4OI4
Engineering Design

VGA Video Signal Generation

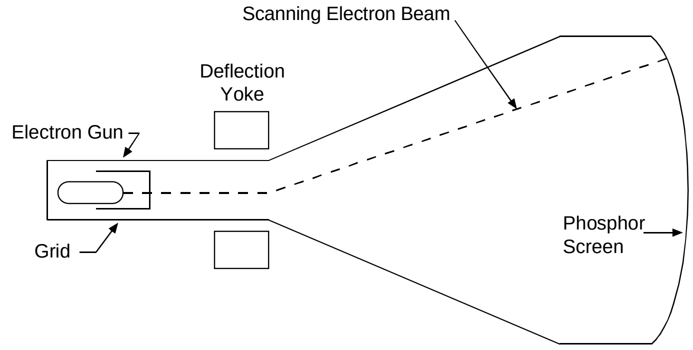
McMaster
University

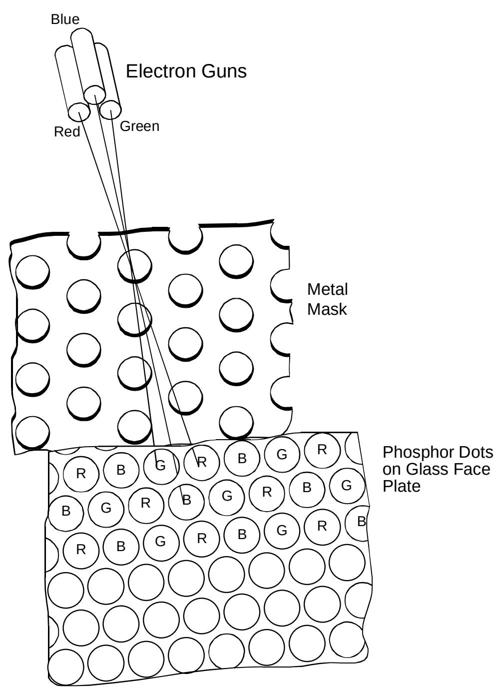
Video display

- VGA video signal: 5 active signals
  - Horizontal Sync. & Vertical Sync.: TTL logic levels
  - RGB: analog signals (0.7 to 1 volt peak to peak)
- Screen has 640x480 pixels
- Video signal redraws the entire screen 60 times per second

Video display

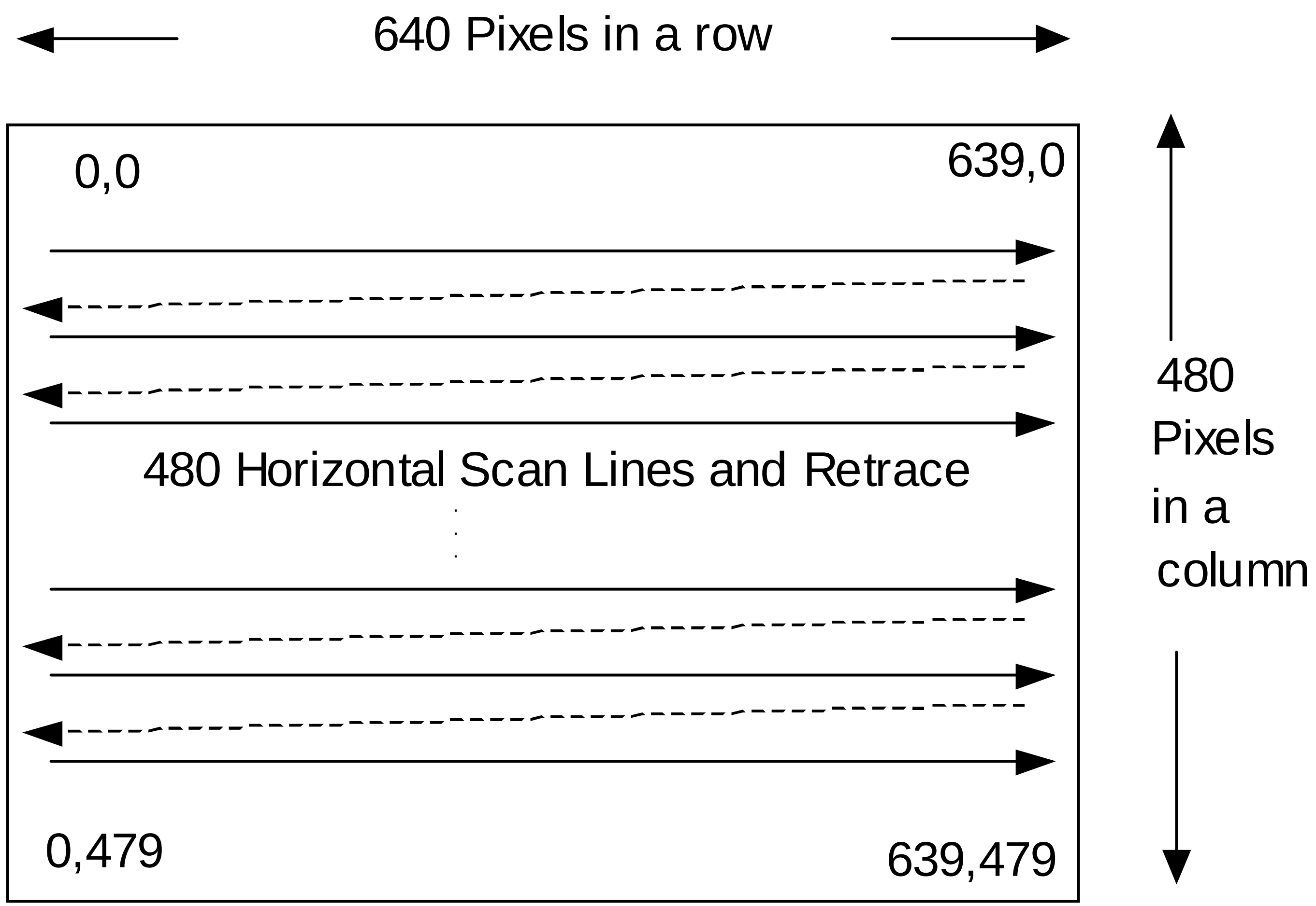
- Major component inside a VGA computer monitor is the color CRT.
- Electron beam is scanned over the screen in a sequence of horizontal lines to generate an image
- The deflection yoke deflects the electron beam to the appropriate position on the face of CRT
- Light is generated when the beam is turned on by a video signal and it strikes a color phosphor on the CRT.
- Face of CRT contains three different phosphors one type for each primary color (red, green, blue)





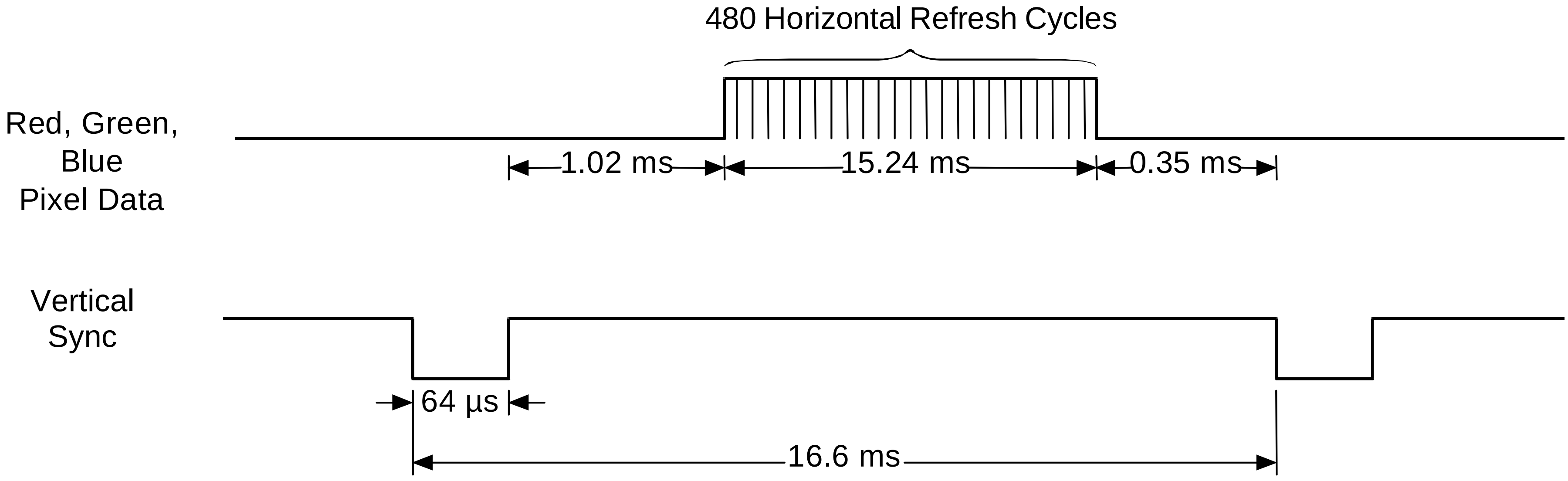
Color CRT and Phosphor Dots on Face of Display.

McMaster University



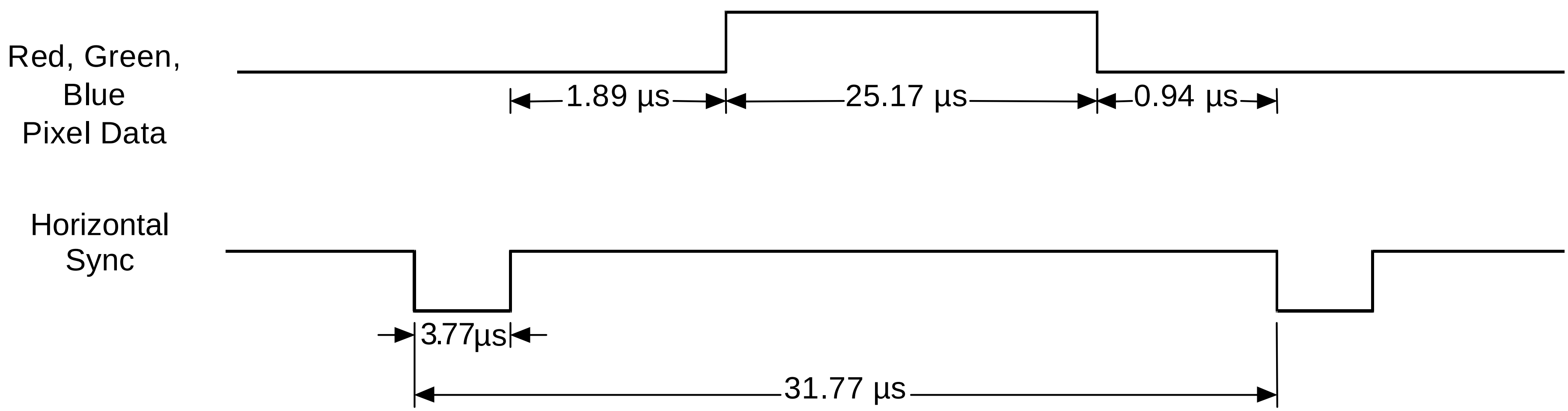
VGA Image - 640 by 480 Pixel Layout.

McMaster University



Vertical Sync Signal Timing.

McMaster University



Horizontal Sync Signal Timing.

McMaster University

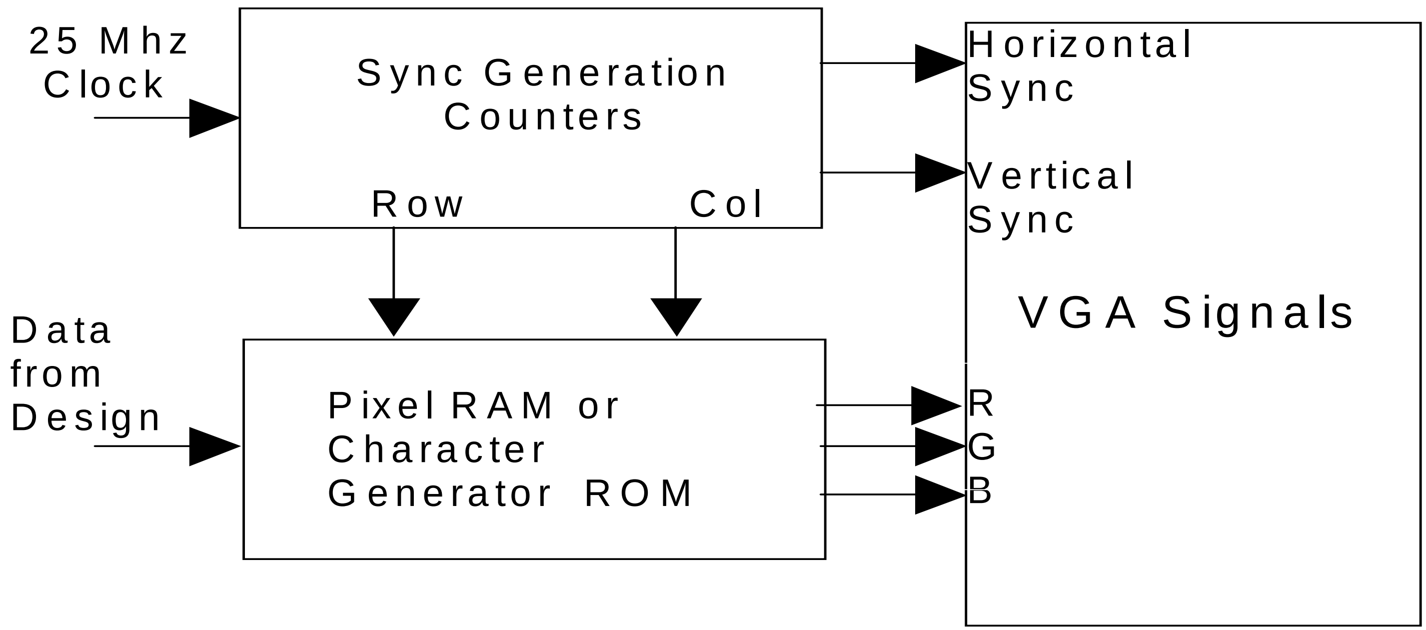
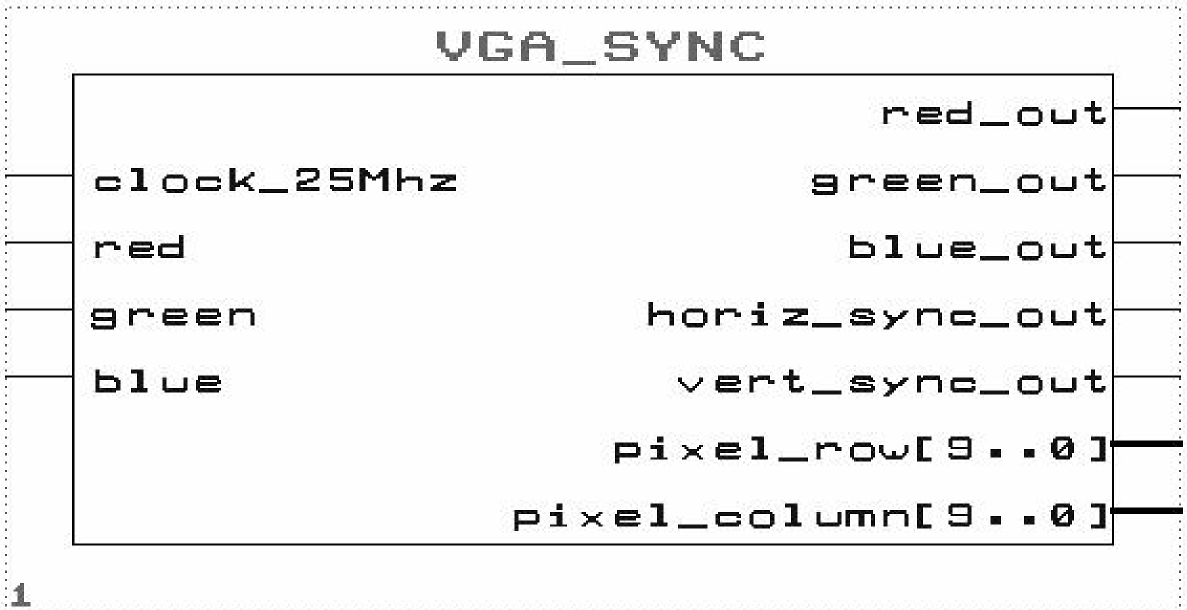


Figure 9.5 CLPD based generation of VGA Video Signals.



UP1core VGA_SYNC

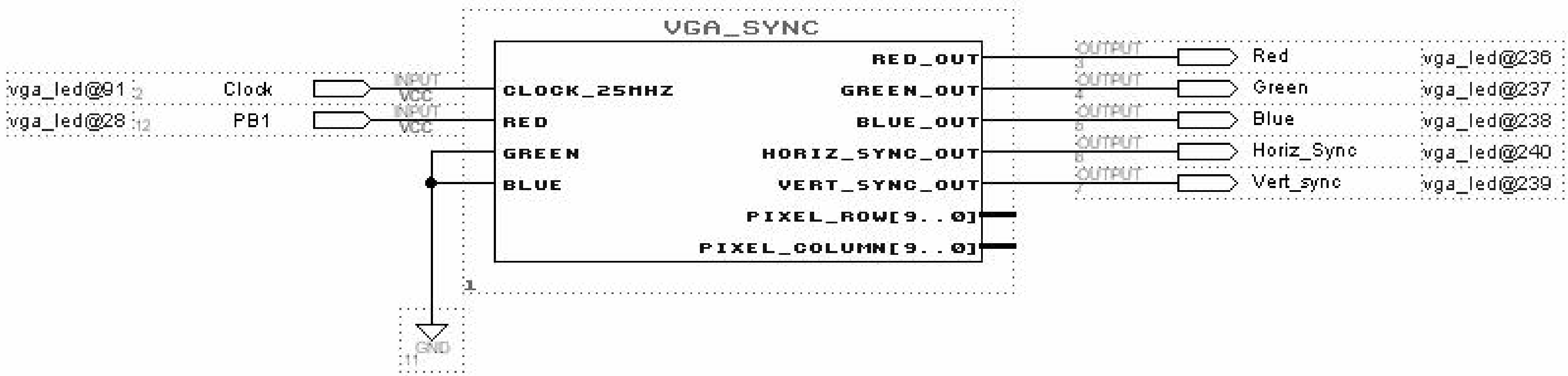
```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
USE  IEEE.STD_LOGIC_ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;

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;


PROCESS
     BEGIN
     WAIT UNTIL( clock_25Mhz'EVENT ) AND ( clock_25Mhz = '1' );
```

```
                  --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
                  --
IF ( h_count = 799 ) THEN
     h_count <= "0000000000";
ELSE
     h_count <= h_count + 1;
END IF;

                  --Generate Horizontal Sync Signal using H_count
IF ( h_count <= 755 ) AND  (h_count => 659 ) THEN
     horiz_sync <= '0';
ELSE
     horiz_sync <= '1';
END IF;
```

McMaster University

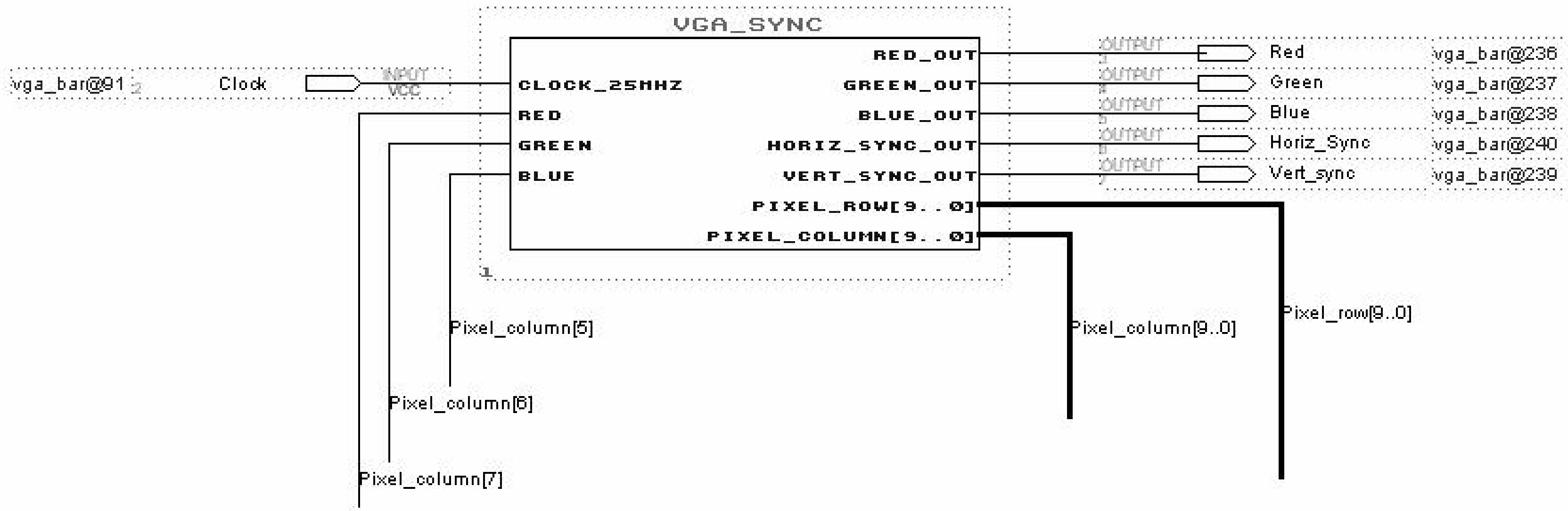
```
        --V_count counts rows of pixels (480 + extra time for sync signals)
        --
        --  Vert_sync     --------------------------------------_______------------
        --  V_count       0                   480                 493-494      524
        --
IF ( v_count >= 524 ) AND ( h_count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

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

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;
```



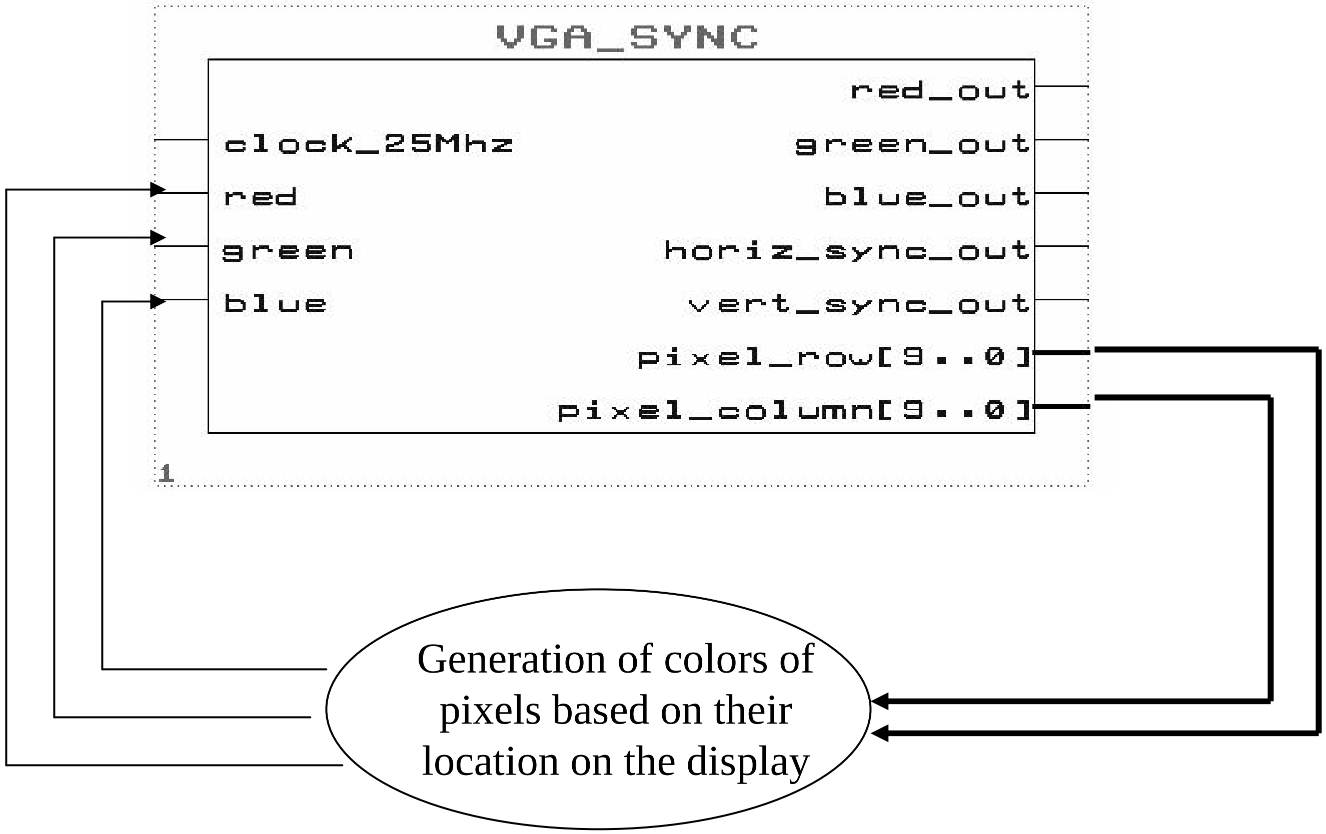
Video LED Design Example

McMaster University



Video Color Bar Design Example

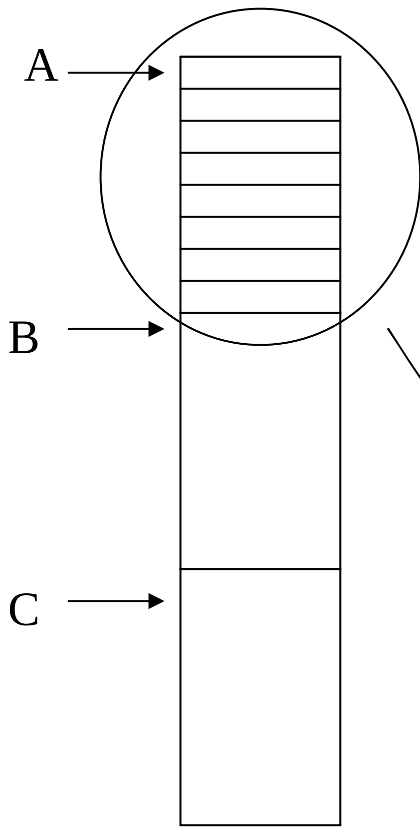
VGA display


VGA_SYNC
red_out
clock_25Mhz
green_out
red
blue_out
green
horiz_sync_out
blue
vert_sync_out
pixel_row[9..0]
pixel_column[9..0]
1
Generation of colors of pixels based on their location on the display

Character Display

- Displaying textual data: a pixel pattern or font is needed to display each different character
- Character font can be stored in a ROM implemented inside the FLEX.
- A memory initialization file (.mif) is used to initialize the ROM
- CHAR_ROM function in UP1core functions is a character ROM.
- Characters are stored in consecutive memory cells.
- Each character is stored in eight memory cells (each memory cell 8 bits)
- Each character consists of an 8x8 dot map

Character Display



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

Character Display

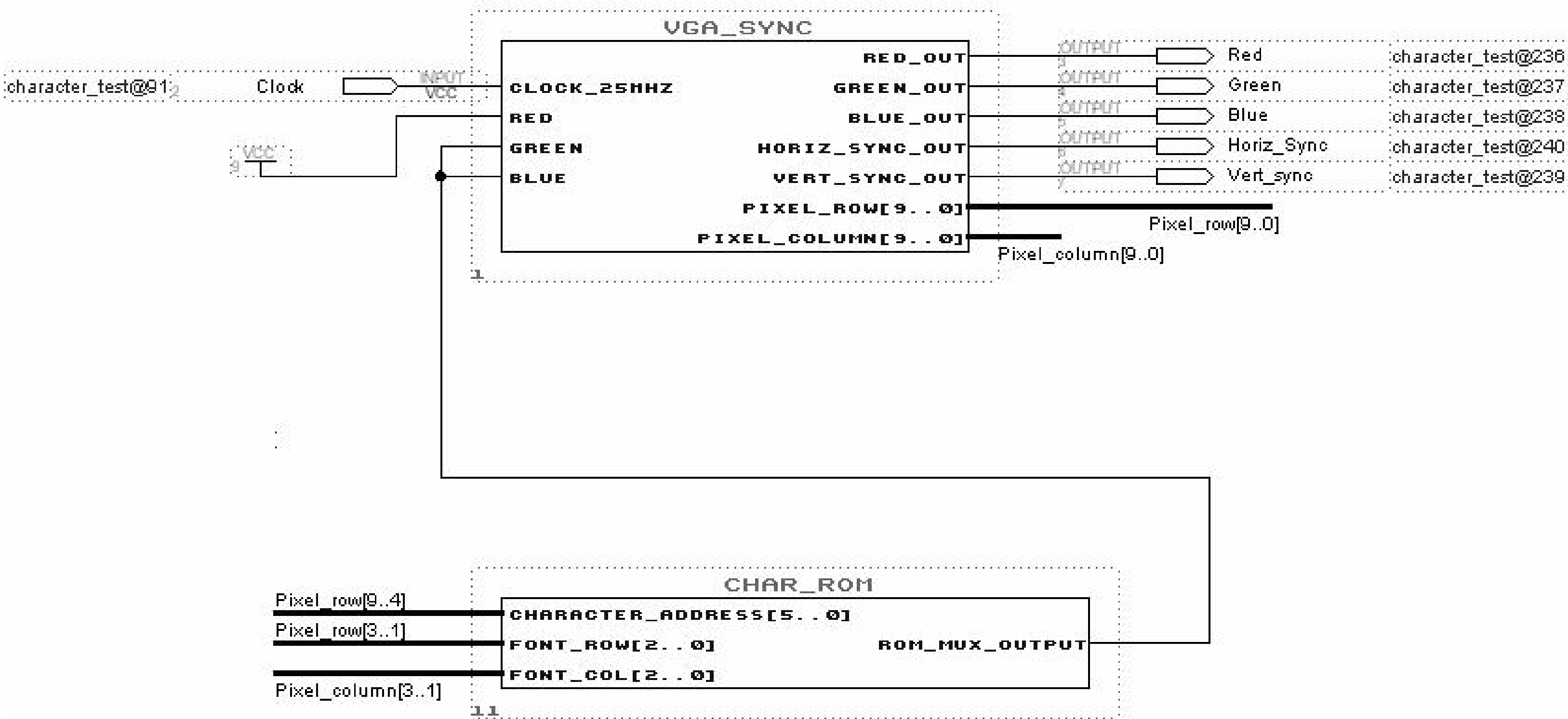
- The function char_rom has three inputs:
  - character_address: the starting address of memory location containing the character we want to be displayed
  - font_row and font_col: inputs that determine which bit of the memory partition containing the character should be displayed at a particular time (which dot of the character font should be displayed)
- If each dot in a character font is mapped to pixel on the display, each character requires 8x8 pixels on the display.
- If each dot in a character font is mapped to a 2x2 pixel area on the display, each character requires 16x16 pixels on the display.

UP1core CHAR_ROM

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*

```
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(     character_address     : IN      STD_LOGIC_VECTOR( 5 DOWNTO 0 );
             font_row, font_col    : IN      STD_LOGIC_VECTOR( 2 DOWNTO 0 );
             rom_mux_output        : OUT     STD_LOGIC);
END Char_ROM;

ARCHITECTURE a OF Char_ROM IS
   SIGNAL   rom_data               : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
   SIGNAL   rom_address            : STD_LOGIC_VECTOR( 8 DOWNTO 0 );
BEGIN
                    -- Small 8 by 8 Character Generator ROM for Video Display
                    -- Each character is 8 8-bit words of pixel data
 char_gen_rom: lpm_rom
    GENERIC MAP (
         lpm_widthad            => 9,
         lpm_numwords           => "512",
         lpm_outdata            => "UNREGISTERED",
         lpm_address_control    => "UNREGISTERED",
                                -- Reads in mif file for character generator font data
         lpm_file               => "tcgrom.mif",
         lpm_width              => 8)
    PORT MAP ( address => rom_address, q = > rom_data);
        rom_address <= character_address & font_row;
                    -- Mux to pick off correct rom data bit from 8-bit word
                    -- for on screen character generation
        rom_mux_output <= rom_data (
              (CONV_INTEGER( NOT font_col( 2 DOWNTO 0 ))) );
END a;
```

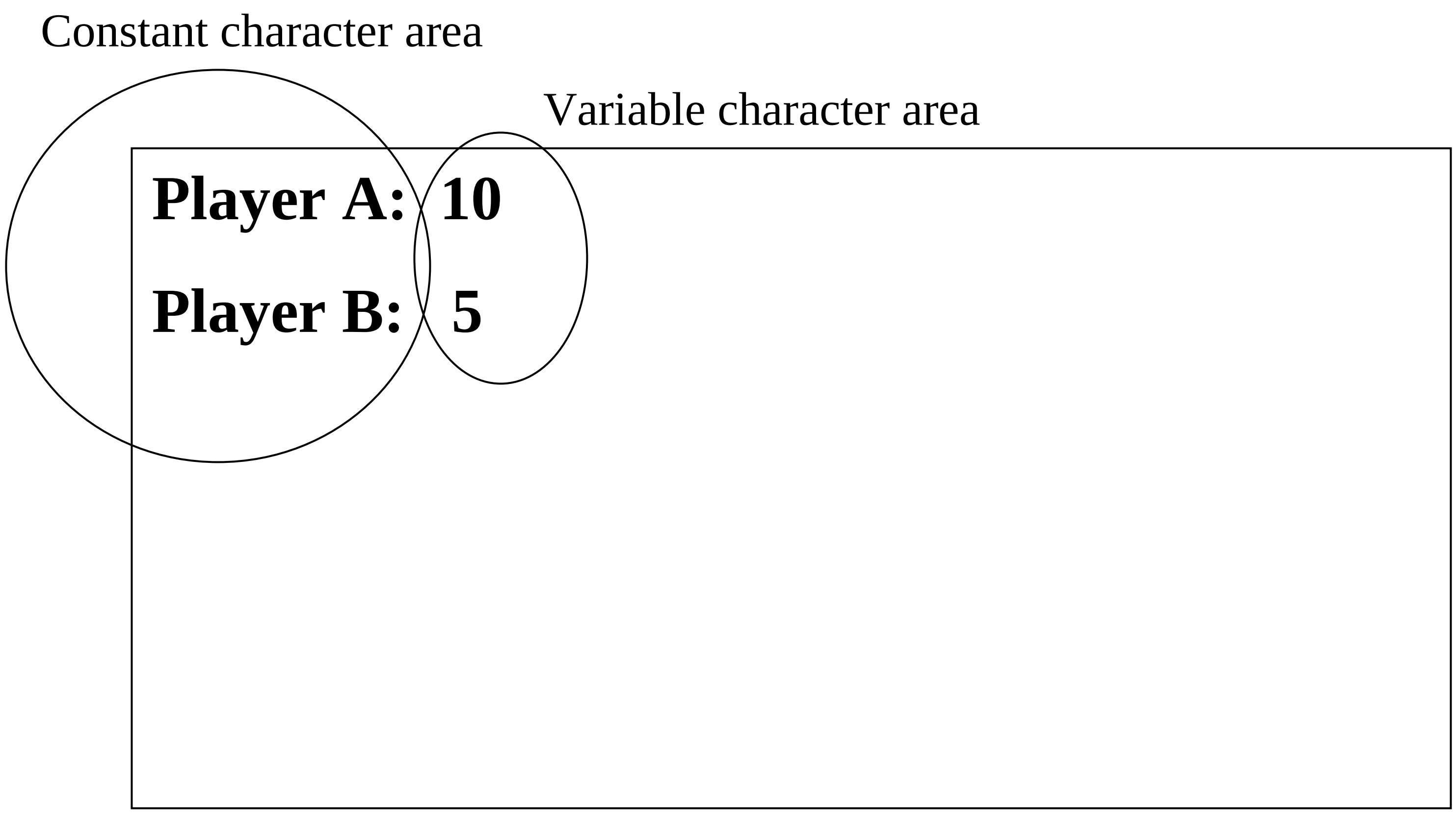


Character Test Design Example

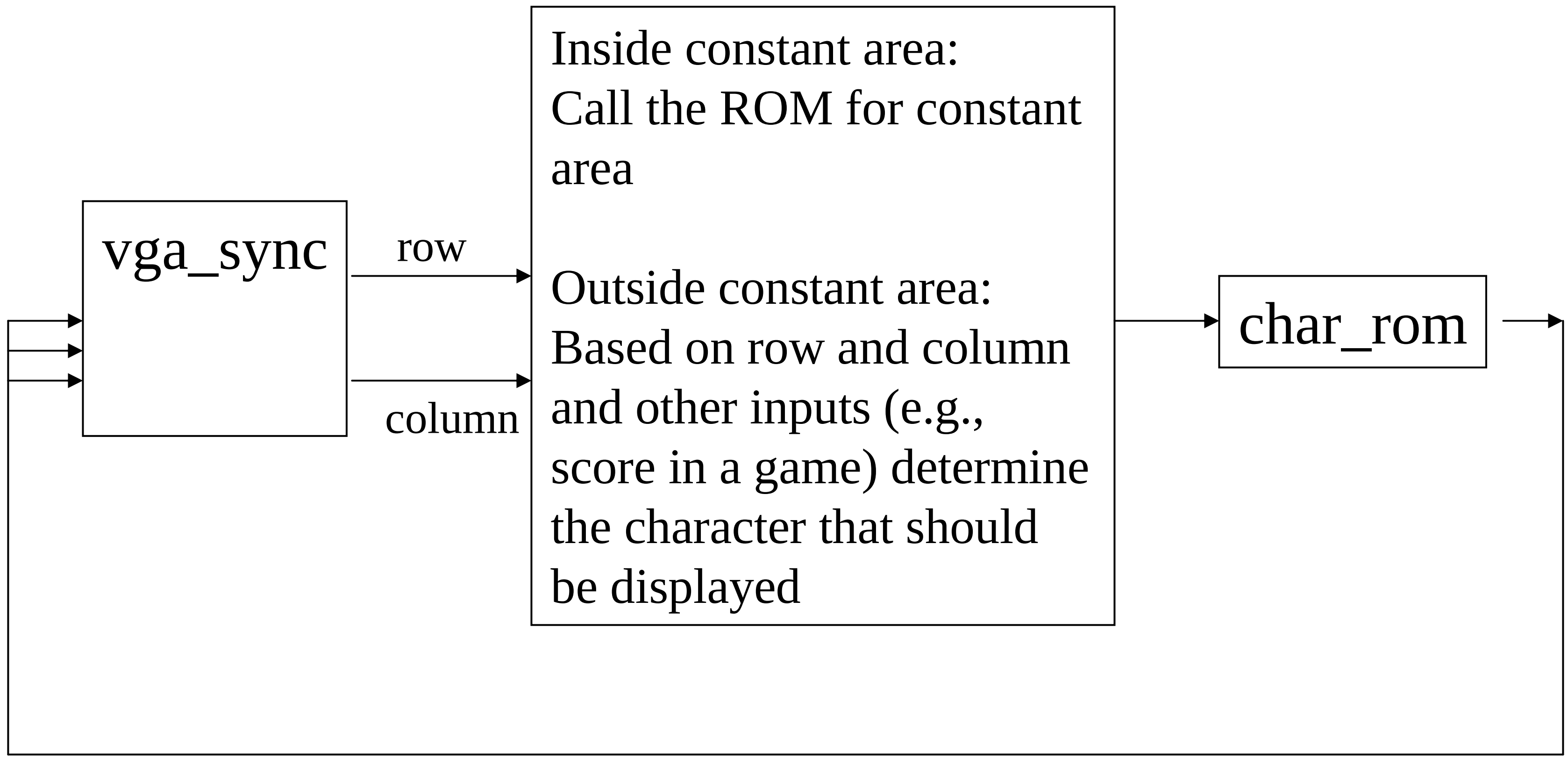
Character Display

- General guidelines for character display in complex designs:
  - Constant character areas (characters which do not change) can be stored in a small ROM (using lpm_rom megafunction)
  - At each clock cycle, a process containing a series of CASE statements is used to select the character to be displayed
  - To do this CASE statements check the row and column counter outputs from the vga_sync to determine the character that is currently being displayed
  - The CASE statements then output the character address for the desired character to the ROM

Character Display



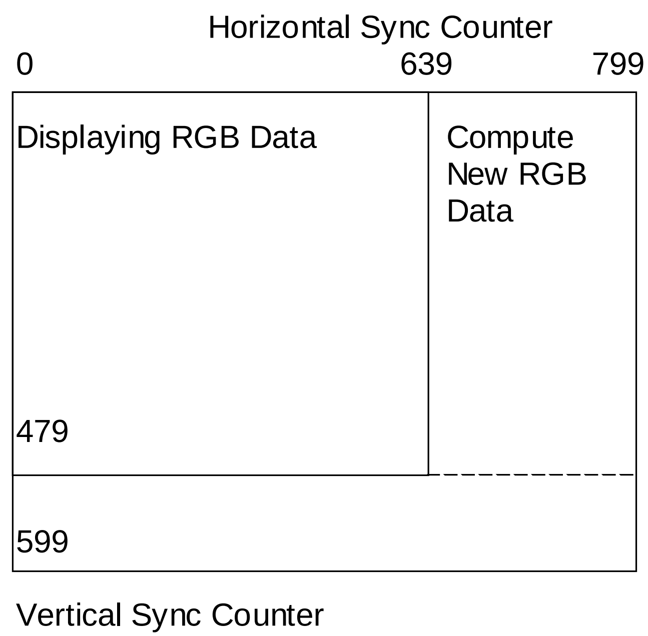
Character Display



Graphic display

- One approach to graphic display is to use a RAM is to keep the color information of each pixel (pixel RAM)
- The output of this RAM is fed to RGB signal of the vga_sync
- To avoid flicker and memory access conflicts, the pixel RAM should be updated during the time RGB signal is not being displayed.
- This is the time the beam is returning to the beginning of each line or the beginning of the display (fly-back time)

Graphic display



Display and Compute clock cycles available in a single Video Frame.

Graphic display

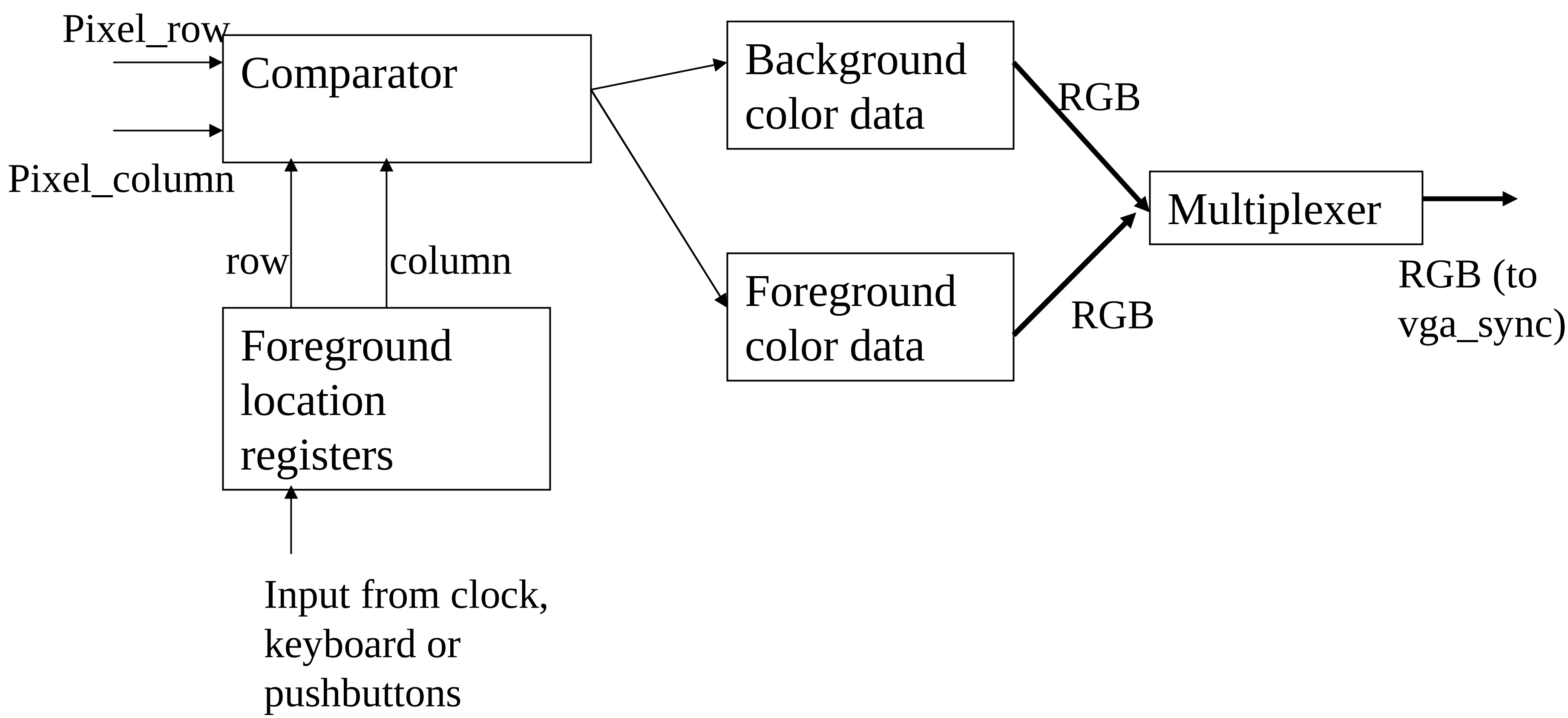
- General guidelines for simple graphic display:

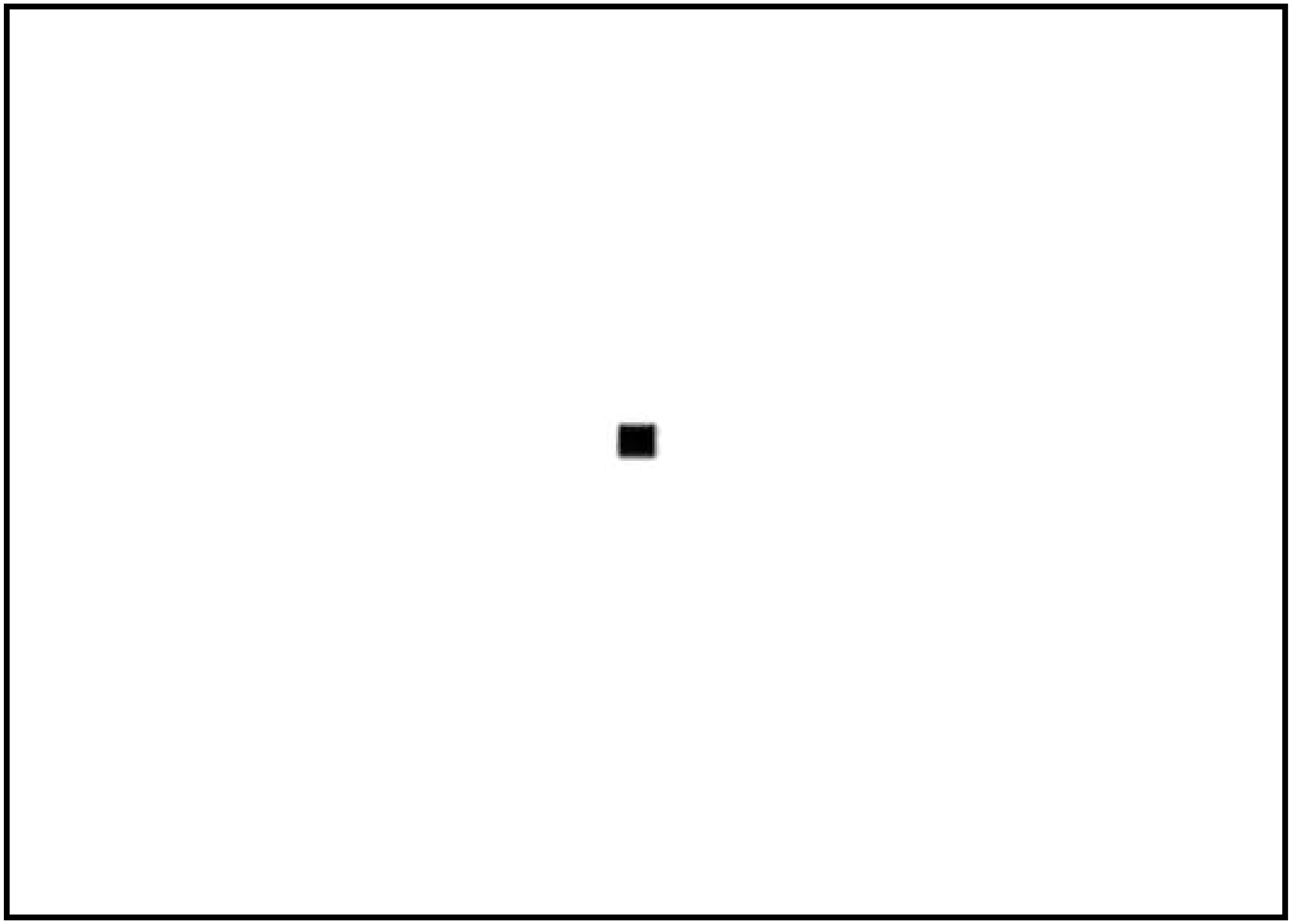
1. The background image can be the default color value and not stored in the video RAM
2. Comparators check the row and column counts and detect when another image (foreground) other that the background should be displayed
3. When the comparator signals that the foreground should be displayed, the foreground’s color data instead of background is switched to the RGB

Graphic display

- The foreground can be moved using the following technique:
  - The current location (row and column) of the foreground is stored in registers
  - These registers are used as the comparator inputs
  - The registers are incremented or decremented based on some other inputs (time, keyboard, pushbuttons, …)

Graphic display





Bouncing Ball Video Output.

McMaster University

```
ENTITY ball IS
    PORT(
        SIGNAL Red, Green, Blue               : OUT STD_LOGIC;
        SIGNAL vert_sync_out                  : IN STD_LOGIC;
        SIGNAL 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 center  X address
    Ball_X_pos <= CONV_STD_LOGIC_VECTOR( 320,11 );
                                            -- Colors for pixel data on video signal
    Red         <=  '1';                    -- Turn off Green and Blue to make
                                            -- color Red when displaying ball
    Green       <= NOT Ball_on;
    Blue        <= NOT Ball_on;
```

McMaster University

```
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;

Move_Ball:
    PROCESS
    BEGIN
                            -- Move ball once every vertical sync
    WAIT UNTIL Vert_sync'EVENT AND Vert_sync = '1';
                            -- Bounce off top or bottom of screen
        IF Ball_Y_pos >= 480 - Size THEN
             Ball_Y_motion <= - CONV_STD_LOGIC_VECTOR(2,11);
        ELSIF  Ball_Y_pos <= Size   THEN
            Ball_Y_motion <= CONV_STD_LOGIC_VECTOR(2,11);
        END IF;
                            -- Compute next ball Y position
        Ball_Y_pos <= Ball_Y_pos + Ball_Y_motion;
    END PROCESS Move_Ball;
END behavior;
```