SANDIA REPORT

SAND97-0843 • UC-705
Unlimited Release
Printed April 1997

HyperForest: A High Performance Multi-Processor Architecture for Real-Time Intelligent Systems

Pablo Garcia, Jr., Juan P. Rebeil, Howard Pollard

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A05
Microfiche copy: A01

SAND97-0843
Unlimited Release
Printed April 1997

Distribution
Category UC-705

HyperForest: A High Performance Multi-processor Architecture for Real-Time Intelligent Systems

Pablo Garcia, Jr. and Juan P. Rebeil
Intelligent Systems Dept. II
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1006

Prof. Howard Pollard
Electrical Engineering and Computer Engineering Dept.
University of New Mexico
Albuquerque, NM 87102

Abstract

Intelligent Systems are characterized by the intensive use of computer power. The computer revolution of the last few years is what has made possible the development of the first generation of Intelligent Systems. Software for second generation Intelligent Systems will be more complex and will require more powerful computing engines in order to meet real-time constraints imposed by new robots, sensors, and applications. A multiprocessor architecture was developed that merges the advantages of message-passing and shared-memory structures: expandability and real-time compliance. The HyperForest architecture will provide an expandable real-time computing platform for computationally intensive Intelligent Systems and open the doors for the application of these systems to more complex tasks in environmental restoration and cleanup projects, flexible manufacturing systems, and DOE's own production and disassembly activities.

Table of Contents

HyperForest: A High Performance Multi-processor Architecture for Real-Time Intelligent Systems

I.- Introduction

Current computing platforms used in Intelligent Systems are expandable to a certain extent, but will not provide the floating-point throughput and real-time capabilities that future Intelligent Systems will require. World models will become more complex as larger sections of the real world are modeled with ever increasing resolutions. Collision avoidance may require that each point on the robot be compared with objects in the world model and the robot path altered accordingly. The number of arithmetic operations for a six degree-of-freedom (DOF) robot varies from 1500 with the inverse kinematics Newton-Euler formulation to over 6000 with resolved motion adaptive control methods. A robot sampling frequency as high as 5 KHz is anticipated, which translates to 30 MFLOPS of sustained floating-point throughput just for robot kinematics control. Sensors are used to obtain data about the environment. They need to be serviced in real-time and their data used for trajectory modification and world model updates. Sensor bandwidths vary from obtaining a couple hundred bytes per second from an ultrasonic sensor to obtaining digitized TV images at video rates. The efficient fusion of sensor data from different sources is what enables an Intelligent System to respond promptly in dealing with the real world. However, sensor data fusion requires additional real-time computing resources.

The use of redundant robot manipulators will demand more computationally intensive control algorithms due the higher number of links. Long reach robot arms, heavier payloads, and faster robot speeds will force kinematics control algorithms to include the effect of non-linearities such as gravity loading, Coriolis centripetal forces, and flexing of robot links among others. The addition of non-linearities will demand at least an order of magnitude increase in floating-point performance alone. World models will become more complex as larger sections of the real world are modeled with ever increasing resolutions. Collision avoidance algorithms may require that hundreds of points on the robot's surface be compared with objects in the world model and the robot path altered accordingly.

The goal of this project was to develop an expandable multiprocessor architecture that will satisfy the computational and real-time constraints of second generation Intelligent Systems.

II.- Background

A.- Message-Passing and Shared-Memory Architectures

The two main factors that determine system performance in a multiprocessor design are the processing power of the individual nodes and the delays caused by inter-processor communication. A crucial decision for multiprocessor design is the choice between shared memory and message passing architectures. Message passing structures, such as hypercubes and meshes, provide system expandability and programming abstraction but have the disadvantage of excessive overhead due to message routing. Shared bus architectures have the disadvantage of limited bus bandwidth, but have lower communication overhead. On the other hand, Intelligent Systems control programs are characterized by being decoupled between tasks but with high serial dependencies within a task (for example, kinematics control programs are decoupled from processing of sensory data.) Such characteristics result in poor hardware utilization when these programs are executed in architectures that aim at massive parallelisms such as hypercubes.

A multiprocessor architecture for Intelligent Systems must merge the advantages of message-passing and shared-memory structures to reduce delays in inter-processor communication and provide system expandability, real-time response times, an efficient interface to a wide variety of sensors, and the capability to share large data sets. It must be able to take advantage of fine (instruction level) and coarse (task level) parallelism in control programs.

As part of a Laboratory Directed Research and Development (LDRD) project, we developed a hybrid architecture that takes advantage of control program characteristics by being expandable to tens of high performance Digital-Signal-Processor (DSP) nodes. Furthermore, the floating-point engines of DSP processors are designed to exploit fine level parallelism in matrix and vector operations, both heavily used in control programs. This hybrid architecture will support both message passing and shared memory paradigms in hardware.

B.- Previous Work on HyperTrees

A team led by Prof. David Patterson at the University of California at Berkeley defined an architecture called X-tree in the late 1970's. This architecture is based on binary trees with extra links in each node. These extra links then could be used to form other types of structures to reduce the connection distance between nodes.

J. R. Goodman and C. H. Sequin presented a paper in 1981 titled "HyperTree: A multiprocessor interconnection topology." This paper described an interconnection topology for incrementally expandable multicomputer systems, which combined the easy expandability of tree structures with the compactness of the n-dimensional hypercube. The addition of n-cube links to the binary tree structure provided direct paths between nodes which have frequent data exchange in algorithms such as sorting and fast Fourier transforms (FFTs).

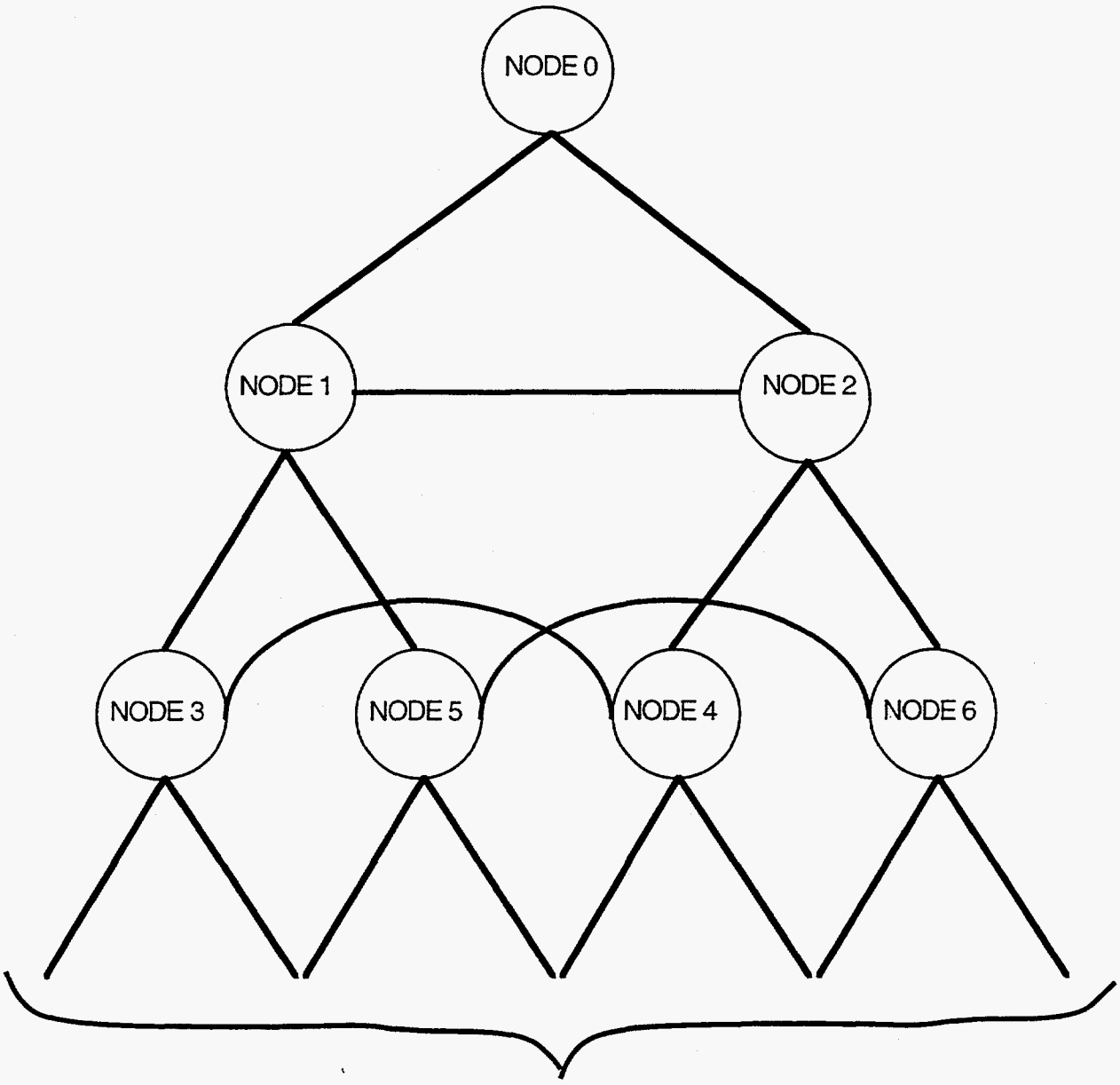


Figure 1.- 3-Level HyperTree

This paper presented a very interesting idea, and one that we applied to our work. The architecture they presented was a hybrid between a binary tree and a hypercube, although for simplicity, it is limited to hypercubes of size 1 and 2. The basic idea was to add communication links to the nodes of a binary tree, and to use the extra links to connect nodes at the same level in a hypercube. Figure 1 shows a 3 level HyperTree. In this figure you can see that with 4 links per node, 3 of the links are used for a regular binary tree structure, and the fourth port to connect the node to another node in the same binary tree level. A similar approach is used for 5 links per

node, but with a hypercube of size 2. The worst-case and average distances are better than in the simple binary tree, and fault tolerance is also improved by having alternate paths between nodes. The HyperTree can be easily expanded, unlike the hypercube and binary tree that require the addition of a large number of nodes.

Although the HyperTree can be expanded, the interconnections grow more complicated as more levels are added to it. Figure 2 shows the interconnection topology for a 6-level HyperTree.

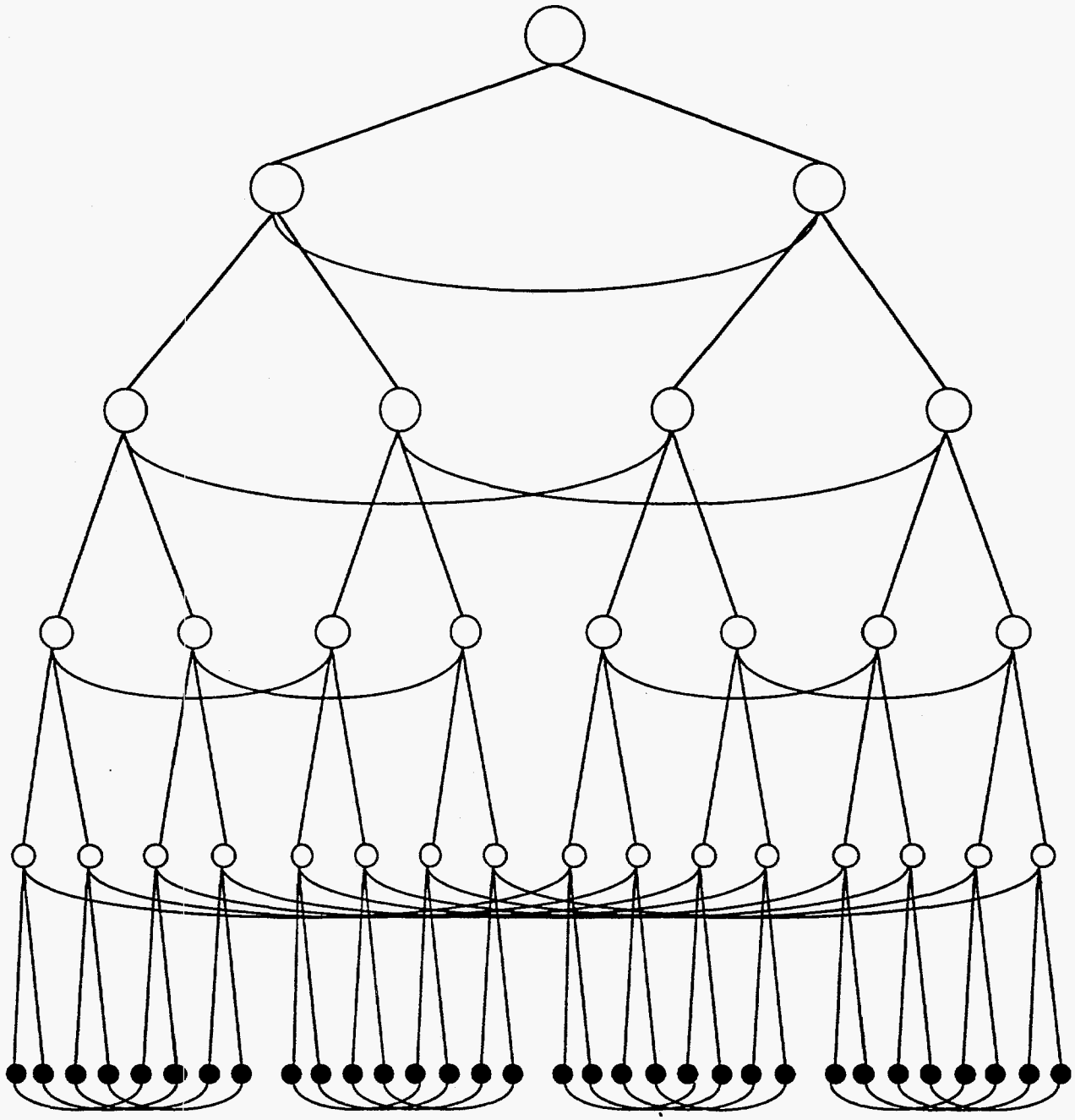


Figure 2.- 6-Level HyperTree

III.- HyperForest Architecture

A.- HyperForest Message-Passing Layer Definition

Although the HyperTree is very interesting, the are some limitations to how well it can be expanded and in how to implement it in hardware in a compact package. One of our goals was to have a VME-bus compatible implementation that could easily be expanded to more nodes if higher processing power was needed. We decided to limit the size of a HyperTree to three levels (7 nodes) and to grow it in the number of trees and not in growing a single tree. By doing it this way we had a collection of 3-level HyperTrees, which we very cleverly named a HyperForest.

In order to achieve this vision, we needed to make some modification to the basic HyperTree structure, specially in the number of communication links available at each node. We selected 6 communication links per node mainly because we had a Digital-Signal-Processor in mind for the hardware implementation and it had 6 parallel communication links available. Figure 3 shows a node and its communication links.

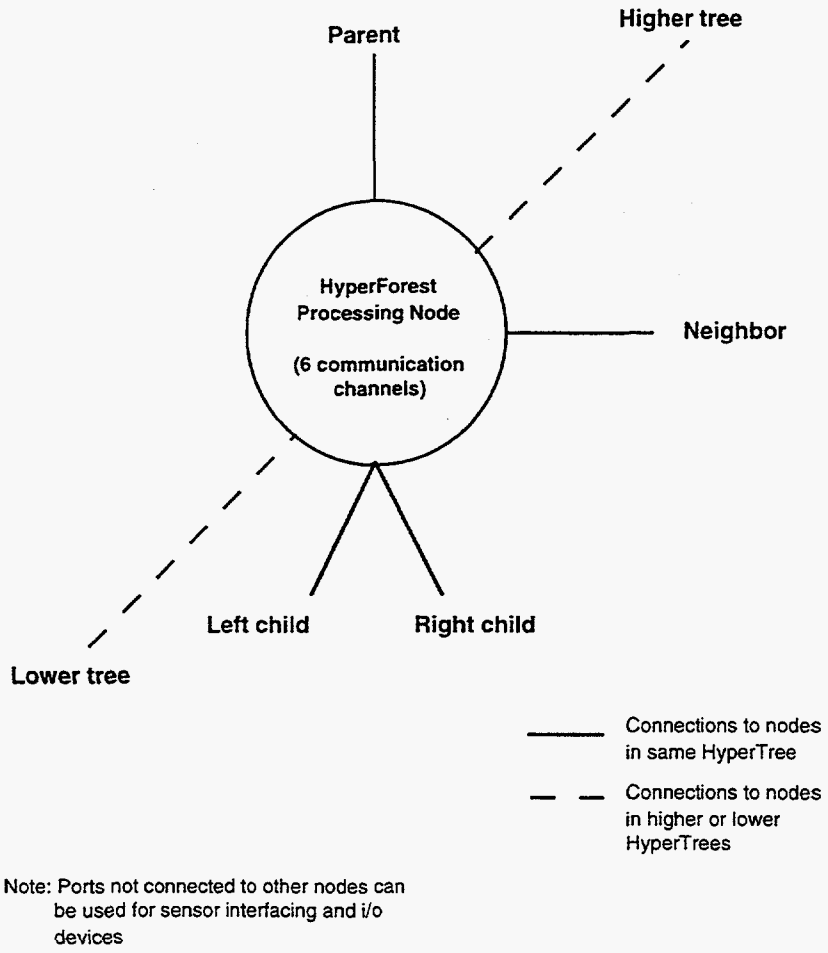


Figure 3.- HyperForest Node

For each node, the Parent, Left Child, and Right Child links are used to form binary tree structure; the Neighbor link connects to a node at the same level in the binary tree to form a hypercube size 1 at that level. Two other links, Higher Tree and Lower Tree connect the node to its counterpart node in other two trees in the HyperForest. Figure 4 shows our modified 3-level HyperTree with six communication links per node.

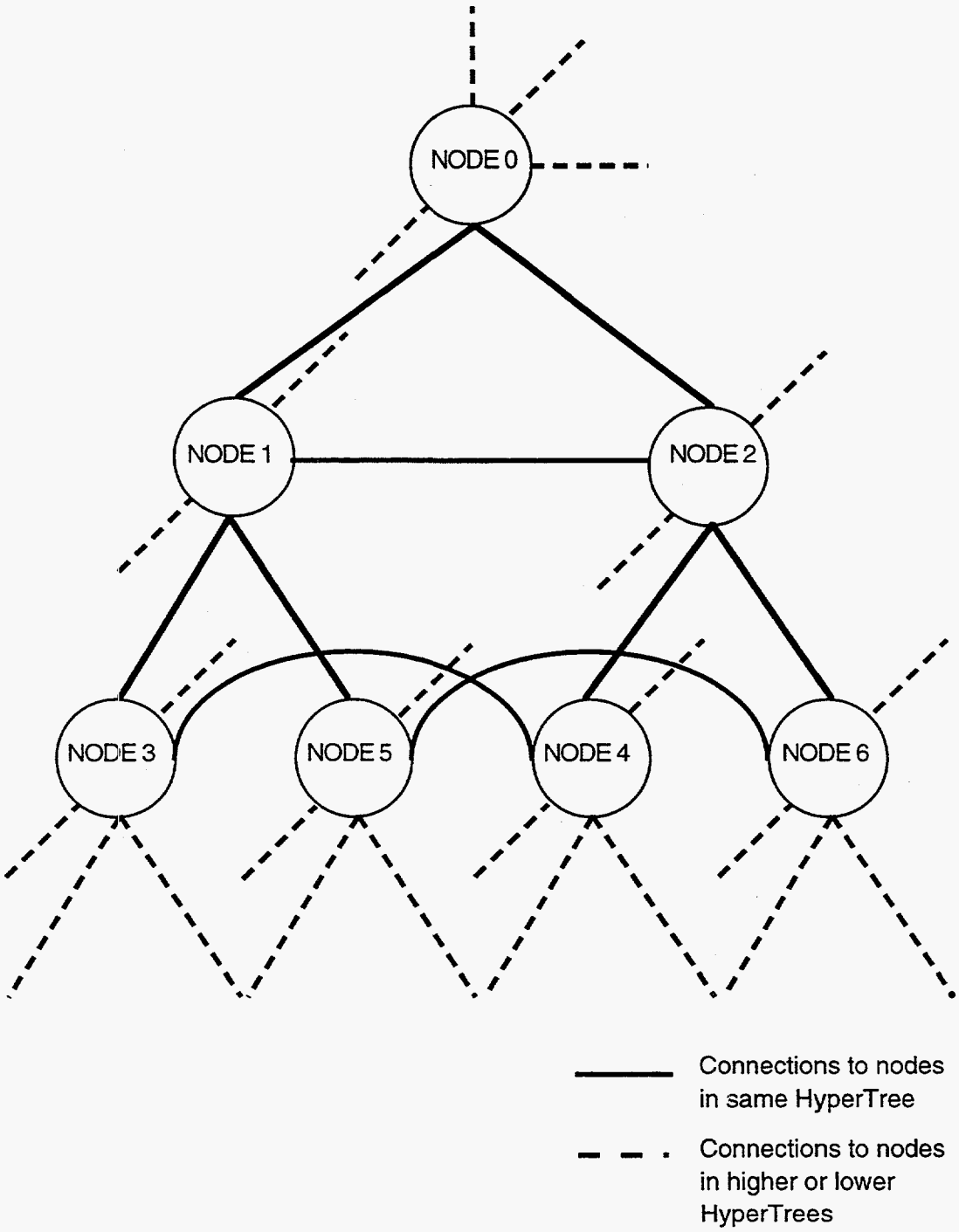


Figure 4.- Modified HyperTree with 6 Communication Links per Node

Creating a HyperForest multiprocessor with the modified HyperTrees is very easy. All it takes is to stack the HyperTrees on top of each other. All the communication links are bi-directional and there are a number of paths from any one node to another node, independently if they are in the same HyperTree or not. Figure 5 shows how the HyperTrees can be stacked.

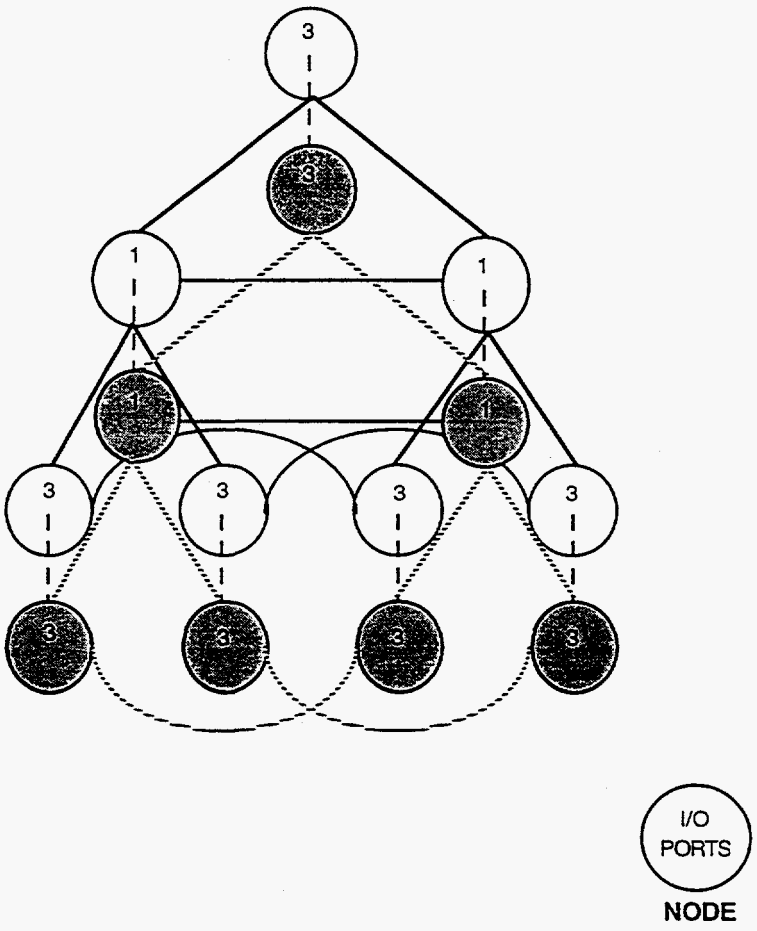


Figure 5.- HyperForest with 2 HyperTrees

In Figure 5, the number in each node represents the number of communication links available for expansion. These can be used to connect to more HyperTrees and/or for input/output (I/O) to devices such as sensors through a standard interface.

B.- HyperForest Shared-Memory Layer Definition

This is a very standard architecture where a number of processing nodes share a global bus to access memory and peripherals. We wanted this layer to be completely independent of the message-passing one. We selected the Texas Instrument TMS320C40 DSP processor because in addition to the 32-global memory bus it also has 6 8-bit 20Mbyte/sec bi-directional communication links designed for interfacing without any glue logic to other processors of the same type. Based on this decision, each node in the HyperForest would look like Figure 6.

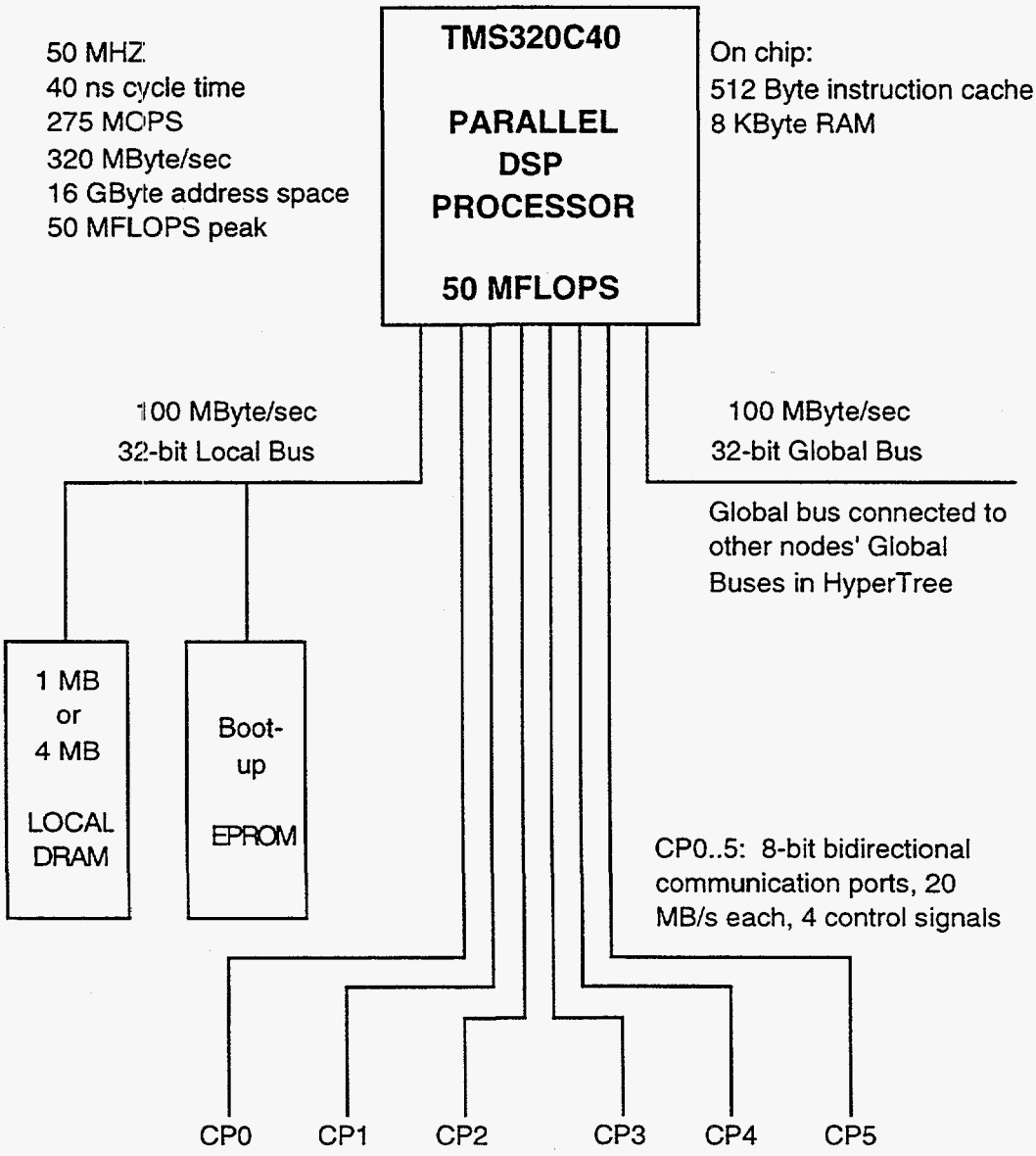


Figure 6.- HyperForest Node Based on Texas Instruments TMS320C40 Processor

Figure 7 shows the shared-memory layer for each of the HyperTrees in the HyperForest. Nodes in different HyperTrees can also communicate with each other by issuing a request to own the global bus of its HyperTree and using the VMEbus interface to communicate with a node in another HyperTree.

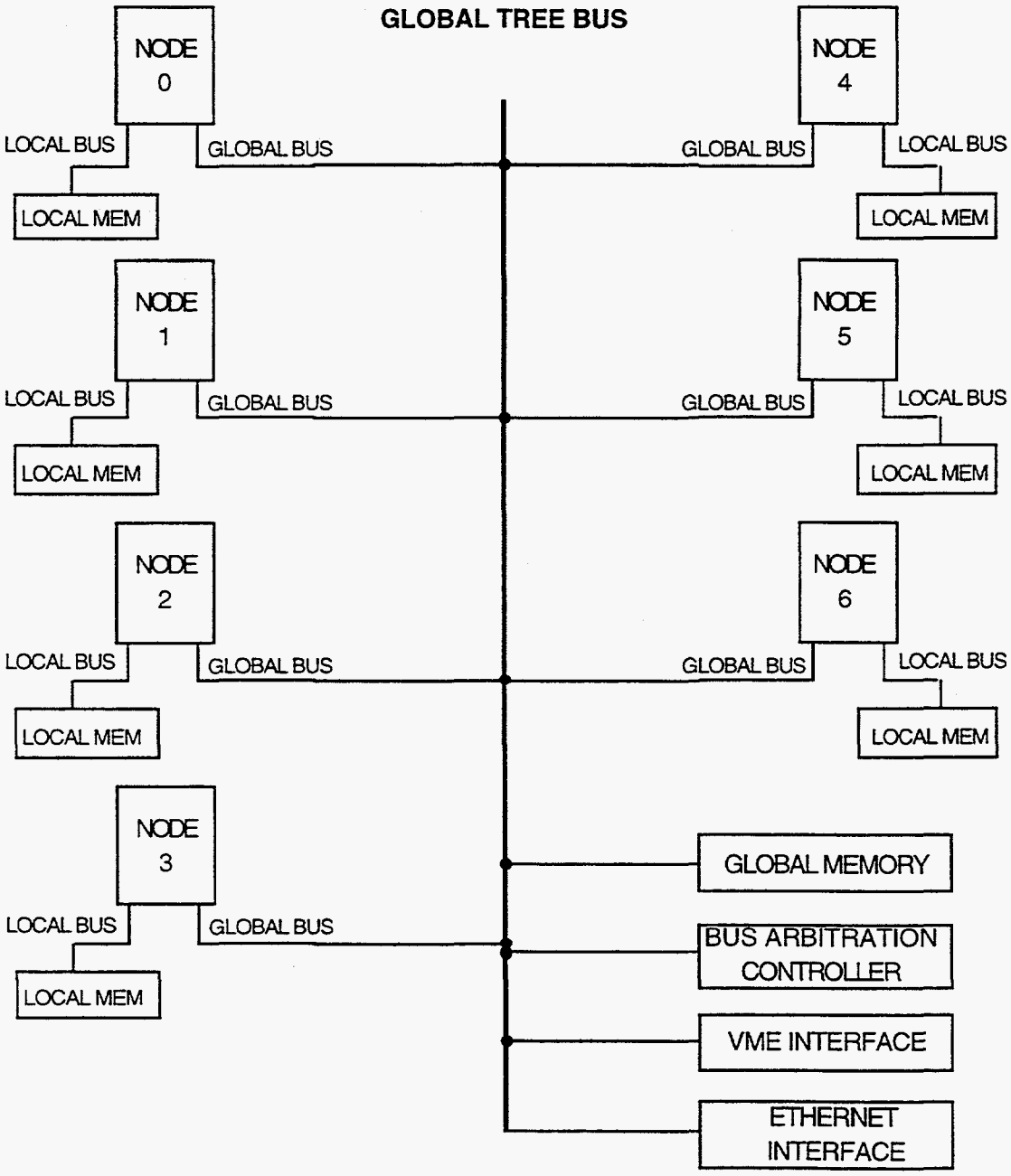


Figure 7.- HyperTree Shared-Memory Layer

C.- Development Environment

Figure 8 shows the development environment for HyperForest applications. It consists of a number of HyperTrees (1 or more) linked by the communication links coming in/out of each node as well as by each HyperTree's interface to a VMEbus. This industry standard in very popular in the computer control community and there is a large number of third-party boards available with processors, memory, video digitizers, motor controllers, digital and analog I/O etc. The VMEbus cage serves as host for custom-designed I/O boards connected to the HyperTree nodes via their communication links.

A Sun Microsystems SPARC-compatible VME board serves as the front end of the HyperForest Trees and it uses the real-time operating system VxWorks. The commercially available SPOX operating system was ported to each node in the HyperForest. It was selected because it supported the TMS320C40, had a small kernel, was real-time, and had communication routines compatible with VxWorks.

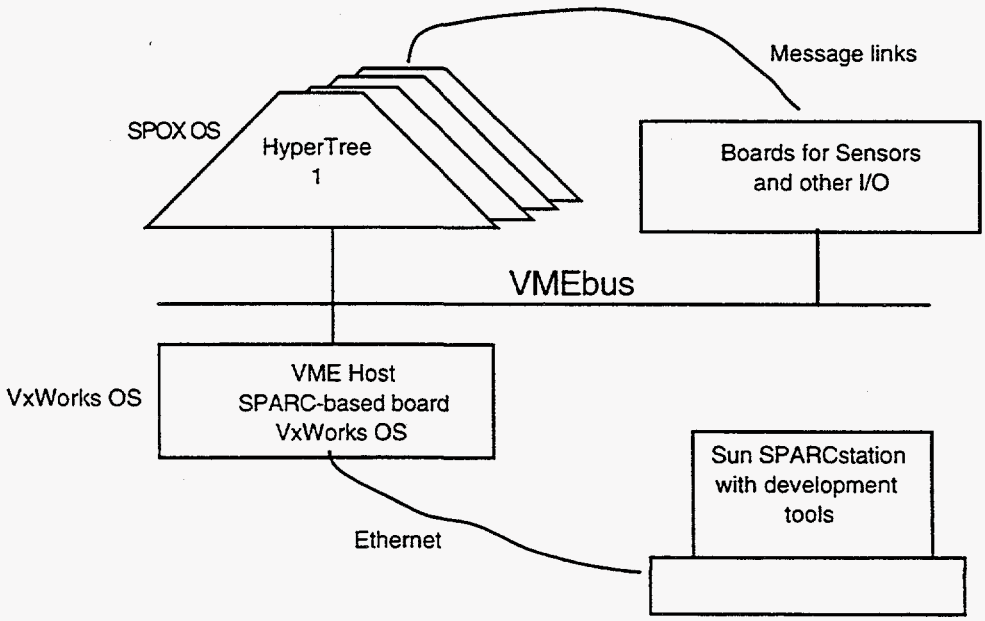


Figure 8.- HyperForest Development Environment

IV.- Comparison with Other Message-Passing Architectures

Hypercubes

Hypercubes are very popular with the massive-parallel computing community. But they do not scale very well in the sense that to take advantage of the architecture all the nodes have to be populated. In other words, the scaling is 2, 4, 8, 16, 32, 64, etc. If for example, an application requires 17 nodes, a 32 node computer will have to be build with all 32 nodes.
For an n-dimensional hypercube, the worst and average distances are given by:

$$Dw = n$$
$$Dave = n/2$$

for example:

n	#nodes	Dw	Dave
3	8	3	1.5
4	16	4	2
5	32	5	2.5
6	64	6	3

The n-dimensional hypercube architecture is not truly expandable and the nodes require additional links as the dimension grows. Incompletely populated hypercubes lack some of its characteristics (i.e. Dw and Dave shown above don't hold anymore).

Figure 9 shows a comparison of average Hamming distances for hypercubes and HyperForests of various sizes. From this plot we see that the HyperForest compares very well with hypercubes of up to 32 nodes.

Binary Trees

For a binary tree with n-levels, the distances are given by:

$$Dw = 2(n - 1)$$
$$Dave = 2(n - 1) - 2 + (2/N)$$

where N is the number of leaf nodes (i.e. $N = 2^{(n-1)}$).
For example:

n	#nodes	Dw	Dave
3	7	4	2.5
4	15	6	4.25
5	31	8	6.125
6	63	10	8.062

Binary tress are easily expandable and unbalanced trees still keep most of their properties. Unfortunately, Dw and Dave for binary tress are worse than those of hypercubes.

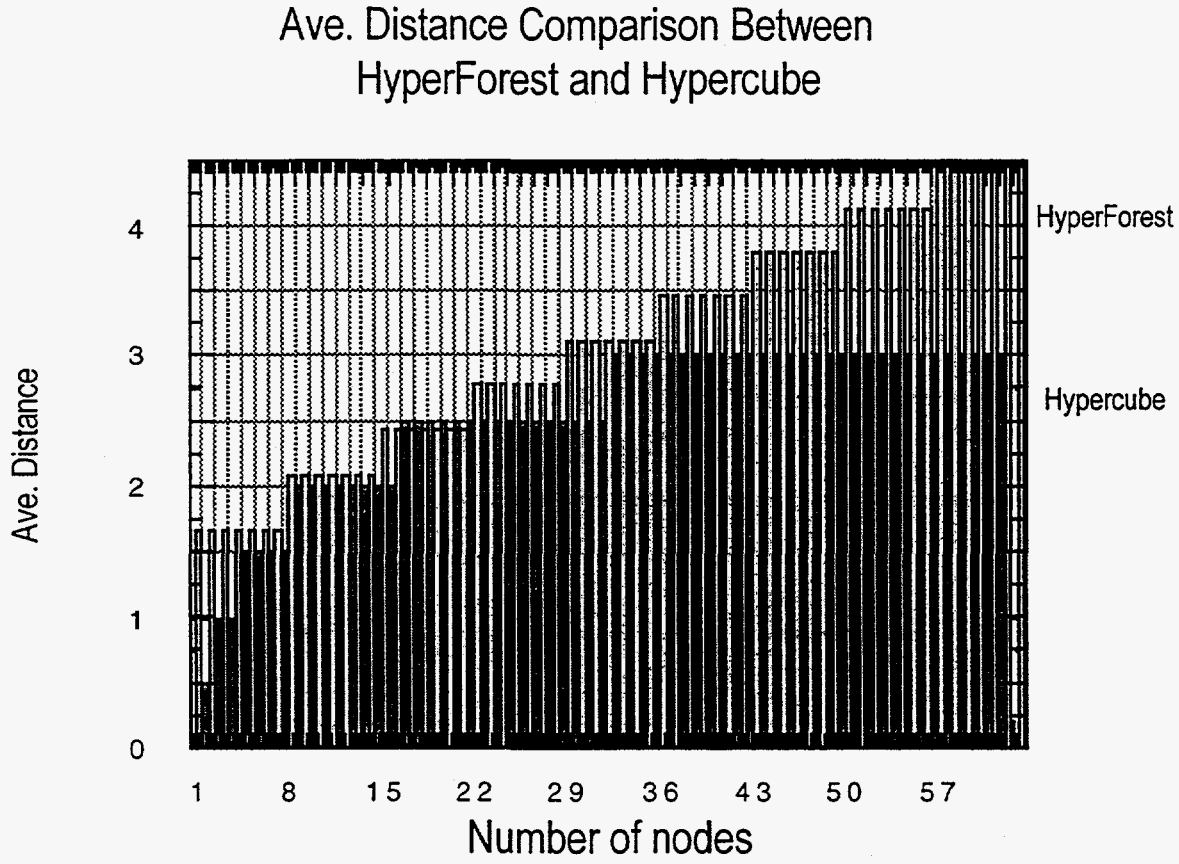


Figure 9.- Average Distance Comparison between HyperForest and Hypercubes

HyperTrees

HyperTrees use additional links to combine the hypercube and binary tree architectures into one. The additional links at the nodes provide redundant paths. Messages originating at leaf nodes never need to travel higher than half the height of three to reach any other leaf node. The Dw and Dave are taken from Goodman and Sequin's paper where n is number of levels and a one-dimensional hypercube is used at each level of the tree:

Dw = 1.5(n - 1) - 0.5((n-1) mod 2)

Dave = 1.25(n-1) - 1.33 + (4/3N) - 0.08((n-1) mod 2)

where N is the number of leaf nodes (i.e. N = 2^(n-1)). For example:

n	#nodes	Dw	Dave
3	7	3	1.5
4	15	4	2.5
5	31	6	3.4

These numbers are for the basic HyperTree as presented by Goodman and Sequin.

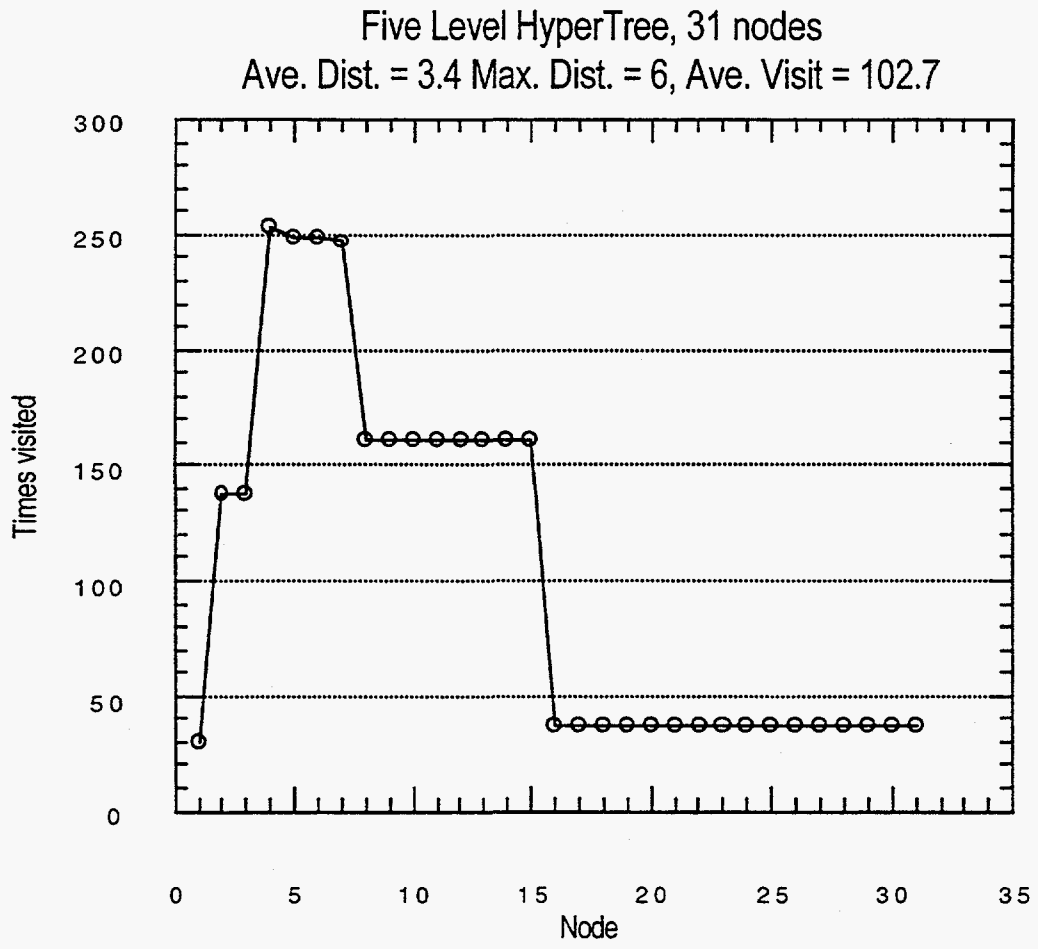


Figure 10.- Plot of Message Traffic on a Five-Level, 31-Node HyperTree

Figure 10 shows the result of a simulation of a five level HyperTree (31 nodes) with messages sent from each node to every other node. From this plot we see that nodes 4, 5, 6, and 7 had more traffic that all the other nodes and represent a bottleneck in this message-passing architecture.

HyperForest

The HyperForest architecture is a 3-d collection of modified HyperTrees (limited to 3-levels, 7 nodes). It keeps the characteristics of the HyperTree, but direct connections are available to other HyperTrees. Only 3-level HyperTrees are used: Dw and Dave for a 3-level HyperTree (7 nodes) are identical to those of a 3-dimensional hypercube (8 nodes).

Let T be the number of modified HyperTrees in a HyperForest machine, then:

$Dw = (T - 1) + 3$

Dave = computed in simulation

For example:

T	#nodes	Dw	Dave
1	7	3	1.5
2	14	4	2.03
3	21	5	2.43
4	28	6	2..78
5	35	7	3.12
6	42	8	3.46

We see that HyperForests with up to 4 trees (28 nodes) compare favorably with up to 5-dimensional hypercubes (32 nodes) in terms of Dw while being more easily expanded. Calculations for HyperForest Dw did not take into account the use of shared-buses for each tree. Figure 11 shows the result of a simulation of a four tree HyperForest (28 nodes) with messages sent from each node to every other node.

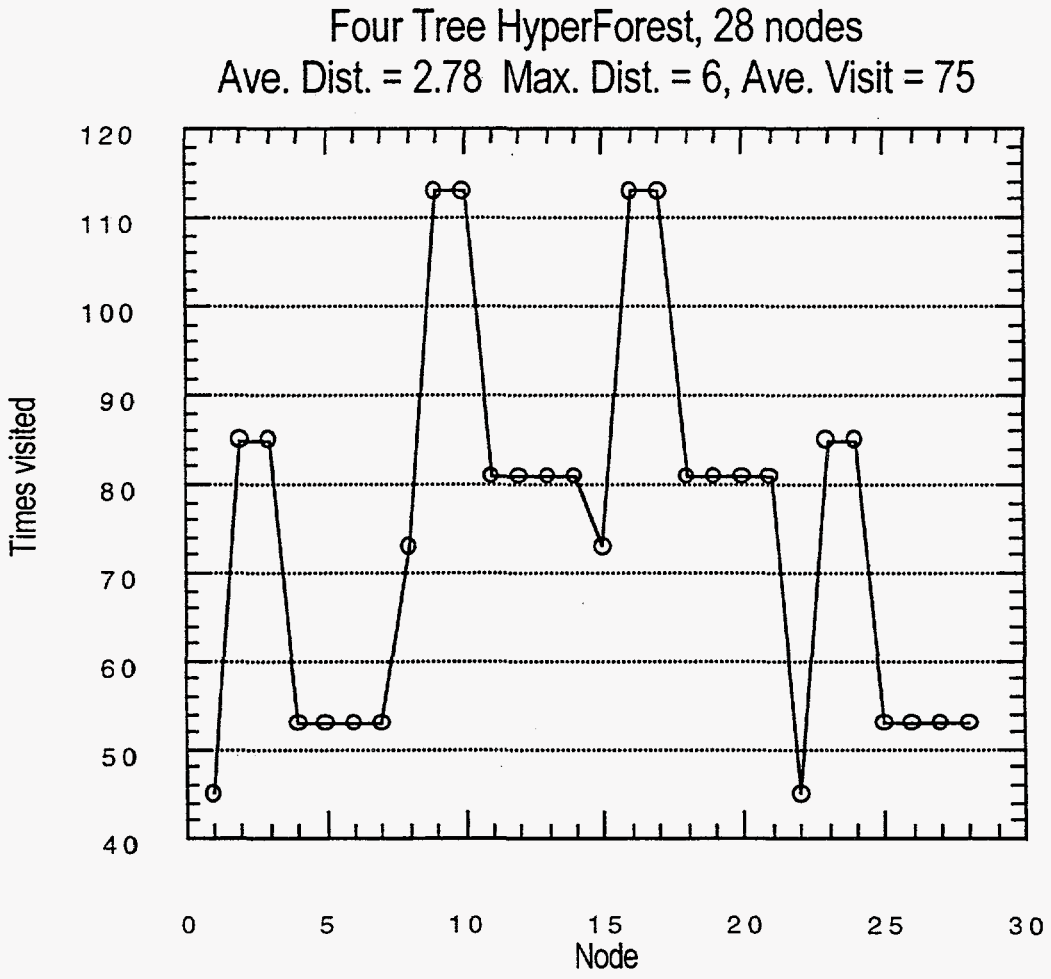


Figure 11.- Plot of Message Traffic on a Four-tree, 28-Node HyperForest

The routing algorithm for the messages consisted on taking the shortest path to the destination node if it was in the same HyperTree; or the shortest path to the node in the source HyperTree that was directly above or below the destination node and then traveling up or down to the destination node. With this algorithm, we see from the plot in Figure 11 that nodes 2 and 3 (the children of each root node) had more traffic and could become bottlenecks. This can be solved by either taking alternate routes if the traffic through those nodes is heavy or by committing an extra communications link to connect nodes 2 and 3 and double the available bandwidth between them from 20MBytes/sec to 40 MBytes/sec.

V.- Performance Summary

Floating Point Performance

50 MFLOPS peak/node
350 MFLOPS peak/tree
1400 MFLOPS for a 4-tree HyperForest

Inter-processor communication

20 MByte/s per message-passing link
9 links (180 MByte/s) for communication within each HyperTree
100 MByte/s global shared bus within HyperTree
7 links (140 MByte/s) to each of two neighboring trees
Message passing Hamming distances:

# HyperTrees	Worst case	Ave distance
1	3	1.67
2	4	2.08
3	5	2.43
4	6	2.78
5	7	3.12
6	8	3.46
7	9	3.79
8	10	4.13
9	11	4.46

Memory

On-board each node CPU
- 512 Bytes instruction cache
- 8 KBytes single cycle RAM

Local Memory for each node
- 32-bit local 100 MByte/s bus
- 16 KByte fast copy back cache
- 4 MByte DRAM

Global Memory for each tree
- 32-bit global 100 MByte/s bus
- 4-16 MByte page mode DRAM

Input/Output

Sensor interfaces
- 10 dedicated links (200 MByte/s) from each HyperTree
- Memory-mapped on 100 MByte/s global bus

Other
- 40 MByte/sec VME interface on global bus

Table 1.- HyperForest Performance Summary

HyperForest	One HyperTree	Two HyperTrees	Three HyperTrees
Floating Point (peak MFLOPS)	350	700	1050
Data Accesses (peak MOPS)	595	1190	1785
Interprocessor Bandwidth MBytes/s (channels)	180 (9)	500 (25)	820 (41)
20 MByte/s Channels Available for I/O	24	34	44
Max Number of Nodes	7	14	21

VI.- Hardware Implementation

The basic objective of the hardware portion of the HyperForest project is to create a hardware platform that will implement the HyperForest concept in such a way that the system can be created in a reasonable size, be interfaced to other system components, and still maintain the characteristics of a HyperForest system. This will allow the exploration of both shared memory and message passing paradigms on the same platform, and also permit high speed computations needed for real time control and robotics applications.

The block diagram of the implementation is shown in Figure 12. This block diagram demonstrates the results of a variety of design decisions that dealt with the realities of creating a HyperForest system utilizing commercially available TMS32C040 modules. The basic building block of the HyperForest implementation is a TIM module, which has the following characteristics:

Processor: Texas Instruments TMS32C040
Salient TMS32C040 Characteristics:
  Two independent memory bus systems: 32 bits Address; 32 Bits data
  Built in floating point capabilities
  Six byte-wide communication links with individual DMA controllers
Memory: 8 MBytes total; 4 MBytes on bus not available externally
  and 4 MBytes on bus which is visible externally
Memory interconnect: bus system with 32 bits address, 32 bits data
Message passing links: Six byte-wide communication links with DMA

The connection scheme which is demonstrated in Figure 12 includes both the global bus and the individual point-to-point connections. As seen on the diagram, the global connection is common to all modules, and contains the control lines (BUSCONT), the address lines (ADRBUS), and the communications is the VME interface, which allows connection of this bus to a VME bus based system.

The point-to-point links which are included in the diagram connect the various TIM modules together into a HyperForest connection, which consists of a binary tree organization augmented with additional links allowing further expansion of the system. Node 1 is connected to the left child via link 1 and the right child via link 2. Since Node 1 is in the root node position, there is no connection to a parent. Nodes 2 and 3 are connected to the parent (which in this case is Node 1 for both Nodes 2 and 3) using Link 3, while again Link 1 is used to connect to the left child and link 2 is used to connect to the right child. The four nodes on the next level, nodes 4, 5, 6, and 7, are connected to their parent nodes by using link 3. Since there are no designated children nodes for these four nodes, the remaining links can be used as needed to connect to other modules on other HyperForest boards. At each level of the binary tree (except the root level) nodes are connected to another node at the same level. Nodes 2 and 3 are connected together on their level, while Nodes 4 and 6 are connected together, and Nodes 5 and 7 are connected together.

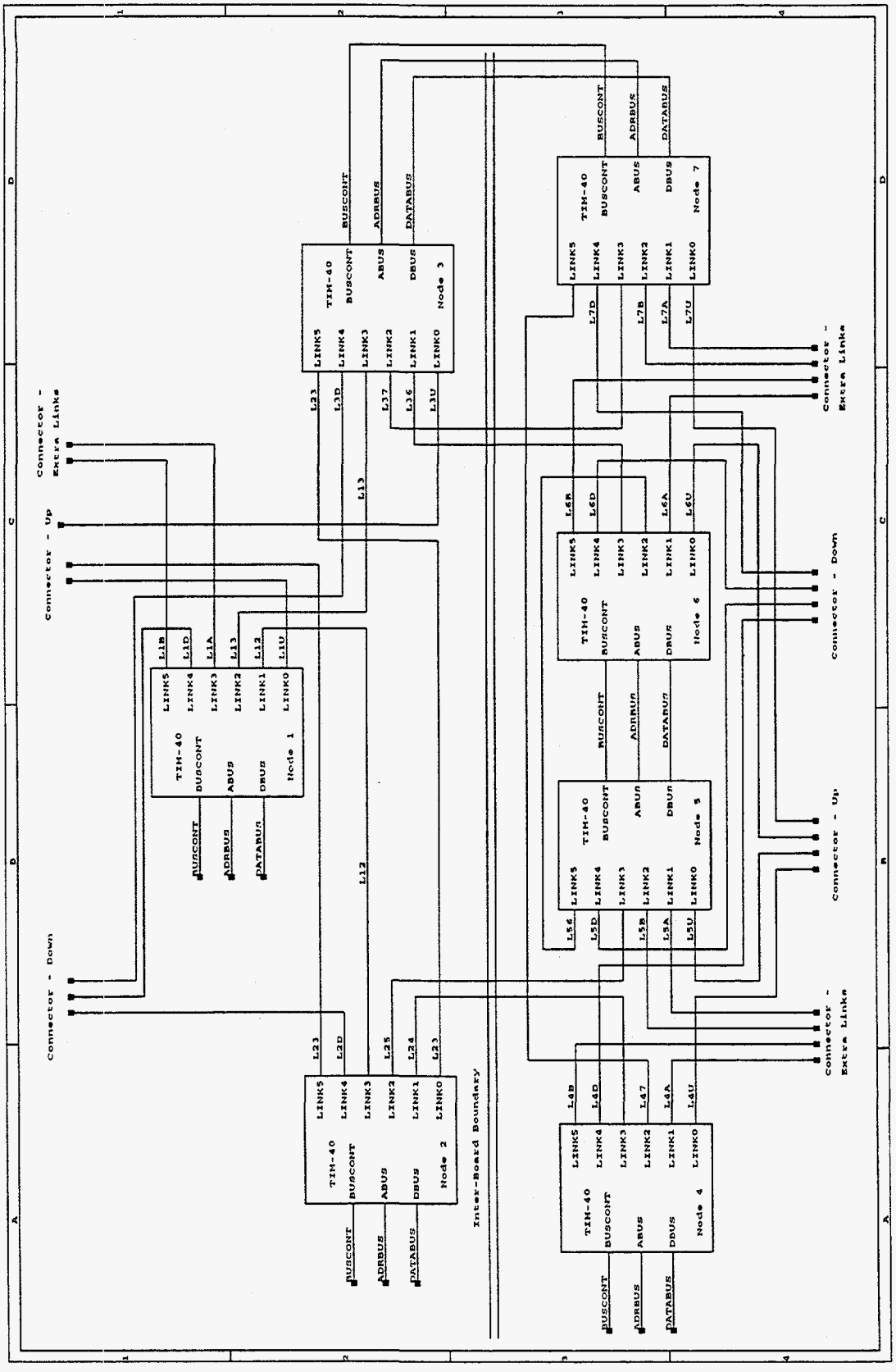


Figure 12.- Block Diagram of HyperForest's Hardware Implementation

Another salient characteristic of the HyperForest connection mechanism is the connection of the nodes of the binary tree organization to nodes of a similar binary tree. Note that the organization represented in Figure 12 has sufficient links available to allow this connection mechanism to be achieved. Each of the nodes in the diagram has sufficient links to be connected to two other nodes, one in each of similar binary trees adjacent to the binary tree shown in the diagram. The remaining links can be used for I/O purposes if needed.

Another characteristic of the implementation of the HyperForest system is the division of the nodes to fit physically on two boards. In order to accomplish this, the boards must be connected together with a connection system capable of providing a path for both the global bus and the links connecting the second and third levels of the binary tree together.

In order to test implementation techniques for the system, a bread-board was implemented which helped to isolate some of the problems and give experience with the various system components. This breadboard was built with wire-wrap techniques so that connections could be easily moved to identify the effect of different connection schemes. The wire-wrap techniques worked well, but presented a challenge to connect to the TIM modules, since the Hiroshe connectors used for TIM module interconnects were not easily mated to a wire-wrap scheme that utilized posts mounted on 0.1" centers. Nevertheless, adapters were made that allowed the TIM modules to mate to the wire-wrap board, and the project proceeded. The breadboard contains two TIM modules, a moderate amount of static RAM, the chips involved in the VME interconnection scheme, the chips involved in the boot up sequence, and the programmable chips to control the bus interaction in the system. The breadboard also contains connectors to check the point-to-point message passing connections, and a JTAG interface for debug purposes. Utilizing the breadboard, hardware solutions were checked for the programmable devices to control bus interaction within the board, as well as the connection to the VME bus. Also included in the checkout methodology were implementations for power-on reset, for forced reset, for debugging methods using external compilers and loaders through the JTAG interface, and the impedance matching techniques needed to improve the fidelity of the signals involved in the various transactions. All of these individual test areas contributed to a fuller knowledge of the challenges facing the creation of the printed circuit board version of the system.

The knowledge and experience garnered in the creation of the breadboard version of the test set, as well as the results of simulations and the work done with programmable logic, were needed to create the final two-board set of printed circuits which together implement a HyperForest subsystem. Rather than give a block diagram of the board, Figure 13 gives a layout of the first of the two boards used in this implementation. The diagram shows the results of many of the design decisions that were needed to create a printed-circuit version of the HyperForest. The various elements of the system included on the board are:

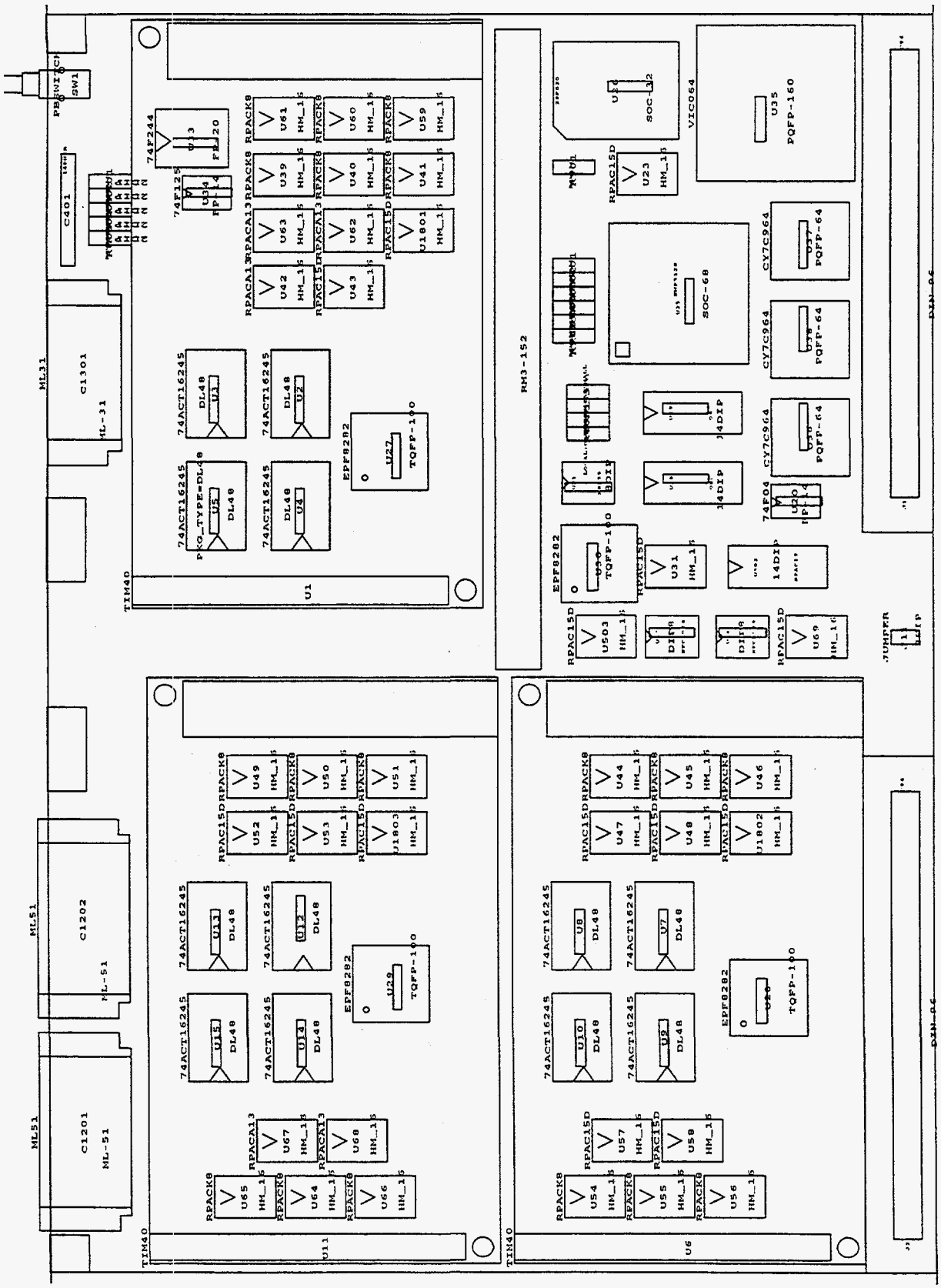


Figure 13.- HyperTree Board 1 Layout: 3 Nodes Plus Control Logic

A) The VME interface. The boards have been created in what is known as a "2-high" VME form factor, so that it can easily reside in commercially available card cages, and so that it can easily interface to other systems which abide by the VME bus protocol. Thus, the 96-pin DIN edge connectors are used to connect to VME address, data, and control lines for the VME interconnection. The integrated circuits involved in this interaction include a VIC064 and three CY7C964 from Cypress Semiconductor, as well as a EPF8282 FLEX programmable controller from Altera.

B) The bus control system. Each of the TIM modules, as well as the VME interface already mentioned, interfaces to an internal 32 bit data bus, which has associated with it a 32 bit address bus and appropriate control lines. One of the EPF8282 FLEX programmable controllers from Altera has been dedicated to the management of the transactions on this bus. Hence, the programmable controller must handle arbitration for the bus, and interface with each TIM module and the VME subsystem to assure that only one module attempts to obtain control of the bus at any one time.

C) The boot-up circuitry. Normal boot-up of the system is achieved by establishing in a TIM module a program which can then be used to load even larger and more complex programs. The TIM modules have been created in such a way that upon reset they are waiting for a program to be loaded through a point-to-point port. Therefore, the boot up procedure is to send to such a port the sequence of commands necessary to set up a common bootstrap program. An Intel 28F020 FLASH ROM (256 KByte) provides the storage needed for the download program storage, and an Altera 5128 programmable controller provides the control for the process, including the addressing and sequencing of activities.

D) TIM modules for Nodes 1, 2, and 3. The first board provides the space necessary for three of the seven nodes needed for a HyperForest subsystem. Each of the nodes contains not only the required TIM module, but also an EPF8282 FLEX programmable controller to control the interaction of the node with the internal global bus, 74ACT16245 transceivers to isolate the global bus activity from the TIM module itself, and the appropriate resistor networks to match impedance and provide improved signal quality for the point-to-point connections. The use of the transceivers allows the various TIM modules to operate independently, isolating the global bus connection of each module from the corresponding connections of other modules.

E) Interconnection methods. There are three basic connector systems in use on the subsystem. Already mentioned is the set of two 96-pin DIN connectors used for the VME connection; this set of connectors also supplies the power used on the board. Another connection which is needed is the connection between the two boards; this is accomplished with a 152 pin connector which utilized three rows of pins. This quantity of interconnections is needed for the address, data, and control lines of the global bus, plus the point-to-point connections needed between the second and third layer of the binary tree structure of the HyperForest architecture. The third set of connectors is included at the edge of the system, where 31- and 51- pin connectors are used to provide connections for 2- and 3-sets of point-to-point port signals. This allows the board to be connected to other HyperForest subsystems in the HyperForest system interconnect scheme.

F) Miscellaneous glue subsystems. These include a common oscillator for all TIM modules, a 64 MHz oscillator needed by the VME bus subsystem chip, reset circuitry, including both power-up reset and operator-initiated reset, JTAG interface circuitry, and assorted pullup and impedance matching resistances.

Included in Appendix A are the eighteen sheets of schematics which define the first board. Sheet 1 contains the information for Node 1, including the TIM module, its interconnects to point-to-point connections, bus connections, and control lines. It also contains the bus transceivers with appropriate control lines. Sheets 2 and 3 contain the same information for Nodes 2 and 3. Sheet 4 contains the reset circuitry, the oscillators, assorted pullups, and the serial PROMs that establish the circuitry of the RAM based programmable logic on the board. Sheet 5 contains the ROM for the boot code and the controller which controls its interaction with Node 1. Sheet 6 contains the programmable controllers that control the bus interaction for Nodes 1, 2, and 3. Sheet 7 contains the programmable controller that handles bus arbitration and access control to the bus for the nodes and the VME bus controller. Sheet 8 contains the JTAG header and associated circuitry. Sheets 9 and 10 contain the VME bus interface circuitry, while sheet 11 contains the connectors for the VME bus interaction. Sheets 12 and 13 contain the communication port connectors, while sheet 14 shows the connector which is used to connect the two boards together. Sheets 15, 16, and 17 show the impedance networks to minimize noise on the point-to-point connections. Sheet 18 contains assorted pullups needed for the implementation, and sheet 19 shows the bypass capacitors used on the board.

A layout of the second board of the 2-board HyperForest subsystem is shown in Figure 14. This board duplicates many of the portions of the first board, yet has some unique features:

A) Bus system. The control of the bus is handled by the circuitry on the first board, and hence the second board does not have responsibility for control. But it does have the appropriate circuitry at each TIM module to participate in the interactions required for global bus transfers.

B) TIM modules for Nodes 4, 5, 6, and 7. As with the first board, with each of these TIM modules includes a EPF8282 FLEX programmable controller to control the bus interaction and appropriate bus transceivers and resister networks to handle interface with other system components.

C) Interconnection system. The same connector types are present on the second board, and they have most of the same responsibilities. The 152 pin inter-board connector is used for global bus interactions and point-to-point connection with the other nodes of the HyperForest subsystem. The edge is populated with 31- and 51-pin connectors for 2- and 3-sets of point-to-point connections used in HyperForest point-to-point interconnections. Also, two 96-pin DIN connectors are available as per VME specification. However, the only use for these connectors on this board is to provide power for the circuitry.

Included in Appendix B are the twenty sheets of schematics which define the second board. Sheet 1 contains the information for Node 4, including the TIM module, its interconnects to point-to-point connections, bus connections, and control lines. It also contains the bus transceivers with appropriate control lines. Sheets 2, 3, and 4 contain the same information for Nodes 5, 6, and 7. Sheet 5 contains the clock drivers and the JTAG chain. Sheets 6 and 7 contain the programmable controllers that control the bus interaction for Nodes 4, 5, 6, and 7. Sheet 8 shows the connector which is used to connect the two boards together. Sheets 9, 10, 11, and 12 contain the connectors used for the communication links, while sheet 13 contains the VME bus connectors which supply power to the board. Sheets 14, 15, 16, 17, and 18 contain resistor networks for impedance matching on the point-to-point connections for the nodes. Sheet 19 contains bypass capacitors used on the board, and Sheet 20 contains pullups used by integrated circuits on the board.

Four sets of these boards were fabricated, and are ready for checkout. The checkout needs to verify that the programmable logic can appropriately control the bus interaction, the arbitration, and the interaction with the VME logic. The checkout also needs to create the code section necessary for the download sequence to load executable code into the system. Once this has been established, techniques useful for real-time operation in control applications can be implemented.

VII.- Communicating With I/O and Peripherals

In the industrial automation world, recent years have seen a falling from favor of the traditional method of wiring sensors, actuators, and other devices directly to a control unit such as a Programmable Logic Controller (PLC) or a personal computer (PC). Discrete wiring is being replaced by "fieldbuses" or "devicebuses" that allow the connection of many devices (equipped with the appropriate communication electronics) to a single cable bus via tees in much the same manner as thin-net ethernet. The rationale for this approach is that the higher cost of the new communications-enabled devices is more than offset by the savings in wiring in applications where the slower response of the new devices (due to them sharing a single communications channel) is not an issue.

This trend towards devicebuses in the industrial automation arena has been fortunate for HyperForest. Whereas the traditional approach to connecting peripheral devices would require a different hardware and software interface to the HyperForest architecture, all types of devices on a devicebus can be accessed through a single hardware/software interface to the devicebus.

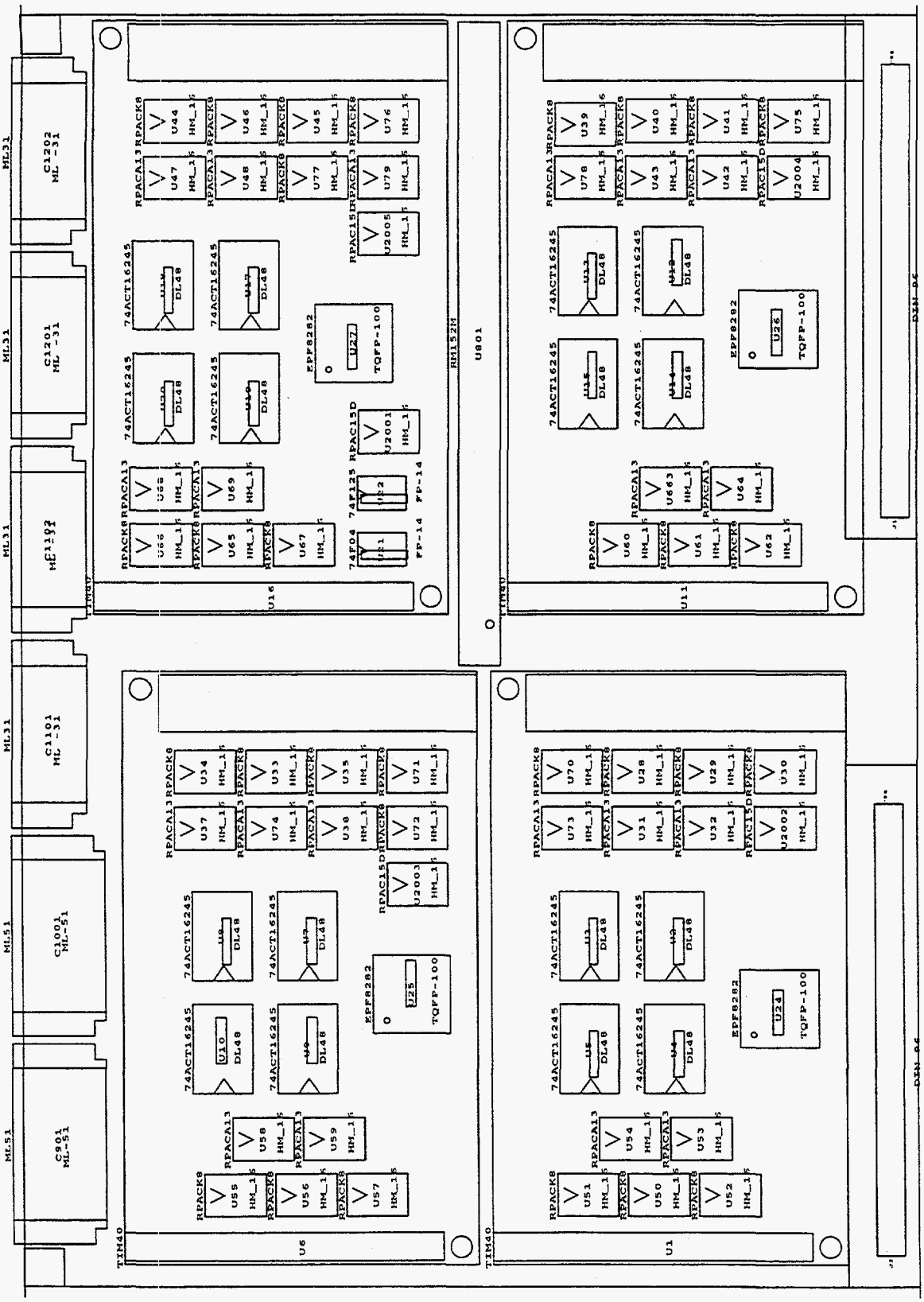


Figure 14.- HyperTree Board 2 Layout: 4 Nodes

At the time of this writing, there are many different competing fieldbus and devicebus standards. The more complex "fieldbus" standards such as Profibus are meant for communication with complex devices common in the process industries. Intermediate complexity "devicebuses" such as DeviceNet are intended for discrete manufacturing applications (e.g., assembly) where the devices range in complexity from a simple mechanical switch to moderately sophisticated devices such as pneumatic manifolds and motor drives. At the low end of the complexity scale, "bitbuses" such as ASI can only communicate with simple on/off sensors and actuators.

We selected DeviceNet for use with HyperForest because it is the standard with the most industry support within the US and the range of devices available for it is ideal for most robotics and automation applications

DeviceNet is based on an automotive communications bus called Controller Area Network (CAN). CAN is a differential serial bus developed in Germany by Bosch for Mercedes Benz and BMW as a means for the various electronic control units in their automobiles to communicate with each other. Due to the economies of scale in the auto industry, the integrated circuits (ICs) needed to implement CAN communications quickly became plentiful and inexpensive. Allen Bradley (AB) in the US used CAN hardware with additional hardware and software protocol standards to create the open, but proprietary, DeviceNet standard. AB later donated the standard to a consortium of industrial automation companies called the Open DeviceNet Vendors Association (ODVA) to allay industry fears that AB would have an unfair advantage in the DeviceNet market or that it would close the standard once it became popular.

At the lowest level, DeviceNet is a 5-wire bus (2 differential serial lines, 2 lines for 24 VDC, and a shield) that can operate at 125 kbaud, 250 kbaud, or 500 kbaud. The message packets are short; they consist of a header, up to 8 bytes of data, and a trailer. Devices transmit only when the bus is quiet; in the event 2 devices start transmitting simultaneously, the first device to transmit a 1 but detect a 0 on the bus immediately stops transmitting, ceding control of the bus to the other device (as it turns out, the one with the lower address). Each device receives every message on the DeviceNet bus, but usually only processes packets intended for it. A given DeviceNet can have up to 64 devices with different addresses, but this is not a serious limitation in most applications since each device can have many I/O points and/or functions.

DeviceNet uses a master-slave control scheme. The DeviceNet master is a board or module that plugs into the bus of the system controller. These boards are available for several bus architectures including VME, ISA, PCMCIA, PC104, as well as for several popular PLCs. The variety of available DeviceNet slave devices is extensive: discrete I/O, analog I/O, photoelectric sensors, AC motor starters, AC and DC motor drives, servo and stepper motor controllers, pneumatic manifolds, RS232 translators, barcode scanners, potentiometers, man-machine interfaces (MMIs), encoders, vacuum instruments, etc. Figure 15 shows a typical DeviceNet network installation.

The typical application usually has a single DeviceNet master, in our case, the HyperForest VME chassis with an S-S Technologies DeviceNet VME interface board. The DeviceNet board

consists of a CPU that is loaded with scanner software that continually queries all known devices on the DeviceNet and writes the results in a section of dual port RAM that is accessible to the HyperForest via the VME bus. DeviceNet output devices are controlled in the same manner; HyperForest writes the appropriate data to the DeviceNet board's dual port RAM, which the scanner process continually scans for changes and sends the appropriate commands encoded in packets to the DeviceNet wire. The high baud rates available, together with the short packet lengths, would seem to suggest that DeviceNet can have a high refresh rate. Unfortunately, the master-slave protocol slows performance considerably. Typical refresh rates tend to run in the tens of milliseconds, making DeviceNet inappropriate for some of the more demanding real-time applications. Also, it is arbitrated; hence, inderteminate (i.e. not applicable to hard real-time applications).

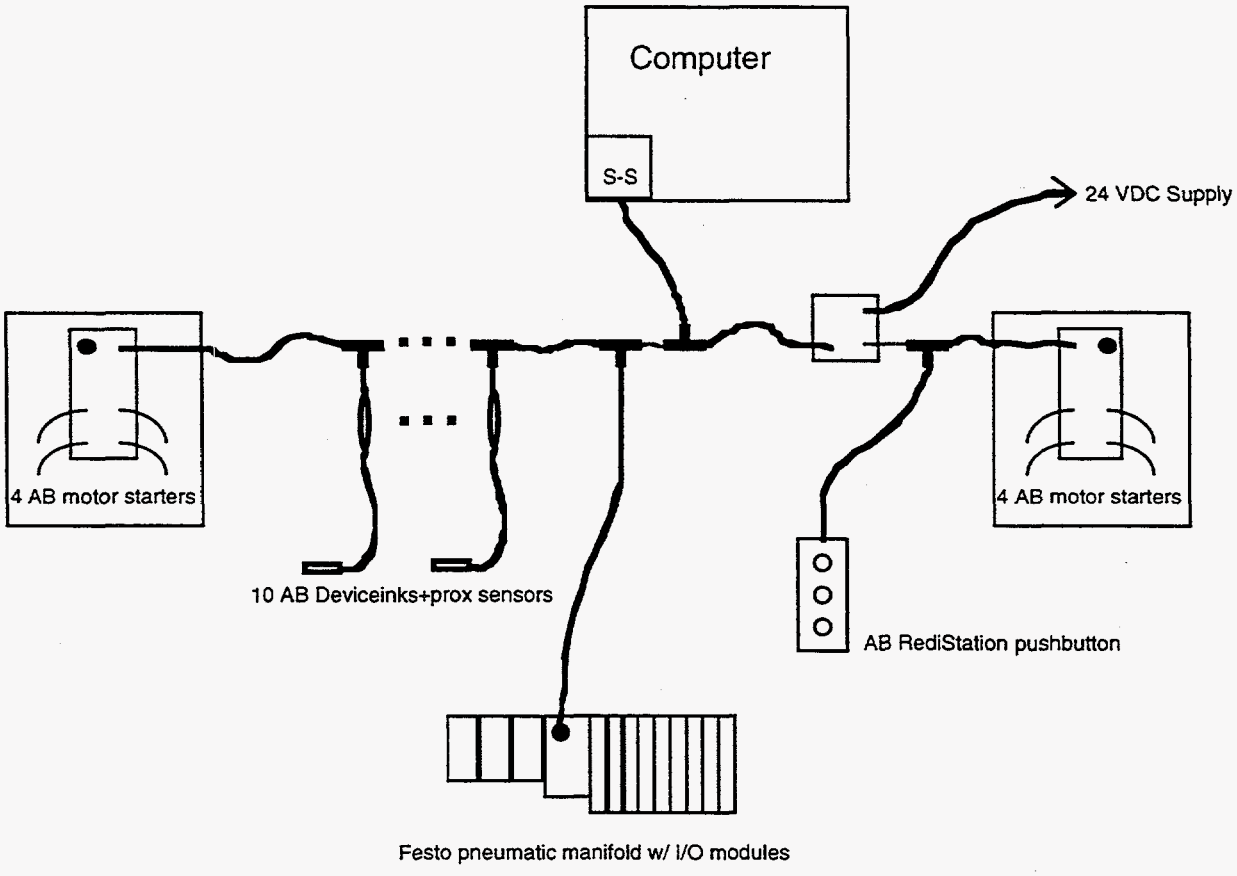


Fig. 15.- Typical DeviceNet Network

VIII.- Conclusions

The great advances in computer technology in the last few years made it possible to have large computational capacity at a relatively low cost. You can do today in a Pentium-Pro based systems what it took many older generation processors and specialized busses such as VMEbus or S-bus. Specialized architectures such as HyperForest will still have niche applications, but for the most part, a larger size of problem can be solved today with the advances in the Personal Computer industry such as Windows NT, Pentium-Pro processors, high speed networks, etc. The main advantages here being the wide availability of software tools, the number of people that know how to use them, and the low cost of the hardware.

The work on this LDRD provides an architecture that can be tailored to a niche application in terms of number of processors (or computing power desired) and interconnects to sensors and other devices. We did not get to apply it to a big problem, but this work that will hopefully continue both at Sandia and at the University of New Mexico.

IX.- References

Agarwal, A., B. H. Lim, et al. (1990). APRIL: A processor architecture for multiprocessing. 17th IEEE Annual International Symposium on Computer Architecture, 104-114.

Ahmad, S. and B. Li (1987). Optimal design of multiple arithmetic processor-based robot controllers. IEEE Robotics and Automation Conference, 660-663.

Allison, D. A. (1990). System issues in embedded control. IEEE COMPCON, 214-215.

Appiani, E., B. Conterno, et al. (1989). "EMMA2, a high-performance hierarchical multiprocessor." IEEE Micro (February): 42-56.

Aras, C. M. and R. C. Luo (1989). HMP: A hierarchical multiprocessor computer architecture for multi-sensor based robotic tasks. IEEE International Symposium on Intelligent Control, Albany, NY, USA, 487-492.

Bhuyan, L. M. and D. P. Agrawal (1984). "Generalized hypercube and hyperbus structures for a computer network." IEEE Transactions on Computers C-33(4): 323-333.

Bhuyan, L. N., Q. Yang, et al. (1989). "Performance of multiprocessor interconnection networks." IEEE Computer (February): 25-37.

Carlson, D. A. (1988). "Modified mesh-connected parallel computers." IEEE Transactions on Computers 37(October): 1315-1321.

Carr, L., R. Kibler, et al. (1989). G-32 A high performance VLSI 3-D computer. 22th Annual Hawaii International Conference on System Sciences, Kailua-Kona, HI, USA, 127-134.

Chang, P. R. and C. S. Lee (1988). Residue arithmetic VLSI array architecture for manipulator pseudo-inverse jacobian computation. IEEE Robotics and Automation Conference, 297-302.

Chow, E., H. Madan, et al. (1987). A real-time adaptive message routing network for the hypercube computer. IEEE Real Time Systems Symposium, 88-96.

Cogswell, B. and Z. Segall (1991). MACS: a predictable architecture for real time systems. IEEE Real Time Systems Symposium, 296-305.

Despain, A. M. and D. A. Patterson (1978). X-tree: A tree structured multi-processor computer architecture. 5th Annual Symposium on Computer Architecture, Palo Alto, CA, USA, 144-151.

Dinning, A. (1989). "A survey of synchronization methods for parallel computers." IEEE Computer (July): 66-77.

Duncan, R. (1990). "A survey of parallel computer architectures." IEEE Computer (February): 5-16.

Fathi, E. T. and M. Krieger (1983). "Multiple microprocessor systems: what, why, and when." IEEE Computer (March): 23-32.

Fiddler, J., D. N. Wilner, et al. (1990). Multiprocessing: An extension of distributed, real-time computing. IEEE COMPCON, 216-218.

Fornard, R. J. and E. W. Davies (1989). Analysis and implementation of hierarchical real-time architectures. Euromicro Workshop on Real Time, Como, Italy, 66-73.

Fotakis, D. and N. Bourbakis (1987). A RISC-type structural design of the HERMES multiprocessor kernel. Supercomputing, 1st International Conference, Athens, Greece, 1011-1030.

Franzkowiak, G. H. and H. Schmid (1983). Efficiency analysis of single bus multiprocessor architectures. IEEE Real Time Systems Symposium, 51-60.

Furth, B., D. Gluch, et al. (1991). The REAL/STAR 2000: A high performance multiprocessor computer for telemetry applications. ITC/USA/'91 International Telemetering Conference, Las Vegas, NV, USA, 365-373.

Gerber, R. and I. Lee (1989). Communicating shared resources: a model for distributed real-time systems. IEEE Real Time Systems Symposium, 68-78.

Goodman, J. R. and C. H. Sequin (1981). "HyperTree: A multiprocessor interconnection topology." IEEE Transactions on Computers C-30(December): 923-933.

Goodman, J. R. and P. J. Woest (1988). The Wisconsin Multicube: A new large-scale cache-coherent multiprocessor. 15th IEEE Annual International Symposium on Computer Architecture, 422-429.

Graham, J. H. (1989). "Special computer architectures for robotics: tutorial and survey." IEEE Transactions on Robotics and Automation 5(October): 543-554.

Han, C. C. and K. J. Lin (1989). Scheduling parallelizable jobs on multiprocessors. IEEE Real Time Systems Symposium, 59-67.

Huffstutter, K. L. (1990). A distributed processing system design verification process. IEEE/AIAA/NASA 9th Digital Avionics Systems Conference, Virginia Beach, VA, USA, 458-461.

Hurt, A. D. and J. R. Heath (1982). A hardware task scheduling mechanism for a real-time multi-processor architecture. IEEE Real Time Systems Symposium, 113-123.

Jackson, J. H. (1991). The Data Transport Computer: A 3-dimensional massively parallel SIMD computer. IEEE COMPCON, 264-269.

Joseph, M. and A. Goswani (1988). What's 'real' about real-time systems. IEEE Real Time Systems Symposium, 78-85.

Kazanzides, P. (1988). Multiprocessor control of robotic manipulators. Ph. D. Dissertation, Brown University

Kejian, M. and L. Yongxi (1989). A multiprocessor simulation computer - MPSC-1. Beijing International Conference on System SImulation and Scientific Computing, Beijing, China, 208-210.

Khosla, P. K. and S. Ramos (1988). A comparative analysis of the hardware requirements for the Lagrange-Euler and Newton-Euler dynamics formulations. IEEE Robotics and Automation Conference, 291-296.

Kirk, D. B. (1988). Process dependent static cache partitioning for real-time systems. IEEE Real Time Systems Symposium, 181-190.

Kirk, D. B. (1989). SMART (Strategic Memory Allocation for Real-Time) cache design. IEEE Real Time Systems Symposium, 229-237.

Kirk, D. B. and J. K. Strosnider (1990). SMART (Strategic Memory Allocation for Real-Time) cache design using the MIPS R3000. IEEE Real Time Systems Symposium, 322-330.

Kovaleski, A., S. Ratheal, et al. (1986). An architecture and an interconnection scheme for time-sliced buses in real-time processing. IEEE Real Time Systems Symposium, 20-27.

Kung, A. and R. Kung (1985). Galaxy: A distributed real-time operating system supporting high availability. IEEE Real Time Systems Symposium, 79-87.

Lenoski, D., J. Laudon, et al. (1992). The DASH prototype: implementation and performance. IEEE 19th Annual International Symposium on Computer Architecture, 92-103.

Leung, S. S. and M. A. Shanblatt (1988). Computer architecture design for robotics. IEEE Robotics and Automation Conference, 453-456.

Li, X. and M. Malek (1988). Analysis of speed up and communication/computation ratio in multiprocessor systems. IEEE Real Time Systems Symposium, 282-288.

Light, R. A. (1982). A real-time executive for multiple microprocessor systems. IEEE Real Time Systems Symposium, 143-150.

Lin, H. and W. Yueming (1985). VLSI systolic architecture for real-time digital signal processing. IEEE Real Time Systems Symposium, 141-146.

Ling, Y.-L. C. (1986). Hierarchical multiprocessor architecture design in VLSI for real-time robotic control applications. Ph. D. Dissertation, Ohio State University

Malbasa, V. (1987). RAMS- a multiprocessor system for dynamic system simulation. IEEE Real Time Systems Symposium, 64-69.

Markenscoff, P. (1982). Analysis of a multiple processor system for real time applications. IEEE Real Time Systems Symposium, 45-56.

Männer, R. and O. Stucky (1990). "The Polyp multiprocessor: Architecture and applications in nuclear physics." Computers in Physics 4(May/June): 267-274.

Molini, J. J., S. K. Maimon, et al. (1990). Real-time system scenarios. IEEE Real Time Systems Symposium, 214-225.

Mun, W. (1989). Pipelined multiprocessor computer architecture and fast parallel algorithms for real-time robot control. Ph. D. Dissertation, Oregon State University

Naghdy, F., C. K. Wai, et al. (1988). Multiprocessing control of robotic systems. IEEE Robotics and Automation Conference, 975-977.

Niehaus, D. (1991). Program representation and translation for predictable real-time systems. IEEE Real Time Systems Symposium, 53-63.

Olson, R. (1986). Realtime response on a message based multiprocessor. IEEE Real Time Systems Symposium, 28-35.

Rajkumar, R., L. Sha, et al. (1988). Real-time synchronization protocols for multiprocessors. IEEE Real Time Systems Symposium, 259-269.

Rasmussen, R. D., N. J. Dimopoulos, et al. (1987). MAX: advanced general purpose real time multicomputer for space applications. IEEE Real Time Systems Symposium, 70-78.

Rayfield, J. T. (1988). Armstrong, a loosely-coupled multiprocessor testbed for reconfigurable topologies. Ph. D. Dissertation, Brown University

Rettberg, R. D., W. R. Crowther, et al. (1990). "The Monarch parallel processor hardware design." IEEE Computer (April): 18-30.

Rowe, P. K. and B. Pagurek (1987). Remedy: A real-time multiprocessor, system level debugger. IEEE Real Time Systems Symposium, 230-240.

Saponas, T. G. (1986). A real-time distributed processing system. IEEE Real Time Systems Symposium, 36-43.

Schwan, K., T. Bihari, et al. (1985). GEM: Operating system primitives for robots and real-time control systems. IEEE Robotics and Automation, 807-813.

Schwan, K., W. Bo, et al. (1986). A high-performance, object-based operating system for real-time, robotic applications. IEEE Real Time Systems Symposium, 147-156.

Sha, L., R. Rajkumar, et al. (1990). Real-time scheduling support in Futurebus. IEEE Real Time Systems Symposium, 331-340.

Shin, G. K. (1991). "HARTS: A distributed real-time architecture." IEEE Computer (May): 25-35.

Stenström, P. (1990). "A survey of cache coherence schemes for multiprocessors." IEEE Computer (June): 12-24.

Tamir, Y. and G. L. Frazier (1988). Support for high priority traffic in VLSI communication switches. IEEE Real Time Systems Symposium, 191-200.

Tan, W. S., S. H. Russ, et al. (1991). GT-EP: A novel high-performance real-time architecture. IEEE 18th Annual International Symposium on Computer Architecture, 12-21.

Thakkar, S., M. Dubois, et al. (1990). "Scalable shared-memory multiprocessor architectures." IEEE Computer (June): 71-73.

Toda, K., K. Nishida, et al. (1990). CODA: A multiprocessor architecture for sensor fusion. 5th IEEE International Symposium on Intelligent Control, Philadelphia, PA, USA, 261-266.

van Dijk, G. J. W. and A. J. van der Wal (1991). "EMPS: The design of an architecture for a distributed homogeneous multiprocessor system." Microprocessors and Microsystems 15(May): 187-194.

Vranesic, Z. G., M. Stumm, et al. (1991). "Hertor: A hierarchically structured shared-memory multiprocessor." IEEE Computer (January): 72-78.

Walter, C. J., R. M. Kieckhafer, et al. (1985). MAFT: A multicomputer architecture for fault-tolerance in real-time control systems. IEEE Real Time Systems Symposium, 133-140.

Wang, Y. (1988). RIPS: a computer architecture for advanced robot control. Ph. D. Dissertation, University of California, Santa Barbara

Wang, Y., A. Mangaser, et al. (1992). "The 3DP: A processor architecture for three-dimensional applications." IEEE Computer (January): 25-36.

Wedde, H. F., G. S. Alijani, et al. (1988). MELODY: a distributed real-time testbed for adaptive systems. IEEE Real Time Systems Symposium, 112-119.

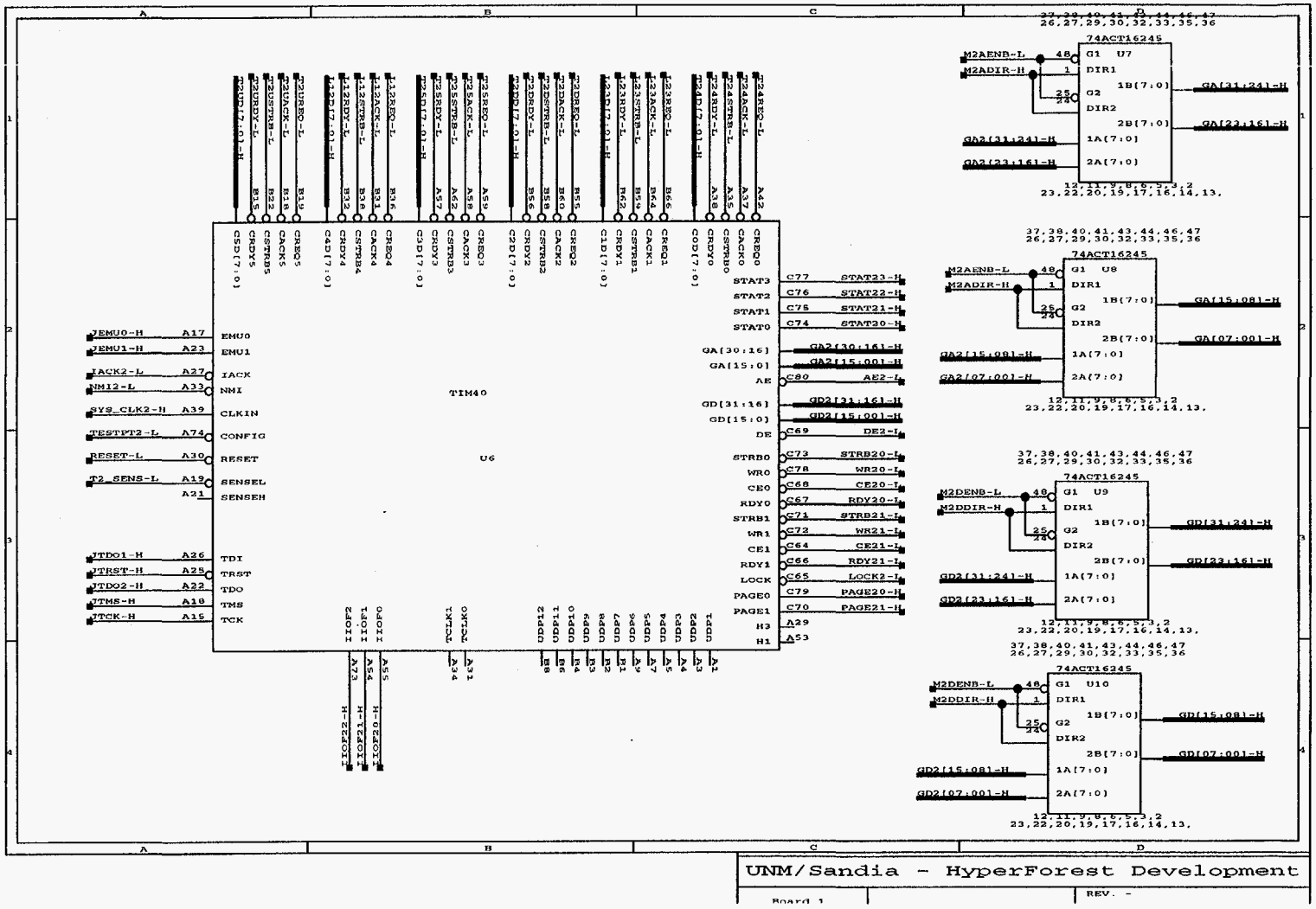
Wilkinson, N. A., M. S. Atkins, et al. (1988). A real time parallel processing data acquisition system. IEEE Real Time Systems Symposium, 54-59.

Woodbury, M. H. (1986). Performance modeling and measurement or real-time multiprocessors with time-shared buses. IEEE Real Time Systems Symposium, 109-118.

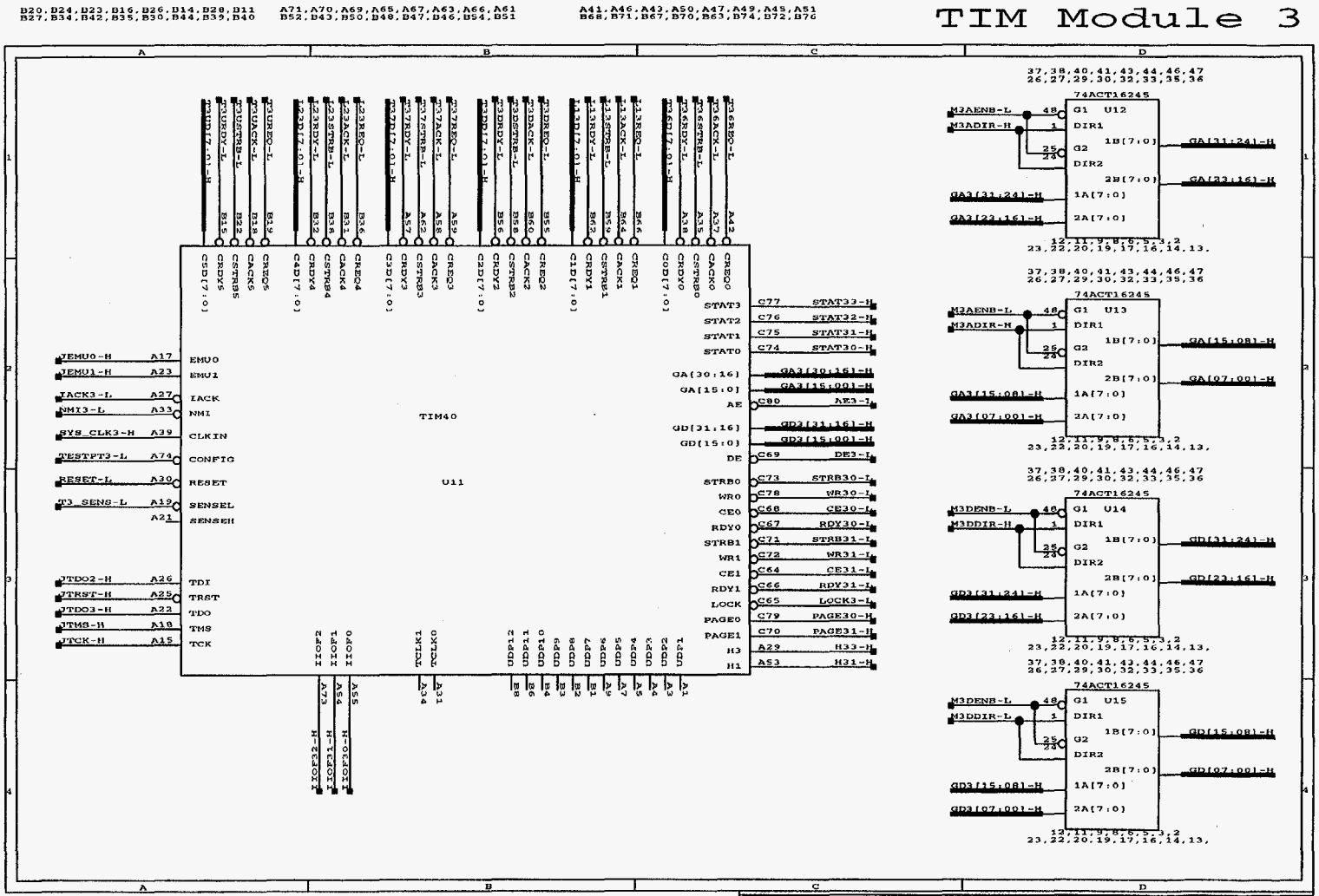
Appendix A

Schematics for Board 1

TIM Module 2

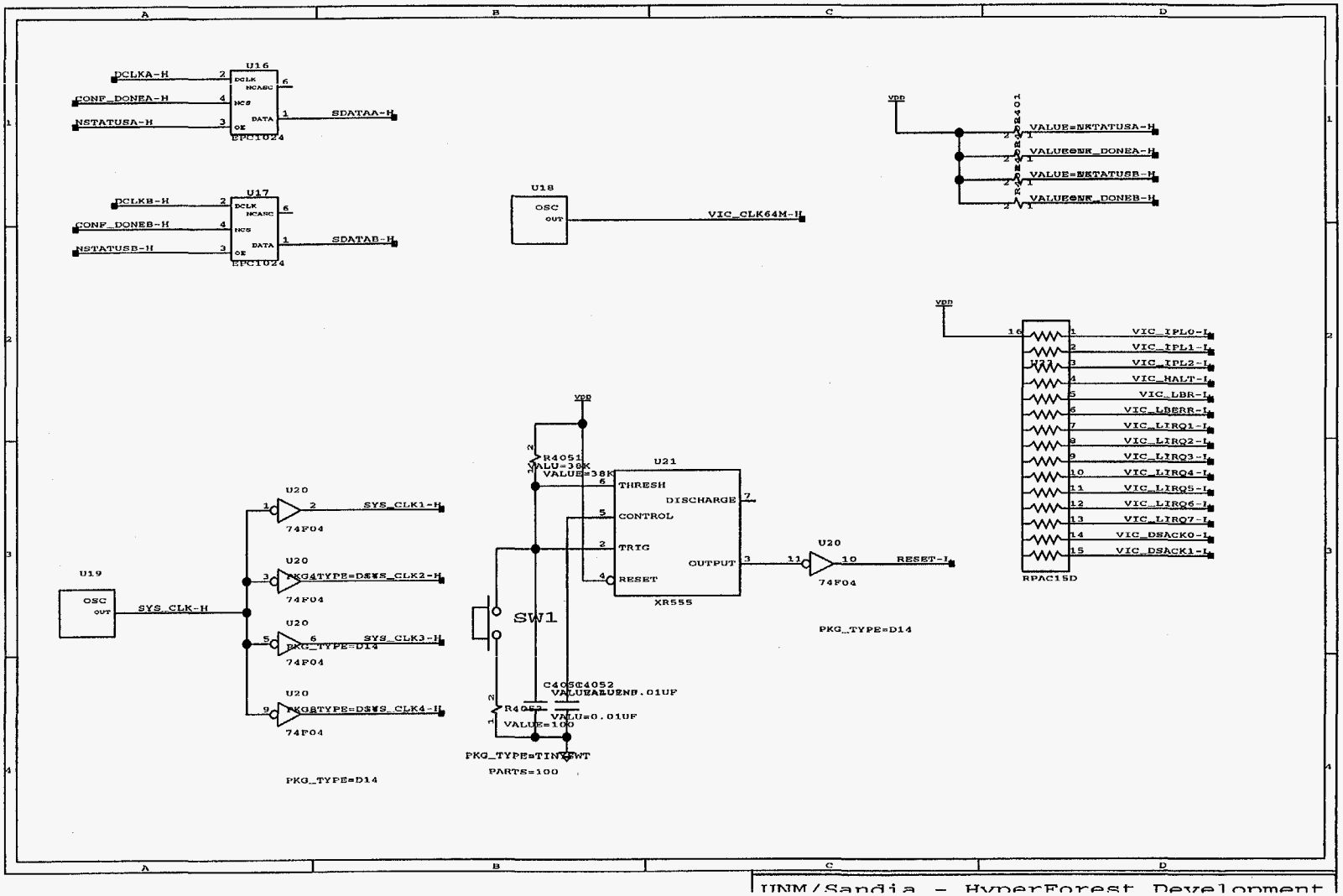

TIM40
U6
74ACT16245
U7
U8
U9
U10
UNM/Sandia - HyperForest Development
Board 1
REV. -

TIM Module 3

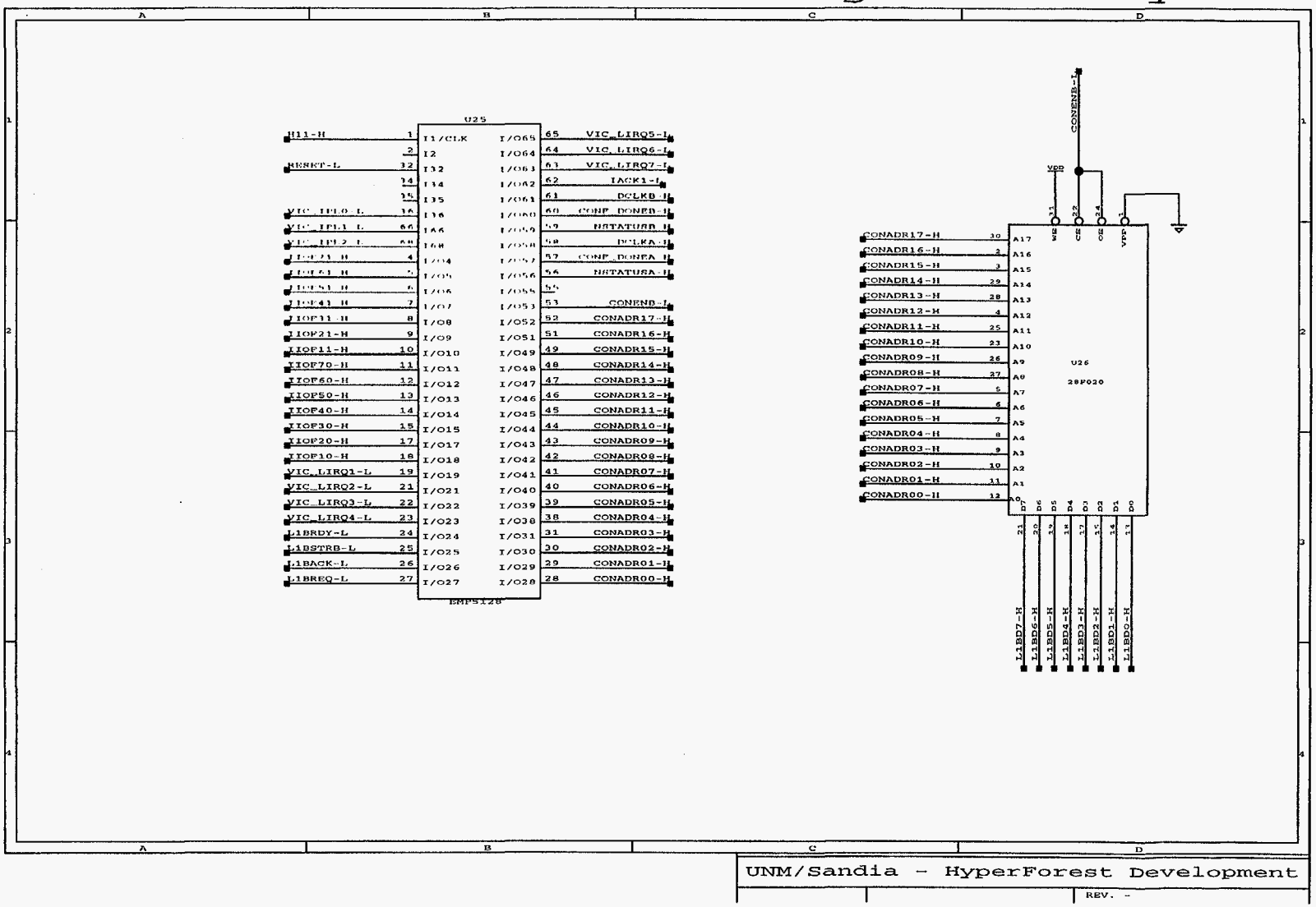


UNM/Sandia - HyperForest Development

REV. -


U16
DCLKA-H
CONF_DONEA-H
NSTATUSA-H
SDATAA-H
EPC1024
U17
DCLKB-H
CONF_DONEB-H
NSTATUSB-H
SDATAB-H
EPC1024
U18
OSC
OUT
VIC_CLK64M-H
VDD
R4051
VALUE=38K
U21
THRESH
DISCHARGE
CONTROL
TRIG
OUTPUT
RESET
XR555
SW1
C4052
VALUE=0.01UF
R4053
VALUE=100
U19
OSC
OUT
SYS_CLK-H
U20
74F04
SYS_CLK1-H
SYS_CLK2-H
SYS_CLK3-H
SYS_CLK4-H
PKG_TYPE=D14
PKG_TYPE=TINYSWT
PARTS=100
RESET-L
R4401
VALUE=NSTATUSA-H
VALUE=CONF_DONEA-H
VALUE=NSTATUSB-H
VALUE=CONF_DONEB-H
RPAC15D
VIC_IPL0-L
VIC_IPL1-L
VIC_IPL2-L
VIC_HALT-L
VIC_LBR-L
VIC_LBERR-L
VIC_LIRQ1-L
VIC_LIRQ2-L
VIC_LIRQ3-L
VIC_LIRQ4-L
VIC_LIRQ5-L
VIC_LIRQ6-L
VIC_LIRQ7-L
VIC_DSACK0-L
VIC_DSACK1-L

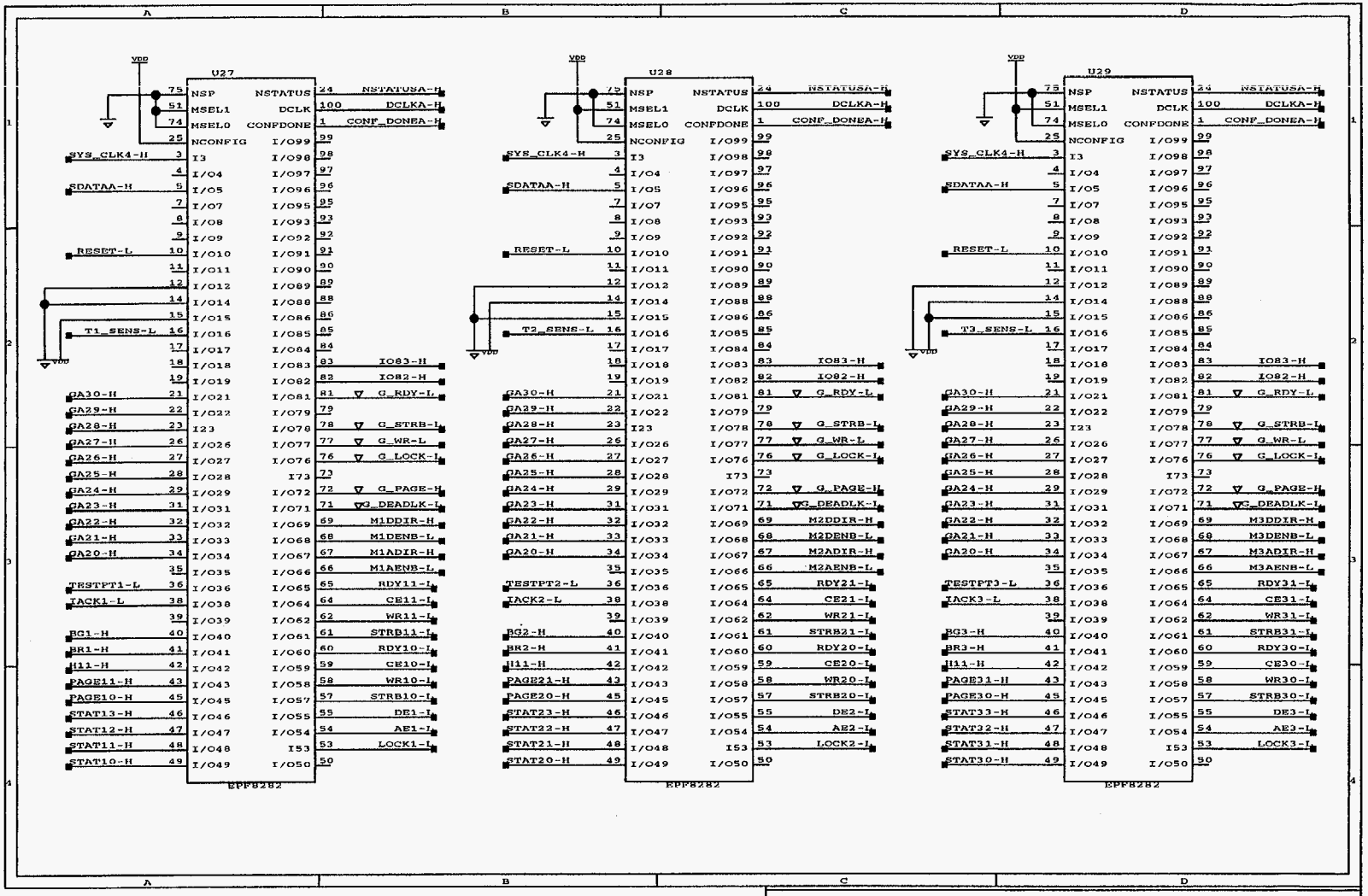
EPROM Boot and Configuration System



UNM/Sandia - HyperForest Development

REV. -

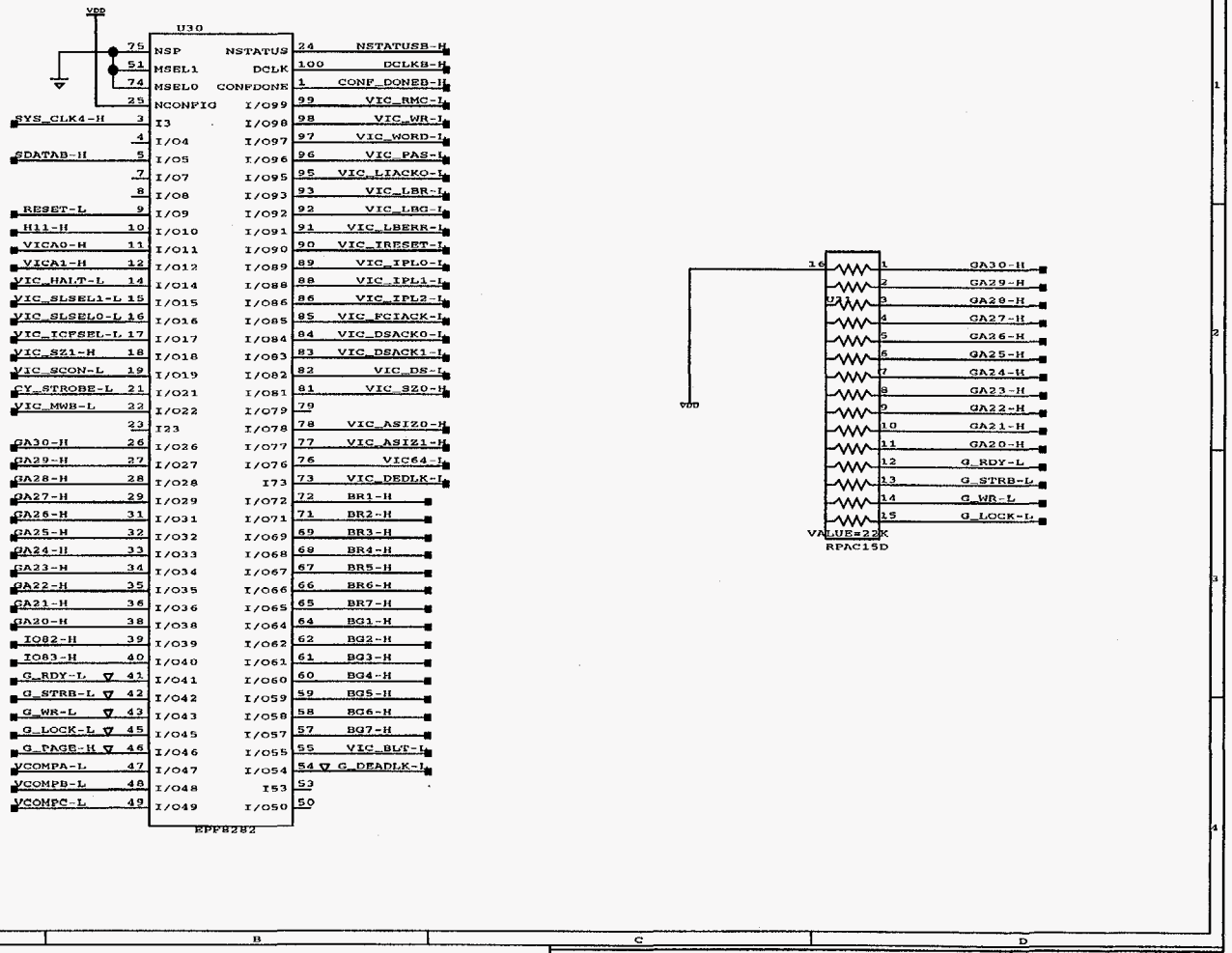
Bus Control



UNM/Sandia - HyperForest Development

REV. -

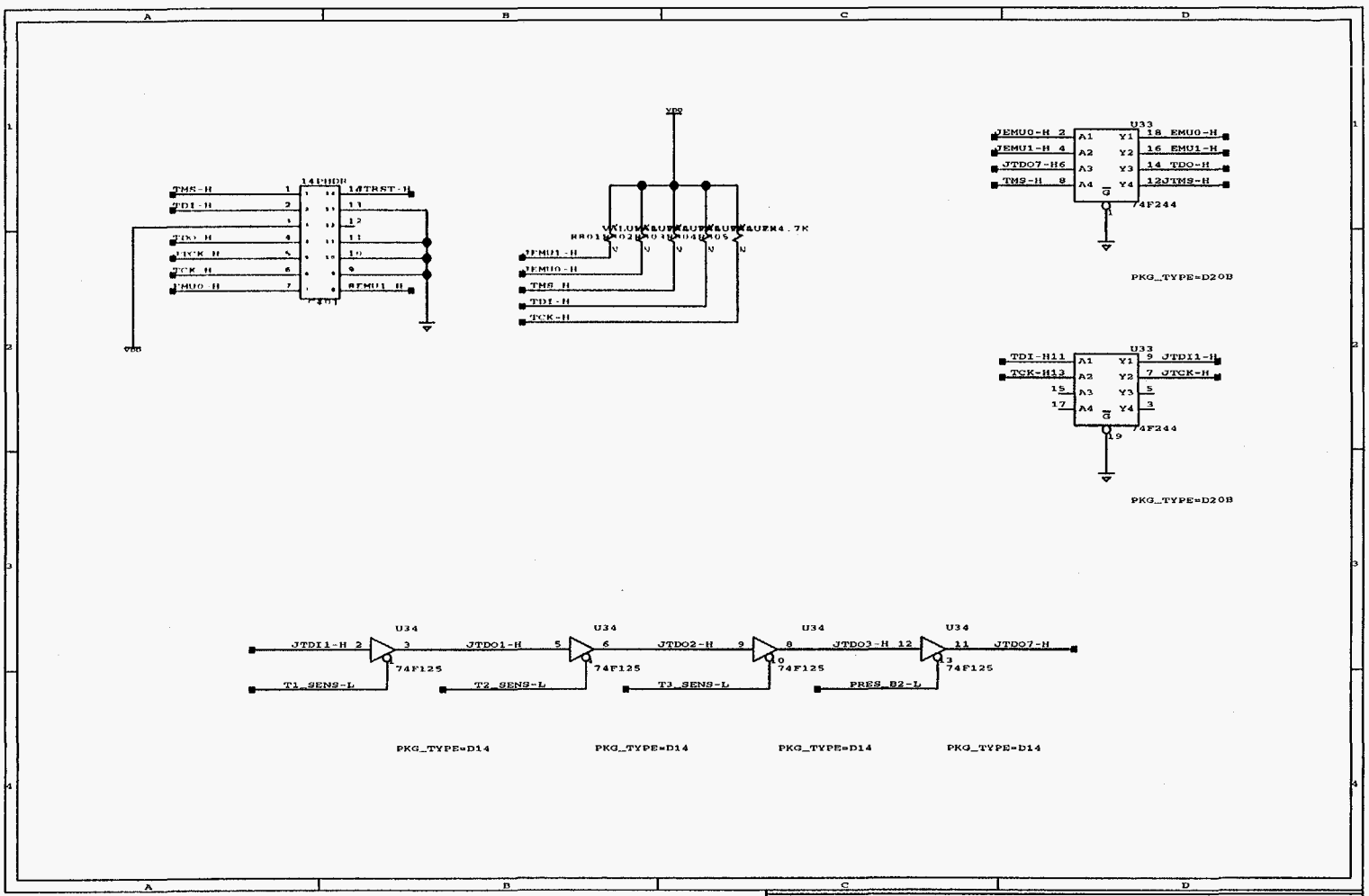
Arbitration System



UNM/Sandia - HyperForest Development

REV. -

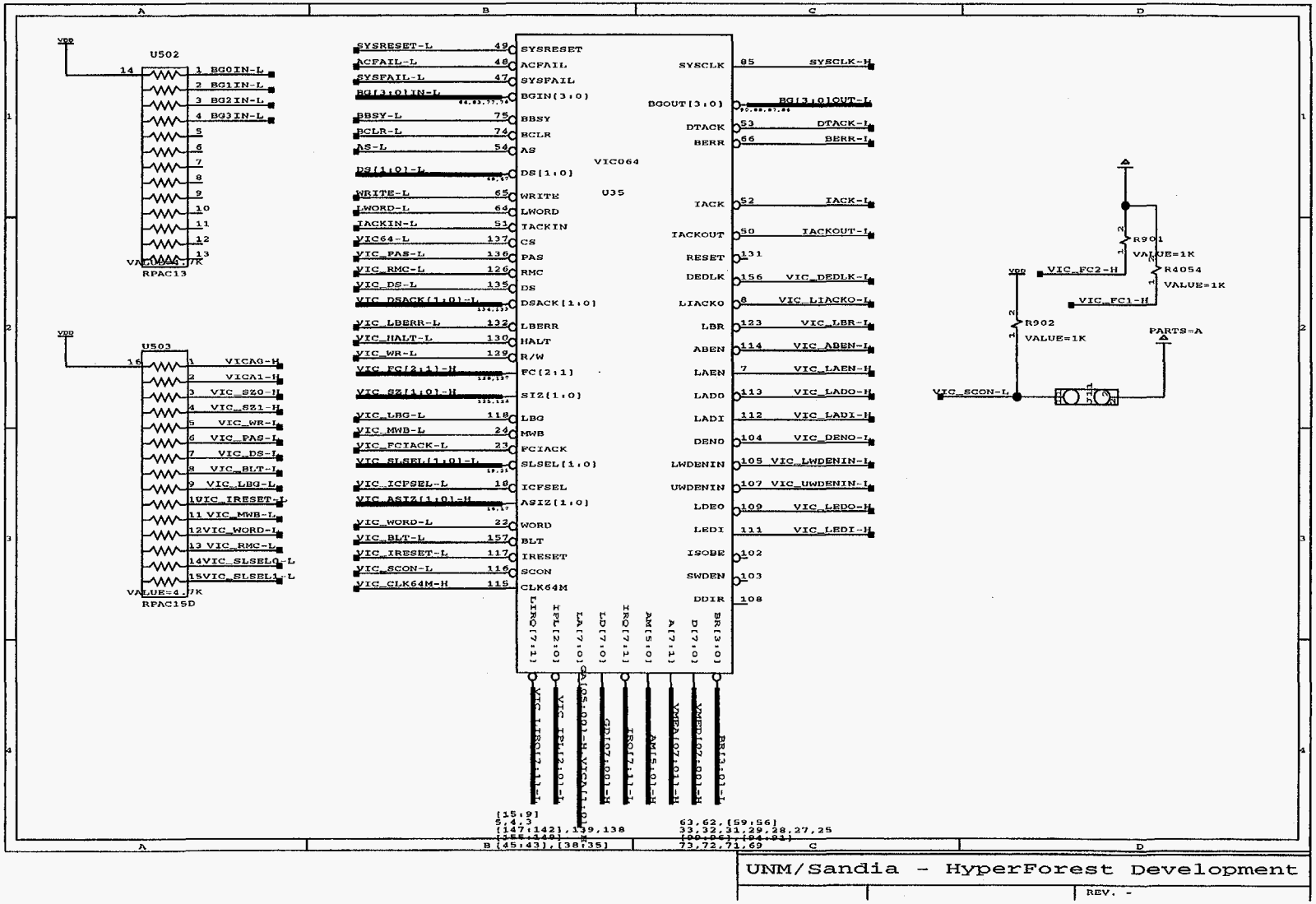
JTAG Hdr and Cktry



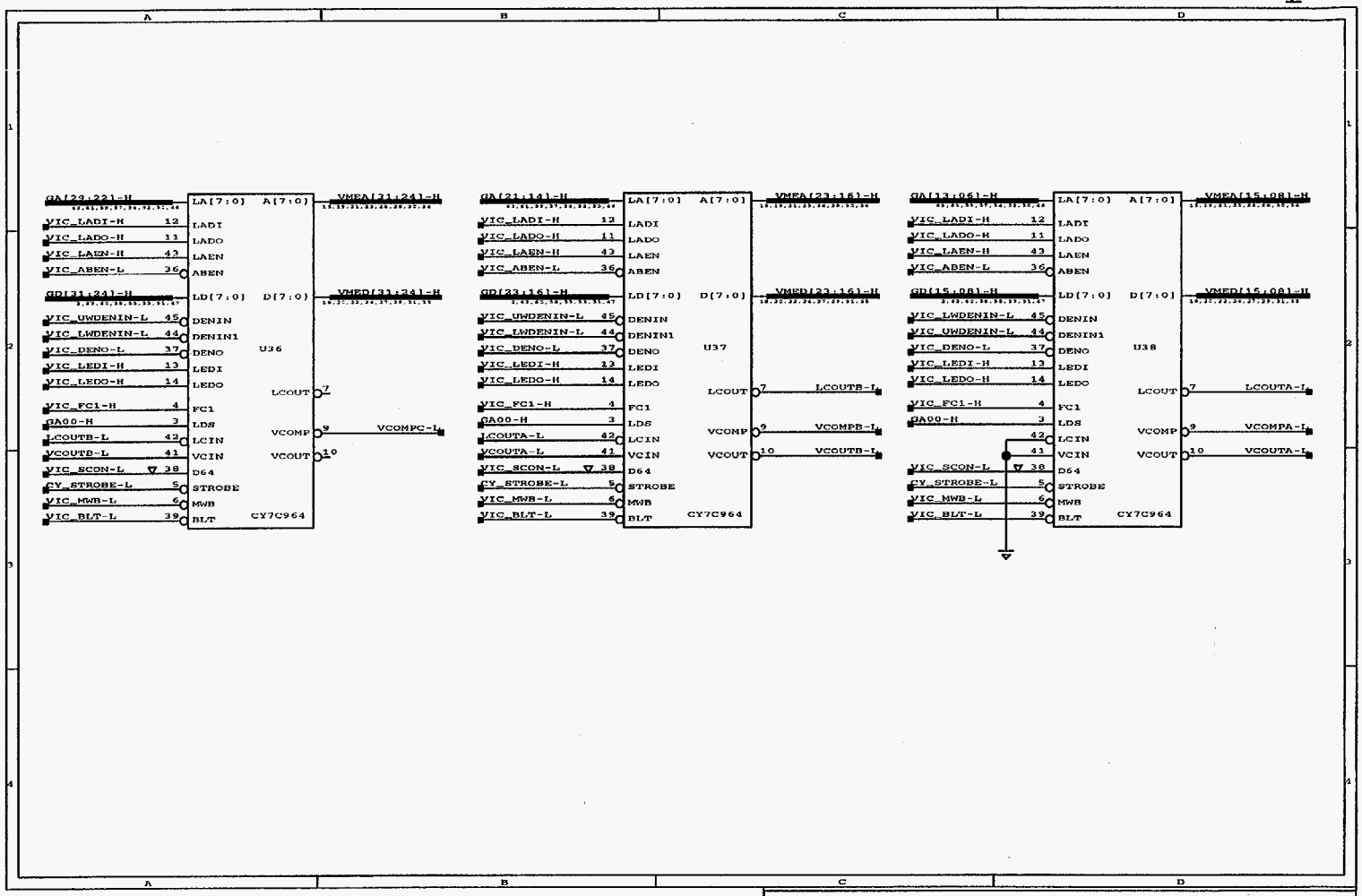
UNM/Sandia - HyperForest Development

REV. -

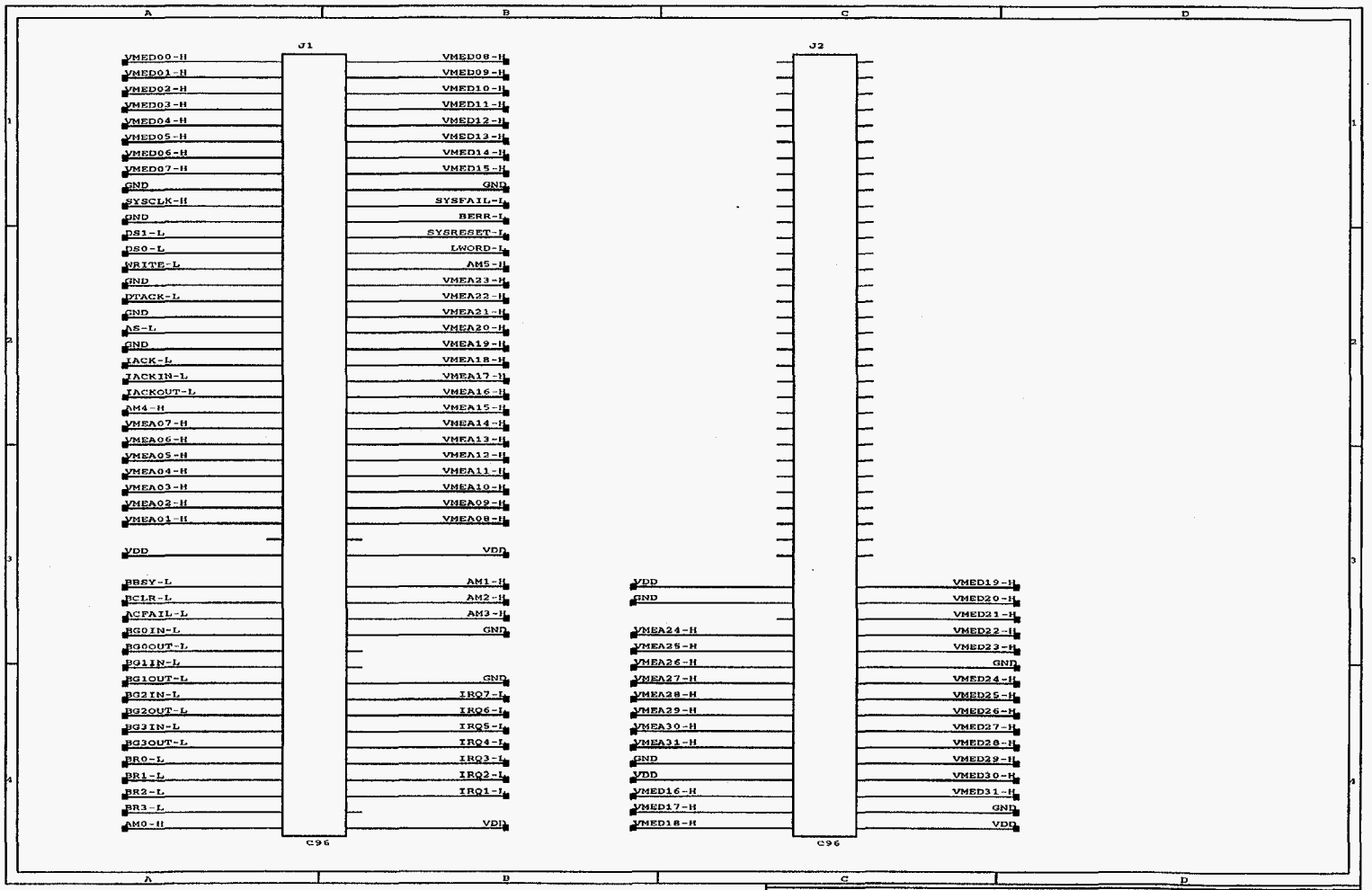
VME Bus Interface Controller



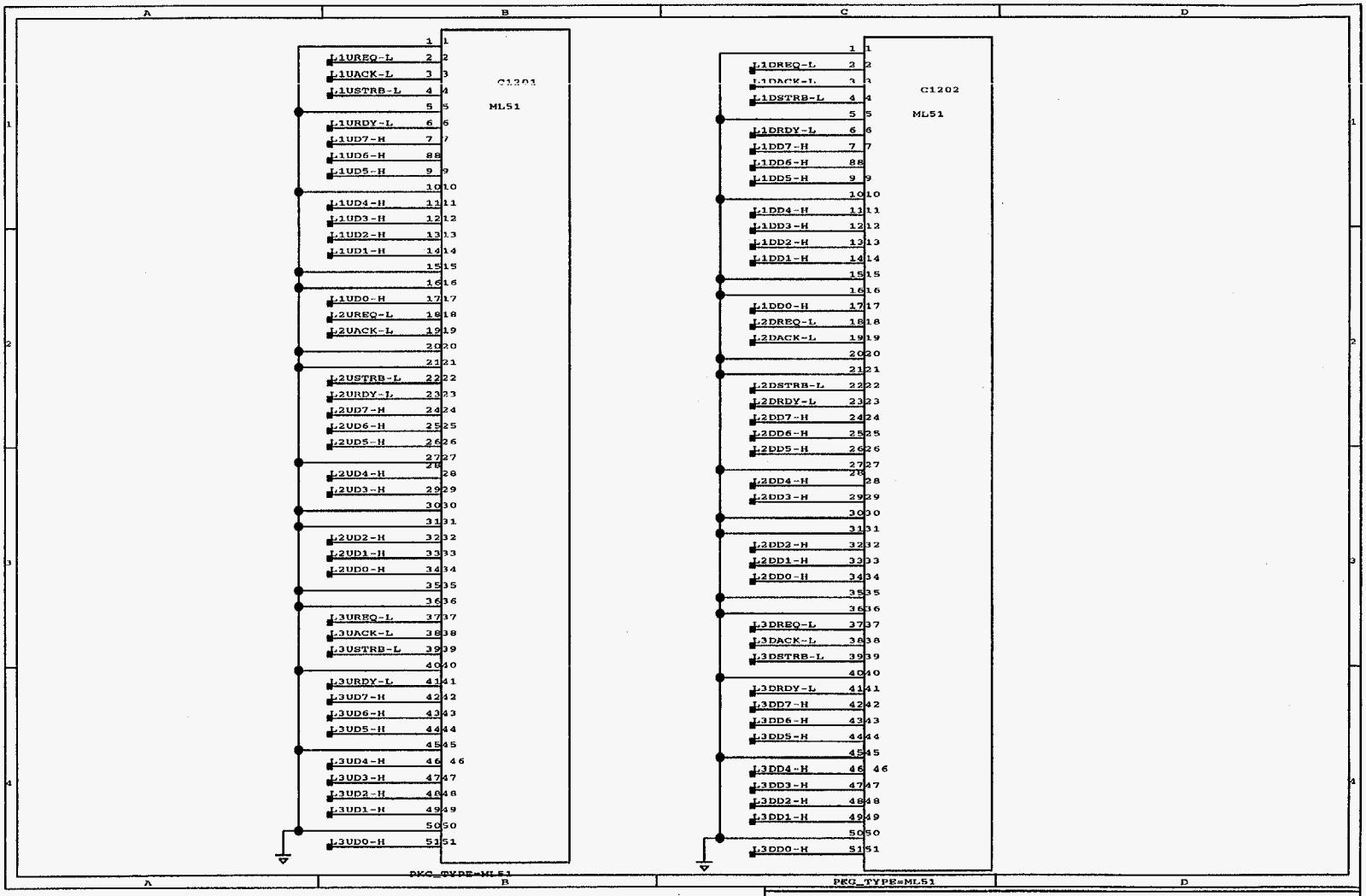
VME Bus Interface Chips



VME Bus Connectors



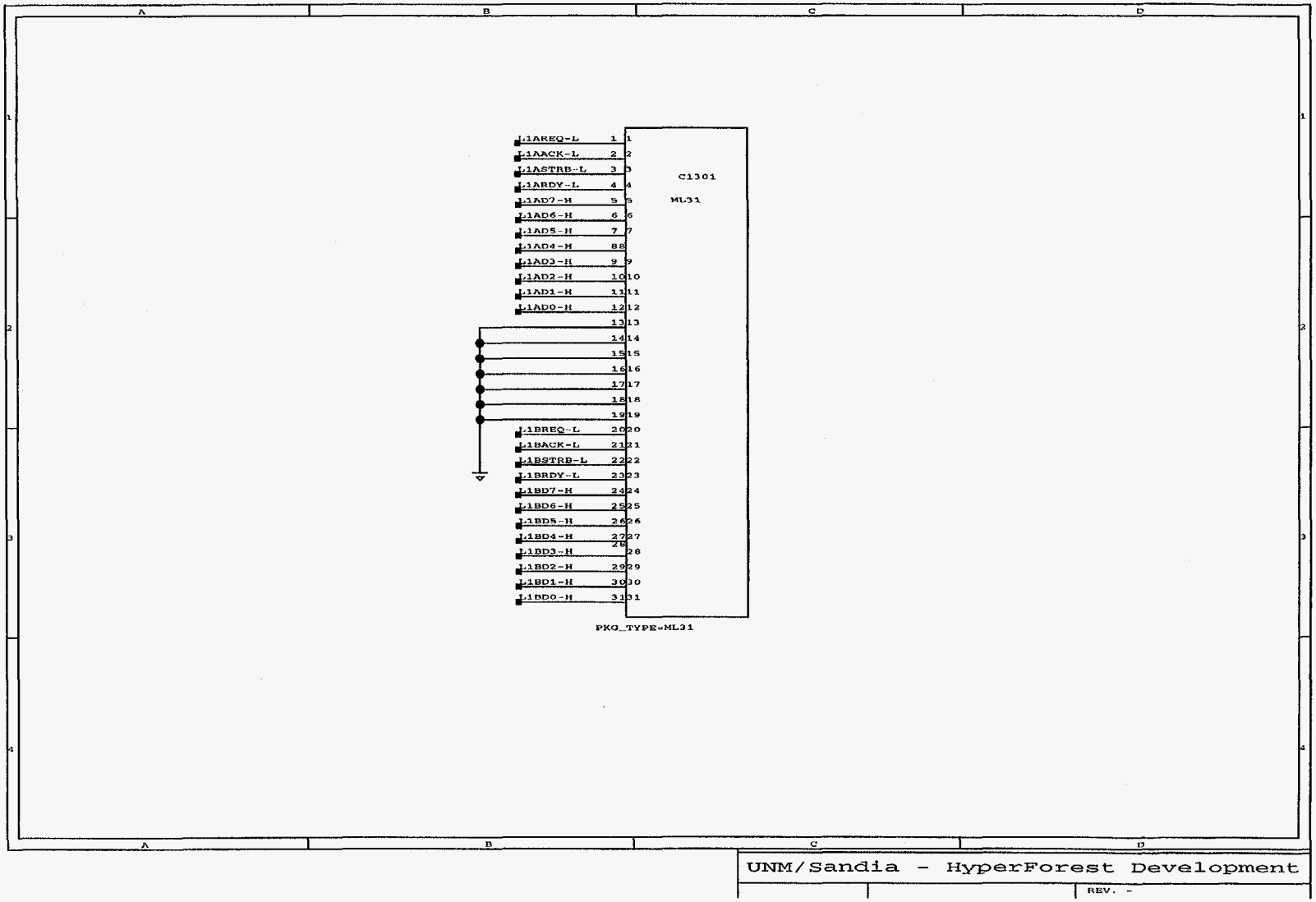
Comm. Port Connectors,Up and Down Links



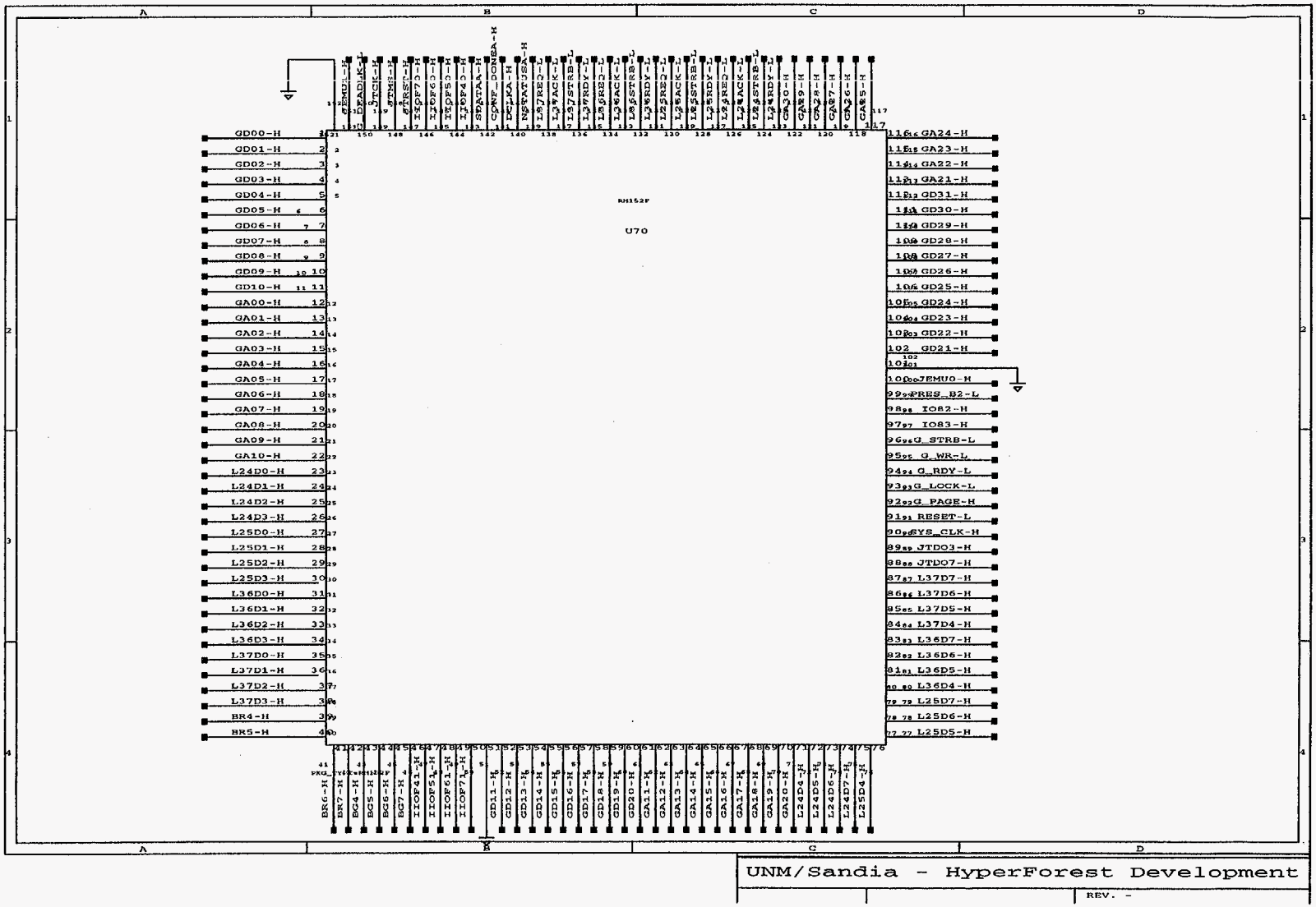
UNM/Sandia - HyperForest Development

REV. -

Communication Port Connectors, Extra Links



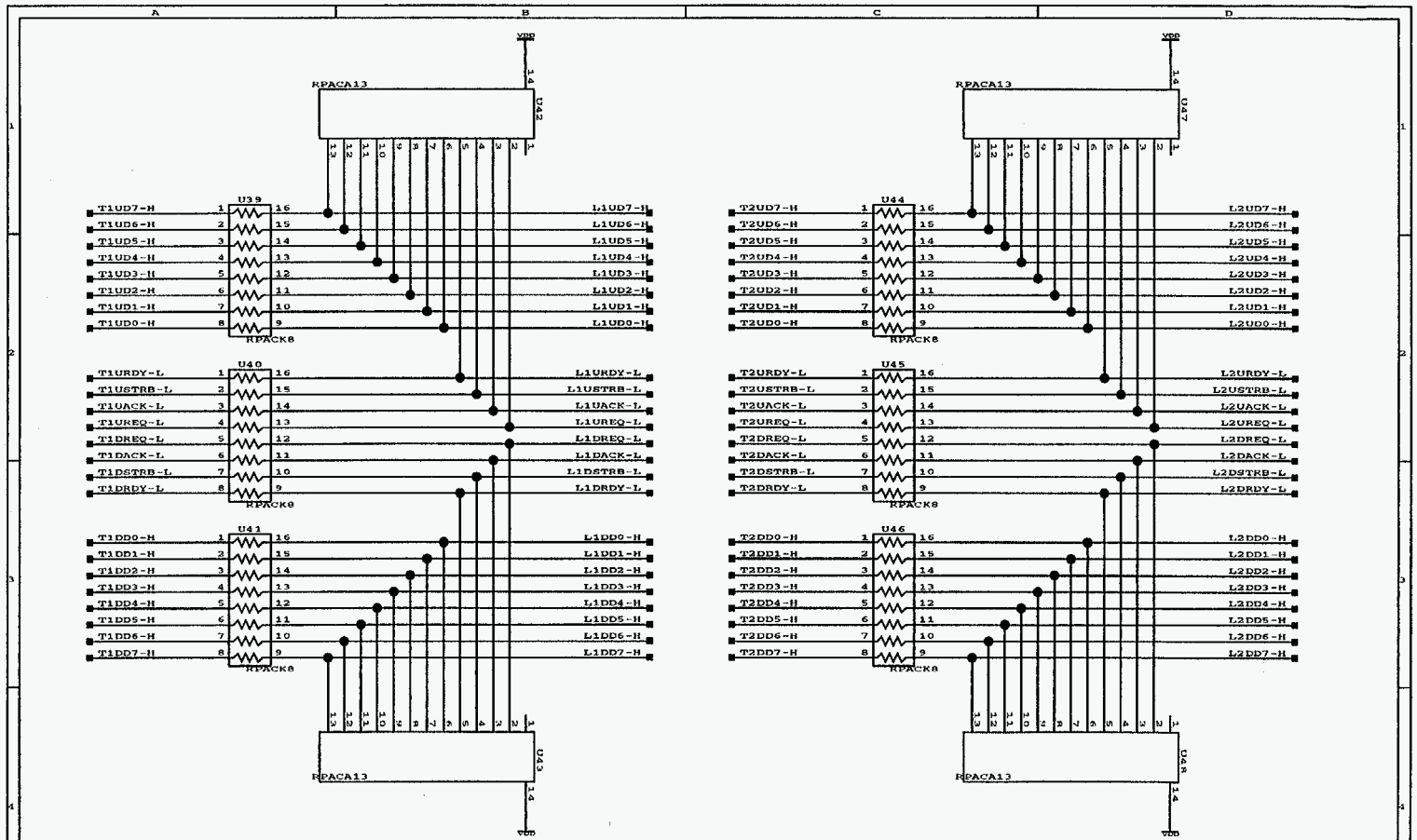
Interboard Connectors



UNM/Sandia - HyperForest Development

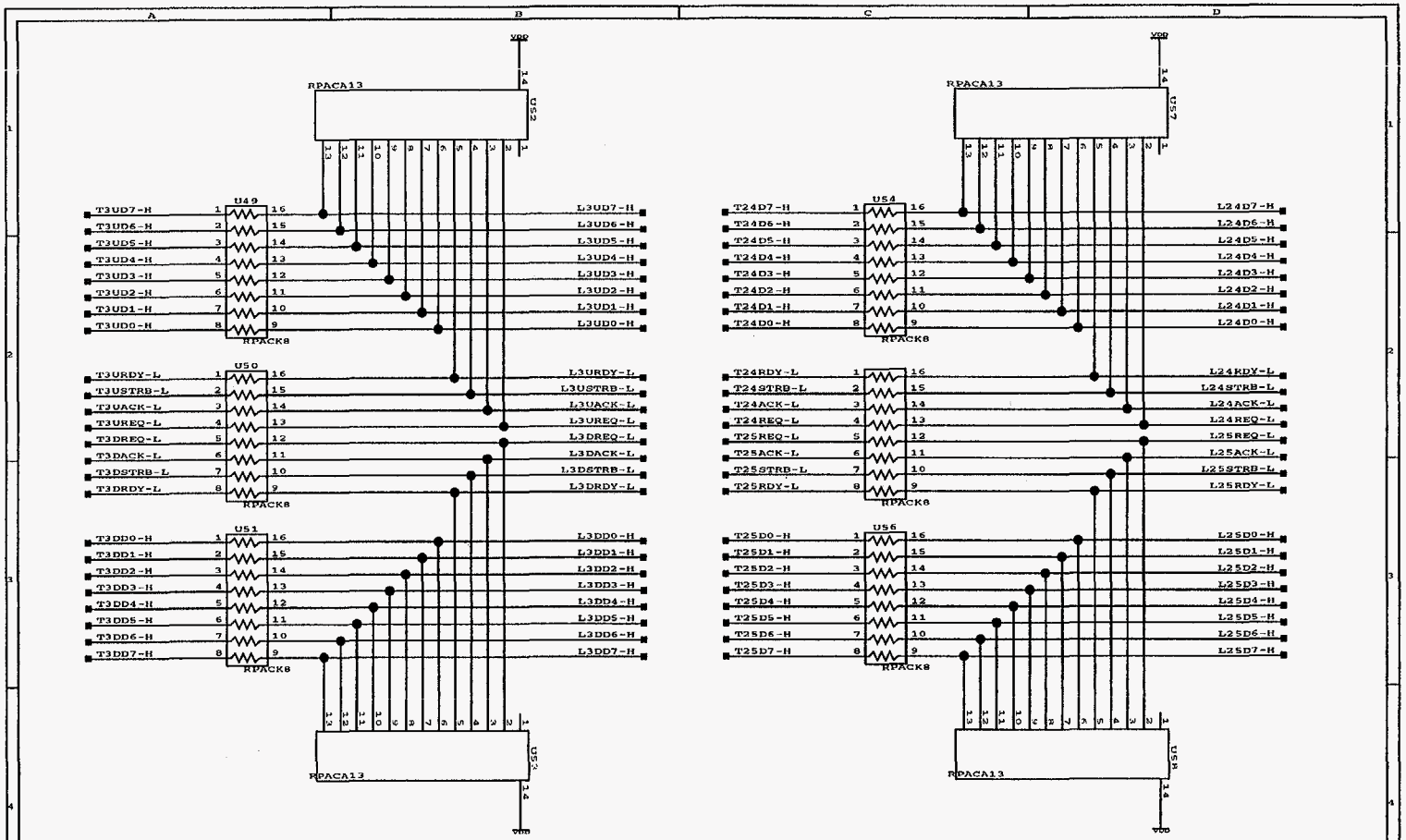
REV. -

Termination Resistors



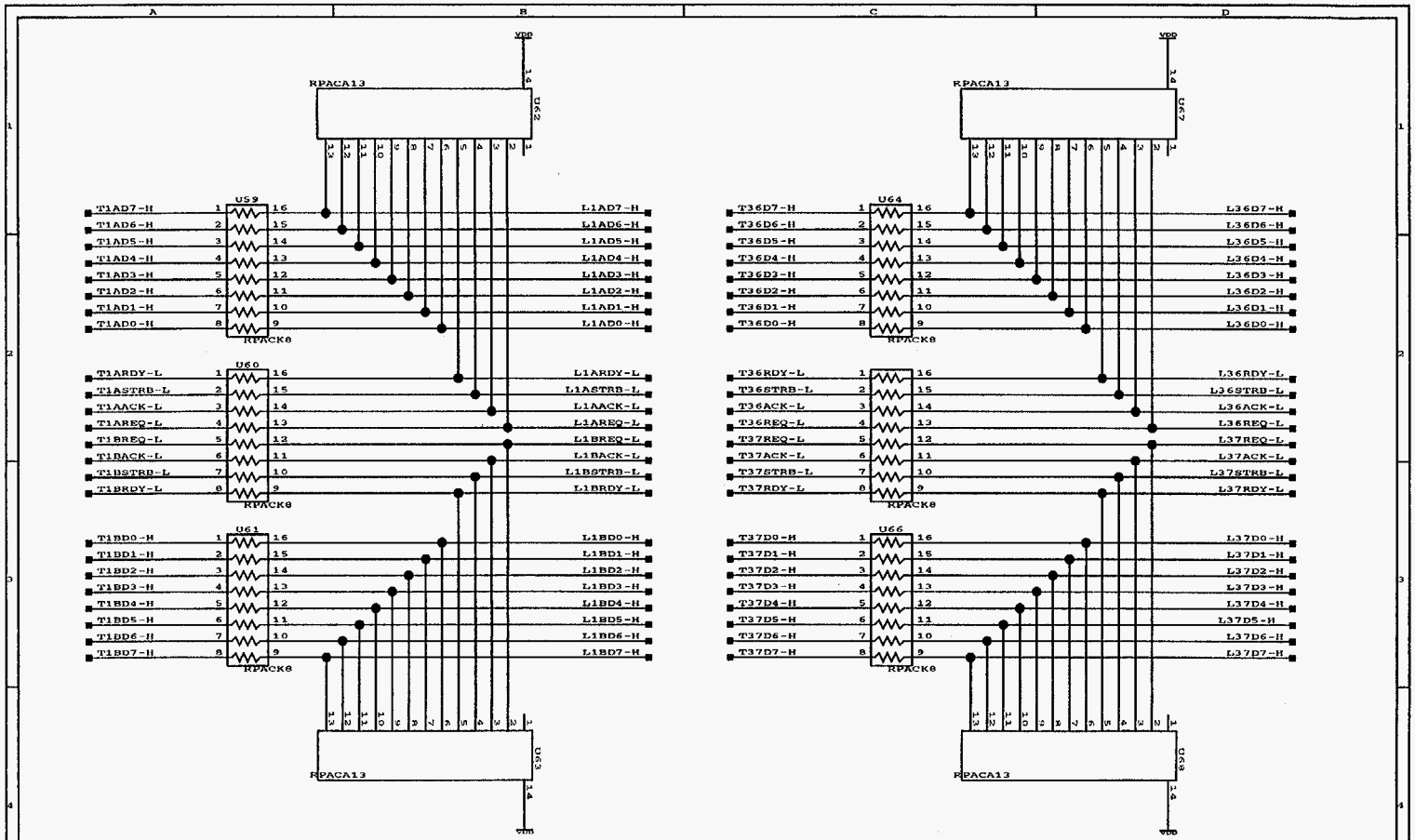
UNM/Sandia - HyperForest Development

Termination Resistors



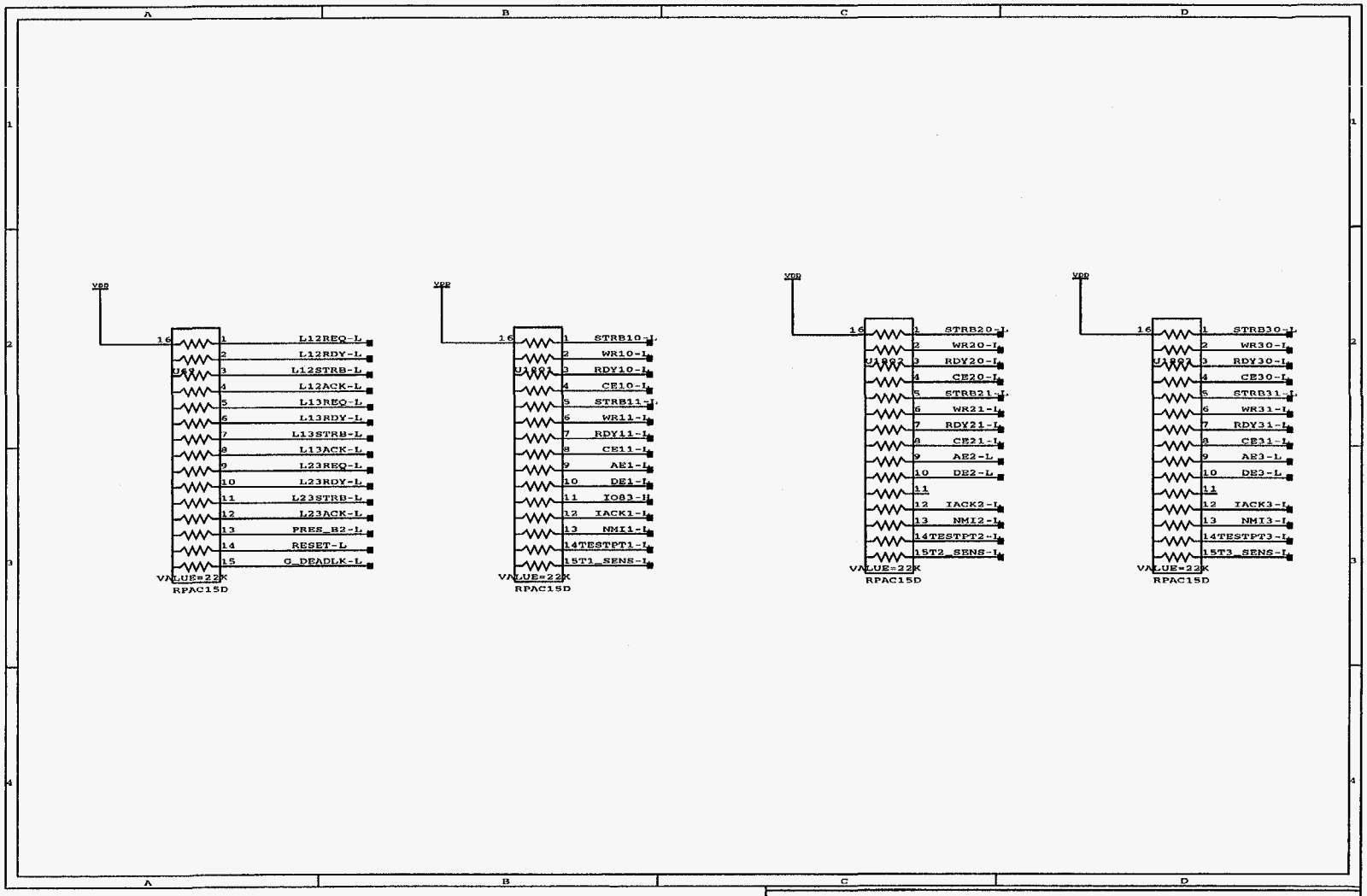
UNM/Sandia - HyperForest Development

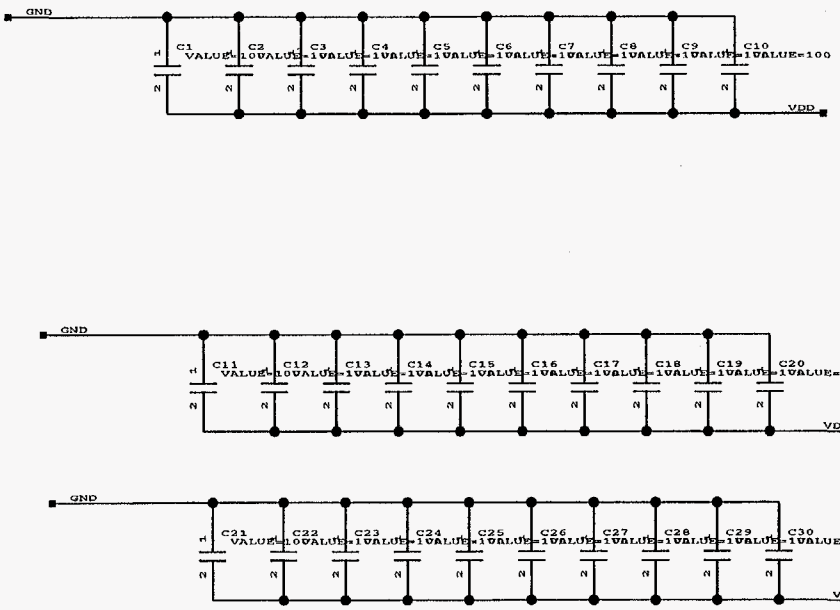
Termination Resistors



UNM/Sandia - HyperForest Development

	PAGE OF	REV. - DRAWN BY:


A
B
C
D
1
2
3
4
VDD
16
U88
1 L12REQ-L
2 L12RDY-L
3 L12STRB-L
4 L12ACK-L
5 L13REQ-L
6 L13RDY-L
7 L13STRB-L
8 L13ACK-L
9 L23REQ-L
10 L23RDY-L
11 L23STRB-L
12 L23ACK-L
13 PRES_B2-L
14 RESET-L
15 G_DEADLK-L
VALUE=22K
RPAC15D
VDD
16
U1883
1 STRB10-L
2 WR10-L
3 RDY10-L
4 CE10-L
5 STRB11-L
6 WR11-L
7 RDY11-L
8 CE11-L
9 AE1-L
10 DE1-L
11 IO83-H
12 IACK1-L
13 NMI1-L
14TESTPT1-L
15T1_SENS-L
VALUE=22K
RPAC15D
VDD
16
U1882
1 STRB20-L
2 WR20-L
3 RDY20-L
4 CE20-L
5 STRB21-L
6 WR21-L
7 RDY21-L
8 CE21-L
9 AE2-L
10 DE2-L
11
12 IACK2-L
13 NMI2-L
14TESTPT2-L
15T2_SENS-L
VALUE=22K
RPAC15D
VDD
16
U1881
1 STRB30-L
2 WR30-L
3 RDY30-L
4 CE30-L
5 STRB31-L
6 WR31-L
7 RDY31-L
8 CE31-L
9 AE3-L
10 DE3-L
11
12 IACK3-L
13 NMI3-L
14TESTPT3-L
15T3_SENS-L
VALUE=22K
RPAC15D
UNM/Sandia - HyperForest Development
REV. -



UNM/Sandia - HyperForest Development

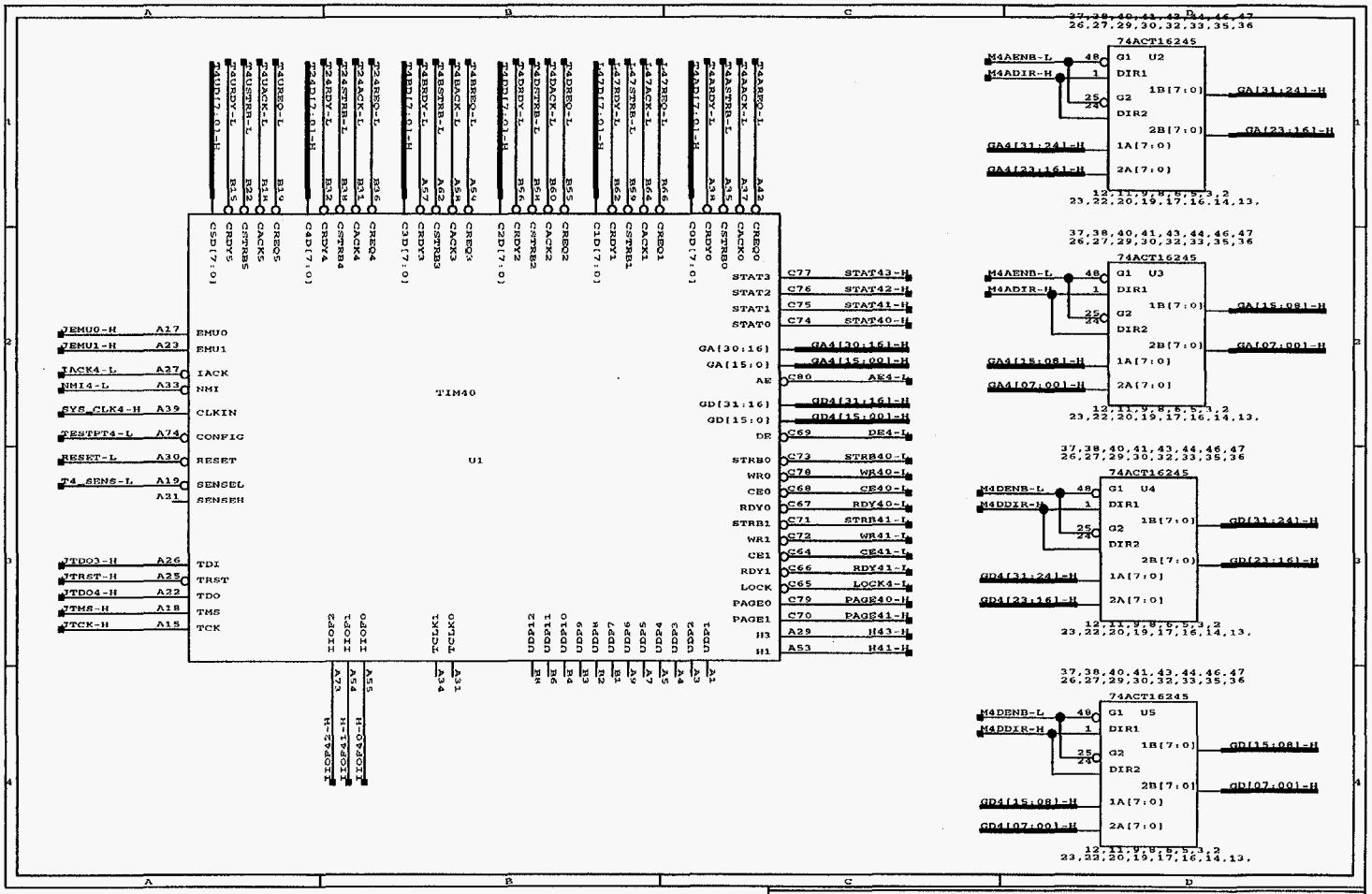
	PAGE OF	REV. - DRAWN BY:

Appendix B

Schematics for Board 2

B20,B24,B23,B16,B26,B14,B28,B11 B27,B34,B42,B35,B30,B44,B39,B40 A71,A70,A69,A65,A67,A62,A66,A61 B52,B43,B50,B48,B47,B46,B54,B51 A41,A46,A43,A50,A47,A49,A45,A51 B68,B71,B67,B70,B63,B74,B72,B75

TIM Module 4

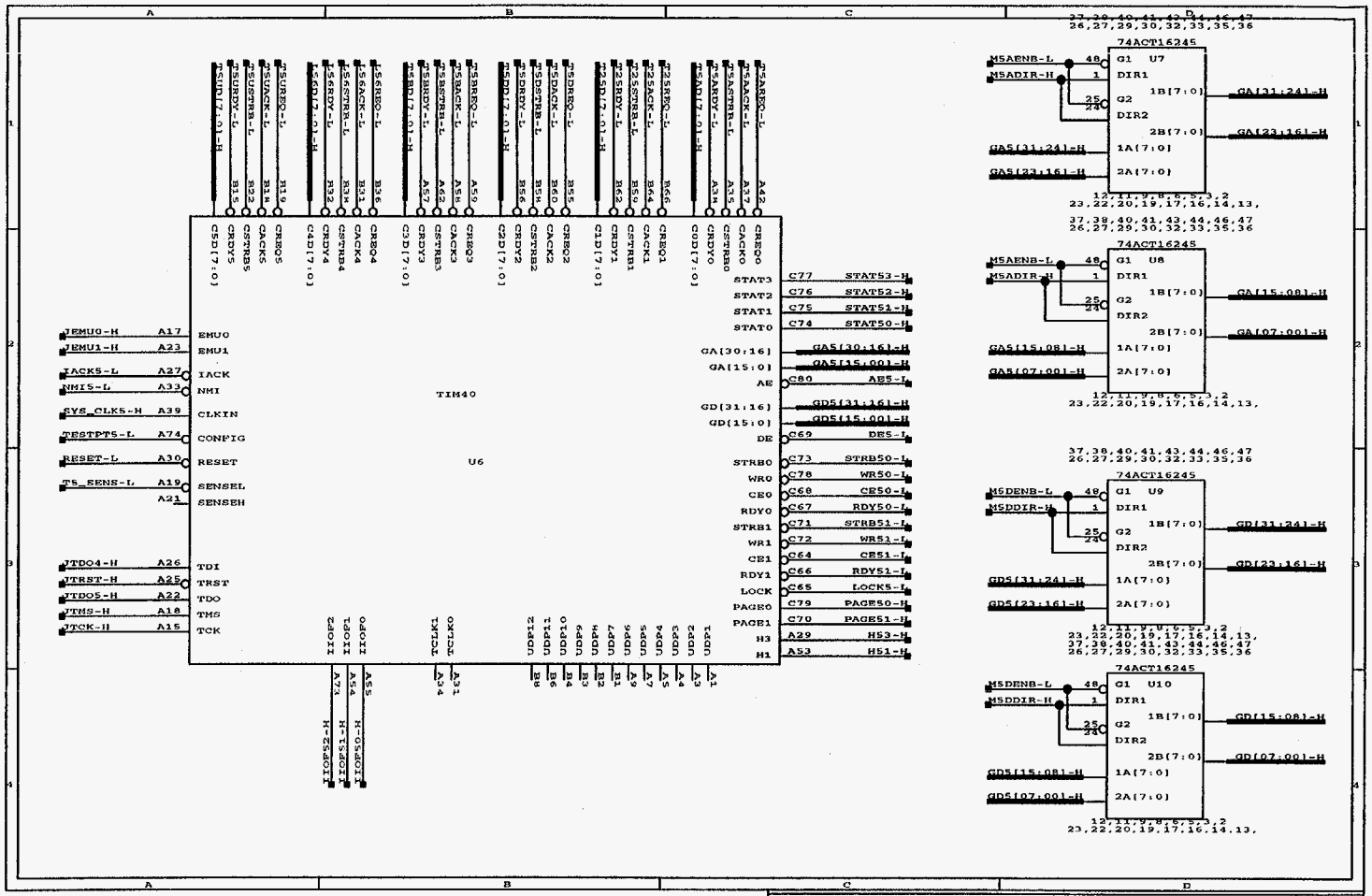


UNM/Sandia - HyperForest Development

	PAGE 1 OF	REV. -
		DRAWN BY:

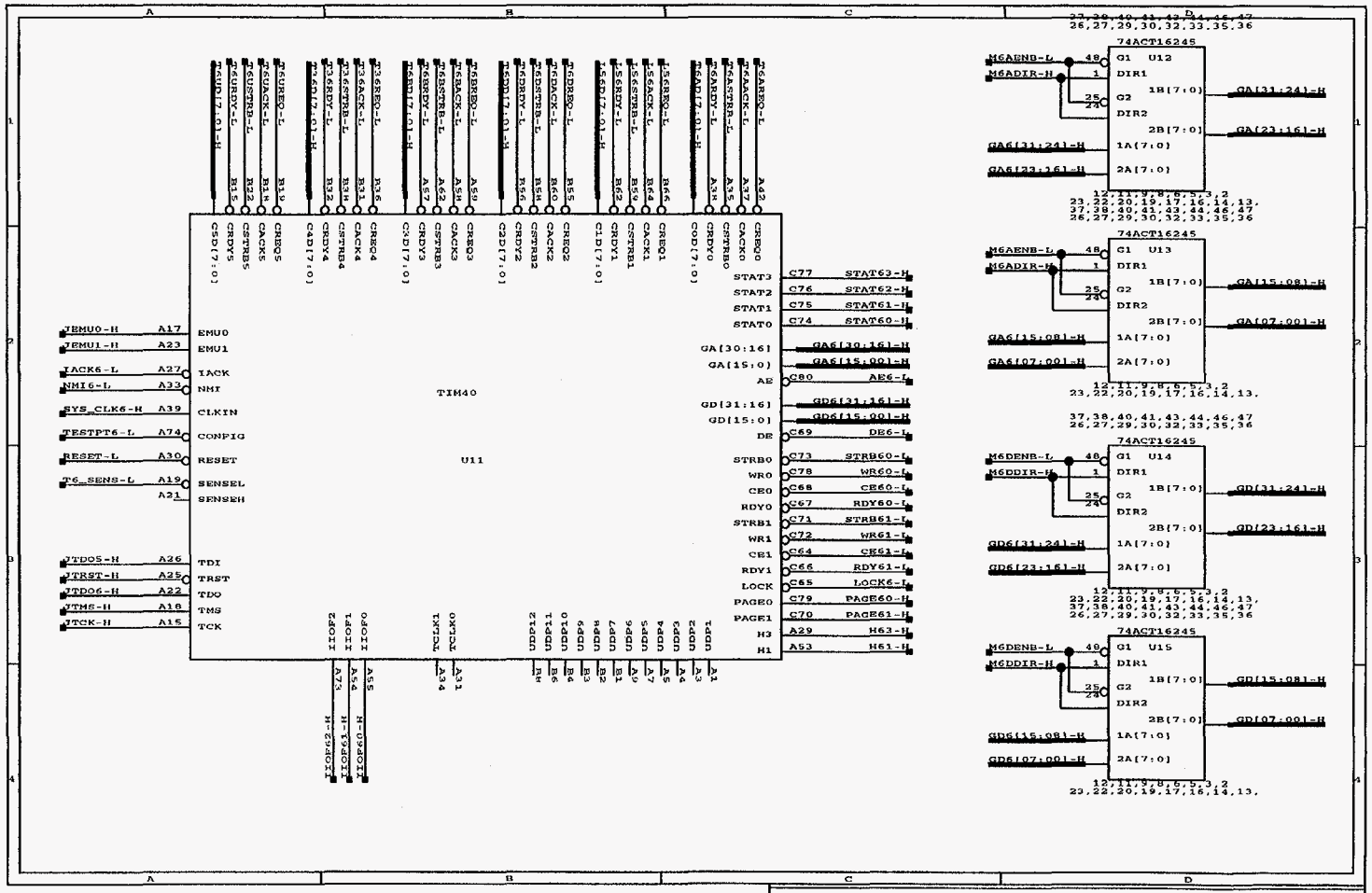
C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15

TIM Module 5


TIM40
U6
74ACT16245
U7
U8
U9
U10
UNM/Sandia - HyperForest Development
Board 2
PAGE 1 OF
REV. -
DRAWN BY:

B20,B24,B23,B16,B26,B14,B28,B11 B27,B34,B42,B35,B30,B44,B39,B40 A71,A70,A69,A65,A67,A63,A66,A61 B52,B43,B50,B48,B47,B46,B54,B51 A41,A46,A43,A50,A47,A49,A45,A51 B68,B71,B67,B70,B63,B74,B72,B76

TIM Module 6



C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15

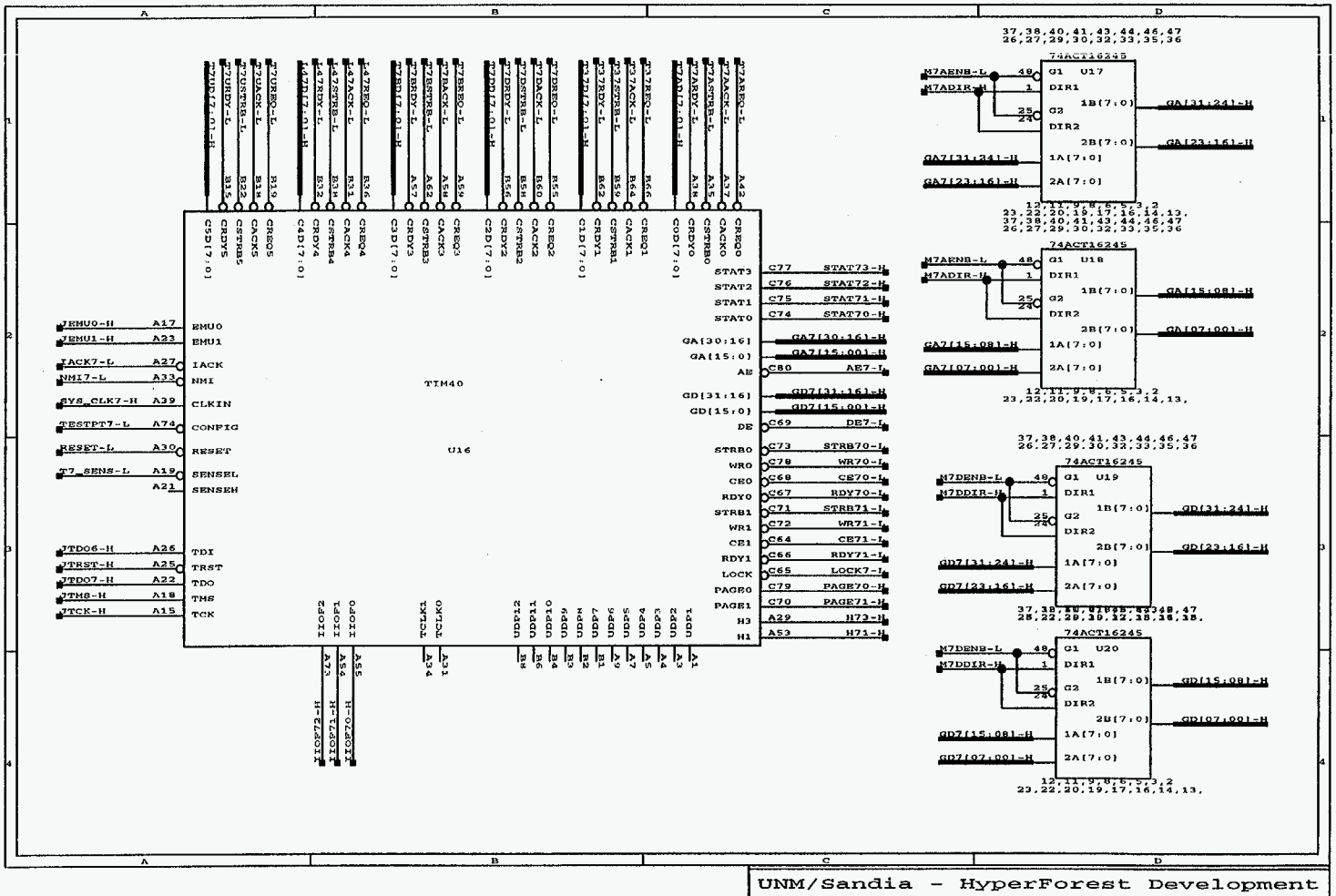
UNM/Sandia - HyperForest Development		
Board 2	PAGE 1 OF	REV. - DRAWN BY:

B20,B24,B23,B16,B26,B14,B28,B11
B27,B34,B42,B35,B30,B44,B39,B40

A71,A70,A69,A65,A67,A63,A66,A61
B52,B43,B50,B48,B47,B46,B54,B51

A41,A46,A43,A50,A47,A49,A45,A51
B68,B71,B67,B70,B63,B74,B72,B76

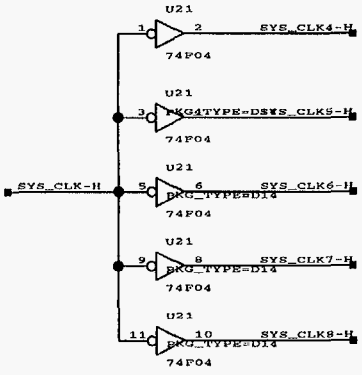
TIM Module 7

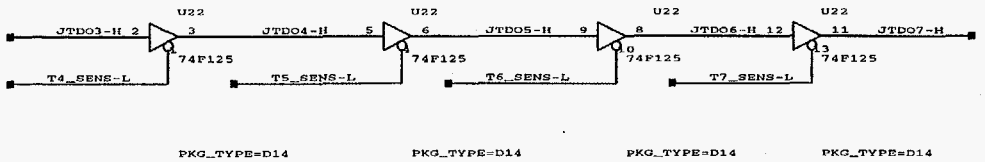


UNM/Sandia - HyperForest Development	
Board 2	PAGE 1 OF
REV. -	DRAWN BY:

C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15

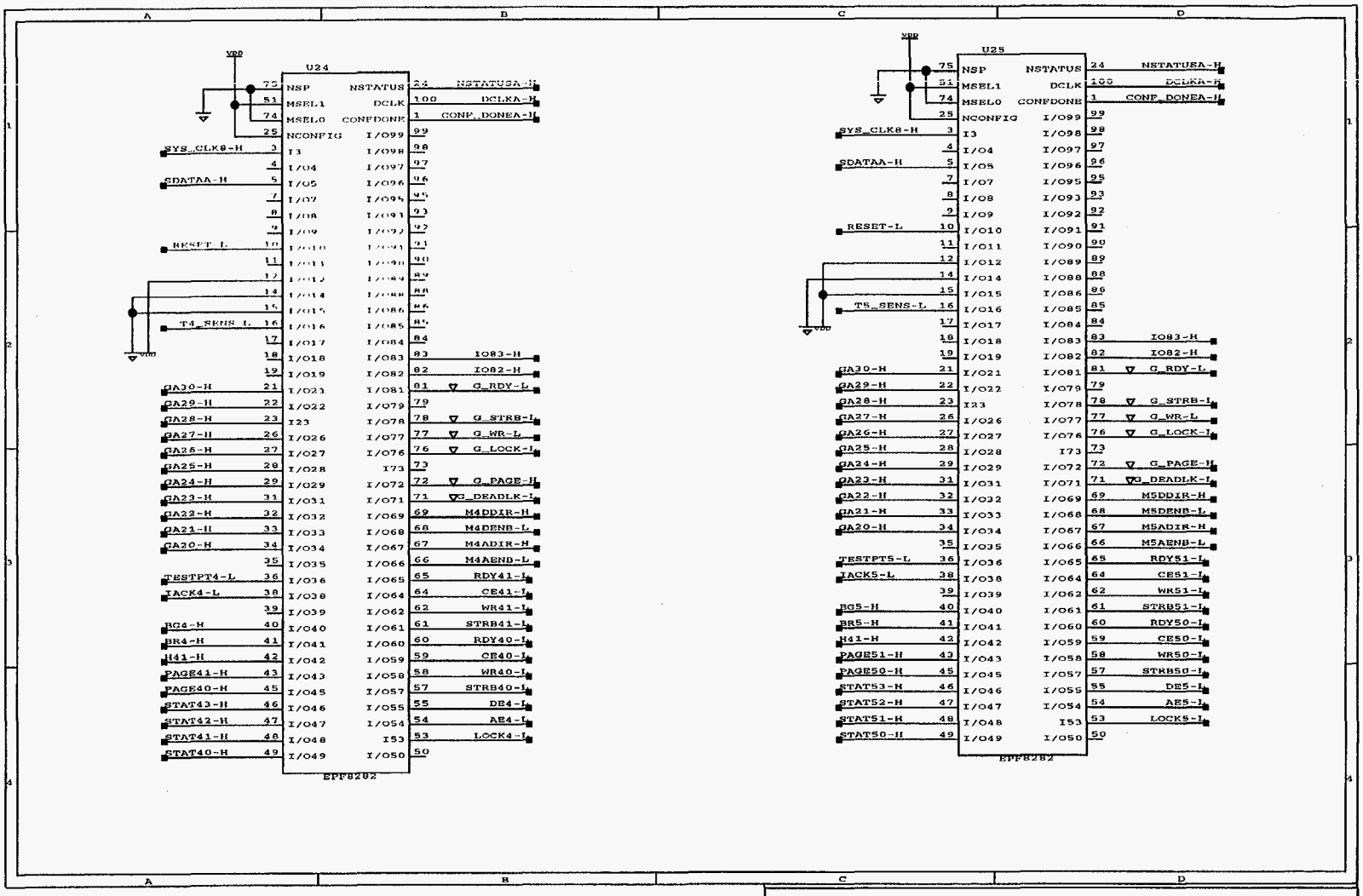
Clock, Pull Ups, JTAG Chain





UNM/Sandia - HyperForest Development		
Board 2	PAGE OF	REV. - DRAWN BY:

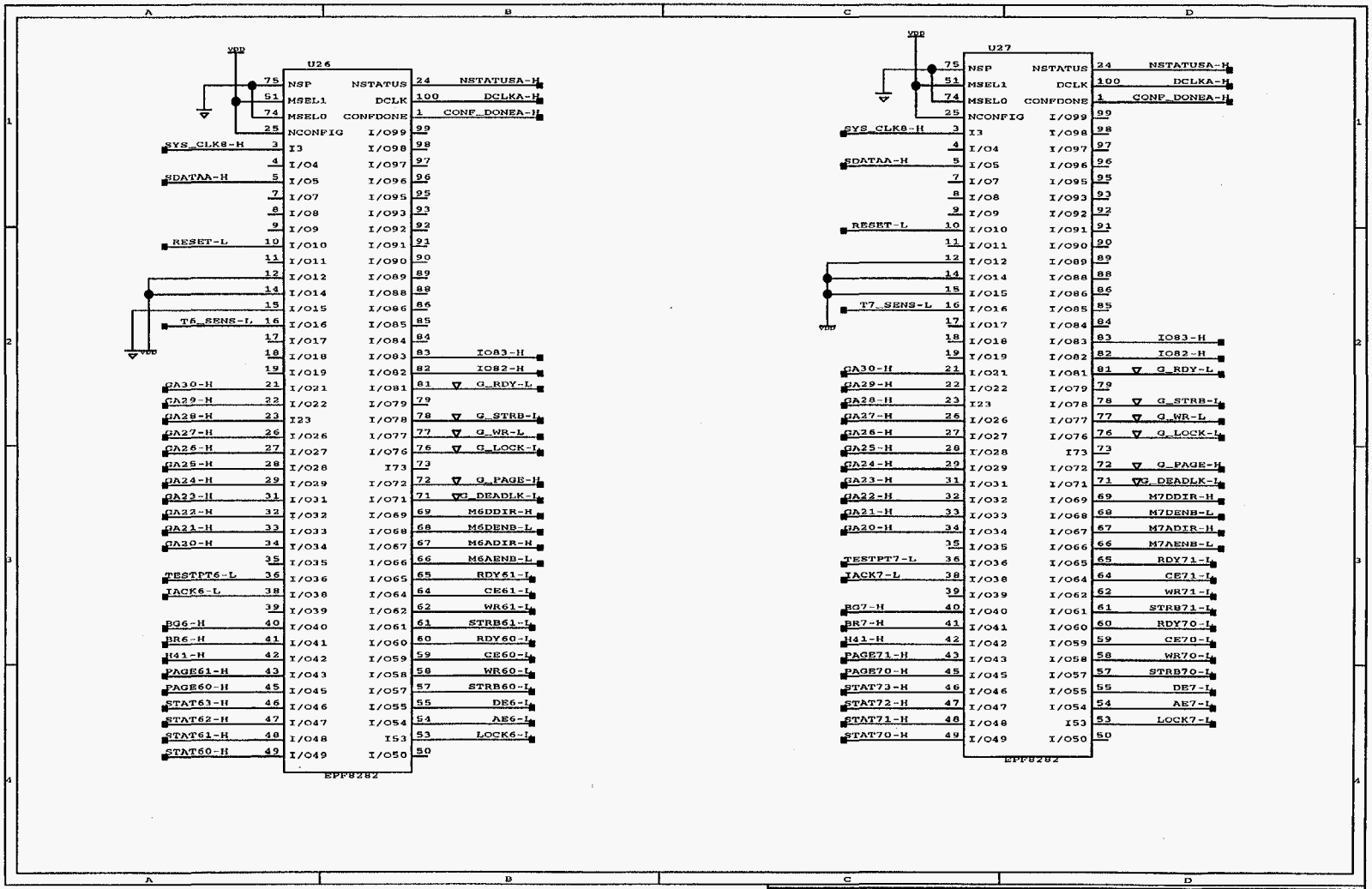
Bus Control, Modules 4 and 5



UNM/Sandia - HyperForest Development

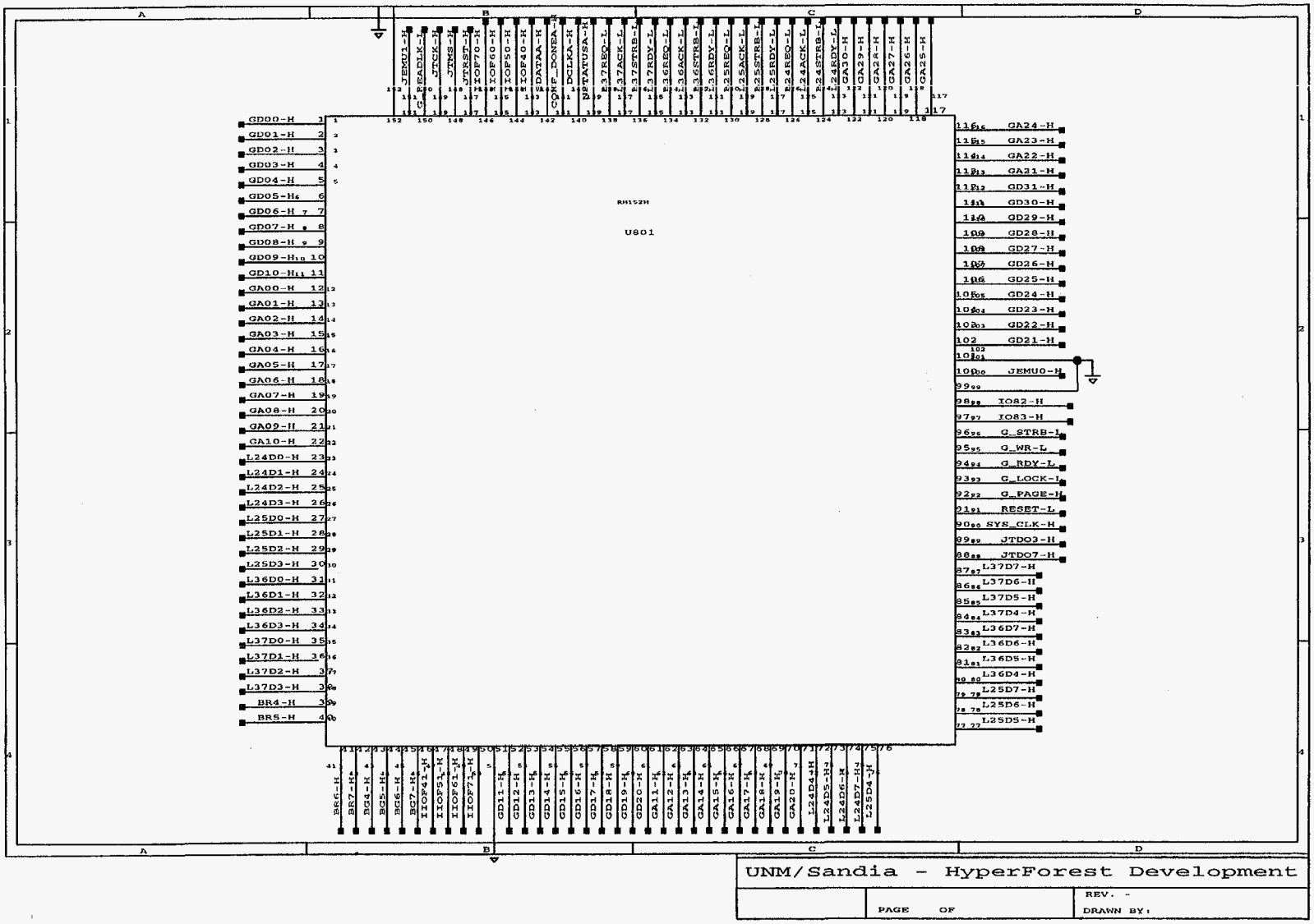
	PAGE OF	REV. - DRAWN BY:

Bus Control, modules 6 and 7



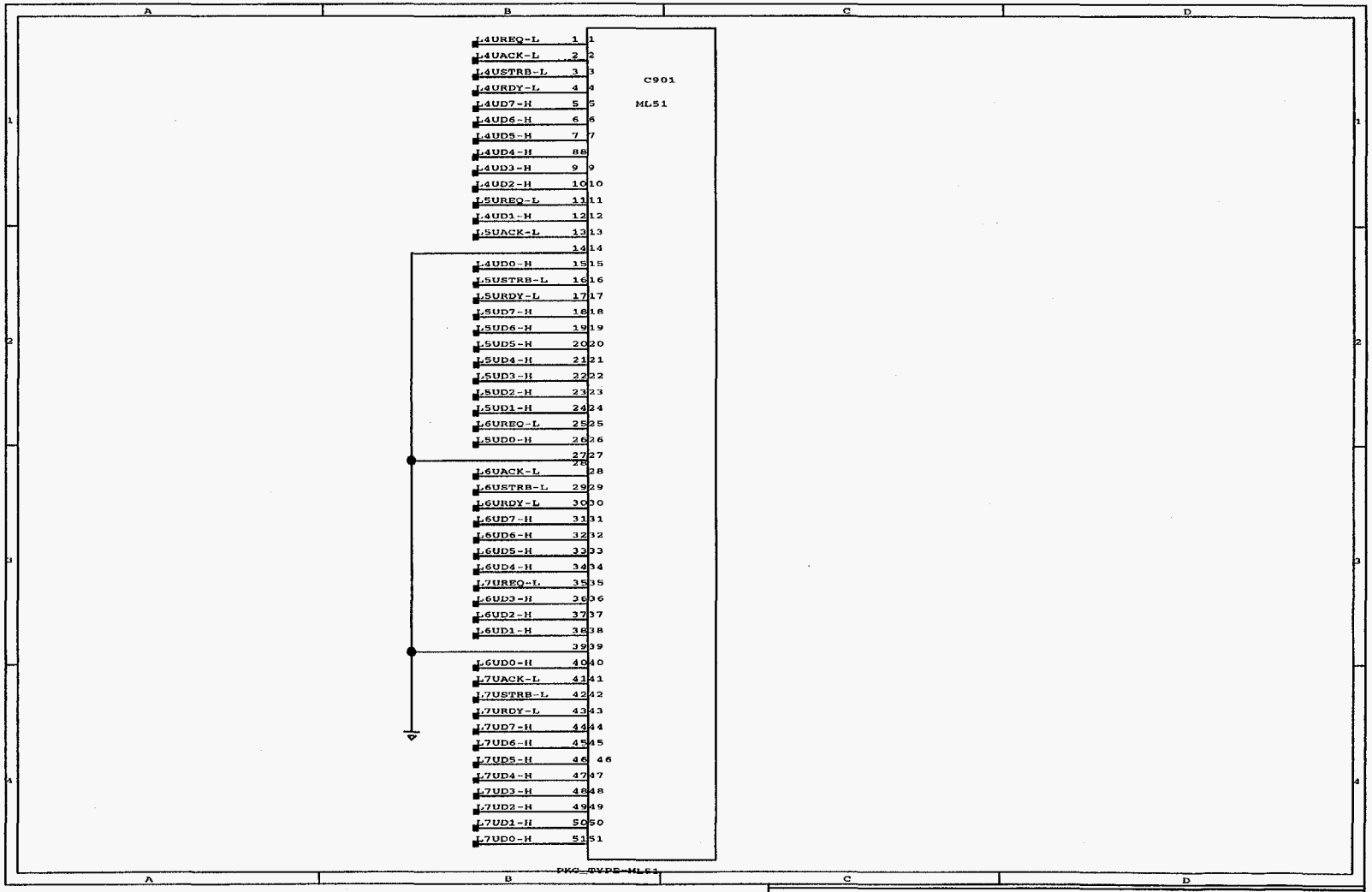
UNM/Sandia - HyperForest Development		
	PAGE OF	REV. - DRAWN BY:

Interboard Connectors



UNM/Sandia - HyperForest Development		
	PAGE OF	REV. - DRAWN BY:

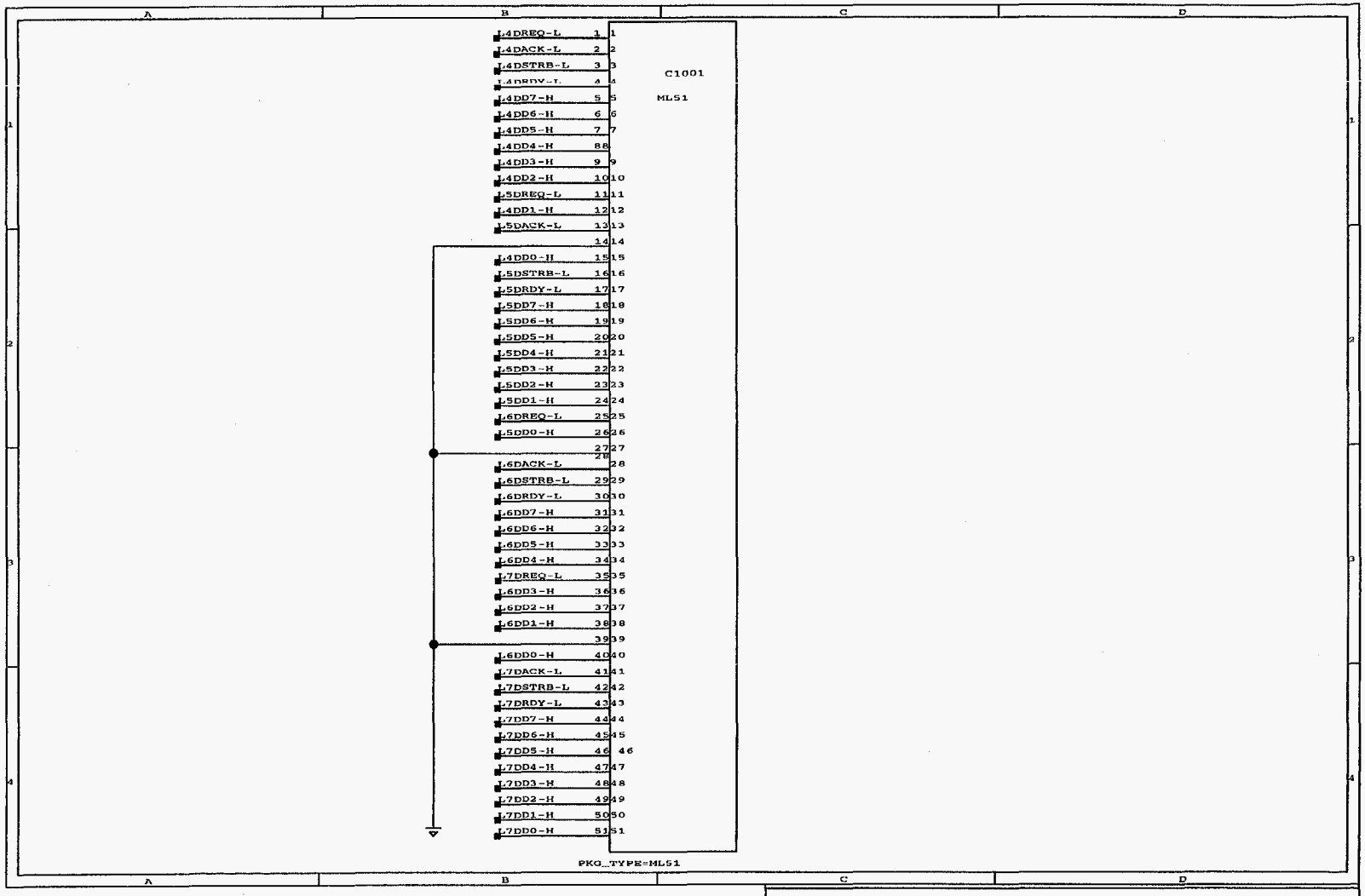
Comm. Port Connectors,Up Links



UNM/Sandia - HyperForest Development

	PAGE OF	REV. - DRAWN BY:

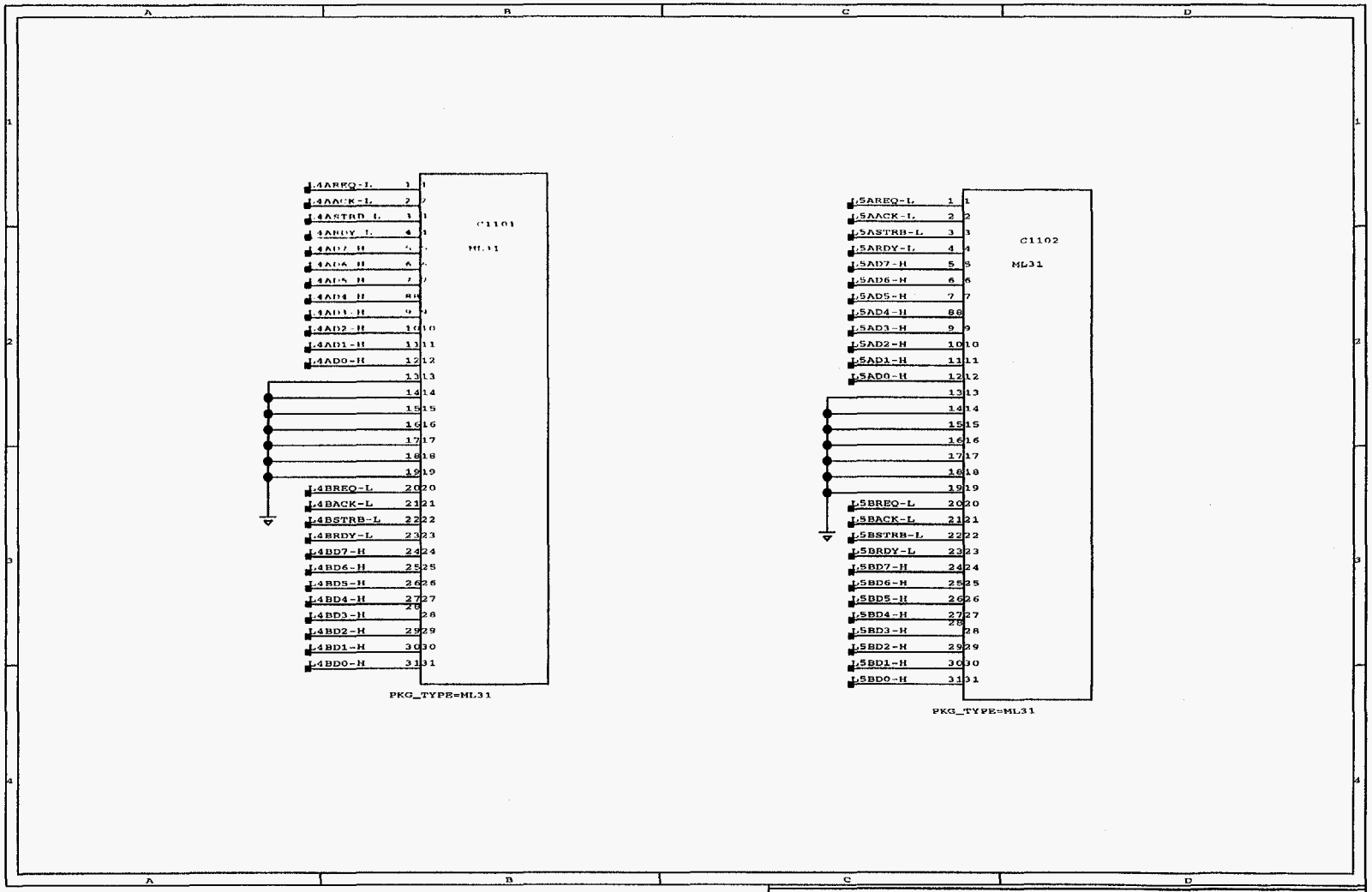
Comm. Port Connectors, Down Links



UNM/Sandia - HyperForest Development

	PAGE OF	REV. - DRAWN BY:

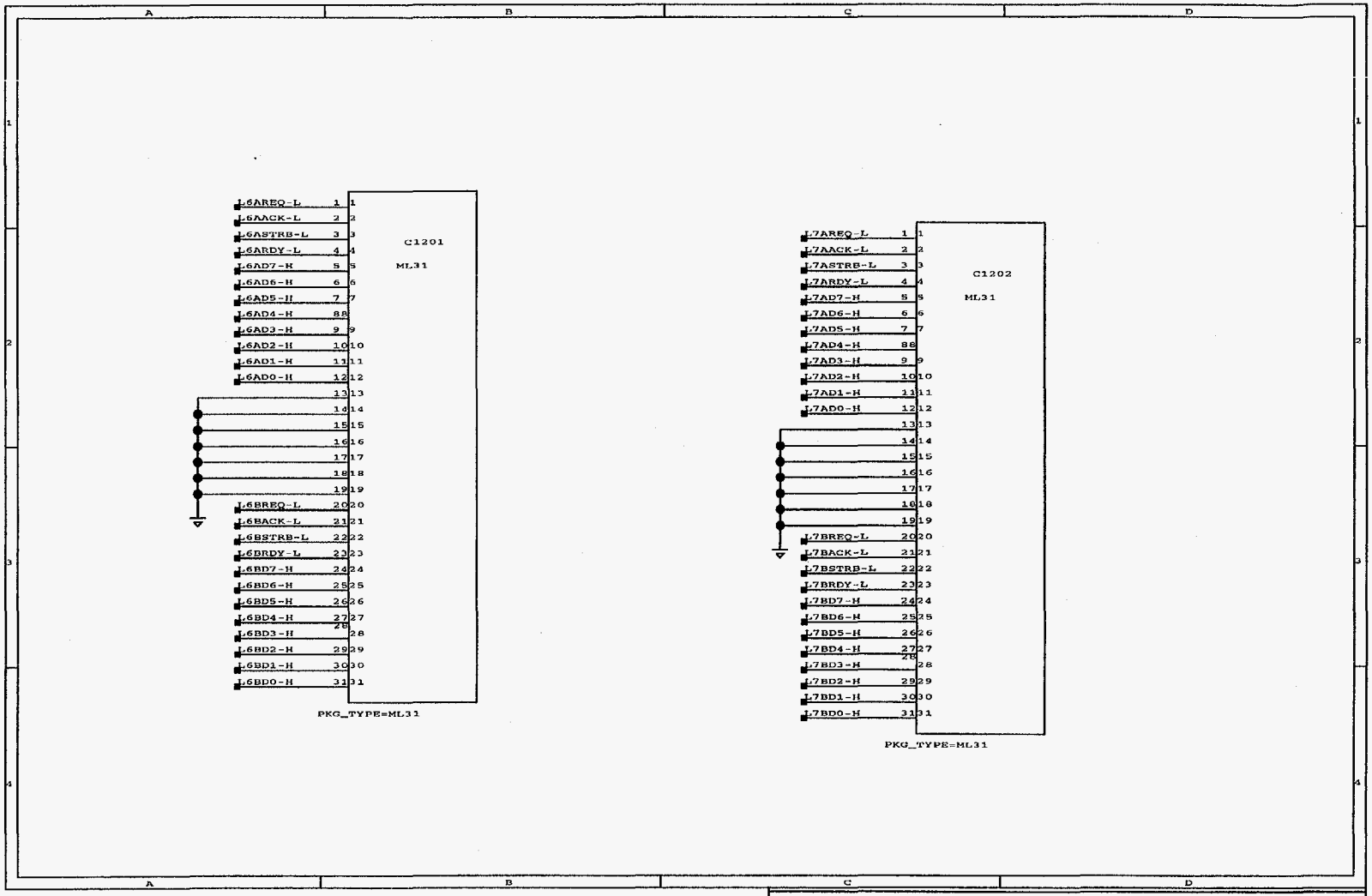
Comm Port Conn, extra links 4 and 5



UNM/Sandia - HyperForest Development

	PAGE OF	REV. - DRAWN BY:

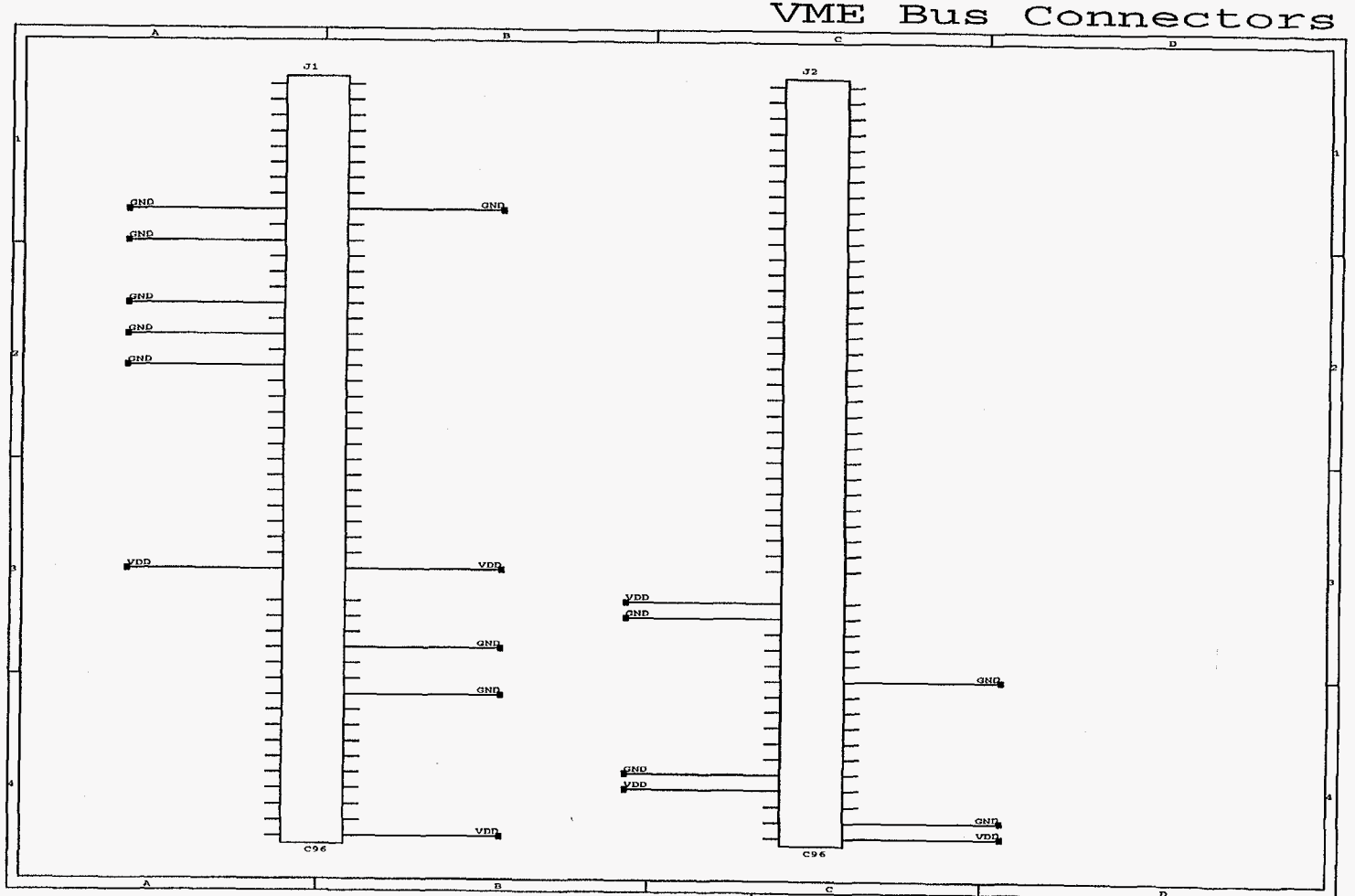
Comm Port Conn, Extra Links 6 and 7



UNM/Sandia - HyperForest Development

	PAGE OF	REV. - DRAWN BY:

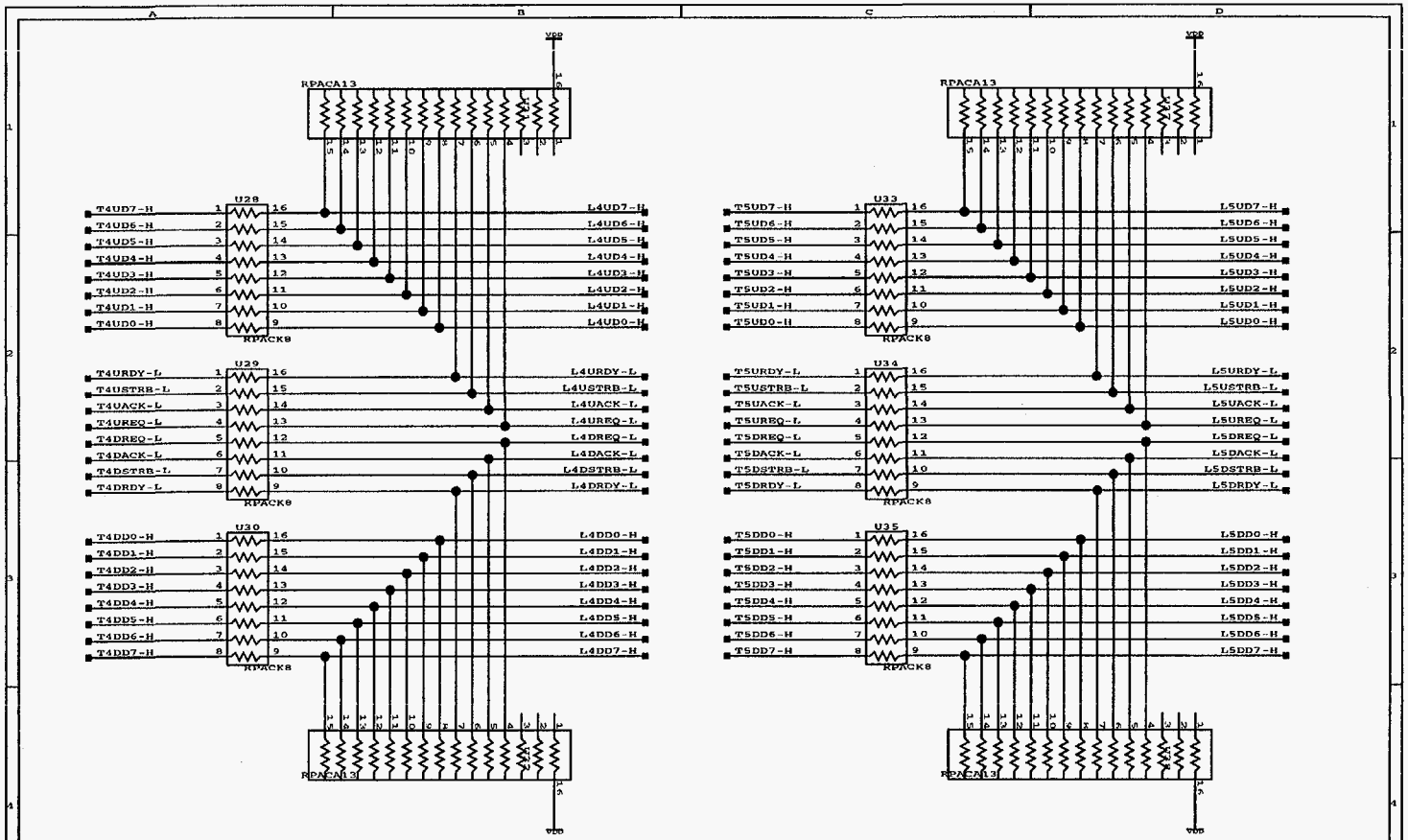
VME Bus Connectors



UNM/Sandia - HyperForest Development

	PAGE OF	REV. - DRAWN BY:

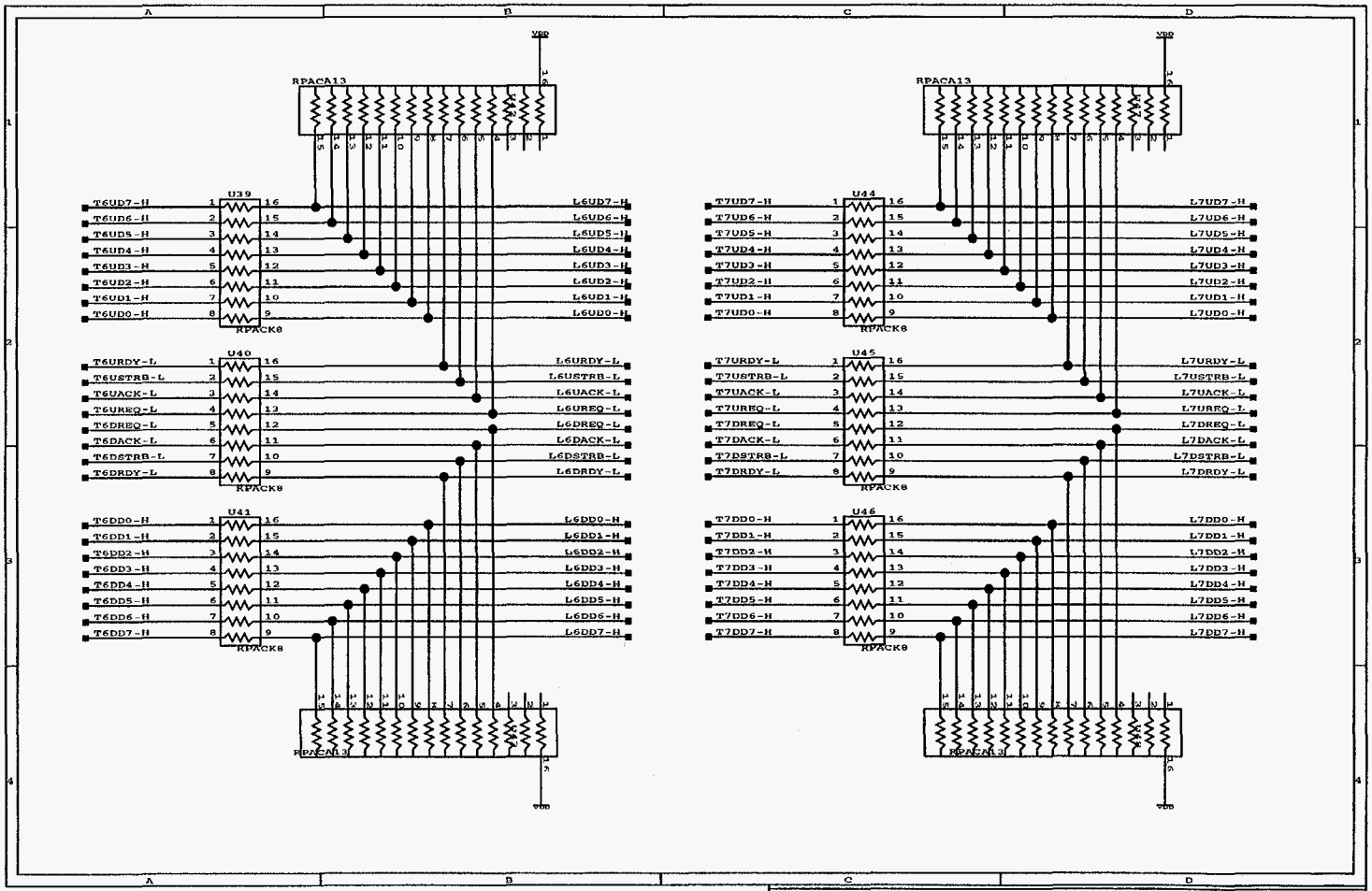
Termination Resistors



UNM/Sandia - HyperForest Development

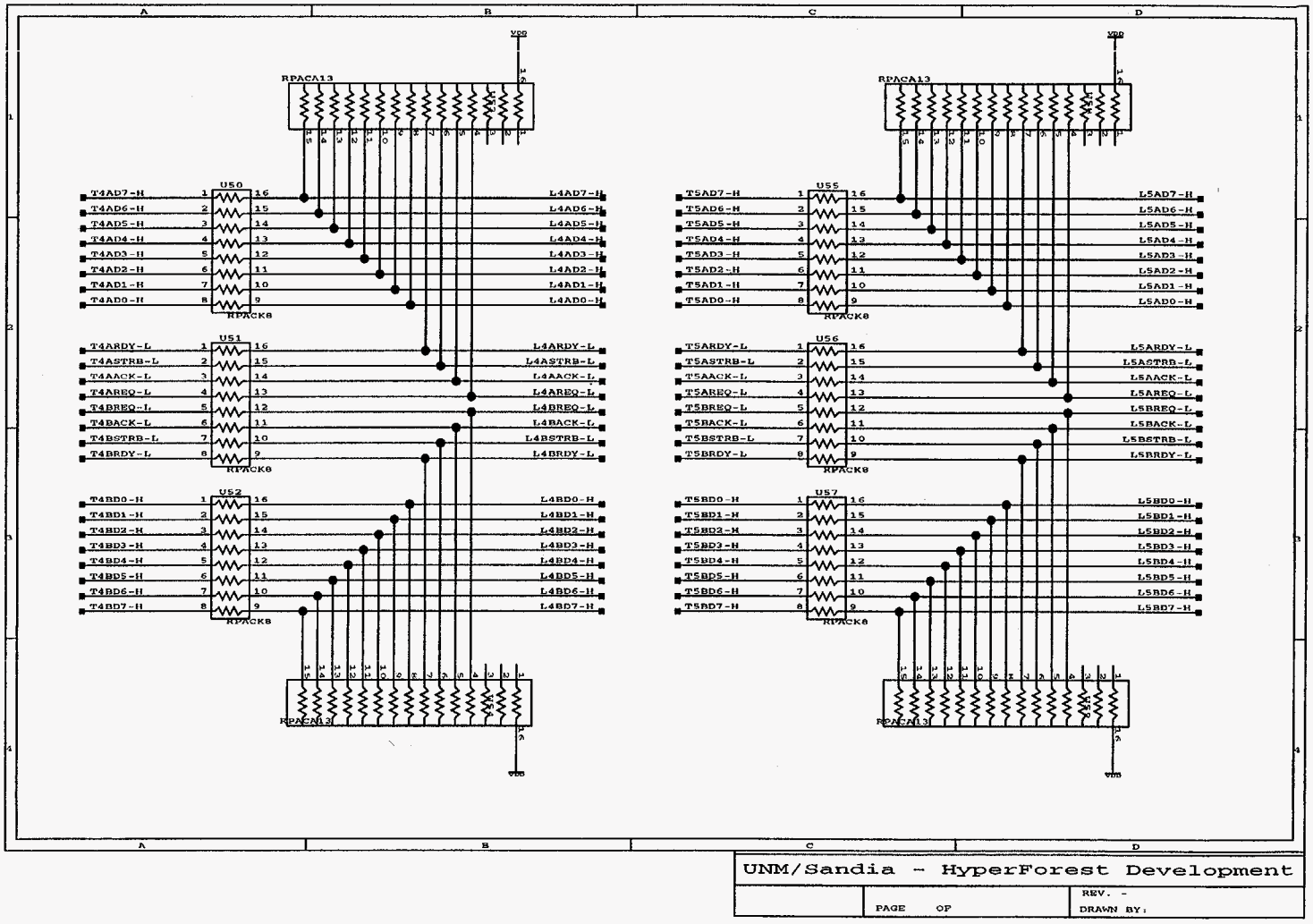
	PAGE OF	REV. - DRAWN BY:

Termination Resistors



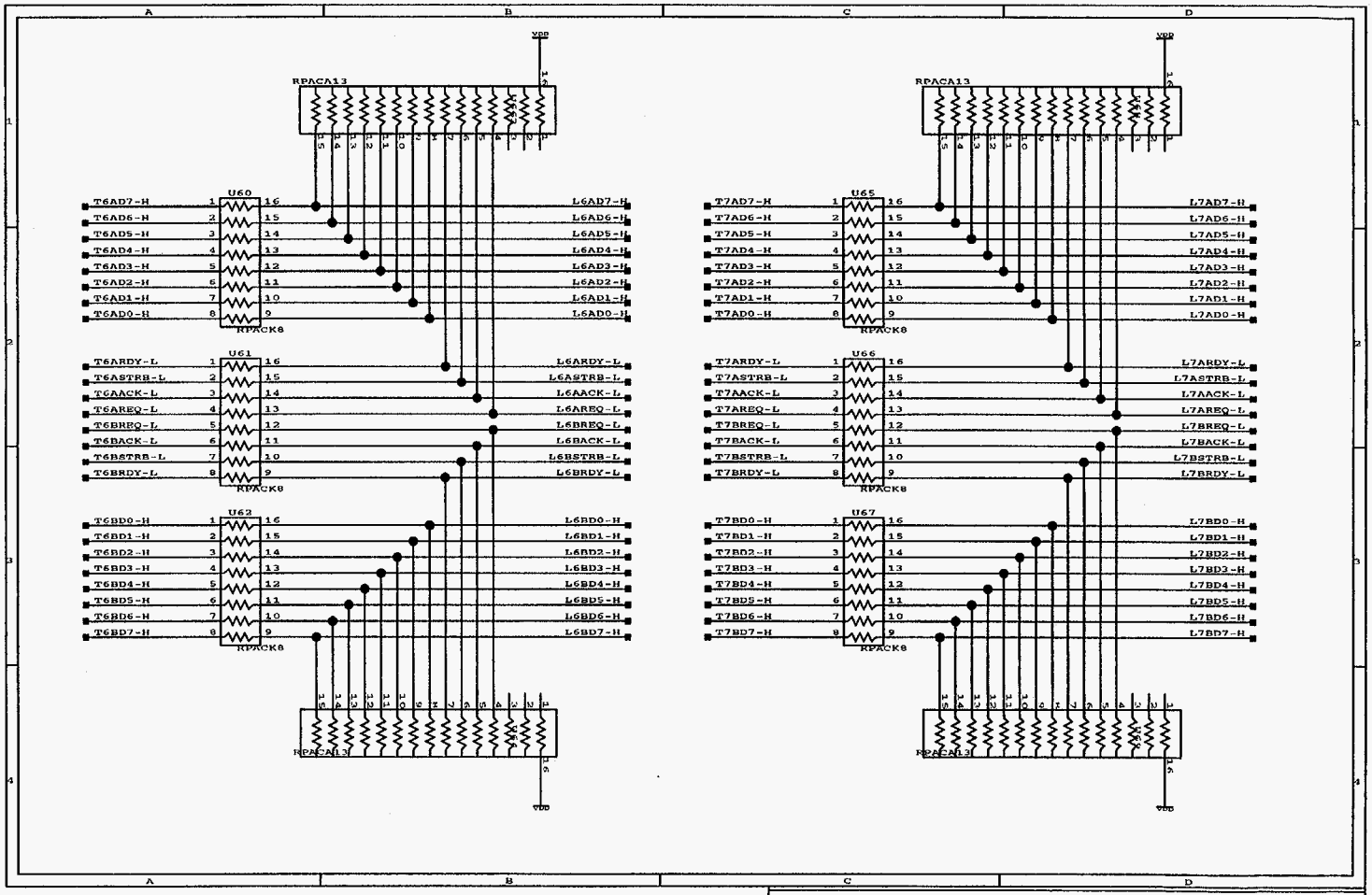
UNM/Sandia - HyperForest Development		
	PAGE OF	REV. - DRAWN BY:

Termination Resistors



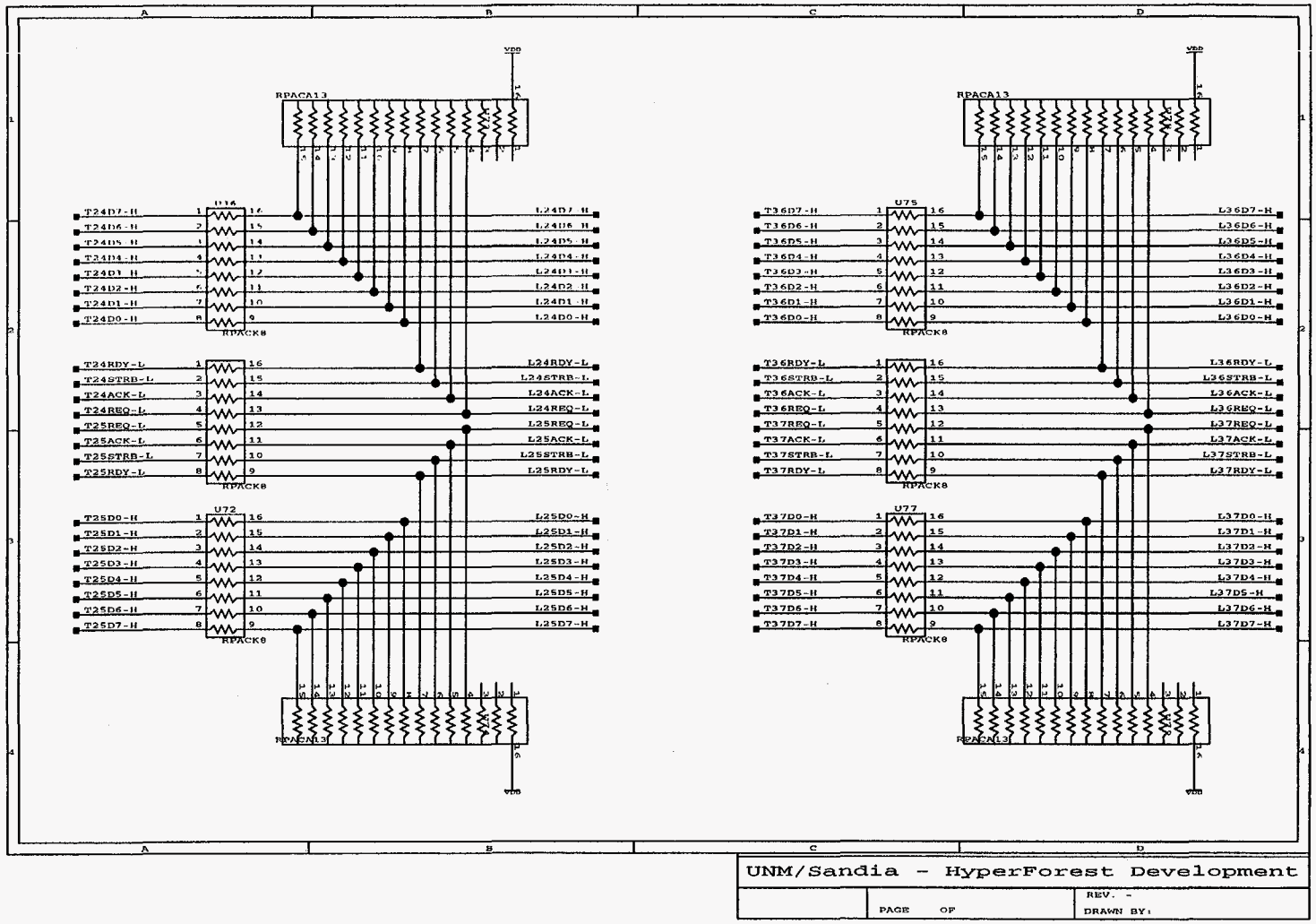
UNM/Sandia - HyperForest Development		
	PAGE OF	REV. - DRAWN BY:

Termination Resistors

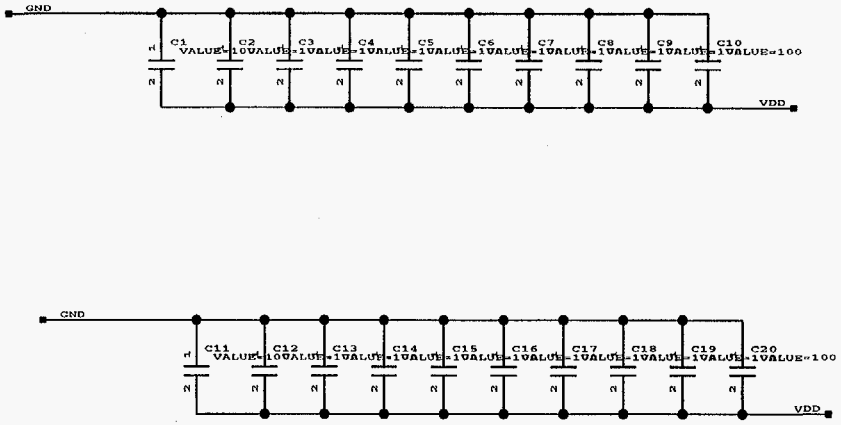


UNM/Sandia - HyperForest Development		
	PAGE OF	REV. - DRAWN BY:

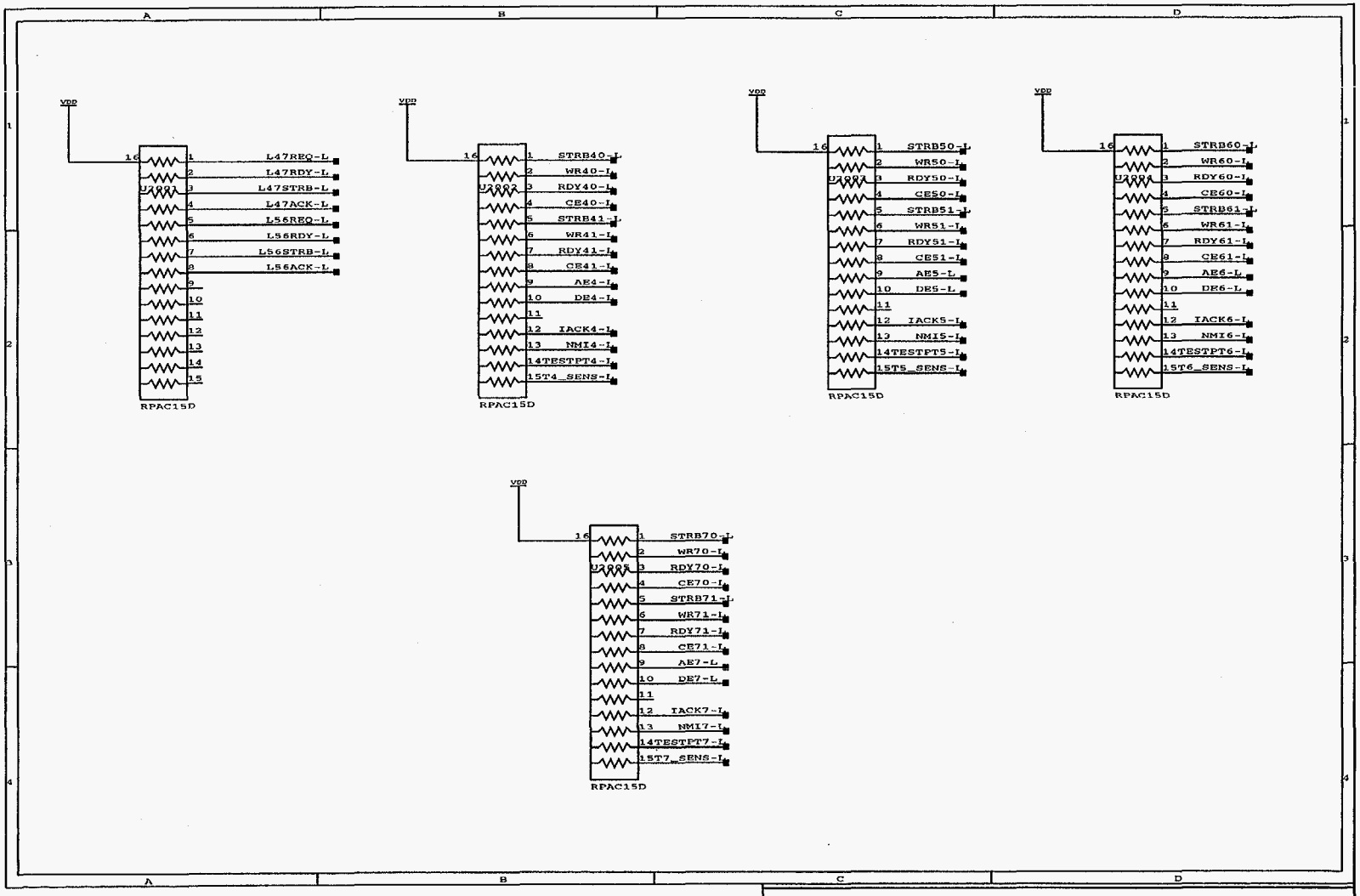
Termination Resistors



UNM/Sandia - HyperForest Development		
	PAGE OF	REV. - DRAWN BY:



UNM/Sandia - HyperForest Development		
	PAGE OF	REV. - DRAWN BY:



UNM/Sandia - HyperForest Development

	PAGE OF	REV. - DRAWN BY:

DISTRIBUTION:

2	MS-0619	Review & Approval Desk, 12690 For DOE/OSTI
1	MS-0188	LDRD Office, 4523
5	MS-0899	Technical Library, 4414
1	MS-1002	P. J. Eicker, 9600
5	MS-1006	P. Garcia, 9671
3	MS-1006	J. P. Rebeil, 9671
3	MS-1006	H. Pollard
1	MS-1007	A. T. Jones, 9672
6	MS-1008	S. Blauwkamp, 9621
1	MS-9018	Central Technical Files, 8940-2