

PROYECTO PROTOCOLO UART FULL-DUPLEX

IE-0523: CIRCUITOS
DIGITALES II

ISAÍ GONZÁLEZ S.
DANIEL DE LA O R.
JAVIER SACA

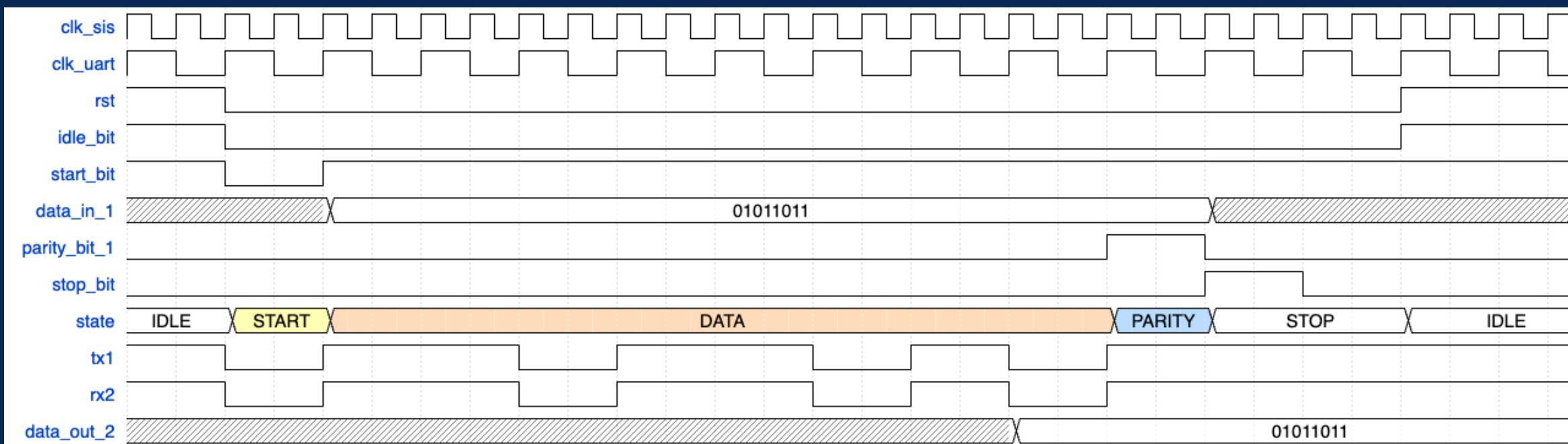
INDICE DE CONTENIDOS

```
4  # Prevent database truncation if the transaction fails
5  abort("The Rails environment is running in production mode!
6  require 'spec_helper'
7  require 'rspec/rails'
8
9  require 'capybara/rspec'
10 require 'capybara/rails'
11
12 Capybara.javascript_driver = :webkit
13 Category.delete_all; Category.create!(name: "Electronics")
14 Shoulda::Matchers.configure do |config|
15   config.integrate do |sp|
16     sp.with.test_framework :rspec
17     sp.with.library :rails
18   end
19 end
20
21 # Add additional requires below this line if you need them
22
23 # Requires supporting ruby files with custom matchers
24 # in ./support and its subdirectories. This directory also
25 # contains a script that provides the convenience of
26 # running rspec with --tag=feature.
27 # in _spec.rb will both be required by default. If you do not
28 # want the convenience of running via RSpec's #run, consider
29 # ending with ___. You can configure this pattern with
30 # --tag :unit, :integration, or :feature
31
32 # Not found for 'mongoid'
33
```

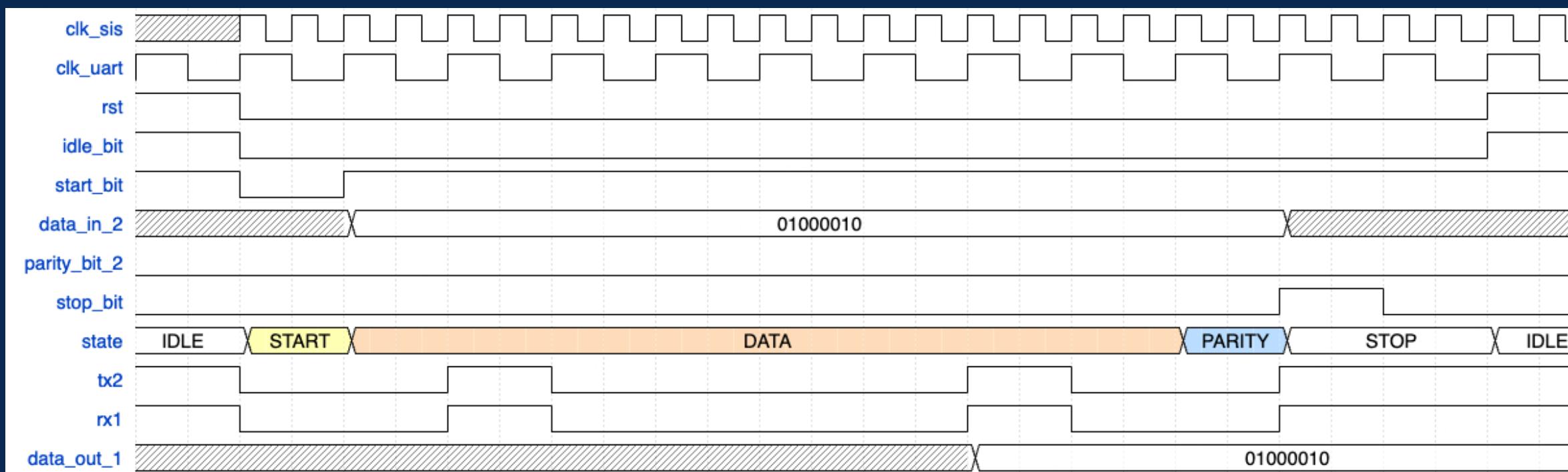
1. UART1 (TRANSMISSOR)	3
2. UART2 (RECEIVER)	5
3. WAVEFORM GTKWAVE	7
	9
	11

/ WAVEDROAM:

DATA_IN 1



DATA_IN 2



1 /
UART1(TRANSMISSOR)

1/ UART1

```
module UART1_dut(
    input wire clk_uart,           // Reloj del sistema
    input wire clk_sis,            // Reloj para las operaciones UART
    input wire rst,                // Reset
    input wire start_bit,          // Indica el inicio de la transmisión
    input wire [7:0] data_in_1,     // Dato para paridad y transmisión en tx1
    input wire stop_bit,           // Indica el fin de la transmisión
    input wire rx1,                // Línea de recepción serie desde UART2
    output reg tx1,                // Línea de transmisión serie hacia UART2
    output reg [7:0] data_out_1    // Salidad paralela desde UART2
);

// Declaración de los estados del transmisor UART
localparam IDLE = 3'b000,
    START = 3'b001,
    DATA = 3'b010,
    PARITY = 3'b011,
    STOP = 3'b100;

reg [2:0] state, next_state;        // Estado actual y siguiente
reg idle_bit;                      // Bandera para indicar si el sistema está en reposo
reg [3:0] cnt_bits, next_cnt_bits; // Contador para rastrear los bits transmitidos
reg parity_bit_1;                  // Almacena el bit de paridad calculado

// Lógica Secuencial
always @(posedge clk_uart or posedge rst) begin
    if (rst) begin
        // Reinicio a estado inicial
        state      <= IDLE;           // Comienza en el estado IDLE
        tx1       <= 1'b1;             // Línea en alto (reposo)
        parity_bit_1 <= 1'b0;          // Inicializa el bit de paridad
        cnt_bits    <= 4'b0;           // Inicializa el contador de bits
        idle_bit   = 1;                // Inicializa el idle_bit en alto
    end else begin
        state      <= next_state;
        cnt_bits   <= next_cnt_bits;
    end
end

```

```
// Máquina de estados
// Se va armando la señal de tx1.
case (state)
    IDLE: begin
        idle_bit = 1;
        tx1 = 1'b1;
        if (idle_bit)
            next_state = START;
    end

    START: begin
        idle_bit = 0;
        tx1 = 1'b0;
        next_state = DATA;
    end

    DATA: begin
        // Transmite los bits de datos en serie uno por uno
        idle_bit = 0;
        tx1 = data_in_1[cnt_bits];    // Selecciona el bit correspondiente LSB
        next_cnt_bits = cnt_bits + 1;
        data_out_1[0 + cnt_bits] = rx1;
        if (cnt_bits == 7)           // Si todos los bits han sido transmitidos
            next_state = PARITY;
    end

    PARITY: begin
        idle_bit = 0;
        parity_bit_1 = ^data_in_1;    // Calcula la paridad XOR de los datos
        tx1 = parity_bit_1;          // Transmite el bit de paridad
        next_state = STOP;           // Cambia al estado de parada
    end

    STOP: begin
        idle_bit = 0;
        tx1 = 1'b1;                  // Transmite el bit de parada (alto)
        if (stop_bit)
            next_state = IDLE;
    end
endcase

```

2 / U A R T 2
(RECEIVER)

2 / UART 2

```
module UART2_dut(
    input wire clk_uart,           // Reloj del sistema
    input wire clk_sis,            // Reloj para las operaciones UART
    input wire rst,                // Reset
    input wire start_bit,          // Indica el inicio de la transmisión
    input wire [7:0] data_in_2,     // Dato para paridad y transmisión en tx2
    input wire stop_bit,            // Indica el fin de la transmisión
    input wire rx2,                // Línea de recepción serie desde UART1
    output reg tx2,                // Línea de transmisión serie hacia UART1
    output reg [7:0] data_out_2    // Salidad paralela desde UART1
);

// Declaración de los estados del transmisor UART
localparam IDLE = 3'b000,
          START = 3'b001,
          DATA = 3'b010,
          PARITY = 3'b011,
          STOP = 3'b100;

reg [2:0] state, next_state;        // Estado actual y siguiente
reg idle_bit;                      // Bandera para indicar si el sistema está en reposo
reg [3:0] cnt_bits, next_cnt_bits; // Contador para rastrear los bits transmitidos
reg parity_bit_2;                  // Almacena el bit de paridad calculado

// Lógica Secuencial
always @(posedge clk_uart or posedge rst) begin
    if (rst) begin
        // Reinicio a estado inicial
        state      <= IDLE;           // Comienza en el estado IDLE
        tx2        <= 1'b1;            // Línea en alto (reposo)
        parity_bit_2 <= 1'b0;          // Inicializa el bit de paridad
        cnt_bits    <= 4'b0;            // Inicializa el contador de bits
        idle_bit   = 1;                // Inicializa el idle_bit en alto
    end else begin
        state      <= next_state;
        cnt_bits   <= next_cnt_bits;
    end
end

```

```
// Máquina de estados
// Se va armando la señal de tx2.
case (state)
    IDLE: begin
        idle_bit = 1;
        tx2 = 1'b1;
        if (idle_bit)
            next_state = START;
    end

    START: begin
        idle_bit = 0;
        tx2 = 1'b0;
        next_state = DATA;
    end

    DATA: begin
        // Transmite los bits de datos en serie uno por uno
        idle_bit = 0;
        tx2 = data_in_2[cnt_bits];    // Selecciona el bit correspondiente LSB
        next_cnt_bits = cnt_bits + 1;
        data_out_2[0 + cnt_bits] = rx2;
        if (cnt_bits == 7)           // Si todos los bits han sido transmitidos
            next_state = PARITY;
    end

    PARITY: begin
        idle_bit = 0;
        parity_bit_2 = ^data_in_2;    // Calcula la paridad XOR de los datos
        tx2 = parity_bit_2;          // Transmite el bit de paridad
        next_state = STOP;           // Cambia al estado de parada
    end

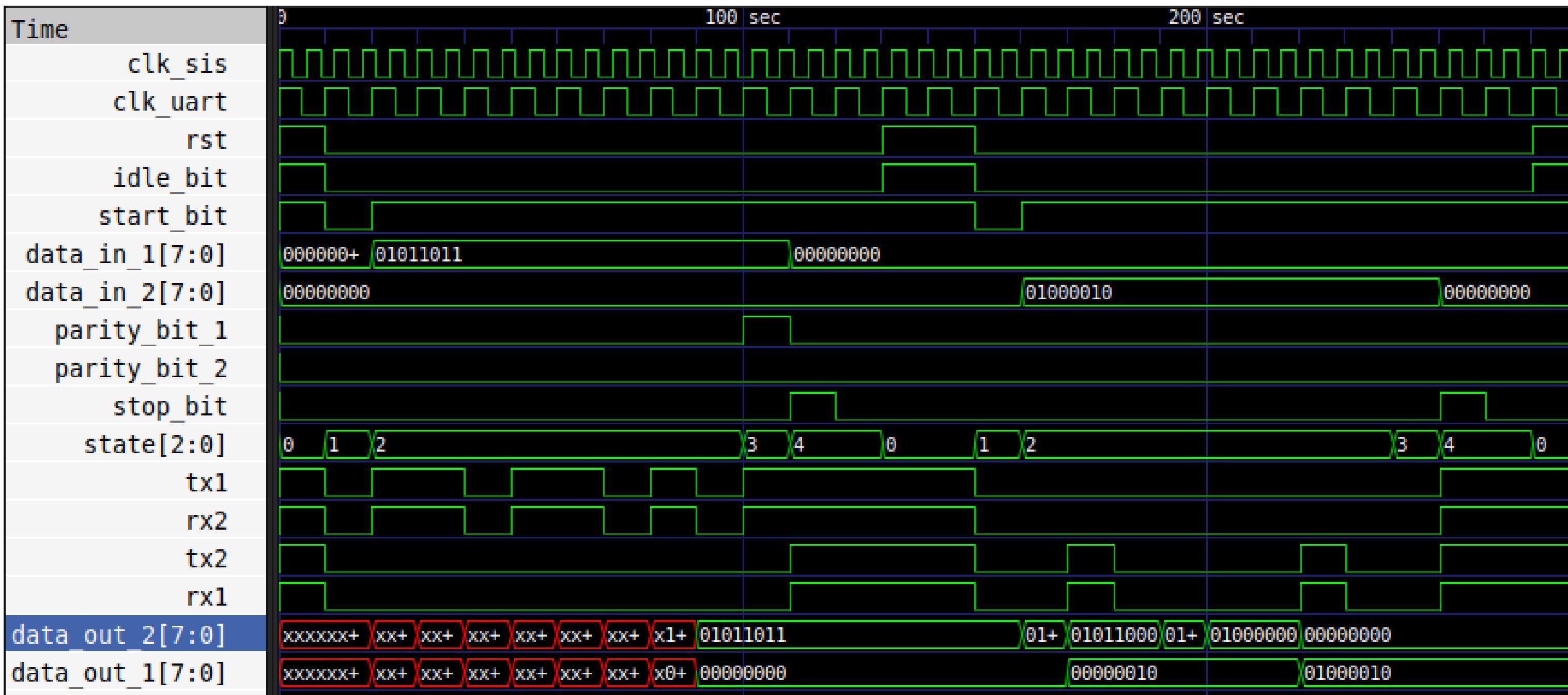
    STOP: begin
        idle_bit = 0;                // Transmite el bit de parada (alto)
        tx2 = 1'b1;
        if (stop_bit)
            next_state = IDLE;
    end
endcase

```

3 / WAVEFORM

GTKWAVE

3 / WAVEFORM



\TESTBENCH

```
'include "UART_dut.v"
`include "UART_tester.v"

module UART_tb;
    wire clk_uart;
    wire clk_sis;
    wire rst;
    wire start_bit;
    wire [7:0] data_in_1;
    wire [7:0] data_in_2;
    wire stop_bit;
    wire tx2;
    wire tx1;
    wire rx2;
    wire rx1;
    wire serial1;           // Cable de conexión entre tx1 y rx2
    wire serial2;           // Cable de conexión entre tx2 y rx1

    UART1_dut dut1 (
        .clk_sis(clk_sis),
        .clk_uart(clk_uart),
        .rst(rst),
        .start_bit(start_bit),
        .data_in_1(data_in_1),
        .stop_bit(stop_bit),
        .tx1(serial1),        // Conexión entre UARTs
        .rx1(serial2)         // Conexión entre UARTs
    );

    UART2_dut dut2 (
        .clk_sis(clk_sis),
        .clk_uart(clk_uart),
        .rst(rst),
        .start_bit(start_bit),
        .data_in_2(data_in_2),
        .stop_bit(stop_bit),
        .tx2(serial2),        // Conexión entre UARTs
        .rx2(serial1)         // Conexión entre UARTs
    );

    UART_tester tester (
        .clk_sis(clk_sis),
        .clk_uart(clk_uart),
        .rst(rst),
        .start_bit(start_bit),
        .data_in_1(data_in_1),
        .data_in_2(data_in_2),
        .stop_bit(stop_bit)
    );

```

MUCHAS GRACIAS

```
    render() {
      return (
        <React.Fragment>
          <div className="py-5">
            <div className="container">
              <Title name="our" title="product">
              <div className="row">
                <ProductConsumer>
                  {(value) => {
                    |   |   |   console.log(value)
                  }}
                </ProductConsumer>
              </div>
            </div>
          </div>
        <React.Fragment>
```