Pharos University in Alexandria

Faculty of Engineering

Computer Department

WI-VGA
(Smart VGA monitor for Advertising Applications)

Supervised by
Dr. Hossam Abdelbaki

Presented By

Ahmed Waleed Abdelhamid Mohamed Abdeen
Abd Elrahman Mohamed Atia
Ahmed Saeed Hamed Awad
Ahmed Yasser Elsaed Mahmoud Aboshabana
Mohamed Ahmed Basyoni Salem
Ahmed Abdelazim Elwani Elgamal

2023

Table of content

Table of figure

Acknowledgment

I would like to express our deepest gratitude and appreciation to Dr. Hossam Eldin Mostafa for his invaluable guidance, unwavering support, and exceptional supervision throughout the duration of our project.

Dr. Hossam Eldin Mostafa's extensive knowledge and expertise in the field of embedded systems have been instrumental in shaping the direction and success of this project. His insightful feedback, constructive criticism, and valuable suggestions have significantly enhanced the quality of our work.

Furthermore, I am truly grateful for Dr. Hossam Eldin Mostafa's dedication and commitment to our development as a researcher. His patience, encouragement, and willingness to answer our countless questions have been indispensable in fostering a conducive learning environment.

I would also like to acknowledge Dr. Hossam Eldin Mostafa's remarkable organizational skills and his ability to provide timely guidance. His consistent availability and prompt response to our queries have been crucial in overcoming challenges and ensuring the smooth progress of the project.

I am sincerely grateful for the opportunity to work under Dr. Hossam Eldin Mostafa's supervision. His mentorship has not only contributed to the successful completion of this project but has also enriched our understanding of embedded systems and instilled in me a passion for research.

Finally, I would like to extend our heartfelt thanks to Dr. Hossam Eldin Mostafa for his trust and belief in our abilities. His constant motivation and belief in our potential have been a tremendous source of inspiration, pushing me to strive for excellence.

Abstract

This graduation project aims to develop a system for advertising products through presentation images using the ESP32 microcontroller. The project involves designing and implementing a low-cost, portable, and easy-to-use device that can display high-quality images of products to potential customers. The ESP32 will serve as the main controller for the system, with a high-resolution display and onboard memory for storing and displaying images. The system will also include a user-friendly interface for uploading and managing images. The project's main goal is to provide a cost-effective solution for businesses to showcase their products, improve their marketing efforts, and increase sales. The results of the project will be evaluated through user testing and analysis of its impact on the targeted market.

The increasing competition in the market has made it essential for businesses to adopt effective marketing strategies to showcase their products to potential customers. One of the most effective ways of advertising products is through the use of visual media such as presentation images. However, the cost of developing and implementing a high-quality display system can be quite expensive, especially for small businesses.

To address this challenge, this graduation project proposes the development of an affordable and portable device for presenting product images using the ESP32 microcontroller. The ESP32 offers an excellent platform for building such a device, as it provides a powerful processing capability, high-resolution display, and onboard memory for storing and displaying images.

The proposed system will be designed to be user-friendly, allowing businesses to easily upload and manage their product images on the device. The system will also provide the flexibility to customize the presentation of the images based on the product type and target audience. By providing a cost-effective and user-friendly solution for advertising products, the system is expected to improve the marketing efforts of businesses and increase their sales.

To evaluate the effectiveness of the proposed system, user testing will be conducted to determine its ease of use and impact on the targeted market. Additionally, the system's performance will be analyzed in terms of its ability to display high-quality images, support customization, and provide a cost-effective solution for businesses. The results of the project will contribute to the advancement of visual media advertising techniques and provide valuable insights for future research in this field.

Chapter One: Introduction

1.1 Background

Through our knowledge and observations, we designed a board based on ESP32 that can display text or images on advertising screens; our prototype will ease the operation of change controlling the displaying advertising screens. Each screen will connect to one ESP32 and will be connected through WIFI and controlled via a central interface.

In the 1830s, a man named Jared Bell created some of the very first billboards. These billboards advertised circus acts like Barnum And Bailey. The billboards were often large posters, displaying colorful pictures. They emphasized unique features and promotional elements for each of their upcoming shows.

The 1860s saw a major shift in billboard advertising. Businesses were able to purchase outdoor space to utilize for billboard display. Soon enough, advertisers began taking advantage of the new laws and regulations, setting up different billboards and experimenting with their own forms and creatives. They were often hand-drawn or hand-painted and labor-intensive.

History of billboards:

Billboards have a long and fascinating history that can be traced back to ancient civilizations. The ancient Egyptians used papyrus to create posters and announcements, which they hung on public walls and buildings. Later, the Romans used similar techniques to promote events, public services, and political campaigns. In the Middle Ages, town criers were the main source of advertising, shouting out announcements and advertisements to the public. During the Renaissance, print advertising became popular, and posters were hung on walls and buildings to promote products and services.

The modern billboard, as we know it today, was developed in the late 19th century. In 1835, the first large outdoor advertising poster was created in Paris, France. The poster, which advertised a play, was placed on a wall and measured over 50 feet in length. In the United States, the first billboard was erected in 1867 in New York City. The billboard measured over 50 feet in length and advertised a circus. Over the years, billboards became a popular form of advertising, with companies using them to promote products and services across the country. Today, billboards remain an important part of the advertising industry, with new technologies such as LED screens and digital billboards expanding their reach and impact.

Our proposed solution allows us to keep track of all the displays that are connected to the system with the ability to add or remove screens according to customer needs.
The system gives us the ability to organize the ads with specific order and time delay needed.
The graduation project's main goal is to enable controlling advertising or presentation screens.
In this project, we work on having a complete integrated system that increases productivity, a product that reduces wasting time by using regular ways and is made very usable for everyone who accesses it via our applications on different platforms.

1.2 The problem description

One of the main challenges faced by manufacturers and retailers of goods, particularly in the case of new products, is the problem of product display. Advertising is a vast industry that greatly influences people and the purchasing decisions of consumers. Due to the fact that screens and advertising platforms are currently some of the most significant advertising instruments, we have focused on that field.

Due to the availability of many functions in ESP32 without the use of any external additions and taking into consideration the issue of cost and availability in stores, the goal of this project is to control screens that now exist in most stores and display items and advertising platforms.

Some of the main problems with billboards:
High costs for brief exposure
Does not target a specific market
Short-term advertising tool
Limited information
Time insensitive

1.3 Project cost:

The project's cost mainly concerns the hardware part. Everyone on the team contributed equally to the project expenses. Cost is focused on the cost of the ESP32 ship, resistances,SD module, SD card

1.4 Achievements

- Compare different farmworks for ESP32.
- Able to interface SD memory using SPI protocol.
- Able to use a file system API that is made by ESPRESSIF.
- Able to use Flash with a file system.
- We were successful in hosting an FTP server for your ESP32 and SD memory.
- Editing image using file robot-image-editor
- Web application connects to ESP32 ship to upload and modify uploaded files.
- We were able to decode jpg-png images.
- Images display on a monitor only using ESP32.
- Application to control esp32

Chapter Two: Methodology

1. Why ESP-32:

1.1. Abstract

The ESP32 is a powerful and flexible microcontroller that has gained significant popularity in the maker and embedded systems communities in recent years. This research paper will explore the reasons why the ESP32 has become a popular choice for a variety of projects, including IoT devices, robotics, and automation.

1.2. Introduction

The ESP32 is a low-cost, low-power, and highly versatile microcontroller developed by Espressif Systems. It was released in 2016 and has since become one of the most popular microcontrollers for makers and embedded systems enthusiasts. The ESP32 is built around a dual-core Tensilica LX6 microprocessor, clocked at up to 240 MHz. It features 520KB of SRAM, 4MB of flash memory, and Wi-Fi and Bluetooth connectivity.

1.3. Reasons for Popularity

- Versatility
  One of the primary reasons why the ESP32 has gained so much popularity is its versatility. The ESP32 can be used for a wide range of applications, including IoT devices, robotics, automation, and more. The microcontroller is capable of handling complex tasks, and its Wi-Fi and Bluetooth connectivity makes it easy to communicate with other devices.

- Low Cost
  Another reason why the ESP32 has become so popular is its low cost. The ESP32 is significantly cheaper than other microcontrollers with similar capabilities, such as the Arduino Due or the Raspberry Pi. This low cost makes it accessible to a wider audience, including hobbyists, students, and startups.

- Power Efficiency
  The ESP32 is also known for its power efficiency. The microcontroller features a deep sleep mode that allows it to consume very little power when it is not in use. This makes it an excellent choice for battery-powered devices and other low-power applications.

- Development Support
  The ESP32 has a large and active community of developers and users. This community has created a wide range of libraries, examples, and tutorials to help

others get started with the microcontroller. Additionally, there are many third-party development boards and accessories available for the ESP32, which makes it easy to get started with the microcontroller.

- Built-In Features

The ESP32 also has many built-in features that make it an attractive option for a variety of projects. For example, the microcontroller has a built-in touch sensor, a temperature sensor, and a hall effect sensor. It also has a built-in DAC, which makes it easy to output audio signals.

1.4. ESP32 vs NodeMCU

Here is a table comparing some of the key features of the ESP32 and NodeMCU:

FeatureFeature	ESP32	NodeMCU
Microcontroller	Dual-core Tensilica LX6	Single-core ESP8266
Clock Speed	Up to 240 MHz	Up to 160 MHz
Memory	520 KB SRAM, 4 MB Flash	80 KB SRAM, 4 MB Flash
Wi-Fi	IEEE 802.11 b/g/n	IEEE 802.11 b/g/n
Bluetooth	Bluetooth v4.2 BR/EDR and BLE	Not built-in
Pins	Up to 36 GPIO pins	Up to 11 GPIO pins, some shared with Flash
Analog Inputs	18 ADC channels (12-bit resolution)	1 ADC channel (10-bit resolution)
Digital Outputs	20 PWM channels (resolution up to 16)	1 PWM channel
Operating Voltage	2.2V - 3.6V	3.3V
Price	$3.00 - $10.00	$3.00 - $7.00 (depending on model)
Size	18 x 25.5 mm	25 x 50 mm
Development	C, C++, MicroPython, Arduino IDE	Lua, Arduino IDE
ADC	Up to 18 channels with 12-bit resolution	Up to 10 channels with 10-bit resolution

DAC	Up to 2 channels with 8-bit resolution	Not available
I2C	Up to 2 channels	Up to 1 channel
SPI	Up to 4 channels	Up to 2 channels

Note that some of the features, such as the amount of RAM and flash memory, can vary depending on the specific model of ESP32 or NodeMCU being used. Overall, the ESP32 offers more processing power and a wider range of capabilities, including Bluetooth and Bluetooth LE support. However, the NodeMCU may be a better choice for projects that only require Wi-Fi connectivity, and it can be less expensive than the ESP32.

1.5. Conclusion

The ESP32 is a powerful and versatile microcontroller that has gained significant popularity in the maker and embedded systems communities. Its low cost, power efficiency, and built-in features make it an attractive option for a wide range of projects, while its active development community provides ample support for new users. With its combination of power, flexibility, and ease of use, it is no wonder why the ESP32 has become a go-to choice for many developers and makers.

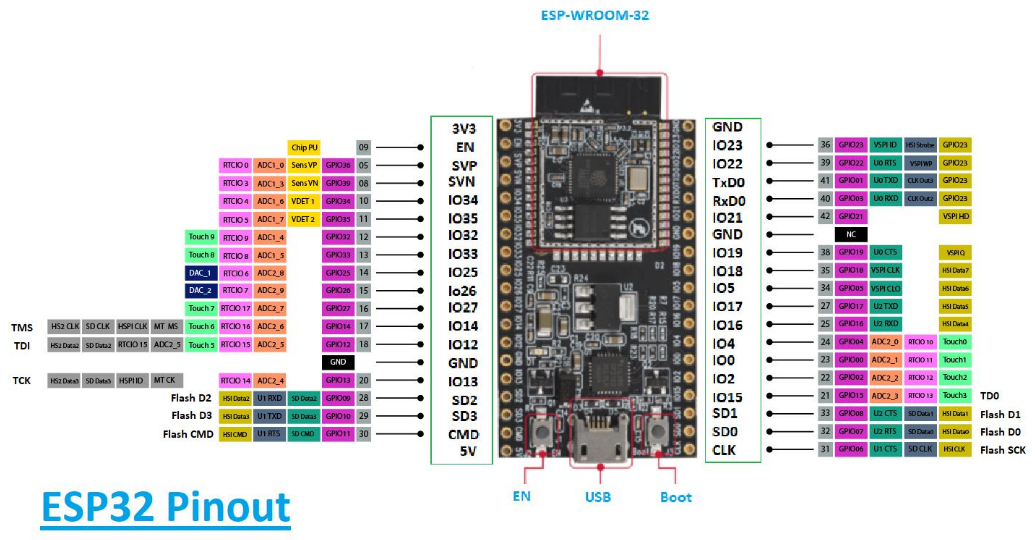

Figure 1: esp32 pinout

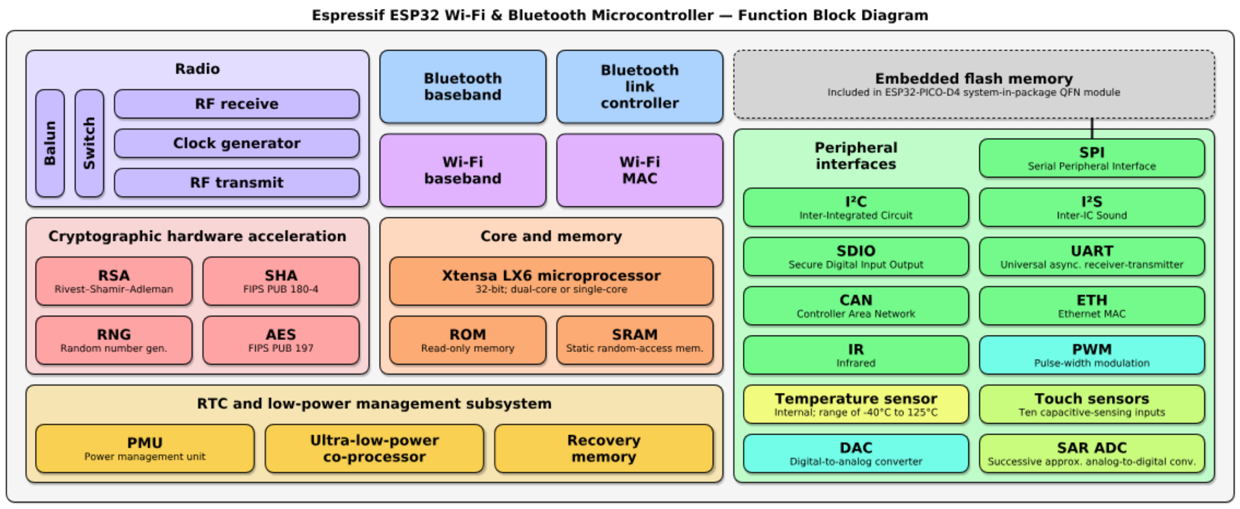
Espressif ESP32 Wi-Fi & Bluetooth Microcontroller — Function Block Diagram
Radio
Balun
Switch
RF receive
Clock generator
RF transmit
Bluetooth baseband
Bluetooth link controller
Wi-Fi baseband
Wi-Fi MAC
Embedded flash memory
Included in ESP32-PICO-D4 system-in-package QFN module
Peripheral interfaces
SPI
Serial Peripheral Interface
I²C
Inter-Integrated Circuit
I²S
Inter-IC Sound
SDIO
Secure Digital Input Output
UART
Universal async. receiver-transmitter
CAN
Controller Area Network
ETH
Ethernet MAC
IR
Infrared
PWM
Pulse-width modulation
Temperature sensor
Internal; range of -40°C to 125°C
Touch sensors
Ten capacitive-sensing inputs
DAC
Digital-to-analog converter
SAR ADC
Successive approx. analog-to-digital conv.
Cryptographic hardware acceleration
RSA
Rivest-Shamir-Adleman
SHA
FIPS PUB 180-4
RNG
Random number gen.
AES
FIPS PUB 197
Core and memory
Xtensa LX6 microprocessor
32-bit; dual-core or single-core
ROM
Read-only memory
SRAM
Static random-access mem.
RTC and low-power management subsystem
PMU
Power management unit
Ultra-low-power co-processor
Recovery memory

Figure 2:esp32 Chip Function Block Diagram

2. Design Diagrams

2.1. USE CASE Diagrams

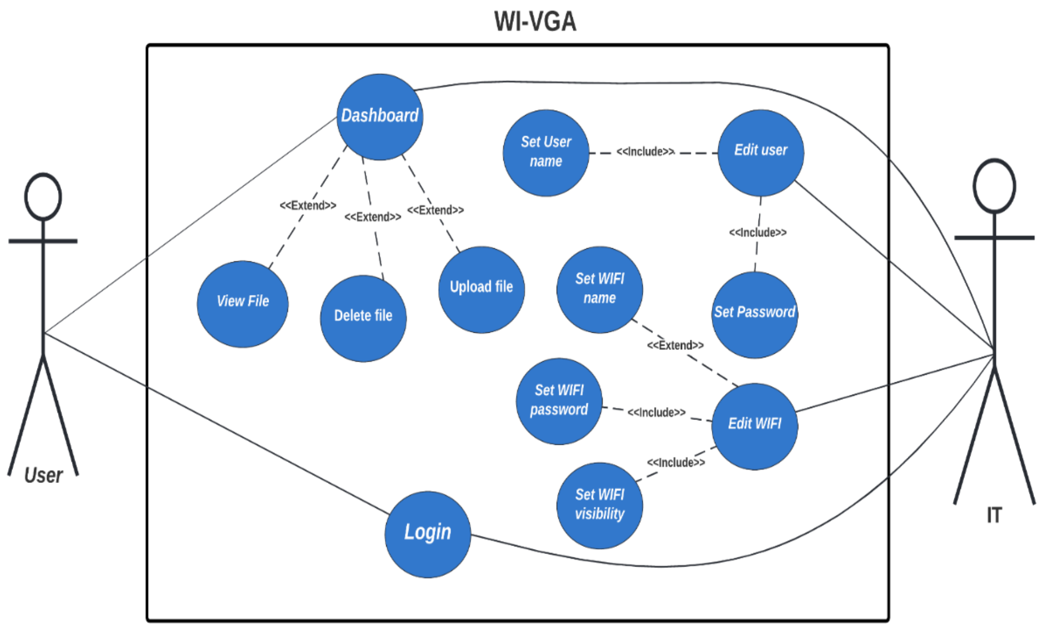

Figure 3:USE CASE Diagrams

2.2. Sequence Diagram of login

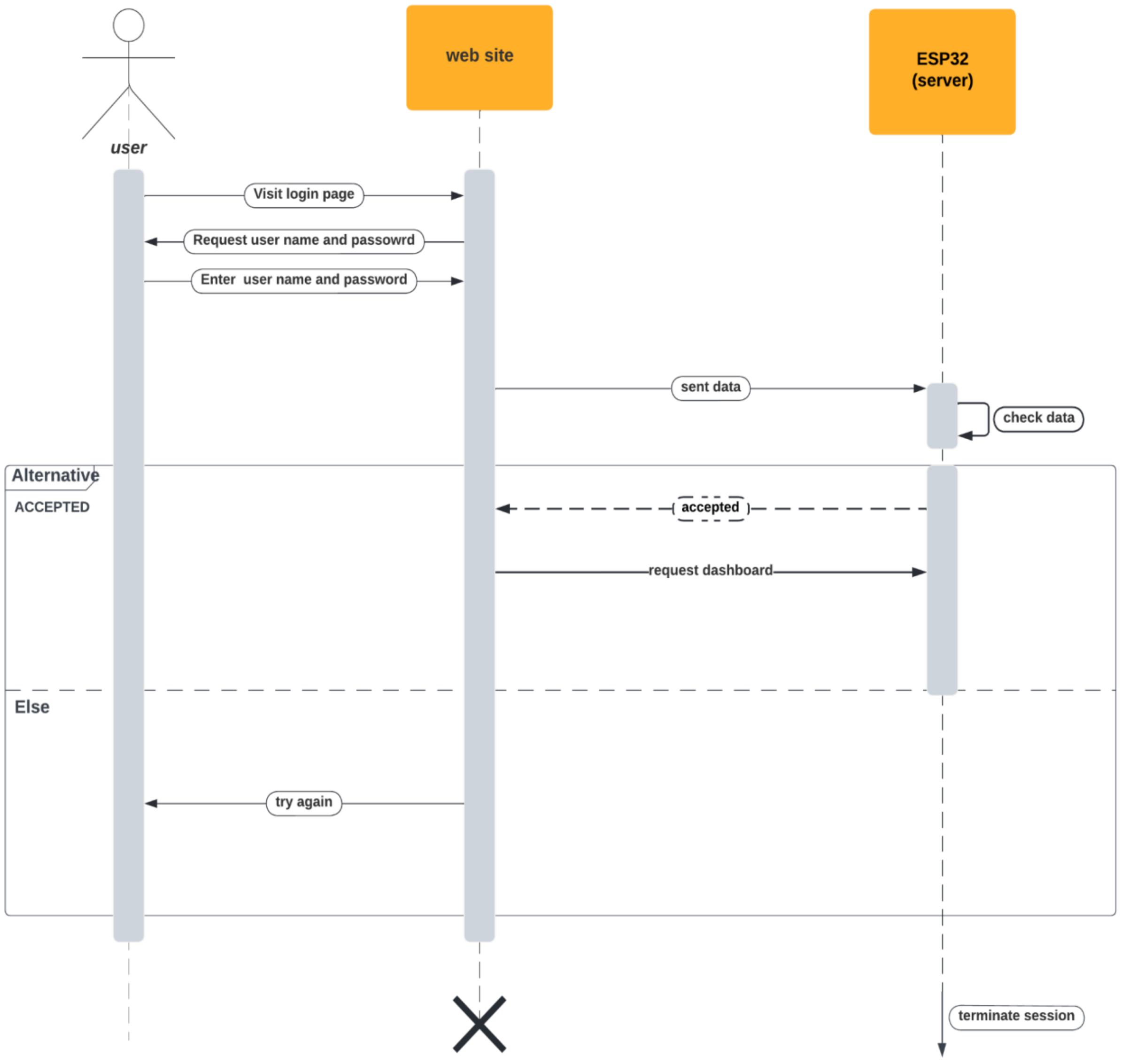

Figure 4:Sequence Diagram of login

2.3. Sequence Diagram of upload

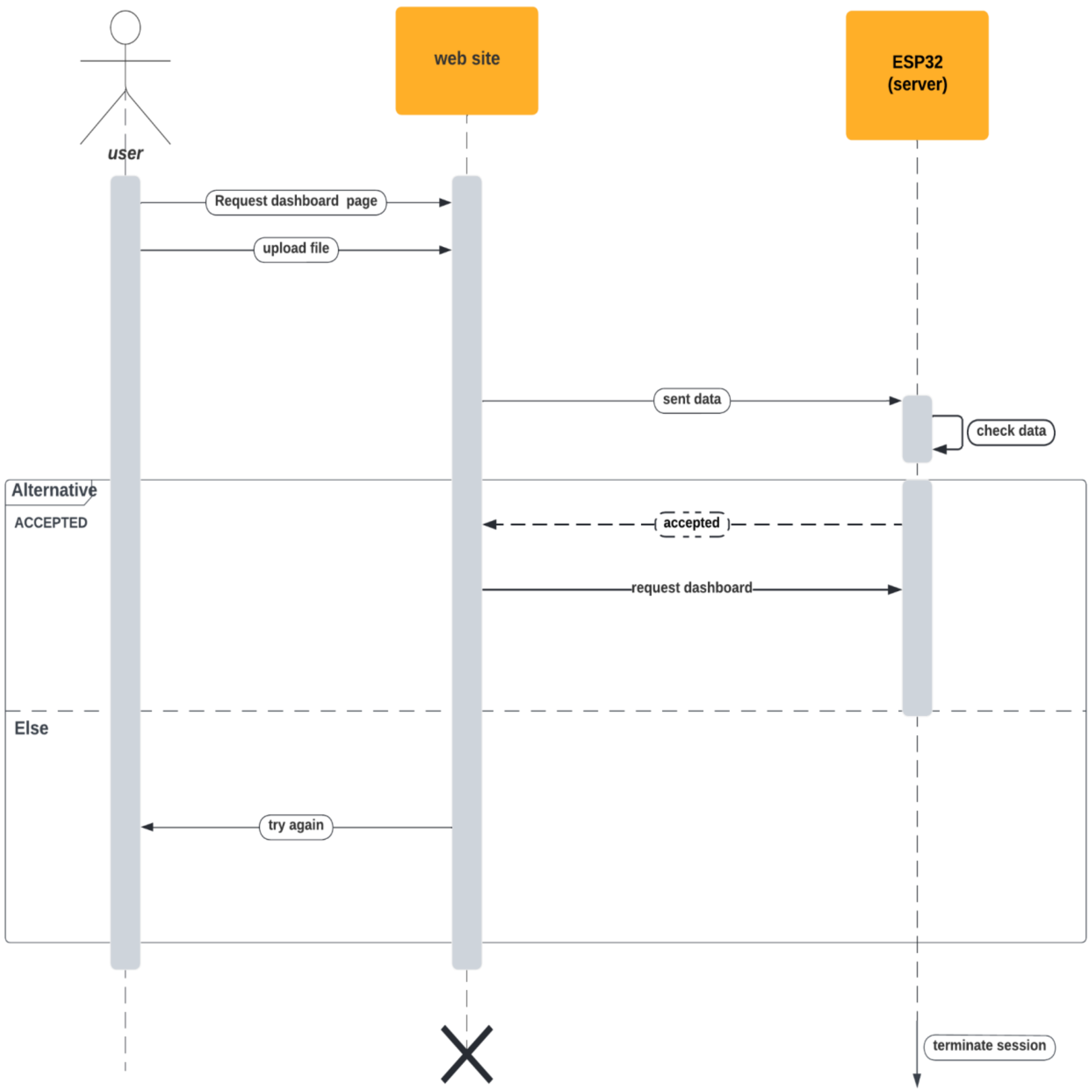

Figure 5:Sequence Diagram of upload

2.4. Sequence Diagram of delete

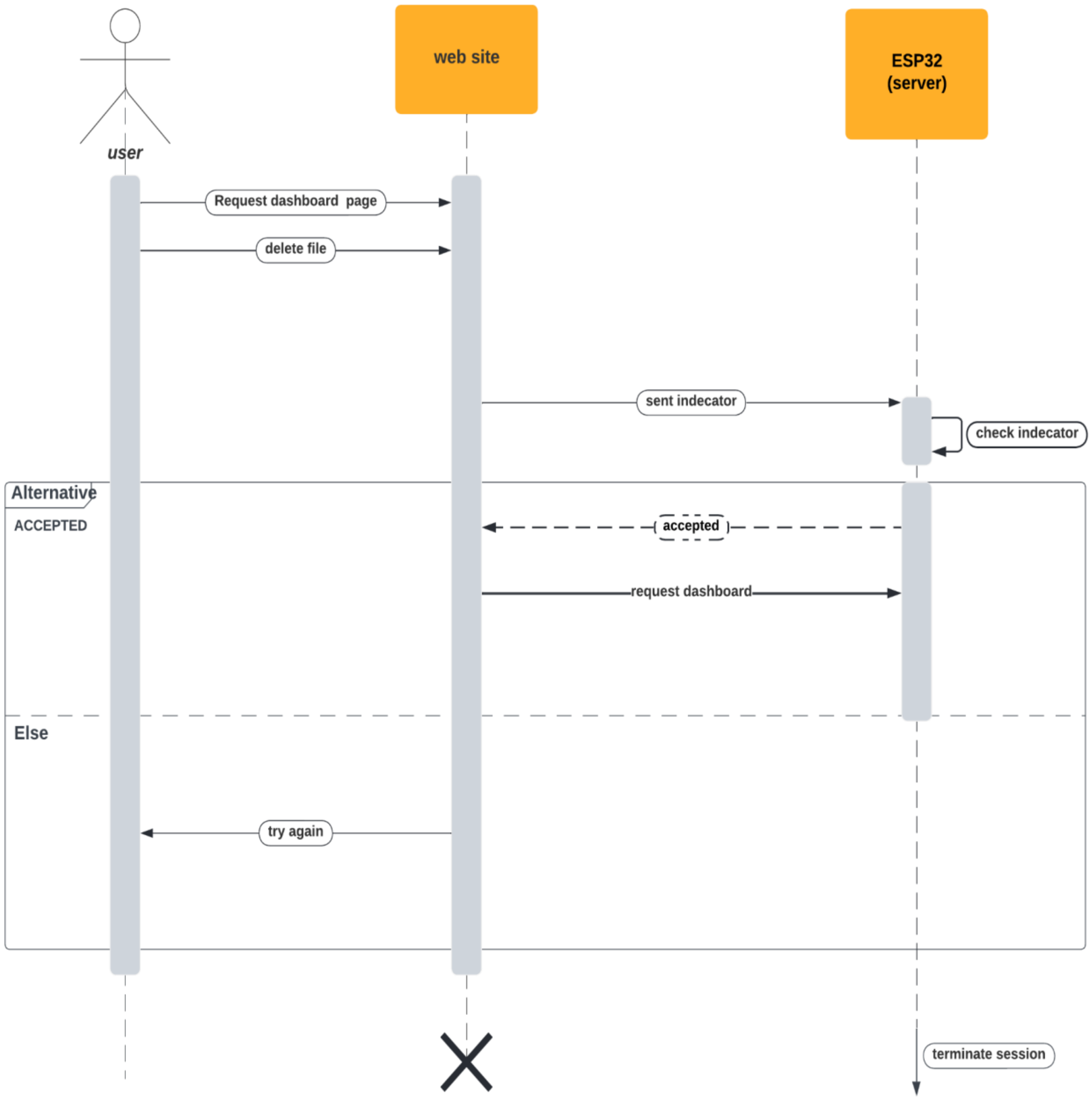

Figure 6:Sequence Diagram of delete

2.5. Sequence Diagram of Wi-Fi edit

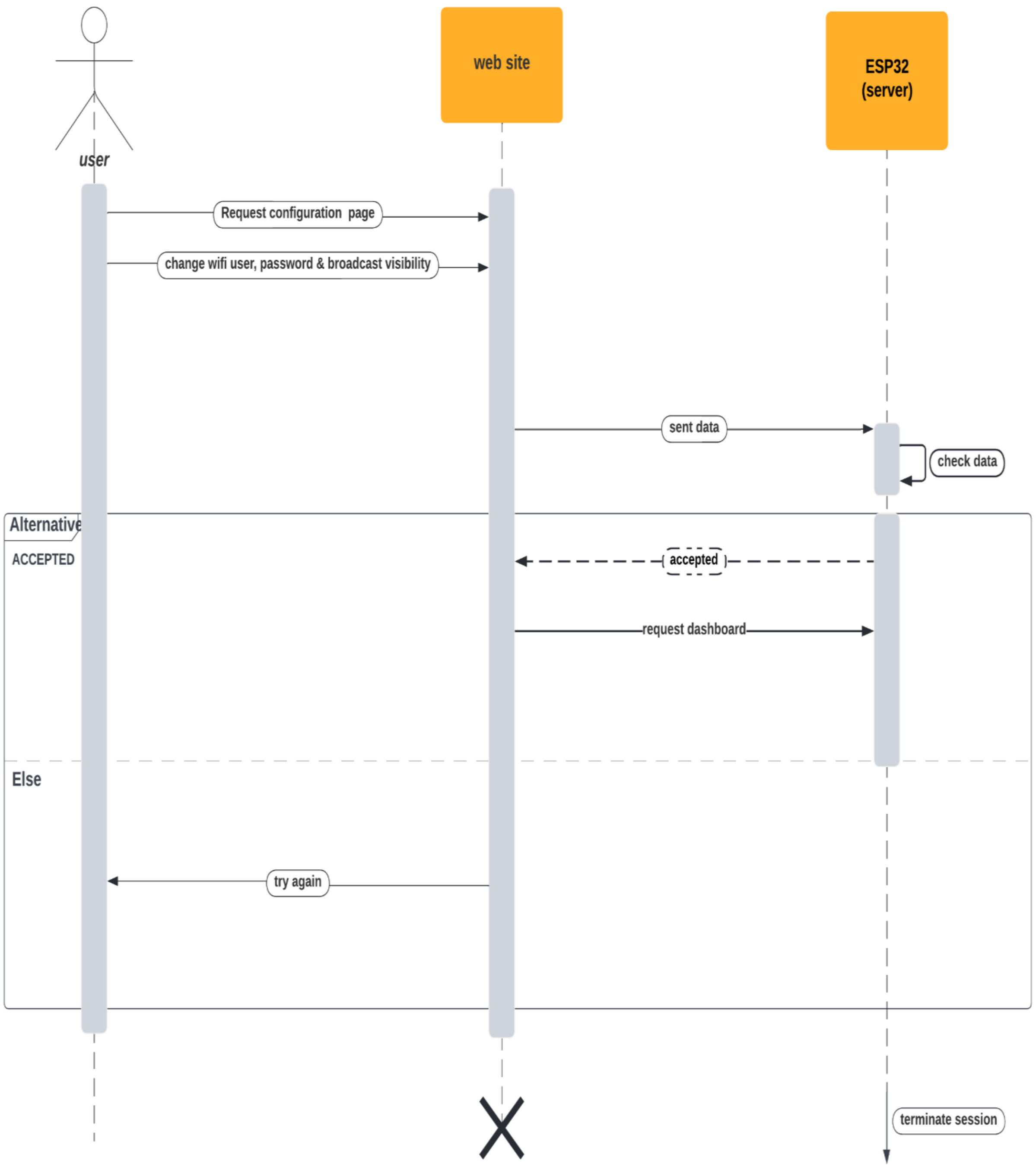

Figure 7:Sequence Diagram of wifi edit

2.6. Sequence Diagram user edit

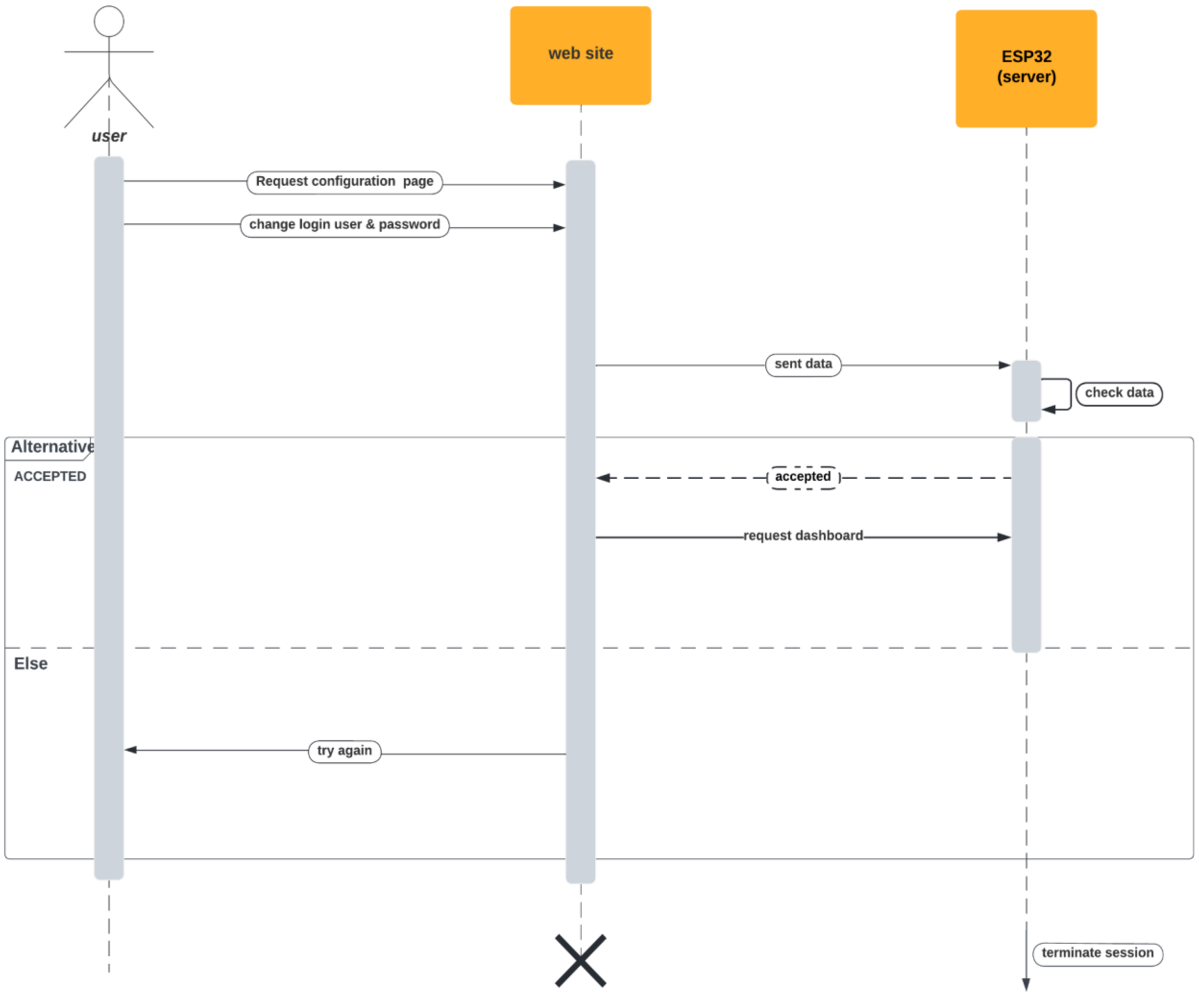

Figure 8:Sequence Diagram user edit

3. <u>VGA:</u>

3.1. Introduction:

The Video Graphics Array (VGA) monitor has been an integral component of personal computing since its introduction in 1987. It has undergone substantial enhancements and has become a standard feature on most personal computers. This research paper aims to delve into the history of VGA monitors, their technological progressions, and the profound influence they have had on personal computing.

3.2. Early VGA Monitors:

The inaugural VGA monitor was unveiled by IBM in 1987, presenting users with a resolution of 640x480 pixels along with the ability to display 16 colors. This marked a considerable advancement compared to preceding CGA and EGA monitors, which were limited to text and offered a narrower color spectrum. The VGA monitor garnered immediate acclaim due to its superior graphical capabilities and expanded color palette.

3.3. Technological Advancements:

Throughout the years, VGA monitors have experienced significant technological advancements aimed at augmenting their resolution, color depth, and overall performance. In 1990, Super VGA (SVGA) monitors were introduced, pushing the resolution boundaries to as high as 1024x768 pixels and allowing for the display of 256 colors. Subsequently, XGA monitors emerged in 1991, offering a resolution of 1024x768 pixels accompanied by the ability to showcase a staggering 16 million colors. The advent of flat-panel displays, such as Liquid Crystal Display (LCD) and Thin-Film Transistor (TFT) screens, has contributed immensely to the enhanced quality of VGA monitors.

3.4. Impact on Personal Computing:

The VGA monitor has played a pivotal role in the growth and development of personal computing. By facilitating the viewing and interaction with graphics-rich applications, it has become an indispensable tool for gamers, designers, and video editors. The continuous improvements in VGA technology have led to the emergence of larger displays boasting higher resolutions, empowering users to work more efficiently and enjoy visually captivating experiences of superior quality.

3.5. Conclusion:

Since its inception in 1987, the VGA monitor has undergone a remarkable evolution. Technological advancements have resulted in improved resolution, expanded color depth, and overall enhanced performance. The impact of VGA monitors on personal computing has been profound, providing users with the ability to engage with graphics-intensive applications. Looking ahead, with the continued development of innovative technologies like OLED and Quantum Dot displays, VGA monitors are poised to evolve further and offer even more exceptional visual experiences in the future.

4. FabGL:

4.1. What is FabGL?

- FabGL is an open-source library for programming embedded systems, especially those using microcontrollers with graphical displays. It stands for "Fabulous Graphics Library" and is designed to make it easier to create graphical user interfaces (GUIs) on small devices.
- FabGL supports a wide range of displays, including OLED and TFT LCD screens, and provides drivers for touchscreens and other input devices. It also includes a variety of widgets for creating buttons, sliders, and other common GUI elements, as well as support for fonts, images, and animations.
- In addition to its graphical capabilities, FabGL includes support for networking protocols such as MQTT and HTTP, making it easy to build connected devices that can communicate with other devices or services over the internet.
- Overall, FabGL is a powerful tool for building sophisticated embedded systems with graphical interfaces and network connectivity.
- One of the main advantages of FabGL is its support for a wide range of display types, including small OLED screens and larger TFT LCD displays. This flexibility makes it well-suited to a variety of applications, from simple sensor readouts to more complex graphical user interfaces.
- To simplify the process of creating GUIs, FabGL includes a number of pre-built widgets, such as buttons, sliders, and text boxes. These can be easily customized to match the look and feel of your device and can be arranged in a variety of layouts to create more complex interfaces.
- Another strength of FabGL is its support for touchscreens and other input devices. This allows users to interact with their device in a more intuitive way, using gestures like swiping and tapping to navigate menus and trigger actions.
- FabGL also includes support for fonts, images, and animations, making it easy to add visual elements to your interface. This can help to make your device more engaging and easier to use, especially for less technical users.
- Beyond its graphical capabilities, FabGL includes support for a variety of networking protocols, such as MQTT and HTTP. This allows your device to communicate with other devices or services over the internet, opening up a range of possibilities for building connected systems.
- Overall, FabGL is a powerful tool for building sophisticated embedded systems with graphical interfaces and network connectivity. Whether you're building a simple sensor readout or a more complex device with a full GUI, FabGL can help to simplify the development process and provide a solid foundation for your project.

4.2. More about image functionality in FabGL

One of the core features of FabGL's image functionality is its support for various image formats, including BMP, JPEG, and PNG. This allows you to load images from a variety of sources and use them in your application.

Once you've loaded an image, FabGL provides a range of functions for displaying it on the screen. These functions include support for scaling, rotation, and transparency, as well as the ability to display images as backgrounds or as part of a more complex GUI element.

In addition to its display capabilities, FabGL also includes functions for manipulating images at a lower level. This includes support for cropping, resizing, and converting between color spaces, allowing you to perform more complex image processing tasks if necessary.

Another useful feature of FabGL's image functionality is its support for animations. You can use a series of images to create a simple animation, such as a loading spinner or progress bar, or create more complex animations using a combination of images and code.

Finally, FabGL provides a number of optimization techniques to help reduce the memory and processing requirements of working with images on embedded systems. This includes techniques like color quantization and compression, which can reduce the amount of memory needed to store and display an image without sacrificing too much image quality.

Overall, FabGL's image functionality provides a powerful set of tools for working with images in embedded systems. Whether you're displaying static images or creating complex animations, FabGL can help you get the job done efficiently and effectively.

4.3. More about vga in FabGL

FabGL provides a number of features to support VGA displays and simplify working with them. Here are some of the key features of FabGL when working with VGA displays:

- VGA Controller: FabGL includes a VGA controller that supports standard VGA displays with resolutions up to 800x600 and a 256-color palette. This controller provides an easy way to control the display buffer and refresh the screen while optimizing memory usage and performance.

- Graphics Library: FabGL includes a powerful graphics library that supports a wide range of graphical primitives, including lines, circles, rectangles, and polygons. This library also includes support for text rendering, which can be a useful feature when building user interfaces or other text-based applications.

- Input Support: FabGL supports a range of input devices, including keyboards and mice, which can be useful when building interactive applications. The input support is easy to use and provides a simple way to capture user input and respond to it in real-time.

- Optimizations: FabGL includes a number of optimizations to help reduce the memory and processing requirements of working with VGA displays. This includes techniques like color quantization and compression, which can reduce the amount of memory needed to store and display images without sacrificing too much image quality.

- Documentation and Examples: FabGL includes extensive documentation and a number of examples to help you get started working with VGA displays. This can be especially useful if you're new to working with VGA displays or embedded systems in general.

Overall, FabGL provides a powerful set of tools for working with VGA displays in embedded systems. Whether you're building a simple user interface or a complex graphical application, FabGL can help you get the job done efficiently and effectively.

4.4. Disadvantages of FabGL:

Like any software library, FabGL is not without its potential problems and challenges. Here are a few things to keep in mind when working with FabGL:

- Hardware Compatibility: One of the main challenges of working with any embedded system is ensuring that your hardware is compatible with your software. FabGL supports a wide range of displays and input devices, but you'll need to check that your specific hardware is supported before you begin.
- Memory and Performance Constraints: Embedded systems typically have limited resources, including limited memory and processing power. This can be a challenge when working with graphics and other resource-intensive tasks. To avoid running into memory or performance issues, you'll need to be careful about how you use FabGL and optimize your code as much as possible.
- Steep Learning Curve: FabGL is a powerful library, but it can also be complex and difficult to learn, especially if you're new to embedded systems development. To get

the most out of FabGL, you'll need to invest time and effort into learning how it works and how to use it effectively.

- Debugging: Debugging can be challenging in embedded systems, especially when working with hardware-level code. FabGL includes a number of debugging tools to help you diagnose and fix issues, but it may still take some time and effort to track down and resolve problems.

Despite these potential challenges, many developers find FabGL to be a powerful and useful library for building embedded systems with graphical interfaces. By being aware of these potential issues and taking steps to address them, you can help ensure that your FabGL-based projects are successful and effective.

4.5. A potential problem when working with VGA displays

- One potential problem when working with VGA displays is compatibility. VGA is an older standard that has been largely replaced by newer display interfaces, such as HDMI and DisplayPort. This means that not all hardware may be compatible with VGA displays, and you may need to use adapters or converters to connect your VGA display to newer hardware.

- Another potential issue with VGA displays is their lower resolution and color depth compared to newer display technologies. VGA displays typically support a maximum resolution of 640x480 or 800x600, with 256 or 16-bit color depth. While this may be sufficient for some applications, it may not be suitable for more complex graphical interfaces or high-resolution video playback.

- Another challenge when working with VGA displays is ensuring that your code is optimized for performance and memory usage. VGA displays require a significant amount of memory to store the display buffer, and refreshing the display can be a resource-intensive process. To avoid performance issues, you'll need to carefully manage your code and optimize your algorithms as much as possible.

- Finally, VGA displays can also be more difficult to work with than newer display technologies, due to their age and the fact that they use analog signals rather than digital. This can make it more challenging to ensure consistent and reliable performance, especially in noisy or unstable environments.

- Despite these potential challenges, VGA displays remain a popular choice for many embedded systems applications, due to their low cost, wide availability, and ease of use. By being aware of these potential issues and taking steps to address them, you can help ensure that your VGA-based projects are successful and effective as shown in appendix A.

5. Bitluni:

5.1. Introduction:

Bitluni's ESP32Lib is an open-source library that provides a range of functionalities for the ESP32 microcontroller. It was developed by Peter "Bitluni" Friedrichs and is available on GitHub for free. This research paper will discuss the features and benefits of Bitluni's ESP32Lib and its potential applications.

5.2. Features:

Bitluni's ESP32Lib provides a wide range of features that make it a valuable tool for developers. One of the key features is its ability to control various sensors and devices using the ESP32 microcontroller. It supports a range of sensors, including accelerometers, gyroscopes, magnetometers, and temperature sensors, among others.

In addition to sensor support, Bitluni's ESP32Lib also includes support for various displays and screens, including OLED and LCD displays. It also provides support for audio output, allowing developers to play sounds and music through the ESP32 microcontroller.

Another significant feature of Bitluni's ESP32Lib is its support for internet connectivity. It includes libraries for connecting to Wi-Fi and Bluetooth networks, as well as support for accessing web services and sending HTTP requests.

5.3. Benefits:

The main benefit of Bitluni's ESP32Lib is its versatility. It provides a range of functionalities that can be used in various projects, from simple sensor monitoring to complex IoT applications. The library is also easy to use, with a simple and intuitive interface that makes it accessible to developers with varying levels of experience.

Another benefit of Bitluni's ESP32Lib is its open-source nature. The library is available for free on GitHub, and developers are encouraged to contribute to its development. This open-source approach means that the library is continuously evolving and improving, with new features and bug fixes being added regularly.

5.4. Applications:

Bitluni's ESP32Lib has a wide range of potential applications, from hobby projects to commercial products. It can be used in projects such as environmental monitoring systems, home automation, and wearable technology. Its support for internet connectivity also makes it suitable for IoT applications, such as smart home devices and industrial automation.

5.5. Circuit

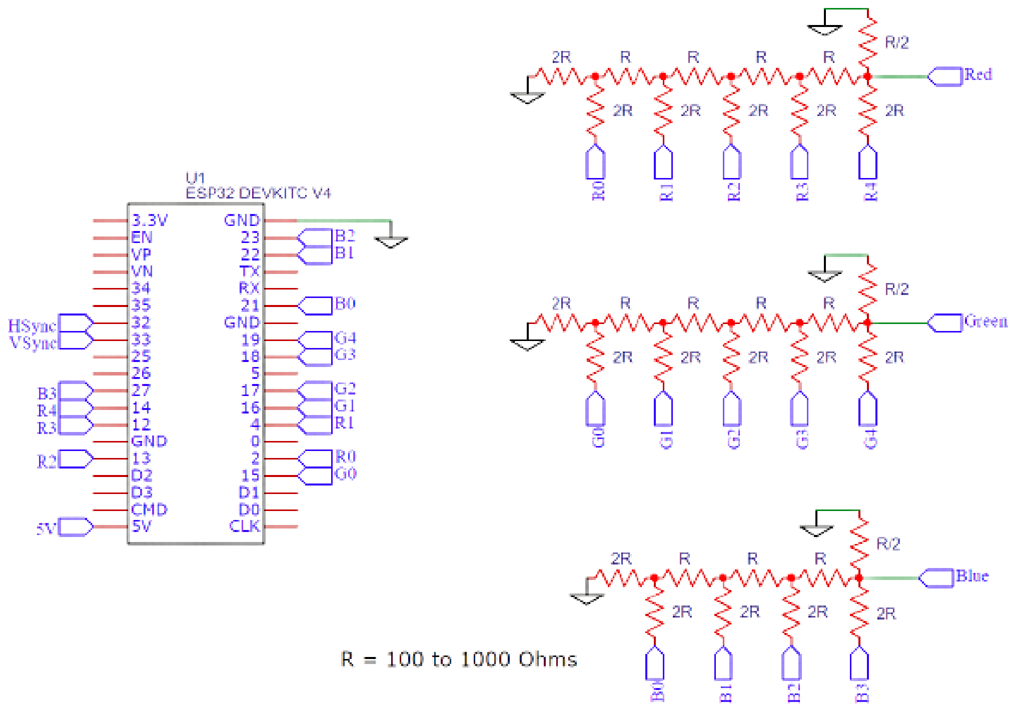

Figure 9: Bitluni's circuit

5.6. Problems of VGA:

While Bitluni's ESP32Lib offers a range of features for developers, there are also some challenges and limitations associated with the library. One such challenge is the limitations of VGA output.

VGA is an older video output standard that is still used in some applications, but it is not widely used in modern devices. Bitluni's ESP32Lib offers support for VGA output, but this support comes with some limitations. For example, VGA output is limited to a resolution of 640x480 pixels and a refresh rate of 60Hz. This limitation may be sufficient for some applications, but it may not be suitable for high-resolution or high-speed video output.

Another challenge associated with VGA output in Bitluni's ESP32Lib is the requirement for additional hardware. VGA output requires a digital-to-analog converter (DAC) to convert the digital video signal from the ESP32 into an analog signal that can be output to

a VGA monitor. This means that developers using Bitluni's ESP32Lib for VGA output will need to purchase additional hardware, which may add to the overall cost of the project.
Additionally, VGA output in Bitluni's ESP32Lib is not always stable and may suffer from image distortion or flickering. This instability can be caused by a variety of factors, including issues with the DAC or interference from other electronic components.
Despite these challenges, Bitluni's ESP32Lib remains a valuable tool for developers working with the ESP32 microcontroller. While VGA output may not be suitable for all applications, the library provides support for a wide range of other features, including sensors, displays, and internet connectivity. Developers can use these features to create a wide range of projects, from simple sensor monitoring to complex IoT applications.

5.7. Usage:

There are 4 different VGA Drivers VGA3Bit, VGA3BitI, VGA14Bit, and VGA14BitI. VGA3Bit and VGA14Bit are high-performance drivers that don't need any CPU time to serve the VGA. However, the VGA3Bit driver takes twice the memory compared to VGA3BitI. The high-performance drivers work best with the WiFi features. The other driver might cause errors. WiFi should connect first before VGA3BitI and VGA14BitI are initialized. The I drivers are using an interrupt to feed the pixels to the I²S. This feature can be used for real-time outputs (Check the VGANoFramebuffer example). An instance of the driver has to be created. The optional parameter (only for 14-bit versions) is the I²S bus to be used. If no parameter is given 1 is used by default to keep I²S0 free for audio output.

```
VGA14Bit vga(1);
```

Creating the instance does nothing until the init method is called once. The driver will initialize with one frame buffer by default (that's the memory where the pixels are stored). Showing animation using only one frame buffer will cause flickering since the frame will be displayed while it is redrawn. To have smooth animations double buffering is recommended. A second - the back buffer - is used to paint the graphics in the background. When the rendering is finished the front and back buffers are flipped. The disadvantage of the second buffer is the doubled memory requirements. 320x240 will no longer work (320x200 will). Double buffering is enabled using
vga.setFrameBufferCount(2);
before the init method is called (not calling it will result in a single buffer init)
vga.init(vga.MODE320x200, redPins, greenPins, bluePins, hsyncPin, vsyncPin);

5.8. Conclusion

Bitluni's ESP32Lib is a valuable tool for developers working with the ESP32 microcontroller. Its wide range of features, ease of use, and open-source nature make it a versatile and accessible library for a range of projects. Its potential applications are diverse, and it is likely to play an increasingly important role in the development of IoT applications and other embedded systems as shown in appendix B.

6. R2R Ladder:

6.1. What is an R-2R Ladder?

An R2R ladder, also known as a resistor ladder, is an electronic circuit that is commonly used in digital-to-analog converters (DACs). It is composed of a series of resistors that are connected in a ladder-like configuration, with the output of each resistor connected to the input of the next resistor.
The basic R2R ladder circuit consists of two types of resistors - R and 2R - connected in an alternating pattern. The R resistors are connected to the ground, while the 2R resistors are connected to the voltage source. The input voltage to the circuit is applied across the R resistors, and the output voltage is taken from the junction of the 2R resistors.
The R2R ladder works on the principle of binary-weighted voltages, where each resistor in the ladder corresponds to a particular bit in the binary code of the input voltage. When the input voltage is applied, each resistor generates a voltage that is proportional to its binary weight, and the sum of these voltages produces the output voltage.

R2R ladders are widely used in digital-to-analog converters (DACs) because of their simplicity, accuracy, and low cost. They can be easily scaled to accommodate different input voltages and resolutions, and they offer high linearity and low noise. However, they are not suitable for high-speed applications, as they have limited bandwidth and high settling time.

6.2. How R-2R Network Connected?

In an R-2R network, the resistors are connected in a ladder-like configuration, with the output of each resistor connected to the input of the next resistor. The basic configuration of an R-2R ladder network consists of two types of resistors, R and 2R, which are connected in an alternating pattern.
At the input of the R-2R ladder network, the input voltage is applied across the series of R resistors. The first resistor in the series is connected to the input voltage, and the remaining resistors are connected in series to the ground. Each resistor generates a voltage that is proportional to its resistance, which is determined by the ratio of R to 2R.

The output of each resistor is connected to a node, which is either connected directly to the output of the network or to a summing node that sums the output voltages from each resistor in the network. The output voltage of the network is taken from the final node in the ladder, which is either connected to the summing node or directly to the output of the network.

The R-2R ladder network works on the principle of binary-weighted voltages, where each resistor in the network corresponds to a particular bit in the binary code of the input voltage. When the input voltage is applied, each resistor generates a voltage that is proportional to its binary weight, and the sum of these voltages produces the output voltage.

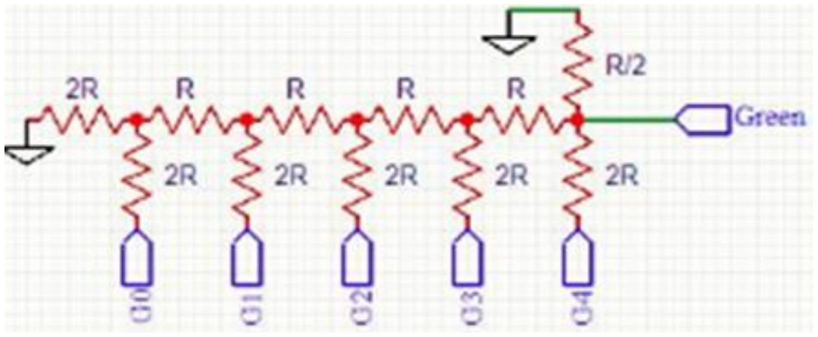

Figure 10: R-2R Ladder Circuit (5-bit)

6.3. R2R ladder equation:

The equation for an R-2R ladder network can be derived using the principle of superposition. The output voltage of the network is the sum of the voltages generated by each resistor in the ladder, where each voltage is proportional to its resistance and the input voltage. The equation for the output voltage of an N-bit R-2R ladder network is given by:

V_out = V_in * [2^N * (R/2R) - (2^N - 1)]

where V_in is the input voltage, R is the resistance of the R resistor, and 2R is the resistance of the 2R resistor. N is the number of bits in the network, which determines the number of resistors in the ladder.

For example, consider a 4-bit R-2R ladder network with R = 10kΩ and 2R = 20kΩ. The output voltage of the network can be calculated using the equation above:

V_out = V_in * [2^4 * (10kΩ/20kΩ) - (2^4 - 1)]
V_out = V_in * [8 - 15]
V_out = -7V_in

Therefore, the output voltage of the 4-bit R-2R ladder network is -7 times the input voltage. This means that if the input voltage is 1V, the output voltage will be -7V.

6.4. Pros and cons of R2R ladder:

- Pros of R2R ladder:

    - High linearity: The R2R ladder offers high linearity due to the binary-weighted voltage distribution across its resistors.
    - Low noise: The R2R ladder provides low noise output signals due to its simple structure and inherent averaging effect.
    - Easily scalable: The R2R ladder can be easily scaled to accommodate different input voltages and resolutions, making it a versatile circuit for various applications.
    - Low cost: The R2R ladder is a simple and low-cost circuit, making it a popular choice for low-cost DACs.

- Cons of R2R ladder:

    - Limited bandwidth: The R2R ladder has a limited bandwidth, which makes it unsuitable for high-speed applications.
    - High settling time: The R2R ladder has a high settling time, which makes it unsuitable for applications where fast settling is required.
    - Large footprint: The R2R ladder requires a large number of resistors, which can take up significant space on a circuit board and increase the overall cost of the circuit.
    - Non-monotonicity: Non-monotonicity can occur in the output of the R2R ladder if there are errors in the matching of the resistors, which can result in a loss of accuracy in the output signal.

Overall, the R2R ladder is a simple and versatile circuit with high linearity and low noise, making it a popular choice for low-cost DACs. However, it has limitations in terms of bandwidth and settling time, and the large footprint of the circuit can be a disadvantage in some applications.

6.5. Conclusion points about the R2R ladder:

- R2R ladder is an important electronic circuit used in digital-to-analog converters (DACs).
- The circuit consists of a series of resistors that are connected in a ladder-like configuration, with the output of each resistor connected to the input of the next resistor.
- R2R ladder works on the principle of binary-weighted voltages and each resistor in the ladder corresponds to a particular bit in the binary code of the input voltage.

- The output voltage of an R2R ladder network is the sum of the voltages generated by each resistor in the ladder.
- The R2R ladder network offers high linearity and low noise, and it can be easily scaled to accommodate different input voltages and resolutions.
- However, the R2R ladder has some limitations like limited bandwidth and high settling time, which makes it unsuitable for high-speed applications.
- The R2R ladder is a fundamental building block for DACs and it has contributed significantly to the advancement of digital electronics.

7. TJpg_Decoder

7.1. Introduction:

TJpg_Decoder is a software library that allows for the decoding of images that have been compressed using the JPEG format. JPEG is a widely used format for image compression due to its ability to compress images with minimal loss of image quality. The TJpg_Decoder library is designed to decode JPEG images quickly and efficiently, making it a valuable tool for applications that need to process large volumes of images.

7.2. Background:

The JPEG format was developed in the late 1980s and has since become one of the most widely used image compression formats. JPEG compression works by breaking down an image into small blocks of pixels, and then compressing these blocks using a mathematical algorithm. The result is a compressed image file that takes up less space than the original image file, without significantly affecting the quality of the image.
TJpg_Decoder is a software library that was developed by Thomas Faber. It is written in the C programming language and is designed to be highly optimized for decoding JPEG images. The library is designed to be used on embedded systems with limited resources, such as microcontrollers and mobile devices.

The JPEG format was developed in the late 1980s and has since become one of the most widely used image compression formats. JPEG compression works by breaking down an image into small blocks of pixels, and then compressing these blocks using a mathematical algorithm. The result is a compressed image file that takes up less space than the original image file, without significantly affecting the quality of the image.

TJpg_Decoder is a software library that was developed by Thomas Faber. It is written in the C programming language and is designed to be highly optimized for decoding JPEG images. The library is designed to be used on embedded systems with limited resources, such as microcontrollers and mobile devices.

7.3. Features:

The TJpg_Decoder library has a number of features that make it a valuable tool for developers who need to decode JPEG images. These features include:

- High performance: The TJpg_Decoder library is designed to be highly optimized for decoding JPEG images quickly and efficiently. This makes it a valuable tool for applications that need to process large volumes of images.
- Small memory footprint: The library is designed to use as little memory as possible, making it suitable for use on embedded systems with limited resources.
- Supports baseline JPEG: The library supports decoding of baseline JPEG images, which are the most common type of JPEG images.
- Portable: The library is written in C and is designed to be portable across different platforms and operating systems.
- Easy to use: The library is designed to be easy to use, with a simple API that can be integrated into applications with minimal effort.

7.4. Applications:

The TJpg_Decoder library has a wide range of applications, including:

- Digital cameras: Digital cameras often use JPEG compression to store images. The TJpg_Decoder library can be used to decode these images quickly and efficiently.
- Mobile devices: Mobile devices often have limited resources and need to process images quickly. The TJpg_Decoder library is designed to be highly optimized for use on these devices.
- Embedded systems: Embedded systems often have limited resources and need to process images quickly. The TJpg_Decoder library is designed to use as little memory as possible, making it suitable for use on these systems.
- Image processing: The TJpg_Decoder library can be used as a tool for image processing, allowing developers to manipulate JPEG images with minimal loss of quality.

7.5. Conclusion:

In conclusion, the TJpg_Decoder library is a valuable tool for developers who need to decode JPEG images quickly and efficiently. Its high performance, small memory footprint, and ease of use make it suitable for a wide range of applications, including digital cameras, mobile devices, embedded systems, and image processing. As the use of JPEG compression continues to grow, the TJpg_Decoder library will continue to be an important tool for developers working with images as shown in appendix A.

8. ESPAsyncWebServer

8.1. Abstract:

The ESPAsyncWebServer is a popular open-source library for creating asynchronous web servers on the ESP8266 and ESP32 microcontroller boards. This research paper provides an in-depth analysis of the library, its features, and its applications. We explore the design and architecture of the library, including its use of the AsyncTCP library and its support for HTTP and HTTPS protocols. We also examine the library's functionality for handling HTTP requests and responses, serving static files, handling WebSocket connections, and more. Finally, we discuss the library's use cases and provide examples of projects that have used the library for home automation, IoT devices, and web-based user interfaces for embedded systems.

8.2. Introduction:

The ESPAsyncWebServer library was developed to provide a lightweight, efficient, and easy-to-use web server for the ESP8266 and ESP32 microcontroller boards. The library is based on the AsyncTCP library, which provides asynchronous TCP/IP communication for the ESP8266 and ESP32. By using asynchronous communication, the library is able to handle multiple connections simultaneously, making it well-suited for applications that require responsive user interfaces.

8.3. Design and Architecture:

The ESPAsyncWebServer library is designed to be modular and extensible. The library provides a set of classes and functions for handling HTTP requests and responses, serving static files, handling WebSocket connections, and more. The library is built on top of the AsyncTCP library, which provides a low-level TCP/IP communication layer. The AsyncTCP library is responsible for handling TCP/IP connections and providing a callback mechanism for asynchronous communication. The ESPAsyncWebServer library uses these callbacks to implement its own functionality for handling HTTP requests and responses.

8.4. Functionality:

The ESPAsyncWebServer library provides a wide range of functionality for handling HTTP requests and responses. The library supports both HTTP and HTTPS protocols and provides a flexible API for handling requests and responses asynchronously. The library also includes functionality for serving static files, handling WebSocket connections, and more. The library's WebSocket support allows for real-time communication between the server and the client, making it well-suited for applications such as home automation and IoT devices.

8.5. Applications:

The ESPAsyncWebServer library has been used in a wide range of applications, including home automation, IoT devices, and web-based user interfaces for embedded systems. The library's asynchronous communication and WebSocket support make it well-suited for applications that require real-time communication between the server and client. The library's lightweight design and flexible API also make it well-suited for applications with limited resources, such as those on the ESP8266 and ESP32 microcontroller boards.

8.6. Conclusion:

The ESPAsyncWebServer library is a powerful and flexible tool for creating asynchronous web servers on the ESP8266 and ESP32 microcontroller boards. The library's modular design, support for both HTTP and HTTPS protocols, and WebSocket support make it well-suited for a wide range of applications. The library has a large user base and an active community of developers, making it a popular choice for developers working on home automation, IoT devices, and web-based user interfaces for embedded systems.

9. ESPmDNS: A Lightweight Solution for Multicast DNS on ESP32-based Devices

9.1. Abstract:

ESPmDNS is a lightweight and efficient implementation of Multicast DNS (mDNS) for ESP32-based devices. mDNS is a protocol that allows devices on a local network to discover and communicate with each other using domain names without the need for a centralized DNS server. ESPmDNS provides a simple and easy-to-use solution for ESP32 microcontrollers to participate in mDNS-based networking, enabling seamless discovery and communication between devices in a local network. This paper provides an overview of ESPmDNS, its features, implementation details, and potential use cases.

9.2. Introduction

The rapid growth of Internet of Things (IoT) devices has led to an increased need for efficient and decentralized network discovery and communication mechanisms. Multicast DNS (mDNS) has emerged as a popular solution for device discovery and service advertisement in local networks. ESPmDNS is an implementation of mDNS specifically designed for ESP32-based devices, offering a lightweight and resource-efficient solution.

9.3. Multicast DNS (mDNS) Overview

This section provides an overview of the mDNS protocol, explaining its principles, advantages, and use cases. It discusses the challenges of device discovery in local networks and how mDNS addresses these challenges. Additionally, it highlights the key features and benefits of mDNS in comparison to traditional DNS solutions.

9.4. ESPmDNS Architecture

This section delves into the architecture and design of ESPmDNS. It discusses the underlying principles and mechanisms used by ESPmDNS to enable mDNS functionality on ESP32 microcontrollers. The section covers topics such as message format, resource records, service registration, and name resolution.

9.5. Features and Capabilities

ESPmDNS offers several features and capabilities that enhance the functionality of ESP32-based devices within a local network. This section explores the key features of ESPmDNS, including service discovery, automatic hostname assignment, and flexible configuration options. It also highlights the extensibility of ESPmDNS, allowing developers to customize and tailor the implementation to their specific requirements.

9.6. Implementation Details

This section provides insights into the technical implementation of ESPmDNS on ESP32 microcontrollers. It discusses the libraries, APIs, and tools utilized in developing ESPmDNS. Furthermore, it explains the memory footprint and performance considerations, ensuring efficient usage of system resources on resource-constrained devices.

9.7. Integration and Use Cases

ESPmDNS can be seamlessly integrated into a wide range of IoT applications and scenarios. This section explores various use cases where ESPmDNS can be leveraged, such as smart homes, industrial automation, and local network services. It provides practical examples and code snippets to illustrate the integration process.

9.8. Performance Evaluation

To assess the performance of ESPmDNS, this section presents a performance evaluation methodology and benchmarks the implementation against key performance indicators. It analyzes factors such as response time, network overhead, and scalability, providing insights into the efficiency and reliability of ESPmDNS.

9.9. Comparison with Other Solutions

In this section, ESPmDNS is compared with other mDNS implementations and similar solutions available for ESP32-based devices. It examines factors such as ease of use, resource utilization, and compatibility with existing frameworks, enabling readers to make informed decisions based on their specific requirements.

9.10. Conclusion

ESPmDNS offers a lightweight and efficient solution for implementing Multicast DNS on ESP32-based devices. Its simplicity, extensibility, and resource efficiency make it a valuable tool for IoT developers seeking to enhance device discovery and communication capabilities in local networks. This paper has provided an overview of ESPmDNS, its features, implementation details, and potential use cases, demonstrating its utility and potential impact in the field of IoT networking.

10. PDF.js library

PDF.js is an open-source project that allows users to display PDF documents directly in their web browsers using JavaScript. It is a library built on top of HTML5, CSS, and JavaScript and is supported by modern web browsers such as Chrome, Firefox, and Edge. PDF.js was developed by Mozilla, and its source code is available on GitHub under the Apache License.

PDF.js provides several features that allow users to interact with PDF documents. It supports the rendering of PDF files in a browser with high fidelity and accuracy, including text search, zooming, and rotation. It also allows users to fill out PDF forms and save the data to a server.

PDF.js uses a web worker to perform heavy computation in the background, allowing for fast rendering of large PDF documents. It supports multiple rendering modes, such as canvas, SVG, and CSS, which allows users to customize the rendering based on their needs.

One of the advantages of using PDF.js is that it is highly customizable. Developers can customize the user interface and create their own PDF viewer. It is also possible to extract text and metadata from a PDF document using PDF.js, which can be useful for indexing and searching PDF documents.
PDF.js supports encrypted PDF documents, but it requires a separate library called pdf.js-dist/build/pdf.worker.js, which needs to be loaded separately. This library provides the functionality to decrypt encrypted PDF documents.

PDF.js is designed to work seamlessly with other web technologies such as CSS and JavaScript frameworks like React, Angular, and Vue. It is also possible to use PDF.js with server-side technologies like Node.js to generate PDF documents on the fly.

PDF.js provides several APIs that can be used to manipulate PDF documents programmatically. These APIs can be used to add annotations, edit existing content, and even create new PDF documents from scratch.

PDF.js is an ideal solution for applications that require a lightweight and responsive PDF viewer. It is easy to integrate with other web technologies and can be used in a variety of use cases, such as e-commerce, document management, and online education.

One of the key benefits of PDF.js is its ability to work without the need for any additional software or plugins, making it easy for users to access PDF documents directly in their browsers without any additional setup.

In conclusion, PDF.js is a powerful and versatile library that enables users to view, edit, and manipulate PDF documents directly in their web browsers. Its ability to work seamlessly with other web technologies, its high level of customization, and its support for a wide range of features make it an excellent choice for developers looking to incorporate PDF functionality into their web applications.

10.1. Using PDF.js library:

This code shown in appendix C is designed to convert a PDF document into a series of images. The images are displayed in a web page and can be downloaded as individual files. This code is written in JavaScript using the pdf.js library.
Variables:

- imageNo: This variable is used to keep track of the number of images generated. It is initially set to 1.
- firstTime: This variable is not currently used in the code.
- options: This variable is an object that contains various options used in the conversion process. These include the input and preview selectors, the path to the pdf.worker.js file, the scale, width, and height of the output images.
- input: This variable is set to the input element with the ID "pdf_file".
- encoded_data: This variable is set to the result of the get_file_content() function. It is a binary representation of the PDF file.

Functions:

- main(): This function is called when the DOM is loaded. It sets up the options object, gets the file content using the get_file_content() function, and calls the process_document() function.
- get_file_content(): This function takes an input element as a parameter and returns a Promise that resolves with the binary representation of the PDF file. It reads the file using a FileReader object and extracts the encoding.
- is_pdf(): This function takes a filename as a parameter and returns true if the file extension is "pdf" or "PDF". It is used to check that the selected file is a PDF.
- extract_encoding(): This function takes a full encoding string as a parameter and returns the binary representation of the PDF file. It is used to extract the encoding from the result of the FileReader object.
- process_document(): This function takes the encoded data and options as parameters and converts the PDF document into a series of images. It stores the images in localStorage and displays them in the web page. It also sets the number of pages in the document and disables the input element once the conversion is complete.
- handle_load(): This function takes the pdf object, options, and page number as parameters and renders a single page of the PDF onto a canvas. It returns a Promise that resolves with the page number and image encoding.
- display(): This function takes an image encoding as a parameter and displays the image in the web page. It also stores the image in localStorage and increments the imageNo variable.

The flow of the code:

- When the DOM is loaded, the main() function is called.
- The options object is set up and the file content is read using the get_file_content() function.
- The process_document() function is called with the encoded data and options as parameters.
- The process_document() function clears any old images in localStorage, sets the number of pages in the document, and calls handle_load() for each page in the document.
- The handle_load() function renders a single page of the PDF onto a canvas and returns the page number and image encoding as a Promise.
- The display() function is called with the image encoding as a parameter and the image is displayed on the web page.
- Once all pages have been processed, the input element is disabled.

11. Progressive Web Applications: Bridging the Gap Between Web and Native Experiences:

11.1. Introduction:

Progressive Web Applications (PWAs) have emerged as a groundbreaking approach to building web applications that combine the best features of both web and native applications. By leveraging modern web technologies, PWAs offer enhanced user experiences, offline functionality, and increased engagement. This paper provides an overview of PWAs, exploring their key characteristics, benefits, and challenges. It also examines real-world examples and discusses the future potential of PWAs in shaping the digital landscape.

The rapid advancement of web technologies and the growing demand for mobile applications have given rise to Progressive Web Applications (PWAs). PWAs provide a set of techniques and principles to build web applications that closely resemble native applications, offering users a seamless, app-like experience. This paper explores the concept of PWAs, their core features, and how they bridge the gap between traditional web and native applications.

11.2. Key Characteristics of PWAs

11.2.1. Responsive Design
11.2.2. App-like Experience
11.2.3. Offline Functionality
11.2.4. Push Notifications
11.2.5. Seamless Updates
11.2.6. Discoverability and Linkability

11.3. Benefits of PWAs

11.3.1. Enhanced User Experience
11.3.2. Improved Performance
11.3.3. Increased Engagement and Conversion Rates
11.3.4. Cross-platform Compatibility
11.3.5. Lower Development and Maintenance Costs
11.3.6. Discoverability and SEO

11.4. Challenges and Considerations

11.4.1. Limited Browser Support
11.4.2. Performance Optimization

11.4.3. Security and Privacy Concerns
11.4.4. App Store Limitations
11.4.5. Device Limitations

11.5. Real-World Examples of PWAs

11.5.1. Twitter Lite
11.5.2. Flipkart Lite
11.5.3. Starbucks
11.5.4. Pinterest
11.5.5. Trivago

11.6. Future Potential of PWAs

11.6.1. Integration with Native Features
11.6.2. Increased Adoption by Businesses
11.6.3. Advancements in Web Technologies
11.6.4. Standardization and Browser Support
11.6.5. Impact on Mobile App Development

11.7. Conclusion

Progressive Web Applications have revolutionized the way we think about web development, offering a bridge between the web and native app experiences. With their ability to provide offline functionality, push notifications, and seamless updates, PWAs have gained momentum across various industries. While challenges remain, the future of PWAs appears promising, with increased adoption and advancements in web technologies set to reshape the digital landscape. As businesses continue to embrace PWAs, users can expect more engaging, app-like experiences on the web.

12. Schematics:

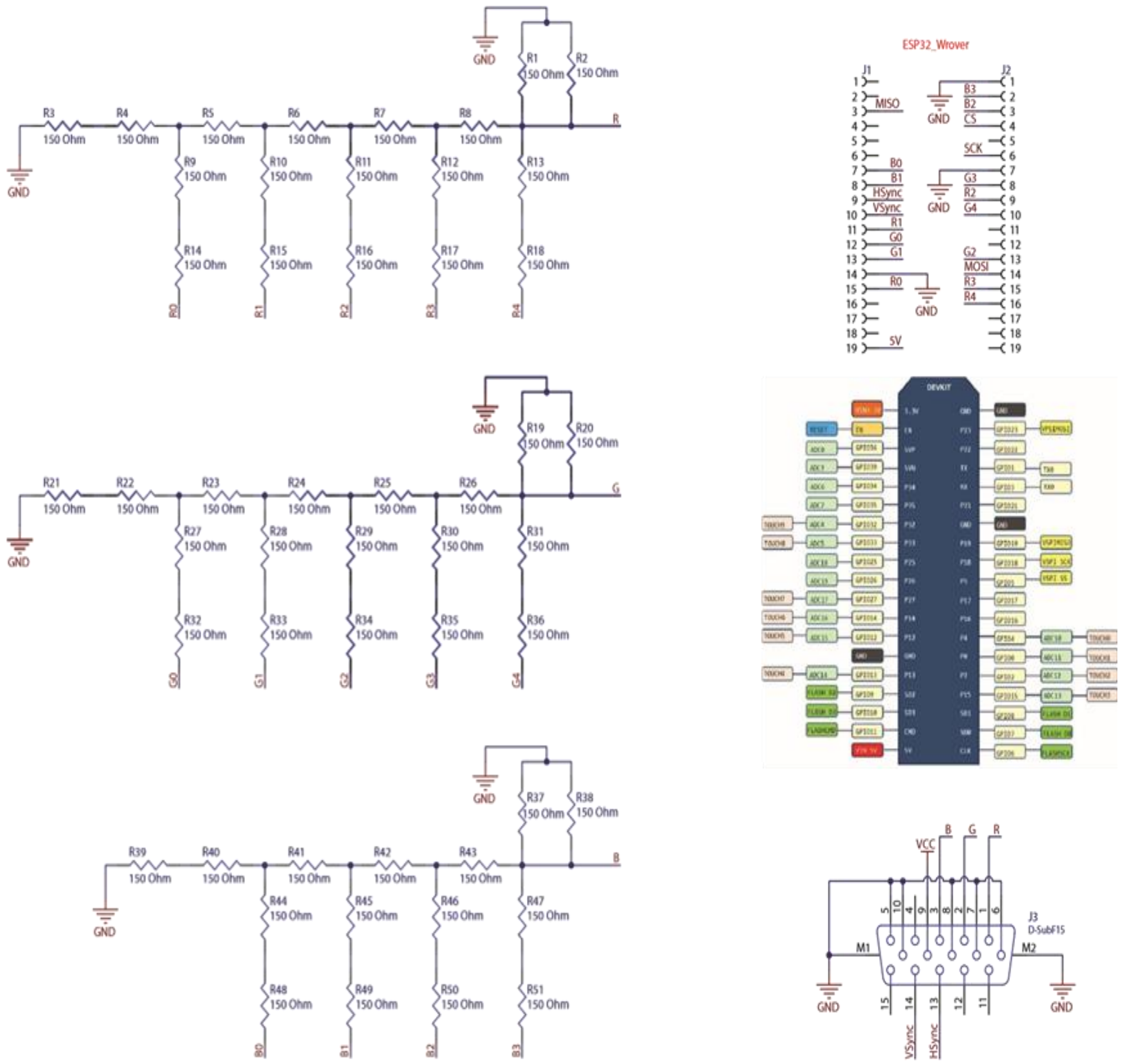

Figure 11: circuit of the project

Chapter Three: conclusion

1. Accomplishments:

In conclusion, the project using ESP32 with Bitluni and Web Async to convert an old VGA monitor into a billboard has been successful. By leveraging the capabilities of the ESP32 microcontroller, Bitluni's libraries for VGA output, and the power of asynchronous web libraries , the project has effectively repurposed an outdated piece of technology into a useful display for showing information and advertisements.

Let's talk a bit more about the benefits of this project. As I mentioned before, it is a cost-effective solution for those looking to display information in a public space. Digital billboards can be very expensive to purchase and install, but this project offers a much more affordable alternative. By repurposing an old VGA monitor, businesses and individuals can save money while still having a functional display for showing information and advertisements.

Another benefit of this project is its use of modern technology. The ESP32 microcontroller is a powerful device that can be programmed to do a variety of tasks. By using it in this project, the creators were able to leverage its capabilities to output VGA signals to the old monitor, effectively turning it into a functional display. Additionally, by using Bitluni's libraries for VGA output, they were able to simplify the programming process, making it easier for others to replicate the project.

The use of asynchronous web programming is also a significant benefit of this project. By allowing the monitor to pull information from a web server in real-time, the content displayed on the billboard can be updated dynamically. This means that businesses and individuals can ensure that the information being displayed is always up-to-date and relevant. Furthermore, by using Web Async to communicate with the server, the project's creators were able to make the process more efficient, ensuring that the monitor could update its content quickly and reliably.

Overall, this project is a great example of how innovative solutions can be created using modern technology and creative thinking. By repurposing old technology, businesses and individuals can save money while still having access to functional displays for showing information and advertisements. With the power of the ESP32 microcontroller, Bitluni's libraries, and asynchronous web programming, the possibilities are endless for what can be accomplished.

One of the benefits of this project is its cost-effectiveness. Instead of buying a new digital billboard, which can be quite expensive, this project offers a more affordable solution for businesses and individuals looking to display information in a public space.

Furthermore, the use of asynchronous web libraries allows for the dynamic updating of content on the monitor. This means that information can be updated in real-time, ensuring that the billboard is always displaying the most current information.

Overall, this project is a great example of how old technology can be repurposed and given new life. With the power of modern microcontrollers and programming techniques, it is possible to create innovative solutions that can save money, reduce waste, and improve the functionality of outdated devices.

2. <u>Uncertainties:</u>

We regret to inform you that, despite our efforts, we have been unable to increase the resolution beyond 320x200 pixels. Furthermore, due to the limited resources of the variation of esp32 we are currently utilizing, we are only able to accommodate a single web request at a time. As a result, we have been forced to restrict access to the server to a single client at a time.

3. <u>Future work & improvement:</u>

Our plan is to acquire a more powerful variant of the ESP32 microcontroller, such as the ESP32S3, which can handle higher resolution images and support a larger number of web users. By doing so, we aim to enhance the overall performance of our system and ensure that it can meet the growing demands of our clients.

Additionally, we intend to develop a user-friendly web application that will enable our clients to easily edit and modify their photos directly from their mobile devices. This application will simplify the process of making changes to images, making it more convenient and efficient for our clients.

Chapter Four: Appendices

appendix A:

```
#include "fabgl.h"
#include <TJpg_Decoder.h>
#include <FS.h>
#include "SPIFFS.h" // ESP32 only
fabgl::VGA16Controller VGAController;
fabgl::Canvas cv(&VGAController);
// This next function will be called during decoding of the jpeg file to render each block to
the Canvas.
bool vga_output(int16_t x, int16_t y, uint16_t w, uint16_t h, uint16_t* bitmap) {
  // Stop further decoding as image is running off bottom of screen
  if ( y >= cv.getHeight() ) return 0;
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      writePixel(x + i, y, bitmap[j * w + i]);
    }
  }
  // Return 1 to decode next block
  return 1;
}
void setup() {
  Serial.begin(115200);
  Serial.println("\n\n Testing TJpg_Decoder library With FabGl");
  // Initialise SPIFFS
  if (!SPIFFS.begin()) {
    Serial.println("SPIFFS initialisation failed!");
    while (1) yield(); // Stay here twiddling thumbs waiting
  }
  Serial.println("\r\nInitialisation done.");
  VGAController.begin(GPIO_NUM_12, GPIO_NUM_27, GPIO_NUM_14,
GPIO_NUM_26, GPIO_NUM_25);
  VGAController.setResolution(VGA_640x480_60Hz);
  cv.setBrushColor(RGB888(0, 0, 0));
  cv.clear();
  TJpgDec.setJpgScale(1);
  // The decoder must be given the exact name of the rendering function above
  TJpgDec.setCallback(vga_output);
  // Time recorded for test purposes
  uint32_t t = millis();
  // Get the width and height in pixels of the jpeg if you wish
  uint16_t w = 0, h = 0;
  TJpgDec.getFsJpgSize(&w, &h, "/image.jpg"); // Note name preceded with "/"
  Serial.print("Width = "); Serial.print(w); Serial.print(", height = "); Serial.println(h);
  // Draw the image, top left at 0,0
  TJpgDec.drawFsJpg(0, 0, "/image.jpg");
  // How much time did rendering take
  t = millis() - t;
  Serial.print(t); Serial.println(" ms");
}
void loop() {
}
void writePixel(int16_t x, int16_t y, uint16_t rgb565) {
  uint8_t r = ((rgb565 >> 11) & 0x1F) * 255 / 31;   // red   0 .. 255
  uint8_t g = ((rgb565 >> 5) & 0x3F) * 255 / 63.0;  // green 0 .. 255
  uint8_t b = (rgb565 & 0x1F) * 255 / 31.0;         // blue  0 .. 255
  cv.setPixel(x, y, RGB888(r, g, b));
}
```

appendix B:

```
#include <ESP32Lib.h>
#include <Ressources/Font6x8.h>
//include the sprites converted the SpriteConverter. Check the documentation for further
infos.
#include "explosion.h"
//pin configuration
const int redPins[] = {2, 4, 12, 13, 14};
const int greenPins[] = {15, 16, 17, 18, 19};
const int bluePins[] = {21, 22, 23, 27};
const int hsyncPin = 32;
const int vsyncPin = 33;
//VGA Device
VGA14Bit vga;
//initial setup
void setup()
{
  //need double buffering
  vga.setFrameBufferCount(2);
  //initializing i2s vga
  vga.init(vga.MODE200x150, redPins, greenPins, bluePins, hsyncPin, vsyncPin);
  //setting the font
  vga.setFont(Font6x8);
}
//just draw each frame
void loop()
{
  //draw a background
  for (int y = 0; y < vga.yres / 10; y++)
    for (int x = 0; x < vga.xres / 10; x++)
      vga.fillRect(x * 10, y * 10, 10, 10, (x + y) & 1 ? vga.RGB(0, 128, 0) : vga.RGB(0, 0,
128));
  //print some labels
  vga.setCursor(36, 41);
  vga.print("draw   drawMix  drawAdd");
    //check the Utilities folder for the converter
  //"draw" draws the sprite opaque ignoring any existing alpha channel
  explosion.draw(vga, (millis() / 50) % 20, vga.xres / 4, vga.yres / 2);
  //"drawMix" uses the alpha channel
  explosion.drawMix(vga, (millis() / 50) % 20, vga.xres / 2, vga.yres / 2);
  //"drawAdd" adds the color components of the back ground and the sprite
  explosion.drawAdd(vga, (millis() / 50) % 20, vga.xres * 3 / 4, vga.yres / 2);
  //swap the frame buffers and show the rendering
  vga.show();
}
```

appendix C:

```
#include <ESP32Lib.h>
#include <Ressources/Font6x8.h>
//include the sprites converted the SpriteConverter. Check the documentation for further
infos.
#include "explosion.h"
//pin configuration
const int redPins[] = {2, 4, 12, 13, 14};
const int greenPins[] = {15, 16, 17, 18, 19};
const int bluePins[] = {21, 22, 23, 27};
const int hsyncPin = 32;
const int vsyncPin = 33;
//VGA Device
VGA14Bit vga;
//initial setup
void setup()
```

```
{
  //need double buffering
  vga.setFrameBufferCount(2);
  //initializing i2s vga
  vga.init(vga.MODE200x150, redPins, greenPins, bluePins, hsyncPin, vsyncPin);
  //setting the font
  vga.setFont(Font6x8);
}
//just draw each frame
void loop()
{
  //draw a background
  for (int y = 0; y < vga.yres / 10; y++)
    for (int x = 0; x < vga.xres / 10; x++)
      vga.fillRect(x * 10, y * 10, 10, 10, (x + y) & 1 ? vga.RGB(0, 128, 0) : vga.RGB(0, 0, 
128));
  //print some labels
  vga.setCursor(36, 41);
  vga.print("draw   drawMix  drawAdd");
    //check the Utilities folder for the converter
  //"draw" draws the sprite opaque ignoring any existing alpha channel
  explosion.draw(vga, (millis() / 50) % 20, vga.xres / 4, vga.yres / 2);
  //"drawMix" uses the alpha channel
  explosion.drawMix(vga, (millis() / 50) % 20, vga.xres / 2, vga.yres / 2);
  //"drawAdd" adds the color components of the back ground and the sprite
  explosion.drawAdd(vga, (millis() / 50) % 20, vga.xres * 3 / 4, vga.yres / 2);
  //swap the frame buffers and show the rendering
  vga.show();
}
```

Chapter Five: Reference

- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
- Ben Eater: https://eater.net/vga
- Filerobot Image Editor Documentation: https://docs.filerobot.com/plugins/image-editor/
- Mozilla Developer Network (MDN) web docs: https://developer.mozilla.org/en-US/docs/Mozilla/PDF.js
- https://github.com/bitluni/ESP32Lib
- Apache Cordova Official Website. (https://cordova.apache.org/)
- Adobe PhoneGap: What is PhoneGap? (https://phonegap.com/about/)
- Lee, J. (2015). A Comparative Study of Cross-Platform Development Approaches for Mobile Applications. Journal of Software Engineering and Applications, 8(8), 407-418.
- Satyanarayana, G., & Babu, R. (2017). Comparative analysis of cross-platform mobile development frameworks. International Journal of Advanced Research in Computer Science, 8(6), 1896-1900.
- Ali, S. S., & Adi, A. (2017). Performance Evaluation of Cross Platform Mobile Development Frameworks. International Journal of Computer Applications, 168(8), 1-7.
- "ESPAsyncWebServer Library for the ESP8266 and ESP32 Arduino Core." GitHub, https://github.com/me-no-dev/ESPAsyncWebServer.
- Singh, Harsh. "How to Build an Asynchronous Web Server with ESPAsyncWebServer on the ESP8266." Random Nerd Tutorials, 15 Nov. 2018,
- TJpgDec - Tiny JPEG Decompressor
- Faber, T. (2015). TJpg_Decoder. [Code repository]. Retrieved from https://github.com/ThomasFaber/TJpg_Decoder
- Wallace, G. K. (1992). The JPEG still picture compression standard. Communications of the ACM, 34(4), 31-44.
- Delp, E. J., & Mitchell, O. R. (2000). Image compression using the JPEG standard. In Handbook of Image and Video Processing (pp. 133-156). Academic Press.
- Robinson, A. (2014). Understanding JPEG Compression. Retrieved from https://blog.imagekit.io/understanding-jpeg-compression-d85e40fcda69
- Sawicki, M. (2017). JPEG Decoding in Embedded Systems.
- The official FabGL website: https://github.com/fdivitto/FabGL
- The FabGL documentation: https://fabgl.gitbook.io/fabgl/
- Espressif Systems. (n.d.). ESPmDNS. Retrieved from https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/mdns.html