Sistemas Digitais

2020/2021

2º Projeto

Gerador de Código de Barras

(Máquina de Estados)

Membros do grupo:

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

Introdução

Neste trabalho foi-nos pedido que implementássemos um leitor de código de barras, sob a forma de uma máquina de estados finita, no software ISE Design Suite 14.7 da Xilinx. Esta máquina tem como objetivo fazer a leitura do código de barras e a gestão do troco.

Uma vez que já sabemos os respetivos valores que PG e I podem assumir, podemos abordar este problema com a máxima eficiência relativamente à gestão de memória.

MainStateMachine Module

Esta é a máquina de estados geral do programa, responsável por gerir a execução dos outros módulos.

Nesta máquina de estados, avançamos para o **estado 1** caso seja inserido algum valor.

Consoante o valor inserido, a máquina avança do **estado 1** para um estado que indica se é válido ou não, o valor em questão.

- **I Válido -** Neste trabalho I (valor inserido) só é valido caso seja maior que PG (valor a pagar).
- **I Inválido** - I menor que PG.

O resto do projeto funciona mediante o comportamento desta máquina.

Estado 0	Estado inicial, espera pela inserção de algum valor
Estado 1	Dinheiro inserido na máquina
Estado 2	Dinheiro inserido não é válido
Estado 3	Dinheiro inserido é válido
Estado 4	Deu dinheiro ao utilizador (se tal foi necessário)

Como uma das funcionalidades extra, em cada estado temos uma saída *"message"* que nos dá à saída uma mensagem associada a cada estado.

MoneyToGive Module

Este módulo é responsável por indicar à máquina de estados que devolve o dinheiro, o valor total de dinheiro a dar. Caso a máquina não tenha que devolver dinheiro, o módulo terá como output o valor 31 em vez do dinheiro a devolver, pois 31 é o estado que indica que não há dinheiro a devolver. 0 seria o estado onde a máquina que dá o troco espera por input.

Neste módulo recebemos como entrada o **mainState**, que é o estado interno da máquina de estados principal. O **mainState** ditará o comportamento que o módulo irá tomar, pois permite verificar se o dinheiro que foi inserido é ou não válido.

Estado 2	Dinheiro inserido inválido, retorna o dinheiro inserido
Estado 3	Dinheiro inserido válido, calcula e retorna o troco a dar

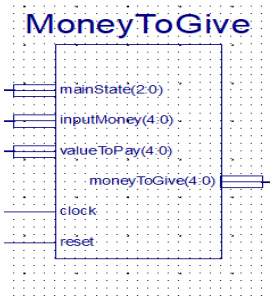


Figura 1.1- MoneyToGive Module

GiveMoneyStateMachine Module

Com a saída que vem do módulo anterior é acionado o comportamento da máquina.

Como isto é uma máquina que transita consoante os vários possíveis de estado temos o diagrama de estados em baixo demonstrado (*figura 1.2*).

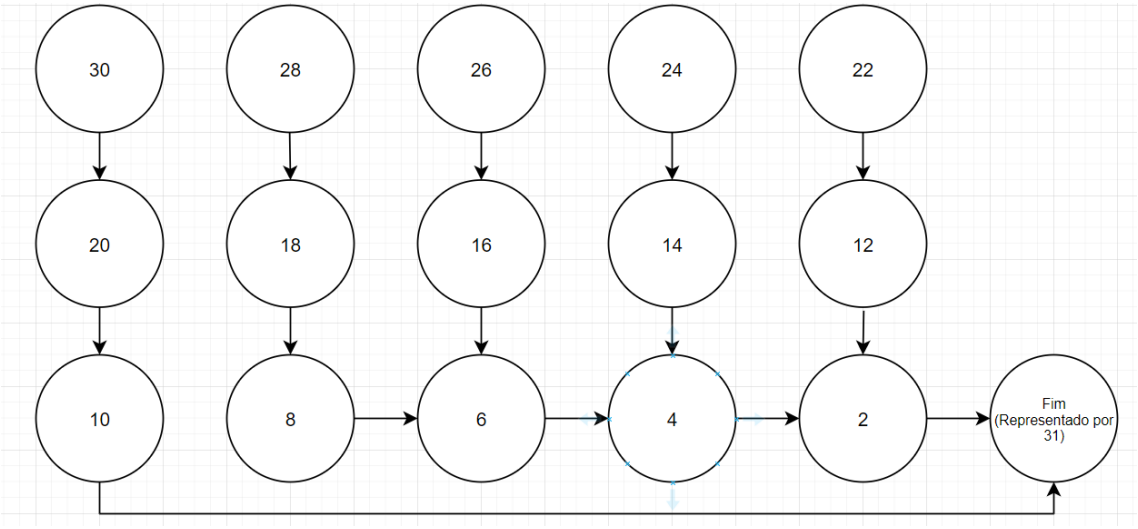


Figura 1.2- Diagrama de estados GiveMoneyStateMachine

Consoante a transição pelos diversos estados são acionadas as saídas que dão o respetivo troco em moedas de 2€ ou notas de 10€, como também a saída que indica o fim da respetiva operação. Após o fim da transação por parte da máquina de estados responsável

pelo dinheiro, a máquina de estados principal muda para o seu último estado, onde retorna uma mensagem de sucesso.

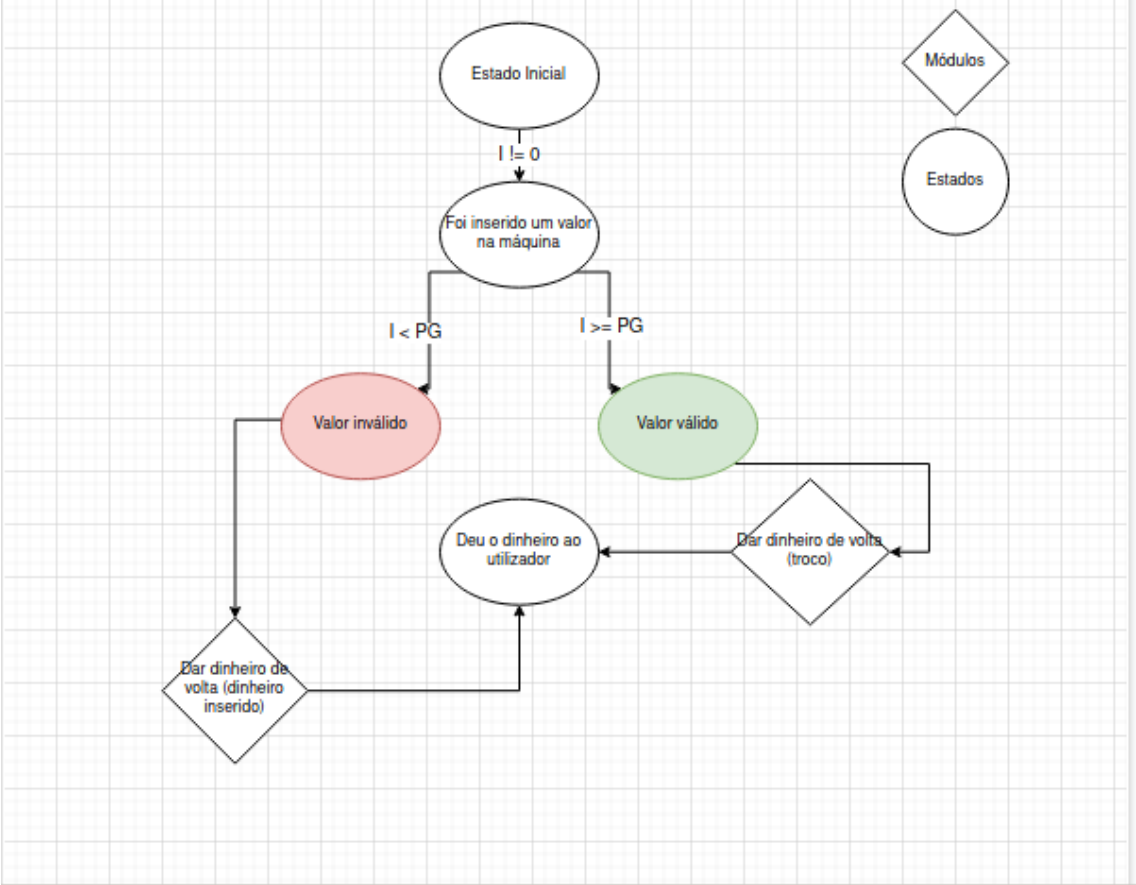
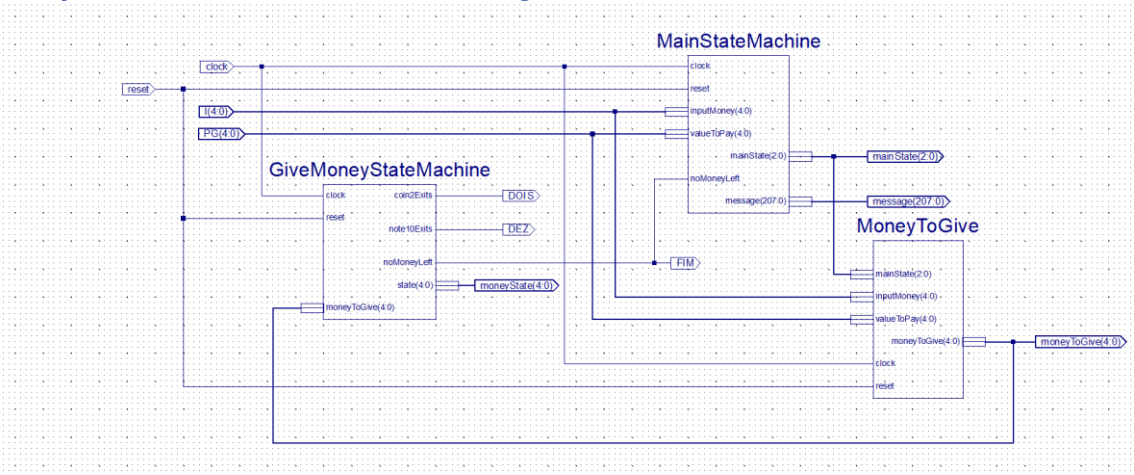
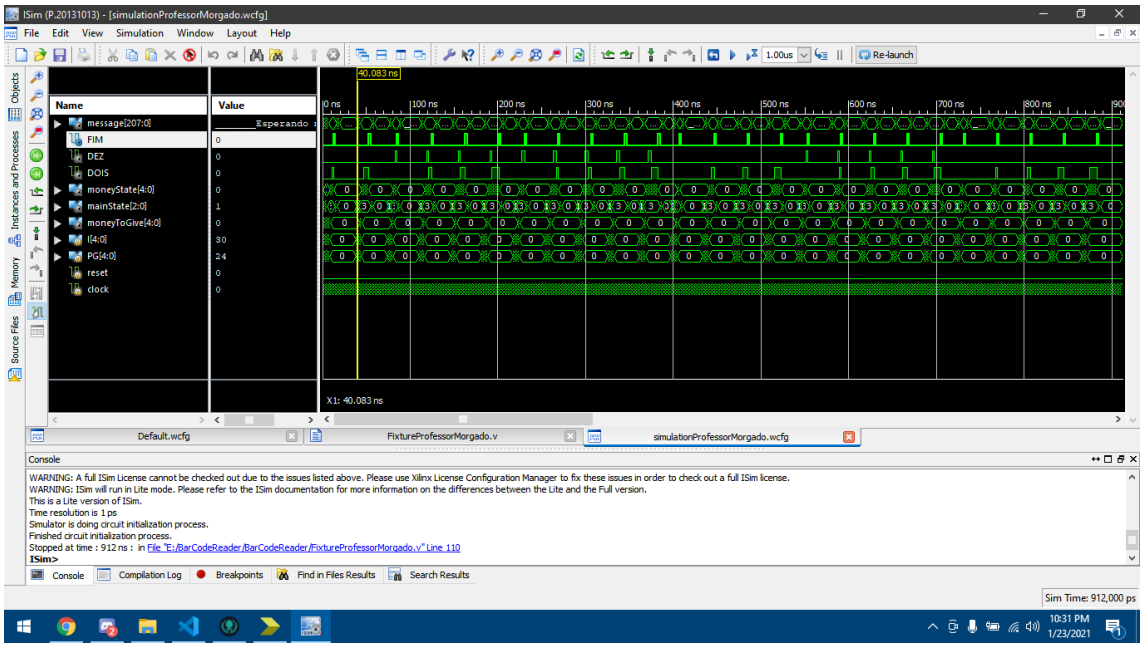
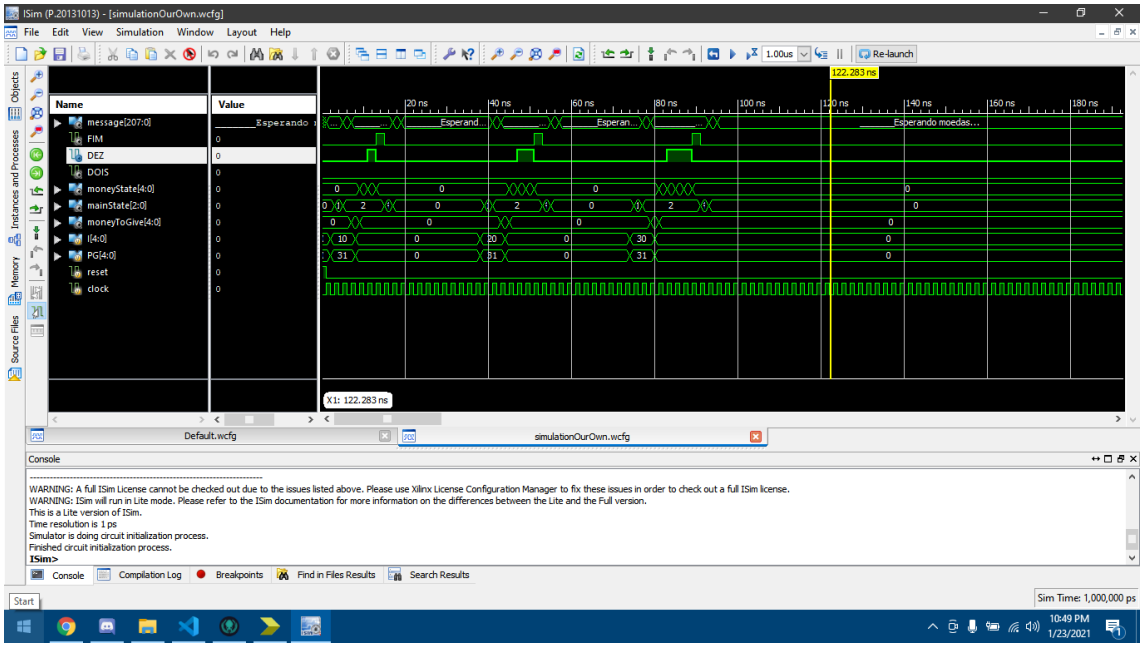


Figura 1.3- Esquema que representa o funcionamento da máquina

Esquema Elétrico e Simulação







ANEXO

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:12:00 01/10/2021
// Design Name:
```

```
// Module Name:    GiveMoneyStateMachine
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module GiveMoneyStateMachine(clock, reset, moneyToGive, coin2Exits, note10Exits,
noMoneyLeft, state);

    input clock;
    input reset;
    input [4:0] moneyToGive;

    output coin2Exits;
    output note10Exits;
    output noMoneyLeft;

    reg coin2Exits;
    reg note10Exits;
    reg noMoneyLeft;

    output state;
    reg [4:0] state; // 17 different states of money still to give.
```

```
        // State 0 - Waiting for machine start


        // State 31 - Since the machine won't need to give 31 euros,
        // we use this state as the end state, to represent that there is no more money left to
give.


        // State 30 = 30 Euros left to give
        // Since the maximum value that a user has to pay is 28,
        // this last state is not needed to solve the problem that was given.
        // Nonetheless, we decided to use this state in case the user enters 30 euros,
        // and an error ocurrs in the machine, giving the user it's 30 euros back.



        always @ (posedge clock or posedge reset)

                begin

                        if (reset)

                                begin
                                        coin2Exits = 0;
                                        note10Exits = 0;
                                        noMoneyLeft = 0;
                                        state = 5'd0;
                                end

                        else

                                case (state)
```

```
5'd0: // Waiting for start
        begin

                noMoneyLeft = 0;
                coin2Exits = 0;
                note10Exits = 0;
                state = moneyToGive;
        end

5'd31: // No money left to give
        begin
                noMoneyLeft = 1;
                coin2Exits = 0;
                note10Exits = 0;
                state = 5'd0;
        end

5'd2: // 2 Euros
        begin
                coin2Exits = 1;
                note10Exits = 0;
                state = 5'd31;
        end

5'd4: // 4 Euros
        begin
                coin2Exits = 1;
                note10Exits = 0;
                state = 5'd2;
        end
```

```
5'd6: // 6 Euros
        begin
                coin2Exits = 1;
                note10Exits = 0;
                state = 5'd4;
        end

5'd8: // 8 Euros
        begin
                coin2Exits = 1;
                note10Exits = 0;
                state = 5'd6;
        end

5'd10: // 10 Euros
        begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd31;
        end

5'd12: // 12 Euros
        begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd2;
        end

5'd14: // 14 Euros
        begin
                coin2Exits = 0;
```

```
                note10Exits = 1;
                state = 5'd4;
            end

        5'd16: // 16 Euros
            begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd6;
            end

        5'd18: // 18 Euros
            begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd8;
            end

        5'd20: // 20 Euros
            begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd10;
            end

        5'd22: // 22 Euros
            begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd12;
            end
```

```
5'd24: // 24 Euros
        begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd14;
        end

5'd26: // 26 Euros
        begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd16;
        end

5'd28: // 28 Euros
        begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd18;
        end

5'd30: // 30 Euros
        begin
                coin2Exits = 0;
                note10Exits = 1;
                state = 5'd20;
        end

endcase
```

```
                    end


endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    15:08:27 01/16/2021
// Design Name:
// Module Name:    MainStateMachine
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module MainStateMachine(clock, reset, noMoneyLeft, inputMoney, valueToPay, mainState,
message);


input clock;
input reset;
input noMoneyLeft;
input [4:0] inputMoney;
```

```
input [4:0] valueToPay;

// State 0 - Initial state, waiting for user input
// State 1 - Money was inserted in the machine
// State 2 - Money inserted in the machine was invalid
// State 3 - Money inserted in the machine was valid
// State 4 - If necessary, money was given back to the user

output [2:0] mainState;
reg [2:0] mainState; // 5 states. 3 bits necessary no store the mainState value.

output message;
reg [207:0] message; // The biggest message consists of 208 bits.

// Maximum state transitions along the program:
// Main machine: Waiting, Input money, Valid money (28â,¬ input to pay 2â,¬)
// Change machine: , 18â,¬, 8â,¬, 6â,¬, 4â,¬, 2â,¬, Ended giving money,
// Main machine

always @ (posedge clock or posedge reset)

        begin

                if (reset)

                        begin
                                message = "Inicializando a maquina.";
                                mainState = 3'd0;
                        end

                else
```

```
                case(mainState)

                    // Move to state 1 only after the user inputs some amount of
money
                    3'd0:
                        begin
                            message = "Esperando moedas...";
                            if (inputMoney != 0) mainState = 3'd1;
                        end

                    3'd1:
                        begin
                            message = "Valor inserido.";

                            // * Valid money input by the user
                            // Move to state 3 only if the user inputs
enough money to pay.
                            if (inputMoney >= valueToPay) mainState =
3'd3;

                            // * Invalid money input by the user
                            // Move to state 2 if the user hasn't input a
valid money amount
                            else mainState = 3'd2;

                        end

                    // Move to state 4 only after the money state machine has
finished giving the money back
                    3'd2:
                        begin
                            message = "Valor invalido.";
```

```
                        if (noMoneyLeft) mainState = 3'd4;
                    end

                // Move to state 4 only after the money state machine has
finished giving the change
                3'd3:
                    begin
                        message = "Valor valido.";
                        if (noMoneyLeft) mainState = 3'd4;
                    end

                3'd4:
                    begin
                        mainState = 3'd0;
                        message = "Obrigado pela preferencia.";
                    end

            endcase

        end

endmodule
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    00:20:33 01/21/2021
// Design Name:
// Module Name:    MoneyToGive
// Project Name:
```

```
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module MoneyToGive(clock, reset, mainState, inputMoney, valueToPay, moneyToGive);
	input clock;
	input reset;
	input [2:0] mainState;
	input [4:0] inputMoney;
	input [4:0] valueToPay;

	output [4:0] moneyToGive;
	reg [4:0] moneyToGive;

	always @ (posedge clock or posedge reset)

		begin

			// If the main state machine still hasn't validated any input, this module isn't called, and outputs X.
			// While this machine outputs 0 the change machine keeps waiting for input.
			// With this in mind, we need to reset this machine at the beginning, so the change machine waits for input.
			if (reset) moneyToGive = 0;
```

```
                else
                        begin

                                // If input was invalid, return the money input by the
user
                                if (mainState == 3'd2) moneyToGive = inputMoney;

                                // If input was valid, calculates and gives the change
                                else if (mainState == 3'd3)
                                        begin

                                                // If the input money is the same as
value to pay, the machine doesn't give any change, ending the process.
                                                // Ending the process isn't setting
moneyToGive to 0, because 0 is the state reserved for waiting for input.
                                                // Instead we change the state to 31.

                                                if ( (inputMoney == valueToPay) &&
(inputMoney != 0) && (valueToPay != 0)) moneyToGive = 5'd31;

                                                else moneyToGive = inputMoney -
valueToPay;

                                        end

                        end


        end

endmodule
```