VGA Screen

The screen is broken up into blocks resulting in resolution of 80x60. The coordinates of the top-left, top-right, bottom left and bottom right blocks are: (0,0), (79,0), (0,59), (79,59) respectively as shown in Figure 1.

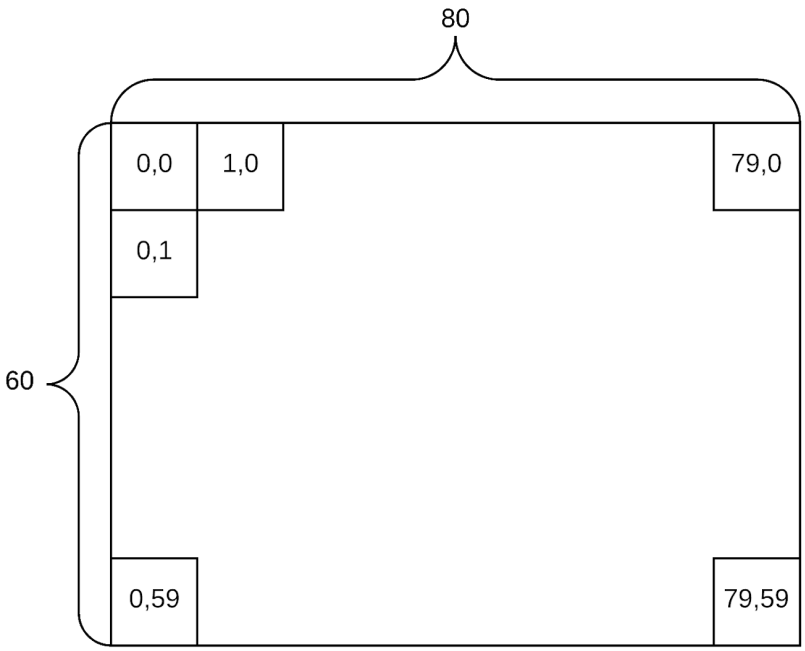

Figure 1: VGA Display Resolution

A VGA display works by writing a color to each pixel of the display in a continuous stream. The display starts at the top left pixel of the screen (0,0) and moves to the right, one row at a time. When the display gets to the last pixel on the bottom right (79,59), it starts over again at the top left corner (0,0). The display has to be continuously refreshed to create the illusion of motion.

VGA Color Data

Color information on the Basys3 VGA peripheral is controlled by 3 4-bit inputs, Red, Green, and Blue. This total color value in 12-bits is non-ideal for interfacing with the RAT which is an 8-bit device. To adapt, the color information for the VGA driver will be reduced to 8 bits. The 3 MSBs will control Red, the next 3 bits will control Green, and the 2 LSBs will control Blue. This results in the 8-bit color data (wd, rd) signals to be represented as: RRRGGGBB. Having more red bits asserted and no other bits asserted, makes the block "more red", so “full red” is "11100000", “full blue” is "00000011", and full green is "00011100". You can mix colors: white is "11111111" black is "00000000". This 8-bit color allows for 256 different colors to be displayed on the display

The 8-bits is extended to 12-bits in the constraints file by reusing bits. For Red and Green, the 3-bit color data is expanded to 4-bit by using the lsb twice. The 3-bit value $B_2B_1B_0$ is expanded to 4-bit value $B_2B_1B_0B_0$. For Blue, the 2-bit color data is expanded to 4-bit by using each bit twice. The 2-bit value B_1B_0 is expanded to 4-bit $B_1B_1B_0B_0$.

VGA Hardware

The VGA hardware driver consists of three different sub-circuits, a clock divider (vga_clk_div), a dual-ported memory(frameBuffer), and a VGA driver state-machine (vga_out). An overview of this circuit is shown in Figure 2.

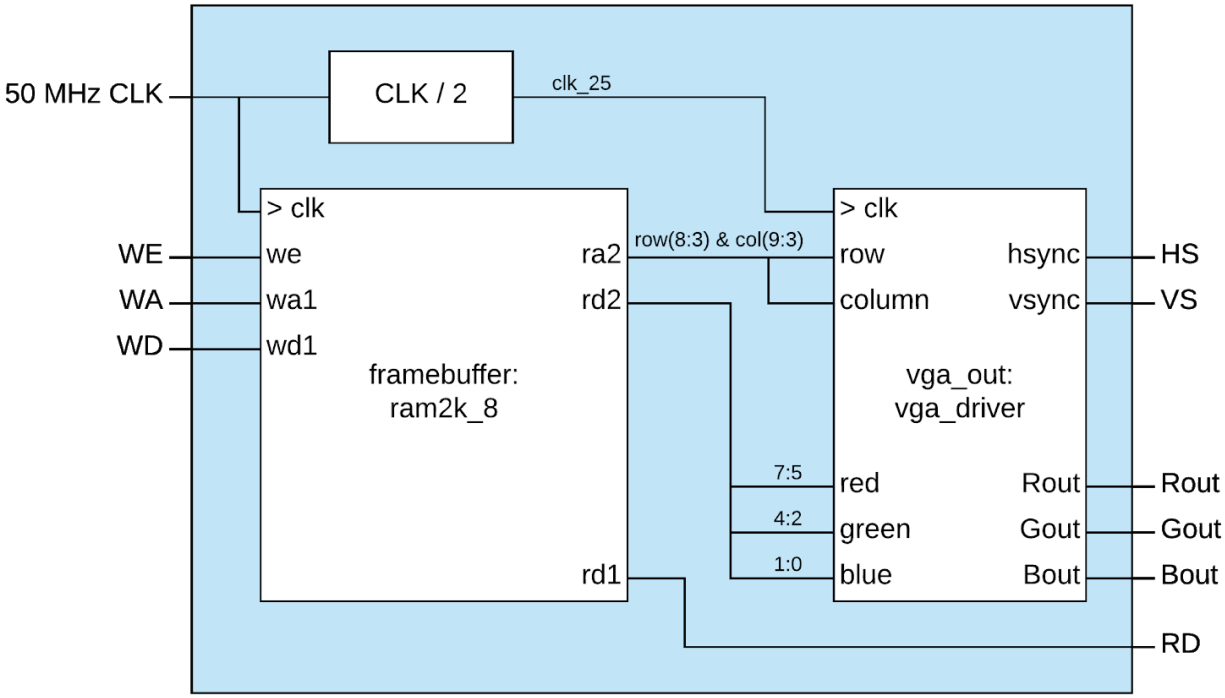

FIgure 2: VGA Driver

This circuit allows color data (wd) for each block to be stored into a frame buffer. Each block location (80x60) corresponds to a specific address (wa) in the framebuffer. The VGA driver state-machine reads from the framebuffer and refreshes the vga display. The VGA driver scans addresses (ra2) in the framebuffer and refreshes the display with the saved color (rd2) for each pixel. The VGA driver also creates the necessary timing signals to control the connected vga display. The horizontal sync and vertical sync (HS, VS) signals control the display resolution and refresh rate. Once data is written to the framebuffer, the VGA driver will automatically update the display as it is refreshed.

Writing color to the screen only requires interfacing with the framebuffer which is simply a block of memory. The data written (wd1) to the framebuffer is the 8-bit color value. The address (wa1) where the data is saved will correspond to the location, or pixel on the screen.

Framebuffer Address

The first block in the top left corner will correspond to address 0. The next block to the right will be address 1. This will continue to the last block on the top row which will correspond to address 39. The leftmost block on the 2nd row will not correspond to address 80 however. To make it easier to

program, each row will jump by power of 2, so the leftmost block on the 2nd row will correspond to address 128. The next block to the right will be 129, etc. The rightmost block on the 2nd row will be at address 187. The leftmost block on the 3rd row will be 256. So each column will increment the address by 1 while each row will increment the address by 128.

If every block had a consecutive address, so if the first block in row 2 had address 80 instead of 128, to have enough addresses for each block in the display, the address space would need to go from address 0 to 80x60 -1 = 4799. This would require a 13-bit address.

If the address was instead made by using the row and column value, the row must go from 0-59 which requires 6 bits, and the columns must go from 0-79 which requires 7 bits. These combined is also 13 bits. However it is easier to organize an address by using row and column values directly. So the 13-bit address can be comprised of $C_6C_5C_4C_3C_2C_1C_0R_5R_4R_3R_2R_1R_0$ When organizing the address in this way, the least significant bit of the column corresponds to the 7th bit of the address. $2^7 = 128$ which is why the rows increments the address by 128. If the address was done as consecutive values, it would require math operations to calculate the address from the rows and column values.

Interfacing to the VGA Driver

The RAT can only interface with peripherals in 8-bit chunks. This means multiple outputs will be required to write data to the framebuffer of the VGA driver. The address is 13-bits and will require two outputs. The color data is 8-bits and will require a third output. So to write color to a single block of the screen will require three OUT instructions. This means the RAT WRAPPER will need 3 different PORT_IDs to identify the connections to the VGA driver, 2 for the address (high and low) and 1 for the color data.

PORT_ID - Hi Address									PORT_ID - Lo Address							PORT_ID - Color Data							
x	x	x	x	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0

The multiple OUT instructions creates a potential problem for when the data should be saved to the framebuffer. Due to how the RAT MCU is connected to the output peripherals, data is held constant on all of the output lines until it is explicitly changed. This means that when a new writing to the framebuffer for changing the color of a block, executing an OUT instructions to update the Hi Address will not modify the Lo Address, but the Lo Address will remain whatever the same as it was previously set to. This creates the potential to affect an unintended block. The framebuffer should not save the color data until after all 3 OUT instructions have been executed. To achieve this, the framebuffer includes a write enable bit (we) that the RAT WRAPPER will manipulate. The VGA driver will be connected in the RAT WRAPPER in a manner so that the data will only be written to the framebuffer when Color Data is output. This means that the address (Hi Address and Lo Address) will have to be output first. The RAT WRAPPER will set the write enable (we) bit high for 1 clock cycle after an OUT to the Color Data PORT ID so that color data will be saved to the address and not overwrite any other addresses when future OUT instructions to new Hi or Lo Addresses are performed.

Reading from the VGA Driver

The VGA screen can contains a significant amount of data for the screen. The 80x60 individual blocks is 4800 locations. No internal memory in the RAT can contain 4800 items. This can make keeping the state of the program difficult to save in the RAT MCU. For example, if a maze program would use the VGA driver, the maze could be created programmatically by writing wall blocks as a different color to the VGA driver. However, those wall locations could not all be easily saved in the internal memory of the RAT MCU to use when trying to determine how the user can properly move throughout the maze. The data is saved in the framebuffer though, so if the RAT MCU was able to read the data back from the framebuffer, it would not need to be saved internally. Keeping with the maze example, the RAT MCU could read the current color at an address in the framebuffer to determine if the user's move is valid or if there is a wall blocking the move. The VGA driver provides this capability with RD output. RD is the 8-bit color data that is currently saved in the framebuffer for the current address (wa1). The RD will be routed in the RAT WRAPPER as an input with a PORT_ID value VGA_READ. So any address in the framebuffer can be read by setting an address with 2 OUT commands to the same Hi and Lo Address PORT_IDs followed by an IN command from the VGA_READ PORT_ID. The OUT commands to the Hi and Lo Address will not change any data in the framebuffer because the data is only saved with an OUT to Color Data.