Sistemas Digitais

2020/2021

1º Projeto

Gerador de Código de Barras

Membros do grupo:

- Hugo Rocha nº 2046019;
- Sérgio Oliveira nº 2046719;

Introdução

Neste trabalho foi-nos pedido que pedido que implementássemos um gerador de código de barras no software ISE Design Suite 14.7 da Xilinx. O valor da soma dos números mecanográficos de ambos os alunos envolvidos é igual a 54, logo a versão a cumprir seria a versão X onde será necessário abordar o valor PG em excesso 5 convertido para hexadecimal.

Inicialmente, pensámos em guardar o tipo de cliente numa só variável, onde associávamos o valor lógico 0 ao ClientA, e 1 ao ClientB. Esta implementação não permitiria verificar se um dos botões dos clientes foi ou não acionado.

Módulo Verilog - ValueToPay

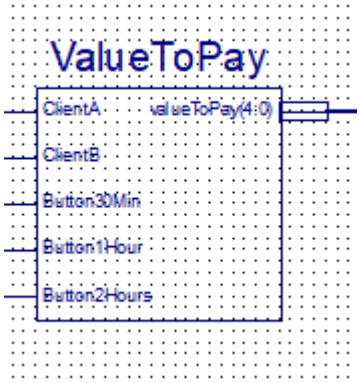
Decidimos que o ValueToPay toma o valor 0 quando é tomado por parte do utilizador um comportamento inesperado.

A natureza não dinâmica da máquina permite uma série de otimizações, pois o número de valores que o ValueToPay pode assumir é fixo:

- Sabemos à partida que o valor máximo do ValueToPay é 28€, fazendo com que tenhamos que alocar apenas 5 bits em memória.

ClientA	ClientB	Button30Min	Button1Hour	Button2Hours	ValueToPay
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	10000 (16€)
0	1	0	1	0	01000 (8€)
0	1	0	1	1	11000 (24€)
0	1	1	0	0	00100 (4€)
0	1	1	0	1	10100 (20€)
0	1	1	1	0	01100 (12€)
0	1	1	1	1	11100 (28€)
1	0	0	0	0	0
1	0	0	0	1	01000 (8€)

1	0	0	1	0	00100 (4€)
1	0	0	1	1	01100 (12€)
1	0	1	0	0	00010 (2€)
1	0	1	0	1	01010 (10€)
1	0	1	1	0	00110 (6€)
1	0	1	1	1	01110 (14€)
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0



Módulo Verilog – StudentNumbers

Caso apenas o ClientA esteja ativo (1), o módulo produz uma saída equivalente à concatenação dos números A1 e A2 (20460192049719).

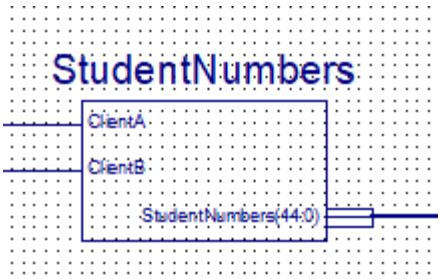
Caso apenas o ClientB esteja ativo (1), o módulo produz uma saída equivalente à concatenação dos números A2 e A1 (20497192046019).

Qualquer outro comportamento é inesperado, e produz uma saída a 0.

A natureza não dinâmica da máquina permite uma série de otimizações, pois o número final concatenado dos alunos é fixo:

- A concatenação dos números dos alunos pode ser pré-calculada.
- O número concatenado final será um número composto por 14 dígitos decimais. Se todos os dígitos forem 9, este número ocupará 47 bits em memória. Sabermos à partida o número de cada aluno permite-nos poupar 2 bits em memória.

ClientA	ClientB	StudentNumbers
0	0	0
0	1	A2 A1
1	0	A1 A2
1	1	0

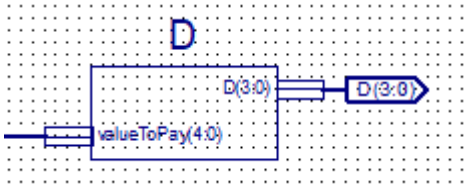


Módulo Verilog – D

A natureza não dinâmica da máquina permite uma série de otimizações, pois o número de valores que o ValueToPay pode assumir é fixo:

- O D pode ser pré-calculado.
- Sabemos à partida que o valor máximo de D é 14, quando o ValueToPay é 20, fazendo com que tenhamos que alocar apenas 4 bits em memória.

ValueToPay	D (decimal)
10000 (16€)	10
01000 (8€)	2
11000 (24€)	3
00100 (4€)	13
10100 (20€)	14
01100 (12€)	6
11100 (28€)	7
01000 (8€)	2
00100 (4€)	13
01100 (12€)	6
00010 (2€)	11
01010 (10€)	4
00110 (6€)	0
01110 (14€)	8
00000 (Error)	0

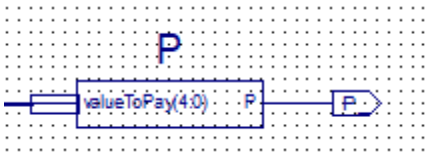


Módulo Verilog – P

A natureza não dinâmica da máquina permite uma série de otimizações, pois o número de valores que o ValueToPay pode assumir é fixo:

- O P pode ser pré-calculado.

ValueToPay	P (binário)
10000 (16€)	1
01000 (8€)	0
11000 (24€)	0
00100 (4€)	0
10100 (20€)	0
01100 (12€)	1
11100 (28€)	0
01000 (8€)	0
00100 (4€)	0
01100 (12€)	1
00010 (2€)	0
01010 (10€)	1
00110 (6€)	0
01110 (14€)	1
00000(Error)	0

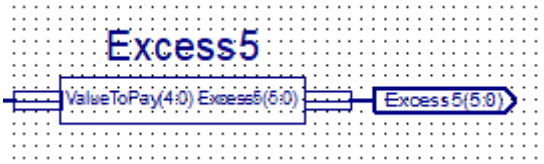


Módulo Verilog - Excess5

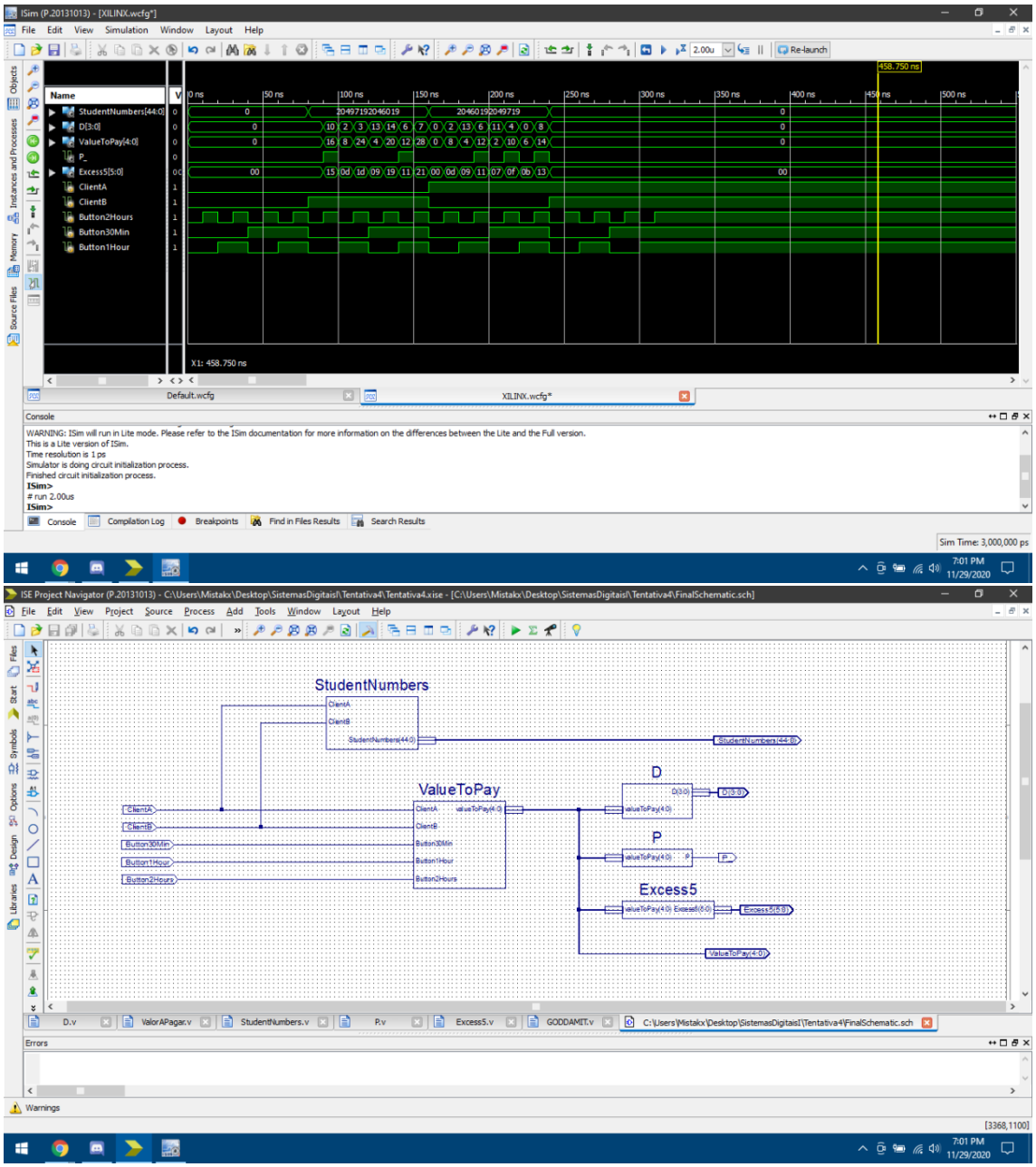
A natureza não dinâmica da máquina permite uma série de otimizações, pois o número de valores que o ValueToPay pode assumir é fixo:

- O excesso 5 pode ser pré-calculado.

ValueToPay	Excess5 (decimal)
10000 (16€)	21
01000 (8€)	13
11000 (24€)	29
00100 (4€)	9
10100 (20€)	25
01100 (12€)	17
11100 (28€)	33
01000 (8€)	13
00100 (4€)	9
01100 (12€)	17
00010 (2€)	7
01010 (10€)	15
00110 (6€)	11
01110 (14€)	19
00000 (Error)	0



Esquema elétrico Final/ Resultado Simulação



Conclusão

Ao longo do projeto e sua respetiva declaração de variáveis foi tido o cuidado de reservar apenas o espaço necessário, poupando o máximo de bits possível.

A implementação baseada em pré-cálculos foi também esta a melhor abordagem a nível de processamento e custo da máquina.

Anexos

```
module D(valueToPay, D);

   input[4:0] valueToPay;
   output [3:0] D; // Maximum D (When ValueToPay is 20): (14)10 - (1110)2
   reg [3:0] D;

   // Hugo - 2046019
   // Sérgio - 2049719

   // 2+0+4+6+0+1+9 + 2+0+4+9+7+1+9 = 54

   always@(valueToPay) begin

   // Value to Pay
      case(valueToPay)

      // 2 Euros (56%15)
         5'd2: D = 4'd11;
      // 4 Euros (58%15)
         5'd4: D = 4'd13;
      // 6 Euros (60%15)
         5'd6: D = 4'd0;
      // 8 Euros (62%15)
         5'd8: D = 4'd2;
      // 10 Euros (64%15)
         5'd10: D = 4'd4;
      // 12 Euros (66%15)
         5'd12: D = 4'd6;
      // 14 Euros (68%15)
         5'd14: D = 4'd8;
      // 16 Euros (70%15)
         5'd16: D = 4'd10;
      // 20 Euros (74%15)
         5'd20: D = 4'd14;
      // 24 Euros (78%15)
         5'd24: D = 4'd3;
      // 28 Euros (82%15)
         5'd28: D = 4'd7;

      // No clients selected is undesired behaviour
      // ClientA and ClientB is undesired behaviour
      // ClientA or ClientB, but no time button pressed is undesired behaviour
      5'd0: D = 4'b0;

      endcase

   end


endmodule
```

```
dule ValueToPay(ClientA, ClientB, Button30Min, Button1Hour, Button2Hours, valueToPay);

 // Inputs
 input ClientA;
 input ClientB;

 input Button30Min;
 input Button1Hour;
 input Button2Hours;


 // Output
 // The maximum value is 28 Euros in decimal, or 0b11100, so the maximum valueToPay size is 5 bits.
 output[4:0] valueToPay;
 reg [4:0] valueToPay;


 always@(Button30Min, Button1Hour, Button2Hours, ClientA, ClientB) begin

    // Client A

       if (ClientA && !ClientB && Button30Min && !Button1Hour && !Button2Hours) // Button30 - 2 Euros
          valueToPay = 5'd2;
       else if (ClientA && !ClientB && !Button30Min && Button1Hour && !Button2Hours) // Button1 - 4 Euros
          valueToPay = 5'd4;
       else if (ClientA && !ClientB && !Button30Min && !Button1Hour && Button2Hours) // Button2 - 8 Euros
          valueToPay = 5'd8;
       else if (ClientA && !ClientB && Button30Min && Button1Hour && !Button2Hours) // Button30 && Button1 - 6 Euros
          valueToPay = 5'd6;
       else if (ClientA && !ClientB && Button30Min && !Button1Hour && Button2Hours)  // Button30 && Button2 - 10 Euros
          valueToPay = 5'd10;
       else if (ClientA && !ClientB && !Button30Min && Button1Hour && Button2Hours) // Button1 && Button 2 - 12 Euros
          valueToPay = 5'd12;
       else if (ClientA && !ClientB && Button30Min && Button1Hour && Button2Hours) // Button 30 && Button1 && Button 2 - 14 Euros
          valueToPay = 5'd14;

    // Client B
       else if (ClientB && !ClientA && Button30Min && !Button1Hour && !Button2Hours) // Button30 - 4 Euros
          valueToPay = 5'd4;
       else if (ClientB && !ClientA && !Button30Min && Button1Hour && !Button2Hours) // Button1 - 8 Euros
          valueToPay = 5'd8;
       else if (ClientB && !ClientA && !Button30Min && !Button1Hour && Button2Hours) // Button2 - 16 Euros
          valueToPay = 5'd16;
       else if (ClientB && !ClientA && Button30Min && Button1Hour && !Button2Hours) // Button30 && Button1 - 12 Euros
          valueToPay = 5'd12;
       else if (ClientB && !ClientA && Button30Min && !Button1Hour && Button2Hours)  // Button30 && Button2 - 20 Euros
          valueToPay = 5'd20;
       else if (ClientB && !ClientA && !Button30Min && Button1Hour && Button2Hours) // Button1 && Button 2 - 24 Euros
          valueToPay = 5'd24;
       else if (ClientB && !ClientA && Button30Min && Button1Hour && Button2Hours) // Button 30 && Button1 && Button 2 - 28 Euros
          valueToPay = 5'd28;


    // No clients selected is undesired behaviour
    // ClientA and ClientB is undesired behaviour
    // ClientA or ClientB, but no time button pressed is undesired behaviour
       else
          valueToPay = 5'd0;

 end

dmodule
```

```
module StudentNumbers(ClientA, ClientB, StudentNumbers);

   input ClientA, ClientB;

   output[44:0] StudentNumbers;
   reg [44:0] StudentNumbers; // Concatenated numbers

   // The student numbers have 7 decimal digits.
   // Nowhere in the project requirements is it stated that it's expected in the future to be able to change the machine behaviour.
   // With this in mind, we can hard code the student numbers, and pre-calculate that the concatenation will take 45 bits in memory.
   // With this pre-calculation we save a maximum of 2 bits in memory (when all the digits are 9).
   // The hard-coded nature of the numbers also alows us to pre-calculate their concatenation value, saving processor time.

   // Hugo - 2046019
   // Sérgio - 2049719


   always@(ClientA, ClientB) begin



      if (ClientA && !ClientB) // ClientA
         StudentNumbers = 45'd20460192049719;

      else if (ClientB && !ClientA) // ClientB
         StudentNumbers = 45'd20497192046019;


         // No clients selected is undesired behaviour
         // ClientA and ClientB is undesired behaviour
      else
         StudentNumbers = 45'd0;

   end

endmodule
```

```
module P(valueToPay, P);

   input[4:0] valueToPay;
   output P; // P is 1 or 0
   reg P;

   // Hugo - 2046019
   // Sérgio - 2049719

   // 2+0+4+6+0+1+9 + 2+0+4+9+7+1+9 = 54


   always@(valueToPay) begin

   // Value to Pay
      case(valueToPay)

      // 2 Euros (54 + 2)
         5'd2: P = 1'd0;
      // 4 Euros (54 + 4)
         5'd4: P = 1'd0;
      // 6 Euros (54 + 6)
         5'd6: P = 1'd0;
      // 8 Euros (54 + 8)
         5'd8: P = 1'd0;
      // 10 Euros (54 + 1 + 0)
         5'd10: P = 1'd1;
      // 12 Euros (54 + 1 + 2)
         5'd12: P = 1'd1;
      // 14 Euros (54 + 1 + 4)
         5'd14: P = 1'd1;
      // 16 Euros (54 + 1 + 6)
         5'd16: P = 1'd1;
      // 20 Euros (54 + 2 + 0)
         5'd20: P = 1'd0;
      // 24 Euros (54 + 2 + 4)
         5'd24: P = 1'd0;
      // 28 Euros (54 + 2 + 8)
         5'd28: P = 1'd0;


      // No clients selected is undesired behaviour
      // ClientA and ClientB is undesired behaviour
      // ClientA or ClientB, but no time button pressed is undesired behaviour
         5'd0: P = 1'd0;

      endcase

   end
```

```
module Excess5(valueToPay, Excess5);

   input[4:0] valueToPay;
   output [5:0] Excess5; // Maximum (When ValueToPay is 28): (28 + 5)10 = (100001)2
   reg [5:0] Excess5;

   // Hugo - 2046019
   // Sérgio - 2049719

   // 2+0+4+6+0+1+9 + 2+0+4+9+7+1+9 = 54

   always@(valueToPay) begin

   // Value to Pay
      case(valueToPay)

      // 2 Euros (2 + 5)
         5'd2: Excess5 = 6'd7;
      // 4 Euros (4 + 5)
         5'd4: Excess5 = 6'd9;
      // 6 Euros (6 + 5)
         5'd6: Excess5 = 6'd11;
      // 8 Euros (8 + 5)
         5'd8: Excess5 = 6'd13;
      // 10 Euros (10 + 5)
         5'd10: Excess5 = 6'd15;
      // 12 Euros (12 + 5)
         5'd12: Excess5 = 6'd17;
      // 14 Euros (14 + 5)
         5'd14: Excess5 = 6'd19;
      // 16 Euros (16 + 5)
         5'd16: Excess5 = 6'd21;
      // 20 Euros (20 + 5)
         5'd20: Excess5 = 6'd25;
      // 24 Euros (24 + 5)
         5'd24: Excess5 = 6'd29;
      // 28 Euros (28 + 5)
         5'd28: Excess5 = 6'd33;

      // No clients selected is undesired behaviour
      // ClientA and ClientB is undesired behaviour
      // ClientA or ClientB, but no time button pressed is undesired behaviour
         5'd0: Excess5 = 6'd0;

      endcase

   end

endmodule
```