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

Objetivo General

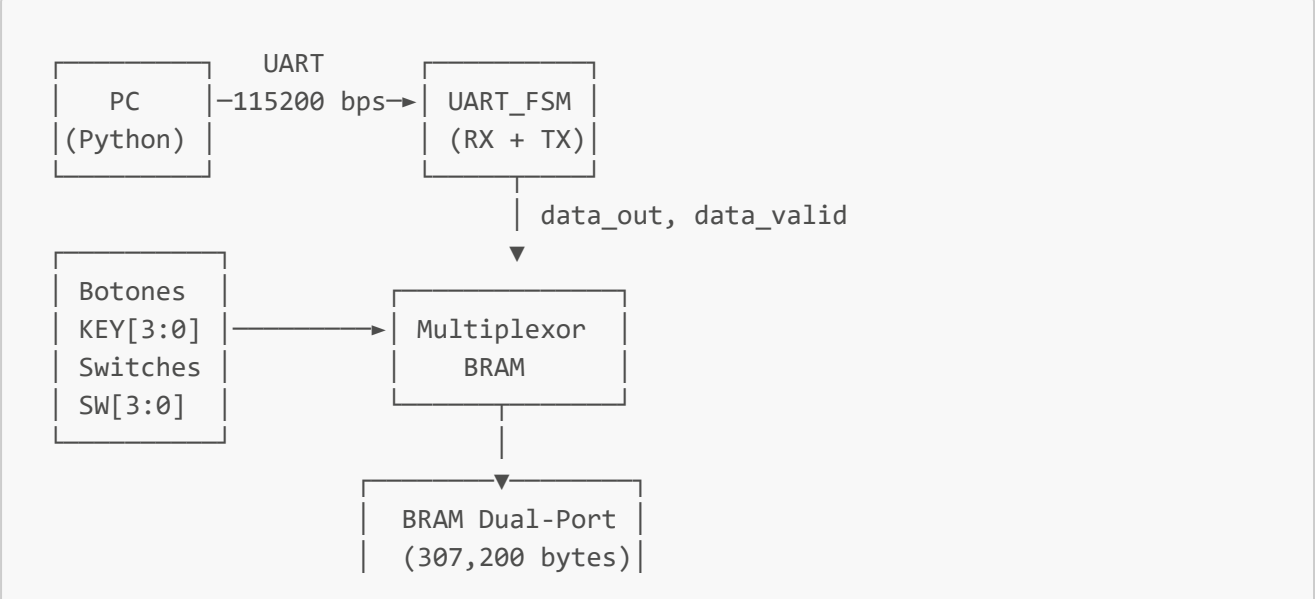
Diseñar e implementar un sistema embebido en FPGA capaz de recibir, almacenar, visualizar y editar imágenes mediante protocolos de comunicación estándar.

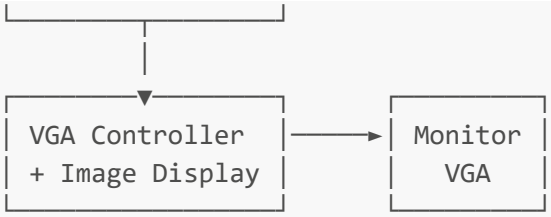
Objetivos Específicos

- Implementar receptor UART a 115200 baudios con detección de errores
- Diseñar sistema de almacenamiento en BRAM dual-port de 307,200 bytes
- Generar señales VGA compatibles con 640×480 @ 60 Hz
- Desarrollar sistema de cursor interactivo con edición de píxeles
- Validar funcionamiento mediante simulación y pruebas en hardware

---

Arquitectura del Sistema





Infraestructura Hardware

Tarjeta DE1-SoC (Terasic)

- **FPGA:** Cyclone V 5CSEMA5F31C6
- **Logic Elements:** 32,070
- **Memory Bits:** 4,065,280 (M10K blocks)
- **Reloj:** 50 MHz

Periféricos Utilizados

- **Entradas:** 4 botones (KEY), 4 switches (SW), UART RX
- **Salidas:** VGA (24-bit RGB + sync), UART TX, 10 LEDs
- **Interfaces:** USB-UART (FT232), VGA DAC (ADV7123)

Infraestructura Software

Herramientas de Desarrollo

- **Quartus Prime Lite 20.1.1:** Síntesis, place & route, programación
- **ModelSim:** Simulación funcional de testbenches
- **Python 3.11:** Script de transmisión de imágenes
- **VS Code:** Editor con extensión Verilog

Librerías Python

```
import serial      # PySerial: comunicación UART
import numpy       # Procesamiento de arrays
from PIL import Image  # Manipulación de imágenes
```

Módulo UART_FSM

Características

- **Baudrate:** 115200 bps
- **Formato:** 8 bits datos, 2 stop bits, sin paridad
- **Sincronización:** 2 flip-flops anti-metaestabilidad

- **Muestreo:** Centro del bit (HALF_TICK = 217 ciclos)
- **Eco:** TX retransmite bytes recibidos

FSM del Receptor

```
RX_IDLE → RX_START → RX_DATA[0:7] → RX_STOP1 → RX_STOP2 → RX_VALID_DATA
```

Detección de Errores

- Frame error si stop bits ≠ 1
- Glitch rejection en start bit

Módulo Cursor Control

Funcionalidad

- **KEY[0]:** Arriba | **KEY[1]:** Abajo
- **KEY[2]:** Izquierda | **KEY[3]:** Derecha
- **SW[0]:** Enable pintar (1 = escribir, 0 = solo mover)
- **SW[3:1]:** Color RGB a pintar

Cálculo Optimizado de Dirección

```
// Evita multiplicación hardware
addr = (cursor_y << 9) + (cursor_y << 7) + cursor_x
     = y × 512 + y × 128 + x
     = y × 640 + x
```

Límites

- X: [0, 639] | Y: [0, 479]

Módulo VGA Controller

Timing 640×480 @ 60Hz (25 MHz)

Parámetro	Horizontal	Vertical
Display	640 px	480 líneas
Front porch	16 px	10 líneas
Sync pulse	96 px	2 líneas
Back porch	48 px	33 líneas

Parámetro	Horizontal	Vertical
Total	**800 px**	**525 líneas**

Frecuencia de refresco: 25 MHz / (800 × 525) ≈ **59.52 Hz**

VGA Image Display + Cursor

Características

- **Formato RGB111:** 00000RGB (3 bits LSB de byte)
- **Expansión a 8 bits:** 0 → 0x00, 1 → 0xFF
- **Cursor parpadeante:** ~2 Hz (25 MHz / 2 = 12.5M ciclos toggle)
- **Inversión de color:** Pixel cursor = ~pixel original

Lógica del Cursor

```
show_cursor = (hcount == cursor_x) &&
              (vcount == cursor_y) &&
              cursor_visible;

red_out = show_cursor ? ~pixel_red : pixel_red;
```

Multiplexado de Escritura BRAM

Prioridad: UART > Cursor

```
// UART tiene prioridad absoluta
assign bram_wr_addr = uart_valid ? write_addr : cursor_addr;
assign bram_wr_data = uart_valid ? uart_data  : cursor_data;
assign bram_wr_en   = uart_valid | cursor_write_en;
```

Razón

- Evitar corrupción de datos durante transmisión de imagen
- UART escribe 307,200 bytes secuencialmente
- Cursor solo escribe píxeles individuales bajo demanda

Recursos Utilizados

Cyclone V 5CSEMA5F31C6

Recurso	Usado	Total	%

Recurso	Usado	Total	%
Logic Elements	286	32,070	< 1%
Registers	277	-	-
Memory Bits	2,457,600	4,065,280	60.4%
DSP Blocks	0	84	0%
PLLs	0	6	0%

Nota: 0 DSP gracias a shift-add en lugar de multiplicación

Resultados de Simulación

Testbenches Implementados

1. **uart_rx_tb.v:** Verificación del receptor UART
    - ✓ Bytes válidos (0x00-0xFF)
    - ✓ Detección de frame errors
    - ✓ Rechazo de glitches
2. **tb_uart_bram.v:** Integración UART → BRAM
    - ✓ Escritura correcta en memoria
3. **vga_system_tb.v:** Timing VGA
    - ✓ Hsync: 800 ciclos | Vsync: 525 líneas
4. **cursor_system_tb.v:** Sistema completo
    - ✓ Prioridad UART | ✓ Límites | ✓ Edición condicional

Análisis de Timing

Reporte Multicorner

- **Reloj 50 MHz:** Período = 20 ns
- **Reloj 25 MHz:** Período = 40 ns
- **Worst-case Slack (Setup):** -5.547 ns
- **Worst-case Slack (Hold):** -0.251 ns

Interpretación

Los valores negativos corresponden a **condiciones extremas**:

- Temperatura: 85°C (vs. 25°C en laboratorio)
- Voltaje: mínimo (slow corner)

- Proceso: fabricación lenta

En la práctica: El diseño funciona correctamente en condiciones de laboratorio.

Causas de Violaciones de Timing

Principales Factores

1. **Divisor de reloj por toggle** (clk_divider)
    - No es un reloj "verdadero" (no usa PLL)
    - Crea rutas combinacionales largas
2. **Cálculo de direcciones BRAM**
    - Operaciones shift-add: `(y << 9) + (y << 7) + x`
    - Múltiples niveles de lógica combinacional
3. **Contadores VGA grandes**
    - hcount (0-799), vcount (0-524)
    - Incremento + comparación en mismo ciclo

Solución Recomendada

Usar PLL para generar 25 MHz o reducir frecuencia a 40 MHz

DEMOSTRACIÓN EN HARDWARE

Mostrando:

- Transmisión de imagen vía UART (Python)
- Visualización en monitor VGA 640×480
- Control de cursor con botones
- Edición de píxeles en tiempo real
- Selección de color con switches

Script Python de Transmisión

```python
def send_image_uart(image_path, port='COM4'):
    # Cargar y redimensionar imagen
    img = Image.open(image_path).convert('RGB')
    img = img.resize((640, 480), Image.Resampling.NEAREST)

    # Convertir a RGB111
    for y in range(480):
```

```
            for x in range(640):
                r, g, b = img.getpixel((x, y))
                byte = ((r > 127) << 2) | ((g > 127) << 1) | (b > 127)
                ser.write(bytes([byte]))

        print("307,200 bytes transmitidos")
```

Tiempo de transmisión: ~27 segundos @ 115200 bps

Resultados Obtenidos

✓ Transmisión UART

- 307,200 bytes recibidos sin errores
- Eco funcional (verificación byte por byte)
- Sin frame errors durante transmisión continua

✓ Visualización VGA

- Imagen estable sin parpadeos
- Sin artifacts visuales (tearing, ghosting)
- Colores correctos en paleta RGB111 (8 colores)

✓ Sistema de Cursor

- Movimiento fluido con botones
- Edición condicional según SW[0]
- Selección de color funcional
- Respeto de límites de pantalla

Análisis de Resultados

Fortalezas del Diseño

1. **Sincronización robusta:** 2-FF en entradas, muestreo central UART
2. **Multiplexado eficiente:** Prioridad UART evita corrupción
3. **Optimización de recursos:** < 1% LEs, 0 DSP blocks
4. **Arquitectura modular:** Fácil expansión y mantenimiento

Limitaciones Identificadas

1. **Profundidad de color:** Solo 8 colores (RGB111)
2. **Timing:** Slack negativo en análisis multicorner
3. **Velocidad UART:** Transmisión lenta (~27 seg)
4. **Sin buffer:** No se pueden editar píxeles durante recepción

---

Conclusiones

1. **Integración exitosa** de UART, BRAM dual-port y VGA en FPGA
2. **Sistema interactivo funcional** con cursor controlado por botones
3. **Eficiencia de recursos** mediante técnicas de optimización (shift-add)
4. **Validación completa** a través de simulación y pruebas hardware
5. **Cumplimiento de especificaciones:** 640×480, 115200 bps, tiempo real

Trabajo Futuro

- Implementar PLL para mejorar timing
- Aumentar profundidad de color (RGB888)
- Agregar buffer circular para escritura asíncrona
- Soporte para resoluciones superiores (1024×768)

¡Gracias!

Preguntas

Contacto:
Emily Perea Córdoba
Juan David Figueroa
Cristian Camilo Vélez

Universidad Tecnológica de Pereira
Facultad de Ingenierías
Programa de Ingeniería de Sistemas y Computación
