Lista 3

Integrantes:

- Amanda Quirino Rodrigues Dos Santos
- Igor Rafael De Oliveira Barbosa

Questão 1:

a)

```
module Subtractor(input wire A, B, Bin,
                  output wire D, Bout);

  assign D = (A ^ B) ^ Bin;
  assign Bout = (Bin & (~A | B)) | (~A & B);
endmodule
```

b)

```
module Full_Subtractor(input wire A, B, Bin,
                  output wire D, Bout);

  assign D = (A ^ B) ^ Bin;
  assign Bout = (Bin & (~A | B)) | (~A & B);
endmodule

module Subtractor_4bits(input wire [3:0] A, B,
                  input wire Bin,
                  output wire [3:0] D,
                  output wire Bout);

  wire Bout1, Bout2, Bout3;


  Full_Subtractor full0 (A[0], B[0], 0, D[0], Bout1);

  Full_Subtractor full1 (A[1], B[1], Bout1, D[1], Bout2);

  Full_Subtractor full2 (A[2], B[2], Bout2, D[2], Bout3);

  Full_Subtractor full3 (A[3], B[3], Bout3, D[3], Bout);
```

```
endmodule
```

Questão 2

a)

D muda para 1 em 20ns, portanto os comandos dentro do always serão executados nesse momento, então A muda para 1 em 25ns, pois o comando que muda A tem delay de 5ns, logo 20ns+5ns=25ns, e B muda para 1 em 20ns,pois não há delay e estamos usando , por último C muda para 1 depois de um delay de 10ns, logo C muda em 20ns+5ns+10ns = 35ns, ou seja:

D muda para 1 em 20ns,

C muda para 1 em 35ns,

B muda para 1 em 25ns e

A muda para 1 em 25ns.

b)

D muda para 1 em 20ns, portanto os comandos dentro do always serão executados nesse momento, então A receberá 1 5ns depois, logo A muda para 1 em 25ns. Já B assume 1 em 20ns, pois receberá o valor da operação 0 + 1. Por último, C não irá mudar, pois a única oportunidade de mudar seria se B fosse 1 no momento da atribuição, porém como no momento em que o comando que atribui valor a C, B tem valor 0, então C receberá valor 0 10ns após o comando ser executado, ou seja, C não mudará, portanto:

D muda para 1 em 20ns,

C nunca muda,

B muda para 1 em 20ns e

A muda para 1 em 25ns.

Questão 3

a)

O código do meio somador será compilado, porém não será simulado corretamente, pois como temos o comando “always @(x)” então isso significa que o bloco que está dentro do always só será executado quando a variável x mudar de valor, porém nós queremos que esse bloco seja executado sempre que o add mudar.

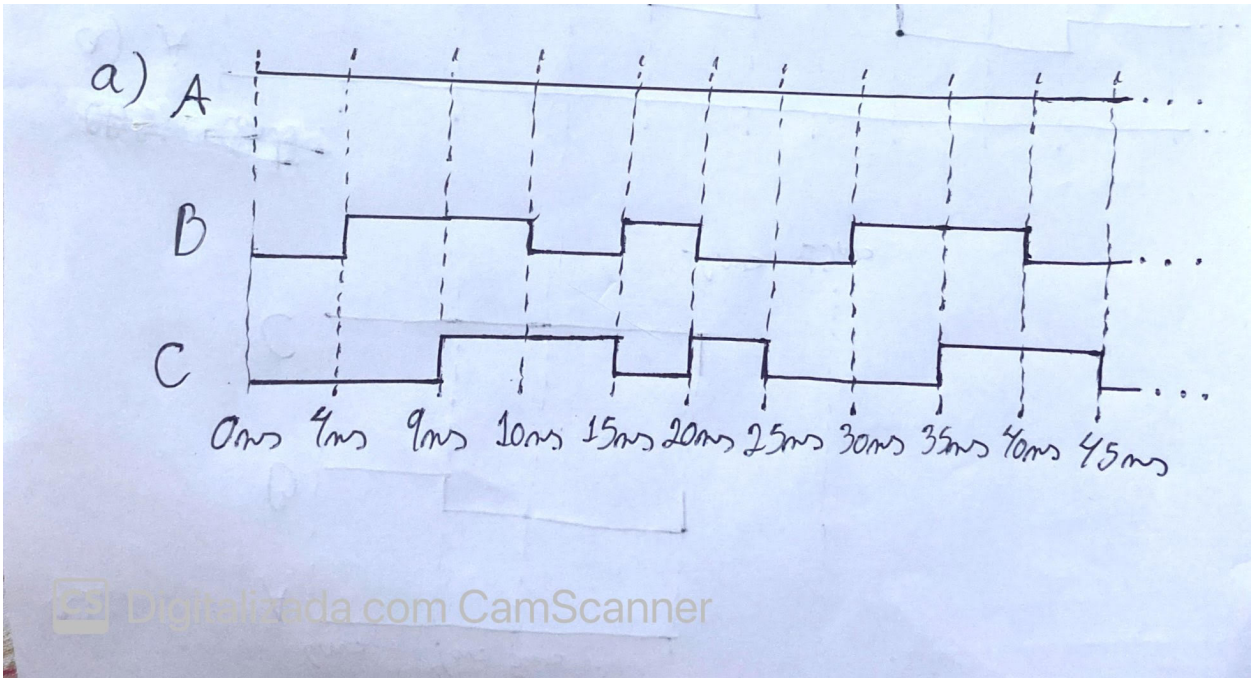
b)

O código do MUX 4-para-1 funcionará corretamente na simulação e na síntese, pois no comando “always” está todas as variáveis cuja mudanças poderiam mudar a saída, como as variáveis A e B que são os bits de seleção, que farão com que a variável sel possa assumir valor 0, 1, 2 ou 3, e as variáveis I0, I1, I2 e I3 que são as entradas.

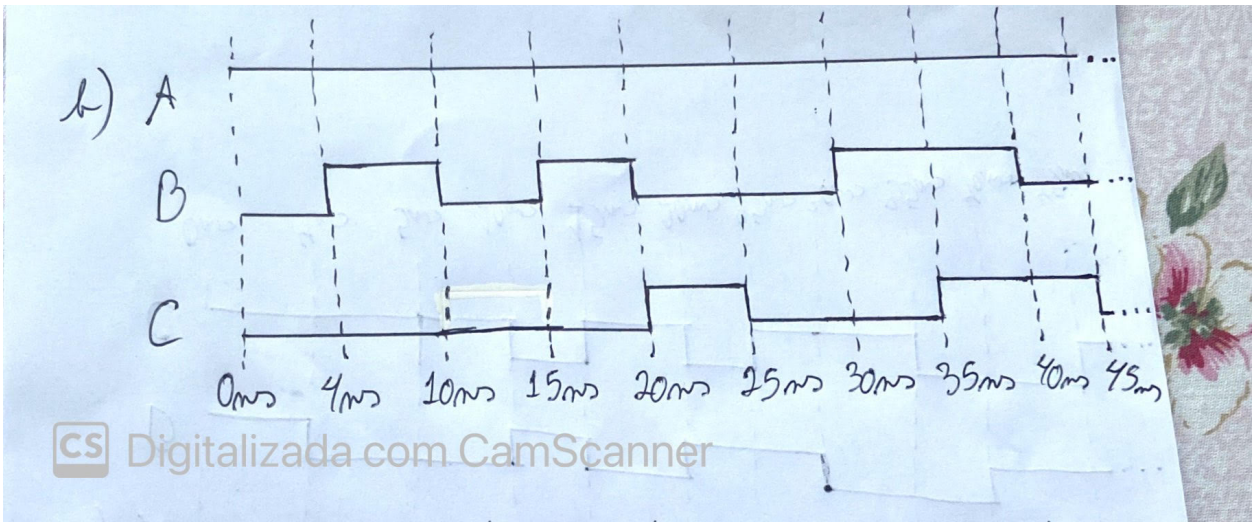
Questão 4

Tanto na letra a) quanto na letra b), assumimos que A sempre será 1.

a)



b)



Questão 5

a)

Se o cálculo do tamanho de uma ROM pode ser feito por: (2^(inputs))*outputs temos que:

Somador Completo de 8 bits: (16 + 1) inputs, por causa dos dois números de 8 bits e do carry in, e (8 + 1) outputs, por causa do número binário de 8 bits que saíra e do carry out. Dessa forma, o menor tamanho da ROM seria (2^(17))*9.

MUX 4-1: (4 + 2) inputs, por causa dos 4 bits de entrada e dos 2 bits de seleção, e (1) output. Assim, o menor tamanho da ROM é (2^(6))*1.

Decodificador 3-8: (3) inputs e (2^3=8) outputs. Dessa maneira, o menor tamanho da ROM é (2^(3))*8.

b)

Seguindo a mesma lógica da letra A, temos:

Conversor de BCD para Binário: (8) inputs e (7) outputs. Assim, o menor tamanho da ROM é (2^(8))*7.

Somador de 32 bits: (32*2=64) inputs e (32+1) outputs. Dessa forma, o menor tamanho da ROM é (2^(64))*33.

Codificador de prioridade 8-a-3: (8) inputs e (3+1) outputs. Dessa maneira, o menor tamanho da ROM é (2^(8))*4.

Questão 6

a)

```
module ROM4_3(ROMin, ROMout);
  input[3:0] ROMin;
  output[2:0] ROMout;
  reg[2:0] ROM16X3 [15:0];

  initial
    begin
      ROM16X3[0] <= 3'b000;
      ROM16X3[1] <= 3'b001;
      ROM16X3[2] <= 3'b001;
      ROM16X3[3] <= 3'b010;
      ROM16X3[4] <= 3'b001;
      ROM16X3[5] <= 3'b010;
      ROM16X3[6] <= 3'b010;
      ROM16X3[7] <= 3'b011;
      ROM16X3[8] <= 3'b001;
      ROM16X3[9] <= 3'b010;
      ROM16X3[10] <= 3'b010;
      ROM16X3[11] <= 3'b011;
      ROM16X3[12] <= 3'b010;
      ROM16X3[13] <= 3'b011;
      ROM16X3[14] <= 3'b011;
      ROM16X3[15] <= 3'b100;
    end
  assign ROMout = ROM16X3[ROMin];
endmodule
```

Bancada de Teste

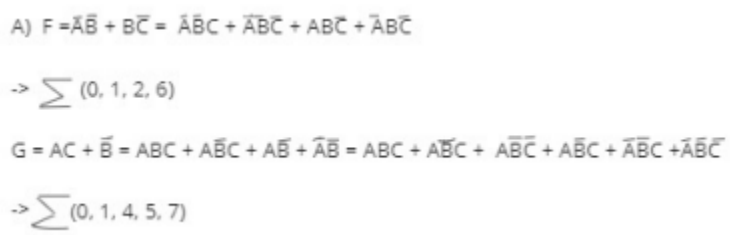
```
`timescale 1ns/1ps

module top;
  reg[3:0] ROMin_tb;
  wire[2:0] ROMout_tb;

  ROM4_3 uut(
    .ROMin(ROMin_tb),
    .ROMout(ROMout_tb)
  );

  initial
    begin
      ROMin_tb=4'b0000;
      #10 ROMin_tb=4'b0001;
      #10 ROMin_tb=4'b0010;
      #10 ROMin_tb=4'b0011;
      #10 ROMin_tb=4'b0100;
      #10 ROMin_tb=4'b0101;
      #10 ROMin_tb=4'b0110;
      #10 ROMin_tb=4'b0111;
      #10 ROMin_tb=4'b1000;
      #10 ROMin_tb=4'b1001;
      #10 ROMin_tb=4'b1010;
      #10 ROMin_tb=4'b1011;
      #10 ROMin_tb=4'b1100;
      #10 ROMin_tb=4'b1101;
      #10 ROMin_tb=4'b1110;
      #10 ROMin_tb=4'b1111;

      #1000;
    end

    initial begin
        $dumpfile("design.vcd");
        $dumpvars(0,top);
    end

    initial begin
        $monitor("t=%3d ROMin_tb=%d, ROMout_tb=%d\n", $time, ROMin_tb, ROMout_tb);
    end

endmodule
```

b)

```
module question_6_b (my_input, count);
    input[11:0] my_input;
    output[3:0] count;
    wire[2:0] K, X, Y;

    ROM4_3 R1(my_input[11:8], K);
    ROM4_3 R2(my_input[7:4], X);
    ROM4_3 R3(my_input[3:0], Y);

    assign count = {1'b0, K} + X + Y;

endmodule
```

Questão 7

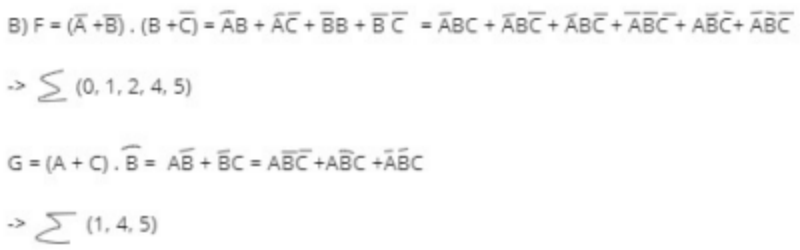
a)




```
module question_7 (my_input, out, F, G);

    output [0:1] out;
    input [0:2] my_input;
    reg [0:1] out;
    reg [0:1] rom8x2 [0:7];
    output wire F, G;

    initial begin
        rom8x2[0] = 2'b11;
        rom8x2[1] = 2'b11;
        rom8x2[2] = 2'b10;
        rom8x2[3] = 2'b00;
        rom8x2[4] = 2'b01;
        rom8x2[5] = 2'b01;
        rom8x2[6] = 2'b10;
        rom8x2[7] = 2'b01;
    end

    assign out = rom8x2[my_input];
```

```
    assign F = out[0];
    assign G = out[1];

endmodule
```

b)



B) $F = (\overline{A} + \overline{B}) \cdot (B + \overline{C}) = \overline{A}\overline{B} + \overline{A}\overline{C} + \overline{B}B + \overline{B}\,\overline{C} = \overline{A}BC + \overline{A}B\overline{C} + \overline{A}B\overline{C} + \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C}$

$\rightarrow \sum (0, 1, 2, 4, 5)$

$G = (A + C) \cdot \overline{B} = A\overline{B} + \overline{B}C = A\overline{B}\overline{C} + A\overline{B}C + \overline{A}\overline{B}C$

$\rightarrow \sum (1, 4, 5)$

```
module question_7 (my_input, out, F, G);

    output [0:1] out;
    input [0:2] my_input;
    reg [0:1] out;
    reg [0:1] rom8x2 [0:7];
    output wire F, G;

    initial begin
        rom8x2[0] = 2'b10;
        rom8x2[1] = 2'b11;
        rom8x2[2] = 2'b10;
        rom8x2[3] = 2'b00;
        rom8x2[4] = 2'b11;
        rom8x2[5] = 2'b11;
        rom8x2[6] = 2'b00;
        rom8x2[7] = 2'b00;
    end

    assign out = rom8x2[my_input];

    assign F = out[0];
    assign G = out[1];

endmodule
```

Questão 8

a)

A empresa deve escolher o SRAM FDGA, pois ele é mais volátil e permite que, ao longo das revisões, seja reprogramado, fazendo ele um ótimo componente para protótipos. Como a cada inicialização ele deve ser reprogramado, há uma certa facilidade em sua revisão e refinamento.

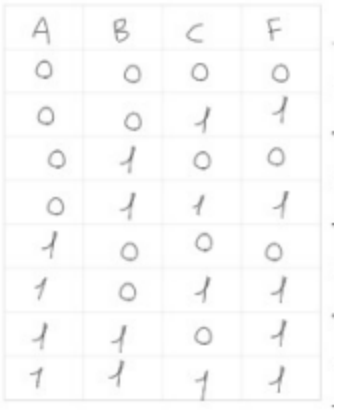
b)

Certamente o MPGA é o ideal para esse caso. Porque ele tem um custo reduzido, em relação ao FDGA, tornando-o bem mais viável devido à alta escala de produção

Questão 9

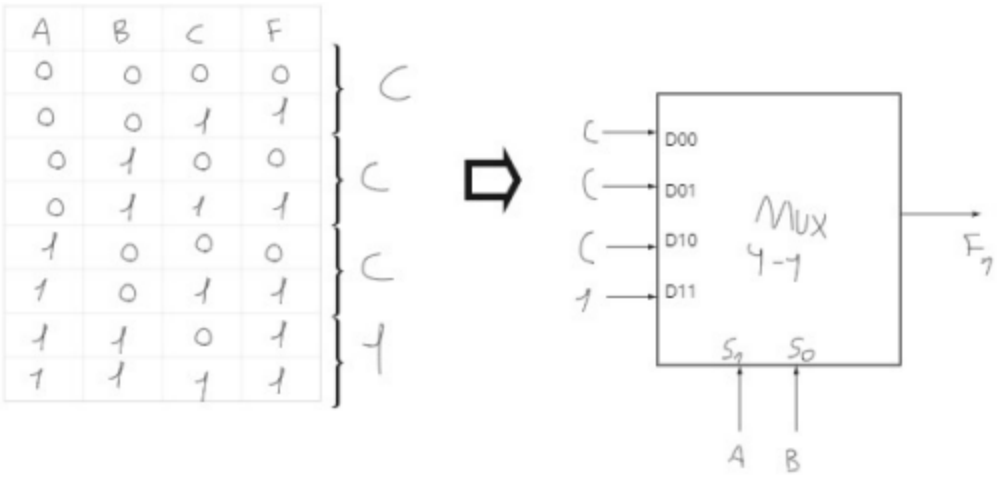
a)

Se a tabela verdade pode ser escrita como:



A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Percebemos que podemos montar uma relação em que A e B assumem os valores dos bits de seleção do MUX 4-1. Dessa forma, ao analisarmos a tabela verdade, temos as seguintes sequências de entradas:



b)

Seguindo a lógica da letra a), podemos deduzir a mesma coisa. De início fazemos a tabela verdade da função e analisamos um padrão, o qual possa ser usado para montar um MUX 4-1. Dessa maneira, temos que o menor MUX 4-1 para F1 é: