Porting Apple2fpga to Xilinx Zynq

Retrocomputing Revisited

Feng Zhou, July 2021

Retro-computing and retro-gaming has become an area of interest for computer scientists, hobbyists [1] [2] [3] and even consumers [4]. Among the projects, the approach of FPGA-based emulation of old computers, game consoles or arcade machines, has shown advantages like more accurate timing, the ability to work directly with vintage peripherals, and etc. The most popular project in this area, MiSTer [5], supports only Intel/Altera FPGA boards, not Xilinx ones. Therefore it would be interesting to explore if we can do some emulation on Xilinx FPGAs, and have some fun in the process.

Instead of working directly with MiSTer, which is a large project, I chose to port a much smaller project, Stephen A. Edwards's Apple2fpga [6], to Xilinx's platform. Here I document my experience in design and implementing the port. It shows that it could be done, along with some new features, at low cost and good quality.

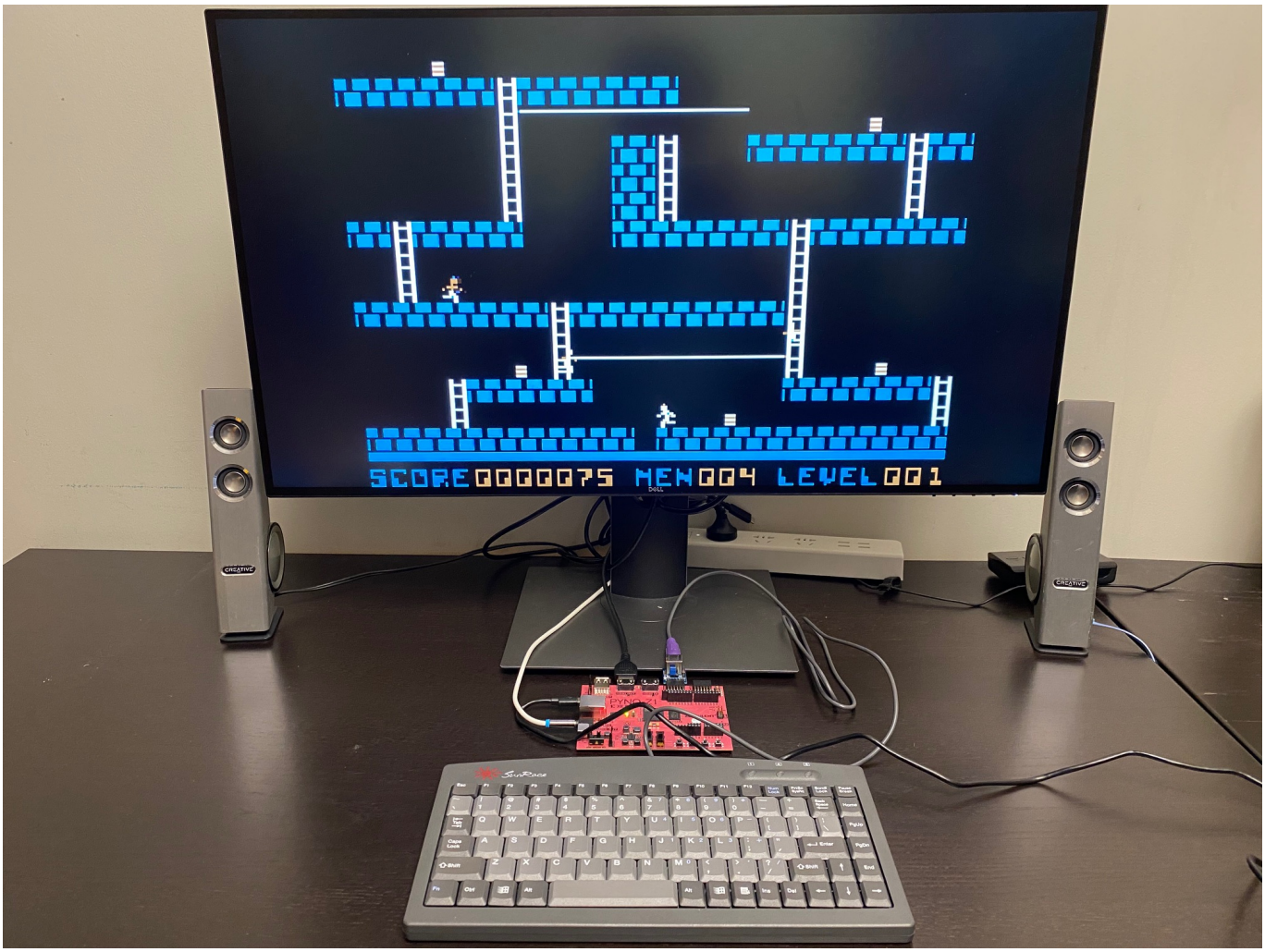
Apple II's legacy

The Apple II's story is well-known. Two young Steves, one tech wizard (Steve Wozniak) and one business genius (Steve Jobs), saw the coming wave of microcomputers / home computers, and started Apple to seize the opportunity. First they created a semi-successful first generation product, Apple I, in 1976. Then they went on to produce the true star product, Apple II, the next year. It is one of the world's first highly successful microcomputers. Through its over 15 years run, the Apple II series sold over 5 million units [7].

The Apple II even had an impact in China. I grew up in a small town in east China, 200 km from Shanghai. In 1988, at grade 4, I had my first experience with computers at school. I remember the newly-built computer lab had about 15 brand-new computers. All but one of the computers are LASER 310's from VTech, a Hongkong company. The other, more *advanced* computer, is the Apple IIe. I was lucky enough to get a fair amount of time on the Apple (and more time on the LASERs obviously) and it literally set in motion my interest and career in computing.

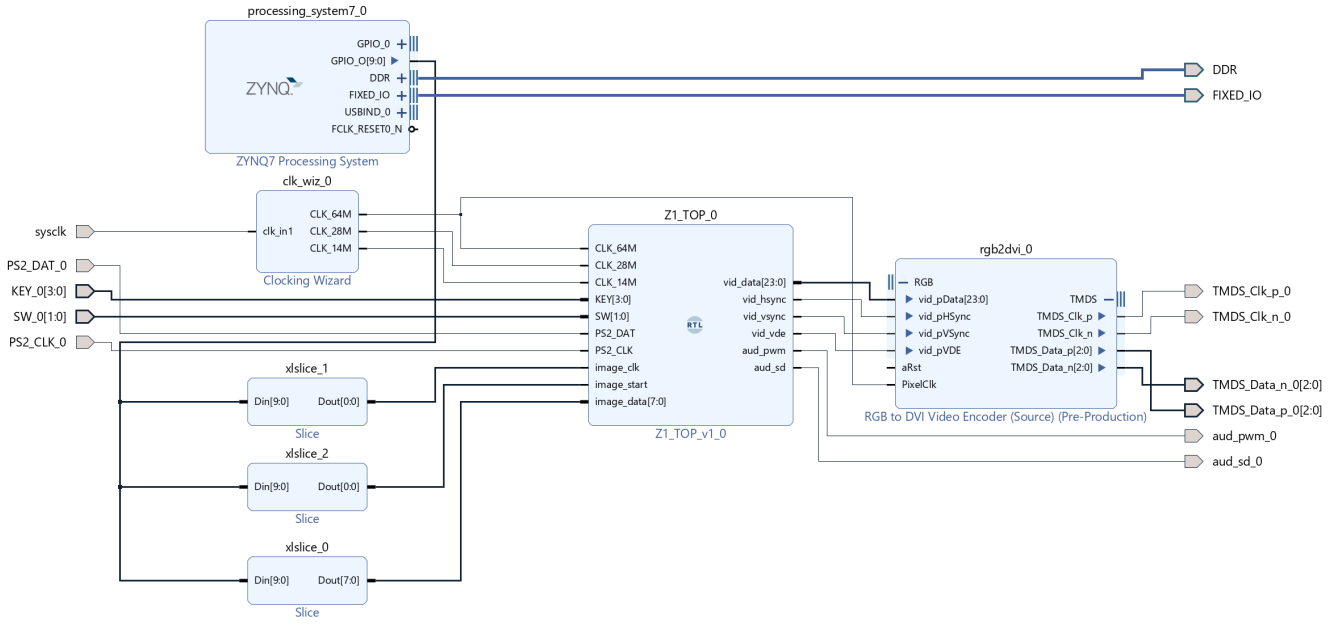
Porting Apple2fpga from DE2 to PYNQ-Z1

This is the final setup of PYNQ-Z1 emulated Apple II+.



The main chip of PYNQ-Z1 is Xilinx ZYNQ XC7Z020-1CLG400C, a so-called FPGA SoC (System-on-Chip). In addition to FPGA logic, it also contains a dual-core ARM A9 CPU, hence the SoC name.

Here is the block design of our code.



The bulk of the system is inside the RTL module Z1_TOP_0 (middle). The ZYNQ7 Processing System (top-left) is the CPU in the SoC (PS in Xilinx terms). The ARM CPU is connected to the Z1_TOP_0 module with GPIO pins, to

deliver disk images as we will discuss later. The video output out of Z1_TOP_0 goes through a standard RGB to DVI Video Encoder module to become valid DVI/HDMI signals.

Most of the porting is quite straight-forward. Only two parts are significantly different, the video system and disk emulation. Let us go over the main subsystems of Apple II+.

The Clocks, CPU and Memory

At the core of Apple II+ lies the clock generation circuits and 6502 CPU. These parts remain mostly unchanged. We retain the timing generation logic of Apple2fpga, generating every clock from a 14.31818 Mhz master clock. And the 8-bit 6502 CPU runs at about 1 Mhz, the classic Apple II speed.

Apple2fpga emulate the 48 KB Apple II+ RAM through the DE2 on-board SRAM. PYNQ-Z1 does not have SRAM. It instead provides 512MB of DDR3 DRAM, and 630 KB of Block RAM (BRAM) on-chip. The BRAM is much easier to program than DDR, and large enough for us. Therefore the 48 KB of Apple II RAM is emulated with BRAM in our design.

The DVI ("HDMI") Signal Generator

The video part is where the real work began. The Apple II video system is very primitive by today's standards. Text mode is 40x24 characters. The HIRES (high-resolution) mode is 280x192 with 4 colors, in addition to black and white. Apple2fpga / DE2 basically "upscales" this to double resolution 560x384 and physically outputs the signal through the analog VGA port. PYNQ-Z1 does not have a VGA port, and even VGA monitors are hard to find now. It does have HDMI ports for both input and output. However in bare metal FPGA style, the ports are directly connected to FPGA circuits, and the programmer is tasked with providing every bit of the digital video signal. Naturally it took some work to get it working.

Xilinx provides a suite of IPs (packaged FPGA modules) to help the programmer drive the video ports. These include a timing generator (Video Timing Controller), a bridge between "video protocol" and HDMI ports (AXI4-Stream to Video Out) and etc. The idea is to let the programmer use a separate video streaming protocol (on top of the ARM-standard AXI4-Stream interface) to make video processing easier, without the need to adhere to strict video timing requirements. However, after some experimentation I found that this extra level of abstraction probably introduces more complexity for us, and also more buffering and latencies than necessary.

Our solution did not use the Xilinx video IPs. As the block design shows, video output from our code goes directly to `rgb2dvi`, a standard (provided by the board designer Digilent) module to encode the video and generate TMDS signals required by DVI/HDMI. The video data from Z1_TOP_0 should already obey correct timings. The design is actually similar to Apple2fpga's original VGA line doubler. However instead of doubling the resolution, we actually tripled the resolution, from 280x192 to 840x576. It is necessary because the DVI/HDMI interface actually only supports pixel clocks of at least 40 Mhz, and the double resolution has a pixel clock lower than that.

The Apple II outputs 60fps NTSC video. During the process, mainly 3 signals represent the whole video signal, VIDEO (the 1-bit video signal), HBL (horizontal blank) and VBL (vertical blank). VIDEO is a bit-stream at 14.31818 Mhz, scanning the screen line-by-line. And 2 bits of VIDEO corresponds to 1 dot on screen in HIRES. As discussed in [6], the line doubler/tripler approach works by buffering one or two lines of pixels, and stays in sync with the whole screen-scanning process. Given the Apple II's pixel clock of 14.31818 / 2 = 7.15909 Mhz, our line-tripled video thus has pixel clock of 7.15909 * 9 = 64.43181 Mhz, as one pixel becomes 3x3 and 9

pixels. This is why we need the 64 Mhz clock from our clock source. And our code reads 2-bits of VIDEO data in every 9 cycles of the 64 Mhz clock and puts them into our line buffer.

Note that although physically the ports are HDMI ports, we do not generate a proper *HDMI* signal. It is a DVI signal that goes through the HDMI port, also without audio.

Color Decoding

There is much discussion about decoding / demodulation of Apple II colors online, for example [8] and [9]. The Apple II color system is quite peculiar, as it makes use of the different phases of the NTSC video to generate signals that would be recognized as different colors by the TV/monitor. The interested reader is referred to [6] for a discussion of how to decode such colors, the approach we inherit here. The only difference is the issue of clocking. The original line doubler has a clock of 28 Mhz, exactly twice of the VIDEO signals 14 Mhz. Our line tripler needs 64 Mhz, which is 4.5 times of VIDEO's 14 Mhz. Therefore as a fact of life, the timing became more complicated.

The Disk Emulator and Host (ARM-side) Program

The Apple II+ has one or two 5.25-in 140KB floppy drives. Apple2fpga emulates one drive that is read-only. We basically provide the same functionality with different implementations. In Apple2fpga, a track of raw disk data (6656 bytes) is stored in RAM and the whole disk image is on SD Card, accessed through an SPI interface. The PYNQ-Z1 unfortunately does not have a direct interface from FPGA to the on-board MicroSD Card, which is connected exclusively to the ARM CPU. Therefore we would naturally need help from the CPU to access our disk image.

Our solution of the disk emulator consists of three parts. One is the FPGA module, which stores not one track, but the whole disk image, in on-chip BRAM. Remember the chip has 630 KB of total BRAM, more than enough for emulating both the RAM and holding a disk image, 227.5 KB in raw(nibblized) form. The second part is the interface between the ARM CPU and FPGA (PL in Xilinx terms). We simply use 10 configured GPIO pins for simplicity and portability, although Xilinx's recommended interface of AXI4 would certainly be another way.

The third and last part is the program running on the ARM CPU. We have a choice of running without a real operating system ("baremetal"), or running on Linux. For simplicity we chose baremetal, with "FATFS" support. It allowed us to read image files from a FAT32 file system on the MicroSD Card. And the user can choose which image to send to the emulator, with attached USB-Serial console. We found this more convenient than Apple2fpga's SDCard-without-file-system approach.

The Sound Interface

The Apple II+ sound system is very simple. A one-bit signal controlled by accessing the "soft switch" memory address at $C030. There is no MIDI, no PCM playback at all, just simple square waves driven by the one-bit signal. For Apple2fpga, driving the audio chip on board DE2 was actually a quite convoluted process involving I2C and tricky timing. For us, the PYNQ-Z1 has a mono Pulse-Width-Modulation (PWM) audio output, which is very simple to drive from an FPGA. In short, PWM is a simple way to simulate analog signals with digital ones. With PWM, the level of analog signal is represented by the duty cycle of the digital signal. Our module simply generates the PWM signal with either close-to-zero duty cycle (when SPEAKER is 0), and close-to-one duty cycle (when SPEAKER is 1).

Quick TODOs

A few things come to mind,

- MiSTer-style OSD interface to choose images and settings, so that the user does not need a USB-Serial console.
- Read-Write disk emulation.
- Apple IIe emulation.
- USB keyboard support.

References

[1] The IBM1401 Demo Lab at Computer History Museum.

[2] Light Years Ahead | The 1969 Apollo Guidance Computer, talk by Robert Wills, 2020

[3] The Internet Archive Software Library.

[4] NES Classic Edition.

[5] The MiSTer Wiki.

[6] Retrocomputing on an FPGA, Stephen A. Edwards, 2007

[7] Wikipedia: Apple II Series

[8] Apple II Colors

[9] HIRES Graphics on Apple II