UNIVERSIDADE FEDERAL DE PERNAMBUCO
CIN-CENTRO DE INFORMÁTICA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

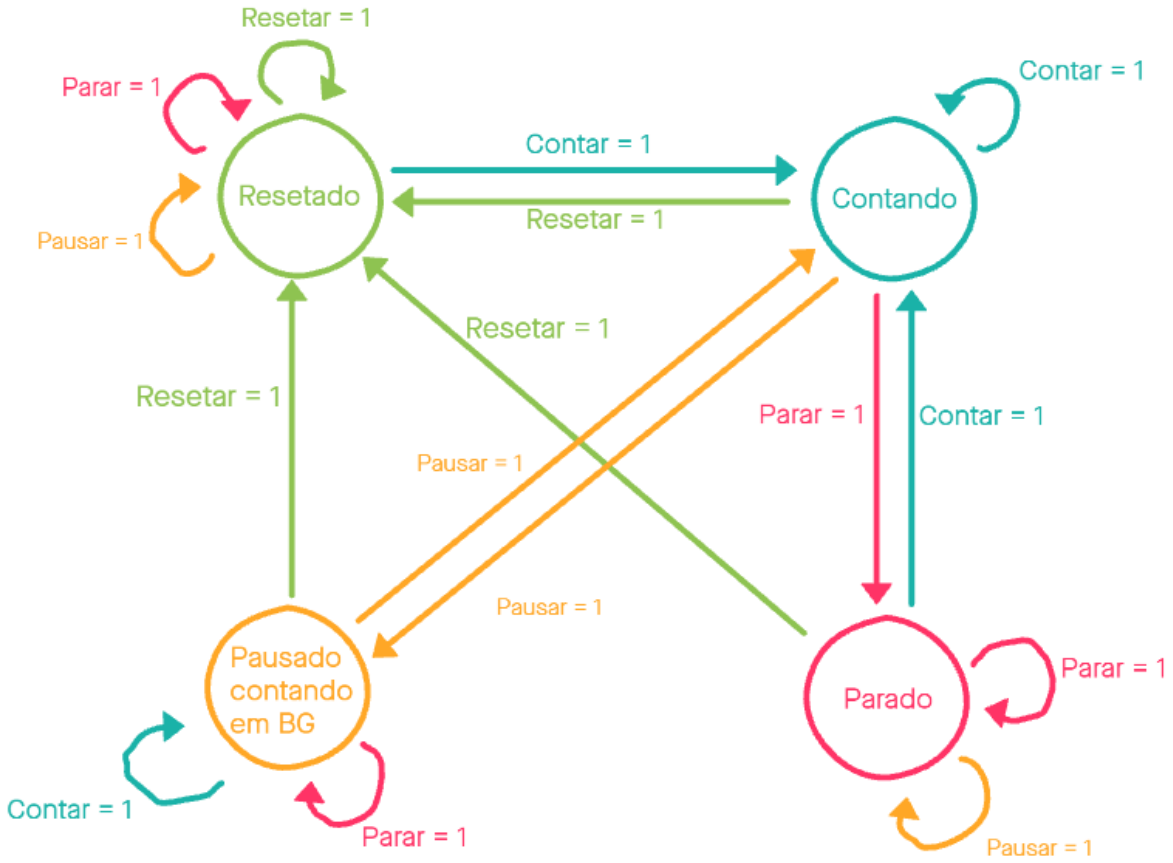
Gabriel de Oliveira Pessoa (gop2)
Lorena Carla Jordão Braga Vilaça (lcjbv2)
Vinicius Torres de Macedo (vtm)

Projeto de Sistemas Digitais

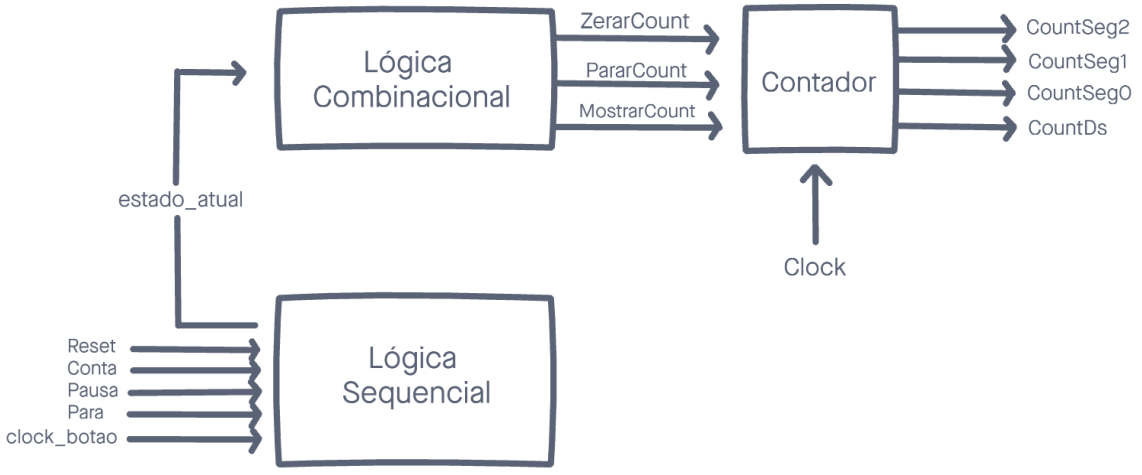
(Cronômetro Digital)

1. Visão Geral do Projeto

Com base nas especificações do projeto nós criamos o diagrama de estados a seguir, que ordena o funcionamento da máquina de estados:



Em seguida, desenvolvemos o código em verilog utilizando o Quartus como base. A visualização da estruturação do algoritmo pode ser representada com o diagrama abaixo.



2. Implementação Verilog

Primeiramente, criamos um único módulo para simplificar o desenvolvimento e declaramos todos os inputs, outputs e variáveis auxiliares utilizadas ao decorrer do projeto. Segue a respectiva parte do código:

```
module MaquinaEstados(reset, conta, pausa, para, clock, clock_botao,
CountDs, CountSeg0, CountSeg1, CountSeg2);

//Definindo inputs e outputs
input reset;
input conta;
input pausa;
input para;
input clock;
input clock_botao;

reg reset_old;
reg reset_raise;
reg conta_old;
reg conta_raise;
reg pausa_old;
reg pausa_raise;
reg para_old;
reg para_raise;

output reg [3:0] CountDs;
output reg [3:0] CountSeg0;
output reg [3:0] CountSeg1;
output reg [3:0] CountSeg2;

reg [1:0] estado_atual;
reg [3:0] CountSeg0Aux;
reg [3:0] CountSeg1Aux;
reg [3:0] CountSeg2Aux;
reg [3:0] CountDsAux;
reg ParарCount;
reg ZerarCount;
reg MostrarCount;
```

Para que a inicialização dos valores ocorra sem problemas, definimos os estados da máquina de estados como parâmetros e utilizamos o comando *initial* para inicializar os valores.

```
//00 = reset, 01 = contar, 10 = pausar, 11 = parar
parameter resetar = 0, contar = 1, pausar = 2, parar = 3;

//inicialização
    initial begin
        estado_atual <= resetar;
        CountDs <= 4'd0;
        CountSeg0 <= 4'd0;
        CountSeg1 <= 4'd0;
        CountSeg2 <= 4'd0;
        CountDsAux <= 4'd0;
        CountSeg0Aux <= 4'd0;
        CountSeg1Aux <= 4'd0;
        CountSeg2Aux <= 4'd0;
        ParарCount <= 1'd1;
        ZerarCount <= 1'd0;
        MostrarCount <= 1'd1;
        reset_old <= 1'b1;
        reset_raise <= 1'b0;
        conta_old <= 1'b1;
        conta_raise <= 1'b0;
        pausa_old <= 1'b1;
        pausa_raise <= 1'b0;
        para_old <= 1'b1;
        para_raise <= 1'b0;
    end
```

Na parte sequencial (Como ilustrada na Seção 1), implementamos a detecção da transição positiva para ativar a função dos botões (mudança de estado) apenas quando o usuário soltar o mesmo. Para isso, usamos mais 8 variáveis do tipo *reg,* 4 para armazenar o estado anterior do botão e 4 para ativar a função. Nessa etapa, decidimos usar o *clock_botao* para diminuir o tempo de atraso da resposta e consequentemente detectar mais rapidamente a transição entre estados.

```
always @ (posedge clock_botao)begin//parte sequencial cronometro

    // detect rising edge
    if (conta_old != conta && conta == 1'b1)begin
          conta_raise <= 1'b1;
    end
    conta_old <= conta;

    if (reset_old != reset && reset == 1'b1)begin
          reset_raise <= 1'b1;
    end
    reset_old <= reset;

    if (pausa_old != pausa && pausa == 1'b1)begin
          pausa_raise <= 1'b1;
    end
    pausa_old <= pausa;

    if (para_old != para && para == 1'b1)begin
          para_raise <= 1'b1;
    end
    para_old <= para;


   if(reset_raise == 1'b1)begin
      estado_atual <= resetar;
      reset_raise <= 1'b0;
   end
   else if((conta_raise == 1'b1) && (estado_atual != pausar))begin
      estado_atual <= contar;
      conta_raise <= 1'b0;
   end
   else if((pausa_raise == 1'b1) && (estado_atual == pausar))begin
      estado_atual <= contar;
      pausa_raise <= 1'b0;
   end
   else if((pausa_raise  == 1'b1) && (estado_atual != pausar))begin
      estado_atual <= pausar;
      pausa_raise <= 1'b0;
   end
   else if((para_raise  == 1'b1) && (estado_atual == contar))begin
      estado_atual <= parar;
      para_raise <= 1'b0;
   end

end
```

A parte combinacional, como podemos observar, é mais simples, e fica encarregada apenas de definir a ativação das saídas que serão usadas no contador adiante.

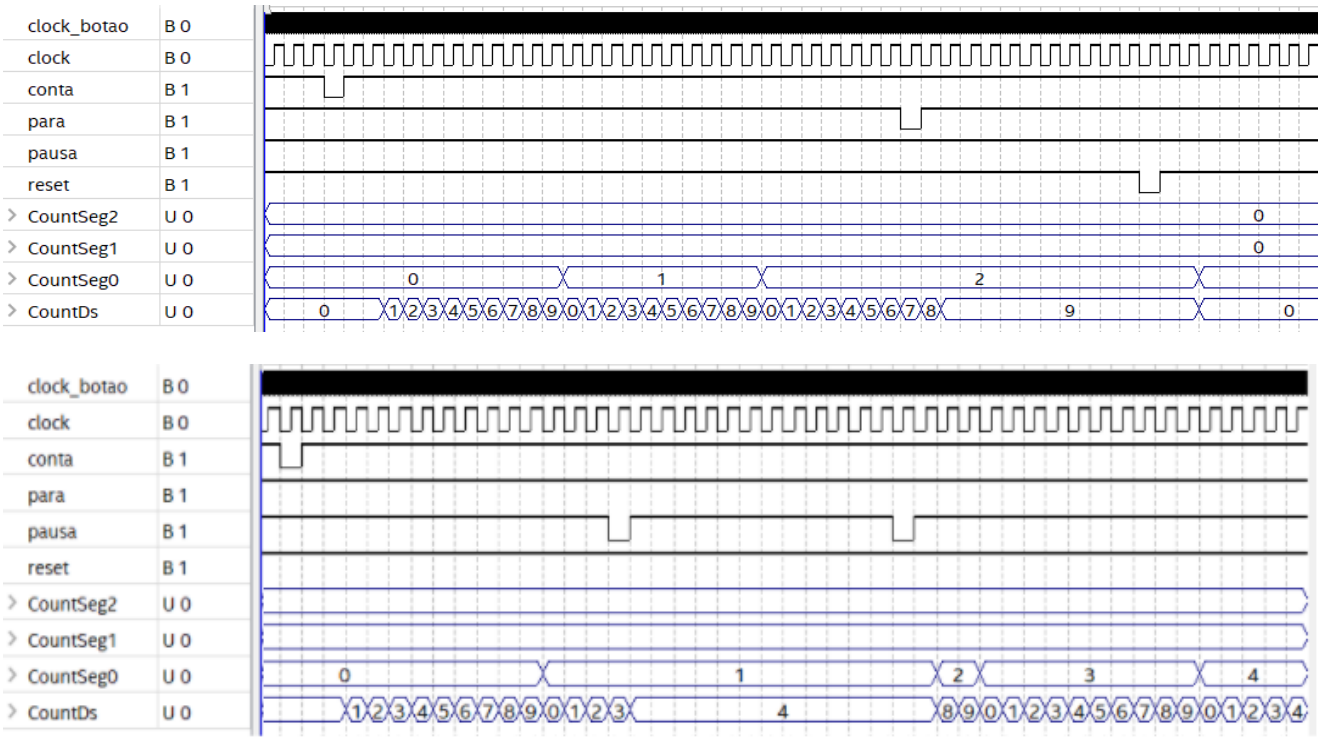
```
always @ (*) begin //parte combinacional
    case(estado_atual)
    resetar:begin
        ZerarCount <= 1'd1;
        PararCount <= 1'd1;
        MostrarCount <= 1'd1;
        end
    contar:begin
        ZerarCount <= 1'd0;
        PararCount <= 1'd0;
        MostrarCount <= 1'd1;
        end
    pausar:begin
        ZerarCount <= 1'd0;
        PararCount <= 1'd0;
        MostrarCount <=  1'd0;
        end
    parar:begin
        ZerarCount <= 1'd0;
        PararCount <= 1'd1;
        MostrarCount <= 1'd1;
        end
    endcase
end
```

Por fim, o contador utilizando o *clock,* realiza a contagem de décimos de segundos e consequentemente de segundos, definindo o valores dos outputs do módulo (CountDs, CountSeg0, CountSeg1, CountSeg2). Segue o código abaixo.

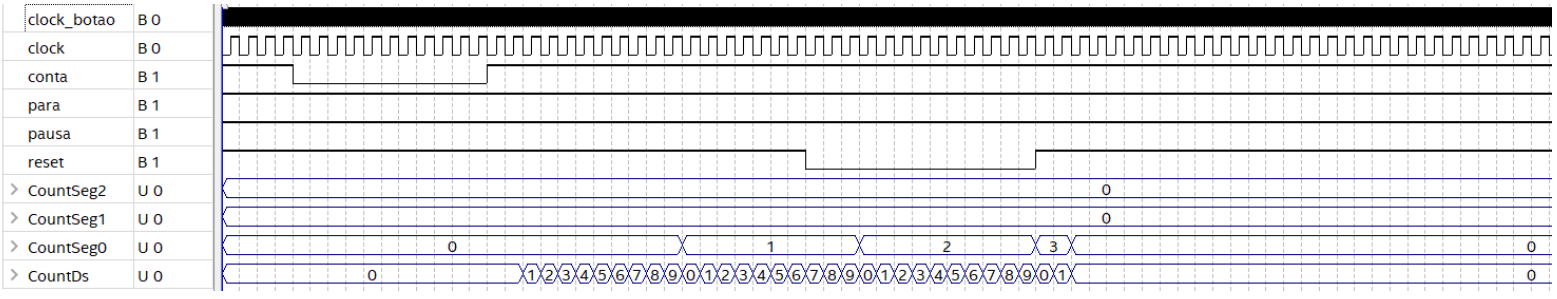
```
//contador
always@(negedge clock) begin

    if(ZerarCount) begin
        CountDsAux <= 4'd0;
        CountSeg0Aux <= 4'd0;
        CountSeg1Aux <= 4'd0;
        CountSeg2Aux <= 4'd0;
    end
    else begin
        if(!ParарCount)begin
            if(CountDsAux != 4'd9)begin
                CountDsAux <= CountDsAux + 4'd1;
            end
            else begin
                CountDsAux <= 4'd0;
                if(CountSeg0Aux != 4'd9)begin
                    CountSeg0Aux <= CountSeg0Aux + 4'd1;
                end
                else begin
                    CountSeg0Aux <= 4'd0;
                    if(CountSeg1Aux != 4'd9)begin
                        CountSeg1Aux <= CountSeg1Aux + 4'd1;
                    end
                    else begin
                        CountSeg1Aux <= 4'd0;
                        if(CountSeg2Aux != 4'd9)begin
                            CountSeg2Aux <= CountSeg2Aux + 4'd1;
                        end
                        else begin
                            CountDsAux <= 4'd0;
                            CountSeg0Aux <= 4'd0;
                            CountSeg1Aux <= 4'd0;
                            CountSeg2Aux <= 4'd0;
                        end
                    end
                end
            end
        end
    end


        if(MostrarCount)begin
            CountDs <= CountDsAux;
            CountSeg0 <= CountSeg0Aux;
            CountSeg1 <= CountSeg1Aux;
            CountSeg2 <= CountSeg2Aux;

        end

 end
```

Para verificar o funcionamento do algoritmo, iremos realizar as simulações (waveforms) na seção a seguir.

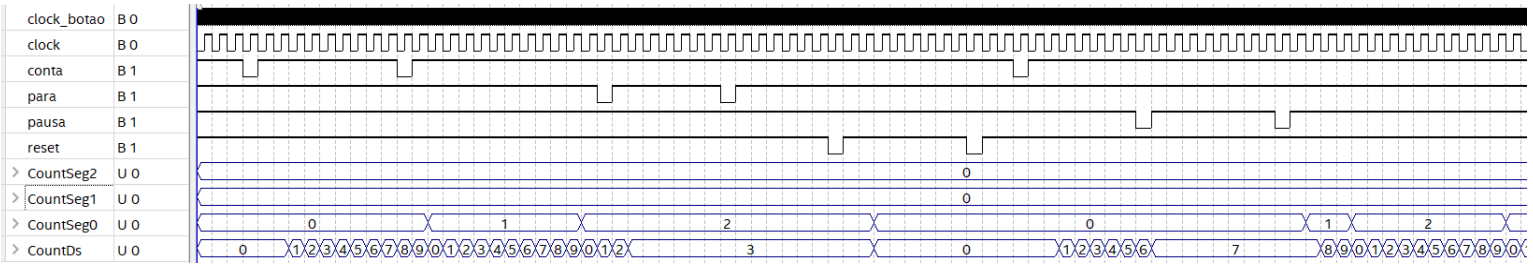
3. Simulação (Waveforms)



O Waveform acima mostra um teste completo, iniciando a contagem a partir do momento que se aperta o botão ***conta*** (é importante ressaltar que para representar segundos reais o *clock* precisa estar sincronizado em um período de 0,1 segundos). Depois disso, apertamos o botão ***para***, ***reseta*** e em seguida voltamos a contar apertando o botão ***conta***, por último, apertamos o botão ***pausa*** 2 vezes, parando a contagem no display e retomando com a contagem que acontece em background, após a segunda apertada, como solicitado.



O Waveform acima mostra o teste do botão, que mesmo pressionando o botão por vários períodos do *clock*, a ativação só acontece quando o botão deixa de ser pressionado.



Por último o waveform testando os casos dos botões sendo pressionados múltiplas vezes. Ou seja, caso os botões ***conta***, ***para*** ou ***reset*** sejam apertados mais de uma vez em sequência, nada acontecerá. Apenas apertando o botão ***pausa*** duas vezes que o cronômetro irá pausar a contagem após pressionado pela primeira vez, continuará a contagem em background, e quando pressionado pela segunda vez o display mostrará a contagem voltará de onde parou.