effectively into the platform, the documentation should be regularly updated with guidelines on how to use any newly added module. Also, testing and evaluation of all modules after any new addition is required to ensure compatibility and to define the module's capabilities to provide users with ample instructions for the new modules. Additionally, the UI should be updated, if necessary, to support the operations and interactions required by the newly added components.

While the current platform was developed and tested without additional modules, certain pins were left unused, making them available for connecting supported modules. For example, pins that support USART and the I2C protocol remain accessible, providing users the freedom to interface with modules supporting these communication standards.

3.1.7 Scalable

A scalable platform must be able to adapt to the institute's different needs over time. It must be able to scale in both ways upwards and downwards depending on current needs. Offering cost-effectiveness and allowing spare parts to be reused in other projects.

For any platform to be scalable it should offer the flexibility for administrators to add or remove components as required. To accomplish this the system must be designed to handle multiple instances of the system. Ideally, a truly scalable platform can be replicated by creating a copy of the existing system and deploying it as needed. Permitting the platform to utilize all available resources without any additional changes. While the application itself requires no modifications to handle any number of additional systems, certain changes are necessary for the server and database to achieve full scalability.

Due to time and resource constraints, the platform is not fully scalable at this stage. Future work could focus on improving the scalability of the platform. Addressing the server and database considerations could enable the platform to handle more users and be sustainable to be used as an educational platform.

3.2 Apparatus

This section provides an overview of the hardware and software components used in the platform and their selection criteria. Providing a clear understating behind each choice to accommodate the platform. Further, the platform layout is presented along with connection diagrams for each hardware component.

3.2.1 Platforms Connection layout

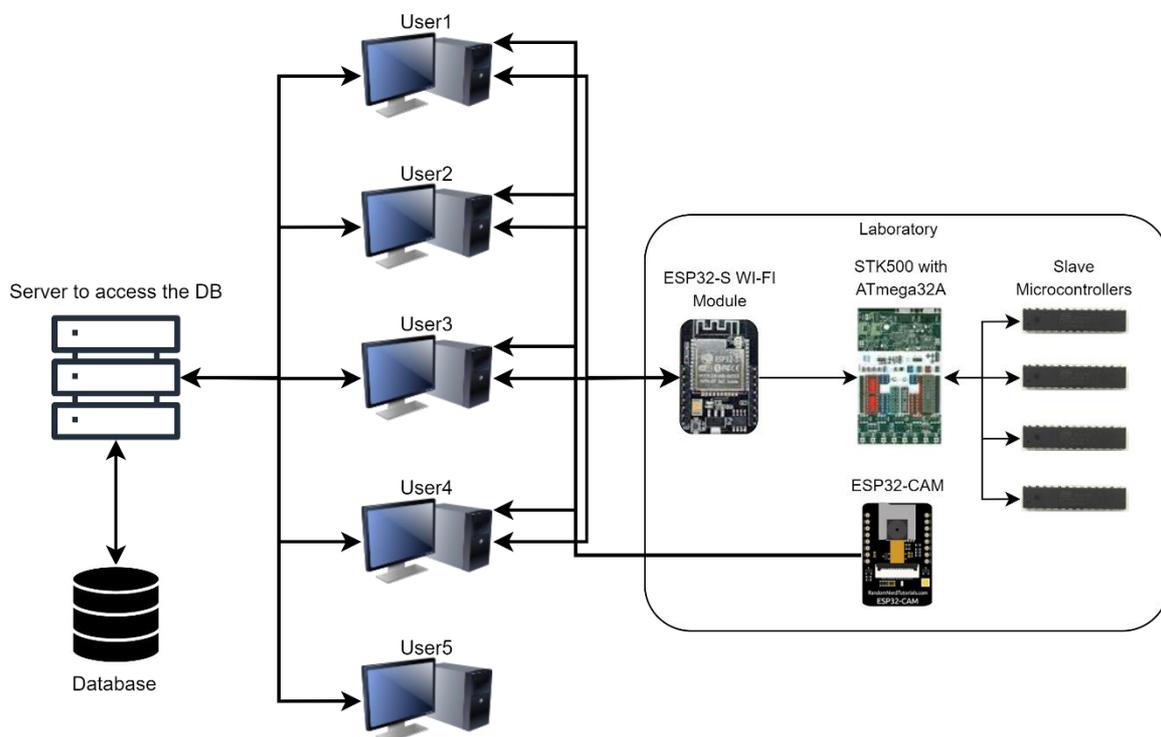


Figure 3.11 - Platforms Connection Layout

The overall platform layout as illustrated above in Figure 3.11 consists of a server that acts as an Intermediate for the UI and database. The system supports up to 4 simultaneous users, each of them interacting with their microcontroller independently. An ESP32-CAM camera is utilized to provide a live stream, allowing users to observe results in real-time.

The ESP32-S communicates with the ATMEGA32A within the STK500 development board through the USART communication interface providing a bridge between the Wi-Fi network

and the master microcontroller. Allowing the reception of files and commands from users via the UI.

Also, the system includes four slave microcontrollers consisting of two ATMEGA328P and two ATMEGA168P. These microcontrollers are connected to the ATMEGA32A using the SPI interface. Overall, this architecture allows multiple users to interact with the platform simultaneously, observe real-time results, and control a slave microcontroller using the user interface.

3.2.2 Hardware Components

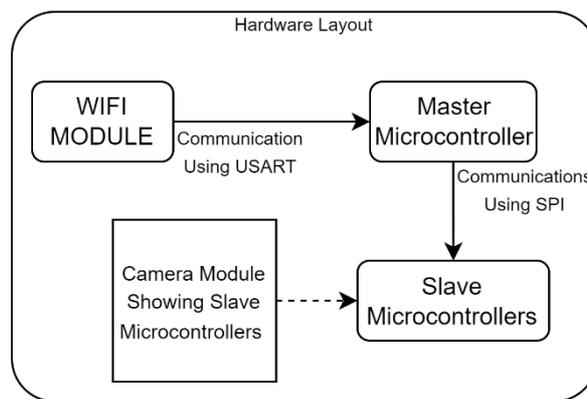


Figure 3.12 - Hardware Layout

Illustrated above is the connection of the selected hardware components. The selection criteria are mentioned in the sections below providing an overview for each choice.

3.2.2.1 Selection Criteria

Several factors were taken into account in the selection of the hardware components. Primarily, as the main functionality of the embedded system is focused on the interaction between the end-user and the slave microcontroller, the selected type and specific models of slave microcontrollers acted as the main determinant for the selection of all other equipment, to enable compatibility. Furthermore, the selection had under consideration the fulfillment of the additional objectives previously mentioned, whilst taking into consideration the product's accessibility and cost-effectiveness.

3.2.2.2 ATMEGA328P & ATMEGA128P – Slave Microcontroller

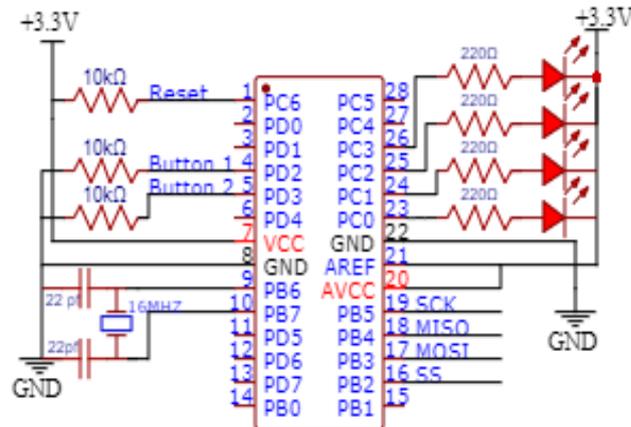


Figure 3.13 - ATmega328P and ATmega168P Diagram

The selection of the slave microcontroller should fulfill various technical characteristics to ensure the desired functionality of the embedded system. Firstly, the microcontroller should support the synchronous serial communication interface (SPI) protocol to allow programming using the AVR910 protocol, along with ground and Vcc out pins. Even though these are usually found in most microcontrollers, for this research the controllers should also include digital and analog i/o pins and should allow connectivity with a timer crystal.

Taking into consideration the most common microcontrollers used in educational environments, the ATMEGA328P was chosen to fulfill two of the four available slots. The ATMEGA328P has a very low cost, has ample memory, and plentiful digital and analog i/o pins which should accommodate future needs for additional components to be inserted into the board. To test the ability of the system to accommodate different models of microcontrollers, the ATMEGA168P was chosen, as it possesses half the memory compared to the ATMEGA328P. This should give an idea of any potential issues that could arise using a different microcontroller or a microcontroller with less memory.

Furthermore, based on available GPIO pins and according to the specs of the selected microcontroller each of the slave microcontrollers got two software buttons and four LEDs.

The number of LEDs was decided first to occupy only the analog input pins. In this case the pins from PC0 to PC3 while leaving unused the pins PC4 and PC5 that support the I2C protocol. Further, the number of buttons was chosen based on the number of LEDs and they were connected to the pins PD2 and PD3. Allowing users to utilize the interrupt functions of those pins. Also, the 16MHz oscillator provides sufficient processing power for the microcontroller while ensuring stable and accurate operation.

3.2.2.3 ATmega32A - Master Microcontroller

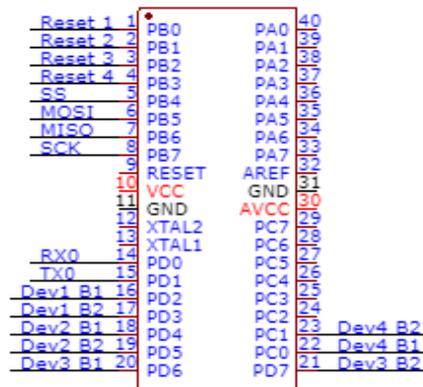


Figure 3.14 - ATMEGA32A Diagram

The selected microcontroller should be able to act as the system's master microcontroller. It should be capable of programming the slave microcontrollers using the AVR910 instructions. Also, ample I/O pins are required to support the necessary functionalities. Such as software buttons for each slave microcontroller, along with their individual reset lines. Further, although most AVR microcontrollers support protocols such as USART and I2C the selected microcontroller must support either to communicate with the Wi-Fi module.

Considering widely used microcontrollers in educational environments and other applications a few options were available. Access to more detailed guides and supported libraries was also a consideration for the selection. Allowing to simplify the development process, troubleshooting and finding solutions to commonly known issues. As a result, two options were considered the ATMEGA16A and ATMEGA32A. Since, the ATMEGA32A offers double flash memory and SRAM, it was chosen to act as the systems master microcontroller.

3.2.2.4 ESP32-S - Wi-Fi Module

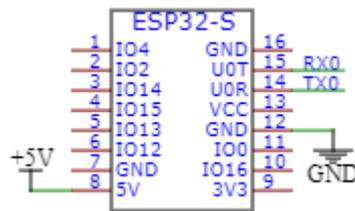


Figure 3.15 - ESP32-S Diagram

The selected Wi-Fi module should be reliable, cost-effective, and easily programmable. A viable option could be the ESP8266 System on Chip (SoC). Widely used in IoT devices, this SoC comes with default firmware containing AT commands, enabling internet capabilities for any microcontroller. Additionally, it is compatible with the Arduino IDE, thus simplifying the programming process. A module utilizing this SoC is the ESP12-E which is a more suitable choice for the early development stages.

Another more advanced and powerful option is the ESP32-S module containing the ESP32 SoC and providing compatibility with WPA2-Enterprise networks. This makes it more suitable to be deployed in more robust and secure networks that require WPA2-Enterprise compatibility.

While both the ESP12-E and ESP32-S offer Wi-Fi connectivity, the ESP12-E is preferred for its cost-effectiveness. The ESP32-S was chosen because it supports WPA2-Enterprise, making it an appropriate option for settings that demand higher levels of security.

3.2.2.5 ESP32-CAM - Camera

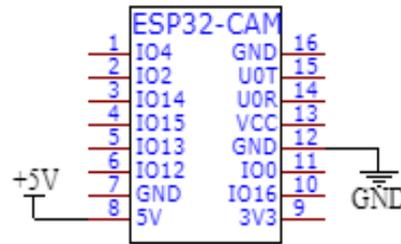


Figure 3.16 - ESP32-CAM Diagram

The selected camera for the system should be reliable, cost-effective, low power, easily programable, and have support for the WPA2-Enterprise network. A compelling choice is the ESP32-CAM which combines an ESP32-S SoC with the OV2640 camera sensor. The 2MP OV2640 camera sensor is an entry-level camera module and offers stability and reasonable video quality. Also, it is programable using the Arduino IDE and an FTDI module. The camera module could also be upgraded if a different streaming quality is required. For the more powerful 5MP OV5640 camera sensor or other supported modules.

For the platform's needs, the ESP32-CAM with the OV2640 sensor provides a well-balanced and cost-effective solution. It meets the requirements for reliability, low power consumption and support for WPA2-Enterprise networks.

3.2.3 Software Components

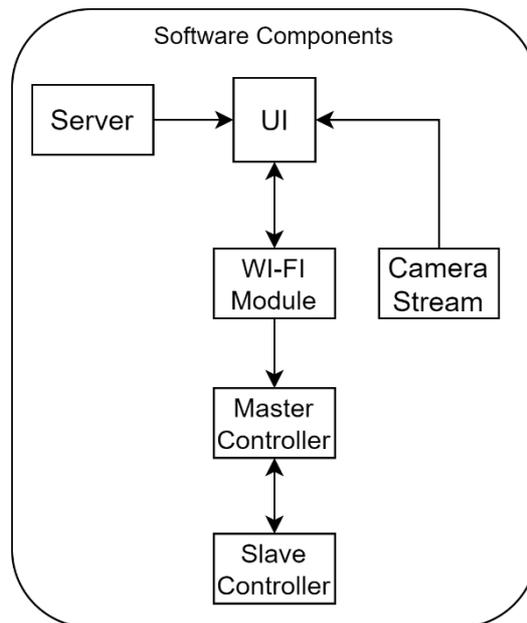


Figure 3.17 - Software Components

The software components also take part in providing a seamless interaction between users and the embedded system. The software components and their corresponding functionalities are described in the section below. The software generally could be divided into two main sections: the User Interface (UI) and the Backend.

3.2.3.1 Programming Languages

For the development of the User Interface (UI), a high-level programming language is required. Python was selected since, it offers an extensive standard library, ease of use, and availability of GUI frameworks like Tkinter. Further, the database can be created utilizing Python's SQLite3 library. Access and exchange of information could be established between the database and server by also using sockets and the SQLite3 library. On the other hand, programming microcontrollers require a low-level programming language to offer direct hardware control, efficient memory management, and fast execution speed. Atmel Studio offers support for both C and assembly languages. The wide variety of supported ATmega microcontrollers makes it ideal for programming the ATMEGA microcontrollers. While ESP32-S and ESP32-CAM utilize the Arduino IDE with its modified version of C++.

3.2.3.2 User Interface Design

Designing the User Interface (UI) poses several challenges. One of the main challenges is to seamlessly synchronize the interface with the server which acts as the intermediate between the database and UI. Allowing user authentication, reservation management, microcontroller assignment, and providing the IP addresses for the video stream, and Wi-Fi module. Also, it is desired for the graphics to have a modern look therefore, the selected GUI framework was not Tkinter but a similar UI library based on Tkinter called CustomTkinter [10].

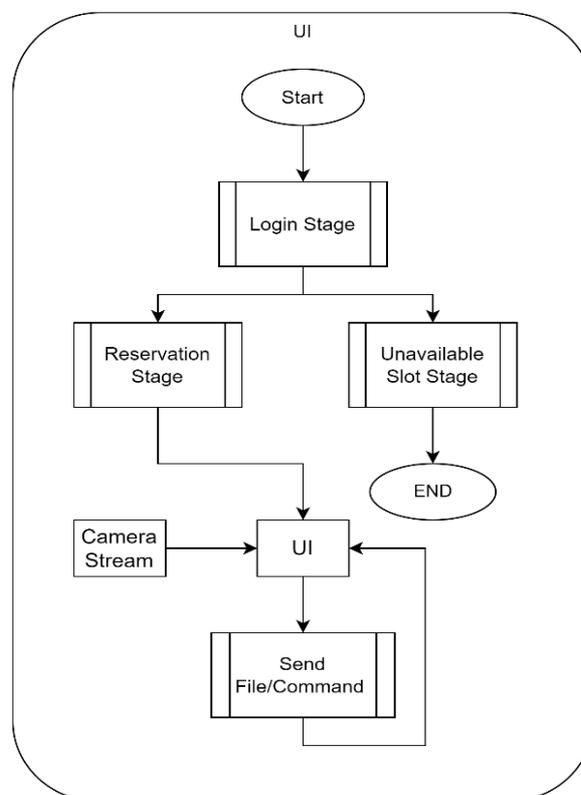


Figure 3.18 - User Interface Flowchart

Booting the application, the user should be presented with the login portal. To access the platform a valid set of credentials is required. Upon verification, one of three possible actions can occur. The first possible action is to be prompted with a reservation time selection porta. Users could choose from a dropdown menu containing all the available periods, and their desired reservation period. Which then should be synchronized with the database, ensuring a reserved microcontroller for the specified period. Another possible action is to directly

access the platform's UI without requiring to make a reservation this should happen only when a reservation already exists in the database. Lastly, a notification panel is essential to inform users when there are no available reservation slots along with information about the next available slot.

After credential validation and reservation period selection, users should be directed to the platform's UI. The User Interface could be divided into two distinct parts, the control panel and a frame containing the live video streaming. All useful information to the user should be displayed in the control panel, along with the UI elements. The information provided by the control panel must contain the assigned slave microcontroller model, reservation expired time, and which file is selected for uploading. Further, it should contain UI elements such as a file selector to browse and select files and buttons to enable specific actions. Each button is required to trigger a different desired function for the system (start the uploading process/programming, send commands). The frame containing the live video stream needs to be refreshed regularly as new frames arrive and process the incoming frame in such a way that each user gets to see only its assigned microcontroller. A requirement for the refresh process is to occur without affecting the User Interface responsiveness.

Moreover, it is mandatory from the UI to log out any users with an expired reservation and erase their slave microcontroller. Erasing a microcontroller must happen only when the user desires, and when their reservation is expired. If the user chooses to exit the UI before a reservation expires the slave microcontroller should not be erased.

3.2.3.3 Backend Development

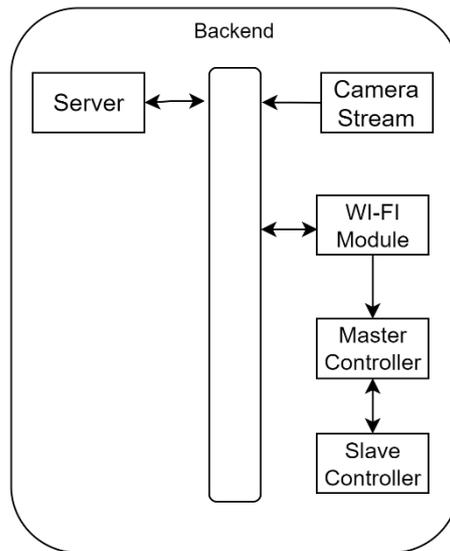


Figure 3.19 - Backend Components

The back-end software development process includes issues related to Wi-Fi module functionality, server functionality, master microcontroller tasks, and video streaming (Figure 3.19). The primary objective is to ensure reliable file transmission and commands. To achieve this, the connection between the Wi-Fi module and the UI should utilize Websockets, which was determined to be the most suitable approach after considering various available options.

The Wi-Fi module should accept new clients sequentially meaning that while a user uploads a file no other users can access the Wi-Fi module to upload a file or commands this approach is further discussed below. Also, a flow control implementation could be essential to manage larger file uploads.

The server should be able to interact with the database and handle multiple user requests at once, without creating races (e.g., both users try to acquire the same slave microcontroller). All the data involving user authentication, reservations, and client assignment should be processed by the server before being forwarded to the application. Also, communication with the application should be established through a secure connection.

The master microcontroller should be able to receive commands and files via the Wi-Fi module. Also, differentiating those input destinations is essential to accomplish the system's desired functionality. File handling capabilities must exceed the allocated buffer size to allow flexibility in file sizes and improved resource utilization. Further, in order to provide a comprehensive user experience, all desired functions for all slave microcontrollers must be supported.

Lastly, the camera should offer users a consistent and stable video streaming experience. The video quality must allow users to differentiate the available LEDs and the latency minimal to provide a smoother interaction with the system.

3.2.3.4 Considerations and Decision for File Transmission and Programming

This section compares the sequential and simultaneous approaches for file transmission and the programming of the microcontrollers. The two different processes are presented along with their possible implementation.

Implementing, sequential programming involves processing clients or tasks one at a time, completing each task before moving on to the next. In this context, it means that the Wi-Fi module accepts and processes clients in sequence. Before accepting a new client, the former client must finish sending the file and the programming of the designated slave microcontroller. This process also extends to task handling.

On the other hand, concurrent programming involves processing multiple clients or tasks at the same time. This method allows for parallel execution, in which multiple clients can send files and program slave microcontrollers without waiting for the previous task to be completed. Also, this extends to task management which means if one user uploads a file another can send and execute a command without being affected by other users' actions.

However, implementing simultaneous programming would have introduced significant complexity. The ATMEGA32A would have to perform a number of complex tasks to implement the simultaneous programming method. It would have to track multiple clients and the pages

that they are programming. Manage multiple file receptions to ensure that they are processed in the correct order. Introduce a delay in each cycle to ensure that the target microcontroller is in programming mode before data is written.

For example, when the buffer is full, it writes the code to the selected slave microcontroller. Then if a client wants to send data, it must save the page where the programming was paused along with the destination and pull low the reset line (Reset disabled) of the selected slave microcontroller. Afterward, it needs to pull the reset line high (Reset active) for programming to begin for another slave microcontroller. Write the code, save the page number and destination, and then pull low the reset line to be able to continue with the previous or next client. This process would have to be repeated each time the received buffer is full and the pages are written in the selected microcontroller.

This process would also require solving the problem of receiving multiple files at once and differentiating them. Each packet of data must have a header for where it belongs, or users must send data in chunks of a pre-defined size and wait for that to arrive before other users can send theirs. This load could be the responsibility of the ESP32-S (Wi-Fi module). Additionally, a delay of about 25ms needs to be introduced in each cycle after the reset line was pulled high, ensuring that the target microcontroller entered its programming mode before starting to write data.

Overall, given these complexities and the need to ensure reliable transmission, the decision was made to pick the sequential programming method. This approach, while simpler and lacks support for instant interaction with the system. It ensures reliable transmission and avoids conflicts by processing one client at a time.

3.3 Assumptions and Limitations

Various assumptions and limitations were considered throughout the experiment's execution. Initially, the system compatibility was tested only on two types of devices, regardless of their broad capabilities, due to component availability and consideration of cost-effectiveness. The time delay between the virtual button trigger and visual feedback was tested manually. While the application is designed to operate on multiple operating systems including Linux, Windows, and MacOS, tests were conducted only on Windows and Linux environments.

Although the system should be fully operational without an STK, the tests were performed with the master microcontroller in the STK. These tests only included basic functions of the slave microcontroller such as digital out. Therefore, other features such as analog read and write, and pulse width modulation were not tested. The system does not accommodate a method to receive messages from the master or slave microcontroller, therefore this prohibits testing for serial communication.

Lastly, as with any online system, regardless of its intended use, it is good practice to assume all possible exploitations of the systems. However, as the security of the system is not of primary concern for this research, it is assumed that users have no malicious intent. Further, the database reservation slots operate under the assumption that the user's system DATETIME is similar to the DATETIME of the server system.

Chapter 4 User and Institution System Perspective

4.1 User Perspective

The following section provides instructions for installing the application to the user's computer, using the application, and writing programs for the provided microcontrollers. Further, the provided instructions are verified for devices using Windows and Ubuntu Operating Systems but not for MacOS.

4.1.1 Installation instructions for Ubuntu/Windows

Before running the code for the application, follow these steps to install the required Python packages from the provided folder:

1. Ensure that your computer has Python 3 Installed.
2. Open a terminal and navigate to the directory where the "requirements.txt" file is located.
3. Use `pip install -r requirements.txt` command to install all the necessary packages.
4. If a specific packet was not installed correctly, use the `pip install <package_name>==<version>` command to try installing it separately. The correct name of every packet along with its version can be found in the "requirements.txt" file.

4.1.2 Using the application

Before running the application script, connect to the university's network directly or via the university's VPN. Afterward, open a terminal and navigate to the directory that contains the "main.py" file, and use the `python3 main.py` command to start the application.

If any abnormalities are encountered during the application startup, it might be due to the utilization of the CustomTkinter library for the GUI. In such cases, to resolve the issue manually re-install the CustomTkinter library.

Once the application is launched a login portal is opened (Figure 5.2). Use the provided username and password to log in. After the credentials are verified a reservation panel should be displayed. Use the drop-down menu to choose the duration of the reservation (Figure 5.3), which cannot be changed later. If the reservation expires while using the application an automatic logout will occur. Further, if the application is closed before the time expires when login back in the reservation will still exist unless the period has expired (if a reservation happens at 14:00 for 30 minutes it will expire at 14:30 under any conditions). The UI should be displayed without the need for a reservation when a reservation exists. Lastly, if no slots are available a notification panel will be displayed with information about the next available slot (Figure 5.5).

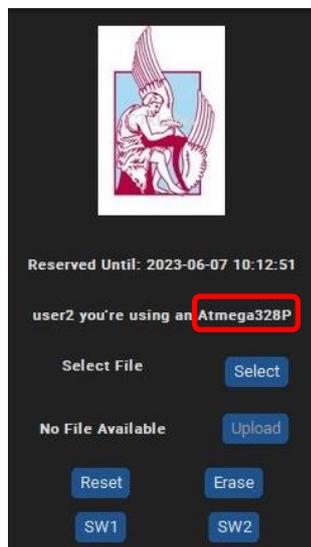


Figure 4.2 - Control Menu

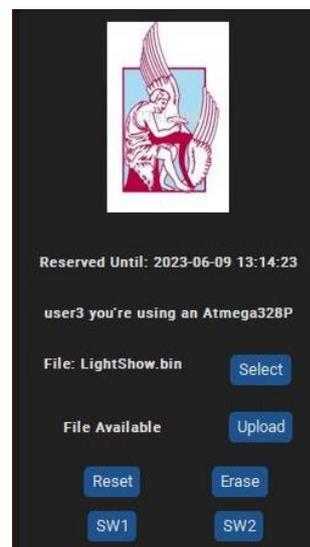


Figure 4.1 - Available file to be Uploaded

At the main window, the control panel is located on the left side. There, a reservation expiration time is displayed. The user's ID is next to the slave microcontroller part number Atmega328P or Atmega168P. Followed by buttons that give access to the system's supported functions. A file browser is included to search and select files along with an upload button to

send the code file. At the bottom of the control panel the reset, erase, and the two buttons labeled SW1 and SW2 are located (Figure 4.2). The upload button becomes available after selecting a file as illustrated in Figure 4.1.

When selecting a file two options are available, to select a .bin or .hex file. The Atmel Studio provides only hex files select the .hex inside the projects folder and the program will create a .bin file and store it in the location of the project so it can be used the next time. The .bin file is required to successfully program the microcontroller.

Before trying to upload any code is recommended to Erase the microcontroller first, and upload the “Blink.bin” file before uploading any other code. This helps to verify that the system is working properly. Before writing a program consult the “Blink” and “Buttons” programs provided in the Example Applications section.

4.1.3 Writing programs

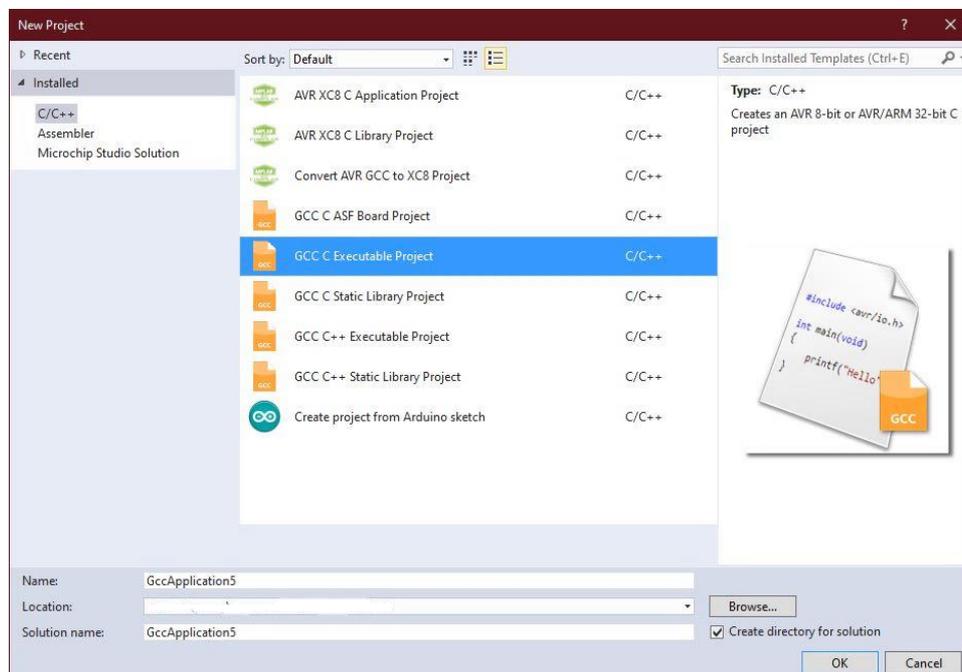


Figure 4.3: New Project

To write programs use the current version of Atmel Studio, create a new project select the “GCC C Executable Project” option give it a name and then press “OK” Figure 4.3.

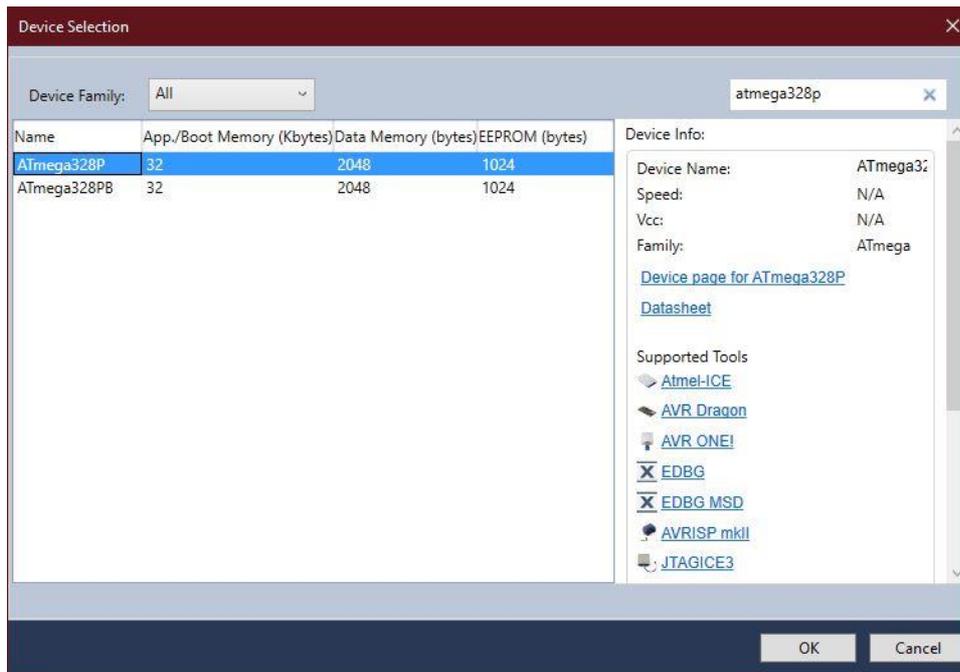


Figure 4.4: Device Selection

In the next windows select the allocated slave microcontroller name (ATMEGA328P or ATMEGA168P) as illustrated in Figure 4.4. At the tools window which can be accessed by pressing the Simulator (1st rectangle below) select the simulator from the options in the 2nd rectangle in Figure 4.5.

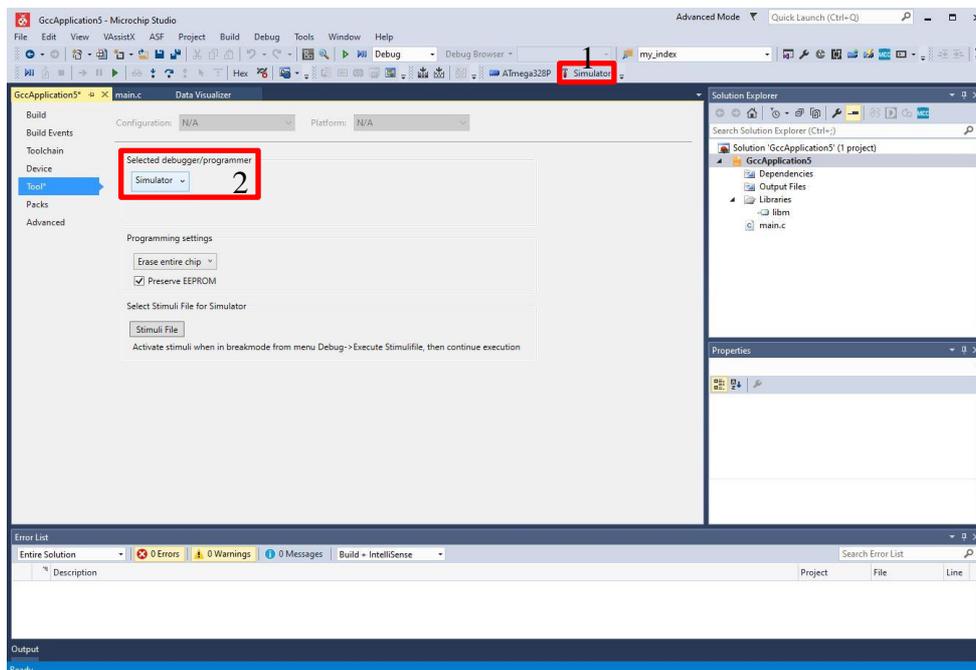


Figure 4.5: Programmer Selection

Before writing any code, ensure that the F_CPU is set to 16MHz at the top of the code, as shown in the Example Applications. This value corresponds to the frequency of the oscillator used by the slave microcontroller. Failing to set the F_CPU correctly could lead to issues when using functions or libraries that rely on this value for their operation.

To provide visual feedback, the slave microcontroller is connected to four LEDs on PortC (pins 0-3). These can be set as outputs using the command `DDRC |= 0b00001111;` and then the LEDs can be utilized as desired. Additionally, two software buttons are connected to PortD (Pins 2-3). To set these as inputs, use the command `DDRD &= ~(1 << PD2 | 1 << PD3);`. For guidance on how to check if a button is pressed, please refer to the Buttons example.

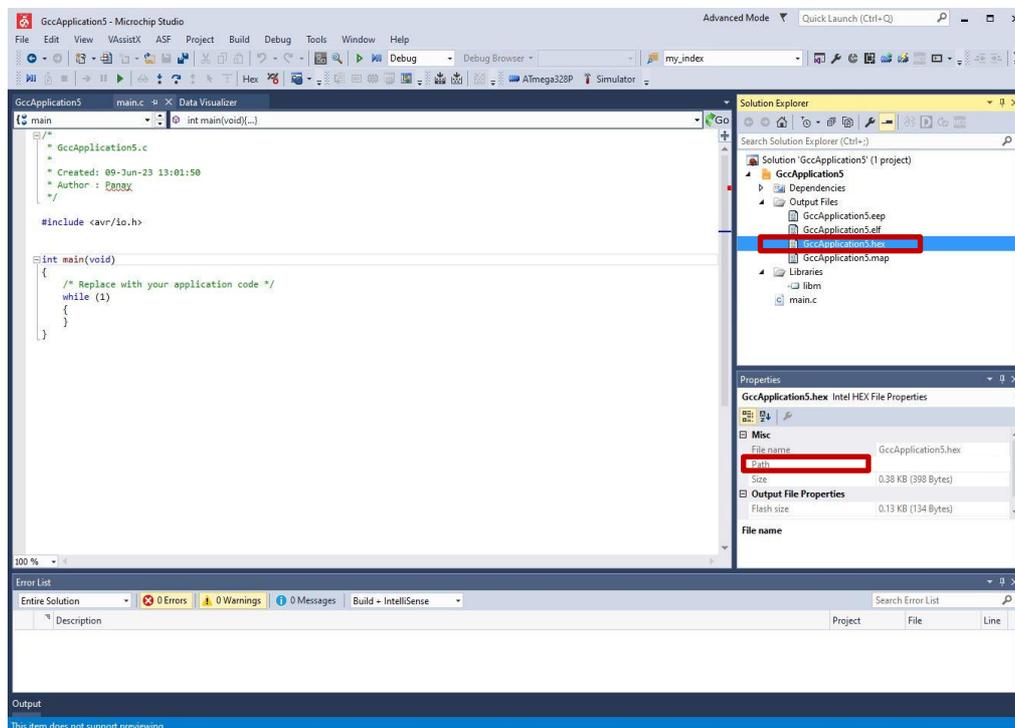


Figure 4.6: HEX File Location

The .hex file is created when the program gets build (Build tab) and to locate the .hex file (Figure 4.6 1st square) there is a path provided in the properties window 2nd square.

To upload the code and program to the microcontroller locate and select the HEX file using the applications file browser. Afterward, the selected file presented in the control panel should have a BIN extension and the upload button should be enabled. By pressing the

button, the file is used to program the microcontroller. If everything is correct with the code and the LEDs are utilized the desired output is expected in the video stream.

4.2 Engineer's Perspective

The following programs are utilized by the platform. Each programs functionality is described in detail providing a better understanding of how the platform works. While it equips engineers with the necessary information to make modifications and resolve any issues that may arise.

4.2.1 setupdb.py

Using the following command `python3 setupdb.py` will create the database table.

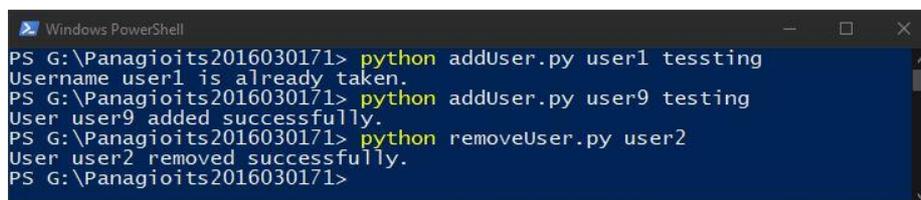
Field name	Type
username	VARCHAR (255) NOT NULL UNIQUE
password	VARCHAR (255) NOT NULL
client	INTEGER UNIQUE CHECK (client >= 1 AND client <= 4)
reserve_time	DATETIME

Table 1 - Database Fields and Types

The table above contains all the required fields for the system's database. The username and password cannot be Null and the username must be unique. The client number should be from 1-4 since that is how many slave microcontrollers the current system supports. And the reserve time is the time of expiration for the user's reservation.

Also, the script adds a few usernames and passwords to the database. To add a level of security the passwords are encrypted with a sha256 encryption provided by the "hashlib" library.

4.2.2 addUser.py



```
Windows PowerShell
PS G:\Panagioits2016030171> python addUser.py user1 tessting
Username user1 is already taken.
PS G:\Panagioits2016030171> python addUser.py user9 testing
User user9 added successfully.
PS G:\Panagioits2016030171> python removeUser.py user2
User user2 removed successfully.
PS G:\Panagioits2016030171>
```

Figure 4.7 - Adding / Removing User From DB

This script is used to add a new user to the database using the following command inside the database directory `python3 addUser.py <username> <password>`. The username and password can be anything except "NULL" and the username must be unique (Figure 4.7).

4.2.3 removeUser.py

To remove a user since only the administrator has access to the database only a username is required. To remove the user, use the `python3 removeUser.py <username>` command (Figure 4.7).

4.2.4 serverv2.py

The server acts as an intermediate between the Database and Application, and it has the following functionalities:

- authenticate users
- provide IP addresses for the system Wi-Fi modules
- add reservations to the Database
- provide the user's client number to the Application
- decline access if there are no slots available
- remove expired reservations from the Database

The server script must be executed after the database is set up and before powering on the system. To start the server, use the `python3 serverv2.py` command.

The server has 4 threads each supporting a different functionality. Two threads are used to receive the module's IP address. The other one is used to check for an expired reservation every 15 seconds and remove it from the database. Further, the 4th thread is used to connect with the Application receive the user's credentials and authenticate them. In succession, if the credentials match a database record it makes the following actions:

1. Checks if the user has a reservation, if it does it sends to the application the remaining time for the reservation and the user's assigned client number.
2. If the user needs to make a reservation a search is performed to determine whether there are any available slots. If a slot is available, it assigns a client number to the user and waits for the reservation period to update the database with it.
3. If no slots are available, it searches the database for the closest expiration time and sends that to the application.

4.2.5 CameraStreamWebSockets.ino

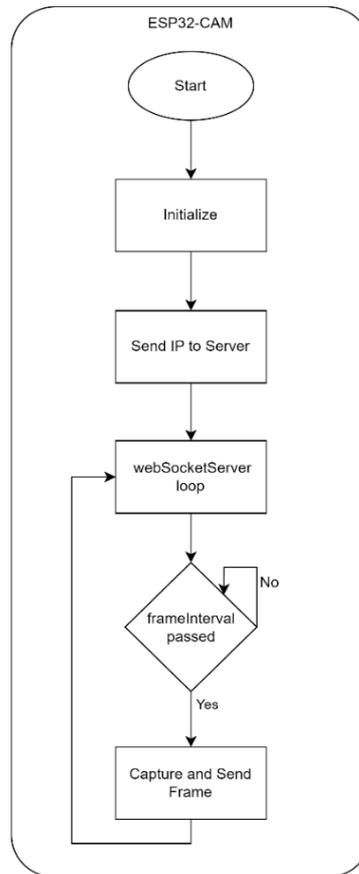


Figure 4.8 - ESP32-CAM Flowchart

This Arduino sketch is used to program the ESP32-CAM. First, each GPIO is defined for the specific module model which is an AI-THINKER. Then a username and password are declared to be used to connect to the university's Wi-Fi. After that, there are two instances one for an AsyncWebServer that is used to host an HTTP web page which allows to verify if the connection to the university's network is established correctly this instance can be removed. The other one is for a WebSocketServer which is used to send the frames to all connected users. Moreover, a frame interval is introduced to control the time between frame captures and broadcasts to the users.

The `initializeCamera()` function initializes the camera with specific configurations the most important parts are the `pixel_format` which is set to `jpeg` to reduce data size, the `frams_size` is set to `VGA` higher qualities are laggier, `grab_mode` to `CAMERA_GRAB_WHEN_EMPTY` the

camera will grab a new frame when the frame buffer is empty, `jpeg_quality` is set to 8 which provided the best result in terms of response, stability, and quality, and the `fb_count` is set to 1. This setting can change to accommodate the network conditions the camera will be operating on.

The `broadcastFrame()` function broadcast the frames to all connected users.

The `sendIPAddress()` function sends the IP address to the server.

In the `setup()` function all the required initializations are made. The connection to the wifi is established using `WiFi.begin()`, the `initializeCamera()` function is called to initialize the camera, both servers get initialized, and the `AsyncWebServer` posts a Hello, world! message to the IP address, and the `sendIPAddress()` is called to send that IP address to the server.

Finally, in the `loop()` function, `websocketServer.loop()` is used to check for WebSocket events, and using the interval every 50ms a new frame is broadcast.

4.2.6 WebSocketServer.ino

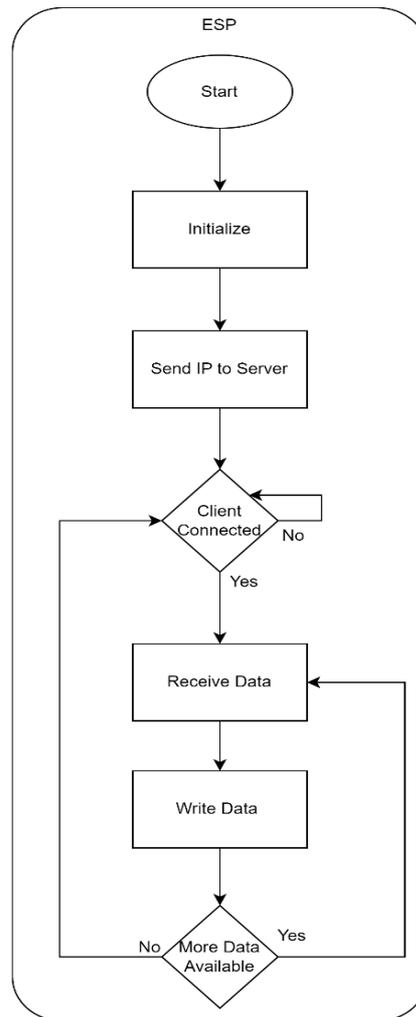


Figure 4.9 - Wi-Fi Module Flowchart

This Arduino sketch is used to program the ESP32-S. Also, it can program the ESP12-E by changing the wifi libraries. This sketch uses a custom WebSocketServer library and the `WebSocketServer.h` file can be found [here](#) on GitHub along with instructions on how to install the library. Then a username and password are declared to be used to connect to the university's Wi-Fi. After that, there is the WiFiServer instance with the port that is going to be used and the instance of the WebSocketServer. Moreover, there is a Data string to store the received data and a WiFiClient instance for the clients.

The `sendIPAddress()` function sends the IP address to the server.

The `writeString()` function is called to write the received data to serial one by one.

In the `setup()` function all the required initializations are made. It sets up the serial communication at 76800bps, connects to the Wi-Fi using `WiFi.begin()`, the server gets initialized, and the `sendIPAddress()` is called to send that IP address to the server.

Finally, in the `loop()` function, It checks for a connected client, performs a WebSocket handshake, and then enters a loop. There it reads data from the WebSocket connection and calls `writeString()` to send this data over the serial connection. Then a delay function is called making sure there is enough time to write the data. Then a response is sent back to the client to inform that the data was received and in order to send more data or to disconnect.

4.2.7 Atmega32Av2.atsIn

The main code for this project is in the Main.c file. This project includes the myAVR910.c, SPI_At16.c, and usart.c files, along with their respective .h counterparts.

4.2.7.1 Usart file

This code includes routines for initializing the USART, receiving a character, transmitting a character, and sending a string of characters.

The `UART_init()` function initializes the USART protocol. Specifies the baud rate and enables transmission and reception of 8-bit characters. Also, the baud rate prescaler is calculated based on the system clock and the desired baud rate and loaded into the appropriate registers.

The `USART_RxChar()` function waits until a character is received (RXC flag in the UCSRA register), then returns the received character using the UDR register.

The `USART_TxChar()` function waits until the transmit buffer is empty (UDRE flag in the UCSRA register), then loads the character to be transmitted into the UDR register.

The `USART_SendString()` function writes a string of characters. This is accomplished by calling the `USART_TxChar()` function for each character in the string. When a null character is encountered that's the end of the string.

4.2.7.2 SPI file

This code includes functions to initialize the SPI and to send and receive data.

The `SPI_Init()` function configures the necessary hardware settings for SPI communication. It sets the MOSI (Master Out Slave In), SCK (Serial Clock), and SS (Slave Select) pins as output and the MISO (Master in Slave Out) pin as input. It also configures the SPI Control Register (SPCR) to enable SPI, set the microcontroller as the master, and set the SPI clock rate to $f_{osc}/128$.

The `SPI_Write()` function sends a byte of data over SPI. It writes the data to the SPI Data Register (SPDR), then waits until the transmission is complete. After the transmission, it reads the SPDR to clear the SPIF flag in the SPI Status Register (SPSR), which indicates that the transmission is complete.

The `SPI_Read()` function reads a byte of data over SPI. It sends a the byte 0xFF to initiate the SPI clock and waits until the reception is complete. Then it returns the received data.

4.2.7.3 myAVR910 file

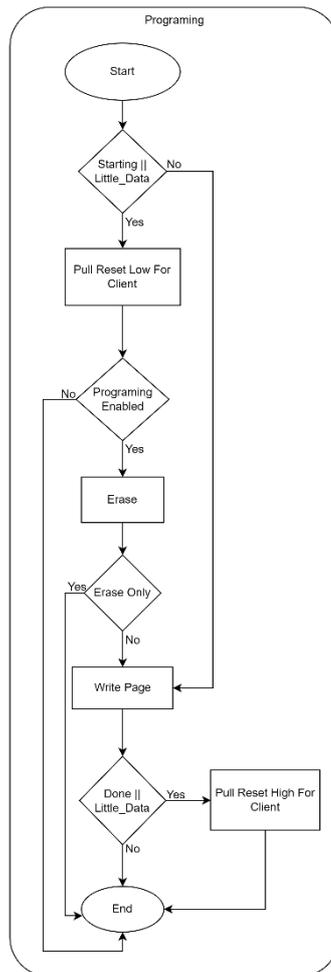


Figure 4.10 - AVR910 Programming Flowchart

The provided code is an implementation of programming an AVR microcontroller using the AVR910 In-System Programming over the SPI protocol. The main function, `program ()`, manages the programming process, which includes entering the serial programming mode, enabling programming, reading the microcontroller's signature, erasing the chip, and loading and writing memory pages.

Further, the `program ()` function has 4 different states. The first state named Starting is used at the beginning of the programming and it pulls the reset line low (active) on the selected microcontroller to enter the programming mode. Also, in this state, the “enableProgramming” function is called to check if the microcontroller can be programmed.

The “program” function can also be called to erase the microcontroller not to program it. The Starting is like the Little Data state but in the Little Data state when the pages are written the reset line is pulled high to exist the programming mode. The Starting and Middle stages keep the reset line low because the program expects more data. When the last bit of data is sent, the state is Done to inform that no more data are expected and to pull the reset line high.

For the “program” function to erase a microcontroller it follows the same proses as programming it needs to call the “enableProgramming” function to enter in programming mode and instead of writing data it calls the “chipEraes” function to erase them. Then releases the reset line and the erase is completed.

Finally, to write the code the “loadMemoryPage” function loads a page of data into the microcontroller's memory buffer using the Load Program Memory Page command (0x40 for low byte, 0x48 for high byte). The “writeFlashMemoryPage” function writes a page of data from the memory buffer to the flash memory using the Write Program Memory Page command (0x4C).

The `enableProgramming()` function sends the Programming Enable command (0xAC, 0x53), as specified in the AVR910 protocol, to the microcontroller, which is necessary to start the programming process.

The `readVendorCode()`, `readPartFamilyAndFlashSize()`, and `readPartNumber()` functions read the microcontroller's signature, which includes the vendor code, part family and flash size code, and part number code, respectively. These functions use the Read Signature Bytes command (0x30) from the AVR910 protocol.

The `chipErase()` function sends the Chip Erase command (0xAC, 0x80) from the AVR910 protocol to erase the microcontroller's memory, preparing it for new data.

The `readProgramMemory()` function reads a byte of data from the program memory using the Read Program Memory command (0x20 for low byte, 0x28 for high byte).

The `poll()` function is used to wait until the microcontroller is ready for the next command.

4.2.7.4 Main file

This contains the main function for the `Atmega32Av2.atstln` project. It is used to initialize USART, SPI, and all the required Ports are set as inputs or outputs.

Starting the CPU frequency is defined based on the oscillator that is used. All the required libraries are imported and a buffer is declared to store received data. After, all the required variables are initialized.

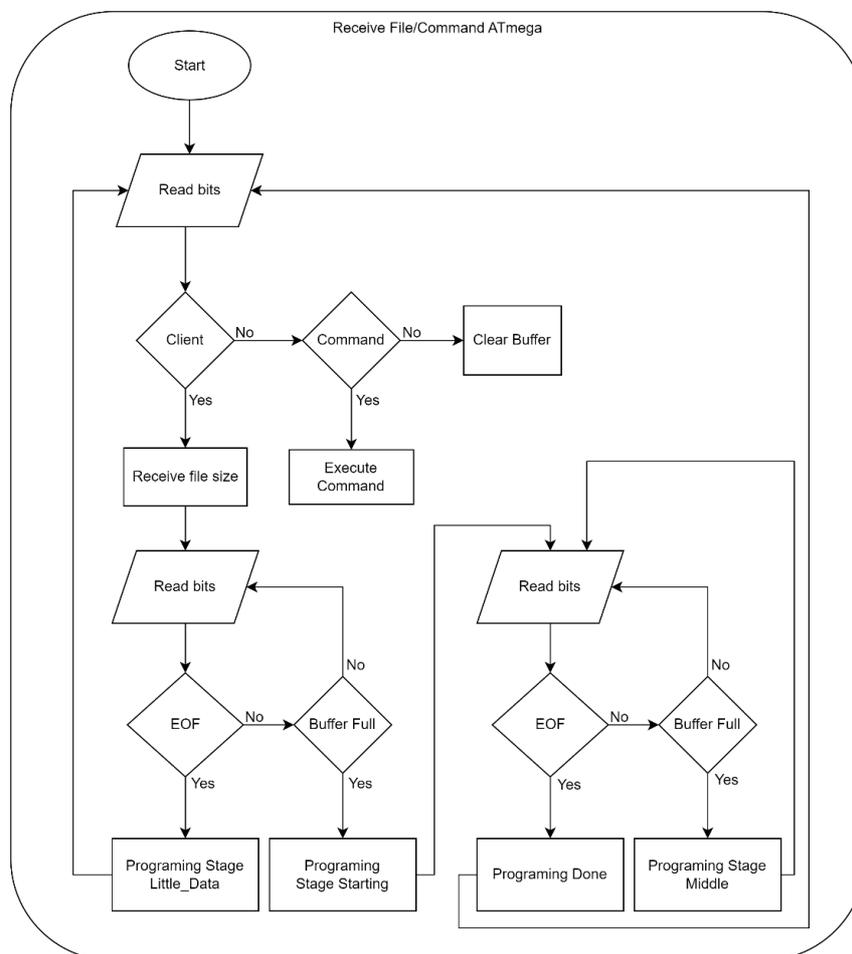


Figure 4.11 – ATMEGA32A File Receive

The flowchart illustrates the functionality inside the Interrupt Service Routine (ISR) for USART data reception. When a character arrives, the ISR is called. There the character is

stored in the buffer and a set of checks is performed. When a client is connected, it is expected to send either a client number or a command. If none of them match the buffer is cleared. When a command is sent the appropriate action is executed. If a client number is received then the next expected information is a file size. When that is received the programming is starting. There are 4 programming stages the Little Data which is used when the file was received but the buffer is not full. The Starting stage for when the buffer is full and more data are expected this stage writes the first two pages on the microcontroller. The Middle stage is for when the buffer fills a second or third and so on. The Done stage is for when more than 2 pages were written and the buffer is not full but the file was fully transmitted.

The available commands are reset, erase, and 2 software buttons for each client. After each action, the required rests are made like buffer reset, buffer index, stage indicators, file size, page numbers, and button resets. The master microcontroller connects with the slave microcontrollers using the Ports in Table 2. Each software button initially is set to High.

Microcontroller Number	Reset	Software Buttons
1	PB0	PD2/PD3
2	PB1	PD4/PD5
3	PB2	PD6/PD7
4	PB3	PC0/PC1

Table 2 - Utilized Ports

For the master microcontroller to control the software buttons when a button press command arrives a check is performed to determine the current state of the Pin Low or High. Then the state is reversed and a delay of 20 milliseconds is called to make sure the Pin status changed. Also, when an erase command arrives the software buttons Pin a reverted to their original state.

4.2.8 Main.py

The “main.py” file contains the code for the application. The user interface is configured using the “CustomTkinter” libraries. The communication with the server is handled by the “socket” library, the video stream frames are received utilizing the “websocket” library and the file transmission also uses the “websocket” library.

The `TryLoginServer()` function connects with the server to authenticate users, receive client number, reservations expiration time or decline access if no slots are available. When a client needs to make a reservation the `load_reservation()` is called and after the user selects a reservation period that is sent to the server to update the database. If no slots are available the `show_next_client_time(time_only_str)` is called to display the earliest time a slot is freed. Further, if the client has already made a reservation the `load_app(reservation_time_minutes)` is called to load the UI.

Inside the `load_app(reservation_time_minutes)` after the UI is loaded, the reservation time is converted to milliseconds, and a timer is set to automatically log out the user when that time passes and the `CameraStream()` is called. The `CameraStream()` connects to the camera module using the IP it was sent by the server. When a frame is sent it is processed and divided into 4 parts since the system only uses one camera for 4 users. Afterward, only the frame corresponding to the current user is displayed.

Interacting with the UI when the select button is pressed the `choose_button_event()` is called. A file explorer is opened which only displays HEX and BIN files. If a HEX file is selected it gets converted to a BIN using the “hex2bin” function from the “intelhex” package and saved to the directory of the HEX file. Also, the upload buttons get enabled to enable users to upload the selected file.

Pressing the upload button the `upload_button_event()` is called. This function tries to connect to the ESP32-S to upload the file. When a connection is established a client identification is sent (e.g., client1), the file size is sent and finally, the file is sent in chunks of

256 bytes. After each “ws.send()” an acknowledgment packet is anticipated before sending more data.

Sending commands to the system calls one of the following functions to send the user identification, such as “C1” alongside the relevant command:

- `Switch_button_event_1()` sends C1SW1
- `Switch_button_event_2()` sends C1SW2
- `Dev_Rst()` sends C1Erase
- `Dev_Ers()` sends C1Reset

Moreover, each window has an exit function that closes the application when called. The `exit_bot_Timer()` is called when the reservation period expires. Before closing the application the `Dev_Ers()` is called to erase the user's slave microcontroller and the `stop_event.set()` is used to terminate the camera thread. The rest of the exit function simply closes the current window except the `exit_bot_app()` which also stops the camera thread.

Lastly, a logger is included in the application for debugging purposes and currently only logs events where exceptions occur. There are additional functions related to the user interface that are not described above and are responsible for managing the application's visual elements.

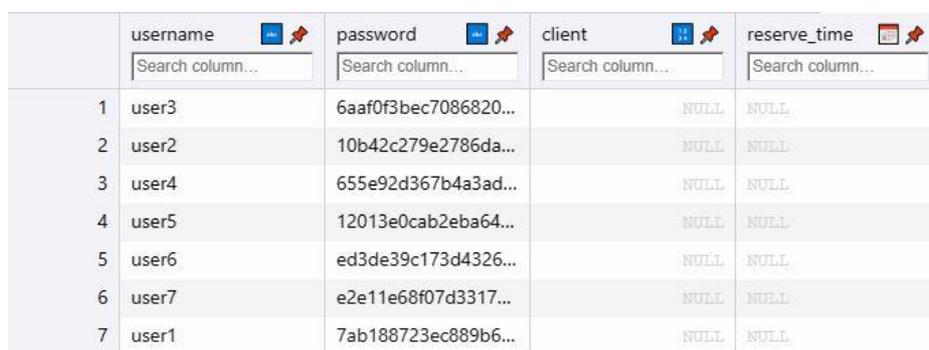
Chapter 5 System Verification and Evaluation

5.1 Testing and validation procedures

In this section, comprehensive testing and validation procedures are outlined to assess the functionality of the developed educational platform. Each of the main objectives along with the overall system is evaluated and the results are presented. Subsequently, the following subsection provides the testing setup layout offering a better understating of the testing environment.

5.1.1 Initial Development and Component Testing

To test the system setup, the instructions included in the user guide in section 4.1 were followed. Simultaneously, various procedures were followed to assess the functionality of the system and identify any potential issues.



	username	password	client	reserve_time
1	user3	6aaf0f3bec7086820...	NULL	NULL
2	user2	10b42c279e2786da...	NULL	NULL
3	user4	655e92d367b4a3ad...	NULL	NULL
4	user5	12013e0cab2eba64...	NULL	NULL
5	user6	ed3de39c173d4326...	NULL	NULL
6	user7	e2e11e68f07d3317...	NULL	NULL
7	user1	7ab188723ec889b6...	NULL	NULL

Figure 5.1 - Database Records

To test the database setup, the database Python script was executed on laboratory computers. The database was then opened to verify its successful creation. (Figure 5.1)

Subsequently, the “server script” was executed on the laboratory computer, the embedded system was powered on and connected to the institute's network, and the application was installed on an external computer connected to the same network, acting as the user computer. It should be noted, that the successful completion of these operations is simultaneously confirmed by assessing its constituent operating parameters.

5.1.2 Login System and Database Testing

To assess the login stage, the database was deployed on the laboratory computer. To ensure that the login procedure works as intended multiple credentials were logged within the credential database. (Figure 5.1)

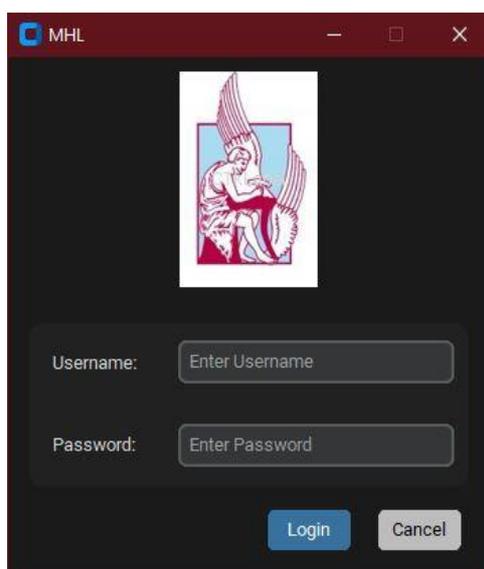


Figure 5.2 - Login Portal

Subsequently, the application was booted on the user's system. As shown in Figure 5.2 above, the database system was successfully prompted to the application portal. To test the validation procedure of the login credentials the following data shown below in- Testing Login Entries were logged.

Username	Password	Expected Result	Result
user1	tuc	ACCESS GRANTED	ACCESS GRANTED
user1	Aa2A1As	ACCESS DENIED	ACCESS DENIED
user2	tuc	ACCESS DENIED	ACCESS DENIED
user52	"Sd12A!as"	ACCESS DENIED	ACCESS DENIED
NULL	NULL	ACCESS DENIED	ACCESS DENIED

Table 3 - Testing Login Entries

The table indicates that all experimental results in column four, are identical to the expected results. This confirms that only when a correct set of credentials is logged (row1) access is granted. As row 3 is rejected, it means that the query correctly cross-matches the data as a set and not exclusively. The rejection of the entry in row 5 indicates that the application correctly identifies when no text is contained and therefore prohibits access.

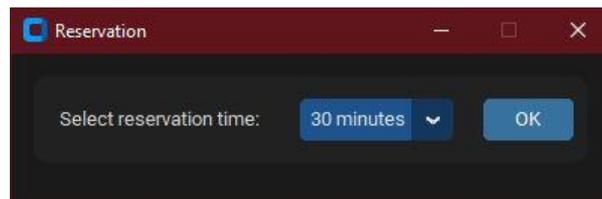


Figure 5.3 - Reservation Panel

Figure 5.3 illustrates the reservation panel which was prompted in the events where access was granted. To test the reservation stage five trials were executed. Each had a unique set of credentials from the database.

	username	password	client	reserve_time
1	user1	7ab188723ec889b6...	1	2023-08-03 12:58:0...
2	user3	6aaf0f3bec7086820...	NULL	NULL
3	user2	10b42c279e2786da...	2	2023-08-03 14:02:0...
4	user4	655e92d367b4a3ad...	4	2023-08-03 12:48:0...
5	user5	12013e0cab2eba64...	3	2023-08-03 12:34:3...
6	user6	ed3de39c173d4326...	NULL	NULL
7	user7	e2e11e68f07d3317...	NULL	NULL

Figure 5.4 - Database with Populated Fields

The first four did populate the database with the credentials and the reservation duration, as well as the microcontroller slot number. (Figure 5.4)

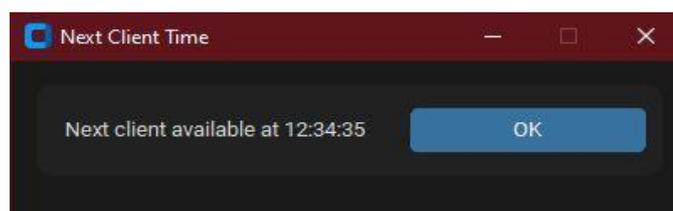


Figure 5.5 - Next Available Slot Panel

The fifth login attempt was denied access as intended, regardless of the correct set of credentials (Figure 5.5). It should be noted, that a sixth login trial was further executed, with the credentials of the first user. The software did identify that the database included the same credentials from the user, and it successfully skipped the reservation stage, thereby prompting the user to use the User Interface for the remaining duration of their reservation.

	username	password	client	reserve_time
1	user1	7ab188723ec889b6...	1	2023-08-03 12:58:0...
2	user3	6aaf0f3bec7086820...	NULL	NULL
3	user2	10b42c279e2786da...	2	2023-08-03 14:02:0...
4	user4	655e92d367b4a3ad...	4	2023-08-03 12:48:0...
5	user5	12013e0cab2eba64...	NULL	NULL
6	user6	ed3de39c173d4326...	NULL	NULL
7	user7	e2e11e68f07d3317...	NULL	NULL

Figure 5.6 - Removed Reservation

With the expiration of the user’s time duration, the slots were updated to be empty and available, whilst logging out the appropriate user. As shown in (Figure 5.6) “user5” no longer has a reservation.

	username	password	client	reserve_time
1	user1	7ab188723ec889b6...	1	2023-08-03 12:58:0...
2	user3	6aaf0f3bec7086820...	NULL	NULL
3	user4	655e92d367b4a3ad...	4	2023-08-03 12:48:0...
4	user5	12013e0cab2eba64...	NULL	NULL
5	user6	ed3de39c173d4326...	3	2023-08-03 15:35:3...
6	user7	e2e11e68f07d3317...	NULL	NULL
7	user9	cf80cd8aed482d5d...	NULL	NULL

Figure 5.7 – Adding & Removing Users

Lastly, as illustrated in (Figure 5.7) above running the appropriate scripts “addUser.py” adds (user9) or “removeUser.py” removes (user 2) users as needed.

5.1.3 File Transmission Testing

File transmission tests were conducted to ensure that the application allows users to select only the appropriate file types. During testing, it was confirmed that the conversion from a hex file to a binary format works as intended. When navigating the user interface, only the formats "hex" and "bin" were visible for selection, and selecting one of them made only that format available for selection.



Figure 5.9 - Control Panel Without a Selected File



Figure 5.8 - Control Panel with a Selected File

The automatic conversion was validated by observing that when a hex file is selected without a corresponding bin file, the UI displays the file with a .bin extension. The newly created file is now located in the same directory as the hex file, for future use. Additionally, the application's upload button was disabled before file selection (Figure 5.9) and became enabled after the selection (Figure 5.8) to enable users to upload their chosen files. This ensures that when pressing the upload button, a file to be uploaded is selected.

To assess the correct arrival of files at the Wi-Fi module, a text file was used, making it easier to verify data accuracy. Smaller files transmitted flawlessly, but for larger files, inconsistencies were observed. To address this issue, the chunk size was reduced, and a small delay was introduced in the Wi-Fi module between receiving and sending acknowledgments

for received data. Allowing time for the data to be processed correctly. Repeating the test demonstrated no data loss for larger files.

Considering that the files need to be passed to the master microcontroller, a similar test was conducted. This time the master microcontroller output was observed. As before, some adjustments are required due to inconsistencies when handling larger files. The necessary adjustments included adjusting the USART Baud Rate.

Furthermore, a test was performed to validate that only one user can upload at a time. By simulating two or more users attempting to upload different files simultaneously, it was observed that the files arrived at the Wi-Fi module in succession, demonstrating the proper handling of one user at a time policy.

5.1.4 Microcontroller Programming Testing

Following the file transmission tests, a series of programming tests were conducted to validate the operation of the slave microcontrollers. The tests aimed to ensure the programming method can handle various file types (provide different functionality) and sizes. Several programs were written and tested using the STK500 for both types of slave microcontrollers to assess their functionality before testing the system's programming method.

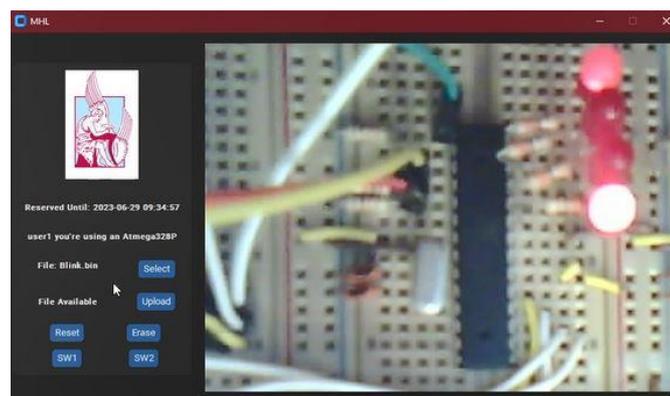


Figure 5.10 - Blink File LEDs OFF

Firstly, a file smaller than the master microcontroller buffer was uploaded using the UI to the user's designated microcontroller. By observing the output of the slave microcontroller, the successful programming process was confirmed. As illustrated in Figure 5.10 the expected

output is for the two middle LEDs to be off. After, a few milliseconds the expected output is for all the LEDs to turn on as illustrated below in Figure 5.11.

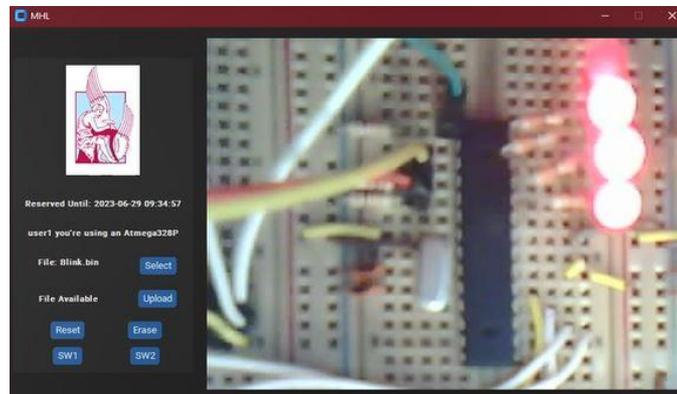


Figure 5.11 - Blink File LEDs ON

This initial test ensured that the system's basic programming functionality along with the file conversion above was working as expected.

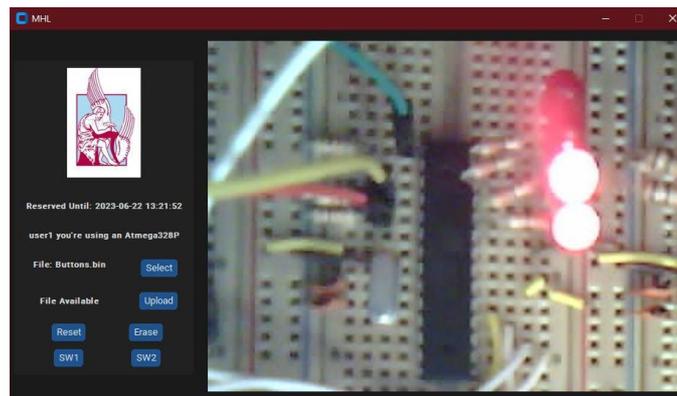


Figure 5.12 - Buttons Test File

Subsequently, interactive files that required user input, such as pressing buttons in the user interface and observing the output on the slave microcontroller, were uploaded (Figure 5.12). This test aimed to verify that the intergraded software buttons work properly and users could successfully send commands to the slave microcontroller without encountering any issues.

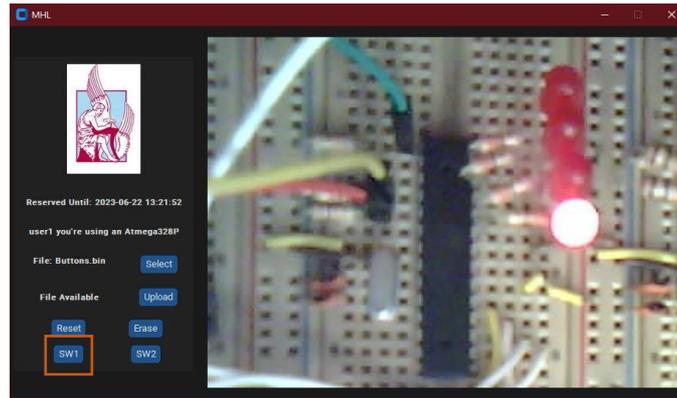


Figure 5.13 - Buttons File Button1 Pressed

Pressing the SW1 in the control panel is expected to turn off the upper LED, as can be observed in Figure 5.13. Similarly, pressing the SW2 from the initial state (Figure 5.12) the expected output is for the lower LED to turn OFF. (Figure 5.14)

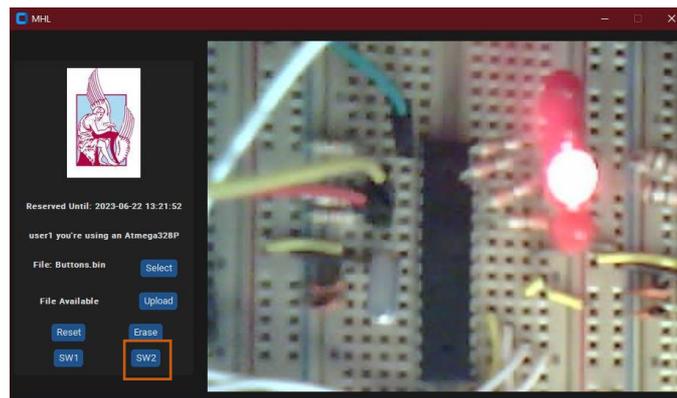


Figure 5.14 - Buttons File Button2 Pressed

The final file upload tests involved larger files, that exceed the master microcontroller buffer capacity by multiple times. These tests allowed the optimization of the USART and SPI configurations to handle programming with large files without overwhelming the master microcontroller.

5.1.5 Real-Time Camera Access Testing

The real-time camera access testing aimed to ensure that every user could access the camera stream, observe the LEDs status, and interact with the system without encountering significant delays. The system was connected to both the university network and a local network to compare performance under different network conditions. Throughout the testing, latency measurements were taken in various scenarios to evaluate the system's responsiveness.

To assess the latency when receiving frames from the ESP32-CAM, two factors were observed. Firstly, the time between uploading a program or sending a command and observing the change in the end user's UI was measured. Secondly, the minimal delay required for LED status changes to be noticeable by the end user.

For the first factor, the "Buttons" program was used, as it initially sets the microcontroller LEDs to a static stage, that allows changes to be observed. Further, it supports button pressing, allowing measurement of the time between an action in the UI (e.g., pressing a button) and the change in the video stream. For the second factor, the "MemoryRead" program was suitable, as it consistently changes the LEDs every 0.5 seconds. While the actual minimal delay cannot be measured precisely due to network intricacies. It should be more than 15ms since that is the frame interval used by the ESP32-CAM, and by observing the output of the "MemoryRead" program it does not require more than 0.5 seconds. Also, the results of the tests that were conducted on both the university network and a local network showed lower latency and support for better quality in the local network.

Moreover, the smooth frame delivery was assessed while multiple users connected to the system or log out. The test confirmed that the stream for other users remains stable and unaffected, maintaining a consistent user experience. Despite the increased latency and lower quality, the experience remains identical in both network conditions. With minimal frame losses and the UI remain responsive even when occasional frame losses occurred.

5.1.6 Multi-Microcontroller Integration Testing

To ensure the success of multi-microcontroller integration, testing focused on verifying whether each user could upload different programs to their assigned microcontroller and interact with it. Additionally, it was essential to assess whether running all the microcontrollers simultaneously would create any stability issues and affect the system's overall responsiveness.

For the initial test, all users were logged in and each user uploaded the same file in succession. All users initially uploaded the same file to ensure that every microcontroller performs the same.

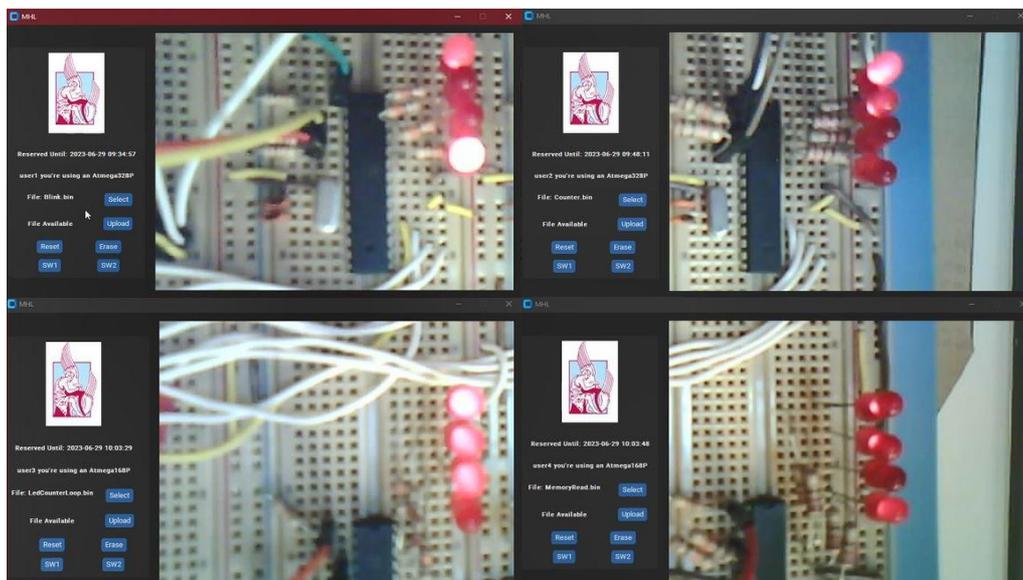


Figure 5.15 - All Users Upload Different Files

After validating the microcontroller's correct operation, users proceeded to upload different codes to their corresponding slave microcontroller individually, in pairs, and finally, all together (Figure 5.15). This test validated that the system can handle multiple users trying to program their slave microcontroller also, that each slave microcontroller can be programmed and controlled individually.

5.2 System Evaluation

The system was evaluated through a series of tests designed to utilize all available resources. These tests included writing several programs, as well as testing the system's ability to reset and erase each microprocessor.

In a home network environment, the system demonstrated promising results. With only a minor delay associated with the ESP32-CAM video stream, being noticed. The upload time for large files, which was a concern, proved to be a non-issue. Even when multiple users attempted simultaneous uploads, the delay was not significant enough to cause problems.

Connecting the system to the university's network, to access the system remotely users need to connect via VPN or be connected to the university's network. The tests included both cases and revealed a noticeable increase in the stream delay. The delay was measured manually between a button press and visual feedback using a digital timer and it was around 2 seconds. Adjustments were made to the stream quality and frame buffer settings to reduce this delay, resulting in minor improvements. While the stream delay can be noticeable, it does not affect the other functionalities of the system.

During testing, some inconsistencies were detected regarding expiration times. These irregularities occur when a user logs in right before the reservation expiration time. In this case, the user is occasionally logged out automatically, instead of being prompted to the user interface. In such cases, a second attempt after a brief delay would solve the issue. Similarly, if the system resets, the first attempt to program a microcontroller afterward may not be successful. In this case, a consecutive attempt was found to be successful.

Additionally, regarding the functionality of the platform, the slave microcontroller only gets erased without users input when an active user session is ongoing and the reservation expires. If the user is not logged in during the expiration the microcontroller will be erased when the microcontroller is assigned to another user. Further, the application when packeted into an executable has issues due to the utilization of the "CustomTkinter" package.

The most significant reliability concern is the streaming quality, which could be improved with better hardware or adding an external antenna to the camera module to boost the connection with the university's network. This could enhance the streaming quality and overall user experience with the system.

Chapter 6 Conclusion and Future Work

6.1 Conclusion

This thesis aimed to address the challenge of multi-user access within an educational system, and it focused on enhancing an existing system to support multiple users. Thereby it provided students with a more hands-on experience that would otherwise be limited by the availability of hardware and the requirement of their physical presence. The study successfully implemented a user-friendly UI that supports login functionality and a reservation system which allocates microcontrollers based on availability. Furthermore, camera streaming was integrated successfully to enhance the user experience by providing real-time visual feedback.

Within experimentation, tests deemed the overall operation of the system successful, including the log-in and reservation slot feature of the user interface. Further, a successful upload of .bin files was observed, with a simultaneous real-time video feed from the webcam. Some instabilities were detected within the log-in and reservation phase of the system, with unsuccessful login and undesired termination of the application within the user system. However, most minor inconsistencies were found to be due to DATETIME configuration errors, which only necessitate alterations within the setup code and therefore do not countervail the overall feasibility of the embedded system.

Consequently, the insights gained from this paper displayed the potential of this configuration within the embedded-system realm. It demonstrates adequate flexibility and scalability, fulfilling basic prerequisites entailed within any educational medium. Further, its user-centric design, accompanied with future developments within the field can revolutionize the way students engage with educational platforms whilst also acquiring practical familiarity with embedded systems.

6.2 Future work

This research is a stepping stone towards an inclusive education regarding embedded systems. Various future works could enhance the capabilities of this system, whilst simultaneously diminishing its limitations and vulnerabilities. Initially, some additional tests are required to further assess the feasibility of the system. This includes tests regarding microcontroller functionalities, such as analog-read and write, and pulse-width-modulation. To accommodate those functionalities, it is important that a communicative medium between the microcontroller and the user interface to be developed. Therefore, it is suggested that a virtual terminal is added to the user interface, returning data from the microcontroller.

Further, future research could focus on eliminating the requirement for an additional computer to run the server and host the database. This could be achieved by using a Raspberry Pi, which offers better camera and Wi-Fi capabilities than the ESP32 and even supports an Ethernet connection that could improve performance and streaming quality. A display for printing messages would also be a welcome addition, either physically or within the UI.

Finally, additional modules can be added to the platform in the future to accommodate the specific needs of different faculties. An effective feature could also be the automatic detection of the slave-microcontroller model which will allow the removal and addition of microcontrollers from the system without the need for system changes. It should be noted, that alterations are also required within the setup stage for the system to enable the handling of additional users and microcontrollers. Additionally, the server must also be changed to allocate the users evenly along all available systems, and the database needs to withhold more information to differentiate the systems (e.g., IP addresses for each system and the number of available microcontrollers). Parallely, the development of a separate admin panel could improve the overall management of the platform allowing the monitoring of different systems, the moderation of users, and the testing of new modules before deploying them.

Appendix

7.1 Hardware Schematics

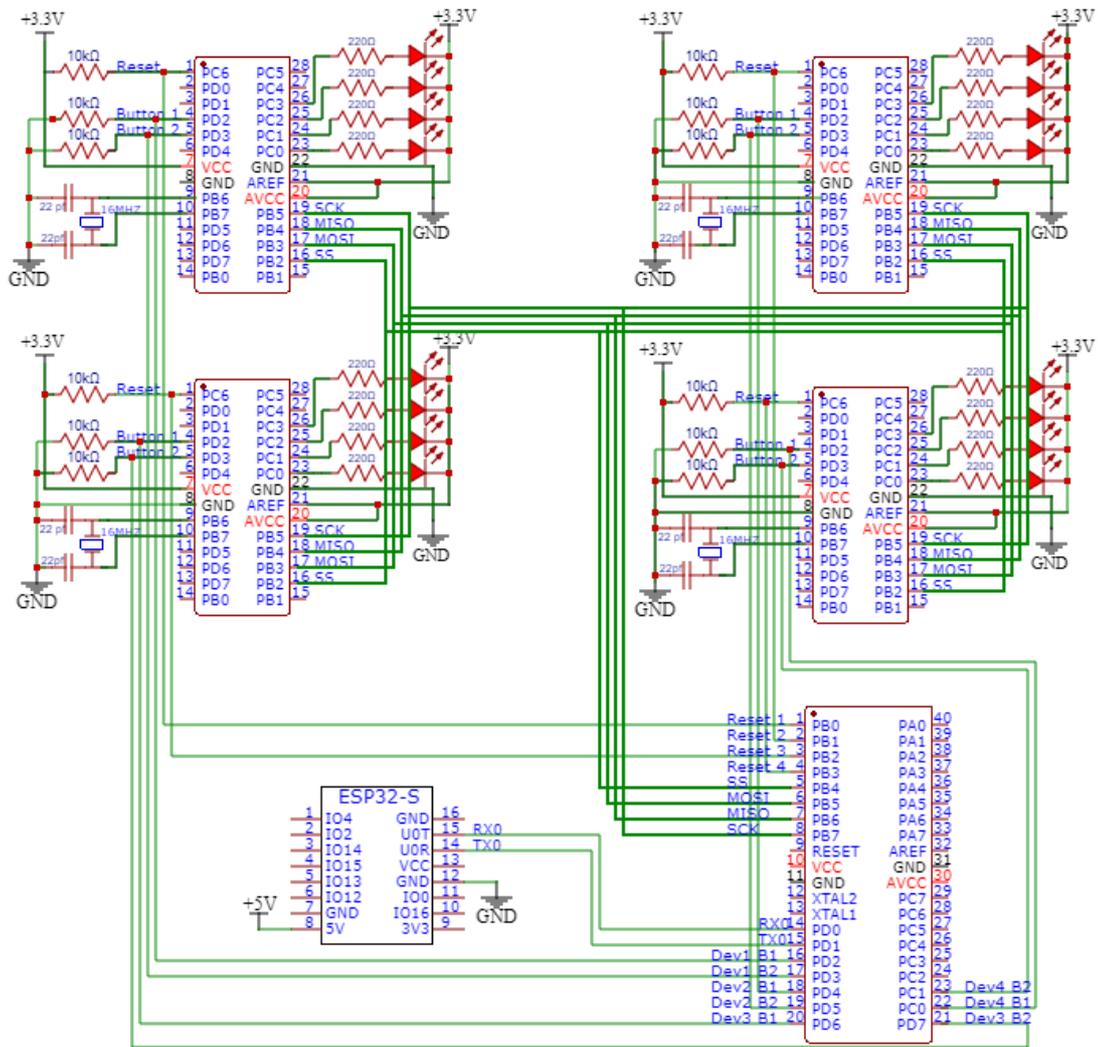


Figure 0.1 - Hardware Schematics

7.2 Example Applications

The following examples were used to test and validate the correct operation of the system. The hex file of each program is provided within the app to allow users to check if their microprocessor is operating as expected.

```
#define F_CPU 16000000UL

#include <avr/io.h>           /* Include AVR std. library file */
#include <util/delay.h>       /* Include Delay header file */

/* This program is used to set the LEDs on Port C 0 and 3 to ON And 1 2 to OFF
wait for 1 Second and set All the LEDs ON wait for 1 second and repeat*/
int main(void)
{
    // Set Port C pins 0, 1, 2, and 3 as outputs
    DDRC |= 0b00001111;

    while (1) {
        PORTC = 0x06; // Set PORTC pins 1 and 2 to high (0110)
        _delay_ms(1000); // Delay 1 sec

        PORTC = 0x00; // Set PORTC pins 0, 1, 2, and 3 to low (0000)
        _delay_ms(1000); // Delay 1 sec
    }
}
```

Figure 0.2 - Blink

The Blink program sets PortC (Pins 0-3) as outputs and then turns on and off the two LEDs in the middle every 1 sec. It is a small and simple program to test that everything works before using the microcontroller.

```
#include <avr/io.h>

int main(void)
{
    // Set pins D2 and D3 as inputs
    DDRD &= ~(1 << PD2);
    DDRD &= ~(1 << PD3);

    // Set pins C0 and C1 as outputs
    DDRC |= (1 << PC0);
    DDRC |= (1 << PC1);

    while(1)
    {
        // Check if button on D2 is pressed
        if (PIND & (1 << PD2))
        {
            PORTC |= (1 << PC0); // Turn on LED on C0
        }
        else
        {
            PORTC &= ~(1 << PC0); // Turn off LED on C0
        }

        // Check if button on D3 is pressed
        if (PIND & (1 << PD3))
        {
            PORTC |= (1 << PC1); // Turn on LED on C1
        }
        else
        {
            PORTC &= ~(1 << PC1); // Turn off LED on C1
        }
    }

    return 0;
}
```

Figure 0.3 - Buttons

The “Buttons” program (Figure 0.3) is designed to validate the functionality of the software buttons that are connected to Port D (pins 2-3). The program sets these pins as inputs and continuously checks their state in a loop. When a button is pressed, the corresponding LED on Port C (either pin 0 or 1) is turned on. If the button is not pressed, the LED remains in its default stage. This program is also used to inform the users on how to detect button presses.

The “LedCounterLoop” program is a simple LED chaser. It sequentially turns on and off four LEDs connected to pins PC0-3 of the microcontroller.

The “Counter” program operates as a binary counter using four LEDs connected to the microcontroller and is designed to test that loops are used correctly. It continuously counts from 0 to 15 and displays each count in binary form using LEDs. Each number displayed for half a second before moving to the next number.

The “MemoryRead” program is designed to test the microcontroller's ability to access program memory. It uses a memory array to create a pattern and show it utilizing the LED lights. The program sets PortC (pins 0-3) as outputs then it cycles through each element in the array. Each element represents a different pattern of on and off states for the LEDs. The program reads the pattern from program memory, applies it to the LEDs, waits for half a second, and then moves on to the next pattern. This process repeats indefinitely, creating a looping light show on the LEDs.

The “Morse Code” program is designed to check whether a larger program that has multiple functions and is memory-based can be uploaded without any problems. The program translates the phrase "HELLO WORLD" into Morse code and displays it using LEDs. It continuously loops through each character in the phrase, converts it to Morse code, and represents the Morse code as a series of short and long blinks on the LEDs. After displaying the entire phrase, it repeats the process.

The “LightShow” program is the larger program that was tested and is designed to test the microcontroller's watchdog timer functionality and its ability to handle interrupts. It uses the LEDs to display a random patterns creating that way a light show. The watchdog timer is set to trigger an interrupt every 250ms. When the interrupt is triggered, a counter is incremented. Once the counter reaches 4 (which equates to 1 second), a new random pattern is generated and the counter is reset. The LEDs are then updated to reflect the new pattern. This process repeats indefinitely, creating a continuously changing light show on the LEDs.

7.3 Parts List

- 1 x STK500 equipped with an ATMEGA32A
- 2 x ATMEGA328P
- 2 x ATMEGA168P
- 1 x ESP-12E Wi-Fi Module not used in the final iteration
- 1 x ESP32-S Wi-Fi Module Replaced ESP12-E
- 1x 11MHz Quartz Crystal Oscillator
- 4 x 16MHz Quartz Crystal Oscillators
- 8 x 22uf capacitors
- 16 x Red LEDs
- 16 x 330 Ω Resistors
- 12 x 10 k Ω Resistors
- 1 x Ai Thinker ESP32-CAM Camera Module
- 2 x MB102 Breadboard Power Supply Module 3.3V / 5V

References

- [1] L. Emmanouil, Design and Implementation of an Integrated System for Logic Design exercises, 2020.
- [2] R. H. a. J. Riddley, "FPGA e-Lab, a technique to remote access a laboratory to design and test," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, 2007.
- [3] A. A. e. al., "Wireless ATMEL AVR In-Circuit Serial Programmer based on Wi-Fi and ZigBee," in *2020 16th International Computer Engineering Conference (ICENCO)*, 2020.
- [4] P. A. e. al., "Development and Application of Remote Laboratory for Embedded Systems Design," in *Proceedings of 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 2015.
- [5] T. Michalec, M. Wojczuk, R. Brzoza-Woch and T. Szydło, "Remote Programming and Reconfiguration System for Embedded Devices," in *Proceedings of the 2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Leipzig, Germany, 1–4 September 2019.
- [6] N. Abekiri, A. Rachdy, M. Ajaamoum, B. Nassiri, L. Elmahni and Y. Oubail, "Platform for hands-on remote labs based on the ESP32 and NOD-red," *Science Africa*, vol. 19, no. e01502, 2023.
- [7] D. L.-d.-i. a. P. O. J. García-zubía, "Evolving towards better architectures for remote laboratories: a practical case," *International Journal of Online Engineering*, vol. 1, no. 2, 2005.
- [8] A. Kalliontzis, Design and Implementation of an Embedded System for Remote Access of Microcontroller of University Laboratory Exercises, Under review.
- [9] "AVR910: In-System Programming," 2016.
- [10] "CustomTkinter UI-Library".
- [11] "ESP-12E WiFi Module," 2015.
- [12] "ESP32-CAM Camera Module," 2017.
- [13] "AVR STK 500," 2003.
- [14] "Install pip on windows".