UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Camila Xavier de Medeiros (cxm)
Lorenzo Fontenelle Chaves (lfc4)
Maria Luísa Mendes de Siqueira Passos (mlmsp)
Mário da Mota Limeira Neto (mmln)

PROJETO DE SISTEMAS DIGITAIS DE COMUNICAÇÃO UART

RECIFE, PERNAMBUCO
2021

Sumário

1. **Introdução**

Neste relatório iremos apresentar e explicar, detalhadamente, o projeto da segunda unidade da disciplina Sistemas Digitais, que consiste na aplicação dos assuntos explorados em aula, através da implementação de módulos de transmissão e recepção compatíveis com o protocolo UART (Universal asynchronous receiver/transmitter).

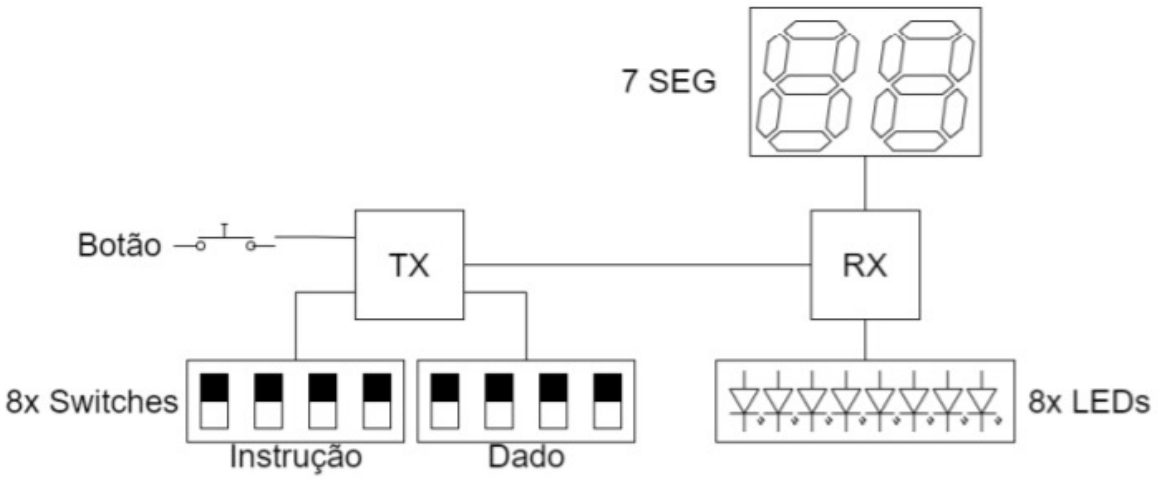
O mesmo consiste na implementação de dois módulos, TX e RX, responsáveis por transmitir e receber dados, respectivamente. Tais dados são inseridos por 8 switches que indicam valores e instruções a serem realizadas pelo sistema e devem ser apresentados em 2 displays de 7 segmentos e 8 LEDs.

Todos os circuitos, códigos e diagramas apresentados neste relatório foram implementados, testados e simulados no Quartus. Além disso, também implementamos máquinas de estado dos módulos RX e TX, para melhor compreensão do projeto.

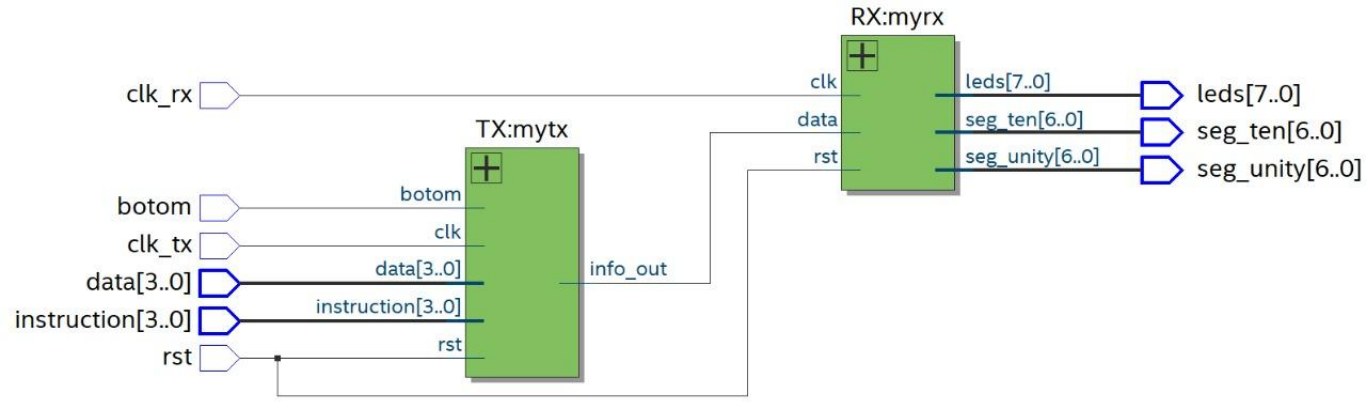
2. Visão Geral do Projeto

2.1. Instruções e visão geral em um diagrama de blocos do projeto:

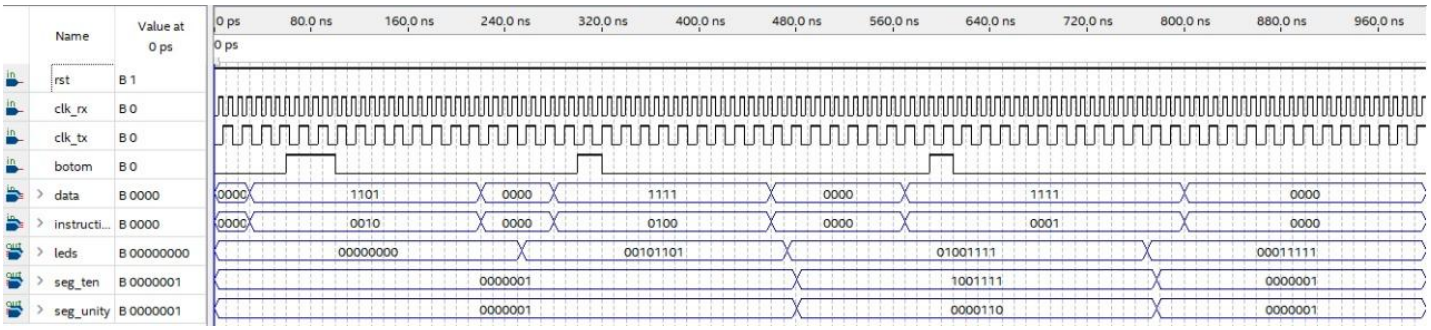
Instruções:
- 0x01 - Limpar os displays de 7 segmentos;
- 0x02 - Carregar dados até em um registrador interno no módulo RX;
- 0x04 - Mostrar os dados armazenados no registrador interno nos displays de 7 segmentos;



2.2. Diagrama de blocos do UART (com respectivas entradas e saídas)



2.3. Waveform do UART



2.4. Código do UART

```
module uart (
input clk_rx, //clock do rx
input clk_tx, //clock do tx
input rst, //reset
input botom, //botao
input [3:0] data, //dado
input [3:0] instruction, //instruçao
output [7:0] leds, //os 8 leds
output [6:0] seg_unity, //unidades do led de 7 segmentos
output [6:0] seg_ten //dezenas do led de 7 segmentos
);

wire tx_to_rx;

   TX mytx(
      .clk (clk_tx),
      .rst (rst),
      .data (data),
      .instruction (instruction),
      .botom (botom),
      .info_out (tx_to_rx)
);

   RX myrx(
      .data (tx_to_rx), //output do tx
      .clk (clk_rx),
      .rst (rst),
      .seg_unity (seg_unity),
      .seg_unity (seg_unity),
      .seg_ten (seg_ten),
      .leds (leds),
      .state ()
);

endmodule
```

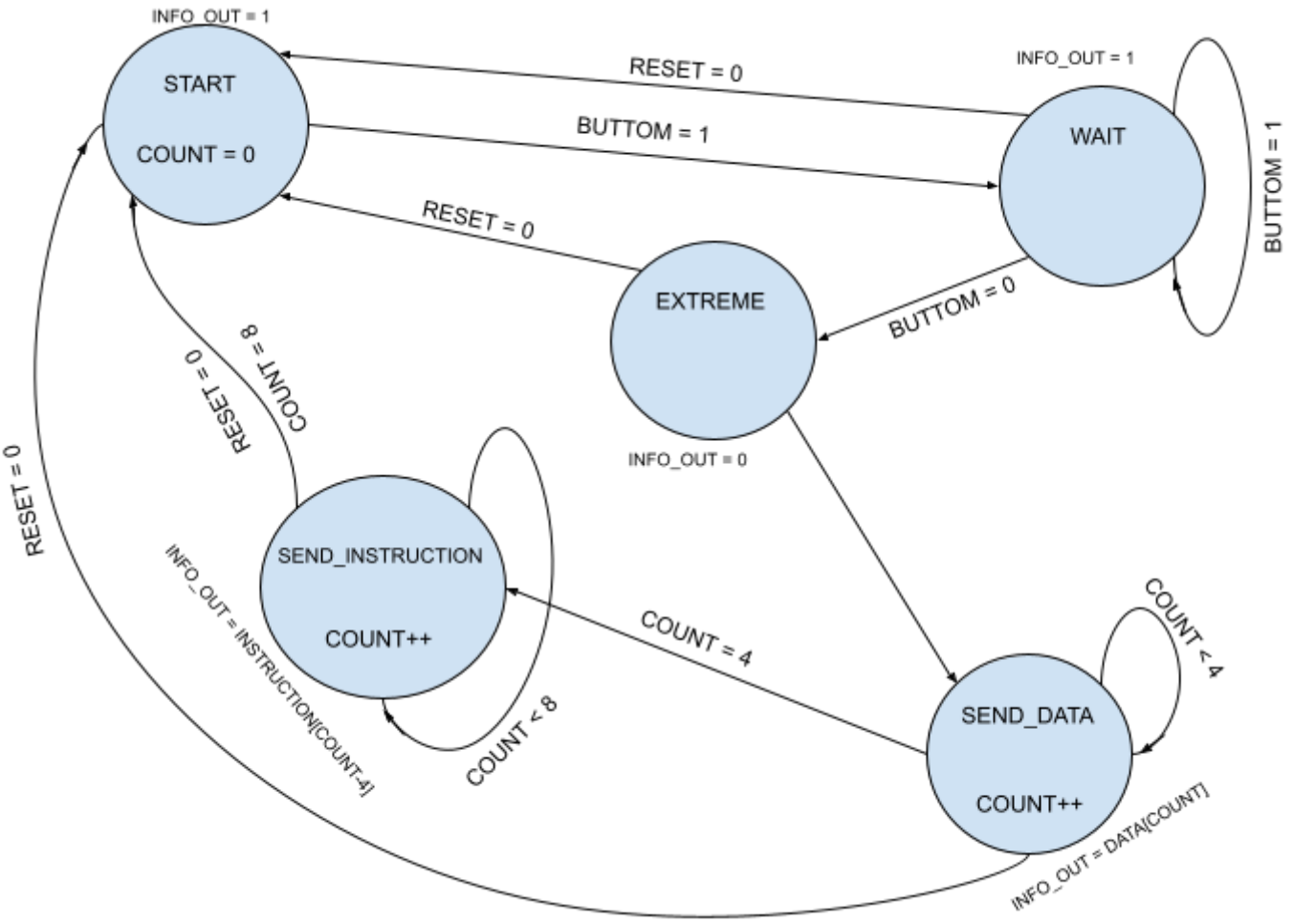
3. Módulo TX

3.1. Máquina de estados do TX

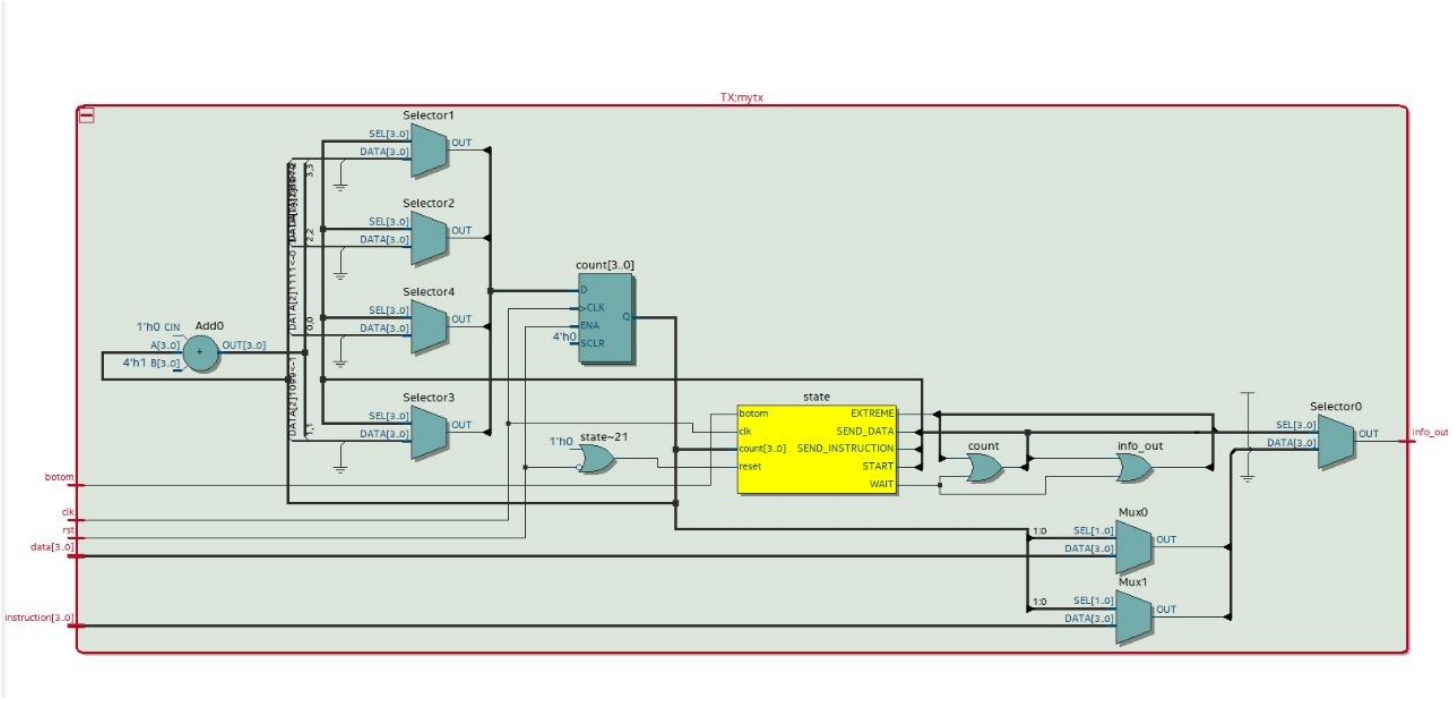
Esse módulo conta com uma máquina com quatro estados, sendo eles:

- START
- WAIT
- EXTREME
- SEND_DATA
- SEND_INSTRUCTION

A máquina sairá do estado de START quando o botão for ativado e entrará no estado de WAIT. Após o botão ser desativado passará para o estado EXTREME, quando as informações de saída vão ser zeradas e seguirá para os estados SEND_DATA, momento no qual as informações de saída serão contabilizadas. Por fim, a máquina partirá para o último estado, o SEND_INSTRUCTION, e as informações que dizem respeito às instruções serão contabilizadas, enviadas e esse ciclo será reiniciado. Caso o reset seja acionado a máquina retornará para o estado START.



3.2. Diagrama de blocos do TX



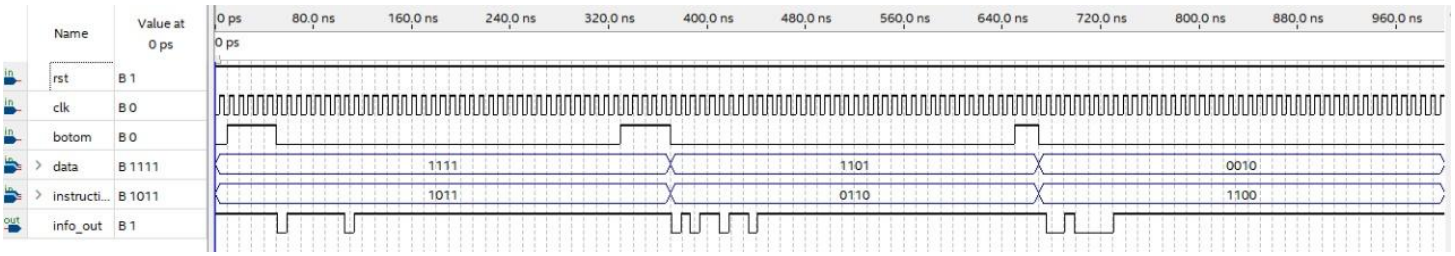
3.3. Código do TX

```
module TX(clk, rst, botom, instruction, data, info_out);

    input [3:0]data;
    input [3:0]instruction;
    input botom;
    input clk, rst;
    output reg info_out;

    reg [3:0]count;
    reg [2:0]state;

    parameter START = 3'b000,
              WAIT = 3'b001,
              EXTREME = 3'b010,
              SEND_DATA = 3'b011,
              SEND_INSTRUCTION = 3'b100;

    initial begin
        info_out <= 1'd1;
        state <= START;
        count <= 4'd0;
end

    always@(*) begin
        case(state)
            START: begin
                info_out <= 1;
            end
            WAIT: begin
                info_out <= 1;
            end
            EXTREME: begin
                info_out <= 0;
            end
            SEND_DATA: begin
                info_out <= data[count];
            end
            SEND_INSTRUCTION: begin
                info_out <= instruction[count - 4];
            end
            default: begin
                info_out <= info_out;
            end

        endcase

    end

    always@(posedge clk or negedge rst) begin

        if(rst == 0) begin
            state <= START;
        end
```

```
56          else begin
57              case(state)
58                  START: begin
59                      count <= 0;
60                      if(botom == 1) begin
61                          state <= WAIT;
62                      end
63                  end
64                  WAIT: begin
65                      if(botom == 0) begin
66                          state <= EXTREME;
67                      end
68                  end
69                  EXTREME: begin
70                      state <= SEND_DATA;
71                  end
72                  SEND_DATA: begin
73                      count <= count + 1;
74                          if(count == 3) begin
75                              state <= SEND_INSTRUCTION;
76                          end
77                  end
78                  SEND_INSTRUCTION: begin
79                      count <= count + 1;
80                          if(count == 7) begin
81                              state <= START;
82                          end
```

```
83                  end
84                  default: begin
85                      state = state;
86                  end
87
88              endcase
89
90          end
91
92      end
93
94  endmodule
95
```

3.4. Waveform do TX

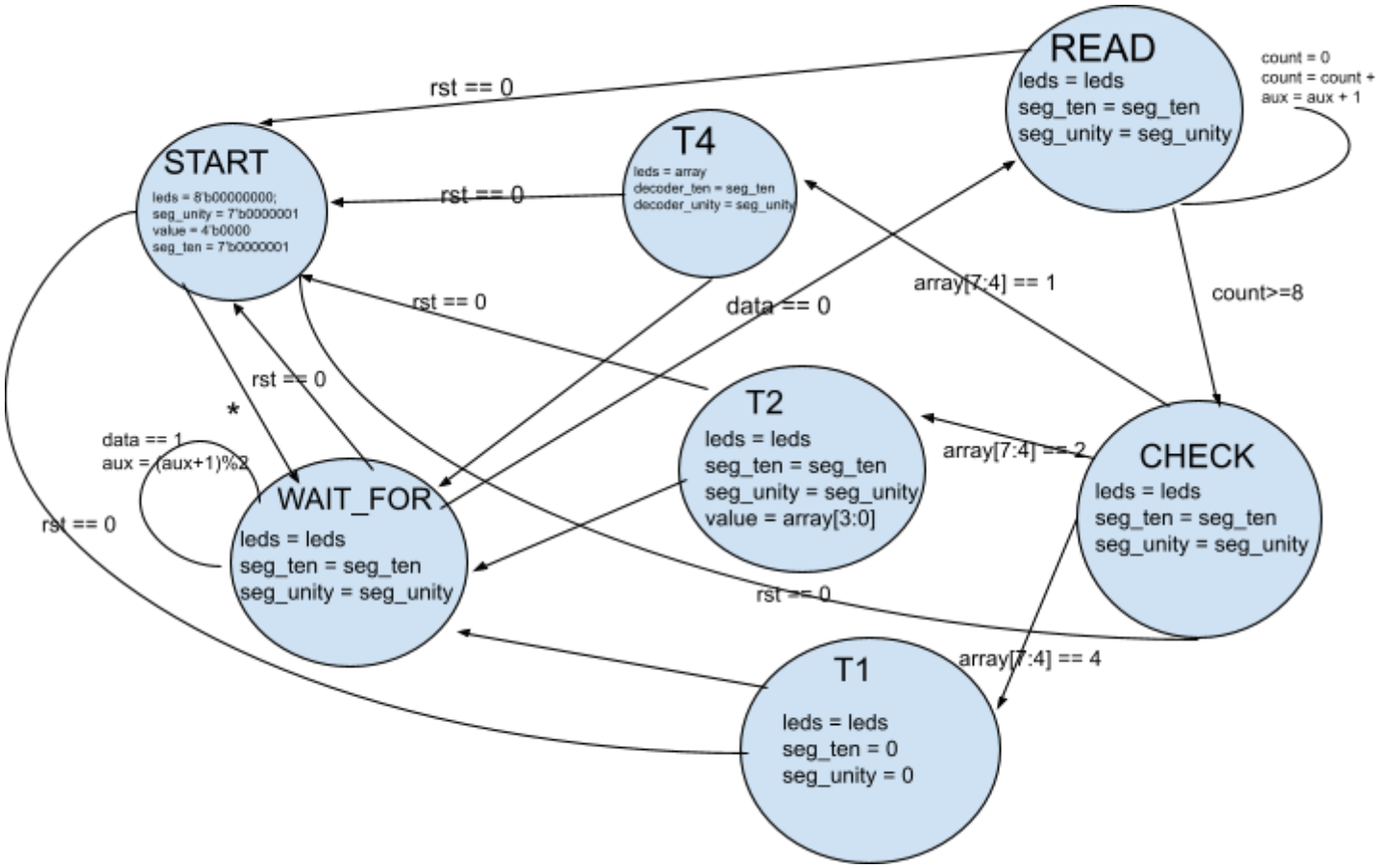


4. Módulo RX

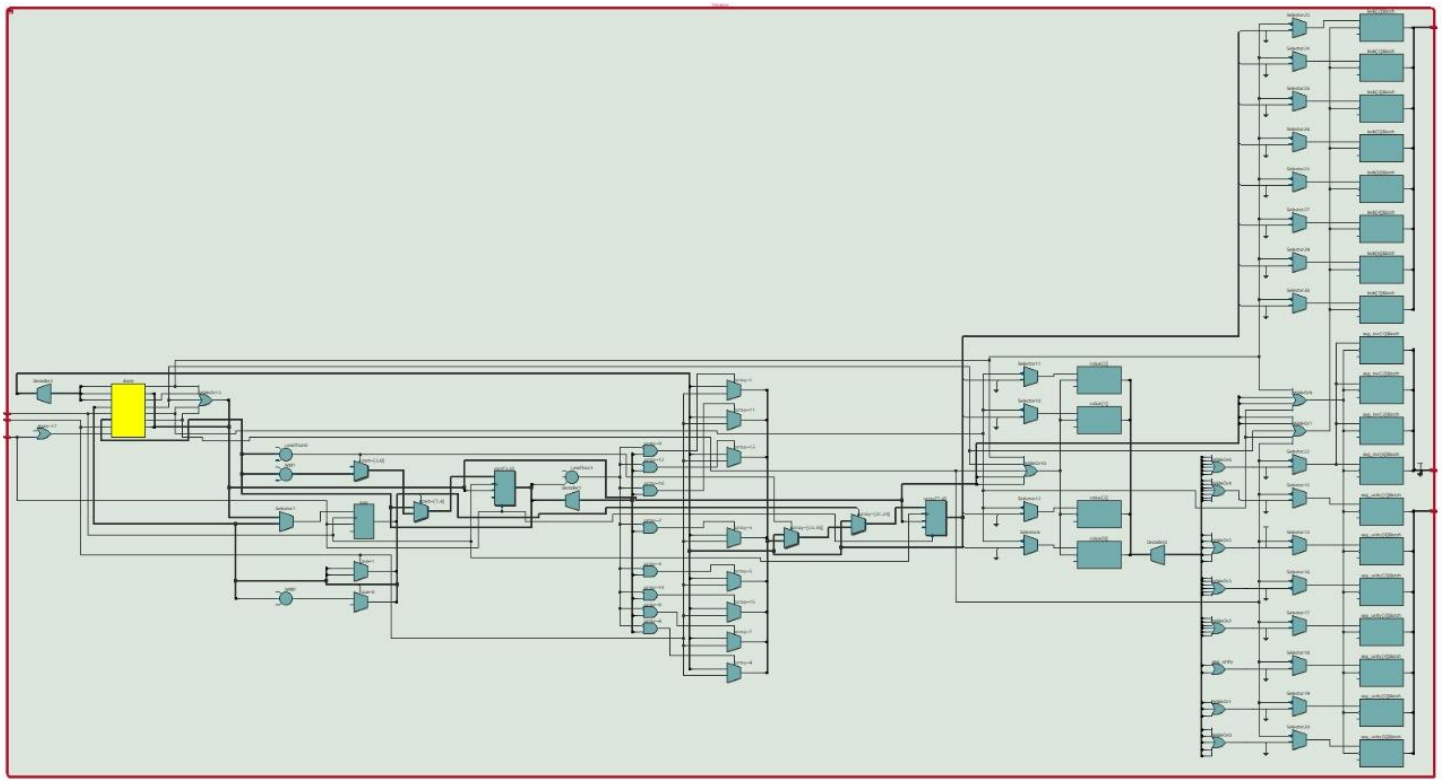
4.1. Máquina de estados do RX

A máquina conta com 7 estados que denominamos: START, WAIT_FOR, READ, CHECK, T1, T2 e T4.

A princípio, a máquina sai do estado START para o estado WAIT_FOR, recebe os dados, indo então para o estado READ, no qual fará a devida interpretação das informações. Então, haverá uma análise das informações no estado CHECK, que dependendo da instrução recebida será encaminhada para o devido estado. A limpeza de dados ocorrerá nos estados WAIR_FOR e START. A máquina voltará para seu estado inicial (START) se houver reset.



4.2. Diagrama de blocos do RX



4.3. Código do RX

```
1   module RX( input data, input clk,input rst, output reg[6:0] seg_unity,output //RX
2
3   reg[6:0] seg_ten, output reg[7:0] leds);
4    reg [2:0] state;
5    reg [3:0] value;
6    reg [7:0] array = 8'b00000000;
7    reg aux = 0;
8    reg [3:0] cont = 4'b0000;
9
10   parameter start = 3'b000, wait_for = 3'b001, read = 3'b010, check =
11   3'b011, t1 = 3'b100, t2 = 3'b101, t4 = 3'b110;
12
13
14  always@(*) begin
15     case(state)
16        start: begin
17           leds <= 8'b00000000;
18           seg_ten <= 7'b0000001;
19           seg_unity <= 7'b0000001;
20           value <= 4'b0000;
21        end
22        wait_for: begin
23           leds <= leds;
24           seg_ten <= seg_ten;
25           seg_unity <= seg_unity;
26        end
27        read: begin
28           leds <= leds;
```

```
29          seg_ten <= seg_ten;
30          seg_unity <= seg_unity;
31      end
32      check: begin
33          leds <= array;
34          seg_ten <= seg_ten;
35          seg_unity <= seg_unity;
36      end
37      t1: begin //limpa
38          leds <= leds;
39          seg_ten <= 7'b0000001;
40          seg_unity <= 7'b0000001;
41      end
42      t2: begin //carrega
43          value <= array[3:0];
44          leds <= leds;
45          seg_ten <= seg_ten;
46          seg_unity <= seg_unity;
47      end
48      t4: begin // mostra
49          leds <= leds;
50          case(value)
51              4'd0: begin
52                  seg_unity[6:0] <= 7'b0000001; // 0
53                  seg_ten[6:0] <= 7'b0000001;
54              end
```

```
55              4'd1: begin
56                  seg_unity[6:0] <= 7'b1001111; // 1
57                  seg_ten[6:0] <= 7'b0000001;
58              end
59              4'd2: begin
60                  seg_unity[6:0] <= 7'b0010010; // 2
61                  seg_ten[6:0] <= 7'b0000001;
62              end
63              4'd3: begin
64                  seg_unity[6:0] <= 7'b0000110; // 3
65                  seg_ten[6:0] <= 7'b0000001;
66              end
67              4'd4: begin
68                  seg_unity[6:0] <= 7'b1001100; // 4
69                  seg_ten[6:0] <= 7'b0000001;
70              end
71              4'd5: begin
72                  seg_unity[6:0] <= 7'b0100100; // 5
73                  seg_ten[6:0] <= 7'b0000001;
74              end
75              4'd6: begin
76                  seg_unity[6:0] <= 7'b1100000; // 6
77                  seg_ten[6:0] <= 7'b0000001;
78              end
```

```
79              4'd7: begin
80                 seg_unity[6:0] <= 7'b0001111; // 7
81                 seg_ten[6:0] <= 7'b0000001;
82              end
83              4'd8: begin
84                 seg_unity[6:0] <= 7'b0000000; // 8
85                 seg_ten[6:0] <= 7'b0000001;
86              end
87              4'd9: begin
88                 seg_unity[6:0] <= 7'b0001100; // 9
89                 seg_ten[6:0] <= 7'b0000001;
90              end
91              4'd10: begin
92                 seg_unity[6:0] <= 7'b0000001; // 10
93                 seg_ten[6:0] <= 7'b1001111;
94              end
95              4'd11: begin
96                 seg_unity[6:0] <= 7'b1001111; // 11
97                 seg_ten[6:0] <= 7'b1001111;
98              end
99              4'd12: begin
100                seg_unity[6:0] <= 7'b0010010; // 12
101                seg_ten[6:0] <= 7'b1001111;
102             end
103             4'd13: begin
104                seg_unity[6:0] <= 7'b0000110; // 13
105                seg_ten[6:0] <= 7'b1001111;
106             end
```

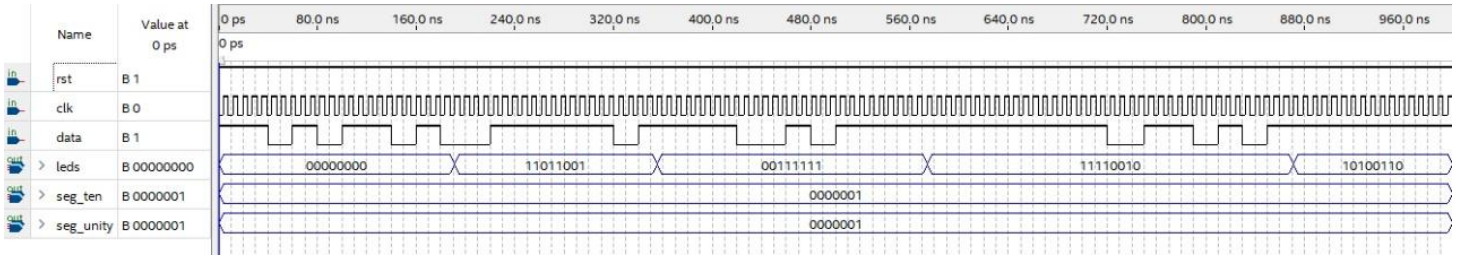
```
107             4'd14: begin
108                seg_unity[6:0] <= 7'b1001100; // 14
109                seg_ten[6:0] <= 7'b1001111;
110             end
111             4'd15: begin
112                seg_unity[6:0] <= 7'b0100100; // 15
113                seg_ten[6:0] <= 7'b1001111;
114             end
115          endcase
116       end
117    endcase
118 end
119
120 always@(posedge clk or negedge rst) begin
121    if(rst == 0) begin
122       state <= start;
123       array <= 0;
124       cont <= 0;
125    end
126    else begin
127       case(state)
128          start: begin
129             state <= wait_for;
130          end
131          wait_for: begin
132             if(data == 0) begin
133                aux <= aux + 1'd1;
```

```
133                 aux <= aux + 1'd1;
134                 if(aux == 1) begin
135                     aux <= 0;
136                     state <= read; //start point
137                 end
138             end
139             else begin
140                 state <= wait_for;
141             end
142         end
143         read: begin
144             aux <= aux + 1'd1;
145             if(aux == 1) begin
146                 aux <= 0;
147                 if(cont < 8) begin
148                     array[cont] <= data;
149                     cont <= cont + 4'b0001;
150                     state <= read;
151                 end
152                 else begin
153                     cont <= 0;
154                     state <= check;
155                 end
156             end
157         end
158         check: begin
159             case(array[7:4])
160                 4'd1: begin // limpar dados
```

```
160                 4'd1: begin // limpar dados
161                     state <= t1;
162                 end
163                 4'd2: begin // salvar dados
164                     state <= t2;
165                 end
166                 4'd4: begin // mostrar dados
167                     state <= t4;
168                 end
169                 default: begin // fazer nada
170                     state <= wait_for;
171                 end
172             endcase
173         end
174         t1: begin
175             state <= wait_for;
176         end
177         t2: begin
178             state <= wait_for;
179         end
180         t4: begin
181             state <= wait_for;
182         end
183         default: begin
184             state <= wait_for;
185         end
186     endcase
```

```
186     endcase
187   end
188 end
189 endmodule
```

4.4. Waveform do RX



5. Conclusão

Com a finalização deste projeto UART e a utilização da linguagem Verilog, nossa equipe teve a oportunidade de aplicar os conhecimentos vistos em aula de maneira tal que conseguimos ver como circuitos digitais básicos que existem no nosso cotidiano realmente funcionam. O projeto conta com a divisão em módulos, que foram testados na plataforma Quartus e explicados individualmente, sendo, então, consolidado o funcionamento do trabalho como um todo.

Logo, concluímos que há uma grande importância em conhecer o funcionamento de uma UART e suas aplicações para o desenvolvimento de um profissional da área de tecnologia. Ainda, pudemos observar, na prática, como são suas waveforms (desenvolvidas no Quartus). Dessa forma, esse trabalho nos forneceu não só conhecimento teórico, mas também prático e usual.