Re-Animating the PDP-11/70

Introduction

Light and switchs – a lost era of computer style

When computers first appeared, one of the most distinctive things about them was the complex array of blinking lights and switches. It was an exciting look, and for many years the rows of mysterious blinking lights became the very stereotype of what a computer was.

By the early 80's, microprocessor based front-end processors connected to terminals replaced the lights and toggles, and computers morphed into bland metal boxes with little more than a power switch and an "on" light. The visual excitement of large computers was lost. This point hit home when I was in college and the local TV news crew came to our school's computer center to cover the installation of an expensive new mainframe. When we watched the report on TV that evening we were chagrined to see it featured video of...the *air conditioner*. Of course, it was the only machine in the room with blinking lights.

Figure 1 - PDP-11/70 Front Panel

Like many who became acquainted with computers in the mid-70s, one of the first machines I used was the DEC PDP-11. Of the PDP-11's built with blinking lights and switches, the largest was the 11/70. As a computer it's small by

today's standards (your cell phone has more compute power), but physically it was non-trivial. A typical installation occupied at least two (usually three or four) full height equipment racks painted black with pink and purple trim. Each DEC operating system had a unique light show running on the control panel when the machine was idle. If the machine was busy the lights sparkled and dimmed as it crunched away

I've always missed that visual and tactile aspect of computers that's been lost since the advent of microprocessors. When a console for a PDP-11/70 came up for sale on Ebay, I couldn't resist the opportunity to bring it back to life.

Re-animating the console

The simplest approach to this would be just to hook up a simple micro-controller to blink the lights in a pre-defined way. But why not go further? By putting the PDP-11 console up on the local network, all sorts of possibilities open up for allowing the light display to change on the fly by remote control. By making all of the lights and switches available over the net, the panel becomes a dynamic work of art, rather than a static artifact. Indeed, connecting every switch and light opens the door to a full simulation of the original computer.

I had purchased a surplus PC based SBC with an eye toward using it to drive the panel, but it really wasn't a good fit. It required a noisy fan and disk drive, and accessing its limited parallel I/O required development in Linux or DOS - really overkill for this application.

With the eZ80F91 I saw a perfect fit for this project!

- It is network ready
- It has lots of readily accessible parallel I/O
- Great development and debugging tools
- Small package
- Low power

Project Design

The Panel

The PDP-11/70 panel is a classic example of mid-70's hardware design, built entirely around 7400 series TTL. It uses LEDs for the indicators (the previous model, the PDP-11/45, still used incandescent lamps). The panel connected to the rest of the computer via three 40-pin ribbon cables, along with another connector for additional power.

While a few of the LEDs are driven by local logic on the panel's circuit board, the vast majority are connected (via 7404's and the like) to individual pins on the ribbon connectors. Likewise, each of the toggle switches corresponds to a ribbon connector pin as well. The 22 "register" switches on the left are connected to directly to the power supply via pull-ups. The seven "action" switches on the right are fully debounced with NAND-gate flip-flops. The two knobs are decoded to three and two bits by logic on the board. This logic also drives the LEDs corresponding to the knob settings as well.

In sum, the panel has:

22	Switch register outputs
7	Action switch outputs
6	Knob bits (3+2), plus the Panel Lock
35	Total output bits
16	Data register LEDs
22	Address LEDs
13	Status LED inputs
51	LED Input bits
86	Total I/O lines

Panel controller overview

While the eZ80F91 has a generous helping of parallel I/O, it clearly needs some more for this application. I decided to use a Xilinx XC95108 CPLD to provide the additional parallel I/O. While the chip is relatively expensive (US$30 qty 1), it's very flexible and provides 69 I/O lines in a single package. I used 61 of these lines to interface to the panel and seven to interface to the eZ80F91. Another nice feature of the XC95108 is it interfaces directly to both the 3.3v logic of the eZ80F91 and the 5v logic of the panel.

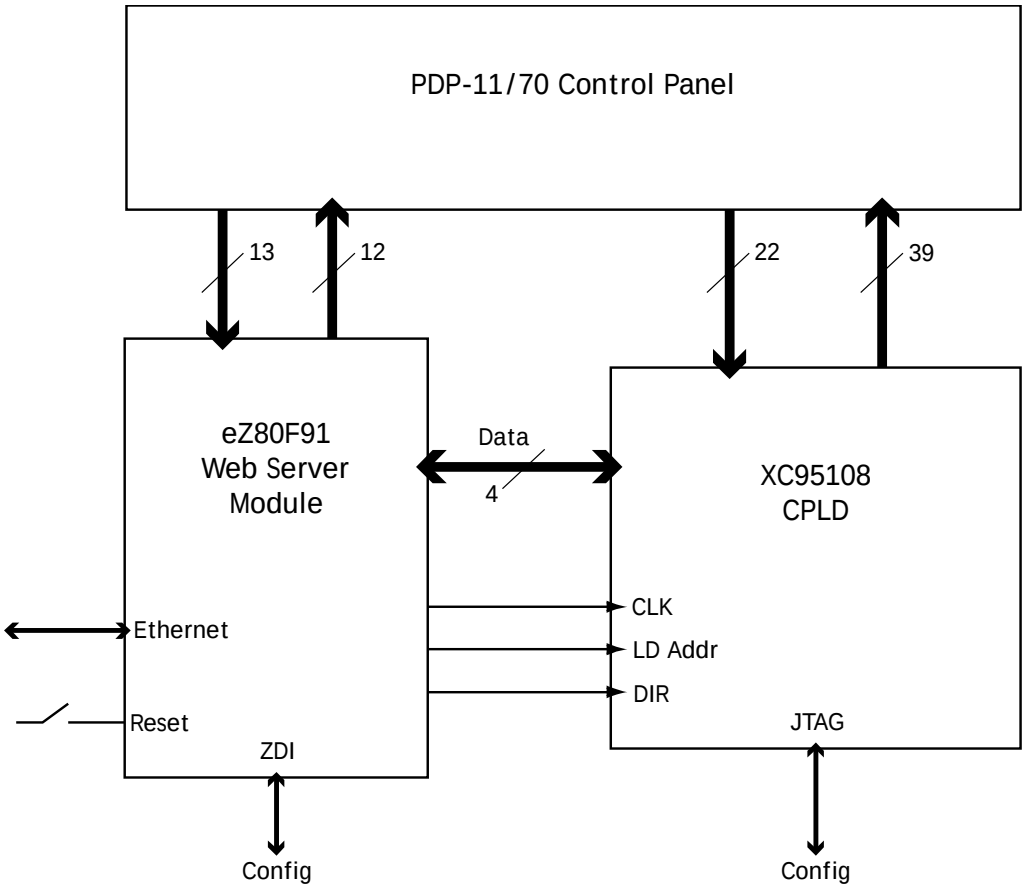

Figure 2 - System Block Diagram

The eZ80F91 takes on the rest of the panel interface, driving 9 status lines (via 10K pull-up resisters) and handling 13 inputs. The inputs from the panel (driven by TTL gates) are connected directly to the eZ80F91, since it is 5v tolerant. To drive the TTL gate inputs, the eZ80F91 GPIO ports are configured in "open drain mode" and pulled up to +5v via 10KΩ resister networks. The remaining seven I/O lines are used to interface to the CPLD.

The I/O lines from the eZ80F91 and the CPLD are connected to the panel using the same ribbon connectors the panel originally used to connect to the PDP-11/70 CPU. These cables also supply some of the +5V power to the panel (for some odd reason, the panel requires three separate +5V supply lines). The system is supplied with +5v from an external power supply. An LTC1086-3.3v regulator provides power for the eZ80F91 module. The only other components required are bypass capacitors, a reset switch, and connectors for programming the eZ80F91 and the CPLD.

The CPLD

The CPLD, a Xilinx XC95108, is configured as a bi-directional parallel port with 22 inputs (for each of the register switches) and 39 outputs (16 data lights, 22 address lights, and an extra status light).

The values for the I/O lines are transferred four bits at a time via a bi-directional data bus between the CPLD and the eZ80F91's GPIO port C. Each group of four I/O lines is given a unique address.

In addition to the four data lines, the CPLD has three control lines to interface with the CPU. These are:

- CLK – Clocks in the I/O address or output data
- LOAD_ADR if high, the data on the bus is clicked into the I/O address register
- DIR – if high, switch data is driven from the CPLD onto the data bus.

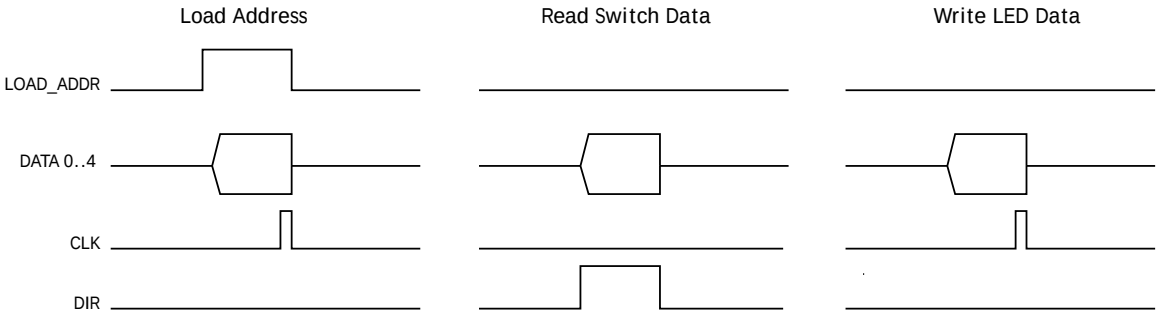

Figure 3 - CPLD Interface Timing

To read the value of the switches, the eZ80F91 first chooses the address of the group of four switches to read. This is driven onto the CPLD's data bus, LOAD_ADR is set high and CLK is taken high then low. This selects the group of four switch values to read. Then DIR is asserted, and the switch values are read from the data bus.

To write to the LEDs, the I/O address is first selected as above. Then the CPU drives the values onto the data bus and asserts the clock line to load the data.

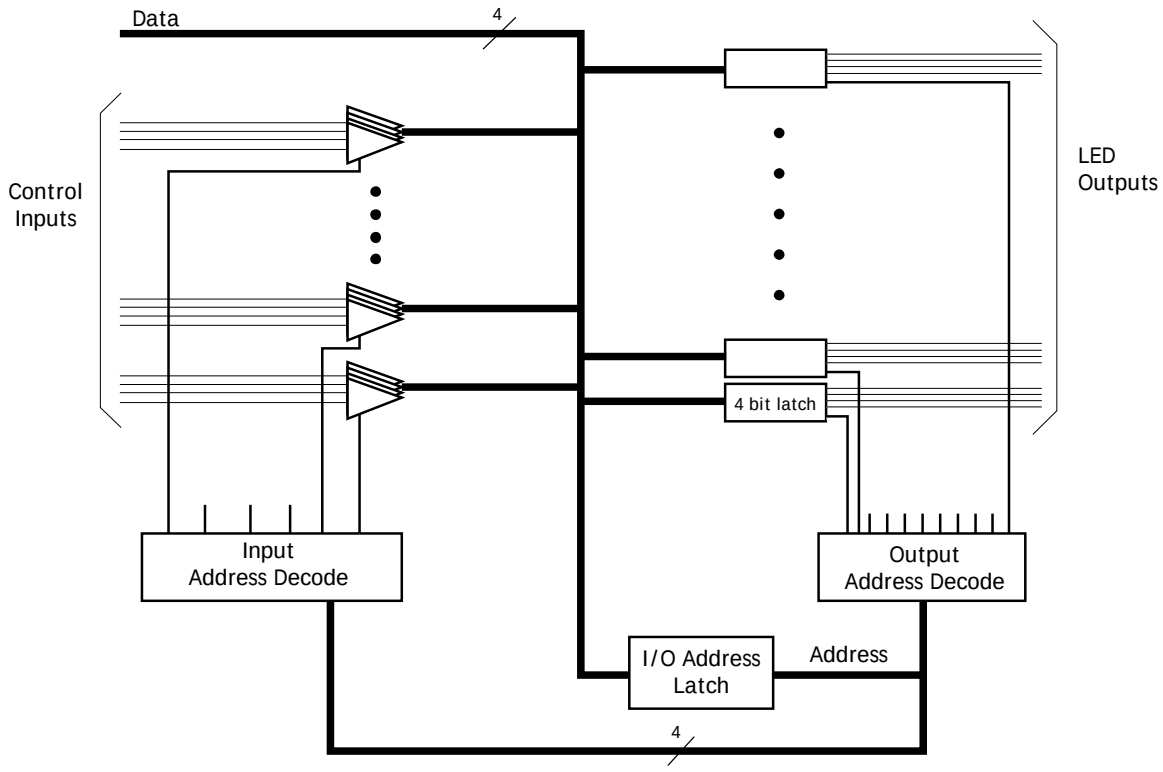

Figure 4 - CPLD Block Diagram

Internally, the CPLD has a group of five input buffers and ten output latches each of which handles four data lines. An address latch is shared for both input and output. The output of the address latch is decoded and used to enable a particular output latch or input buffer. When the DIR line is high, the output of the selected input buffer is driven onto the data bus. When it is low, data on the bus is read into the selected output latch when CLK goes high.

The CPLD has no trouble keeping up with the eZ80F91, which, completes an I/O write every 60us or so. Thus all of the CPLD's output latches are updated in less than a millisecond.

A JTAG header on the board allows the CPLD to be configured in place. The CPLD stores the configuration in Flash memory, so the configuration remains in place after the power goes off.

Design Process

Initial Testing

I acquired a eZ80F91 development board to have a stable test environment for initial software development. This avoids the double-trouble of trying to debug

both hardware and software at the same time. At $99 the Zilog development kit is a great bargain (the wall adapters, ethernet cables, and Ethernet router would cost nearly that much if purchased retail…). Trying out the demos on the development board is a great way to familiarize yourself with the software development process.

The CPLD was defined in Verilog, (see file AllPanel.v) and compiled with Xilinx's WebPack toolset. WebPack includes a behavioral simulation package called ModelSim that was useful at catching few early errors.

The development board was hooked up with clipleads to a Digilent XC95 development board for the XC95108 CPLD. This lash-up allowed me to verify the communication between the eZ80F91 and the CPLD.

Finally, I did a couple more clip-lead tests to verify the eZ80F91 and the CPLD could properly drive the TTL on the panel.

Circuit Board Design

While the project could have been fabricated with wire-wrap, a circuit board gives a much more polished result. With over four hundred connections, hand layout was not really practical. I used Target-CAD to generate the PCB layout from the schematic. Since I don't have much experience at SMT soldering, I opted for through-hole construction. This made the final assembly very straightforward.

One interesting feature of CPLD design is the definition of the I/O pins for the device is completely arbitrary. In fact, it's specified in a separate file from the Verilog source. In my first version of the schematic, I left the CPLD pins in numerical order and connected them arbitrarily by where the fell in the schematic. However, Target's auto-router gave up in despair after half an hour of computation, with many of the signals left unrouted. I discovered it was necessary to assign the CPLD pins so they're near the items they connect to give the auto-router the best chance of working. With careful pin assignment, only a handful of the connections still needed hand routing.

Software

Currently the software performs the basic, low-level access to the Panel. A library of routines (Panel_IO.c) interfaces with the GPIO ports to access all of the lights and switches. These are grouped by their placement on the panel, for

example the switch register, the address and data lights, the status lights and the various control switches.

The hardware is designed so that all of the control switches are routed to port D on the eZ80F91. Since these are fully debounced by circuitry on the PDP-11/70 panel, the CPU may use these as edge-triggered interrupts.

The next phase of the software (currently under development) is to make the switch and light information available over the Ethernet. The Panel controller will listen for connections on a specified port, and report the switch settings and accept bit patterns for the light display. Higher-level operations (such as installing a particular animation pattern) are the next step. A simple sequence of bit patterns for display can be downloaded to the panel controller for playback.

With a bit of clever scripting on another host on the network, the Panel becomes a dynamic display that responds to external events. For example, the speed and direction of the spinning lights can respond to say, your company's stock price or the weather.

Another possibility is to have the eZ80F91 emulate the PDP-11/70. Since the PDP-11/70's clock speed was just a few Mhz, the eZ80F91 could conceivably emulate it at something approaching full speed.

Conclusions

One of the amazing discoveries during this project is how much design software is available for free. The excellent Zilog development tools are included with the development kits. The Target CAD system is available free[1] for customers of the European PCB-Pool circuit board fabrication service. And Xilinx gives away a complete toolset for CPLD/FPGA development and simulation. The companies doing this have a clear economic motive, since hooking people on software tools is a great way to lock in chip and service sales. But it's a huge win for doing advanced development on a modest budget.

I found it very rewarding to bring back a wonderful relic of past computing without investing in several square feet of floor space (and kilowatts of power). Most everybody who sees the panel in operation can't resist flipping the switches to see what they do, something few were allowed to do with a $150,000 PDP-11/70 25 years ago.

---

[1] I opted to purchase a copy for a modest price, so I could take advantage of additional printing features. The project could have been completed with the free copy, though.

I also discovered the practical nature of all those switches and lights. Contrary to what some may think, the front panels were rarely used to enter programs into the machine. It's way too tedious, and the magnetic core memory of the era saved everything even when the power was off. The panels were primarily the domain of the service technician, who used the lights as a window into the machine as he probed step by step to find a bad memory bit or a hung device.

At one point during the final testing of the panel controller before I hooked everything up, I was having a tough time making sense of my board's output, even with a multi-channel logic analyzer attached. Then it hit me – just plug the panel in! Sure enough, a few minutes of flipping the switches and watching the lights revealed the problem, an obscure typo in the CPLD pin configuration file. It's still useful after all these years.

References

A great web site featuring a number of wonderful panel restorations by David House is at http://www.thegalleryofoldiron.com/

You can find out lots more about the PDP-11/70 on the web:
http://www.google.com/search?&q=pdp-11%2F70

Details about the Target CAD system are at:
http://www.ibfriedrich.com/english/index.htm

Information about the XC95108 CPLD:
http://www.xilinx.com/bvdocs/publications/95108.pdf