FPGA with Multiple VGA Inputs and Outputs

Research

TDT4295 Computer Project Autumn 2022
Graphics Group

September 9, 2022

Abstract

The goal is to build a board with two video inputs, and two video outputs, and let the user control how the inputs are used to generate outputs. An MCU handles the user interface, while an FPGA accelerates the video input reading and video output writing. It should be able to output effects such as picture-in-picture, split-screen, and chroma keying, independently to both outputs. As an extra piece of functionality, we could allow saving a frame to an SD card, and later use the frame as an input.

Contents

1 Capabilities we want **2**

2 Components **3**

3 VGA **4**
3.1 Analog-to-Digital 5
3.2 Digital-to-Analog 5
3.3 VGA timings 5

4 Frame buffers **6**
4.1 What needs to be buffered? 6
4.2 Frame buffer sizes 6

5 External SRAM **8**
5.1 What about DRAM? 9

6 Bus **9**

7 Clock speeds **9**

8 User interface **10**

9 SD card **10**

10 Conclusion **10**

1 Capabilities we want

We want to have two outputs, dubbed output C and D. For each output, we want to be able to configure what to show. We can display one ore more of the following:

- input A
- input B
- Image from SD card

It's reasonable to set an upper limit of two "layers".

Rendering one layer on top of another is not much fun, so we need ways of combining them.

- Scale the top layer down to 50%, place it somewhere. (Picture-in-picture)
- Scale both layers down to 50%, place them side by side (split screen)
- Chroma key the top layer
- Multiply or add the colors
- Transition from top to bottom layer

- Fade from one input to another (similar to transition from top to bottom)

There are lots of options, so we might want to use an LCD screen to pick, and not restrict ourselves to a set of buttons we define on the PCB.

Regarding the picture-in-picture and side-by-side modes, we can hard code in a few placement options (center, corners, edges). Or, we could instead make scaling and translation independent of effects. The scaling could have more options than just 50% size, and translations could be any integer, allowing us to slide layers around, while also applying effects such as chroma key.

Finally, it should be possible to copy the configuration of one output onto the other output. This lets "the director" have a preview before moving the configuration to the other screen.

2 Components

- MCU: EMF32GG - Reference Manual on BB
  - High Performance 32-bit processor @ up to 48 MHz
  - 128 KB RAM
  - Up to 90 General Purpose I/O pins
- FPGA: ARTIX-7 XC7A35T (on the dev board). See the Series Overview.
  - 33 280 logic cells
  - 50 x 36 Kib = 1 800 Kibibits of block RAM (dual-port)
  - 250 Max User I/O
  - 4 GTP transceivers (maybe useful, not sure)

  On the PCBs we make, we use the ARTIX-7 A100
  - 135 x 36 Kib = 4 860 Kibibits of block RAM (dual-port)

  - 300 Max User I/O

- SPI Bus (see Section 6)
- Clock circuit (see Section 7)
- Extra SRAM (see Section 5)
- VGA ports. 2x in and 2x out (see Section 3)
- Power circuit
- Buttons, LEDs and LCD for UI (see Section 8)
- SD card reader (see Section 9)

3 VGA

Note: "VGA" often refers to the display hardware used by IBM PS/2, or the 640x480 resolution it had. We care about the actual video signal being sent over the wire.

Using the same connector, a lot of possible display resolutions have been added. See Extended Graphics Array.

A monitor can communicate with the video source to say what outputs it supports, but our output can ignore this, and just send data over the wire. This is what Ben Eater does in his graphics card.

For our VGA inputs, it might be a good idea to implement the monitor description protocol, such that the devices providing inputs give us video in our native frame buffer size, with known timings.

VGA has analog color signals, so we must somehow make it digital, process it, and turn it back into analog output.

3.1 Analog-to-Digital

The ARTIX-7 FPGA contains a 12-bit Analog-to-digital converter (called XADC). It can take up to 17 analog inputs, by multiplexing. We only need 6 (2 inputs·3 colors). However, the ADC has a bandwidth of 1 million samples per second. That is not enough.

The proper ADC to use would be a dedicated one. For instance TI's TVP7002. That should support as high as 1080p, because the ADC supports higher than the 148.5 MHz necessary for 1080p@60.

3.2 Digital-to-Analog

Good resources have been written on making DACs. Eli Gladman designed the rgbs888 which uses an R-2R DAC. No matter what bit depth we choose, this is probably a good design. Ben Eater also used resistors to create a DAC for VGA.

The VGA color outputs should be between 0V - 0.7V for full dark to full brightness. This range is easy to achieve using R-2R.

If we want to use a proper DAC, which simplifies all of the analog sections and should reduce size, we can use e.g. TI's THS8136. The advantage is it should make our life simpler, but we can't cheat by changing resistor values.

3.3 VGA timings

By outputting signals on the red, green and blue channels, as well as h_sync and v_sync, we can send an image to the monitor. To pick a resolution and get the timings required, see VGA timing. There is also this blog article here that details the most common resolutions up to 1080p.

A lower bound is the standard VGA 640 x 480 @ 60Hz. The website tells us that this requires a pixel frequency of 25.175 MHz. This means two things: The VGA part of the FPGA should have a clock speed divisible by this, and we have ~ 39.72 ns to process each pixel.

Another, slightly higher fidelity option in SVGA, 800x600 @ 60Hz. The nice thing about this is the pixel frequency of 40 MHz. This gives exactly 25 ns of pixel time.

The higher resolutions use HDMI, which would save us the use of ADCs and DACs, but have other implementation challenges. Also note that 720p and 1080p have pixel timings of respectively 13.5 ns and 6.7 ns.

4 Frame buffers

We need to pick a size for our inputs. It should be standard, and not too big. One concern is the size of RAM. The FPGA dev board has a smaller built in RAM than the FPGA we will use on the PCB, but we can also add external SRAM to get more space.

Another concern is the throughput, both over the VGA ports, and to/from the memory. We need to pick a framebuffer size small enough to get all the data in and out every frame.

How many bits of color depth we store will affect both memory requirements and how advanced the output DAC needs to be. Depending on the resolution, we have different amounts of time to process each pixel.

4.1 What needs to be buffered?

At least one input signal must be buffered, since the inputs won't be in phase. To be able to translate both inputs, we must buffer them both.

Finally, the image saving / displaying might require storing the image in RAM, depending on the speed of the SD card operations. If we want to save on chips, we can require that loaded images always replace one of the input VGAs, thus repurposing the same frame buffer.

4.2 Frame buffer sizes

As a lower bound, we can try using the original VGA mode. 640x480 pixels, with an 24-bit color depth, gives a size of

$$640 \cdot 480 \cdot \frac{3 \text{ colors}}{\text{pixel}} \cdot \frac{1 \text{ byte}}{\text{color}} = 900\text{KiB}$$

By reducing the color depth, which is a sensible thing to do, we can get more room.

By giving each channel 5 bits, a pixel fits in 2 bytes. The green channel can even get 6! This gives a memory usage of

$$640 \cdot 480 \cdot \frac{2 \text{ bytes}}{\text{pixel}} = 600\text{KiB}$$

To give an idea of what a 640x480 image with 16-bit color depth looks like, see Figure 1. For the same images with a naive 8-bit color palette, see Figure 2

Figure 1: 640x480 with (5,6,5)-bit color depth. Images taken from Unsplash.

Figure 2: 640x480 with (2,3,3)-bit color depth. Images taken from Unsplash.

For reference, a 800 x 600 image with 16-bit colors takes up 937.5 KiB

If we want to design for the very highest resolution we could go for, 1080p with 24b colour is 6075 KiB. At 60Hz that is a throughput of $\sim$ 356 MiB/s.

5 External SRAM

As the framebuffer section showed, a single framebuffer will be at least 600KiB. The onboard block RAM in the dev board FPGA is 1800Kib = 225KiB. The final FPGA has 607 KiB of RAM, which is just enough for one framebuffer. It thus seems sensible to add an external SRAM chip. There are guides that show examples of using SRAM from Verilog on an FPGA.

It's worth noting that the async SRAM they used doesn't take a clock signal. Instead it takes some time between the address is stable, and the data is ready to be read.

Table 1 shows how much time you have per pixel, and the minimum throughput required to read the buffer 60 times a second.

Resolution	Pixel time	Min. throughput per buffer
VGA 640 x 480 8b	39.72 ns	140.6 Mib/s
VGA 640 x 480 16b	39.72 ns	281.3 Mib/s
SVGA 800 x 600 16b	25ns	439.5 Mib/s
720p 24b	13.4ns	1.27 Gib/s
1080p 24b	6.7ns	2.85 Gib/s

Table 1: Pixel times and required througput to output different frame buffers

Looking at Mouser, if we use the IS64WV51216EEBLL-10CT2LA3 512Ki x 16bits SRAM with 10ns read cycle, we can fit 3 read/write operations per pixel in the VGA output. Recall that a 640x480 @ 60Hz gives us $\sim$ 39.72 ns per pixel.

SVGA, aka 800x600, only gives us 25 ns per pixel. We tried looking for SRAM chips with 32 bit data buses, which would allow us to read/write two pixels per access. One such chip is GS881Z32CGD-150I, costing 160kr. With 256K 32-bit rows, it can store an entire SVGA frame buffer. By storing pixels two at a time, we get 50 ns per two pixels. This SRAM is **synchronous**, which means we give it a clock. That clock can have a period down to 7.5ns. It should then be very safe to operate with an e.g. 10ns clock.

We will need to read the SRAM data sheet thoroughly, to make sure we can actually access the RAM like we think. The data is input and output one cycle after the address!

5.1 What about DRAM?

DRAM requires more work, such as a clock and refreshing, but it can also go fast for cheap.

The Arty dev boards we have do include a DDR3 chip and it looks like it shouldn't be tremendously difficult to use if we use Xilinx's IP. Digilent has a guide here. **However**, the throughput of this DDR3 is 667Mbps, which is enough to read and write to one 640x480 16b frame buffer, but not two.

On Mouser we find DRAM chips like the IS42S16100H-6TLI for only 19kr, which has room for 1Mi 16b pixels, and an access time of 6ns. Its access patterns are however much more complicated, with burst reads and rowns and columns and stuff.

6 Bus

The SPI interface is a synchronous serial databus. It has a clock signal from the master, and full duplex data lines. We can also add *slave select* lines to connect several components. The MCU will be the master, with the FPGA, SD card and LCD as slaves. SD cards are already built around SPI, so that's perfect.

7 Clock speeds

The Cortex-M3 in the MCU can go up to 48MHz.

Depending on the length of the longest critical path in our FPGA design, we can run it at 100MHz - 300MHz, was a ballpark range provided by the TA. We could try to pipeline the operations to get it even faster.

We will need to handle a number of FIFOs on the FPGA, because we have a number of clock domains. VGA runs at wildly different frequencies depending on resolution and framerate and the FPGA logic will have some (quite high) frequency. Then, each VGA output controller will also need to run in a separate domain, and for the

various external memories those will run at a much higher frequency than the FPGA logic because otherwise the memories' throughput can't be fully utilised. Fortunately however we can generate all the clocks on the FPGA side off a single clock generator circuit and PLLs.

A final thing that affects clock speeds is the amount of electromagnetic interference on the PCB. See this article for tips to reduce EMI on our PCB.

8 User interface

The exact interface will have to be based on the features we will end up supporting, but we will of course need to pick a UI before designing the PCB.

9 SD card

Read about SD card over SPI here.

10 Conclusion

Considering the different options for video resolution, color depth, frame buffer storage and processing, we can try to come up with a design we think might work.

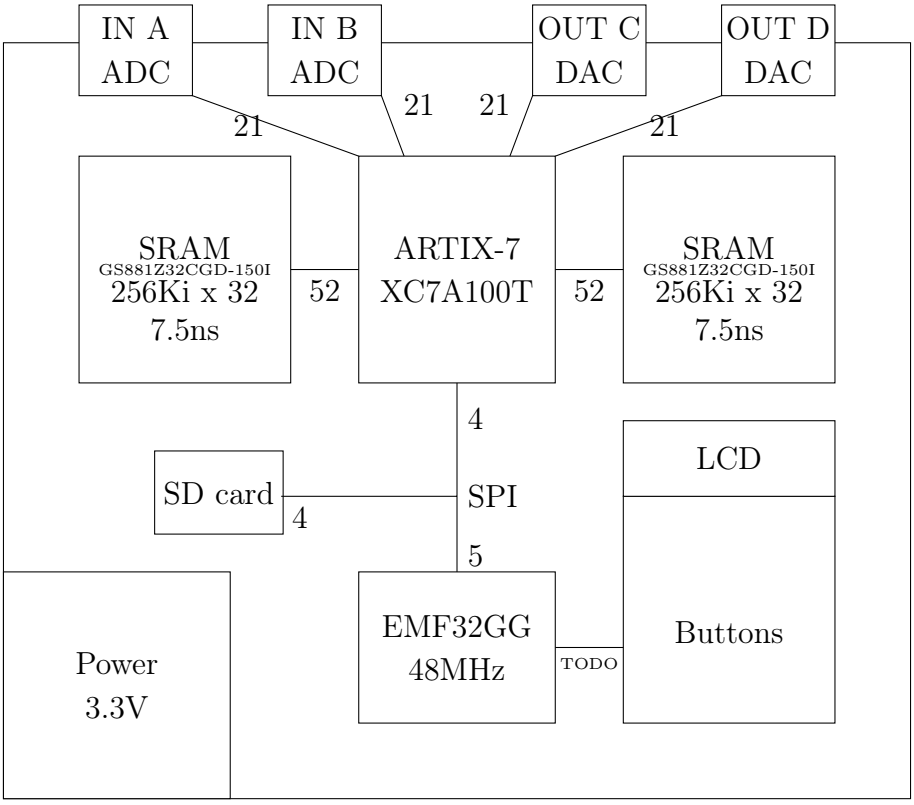

Figure 3: Overview of which components are connected, and how many pins the connections take

.