

EyeSPy

Authors: Varun Rajesh, Neelansh Kaabra, Michael Lang
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

Abstract— Our project aims to offer a portable and cost-effective solution for enhancing campsite safety with a Camping Perimeter Security System. Designed specifically for the wilderness, this system provides an early warning to campers about potential intrusions within a 50-meter range, allowing ample time to take appropriate measures. Unlike existing models that are expensive and reliant on internet connectivity, our system ensures complete 360-degree surveillance of the campsite without the need for external networks. With a substantial battery life that supports three days of operation, it is ideal for remote outdoor use.

Index Terms—Camera, Compression, Computer Vision, ESP32, FPGA, IR, JPEG, Open-Source, Discrete Cosine Transform

1 INTRODUCTION

In an era where outdoor activities are increasingly popular, ensuring the safety of campers in remote locations has become paramount. The use case of our project is a Camping Perimeter Security System designed to provide early warning signals to individuals in outdoor environments. The importance of such a system is underscored by the growing need for security measures in camping areas, where the remoteness and lack of connectivity render traditional systems ineffective.

Campers require a system that is not only portable and robust but also operates independently of internet connectivity. Our product serves this need by offering a stand-alone, comprehensive surveillance solution that monitors the vicinity of the campsite. This system is particularly crucial for campers who venture into areas where wildlife or other potential threats may encroach upon their temporary habitat. By using our system, campers can enjoy peace of mind, knowing they have complete surveillance to any possible disturbances with ample time to react.

Current competing technologies typically rely on a stable internet connection and are often cumbersome, making them unsuitable for the unpredictability and mobility inherent in camping. The ones that are portable and easy to use, are generally way too expensive for a camping purpose. Our approach offers a significant advantage by delivering a system that is lightweight, low-cost, and autonomous, requiring no external power or internet sources.

The primary goal of our project is to enhance the safety and security of outdoor enthusiasts by delivering a user-

friendly and reliable security system that adapts to the various challenges posed by wilderness environments. Secondary goals include ensuring the system's affordability and ease of use, promoting wider accessibility and adoption.

2 USE-CASE REQUIREMENTS

The Use-Case Requirements for our product can be divided into three major components, each pivotal to our project's success:

2.1 Continuous Streaming

The system is designed to provide continuous, around-the-clock surveillance of the campsite through a sophisticated streaming mechanism. Our product incorporates multiple camera nodes strategically placed to cover the entire camping area, ensuring comprehensive monitoring. The cameras will help provide surveillance and allow for campers to take adequate actions upon seeing a threat. The quality of the video frames will be such that the user can easily discern between the objects seen and take appropriate measures accordingly. The system will be optimized for energy efficiency, guaranteeing adequate streaming for a 3 day camping trip without compromising the system's effectiveness.

2.2 Portable Camera Nodes

Camera nodes are the pivotal components of the security system, designed for portability and resilience. Since we envision our product to be used in a campsite, and campers will be moving every few hours, our entire system will be easy to install without burdening the user. The product has to be fully wireless and weather-resistant, capable of functioning in diverse environmental conditions.

2.3 Reduced Costs

A key requirement for the system is affordability, ensuring that it is accessible to a wide range of outdoor enthusiasts. To this end, our entire system should not be highly expensive and strike a balance between cost and functionality. This cost strategy is crucial in providing a competitive advantage in the market, making advanced camping security systems economically viable for consumers.

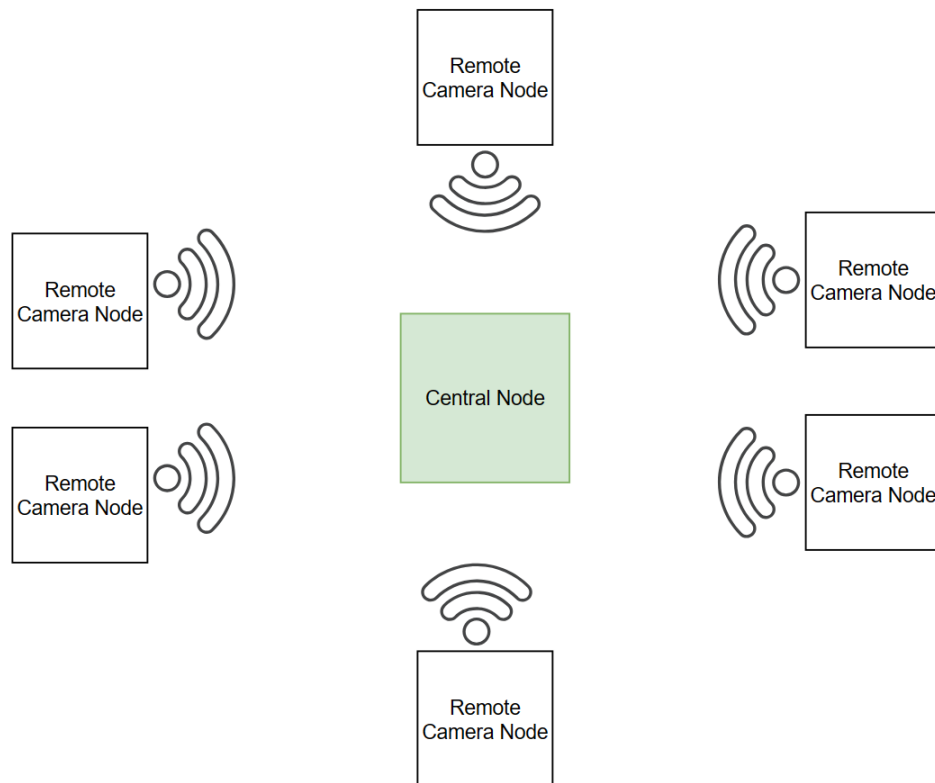


Figure 1: The overall architecture with 6 remote camera nodes

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

The overall architecture of this project consists of a distributed architecture of remote nodes and a single central node. Figure 1 shows off this high level description of the minimum viable product: 6 remote camera nodes and a single central node. The remote camera nodes stream their video feeds to the central node which displays it.

Taking a closer look, the remote node will consist of an OV2640 camera. The OV2640 camera will be modified to remove its IR filter, and a motorized IR filter will be placed over the camera. This will be used such that the camera can support both day-time running and night-time running. To further support the night-time running, the remote node will be fitted with IR LEDs.

The remote node will also house an ESP32 microcontroller. This will be responsible for interfacing with the camera itself. Then, it will perform a modified form of JPEG encoding and Run Length Encoding with the goal of minimizing the amount of data that has to be transmitted. Finally, using the onboard WiFi of the ESP32, the information from each frame will be transmitted to the central node.

This will all be powered by an appropriately sized LiPo battery to support a 24 hour runtime. In addition, a 3D printed enclosure will be placed around these elements to

create a weather resistant enclosure.

On the other side of the system is the central node. The central node is responsible for ingesting the video streams from the remote camera nodes. The central node consists of an ESP32 and a Lattice ECP5 FPGA.

The ESP32 will act as a transceiver, receiving wireless packets from the remote camera nodes and then transmitting them to the FPGA over a QuadSPI interface. The FPGA will then apply the opposite of the encoding algorithm implemented on the camera nodes to recover the video stream. The FPGA will then store these video frames in its memory and will create a composited image of all the camera streams. This final video stream will then be outputted to a portable monitor. This will all be powered by an appropriately sized LiPo battery to support a 24 hour runtime.

4 DESIGN REQUIREMENTS

Each of our three main use case requirements can be derived into multiple design requirements that support them. We break this down by use case requirement.

4.1 Continuous Streaming

From our requirement of being able to monitor a full 360 degree coverage of our campsite, we have identified that we

will require 6 cameras to cover the full field-of-view. Then to support the ability to identify the objects that are within the vicinity of the camera, we will require the video feeds to be 240p resolution with a frame rate of 10Hz.

To ensure that the user can use it in any sort of conditions, even during the night time will require that our solution have night vision capabilities.

Then, to ensure that the user can actually see the video streams that are being transmitted, it is required to have at least a 720p display. This is the minimum resolution for an HD display and provides adequate resolution to discern objects and monitor their surroundings.

These come together to form our first main set of design requirements, we must be able to stream at least 6 cameras, at 240p, at 10Hz. This must be displayed on a 720p monitor without dropping more than 10% of the frames.

4.2 Portable Camera Nodes

For our requirement of having a portable camera node, we are able to derive some more design requirements. A portable camera system cannot contain many, if any, wires. This would make the system clunky and incompatible for wilderness uses cases. This means that our system must be wireless and run on its own batteries.

To make it easy to setup the cameras anywhere, it is required to have at least a 50m range from the portable remote camera nodes to the central node. This will also provide the user with sufficient response time to take appropriate action if something is detected around the campsite.

In addition, to make sure that our solution is compatible with wilderness use cases, we must make sure that it is fully weather resistant and can handle being in the rain.

4.3 Reduced Costs

Having done a survey of the market of products for security systems, we have determined that the current solutions are wildly expensive. Because of this, we have determined that a fair price for each of these components is as follows.

The remote camera node should be at most \$50 while the central node should be at most \$150. This means that the cost of a total system comes out to be less than \$500.

5 DESIGN TRADE STUDIES

5.1 Remote Node Compute Selection

The remote camera nodes have to take in the data that is sent from the camera's image sensor and then transmit that data to the central node to display to the camper. The need to decode raw camera sensor data and transmit that data over Wi-Fi can require a substantial amount of processing. Furthermore, to maximize transmission efficiency and battery life, we plan on using a compression system to minimize the size of each frame. Using compression also

extends the range of our system since the data rate requirements are lower and thus a less aggressive and more noise tolerant Wi-Fi modulation schema can be used. The use of compression requires that the compute node be sufficiently powerful to complete all the compression steps in 100 ms (10 fps). Even though compression does take energy to run, it is all made up in the time saved transmitting RF signals.

The remote camera node also needs to have sufficient amounts of memory to serve as a frame buffer. A frame buffer is needed to enable compression and the benefits that it brings. Frame buffers must reside in fast and high endurance memory. The needs for high speed comes from the fact that this frame buffer is in the hot path of the processing and high endurance since every frame will write to this buffer

Power Consumption is also an important factor. As a battery operated system, minimizing power consumption can lead to either better system endurance or a more portable system. Thus, the compute node shouldn't have too onerous power requirements to maintain our run time and portability requirements

The platform must also have built-in Wi-Fi connectivity. Having Wi-Fi built into the compute module means that it will be significantly easier to implement the needed functionality and save on BOM cost since we won't need to purchase a separate chip.

Cost is the final factor that was considered. Consideration of cost is primarily driven by the requirement to minimize cost and stay below \$50. Thus, the node that we choose should reflect this requirement.

5.1.1 CC3200

The CC3200 is an ARM M4 based Wi-Fi MCU that is made by Texas Instruments (TI). The power consumption figures are very respectable at 720mW at full transmit power and the CPU at full utilization. However, the deal breaker for our purposes was the chip's unit cost. TI is asking for \$50 for just a single chip if not bought in bulk. The remote node being below \$50 is a requirement and using this chip would immediately mean that we would be unable to meet the target. Other unattractive properties include a slow CPU that can only be clocked as high as 80 MHz.

5.1.2 Raspberry Pi Zero W

The Raspberry Pi Zero W (RPI Zero W) is a single board computer that runs a full Linux installation with built-in Wi-Fi capability. The RPI Zero W is comparatively extremely cost-effective compared to the CC3200, at a retail price of only \$15. Furthermore, it has a powerful CPU that is clocked at a whopping 1 GHz. The CPU is also coupled with an extremely strong memory subsystem with a capacity of 512 MB. Of note, is that since the RPI Zero W runs a full Linux operating system, some of the compute and memory will be unavailable to us due to the overhead of Linux. The biggest issue stopping us

from using the RPi Zero W is its unacceptably high power consumption of 2.5W. This level of power consumption is fundamentally incompatible with our requirements to be portable and lightweight. Therefore, we decided not to proceed with using the RPi Zero W

5.1.3 ESP8266

The ESP8266 is a Wi-Fi microcontroller designed by Espressif Systems that integrates a Tensilica L106 CPU running at 160 MHz. A clock rate of 160 MHz is sufficient for our needs of running the compression algorithm at 10fps. The unit cost is also acceptable for our requirements at only \$8 per chip. Power consumption is also acceptable at 600mW. However, the issue with the ESP8266 is its minuscule amount of memory. The ESP8266 has only 80 Kb of user accessible memory. A full frame buffer of 240p requires at least 230 Kb. Without the ability to store a full frame, it becomes very difficult to implement the compression and transmission code. To avoid making the system needlessly complicated, bug prone, and to leave room for expandability it was decided against using the ESP8266 as the compute for the remote node

5.1.4 ESP32

The ESP32 is the successor to the ESP8266 that further builds on where the ESP8266 left off. The ESP32 has a dual core Tensilica LX6 running at 240 MHz. Compared to the ESP8266, the ESP32's CPU improves both on the clock speed and adds a second parallel processing core, giving us a 3x improvement in the number of clock cycles available to use. Furthermore, the ESP32 also dramatically improves on the weak memory system of the ESP8266. The ESP32 contains 520 kB of on board SRAM and up to 4 MB of offboard PSRAM. The greatly strengthened memory and compute system also serves as a good foundation for future expansion should we want a higher resolution or frame rate. The ESP32 is also extremely cost effective at only around \$2.24 each at retail pricing. Such a low price gives us a lot of margins to hit our needed budget of \$50. The final factor that made the ESP32 very attractive to us was its power efficiency. Even at full CPU load and transmitting over Wi-Fi, the ESP32 draws less than 800mW. Most of that power consumption is a result of the Wi-Fi transmitter transmitting. With just the CPU running at the maximum clock speed of 240 MHz, it draws only around 130 mW, as a result of it being fabricated on a TSMC 40 nm processes.[5]

5.2 Compression Algorithm Selection and Modifications

5.2.1 Delta Compression

The idea behind delta compression is to only transmit the difference between frames instead of transmitting the complete frame. This relies on the fact that during most

of the time, there will be no significant differences in subsequent frames since the camera is just looking at a static scene. In theory, if no pixel changed then only a no change message needs to be send yielding extremely high compression ratios. The algorithm is also trivial to implement and run since it just needs to compare the pixels without the need to do any complicated steps. As good as these properties are, the issue of this algorithm is that each frame builds on the next one. Therefore, if a message was dropped then following frames will start having artifacts. This is still a problem even if we use a reliable transport protocol such as TCP since it is possible that the camera loses communication with the central node and will result in a TCP timeout. Further compounding the issue is that if there is a major change, such as an animal entering the frame, there will be a spike in the bandwidth demanded since now every single pixel that was updated needs to now be sent over. Therefore, to ensure that we meet the requirement of 10fps even when there are large changes, we will need to design the system to handle transmitting a complete uncompressed frame in less than 100ms, even though most of the time we will not be using all that bandwidth.

5.2.2 H.264 & MPEG-2

H.264 and MPEG-2 are two very commonly used video compression algorithms. Unlike delta compression, H.264 and MPEG-2 are tolerant to packet loss since they don't depend on the previous frames. Also unlike deltas, H.264 is able to reliably achieve 2000:1 compression ratio. However, the issue with these two algorithms are their very high implementation and runtime complexity. H.264 encoding is particularly taxing with desktop level hardware being able to only achieve 10fps or so. This makes it immediately not viable for our platform of a low-power microcontroller unless we include a dedicated co-processor. Including a co-processor would drive up the cost and also increase the power consumption, negatively impacting our requirements

5.2.3 JPEG

JPEG encoding is most commonly used for compressing pictures but nothing stop this from being used to compress a stream of pictures that form part of a video stream. JPEG has reasonable implementation and runtime complexity. The only heavy step in the pipeline is the discrete cosine transform and the inverse discrete cosine transform. Even these are not too bad since many methods exist to speed up the computation. JPEG is able to achieve a 10:1 compression ratio depending on image data. Even though this is nowhere close to the 2000:1 of H.264, it is sufficient for our needs of compressing a 240p image. The 10:1 ratio is also only a general case with JPEG steps allowing the compression quality to be adjusted so that we can hit the needed ratio. Another attractive property of JPEG compression is that it splits the image into minimum coding blocks (MCU) before running the compression on the MCUs.[6]. Therefore, if for some reason a MCU is cor-

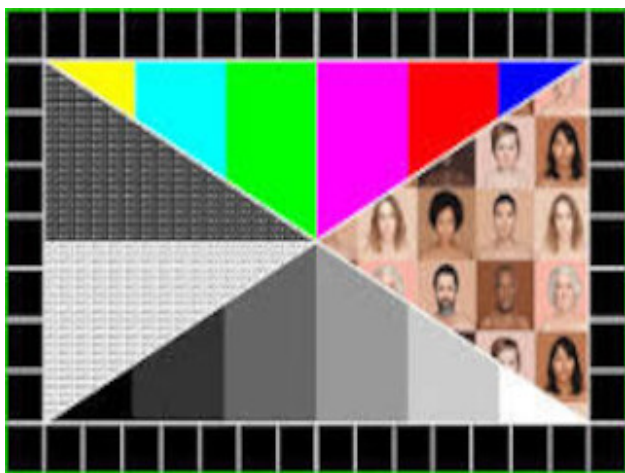
rupted or dropped during transmission, only that MCU will be affected. Since there is also no inter-frame dependency, the MCU data is completely refreshed when the next frame is received. If an MCU does get corrupted or lost then it will only last for a single frame and the next frame's data will fix it. MCUs are also typically very small, only around 8x8 pixels. Any damaged data will be limited to a 8x8 MCU which at 720p will be almost invisible

5.2.4 JPEG Cr Cb Channel Truncation and Packing

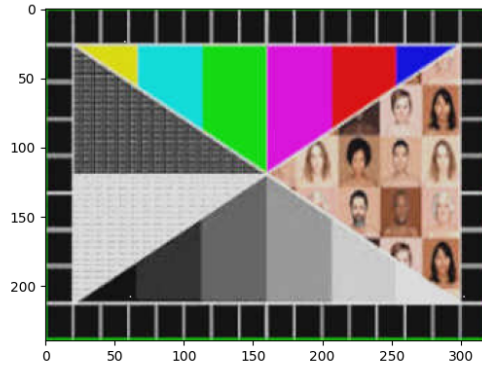
The first step of JPEG compression is to transform the color space from RGB to YCrCb. Instead of RGB where all three channels are color, YCrCb only has two color channels, Cr and Cb, with one luminance channel, Y. The human eye is more sensitive to luminance changes so by isolating the luminance to a single channel, the compression is able to better preserve the image. In RGB, this is not possible since the algorithm will have to ensure that the overall luminance doesn't change too much by juggling three channels instead of just one channel in YCrCb. Converting to YCrCb also enables other methods to be used to trick the human eye into thinking the compressed image is closer to the original.[1]

One method that is possible is to truncate the lower 4 bits of Cr and Cb before repacking them into a combined 8 bit channel. This takes advantage of the fact that human are less sensitive to color changes and therefore the lost information is not significant. By doing so each pixel turns from 24 bits into only needing 16 bits in their uncompressed form

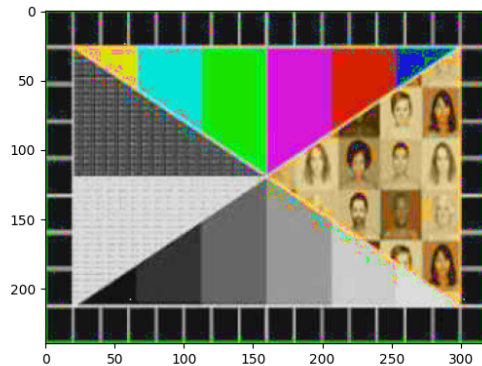
The following series of figure illustrates that effect of this change



Original Uncompressed Image

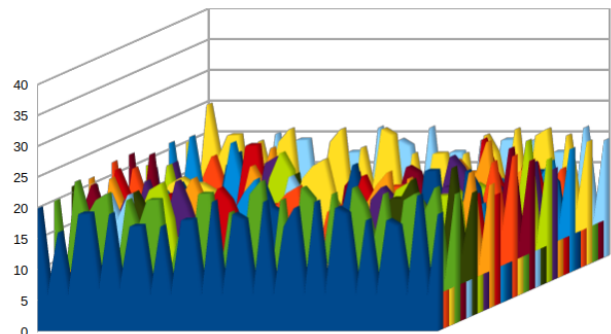


Compressed With No Truncation

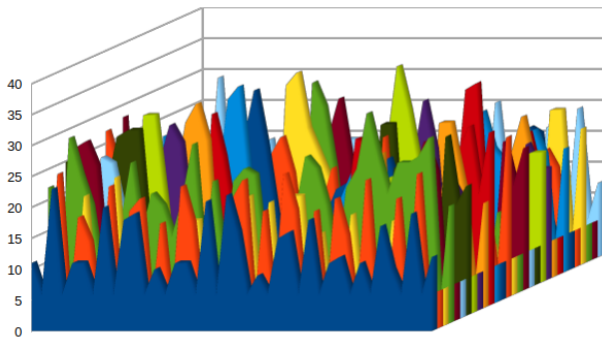


Compressed With CrCb Truncation

It is evident that the CrCb truncation results in noticeable degradation in quality, particularly in the regions where there are significant changes. While there is a compression improvement, the improvement was only around 5%. The reason for this is a increase in entropy of the data. The combined CrCb channel now has almost double the entropy compared to encoding them alone. Therefore, it negates most of the space saving. This is clearly evident when we look at the final MCU encoded length. The following graphs on the Z-axis show the number of coding blocks used for each MCU. X and Y-axis show the index of the MCU



Original Entropy on Cr Channel



Entropy of combined CrCb Channel

As a result of these properties, we have decided to shelf this idea unless the need for this arises. A 5% increase can be useful in case we need but the artifacts make it unwise to use unless absolutely needed

5.2.5 JPEG No Change Control Message

Drawing from the ideas of delta and looking into the actual use case of the camera, we decided to incorporate a no-change data message. If there is not a significant change in the image, then we will only send a no change message and skip the JPEG encoding. Only when there is a large change will JPEG encoding be used. What is deemed significant will be up to the user to select according to the specific situation and battery life requirement. Regardless of what sensitivity level is set by the user we expect this will contribute significantly to extending the system battery life. We used the following equations to produce an estimate of energy savings. The calculations assume a Wi-Fi data rate of 11 Mbps, 4 bytes for no change message, 0.8W during TX, 6:1 compression ratio for JPEG

$$S_{JPEG} = 240 * 320 * 3/6 = 38400bytes$$

$$T_{Nochange} = 4bytes/11Mbps = 0.36\mu s$$

$$T_{JPEG} = 38400/11Mbps = 3490\mu s$$

$$E_{Nochange} = 0.8W * 0.36\mu s = 2.9\mu J$$

$$E_{JPEG} = 0.8W * 3490\mu s = 2792\mu J$$

The following Table 1 shows the total energy consumption over 100 frames, or 10 seconds with varying percentages of time when there is no significant change

It is clearly evident that incorporating this in our design could save a lot of energy, up to almost 10x in the case where 90% of the frames don't have any change

5.3 Central Node Compute Selection

The central node will be responsible for handling the input of the 6 (or more) camera streams that are being fed in by the remote nodes. This main requirement sets out a few considerations in place for what we select for the central node compute.

One of the most obvious requirements is that the central compute needs to be powerful enough to decompress

the incoming video streams. Beyond raw processing power, this requires us to have enough memory on our compute to store the decoded frames into the main frame buffer that will be eventually displayed.

Next there needs to be a way to actually drive a display. One of our design requirements is that the display driven needs to be at least 720p. Because of this, we cannot use VGA which caps out at 480p. We need to find a node that supports protocols like DisplayPort, HDMI, or DVI.

The next requirement is that the compute has to be power efficient. Since this central node is battery powered and is meant to be a part of a portable system, we would like the compute to consume as little power as possible.

The compute must also have built-in Wi-Fi connectivity. Having Wi-Fi built into the compute means that it will be significantly easier to implement the needed functionality and save on BOM cost since we won't have to purchase a separate Wi-Fi module and integrated that into our design.

Another requirement is to have an open-source toolchain for these computes so that the design is extensible in the future.

Finally, the last main requirement is for the compute to be relatively inexpensive. One of our requirements is that the central node costs less than \$150, which means that the actual compute shouldn't exceed \$100.

These driving factors led us to look into three main classes of products, microcontrollers, System-on-Chips, and FPGAs. To summarize the tradeoff study completed, Table 2 is provided at the end of this section.

5.3.1 Microcontrollers

Of the three main classes of products explored, microcontrollers are typically the lowest powered and cheapest product, but that comes with the caveat that they are lacking in features and lacking in raw compute horsepower.

The main microcontroller explored here was the ESP32. This was explored because it was chosen for the remote node and would make the integration between the two platforms extremely simple. This product does satisfy some of our requirements however it does lack into main areas. The first of which is that there's no easy way to interface the ESP32 with a high resolution display. The second issue is that given the ESP32s 240MHz cores, it will not suffice to be able to run decompression on the 6 incoming video streams.

Due to these reasons we continue our exploration into other options.

5.3.2 System-on-Chips

System-on-Chips are typically processors that run full blown operating systems on them. Common examples of these are the Raspberry Pi family of products. We take a look at two of the Raspberry Pi's that seem appropriate for our use case: the RPI 4 W and the RPI 0 W.

The Raspberry Pis are attractive due to their high compute capabilities. The RPI 4 W has a four cores clocked

Table 1: Energy Usage Comparison

Percent of No Change	JPEG	JPEG + No Change
50%	279200	139745
75%	279200	70017.5
90%	279200	28181

at 1.5GHz with 4GB of RAM which is more than sufficient for running the decompression algorithms. It also supports display out over HDMI. The main issue however with this is that the power consumption of the product is too high.

Likewise, a similar story can be said of the RPI 0 W. The single core of the RPI 0 runs at 1.0GHz with 512MB of RAM which is also more than sufficient for running the decompression algorithm. It also supports display out over HDMI. The main issue again is the power consumption.

5.3.3 FPGA

FPGAs are usually extremely energy efficient compared to their microcontroller/System-on-Chip counterparts. To be consistent with the goal of using open-source toolchains, we chose the Lattice ECP5 FPGA with the ULX3S carrier board.

This FPGA is an attractive choice because of its fast 400MHz clock speed. Even though this lower than the clock speed of the other System-On-Chips, the main benefit is that every single clock cycle is significantly more powerful since it's an FPGA. Similarly, it has a 32MB DRAM chip on board which is large enough to store the frame buffer for the display.

Another benefit of this choice is that it contains an on-board GPDI connector, this is essentially an HDMI connector without the licensing attached with HDMI. The FPGA supports driving a DVI signal over the GPDI connector for an external monitor. Finally, the power consumption of this FPGA is very reasonable for the amount of performance it offers.

Finally, the choice of this specific FPGA is powerful because onboard the carrier board is a Wi-Fi enabled ESP32 which makes it very simple to setup a Wi-Fi access point for the remote nodes.

5.4 Communication Protocol Selection

The main driving design requirements for the selection of communication protocol are the requirements of having at least 50m+ range, having enough throughput to support video streaming, and to be an extensible platform in terms of current and future availability.

There were four main protocols that were considered for this. Bluetooth Low Energy, LoRa, ESPNow, 2.4 GHz Wi-Fi.

5.4.1 Bluetooth Low Energy

Starting with Bluetooth low energy, this protocol does a few things well. It is an extensible platform as it is widely available and supported by multiple devices, and it can also support the necessary 50m range required, however the main drawback is that it does not support the necessary throughput. Bluetooth LE is only capable of up to 2 Mbps.

5.4.2 LoRa

Next we looked at LoRa, this protocol does a few things well. It is an extensible platform as it is widely available and supported by multiple devices. It has a range significantly greater than the 50m required for this product, however the main drawback, similar to that of Bluetooth Low Energy, is that it does not support the necessary throughput. LoRa is rated up to only 250 kbps.

5.4.3 ESPNow

Next we looked at ESPNow. This protocol is well suited for the task of streaming video. It can support the 50m range required as well as has sufficient throughput of 54 Mbps to support video streaming. However, the main drawback is that it is not an extensible platform for future development which could potentially drive up costs in the future. This protocol is tied to the ESP family of products and there is no guarantee that it will be supported in the future.

5.4.4 2.4 GHz Wi-Fi

Finally, we looked at 2.4 GHz Wi-Fi. This protocol seems to check all the boxes laid out in our design requirements. It can support the 50m range required as well as can support up to 150 Mbps of throughput which is well compatible for video streaming. Finally, Wi-Fi is also an extensible protocol and is extremely available currently, and there are no plans in the future for the world to switch away from Wi-Fi.

6 SYSTEM IMPLEMENTATION

A detailed visual of the system implementation is displayed in Figure 2.

Table 2: Central Compute Tradeoff

	ESP32	RPI 4 W	RPI 0 W	ECP5
Clock Speed	240MHz x2	1.5GHz x4	1GHz x1	400MHz x1
RAM	4 GB	512 MB	536 kB	32 MB
Cost	\$55	\$15	\$3	\$60
Power	8000 mW	2500 mW	660 mW	600 mW

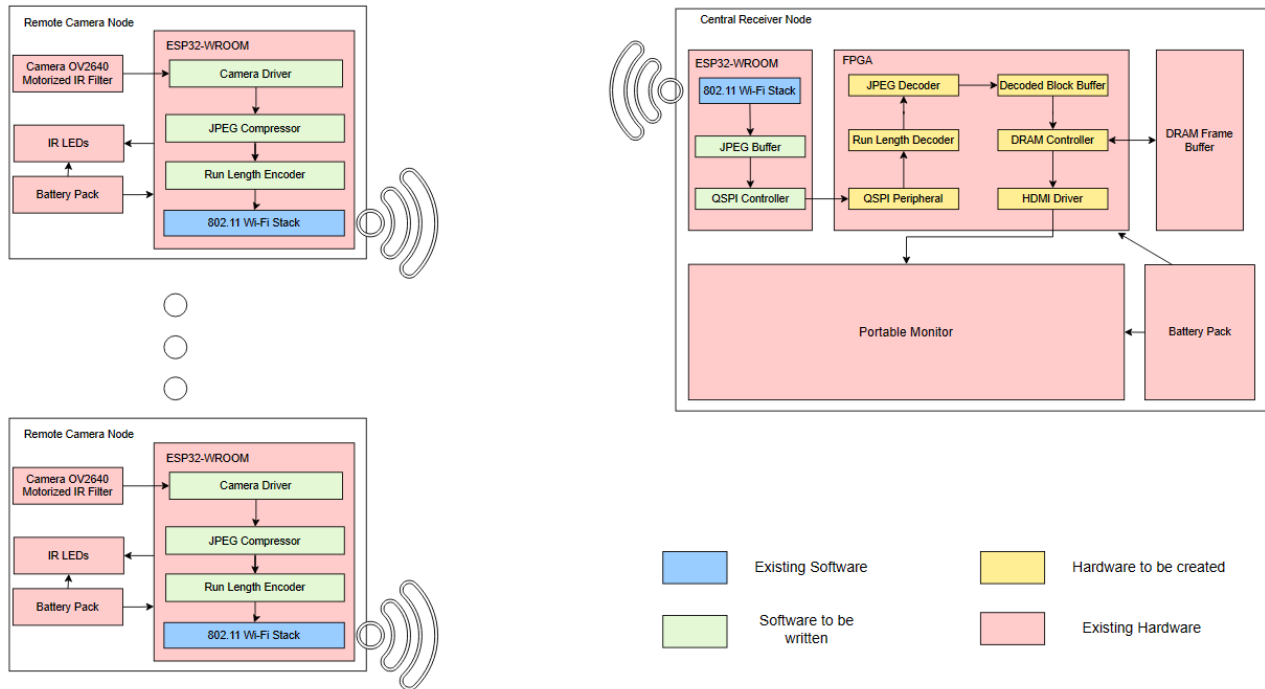


Figure 2: A detailed look into both the remote camera node and the central node with a legend detailing what parts of the stack come from where.

6.1 Remote Node Implementation

6.1.1 Color Space Conversion and Change Detection

Once the camera finishes capturing a full frame the first step that will be undertaken is the conversion of the RGB565 that is returned by the OV2640 into YCrCb color space. Not only is this critical to following JPEG compression steps but this also makes detecting changes in the field of view easy. The color space we will be using is the one specified in JFIF which itself is a modification of the color conversions used in Rec. 601. The follow equations are used to perform the conversion[3]

$$\begin{aligned} Y &= (0.299 * R) + (0.587 * G) + (0.114 * B) \\ Cb &= (-0.1687 * R) - (0.3313 * G) + (0.5 * B) \\ Cr &= (0.5 * R) - (0.4187 * G) - (0.0813 * B) \end{aligned}$$

Detecting changes in the RGB is much harder due to the fact that the three channels of RGB only encode color. It is very possible for the color of a pixel to change slightly due to noise and natural variances. So if one were to implement a change detector in RGB then it would have to somehow account for all 3 channels as once. Additionally, when operating in IR mode and in night, we don't expect any primary colors to show up making detection even harder. But in YCrCb space this becomes much easier. If there is a significant change such as an animal entering the frame, we expect that there will be a lot of luminance changes whether in IR mode during the night or during daytime. Therefore, in YCrCb, the change detection only needs to work on the Y change and doesn't have to account for slight color changes. Other benefits include easier computation due to only needing to account for 1 channel and lower memory requirements from only having to keep around 1 channel. The change detector will be implemented as follows where *MCHANGE* being the maximum number of pixels allowed to change before declaring it a significant change and *THRESHOLD* being the value difference tolerated before saying that pixel has changed

$$c(i, j) = \begin{cases} 1 & \|Y'[i][j] - Y[i][j]\| \geq THRESHOLD \\ 0 & else \end{cases}$$

$$s() = \begin{cases} CHANGE & \sum_{i=0}^{239} \sum_{j=0}^{319} c(i, j) \geq MCHANGE \\ NO CHANGE & else \end{cases}$$

Should *s()* return no change then the JPEG steps will be skipped and the remote camera will instead send a **NO CHANGE** message to the central node. Only if a change will detected will the JPEG encoding steps proceed. Such a design was chosen since most of the time there will be no change in the camera's picture and it is much more

efficient to just not update the image than going through the expensive process to encode then send the same data over

6.1.2 JPEG Encoding

Should a frame be JPEG encoded, the remote camera node will first perform a discrete cosine transform (DCT) on a 8x8 submatrix of the picture, which is more formally called a minimum coding unit (MCU)[6]. Within each MCU, a type-II DCT transformation is performed to transform the 8x8 matrix into a matrix of coefficients that correspond to certain frequencies. Once the matrix is transformed in a series of coefficients of frequencies, the amount compression can be easily adjusted by including or omitting certain coefficients. The type-II DCT also conveniently places the lower frequencies towards to top left corner of the matrix, making it easy to decide how much data to include[4]. The type-II DCT is governed by the following equations:

$$\begin{aligned} \alpha(u) &= \begin{cases} 1/\sqrt{2} & u = 0 \\ 1 & u \neq 0 \end{cases} \\ coeff_{ij} &= value_{ij} \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right) \\ mcu_{ij} &= 0.25\alpha(u)\alpha(v) \sum_{u=0}^7 \sum_{v=0}^7 coeff_{ij} \end{aligned}$$

Of note, is that the DCT is the first step that is not fully reversible[6]. In extreme situations, the frequency of the changes might be so big no frequency component within the 8x8 matrix is able to cover it. However, this is very unlikely given that the coding unit is only 8x8.

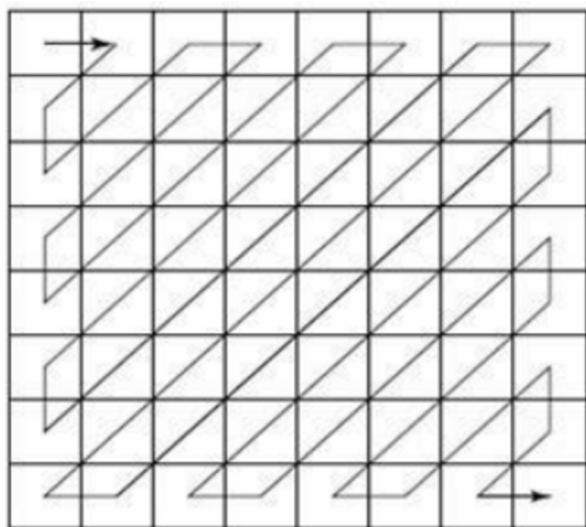
6.1.3 Quantization

After DCT encoding, the next step is quantization. The quantization matrix exploits the DCT property that most of the substance of a image will still be present if the higher frequency components are removed[4]. Therefore, what happens in the quantization step is that each coefficient in the coefficient matrix is divided by a value. The value that it is divided by depends on the coefficient's location. Coefficients located closer to the top left are divided by a smaller value and coefficients located closer to bottom right are divided by higher values. The results are then rounded to the closest integer number. If the coefficient is deemed not significant given its original value and the dividing factor the end result will be 0. This property is then used by the run length encoder to achieve the compression of data

Like the DCT step, the quantization step is irreversible given that it rounds and is a major source of error in the final image. But given that this loss is concentrated on the higher frequency components, its effect on the quality of the final image is not significant

6.1.4 Run Length Encoder

Drawing from the property introduced by the quantization step where insignificant components are zero-ed out, the run length encoder uses this to finally compress the data. Instead of explicitly encoding each and every single zero, the number of zeros before a non-zero value is encoded. This way a single byte can potentially cover many bytes of zero. To even further increase the compression ratio, the run length encoder traverse the array such that it goes from coefficients of lower frequency to coefficients of high frequency. Such a traversal maximizes that chances to encounter a chain of all zeros since high frequency components are likely to have been zero'd out by the quantization step. The traversal method used is called zigzag and the following graphic illustrates the method [6]



Logically, the run length encoder outputs a data stream of tuples. The first element of the tuple is the non-zero value and the second element is the number of 0s the go before this non-zero value. After encoding a full MCU, the run length encoder will terminate by appending a special block of (0,0).

While other more efficient methods of encoding data do exist, this tuple data structure was chosen primarily to aid the FPGA decoder in parallel decoder. By having the special (0,0) block, the FPGA can use this block to delimit where each thread should start and end therefore making it easy to decode MCUs in parallel. Other methods would force the FPGA to perform the MCU decoding in series, negating a lot of the benefits of a FPGA-based solution.

In code, the following C-structure is used to represent the tuple

```
typedef struct {
    int8_t val;
    uint8_t zeroLen;
} rle_t;
```

The run length encoder encodes the channels in order of Y, Cr, Cb. Within each channel it traverses the MCUs in row-major order. Within each MCU the encoder fol-

lows the aforementioned zigzag method. For each tuple produced, including the special (0,0), it is appended to the end of the result array.

6.2 Central Node Implementation

6.2.1 ESP32 Transceiver

The purpose of the ESP32 transceiver is twofold. It will be the access point to which all of the remote nodes will connect to and it will be the communication bridge to the FPGA over which this data will be transmitted.

We will be using ESP32 with builtin Wi-Fi capabilities and so the wireless functionality is provided as off-the-shelf code and we will be using this to setup the access point. The remote nodes will transmit their frames over this wireless connection. The ESP32 will be responsible for buffering the incoming data until it receives all the information required for one MCU.

Once the ESP32 has received the data for this MCU, it will transmit it along with addressing information, over a QuadSPI interface to the FPGA. The QuadSPI interface is selected as an easy way of having a high bandwidth connection between the ESP32 and the FPGA.

6.2.2 Run Length Decoding

The run length encoding outputs tuples that express the number of zeros before the current values and the current non-zero value. The blocks are expressed as the same way as in the previous section

Henceforth, the structure will be represented as a tuple of (val,zeroLen). Each of these tuples will be packed together in an array. Each channel's array is then concatenated in the order of Y, Cr, Cb;

All values of *val* are allowed without any limitations. The field of *zeroLen* allows all values from 1-63. The only tuple that is the exception is the end of minimum coding unit (MCU) tuple which is expressed as (0,0).

Within the MCU, the order that is followed is the zigzag array traverse method. For each tuple in the MCU, the decoder will write the amount of zeros in order of zigzag and then write the non-zero values. Any MCU matrix values that are not directly addressed are implicitly set to 0. The MCU decoding continues until it sees a end of MCU tuple (i.e (0,0)). This process is then repeated for every MCU in the 240p image in row-major order. After compiling a complete 240p channel, this is then repeated again the subsequent channels following the Y, Cr, Cb order.

Each MCU is independent to every other MCU in the image, even across different channels. We observe that this decoding is very parallel and very suited for acceleration in hardware. The only part that can't be parallelized is the initial step of finding the end of MCU markers that are embedded in the data. After these markers are found then the actual writing of data is easy.

6.2.3 JPEG Decompression

JPEG decompression happens in the reverse order of the JPEG compression steps. For decompression, once a 8x8 MCU matrix has been run length decoded, the first operation that is done on it is to dequantize the matrix. Dequantizing the matrix simply involves multiplying each value in the decoded MCU by the quantization matrix values in a element by element order. However, the dequantization step doesn't actually perform any decompression in to reverse the compression introduced by the quantization step. The dequantization step merely scales up the non-zero values to be close to their original value. The zero values are not scaled and thus are lost forever to the rounding that happened in the quantization step. The pseudocode for this operation is expressed below

```
for i in range(8):
    for j in range(8):
        mcu[i][j] *= quantizeMatrix[i][j]
```

After the quantization is completed, the matrix then has the inverse discrete Fourier Cosine Transform (iDCT) applied to it. The iDCT step as the name implies reverse the DCT step done in the encoding. More concretely, the iDCT step transforms the matrix from a series of cosine frequencies with varying weighting factors to a 8x8 matrix of regular scalar values and not a matrix of cosine weight.[4] Some data is also further lost in the this due to the use of integer values but it should be relatively small compared to the data that is lot during the encoding steps. In line with the original JPEG standard, the modified JPEG algorithm that we will be using will use the type-II iDCT. The following equation governs the iDCT step

$$\alpha(u) = \begin{cases} 1/\sqrt{2} & u = 0 \\ 1 & u \neq 0 \end{cases}$$

$$dct_{ij} = DctCoeff_{ij} \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

$$mcu_{ij} = 0.25 \sum_{u=0}^7 \sum_{v=0}^7 \alpha(u)\alpha(v)dct_{ij}$$

Following the iDCT steps, the MCU have now been transformed into matrices that hold a non-normalized image data of either Y, Cr, or Cb. The last remaining step is to then normalize the YCrCb data and then convert the colors into RGB values before being written in to the frame buffer. The color space conversion is that we are using is the standard transform that is specified in the JFIF standard, which itself is a variant of the transform specified in Rec. 601 that supports the full range of uint8 values. The following constant are used in our program[3]

$$R = Y + (1.402 * Cr)$$

$$G = Y - (0.344136 * Cb) - (0.714136 * Cr)$$

$$B = Y + (1.772 * Cb)$$

6.2.4 Memory Subsystem

Once an MCU has been decompressed, there needs to be a way of storing that data so that it can be displayed. At a high level, multiple MCUs get buffered in BRAM and then when enough of them have been accumulated, they get written to DRAM.

Typically, the DRAM chip (W9825G6KH-6) lets you perform reads and writes with a word size of 2 bytes. However, with any DRAM, there is a CAS latency associated with it. The CAS latency of this chip is 2 clock cycles. We can calculate the theoretical max data rate as:

$$f w / (CAS + 1) \quad (1)$$

where f is the clock frequency, w is the word size, and CAS is the CAS latency. The clock speed of the DRAM is at most 165MHz, the word size is 2 bytes, and the CAS latency is 2 clock cycles, thus the theoretical max data rate is 110 MBps.

Now we can calculate what the necessary data rate is to support a 720p display with

$$h v p f \quad (2)$$

where h is the number of horizontal pixels, v is the number of vertical pixels, p is the number of bytes per pixel, and f is the frequency at which the display is updated. For a 720p display, h is 1280, v is 720, p is 2 bytes, f is 60 Hz. This yields a minimum data rate of just over 110 MBps.

As we can see here the theoretical max data rate is similar to that of the required data rate for the display. This is clearly not feasible for two reasons. The first is that the theoretical max data rate will never actually be achieved because DRAM has to spend time refreshing the memory banks so the actual data rate will be less. The second is that if we spend all of our time reading from DRAM, there's no time to actually write into DRAM updating the displayed image.

To get around this with opt to using something called page writes and page reads which changes the transaction to look like 512 consecutive 2 byte writes. This increases our theoretical data rate to 328 MBps. This is now more than enough to support reading and writing from DRAM for the display.

To support the page writes we have to add the aforementioned buffering to the MCU data. Every MCU is 8x8 with 2 bytes per pixel. We buffer 40 of these MCUs into a 320x8 pixel array. We opt to use 320x8 as this evenly fits into a 1280x720 display. Then once the buffer of MCU data is full, we then utilize page writes to write each row of the 320x8 array to DRAM. This means that each page write writes 320 pixels per write.

6.2.5 GPDI Driver

Due to the requirement of not wanting to license the HDMI specification, we opt to use the GPDI connector on board the ULX3S. We drive this with a DVI style signal which is compatible with the HDMI specification. There

are three main parts with the GPDI driver. The first is a VGA style display driver, then a TMDS encoder, and then a DDR clock out of the data.

First we begin with the VGA style display driver. The basis for this DVI style signal is an VGA style display signal. The general format of this signal is that row-by-row the data for each pixel is driven out. Then there are blanking periods following each row, and then following each frame to allow for the display to synchronize on the frames.[2]

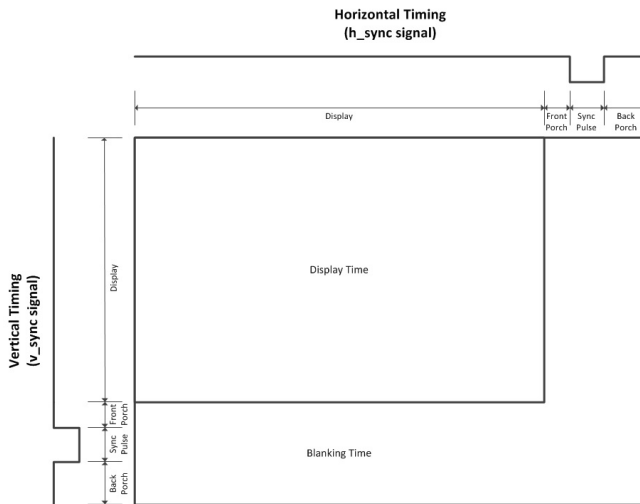


Figure 3: A diagram of the VGA timing specification

For the standard CVT VGA timing for 720p displays, we would need to run our pixel clock at 372.5 MHz. This is extremely close to the maximum frequency of the FPGA at 400 MHz. This makes it difficult to synthesize a design that will actually meet timing. Instead we opt to use something called CVT-RB. The RB stands for reduced blanking which means that the VGA driver spends less time in the horizontal/vertical porches. This lets us run our design at 320 MHz instead which is much easier to meet timing for.

Specifically this means that in the CVT mode, we spend 384 pixels blanking in the horizontal direction and 28 lines blanking in the vertical direction. In the CVT-RB mode, we spend only 160 pixels blanking in the horizontal direction and 21 lines blanking in the vertical direction.

The next part of the protocol is called TMDS encoding the signal. TMDS is an 8 bit to 10 bit conversion protocol that aims to minimize the number of bit changes in the signal. This minimizes the amount of interference and the decreases the likelihood of having bit errors in the signal.

TMDS operates by computing two representations of a byte. They both start with the first bit but then either applies the XOR or XNOR operator on each successive bit pair. Then, the TMDS encoder looks at which of these representations has the least number of bit flips and transmits that message. This gives us 8 bits of the 10 bit TMDS message. The next 9th bit is used to represent whether the XOR or the XNOR operator was used. The 10th bit determines whether the output byte will be inverted or not inverted. This is used to ensure that the DC offset of the

byte (essentially the average voltage of the line) is as constant as possible.

7 TEST & VALIDATION

Outlined below are the tests we plan on conducting to ensure effectiveness of our design and implementation while being able to hit all the requirements set in the use-case and design parts.

7.1 Tests for Ensuring Run Length of the System

One of the major requirements set in both the Use-Case and Design sections was to ensure that the entire system runs for at least 24 hours, to allow for 3 days of adequate streaming for a camping weekend. To ensure that our system is able to provide clear streaming without overheating and being able to maintain IR functionality, we plan on testing our system at different hours of day and night, consecutively for 3 days, replicating the real life scenario. We will also test IR functionality in darker hours, and see the effect different situations have on the battery life of the camera and central nodes. We will see how having the system run for 4 hours, 6 hours and 8 hours at a stretch affects the battery life and design our system accordingly in an iterative process.

7.2 Tests for Wireless Streaming

We need to ensure that the data being transferred between the camera nodes and the central receiver node is correct, and enough for our use case of being able to stream continuously, while being able to easily discern between different objects in the stream. Our design requirements described that we need to ensure no more than 10% of frames get dropped. Keeping this requirement in mind we will transmit varying frames from the remote to central nodes, and measure the amount of frames sent vs the number of frames received. This test will be conducted for varying timings (2, 4, 6 hours), essentially checking that the system is able to transmit enough frames even if the nodes are run for hours at a time. Our tests will also be conducted with a 50m range and varying objects such as trees and bushes to simulate real life conditions.

7.3 Tests for Central Receiver Display

The Display on the Central Receiver needs to be able to receive all the frames with sub 10% drops and ensure adequate display driving for 24 hours. This will involve sending frames from the 6 remote nodes and then receiving them on the central receiver, then displaying all the 6 streams concurrently on the screen, without excessive lags and glitches in the display. We will be measuring the frame rate and display diagnostics including streaming capability

and quality. This tests stems from our use case requirement of being able to display 6 streams at the same time, allowing for continuous observation of all surroundings to complement our campsite security purpose.

7.4 Tests for Performance

Any security systems needs to be efficient enough to display the streams to the user in a timely manner, for appropriate action to be taken. From this essential requirement stems the needs for one of the most important test for our system. Performance testing would involve majorly measuring the throughput of the compression and decompression algorithms and hardware. We will run our algorithms on 100+ outdoor images and frames from actual camera feed and ensure that compression takes less than 100ms, while decompression is possible in ≤ 16 ms (to allow for decompression from at least 6 camera nodes). We will be targeting a 5:1 compression ratio.

7.5 Tests for Scalability

Our system provides security surveillance to campsites, and in the future might need to expand to allow for more than 6 camera nodes. Our use case necessitates expandability and consequently our system has to allow for usage of more than 6 camera nodes for bigger campsites and extensive surveillance purposes. We plan on first reaching a suitable working performance with at least 6 camera nodes, to set a baseline for our testing, and after that try testing our system with 8, 10 and 12 camera nodes to determine our system limitation without increasing the total cost by a lot.

8 PROJECT MANAGEMENT

8.1 Schedule

We are on track as of now and are maintaining a steady pace as per our schedule. We have started work on almost all components and plan on finishing in due time. The schedule via a gantt chart is shown in Fig. 5.

8.2 Team Member Responsibilities

We have divided team member tasks and responsibilities based on each member's past experience in each field and interests. Varun Rajesh would be working mainly on the FPGA on the receiver node which performs the decompression of the incoming frames from the camera nodes and then drives them to the display for concurrent viewing. Neelansh Kaabra will be working on developing the software stack for the decompression, and will be implementing the receiver ESP32 architecture on the receiver node to adequately receive all the incoming frames from multiple data points. Michael Lang will be working on developing the remote camera nodes, the compression algorithm and

the transmission from the remote camera nodes to the central receiver node. The testing and verification of all the components will be done together involving everyone. All of us will be working to a certain extent on each part of the system and maintain a constructive environment throughout the project.

8.3 Risk Mitigation Plans

Given the high amount of wireless data transmission and high performance needs we have quite a few risks to take care of. That being said, we have risk mitigation strategies in place as follows:

8.3.1 Computing Frame Compression Fast Enough

Given that we want continuous streaming and consequently continuous compression of the captured camera feed, we need to be able to compute the frame compression fast enough. This concern also arises since we are using ESP32's (which do not have extremely high compute capabilities) to maintain low cost. If this is a problem, we plan on either optimizing our algorithm, or using the hardware accelerators present on the ESP32s for our transmission. Our solution also involves using parts of the Delta technique, so most of our streaming time would involve sending minimal delta (assuming low activity in the surroundings).

8.3.2 Stream enough data over the wireless connection

Since we plan on sending data streams from 6 camera nodes, there is a high amount of data being transmitted, and consequently high risk of congestion on the receiver node. We plan on adding more data access points on the receiver ESP32 if this occurs.

8.3.3 Decompress incoming frames fast enough

We need to be able to decompress the incoming frames from all the 6 camera nodes fast enough. If the decompression algorithm isn't fast enough with our current choice of FPGA, we plan on either optimizing our decompression algorithm as the first alternative, and worst case use a larger FPGA with more compute power.

8.3.4 Minimize power consumption

One of our major use case requirements was to ensure at least 24 hours of runtime. This is also one of our concerns as we have a lot of different components running at the same time (cameras, ESP32's for sending and receiving, FPGA, Display, etc.). If we have issues with this, we plan on either looking through our system, finding the most power consuming parts, and try to optimize them. Worst case we would shift to larger battery packs.

Table 3: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost / Unit	Total
ESP32 Microcontroller	ESP32-CAM	Espressif	6	\$3	\$18
Open Source FPGA	ULX3S	Radiona	1	\$60	\$60
Camera	OV2640	OmniVision	6	\$1.68	\$10.08
Motorized IR Filter	M12IR	Zerone	6	\$3.52	\$21.12
Patch Antenna	2308	Adafruit	6	\$2.50	\$15.00
3D Printer Filament	PLA	Elegoo	1	\$13.85	\$13.85
Portable Monitor	1366x768 11.6"	wanlusiri	1	\$54	\$54
100Wh Power Bank	27W 100Wh	Talentcell	1	\$48	\$48
16Wh Power Bank	354	Adafruit	6	\$19.95	\$119.70
Total					\$360

9 RELATED WORK

The following related works provide valuable insights for our product:

1. Arlo Pro Series (Arlo Technologies): Arlo's line of wireless security cameras is renowned for its ease of installation, high-quality video, and efficient battery use. The Arlo Pro series offers features like motion detection and night vision, similar to our focus on change detection and IR functionality. However, we differentiate ourselves with the JPEG no-change control mechanism, designed to optimize data transmission efficiency and extend battery life further.

2. Wyze Cam Outdoor (Wyze Labs): This product is designed for outdoor surveillance with features such as wireless operation, weather resistance, and low-power consumption modes. Wyze Cam Outdoor's approach to energy efficiency and its use of Wi-Fi for communication are paralleled in our design. Nonetheless, we advance beyond by integrating FPGA-based processing for enhanced video decompression and display capabilities.

3. ESP-EYE (Espressif Systems): The ESP-EYE development board from Espressif Systems features an ESP32 chip and supports image recognition and audio processing. While primarily a development tool, its use of the ESP32 for low-power, high-performance processing is mirrored in our remote node design. We extend this concept by focusing on outdoor surveillance and incorporating a system-wide approach to energy efficiency and data management.

4. LoRa-based Surveillance Systems: Various projects have explored the use of Long Range (LoRa) technology for remote surveillance, capitalizing on its long-range communication capabilities and low power consumption. While LoRa excels in range and power efficiency, its limited data throughput makes it less suitable for high-definition video streaming. Our choice of 2.4 GHz Wi-Fi addresses this limitation, providing a balance between range, throughput, and energy efficiency.

10 SUMMARY

The Design Review Report details our innovative surveillance system designed for outdoor and remote monitoring. The report emphasizes energy efficiency, particularly through a novel JPEG no-change control mechanism that significantly extends battery life by avoiding unnecessary data transmission. The central node, powered by an FPGA, offers an optimal balance of processing power and energy efficiency, crucial for handling multiple video streams. The system employs 2.4 GHz Wi-Fi for reliable communication, ensuring efficient video streaming across the network.

Key components include JPEG compression for efficient data handling, sophisticated remote node implementation for capturing and processing video, and a central node design focused on minimal power consumption and high performance. The selection of an FPGA-based solution underscores the project's commitment to cost-effectiveness and performance.

Challenges ahead include optimizing energy use, ensuring dependable Wi-Fi transmission outdoors, and managing the computational demands of real-time video processing. Overcoming these obstacles is crucial for our system to meet its goal of enhancing outdoor campsite security monitoring effectively.

Glossary of Acronyms

- BRAM - Block Random Access Memory
- CAS - Column Address Strobe
- CVT - Coordinated Video Timiming
- DCT - Discrete Cosine Transform
- DRAM - Dynamic Random Access Memory
- FPGA - Field Programmable Gate Array
- FPS - Frames Per Second
- GPDI - General Purpose Differential interface

- iDCT - Inverse Discrete Cosine Transform
- IR - Infrared
- JFIF - JPEG File Interchange Format
- JPEG - Joint Photographic Experts Group
- JPEG - Joint Photographic Experts Group
- LoRa - Long Range
- MCU - Minimum Coding Unit
- MPEG-2 - Moving Picture Experts Group-2 Standard
- PSRAM - Pseudo Static Random Access Memory
- Rec. 601 - International Telecommunications Union Recommendation 601
- RGB - Red Green Blue
- RPi - Raspberry Pi
- TCP - Transmission Control Protocol
- TMDS - Transition Minimized Differential Signaling
- TSMC - Taiwan Semiconductor Manufacturing Cooperation
- VGA - Video Graphics Array
- YCrCb - Luminance, Chrominance Red, Chrominance Blue

References

- [1] Christian J. van den. Branden Lambrecht. *Vision models and applications to image and video processing*. Springer, 2011.
- [2] Digital Display Working Group. *Digital Visual Interface*. Tech. rep. 1999.
- [3] Eric Hamilton. *JPEG File Interchange Format*. Tech. rep. C-Cube Microsystems, 1992.
- [4] David Marshall. *The Discrete Cosine Transform (DCT)*. URL: <https://users.cs.cf.ac.uk/dave/Multimedia/node231.html>.
- [5] Espressif Systems. *ESP32 series datasheet*. Tech. rep. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.
- [6] International Telecommunication Union. *INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES*. Tech. rep. International Telecommunication Union, 1992.

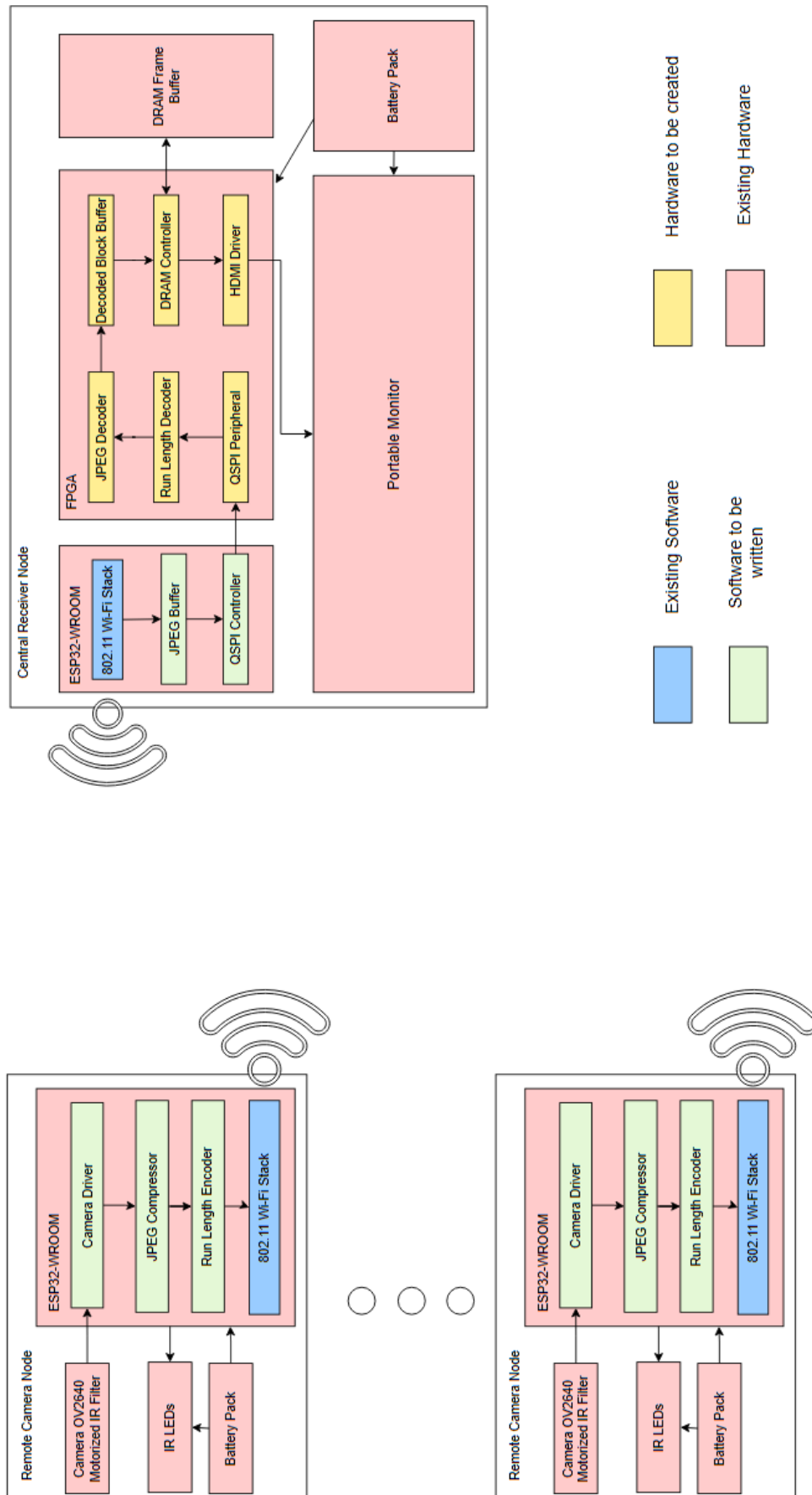


Figure 4: A full-page version of the same system block diagram as depicted earlier.

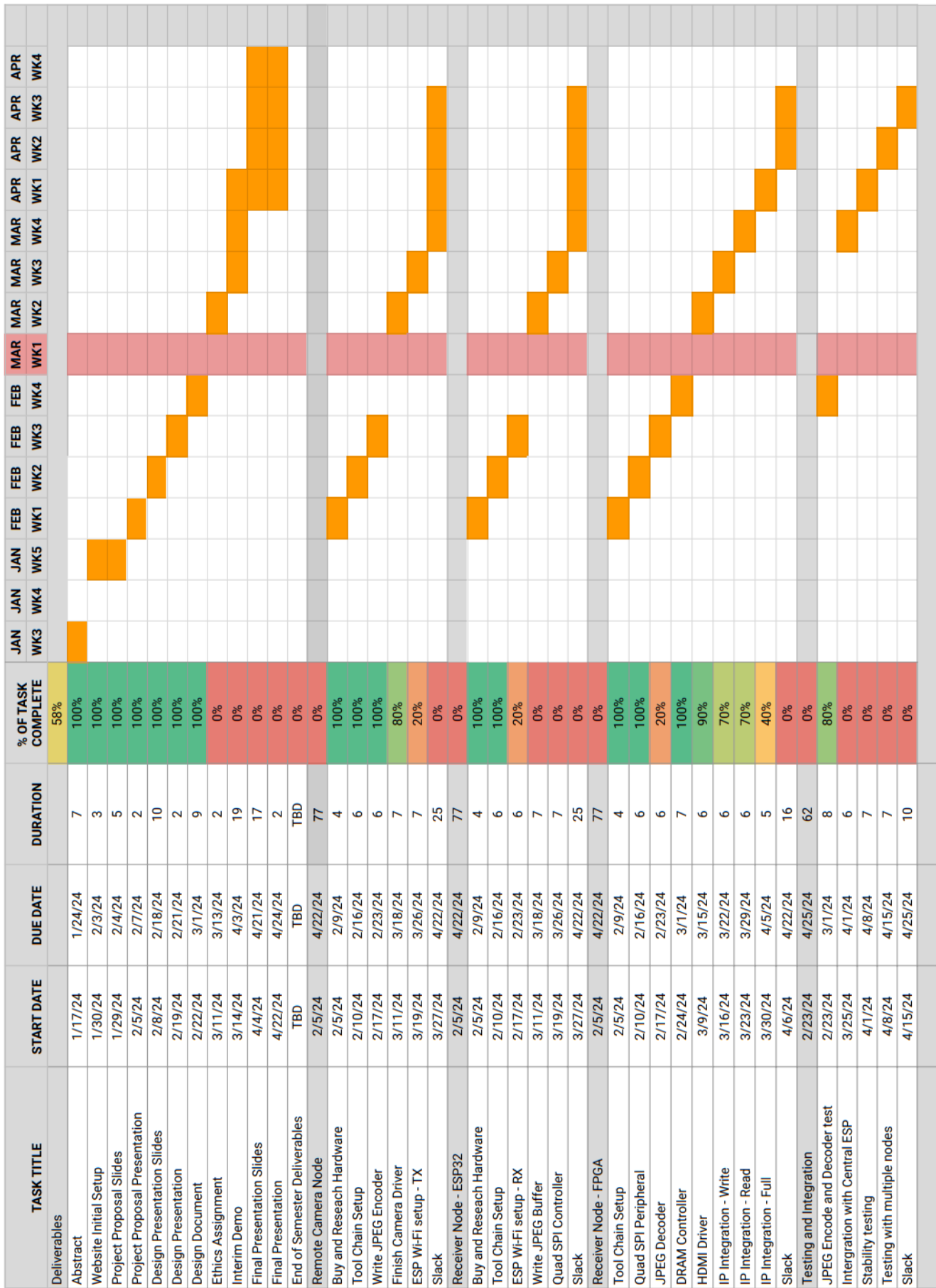


Figure 5: Gantt Chart