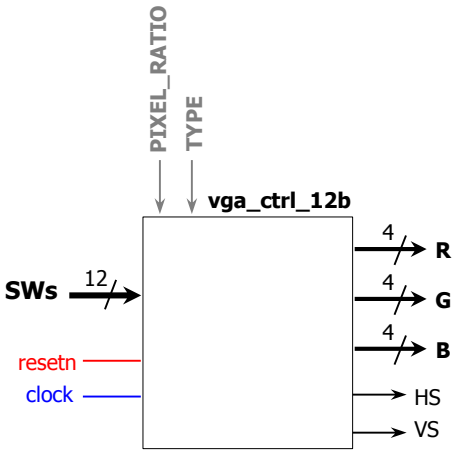
VGA Controller

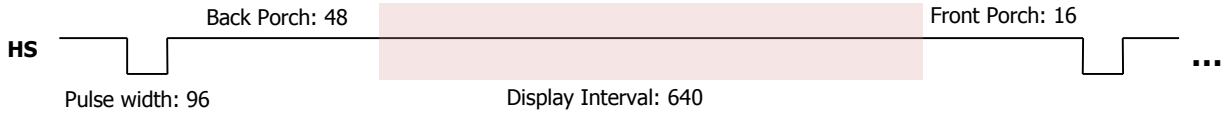
Introduction

- The core 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 System

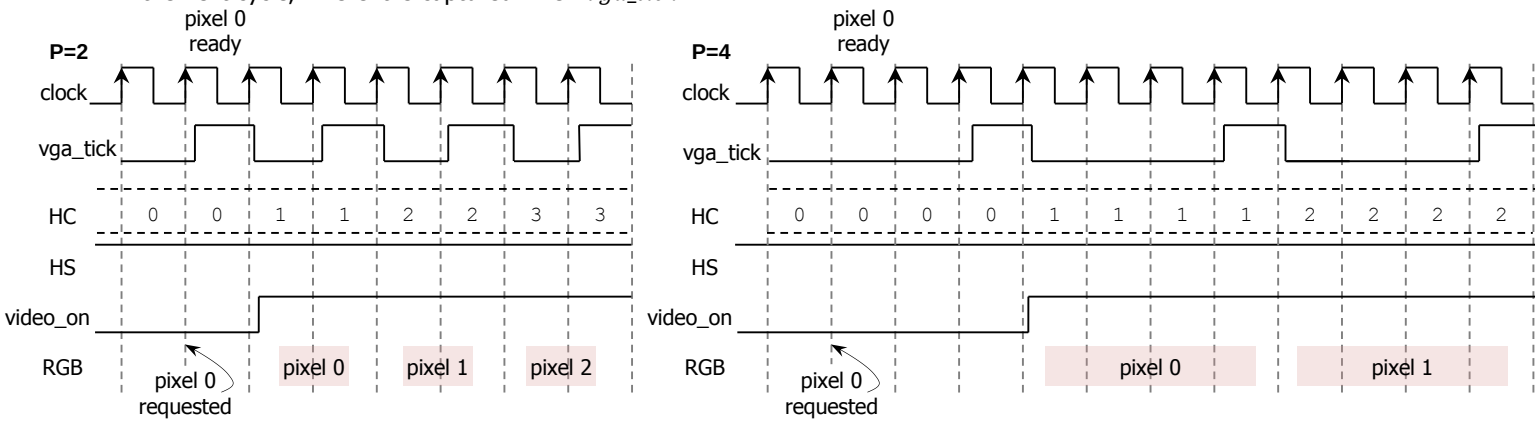
- VGA timing: It requires fine control over the HS and VS signals. The figure shows the timing for HS using a 25 MHz pixel clock (ratio at which a pixel can be written) and 640 columns and 480 rows (640x480).

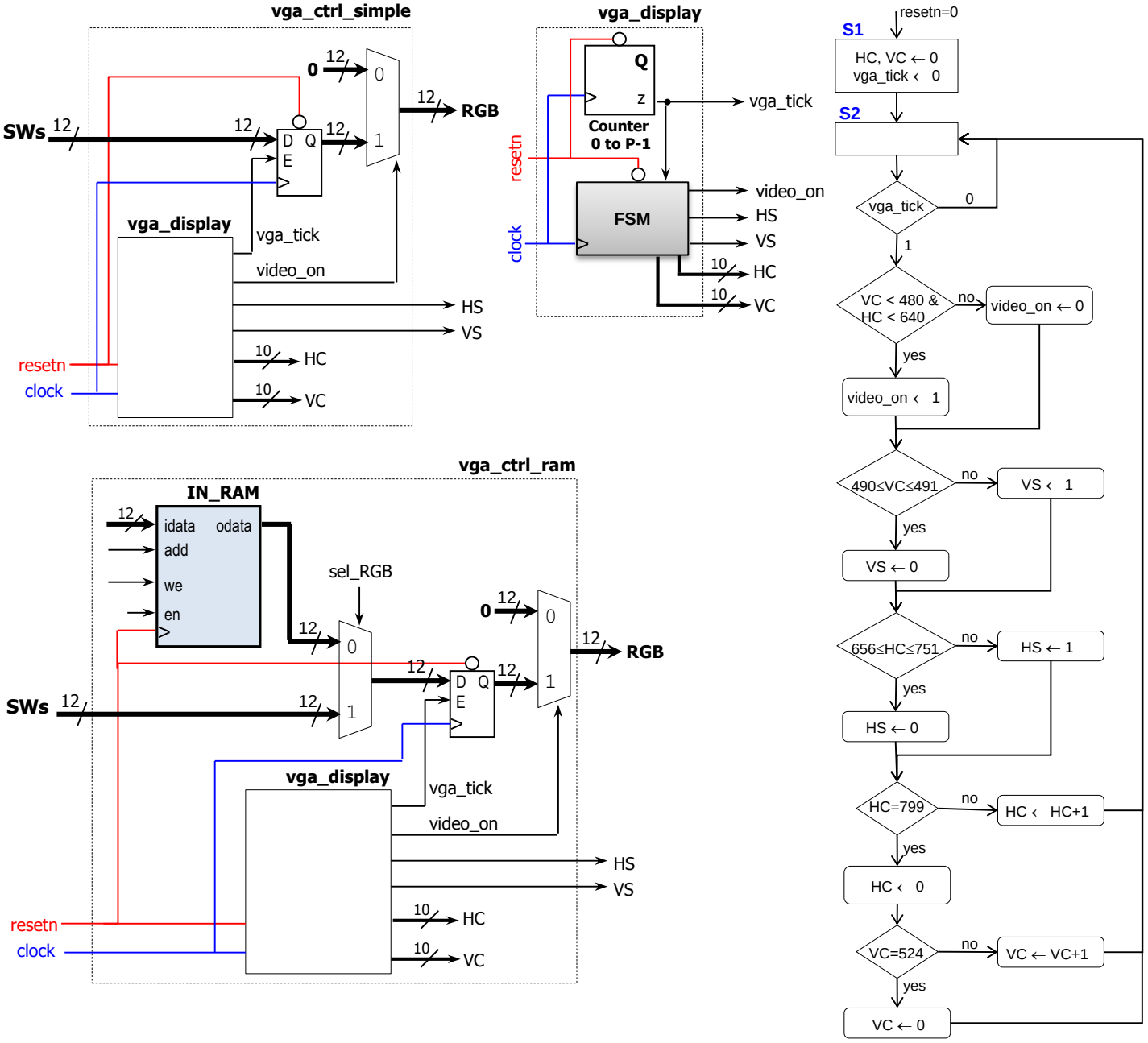


- The digital system then requires controlling the HS and VS signals as well as placing the RGB signals at the correct times. There are two versions of the 'vga_ctrl_12b': SIMPLE and MEMORY:
  - ✓ SIMPLE: It accepts as 12-bit input and displays the specific color on the screen. 12-bit color: Controlled 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.
- A block diagram is presented for both SIMPLE and MEMORY cases in the figure next page. The most important component is the 'vga_display' block. This block generates the HS and VS signals. In addition, it generates the pixel clock (vga_tick) as well as a $video\_on$ signal that tell us when we can write pixels on the screen.

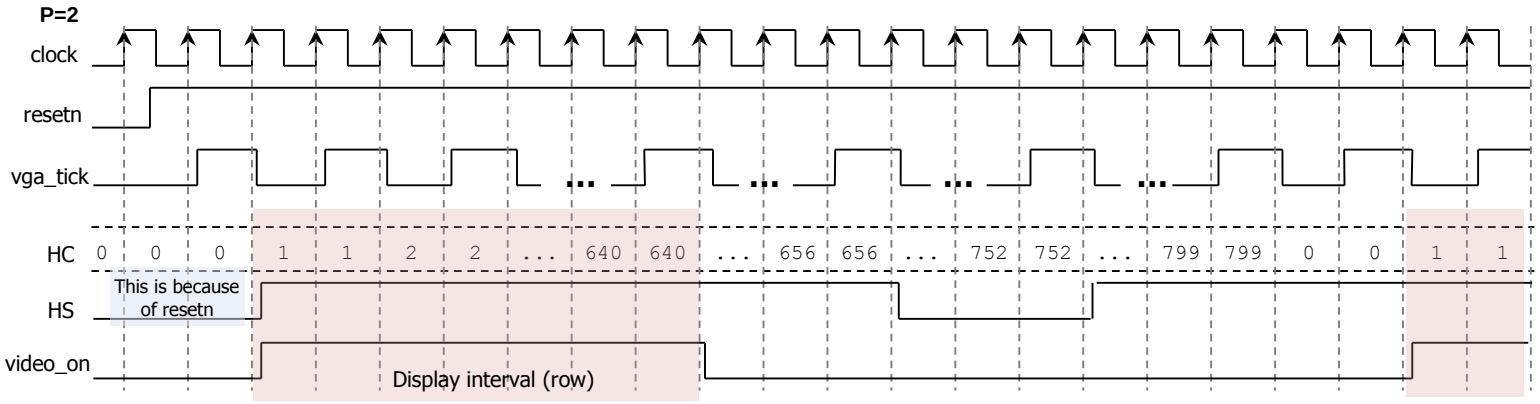
VGA_DISPLAY BLOCK

- HC, VC: 10-bit counter outputs. HS, VS, and $video\_on$ are registered outputs (they only change on the clock edge). When resetn=0: HC= VC = 0, HS = VS=0.
- vga_tick: This signal comes from a modulo-P counter. It is also called the pixel clock: 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.
- The core is designed so that the proper 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. This is true for $P \geq 2$ (only if data takes at most P-1 cycles to appear from the moment it is requested). For the SIMPLE case, this is guaranteed. For the 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 generation should not require registers for simplicity. In the figure below (the memory only takes one cycle to produce the output), if HC=VC=0, the first pixel is requested and appears on the register input on the next cycle, where it is captured when $vga\_tick = 1$.





- In the timing diagram P =2 → HC is two cycles ahead of HS (the same happens with VC and VS). What really matters for the RGB is the time HS and VS are in 1. Actually, the fact that HC and VC are ahead in time help us to request data from memory and place it when $video\_on = 1$ and HS is 1 (or VS=1).



SIMPLE MODE

- 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 MODE

- 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 case, the code needs to be modified (the part of the memory address).
- IN_RAM: 256x256 16-bit words. We use this memory to place our 12-bit color image of size 256x256. The image can be positioned anywhere in the screen by adequately controlling the signal 'sel_RGB'. As the code stands, the image is place of the top-left corner (sel_RGB = 1 when HC < 256 and VC < 256). By playing with the condition for sel_RGB, we can position the image anywhere.
- Memory addressing scheme: The address of the memory depends on HC and VC. For square images whose side is a power of 2, the address is simply $VC(n-1:0)\&HC(n-1:0)$, where n=number of pixels per row (or column) in an square image whose side is a power of 2. This is true because $VC(n-1:0)\&HC(n-1:0) = VC \times 2^n + HC$. For other images, this equation needs to change.
- 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 an operation to an image (like gamma correction based on LUTs), we just need to place that LUT after the RAM.
- The 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)