VGA output using a 36-pin STM32

By Artekit(https://www.artekit.eu/author/admin/)
December 8, 2012(https://www.artekit.eu/2012/12/08/)
56 Comments(https://www.artekit.eu/vga-output-using-a-36-pin-stm32/#comments)

Thinking about old video game consoles and arcade machines (very old, like those in the 70's/80's) it came to our minds what can be done today using very low-cost microprocessors. Generally, these microprocessors weren't even created to do this task, so the challenge began, and we started to think the way to output video to a screen with few or no external components at all.

We have picked a 36-pin, 72 MHz STM32 (STM32F103T8U6), fast enough to generate monochrome video synchronism and dot signals. We use a couple of timers and the SPI (this way the refresh of the frame buffer is done automatically). And the final result is a pretty decent monochrome VGA output with 400 x 200 dots resolution.

Click here to download the complete source code (/resources/blog/artekit_vga.zip).

The project is in KEIL uVision format. You can download the KEIL uVision evaluation version from ***www.keil.com*** **(http://www.keil.com)**

The list of materials:

- A board with a STM32F103T8U6 or similar. We use the **AK-STM32-LKIT (/products/devboards/ak-stm32-lkit/)**.
- A female VGA connector (DB15).

Even if the frame buffer is 400×200 pixels length, the output resolution is 800×600 at 56Hz. We will be painting every horizontal dot twice, and every line will be repeated 3 times. This way we fill the entire screen.

Another reason we have chosen 800×600 @ 56Hz is because of the pixel clock: this resolution uses a 36 MHz pixel clock, that is a multiple of 72Mhz, the frequency of the STM32. Since we will be generating the pixels signal with the SPI, we can divide the STM32

clock with the SPI preescaler to get a 18MHz pixel clock, and paint every pixel twice. The SPI MOSI line will stay high or low twice the time needed to output a single pixel for a 800 pixels horizontal resolution.

(https://www.artekit.eu/wp-content/uploads/vga-screen.png)

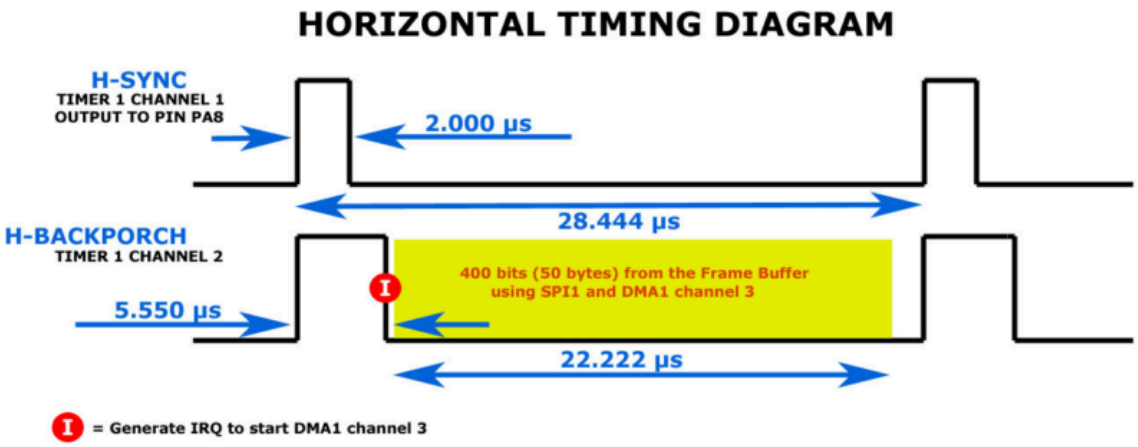
The frame buffer is composed of an array of 52×200 bytes. 50 x 8 = 400 pixels (every bit is a pixel). The two remaining bytes will simulate the blanking interval for every line.

```
1  #define VID_VSIZE 200
2  #define VID_HSIZE 50
3
4  __align(4) u8 fb[VID_VSIZE][VID_HSIZE+2];
```

Everything we write in this piece of RAM will be output directly to the screen without intervention of the application: the DMA is set to automatically read from the frame buffer and output the values to the SPI MOSI pin.

The horizontal synchronism

The horizontal synchronism signal and the back porch time are generated using the TIM1 timer, channels 1 and 2 (respectively). The TIM1 channel 1 is connected to the pin PA8.



(https://www.artekit.eu/wp-content/uploads/VGA-H1.png)

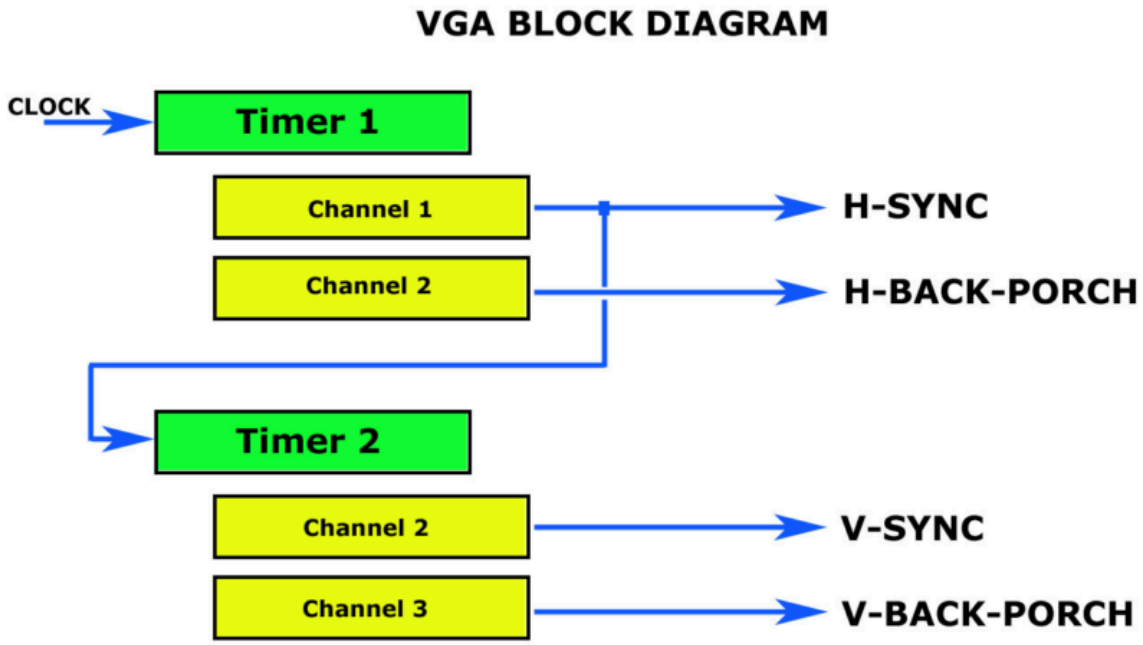
The H-SYNC timer 1 channel 1 (pin PA8) will actually generate the horizontal synchronism that the monitor will receive.

The H-BACKPORCH timer 1 channel 2 signal is calculated from the sum of the horizontal synchronism time and the back porch time. This timer will generate an interrupt that will be used to fire the DMA request to start sending pixels through the SPI.

This is repeated for every line in the frame buffer.

The vertical synchronism

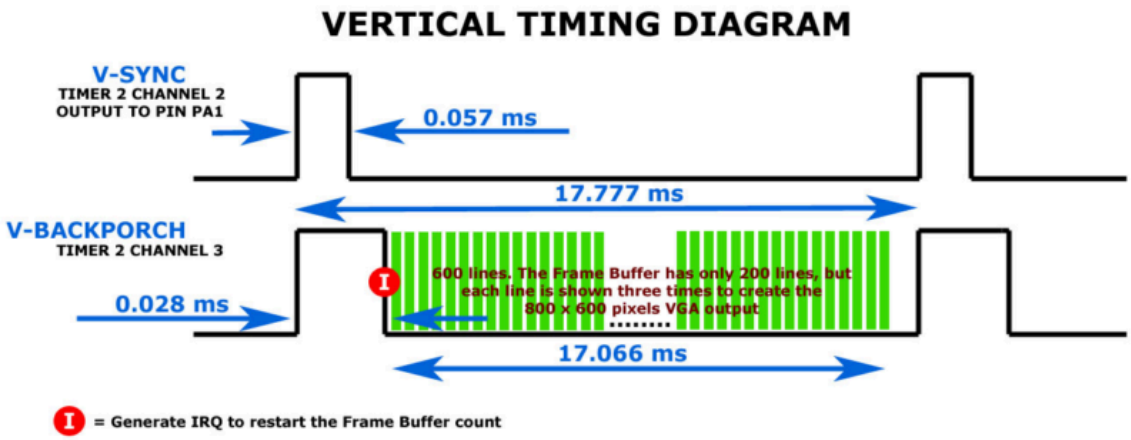
The vertical synchronism is generated using the TIM2 timer, but in slave mode. The TIM2 timer counts the H-SYNC pulses generated by its master, the TIM1 timer.



(https://www.artekit.eu/wp-content/uploads/vga-timers.png)

The TIM2 timer channel 2 outputs the V-SYNC pulse through pin PA1.

The TIM2 timer channel 3 will trigger an interrupt when the timer counter reaches the sum of the V-SYNC and vertical back porch time. This interrupt will set a variable indicating that the scanning is within a valid frame and the DMA can start sending pixels to the screen.



(https://www.artekit.eu/wp-content/uploads/VGA-V.png)

Pixel generation

Pixels are generated using the SPI MOSI pin (PA7). The timer TIM1 channel 2 generates an interrupt that will enable DMA TX requests to the SPI. The DMA will read a line from the frame buffer and will put the values in the SPI DR register.

The DMA is set to generate an interrupt after a single line is sent, where the line number is incremented. Since we are sending each line three times, we will increment a counter in this interrupt. When the three lines have been sent, we set the DMA pointer to the next line in the frame buffer.

When all the lines have been sent, the DMA cycle is disabled until the next valid frame interrupt (TIM2 channel 3).

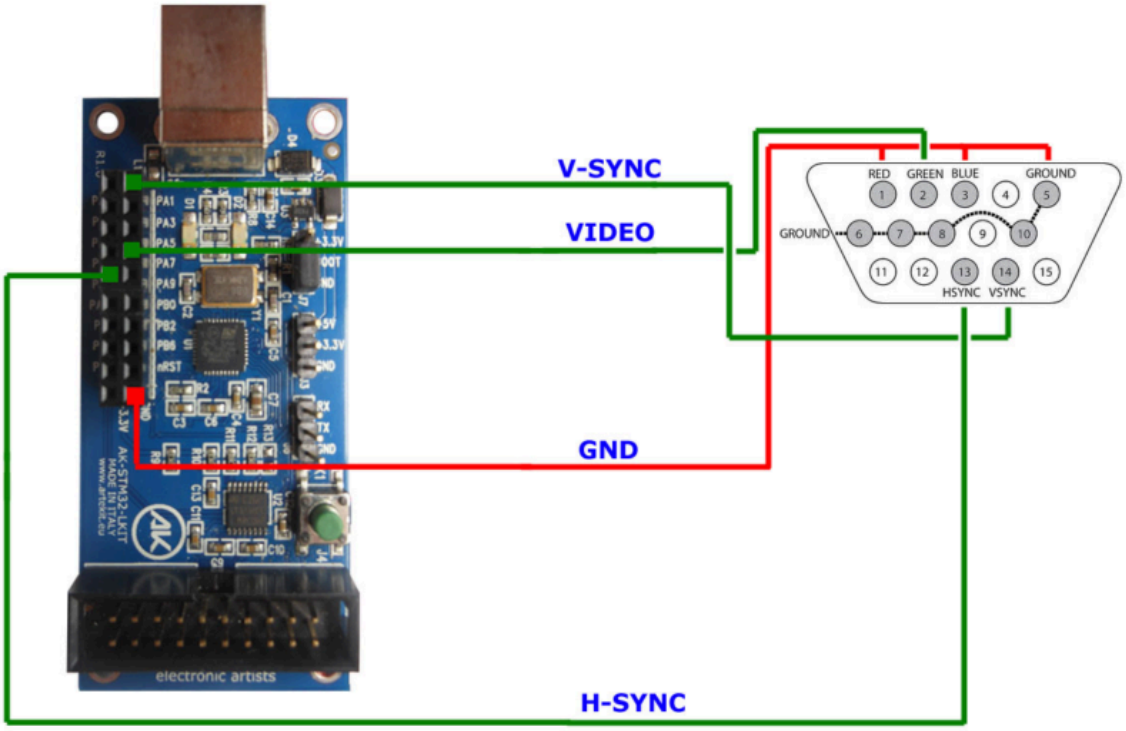
Connections

To use this example you will only need some wires and a female VGA connector.

The VGA standard says that the output signals should be 0,7V to 1V, so you may want to put a voltage divider in the pixel line (serial 68 ohm resistor and a 33 ohm resistor to ground, better with a 47pF in parallel with the 68 ohm resistor). We have tested a couple of LCD monitors without the divider and it went just fine.

Note that the pin layout is referred to the **AK-STM32-LKIT (/products/devboards/ak-stm32-lkit/)** expansion connector, but the pin names are valid for any STM32. Check your chosen STM32 datasheet to see if the timers and SPI pins matches the design.

Note: we use the green color (pin 2 of the VGA connector) to emulate the old style green phosphor monitors, but you can use another color combination using the RED/GREEN/BLUE DB15 pins. It is possible to create up to 8 color combination.



(https://www.artekit.eu/wp-content/uploads/vga-connections.png)

AK-STM32-LKIT pin	VGA connector pin	Description
PA1	Pin 14	Vertical sync
PA7	Pin 2	Green
PA8	Pin 13	Horizontal sync
GND	Pin 5	Ground

Conclusion

We have created a VGA controller using a very low cost microprocessor/development board. The method used it's certainly not the only way to do it, but this one uses no external components besides a VGA connector.

If you are using this example with a bigger STM32, you can try to use double buffering and to write to the frame buffer while the DMA is disabled, to avoid tearing.

You may **download the source code (/resources/blog/artekit_vga.zip)** for this project. There you will also find a utility library to draw lines, points, circles, bitmaps, character generation, bit blit and more.

In the next blog entry we will be implementing a video game using this VGA example.

Have fun!

The Artekit Team.
