Sistema de Visualización de Imágenes con Control Interactivo

Proyecto Final - Electrónica Digital II

Integrantes:

- Emily Perea Córdoba
- Juan David Figueroa
- Cristian Camilo Vélez

Fecha: 1 de diciembre de 2025

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

Tabla de Contenidos

---

1. Introducción

Este proyecto implementa un sistema embebido en FPGA que permite la transmisión, almacenamiento y visualización de imágenes de 640×480 píxeles con formato RGB de 3 bits. El sistema utiliza comunicación UART para recibir datos desde un computador, memoria BRAM dual-port para almacenamiento, y protocolo VGA para visualización en monitor. Adicionalmente, incorpora un sistema de cursor interactivo controlado por botones que permite la edición de píxeles en tiempo real.

El diseño fue implementado en una tarjeta DE1-SoC de Terasic, utilizando un FPGA Cyclone V (5CSEMA5F31C6) y desarrollado en Verilog HDL mediante Quartus Prime Lite Edition.

---

2. 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

1. Implementar un receptor UART a 115200 baudios con detección de errores de trama
2. Diseñar un sistema de almacenamiento basado en BRAM dual-port de 307,200 bytes
3. Generar señales VGA compatibles con resolución 640×480 @ 60 Hz
4. Desarrollar un sistema de cursor interactivo con capacidad de edición de píxeles
5. Validar el funcionamiento mediante simulación y pruebas en hardware

---

3. Descripción del Sistema

El sistema completo consta de dos partes principales:

Parte 1: Sistema de Visualización de Imágenes

- **Recepción UART:** El sistema recibe 307,200 bytes secuencialmente desde un computador a través del protocolo UART a 115200 baudios. Cada byte representa un píxel en formato RGB de 3 bits (1 bit por canal).
- **Almacenamiento:** Los datos recibidos se almacenan en una memoria BRAM dual-port generada mediante IP Catalog de Quartus.
- **Visualización VGA:** El sistema lee la BRAM y genera señales VGA para mostrar la imagen en un monitor de 640×480 píxeles a 60 Hz.

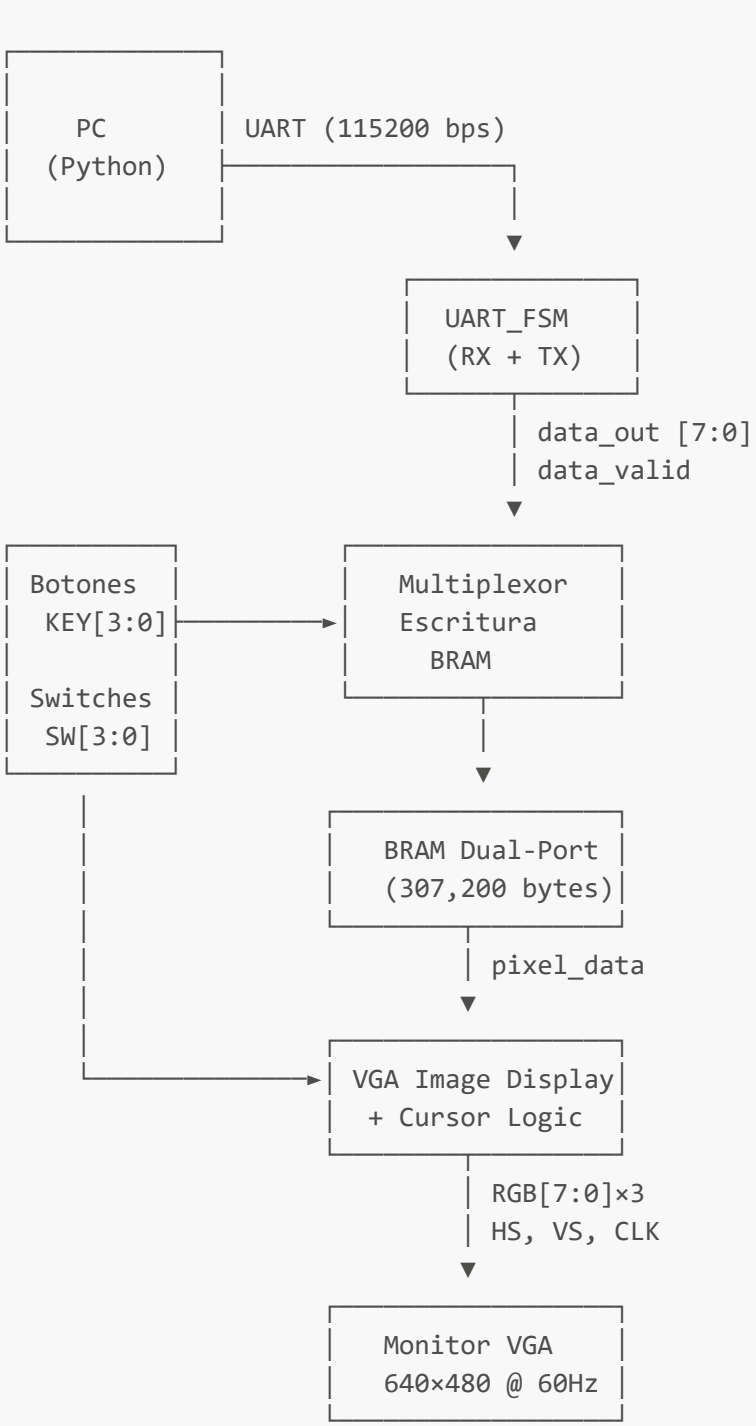
Parte 2: Sistema de Cursor Interactivo

- **Control de Cursor:** Cuatro botones (KEY[3:0]) permiten mover el cursor en las direcciones: arriba, abajo, izquierda y derecha.
- **Modo de Edición:** Un interruptor (SW[0]) habilita el modo de edición. Cuando está activo, al mover el cursor se escribe en la BRAM el color seleccionado.
- **Selección de Color:** Tres interruptores (SW[3:1]) permiten seleccionar el color RGB a pintar.
- **Visualización del Cursor:** El cursor se muestra como un píxel parpadeante a ~2 Hz mediante inversión del color actual.

---

4. Arquitectura del Sistema

Diagrama de Bloques General



Jerarquía de Módulos

```
fpga_top
├── Power-On Reset (POR)
├── clk_divider (50 MHz → 25 MHz)
├── uart_fsm (RX + TX con eco)
├── cursor_control (manejo de botones y switches)
├── ram_2port (BRAM dual-port IP)
├── vga_controller (generación de señales de sincronización)
└── vga_image_display (lectura BRAM y generación RGB)
```

5. Módulos Implementados

5.1 Módulo `uart_fsm.v`

Descripción: Implementa un transceptor UART completo con receptor y transmisor.

Características:

- Baudrate: 115200 bps
- Formato: 8 bits de datos, 2 stop bits, sin paridad
- Sincronización de entrada con 2 flip-flops (anti-metaestabilidad)
- Muestreo en el centro del bit para mayor robustez
- Detección de errores de trama (frame_error)
- Modo eco: transmite de vuelta los bytes recibidos

Parámetros:

```
localparam BAUD_TICK = 434;       // 50MHz / 115200 ≈ 434
localparam HALF_TICK = BAUD_TICK / 2;  // 217
```

FSM del Receptor:

- `RX_IDLE`: Espera start bit (flanco de bajada)
- `RX_START`: Verifica start bit en medio del período
- `RX_DATA`: Recibe 8 bits de datos (LSB primero)
- `RX_STOP1` / `RX_STOP2`: Verifica ambos stop bits
- `RX_VALID_DATA`: Entrega byte válido

FSM del Transmisor:

- `TX_IDLE`: Línea en alto, espera byte a transmitir
- `TX_START`: Envía start bit (0)
- `TX_DATA`: Envía 8 bits de datos
- `TX_STOP1` / `TX_STOP2`: Envía stop bits (1)

5.2 Módulo `cursor_control.v`

Descripción: Controla la posición del cursor y genera señales de escritura en BRAM.

Características:

- Sincronización de botones con 2 flip-flops
- Detector de flancos de bajada para detectar presión de botones
- Límites de pantalla: X=[0, 639], Y=[0, 479]
- Cálculo optimizado de dirección: `addr = (y << 9) + (y << 7) + x`
- Formato de color: 00000RGB (5 bits superiores en 0)

Lógica de Control:

```
// KEY[0] = Arriba,  KEY[1] = Abajo
// KEY[2] = Izquierda, KEY[3] = Derecha
// SW[0] = Enable edición (1 = pintar, 0 = solo mover)
// SW[3:1] = Color RGB (R, G, B)

cursor_write_en = SW[0] && (|key_press);  // Escribe si está habilitado Y se presiona botón
```

5.3 Módulo `vga_controller.v`

Descripción: Genera señales de sincronización VGA para resolución 640×480 @ 60 Hz.

Timing VGA 640×480 @ 60Hz (reloj 25.175 MHz ≈ 25 MHz):

Parámetro	Horizontal	Vertical
Display	640 píxeles	480 líneas
Front porch	16 píxeles	10 líneas
Sync pulse	96 píxeles	2 líneas
Back porch	48 píxeles	33 líneas
Total	**800 píxeles**	**525 líneas**

Frecuencia de refresco:

```
f_refresh = 25 MHz / (800 × 525) ≈ 59.52 Hz
```

Señales generadas:

- `hsync`: Sincronización horizontal (activo bajo)
- `vsync`: Sincronización vertical (activo bajo)
- `display_enable`: Alto durante área visible (640×480)
- `hcount`, `vcount`: Contadores de posición actual

5.4 Módulo `vga_image_display.v`

Descripción: Lee datos de BRAM y genera señales RGB con soporte para cursor parpadeante.

Características:

- Extracción de formato RGB111: 00000RGB
- Expansión a 8 bits: cada bit se replica (0→0x00, 1→0xFF)
- Generador de parpadeo del cursor (~2 Hz)
- Inversión de color en posición del cursor cuando está visible
- Salida deshabilitada durante blanking

Cálculo de dirección de lectura:

```
addr = (y << 9) + (y << 7) + x  // y × 640 + x
```

Lógica del cursor:

```
show_cursor = (hcount == cursor_x) && (vcount == cursor_y) && cursor_visible;
red_final   = show_cursor ? ~red_value : red_value;
```

5.5 Módulo `clk_divider.v`

Descripción: Divide el reloj de 50 MHz a 25 MHz para VGA.

Implementación:

```
always @(posedge clk_50mhz) begin
    if (reset)
        clk_25mhz <= 0;
    else
        clk_25mhz <= ~clk_25mhz;  // Toggle cada ciclo = división por 2
end
```

5.6 Módulo `ram_2port.v`

Descripción: Memoria BRAM dual-port generada mediante IP Catalog de Quartus.

Especificaciones:

- Capacidad: 307,200 bytes (640 × 480 píxeles)
- Ancho de palabra: 8 bits
- Puerto A (Escritura): UART o Cursor
- Puerto B (Lectura): VGA
- Latencia de lectura: 1 ciclo de reloj

5.7 Módulo `fpga_top.v` (Top-Level)

Descripción: Integra todos los módulos y gestiona el multiplexado de escritura en BRAM.

Multiplexor de Escritura (Prioridad: UART > Cursor):

```
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;
```

Power-On Reset (POR):

```
reg [15:0] por_cnt = 16'hFFFF;
wire reset = (por_cnt != 0);  // Reset activo durante ~1.3 ms
```

---

6. Resultados de Simulación

6.1 Testbench `uart_rx_tb.v`

Objetivo: Verificar el funcionamiento del receptor UART.

Pruebas realizadas:

1. Recepción de bytes válidos (0x00 a 0x07)
2. Patrones específicos (0xAA, 0x55, 0xFF)
3. Byte con error de frame (stop bit incorrecto)
4. Bytes consecutivos rápidos
5. Glitch en start bit (falso inicio)

Resultados:

- ✓ Todos los bytes válidos recibidos correctamente
- ✓ Frame error detectado cuando stop bit = 0
- ✓ Falsos inicios ignorados correctamente
- ✓ Sin errores en transmisión continua

6.2 Testbench `tb_uart_bram.v`

Objetivo: Verificar la integración UART → BRAM.

Pruebas realizadas:

1. Envío de byte 0x10 por UART
2. Verificación de escritura en BRAM[0]

Resultados:

- ✓ Byte almacenado correctamente en dirección 0
- ✓ Señal `data_valid` genera escritura en BRAM

6.3 Testbench `vga_system_tb.v`

Objetivo: Verificar generación de señales VGA.

Pruebas realizadas:

1. Verificación de timing de hsync y vsync
2. Contador de píxeles y líneas

3. Señal display_enable en área correcta

Resultados:

- ✓ Hsync: período de 800 píxeles
- ✓ Vsync: período de 525 líneas
- ✓ Display_enable activo solo en 640×480

6.4 Testbench `cursor_system_tb.v`

Objetivo: Verificar sistema completo con cursor.

Pruebas realizadas:

1. Carga de datos por UART (4 bytes)
2. Movimiento de cursor sin pintar (SW[0]=0)
3. Movimiento de cursor pintando (SW[0]=1)
4. Cambio de color de pintura
5. Prioridad UART sobre cursor
6. Verificación de límites (esquinas)

Resultados:

- ✓ UART tiene prioridad sobre escritura del cursor
- ✓ Cursor se mueve correctamente con botones
- ✓ Escritura en BRAM solo cuando SW[0]=1
- ✓ Límites respetados: (0,0) y (639,479)
- ✓ Color configurable mediante switches

---

7. Implementación en Hardware

7.1 Asignación de Pines (DE1-SoC)

Señal	Pin	Descripción
CLOCK_50	AF14	Reloj 50 MHz
KEY[0]	AA14	Botón arriba
KEY[1]	AA15	Botón abajo
KEY[2]	W15	Botón izquierda
KEY[3]	Y16	Botón derecha
SW[0]	AB12	Enable pintar
SW[1]	AC12	Color B
SW[2]	AF9	Color G
SW[3]	AF10	Color R

Señal	Pin	Descripción
UART_RXD	AF25	UART entrada
UART_TXD	AG25	UART salida
VGA_HS	B11	VGA Hsync
VGA_VS	D11	VGA Vsync
VGA_CLK	A11	VGA clock
VGA_BLANK_N	F10	VGA blank
VGA_R[7:0]	A13-F13	VGA rojo
VGA_G[7:0]	J9-E11	VGA verde
VGA_B[7:0]	B13-J14	VGA azul
LEDR[9:0]	V16-Y21	LEDs debug

7.2 Recursos Utilizados

Cyclone V 5CSEMA5F31C6:

- Logic Elements: ~286 / 32,070 (< 1 %)
- Registers: ~277
- Memory Bits: 2,457,600 / 4,065,280 (23 %)
- DSP Blocks: 0 / 84
- PLLs: 0 / 6

7.3 Temporización

Análisis de Timing:

- Reloj principal: 50 MHz → Período: 20 ns
- Reloj VGA: 25 MHz → Período: 40 ns
- Worst-case Slack (Setup): -5.547 ns
- Worst-case Slack (Hold): -0.251 ns

Nota: Los valores negativos de slack reportados por el análisis multicorner corresponden a condiciones worst-case (temperatura 85°C, voltaje mínimo, proceso de fabricación lento). En pruebas de laboratorio a temperatura ambiente (~25°C), el diseño opera correctamente sin errores de sincronización observable. Las violaciones de timing se deben principalmente al uso de divisor de reloj por toggle en lugar de PLL y a rutas combinacionales largas en el cálculo de direcciones BRAM. Para aplicaciones críticas se recomienda optimización mediante PLL para generación de relojes o reducción de frecuencia operativa.

7.4 Pruebas en Hardware

Configuración:

1. Compilar proyecto en Quartus Prime

2. Programar FPGA con archivo .sof
3. Conectar monitor VGA
4. Conectar cable USB-UART

Script Python para transmisión de imagen:

```python
import time
import numpy as np
from PIL import Image
import serial

# Configuración de umbral para conversión RGB111
THRESHOLD = 127

def rgb_to_rgb111_byte(r, g, b, threshold=THRESHOLD):
    """Convierte pixel RGB (0-255) a formato RGB111 (00000RGB)"""
    R = 1 if r > threshold else 0
    G = 1 if g > threshold else 0
    B = 1 if b > threshold else 0
    return (R << 2) | (G << 1) | B

def send_image_uart(image_path, port='COM4', width=640, height=480):
    """Envía imagen por UART a FPGA"""

    # Cargar y redimensionar imagen
    img = Image.open(image_path).convert('RGB')
    img = img.resize((width, height), Image.Resampling.NEAREST)
    arr = np.array(img)

    # Convertir a RGB111 byte por byte
    pixels = []
    for y in range(height):
        for x in range(width):
            r, g, b = arr[y, x]
            byte = rgb_to_rgb111_byte(r, g, b)
            pixels.append(byte)

    flat = np.array(pixels, dtype=np.uint8)
    print(f"Tamaño total: {len(flat)} bytes")

    # Configurar puerto serial
    ser = serial.Serial(
        port=port,
        baudrate=115200,
        bytesize=serial.EIGHTBITS,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_TWO,
        timeout=1
    )

    time.sleep(2)  # Esperar estabilización FT232
```

```
    # Transmitir imagen
    print("Transmitiendo imagen...")
    for value in flat:
        ser.write(bytes([value]))

    ser.close()
    print("Transmisión completada")

# Uso:
# send_image_uart("imagen.jpeg", port="COM4")
```

Resultados:

- ✓ Imagen recibida y mostrada correctamente
- ✓ Cursor visible y móvil
- ✓ Edición de píxeles funcional
- ✓ Selección de color correcta
- ✓ Sin artifacts visuales

---

8. Conclusiones

1. **Integración exitosa:** Se logró implementar un sistema completo que integra comunicación UART, almacenamiento en BRAM, visualización VGA y control interactivo mediante botones.
2. **Robustez del UART:** La implementación del receptor UART con sincronización de 2 etapas y muestreo en el centro del bit demostró ser robusta ante variaciones de timing y ruido.
3. **Eficiencia en cálculo de direcciones:** El uso de operaciones de shift y suma en lugar de multiplicación redujo significativamente el uso de recursos lógicos.
4. **Visualización efectiva del cursor:** La técnica de inversión de color para el cursor parpadeante resultó efectiva para visualización sobre cualquier fondo.
5. **Multiplexado de escritura:** La implementación de prioridad UART > Cursor evitó conflictos de escritura en BRAM y garantizó la integridad de los datos recibidos.
6. **Optimización de memoria:** El uso de BRAM dual-port permitió acceso simultáneo para escritura (UART/cursor) y lectura (VGA) sin degradación de rendimiento.
7. **Cumplimiento de especificaciones:** El sistema cumple con todos los requerimientos del proyecto, soportando resolución 640×480, comunicación a 115200 bps y control interactivo en tiempo real.
8. **Escalabilidad:** La arquitectura modular permite futuras expansiones como soporte para mayor profundidad de color, resoluciones superiores o modos de edición adicionales.

---

9. Referencias

1. Terasic DE1-SoC User Manual, January 2019
2. Intel Quartus Prime Standard Edition Handbook Volume 1: Design and Synthesis
3. Analog Devices ADV7123 Triple 10-Bit High Speed Video DAC Datasheet
4. VESA Display Monitor Timing Standard, Version 1.0, Rev. 0.8
5. Pong P. Chu, "FPGA Prototyping by Verilog Examples", Wiley, 2008
6. Intel Cyclone V Device Handbook Volume 1: Device Interfaces and Integration
7. Serial UART Information, www.lammertbies.nl/comm/info/serial-uart

---

Anexos

Los códigos fuente completos, archivos de simulación y scripts de prueba están disponibles en el repositorio del proyecto.

Fecha de entrega: 1 de diciembre de 2025