UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Emanuel de Oliveira
Gabriel Bazon
Giovanna Velasco
Guilherme Schmidt
João Pedro Gomes
João Victor Alves
Luana Hartmann

Projeto: Quarta Parte

São Carlos
2023

Emanuel de Oliveira
Gabriel Bazon
Giovanna Velasco
Guilherme Schmidt
João Pedro Gomes
João Victor Alves
Luana Hartmann

Projeto: Quarta Parte

Trabalho apresentado à disciplina de Sistemas Digitais da Escola de Engenharia de São Carlos, Universidade de São Paulo, como parte dos requisitos para a aprovação na disciplina.

Área de concentração: Engenharia de Computação

Orientador: Prof. Dr. Maximiliam Luppe

São Carlos
2023

Sumário

1 Introdução

O presente documento descreve o desenvolvimento e a implementação em Verilog de um Conversor Analógico para Digital de Rampa Dupla.

Para tal, será empregado o Contador BCD de 0 a 999, implementado na parte anterior, em conjunto com uma Máquina de Estados Finitos, denominado controlador, a ser implementado neste relatório.

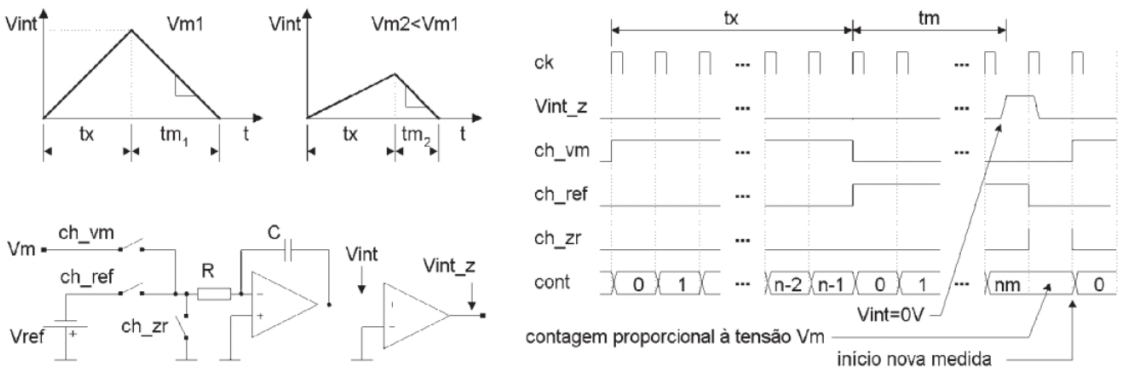
Portanto, passemos à descrição do Conversor.

2 Conversor Analógico para Digital de Rampa Dupla

Nesta seção, descreveremos o modo de funcionamento do Conversor em estudo, junto a seus principais componentes.

No conversor, a voltagem a ser medida Vm é aplicada sobre um circuito integrador por um período de tempo fixo tx, correspondente a um certo número de ciclos de clock. A seguir, uma tensão de referência $Vref$, de polaridade oposta, passa a ser aplicada sobre a entrada do integrador, descarregando o capacitor. Então quando a tensão de saída do integrador $Vint$ torna-se nula, a contagem é finalizada, medindo-se então o tempo decorrido para a descarga do integrador, sendo que este tempo é diretamente dependente das tensões de entrada e de referência, permitindo então determinar a entrada a partir dos parâmetros dados. A Figura 1 ilustra o processo.

Figura 1 – Parte analógica do ADC, acompanhada de gráficos ilustrativos da tensão de saída do integrador em função do tempo decorrido. São encontrados também os sinais de controle do Conversor em função de seu estado de funcionamento, à direita.

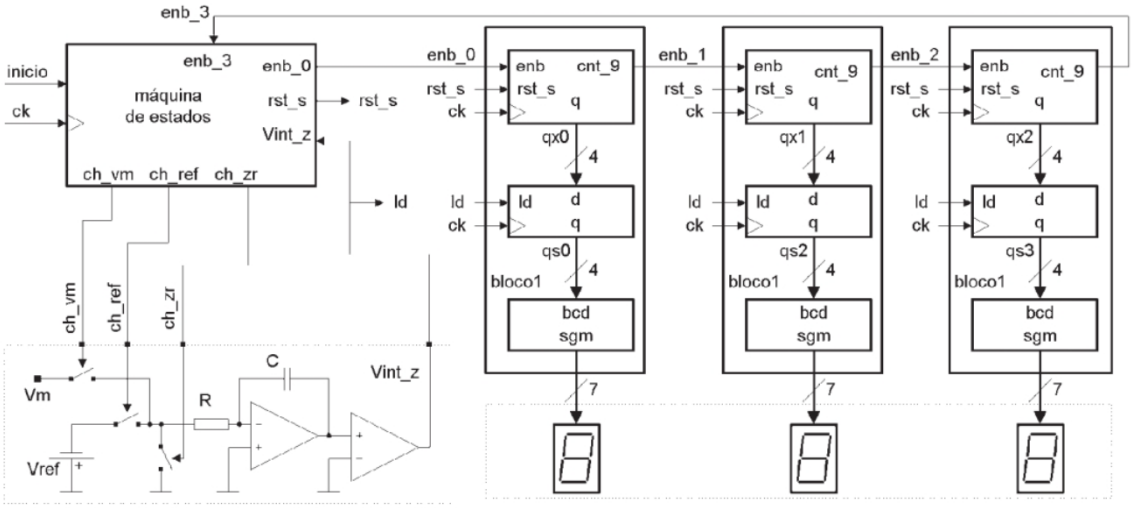


À direita da Figura 1 também nos traz os sinais de controle do Conversor em função de seu estado de funcionamento. Portanto, traremos o significado de cada sinal:

- *ck*: trata-se do clock do circuito;
- $Vint\_z$: sinaliza o momento em que $Vint$ torna-se zero;
- ch_vm: sinal que controla a entrada da tensão Vm no integrador;
- ch_ref: sinal que controla a entrada da tensão $Vref$ no integrador;
- ch_zr: sinal que leva a saída do integrador a zero;
- *cont*: trata-se do output do contador.

Portanto de forma a lidar com tais sinais e promover o funcionamento do conversor, mostra-se necessário o empregado de uma Máquina de Estados, que irá receber os sinais do circuito e promover as corretas saídas de forma a viabilizar o procedimento de medição. A Figura 2 traz o circuito completo, com a parte analógica, máquina de estados, contador, registrador e display.

Figura 2 – ADC completo, contendo todos os seus componentes



Portanto, a seguir descreveremos em detalhes a máquina de estados e seu funcionamento dentro do ADC.

3 Máquina de Estados

A partir das informações acerca do funcionamento do conversor providas na última seção, fica claro que existem diversos estágios em seu funcionamento, os quais devem ser controlados por uma máquina de estados. Nesse sentido, abaixo são identificadas as saídas em cada estado.

1. Carregamento do integrador
   - $ch\_vm = 1$. Permitindo a entrada de Vm no integrador.
   - $ch\_ref = 0$.
   - $ch\_zr = 0$.
   - $rst\_s = 0$.
   - $enb\_0 = 1$. Permitindo a contagem no contador.
2. Descarregamento do integrador
   - $ch\_vm = 0$.
   - $ch\_ref = 1$. Permitindo a entrada de $Vref$ no integrador.
   - $ch\_zr = 0$.
   - $rst\_s = 0$.
   - $enb\_0 = 1$. Permitindo a continuidade da contagem no contador.
3. Anulamento do integrador
   - $ch\_vm = 0$.
   - $ch\_ref = 0$.
   - $ch\_zr = 1$. Zerando a saída do integrador.
   - $rst\_s = 0$.
   - $enb\_0 = 0$. Parando o contador.
4. Reset
   - $ch\_vm = 0$.
   - $ch\_ref = 0$.
   - $ch\_zr = 0$.
   - $rst\_s = 1$. Reiniciando o contador.

- $enb\_0 = 0$.

Destaca-se que o transição do estado 1 para o estado 2 é ativada pelo sinal enb_3 produzida pelo contador, indicando o fim da contagem, enquanto a transição de 2 para 3 dá-se pelo sinal $Vint\_z$ produzido quando a saída do integrador chega a zero, e também responsável por ativar o armazenamento dos registradores. Por fim, a passagem de 3 para 4 ocorre em um ciclo de clock adicional. Deste modo, finaliza-se a operação.

Ressalta-se também que o início do funcionamento da máquina de estados ocorre por meio da entrada *inicio*, que envia um sinal de reset para o contador e ativa o estado 1 (*inicio* deve ser desativado para iniciar o funcionamento). Logo, temos 3 entradas para o controlador:

- *inicio*;
- $Vint\_z$;
- enb_3.

Portanto, partimos à implementação em Verilog.

4 Implementação da Máquina de Estados

Para a implementação do circuito, será utilizada a linguagem de descrição de hardware Verilog. Destaca-se que o código pode ser também encontrado no repositório do projeto. Ressalta-se que a implementação baseia-se nas descrições anteriores da Máquina de Estados.

Abaixo segue o código criado e circuito esquemático correspondente ao módulo descrito (Figura 3).

```
module control_machine( input inicio,
                        input clk,
                        input enb_3,
                        input Vint_z,
                        output ch_vm,
                        output ch_vr,
                        output ch_zr,
                        output rst_s,
```

```
                        output enb_0);
  reg working;
  reg [2:0] next_phase;
  always @(posedge clk) begin
    // Inicia o sistema
    if(inicio) begin
      working = 1'b1; // Inicia operacao
      next_phase = 1; // Inicia na primeira fase do controlador
      rst_s = 1'b1;
      enb = 1'b0;
    end

    // Logica das fases de operacao
    else if(working == 1'b1) begin
      if(next_phase == 1) begin
        ch_vm = 1'b1;
        ch_vr = 1'b0;
        ch_zr = 1'b0;
        rst_s = 1'b0;
        enb_0 = 1'b1;
        next_phase = 2;
      end else if (next_phase == 2 && enb_3) begin
        ch_vm = 1'b0;
        ch_vr = 1'b1;
        next_phase = 3;
      end else if (next_phase == 3 && Vint_z) begin
        ch_vr = 1'b0;
        ch_zr = 1'b1;
        enb_0 = 1'b0;
        next_phase = 4;
      end else if (next_phase == 4) begin
        ch_zr = 1'b0;
        rst_s = 1'b1;
        working = 1'b0; // Finaliza operacao
      end
    end
  end
```

```
endmodule
```

A seguir temos os resultados de uma simulação realizada sobre o mesmo, em que ele produz as saídas esperadas dada a entrada aplicada, mudando de estado adequadamente, conforme demonstrado na Figura 4. Ressalta-se que para a primeira mudança de estado, de 'indeterminado' para 1, foi aplicada a entrada *inicio*.

5 Controlador

A partir de todos os módulos até então implementados, resta apenas uní-los no Controlador já definido. Portanto, partiremos diretamente a sua implementação em Verilog.

Abaixo segue o código criado e circuito esquemático correspondente ao módulo descrito (Figura 5).

```
// Decodificador BCD para display 7 segmentos, produzido na parte 1
module BCDto7seg #(parameter common_cathode=1)(output a, b, c, d, e, f,
↪  g, input [3:0] BCD);
  reg [6:0] seg;

  generate
    if (common_cathode == 1)
      always @ (BCD)
        begin
          case(BCD)
            0: seg = 7'b1111110;
            1: seg = 7'b0110000;
            2: seg = 7'b1101101;
            3: seg = 7'b1111001;
            4: seg = 7'b0110011;
            5: seg = 7'b1011011;
            6: seg = 7'b1011111;
            7: seg = 7'b1110000;
            8: seg = 7'b1111111;
            9: seg = 7'b1111011;
            default: seg=7'b1001111;
          endcase
```

```
        end
        else
      always @ (BCD)
        begin
          case(BCD)
            0: seg = 7'b0000001;
            1: seg = 7'b1001111;
            2: seg = 7'b0010010;
            3: seg = 7'b0000110;
            4: seg = 7'b1001100;
            5: seg = 7'b0100100;
            6: seg = 7'b0100000;
            7: seg = 7'b0001111;
            8: seg = 7'b0000000;
            9: seg = 7'b0000100;
            default: seg=7'b0110000;
          endcase
        end
  endgenerate
  assign {a,b,c,d,e,f,g} = seg;
endmodule

// Flip-Flop tipo D com sinal 'enb'
module dff_4bit(output reg [3:0] q, input [3:0] d, input clk, enb);
  always @ (posedge clk)
    if (enb)
      q = d;
endmodule

// Contador parametrizavel comportamental, produzido na parte 2
module behav_counter #(parameter width=4, max_count=10)(input clk, rst_s,
↪  enb, output reg [width-1:0] q, output cnt_max);

  always @ (posedge clk, posedge rst_s)
    begin
      cnt_max<=0;
      if (rst_s)
```

```
        q <= 0;
      else if (enb)
        begin
          if (q < max_count-2)
            q<=q+1;
          else if (q == max_count-2)
            begin
              q<=q+1;
              cnt_max<=1;
            end
          else
            q<=0;
         end
     end

endmodule

// Contador digital BCD de 000 a 999, com 21 fios de output para 3
↪  displays de 7 segmentos
module counter_999 (output [20:0] display, output cnt_max, input clk,
↪  enb, rst_s, ld);

  reg [3:0] qx[3];
  reg [3:0] qs[3];
  wire max[3];

  // Gera 3 contadores de 0 a 9 em sequência
  genvar i;
  generate
    for (i=0; i<3; i=i+1)
      begin
        if (i == 0)
          // Primeiro contador
          behav_counter #(.width(4), .max_count(10))
          u0 (.q(qx[0]), .cnt_max(max[0]), .clk(clk), .enb(enb),
↪  .rst_s(rst_s));
        else
```

```
        // Contadores seguintes, com a saida 'cnt_max' do anterior
↪  entrando em 'enb' no seguinte
        behav_counter #(.width(4), .max_count(10))
        ui (.q(qx[i]), .cnt_max(max[i]), .clk(clk), .enb(max[i-1]),
↪  .rst_s(rst_s));

      dff_4bit register (.q(qs[i]), .clk(clk), .d(qx[i]), .enb(ld));

      // Decodificador BCD para 7 segmentos recebendo 'q' do contador
↪  atual e retornando os 7 fios de display
      BCDto7seg #(.common_cathode(1))
      bcdi (.a(display[i*7]), .b(display[1+i*7]), .c(display[2+i*7]),
↪  .d(display[3+i*7]),
       .e(display[4+i*7]), .f(display[5+i*7]), .g(display[6+i*7]),
↪  .BCD(qs[i]));
    end
  endgenerate

  assign cnt_max = max[2];

endmodule

module control_machine( input inicio,
                        input clk,
                        input enb_3,
                        input Vint_z,
                        output ch_vm,
                        output ch_vr,
                        output ch_zr,
                        output rst_s,
                        output enb_0);
  reg working;
  reg [2:0] next_phase;
  always @(posedge clk) begin
    // Inicia o sistema
    if(inicio) begin
      working = 1'b1; // Inicia operacao
```

```
      next_phase = 1; // Inicia na primeira fase do controlador
      rst_s = 1'b1;
      enb = 1'b0;
    end

    // Logica das fases de operacao
    else if(working == 1'b1) begin
      if(next_phase == 1) begin
        ch_vm = 1'b1;
        ch_vr = 1'b0;
        ch_zr = 1'b0;
        rst_s = 1'b0;
        enb_0 = 1'b1;
        next_phase = 2;
      end else if (next_phase == 2 && enb_3) begin
        ch_vm = 1'b0;
        ch_vr = 1'b1;
        next_phase = 3;
      end else if (next_phase == 3 && Vint_z) begin
        ch_vr = 1'b0;
        ch_zr = 1'b1;
        enb_0 = 1'b0;
        next_phase = 4;
      end else if (next_phase == 4) begin
        ch_zr = 1'b0;
        rst_s = 1'b1;
        working = 1'b0; // Finaliza operacao
      end
    end
  end
endmodule

module controler( input inicio,
                  input clk,
                  input Vint_z,
                  output ch_vm,
                  output ch_vr,
```

```
                output ch_zr,
                output [20:0] display);

  wire enb_0, enb_3, Vint_z, rst;

  control_machine contr(.inicio(inicio), .clk(clk), .enb_3(enb_3),
↪  .Vint_z(Vint_z),
  .ch_vm(ch_vm), .ch_vr(ch_vr), .ch_zr(ch_zr), .rst_s(rst),
↪  .enb_0(enb_0));

  counter_999 cnt(.display(display), .cnt_max(enb_3), .clk(clk),
↪  .enb(enb_0), .rst_s(rst), .ld(Vint_z));

endmodule
```

Neste ponto, impossibilitada uma simulação adequada dada a ausência da parte analógica, consideram-se os resultados conforme a simulação realizada sobre a máquina de estados.

6 Conclusão

A partir do exposto, pode-se concluir que o Controlador foi implementado de modo adequado, uma vez que as simulações geradas a partir da linguagem Verilog fornecem resultados corretos para os testes realizados.

No mais, ressaltam-se que não houveram grandes dificuldades nesta seção do trabalho, dada a familiaridade com o assunto, já adequadamente trabalhado em partes anteriores.

7 Bibliografia

V. P. Nelson. *Digital logic circuit analysis and design*. Prentice Hall, 1995

Figura 3 – Circuito esquemático da Máquina de Estados

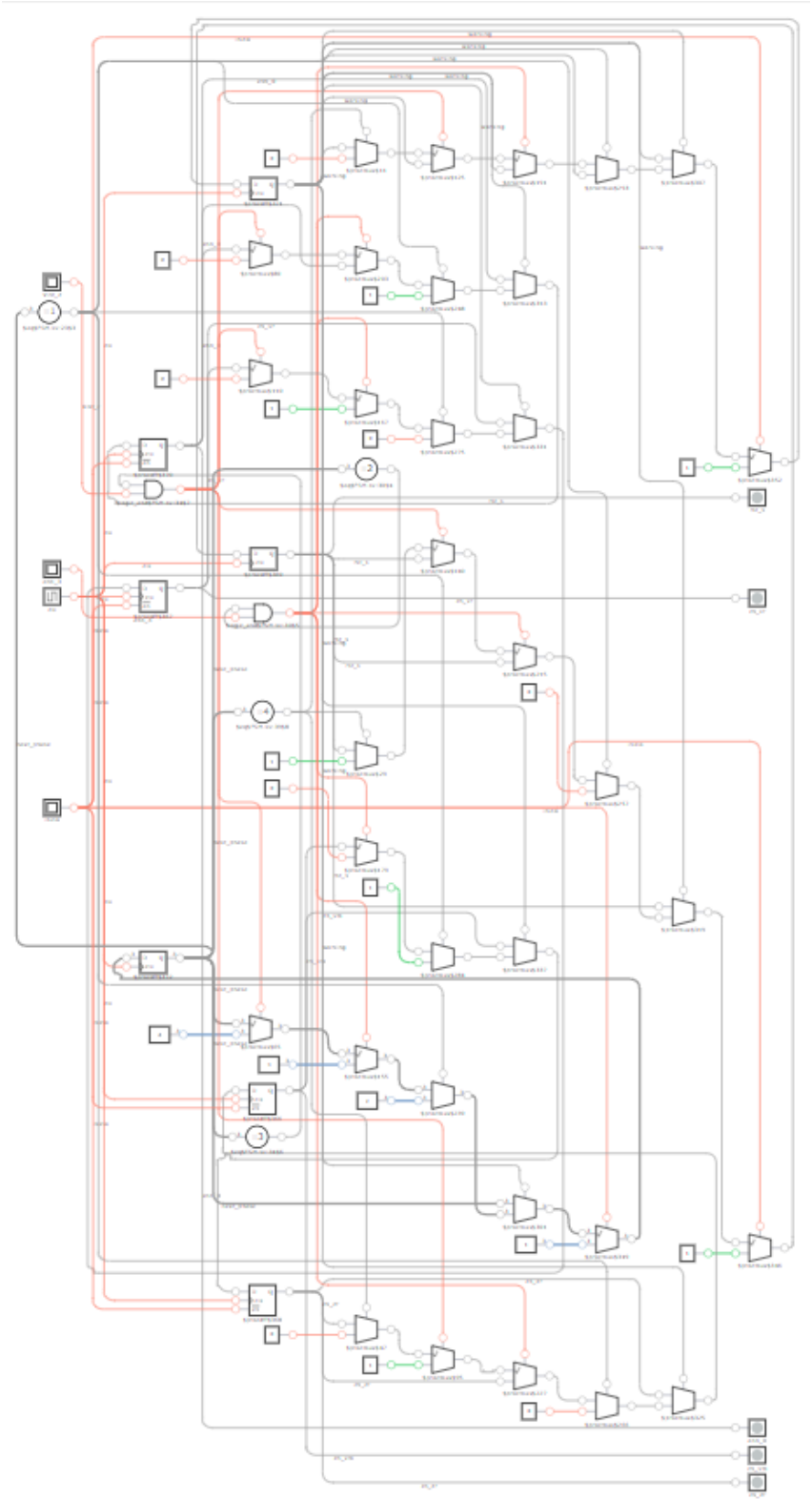


Figura 4 – Resultados da simulação da Máquina de Estados

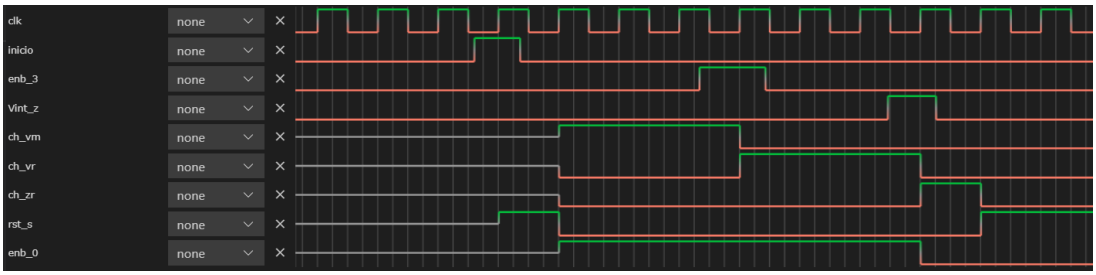


Figura 5 – Circuito esquemático do Controlador