# Laboratorio 3: Taller de Diseño Digital
Tecnológico de Costa Rica

Bryan Esquivel Flores
*2020035806*
*zayus@estudiantec.cr*
Alajuela, Costa Rica

Freddy Mora Bolaños
*2021040443*
*freddy.mora@estudiantec.cr*
Alajuela, Costa Rica

Andrés Vargas Arce
*2018151379*
*anbo80@estudiantec.cr*
Alajuela, Costa Rica

November 29, 2024

## Introducción

Este proyecto de laboratorio tiene como objetivo el diseño y desarrollo de un sistema embebido que permita la comunicación de datos entre una FPGA (Nexys4) y un dispositivo secundario para el despliegue de imágenes en un panel LCD. La arquitectura principal está basada en un microcontrolador de 32 bits, implementado con el subconjunto RV32I de RISC-V. El sistema incluye componentes esenciales como memoria ROM y RAM, y se comunica con periféricos como LEDs, interruptores, botones, y dos módulos UART para la transmisión y recepción de datos.

El proyecto se organiza en tres modos de operación: Reposo, donde el sistema espera y realiza sondeo de los periféricos; Almacenamiento, en el que el sistema recibe y almacena imágenes desde una PC; y Desplegar, donde se envían las imágenes almacenadas al dispositivo secundario para su visualización. Además, el desarrollo incluye la implementación de un programa en lenguaje ensamblador, que debe correr indefinidamente y gestionar la comunicación, almacenamiento y despliegue de las imágenes a través de comandos y paquetes de datos.

## 1 Conversor PNG a binario

En esta primera parte, se toma una imagen, cuyo objetivo, es que sea pasada al código RGB, siempre tomando en cuenta el tamaño de la imagen de 135x240 pixeles ya que esta imagen va a tener como fin ser llevada a cabo a la FPGA,dicho

esto, cada pixel va a estar a 16 bits. Además, genera un archivo.txt con los datos recolectados de la imagen, ya que se genera un texto muy extenso. A continuación, se muestra el programa realizado mediante el programa Python:

```python
from PIL import Image
import os

def rgb888_to_rgb565(r, g, b):
    """Convierte un píxel RGB888 a RGB565."""
    r_5 = (r >> 3) & 0x1F  # 5 bits para R
    g_6 = (g >> 2) & 0x3F  # 6 bits para G
    b_5 = (b >> 3) & 0x1F  # 5 bits para B
    return (r_5 << 11) | (g_6 << 5) | b_5

def image_to_rgb565_flat(image_path, width=135, height=240):
    """Convierte una imagen a una lista de valores RGB565 en formato binario continuo."""
    if not os.path.exists(image_path):
        raise FileNotFoundError(f"El archivo no existe: {image_path}")

    try:
        img = Image.open(image_path)
        img = img.convert("RGB")  # Asegura que la imagen esté en RGB
        img = img.resize((width, height))  # Redimensiona a 135x240
        pixels_rgb565 = []

        for y in range(height):
            for x in range(width):
                r, g, b = img.getpixel((x, y))
                rgb565 = rgb888_to_rgb565(r, g, b)
                pixels_rgb565.append(f"{rgb565:016b}")  # Convertir a binario de 16 bits

        return "".join(pixels_rgb565)  # Combinar todo en un solo texto
    except Exception as e:
        print(f"Error al procesar la imagen: {e}")
        return None

# Cambia esta ruta por la ruta correcta a tu imagen
image_path = "C:/Users/Estudiante/Documents/1_azul.png"

try:
    # Convierte la imagen a formato RGB565 con tamaño 135x240 y en texto continuo
    pixels_rgb565_flat = image_to_rgb565_flat(image_path, width=135, height=240)
    print (len(pixels_rgb565_flat))
    if pixels_rgb565_flat:
        # Imprime toda la matriz como texto continuo en binario
        print("Valores RGB565 en binario (como texto continuo):")
        print(pixels_rgb565_flat)

        # Opción para guardar en un archivo
        with open("output_rgb565_135x240_continuo.txt", "w") as file:
            file.write(pixels_rgb565_flat)
        print("El texto binario continuo se ha guardado en 'output_rgb565_135x240_continuo.txt'")
except FileNotFoundError as e:
    print(e)
```

Figura 1: Código de conversión de imagenes

# 2 FPGA principal

En esta sección, se va a tomar los números que están convertidos anteriormente, los cúales van a hacer leidos por una NEXYS4DDR. Teniendo en cuenta que se va a desarrollar un sistema computacional, el cúal va a tener un procesador en RISC-V, estos van a una interfaz de la UART, tomando en cuenta los switches y LEDS correspondientes. A continuación se explicará el código de RISC-V:

1. Se revisa los registros que contienen los valores para enviar o recibir una image.

2. Una vez revisado el dato o código se comenzará el proceso de recibir o enviar. El ciclo encargado de esto es el RECEIVE-IMAGE y SEND-IMAGE respectivamente.

3. En cada caso se envian en la respectiva UART, 64800 bytes que es el total por cada imagen.

La Uart utilizada para la FPGA principal es muy similar a la UART utilizada en la segunda, lo unico que cambia es la implementación con el procesador y las instrucciones RISCV. En este caso se tienen las entradas reg_data, estas controlan el flujo de datos según las entradas del procesador.

```
module UART_custom(
    input clk,
    output ser_tx,
    input  ser_rx,

    output [31:0] uart_c,
    output        uart_r_ready,

    input         reg_dat_we,
    input         reg_dat_re,
    input  [31:0] reg_dat_di,
    output [31:0] reg_dat_do,
    output        reg_dat_wait
);
```

Figura 2: Entradas y salidas UART principal.

# 3 FPGA secundaria

Esta sección va a tomar dos FPGAS, las cúales van a hacer unidos mediante la UART, una FPGA, va a estar conectada a una computadora recibiendo los números binarios, mientras que la otra FPGA va estar conectada a una LCD y esta va a representar el producto final de la imagen dada anteriormente, además de ello se va a tener un panel en la segunda FPGA. A continuación se muestra el código de la UART secundaria:

```
1  module Uart (
2      input logic clk,
3      input logic KeyP,
4
5      input uart_rx,
6      output uart_tx,
7      output logic [7:0] data_out
8  );
```

Figura 3: Entradas y salidas de la UART secundaria

```
module Uart (
    input logic clk,
    input logic KeyP,

    input uart_rx,
    output uart_tx,
    output logic [7:0] data_out
);

localparam DELAY_FRAMES = 868;// 100Mhz / 115200 bauds
localparam HALF_DELAY_WAIT = (DELAY_FRAMES / 2);
//---------------parte que recibe ----- //
reg [3:0] rxState = 0;
reg [12:0] rxCounter = 0;
reg [7:0] dataIn = 0;
reg [2:0] rxBitNumber = 0;
reg byteReady = 0;

localparam RX_STATE_IDLE = 0;
localparam RX_STATE_START_BIT = 1;
localparam RX_STATE_READ_WAIT = 2;
localparam RX_STATE_READ = 3;
localparam RX_STATE_STOP_BIT = 5;

always @(posedge clk) begin
    case (rxState)
        RX_STATE_IDLE: begin
            if (uart_rx == 0) begin
                rxState <= RX_STATE_START_BIT;
                rxCounter <= 1;
                rxBitNumber <= 0;
                byteReady <= 0;
            end
```

Figura 4: Código 1 de la UART secundaria

```
        end
        RX_STATE_START_BIT: begin
            if (rxCounter == HALF_DELAY_WAIT) begin
                rxState <= RX_STATE_READ_WAIT;
                rxCounter <= 1;
            end else
                rxCounter <= rxCounter + 1;
        end
        RX_STATE_READ_WAIT: begin
            rxCounter <= rxCounter + 1;
            if ((rxCounter + 1) == DELAY_FRAMES) begin
                rxState <= RX_STATE_READ;
            end
        end
        RX_STATE_READ: begin
            rxCounter <= 1;
            dataIn <= {uart_rx, dataIn[7:1]};
            rxBitNumber <= rxBitNumber + 1;
            if (rxBitNumber == 3'b111)
                rxState <= RX_STATE_STOP_BIT;
            else
                rxState <= RX_STATE_READ_WAIT;
        end
        RX_STATE_STOP_BIT: begin
            rxCounter <= rxCounter + 1;
            if ((rxCounter + 1) == DELAY_FRAMES) begin
                rxState <= RX_STATE_IDLE;
                rxCounter <= 0;
                byteReady <= 1;
```

Figura 5: Código 2 de la UART secundaria

```
                end
            end
        endcase
    end

    always @(posedge clk) begin
        if (byteReady) begin
            data_out <= dataIn;
        end
    end



    //--------------------Parte que envia----------------//
    reg [3:0] txState = 0;// estado actual
    reg [24:0] txCounter = 0;// ciclos de reloj
    reg [7:0] dataOut = 0;// byte que se envia
    reg txPinRegister = 1;// valor a adjuntar en uart_tx
    reg [2:0] txBitNumber = 0;// bit enviando
    reg [3:0] txByteCounter = 0;

    assign uart_tx = txPinRegister;

    // parte a cambiar
    localparam MEMORY_LENGTH = 1;
    reg [7:0] code_uart = 2'b10 ;
```

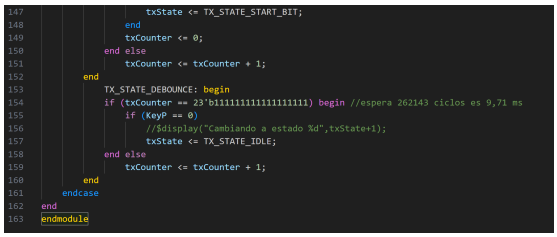
Figura 6: Código 3 de la UART secundaria

```
    /*
    initial begin
        code_uart[0] = "Hola";
    end
    */

    localparam TX_STATE_IDLE = 0;
    localparam TX_STATE_START_BIT = 1;
    localparam TX_STATE_WRITE = 2;
    localparam TX_STATE_STOP_BIT = 3;
    localparam TX_STATE_DEBOUNCE = 4;
    always @(posedge clk)begin
        case (txState)

            TX_STATE_IDLE: begin

                if (KeyP == 1) begin //cuando se presiona una tecla
                    txState <= TX_STATE_START_BIT;
                    //$display("Cambiando a estado %d",txState+1);
                    txCounter <= 0;
                end
                else begin
                    txPinRegister <= 1;
                end
            end
            TX_STATE_START_BIT: begin
                txPinRegister <= 0;
                if ((txCounter + 1) == DELAY_FRAMES) begin //espera 312us
                    //$display("Cambiando a estado %d",txState+1);
                    txState <= TX_STATE_WRITE;
                    dataOut <= code_uart[txByteCounter];// guardo el caracter
                    txBitNumber <= 0;
```

Figura 7: Código 4 de la UART secundaria

```
                    txCounter <= 0;
                end else
                    txCounter <= txCounter + 1;
            end
            TX_STATE_WRITE: begin
                txPinRegister <= dataOut[txBitNumber];// envia el bit
                if ((txCounter + 1) == DELAY_FRAMES) begin//espera 312us
                    if (txBitNumber == 3'b111) begin // cuenta hasta 8bits
                        txState <= TX_STATE_STOP_BIT;
                        //$display("Cambiando a estado %d",txState+1);
                    end else begin
                        txState <= TX_STATE_WRITE;
                        txBitNumber <= txBitNumber + 1;
                    end
                    txCounter <= 0;
                end else
                    txCounter <= txCounter + 1;
            end
            TX_STATE_STOP_BIT: begin
                txPinRegister <= 1;
                if ((txCounter + 1) == DELAY_FRAMES) begin// espera 312us
                    if (txByteCounter == MEMORY_LENGTH - 1) begin
                        //$display("Cambiando a estado %d",txState+1);
                        txState <= TX_STATE_DEBOUNCE;
                    end else begin
                        txByteCounter <= txByteCounter + 1;
```

Figura 8: Código 5 de la UART secundaria



```
                    txState <= TX_STATE_START_BIT;
                end
                txCounter <= 0;
            end else
                txCounter <= txCounter + 1;
        end
        TX_STATE_DEBOUNCE: begin
        if (txCounter == 23'b11111111111111111) begin //espera 262143 ciclos es 9,71 ms
            if (KeyP == 0)
                //$display("Cambiando a estado %d",txState+1);
                txState <= TX_STATE_IDLE;
        end else
            txCounter <= txCounter + 1;
    end
    endcase
end
endmodule
```

Figura 9: Código 6 de la UART secundaria