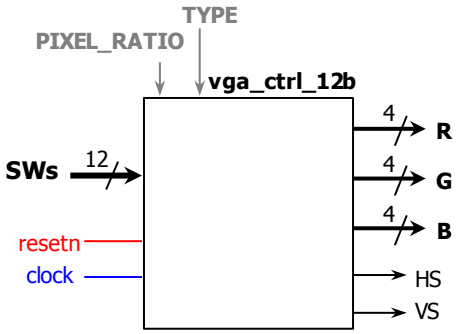
VGA Controller

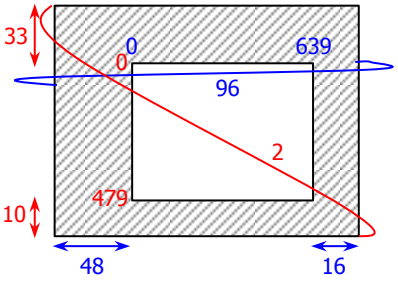
Introduction

- 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.

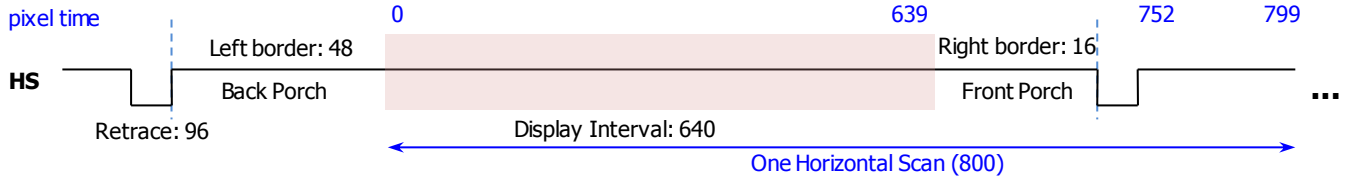


VGA Display Timing

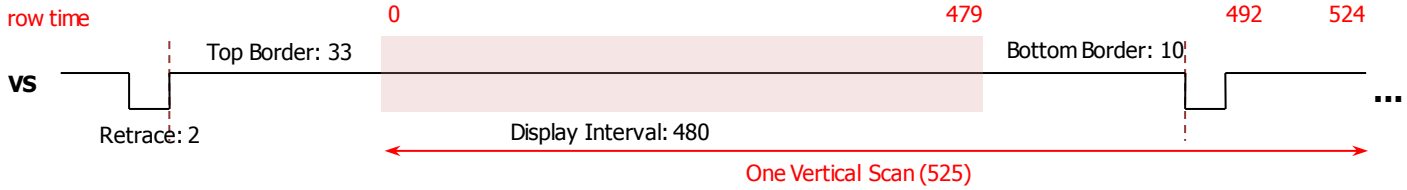
- **VGA timing**: It requires fine control of the HS and VS signals. Here, we use a frame of 640 columns by 480 rows (640x480) and a pixel clock (ratio at which a pixel can be written) of 25 MHz. For other timings, refer to the VGA VESA® documentation.
  - ✓ The Pixel time (time interval at which a pixel is present) is then 40 ns.
  - ✓ For a 640x480 frame: 640x480 is the display area, however we need the time for 800 columns and 525 rows. This results in a refresh rate of $\frac{25000000}{800x525} = 59.52\ fps$.



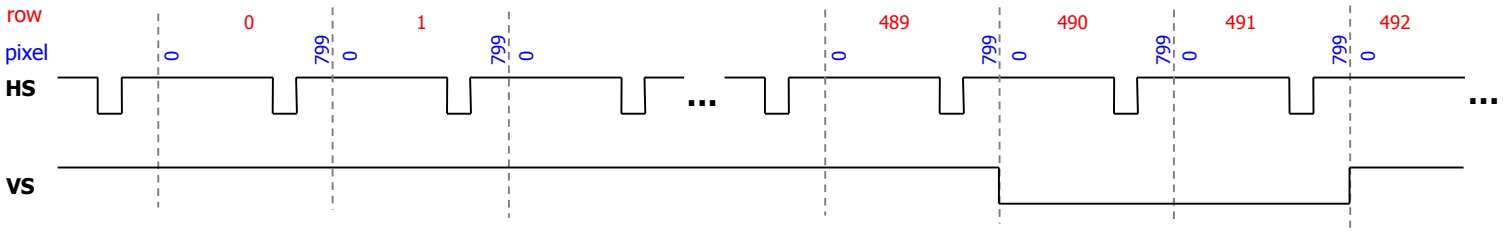
- **Timing for HS** (see figure below)**:** HS keeps track of pixels written on a row. For an 800-pixel row, we enumerate the pixels this way: display interval (0 to 639), right border (640 to 655), retrace (656 to 751), and the left border (752 to 799).



- **Timing for VS** (see in figure below): VS keeps track of the row we are at. Row time: 800 pixel times. For 25 MHz clock, this amounts to 800x40ns = 32 us. For a 525-row frame, we enumerate the rows 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. Display Interval: HS ∈ [0, 639] and VS ∈ [0, 479]. **Important**: outside of the display interval, we must to set the RGB values to 0.

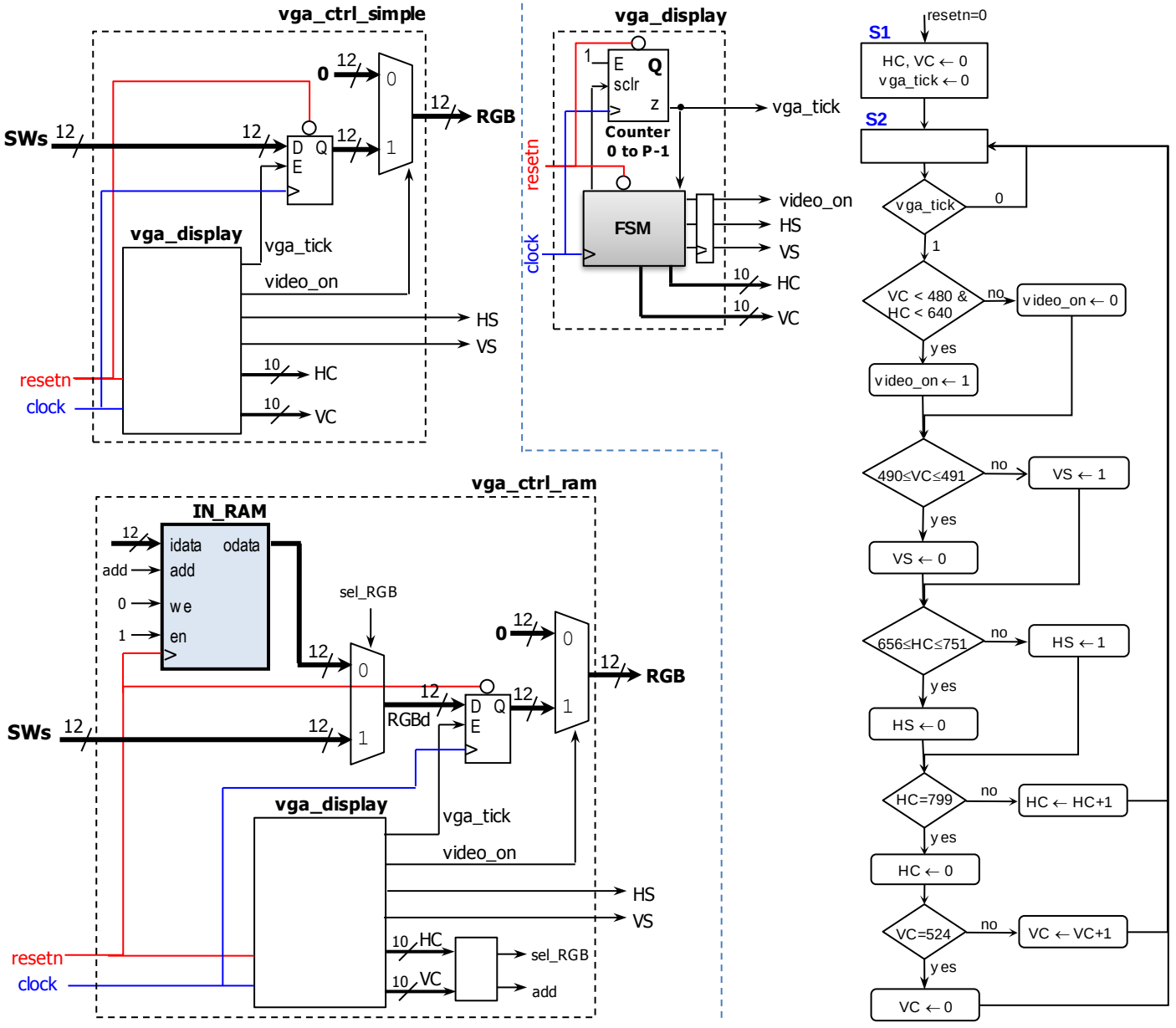


VGA Display System

- 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 a 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.

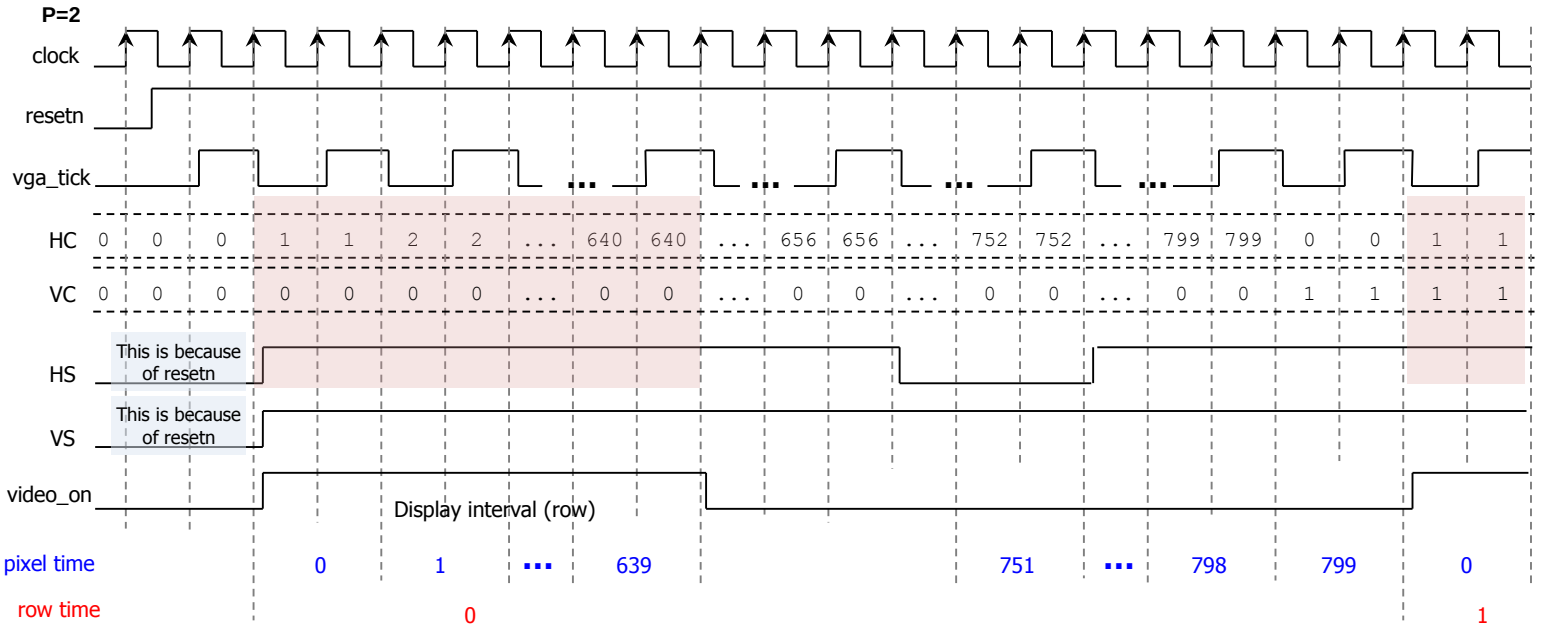
VGA_DISPLAY BLOCK

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



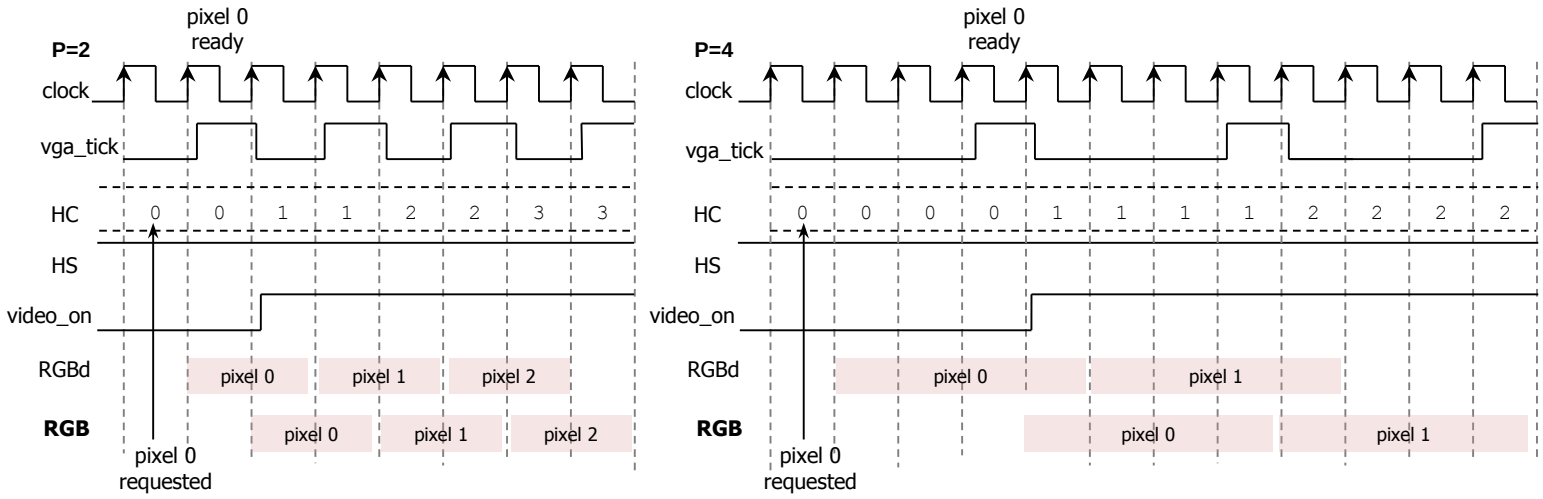
Timing Diagram during operation

- 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 in Retrace. So, the first row (or even first frame) might be incorrect, but the next frames will be correct.
- Note that the values of HC, VC, video_on, HS, VS are only updated when $vga\_tick = 1$ and rising edge of the FPGA clock.
- 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 is what happens in Nexys-4). This allows the user to use HC and VC as coordinates, find the pixel, and then placing it on RGB.
- 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

- Note that the register whose enable is controlled by vga_tick makes sure that data is kept for the duration of a pixel time. It also makes sure to capture the proper pixel at the right time.
- 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 new HC and VC, the circuit has up to P-1 cycles to place the data on RGBd 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).



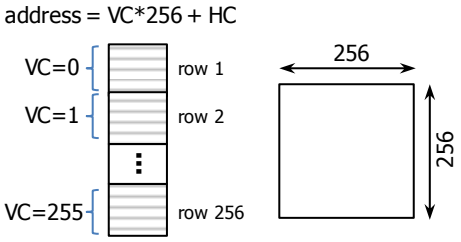
SIMPLE CASE (VGA_CTRL_SIMPLE)

- 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 (VGA_CTRL_RAM)

- Here, we use the available memory in the FPGA (BRAM). The Nexys-4 Board contains the XC7A100T Artix-7 FPGA. BRAM word size is restricted to 8, 16, and 32. We use 16 bits for each pixel (only 12 actually needed, the 4 MSBs are unused).
- VGA frame size is 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 only works for square images whose sides are power of 2. So, the maximum square size that is supported is 256x256. 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 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 2^n is the number of pixels per row (or column). Note that we can simplify the equation as $VC(n-1:0)\&HC(n-1:0) = VC \times 2^n + HC$. This simplifies the required hardware.



IN_RAM (memory block):

- In the given code, the memory is set to 256x256 16-bit words. The memory stores a 12-bit color 256x256 image. You can easily change it for any $2^n \times 2^n$ image. For other cases, including $nc \times nr$, you need to make more modifications to the code.
- 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.
- The signal sel_RGB controls when the pixels from the memory are displayed:
  - ✓ 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).
- An image can be placed anywhere in the screen by controlling the conditions for sel_RGB signal and the address (add). For displaying purposes, when sel_RGB has to change, it must do it (at the latest) the moment vga_tick=1. We meet this requirement, since sel_RGB changes (when it needs to) right after we get a new HC and VC.
  - ✓ For the 256x256 image to appear starting from the position (0,0) in the screen (top-left corner), we need (default in the given code):
    - sel_RGB = 1 when HC<256 and VC<256
    - Address formula: add = VC×256+HC.
  - ✓ For 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 (coordinates translation): (VC-128)×256+(HC-128). Note that if VC or HC are lower than 128, we can assign the result to 0, but it is irrelevant as sel_rgb will be 0.

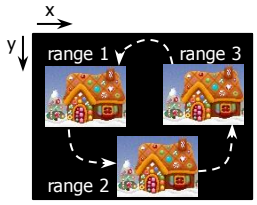
Making the image move

- In vga_ctrl_ram, sel_RGB and add are generated based on a rule HC and VC. If we allow for that rule to change over time, we can make the image move.
- In the example, we make a 128x128 image move in the following pattern every second: range 1 → range 2 → range 3 → range 1 … To do this, the block that generates sel_RGB and add should be an FSM (figure shown) where the rules for sel_RGB and add change every 1 second:

- Rule 1 (in state S1):
  - ✓ sel_rgb=1 when HC, VC ∈ range 1
  - ✓ Address: add = (VC-50)x128 + (HC-50)

- Rule 2 (in state S2):
  - ✓ sel_rgb=1 when HC, VC ∈ range 2
  - ✓ Address: add = (VC-200)x128 + (HC-100)

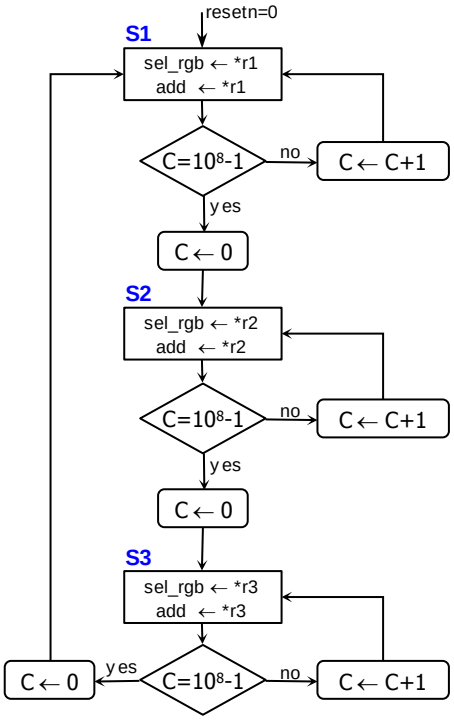
- Rule 1 (in state S3):
  - ✓ sel_rgb=1 when HC, VC ∈ range 3
  - ✓ Address: add = (VC-50)x128 + (HC-300)



Range 1 :
(50,50) to (128+49,128+49)

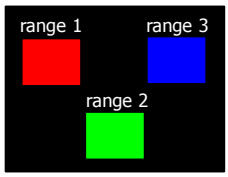
Range 2 :
(100,200) to (128+99,128+199)

Range 3 :
(300,50) to (128+299,128+49)

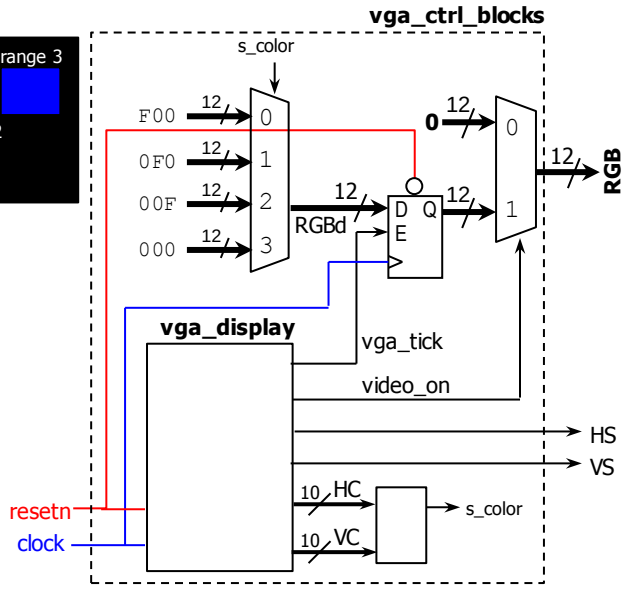


PLACING RECTANGLES ON THE DISPLAY

- If we want to show rectangles, each with different colors, we can use this architecture (based on vga_ctrl_simple).
- For example, if we want 3 rectangles (red, green, blue) over a black background, we could do:



- If HC,VC ∈ range 1 → s_color=00:
  ⇒ Rectangle (range 1) will be of color `F00` (red).
- If HC,VC ∈ range 2 → s_color=01:
  ⇒ Rectangle (range 2) will be of color `0F0` (green).
- If HC,VC ∈ range 3 → s_color=10:
  ⇒ Rectangle (range 3) will be of color `00F` (blue).
- If HC,VC ∉ range 1, range 2, or range 3 → s_color=11:
  ⇒ Remaining area will be of color `000` (black).
- We could make one of more rectangles move if the circuit that generates s_color is a FSM where the ranges for s_color change say every second.



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)