Second Version of Lab Manual. Final revisions have been made for the material in the Appendices dealing with everything except Xilinx FPGA use, which still requires some further checkout. I will be printing hardcopies of the remaining pages of the manual this week (October 12th. I still reserve the right to make further changes to Lab C.)

Brown University
Division of Engineering

Engineering 163 - Digital Circuit Design

Semester I - 2004-05

The Student Lab Manual

Wm. R. Patterson
J. D. Daniels

Table of Contents

1. General Introduction

Whatever the opinion of its participants, the purpose of sex is combining design information in chromosomal conjunctions[1]. Disparate parents produce compromise offspring. This gives biological enterprise a rapid rate of evolutionary change, while allowing the bulk of cellular replication to proceed by meiosis, a process that is so exact that it produces monotonously invariant offspring. In other words, sex makes life interesting.

Two of the most interesting features of living things are autonomous reproduction and the ability to do design optimization on the fly by genetic modifications during that reproduction. That attempted optimization is the ultimate difference between you and a brother or sister and is the engine of evolution. Until very recently, no man-made objects came close to having these properties. Computers, however, have prompted speculation about the possibility of machine biology for many years now[2], and recent developments in the use of computers both to design and build new machines have suggested how such things may eventually come to be. For example, in a paper by Lipson and Pollack in the August 31, 2000 issue of *Nature*, researchers at Brandeis University reported using genetic algorithms to design simple robots. The algorithms tried random variation of part sizes and shapes against a measure of robotic usefulness. After some number of "generations", the designs were fabricated without human intervention from extruded plastic in a rapid-prototyping system. People snapped in a battery and motor, set them down, and watched them walk. This emulates rather well in a simple system a rapid evolutionary process that does not have to wait for extended natural trials. Presumably this is a next step in the process.

In Engineering 163 you use software to do schematic capture, to simulate systems, and to download designs into programmable hardware. These are primitive but necessary steps in designing and reproducing machines and are already done with minimal guidance from fairly abstract descriptions. Ways to eliminate your part of the process will probably occur to you as you do these things. True machine autonomy will begin when engineers lose effective control of the design process or disappear altogether[3].

Analogs to many internal cellular mechanisms are clearly evident in the way computers are built. It seems very likely that the autonomy of biological systems will one day govern the building of machines with humans reduced to the roles of mutagens and growth hormones. One can even identify emergent mechanisms that play the same role for machines as sex for biology and predict the evolution of these mechanisms from the episodic and primitive sex of bacteria into more elaborate forms analogous to what preoccupies anthropologists and adolescents.

Just after the Second World War, John von Neumann applied automata theory to computing machinery, devising the "von Neumann machine" model of a computer. Although systems models have become more intricate, their basic ideas remain the same. In Engineering 163 you learn the rudiments of how to build the logic for such machines. At the same time, von Neumann applied his ideas to cellular reproduction. He identified certain abstract mechanisms that would have to have

[1] Particularly unfortunate are views that it is a form of urban public entertainment or a grave Matter of State.
[2] Cf., for example, the play, R. U. R., by Karel Capek (1920).
[3] Perhaps dehydrated by HAL.

physical realizations in all biological cells. He listed the need for *logic organs* and for *separate* logical instructions that cells or machines would have to have. Those instructions would contain the ones needed for a cell to build a copy of itself by following them, including an instruction to make a copy of the list of instructions. Today we know the physical implementation of instructions in biology is in DNA and of the construction machinery in RNA and protein. (Admittedly there is some blurring of the distinction in such things as retroviruses where RNA carries information too.) The vocabulary of the process is the codons of DNA, triplets of GACT amino acid sequences that determine protein structures. We recognize the absolute distinctions between the instructions and the machinery that embody von Neumann's suggestions. The only man-made machine to have similar capabilities is the digital computer. It alone can hold an encoded copy of its own design, can be given the means to reproduce itself (an automated factory rather than a mother), and the instruction to include its own design into the files of its offspring. (Given how buildings are built, there is even no reason computers can't be downloaded with instructions on how to build their own mothers -- blueprints!) Given the successes of both biology and computers, it seems likely that the convergence of their properties will continue for a long time.

Further complicating life for those of us who would be seers, is the introduction of a whole new regime between the biological and computer worlds in nanotechnology. In this domain, computers design and control the construction of extraordinarily small, often molecular scale, machines through principally chemical and biological means. These are envisioned as symbiotic to our biological systems for such applications as internal repairs, drug delivery, disease diagnosis, etc. Ideas are being developed for extraordinarily sensitive detectors – single molecule detection – of specific proteins. Perhaps they will even be adapted for fast, cheap genome sequencing. These toys suggest very different scenarios for the convergence of biology and computation, but, so far, they are not designed for autonomous reproduction.

From the Egyptian Old Kingdom (*ca*. 2400 B. C.), when we know papyrus was used for writing material, until the late 1970's, engineers proceeded about their work in much the same way. Designs began with handmade drawings, which were made into models or breadboards for testing. Drafting skills were highly prized. Some prototypes such as ships' models could achieve the status of folk art. Breadboards were scaled up or reproduced as needed. This is the model of engineering which influenced the current structure of Engineering 163 and which can potentially make this course a great deal of fun for you. You can get much satisfaction from seeing the widget you thought up actually function in front of you. You should be aware, however, that this model for engineering and for teaching is probably already obsolete. We only continue the practice because it remains a good way to teach how the product actually works. Like many engineers in my generation, I regret the passing of the need of all engineers to learn to solder. The smell of burning flux is perfume to us. However, the reciprocal bootstrap effects of computer design and computer usage is changing the nature of the process to one in which computers play the dominant role. Eventually, most systems, large and small, will be designed entirely on computers and will be built on automated equipment driven from their output. Through logic synthesis from high-level specifications, even many of the nominally creative aspects of design will be removed from our reach. Eventually, even in this course, routine designs will be done by giving system-level specifications to a program and then skipping directly to testing the resulting device. For example, I introduce the use of VHDL -- the VHSIC Hardware Description Language -- as a tool for synthesis of logic from a high level description as a lecture and lab topic. In lab F, you have the opportunity to build a simple 8-character display sign by writing down only how the data is to be taken out of a memory and

is to be put onto a display. Deriving the Boolean expressions, deciding on an implementation strategy, and wiring the gates are all automatic.

In response to this scenario and to the practice of the profession, we require you to use some *industrial strength* CAD software. You will learn schematic capture in *DxDesigner*, a product of the Mentor Graphics Corp. who have graciously given us the rights to use it at nominal cost. This tool is one of the most popular starting points for printed circuit board design. On the other hand, we no longer do circuit simulation of discrete circuits but only of programmable devices. (Some of our simulation is applicable to fairly complex board level designs when, as is now frequently the case, hardware descriptions of all the simulatable chips on the board are available from their vendors. We do simulation of programmable rather than custom parts because these parts are the easiest to use and simulate for small sophisticated systems.) You do labs using Xilinx XC9572XL complex programmable devices (CPLDs) and XCS05XL field programmable gate arrays (FPGAs). For their simulation, we will use a mix of Xilinx software to program the devices and ALDEC software to simulate them. This is the second year we are using the ALDEC software and the Xilinx hardware is also being newly upgraded this year. Long experience suggests that this material will not be integrated without a certain amount of difficulty I ask your indulgence.

In the last part of the course, several experiments will use the Xilinx Corporation's field programmable logic arrays (FPGAs). I will discuss the underlying ideas for these parts in class, and the lab manual, and other handouts will show you the necessary manipulations. The designs are not especially difficult, but the level of detail can be intimidating. Since this style of logic design is clearly a prevailing pattern, you should regard these exercises as reality-therapy. (In terms of sheer number of designs, as opposed to the sophistication or cost of design, it is estimated that only 3500 custom integrated circuit designs are done per year in the United States, but the more than 10 times as many FPGA designs will be built. The reason is economics – IC's are too expensive to recover costs easily while FPGAs may have high part cost but their design cost is low.)

These topics, which were introduced to this course in the early 1990's, are not likely to be permanent but rather will prove at once transitional and transitory. There is much more to come. Schematic drawing will largely disappear for sophisticated systems because their complexity will be too high for such a representation to be useful. This is already the case for any overall schematic of a microprocessor, FPGA, or other truly complex chip. As a profession, we seem to delight in devising ways to automate each other out of business.

I think the nature of the sea change in our profession is most clearly seen in the biological analogies. An electronics factory is already rather like a living cell with a computer playing the role of nucleus. The computer nucleus sends its messenger RNA (m-RNA) in the form of encoded serial assembly files to the factory floor. There, pick-and-place robots - really mechanical ribosomes - select the new computer's chips and insert them into the appropriate locations. The process mimics the attachment of amino acids to the peptide chains of proteins and enzymes as they are synthesized in cells. The tapes or webs holding components for the robots are similar in function to the t-RNA (transfer RNA) of cells. Finally the memories of the new machine are loaded with design files from the host computer -- DNA passing on the genetic heritage. (I suppose a computer loaded only with application software but no reproductive capability will eventually be regarded as a sort of mechanical eunuch.)

Even the tight relationship of function to form that exists in biological molecules[4] exists already in machines. For example, there are a myriad of details of electrical function, semiconductor properties, and device physics that are critical to the operation of integrated circuits. Yet these are largely irrelevant in the design of such circuits once the manufacturing process is settled. Instead, the physical expression of functionality is a graphics problem of mapping logical functions into simple shapes on the surface of the semiconductor. The shapes define transistor gates and drains and expand into the larger networks of rectangles that are logic gates, busses, registers, etc. The idea of a transistor as simply the intersection of red and green squares on a computer screen has permeated Computer Science, driving out electrophysics and reducing all functionality to shape. (Interestingly, a similar abstraction seems to afflict biological studies. Stereochemistry, driven by computer graphics and largely stripped of quantum mechanical considerations, has come to dominate biochemistry. It is seen as the way to design drugs, to understand cellular function, etc. Only bond angles, folding constraints, and like crude structural measures are thought of as the key to the future.)

Sex in this context is the exchange of design ideas. So far, humans have done most cross-fertilization for new models. Impressed by the features of others' machines, engineers have not hesitated to borrow ideas - at least within the loose rules of patent law. Perhaps this is the equivalent of bacterial sex[5]. In the long term, hardware description languages (HDL's in the jargon of the trade) will provide a common descriptive vocabulary akin to the codons of DNA and m-RNA. The VHDL language you see in this course is a clunky first version of what might someday serve the purpose of DNA. (There is a competitive HDL called Verilog that is finding wide acceptance in industry. While it is less verbose, it too suffers from synthesis limitations and a certain clunkiness. By the rules of standards making, each of these languages is revised and extended every five years, marking a sort of glacial pace toward fluidity. The problems with simulation time caused by the generality this verbosity allows have even caused some design groups to resort to C++ simulations of circuits. Within the last couple of years, several small companies have begun offering general-purpose tools that allow one to start the HDL process with appropriately structured C code.) Already serious attempts are being made to bind long sequences of statements in these languages into the electronic equivalent of genes and chromosomes. Companies building design automation tools are offering special tools that allow ever higher levels of abstraction by giving compilers the ability to write lower level descriptions -- bus/register level -- from top-level functional statements. Another similar trend is to “silicon sockets,” the idea of selling HDL descriptions of such modular units as complete microprocessors, MPEG decoders, I/O interfaces, etc. that can be dropped into a user's design and synthesized with confidence that they will work. Already one can get free VHDL describing simple processors in terms that allow synthesizing them in FPGA programmable logic. The buzz words describing such reuse and transfers are “SOCs- Systems On a Chip” and IP – Intellectual Property. Some successful companies do nothing really except generate IP – an example is

[4] The premier example of this is hemoglobin, which exists in numerous chemical variants for different species. The key to its ability to bind oxygen delicately and reversibly for transport through the body is a common “quasi-mechanical” structure that changes subtly on oxidation. Iron provides both the coloration of blood and the chemical site for oxygen attachment. However, this is not the strong connection of the typical inorganic iron to oxygen bond, but something more akin to delicately grabbing a molecule – molecular tweezers as it were. The iron atoms are encased in shells of other atoms that partially shield the oxygen from them. A change in the shape of this shell induced chemically by CO_2 and other materials pops the oxygen off where it is needed – where prior oxygen has been used to make CO_2. Thus in the reverse of the architectural maxim, function follows form.

[5] The discovery that many bacteria do indeed sometimes reproduce sexually earned Joshua Lederberg a share of the Nobel Prize in Physiology and Medicine for 1958.

MIPS, the computer architecture and chip design house that does general purpose and embedded processors. The transfer of code for such IP/SOC products is through files that are like the DNA genes and chromosomes that code for the differentiation of lungs, kidneys, hearts, and hands. (We have only a very limited idea of what the code for lungs or livers looks like and are a long way from breaking it. Similarly, when one buys code for a processor say that will be instantiated within a custom integrated circuit or in an FPGA, you are not supposed to break the code, just to use it.) While such efforts may initially limit design structures, this limitation will eventually only seem the equivalent of matching species. Within the mating of machines of the same species, it will still be possible to vary features without injuring the viability of the offspring. Buses may expand or contract, register files lengthen or shrink, but the race marches onward. Eventually, the autonomous exchange of design files between computer factories over networks without human intervention will be possible. Sex!

This sexuality may seem bizarre. It is likely to be promiscuous; networks seem to inspire casual usage -- think of how people use networks for email and web surfing already. The source of a machine's sex drive may be a high-priority, gopher program. Jealous human managers will have to alternate between the roles of chaperone (protecting the family honor, *i.e.*, patents and copyrights) and marriage broker (finding suitable sources of cross fertilization to improve performance). In this context, incest is drawing from too small a pool of design ideas, causing reinforcement of design weaknesses. The taboo against it is likely to be strong because the effects of it are already obvious -- think of the fate of certain IBM and DEC product lines, the designers of which were too inward looking. The Digital Equipment Corporation story is particularly cautionary. They built the second largest computer company in the world around inexpensive machines that used very complex instruction sets. Unfortunately for them, it turned out that such processors were condemned by a fundamental design flaw to run more slowly than machines based on other strategies. Despite a successful redesign of their products into the Alpha architecture, they were unable to regain their position, and the corporate identity was swallowed up by absorption into Compaq Corp for which the Alpha line was only one part of business tactics. More recently still, the Alpha group was sold to Intel, which was looking for talented designers, not necessarily another product line. The remainder of DEC, its service business, was then absorbed into the new Hewlett-Packard behemoth.

Machines may have a clear preference for bisexuality as networks usually lack a sense of directionality. One imagines the ploys of a young machine – "You show me yours, and I'll show you…" New forms of autoerotic reproduction are possible and are already evident in engineering practice. A computer can consider variations on its own design, estimate their effect on benchmark performance, and implement variant offspring by itself[6]. Sociopathic behavior may appear. Machines descended from IBM's, for example, might exhibit a sense of ethnic or racial prejudice in refusing to exchange files with the descendants of Fujitsu. (I am not talking about user data file incompatibility, which is self-defeating, but about the important genetic stuff!) Today's machine designers, although tending to libertarian views themselves, talk endlessly of hierarchy in the structure of their products. If this central importance of rank and position should continue, then machine society may well reflect that. Wide-word machines might be the Alphas of this world[7] while RISCs serve Beta and Gamma roles and CISCs are condemned to an unhappy existence as the Del-

[6] This clearly gives new meaning to certain vulgar admonitions.

[7] Alpha as a rank should not be confused with the unfortunate Alpha RISC machine mentioned above that has all but lost the battle against extinction that is the common fate of genetic dead ends.

tas and Epsilons doing data logging, coolant monitoring, and bathroom control for whatever humans remain. Irresponsible and unchecked reproduction is foreshadowed in the well-known story of the Sorcerer's Apprentice[8]. Finally, suppose a supercomputer or workstation factory exchanges files with one that makes appliance microcontrollers. Is this a machine form of statutory rape?

Machine sex may even prove to have some advantages for evolutionary change over the more conventional kind. As far as we know, DNA combination in nature is more or less a random process. Only with clever encoding and with a very near match of species is an even functional creature possible. Real improvement or retrogression usually takes many generations. Some have even suggested that survival of the fittest as the sole criterion does not assure evolutionary progress. Cats, for example, may be evolving toward dumber less competent creatures, aided by a symbiotic relation with humans who prefer lunkheads. However, "genetic algorithms" are already being used to solve some intractable math problems and to design simple robots as in the Brandeis work mentioned earlier. These techniques combine random "mutations" with immediate measures of success. Machines do not need to raise offspring to maturity to find out if they are satisfactory or not, a great shortcut. (Perhaps we should all be thankful that such measures were not available to our parents. You might have been your 32nd sister if they were.)

Of course, the job of predicting the future is a thankless one. It is easy to over-predict – think of the Jetsons! It is altogether possible that my predictions of machine autonomy through HDL evolution will prove as elusive as the flying cars that the Jetsons showed us in the 1960's for the way we would live today. Current hardware languages lack the sort of encoding that can benefit from the random experimentation that drives biological systems. Genes evolve by random alterations of individual "letters" along the DNA chain. The randomness results from quantum mechanically improbable forms of the amino acids that can allow incorrect letter substitutions at random sites but with probabilities of the order of 1 wrong letter per billion or so transcriptions. It has been suggested that larger simultaneous changes may be possible by whole segment replication or resplicing from similar random mechanisms. (Incidentally, there is even some parallelism between the transcription process and the way digital media store and retrieve data. Both involve error correction methods to achieve high accuracies. Digital systems scan bit streams for errors detectable using extra stored bits to allow errors to be found and corrected – see Lab 1. DNA transcription would be accurate to an error rate of only about 1:400,000 without a process of molecular mismatch detection and correction which uses helper molecules to monitor DNA reproduction for mechanical strain caused by mismatched base pairs.) Most biological substitutions either do nothing or kill the baby. A few result in improved offspring. Unfortunately in most HDLs, there is no way to make a sensible and small random change without making syntax errors. Changes have to be fairly well thought out and genetic algorithms do not work efficiently in modifying complex behavior. All this is a barrier to creative automata.

Looking at the immediate evolution of practice in circuit design and computer architecture, the role of the human is not lessening but evolving. In this course I take some class time to talk about transistors. I have long had some misgivings about that practice since it was not clear to me that such knowledge was actually necessary. Recent evolution in computer tools and the growing necessity for all electrical engineers to know some IC design has made it clearer that understanding the physical layer of design is becoming more important. Tool designers are making the physical layer visible early in the design process to influence the most abstract level of design. The reason

[8] Most clearly expressed, perhaps, in the *oeuvre* of M. Mouse.

is that different architectures may have inherent advantages and disadvantages given how they will be realized. Without some knowledge of what the synthesis tools will do with a specification, you may not be able to devise sensible systems. Thus while you can do more design by programming, you may need physical insight to understand why different abstract forms of the same operation will appear to have different results.

But regardless of how clever machines may become or how insignificant man may become in their design, it is clear that machines will continue becoming more powerful. Change is still the order of the day. For the last forty years, the primary driving force for change has been the development of techniques for manufacturing ever smaller and more densely interconnected transistors. (The integrated circuit only dates from 1959 – less than fifty years ago.) When I graduated from college, the first integrated circuits had been on the market for about one year and contained on average only about 20 transistors. I had never had a chance to actually use one myself. Now memories with well over 256 million transistors sell for a price much below the inflation-adjusted price of even one of those 1963 circuits. Without meaning to denigrate this accomplishment, one should note that it is only a single moment in a continuous process of growth in device complexity. Our confidence in continuing this rate of change is based on the fact that the lines of development are clear for this exponential growth to continue for at least another ten years. Optical lithography, much to most people's surprise, will be able to build devices with features of 0.065 µm - almost a factor of two smaller than current practice. (Intel currently manufactures processors using feature of 0.09 µm and has announced successful operation of processor with 0.065 µm with scheduled shipment dates in 2005. They also have expressed belief that feature sizes down to 0.04 µm will be possible without major conceptual changes to production.) Many observers believe that 0.03 µm will be possible. (Only a relatively small subset of manufacturers has 0.09 µm lithography in full production, while more are still just ramping up.) Manufacture with .25 µm features is still common in many very cutting-edge devices. Since this is a linear dimension, a twofold shrinkage implies that the circuit density will be 4 or 5 times higher. Already the 256 Mb memory is an old product, most commonly built with 0.13 µm techniques, and 100 Megatransistor processors are being reduced to practice. Even 1 Gb and 4 Gb memories have struggled to life in the laboratory. The 1Gb memory should move into production within the next two years. Clock rates that have now reached 3.3 GHz and will likely exceed 5 GHz in a few more years. (Clock rate is a little bit deceptive. Reducing the size of transistors is no longer having the dramatic effect on gate speed that is once did as the effect of size on the ratio current capacity to capacitance is more modest at very small sizes. Much of clock rate change comes from pipelining and too many pipeline stages eventually become counterproductive however much it may increase the clock rate.)

In 1994 the Semiconductor Industry Association published a "National Roadmap for Semiconductors," in which the authors confidently extrapolated current trends to 2010. The size and speed of the circuits they foresaw were astonishing. The 2003 roadmap update continues the same rate of extrapolation for another five years. While some of us are skeptical of this process continuing forever, no one believes the industry will not soon achieve another order of magnitude increase in circuit compactness and performance. Recent chip sizes have tended to be smaller than previous generations because of cost considerations and because learning to use a huge number of transistors effectively takes time. That small-chip trend may also reverse and larger chip sizes will further increase the capacity of individual components.

An interesting sidelight of the development of ever-smaller transistors has been an ever-increasing cost for any sort of custom circuit. It now costs approximately a million dollars just to make the mask set required for a state-of-the-art 0.09 µm processor. (Masks are 1:1 glass photographs of the required material patterns on a chip that are used to form those elements through photochemical reactions.) Moreover, it has been reported that unforeseen physical effects end up requiring more frequent redesigns with their concomitant mask respin costs. Designs that exploit the manufacturing technology to its fullest have non-recurring-engineering costs (NRE, in the language of the trade) that may run to hundreds of millions of dollars. One result of this is changing economics for custom versus programmable logic design. Worldwide starts of custom circuit are reported to have dropped from almost 10,000 per year in the 1990s to about 3500 today. After the large microprocessor companies, the first ones to take advantage of 0.09 µm technology have been the FPGA companies like Xilinx. Their products can be scaled up in device count almost indefinitely with little additional engineering because their chips have largely repetitive circuits with very regular layout and interconnections. It is easy to exploit the benefits of reduced size and the volume of chips produced for multiple markets makes mask costs affordable. Xilinx has announced that volume pricing on a 1 million gate equivalent FPGA will be under $ 15 within calendar 2004.

Early in your career, it is likely that processor chips with well over a billion devices will be made routinely somewhere. There are even several possible candidates for really drastic changes over still longer periods. For example, the possible bandwidth (*i.e.*, speed) and size advantages of optical systems may result in integrated electro-optical widgies that will displace many of the things you learn about here. (Then again, there is a possibility this will prove a dead-end. While electro-optics will certainly find use in interconnections, the search for practical logic elements has been more difficult and less fruitful than many expected. Similarly, the density advantage alleged for optical systems has eroded as conventional ones have shrunk the size of the elemental logic element to sizes below the wavelength of the light used in most electro-optical systems.) An even more remote possibility is that studies of how biological systems process information may lead to very different means of organizing information systems, that is, to systems with the properties of real neural networks, including "intelligence." Although an enormous amount is known about the physiology of the primate brain and there is a similar understanding of individual neurons, little is really understood about information organization. One hopes that Nature has some little trick for handling information that will be amenable to electronic construction and will give machines a crack at the sort of fuzzy creative thought we enjoy. The comparison is to something like DNA where the very simple trick of data storage is sequential base pairs held together by a few phosphorous atoms on the ends. It took a long time to discover that the "uninteresting" deoxyribonucleic acid molecules were the site for genetic information of this simple form and longer still to work out the details. Perhaps information in the brain is stored and processed with some similarly elegant trick. Brown's Brain Sciences Institute has a number of programs working on learning more about this, but it is a formidably difficult problem. That makes it difficult to know what implications such techniques may have for machines.

Already by some simple measures, fast processors reach the same rate of computational evaluations as the human brain – perhaps even faster. The human brain has about 10 billion neurons, the firing of which encodes our thoughts and memories. Physiology limits firing rates to a maximum of around 800 Hz, so regardless of what such a signal encodes, a recalculation of that information cannot happen any faster than this rate. So the brain can only do calculations at a rate of about $8 \cdot 10^{+12}$ node evaluations per second. (Since the state of the neuron is essentially digital –

firing or not-firing states – it is unlikely that information production exceeds 1 bit per evaluation or $8 \cdot 10^{+12}$ bits per second.) The data from these nodes fans out to an average of around 10,000 other neural inputs or synapses, for a total of 10^{+14} interconnections. It is thought that synapse strength and connectivity contribute to our durable memory capacity. By comparison, a middle-sized workstation has about 10 million nodes working continuously and simultaneously. They do evaluations at a sustained rate of over 1 Gb per second each, for a total evaluation rate of 10^{+16} bits per second. (This probably overstates the rate of new information generation because the new evaluations are not all suitably new results. Still the rate of new information is a fairly large proportion of this.) While working interconnections are modest compared to the brain's wiring, the computer does have on the order of 10 billion memory nodes that can be accessed quite quickly – average latency on the order of 1 ns including initial access restrictions – and with greater reliability than your average human's. Clearly the workstation is faster at what it does but has relatively restricted communications paths, while the brain is slower with massively parallel interconnections. Again the implications are unclear. It seems unlikely the machine will ever achieve the interconnection density of the brain or that the brain will achieve the speed of silicon.

The comparative intelligence of men and machines sometimes seems a matter of selecting your examples as to which is the sharper creature. James Trefil, a physicist at George Mason University, has written[9] about the comparative intelligence of men and advanced machines. He argues that machines are unlikely to develop the same intelligence as people, but that this does not preclude a formidable intelligence of a different type. Given such interspecies differences, I suspect there is even the possibility of competition for habitat (dry, air-conditioned spaces) and food (electricity). With competitors like HAL, we were well advised to keep a supply of silver stakes handy. (After all, silver is the best solid conductor of electricity and so should be even more effective against a silicon monster than against a vampire.)

In Engineering 163, you begin your acquaintance with the design of logical systems. You start by designing small systems with TTL integrated circuits, building them, and then fiddling until they work. The fiddling can go from minor rewiring to complete redesign as one gets accustomed to the subtlety of the problem. This method can be very satisfying and great fun. (Please do not assume that "fiddling" alone is the right way to design. Care in analysis and simulation is the only route to larger systems. Fiddling is simply a necessary step in learning to ask the right questions.)

Your kits are largely made from "TTL" or Transistor Transistor Logic parts because these are the main commercial source of very small blocks of logic. But it is important to realize that TTL is not the only or even the dominant expression of logical systems. In fact, the main driver for my changing the simpler labs is that the components are so obsolete they are disappearing from the market altogether. (If you decide to keep your kit at the end of the course, you will have a premade antique collection.) The dominant manufacturing technology for circuits is CMOS - the Complementary Metal Oxide Semiconductor process, which we will examine briefly in explaining how gates actually do their work. For labs two and six in which you measure gate properties we will supply some "LVTTL" or low-voltage TTL parts. Despite the name these are actually CMOS devices. They are intended to supply small logic blocks for systems using power supply voltages

[9] James Trefil, *Are We Unique?*, John Wiley, New York, 1997.

less than the 5 volts that was standard for many years for TTL. Their working range is 2 to 5.0 volts and you will look at their speed and power performance over that range.

As another hands-on alternative to TTL, we will explore the use of CMOS logic arrays that can be customized by programming from software to realize a wide range of modestly complex functions. We will use the Xilinx FPGAs (Field Programmable Gate Arrays) because they are indefinitely reprogrammable with rapid turnaround. The Xilinx Corporation has kindly sold us a substantial set of software at a deep discount, and we have built prototyping systems to ease the problem of incorporating these sophisticated devices onto your breadboards. This year we are updating the prototyping system, so please be patient with our problems.

Complex programmable logic devices (CPLDs in the trade jargon) are another competitive area of development. They are generally smaller and less flexible than FPGAs, but their propagation times are fast and better controlled. (As with many general statements about technology, this one has to be qualified. Improving speeds in FPGAs have substantially reduced their delay times until now one has to examine particular applications to decide whether a CPLD or an FPGA will do the best job. CPLDs, however, do have the curious properties of more efficiently generating simple Boolean functions of many inputs and of having propagation times more or less independent of logical function. They are also non-volatile, that is, you do not have to reprogram every time you turn the circuit on.) You do Lab 5 with a CPLD and may use it again in Labs 7, A, and B with some restrictions. The ninth lab is on timing simulation, thus introducing the problem of verifying designs before they are built. Lab F uses VHDL to synthesize a circuit and you are free to apply those ideas to labs C, D, and E too.

The Mentor Graphics *eProduct Designer*, Aldec Active-HDL simulation software, and Xilinx programmable design software for this course will run on the workstations both in the Hewlett Instructional Computation Facility and in the lab. (The third edition of the class text -- *Digital Design: Principles and Practices* -- includes a CD-ROM with the PC version of Xilinx's student FPGA design package. You may use this too for your labs.) We will use the PCs in the Electronics Laboratory to download design files into the FPGAs or CPLDs. Those computers are part of the same network as the computing lab, so incremental changes to designs can be done at any workbench. We would encourage you to make use of this software early and often. Ideally you would use it to make schematics before attempting to build any of the more complicated labs. One of the early help sessions will be devoted to software demonstrations. All the software has undergone major upgrades this summer, so there may be some glitches as I try to adjust the documentation on how to use it.

I have you start the programming of CPLDs using the older language ABEL that is covered in some detail in the textbook. However, this language is really obsolete and I continue its use because structurally it is closer to the Boolean logic and storage nodes that are the physical elements of digital systems. This is a course in digital logic, not on computer architecture (EN164), integrated circuit design (EN160), or design tools (EN293 in some years). Consequently, I restrict the use you can make of the more complex syntactical constructs of ABEL. Generally, you can't use state machine syntax in the first state machine lab (lab 5) or arithmetic constructs in the arithmetic lab (lab B). At the end of the course, I introduce VHDL, one of the two popular current languages. In using that for lab F, you are free to be as convoluted and syntactically elegant as you can manage.

Every time one changes software or brings in new hardware, there are always unexpected problems. In the dark days when your schematic is askew, the simulation program gives you only impossible to understand error messages, or the logic analyzer pattern is completely unrecognizable, please remember that such experience is very common – part of the human condition. New tools are often hard to master, but they are necessary. Just resolve that if you ever design such programs or instruments yourselves, that they should be easier to use.

The emphasis throughout Engineering 163 is on the design, construction, and verification of digital circuits. In class, we will discuss the conceptual and physical building blocks available to the designer, and show how these tools and techniques are used in various design situations. In the lab, you will take this design theory and put it to use by building actual hardware-based solutions. This may well be your only exposure to full responsibility for making a product work ever! (Engineering, like Computer Science, is a surprisingly social endeavor - most products come from development teams.)

Reflecting the fact that this is primarily a laboratory course, many of the customary requirements of engineering courses have been scaled down. There are only two lab reports and one schematic to compose, and these will be graded on a satisfactory/must be redone basis. The computer simulation lab and the logic analyzer exercise will be at least partially self-documenting. There are only minimal fixed deadlines for individual assignments. While there will be a midterm and a final examination they are intended to encourage you to think of the lab material in a more general framework. Your grade will be calculated as an average of your lab grade and the two exam grades. We expect the exams to test how well you respond to freshly posed problems of a similar nature to those found in the labs.

In the lab you design and build a particular circuit and then demonstrate that lab solution to one of the TAs. **You must be prepared to show a schematic of your system done in your own writing or own schematic entry and to explain it to the TA.** After studying your approach, the TA will ask you a question about its performance or its design; if you answer the question correctly, the TA signs your scorecard, and you are free to move on to the next lab challenge.

In Engineering 163, there are no specified times when you have to appear in the lab. You may design and build your circuit wherever you like. We will provide the necessary equipment to test and debug your circuit and will also provide a group of well-qualified teaching assistants to help you with any unusual problems. If you run into problems with malfunctioning equipment or software, please bring it to the attention of a Head TA or to me.

The liberalized format of Engineering 163 presents you with a number of challenges quite apart from actually building each lab assignment. By the time you take this course (typically in the junior or senior year) we expect that you have learned to think creatively, to work consistently, and to budget your time. The minimal deadlines established for completion of the labs force you to set goals for your own performance. While many design techniques are discussed in class, it is your responsibility to put broad concepts into practice. Making choices is part of the process.

Despite appearances, Engineering 163 is not designed to be a grueling entrance exam for the hardware designers' club. It is designed to be a broad introduction to many facets of digital de-

sign. It should give everyone a chance to appreciate the difficulty of implementing straightforward tasks in a constrained environment. The constraints include time and space, as well as materials and reliability. The labs often demonstrate that seemingly elegant paper solutions do not always work well when they are translated into hardware. While this can lead to frustration, it should also lead to a deeper understanding of problem solving and of the relationship between the physical and logical worlds.

Class will meet on Mondays and Wednesdays, from 3:00 PM until about 4:20. The lab will be open day and night most of the week with TA coverage from 9:00 AM until about 5:00 PM, with convenient evening hours twice a week. The lab will be in the Hewlett Electronics Laboratory, room 196 of the Giancarlo addition. You are not required to attend class, although I hope that the relatively small class size will make the lectures more attractive. Similarly the relevant readings are optional. We do not wish to waste your time by forcing you to sit through boring lectures or to read irrelevant ramblings meant to up the page counts of books. However, the TAs are not obliged to repeat lecture or text material to exempt you from learning it on your own.

2. Evaluation

For a variety of reasons, the evaluation policy used in EN-163 has changed over the years, following the convictions and whims of the instructor in charge. Some take a rather liberal approach, allowing your circuits alone to represent the depth and breadth of your digital logic design knowledge. Others of us use points-oriented grading schemes of Baroque complexity that make your final grade dependent on how well you score on mid-term and final examinations, in addition to the number of design exercises you complete. Each of these approaches has its benefits, and each has its drawbacks.

I follow a grading policy based on a point system. There will be a total of 165 points available, 65 points from exams and 100 points allocated for completed lab work as tabulated below. Passing performances on both exams and lab work **separately** are required to pass the course, but a curve will be applied to your total grade to determine the letter grade reported to the Registrar. I expect that an **A** will require about 135 points and a **C** will cost about 80 points. These boundaries have served me fairly well over the last couple of years.

2.1. Grades

Your lab grade is based on the number of lab challenges you complete. There is no partial credit on the lab grades. Instead you will receive points for each successfully completed lab challenge or piece of written lab work. There are 16 hexadecimally numbered labs of which you may do up to 15, including only 2 from D, E, F. With the schematic drawing and the logic analyzer points, there are a total of 100 possible lab points. The distribution of points is:

Lab 0 through Lab 5: 5 pts. ea.

Lab 6 through Lab 8 and software simulation Lab 9: 6 pts. ea.

Lab A through Lab C, and up to two from the group Labs D, E, and F: 7 pts. ea.

A schematic for either Lab 1, 7, 8, or B: 6 pts.

Logic analyzer measurement challenge (see section 9. 6): 5 pts.

To pass the course, however, you must have passing grades on **BOTH** the examination and lab parts of the course. A passing grade on the examination part of the course is 48% of the sum of the maximum number of points possible on the exams. A passing grade on the lab part of the course is **53 points**, which **must include credit** for Labs 1 and 2; for one lab from the set 7 and 8; for Lab 9 and Lab C; for the Logic Analyzer Challenge, and for a schematic of either lab 1, 7, 8, or B. **THERE WILL BE NO EXCEPTIONS TO THIS POLICY.** Please note that the use of a logic analyzer is mandatory. The laboratory has a superb collection of new instruments and their use is to hardware what a debugger is to software. You don't know much about either discipline without knowing its fundamental test tools.

A word of advice: labs 7 and 8 involve systems with mixed analog and digital signals. Many students find these labs time consuming because they find it difficult to be systematic about dealing with the analog part. The letter series labs may also take significant amounts of time. However, they are purely digital. If you find that the first analog-to-digital converter lab that you do takes too long, then do the letter labs next, particularly the required Xilinx lab, Lab C. Only do the other A/D lab if you have time toward the end of the course.

Scuttlebutt variously suggests that either lab 7 or 8 is much easier than the other. I believe they are about equal difficulty. A careful and systematic approach is critical to success within reasonable time limits on either of them. The use of simulation for the digital section in advance can save enormous amounts of time on these labs and will provide a basis for logic analyzer measurements comparing expected and actual performance.

Section 7 of this manual contains a scorecard for recording your performance on each of the EN-163 labs; each time you successfully complete a lab, a TA will sign the line associated with that challenge and make an entry on our backup sheets. To receive your grade, you must turn in your scorecard at the end of the semester, which is defined as no later than 5:00 PM of the day of the scheduled final examination. **Do Not Lose Your Scorecard!!!** We try to keep duplicate records, but the backups are hard to search, and there are **NO guarantees.** People who arrive empty-handed at the end of the semester will receive No Credit for the course.

2.2. Partial Credit

The nature of EN-163 precludes the assignment of partial credit for any of the Labs. If, for example, you are told to build a circuit that displays the digits 0, 1, 2, 3, 4, 5, 6, and 7 in a particular order, and you offer a circuit that displays all but the digit 3, you get no credit for the Lab until you troubleshoot the circuit and get it working as requested. This policy just reflects the nature of digital system design, and it is one from which no deviations will be made.

2.3. Incompletes

No grade of *Incomplete* for unfinished labs will be given unless you present evidence from a physician or a Dean of a serious and protracted medical problem. Because of the nature of the course, there is no way of dealing with students after the end of the semester. The TAs are gone, the lab space is reorganized and used by other courses and other students, and all parts must be checked, cleaned, and inventoried. If you need to pass EN-163 to complete the departmental requirements for a specific Brown degree, make sure you *at least* qualify for a grade of C. I have no sympathy for someone who shows up at the end of the semester with a nearly-blank scorecard in hand, and claims that he or she needs this course to graduate. The time to consider the relative importance of EN-163 to your educational experience is ***now.***

2.4. The Fault Tolerance Question (FTQ)

When you demonstrate a circuit for a TA, you and the TA will spend a couple of minutes verifying that the circuit meets all requirements. You then show a schematic and explain the operation. (NOTE: This procedure is another change from prior years.) The schematic does not have to be machine generated although that would be preferable. It does have to be both neat and accurate. If you are asked where a particular connection point on the schematic is and reply that well you changed that, then you get to go away and do the documentation over again. The TAs get to decide whether the drawing is neat enough to understand. The TA will then check your breadboard serial number and ask a Fault Tolerance Question (FTQ). These questions may take one of three forms, at the option of the TA. For the first type of question, the TA will choose one wire in your circuit and ask what, if anything, will go wrong with your circuit after the wire is removed. For the second type of FTQ, the TA will introduce a problem into your circuit, leaving you the responsibility of finding and correcting the problem. The third type of FTQ will allow the TA to ask you general questions about the operation of your circuit, such as "What will happen to the output of the Q1 flip-flop if both inputs to this NAND gate are grounded?" It is a TA's prerogative to ask for documentation before an FTQ.

These notions of an FTQ are based on TTL implementations of small systems. The labs based on programmable devices (CPLD or FPGA) pose special problems for framing good FTQs. In preparation for an FTQ or even for certification of compliance with specifications, you **must make a printed copy of the ABEL file, the Xilinx schematic, or the VHDL source file in addition to the wiring diagram.** At the least, the TA will want to see this and ask how you designed it. She may ask variants on the hardware FTQs based on changes to operation of the circuit that will be caused by a change to the schematic, VHDL, or Boolean description and may expect you to compile and demonstrate the expected result on the spot.

If you answer the FTQ correctly, the TA will sign your scorecard, congratulate you, and you can celebrate! (As a precautionary measure, you might want to be sure the TA enters your success on the backup sheets just in case you lose your scorecard.) If you do not answer the FTQ correctly, the TA will be obliged to ask you **two more** FTQ's. For every FTQ that you miss, two more must be asked. If several FTQ's are answered incorrectly, we will begin to wonder how you can know so little about a circuit that you designed, built, and tested yourself. We will begin to wonder about plagiarism. If you miss three FTQ's in a row, your circuit breadboard will be tempo-

rarily confiscated and you will be referred to the head TA for further consultation. If you are guilty of demonstrating a circuit that someone else designed, you will automatically lose credit for that lab. (If this is a required lab, you may not be able to pass the course.) If a second offense occurs, standard academic discipline procedures will be initiated.

We are aware that a certain amount of shopping around is done by students looking for a soft-hearted TA. To limit this behavior, no one can have more than five labs checked by any one TA. All the TAs will have been told that we regard FTQ's as serious matters, and no TA has to be a patsy.

For most of the labs, once an FTQ (or a series of FTQ's) have been answered correctly, you have met all the requirements for that lab. There are a few exceptions to this rule. Labs 2 and 6 involve the measurement of important properties of your chips, such as power dissipation, logic levels, propagation delay, etc. They require data and the answers to some questions to be handed in as a report in lieu of an FTQ. These write-ups will be graded only on a *satisfactory/must-be-corrected* basis and will be handed back only after all reports for that lab were due. **You must both take the data and write the report individually!** It is perfectly okay to give each other advice and information, but you are responsible for doing your own work.

Similarly lab 9 on simulation and the schematic drawing by computer design entry require documentation in lieu of FTQs. All documentation is handed in through a box in room 196. We still have to set that up but will mark it with an appropriate sign.

An FTQ can be fairly difficult sometimes. We will try not to choose bizarre wires or concoct impossible-to-analyze situations. In any event, you are responsible for giving a **complete** account of impending faults in your circuit *before the wire is pulled or the software program modified.* If you mention that X and Y will go wrong, but only X goes wrong when the wire is pulled, your answer is incorrect. On the other hand, if you mention that X and Y will go wrong, and X and Y *and Z* actually go wrong, then your answer is also incorrect. In some cases, you may want to list contingencies: “If the input goes HIGH, then X will occur, but if the input goes LOW, Y will occur.” This may be particularly important for CMOS circuits for which an uncommitted input is not in a well-defined state. Vague answers, like “the circuit will not work” or “it would take a scientist to figure it out” are not acceptable. Neither is going to lunch; once an FTQ is asked, you must answer it during the same session. Any interruption in the FTQ process between a question and an answer will be recorded as a question failure. You may not break off a losing streak of questions without leaving your breadboard and a record of the missed questions with the TA for passing on to the next TA who takes up the questioning. You may only break off such a streak for good cause, and ignorance is not a good cause. Be sure you understand your system before you ask for an FTQ. The TA may pass you on to his or her successor if there is a shift change during your questioning.

It is the privilege of the TA to choose an appropriate FTQ. If you feel that you have been given an unfair question, first try to work out an answer. If you fail the question, you may appeal it to the head TA; if this brings no satisfaction, you may bring your complaint to me. If I feel your complaint is justified, you will be asked a single replacement question; otherwise you will be referred back to the original TA who will ask the normal two-question FTQ follow-up.

Because an FTQ may be non-trivial, take a reasonable amount of time to answer a question. Allocate *at least* fifteen minutes to this process. Do not let the atmosphere of the lab make you anxious - getting the first question correct can save you a lot of time. We expect that over the course of the semester you will spend from 2 to 4 hours answering FTQs.

2.5. Documentation

Proper documentation is an essential aspect of the design process; I expect you to get in the habit of specifying your circuit completely while you are devising and building it. Circuit diagrams, written circuit descriptions, timing diagrams, truth tables, etc., are the counterparts in hardware design to the documentation of computer programs which is so much a part of satisfactory programming. To encourage good habits in this area, we are requiring a carefully done schematic for one lab you have completed. (You may not hand in documentation for a lab that does not work! We may reject a schematic for a bad design even though you may have gotten it past a TA. For example, some people carefully adjust components to get a poor design to work for a while.) The schematic must be done using the *DxDesigner* computer aided design (CAD) package. This manual includes a brief description of the standards that these drawings must meet and some examples of appropriate drawing techniques. We may even **require redrawing** if your schematic does not meet some of the standards **for an esthetic layout** even if the interconnections are correctly expressed. Also, you cannot expect much professorial or TA help with a non-working breadboard unless your working drawings are reasonably complete and legible. In fact, the TAs have been instructed *not* to answer questions about circuits unless they see a reasonable amount of documentation and to require a schematic as part of the evaluation process. (As a friendly hint for lab 5 and up, it might pay to do schematic entry and simulation before coming to the lab or pulling out your breadboard. Done carefully, this will allow you to exchange a little time at the computer for a lot of time in room 196.)

2.6. Deadlines

Many years ago, I experimented with deadlines for general groups of labs. As a result of its success in retaining the sanity of both students and TAs during the course, I continue the tradition. The lab groups, and associated completion dates, are:

Group one:

Labs 0 through 3 may be checked off only until October 13, 2004.

Group two:

Labs 4 through 9 may be checked only until November 17, 2004.

Group three:

Labs A through C, and the schematic diagram requirement may be checked off only until December 7, 2004.

You may obtain credit for the labs within each group *only* up to the posted date. (Note that labs may be completed as soon as you wish, but they must be completed by the given enpumpkina-

tion date.) The last week before the final exam will be a grace period. During this week, you may have up to one additional lab from each group checked off.

Labs D through F may be checked off at any time until the last day of the semester. This year, the Engin 163 exam comes fairly late in the exam period, on the afternoon of Monday Dec. 20, 2004. I will allow labs to be checked off until noon on Friday Dec. 18th since that is the latest that grades can be integrated before I have grades due. Also keep in mind that **TA assistance and availability for FTQ's decreases during exam period** for the obvious reason.

The completion groups were instituted for several reasons. First, EN-163 is allocated a fixed amount of lab space, a small number of TAs, and a finite amount of circuit testing, trouble-shooting, and demonstrating equipment. It has been our experience that most people (including me!) tend to procrastinate when not confronted with fixed deadlines. Without deadlines, some students fall hopelessly behind by the end of the semester; we are then inundated with a variety of excuses (some more creative than others) explaining why a particular student should be allowed to pass the course while not meeting the stated requirements. The less original of these excuses deal with crowded lab space, insufficient number of power supplies, non-existent TAs, etc. To short-circuit these excuses, as well as to distribute the workload for TAs more evenly, we started a deadline system.

A second compelling reason for the institution of the deadline approach, however, is that by concentrating on a group of labs in a given number of weeks, help sessions can be scheduled as necessary to assist those who appear to be having difficulty. By monitoring progress on individual labs, we can get an idea of how people are coping with the given challenges, some of which have been altered from previous years.

Note that students who thrive on the excitement generated by trying to complete the course at the last possible moment will especially appreciate and benefit from the deadline approach, since there are now four “last possible moments” to contend with instead of only one!

2.7. The EN-163 Exclusion Principle

For each day that the lab is open, **you may have only one lab signed off**. This will still allow ample time to complete as many labs as you wish - after all, the typical academic semester has at least 70 class days but only 15 or so labs have been defined for the entire course. The Exclusion Principle has two purposes: one, it prevents one individual from monopolizing TA time, and two, it provides one more incentive for you to distribute your circuit-demonstrating eggs into several daily baskets. Deviations to this policy will not be considered. Please note that this has implications for how much you can hope to get done in the end-of-semester grace period.

2.8. Lab Schedule

The EN-163 lab will be in Giancarlo room 196, the Hewlett Electronics Laboratory. It will be open every day while the University is in session. Use the door nearest the ramp out of the building as its electric lock will be open. TAs will be available for some 40 plus hours a week in-

cluding at least two evenings during the week, probably Wednesday and Thursday, from 7:30 to 9:30.

We will try to keep a TA schedule posted in the lab and on the web, telling when each TA will be available for troubleshooting help and for checking off labs. No TA will be assigned to work when class is in session. Also, although we will try to keep most lab slots staffed with at least one TA, there will undoubtedly be times when the lab has a TA scheduled but he does not turn up. The TAs have job interviews and other personal obligations and while I ask them to arrange for substitutes, this is not always practical. Do not schedule yourself so tightly that you miss getting a lab signed off because a TA overslept. The responsibility for getting labs checked off in a timely fashion is yours, not the TA's.

You are encouraged to come to the lab at times when the crowd is small (such as 9 AM:) you will get more attention and have shorter waits for lab sign-offs.

2.9. Teacher Evaluation

Besides my obligation to evaluate you, you will have an opportunity to evaluate me at the end of the semester. The Division of Engineering arranges for a secret ballot evaluation of teaching. It distributes an official form that is used for class assignment, salary adjustment, and (occasionally) tenure review. There is also a form designed to measure whether I have met the course goals for ABET. (Those goals are listed on the handout I give the first day of class.) In addition to the ABET form, I may hand out a second evaluation form that will give you a chance to comment directly to me in greater detail on my performance. Finally, I distribute forms for the *Critical Review*. I take these reviews quite seriously, and would appreciate your doing the same. (By the way, I do not read any of them until after the final exam. The Division even has a policy of retaining forms in the Dean's office until after exams are over and grades are submitted.)

2.10. Collaboration

Unlike some courses, you are welcome to collaborate with your fellow students during the semester. Discussing problems with a friend is a good way to troubleshoot your circuit, and can be a source of self-improvement for both of you. Be sure that you completely understand any advice offered by anyone, however, since this advice may be wrong - providing even more frustration than the original problem that you yourself devised. Make sure that you will not be stumped on an FTQ.

In general, two kinds of collaboration can be distinguished: *soft collaboration* and *hard collaboration*. Soft collaboration is a perfectly acceptable process of seeking and offering advice on the design of a particular lab solution. We assume that you will understand any advice given, and hence will benefit from the information exchange. Note that while you may get design help from a friend, **you must use your own kit to construct the solution, you must generate your own documentation, you must be ready to explain your system with understanding, and you must answer your own FTQ.**

Hard collaboration means the use of another person's circuit board, wiring, or documentation during your FTQ. It also encompasses blind copying of another's design without sufficient understanding to pass the FTQ process. In labs requiring computer files, you must generate your own files individually, making your own choice of variable names, comment records, indentation, etc. **All the data and reports for labs 2 and 6 and the schematic must be done individually.** I will take a very dim view of nearly identical reports. Copying is hard collaboration! None of the forms of hard collaboration will be tolerated. If you feel compelled to use a friend's design for a given lab challenge, you must at least go through the process of making your own schematic from scratch not Xerox, wiring up the solution (including going through the computer entry process for PLD or FPGA designs), and debugging it yourself! All the wiring and troubleshooting associated with lab challenges must be performed on the circuit boards issued to you at the beginning of the semester. TAs will be required to match the numbers found on your scorecard with those on the back of your boards. If you are found using someone else's board, you will automatically lose credit for the lab you are trying to demonstrate. (If this occurs during one of the labs required to pass, you cannot possibly pass the course.) You may also get your friend in trouble for loaning his or her boards. In serious cases, you may be referred to the appropriate Dean for action by the Academic Code Committee and will probably receive a grade of NC for the course. Penalties for tampering with someone else's board or kit will be dealt with even more stringently.

Improper collaboration, besides wasting the three thousand-plus dollars you pay for taking this course, is a serious matter and will not be tolerated.

3. Getting Started

With this lab manual, you will also receive your own chip set. Look over Lab 0, work out a solution on paper and test it conceptually until you are convinced it will work. Then assemble and test it. Be especially careful about connections to the power supply. The easiest way to damage a chip is to get the power supply connections backwards. The +5 volt supply lead must go to the VCC lead of each device, and the ground or 5 volt return lead must go to the GND pin of each device. If you are using LVTTL parts, be careful not to apply an overvoltage. Failure to be careful may result in wholesale damage to your chips. We realize that some of your chips may be defective as a result of prior use, but we will not replace components damaged by carelessness without some charge. Another point to be especially careful about is not to connect the output of a chip to +5 volts or even worse connect any pin of a logic chip to plus or minus 12 volts as may happen in labs 7 or 8. (Chips usually are more tolerant of outputs accidentally connected to ground.) **In the labs the use the CPLDs, which is half the labs and most of the required one, you must program the device before you connect it to other wiring. We have special cables on the power supplies so you can program the XC9572XL and then remove the power cable and substitute yours.**

Nota Bene: Massive destruction of chips or melted or burnt areas on protoboards or keypads will result in appropriate adjustments to the **price at which we will buy back your kit.** Your keypads are mechanically very robust, but they can be ruined easily by using them to short out the power supply, i.e. to connect VCC to GND. This melts the plastic film under the buttons, causing them to stick. You must **use limiting or pull-up resistors** with the keypad. We are not sympathetic to damaged keypads.

When you are sure the circuit works properly, collar a TA, demonstrate it, explain it, and answer an FTQ. With this signoff, you are well underway! We strongly recommend that you try to get Lab 0 finished during the first real week of classes. Remember the deadline system, and remember that the lab gets busier and the challenges harder as the semester progresses.

If Lab 0 has you baffled, if you do not know who Herr Georg Ohm was (and why he is remembered -- his lab equipment is carefully and reverently preserved in the Deutches Museum in Munich), if you do not know the difference between voltage and ground or capacitance and resistance, do not despair. Help sessions are scheduled throughout the semester. The early sessions will provide a practical explanation of electronics, and should allow you to get started. Among other things, we will explain how a breadboard works, how to strip wire, how to distinguish between capacitors and resistors, and how to use power supplies and logic probes. One purpose of EN-163 is to remove the mystique associated with digital hardware. Throughout the course you will have every opportunity to have your fears explained away; your responsibility is to take advantage of the help available.

4. Circuit Construction Guidelines

4.1. General Hints

Be a little compulsive. Organize the components in your chip set so that you do not waste time looking through the entire lot each time you need a particular chip. If you sort them in numerical order, you will easily find any part you need. You should also lay down a piece of aluminum foil to press onto the pins of the CMOS chips; this will short their pins together and prevent the chip from being destroyed by high-voltage static electricity. (CMOS parts including CPLDs are the most vulnerable parts, but TTL parts can be damaged too. The pink foam we sometimes use for kits is pretty good without aluminum foil, but be careful if you substitute other materials.)

Another suggestion is to make a set 3x5 index cards that summarize the pin connections and logical functions of each of the chips in your Bag-o-Chips. This may cost a bit of time now, but it will save time in the long run. These cards will serve as handy references on how each chip behaves, and will allow you to avoid leafing through your lab manual each time you need to know how the chip works (especially during the answering of FTQ's!). The automated form of this is to make use of the *DxDesigner* libraries and draw everything on-line.

In general, you should start a lab challenge by working through the requirements of each subcircuit. Construct truth tables, if appropriate, showing the relationships among the various inputs and outputs of your system. **Try to decompose one large problem into several smaller problems**, keeping in mind the limitations of your chip selection and the need to integrate each of the pieces into a whole. Once you understand what needs to be done, work out a complete pencil-and-paper solution. Hand simulate the circuit to see what happens during various combinations of inputs. Make sure that most of your logic errors are caught while you are still working in a pencil-and-paper mode; it is easier to erase a connection thane a circuit. You **may wish to use *DxDesigner*** to turn your hand sketch into a full schematic. This will help with accuracy of pin assignments, with finding missing signals, and with legibility. Further testing by simulation at the design stage is standard practice and is much recommended. With practice, the time to draw and

simulate a circuit is short enough that it is completely offset by the time saved in construction and debug. Remember that neat drawings help you work out ideas and build things with minimum errors. Per the comments above, this effort will also guarantee better help from TAs or from me.

After designing a solution, build your circuit gradually. Isolated sub-sections of the design can then be tested apart from the rest of the solution. If your circuit evolves in a step-wise manner, you will be able to understand its operation better, and you will be less likely to be baffled by a hard FTQ.

When you assemble your circuit, do it neatly with wires that are not so long that they rise above the breadboard in a confused mass. Make sure that only a minimum of bare wire is exposed throughout your circuit; this will cut down on intermittent problems that result from improper insulation. **Do not make wires so short** that they loop tightly over the top of individual chips, however, since this will prevent you from exchanging a chip if it fails. (This also prevents testing it or attaching a logic analyzer to it if you *suspect* it has failed. Logic analyzer connections are usually made with a clip that fits over the chip. Wires over the chip prevent doing this.) The CPLD board has a special connector to make connections to the logic analyzer. Assigning pins to make that connection give a proper analyzer display can help too.

When you wire a circuit, use a consistent color-coding scheme. For example, make all +5v lines red, all ground lines brown (or any other Earth tone, for that matter), etc. To aid in troubleshooting, use a variety of wire colors throughout your circuit so that co-located wires can be easily distinguished. For parallel lines of information (busses) you may want to use wire colors corresponding to the resistor color code.[10]

4.2. The 163 Lab Environment

Besides being the place where your circuits will be tested by TA's, room 196 is where you can go to use a power supply and logic probe for troubleshooting, to talk with fellow students about the lab challenges, and generally to bask in the high-tech atmosphere of the new building. Do not, however, make a nuisance of yourself! No loud social gatherings; no smoking; no drugs; no food fights or practical jokes. There are classes just across the hall for much of the day. Observe normal lab courtesy. Turn off equipment when you are done. Return all wires and cables to their normal places. Clean up little wires or scraps of paper. Do not remove anything from the lab (except for your lab kit) without permission of the technician in charge (Arpie). If you break something, or notice something is broken, notify a TA. **OF PARTICULAR IMPORTANCE: place all parts of any broken leads** for the scopes or analyzers in the boxes provided for them. These leads are very expensive ($5 apiece for the analyzers) and can often be repaired at no cost.

Get in the habit of checking the whiteboards for the latest news concerning the course. If there is a change in the lecture schedule, or if some particularly useful lab information is discovered (such as the opening or closing of the lab at non-standard times), it will be posted there and on the class web-site.

[10] The resistor color code is a series of color and digit assignments. See the back of the data sheet section of this manual for the code's definition.

For some labs, you will want to use an oscilloscope to monitor rapidly changing voltages. There are quite a few new digital oscilloscopes. If you do not know how to use one, please ask a TA. He or she can explain time bases, vertical calibration, triggering, dual trace operation and other features. Because TAs are only a year ahead of you, they may not know all their details. The manuals for them are in the lab -- be persistent. Please be careful when using the scopes -- especially with the attached probes. These probes are easy to damage, and cost $ 110 each to replace. Similarly the logic analyzers have delicate probes that need care in handling. Each little wire from the analyzer is five dollars to replace, and the ends break off them very easily. We have some spares but cannot afford huge quantities. If one breaks for you, please **PUT THE BROKEN END AND THE WIRE ITSELF** in one of the boxes we keep for that purpose. We can easily repair the probes if we get the pieces back! There are a number of manuals in the lab with the analyzers that discuss the instruments in detail. On the other hand, we do have some leads and you shouldn't have to jury rig everything. Ask the TAs to get more leads if the supply is getting low.

There are 16 PCs in the lab to support the later labs. These are the ones on the workbenches and they run from the same server as the Computing Facility uses. Please resist the temptation to customize the desktops or to install games, email, etc. It goes without saying that the University strictly prohibits the installation of pirated software. All such entrepreneurial effort just makes our lives more complicated in keeping a common core of dedicated software working properly.

If you are unfamiliar with any other pieces of equipment in the lab, ask a TA for assistance.

4.3. Logic Probes

All of the power supplies are equipped with logic probes that can quickly tell you the voltage levels of any point in your circuit. The LOW LED is on for voltages less than 0.8v; the HIGH LED is on the voltages greater than 2.4v. If the logic probe measures voltages between 1.0v and 2.2v (or if it is unconnected) no lights will glow. A third LED will light if the probe detects a pulse train.

4.4. Power Supplies

The 163 lab room is a spacious place where approximately sixteen circuit testing stations have been set up. The heart of each of these stations is the power supply. To provide life to your circuit, connect a wire from the +5v terminal to the appropriate slot on your breadboard and a wire from the ground terminal to your ground pins. Always double check that you have the proper polarity of power *before* turning on your circuit. **Reversal of the power supply leads could damage all of the chips on your breadboard!**

For your own reasons, you may wish to avoid the bustling atmosphere of the 163 lab and instead test your circuits in the privacy of your own room. The advantages of this approach are that you will be able to control the noise level around you, and you will also (presumably) be able to control the size of the crowd waiting in line for you to finish. The disadvantages of this approach are that there will typically be no TA to offer advice, and that an alternate source of 5 volt power is necessary. There are a number of solutions to this latter problem.

Because voltage fluctuations cause circuits to malfunction and because TTL circuits draw large currents, any approach based on batteries is a marginal solution to your problem. The parts for a simple power supply can be purchased from Radio Shack for about $20. If you are interested in building one, I can give you a circuit diagram and some guidance on how to do it. For more money, it is possible to purchase a fully assembled 5v power supply that will deliver 1 or more amps of current. For example, a 5v/5A supply with auxiliary +-12-15v outputs costs about $ 90.

4.5. Bypassing

No power supply is perfect in the sense that a constant potential difference of exactly +5v is present between every pair of connections to VCC and GND on every extension of every wire of the power-to-ground system. The relative immunity of digital systems to extraneous potential variations (noise!) is a major reason for the ubiquity and power of digital logic, but that immunity is not infinite. Because of small series resistances and inductances associated with the power supply and its wiring, noise, that is, transient current spikes generated by pulse circuits or by TTL gates changing state, can cause voltage changes on the power wiring and make your logic circuits malfunction. Sequential circuits -- counters, flip-flops, etc. -- are especially sensitive. Other noise may come from equipment turned off and on somewhere else in the lab or from the 60 cycle noise generated by fluorescent lights, or from *reflections* of signals sent over long wires in your circuit. (In fact, a major subdiscipline has grown up around designing out problems with power and signal distribution. It goes by the name of signal integrity analysis and design.)

To protect your circuit from transient voltage spikes on the power supply lines, you can "bypass" or "decouple" the power supply by placing capacitors directly from power to ground at key places in your circuit. Key places are typically clock pulse generating or clock pulse receiving chips, such as oscillators, multivibrators, counters, or flip flops. You have several 0.1μF capacitors among your parts that can be used for bypass. More are available if necessary. Because it takes an appreciable amount of charge to change the voltage across such a capacitor, these will divert some of the current causing voltage transients. **Be aware of the possibility of bypass problems when troubleshooting your circuits**! If a counter seems to skips values or a flip-flop changes state for no logical reason, a bypass capacitor from VCC to GND at that chip may be necessary. Please be aware too that the 74ACT04 chip may be a source of extra noise and may require its own bypass capacitor. All except the simplest labs require at least a couple of bypass capacitors. One other possibility to check when troubleshooting an erratic circuit is that the supply voltage **at the chips** may be less that the 4.75 volts required for proper operation. Measure it directly at the chip with a multimeter.

4.6. Debouncing

Mechanical switches, such as those on your keyboard, are another source of unwanted pulses. Suppose you press one of your keyboard buttons. Initially, the two internal wires are separated (or form an "open circuit"). Pressing the button will close the circuit, and cause contact between the two wires. Instead of making a single continuous contact, however, the two wires actually make several bouncing contacts on a microscopic scale of millisecond duration. Looking at

the event with an oscilloscope you may see a series of pulses instead of the single pulse you desire. Figure I illustrates this problem and shows a method of *debouncing* a 2 wire push button.

The solution idea here is similar to the solution of the bypass problem. Use a capacitor to absorb some of the current associated with the unwanted pulse activity. In Fig. I, we have a large resistance connected from the power supply to the gate of an inverter. If there is no other current at that logic gate input, the capacitor will remain charged to the power supply voltage. If the gate input is grounded, as it can be with the push of a button, the gate voltage will change immediately. If mechanical bounce occurs, and the contacts momentarily part, the capacitor will recharge with a time constant of about $R \cdot C$. The actual time this circuit takes from the opening of the switch contacts until the signal V_A crosses the inverter threshold depends on the type of inverter. In lab 2 you will find that TTL gates have substantial current flowing thru input terminals. These currents charge the capacitor faster than the resistor alone would. With 74LSxx series gates, C must be several microfarads to get a 10 ms. hold time. If the gate is a CMOS gate, then it usually has no steady input current. In that case, an RC time constant of 10 ms as in Fig Eq II would result in a hold time of 3 milliseconds. [The exact result is $\Delta t = -R \cdot C \ln\left[(V_{CC} - V_{THG})/(V_{CC})\right]$.] If, well before the 3 milliseconds elapse, mechanical contact is made again, the gate voltage will again drop to 0v and be kept *below the threshold* that causes a change of state for the gate. Thus, the output of the gate will remain low in spite of small changes going on at the input due to mechanical bounce. You may first encounter the need to put this theory into practice in Lab 4.

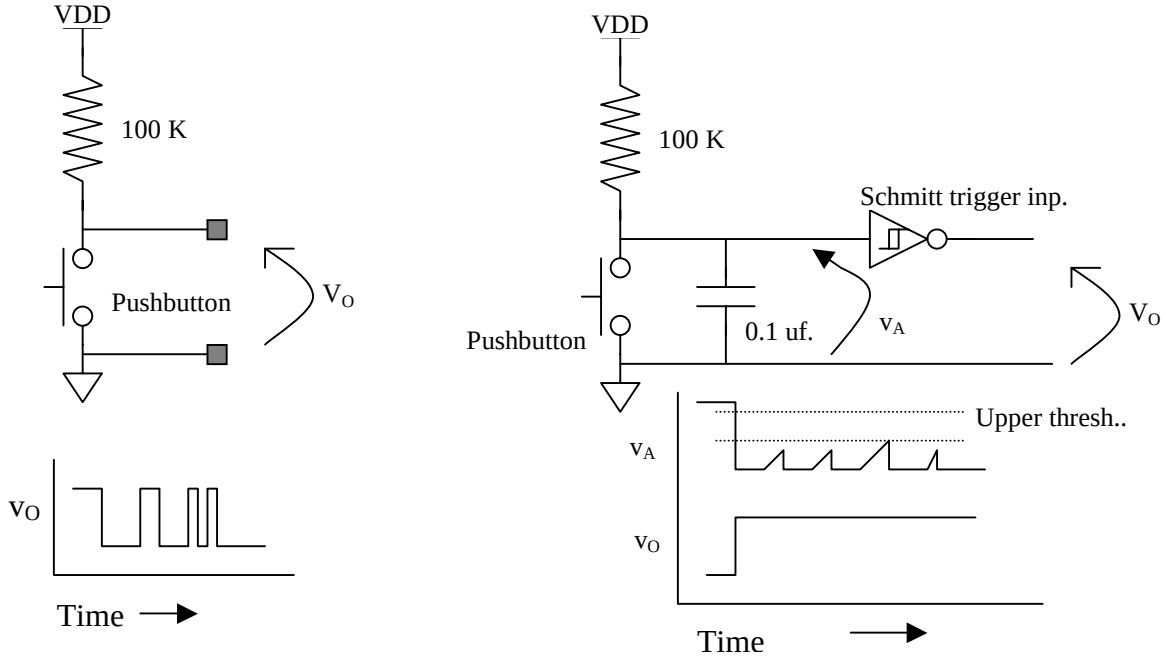


Figure I: Debouncing a single-contact mechanical switch or button.

When the mechanical signal source is a switch instead of a pushbutton, an alternative method of debouncing the switch signal is available. Figure II shows two variants on the technique, one using the preset/clear lines on a flip-flop and the other using two NAND gates con-

nected as an RS flip-flop. The idea is that the SPDT switch bounces on the make or break of the moving contact with either of the two stationary ones but does not bounce back and forth from the one to the other. The flip-flop responds to the first time each contact is made and ignores subsequent bounces until the other contact is closed again.

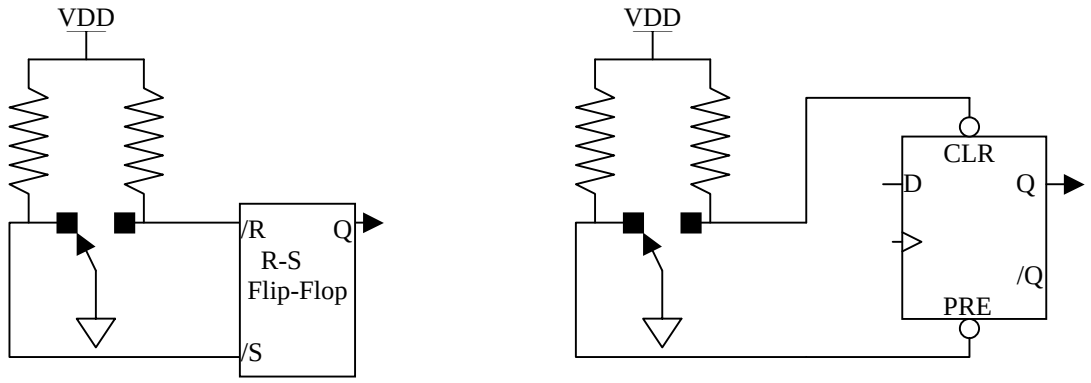


Figure II: Using flip-flops to debounce an SPDT switch.

4.7. Digital Circuits

Digital systems have an inevitable tendency to expand. One can almost always find a use for a still bigger computer or controller. The success of designers in meeting this amorphous demand has depended among other things on agreement for the interfaces between chips. Without such agreement, it would not be possible to interconnect chips from different vendors. You have chips in your kit which meet common standards while being made from very different types of transistors. In labs 2 and 6 you explore both the nature of the interface specifications and the differences between the two chip types. We will talk more about these issues in class, but since some of you will want to tackle the lab before I get to the subject in class, I thought a few words about chip types might be in order here.

In the beginning, Bardeen, Brattain, and Shockley created the bipolar junction transistor. (Our world began in December 1948.) For twenty years there was no other alternative. During that time Texas Instruments Corporation introduced the first really successful family of digital integrated circuits, the so-called TTL devices, which they named the SN7400 series. The success of these devices established their power supply requirements and their interface definitions of high and low levels as the *de facto* standard of the industry to this day. Their devices have since gone through several generations of development, getting faster and more economical and using less power. Each new generation of parts was distinguished by sandwiching a new letter into the part name, e.g. 74S00, 74LS00, 74ALS00, 74F00, etc. You are using mostly 74LSxx parts because these are about the least expensive and most forgiving parts on the market.

In the late nineteen sixties, transistors of a new type, the MOSFETs, came on the market. Today's descendants of those devices, usually CMOS circuits, include the 74ACT04 chip from your set, the CPLDs and FPGA you use, and virtually all memories. (A “C” in the middle of the

part number often indicates that it is a CMOS device. The C in CMOS stands for Complementary and indicates that the device uses two different types of MOSFET which are complementary to each other in function.) Even the 74LV14 chips tested as part of labs 2 and 6 are actually CMOS devices. The advantages of these parts are low input current, low power usage at slow speeds, and a very high density of transistors. This last means high potential functionality for a chip.

Originally, the TTL parts were much faster and rather more expensive. Other parts based on bipolar transistors were even faster, but all types of bipolar devices used much more power. Recent developments have blurred many of these distinctions. Your 74ACT04, for example, is almost as fast as the fastest TTL parts, but its power advantage is marginal when used at full speed. The problems of noise on the power distribution lines of systems using 74ACT parts are about the worst of any generation of commercial part and stimulated industry discussion of new interfaces and new means of packaging chips. After the initial introduction of 74ACT parts that copied the pinout of the standard TTL parts, companies found that the power supply problems were so severe that a new set of packages to reduce the problem were necessary. Traditionally the other advantage of TTL lay in its ability to drive heavier loads. This advantage has also eroded. At the same time system level considerations are driving computer design toward new, lower voltages. A standard of 3.3 volts for power supplies is now widespread, and parts requiring down to 2.0 volts are already available. Dense memory chips and advanced processors already use supplies of 1.2 to 1.8 volts for their core logic. Standardization is elusive and multiple supply voltages have become the norm. The supplies in the lab provide the 3.3 volt standard output now. New technologies have emerged that combine bipolar and CMOS devices, and even newer devices promise advantages by adding some germanium to the silicon that is that basic material of most systems now. Overall the commercial situation appears very fluid.

From your point of view, what we hope you will notice about the devices in your kit are the input currents, and the power versus frequency relations. One of the important results of the difference in input currents is that TTL devices have uncommitted inputs which are always high, while CMOS devices have undetermined values for uncommitted inputs. Also, this behavior and the high power supply noise of the newer CMOS parts can make them not interchangeable with TTL despite having nominally identical interface specifications.

It is good construction practice never to leave an input floating; connect it to either VCC or GND. Good design tools will flag such open pins as errors during design checks. Also with your CMOS parts, as with the TTL parts, it is very important to bypass power connections.

5. Design and Troubleshooting

5.1. Design

You may find throughout the course that inspiration is not enough to keep you working on lab challenge designs. Develop a moderate weekly schedule for steady progress. Your goal could initially be to design or build a lab solution during each session, although later labs may require considerably more time and effort. Do not find out the hard way that necessity is not always the mother of invention, especially at 4:30 pm on the last day you can get a certain lab checked off.

However well you organize your time, you may still be frustrated by problems. The problems will be related to either incomplete circuit design, sloppy construction, or unsystematic troubleshooting.

Design problems can be approached in one of two ways. First, you can wait for inspiration to strike. This may involve re-reading a lab challenge, and then taking a walk in the rain. (A shower may be substituted on those occasions that the sun shines for several successive days in Providence.) At the end of your walk, you may have subconsciously thought about the problem enough to have a good first approximation for a solution. A second approach may be to sit down somewhere and think very hard about the nature of the problem and how it might be solved. You can read the recommended texts for the course. You could talk to friends. You might even pay attention during lecture! Another good approach might be to study the actions of various chips in your kit, and see if this provides a clue to how a problem may be solved.

Try to break the problem into several smaller ones. For example, if part of your assignment calls for a display, design and build the display portion of your circuit (being careful not to design yourself into a corner!). Seeing a portion of your assignment work may provide the incentive to explore other aspects of the challenge. An unrecommended approach to design is the semi-random connection of various inputs and outputs in your circuit "just to see what happens." This approach is usually taken with a circuit that almost works, but still has a few minor problems.

Suppose you have tried to think of *some* solution for a couple of hours, and still are getting nowhere. You are now at *frustration level zero*. Try (in the following order) these steps:

1) dreaming,
2) asking another student for help – soft collaboration,
3) consulting a TA for a hint,
4) seeking an appointment with the professor,
5) going to the Emerald City for an appointment with the Wizard of Oz.

Help from step 4 is guaranteed to provide you with a basis for hope.

5.2. Troubleshooting

Suppose you have a design that, on paper, looks as if it should work and yet does not give proper outputs after you have hooked it up and tested it. You have reached *stage one frustration*. First, calmly check your wiring. Are the power supply connections made properly? Are all the inputs to each gate accounted for? Do all of the outputs go somewhere? Next, check the power supply. Are you getting the proper voltage? Is the polarity correct? If the power supply has been reversed, all the chips in the circuit may be damaged. Is there an adequate number of bypass capacitors? Does one chip seem like the culprit? Take it out and test it by itself; you may have a defective chip which you should exchange for a good one. Are you having problems with clock signals? If there is a switch in the system, you may need to bypass or debounce its signal as described in the Construction Guidelines.

Use the logic probe or logic analyzer to test activity at various pins. (The analyzer is really not that hard to use and is very powerful in displaying the action of a system. Use it early and of-

ten.) There are clips in the lab that go over DIP integrated circuits and make easy connection of the logic analyzer to all the pins. The CPLD board has a logic analyzer plug that connects 16 pins to the analyzer in one push. Ask the TA for a clip or a plug for the CPLD and lay out your wiring to accommodate them. If you have timing signals, you may want to sketch what you expect the timing waveforms to be and to make sure that the proper events happen in the correct sequence by consulting the propagation delays listed in the data sheets. (Remember that simulation can be part of your paper design and the comparison of what is predicted and what is measured is a wonderful diagnostic method.) You may want to single step through the states of your circuit. Examining timing waveforms with an oscilloscope, you may discover noise on a clock signal or discover a signal that does not meet the specifications for a HIGH or LOW. Particularly in labs that use sequential circuits, you should use the analyzer to see if one event is happening before or after it ought to according to your design. You may have to introduce or remove delays from your circuits. You designed the circuit and ought to know the sequence of events it should display. Compare what happens with what you designed. In fact developing the ability to compare design to reality is the chief reason to require labs rather than doing everything in simulation.

You may want to double-check interfaces between one kind of chip and another. For example, if you have an output from a gate driving an LED, the voltage across the LED in the HIGH state may be clamped lower than the HIGH logic levels required for other connections to that output. Or, you may have a CMOS chip fanning out to too many TTL chips. Or, you may have a TTL chip unable to reach threshold voltage on a CMOS input. Is your problem intermittent? You may want to make sure that the wiring has been done in a neat manner and you may wish to wiggle the wires in their breadboard socket holes to make sure there are no inadvertent open circuits. Bare wires may cause unwanted short circuits.

Let us suppose, however, that you have checked all these possibilities and your circuit still does not work. This is *stage two frustration.* What should you do? Perhaps the first thing to do is ***not*** give in to the usual behavioral impulses attendant on frustration. These impulses will urge you to go eat or watch TV or talk with your friends or read a magazine or go to a movie or think about unnatural acts or just go to sleep - or *anything* so that you won't have to think about your circuit that doesn't work. First, try a couple of hours of redoubled effort and check your circuit again. Make sure your design is properly implemented and all of the problems you can imagine are taken care of. A growing impression you will develop in stage two frustration is that you are repeating over and over the same checks on your circuit, and they seem to have no beneficial effect.

Suppose learning by repetition and living with stage two frustration still leave you with a circuit which does not meet requirements. The next thing to do is sleep on it. We are not talking about a Rip Van Winkle solution here - just overnight, or at most forget it for the weekend. You will be hoping that a subconscious integration of the paradoxes and dilemmas of your case will lead to an intuitive insight - dream therapy, if you like!

Now suppose it is the next day, and the consolation of sleep did not help. You stare blankly at your circuit. You are entering *stage three frustration.* What next? Seek out the help of another student. You may know someone who has already done the lab or know a student from last year's 163 course, or you may have a sympathetic friend to whom you can explain your situation out loud and who may be able to spot an oversight. Of course, you will want to restrict your give-and-take

with other students to "soft collaboration." Remember that any advice you get from other students may be tested by the Fault Tolerance Question when you finally demonstrate your circuit!

Suppose the worst - that you find no peer help for your problem and you feel yourself sinking into stage *four frustration!* You will now want to see a Teaching Assistant. Go into the 163 lab and explain your problem to a TA. She will start a discussion with you. She will probably want to see your circuit design drawn on paper, so **be prepared with legible documentation**. (Sloppy documentation is very frustrating for both TAs and me. Do not try to transfer your frustration to one of us!) She may also set up some test, the results of which you both will study. She may want to swap a couple of replacement chips in and out. If she finds you have made a trivial error, she will point it out to you and you will be home free. If you are way off base though, the TA's are instructed not to give you a complete answer to your problem. The TA will suggest to you ideas which should help you do the design yourself. During this time you must keep in mind that the lab challenges are not simply homework problems that you are doing as practice for exams; the challenges are the primary basis of your grade itself, and as such, we and the TA's constrain ourselves not to spoon-feed solutions to you.

Suppose you take the ideas of the TA and attempt to implement them. You may modify your circuit; you may start over and build another one. But it is possible you will continue to fail. You are convinced the TA is a bozo. Things are worse now. You will have slipped to *stage five frustration.* Why might this be? The TA may not have understood your design, or may not have been willing to give you enough information for you to navigate out of your particular maze, or you may have simply been too depressed or anxious to appreciate the advice, or the TA *may really be a bozo!* In the depths of stage five frustration you will now want to seek out a Professor for trouble-shooting advice. I am always available after class. Also I am glad to answer questions if you find me in my office (Room 449) or labs (Rooms 195, 325 or 703). For more extended advice, make an appointment. When you do come, however, make sure that you come prepared with legible documentation of your design and have your circuit wired neatly. Be willing to describe your problem and the attempts you have made to solve it. I will try to understand your circuit enough to make a judgment about it. Be willing to endure a Socratic dialogue. I will tell you whether I think your design can work, and if I think it cannot, I will suggest some sort of redesign. Ask as many questions as you like. Do not leave until you feel at least a little optimistic that further work will be profitable. Remember that I cannot tell whether you understand something unless you tell me honestly if you do or don't. Don't try to hide your confusion when I say something mysterious. Make your feelings known if you think your progress has been halted entirely![11]

A final word on frustration: if the results of tests on your circuit indicate it, don't be afraid to acknowledge that you may have designed yourself into a dead end and that the best strategy may be to pull out all your wires and chips and start with a fresh idea. Perhaps the most bitter students we see are those who have become enamored of a particular design which it turns out will never work for a certain challenge. Such students drift from anger to cynicism to apathy. They become blind to the faults of the circuit at hand. They view it as possessed by supernatural forces beyond the understanding of student logic. They kill time by watching movies like "The Exorcist" when all they really need is a good EXOR gate. In professional life such tendencies are a devastating handicap.

[11] As much as possible, you will want to confront the source of your frustration. Do not try to suppress your emotional reaction to frustration. Remember, Valium is not included in the Bag-o-chips!

This advice about frustration may seem silly as you proceed with the early labs. We hope the advice will still seem silly as you continue on, but experience indicates that this may not be the case. You should be aware that the later labs, particularly 7, 8, A, and B, may require much more attention to detail and knowledge of chips.

6. Homework, Lectures, and Textbooks

6.1. Homework

EN-163 is largely a laboratory course, and hence has very little written homework. (I will give two optional problem sets but these are recycled from year to year. I will neither collect nor grade the answers.) It does have exams. The purpose of the mid-semester is to give you practice in answering abstract design questions; think of it as a structured means of studying for the Final exam. The mid-semester exam will actually come fairly late, probably in early November. I will grade and return it within a week or two. The Final Examination will also be graded, but will not be returned immediately. (You will have a chance to look at it as soon as I grade it.) You can pick it up after the beginning of second semester. If you want additional practice doing problems, try the ones at the end of each chapter of the textbook.

The major "homework" component of the course is the design and testing of the laboratory challenges. These labs are for your benefit, and should be done by you alone. Beat-the-clock cookbook labs are gone. No more lab partners with the IQ of a tree on one hand or the experience of Thomas Edison on the other. No more lab reports graded from 1 to 10 on neatness and precognition.

Unfortunately, some participants feel that the entire goal of the course is to complete the seven or nine or fourteen labs required to get a specific grade. The exams are intended to discourage this, but perhaps this is an unavoidable by-product of our general approach. I hope you will spend the time to expand your understanding of design in general, and to obtain an appreciation of the subtleties of digital design. One way to do this is to attend lecture and to read the readings. Another is to allow yourself to worry about larger questions, not just "can something other than a lecture help me design lab 3 in the shortest amount time."

6.2. Lectures

The sequence of lecture topics with recommended reading and relevant labs are listed in the course syllabus, given below. This syllabus is approximate, and we will deviate from it. (I want to move the VHDL material earlier, but even after several years I have not decided exactly how to do it.) The presentations will be elementary enough for all students to get the general drift of the lectures, yet not so simplistic that they will be totally boring to the somewhat-initiated.

Some of you may decide that the lectures are too simple or too sophisticated and not attend. If, however, you seek troubleshooting advice from a TA for a problem that had been explained in a lecture you did not attend, the *TA may give your problem a low priority* and urge you to look over

another student's lecture notes. Furthermore, **ALL** the material covered in the lectures is potential subject matter for questions on the examinations. For example, I believe that you should know at least a little about transistor circuits because you cannot do chip-level logic design without it and all logic design today tends ultimately to be done on chip.

6.3. Textbooks

The **suggested** textbook for EN-163 this year is *Digital Design: Principles and Practices, 3rd Edition*, by John F. Wakerly (Prentice Hall, 1999). An excellent reference for the last part of the course is *Analysis and Design of Digital Systems with VHDL*, by Allen Dewey (PWS Publishing, Boston, 1997), which I will put on reserve. I will not be following either book closely but hope that they will serve as useful and encyclopedic references. I have not found a fully satisfactory textbook. Wakerly is comprehensive and well written, as good a text as any I know of. Still he does not cover A/D conversion at all and covers several things lightly, *e.g.*, VHDL, memory, and FPGAs. Dewey has complementary strengths with particular emphasis on VHDL. There are several references to earlier texts in this manual. The reference (J & K) is to Johnson and Karim, *Digital Design, a Pragmatic Approach*, PWS, 1987. Another much older book, *Digital Integrated Electronics*, by Taub and Schilling, is particularly useful on analog-to-digital conversion and gate properties. It is called (T&S) in the lab manual and will be available on reserve at the Sciences library.

It should be noted, however, that digital electronics is a rapidly changing technology, particularly with respect to how many gates of what speed can be packaged on one chip and to a lesser extent with respect to what strategies are useful in their deployment. Any textbook purchased today is probably already out-of-date to some degree. One way to try to keep up with developments in digital electronics is via subscription to a magazine like *EDN* (formerly Electronic Design News).

6.4. Syllabus

This lecture schedule is approximate. I revise my lectures, pruning, shifting, and adding material. Please regard this as a rough outline only. All readings refer to: **Wakerly**, unless otherwise noted.

Fall 2004-05 Lecture Schedule			
Week	*Topic(s)*	*Lab Hints*	*Reading*
8 Sept.	**Logic Gates**: Boolean algebra; truth tables; DeMorgan's theorems; Sum of products; Karnaugh maps; Read-only-memory (ROM); Table look-up & MUX	0 and 1	Ch. 2.1, 2.2, 4.1 - 4.3
15 Sept.	**Logic Gates, Cont'd**: Design of circuits from logic expressions. Glitches and hazards; Noise margin and propagation time; Comparison of logic families: TLL, ECL, CMOS. Use of tri-state and open collector outputs.	2	Ch. 4.5, 5.2 – 5.8
22 Sept.	**Gate Realizations**: Signal levels; MOSFET transistors; the NMOS inverter; CMOS static gates; transmission gates; floating gate transistors and programmable logic.	3	Ch. 3
29 Sept.	**Flip Flops**: Types: R-S, JK, D; edge triggered, Master-slave, level sensitive. Gate realizations.	3	Ch. 7.1 – 7.2, 8.2
6 Oct.	**Sequential Circuits**: Counters; Synthesis by excitation tables and by ROM; finite state machines. State diagrams.	4	Ch. 7, 8
11 Oct.	**Columbus Day Holiday** No class		See footnote.[12]
13 Oct.	**Finite State Machines Continued and Use of CPLDs**: also one-shots;	5, 7, A, and D	Ch. 7, 8 & 74LS123 Data Sheet

[12] Recommended readings are: **De Kay**, **James**, *Meet Christopher Columbus* (Illustrated), Random House (1968) and **Lelend**, **Charles G**., *FUSANG, or the discovery of America by Chinese Buddhist Priests in the 15th Century*, Gordon Press (1977).

2004-05 Lecture Schedule (Cont'd)			
Week	*Topic(s)*	*Lab Hints*	*Reading*
20 Oct.	**Digital to Analog Conversion:** Analog switches, current steering, comparators, op-amps, resistive ladders	7	(T& S has coverage)
27 Oct.	**A/D Conversion:** Flash Conversion, successive approximation, and integrating converters. Also sample and hold	8	
3 Nov.	**Computer design tools:** schematic capture and timing simulation; **Buses and data transfer:** RTL machines.	9 & Schematic	Lab manual sect. 10 and handouts.
10 Nov.	**Memory:** RAM static and dynamic; ROM and memory as logic; Xilinx FPGAs and their software	A and C thru F	Ch. 10.1–10.4, hand-outs
17 Nov.	**Arithmetic:** Parallel and serial adders, number representation, and multiplication. Carry considerations, ALU's.	B	Ch. 2.6, 2.8, 5.10, 5.11
24 Nov.	**Makeup:** I will probably be behind at this point.	C,E	
1 Dec.	**VHDL for synthesis:** An introduction to a subset of the VHDL language suitable for synthesizing logic.	F	Ch. 4.7

7. Scorecard

Name_____ Board #1_____ Board #2 _____ Board #3_____

Lab	*Title*	*Date Completed*	*Checker's Initials*
0	One Bit Full Adder with NAND & EXOR		
1	Error Detection and Correction in 4 Bit Data		
2	Logic Family Properties and Logic Voltage Levels		
3	Two Digit Common Cathode Multiplexed Display		
4	16 Button Matrix Keyboard Encoding		
5	Counter with External Control (Please circle which implementation > TTL CPLD)		
6	Propagation Delay Measurement Using a Scope		
7	Dual Slope A/D Converter		
8	Successive Approximation A/D Converter system		
9	Software simulation of either Lab 5, 7, 8, A, B, or D (Please indicate which one >)		
A	RAM Memory Subsystem with Bus		
B	4-Bit x 4-Bit Hexadecimal Multiplier (Please circle which implementation > TTL FPGA)		
C	JTAG Boundary Scan Register Implemented in Xilinx Array – Note: this lab is being revised		
D	DRAM Controller Implemented in Xilinx Array		
E	Music Box (Xilinx FPGA based logic)		
F	An Electric Sign Using VHDL Synthesis Targeting a Xilinx FPGA		
	DxDesigner Schematic for either Lab 1, 7, 8, A, or B. (If checked by TA, please indicate which lab →)		
	Logic analyzer applied to Lab 7, 8, A, B, D or F. (Please indicate which lab →)		

8. Bag-o-Chips Inventory

Item	Qty	Part #	Cost
NAND gates, two input TTL, four gates/chip	2	74LS00	$0.58
Hex inverters; CMOS, 6 inverters per package	1	74ACT04	0.56
Hex inverters; TTL, open collector, high voltage	1	7406	0.43
Hex inverters; Schmitt trigger inputs	1	74LS14	0.57
D Flip-Flops; two FFs/chip	3	74LS74	1.20
JK Flip-Flops with PRESET & CLEAR; two/chip	1	7476	0.58
EXOR gates; 4/chip	3	74LS86	1.38
One-shot (monostable multivibrator); 2/chip	1	74LS123	0.54
1 of 4 Decoder/Demultiplexers; two/chip	1	74LS139	0.66
8 -> 1 Multiplexer	1	74151	0.64
2-> 1 Multiplexers; four/chip	1	74157	0.64
Shift register, serial in-parallel out, 8-bits	1	74LS164	0.88
Counter, binary up/down, TRI-STATE, Sync. & Async. CLR	1	74LS169	0.60
D-FF latches; six/chip	1	74LS174	0.70
Shift Register 4-bit, parallel access	1	74195	0.68
Inverters; TRI-STATE	1	74LS240	0.80
Analog switch/multiplexor; four/chip	1	CD4066B	0.51
Comparator, open collector output	1	LM311	0.75
Operational Amplifiers; two/chip, BIFET	1	LF353	0.60
Timer, astable or monostable	1	555	0.65
Light-emitting diodes	1	NSL5056	0.80
Matrix keyboard	1	11KS121	4.95
Miniature toggle switch SPDT	1	245C	1.60
Eight position DIP switch	1		2.10
20K potentiometer	1		0.21
Resistors: 100(2), 510(1), 1K(6), 10K(6), 100K(3), 1M(2), 10M(1), 39K(1), 5.1K(2); (14 W.)			0.92
Resistors: 20K (precision 1% ¼ W, for A/D labs)	6		0.55
Capacitors: 0.001µF(2), 0.01(2), 0.1(6), 1(+/- 20%), 50pf(2)			0.92
Wire cutter/stripper (Miller)	1	Model 100	2.80
Solderless Breadboards, each individually numbered	2	EXP 300	7.90
Cable for protoboard to CPLD board connection	1	In-house assembly	5.61
Total Cost			$42.20

9. The Lab Challenges

9.1. Displays and Light Emitting Diodes

The Care and Feeding of Displays and LEDs

WARNING: It is fairly easy to burn out part of a hexadecimal display or to destroy an LED. We look with disfavor on such actions as they usually mean you have been careless. This section describes the electrical characteristics of these devices and how to avoid problems with them. There is also information on wiring the two digit displays so that both digits are usable at the same time. This process, called multiplexing, is needed for the first time in lab 3.

How can you burn out a segment of a display? *By allowing too much current to pass through the segment.* How much is too much? The data sheet for the display gives several pieces of information: under absolute maximum ratings is IFP = 60mA., which means "Peak forward current per segment not to exceed .06 amperes" (60 "milliamperes"). Also under absolute maximum ratings is I_F = 20 mA., which means that for long term operation with more than one segment lit, the average current through a segment must not exceed .02 amperes. In testing a display with one segment lit for short periods of time, it is probably safe to use a current between these two numbers, but nearer to the latter. A maximum of 30 mA. is safe choice. There is also a typical rating of V_F = 2.1 volts @ 20 mA., meaning that about 2.1 volts must appear across a segment in order to pass 20 mA through it. Thus, the potential difference across R in Fig. Disp-i is V = 5 – 2.1 =2.9 volts. According to Ohm's law,

$$V = IR = 2.9v = (30mA.) \times R$$

thus

$$\mathrm{R} = \frac{2.9}{30 \cdot 10^{-3}} = 97\Omega$$

so a resistance less than 97 ohms in series with 5v and a forward biased (operating) segment may cause irreversible damage.

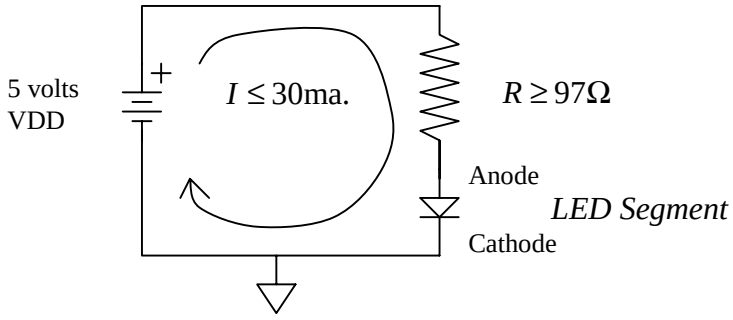


Figure Disp-i: LED biasing.

When you are probing pairs of pins to determine the pinout of the display you will be safe if you use a 100Ω (brown-black-brown) resistor in series with +5v and ground. Later, after you have

determined which pins connect to the segment anodes (and which connect to the common digit cathodes) you will be using the displays on the CPLD-II or BUXUSP-II boards, and these have current limiting resistors built-in.

If you place *too much* resistance in series with a segment (> 1K) the segment will be too dim for good visibility.

Each segment is a light-emitting diode (LED), with an output wavelength of 656nm., which corresponds to red light. The electrical characteristics of diodes are discussed in chapter 3 (sec. 3.9) of Wakerly, and in Taub and Schilling (T&S) chapter 1. Both books will be on reserve at the Sciences library for this course. The I-V plots of a silicon diode and of an LED are shown superposed in Fig. Disp-ii. The primary difference in their electrical characteristics is the difference in the voltage required to produce appreciable current flow through the diode, which is called the turn-on voltage V_{on}.

Note that a diode does not obey Ohm's law! For one direction of applied voltage, there will be no current flow or light output. In the other direction (called the "forward" direction) the current flow and light output will rise rapidly once the potential exceeds V_{on}. For a regular p-n junction silicon diode $V_{on} \approx 0.7$V; for our LEDs $V_{on} \approx 1.7$V. If a diode is placed across the output of a logic chip (i.e. between the output pin and ground) the chip output will be clamped to V_{on} when the chip output is driven to a "HIGH" state! A value of 1.7 volts is *less than the minimum high output* expected by other chip inputs. Loading chip outputs with LEDs can be a source of problems *unless* a "big enough" resistor is placed in series with the diode to increase the voltage across the combination. Depending on the driver chip characteristics, "big enough" may be more than 100 Ω.

What does it mean that your display is a "common cathode" device? Cathode and Anode are terms that refer to the two ends of a diode. Common cathode means that all the cathode ends of the seven segment diodes (and the decimal point) are connected together at *one* pin, which normally should have a pathway to ground (perhaps through a transistor or a gate, if both digits are to be displayed "simultaneously"). See Figure Disp-iii. .

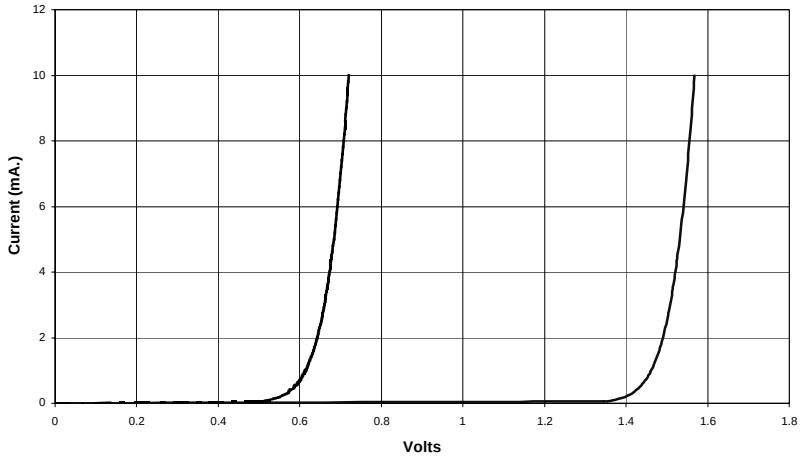


Figure Disp-ii: Diode I-V plots.
(The left curve is a silicon diode and the right one is a red LED.)

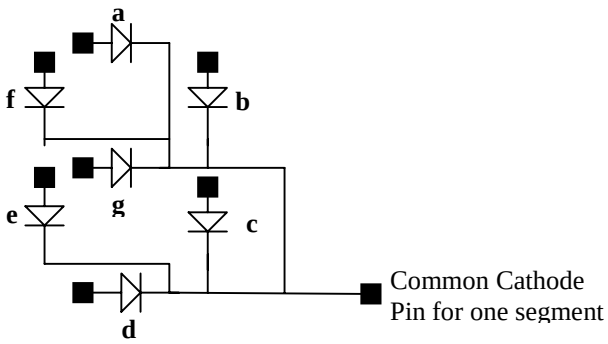


Figure Disp-iii: Common cathode display internal wiring. (Segments are labeled a – g. Black rectangles are pin connections.)

What is a "multiplexed" display? It is one in which two or more digits time-share a common set of wires and decoding logic. Multiplexing is used to save wiring and cost. It is done by connecting the anode of one segment of anode of the same segment of every other digit in the display. All the cathodes of each digit are then wired to a line unique to that digit. Thus, for a two digit display there is one connection to each pair of segment anodes (eight pair in all, counting the decimal point). See Figure Disp-iv.

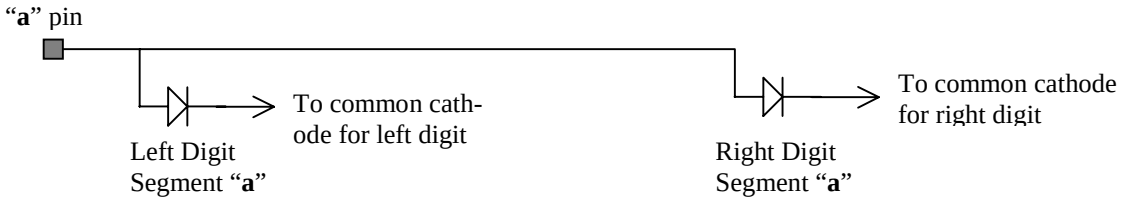


Figure Disp-iv: Display segment multiplexing.

The signals for the left and right digit segments must share these common points. The segment information must be *multiplexed* (i.e. it must share, in time, the common set of anode wires). Data for the right segment is applied to the anode lines when the right segment cathode is grounded and left data when the left cathode is grounded. By rapidly changing back and forth between these two conditions, say at a rate of several hundred times a second, one can make both digits appear to display the appropriate data continuously.

Your displays are common cathode (with the decimal points connected to the same common cathodes) but are not multiplexed. Each segment anode has a separate pin. When one needs to use a minimum number of pins to drive both digits, one simply connects the anodes together pair wise between digits to make a display that is multiplexed.

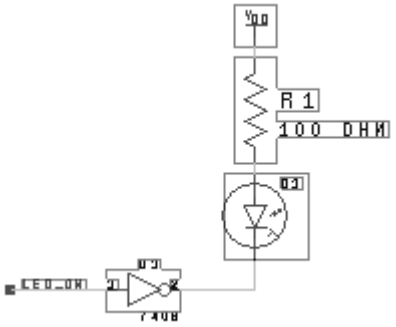


Figure Disp-v: Driving an LED from a logic gate.

The other LEDs in your kit, two NSL5056's, must also be protected from too much current - with a current limiting resistor ($\geq$100 ohms). Placing these diodes directly across +5v to ground will burn them out. A common requirement is to turn on an LED from a logic signal. It happens for reasons we will discuss in class that almost all logic gates can conduct more current from output to ground in the LOW state than from VDD to output in the HIGH state. For this reason and a couple of others, the circuit shown in Figure Disp-v is the preferred arrangement. Note that the LED lights for a 0 on the output or a HIGH on the input of the inverter. The inverter can be an open collector or open drain type such as the 7406.

9.2. The Numbered Labs

9.2.1. Lab Zero

One Bit Full Adder

Requirements: Design and build a one bit full adder using only **one** 74LS00 (quad NAND) and **one** 74LS86 (quad EXOR) for logic. The adder is to have *four* outputs, two of them are the usual **Sum** and **Carry-out** signals; the other two are outputs normally used by fast carry-look-ahead logic. The latter two depend only on A and B and are called **Propagate** and **Generate**. The three signals, "Carry-out", "Propagate", and "Generate" are related to each other as

$$C_{out} = C_{IN} \cdot P + G$$

The Generate signal is partially listed in the table below to get you started. This equation is not enough to define *P* unambiguously. However the further constraint that

$$P \cdot G = 0$$

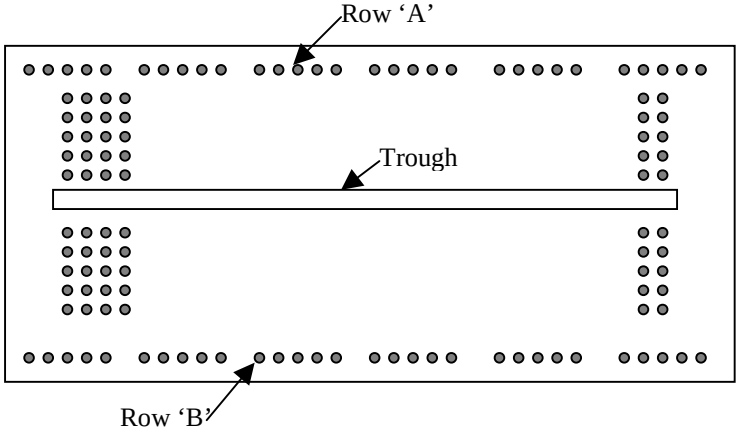
does define it sufficiently. Another way to describe the Propagate signal is that it is HIGH if and only if the values of *A* and *B* are such that both C_{IN} = '1' would imply C_{OUT} = '1' and C_{IN} = '0' would imply C_{OUT} = '0'. (Note that one implication of this statement is that *P* and *G* are never simultaneously true. When *G* is HIGH, then C_{OUT} is HIGH even if C_{IN} is LOW.) This definition of *P* is deliberately slightly different than some common definitions of this signal. Initially, the Sum and Carry-out signals must be displayed with two LEDs. Be prepared to connect the LEDs to the *P* and *G* outputs when asked to do so by the TA. Set up your DIP switch contacts 1 to 3 to be *A*, *B*, and C_{IN} respectively. (WARNING: be sure to do this in such a way as not to damage the switch and so as to get full noise margins. **Use pull-up resistors.** See comments about pushbuttons under "Debouncing" in the introduction for how the circuit would look.)

Inputs Outputs .						
Carry In	A	B	Sum	Carry Out	Generate	Propagate
0	0	0	0	0	0	
0	0	1	1	0	0	
0	1	0	1	0	0	
0	1	1	0	1		
1	0	0	1	0		
1	0	1	0	1		
1	1	0	0	1		
1	1	1	1	1		

Discussion: The full adder is discussed in chapter 5.10 of Wakerly and on pp. 357-360 of T&S. The truth table is given here. You should be able to fill in the *P* and *G* columns from the description given above.

Of course, with lab 0 (as with all demonstrations of working labs) you will be asked a Fault Tolerance Question: "What, and only what, will go wrong with your circuit \ if such-and-such a wire is removed?" To achieve credit for Lab 0 you must answer a Fault Tolerance Question correctly. In preparation for the question, you must have a legible schematic.

The holes in your breadboard are connected as shown in Fig. 0-i. Normally you connect +5v to row *A*, GND to row *B*, and straddle the chips across the trough to have each pin contact a different column C_i. Determine the LED anode and cathode pins by testing with a 100Ω resistor connected between the diode and VCC. Truth tables for NAND and EXOR are shown below:



All holes in row 'A' are connected. Likewise all holes in row 'B' are connected, but neither row connects to anything else. These rows are normally for VDD and Gnd.

Each set of 5 holes in a column are connected, but column on opposite sides of the trough are not connected.

Figure 0-i: Breadboard interconnections

Inputs	NAND	EXOR
0 0	**1**	**0**
0 1	**1**	**1**
1 0	**1**	**1**
1 1	**0**	**0**

Designing lab 0 is largely the clever application of DeMorgan's theorems, not necessarily to minimize the logic but to push it into a form that fits the two chips. For further help come to class and consult Wakerly chapter 4 or any text that discusses combinational logic.

The Propagate and Generate outputs are used in systems that calculate C_{OUT} by different means. These systems go under the names of carry-look-ahead and Manchester carry adders. It is unlikely that one would use both C_{OUT} and Propagate/Generate logic in the same adder. Nonetheless it can be done within the limits of your logic, and an actual system might use this circuit and prune it according to which type of block was needed at a certain point.

9.2.2. Lab One

Error Detection and Correction for 4 Bit Data

Requirements: Set up one of your DIP switches to simulate seven bits of parallel data within a communication or storage system requiring error correction. The four least significant bits shall be the data bits Q_A through Q_D. (Q_A is the least significant bit.) The next three bits are error correction bits assumed to have been derived from the Venn diagram of Figure 1-i with each circle having even parity. ECC_1 is the fifth significant bit, ECC_2 the sixth, etc. Assuming that no more than one bit can be in error, devise a circuit that continuously displays the correct (or corrected) hexadecimal representation of the data bits Q_A through Q_D on your LED seven segment display. Also light one LED if there is an error in any of the seven bits.

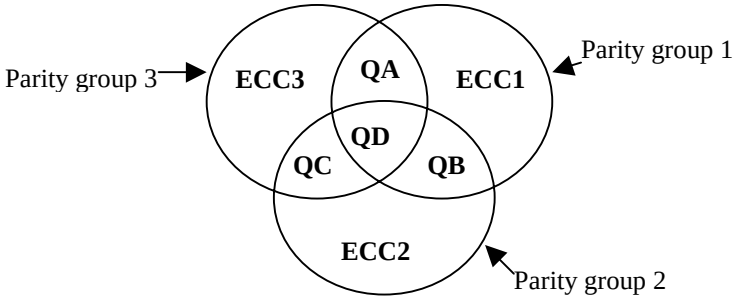


Figure 1-i: Venn diagram of error correction bit assignment.

Discussion: There are many situations in digital systems in which data bits cannot be counted upon to be completely accurate. One such situation commonly occurs in communication systems where noise on a channel can occasionally cause one of the bits in a data stream to be interpreted incorrectly as its complement by the receiver. Memory systems are also subject to soft errors (e.g. alpha particles from trace radioactive contaminants in the packaging materials hitting a memory chip can cause a single location to change bit polarity). All of these sources of error are to a greater or lesser extent unavoidable. Typically many of these sources of error are random in occurrence and relatively infrequent (say one error in every 10^5 bits or more). For this reason, single bit errors are the most common, and two nearby bits in error are very infrequent. However, with the high transmission rates and large storage capabilities of current machines, random bit errors, even at very low rates of occurrence, would render the systems useless if it were not possible to detect and correct most of them.

Errors in the data bits can be detected by introducing additional “error correction” bits which have a specific, known relationship to the data bits. The data bits are divided into groups, and error correction bits are assigned to each group to give the group a particular *parity*. A group is said to have “even” parity if it has an even number of '1s', and odd parity if the number of '1s' is odd. For four bit parallel data, there is a way to add three correction bits to form three parity groups from which single bit errors can be both detected and corrected. Figure 1-i shows the simple Venn diagram on which the technique is based.

Each circle represents a parity group. The ECC bit in each circle is the added correction bit needed to make the parity of all the bits in that circle even. For example, parity group 1 contains ECC_1 and the data bits Q_A, Q_B, and Q_D; if $Q_DQ_BQ_A$ = 001, then ECC_1 = 1. The value of ECC_1 is chosen to make the parity of this entire group even. Similarly the values of ECC_2 and ECC_3 are chosen to make groups 2 and 3 have even parity.

If any single bit is changed, the parity of one or more of the groups will be changed. You can convince yourself that an error in any one of the seven bits will give rise to a unique parity pattern, so that the specific bit which is in error can be detected. For example, an error in Q_B causes odd parity in groups 1 and 2 but not 3. There are three parity groups yielding $2^3 = 8$ error states: one for no error and one for each of the seven possible single bit errors. Since one can discover which bit is in error, one can correct it, so this scheme can both detect and correct all *first order* errors (i.e. no more than one bit altered). For more information, you might consult R.J. McEliece, *Scientific American*, January 1985, pp. 88.

Incidentally, if two bits of the seven are altered, this scheme will misdiagnose the error, and the correction scheme will typically make things worse. There is, however, a way to add a fourth correction bit which (through the use of additional parity checks) will detect the existence of a second order error (two bits altered). Unfortunately, this scheme does not provide enough information to correct the second order error, but it does allow one to detect the existence of an unrecoverable error. **You might want to think** about how this could be done.

9.2.3. Lab Two

Logic Family Properties and Logic Voltage Levels

Notes: Lab 2 is one of only two *measurement labs*; the other labs are synthesis (design) exercises. The *raisson d'etre* for labs 2 and 6 is that the properties you measure are the absolute minimum electrical characteristics which a digital-system designer must understand to make intelligent decisions. Because of the different type of experiment, *hand in a report* instead of demonstrating a circuit for Lab 2. Be prepared to answer questions about your measurement procedures, which we may ask you if the report is not completely satisfactory.

To prepare yourself for this lab, you may find it worthwhile to refer to T&S, chapters 6 and 8. The material is also covered in chapter 3 of Wakerly. Also, read the lab description carefully and be sure you understand the procedures and explanations *before* you come in to the lab. There are only a limited set of meters for this lab, so please do not waste time by attempting to do it unprepared! Unfortunately experience indicates that digital voltmeters may be stolen unless we try to keep track of them. Our new Agilent meters cost 270 dollars to replace and our older meters cost around forty dollars each. Replacement becomes a substantial cost even in limited quantities. Therefore, the TAs may check meters out to you, and you are responsible for them until they are checked back in.

Organize your report in the spirit of Computer Science documentation standards or Engineering core-course labs. There will be a place for handing in your report in room 196. We have not set that up yet, but we will tell you the procedure soon and ask that you not hand things to me. I lose them. The report must be **typeset** and contain three basic parts. First is a brief introduction stating **your** understanding of the importance of interface standards and of large noise margins incorporated into such standards. Second, tabulate your measured data with units and measurement conditions clearly labeled. Finally, answer the questions that follow the measurement requirements. The report will be graded and returned shortly after the deadline for doing this lab. The report will be judged satisfactory and you will receive automatic credit for it or else some specified part must be done again or must be explained better. If only minor corrections are wanted, the TAs will judge your response to the required corrections and will check off your scorecard in the usual way. Reports needing more substantial corrections will have to be turned in again. Write-ups that do not require corrections will be recorded automatically. Your returned report will indicate which process to follow.

Measurement Requirements:

1) **Logic Levels and Noise Margins - TTL Family:** For each of your four, standard-TTL inverter chips 74LS14, 7406, 74LS240, and 74ACT04, graph the voltage out as a function of the input voltage over the range $0 < V_{IN} < 5v$. (The 7406 will not act as an inverter without a pull-up resistor; for standardization, use 1 K ohm.) Take sufficient measurements to be able to plot the output transition from high to low with good accuracy. Pay particular atten-

tion to the gate threshold levels and to the values of input that produce .8 and 2.4 volt outputs. To make it easy for you to do this, we have several sawtooth waveform generators that plug into the power supplies in the lab. (Please do not use the Agilent function generators for this purpose – they are too easy to damage.) They produce a roughly 1 KHz sawtooth wave that goes from .1 volts to 4.5 volts. You apply this signal to the input of the gate **with a small (50 pf) capacitor** also connected directly from input to ground. (The reason for this capacitor is to prevent any interaction of the gate with the source, especially for Schmitt trigger devices.) Use a scope with two identical 10X probes to measure the input and output simultaneously. Set the scope in X-Y mode with the gate input on the X axis. The resultant plot is the input-output relation directly. Take enough data that you can reproduce this curve accurately in your report. The 74LS14 and 74LS240 are Schmitt trigger devices and show two thresholds with hysteresis. When you draw the figure for your report be sure to capture this effect. Mark the transition lines for the rising and falling inputs clearly. There may be some difficulty getting good data on the 74ACT04. This is because it is a very fast, high-gain device that may actually oscillate in your breadboard when the input is near threshold. If this happens to you, the line on the scope connecting high and low levels will be blurry and unstable. You can represent that in your report with a fat, blurry line. Do be sure to use a bypass capacitor directly between VCC and GND on all devices being measured! On the 74ACT04, be sure to ground any unused inputs too.

2) **Low Voltage Device Logic Levels:** We will have some 74LV04A inverters available on adapters in the lab. The data sheets for this device are in the back of the manual. The main feature of these parts is that they are designed specifically for use with power supply voltages from 2.0 to 3.3 volts as part of the trend to lower supply voltages, but they will withstand 5.0 volts. Repeat the logic level measurement that you did above for the 74ACT04 device except use a variable power supply. Measure the input-output voltage curve for VDD equal to 3.3 volts and again for 2.0, 2.5, 4.0 and 5.0 volts. (If you set up right for this, the measurement times are trivial.) Plot the gate threshold voltage from this measurement against supply voltage.

3) **Open Collector Outputs:** You have two chips with open collector outputs: 7406 and LM311. Normally these devices require “pull-up” resistors on their outputs. By measuring the 7406, we explore why this is necessary. With your 7406 powered by +5v, measure the output voltage of an inverter with no pull-up resistor on the output for the input grounded (logical '0') and again for the input connected to +5 volts (logical '1'). Repeat these two measurements with a “pull-up” resistor tied from the output pin to +5v.

What happens to the output voltage when the pull-up resistor is tied to +12v? (DO NOT POWER THE CHIP WITH +12v THROUGH PIN 14! KEEP PIN 14 AT +5v!) .

Can one of the 7406 inverter gates drive another TTL inverter gate properly without a pull-up resistor at the interconnection between them? Try it by measuring the logical state of the output of one inverter of the 7406 with a pull-up resistor when its input is driven by another section without one. Apply logic LOW and HIGH to the input of the first inverter, measuring the output of the second gate. Also measure the input voltage of the second gate (output of the first inverter) for the same conditions. (This data is needed to answer one of the later questions.)

4) **TRI-state outputs:** You have two chips with TRI-state outputs - 74LS240 and 74LS169. With your 74LS240 gates *disabled*, measure the voltages on an output pin when the corresponding input is HIGH and LOW. (Be sure to power the chip with +5v and GND!) Measure how the two output voltages change when you *enable* the gate.

5) **Unconnected Inputs:** Frequently, systems have unused gate inputs which need to be either '0' or '1' at all times. Making no connection to a pin may seem an economical way to get such a condition, but it can cause problems. The purpose of these measurements is to determine the logic level of an unconnected input, if indeed it is determined at all, for a 74LS14 inverter and for a 74ACT04 (CMOS) inverter. The actual input current of an unconnected pin is identically zero. To get an idea of how well determined the logic state is, you also measure the input current at full LOW and HIGH levels to see how the current changes. One can measure the logic level associated with an open connection by measuring the *output* of a gate with no input connected and inferring the input state. It is sufficient for the 74LS14 to do this once. For the 74LS14, also measure the voltage at its input with nothing but the voltmeter connected to that input as a rough measure of the unconnected voltage. Then measure the input current with the input LOW and HIGH. To do this, connect a 510 ohm resistor from the input to GND and then from input to VCC. Measure the voltage across the resistor with the multimeter. Be sure to observe the sign of the voltage (the direction of the current) as well as its value.

   The 74ACT04 acts differently. First determine if an uncommitted input has an apparent definite value. Is it HIGH or LOW? Record the measurement you do to measure it and describe your procedure in your writeup. Then measure the input current for LOW and HIGH inputs by connecting a 10 Megohm resistor from input to GND and then from input to VCC. Measure the voltage across the resistor with your multimeter. (Gate currents of MOSFETS are invariably below 10^{-14} amps. If you measure any significant current, it is likely to be from protection circuits rather than the gate input itself.) Note that these input currents are very temperature sensitive, are specified in the data sheet, and may actually go either direction. Different manufacturers make nominally the same device with different input currents. Given the current can go either direction, is the input " well determined?" Record the input HIGH and LOW currents. If you wish to (i.e. it is not required), it is instructive to test how quickly a gate goes to a known condition. To do this, connect a short wire to one input of the 74ACT04. Then, measuring the output continuously, connect the wire momentarily to ground. (It is best to use an oscilloscope for this measurement as the output may change a second or so after you disconnect the input. This is hard to see unambiguously with a multimeter.) From the output state, what must be the input state immediately after you open the ground connection? Then momentarily connect to +5 volts and note the output immediately after you disconnect the input. What must be the input state then? (Do not try to measure the input voltage with a voltmeter during this measurement. Do not touch the bare part of the wire.)

6) **Power Consumption:** Measure the power supply current and calculate the power consumption for your 74LSL14, 74ACT04 and 74LV04A inverter chips. Do two cases - one with all inputs high, the other with all inputs low. For the 74LV04A, measure with VDD = 3.3V and again with 5V. (See the discussion below for how to measure CMOS and TTL power.

The measurement technique differs slightly between the two because of the range of power consumption.)

7) **Fanout Limitations:** Figure 2-i shows a simplified model of the output of an inverter based on an electrically controlled switch and two resistors. The resistors mimic the change in output voltage with output current. This model is not very good for TTL gates because their transistors do not obey Ohm's law. However, it is a reasonable approximation for CMOS gates. R_L and R_H are the on-resistances of the NMOS and PMOS output transistors respectively. The model is good enough that we will use it for you to find an upper limit on fanout from your 74ACT04 to TTL gates like the 74LS14.

   Measure R_L and R_H for one section of a 74ACT04. Probably the most reliable method is to connect a resistor ($\approx 510\ \Omega$) from output to ground, put the output in the logic HIGH state, and measure the potential difference between the V_{CC} and the output pins. From this you calculate R_H using Ohm's law. A similar arrangement but with the output LOW and the resistor between the output and V_{CC} will give R_L. The internal resistances shown in Figure 2-i will limit the amount of current which your chip can handle before noise margins are compromised.

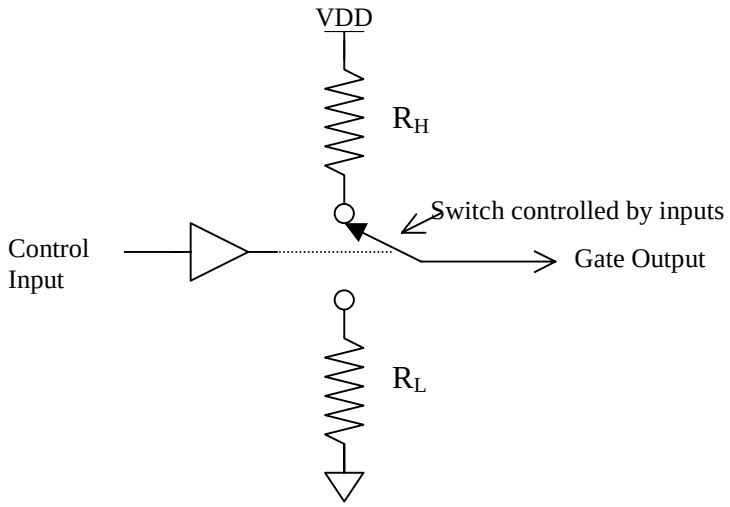


Figure 2-i: Simplified switch model for the output of a gate.

Questions and Interpretation:

1.) From the data from part (1) of your measurements, what is the gate-threshold voltage, V_{THG}, for each standard 5-volt gate type? (This is the input voltage for which the output voltage has the same value. For Schmitt trigger devices, 74LS14 and 74LS240, there are two such thresholds.)
2.) Assume that the required input voltages for logic HIGH and LOW are V_{IH} = 2.0 and V_{IL} = 0.8 volts respectively. (These are the data sheet standards.) Based on the actual V_{OL} and V_{OH}, what are the high and low level noise margins, ΔV_H and ΔV_L, for each chip? (Here we are asking about the actual rather than the very conservative guaranteed margins. In this sense, the noise margin is the difference between what is required at an input to which the output of the gate might be con-

nected and the actual value of the gate output when its own input is at V_{IH} or V_{IL}. The latter is part of your measurements in part 1.) Finally, for the 74LV04 operating at 3.3 volts, what are the noise margins if $V_{IH} = 0.3\ V_{DD}$ and $V_{IL} = 0.3V_{DD}$?

3.) Why are the output voltages of the 7406 so changed by the pull-up resistor? Please be explicit and write a short paragraph in clear English. We are looking for an indication that you understand the roles of the resistor, power supply, and transistor switch. Do not be afraid of Ohm's law as part of the explanation.

4.) This question looks at the properties of the open-collector 7406: What does the data sheet for the 7406 claim is the maximum allowable output voltage? What does the data sheet claim is the maximum allowable *current* that can flow through the output transistor before it may be DESTROYED? How much current flows through a 1K pull-up resistor when it is tied to +12v and V_{OUT} = LOW? What is the lowest value of pull-up resistor that can be used with the 7406 and a +12v pull-up supply?

5.) What logic function is realized if all six 7406 outputs are connected to the *same* 1K pull-up resistor? Use DeMorgan's theorems to express this result in two forms.

6.) If an open collector device with no pull-up resistor drives a TTL input, does the circuit work at all? What is the noise margin of such an interconnection when the interconnection point itself (the output of open-collector inverter and input of the second device) is in the HIGH state? (To answer this sensibly, you need the measurements of this situation that you did for part (2) above. The answer is the difference between the measured input without a pull-up resistor and the measured input that will make the output 0.8 volts.)

7.) Is the state of an unconnected CMOS input well determined? Why? (Remember that a MOSFET gate draws almost no steady current. The only current at a CMOS chip input is the small, random, temperature-sensitive current of its static protection circuit. The direction and magnitude of this current depends on manufacturer, input voltage, temperature, and conditions during manufacture and is not guaranteed.)

8.) Draw a circuit diagram showing how to configure your 74LS240 as a 4-bit wide, 2-to-1, multiplexer.

9.) For the worst-case input and output currents listed in the 74LS14 data sheet, calculate the maximum fanout for a 74LS14 output projecting to other 74LS14 inputs.

10.) How many regular LS-TTL inputs can one of your 74ACT04 (CMOS) outputs drive high *and* low? Assume that the output has to be limited to 0.2 volts for low and 3.5 volts for high. Use the worst-case input currents from the data sheet for the 74LS14 and your measured values of R_H and R_L for the 74ACT04. (You may well find the answer is quite large. Other factors including capacitive loading and wiring inductance generally make it unwise to try that large a fanout.) As a *Challenge to the Bored*, use the fact that the circuit must work at 150 deg. C and that the resistances of the CMOS gate increase as $T^{1.5}$, to calculate the fanout limit over temperature. (T is the absolute temperature and at 25 deg. C. T =300 deg. absolute.)

Discussion: You should be able to make most of the required measurements with one of the digital multimeters available in the lab. The I/O voltage relationships require one of the 100 MHz scopes. These scopes are set up so that a trace can be recorded and stored in an EXCEL file on a PC if you so wish.

T&S chapters 4 through 8 contain an exhaustive description of logic family characteristics. Another very good discussion is in Hodges and Jackson, *Analysis and Design of Digital Integrated Circuits*, on reserve in the library. The current edition of your text, Wakerly, has a nice discussion of the properties of various gate circuits in chapter 3 including extensive coverage of TTL gates in sections 3.10 and 3.11. We will spend much of the class time available for discussing gate characteristics in talking about CMOS circuits.

Noise Margins: All logic systems have significant amounts of noise - unwanted and unintended voltages added to the real signals of the system. The noise is usually induced by electromagnetic coupling between the different signal wires. While it can be minimized, it cannot be eliminated. Much of the appeal of digital systems comes from their relative immunity to such noise. Figure 2-ii shows a circuit model for what is going on as a noise voltage adds to the output of a gate, U1, before the signal reaches the input of gate U2. With well-designed gates, the voltage at the input of U2 which is necessary for U2 to recognize a logic LOW, V_{IL}, is more than the output of U1 in its LOW state, V_{OL}. For U2 to function properly we must have

$$V_{OL} + |V_{NOISE}| < V_{IL}$$

The maximum noise which will still result in this inequality being met is the *low level noise margin*, $\Delta V_L = V_{IL} - V_{OL}$. There is a similar consideration when the node is in the HIGH state for a *high level noise margin,*

The standard specification for TTL circuits assumes that V_O will always be less than V_{OL} = 0.4 volts in the low state, that $V_{IL} \geq 0.8$ volts, and that ΔV_L is then 0.4 volts. Similarly the output high voltage is 2.4 volts minimum, the input high level is 2.0 volts and ΔV_H is 0.4 volts too. Now all these numbers are very conservative since they have to reflect the worst possible variations of manufacturing, temperature changes, power supply variations, etc. In this experiment you are looking at how much extra margin there may actually be in more typical cases. (This is not to suggest that it is okay to build things with more than rated noise levels!)

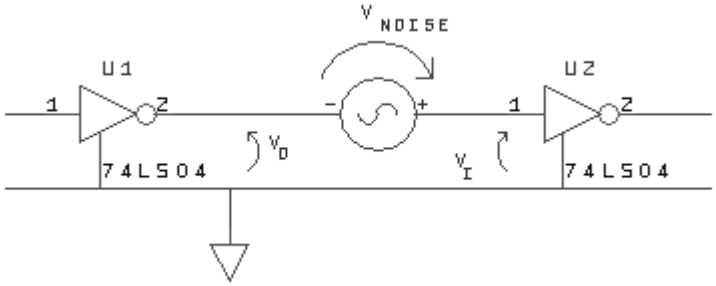


Figure 2-ii: Model of how noise adds to a signal between gates.

Threshold Voltage: Except for gates with Schmitt trigger inputs like the 74LS14, there is an input voltage for which the output voltage is equal to the input. This is the "gate threshold voltage" V_{TH} or V_{THG}. (In practice this may be hard to measure if the circuit oscillates for inputs near threshold as your 74ACT04 may.) It is useful as a rough measure of the value of input at which the output is most dependent on the input. It or an average data sheet value for it is often used as a reference level for timing measurements on the assumption that this is the value of input at which most electrical activity inside a gate actually happens. Nothing much happens inside a gate until its input approaches this level, so it makes sense not to count the time it takes for input to rise to this level as part of the propagation time through that gate. Similarly, its output must change to a value near gate-threshold before anything will happen in a subsequent gate. The propagation time through the gate should not be counted as over until the next gate begins to respond to its input. Therefore the time between input and output threshold crossings is a good simple measure of propagation time.

The 74LS14 and 74LS240 have what are called "Schmitt-trigger input circuits". They do not have an equilibrium condition with equal input and output voltages. Instead internal circuitry sets two different levels of input at which the output will go abruptly from high-to-low or vice versa. The two levels are distinguished by the initial state of the gate. Thus there are effectively two different thresholds depending on the transition direction. The advantages of such an arrangement are a much higher noise margin and an ability to respond sensibly to slowly changing input signals. These devices are often used as interfaces between slow and fast systems.

Fanout: Fanout is the number of connections to other devices from a given gate output. There is a maximum number of such connections that can be made if the output logic levels and the transition speed are not to be affected by the load. With TTL gates, the primary limitation is the DC current drawn by the gate inputs. Suppose you are attempting to keep an output at a high level. The more inputs that one output projects to (fanout) the more current is required to leave the output pin to supply the various inputs. The addition of too many output loads (i.e., inputs to which the output is connected) will cause so much current to pass from the power supply through the gate that the voltage at the output will drop *below the high level required by the various inputs* or at least below an acceptable noise margin. A similar argument with I_{OL} and I_{IL} currents can be used to calculate a fanout for the *low level* output voltage. The lesser of the two fanout numbers is the "worst case" fanout. This value is a direct and rigid limitation on the number of inputs a designer can connect to one output.

Since more current is required to change a logic level than to maintain it, fanout is often limited to less than DC measurements might suggest. (The reason for the difference between the static and dynamic currents is the input and wiring capacitances which have to be charged or discharged. The charge in those capacitances contributes to the load current of the gate only as it is changed, "i.e." during transitions.) In fact, chips which are built from MOSFET transistors (such as your 74ACT04 and MCM6268P55 chips) have negligible static input currents, and so the only limitation on fanout comes from the dynamic load, the current associated with changing output states. In this case, fanout is limited by the acceptable speed of the system.

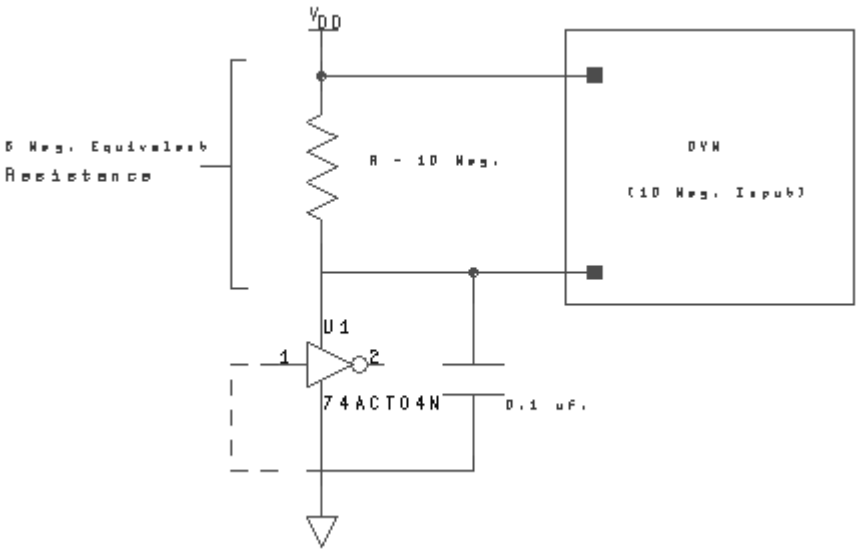


Figure 2-iii: Scheme for CMOS current measurement.

Power: The power used within a chip is the product of the current *through* the chip from its V_{CC} terminal to its GND terminal times the voltage *across* those terminals, that is: Power = (current through) x (voltage across). This is the rate at which the power supply is doing work (mostly heating the chip) on the circuit.

Since a flow of current can only be measured ***by routing it through a meter***, put the meter "in series" with the supply and the V_{CC} terminal. (Be sure to have a bypass capacitor across the integrated circuit too.) If you don't understand the term "in series", please ask a TA about it. Misconnecting the meter in current mode can damage it! If the meter doesn't register any current, ask a TA to check the fuse in the meter. For the 74LS14, you can measure the current with the DVM on a suitable current range. For the CMOS chip, the current will be too low to measure accurately with the DVM on any of its current ranges. However, if you connect a 10 Megohm resistor in series with the V_{DD} terminal, as shown in Figure 2-iii, and use the DVM on a *voltage* range to measure the potential drop across the resistor, then Ohm's law can be used to find the current. The resistance of the meter itself when used as a voltmeter is 10 Megohms. Since this is not negligible compared to the 10 Megohm sensing resistor, one has to use the effective resistance of the combination of the two in parallel, namely 5 Megohm. The corresponding voltage across the chip is the power supply voltage minus the drop across the series-measuring resistor. **WARNING:** Do not try to measure the voltage across the inverter and infer the current through it by Kirchoff's voltage law. The multimeter current is not negligible compared to the CMOS current.

9.2.4. Lab Three

Multiplexed Display

Requirements: Program one of the Engineering 163 CPLD boards to act as a seven-segment decoder for its on-board display. (Please see the Appendix section 10 for information on this board and for notes on the programming procedure for the XC9572XL on the board.) The resulting characters should have unique proper forms for all 16 possible inputs as shown in Fig. 3.i. Using the discrete parts from your bag-o-chips, wire your 8-bit DIP switch so that the upper and lower sets of four switches control the left and right hexadecimal character of the on-board display. Use pull-up resistors as appropriate. Arrange a circuit with a toggle switch such that in one position of the toggle switch the display shows the two numbers set on the DIP-switch with no flicker in either digit. The display should update immediately when the DIP-switch setting changes. When the toggle switch is in the other position, the display shall show "A" on the left digit and the right digit shall be off entirely. I realize that in principle **this multiplexing could be done in the CPLD but that is not allowed** for this lab, which aims to show you how bus multiplexing is customarily done. The only connections to the user cable of the CPLD board are to the two cathodes, the four data lines, and two power connections. If you need more pull-up resistors than are available in your kit, you can get those in the lab.

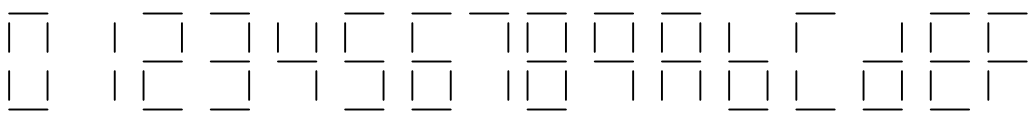


Figure 3.i: Hexadecimal characters on a 7-segment display

As with all labs requiring programming, you must have a hard copy of the program for your device ready for the TA. FTQ's may require you to predict the effect of a change to that code, and you must show mastery of the programming sufficient to carry the suggested change through to a demonstration on your hardware.

Discussion: Figure 3.ii shows how the toggle switch connects between terminals in its two positions.

To use the CPLD Board's display, you need to determine the pinout of the display chip. One way to do this is to use a loose display and to consult the earlier discussion of Displays and LEDs (Lab Minus One) for how to do it. Probe pairs of display pins with a ground wire and a wire connected to a 100 Ω resistor, the other end of which is connected to the +5 volt supply. You will soon discover which two pins are the common cathodes for the digits, which pins control the seven segment anodes, which ones control the decimal points and which pins have no connection (NC).

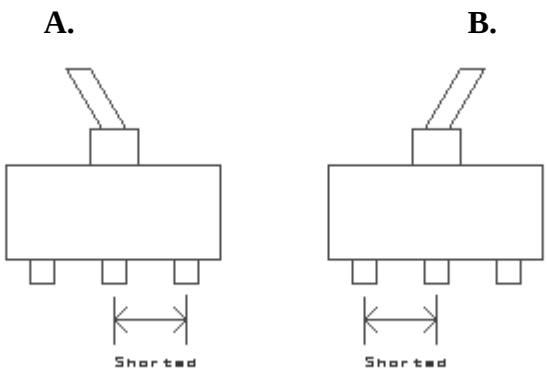


Figure 3-ii: Toggle positions and resulting connections.

<u>WARNING!!</u> When you start to use a CPLD board for the first time, you *<u>MUST PROGRAM IT BEFORE YOU CONNECT IT</u>* to your protoboard. The CPLDs are easily damaged if an output pin is connected to a signal source, as it might be if a prior user had different pinouts programmed than you use. To power up the board for programming there are special power cables that connect to the ribbon cable plug on the board. Ask a TA!

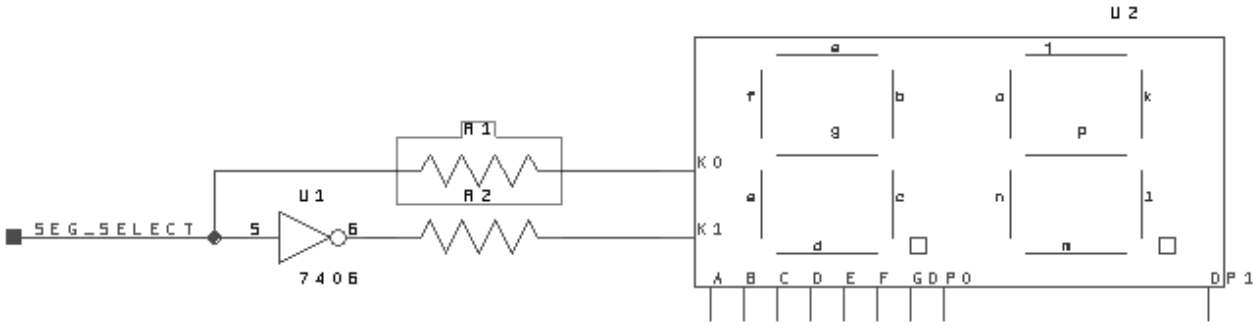


Figure 3-iii: Cathode drive circuit with current limiting resistors; R1 and R2 are already on the CPLD Board. The 7406 is not, and the reason for its use is that the required display current is larger than the XC9572XL output rating.

You want a flicker-free (>50 Hz) rate of switching back and forth between the halves of the display. In one phase the left digit will be on for a few milliseconds while the right digit is off, in the second phase the right digit will be on while the left is off. Remember that a display digit is ON when the cathode pin for that side is grounded by a LOW output on the logic gate driving it. Also remember that the current required to drive all segments of the display is too large for a simple TTL gate and is marginal for the XC9572XL. I show 7406 inverters in this application because they are capable of sinking 40 ma and the display requires 40 ma for an "8" with decimal point. To control switching, you can build a pulse generator using the NE555 timer chip to drive one side. An oscillator example circuit is included in the data sheets at the back of this manual. The output transistor of the NE555 can "sink" the current from one digit, while an inverter derives the signal to switch the other digit. Given that one digit must be blanked part of the time, you may need more complex logic than is shown in Fig. 3-iii.

Once you are able to switch the two digits off and on rapidly, you will need to coordinate that action with the proper signals for the segments. You will need to *multiplex* the four inputs to the XC9572XL with three different signals. You will want the oscillator and the output of your toggle switch to control that multiplexing.

See Lab 5 and section 10 of this manual for more information on the CPLD-II board, the ABEL programming language, and downloading procedures.

9.2.5. Lab Four

Keyboard Encoding

Requirements: Arrange a circuit such that each of 16 different buttons on your keyboard represents a different hexadecimal number on one digit of the LN524RK display on a CPLD board. (Consult Lab 3 for further information about these displays.) It is not necessary that the displayed digit correspond to the key marking, but that would be nice. Have only the decimal point and nothing else light when no button is pressed and have the decimal point go off when anything is pressed. You may use any amount of combinational logic in the CPLD that you wish, but any flip-flops you need must be external. (This circuit may use a counter of some kind and I want you to assemble that in some fairly primitive way to learn a bit about counters.) As usual, you must have a current schematic and a hard copy of any program you use when you ask for evaluation. You must design your circuit so that it is safe for any number of buttons to be pressed simultaneously. Remember that pressing more than one button at a time may connect gate outputs together, a condition called output contention. Such contention can cause unacceptable current and power levels when gates pull against each other. That must be avoided. The TA will want to know how you have prevented contention by design as a separate issue from any FTQ.

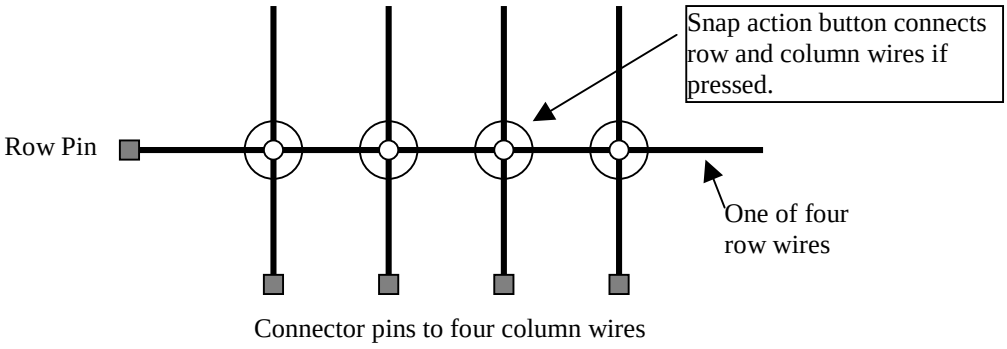


Figure 4-i: Connecting the row and column wires via a button press.

Discussion: Your keyboard has a matrix of 4 by 4 wires (8 wires total). When a button is pressed a unique pair of row and column wires is connected (shorted) together. (See Figure 4-i.) You will need to devise a method for determining which pins go with which row or column. One possibility is to use the ohmmeter of the DVM from the Lab 2 set-up.

You may want to build a circuit that *scans* the keyboard by sending test signals to the columns while the outputs of the rows are analyzed. For the fault tolerance question we may ask you to predict what response your circuit will give when *two* buttons are pressed simultaneously and why! Read section 5.7 of Wakerly about multiplexing and demultiplexing; this should be very helpful.

A word of warning: the keypads are easily destroyed by melting the button supports with too much current. The commonest way this happens is to use a button to short the power supply from VCC to GND or to a TTL output that is low. **DO NOT CONNECT A KEYPAD WIRE TO VCC EVER!** Use 1 K pull-up resistors on either the row or column wires and let the keypad pull down to GND or to the output of a gate.

9.2.6. Lab Five

Counter with External Control

Requirements: This lab must be built with the Xilinx XC9572XL logic chip on the class CPLD-II board. This time you may use the flip-flops inside that device. (Programmable logic is very important in prototyping and small-volume production. Here is where you start to learn its full capabilities. I have chosen the Xilinx parts because they are inexpensive, Xilinx supports us with software very well, and these devices are currently popular ones of their kind. Other vendors make similar product. For example, PAL and GAL are acronyms for Programmable Array Logic and Generic Array Logic devices that are smaller variants of the PLD. The acronyms are trademarks of Advanced Micro Devices Inc. used for products from Vantis, and from Lattice Semiconductor both of which also make larger CPLDs of their own types.)

Design and build a counter circuit that is clocked by a switch or pushbutton with no skipped steps. The output of the counter should be displayed on the seven-segment display on the CPLD board. The system has the following features: There are two control signals that you can generate with two sections of your DIP-switch. One such signal is called "Abbreviate" and the other is called "Halt." When the clock switch is flipped back and forth with *no* control signal asserted, it is in a "continue" mode, and the display steps through the count 2, 3, 1, 6, 0, 2, 3, 1, 6, 0, 2, etc on the right display digit. When the "Abbreviate" signal is asserted while the counter is clocked, the sequence cycles normally up through the number "2". On the next clock, the counter should begin the 'abbreviated' sequence 6, 5, 6, 5, 6, etc. The abbreviated sequence, including the first 6 should display on the left digit! If the system is displaying 6 on the right digit at the time "Abbreviate" is asserted, it should go through the entire normal sequence on the right digit before going to the abbreviated sequence on the left display digit. If the Abbreviate signal is deasserted in state 5, the system should go to 6 (on the left digit) before resuming its normal sequence. If the Abbreviate signal is not asserted in state 6, the system should go to 0 (on the right digit) next and continue its normal sequence. While the "Halt" signal is asserted, the counter should not change state when toggling the clock switch any number of times. Similarly, if the clock switch is left in one position, the output of the counter should not change when the "Halt" signal is toggled, regardless of which position the clock switch is in.

Challenge to the Bored: If you accept the challenge, the 6, 5, 6, 5 abbreviated sequence should count slowly (about once per second) by itself without your needing to pump the toggle switch. The toggle switch will now have no effect. When the abbreviate is asserted and the counter has toggled up to 6, the first 6 on the left digit must display for at least a second, maybe longer, and not be skipped as the steady automatic counting starts. "Halt" also stops the automatic counting in the abbreviated count mode. (The point of this subsidiary exercise is to get you thinking about the problems of switching clock sources. You may exploit the relative slowness of the clocks in this particular problem.)

You ***may NOT use the STATE MACHINE*** facilities of the ABEL language or the programming software for this lab. (One of the major pedagogical purposes of the lab is to make clear

to you what such machines are all about. Automating the synthesis would defeat that goal.) You will have to have a printout of your data entry file for the XC9572XL. The TA may base her FTQ on a change to that file. (Remember: that file must be your own work -- original design and actual typing.) You will still have to wire up the input circuitry (including the debouncing circuit) and the cathode drive connections. (And any circuitry to support the Challenge to the Bored if you are bored.)

WARNING!! When you start to use a CPLD board for the first time, you *MUST PROGRAM IT BEFORE YOU* connect it to your protoboard. The CPLDs are easily damaged if an output pin is connected to a signal source, as it might be if a prior user had different pinouts programmed than you use. To power up the board for programming there are special power cables that connect to the ribbon cable plug on the board. Ask a TA!

Discussion: Debounce your toggle switch or clock button so the counter does not skip numbers as it sequences. The introduction of this manual describes two possible procedures for preventing false clocking by the switch. Read chapters 7, 8, and section 9.1 of Wakerly for a discussion of flip-flops and counters. Do your design systematically with the usual tools for finite state machine design. *Ad hoc* methods may work, but you should avoid them as they are basically dead ends. As usual, we will ask you a Fault Tolerance Question and it may be based on the code in your PLD. You **must have a hard copy** of that code available to the TA at the time of evaluation and that copy must be current.

Programming the CPLD requires telling appropriate software how the flip-flops and gates of the device are to be configured. For larger devices, this can be done with the VHDL or Verilog hardware description languages. However, those languages cover a wide range of expression for functionality and are designed for efficient representation with only limited regard for the relation between expression and machine structure. I believe the older, somewhat less complex language, ABEL, will link you better to the actual hardware. My thinking is akin to the once common practice of using PASCAL to teach introductory programming because it was designed with teaching in mind. The Xilinx ABEL compiler runs on the machines both in the Hewlett Computing Facility and in room 196 so designs can be revised as you debug them.

The following example shows the syntax of an ABEL file and is annotated to explain some of its features. It is compiled into JEDEC files (".jed" extension) to encode the pattern of gate charges needed to program the device. That data is downloaded into the XC9572XL through a JTAG port while the device is actually hooked up to your protoboard. (Lab C is about the JTAG standard and the ideas behind it and there are references given in that lab that explain the idea further.) The ABEL language is actually fairly complicated and sophisticated. It is capable of much more than the simple registered logic for which we use it. There are references to the language in Wakerly, and in the on-line help sections of both *DxDesigner* and the Xilinx ISE software.

The inclusion of a comment block at the beginning with your name, the date and a circuit name is *NOT OPTIONAL*. You must include such a section properly filled in. In the body of the file, the lines containing the keyword "PIN" assign signals to specific pin numbers and set the functions of the pins including whether they are attached to flip-flops or not. Because there are several registered nodes in the XC9572XL devices that are not connected to external pins, it is possible to assign some output signals that are not needed externally to buried nodes. The syntax to do that

uses the keyword NODE in place of PIN and leaves out any node number. The attribute following the keyword ISTYPE determines if a pin is an input (default), a combinational output (COM), or is attached to a D-flip-flop (REG). Comments are set off by a quote or double slash. There are two possible sets of Boolean operators. The default set is !, &, #, $, and ! for NOT, AND, OR, XOR, and XNOR. If the "@Alternate;" statement is included, then the symbols /, *, +, :+:, and :*: are the logical operations NOT, AND, OR, XOR, and XNOR. For example, the line

Q1 = /Q2

means Q1 is the complement of Q2 and is equivalent to driving Q1 from Q2 through an inverter. Variable names must begin with an alphabetic character and contain no more than 12 alphanumeric characters total. The language makes no distinction between upper and lower case letters. I suggest making variables in all uppercase characters as being more readable. There is a set of operations available in the language specifications that automates the synthesis of finite state machines. I have not included examples of such statements because you are not allowed to use them in this lab. (You may do so in labs 7 and A and will find examples of the syntax in both Wakerly and the *eProduct Designer* help files.) An abridged data sheet for the XC9572XL including pinout and reserved pins is in the device data section of this manual. There is a complete set of data sheets on the class web site that were taken from the Xilinx site.

9.2.6.1. Lab Five: ABEL Implementation of a Counter – An Example

```
Module  ctrpld2  "Note: a 'Module' line followed by project name is required.  This statement
                  “marks the beginning of a section that must terminate with an 'END' statement.
                  "Also the quote character marks the beginning a comment that extends until the next
                  “quote or the end of the line.

Title 'CPLD Board version of 2-Bit Counter'

“NAME: WRP    DATE: 10/22/2002 //Mandatory NAME/DATE comment! (WRP’s Mandate!)

@Alternate;             //Changes the Boolean operator set to what I like. (Not mandatory) You
        // can leave it out and use the operators ! & # $ !$ instead. The double '//' starts a
        // comment the same way a quote does, but the comment ends with end of line regardless of
        // any slashes or quotes in the line.

Q1, Q0 PIN  14,18           ISTYPE 'REG';     //Assigns the signal names Q1 and Q0 to pins 14 and 18
                        // respectively. Also directs that these are registered (D-flipflop) outputs.
                        // More than one signal/pin number of the same type can share a line.
GOING  PIN   13   ISTYPE 'COM';     //Combinational output called GOING on pin 13.
CLOCK PIN       5 ;         // CPLD wiring forces clock to be on pin 5.
/OE   PIN   42;             // Negative true input.  Global tristate must be on pin 42.
UP    PIN   2;
HALT  PIN   3;

EQUATIONS

[Q1..Q0].CLK  =  CLOCK;  //The notation [Q1..Q0] specifies all the signals in
                //the "range" q1 to q0.  This does not save much space here,
                //but it would help, for example, with [q10..q0] meaning q10, q9, q8, q7,...q1, q0.

[Q1..Q0].OE       =  OE;     //Sets output enable of outputs from OE input.
GOING.OE          =  OE;

“The equations below specify a 2-Bit up/down counter with HALT and output enable features.
“The := symbol is the assignment for a registered signal and it implies that the D-input of the
“register is the Boolean expression on the right.

Q1        :=  HALT*Q1 + /HALT*(UP*Q0 + /UP*(/Q1*/Q0 + Q1*Q0));
Q0        :=  HALT*Q0 + /HALT*/Q0;

GOING  = /HALT;  //Note that ‘=’ is the symbol for combinational assignment.

// Required END statement to match MODULE statement
END;
```

9.2.7. Lab Six

Propagation Delay Measurements

Requirements: Using a high-speed oscilloscope, measure the timing properties and power dissipation of several of your chips. Also probe the behavior of the 74LV04 chips as a function of power supply voltage so you will see what lowering that voltage does to speed and power. *Hand in a report* describing your results, in the spirit of Lab 2. The report must be **typeset** and should include three things: first, an introduction explaining in your own words why these measurements are of interest to a system designer. In your discussion, distinguish between your measurements and the *worst-case* conditions for the same values. Second, give the results of your measurements - clearly labeled and tabulated with units. Finally, answer the questions that follow the measurements. Make sure you see a TA if you are unfamiliar or uncomfortable with the use of an oscilloscope.

There are several instruments that you may choose from for this lab. The 100 MHz digital oscilloscopes at each workstation are quite satisfactory for most purposes but may make some transient events appear slower than they really are. There are also two 500 MHz scopes on rolling carts that will resolve much more interesting behavior in your circuit. Finally, there are scopes built into some of the logic analyzers and these have time resolution similar to the 500 MHz stand-alone scope. The stand-alone instruments have advantages in dynamic range and storage depth but the logic analyzers let you do triggering on logic conditions. You are free to choose among these tools, but I **strongly recommend using the fast scope** for the setup and hold measurements. Manuals for everything are in the lab, and the TAs will do their best to help. However, they will be little better off than you as far as prior knowledge of these instruments is concerned because they probably do not have much experience.

Measurements: First, measure the propagation delay for both possible directions of input transition (t_{LH} and t_{HL}) on three of your chips: 74LS14, 74LS240, and 74LV04. Also measure the output rise and fall times for these gates. Use the 3.3 volt fixed power supply when doing these measurements on the 74LV04. Delay times are measured between "gate threshold" voltage levels and you should take those values for the LS chips from your measurements for Lab 2. The reference level for the 74LV04 is .5*VDD or 1.65 volts. There are several ways one might define the rise and fall times. As shown by t_r and t_f in Fig. 6-i (b), we will define them in terms of the time to go between the defined HIGH and LOW voltages. For the LS parts, these values are 2.4 and 0.8 volts. The 74LV04 parts use .7*VDD and .3*VDD levels. With the 74LS240 also measure the delay from the enable input to the gate output for both directions of the enable input transition. To make the change of state visible use a 1 K ohm pull-up resistor on the output and tie the gate input to make the output LOW when enabled. Such a circuit makes the relation between the enable input and the output similar to that between the input and output of an inverter.

Figure 6-i (a) shows the measurement arrangement. Adjust the oscilloscope so that the horizontal sweep is triggered **only by the input** waveform. Then superpose the input and output waveforms, being careful to match the zero levels exactly. Measure both t_{LH} and t_{HL} between the threshold crossing points of the input and output waveforms, using the threshold voltages determined from Lab 2 as the reference levels for the time measurement. In your report, please be sure to spec-

ify what level you used. See Fig. 6-i (b) for a sketch of what the scope trace should look like. These delay times will be quite short, typically a few nanoseconds. To measure them, you will need to use the fastest horizontal sampling rates possible and will find the measurement resolution is coarse. Nonetheless, these are not negligible times for any serious attempt at a fast system.

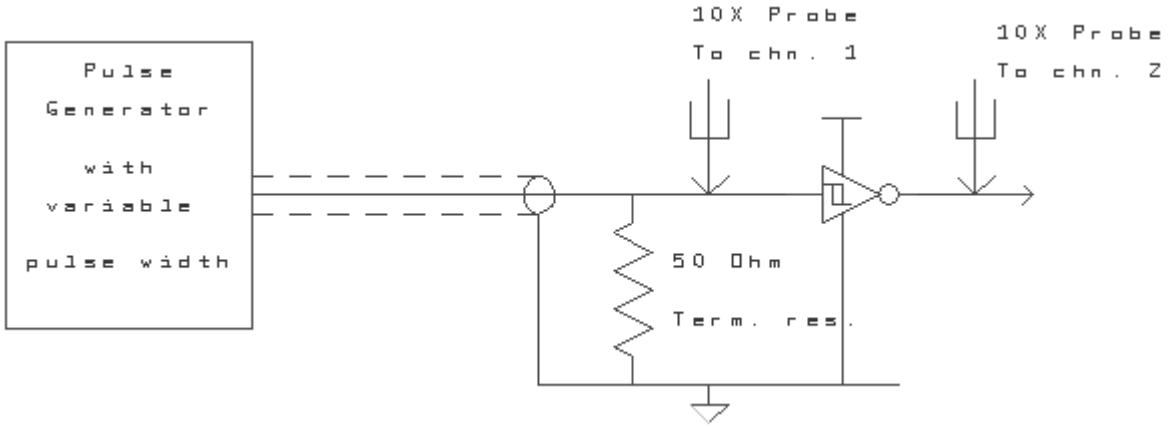


Figure 6-i(a): Test setup for propagation delay

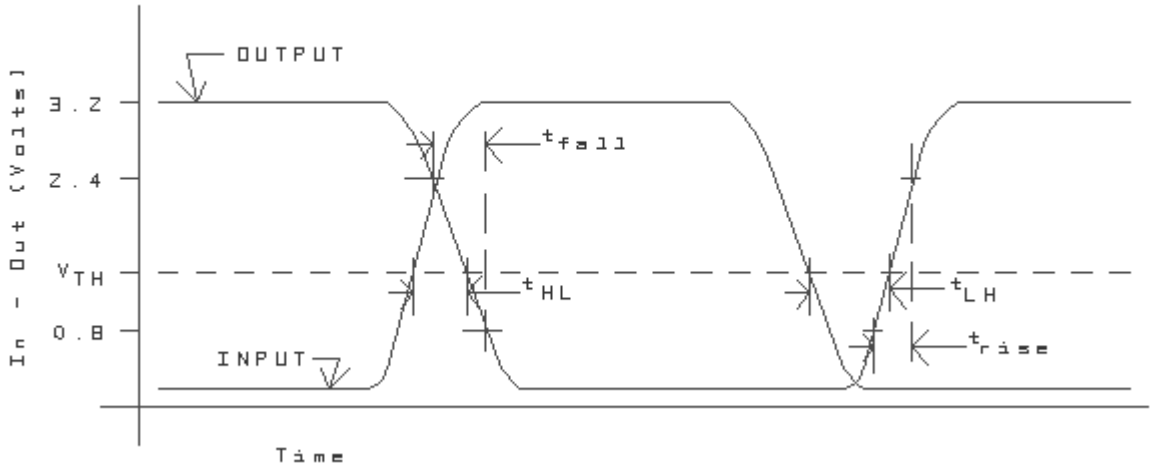


Figure 6-i(b): Superposed, delayed waveforms

Second, connect five of the six inverters in your 74LV04 in tandem to form a ring oscillator, as shown in Figure 6-ii. This time use a **variable** power supply and make separate measurements with 2.0, 2.5, 3.3 and 5.0 volts on VDD. For each supply voltage, determine the frequency of oscillation. Connect a 50 pf. capacitor from the output to ground. What are the new frequencies of oscillation?

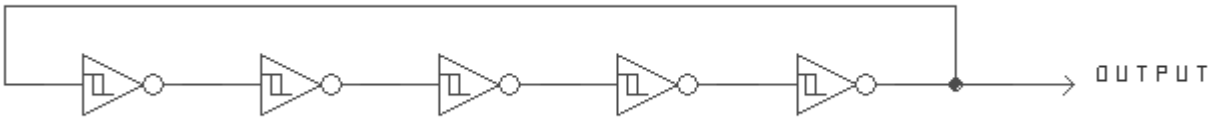


Figure 6-ii: Ring oscillator

Third, measure the power consumed by your ring oscillator at the same set of supply voltages. You should have a bypass capacitor between the VCC and GND pins on your chip. You can measure the current into the oscillator by connecting the ammeter part of the DVM from Lab 2 between the positive supply connection and the VCC pin of the 74LV04A. This is the same arrangement as you used in lab 2 to measure the current drawn by your TTL chips. (Check that the frequency and amplitude of the output are not changed appreciably by your ammeter.) Also measure the **change** in power dissipation of the ring oscillator if you load one gate with a 50 pf. capacitor from output to ground. [Note: it may be possible to use the meter in the power supply to measure current too.]

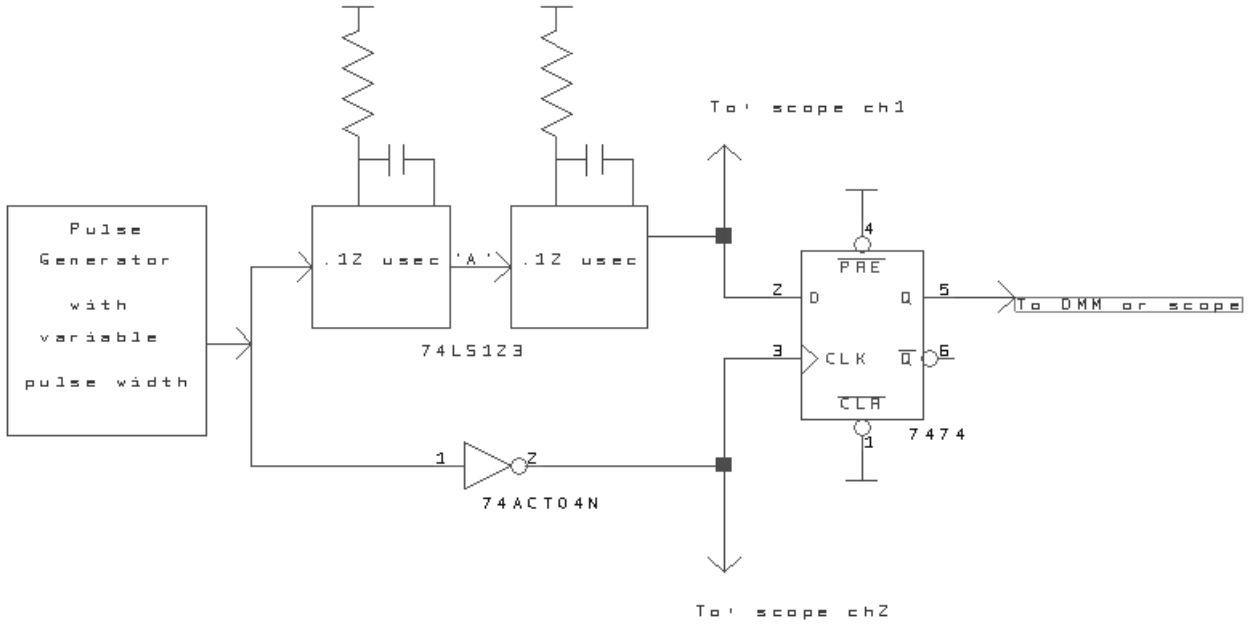


Figure 6-iii(a): System for measurement of setup and hold times

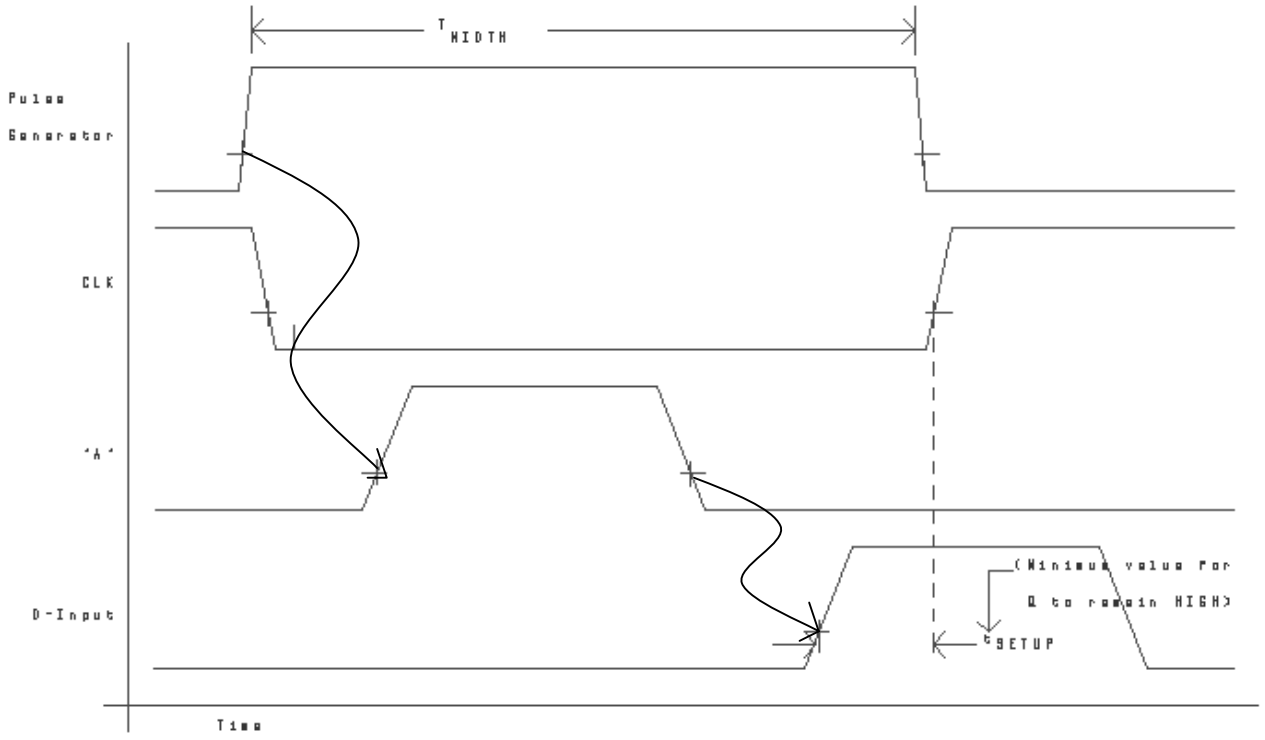


Figure 6-iii(b): Timing diagram for measurement of setup time

Finally, measure the setup and hold times for the D input of one half of one of your 74LS74 dual D-flip flop chips. Make measurements for two cases, first for the D input high before the rising clock edge and then for it low before the clock edge. (The first case corresponds to a low-to-high transition on the output; the second case to a high-to-low transition.) Figure 6-iii shows one way to do this using a variable width pulse generator as a signal source. (We have some home-made pulse generators that allow very fine adjustment of the width of a pulse with a very fast edge. There are also some new Agilent pulse generators. You may choose between them, but I believe the home-made ones may still be the best for this particular task.) The two halves of the 74LS123 should be set to give pulses at the D input of the 74LS74 which are about .15 microseconds wide starting about .25 microseconds after the leading edge of the input clock. (I call your attention to the data sheet limitation that the timing resistor used with the 74LS123 must not be smaller than 5 kilohms.) Please note that the **second section of the 74LS123 is triggered by the trailing edge** of the first section as shown by the lines of causality in the timing diagram. Be sure to connect the two sections together in such a was as to get that relationship. By varying the pulse width of the incoming pulse, you can move the rising edge of the 74LS74 clock relative to the pulse on the D input. The setup pictured in the timing diagram in Fig. 6-iii is for the D input driven by the Q output from the 74LS123 and so measures the setup time for a low-to-high transition of the flip-flop. In this figure, the setup time is simply the minimum time lag between the leading (rising) edge of the D pulse and the clock edge for which the 74LS74 output will remain HIGH, that is, will match the D pulse. Similarly the hold time is the minimum time lag between the falling edge of the D pulse and the clock edge for which the 74LS74 output will remain HIGH. The second case for a

HIGH to LOW transitions on the output of the 74LS74 is measured by changing the polarity of the pulses to the D input. You can do the inversion by changing the connection from the 74LS123 from Q to $\overline{Q}$. In this case the times are the minimums that keep the 74LS74 output LOW.

The setup and hold times are very fast. Setup time is likely to be in the 1 to 5 nanoseconds range and hold time may sometimes even be negative. (Negative hold time just means that the D signal does not have to persist all the way up to the clock transition to be recognized!) You will need to be very careful to keep the zero levels of the two channels of the scope the same and to use a consistent threshold level around 1.3 volts.

Questions:

1.) What delay would you measure if you put an even number of identical inverters in tandem and measured the total delay of the circuit? Suppose the number of inverters is $2 \cdot N$. Express your answer in terms of N, and the times t_{LH} and t_{HL}.

2.) Why is the reference voltage level for the delay measurements chosen to be V_{THG}?

3.) Which of your rise, fall, or delay measurements in the first measurements of this lab are seriously limited by the speed of response of the oscilloscope? (The apparent rise time of a very fast pulse on the 100 MHz scopes in room 196 is about 3.9 ns. There is a very good match between the speeds of the two vertical channels.)

4.) In your ring oscillator measurements, what determines the period of oscillation? How does it relate to the measurements of t_{LH} and t_{HL}?

5.) As power supply voltage goes down, what happens to the speed and power dissipation of a gate (or system)? (Show a plot from your ring oscillator data.) Does the power dependence on supply voltage match predictions from class? (I expect a quantitative comparison to an actual formula.) The industry trend is toward lower supply voltages. Does this make sense for speed? Why? Why do people lower this voltage if they are not concerned about saving fossil fuel?

6.) A CMOS gate input appears to the output of the gate which is driving it as a capacitor to ground. Based on what happened with the 50 pf. capacitor and based on the typical data sheet value of the gate input capacitance, what should be the effect of connecting each output of the ring oscillator to two additional gates? (This is a total of 10 gates beyond the 5 in the oscillator itself. See the data sheet for the typical capacitance per input gate connection.) This is the effect of fanout in a system. Give a qualitative answer for speed and a quantitative one for power.

7.) How does the power dissipation of the operating ring oscillator compare to power measured for the same chip in Lab 2? Why? What is the *effective* capacitance charged and discharged by one gate on each transition of its output?

8.) Is the *change* in power dissipation of the ring oscillator when you add the 50 pf. capacitor to the circuit consistent with the theoretical value? (Calculate a theoretical value for the change and compare it with the measured value.)

9.) Your measurements are a single sample for each device or circuit you use. As such they do not reflect the worst case conditions. How do your measurements of setup and hold time for the 74LS74 compare to the data sheet values for *typical* and *worst-case* conditions?

Discussion: In all measurements it will probably prove necessary to **place a bypass capacitor** (0.1 μfd.) directly between the VCC/VDD and GND pins of the chip being measured. For best results, keep the capacitor leads short.

The pulse generators we suggest you use in this lab were home-built to give fast rise times and finely adjustable pulse widths. Both the generator and the coaxial cable used to connect it to your circuit require that you terminate the output with a resistor between 47 and 51 ohms mounted at the circuit end of the cable as near as possible to the gate being tested. (See Figure 6-i (a).) Our pulse generators have the terminating resistor plugged onto a tee at the end of the cable, and no extra resistor is necessary on your board. However, be sure that the pulse generator you use does have a tee with both leads and a terminator on it. Also keep the leads from the tee as short as possible and keep them close together. These comments about terminating cables and wiring to your board are equally true of the Agilent pulse generators. If you choose to use them, you will still have to be careful of the cabling. They are also not as easy to adjust as the home-built units.

Use two **identical** 10X probes to monitor the input and output waveforms. Signals travel down the oscilloscope cables at about eight inches per nanosecond. If the cables are not the same length, then the delays you measure will be off by the difference in the times it takes the two signals to go down their respective probe cables. Please be careful with the scope probe tips! (They break easily and are very expensive.)

Because the bandwidth of the oscilloscope is finite, there is a limit to how fast the scope can move the electron beam that makes the actual display. This limits the apparent fastest rise time to a little under $\frac{1}{\pi F_{BW}}$ where F_{BW} is the bandwidth of the scope. For this reason the 100 MHz scopes are limited to about 3.9 ns apparent rise time, and the scope time resolution is limited around 1.0 ns. The 500 MHz scopes are proportionately faster.

9.2.8. Lab Seven

Dual Slope A/D Converter

Requirements: Design and build a 5-bit A/D converter based on the dual slope integrator technique. Generate a variable input signal in the range of zero to plus 5 volts with your potentiometer. Display the answer on one digit of your display plus an LED for your MSB. The circuit should update the display a few times a second or so, so that it follows the manual rotation of the potentiometer. For Lab 7 to be signed off, we must see all the digits in proper sequence as the pot rotates. It is possible in testing the lab that either 0 or 31 will not appear because of mismatched resistors or because of offset in the comparator. This is an allowed deviation from the full sequence of digits.

Figure 7-i shows the recommended system in block form. The placement of the analog switches in the circuit is dictated by the requirement that the potential connected to any input of the particular switches you are using must be between +5V and ground. If you wish to try any variation on the placement of these switches, please keep this constraint in mind.

You will need at least a five bit binary counter. The four least significant bits **must** be realized with the 74LS569 counter. (I want you to know about counter chips.) You may use the CPLD board for any additional control logic, for the display, and to extend the counter if you so wish. The total number of cycles of the master clock in a conversion cycle may not exceed 66. The TA may check this by looking at the duty cycle of C1 with an oscilloscope.

Arrange your clock frequency to minimize the sensitivity of the circuit to 60Hz noise. This is done simply by making the signal integration period an integer multiple of the period of 60 Hz line noise, i.e. a multiple of 1/60th of a second, so that the integral of any noise injected from the AC power lines is zero.

Discussion: Dual slope analog to digital converters are widely used whenever relatively slow conversion rates are required. One application that you have seen already, probably without realizing it, is the digital voltmeters used in Lab 2. Other applications include thermocouple readouts, and appliance controls. The reason for this popularity is that the method allows very low cost, high precision conversion. Single chip 13 bit converters are available for under a dollar in quantity. If used with care, the technique is capable of realizing very high precision. It has been used, for example, to build high precision DC voltmeters with more than 20 bit resolution.

Labs 7 and 8 make use of operational amplifiers (op-amps to the initiated), which are discussed in passing in J&K, Ch. 14, and at length in T&S chapter 2. Figure 7-iii shows a conceptual view of such a device. An amplifier is simply a circuit that has an output signal that is *proportional* to its input but larger in magnitude. It differs from a logic device in that the relation between input and output is linear (straight line) rather than deliberately distorted as is done in digital systems to exploit the HIGH/LOW nature of digital signals. An op-amp is a particular type of amplifier which has a large gain ($A > 10^5$) and a high input impedance (many megohms). It is a widely used com-

ponent because it can be tailored to a wide range of uses with external resistors and capacitors. You have two op amps in your kit in the LF353 dual op amp chip. In these labs they are used as *interfaces* between the digital and analog domains.

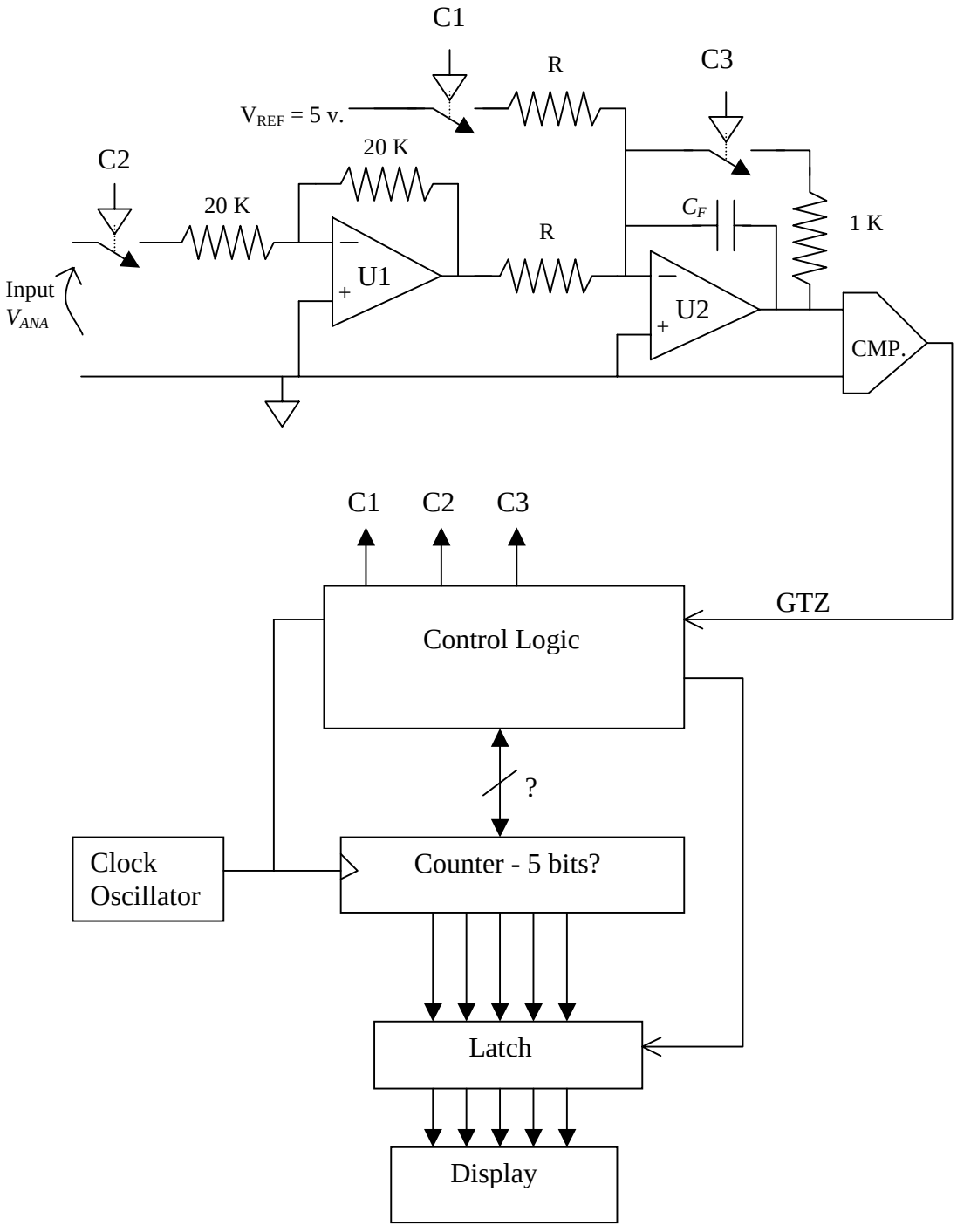


Figure 7-i: Block diagram of a dual-slope A/D converter

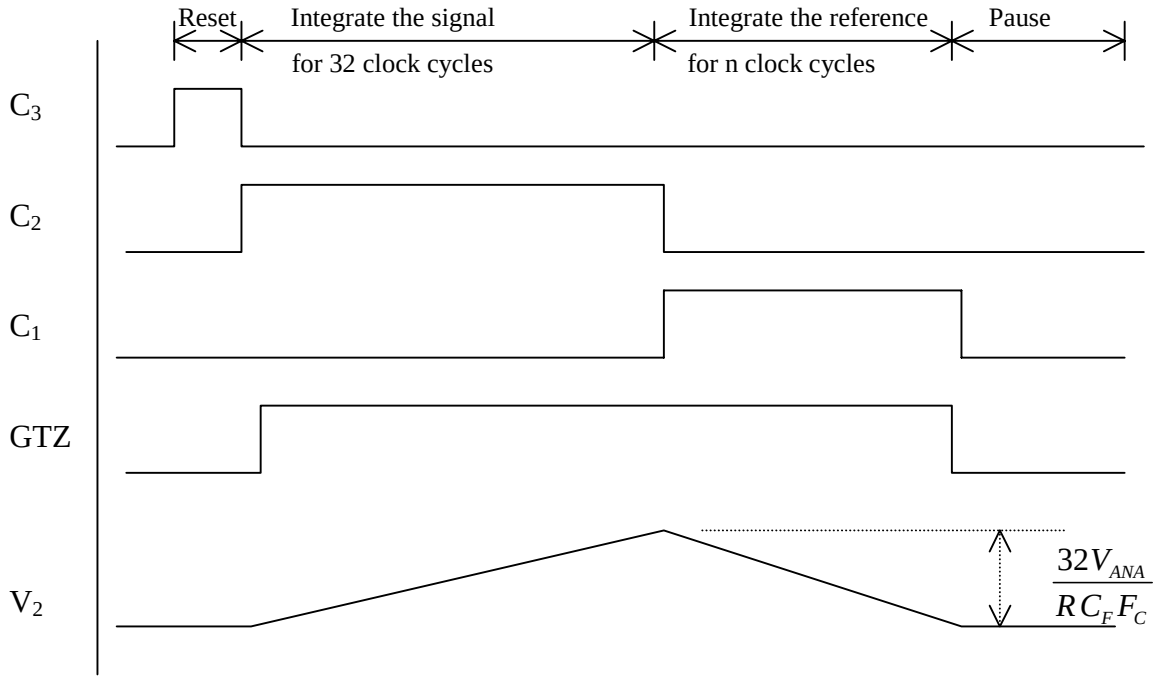


Figure 7-ii: Timing diagram of the dual-slope A/D converter

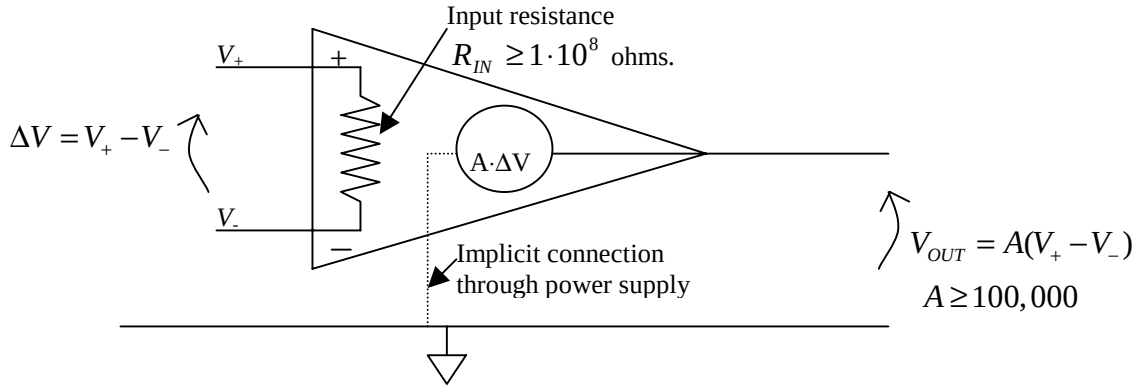


Figure 7-iii: Operational amplifier conceptual view

The LF353 requires +12v on pin 8 and -12v on pin 4, but it does not have a separate ground pin. In the configuration of Figure 7-iv, an op amp provides a *negative linear* gain for output in the range $-12_V < V_{OUT} < +12_V$. The magnitude of this gain is $\frac{-Rf}{Rs}$, and depends only on the resistor values. If V_{IN} and/or the gain $\frac{-Rf}{Rs}$ is too large, the op amp output will "saturate" at nearly -12 or +12 volts. (Saturation of an amplifier is simply a condition in which the output is unresponsive to the input because it is already as high or low as the available power supply will let it get. Such a

condition roughly corresponds to the HIGH or LOW state of a digital output.) Amplifier U1 in your ADC (Fig. 7-i) is connected in precisely this configuration with $Rf = Rs$. Thus the voltage at the output of U1 is equal to minus the analog input when SW2 is closed and zero when the switch is open.

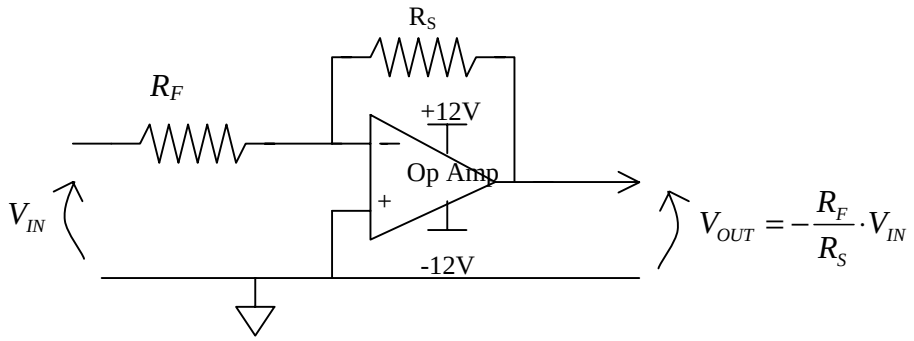


Figure 7-iv: Op-amp as a negative gain, wideband amplifier.

The term "operational amplifier" comes from the fact that these devices can be made to do arithmetic operations on analog signals, including the operation of integration with respect to time. In your system, the second op amp, U2, is wired as a resettable integrator; figure 7-v shows that part of the circuit separately. When the switch SW3 is closed, the output of the amplifier is connected directly to its inverting input. The output potential V_2 is zero, and there is no charge stored in capacitor C_F. (Remember the relation between charge in a capacitor and the potential across it is: $Q = C_F \dot{V}$.) Because the gain, A, of the amplifier is very large, the potential across the input terminals, which is $\frac{V_2}{A}$, is always nearly zero so long as the output is not in saturation. When SW3 opens, say at t = 0, the current flow through resistor R must go either into the amplifier input or into the capacitor C_F. Since the amplifier input resistance is very large and the input voltage is very small, most of the current will go into the capacitor. The current is given by Ohm's law as $\frac{V_{sig}}{R}$, and is the rate of flow of charge into the capacitor, i.e. $\frac{dQ}{dt}$. Thus

$$\frac{dQ}{dt} = \frac{V_{sig}}{R} = -C_F \cdot \frac{dV_2}{dt}$$

and

$$V_2 = \frac{-1}{(R \cdot C_F)} \int_0^t V_{sig}\, dt$$

To see how this might work, consider applying a constant voltage at the input *Vsig.* The current through R is simply a constant, $\frac{V_{sig}}{R}$. At t = 0 the output voltage will begin to go negative starting

from zero; the rate of change will be $\frac{-V_{sig}}{(R \cdot C_F)}$ volts per second. The output will look like a linear ramp, which attains a value of $-V_{sig} \cdot \frac{T}{(R \cdot C_F)}$ at time T.

The dual slope A/D converter works by coupling the resettable integrator with switches that change the input to the integrator, with a comparator which determines whether the output of the integrator is positive or negative, with a counter that measures out time intervals and with some logic that executes a simple control algorithm. The process begins with a short period in which SW3 closes to reset the integrator. This time is shown on the timing diagram, Fig. 7-ii, where C_3 is "high." Such a **closure of SW3 must occur once each** conversion cycle regardless of what the comparator signal is like. This period ends when the counter clocks into its zero state. Then the process proceeds the following way:

(1) SW3 opens and SW2 closes for N clock cycles; in this case N = 32. During this time, U2 integrates the input signal V_{ANA} after it has been inverted by U1. By the argument given above, the output of the integrator at the end of this time will be $V_{ANA} \cdot \frac{N}{(R \cdot C_F \cdot F_C)}$ where F_C is the clock frequency since $\frac{N}{F_C}$ is the integration time, i.e. the length of time of this part of the conversion. This situation is shown on the waveform of V_2 in the timing diagram.

(2) SW2 opens and SW1 closes as the counter counts over to zero again. The integrator then integrates the constant +5V reference voltage, and its output voltage decreases linearly with time. When the output reaches zero as determined by the comparator, the contents of the counter are latched into the output register as the digital output of the system, and SW1 opens to prevent the integrator output from going excessively negative and damaging SW3. The change in the integrator output during this part of the cycle is:

$$\Delta V_{INT} = 5 \cdot \frac{n}{(R \cdot C_F \cdot F_C)}$$

where n is the number of clock cycles in this period, i.e. the number latched into the output register. Equating this change to the voltage at the beginning of this part of the cycle, gives:

$$V_{ANA} = 5 \cdot \frac{n}{N}$$

volts. Thus n is a digital representation of the original analog voltage.

(3) The system then counts with switches SW1 and SW2 open until a convenient time to restart the cycle. (SW3 may be open or closed in this time at your option.) Since there are often advantages to equally spaced samples, you might make the overall cycle length of your converter constant.

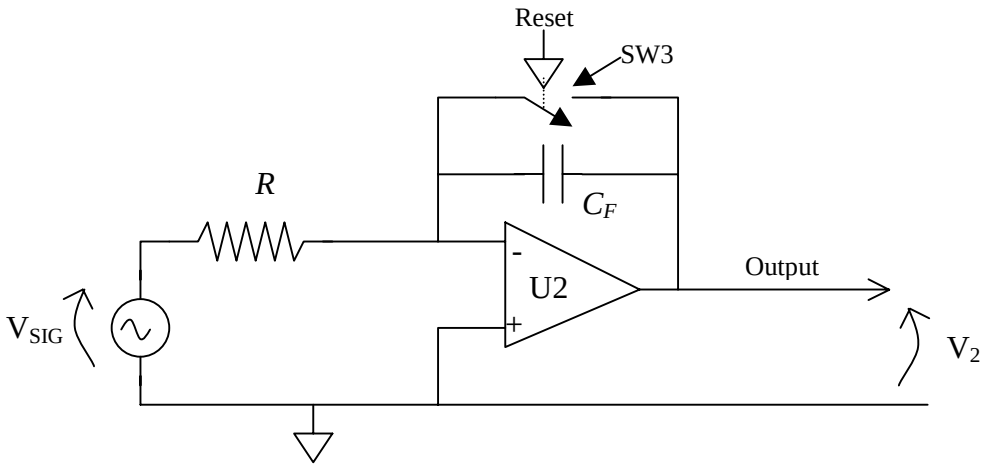


Figure 7-v: Op amp as a resettable integrator.

Aside from the op amps and comparator, the new components being introduced in this lab are the bilateral analog switches. There are four of these in one CMOS package in your kit. Their operation is quite straightforward. Two pins of the package look like an electrical switch in series with about a 250 ohm resistor; the switch can be opened or closed in substantially less than a microsecond by a digital signal applied to a control pin. The switch is closed when this .q "high." To make the device work, one has to connect the VDD pin to +5V, and the VSS pin to GND. The switch terminals can carry analog signals so long as the potential of the signal lies between GND and +5V. The 1K ohm resistor shown in series with SW3 is there to protect SW3 should the integrator output go above 5V or below GND. These devices are easily damaged by inputs out of this range, so please be careful and keep this in mind during debugging.

There are several design choices you must make in this lab. First you need to select a clock frequency, F_c this is done on the basis of needing to update the output a few times a second. An optimum choice would make the period of integration of the signal an integer multiple of $\frac{1}{60}$ second in order to maximize the rejection of 60Hz noise in the analog input signal. (60 Hz is the frequency of the power mains in this country and there is a tendency for noise at that frequency to appear in many systems.) Second, you need to choose values for R and C_F such that with full scale, i.e. +5V, input, the output of the integrator will go up to more than 2 volts but less than 5 volts. This insures that the amplifier will not saturate but that the comparator will have a reasonable signal to measure. Third, you need to extend your 4-bit counter to make a 5-bit counter. Finally you need to devise a logic circuit which will generate the necessary control signals for the conversion. Although this can be designed on an *ad hoc* basis, we believe you would be better off if you were systematic about applying standard finite state machine design techniques to this problem.

The use of the comparator in this lab poses some simple **problems**. The comparator in your kit, the LM311, can easily withstand input voltages up to +18 volts above ground regardless of the voltage on its VCC power line. However, in the negative direction the device can be destroyed instantly by voltages which make an input go negative with respect to its negative power line (pin 4). Since the op-amp requires a -12 volts on its VEE power line (pin 4), we recommend that the comparator be used with -12 volts too. This will reduce the chance of an accidental burnout. Both the

+12 V and -12V power lines should have bypass capacitors (.1 μ*f*d.) connected to ground. One other problem with comparators is that they tend to oscillate, that is their output goes up and down from logic '0' to logic '1' and back, periodically when the inputs are nearly equal. This can be stopped but it is difficult to do, particularly with protoboard construction. It is better to recognize this possibility while designing the logic to control the counter and latch, so that spurious transitions of the comparator output are ignored. **Do not use** the unconditioned output of the comparator to drive the display latch directly.

Summary of Hints:

(1) Be sure C3 will go high at least once each conversion cycle regardless of the comparator output (which may never go positive!). Remember, if you use a CPLD, then the maximum clock cyles per conversion is 66.

(2) Be systematic in designing the logic; use finite state machine techniques.

(3) Use bypass capacitors liberally.

(4) Use -12 volts from the power supply for the V_{EE} pins of both the comparator and the op-amp.

(5) The comparator output may be either '0' or '1' at the instant upwards integration starts.

(6) Do not drive the output latch directly with the comparator output. Spurious transitions on the comparator signal will cause malfunctions.

9.2.9. Lab Eight

Successive Approximation A/D Converter (ADC) System with Sample and Hold

Requirements: Design and build a 4-bit ADC which uses the successive approximation method to search for a digital representation of an analog signal. The system shall include a sample and hold circuit which can stabilize the analog signal (i.e. "hold" the converter input constant) during a conversion. The system shall accept a "start of conversion" (SOC) signal from an external circuit. On the positive going edge of that signal, the analog signal will be held and the conversion started. After no more than 100 microseconds, the conversion will be complete and the result latched into a register. At that time, the sample and hold circuit can return to tracking the analog signal. The converter will be expected to respond properly to SOC pulses with repetition rates up to 9 KHz. The SOC pulse width will lie between 10 and 30 microseconds. Figure 8-i is a block diagram of the system. The circuit should be designed for a maximum analog input voltage range of between 2 and 3 volts peak to peak.

To do this, you need to build a D/A converter (DAC) as shown in Figure 8-ii. As noted on this figure, the operation of the ladder the ladder being terminated in resistances of 2R ohms. The practical implication of this is shown in Figure 8-iii, in which both an output through an op-amp and one directly into a comparator are shown. Your kit contains precision 10K and 20K resistors from which to build your network. With these, you should encounter no problems with your ladder being inaccurate or non-monotonic. There is a more detailed discussion of the ladder in T&S, p. 497.

To test the output of your D/A converter, you should measure the output voltage levels with a voltmeter to verify that the output shifts according to the weighted binary influence. When the DAC is driven by TTL chips, as it is in this lab, the output of the DAC ladder network will range between +.1 volts and +2.0 volts. The outputs of all our test fixtures and of most of the function generators in the lab are symmetric about zero volts, i.e. in your case varies between +1.0 and - 1.0 volts. Therefore, it is necessary to offset the signal from zero by about 1.0 volts, if the DAC is to be able to approximate it. A technique to do this is shown as part of the sample and hold circuit shown in Fig. 8-v; it consists of a capacitor and resistor network between the signal source and the sampling switch. This supplies a fixed offset of about 1.4 volts and eliminates any offset the signal might have.

The operation of counters and comparators can be adversely affected by signals on the power supply connections. Please reread the comments on bypassing in the introduction to this manual and remember that they may apply especially to this lab.

To test your circuit for the TA, you will connect it to one of the ADC test fixtures we will have in the lab. These fixtures consist of a triangle wave generator and a DAC which is connected to an oscilloscope. The triangle generator supplies SOC pulses synchronized to a 2 volt peak to peak triangle wave. If you connect the output of your latch to the DAC and connect the analog input to the ramp signal, the oscilloscope should show a "staircase" approximation to the ramp, with

each "step" corresponding to a digital code from your converter. All sixteen codes (0 through F) must be visible in the correct order for your lab to be checked off. Fig. 8-iv shows an actual picture of such a waveform. In the same test, the TA will also check the output of your sample and hold circuit.

We have only three or four test fixtures for this lab. This means that there will be times when the demand for the use of these fixtures exceeds the supply. Please **test your circuit without the fixture** before you ask for a final test run. In testing your circuit before trying it on one of the test fixtures, you might find it useful to derive SOC pulses by counting down the free running successive approximation register (SAR) clock with another counter chip. Then look to see that all your clock signals are what you designed for, that the analog switch is working, and that the SAR output is reasonable.

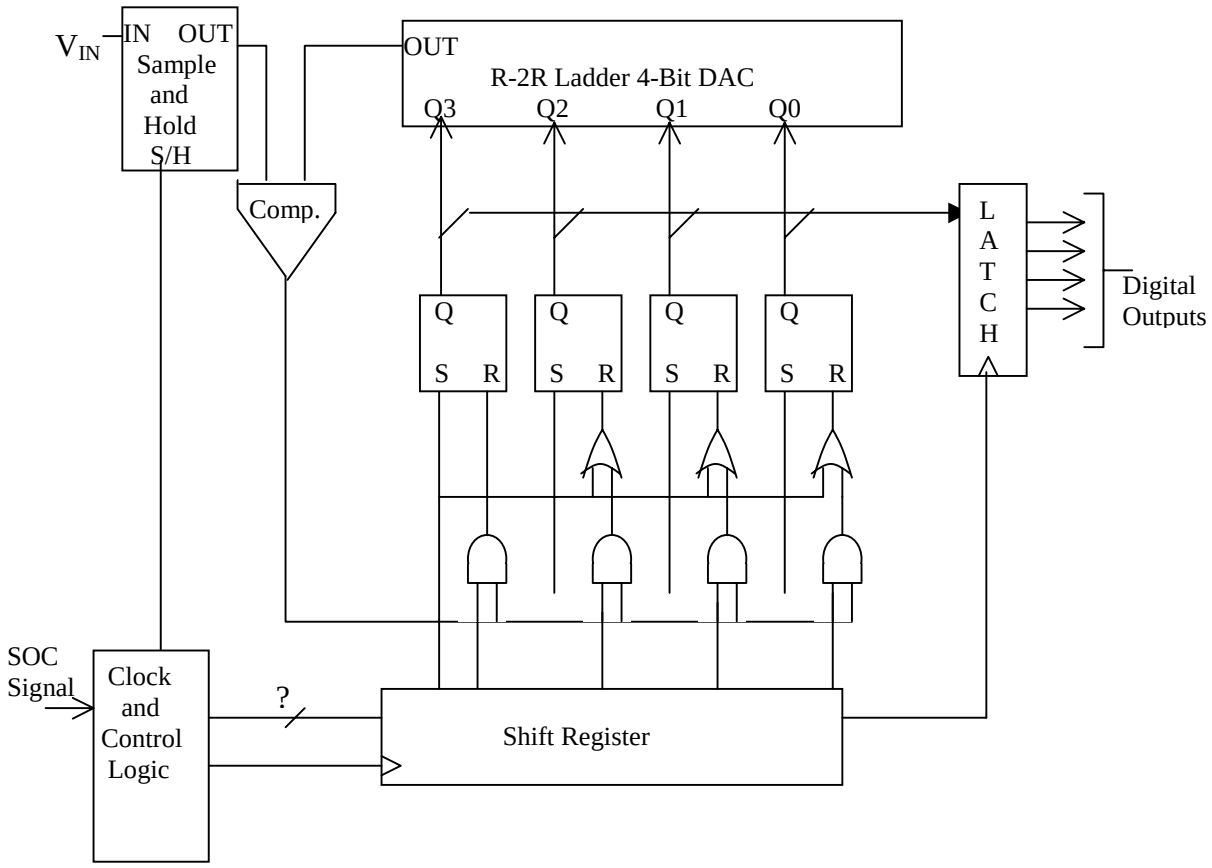


Figure 8-i: Block diagram of a successive approximation ADC system

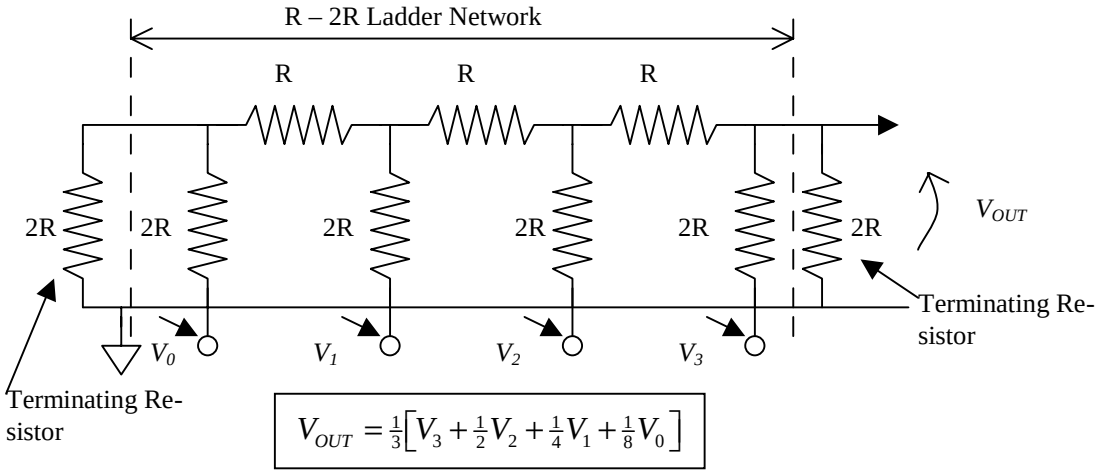
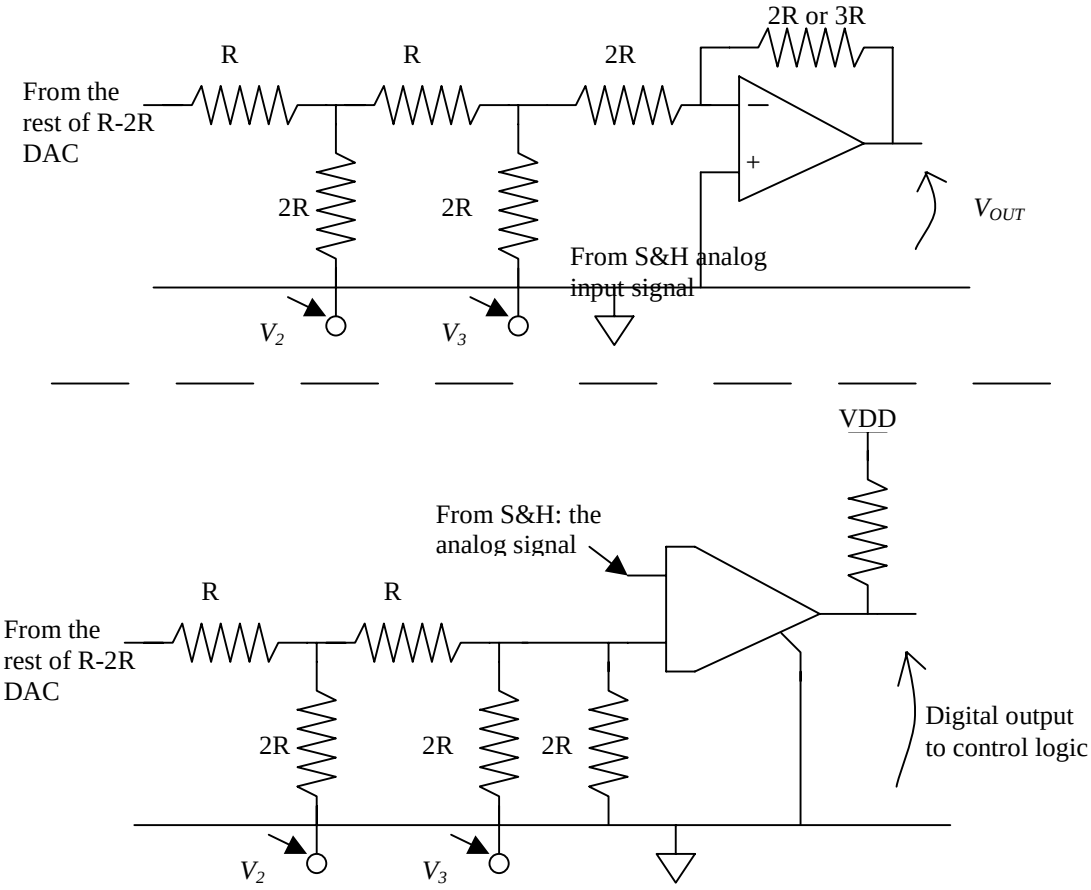


Figure 8-ii: R-2R DAC.

Figure 8-iii:



Connection of R-2R ladder to an op-amp or comparator

WARNING: the one even moderately challenging part of the lab is the "Clock and Control Logic" block. Do NOT use monostable multivibrators (one-shots) to build this. They can never work well in this application. Also you may NOT use the CPLD board for this lab. You have plenty of TTL

parts to do the job, and the wiring is straightforward and not appreciably changed by using a CPLD.

Discussion: Successive approximation A/D converters are widely used for medium speed, moderate accuracy applications. They are generally more expensive and faster than dual slope methods and slower than flash converters. With present design techniques, the conversion time per bit generally lies between .15 and 6 microseconds per bit, depending on the total number of bits. (High accuracy converters must generally allow longer times for their DACs to settle than do units with fewer bits.) Up to about 16 bit converters are readily available; a typical mid-range of converter performance would be a 12 bit unit with 15 microsecond conversion time at a cost of fifteen or so dollars each in moderate quantity. (Even 16 bit converters have come down to modest prices recently.)

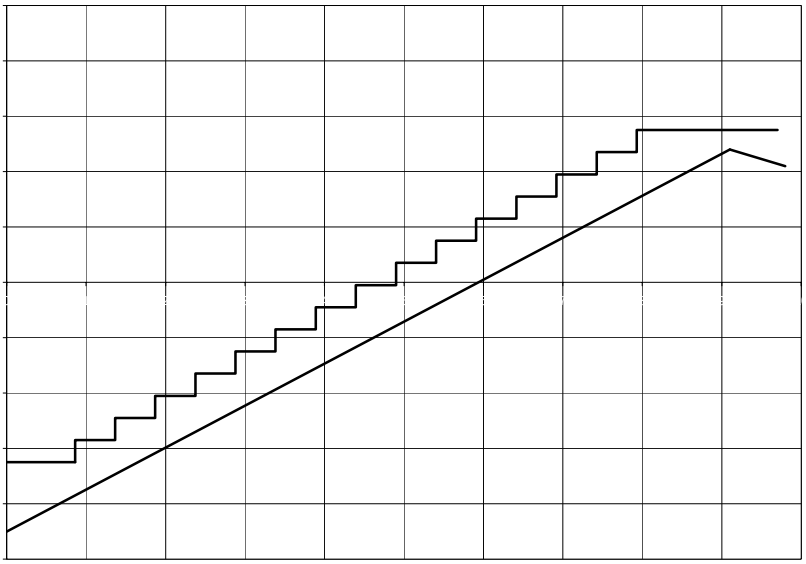


Figure 8-iv: Oscilloscope waveform at the output of test fixture for a ramp input

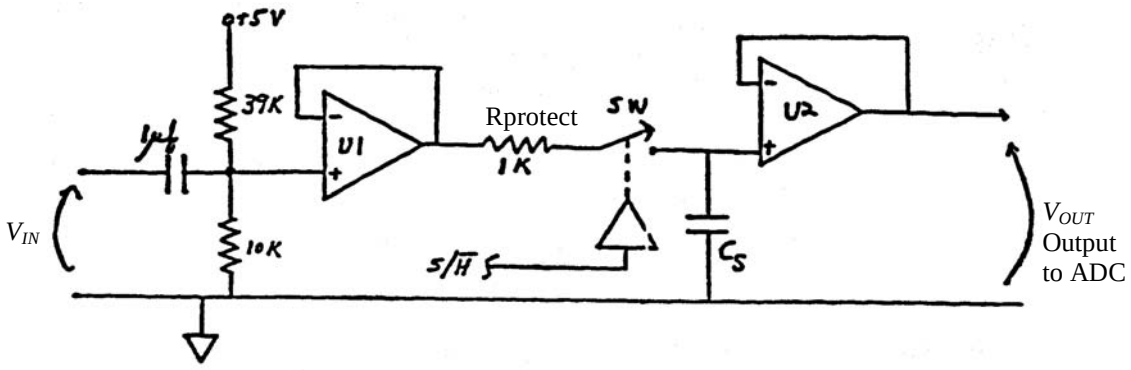


Figure 8-v: Sample and Hold circuit with input offset.

Figure 8-v shows the general form of a multi-purpose sample and hold circuit that can be built from your chip set. The purpose of a sample and hold circuit is to prevent the signal which

the converter is trying to digitize from changing during the time that the converter is operating on it. For the converter to digitize the signal accurately, the signal on which it operates must not change during the conversion by more than one half the amount corresponding to one least significant bit change in the digital output. However, in most situations, the incoming signal changes by a large fraction of the total scale between samples, and the sample separation is only slightly longer than the conversion time.

The sample and hold circuit works by charging a capacitor (C_S in Fig. 8-v) through a switch from the input signal so that as long as the switch is closed, the capacitor voltage is the same as the signal. When the conversion is about to start, the switch opens; since there is now no mechanism by which the charge in the capacitor can change, the voltage across the capacitor remains constant until the end of conversion. The principle is the same as that used in the dynamic RAM storage cell. At the end of conversion the switch closes again so that the capacitor will "track" the incoming signal until the next conversion cycle begins.

The circuit of Fig. 8-v serves a second purpose peculiar to the constraints of this lab. The capacitor (1 μ*f*d.) at its input together with the two resistors and op amp U1 to which the capacitor attaches is used to introduce a DC offset into the incoming signal so that signal seen by the converter has the same range of voltage as the DAC output. The function of the resistor between U1 and the sampling switch is to protect the switch should the potential at the output of U1 exceed the allowed input range of the switch chip. The amplifier U2 is wired to have a gain of 1. Its function is to isolate the hold capacitor from any current drain and hence voltage drift during the hold time due to current drawn by the input of the comparator.

The choice of the sampling capacitor, C_S, is limited by the need for the signal on it to follow the incoming signal with sufficient accuracy when the sampling switch is closed. If the capacitor is too large, its voltage will tend to lag behind the input signal and will tend to vary with time less than the input signal does because not enough current can be passed by the switch to change the capacitor's charge rapidly enough. The switch current is limited by the resistance of the switch itself plus that of the protection resistor. For the circuit to work properly for a four bit ADC with 9 KHz conversion rate, the product of $(R_{protect} + R_{SW}) \cdot C_S$ must be less than 2 microseconds. The value of R_{SW} is about 250 ohm. One needs to select C_S to be as large as possible consistent with this requirement.

The operation of the successive approximation converter itself requires a series of single pulses. The first of these sets the most significant flip-flop in the successive approximation register (SAR) high and all the others low. (The SAR is the set of flip flops connected to the DAC.) The second pulse resets this flip flop if the input signal is less than the DAC output. The third pulse sets the next most significant bit in the SAR, and the process continues through the rest of the bits until all have been set and conditionally reset. After the SAR has settled to its final value, it is necessary to latch that answer into an output register and to return the sample and hold circuit to the sample mode. Although there are several ways to generate the necessary sequence of pulses, we suggest you use a shift register as the basis for a simple scheme.

This system is an example of synchronizing a clocked system to an external asynchronous event, namely the SOC pulse. Any of the standard methods of doing this will work. Again this circuit is likely to be susceptible to power supply noise and will require bypass capacitors.

The practice of connecting the DAC inputs, i.e. the 2R resistors in the DAC network, directly to the SAR flip-flop outputs is not very satisfactory. The output voltages of the gates are not well matched and are not independent of temperature. It is possible to do so in this lab because of the relatively small number of bits to be converted. A small but significant improvement in the circuit performance is possible within the constraints of your Bag-O-Chips by using the fact that CMOS chips generally have output signal voltage levels which are much closer to the power supply and ground levels and which are more reproducible than are the output levels of TTL chips. For this reason, you could use your 74ACT04 inverter chip between the *Q* outputs of the SAR and your DAC network. This would result in more evenly spaced DAC levels and in a slightly wider range of output voltage for the DAC.

Finally, please reread the section of lab 7 which deals with the connection of power to the comparator. You will want to use -12 V on the V- line (VEE) of the comparator (pin 4), and you will need to bypass both power lines to ground.

9.2.10. Lab Nine

Software Simulation of a Logic System:

WARNING: We have installed new versions of all software this fall and they only became functional the first week of classes. Because they were so late coming on-line, I have not had a chance to retest instructions for their use. I will try to do so shortly, but please regard this set of instructions as preliminary and consult the revised handouts on software use before actually doing the lab.

Requirements: Simulate a portion of either Lab 5, 7, 8, A, B, D or F. Labs 5, 7, and A use an XC9572XL CPLD for most of their logic and the simulation of these labs uses only the Aldec *Active-HDL* software to turn a file produced by the ABEL translator in the Xilinx ISE package into a timing diagram. Labs D and F use the Xilinx XCS05XL field programmable gate array and are also simulated in the Aldec package. Lab 8 uses discrete parts and is simulated in the Mentor/Innoveda *ViewSim* tool directly from a schematic produced in *DxDesigner*. Instructions for using the Active-HDL tool are already posted on the class web-site. I have a handout from prior years for the ViewSim methodology and will post that after some slight editing to remove its coverage of PAL simulation. Instructions on simulating the new XCS05XL parts will have to wait until I have a chance to work with those parts. (I expect to have boards working with those parts in early November 2003.

The exact requirements for what to simulate in each possible lab and for the data and time scale to display are given below. To receive credit for this lab, you hand in printed copies of the required timing diagrams and supporting documentation. If your design uses the CPLD, then you should also include your ".abl" file. If it uses the *DxDesigner* and *ViewSim* combination, then supply copies of the schematic and the list of *ViewSim* commands (that is, the ".cmd" file). The schematic must show the labels of the signals appearing in your simulation. If your design uses an underlying VHDL description (lab F or perhaps lab D), then the VHDL files must be included too.

WARNING: We reserve the right to reject an unworkable design even if it is one you have gotten it past a TA for lab credit. Something that works with things tweaked just right is not a properly robust design. Done right, a simulation can show how such a circuit fails.

GENERAL REQUIREMENT: any bus that is part of your simulation must be displayed on your timing diagram with hexadecimal radix. Use sufficient print space so that the hex notation is readable. You may also put the individual bit lines of the bus on the diagram, as you choose. Try to have the order of the signals on the diagram make some sort of sense, *e.g.*, put the clock at the top, then inputs and then outputs in some reasonable order going down the page.

For Lab 5: Treat the clock as a continuous, free-running, square-wave with period of 1.0 microseconds. Make the "Abbreviate" and "Halt" signals arbitrarily specified so that the system executes one complete cycle of the long count from 0 back to 0, then halts for two clock cycles, and finally executes the abbreviated sequence from 0 to 6 and on through 7, 6, and 7. Make the "Halt" signal bounce between clock edges to show it has no effect on the state of the counter. Do this once with the clock HIGH and again with the clock LOW.

As outputs, please show the state bits as a single bus and the segment and decimal point drive lines. Put the clock, Abbreviate, and Halt lines at the top. Prepare two hardcopies: the first should show the sequences matching the requirements of the lab. If necessary for clarity, you may split this printout onto two pages. The second printout is to be on an expanded scale that show the propagation of the clock signal to the change in outputs out of the 0 state such that one can see the delay times.

For Lab 7: Treat the free running clock oscillator signal and the comparator output, **GTZ**, as inputs to your system. If you used the CPLD, simulate that with *Active-HDL*. If not, simulate the control logic, the five bit counter, and the latch with *ViewSim*. (See Figure 7-i.) While it is possible to simulate both the analog and digital sections of the system by coupling Active-HDL or ViewSim to a SPICE simulation, this is too complicated for the time available in this course. Therefore, you should simply treat **GTZ** as an externally specified signal and make it do what one would expect in a real conversion. Simulate two cycles of the A/D converter, one with an analog input corresponding to about one-third full scale and one with an input exceeding full scale. For the run at one-third full scale, make the GTZ signal changes to signal integration back to zero at a down count of 12. Then make the **GTZ** line oscillate or bounce back and forth from 0 to 1 and back a few times between when the reference integrates back to zero and when the conversion cycle begins again. Include at least one time with the **GTZ** line high at a clock edge well after the signal integrates back to zero. This must show that the latched data remains stable. The run simulating an over-range input will have **GTZ** high throughout the down integration and must show the generation of a reset signal at the appropriate time.

As outputs, please show the control signals C1, C2, and C3; the latch clock line; and the five data output lines. If your simulation is of a discrete version of the system, it is sufficient to show the binary counter outputs to the display logic rather than the display segment signals, that is, you don't have to show the display decoder logic. Do include segment signals for a CPLD simulation. Put the two input lines (clock and GTZ) at the top of your timing diagram. Use a free running clock period equal to your design value.

Prepare three hardcopy outputs. Two of these are to be on the most compressed time scale which will still reliably show the free running clock. Both should show the data being latched. (One is for the overrange condition, the other for one-third scale conversion.) The third should be done with a highly expanded time scale to show the time delays during the one-third full-scale conversion between the first edge of the comparator signal and the resultant change of state of the C2, C3, and data signals.

For Lab 8: Treat the free running clock, the comparator output, and the SOC line as inputs to your system. Simulate the clock and control logic; the successive approximation register (SAR); and the data latch. Run the simulation for one complete conversion cycle, starting from a rising edge of SOC. Make the comparator output correspond to the conversion of an input signal which gives a digital output of **1010**.

Put the three inputs at the top of your timing diagram. As outputs, show the sample and hold line, the data latch clock, the shift register serial input, the SAR output bits, and the digital output bits. You may include one or two other control signals if you think they would help explain

the circuit operation and if they are well labeled. Make two timing diagrams. One should show the complete conversion cycle on a suitable time scale. The other should be on an expanded time scale to show the setting and resetting of either the second or fourth SAR bit on a scale that shows the signal propagation times.

For Lab A: Simulate three bus accesses of 8 data transfers each. First, do a WRITE followed by a READ transfer at hex address 0x4010. On the WRITE access, make the data a simple count sequence 0x00, 0x11, 0x22, 0x33, ... 0x88. Repeat both sequences for a base address of 0x8a1b but change the data values. Finally, do the same WRITE sequence to hex address 0x0000, which should show no response at that memory location. You only need to simulate the XC9572XL but you must show all the control signals on the memory chips and the external address counter. Format address and data lines in your timing diagram as hexadecimal busses and print on a scale that we can read their values.

Display and print a separate page for each address range, that is, WRITE and READ accesses can go on the same sheet. Use a clock period of 200 ns. On the plots, show the clock, *BURST*, the PC data bus, the memory data bus, the memory address bus, the memory control signals (*CS*0, *CS*1, *R/WR*, *OE*), and the state bits of your controller. Show all buses as vectors with hexadecimal radix. By hand, annotate the state conditions of your controller to show what part of the access cycle each corresponds to. If any of the data is not readable because the time scale is too long, then print an expanded figure showing just the address phase and first data transfer of the WRITE sequence to 0x4010.

If you needed to handle bi-directional signal lines, you would have to adjust the stimulus to drive the data lines only when there is no contention for them. The “Stimulators” dialog in Active-HDL allows you to specify the “strength” with which you drive a line and the appropriate choice of strength would do the trick. However, you are not simulating the memory this year (I may fix that next year), so the data lines can be driven entirely by the external stimulus. You are only looking at the timing relations with the memory control and address signals.

The real use of simulation in this lab is to get the timing of the control signals correct. You may wish to do the simulation before doing the lab!

For Lab B: Simulate the operation of the CPLD alone – you can set the external signals manually in Active-HDL to mimic the function of the 74195 serial shift register. The simulation must show clearing internal registers, loading the multiplier, and completing the registered product. Multiply “1010” by “0110” and display the 7 segment decoded result.

For Lab D: Simulate writing “1010” to address 0xa100 to match Figure D-iii and follow that by reading back this data. The write address means that the low byte is zero and the high byte (PORTB initial output) is hexadecimal “a1”.

9.3. The Lettered Labs

9.3.1. Lab A:

A RAM Memory Subsystem with Multiplexed Bus

Requirements: We have programmed the PCs in room 340 to simulate an 8-bit bus that uses burst mode data transfers. You are to build a 64 K x 8 memory subsystem to sit on this bus and act as a peripheral memory. (You will only actually use 48K of that range.) Figure A-i shows a block diagram of what you have to build. To simplify the wiring, we have also built small boards that can plug into one of your protoboards and can provide connections to two 32 K x 8 memory chips. This board contains the components inside the box marked with a dashed outline and the label "PREWIRED" on the lower right side of the block diagram. (Figure A-iii is the actual schematic of this memory board and a better copy of this will be in the lab.) The connector pins on the left in this diagram are the lines to the PC. Please use those pin connections to be compatible with the test program. (Remember that ground and power connections are also necessary.)

Notice that an 8-bit bus can certainly not give a 16 bit address while simultaneously handling 8 bits of data. Instead this system borrows a technique widely used on such popular buses as the PCI bus. Figure A-ii shows the timing diagrams for both the read and write operations. Transfers start with the assertion of the negative-true signal called *BURST*. This begins an address phase that lasts two clock cycles. In testing the system, the PC puts the high order address byte on the bus with an adequate setup time before the first *falling* clock edge in the address phase. At this time, the R/WR signal is also valid. It determines the direction of data transfers. [Note: this signal may not remain valid in subsequent clock cycles. If this signal is HIGH during the address phase, then subsequent transfers are READ operations, that is, they transfer data from the SRAM to the PC. Similarly if this signal is LOW during the address phase, then the PC writes to the SRAM.] The PC puts the low order address byte on the bus before the next rising clock edge, but data should not be latched until the falling edge. (All data transfers are relative to the falling edge of the clock because of the peculiar way our PC interface works.) The second falling clock edge of the address phase ends that phase of memory accesses.

On subsequent cycles of the clock, the memory will either read back or write out an 8-bit byte on each falling clock edge. The direction of the transfer is determined by the value of the R/WR line during the address phase. The address phase of the transfer is the two clock cycles with data labels "H.A." (HIGH_ADDRESS) and "L.A." (LOW_ADDRESS). In these cycles, the address bytes are set up and held to be loaded into the latch or counter on the falling edge. At each falling edge of the data phase, the memory subsystem must increment the physical address without further address data from the PC. However, the address count does not have to cross the counting boundary between the low and high bytes of the address. The implication of this is that transfers may only be in blocks of 1 to 256 bytes. The actual data should be valid from the rising edge of the

clock to the falling edge that terminates a given transfer cycle. This is shown as the "D.V. (Data Valid) parts of the timing diagram. The black block on the *BURST* signal at the left end means that this signal may be **either HIGH or LOW** during the rising edge of the clock. However, it will be set up with adequate margin to the falling edge of the clock in the first cycle of addressing.

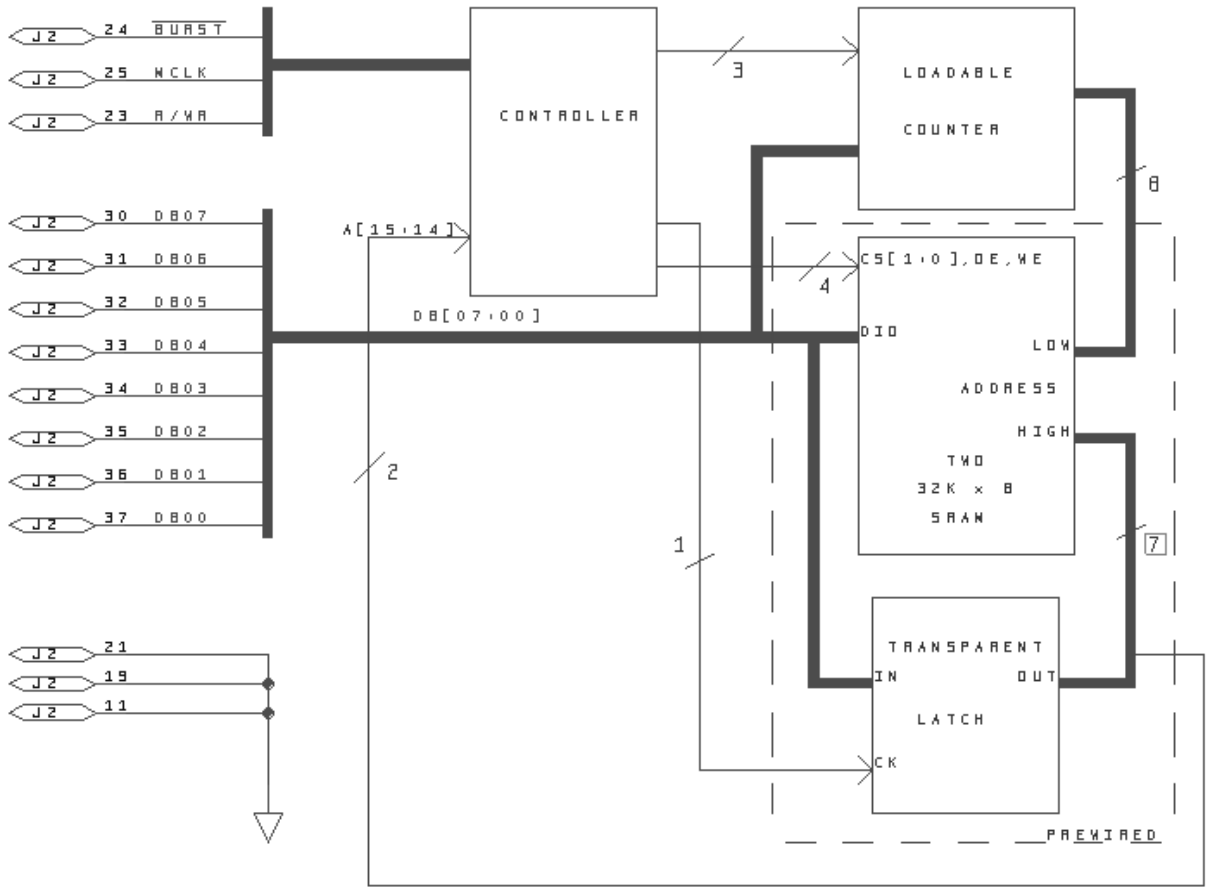


Figure A-i: Block diagram of RAM system with burst mode, bidirectional 8-bit bus.

It is quite uncommon to have a board, card, or other subsystem on a bus that responds to every address the bus can generate. Instead several subsystems may share a bus with each of them performing different tasks. The controlling processor selects the desired activity by the range of addresses it interrogates. Design your memory board so that **it only responds as a memory for addresses** that equal or exceed 0x4000 (16-bit hex address). This would still allow a 48 K memory but would leave 16K of address space for other peripherals.

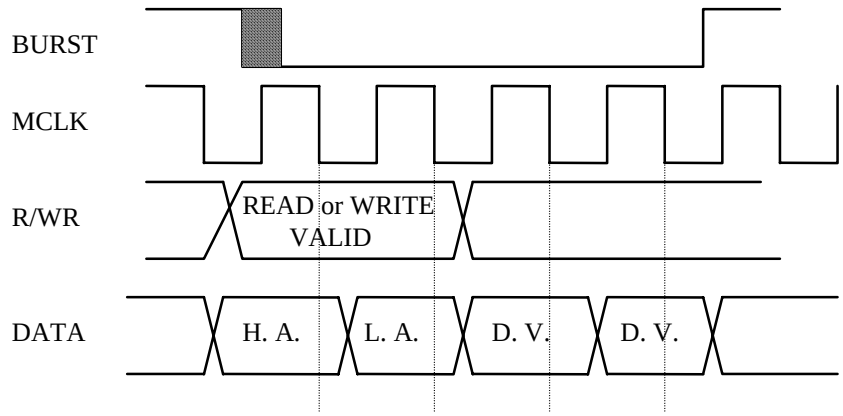


Figure A-ii: READ/WRITE timing diagrams.

Figure A-iii is the schematic of the memory chip connections available on the plug-in boards in the lab. Connector J1 is the ribbon connector to your board. Please note the pin numbers on that connector and remember the need to make power and ground connections to the memory chips as well as to your own circuitry. Also, please note that you must do the address decoding for chip selects. In implementing your circuit, you may use the class CPLD board to implement the controller. You are free to program it using whatever level of software tool you wish, that is you may use the state machine syntax in ABEL to simplify your work. Even with the CPLD board there are not enough pins available to implment the entire controller, so you may have to do some of the address generation with a discrete IC. Any other parts in your kit are also available for implementing the circuit. Please note that the latch incorporated into the memory subsystem is a 74LS373 and is a **transparent** latch. The computerized test sequence is listed and partially explained by the test program as it proceeds.

Notice that the correct time to latch addresses and data is the falling edge of the clock. To improve noise margins and assure reliable clocking, I recommend using a Schmitt-trigger inverter (*e.g.*, a 74LS14) on the clock line and programming the CPLD to respond to the rising edge. (My original design was a discrete logic design and used one inverter chip, two PALs and a 74LS569 that only responded to rising clock edges.) The only tricky part of the design is assuring proper timing of the write pulses on either the *R/WR* or *CSx* lines. (Please note the term “write pulses” in the last sentence. It is not possible to set the memory to write and change addresses while in that state. It will cause spurious write operations.) I have not tested this lab with the CPLD board yet, but I believe it should work fine. I found at least one problem by looking at the simulation and used spare sections of the inverter to improve timing margins. Since I fixed the problem in simulation, I don't know whether the fix was absolutely necessary, but I prefer conservatism. Simulation makes conservative design attractive. Finally, I would point out that this is an ideal lab for using a logic analyzer. There are clips in the lab that let you attach the analyzer probes to any of the pins of a DIP package easily and this makes hookup to the packages on the memory board trivial. The logic analyzer connection on the CPLD board makes most of the connections available too.

Discussion: To do this lab, you check out a protoboard with ribbon connectors for the two cables (PC and memory board) already plugged into them. The ground connections are also jumpered together and there is an RC filter network on the clock line. We make these boards available to simplify the problem of connecting your circuits to the computer and to the memory board. You must make power and ground connections from this connector board to your circuit yourself. One connector has 26 pins and receives the cable from the memory board. The other has 40 pins and is for the cable from the computer. Although the connector types are similar, the pin numbering is different. Please try to build and test your system quickly and return the connector boards since we don't have many of them

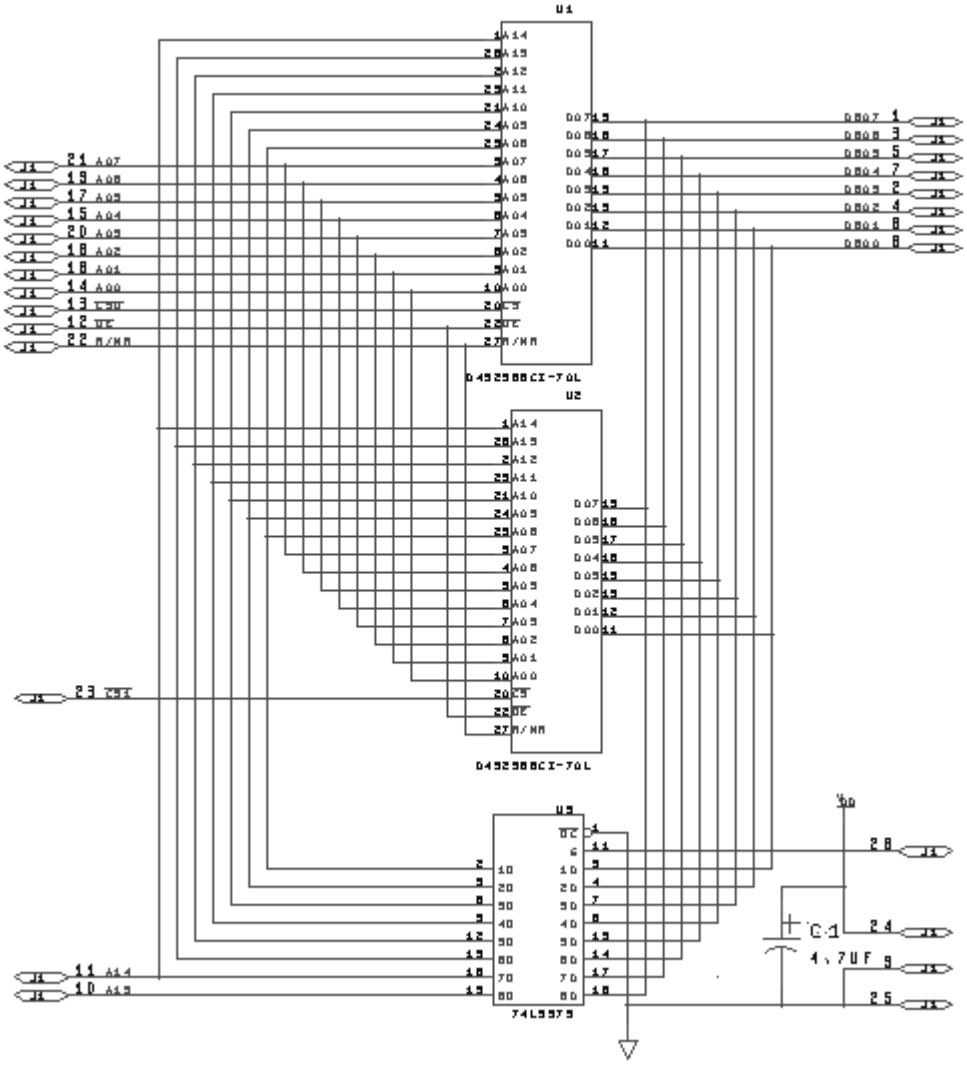


Figure A-iii: Schematic of the SRAM assembly (prewired).

In addition to the connector board, we will have 9-pin flat ribbon cables available with pins in one end and a female connector on the other. You can use this to connect the pins on the CPLD board that normally serve the 7-segment display to your protoboard. With those connections you can do all address counting inside the XC9572XL CPLD. Please be careful to keep these cable as-

semblies together and do not get them mixed up with the JTAG cables we use to program the CPLDs.

We have used ribbon cable to interconnect the memory circuit, the computer, and your circuitry. This is convenient, but it may cause glitches from cross talk between wires. To lower that risk, we have put an RC circuit to filter the clock line from the computer and another to filter the clock line to the latch on the memory board. Please do not move or modify the RC networks on the connector boards. If you have some nearly correct but irregular behavior in your completed system, please talk to me as I may have a suggestion as to the cause of the problem.

You will probably find life is simpler if you attach the connector board to your kit boards in such a way that the two ribbon cables will not go over your circuitry. This will make it easier to use the logic analyzer in case there is any problem. Figure A-iv shows a top view of the physical layout of the two connectors. Please note that the numbering schemes on the two connectors are completely different. (The 40 pin connector is an adapter from the 37-pin D-shell on the computer and has a non-standard numbering. The labels "NC" mean those pins are unused and do not count in the numbering on the schematic. The schematic numbering of the 40 pin connector in the lab manual is that of the computer connector.)

A "bus" is nothing more than a set of wires used for parallel transfer of multi-bit data. It usually has one or more control lines associated with it to signal when data is valid and sometimes to select the source of the data. Buses are used extensively in microprocessor systems to interconnect the CPU with memory and peripherals, in multiprocessor computers to effect interprocessor communications, and in minicomputers and enterprise servers to connect I/O channels with the CPU, etc. They vary in width (i.e. number of wires) from typically 4 to 128 or more bits.

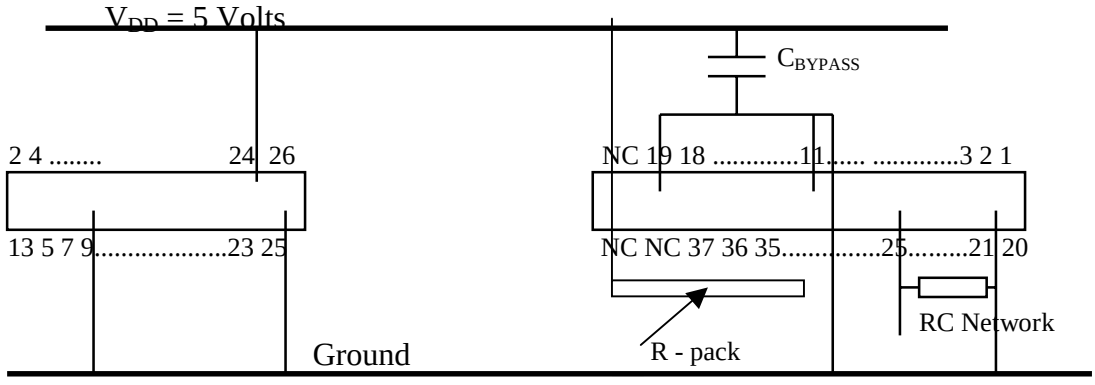


Figure A-iv: Physical pin locations of the PC and SRAM cable connectors

There are a number of terms commonly associated with busses. If a bus has one or more elements connected to it that can act as both a source for and receiver of data, then the bus is said to be *bidirectional*. If one of the control lines is a clock, and all transfers are linked to clock transi-

tions, then the bus is said to be *synchronous*. The alternative is to have one or more "handshake" lines, which acknowledge the presence of data at the receiving element; the transfer rate is determined by signal propagation time and by receiver response time. In this case, the transfer, and hence the bus, is said to be *asynchronous*.

In making fast buses, there are a couple of very obvious problems. If the subsystem that controls the bus has to specify a new address for each datum, the process will be slowed by the time it takes to send and receive the address. Also, having separate pins for the address requires more bus wires. This makes the system more expensive and because of increased physical size it will be even slower. By using burst mode transfers with local address generation, one can eliminate both of these problems. This is the motivation behind the PCI bus design that has become the standard for much of the PC and Macintosh world. Your bus differs from the PCI primarily in being smaller and slower. (The PCI bus may be either 32 or 64 bits wide and may run up to 66 MWps.) The PCI bus also has a special set of configuration ID select lines, one for each card slot. During a "configuration" phase at power-up, the PC uses these lines to interrogate each card in turn. In this way, it can determine what boards are plugged in to its bus, what size block of addresses each requires, and can program their base addresses to be compatible with each other. There are more control lines to allow variable speeds of transfer and other features. However, the basic idea of the burst-mode, multiplexed, synchronous, bi-directional bus is the same as for this lab challenge. There are also several contenders for new memory buses, including RAMBUS, that exploit the same basic idea.

9.3.2. Lab B:

4-Bit by 4-Bit Positive Hexadecimal Multiplier

Requirements: Design and build a circuit that multiplies two 4-bit numbers together and displays the eight-bit result. Consider the two inputs as positive hexadecimal numbers, *not 2's complement.* One of the input numbers must be entered via your keyboard, the other should be entered with one of your DIP switches. Use pull-up resistors and the switches to pull down. The keyboard may be used as the source of an undecoded, four bit number, i.e. four buttons can be pressed in binary combinations to make one input. However, as a challenge to the bored, you should know that it is possible to decode the keyboard fully as well as do the multiply within the constraints of your chip set. The computation (not including the loading of one of the numbers) must be done in less than ten clock pulses.

$P_{3:0}$ $Q_{3:0}$																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1														
2	0	2	4													
3	0	3	6	9												
4	0	4	8	C	10											
5	0	5	A	F	14	19										
6	0	6	C	12	18	1E	24									
7	0	7	E	15	1C	23	2A	31								
8	0	8	10	18	20	28	30	38	40							
9	0	9	12	1B	24	2D	36	3F	48	51						
A	0	A	14	1E	28	32	3C	46	50	5A	64					
B	0	B	16	21	2C	37	42	4D	58	63	6E	79				
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90			
D	0	D	1A	27	34	41	4E	5B	68	75	82	8E	9C	A9		
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Figure B-i: 4-Bit Hex Multiplication Table

If you use a programming language to prepare the jed file for your circuit, you may not use any of the arithmetic constructs of the language. For example, you may not use the vector notation of ABEL (such things as set definitions of the form INPUT = [d,c,b,a] or values INPUT1 = [0,0,0,1]). The most obvious example is you may not use the "+" operator as addition. (If you use the default operator set, then "+" will do arithmetic addition on two multi-bit operands as if they were unsigned binary numbers, *e.g.,* OUTPUT = INPUT + [0,0,1,0]. The point of this lab is to get

you to think about the structure of arithmetic hardware, not to learn the syntax of comprehensive languages.

If you use an "add-and-shift" scheme, then the multiplier has to be converted to a serial bit stream. You must do that with an external shift register, so that only one pin carries that operand into the multiplier system. You may use another pin to load that chip. It is immaterial as to whether the keypad or dip switch operand is the multiplier.

Your circuit must include a continuous clock so that the answer appears to be displayed continuously as the dip switch or push buttons are changed. You should also build a single-step clock into your circuit so you can troubleshoot by observing intermediate "add-and-shift" results and so you can demonstrate the number of pulses to do the multiply. Such a feature may make the Fault Tolerance Question easier to answer, since you will not have to compute by hand *all* intermediate results for your altered circuit. See Figure B-i for the hex multiplication table that your circuit must realize.

You have two choices as to how to implement the circuit. You may build it with the CPLD board or on the BUXUSP-II board using the Xilinx XCS05X Field Programmable Gate Array (FPGA). If you use the BUXUSP-II, certain extra requirements and opportunities apply:

(1) You may use both the built-in LED display and the single step clock by suitable jumpering. You will have to provide a suitable clock to the display for it to show both digits clearly but that clock may be derived from the board's crystal oscillator if you so wish.

(2) If you wish to decode the keypad completely, you may use up to a total of 5 output lines from the FPGA. If you choose not to decode the keypad, then you may use a second section of DIP switch (with pull-up resistors, of course,) for data entry through the external shift register.

Discussion: Consider an add-and-shift scheme for your design, perhaps an extension of the serial adders discussed in your texts. (Wakerly also discusses the add-and-shift multiplier as an example of a synchronous system design. His example has more detail that may be less immediately applicable.)

The requirements are basically the same for implementing this experiment in the CPLD board as for doing it with the FPGA. I do ask that the serialization of one value be done with separate discrete chips.

9.3.3. Lab C:

Implement a Segment of a Boundary Scan Register in a Xilinx FPGA

WARNING! AVIS! ACHTUNG! I am hoping to make significant changes to lab C this year. Because of changes in computer software, I have some work to do before I can make those changes. If I find the energy to make these changes before you reach this lab, then the handout I will give in class will supercede the data given in this manual. **NOTE: I reserve the right to change this lab up until you do it. I am considering changing the FTQ/testing procedures to include some JTAG programming.**

Requirements: Boundary Scan Technology is a way of including extra circuitry within complex chips to make chip and board testing possible without mechanical probing. IEEE standard 1149.1 defines a particular form of boundary scan that many manufacturers have built into their products. More are doing so as package count and board size decrease making comprehensive testing possible at minimal cost. This standard is well adapted for testing printed circuit board connections and for initiating internal chip tests. The basis of the method is a serial/parallel shift register, called a scan register that is built between the pins of the package and the core of the chip where the chip's main logic resides. A finite state machine controller regulates the function of the scan register. Four extra pins initiate and control the testing -- serial input (TDI), serial output (TDO), test mode select (TMS) and test clock (TCK). You are to build a simplified version of such a system into a Xilinx FPGA. (Ironically, the chip you are building on, the XCS05XL, already has a JTAG interface. You duplicate some of its functions.) Your system will have two stages in its scan register, a controller consisting of an instruction shift register and a decoder, and full multiplexing of serial data including bypass of serial input during normal (non-testing) operation.

Figure C-i shows the general arrangement of your test logic. The pins shown are the I/O pads of the XCS05XL. The “Application Logic” in Figure C-i refers to the (unused) Configurable Logic Blocks (CLBs) at the center of the chip that are free for implementing a user's design. Since you are only building test logic (no “Application Logic”), route the input signal back through the output stage. Connect the control signals for the two stages in parallel and bring them to the control logic as shown. The serial input (TDI) connects to a bypass shift register, to the instruction register, and to the daisy-chained scan register; an output multiplexer feeds the outputs of these registers to the TDO pin. Since we will use a PC to test your circuit, use the pin allocations of Fig. C-i as standard connections.

Since the design makes extensive use of flip-flops in shift registers, it is very important that all flip-flops be clocked simultaneously. Any skew between clock signals will result in bad data. All FPGA devices have special nets that distribute clock signals so that skew is controlled. In the SpartanXL family, one sets up a clock network with any of eight special drivers called “BUFGPS” devices in the schematic library. (See also Section 11.4.) Please note the position of that driver on the block diagram and include it in your system.

Fig. C-ii shows the functional block diagram of a single stage of the scan register. The scan blocks are built from CLBs on the periphery of the CLB section. Each stage of this register can do four basic functions depending on its control lines. Data signals can pass through the block, be captured in a flip-flop, or be replaced by a bit from another flip-flop. In addition, data can be shifted into or out of the scan register flip-flops serially through the SIN and SOUT pins.

To make the scan register do useful operations, the test engineer sends instructions into an instruction register that is just a three-bit shift register enabled by the TMS line and clocked by TCK. Instructions enter serially from the TDI pin with ***the least significant instruction bit coming first.*** The outputs of the instruction register are decoded and applied to the scan register control lines. (This control structure is the main simplification for this lab. Real controllers use a standard, 16-state state machine that has considerably more flexibility in the commands it can execute. The IEEE standard is expressed in terms of a state transition diagram.) Table C-I shows the truth table for designing the Instruction Decode Logic. Each line corresponds to a standard test operation. The bit pattern we will use for trying that operation on the PC test system is on the left of the table. The control signals are on the right. Where ENABLE TCK is shown, it means that the test clock signal TCK is active for that flip-flop.

To gain credit for the lab, you must demonstrate to a TA that your circuit passes the automatic test of function. The test software lists any operation that your circuit fails. **You must also show the TA,** using the FPGA Editor in the Xilinx Accessories where the CLBs (the Xilinx Configurable Logic Blocks) that implement your boundary scan cells are on the periphery of the core. In place of an FTQ, I eventually hope to have you program your interface to determine the integrity of an external connection to the FPGA.

Discussion: One of the persistent problems of digital system manufacture is how to determine whether a system is working completely or not. It is often essentially impossible to force a machine to go through all possible states by presenting it with input patterns because there are so many possible conditions. It might take a fast tester more than the nominal lifetime of the product to get through them all. (The classic case showing the consequence of failing to test all possibilities happened to Intel a couple of years ago. The first Pentiums had a bug that resulted in reduced precision for floating point operations in certain rare instances. When the problem turned up, Intel's public relations staff even tried a probabilistic analysis to show that most users had only a one in a million chance of being hurt by the bug. The public didn't buy that line! The damage to the company image and the recall costs were incredible -- the official write-off for fixing the problem was 300 million dollars.) Already for many integrated circuits, testing is the most costly part of the manufacturing cycle. Devising solutions to these problems keeps many engineers regularly employed. In fact it seems to be an attractive career for those with a taste for crossword puzzles as somewhat the same skills are called for.

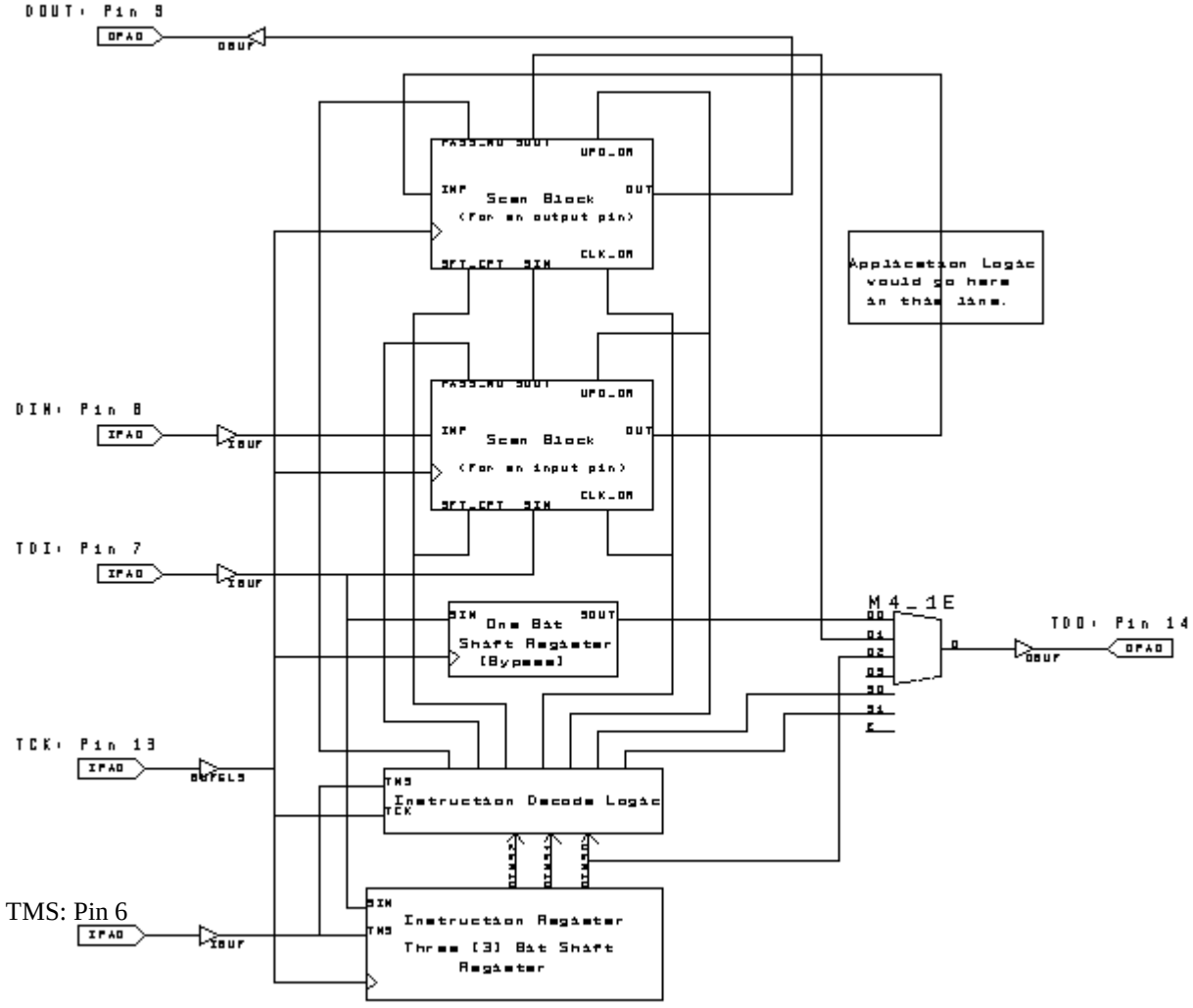


Figure C-i: Simplified Boundary Scan (JTAG)Logic

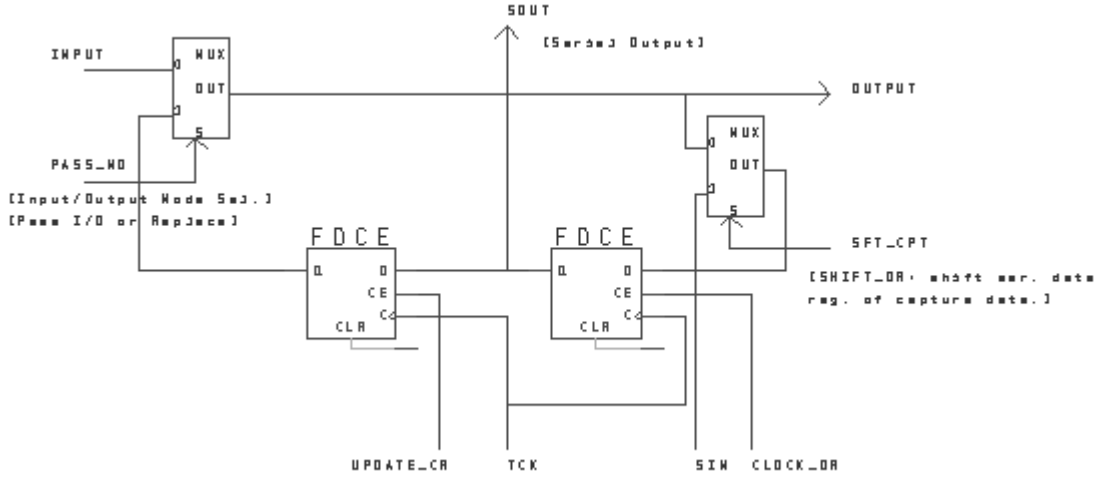


Figure C-ii: Functional Block Diagram of a Scan Stage

TMS	QINS2	QINS1	QINS0	INSTRUCTION	PASS_MD INP. DAT	PASS_MD OUT DAT	SHFT_CPT	CK_DR	UPD_DR	OUTPUT MUX
0	0	0	0	BYPASS: Normal (non-test) chip operation.	0	0	0	0	0	00
0	0	0	1	LOAD_D: Store data from scan register into "update register" for use as stimulus in subsequent test.	0	0	0	0	ENABLE TCK	00
0	0	1	1	INTEST: Substitute stored data for inputs and latch outputs into scan register.	1	0	0	ENABLE TCK	0	00
0	1	1	1	EXTEST: Substitute stored data for outputs and latch inputs into scan register.	0	1	0	ENABLE TCK	0	00
0	0	1	0	SAMPLE: Latch data on pins into scan register.	0	0	0	ENABLE TCK	0	00
0	1	1	0	TSHIFT: Shift scan register contents in/out; otherwise normal pin operation.	0	0	1	ENABLE TCK	0	01
1	X	X	X	LOAD_I: Shift instruction in/out; non-test operation.	0	0	0	0	0	10

Table C-I: Instruction Format and Decode Logic Data

In the last fifteen years, trends in packaging have aggravated the test problem at the circuit board level. The parts in your kit are in DIPs or Dual In-line Packages. These have pins that are 0.1 inches apart and extend all the way through the printed circuit board they are mounted on. Because pins extend through the board, it is not practical to put parts on both sides of the board. Tests on the board can be done easily by using what is called a “bed-of-nails” fixture to make contact to the board. As the name implies, this device has spring-loaded pins (the “nails”) on a frame that matches the placement of pins on the board. These make contact to the bottom side of the board for all integrated circuit pins simultaneously, making both conductivity and signal checks possible. However, boards now make almost exclusive use of surface mount parts. Surface mount devices have pins on 0.0195-inch centers and these solder to only one surface of the board. With through-holes eliminated, designers can put parts on both sides. Now one cannot use a bed of nails fixture since placement tolerances are too tight and access is blocked by other parts anyway.

In the mid-1980's, an international industry group, JTAG (the Joint Test Action Group), proposed a system for standardized implementation of a boundary scan technique to solve the problem of low-cost board testing. This eventually resulted in the IEEE standard 1149.1. Since then the standard has been included in the design of many products of many companies from AMD to Xilinx.

Figure C-iii (from the Abramovici reference given below) shows what is added to an integrated circuit to make it compatible with the boundary scan standard. The dotted area marked "Application Logic" is the actual functional logic of the chip. The other elements are all part of the test logic. The boundary scan cells along the edges are the blocks you are building and testing for this lab. They either pass data through in normal operation or allow the user to capture and replace data entering or leaving the pins of the chip. Replacement data is clocked in serially along the boundary scan path from the TDI (Test Data In) pin. The captured data goes serially to the TDO (Test Data Out) pin along the same path after the test.

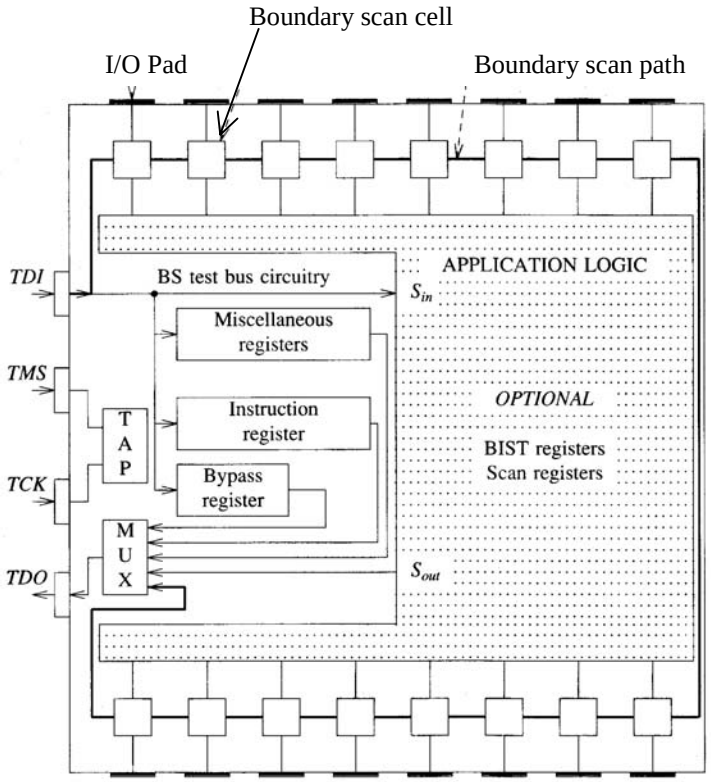


Figure C-iii: Chip architecture for Boundary-Scan implementation

The block marked TAP on the left side of the chip is the key to how the system works. This is the Test Access Port controller, a finite state machine which is clocked by the TCK (Test Clock) line and which has TMS (Test Mode Signal) as its only input. (Abramovici, et al. give the state diagram for this controller as their figure 9-53.) This controller handles clocking commands and data in serially through the TDI pin. The commands set the function of the scan cells and also enables or disables the flip-flops the TCK clock line at the flip flops. By manipulating the TMS and TMI lines as TCK is clocked, the user can do at least the following functions:

(1) Latch all chip input and output data at one moment and read it out serially through TDO (Test name: SAMPLE)

(2) Replace input bits by data previously read in serially through TDI and latch the resultant output data for later readout (Test name: INTEST)

(3) Replace output data with test data previously read in serially through TDI and latch the resultant input data for later readout (Test name: EXTEST)

(4) Pass serial data for testing other chips through a short (1-bit) path to reduce overall test time. At the same time the scan register does not affect input and output signals. (Test name: BYPASS)

(5) Activate any built-in self test that the chip might have. (The ea self-test circuit by which the chip determines its own status is optional but common in chips of a size for which the JTAG overhead is reasonable. Test name: BIST)

The overhead for implementing these tests is obviously fairly modest, as they require only four extra pins per chip. Serial data paths are slow, but they minimize the number of pins and printed-circuit traces.

In this lab, your system will do the BYPASS, SAMPLE, EXTEST, and INTEST tests as well as shifting data in and out of the scan register. However, your circuit uses a simpler method of getting instructions into the system. Instead of the canonical state machine, you use a straightforward shift register to capture an instruction. Like the real versions of the JTAG protocol, yours has to decode the instructions for each test. In your version of the system, EXTEST and INTEST are two-step operations since it is necessary to first load the substitution data register with a LOAD_D instruction.

Notice that your system enables the TCK clock at the flip-flop rather than gating the clock as the JTAG design suggests. Generally I have taught you that gating clock lines is bad practice. It is feasible in the standard JTAG design because the gated clock line can be distributed without skew if carefully designed and because the state of the TCK is always low when the TMS line changes value. This constraint guarantees no spurious clocking. You do not have the option of controlling skew on a clock line routed as a normal signal. For this reason I have suggested routing enable lines and using a global clock line for this function. I believe it makes the system more robust.

Figure C-iv shows diagrammatically how the scheme works on a board with two chips configured for boundary scan testing. At one edge of the board, there is a test connector where the user can have access to the test pins during board testing. Some form of automated test equipment (ATE) would supply these signals during manufacturing checkout and any other time one wished to do a system evaluation. The TCK line is common to both chips and originates at the test edge connector. The two TMS lines go separately to the tester to maximize the flexibility of the system. The two serial paths through the chips connect serially between each other, and the chain begins and ends at the test connector.

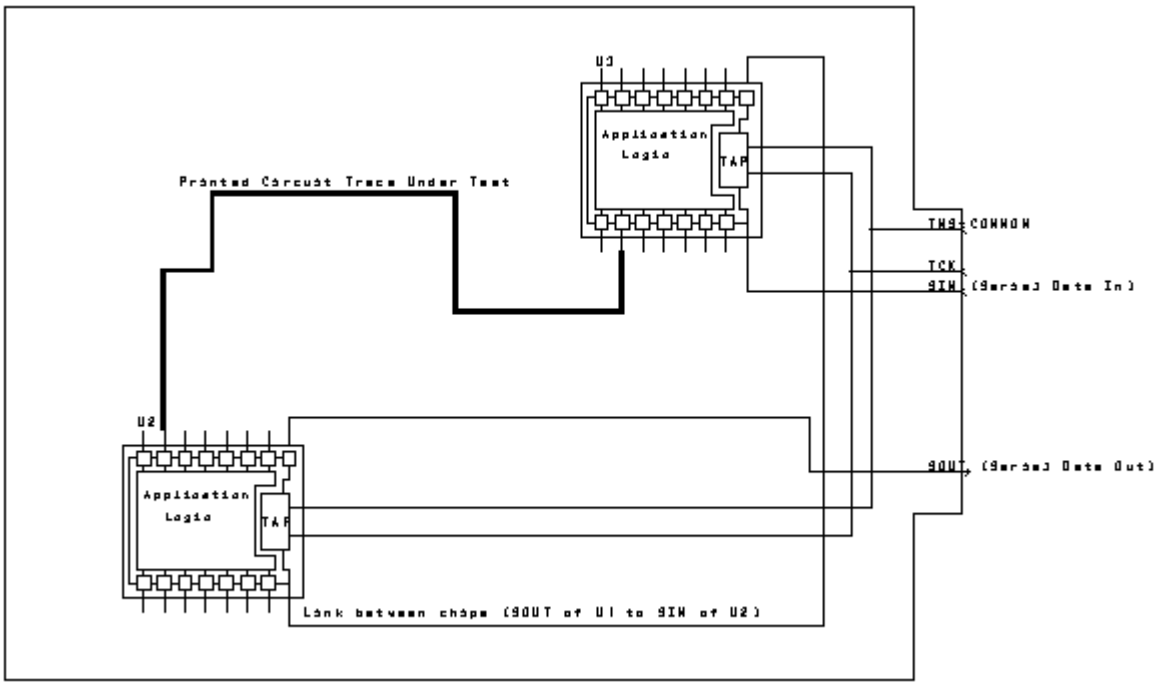


Figure C-iv: Arrangement of Boundary-Scan testable chips on a PC board

As a simple application of the test capability, consider the printed circuit trace marked .q "PC-Trace Under Test" in Fig. C-iv. We wish to determine if that line has continuity or not. (It might not, for example, if the solder on one of the IC pins were defective.) To test it, we need to apply a signal at one end and see if it is received at the other. To be sure of the situation, we would probably make two tests, one with logic HIGH and one with LOW. Fig. C-v shows how the scan register cells at U1 and U2 are configured for the test. (The configuration of the scan cells was set by an EXTEST instruction clocked into the TAP controller before the test begin.) First, one clocks a test bit serially into the QA flip-flop of U1 using a LOAD_D instruction. Then both chips receive an EXTEST instruction and TCK is clocked to latch the data. Finally, the captured data is clocked out with another LOAD_D instruction. (One advantage of the more complicated JTAG TAP controller is that the explicit LOAD_D instruction can be avoided.) A much more extensive discussion of the characteristics and use of the technique are given in the references below.

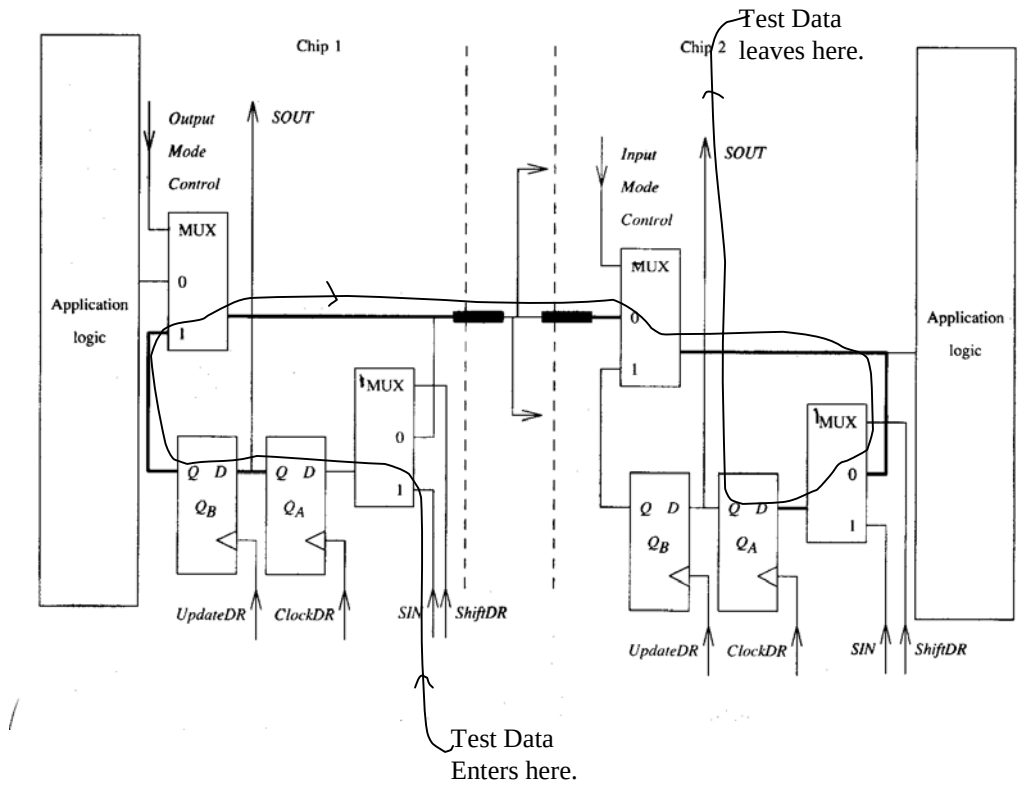


Figure C-v: Scan cell configurations for testing a PC trace

Designs in a Xilinx programmable gate array must ultimately be implemented in Configurable Logic Blocks (CLBs), Input-Output Blocks (IOBs), and interconnects. The specification of these elements is greatly simplified by reducing the process to drawing a *DxDesigner* schematic, placing symbols for the CLBs and IOBs and wiring them with nets in the usual fashion. Routing software from Xilinx assigns the symbols to particular locations on the array and does the wiring automatically from the *DxDesigner* wire-list. (Xilinx is as much a software company as a chip company. Although their sales are over $ 1.4 billion per year, they are fab-less and actually farm-out all their chip manufacture to foundries.)

To simplify the process even further, Xilinx supplies specialized symbols for many common digital functions that relieve the user of having to deal with the details of setting up a CLB or a simple input or output. For the XCS05XL we use in this lab, these are found in the Spartan XL library. We have also made a small library for you called en0163/xilinx. In it you will find two symbols that may be useful for this application. Each consists of 2→ 1MUX coupled to a D flip-flop. The distinction is in the interconnection. This relieves you of the burden of having to learn the way to construct symbols or to configure CLBs. **Note: you still need to understand the functionality of a CLB and what needed to specify its operation. A potential exam question would be the conversion of a block diagram into 1 - 3 CLBs, given the Xilinx configuration data tables.**

Here are a handful of observations and reminders about the problem that you need to keep in mind.

(1) Scan blocks that support input pins are identical to those that support output pins. The only differences are the direction of signal flow and the source of each block's mode control signal. (The PASS_MD connection to each scan block determines whether its input is replaced by a data bit from the test flip-flop. In general the PASS_MD signals for input and outputs are separate but are shared between all input and output blocks respectively.) More complicated blocks are needed to support bidirectional or tri-state pins.

(2) Input-Output pin assignment is handled by the "LOC" attribute assigned to each pad. See the section of this manual on **Hints for Using Workvew Office and Xilinx Alliance Software** for details.

(3) There is no reference designator for Xilinx macros -- the blocks that you seem to be placing on the schematic -- since they are not actual components.

The Xilinx Corporation has made all its parts since the XC4000 series of FPGAs including your XCS05XL devices compatible with the 1149.1 standard. There is a full boundary scan register and controller built into each chip in addition to the chip's customizable logic. The overhead is modest when the scan circuits are specialized and are not themselves programmable. You are likely to find that trying to put such circuitry into the user-customizable part of an XCS05XL uses up far too much of the resources of the chip to be sensible, and I would agree with you. The basic point of this lab is to make you aware of testing as a general issue and of boundary-scan logic as a particular partial solution to test problems. At the same time it will introduce you to another alternate to TTL as a way of building systems.

While there are many references on testing, there are two I have found particularly readable on the subject of Boundary Scan Technology. The most complete is chapter 9 section 9.10 in the book by Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990. There was also an article in the *IEEE Spectrum* magazine, *Testability on TAP*, by Colin M. Maunder and Rodham E. Tullross, Feb. 1992, pps 34-37. I will place the Abramovici book on reserve at the Sciences Library

9.3.4. Lab D:

DRAM Controller in a Xilinx FPGA

Requirements: Upon request, we will supply you with a Dynamic RAM module with 1 Megaword of 8-bit, byte-wide storage based on Hitachi DRAMs. (The data sheet I have available is from a Motorola product that was pin compatible.) The module is is already mounted on a small printed board with a connector to the BUXUSP-II user interface, a logic analyzer connector, and a protection diode. (For your convenience 4 data bits, 4 address bits, and all the control signals are also wired into a header which plugs directly into the logic analyzer. That makes using the analyzer very easy.) The only other connection is a **cable for 5_volt power.** The schematic is shown below as Figure D-iv. Program the XCS05XL on the BUXUSP-II board to be a controller for the memory, such that it is possible to read and write the bottom 64K of memory from the PC. The Xilinx array refreshes the memory as needed, handles address multiplexing, and generates the control signals /*RAS*, /*CAS*, and *R/WR*. Figure D-i-a shows the kind of overall system that this lab is intended to mimic. Figure D-i-b shows **what you will actually build** in block form.

The computers to which the BUXUSP-II boards are connected have 24 digital input-output pins that are programmed in three groups, PORTA, PORTB, and PORTC. DRAM address lines invariably require multiplexing. In this case the software on the PC will do the actual multiplexing but will rely on the signal ASEL on PORTC5 for determining when to supply the high and low bytes. PORTC provides all the necessary control signals. It is broken into two halves: PORTC[0:3] are write lines originating on the PC that signal the need for data transfer, and PORTC[4:7] are PC read lines for handshake signals. Not all of these are used. The schematic drawing of the BUXUSP-II board, which is the example figure in section 10.1 of this manual, gives the assignment of port signals to pins on the XCS05XL FPGA and on the user-interface cable. The assignment of logical functions to these lines is tabulated below.

PORT C to Xilinx Control Line Assignment				
PORT C Line	C – Pin Type	Xilinx Pin	Xi – Pin Type	Line Function
PORTC0	Out	10	In	AS – Address Strobe
PORTC1	Out	3	In	W/R – Write / Read line
PORTC2	Out	13	In	DCT – Data Cycle Terminate: on READ signals the PC has finished storing the data; on WRITE acknowledges DTACK and terminates cycle.
PORTC4	In	5	Out	/DTACK – Data Acknowledge (Address, read, and write)
PORTC5	In	6	Out	ASEL – Address multiplex request line

Figures D-ii and D-iii are the timing diagrams for the transfer of data between the computer and the DRAM subsystem in READ and WRITE operations respectively. The required timing of the signals to the DRAM chip itself are given in the data sheet for that device. There are five control signals actually used in data transfer. Three of these originate at the PC: *WR/R* is the comple-

ment of the DRAM *R/WR* pin. It is valid only during high address latching and determines the direction of transfer. Two other signals from the PC initiate and terminate actions. The Address Strobe line, *AS*, initiates transfers by indicating when a valid high-order address byte is on the PORTA lines; the Data Cycle Terminate, DCT, line terminates read operations by signaling when the computer has accepted the data from the memory and terminates WRITE operations by acknowledging the controller's DTACK signal. The Address Select signal, ASEL, comes from the controller and signals when the computer may put the low-order address byte onto PORTA. The Data Acknowledge line, *DTACK*, also comes from the controller, and it signals when the controller is done with the address bits (leading edge transition) and is done with either writing data or reading it out (trailing edge). This handshake procedure is similar to several used by different microprocessors, most notably those with memory available on peripheral buses. This particular bus protocol is slower than is usually used in main memory applications where DRAM is most frequently found.

All cycles start when the computer asserts the Address Strobe line, *AS*, high. If any refresh is needed, the controller waits to acknowledge this transition until refresh is finished. Subsequent action depends on whether this is a read or a write operation. A more detailed discussion of the timing diagrams is given below. Obviously the main design problem is how to translate from the set of signals from the PC to those required by the DRAMs.

To gain credit for the lab, you must demonstrate to a TA that your circuit passes the automatic test of function, a special program on the PC that exercises your memory. We have given the test program operating modes in which, regardless of errors, the software will do a 256-byte burst of read or write operations. This can let you capture a cycle easily with the logic analyzer or a scope for debug. (Remember too that the refresh cycles should run continuously and therefore can be checked with the logic analyzer easily.) The software is fully windows-compatible – you just pull down the Run menu to access the various test modes. FTQs are still TBD. (How's that for acronym density?)

Discussion: You may use either schematic capture or VHDL to enter your design. If you choose VHDL, please see notes at the end of Lab F for the way to compile your files.

Most of this problem is doing the timing conversion. The details of the timing cycles are shown in Figs. D-ii and D-iii. The READ cycle operation is:

(1) The computer begins the cycle by setting *AS* high. The address data has sufficient setup time that the controller may use the address data immediately.

(2) If a refresh cycle is in progress, the controller completes it and the computer waits for an acknowledgement in the form of the request for the low-order address byte.

(3) The controller latches the high address byte data in the DRAM by asserting */RAS* low, and then asserts the ASEL line to ask for the low-order address.

(4) After the computer responds by putting out the low-byte and removing the AS signal, the controller asks the DRAM for data. Shortly thereafter, the controller asserts the *DTACK* line low to show that READ data is available. This signals the computer to accept and store the data.

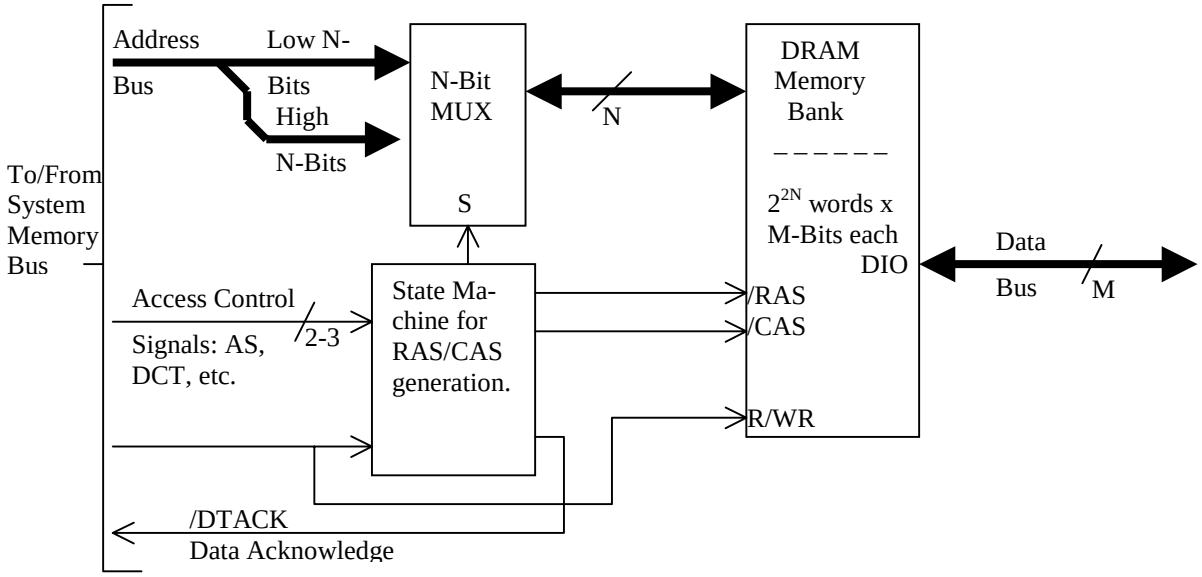


Figure D-i-a: Block Diagram of a General DRAM Memory System

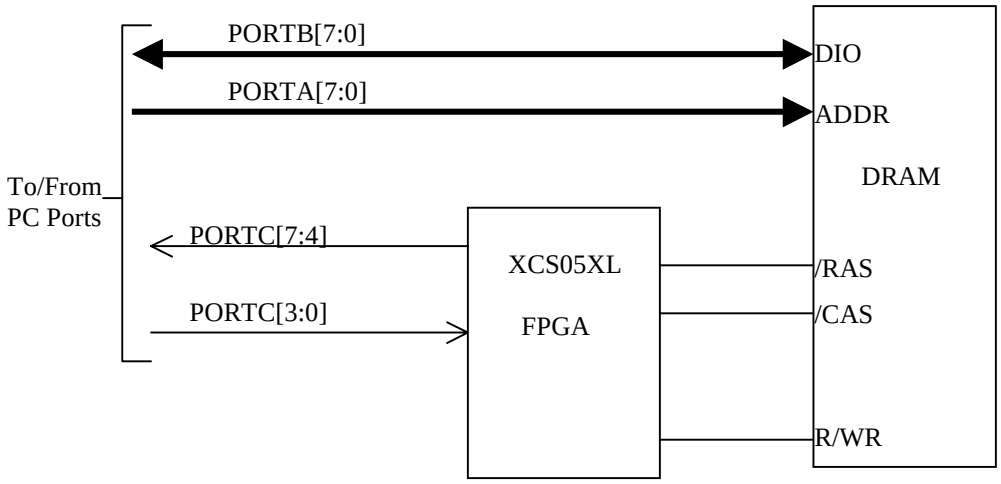


Figure D-i-b: Block Diagram of Your Memory System

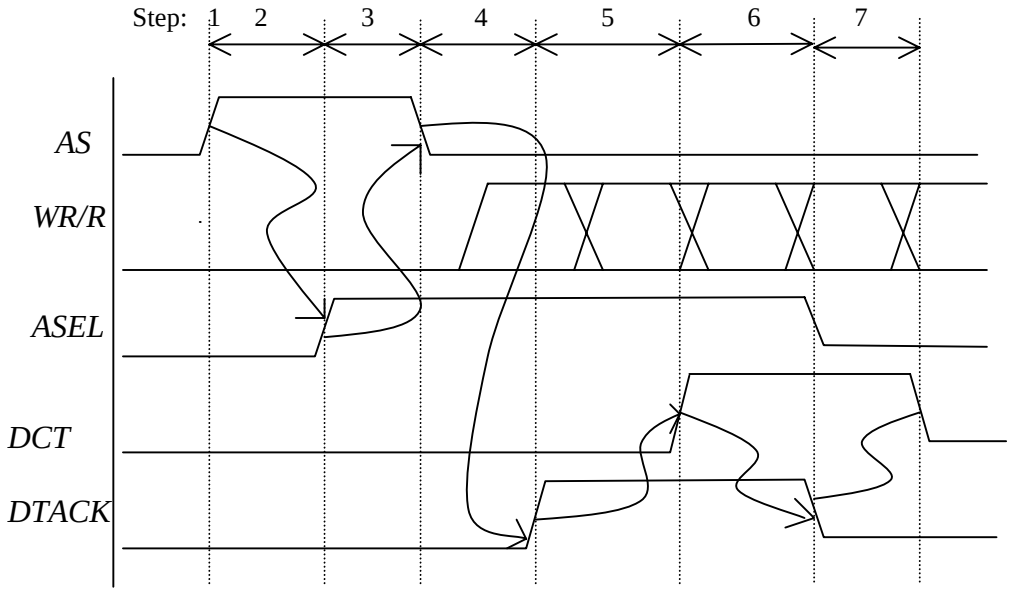


Figure D-ii: Timing for READ Operation

(5) The computer responds by reading the data through PORTB, and then setting the *DCT* high.

(6) When the controller receives the DCT signal, it turns off the DRAM and deasserts its DTACK line.

(7) Finally the PC removes its DCT signal terminating the access cycle.

The write operation is very similar. The addressing sequence (steps 1 – 3) is the same. Valid WRITE data is present on PORTB before the AS signal goes low. The controller writes the data to the DRAM and then signals with DTACK that it is done. The PC responds with DCT and the cycle terminates with removal of DTACK and DCT one after the other. The only major difference between READ and WRITE is the transfer direction determined by the value of *WR/R* at the time *AS* goes high. Probably there will also be a slight difference in the times between edges. **NOTE: the W/R is only guaranteed correct while AS is high. It therefore has to be latched to be used while CAS is asserted. It is also of opposite polarity to the convention for the DRAMs themselves.**

I suggest that a Moore finite state machine is an appropriate method of design. It is probably often not the optimum method for a DRAM controller when there is no easy way to synchronize the state machine clock with the data requests. In this lab our primary objective is to get you to think about timing considerations and about DRAM characteristics. Secondarily you are learning about FPGAs. We do not really care that the total transfer time will be very long. For simplic-

ity, just use the external clock. This gives a clock cycle just a bit longer than 0.5μsec, a very easy-to-live with rate.

A couple of design hints to make life easier:

(1) Use the "CAS-before-RAS" refresh mode. This eliminates the need for a refresh address counter. You still need a counter for clock cycles between refreshes. Thirty two clock cycles is a reasonable maximum number between refreshes. (That would mean that the 512 required refreshes will take place in about 20 ms.)

(2) There are at least three signals coming into the controller that determine its transitions between states. Because the logic functions of these FPGAs are limited to five inputs and the ratio of flip-flops to logic blocks is 2:1, the finite state machine might be implemented as a "one-hot" design. The total number of required states looks favorable for this style to me, but, of course, the choice is up to you.

(3) Remember that one really has to be careful of glitches on the $\overline{RAS}$ and $\overline{CAS}$ lines. FPGA logic often has long, unexpected (if you have not simulated) differential delays in logic that can result in hazards and glitches if you are not systematic in preventing them..

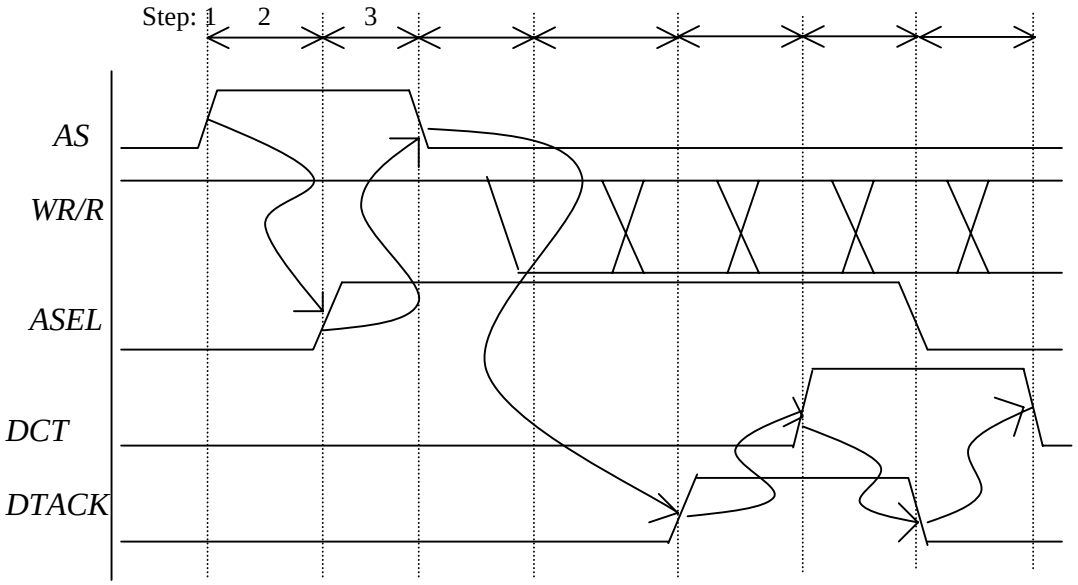


Figure D-iii: Timing for WRITE Operation

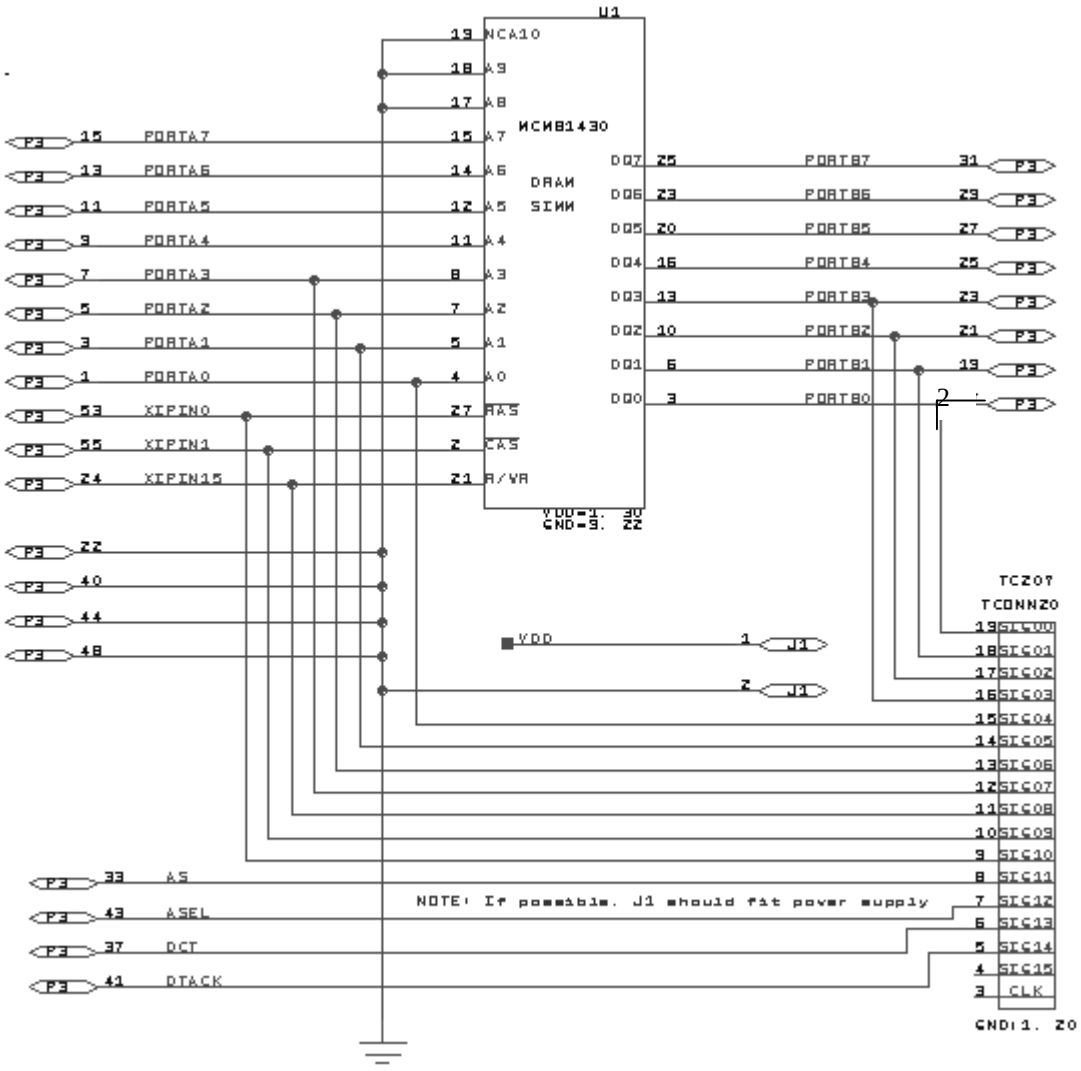


Figure D-iv: Schematic diagram of the DRAM module for Lab D. Note the labeling of pins on the analyzer connection showing their designation as analyzer channels.

9.3.5. Lab E:

Music Box/Greeting Card Sound from a Xilinx FPGA

Requirements: Upon request, we will supply you with a small loudspeaker (complete with amplifier). We will also make available 27C64 PROMs with music frequency data in one of two formats. Connect and program a Xilinx XCS05XL so that it will use the PROM data to play music into the speaker. To reduce the amount of work you have to do and to improve the chances of the PROM surviving, we have put the PROMs on a pre-wired breadboard so that you do not have to wire the actual PROM connections. There will be a schematic available of the pin assignments. You still have to worry about the speaker hookup and the beat oscillator. Use the pot in your kit to make the beat of the music adjustable from funereal to way too fast. Figure E-i shows in block form how the system is supposed to work. The "beat oscillator", which sets the rate at which notes are played, may be either inside or outside the XCS05XL at your option. The frequency synthesis may be done either by counting down to zero or by an adder/accumulator method. The two methods are shown in block form in Fig. E-ii and discussed at greater length below. The test of whether the circuit works is whether the tune is recognizable!

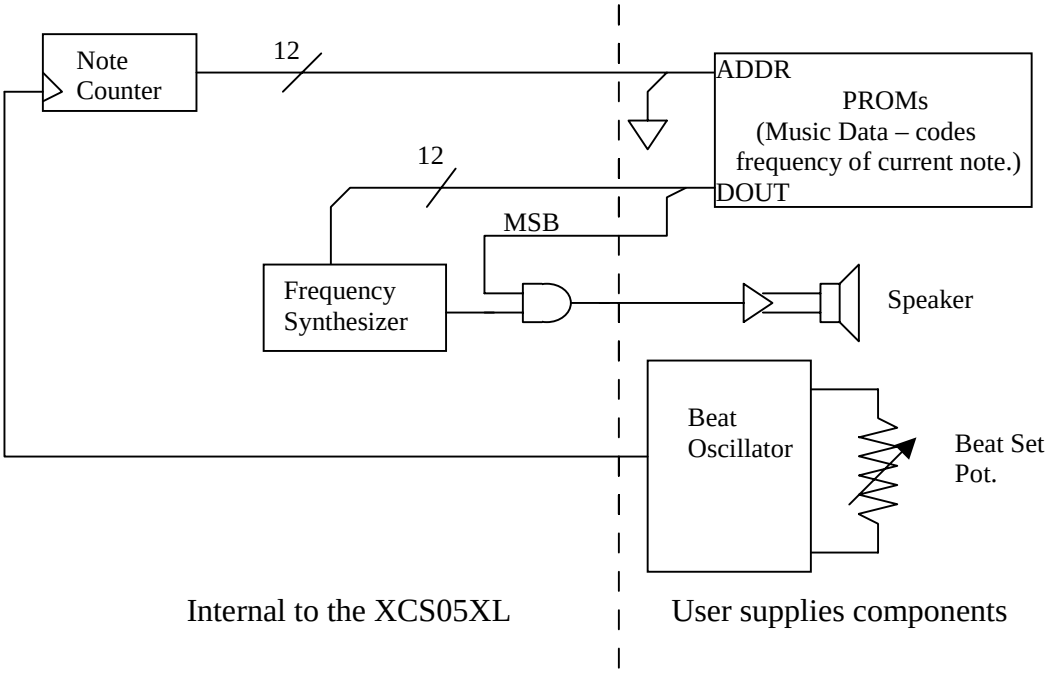


Figure E-i: Block Diagram of Music Box System.

Discussion: Most of you have probably seen and heard one of the musical greeting cards that are especially popular at Christmas. You open one, and it starts to play a tinny version of *Jingle Bells* or *O Tannenbaum*. These are really a remarkable *tour de force* of manufacturing. For a manufacturing cost that has to be well under one dollar, each card includes an on/off switch, a battery, a

programmable oscillator with enough ROM for the tune, and a speaker (actually a transducer). The whole system has to be wired and shrunk to fit inside the card. Despite its cost, the battery seems to last for an hour or so of playing. In this lab, we ask you to imagine being in the early stages of designing a new product of this kind, say a music box that is to have slightly better sound quality than a greeting card. One of the first questions you want to ask is what minimum algorithm will get adequate sound quality. At this stage, the cost minimization is not critical (that comes later). As is usual, development time is important, so the use of discrete PROMs and FPGAs for exploration seems justified. To get the hang of things, you will start off with a primitive scheme that produces a gated square wave.

The music produced by your system is very simple. It is a single line of notes without much subtlety of intonation or timbre. All one has to do is generate a signal with the appropriate variation of frequency and with breaks for rests and for the start of each note. The function of the PROM is to contain the sequence and duration of the notes. The twelve least significant bits of the data specify the frequency directly as the bit pattern needed to run a frequency synthesizer. There is no decoding necessary such as one might use to reduce the required amount of memory. The most significant PROM bit is used to gate the output and so specifies rests and breaks. The duration of notes is determined by how many times the frequency data is repeated in the PROM. Thus the PROM address is simply incremented at a constant rate proportional to the beat of the music. Sixteenth notes in the music are repeated four times in the PROM, so a quarter note is 16 PROM locations. Music is normally played at between 50 and 150 quarter notes per minute, which sets the required range of frequency of the beat oscillator of Fig. E-i. (As a deviation to the policy of no non-kit parts, we will make available some extra capacitors for this timing function. You will have a range to choose from.) The total capacity of the PROM is more data than I had the patience to enter. Therefore, please tie the most significant bit of the PROM address low, cutting the memory size in half.

There are several ways a digital system can convert a binary number into the frequency of a chain of pulses. Two such ways are depicted in Fig. E-ii, and we have made PROMs with appropriate contents for each method. The simplest method is the reloadable counter shown on the left in the figure. The counter is driven by a fixed frequency oscillator. Every time it counts up to an all ones state, i.e. the carry-out condition, it is reloaded on the next clock cycle with the number specified by the PROM bits. The output of the subsystem is a chain of carry-out pulses with constant frequency. If M is the modulus of the counter and N is the number to be loaded, then the output pulse rate is

$$f = \frac{f_c}{(M - N)}$$

for $0 \le N \le M - 2$. The frequency range is from f_c / M to $f_c / 2$ but the spacing between possible frequencies is not uniform. (It is obviously also possible to use a down-counter and to reload on detecting all zeros. The result is basically the same.)

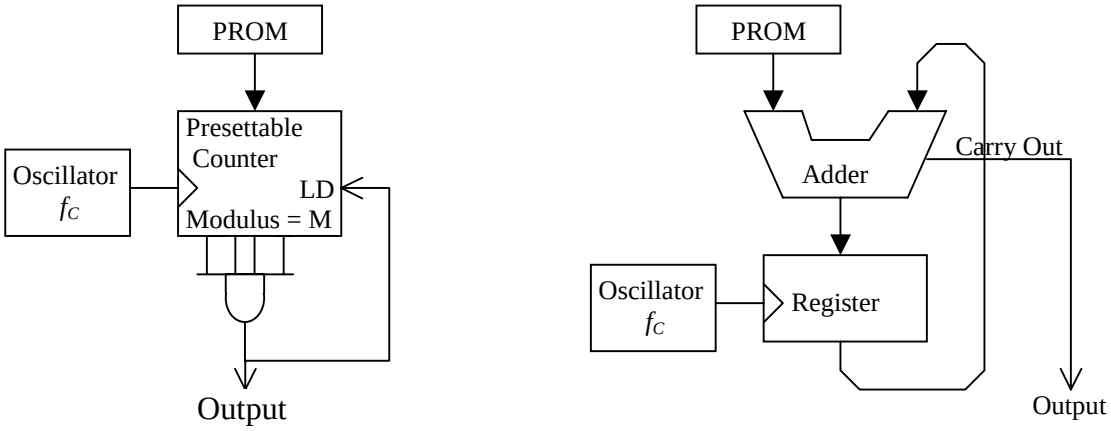


Figure E-ii: Two Methods of Frequency Synthesis

The second method, shown on the right in Figure E-ii, uses a combination of an adder and an accumulator. At the beginning of each cycle of the master clock, the value of the PROM output (considered as an unsigned binary number) is added to the contents of the register and the new sum is stored in the register. Carry-outs from the adder are the output pulses of the synthesizer. While the time between such pulses is not constant, the *average* rate of the pulses is

$$f = \frac{f_c \cdot N}{M}$$

where M is now the modulus of the adder and N must be in the range of $1 \le N \le M/2$. Again the frequency range is from f_c / M to $f_c / 2$, but the spacing in this case is uniform. The uniformity of frequency spacing makes this method an attractive one for commercial frequency synthesizers, which use a variety of tricks to even out the spacing between pulses. (It would be well worth your while to be sure you understand how the adder system works. Try seeing what will happen with an adder of modulus eight, i.e. 3 bits. Considering the N = 1,2,…5 cases should prove the formula as well as show the reason for the constraint for $N \le M/2$. (The *M/2* comes from the fact that if there is overflow on two successive clock pulses, there is only one output pulse, not two.)

You are free to choose whichever method appeals to you. To make a rational choice, consider the advantages of each: the counter method uses the minimal number of components and gives a pulse train with absolutely equal spacing. This means a pure tone. The adder method operates with a much lower clock rate for a given resolution at the upper end of its range. (You might consider the implications of the clock rate for the power used by your music box.) Moreover, the variation of time from pulse to pulse acts as a kind of frequency modulation, which will give some timbre to the notes. Frequency modulation as a way of generating timbre has been used in a number of music synthesizer systems. Whether the effect in this system is attractive or not is an empirical matter.

The PROM contents have been loaded on the assumption that the range of output frequency will be approximately from the A below middle C (A2 = 110 Hz. on the American Standard pitch scale) to the second A above middle C (A5 = 880 Hz.). Neither frequency will be generated exactly, but both will be fairly close. The PROM number, *N*, for the 880 Hz. note will be 3572 for the counter method and 2002 for the adder method.

In addition to the synthesizing circuit, you will need counters before and after the synthesizer. You will be starting from the crystal controlled oscillator on the BUXUSP-II board, which is at 1.843 MHz. You will need to scale that frequency down with a counter to derive the clock for the synthesizer. Since the output pulses are short duty cycle and the signal to the speaker is to be a square wave, you will also need to divide the output by two. In the case of the adder circuit, you might make the second counter a divide-by-four counter so that the short- term variations in the output frequency will be smaller. Figure E-iii shows the wiring of PROMs that we have set up for you already in the lab.

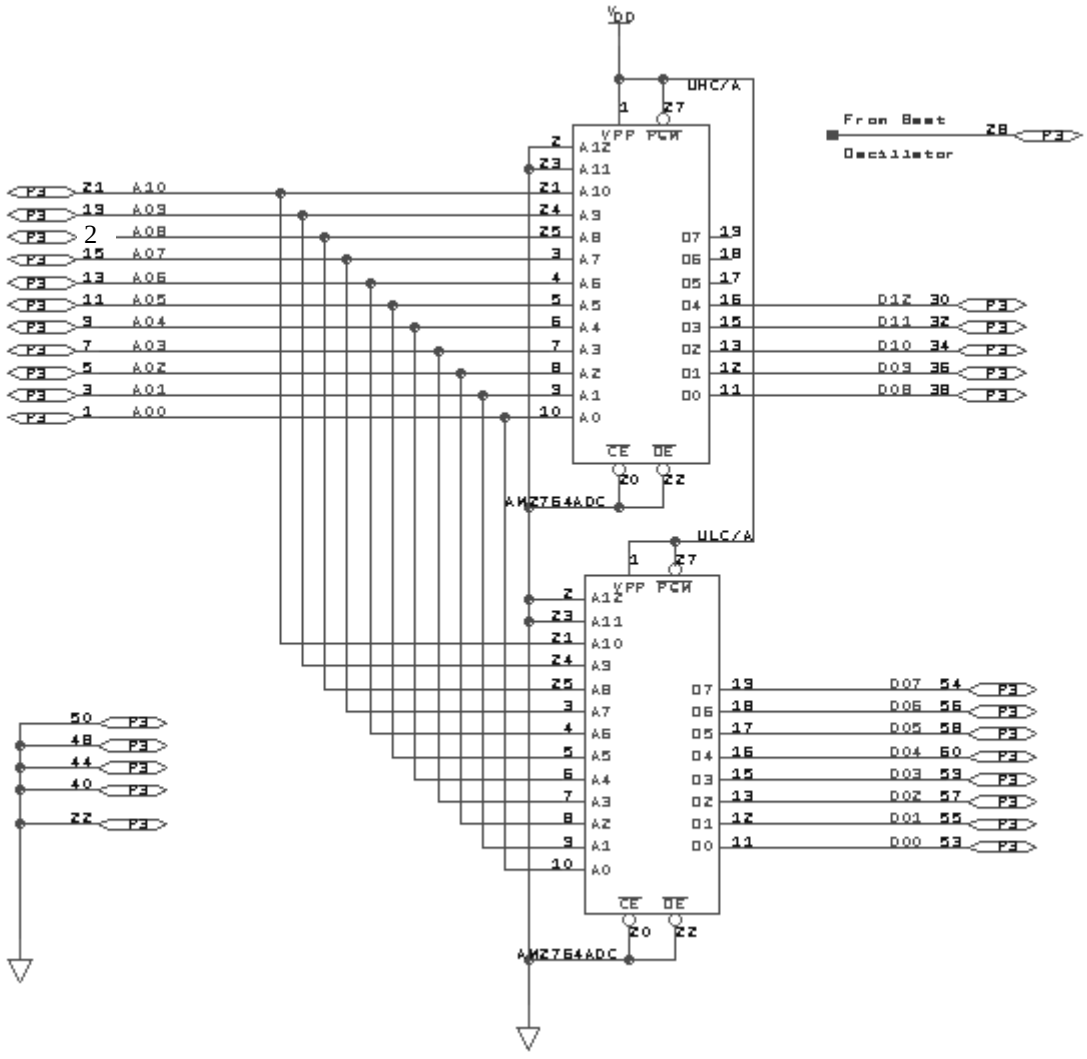


Figure E-iii: Prewired PROMs for Lab E

9.3.6. Lab F:

Build an Electric Sign Using VHDL Code to Program a Xilinx FPGA

Requirements: We have built a small circuit that attaches to the BUXUSP-II systems . It has eight LED displays, two memory chips, and some current limiting resistors. Figure F-i shows the schematic of the subsystem, including its connections to the 60-pin ribbon connector on the BUXUSP-II board. We have also written a test program for the PC that can accept a string from the keyboard and encode it for entry into those external memories. You have to program the XCS05XL on one of the BUSUSP boards to turn this system into an electric sign. (This is the kind of a sign that shows a printed message scrolling across it from right to left.) The circuit to do this consists of a number of counters, a finite state machine to generate control signals, an address generator for the display memory, and a driver for the display cathodes.

The point of this lab is not to set a difficult design so much as to give you a first opportunity to use the hardware description language VHDL. I have adopted a very simple approach to a pure VHDL design for the lab. You can do the entire lab from a single VHDL text file. This file, called "xlabf99.vhd" can be copied from the Engineering server in the directory "P:\ePD\en163". You edit the file to add the architecture descriptions appropriate for the three blocks (fsm_ctrl, addr_gen, and k_demux) that are discussed below. There are comments in the file telling you where the three pieces go. All blocks are similar to the examples I have discussed in class. All the wiring and pin numbering are taken care of either in the structural part of this file or in a constraint file "xlabf99.ucf" that you **must copy from the same directory location**.

When you have the sign working, you must have printouts of the ".vhd" file to show the TA. The FTQ may involve either a modification of the VHDL or an explanation of how it is supposed to work.

Discussion: The sign you build in this lab is an extension of what you did for the multiplexed display in lab 3. The display digits have common cathode wiring and all eight have their segment pins wired in parallel to form an eight character multiplexed display. The two memories are wired as a single register set with 64 words of 16 bits each. It happened that we had some old small memories with the unusual feature of separate data input and output pins with which to build this system. The data outputs drive the display segments directly with current limiting resistors. The cathodes go to pins on the Xilinx XCS05XL. The data inputs of the memory go directly to the Ports A and B of the PC, so no multiplexing is needed on the data pins for memory-write operations. For a display that does not flicker, all the cathodes have to be multiplexed-on sequentially at least once every thirtieth of a second. In the structural VHDL I give you, the 3 least significant bits of the 24-bit counter cycle from 0 to 7 continuously and with suitable decoding and shutoff, they can be used to drive the cathodes.

The memory address has to increment as the cathodes are strobed so that the proper data is on the segment pins for the cathode that is low at that instant. Overall the addresses have to advance at a rate of about one character every second to scroll the display. There is a set of six char-

acter msb count bits of the 24-bit counter that count cyclically from 0 through 63 at a rate of about one count per second. These can be the rest of what is needed to generate the memory addresses. Most of the time, this address generation and cathode strobe action is all the system does.

When the PC wishes to update the message on the sign, it writes new character data to Ports A and B, writes a character address to Port C bits PORTC[6:1], and then asserts the PORTC0 bit for a couple of microseconds. Your circuit must recognize this signal, turn off the display, multiplex the new address onto the memory address bus, and generate a strobe pulse to the /WE (Write Enable) line of the memory. A single clock cycle of the 1.834 MHz clock on the BUXUSP-II board is adequate to write the data. When the PORTC0 bit returns to 0, your circuit should return to normal operations. A finite state machine should be the circuit of choice to control this sequence. You may wish to put a synchronizing flip-flop in line with the PORTC0 input. You will certainly want to assure that the address is stable before asserting WR_N and that it stays that way well after the write pulse is done. Remember that the write enable line of a memory is very susceptible to glitches and do your design accordingly. There is no great rush to write the data, as 64 write operations of say 10 clock cycles of 1.843 MHz is still only 349 microseconds, a trivial amount of time for the display to be blanked.

I think that the natural division of the circuit is into blocks for the finite state machine, the address generator, and the cathode strobe mechanism. I have put equivalent blocks (entities) for VHDL code into the structural VHDL file you just downloaded. Figure F-ii shows a block schematic with those elements interconnected as the structural VHDL connects them. (Right now, you have to access that figure with DxDesigner – I have not been able to get it legible as a paste-in figure. It is also posted in the lab on the wall by the hall.) The displays have 13 segments plus a decimal point. I have made a lookup table to convert the keyboard characters 0 - 9, A - Z, and the special characters [,], +, -, /, \, !, and . into the nearest approximate display as the central part of the test program. The results are readily legible and correct operation can be sensed immediately.

There are two software choices for how to compile your VHDL into a Xilinx bit file. If you wish to do simulation at any point, it is best to start with Aldec and use its gui interface to launch functional simulation, then do synthesis and implementation followed by timing simulation. I have not yet written up how to do that but it is quite straightforward and begins with the functional simulation procedure you used for lab 9. The simple direct compilation using the Xilinx ISE 4.2 toolset, which is only available on the machines in Room 196, is the following:

1. Start *Project Navigator* as in lab C and choose File/New Project from its file menu. In the dialog box that follows, choose the SpartanXL family and the XCS05XL-4PC84 device, and change the design flow to "Synplicity Pro VHDL." (This means you will actually use the Synplicity Pro™ synthesis tool. This is one of the best tools for FPGA work on the market and runs under a separate license.) Enter a name for your project, and modify the name for the directory in which the work will be stored to something convenient for you.

2. Right click on the line "XCS05XL-5PC84 Synplify Pro VHDL" in the "Sources" window. Select Add Sources and browse to your vhd project file. Accept that choice and also select "VHDL Module" for the file type.

3. Now sequentially double click or right click on the Synthesis, Implementation, and Generate Programming File entries in the "Process" window. This is the same procedure as Lab C and should be familiar by now. Correct any problems with your source code until you are able to run to completion.

4. Check that the software has found and used the "ucf" – Universal Constraints File – that forces pin wiring to connect to the lab setup properly. Expand the "Design Constraints" entry in the "Process" window and choose "Edit Constraints File (Constraints Editor)". After a moment, a spreadsheet-like window will open. Select the PORT tab and check that pin entries are available for all pins and that they correspond to the pins on the block diagram. If they do not, then your copy of the constraints file is not in the right directory. Close the constraints editor.

5. Download the bit file to a BUXUSP-II board to which the sign protoboard has been connected. Run the test software for lab F from the D:\Test Software\En163 directory.

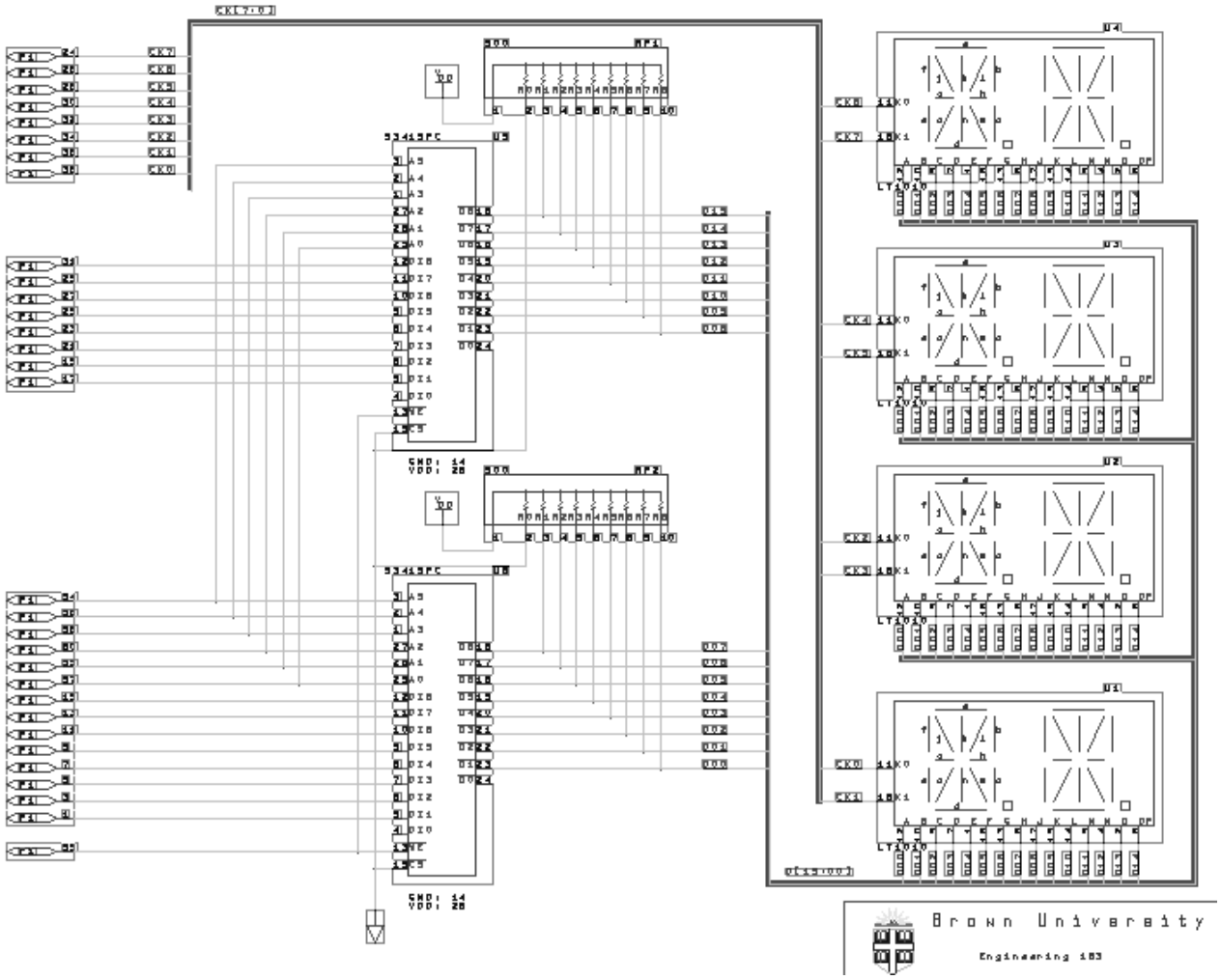


Figure F-i: Display Schematic with Memory and LED displays.

Please check the DxDesigner schematic “xsign_blockdiagram” in the Engineering 163 DxDesigner library. I could not paste a legible version of this picture into the manual.

Figure F-ii: Block Diagram Schematic of XCS05XL Blocks – Entities and Wiring.

9.4. The Use of a Logic Analyzer

The Use of a Logic Analyzer - Instrumentation Demonstration

We have a dozen new logic analyzers purchased in the summer of 2001 at a very deep discount from and courtesy of Agilent Technology Inc, the successor to Hewlett-Packard's Test and Measurement Instrument Division. I should like to express my gratitude to them for the gift. These instruments are essentially multi-channel oscilloscopes especially designed to display signals from logic systems. Depending on the particular box, these will display 34, 68, or 102 inputs simultaneously. Input signals are converted to logic levels and may be displayed either as timing diagrams or as state tables. Synchronization is extremely flexible, being software-selectable on such logic events as the occurrence of a particular word or the first edge of a given signal after a certain word. Seven of the instruments also have a two-channel digital oscilloscope that can be triggered from the logic analyzer section. You may use that capability in lab 6.

We are requiring you to learn to use State/Timing section of this instrument because it is the fundamental tool for logic circuit analysis and debugging. The immediate reward is a modest number of points when you demonstrate that you have used the machine to characterize one of the labs which are open to simulation in lab 9. To gain these points, you must demonstrate to a TA that you have the same signal pattern as you would be required to simulate for lab 9. The signals you display and the trigger points and time scales to demonstrate are those called out in lab 9. You also **must set up the screen labels of the analyzer** to show the proper names of the signals, that is, the same mnemonic names that appear on your schematic. (The schematic must have sensible mnemonics!) Ideally, you should plan on doing both the software simulation and the logic analyzer measurement on the same lab. Direct comparison of the simulation printout with the analyzer screen would certainly be the simplest way to convince the TA of your accomplishment. When the TA is satisfied with your measurements, she will sign off your scorecard in the usual way.

Attaching an analyzer to your circuit should be fairly straightforward. There are multipin clips in the lab that fit over integrated circuit packages to contact all pins at once. We have 16, 20, 28, and 40 pin versions of these. If you space your devices astutely, then you will not have a problem with 14, 18, or 24 pin packages as well. The one thing I ask of you is that you respect the equipment. The analyzer leads are fragile and we expect some breakage. They are also expensive -- more expensive than we can afford to replace unless you use some care. Most of the time only the end breaks off and we can put it back on. We can only do that if we have both the end and the wire. **IF A LEAD BREAKS, PUT ALL PIECES OF IT IN ONE OF THE MARKED PLASTIC BOXES FOR BROKEN LEADS!**

Well-written manuals on how to set the controls are with the machines in the lab. The TAs can be of some help, but some of this you will have to work out by trial and error.

10. The Engineering 163 CPLD-II Board

Introduction: I have built a small printed circuit assembly that I call the Engineering 163 CPLD-II Board as a partial response to the obsolescence of the construction style that you use in the lab. (The "-II" means second generation – the first board had design and cost problems and was re-built.) The integrated circuits in your kit are in DIPs or Dual In-line Packages that can plug into protoboards. They are increasingly difficult to get because they really do not represent current packaging technology. Still, if you actually want to design anything electronic, you will not find it in a single package already – otherwise it would not be a "design" problem for you! My goal in this course is not simply to teach Boolean algebra but to develop the mindset that can partition a system, design its pieces, assemble it, and understand it well enough to debug the final product, whether it has simple soldering problems or more complex oversights of design specification or implementation. To achieve that goal within a hands-on context requires finding ways to use modern packaging while still allowing you to assemble and debug the product.

The main problem with conventional gates such as those you use for labs 0 and 1 is that they are not very flexible without extensive wiring. Programmable Logic Devices or PLDs attempt to solve this problem by building large AND-OR circuits that can do SOP logic and by connecting the logic outputs to simple registers. The gates themselves use NOR gate logic of the kind we first looked at in class, namely N-channel MOSFETs in parallel and connected to a pull-up device. Their flexibility comes from using floating gate transistors for the pull-down devices in the AND plane part of the NOR gates. By suitably stressing these transistors with a relatively high voltage, one can raise or lower their device threshold voltage, effectively wiring them into the circuit or taking them out. (The programming voltage is generated on-chip. I talked about how these devices work in class.) This is wiring flexibility without physical rewiring. That "rewiring" is called programming and is done from compiled text descriptions of the logic in ABEL or another standard format. Once programmed, the circuit principles are the same as other gates, but the choice of an N-MOSFET circuit rather than CMOS for the logic sections has consequences. In comparison to CMOS static gates like the 74ACT04 chips in your kit, PLDs can:

- Implement multiple SOP expressions with large numbers of inputs but usually with a limited numbers of product terms. In comparison, CMOS gates generally do only one product term or a simple OR of a small number of inputs.
- PLDs are a little slower than discrete gates and much slower than gates that are interconnected within a chip. Discrete gates pay a very large speed penalty for connecting off-chip, the only way that you can do external wiring without changing circuit manufacture.
- PLDs draw a large amount of power continuously because of their pull-up devices. By comparison, static CMOS gates draw almost no power unless they are changing output state. Almost all power in static CMOS gates is dynamic power from charging and discharging stray capacitance.

With the CPLD Board, you can embed a current generation Complex Programmable Logic Device (CPLD) into a breadboard system and make it interact with other components. This chip, a Xilinx XC9572XL, is capable of implementing substantial amounts of logic including state machines and comes in a 44-pin plastic leadless chip carrier (PLCC) that cannot be hooked to a protoboard without something like my new printed circuit board.

CPLDs are one of two classes of device that attempt to solve the problem of providing a flexible means to build large amounts of custom-tailored logic without the development costs of custom or semi-custom integrated circuits. The other class of such devices is the Field Programmable Gate Array (FPGA), one of which is used as the basis of labs C through F. They address markets for prototype development and small volume manufacturing. (By small volume I generally mean something under 20,000 to 100,000 units per year. The non-recurring costs for integrated circuit development are quite high and generally limit the use of application specific circuits to products with large unit sales.)

Of course, there are tradeoffs to be made in deciding whether to implement a system with a CPLD or an FPGA. Programmable logic devices are generally built using the same technology as flash memories and other electrically programmable, read-only-memories. (The basis of all these devices is the floating gate MOSFET that I discuss at least briefly in class.) Most of the deciding factors in the choice between the two approaches are set by the properties of the floating gate technology. The main qualitative features of the CPLD are:

- The logic pattern you program is non-volatile, that is, once programmed it does not have to be reprogrammed whether or not you turn off the power. (Charge stored on a floating gate is generally stable for decades.)
- The circuit topology allows the formation of Boolean products with very large numbers of inputs at no speed penalty.
- Propagation delays from pin-to-pin are relatively uniform and easy to predict.
- Some devices, including the one used on our board, can be programmed while in the circuit at very low cost, allowing optional design features and upgrades and further lowering the cost of initial product.
- One disadvantage is that compared to FPGAs, CPLDs offer only moderate speed and density. (For example, the one on our board has a 10ns nominal delay and 72 flip flops whereas the comparable generation FPGA of the same price had 4ns best-case delay and about 144 flip flops.) This is the result of more graceful scaling of FPGA architectures.

The Board Itself: Figure 10.1 shows a block diagram of what I have built. The actual board encompasses the things inside the dotted rectangle. There is a 26-pin ribbon cable connector on the board and the cables we provide have a 24-pin DIP plug on the user end. (This is the rectangle on the left in the figure outside the dotted box.) You can insert that plug into a protoboard just as you would a DIP integrated circuit. It affords what I hope is a quick and convenient way to make up to 18 logic I/O connections to the XC9572XL CPLD. The board also has a two-digit, seven-segment display with current limiting resistors wired to 8 other pins of the CPLD. The display has a common-cathode configuration with the two cathode terminals also going to the DIP plug. Three pins carry power and ground connections and one more of the 24 pin DIP connector pins is an I/O line that can be dedicated as a global reset line.

Every member of the Xilinx XC9500XL family of CPLDs has an industry standard JTAG test port inside it. The device can be programmed *in situ* through that port with a suitable programming cable attached to a dedicated connector on the board.. (The JTAG idea and the standards to implement it are covered in Lab C. This industry standard provides a scheme for testing completed boards and circuits using a four-wire serial interface.)

<u>WARNING!!</u> When you start to use a CPLD board for the first time, you *<u>MUST PROGRAM IT BEFORE YOU</u>* connect it to your protoboard. The CPLDs are easily damaged if an output pin is connected to a signal source, as it might be if a prior user had different pinouts programmed than you use. To power up the board for programming there are special power cables that connect to the ribbon cable plug on the board. Ask a TA!

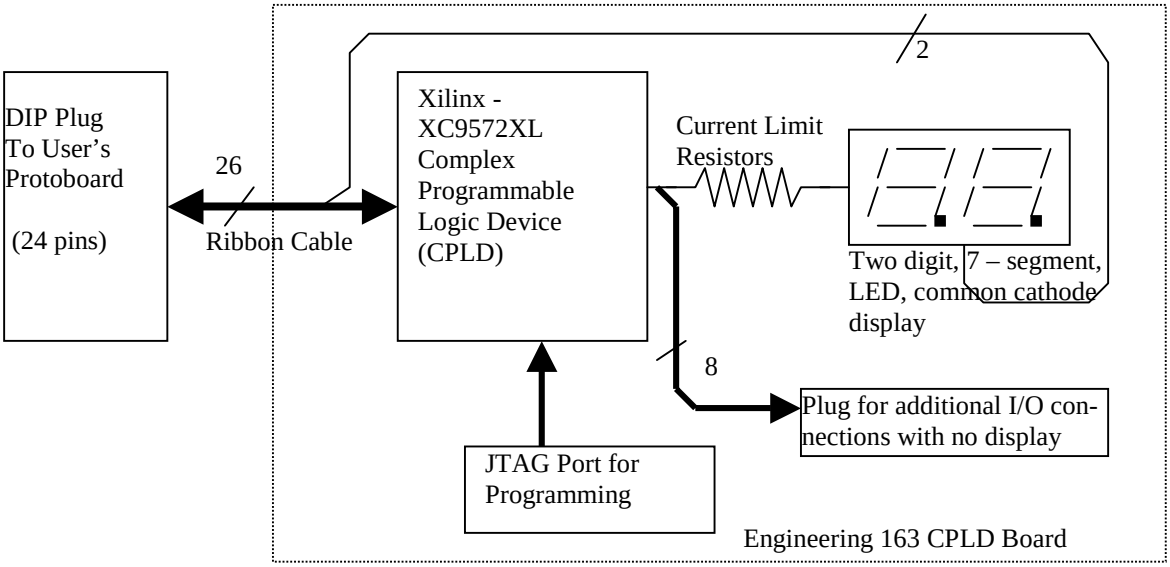
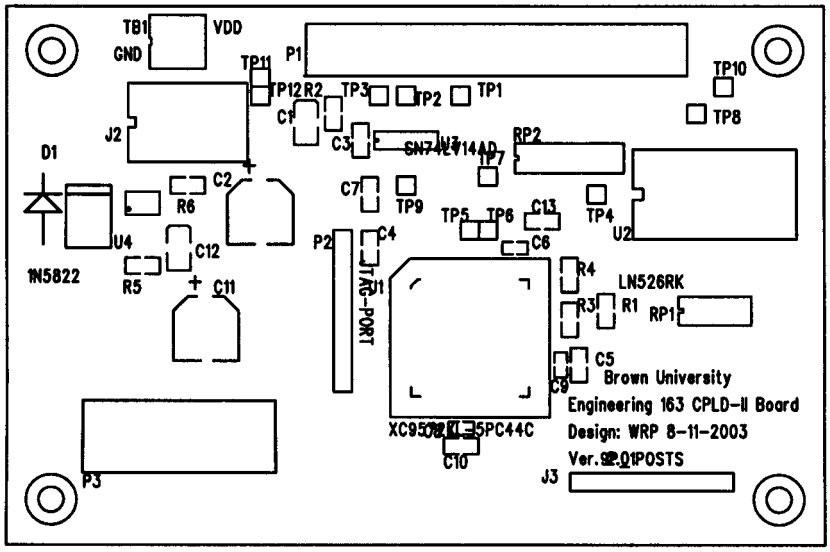


Figure 10.1: Block Diagram of the Engineering 163 CPLD Board based on the Xilinx XC9572XL

Figure 10.2 shows the top silkscreen pattern for the board along with a table identifying the various connectors. The user cable connector P1 is at the top and the JTAG connector P2 at the bottom. I added a connector compatible with the logic analyzers so that 16 channels of logic analyzer connections can be made with one cable to the first 16 signals in the user cable. The connections to the display also come to a connector so you can probe them during debug or, if the display is not needed, they can be used for additional I/O connections. When I was done with the basic functionality of the board, I noticed extra resources available in spare inverters and extra I/O terminals. I brought all these to test point patterns on the printed wiring so one could use them if needed. Some also go to another connector that allows jumper configuration changes. I do not anticipate that you will need to change these.

The full schematic for the board is shown in Figure 10.3. The XC9572XL requires a 3.3 volt power supply while you usually use a 5.0 volt supply. (The reason for the choice is cost and probable lifetime. Xilinx has raised the cost of their 5-volt parts to almost 3 times the cost of the 3.3 volt parts. Also, they are clearly signaling that they regard the 5-volt parts a obsolete and they may remove them from production soon.) Therefore, I included a regulator chip (U4) on the board with its supporting components to generate 3.3 volts from 5.0 volts. The XC9572XL will tolerate 5-volt input signals (it can be destroyed by 12 volts, so be careful). However, its output is only 3.1 volts or so, so while it is compatible with TTL parts, it may give trouble with some CMOS parts. In addition to the components already mentioned, there is the usual complement of bypass capacitors. In using the board you have to supply power and ground through the ribbon cable. There is a large diode (D1) connected from VDD to GND that I hope will partially protect the board if you connect

power incorrectly. (Please don't do that anyway!) The XC9572XL has certain pins that are the only ones you can program to be global clocks, something you need to do for state machines and counters. Because ribbon cable connections are notoriously poor for fast clock connections and because clock edge problems are both obscure and difficult to fix, I have sent the signal for the GCK1 clock on pin 5 through an RC filter and two stages of Schmitt trigger inverters. This preserves the sense of the clock signal while smoothing it somewhat and exploiting the noise margin advantages of a Schmitt trigger device. (The Schmitt trigger sections are in the 74LV14 hex inverter – U1.) This, of course, means that pin 5 of the CPLD cannot be an output. There are similar limitations on the placement of three-state output enables and of a global reset line. I have made sure that pins for three-state enable and for reset are available on the 24-pin DIP plug.



Ref.	Function
P1	Ribbon cable to user
P2	JTAG program cable
P3	Logic Analyzer plug
J2	Jumpers (leave alone)
J3	Optional I/O points

Figure 10.2: Silkscreen Pattern and Connector Functions for the CPLD Board

HINTS: Xilinx recommends against doing what I told you not to do too, namely leaving unused inputs unconnected. It is good practice that uncommitted pins should be tied HIGH or LOW with a resistor. (You can tie a bunch of such pins together and then through a resistor to ground or VDD. This prevents trouble in case one gets accidentally programmed as an output due to a typo.) Also there is a set of 4 pins tied to test points [TP4, TP6, TP7, and TP8 on the schematic] to which I have added individual pull-up resistors. I recommend using those pins for outputs you don't need to connect elsewhere but which you might want to probe with the scope or logic analyzer during debug. There is also a 20-pin header that mates with a special plug we have in the lab to go on a logic analyzer cable. This can connect 16 analyzer inputs to 16 of the CPLD inputs as shown in the pin assignment Table 10.1.

Wiring: The 44-pin package of the XC9572XL, the ribbon cable, and the users' DIP plug all use different numbering conventions. (We have included this adapter cable in every kit. We recommend that you minimize pulling the DIP plug in and out of your board, as the pins are easy to break. Leave it plugged in between labs.) As a result, figuring out how to make connections on the

protoboard is non-trivial. The pin numbers of the DIP plug follow the same conventions as the through-hole integrated circuits in your kit. The ribbon cable, the connector for which is the only part of the protoboard connections to the XC9572XL actually shown on the schematic, uses still another convention. To simplify your task, Table 10.1 below shows how those numbers line up with the pins of the ribbon cable and the pins of the XC9572XL. For wiring your protoboards from the DIP plug, you may ignore the ribbon cable column. The table also points out those pins with special functionality, such as possible global clock or output enables in the CPLD. Complete pin-out details are given in the data sheets at the end of this manual.

The connections to the display are shown on the schematic. Notice that the Xilinx pins 33, 29,28, 27, 26, 25, and 24 each connect to two segments of the display, one from the left and one from the right. This makes multiplexing the display trivial by if you drive the two display cathode connections to ground alternately. When most of the display segments are lit, the cathode load current exceeds the maximum pin current of the XC9572XL, so we recommend using a 7406 for driving the cathodes. See lab 3 for a further discussion.

The XC9572XL CPLD: A partial data sheet for the XC9572XL itself is in this manual and the full set of data from Xilinx on this family of parts is available both on their web site and on the Engineering 163 site. The parts are built using floating gate N-channel MOSFETs to implement non-volatile programmable logic in the form of AND-OR trees. I will talk about the transistors and the logic trees in class. This part can form up to 72 output signals that can be either registered or not. (However, the package only allows 34 of these to connect to physical package pins. The other nodes are "buried" but may be used for hidden logic variables.) Each output can have 5 Boolean product terms with up to 36 variables in a term. More terms are available if one forms fewer outputs. There is a large switch matrix before the AND-OR trees that gives very flexible feedback logic for state machines. The architecture implies some complicated rules for how logic is formed, but much of the burden of those details is taken care of in the software.

Programming the Xilinx CPLDs: As a practical matter, generating instructions for modifying the floating gate transistors to realize your logic can only be done through a suitable CAD tool. There is too much data that has to be put into proprietary formats to do the job by hand. Thus what you see when you go to use these devices is some description determined by the software rather than the CPLD's own architecture. With the Xilinx software, there are three possible forms for specifying the desired logic:

1. ABEL programming language
2. VHDL or Verilog HDL's
3. Schematic entry through either DxDesigner or Xilinx's own tool

Schematic entry does not make a lot of sense for small circuits that are heavy on large Boolean expressions such as are common in CPLDs. The VHDL and Verilog languages have more syntax structure for dealing with complex situations than is needed for CPLD use and they tend to hide the link to hardware rather than expose it. ABEL on the other hand is a simple language that maps the basic logic structure of the CPLD very well. Like PASCAL or JAVA as beginning programming languages, ABEL is a good introduction to a Hardware Description Language. For this reason, I want you to do the programming of the board in ABEL.

The class text by Wakerly has very good sections on ABEL usage. There is a full description of the language in the on-line help section of *eProduct Designer* through the Intelliflow tool. (Do **NOT TRY TO USE** that tool for the XC9572XL!) There is also a reasonable coverage of ABEL for the Xilinx ABEL compiler in the on-line help for their tools. This especially includes the available signal qualifiers. (A signal qualifier is something like the ".FB" that is appended to feedback signal in a file to make a registered signal go directly from flip-flop to AND plane.) Finally, there is an example program accompanying lab 5.

Table 10.1: Pin Assignments Between XC9572 and User Connections

DIP Plug (Protoboard)	Ribbon Cable	Pin on XC9572	Logic Anal. Chan.	Function
1	2	5	4	Input to clock buffer and global clock pin 5 of XC9572.
2	4	1	0	
3	6	2	1	
4	8	3	2	
5	10	4	3	
6	12	6	5	
7	14	7	6	
8	16	8	7	
9	18	9	8	
10	20			Ground
11	22			Display Cathode
12	24			Ground
13	23			Display Cathode
14	21	39		I/O or global reset
15	19	42		I/O or global tristate 1
16	17	22		
17	15	20		
18	13	19	14	
19	11	18	13	
20	9	12	10	
21	7	13	11	
22	5	14	12	
23	3	11	9	
24	1			VDD
	26			Ground
	25	40	15	I/O or global tristate 2

Hints on ABEL Compilation: this procedure should be sufficient for you to enter your ABEL design, select the type of CPLD to target, and process your file for simulation and implementation. There is a problem with selecting the appropriate version of the Xilinx software for this procedure. Please read the document *Versions of Xilinx Software – 2004* on the class web site. You will type in an ABEL description similar to the example in this lab and will get out several new files. These include a schematic symbol to use for system-level simulation, a ".vhd" file (a VHDL file that one uses for functional simulation), and a ".jed" (or JEDEC) file that is used to program the actual device. Compilation may be done on any machine on the Engineering server. Downloading is done on one of the machines in the Hewlett Computing Lab.

1) Select the ISE 6 version of the Xilinx software by the menu sequence /Start/Electrical/Xilinx/Xilinx ISE 6/Run Me First. This runs a batch file that selects the ISE 6 version for you.
2) Open the compile software with the menu sequence /Start/Electrical/Xilinx/Xilinx ISE 6/Project Navigator. In Project Navigator, select the File/New menu sequence to open a project dialog. Supply your ABEL file name (without the ".abl" extension) as the project name, change the project directory name to your root directory or to U:\Wv and select "HDL" as the "Top-level Model Type. Click the Next softbutton and set up the project's configuration as: Device Family = XC9500XL CPLDs; Device = XC9572XL; package type = PC44; Speed Grade = -10; Synthesis Tool = XST (ABEL); and Simulator Tool: Other. Hit "Next".
3) In the next dialog, select New Source and then ABEL - HDL Module filling in your project name with an ".abl" extension as a File name for your ABEL code. For purposes of the lab in this course you can usually then click Next and Finish until the dialogs stop and a panel opens the outline of a module for your ABEL code. Please remember that the Module name **on the module line of the abl file cannot have more than 8 characters.**
4) You now enter your ABEL code. You can do this by typing directly into the module window or you can copy and paste from a file written with an Editor program, such as "Notepad" or "Wordpad." The module window has extra help in its color coding of ABEL syntax to aid in error detection.
5) On the left side of the Project Navigator screen, there are two panels: an upper one showing "Sources in Project" and a lower one for "Processes in Sources." In the lower window for Processes left click on "Implement Design." It will now show several possible actions: Compile, Translate, Fit, and Generate Program File. Double click on "Compile." During this step, the compiler will analyze your file for basic syntax and will create the VHDL and several other intermediate files. Most syntax errors are caught at this point, and, if there are errors in pin or node allocations they will turn up here too. You can edit your source file within the Project Navigator using its built-in color-coded editor. An "!" mark next to the "Compile" option means that a warning has been issued. Be sure to scroll through the messages in the bottom window to see whether these are significant or not.
6) When the file compiles properly, go on to the Implement process and right click and run it. This step maps your netlist into the actual chip wiring and will show whether your logic fits within the chip.

Figure 10.3: Schematic Diagram of the Engineering 163 CPLD Board

Figure 10.3 (Continue): Schematic Diagram of the Engineering 163 CPLD Board

7) Implementation should include running Generate Program File. This step produces a JEDEC industry standard file (with file extension ".jed") describing how the device will be programmed. At this point, the next step is to use the downloading cable in the lab to put your program into a CPLD board.

Downloading Instructions:

To program your circuit, find a machine in Room 196 that has a Xilinx JTAG downloading cable on it. There are more boards in the lab than there are JTAG cables, but once loaded your part will retain its contents even if you disconnect it from power to move it to another machine or to put it aside. Each PC with a JTAG cable will also have a ribbon cable connected to an adjacent power supply for **powering your CPLD board during initial programming. This is important because it prevents destruction of CPLDs by mismatches between your pin allocations and those used by the last person who used the board.**

1) The JTAG cable connects to the CPLD boards by a short flat ribbon cable that is keyed to prevent you from putting it on the wrong way. Hook up the board to the power supply with **the dummy power-only cable**. Ask a TA if you have any doubts about how to make connections. With these two connections, you can turn on power and check that the LED on the JTAG cable turns from yellow to green.
2) From the start menu choose /Start/Programs/Electrical/Xilinx ISE 6/Accessories/Run Me First to select the ISE 6 version software.
3) From the start menu choose /Start/Programs/Electrical/Xilinx ISE 6/Accessories/iImpact to open the downloading tool. When it opens, there will be a series of 3 dialog windows. The default values in these should be correct. The first is Operation Mode Select – choose Configure a JTAG Port. In the second you select Boundary Scan Mode. In the third you select "Automatically Connect".
4) After these selections, the program will find the cable and test its availability. If successful, it will open a dialog in which you select your configuration or ".jed" file. The result will be a display showing an icon for your XC9572XL. Right click on the icon and select "Load Configuration". This will open a file dialog in which to select the JEDEC file <your_project>.jed.
5) Do the right click on the icon again and select "Program" to do the actual download. In the dialog the opens, make sure to check both Erase and Verify before hitting OKAY.
6) If the *iImpact* program says "Programmed Successfully" then you are ready to test your system.

11. Schematics and Timing Diagrams

11.1. Documentation

Documentation of Circuits: Schematics and Timing Diagrams

Engineers earn their pay by devising ways to make things and by supervising manufacturing. Commonly, electrical engineers design devices that may range in complexity from five dollar dining room light dimmers to multimillion dollar satellites and utility systems. Almost never do they actually make anything themselves for reasons ranging from economics (they get paid too much...) to reliability (technicians are often better craftsmen...). An obvious consequence of this division of labor is that engineers must both communicate the results of their efforts and understand such communications. Thanks to the work of St. Cyril, Gutenberg, Xero, and others, most of this information is in written or pictorial form. An IEEE survey a few years ago found that the average engineer spends about thirty percent of his or her time generating this documentation.

There are two general types of documentation providing different kinds of information, one set of data for users and managers and another set for manufacturing. We are primarily concerned with the latter. Manufacturing documentation is diverse and includes at least such things as:

1. Block diagrams
2. Schematic diagrams
3. HDL design and testbench files
4. Parts lists (Bill of Materials – BOM)
5. Net lists (lists of interconnection for printed circuit fabrication)
6. Programmable device design files and/or ASIC (Application Specific Integrated Circuits, *i.e.*, custom chips) specifications
7. PCB (printed circuit board) plots of interconnections, particularly silk-screen and PCB assembly drawings showing part locations and annotation
8. Layout and assembly drawings
9. Circuit descriptions
10. Simulation documentation
11. Subsystem timing diagrams
12. Test specifications for verification of functionality and reliability.
13. Reports of calculations that insure some requirements are met by design. This is not a requirement in this course but is certainly a common one in most environments.

The schematic diagram shows all the interconnections in a system in forms that are easy to relate to their functions. At least in small systems, it is the commonest and earliest product of the design process. Of all the forms of documentation mentioned above, the schematic is the most standardized and the least subject to idiosyncratic company customs or to other situation-specific rules. We are asking you to draw schematics of your labs to this more or less universal form.

Current professional practice is to draw schematics with appropriate Computer-Aided-Design (CAD) programs, a process called schematic capture. The reason for this development is that a schematic file can be used both to simulate the circuit and to generate the engineering documentation for building it more or less automatically with much reduced engineering effort. The process is non-trivial and requires adding design information at each additional step. However, it guarantees consistency of part types and their interconnections between the schematic, the simulation, and the implementation whether on a Multi-chip Module or on a printed circuit. Changes to a system can be made rapidly at low cost to fix mistakes or to add new features. The result is great improvement in engineering productivity. The computer-drawn schematic will be the basic design expression for digital systems for some years to come. Eventually even higher, more abstract levels of system description will displace it, and it too will be a derived product.

Given this reality, we have decided that you must do your schematics with a CAD tool. The Mentor Graphics Corporation has graciously made their *eProduct Designer* system available to us inexpensively. It is available on the workstations in the Instructional Computing Facility and in room 196. The section of this manual on *DxDesigner* hints tells you how to gain access to it.

Handing in schematics: You can print your schematics on a laser printer in the instructional computing lab. Hand in printed copies of your schematics to the specifications given below through the box in Room 196, the same procedure as for labs 2 and 6.

Unfortunately, learning to use a complex system like *DxDesigner* takes some time. There will be a couple of help sessions covering this material. There is on-line documentation within the program, but trying out the example schematic in the hints section of this manual is probably the fastest way to learn the system. As examples to guide your drawings, you might consult the schematic for the CPLD-II board in section 10 or Figure 11.1 that shows a schematic for the prototyping board used in your later labs. In making drawings, please adhere to the following ground rules:

(1) **Initial Setup:** You must begin each drawing by enclosing it in a frame, called a “sheet”, which corresponds to selecting a paper size for the full-size picture. A complex system may have its schematic run over several sheets. The frame also sets up some crude position coordinates and has a block in the lower right corner for title information. You must fill in that block with a title that is consistent over all sheets of a schematic. You may add a subtitle unique to each sheet for multiple sheet drawings. Also fill in the date, your name, a version number, and the sheet numbering. (Even a single-sheet schematic must be marked “SH 1 of 1”.) The sheets are treated as standard components and are found in the en163 component library as the files: “asheet.1”, “bsheet.1”, and “csheet.1”. A single “csheet” should be about right for your first drawings.

(2) **Components and Symbols:** The part symbols for a drawing are invariably drawn from a symbol library. In this way, symbols are standardized, and much other information such as the package type, vendor part number, simulation model parameters, etc. is bound to the schematic at the same time with no extra effort from the designer. Most of the parts you will need are in the “en163” library. This library includes all the integrated circuits from your kit as well as resistors, capacitors, mechanical switches, pins, power symbols, etc.

(3) **Signal Flow:** When practical, signals should flow from left to right and top to bottom. This is not always possible and should not be regarded as an ironclad rule, but it does make reading a schematic easier. The commonest exception is that jacks and plugs, which contain both inputs and outputs, may be drawn with all connections together on either side of the page. Bi-directional lines may approach an integrated circuit from either side of the block.

(4) **System-Level Interconnections:** Connections between points in a system are made by drawing a wire, called a "net," between them. When parts are too far apart or appear on separate sheets (pages), then you connect separate wires by giving each of them a common label. It is good practice to label even a signal that only runs on a single line if it will be referred to in simulation or may be important to probe in debug. [See also the discussion of "labeling" below.] Often, signals have to leave your system through some external connection. Sometimes this is through an actual connector, but you will sometimes do it with just a wire from your breadboard to instrumentation. In the latter case, mark these points with an appropriate "pin." The ones available in the en163 library are "in.1", "out.1", and "bi.1". The pin is simply an indicator for the human reader. To help him or her understand the purpose further, add a text description of the connection.

(5) **Connectors:** Connections through real physical connectors use a different set of symbols. Like other actual components, connectors also have information about their physical structure in their library descriptions. You will probably not have any connectors in your early labs because you only use solderless breadboards. In lab A and the Xilinx labs C through F, there are computer and test jig connectors that you can put on a schematic with the connectors in the en163 library. An example of such usage is embedding the CPLD board in a schematic. The symbol for the DIP-24 plug from the CPLD-II board is in the en163 library too.

(6) **Labeling:** The drawing program lets you label a signal wire or net when you enter the "Add Label" mode and then double click on the wire to be labeled. The label is an alphanumeric text string that identifies the net in a readable form. In both simulations and netlists, a wire is referred to by its label. (Simulations use the label as the name of the signal carried by that net. Unlabeled nets are automatically given unintelligible but unique names.) The label will follow a wire no matter how far it is extended in a drawing or how much it is moved. Labels even cross from sheet to sheet within a given project. All nets in a project that have the same label name are assumed to be connected together. Assigning **a label is NOT** the same as just placing a piece of text next to a wire!

At the very least, **you must label** all intersheet signals, all wires which come from outside the system, all signals that are to be simulated, all bus wires, and the principal control signals. Any signal that is named in the requirements of a lab should have that name as a label in the *DxDesigner* schematic. How many of the remaining wires are labeled is a matter of taste. The goal is a balance between a well-annotated drawing and a cluttered one -- an aesthetic judgment. It is probably better to err on the side of too many rather than too few labels.

(7) **Bus Notation:** Frequently several different but closely related signals are connected along roughly parallel paths between a particular group of chips. Such a set of lines is called a

bus. Typical examples include data, address, and timing buses in computers. The example diagrams have several buses on them. To save drawing area and to emphasize the parallelism of their function, one usually draws all of these wires as a single line, representing a cable or a multiconductor set. *DxDesigner* offers a special command for this purpose, "Add Bus", which draws a bus as thick line. All buses must be labeled, preferably with a very short abbreviation of its function followed by a pair of numbers in brackets. The numbers tell the range of subscripts that distinguish individual wires. Each wire connecting to a bus must be labeled with the bus name concatenated with a signal number within the range of subscripts. For example, ADDR15, ADDR14, ... ADDR02, ADDR01, and ADDR00 all connect to the bus ADDR[15:00]. (The order of the numbers, that is [15:00] versus [00:15] is important when joining bus segments. Industry usage to minimize chances or errors is high to low, i.e., [15:00] is preferred.) *DxDesigner* allows one to label wires with automatic generation of a number suffix in certain circumstances. [See the *Viewdaw Hints* section.]

Remember it is only appropriate to use the bus convention for **closely related** signals. It is not correct to use it just to simplify drawing unrelated signals that happen to be going in roughly the same direction across the screen or paper. Use the bus drawing convention sparingly until you thoroughly understand the idea.

(8) **Power Connections:** The five volt power supply net automatically has the name VDD. Similarly the ground connection is labeled GND. Power connections to purely digital circuits are generally not shown but are handled by the library assignment of certain pins to the appropriate power nets. Analog and mixed function chips have their power connections shown explicitly on the part symbol, and these must be connected to the appropriate power nets by the designer. You will usually treat those other power distribution nets as simple labeled nets with whatever label is convenient. Where connections to VDD and GND are required, one uses the components "pwr" and "gnd". Two electrically equivalent versions of each are available with slightly different drafting conventions with file name extensions ".1" and ".2". (There is also a copy of the preferred "pwr" symbol stored as "vdd".) The label VCC is not used for power at all unless the designer makes that arrangement herself.

(9) **Reference Designators and Component Identification:** Every physical component, whether integrated circuit, switch, resistor, transistor, etc., must have a reference designator. This is a string that uniquely identifies the particular component in the system. There are standard conventions on the choice of such names. Typically an integrated circuit would have the form U1 or U22, for example. A resistor would be R3 or R4, meaning the third or fourth resistor on the drawing. Each component in the library comes with a reference designator. For example, all integrated circuits have the reference designator U? in the library. The first character or characters are the standard prefix characters for that part type. You have to replace the question mark with a number unique to particular part. You assign the reference designator by double clicking on the component with the mouse (you may have to enter "Select Mode" first). Choose the "Attributes" tab in the dialog, scroll down to REFDES to enter the new name, and hit either the Set or OK softbutton. Remember that the first character of the reference designator should be the same as the library default character.

Do **NOT** try to distinguish different sections of the same component with a suffix, *e.g.*, U3A or U3B. *DxDesigner* will consider this to be two different components altogether. Parts with multiple sections, like the 74LS00, are given the same reference designator for all sections, making sure that pin numbers of all sections are different.

Figure 11.1: Schematic of the BUXUSP-II prototyping board.

See printed foldout sheet for this schematic!

11.2. Timing Diagrams

Lab 9 involves software simulation of a circuit's performance. Most simulations will just be for one or another programmable device. Xilinx software, which is used to turn the netlist into programming code, also produces a simulation file. [You have to supply netlist information by doing schematic capture before trying the simulation.] You will transfer that file to Aldec's Active HDL software for simulation. The printout from that program is a timing diagram. In our experience such a diagram is the best tool for designing small timing systems such as those in Labs 8 and A. Usually one makes the diagram of the necessary timing signals before trying to design the clock circuitry and then checks that the final circuit will realize this specification.

Such diagrams are frequently used in system specifications, particularly interface specifications. Figure S-ii has an example of such a drawing. The drawing is part of a data specification from a Cypress Semiconductor VMEBus controller.

It is useful and usual to annotate timing diagrams with indications of causal relations. A curved line extending from an edge of one signal to an edge of another represents a causal link and the edges will be separated by the signal propagation delay time. If that time is short compared to the scale of the time axis of the drawing, the edges may appear synchronous, but that is not actually the case. Two transitions that appear synchronous and are not connected by a "cause and effect line" probably have a common cause and are indeed roughly synchronous. I have used this convention in the timing diagrams that accompany several of the labs in this manual.

The details of using the Aldec *Active HDL* software will be covered in a separate handout.

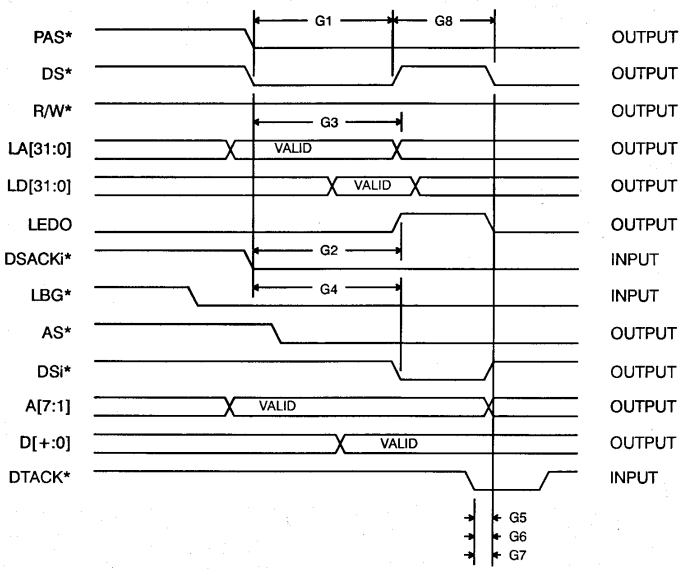


Figure 11.2: Timing diagram example.

11.3. Using the DxDesigner Schematic Capture Software

Hints on the Use of the *DxDesigner* Schematic Capture System in *eProduct Designer*

The starting point for the design and construction of circuits is usually entering a schematic into a CAD system, and from this drawing you can do simulation and construction with minimal additional effort. The simplest such schematic involves placing preexisting symbols and wiring them up. These notes should help you do this with a minimum of problems using the *DxDesigner* tool in the *eProduct Designer* toolset. These notes were written assuming that you will use the workstations in the Hewlett Computing Facility. There is a much more exhaustive guide from the Mentor Graphics Corporation itself available as on-line documentation. (On-line documentation is in the Help pull-down menu invoked on the right side of the menu bar as is customary in Windows-based software.) Since *eProduct Designer* is available in room 196, you can make changes while working on the lab. This should be useful in the CPLD (TM) or Xilinx-based labs if you need to make minor changes to a schematic. I have set up EN-163 parts libraries for both discrete and Xilinx parts, and you will want to take your symbols from these libraries most of the time.

The designers of *DxDesigner* have used certain common engineering terms in slightly specialized ways. Understanding this vocabulary usage is important for minimizing frustration. **A Component,** for example, includes not just integrated circuits, resistors, etc. Even the drawing frame that determines the size of the page and provides a place for descriptive text is a component. (Following usual drafting practices most *DxDesigner* libraries have sheets labeled in standard sizes A, B, C, D.... Often these are called "asheet.1" or just "a.1") Connections to VDD or GND use "pwr", "vdd", or "gnd" symbols found in the component library even though these are symbols of connections rather than actual parts. Symbol libraries may offer several variants on a symbol with different drafting practices for each; the variations are marked by different file extensions, for example "xxxx.1" or "xxxx.2" will be different forms of the same part "xxxx". **A Net** is a wire interconnecting two or more points. It extends however far one specifies a common connection. It is drawn with the command "Add Net" from the pull-down menu. To extend it from one sheet to another, the two segments must have a common "Label". Labels are attached to nets by making the "Add Label" choice from the pull-down menu as described in the tutorial below. Remember that labels are different from simple text, which only adds English annotation but does not affect connections. **Buses** appear as thick lines on the schematic and **must be labeled** so that the individual wires in the bus can be distinguished. Normally, the bus label is an alphabetic prefix followed by a bracketed indication of the number of lines in the bus. For example, an address bus with 16 lines might be labeled ADDR_BUS[15:00]. (Labels are attached to buses the same way as to single net wires.) Subsequently, individual wires attaching to the bus must have labels of the form ADDR_BUS01 for the second wire in the group. (The first is ADDR_BUS00.)

Every component, symbol, net, etc. has a set of non-graphical information that is recorded and tracked during the schematic drawing. This information includes such things as package type for printed circuit layout, logic information and electrical data for simulation, section or pin data for multi-part components (e.g., which of the 4 sections of a 7400 Quad NAND gate is to be used at

a given location), etc. Collectively, this information is known as the **attributes** of the object. Usually this information comes from the library. You only need to change it when the default information is incorrect or insufficient. (For example, resistors and capacitors need values added to them.) The one exception to this rule that **everyone** has to take care of is the **Reference Designator**. Library parts come with reference designators which are not unique, and *DxDesigner* does not generate unique ones as components are placed. The default designators are usually a prefix followed by a question mark. The question mark has to be changed to a number so that the components can be tracked uniquely. For example, all integrated circuits usually have the default designator "U?". You would change this to something like "U2", keeping the "U" as a prefix but numbering the ICs uniquely. The procedure to do so is in the tutorial example below.

DxDesigner runs on the machines in the instructional computing laboratory and on the machines in Room 196 as well. Everyone enrolled in the course has their own account and should customize its password once they begin to use the machines. Please do not share accounts or put off getting your account going. The first time you wish to use the *eProduct Designer* toolset, please follow the brief procedure to set up directories and initialization files that is posted on the class web site as a separate handout. Note that this procedure is executed only once and may have to be done on a particular machine. After you have done the startup procedure you can log onto any other machine for using the program.

How DxDesigner is organized:

DxDesigner has a huge number of commands available because there is a lot of potential detail in a schematic. Since different users have different fine motor skills, *DxDesigner* provides several ways to do the commoner tasks. To help you get a working understanding of the program quickly, we will explain the command entry process first. Since certain groups of commands affect similar aspects of the drawing and interact with the mouse in similar ways, we will first discuss how you select and move objects, change screen views, and move between sheets and levels. Finally there is a tutorial guide to drawing a simple schematic.

The Mouse: Like most graphics programs with extensive user input, this program is mouse-driven for menu choices, parts placement, node selection, wire routing, etc. For most commands, the mouse behaves the same way. You use the **Left Button** to select menus, to hit soft buttons, and to select and move objects. If an object has attributes, selecting it and double clicking the left button will open a dialog through which you can change attributes such as reference designators, component values, speed grades, etc. The attribute dialog also provides for labeling and marking signals as negative true, *i.e.*, inverted. The **Right Button** opens a short menu of commonly used commands for whatever is selected. If nothing is selected at that time, this menu still offers a chance to move to any other sheets in the project.

Command Entry: The mouse and the keyboard are used together to enter commands in five different ways.

(1) The mouse can activate pull-down menus located at the top of the drawing window. They are labeled: File, Edit, View, Add, Project, Tools, Window, and Help. In the command ta-

bles below, the first word of the menu choice column is the name on the pull-down menu bar.

(2) Certain keys can be used as single stroke commands. These are listed in the command tables in the "Single Key" column. If marked "<Ctrl>x", it means hold down the <Control> key then hit the "x" key simultaneously. You will often find that this is the most efficient way to enter frequently used commands.

(3) Many of the most used commands are also available as soft-buttons activated by single clicking the left mouse button on icons around the edge of the drawing. In the tables below, the icon buttons are labeled by position. The buttons referred to as "Tx," where "x" is a number, are on the top left edge of the screen. The number specifies position counting from the left to right along that bar. Similarly "Bx" denotes a button at the bottom left counting left to right; "Lx" a button on the top left counting top to bottom; and "Rx" a button on the top right counting top to bottom.

(4) Entering more complex commands, or frequent commands that do not have single keystroke or icon equivalents is done by hitting the <Space> bar to activate a command line at the upper left edge of the main window. (This line is called a "dockable" window because you can move it by clicking on its frame and pushing with the mouse. I myself like it where the designers put it.) Then type in a command. Again, the keywords for the commonest commands are included in tables II and III. This mode of entry is most useful for commands requiring that you type in strings, such as changing reference designators or other attributes.

(5) The right mouse button opens a menu that changes entries to anticipate the most the probable next operations.

Selecting and moving objects: If you want to change anything about something already on your drawing, whether its position, value, label, attributes, etc., you first have to select it. To do so, you have to be in "select mode," which is reached by pushing the top left softbutton. (It's marked by an arrow. Alternately, the single key "s" will do the same thing.) Then put the mouse cursor on the object and click the left button. A white box will appear around the object. To move it, just continue to hold the button down and drag the object by moving the mouse. To select more objects at the same time, hold the <Control> key down while clicking on successive elements. Any command to cut, copy, or paste will affect all the objects selected at that time. To change values, labels, or other attributes, select the object or label, double click the left mouse button, and enter the new information in the dialog box.

Changing Views: There is usually more detail in a schematic than can be clearly displayed at once. To make features legible, one zooms in and out and changes the displayed area. The special function keys at the top of the keyboard do these operations. Table I lists the particular function of each of these keys along with the alternate ways to do these commands.

Table I: Screen View Control – Zoom, Pan, and Refresh			
Key	Softbutton	Pull-Down Menu	Function
<F4>	R1	View Full	Return to the full of Home view of drawing
<F5>		View Refresh	Repaint the screen; needed after edit operations for cleanup.
<F6>		View Pan	Move the drawing view so that the current cursor position is centered on screen.
<F7>	R2	View Out	Zoom out by one step
<F8>	R3	View In	Zoom in by one step
<F9>	R4	View Zoom	Make a rectangular area selected by the Mouse the full screen view. Push left mouse Button and drag the mouse cursor over the Region to be the new view.

Multiple sheets and other levels: While Engineering 163 drawings tend to be fairly small, most diagrams require more than on sheet or page to hold all their parts. In the Mentor Graphics system, each sheet is a separate schematic file and is numbered by a simple numerical extension, for example, "my_junk.3" would be the third page of a schematic for the "my_junk" project. Within a schematic, each symbol links to other files having physical data and simulation models. There is a group of commands that enable you to move around within this hierarchy and Table II summarizes them.

Table II: Moving Between Levels			
Softbutton	Command Line	Right Mouse Button	Function
R8	psym	Symbol	Open the symbol file for a selected object for Editing. You generally will not have to do this.
R9	Psch	Schematic	Push to the underlying schematic for a selected object. Often an underlying schematic tells how a device is to be simulated.
R10	psheet	Next Page –or- Go to Page	Push (go to) the next sheet. (With R10 or the Go To Page command you can go to an arbitrary sheet.)

An Example Schematic:

The following steps will guide you through a simple example. Try them to get a feeling for how the software works. We have included the resulting schematic as Figure 11.3.

(1) Open *DxDesigner* following the instructions above. The first time you do this, the program comes up with all its bells and whistles showing. This is a more complicated system than is needed for a first cut at schematic capture. I recommend that you close all the tools to the right, left, and below the drawing window. Each of these three subpanels has a small x in its upper right corner and selecting that kills the panel. Should you need them, there are buttons on the toolbar to restore them. The remaining toolbars are "dockable" and can be moved around. I find that the best arrangement is to maximize drawing space while putting the most used buttons along the sides of the drawing. You move the tool sets by dragging one corner of each bar with the mouse until it is where you want it. The table of button assignments shown below is based on one placement of the button toolbars. We will demonstrate this operation in a help session.

(2) From the File pull-down menu, select "New" and assign your schematic a new project name. This will open a blank, black screen for drawing. (You may want to resize your screen with the mouse on the lower right frame corner to make the working boundary coincide with the white box.)

(3) Go to the Add menu and select "Add Component" for placing new components to your drawing. A window listing parts directories and components will appear. Select the "P:\ ePD\ en163" directory.

(4) From the parts list, select the "csheet.1" with the left mouse button. Holding the button down, pull the cursor to the lower left corner of the sheet. When the sheet frame is centered on the working area, release the mouse button. Congratulations, you have just laid down a sheet frame.

(5) Position the cursor just above and to the left of the title block on the lower right corner. Hit the special function key <*F9*>; holding the left mouse button down, drag the cursor down and to the right until the title block is surrounded by a green rectangle. Release the button and now the title block will fill the screen.

(6) From the top menus select "Add Text." Place the mouse cursor next to the "Designer:" annotation, click the left mouse button, and enter your initials. Accepting the entry will mark the drawing as yours. To complete the rest of the title information move the mouse cursor to each entry, click the left button, and type the entry. [NOTE: filling in the title block is not optional. You must fill it in completely. This is the very minimal documentation control that I impose. Industrial requirements are far more stringent.]

(7) Return to the full screen view by hitting the special function key <*F4*>.

(8) Again use the top menu to select "Add Component". You will still be in the parts library for en163. Move the cursor to the "74ls00.1" selection, hold down the left button on the mouse and drag the gate to the right of center on the drawing.

(9) Repeat the process selecting the "74ls569.1" chip. Place it along the same line but to the left by an inch or two from the NAND gate. If the component selection window is in the

way, drop the part anywhere convenient. Then move or close the selection window, and use the mouse to select and reposition the component. (You can move the component window out of the way by dragging a point on its top bar.)

(10) Zoom in on the two integrated circuits by putting the mouse cursor above and to the left of the 74ls569. Hit *<F9>*, drag the mouse down and to the right past the NAND gate, and release the button. Next select the 74ls569 by entering "select mode" and clicking on the 75ls569 symbol. Keeping the cursor on the symbol, press left mouse button and hold it down. Drag the symbol with the mouse until the QD pin is on the same horizontal line as the top input pin of the NAND gate.

(11) From the menus, select the command "Add Net". Put the cursor on the QD pin of the 74ls569, hold down the left mouse button, and drag the cursor over to the upper input pin of the NAND gate. A red line will appear representing a wire, part of a net. When you reach the pin, release the button. Repeat the mouse action to connect the output of the gate to the *LOAD* pin of the 74ls569. Start again with the cursor on the lower input of the NAND gate a wire left and down. Releasing the button will make a temporary break in the net for later connection.

(12) Now we will fix the reference designators of the two parts. Enter "select mode" and select the NAND with the mouse. (The shortcut to "select mode" is the "s" key.) Then type: "<Space bar>refdes U1<Enter>". The text will appear in the command line window at the bottom left edge of the schematic window. (The space bar is the key that activates the command line. Incidentally, this line is a "dockable" toolbar that can be repositioned with the mouse. Personally, I like it best where the designers put it.) The reference designator on the NAND gate will change from U? to U1. Repeat the process for the 74ls569, naming it U2. [Note: reference designators are a mandatory part of a drawing, not optional. However, you have some freedom in assigning them as long as you keep the prefix for the part and you make the designator unique to each component package.]

(13) Next we need to add another two input NAND gate. There are two ways to do this, namely returning to the component menu to retrieve another section or copying the existing gate. For variety, let us try the latter. Enter "Select Mode" again (icon L1 or keystroke "s"), and click on the first 74ls00 section already on the drawing. From the top pull-down menu, choose "Edit Copy" then "Edit Paste." Click the left mouse button to make the part appear and drag it until it is positioned somewhat above the original part. Releasing the button locks it into place. By default, its pinout will be the same as the first gate, and so it must be changed. Now suppose we wish to make this gate the one using pins 8, 9, and 10. Select the new gate with the mouse select button. From the menu, choose the command "Edit Slot". A window will open in which to type the number 3. This will change the "Slot" number to three, which is the third gate in the package. (You might find it easier to use the command line entry "<Space bar>slot 3<Enter>"to do the same thing.) Finally, change the reference designator on this part to U1 to show it is part of the same package as the first gate.

(14) Wire the inputs of this gate to QC and QD of the 74ls569 with the "Add Net" command. It may be easier to attach QC to pin 9 of U1 first. You may also have to release and repress the mouse button once as you drag the wire to make the wire bend where you want it to.

(15) Now we will label one of our lines. Enter "Select Mode" (keystroke "s"), place the mouse cursor on the top segment of the net connecting the pin 3 output of U1 to the 74ls569 synchronous load pin, and click once to select that wire. Double click the left mouse button to open the attributes dialog for the net you have selected. Type the string "CTR_RELOAD<Enter>", in the value box. This signal is actually negative true, that is, the counter resets when this signal is LOW. To indicate this on the label, check the "Inverted" box on the same dialog form. (Do NOT change the default values of visibility or scope from visible/local.) Hit OKAY to accept the label and close the window.

(16) Finally let us add a bus to the schematic and connect a wire or two to it. From the menus, select the command for "Add Bus". Put the cursor an inch above the 74ls569, hold down the left mouse button and drag the cursor to the right for about three inches. Release the button to end the segment. Label the new bus with the same procedure as for the CTR_RELOAD line, except call it "CTR_DATBUS[7:0]", implying an eight wire bus. If the label is obscured by the bus itself, you can reposition it by selecting just the label itself and dragging it with the mouse. Using the usual "Add Net" command, add a wire from QA of U2 to the bus. Label this wire "CTR_DATBUS0", meaning it is to be the lowest order bit line of the 8 bit counter bus.

(17) The net label may be awkwardly oriented. To rotate it, select just the label itself (NOT the net). From the menus, select "Edit Rotate". With the cursor on the lower left corner of the label press the execute button. You can then reposition it by dragging with the mouse.

(18) Connect nets from QB, QC, and QD to the CTR_DATBUS bus. It often happens that one has a group of lines like this attached to a bus, and they have to be labeled sequentially. Select the line extending from the bus to QB of the 74ls569. Label it the same way as for CTR_DATBUS0 except use the label string CTR_DATBUS[1:3]. The result will be a label for CTR_DATBUS1. Now select the line attached to QC, hit the right mouse button, and select "Next Label". This will label that line CTR_DATBUS2. Repeat for QD to make it CTR_DATBUS3.

(19) Now add, position, and wire the components that will be U3, R1, and VDD as shown on the drawing. Change their reference designators and label the dangling output wire. The last thing to do is to set the resistor value. Select the resistor and double click the left mouse button. This will open a dialog box with several forms. Click on the Attributes tab, select the "Value=?" line in the attributes list, and type the actual value, 1K, into the value box. Accept the change with the OKAY softbutton, and you are done drawing this schematic.

(20) Finally we store the figure that has been drawn so far. Select the "File Save+Check" command. *DxDesigner* will use the file name you specified in opening the window. Some data about the schematic will be shown either at the bottom of the screen or in a new window. It should say that there are no errors or warnings. If there are any, then the window information will include a list of warnings and errors.

(21) Now you may wish to print what you have drawn. Pull down the File menu and choose "Print Setup". Make sure that the mode is set for Landscape printing and NOT Portrait. Then again pull down the File menu and select "Print". The result will appear on one of the laser printers at the middle of the lab.

After you have used the menu commands a few times, you will realize that pulling down a menu and making selections is a cumbersome method of changing between the commonest operating modes. Depending on the command, you will want to use either the single keystroke command, the icon soft button, or the command line word entry. The correspondence of keystroke, icon, and word commands to menu selections for the commonest choices is given in Table III. More comprehensive but less accessible lists are given in the Help menu.

Table III: Correspondence of Typed Commands with Menu Choices				
Menu Path	Soft Button	Command Entry	Single Key	Function
Edit Select	L1		s	Enter Select Mode in which the left mouse button selects objects for editing – moving changing attributes, etc.
Add Comp	L2	comp	c	Put a part or symbol from a library on the drawing
Add Net	L4	net	n	Connect a wire between two or more points.
Add Bus	L5	bus	b	Add a selection of a bus to the schematic.
Add Label		label	l	Attach a label to a net or bus.
Add Box	L9	box		Draw a rectangular box; mouse sets corner positions.
Add Circle	L10	circle		Draw a circle; mouse sets center, radius.
Add Line	L11	line		Draw a straight line; NOT the same as adding a wire. (Use Add Net for wiring.)
Add Text	L12	text		Put some text on the drawing; Not the same as a label.
Edit Copy	T6	copy	<Ctrl>c	Make a duplicate of a selected object already on the screen.
Edit Cut	T5	bcut	<Ctrl>x	Cut selected section and put in buffer.
Edit Paste	T7	bpaste	<Ctrl>v	Paste buffer onto screen at cursor.
Edit Delete	B1	del	<Del> or d	Delete selected objects.
Edit Re-flect	B6/7	reflect		Mirror reflect a selected object about a line set by the mouse cursor. Softbutton (icon) method separates vertical and horizontal reflections onto two buttons.
Edit Rotate	B3	rotate		Turn an object about the cursor position by 90 degrees. The rotate command entry also requires a mouse click to activate.

Table III Continued: Correspondence of Typed Commands with Menu Choices				
Menu Path	Soft Button	Command Entry	Single Key	Function
Edit Slot		slot		Change the section of a multipart component used for a selected location. See discussion in text.
File Save + Check	T3	write	w	Store the work done on this sheet on disk. also check for errors.
File Print	T8		<Ctrl>p	Print to the laser printer. (Be sure to have the print view set to landscape and NOT portrait before printing.)
File Exit				End the DxDesigner session.

Note: <Ctrl>x means hold the Control key and simultaneously hit the next character, “x”.

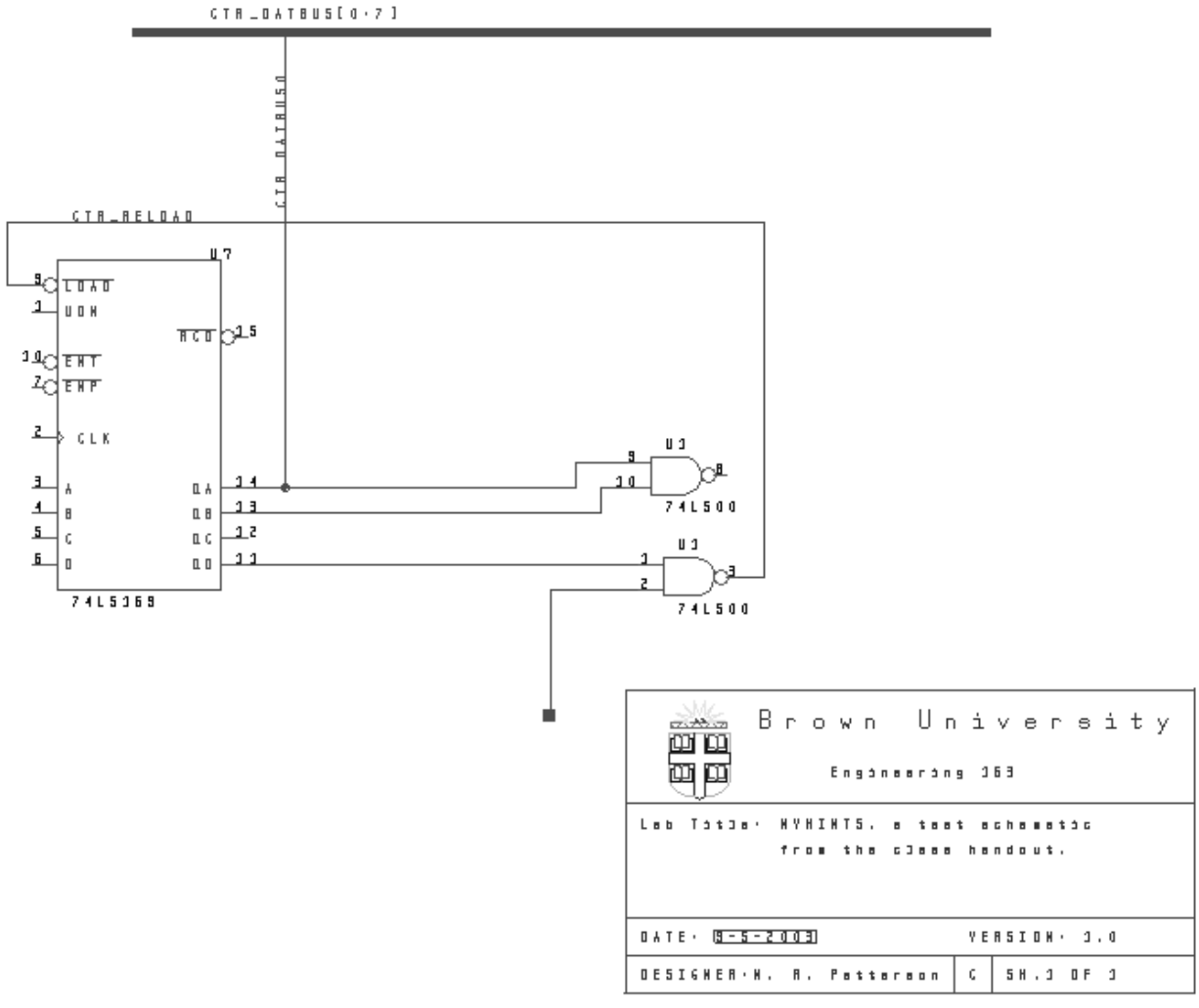


Figure 11.3: Schematic of the tutorial example.

11.4. Hints for Using Schematic Capture FPGA Software

Using *eProduct Designer* and Xilinx Alliance Software for Xilinx FPGA Design

The ultimate instructions to a RAM-based programmable gate array, such as one of the Xilinx FPGAs, have to be a bit-stream. Such a file has a proprietary format and is essentially impossible for a designer to write directly. To sell its products, the Xilinx Corporation has developed a range of design tools that take a high-level description and process it into the programming data. This translation is sufficiently complicated that the software itself has become a major part of the company's efforts and products. Among other things, the software allows one to specify designs in terms of a schematic (Lab C/D/E), a hardware description language (HDL) text (Lab F), or a mixture of the two. This note deals only with the simple schematic form of entry, but there will be an additional handout for Lab F. The newest edition of Wakerly comes with the ISE 4.2 student version of the Xilinx software on a CD-Rom and you may use that software if you wish -- but you must be prepared to show the floorplan of your chip in the lab. We have found there are problems with this software running under Windows XP, so use caution. It can convert the *DxDesigner* "edn" file into a programming "bit" file. Unfortunately, it does not do wirelist generation for our parts, which still must be done with *DxDesigner* or the Aldec/Synplicity toolset.

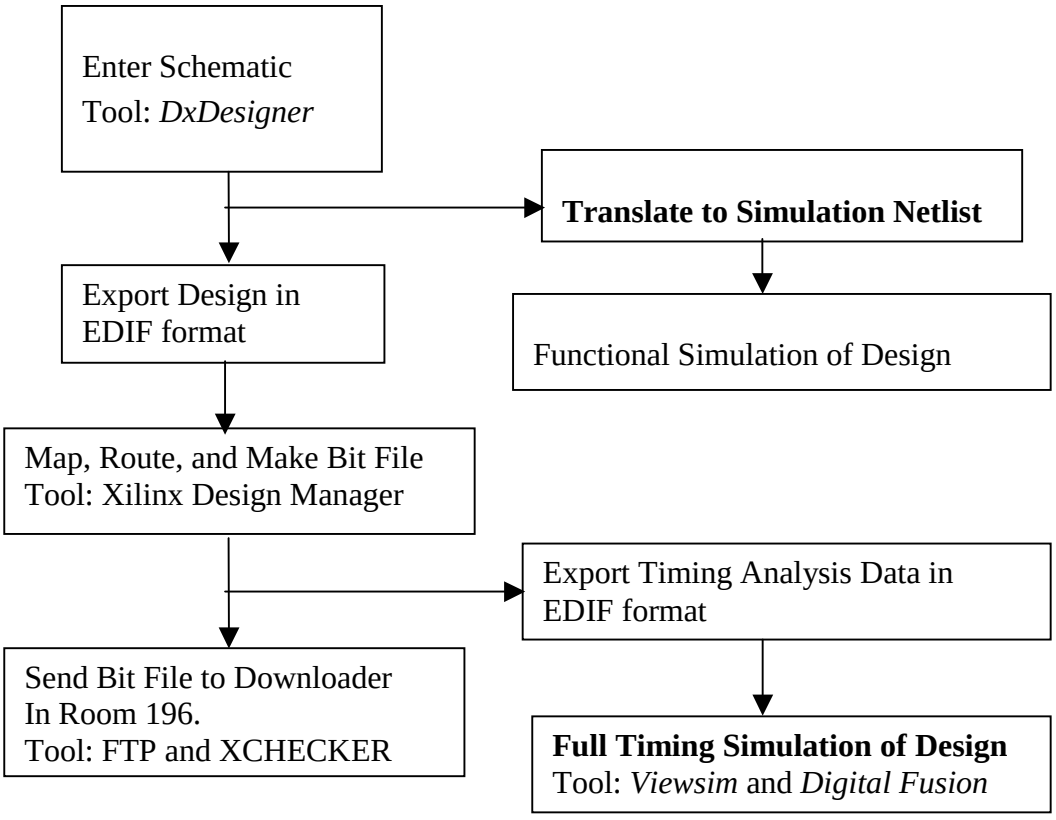


Figure 11.4-i: Flowchart of the FPGA Design Process from a Schematic.

The other tools we make available provide an interface to the *DxDesigner* schematic capture package. The nice thing about such a system is that it can target devices from any vendor (Xilinx is not the only FPGA house – they only sell roughly half the world's FPGAs) and it can give a common data entry/documentation control system for all aspects of system design. The downside is that it may be more complicated than desirable for some needs. Figure 10.4-i shows the flow of data between tools, beginning with schematic entry and going on through transfer of a bit file that configures your FPGA. (The EDIF format files referred to in the figure are ones written in an industry-standard form called Electronic Design Interchange Files format.) Along the way, one has an opportunity to do either functional simulation or full timing simulation. In a complicated design, one would do functional simulation first until the system seemed to work before bothering to route the chip, to analyze it thoroughly, or to try it. One comment about the approach here is that schematic entry for programmable design is becoming less popular. More often than not, engineers use a hardware description language (HDL), either VHDL or Verilog, for design entry and a simulation and synthesis toolset like the Aldec and Synplicity combination. I hope to arrange this for lab F.

The design process begins in *DxDesigner* with a schematic of a system that is to go into the FPGA. The "components" are generic gates, flip-flops, registers, adders, etc. from a special library set. The library also has symbols for CLB and IOB blocks that one can configure, place, and route directly, insuring that the designer can control how critical parts of the system are implemented. (CLBs are Configurable Logic Blocks, the repeating unit of logic in the Xlinx array. Similarly, IOBs are the Input Output Blocks associated with each pin; these can be configured for input, output, or bidirectional data transfer with and without register flip-flops.)

The "parts" are not, of course, real components but rather are a set of symbols that tell the Xilinx software the necessary functionality. That software assigns functions to particular Configurable Logic Blocks (CLBs), routes the necessary wiring between blocks, and converts the resulting map into a bit-stream. All this processing after the schematic entry can be completely automatic for small designs such as the ones for this class. (The Xilinx software allows for very flexible interaction in this process for more complex designs where different design specifications must be merged or where routing has to take subtle speed considerations into account. We will only use a small fraction of its capability.)

To illustrate some of the main features of the process from schematic entry to a running circuit, we will consider a simple scrambler circuit sometimes used in serial communications. This is a seven-bit serial shift register with the serial input made from the XOR of the seventh bit with a data input. The circuit has one output (bit-seven) and a clock. Figure 11.4-ii shows the finished schematic. The step-by-step procedure shows how to work with the software.

(1) The process begins with setting up a properly annotated schematic sheet in *eProduct Designer* by the same procedure used in the hints on standard schematic entry. The drawing sheets, for example, the "csheet.1" components, are the **ONLY** components that you may take from the standard library. These just contain text and graphics. All other components must be from either the "spartanxl" or the "P:\en0163\xilinx" libraries. The latter has symbols that might make lab C easier. NOTE: YOU MUST USE THE SPARTANXL

LIBRARY, NOT THE SPARTAN LIBRARY. The XL library may not be included in the DxDesigner.ini file that was copied into your wv directory early in the course. If that is the case either recopy the ini file or edit in a correction. (Turn "Spartan" into "SpartanXL" in the list of directories near the end of the file.)

(2) Use the menu command "Edit/Component," the L2 softbutton, or the keyboard command "comp" to lay down components. In this example, the first component is the output connection for the seventh bit of the shift register. The output connection requires an output pad (the "opad.1" in the spartanxl library) and an output buffer (the "obuf.1" in the spartanxl library). Place the pad toward the right side of the page and the buffer slightly to its left.

(3) Now add seven D flip-flops and wire them together and to the output buffer and pad as shown. (The flip-flop part is in the spartanxl library as "FD.1".) The details of selecting components and adding nets are exactly the same as for conventional components as given in the handout on schematic capture. A description of the functionality of the parts is in the Xilinx Libraries Guide. Pay careful attention to the compatibility of any part you want to use with the SpartanXL family of parts.

Add an XOR gate and wire it to the input and shift register as shown.

(4) All FPGAs have special provisions for distributing clock signals. As discussed in the Xilinx data book, the SpartanXL family has four metal column lines for each column of the array that are dedicated to clock distribution. These can be driven by any of eight specialized global clock drivers at the corners of the chip. A driver network of one global driver (called a "BUFGPS" in the library) and a set of these metal lines forms a low-skew clocking net. You must use such a network for all clocks if multiple clocked flip-flops are to be reliable. Note that if a global clock driver gets its input from an external pin, that pin should be one of the dedicated pins for that purpose. To implement the clock network for this scrambler, add an input pad (the "ipad.1" component) and a BUFGPS clock driver to the schematic. Connect them together and use the BUFGPS output to drive the clock lines of the register.

(5) Next, assign pin numbers to the input and output pins of the XCS05XL. This is done by adding an attribute called "LOC" along with the desired pin number as its value to each pad. In select mode, double click on an input or output pad with the mouse. In the dialog box, select the attributes tab. Type "LOC" in the Name box and the string "Pnn" in the value box where <nn> is the desired pin number. Hit the Set and OK buttons. If the LOC attribute is in an awkward place on the drawing, select just the attribute with the mouse and drag the LOC=Pxx label next to the pin. Repeat this process for all pads. [The designers at Xilinx recommend that you do not usually assign pin numbers, because that makes it more difficult for their software to squeeze a complicated design into a small chip. Leaving the placement software more latitude results in faster and denser designs. However, often when a design has to fit a preexisting socket, as is the case here, one has to make such assignments. For small designs like ours, there is little difference in the final outcome.]

(6) Label all the important nets, particularly anything you may want to refer to in simulation. Labels on the lines between pads and buffers become the signal names at the schematic level, so all those nets should have explicit labels.

(7) At this point, you may do a functional simulation of your design. The standard process is to use the *Viewsim* tool to generate a netlist and to use *Digital*Fusion to do the simulation with your own command file. [This is not required, but if you should have a problem with a lab, this is a good way to find it.]

(8) The next step is to translate the design from the *DxDesigner* schematic file into a Xilinx bit stream. Begin by executing a File/Write+Check command to store the final design and to check for gross errors. Then from the Tools pull-down menu, select Tools/Export EDIF Design. [EDIF means Electronic Data Interchange Format and is an industry standard for exchanging electronic design data between tools from different vendors.] A window with multiple tabs will open, and you select the "Netlist Writer" tab to indicate that you want to write a file with your circuit interconnections to the Xilinx toolset. On the window form, you must also select your design file as the input (you can type in the file name or browse to it with the Browse button) and you **must set the LEVEL box to Xilinx M1**. (The LEVEL box is in the middle of the window and determines how much of the design hierarchy is written out. The Xilinx M1 choice is saying you only want the Xilinx components connected and not any simulation-only schematic that may also be in the library.) Hit the APPLY button and see if the system processes your design without errors or serious warnings.

(9) Now you leave Mentor Graphics tools to invoke the Xilinx proprietary translation software. The **BUXUSP-II device can only be compiled in the Xilinx ISE 4.2 suite but must be programmed with the ISE 6.2 suite.** (See web site handout on *Choosing Versions of Xilinx Software*.) From the Windows Start menu, pick the sequence: Start/Electrical/Xilinx/**Xilinx ISE 4**/Run Me First. This sets up the Version 4.2 of the software. Then open the compiler with the sequence /Start/Electrical/Xilinx/Xilinx ISE 4/Project Navigator. This command starts the Xilinx *Project Navigator* software, which takes a moment to initialize.

(10) From the File pull-down menu, choose New Project. In the dialog box, choose the SpartanXL family and the XCS05XL-4PC84 device, and accept the "edn" design flow. Enter a name for your project, and modify the name for the directory in which the work will be stored to something convenient for you. (Please do not leave working files on our D: drive.) Now use the mouse to select the "XCS05XL-4PC84" line in the "Sources in Project" window (upper left) of *Project Navigator*. Three options will pop up in the "Processes for Current Source" window - Translate, Implementation, and Generate Programming File. Run each of these in sequence by double-clicking on it. If an error appears among the reports in the lowest window, it is necessary to go back to eProduct Designer to fix the problem. Check the Xilinx reports for the nature of the problem before returning to *DxDesigner*.

(11) Before leaving the *Project Navigator*, you might want to see what your system actually looks like on the FPGA chip. (The term "might want to see" suggests more latitude than

you actually have!) Xilinx provides a way to visualize and edit the connections through their *FPGA Editor*. Invoke it from the /Start/Programs/Electrical/Xilinx ISE 4.2 /Accessories menu sequence. A floorplan of your chip showing the routing of connections and the logical function maps will appear. You navigate around the picture with the usual zoom and pan commands found under the View pull-down menu. If you double-click on a CLB, you get a picture of how it is configured. To be sure that you have seen this representation of what is going on, **the TAs will probably (I hope) ask you to bring this view up** on one of the machines in Room 196 before checking off lab C.

(12) The final step before testing the circuit is to download the bitstream file to the device. Before closing *Project Navigator*, be sure you know where the file <your_project>.bit is located on your U:\ drive.

(13) In room 196, select a computer with a BUXUSP board connected to it. There are three cables to be concerned with: a 37 wire cable to the back of the PC, a small gray ribbon cable from the Xilinx Parallel IV programming cable to the 14 pin 2mm connector on the BUXUSP board for configuring the FPGA, and the two wire power cable that plugs into an adjacent power supply. If this is lab D, E, or F, you will need the appropriate board connected to the User Cable from the BUXUSP-II board too.

(14) Use the menu sequence: /Start/Electrical/Xilinx/Xilinx ISE 6/Run Me First to return to the version 6.2 of the software. Then invoke: /Start/Electrical/Xilinx/Xilinx ISE 6.2 /Accessories/iImpact to start the same downloading software as is used for the CPLD boards. Accept the default "Configure Device" choice in the first dialog window but **change the default choice to "Slave Serial Mode" for the second window** that asks about the download mode.

(15) In the download file dialog, choose your <your_project>.bit file. Right click on the FPGA icon in the middle of the screen and select "Program" with the left button. The device programs almost instantly with the usual Programmed Successfully message.

(16) Test programs for labs C, D, and F are in the D:\Test Software directory.

If the download is successful, an LED in the light bar on the BUXUSP-II board will go on. The circuit is now ready for testing. It is, of course, only possible to download from the machines in Room 196 that have BUXUSP-II boards attached to the Xilinx Parallel-IV cables – the same programming pod but a smaller ribbon cable link than is used with the CPLD boards. Directions for running test programs for each lab will be posted in the lab.

Discussion: Finding parts for your designs can be done either by browsing through the spartanxl or en0163/xilinx libraries or by looking at online documentation. That data is available through the Help pull-down menu of the Xilinx Project Manager. [Use the Start/Programs/Electrical/Xilinx ISE 4.2/Project Manager/Help/Online Documentation menu sequence. In the documentation viewer, go to the Xilinx Answer Books/Libraries Guide.] Please pay particular attention to the compatibility of any part you wish to use with the SpartanXL family of devices.

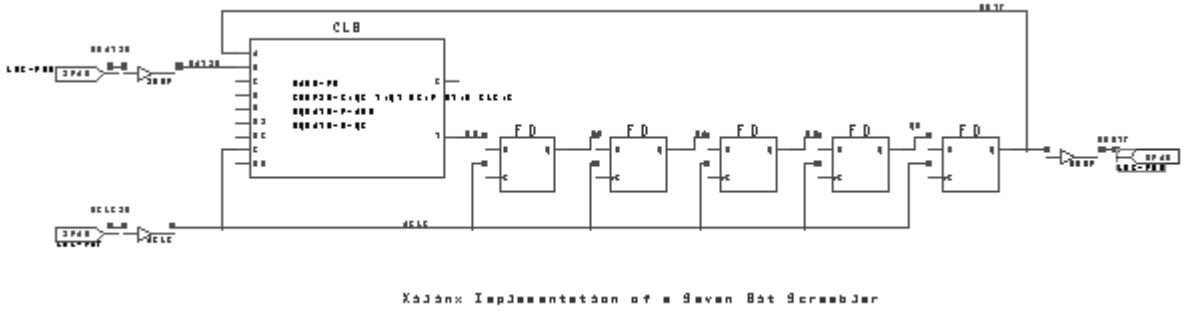


Figure 11.4-ii: Tutorial Example of FPGA Design -- 7-Bit Scrambler.

11.5. Using the BUXUSP-II Boards

BUXUSP-II: Brown University/Xilinx
Universal Small-scale Prototyping Board – the Second Generation

BUXUSP-II is a printed circuit board with a Xilinx XCS05XL field programmable gate array on it that makes it easy to connect such a chip to other components and to test the resulting logic system from a PC. The design philosophy behind BUXUSP-II began with the premise that the use of FPGAs should be taught only in the context of embedding them in systems rather than using them as independent, self-contained units. It now serves as the basis for the last four labs of Engin 163. The types of lab projects that one might use it for are typified by the DRAM controller of Lab D. With the DRAM controller, the Xilinx array connects to a DRAM memory chip on a protoboard and handles the tasks of address and data multiplexing, memory refresh, and bus handshaking as a PC simulates a memory bus over which it reads and writes the DRAM contents.

The schematic for the system is shown in full as an example in section 11.1 on schematic documentation. This is a second foldout page of the manual and therefore pretty much has to be used in paper format. The first page of the schematic shows the allocation of connections to the 60 pin cable for student circuits. The connector for that cable is P4. The list of features given below describes the resources available through this connector.

Main Features of BUXUSP-II:

Physical link to the user: a 60 conductor ribbon cable from P1 connects the BUXUSP-II board to your hardware. If your components are on solderless breadboard, then the ribbon cable terminates in a wire-wrap header. Bending the pins on this header allows it to plug directly into a piece of the same breadboard. If the project is on wire-wrap itself, then one wires a suitable header into the circuit directly. In both cases the interconnection is durable and able to withstand substantial abuse. The BUXUSP-II board can also be easily disconnected to free it for other use while repairs or redesign are done on the user's parts.

BUXUSP-II has two places for a crystal oscillator clock source. One is a permanent 1.843 MHz oscillator and the second is a socket for standard 14-pin DIP oscillators. A jumper block allows choosing between these.

There are several lines dedicated to clock signals for the FPGA. Two of the PC connections (PORTC0 and PORTC2) and one user cable signal have RC terminating networks and Schmitt trigger buffers that assure clean clock signals. MANCLK and the on-board crystal oscillator attach to pins that can access global clock networks too.

24 Input-Output pins from the XCS05XL are routed to a connector that is compatible with a widely used parallel interface for the PC. These lines are for testing systems and for displaying results through the PC. All these lines are also available on the cable to the user's system.

23 Input-Output pins are dedicated strictly for connections between the XCS05XL and the user's system with no other role.

BUXUSP-II has a miniature toggle switch that is connected to produce a completely debounced manual clock. Its output goes to an LED display and to the user cable. It is intended for single stepping clocked systems or wherever a very slow clock would be useful. The signal is called MANCLK and is found on the lower left side of the second sheet of the schematic.

There is a two digit, seven-segment, LED display that has current limiting resistors connecting its segment pins to pins of the FPGA. The cathodes connect to TTL devices that provide digit selection and on/off control by pins on the user cable.

There is a four bar, LED display for diagnostics. The first LED shows VCC status, the second FPGA programming status, the third the manual clock state, and the last shows the state of a user line. Unfortunately, this device is not easy to see clearly which segments are lit.

The RESET pushbutton clears all flip-flops in the XCS05XL.

12. Component Data Index

Manufacturers' Device Data Sheets

We have appended manufacturers' data sheets for the active devices in your Bag-of-Chips. They are ordered by type of device (e.g. TTL, CMOS, etc) and within each group the devices are ordered by type number. The groupings are as follows:

1. 74' series TTL, LVTTL, and ACT logic.
2. Xilinx CPLD and FPGA pinouts and overview.
3. CMOS RAM and DRAM memories.
4. Analog integrated circuits and switches (MC14066B, LM311, LF353, NE555).
5. Selected passive components.