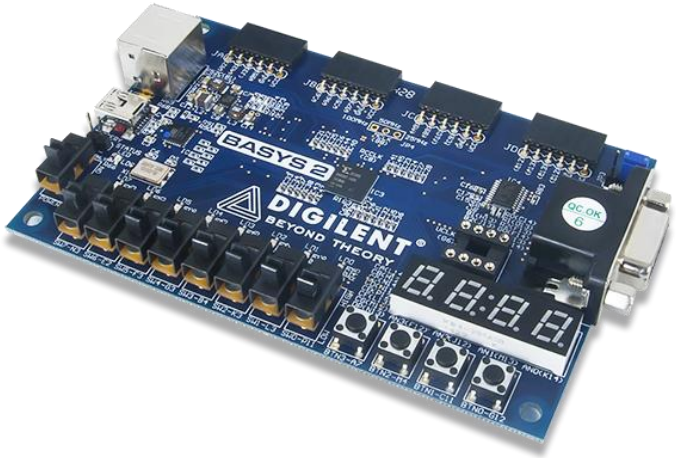
TÉCNICO LISBOA

Eletrónica II

3º TRABALHO DE LABORATÓRIO – **RELATÓRIO**

CIRCUITOS DIGITAIS

GRUPO 6



Identificação dos Alunos:

Nome: **Matilde Pereira Moreira** Número: **84137**
Nome: **João Pedro Costa Luís Cardoso** Número: **84096**
Nome: **Bernardo Miguel Gonilho** Número: **84016**

Docente: **Dinarte Vasconcelos**

1ºSemestre 2018-2019

CONTEÚDO

INTRODUÇÃO

No âmbito da disciplina de Eletrónica II e tendo em conta o terceiro trabalho experimental, pretende-se implementar em FPGA (na placa *Basys 2*), uma máquina de estados que é responsável pelo controlo de uma máquina de lavar a roupa, através da utilização da linguagem de descrição de h*ardware* Verilog.

O software utilizado para o desenvolvimento do programa é o ISE WebPack 14.2 [1] fornecido pelo fabricante da FPGA, XILINX e que se encontra instalado nos computadores do laboratório de Eletrónica II. De modo a simular a máquina de estados, sem ser necessária a própria FPGA, recorre-se a um simulador interno do ISE WebPack 14.2, ISIM.

Sucintamente, existem 6 pontos essenciais a destacar neste trabalho experimental e que, por sua vez, irão ser analisados pormenorizadamente nos capítulos seguintes:

- **Noções Básicas de Verilog** dadas em aulas teóricas e vistas em documentos publicados na página da disciplina, nomeadamente o ficheiro: PetervrIK.pdf. [2]
- **Conhecimento da FPGA – Basys2** que é usada em laboratório.
- **Dimensionamento e Funcionamento da máquina de Estados a implementar.**
- **Código em Verilog do módulo principal, do secundário e do testbench.**
- **Implementação do código na Basys2.**
- **Simulações e análise dos resultados obtidos.**

[1] **ISE WebPack 14.2 :** https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools/archive.html
[2] **PetervrIK:** https://fenix.tecnico.ulisboa.pt/downloadFile/282093452051899/PetervrlK.pdf

NOÇÕES BÁSICAS DE VERILOG

De modo a desenvolver o código, para implementar a máquina de estados mencionada na introdução, surge a necessidade de abordar alguns conceitos da linguagem de *hardware* Verilog, que são usados não só no módulo principal (**washing_machine.v**), como também no módulo secundário (**contagem.v**).

module/endmodule

- É a unidade fundamental de design em Verilog.
- Todo o código deve ser estruturado neste tipo de unidade.
- Dentro de um módulo são declarados:
  1. os *ports*;
  2. os tipos de dados utilizados;
  3. a descrição da funcionalidade do circuito (estrutural ou comportamental);
  4. a possibilidade de incluir especificações de temporização;
- Declaração de um módulo:

```
module nome_modulo (port list);
    input port_list_input;
    output port_list_output;
        ... código ...
endmodule
```

Ports (input e output)

- São os meios primários de comunicação entre o módulo e toda a envolvente, nomeadamente a FPGA.
- Existem 3 tipos fundamentais de *ports* em Verilog: *ports* de entrada (*input*), de saída (*output*) e de entrada-saída (*inout*).
- **Exemplos:**

```
output [2:0] Out; // saída de 3 bits
input In1, In2; //2 entradas de 1 bit
```

begin / end

- Utilizado para iniciar uma sequência de comandos.
- **Sintaxe e Semântica:**

```
begin
    comando1;
    comando2;
        (...)
    comando N;
end
```

reg

- Sinais deste tipo podem representar registos e são usados quando se deseja guardar um valor até que outro valor seja enviado para esse sinal ou não.
- **Exemplo :**

```
reg [1:0] estados; //estados é um
                   registo de 2 bits
```

wire

- Sinais do tipo wire representam fios ou barramentos (vários bits) e servem para transportar um valor lógico desde a origem que o produz.
- **Exemplo:**

```
wire clock; //fio de 1 bit
```

assign

- É usado para "atribuição contínua" ou em outras palavras para modelar lógica combinatória.
- Como a palavra contínua sugere, os sinais envolvidos nas instruções de atribuição são monitorizados continuamente e quanto há mudanças as alterações são refletidas instantaneamente.

parameter

- Define uma constante que pode ser definida quando se instancia um módulo.
- Para além de aumentar a legibilidade do código, também torna a descrição genérica, pois funciona como rótulo.
- **Exemplo:**
  parameter WIDTH = 32;

case/endcase

- Permite uma ramificação de múltiplos caminhos baseada na comparação da expressão com uma lista de opções de casos.
- O Default é usado para descrever uma ação padrão, ou seja, caso a entrada não seja nenhuma das previstas pelo programador, um código "padrão" também definido pelo programador será executado, contudo o seu uso não é necessariamente obrigatório, às vezes.
- Declaração de um case:

```
case (expressão)
        case_opcao1:
                begin
                        ... código...
                end
        ... mais blocos de instrução case consoante os casos...
        default:
                begin
                        ...código...
                end
endcase
```

always@

- Utilizado para realizar instruções sempre que houver mudança em alguma entrada/saída pré-determinada.
- **posedge** = o bloco always executará na subida do clock;
- **negedge** = o bloco always executará na descida do clock;
- Blocos combinatórios, sequenciais e síncronos
- **Sintaxe e Semântica:**
  always@(parâmetros de sensibilidade)

Operadores Lógicos

- Quando estes operadores são utilizados, retornam um valor de um bit que representa falso ou verdadeiro, isto é, 0 ou 1, respetivamente.

! – negação

&& - "and"

|| - "or"

if/else

- Executa uma instrução ou bloco de instruções dependendo do resultado da expressão de condição no if;
- Se as expressões condicionais em todas as if's forem avaliadas como falsas, as declarações no bloco else são executadas.

```
if (condição) begin
    ...código...
end
else begin
    ... código ...
end
```

RTL Assignments

- **Atribuições procedimentais (blocking) –** avaliadas em série.
  Exemplo:

```
y = a+b;
```

- **Atribuições non-blocking –** avaliadas em paralelo;
  Exemplo:

```
y <= a+b;
```

CONHECIMENTO DA FPGA – BASYS2

A placa Basys2 é uma plataforma de desenvolvimento e implementação de circuitos digitais reais. É construída ao redor de um *Xilinx Spartan-3E Field Programmable Gate Array* (FPGA) e de um controlador *USB Atmel AT90USB2*, a placa Basys2 fornece hardware adequado para receber circuitos, que vão desde dispositivos de lógica básica a controladores complexos.

A Basys 2 é conectada a um computador via conexão USB e depois de ligada, a FPGA deve ser configurada antes de executar qualquer função. Durante a configuração no ISE WebPack 14.2, um arquivo .bit é gerado, caso o programa não apresente nenhum erro ou warning que o impeça. De seguida transfere-se esse arquivo .bit para os registos dentro do FPGA para definir as funções lógicas de circuitos.

É usado um programa da Digilent designado por *Adept* para configurar o FPGA com o arquivo .bit criado. Esse programa usa o cabo USB para transferir o arquivo selecionado pelo computador para a FPGA (via JTAG da FPGA que é uma porta de programação). Deve-se salientar, que a FPGA permanece configurada com o mesmo ficheiro .bit até que seja redefinido por um evento de ciclo de energia. A ROM Flash retém o bit até que ele seja reprogramado.

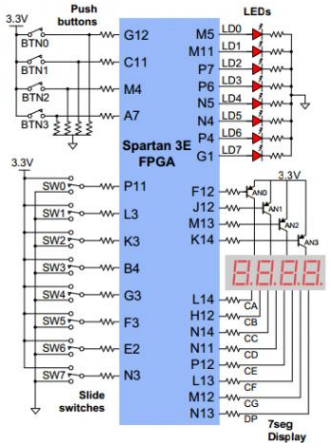


FIGURA 1 : BASYS2 I/O CIRCUITOS

Relativamente ao terceiro laboratório, são usados os 4 botões (*Push Buttons)* e um dos switches slide que representa o Reset.

Recorrendo ao ficheiro wash basys2.ucf fornecido pelo docente, na página da disciplina, sabe-se quais são os pinos da Basys2, que correspondem aos inputs e outputs da máquina de estados a implementar. Por outras palavras, as entradas start, full e empty dos estados correspondem aos pinos A7, M4, G12 da FPGA, respetivamente.

Os Leds da FPGA servem para visualizar em que estado nos encontramos, isto é, os estados Ready, Water_in, Wash, Drain e Speed correspondem aos Leds G1, P4, N4, N5, P6 da Basys2, respetivamente.

Relativamente à saída heat_r, esta é visível na FPGA através de um LED (P7) que só se encontra aceso quando nos encontramos no estado Wash (LED N4 aceso) e se se premir o botão do cold (C11)

DIMENSIONAMENTO E FUNCIONAMENTO DA MÁQUINA DE ESTADOS

A máquina de estados é inicializada no estado ***Ready.*** É a entrada "Start" que vai determinar se a máquina se mantém nesse estado ou se avança para o estado seguinte. Esta entrada simboliza o início do funcionamento da máquina de lavar a roupa. Se a entrada "Start" se encontra a '1', então a máquina passa para o estado seguinte, ***Water_In***, ao passo que se "Start" estiver a '0', a máquina mantém-se no estado atual, ***Ready***.

Uma vez no estado ***Water_in*** (água a entrar na máquina de lavar a roupa), a passagem para o próximo estado será determinada pela entrada "Full", que significa que a água atingiu o nível adequado. Quando a entrada "Full" estiver a '1', a máquina de estados passa para o estado ***Wash***. Caso a entrada "Full" se encontre a '0', isto é, não seja ativada, o estado permanece no atual, ***Water_in***.

Quando a máquina de estados estiver no estado ***Wash*** (que simboliza que a máquina de lavar a roupa está a lavar), é inicializado um contador que vai contar 4 segundos (equivale a 2.0000.0000 ciclos para uma frequência de 50MHz) até a máquina passar para o estado seguinte, ***Drain***. Sempre que a máquina se encontra no estado ***Wash***, e for ligada a entrada "cold" (temperatura abaixo do valor programado), a saída "heat_r" / heat_water é acionada (que faz com que a temperatura da água aumente).

O estado ***Drain*** simboliza a água a baixar de nível, ou seja, a máquina de lavar está a esvaziar a água. A partir deste estado, é a entrada "Empty" (máquina já não tem água) a responsável pela determinação do estado seguinte. Se a entrada "Empty" se encontrar a '1' a máquina passa para o estado ***Speed***. Se estiver a '0', o estado ***Drain*** é mantido. Uma vez no estado ***Speed***, é iniciado um novo contador que vai contar até 2 segundos (1.0000.0000 ciclos para uma frequência de 50MHz). Após esse tempo, a máquina volta ao estado ***Ready***.

Em qualquer estado em que a máquina se encontre, se se ativar o botão do "Reset", a máquina de estados passa imediatamente para o estado inicial, ou seja, o estado ***Ready***. Na figura 2 está representado o diagrama da máquina de estados implementada em laboratório tendo em conta os parâmetros mencionados.

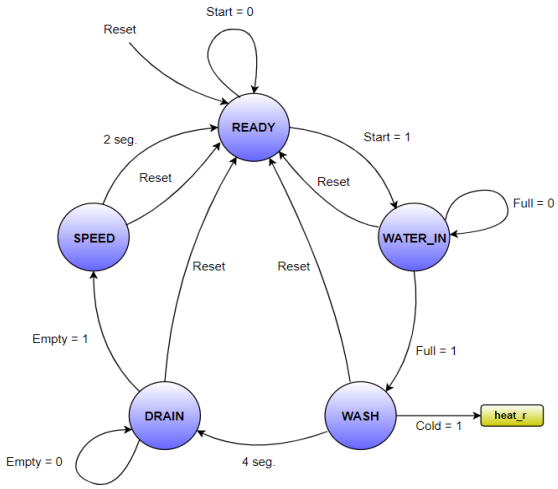


FIGURA 2 : DIAGRAMA DA MÁQUINA DE ESTADOS IMPLEMENTADA

CÓDIGO DESENVOLVIDO EM VERILOG

1. MÓDULO PRINCIPAL – WASHING_MACHINE.V

No módulo principal encontra-se todo o código referente à caracterização da máquina de estados a implementar, quer isto dizer que é nesta parte do ficheiro em Verilog, por exemplo, que se define o número de bits a usar para definir os estados ou que se decide o que acontece com os estados quando se prime o switch do reset ou quando determinadas entradas são ativadas.

```
//////////////////////////////////////////////////////////////////////////////////
// Universidade: Instituto Superior Técnico
// Curso: Engenharia Electrotécnica e de Computadores
//
// Disciplina: Eletronica II
// Ano-Semestre: 2018/2019 - 1ºSemestre
//
// Trabalho realizado por:
//                            Matilde Moreira, nº84137
//                            João Pedro Cardoso, nº 84096
//                            Bernardo Gonilho, nº 84016//
// Docente: Dinarte Vasconcelos
// Data de criação: 10:54:56 06/12/2018
// Nome do Projeto: washing_machine
//
//
// Descrição: Neste projeto pretende-se implementar em FPGA uma máquina de
// estados que controla uma máquina de lavar roupa através da linguagem de
// descrição de hardware Verilog
//////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps

/* Modulo principal que controla a máquina de estados*/
module
washing_machine(clk,reset,start,full,cold,empty,ready,water_in,wash,drain,s
peed,heat_r);
/* Definicao dos inputs e outputs do ficheiro wash_basys2.ucf */
input clk,reset,start,full,cold,empty;
output ready,water_in,wash, drain, speed, heat_r;
/* A máquina tem 5 estados logo precisa-se de um registo de 3 bits para
poder definir o estado */
reg [2:0] state,nxt_st;

/* O enable funciona como um switch que controla o counter presente no
modulo contagem */
reg enable;

/* contador para contar o número de ciclos (f=1/T=1/(5ns)=50MHz) -->nºbits
= log2(50MHz) aprox. 28bits */
wire [27:0]counter;

/* numero de ciclos correspondentes a 4 segundos (200000000), para uma f=50
MHz - no testbench usou-se como exemplo 28'd10 */
parameter wash_count =200000000;

/* numero de ciclos correspondentes a 2 segundos (100000000), para uma f=50
MHz - no testbench usou-se como exemplo 28'd5 */
parameter speed_count=100000000;
/* atribuição de nomes a cada estado */
parameter SReady = 3'b000;
parameter SWater_In = 3'b001;
```

```
parameter SWash = 3'b010;
parameter SDrain = 3'b011;
parameter SSpeed = 3'b100;

/* Invoca o modulo contagem inicializando os inputs e outputs desse módulo
para efetuar a contagem ao longo do programa */
contagem clock (.clk(clk),.enable(enable),.contador(counter));


/* Consoante o estado se é Wash ou Speed e se o counter não atingiu a
contagem pretendida o botão enable encontra-se ativo*/
always@(state or counter)
begin
      if ((state == SWash && counter < (wash_count)) ||(state == SSpeed &&
counter < (speed_count))) begin
            enable <= 1;
            end
      else enable <= 0;
end

/* Decisao dos estados seguintes consoante as entradas dos estados atuais*/
always@(state or start or full or empty or counter)
begin
      case(state)
            SReady: begin
                  if (start)
                        nxt_st <= SWater_In;
                  else
                        nxt_st <= SReady;
            end
            SWater_In: begin
                  if (full)
                        nxt_st <=SWash;
                  else
                        nxt_st <= SWater_In;
            end
            SWash: begin
                  if (counter == wash_count)
                        nxt_st <= SDrain;
                  else
                        nxt_st <= SWash;
            end
            SDrain: begin
                  if (empty)
                        nxt_st <= SSpeed;
                  else
                        nxt_st <= SDrain;
            end
            SSpeed: begin
                  if (counter == speed_count)
                        nxt_st <= SReady;
                  else
                        nxt_st <= SSpeed;
            end
            default: begin
                  nxt_st <=SReady; /* como usamos 3 bits e so temos 5
estados temos de ter em conta para os estados estarem bem definidos */
            end
      endcase
end
```

```
/* se ativarmos o switch do reset o estado passa a ser o Ready caso
contrario atualiza-se o estado para o estado seguinte*/
always@(posedge clk or posedge reset)
begin
      if (reset)begin
            state <= SReady;
            end
      else state <= nxt_st;
end

/* consoante um dado estado o led respetivo a esse mesmo estado acende-se*/
assign ready = (state==SReady);
assign water_in = (state==SWater_In);
assign wash = (state==SWash);
assign drain = (state==SDrain);
assign speed = (state==SSpeed);

/* no caso de nos encontrarmos no estado wash e se se premir o botão cold
acende-se o led correspondente a heat_r */
assign heat_r = (state==SWash && cold==1);

endmodule
```

2. MÓDULO SECUNDÁRIO – CONTAGEM.V

```
/* Modulo secundario para a contagem dos ciclos */
module contagem (clk,enable,contador);
input clk;
input enable;
output reg [27:0] contador;

/* A cada flanco ascendente do relógio caso nos encontremos no estado Wash
e Speed e não tenhamos atingido o numero de ciclos necessários
      o contador incrementa, caso contrário está a zero */
always@(posedge clk)
begin
      if(enable == 1) contador <= contador + 1;
      else contador <= 0;
end

endmodule
```

3. FICHEIRO PARA SIMULAÇÃO – TESTBENCH.V

```
`timescale 1 ns / 1 ns
//////////////////////////////////////////////////////////////////////////////////
///////
// Universidade: Instituto Superior Técnico
// Curso: Engenharia Electrotécnica e de Computadores
//
// Disciplina: Eletronica II
// Ano-Semestre : 2018/2019 - 1ºSemestre
//
// Trabalho realizado por:
//                                  Matilde Moreira, nº84137
//                                  João Pedro Cardoso, nº 84096
//                                  Bernardo Gonilho, nº 84016
//
// Docente: Dinarte Vasconcelos
// Data de criação: 10:54:56 06/12/2018
// Nome do Projeto: testbench.v
//
// Notes:          Clocked at 50 MHz (T = 20 ns)
//                 To reduce simulation time, use smaller time delays in the
//                 state transitions Wash->Drain and Speed->Ready.
//                 Example: Wash->Drain 160 ns      Speed->Ready 80 ns
//////////////////////////////////////////////////////////////////////////////////

module testbench;
      // Inputs da maquina de estados
      reg reset;
      reg clk;
      reg start;
      reg full;
      reg empty;
      reg cold;
      // Outputs da maquina de estados
      wire ready;
      wire water_in;
      wire heat_r;
      wire wash;
      wire drain;
      wire speed;


      // Instantiate the Unit Under Test (UUT)
      washing_machine uut (
            .reset(reset),
            .clk(clk),
            .start(start),
            .full(full),
            .empty(empty),
            .cold(cold),
            .ready(ready),
            .water_in(water_in),
            .heat_r(heat_r),
            .wash(wash),
            .drain(drain),
            .speed(speed)
      );

   always
   #10.0 clk = ~clk;
```

```
   initial begin
      // Inicializa os Inputs para que o estado inicial seja o Ready
            reset = 1;
            clk = 1;
            start = 0;
            full = 0;
            empty = 0;
            cold = 0;

      // Display initial message
      $display("Washing Machine FSM - TestBench 1.0");
      //Verifica se o estado inicial é o Ready e caso não seja é presentada
uma mensagem que indica o erro e a simulação termina
            @(negedge clk)
                  if(uut.state != 3'b000) begin
                        $display(" State Error [%b] ,
Time=%t",uut.state,$time*10);
                        $finish;
                        end
                  else
                        $display("Ready State OK");
      //Alteração da entrada para que no outro momento start seja colocado
a 1 e passe para o próximo estado
      @(negedge clk)
                  reset=0;
            @(negedge clk)
            begin
                  start=1;
                  $display("Time = %t ns, start assigned '1'", $time*10);
            end

      //Verifica se o estado é o Water_In e caso não seja é apresentada uma
mensagem que indica o erro e a simulação termina
            @(negedge clk)
                  if(uut.state != 3'b001) begin
                        $display(" State Error [%b] ,
Time=%t",uut.state,$time*10);
                        $finish;
                        end
                  else
                        $display("Water_In State OK");

      //Alteração da entrada para que no outro flanco descendente do clk,
full seja colocado a 1 e passe para o próximo estado
            @(negedge clk)
                  start = 0;

            @(negedge clk)
                  full = 1;
                  $display("Time = %t ns, full assigned '1'", $time*10);

      //Verifica se o estado é o Wash e caso não seja é apresentada uma
mensagem que indica o erro e a simulação termina
            @(negedge clk)
                  if(uut.state != 3'b010) begin
                        $display(" State Error [%b] ,
Time=%t",uut.state,$time*10);
                        $finish;
                        end
                  else
                        $display("Wash State OK");
```

```
            @(negedge clk)
                    cold = 1;
            $display("Time = %t ns, cold assigned '1'", $time*10);

      //Verifica se o heat_r foi accionado e caso não seja é apresentada
uma mensagem que indica o erro e a simulação termina
            @(negedge clk)
                  if(~uut.heat_r) begin
                        $display(" Heat Water Error [%b] ,
Time=%t",uut.state,$time*10);
                        $finish;
                        end
                  else
                        $display("Heat Water OK");

            @(negedge clk)
                  cold = 0;
                  $display("Time = %t ns, cold assigned '0'", $time*10);

      //Verifica se o heat_r foi accionado e caso seja é apresentada uma
mensagem que indica o erro e a simulação termina
            @(negedge clk)
                  if(uut.heat_r) begin
                        $display(" Heat Water Error [%b] ,
Time=%t",uut.state,$time*10);
                        $finish;
                        end
                  else
                        $display("Heat Water OK");

            @(negedge clk)
                  full = 0;

            while(drain != 1) begin #10; end
4           //Verifica se o estado é o Drain e caso não seja é apresentada
uma mensagem que indica o erro e a simulação termina
                  @(negedge clk)
                        if(uut.state != 3'b011) begin
                              $display(" Drain State Error [%b] ,
Time=%t",uut.state,$time*10);
                              $finish;
                              end
                        else
                              $display("Drain State OK");

                  @(negedge clk)
                        empty = 1;
                        $display("Time = %t ns, reset assigned '0'",
$time*10);

            //Verifica se o estado é o Speed e caso não seja é apresentada
uma mensagem que indica o erro e a simulação termina
                  @(negedge clk)
                        if(uut.state != 3'b100) begin
                              $display(" State Error [%b] ,
Time=%t",uut.state,$time*10);
                              $finish;
                              end
                        else
                              $display("Speed State OK");
                  @(negedge clk)
                        empty = 0;
```

```
            while(ready != 1) begin #10; end
                  $stop;
            end

      always@(posedge ready)
         $display("Time = %t ns, entering Ready state", $time*10);
      always @(posedge water_in)
         $display("Time = %t ns, entering Water In state", $time*10);
      always @(posedge wash)
         $display("Time = %t ns, entering Wash state", $time*10);
      always @(posedge drain)
         $display("Time = %t ns, entering Drain state", $time*10);
      always @(posedge speed)
         $display("Time = %t ns, entering Speed state", $time*10);
      always @(posedge heat_r)
         $display("Time = %t ns, heat_r asserted", $time*10);
      always @(negedge heat_r)
         $display("Time = %t ns, heat_r de-asserted", $time*10);

endmodule
```

IMPLEMENTAÇÃO NA BASYS2

Depois de gerar o ficheiro .bit e de enviar via USB para a FPGA, testou-se na placa acionando ou os botões (para ativar as entradas) ou o switch que representa o Reset e verificou-se as mudanças de estado representadas, teoricamente, na Figura 2, pela mudança dos LEDs.

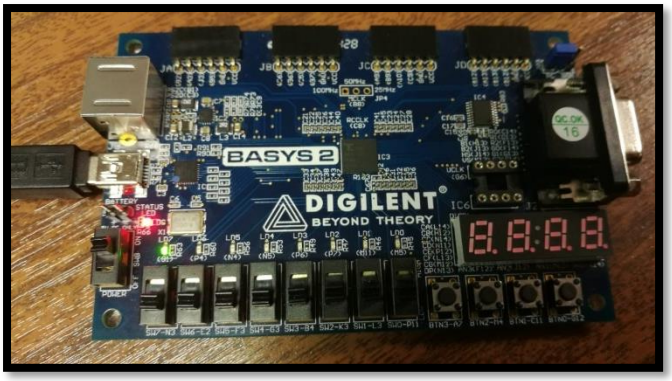


FIGURA 3 : ESTADO - READY (LED G1 ATIVO)

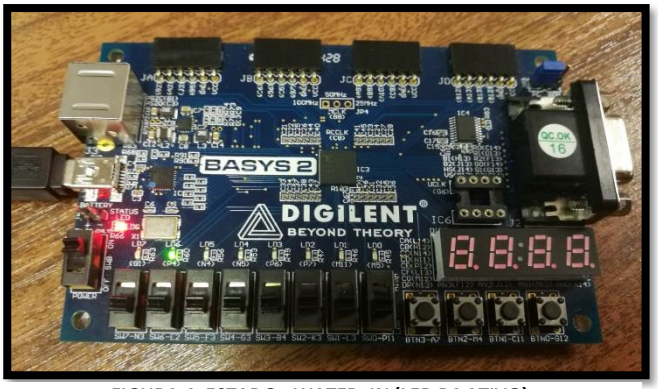


FIGURA 4: ESTADO - WATER_IN (LED P4 ATIVO)

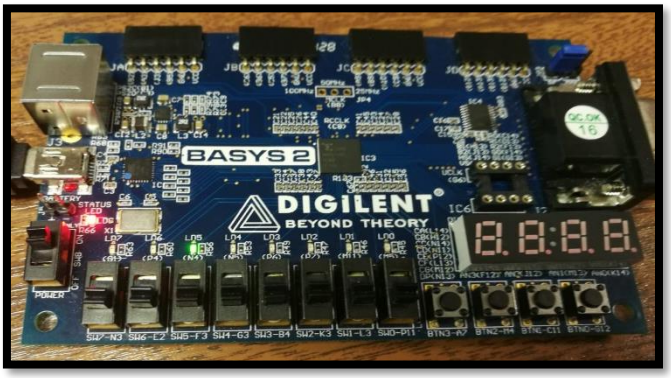


FIGURA 5 : ESTADO WASH (LED N4 ATIVO)

FIGURA 6 : ESTADO WASH COM A SAIDA HEAT_R ATIVA (LEDS N4 E P7 ACESOS)

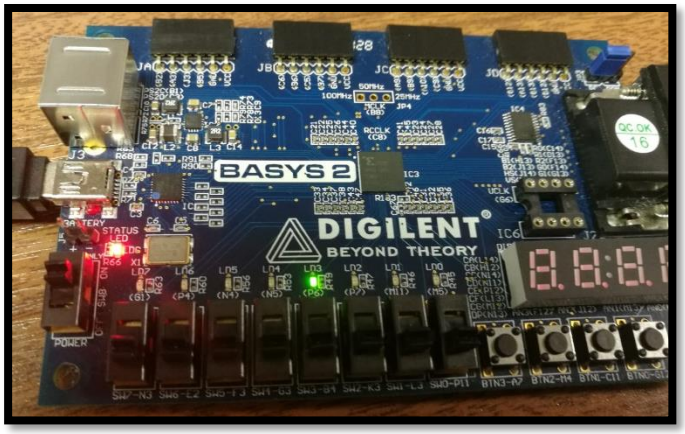


FIGURA 7 : ESTADO SPEED (LED P6 ACESO)

SIMULAÇÕES E ANÁLISE

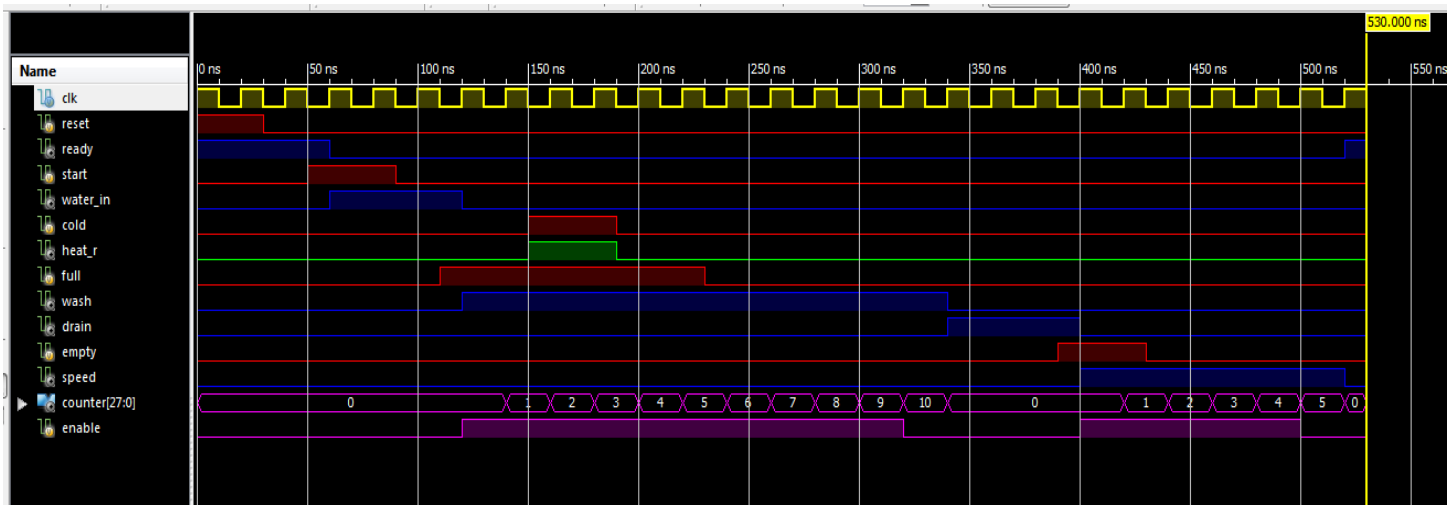


FIGURA 8 SIMULAÇÃO DA MÁQUINA DE ESTADOS NO XILINX

A simulação efetuada no Xilinx está representada na Figura 8 e para a sua melhor compreensão colocou-se a vermelho os sinais de entrada reset, start, full e empty, assim como o cold. A azul é possível verificar os estados ready, water_in, wash, drain e speed. A saída heat_r encontra-se assinalada a verde e o contador e o enable encontram-se a roxo.

De forma a evitar latches, perda de sincronismo e até mesmo instabilidade da máquina de estados, decidiu-se que as mudanças de estado iriam ocorrer em flancos ascendentes do relógio (clk) e que as mudanças dos valores das entradas ocorreriam nos flancos descendentes do relógio, isto é a justificação para o qual , quando à ativação de uma dada entrada (em negedge clk) o próprio estado não muda automaticamente.

Como é possível verificar pela Figura 8 e pelo código: testbench.v, a simulação começa no estado **ready**, que corresponde a exclusivamente reset = '1' e todas as outras entradas a '0'.

No próximo ciclo do relógio, nomeadamente no flanco descendente, a entrada reset passa a '0'. Não se definiu a mudança de reset = '0' e start = '1' no mesmo ciclo de relógio de forma a evitar os problemas anteriormente mencionados. Isto implica que só no outro ciclo de relógio (negedge) é que start fica a '1'. Como os estados mudam a cada flanco ascendente implica que no próximo ciclo, os estados alteram-se, como previsto, ou seja, o estado passa para **water_in**.

No flanco descendente de clk seguinte a entrada start fica novamente a '0' e no posterior flanco descendente, a entrada 'full' fica ativada a '1'. Isto vai implicar que o estado se altere de **water_in** para **wash**, no próximo posedge clk.

Como se sabe no estado **wash** e até mesmo no **speed** a condicionante para o estado seguinte é o próprio contador e não uma entrada acionada por switch ou botões. Esse contador está sincronizado com o próprio relógio e só começa a sua contagem no flanco ascendente do relógio e um ciclo de relógio depois de ter a noção que se encontra num desses estados, caso contrário como é possível verificar o contador (counter) está a 0.

Como o número de ciclos a implementar na FPGA era 2.0000.0000 para **wash** e 1.0000.0000 para o estado **speed**, diminuiu-se o número de ciclos na simulação, dado que corre em ns e iria demorar bastante tempo até que mudasse de estado. Sendo assim para o estado **wash** considerou-se 10 ciclos e para o **speed** 5 ciclos, como é possível comprovar.

Durante o estado **wash** se se ativar o cold a '1' repara-se que a saída heat_r fica automaticamente ativa. A diferença relativamente às desfasagens anteriormente obtidas entre as mudanças de entradas e as de estado face à que se obteve para a heat_r prende-se com o facto da ativação do heat_r não estar necessariamente dependente do clock, pois como não se considerou heat_r sendo um estado, não o incluímos no nosso bloco always@ dependente do flanco ascendente do relógio, pelo que na simulação, o led teoricamente acender-se-ia mal o cold fosse premido.

Quando o counter atinge os 10 ciclos, o estado passa para **drain,** o enable e o contador encontram-se a '0', visto o estado atual não ser nenhum dos dois estados **wash** ou **speed.** Quando a entrada full passa a '1', no flanco ascendente do clk do ciclo seguinte, os estados alteram-se e passa para **speed.**

Como já foi referido, o estado **speed** é condicionado pelo contador, pelo que passado os 5 ciclos, o estado altera-se para **ready.** E a simulação termina dada a presença de: `$stop;`

É apresentada na figura seguinte uma versão completa da simulação , ISim, consoante a adição no ficheiro de testbench.v de mensagens de aviso e auxílio para a compreensão da mudança de estados e se essa alteração estava a correr como o que era previsto.

```
--------------------------------------------------------------------------------
This is a Full version of ISim.

# run 1000 ns
Simulator is doing circuit initialization process.
Washing Machine FSM - TestBench 1.0
Time =              0 ns, heat_r de-asserted
Finished circuit initialization process.
Time =              0 ns, entering Ready state
Ready State OK
Time =         500000 ns, start assigned '1'
Time =         600000 ns, entering Water In state
Water_In State OK
Time =        1100000 ns, full assigned '1'
Time =        1200000 ns, entering Wash state
Wash State OK
Time =        1500000 ns, cold assigned '1'
Time =        1500000 ns, heat_r asserted
Heat Water OK
Time =        1900000 ns, cold assigned '0'
Time =        1900000 ns, heat_r de-asserted
Heat Water OK
Time =        3400000 ns, entering Drain state
Drain State OK
Time =        3900000 ns, reset assigned '0'
Time =        4000000 ns, entering Speed state
Speed State OK
Time =        5200000 ns, entering Ready state
Stopped at time : 530 ns :  in File "F:/washing_machine/washing_machine/test_bench.v" Line 167
ISim>
```

FIGURA 9 : VERSÃO COMPLETA DO ISIM

CONCLUSÕES

O terceiro laboratório proporcionou um estudo aprofundado sobre a linguagem de descrição de *hardware* Verilog, bem como conceitos relacionados com circuitos digitais, nomeadamente FPGA's.

Ao longo de todo o código desenvolvido, para a posterior implementação ou até mesmo simulação, evitou-se a existência de elementos latch, isto é, elementos assíncronos, independentes do clock, dado que a máquina de estados para ser estável deve ser síncrona com o relógio e caso existisse um elemento latch o circuito tornar-se-ia instável. Para além disso, também era pedido que a máquina de estados fosse síncrona com o flanco ascendente de relógio da placa, isto é, que os eventos, nomeadamente a mudança de estados fosse dependente e alterada consoante o flanco ascendente do relógio (posedge clk) e caso a entrada do próximo estado se encontrasse ativa ('1').

Considera-se, portanto, que o projeto desenvolvido foi realizado com sucesso, como se pode verificar pelos resultados obtidos na implementação na Basys2, na análise do código desenvolvido, assim como nos comentários ao longo do mesmo e na própria simulação.