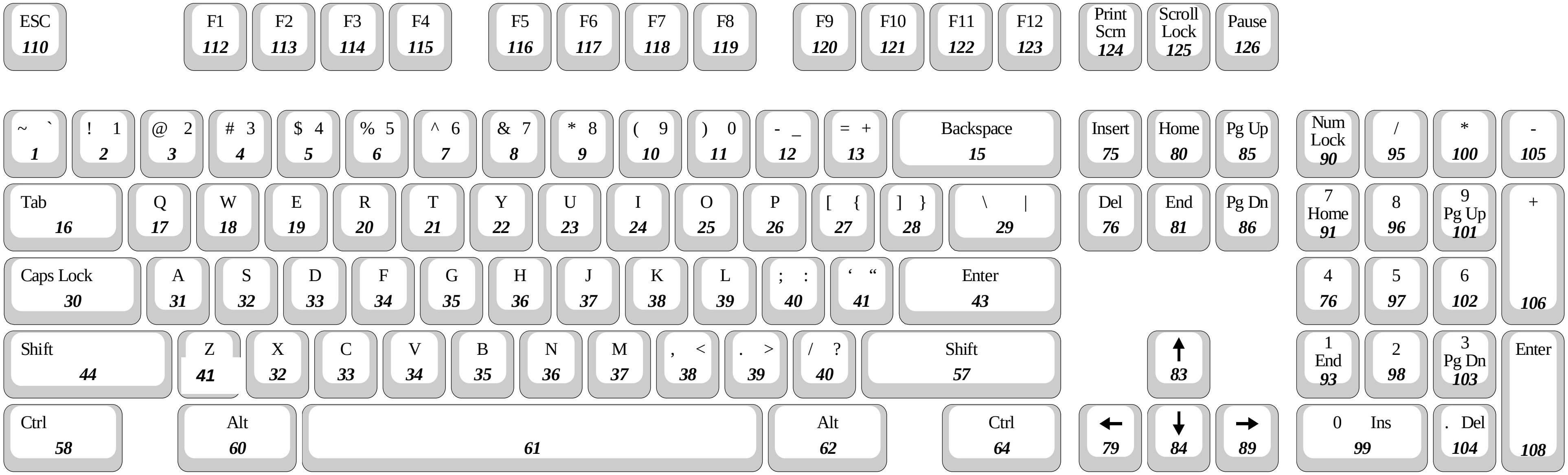
Exercícios com VHDL

GPqCom

Prof. Raimes Moraes

1 - Identificação de teclas



ESC 110 | F1 112 | F2 113 | F3 114 | F4 115 | F5 116 | F6 117 | F7 118 | F8 119 | F9 120 | F10 121 | F11 122 | F12 123 | Print Scrn 124 | Scroll Lock 125 | Pause 126

~ ` 1 | ! 1 2 | @ 2 3 | # 3 4 | $ 4 5 | % 5 6 | ^ 6 7 | & 7 8 | * 8 9 | (9 10 |) 0 11 | - _ 12 | = + 13 | Backspace 15 | Insert 75 | Home 80 | Pg Up 85 | Num Lock 90 | / 95 | * 100 | - 105

Tab 16 | Q 17 | W 18 | E 19 | R 20 | T 21 | Y 22 | U 23 | I 24 | O 25 | P 26 | [{ 27 |] } 28 | \ | 29 | Del 76 | End 81 | Pg Dn 86 | 7 Home 91 | 8 96 | 9 Pg Up 101 | + 106

Caps Lock 30 | A 31 | S 32 | D 33 | F 34 | G 35 | H 36 | J 37 | K 38 | L 39 | ; : 40 | ‘ “ 41 | Enter 43 | 4 76 | 5 97 | 6 102

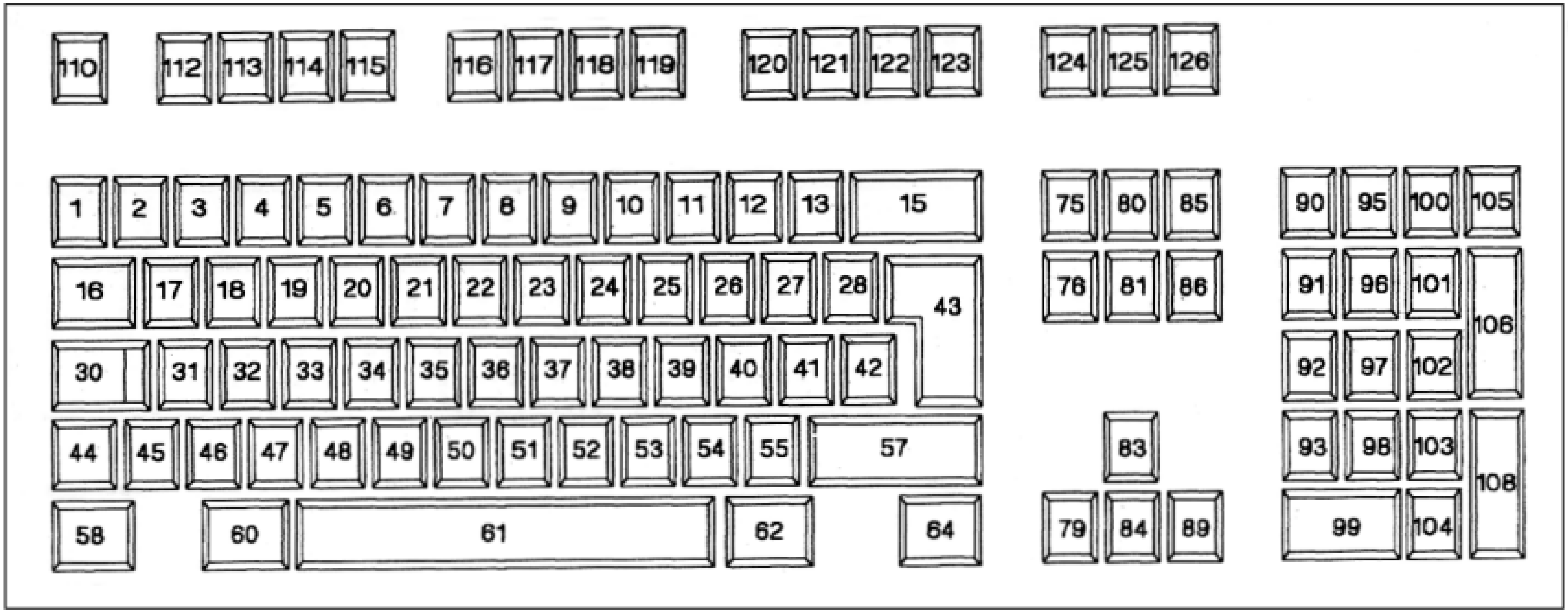
Shift 44 | Z 41 | X 32 | C 33 | V 34 | B 35 | N 36 | M 37 | , < 38 | . > 39 | / ? 40 | Shift 57 | ↑ 83 | 1 End 93 | 2 98 | 3 Pg Dn 103 | Enter 108

Ctrl 58 | Alt 60 | 61 | Alt 62 | Ctrl 64 | ← 79 | ↓ 84 | → 89 | 0 Ins 99 | . Del 104

Teclado do PC

Nro. das teclas para identificação do *Scan Code*

1 - Identificação de teclas



Teclado do PC

Nro. das teclas para identificação do *Scan Code*

Scan Codes

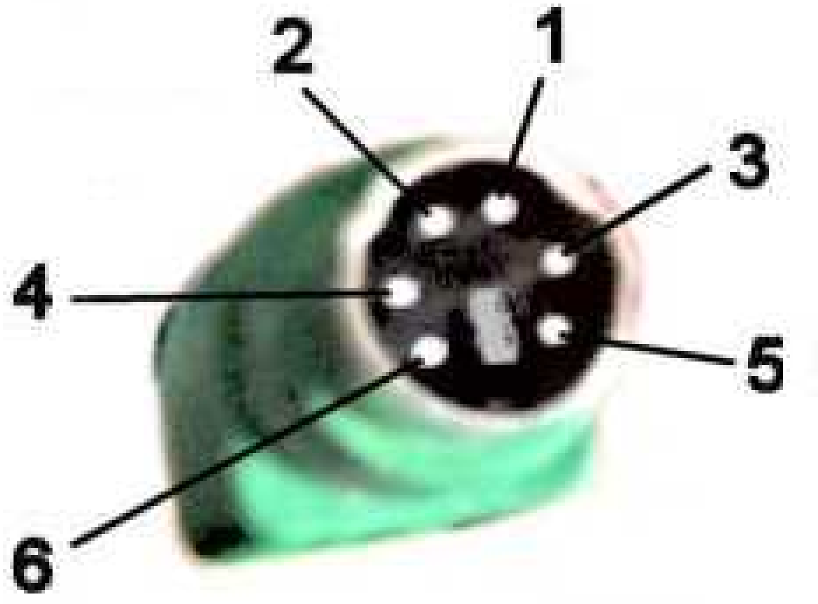
Key#	Make Code	Break Code	Key#	Make Code	Break Code	Key#	Make Code	Break Code
1	0E	F0 0E	31	1C	F0 1C	90	77	F0 77
2	16	F0 16	32	1B	F0 1B	91	6C	F0 6C
3	1E	F0 1E	33	23	F0 23	92	6B	F0 6B
4	26	F0 26	34	2B	F0 2B	93	69	F0 69
5	25	F0 25	35	34	F0 34	96	75	F0 75
6	2E	F0 2E	36	33	F0 33	97	73	F0 73
7	36	F0 36	37	3B	F0 3B	98	72	F0 72
8	3D	F0 3D	38	42	F0 42	99	70	F0 70
9	3E	F0 3E	39	4B	F0 4B	100	7C	F0 7C
10	46	F0 46	40	4C	F0 4C	101	7D	F0 7D
11	45	F0 45	41	52	F0 52	102	74	F0 74
12	4E	F0 4E	43	5A	F0 5A	103	7A	F0 7A
13	55	F0 55	44	12	F0 12	104	71	F0 71
15	66	F0 66	46	1A	F0 1A	105	7B	F0 7B
16	0D	F0 0D	47	22	F0 22	106	79	F0 79
17	15	F0 15	48	21	F0 21	110	76	F0 76
18	1D	F0 1D	49	2A	F0 2A	112	05	F0 05
19	24	F0 24	50	32	F0 32	113	06	F0 06
20	2D	F0 2P	51	31	F0 31	114	04	F0 04
21	2C	F0 2C	52	3A	F0 3A	115	0c	F0 0C
22	35	F0 35	53	41	F0 41	116	03	F0 03
23	3C	F0 3C	54	49	F0 49	117	0B	F0 0B
24	43	F0 43	55	4A	F0 4A	118	83	F0 83
25	44	F0 44	57	59	F0 59	119	0A	F0 0A
26	4D	F0 4D	58	14	F0 14	120	01	F0 01
27	54	F0 54	60	11	F0 11	121	09	F0 09
28	5B	F0 5B	61	29	F0 29	122	78	F0 78
29	5D	F0 5D	62	E0 11	E0 F0 11	123	07	F0 07

The remaining key codes are a function of the shift, control, alt, or num-lock keys.

Conector PC/Teclado

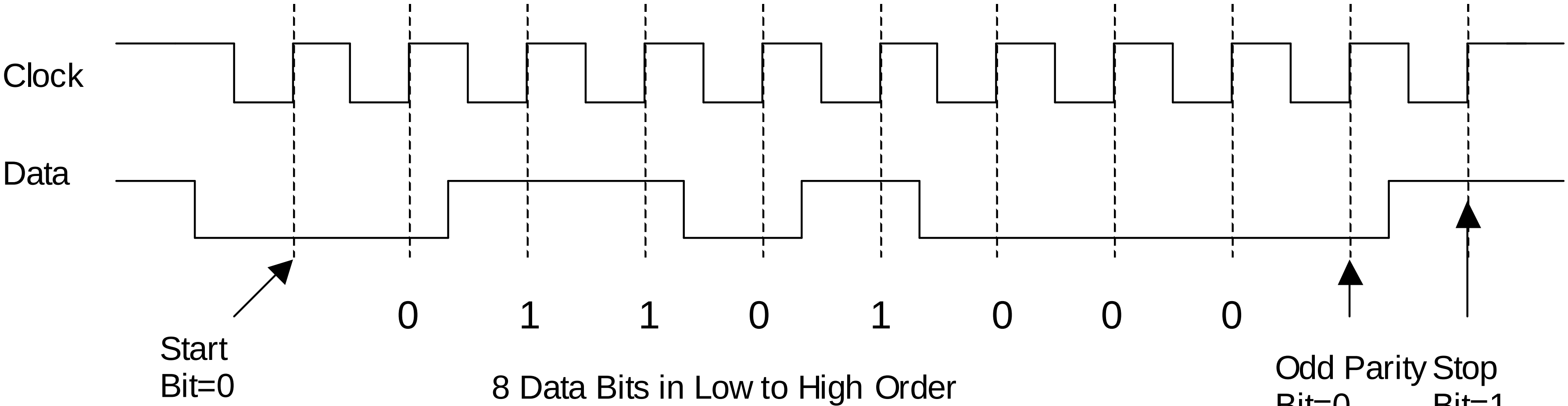
PS/2 keyboard connector (MINI-DIN6)

Connector Pin #	Purpose
Pin 1	KBDAT (data)
Pin 2	not used
Pin 3	GND
Pin 4	VCC (+5V)
Pin 5	KBDCLK (clock)
Pin 6	not used

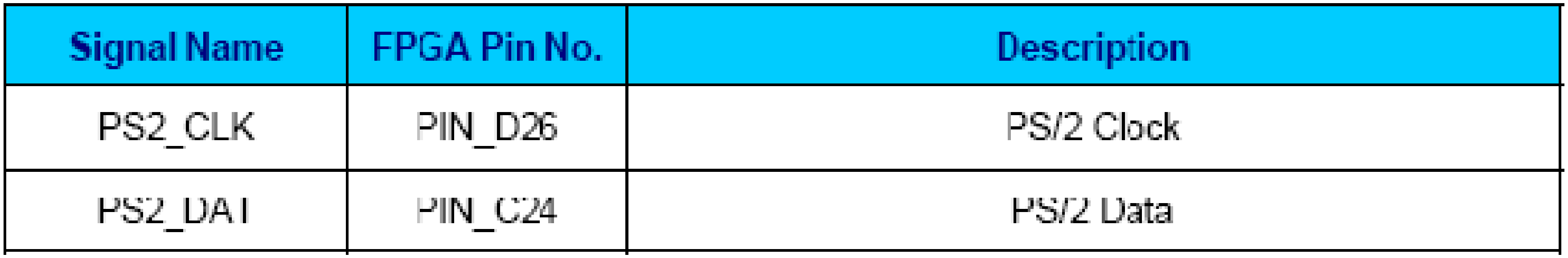


Protocolo de envio do *scan code* pelo teclado

Período do Clock do Teclado: 70 us



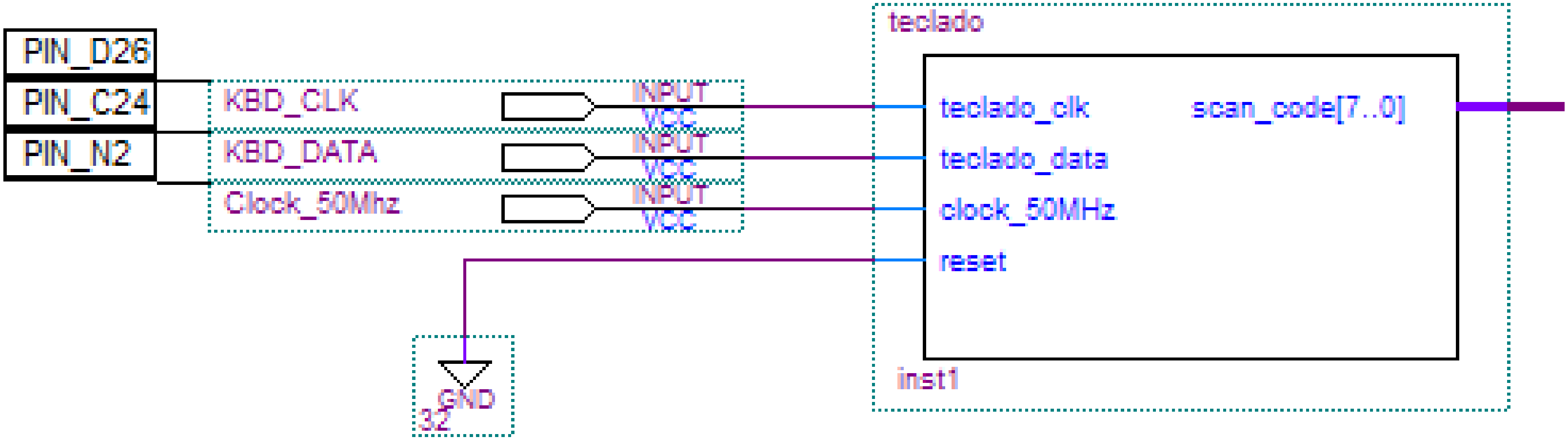
8 Data Bits in Low to High Order
Scan Code shown is 16H for a “1” character
which is keyboard key #2



Signal Name	FPGA Pin No.	Description
PS2_CLK	PIN_D26	PS/2 Clock
PS2_DAT	PIN_C24	PS/2 Data

Objetivo: Criar 2 módulos. Um para receber o *scan code* da tecla pressionada; outro para mostrar os valores correspondentes das teclas de 0 a 9 e A a F no display de sete segmentos. OBS: Possível colocar os 2 processos juntos sob a mesma arquitetura.

Primeiro Módulo
Recebe *scan code*

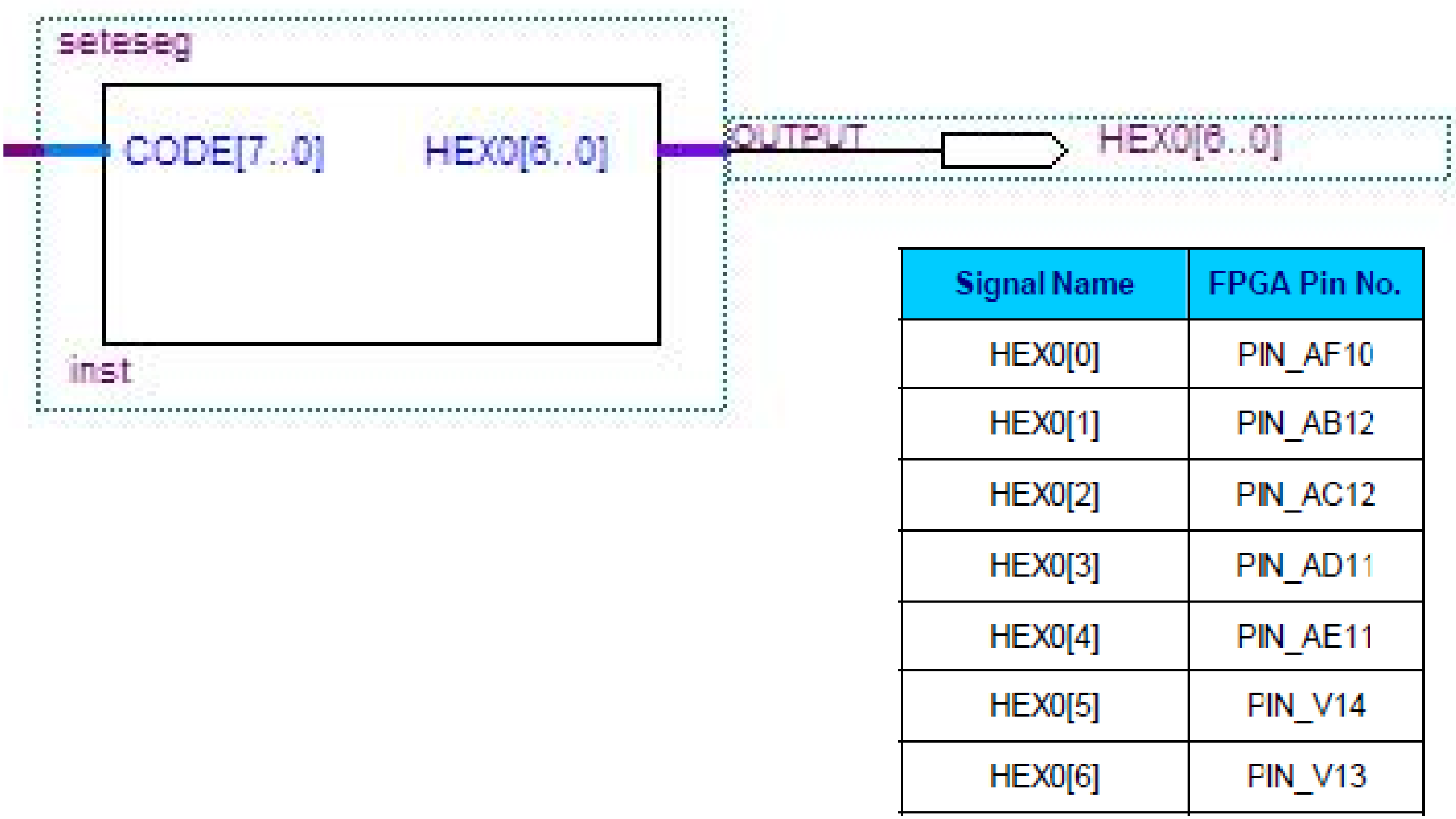


```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;
ENTITY teclado IS
        PORT(   teclado_clk,teclado_data,clock_50MHz,reset  : IN        STD_LOGIC;
                scan_code           : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
          );
END ENTITY teclado;

ARCHITECTURE rtl OF teclado IS
        signal debounce_ok                                  : std_logic;
        signal clock                                        : std_logic;

        BEGIN

-- Processo utilizado para dividir o clock de 50MHz para 25MHz
half_clk: PROCESS
        variable  debounce : std_logic_vector(7 downto 0);
BEGIN
        WAIT UNTIL clock_50MHz'EVENT AND clock_50MHz= '1';
        IF clock = '1' THEN     clock <= '0';
        ELSE  clock <=  '1';
        END IF;
END PROCESS half_clk;
```

```
-- Processo utilizado para eliminar bouncing do teclado

filtra_clk: PROCESS
variable debounce                                    : std_logic_vector(7 downto 0);
BEGIN
        WAIT UNTIL clock'EVENT AND clock= '1';
        debounce(6 DOWNTO 0) := debounce(7 DOWNTO 1) ;
        debounce(7) := teclado_clk;
        IF debounce = x"FF" THEN     debounce_ok <= '1';
        ELSIF  debounce = x"00" THEN debounce_ok <= '0';
        else debounce_ok <= debounce_ok;
        END IF;
END PROCESS filtra_clk;
```

```
-- Processo que recebe dados seriais oriundos do teclado
PROCESS (RESET,debounce_ok)
         variable start_ok                   : std_logic;
         variable count_in                   : integer range 0 to 15;
         variable bit_in                     : std_logic_vector(8 downto 0);
BEGIN
IF RESET='1' THEN           start_ok := '0';    count_in  :=  0;
ELSIF                      (debounce_ok'EVENT AND debounce_ok='1') THEN
-- identifica start bit
 IF teclado_DATA ='0' AND start_ok ='0' THEN  start_ok := '1';
-- recebe byte
 ELSE    -- recebe 8 bits de dados e compõe dado em formato paralelo
         IF start_ok = '1' THEN
                           IF count_in < 9 THEN count_in    := count_in + 1;
                                    bit_in(7 DOWNTO 0) := bit_in(8 DOWNTO 1);
                                    bit_in(8)    := teclado_DATA;

                           -- disponibiliza dado lido; atualiza flags

                           ELSE     count_in     :=  0;  start_ok :='0';
                                    scan_code <= bit_in(7 DOWNTO 0);
                           END IF;
         END IF;
  END IF;
END IF;
END PROCESS;
END ARCHITECTURE;
```

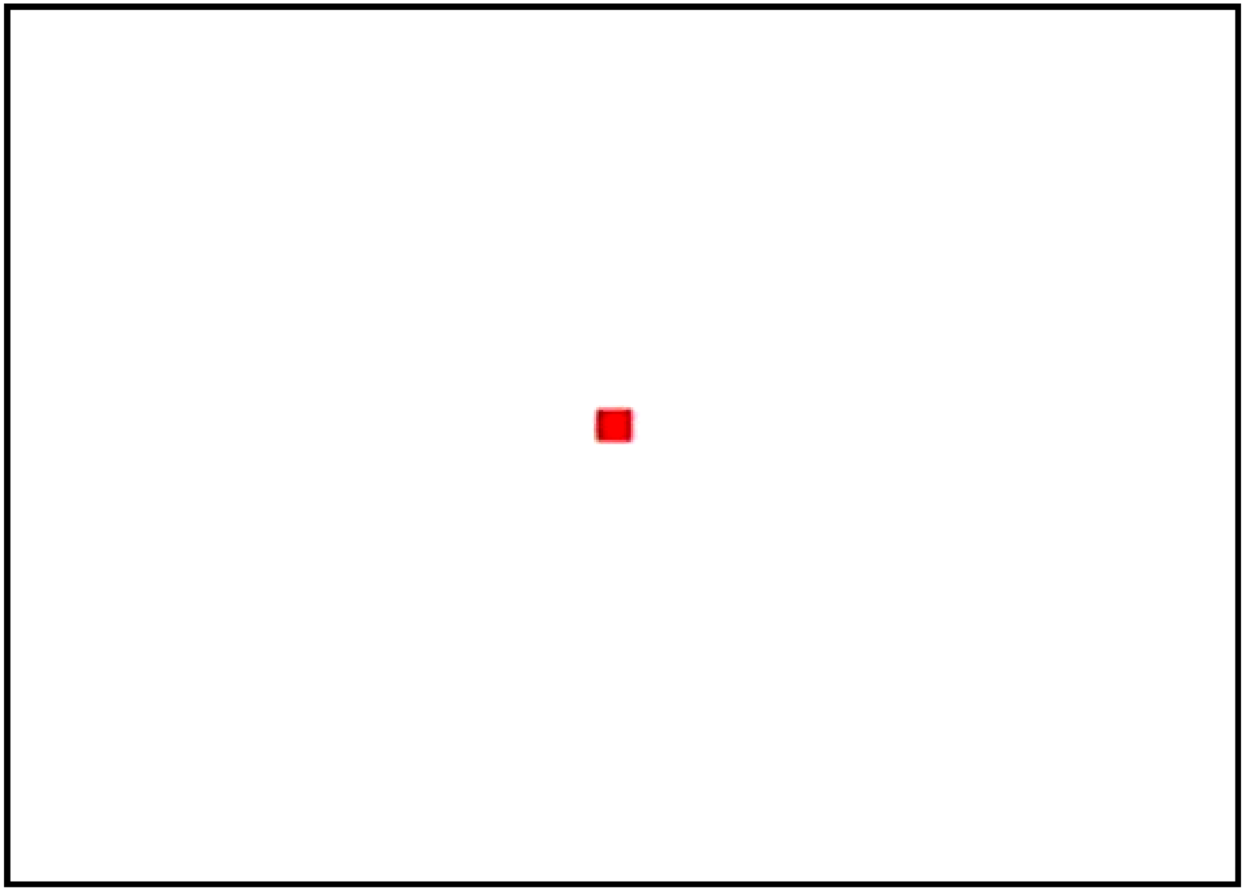
Segundo Módulo
Gera dado para display de 7 segmentos

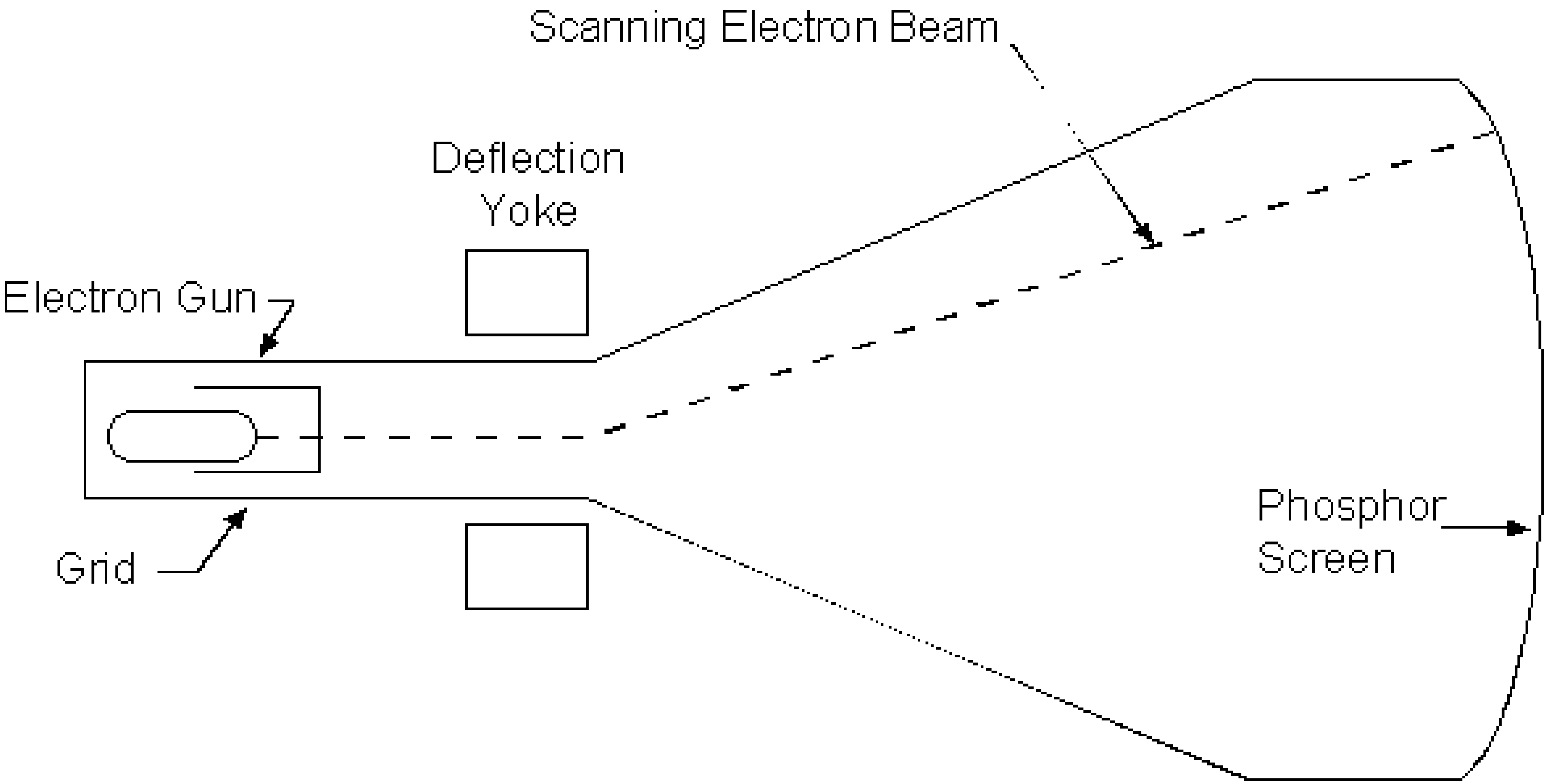


Signal Name	FPGA Pin No.
HEX0[0]	PIN_AF10
HEX0[1]	PIN_AB12
HEX0[2]	PIN_AC12
HEX0[3]	PIN_AD11
HEX0[4]	PIN_AE11
HEX0[5]	PIN_V14
HEX0[6]	PIN_V13

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY seteseg IS
PORT   (CODE            : IN          STD_LOGIC_VECTOR(7 DOWNTO 0);
        HEX0            : OUT         STD_LOGIC_VECTOR(6 DOWNTO 0));
END ENTITY seteseg;
 ARCHITECTURE mostra OF seteseg IS
 BEGIN
   PROCESS (CODE)
   BEGIN
     CASE CODE IS                                  --    GFEDCBA
            WHEN X"45"            => HEX0    <= "1000000";
            WHEN X"16"            => HEX0    <= "1111001";
            WHEN X"1E"            => HEX0    <= "0100100";
            WHEN X"26"            => HEX0    <= "0110000";
            WHEN X"25"            => HEX0    <= "0011001";
            WHEN X"2E"            => HEX0    <= "0010010";
            WHEN X"36"            => HEX0    <= "0000010";
            WHEN X"3D"            => HEX0    <= "1111000";
            WHEN X"3E"            => HEX0    <= "0000000";
            WHEN X"46"            => HEX0    <= "0010000";
            WHEN X"1C"            => HEX0    <= "0001000";
            WHEN X"32"            => HEX0    <= "0000011";
            WHEN X"21"            => HEX0    <= "1000110";
            WHEN X"23"            => HEX0    <= "0100001";
            WHEN X"24"            => HEX0    <= "0000110";
            WHEN X"2B"            => HEX0    <= "0001110";
            WHEN OTHERS           => HEX0    <= "0111111";
     END CASE;
  END PROCESS;
 END ARCHITECTURE;
```

2 – Gerar cursor em monitor RGB





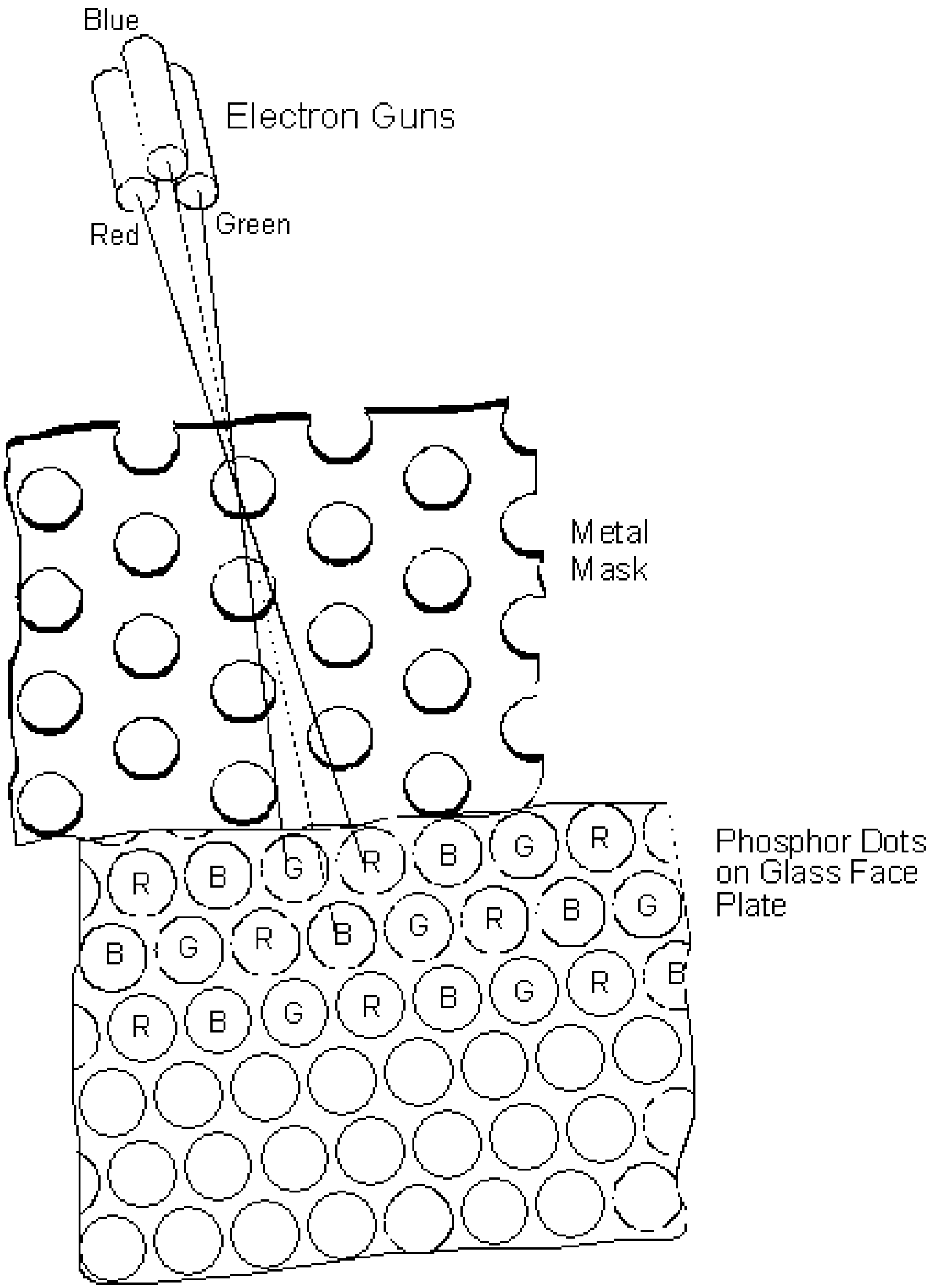
Feixe de elétrons defletido por campo magnético gerado por bobinas

Informação de cor (RGB) é utilizada para controlar intensidade do feixe de elétrons

O TRC contém 3 diferentes fósforos para geração de cor.

O padrão VGA contém 640 por 480 pixels.

Toda a tela deve sofrer *refresh* à taxa de 60Hz para evitar *flicker*.



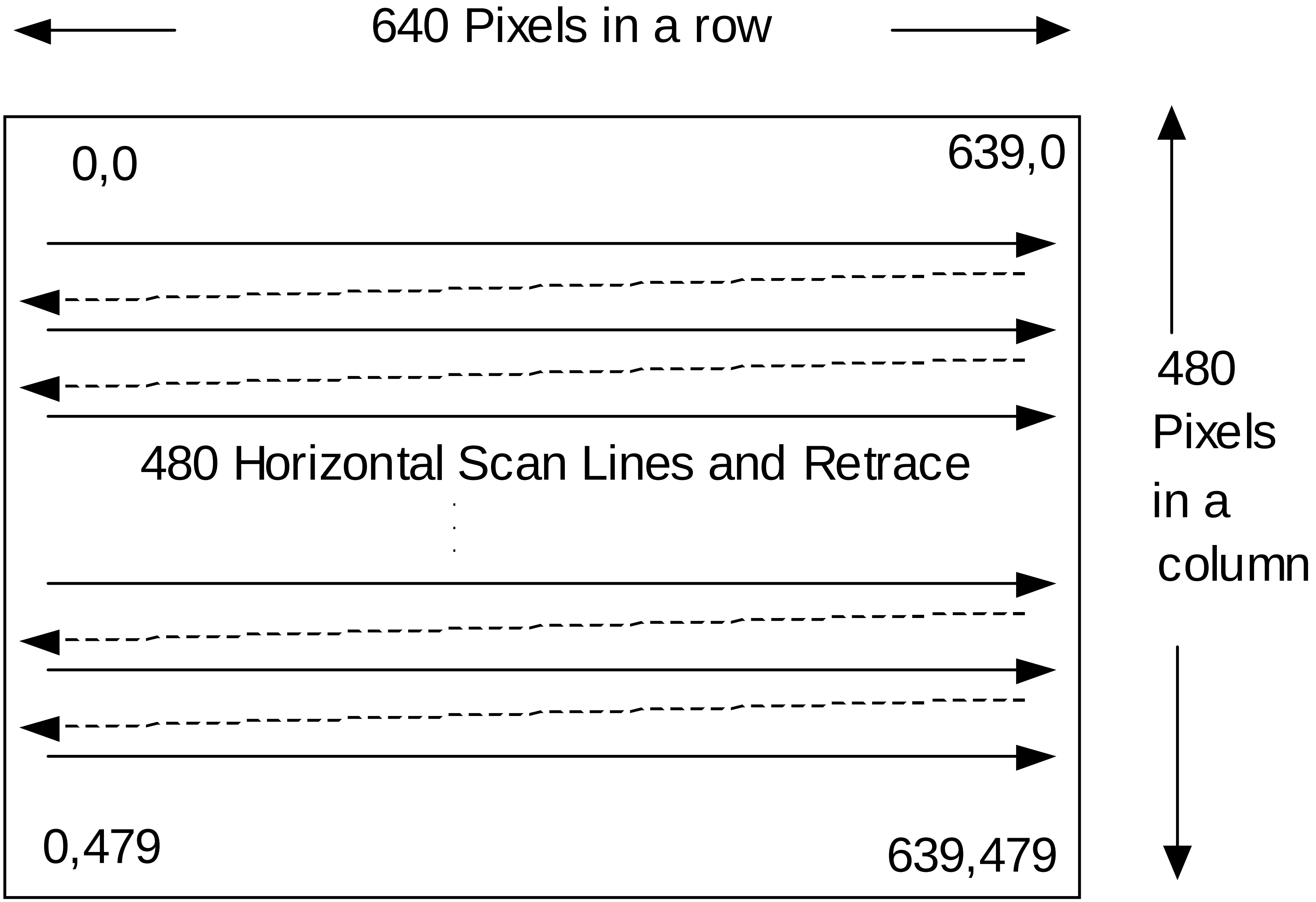
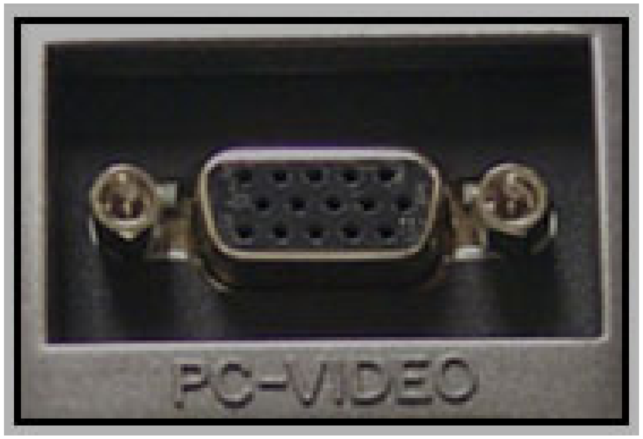
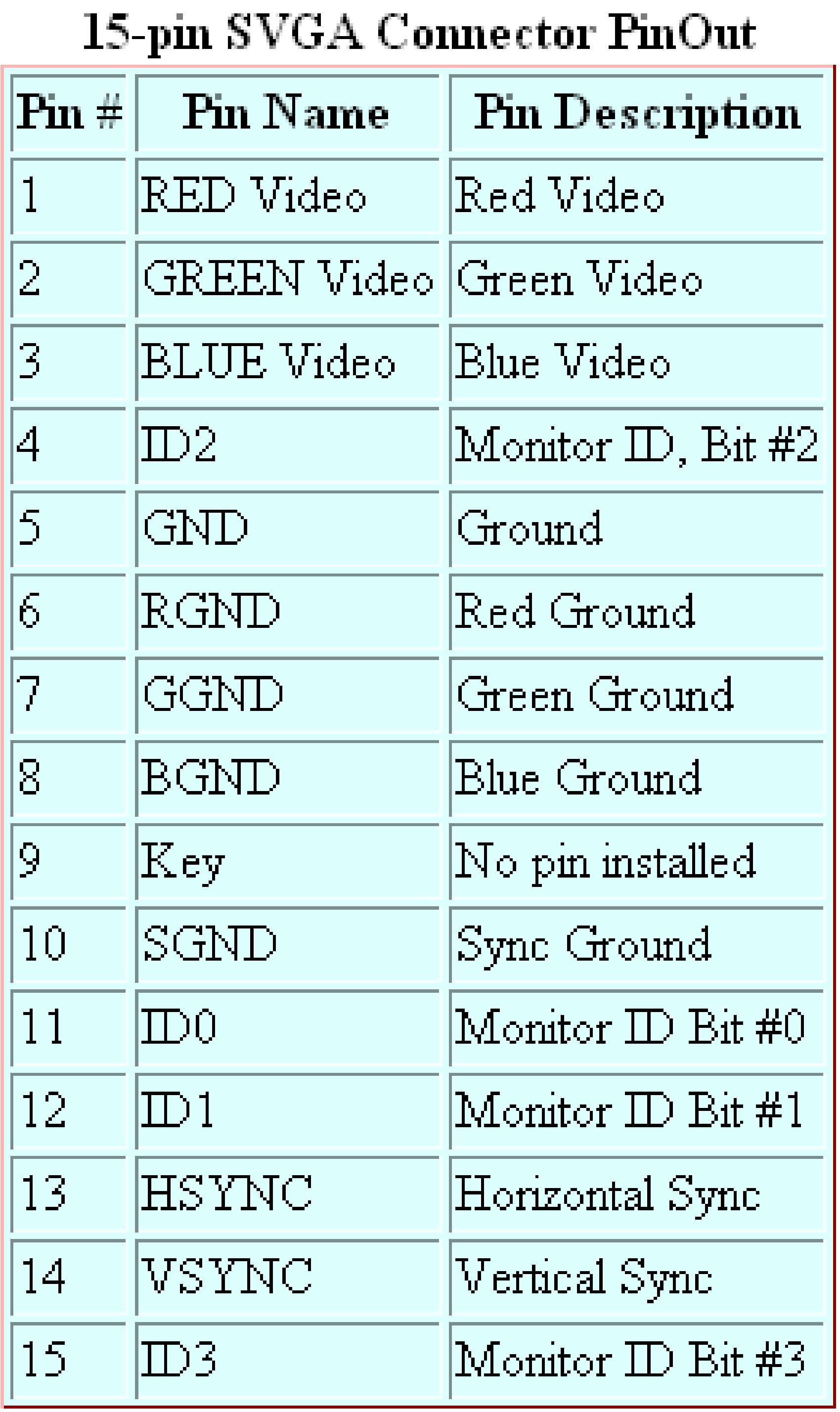


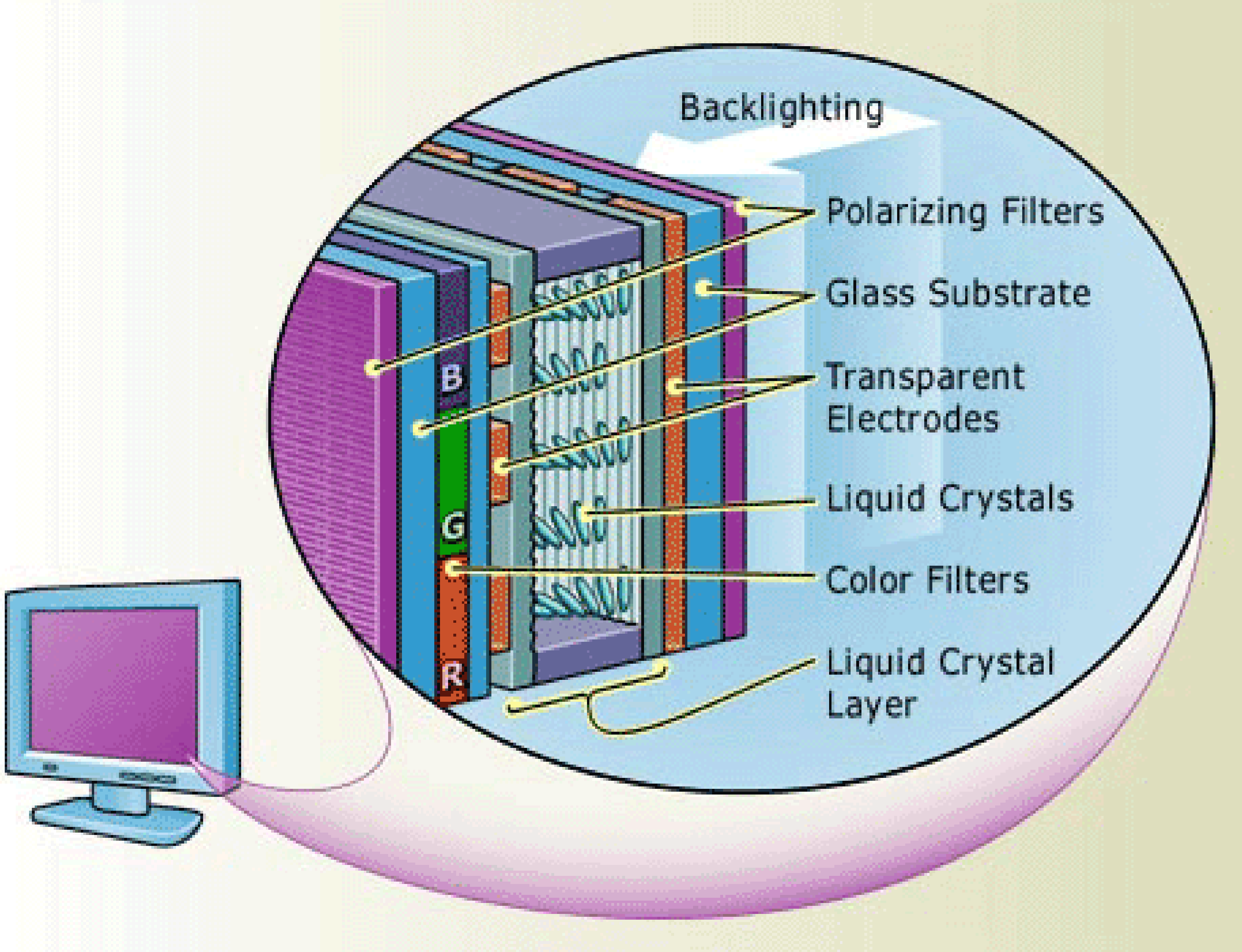
Imagem VGA - 640 pixels por linha. 480 pixels por coluna.
Feixe de eletron deve ser modulado para compor a imagem enquanto se desloca da esquerda para a direita e de cima para baixo. Nas outras direções, o canhão é desligado.





15-pin SVGA Connector PinOut

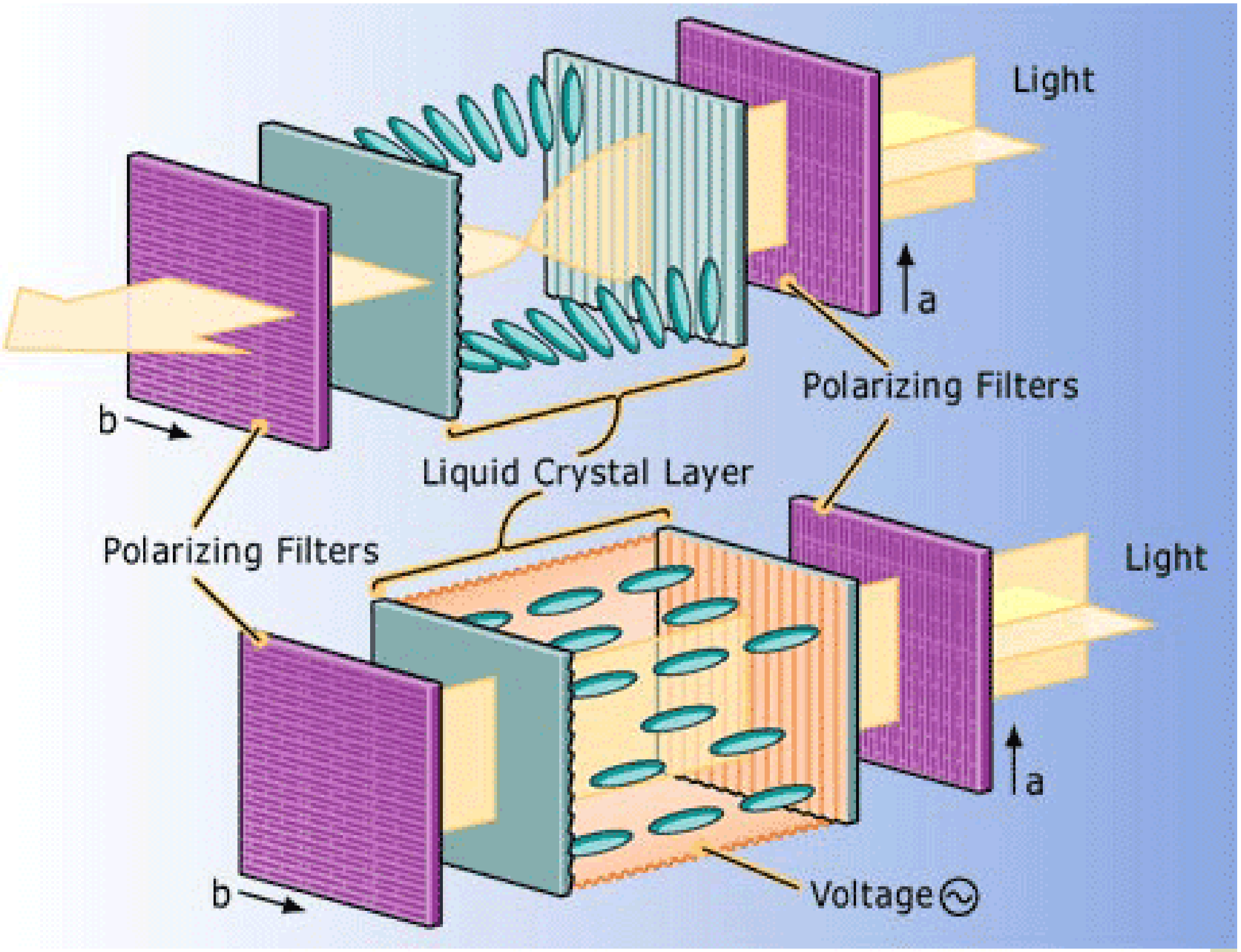
Pin #	Pin Name	Pin Description
1	RED Video	Red Video
2	GREEN Video	Green Video
3	BLUE Video	Blue Video
4	ID2	Monitor ID, Bit #2
5	GND	Ground
6	RGND	Red Ground
7	GGND	Green Ground
8	BGND	Blue Ground
9	Key	No pin installed
10	SGND	Sync Ground
11	ID0	Monitor ID Bit #0
12	ID1	Monitor ID Bit #1
13	HSYNC	Horizontal Sync
14	VSYNC	Vertical Sync
15	ID3	Monitor ID Bit #3



In an active-matrix color LCD, the backlight shines through a sandwich of filters, glass, and liquid crystals. First, a layer of polarizing filters align the light rays. The light then passes through cells filled with liquid crystal that, when twisted by an electrical field, bend the light. Finally, varying amounts of light drift through colored filters; these colors combine to produce the specific hue of each pixel.

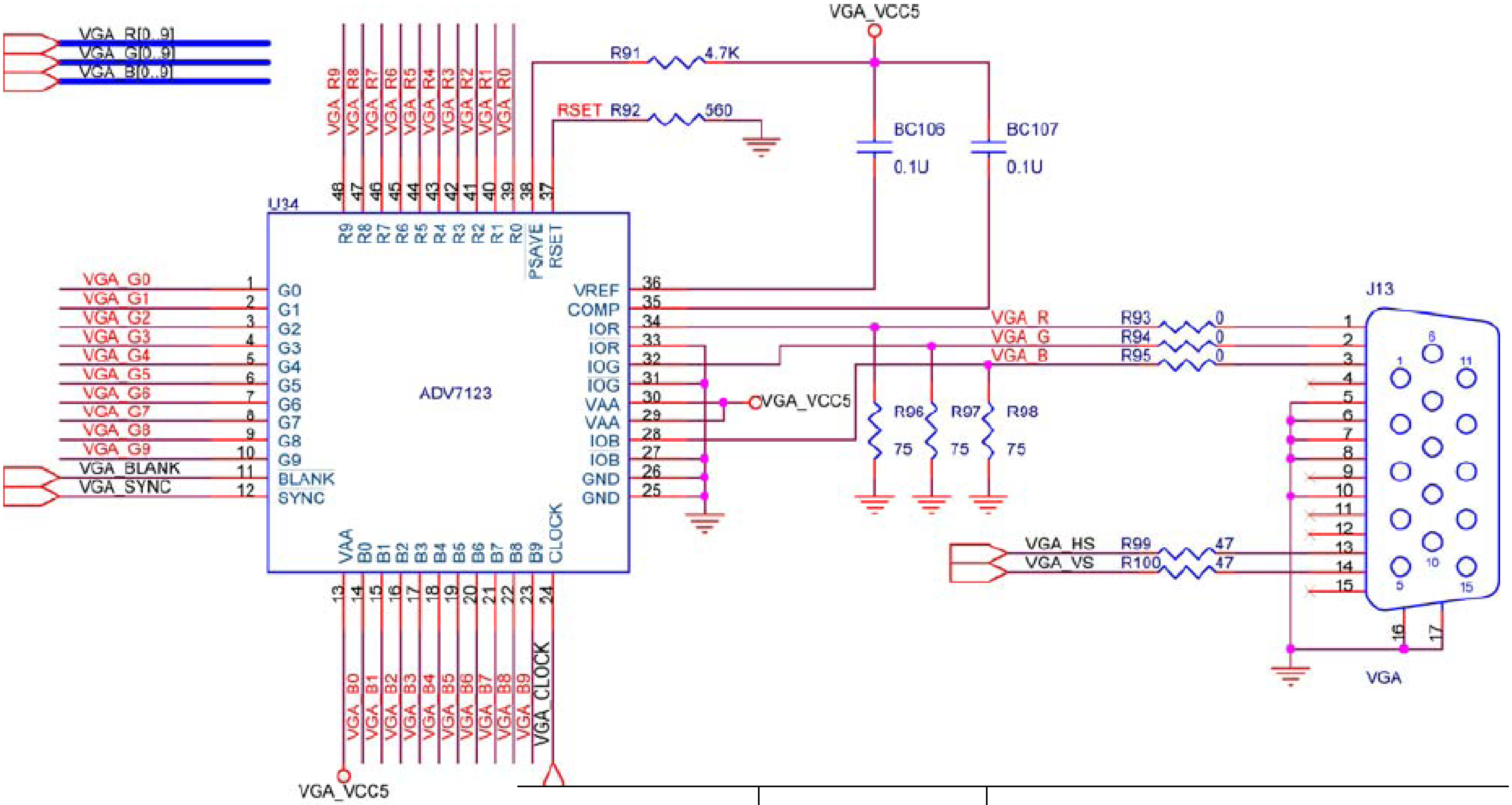
Illustration by Steve McGuire

No LCD colorido, cada pixel é composto de três células de cristal líquido. Cada uma destas três células contém um filtro (vermelho, verde ou azul). A luz passa através do filtro produzindo as cores na tela.



A pair of polarizing filter layers work with the liquid crystals to control emitted light. As light passes through the first filter (a), only vertically aligned light waves remain. If the liquid crystals are in their natural state, they are twisted--which causes the light wave to turn horizontally. If an electric field is applied, the liquid crystals straighten and the cell doesn't bend the light. Since the second filter (b) only lets horizontal light waves through, light that passes through the straight liquid crystals is blocked by the second filter.

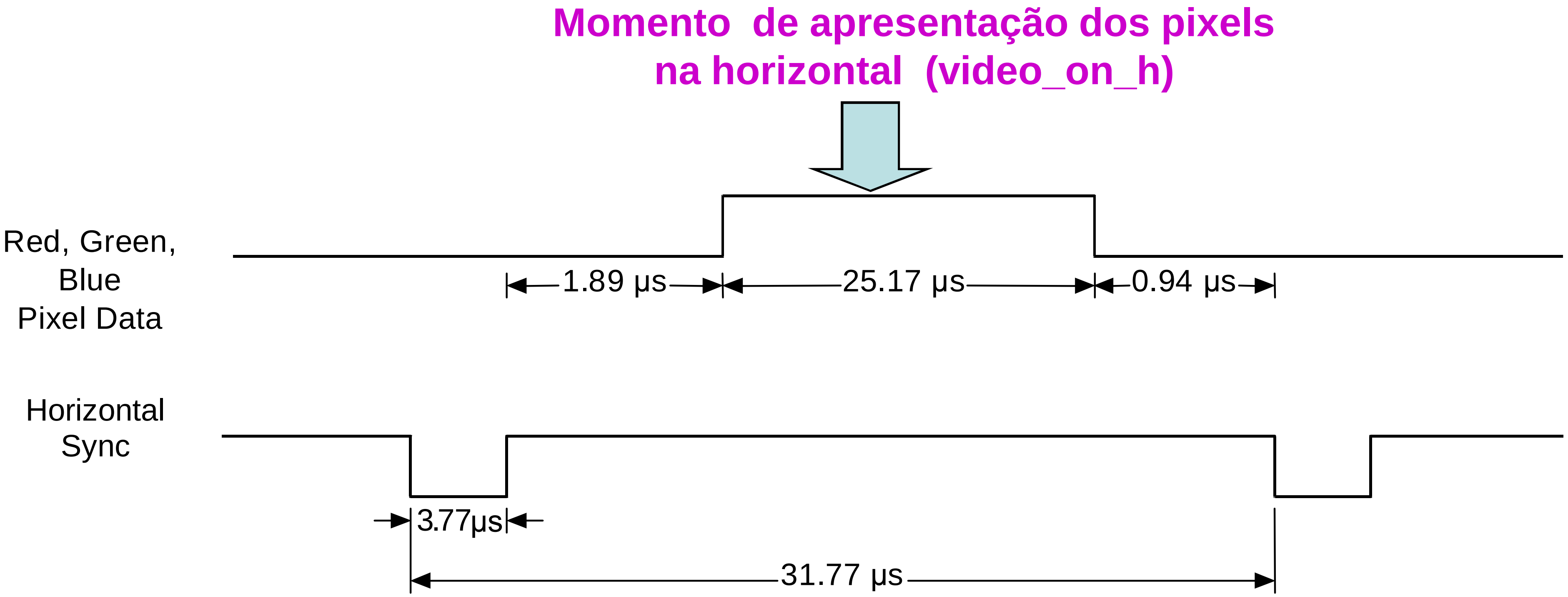
Illustration by Steve McGuire



DE2

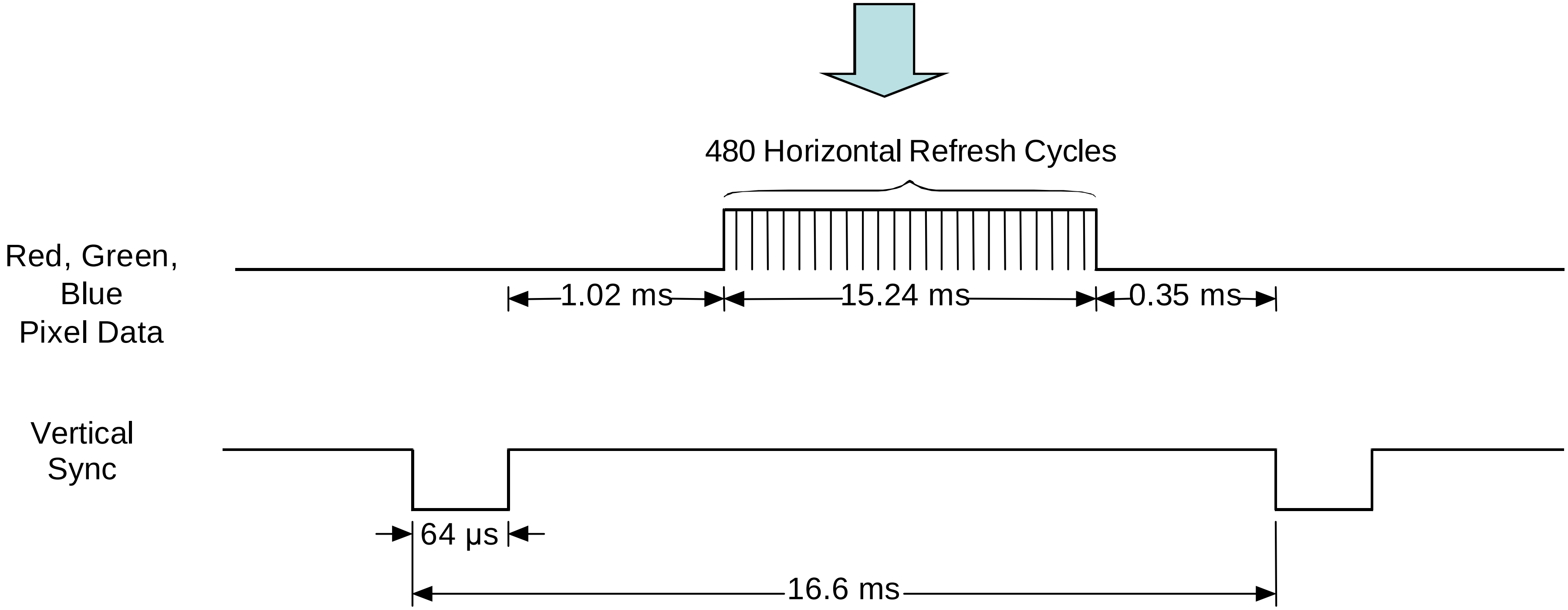
VGA_CLK	PIN_B8	VGA Clock
VGA_BLANK	PIN_D6	VGA BLANK
VGA_HS	PIN_A7	VGA H_SYNC
VGA_VS	PIN_D8	VGA V_SYNC
VGA_SYNC	PIN_B7	VGA SYNC

O vídeo VGA possui 5 sinais de entradas: 3 analógicos (RGB: 0,7 a 1,0 V) e 2 digitais para controlar o deslocamento do feixe de elétrons na horizontal (h_sync) e vertical (v_sync).

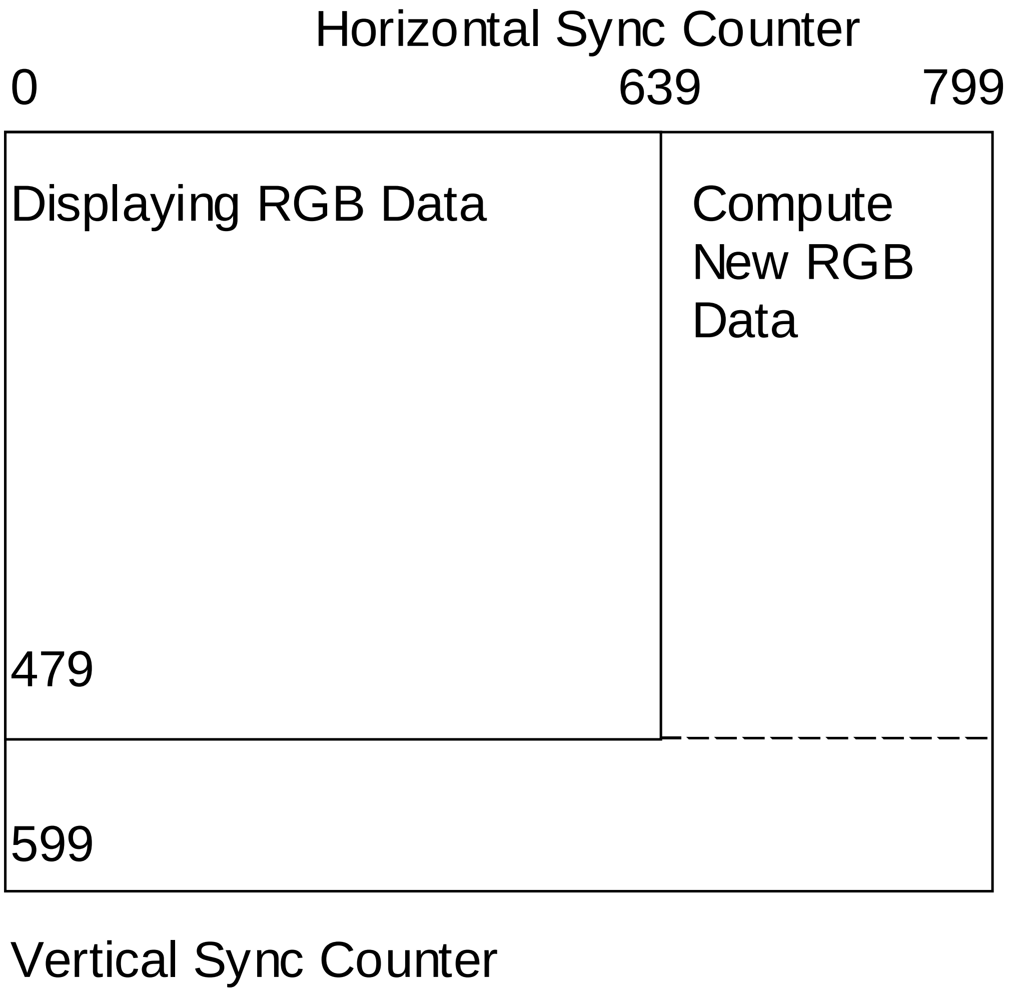


Padrão de geração do sinal digital h_sync para deslocar o feixe de elétrons na horizontal.

Momento de apresentação dos pixels na vertical (video_on_v)



Padrão de geração do sinal digital v_sync para deslocar o feixe de elétrons na vertical.



Horizontal Sync Counter

0

639

799

Displaying RGB Data

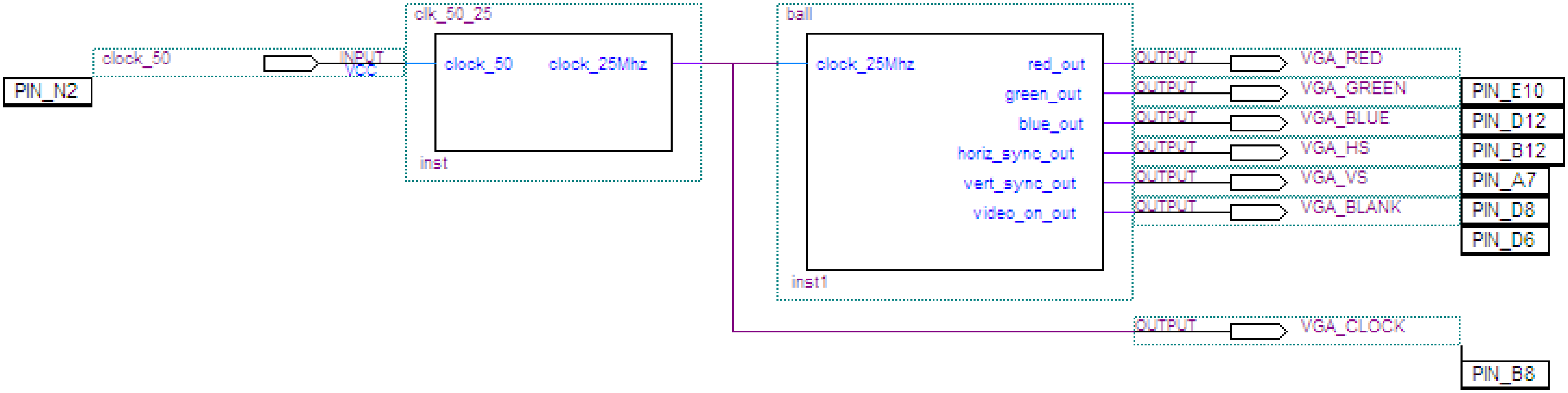
Compute New RGB Data

479

599

Vertical Sync Counter

Geração de h_sync e v_sync para apresentar um frame de vídeo.



```
library IEEE;
use  IEEE.STD_LOGIC_1164.all;

ENTITY clk_50_25 IS
   PORT(clock_50: IN        STD_LOGIC; clock_25Mhz:  BUFFER  STD_LOGIC  );
END clk_50_25;

ARCHITECTURE a OF clk_50_25 IS
BEGIN

process(clock_50)
begin
  if     (clock_50'event and clock_50  = '1') then clock_25Mhz  <= not clock_25Mhz;
 end if;
end process;
end;
```

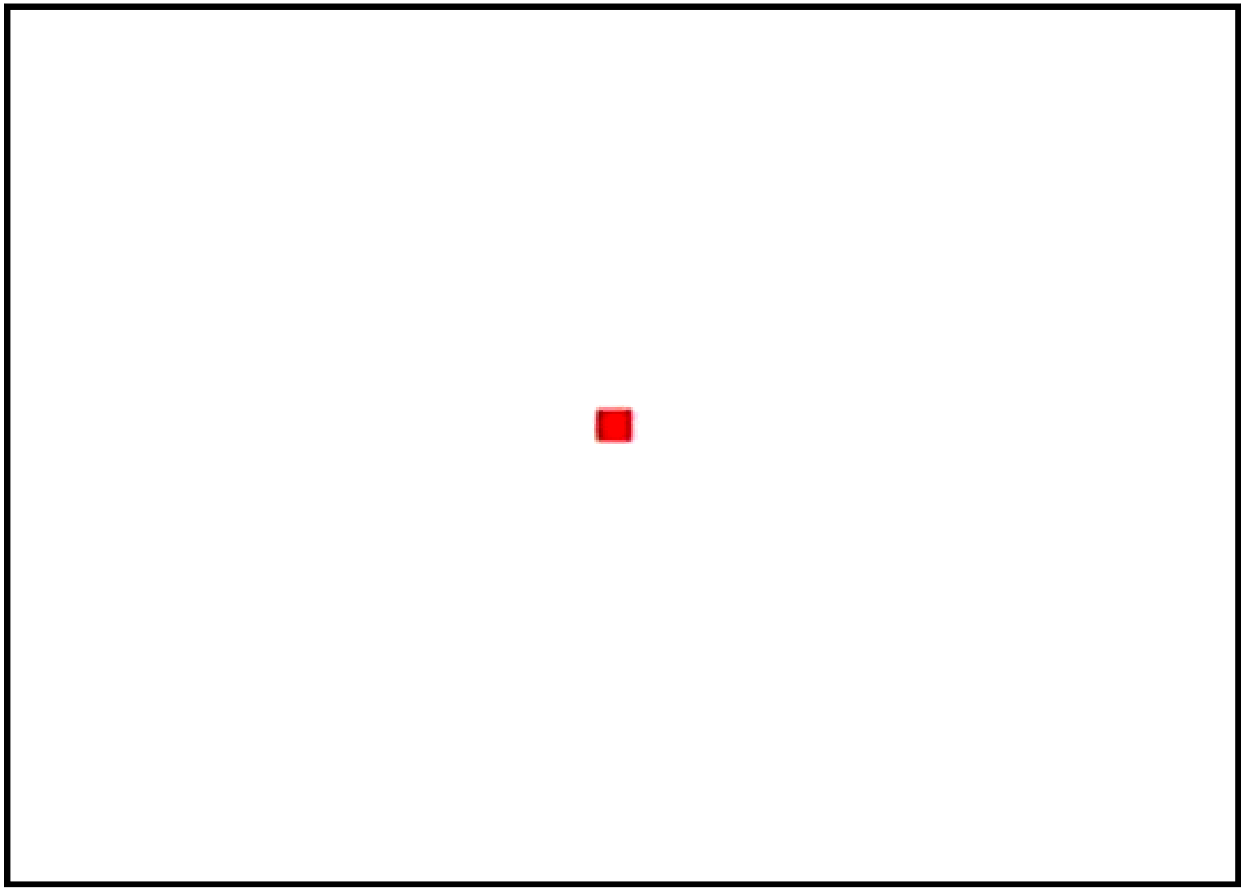
```
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY ballf IS
        PORT( clock_25Mhz           : IN        STD_LOGIC;
red_out, green_out, blue_out, horiz_sync_out, vert_sync_out, video_on_out: OUT
                                                                  STD_LOGIC);
END ENTITY ballf;

ARCHITECTURE a OF ballf IS
BEGIN
-- video_on está em '1' apenas qdo dados RGB são mostrados
-- 480 linhas (video_on_v = '1'); 640 pixels por linha (video_on_h = '1')

PROCESS
VARIABLE h_count, v_count :INTEGER RANGE 0 TO 800;
VARIABLE video_on, video_on_v, video_on_h, horiz_sync, vert_sync, red, blue, green :
STD_LOGIC;
BEGIN
        WAIT UNTIL(clock_25Mhz'EVENT) AND (clock_25Mhz='1');

        video_on := video_on_H AND video_on_V;
        h_count := h_count + 1;
        red       := '1';
        IF (h_count = 799) THEN     h_count := 0;       END IF;
```

```
--  H_count conta pixels horizontais (640 + tempo extra para sinais de
--                  mostra pixel até 639 (video_on_h := '1';)        sincronismo)
--  Horiz_sync  ----------------------------------___________--------
--  H_count      0                          640     659          755      799
case h_count is
          when 0 to 315 =>    video_on_h := '1';   horiz_sync := '1'; blue := '1';

          when 316 to 323 => video_on_h := '1';  horiz_sync := '1';

                    if (v_count > 231) and (v_count < 247) then blue := '0';
                                                        else      blue     := '1';    end if;

          when 324 to 639 => video_on_h := '1';  horiz_sync := '1'; blue := '1';

          when 640 to 658 => video_on_h := '0'; horiz_sync := '1'; blue  := '1';

          when 659 to 755 => video_on_h := '0'; horiz_sync := '0'; blue  := '1';

                              if (h_count = 699)   THEN  v_count := v_count + 1;
                              end if;
          when others =>      video_on_h := '0';  horiz_sync := '1'; blue:= '1';
end case;
green:=blue ;
```

```
-- V_count conta linha de pixels (480 + tempo extra para sinal de sincronismo)
--                  mostra pixel até 479 (video_on_v := '1';)
--  Vert_sync     ------------------------------------------------____________------------
--  V_count       0                               480        493-494         524
--
           case V_count is

                      when 0 to 479 =>
                                            video_on_v := '1';          vert_sync  := '1';

                      when 480 to 492 =>
                                            video_on_v := '0';          vert_sync  := '1';

                      when 493 to 494 =>
                                            video_on_v  := '0';         vert_sync  := '0';

                      when 495 to 524 =>
                                            video_on_v := '0';          vert_sync  := '1';

                      when others =>
                                           video_on_v := '0';           vert_sync  := '1';
                                           v_count := 0;
           END CASE;
```

```
-- Coloca sinais de vídeo em DFFs para eliminar atrasos que causa imagem
desfocada
          red_out        <= red AND video_on;
          green_out      <= green AND video_on;
          blue_out       <= blue AND video_on;
          horiz_sync_out <= horiz_sync;
          vert_sync_out  <= vert_sync;
          video_on_out   <= video_on;

END PROCESS;
END ARCHITECTURE;
```

3 – Mover o cursor *up and down*



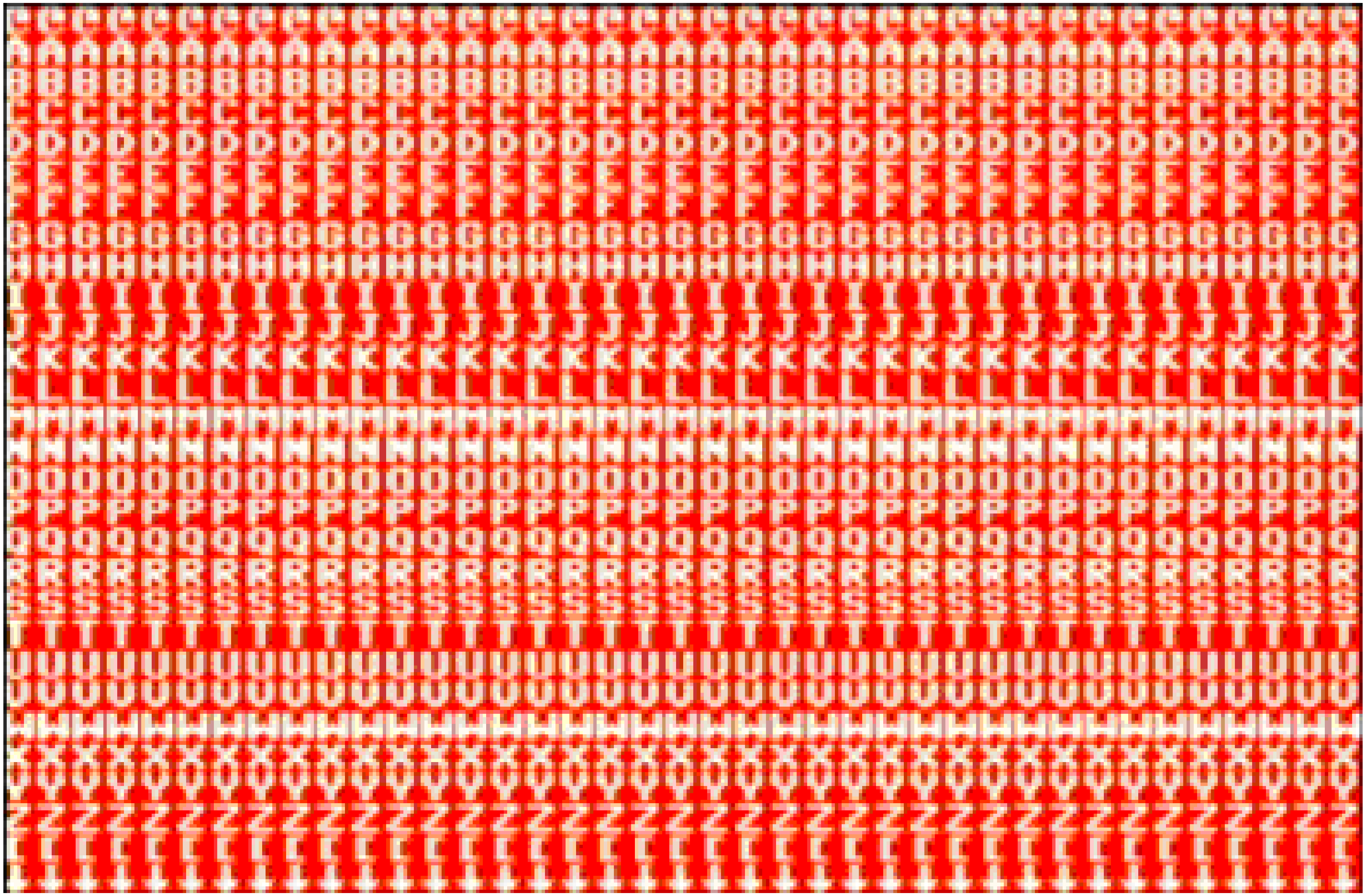
```
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY ball IS
PORT(   clock_25Mhz       : IN        STD_LOGIC;
red_out, green_out, blue_out, horiz_sync_out, vert_sync_out, video_on_out: OUT
STD_LOGIC);
END ENTITY ball;

ARCHITECTURE a OF ball IS
BEGIN
PROCESS
VARIABLE h_count, v_count, aux, aux2 :INTEGER RANGE 0 TO 800;
VARIABLE video_on, video_on_v, video_on_h, horiz_sync, vert_sync, red, blue,
green, dir : STD_LOGIC;
BEGIN
        WAIT UNTIL(clock_25Mhz'EVENT) AND (clock_25Mhz='1');

        video_on := video_on_H AND video_on_V;
        h_count := h_count + 1;

        IF (h_count = 799) THEN h_count := 0; END IF;
```

```
case h_count is

        when 0 to 315 =>    video_on_h := '1';    horiz_sync := '1';     red  := '1';

        when 316 to 323 => video_on_h := '1';   horiz_sync := '1';

                        if (v_count > aux ) and (v_count < aux2)  then   red  := '0
                        else       red        := '1';     end if;

        when 324 to 639 =>   video_on_h := '1';    horiz_sync := '1';   red  := '1';

        when 640 to 658 =>   video_on_h := '0';    horiz_sync := '1';   red  := '1';

        when 659 to 755 =>   video_on_h := '0';    horiz_sync := '0';   red  := '1';

                             IF (h_count = 699)   THEN       v_count := v_count+1;
                             END IF;

        when others =>       video_on_h := '0';   horiz_sync := '1';    red := '1';

end case;

green    := '1';
blue     := red;
```

```
case V_count is

                  when 0 to 479 =>      video_on_v := '1';    vert_sync  := '1';

                  when 480 to 492 =>    video_on_v := '0';    vert_sync  := '1';

                  when 493 to 494 =>    video_on_v := '0';    vert_sync  := '0';

                  when 495 to 524 =>    video_on_v := '0';    vert_sync  := '1';

                  when others =>        video_on_v := '0';    vert_sync  := '1';
                                        v_count := 0;
                                    if (dir = '0' ) then
                                             if (aux < 464) then  aux   := aux + 8;
                                                                    aux2 := aux + 8;
                                             else dir := '1‘;       end if;
                                    else
                                             if (aux > 0 ) then  aux2 := aux   - 8;
                                                                  aux    := aux2 - 8;
                                             else dir := '0';      end if;
                                    end if;
end case;
```

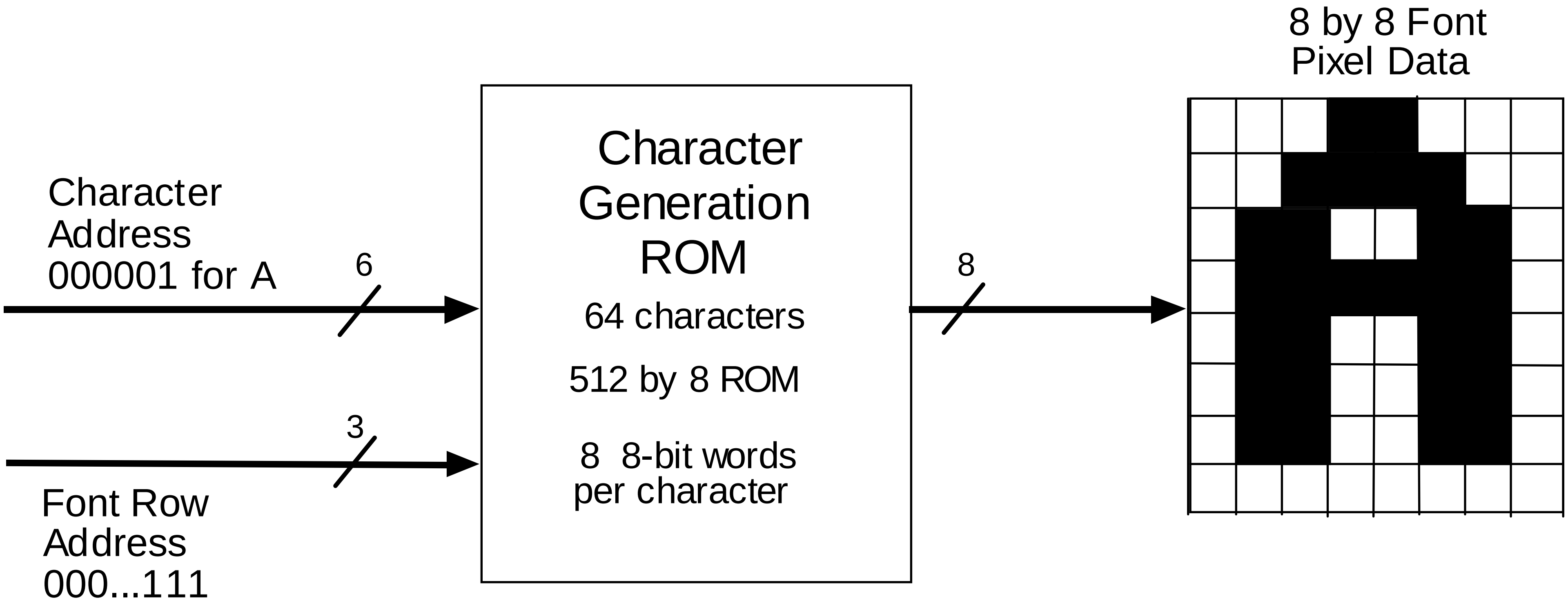
```
-- Coloca sinais de vídeo em DFFs para eliminar atrasos que causam imagem
-- desfocada
             red_out         <= red AND video_on;
             green_out       <= green AND video_on;
             blue_out        <= blue AND video_on;
             horiz_sync_out  <= horiz_sync;
             vert_sync_out   <= vert_sync;
             video_on_out     <= video_on;

END PROCESS;
END ARCHITECTURE;
```

4 – Enviar caracteres para monitor do PC



```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[
```

Geração de caracteres



Character Address 000001 for A

6

Font Row Address 000...111

3

Character Generation ROM

64 characters

512 by 8 ROM

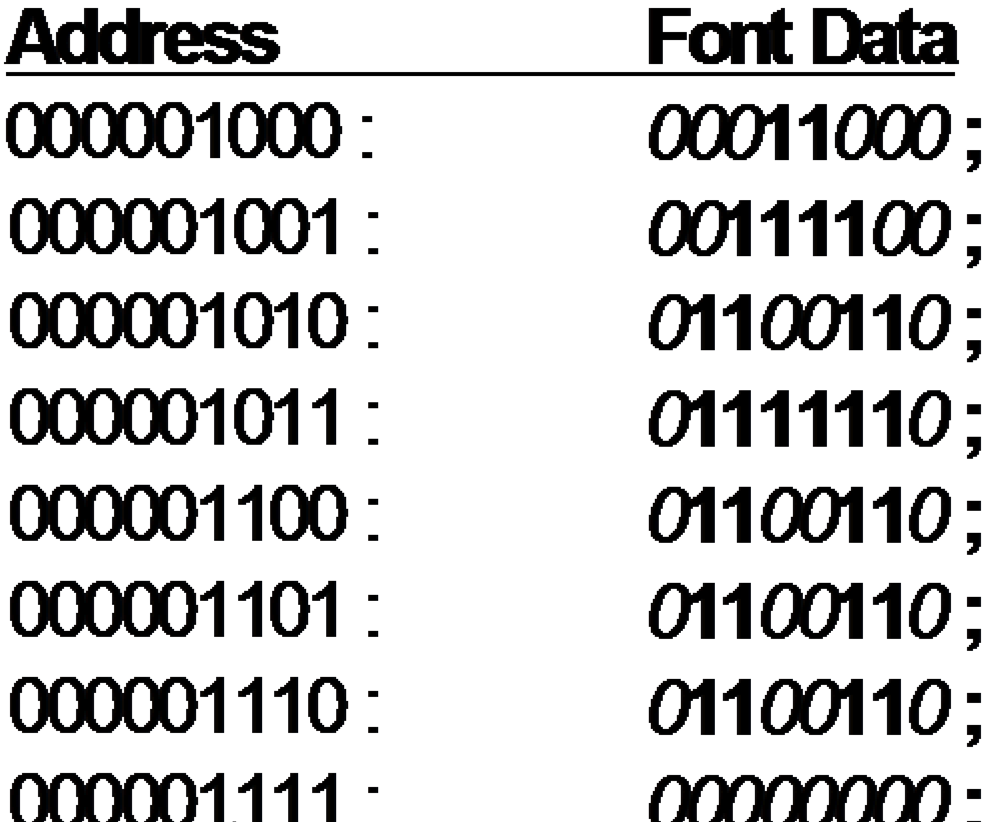
8 8-bit words per character

8

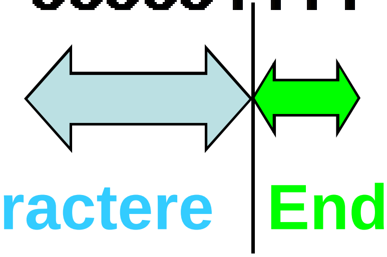
8 by 8 Font Pixel Data

8 linhas x 64 (2^6) caracteres = 512 bytes

Geração de caracteres



Address	Font Data
000001000 :	00011000 ;
000001001 :	00111100 ;
000001010 :	01100110 ;
000001011 :	01111110 ;
000001100 :	01100110 ;
000001101 :	01100110 ;
000001110 :	01100110 ;
000001111 :	00000000 ;



End. Caractere

End. da Linha

Esquema de geração do caractere

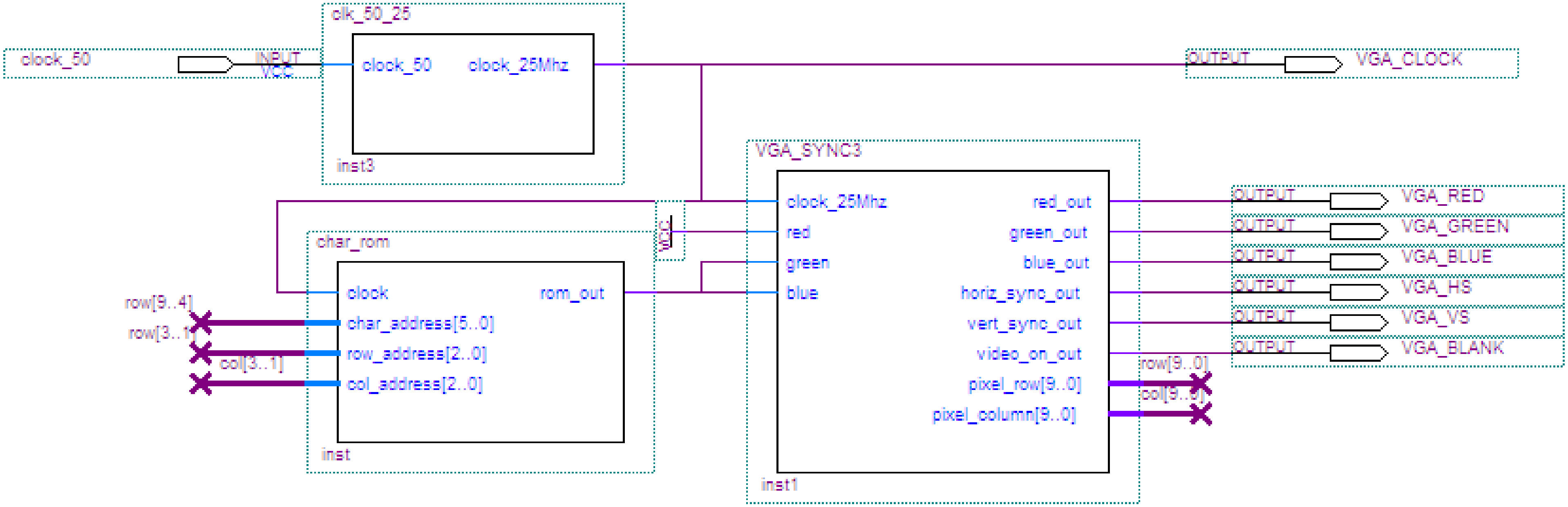
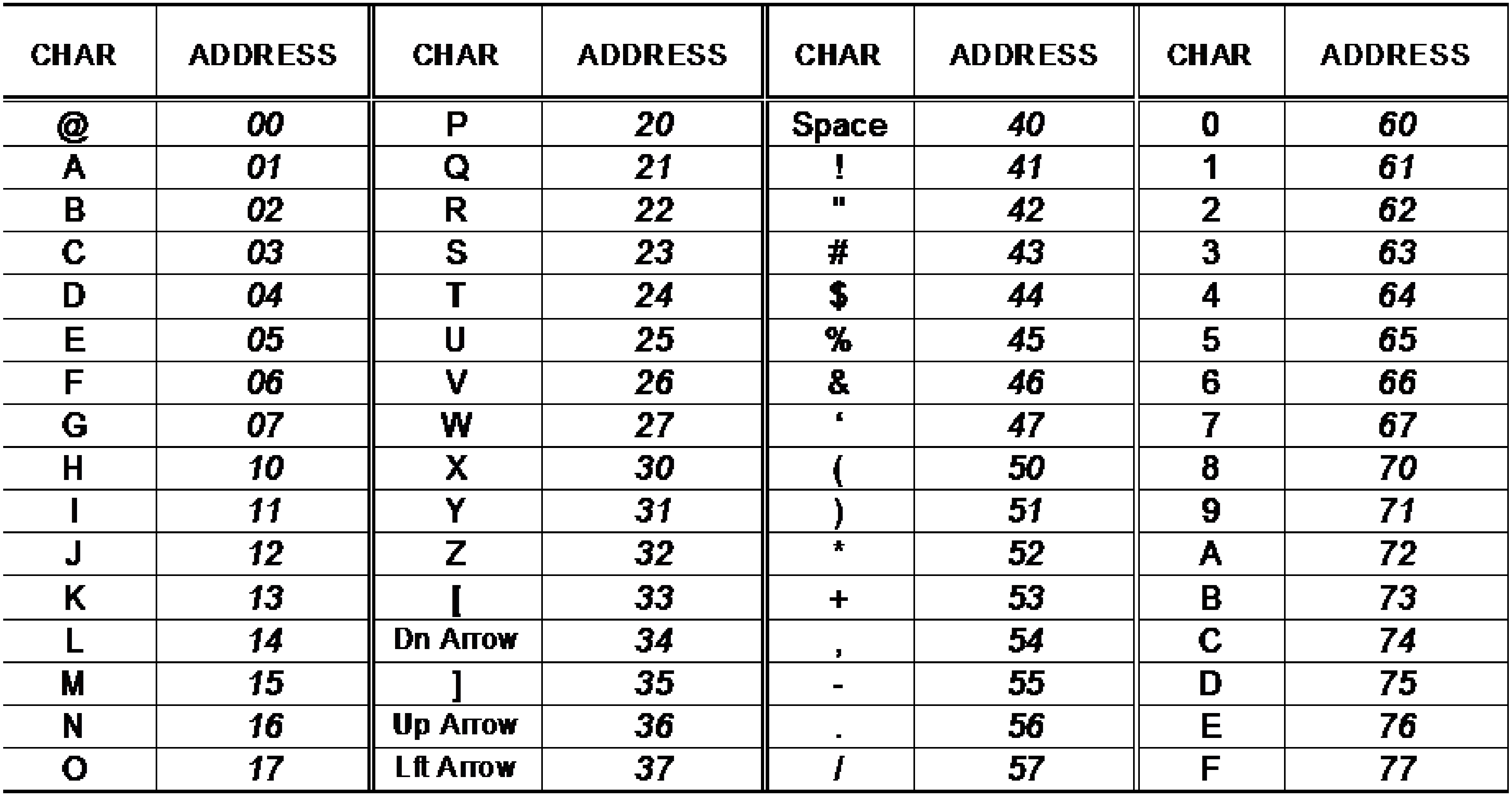


Tabela de endereço da memória para geração de caracteres

Table 9.1 Character Address Map for 8 by 8 Font ROM.



CHAR	ADDRESS	CHAR	ADDRESS	CHAR	ADDRESS	CHAR	ADDRESS
@	*00*	P	*20*	Space	*40*	0	*60*
A	*01*	Q	*21*	!	*41*	1	*61*
B	*02*	R	*22*	"	*42*	2	*62*
C	*03*	S	*23*	#	*43*	3	*63*
D	*04*	T	*24*	$	*44*	4	*64*
E	*05*	U	*25*	%	*45*	5	*65*
F	*06*	V	*26*	&	*46*	6	*66*
G	*07*	W	*27*	‘	*47*	7	*67*
H	*10*	X	*30*	(	*50*	8	*70*
I	*11*	Y	*31*	)	*51*	9	*71*
J	*12*	Z	*32*	*	*52*	A	*72*
K	*13*	[	*33*	+	*53*	B	*73*
L	*14*	Dn Arrow	*34*	,	*54*	C	*74*
M	*15*	]	*35*	-	*55*	D	*75*
N	*16*	Up Arrow	*36*	.	*56*	E	*76*
O	*17*	Lft Arrow	*37*	/	*57*	F	*77*

```
Depth = 512;
Width = 8;
Address_radix = oct;
Data_radix = bin;
% Character Generator ROM Data %
Content
Begin
000  : 00111100 ; %   ****   %
001  : 01100110 ; %  **  **  %
002  : 01101110 ; %  ** ***  %
003  : 01101110 ; %  ** ***  %
004  : 01100000 ; %  **      %
005  : 01100010 ; %  **   *  %
006  : 00111100 ; %   ****   %
007  : 00000000 ; %          %

010  : 00011000 ; %    **    %
011  : 00111100 ; %   ****   %
012  : 01100110 ; %  **  **  %
013  : 01111110 ; %  ******  %
014  : 01100110 ; %  **  **  %
015  : 01100110 ; %  **  **  %
016  : 01100110 ; %  **  **  %
017  : 00000000 ; %          %
```

Conteúdo da Memória

TCGROM.MIF

Instanciação de componente lpm_rom da biblioteca da Altera

Altera recomenda o uso da *megafunction altsyncram* vez da *megafunction lpm_rom*. Esta *megafunction* é fornecida apenas para compatibilidade com versões anteriores

LPM ROM

```
component LPM_ROM
    generic (
                LPM_WIDTH : natural;   -- MUST be greater than 0
                LPM_WIDTHAD : natural;   -- MUST be greater than 0
                LPM_NUMWORDS : natural := 0;
                LPM_ADDRESS_CONTROL : string := "REGISTERED";
                LPM_OUTDATA : string := "REGISTERED";
                LPM_FILE : string
           );

        port (ADDRESS : in STD_LOGIC_VECTOR(LPM_WIDTHAD-1 downto 0);
              INCLOCK : in STD_LOGIC := '0';
              Q : out STD_LOGIC_VECTOR(LPM_WIDTH-1 downto 0));
end component;
```

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY char_rom IS
        PORT
        (       clock           :in   STD_LOGIC;
                char_address    :in   STD_LOGIC_VECTOR (5 DOWNTO 0);
                row_address     :in   STD_LOGIC_VECTOR (2 DOWNTO 0);
                col_address     :in   STD_LOGIC_VECTOR (2 DOWNTO 0);
                rom_out         :out STD_LOGIC
        );
END ENTITY char_rom;
```

```
ARCHITECTURE SYN OF char_rom IS

          SIGNAL rom_data                    : STD_LOGIC_VECTOR (7 DOWNTO 0);
          SIGNAL rom_address                 : STD_LOGIC_VECTOR (8 DOWNTO 0);
BEGIN
lpm_rom_component : lpm_rom
          GENERIC MAP (
                    lpm_widthad => 9,
                    lpm_numwords => 512,
                    lpm_outdata => "UNREGISTERED",
                    lpm_address_control => "REGISTERED",
                    lpm_file => "TCGROM.MIF",
                    lpm_width => 8
                            )
          PORT MAP (          inclock => clock,
                              address => rom_address,
                              q => rom_data);

rom_address <= char_address & row_address;

                              -- Seleção da coluna para definir bit de saída
rom_out <= rom_data ( (CONV_INTEGER(NOT col_address(2 DOWNTO 0))));
-- not faz endereçamento da esquerda para a direita (000 => 111 ;001 => 110; etc
END ARCHITECTURE ;
```

```
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY VGA_SYNC3 IS
  PORT( clock_25Mhz, red, green, blue                          : IN  STD_LOGIC;
        red_out, green_out, blue_out                           : OUT STD_LOGIC;
        horiz_sync_out, vert_sync_out, video_on_out : OUT STD_LOGIC;
        pixel_row, pixel_column: OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END ENTITY VGA_SYNC3;

ARCHITECTURE a OF VGA_SYNC3 IS
BEGIN
PROCESS
        VARIABLE h_count, v_count :INTEGER RANGE 0 TO 800;
        VARIABLE horiz_sync, vert_sync : STD_LOGIC;
        VARIABLE video_on, video_on_v, video_on_h : STD_LOGIC;
BEGIN
        WAIT UNTIL(clock_25Mhz'EVENT) AND (clock_25Mhz='1');
        video_on := video_on_H AND video_on_V;
        h_count := h_count + 1;
        IF (h_count = 799) THEN  h_count := 0;      END IF;
        pixel_column <= conv_std_logic_vector(h_count,10);  -- End. da coluna
        pixel_row <= conv_std_logic_vector(v_count,10); -- End. do char e row
```

```
--  H_count conta pixels horizontais (640 + tempo extra para sinais de
sincronismo)
--
--  Horiz_sync  -----------------------------------__________--------
--  H_count       0            640          659       755    799
--
          case h_count is


                    when 0 to 639 =>     video_on_h := '1';       horiz_sync := '1';

                    when 640 to 658 => video_on_h := '0';       horiz_sync := '1';

                    when 659 to 755 => video_on_h := '0';       horiz_sync := '0';

                        IF (h_count = 699)   THEN  v_count := v_count + 1; END IF;

                    when others =>       video_on_h := '0';       horiz_sync := '1';


          END CASE;
```

```
-- V_count conta linha de pixels (480 + tempo extra para sinal de sincronismo)
--  Vert_sync      -------------------------------------------_______------------
--  V_count        0                              480         493-494        524
          case V_count is

                    when 0 to 479 =>     video_on_v := '1';      vert_sync  := '1';

                    when 480 to 492 => video_on_v := '0';       vert_sync  := '1';

                    when 493 to 494 => video_on_v := '0';       vert_sync  := '0';

                    when 495 to 524 => video_on_v := '0';       vert_sync  := '1';

                    when others =>       video_on_v := '0';      vert_sync  := '1';
                                       v_count := 0;
          END CASE;
-- sinais de vídeo em DFFs para eliminar atrasos que causa imagem desfocada
                    red_out            <= red AND video_on;
                    green_out          <= green AND video_on;
                    blue_out           <= blue AND video_on;
                    horiz_sync_out  <= horiz_sync;
                    vert_sync_out    <= vert_sync;
                    video_on_out     <= video_on;
END PROCESS;
END ARCHITECTURE:
```

5 – Enviar mensagem de texto para monitor do PC

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS
CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

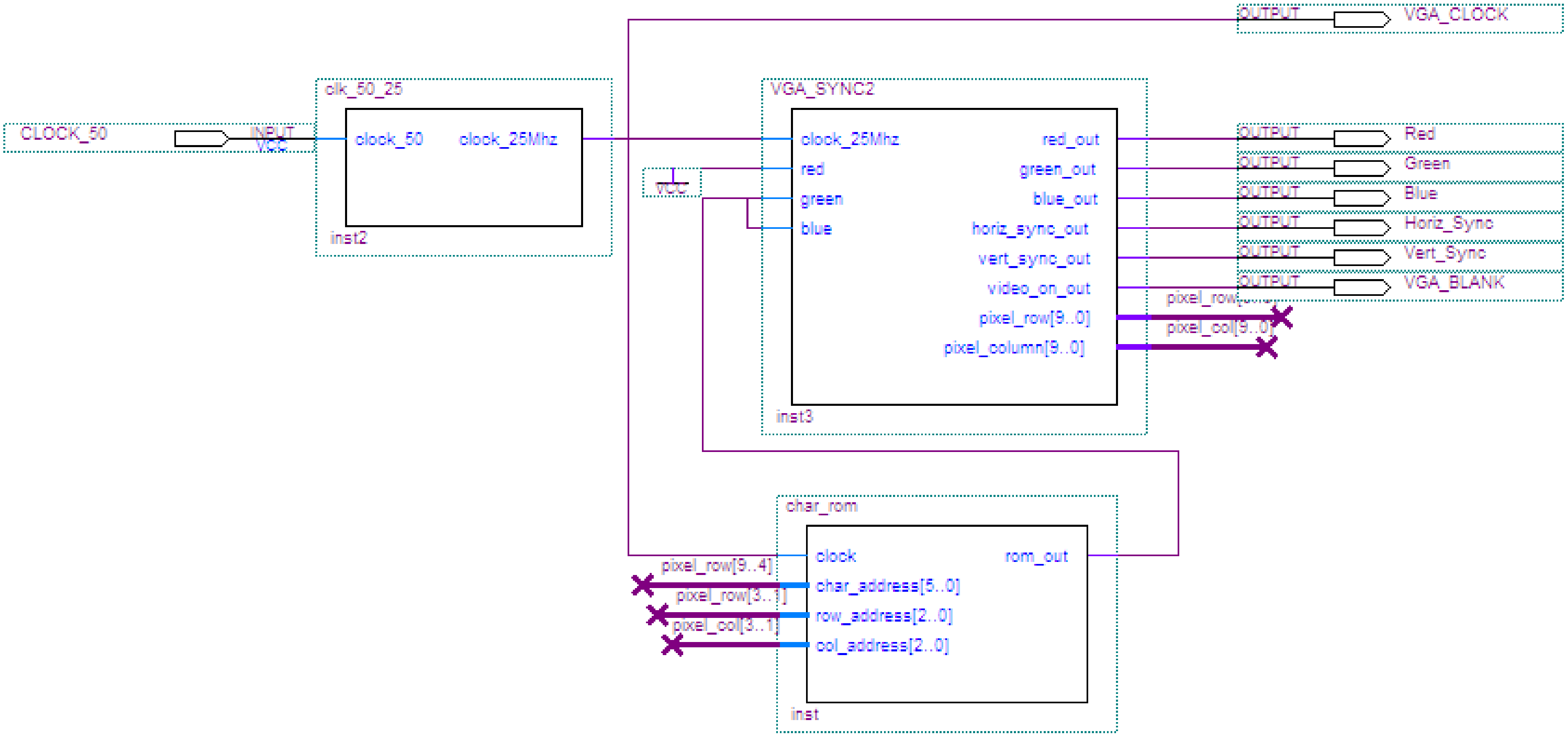
CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS

CURSO DISPOSITIVOS LOGICO-PROGRAMAVEIS


OUTPUT
VGA_CLOCK
clk_50_25
VGA_SYNC2
CLOCK_50
INPUT
VCC
clock_50
clock_25Mhz
inst2
clock_25Mhz
red
green
blue
red_out
green_out
blue_out
horiz_sync_out
vert_sync_out
video_on_out
pixel_row[9..0]
pixel_column[9..0]
VCC
inst3
OUTPUT
Red
OUTPUT
Green
OUTPUT
Blue
OUTPUT
Horiz_Sync
OUTPUT
Vert_Sync
OUTPUT
VGA_BLANK
pixel_row[9..0]
pixel_col[9..0]
char_rom
clock
rom_out
pixel_row[9..4]
char_address[5..0]
pixel_row[3..1]
row_address[2..0]
pixel_col[3..1]
col_address[2..0]
inst

```
library IEEE;
use  IEEE.STD_LOGIC_1164.all;

package mensagem is
--  40 LETRAS  -- Cd. LETRA CONTEM 6 BITS

constant NRO_LETRAS:  INTEGER :=  40;

 type PALAVRA is array (0 to NRO_LETRAS-1) of STD_LOGIC_VECTOR (5 downto 0);

              constant DADO:  PALAVRA:= PALAVRA'(
--                                  CODIGO LETRAS                                          TEXTO
                            O"40",O"03",O"25",O"22",O"23",O"17",O"40",O"04",    --  Curso D
                            O"11",O"23",O"20",O"17",O"23",O"11",O"24",O"11",    -- ispositi
                            O"26",O"17",O"23",O"40",O"14",O"17",O"07",O"11",    -- vos Logi
                            O"03",O"17",O"55",O"20",O"22",O"17",O"07",O"22",    -- co-Progr
                            O"01",O"15",O"01",O"26",O"05",O"11",O"23",O"40"     -- amaveis
                                                  );

end package mensagem;
```

```
use work.mensagem.all;
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY  VGA_SYNC2 IS
PORT(clock_25Mhz, red, green, blue                          : IN     STD_LOGIC;
red_out, green_out, blue_out, horiz_sync_out, vert_sync_out, video_on_out
                                                            : OUT STD_LOGIC;
pixel_row, pixel_column: OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END ENTITY VGA_SYNC2;

ARCHITECTURE a OF VGA_SYNC2 IS
        BEGIN
PROCESS
        VARIABLE CONTA_PAL:INTEGER RANGE 0 TO NRO_LETRAS-1 :=0;
        VARIABLE AUX:STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
        VARIABLE CONTA_LIN:STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
        VARIABLE h_count, v_count :INTEGER RANGE 0 TO 800;
        VARIABLE horiz_sync, vert_sync,video_on, video_on_v, video_on_h :
                                                                   STD_LOGIC;
```

```
BEGIN
        WAIT UNTIL(clock_25Mhz'EVENT) AND (clock_25Mhz='1');
        video_on := video_on_H AND video_on_V;

        h_count := h_count + 1;
        AUX:=AUX+1;
        IF (AUX = 0) THEN CONTA_PAL:=CONTA_PAL+1; END IF;

        IF (h_count = 799) THEN
                                h_count         := 0;
                                CONTA_PAL       :=0;
                                AUX             := (others=> '0');

        END IF;

         pixel_row(3 downto 1)          <= CONTA_LIN(3 downto 1);
         pixel_row(9 downto 4)          <= DADO(CONTA_PAL);
         pixel_column (3 downto 1)      <= AUX(3 downto 1);
```

```
--  H_count conta pixels horizontais (640 + tempo extra para sinais de
sincronismo)
--
--  Horiz_sync  -----------------------------------__________--------
--  H_count      0              640            659        755    799
--
            case h_count is

                        when 0 to 639 =>     video_on_h := '1';       horiz_sync := '1';

                        when 640 to 658 => video_on_h := '0';       horiz_sync := '1';

                        when 659 to 755 => video_on_h := '0';       horiz_sync := '0';

                            IF (h_count = 699)  THEN  v_count          := v_count + 1;
                                                       CONTA_LIN  := CONTA_LIN+1;
                            END IF;

                        when others =>   video_on_h := '0';            horiz_sync := '1';


            END CASE;
```

```
-- V_count conta linha de pixels (480 + tempo extra para sinal de
sincronismo)
--
--  Vert_sync      ------------------------------------------_______-----------
--  V_count         0                                    480    493-494         524
--
          case V_count is

                    when 0 to 479 =>     video_on_v := '1'; vert_sync  := '1';

                    when 480 to 492 => video_on_v := '0'; vert_sync  := '1';

                    when 493 to 494 => video_on_v := '0‘;  vert_sync  := '0';

                    when 495 to 524 => video_on_v := '0‘; vert_sync  := '1';

                    when others =>       video_on_v := '0'; vert_sync  := '1';
                                         v_count := 0;
                                         CONTA_LIN:= (OTHERS=>'0');
          end case;
```

```
-- Sinais de vídeo em DFFs para eliminar atrasos que causa
-- imagem desfocada

            red_out             <= red AND video_on;
            green_out           <= green AND video_on;
            blue_out            <= blue AND video_on;
            horiz_sync_out      <= horiz_sync;
            vert_sync_out       <= vert_sync;
            video_on_out        <= video_on;

END PROCESS;
END ARCHITECTURE;
```