(3)

Hardware Interface and Casing for Electric Legos Kit

by

Hyung S. Chang

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 22, 1997

Author_____
Department of Electrical Engineering and Computer Science
May 22,1997

Certified by_____
Professor Gill Pratt

Accepted by_____
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Hardware Interface and Casing for Electric Legos Kit
by
Hyung S. Chang

Submitted to the
Department of Electrical Engineering and Computer Science

May 22, 1997

In Partial Fulfillment of the Requirements of the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Currently, there is a need to replace the outdated lab kits being used in several lab classes here at MIT. A new design using Xilinx FPGAs in a scalable interconnect network has been created for 6.004 (Computation Structures) and is still undergoing a few design and implemtation.changes. Previously, in my AUP, the skeletal "Electric Lego" block was created allowing several FPGAs to interconnect. For this thesis, a protoboard was designed as a base for the lego blocks and as a path for programming them. The protoboard uses the parallel port and a simple request/acknowledge protocol designed by Andrew Huang to communicate to the computer that the students use to draw their design, and has its own FPGA to handle communications to the lego blocks. I also cover the design and simulation software (ActiveCAD by Aldec) used by the students in experimental sections for 6.004 this semester (for which I was a TA this semester and worked directly with the students).

Thesis Supervisor: Gill Pratt
Title: Assistant Professor, MIT EECS

Acknowledgements

First and foremost, I would like to thank Professor Gill Pratt for giving me this opportunity to work on a project that will directly influence people, and for putting up with me over the years. I also need to mention Andrew Huang in these opening sentences for all of his work towards my thesis, without which it may never have gotten done. And to everyone who has worked on this project and helped me in one way or another: Randy Sargent, Anne Wright, Carl Witty, Clive Bolton, Lisa Kozsdiy, Jay Grabeklis, James Clark, Holly Gates, Professor Don Troxel, John Sweeney.

Next I would like to thank all of the people who have helped me get through the whole college experience. My parents who paid for all those years of partying. My sisters whom I missed dearly since I came to college. All of my brothers at Beta Theta Pi who stood by me whenever I needed them. And Anne Hunter, without whom I may never have graduated.

Finally, this section would not be complete without all of the students using the new lab kits in the experimental sections of 6.004 this semester. They went through a lot of trouble as our guinea pigs, using buggy software and buggy labs, spending hours on end trying to get these things to work while quietly cursing us. Thanks for all the feedback and help!

Table of Contents

1. Introduction

Many lab classes here at MIT rely on an outdated technology to give students an on-hand experience of the topics they teach. By doing so, the lessons are ineffective at best and often times lead to wasted time and frustration on behalf of the student. The causes of this ineffectiveness vary from worn-out parts to archaic methods that can be handled much more effectively to technology that has not changed with the course curriculum. To counter this, a new lab kit has been designed for 6.004 (Computation Structures) under the supervision of Gill Pratt.

In 6.004, the degredation of the current technology can be seen in both the mechanical implementation as well as the circuitboards and integrated circuit (IC) chips used. The current lab kit is based on a proto-board technology that many other lab classes at MIT use. I will differentiate proto-board from the printed circuitboard we designed by the use of a hyphen. A proto-board is a plastic plate with a grid of metal strips under a grid of sockets, with four or five sockets per strip. Each strip is then a node where several wires can be connected. The benefit of using proto-boards to construct circuits is that they

provide the student with a physical representation of the circuit that is easy to wire and debug. Unfortunately, as these proto-boards are used year after year, the contacts in the sockets wear out. This combined with bits of wire that break off in the sockets cause inconsistent behavior in the circuit wired on them.

In addition to the proto-boards, pre-made circuitboards and IC chips also tend to wear out and break down. Since a completely wired kit in 6.004 is pretty complicated, this kind of a bug can take several hours to discover, which is frustrating to the student when he discovers the flaw was not in his design.

Another disadvantage of using proto-boards is the fact that the circuit must be hand wired. While this can be beneficial in small designs where it is easy for the students to see their design and debug it; for a complicated design, it takes several hours of work to wire the design and debugging a mess of wires is time-consuming and rarely educational. In fact, this practice is rarely used in industry where most designs are built and simulated on a computer before it is implemented in hardware.

Finally, the design that the student actually wires on his kit is not the design being taught in class. The current kits were made about fifteen years ago and though the technology was cutting edge at the time, it was replaced long ago in industry. The curriculum of the class, being more flexible, has changed with technology and is now teaching a simplified form the DEC Alpha machine, a RISC architecture, known as Beta, as a base for modern computers. However, the circuit wired on the kit implements a nano-instruction architecture called the Maybe machine. As a result, the Beta is implemented on top of the Maybe machine with all of the conversion done in software, which is invisible to the student in software. With this method, the students can see their code run on a machine that

they built, but they rarely understand how the Maybe machine that they just wired works. Nor is it the point of the labs for them to understand this outdated technology.

Currently, using old, unreliable kits and technology that has long ago been abandoned, the students are wiring a computer that runs at 1KHz, a far cry from the 200MHz computers being sold today. Also, with architecture now moving toward parallel machines, it is impossible to imagine proto-board technology being of any use in teaching about computation structures in the future. The new kits that are being designed hope to remove all of these weaknesses.

These new kits are based on a proposal originally presented by Gill Pratt[1] in 1995 dubbed "Electric Legos", where the proto-board technology is replaced by a time-multiplexed, scalable interconect of reconfigurable devices, similar to Professor Ward's NuMesh[2]. The students will do their design and simulation on a computer before it is implemented in hardware, much like industry today, with most of their time spent on design and understanding rather than mindless wiring. In addition, the design that is implemented is the same as the Beta architecture that is being taught in class. These kits have been tested through two semesters of experimental recitations and found the project to be possible, but it is still constantly undergoing changes.

The project itself involves several parts that have been divided up among several different people. The base of the technology, which will be discussed in depth in this thesis, is the physical kit on which the designs are implemented. These kits currently use Xilinx FPGAs (Field Programmable Gate Arrays)[3] as the reconfigurable device from Gill Pratt's proposal and is divided into two main components: the interconnecting lego block, and the protoboard which communicates between the blocks and the computer that holds

the students' designs. Details on the interconnecting lego block, which is simply a skeletal board for the FPGA, can be found in my Advanced Undergraduate Project[4]. This thesis involves the design, evolution, and testing of the protoboard and casing.

In addition to this hardware base, several people are working on the design platform and configuration aspects. The experimental sections this semester used tools provided by Xilinx (Xilinx Foundation Tools) for design, simulation, and compiling. Unfortunately, their tools have two major drawbacks: they are designed for optimization and they do not support incremental design changes. Because they are designed for optimization, compiling a design as complicated as the Beta can take three to four hours on a 200MHz Pentium PC. However, since we are using a scalable structure, and since we are not interested in optimized implementation, the time spent by the Xilinx tools is unecessary. Also, once the design has been compiled, any changes to the design, whether in the schematic entry or anywhere along the compiling process, requires the design to be recompiled from scratch. We hope to replace these tools with a faster compiler that supports incremental design changes, which only recompiles the parts of the design that have been affected by the change.

The compilation process involves three major phases: schematic capture, placer, and router. The Xilinx tools for the schematic capture and the simulator (ActiveCAD by Aldec) were robust and simple enough (with a few minor bugs) that we will continue to use them. A new placer and placer-router interface has been designed by Jay Grabeklis[5] that allows the students to interactively place their own parts while supporting incremental changes for bad placement choices. A new router is being created by Randy Sargent and Carl Witty that will hopefully remove more than 80% of the compile time by removing the

optimization. Unfortunately, the router has not been completed yet, so neither program was tested this semester.

Once complete, the new kits can be used by other classes due to its versatility. Some of the classes here at MIT that may benefit from this are 6.111 (Introduction to Digital Systems Laboratory), 6.115 (Microcomputer Project Laboratory), 6.313 (Contemporary Computer Design), and even 6.823 (Computer System Architecture) where students can implement some of the concepts they learn about DLX[6]. Also, specialized lego blocks can be constructed for specialized purposes such as a cache, or interfaces to PCMCIA cards and the Ethernet. By doing so, the students get more of a feeling that they built a computer in the modern day sense.

2. Goals

In short, the goal of this project is to provide an easy-to-use, modern platform for the students to implement designs they learn in 6.004. The original design by Gill Pratt, on which this design is based, is as follows. The students would work on an integrated "virtual workbench" where they will design, simulate, and debug, using software that is completely integrated. Changes made to a circuit have immediate visual consequences on the concurrently executing simulation. As the future of 6.004 evolves toward the exploration of more general high-performance digital system (such as parallel activity), simulations will not be enough, and a hardware substrate capable of implementing complex general purpose digital systems at high speeds is necesary. The hardware substrate will

contain three parts: a small protoboard area; an internal micro-computer for wave-form generation, oscilloscope functions, and logic tracing; and a time-multiplexed, scalable interconnect for prototyping complex hardware systems (dubbed "Electric Legos").

The lego block (as I will refer to them throughout the thesis) will be a small circuitboard with 4 or 6 connectors and can be combined to create a three dimensional network. Communication will be dictated by a static schedule or a "choreograph" contained in a timing RAM within a router on each module. And the clocking will be based on a single physical clock with each lego block running at some "virtual clock", which is a multiple of the physical clock. These were the original designs of Gill Pratt in 1995 and have since changed considerably. I will discuss in the next section how each goal was implemented or changed in the current form of the new kit without losing sight of the goals.

3. Design Implementation and Changes

As stated previously, the original design has changed considerably. In fact, the design has changed several times since then, but I will focus on only one mid-design phase which I call Anne's phase after Anne Wright, who did most of the hardware work before I became involved. In addition, this section will cover some minor decisions that were too intricate to discuss in the original proposal, such as the power supply.

There seem to be three major categories that these designs and decisions fall into, with each category fanning out into more detailed categories: *Inter-Lego Communication, Kit communication, and Component Choices. Inter-Lego Communication* discusses the

clocking scheme, individual block configuration, and block to block communication. This section deals less with the scope of this thesis, but is important in the view of the hardware section as a whole. *Kit Communication* involves the communication between the kit and the computer as well as the communication between the kit and the lego blocks. Finally, the *Component Choices* section is a catch-all section for design choices that did not fit into either of the two categories above.

3.1 Inter-Lego Communication

This section will cover the design changes and implementations involving the communication between lego blocks. Though this thesis pertains solely to the main kit, these design decisions are important in seeing the hardware as a whole unit, as well as influencing the design of the kit itself.

The communication of the lego block, in general, seems to fall into four categories: physical link, clocking, configuration, and data communication. The original proposal describes this section in detail compared to other sections of the project, covering three of these four categories, configuration is not mentioned. By Anne's phase, all of the categories are covered in much more detail and very much resembles the final design of the lego block.

Gill Pratt's Proposal

There will be four to six connectors (half on top side, half on the bottom), that allow us to build a three dimensional network. If four connectors are used, square modules will be connected in the topology of a diamond, while six connectors will use hexag-

onal modules creating a Cartesian mesh.

As for the clocking, it will be accomplished in a distributed fashion by synchronizing neighboring oscillators. With all of the oscillators running some high physical clock rate, each the lego block creates its own "virtual clock" depending on its individual need, where a "virtual clock" is some multiple of the physical clock rate. He anticipates a physical clock rate of 200MHz with the virtual clock using less than 10 physical clocks. He also discusses the possibility of an asynchronous system where multi-rate sampling can be used to communicate between blocks.

The data communication will be dictated by a static schedule or a "choreograph" contained in a timing RAM within a router on each chip. This means that each lego block will have its own routing circuitry and run similar to NuMesh, with the routing data constructed as the design is compiled. This also means that the routing maps are static and cannot be reconfigured during operation. Finally, the data can be received from multiple sources during different physical clock cycles and then assembled.

Anne's Phase

The number of connectors per block have been finalized to four with the pincount and type yet undetermined. The guidelines at this point were a mechanically durable, compact structure with enough pins to support a 32 bit data bus.

The clocking scheme is mostly unchanged Using a synchronization circuit developed by Gill Pratt and John Nguyen[7], adjacent lego blocks should have phase errors of less than 0.1% of the cycle. The virtual clock is no longer used, which means that the

physical clock must match the slowest process of the design, and the asynchronous system was also discarded due to its complexity.

Configuration of the lego blocks is handled by the kit, programming each lego block as a data pipe until the edges of the network are reached. The configuration data for each lego block is then loaded, programming the blocks on the outside of the network first and moving towards the origin.

Using FPGAs, there seems to be no need for each lego block to have a router, since the design, though spread over several chips, is actually one unit as opposed to several smaller units passing messages to each other. The data is simply passed from the pin of one FPGA to the pin of an adjacent FPGA, which is adequate as long as the clocks remain synchronized.

3.1.1 Physical Link

The evolution of the physical link throughout this project is that it only gets more clearly defined at each phase. We began with Gill Pratt's proposal that only mentioned the number of connectors desired, then to Anne's phase that narrowed the number desired to four and listed some qualities we desired. Due to the design of the FPGA, there was also a reason to distinguish the North connector from the South connector, and the East from the West, resulting in limited connection possibilites (see my AUP for details). This meant that we also needed a connector that could only connect in one orientation.

We found most of these qualities in a 3x16 connector from AMP. It is a solid connector capable of 1000 or more reconnects, with only one orientation, and keeps the size of the lego block down to 3.75 inches square. Unfortunately, any connector with more

than 48 pins was either too large or too weak, so having chosen this connector, we had to limit the data pins to 29.

We had also tossed around the idea of getting a mechanical engineer to design some sort of an ejector to pull the boards apart, to reduce the wear from students who try to pull them out at odd angles. However, after testing the connectors on the lego blocks, we discarded the idea because the boards were fairly simple to pull apart.

3.1.2 Clock

Gill Pratt's proposal mentioned two specific schemes for clocking: an asynchronous method with multi-rate sampling for communication, and a synchronized system where each lego block takes the global physical clock rate and runs its own virtual clock rate as a multiple of the physical one. Neither of these concepts made it as far as Anne's phase, which is in fact the method being currently used.

By Anne's phase, we use a much simpler, sub-optimal scheme that uses just the global physical clock. It uses a synchronization circuitry designed by Gill Pratt, keeping the phase error below 0.1% of the clock cycle between adjacent lego blocks. However, since each block does not create its own virtual clock, the entire system must run at a rate slow enough to accomodate the slowest part.

If desired, virtual clocking could be added later in hardware, but due to its complexity, it seems more beneficial to divide the clock inside the FPGA with the design than to try to add it in hardware. Using the single physical clock method, the Beta designed this semester used an 8MHz clock.

3.1.3 Configuration

Gill Pratt's proposal did not mention how each of the configurable blocks would be configured. By Anne's phase, all of the intermediate nodes in the network were configured as until the outermost nodes were reached, where each node is a lego block. Once all of the nodes have been accessed, configuration would begin, starting with the outermost nodes and slowly replacing the pipe configuration with the actual data for that node. This method was tested with positive results during Anne's phase using the ISA board, (read the section on Kit Communication); however, multi-node landscapes have not been tested with the current implementation.

Our current implementation of the single node system is to load the FPGA with a function capable of giving the PC access to its SRAM called "kitcomm" and designed by Andrew Huang. With this access, we load the instructions onto the SRAM. Then we overwrite the FPGA with the Beta machine and let it run, leaving the result in the SRAM. Finally, we reload kitcomm to read the SRAM and check the result. The reason we have only tested this method, is that until our versions of the placer and router are created, it is difficult to implement designs that bridge boards.

3.1.4 Data Communications

In Gill Pratt's proposal, he talks about a static schedule or a timing RAM that controls data flow. However, the Beta machine to be implemented in 6.004 is a single unit that may be spread over multiple lego boards. Due to the large number of data lines and the odd size of the bus on the connector (29 pins), some information will have to be sent in smaller packets and reassembled by the receiver. This kind of design does not require a

complicated message passing system. In fact, there are few lab classes at MIT that could take advantage of such a system. While it is true that the focus of computers is shifting to a distributed network, and the class may change to follow it, the routing system may best be left to be implemented by the student.

Therefore, in the current model of the new kit, which was the model during Anne's phase, data pins for adjacent FPGAs are directly connected through the connector and the responsibility of routing the data to its destination lies with the designer. Unfortunately, as I had mentioned before, we have not set up such a multi-node design to see if it would work. Instead, a similar communication link has been set up and tested by Andrew Huang to communicate to the on-board SRAM which uses an 8-bit data bus. Since the Beta is a 32-bit machine, the data is divided into 4 bytes and each byte is stored individually. In addition, when reading the data, the FPGA reads four consecutive bytes and reassembles them before sending it to the Beta. A similar system could be used between the lego blocks, with sixteen bits used for data and other bitsuised to control its flow. See Appendix A for a schematic implmentation of the communication with the SRAM.

3.2 Kit Communication

This is the section that has undergone the most amount of change since the start of this project. In Gill Pratt's proposal, he envisions the kit as a stand-alone microcomputer with the capability of programming the lego block, as well as performing oscilloscope and wave-generation functions. By connecting to a computer via either the serial or parallel port, it would be able to download configuration data and display its oscilloscope output.

By the time Anne's phase came around, the oscilloscope functions had been

dropped, due to time constraints and complexity, but the implementation of a microcomputer was still around. In fact, a working system had been built using the ISA bus to interface to the lego block, including all of the necessary software. A protoboard on which the lego block mounts had been wired up on a proto-board and an old 386 in the office was used as the motherboard. When the lego block had been developed, this setup was successfully tested.

However, the microcomputer concept was dropped eventually due to several factors: time, size, and cost. In order to implement the microcomputer, we would need to obtain a motherboard for each kit, and design an ISA card as well as the protoboard. The time frame we were working with made it difficult to design both of the boards, and all of those boards would have made the kit unreasonably large and heavy. However, the decision to drop the microcomputer was made when Intel refused to donate the motherboards and we could not find another sponsor. The motherboard would have cost several hundred dollars more for each kit. The combination of these factors forced us to remove the microcomputer from the kit and transfer the responsibility of programming the lego block to the protoboard.

The communication resposibilites of the kit can be reduced to communication with the computer and communication with the lego blocks. All of the other internal aspects of the protoboard will be discussed in the next section: *The Current Protoboard.*

3.2.1 Communication with the Computer

Due to the simplification of the kit, we limited the communication between the PC and the computer to be handled over the parallel port. This port is ubiquitous on most

modern computers and it contains a wider data bus than the serial port making the protoboard design easier.

Next, we had to chose a bi-directional communication protocol for the port. To save time and trouble, we started by looking for an existing bi-directional protocol.

Some research led to two reasonable protocols[8]: the ECP and EPP bi-directional parallel port protocols. Both protocols work well in transferring and receiving data and the difference between the two protocols was minimal for our purposes. As a result, we opted for the EPP protocol which seems slightly better at handling more complicated back and forth traffic than the ECP which was optimally designed for passing large amounts of data in one direction. The other benefit of EPP over ECP is that it was designed to communicate with several receivers at the same time, addressing each one as it delivers data. In somewhat complicated designs in the labs at MIT, the likelihood of this kind of setup is fairly high.

As it turns out, the communication between the kit and the computer is so simple, usually no more than a few thousand bytes at a time, that a simple request/acknowledge protocol is all we needed. This new protocol was developed in detail by Andrew Huang and the driver information can be seen in Appendix C. Unfortunately, the kitcomm design is not available in any readable format.

In order to keep flexibility in our protocols, the only things that need to be replaced when using a different parallel port protocol are the drivers, the design in the protoboard FPGA, and a single PAL on the protoboard through which all the port signals pass. The purpose and use of the PAL will be discussed in more detail in the section *The Current Protoboard: Parallel Port.*

3.2.2 Communication with the Lego Blocks

The communication from the kit to the lego boards was much easier to handle. Since protocols and a design already existed which allowed one FPGA to program an adjacent FPGA, we simply mounted an FPGA onto the protoboard whose function is to program the lego blocks in the same manner. Therefore, the connectors on the protoboard are identical to those on the lego blocks.

3.3 Component Choices

Aside from the component choices made for communications earlier, there were several other components and parts that were chosen under strict guidelines. Those components are listed below with the guidelines and explanations supporting its choice.

3.3.1 Fan

We needed a small fan to be mounted under the protoboard. The guidelines for this fan were a small compact size, low power consumption, and low cost. The final decision was to use two 1.6" x 1.6" x 0.75" fans by Elina that costs $7.95 and takes 1.9W of power each.

3.3.2 Power Supply

Oddly enough, the power supply guidelines underwent several changes since Gill Pratt's proposal. In his proposal, he mentioned a 15 V linear power supply to handle some of the earlier labs that use the small proto-board area, and then as a source for any

design that may need it later (RS232 communication?). This remained unchanged until after Anne's phase.

Rather than design our own power supply from scratch, as it was done on the current 6.111 kits, we wanted a smaller, more reliable supply that is out in the market these days. This led to a frenzied search of reasonable power supplies, meaning samll and cheap, with both +5V and 15V and enough power to drive up to 7 fully loaded lego blocks. (7 boards was chosen because that is the number of blocks in a simple three tiered structure with power connections to all of the lego boards directly from the protoboard.) This meant that with each lego board capable of 3W of consumption (much of it from the LEDs), we needed 21W. When we consider the protoboard, the fan, and a safety factor of two, we were looking at about 50W.

Due to market trends and cost considerations, we changed our power supply to two separate units: a +5V switching power supply capable of 60W and a 12V linear power supply with only 6W of power. This division provided the required power and cost guidelines, and there was enough room in the kit for both of the power supplies to lie under the protoboard.

However, our final design only uses the +5V 60W switching power supply because the 12V supply weighing in at 3 pounds, made the kits too heavy. It may be worth noting here that there is still plenty of room for the 12V supply if anyone is interested in the option.

3.3.3 SRAM

The SRAM chip that is provided on every lego block and the protoboard was first introduced at Anne's phase. A version of it used for routing was included in Gill Pratt's proposal, but this one is connected directly to the FPGA and can be used for any general purpose.

The SRAM underwent a minor change recently after testing the lego blocks for one of the 6.004 labs. Our original SRAM was a PSRAM (Psuedo SRAM) which is actually a DRAM with its own internal refresher. It was supposed to behave like an SRAM. Unfortunately, though we do not have to handle the refresh, it takes a cycle during which access is denied. This timing constraint made any interface on the FPGA more complicated. Therefore, though it was originally chosen for its low cost, we went back to the regular SRAM for simplicity.

3.3.4 Case

With a new kit, we want a new case specially designed for the new boards. Origianlly, I looked into making a wooden case similar to those used in 6.111. In the process, we hired Holly Gates to design and manufacture a case for us. Out guidelines were a sturdy, lightweight case that is waterproof, so that the students can carry them through the rain, then plug them in right away.

Holly returned with a lightweight (plastic) waterproof pre-made case that was great for the kit. It was made with a material that he could chemically glue edges to and the edge is bonded as though it were one piece with the kit.

As an added bonus, the cases have a newer sleeker look that makes it seem more

contemporary.

4. The Current Protoboard

The protoboard that has been designed and manufactured is discussed in detail in this section. There are four major design aspects of the protoboard as follows: the physical design of the protoboard, the layout of the components, the parallel port, and the lego block base.

4.1 Physical Design of the Protoboard

Though it may not influence the functionality of the kit, the physical design of the protoboard will affect the size and layout greatly. The two most important aspects of the protoboard were its size and the it contained all of the necessary section. The necessary section of the protoboard are: a proto-board area where the students can wire up the first couple of labs to get a hands-on feeling of physically wiring circuits; a display and input area full of LEDs and switches so that the students can debug their circuits or add interfaces to their later designs; a lego block base where the lego blocks can be mounted and programmed; and room for two connectors -- a parallel and generic 50-pin ports.

Due to the size issue, a couple of the ideas we were throwing around were a folding protoboard and a removable lego block base connected by a ribbon cable to the protoboard. Both were possible, but created mechanical weaknesses that would definitely cause

problems as they wore out.

Luckily, a suitable large case was found (see Design Implementation and Changes: Component Choices: Case) and a single printed circuitboard was used. This also created more room underneath for a second power supply if desired. The final dimensions of the protoboard are 17" x 11".

4.2 Layout of Components

The layout of the protoboard involves placement of the necessary sections mentioned earlier and the routing of the actual board. For the proto-boards, we took a very traditional approach and placed power strips next to each protoboard, then placed all of the 12 switches (8 toggle and 4 pushbutton) and 8 LEDs along the top edge of this section. Two more proto-boards were added to interface with one side of the on-board FPGA and the other interface with a generic 50-pin connector. The generic 50 pin connector is the same type used in 6.111 kits to allow kits to connect with other 6.111 kits. The existence of this connector allows this kit to easily connect to the current 6.111 kits. All of these sections were placed on the right side of the protoboard. The only factor for this decision was that there are more right-handed people and they will find it easier to wire while the lego blocks are connected if the proto-boards were placed on the right side.

This leaves the entire left half for the lego block base and chips associated with it. Extra 3x16 connectors were placed around this area to provide extra power to a 3 x 3 grid of lego blocks. The total area taken up by this 3 x 3 grid of lego blocks is 10" x 10", so there was plenty of room.

Unfortunately, the place and route software was changed from PAD, a PC based

program to Allegro, a UNIX based program from PAD, a PC based program to Allegro, a UNIX based program for two reasons. Our version of PADS was outdated and Allegro is a more common, more robust layout editor. However, as with most UNIX programs, it was incredibly powerful but incredibly hard to setup and use. Since I was unable to learn the program in the given time frame, Gill Pratt did the layout and routing of the proto-board. His design can be seen in Appendix B

4.3 Parallel Port

The parts and protocols described in this section are required to communicate between a computer and the lab kit. The work done for this section is as follows: the schematic design and general protocol were developed by myself, Gill Pratt, and Andrew Huang, and the PAL file presented in Appendix D, which implements the protocols discussed here were written by Gill Pratt.

The parallel port is the communication medium between the kit and the computer that will program the lego blocks. To communicate through this port, a protocol for transferring data must be used. There are many existing protocols, some designed for single direction transfer, some designed for bi-directional transfer, some for bulk data, some for communication to multiple devices. In the new labs this semester, Andrew Huang designed a very simple request/acknowledge protocol specifically for the labs, but the hardware is designed to handle any protocol. This section describes how to implement any protocol, showing at each point how Andrew's protocol is implemented.

The most important part of implementing a protocol is deciding which one to use. Then each of the communication nodes must be configured to handle it.

For this kit, only a few thousand bytes are transferred during any program. Therefore, either of the existing bi-directional protocols I discussed in the section *Kit Communication: Communication* with the Computer (ECP and EPP) would be overly complicated. In fact, the configuration of the FPGAs are even simpler using one input pin for serial data and several more to control the data flow. Andrew's protocol is used for communication once the configuration is complete.

The protocol is implemented in the computer by creating a new parallel port driver (or downloading an existing one). This driver will control the signals necessary to implement the protocol. A copy of Andrew's driver can be seen in Appendix C.

On the protoboard, the first thing that must be done is program the FPGA. This protocol is so simple that it is easily handled by the computer using any driver. Once the FPGA is programmed, it then uses the request/acknowledge protocol in kitcomm to transfer bytes of data at a time.

Currently, all of the port signals go through a PAL chip which simply passes them along while the FPGA is being configured When the configuration is complete, the outputs are tri-stated so that the FPGA can control the port signals. The code for the PAL chip currently used can be seen in Appendix D.

Though used in a simple manner in our design, the PAL chip can provide an external source of logic if a more complicated protocol is used. Only the code needs to be rewritten and programmed into the PAL chip to implement a new design.

The parallel port is a widely available port that allows most modern computers to communicate to the kit, and our flexible, reprogrammable design allows any protocol to be implemented. This robustness will make the kit more attractive for other uses and keeps it

from getting outdated as protocols evolve.

4.4 Lego Block Base

The lego block base includes the on-board FPGA and all of the connectors that provide extra power paths to lego blocks in the network. The extra paths to power and ground reduce the amount of noise throughout the system by providing a return path for the signals that go directly to the power supply instead of going through other lego blocks. We placed these so that the first seven logo blocks - including the first block on the proto-board, two attaching directly to the north and south of the first, then four at the far corners of the lego block base - all have direct access to power from the protoboard. In addition to the noise reduction, they add stability by not leaving "hanging" blocks until several layers up. Figure 1 shows how the first ten blocks will connect securely in a diamond lattice.

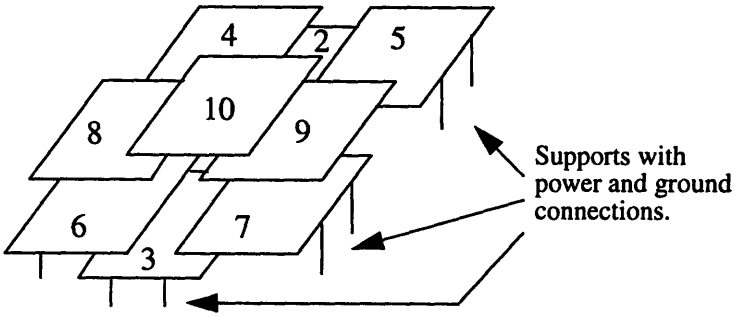


Figure 1: Ten Lego Blocks in a Diamond Lattice. The blocks are numbered here from the bottom up. Block 1 is on the protoboard, directly below Block 10, and is not shown in the figure. Blocks 2 and 3 connect to the North and South connectors of Block 1, respectively. On the third layer, Blocks 4 and 5 connect to Block 2, and Blocks 6 and 7 connect to Block 3. The fourth layer has Blocks 8 and 9, then these are connected through Block then which is alone on the fifth layer. Since the protoboard supports Blocks 2 through 7, the entire stucture is very stable.

The wiring of the FPGA on the protoboard is very similar to the ones on the lego block. The differences lie in the East and West connector pins. To the west, logic from the parallel is connected so that the FPGA can actively use it to communicate bi-directionally

to the computer. This process is described in more detail in the previous section. To the east, all of the data pins route directly to nets on a small proto-board. This allows students to interact with the lego blocks, such as using switches to input values or using the outputs to drive an external device.

5. Testing

Much of the testing for these new kits were done for the experimental section labs[10]. Nearly all of the pre-lab testing and lab construction was done by Andrew Huang, so credit for this section goes to him.

In the new labs, Labs 4, 5, and 6 used the schematic editor and simulation software that will be the standard in future labs. In addition, Lab 6 used the Xilinx compiling tools, which we hope to replace with our own faster version, and "Electric Lego" hardware. In Lab 4, the students construct the ALU hardware including the barrel shifter and control logic for the function and input bits. In Lab 5, the students build the remainder of the Beta machine as taught in the lectures[11]. Unfortunately, to keep the size of the Beta down (for Lab 6), the schematic designs were given to them and they only had to draw it in. However, they still seemed to understand it more fully than students who were working with the old kits. In Lab 6, a pre-made communication block used to send and receive signals off of the FPGA was added (the point of these labs is not for them to learn the intricacies of FPGA programming - yet), and the design was compiled (a process which currently takes three to four hours).

While working with the students, I noticed that the primary source of frustration this semester was the schematic editor and simulator software. This software contained several bugs which made the editor and the simulator inconsistent. This led to student confusion and in-depth debugging (check to see if a connection that appears connected actually is). Unfortunately, the work for this lies with the creator, Aldec and Xilinx. Luckily, once the students became more accustommed to its quirky behavior, they were able to design more complicated structures without too much trouble. In other words, small bugs in the software greatly reduce the learning curve for this otherwise complete and simple-to-use software.

The final design that went into the lego blocks this semester was a simplified version of the Beta machine. The machine was simplified by removing the barrel shifter from the ALU. It had to be simplified in order to place it all within one FPGA, thus avoiding complicated inter-lego communication handling. However, once the design was compiled it downloaded easily to the lego blocks, thanks mostly to Andrew Huang and Matt Congo who worked hard to get it this way.

Most of the information and files used in this process can be located in the new lab kit home page[9].

6. Future Extensions

Although this past semester could be marked as a successful one, there is still much work left to be done before we can achieve the level we desire, where each student

has his own kit, and can design, compile, and download his projects from the comfort of his own room. Most of the work is left in finishing the compilation software, specifically the router, then integrating the placer designed by Jay Grabeklis.

There are also some more changes and additions that need to be made on the new kit before this level can be accomplished. To start with, there were many test that have not been run yet involving the use of multiple lego blocks. These tests should be run not only for large Beta architecture designs, but also for other large digital designs such as those that might be produced in 6.111. This way the kits will be optimized to be useful for a variety of classes.

To complete the design of the kits, tools must be added and a storage space created for the parts they will be using on the proto-board area. Then each lego block, as well as each kit should be numbered, as it easy for these parts to disappear, and they are not cheap.

Once all of this has been accomplished, we will have fully functioning lab kits and new labs written for them. However, there are a few more problems left to solve before this project is complete.

First of all, we need to prevent students from simply copying labs from each other or from previous students. Since this is such an easy things to do in software, and such a hard thing to prevent, future designers for this class will have a lot of work to do here.

Also, the network of computers in the lab needs to be configured by somebody with Windows 95 networking knowledge. Although we managed to get through the semester with what we had, nobody in our group had had much experience with Windows networking systems. As a result, the following things have to be done:

- Students must be able to make private accounts so that their work is protected from

plagerism and alteration.

- The network whould be transparent, meaning that students should be able to use any computer and begin working without having to constantly download their design from their previous computer.
- Printers should be made accessible from the students' computer, not only from the administrative computers.

I have begun a web page for the lab kit to display the most recent status of the lab kit. I hope that those who take over the project use this resource to keep everyone else updated[12]

7. Conclusion

In summary, there was a great need for new lab kits in 6.004. Ones that could effectively teach students about the complicated circuits they were building. By using scalable, interconnected lego blocks, each with a configurable FPGA on them, we could provide a medium for the students to build almost limitless structures, while interactively simulating them in software, then watching it run on hardware.

The labs have been written, several working kits have been mostly constructed (see Future Extensions to see what is left), and part of our specially designed compiler has been written. However, there is still a lot of work to be done in several different areas before this project is complete.

Once completed, this design should be useful for hopefully the next ten years or

so. The technology implemented here is relatively new to the industry and carries with it enough potential to be the industry's medium of the future. It also carries with it the ability to implement any hardware structure, so even if FPGAs are not the wave of the future, they should be able to route the design.

Bibliography

1. Pratt, Gill. "Electric Legos: The next digital prototyping technology". 1995: http://www.ai.mit.edu/people/hschang/6.004/gillprop.html

2. Shoemaker, David and Metcalf, Chris and Ward, Steve. "NuMesh: A Communication Architecture for Static Routing". 1995: NuMesh Group, MIT LCS

3. *The Programmable Logic Data Book, Third Edition.* Xilinx. 1994.

4. Chang, Hyung. "Hardware Design of Electric Legos" 1996: Advanced Undergraduate Project, MIT. http://www.ai.mit.edu/people/hschang/6.004/aup.html

5. Grabeklis, Jay. "A Virtual Workbench for the New 6.004 Labkit". 1996: Advanced Undergraduate Project, MIT

6. Hennessey, John L and Patterson, David A. *Computer Architecture a Quantitative Approach.* 1996: Morgan Kaufman Publishers

7. Pratt, Gill and Nguyen, John. "Distributed Synchronous Clocking". Proceedings of the Sixteenth Conference on Advanced Research in VLSI. 1995: IEEE Computer Society

8. "IEEE 1284 Introduction". http://www.fapo.com/1284.htm

9. "IEEE 1284 EPP - Enhanced Parallel Port Mode". http://www.fapo.com/eppmode.htm

10. "6.004 New Lab Kit Home Page". http://cs00.mit.edu

11. "MIT 6.004, Spring, 1997". http://cag-www.lcs.mit.edu/6.004/

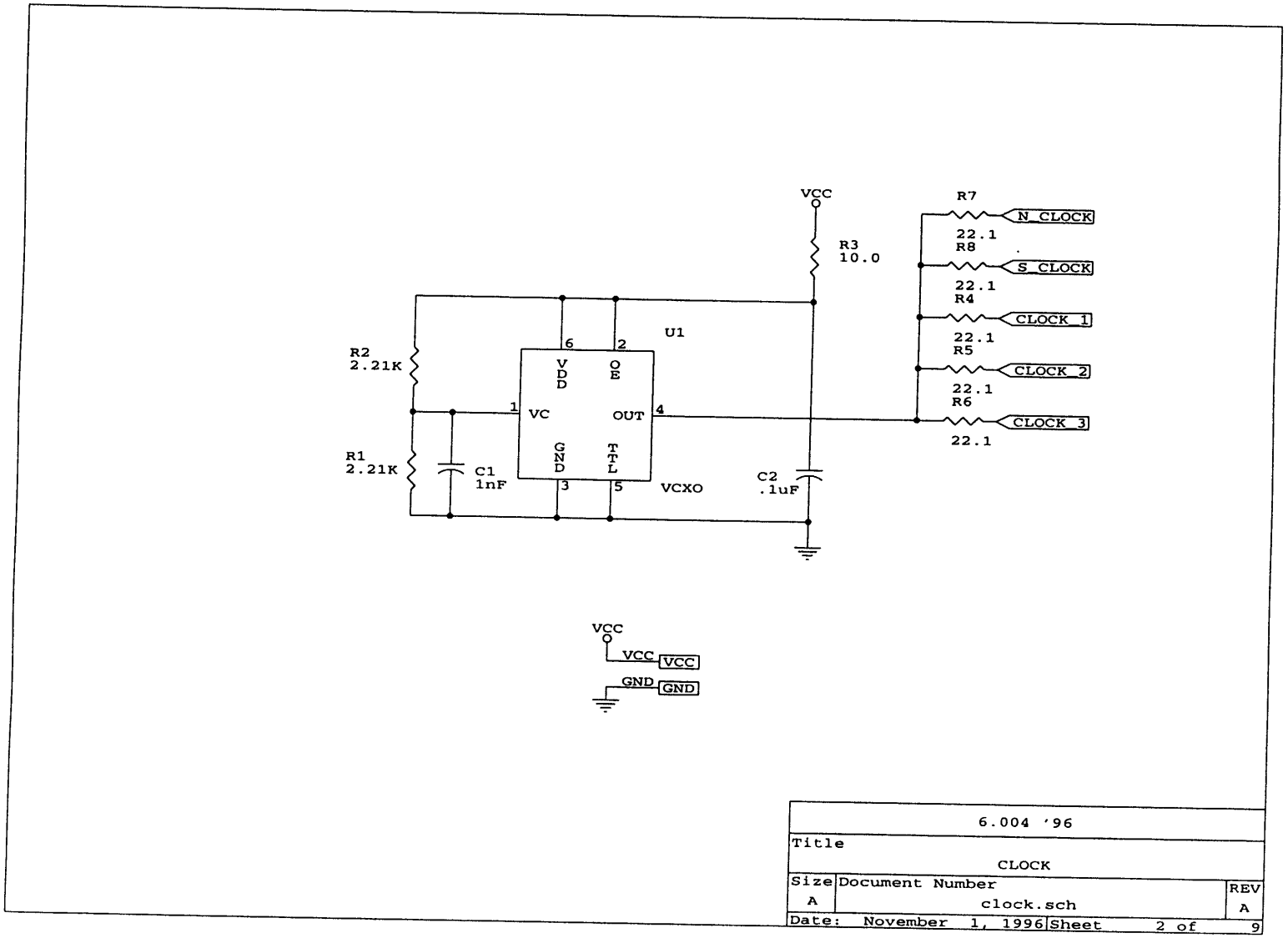
12. "New 6.004 Lab Kit - Electric Legos". http://www.ai.mit.edu/people/hschang/6.004/index.html

Appendix A: Protoboard Schematics

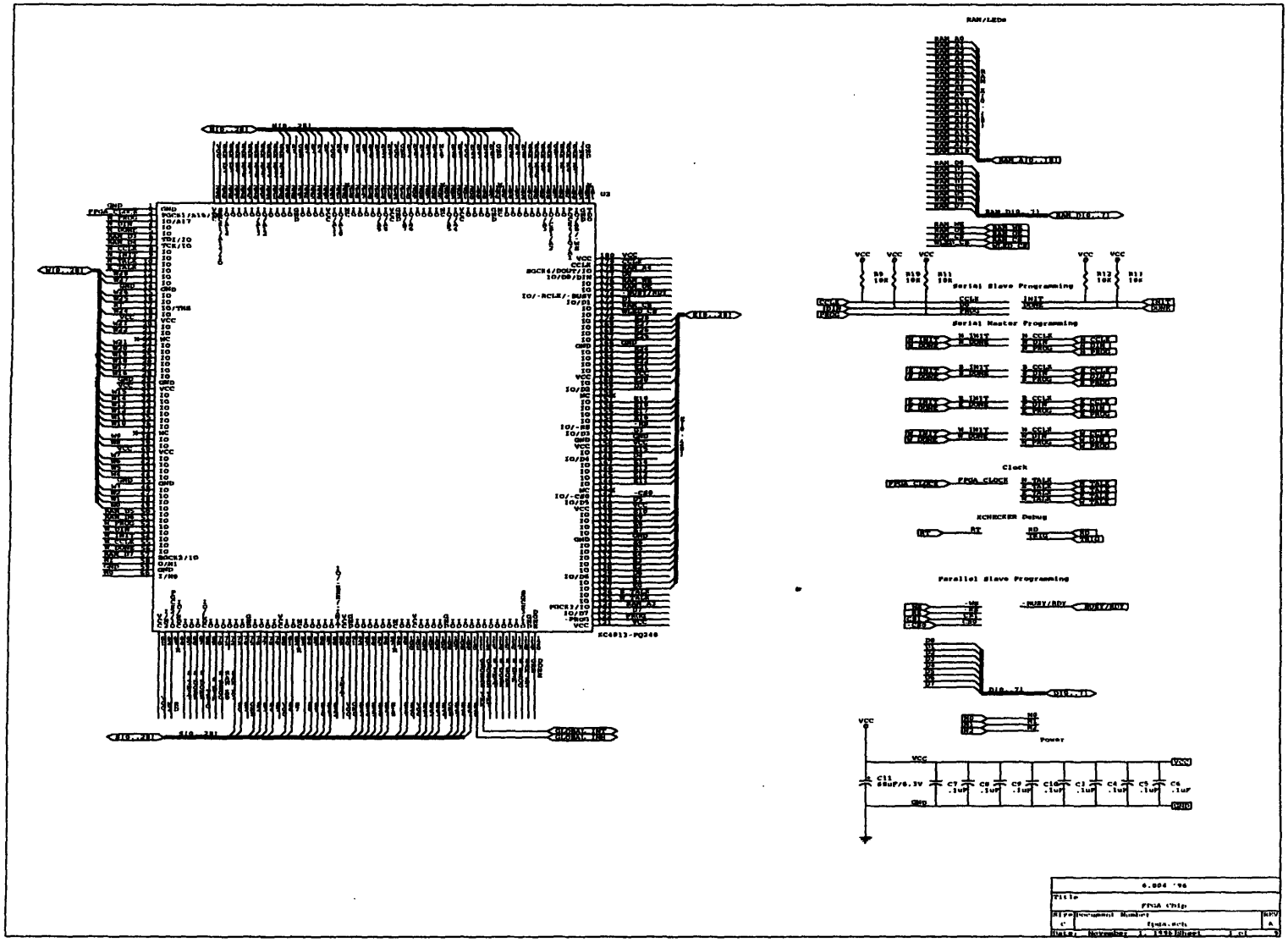
New 6.004 Lab Kit

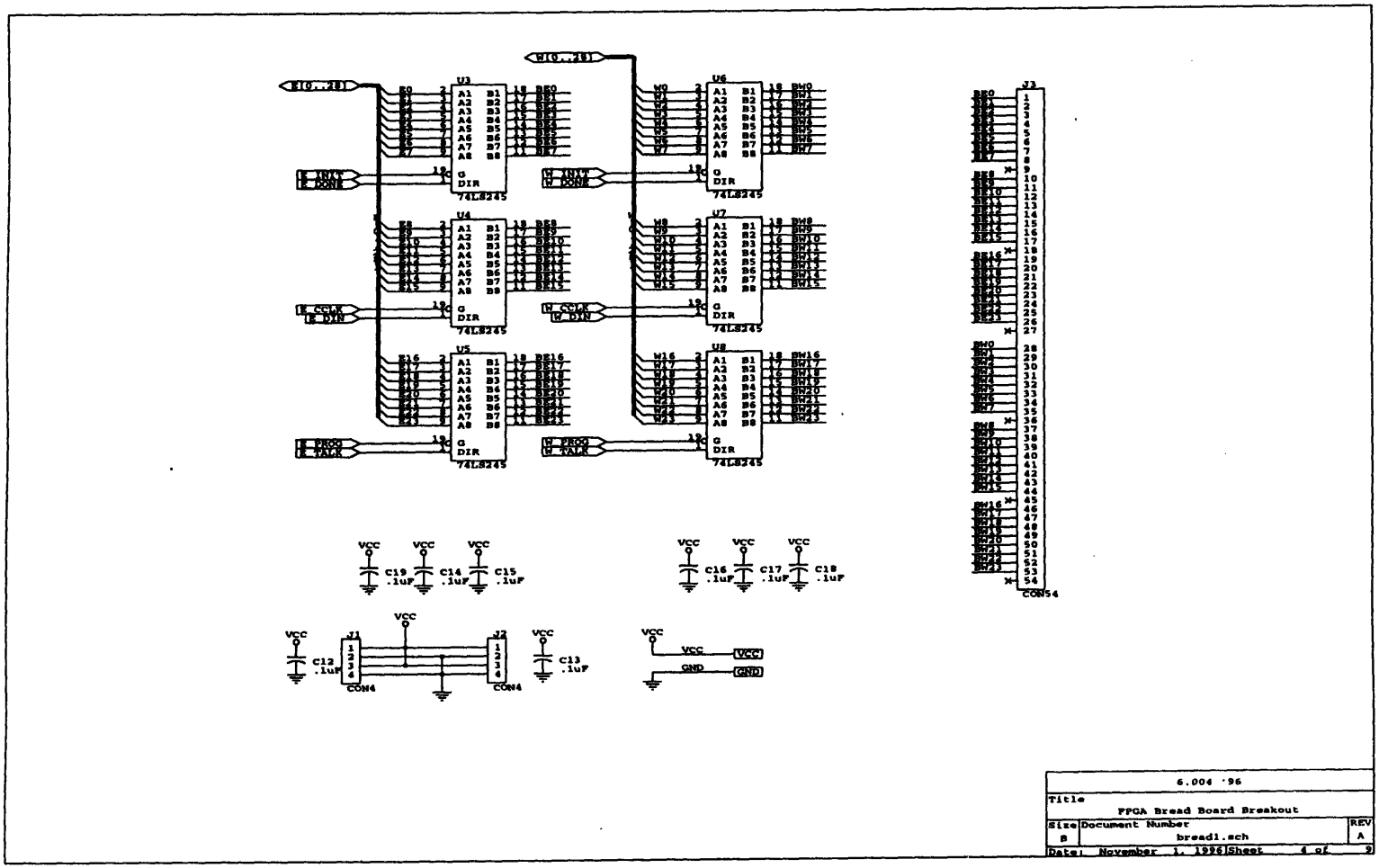
LINK
CLOCK.SCH
FPGA.SCH
BREAD1.SCH
BREAD2.SCH
COMM.SCH
RAM.SCH
NCONN.SCH
SCONN.SCH

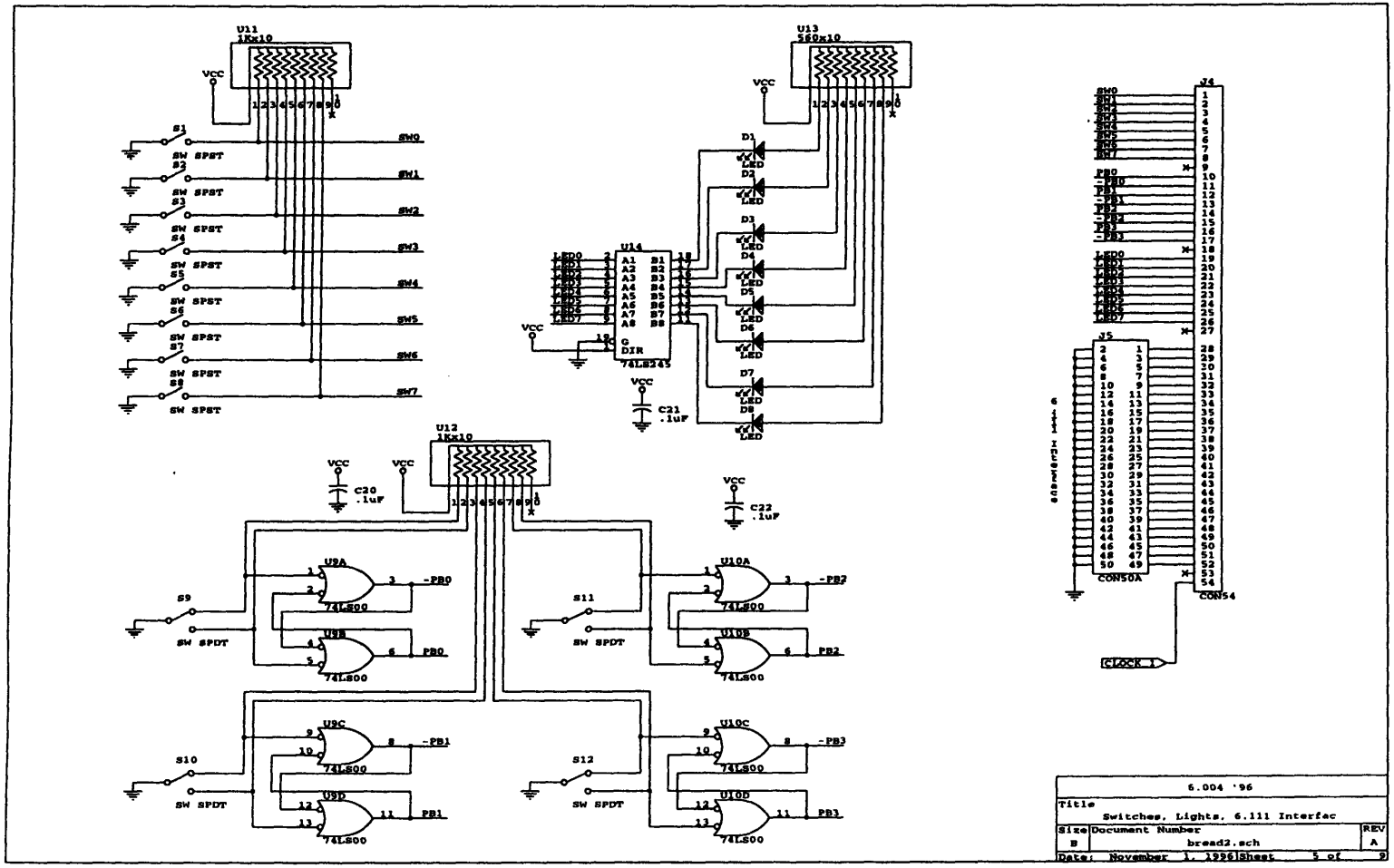
MIT 6.004 - New Lab Kit		
Title: Title Sheet		
Size: A	Document Number: nproto.sch	REV: A
Date: August 16, 1996	Sheet 1 of 9	

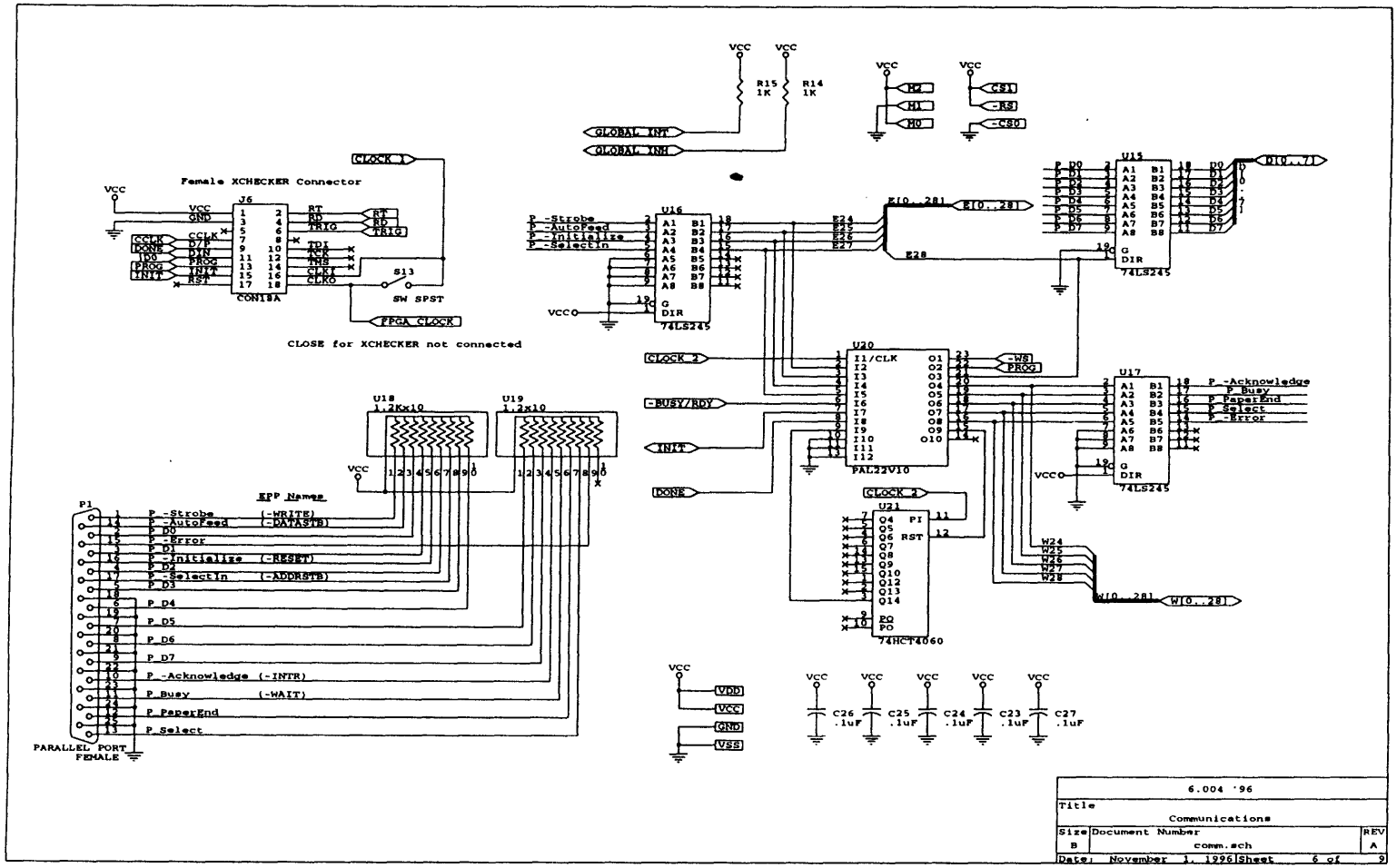


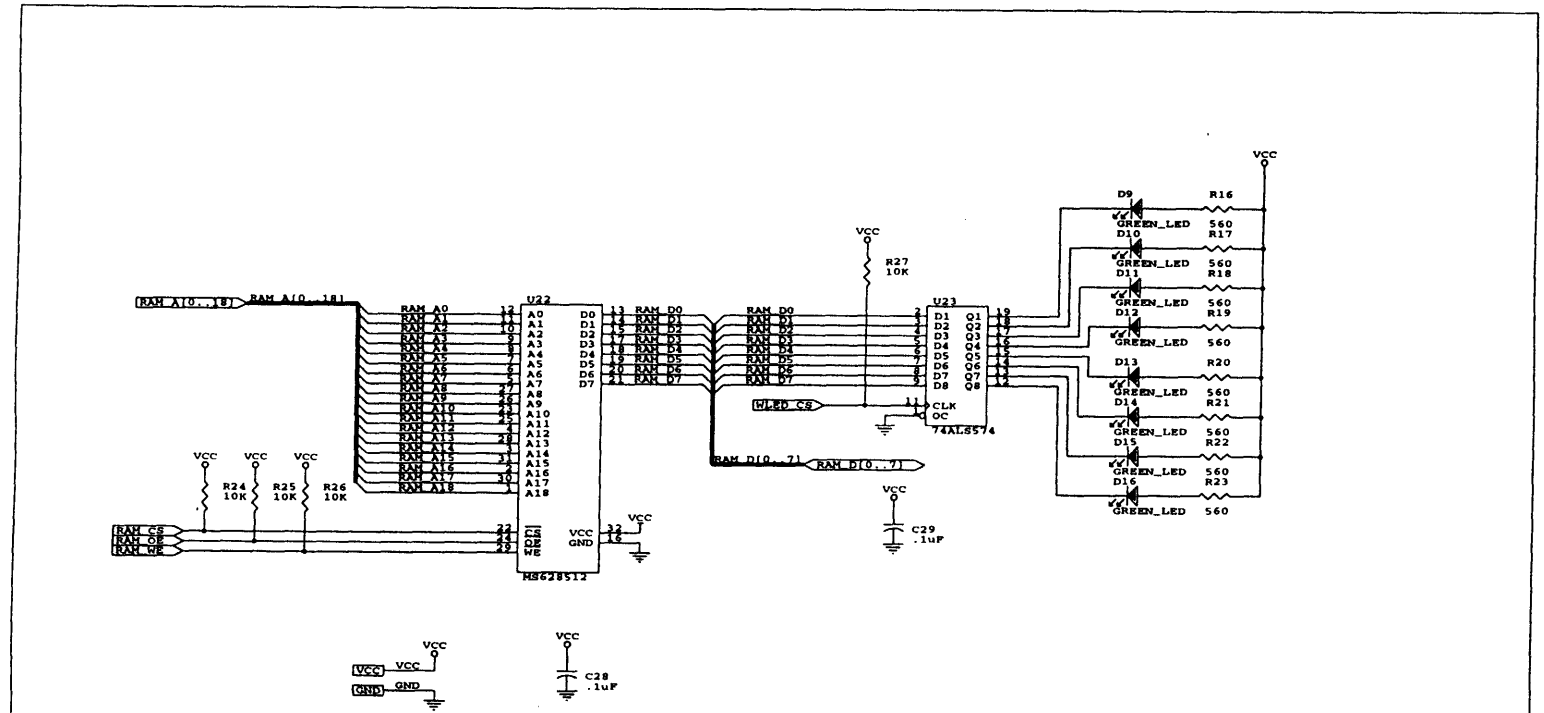
6.004 '96		
Title: CLOCK		
Size: A	Document Number: clock.sch	REV: A
Date: November 1, 1996	Sheet 2 of 9	


RAM/LEDs
Serial Slave Programming
Serial Master Programming
Clock
XCHECKER Debug
Parallel Slave Programming
Power
FPGA Chip

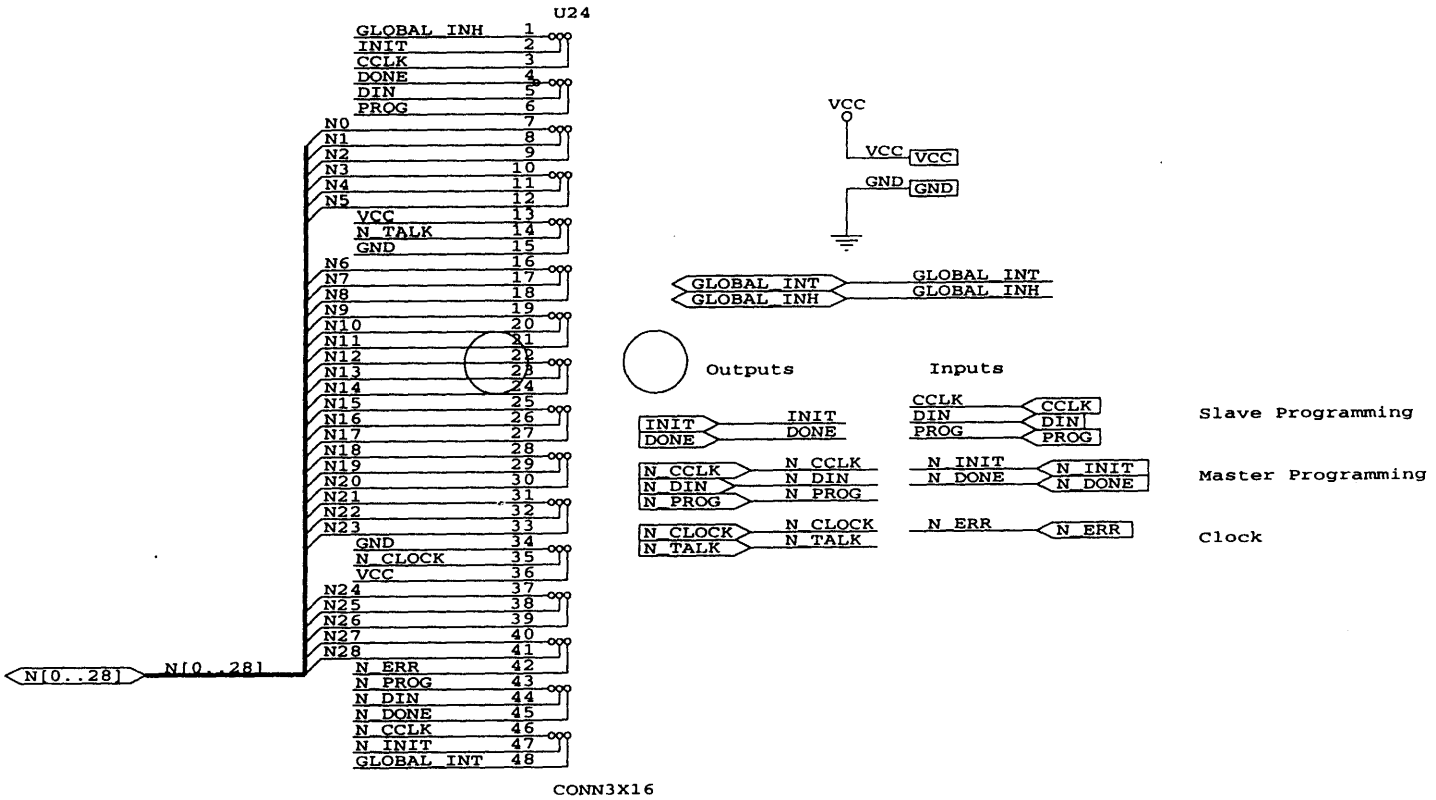

B[0..28]
W[0..28]
U3
U4
U5
U6
U7
U8
74LS245
J3
CON54
C19 .1uF
C14 .1uF
C15 .1uF
C16 .1uF
C17 .1uF
C18 .1uF
C12 .1uF
C13 .1uF
J1
J2
CON4
VCC
GND
6.004 '96
Title
FPGA Bread Board Breakout
Size B
Document Number
bread1.sch
REV A
Date: November 1, 1996
Sheet 4 of 9


U11
1Kx10
VCC
S1
SW SPST
S2
S3
S4
S5
S6
S7
S8
SW0
SW1
SW2
SW3
SW4
SW5
SW6
SW7
U13
560x10
D1
D2
D3
D4
D5
D6
D7
D8
LED
U14
74LS245
C21
.1uF
U12
C20
.1uF
C22
.1uF
U9A
U9B
U9C
U9D
74LS00
S9
S10
SW SPDT
-PB0
PB0
-PB1
PB1
U10A
U10B
U10C
U10D
S11
S12
-PB2
PB2
-PB3
PB3
J4
J5
CON50A
CON54
6.111 Interface
CLOCK
6.004 '96
Title
Switches, Lights, 6.111 Interfac
Size
B
Document Number
bread2.sch
REV
A
Date: November 1, 1996
Sheet 5 of 9

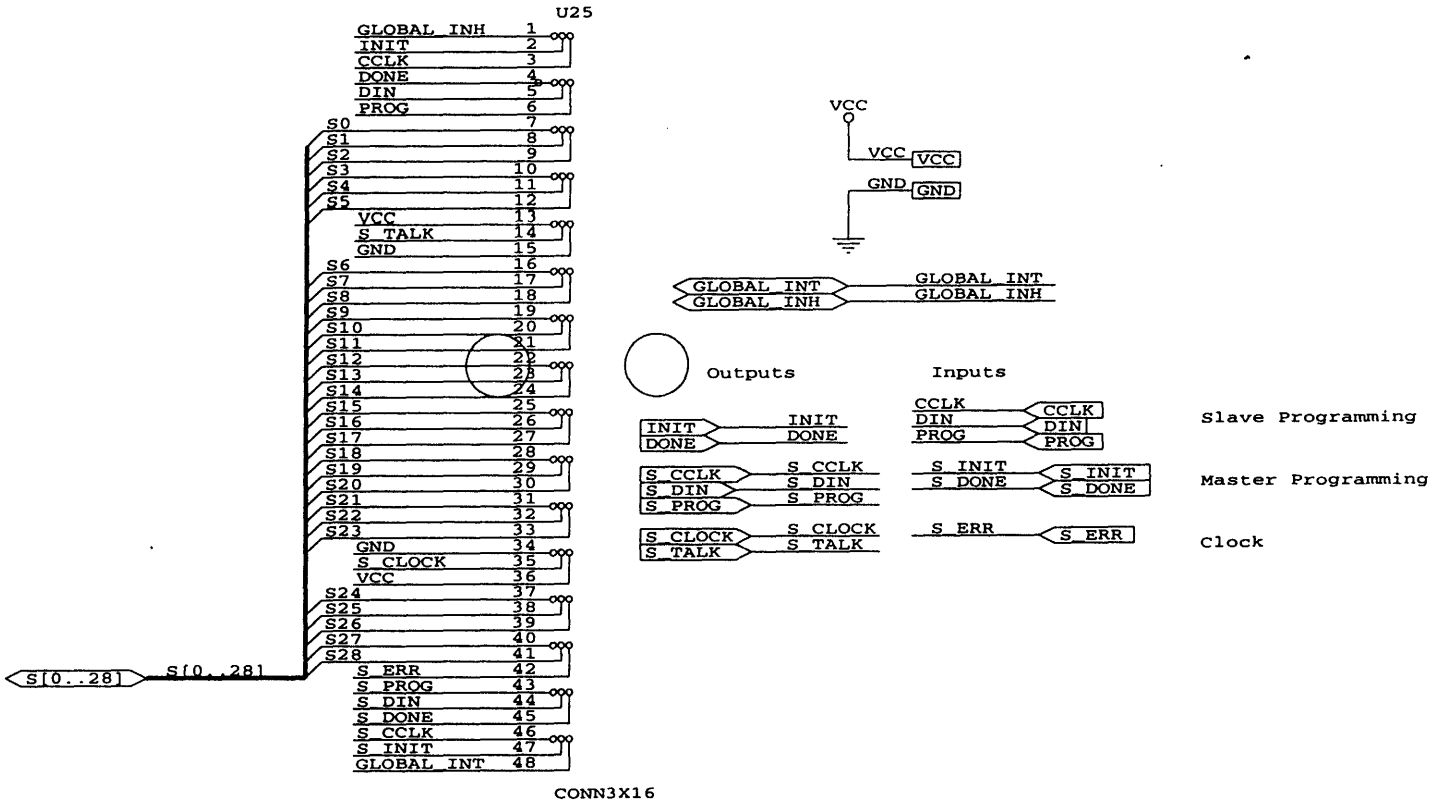

Female XCHECKER Connector
J6
CON18A
CLOCK_1
FPGA_CLOCK
S13
SW SPST
CLOSE for XCHECKER not connected
R15 1K
R14 1K
GLOBAL_INT
GLOBAL_INH
M2
M1
M0
CS1
-RS
-CS0
U15
74LS245
U16
74LS245
U20
PAL22V10
U17
74LS245
U18 1.2Kx10
U19 1.2x10
U21
74HCT4060
CLOCK_2
-BUSY/RDY
INIT
DONE
-WS
PROG
P1
EPP Names
P_Strobe (-WRITE)
P_AutoFeed (-DATASTB)
P_Error
P_Initialize (-RESET)
P_SelectIn (-ADDRSTB)
P_D0
P_D1
P_D2
P_D3
P_D4
P_D5
P_D6
P_D7
P_Acknowledge (-INTR)
P_Busy (-WAIT)
P_PaperEnd
P_Select
PARALLEL PORT FEMALE
VDD
VCC
GND
VSS
C26 .1uF
C25 .1uF
C24 .1uF
C23 .1uF
C27 .1uF
6.004 '96
Communications
comm.sch
November 1, 1996
6 of 9



6.004 '96		
Title: RAM + LEDs		
Size: B	Document Number: ram.sch	REV: A
Date: November 1, 1996	Sheet 7 of 9	

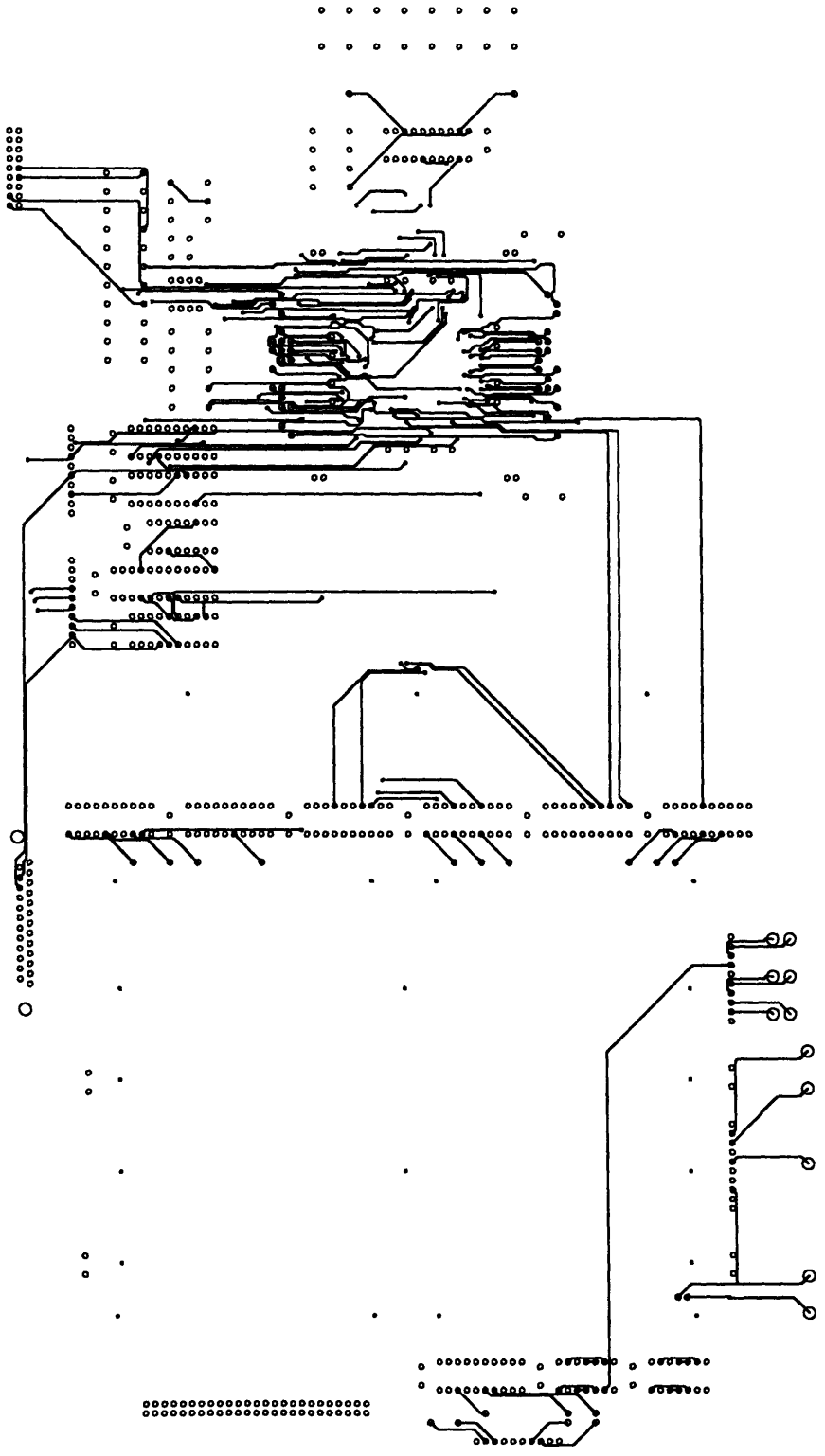


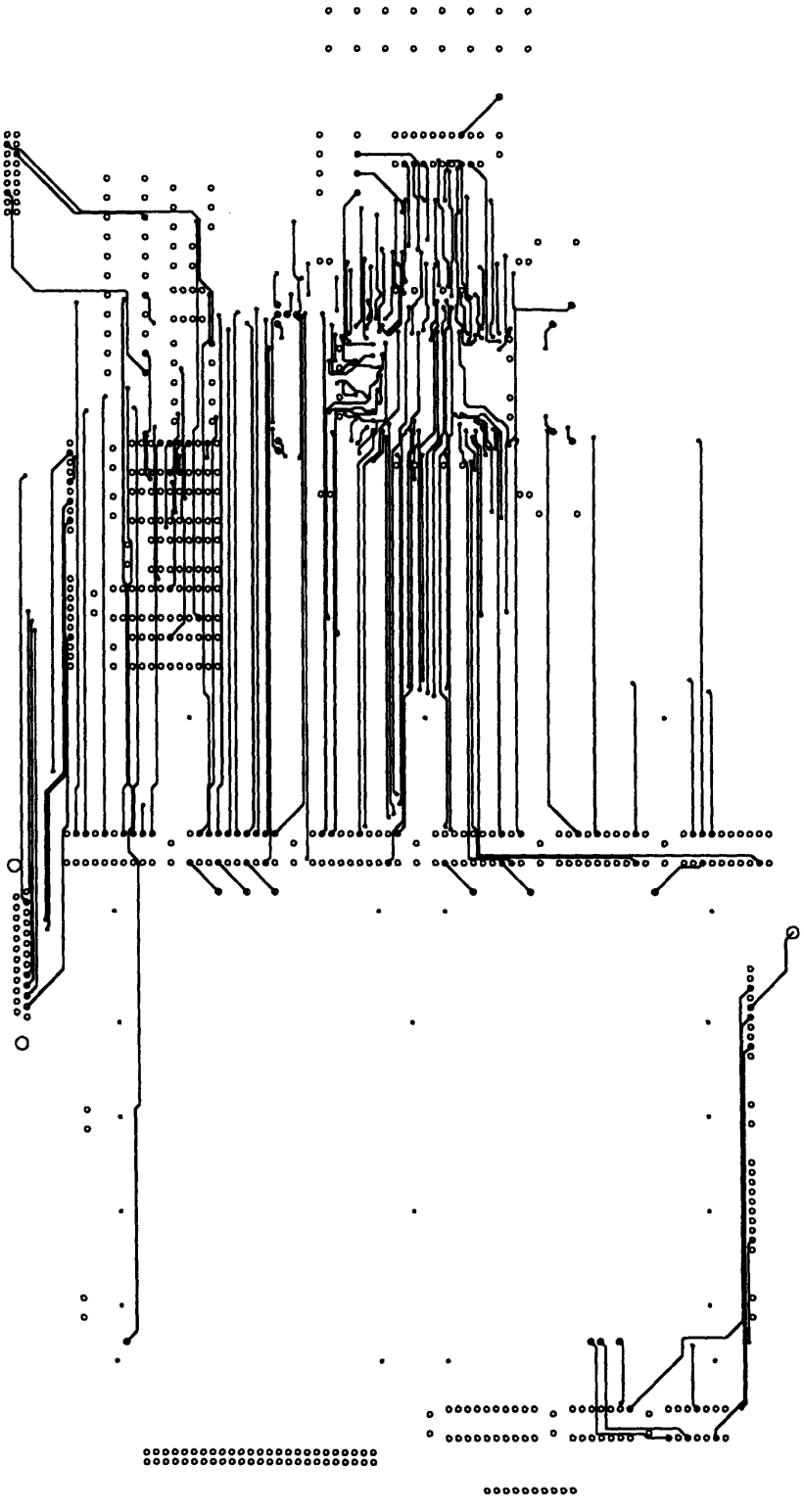
MIT 6.004 - Electric Legos		
Title: North Connector		
Size: A	Document Number: nconn.sch	REV: A
Date: August 14, 1996	Sheet 8 of 9	

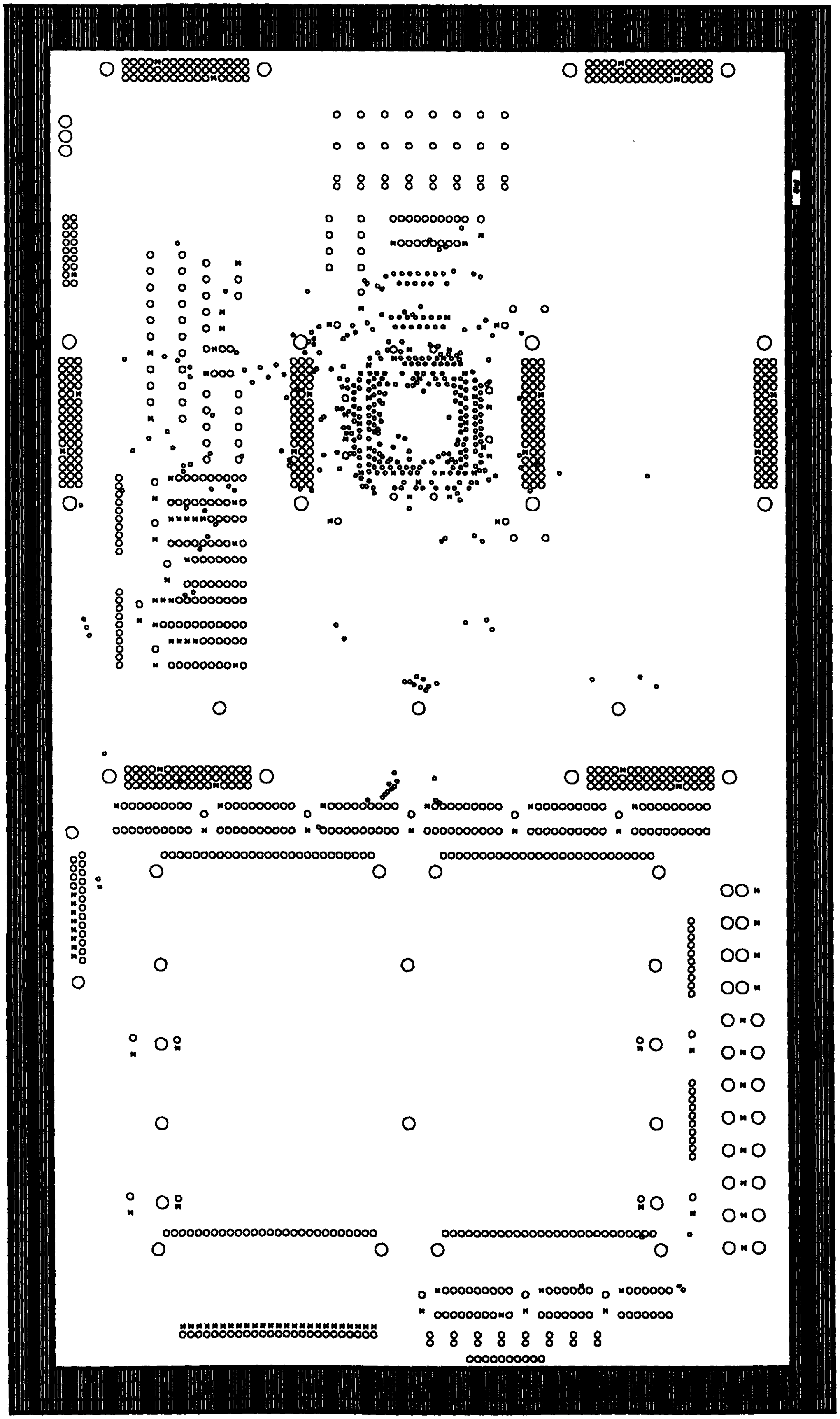


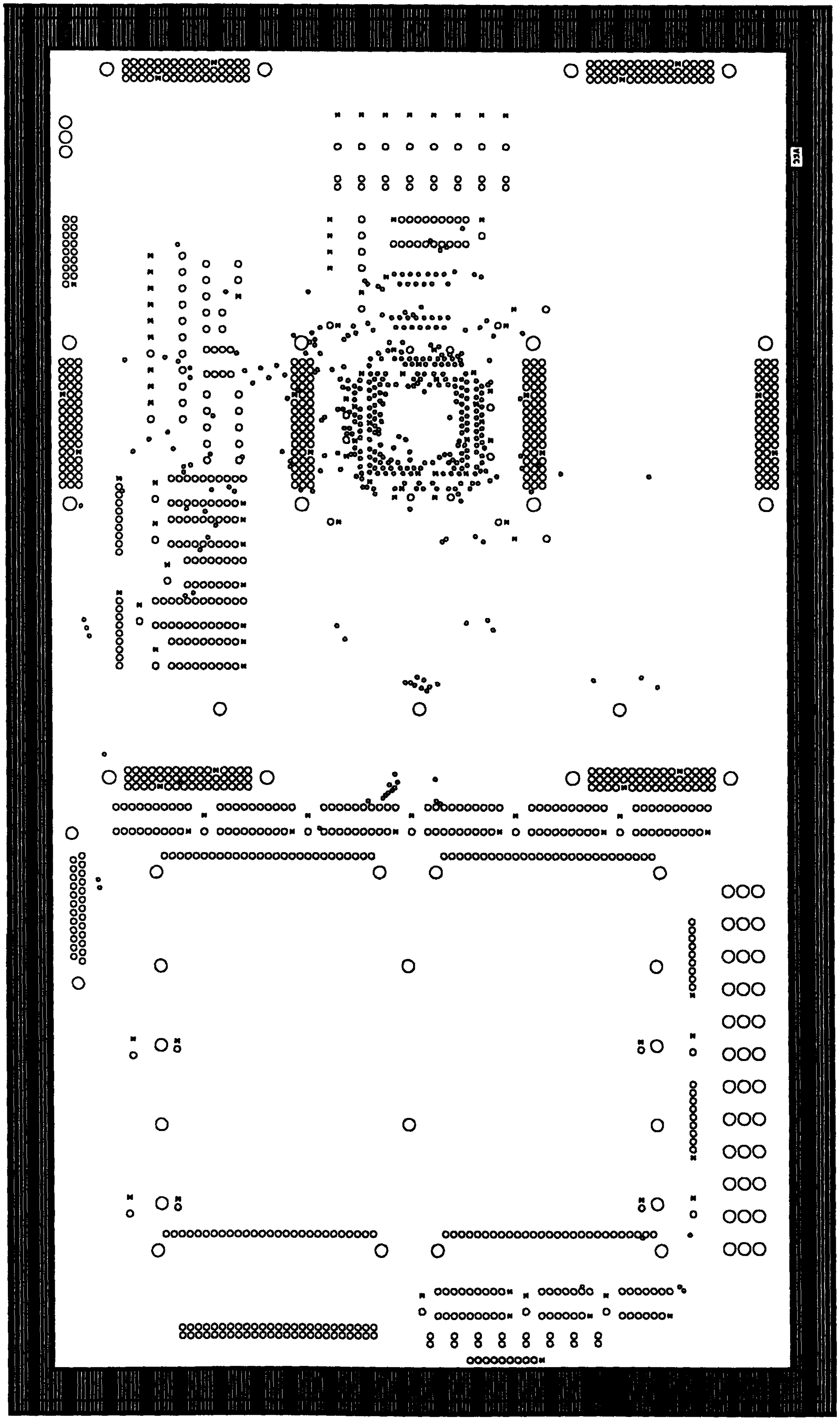
MIT 6.004 - Electric Legos		
Title: South Connector		
Size: A	Document Number: sconn.sch	REV: A
Date: August 14, 1996	Sheet 9 of 9	

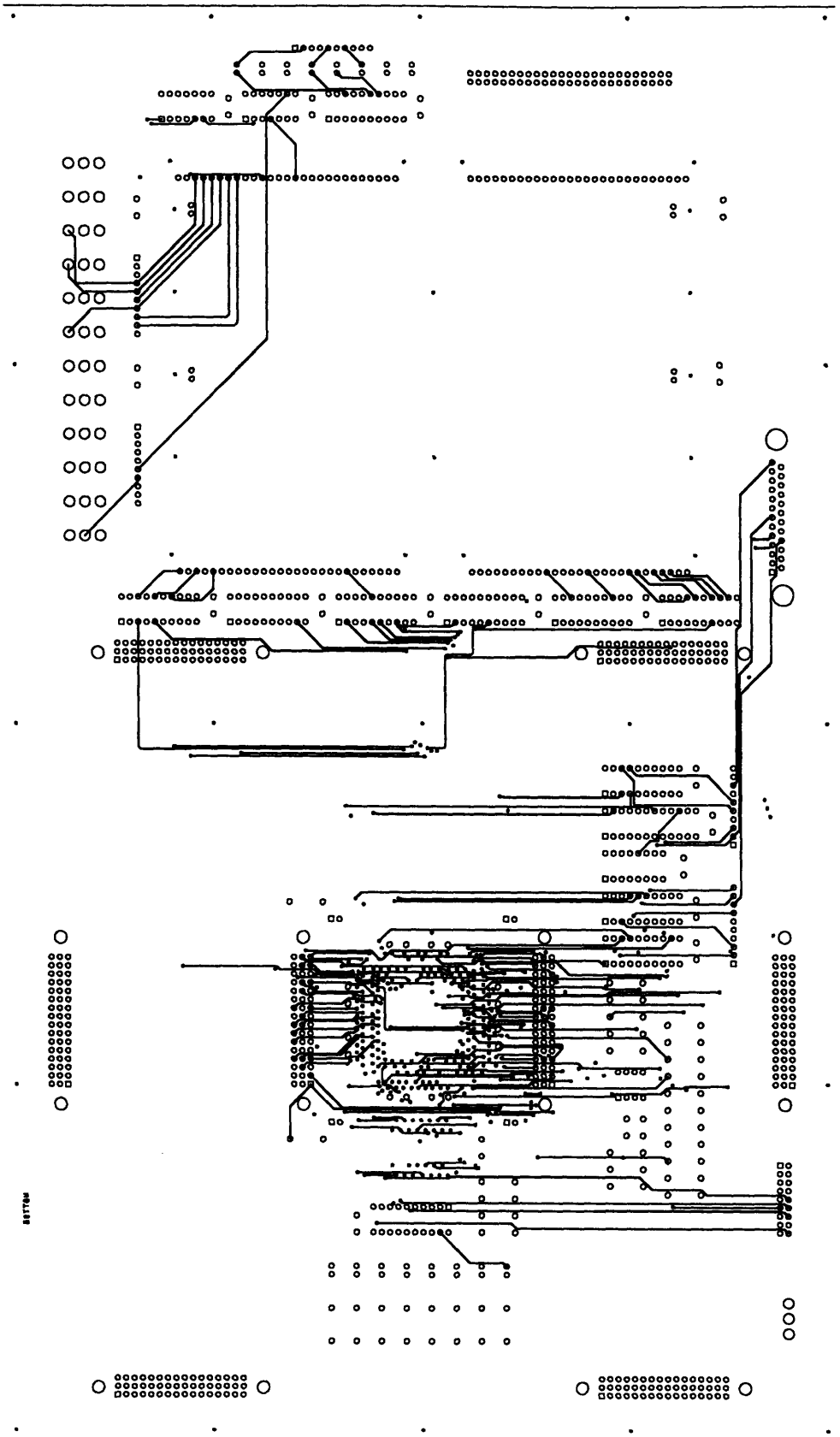
Appendix B: Protoboard Layout

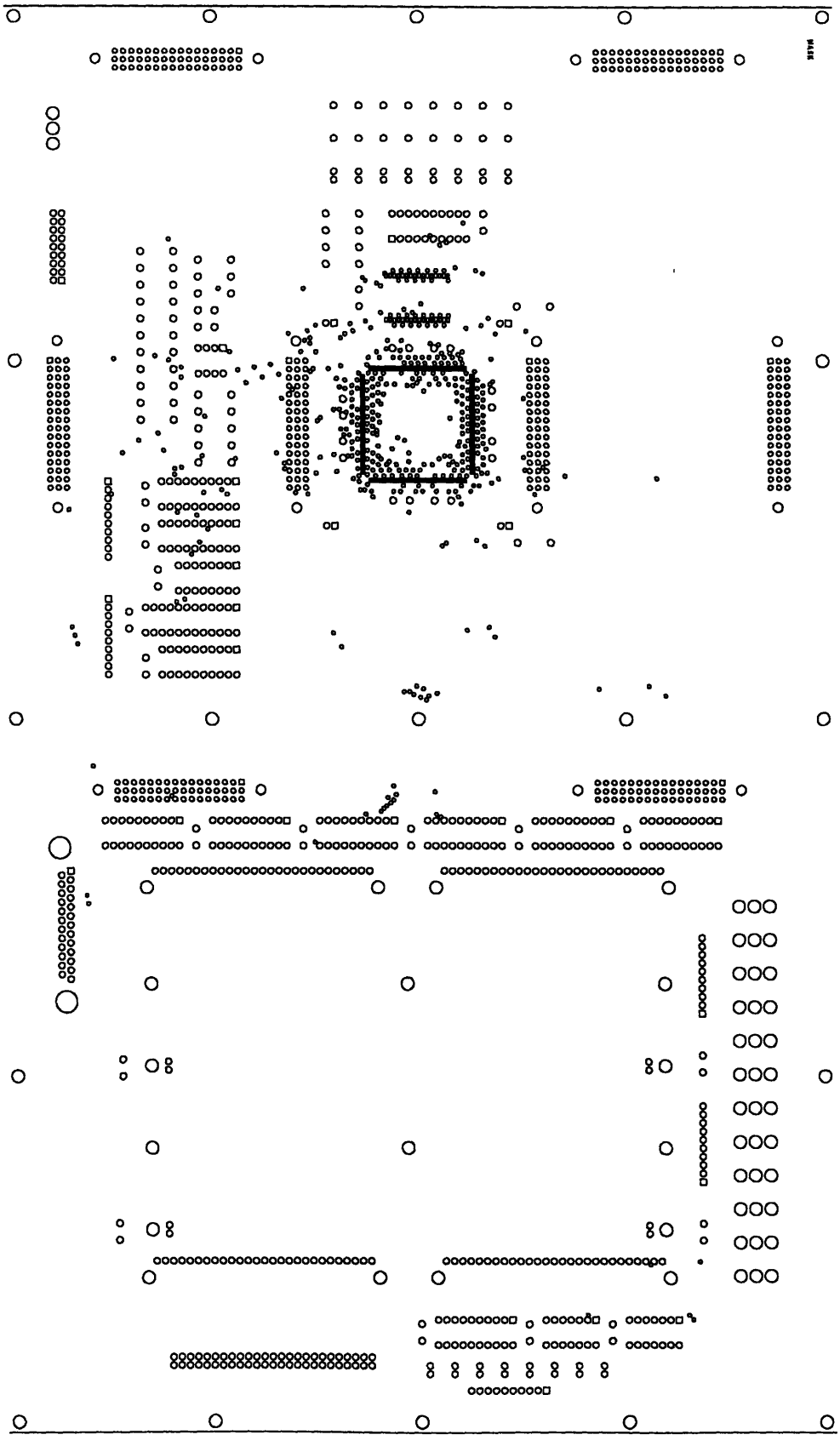


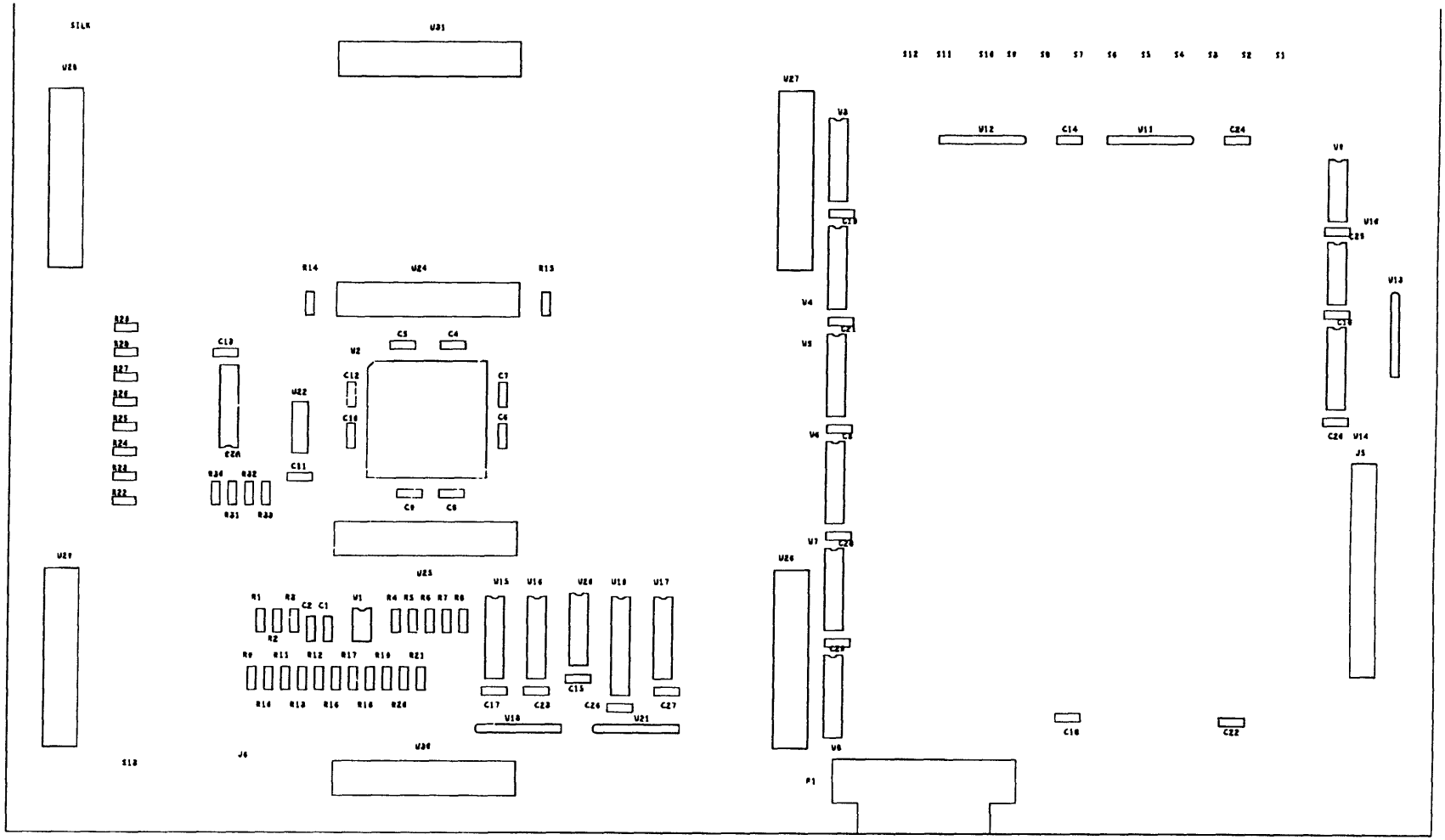


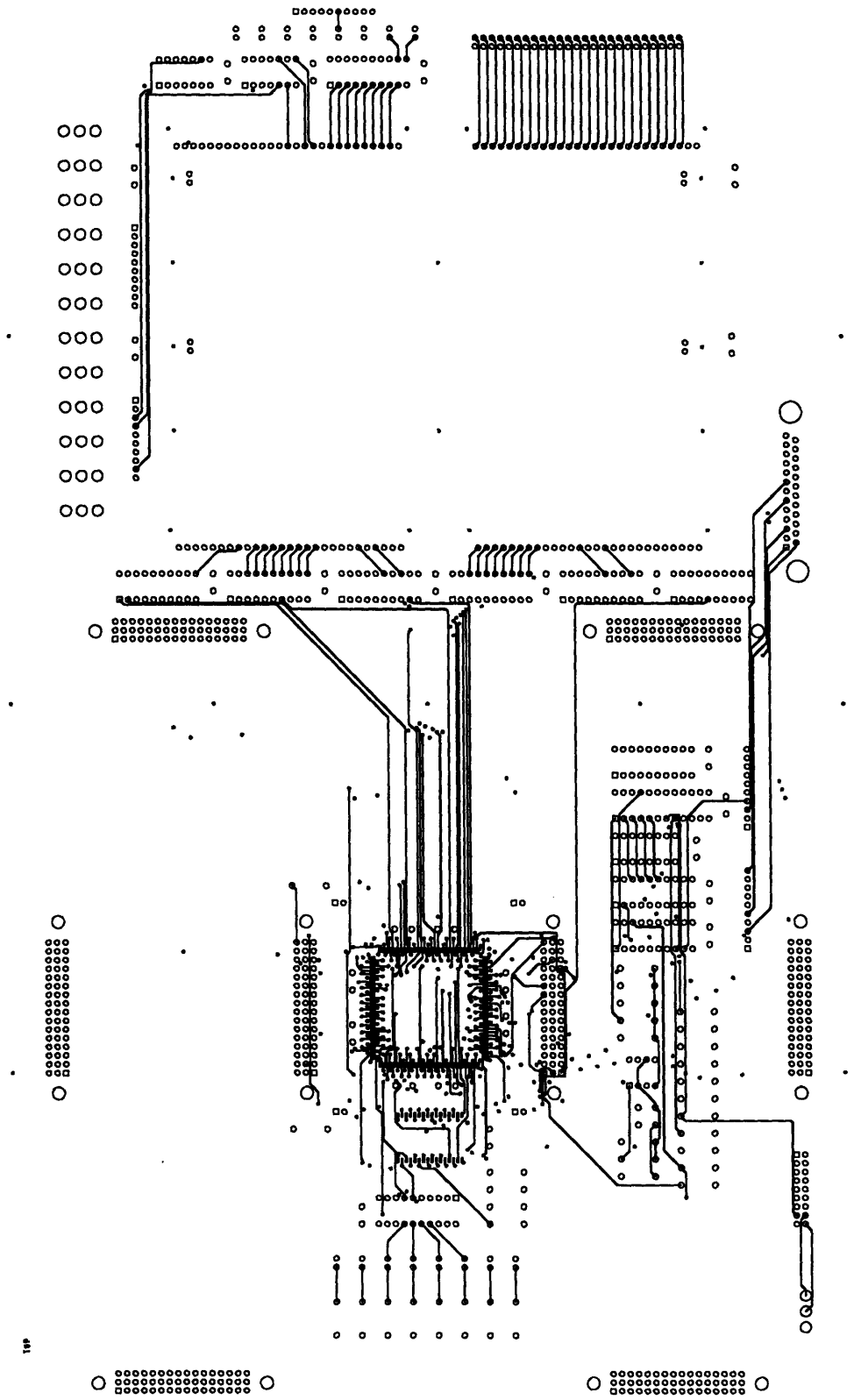



32A






SILK
U31
U28
U27
S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1
U12
C14
U11
C24
U9
U14
C25
U13
C19
U3
C1
U4
U5
C2
U6
C3
U7
C20
U26
C29
U8
P1
C18
C22
C26 U14
J5
S23
S20
S27
S26
S25
S24
S21
S22
R14
U24
R15
C13
C5
C4
U2
C12
C7
U22
C10
C4
C23
R34 R32
C11
R31 R30
C9
C8
U25
U29
R1 R2 C2 C1 U1 R4 R5 R6 R7 R8
R3
R9 R11 R12 R17 R19 R21
R14 R13 R16 R18 R20
U15 U16 U20 U18 U17
C17 C21 C15 C24 C27
U18
U21
J6
U30
S18



Appendix C: Parallel Port Driver

pport.c

```
/****
 * pport.c                    Andrew Huang                          MIT
 *
 * 4/12/97                                                         6.004
 ****/

/*** defines ***/
/* activate linux compatibility stuff */
#define DOS 1

/*** includes ***/
#include "pport.h"

/*** typedefs ***/

/*** globals ***/

/*** func protos ***/
static int flip_byte(int byte);

void init_ports() {
  /* init_ports */
  /* locals */

  /* body */

#ifdef LINUX
  ioperm((unsigned long) LP1_BASE, (unsigned long) 0x3, 0x2 );
#endif

} /* init_ports */

static int flip_byte(int byte)
{
  int bit=0, result = 0;

  for (bit = 0; bit < 8; bit++)
    if (byte & (1 << bit))
      result |= (1 << (7 - bit));

  return (result);
}

int main() {
  /* main */
  /* locals */
  unsigned int i, count;
  long  wait;
  FILE *bitstream;
  char  c;
  char  cr;
  char  temp;
```

```
/* body */
/* init any machine-specific stuph */

init_ports();

if( NULL IS (bitstream = fopen( "kitcomm.bit", "r" )) ) {
  printf("Can't open kitcomm.bit for reading.\n");
  exit(0);
} /* if */

/* initialize the LCA */
temp = (char) inb( LP1_CR ) & ~CR_INIT;
printf( "init byte: %X\n", 0xFF & temp );
outb( (char) temp, LP1_CR );
for( wait = 0; wait < 10000; wait++ )
 ; /* wait for LCA to reset */
temp = (char) inb( LP1_CR ) | CR_INIT;
printf( "init byte: %X\n", 0xFF & temp );
outb( temp, LP1_CR );

/* wait for /init to go high */
count = 0;
while( !(inb( LP1_SR ) & SR_ERR) ) {
  count++;
  if( count > TIMEOUT ) {
    printf( "Init did not go high; is kit connected?\n" );
    goto test;
  } /* if */
} /* while */

/* upload the bitstream */
c = 0;
count = 0;
while( (unsigned char) c AINT 0xFF ) {
  fread( &c, 1, 1, bitstream );
  count++;
} /* while */

outb( (char) 0xFF, LP1_DR );
outb( (char) (inb( LP1_CR ) | CR_STB) & 0xFF, LP1_CR );
outb( (char) (inb( LP1_CR ) & ~CR_STB) & 0xFF, LP1_CR );

while( !( inb( LP1_SR ) & SR_BUSY) ) {
  if( wait++ > 40000 ) {
    printf( "Wait count exceeded, exitting.\n" );
    goto test;
  } /* if */
} /* while */

while( !feof( bitstream ) ) {
  fread( &c, 1, 1, bitstream );
  count++;
```

```
    /* wait for RDY/BSY to go high again */
    wait = 0;
    while( !( inb( LP1_SR ) & SR_BUSY) ) {
      if( wait++ > 40000 ) {
printf( "Wait count exceeded, exitting.\n" );
goto test;
      } /* if */
    } /* while */

    if(!( inb(LP1_SR) & SR_ERR)) {
      printf( "Got error while downloading, exitting.\n" );
      goto test;
    } /* if */

    outb( (char) flip_byte(c) & 0xFF, LP1_DR );

    outb( (char) (inb( LP1_CR ) | CR_STB) & 0xFF, LP1_CR );
    for( wait = 0; wait < 1000; wait++ )
;

    outb( (char) (inb( LP1_CR ) & ~CR_STB) & 0xFF, LP1_CR );

    /* wait for RDY/BSY again */
    wait = 0;
    while( !( inb( LP1_SR ) & SR_BUSY) ) {
      if( wait++ > 40000 ) {
printf( "Wait count exceeded, exitting.\n" );
goto test;
      } /* if */
    } /* while */

    if( !(count % 1024) ) {
/*      printf( "count: %d\n", count ); */
printf( "." );
    }

    /*    printf("byte: %X, cr: %X, sr: %X\n", c & 0xFF, inb( LP1_CR ) &
0xFF, inb( LP1_SR) & 0xFF );
    getchar(); */

  } /* while */

test:
  printf( "\nCompleted with download to Kit: %d bytes.\n", count );

  outb( inb(LP1_CR) | CR_STB, LP1_CR );
  outb( inb(LP1_CR) | 0x3, LP1_CR );

  printf( "Default CR value: %X, SR value: %X\n", 0xFF & inb(LP1_CR),
0xFF & inb(LP1_SR) );
  exit(0);
  printf( "Now testing design.\n" );

  outb( inb(LP1_CR) | CR_AUTO, LP1_CR ); /* turn off writes */
```

```
  printf( "strobe to 1 (inverted to 0 by hardware)\n" );
  outb( inb(LP1_CR) | CR_STB, LP1_CR );
  getchar();
  printf("Asserting 0xA2 to data bus\n" );
  outb( 0xA2, LP1_DR );
  printf("Writing into register\n" );
  outb( inb(LP1_CR) & ~CR_AUTO, LP1_CR );
  outb( inb(LP1_CR) | CR_AUTO, LP1_CR );
  printf("Press enter to display register:\n");
  getchar();
  outb( inb(LP1_CR) & ~CR_STB, LP1_CR );
  printf( "Press enter to modify LED to 0xF0:\n" );
  getchar();
  outb( 0xF0, LP1_DR );
  outb( inb(LP1_CR) & ~CR_AUTO, LP1_CR );
  outb( inb(LP1_CR) | CR_AUTO, LP1_CR );
  printf( "Press enter to resume cylon mode\n" );
  getchar();
  outb( inb(LP1_CR) | CR_STB, LP1_CR );


} /* main */
```

pport.h

```
/****************************** includes
******************************/
#ifdef LINUX
/* linux stuph, not all of it is necessary */
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/busmouse.h>
#include <linux/signal.h>
#include <linux/errno.h>
#include <linux/mm.h>

/* assembly stuph, not all of it is necessary */
#include <asm/io.h>
#include <asm/segment.h>
#include <asm/system.h>
#include <asm/irq.h>

#include <unistd.h>
#endif

#ifdef DOS
#include <dos.h>
#define  inb(a)     inp(a)
#define  outb(a,b)  outp(b,a)
#endif

#include <stdio.h>

/********************************* defines
*********************************/

/* stylistic defines */
#define IS   ==
#define AINT !=
#define AND  &&
#define OR   ||

/* kit timeout in hard-coded loops */
#define TIMEOUT 100000

/* hardware naming abstractions */
/* information from
       http://www.paranoia.com/~filipg/HTML/LINK/PORTS/F_PARALLEL5.html
    by
       Kris Heidenstrom <kheidenstrom@actrix.gen.nz>, 10/19/1994

    and

       http://www.paranoia.com/~filipg/HTML/LINK/PORTS/F_PARALLEL1.html
    by
       Zahahai Stewart <parport@hisys.com>, 1/9/94
                          (general good stuff here)
```

```
 */

/*** offsets for parallel port ***/
/* I'm hard coding in the device address of lpt1 because this needs
 * to be portable between linux and DOS.  :-P   Wait...that doesn't
 * quite make sense, but let's just say it's harder to code an autodetect
 * sequence for both linux and DOS than to assume that lpt1 is always at
 * the same place (which it usually is, and most people only use lpt1
 * anyways).
 */

#define LP1_BASE  0x378
#define DR_OFF    0x0
#define SR_OFF    0x1
#define CR_OFF    0x2

#define LP1_DR    LP1_BASE + DR_OFF
#define LP1_SR    LP1_BASE + SR_OFF
#define LP1_CR    LP1_BASE + CR_OFF

/****
 * 7. Data Register
 *
 * The data register is at IOBase+0. It may be read and written (using
 * the IN and OUT instructions, or inportb() and outportb() or inp() and
 * outp()). Writing a byte to this register causes the byte value to
 * appear on pins 2 through 9 of the D-sub connector (unless the port is
 * bidirectional and is set to input mode). The value will remain latched
 * until you write another value to the data register. Reading this
 * register yields the state of those pins.
 *
 *         7 6 5 4 3 2 1 0
 *         * . . . . . . .  D7 (pin 9), 1=High, 0=Low
 *         . * . . . . . .  D6 (pin 8), 1=High, 0=Low
 *         . . * . . . . .  D5 (pin 7), 1=High, 0=Low
 *         . . . * . . . .  D4 (pin 6), 1=High, 0=Low
 *         . . . . * . . .  D3 (pin 5), 1=High, 0=Low
 *         . . . . . * . .  D2 (pin 4), 1=High, 0=Low
 *         . . . . . . * .  D1 (pin 3), 1=High, 0=Low
 *         . . . . . . . *  D0 (pin 2), 1=High, 0=Low
 *
 ****/

/* no bit twiddles here */

/****
 * 8. Status Register
 *
 * The status register is at IOBase+1. It is read-only (writes will be
 * ignored). Reading the port yields the state of the five status input
 * pins on the parallel port connector at the time of the read access:
 *
 *         7 6 5 4 3 2 1 0
 *         * . . . . . . .  Busy . . (pin 11), high=0, low=1 (inverted)
```

```
 *          . * . . . . . .  Ack  . . (pin 10), high=1, low=0 (true)
 *          . . * . . . . .  No paper (pin 12), high=1, low=0 (true)
 *          . . . * . . . .  Selected (pin 13), high=1, low=0 (true)
 *          . . . . * . . .  Error. . (pin 15), high=1, low=0 (true)
 *          . . . . . * * *  Undefined
 *
 ****/

#define SR_BUSY  0x80
#define SR_ACK   0x40
#define SR_NP    0x20
#define SR_SEL   0x10
#define SR_ERR   0x08

/****
 * 9. Control Register
 *
 * The control register is at IOBase+2. It is read/write:
 *
 *          7 6 5 4 3 2 1 0
 *          * * . . . . . .  Unused (undefined on read, ignored on write)
 *          . . * . . . . .  Bidirectional control, see below
 *          . . . * . . . .  Interrupt control, 1=enable, 0=disable
 *          . . . . * . . .  Select . . (pin 17), 1=low, 0=high (inverted)
 *          . . . . . * . .  Initialize (pin 16), 1=high, 0=low (true)
 *          . . . . . . * .  Auto Feed  (pin 14), 1=low, 0=high (inverted)
 *          . . . . . . . *  Strobe . . (pin 1),  1=low, 0=high (inverted)
 *
 ****/

#define CR_BID  0x20
#define CR_INT  0x10
#define CR_SEL  0x08
#define CR_INIT 0x04
#define CR_AUTO 0x02
#define CR_STB  0x01

/****
 * 10. Bidirectional Control Bit
 *
 * The bidirectional control bit is only supported on true bidirectional
 * ports - on other ports, it behaves like bits 7 and 6. On a proper
 * bidirectional port, setting this bit to '1' causes the outputs of the
 * buffer that drives pins 2 through 9 of the 25-pin connector to go into
 * a high-impedance state, so that data can be _input_ on those pins.
 *
 * In this state, values written to the data register will be stored in
 * the latch chip, but not asserted on the connector, and reading the
 * data register will yield the states of the pins, which may be driven
 * by an external device without stressing or damaging the port driver.
 *
 * Also note that on some machines, another port must be set correctly to
 * enable the bidirectional features, in addition to this bit.
 ****/
```

pport.map

```
 Start  Stop   Length Name                   Class
 00000H 00641H 00642H _TEXT                 CODE
 00642H 00642H 00000H C_ETEXT               ENDCODE
 00650H 006AAH 0005BH NULL                  BEGDATA
 006ACH 008B1H 00206H _DATA                 DATA
 008B2H 008B2H 00000H CONST                 CONST
 008B2H 008B2H 00000H _BSS                  BSS
 008B2H 008B2H 00000H STACK                 STACK

 Origin   Group
 0000:0   CGROUP
 0065:0   DGROUP

  Address         Publics by Name

 006A:005D       __8087
 0000:0000  Abs  __acrtused
 006A:0053       __BASE
 0000:0372       __chkstack
 0000:0389       __chkstk
 006A:0055       __datapar
 006A:0061       __dos
 006A:0063       __doserrno
 0000:37E8       __entry
 0000:0359       __exit
 006A:0059       __heapbottom
 6000:3720       __LCMP@
 C000:0000  Abs  __main
 7001:37DC       __okbigbuf
 006A:0063       __oserr
 006A:0061       __osmajor
 006A:0062       __osminor
 006A:0057       __pastdata
 006A:005B       __progpar
 006A:005F       __psp
 0000:37E5       __stack
 0065:0262       _edata
 0065:0262       _end
 006A:0063       _errno
 A000:37F7       _exit
 6000:397C       _fopen
 0000:37DE       _fread
 D000:37F0       _free
 0000:372E       _inb
 0000:03AE       _init_ports
 0000:03E6       _main
 7001:37C6       _outb
 0000:3728       _printf
 6000:37F4       _sbrk
 0000:3957       _usleep

  Address         Publics by Value
```

```
 0000:0000  Abs  __acrtused
 0000:0000  Abs  __main
 0000:0359       __exit
 0000:0372       __chkstack
 0000:0389       __chkstk
 0000:03AE       _init_ports
 0000:03E6       _main
 006A:0053       __BASE
 006A:0055       __datapar
 006A:0057       __pastdata
 006A:0059       __heapbottom
 006A:005B       __progpar
 006A:005D       __8087
 006A:005F       __psp
 006A:0061       __dos
 006A:0061       __osmajor
 006A:0062       __osminor
 006A:0063       __doserrno
 006A:0063       __oserr
 006A:0063       _errno
 0065:0262       _end
 0065:0262       _edata
 E001:37D5       __stack
 6000:397C       _fopen
 0000:37EC       __okbigbuf
 5000:3957       _usleep
 0000:3720       __LCMP@
 0000:372E       _inb
 8000:37E8       __entry
 7001:37C6       _outb
 6000:37F4       _sbrk
 A000:37F7       _exit
 0000:3728       _printf
 0000:37DE       _fread
 0000:37F0       _free

Program entry point at 0000:0100
```

Appendix D: PAL File

parallel.pal

```
module pport
TITLE 'pport PAL for Electric Legos'

pport device  'P22V10';

"Input pins

CLK             pin 1;

STROBE_         pin 2;
AUTOFEED_       pin 3;
INITIALIZE_     pin 4;
SELECT_INPUT_   pin 5;

RDY_BUSY_       pin 6;
INIT_           pin 7;
DONE            pin 8;

TIMER_IN        pin 9;

PIN_10           pin 10;
PIN_11           pin 11;
PIN_13           pin 13;

"Outputs

PIN_14          pin 14;

TIMER_OUT       pin 15;

ERR_            pin 16;
SELECT          pin 17;
PAPER_END       pin 18;
BUSY            pin 19;
ACKNOWLEDGE_    pin 20;
DIR             pin 21;
PROGRAM_        pin 22;
WS_             pin 23;


EQUATIONS

PIN_14.OE = 0;
TIMER_OUT.OE = 1;
ERR_.OE = !DONE;
SELECT.OE = !DONE;
PAPER_END.OE = !DONE;
BUSY.OE = !DONE;
ACKNOWLEDGE_.OE = !DONE;
DIR.OE = !DONE;
PROGRAM_.OE = 1;
WS_.OE = !DONE;
```

```
DIR = 1;

SELECT = 1;
PAPER_END = 0;

!PROGRAM_ = !INITIALIZE_;
!ERR_ = !INIT_;
BUSY = !RDY_BUSY_;
!WS_ := !STROBE_;
!ACKNOWLEDGE_ := !WS_;

end pport;
```