003- net s<2>  loc=p19;  
004- net s<3>  loc=p17;  
005- net s<4>  loc=p18;  
006- net s<5>  loc=p14;  
007- net s<6>  loc=p15;  
008- net rst  loc=p45;  // microcontroller reset  
009- net oeb  loc=p62;  // RAM output enable  
010- net dipsw<1>  loc=p6;  // DIP switch inputs  
011- net dipsw<2>  loc=p7;  
012- net dipsw<3>  loc=p11;  
013- net dipsw<4>  loc=p5;  
014- net dipsw<5>  loc=p72;  
015- net dipsw<6>  loc=p71;  
016- net dipsw<7>  loc=p66;  
017- net dipsw<8>  loc=p70;  
018- net spareb  loc=p67;  // SPARE pushbutton input  
019- net resetb  loc=p10;  // RESET pushbutton input  
020- net lsb<0>  loc=p11;  // XStend left LED digit segments  
021- net lsb<1>  loc=p2;  
022- net lsb<2>  loc=p3;  
023- net lsb<3>  loc=p75;  
024- net lsb<4>  loc=p79;  
025- net lsb<5>  loc=p82;  
026- net lsb<6>  loc=p83;  
027- net lsb<7>  loc=p84;  
028- net rsb<0>  loc=p58;  // XStend right LED digit segments  
029- net rsb<1>  loc=p56;  
030- net rsb<2>  loc=p54;  
031- net rsb<3>  loc=p55;  
032- net rsb<4>  loc=p53;  
033- net rsb<5>  loc=p57;  
034- net rsb<6>  loc=p61;  
035- net rsb<7>  loc=p34;  
036- net db<1>  loc=p44;  // XStend bargraph LED segments  
037- net db<2>  loc=p43;  
038- net db<3>  loc=p41;  
039- net db<4>  loc=p40;  
040- net db<5>  loc=p39;  
041- net db<6>  loc=p37;  
042- net db<7>  loc=p36;  
043- net db<8>  loc=p35;  

Displaying Graphics from RAM Through the VGA Interface

This section discusses the timing for the signals that drive a VGA monitor and describes a VHDL module that will let you drive a monitor with a picture stored in RAM.

VGA Color Signals

There are three signals -- red, green, and blue -- that send color information to a VGA monitor. These three signals each drive an electron gun that emits electrons which paint one primary color at a point on the monitor screen. Analog levels between 0 (completely dark) and 0.7 V (maximum brightness) on these control lines tell the monitor what
intensities of these three primary colors to combine to make the color of a dot (or pixel) on
the monitor's screen.

Each analog color input can be set to one of four levels by two digital outputs using a
simple two-bit digital-to-analog converter (see Figure 7). The four possible levels on each
analog input are combined by the monitor to create a pixel with one of $4 \times 4 \times 4 = 64$
different colors. So the six digital control lines let us select from a palette of 64 colors.

![Figure 7: Digital-to-analog interface to a VGA monitor.](image)

**VGA Signal Timing**

A single dot of color on a video monitor doesn't impart much information. A horizontal line
of pixels carries a bit more information. But a frame composed of multiple lines can
present an image on the monitor screen. A frame of VGA video typically has 480 lines and
each line usually contains 640 pixels. In order to paint a frame, there are deflection circuits
in the monitor that move the electrons emitted from the guns both left-to-right and top-to-
bottom across the screen. These deflection circuits require two synchronization signals in
order to start and stop the deflection circuits at the right times so that a line of pixels is
painted across the monitor and the lines stack up from the top to the bottom to form an
image. The timing for the VGA synchronization signals is shown in Figure 8.

Negative pulses on the *horizontal sync* signal mark the start and end of a line and ensure
that the monitor displays the pixels between the left and right edges of the visible screen
area. The actual pixels are sent to the monitor within a 25.17 \( \mu \text{s} \) window. The horizontal
sync signal drops low a minimum of 0.94 \mu s after the last pixel and stays low for 3.77 \mu s. A new line of pixels can begin a minimum of 1.89 \mu s after the horizontal sync pulse ends. So a single line occupies 25.17 \mu s of a 31.77 \mu s interval. The other 6.6 \mu s of each line is the horizontal blanking interval during which the screen is dark.

In an analogous fashion, negative pulses on a vertical sync signal mark the start and end of a frame made up of video lines and ensure that the monitor displays the lines between the top and bottom edges of the visible monitor screen. The lines are sent to the monitor within a 15.25 ms window. The vertical sync signal drops low a minimum of 0.45 ms after the last line and stays low for 64 \mu s. The first line of the next frame can begin a minimum of 1.02 ms after the vertical sync pulse ends. So a single frame occupies 15.25 ms of a 16.784 ms interval. The other 1.534 ms of the frame interval is the vertical blanking interval during which the screen is dark.

![Video Line Diagram]

**VGA Signal Generator Algorithm**

Now we have to figure out a process that will send pixels to the monitor with the correct timing and framing. We can store a picture in the RAM of the XS Board. Then we can
retrieve the data from the RAM, format it into lines of pixels, and send the lines to the monitor with the appropriate pulses on the horizontal and vertical sync pulses.

The pseudocode for a single frame of this process is shown in Listing 16. The pseudocode has two outer loops: one, which displays the \( L \) lines of visible pixels, and another, which inserts the \( V \), blank lines and the vertical sync pulse. Within the first loop, there are two more loops: one, which sends the \( P \) pixels of each video line to the monitor, and another, which inserts the \( H \), blank pixels and the horizontal sync pulse.

Within the pixel display loop, there are statements to get the next byte from the RAM. Each byte contains four two-bit pixels. A small loop iteratively extracts each pixel to be displayed from the lower two bits of the byte. Then the byte is shifted by two bits so the next pixel will be in the right position during the next iteration of the loop. Since it has only two bits, each pixel can store one of four colors. The mapping from the two-bit pixel value to the actual values required by the monitor electronics is done by the \texttt{COLOR\_MAP()} routine.

- Listing16: VGA signal generation pseudocode.

```c
/* send \( L \) lines of video to the monitor */
for line_cnt=1 to \( L \)
  /* send \( P \) pixels for each line */
  for pixel_cnt=1 to \( P \)
    /* get pixel data from the RAM */
    data = RAM(address)
    address = address + 1
    /* RAM data byte contains 4 pixels */
    for \( d=1 \) to \( 4 \)
      /* mask off pixel in the lower two bits */
      pixel = data & 00000011
      /* shift next pixel into lower two bits */
      data = data>>2
      /* get the color for the two-bit pixel */
      color = COLOR\_MAP(pixel)
      send color to monitor
      \( d = d + 1 \)
    /* increment by four pixels */
    pixel_cnt = pixel_cnt + 4
  /* blank the monitor for \( H \) pixels */
  for horiz\_blank\_cnt=1 to \( H \)
    color = BLANK
    send color to monitor
    /* pulse the horizontal sync at the right time */
    if horiz\_blank\_cnt>HB0 and horiz\_blank\_cnt<HB1
      hsync = 0
    else
      hsync = 1
    horiz\_blank\_cnt = horiz\_blank\_cnt + 1
  line_cnt = line_cnt + 1
  /* blank the monitor for \( V \) lines and insert vertical sync */
  for vert\_blank\_cnt=1 to \( V \)
    color = BLANK
    send color to monitor
```

26
/* pulse the vertical sync at the right time */
if vert_blank_cnt>VBO and vert_blank_cnt<VBl
    vsync = 0
else
    vsync = 1
    vert_blank_cnt = vert_blank_cnt + 1
/* go back to start of picture in RAM */
address = 0

Figure 9 shows how to pipeline certain operations to account for delays in accessing data from the RAM. The pipeline has three stages:

Stage 1: The circuit uses the horizontal and vertical counters to compute the address where the next pixel is found in RAM. The counters are also used to determine the firing of the sync pulses and whether the video should be blanked. The pixel data from the RAM, blanking signal, and sync pulses are latched at the end of this stage so they can be used in the next stage.

Stage 2: The circuit uses the pixel data and the blanking signal to determine the binary color outputs. These outputs are latched at the end of this stage.

Stage 3: The binary color outputs are applied to the DAC, which sets the intensity levels for the monitor’s color guns. The actual pixel is painted on the screen during this stage.

VGA Signal Generator in VHDL

The pseudocode and pipeline timing in the last section will help us to understand the VHDL code for a VGA signal generator shown in Listing 17. The inputs and outputs of the circuit as defined in the entity declaration are as follows:
clk: The input for the 12 MHz clock of the XS Board is declared here. This clock sets the maximum rate at which pixels can be sent to the monitor. The time interval within each line for transmitting viewable pixels is 25.17 μs, so this VGA generator circuit can only put a maximum of 25.17 ms × 12 MHz = 302 pixels on each line. For purposes of storing images in the RAM, it is convenient to reduce this to 256 pixels per line and blank the remaining 46 pixels. Half of these blank pixels are placed before the 256 viewable pixels and half are placed after them on a line. This centers the viewable pixels between the left and right edges of the monitor screen.

reset: This line declares an input, which will reset all the other circuitry to a known state.

hsynca, vsynca: The outputs for the horizontal and vertical sync pulses are declared. The hsynca output is declared as a buffer because it will also be referenced within the architecture section as a clock for the vertical line counter.

rgb: The outputs that control the red, green, and blue color guns of the monitor are declared here. Each gun is controlled by two bits, so there are four possible intensities for each color. Thus, this circuit can produce $4 \times 4 = 64$ different colors.

address, data: These lines declare the outputs for driving the address lines of the RAM and the inputs for receiving the data from the RAM.

cbe, oeb, web: These are the declarations for the outputs which drive the chip-select, output-enable, and write-enable control lines of the RAM.

The preamble of the architecture section declares the following resources:

hcna, vcnt: The counters that store the current horizontal position within a line of pixels and the vertical position of the line on the screen are declared on these lines. We will call these the horizontal or pixel counter, and the vertical or line counter, respectively. The line period is 31.77 μs that is 381 clock cycles, so the pixel counter needs at least nine bits of resolution. Each frame is composed of 528 video lines (only 480 are visible, the other 48 are blanked), so a ten bit counter is needed for the line counter.

pixr: This is the declaration for the eight-bit register that stores the four pixels received from the RAM.

blank, pblank: This line declares the video blanking signal and its registered counterpart that is used in the next pipeline stage.

Within the main body of the architecture section, these following processes are executed:

inc_horiz_pixel_counter: This process describes the operation of the horizontal pixel counter. The counter is asynchronously set to zero when the reset input is high. The counter increments on the rising edge of each pixel clock. The range for the horizontal pixel counter is \([0, 380]\). When the counter reaches 380, it rolls over to zero on the next cycle. Thus, the counter has a period of 381 pixel clocks. With a pixel clock of 12 MHz, this translates to a period of 31.75 μs.

inc_vert_line_counter: This process describes the operation of the vertical line counter. The counter is asynchronously set to zero when the reset input is high. The counter increments on the rising edge of the horizontal sync pulse after a line of pixels is completed. The range for the horizontal pixel counter is \([0, 527]\). When the counter
reaches 527, it rolls over to zero on the next cycle. Thus, the counter has a period of 528 lines. Since the duration of a line of pixels is 31.75 \( \mu \)s, this makes the frame interval equal to 16.76 ms.

**generate_horz_sync:** This process describes the operation of the horizontal sync pulse generator. The horizontal sync is set to its inactive high level when the reset is activated. During normal operations, the horizontal sync output is updated on every pixel clock. The sync signal goes low on the cycle after the pixel counter reaches 291 and continues until the cycle after the counter reaches 337. This gives a low horizontal sync pulse of \((337-291)=46\) pixel clocks. With a pixel clock of 12 MHz, this translates to a low-going horizontal sync pulse of 3.83 \( \mu \)s. The sync pulse starts 292 clocks after the line of pixels begin, which translates to 24.33 \( \mu \)s. This is less than the 26.11 \( \mu \)s we stated before. The difference of 1.78 \( \mu \)s translates to 21 pixel clocks. This time interval corresponds to the 23 blank pixels that are placed before the 256 viewable pixels (minus two clock cycles for pipelining delays).

**generate_vert_sync:** This process describes the operation of the vertical sync pulse generator. The vertical sync is set to its inactive high level when the reset is activated. During normal operations, the vertical sync output is updated after every line of pixels is completed. The sync signal goes low on the cycle after the line counter reaches 493 and continues until the cycle after the counter reaches 495. This gives a low vertical sync pulse of \((495-493)=2\) lines. With a line interval of 31.75 \( \mu \)s, this translates to a low-going vertical sync pulse of 63.5 \( \mu \)s. The vertical sync pulse starts 494 \( \times \) 31.75 \( \mu \)s = 15.68 ms after the beginning of the first video line.

**Line 91:** This line describes the computation of the combinatorial blanking signal. The video is blanked after 256 pixels on a line are displayed, or after 480 lines are displayed.

**pipeline_blank:** This process describes the operation of the pipelined video blanking signal. Within the process, the blanking signal is stored in a register so it can be used during the next stage of the pipeline when the color is computed.

**Lines 104 – 106:** On these lines, the RAM is permanently selected and writing to the RAM is disabled. This makes the RAM look like a ROM, which stores video data. In addition, the outputs from the RAM are disabled when the video is blanked since there is no need for pixels during the blanking intervals. This isn't really necessary since no other circuit is trying to access the RAM.

**Line 113:** The address in RAM where the next four pixels are stored is calculated by concatenating the lower nine bits of the line counter with bits 7,8,5,4,3 and 2 of the pixel counter. With this arrangement, the line counter stores the address of one of \(2^9=512\) pages. Each page contains \(2^8=64\) bytes. Each byte contains four pixels, so each page stores one line of 256 pixels. The pixel counter increments through the bytes of a page to get the pixels for the current line. (Note that we don't need to use bits 1 and 0 of the pixel counter when computing the RAM address since each byte contains four pixels.) After the line is displayed, the line counter is incremented to point to the next page.

**update_pixel_register:** This process describes the operation of the register that holds the byte of pixel data read from RAM. The register is asynchronously cleared when the VGA circuit is reset. The register is updated on the rising edge of each pixel clock. The pixel register is loaded with data from the RAM whenever the lowest two bits of
the pixel counter are both zero. The active pixel is always in the lower two bits of the register. Each pixel in the RAM data byte is shifted into the active position by right shifting the register two bits on each rising clock edge.

**map_pixel_to_rgb:** this process describes the process by which the current active pixel is mapped into the six bits that drive the red, green and blue color guns. The register is set to zero (which displays as the color black) when the reset input is high. The color register is clocked on the rising edge of the pixel clock since this is the rate at which new pixel values arrive. The value clocked into the register is a function of the pixel value and the blanking input. When the pipelined blanking input is low (inactive), the color displayed on the monitor is red, green, blue, or white depending upon whether the pixel value is 00, 01, 10, or 11, respectively. When the pipelined blanking input is high, the color register is loaded with zero (black).

- Listing17: VHDL code for a VGA generator.

```vhdl
001- LIBRARY IEEE;
002- USE IEEE.STD_LOGIC_1164.ALL;
003- USE IEEE.std_logic_unsigned.ALL;
004-
005- ENTITY vga_generator IS
006- PORT
007- ( 
008-   clk: IN STD_LOGIC; -- VGA dot clock
009-   reset: IN STD_LOGIC; -- asynchronous reset
010-   hsync: OUT STD_LOGIC; -- horizontal (line) sync
011-   vsync: OUT STD_LOGIC; -- vertical (frame) sync
012-   rgb: OUT STD_LOGIC_VECTOR(5 DOWNTO 0); -- red,green,blue colors
013-   address: OUT STD_LOGIC_VECTOR(14 DOWNTO 0); -- address into video RAM
014-   data: IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- data from video RAM
015-   ceb: OUT STD_LOGIC; -- video RAM chip enable
016-   oeb: OUT STD_LOGIC; -- video RAM output enable
017-   web: OUT STD_LOGIC -- video RAM write enable
018- );
019- END vga_generator;
020-
021- ARCHITECTURE vga_generator_arch OF vga_generator IS
022- SIGNAL hcnt: STD_LOGIC_VECTOR(8 DOWNTO 0); -- horiz. pixel counter
023- SIGNAL vcnt: STD_LOGIC_VECTOR(9 DOWNTO 0); -- vertical line counter
024- SIGNAL pixrg: STD_LOGIC_VECTOR(7 DOWNTO 0); -- byte register for 4 pix
025- SIGNAL blank: STD_LOGIC; -- video blanking signal
026- SIGNAL pblank: STD_LOGIC; -- pipelined video blanking signal
027- SIGNAL int_hsync: STD_LOGIC; -- internal horizontal sync.
028- BEGIN
029-
030-   inc_horiz_pixel_counter:
031- PROCESS(clk,reset)
032-   BEGIN
033-     IF reset='1' THEN -- reset asynchronously clears pixel counter
034-       hcnt <= "000000000";
035-     ELSIF (clk'EVENT AND clk='1') THEN
036-       IF hcnt<380 THEN -- pixel counter resets after 381 pixels
```
hcnt <= hcnt + 1;
ELSE
hcnt <= "000000000";
END IF;
END IF;
END PROCESS;
inc_vert_line_counter:
PROCESS(int_hsyncb.reset)
BEGIN
IF reset='1' THEN -- reset asynchronously clears line counter
vcnt <= "0000000000";
ELSIF (int_hsyncb EVENT AND int_hsyncb='1') THEN
IF vcnt<527 THEN -- vert. line counter rolls-over after 528 lines
vcnt <= vcnt + 1;
ELSE
vcnt <= "0000000000";
END IF;
END IF;
END PROCESS;
generate_horiz_sync:
PROCESS(clk,reset)
BEGIN
IF reset='1' THEN -- reset asynchronously inactivates horiz sync
int_hsyncb <= '1';
ELSIF (clk'EVENT AND clk='1') THEN
IF (hcnt>=291 AND hcnt<337) THEN
-- horiz. sync is low in this interval to signal start of new line
int_hsyncb <= '0';
ELSE
int_hsyncb <= '1';
END IF;
END IF;
hsyncb <= int_hsyncb; -- output the horizontal sync signal
END PROCESS;
generate_vert_sync:
PROCESS(int_hsyncb,reset)
BEGIN
IF reset='1' THEN -- reset inactivates vertical sync
vsyncb <= '1';
-- vertical sync is recomputed at the end of every line of pixels
ELSIF (int_hsyncb'EVENT AND int_hsyncb='1') THEN
IF (vcnt>=490 AND vcnt<492) THEN
-- vert. sync is low in this interval to signal start of new frame
vsyncb <= '0';
ELSE
vsyncb <= '1';
END IF;
END IF;
END PROCESS;
-- blank video outside of visible region: (0,0) -> (255,479)
blank <= '1' WHEN (hcnt>=256 OR vcnt>=480) ELSE '0';
-- store the blanking signal for use in the next pipeline stage
pipeline_blank:
PROCESS(clk, reset)
BEGIN
IF reset='1' THEN
  pblank <= '0';
ELSIF (clk'EVENT AND clk='1') THEN
  pblank <= blank;
END IF;
END PROCESS;

-- video RAM control signals
ceb <= '0'; -- enable the RAM
web <= '1'; -- disable writing to the RAM
eob <= blank; -- enable the RAM outputs when video is not blanked

-- The video RAM address is built from the lower 9 bits of the vert
-- line counter and bits 7-2 of the horizontal pixel counter.
-- Each byte of the RAM contains four 2-bit pixels. As an example,
-- the byte at address \texttt{^h1234-^h0001,0010,0111,0100} contains the pixels
-- at \texttt{(row,col)=(^h048,^hD0),(^h048,^hD1),(^h048,^hD2),(^h048,^hD3)}.
address <= vcnt(8 DOWNTO 0) & hcnt(7 DOWNTO 2);

update_pixel_register:
PROCESS(clk, reset)
BEGIN
IF reset='1' THEN -- clear the pixel register on reset
  pixrg <= "00000000";
ELSIF (clk'EVENT AND clk='1') THEN
  -- pixel clock controls changes in pixel register
  -- the pixel register is loaded with the contents of the video
  -- RAM location when the lower two bits of the horiz. counter
  -- are both zero. The active pixel is in the lower two bits
  -- of the pixel register. For the next 3 clocks, the pixel
  -- register is right-shifted by two bits to bring the other
  -- pixels in the register into the active position.
  IF hcnt(1 DOWNTO 0)="00" THEN
    pixrg <= data; -- load 4 pixels from RAM
  ELSE
    pixrg <= "00" & pixrg(7 DOWNTO 2); -- R-shift pixel register
  END IF;
END IF;
END PROCESS;

-- the color mapper translates each 2-bit pixel into a 6-bit
-- color value. When the video signal is blanked, the color
-- is forced to zero (black).
map_pixel_to_rgb:
PROCESS(clk, reset)
BEGIN
IF reset='1' THEN -- blank the video on reset
  rgb <= "000000";
ELSIF (clk'EVENT AND clk='1') THEN -- update color every clock
-- map the pixel to a color if the video is not blanked
IF pblank='0' THEN
  CASE pixr(1 DOWNTO 0) IS
    WHEN "00" => rgb <= "110000"; -- red
    WHEN "01" => rgb <= "001100"; -- green
    WHEN "10" => rgb <= "000011"; -- blue
    WHEN OTHERS => rgb <= "111111"; -- white
  END CASE;
ELSE -- otherwise, output black if the video is blanked
  rgb <= "000000"; -- black
END IF;
END IF;
END PROCESS;
END vga_generator_arch;

• Listing18: XS40 UCF file for the VGA signal generator.

001- net clk      loc=p13;
002- net reset   loc=p44;
003- net data<0> loc=p41;
004- net data<1> loc=p40;
005- net data<2> loc=p39;
006- net data<3> loc=p38;
007- net data<4> loc=p35;
008- net data<5> loc=p81;
009- net data<6> loc=p80;
010- net data<7> loc=p10;
011- net address<0> loc=p3;
012- net address<1> loc=p4;
013- net address<2> loc=p5;
014- net address<3> loc=p78;
015- net address<4> loc=p79;
016- net address<5> loc=p82;
017- net address<6> loc=p83;
018- net address<7> loc=p84;
019- net address<8> loc=p59;
020- net address<9> loc=p57;
021- net address<10> loc=p51;
022- net address<11> loc=p56;
023- net address<12> loc=p50;
024- net address<13> loc=p58;
025- net address<14> loc=p60;
026- net ceb     loc=p65;
027- net web     loc=p62;
028- net oeb     loc=p61;
029- net rgb<0>  loc=p25;
030- net rgb<1>  loc=p26;
031- net rgb<2>  loc=p24;
032- net rgb<3>  loc=p20;
033- net rgb<4>  loc=p23;
034- net rgb<5>  loc=p18;
035- net hsyncb    loc=p19;
036- net vsyncb    loc=p67;

- Listing 18: XSG5 UCF file for the VGA signal generator.

001- net clk       loc=p9;
002- net reset     loc=p46;
003- net data<0>   loc=p44;
004- net data<1>   loc=p43;
005- net data<2>   loc=p41;
006- net data<3>   loc=p40;
007- net data<4>   loc=p39;
008- net data<5>   loc=p37;
009- net data<6>   loc=p36;
010- net data<7>   loc=p35;
011- net address<0> loc=p75;
012- net address<1> loc=p79;
013- net address<2> loc=p82;
014- net address<3> loc=p84;
015- net address<4> loc=p1;
016- net address<5> loc=p3;
017- net address<6> loc=p83;
018- net address<7> loc=p2;
019- net address<8> loc=p58;
020- net address<9> loc=p56;
021- net address<10> loc=p54;
022- net address<11> loc=p55;
023- net address<12> loc=p53;
024- net address<13> loc=p57;
025- net address<14> loc=p61;
026- net ceb       loc=p65;
027- net web       loc=p63;
028- net oeb       loc=p62;
029- net rgb<0>    loc=p21;
030- net rgb<1>    loc=p23;
031- net rgb<2>    loc=p19;
032- net rgb<3>    loc=p17;
033- net rgb<4>    loc=p18;
034- net rgb<5>    loc=p14;
035- net hsyncb    loc=p15;
036- net vsyncb    loc=p24;

The steps for compiling and testing the VGA design using an XS40 combined with an XStend Board are as follows:

- Synthesize the VHDL code in the VGA40\VGA.VHD file for an XC4005XL FPGA.
- Compile the synthesized netlist using the VGA40.UCF constraint file (Listing 18).
- Mount an XS40 Board in the XStend Board and attach the downloading cable from the XS40 to the PC parallel port. Apply 9VDC through jack J9 of the XS40. Place shunts on jumpers J4, J7, and J8 of the XStend Board to enable the LED displays. Remove the shunt on jumper J17 to keep the XStend codec serial output from...
interfering with the DIP switch logic levels. Set all the DIP switches to the OPEN position.

- Attach a VGA monitor to the DB-HD15 connector (J5).
- Download the VGA40.BIT file and a video test pattern into the XS40/XStend combination with the command: XSLOAD TESTPATT.HEX VGA40.BIT.
- Release the reset to the VGA circuitry with the command: XSPORT 0.
- Observe the color bars on the monitor screen.

The steps for compiling and testing the design using an XS95 combined with an XStend Board are as follows:

- Synthesize the VHDL code in the VGA95\VGA.VHD file for an XC95108 CPLD.
- Compile the synthesized netlist using the VGA95.UCF constraint file (Listing 19).
- Generate an SVF file for the design.
- Mount an XS95 Board in the XStend Board and attach the downloading cable from the XS95 to the PC parallel port. Apply 9VDC though jack J9 of the XS40. Place shunts on jumpers J4, J7, and J8 of the XStend Board to enable the LED displays. Remove the shunt on jumper J17 to keep the XStend codec serial output from interfering. Set all the DIP switches to the OPEN position.
- Attach a VGA monitor to the DB-HD15 connector (J5).
- Download the VGA95.SVF file and a video test pattern into the XS95/XStend combination with the command: XSLOAD TESTPATT.HEX VGA95.SVF.
- Release the reset to the VGA circuitry with the command: XSPORT 0.
- Observe the color bars on the monitor screen.

Reading Keyboard Scan Codes Through the PS/2 Interface

This example creates a circuit that accepts scan codes from a keyboard attached to the PS/2 interface of the XStend Board. The binary pattern of the scan code is displayed on the bargraph LEDs. In addition, if a scan code for one of the keys '0'—'9' arrives, then the numeral will be displayed on the right LED display of the XStend Board.

The format of the scan code transmissions from the keyboard are shown in Figure 10. The keyboard electronics drive the clock and data lines. The start of a scan code transmission is indicated by a low level on the data line on the falling edge of the clock. The eight bits of the scan code follow (starting with the least-significant bit) on successive falling clock edges. These are followed by an odd-parity bit and then a high-level stop bit.

When the clock line goes high after the stop bit, the receiver (in this case, the FPGA or CPLD on the XS Board inserted in the XStend Board) can pull the clock line low to inhibit any further transmissions. After the clock line is released and it returns to a high level, the