How to Use Free Software in FPGA Embedded Designs

Looking to bring the Free Software/Open Source Software movement into the realm of hardware design, Andrey Filippov built a low-cost, high-performance network camera using a Spartan-IIE FPGA and free WebPACK development software from Xilinx.

by Andrey Filippov, Ph.D.
President
Elphel, Inc.
andrey@elphel.com

Just recently, Gordon Moore "renewed" his law for another decade, predicting that the density of integrated circuits will continue to double every 18 months. At this daunting pace, how is it possible to keep up with the ever-growing challenges of electronics design? One answer is to effectively reuse what others have done before, so you can focus on the really new issues.

The growing number of successful adoptions of the Free Software and Open Source Software (FS/OSS) development models has proved the effective reuse of software in applications development.

Xilinx programmable logic devices represent a "frontier" between the hardware and the software worlds. They offer a unique opportunity to apply FS/OSS solutions to hardware design.

I strongly believe that the potential for successful applications of the FS/OSS models to hardware design is even higher than for software itself – you can always make a profit by selling the actual hardware. In this article I'll show how this approach helped me to create a high-speed, high-resolution network camera (a class of cameras that do not need additional computers to serve digital images and video over the LAN or the Internet).

Background

I had my first experience with GNU/Linux less than two years ago when I realized I would have to program the network cameras I designed around the ETRAX100LX processor. The processor, made by Axis Communications AB, was optimized for running GNU/Linux. Having the experience of designing many microprocessor-based systems that were mostly programmed in assembly languages, I was really amazed that in just a couple of weeks my camera was able to serve JPEG images over a LAN using HTTP protocol. It was possible to control acquisition and view images using any standard web browser.

That Model 303 camera combined software from multiple sources – GNU/Linux itself, specific code for the ETRAX platform, and many, many others. I just had to add specific drivers for my hardware and modify some of the existent programs. What impressed me the most was the ease of navigation in big, third-party software systems where I could find just the right place to insert small patches of code to modify functionality.

Being primarily an electronics designer, I started to think it was possible to make similar productivity enhancements to the hardware/FPGA part of the camera system.

Design Goals

Starting the design of a new camera – Model 313, shown in Figure 1 – I had the following goals in mind:

- **A high-performance, simple network camera** that fully supports the frame rate/resolution of megapixel CMOS sensors. That means 15 fps at 1280x1024 resolution (or proportionally, 60 fps at 640x512 resolution, for example).
- **Use a reconfigurable FPGA** for image acquisition/processing/compression – as opposed to two specialized ASIC chips and one-time-only programmable devices (as was used in the model 303 camera). That goal came from the following requirements to:
  - **Simplify the product development**. I estimated if that if I were to completely troubleshoot a one-time programmable FPGA by simulation, it would be too difficult and time consuming. On the other hand, the Xilinx Spartan™-IIE XS300E chip was five times higher (in gate count) than the biggest anti-fuse FPGA I was able to design. Having the option to switch between simulation and actual hardware testing proved to be much more efficient.
  - **Make the system flexible with upgradeable "hardware" algorithms** in the same way as it is done with software – this would significantly increase the lifespan of the product.
  - **Increase the number of possible product applications** by providing a customizable development platform.
- **Use free, downloadable development tools** so you could customize my product without spending thousands of dollars on software.

Selecting the Right FPGA

When I started to think about the new camera design, my only FPGA experience was with anti-fuse devices of up to 60K gates. So, I was open to consider different FPGAs that matched my design goals. (At that point, I was not even sure it was possible.)

Soon I came upon a good candidate. It was the largest (at that time) member of the Xilinx Spartan-IIE family – a 300K-gate XS300E.

To find out if the XS300E was capable of handling JPEG compression of 1.3 megapixel images at 15 fps, I "window-shopped" for commercial IPs that performed similar tasks. I was able to find some device utilization specs that indicated that I would have enough room to implement all the functions I needed: frame buffer SDRAM control, image fixed patter noise elimination, color space conversion, and CPU interfacing.

Figure 1 - Model 313 Elphel network camera

I did not use that IP, however, because it would prevent me from having an open source design. So, I just used the IP specs as a temporary substitute for my own lack of expertise in Xilinx devices.

Knowing that my project was doable in terms of hardware, I downloaded for free WebPACK™ 4.2i software from the Xilinx website to try it out. I wanted to see if there was anything else I was not aware of, at the moment, that would prevent me from following my plan.

To get some initial experience with both Xilinx devices and software, I read Xilinx Application Note XAPP610 and decided to try an 8x8 DCT core (needed for JPEG compression). The application note provided Verilog sources, which allowed me to synthesize, map, and place-and-route the design – but I had problems trying to simulate the design.

It turned out that the "lite" version of a third-party simulator bundled for free with the WebPACK software had limitations on design complexity. The most obvious problem was the 500-line limit on the source code. The XAPP610 DCT implementation alone was bigger, but I was able to try simulation by removing comment lines and combining multiple lines into one.

The "lite" simulator did run, but that workaround trick would not work for my complete project, so I had to forget about that simulator and use something different.

All the rest of the WebPACK software worked just fine for me. Eventually, I was able to utilize more than 98% of the chip's resources to meet all of my timing constraints.

System Architecture

Now that I knew I could fit my design to a Spartan-IIE, I switched to the schematic and PCB design. The Model 313 camera consists of two boards, shown in Figure 2. The small board has just the CMOS image sensor and closely related circuitry.

The main board consists of a 1.48 by 3.50 inch, four-layer PCB that has the following principal components:

- **32-bit, 100 MHz processor** (ETRAX100LX, Axis Communications) running a

Figure 2 - Main PCB of Model 313 camera

GNU/Linux operating system. The processor has multiple embedded peripherals, including a network MAC.

- **10/100 Mb Ethernet PHY** (BCM2521A4KPT, Broadcom).
- **16 MB (4Mx32) SDRAM** system memory (MT48LC4M32LFFC-8, Micron).
- **8MB (4Mx16) flash memory** (MT28F640J3FS-12, Micron) used to store GNU/Linux, applications, Web pages, configuration files, as well as bitstreams for the FPGA configuration.
- **Spartan-IIE, 300K-gate, reconfigurable FPGA** (Xilinx XC2S300E-6FT256) used to control image acquisition, processing, frame storage in SDRAM, image compression, and transfer to the system memory using the processor DMA channel. The FPGA is configured using JTAG pins connected to the general-purpose parallel port of the processor.
- **Image memory, 16 MB (8Mx16) SDRAM** (MT48LC8M16LFFF-8, Micron) connected directly to the FPGA so that accesses do not interfere with the system bus. This memory is dedicated to the image frame buffer that is used to store both the uncompressed image data as well as calculated coefficients for the on-the-fly FPN (fixed pattern noise) elimination (subtraction of the background frame and multiplying each pixel by its reverse sensitivity).
- **3-PLL programmable clock generator** (CY22393FC, Cypress). This part provides fixed frequencies that have to be available during system startup (20 MHz processor PLL input and 25 MHz for the network PHY). It also generates two programmable clocks that add extra flexibility for FPGA operation (in other words, it is possible to compare simulation results with the actual maximal operation frequency for any particular module).
- **IEEE 802.3af compliant power over LAN** uses an isolated DC-DC converter to supply 3.3V from the input 48 VDC. An additional converter provides 1.8V to the Spartan core.

FPGA Code

Most of the system functionality is implemented in the Xilinx Spartan-IIE FPGA. The code is written in Verilog HDL and is available for download at my Elphel website under the GNU/GPL (general public license) license. It is designed around a four-channel SDRAM controller that uses embedded block RAM modules as "ping-pong" buffers to provide quasi-simultaneous block access for the following data sources and receivers:

- Image data from the sensor, either processed or raw, one- or two-bytes per pixel, arranged as 256 (128) pixel lines
- Calibration data to the FPN elimination module prepared by software in advance, 128x16-bit blocks
- Data to the JPEG compressor, arranged as square blocks of 16x16 bytes

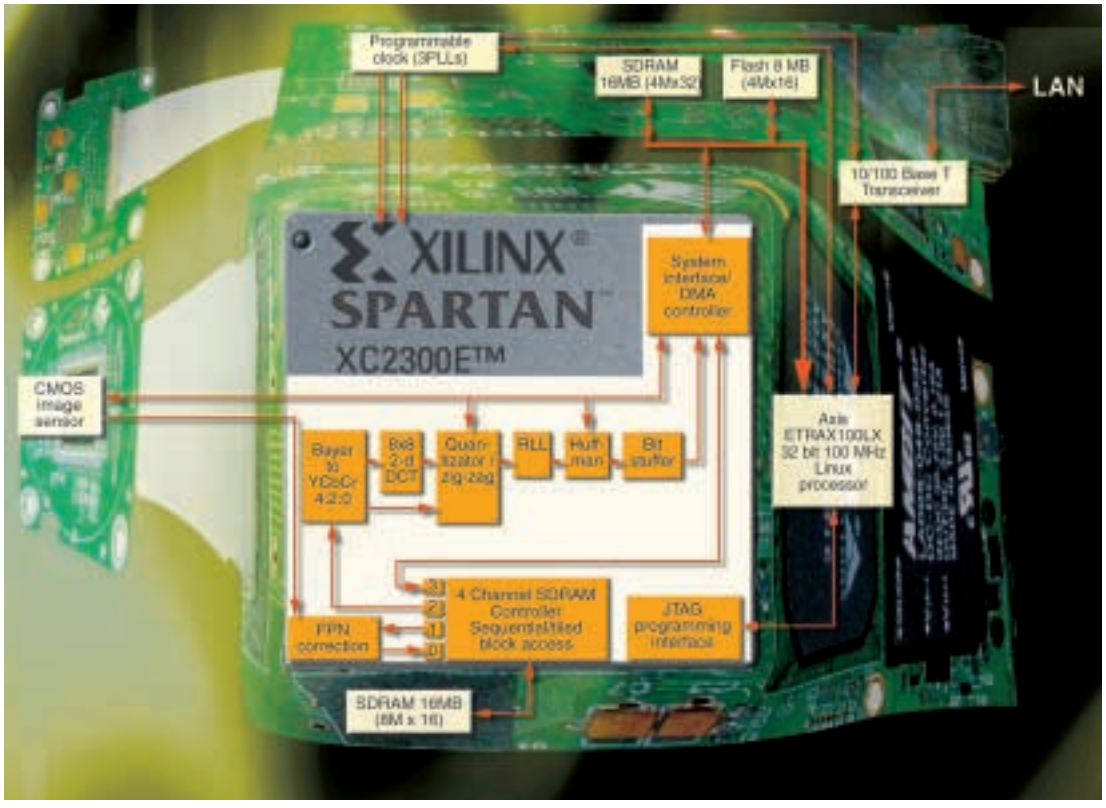

Figure 3 - The importance of the JPEG encoder in the Spartan-IIE dwarfs the other components on the PCB.

- CPU access to the SDRAM (normally used to read raw sensor data and write back the calibration data for the FPN elimination).

The JPEG encoder uses two-thirds of the FPGA resources, as shown in Figure 3. The encoder consists of the chain of the processing modules, some of which use block RAM for data buffering and table storage:

– Bayer-to-YCbCr converter

– 8x8 DCT based on the Xilinx XAP610, modified to provide block-asynchronous operation and to increase dynamic range

– Quantizator and zigzag encoder

– RLL encoder

– Huffman encoder

– Bit stuffer.

The output data is transferred to the system memory using the CPU DMA channel.

Results

Since the Elphel Model 313 reconfigurable, high-resolution network camera was first described in the LinuxDevices online magazine Dec. 3, 2002 (and Dec. 4, in Slashdot), many of the inquiries I received have been about its open nature as a user-customizable platform.

There are four possible kinds of customization of the camera. Three of them are inherited from the previous Model 303 camera and are related to the open software:

1. Modification of the user interface with Web design tools
2. Addition of new applications (or modification of existent ones) that can be downloaded to the camera
3. Modification of the kernel (Linux) to add new drivers.

However, only the use of the Xilinx reconfigurable FPGA – supported by the free-for-download ISE WebPACK development tools – made it possible to increase the camera performance nearly 100 times. This hardware/software customization turned out to be the most powerful of the four possible types of customization:

4. Modification of the "heart" of the camera – the Spartan-IIE XS300E FPGA.

As of now, the camera has only a baseline JPEG compression algorithm implemented in the FPGA, which can serve still JPEG images and Apple Quicktime movies that are made of a sequence of the JPEG-encoded frames.

By adding new software to the camera processor and HDL code to the FPGA, you can use the camera for experiments with advanced image/video compression algorithms – such as 2000JPEG for better size/quality still images or MPEG (-1, 2, 4) for video. Additionally, you could program the camera processor and FPGA for motion detection, pattern recognition, particle discrimination — or whatever else you may think of.

Conclusion

So how did the FS/OSS approach help me to create the Elphel Model 313 camera? Maybe it did not cut my FPGA development time in the same proportion as it did for the software development – the area where it is really mature. But the overall FS/OSS co-design technique really worked:

- I used free Xilinx 2-d DCT implementation that utilized about 30% of the chip resources. It did not meet exactly the requirements of my design – but this is where the advantage of free code is obvious: I could modify the code in a way I liked. With closed proprietary IPs you can't do this.
- The resultant product has unique features that I was desperately looking for as a customer. I wanted a camera I could modify to fit my needs. In many cases, the required modifications I wanted were really minor – but still impossible in a proprietary camera.

On several occasions I was asked, "Isn't that scary to open your design? What if somebody will use it and get all the money?"

- First of all, the license used for the Elphel products (GNU/GPL) is not exactly the same as the public domain. Any company that wants to manufacture cameras based on Elphel designs will have to play by the same rules – the GPL legal protection is no weaker than that of closed source proprietary licenses.
- It is actually impossible to steal this kind of an open design. Even if someone in some far-away country (where there are no copyright laws and GPL is not enforceable) manufactured a derivative closed product, it would lack the critical feature of Elphel cameras – their custom reconfigurability with Spartan FPGAs.
- Elphel has more cameras under development, and I would consider wider availability of the products based on the Model 313 design as free promotional material for my company. Σ

URLs for more information:

- ***www.xilinx.com/spartan2e/*** – Spartan-IIE FPGA family
- ***www.xilinx.com/ise/webpack5/*** – latest free ISE WebPACK software
- ***www.xilinx.com/xapp/xapp610.pdf*** – Application Note XAPP610 "Video Decompression Using IDCT"
- ***www.elphel.com*** – schematics and software for Model 303 and 313 network cameras
- ***developer.axis.com/products/etrax100lx/*** – Axis ETRAX 100LX platform
- ***www.fsf.org*** – Free Software Foundation
- ***www.opensource.org*** – Open Source™ Initiative
- ***www.linuxdevices.com*** – previous articles on network cameras.