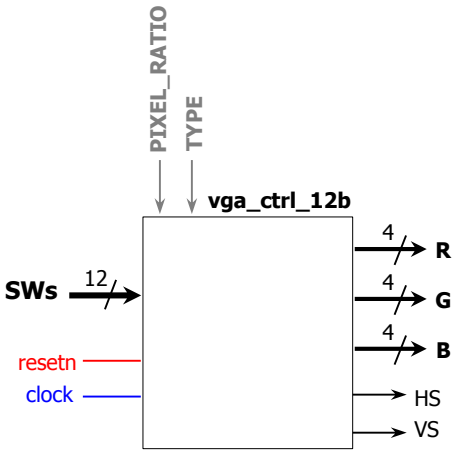
VGA Controller

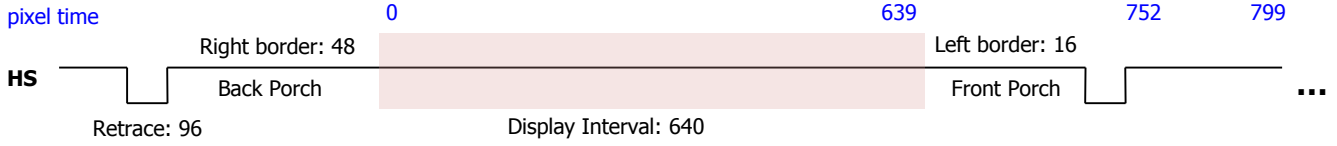
Introduction

- The architecture presented here outputs 12-bit color (4 bit per color). For other cases (e.g.: 3-bit color, 9-bit color, 15 bit color), the code requires minor modifications.
- The figure depicts the VGA core. There are two parameters:
  - ✓ TYPE:
    - BASIC: Simple with 3 input bits: 8-bit color
    - BASIC12: Simple with 12 input bits: 12-bit color
    - MEMORY: With memory: 256x256 12-bit color image
  - ✓ PIXEL RATIO: It is the ratio between the input clock and the pixel clock (25 MHz). Accepted values: 4, 2.

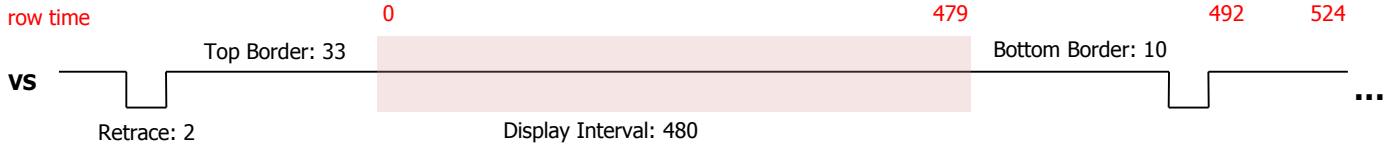


VGA Display Timing

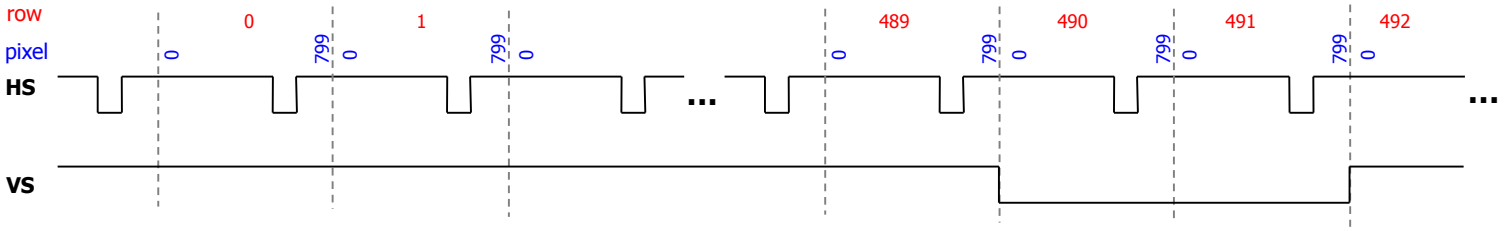
- VGA timing: It requires fine control over the HS and VS signals. Here, we use a 25 MHz pixel clock (ratio at which a pixel can be written) and a frame of 640 columns by 480 rows (640x480).
- The figure below shows the timing for HS. HS can be used to keep track of pixels written on a row. Pixel time: time interval at which a pixel is present. For 25 MHz clock, this amounts to 40 ns. For a row, we need 800 pixels. We decided to enumerate them this way: display interval (0 to 639), right border (640 to 655), retrace (656 to 751), and the left border (752 to 799).



- The figure below shows the timing for VS. VS can be used to keep track of the row we are at. Row time: 800 pixel times. For 25 MHz clock, this amounts to 800x40ns = 32 us. For a frame, we need 525 rows. We decided to enumerate them this way: display interval (0 to 479), bottom border (480 to 489), retrace (490 to 491), and the top border (492 to 524).



- The figure below shows the timing for HS and VS. Note how VS is 0 at rows 490 to 491. The display interval is 0 to 639 for HS and 0 to 479 for VS. Outside of the display interval, we are supposed to set the RGB values to 0.



VGA Display System

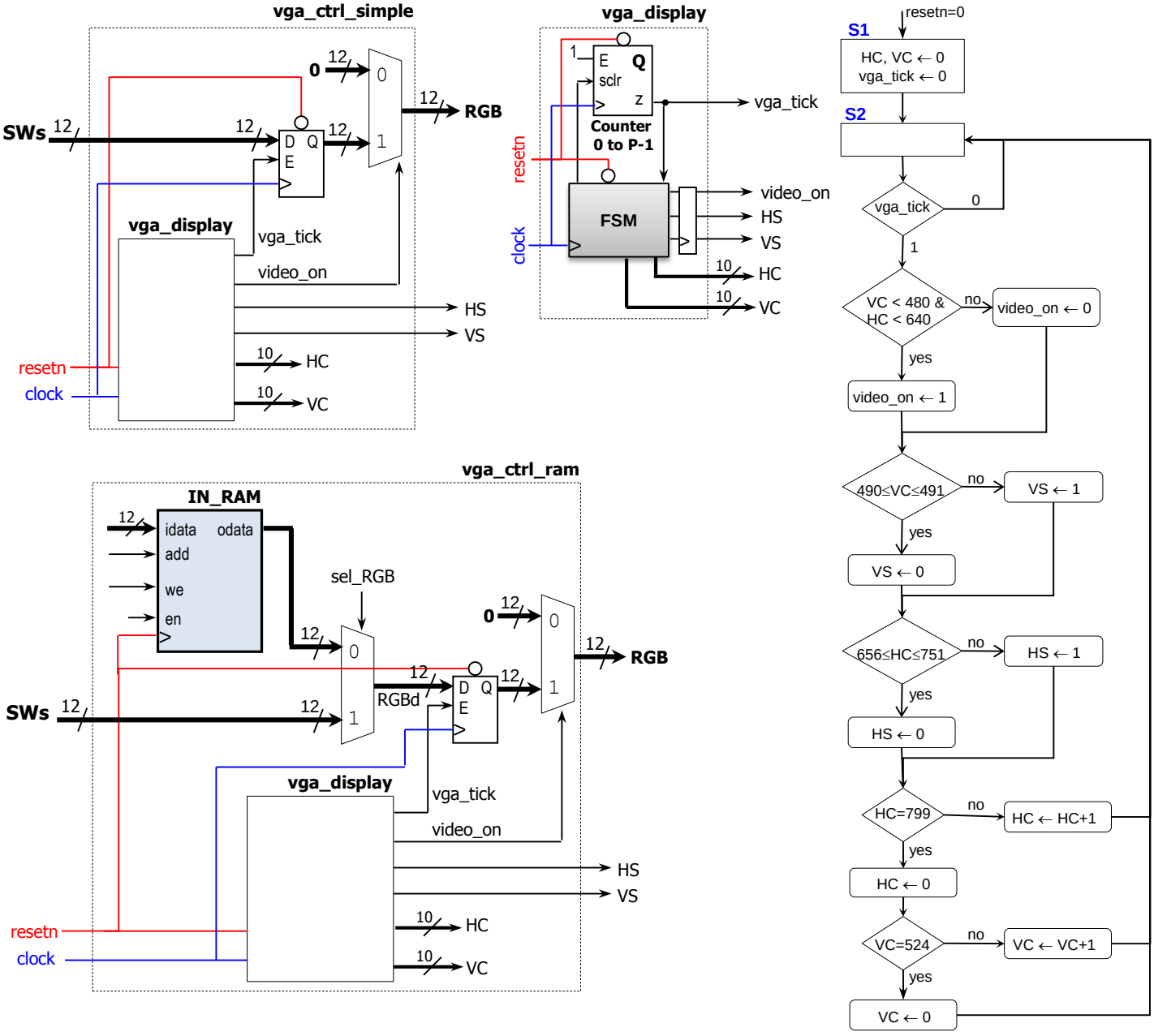
- The digital system is in charge of controlling the HS and VS signals as well as placing the RGB signals at the correct times. There are two versions of the 'vga_ctrl_12b' core: SIMPLE and MEMORY:
  - ✓ SIMPLE: It accepts as 12-bit input and displays the specific color on the screen. 12-bit color: Controlled by 12 switches. 3-bit color: Controlled by 3 switches (the 12-bit input is created by repeating the values).
  - ✓ MEMORY: 12-bit color is controlled by contents of a memory. The background color is controlled by switches.
- The most important component of the architecture is 'vga_display' block.

VGA_DISPLAY BLOCK

- This block generates the following signals:
  - ✓ HS and VS. Registered outputs.
  - ✓ Pixel clock (vga_tick): The frequency of this signal is 25 MHz. This signal comes from a modulo-P counter. Its frequency is the one at which input data can be placed on the displaying intervals. We use a 640x480 display with a 25 MHz pixel

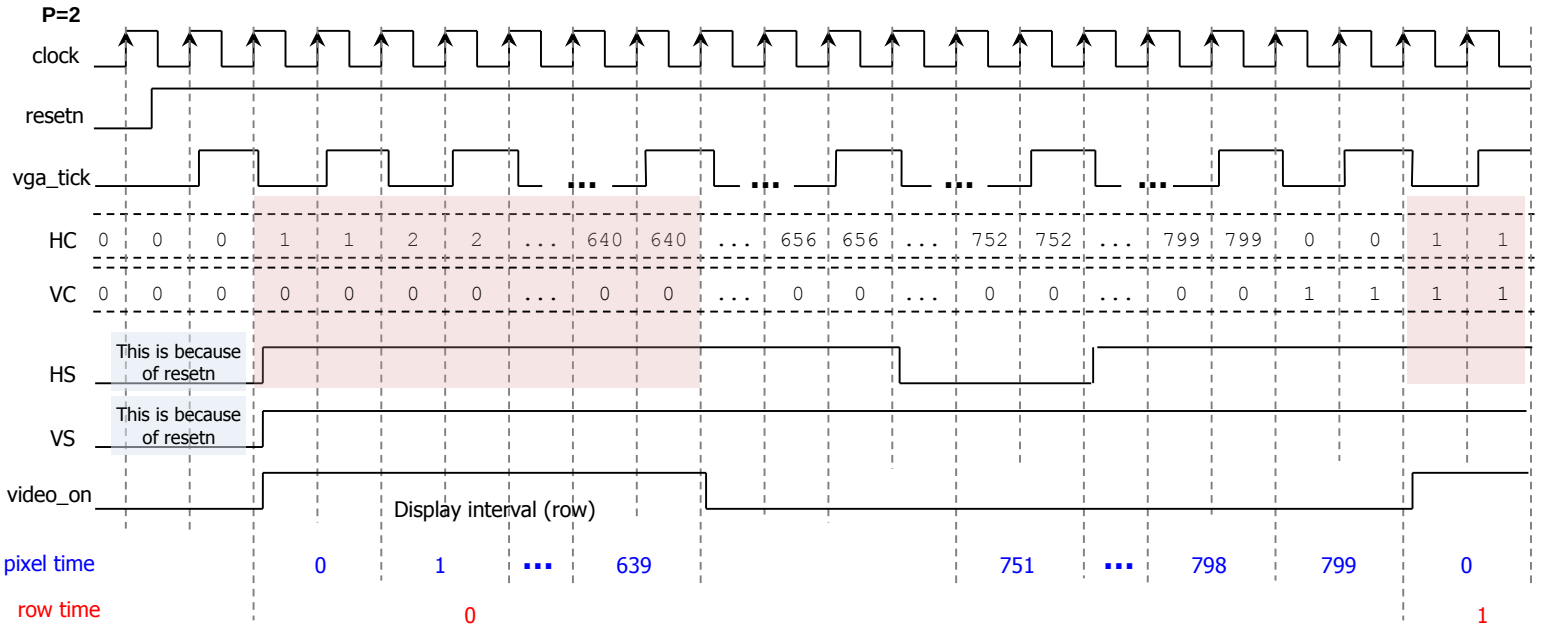
clock. The 25 MHz signal is generated from the main clock. For a 100 MHz input clock (like in the NEXYS4 Board), we need a counter modulo-4 (P=4). Other boards have an input clock of 50 MHz, requiring a modulo-2 (P=2) counter.

- ✓ HC and VC: These are 10-bit counters that provide a location so that the user can place a pixel at the given location. For a given HC and VC, the user must place the RGB data when the vga_tick pulse hits.
- ✓ $video\_on$: This registered signal, when asserted, indicates that we can write pixels on the screen. If it is 0, the RGB data should be 0.



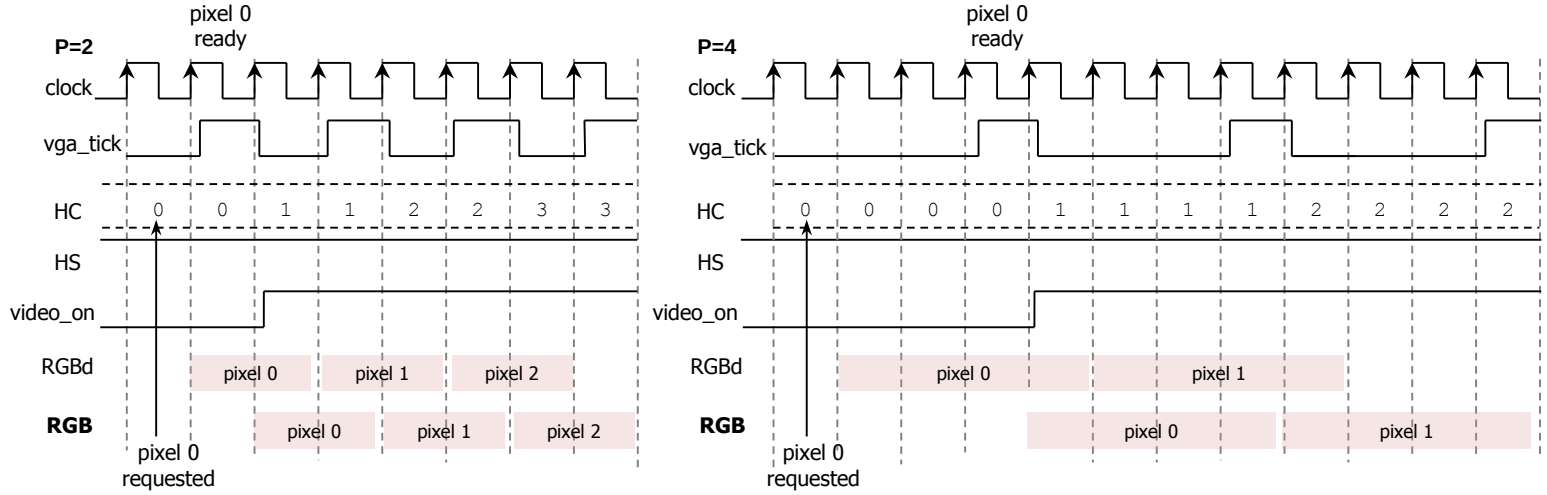
Timing Diagram during operation

- Here we show a timing diagram from power-up. At the beginning $HS = 0$, and it becomes 1 once $video\_on = 1$. After this, HS will be 0 only when doing retrace.
- Note that the values of HC, VC, video_on, HS, VS are only updated when $vga\_tick = 1$.
- Note that the HC value is one pixel time ahead of the actual pixel time. In the example, with P=2 → HC is two clock cycles ahead of the pixel time (as defined by HS). VC is also two clock cycles ahead of the pixel time (and of row time). If P=4, HC and VC are four clock cycles ahead of the pixel time.
- This allows the user to use HC and VC to know the coordinate before placing the pixel.
- This configuration allows us to request data (say from memory) and place it when $video\_on = 1$; the user must place the data before a clock tick and $vga\_tick = 1$. What really matters for the RGB is the pixel time when HS and VS are 1.



VGA_CTRL_SIMPLE AND VGA_CONTROL_RAM

- These circuits are designed so that the right data is present for HS and VS when $video\_on = 1$. For example, on the first *pixel clock cycle* (P clock cycles*)* that $video\_on = 1$ (right after being 0), we need to place the first pixel on RGB. In the previous clock cycle, the vga_tick signal was asserted (for HC=0), which means that the first pixel had to be captured there. So, from the moment we get a certain HC and VC, the user has at most P-1 cycles to generate the data so that it is captured when $vga\_tick = 1$. The captured data will appear on RGB.
  - ✓ SIMPLE case: this is guaranteed as data is generated immediately.
  - ✓ MEMORY case: the memory should take at most P-1 cycles from the time the address (based on HC and VC) is generated. The address generator should not include registers for simplicity. The figure below shows a portion of normal operation; the memory only takes one cycle to produce the output (on RGBd). For HC=VC=0, the first pixel is requested and appears on the register input (RGBd) on the next cycle, where it is captured when $vga\_tick = 1$ (this data appears in RGB).



'Simple' Case

- Here, RGB inputs come from switches. They are registered only when vga_tick is enabled. When $video\_on = 1$, the registered RGB color appears on the entire screen. Timing here is not an issue since the same color appears in the whole screen.

'Memory' Case

- Here, we have to use the memory available (BRAM) in the specific chip. In the case of the NEXYS-4 Board, we have the XC7A100T Artix-7 FPGA. Word size in the BRAM are restricted to 8, 16, and 32. So, we use 16 bits to represent each pixel (we only need 12, so the 4 MSBs are wasted).
- VGA has a maximum size of 640x480. For the XC7A100T Artix-7 FPGA, an image this size does not fit. The maximum possible size that fits is 480x576. The given code works for square images whose sides are power of 2. For other cases, the code needs to be modified (the memory address generation).

- Memory addressing scheme: If we store the image on the memory in a raster-scan fashion, the address of the memory depends on HC and VC as follows:
  - ✓ For an image of size $nc \times nr$, where nc is the number of columns and nr the number of rows, the address is given by: $VC \times nc + HC$. For example, for an image of size 640x480, the address would be: $VC \times 640 + HC$.
  - ✓ For square images whose side is a power of 2, the address is $VC \times 2^n + HC$, where n=number of pixels per row (or column). Note that we can simplify the equation as $VC(n-1:0)\&HC(n-1:0) = VC \times 2^n + HC$.

- IN_RAM: By default the example is set to 256x256 16-bit words. You can easily modify it for any $n \times n$ image, where n is a power of 2. For other cases, including $nc \times nr$, you need to make some modifications to the code.
- IN_RAM: We use this memory to place a 12-bit color 256x256 image. If sel_RGB=1, the image pixels are displayed. If sel_RGB=0, we are outside the area of the image, and the background is displayed (color set by SWs). The image can be positioned anywhere in the screen by controlling the conditions for the 'sel_RGB' signal as well as the address formula.
  - ✓ If we want the 256x256 image to appear starting from the position (0,0) in the screen (top-left corner), we need:
    - $sel\_RGB = 1$ when $HC < 256$ and $VC < 256$.
    - Address formula: $VC \times 256 + HC$.
  - ✓ If we want the 256x256 image to appear starting from the position (128,128) in the screen, we need:
    - $sel\_RGB = 1\ when\ 128 < HC < 256 + 128\ and\ 128 < VC < 256 + 128$.
    - Address formula (requires modification, coordinates translation): $(VC - 128) \times 256 + (HC - 128)$

- IN_RAM: In the given code, the 'en' input is set to '1', and the 'we' input is set to zero. The image is pre-loaded in the IN_RAM from a text file (created from MATLAB from an actual image). The memory contents can always be modified by using the 'we', 'add', and 'idata' inputs.

Final considerations

- If we want to apply a pixel-to-pixel operation to an image (like gamma correction based on LUTs), we just need to place that LUT after the RAM.
- In the example, a 256x256 16-bit word memory was instantiated as a collection of Block RAM primitives. You can also use the Xilinx Core Generator GUI to generate customized BRAM-based memories.
- Project: `vga_control.zip`
  Contents: VHDL files, text file (image).
  Ancillary file: MATLAB script (image to text)