ECE532: Digital Systems Design

Final Project Group Report

April 7, 2025

LumiSync – FPGA-Powered Ambient Lighting for Monitors

Group 30:

Junchen Zhu

Nuo Si

Lihao Xue

Ruogu Xu

1. Overview

1.1 Background

The ambient lighting system for smart screens originates from Philips' early Ambilight technology introduced in televisions, which enhances the immersive viewing experience by projecting light that matches the image behind the screen. In recent years, this concept has been applied to standalone devices and lighting systems to create dynamic backlighting effects around screens such as TVs and monitors. Various implementation schemes have emerged on the market, represented by products such as the **Philips Hue Play HDMI Sync Box**, the **Govee Envisual series**, and the **Corsair iCUE Smart Lighting Strip.**

Philips Hue Play HDMI Sync Box [1] is a hardware device that synchronizes screen video content with Hue smart lights. It provides four HDMI inputs, capable of capturing video signals and controlling up to 10 Hue color bulbs via the Hue Bridge to achieve instant color-matching backlight effects. Thanks to its dedicated hardware architecture and the Hue Entertainment Zone wireless protocol, the Hue Sync Box achieves near-perceptual delay-free synchronization performance (users can hardly perceive desynchronization between light and screen when watching movies or playing games), and is considered one of the best synchronization solutions currently. It is mainly targeted at home theater and console gaming users, emphasizing a plug-and-play high-definition audiovisual lighting synchronization experience. However, the product is expensive (for over $500), and must be used in conjunction with Hue series lights and bridge.

The Govee Envisual immersive backlight kit [2] adopts a completely different approach from Hue. It captures the screen's colors through a camera mounted on the screen edge, then drives an LED strip to change accordingly. The advantage of this architecture is compatibility with any content (including built-in smart TV apps) without interfering with the video signal, offering lower installation cost and ease of use for users looking for a low-barrier ambient light experience. The Govee system uses algorithms to map the colors captured at the screen edge to LED zones, creating an effect similar to Ambilight. However, due to the inherent delay in camera capture and image processing, the light response shows a delay of "tenths of a second" in scenes with fast image switching. In certain fast-action movie scenes, such delay and color inaccuracies can be noticeably out of sync, and for fast-changing scenes like gaming, this delay significantly deteriorates the experience. Although Govee provides game modes with faster response to mitigate the delay, it still falls far behind Hue’s hardware pass-through scheme in terms of instantaneous synchronization.

Corsair's iCUE Smart Lighting Strip [3] represents a class of PC-platform software-hardware combined solutions. This solution targets desktop computer and monitor environments: by running iCUE software on a PC to capture screen content, and using the Lighting Node Pro to control connected LED strips, screen-synchronized backlighting is achieved. The advantage of this software-hardware combined architecture lies in its high flexibility—users can customize lighting effects through software and link with other PC peripherals, even supporting synchronization across multi-display environments. Its primary use case is immersive desktop lighting for PC gamers, as well as lighting shows inside/behind the computer. However, such schemes rely on host computing resources for real-time video signal processing, potentially imposing additional burden on the CPU/GPU during prolonged operation, and real-time screen capture also raises certain privacy concerns. Compared to standalone hardware schemes, PC software schemes trade off real-time performance and system stability, but are easier to deploy and expand, with hardware costs mainly coming from LEDs and controllers.

In summary, current ambient lighting systems each have their strengths: hardware pass-through solutions like Hue Sync Box offer the lowest latency and best effects, but are costly and ecosystem-bound; camera-based visual capture solutions like Govee are easier to install and have good source compatibility, but respond slower; software-driven solutions like Corsair iCUE are flexible and powerful, but rely on computing resources and are PC-only. This diversity provides a reference and comparison foundation for the LumiSync project.

1.2 Motivation

Although multiple ambient lighting solutions exist on the market, limitations and areas for improvement still exist, motivating our development of an FPGA-based LumiSync smart ambient lighting system.

First, high-performance existing solutions are expensive. Taking the Philips Hue Sync Box as an example, its sync box alone is costly, and combined with Hue lights and bridge, the overall system investment is significant. This poses a high barrier for ordinary users and developers. In contrast, we aim to explore the use of general-purpose programmable hardware (FPGA) and inexpensive addressable LED strips (such as WS2812B) to implement similar functionality, thereby reducing hardware costs and eliminating dependence on vendor ecosystems.

Second, current solutions each have functional limitations: the Hue Sync Box cannot be used outside of its Hue ecosystem; Govee's camera solution requires complex calibration and suffers from accuracy degradation in darkness or under ambient light interference; Corsair iCUE is limited to PC platforms and depends on system performance. We are motivated to design an independent and self-contained system that can be plug-and-play for any HDMI output device

(game consoles, TV boxes, PCs, etc.), ensuring ultra-low-latency synchronization while offering better user openness and customizability.

Third, real-time performance is the core of immersive lighting experiences. Although commercial products like Hue have achieved very low latency, as developers, we wish to verify whether FPGA hardware acceleration can achieve or even further optimize end-to-end latency, ensuring that lighting changes are strictly synchronized with screen content.

Moreover, from both academic and engineering practice perspectives, the acquisition and processing of high-speed HDMI video streams, hardware implementation of image color extraction algorithms, and the integration of FPGA with peripherals (LEDs, Bluetooth) are all highly challenging topics. These challenges strongly attract our team and drive us to develop the LumiSync system as a course design project. During development, we also aim to gain in-depth understanding of software-hardware co-design, such as: how to implement all related functions in a pure hardware architecture, how to handle coordination between high-speed data streams and real-time control; whether FPGA hardware acceleration can achieve or even further optimize end-to-end latency, ensuring strict synchronization between lighting changes and screen content. These motivations collectively form the rationale for the LumiSync project.

1.3 Goals

The overall objective of the LumiSync project is to implement a low-latency, high real-time performance smart ambient lighting system under a pure FPGA hardware architecture. Specific goals are as follows:

1. HDMI Signal Pass-through: Establish HDMI input/output channels through the FPGA, store video input into a buffer, and use MicroBlaze to switch output signal sources: achieve control over displaying color bars / original input video / inverted input video, etc. Ensure uninterrupted video signal transmission from the source to the display, achieving user-transparent “pass-through” connections.
2. Real-time Screen Edge Color Extraction: Hardware modules divide the edge regions of the input video frames and calculate the average RGB values for each region.
3. Ultra-low Latency Lighting Synchronization: Utilize the FPGA's parallel processing capabilities, employ multiple FIFOs to immediately use extracted color data to drive the LED strip, with the system’s target end-to-end latency controlled within 20 ms.
4. WS2812B LED Strip Control: Achieve direct driving of WS2812B-type RGB LED strips, with precise control signals and high refresh rate.

5. Remote Control and Interconnectivity: Through a Bluetooth module and custom mobile app, implement a complete interactive system to remotely adjust display settings, LED lighting modes, colors, etc.

1.4 Block Diagram

The Ambilight system is built on the Xilinx Artix-7 (Digilent Nexys Video) platform, utilizing the AXI bus to interconnect various modules. The system first captures the HDMI video signal through the DVI2RGB IP core, converting it into an RGB888 format. A video frame buffer temporarily stores the frames. The MicroBlaze processor is used to process the video signal, such as setting the resolution and performing color inversion.

The video capture module extracts edge region data, filters pixel information, and computes the average RGB values for the corresponding regions. The processed colors are then sent to a light_strip_output IP, which generates reset codes for the RGB LED strip driver, controlling the RGB LEDs to create ambient lighting that responds to video content.

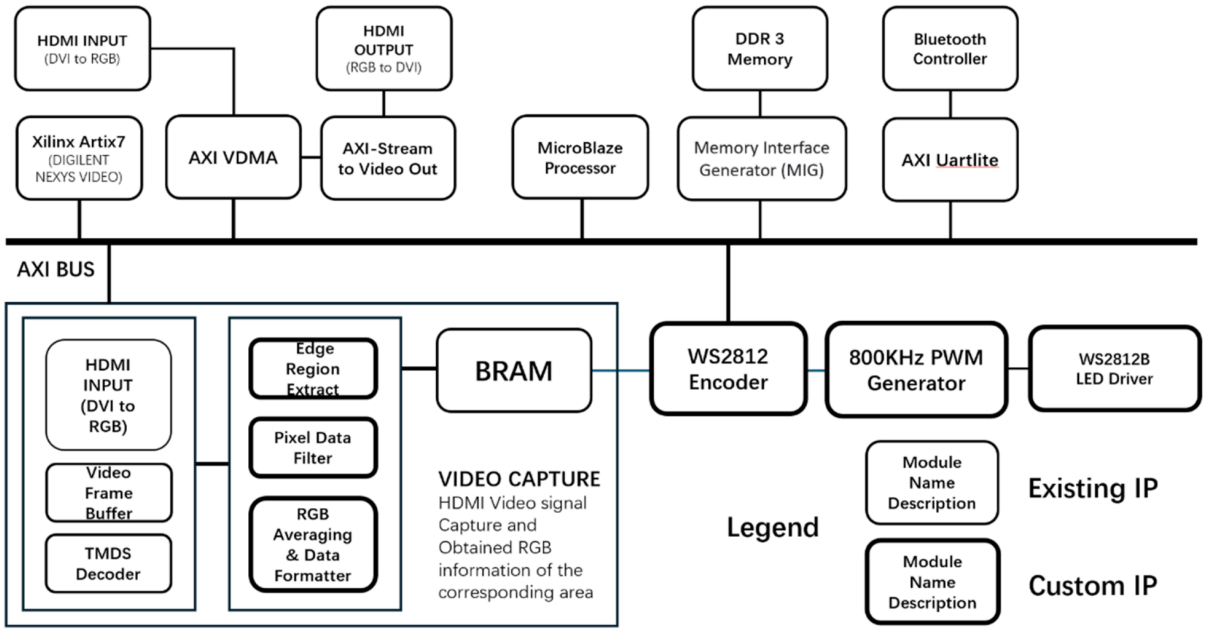

Figure 1: LumiSync Project Block Diagram

1.5 Project Advantages

Compared with existing solutions such as Philips Hue, Govee, and Corsair, the core differences of LumiSync are reflected in:

1. Pure hardware architecture completely based on FPGA, avoiding software layer interference, and achieving more precise latency control.
2. Latency: target end-to-end latency <20ms, performance close to Hue Sync Box, significantly better than Govee (camera delay 0.3~0.5s) and Corsair.
3. Real-time performance and complexity: pipeline design ensures extremely low frame-level processing latency, achieving high performance and determinism.
4. Control method: dominated by local hardware + supplemented by Bluetooth control, runs independently without PC or ecosystem accessories, superior to architectures relying on apps (Govee) or bridge (Hue), and runs fully locally without privacy concerns.
5. Scalability: architecture reserved with multi-channel interfaces and resource space, can be extended to multi-screen input, multi-strip output, theoretically capable of multi-node linkage similar to Hue.

1.6 Technical Specifications

LED Strip Update Latency Analysis

1. Our project uses video input at 60 FPS, with a total of 98 LEDs, The density is 0.6 LEDs per centimeter.
2. The theoretical LED strip update latency is 19.7 ms, which includes: 16.6 ms for displaying a single video frame and 2.94 ms for LED strip communication time.

Video Partitioning and LED Update Frequency

1. The screen is divided into 31 × 18 = 558 video regions.
2. The video resolution is 1024 × 768, and the physical display size is 56 cm × 32 cm (standard 24-inch monitor).
3. LED strip data is updated once every 5 video frames, resulting in a theoretical LED refresh rate of 12 FPS.

1.7 Brief Descriptions of the IP

The core IP modules used in the project include:

1. Vivado HDMI RX/TX IP: responsible for receiving and transmitting HDMI video streams, handling TMDS protocol, converting to/from RGB, clock recovery, etc.
2. Color Analysis Module: self-developed IP, dynamically extracts edge RGB average values based on region division and stores them into FIFO using FPGA parallel logic.
3. WS2812B Control Module: self-developed IP, filters RGB data and encodes it into serial pulse waveforms required by the LED protocol to achieve high-frequency driving.

4. UART Communication Module: AXI UART Lite IP provided by Xilinx, connected to a Bluetooth module, supports serial port control. Also implements a complete control system.
5. MicroBlaze Soft Processor: switches and controls data channels, including resolution adjustment, the output of color bars / original input video / inverted input video, etc.
6. Clock Management Module: generates multiple dynamic clock frequencies required for HDMI and LED driving, ensuring stable system operation.

2. Outcome

Overall, the LumiSync project successfully processes real-time video input from an HDMI source, performs RGB edge color analysis using custom FPGA logic, and drives a WS2812B LED strip with ambient lighting effects that reflect the screen content. Additionally, we implemented a mobile control interface through Bluetooth, enabling users to switch modes and control LED behavior remotely.

2.1 Results

We were able to meet all of our core project goals and demonstrate a working real-time ambient lighting system using the FPGA. The HDMI input and output worked reliably at 1024×768 resolution and 60 FPS, with no visible artifacts and no perceptible delay between the input source and the display. The system passed video content through the FPGA while also extracting edge color information for the LED control. This setup is shown in Figure 2.1, where the laptop feeds video into the FPGA, and the external monitor displays the passthrough output in perfect sync.

To drive the ambient lighting, we divided the screen into 558 edge regions and used a custom IP to calculate the average RGB color in each one. This data was then passed to another IP block that generated precise PWM signals to control 98 WS2812B LEDs. We updated the LEDs once every 5 video frames, giving us an update rate of 12 FPS, which turned out to be smooth enough to match the video transitions in real-time.

We measured the end-to-end system latency — from HDMI input to LED update — to be around 19.7 ms, which met our expectations for real-time behavior. Visually, the LED colors changed in sync with fast scene changes in the video, and there was no noticeable lag.

Figure 2: RGB Color Bar Demonstration

We also implemented Bluetooth control using the BT2 Pmod and MicroBlaze processor. The user could send commands through a terminal app on their phone or computer to switch LED modes, select fixed colors, or turn the lights on/off. This worked reliably throughout our testing and added flexibility to the system.

One feature we initially planned to include was SD card playback, which would allow the system to display stored images or video without an external HDMI input source. We were able to read data from the SD card into memory using MicroBlaze, but the HDMI output was unstable and displayed noise. We believe this was due to formatting or synchronization issues in the frame buffer. Since the core system was already functional and met our complexity requirements, we decided to leave this part out of the final demo.

Overall, we were happy with the results. The system was stable in hardware, the lighting looked great and reacted in real-time, and we successfully integrated multiple hardware and software modules into one working design.

2.2 Future Work and Improvements

One of the biggest features we wanted to include—but didn't fully finish—was the SD card video playback. We made good progress reading data into memory using MicroBlaze, and we could see that data was being accessed and written. However, the HDMI output from that path was mostly visual noise. We believe the issue was related to frame formatting or timing alignment between the memory buffer and the video pipeline. With more time, we think this could be completed, allowing the system to run independently of an external HDMI source.

Our Bluetooth interface currently works through a terminal-based app, which is functional but not very user-friendly. In future versions, we would like to build a proper mobile app with a graphical UI. Features like color pickers, sliders, and effect presets would make the system feel much more polished and accessible. We also thought about scalability. The current setup works well for a standard monitor with a single LED strip. But with more output ports and configuration options, the system could support ultra-wide monitors, multiple LED strips, or even room-scale ambient lighting. This would require careful planning in terms of synchronization, power, and control logic, but the architecture we've built could be extended to support it.

3. Project Schedule

Our project schedule and milestones are outlined in Table 1. In addition, the project proposal's milestones will be compared to the actual progress achieved during project execution.

Milestone	Objective	Deliverables
1	HDMI Input/Output.	Set up the FPGA environment and test HDMI input/output passthrough.
2	RGB Analysis	Implement basic RGB analysis for screen edges.
3	WS2812B LED Strip Test	Drive WS2812B LED strip with static color-based on-screen content.
4	Optimization for LED Strip + Mid-project Demo Preparation.	Add real-time dynamic effects and optimize for low latency. Prepare for the mid-project demo.
5	Bluetooth Implementation	Integrate IoT functionality for mobile control.
6	Integration Test + Final Demo Preparation	Test all features and prepare for the final demo.

Table 1: Initial Milestones

The original table presents a clear breakdown of the project milestones—from setting up the FPGA environment and testing basic functionalities to integrating complex modules such as HDMI, Bluetooth, and LED control. It is evident that the initial schedule was designed with a balanced approach, and the goal is to validate each component separately before moving into full system integration. This phased approach not only allowed for focused testing of individual modules but also provided a structured framework to identify and address issues as they arose quickly.

The initial milestones were similar to the planned milestones. However, as we learned more and tried more with different components, we began to deviate from the specifics of the plan. Our team always tried to address the expected challenges, but there were always unforeseen challenges as the project unfolded, all of which changed our timeline and changed our focus at various stages. But we still use it as a reference for weekly project progress. Our actual milestones are as follows.

Milestone	Objective	Deliverables	Challenges	Next Steps
1 (Week 5)	• System architecture definition • FPGA environment setup •WS2812B protocol research	• Completed technical roadmap • Vivado environment ready • LED protocol documentation	None explicitly stated	• Refine HDMI I/O pipeline • Develop RGB extraction module • Prepare hardware components
2 (Week 6)	• HDMI passthrough implementation • Bluetooth module integration	• Functional HDMI with low latency • UART Bluetooth communication established	• iOS device incompatibility • PMOD IP limited to Nexys Video	• Implement SD card interface • Begin LED driver development
3 (Week 8)	• WS2812B LED control • SD card image processing	• Working Neopixel controller (VHDL) • BMP image reading via FatFs	• HDMI output incomplete • Vivado SDK version issues • LED chain	• Continue SD-to-HDMI integration • Optimize LED timing control

		• AXI VDMA configured	timing uncertainties	
4 (Week 9)	• Bluetooth system migration • LED-HDMI integration	• Bluetooth moved to Vitis • Initial LED-HDMI testing complete	• Hardware failures (Nexys Video) • LED clock signal issues (800kHz requirement) • SD card access problems	• Finalize Bluetooth-HDMI integration • Resolve LED timing issues • Fix SD card functionality
5 (Week 10)	• Full system integration • RGB data correction	• GRB format fixed • Bluetooth-HDMI -LED operational	• Pmod port conflicts • LED strip length mismatch • HDMI output noise	• Test alternative Pmod ports • Acquire longer LED strip • Continue SD card debugging
6 (Week 11)	• Final system validation	• New LED strip installed (24" monitor) • All features functional	No major challenges reported	• Record demonstration video • Prepare final presentation materials

Table 2: Actual Milestones

The projected milestones started with FPGA environment setup, RGB analysis, and basic LED control. During the early stages, as outlined in the first milestone, the team concentrated on building a solid foundation by thoroughly setting up the environment and performing preliminary hardware tests. This initial phase was critical since any shortcomings at this stage could cascade into later phases. For example, the challenges faced with HDMI I/O and LED protocol verification underscored the importance of rigorous testing and laid the groundwork for further refinements.

In the subsequent module development phase, the emphasis shifted towards developing and verifying independent submodules, such as the RGB extraction module, WS2812B LED driver, and Bluetooth communication interface. As shown in the table, the project evolved as the team learned more and worked on various components. Similar to the report, the team encountered

many unforeseen issues, such as compatibility issues (iOS devices, PMOD IP), hardware failures (HDMI output, LED timing), and resource allocation (Pmod port conflicts). These required the team to flexibly adjust the plan, which was the largest cause of milestone deviations.

Major timeline changes, for example, the integration of the Bluetooth module into the system was delayed due to compatibility issues. These unexpected issues indicated that even though the team expected to resolve the challenges in week 10, our final milestones required adjustments to system verification and debugging. This iterative process was crucial, as it allowed the team to refine the interface standards and timing requirements across modules, and ensure that they would integrate seamlessly during the system integration phase.

Regarding the next steps, as the team expected, the plan might need to be modified after testing and debugging. In the case of our project, resolving hardware conflicts (LED timing and SD card) changed our testing phase, which delayed the final demo preparation.

Finally, the performance optimization phase focused on enhancing the overall system by reducing latency and improving user experience—especially through refining the GUI. The table's final milestones indicate that, despite time constraints, the team was able to achieve significant performance gains. Lessons learned from this phase, such as the need for automated debugging and better error reporting systems, are invaluable for future projects and iterations.

This section reflects how real challenges led to the adjustment of milestones. The team anticipated similar shifts in their project as actual testing and integration presented unforeseen difficulties, especially with hardware and component integration. As with regular design projects, our project schedule required flexibility. The integration steps (HDMI, Bluetooth, LED control) were laid out at the outset, and although unforeseen issues were expected to change deadlines and goals, the team benefited from a clear plan of expected design steps and was able to complete most of the project’s goals within the time frame.

4. Description of the Blocks

4.1 Digilent IPs:

We have used three Digilent IPs for our project. The Dynamic Clock Generator is used to generate pixel clocks for video in different resolutions. The DVI to RGB Video Decoder takes TMDS and converts it to RGB for processing. The RGB to DVI Video Encoder finally takes processed/unprocessed RGB data and converts it to TMDS for HDMI output. The IPs are found on Github from Digilent’s repository [4].

4.1.1 Dynamic Clock Generator

The Dynamic Clock Generator is a specialized FPGA logic block that provides a reconfigurable clock source at run‑time, without needing to rebuild or re‑synthesize the entire FPGA design. It is particularly common in video and display applications such as HDMI/DVI output, where you often need to generate a pixel clock that matches the desired video resolution and refresh rate (e.g. ~148.5 MHz for 1080p@60 Hz), and a 5x serial clock for the high-speed TMDS serial interface.

The IP is designed with a standard AXI-Lite interface, allowing for register writes from an on-chip processor like the Microblaze we have in our design. Digilent has provided its driver in C and clocks are automatically generated when the user selects a resolution. This flexibility simplifies multi‑resolution video designs and enables more adaptive systems that can switch display modes or frame rates as required.

4.1.2 DVI to RGB Video Decoder

The DVI-to-RGB (Sink) IP core takes in transition-minimized differential signaling (TMDS) data and clock channels from a DVI source and converts them into a parallel 24-bit RGB output along with the necessary horizontal and vertical sync signals and a pixel clock. Internally, it recovers the character-rate clock from the TMDS clock channel using Xilinx clocking primitives, deserializes the three TMDS data channels, and applies DVI-compliant decoding to reconstruct valid pixel data and blanking intervals. The IP also includes logic to align data channels, remove skew, and maintain DC balance, ensuring robust, reliable video output.

In the design, we have constrained the TMDS clock to 80 MHz (even though 120 MHz is the maximum supported for the FPGA part) to achieve simpler timing closure. By limiting the maximum clock frequency, the IP can more reliably meet internal timing requirements, particularly in FPGAs where speed grade or layout might otherwise cause violations at higher clock rates. Internally, we are targeting a resolution of 1024 x 768 with a clock frequency of 65MHz, this both ensures timing requirements and full-screen display on a standard 24-inch monitor.

Additionally, the IP offers optional Display Data Channel (DDC) functionality and an internal EDID. This allows the connected video source to read out supported resolutions and choose an operating mode that keeps the pixel clock at or below your desired limit (in this case, 80 MHz). Once locked and aligned, the IP produces a stable 24-bit RGB output, along with synchronization signals and a “valid” indicator for downstream use. In our design, the RGB output is sent to Video In to AXI4-Stream IP that transforms the data into AXI4-Stream format.

4.1.3 RGB to DVI Video Encoder

The RGB-to-DVI (Source) IP core transforms a 24-bit parallel RGB video signal from AXI4-Stream to Video Out IP into transition-minimized differential signaling (TMDS), which is then driven out through three data channels and one clock channel according to the DVI 1.0 specification. Internally, the IP first encodes the 8-bit color components (along with horizontal/vertical sync and blanking indicators) into 10-bit TMDS symbols designed to balance the bit transitions. After encoding, dedicated Xilinx serializer primitives output the data at 10 times the pixel clock rate, while the clock channel is sent at the base pixel frequency.

This IP can either generate the 5×-faster "serial clock" internally via a PLL or accept it from an external source that already synthesizes the required clock signals. In our design the serial clock is from the Dynamic Clock Generator IP. The key is to ensure that the pixel clock and the serial clock remain phase-aligned, typically through dedicated clock routing resources (BUFIO/BUFR) so that the high-speed serializers receive clean signals.

4.2 Xilinx IPs

4.2.1 Microblaze IP & Associated Blocks

We instantiated the MicroBlaze soft processor as the main control processor to manage UART communication for Bluetooth control and general system management. It interfaces with various peripherals through an AXI interconnect and runs a lightweight firmware developed in the Xilinx Vitis. [5]

To assist the processor, we included an AXI Interrupt Controller (microblaze_0_axi_intc) from the Vivado IP library. This provided infrastructure for handling interrupts, although in our final version, we primarily relied on polling methods.

We also used the MicroBlaze Debug Module (MDM v3.2) for early-stage debugging. This allowed us to monitor and debug the processor registers and memory operations during development.

4.2.2 AXI GPIO

We used three AXI GPIO IP blocks to provide general-purpose input/output interfaces controlled by the MicroBlaze processor:

- The first AXI GPIO (axi_gpio_LED_color) allowed us to manually set specific color values for testing the WS2812B LED strip. It was very helpful in the early stages to quickly verify that the LED hardware and PWM driver were functioning properly.

- The second AXI GPIO (axi_gpio_strip) was configured to manage on/off for the LED driver IP.
- The third AXI GPIO (axi_gpio_video) interfaced with the video pipeline and allowed us to read back video-processing-related status signals. This was particularly useful when debugging or verifying video pipeline operation and system synchronization.

All three GPIO blocks used the standard AXI Lite interface, making them straightforward to access from our MicroBlaze firmware.

4.2.3 AXI Video Direct Memory Access

AXI Video Direct Memory Access (AXI VDMA) is a high-performance DMA module designed for video data provided by Xilinx. It provides direct data movement between DDR memory and the AXI4-Stream video interface. In modern video systems, frame buffers are often used to cope with different frame rates or different resolution conversion requirements, and VDMA provides an efficient implementation of this cache mechanism [8].

AXI VDMA contains two independent channels. The MM2S channel (Memory Mapped to Stream) is responsible for reading image data from DDR memory and outputting it to the video processing pipeline or display module through the AXI4-Stream interface. The S2MM channel (Stream to Memory Mapped) is responsible for receiving AXI4-Stream format data from the video input module and writing it to the DDR memory.

The IP has built-in AXI DataMover for actual data movement, supports burst transmission and automatic address management, and has a protection mechanism for address boundaries such as 4KB boundaries, which can avoid errors caused by data crossing boundaries. AXI VDMA supports up to 32 frame buffers and can realize automatic loop mode. This means that it is possible to read and write video frames in a continuous loop, providing an elastic cache in video processing. In addition, AXI VDMA allows the video interface and memory interface to use clocks of different frequencies. Asynchronous FIFO buffering is used to achieve smooth data transmission across clock domains [9].

This project uses AXI VDMA to cache the video data collected from HDMI input into DDR memory in real-time and is also used to read data from memory to display output in real-time. By configuring the frame size, cache address, and interrupt mechanism of MM2S and S2MM channels, AXI VDMA realizes the decoupling of video input and output. This effectively increases the continuity and stability of video frame transmission.

4.2.4 AXI Uartlite

AXI Uartlite is a lightweight serial communication IP core officially provided by Xilinx, which implements the standard UART communication protocol based on the AXI4-Lite interface. This

module is widely used in the design of simple control communication, status monitoring and debugging interfaces.

The AXI Uartlite module consists of the following parts. First, the interface logic of AXI4-Lite can achieve seamless connection with processors such as MicroBlaze, and simplify the register read and write control process [10].

Secondly, the UART core logic includes: transmit and receive FIFO, baud rate generator (BRG), control register and status register, and interrupt control logic. The transmit and receive have independent FIFOs with a depth of 16 characters to prevent data overflow. The baud rate generator flexibly adjusts the serial communication baud rate by setting different register values. The control register and status register are used to configure the number of data bits (5-8 bits), parity check mode, and real-time query of FIFO status. The interrupt control logic provides interrupt functions triggered by data transmission completion and data reception events to optimize the real-time response of the processor [11].

In this project, AXI Uartlite is specifically used to drive an external Bluetooth communication module. Through the standard UART interface provided by Uartlite, the FPGA can send system status information, control commands, or processed data to Bluetooth terminal devices such as mobile phones in real time wirelessly to achieve remote data monitoring, parameter adjustment or control functions. Besides, AXI Uartlite is equipped with a MicroBlaze processor so that it can send key status information such as frame rate and cache status to the PC terminal.

Compared with the traditional serial cable connection method, this design uses Bluetooth wireless transmission to not only improve the flexibility and convenience of data interaction but also effectively expand the system's adaptability in various wireless application scenarios, significantly enhancing the overall interactivity and user experience of the project.

4.2.5 AXI Timer

We included the AXI Timer (axi_timer_0) primarily to measure and verify critical timing in our system, as accurate timing was essential for achieving real-time responsiveness. Specifically, we used the AXI Timer to precisely measure the latency between receiving HDMI video frames and updating the LED strip output. By reading the timer registers from the MicroBlaze firmware at specific checkpoints, we confirmed that the total latency from HDMI input capture to LED response was consistently around 19.7 milliseconds, meeting our target of low-latency real-time performance.

In addition to latency verification, we utilized the timer during debugging sessions to check synchronization and timing constraints between various IP blocks, ensuring stable and correct data flow across the system. The AXI Timer's straightforward AXI Lite interface simplified

integration into our firmware, making these timing diagnostics both accurate and easy to perform throughout the development process.

4.2.6 Memory Interface Generator

We used the Xilinx Memory Interface Generator (MIG 7 Series) to interface our FPGA design with the Nexys Video board on board DDR3 memory. This IP provided high-speed memory access, essential for buffering HDMI video frames. MIG was configured to operate with AXI interfaces, allowing it to communicate directly with AXI VDMA IP and the MicroBlaze processor. By storing video data in external memory, we were able to handle real-time video streaming and processing without running into BRAM capacity limitations.

The MIG 7 Series was critical in maintaining smooth, uninterrupted HDMI pass-through and ensured stable video frame buffering, which was a key part of achieving our target performance of 60 FPS at 1024 x 768 resolution.

4.2.7 Video In to AXI4-Stream

The Video In to AXI4-Stream IP core is used to convert external video input signals, such as RGB or YUV video data input from the HDMI interface, into a standard AXI4-Stream data stream. It solves the problem of data interface standardization between traditional video interfaces and FPGA internal video processing logic.

The main functional structures of the Video In to AXI4-Stream IP core include video signal capture and analysis, format converter, and asynchronous FIFO buffer. Video In to AXI4-Stream can automatically extract the line synchronization (HSYNC), field synchronization (VSYNC), and data valid signal (DE) of the input video data to determine the start and end positions of the video frame. It also supports converting the input video data into the AXI4-Stream format, including adding TUSER signals to mark the start of the video line, and the start and end of the field. The purpose of this function is to enable subsequent modules to correctly identify the video timing. In addition, this IP core also provides a data cache mechanism across clock domains. It allows the video input interface and the internal processing module to use different clock frequencies to avoid data loss or jitter caused by clock differences.[12]

This project uses this IP module to implement a standardized data interface from video input to video processing logic. This module parses the original signal of the video input interface and converts it into an AXI4-Stream data stream, which is then output to the S2MM interface of the VDMA. The input source of Video In to AXI4-Stream core is the HDMI input signal in RGB format, and its output is connected to the VDMA core, following the AXI4-Stream protocol.

4.2.8 Video Timing Controller

We used two instances of Xilinx's Video Timing Controller IP to manage the HDMI input and output video signals. One VTC was configured as a timing detector for the incoming HDMI signal, automatically extracting critical parameters such as resolution, synchronization signals, and refresh rate from the input source. This capability was particularly useful because it simplified integration—we didn't need to manually specify all timing details, which can be error-prone, especially for various HDMI sources.[7]

The second VTC served as a timing generator for our HDMI output pipeline. It produced consistent and stable video timing signals, ensuring that our HDMI output matched the timing parameters extracted from the input VTC. This configuration helped us avoid synchronization issues, reduced video artifacts and allowed for a seamless, real-time video pass-through at our targeted 60 FPS and 1024×768 resolution.

Working with VTC simplified the handling of the complexity of video signal synchronization, making our design more robust and adaptive to different HDMI input sources. Initially, we underestimated how challenging timing synchronization might be; the VTC IP greatly reduced these challenges and was essential for achieving stable video performance throughout the project.

4.2.9 AXI4-Stream to Video Out

The AXI4-Stream to Video Out module is used to convert the AXI4-Stream video data stream processed by the FPGA into a standard external video interface signal such as VGA or HDMI signal, and drive the display device.

The internal structure and functional features of the module include AXI4-Stream decapsulation, video timing and synchronization generation, video frame processing function, and flexible configuration of resolution and timing. AXI4-Stream decapsulation converts the pixel data and synchronization signals in the AXI4-Stream data packet into a parallel pixel data stream by extracting and rearranging them. In addition, AXI4-Stream to Video Out generates the HSYNC, VSYNC, DE signals, and pixel clock signals required by the external display according to the resolution and timing standards configured by the user. It also supports automatic frame repeating and black frame filling when the video frame is lost or the input data is insufficient, which can effectively avoid the phenomenon of screen distortion or signal loss in the video output. Users can dynamically adjust the resolution, frame rate, and pixel clock frequency of the video output by configuring registers to meet the compatibility requirements of different displays.[13]

In this project, this module is located at the end of the video processing pipeline. It repackages the cached video data read from the MM2S channel of VDMA into a standard video signal output and sends it to RGB to DVI Video Encoder core for the display device to present the final

processing effect. By flexibly configuring the interface parameters, the project can be compatible with different display terminals such as HDMI displays, and can guarantee the quality and stability of the video signal to a certain extent.

4.3 Customer IPs

4.3.1 LED_Strip_Output

The LED_Strip_Output module is responsible for extracting the RGB values from the processed video edge color data and using them as input to control the LED strip, ensuring synchronized color variation with screen content. This IP has the following interfaces:

```
module light_strip_output(
    input                   pclk,           // cmos pixel clock, video clk 65M
    input                   reset,          // reset signal
    input                   hs,             // horizontal synchronization
    input                   vs,             // frame synchronization, down edge positive
    input                   de,             // video valid
    input[23:0]             data888,        // RGB888 data from HDMI encode
    output                  light_rgb_data, //  data to light strip
    output                  uart_data_en    //  data enable signal
   );
```

This module divides the entire screen into 558 regions (with 98 edge regions of the screen corresponding one-to-one with the 98 LED pixels on the strip), and computes the average color of each region in real time. Operating under the pixel clock, the module utilizes horizontal sync (HS) and vertical sync (VS) signals, and applies specific algorithms to calculate and extract edge pixels, performing accumulation and summation of RGB values for pixels within the same region. To enhance stability, the module adopts an inter-frame accumulation filtering strategy, updating the region color once every five frames to ensure smooth transitions. At the end of each frame, the calculated average RGB values for all regions are written into on-chip BRAM. Subsequently, a finite state machine (FSM) within the module sequentially reads the region color values from BRAM and outputs the data through a FIFO queue to the subsequent LED driver module. The use of FIFO decouples video capture from LED transmission, ensuring continuous LED data output without frame loss.

Precise region division counting logic, wide-bit accumulation registers to prevent overflow, and frame-based timing control mechanisms.

Ensuring real-time processing and synchronization under high-resolution video input. Through a well-structured region division algorithm, BRAM caching, and a dual-buffer output mechanism, the module achieves accurate mapping of screen edge colors to LED strip display.

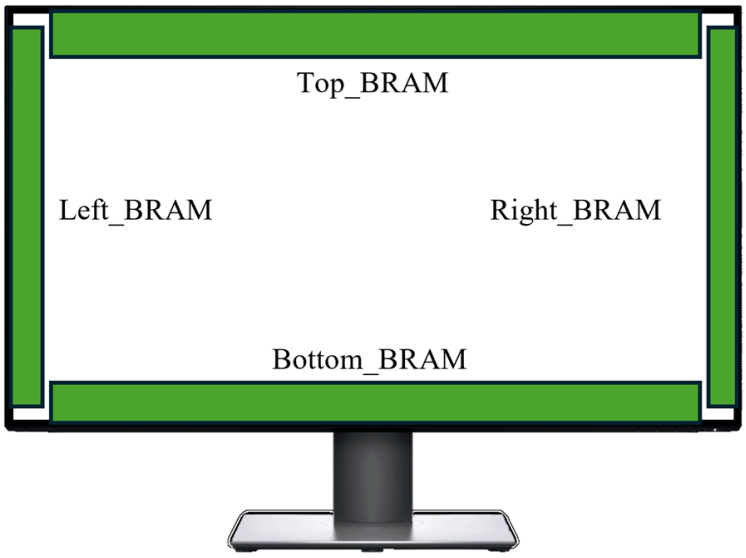

Figure 3: Schematic diagram of the display regions

4.3.2 RZ_Code

The RZ_Code module is responsible for converting RGB color values into 800kHz PWM sequences conforming to the single-wire Return-to-Zero (RZ) encoding protocol defined by WS2812B, thereby driving the LED strip to accurately display colors. This IP has the following ports:

```
module RZ_Code(
    input                   clk,            //   Pixel clock
    input                   rst_n,          //   reset signal
    input   [23:0]          RGB,            //   the order is GRB
    input                   tx_en,          //   send data enable
    output                  tx_done,        //   send data finish
    output                  RZ_data         //   Return_Zero code data
);
```

The module employs the RZ encoding principle: it transmits 24-bit RGB data bit-by-bit in GRB channel order, with each bit lasting approximately 1.25 microseconds. A Logic 1 is represented by a high-level pulse of about 0.8 microseconds, while a Logic 0 is represented by a high-level pulse of about 0.4 microseconds, followed by a return to zero.

Data transfer time(TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50μs	

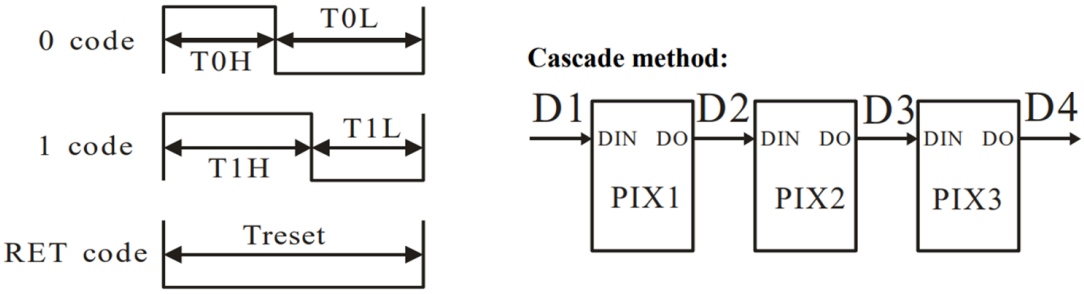

Figure 4: WS2812B Driver Protocol Timing Schematic

The module directly utilizes the display's Pixel Clock to achieve precise timing. An internal counter defines the 1.25 microsecond bit period, and the output pulse width is controlled by comparing counter values.

A finite state machine begins sequential output of 24-bit color data upon detection of a valid transmit enable signal (tx_en), and upon completion of the current pixel data transmission, it generates a transmission done flag (tx_done) to notify the upstream module to load the next RGB value and initiate LED strip reset. This enables seamless serial driving of multiple LED pixels.

Calibration presets for the pulse-length counter in RZ_Code (to ensure pulse widths strictly conform to protocol specifications), and bit sequence output logic based on GRB order.

Strictly adhering to the extremely narrow timing tolerances of the WS2812B protocol: precise control over each PWM pulse width is essential, as any small timing deviation accumulates and can ultimately lead to display disorder. In addition, after all data is transmitted, the bus must remain at a low level for over 50 microseconds to trigger internal data latching within the LED. Through meticulous counter parameter design and FSM flow planning, this module reliably drives WS2812B LED strips, enabling accurate color representation in accordance with RGB data.

5. Design Tree

Github Repository: https://github.com/Madnessvia/LumiSync

We have three main folder: **golden**, **hw**, and **sw**

The **golden** folder contains the bitstream used to program the FPGA, and the software program that runs on the FPGA, these files can be used directly without need to download and run other files.

The **hw** folder contains all the sources necessary to generate the bitstream in Vivado, including our custom IP, imported third-party IPs, block diagram, Verilog modules and constraints. The project file is also included.

The **sw** folder contains all the sources necessary to generate the software program in Vitis. There are five folders, each containing source code and header files, and a video_demo.c file as the program main. A README file is also included with detailed structure of the directory and instructions to reproduce our project.

6. Tips and Tricks

If you truly believe something should work after you have done a thorough test and simulation, it might be a problem with the board, and try to borrow another board to try it out. It is strongly recommended to test the board as soon as you sign out one from the lab, there are plenty of demo projects from the board vendor.

Start integration as soon as two independent components are verified working. Leaving all integration at the end is never a good plan even though all the individual components are completed ahead of time. There are unpredictable problems such as software versions and compatibility issues that will cost a lot of time to solve.

If you are buying external parts that require a power source, do not draw power from the FPGA board. These educational boards are not as strong as you think, you could burn it even though the current draw is much less than the rated number. In our case, we were using the board to drive ~100 LEDs, the board has built-in resistors as well as our LED strip. However, everything was good until we added HDMI passthrough functionality to our design, which adds to power utilization. Although the power consumption is theoretically under the max it can draw, the board was actually burnt and started malfunctioning. An advice is to buy an external power source but also make sure to share a common ground with the board if you are using the board's GPIO to

send data. We used a USB to Serial dongle with a 5V power pin that shares a common ground with the FPGA through the laptop.

7. Video

A video demo has been posted on Youtube: https://www.youtube.com/watch?v=ZBh9cPyp7LM

Reference

[1] “Smart lighting,” *Philips Hue*. https://www.philips-hue.com/en-ca

[2] “Govee Envisual TV Backlight T2,” *Govee.com*, 2025. https://ca.govee.com/products/govee-envisual-tv-backlight-t2 (accessed Apr.6, 2025).

[3] “iCUE LS100 Smart Lighting Strip Starter Kit,” *CORSAIR*, 2025. https://www.corsair.com/tw/en/p/ambient-lights/cd-9010001-na/icue-ls100-smart-lighting-strip-starter-kit-cd-9010001-na (accessed Apr. 6, 2025).

[4] Digilent, “Digilent/Vivado-Library,” GitHub, https://github.com/Digilent/vivado-library (accessed Apr. 7, 2025)

[5] Xilinx, "MicroBlaze Processor Reference Guide," UG984 (v2023.2). [Online]. Available: https://docs.xilinx.com/r/en-US/ug984-vivado-microblaze-ref

[6] Xilinx, "7 Series FPGAs Memory Interface Solutions," UG586 (v4.3). [Online]. Available: https://docs.xilinx.com/r/en-US/ug586_7Series_MIS

[7] Xilinx, "Video Timing Controller Product Guide," PG016 (v7.2). [Online]. Available: https://docs.xilinx.com/r/en-US/pg016_v_tc

[8] Xilinx, "Video LogiCORE IP AXI Video Direct Memory Access (axi_vdma)," DS799 (v3.01.a). [Online]. Available: https://docs.amd.com/v/u/en-US/ds799_axi_vdma

[9] Xilinx, "AXI Video Direct Memory Access v6.3 LogiCORE IP Product Guide," DS799 (v6.3). [Online]. Available: https://docs.amd.com/r/en-US/pg020_axi_vdma

[10] Xilinx, "LogiCORE IP AXI UART Lite," DS741 (v1.02a). [Online]. Available:https://docs.amd.com/v/u/en-US/axi_uartlite_ds741

[11] Xilinx, "AXI UART Lite v2.0 LogiCORE IP Product Guide," PG142 (v2.0). [Online]. Available: https://docs.amd.com/v/u/en-US/pg142-axi-uartlite

[12] Xilinx, "Video In to AXI4-Stream v5.0 LogiCORE IP Product Guide," PG043 (v5.0). [Online]. Available: https://docs.amd.com/r/en-US/pg043_v_vid_in_axi4s

[13]Xilinx, "AXI4-Stream to Video Out v4.0," PG044 (v4.0). [Online]. Available: https://www.xilinx.com/support/documents/ip_documentation/v_axi4s_vid_out/v4_0/pg044_v_axis_vid_out.pdf