SISTEMA DE VISUALIZACIÓN DE IMÁGENES CON CONTROL INTERACTIVO

Proyecto Final - Electrónica Digital II

Integrantes:

Emily Perea Córdoba • Juan David Figueroa • Cristian Camilo Vélez

Universidad Tecnológica de Pereira | Diciembre 2025

OBJETIVOS DEL PROYECTO

OBJETIVO GENERAL

Diseñar e implementar un sistema embebido en **FPGA** capaz de recibir, almacenar, visualizar y editar imágenes mediante protocolos de comunicación estándar.

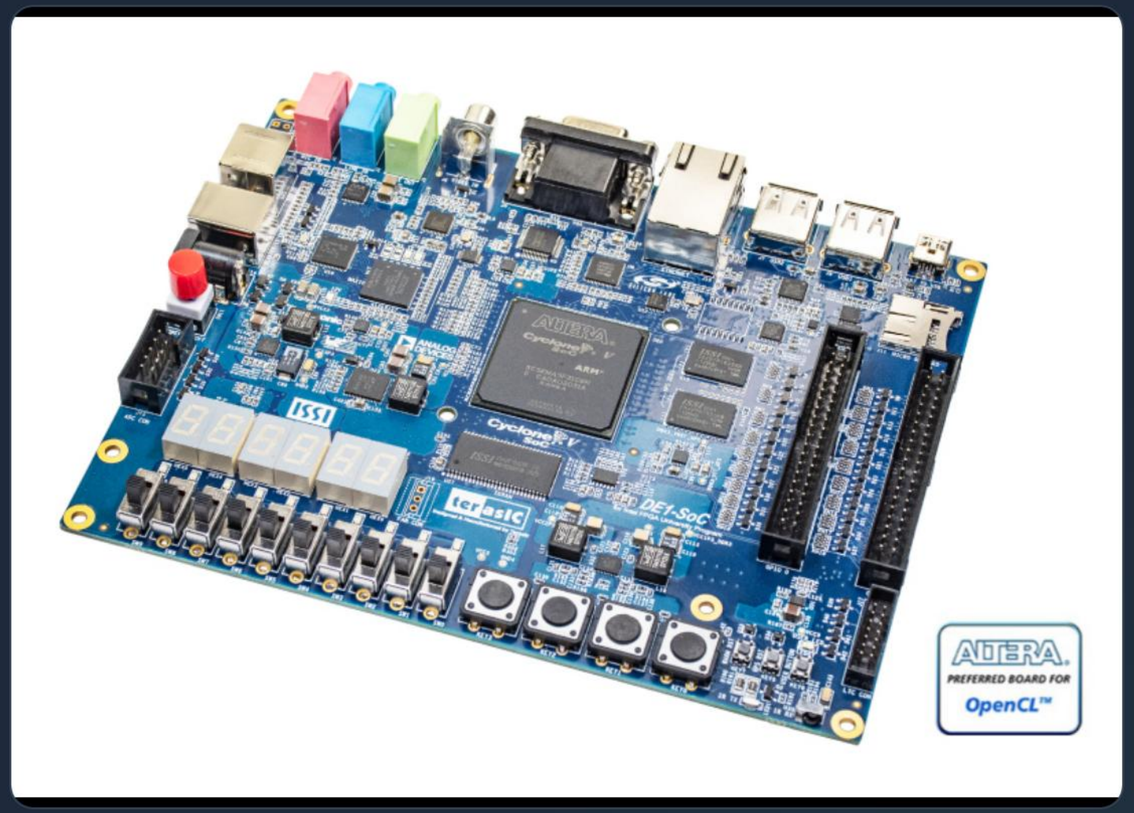
OBJETIVOS ESPECÍFICOS

- Implementar receptor **UART a 115200 baudios** con detección de errores.
- Diseñar sistema de almacenamiento en **BRAM dual-port** (307,200 bytes).
- Generar señales **VGA 640×480 @ 60 Hz**.
- Desarrollar un sistema de cursor interactivo para edición de píxeles.

ARQUITECTURA DEL SISTEMA

FLUJO DE DATOS

- **PC (Python):** Origen de la imagen y procesamiento previo.
- **UART (RX+TX):** Interfaz de comunicación serie de alta velocidad.
- **FPGA (Control + BRAM):** Lógica de control, FSMs y memoria de video.
- **VGA Controller:** Generación de sincronismo y señales RGB.
- **Monitor:** Visualización final e interacción.



INFRAESTRUCTURA HARDWARE Y SOFTWARE

HARDWARE

Tarjeta DE1-SoC (Terasic)
FPGA Cyclone V 5CSEMA5F31C6
Reloj: 50 MHz

HERRAMIENTAS

Quartus Prime Lite 20.1.1
ModelSim (Simulación)
VS Code (Verilog)

SOFTWARE PC

Python 3.11
Librerías: PySerial, NumPy, PIL (Pillow)

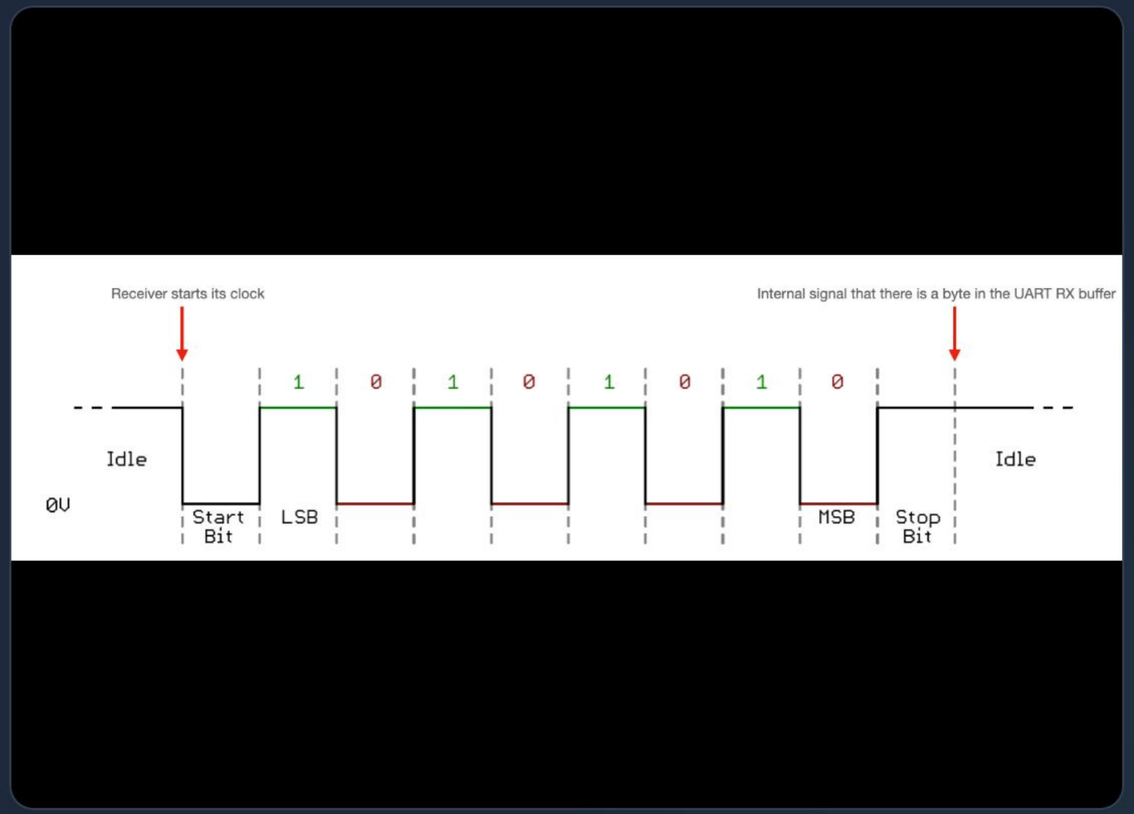
MÓDULO UART_FSM

ESPECIFICACIONES

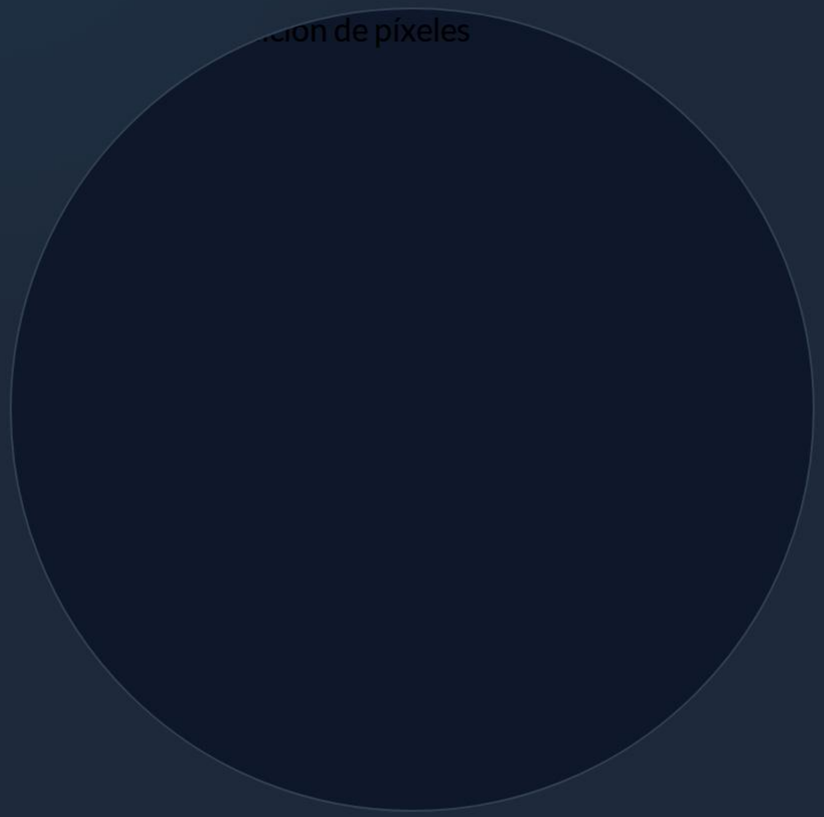
- **Baudrate:** 115200 bps
- **Formato:** 8 bits datos, 1 start, 2 stop bits, sin paridad.
- **Muestreo:** Centro del bit (HALF_TICK = 217 ciclos).
- **Sincronización:** 2 flip-flops para evitar metaestabilidad.

MÁQUINA DE ESTADOS

IDLE → START → DATA[0:7] → STOP1 → STOP2



MÓDULO DE CONTROL DE CURSOR



FUNCIONALIDAD

- **Movimiento:** KEY[3:0] (Arriba, Abajo, Izq, Der).
- **Acción:** SW[0] Enable (Pintar/Mover).
- **Color:** SW[3:1] Selección RGB.

CÁLCULO DE DIRECCIÓN OPTIMIZADO

Evita multiplicación costosa en hardware:

$$Addr = (y << 9) + (y << 7) + x$$

Equivalente a: $y \times 640 + x$

CONTROLADOR VGA (TIMING)

Configuración para **640×480 @ 60Hz** con Pixel Clock de 25 MHz.

PARÁMETRO	HORIZONTAL (PÍXELES)	VERTICAL (LÍNEAS)
Active Display	640	480
Front Porch	16	10
Sync Pulse	96	2
Back Porch	48	33
Total	800	525

LÓGICA DE VISUALIZACIÓN Y MULTIPLEXADO

FORMATO DE COLOR & CURSOR

RGB111: 3 bits (1 bit por canal) expandidos a 8 bits (0x00 o 0xFF).

Efecto Cursor: Inversión lógica del color bajo el cursor para máxima visibilidad.

```
pixel_out = show_cursor ? ~pixel : pixel;
```

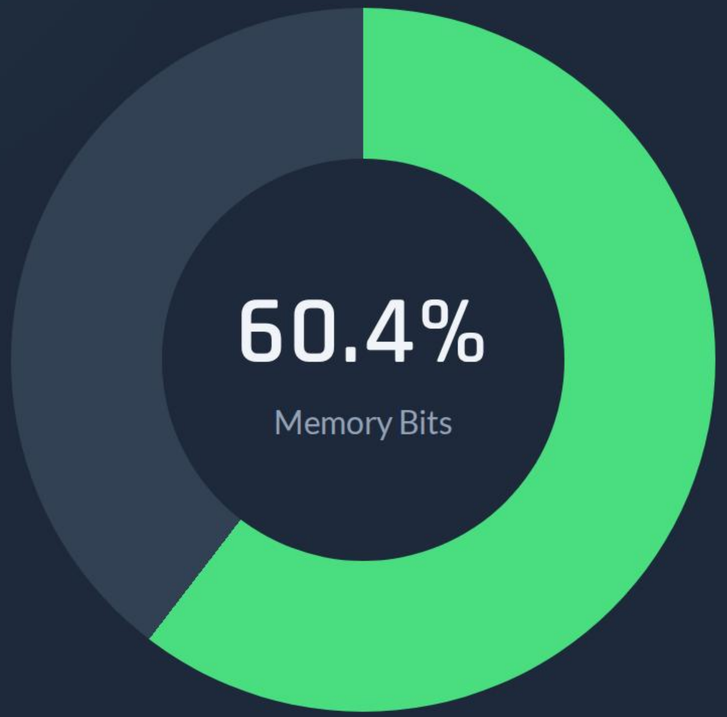
MULTIPLEXADO DE MEMORIA (BRAM)

Se implementa un sistema de prioridades para evitar la corrupción de datos durante la escritura.

- **Prioridad Alta:** UART (Recepción de imagen).
- **Prioridad Baja:** Cursor (Edición manual).

UART escribe 307,200 bytes secuencialmente; el cursor escribe píxeles individuales bajo demanda.

USO DE RECURSOS (CYCLONE V)



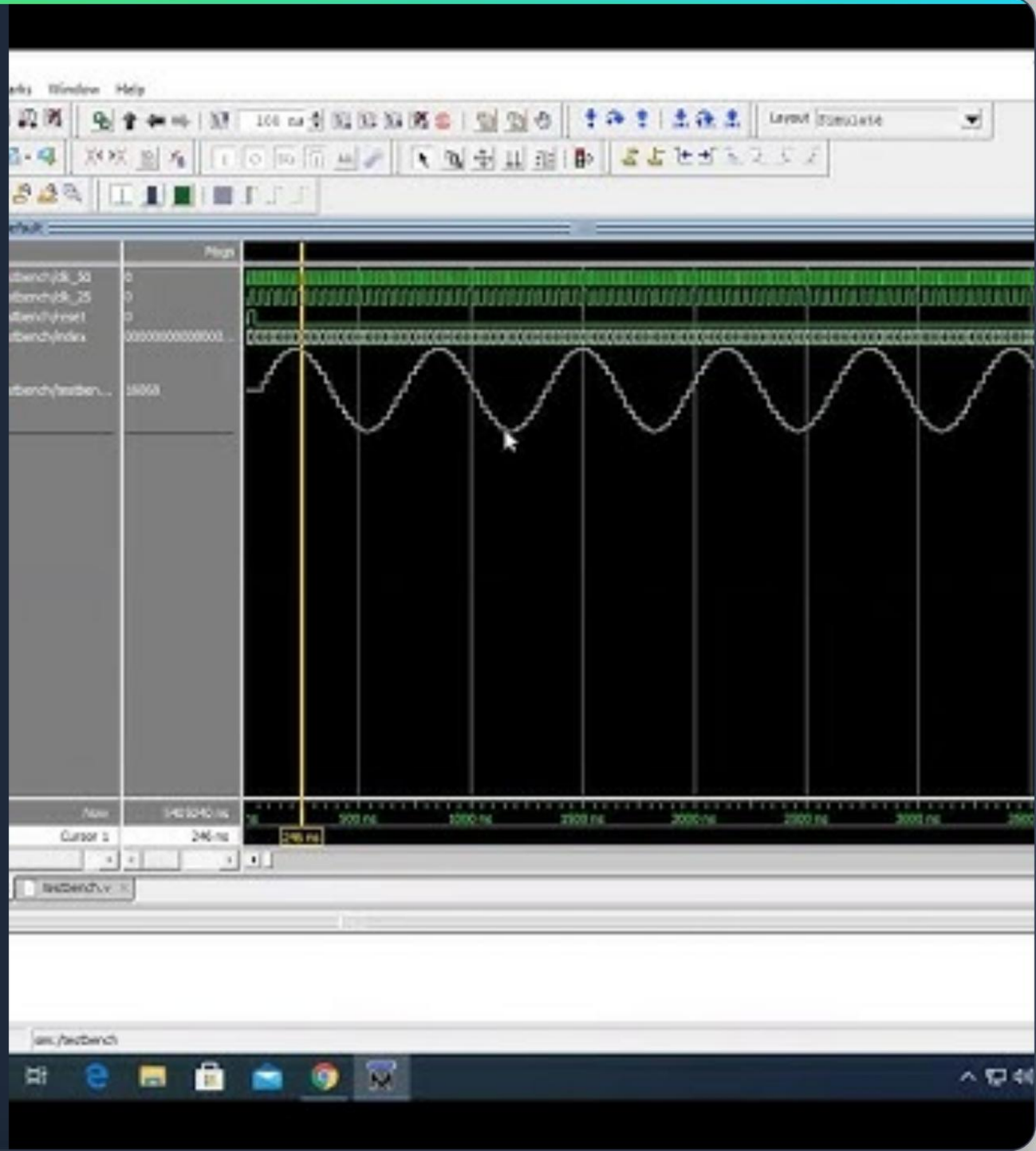
- Memory Bits (2,457,600 bits)
- Disponible

- ✓ Logic Elements: < 1% (286 / 32,070)
- ✓ DSP Blocks: 0% (Optimización Shift-Add)

RESULTADOS DE SIMULACIÓN

VALIDACIÓN DE TESTBENCHES

- **uart_rx_tb.v:** Verificación de recepción de bytes válidos y rechazo de glitches.
- **vga_system_tb.v:** Comprobación de timings (Hsync 800 ciclos, Vsync 525 líneas).
- **cursor_system_tb.v:** Validación de prioridades y límites de pantalla.
- **tb_uart_bram.v:** Integración correcta escritura/lectura en memoria.



ANÁLISIS DE TIMING

REPORTE MULTICORNER

Worst-case Slack (Setup): -5.547 ns

Worst-case Slack (Hold): -0.251 ns

Nota: Los valores negativos son teóricos bajo condiciones extremas (85°C, voltaje mínimo). En laboratorio (25°C) funciona correctamente.

CAUSAS Y SOLUCIONES

Factores:

- Divisor de reloj por toggle (Ripple Clock) en lugar de PLL.
- Cálculo combinacional profundo para direcciones BRAM.
- Contadores VGA grandes (comparadores de 10 bits).

Solución: Implementar PLL dedicado para reloj de 25 MHz.

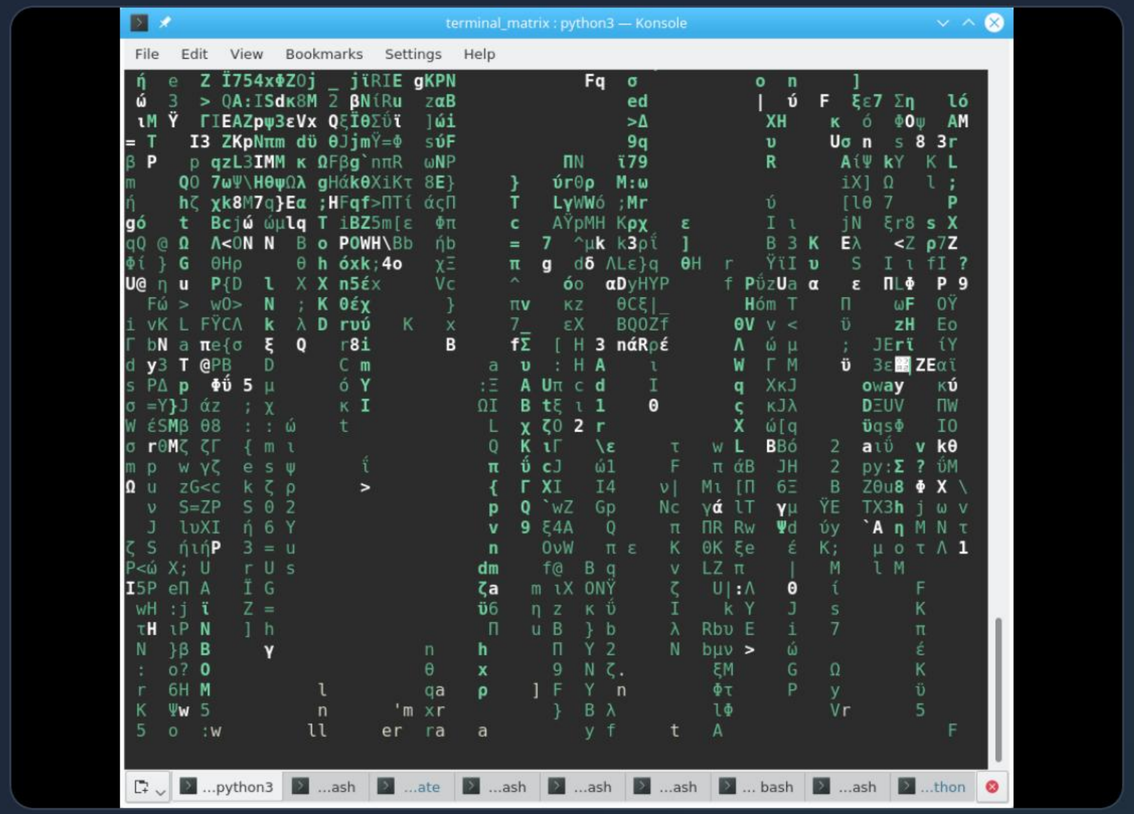
SCRIPT DE TRANSMISIÓN (PYTHON)

PROCESAMIENTO Y ENVÍO

1. Carga de imagen con **PIL**.
2. Redimensionamiento a **640x480** (Nearest Neighbor).
3. Conversión de RGB888 a **RGB111** byte a byte.
4. Transmisión serie vía **PySerial**.

Tiempo de transmisión: ~27 segundos

Total Bytes: 307,200



ANÁLISIS DE RESULTADOS

FORTALEZAS

- **Sincronización Robusta:** Muestreo central y doble FF en UART.
- **Eficiencia:** Uso mínimo de recursos lógicos (< 1%) y 0 DSPs.
- **Modularidad:** Arquitectura limpia y fácil de mantener.
- **Estabilidad:** Imagen VGA estable sin parpadeos ni artefactos.

LIMITACIONES

- **Profundidad de Color:** Limitada a 8 colores (RGB111).
- **Velocidad:** UART a 115200 es lento para imágenes completas.
- **Timing:** Slack negativo teórico.
- **Interrupción:** No se puede editar mientras se recibe una imagen.

CONCLUSIONES Y TRABAJO FUTURO

CONCLUSIONES

- Integración exitosa de protocolos UART, memoria BRAM y visualización VGA.
- Sistema interactivo funcional en tiempo real con FPGA.
- Validación completa mediante simulación y pruebas en hardware.

TRABAJO FUTURO

- Implementar **PLL** para corregir timing.
- Aumentar profundidad a **RGB888**.
- Buffer circular para escritura asíncrona.
- Soporte para resoluciones mayores (1024x768).

¿PREGUNTAS?

Gracias por su atención.

Contacto del Equipo:

Facultad de Ingenierías | Ingeniería de Sistemas y Computación

Universidad Tecnológica de Pereira

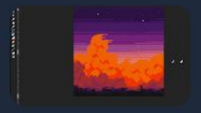
IMAGE SOURCES

https://www.terasic.com.tw/attachment/archive/836/image/top45_01.jpg

Source: www.terasic.com.tw

https://vanhunteradams.com/Protocols/UART/uart_timing.png

Source: vanhunteradams.com

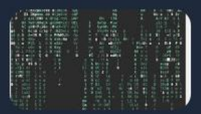


https://img.itch.zone/aW1nLzE3MTIwMjIxLnBuZw==/original/NTuV%2FB.png

Source: orama-interactive.itch.io

https://i.ytimg.com/vi/jo_5X5CyLzg/hq720.jpg?sqp=-oaymwEhCK4FEIIDSFryq4qpAxMIARUAAAAAGAElAADIQj0AgKJD&rs=AOn4CLDQppDz7saspy6a44Ezb2HPuUwMlw

Source: www.youtube.com



https://raw.githubusercontent.com/jsbueno/terminal_matrix/gh-pages/screenshot.png

Source: github.com