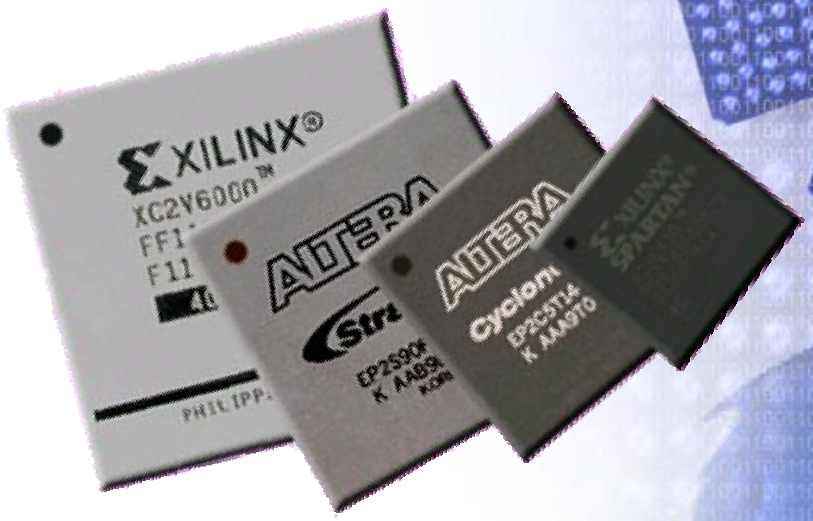
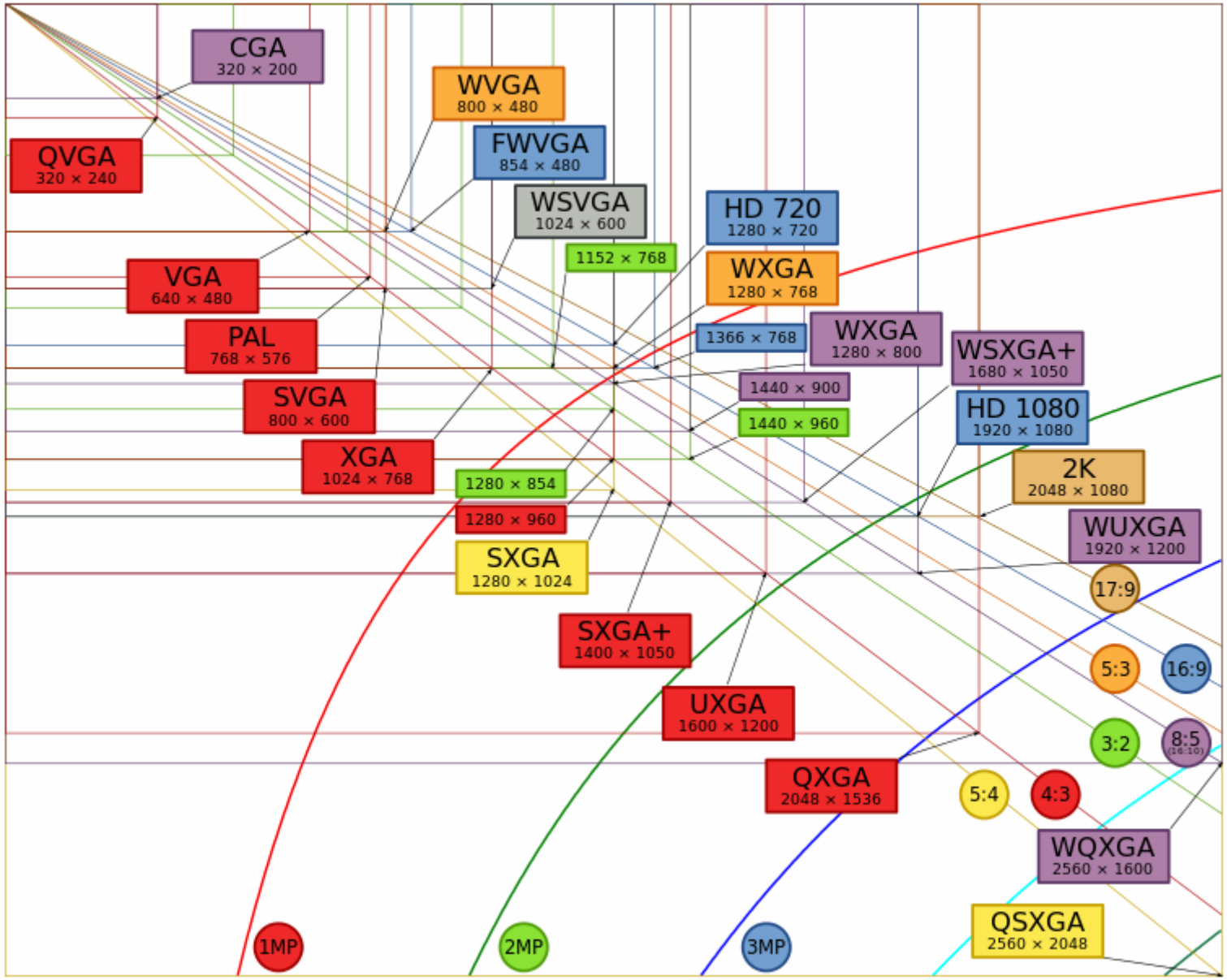
Progetto di un VGA controller



Standard VGA – come nasce..

Nel 1987 la IBM lanciò la VGA (Video Graphics Array) insieme ai suoi nuovi PC PS/2. Inizialmente i chip VGA vennero costruiti sulle motherboard PS/2 ma visto il successo e la crescente domanda, la IBM pensò bene di creare delle schede grafiche dedicate in cui alloggiava i circuiti integrati VGA. Questo standard poteva fornire una risoluzione di 640x480 a 16 colori o di 320X400 a 256 colori ed inoltre poteva gestire una speciale simulazione che rendeva il monitor monocromatico o meglio a toni di colore. Ovviamente per usare la VGA c'era bisogno di un monitor di tipo VGA. Ancora oggi, lo standard VGA viene usato ed è il minimo richiesto per un moderno PC.

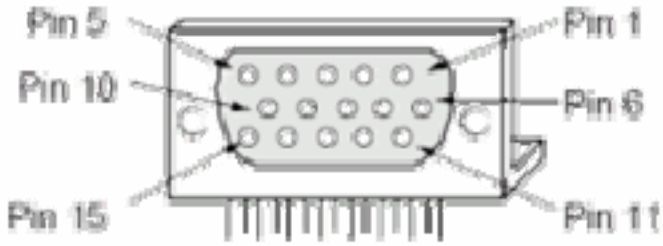
Gli standard Video



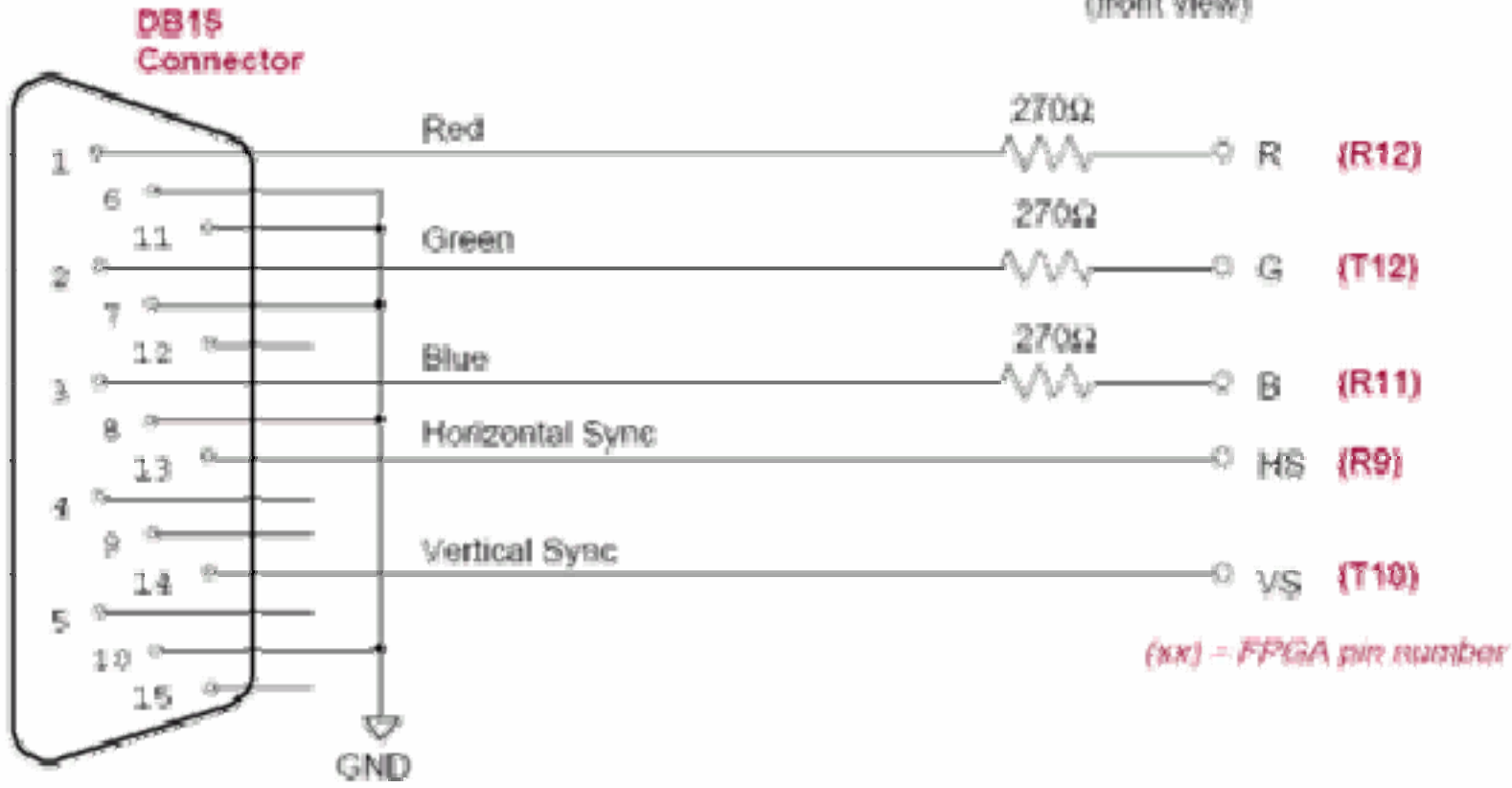
Specifiche temporali dei segnali

Tipo	Analogico
Risoluzione	640h × 480v
Frequenza di clock	25,175 MHz
Freq. Orizz.	31,469 kHz
Freq. Vert.	59,94 Hz

VGA Connector



DB15 VGA Connector
(front view)



VGA Standard

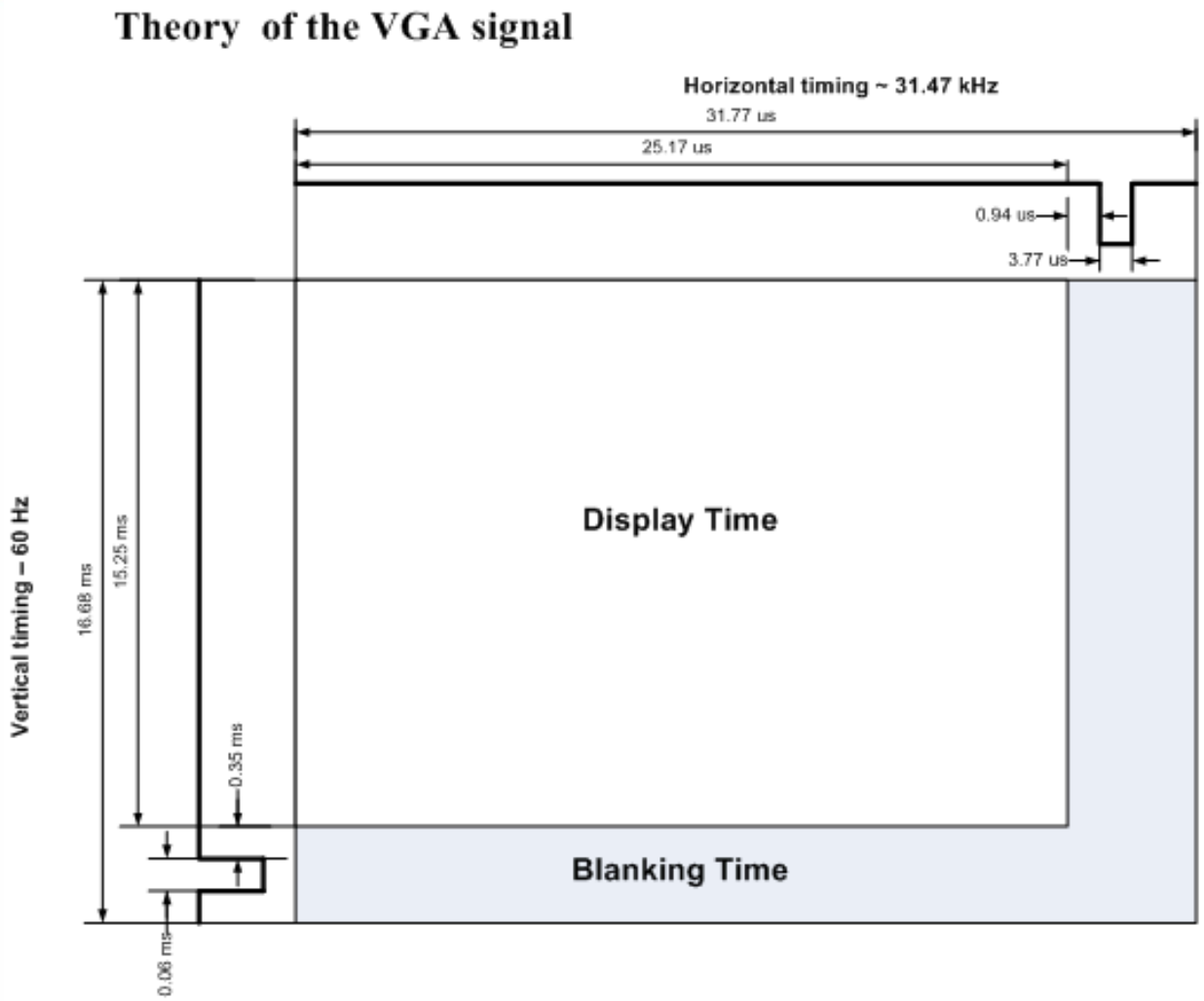


Figure 1. VGA Timing

"VGA industry standard" 640x480 pixel mode

General characteristics

```
Clock frequency 25.175 MHz
Line  frequency 31469 Hz
Field frequency 59.94 Hz
```

One line

```
  8 pixels front porch
 96 pixels horizontal sync
 40 pixels back porch
  8 pixels left border
640 pixels video
  8 pixels right border
---
800 pixels total per line
```

One field

```
  2 lines front porch
  2 lines vertical sync
 25 lines back porch
  8 lines top border
480 lines video
  8 lines bottom border
---
525 lines total per field
```

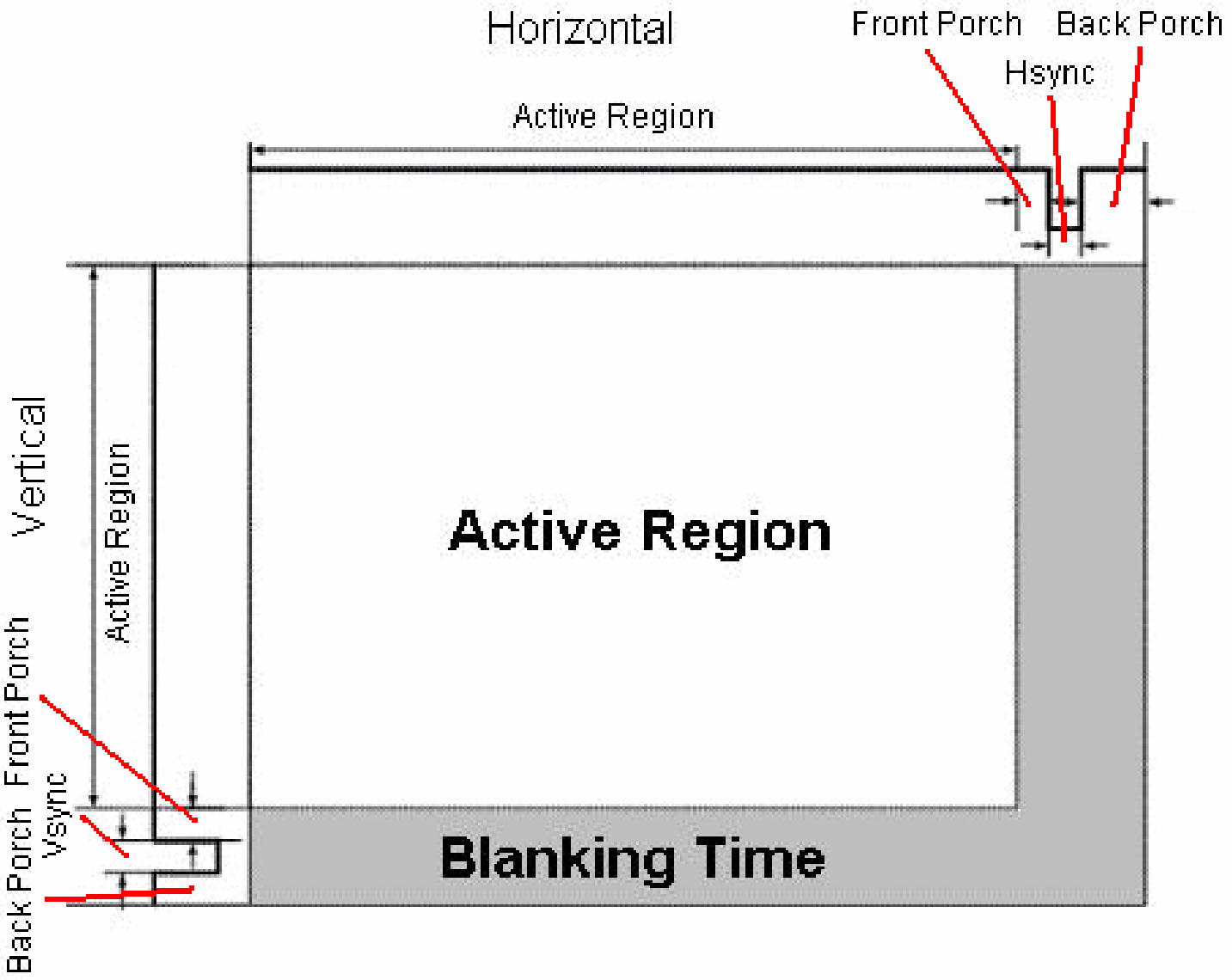
Other details

```
Sync polarity: H negative, V negative
Scan type: non interlaced.
```

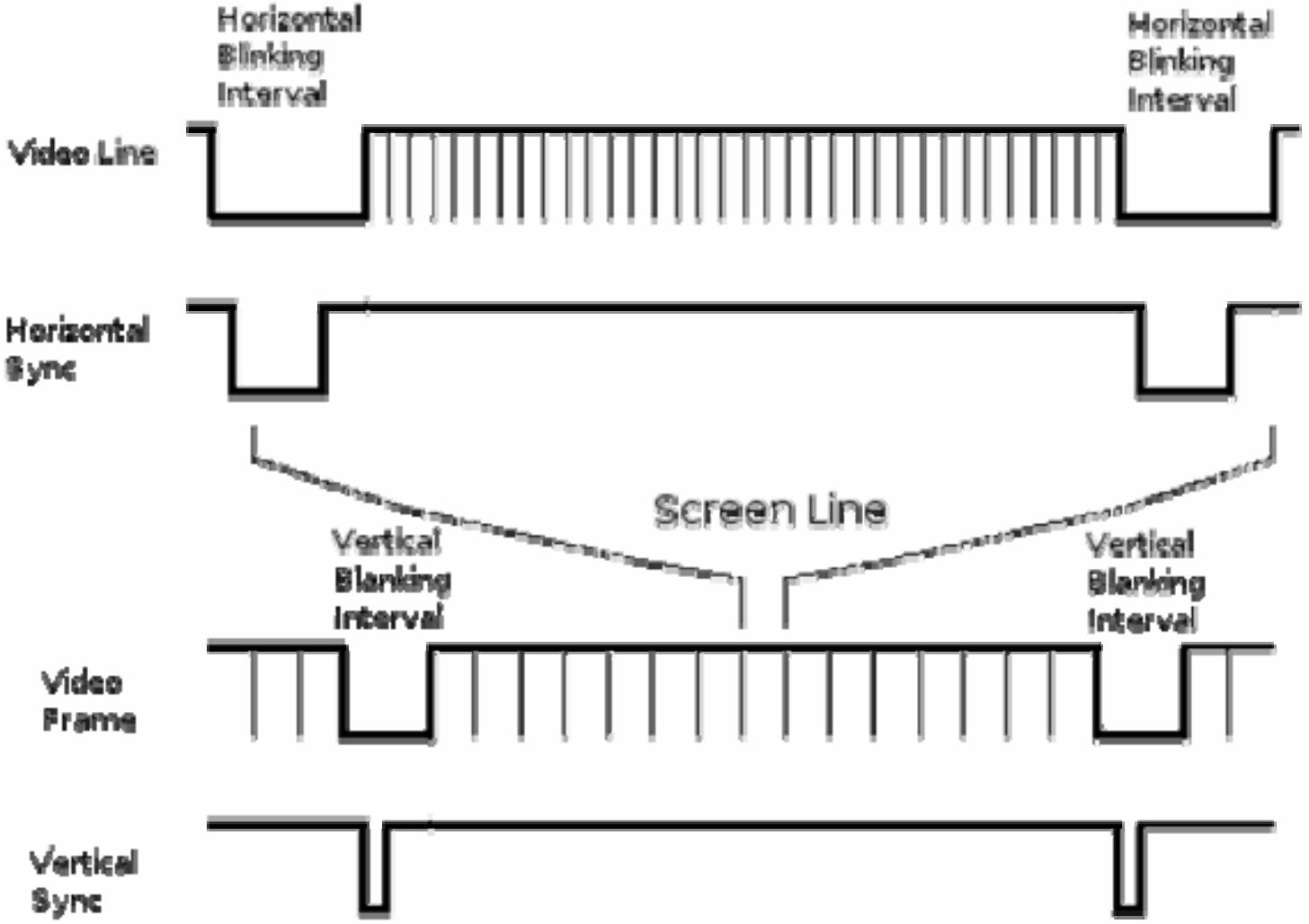
Information source

- Article "Re: VGA specifications ,where ?" posted 19 Nov 1997 to sci.electronics.design newsgroup by Jeroen Stessen

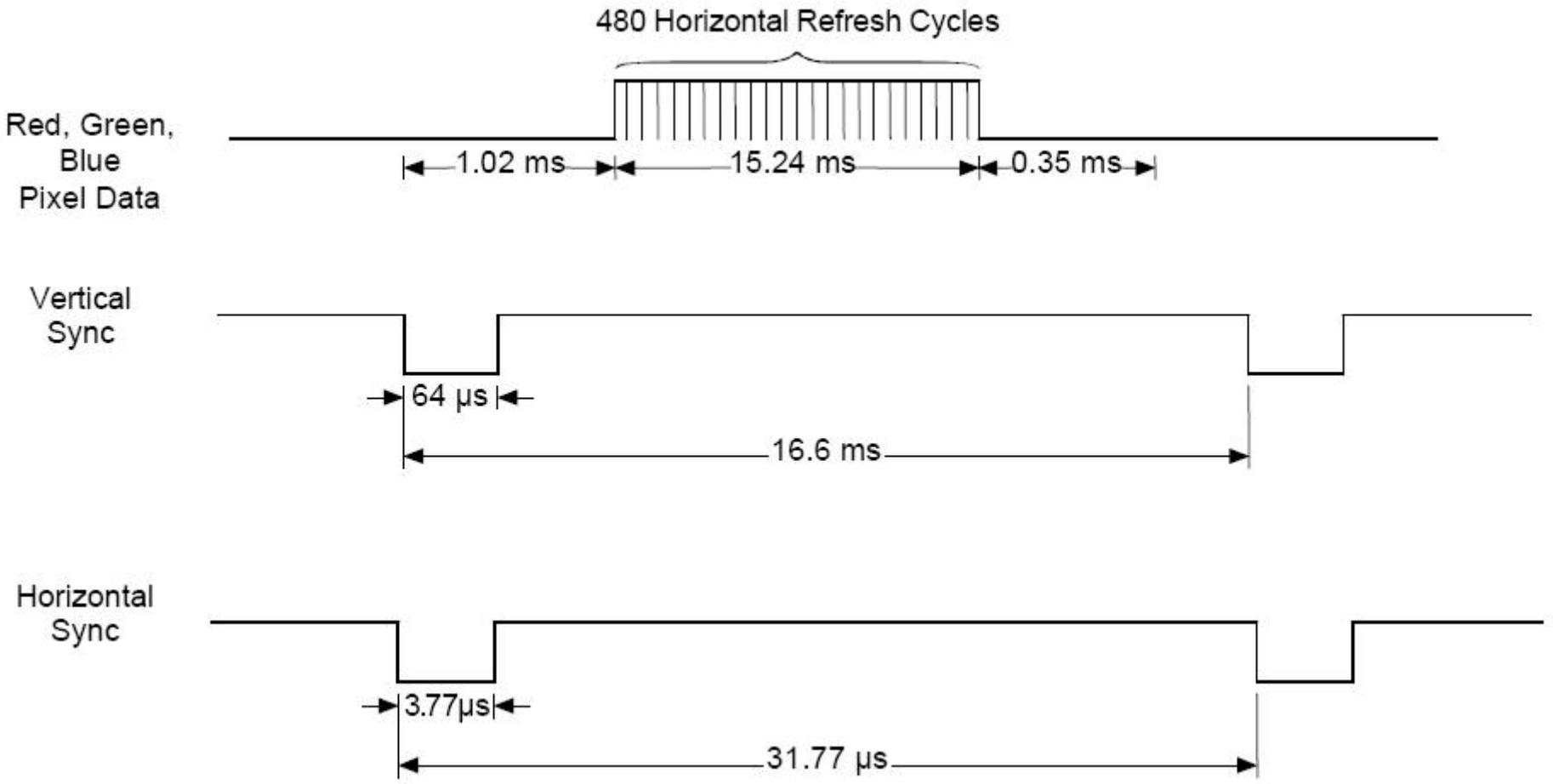
Regione visibile attiva



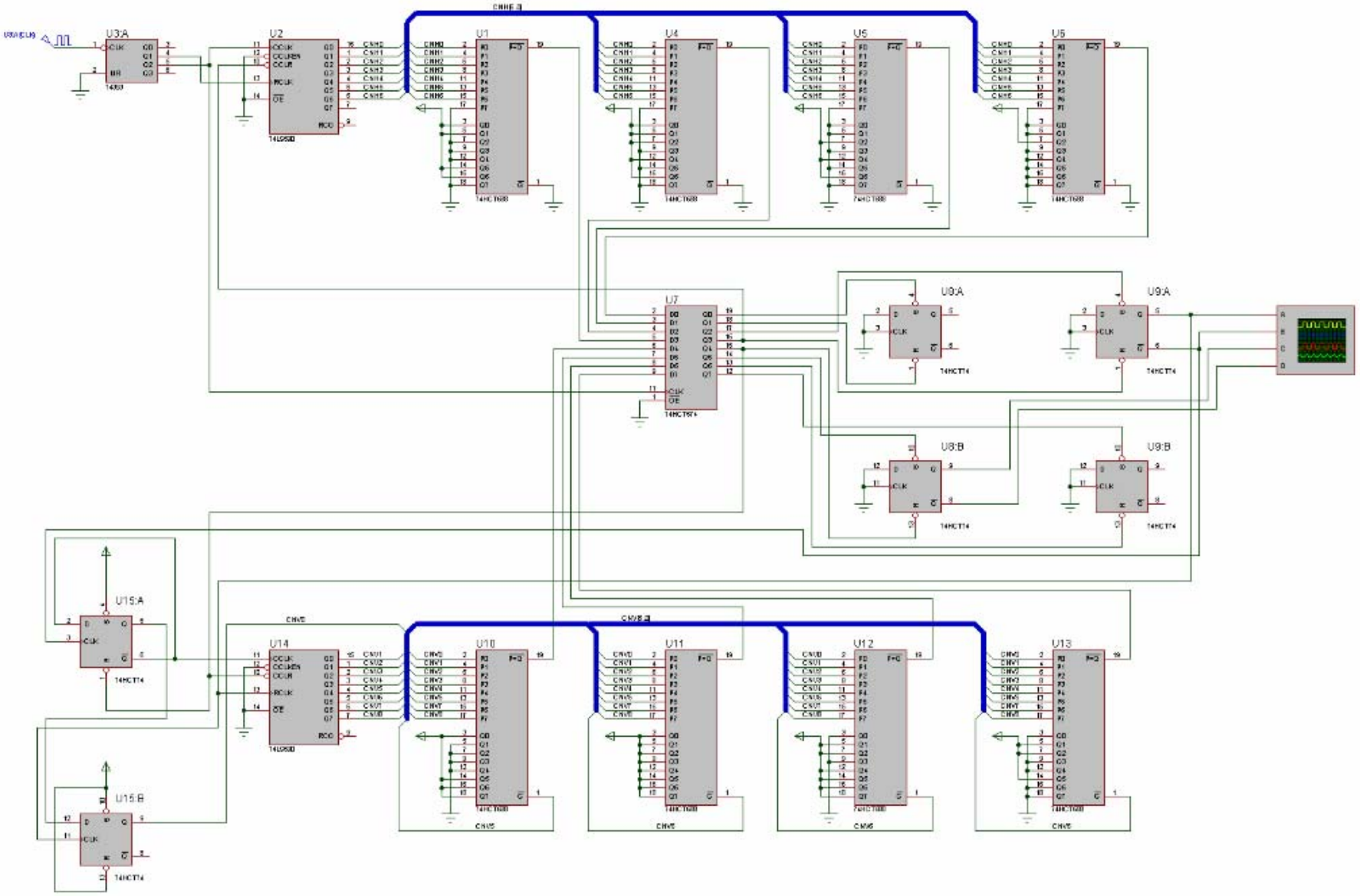
VGA Sync


Horizontal
Blinking
Interval
Horizontal
Blinking
Interval
Video Line
Horizontal
Sync
Screen Line
Vertical
Blanking
Interval
Vertical
Blanking
Interval
Video
Frame
Vertical
Sync

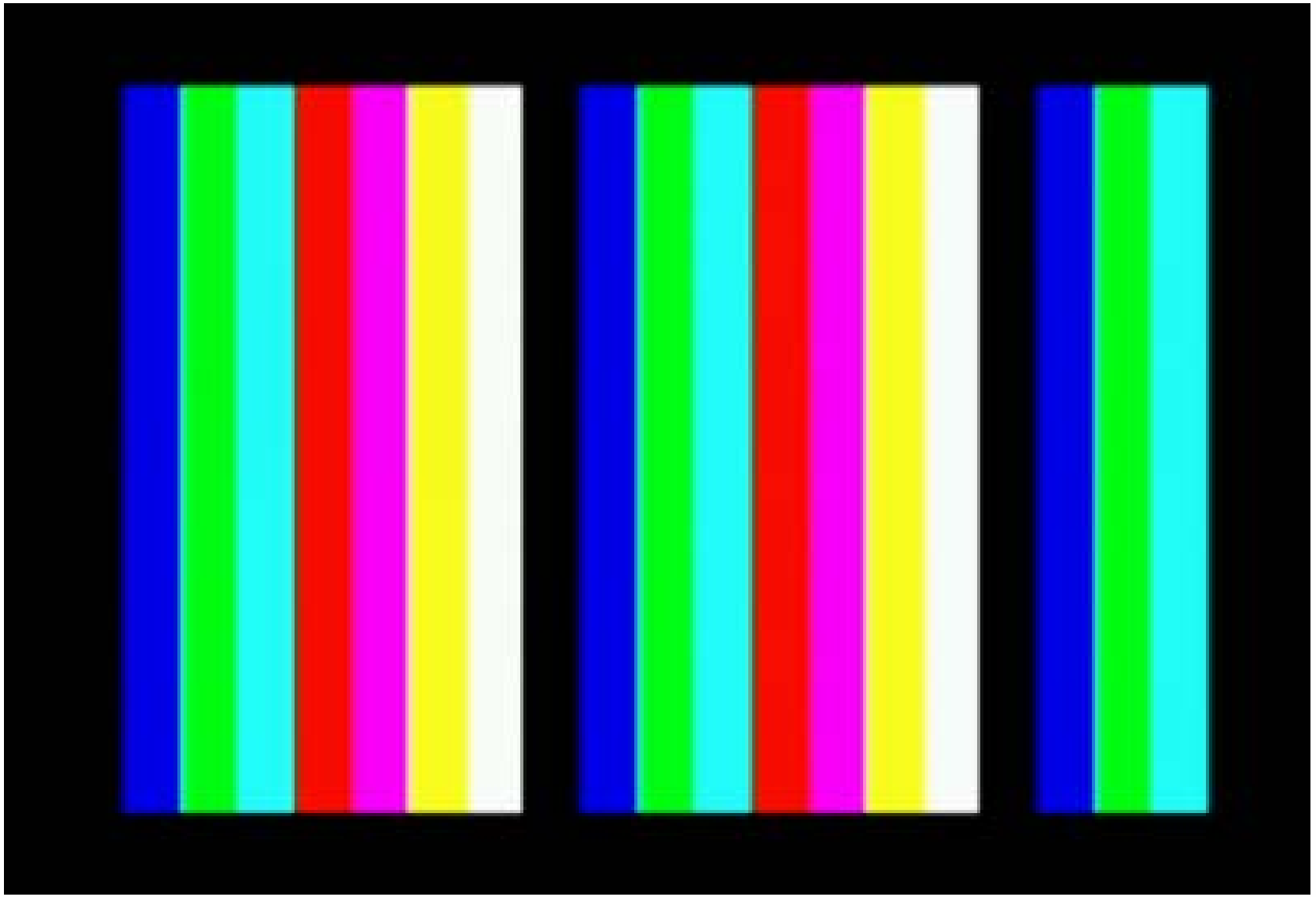
VGA Color and Sync



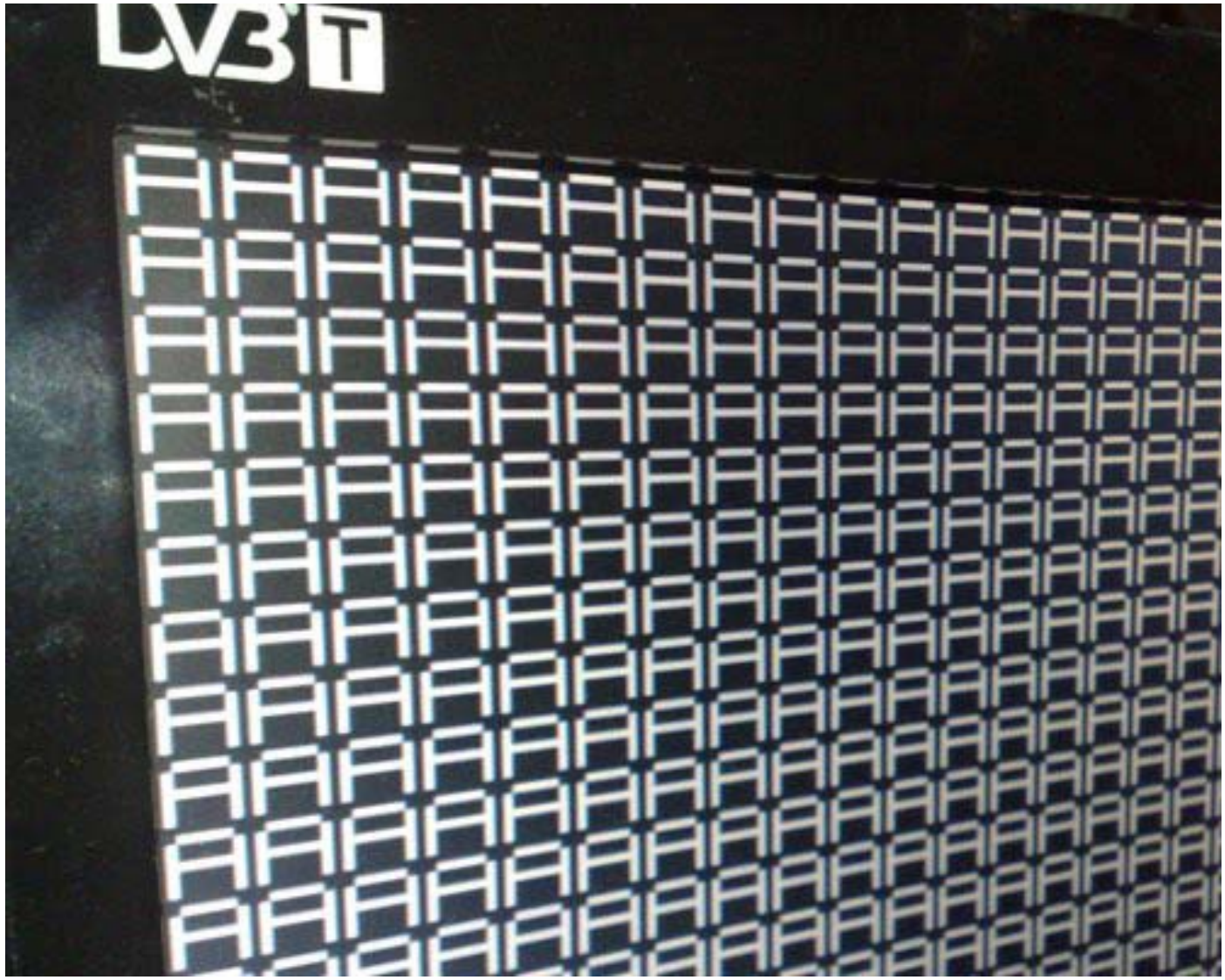
Schematic VGA controller



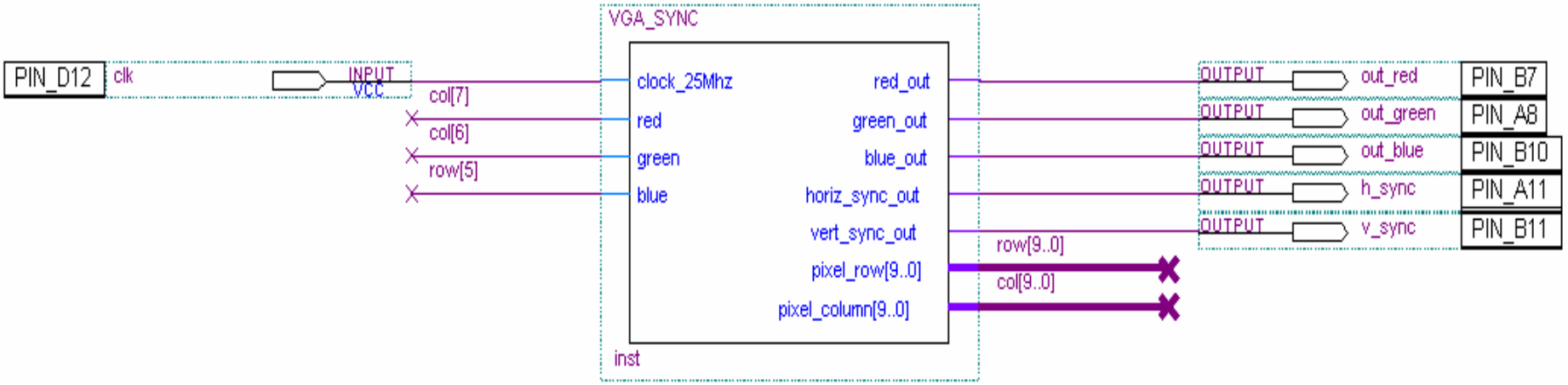
Color Bar



Character Bar



Schema VGA controller



VHDL per VGA controller - 1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY VGA_SYNC 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 :  OUT STD_LOGIC;
         pixel_row, pixel_column :        OUT STD_LOGIC_VECTOR( 9 DOWNTO 0 ));
END VGA_SYNC;

ARCHITECTURE a OF VGA_SYNC IS
   SIGNAL horiz_sync, vert_sync : STD_LOGIC;
   SIGNAL video_on, video_on_v, video_on_h : STD_LOGIC;
   SIGNAL h_count, v_count : STD_LOGIC_VECTOR( 9 DOWNTO 0 );
BEGIN
   -- video_on is High only when RGB data is displayed
   video_on <= video_on_H AND video_on_V;

-- Generate Horizontal and Vertical Timing Signals for Video Signal
-- H_count counts pixels (640 + extra time for sync signals)
-- Horiz_sync------------------------------_______________ --------
-- H_count 0         640                   659       755       799
```

VHDL - 2

```
PROCESS
BEGIN
WAIT UNTIL( clock_25Mhz'EVENT ) AND ( clock_25Mhz = '1' );
-- Generate Horizontal Sync Signal using H_count
    IF ( h_count = 799 ) THEN
        h_count <= "0000000000";
    ELSE
        h_count <= h_count + 1;
    END IF;

-- qui generiamo H_sync tra 659 e 755
    IF (h_count >= 659) AND (h_count <= 755) THEN
        horiz_sync <= '0';
    ELSE
        horiz_sync <= '1';
    END IF;
```

VHDL - 3

```
-- V_count counts rows of pixels (480 + extra time for sync signals)
-- Vert_sync ------------------------------_______________-------------
-- V_count   0              480             493         494             524
-- Generate Vertical Sync Signal using V_count
   IF ( v_count >= 524 ) AND ( h_count <= 699 ) THEN
       v_count <= "0000000000";
   ELSIF ( h_count = 699 ) THEN
       v_count <= v_count + 1 ;
   END IF;

   IF ( v_count >= 493 ) AND ( v_count <= 494 ) THEN
       vert_sync <= '0';
   ELSE
       vert_sync <= '1';
   END IF;

-- Generate Video on Screen Signals for Pixel Data
-- pixel colonne tra 0-639 (640 colonne)
   IF ( h_count <= 639 ) THEN
       video_on_h <= '1';
       pixel_column <= h_count;
   ELSE
       video_on_h <= '0';
   END IF;
```

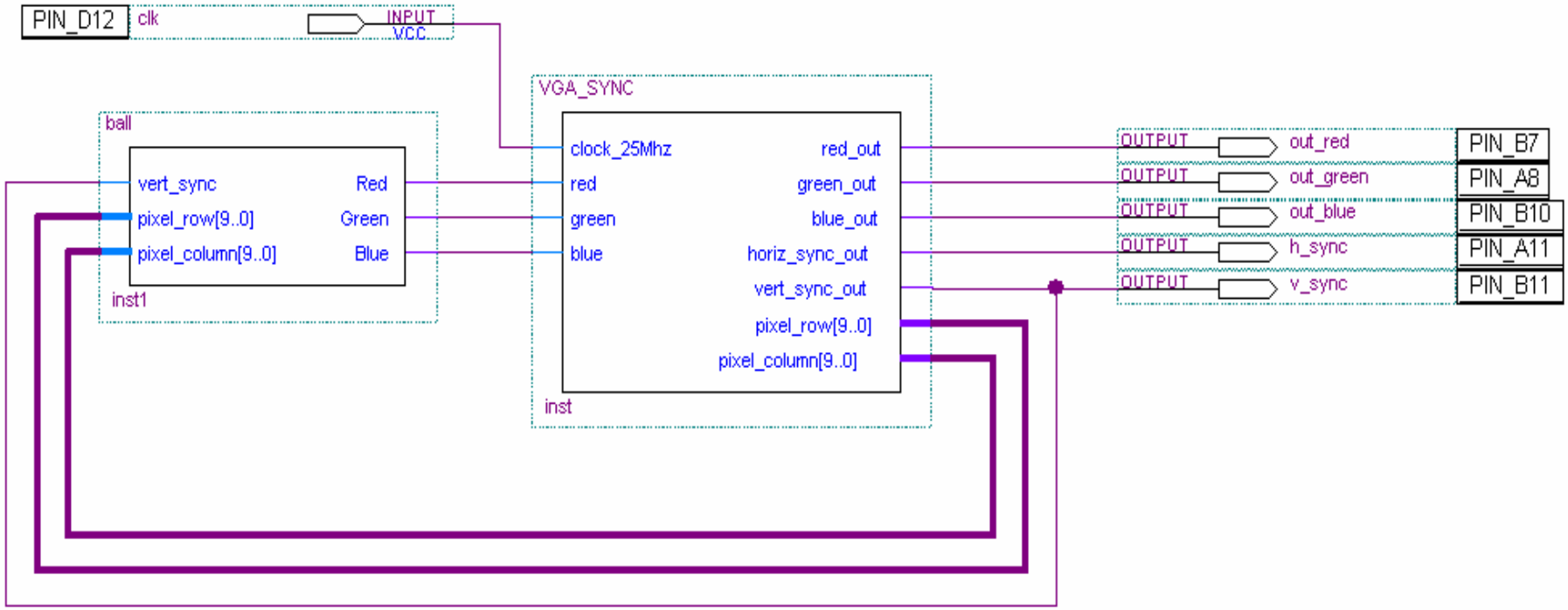
VHDL - 4

```
-- pixel righe tra 0-479 (640 righe)
    IF ( v_count <= 479 ) THEN
         video_on_v <= '1';
         pixel_row <= v_count;
    ELSE
         video_on_v <= '0';
    END IF;

-- Put all video signals through DFFs to eliminate
-- any delays that can cause a blurry image
-- Turn off RGB outputs when outside video display area
         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;
END PROCESS;
END a;
```

Schema VGA con “pong” controller



VHDL per oggetto “pong” - 1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ball IS
PORT( Red, Green, Blue : OUT STD_LOGIC;
      vert_sync :        IN  STD_LOGIC;
      pixel_row, pixel_column : IN STD_LOGIC_VECTOR( 9 DOWNTO 0 ));
END ball;

ARCHITECTURE behavior OF ball IS
    -- Video Display Signals
    SIGNAL reset, Ball_on, Direction : STD_LOGIC;
    SIGNAL Size : STD_LOGIC_VECTOR( 9 DOWNTO 0 );
    SIGNAL Ball_Y_motion : STD_LOGIC_VECTOR( 10 DOWNTO 0 );
    SIGNAL Ball_Y_pos, Ball_X_pos : STD_LOGIC_VECTOR( 10 DOWNTO 0 );

BEGIN -- Size of Ball
    Size <= CONV_STD_LOGIC_VECTOR (8,10 );
    Ball_X_pos <= CONV_STD_LOGIC_VECTOR(320,11 );
    Red <= '1';
    Green <= NOT Ball_on;
    Blue <= NOT Ball_on;
```

VHDL - 2

```
-- Ball center X address
-- Colors for pixel data on video signal
-- Turn off Green and Blue to make
-- color Red when displaying ball
RGB_Display: PROCESS ( Ball_X_pos, Ball_Y_pos, pixel_column, pixel_row, Size )
BEGIN
    -- Set Ball_on = '1' to display ball
    IF (Ball_X_pos <= pixel_column + Size ) AND
       (Ball_X_pos + Size >= pixel_column ) AND
       (Ball_Y_pos <= pixel_row + Size ) AND
       (Ball_Y_pos + Size >= pixel_row ) THEN
        Ball_on <= '1';
    ELSE
        Ball_on <= '0';
    END IF;
END PROCESS RGB_Display;
```

VHDL - 3

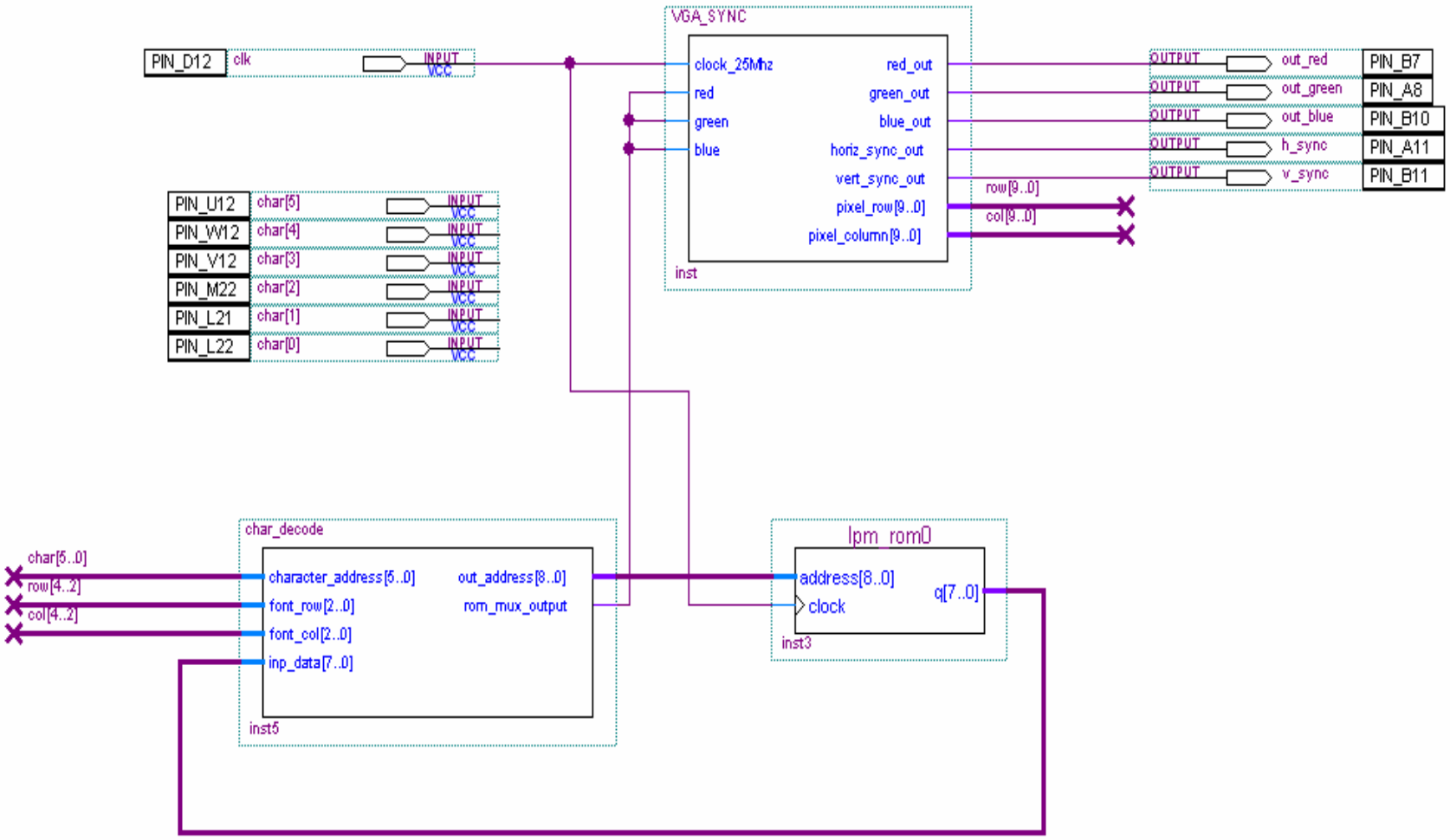
```
-- Move ball once every vertical sync
-- Bounce off top or bottom of screen
Move_Ball:PROCESS
BEGIN
   WAIT UNTIL Vert_sync'EVENT AND Vert_sync = '1';

   IF Ball_Y_pos >= 480 - Size THEN
       Ball_Y_motion <= CONV_STD_LOGIC_VECTOR(-4,11);
   ELSIF Ball_Y_pos <= Size THEN
       Ball_Y_motion <= CONV_STD_LOGIC_VECTOR(4,11);
   END IF;

   -- Compute next ball Y position
   Ball_Y_pos <= Ball_Y_pos + Ball_Y_motion;

END PROCESS Move_Ball;
END behavior;
```

VHDL generatore di caratteri



VHDL – rom con caratteri - 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY lpm_rom0 IS
    PORT
    (
        address : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
        clock   : IN STD_LOGIC ;
        q       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END lpm_rom0;
```

VHDL - 2

```
ARCHITECTURE SYN OF lpm_rom0 IS

    SIGNAL sub_wire0: STD_LOGIC_VECTOR (7 DOWNTO 0);
    COMPONENT altsyncram
    GENERIC (
        clock_enable_input_a          : STRING;
        clock_enable_output_a         : STRING;
        init_file                     : STRING;
        intended_device_family        : STRING;
        lpm_hint                      : STRING;
        lpm_type                      : STRING;
        numwords_a                    : NATURAL;
        operation_mode                : STRING;
        outdata_aclr_a                : STRING;
        outdata_reg_a                 : STRING;
        widthad_a                     : NATURAL;
        width_a                       : NATURAL;
        width_byteena_a               : NATURAL
    );
```

VHDL - 3

```
    PORT (  clock0   : IN STD_LOGIC ;
            address_a    : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
            q_a : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
    END COMPONENT;
BEGIN
    q    <= sub_wire0(7 DOWNTO 0);

    altsyncram_component : altsyncram
    GENERIC MAP (
        clock_enable_input_a => "BYPASS",
        clock_enable_output_a => "BYPASS",
        init_file => "tcgrom.mif",
        intended_device_family => "Cyclone II",
        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
        lpm_type => "altsyncram",
        numwords_a => 512,
        operation_mode => "ROM",
        outdata_aclr_a => "NONE",
        outdata_reg_a => "CLOCK0",
        widthad_a => 9,
        width_a => 8,
        width_byteena_a => 1 )
```

VHDL - 4

```
    PORT MAP (
        clock0 => clock,
        address_a => address,
        q_a => sub_wire0
    );
END SYN;
```

VHDL – decoder caratteri

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity char_decode is
   port (character_address:   in std_logic_vector(5 downto 0);
         font_row, font_col:  in std_logic_vector(2 downto 0);
         inp_data:            in std_logic_vector(7 downto 0);
         out_address:         out std_logic_vector(8 downto 0);
         rom_mux_output:      out std_logic
        );
end char_decode;

architecture a of char_decode is
begin
       out_address <= character_address & font_row;
       rom_mux_output <= inp_data (conv_integer(not font_col ( 2 downto 0)));

end a;
```