

Lab9500 – The Heart of a first Digital Design Course

Who am I?

I am Eugene Zumchak. EZ-Ware comes from my initials. I am an electrical engineer with a B.E.E. and M.E.E degrees from Cornell U. I am a part time instructor at the University of Buffalo. In the fall, I teach the evening section of the first digital course, and have been doing so for seven years. Digital design has advanced exponentially since I went to school. The early days used discrete transistors for logic. Integrated circuit logic families started in the early sixties. Just into the seventies, the microprocessor was invented and we haven't looked back.

Digital Design

The teaching of digital design has lagged somewhat the progress in the commercial world. Textbooks are only now beginning to reflect the changes of the last decade. The trend is away from discrete logic chips such as gates, counters, registers, etc. and towards programmable logic with a large number of macrocells in a physically small-sized package at a very reasonable price. It hasn't been practical to design a 4-bit counter using flip-flops and gates for over three decades. It made more sense to buy a 4-bit counter in a logic family. Today it doesn't make sense to build up circuits using counter chips, when an entire circuit can be designed and "wired" in a single chip.

It was traditional to spend a large fraction of a digital course learning to perform the manipulation and reduction of Boolean equations, most of which was made unnecessary once Karnaugh map techniques were learned. Today, there even is some doubt about whether one needs to learn to use a K map. More about this in a minute.

It seems a little foolish to spend time designing master-slave flip-flops when no one has had to make his own flip-flop for nearly 40 years. While some of this stuff is interesting, much is taught because of inertia, just because it has been taught for decades. There is serious doubt that much of it needs to be learned. There is a lot more to learn than twenty years ago. Obviously some choices have to be made.

The course I have been giving does not include a lab. I felt that it was essential that the students have some hands-on experience to reinforce the material. The school wasn't able to provide any funds for lab even in the long run. So I decided to design my own lab materials. What I came up with I call Lab9500.

Today almost all non-trivial digital design involves some kind of programmable logic. For a couple of decades the architecture of PAL devices has been very popular with designers. Originally this was fusible link bipolar logic. The chips were moderately

expensive, took quite a bit of power, required a non-trivial programmer and could not be reused. Eventually this style of chip became available in CMOS technology. Lattice made what it called a GAL using EEPROM technology which not only could be erased and reused, but didn't require a windowed package.

A single 20-pin GAL (GAL16V8) could replace most of the 20-pin PALs, and a GAL20V10 could replace most of the 24-pin GALs, although it remained somewhat more expensive. Unlike the PAL, the GAL had a more versatile architecture that enabled an output to be programmed as either a combinational or registered output of either polarity. This building block output was called a macrocell. A 20-pin GAL had eight macrocells and a 24-pin version, ten.

Today several vendors offer what are called CPLD's which pack more macrocells into a package for a very reasonable price. Unlike the original PALs that could theoretically be hand coded, the complexity of today's CPLD's requires several layers of software. There is lots of good news. Much software is provided free by chip vendors. The programming hardware is provided internal to the chip. Only a simple serial interface is required to get programming data into and out of the chip. No expensive programmer is needed. The software to code the chips is free and runs on a PC. Finally, the price of CPLD's themselves is extremely reasonable. In fact, the XC9572XL used on the LAB9500 costs less than \$3.00! Imagine, 72 macrocells. There is almost nothing that one considers in an entry-level digital design course that can't be implemented with 72 macrocells.

Lab9500 Philosophy

Once it was established that the school would be unable to help me financially with obtaining lab equipment, the only short-term choices remaining were to finance it myself or let the students finance it. Neither seemed attractive. I was willing to put up the development costs, but the students would have to purchase the boards. The bad news was the students would have to buy the lab equipment. The good news was that they would be able to keep it.

To make it useful, the board needed to have sufficient I/O on it. To make it affordable, the cost had to be kept low.

Traffic Light

In past courses I have gotten a lot of mileage out of a traffic light. First, everyone knows and understands a traffic light. A traffic light is an example of a state machine. A simple light with overlapping reds has six states. The states need to be decoded to drive the lights (LEDs). An early test version could use a debounced pushbutton to advance the states manually. Eventually, a one-second clock would be needed to clock the timer that determined the duration of each state. As you can see, there is a lot of practical experimentation before a full working traffic light is realized. The traffic light state machine could be enhanced to include a night mode where the amber light flashes in the main direction and the red light flashes in the side direction. Also a "trip" mode could be

added to permit the main direction to remain green until a car approaching from the side tripped a road switch to initiate a side cycle. Finally, it would be nice if the durations of the green light for both directions were programmable. Such a full-featured traffic light can be implemented on the LAB9500 board.

Time Base

To observe the action of counters and provide a clock for traffic light timing, a relatively slow time base is needed. One could use an analog timer based on a 555 and provide an adjustment, but it would not be accurate. A 32KHz watch crystal could be used, but fifteen flip-flops would be needed to divide that down to one second. Like most small instruments, the logical power supply would be a wall transformer with a DC output that would be regulated on board. However, if one uses a wall transformer that has an AC output, the AC output could be put through an optical oscillator, squared up and applied to the CPLD. Four flip-flops dividing by fifteen would generate 4 Hz. Two more would provide 1 Hz. The downside is that a bridge and filter capacitor will be needed by the on-board regulated supply. Accordingly, the Lab9500 is designed to use an AC wall transformer and deliver a very accurate 60 Hz signal to the CPLD.

One can, however, envision valuable experiments that require a somewhat faster clock. As an option, a socket is provided for a very low-cost PIC processor with built-in 4 MHz (factory trimmed) oscillator that is pre-programmed to generate 1 MHz, 250 KHz, 100 KHz, 10 KHz, 1 KHz, 60 Hz and clocks suitable to clock a state machine that implements a 9.6Kbaud UART transmitter or receiver. The latest version of the Lab9500 also provides for a DIP oscillator module.

Programmer

The Lab9500 would be seriously compromised if an external programmer were needed to program the chip. The programmer interface suggested by Xilinx, hooks up to a printer port on a PC, takes only a few dollars worth of parts, and uses a small cable to connect to the CPLD board. It should be noted the XC9572XL chosen is a surface mount 64-pin flatpack with pins on 500 micron centers. The chip is usually intended to be programmed after being soldered to the board. An instructor at Washington State University, Clint Cole, suggested including the programming interface on the board itself, instead of building a separate programmer. All that is required then is a standard DB25 male/female cable to attach to a printer port. This idea helped to keep costs down. The Lab9500 comes with an AC wall transformer, and the DB25 cable to make it ready to use "out of the box". The package also includes a hardcopy lab manual of experiments and a CD ROM with additional materials (hopefully including the free Xilinx software which can be downloaded in any case off the Internet).

On-Board I/O

An important consideration is including enough I/O on the board so that additional interfaces are unnecessary. Six bright LEDs in red, green and amber serve to represent a

traffic light. A bank of eight additional LEDs can serve as general-purpose outputs. On my traffic light, I used them to show the down-counter in action. I also used them to show the registers that hold the programmable green light times. For inputs, I use a ten position DIP switch for fixed inputs and three pushbuttons (two debounced) for momentary inputs. Originally I had an 8-position DIPswitch, but with ten, I can use two of them to select one of four different functions for the other eight or for the 8-LED display.

While other types of switches would more convenient to use than a DIP switch, they would take up more board space and significantly increase the cost of parts. As a compromise, all switch inputs are brought to the card edge where a DB15 connector can be soldered on. This permits an external switch assembly to be used if desired.

E-book

It was my original intent to provide a versatile lab board, a hardcopy lab manual of experiments, and software to generate code for the CPLD. On the other hand, to increase the sales potential of the board (and decrease costs), I thought I should make my CPLD development system available to hobbyists and non-electrical engineers. These others will not have a hundred dollar digital design textbook. So I decided to add to the CD ROM a simple digital design e-book. This is not intended to replace a course text, but with contributions from Lab9500 users, it could eventually be more than sufficient, so that for just a little more than the cost of a textbook, the student could have a textbook AND a CPLD development system.