



ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ADEETC - ÁREA DEPARTAMENTAL DE ENGENHARIA DE  
ELETRÓNICA E TELECOMUNICAÇÕES E COMPUTADORES

---

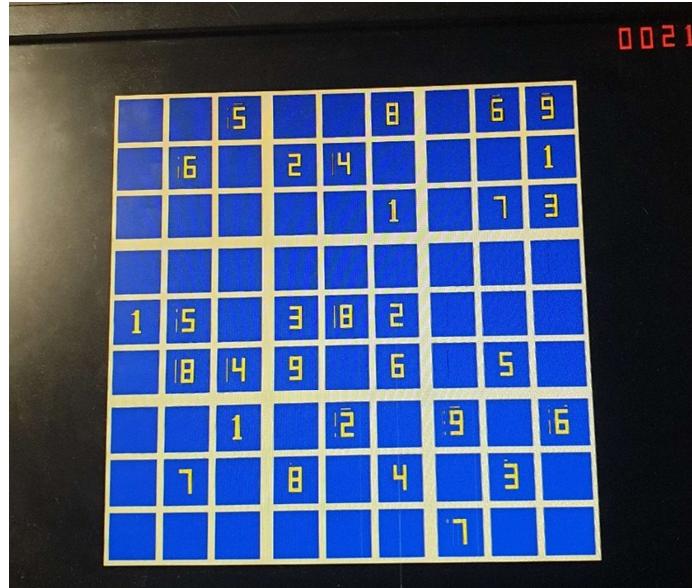
LEETC

Licenciatura em Engenharia Eletrónica, Telecomunicações  
Computadores

---

SEADP – Sistemas Eletrónicos Analógicos e Digitais Programáveis

### Jogo “sudoku” (VGA)



Grupo 3

André Moreira (47125)

João Maia (47130)

Rafael Carvalho (47663)

*Orientador*

---

*Professor [Doutor]* José Rocha

---

*fevereiro, 2022*



# Índice

<b>Índice .....</b>	<b>3</b>
<b>Lista de Figura.....</b>	<b>4</b>
<b>Introdução.....</b>	<b>6</b>
<b>1. Jogo Sudoku.....</b>	<b>7</b>
<b>2. Interface Gráfica .....</b>	<b>8</b>
<b>2.1. Varrimento .....</b>	<b>8</b>
<b>2.2. Desenho do sudoku .....</b>	<b>9</b>
<b>2.2.1 Moldura do sudoku.....</b>	<b>9</b>
<b>2.2.2 Desenho de cada Dígito.....</b>	<b>11</b>
<b>2.2.3 Registos, atualização de valores e verificação do sudoku.....</b>	<b>11</b>
<b>2.2.4 Níveis .....</b>	<b>13</b>
<b>2.2.5 Contador .....</b>	<b>15</b>
<b>3. Display de 7 segmentos .....</b>	<b>17</b>
<b>4 Modulo de Ligação.....</b>	<b>18</b>
<b>5 Atribuição de Pinos à FPGA/Controlo Sudoku.....</b>	<b>20</b>
<b>6. Análise do esquema elétrico. ....</b>	<b>21</b>
<b>7. Melhorias e desafios a futuros colegas .....</b>	<b>24</b>
<b>8. Conclusão .....</b>	<b>25</b>
<b>9. Bibliografia .....</b>	<b>26</b>
<b>ANEXO A .....</b>	<b>27</b>

# **Lista de Figura**

Figura 1 - Exemplo de um Sudoku resolvido .....	7
Figura 2 - Varrimento dos Pixelis.....	8
Figura 3 - Processo responsável pelo varrimento .....	8
Figura 4 - Clock de 25MHz.....	9
Figura 5 - Background do tabuleiro e Bordas.....	9
Figura 6 – Desenho das linhas.....	10
Figura 7 - Tabuleiro.....	10
Figura 8 - Desenho dos números .....	11
Figura 9 - Registos para guardar os números .....	12
Figura 10 - Verificação se o Sudoku foi preenchido corretamente tendo em conta o nível .....	12
Figura 11 - Atualização de um dígito .....	13
Figura 12 - Consoante o valor do dígito devolve o dígito a ser desenhado.....	13
Figura 13 - Atribuição do valor dos dígitos consoante o nível.....	13
Figura 14 - Desenho de um dígito dependente do mapa .....	14
Figura 15 - Nível 0 .....	14
Figura 16 - Nível 1 .....	14
Figura 17 - Clock de 1Hz .....	15
Figura 18 - Incremento do algarismo dos milhares, dezenas, centenas e unidades.....	15
Figura 19 - Contador .....	16
Figura 20 - Ecrã de vitória.....	16
Figura 21 - Controlador "controller" e conversor "Hex2ssd".....	17
Figura 22 - Display de 7 segmentos .....	17
<i>Figura 23 - Alternância entre ecrã de jogo, ecrã de vitória ou apagado .....</i>	18
Figura 24 - vários ecrãs .....	19
Figura 25 - Seleção do ânodo para escrever o segmento respetivo .....	19
Figura 26 - Descrição dos switches e pinos utilizados .....	20
Figura 27 - Atribuição de pinos na FPGA .....	20
Figura 28 - Recursos utilizados .....	21
Figura 29 - Esquema Elétrico do modulo master .....	21
Figura 30 - Esquema Elétrico do modulo vga_sync.....	22
Figura 31 - Esquema Elétrico do modulo hex2ssd .....	22



# Introdução

O último trabalho da unidade curricular de Sistemas Analógicos e Digitais Programáveis deu liberdade aos grupos de optarem por realizar um projeto livre recorrendo à FPGA utilizada ao longo do semestre dada a versatilidade da mesma, que permite realizar um infinito número de projetos, sejam eles didáticos ou comerciais, dependendo apenas da criatividade do utilizador.

Com base no conhecimento adquirido ao longo do semestre, o grupo propôs realizar o jogo “Sudoku” recorrendo a linguagem VHDL para o kit *Basys3*.

Neste jogo utilizar-se-á um monitor de computador através da entrada VGA disponibilizada no kit; contadores, máquinas de estados e registos por forma a implementar a configuração do sudoku; displays de 7 segmentos por forma a tornar a experiência da jogabilidade mais didática, fornecendo informações complementares relativamente à posição do cursor.

O maior desafio deste projeto é trabalhar com a interface gráfica VGA uma vez que todos os outros elementos utilizados para o seu desenvolvimento foram lecionados e utilizados em projetos anteriores.

Este documento regista toda a evolução do projeto, evidenciando alguns módulos elaborados para o funcionamento do jogo.

# 1. Jogo Sudoku

O Sudoku é um jogo japonês que se baseia no posicionamento lógico dos números, sendo as regras bastante simples.

O objetivo é preencher uma gralha de 9x9 com números para que cada fila, coluna e secção 3x3 contenha todos os dígitos entre 1 e 9. Note-se que entre cada coluna, fila e secção os números não podem ser repetidos, sendo que no início do jogo alguns quadrados já se encontram preenchidos.

Simplificando as regras:

- Cada fila contém os números de 1 a 9 sendo que não é permitido repeti-los;
- Cada coluna contém os números de 1 a 9 sendo que não é permitido repeti-los;
- Cada secção contém os números de 1 a 9 sendo que não é permitido repeti-los;

8	2	4	9	5	3	6	7	1
6	3	5	8	1	7	9	2	4
7	1	9	6	2	4	8	5	3
5	8	7	2	9	1	3	4	6
1	4	2	7	3	6	5	8	9
3	9	6	4	8	5	2	1	7
2	6	1	5	4	9	7	3	8
4	7	8	3	6	2	1	9	5
9	5	3	1	7	8	4	6	2

Figura 1 - Exemplo de um Sudoku resolvido

A figura 1 apresenta um Sudoku resolvido onde a cor vermelha representa as colunas, a azul as linhas e a amarela a secção 3x3.

## 2. Interface Gráfica

Neste capítulo iremos explicar a solução adota para que seja possível a realização do jogo no display.

### 2.1. Varrimento

Para a realização do trabalho foi utilizado um display com recurso a uma entrada VGA, assim é preciso compreender como se pode apresentar uma imagem no Display.

Em primeiro lugar, com apoio a documentação, observou-se a resolução a adotar, que é de 640 x 480 Píxel.

O varrimento dos Pixéis é feito linha a linha da esquerda para a direita. Considera-se portanto, o Píxel do canto superior esquerdo como o (0,0) e o Píxel do canto inferior direito como o (640, 480), tendo em conta a resolução adotada.

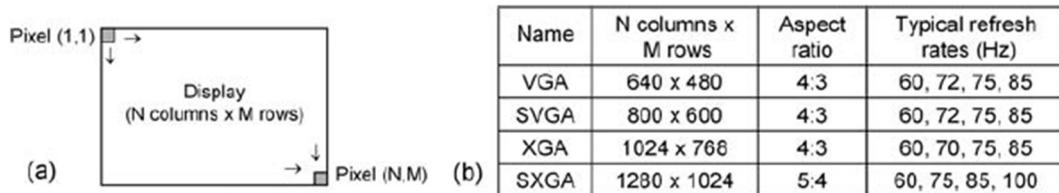


Figura 2 - Varrimento dos Pixéis

```
--!
--! This process generates the main scan counters vscan and hscan
Scan_Counter:process(clk_vga_int, rst)
begin
    if(rst = '1') then
        hscan <= (others => '0');
        vscan <= (others => '0');
    elsif (clk_vga_int'event and clk_vga_int = '1') then
        if (hscan = HSCANEND) then
            hscan <= (others => '0');
            if (vscan= VSCANEND) then
                vscan<=(others => '0');
            else
                vscan<=vscan+'1';
                end if;
            else
                hscan <= hscan + '1';
                end if;
            end if;
        end process;
```

Figura 3 - Processo responsável pelo varrimento

O módulo que trata deste varrimento é o *vga\_sync*, é de salientar que, como indicado, este varrimento tem de ser feito como uma determinada frequência de 25MHz. O processo que faz a divisão de clock está presente na figura 4.

```

--! Divides the master clock - clk (100 MHz) by 1/4 to
--! generate the vga_clk (25 Mhz)
CLKDIVIDER:process(clk, rst)
begin
  if(rst = '1') then
    clk_vga_int <= '0';
    clk_divider <= '0';
  elsif (clk = '1' and clk'event) then
    clk_divider <= not clk_divider;
    if(clk_divider = '0') then
      clk_vga_int <= not clk_vga_int;
    end if;
  end if;
end process;

```

Figura 4 - Clock de 25MHz

## 2.2. Desenho do sudoku

### 2.2.1 Moldura do sudoku

Em primeiro lugar começou por se desenhar o *Background* do tabuleiro que se definiu com a cor azul, desenhado também as linhas delimitadoras da grelha a branco.

Este realiza-se quando a coordenada obtida do módulo *vga\_sync* se encontra entre os limites escolhidos pelo grupo para ser desenhada a *frame*.

```

--Background not in the game
if(current_y <= top_border - border_thickness or current_y >= bottom_border + border_thickness or
   current_x <= left_border - border_thickness or current_x >= right_border + border_thickness) then
  colorOut <= BLACK;

-- Borders
elsif(
  (current_y < top_border and current_y > top_border - border_thickness) or
  (current_y > bottom_border and current_y < bottom_border + border_thickness) or
  (current_x < left_border and current_x > left_border - border_thickness) or
  (current_x > right_border and current_x < right_border + border_thickness)
) then
  colorOut <= WHITE;

--GAME BACKGROUND WITH PLAYER
else
  colorOut <= BLUE;
end if;

```

Figura 5 - Background do tabuleiro e Bordas

De seguida desenhou-se as restantes linhas do tabuleiro. Observando o primeiro *if* da figura 6, tem um comprimento definido por x (de 0x77 a 0x213) e uma largura definida por y (0xAB a 0xB5), correspondente ao intervalo das coordenadas em hexadecimal, sendo então uma linha horizontal. Seguindo o mesmo raciocínio, a alteração da diferença entre as coordenadas, e a sua localização, permite-nos aumentar ou diminuir a espessura da linha e definir a sua posição no display.

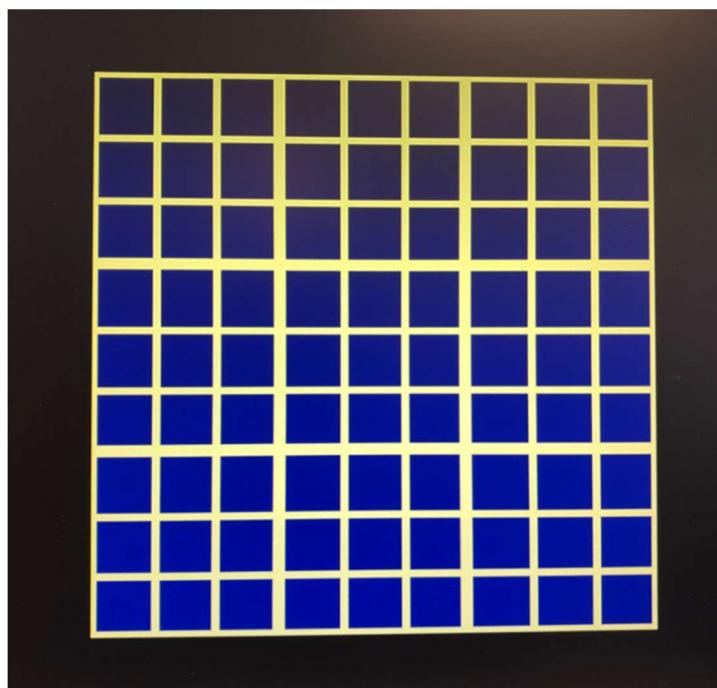
Para as linhas verticais, o comprimento da linha é obtido pela coordenada y, e a espessura e posição pela coordenada x.

A figura 6, mostra algumas das linhas produzidas.

```
--Linha grossas horizontais
if(current_y > x"AB" and current_y <= x"B5" and current_x >= x"77" and current_x <= x"213") then -
    colorOut <= WHITE;
end if;
if(current_y > x"12B" and current_y <= x"135" and current_x >= x"77" and current_x <= x"213") then
    colorOut <= WHITE;
end if;
--Linha grossas verticais
if(current_y > x"30" and current_y <= x"1AF" and current_x >= x"184" and current_x <= x"18E") then
    colorOut <= WHITE;
end if;
if(current_y > x"30" and current_y <= x"1AF" and current_x >= x"FB" and current_x <= x"105") then
    colorOut <= WHITE;
end if;
```

*Figura 6 – Desenho das linhas*

Desenhadas as linhas e o background, conseguiu-se obter o tabuleiro apresentado na figura 7. De referir que no eixo horizontal o tabuleiro vai desde os pixéis 0x77 a 0x213, quanto que no eixo vertical vai de 0x30 a 0x1AF. O módulo responsável por desenhar o tabuleiro é o FrameImage.



*Figura 7 - Tabuleiro*

## 2.2.2 Desenho de cada Dígito

Para completar o sudoku é necessário preencher a *frame* do sudoku com os números. O grupo optou, portanto, desenhar os números com uma dimensão de 18x24 pixéis, sendo estes construídos através de uma matriz em que os valores a '1' formam o número pretendido. Um exemplo é o número 1 mostrado na figura abaixo.

```
CONSTANT Gone: digit18x24 :=(
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0')
  ('0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'));
```

Figura 8 - Desenho dos números

## 2.2.3 Registos, atualização de valores e verificação do sudoku

O Sudoku é composto por 81 dígitos, alguns dos quais já estão definidos no começo do jogo. Quando o jogador insere um dígito, esse tem de ser guardado para que se possa confirmar se o tabuleiro se encontra corretamente preenchido e para que o jogador tenha liberdade de o alterar. Assim foi desenvolvido um processo onde a cada dígito estará associado um registo.

```

process(lvl,
        digit_11,digit_12, digit_13, digit_14, digit_15, digit_16, digit_17, digit_18, digit_19,
        digit_21,digit_22, digit_23, digit_24, digit_25,digit_26, digit_27, digit_28, digit_29,
        digit_31,digit_32, digit_33, digit_34, digit_35,digit_36, digit_37, digit_38, digit_39,
        digit_41,digit_42, digit_43, digit_44, digit_45, digit_46, digit_47, digit_48, digit_49,
        digit_51,digit_52, digit_53, digit_54, digit_55, digit_56, digit_57, digit_58, digit_59,
        digit_61,digit_62, digit_63, digit_64, digit_65, digit_66, digit_67, digit_68, digit_69,
        digit_71,digit_72, digit_73, digit_74, digit_75, digit_76, digit_77, digit_78, digit_79,
        digit_81,digit_82, digit_83, digit_84, digit_85, digit_86, digit_87, digit_88, digit_89,
        digit_91,digit_92, digit_93, digit_94, digit_95, digit_96, digit_97, digit_98, digit_99
)
begin
  if(lvl='0')then
    if(
      digit_11="0100" and digit_12="0001" and digit_13 ="0101" and digit_14 ="0111" and digit_15="0011" and digit_16="1000" and digit_17 ="0010" and digit_18 ="0110" and digit_19="1001" and
      digit_21="0111" and digit_22="0100" and digit_23="0011" and digit_24="0010" and digit_25="0100" and digit_26="0001" and digit_27="0101" and digit_28="0000" and digit_29="0001" and
      digit_31="0001" and digit_32="0010" and digit_33="0000" and digit_34="0100" and digit_35="0101" and digit_36="0001" and digit_37="0100" and digit_38="0111" and digit_39="0011" and
      digit_41="0000" and digit_42="0001" and digit_43="0100" and digit_44="0111" and digit_45="0101" and digit_46="0000" and digit_47="0100" and digit_48="0001" and digit_49="0011" and
      digit_51="0100" and digit_52="0101" and digit_53="0100" and digit_54="0011" and digit_55="0001" and digit_56="0000" and digit_57="0011" and digit_58="0010" and digit_59="0100" and
      digit_61="0000" and digit_62="0001" and digit_63="0100" and digit_64="0101" and digit_65="0001" and digit_66="0110" and digit_67="0011" and digit_68="0101" and digit_69="0111" and
      digit_71="0000" and digit_72="0011" and digit_73="0000" and digit_74="0101" and digit_75="0010" and digit_76="0111" and digit_77="0001" and digit_78="0100" and digit_79="0110" and
      digit_81="0100" and digit_82="0111" and digit_83="0000" and digit_84="0000" and digit_85="0000" and digit_86="0100" and digit_87="0001" and digit_88="0011" and digit_89="0100" and
      digit_91="0101" and digit_92="0100" and digit_93="0001" and digit_94="0001" and digit_95="0110" and digit_96="0011" and digit_97="0111" and digit_98="0010" and digit_99="1000"
    )then
      win<='1';
    else
      win<='0';
    end if;
  else
    if(
      digit_11="0011" and digit_12="0111" and digit_13 ="0100" and digit_14 ="0100" and digit_15="1000" and digit_16="0010" and digit_17 ="0101" and digit_18 ="0110" and digit_19="0001" and
      digit_21="0000" and digit_22="0001" and digit_23="0010" and digit_24="0111" and digit_25="0100" and digit_26="0101" and digit_27="0100" and digit_28="0000" and digit_29="0001" and
      digit_31="0000" and digit_32="0010" and digit_33="0001" and digit_34="0100" and digit_35="0100" and digit_36="0001" and digit_37="0100" and digit_38="0111" and digit_39="0011" and
      digit_41="0000" and digit_42="0001" and digit_43="0100" and digit_44="0111" and digit_45="0101" and digit_46="0000" and digit_47="0100" and digit_48="0001" and digit_49="0011" and
      digit_51="0100" and digit_52="0101" and digit_53="0100" and digit_54="0011" and digit_55="0001" and digit_56="0000" and digit_57="0011" and digit_58="0010" and digit_59="0111" and
      digit_61="0101" and digit_62="0011" and digit_63="0000" and digit_64="0100" and digit_65="0001" and digit_66="0111" and digit_67="0001" and digit_68="0101" and digit_69="0111" and
      digit_71="0100" and digit_72="0101" and digit_73="0100" and digit_74="0000" and digit_75="0100" and digit_76="0111" and digit_77="0001" and digit_78="0100" and digit_79="0101" and
      digit_81="0100" and digit_82="0111" and digit_83="0000" and digit_84="0000" and digit_85="0000" and digit_86="0100" and digit_87="0001" and digit_88="0011" and digit_89="0100" and
      digit_91="0111" and digit_92="0101" and digit_93="0011" and digit_94="0010" and digit_95="0100" and digit_96="0000" and digit_97="0100" and digit_98="0100" and digit_99="0110"
    )then
      win<='1';
    else
      win<='0';
    end if;
  end if;
end process;

```

Figura 9 - Registos para guardar os números

Tendo um registo para cada número, verifica-se registo a registo se os números inseridos pelo jogador correspondem aos da solução respetiva ao nível que está a ser jogado. Caso todos os registos se encontrem corretos irá aparecer uma imagem que o jogador venceu o jogo, tópico esse que será mais aprofundado nos capítulos que se seguem.

Na figura 10, observa-se a solução prevista para dois níveis diferentes do Sudoku.

```

process(lvl,
        digit_11,digit_12, digit_13, digit_14, digit_15, digit_16, digit_17, digit_18, digit_19,
        digit_21,digit_22, digit_23, digit_24, digit_25,digit_26, digit_27, digit_28, digit_29,
        digit_31,digit_32, digit_33, digit_34, digit_35,digit_36, digit_37, digit_38, digit_39,
        digit_41,digit_42, digit_43, digit_44, digit_45, digit_46, digit_47, digit_48, digit_49,
        digit_51,digit_52, digit_53, digit_54, digit_55, digit_56, digit_57, digit_58, digit_59,
        digit_61,digit_62, digit_63, digit_64, digit_65, digit_66, digit_67, digit_68, digit_69,
        digit_71,digit_72, digit_73, digit_74, digit_75, digit_76, digit_77, digit_78, digit_79,
        digit_81,digit_82, digit_83, digit_84, digit_85, digit_86, digit_87, digit_88, digit_89,
        digit_91,digit_92, digit_93, digit_94, digit_95, digit_96, digit_97, digit_98, digit_99
)
begin
  if(lvl='0')then
    if(
      digit_11="0100" and digit_12="0001" and digit_13 ="0101" and digit_14 ="0111" and digit_15="0011" and digit_16="1000" and digit_17 ="0010" and digit_18 ="0110" and digit_19="1001" and
      digit_21="0111" and digit_22="0100" and digit_23="0011" and digit_24="0010" and digit_25="0100" and digit_26="0001" and digit_27="0101" and digit_28="0000" and digit_29="0001" and
      digit_31="0001" and digit_32="0010" and digit_33="0000" and digit_34="0100" and digit_35="0101" and digit_36="0001" and digit_37="0100" and digit_38="0111" and digit_39="0011" and
      digit_41="0000" and digit_42="0001" and digit_43="0100" and digit_44="0111" and digit_45="0101" and digit_46="0000" and digit_47="0100" and digit_48="0001" and digit_49="0011" and
      digit_51="0100" and digit_52="0101" and digit_53="0100" and digit_54="0011" and digit_55="0001" and digit_56="0000" and digit_57="0011" and digit_58="0010" and digit_59="0100" and
      digit_61="0000" and digit_62="0001" and digit_63="0100" and digit_64="0101" and digit_65="0001" and digit_66="0110" and digit_67="0001" and digit_68="0101" and digit_69="0111" and
      digit_71="0100" and digit_72="0011" and digit_73="0000" and digit_74="0101" and digit_75="0010" and digit_76="0111" and digit_77="0001" and digit_78="0100" and digit_79="0110" and
      digit_81="0100" and digit_82="0111" and digit_83="0000" and digit_84="0000" and digit_85="0000" and digit_86="0100" and digit_87="0001" and digit_88="0011" and digit_89="0100" and
      digit_91="0101" and digit_92="0100" and digit_93="0001" and digit_94="0001" and digit_95="0110" and digit_96="0011" and digit_97="0111" and digit_98="0010" and digit_99="1000"
    )then
      win<='1';
    else
      win<='0';
    end if;
  else
    if(
      digit_11="0011" and digit_12="0111" and digit_13 ="0100" and digit_14 ="0100" and digit_15="1000" and digit_16="0010" and digit_17 ="0101" and digit_18 ="0110" and digit_19="0001" and
      digit_21="0000" and digit_22="0001" and digit_23="0010" and digit_24="0111" and digit_25="0100" and digit_26="0101" and digit_27="0100" and digit_28="0000" and digit_29="0001" and
      digit_31="0000" and digit_32="0010" and digit_33="0001" and digit_34="0100" and digit_35="0100" and digit_36="0001" and digit_37="0011" and digit_38="0111" and digit_39="0001" and
      digit_41="0000" and digit_42="0001" and digit_43="0100" and digit_44="0111" and digit_45="0101" and digit_46="0000" and digit_47="0100" and digit_48="0001" and digit_49="0011" and
      digit_51="0100" and digit_52="0101" and digit_53="0100" and digit_54="0011" and digit_55="0001" and digit_56="0000" and digit_57="0011" and digit_58="0010" and digit_59="0100" and
      digit_61="0101" and digit_62="0011" and digit_63="0000" and digit_64="0100" and digit_65="0001" and digit_66="0111" and digit_67="0001" and digit_68="0101" and digit_69="0111" and
      digit_71="0100" and digit_72="0101" and digit_73="0100" and digit_74="0000" and digit_75="0100" and digit_76="0111" and digit_77="0001" and digit_78="0100" and digit_79="0101" and
      digit_81="0100" and digit_82="0111" and digit_83="0000" and digit_84="0000" and digit_85="0000" and digit_86="0100" and digit_87="0001" and digit_88="0011" and digit_89="0100" and
      digit_91="0111" and digit_92="0101" and digit_93="0011" and digit_94="0010" and digit_95="0100" and digit_96="0000" and digit_97="0100" and digit_98="0100" and digit_99="0110"
    )then
      win<='1';
    else
      win<='0';
    end if;
  end if;
end process;

```

Figura 10 - Verificação se o Sudoku foi preenchido corretamente tendo em conta o nível

Caso o utilizar pretenda alterar um valor de um dado registo, a figura 11 apresenta a solução para que tal seja possível.

```
if(update='1')then
    if(row = "0001" and col="0001") then
        digit_11 <= (num);
```

Figura 11 - Atualização de um dígito

Cada número, desde 1 até 9, tem a sua combinação binária associada, assim é necessário atribuir a cada número um dígito a ser desenhado.

```
--Function construction:-----
FUNCTION bcd_to_digit6x8 (SIGNAL input: std_logic_vector(3 downto 0)) RETURN digit18x24 IS
BEGIN
    CASE input is
        WHEN "0000" => return zero;
        WHEN "0001" => return one;
        WHEN "0010" => return two;
        WHEN "0011" => return three;
        WHEN "0100" => return four;
        WHEN "0101" => return five;
        WHEN "0110" => return six;
        WHEN "0111" => return seven;
        WHEN "1000" => return eight;
        WHEN "1001" => return nine;
        when others => return zero;
    END CASE;
END bcd_to_digit6x8;
```

Figura 12 - Consoante o valor do dígito devolve o dígito a ser desenhado

## 2.2.4 Níveis

Como referido anteriormente foram desenvolvidos dois níveis, com dificuldades distintas. Na figura 13 observa-se os dígitos presentes no início do jogo para o nível zero.

```
if(lvl='0')then
    digit_13<="0101"; digit_16<="1000"; digit_18<="0110"; digit_19<="1001";
    digit_22<="0110"; digit_24<="0010"; digit_25<="0100"; digit_29<="0001";
    digit_36<="0001"; digit_38<="0111"; digit_39<="0011"; digit_51<="0001";
    digit_52<="0101"; digit_54<="0011"; digit_55<="1000"; digit_56<="0010";
    digit_62<="1000"; digit_63<="0100"; digit_64<="1001"; digit_66<="0110";
    digit_68<="0101"; digit_73<="0001"; digit_75<="0010"; digit_77<="1001";
    digit_79<="0110"; digit_82<="0111"; digit_84<="1000"; digit_86<="0100";
    digit_88<="0011"; digit_97<="0111";
```

Figura 13 - Atribuição do valor dos dígitos consoante o nível

Consoante o nível atribuído é desenhado no mapa os números já pré-preenchidos. Caso o *colorOut*(11 *downto* 7) o dígito será vermelho, que são dígitos inseridos pelo utilizador.

No caso de *colorOut(11 dowto 4)* o dígito será amarelo, que são dígitos que se encontram no tabuleiro ao início.

```

elseIF (lvl='0' and current_x >= (x"D4"+SPACELEFT) AND current_x < (x"D4"+SPACELEFT+DIGITLENGTH) AND current_y >= (x"35"+SPACEUP) AND current_y < (x"35"+SPACEUP+DIGITHIGH) THEN
    colorOut(11 dowto 4) <=(OTHERS => bcd_to_digit6x8(digit_13)(to_Integer(UNSIGNED(current_y-(x"35"+SPACEUP))),to_Integer(UNSIGNED(current_x-(x"D4"+SPACELEFT))))) ;
elseIF (lvl='1' and current_x >= (x"D4"+SPACELEFT) AND current_x < (x"D4"+SPACELEFT+DIGITLENGTH) AND current_y >= (x"35"+SPACEUP) AND current_y < (x"35"+SPACEUP+DIGITHIGH) THEN
    colorOut(11 dowto 8) <=(OTHERS => bcd_to_digit6x8(digit_13)(to_Integer(UNSIGNED(current_y-(x"35"+SPACEUP))),to_Integer(UNSIGNED(current_x-(x"D4"+SPACELEFT))))) ;

```

Figura 14 - Desenho de um dígito dependente do mapa

As figuras 15 e 16 apresentam os dois níveis do Sudoku propostos pelo grupo, na fase inicial e depois de preenchidos.

	5		8	6	9			
6		2 4			1			
			1	7	3			
1 5		3 8 2						
8 4	9	6		5				
	1	2	9	6				
7	8	4		3				
			7					
4 1 5	7 3 8	2 6 9						
7 6 3	2 4 9	5 8 1						
9 2 8	6 5 1	4 7 3						
3 9 6	4 7 5	8 1 2						
1 5 7	3 8 2	6 9 4						
2 8 4	9 1 6	3 5 7						
8 3 1	5 2 7	9 4 6						
6 7 2	8 9 4	1 3 5						
5 4 9	1 6 3	7 2 8						

Figura 15 - Nível 0

	9 8 2							
8		6	4	9				
9 6	4	1						
2	5	4	6	1				
6 4	8	3	2					
5 3 1	6 2		9	4				
9 6		5		2				
2		6	7	4 5				
5	2 4		9 6					
3 7 4	9 8 2	5 6 1						
1 8 5	7 3 6	4 2 9						
9 6 2	4 5 1	3 7 8						
8 2 7	5 9 4	6 1						
6 4 9	8 1 3	2 5 7						
5 3 1	6 2 7	9 8 4						
4 9 6	1 7 5	8 3 2						
2 1 8	3 6 9	7 4 5						
7 5 3	2 4 8	1 9 6						

Figura 16 - Nível 1

## 2.2.5 Contador

O processo *DigitalClock* apresenta o tempo que o jogador demorou a resolver o jogo. Este tempo é apresentado em segundos e serve como pontuação, quanto mais depressa o jogador concluir o nível melhor a sua performance.

Para elaborar um *clock* para que seja possível contar de segundo a segundo, devidamente o clock de 100MHz para 1Hz.

```
process(clk, counter)
begin
    if rising_edge (clk)then
        counter <= counter + 1;
        if counter = all_ones then
            ctick <= '1';
            counter <=(others=>'0');
        else
            ctick <= '0';
        end if;
    end if;
end process;
```

Figura 17 - Clock de 1Hz

Tendo um cronómetro que conte de um em um segundo, é preciso incrementar os algarismos das unidades, dezenas, centenas e milhar.

```
process(clk,ctick, unidade, dezenas,centenas,milhar, finish)
begin
    if (rising_edge(clk) and finish='0')then
        if(rst='1')then
            unidade<=(others=>'0');
            dezenas<=(others=>'0');
            centenas<=(others=>'0');
            milhar<=(others=>'0');
        else
            if(ctick='1')then
                unidade<= unidade + 1;
                if unidade = "1001" then
                    dezenas<= dezenas +1;
                    unidade<=(others=>'0');
                    if dezenas = "1001" then
                        centenas<= centenas +1;
                        dezenas<=(others=>'0');
                        if centenas = "1001" then
                            milhar<= milhar +1;
                            centenas<=(others=>'0');
                        end if;
                    end if;
                end if;
            end if;
        end if;
    end if;
end process;
```

Figura 18 - Incremento do algarismo dos milhares, dezenas, centenas e unidades

Assim, foi possível realizar o contador presente na figura 19



Figura 19 - Contador

Este módulo permite ainda mostrar ao utilizador uma mensagem “Win” em caso de vitória, assim como o tempo compreendido.

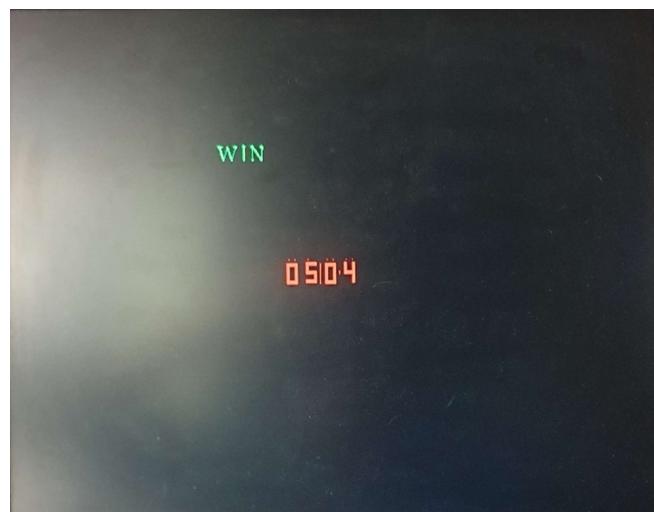


Figura 20 - Ecrã de vitória

### 3. Display de 7 segmentos

Neste capítulo é explicada a solução adotada para que, com o auxílio do display de 7 segmentos, se implemente uma interface que permite ao utilizador saber a posição (quadricula) em que se situa o cursor.

Para tal foi utilizado os módulos desenvolvidos no trabalho laboratorial 3 de SEADP onde realizamos um controlador “*controller*”, que define o que irá aparecer em todos os 4 displays colocando um código na respetiva variáveis/display e o conversor “*Hex2ssd*”do código passado pelo controlador para o display de 7 segmentos.

```
process(row,col,rst)
begin
  if(rst='1')then
    Hex4_plus <= (others=>'1');
    Hex3_plus <= (others=>'1');
    Hex2_plus <= (others=>'1');
    Hex1_plus <= (others=>'1');
  else
    Hex4_plus(4) <= '0';
    Hex4_plus(3 downto 0) <= row;
    Hex3_plus <= "10001";
    Hex2_plus(4) <= '0';
    Hex2_plus(3 downto 0) <= col;
    Hex1_plus <= "10000";
  end if;
end process;
begin
  with Hex_plus SElect
    seg<="1000000" when "00000", --0
    "1111001" when "00001", --1
    "0100100" when "00010", --2
    "0110000" when "00011", --3
    "0011001" when "00100", --4
    "0010010" when "00101", --5
    "0000010" when "00110", --6
    "1111000" when "00111", --7
    "0000000" when "01000", --8
    "0010000" when "01001", --9
    "1000110" when "10000", --C
    "0101111" when "10001", --R
    "1111111" when others;
  end Behavioral;
```

Figura 21 - Controlador “*controller*” e conversor “*Hex2ssd*”

Em suma os dois módulos dão origem ao aspeto da figura 22 onde observamos o display com a seguinte sequência: “número da linha”, “identificador linha (row)”, “número da colina”, “identificador coluna (col)”.

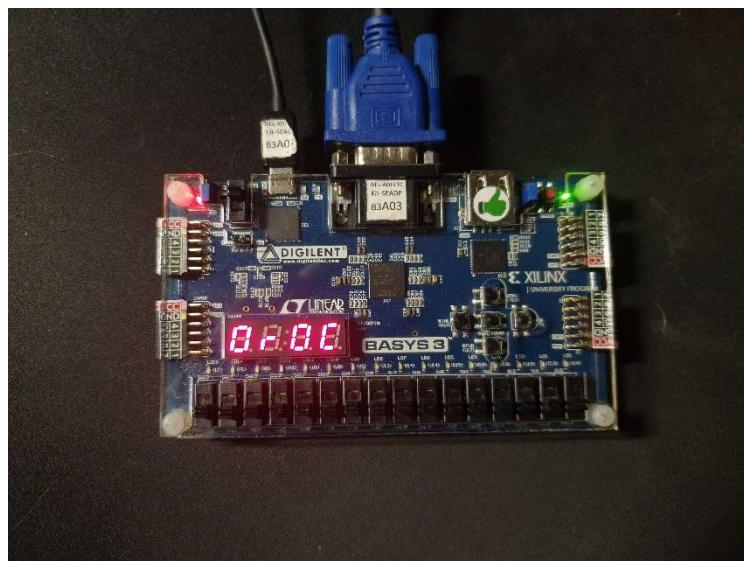


Figura 22 - Display de 7 segmentos

## 4 Módulo de Ligação

O módulo Master é o módulo que interliga todos os módulos referidos anteriormente, sendo ainda responsável por realizar dois processos extras.

Nesses dois processos um é a máquina de estados que muda respetivamente para o estado “init” se o jogo for reiniciado onde o ecrã se encontra apagado, para o estado “play” onde o utilizador irá resolver o sudoku e para o estado “won” que é o estado de vitória.

```
PROCESS (state_reg,colorOutFrame,colorOutNumber, onDisplay, finish, colorOutClock, reset)      PROCESS (CLK , reset)
BEGIN                                         BEGIN
    win<='0';
    state_next<= state_reg;
    vgaRed <= (others>'0');
    vgaGreen <= (others>'0');
    vgaBlue <= (others>'0');
CASE state_reg IS
    when init =>
        if(onDisplay='1')then
            state_next<=play;
        end if;
    when play =>
        vgaRed <= colorOutFrame(11 downto 0);
        vgaGreen <= colorOutFrame(7 downto 4);
        vgaBlue <= colorOutFrame(3 downto 0);
        if(colorOutNumber /= "00000000000") then
            vgaRed <= colorOutNumber(11 downto 8);
            vgaGreen <= colorOutNumber(7 downto 4);
            vgaBlue <= colorOutNumber(3 downto 0);
        end if;
        if(colorOutClock /= "000000000000") then
            vgaRed <= colorOutClock(11 downto 8);
            vgaGreen <= colorOutClock(7 downto 4);
            vgaBlue <= colorOutClock(3 downto 0);
        end if;
        if(finish='1')then
            state_next<=won;
        end if;
        if(reset='1' or onDisplay='0')then
            state_next<=init;
        end if;
    when won =>
        vgaRed <= colorOutClock(11 downto 0);
        vgaGreen <= colorOutClock(7 downto 4);
        vgaBlue <= colorOutClock(3 downto 0);
        win<='1';
        if(reset='1')then
            state_next<=init;
        end if;
    when others =>
end case;
end process;

PROCESS (CLK , reset)
BEGIN
    if rising_edge (CLK)then
        IF reset = '1' THEN
            state_reg <= init;
        else
            state_reg <= state_next;
        end if;
    end if;
END PROCESS;
```

Figura 23 - Alternância entre ecrã de jogo, ecrã de vitória ou apagado

Na figura 24 verifica-se o funcionamento da máquina de estados presente na figura 23, onde observa-se o ecrã de play e o ecrã de won.

No ecrã de vitória (*won*) está presente a mensagem de vitória (*win*) bem como o tempo demorado para na resolução do *Sudoku*.



Figura 24 - vários ecrãs

Por último, para selecionar-mos em qual o display de 7 segmentos queremos escrever, temos de selecionar o respetivo ânodo.

```

anodetick: process(clk, reset)
begin
    if(reset='1')then
        refresh_counter<= (others >='0');
    elsif(rising_edge(CLK))then
        refresh_counter <=refresh_counter +1;
    end if;
end process;
Display_Active<= refresh_counter(19 downto 18);--for reallife
--Display_Active<= refresh_counter(8 downto 7);--for tb
anodeSel: process(Display_Active,ssd1,ssd2,ssd3,ssd4)
begin
    --dp<='1';
    case Display_Active is
    when "00" =>
        an<="0111";
        --dp<='0';
        seg<=ssd4;
    when "01" =>
        an<="1011";
        seg<=ssd3;
    when "10" =>
        an<="1101";
        seg<=ssd2;
    when others =>
        an<="1110";
        seg<=ssd1;
    end case;
end process;

```

Figura 25 - Seleção do ânodo para escrever o segmento respetivo

## 5 Atribuição de Pinos à FPGA/Controlo Sudoku

Por fim, segue-se a descrição dos *switches* utilizados na FPGA:

- *Reset* – reinicia o programa;
- *Update* – atualiza o número selecionado pelo utilizador;
- *Correct* – não desenvolvido mas com o intuito de ajudar o utilizador a resolver o mapa explicado no próximo capítulo;
- *Level* – seleciona o mapa de sudoku a resolver;
- *Number* – o número a colocar. Vai de [0001-1001] ou seja [1-9];
- *Column* – coluna selecionada. Vai de [0001-1001] ou seja [1-9];
- *Row* – linha selecionada. Vai de [0001-1001] ou seja [1-9];

Temos ainda um LED que indica o estado de vitória “WIN” e os 4 LEDS seguintes demonstram os segundos a passar de 0 a 9.

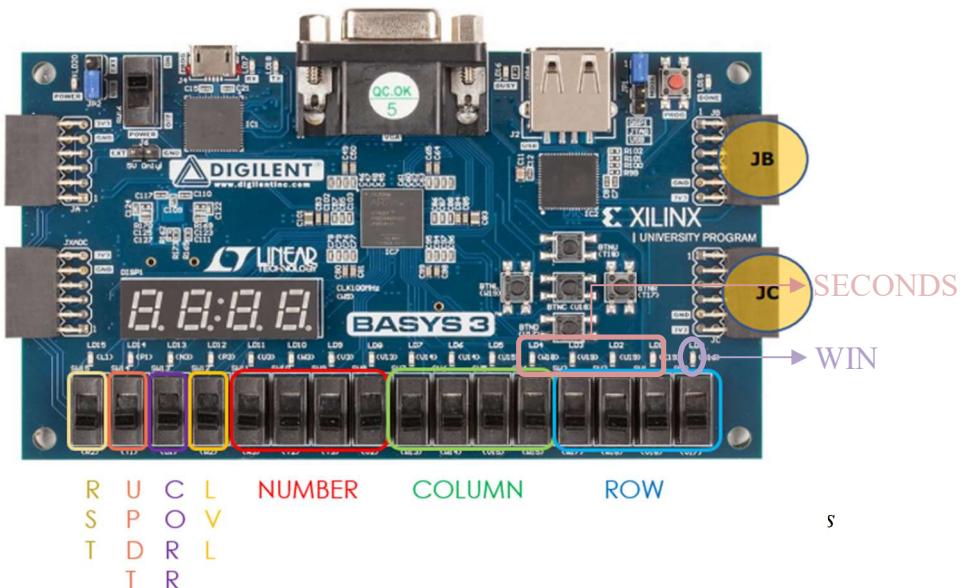


Figura 27 - Atribuição de pinos na FPGA

## 6. Análise do esquema elétrico.

Após encontrada a solução ao projeto do sudoku, compete agora analisar o impacto que este tem na FPGA.

Resource	Utilization	Available	Utilization...
LUT	3930	20800	18.89
FF	357	41600	0.86
IO	46	106	43.40
BUFG	1	32	3.13

Figura 28 - Recursos utilizados

Analisando a utilização de recursos verificámos que para as nossas saídas em função da entrada usamos em logica combinatória 19% de LUTs, que achamos ser um valor razoável dada a complexidade do nosso projeto.

Verificamos também que usamos apenas 1% dos flip-flops disponíveis pelo nosso kit e 43% das portas de entrada e saída deste.

Coloca-se então a questão de onde são usados estes recursos no nosso programa, esperando que os módulos de maior complexidade (os de design gráfico) utilizem mais recursos.

Começando por analisar o módulo (Master) verificamos este se encontra organizado com o uso de alguns multiplexers, registos e portas logicas mas nada de grande consumo.

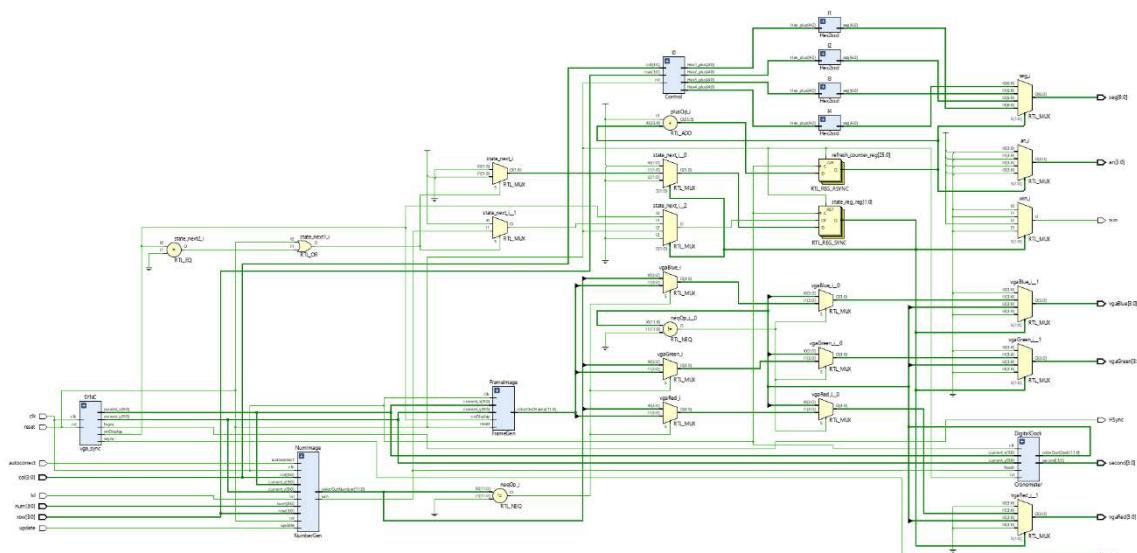


Figura 29 - Esquema Elétrico do módulo master

Partindo em seguida para a analise do modulo de varrimento dos pixeis do ecrã, obsevamos que este se trata de apenas combinações logicas entre as entradas e saidas do processo, havendo apenas um maior consumo de LUTs.

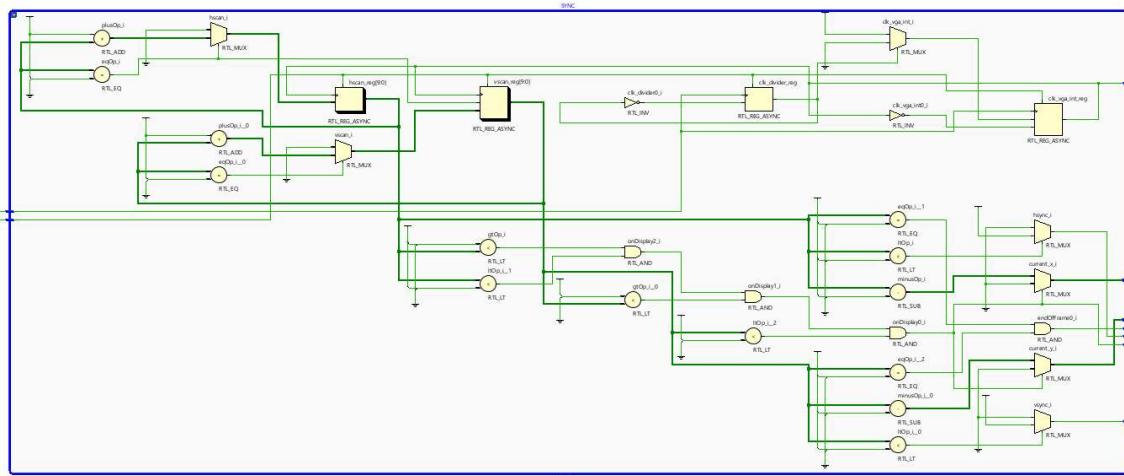


Figura 30 - Esquema Elétrico do módulo vga\_sync

Quanto aos modulos de Controlo e de conversão para o display de 7 segmentos, verificamos que o controlo é uma simples combinação usando 4 multiplexers e que a conversão é algo que foi previamente carregado na ROM da FPGA, sendo estes dois modulos muito simples em relação ao todo do esquema.

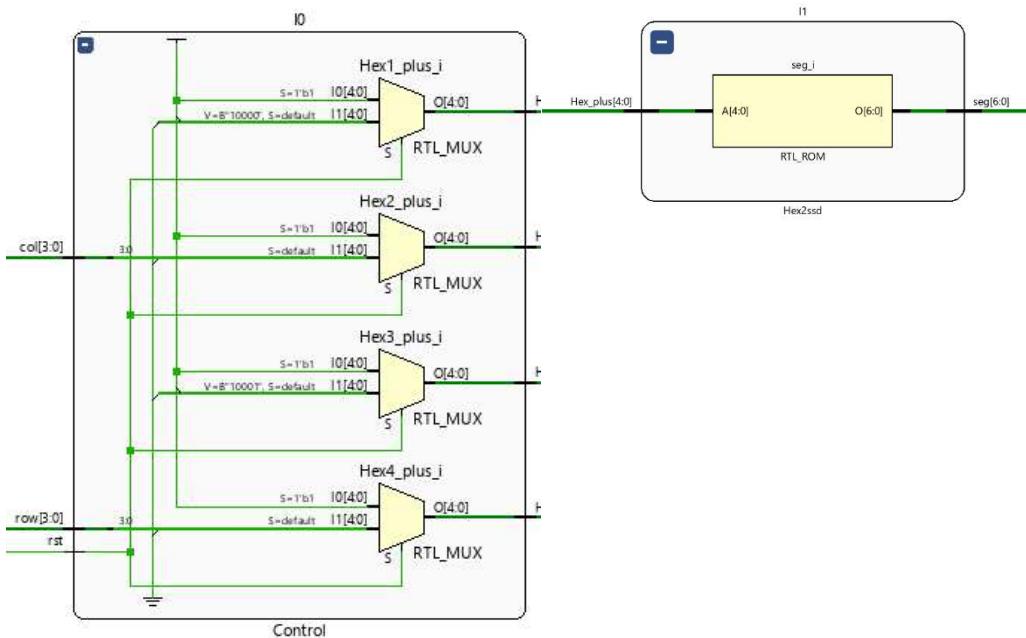


Figura 31 - Esquema Elétrico dos módulos controlo e hex2ssd

Como esperado os 3 modulos mais complexos são o Frame Image, o DigitalClock e o NumImage sendo que a complexidade aumenta respetivamente nesta ordem. De modo a ser possível visualizar os 3 modulos foi colocado no ANEXO A as 3 páginas com os esquemas relativos a estes.

É relevante esclarecer que a complexidade destes módulos deve-se à necessidade de haver grandes contadores, imensos registo e consequentemente inumeras combinações logicas sendo estes três modulos a grande rasão para usarmos tantos flipflops e LUTs.

## 7. Melhorias e desafios a futuros colegas

Este trabalho encontra-se com um bug de sincronismo no display onde alguns dígitos por vezes contem traços em volta. Não conseguimos encontrar a razão pois este bug acontecia de forma aleatória deduzindo que este fosse um problema de sincronismo entre o módulo *VGA\_sync* e o *NumImage*.

Para completar o trabalho desenvolvido, ficaria mais interessante ter uma tela de START onde fosse possível escolher a dificuldade do nível e uma tela de vitória mais completa.

Reconhecemos ainda que em vez de serem usados switches para escolher a "*row*", a "*column*" e dar *update* ao número, poderia ser interessante desenvolver um módulo onde usando um joystick externo à placa *Basys3*, o utilizador pudesse realizar as mesmas funções tendo uma melhor jogabilidade.

Por último deixamos em aberto a possibilidade de se desenvolver um "*autocorrect*", este desafio tem como base o site [sudoku.com](http://sudoku.com) onde existe a opção de o jogador pedir que lhe sejam alertados os erros à medida que este preenche o mapa.

## 8. Conclusão

Com este trabalho expandimos o conhecimento em linguagem VHDL, aprofundando os conhecimentos em registos, maquins de estados, contadores, interface gráfica.

Observa-mos que com os conhecimentos obtidos e com base num pequeno kit de FPGA consegue-se realizar programas de elevada complexidade. Com a utilização deste kit, caso surja atualizações no projeto, seja fácil a sua implementação.

Consideramos que este trabalho trará uma grande importância para o nosso futuro académico, visto que este envolve conteúdos relacionados com a Eletrónica e sendo este uma grande parte do nosso curso de engenharia.

## 9. Bibliografia

Trabalhos e Laboratórios anteriores

- Capítulo 12 do livro “Circuit Desing and Simulation with VHDL”, 2<sup>a</sup> edição, de Volnei A.Pedroni
- “FPGA Prototyping by VHDL Examples – CHAPTER 12 – VGA CONTROLLER I: GRAPHIC”
- <https://github.com/pkashyap95/MazeSolver>
- <https://www.instructables.com/CPE-133/>

# **ANEXO A**

