Ukrainian Catholic University

Bachelor Thesis

Real-time VGA to UART converter using FPGA

Author:
Dmytro Vasylkiv

Supervisor:
Vladyslav Khmelevskoi

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences and Information Technologies
Faculty of Applied Sciences

Lviv 2025

"*Computers are like Old Testament gods; lots of rules and no mercy.*"

Joseph Campbell

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Real-time VGA to UART converter using FPGA

by Dmytro Vasylkiv

Abstract

Developing an embedded system constrained by low throughput is a complex process. When designing the solution, choosing the proper processing unit and designing a suitable data processing pipeline is often the key to achieving the desired application performance. This work uses a field-programmable gate array (FPGA) technology to implement a transmission of real-time video data through the low-bandwidth asynchronous transmitter. The target is to develop a solution that will use the FPGA's advantages to successfully handle the input data volume far exceeding the system's output capacities.

We start by introducing the concepts crucial to understanding the problem's context and underlining the suitability of the FPGA's architecture for such tasks. Then, we proceed to the overview of the already existing solutions in the domain of data compression and serial video transmission. Lastly, we propose our solution, the Run-Length Encoding algorithm combined with the Frame Differencing algorithm, and report its expected resource usage and efficiency. The result is an adaptive solution that provides stable YUV422 video transmission up to 12 Frames per second at cropped VGA's 512x384p resolution.

You can observe the Verilog code and the simulations for the implementation on the GitHub repository, here.

Acknowledgements

I want to express my gratitude to my supervisor, Vladyslav Khmelevskoi, for his guidance and valuable advice during our work.

Also, I cannot leave unmentioned the unwavering support of my family and friends, without which this work would not be possible.

Contents

List of Figures

List of Tables

List of Abbreviations

FPGA	**Field-Programmable Gate Array**
VGA	**Video Graphic Array**
UART	**Universal Asynchronous Receiver-Transmitter**
SDRAM	**Synchronous Distributed Random-Access Memory**
FPS	**Frames per Second**
RLE	**Run-Length Encoding**
PLL	**Phase-Locked Loop**
DSP	**Digital Signal Processor**

Chapter 1

Introduction

1.1 Motivation

This project pursues the aim of transmitting video data from a low-resolution camera to a host device via UART with the use of a low-end FPGA chip. It pushes the boundaries of what's possible without the use of a CPU, while accounting for tight onboard resources and low system throughput. Being subject to the predefined restrictions, we aim to demonstrate the approach that will succeed in transmitting the large amount of video data through the unspecialized, low-bandwidth interface.

1.2 Contributions

In this work, we demonstrated the theoretical throughput capability of the designed system by implementing the real-time VGA-to-UART video data flow using Verilog HDL and AMD's Vivado design software. The design was targeted at an Xilinx Artix-7 FPGA and a 32MB SDRAM module. The total list of contributions :

- A system design that captures the video data from an OV7670 camera module and outputs the data to a UART transmitter
- A lightweight compression algorithm (frame differencing + run-length encoding with tolerance) that reduces the required throughput
- Verilog controllers for interaction with OV7670 camera, compression, and UART transmission
- A Verilog single data rate SDRAM controller for frame differencing
- The receiver-side Python script for decompressing and rendering the video data

1.3 Structure Of The Thesis

Chapter 2 of this thesis describes the basics of the VGA and UART protocols, discusses, without delving into the architectural details, the benefit of FPGA over the more common processing unit options like microcontrollers and graphic cards, and introduces the reader to the YUV color space used for encoding video data here.

Chapter 3 of this thesis mentions the achievements of the other developers and researchers in the sphere of data compression and video transmission using serial protocols.

Chapter 4 formulates the exact problem this project aims to solve and describes the constraints we are subject to in this task, as well as the hardware platform used for modeling the solution. Also, it provides detailed descriptions of the choices made while

designing this system. Appendices A and B should be considered the continuation of it, delving deeper into the details that are too cumbersome to describe in the Chapter itself.

Chapters 5 and 6 sum up this work. Chapter 5 provides the performance estimations of the system in different conditions, as well as the resource use statistics. Chapter 6 wraps up the contributions of this work and outlines potential future directions for this project.

Chapter 2

Theoretical Background

In this chapter, the concepts necessary for understanding the problem of this work are introduced. First, we will overview the basics of classic VGA protocol and the UART communication protocol we use as a representation of a low-bandwidth transmitter. After that, we will describe the specifics of using an FPGA platform for embedded systems, uncovering the reasons for choosing it for this project. Later, we will move on to discussing the importance of choosing the correct color space for our application, and introduce the YUV subsampling.

2.1 VGA Protocol Overview

The Video Graphics Array (VGA) protocol is a widely used solution for transmitting video data. Originally designed for displaying a picture on a computer monitor, the VGA frame capturing principle can also be utilized in cameras to transmit the captured data for further processing.

The VGA standard relies on strict timings and analog signal alteration to produce video data. In its core, there are five signal sources - vertical synchronization (*VSYNC*), horizontal synchronization (*HSYNC*), and *Red*, *Green*, and *Blue* outputs, which specify the values of each pixel of a video frame over a fixed, predefined time period.

Figure 2.1 displays the VGA timing basics. The *VSYNC* and *HSYNC* denote the start of a new frame and a new frame row, respectively. These denotions, in combination with the blanking periods - "porches" - comprise the timing scheme for a VGA controller to rely on when sampling video data. The *R-G-B* signals provide the value of the current pixel and are sampled relatively to the *HSYNC* timing with the account for the "back porch". Depending on the intended frame's size and frames-per-second rate, the VGA signal generator's frequency is adjusted to generate the necessary volume of video data.

2.2 UART

The Universal Asynchronous Receiver/Transmitter (UART) is a communication protocol for serial data transmission. Apart from the GND connection, UART requires only one wire for data transmission, and its simplicity and minimalism have earned it popularity in minimalistic embedded designs. The data transmitted via UART is organized in UART data frames - bit sequences consisting of a transmission start bit, 5 to 9 data bits, an optional parity bit to check for transmission errors, and a transmission stop bit.

The classical UART's baud rate may vary from 9600 baud up to 12 Megabaud while retaining the stability of transmitted data. Despite UART's commonness, it

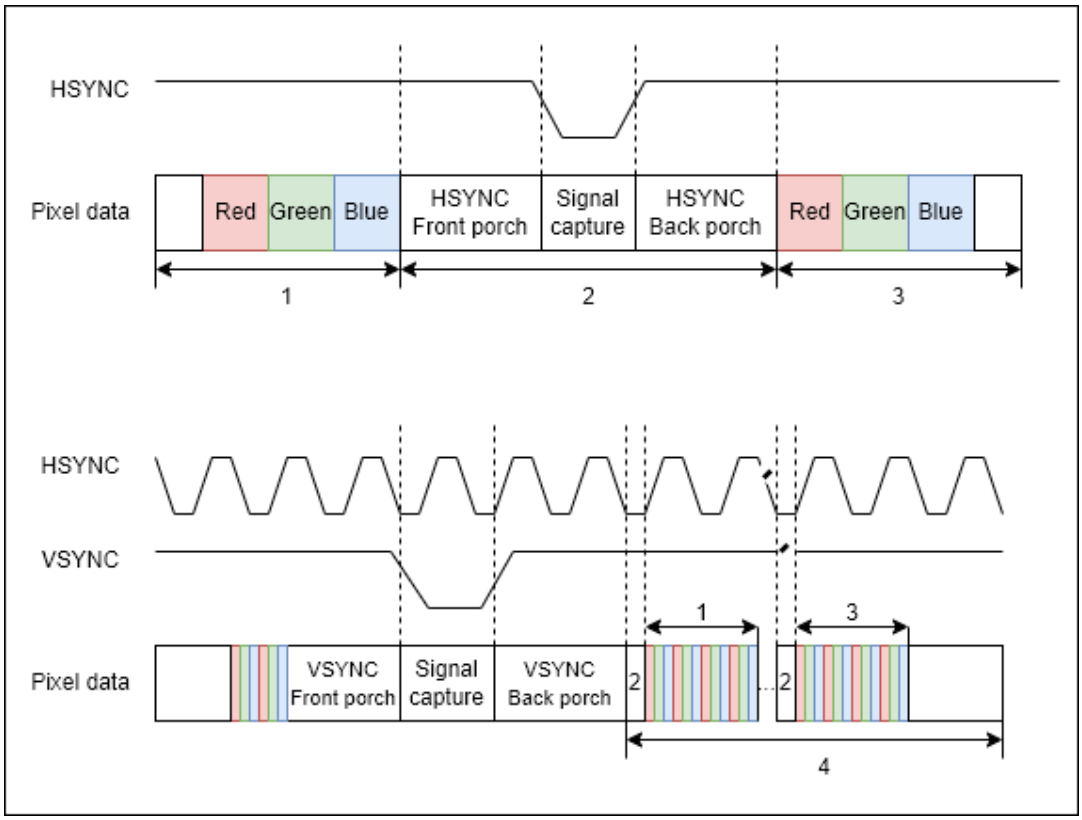

FIGURE 2.1: A simplified VGA timing scheme. 1 - frame's previous row data, 2 - HSYNC blanking period, 3 - frame's next row data, 4 - new video frame's data

is definitely not a choice in an application requiring a high bandwidth. A common UART baud rate is much lower than what the high-end industry chips provide, scoring a maximum baud of 921600.

2.3 FPGA Benefits Over MCU/GPU

The task of developing a real-time video processing system requires a suitable processing unit. The core focus is on the capabilities for quick, robust calculations - the system's driver should yield maximal throughput with minimal latency to surpass the limitations of low bandwidth transmitters. The focus is on choosing a processor with strong capabilities for performing operations in parallel, since the nature of video data strongly encourages the utilization of parallelism.

2.3.1 MCU's And GPU's Drawbacks

The traditional solutions, like microcontroller units (MCUs) and graphics processing units (GPUs), are widely used in embedded systems. However, when the task boils down to an efficiency-dependent edge computing task like video transmission, they all have their drawbacks that make them suboptimal.

Most MCUs lack the necessary power to effectively handle complex calculation tasks, due to their intended minimalistic, low-consuming designs. GPUs, on the other hand, offer massive parallelism and excel in handling large data streams, but come with high energy consumption and price, being an overkill for many edge tasks.

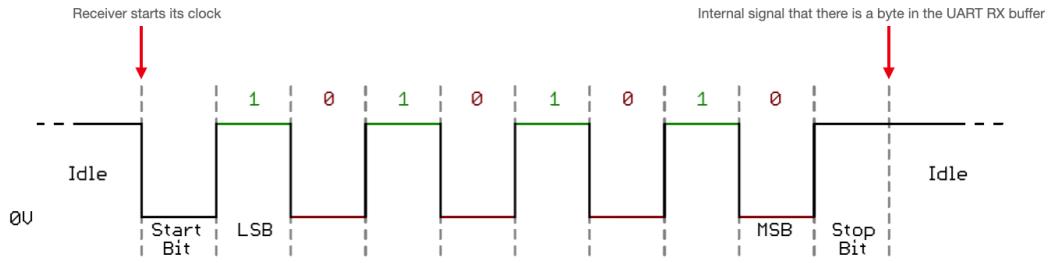

FIGURE 2.2: A visualization of sending an 8-bit value over UART. (Credit to [1], educational use only)

A more balanced alternative is necessary for this type of application that will introduce benefits from both traditional options.

2.3.2 FPGA's Suitability

A field-programmable gate array (FPGA) is a programmable logical block that allows the dynamic implementation of an electronic circuit. Utilizing the concept of lookup tables (LUTs) for implementing virtually any logical function, FPGAs have large potential for creating an optimal, efficient, and robust solution - provided that the FPGA chip has enough resources for it, that is.

The universality of an FPGA is its most substantial advantage over more common options. Unlike rigid application-specific integrated circuits (ASICs) with predefined architecture, FPGAs provide developers with options for on-field design changes or prototyping using hardware instead of software solutions. Unlike MCUs, FPGA chips offer true hardware parallelism abilities, allowing multiple hardware blocks to work simultaneously, which tremendously benefits the task of video processing. FPGAs also consume less power than GPUs in the general case, and finally, they allow for more flexible design solutions than both.

2.4 Color Spaces

The introduction of a suitable color space for the video data is important for achieving a successful encoding performance. RGB, the most common color space for image or video encoding, typically uses a three-channel structure, 8 bits by channel for Red, Green, and Blue, respectively. However, it is not the best choice for video transferring and processing due to the redundancy of the color representation - the human eye is not as perceptive to the changes of color [12].

The YUV color space, in turn, is based on the fact that the human eye is more sensitive to brightness variations rather than color variations. Thus, it operates within the luminance and chrominance values, encoding the brightness as Y and blue-red colors as a U-V pair.

This color space comes with many valuable advantages over RGB, like allowing us to represent colors that could not be adequately represented by RGB [5] and overall being more natural to human perception.

More importantly, it enables different techniques to reduce the size of the captured data. Since the Y-channel carries most of the crucial information about an image, U- and V-channels are often subsampled, resulting in lower bandwidth with little to no compression artifacts. Figure 2.3 shows the most common variants of subsampling, which are widely used in video codecs and encoding algorithms.

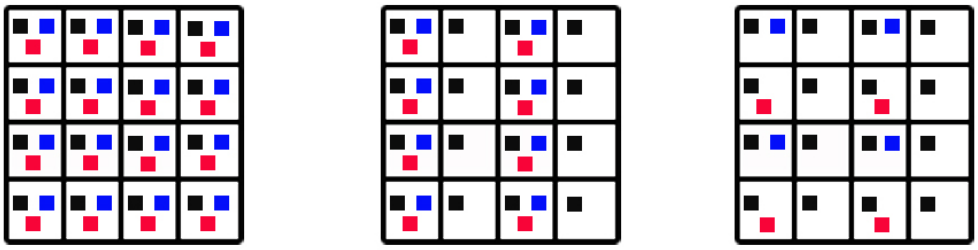

FIGURE 2.3: Different chroma subsampling formats, from left to right: 4:4:4, 4:2:2, 4:2:0. Black, blue, and red squares represent Y, U, and V channels per pixel, respectively (Credit to [15])

This technique alone makes YUV a better fit for bandwidth-limited applications than RGB. But YUV also presents an option for a simple grayscale video frame generation if needed, simply by isolating and transmitting the luma channel only. If color transmission is not a crucial requirement for an application, a simple modification to the embedded system using YUV color space will further reduce the required bandwidth.

Chapter 3

Related Work

This chapter provides an overview of the existing solutions, highlighting the techniques and hardware specifications used to grant an acceptable system's throughput. It provides an overview of their functionality, their main ideas, and their pros and cons. This chapter also seeks to highlight the inspirational points used for designing the proposed solution and to underline the contributions of the implemented solution in the next chapters.

3.1 Serial Protocols For Video Transmission

The lack of bandwidth necessary to transmit large data arrays is a solid reason for not using a serial protocol for handling video transmission whenever possible. An important step for the success of this project is to assess the existing solutions that attempt to do so, and note their functionality, performance and limitations.

3.1.1 FLIR's LEPTON VoSPI

LEPTON thermal camera devices implement Video over Serial Peripheral Interface (VoSPI) to transmit video data on a low-resource embedded system. FLIR's VoSPI [8] is a proprietary protocol that uses a fixed packet structure to transfer the uncompressed video data over an SPI interface. The packet includes a checksum, a packet identifier, and 160 bytes (80 pixels) of payload. The 60 packets sum up a VoSPI frame that represents the 80x60 video frame.

FLIR's VoSPI allows a VoSPI-capable hardware to transfer a grayscale thermal video of 80x60 resolution at the rate of 9Hz. It also provides robust error checking capabilities and stability checks, which are important to keep a stable video data flow in an embedded system.

3.1.2 Arducam SPI module

Arducam SPI cameras [4] offer a solution for embedded systems that do not have the resources for working with a continuous flow of video frame data. Arducam is not used for a continuous video transfer, but rather for filming a picture or a short (up to 91 seconds) video. It uses the camera's built-in JPEG compressor and sends the data over SPI in chunks with a data transfer rate of 4 Mbps - 8 Mbps. Arducam also uses buffering to eliminate the need for constant, real-time frame handling.

It's also worth mentioning that Arducam uses an FPGA-based controller for handling the complex timings of the camera module, as well as for organizing the data arrays writing to the on-board buffer.

3.2 Visual Data Compression

The proper choice of compression algorithms is essential to fully utilize the parallelization and throughput capabilities provided by a processing unit. For the low-resource embedded systems using an FPGA, the key focus is on the techniques that provide easy parallelization of the process, don't require expensive computational operations, and are undemanding in terms of the FPGA's local RAM storage. When observing the existing solutions, the nature of the data the solution is working with should also be aligned with the requirements of a system under development.

3.2.1 Run Length Encoding

Run-length encoding is a simple yet effective lossless algorithm that is widely used in data compression. It is especially effective for generally low-variance data like images and has many variations aimed at reducing the load on the system with low throughput. Classic RLE reaches the data equilibrium by coding the continuous data segments as the *value-amount* pairs, which are then decoded by the receiver.

Despite its assumed effectiveness, there is a potential for encoding to enlarge the frame size instead of reducing it [11]. [18] propose the modified RLE algorithm, which uses the neighboring pixels to predict the next frame's content, and utilizes the 3-bit *amount* limit to prevent the overhead coming from the lengthy *amount* values, achieving the 64% improvement of the compression factor and demonstrating that the modifications of RLE are essential for increasing the effectiveness of the compression pipeline.

3.2.2 Delta Encoding

Applying the Delta encoding to data means transmitting the differences between the adjacent values (e.g. pixels) rather than the values themselves [14]. The first data entry is chosen as the pivot, and the other entries are transmitted relatively to it as difference values. Like RLE, delta encoding is suitable for video frame processing due to the image data's mostly smooth variability.

Similarly, it shares the same flaw of not being able to deal with the occasional unpredictability of the data. An interesting solution comes from the cache line compression background - the Base-Delta-Immediate (BDI) compression[10]. This technique introduces the additional base value used to improve the efficiency of encoding outliers like zeroes. Although the main focus of this research is CPU cache rather than image data compression, this technique demonstrates low compression overhead, which is suitable for real-time applications.

3.2.3 Huffman Encoding

This encoding introduces the Huffman dictionary to reduce the bit length of the transmitted values. This method aims to encode the more frequent values with shorter bit keys from the dictionary and transmit the keys instead of the actual values, which are then decoded into the original data by the receiver [13]. In pair with the limited range of values that can be yielded by the VGA camera (precisely from 0 to 255), Huffman encoding is a promising option for the low-bandwidth platforms limited in resources, like the one used in this project.

In [7], an elegant way to encode the sequence of data is highlighted. It proposes a combination of Delta encoding and Huffman encoding to minimize the size of the Huffman dictionary, the base assumption being that after the Delta encoding, data values

will be clustered around zero, and this skewed distribution of values will be encoded more efficiently using the Huffman dictionary. With all the additional optimizations like the least significant bits discarding and dropping the sign bits for positive values, this method reached the 65.5% compression rate.

3.2.4 Intra- and Inter-Frame Encoding

Video compression can be categorized into intra-frame encoding and inter-frame encoding. The former uses exclusively the data from the current video frame to compress it, whereas the latter utilizes the data from both the current and the previous frame (or frames).

Trait	**Intra-**	**Inter-**
Techniques	Focus on one frame	Focus on frame difference
Computational Complexity	Low	High
Compression Speed	Faster	Slower
Compression Ratio	Lower	Higher
Quality	Higher	Depends on previous frames

TABLE 3.1: Comparison of Intra- and Inter-frame encoding approaches. Adapted from [9]

Table 3.1 briefly summarizes the traits of the two formats. One can observe that, when choosing between an intra- and inter-frame encoding, the choice is about either using fewer system resources and minimizing the latency, or achieving a higher compression rate to offload the video transmitter.

The inter-frame encoders usually use computationally heavy optimization algorithms to achieve high compression rates. In [6], the authors underline the complexity of the inter-frame compression algorithms, particularly MPEG and H.264. The calculations of motion estimation alone required up to 78% of the processing time, with the estimated requirement of total computational power of 300 GIPS (Giga Instructions per second) for a 30-FPS CIF video. Interestingly, the authors also underline the fact that the growth of the video compression standards' computational complexity cannot be fully covered by software enhancement and requires specialized hardware solutions.

The intra-frame encoders, on the other hand, use robust compression techniques on each frame separately - Discrete Cosine Transformation (DCT), Discrete Wavelet Transformation (DWT), and Quantizations. The intra-frame encoding standards like MJPEG use fewer resources while compressing, but are usually not powerful enough to achieve very high compression rates. [16][17]

Chapter 4

Proposed Solution

In this chapter, we provide the complete formulation of the problem this work aims to solve and the constraints it follows, as well as a detailed explanation of the designed solution. This includes the hardware that was chosen as the base platform for modelling this solution, and the solution's Verilog module architecture and data flow.

4.1 Problem Formulation

This work aims to compress and transmit the captured YUV422 video frames through UART fast enough to maintain a semi-real-time video stream on the receiving end. The input is the multichannel video data with a resolution of up to 512x384 pixels and a frame rate of up to 10 frames per second. These limitations were applied to make the task of real-time transmission feasible. Even with these, and accounting for the 4.6 Megabaud UART transmission rate, which the top UART modules propose, the expected video data compression rate should ideally be equal to

$$\begin{aligned}\left(1 - \frac{Rate_{\text{Baud}}}{Size_{\text{Frame}} \times Size_{\text{Pixel}} \times Rate_{\text{Frame}}}\right) \times 100\% \\ = \left(1 - \frac{4.68 \times 10^6}{(512 \times 384) \times 16 \times 10}\right) \times 100\% \\ \approx 86.75\%\end{aligned}$$

Our solution should maintain cohesion while working with lots of different hardware modules simultaneously and surpass the constraints of the UART bottleneck to the maximum possible extent, while accounting for the limited FPGA resources. It naturally pursues minimizing resource usage, allowing for the solution to target most of the low-end FPGA chips.

4.2 Hardware Platform Model

The OV7670 camera module is the VGA camera used for this project. It allows an output of 640x480 pixels up to 30 frames per second. It is also configurable via the onboard SCCB bus, allowing a developer to manipulate internal registers and set the camera's color space, clock frequency divisors, the active frame's scaling, and the use of the camera's image processing functions.

The UART module chosen for this project is an FT232H USB-UART module. With the datasheet-confirmed maximal baud of 12 Megabaud transmission rate, and the 1 Kilobyte transmit buffer, this UART chip is a good choice for heavy-load embedded applications.

This project's implementation used an Xilinx Artix-7 XC7A15T FPGA and a Winbond W9825G6 32MB SDRAM module as its core components. The embedded system we use for the modelling also features an external oscillator, which provides both the FPGA and the SDRAM with a 50MHz clock. We also suppose that the number of GPIO pins we have access to is sufficient to operate the VGA camera and UART module.

It's worth mentioning that the computational capabilities of an FPGA were not considered a key factor - the proposed solution requires minimal resources and is bearable even for low-end chips with minimal tuning, as one can observe in the next chapters. The choices were dictated mostly by the accessibility of the FPGA boards, since, while still popular, such boards are not as ubiquitous as MCU boards.

4.3 Solution Overview

The developed system is fully modular and ensures the total decoupling of its modules. This section, along with the Appendices A and B, fully covers the design decisions made to implement this system.

4.3.1 High-level summary of the system

The design consists of the Verilog modules, each dedicated to a separate role in the VGA-UART transmission system, and all wrapped into the main module. This architecture is designed with the idea to simplify solving the main concerns of the task - enabling efficient frame rows buffering, minimizing the latency of the data flowing through the system, and keeping down the resources required for the implementation.

Each module was developed from scratch, without using the external IP blocks, so that each module suits the purposes of the general system best. The connection between modules, though, was organized with the use of Vivado's Synchronous BRAM FIFOs [2]. Their functionality allows the reliable bufferization and flow control in the implemented system.

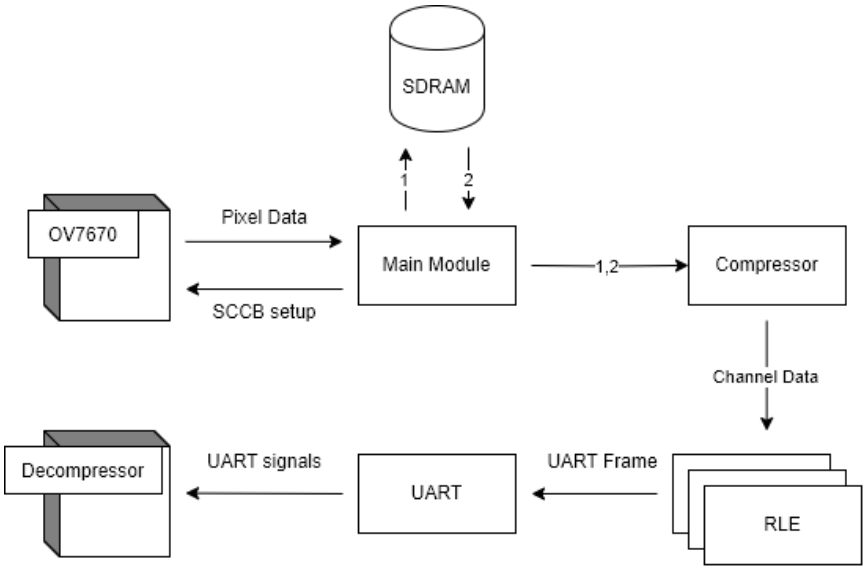

FIGURE 4.1: The data flow scheme of the solution's module architecture. 1 - Data of the Current Frame's N-th Row, 2 - Data of the Last Frame's N-th Row.

Module architecture

Figure 4.1 visualizes the data flow of the video data from the camera (OV7670) down to the receiving side (Decompressor). On the top level, the Main Module orchestrates the entire system, organizing the data flow between major components, implementing the data buffers, and balancing the load between the independent submodules. It accepts the video data input from the camera control module, collecting the data to be burst into the SDRAM module.

After writing the whole frame row into the SDRAM, the Main Module's FSM triggers the data transfer to the Compressor. The Compressor then compresses the changed row areas using multiple instances of the RLE submodules for each channel, processing each one simultaneously. Upon receiving the compressed samples, the Compressor constructs the UART packet, marks it with an ID frame, and sends it via UART, passing the packet's UART frames one by one.

UART packet structure

The solution writes the compressed video data using a predefined packet structure. The packet difference consists of six consecutive frames. One can see Figure 4.2 for the visualization.

The first frame is the Packet ID - metadata for the decompressor, which serves as the checkpoint for transmission errors. The ID is supposed to take values of 0-254, with a single UART frame with the ID value of 255 signaling the decompressor that the new frame begins. The next two frames are the most significant bits of the Y- and X-coordinates of the changed region.

Since we require various chunks of additional info on the decompressor side, we tightly pack all the miscellaneous data into the fourth frame. The first four bits are considered the don't-care bits, and are left for the system's user to define or ignore. The next two bits contain the identifier of the channel (00 being Y, 01 - U, and 10 - V), and the last two bits are each the last bit of Y- and X-coordinates, respectively.

The last two frames of our UART packet are the new value of the run, and the eight-bit run's length value.

The packet conveniently packs the necessary information, squeezing the necessary info about the change into six UART frames. The added metadata ensures that the receiver won't experience a video frame shift once a packet is lost, and can check and recover when an error happens.

The process of decompressing the necessary info consists of verifying the packet ID (and handling its discrepancies), parsing the full X- and Y-coordinates, the channel to be changed, and the new *value-amount* pair to replace the *amount* of values in the selected channel to *value*.

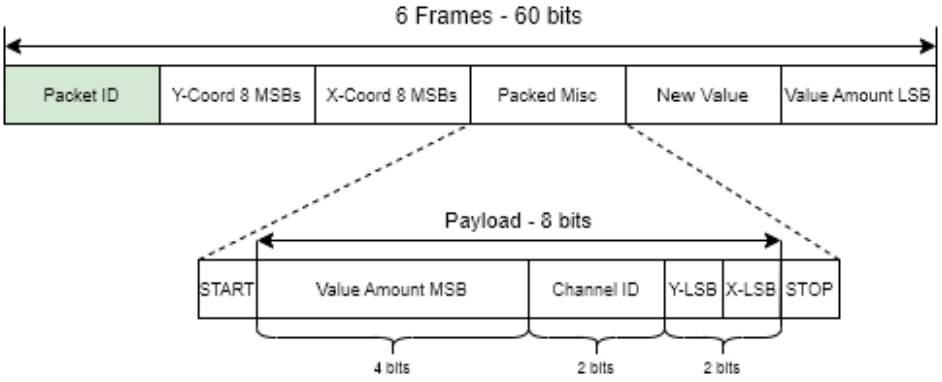

FIGURE 4.2: UART packet structure used by the compressor

4.3.2 Camera Interface

The camera interface uses the OV7670 pixel clock (PCLK) for reading incoming pixel data. The module internally resolves clock domain crossing without using Vivado's asynchronous FIFOs, simplifying the design and ensuring that the instability of PCLK does not disrupt the data flow. An external 12.5MHz clock for enabling the OV7670 is generated by the FPGA's PLL.

SCCB

OV7670 requires configuration via the SCCB bus to reach the required data rate and output video data in the required format. In its core, the SCCB protocol is similar to the I2C protocol, with minor differences in its timings and algorithm. The overview of a 2-wired, three-phase write SCCB controller implemented here can be seen in Appendix A.

4.3.3 SDRAM

Since the RAM resources of an FPGA chip are too sparse to buffer two subsequent video frames, the external SDRAM module is used for this purpose. This solution operates only on a small part of the SDRAM storage, using circular pointers to read from and write to the two-frame-sized buffer space.

Vivado, despite providing the ready IP modules for operating SDRAMs, does so only for a subset of SRAM interfaces, like DDR2 and DDR3 SDRAM [3]. As a necessary part of this work, the minimalistic yet sufficient Verilog SDR-SDRAM controller was developed. You can review the implementation details in Appendix B.

4.3.4 Compression Module

The compressor works with video frame rows. This module implements BRAM buffers for both the current and last frame's rows, sorting out the values by the three color channels separately. It passes down the sorted channel information to the RLE encoders, one for each of the channels. The encoders, in turn, conduct frame differencing pixel by pixel of the current video frame's pixel and its counterpart from the previous video frame, accounting for a preset tolerance threshold. After compressing the differences in values bigger than the tolerance threshold, this module prepares the UART frames and adds data to the UART module's buffer via large Vivado's synchronous FIFO.

This compression method allows for a drastic decrease in the required data stream size, in most cases bringing down the necessary baud rate to acceptable values.

4.3.5 UART Transmitter

The implemented UART simplex module operates the UART, regulates the ticks-per-baud rate, and transmits the data frames of the preset structure. It implements the circular buffer for frames, utilizing the frame pre-loading capabilities of FT232H and regulating the buffer overflow via the valid-ready handshake. This module essentially acts as an abstraction over the UART's buffering and transmission processes, ensuring that no frames are lost during the transmission and maintaining the stability of the transmission itself.

4.3.6 The Receiving Side

On the receiving PC side, a Python script reads from the open serial port, receiving the predefined packet structure and parsing it according to the packet content. The script extracts the coordinates of the compressed changes and the ID of the color channel, and decompresses the pixel change info, updating the next frame with the new frame's data.

It is up to the receiving side to choose the way of handling the ID errors. For example, the Python decompressor script provided as a part of this work drops the packet once the next ID frame is not matched, and tries to track the next packet with the corresponding ID. The script also uses the four MSBs of the packet's MISC frame as the four MSBs of the RLE's *value* to conform to the implementation.

Finally, once the next frame is fully updated, the receiving side flips the previous frame with the next frame, repeating the same process from the beginning.

4.4 Limitations

This solution demonstrates a few limitations that should be considered by those who consider applying the approach described here :

- The choice of the UART hardware model still strongly influences the resulting throughput, and so does the UART buffer size an embedded system can afford. Using more complex algorithms and/or increasing the buffer size drastically would require an FPGA to have DSPs and/or more RAM storage onboard.
- The RLE-with-tolerance method used here is a lossy compression, which can make this solution unsuitable for some applications.
- The system relies on a set of parameters that need to be tuned for a specific application. These include a substantial set of hardware modules' parameters, compression threshold, and active video frame's width and height borders.

Chapter 5

Experiments and Results

In this chapter, the results of the designed system's implementation are provided. We provide the FPGA resource usage statistics and the system's robustness. The system's validity was tested using the Vivado Behavior Simulation environment. The UART module was also separately tested on hardware to prove the concept of the proposed compressor-decompressor structure, verify the robust handling of the heavy data load, and roughly estimate its expected performance.

5.1 Implementation Overview

5.1.1 Post-synthesis Dataflow

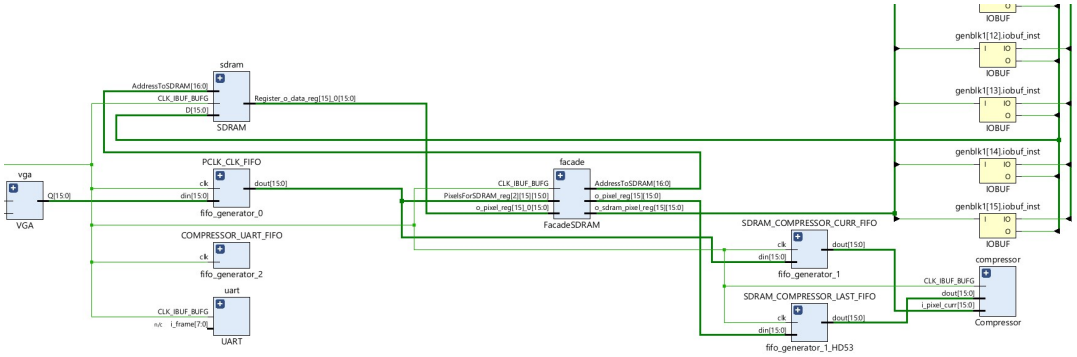

FIGURE 5.1: Result of data flow synthesis of the implemented system. Generated with Vivado

Figure 5.1 is a central fragment of the summary of post-synthesis modules' interconnection, provided by Vivado Synthesizer. The design shows the clear modular structure and the abstraction of the work with SDRAM via the FacadeSDRAM Module mentioned in Appendix B. With this illustration, the important role of Vivado's synchronous FIFOs becomes clear - they latch the data from each fundamental module, ensuring the stable work of the system and reliable information exchange procedures. The part cropped from 5.1 is the tristate buffers used to provide the bidirectional communication with the SDRAM hardware.

5.1.2 Resource Utilization

Resource	Used	Available	Utilization(%)
LUT	969	10400	9.32%
FF	1232	20800	5.92%
BRAM	14.5	25	58%
IO	53	150	35.3%
PLL	1	5	20%
DSP	0	45	0%

Table 5.1: FPGA resource utilization summary (in elements)

Table 5.1 shows the resources usage report of the solution implemented on Xilinx Artix-7 XC7A15T FPGA, provided by Vivado, post-implementation. The usage of most resources is static and remains low. One should point out the lack of use of Digital Signal Processors (DSPs) - the solution can target cheap, low-end FPGAs with little to no modifications.

The critical point is the Block RAM usage (14.5 36-Kilobit blocks). It largely depends on the tunable sizes of the FIFO buffers and especially on the size of the UART frame buffer. The estimation provided here uses the worst-case BRAM usage scenario, assuming the maximum of a 32-Kilobyte UART frame buffer and an 8-Kilobyte Compressor-UART FIFO. It's important to remember that these two grow in size with higher baud rates to grant transmission stability.

5.2 Effectiveness and robustness

Since the system was not tested as a whole on the real hardware, the compression rates and the FPS reached by the system are based on theoretical analysis and simulations. The recorded measurements are provided here to estimate the system's throughput and error handling capabilities.

The measurements were conducted in hardware, using the FT232H UART module and the Artix-7 FPGA to transmit a 10-second video transmission flow emulation, and recorded by the monitoring decompressor script. On the receiver side, a sufficient buffer size of 64MB was set to prevent the receiver from influencing the system's performance. The parameters tuned during the simulation, used to model different scenarios of the RLE encoding, are :

- **Baud rate**: 115200 baud, 921600 baud, 2.34MBaud, 4.68Mbaud
- **Buffer size**: 4KB, 8KB, 16KB, 32KB
- **Average run length**: 4 pixels, 8 pixels, 16 pixels, 32 pixels

These parameters' values were chosen based on the assumptions of the most realistic real-life example values. Also, the simulation did not emulate the frame differencing technique, always assuming the full image encoding, which assumes that the provided data is recorded for a worst-case scenario, and the real-life use case will be more powerful in terms of performance.

5.2.1 Compression Estimation

The effectiveness of the RLE method is variable, and the compression rate depends on the motions inside the video picture and, more importantly, the motion of the camera itself. In the implemented system, due to the packet skipping implemented by the decompressor, the FPS estimation becomes difficult to interpret. However, the system still demonstrates the expected FPS rate growth as the parameters grow higher.

Baud Rate	**Lowest Avg FPS**	**Highest Avg FPS**	**Total Avg FPS**
115200	0.1	0.18	0.31
921600	0.21	3.01	1.28
2343750	0.3	5.01	2.35
4687500	0.4	12.02	3.67

Table 5.2: Average FPS values reached relative to baud rate.

Table 5.2 shows the average frames per second reached for every baud rate during the 10-second measurement period. The Total Avg FPS column shows the average FPS value for a baud rate across all the tested UART frame buffer sizes.

All the highest FPS rates, regardless of the baud rate, are observed with the largest frame buffer sizes. The same is true for the lowest FPS, which is not surprising. The important observation from the table is the steady growth of the Total Average FPS value, and the increase of the difference between the lowest and highest average FPS values with the growth of the baud rate. The former serves as an indicator of the system's steady yet slow performance improvement, based clearly on baud rate increase and without accounting for other parameters the system introduces. The latter serves as an emphasis of the important role the buffer size and the luck for long run lengths play in transmission efficiency.

5.2.2 Error Recovery

We tested the error recovery with the implementation of the decompression algorithm described in 4.3.6. The introduced errors are the result of natural UART transmission throttling, caused by various factors, from insufficient buffer size to noise from the environment.

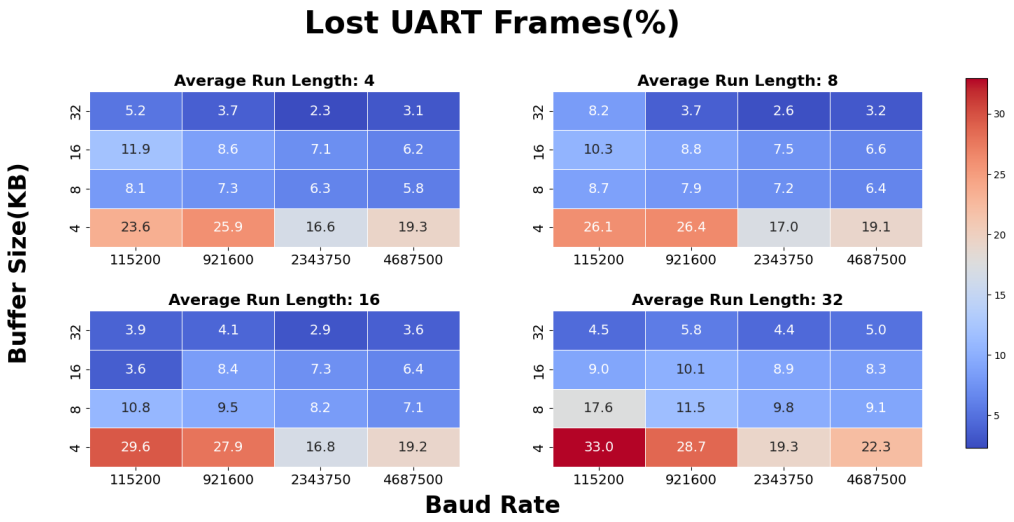

FIGURE 5.2: Fraction of lost UART frames for different run-lengths, relative to baud rates and buffer sizes

Figure 5.2 shows the percentage of the UART data frames lost when recovering from errors. In general, the expected hypothesis of a bigger buffer meaning smoother transmission is true - the lost packets fraction is decreasing with the increase of the UART frame buffer size. The longer runs (16, 32 pixels) are usually more prone to transmission errors, which is because the receiver cannot keep up with the frame redrawing rate that the higher run lengths provide, and not because of the system's specifics.

Overall, the frame loss levels depend on the reliability of the UART transmission and the system's resources it provides for the UART frames bufferization. Other than that, the system is robust in its error handling, not allowing major transmission errors to disrupt the video frame generation.

Chapter 6

Conclusions

6.1 Summary

This thesis has explored the use of FPGAs for implementing the video data transmission over a non-specialized, low-bandwidth UART transmitter. The main challenges, including the hardware controller development, system design, and algorithm creation, were all resolved by designing the proper module architecture for this task and implementing a custom algorithm tailored to suit the predefined constraints of the work.

The result is an embedded system's design that demonstrates the adaptive compression and video transmission capabilities, maintains reasonable resource usage limits, and provides means to ensure the stability of video rendering on the receiving side. We conducted a simulation of transmitting video data of large resolution using an unspecialized, low-bandwidth serial transmitter, contributing to the existing solutions in the sphere of serial video transferring in embedded systems. We also provided useful statistics and estimations, using which a user can implement the solution with the known expected error rate and FPS rate on their low-end FPGA chips.

6.2 Future work

Further work on this system might include the increase of compression efficiency, general system optimization related to the BRAM usage and SDRAM latency, and exploring other measures to make the PCB placement of such a design viable. Choosing even cheaper FPGA chips, SDRAM modules, and UART transmitters should all be possible without major changes to the design presented here.

Appendix A

Appendix A

This Appendix contains a short description of the Verilog SCCB module developed for the OV7670 camera in this project.

A.0.1 Functionality description

Functionality of the SCCB protocol is similar to the I2C protocol, with a few exceptions, like not requiring the ACK signal. The version implemented in the project ignores the ACK signal (notably named Don't-Care bit) from the slave device, submitting to the SCCB requirements. Therefore, this controller, although similar to a basic Verilog I2C module, cannot be used as the I2C module without modifications.

The SCCB 3-phase data transmission process comes in three stages :

- Sending the Device Address - used the 0x42 for OV7670 in write mode
- Sending the camera's Register Address - e.g. the camera's PLL register
- Sending the Register's Value

Since the controller assumes that the slave device responds with no delay, the pause period between the writes is longer than the required timing. Despite that, the controller respects the pauses between writes and during the initial signal stabilization.

A.0.2 The SCCB bus usage

In the VGA-UART project, until the SCCB bus sets the OV7670, the whole system stalls and does not receive data from the VGA module. The configuration dictionary is hardcoded into the VGA module, configuring the Clock Divisors and Color Space to match the required specifications.

Appendix B

Appendix B

This Appendix summarizes the minimalistic Verilog SDRAM controller implemented for this project.

The controller supports the burst reads/writes of size up to 8 words a burst and the regular data refresh operations. The controller operates without the active memory model, working directly with the SDRAM hardware.

B.0.1 Functionality description

Below, in B.1, is the simplified state machine of the controller. We do not explore the respect of timings and NOP-pauses here, which the controller naturally handles alongside the setup, refresh, and read-write processes.

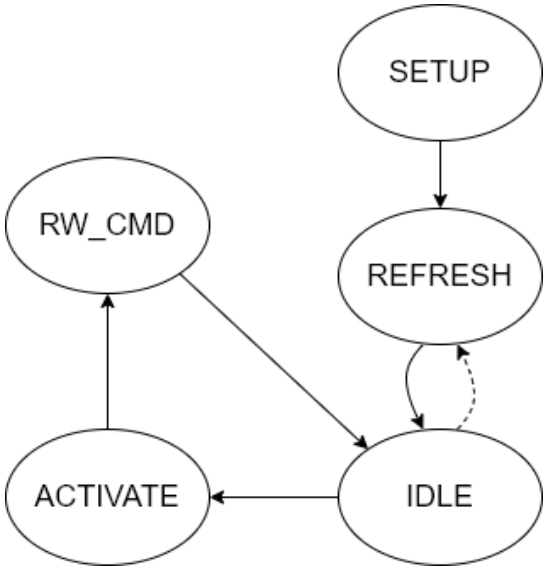

FIGURE B.1: The SDRAM controller's simplified State Machine

After the reset of the system, the SDRAM controller performs the SETUP, programming the registers of the SDRAM module to set burst mode (read/read and write), burst length (from 1 to 8 included, or a full page), addressing method (sequential or interleaved) and the data read latency (2 or 3 clock cycles).

After that, the initial refresh period is entered, and the controller transitions to the IDLE state, expecting a command from the user. After receiving one, it ACTIVATE-s the row in the required bank and transitions to the RW_CMD state, executing the read or write burst command with the auto-precharge after these, and transferring to the IDLE state afterwards. The controller also periodically refreshes the SDRAM data according to the internal clock tick counter.

B.0.2 Limitations

This implementation has a few caveats that should be kept in mind:

- The controller issues an auto-precharge after each read/write operation, regardless of the active row. While not being a bottleneck in the case of the VGA-UART project, it is inefficient and should not be considered a standard practice.
- The use of auto-precharge, despite simplifying the design, also disables the full-page burst mode, which can even further reduce the performance of certain applications.
- The user of this module is responsible for the correctness of the address data, the stability of the burst sequence, and the correct read-write signal edges, which might be inconvenient to use, even with dedicated routines. In the VGA-UART project, a separate SDRAM Facade module was developed for abstracting away the details of working with the SDRAM controller.

Bibliography

[1] Van Hunter Adams. *Universal Asynchronous Receiver Transmitter (UART)*. Accessed: 2025-04-16. 2021. URL: http://vanhunteradams.com/Protocols/UART/UART.html.

[2] AMD. *FIFO Generator*. Accessed: 2025-04-20. URL: https://www.amd.com/en/products/adaptive-socs-and-fpgas/intellectual-property/fifo_generator.html.

[3] AMD. *Memory Interface*. Accessed: 2025-04-16. URL: https://www.amd.com/en/products/adaptive-socs-and-fpgas/intellectual-property/mig.html.

[4] ArduCam. *Arduino Camera: SPI Camera Module from Arducam*. Accessed: 2025-04-20. URL: https://blog.arducam.com/spi-arduino-camera.

[5] Evan Caton. *YUV: Colour and Compression*. Accessed: 2025-03-14. 2021. URL: https://deeprender.ai/blog/yuv-colour-and-compression.

[6] Shao-Yi Chien et al. "Hardware architecture design of video compression for multimedia communication systems". In: *Communications Magazine, IEEE* 43 (Sept. 2005), pp. 122 –131. DOI: 10.1109/MCOM.2005.1497562.

[7] Aaron Cuevas Lopez et al. "Low-Power Lossless Data Compression for Wireless Brain Electrophysiology". In: *Sensors* 22 (May 2022), p. 3676. DOI: 10.3390/s22103676.

[8] FLIR. *Video over SPI (VoSPI) Implementaion Specification*. Accessed : 2025-04-20. Jan. 2013. URL: https://media.digikey.com/pdf/Data%20Sheets/FLIR%20PDFs/VoSPI_Implementation_Spec_Rev007_Sep9_2013.pdf.

[9] Vijaysing Patil. *Understanding video inter-frame compression*. Accessed: 2025-04-26. Jan. 2025. URL: https://www.fastpix.io/blog/understanding-video-inter-frame-compression.

[10] Gennady Pekhimenko et al. "Base-delta-immediate compression: Practical data compression for on-chip caches". In: Sept. 2012, pp. 377–388. DOI: 10.1145/2370816.2370870.

[11] Xutan Peng et al. *Selective Run-Length Encoding*. 2023. arXiv: 2312.17024 [cs.DS]. URL: https://arxiv.org/abs/2312.17024.

[12] Michal Podpora, Grzegorz Korba´s, and Aleksandra Kawala-Sterniuk. "YUV vs RGB – Choosing a Color Space for Human-Machine Interaction". In: *Annals of Computer Science and Information Systems* Vol. 3 (Sept. 2014). DOI: 10.15439/2014F206.

[13] Gautam Ramachandra and S. Murali. "An Optimized Huffmans Coding by the method of Grouping". In: (July 2016). DOI: 10.48550/arXiv.1607.08433.

[14] Steven W. Smith. "Data Compression". In: *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997. Chap. 27, pp. 486–487. URL: www.DSPguide.com.

[15] Karl Soule. *Color Subsampling, or What is 4:4:4 or 4:2:2??* Accessed: 2025-03-18. 2010. URL: https://www.provideocoalition.com/color_subsampling_or_what_is_444_or_422.

[16] Roman Strukov and Vassilis Athitsos. "Evaluation of Video Compression Methods for Network Transmission on Diverse Data: A Case Study". In: *Proceedings of the 16th International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA '23. Corfu, Greece: Association for Computing Machinery, 2023, 300–305. ISBN: 9798400700699. DOI: 10.1145/3594806.3594863. URL: https://doi.org/10.1145/3594806.3594863.

[17] Bernatin Thomas, Sundari Govindarajan, and Dr. V.J.K.Kishor Sonti. "FPGA implementation of modified intra-frame prediction for h.264 video codec". In: *International Journal of Pure and Applied Mathematics* 118 (Jan. 2018), pp. 135–150.

[18] Hayder Waleed, Mohammed Abdullah, and Hassan Awheed Jeiad. "Lossless JPEG image compression based on FPGA". In: *Solid State Technology* 63 (Mar. 2021), p. 3395.