The Reliable Router: An Architecture for Fault Tolerant Interconnect

by

Larry R. Dennison

B.S., Massachusetts Institute of Technology (1980)
S.M., Massachusetts Institute of Technology (1991)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author ... Larry R. Dennison .....
Department of Electrical Engineering and Computer Science
May 24, 1996

Certified by .....
William J. Dally
Professor, Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by .....
F.R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

The Reliable Router: An Architecture for Fault Tolerant Interconnect

by

Larry R. Dennison

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 1996, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The Reliable Router advances the state-of-the-art in high-speed, fault-tolerant communication. It does so through novel ideas in the following areas: router architecture, fault-tolerant networks, interchip signalling, and data retiming (synchronization).

The Reliable Router is a VLSI device implementing a message-passing communication substrate for parallel computers and other micro-area networks. The network topology is a 2-dimensional mesh. It uses wormhole routing, 5 virtual channels per physical channel, 2 priority levels, minimally adaptive and one-fault-tolerant routing. Bandwidth between routers is 3.2Gbit/s, each way. Router latency is about 70ns. The router has been fabricated on a 13.5mm by 15mm die, packaged in a 463-pin PGA, and partial testing performed. A description of the architecture and schematics are given. The design methodology is presented.

The Unique Token Protocol is used to provide fault-tolerance in the network. The protocol keeps two copies of a message in the network, obviating the need for source buffering. A token is used to determine the unique (exactly-once) delivery of a message, nearly completely eliminating the need for duplicate detection at the receiver. Details of the protocol under wormhole-routing are also provided.

Simultaneous Bidirectional Signalling is used for high-performance interchip communication, 200Mbit/s each way per wire. A single wire carries full duplex communication concurrently through the use of waveform superposition. Circuits, noise-reduction techniques, and analysis are given.

Low-Latency Plesiochronous Data Retiming is used for interchip communication. Each router operates in a separate clock domain. Latency is minimized by placing the synchronizer out of the data path. The basic retiming method is fully described, along with circuits and constraints for correct operation. Average latency is one-half a clock period. Extensions for integral subrates and cascaded timing circuits are given.

Thesis Supervisor: William J. Dally
Title: Professor, Artificial Intelligence Laboratory

Acknowledgements

"It's a long hard climb, but I'm gonna get there."

from a Sesame Street song

I've relied on many people over the years. I would like to say thank you to the following people:

Bill Dally, for getting me started, for all his support along the way, and for bringing this project to a successful completion.

My thesis readers, Gill Pratt and Tom Knight, who supplied energy to a rather tired student.

Scott Furman, a great officemate.

The people who worked on the router over the years, Whay Lee, Kin Hong Kan, Ivan Oei, Jonathan Rosenberg, Dan Hartman, Jeff Bowers, and Dave Harris.

Eileen Neilson for getting the no-cost extensions through.

Tom Blackadar for understanding "I gotta get done."

The CVA group, for all their moral support.

Duke Xanthopoulos, officemate, fellow router designer, and among the best of men.

My father and mother, Daniel and Dolores, who encouraged me to succeed by teaching me not to fear failure.

My father-in-law and mother-in-law, James and Lee Moses, for their support (and child-watching).

My children, Daniel, Kathryn, Larry, and Timothy, for their patience while Dad did this.

My wife Judy, who has done more for me than I ever had any right to ask and yet still loves me.

God in heaven, who watches over all I do.

For Judy,
My one and only.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Parallel computers have always needed fast, robust communication between processing elements. The computation being performed usually can not be done by many processors operating in isolation. Intermediate results are shared, work is produced and consumed, and resources are shifted. This is true even when the processors do not directly communicate with each other, as in a shared memory machine.

The flexibility of that communication depends on the agility of the communication fabric. Delay or latency in the communication path often means that processors will go idle while waiting for a piece of information. Low bandwidth, relative to the processors ability to produce or consume it, will result in the processors exchanging less information and recomputing more. The effective utilization of the processors depends on the existance of a high-performance communication fabric.

That same communication fabric could stop the construction of a large parallel machine, owing to the complexity and/or undependability of its realization. System-wide clock skew, exotic cabling, and monolithic power supply distribution will either preclude a large machine on technical grounds or simply cause it to be financially impractical. The communication network needs to be reliable and composable into larger networks, yet be relatively inexpensive.

This thesis covers the design, construction, and testing of a VLSI communication device, the *Reliable Router*. The Reliable Router project was formed to meet the needs outlined above.

The Reliable Router Project

The theme of the Reliable Router project is the development of ideas and techniques which will enable the construction of large, high-performance, reliable parallel computers. That theme implies that there are obstacles to overcome and these should be explicitly stated:

- As a class, larger machines tend to be less reliable than smaller machines. Siewiorek and Swarz define reliability as a probability function of time [33]. This function $R(t)$ gives the conditional probability at time t that the machine

has remained operational in the interval $[0, t]$, given that the machine was operational at time $t = 0$. For large parallel computers, a noticable amount of VLSI and nearly all of the system wiring goes into the interprocessor interconnect. If care is not taken, the expected failures in the interconnect fabric can severely reduce the expected reliability of the entire machine.

- The difficulty to physically realize and maintain a large machine is often overlooked. The router should be designed to facilitate the construction of large machines by limiting the number of global resources such as clocks or power supply voltages required.

- Performance is always an issue. If a machine has a mean-time-between-failures (MBTF) of 1 hour but is ten times as fast as a machine which fails every two hours, the faster machine is still more likely to complete any given computation. Thus, a better metric when evaluating alternative designs is useful work accomplished between failures.

- Interprocessor signalling is limited by the number of physical wires passing through some interface. The interface could be a chip package, a PC board connector, or the number of wires in a cable. Increasing the number of bits/second-wire is always important.

The Reliable Router addresses these obstacles through the following feature set:

- It provides a reliable communication substrate through the use of the Unique Token Protocol. Single point failures do not cause messages to be lost or duplicated.

- The raw communication channel is high-performance. Peak bidirectional (full-duplex) bandwidth between two routers is 6.4 gigabits per second. Latency is approximately 70ns.

- Minimally adaptive routing is coupled with virtual channels to provide additional hot-spot and contention avoidance.

- Deep message buffers help free up virtual channel resources when a message is blocked.

- The number of I/O's is reduced through the use of simultaneous bidirectional signalling.

- The router is designed to obviate the need for a global system clock through the use of plesiochronous data retiming.

Thesis Outline

The router draws inspiration from many sources. This thesis begins with a discussion of prior and related work, presented in Chapter 2.

Chapter 3 gives an overview of the Reliable Router. It provides the context for the presentation of the key contributions which occur in later chapters. The ideas for a fault tolerant protocol, improved interchip signalling, and better data retiming resulted from solving the router design problem.

The first contribution is the Unique Token Protocol. In Chapter 4, the protocol is described as the composition of two components, flow-control and token passing. These components are proven/argued to be correct in the packet-case and then implementation details for wormhole routing are given.

Simultaneous Bidirectional Signalling is the second contribution. Chapter 5 begin with a discussion of signalling and noise sources and works through the problems and solutions of unidirectional signalling. Those basic techniques are then adapted for bidirectional signalling. Finally, a comparison of unidirectional versus bidirectional is made.

The third contribution is Plesiochronous Data Retiming. In chapter 6, the concepts are developed for point-to-point retiming. Timing constraints are given. Finally, the retiming strategy is extended to allow cascaded connections.

The last contribution is the detailed microarchitecture of the Reliable Router, presented in Chapter 7. There is no better way to understand the complexity of a routing algorithm or flow-control policy than to look at the detailed design. The design methodology used to construct the chip is described in Chapter 8.

The Reliable Router has fabricated and designed into a test board. Chapter 9 briefing describes the testing results, as well as the chip errata and desirata. Finally, Chapter 10 summarizes the results and gives future directions.

Chapter 2

Prior and Related Work

The thesis is composed of several ideas from different areas. The areas are fault tolerance, interchip signalling, data retiming, and router architecture. Each of these will now be explored in turn.

2.1 Fault Tolerance

There are two basic requirements for a fault-tolerant network. The first is a routing algorithm which will go around faults. The second is a protocol for failure recovery.

2.1.1 Routing Around Faults

Xanthapoulos[37] describes the routing algorithm used in the Reliable Router, Reliable Adaptive Routing. It is worth restating the routing algorithm evaluation criteria and reviewing the algorithms which influenced Reliable Adaptive Routing.

The following properties are desired in a routing algorithm:

Stateless The routing algorithm should not modify the contents of the message. It is important that the destination be able to verify the message contents using simple mechanisms such as checksums. Altering the message contents invariably leads to more complex verification schemes.

Efficient The routing algorithm should be fast and not occupy a large amount of silicon area.

Dependence on Local Information The routing decision should be based on local information. Global knowledge distribution takes time and resources.

Reasonable Virtual Channel Requirements It should not require large amounts of virtual channels[10]. Virtual channels do not come cheaply[1],

[1] At least, high-performance implementations do not.

Minimality A routing algorithm is miminal if it only allows routing steps which bring it closer to its destination[28, 16]. Minimality is a desirable property for two reasons. In conjuction with

/fairness], minimality will guarantee freedom from livelock and starvation. Minimality also conserves network resources such as link bandwidth and buffers[6].

Full Adaptivity Full adaptivity implies that the routing algorithm can choose any of the possible paths between source and destination, subject to the minimality constraint[28, 18, 17, 1]. It is useful for networks under load imbalance, as the adaptivity allows a wider range of possible paths.

Fault Tolerance Fault-tolerance is the ability of a routing algorithm to bypass faulty communication links. The Reliable Router requires a particular form called One-Fault-Tolerant Routing[18], where the algorithm tolerates at least a single network failure.

Dimension Reversals

Dally and Aoki[7] developed two types of adaptive algorithms (*static* and *dynamic*) based upon the concept of dimension reversals. Both algorithms use multiple virtual channels. One of the virtual channels is reserved for dimension ordered routing. A dimension reversal occurs whenever a message in one of the non-dimension-ordered virtual channels changes dimension opposite the dimension order. The dimension reversal count is kept in the message and begins at 0.

The static version divides the virtual channels into τ classes where τ is the maximum number of allowed dimension reversals. A message with dimension reversal count less than or equal to α, $\alpha < \tau$, is allowed to use the virtual channels in class α. Once a message's dimension reversal count reaches τ, it mustbe routed in class τ using dimension ordering.

The dynamic version divides the virtual channels into adaptive and dimension-ordered classes. Messages originate in the adaptive class. While in the adaptive class, the dimension reversal count is kept as before. When a virtual channel is allocated to a message, it is tagged with a dimension-reversal count of that message. A message with a dimension reversal count of α cannot wait for virtual channels tagged α or lower. If all possible virtual channels are tagged α or lower, the message must be routed using the dimension-ordered channels. The message cannot re-enter the adaptive class.

As the algorithm stands, it is not suitable for use in the Reliable Router, as it modifies the message and is not fault-tolerant. Early work on the Reliable Router extended the dynamic algorithm to include fault-tolerance. However, it kept multiple virtual channels in reserve for fault handling. Further, the computation and communication requirements to determine when to shift to the dimension-ordered network proved to be unwieldly. This led the router team to look for other alternatives.

Dimension reversals did provide an important insight. One could divide the virtual channels into two virtual networks, one adaptive, the other dimension-ordered. One could perform any sort of routing in the adaptive network one wished, as long as the

routing algorithm noticed the conditions for deadlock and shifted to the dimension-ordered network.

The Linder-Harden Algorithm

Linder and Harden[23] took a different approach to provide adaptive routing. They divide the network into several virtual networks. Messages cannot transition between virtual networks. Each virtual network has no cycles in the channel dependency graph and is therefore deadlock-free. A mesh-connected k-ary n-cube requires 2^{n-1} virtual networks. Each virtual network requires either 1 or 2 virtual channels in each direction. Thus, a 2D mesh would require 4 virtual channels in the Y-dimension, and two virtual channels in the X-dimension.

This approach was seen to use far too many virtual channels and to use them in an unbalanced way. The insight gained from Linder-Harden was that it was possible to convert algorithms with deadlock into deadlock-free algorithms by partitioning the domain of the routing function, where each partition would use a different virtual network.

Duato's Contribution

For combined adaptive/dimension-ordered networks, Duato's contribution was to note that messages did not have to remain in the dimension-ordered network, they could freely return to the adaptive network[13, 14]. He noted that messages in the dimension-ordered channels always include other dimension-ordered channels in their "wait-for" graph, serving as an escape hatch.

Most of the problems with dimension-reversal counting ensued because the message was unable to return to the adaptive network. It was therefore desirable to postpone the entry into the dimension-ordered network as long as possible. Under Duato's model, simple deadlock detection mechanisms can be used in the adaptive network, as the performance penalty of shifting to the dimension-ordered network is slight.

Duato's model does not by itself provide fault-tolerance. The concept of shifting the message from virtual network to virtual network led to the search for the least expensive fault-tolerant routing algorithm which could be grafted on the adaptive/dimension ordered substrate.

Turn Model

The Turn Model[18, 17] was the least expensive algorithm, as it could adaptively route messages in a single virtual channel and was provably deadlock-free. Glass and Ni observed that it takes 4 turns in a plane to make a cycle and hence have a cyclic dependency. By prohibiting a routing algorithm from having a complete set of turns, they could prove the algorithm to be deadlock-free. This model is the basis for the fault-handling algorithm used in Reliable Adaptive Routing.

Planar Adaptive

Planar-adaptive routing[3] provided the ability to route around faults. It requires that the routing region be convex, which is difficult to ensure for faults on the edge of the network.

2.1.2 Fault Tolerant Protocols

The bulk of the dynamic fault-tolerant protocols are exemplified by the end-to-end (or point-to-point) variety [36]. These rely on the source keeping a copy, the destination performing duplicate elimination on incoming packets. Wide-area networks are typically composed of routers performing this level of fault-tolerance at the link-level. This link-level retry capability allows them to avoid end-to-end retries in the event of line noise. However, an upper level of end-to-end fault-tolerance is still required to tolerate a hard link failure or the failure of a router.

Simple end-to-end fault recovery mechanisms are assumed by Glass and Ni[18]. The uniform end-to-end nature allows the "goodness" of a routing algorithm to be measured by the number of virtual channels it uses and the number of faults the algorithm is able to route around.

Eslick et al. reduce the duplicate elimination problem by converting messages to idempotent forms[15]. The problem still exists for non-idempotent messages.

A brief discussion of a possible method for dynamic fault recovery can be found in [16]. Here, pipelined circuit switching (PCS) is used to route around static faults. Upon encountering a dynamic fault, the circuit is torn down and special flits are sent toward the sender and receiver indicating that a failure has occurred. The authors mention that this is an active research area, so no further details of the recovery mechanism are given.

2.2 Interchip Signalling

Simultaneous Bidirectional Signalling was developed by Dennison et al[20, 12, 22]. This technique for interchip signalling is fairly unique. Recently, several other researchers have extended our ideas. Intel and Hitachi both use series-terminated drivers[25, 35]. Lee et al. employ differential bidirectional signalling over twin-axial cable [21].

2.3 Low-Latency Plesiochronous Retiming

Plesiochronous retiming has been around for a while, although the solutions have had fairly large latency owing to a synchronizer delay. Related work:

- A retiming method where the synchronizer is moved out of the data path is described by Stewart and Ward[34]. They rely on the clock waveforms being periodic signals. Delay lines are used to *anticipate* the future interactions

between the data and the clocks. Transitions on the data signal are kept one-for-one between the domains. However, cells will be stretched and shortened by this retiming circuit. A flip-flop clocked using the local clock will still undersample or oversample the cells. In effect, the circuit repositions the data transitions to meet the setup and hold time rqeuirements for the local clock domain.

- Mesochronous retiming by insertion of delay in the receiver[30]. This technique creates various delayed versions of the incoming data. It then chooses one of the delayed versions to actually sample using the on-chip clock. The delay adjustment must take place while no data is being transferred. This scheme is also described by Cordell[4].

- Mesochronous retiming by insertion of delay in the transmitter[19]. Here, the delay is inserted by the transmitter instead of the receiver. The transmitter determines the correct amount of delay through either feedback from the receiver or by measuring the actual delay of the transmission line. The delay adjustment should take place while no data is being transferred.

- Rational clocking is a scheme for moving data between two devices whose clocks F_m and F_n are related $F_m = \frac{M}{N} F_n$, where M and N are integers[29]. The observation is that data movement between some pairs of clock edges will satisfy setup and hold times, other pairs of edges will not. Since the edge phase relationship is periodic, the determination of good edge pairs can be done in advance of the data movement and avoid synchronizer penalties.

2.4 Router Architecture

This section gives examples of router architectures and summarizes their features. There are a tremendous number of router/communication devices described in the literature, this sample only includes those deemed interesting.

Ariadne

The Ariadne router is an asychronous, adaptive, pipelined circuit-switched router[1]. Since it is circuit-switched, no data is transferred until the circuit is set up. The delay per node is thus the route of thc head + ack + forward data.

Forward Going Header	$13.75ns + 14.5ns + 2 \times 4.0ns$	$=$	$36.25ns$
Acknowledgement	$13.75ns + 7.5ns + 2 \times 4.0ns$	$=$	$29.25ns$
Data path	$8.5ns + 4.0ns$	$=$	$12.5ns$
Total			$78.0ns$

The data bandwidth is 57MB/s for a byte-wide data path, or a byte every 17.5ns. Ariadne is implemented in the same 0.8 micron process as the Reliable Router. Thus, it can be seen that the Reliable Router is comparable in terms of message latency and offers about 7x the bandwidth.

CHAOS Router

Bolding et al.[2] propose to augment a CHAOS router to detect and route around failures. They point out that the derouting capability of the chaotic algorithm is well-suited for routing around faults. However, the base Chaos router needed to be modified for fault-detection and recovery. The Chaos router uses virtual cut-through, messages can be cut in two during a dynamic fault. They then rely upon end-to-end mechanisms to recover from the failure.

Torus Routing Chip

The Torus Routing Chip (TRC) was the first routing chip designed for mesh and K-ary n-cube networks[9, 5]. It was intended for 2-dimensional networks, but $\frac{n}{2}$ TRCs per logical node could be used to construct an n-dimensional network. Dally[5] points out that the TRC is self-timed to avoid problems with global distribution. Interchip datapaths were byte-wide and operated at about 4MHz in a 3μ CMOS process.

Mosaic

The Mesh router family from Caltech is one of the more dominant/popular router families [32]. Fundamentally, these are asynchronous 2D mesh routers. They have either byte-wide or 16-bit wide data paths. They typically have oblivious X-then-Y routing and no virtual channels. Peak performance is about 100M data transfers per second per link, with node latency around 20ns.

J-Machine

The J-Machine router [27, 26, 8] was the direct ancestor of the Reliable Router. It features byte-wide data paths at 40Mbit/sec-wire, two priority levels, oblivious X-then-Y-then-Z routing.

Cray T3D

The Cray T3D[31] is a 3-dimensional torus. It uses ECL gate arrays clocked at 150MHz. The datapath is 16 bits wide and operates at 150Mbit/wire-sec. Latency is extremely low, about 2 clock periods or 13.3ns.

Chapter 3

Overview of the Reliable Router

The word *router* means different things to different people. The Reliable Router described in this thesis is a VLSI device to facilitate low-latency, high-bandwidth communications between "processors" in a multiprocessor system.

These processors communicate via *messages* and the principal function of the router is to transport these messages from one processor to another. This implies that the processors are all physically interconnected in some way. Further, physical limitations preclude all processors from being directly connected to all other processors. Instead, some sort of communication network is provided where processors share the communication resource.

The Reliable Router is designed to be the physical building block for a communication network. In particular, it can be used to construct two-dimensional mesh networks such as the one shown in Figure 3-1. Section 3.2 will describe how routers are physically interconnected to form a network.

One of the other functions of the router is to navigate the communication network during message transport. This function is implemently differently in other networks. The message may be explicitly routed by the sending processor (source routing), or the processor may simply provide an address for the message to be delivered to and rely upon the network to choose the route through the network (network routing). The Reliable Router adopts the second model and uses network routing. Section 3.1 will provide further details of the routing algorithm.

Allowing the network to handle the details of the routing is important for fault handling. In the case of source routing, knowledge of the fault would have to propagate back to the sending processor. Under network routing, the network is able to route around a fault using only local information. The advantage is that the network routing should be able to respond to a failure much more quickly. Further, the "local knowledge" only approach should provide better scalability for large machines.

The one feature which sets the Reliable Router apart from other routers is that it implements a link-level fault-tolerant protocol, the *Unique Token Protocol.* Fault recovery is handled by the routers physically near the point of failure. The sending processor is not involved and only minimal work is required by the receiving processors. Details of the protocol are provided in Chapter 4.

The remainder of this chapter focuses on describing the Reliable Router from

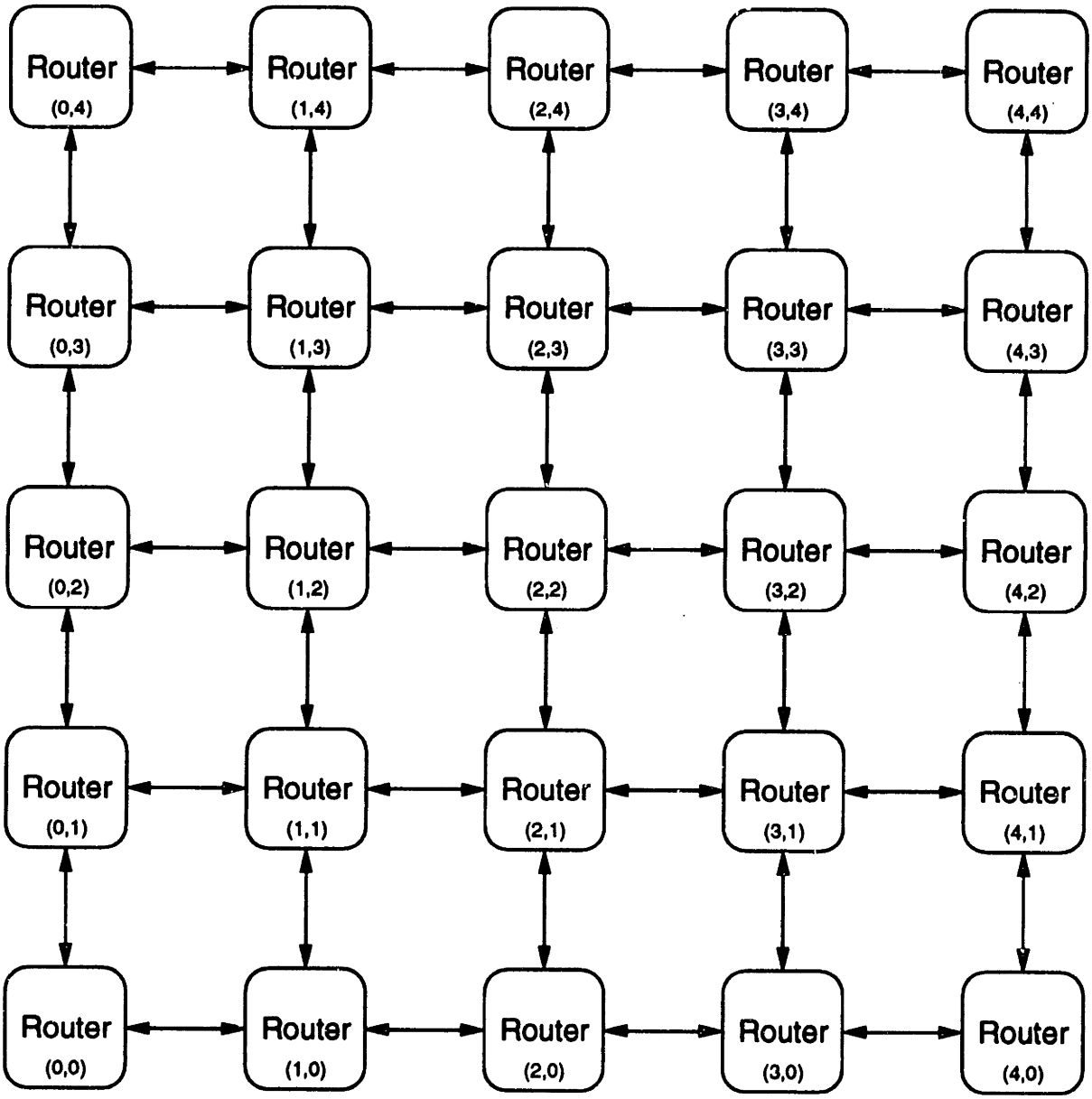


Figure 3-1: Two-dimensional mesh interconnection. Two-dimensional mesh interconnection. Here, 25 routers are arranged in a 5×5 mesh. The routers have addresses corresponding to their (X, Y) location in the mesh.

a interface perspective. Details of the processor interface are provided. Internal implementation details of the router are postponed until Chapter 7.

3.1 Routing

The network routing function is one of the more important aspects of the router from a user's perspective. It determines how the network responds to different traffic patterns. However, the ways in which the network may route a message are constrained by its deadlock avoidance technique. The ability to bypass faults further complicates matters.

Fortunately, there are several good techniques to help build good routing functions which meet the constraints. These techniques all view the physical network as several virtual networks, each with the same topology. However, different routing functions can be used for each virtual network and rules are applied which govern the ability of a message to move from one virtual network to the other. For example, a simple dimension-ordered mesh network can deadlock when transporting messages for a request/response protocol. The deadlock can be avoided by transporting requests in one virtual network, responses in a second virtual network.

An additional level of abstraction, *virtual channels*, may be useful in understanding the overall routing function. When two virtual networks both use the same routing function and impose no restriction on the movement of messages between those networks, it may be easier to think of those two as being a single virtual network. This single network would then have two virtual channels in which one could place messages.

The Reliable Router uses four virtual networks for routing. They are called *minimally adaptive*,*dimension ordered 0*,*dimension ordered 1*, and *fault handling*. Each network performs local routing, i.e. each router examines the address of the message and makes a decision based only on that address and the local state of the router.

3.1.1 Minimally Adaptive Routing

The minimally adaptive network is intended to carry the bulk of the network traffic. When a messages is first sent by a processor, it goes into this virtual network. Most the time, a message has to travel both in X and Y directions to reach its destination. The minimally adaptive network chooses either the X or Y direction, depending on the local congestion. It is called minimally adaptive because each individual routing step must always result in the message being closer to the destination. The message can never be routed away from the destination.

This virtual network has two virtual channels to help mitigate head-of-queue blocking delays. In the Reliable Router, the minimally adaptive network is biased toward using Y channels over X channels. Thus, even if one Y channel is in use and both X channels are free, the router will choose the second Y channel[1].

[1]This simplified the implementation.

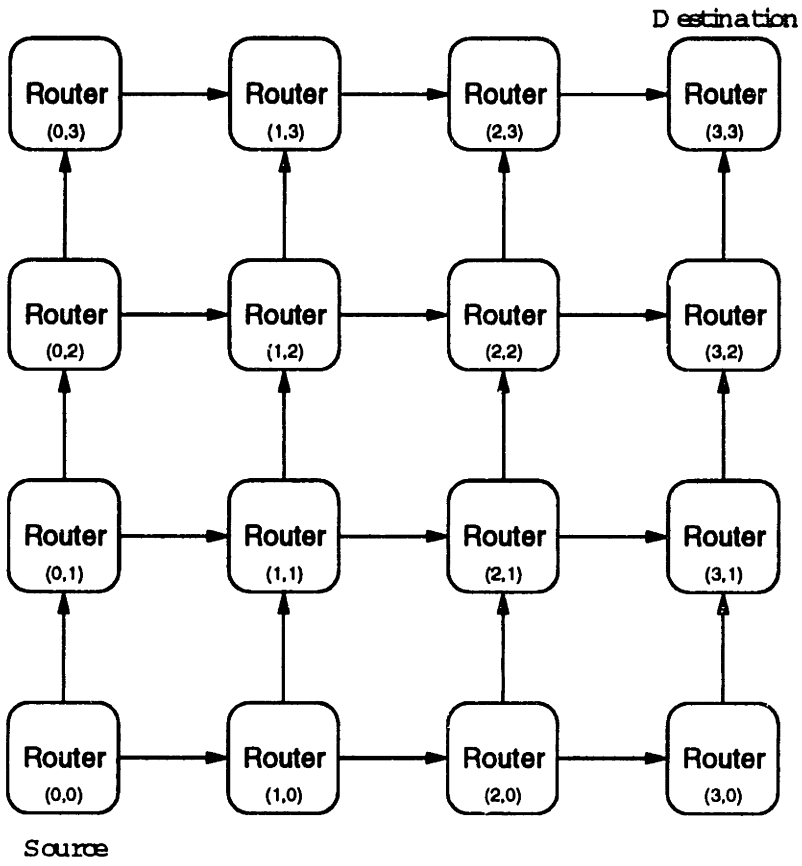


Figure 3-2: Minimally adaptive routing. Suppose router (0,0) is sending to router (3,3). Under minimally adaptive routing, the message could travel over any of the channels shown. Since the message has to travel over a total of 6 channels, 3 in X, 3 in Y, there are $\binom{6}{3} = 20$ different paths possible.

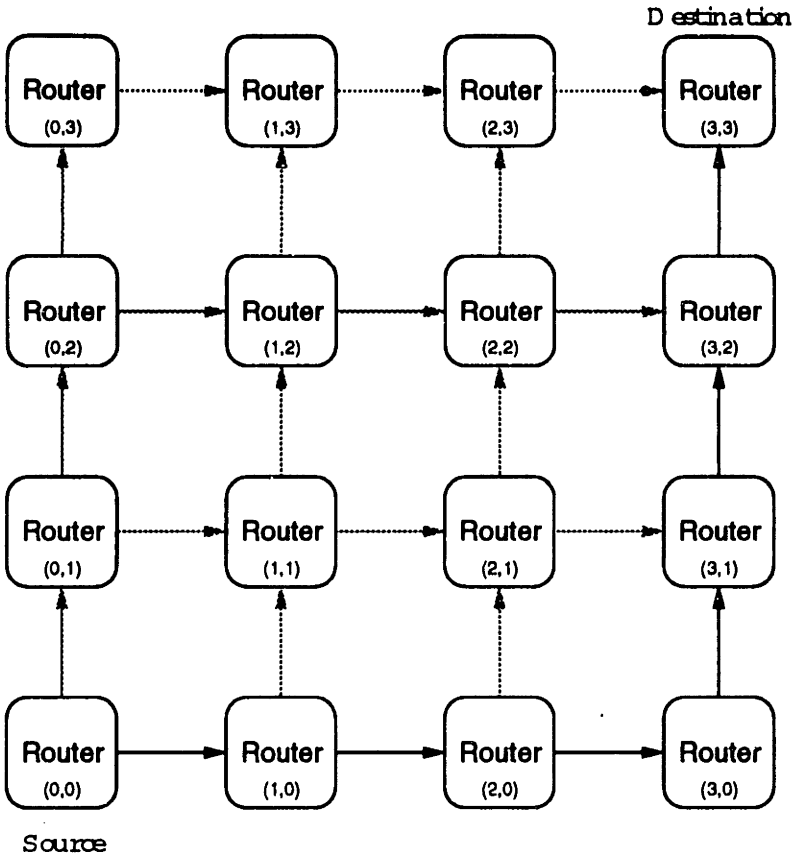


Figure 3-3: Dimension-ordered routing. Suppose router (0,0) is sending to router (3,3). Under dimension-ordered routing, the message travels over the path shown in black.

3.1.2 Dimension-Ordered Routing

By themselves, minimally adaptive networks are not deadlock free. A second deadlock-free network is added to allow messages to "drain" when they are blocked in the minimally adaptive network. This network uses dimension-ordering, X then Y, as its routing function. Note that its bias is reversed from the minimally adaptive router to help avoid congested spots.

As was noted earlier, dimension-ordered networks can deadlock on request-response protocols. The dimension-ordered network is therefore split into two distinct dimension-ordered networks. A message traveling in the minimally adaptive network is assigned to one of the two dimension-ordered networks based on its priority level.

The two priority levels can used for multiple purposes. In a pure message-passing paradigm, they can be used to separate system traffic from regular traffic. In a request-response paradigm, such as reading from shared memory, the two priority levels are used to break the resulting resource dependency cycle. Requests travel at priority 0, responses travel at priority 1.

When a message is first injected into the network, it uses the minimally adaptive

virtual network, irrespective of the priority of the message. The priority is only used when the message is blocked.

Priority does not give a strict ordering to the messages. If it did, it would be possible to construct traffic patterns with starvation. Rather, priority is treated as a hint when arbiting for a given physical channel. When lower-priority messages have waited too long for a physical channel, their priority level is momentarily elevated.

3.1.3 Fault-Handling Routing

X-then-Y dimension ordering assigns X to dimension 1, Y to dimension 0.

The Reliable Router is designed to detect and recover from link failures between any two routers. The fault model is single-wire failures and is detected using parity checking. The faults are presumed to be non-transient so the link is marked as faulty as soon as an error is detected. Further, the fault model assumes that there is at most one fault in the network at any point in time.

Fault handling routing is invoked when:

- the message is unable to make progress on an adaptive channel, either due to congestion or fault, and
- the message can never make progress on a dimension-ordered channel due to a fault.

The fault handling algorithm is:

1. Attempt a route in a productive direction using a fault-handling virtual channel. This step fails only if a productive direction does not exist. i.e. the message needs to travel in only one dimension to reach the destination.
2. If the unmatched dimension is non-zero (X), route in a non-productive direction along dimension 0 using a fault-handling channel. In other words, take a side-step Y. Resume regular routing.
3. If the unmatched dimension is zero (Y), route on a fault-handling channel in dimension 1 (side-step in X). Route in productive dimension 0 (Y) fault-handling channels until the dimension 0 (Y) address is reached. Route in a dimension 1 (X) fault-handling channel until the dimension 1 (X) address is reached.

The two fault handling cases complicate the router somewhat. The first fault handling step could be undone by a subsequent adaptive routing step. This is prevented by precluding backtracking in the adaptive routing.

The second case seems to imply that state is carried along the message. This state is carried by keeping the message in fault-handling channels until it is finally delivered. In the second case, the side-step occurs in X. Any message arriving on an X fault-handling channel must exit on a fault handling channel. The message then travels a step in Y. The router notices that it is one hop away in X, so it continues to use Y fault-handling channels until the proper Y address is reached. Finally, it completes the route in X.

Signal	*Direction*	*Description*
`Data`[15:0]	Bidir	Data wires for transporting user data
`DataParity`[1:0]	Bidir	Intended for byte-wise parity on user data
`Control`[4:0]	Bidir	All the control information
`Parity`	Bidir	Parity over the Data, DataParity, and Control fields.
`Phase`	Bidir	Used to extract time-division multiplexed data and control.
`TxClk+`,`TxClk−`	Output	A differential clock used for retiming the data
`RxClk+`,`RxClk−`	Input	A differential clock used for retiming the data

Figure 3-4: Router-to-router signals.

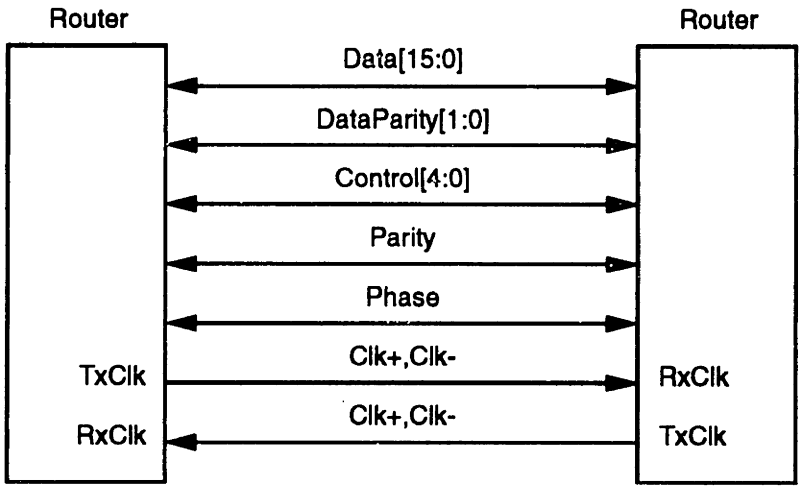


Figure 3-5: Physical interconnect between routers.

3.2 Network Construction

The Reliable Router can be used to construct a two-dimensional mesh network. Each router has four *ports*, which are the physical interfaces to other routers. These ports are known as $X+, X-, Y+, Y-$ and correspond the directions in which a message travels to reach a neighboring router. The physical interface is comprised of 26 wires (see Table 3-4). The electrical signalling is accomplished using simultaneous bidirectional signalling for all wires except the clocks. The clocks use the same signalling levels and are both differential and unidirectional. Additional details of the electrical signalling are provided in Chapter 5.

The clocks are sent along with the data to allow the network to be plesiochronously timed. Each router has its own clock reference which is approximately the same frequency as that used by the other routers. This converts the clock from a single global resource into a plurality of local resources. A network built out of Reliable

	Bit Field					
	22	21	20:18	17	16	15:0
Subframe 0	pe	USR0	VCI	DP1	DP0	Data[15:0]
Subframe 1	Copied Kind		Copied VCI	DP3	DP2	Data[31:16]
Subframe 2	U/D	USR1	Kind	DP5	DP4	Data[47:32]
Subframe 3	Freed			DP7	DP6	Data[63:48]

Table 3.1: Frame format.

Routers does not have a common clock, thus there is no need for a clock distribution network. Further information on plesiochronous retiming is found in Chapter 6.

The actual format of the information passed between routers is shown in Table 3.1. These field are not visible to the user. A detailed description will be postponed until Section 7.5.1.

3.2.1 Messages

Messages, as the fundamental object transported, are visible to the user. From the user's perspective, a message consists of some amount of data and an address to deliver the data to. Messages can vary in size and ideally would not have any length restrictions.

The Reliable Router efficiently copes with variable size messages by using *wormhole routing*. In wormhole routing, a message is decomposed into smaller pieces. Rather than waiting for a complete message to arrive, a router is now able to forward these smaller pieces as they arrive. As flow control is now implemented at the level of these pieces, they are known as *flow control digits* or *flits*.

A user of the Reliable Router must be aware of the flit-level partitioning of the message. The Reliable Router implements a link-level fault-tolerant protocol known as the Unique Token Protocol. This protocol keeps two copies of a message in the network at all times. The Reliable Router implementation keeps two copies of each message flit in the network at all times. However, if a network fault occurs, messages can be cut in two along a flit boundary. Both pieces will be delivered to the destination but it is up to the user to paste the message back together. The unique token protocol is more fully described in Chapter 4.

The protocol requires that messages be partitioned into a head flit, some number of data flits, a tail flit, and a token flit. The head flit contain the destination address, the sender's address, and a sender-relative sequence number. All of these fields must be provided by the user.

Both data and tail flits are handled in an identical fashion by the router. The data flits hold the user's data payload. The tail flit should contain the length of message and is given a separate flit type for the convenience of the user. The length could be kept in the head flit if desired, however the full message length may be not be known in advance in all cases.

Signal	*Direction*	*Description*
pr_clk	Input	Processor Clock
pr_ctl[4:0]	Input	Control inputs
pr_data[31:0]	Input	Data input
pr_par[3:0]	Input	Data parity input
pr_phase	Input	Phase, used to identify flit boundarys
cts_to_proc[4:0]	Output	Clear-to-send output
free_out[4:0]	Output	Virtual channel is free

Table 3.2: Processor input port signals.

The token flit cannot contain any meaningful user data, as it will be lost in the event of network failure. Its sole purpose is to assist the destination with duplicate message elimination.

3.2.2 Processor Port

Processors inject messages into the network and extract messages out of the network via the router's processor port. The processor port is full-duplex. Messages can be sent at the same time as messages are received. Physically, there is a processor input port and a processor output port. Each has a 32-bit wide data bus, byte-wise parity signals, and several control signals. Both input and output ports are plesiochronously timed. However, it is assumed that processor/router interface will most often be operated mesochronously.

The processor port interface is fairly simple. Each data wire can operate at 100Mbits/sec. To ease processor interface timing constraints, the plesiochronous input section adjusts for the mesochronous clock skew automatically. The processor output section is also plesiochronously timed.

There is a complication, which is that the flow control information goes opposite the data direction. These flow control signals are treated as asynchronous signals.

Sending a Message

The interface signals for the processor to send a message are shown in Table 3.2. The processor provides a clock to the router to retime the control, data, parity, and phase inputs. All processor inputs to the router are sampled at the positive edge of the clock.

All flits are 64 bits. Since the data path width is 32 bits, the flit is clocked in on two successive clock edges. Figure 3-6 shows the relationship between phase, control, and data. The router determines that it is has been presented with a flit by inspecting the virtual channel field (control lines when phase=1). If the field is 0, there is no flit. Otherwise, exactly one of the lines should be a 1 to indicate the virtual channel number.

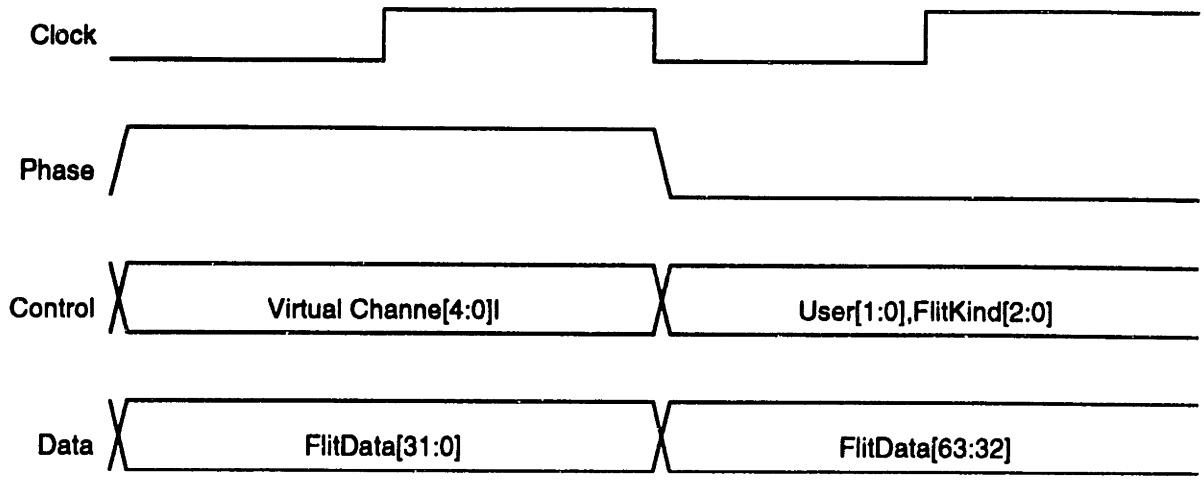


Figure 3-6: Processor input signals.

There are five virtual channels and all are equivalent in function. It is the responsibility of the processor to know which virtual channels are busy and which are free. Since the processor knows the state of all the channels, it can begin sending on any free virtual channel. Once the router receives a flit, the channel should be considered busy. The processor can then supply the remainder of the message, finally ending the message with a token. At this point, the processor is done, but the virtual channel may not be free. Flits may still be in the input buffer within the router, waiting to be forwarded. When the router has finally forwarded the token, it sends a pulse on the virtual channel's `free_out` signal, indicating that the virtual channel is now free.

Very long messages could overflow the router's input buffer if flow control were not provided. Flow control is handled using a simple clear-to-send for each virtual channel. The router has 16-flit deep input FIFOs per virtual channel. Clear-to-send is negated whenever the FIFO depth exceeds 8 elements.

Receiving a Message

Receiving a message is straightforward. Table 3-7 gives the interface signals used. When the router has a flit for the processor, it asserts `pr_valid` for one clock period (while `pr_phase` is high). The first half of the flit is clocked out during that clock period, the second half during the following clock period. While phase is high, the control field gives the encoded virtual channel number. While phase is low, the control gives the flit kind.

3.3 Summary

The Reliable Router offers several distinctive features. It provides fault-tolerance, adaptive routing, virtual channels, deep input buffers, high bandwidth and low-latency. Three of the key technologies, the Unique Token Protocol, simultaneous

Signal	*Direction*	*Description*
`cts_from_proc`	Input	Clear-to-send
`pr_usr`	Output	extra user data bits
`pr_ctlout[2:0]`	Output	Virtual channel, flit kind
`pr_parout[3:0]`	Output	Byte-wise parity
`pr_dataout[31:0]`	Output	Data
`pr_valid`	Output	Valid flit
`pr_phase_out`	Output	Used in the time-multiplexing of the flit.

Figure 3-7: Processor output port signals.

bidirectional signalling, and plesiochronous data retiming are discussed in the following chapters.

Chapter 4

The Unique Token Protocol

The concept of the Unique Token Protocol was previously introduced in [11]. The explanation focused on a particular implementation and was thus somewhat clouded by that implementation. In this chapter, a different approach is taken. The general goals of providing fault tolerance in the network are stated. The protocol is first described in terms of packets, then proofs of correctness of the packet-oriented protocol are given. Once the underlying protocol is shown to be correct, the protocol is extended to cover wormhole routing.

4.1 Goals of Network-Level Fault Tolerance

A general critique of end-to-end fault-tolerant protocols produced the following observations:

- The sender must keep a copy of the packet until it knew that the destination had a copy. This added both bookkeeping and storage overheads.
- An explicit acknowledgment must be sent, which used network bandwidth and careful handling to ensure freedom from deadlock.
- Bookkeeping is needed at the destination to perform duplicate elimination.
- A time-out is required in order to detect when things didn't get through the network. For example, if the sender did not receive an acknowledgment within a certain amount of time, it needed to resend the packet.

The Unique Token Protocol was developed to address these issues. Using the Unique Token Protocol, the sender does not retain a copy of the packet as the network keeps at least two copies at all times. No acknowledgement is sent, the protocol guarantees that at least one copy will be delivered. Duplicate elimination is greatly reduced through the use of a token. Network-level timeouts are no longer required.

4.2 Overview of the Unique Token Protocol

The Unique Token Protocol consists of two distinct components: a method of keeping at least two copies of a packet in the network at all times, and a method of notifying the destination after it has gotten the packet that no additional copies of the packet exist in the network. The first component drops the need for the sender to keep a copy and the need for acknowledgments. The second component helps with the amount of bookkeeping at the destination as well as reducing the need for network-level timing parameters.

The two components are truly distinct and may be used independently. The flow-control protocol which keeps two copies in the network is useful only if one expects node failures. Otherwise, a link-level acknowledgment would suffice. This flow-control algorithm can be used without the token-passing component, an ordinary duplicate elimination approach would work (but not as efficiently).

Similarly, the token-passing passing protocol could be used without keeping two copies in the network. This would result in a system which was tolerant of link failures, yet did not need heavy-weight duplication elimination at the destination.

4.3 Keeping Multiple Copies in the Network

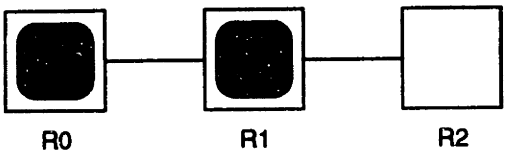
The flow-control protocol used in the Reliable Router is quite simple: copy forward, free backward. When a node sends a copy of the packet, the node does not immediately free the packet's storage. The node instead tells the node it had gotten the packet from to free its storage. Figure 4-1 shows the complete protocol.

4.3.1 Correctness of the Flow-Control Algorithm

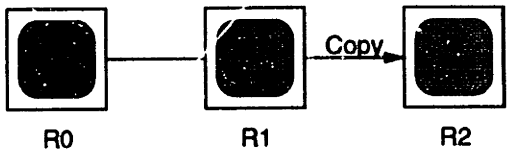
In order for the Unique Token Protocol to operate correctly, it must be shown that at least two copies of a packet are kept in the network at all times using this flow-control algorithm. During the following proof, it will be assumed that the routing algorithm is non-backtracking and will make forward progress.

Theorem 4.1 *The Unique Token Protocol flow-control algorithm maintains at least two copies of a packet in the network on different nodes at all times.*

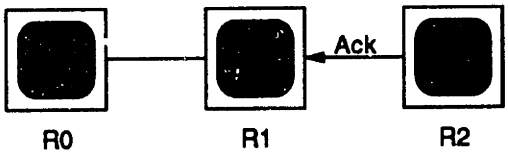
The path that the packet travels from source to destination through the network is called a route. Since the routing algorithm is non-backtracking, each node visited along the route appears in the route exactly once. Thus the route forms a complete order of the nodes in the route, from source to destination. These nodes can then be assigned distinct integral values, with the lowest at the source and the highest at the destination such that the complete ordering of the values matches the complete ordering of the route. Further, the desired trivial assignment $0, 1, 2, 3 \ldots m$ also satisfies the ordering criteria. At any point in time, some of the nodes along the route may have copies of the packet, others will not.



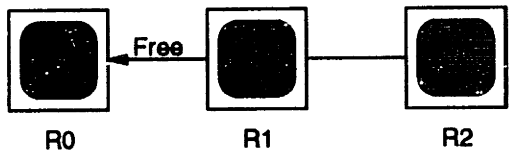
(a) Initial State. Two copies of the packet are in the network.



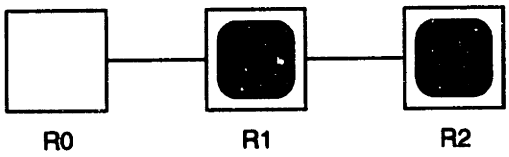
(b) Packet Forwarded. The packet is copied forward, resulting in three copies in the network.



(c) Acknowledgement. R2 acknowledges to R1 that R2 successfully got the packet.



(d) *Freed* flow control signal. R1 notifies R0 that the packet has been copied to R2.



(e) Final state. R0 frees the packet storage.

Figure 4-1: Simplified Packet-oriented Flow Control for the Unique Token Protocol.

A way of expressing the state of a route as the packet travels along it is now needed. This is done by *coloring* the nodes. A node which has never had a copy of the packet is colored white, a node which has a copy of the packet is colored black, and a node which had a copy but no longer does is colored gray.

Observation 4.1 *If node N_i is colored white and $i < m$, then node N_{i+1} is colored white. This follows from the definition of the route.*

Observation 4.2 *If node N_i is colored non-white and $i > 1$, then node N_{i-1} is colored non-white. This too follows from the definition of the route.*

Lemma 4.1 *If node N_i is colored white, then $\forall j, i \leq j \leq m, N_j$ is colored white.*

Proof By induction using Observation 4.1. ■

Lemma 4.2 *If node N_i is colored non-white, then $\forall j, 0 \leq j \leq i, N_j$ is colored non-white.*

Proof By induction using Observation 4.2. ■

Lemma 4.3 *Let N_i be the highest numbered node which is non-white, if such an N_i exists. If N_i exists and $i < m$, node N_i is colored black.*

Proof A non-white node is either black or gray. The proof that it not gray is by contradiction. Suppose that it is gray. Node N_i then had a copy of the packet but erased it. By examination of the flow control algorithm, this could occur only if node N_{i+1} had sent node N_i a freed message. This implies that node N_{i+1} has or had a copy of the packet. Therefore, node N_{i+1} must be non-white. But, node N_i is the highest numbered non-white node, which contradicts. Therefore, node N_i must be black. ■

Lemma 4.4 *Let N_i be the highest numbered node which is colored black, if such an N_i exists. If N_i exists and $1 < i < m$, then node N_{i-1} is colored black. (Figure 4-2).*

Proof By Lemma 4.2, N_{i-1} must be non-white, i.e gray or black. Suppose that it is gray. This would imply that it had gotten a freed message from node N_i allowing it to erase its copy. But, N_i hasn't yet copied the packet forward to N_{i+1}, as N_{i+1} is white. This contradicts, so therefore N_{i-1} is black. ■

The correctness of the flow-control protocol is almost done. From Lemmas 4.1,4.2, and 4.4, one can see that good route colorings consist of 0 or more non-white nodes, followed by 2 black nodes, followed by 1 or more white nodes.

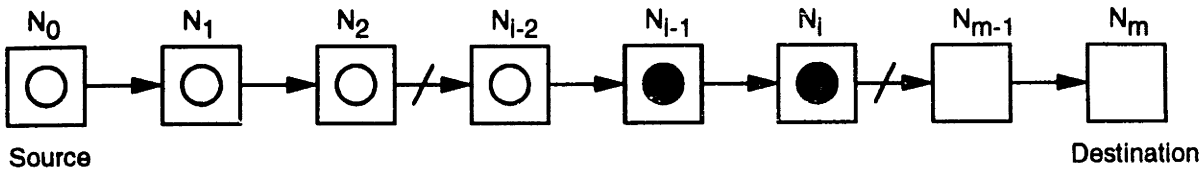


Figure 4-2: The two foremost nodes in a route always have a copy of the packet. Empty squares indicate nodes which have not gotten a copy, filled circles indicate a copy of the packet, empty circles indicate that a copy may or may not be present.

Define $m-1$ equivalance classes of route colorings, $E_1, E_2, E_3 \ldots E_{m-1}$, where equivalance class E_i contains all possible route colorings having node N_i as the highest numbered black node. Observe the class E_1 has a single coloring, node N_0 and N_1 black, node $N_2 \ldots N_m$ white. This coloring corresponds to the initial condition.

To show the correctness of the flow control protocol, one needs to show that all possible colorings produced by the protocol from the initial coloring belong to one of the equivalence classes.

Consider a route coloring belonging to equivalence class E_j. Since it is in the class, nodes N_0 through N_{j-2} are non-white, nodes N_{j-1} and N_j are black, and $N_{j+1} \ldots N_m$ are white. Under the protocol, only a single node changes color at a time, so consider how a node might change color.

Observe that the node color changes are ordered: white to black to gray. Once non-white, a node stays non-white. Once gray, a node stays gray. This implies that the initial sequence of non-white nodes $N_0 \ldots N_{j-2}$ is preserved under all orderings of applications of the protocol. That leaves nodes $N_{j-1} \ldots N_m$. Under the protocol, the only possible color change is for node N_{j+1} to change from white to black. This is acceptable as long as $j \neq m-1$. Thus, for a given route coloring, any valid protocol step produces a coloring in one of the equivalence classes.

By induction starting from the base class E_1, one sees that the set

$$\bigcup_{i=1}^{m} E_i$$

is closed under all possible protocols steps. Therefore, each coloring has at least two black nodes, i.e. two copies of the packet are kept in the network at all times. ■

Theorem 4.1 had two caveats. The first was that it had to begin in a state where there were already two copies in the correct positions. This condition is met in practice by the originator of the message being N_0. It highlights the vunerablity of the network to the loss of the message originator before it has injected the message.

The second caveat had to do with the final recipient of the message, node N_m. Once N_m had a copy, it was no longer possible to guarantee that the network had additional copies. Indeed, one wants the network to dispose of its copies.

Backtracking Routing

In a backtracking network, some of the nodes can be revisited. As long as a node cannot send the packet to itself, any two adjacent nodes in the route will be physically different nodes. In the proof of lemma 4.4, it was shown that the two foremost nodes will have copies of the packet. Since these are known to be physically different nodes, the flow-control algorithm will keep two copies on different nodes at all times.

4.3.2 The Unique Token Protocol Flow-Control Protocol Using Permits

To implement the Unique Token Protocol, at least two copies of the packet must be kept in the network at all times. In the reliable router, this is done as part of the flow control algorithm. For most ordinary networks, flow control is implemented using a clear-to-send signal which indicates that there is storage for another packet.

Higher performance flow-control systems are typically *permit-based.* The sender of a packet knows how much packet storage the receiver has. When the packet is sent, the transmitter decrements the available storage count (its supply of "permits"). When the number of permits reaches zero, the transmitter is blocked and will not send any more packets.

At some point, the receiver of the packet decides that the storage occupied by the packet can be reclaimed. It notifies the transmitter that it has freed up the storage. The transmitter then increments its permit count and transmission continues.

In these permit-based systems, only one copy of a packet is guaranteed to be kept in the network as the receiver of a packet will release the storage as soon as the packet is successfully forwarded. In the reliable router, the storage is released when the packet has been forwarded to *two* other routers. Figure 4-3 shows how the entire process is implemented.

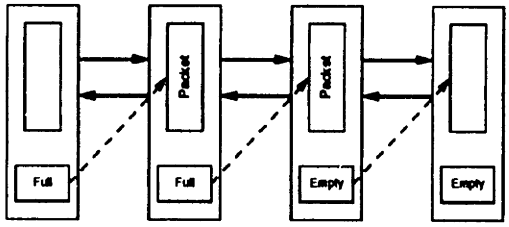
4.4 The Token

The unique token is a mechanism for reducing the processing load at the destination by explicitly telling the destination that the packet arrived exactly once. Before explaining the token portion of the protocol, a lemma is needed.

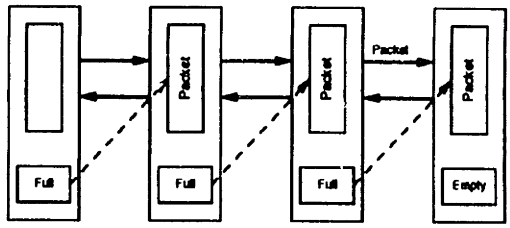
Lemma 4.5 *In order for a destination node to get duplicates of a packet, some node in the network transmitted more copies of the packet than it received. It is presumed that no node can simply make up a copy of the packet it hasn't received.*

Proof The proof is by counting the total transmits and receives for a packet. Each packet receive must have a corresponding transmit. The destination node had at least two receives in order for it to see duplicates. The source node should have transmitted only once. This leaves an imbalance of more receives than transmits.

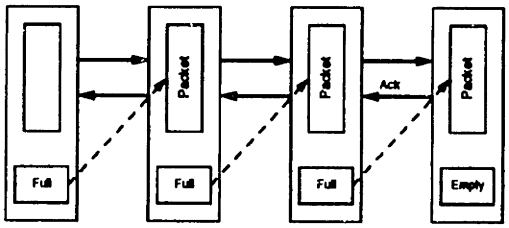
Suppose for each of the remaining nodes the number of transmits is less than or equal to the number of receives. Then the total transmits is strictly less than receives,



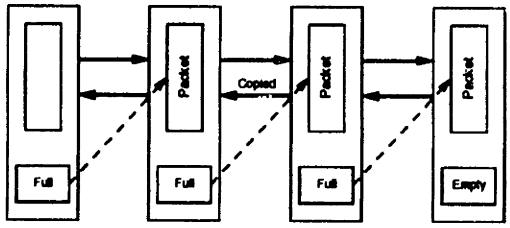
(a) Initial State. Two copies of the packet are in the network.



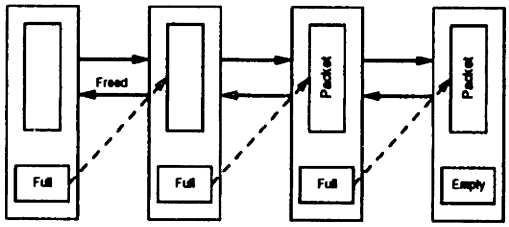
(b) Packet Forwarded. The packet is copied forward, resulting in three copies in the network.



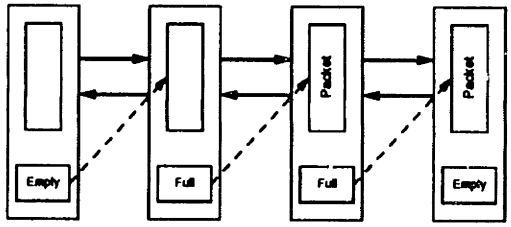
(c) Acknowledgement. Node 4 acknowledges to node 3 that node 4 successfully got the packet.



(d) *Copied* flow control signal. Node 3 notifies node 2 that the packet has been copied to node 4.



(e) *Freed* flow control signal. Node 2 now knows that two copies of the packet exist in the network, one at node 3, the other at node 4. It now releases the storage and tells node 1.



(f) Final state. Node 1 adjusts its flow control counter so that it can transmit another packet to node 2.

Figure 4-3: Packet-oriented Flow Control for the Unique Token Protocol.

which violates the correspondance rule. Therefore, some node must have transmitted more times than it received. ∎

The concept behind the unique token protocol is that it is possible to verify that for each of the nodes along a route, that that node transmitted the packet no more than it was received. This is especially trivial when the router is non-backtracking. The node should receive the packet at most once, so all that is needed is a check to see if the packet was transmitted more than once. The token-passing component of the protocol works as follows:

The sender of the packet generates a unique token.
Until the token reaches the destination or is lost, do:
    Wait for the node which has the token to erase all copies of the packet.
    Has the node which has the token transmitted more than it received?
        Yes. Discard the unique token.
        No. Forward the token to the next node on the route.
            If the next node on the route is unreachable, discard the token.

4.4.1 Correctness of the Token Passing Algorithm

The general proof of the token passing passing algorithm is not yet complete. However, a proof when the routing algorithm is non-backtracking is possible. For some network topologies, such as toruses, it should be possible to construct routing algorithms which are non-backtracking in the presence of faults. In 2D meshes, this is not possible in all cases. However, the proof still holds in the cases when the router does not have to backtrack to get around a particular fault.

Lemma 4.6 ***(The unique token constructs an ordered list of nodes, Figure 4-4)*** *Let $N_0, N_1, \ldots N_j$ be the nodes over which a unique token has traveled. The route from N_0 to N_j is then a complete order.*

Proof By construction of the trivial route labeling. Set i=0. Make the source node the current node and label it N_0. Label the node the current node sent the token to N_{i+1}. Since it sent the token to one node, there will be only one node labeled N_{i+1}. Make this the new current node. Repeat until all nodes have been labeled. Since there is no backtracking, there is no conflict of label assignments. This labeling forms a complete ordering. ∎

Lemma 4.7 *In a non-backtracking network, if N_j has a unique token, then nodes $N_0, N_1, \ldots N_{j-1}$ do not have copies of the packet.*

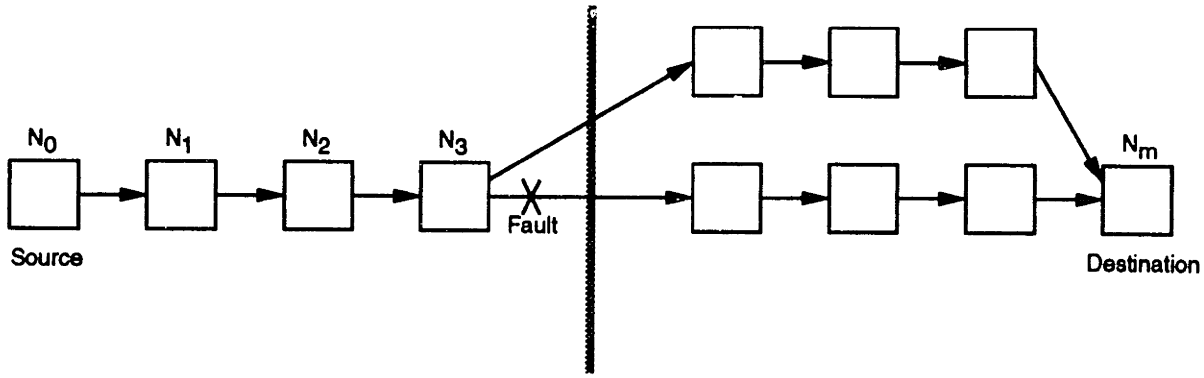


Figure 4-4: The lack of a complete node order. Beginning at the source and continuing up to the point of the fault, it is possible to describe the exact order the unique token traversed the network. After the fault, the graph branches. Between the branches, no temporal ordering relationship exists.

Proof From lemma 4.6, if node N_j has the token, then the trivial labeling exists, it corresponds to the order the token traversed the nodes, and is a complete order. The complete order allows the induction to be well-founded.

Base step: Node N_j has a copy of the token.

Induction hypothesis: If node N_i has a unique token, then N_{i-1} does not have a copy of the packet and N_{i-1} had the token. By definition of the route, the only way for N_i to obtain the token is from N_{i-1}. Therefore, node N_{i-1} must have had the token. By inspection of the protocol, node N_{i-1} must have erased its copy of the packet.

Inductive Step: Since N_j has the unique token, N_{j-1} does not a copy of the packet and it also had the token. Since it had the token, node N_{j-2} also does not have copy of the packet and had the token. By induction, nodes $N_0, N_1, \ldots N_{j-1}$ do not have copies of the packet. ■

Theorem 4.2 *In a non-backtracking network, if the destination has a unique token for a packet, then it will receive no more than one copy of the packet.*

Proof In lemma 4.5, it was established that some node in the system had to transmit more times than it received in order for the destination to receive multiple copies. Since the token passing algorithm verifies the (transmits $\leq$ receives) property for all nodes which had a copy of the packet, the destination received at most one copy by the time of the unique token arrival.

From lemma 4.7, if the destination has the token, no other nodes in the network have copies of the packet, so the destination cannot receive any more copies after the arrival of the token. ■

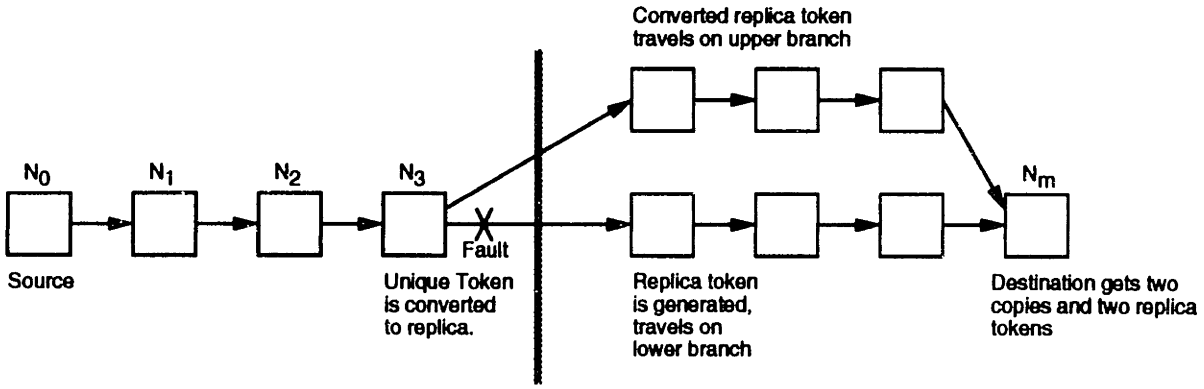


Figure 4-5: The conversion of the unique token to a replica, along with the generation of a token.

4.4.2 Token-Passing Pragmatics

In the previous section, the unique token was following behind the packet, ensuring that no extra copies of the packet were generated. This implies that each of the nodes was keeping some state information around after the packet has been transported, just so the unique token can check it. However, the unique token was simply discarded when it was found that some node transmitted more packets than it received, so the state information was never freed.

This can easily be fixed. Logically, the unique token is freeing up the state information as it is being passed from node to node. If it becomes necessary to discard the unique token, only the uniqueness property of the token is discarded, not the entire token. The non-unique token can then be sent along to all the nodes, freeing up the state information. This non-unique token is called a *replica* token, as its arrival at the destination indicates that multiple copies or *replicas* of the packet could be delivered.

There is one other complication. When a failure occurs, some of the nodes will lose contact with the source node. These nodes will be unable to get a token which originates at the source. They will not free up the state information. To handle this, a node which is waiting for a token and is unable to receive one because of a failure must generate its own replica token. Figure 4-5 shows how replica tokens are both converted and generated.

Theorem 4.3 ***(Exclusivity of token types)*** *For a given packet, either there will be one copy of the packet delivered with a unique token, or one or more copies of the packet delivered with replica tokens.*

Correctness The argument is subtle, as both types of tokens can exist at the same time. In particular, a replica token can be generated before the source node injects the unique token!

The argument looks at the creation of replica tokens, as they have to be created in order for the destination to see both kinds of tokens. One way they are created is

when the unique token comes to a node which has transmitted twice. In this case, the unique token is converted to a replica token, so locally the exclusivity property is maintained.

The other way in which replica tokens are created is when a node becomes severed from the source before it has gotten a token. In a single-point-of-failure model, the route leading up to the point of severage is a chain originating at the source. Therefore, the unique token must encounter the severage point. One of two cases must be true. Either that last node had a copy of the packet after the severage occurred, or it didn't.

If it did not have a copy after the severage happened, that node would not have transmitted twice. In essence, that node has become cut off from the destination, so the unique token will be discarded at that node and the exclusivity is maintained.

If it did have a copy, it would have been transmitted to another node. When the unique token arrives at the node, it detects the transmitted twice case and the uniqueness is discarded. ■

4.5 The Unique Token Protocol Using Wormhole Routing

The reliable router uses wormhole routing, not packet switching. As such, the bookkeeping becomes a bit more complicated. Packets can be sliced in two by a network failure, so mechanisms must be provided to allow the packet to be put back together.

4.5.1 Packet Format and Reconstruction

A packet consists of a head flit, some number of data flits, a tail flit, followed by a token. Head, data and tail flits come in flavors original and restarted. The token is either unique or replica.

The head flit contains the destination node address, the source node address, and a sequence number. The data flits contain user data. The tail flit contains the length of the packet.

The flit flavors, along with the length field in the tail, allow the message to be put back together after a fault. A head flit of type original is always followed by the data flits from the start of a message. A head flit of type restarted indicates that the data flits may have come from the middle of message. A head flit of type restarted will always eventually be followed by a tail flit.

The reconstruction is then simple. Allocate a buffer of the size found in the tail flit. Fill in the starting flits from the message tagged as original. Fill in the ending flits from the message marked as restarted. The message is now reconstructed.

4.5.2 Flow Control

In the router's input controller, explicit storage for head and token flits is provided. If a failure occurs, the head flit must be present to allow the remainder of the message to be rerouted. The storage is explicit for the token, as it requires special handling.

Logically, the arrival of a head flit at a node allocates the storage needed to wormhole route the remainder of the message. That storage and state is called a virtual channel, and the allocation persists until the node forwards the token. In the reliable router, the allocation state for a virtual channel is kept in the previous node's output controller. The sending of the head flit causes the virtual channel to be allocated. However, in order to deallocate the virtual channel, the node forwarding the token notifies the previous node that the virtual channel should be freed.

Data flits are kept in a ring buffer. A copy of each data flit must be on at least two nodes at all times. The flow control algorithm described in section 4.3 is used to ensure this. The actual implementation in the router is complicated by the permit-based flow control. Each time a data flit is copied forward which has not previously been copied, a copied message is sent backward. The arrival of that copied message "frees" the storage in that node, and a corresponding freed message is send to the node before that. This creates another "permit" for the first node.

The qualifier of "not previously copied" is required to keep the counts correct after a fault occurs. Otherwise, the upstream node would have counted more "freed" messages than data flits sent.

4.5.3 Restarting a Message After a Fault

When a fault occurs, the message must be rerouted to a different node. That rerouting causes the type of flits passing through the virtual channel to be changed from flavor original to flavor restarted, and the token to change to type replica. The head flit and all data flits are tagged as needing transmission. The transmission of the message then begins again.

4.5.4 Completing a Message After a Fault

A node may receive part or all of a message and then experience a fault which precludes completion of the message. The recovery procedure is very simple. The node sends as much of the message as it received. If it received the token, it can be forwarded "as is". If it has no token, it generates one of type replica and sends that.

4.6 Summary

This chapter presented the Unique Token Protocol in terms of a flow-control component and a token-passing component. The flow control component was shown to have the property that at least two copies of the packet are kept in the network at all times. The token passing component was shown to properly indicate that the

arrival of a unique token meant that the packet was delivered once, subject to the non-backtracking restriction.

The protocol was then presented in the context of a wormhole-routed implementation. A network implementing the Unique Token Procotol has some costs over a non-reliable network. Chief among these is the storage requirement for two copies of every flit. Additional costs are the complexity of the flow control management and the needs for adequate firewalls.

The protocol does have its benefits. It eliminates the need for source buffering and duplicate elimination. It uses very little extra network bandwidth. Perhaps most importantly, the error recovery takes place around the failure point. This minimizes the amount of time required between the detection of the failure and the invocation of the recovery procedure.

Chapter 5

Simultaneous Bidirectional Signalling

The Reliable Router is designed for high-performance communications. The performance of the router is measured in terms of its bandwidth and its latency. Within the router, one the components critical to obtaining both high bandwidth and low latency is the I/O pad design. The router uses *Simultaneous Bidirectional Signalling* to achieve its communications performance goals.

Several chips implementing bidirectional signalling have been built and tested over the past 5 years. One of the more astonishing results (at least to the experimenters) was that the later test chips were successfully recovering signals out of some truly awful-looking interchip waveforms. Was this a fluke, or was the design really that noise tolerant? What are the real noise margins? To answer those questions, this chapter begins with a general discussion of signalling and noise in the simple unidirectional case. It explores how noise is created in one part of the system and is then coupled into other parts of the system. Once a good understanding of noise is achieved, components of a signalling system are designed and analyzed. The operational characteristics of the entire unidirectional system are determined.

Once unidirectional behavior is fully understood, those components are incorporated into a bidirectional system. Other operational characterizations, such as the common-mode rejection of the receiver, are performed. The additional noise susceptibility of bidirectional signalling over unidirectional signalling is clarified.

5.1 Fundamentals of Signalling

Electrical signals are continuous waveforms. The digital logic abstraction maps the waveform values into discrete 1's and 0's. This mapping of continuous waveforms onto discrete values implies that there are electrical values which are mapped into one logic value which are extremely near other electrical values which map into the other logic value. As an example, consider an ideal CMOS inverter. Electrical voltage values between 0v and VDD/2 result in in a logic 1 output, values greater than VDD/2 and less than or equal to VDD result in a logic 0 output.

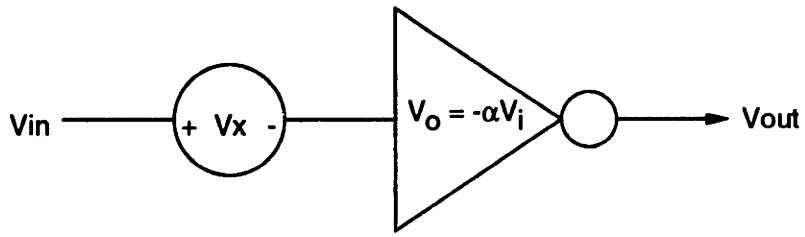


Figure 5-1: Noise and Gain in an Inverter Circuit.

Two real-world effects mar this abstraction. First, the ideal inverter transfer function cannot be built. The inverter itself produces a continuous output, so it is possible to provide an input signal which results in the output being in the middle of its range. An ideal inverter is a perfect "voltage arbiter", the real inverter has a fixed point which is neither a logic 1 nor a logic 0.

The second real world effect is noise. All electrical waveforms have some form of noise which is added to the signalling waveform. The signalling waveform must be kept sufficiently far from the logic decision point such that the probability of the noise causing the signal to be misinterpreted is acceptably low.

Figure 5-1 shows a simple inverter circuit. There is an input signal V_{IN}, a noise source V_X, and a voltage inverter with gain α whose output is V_{OUT}. If this to form the basis of a composable logic family, V_{OUT} is the next gate's V_{IN}, so it is required that:

$$\alpha\left(|V_{IN}| - |V_X|\right) \geq |V_{IN}| \tag{5.1}$$

Treating all quantities as positive and rewriting:

$$V_{IN} > \frac{\alpha}{\alpha - 1} V_X \tag{5.2}$$

The required signal level is a function of the amount of noise and a function of the gain of the circuits. Very high gain circuits can operate at signal levels near the noise level, while low gain circuits need much higher margins. The next section will explore how noise enters into the interchip signalling system.

5.2 Noise

If one had ideal current sources, power supplies, transmission lines, termination resistors, and no parasitics, some truly remarkable bandwidths could be achieved and there would be no need for this chapter. However, the real world deviates from the ideal in many ways and the art of high-speed I/O design is all about knowing where the deviations are and how to either avoid them or compensate for them. The following effects are modeled as noise sources:

- Classical noise sources such as thermal noise and shot noise

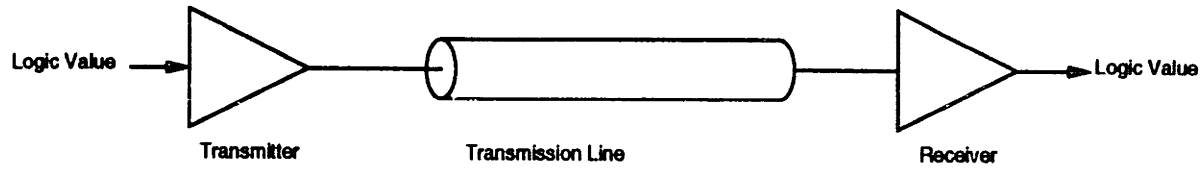


Figure 5-2: Ideal Unidirectional Communication.

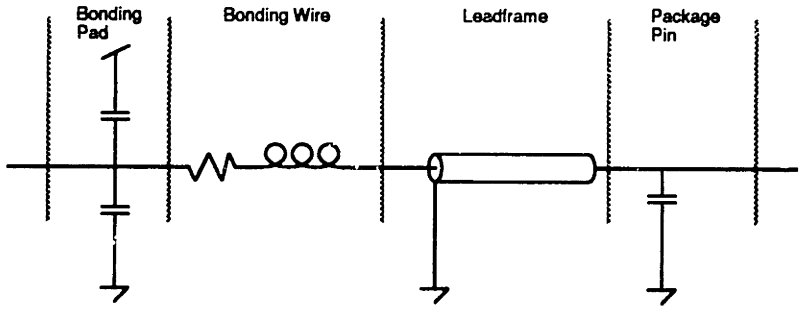


Figure 5-3: Package Model.

- Crosstalk
- Non-linear circuit elements
- Non-ideal voltage and current sources
- Parasitics
- Manufacturing variations

The ideal unidirectional communication system is shown in Figure 5-2. It consists of a transmitter, a transmission line, and a receiver. Subsequent sections will explore the sources of noise in each part of the communication system.

5.2.1 Noise in the Interconnect

The ideal communication system uses ideal transmission lines. A more accurate model of the world needs to account for all of the parasitic and non-transmission line elements. These elements alter the signal and can be modeled as a source of noise. They are physically found between the bonding pad on the die and the attachment to the transmission line. The model shown in Figure 5-3 shows the following elements:

Bonding pad capacitance. The bonding pad is a large area of metal. It has a capacitance to the substrate. In addition, input protection structures add parasitic diffusion capacitances. Unless otherwise noted, all the analysis in this chapter will model this capacitance as 5pF to chip "ground".

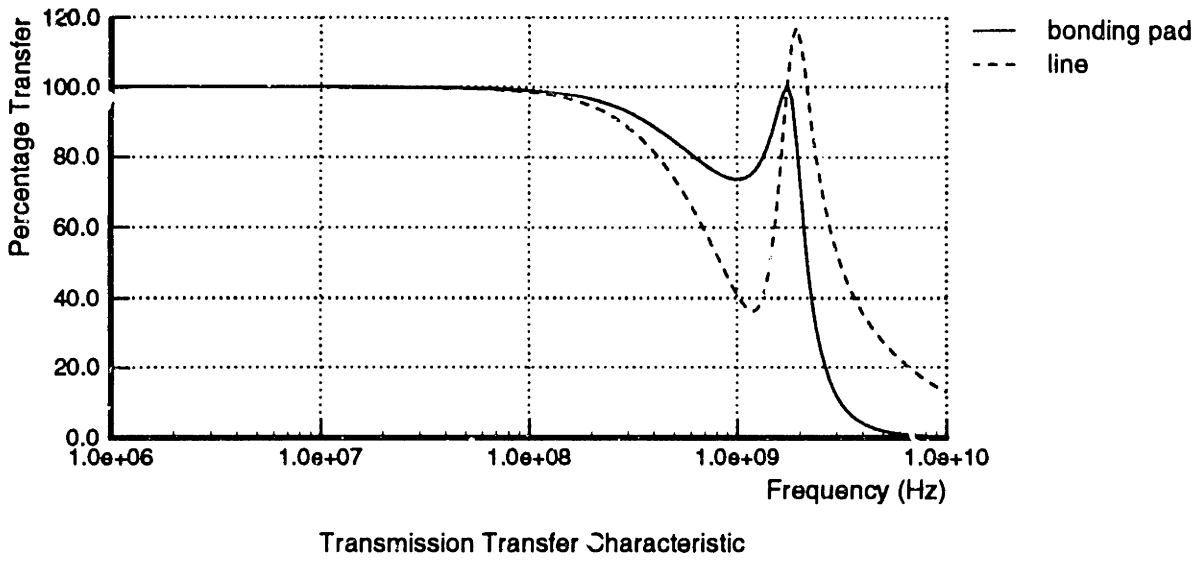


Figure 5-4: Interconnect transfer characteristic at the transmitter. Below 100MHz, very little signal attenuation is seen. At 1GHz, significant attentuation exists.

Bonding wire. The bond wire has both a resistance and an inductive component. The series resistances tend not have as strong an effect as the series inductances at high frequencies and will be omitted from the circuits shown. The inductance will be modeled as 3nH.

Package lead-frame. The term lead-frame is a bit misleading, but it used to describe the wiring from the bond site to the pin of the package. Depending on the type of package, it may or may not be controlled impedance.

Package Pin. The pin going from the package to the board has a parasitic capacitance. This will be modeled as a 5pF capacitor to board "ground".

These additional elements cause the interconnect to behave differently at different frequencies. Figure 5-4 shows the frequency dependence of the transfer curve. The transmitter in this case is trying to drive a 5mA sinusoid into the package model. One can see that the package begins to attenuate the transmitted waveform around 100MHz. Interestingly, the waveform seen at the bonding pad suffers less attenuation than the waveform on the line[1].

The effective input impedance of the package model is shown in Figure 5-5. The roll-off of the impedance above 100MHz is what causes the transfer function to roll-off. By expressing it as an impedance, one realizes that the package is over terminated at high frequencies. Since very sharp signal edges have strong high-frequency components, one would expect some reflections in the transient analysis. Figure 5-6 shows a current driver driving a 100ps edge into the interconnect model. By slowing the

[1]In section 5.4.5, this effect will be shown to reduce the bidirectional noise margin.

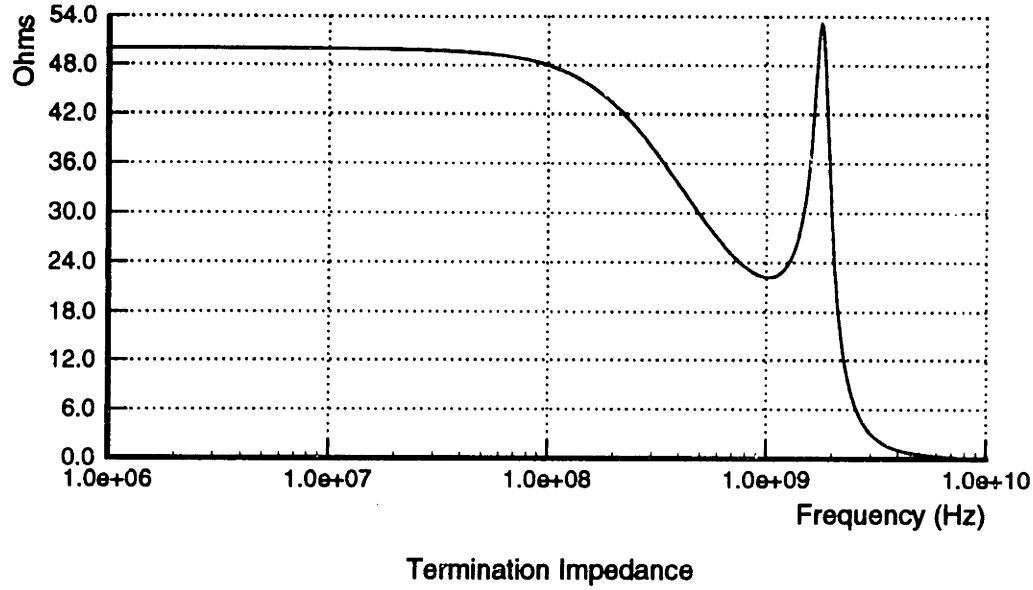


Figure 5-5: Interconnect impedance characteristic at the receiver. For frequencies below 10MHz, essentially of the impedance is caused by the 50 ohm termination. Above 100MHz, the capacitors in series with the termination resistor begin to dominate, causing the impedance to drop.

edge rate down, the amplitude of the reflection can be reduced. Figure 5-7 shows the effect of 1ns edge and Figure 5-8 shows the effect of a 2ns edge. The amplitude of the reflection has been reduced from 45mV to 15mV. Since the magnitude of the refection is proportional the magnitude of the applied signal, the 100ps edge has a 35% reflection, the 2ns edge has a 12% reflection.

Note that the interconnect noise is predominately self-induced noise and is not random. The magnitude of the noise injected is very predictable and the probability distribution for noise magnitude tends to roll off sharply.[2] Further, the magnitude of interconnect noise is directly proportional to the applied signal and is a function of the signal frequency.

5.2.2 Noise in the Transmitter

The role of the transmitter is convert the internal representation of the logic 0-1 into its external form. In this ideal transmitter, a logic 1 is converted into $+i$ current, a logic 0 is converted into $-i$ current. The ideal transmitter shown in Figure 5-2 cannot be realized, as not all supply and signal voltages are referenced to each other via circuit elements, nor are return paths provided for all currents.

Figure 5-9 gives a more complete picture. The internal logic signal to be transmitted is represented by a voltage referenced to the internal logic ground. The voltage

[2]Interconnect noise is like a dog tied to a stake. It can usually be found operating at the limit of its rope.

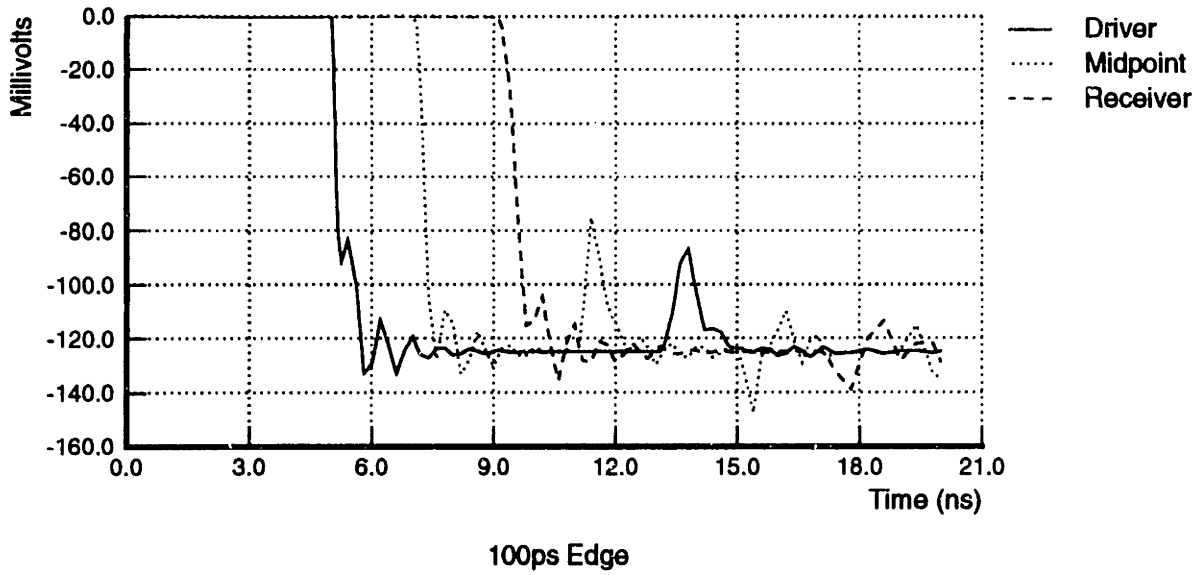


Figure 5-6: The interconnect structure driven by a 100ps edge. The 100ps edge has strong components above 100MHz, resulting in the waveforms displaying noise which resembles a decaying sinusoid.

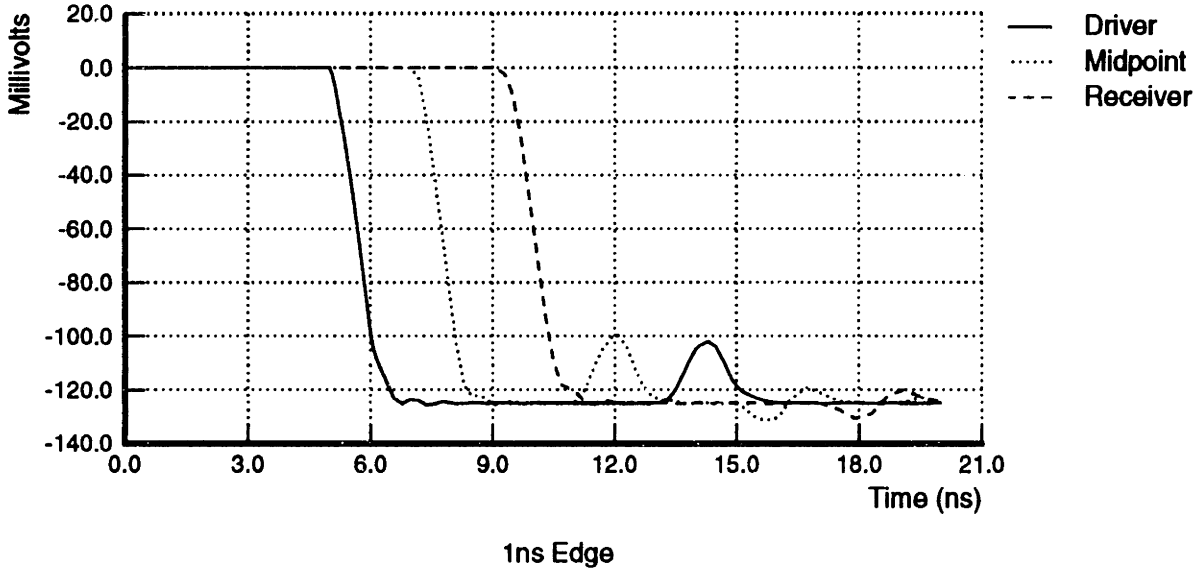


Figure 5-7: The interconnect structure driven by a 1ns edge. The reduction in high-frequency components eliminates the decaying sinusoid noise. There still is a substantial reflection back to the transmitter. The transmitter is terminated, but the first backward reflection is not completely damped, resulting in receiver noise at t=18ns.

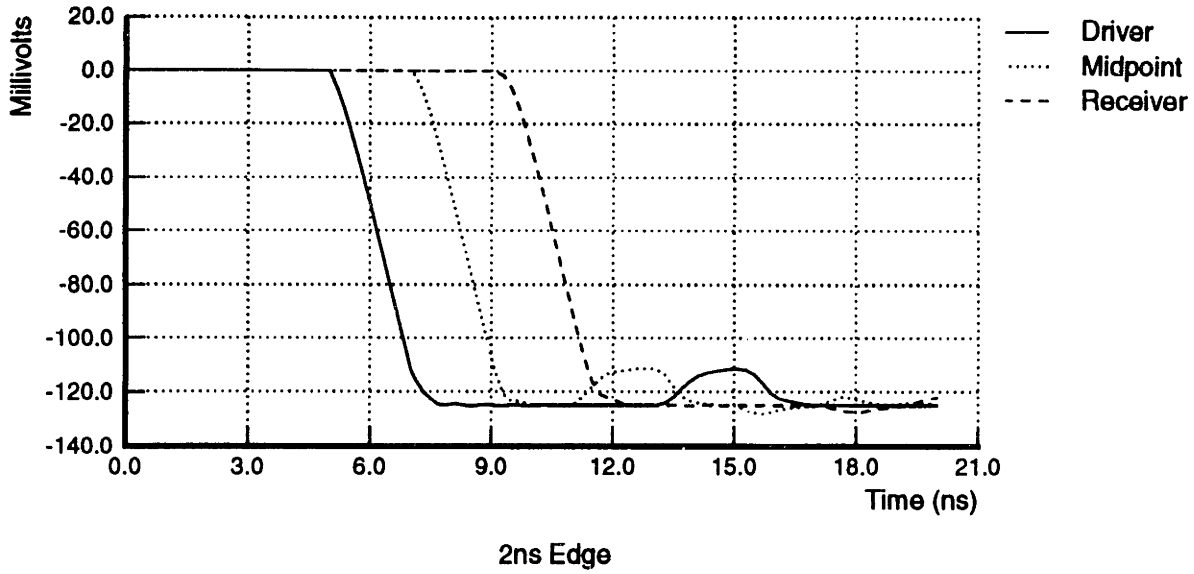


Figure 5-8: The interconnect structure driven by a 2ns edge. The reflection is now about 10% of the signal. The signal at the receiver is quite clean. The receiver still sees a reflection at t=18ns, but is now less than 5% of the signal.

can range from 0v to VDD . The internal logic uses a single VDD supply.

The actual transmitter also uses a single Vdd supply and its own ground. The transmitter's ground must be referenced to the logic ground via a circuit element (e.g. voltage supply, resistor, inductor).

The transmitter's output current must return somehow. The "shield" or return on the transmission line is therefore connected via some circuit elements to one of the transmitter's power supplies.

Transmitter Parasitic Circuit Elements

The next step in the modelling of the transmitter is to insert as many of the known parasitic elements and noise sources as possible. All wires which cross a chip boundary are converted into an inductor with a capacitor to "ground" on each end. These wires include the transmitted signal and its return, and all of the power supplies. Note that the internal logic and the transmitter use different supply wires. Lastly, voltage noise sources are inserted in series with all voltage supplies. Figure 5-10 shows the complete model.

Manifestations of Noise in the Transmitter

Noise is introduced when the signal crosses from the internal logic domain to the transmitter domain. The two domains are coupled via inductors and resistors. Since the current drawn through those inductors/resistors is not typically correlated, there will often be a ground differential between the domains. If the system is to function

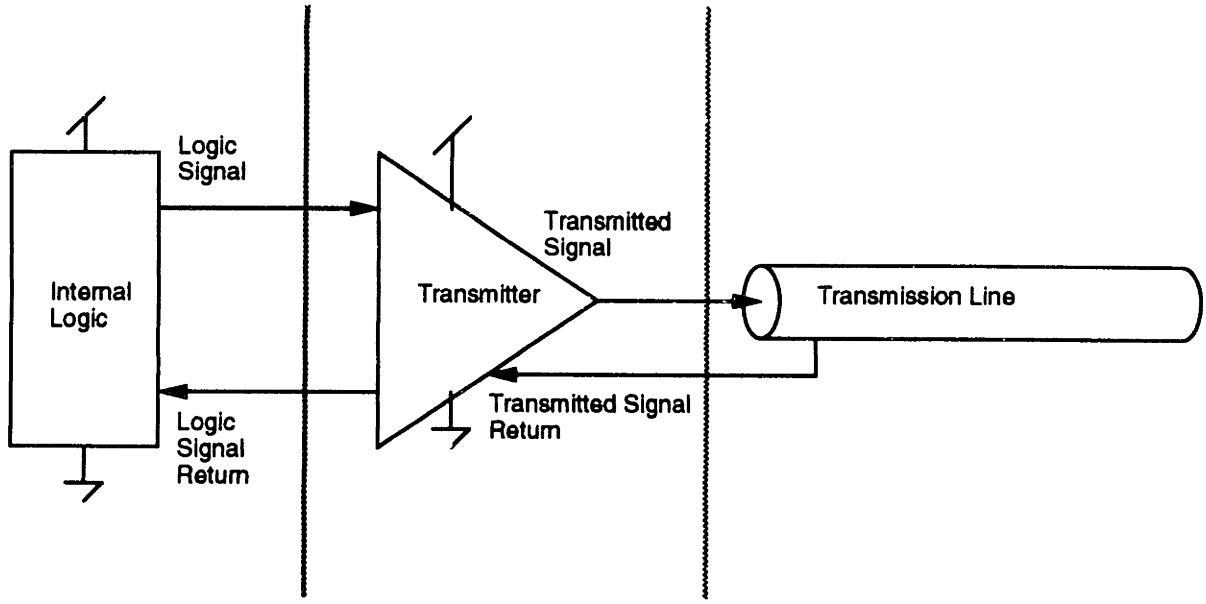


Figure 5-9: Unidirectional Transmitter with Current Return Paths.

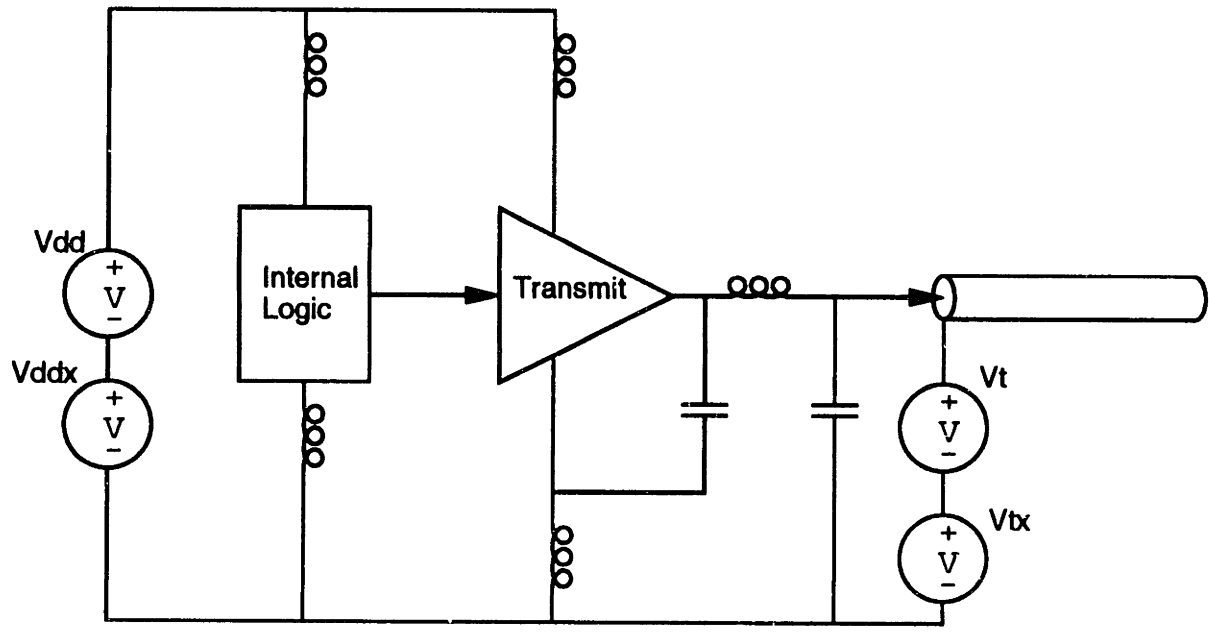


Figure 5-10: Unidirectional Transmitter with Current Return Paths.

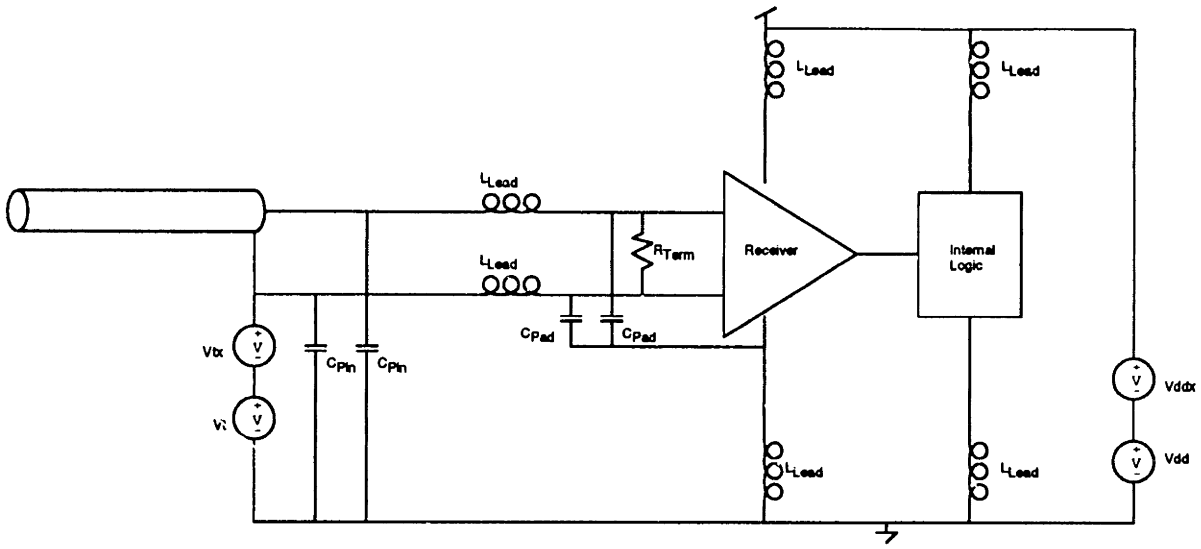


Figure 5-11: Receiver Noise Model.

at all, the worst-case differential cannot cause a constant logic 0 to be interpreted by the transmitter as a 1, nor should a constant 1 be seen as a 0.

The non-ideal nature of the physical circuits will always have an effect on signal transitions. If the signal edge rate is fast compared with the change in ground differential, the transmitter circuit will always see a monotonically increasing (decreasing) input signal. Provided that nothing else is drastically wrong, the transmitter should produce a clean but temporally modulated output edge. When the signal edge rate is slow compared to the change in ground differential, it is possible for the transmitter to see and produce a glitch in addition to temporally modulating the signal.

The connection from internal logic to the transmitter has one other subtle effect. In a CMOS transmitter circuit, the input structure is often an inverter. The NMOS transistor has its back-gate connected to ground, the PMOS transistor has its back-gate connected to Vdd. When the input signal changes, these capacitors must be charged and discharged, with the return current flowing through the power/ground wires. That change in current causes some amount of supply noise in the transmitter.

The last cause of transmitted signal noise is power supply noise. The transmitter is to be a current source, but the parasitic capacitances will allow high-frequency power supply noise to directly couple into the output. In addition, changes in the power supply voltage will modulate the circuit delay characteristics of the transmitter, resulting in additional output timing modulation.

5.2.3 Noise in the Receiver

Many of the same types of noise sources found in the transmitter can also be found in the receiver. The basic receiver model is shown in Figure 5-11.

The receiver output to internal logic signal suffers from the same sorts of noise

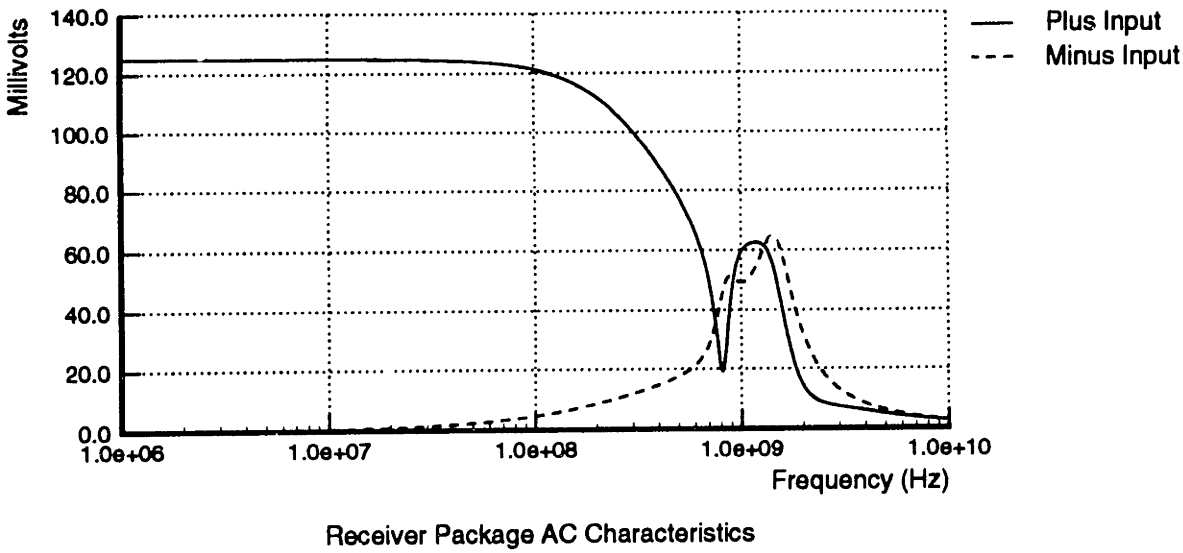


Figure 5-12: Receiver Transfer Characteristic.

problems as its counterpart in the transmitter did. These are mainly affiliated with "ground" differences between the receiver and internal logic. Ideally, the receiver circuit would draw constant current to avoid injecting power supply noise.

Most receiver circuits connect the inputs directly to the gate of a MOS transistor. For high gain amplifiers, the transistor is very wide, resulting in a large input capacitance. Capacitance on a signal is usually bad, as it makes the system frequency dependent. Figure 5-12 illustrates the frequency dependent transfer curves. One can see that the roll-off begins around 20MHz and is fairly substantial by 300MHz.

The parasitic elements will cause a glitch during the transient response. If the return current path is shared amoung several receiver sections, the glitch can be fairly substantial. Here, as with all of the interconnect examples, slowing the edge rate reduces the glitch. Figure 5-13 shows the result of a 2ns edge arriving at the receiver.

5.3 Circuit Techniques

Now that the major sources of noise are understood, circuit techniques can be developed to compensate.

5.3.1 Transmitter

The router uses current mode signalling. A basic current-mode output driver is shown in Figure 5-14. Transistors m1 and m4 act as current sources, m2 and m3 as switches. Relative sizings for m1 through m4 are 4x, 1x, 2x, 8x. Overall sizing is chosen to keep

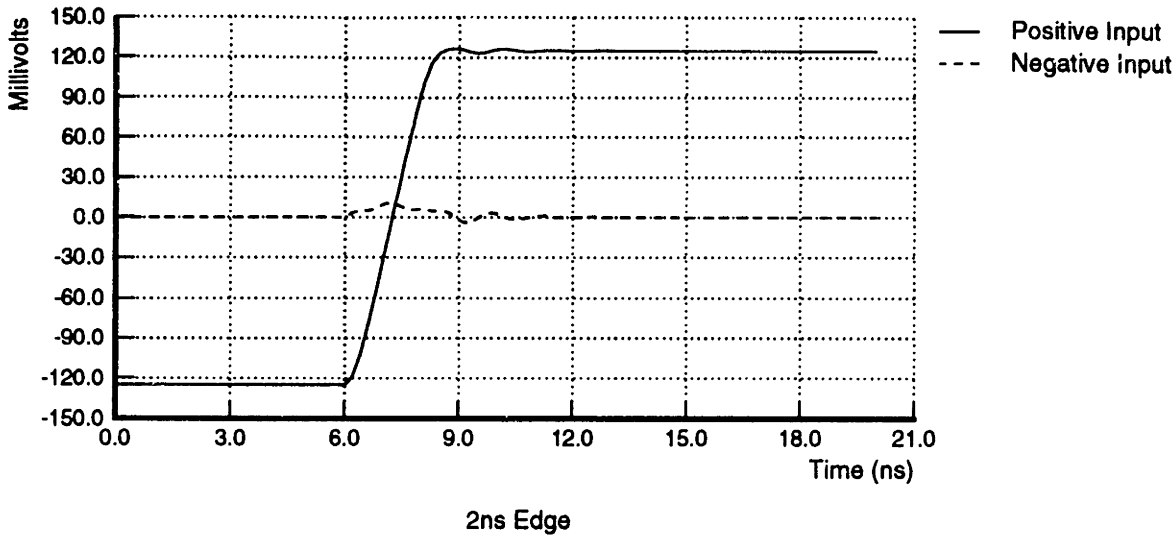


Figure 5-13: Receiver Transfer Characteristic.

m1 and m4 in saturation at the target current level.[3]

Current Steering

One of the problems with the current sources shown in Figure 5-14 is that the nodes n1 and n2 tend to have large parasitic capacitances. When decoupled from the line, these capacitances are charged to the rails. Later, when the output switches, the drive transistor sees the parasitic capacitor, not the current source. This results in undesirable initial current overshoot.

A solution is to *steer* the current from the current sources into either the line or a dummy load. This results in a constant voltage on all nodes in the current source, eliminating the initial overshoot.

The current steering driver has two other beneficial properties. First, it draws constant current from both the VDD and GND supplies, irrespective of the output state. This reduces the self-induced power supply noise which in turn reducing the required number of supply pins and the noise effects on the system logic. Second, the total current supplied from an output driver into VT is zero. This suggests that the VT supply could be as simple as a capacitor and a resistor divider.

In the discussion of the imperfect interconnect (section 5.2.1), it was mentioned that the edge rate of the driver needed to be limited to reduce high-frequency components. This is done by splitting the driver into several small parallel drivers, as shown in Figure 5-39. The turn-on of each driver is then staggered via a buffer chain. This technique allows the construction of not only a slow straight edge, but also a piece-wise approximation to more elaborate curves. These could more more closely match the desired low-pass characteristic.

[3]For those of us who are forgetful, the saturation region is when $V_{DS} \geq V_{GS} - V_T$.

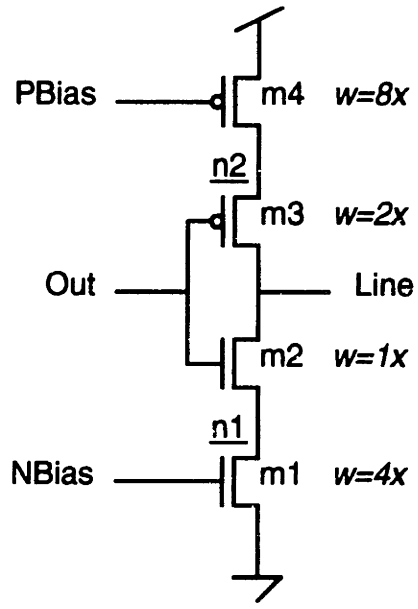


Figure 5-14: Basic Current-Mode Driver.

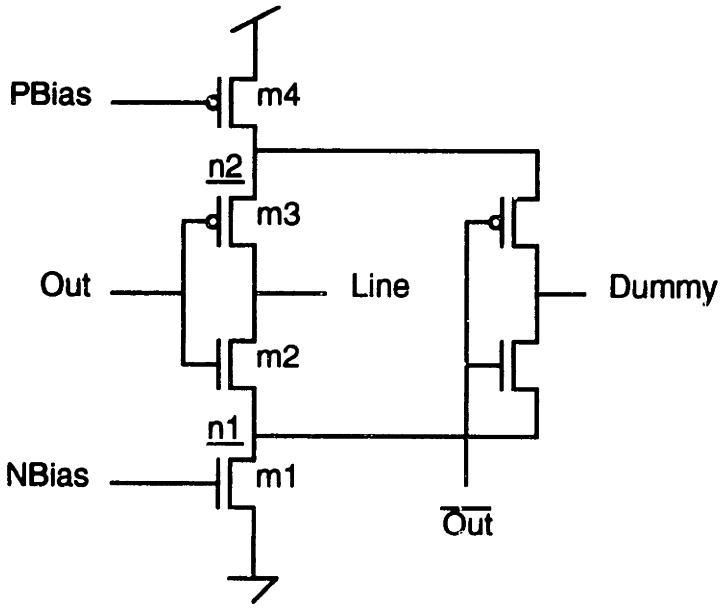


Figure 5-15: Current Sources with Steering.

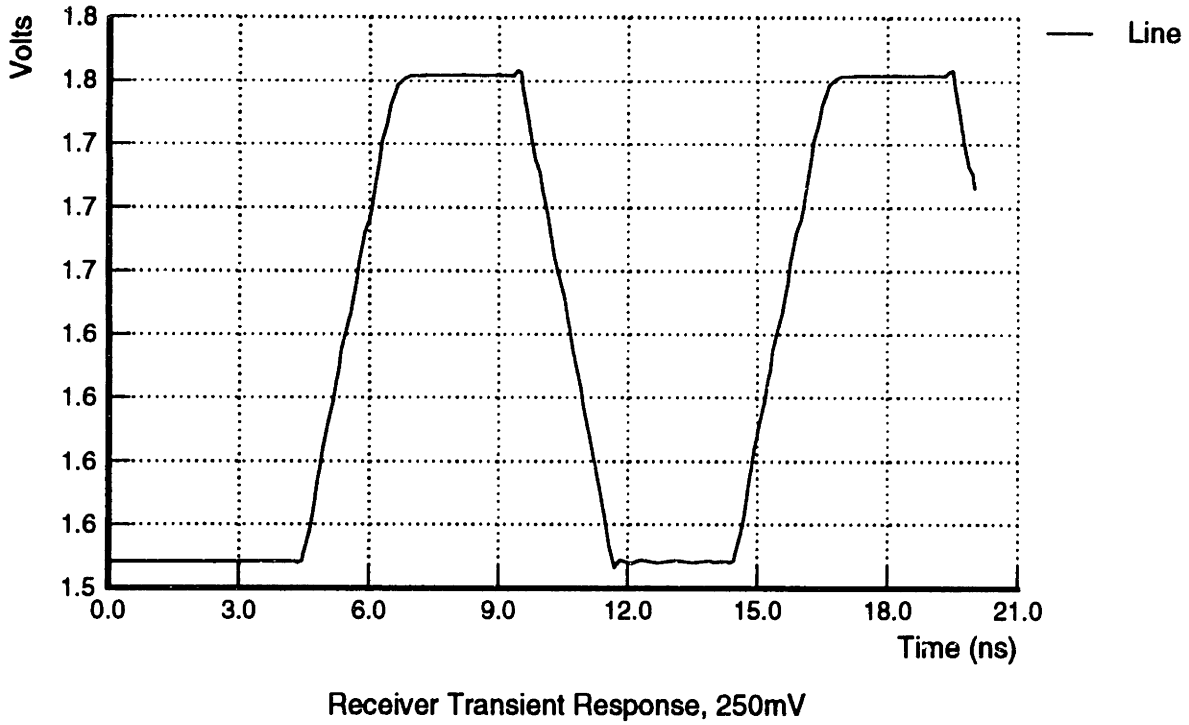


Figure 5-16: Transmitter Transient Response, 200Mbit.

Figure 5-40 show the detail of a transmitter stage. Each stage has a delay of about 2 gate delays. Delay characteristics can be adjusted by varying the voltage on the delay signal.

Figure 5-16 shows the transient response of the slew-rate limited driver. The input to the driver was a full CMOS signal with 200ps edge rates. As can be seen, the driver produces a nearly perfect trapezoidal waveform. Figure 5-17 shows the amount of supply current.

5.3.2 Transmission line and termination

There is very little one can do with circuit techniques to improve the transmission line. To minimize crosstalk, the line is terminated at both ends. The termination resistor is constructed using a CMOS gate. Figure 5-18 shows the effective termination resistance, measured at DC, $V_{DD} = 3.3V$, $V_T = 1.65V$. The resistance does vary as much as 15% over the range. Ideally, the resistance curve would be centered about 50 ohms for a $\pm 7\%$ deviation.

The resistance values were obtained at DC. Figure 5-19 shows the transient response of the termination to 100ps edges. One observes that waveforms look fairly ideal.

Due to process variations, the on-chip resistor needs to be tuned to match the line impedance. The router uses 14 discrete CMOS gates to implement a termination

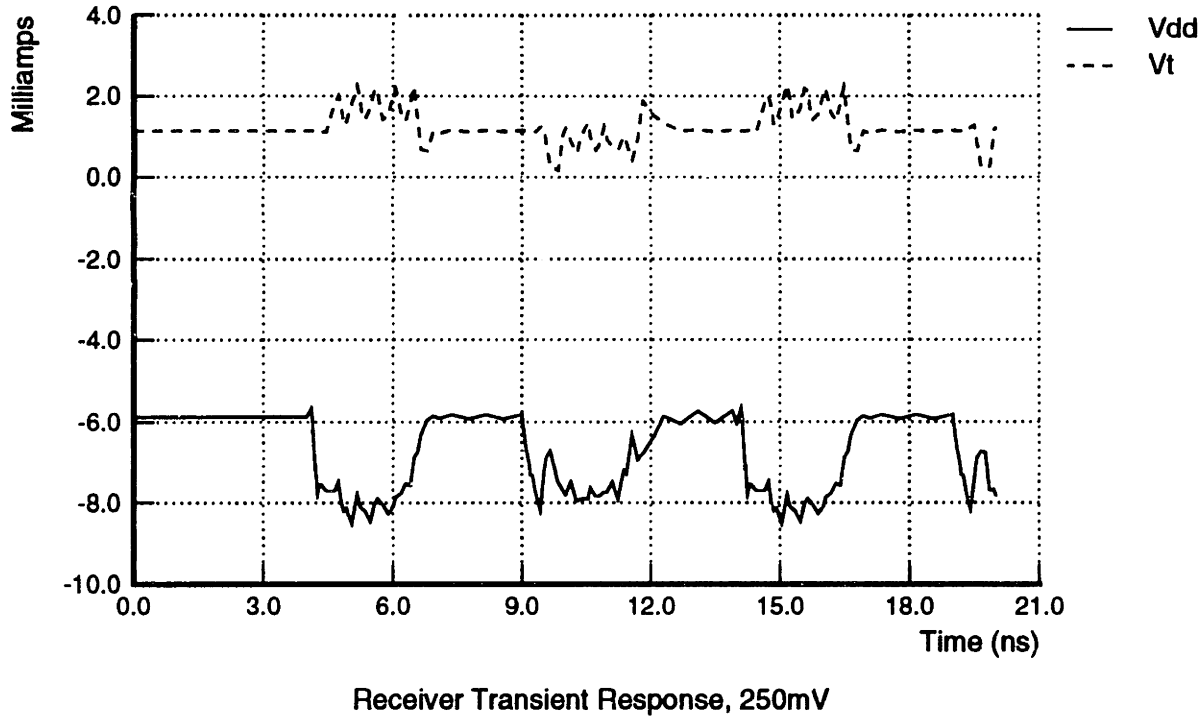


Figure 5-17: Supply Currents, 200Mbit.

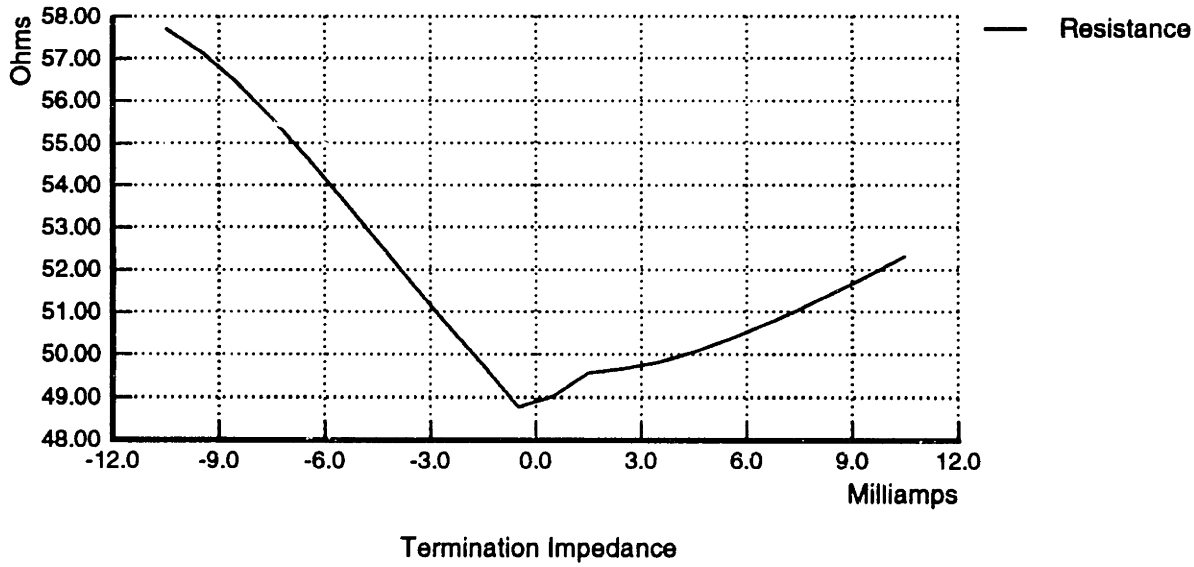


Figure 5-18: Termination resistance measured at DC.

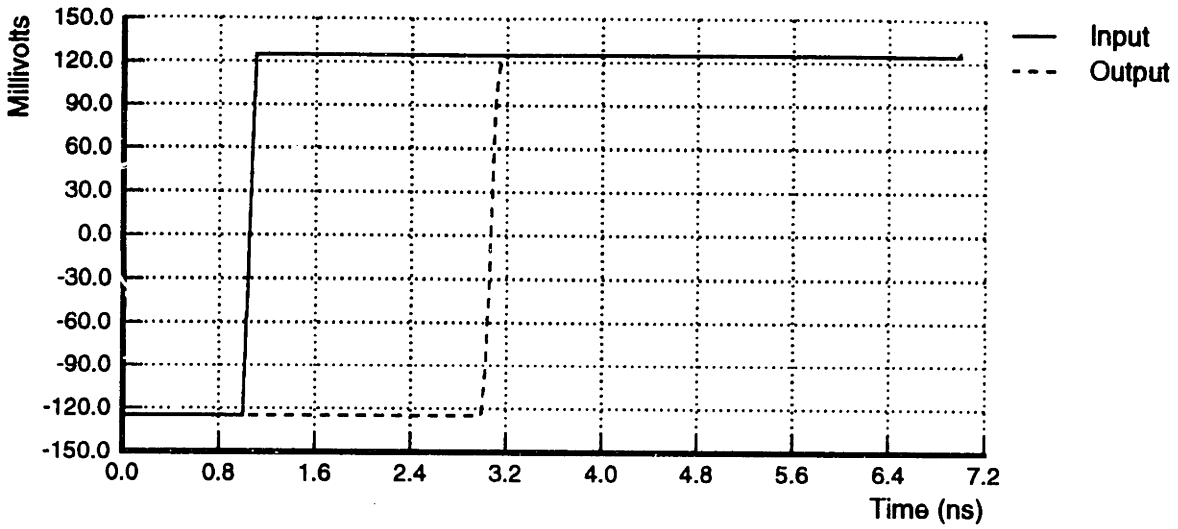


Figure 5-19: Termination Transient Response, 100ps Edges. The terminations are implemented using CMOS transistors. The parasitic capacitors do not affect the quality of the termination.

resistor. These are turned on and off under control from a JTAG register. The capacitances which slowed the very fast edge also allow the terminated signal to couple back into the control voltage. To mininize inter-pad effects, each resistor section buffers the control signal before applying it the the CMOS gate. Figure 5-42 show the circuit used in the router.

To summarize, the implementation of an on-chip termination does not significantly limit the acheivable edge rate, nor does it cause substantial reflective noise due to impedance non-linearities.

5.3.3 Receiver

The Reliable Router's current mode signalling results in the voltage swing on the line being $\pm 125mV$ while the voltage swing out of the receiver is still full CMOS levels. In doing so, the required gain-bandwidth product of the receiver was increased. The input voltage swing of the differential amplifier is 250mV, ignoring noise. The gain required is then 20 (26dB). To achieve a 200Mbit signalling rate, the receiver should have a gain-bandwidth product of 2GHz.

The gain-bandwidth product must be measured large-signal, as the high-frequency output swing must match the low-frequency output swing to avoid inter-symbol effects. Using a good CMOS process, it is fairly easy to construct a standard transconductance amplifiers with a unity-gain frequency near 2GHz. For a standard transconductance amplifier, these gains are only for a limited output range which is typically $\pm 1V$. Outside of that range, the transistors in the output stage come out of satu-

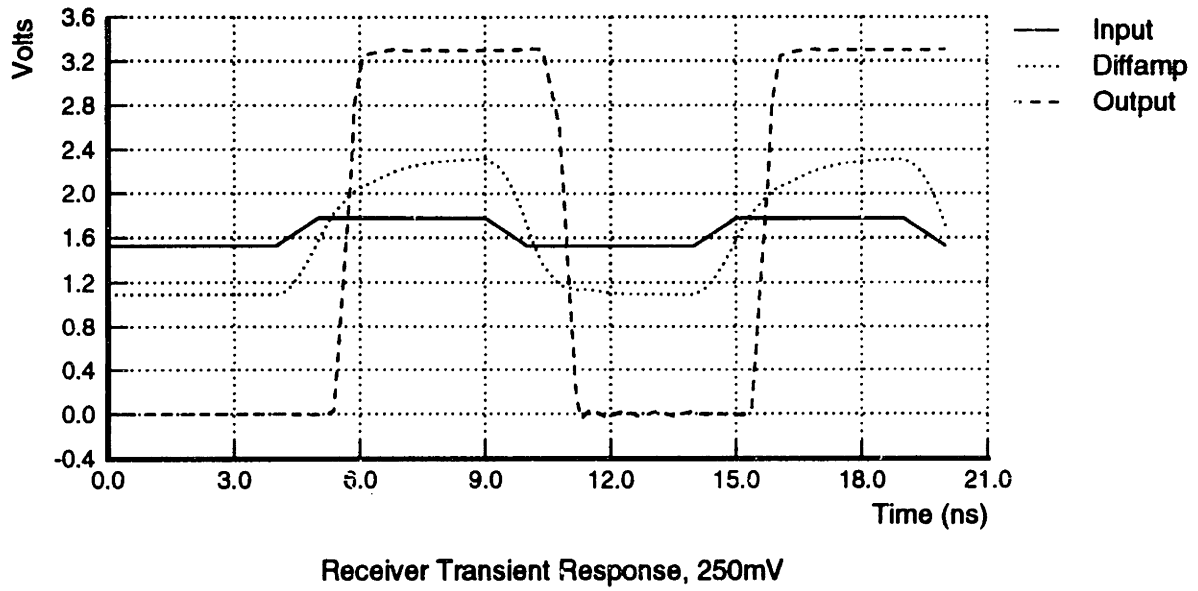


Figure 5-20: Receiver Transient Response, 200Mbit, 250mV. The output of the first yields stage nearly 1.2V of swing.

ration and enter the triode region. Once in triode, they supply less current and the output slew rate falls off.

The transconductance amp used in the router is a standard Chappel differential receiver. It is followed by two inverters which provide the full restoration to CMOS levels. The schematic is shown in Figure 5-43.

The transconductance amplifier can be adjusted to give large output swings by allowing one of the two output drive transistors to enter cut-off. This produces the rail-to-rail swings shown in Figure 5-20. The input waveform (labeled "input") is 250mV peak-to-peak, with 1ns edges. The waveform labeled "diffamp" is the output of the differential amplifier. This output is then cleaned up using a buffer whose output is labeled "output".

The receiver's gain is further illustrated in Figure 5-22. The input waveform has been reduced to 20mV peak-to-peak with 1ns edges. The output waveform shows a bit of intersymbol interference, yet the output waveform is still clean enough to operate at 200Mbit/sec. This implies that over 90% of the input waveform can be noise and the receiver will still extract the correct value.

The receiver does draw variable amounts of current, depending on the input signal and the input edge rates. Figure 5-23 shows the supply current drawn while a 250mV signal is applied. The current spiking is comparable to that in the transmitter.

5.3.4 Undirectional Signalling Summary

This section explored the physical and the circuits used to construct a high-performance, unidirectional signalling system. The circuits developed show excellent performance,

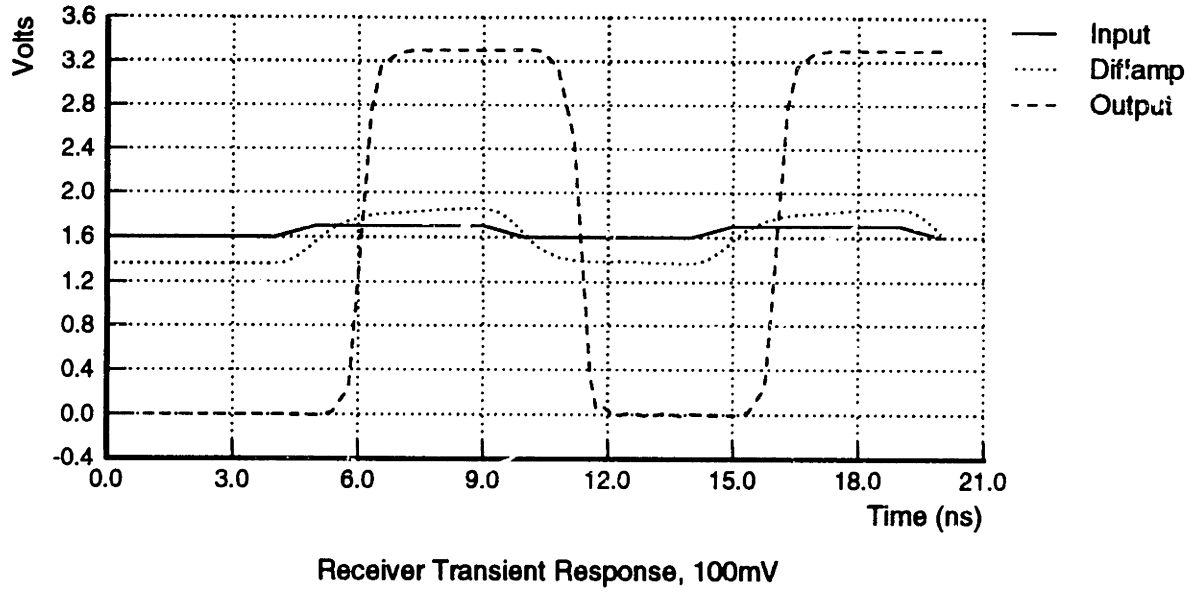


Figure 5-21: Receiver Transient Response, 200Mbit, 100mV. The output of the first stage is reduced. Most of the level restoration is now handled by the buffer.

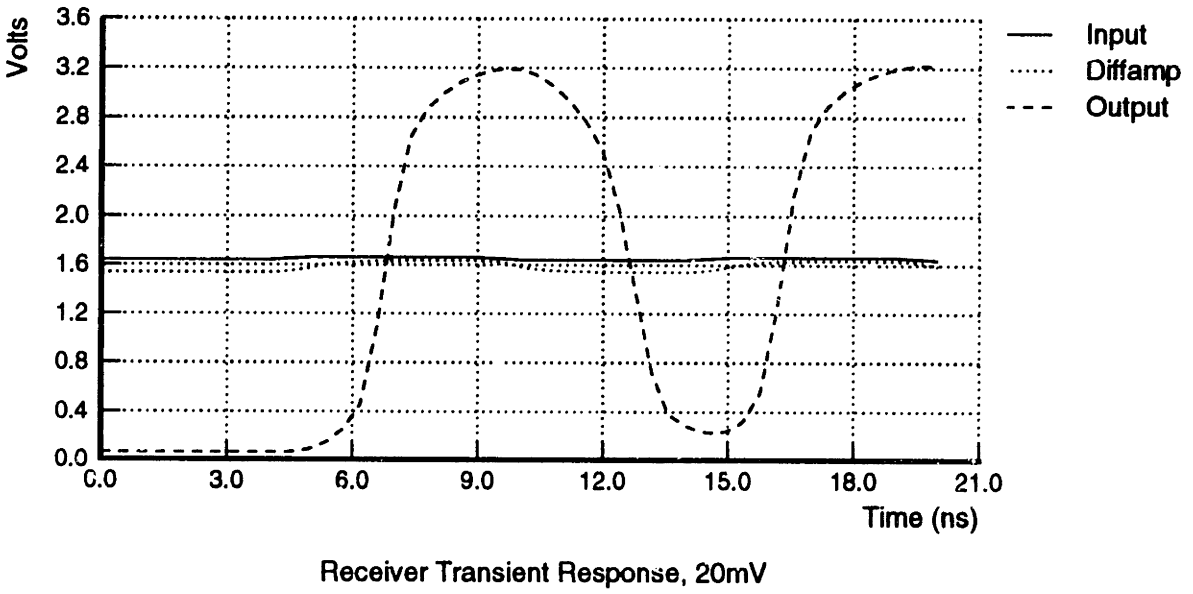


Figure 5-22: Receiver Transient Response, 200Mbit, 20mV. The differential amplifier is providing some gain, but now it is primarily level-shifting the voltage to allow the buffer to do the amplification.

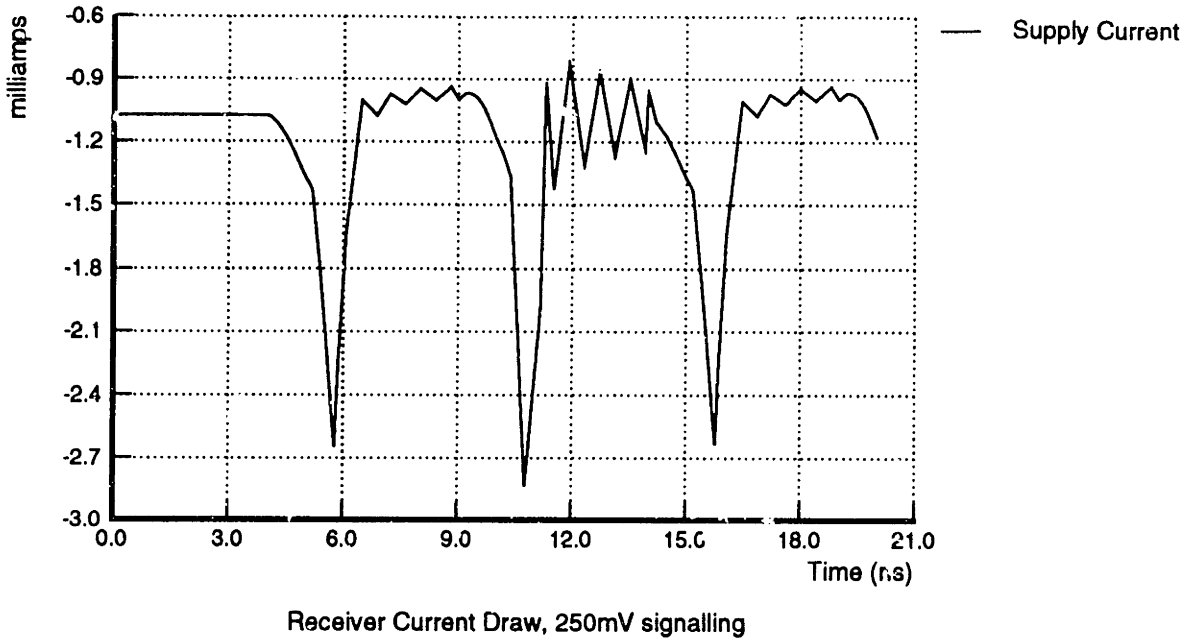


Figure 5-23: Receiver Supply Current, 200Mbit, 250mV. The average current draw is about 1mA, with peaks to 2.8mA.

yet the limiting factor has been the parasitic circuit elements that come with the physical interconnect. Figure 5-24 shows the waveforms as produced by the current mode transmitter and then as seen by the receiver. The signal quality and noise margins are excellent and the receiver should have no difficult restoring the signal to full CMOS levels.

5.4 Bidirectional Signalling

The basic circuits described thus far can be used to implement simultaneous bidirectional signalling. The signalling concept is to treat the terminated transmission as a summing junction for the currents supplied by two drivers. The voltage seen on the junction should be $R_{term}(I_A + I_B)$. Each driver/receiver pair knows the amount of current it sent into the junction and sees the resultant voltage. By constructing a reference waveform and a differential amplifier, the receiver substracts off its contribution, resulting in the signal sent by the other driver. Figure 5-25 shows the circuit topology and Figure 5-26 gives some example waveforms.

The same circuits used in the unidirectional case can used for bidirection signalling. The current-steering current sources can be used to construct the drive and reference waveforms, the termination resistor will convert it into a voltage, and the differential amplifier can be used to subtract off the reference waveform.

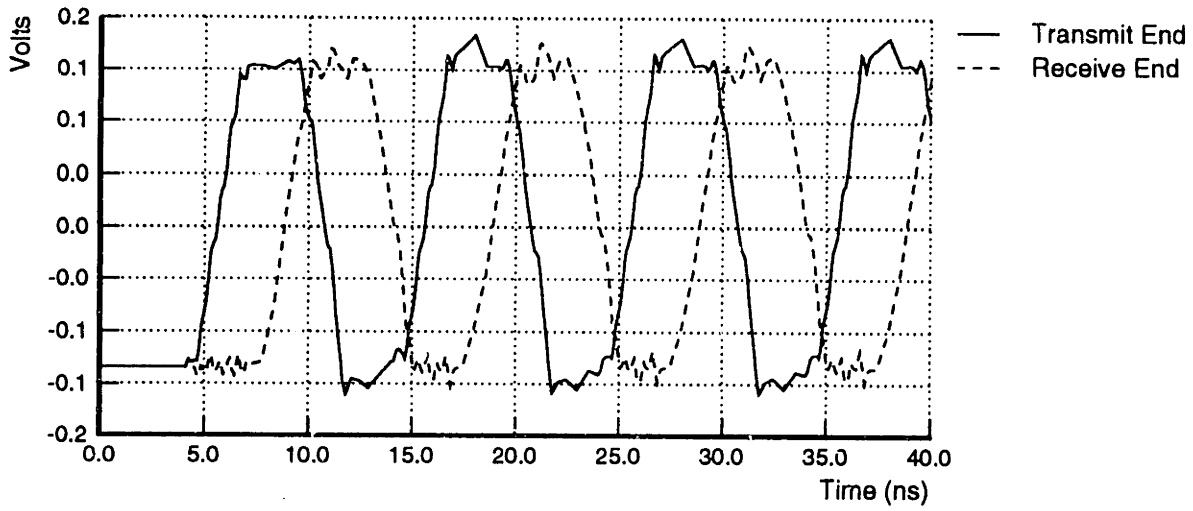


Figure 5-24: Unidirectional Signalling.

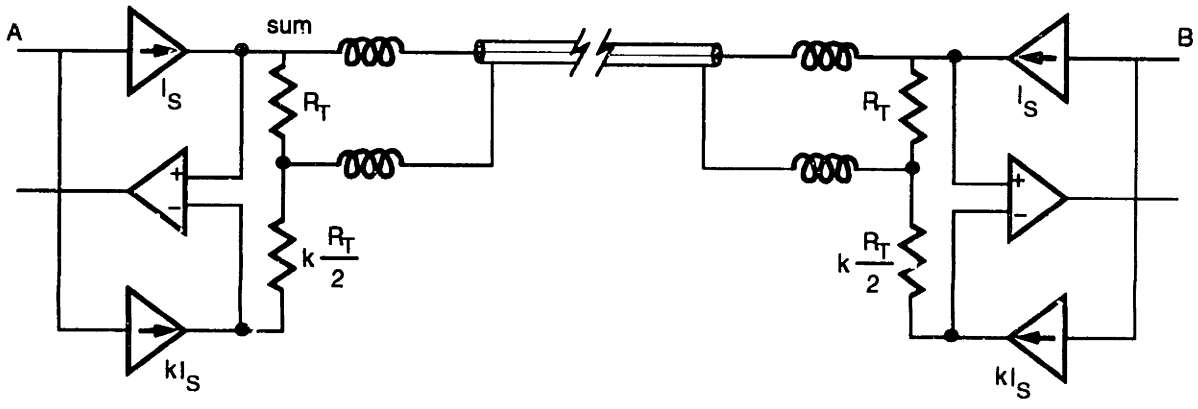


Figure 5-25: Simultaneous bidirectional signalling. This is accomplished by transmitting signals in both directions across a transmission line. At each end, the received signal is recovered by subtracting out the effects of the transmitted signal. A current source scaled by factor k is used to subtract out the transmitted signal to reduce power dissipation.

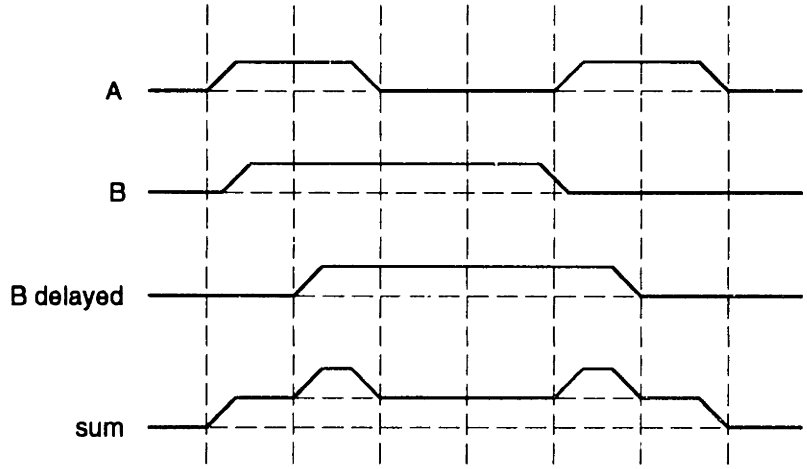


Figure 5-26: Waveforms for bidirectional signalling. The superposition of signal A with a delayed version of signal B gives the waveform on the left end of the transmission line.

5.4.1 Additional Bidirectional Noise Factors

Since the same circuit components are used for bidirectional signalling, one would expect that a bidirectional system could achieve the same bandwidth each direction as a unidirectional system. However, there are some additional limiting factors:

- In the unidirectional case, the actual output impedance of the driver was not much of a concern. In the bidirectional case, the impedance should be high enough to avoid improperly terminating the line.
- The signalling range is twice as large. The inaccuracy of the termination grew with the amplitude of signalling voltage.
- The driver and reference waveforms essentially create a large common mode signal at the receiver's input. This signal must be rejected by the receiver.
- The package parasitics will cause a transient mismatch between the waveform on the line and the reference waveform.

These will now be explored in turn.

5.4.2 Driver Output Impedance

Figure 5-27 shows how the output current changes when an offset voltage is applied. The current changes by 0.2mA over a 300mV range. The effective output impedance is then .3/.0002 = 1500 ohms. This should not any significant impact on the 50 ohm termination. It will change the amplitude of the added voltage by about 5%.

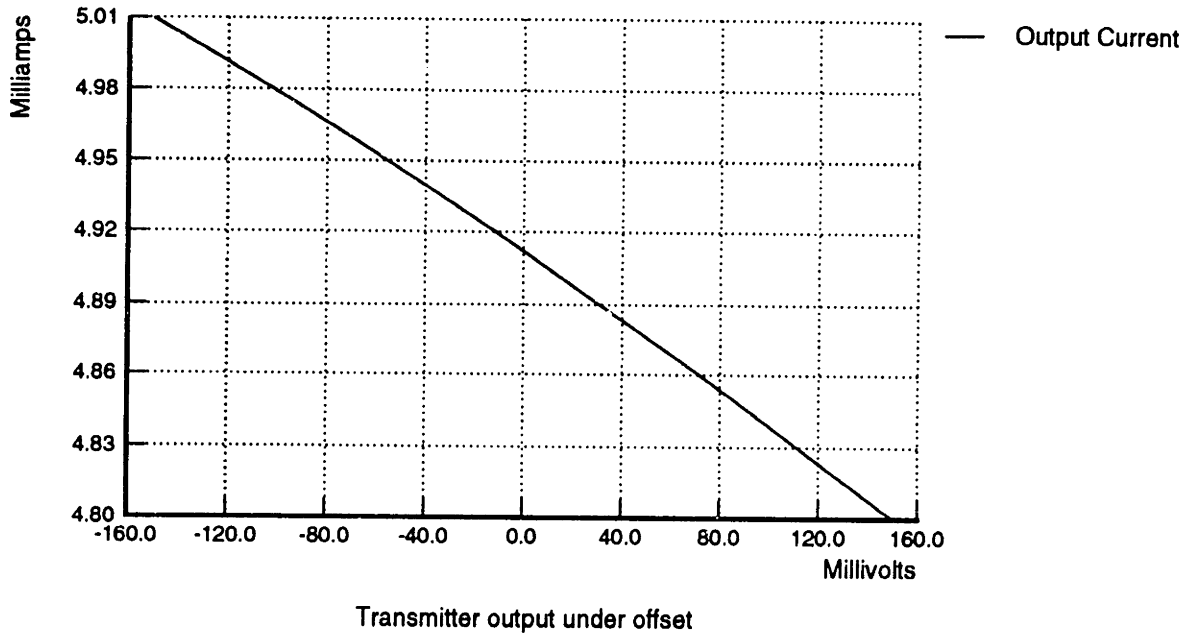


Figure 5-27: Driver output current.

5.4.3 Termination Nonlinearity

The termination resistance was plotted over the entire operating range in Figure 5-18, However, the transient analysis needs to be adjusted for a larger current swing. Figure 5-28 shows the effect of increasing the signal swing to $\pm$10mA, which is a small amount of overshoot.

5.4.4 Large Common-Mode Signal

The large common signal will feed through the differential amplifier onto the gate of the first inverter in the output stage. Figure 5-29 shows the effective logic threshold voltage as being in the range 1.5 to 1.6 volts. As long as the common mode signal does not cause the amplifier to cross that threshold, the correct value will be recovered. Figure 5-30 and Figure 5-31 show the effect of 250mV of common mode noise applied to a 125mV positive and 125mV negative differential input respectively. As can be seen, the amplifier output stays below 0.9V and above 2.7V, resulting in a worst case noise margin of 0.6 volts.

Figure 5-32 and Figure 5-33 reduce the differential voltage to $\pm$50mV. Here, the amplifier output stays below 1.25V and above 2.1V, resulting in a worst case noise margin of 0.2 volts. This ought not to be troublesome, as the buffer is physically next to the differential amplifier and shares the same supply rails. If a fifth of a volt of noise is coupling onto that node, the system is already non-functional.

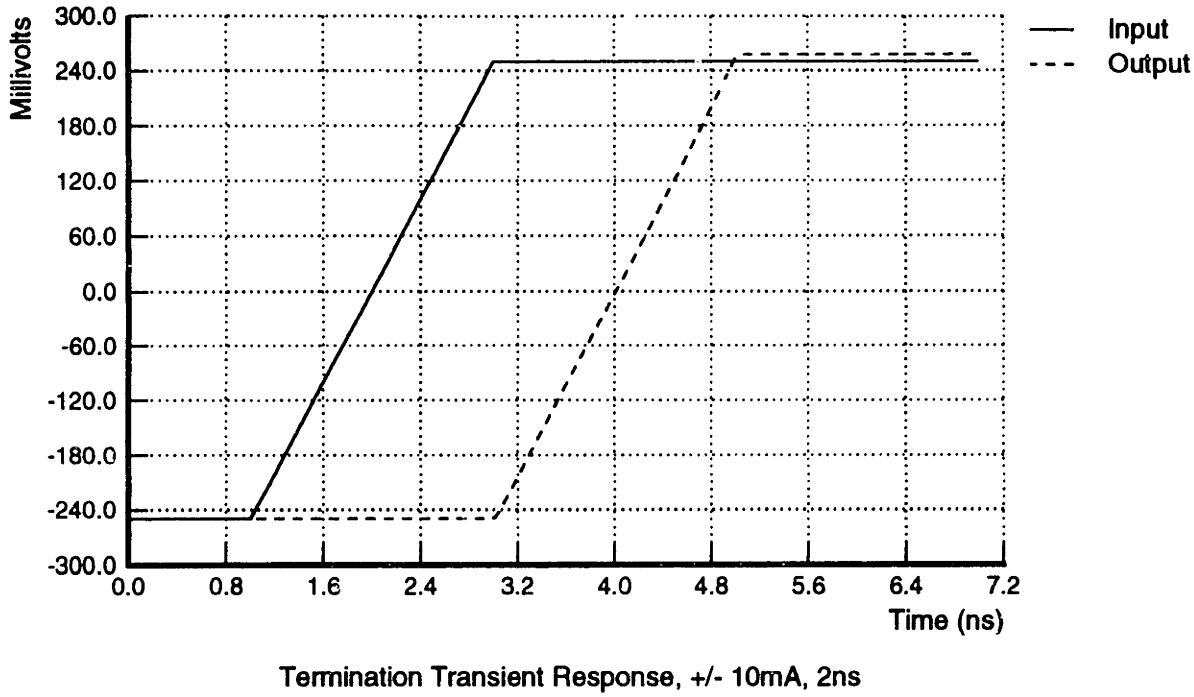


Figure 5-28: Termination under larger current swing.

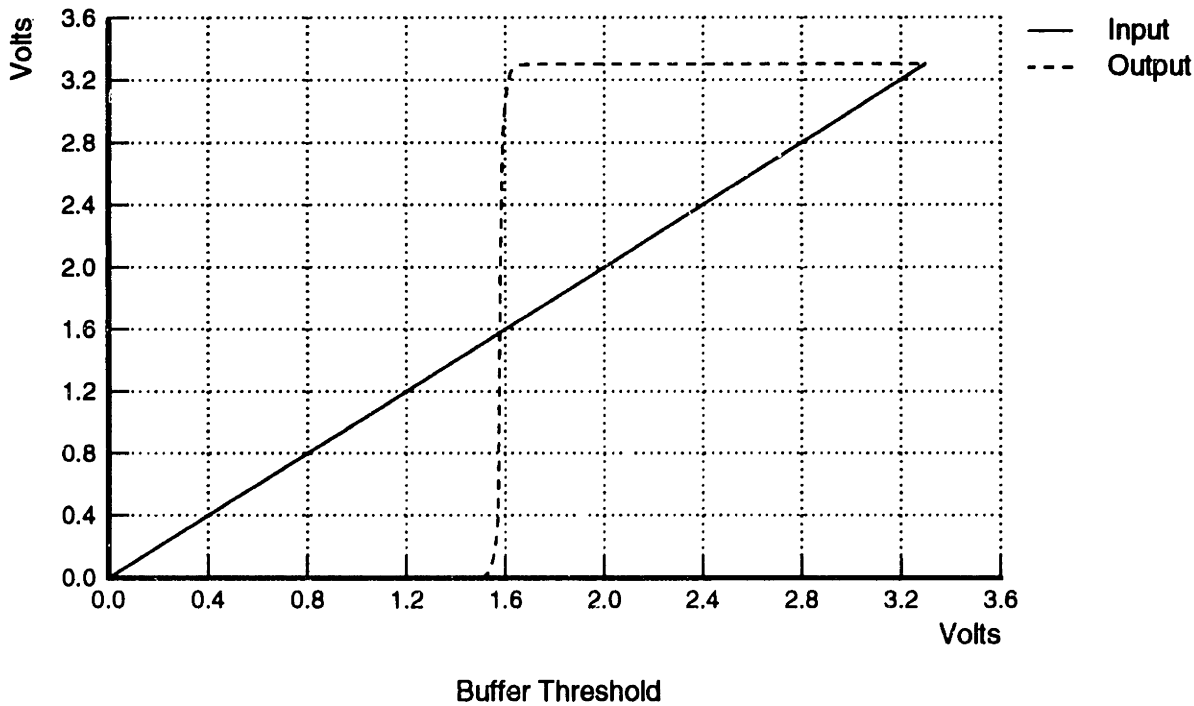


Figure 5-29: Buffer Transfer Curve. Not surprisingly, a CMOS buffer shows very large gains in its DC transfer function.

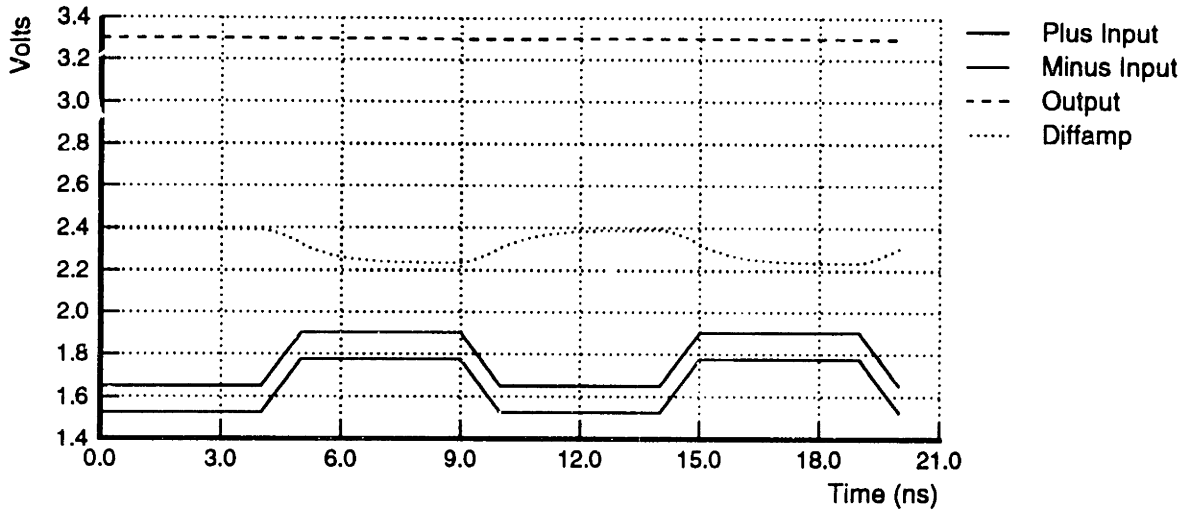


Figure 5-30: Common mode response to a 250mV common mode signal. The differential input corresponds to logic one. The output of the first stage of receiver stays well above the input threshold of the buffer.

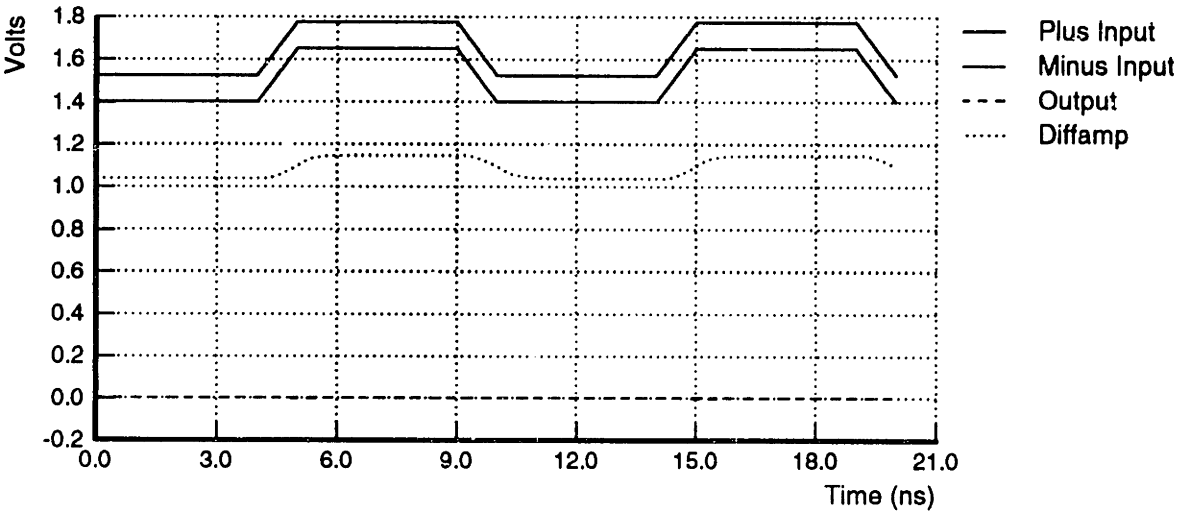


Figure 5-31: Common mode response to a 250mV common mode signal. The differential input corresponds to logic zero. The output of the first stage of receiver stays well below the input threshold of the buffer.

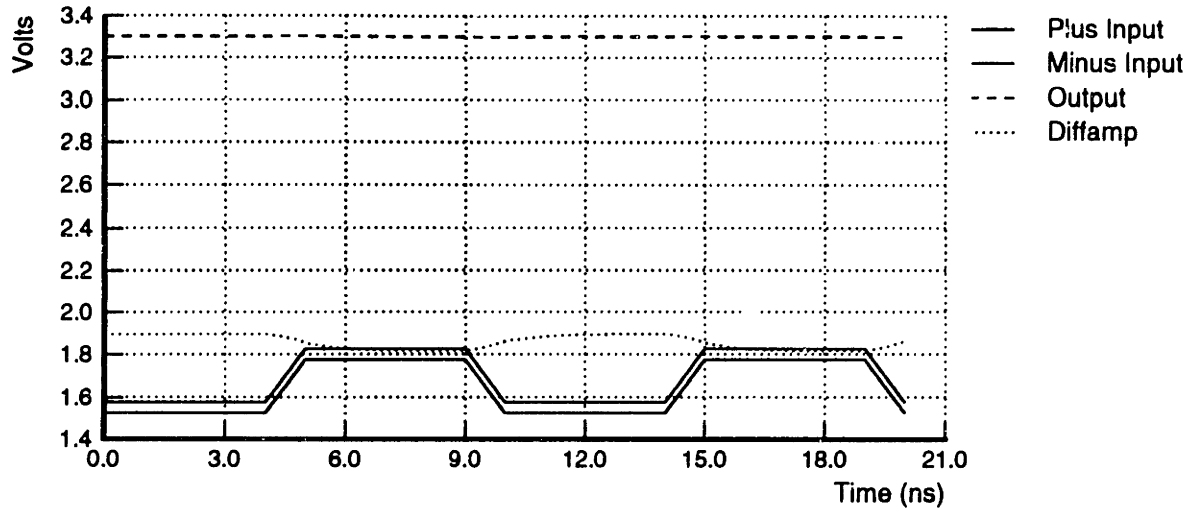


Figure 5-32: Common mode response, 250mV common mode signal, 50mV differential signal. The differential input corresponds to a logic one. The output of the first stage is about .2V above the buffer's threshold.

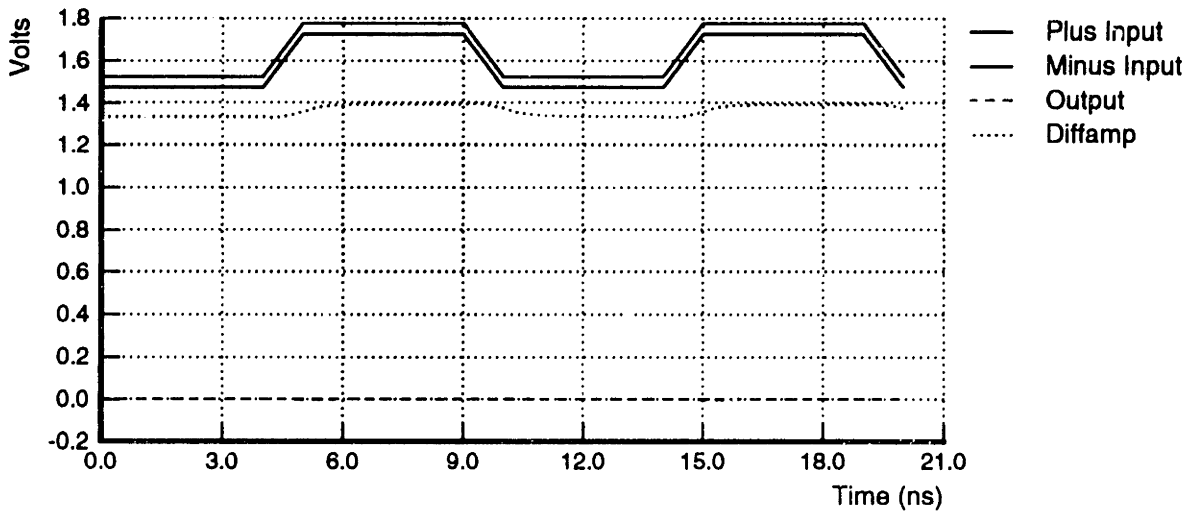


Figure 5-33: Common mode response, 250mV common mode signal, 50mV differential signal. The differential input corresponds to a logic zero. The output of the first stage is about .2V below the buffer's threshold.

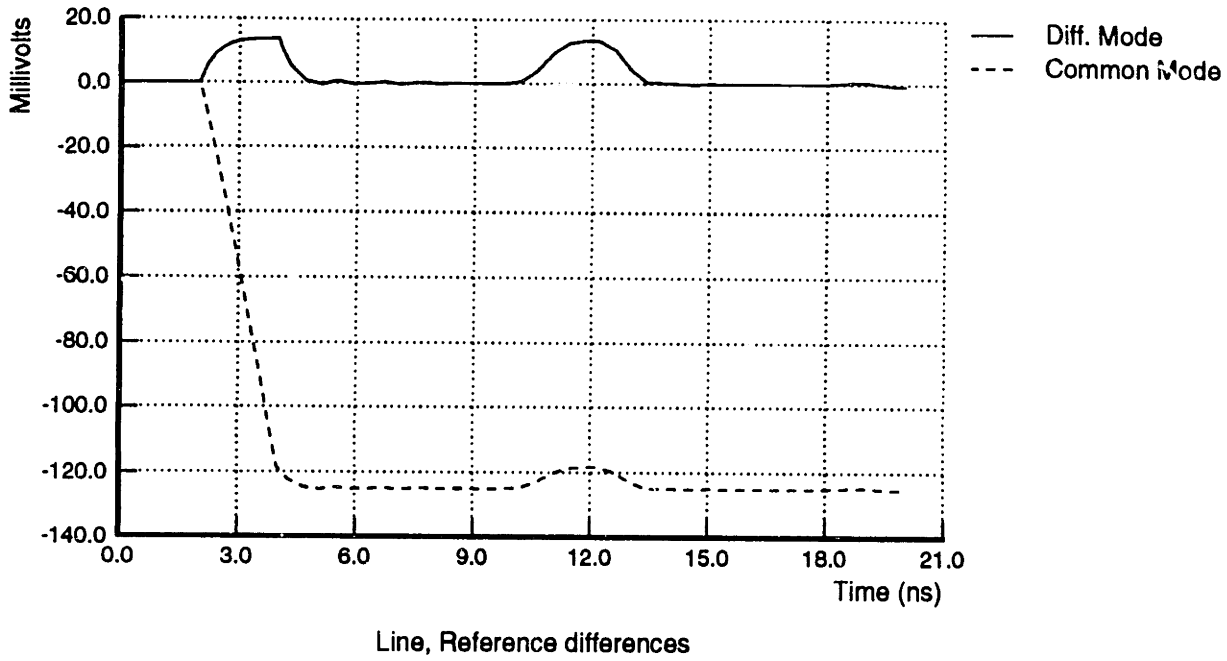


Figure 5-34: Differences between the line and reference waveforms owing to package parasitics.

5.4.5 Discrepencies between the line and reference waveforms

The receiver used in the router has been shown to operate correctly with a differential signal of ±50mV and a common mode signal of ±125mV. Since the intended signal is ±125mV, over half the signal can be lost to noise. Part of this noise margin is lost to a discrepency between the transmitted and reference waveforms. Owing to all the parasitic inductors and capacitors in the system, there will be a transient differential error introduced between the reference waveform and the waveform on the line. The summing node is connected to an on-chip terminator and to an inductor to the transmission line, while the reference waveform merely has the on-chip termination. Figure 5-34 shows the effect of the lead inductance.

The amount of the error is about 15% of the signal amplitude. In Figure 5-34, the error waveform is reflected back to the transmitter. This reflection is already accounted for in the interconnect analysis, it does not need to be accounted for here.

5.4.6 Additional Noise Sources

In bidirectional signalling, a reference waveform is subtracted from the line waveform. These waveforms are constructed by injecting current into resistors; the resistors are tied to the termination voltage, VT . Any noise on VT will manifest itself as common mode noise to the receiver. This noise should be minimized.

The noise on VT is the result of changing the amount of current carried by the VT supply leads. The transmitter steers current, under ideal conditions it draws

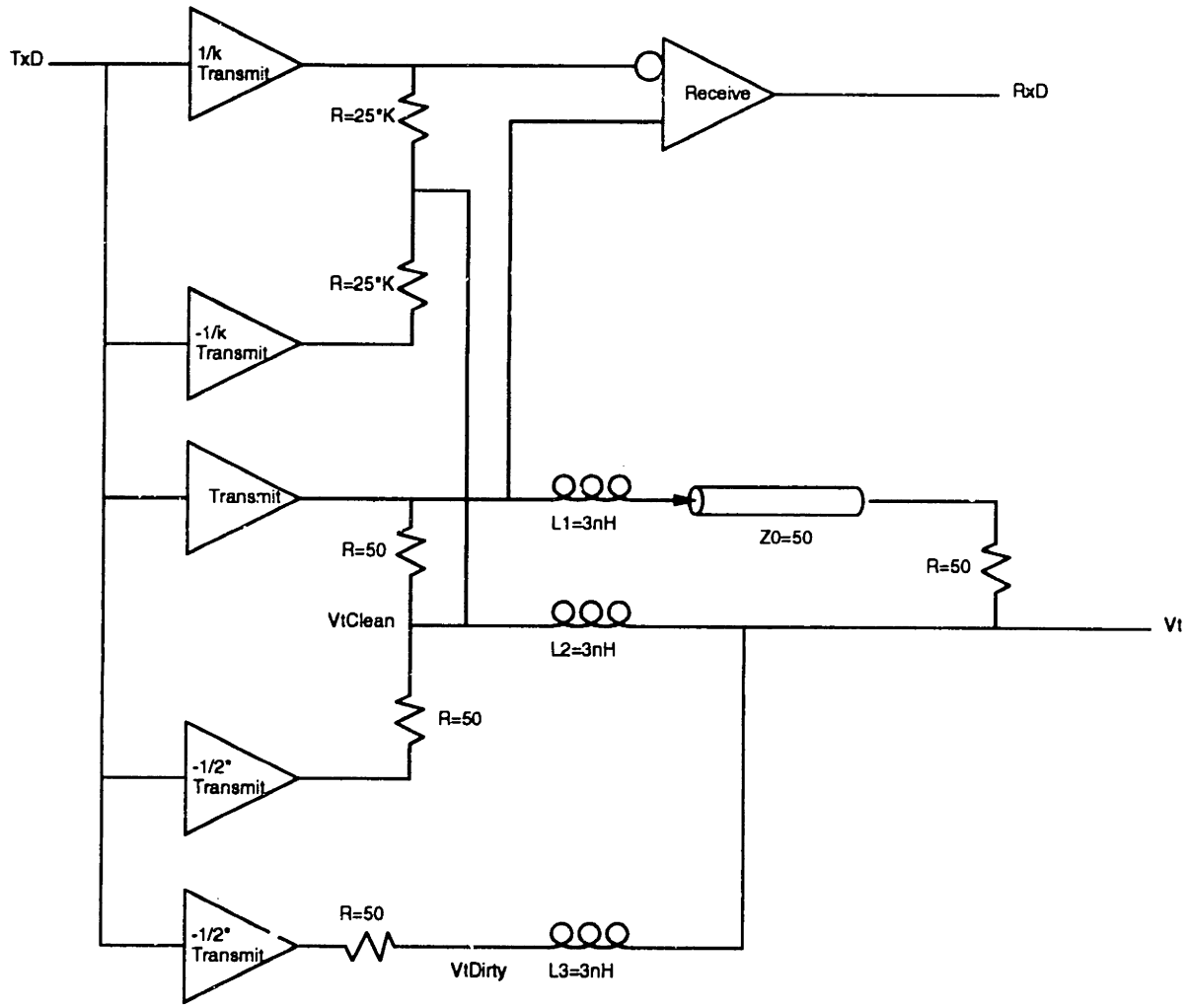


Figure 5-35: Separating out the return current. No current should be flowing across inductor L2, resulting in a clean VT signal on-chip.

no current from the VT supply. The receiver has high-impedance inputs and also draws no current from VT supply. The change in current over the VT supply leads is caused by the return current - current first sent as signal by the transmitter to the receiver and is now returning to the transmitter via the VT supply.

This return current thus causes noise in both transmitter and receiver sections. Within a transceiver, the transmitter section can reduce the common-mode noise it causes in the receiver section by separating out the signalling return current from its other currents and sending the return current over a different supply lead. The circuit is shown in Figure 5-35.

Note that some of the signalling current will be shunted through inductor L2. Inductor L1 is resisting the change in current across it caused by the signal, so the current from the transmitter travels through the local termination, into the VT clean node and out through inductor L2. Without this separation, the change of 5mA

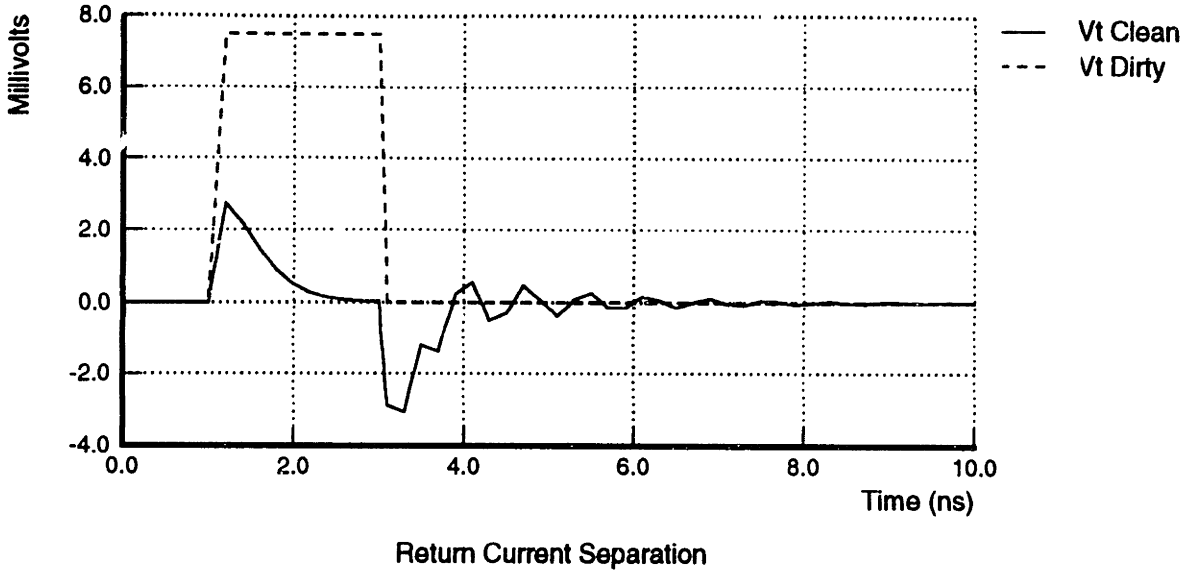


Figure 5-36: Noise on VT clean, VT dirty. Owing to the inductor in the signal path, some current flows briefly through VT clean, resulting in common mode noise.

Cause	Amount	Total
Transmit reflection	12mV	12mV
Line, reference differences	12mV	12mV
Transmit current imbalance	2mV * xmits per VT clean	10mV
Termination mismatch	5mV	5mV
Receiver parasitics	3mV + 3mV * recvs per VT dirty	18mV
Total		57mV

Table 5.1: Total Noise Contributions.

in 2ns through a 3nH inductor would have resulted in 7.5mV of noise added to the internal VT node. Figure 5-36 shows the separation of the currents reduces the noise to around 2mV.

5.4.7 The Noise Margins

The preceeding sections have generated a large amount of numbers. Putting it all together results in Table 5.1. Some of the noise figures depend on the number of transceivers sharing the VT pin. The Reliable Router has 5 transceivers per VT clean or dirty pin. The sum of the noise sources amounts to 57mV. The overall signal swing is $\pm 125mV$, leaving 68mV for the gain-bandwidth of the receive amplifier.

In the simulations, it was shown that 50mV was sufficient to provide a bandwidth of 200Mbit/sec. Further, a 50mV differential input on the receiver was not perturbed by a 250mV common mode signal at 200Mbit/sec. From this, one can conclude that the system should indeed function with around 18mV of noise margin to spare.

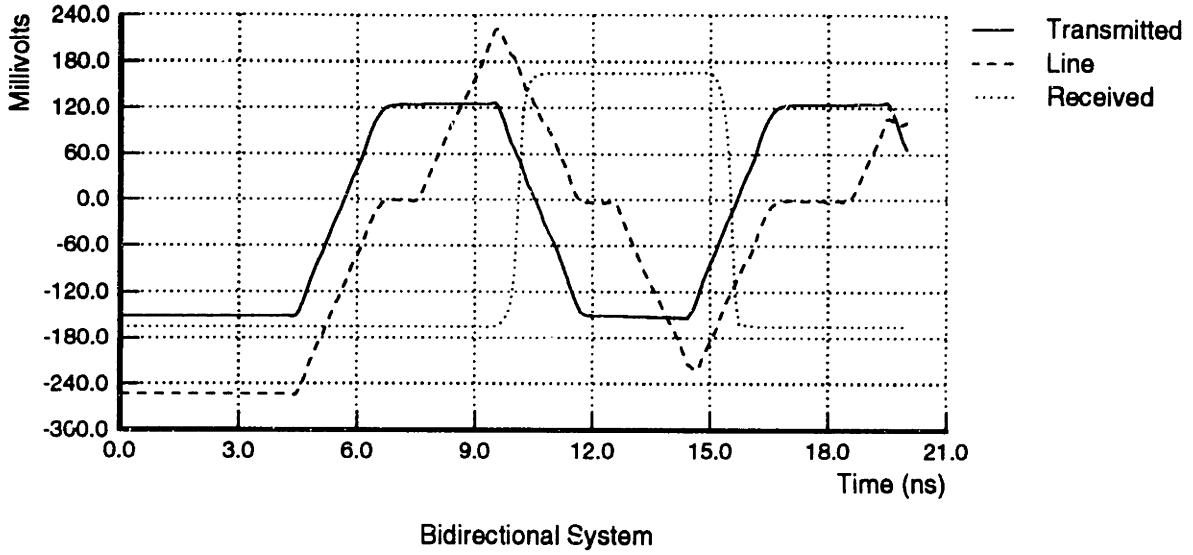


Figure 5-37: Bidirectional signalling with no interconnect parasitics. The received waveform has been scaled down by a factor of 10. When the line is below the transmitted waveform, the received data is a zero. When it is above, the data is a one.

5.4.8 Summary of Bidirectional Signalling

In the previous sections, circuit techniques were explored to implement simultaneous bidirectional signalling. In this section, these techniques are put together and tested.

Figure 5-37 shows the simulation result of a complete system. Two bidirectional transceivers are interconnected with a transmission line. The delay of the transmission line is 2ns. No interconnect parasitics are included in this simulation to show how the circuits would work under ideal conditions. The "transmitted" waveform is the transmitted reference waveform. The "line" waveform is the superposition of the two transmitted waveforms. The recovered data is shown in the "received" waveform and is scaled down in the plot by a factor of ten to match the interchip signalling levels. To avoid occasional HSpice problems[4] with the transmission line and inductor, the voltages are measured relative to VT .

One can see the transmitted waveforms from both tranceivers being added, forming the line waveform. Data is being recovered quite cleanly.

Figure 5-38 shows the simulation result of the above circuit with the parasitic circuit elements added. The waveforms are still fairly clean and the data is being recovered correctly.

[4]The initial voltage condition on the transmission line shield would be 0 volts, not VT as intended.

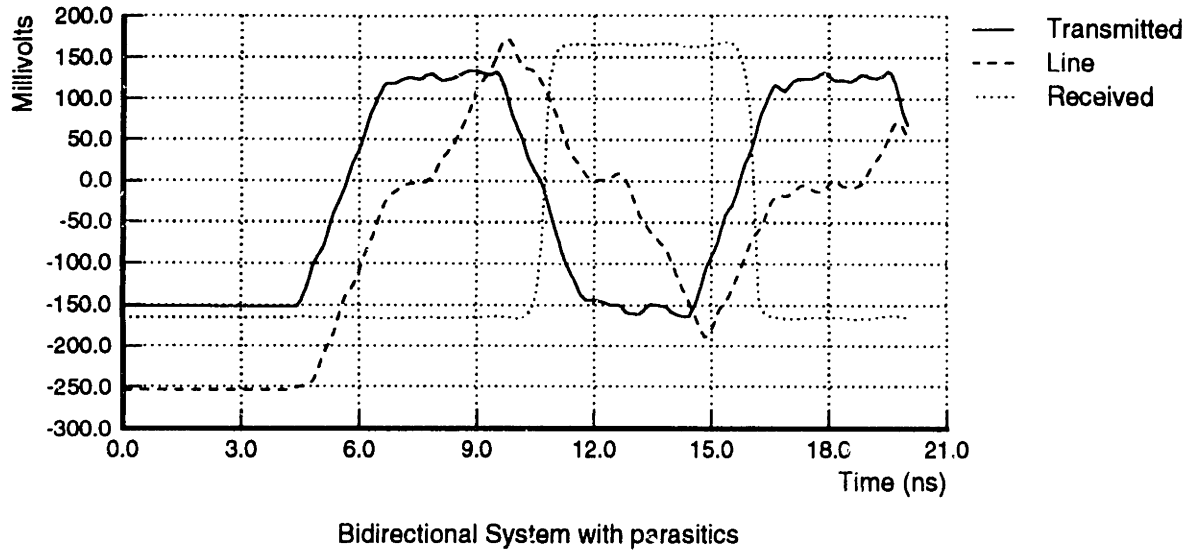


Figure 5-38: Bidirectional signalling with interconnect parasitics. [The received waveform has been scaled down by a factor of 10.

5.5 Summary

High-performance simultaneous bidirectional signalling is acheivable. Through the analysis presented in this chapter, one can see that most of the techniques described are applicable to all types of signalling, not just bidirectional signalling. The key techniques are:

- Accurate modeling of the physical interconnect. Understanding both the transient response and the frequency response of the interconnect. Preconditioning the transmitted waveform to avoid the poorer response areas of the interconnect.
- Knowing where the signal and return currents flow and keeping those paths clean.
- Steering current to avoid supply rail bounce.
- When noise is self-induced and unavoidable, inducing equal and opposite noise.

These techniques make bidirectional signalling practical, but there can be a performance difference between the peak performance of a unidirectional system and the peak one-direction performance of a bidirectional system. The difference stems from the need in a bidirectional system to accurately duplicate the transmitted waveform and the need for large signal, wide-band common mode rejection in the receiver. In today's CMOS processes and packaging technologies, the interconnect effects are the primary limiting factor. One is able to design very good receivers for the signalling frequencies of interest. The interconnect parasitics cause the line waveform to differ from the reference at the transmitter, which results in a bidirectional system having

a somewhat lower noise margin than the corresponding unidirectional system at a given signalling rate.

5.6 Schematics

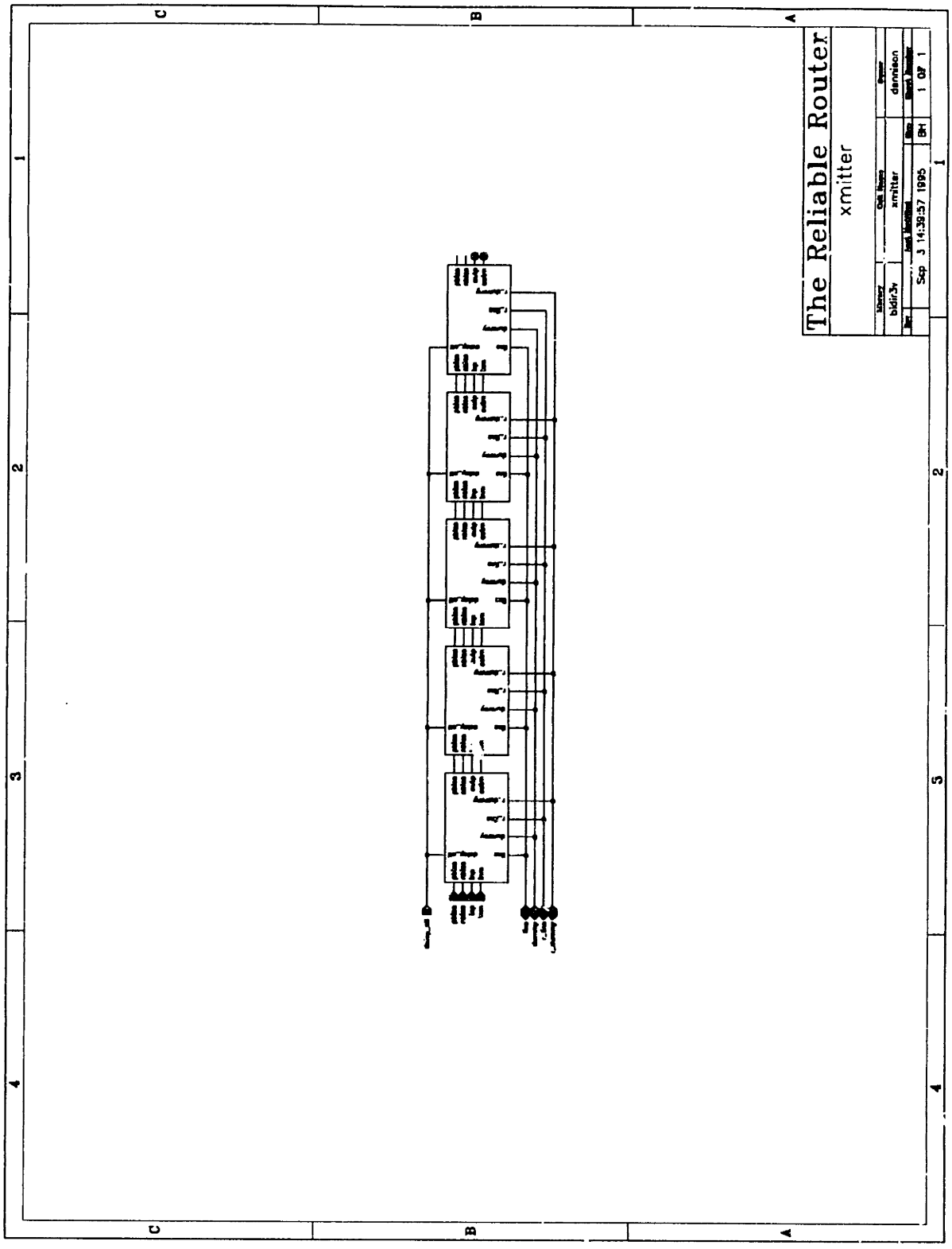


Figure 5-39: Transmitter.

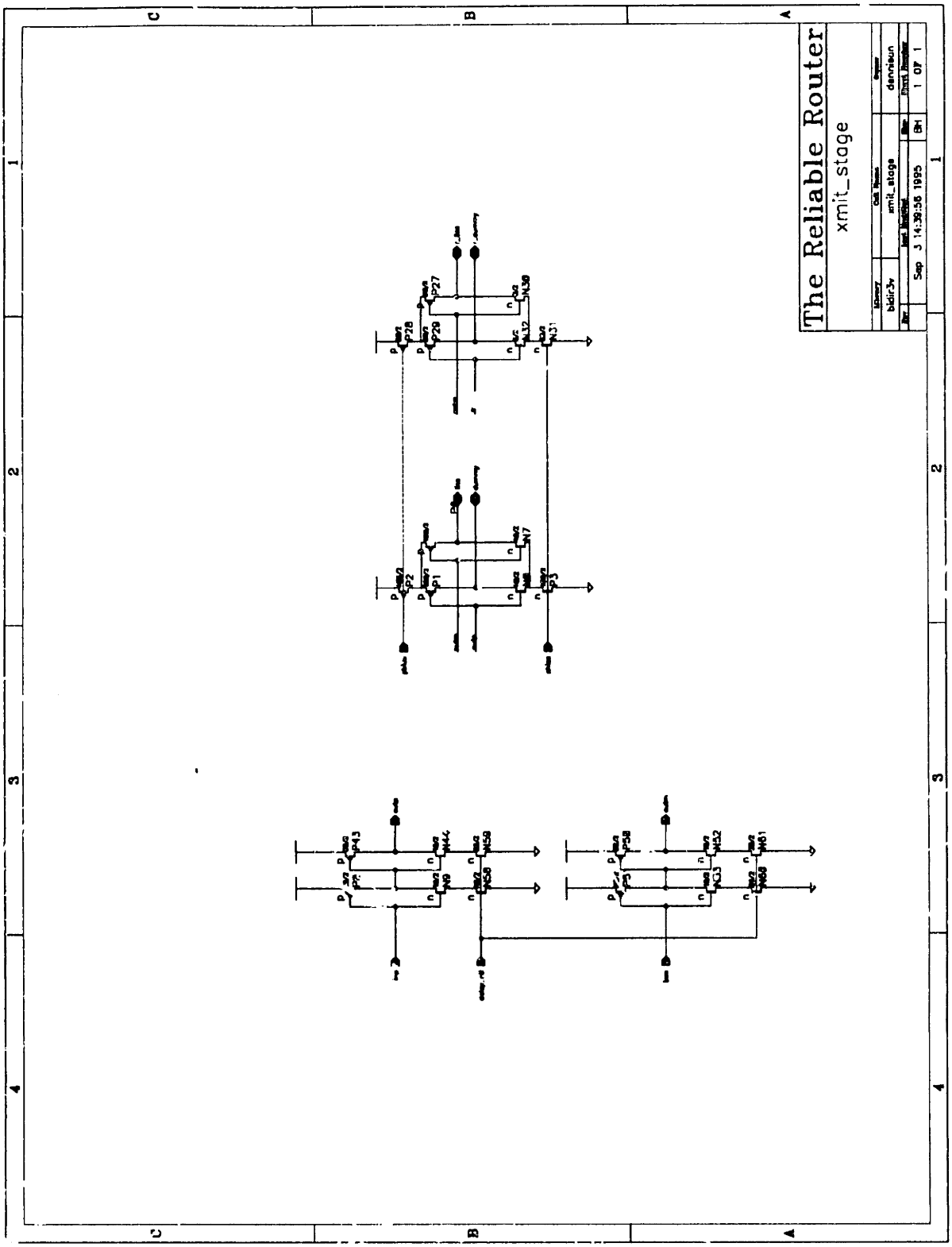


Figure 5-40: Transmitter Stage.

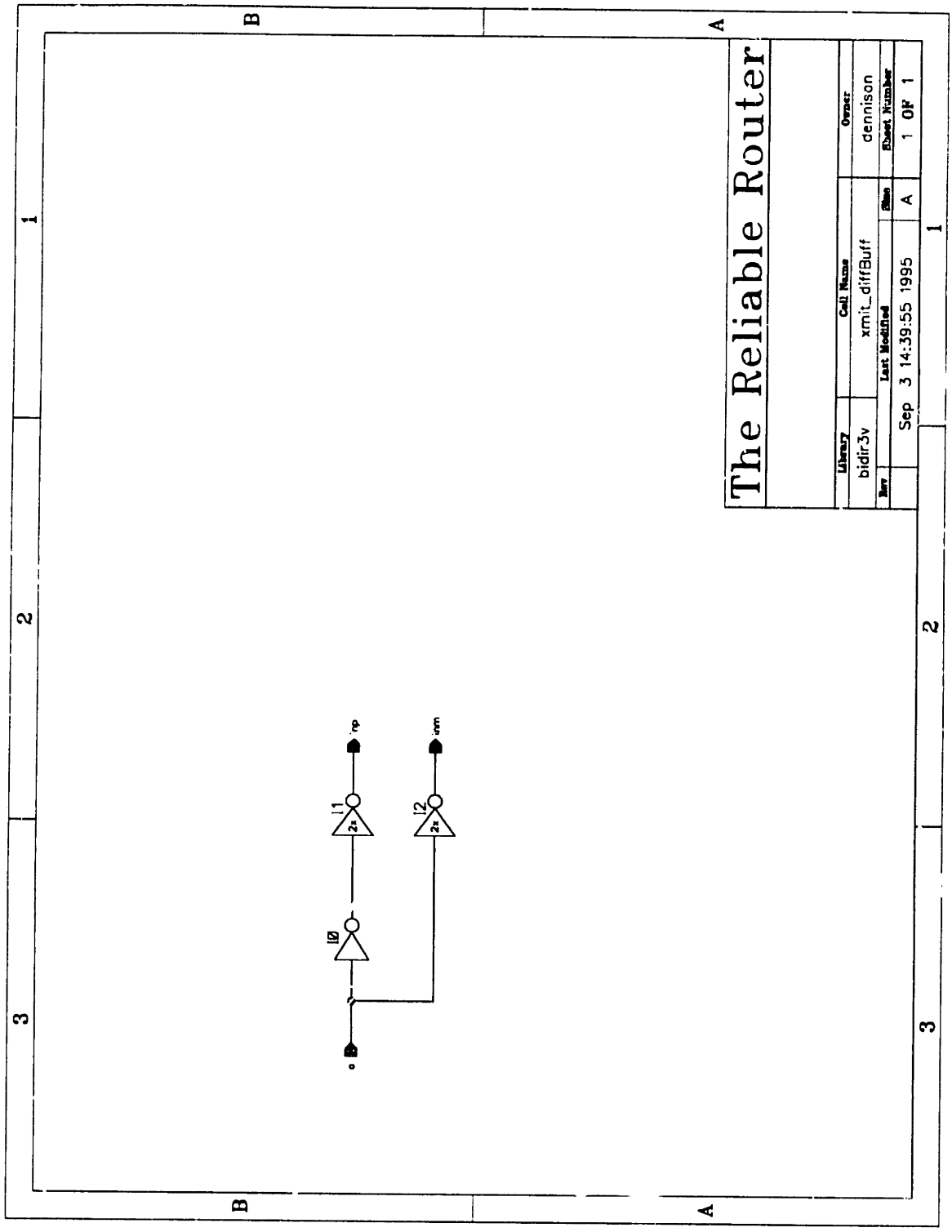


Figure 5-41: The initial differential buffer used in the transmitter.

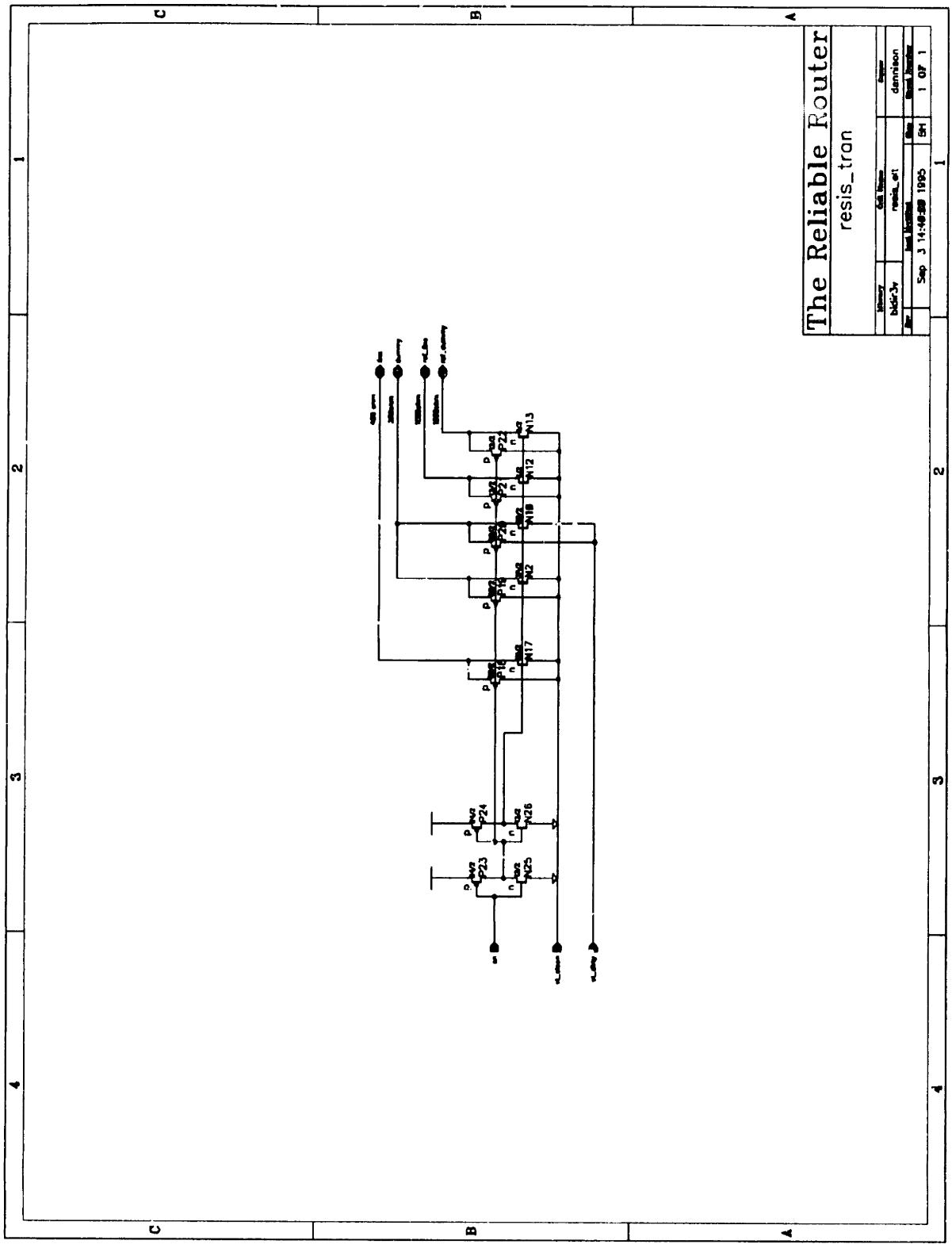


Figure 5-42: Termination Resistor.

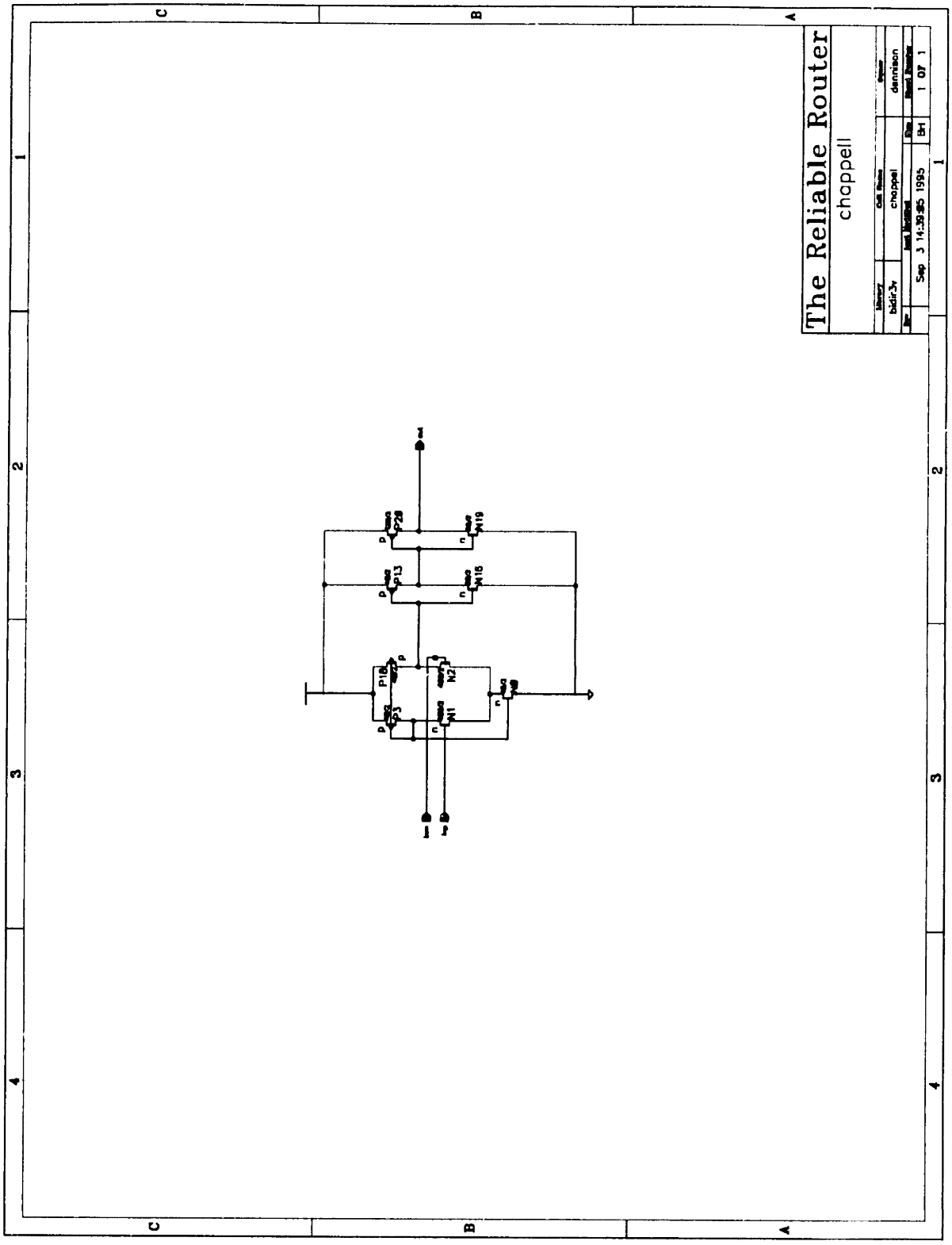


Figure 5-43: Differential Receiver.

Chapter 6

Low-Latency Plesiochronous Data Retiming

One of the many problems facing the designer of a large digital system is clock distribution. A large system is typically composed of communicating subsystems. These subsystems are internally synchronous and clocking uncertainty within a subsystem is well-controlled. Between subsystems, communications becomes more difficult, owing to larger amounts of clock uncertainty. As clock speeds increase or the number of subsystems increase, the uncertainty in the clock distribution network becomes the performance limiting factor.

In addition, the large clock tree is a single point of failure. The builder of a large system tries to avoid single points of failure wherever possible. The designer may not be trying to achieve complete fault-tolerance, but the goals of graceful degradation and hot-plug-in are very important and conflict with a single clock distribution system.

This chapter describes a new approach, *low-latency plesiochronous data retiming*, to the problem of clock distribution and data communication in large systems. The chapter begins with some background into timing methods and then develops the concept of low-latency plesiochronous data retiming. The circuits used to implement the retiming within the Reliable Router are described. Next, the latency and bandwidth reduction for a low-latency plesiochronous system are computed. The effect of timing jitter is considered and shown to cause an additional bandwidth reduction. Two extensions, cascaded retiming stages and integral subrates, are then presented.

6.1 Background

Existing solutions to the clock distribution problem include asynchronous and mesochronous (same clock source, unknown clock phase) timing methods. Messerschmitt [24] offers a good discussion of these terms. None of these methods seemed particularly well-suited for use in the reliable router. Asynchronous timing requires a synchronizer delay for each data item. Mesochronous timing involved some small amount of delay but still required a single system clock source.

A third method, plesiochronous timing, looked promising, as demonstrated in well-known embodiments such as SONET. It removed the need for a single clock and requires no flow control handshaking. The difficulty with plesiochronous timing is keeping the data items transferred one-for-one between transmitter and receiver. Dependent on the relative rates, transmitted data may be either undersampled or oversampled by the receiver. Previous designs used fairly deep FIFOs (> 4 elements!) and had complex finite state machines. In general, prior designs use FIFOs of a depth large enough to compensate for a synchronizer delay.

The solution described in this chapter separates the synchronizer from the data retiming, giving minimum data latency and minimum data storage requirements. By carefully chosing the point at which a data retiming adjustment takes place, data items are kept one-for-one without requiring any additional circuitry.

This solution does differ from other synchronizer-avoidance methods, as it addresses the issue of keeping the data items one-for-one between transmit and receive clock domains. Glasser and Rettberg [30] use dynamic delay adjustment to avoid the synchronizer penalty on the data path, but rely on mesochronous timing to keep the data one-for-one. Stewart and Ward [34] extend the idea of synchronizer avoidance to cover plesiochronous timing, but do not address the issue of keeping data one-for-one while providing minimal latency.

6.2 Plesiochronous Requirements

A pleisochronous system is one where all the clocks operate at approximately the same frequency, f_0. In point-to-point communications, there is a transmitter operating at frequency f_t and a receiver operating at frequency f_r. Both f_t and f_r are in the interval $[f_0 - \Delta f, f_0 + \Delta f]$.

If the transmitter were to send data to the receiver at the transmitter's clock rate, one of two things will eventually happen, dependent on the relative speeds of the two clocks. If the transmitter is running faster than the receiver ($f_t > f_r$), the receiver will be presented with more information than it can handle causing an overrun condition. If the receiver is running faster ($f_t < f_r$), it becomes starved resulting in an underrun condition.

To avoid underruns and overruns, the transmitter and receiver agree that the transmitter will not always send data. The transmitter simply sends data at rate lower than the receiver is operating. The receiver distinguishes between data and non-data, and only processes data. More formally, the transmitter produces a constant stream of cells, where each cell can contain either data or non-data. The rate at which the stream of cells is produced is defined to be the transmitter's frequency. The rate at which the receiver is able to consume data cells is defined to be the receiver's frequency. As long as the transmitter does not send data cells at rate faster than the receiver's consumption rate, no overrun condition will exist. If the receiver can tolerate an occasional delay in the arrival of the next data cell, no underrun condition will exist.

In a system where communication occurs by linking up several point-to-point hops,

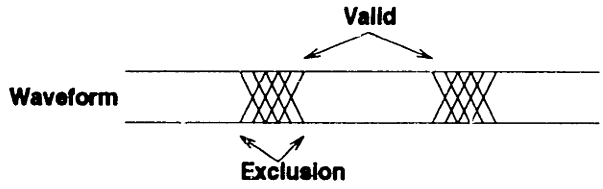


Figure 6-1: Valid and Exclusion Regions.

all transmitters in the system must produce data at a rate lower than the slowest receiver frequency. This is not as bad as it sounds, as this is usually nothing more than the worst-case transmitter/receiver frequency mismatch. Since the transmitter does not know the relative frequencies, it presumes that its clock is fast and the receiver is slow ($f_t = f_0 + \Delta f, f_r = f_0 - \Delta f$). To avoid the overrun, the transmitter sends data at the rate

$$f_d = f_t \frac{f_0 - \Delta f}{f_0 + \Delta f} \tag{6.1}$$

When $f_t > f_r$, the transmitted data rate exactly matches the receiver's consumption rate. When $f_t < f_r$, the data rate is not matched and a degradation occurs which is approximately $2\Delta f$.[1]

6.3 Retiming

Retiming is the movement of a cell from the transmitter's clock domain to the receiver's clock domain. The incoming cells have a portion of the cell time (t_{clk}) where their value is stable and can be sampled by a flip-flop using some clock edge. This is called the valid region, t_{val}. Over the rest of the cell, either the value of the cell is changing or the value has not met the setup or hold times of a flip-flip. This region is called the exclusion region, t_{exc}. By definition,

$$t_{clk} = t_{val} + t_{exc}$$

For ease of explanation, the width of the valid region is assumed to be much larger than the width of the exclusion region ($t_{val} \gg t_{exc}$). A waveform with both regions illustrated is shown in Figure 6-1

In a synchronous system, the cell can be safely retimed into the receiver's clock domain if the receiver's clock edge occurs during the valid region. The design of a correct synchronous system is guaranteeing that the clock edge occurs exactly in that region.

[1] A practical implementation will typically guarantee a a minimum rate of non-data transmissions. This is done by running a counter and stopping data transmission when the counter wraps.

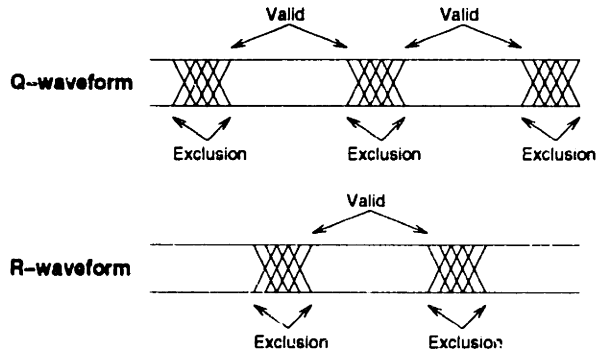


Figure 6-2: Q and R waveforms.

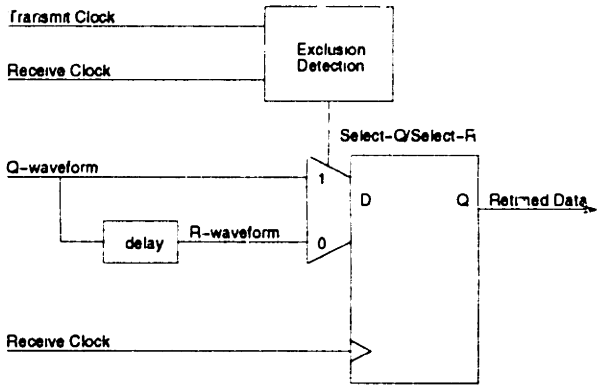


Figure 6-3: Mesochronous Retiming Circuit.

6.3.1 Mesochronous Retiming

For a mesochronous system, the placement of the receiver's clock edge relative to the valid region is not controlled. In fact, the clock edge could occur during the exclusion region. To get around this, a waveform is constructed which is simply the transmitted waveform delayed by half a clock period. For notational ease, the original transmitted waveform will be called the Q-waveform, the delayed will called the R-waveform[2].

It easy to see that if $t_{val} > t_{clk}/2$, at any point in time either the original waveform or the delayed waveform are in the valid region. Now, if the receiver clock edge occurs in the exclusion region of the Q waveform, the R waveform is sampled instead. To build a complete mesochronous retiming circuit, one needs to construct a subcircuit which delays the transmitted cell, a circuit for detecting sampling during the exclusion region, and a multiplexor. A block diagram is shown in Figure 6-3.

When $t_{val} < t_{clk}/2$, multiple delayed versions of the incoming waveform are required. In general, one needs $\lceil \frac{t_{clk}}{t_{val}} \rceil$ versions.

[2]Q because it comes from the Q output of a flip-flop in the implementation, R because it came after Q

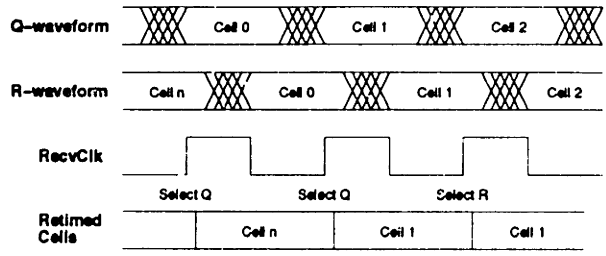


Figure 6-4: Cell Oversampling.

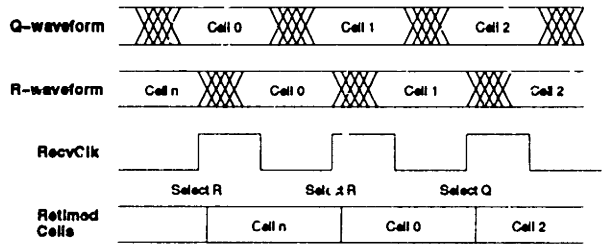


Figure 6-5: Cell Undersampling.

6.3.2 Plesiochronous Retiming

Plesiochronous retiming looks very similar to mesochronous retiming, except the relative phase Θ between the transmit and receive clocks will vary with time

$$\Theta(t) = \Theta_0 + 2\pi(f_t - f_r)t \tag{6.2}$$

To extend the mesochronous case to the plesiochronous case, one could simply allow dynamic switching between the Q and R waveforms. However, indiscriminate switching can result in either duplicate cells or missing cells. Figure 6-4 shows a case where the receiver has been sampling the Q waveform. The receive clock is running slower than the transmit clock, so the receive clock edge encounters Q's exclusion region on the right hand side. The receiver then shifts to sampling the R waveform, resulting in cell 1 appearing twice at the ouput of the receiver.

Similarly, figure 6-5 shows a case where the receiver has been sampling the R waveform. The receive clock is running fast relative to the transmit clock, so the receive clock bumps into R's exclusion region on the left hand side. The receiver then shifts to sampling the Q waveform, causing the receiver output to skip over cell 1.

6.3.3 Correct Plesiochronous Retiming

Suppose one wanted to control the switching between the Q and R waveforms using an automaton clocked in the transmit domain. The phase of the transmit clock relative to the receive clock is unknown, so the best time to flip the control input to the

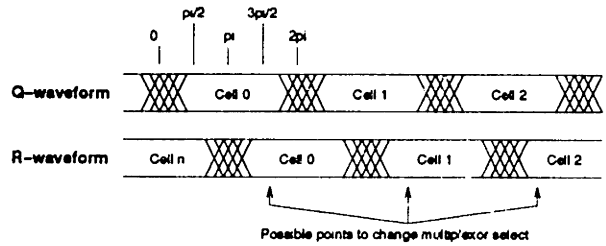


Figure 6-6: Multiplexor Control Switching Time.

multiplexor is when both Q and R are valid and have the same value. This occurs at $\frac{3\pi}{2}$ of the way into a cell, measured on the Q waveform. Figure 6-6 shows the correct point. Of course, the multiplexor should be hazard free. When the circuits are done properly, the receiver is always sampling a waveform during its valid region.

To solve undersampling/oversampling, recall that cells are either data or non-data. It is perfectly acceptable to insert or delete non-data cells, but it would be bad form to insert or delete data cells. If one restricts switching between the Q and R waveforms to the times when both Q and R contain a non-data cell, only non-data cells will be undersampled or oversampled.

There are four cases to consider:

- The receive clock is faster than the transmit clock, and the automaton switches from R to Q.
- The receive clock is faster than the transmit clock, and the automaton switches from Q to R.
- The receive clock is slower than the transmit clock, and the automaton switches from R to Q.
- The receive clock is slower than the transmit clock, and the automaton switches from Q to R.

The case of $f_r > f_t$, $R \rightarrow Q$ is shown in Figure 6-7. The fast receive clock implies that the sampling edge will encounter R's exclusion region on the right hand side. The figure shows that as the switchover is made, no cells are added or dropped. Figure 6-8 illustrates $f_r > f_t$, $Q \rightarrow R$. The fast clock implies that the sampling edge will encounter Q's exclusion region on the right hand side. The figure shows that as the switchover is made, the non-data cell is duplicated. In combination, these two cases show that when the receive clock is faster than the transmit clock, extra non-data cells are generated by the receiver to compensate.

The case of $f_r < f_t$, $R \rightarrow Q$ is shown in Figure 6-9. The fast receive clock implies that the sampling edge will encounter R's exclusion region on the left hand side. The figure shows that as the switchover is made, the non-data cell is dropped. Figure 6-10 illustrates $f_r < f_t$, $Q \rightarrow R$. The fast clock implies that the sampling edge will

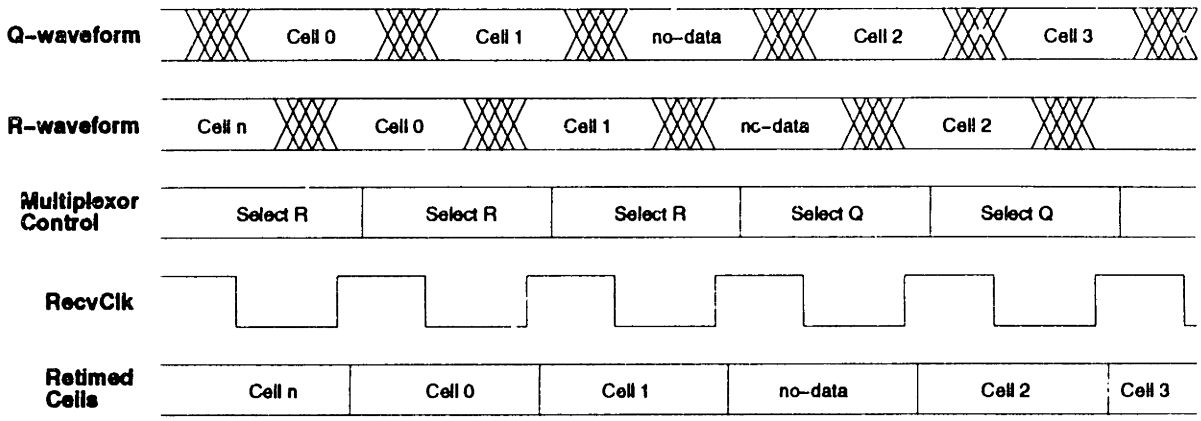


Figure 6-7: RxClk fast, $R \rightarrow Q$.

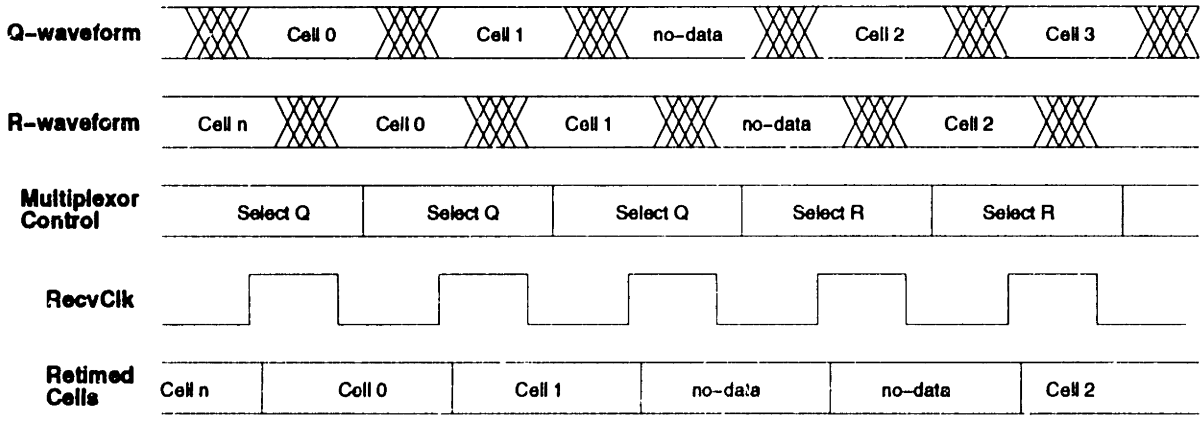


Figure 6-8: RxClk fast, $Q \rightarrow R$.

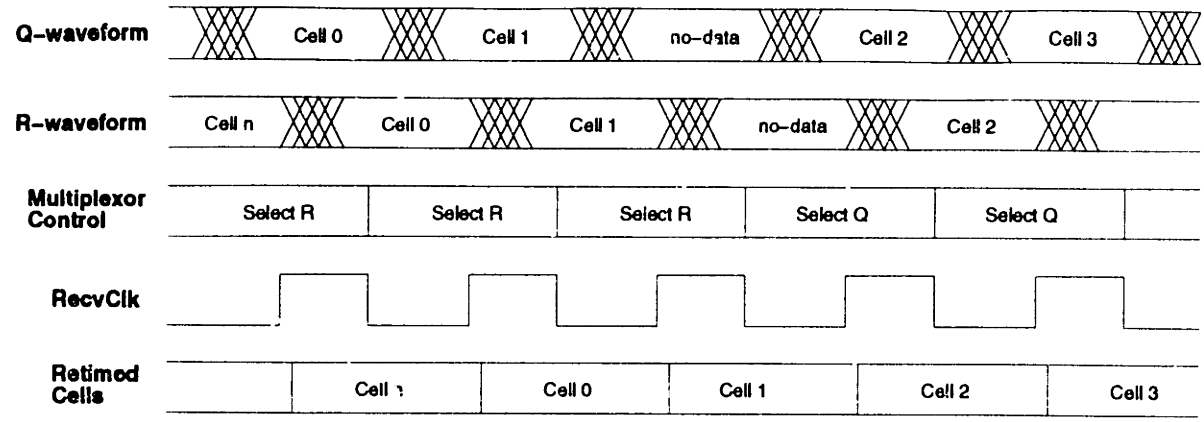


Figure 6-9: RxClk slow, $R \rightarrow Q$.

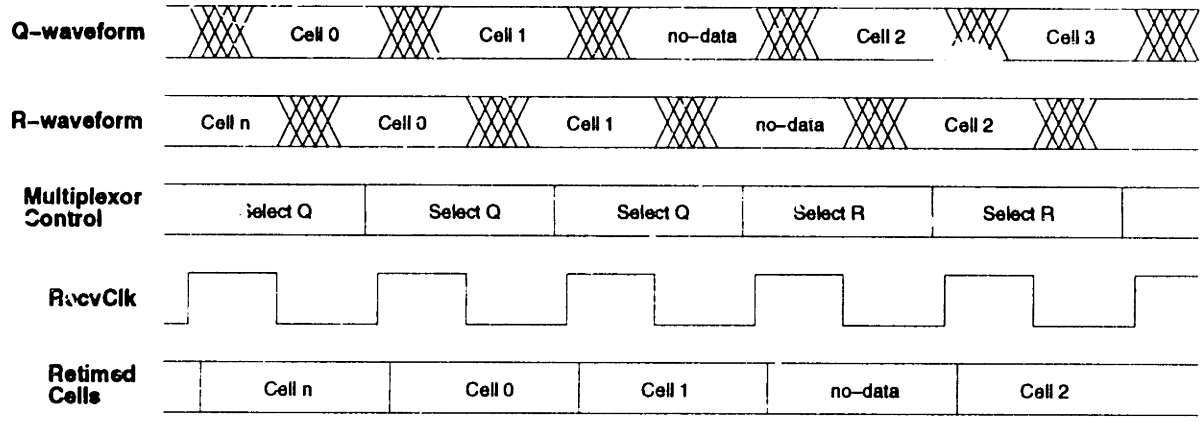


Figure 6-10: RxClk slow, $Q \rightarrow R$.

encounter Q's exclusion region on the right hand side. The figure shows that as the switchover is made, no cells are added or dropped. In combination, these two cases show that when the receive clock is slower than the transmit clock, non-data cells are deleted by the receiver to compensate.

Knowing when Q and R contain non-data implied that the automaton was operating in the transmit clock domain, hence the reason for the earlier supposition.

6.4 Circuit Pragmatics

This section describes the circuits used in the Reliable Router. There are many possible good implementations of plesiochronous communications, so this should be used as guidance only. The circuits involved are used to regain timing margin between devices, to construct the Q and R waveforms, to detect the exclusion region, and to control waveform selection.

6.4.1 Modeling Flip-Flops

Before getting into the circuit details, one needs to have a good understanding of flip-flops. Many VLSI designers pattern their mental model of flip-flops after the ones encountered in standard board-level digital designs. This is a useful model, but it can lead to rather pessimistic designs.

Board-level flip-flips are modeled using setup and hold times. The observation is that those numbers were derived for a number of different devices over process corners, for a number of different signal edge rates, using a range of power supply voltages. One often finds exclusion region widths in the 2-3ns range for TTL devices and in the 5-10ns range for CMOS devices.

However, when one characterizes an individual CMOS flip-flop using HSpice with constant voltage and constant edge rates, one finds that the width of the exclusion region is very small, on the order of of a lightly-loaded inverter delay[3]. As one varies process corners, supply voltage, and edge rate, the exclusion region will shift relative to the clock edge but the overall width will remain small.

One can use this "knife edge" sampling effect in the construction of retiming circuits. Within the area of the retiming circuit, there is little process variation (channel width, length, oxide thickness, etc.) so the flip-flips tend to be matched. The remaining factors are edge rate and power supply voltage. Since the flip-flops tend to use the same clock wire and use the same portion of the power distribution system, the edge rate and supply effects also tend to match. Thus, the location of the exclusion region will match from flip-flop to flip-flop.

One has to be careful with supply voltage, as it varies with time. If one uses techniques which measure the position of the exclusion region at one time and use that measurement at a different time, one must account for the worst case difference in supply voltage between those times.

6.4.2 Circuit Details

Please note that the unit of transfer commonly used by communication engineers is a *cell*. Cells typically do not have flow control[4]. In the parallel processing community, the unit of transfer is usually the *flow control control digit* or *flit*. Descriptions of link-level router operations are presented in terms of cells rather than flits, to emphasize that flow control is not needed.

Between reliable routers, a transmit clock is sent along with a cell. As mentioned in the retiming section, several timing events in the transmit clock domain are required per cell. Needed are:

- Q goes into valid
- Q goes into exclusion

[3]This is about 100ps in the 0.8 micron process used to fabricate the Reliable Router

[4]For example, constant bit-rate streams in ATM. The waters start to muddy with variable and available bit-rate types of traffic.

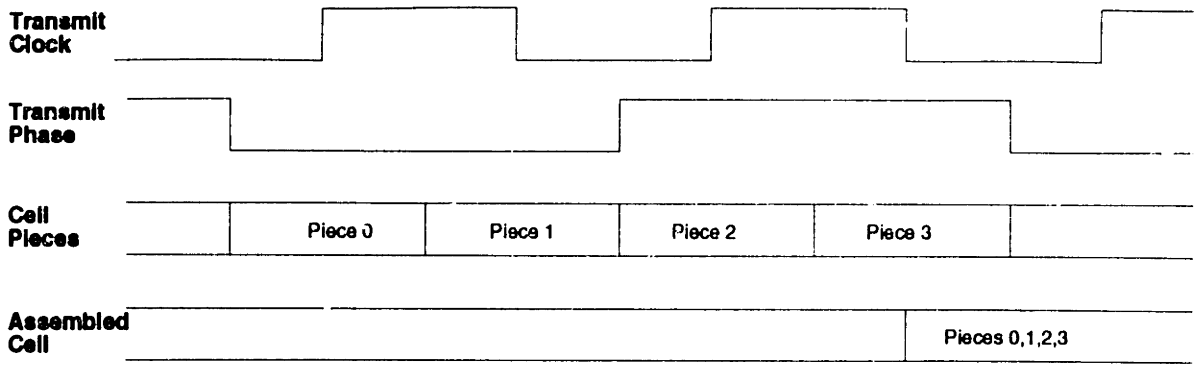


Figure 6-11: Cell reassembly.

- R goes valid
- R goes into exclusion
- Change between Q and R now.

For many reasons unrelated to plesiochronous communications, in the router a cell is broken into four smaller pieces, each transmitted on four subsequent transmit clock edges. Both positive and negative edges are used, so a cell time is two transmit clock periods (the router uses a 100MHz transmit clock). A transmit phase signal is used to distinguish piece number zero from piece number two, and piece number one from piece number three. The edge of the transmit clock indicates whether the piece is even or odd. These pieces are reassembled by the receiver into a complete cell. A sketch of the waveforms in shown in Figure 6-11.

When the cell is finally reassembled, the actual exclusion region for the Q waveform is relatively small and straddles a transmit clock edge. The other three transmit clock edges can be used to construct the R waveform and control the $Q \leftrightarrow R$ crossing point.

The discussion of the exclusion region detection was a little simplified. One does not want to wait until the receive clock is sampling in the exclusion region before deciding to change! Instead, a keep-out region which encompasses the exclusion region is constructed using transmit clock edges. The transmit clock edges thus represent:

- R goes into keep-out, Q goes into valid
- R changes
- Q goes into keep-out, R goes into valid, change mux control here.
- Q changes

Figure 6-12 shows the waveforms and their relative timing.

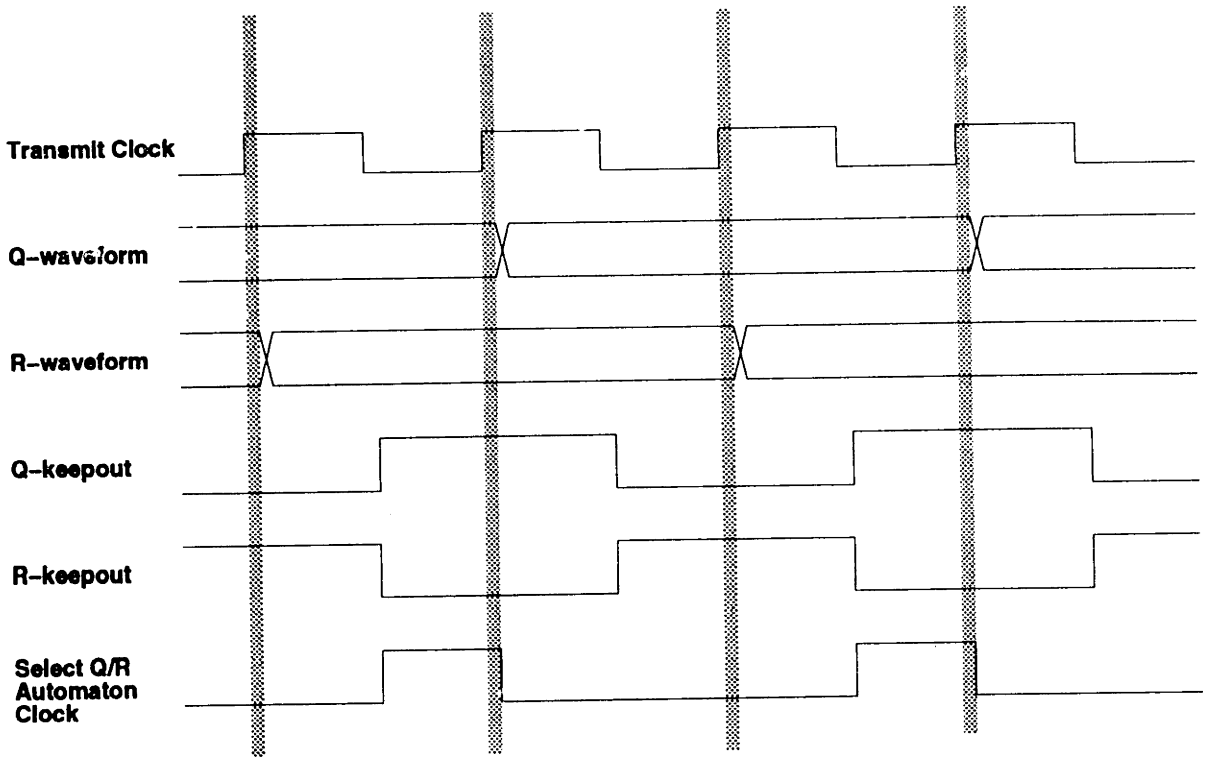


Figure 6-12: Waveforms derived from transmit clock.

To determine if a change between Q and R waveforms is required, the keep-out regions are first sampled using the receive clock. The output of these sampling flip-flops are then fed back into the transmit clock domain using a synchronizer for use by the automaton.

The entire automaton has two states: select-Q and select-R, and is shown in Figure 6-13.

Note that the synchronizer delay does not delay the movement of data. "Stale" in-keepout information is perfectly okay, as long as the relative clock drift is not too fast. Since the delay is out of the critical path, fast metastability resolution is not required of the synchronizer. In the router, 100ns are alloted to synchronizer resolution.

One of the nice features of this implementation is that no circuit tricks were required. Slowing down the clock will improve timing margins, just in case one missed a critical path.

6.5 Latency

An optimal low-latency synchronous retiming is one where the receive clock samples the transmit data at the very beginning of the valid region. The latency for such a retiming is defined to be zero.

A plesiochronous retiming circuit samples either the Q or R waveforms uniformly

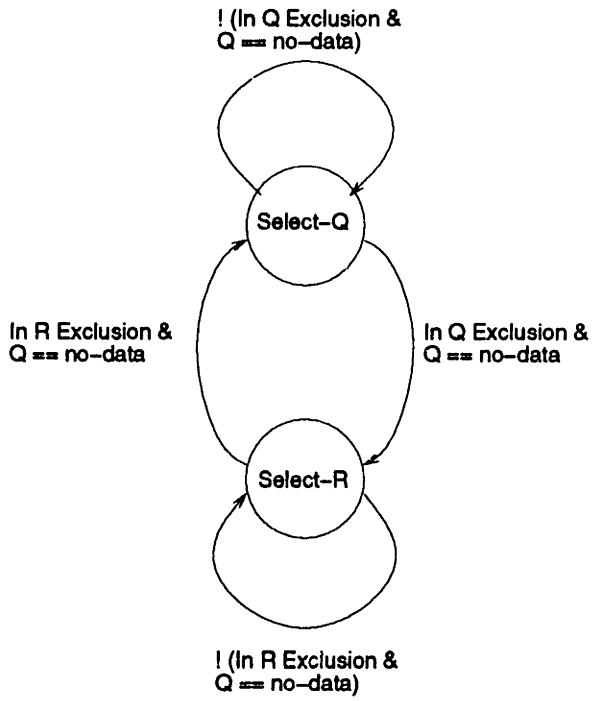


Figure 6-13: Select $Q \leftrightarrow R$ Finite State Diagram.

along their valid regions. When the width of the valid region approaches the entire cell-time, the average latency sampling the Q waveform is 0.5 cell-times. Similarly, the average latency sampling the R waveform is 1.0 cell-times. Thus, the average latency is 0.75 cell-times.

However, when the width of the valid region drops to about half a cell-time, the latency for sampling the Q waveform drops to 0.25 cell-times and the latency for sampling the R waveform drops to 0.75 cell-times. The average drops to .5 cell-times.

A *biased* waveform selection finite state machine would try to use the Q waveform whenever possible. Under these conditions, the average latency is 0.5 cell-times.

6.6 Non-Data Transmission Rate

In the previous section, a distinction between the exclusion region and the keep-out region was made. A non-data cell must be received somewhere in the time it takes the receive clock edge to drift from the edge of the keep-out region to the edge of the exclusion region. This time is the maximum amount of time allowed between transmission of non-data cells.

To begin the calculation, the time from keepout region edge to exclusion region edge, $t_{k \rightarrow e}$, is expressed as a phase angle:

$$\Psi = 2\pi t_{k \rightarrow e} f_r \tag{6.3}$$

From equation 6.2, the change in phase angle with respect to a change in time is:

$$\Delta\Theta(\Delta t) = 2\pi(f_t - f_r)\Delta t$$

For worst case f_t and f_r:

$$\Delta\Theta(\Delta t) = 4\pi\Delta f\Delta t \tag{6.4}$$

Setting 6.3 and 6.4 equal

$$4\pi\Delta f\Delta t = 2\pi t_{k\to e} f_r$$

$$\Delta t = t_{k\to e}\frac{f_r}{2\Delta f}$$

$$\Delta t = t_{k\to e}\frac{(f_0 - \Delta f)}{2\Delta f}$$

Assuming $\Delta f \ll f_0$:

$$\Delta t \approx t_{k\to e}\frac{f_0}{2\Delta f} \tag{6.5}$$

Converting into transmit cell times:

$$\#\text{ cell times} \approx t_{k\to e}\frac{f_0}{2\Delta f} f_t$$

$$\#\text{ cell times} \approx t_{k\to e}\frac{{f_0}^2}{2\Delta f} \tag{6.6}$$

For example, suppose that the keep-out region exceeds the exclusion region by 2ns, the base frequency is 50MHz (i.e. 50 million cells per second), and that the accuracy is ± 5KHz (100 ppm).

$$\frac{2e\text{-}9 \times 50e6 \times 50e6}{2 \times 5e3} = 500 \text{ cell times}$$

In this example, 0.2% of the available bandwidth must be given over to non-data.

6.6.1 Best Case Non-Data Transmission Rate

For an ideal circuit, the width of the exclusion region is 0 and the keepout exactly straddles it (Figure 6-14). The amount the keepout region exceeds the exclusion is then 1/4 of a cell period. Expressed as a phase angle:

$$\psi = \frac{\pi}{2}$$

Setting this equal to equation 6.4:

$$\frac{\pi}{2} = 4\pi\Delta f\Delta t$$

$$\Delta t = \frac{1}{8\Delta f}$$

All plesiochronous retiming methods require sending non-data to prevent queue overflows. These must be inserted at a rate of $2\Delta f$. Even when using ideal circuits, plesiochronous data retiming requires non-data cell transmission rate 4 times higher, $8\Delta f$.

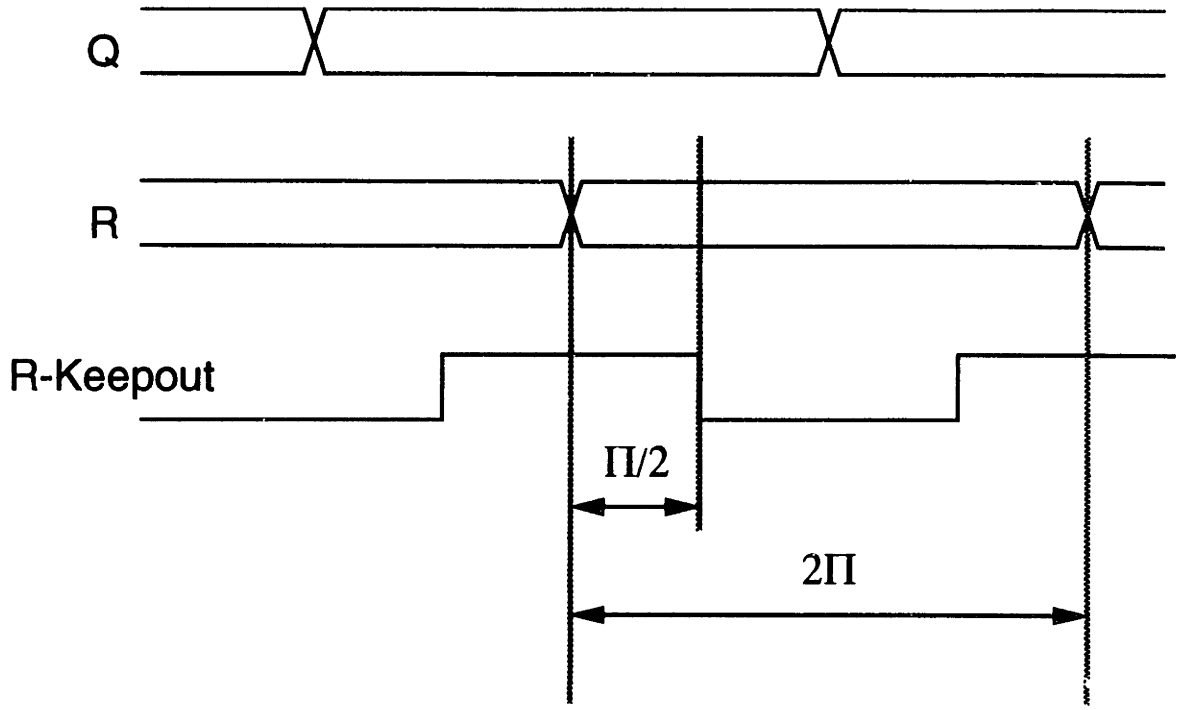


Figure 6-14: Ideal Waveforms.

6.6.2 The Effect of Jitter

No electrical signal is without some noise component. With edge times being non-zero, all signals will have some amount of timing jitter which will reduce timing margins. Plesiochronous retiming relies upon timing relationships between transmitter and receiver being stable for many clock periods. However, plesiochronous retiming will work when both transmitter and receiver frequencies are slowly varying within their limits. Plesiochronous retiming works by adapting to the phase angle differences between the transmitter and the receiver. Slowly varying the frequency amounts to altering the phase angles. As long as the change does not happen too quickly, the system will track the changes.

For example, one could be moving data over a many-kilometer-long coaxial cable hung on some telephone poles. Over the course of a day, the heating and cooling will change the physical length of the cable, resulting in the receiver seeing lower frequencies when the cable is heating, higher frequencies when the cable is cooling. The bandwidth of the noise is very low and should be easily tracked.

When the jitter is at a high frequency, the system cannot track it and instead must reject it by accounting for it in the system timing margins. High-frequency jitter can affect a plesiochronous system in two ways:

- the jitter of the data currently being moved from the transmit to receive clock domains, and
- the validity of the use_q/use_r signal.

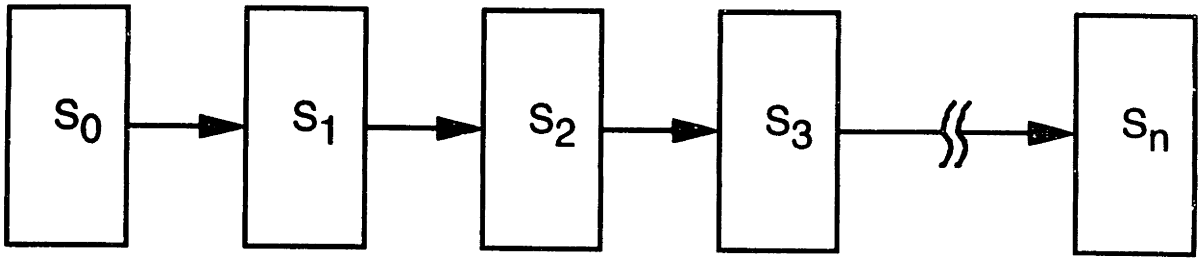


Figure 6-15: Cascaded Retiming Stages.

The jitter on the data can be modeled as decreasing the time from the exclusion region to the keepout region by the amount of the jitter. Replacing $t_{k\rightarrow e}$ with $(t_{k\rightarrow e} - t_{jitter})$ in equation 6.5 yields:

$$\#\text{ cell times} \approx (t_{k\rightarrow e} - t_{jitter})\frac{{f_0}^2}{2\Delta f} \tag{6.7}$$

This causes an increase in the amount of non-data transmissions. Similarily, jitter can cause an error in the use_q/use_r signal by shifting the keepout signal slightly. What can happen is that this error takes place at the time when a non-data cell arrives. The nominal position of the keepout would cause the waveform select state machine to change states, but the actual jittered version left the FSM in its current state. The sampling edge must drift solidly into the keepout by the amount of the jitter before it is certain the state machine machine is in the correct state. Again this can be modeled by further reducing the time from an edge of keepout region to an edge of the exclusion region.

$$\#\text{ cell times} \approx (t_{k\rightarrow e} - 2t_{jitter})\frac{{f_0}^2}{2\Delta f} \tag{6.8}$$

To summarize, jitter not only increases the latency of a system by decreasing the eye opening, it will also necessitate a reduction in the data cell rate.

6.7 Cascading Plesiochronous Retiming Stages

One of the nice features of plesiochronous retiming is that it can move data from transmitter to receiver "open-loop" - all signals travel from transmitter to receiver. Plesiochronous retiming can be extended to allow cascading many stages as shown in Figure 6-15.

6.7.1 The Need for Local Timing Requirements

When several stages are cascaded, each section must ensure that the local plesiochronous timing requirements are met. For example, the stage S_1 transmitter must guarantee that there are enough non-data cell transmissions to allow the clock adjustments in

the S_2 receiver to take place. These local requirements commit each stage to send cells at a rate slightly higher than the data-cell rate.

Consider a long string of m nodes. Let S_0 be operating at the maximum rate, rmax. Let nodes S_n (for $1 < n < m$) be operating at rate $rmax(1-\epsilon)^n$. That is, each node is very nearly at the maximum rate, but each one is just slightly slower then the node before it. Let node S_m be operating at very, very near the minimum rate, $rmin(1+\epsilon)$.

Observe that all nodes (> 0) consume non-data items, none generate them, as all links go from fast to slow.

Suppose that S_{m-1} is not locally enforcing the insertion of non-data items, and is instead relying on S_0 for their generation. Then, one could construct an phase alignment case using nodes $1 \ldots m-1$ such that the series of nodes consumed the next $m-1$ non-data items. This would result in node S_m not receiving a non-data item for a very long time, causing the retiming to fail.

Therefore, node S_{m-1} must be locally enforcing the plesiochronous interarrival requirement. Since the system has no global knowledge (it is all unidirectional communication), all nodes must be enforcing the timing constraint.

Now add a node S_{m+1} to the end of the chain, operating at the minimum rate, rmin. Node S_m is enforcing the plesiochronous interarrival time for non-data items. Node S_m is the bottleneck stage. Therefore, S_0 must send sufficient non-data for its local requirement, and additional non-data to prevent queue overflows at node S_{m+1}.

6.7.2 The Source Data Rate

In a cascaded system, one of the stages will have a physical transport frequency lower than all of the other stages. This stage will be the data transport bottle-neck. It can send data cells no faster than the transport rate, derated by the insertion of non-data cells. Let M be the maximum number of data cells allowed to be transmitted between non-data cells. The slowest transport frequency is $f_0 - \Delta f$. Then, the maximum sustainable data-cell rate is:

$$\frac{M}{M+1}(f_0 - \Delta f) \tag{6.9}$$

To prevent a queue overflow in the bottle-neck stage, the transmitter in stage S_0 must send data cells at a rate less than $\frac{M}{M+1}(f_0 - \Delta f)$. Since the S_0 transmitter does not know whether it is operating slow or fast, it must presume that it is operating fast. The number of cells between non-data transmissions is then:

$$M\frac{f_0 - \Delta f}{f_0 + \Delta f} \tag{6.10}$$

Note that $\frac{(f_0-\Delta f)}{(f_0+\Delta f)}$ term came from the worst-case timing mismatch. From the discussion of best-case non-data insertion:

$$\frac{M}{M+1} \leq 4\frac{f_0 - \Delta f}{f_0 + \Delta f} \tag{6.11}$$

Therefore, cascaded plesiochronous data retiming suffers at most a 25% additional degradation in data bandwidth.

6.8 Integral Subrate Extensions

The method may be extended to allow retiming between clock domains where the frequency of one domain is an integral multiple of the other domain. That is, either $f_t \approx i \times f_r$ or $f_r \approx i \times f_t$ must hold.

When $f_r \approx i \times f_t$, the circuit changes are fairly minor. The receiver must sample both the keep-out windows and the incoming Q or R cell once every i periods. For clock periods when the incoming cell is not sampled, the receiver creates a non-data cell.

If $f_t \approx i \times f_r$, the transmitter must drastically reduce its sending of data to one of i cells. Note that it still must occasionally insert non-data cells to meet the plesiochronous requirements.

This technique is used in the Reliable Router to allow slower-speed processors to interface easily to a higher speed router.

6.9 A Comparison to Mesochronous Techniques

Mesochronous is defined as having the same frequency, but an unknown phase relationship. A stronger definition of mesochronous retiming is needed, which is that the cells going into the retiming circuit are one-for-one with the cells leaving the retiming circuit (ignoring the latency through the circuit). This will called a *pure mesochronous circuit.*

In most systems, the phase angle between the clock domains will vary as a function of time. If the phase angle varies significantly, it should be tracked. In general, this implies the use of a slack buffer or FIFO to make up for the variation in cell delay. The slack buffer must be able to handle both minimum and maximum delays. As result, pure mesochronous retiming circuits will have an additional latency penalty. The penalty depends on the actual distribution of the delay function, but for most circuits is about one-half the overall delay variability. Note that this penalty is paid on top of the mesochronous phase-alignment delay. Therefore, for systems with any phase variability, plesiochronous retiming will offer lower latency.

Of course, a plesiochronous solution works only if the cell stream can be partitioned into data and non-data. This comes at the expense of a small amount of bandwidth and some additional system complexity in the transmitter. A mesochronous transmitter can send anything at any time. The plesiochronous transmitter must be able to occasionally stall in order to send non-data.

The receiver circuits for plesiochronous retiming will tend to be less complex. It was shown that the control circuit for plesiochronous retiming is fairly simple. Mesochronous retiming still needs to determine the relative phase of the incoming data. However, it may also have the added complexity of the slack buffer.

In general, pure mesochronous retiming should be used when it not possible to insert non-data items or when the delay variability is low. Otherwise, the plesiochronous technique outlined in this chapter should be used, even when the system is actually mesochronous.

6.10 Summary

A new technique for plesiochonous data retiming has been described. This technique offers latencies on the order of a fraction of a cell-time as well as modest implementation requirements. These achievements were gained by moving the synchronizer out of the data path and carefully choosing the time to make a phase adjustment.

In addition, the technique allows true unidirectional retiming. The transmitter can send information to a receiver without any flow control information sent back to the transmitter from the receiver.

Extensions include the ability to handle integral subrates and to cascade several stages. Subrates should prove to be useful in the design of systems where multiple clock rates are present. Cascading is useful in systems requiring repeaters.

Chapter 7

The Microarchitecture of the Reliable Router

This chapter describes the overall microarchitecture of the router. It is organized into the following sections: top-level description, clocking and pipelines, global signals, and a detailed description of all modules.

The schematic description of the Reliable Router uses 288 pages, of which 72 pages correspond to the standard cells. At 216 pages, there are simply too many schematics to include in this chapter[1]. To cover all of the material, the subsequent descriptions are purposefully abstract and omit many of the circuit details and logic equations.

7.1 Top Level Organization

Most router block diagrams resemble the one shown in Figure 7-1. There is a crossbar switch in the middle, with some number of input controllers on the left and a corresponding number of output controllers on the right.

The organization of the reliable router is similar and is shown in Figure 7-2. The I/O pads and support logic have been made explicit. Input and output controllers have been grouped together into a *port* module. The reliable router transmits a moderate amount of control information between input controllers. That information has to flow out of the output controller, so the input and output controllers are more tightly coupled.

The reliable router does not have a crossbar per se. All the data lines from all input controllers are bussed to all output controllers. The selector function is carried out in the output controller itself. The actual output-controller arbitration functions are stuffed unceremoniously in a corner of the input controller. None the less, there is logically a crossbar function which is referenced in subsequent logic descriptions.

The entire list of top-level modules is given in Table 7.1. Many of the top-level modules are quite simple and are at the top-level to make the layout and validation processes easier. The hierarchy is mostly natural, but there are exceptions. Detailed module descriptions will be given in Section 7.4.

[1] All of the schematics may be found in Appendix A.

Module	*Description*
core	All six of the router's ports.
leftSide	I/O pads on the "left" side of the chip. Corresponds to the processor input pads.
rightSide	I/O pads on the "right" side of the chip. Corresponds to the processor output pads, JTAG boundary scan pads, and clock input pads.
pr_clk_buffer	The clock buffer for the processor clock.
tap_ctl	The JTAG tap controller.
instructionReg	The JTAG instruction register.
mask_gen	Generates a random number for the crossbar arbitration.
win_enable	Occasionally turns off the crossbar to satisfy plesiochronous contraints.
xbar_to_proc	Handles the case where the processor's clock is slower than the router's clock.
clocks_final	Derives the 50MHz clock from the 100MHz input clock.
clk_buffer	Buffers the 100MHz clock.
clk50_buffer	Buffers the 50MHz clock.
Reset_buffer	Buffers the reset signal.
hs_port	A complete set of bidirectional I/O pads for a port.
bias_blk	Supplies the various analog voltages to the birectional I/O pads.

Table 7.1: Top-level modules found in the router.

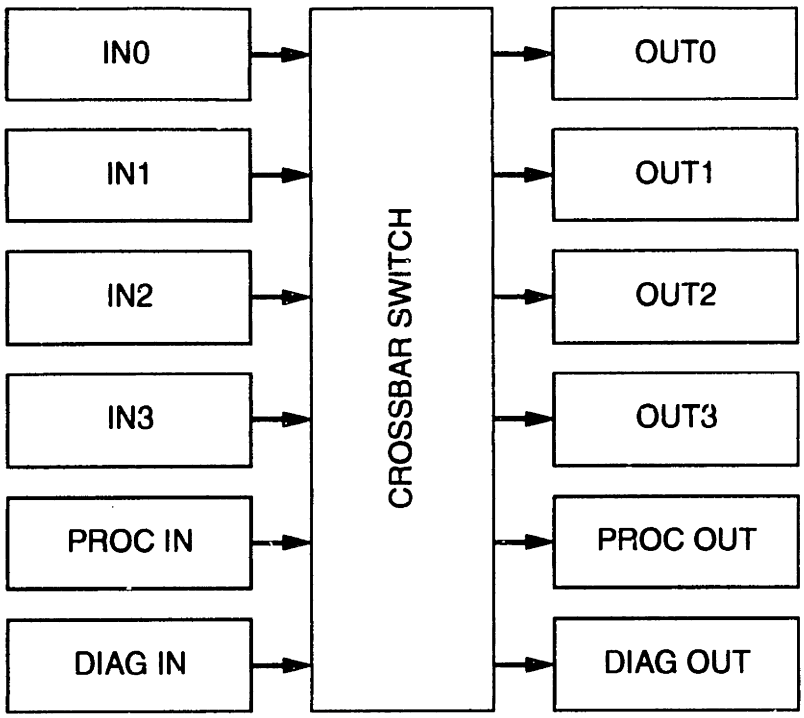


Figure 7-1: Top-level block diagram of a router.

7.2 Clocking and Pipelines

The router requires a 100MHz clock to operate at full speed. However, based upon several factors, it was decided to operate most of the internal logic at 50MHz. The reasons included that the input queues were most naturally a full flit wide (74 bits) as they were storing a total of 90 flits. Further, the control was much simplified when the basic clock rate matched the overall flit processing rate.

To avoid timing problems between the 100MHz and 50MHz clock domains, the edges of the 50Mhz clock occur just *after* the positive edge of the 100MHz clock. In particular, it is easy to go from the from the 50MHz domain to the 100MHz domains[2]. Figure 7-3 shows the relationship between the two clocks.

The clocking methodology varies from module to module. All control and state changes on the positive edge of `clk50`. Most datapath modules make extensive use of latches.

The basic positive-edge triggered flip-flop is shown in Figure 7-4. The static form is derived by added inverters with weak feedback. This flip-flop has the advantage of being a NORA structure. As long as the clocks edges are sharp and the signal between flip-flops non-inverting, the flip-flop will tolerate clock-skew between the positive and negative clocks.

Communication between routers is done plesiochronously. There are two distinct timing pipelines in the router, corresponding to the different clock domains. One

[2]This occurs when sending a flit from the input controller to the output controller

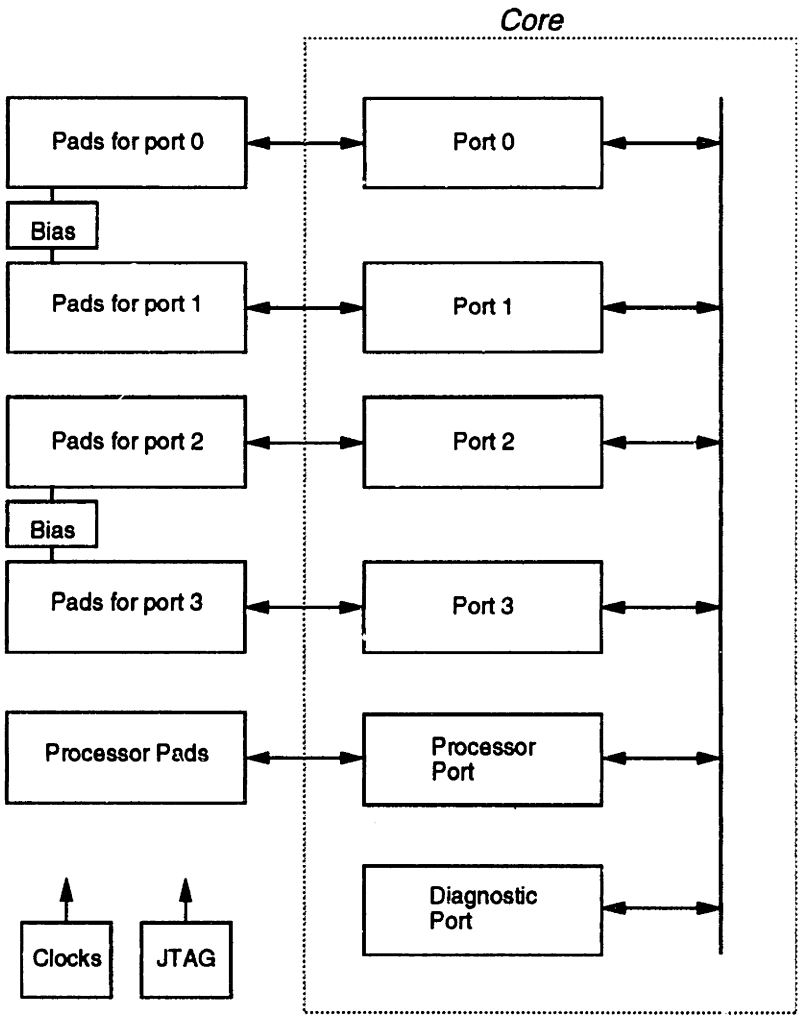


Figure 7-2: Top-level block diagram of the reliable router.

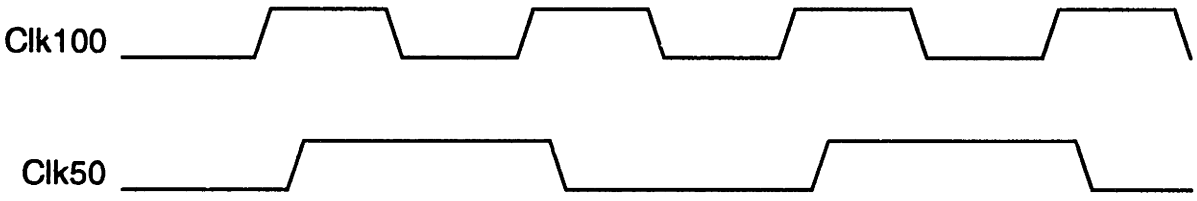


Figure 7-3: The relationship between clk and clk50. An edge of clk50 occurs about 1-2ns after the positive edge of clk.

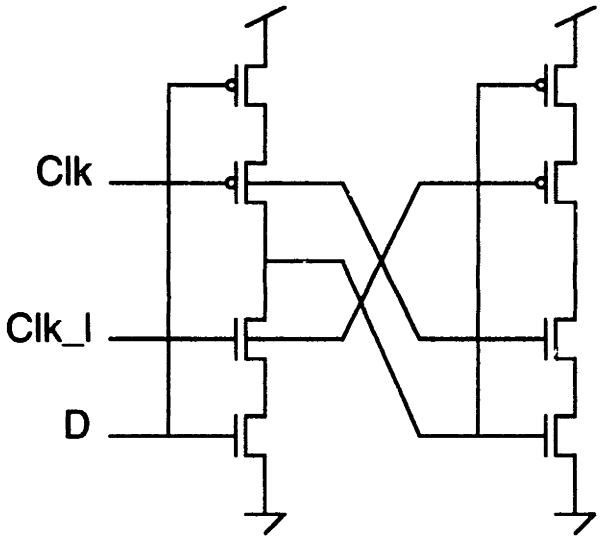


Figure 7-4: The dynamic flip-flop used throughout the router.

pipeline is the the **frontend** module of the input controller, the other pipeline corresponds to the remainder of the flit forwarding.

The **frontend** module reassembles a *frame* from four 23-bit *subframes.* The frontend pipeline is shown in Figure 7-5. Four transmit clock edges are used to reassemble the 23-bit subframes into a frame. An additional transmit clock period is used to check parity on the subframes. If any subframe fails parity checking, the entire frame is *squashed.* Additional delay occurs due the generation of the Q and R signals for plesiochronous retiming. Total delay is an average of 4 transmit clock periods, or about 40ns.

The second pipeline begins when the frame is transferred into the router's `clk50` timing domain and is shown in Figure 7-6. One `clk50` period is used to handle all of the control functions such as routing, resolving input virtual channel contention, and resolving crossbar contention. A second `clk50` period is used to transfer the flit across the crossbar. The flit is moved in two pieces, one on each half `clk50` cycle. The flit movement and outgoing parity computation takes place at the same time. Finally, the outgoing half-frames are again multiplexed into subframes for transmission. Overall delay through this portion of the pipeline is about 3 `clk` periods, or about 30ns.

7.3 Global Signals

This section presents most of the top-level signals used in the reliable router. The descriptions are intended to give the reader a feeling of the communications between major modules and are therefore organized by functional group.

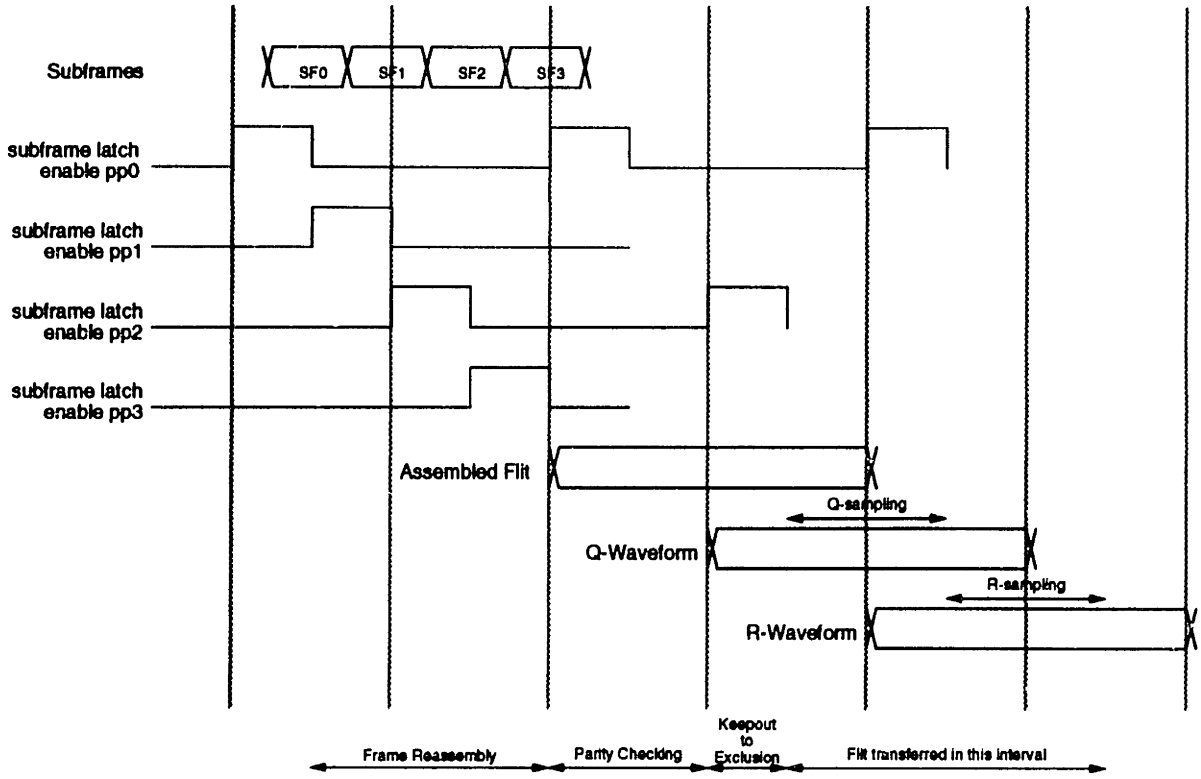


Figure 7-5: The timing pipeline through the frontend.

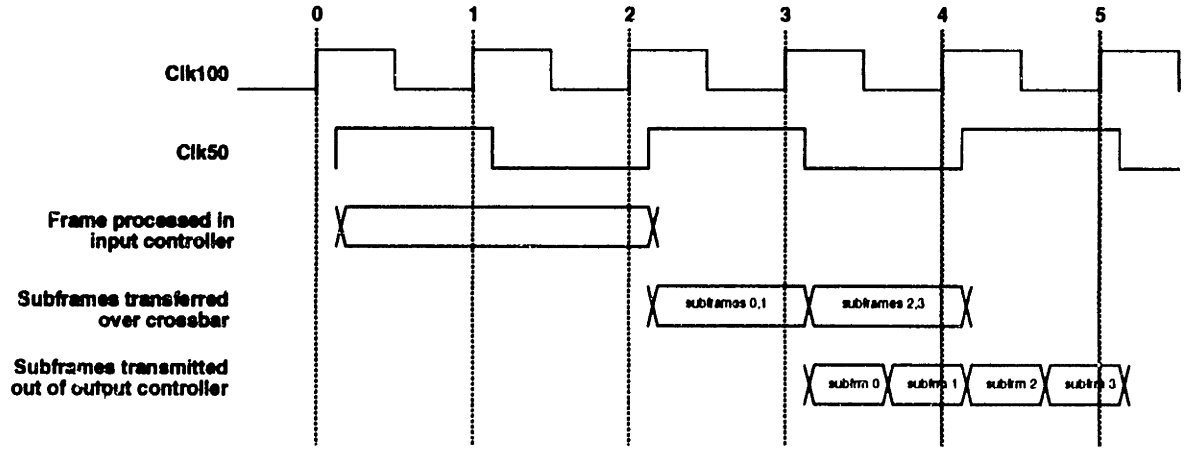


Figure 7-6: The timing pipeline through the input controller, crossbar, and output controller.

Signal	*Source*	*Description*
`clk`	clocks	The 100MHz clock
`clk50`	clocks	The 50MHz cloc
`reset`	clocks	Global reset, active high

Table 7.2: General top-level signals.

Signal	*Source*	*Description*
`ad_or_do`	JTAG Ctlr	Adaptive or dimension-ordered
`x_addr_minus`	JTAG ctlr	The arithmetic negative of the X-address
`y_addr_minus`	JTAG ctlr	The arithmetic negative of the Y-address
`port_status`[3:0]	Input ctlr	The state of each of the ports
`vc_free`[29:0]	Output ctlr	The state of each virtual channel

Table 7.3: Top-level signals used in routing.

7.3.1 General Signals

There are three general signals listed in Table 7.3. Clk and clk50 were covered in the preceding section. Reset is the global asynchronous reset signal.

7.3.2 Routing Signals

The routing logic requires information beyond that found in the header of the message. It needs to know the X and Y address of this router, which of the standard ports are connected and up, and the allocation state of all virtual channels. Further, it is possible to specify that either minimally-adaptive or dimension-ordered routing is be used. The set of routing signals is shown in Table 7.3.

The `vc_free` signals are maintained by the output controllers. A `vc_free` signal is cleared whenever a flit is sent across the crossbar on that particular virtual channel. It is set when the output controller received a "copied-token" indication from the upstream router.

7.3.3 Flow Control Signals

Virtual channel flow control is one of the more complicated aspects of the reliable router. The standard ports (X+,X-,Y+,Y-) use permit-based flow control. As a result, there is a backward flow of permits along the circuit created by the forward movement of the message. This circuit logically connects output virtual channels back to input channels. The signals used are given in Table 7.4.

Signal	*Source*	*Description*
`gbl_copied[29:0]`	Output ctlr	Global "data flit copied" bus
`gbl_freed[29:0]`	Output ctlr	Global "data flit freed" bus
`diag_cts`	Diag port	Clear-to-send to the diagnostic port
`proc_cts`	Proc port	Clear-to-send to the diagnostic port
`xb_to_pr_ok`	Clocks	Crossbar to processor processor "Okay"

Table 7.4: Global signals used in flow control.

The explicit knowledge of the circuit connection is kept in the input controller. As a result, when output controllers receive notification of a flow-control event, they simply broadcast the event on either of the `gbl_copied` or `gbl_freed` busses. It is the responsibility of the input controller to monitor the correct bit on these busses.

Consider an input controller in router R2 which has just sent a data flit to the next router (R3). It tells the previous router's (R1) output controller that it has just copied a data flit. This flow-control event is broadcast throughout router R1 on one of the `gbl_copied` signals. One of the input controllers within R1 has been monitoring that particular `gbl_copied` signal. At this point, routers R1, R2, and R3 all have copies of the same flit, so the input controller in R1 can free that flit's storage. However, the permits for this circuit are kept in router R0, not R1. The input controller in R1 therefore sends a `freed` signal back to router R0. The output controller in R0 broadcasts this freed flow-control event on one of the `gbl_freed` signals. Again, one the input controllers in R0 has been monitoring that particular signal. When the event occurs, the input controller increments its supply of permits.

The diagnostic and processor ports use a simpler clear-to-send mechanism. In addition, the processor port may be operating at a slower clock rate than the router. The `xb_to_proc_ok` qualifies the clock periods in which it is safe to send data to the processor port.

7.3.4 Crossbar Signals

The data and control signals make up the "datapath" from input to output controller and are shown in Table 7.5 and Table 7.6. Datapath signals change on both edges of `clk50`.

Crossbar allocation is mostly straightforward and occurs in the `clk50` period prior to data movement. Each input controller wishing to send data to a particular output controller sends a *bid* to that output controller. The output controller examines all of the bids and determines a winner.

The bids have three levels of priority. Levels 0 and 1 correspond to message levels 0 and 1. Level 2 is the *bumped* priority. When a particular flit is unable to advance across the crossbar for seven consecutive attempts, its priority is bumped up to level

Signal	*Source*	*Description*
d0[35:0]	Port 0	Port 0 data
d1[35:0]	Port 1	Port 1 data
d2[35:0]	Port 2	Port 2 data
d3[35:0]	Port 3	Port 3 data
d4[35:0]	Port 4	Port 4 data
d5[35:0]	Port 5	Port 5 data
ctl0[3:0]	Port 0	Port 0 control
ctl1[3:0]	Port 1	Port 1 control
ctl2[3:0]	Port 2	Port 2 control
ctl3[3:0]	Port 3	Port 3 control
ctl4[3:0]	Port 4	Port 4 control
ctl5[3:0]	Port 5	Port 5 control

Table 7.5: Crossbar datapath signals.

Signal	*Source*	*Description*
oc0_bid[17:0]	All ports	bids for output controller 0
oc1_bid[17:0]	All ports	bids for output controller 1
oc2_bid[17:0]	All ports	bids for output controller 2
oc3_bid[17:0]	All ports	bids for output controller 3
oc4_bid[17:0]	All ports	bids for output controller 4
oc5_bid[17:0]	All ports	bids for output controller 5
bid_ovc0[4:0]	Input ctlr 0	The virtual channel bid by input controller 0
bid_ovc1[4:0]	Input ctlr 1	The virtual channel bid by input controller 1
bid_ovc2[4:0]	Input ctlr 2	The virtual channel bid by input controller 2
bid_ovc3[4:0]	Input ctlr 3	The virtual channel bid by input controller 3
bid_ovc4[4:0]	Input ctlr 4	The virtual channel bid by input controller 4
bid_ovc5[4:0]	Input ctlr 5	The virtual channel bid by input controller 5
win_en	Clocks	Enables crossbar
mask[5:0]	Clocks	Psuedo-random number used to even out chances of winning
win0[5:0]	Port 0	Which input ctlr won the bid for output ctlr 0
win1[5:0]	Port 1	Which input ctlr won the bid for output ctlr 1
win2[5:0]	Port 2	Which input ctlr won the bid for output ctlr 2
win3[5:0]	Port 3	Which input ctlr won the bid for output ctlr 3
win4[5:0]	Port 4	Which input ctlr won the bid for output ctlr 4
win5[5:0]	Port 5	Which input ctlr won the bid for output ctlr 5

Table 7.6: Crossbar control and arbitration signals.

Pad type	description
in	Digital input pad.
in_jt	Digital input with boundary scan.
out	Digital output pad.
out_jt	Digital output pad with boundary scan.
analog	Analog input pad with input protection.
supply_gnd	Ground (VSS) pad.
supply_vdd	Power Supply (VDD) pad.

Table 7.7: Types of digital I/O pads used in the router.

2.

The bids are fully decoded, thus each input controller has 18 different bid lines. It will only assert one of these lines at a time. The arbitration process looks at all 18 bid lines and at the mask lines. The mask lines are used to bump the priority for a given input up one level. Therefore, the arbiter looks like a 24-input priority encoder.

7.4 Top-level Modules

This section describes all of the the top-level modules found in the router with the exception of the `port` module. The port module is complex and will be explained in its own section, Section 7.5.

7.4.1 I/O Pads

The design hierarchy in the router schematics corresponds almost exactly to the layout hierarchy. The layout composition style is based upon rectangular modules with terminals (pins) on the boundary of the rectangles. Because of this, the router does not have a "pad-ring" module. It has modules which correspond to the I/O pads on each of the four sides.

The left and right side I/O pads are predominately standard CMOS digital I/O pads. The exception are the chip and processor clock pads. These signals are differential clocks and the comparators are located near the buffers, so the clock pads are analog input pads. The complete list of pad types is shown in Table 7.7.

The `hs_port`[3] module uses two types of I/O pads. The first type of I/O pad is the *simultaneous bidirectional signalling* pad described in chapter 5. The other is a unidirectional differential variant for clock distribution between routers.

The `bias_blk` module is used to supply bias voltages to the bidirectional pads. The bidirectional pads use two types of current sources, one for pulling up, the other for pulling down. Both types of current sources require bias voltages. A third bias voltage is provided to allow control over the rise/fall time of the bidirectional signals.

[3] `hs_port` is short for high-speed port

Select	*description*
0	Reserved
1	X address minus
2	Y address minus
3	Termination resistance [7:0]
4	Termination resistance [15:8]
5	`jt_en_hs`[3:0], `jt_en_left`, `jt_en_right`, `parity_sense`, `ad_or_do`
6	Select processor clock speed
7	Interchip clock delay
8	Port 0 registers
9	Port 1 registers
10	Port 2 registers
11	Port 3 registers

Select	*description*
12	Port 4 registers
13	Port 5 registers
14	Unused
15	Port 0 I/O pads
16	Port 1 I/O pads
17	Port 2 I/O pads
18	Port 3 I/O pads
19	Left side pads
20	Left side pads
21	Left side pads
22	Right side pads
23	Right side pads
24	Right side pads
25	Right side pads

Table 7.8: Top-level JTAG register map.

7.4.2 Clock Buffers

The global clock and reset signals require extensive power amplification before driving the entire chip. This buffering is done in the modules `pr_clk_buffer`, `clk_buffer`, `clk50_buffer`, and `reset_buffer`. A related module is `clocks_final`, which divides the incoming 100MHz clock in two to produce the 50Mhz clock.

7.4.3 JTAG Modules

There are three types of JTAG modules at the top level: a JTAG controller (`tap_ctl`), an instruction register (`instructionReg`), and several instantiations of an 8-bit data register (`chip_reg8`). The controller implements the standard JTAG tap controller. It provides signals to shift data through scan chains, to parallel load a scan chain into a register, and to parallel load a register into a scan chain.

The JTAG standard defines two types of registers: instruction and data. There is only one instruction register per tap controller. The router uses a 30-bit instruction register. Bits [29:4] form the `select`[25:0] signals, bits [3:0] are used to further select a scan chain within a port. Table 7.8 shows the first level of register decode and Table 7.9 shows how the registers are decoded within a port.

7.4.4 Mask Generation

The crossbar connects input controllers to output controllers. At any point in time, there is the possibility of multiple input controllers all wanting to send a flit to the

Reg #	Description
0	Control register
1	Routing problem
2	Route
3	Route FSM
4	Busy

Table 7.9: JTAG register map for standard ports.

same output controller. An arbiter is used in each output controller to resolve the contention. From the standpoint of fairness, one would like all input controllers to have an equal chance of sending their flit. To avoid pattern-sensitive livelock, one would also like the choosing to be decorrelated with all traffic patterns. The choice should be perfectly random.

The **mask_gen** module calculates a psuedo-random number which alters the order the arbiters look at the input controllers. This effectively randomizes the choosing.

7.4.5 Win Enable

To implement plesiochronous data retiming, a certain percentage of output frames must not have data in them. The easiest way to ensure this is to occasionally turn off the crossbar. The **win_enable** module disables the crossbar approximately one out of a thousand times. The module itself is nothing more than a polynomial counter.

7.4.6 Crossbar to Processor

One of the features of the Reliable Router is the ability to operate the processor interface at a clock rate lower than the router. It is fairly difficult to find CMOS parts that can accept data at 100MHz, but considerably easier at 25MHz. For these lower speed processors, the output of router must be throttled back. The module **xbar_to_proc** generates a periodic signal, **xbar_to_proc_ok**. The input controllers know when a flit is destined for the processor and wait until **xbar_to_proc_ok** is asserted.

7.5 Ports

Ports are the major building block of the router and have input and output controllers. In the router, the input controller is much more complex than the output controller, so the port block diagram in Figure 7-7 shows the port with its input controller expanded one level. The input controller has a plesiochronous frontend, some logic common to all virtual channels such as routing problem computation, logic and control for each

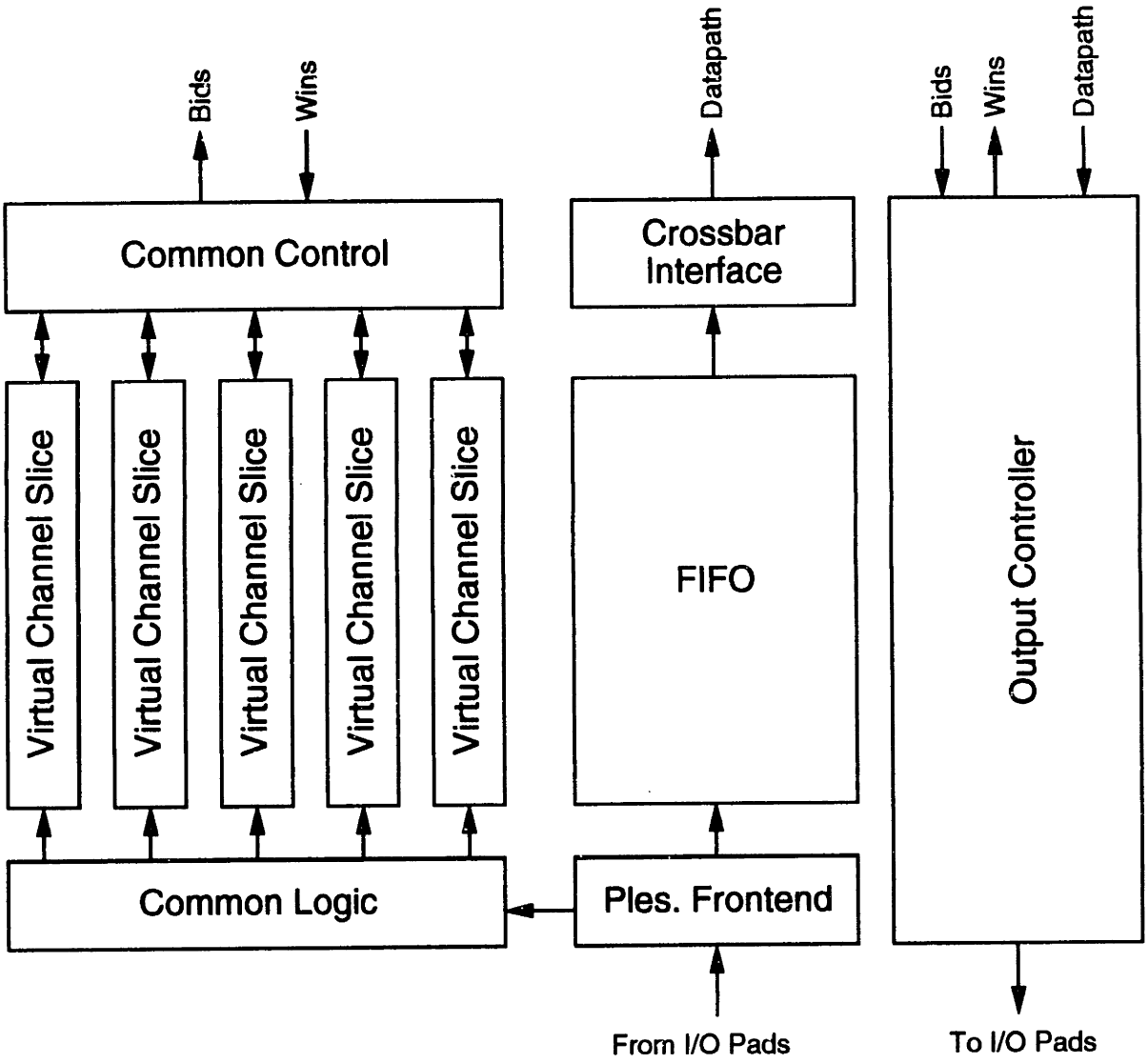


Figure 7-7: Block diagram of a port.

input virtual channel, control logic which coordinates the virtual channels, a FIFO for flit storage, and an interface to the logical crossbar.

7.5.1 Frontend

The `frontend` module is responsible for several functions. It assembles the subframes sent between routers into frames, it checks and maintains link status, and it manages the plesiochronous retiming.

Routers communicate with each other using frames of information. Frames are transmitted continuously and contain the fields shown in Table 7.10. All the subframes contain 16 bits of flit data, along with 2 bits of data parity for end-to-end checking.

	Bit Field					
	22	21	20:18	17	16	15:0
Subframe 0	pe	USR0	VCI	DP1	DP0	Data[15:0]
Subframe 1	Copied Kind		Copied VCI	DP3	DP2	Data[31:16]
Subframe 2	U/D	USR1	Kind	DP5	DP4	Data[47:32]
Subframe 3	Freed			DP7	DP6	Data[63:48]

Table 7.10: Frame Format.

Field	*Description*
`vci`[2:0]	Virtual channel identifier for the flit.
`kind`[2:0]	The kind of flit.
`data`[63:0]	Flit data.
`dp`[7:0]	Flit data parity.
`usr`[1:0]	Spare bits in the flit.
`pe`	Parity error detected flag
`freed`[4:0]	Data flit storage freed.
`copied_kind`[1:0]	The kind of flit copied.
`copied_vci`[2:0]	The virtual channel identifier of the copied flit.
`U/D`	Link up/down

Table 7.11: Frame Field Descriptions.

The field definitions are shown in Table 7.11. The flit-oriented fields have the obvious meanings. After establishing the subframe format, there were two extra bits which became the `usr` bits. They may be used for any purpose.

The parity error field is cumulative. Any parity errors detected by the receiver cause this flag to be set. This allows the transmitter to know that its information was lost and that the link between routers is failing.

The link up/down bit is used bring the link up and down. When a link is to be brought up, both routers must agree that the link is up. During link initialization, the link is down. Both routers monitor the link for parity errors. If no parity errors are seen, each router signals that it is ready by asserting "link up". A router can start sending messages over the link as soon as it sees the "link up" indication from the other router. When a parity error occurs, the link is brought down immediately.

Plesiochronous Considerations

Plesiochronous retiming requires occasional "non-data" transmissions, during which a timing adjustment is made. Frames contain both flits and control information which are not guaranteed to have overlapping non-data times. The plesiochronous adjustment is therefore managed independently for several of the fields.

The flit data, user parity, user bits, flit kind, and flit VCI are managed as one group. The crossbar ensures the proper insertion of non-data items for this group.

The copied kind, copied vci signals are managed as the second group. The copied signals are the direct result of a flit being sent across the crossbar. As such, the crossbar insertion of non-data flits also serves to meter the copied group.

Each freed signal is managed independently, as they are the result of downstream copied operations which could be performed on different routers. The interarrival time of non-data on the freed signals can be guaranteed in one of two ways. The first way takes advantage of the fact that the freed signal is only asserted for data flits, not for head or token flits. As long as the maximum message size does not exceed the number of continuous data flits allowed, ample non-data opportunities exist.

The second way is to realize that the copied/freed signals form a two-hop cascaded plesiochronous system. As long the system requirements for cascaded systems are met, the freed signals will operate correctly.

Link up/down and parity error are used internally to the frontend and are kept in `RxClk` domain. No plesiochronous adjustment is needed.

Clock Phase Generation

A *frame* is sent between routers as four *subframes*. The subframes must be latched on four successive `RxClk` edges. These latch signals are called `pp0`, `pp1`, `pp2`, and `pp3`. The prefix `pp` is an acronym for "phase positive". They are are generated by the `clock_phase` module in the frontend. Figure 7-8 shows the relative timing relationships. Phase 0 is used to latch subframe 0, phase 1 is used to latch subframe 1, etc.

Within the `clock_phase` module, `rx_phase` is sampled at the positive edge of `RxClk`. Its interchip timing constraint is therefore somewhat looser than that of the subframe `RxD` signals.

Latches

The first function of the frontend is to assemble four subframes into a complete frame. To try to minimize clock loading, the assembly is done with latches. The latch structures for retiming the subframes are shown in Figure 7-9. The complete frame appears on the outputs at the rising edge of `pp0`.

Parity Checking

Parity is computed as soon as the frame is assembled. The basic parity circuit is shown in Figure 7-10. The frontend uses up to 10 of the ripple-type stages in a row. The outputs are taken into a standard parity tree. About 10ns is budgeted for parity checking.

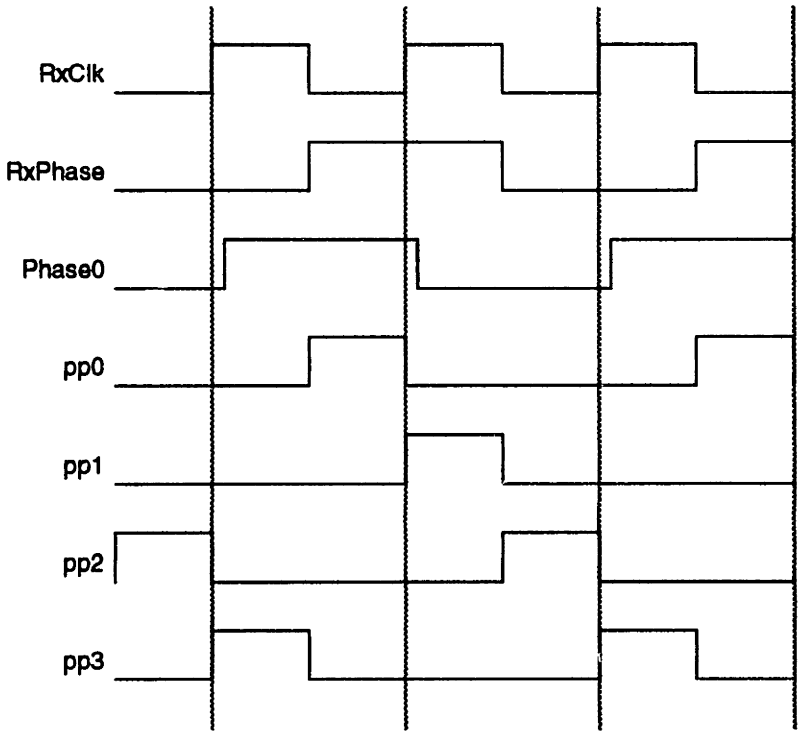
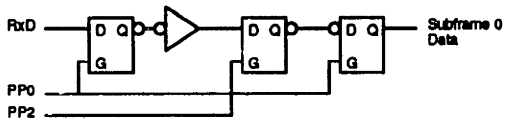
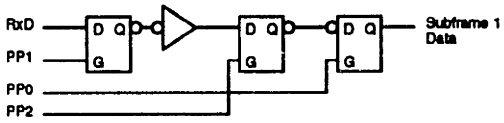


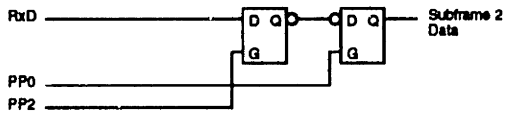
Figure 7-8: Clock phase generation. The frontend uses four different clock phases to latch subframes.



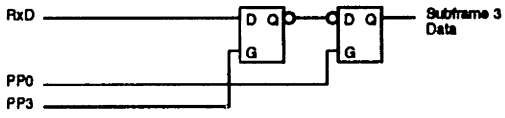
(a) Latch structure for subframe 0



(b) Latch structure for subframe 1



(c) Latch structure for subframe 2



(d) Latch structure for subframe 3

Figure 7-9: Latch structures for assembling a frame.

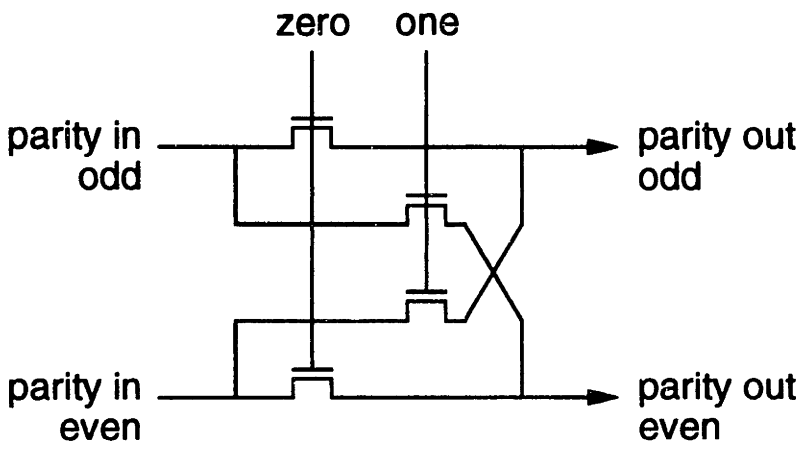


Figure 7-10: Differential ripple parity generation cell.

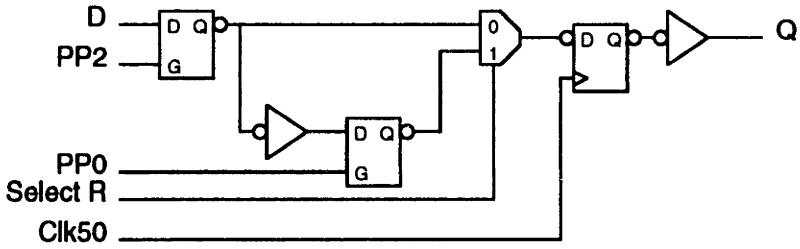


Figure 7-11: Q and R waveform generation, multiplexor.

Squash

The `frontend:final_parity` module implements the squash logic. When any sort of error is detected on a link, the incoming frames are squashed, preventing erroreous information from propagating further. Local link down and remote link down also cause incoming frames to be squashed. The `squash` signal is synchronized in the `glue` module to `clk50` where it becomes the `port_status` signal.

The `final_parity` module generates the `lcl_pe_happened` (parity error detected locally) and `rmt_pe_happened` (parity error detected remotely) signals. These signals can be read under JTAG control.

Q and R Waveform Generation

The last stage of the frontend is the generation of the Q and R waveforms, followed by the multiplexor and flip-flop into the router's internal `clk50` timing domain. The multiplexor has the cover term required for correct plesiochronous operation.

Flit Kind	Encoding
Head Original	000
Head Restart	100
Data	010
Tail	011
Token Unique	001
Token Replica	101
Reserved	110
Reserved	111

Table 7.12: Flit Kinds.

7.5.2 Glue

The glue module is just that. It transforms the tightly encoded signals transported between routers into something more easily used by the port. It decodes the flit kinds shown in Table 7.12 into the `write_head`, `write_data`, and `write_token` signals. The virtual channel identifier (VCI) is also decoded. Note that vci 0 is encoded as 001, vci 1 as 010, etc. This allows the *squash* logic in the front-end to simply crowbar the entire frame to zero in the event of an error.

The `copied_vci` decoding is similar. The assertion of `copied_kind`[0] indicates that a data flit was copied, `copied_kind`[1] indicates that a token flit was copied. The `freed` signals are simply buffered.

Lastly, the `port_status_l` signal coming from the frontend is put through a synchronizer and distributed. The failure detection logic in the frontend operates in a different clock domain so either a synchronizer or the plesiochronous retiming circuit was required. As both the up and down states can persist indefinitely, it was deemed easier to use a synchronizer.

7.5.3 Compute Routing Problem

A routing problem is a concise, easy-to-decode form of the desired next step in the route of the message. The routing is derived from fields found in the head of the message (see Table 7.13) and the incoming virtual channel number. The module contains no state information.

Essentially, the compute routing problem module compares the destination address found in the message with this node's address. It asserts which directions the message can travel in. The module handles both adaptive and dimension-ordered modes. If dimensioned-ordered is asserted and progress needs to be made in the X-dimension, the Y-dimension is deasserted. Additionally, the routing problem can be "forced" under JTAG control.

Bit fields	*Description*
4:0	Address in X
9:5	Address in Y
10	Diagnostic port (processor port when 0)
11	Priority
63:12	User payload

Table 7.13: Head Flit Format.

Bit	*Description*
0	Message needs to travel in X-
1	Message needs to travel in X+
2	Message needs to travel in Y-
3	Message needs to travel in Y+
4	For this node's processor port
5	For this node's diagnostic port
6	Message priority
7	Message priority
8	Destination node is one hop away in X
9	Message entered on a fault-handling channel

Table 7.14: Routing Problem.

Bit Field	Explanation
[4:0]	Output virtual channel
5	Output Controller x-
6	Output Controller x+
7	Output Controller y-
8	Output Controller y+
9	Processor Output
10	Diagnostic Output
11	Priority

Table 7.15: Routing Answer Decomposition.

7.5.4 Virtual Channel Slice

The virtual channel slice (module `vc`) contains nearly all of the control logic for an individual virtual channel. Figure 7-12 shows the functions which go into the virtual channel slice and their interconnection.

Routing Problem Register

A head flit begins each message. This head flit contains the routing information for the rest of the message. To avoid having to read the FIFO each time that the routing information is needed, a copy of the routing problem is keep in the virtual channel slice.

The routing problem register is nothing more than a transparent latch. It is enabled when writing a head flit during the middle 50% of the clock period.

Optimistic Router

The optimistic router "solves" a routing problem, given a particular resource set expressed by the `vc_free` and `port_status` signals. The router performs several routing computations in parallel: adaptive, dimension-ordered, fault-handling, and processor/diagnostic. The router selects amoung the outcomes of these computations, resulting in a routing answer. The bit definitions of the routing answer are given in Table 7.15.

Route

The optimistic router is combinational logic and depends on the `vc_free` signals. Thus, the answer produced by the optimistic router can change over the course of a message. The route module holds the answer (route) from the optimistic router stable during the remainder of the message. Like the routing problem, this register is simply a transparent latch. It is enabled during the middle 50% of the `clk50` cycle

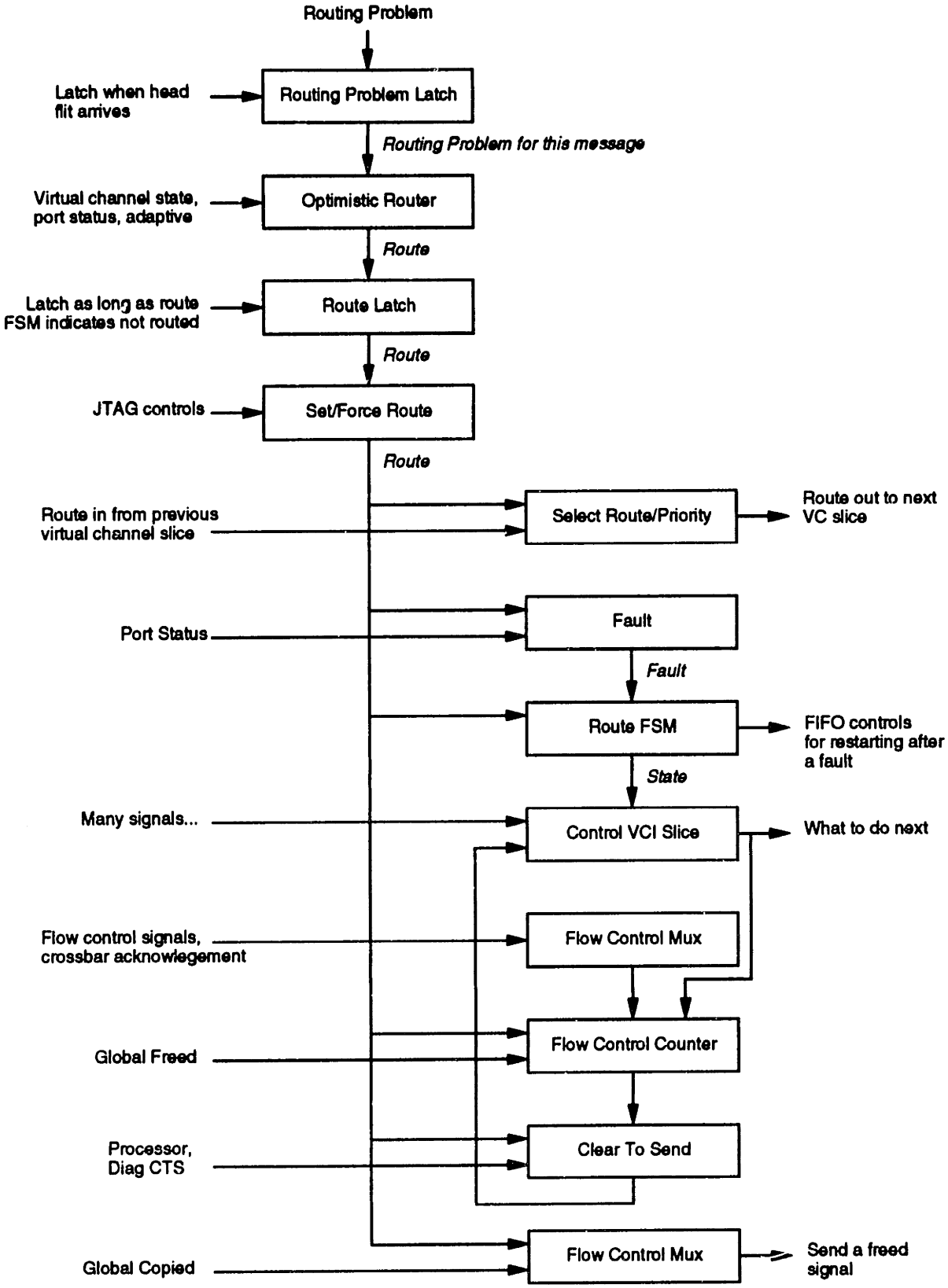


Figure 7-12: The virtual channel slice.

when the `route_fsm` indicates that the message has not been routed. Table 7.15 gives the bit assignments for the routing answer.

Set and Force Route

The `saf_route` module allows the routing answer to be read under JTAG control. It also allows a route to be explicitly set under JTAG control, overriding the optimistic router.

Select Route and Priority

The select route and priority module (`sel_route_priority`) is one portion of a multiplexor which is distributed across the virtual channel slices. The common control logic decides which virtual channel to work on, as indicated by the `read_vci` signals, but the routing information in that virtual channel needs to be brought to the crossbar.

Fault and Clear-To-Send

The fault and clear-to-send conditions depend on the output port the message is routed to. Faults can occur on any of the standard I/O ports, but not on the processor or diagnostic ports. Similarly, permit-based flow control is used by the standard ports while explicit clear-to-send indications are used by the processor and diagnostic ports. The `fault_cts` module unifies these different types of ports so that the remainder of the virtual channel slice does not need to worry about the differences.

Route Finite-State Machine

The route FSM keeps track of the state of the routing of a message. The transition diagram is shown in Figure 7-13. The machine has six states: *idle*, *routed*, *needs_route*, *backup*, *rerouted*, and *needs_reroute*. They are defined as follows:

Idle The idle state is when the virtual channel has no message.

Routed The routed state is entered when the message has been successfully routed. The optimistic router produced an answer, the common control logic decided to work on this input virtual channel, and the head flit was sent across the crossbar to an output controller.

Needs_route Occasionally, a message arrives which is unable to be routed immediately. Reasons for routing failure include the absence of an available output virtual channel or contention for the crossbar. This state indicates that the channel is not idle, but still needs to be routed.

Backup When a fault occurs while a message is in the routed state, a clock period is needed to "backup" the pointers in the FIFO.

Needs_reroute This state is entered after the backup state and is used to remember that a failure occurred while forwarding this message. It behaves just like the needs_route state.

Rerouted The rerouted state is entered when the message has been successfully rerouted. The optimistic router produced an answer, the common control logic decided to work on this input virtual channel, and the head flit was sent across the crossbar to an output controller.

Control VCI Slice

The control VCI slice determines the next step to be taken in the forwarding of this message. There are five different asnswers: do nothing, read the head flit, read the next data flit, read the token, or generate a token.

The submodule `vc_read_head` determines when to read the head flit. Three conditions must hold: there must be a head flit, clear-to-send must be asserted, and the optimistic router must have determined an answer. An additional condition of `xb_to_proc_ok` asserted is added when the message is destined for the processor.

Reading a data flit is determined by the submodule `vc_read_data`. The conditions are that a data flit be present, that clear-to-send be asserted, and that the message is routed. Additionally, `xb_to_proc_ok` should be asserted when the message is destined for the processor. The presence of a data flit is detected by the new arrival of a data flit, or by the signal `rdy_data` from the FIFO.

Sending/generating a token is bit more complicated and is handled in submodule `vc_read_mk_token`. The message must be routed and clear-to-send asserted. The FIFO provides two control signals, `rdy_data` and `not_idle`. `Rdy_data` indicates that there are still data flits needing to be sent in the FIFO, so `rdy_data` must not be asserted when sending a token. The second signal from the FIFO, `not_idle`, indicates that there are data flits in the FIFO which must be held for fault recovery purposes. All copies of the message flits are not erased (freed), so the token cannot yet be forwarded. Therefore, `not_idle` must not be asserted.

The remainder of the conditions include the `xb_to_proc_ok` constraint, and either a token arrival or a failure on the incoming link. The submodule `vc_read_mk_token` also keeps track of the presence of the token.

Flow Control

The reliable router uses permit-based flow control[4]. The permits are kept in the input controllers. When the message is first routed, the input controller has 15 permits corresponding to the total data flit storage in the FIFO, per virtual channel.[5]

The permit count is decremented whenever data flit is successfully sent across the crossbar. The count is incremented on the receipt of a `freed` signal. Of course, both

[4]The processor and diagnostic ports are handled differently using explicit clear-to-sends.

[5]The FIFO actually has 16 flits of storage, but the designer of the FIFO did not properly handle the full case. Rather than redo the FIFO, the maximum number of permits was reduced to 15.

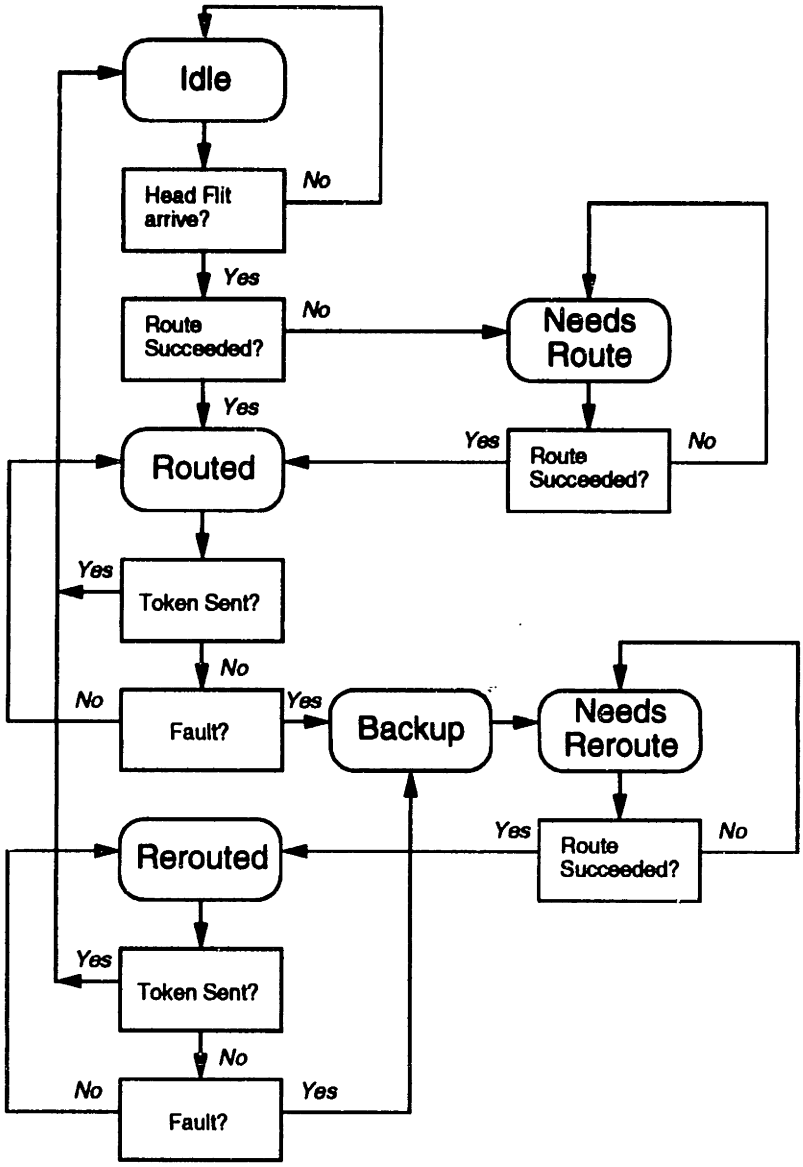


Figure 7-13: Route finite-state machine.

can happen in the same clock period. The count is initialized whenever the `route_fsm` module leaves the *routed* or *rerouted* states.

The module produces the clear-to-send signal. Clear-to-send is asserted as long as the permit count is non-zero. The clear-to-send signal is used by the `cts` module as the `port_cts` input.

Flow Control Multiplexor

In Section 7.3.3, it was mentioned that the flow-control permits flowed opposite the message. The generation of new permits corresponded to storage freeing up in the FIFO of an adjacent router. To manage the backward flow of permits from output controller to input controller, the reliable router uses two 30-to-1 multiplexors in each virtual channel slice. The select inputs to the multiplexors are the route kept in the `route` module. A route is the forward mapping of input virtual channel to output virtual channel. Using the the route to select amoung the flow control information presented by all the output virtual channels performs the reverse mapping.

7.5.5 FIFO

The FIFO provides the buffering for messages. Some buffering is required, as messages can become blocked in the network. Without buffering, these messages would be spread across several routers, tying up virtual channel resources. With buffering, blocked messages become compressed into two routers[6]. The FIFO in the reliable router provides 16 flits (128 bytes) of data storage per virtual channel. In addition, separate storage is provide for the head flit and the token flit.

All FIFO operations take place in a single `clk50` period. Data is written during the first part of the clock period and read during the second half. Writes to the FIFO occur when the frontend logic receives a flit. The signals used are `write_vci`[4:0], `write_head`, `write_data`, and `write_token`. Reads are similar and involve `xbarack`, `read_vci`[4:0], `read_head`, `read_data`, and `read_token`. The FIFO provides the signal `rdy_data` when there are data flits which should be read. The state of the head and token flits is kept in the virtual channel slice.

The write and read operations cause the FIFO state to change beyond what one might expect of a FIFO. To keep the virtual-channel flow control information accurate during a fault, a "first read" bit is kept with each flit. This bit is set when the flit is written and cleared the clock period after it is read.

Internally, the FIFO keeps three pointers: a write pointer, a read pointer, and a free pointer. The write pointer points at the next location to write, the read pointer points at the next location to read, and the free pointer points at the next flit to be freed. The write and read pointers both advance on write and read operations respectively, the freed pointer advances on the receipt of the "freed" signal[7].

[6]Owing to the need to keep two copies of a message in the network

[7]The pointers are *not* reset to zero between messages. This allows the FIFO storage to be checked using a series of small messages.

The FIFO has the ability to *backup*. Backups are needed to recover from a fault so that some flits may be retransmitted, which implies that they need to be read again from the FIFO. During a backup operation, the read pointer is set to the free pointer. Since this takes a clock period to do, the extra "backup" state was inserted into the route finite-state-machine.

The signal `data_rdy` is asserted whenever the read pointer does not equal the write pointer. The signal `not_idle` is asserted whenever the free pointer does not equal the read pointer. Both of these must be non-asserted to allow either the sending or the creation of a token.

7.5.6 FIFO Interface to the Crossbar

The FIFO interface to the crossbar (module `flit_munger`) is a wide register followed by many 2-to-1 multiplexors. The register is to hold the flit data read out of the FIFO for transmission across the crossbar. The multiplexors are used to send the flit in two halves. Sending the flit in two parts significantly reduced the number of wires used in the central wiring channel.

Additional register/multiplexors are used to derive the control fields which go along with the flit data. Additional logic in the flit munger handles the encoding of `read_vci`[4:0] back into 3 bits, generates a token if needed, and adds the restarted field to the flit kind.

7.5.7 Upper Left

The `upperLeft` modules is a collection of small modules gathered together for floor planning purposes. It comprises the busy/free virtual channel state logic, the bid encoder, the arbiter, the common control, and the JTAG subcontroller.

Busy/Free Logic

The busy logic keeps track of the allocation state of the output virtual channels in this port. Whenever a flit is sent to the output controller in this port, the output virtual channel is tagged as busy. Later, when the token flit is finally copied forward *in the downstream router*, the output virtual channel is freed.

Each bit of the allocation state can be individually controlled via the JTAG controller. For example, it is possible to "busy-out" both of the adaptive channels, leaving the router as a single-channel, dimension-ordered router. This control, together with the JTAG control over the routing problem, allows the boundary scan to thoroughly test the routing logic.

The free encoder reduces choice of virtual channels given the optimistic router. For example, adaptive channel 0 is the preferred channel. If it is free, neither adaptive channel 1 or any of the dimension ordered channels will appear as "free". This helped reduce the critical path through the optimistic router.

Bid encoding

When an input controller wishes to a send a flit to an output controller, it first tells the output controller that it has a flit to send. This notification of intent to send is called a *bid*. Each bid has an associated priority level. To simply the arbiter, the bid $\times$ priority space is fully decoded into 18 individual bid lines.

Arbiter

The arbiter looks at all the bids for an output controller and decides which bid wins. This is done using a priority encoder with full look-ahead. Since a priority encoder has a fixed order, one port could cause starvation on another port unless an adjustment is made. This is done by randomly boosting some subset of the bids up in priority.

Common Control

The role of the common control logic is to resolve contention amoung the various input virtual channels. The concept is to look for virtual channels which can actually make progress. For example, if a virtual channel has no flit to send or is blocked on flow-control, it make no sense to choose to give that virtual channel access to the crossbar.

The arbitration is done in the module `lrd_cc`. Each of the virtual channel slices asserts if it has work to do and the nature of the work on the `vec_read_*`[4:0] signals. This module determines the virtual channel to read and the type of flit to be read.

The arbitration is somewhat randomized by the `lrd_cc_vc_pri` module. This module goes round-robin through the virtual channels, giving a virtual channel and all lower-numbered virtual channels higher priority. The arbitration is implemented as a fixed priority encoder, so this prevents starvation.

The `lrd_cc_encode` module generates the *copied* fields which are passed between routers. These fields are qualified by the crossbar acknowledgement, `xbarack`.

JTAG subcontroller

The main JTAG instruction register provides port-level select and four address bits (`jt_low_word`). It up the individual port to provide decoding to select individual registers and to provide the multiplexing of the register data-outs into a single port data-out. This is done the JTAG subcontroller module `jtag_generic`.

The decode and multiplexor functions are provided in the submodule `jtag_decode`. It provides decodes and multiplexing for 10 JTAG data registers.

A control register submodule `jtag_register` supplies all of the miscellaneous control bits such forcing a route, enabling the override of the routing problem, clearing parity errors, and setting the link state.

	Subframe 0	Subframe 1
ctl[4:0]	vci[4:0] (fully decoded)	usr[1:0], flit_kind[2:0]
par[3:0]	Data parity for bytes 3-0	Data parity for bytes 7-4
data[31:0]	Flit data 31-0	Flit data 63-32

Table 7.16: Processor Input Frame Format.

7.6 Processor Port

The processor port reuses many of the same components found in the standard port. The differences are in the frontend (`frontend`) and output controller (`outctl`) modules.

7.6.1 Processor Frontend

The role of the processor frontend is to move signals generated by the processor into router's clock domain, to handle the allocation of input virtual channels, and provide clear-to-send indications to the processor for each of the five input virtual channels. Each of these functions is handled by a different module.

Processor to Router Retiming

The processor input is plesiochronous with respect to the router's clock. It is up to the designer of the external processor interface to ensure that ample non-data items are inserted.

Processor input frames are broken into two subframes, as shown in Table 7.16. Both subframes are sampled into the processor frontend on the rising edge of the processor clock. Subframe 0 is indicated by a 1 on the processor phase input signal, `pr_phasein`. A 0 on the same signal indicates that it is subframe 1. Framing is constantly occurring on the processor input, even when flits are not being sent.

The remainder of the plesiochronous retiming is done in processor frontend the same as was done in the standard frontend.

Input Virtual Channel Allocation

One the more subtle parts of the Unique Token Protocol is that the buffers in the input controllers are not truly free until the token has been sent. In the case of the processor, the processor may have loaded an entire message into the input controller and now must wait until the router has sent the token before reusing the virtual channel.

The processor input virtual channel allocation works as follows. The external processor interface begins with all virtual channels free. It marks a virtual channel as busy as soon as it injects the head flit. Some time later, when the input controller sends the token, the processor frontend will pulse the corresponding `pr_free_out`[4:0]

signal for one processor clock period. This pulse should mark the virtual channel as free.

The module which converts the "send copied token" into the `pr_free_out` signal is called `free_sync`. This module sets an RS latch when it detects "send copied token". The output of the RS latch is put through a synchronizer and edge detecter in the processor clock domain, resulting the the `pr_free_out` signal.

Clear-To-Send to the Processor

Each of the input virtual channels provides a clear-to-send to the processor. These clear-to-send signals are produced using the router's clock and need to be synchronized externally to the processor's clock.

The module `flow_control` in the processor frontend library manages the conversion of permit-base flow control into clear-to-send flow control. The module keeps track of the number of free data flits in the FIFO for a given virtual channel. Clear-to-send is dropped when 6 permits remain, it is raised when at 7 permits are again present.

7.6.2 Processor Output Controller

The processor output controller has three functions: it moves flits from the router's clock domain to the processor clock domain, it synchronizes the processor's clear-to-send signal to the internal clock, and it generates the flow control signals.

Router to Processor Retiming

The router to processor retiming logic is made complicated by allowing the processor to operate a slower speed. The steps involved are: to reassemble the flit from the crossbar, to generate the Q and R waveforms, generate either select Q or select R, move the flit into the processor clock domain, and then multiplex it back into subframes. The reassembling followed by the multiplexing seems unnecessary, but it is required in order to have sufficient timing edges for the plesiochronous adjustment.

The first step is the reassembly of the flit from the crossbar. This is only done when the signal `xb_to_pr_ok` is asserted the previous `clk50` cycle. This produces a flit at a rate one half that of the processor clock. For example, if the processor clock is 100MHz, a new flit is produced every 20ns or at a 50MHz rate. This reassembled flit becomes the Q waveform.

The R waveform is generated from the Q waveform by delaying it a router clock period. When the processor clock equals the router clock, flits are produced every 20ns. To properly position the R waveform for this case implied a delay of 10ns, which is a router clock (`clk`) period. The same delay is used even when the processor is operated at a slower speed.

The generation of select Q and select R begins with the creation of the keepout window. The keepout-Q window is nominally a router clock period in width (10ns) and is centered on the Q-waveform exclusion time. The keepout window is sampled

in the processor half-clock[8] domain and then synchronized back to the router clock domain. The select-Q/select-R finite state machine is updated when both Q and R are valid.

The flit can now be safely sampled by the processor half-clock and is then multiplexed into the two subframes for the processor.

Clear-To-Send Synchronization

The processor provides a single clear-to-send indication to the router. This clear-to-clear is synchronized and then distributed to all ports.

Flow Control Generation

When a flit finally reaches the processor, its storage in the network of routers should be released. If the flit is a data flit, both the global copied (`gbl_copied`) and global freed (`gbl_freed`) signals are asserted. The global copied signal causes the storage in the upstream router to be released, the global freed signal releases the storage in this router. If the flit was a token, the output virtual channel is freed.

7.7 Diagnostic Port

As with the processor port, the diagnostic port differs from a standard port in its implementation of the frontend and output controllers.

7.7.1 Diagnostic Frontend

The diagnostic frontend is principally a large JTAG register which which holds the flit data, kind, and virtual channel identifier. Once this register has been loaded, a second JTAG register called the launch register is written to. The output of the launch register is synchronized into the router's `clk50` domain. The synchronized launch signal is then fed into a rising-edge detector. The rising edge causes the flit to enter the rest of the input controller.

The diagnostic frontend also watches for the arrival of `freed` signals. Data flits can be entered one at a time. The writing of a data flit resets an RS flip-flop. The `freed` sets the flip-flop. The outputs of the flip-flops can be read under JTAG control.

7.7.2 Diagnostic Output Controller

The diagnostic output controller provides a means for capturing a flit when the flit leaves the crossbar, flow control, and virtual channel allocation.

[8] A clock operating at one-half the processor clock speed. The rate of this clock corresponds to the processor flit rate.

Capturing the Flit

The flit capture function is simply a wide parallel-loadable register. This register is loaded with the flit from the crossbar whenever the flit valid (`fv`) is asserted. Flit valid is asserted whenever any of the input controllers is sending a flit to the diagnostic port. The contents of this register can the be loaded into a JTAG scan chain and read out.

Flow Control and Virtual Channel Allocation

As there is only one output register, sending a flit to the diagnostic port is a one-shot operation. The diagnostic clear-to-send signal (`diag_cts`) is asserted under JTAG control. The clear-to-send is negated immediately whenever a flit is sent to the diagnostic port.

Like the processor port, this is the final destination for a flit. The other flow-control signals, global copied and global freed, are asserted whenever a data flit arrives. Similarly, the virtual channel is marked free whenever a token arrives.

7.8 Summary

The design of the Reliable Router explored several microarchitectural points. Some of the insights gained are:

- The crossbar function can be distributed. The insight was the wiring pitch was smaller than the datapath pitch of the FIFO.
- Permit-based flow control causes the output controller to be minimal.
- Permit-based flow control and virtual channels seemed to imply a large bus. The high fan-in multiplexor, although it seemed to be brute force, is the least complex.
- Bookkeeping for maintaining two copies of all flits in the network was high. This logic could be simplified if the entire message fit into the FIFO, as the bookkeeping could then be done at the message level.
- Relatively little logic was used to keep track of the token.
- Giving each virtual channel a copy of the router simplified the design tremendously and eliminated one possible cause of livelock.
- Each port actually has very few interactions with other ports. This can be exploited in the design of the clock distribution system.

Most importantly, the router design was decomposed into many small, easily understood modules. The most complex state machine in the router is the JTAG tap controller. The next most complex is the route finite state machine which has 6 states. The decomposition led to good test coverage at the module level. When system level bugs were found, corrections were usually easy to make.

Chapter 8

Design Methodology

The Reliable Router was implemented using the Cadence integrated circuit design tools. The principal tools used were the schematic capture system, the Verilog simulator, HSpice, the layout editor, and the SKILL [1] language interface. The synthesis and place-and-route tools were not used.

All logic was entered using schematics. It was found that most of the logic structure benefited from human inspection. This was especially evident in the optimistic router, where altering the free virtual channel encoding in the busy module reduced the logic complexity in the route computation. Many of the other modules, such as the plesiochronous frontend, had very specific timing requirements. In a synthesis attempt, one wound up coding in a structural style to achieve the desired result. Further, crafting the layout lead to shifting functionality between modules. Schematics simply proved to be easier.

The supplied place and route tools suffered some limitations. The Reliable Router is composed of several large blocks, the ports. Each port is made of modules which are designed to fit well together. It was discovered that one had very little control over the final width of a module or the exact placement of of pins. Perhaps the most serious flaw was the team's inability to automate the place and route process using scripts or SKILL code. During the design of a previously fabricated test chip, power and ground were not connected to a module. This was the result of running the place-and-route package by hand late at night and neglecting to manually edit some terminals before proceeding.

The remainder of design methodology of the Reliable Router is somewhat unique. It relies on SKILL routines to construct all modules. No hand-editting[2] of the resulting modules is required, which allows the consistent reproduction a cell or module. In theory, one could start with the hand-drawn cells and rebuild the entire router by loading and executing SKILL programs.

The remainder of this chapter presumes some knowledge of the Cadence tools as it explores the components developed for the methodology.

[1] SKILL is a LISP-like language. The Cadence design interface has an interpreter which can be used to control nearly all of the high level tools.

[2] Well, one or two places...

8.1 Standard Cells

The standard cells are laid out in a gate-matrix style. Power and ground rails are 14λ wide to allow three vias vertically. Bottom GND edge to top VDD edge is 66λ. There is space for two internal wiring tracks.

8.1.1 Standard Cell Generation

Most of the standard cells were generated using SKILL code from the schematic database. A tool, *layoutGen* was written for this purpose. It consists of three parts.

The first part is a netlister which flattens the schematics down to the transistor level. It is much easier to draw a NOR2 followed by an inverter than to repeat all the transistors in an OR2 gate.

The second part finds the gates. It first does a *clustering* pass where it separates transistors connected via source or drain regions into separate clusters. Pass gates are pulled out of the clusters at this point. N and P transistors are also separated.

The N and P transistors are inspected for the common series and parallel configurations. A series of N transistors, parallel P transistors is recognized as the common NAND configuration. NOR gates, inverters and multiplexors are also recognized at this point.

The recognized patterns are used to arrange the transistors. The order of the gates may be determined by simply starting at GND (or VDD) on the series transistors. For gates which don't fit any of the predefined patterns, the program simply exhaustively searches the gates, looking for the tightest packing.

Next, the clusters are tried in various orderings and mirrorings, also looking for the tightest packing. For example, a nand gate followed by inverter can share the power and ground contacts.

In the third step, the gates are drawn. A poly river-router is used to connect the top gates to the bottom gates. A very naive one-track metal1 router is used to connect P and N diffusions. An intelligent choice is made for the location of the metal2 output track, to try mininize its interference with the poly inputs.

Lastly, the N-well, the well plugs, and the substrate contacts are drawn and the terminals are added. Figure 8-1 shows the final result, as produced by the code.

Large Inverters

Large inverters are handled as a special case. They require transistor splitting and have multiple substrate and well contacts. In addition, the input is distributed using wide poly.

8.2 Standard Cell Placement and Routing

Standard cell placement is done by hand using SKILL code. The main routine is `instsPlaceRow`. It takes a list of instances and a X-Y coordinate, and then places the instances in a row. An instance can be shifted relative to its neighbor by specifying

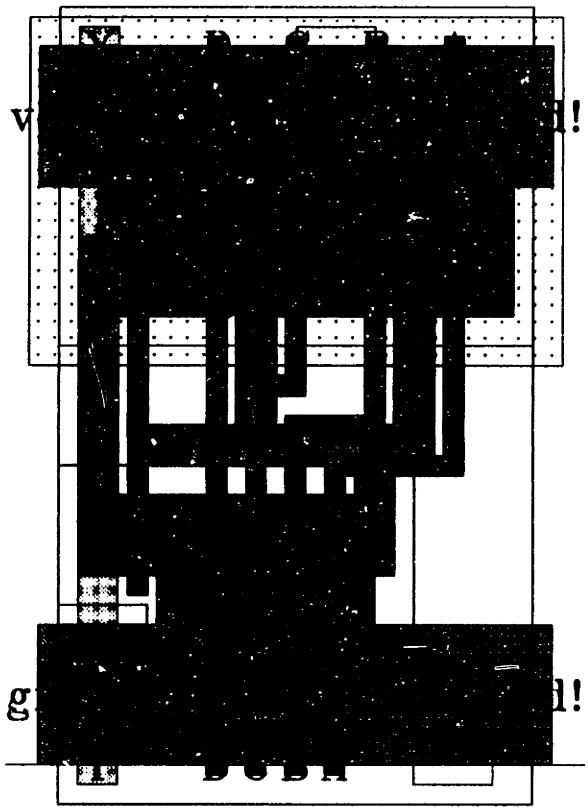


Figure 8-1: Layout of an AND4.

a delta X for that instance. Similarly, an instance can be positioned at an absolute X coordinate (useful for datapaths) or mirrored. The row placement routine handles connecting power and ground between the cells, and it will also add in additional N-well to prevent design rule violations.

Multiple rows are constructed using multiple calls to `instsPlaceRow`. The coding style has been to declare several variables RY0, RY1, RY2, ...which give the Y-coordinates of the rows. Each Y-position is calculated from the previous by adding

$$66 + (\text{trackCountToChannelHeight tracks})$$

to the previous coordinate. It is then easy to adjust the height of any channel.

Routing channels are rectangles. Given standard cells placed in rows, `channelHorzFind` returns an array of rectangles corresponding to the routing channels. Options include extra space to the west and east, an absolute width, and the top and bottom channel heights. The absolute width is very useful for constructing blocks of cells which abut vertically.

8.2.1 Routing

The base routine is called `wireChannel` and is a greedy, one-wire-at-a-time router. It was discovered that small permutations of the routing order were sufficient to achieve good channel density. The wiring routine takes in the channel, a list of poly rectangles, and a list of metal2 rectangles. It finds a metal1 wiring track and connects them up. The `wireChannel` routine takes the following options:

gravity Ordinarily, the router tries to find a track on the side side of the channel with the greatest number of poly pins. The side preference may be overridden by this argument.

trackNum The track number may be explicitly specified. The tracks start from 0. The numbering starts at either top or bottom, dependent on the channel's gravity.

eastExtend,eastExtend Extend the track to the west and east ends of the channel.

pinWest,pinEast Extend the track to the west and east ends of the channel. Add pins.

locatorWest,locatorEast Extend the track to the west and east ends of the channel. Add pins

netName The name of the net used to construct pins.

direction The direction of the pins, e.g. "input".

viaMergeThreshold How close two vias have to be before they are merged into a single via.

trackWidth The width of the metal1 track.

The hard part of the wire router was not track assignment, it was getting the "feeder wires" from the cell pins to the track. The routine has two strategies. Very often, a vertical run is blocked by a poly contact. This is seen when connecting to the gates of a NAND stack. The wiring routine tries to push the contact out of the way and often succeeds. The second strategy is that it can side-step somewhat in the X-direction, creating a jog. This is also quite effective.

The routine `wireVerticalChannel` is very similar to `wireChannel`, except that it handles vertical wiring channels.

8.2.2 Multiple-Channel Router

The channel router is painful to use on multiple rows of standard cells. A multiple-channel wire router was written to help. The multi-channel router takes in a list of channels, a list of output terminals, and a list of input terminals. A terminal is a list consisting of an instance and a pin name.

The multi-channel router first decides the set of channels which need to be routed. It then looks to ensure that there is at least one metal2 jumper over each row of cells. If jumpers are missing, it adds them in. The router then goes channel by channel, trying to connect as many pins as possible. It does correctly handle cells with a pin only on either the top or bottom of the cell (latches or muxes). Like `wireChannel`, `wireMultiChannel` has many options.

gravity The gravity option is passed into `wireChannel`.

westExtend, eastExtend Channels numbers in which the metal1 track is to be extended to the west and/or east edges of the channel.

netName Used to construct pins.

pinWest, pinEast The channels in which to put pins.

locatorWest,locatorEast Like pinWest and pinEast, only using locators.

pinSouth, pinNorth Adds pins on the north and/or south sides.

channelBiasUp Ordinarily, the router is biased toward the lower channels. This reverses the bias.

useChannelZero The router tries not to use channel zero unless required. This allows it to be used freely.

trackInChannelList A list of track assignments. This is useful for ensuring that pins are properly positioned for abutment.

polyJumpers Allows the use of polysilicon jumpers, not just metal2. Useful for slow speed logic and when it just won't route otherwise.

jumperX The router tries to find jumpers around the X-coordinate of the output pin. This overrides the starting position for the search.

8.3 Power and Ground Router

The power and ground router connects the ends of the standard cell rows together using bands of metal2. One can specify if bands are desired on the west and/or east sides, if pins are to placed on each of the four sides, and the width of the strapping. Occasionally, a module was instantiated mirrored, so the capability of swapping the vertical positions of the power and ground straps was added. Additional strapping could be added through the middle of the rows.

8.4 Module-Generator Generator

It would be very time-consuming to enter the SKILL code to route a block of standard cells. Instead, a *module-generator generator* (**mgg**) is used. A module-generator generator takes as input a schematic and produces a SKILL procedure, **mg**. The routine may then be executed to produce the actual layout.

The module-generator generator looks at all of the symbols on a schematic. If a symbol had a layout representation, it is treated as a base cell. Otherwise, the symbol's schematic view is flattened into the netlist and the process repeats[3].

The netlist is then converted into a program which has several parts. The main procedure is called **mg**. It calls routines which do the cell instantiation, placement, power and ground routing, followed by many calls to the individual wire functions. Each net in the schematic is converted into its own wiring routine.

The placement routine is a stub. The user adds several calls to **instsPlaceRow** to place all of the cells. Since an average **mg** call took about 15 seconds, one altered the placement and re-ran it until one was happy with the results.

The routing quality was also improved by some manual intervention. One routed the clock lines first, to keep the "feeder wire" lengths down. Afterwards, the net routing order was permuted until the resulting wiring looked good.

8.5 Validation

The Reliable Router was checked using *flat* DRC and *flat* extractions. It was observed that shorts to underlying modules were not properly handled using hierarchical techniques.

However, some degree of hierarchical checking was needed to expedite the initial module validation. A flat LVS often highlighted too much information to understand where the error was. SKILL code was written to generate LVS views for cells and modules. These were then treated as "black boxes" by the netlister. The tricky part

[3]The proper handling of busses and bundles was very tricky!

was setting up the netlister so that it could do either a flat or hierarchical comparison without either recompiling the technology file or deleting the LVS views. This was done by having the netlister look at the name of the run directory. If it ended in ".abstract", it was a hierarchical LVS.

The top-level chip and core were only checked hierarchically. The extraction files grew too large.

The top-level chip could not be flat DRC'd in one pass. Instead, a shell script was used to iterate a flat windowed DRC over the entire chip.

8.6 Top-Level Power and Ground

The top-level power and ground routing is done in metal3. Each top-level module includes its own metal3 connections. This was done by creating a derivitive cell, usually named something like `cellName_m3`. SKILL code was written which looked at the cell's placement in the module or chip and then added the metal3 bands in the right locations.

For example, the ports looked at their locations in the core module. They added the bands and created modules `nport0_m3`, etc. Each of these module was flat extracted and LVS'd, to ensure that there were no power and ground shorts. The core module then simply abutted the ports to connect the metal3 together. No metal3 was painted over the top of the ports in the core module.

The bulk of the chip power distribution comes from the left and and right side pads. Again, these modules looked into the chip layout for the locations of the metal3 pins on the core module. These were added to the left and right side modules.

The top-level power and ground routing was mostly wiring straight from the pads to the core. Some additional work was needed to connect the random top-level modules, such as the JTAG controller.

The actual construction of the metal3 to metal2 vias was done in SKILL code. The HP26 process does not allow stacking vias. The code looked at the geometry underneath the band for good via locations.

8.7 Other Tricks

This section lists a few of the things the router team learned and developed over the years.

8.7.1 Technology File

It took many months to get the technology file stable. The Reliable Router technology file uses colors very similar to Magic, including the gray background. Some experimentation was required to produce stipple patterns which looked good - a hard case is metal1 over poly over diffusion.

The design rules are almost MOSIS scalable CMOS. The router is fabricated in the HP26 process, so poly surrounding a contact was reduced to 1λ. A strict option

can be turned on in the design rule checker to ensure full compliance with the scalable rules.

The technology file was split into several pieces. This allowed for more rapid experimentation with colors as well as retargetting the basic libraries for a different process. After some time, it was dicovered that the technology file was just standard SKILL code, thus any SKILL constructs could be used in the technology file.

8.7.2 Standard SKILL Library

The standard SKILL library has several components. The `style.il` file defines a small number of useful macros, such as *first*, *second*, etc. for taking elements out of a list. The `point.il` file defines useful operations on points, such as addition, subtraction, delta x, and delta y.

Rectangles are the work-horse of the layout and wiring routines and are defined in `rectangle.il`. There are 36 rectangle-oriented routines and they are listed in Table 8.1. Most of the routines are very simple, yet without them, the resultant higher-level code is very confusing.

All types of contacts may generated using the routines in `contact.il`.

Cells and pins are related. To make naming pins easier, cells must have their pins centered on the edge of a rectangular box. This gives a natural way to talk about the VDD pin on the west side of the cell. By convention, poly pins straddle the edge by 1λ on each side, all others layers straddle the edge by 2λ. Several useful routines are defined, such as `cellPinBox` which returns the rectangle which has edges through the centers of all the pins.

Instances are instances of cells. Like cells, it is useful to find the *pinBox* via `instPinBox`. However, instances can be rotated and mirrored, so `instPinBox` returns the rectangle after transformation. One of the most useful things to do do with an instance is to find the location of a pin. `instFindPinOnSide` return the rectangle for a pin on a particular side of an instance *after transformation.* A standard cell which has GND pins on the west and east sides will have them on the north and south after rotation.

One other very common task is to abut two instances. The routine `instAlignPins` is used to position two instances so that the specified pins overlap. This is useful for constructing a row of standard cells by simply aligning GND pins.

Quite often, one needs to refer a point inside of a cell which is not on the pin box. A data type called a *locator* is defined. Locators must be unique within a cell. They consist of a rectangle, such as metal1, and a label. Their position may be found by the label name. Locators are used extensively in the top-level wiring of the Reliable Router. Horizontal wiring channels brought their wires to a locator, which was then used by the vertical channel router to complete the wire.

8.7.3 Schematic Capture

The canonical design representation is the set of schematics. All other representations must agree with the schematic form. Some simple conventions were adopted:

Routine	Arguments
rectMake	lx ly ux uy
rectCenter	r
rectSize	r
rectWidth	r
rectHeight	r
rectCanon	r
ptInRect	pt rect
rectValid	rect
robustMin	a b
robustMax	a b
rectIntersection	r1 r2
rectIntersectionX	r1 r2
rectIntersectionY	r1 r2
rectUnion	r1 r2
rectUnionX	r1 r2
rectUnionY	r1 r2
rectAllCorners	rect
rect_lx	rect
rect_ly	rect
rect_ux	rect
rect_uy	rect
rectDX	rect dx
rectDY	rect dy
rectSet_lx	rect v
rectSet_ly	rect v
rectSet_ux	rect v
rectSet_uy	rect v
rectStretchX	rect x
rectStretchY	rect y
makeRec	points
rectAddPoint	r p
rect_x_interval	r
rect_y_interval	r
rectBloat	r delta
rectCenterAtX	rect x
rectCenterAtY	rect y

Table 8.1: Rectangle Routines.

- There are two useful sizes of "drawing paper", A-size and BH (for B-Hacked). These had the proper aspect ratios to fit in the 8.0"x10.5" printable area of a standard laserprinter. The BH size was determined by making the boundary progressively bigger until the scaled down output was almost too small to read. All the typical revision block things were discarded as one could find out more by using the Design Manager. This left the greatest amount of white space for actual drawing.
- All wires and symbol pins were drawn on a .125" grid.
- The square that served as the connecting point on a symbol was replaced with a much smaller circle. It is now barely visible on the schematic plots.
- The pin length on a symbol defaulted to .250". It was shortened to .125". Similarly, the inversion bubbles were enlarged to a full .125".
- All schematics should be on a single page when possible.
- The name of a basic gate was left off the symbol. Most people know a NAND4 when they see one.
- All of the basic gates had DeMorgan equivalents for active low-logic.
- These symbols were generated using SKILL code for uniformity.

These simple rules tended to produce schematics which were readable and mostly uncluttered.

8.7.4 Miscellaneous

A standard cell validator was written. Most of its utility resided in cleaning up some layout artifacts, such as mislabeled pins and zero-area shapes.

A utility to renumber instances on a schematic was useful. After many edits, the instance numbers got rather large. It was nice to get to them to be I0, I1, I2... again.

Utilities to list module usage in both schematic and layout forms were needed for garbage collecting the libraries.

8.8 Summary

The resultant tool-set proved to be quite powerful. It adapted itself well to a variety of layout construction problems, ranging from standard cell blocks to datapath to chip composition. The insistence on rigorous checking has paid off by producing an operational device.

However, most of the tools were written on an as-needed basis. It would be very desirable to go over the entire suite. There is some inconsistency in routine naming, and some arguments are no longer needed. Some features were grafted on in an ackward way in order to retain backward compatability. Still, the team would build the chip the same way if they had it to do over.

Chapter 9

Testing

A small test PC board was designed and built. The test board holds two Reliable Routers, clock generation circuitry, and some LEDs. It also has an interface between the JTAG ports on the routers and a parallel digital I/O board, located in an PC. Only preliminary testing has been done to date. The following tests have been tried:

Smoke test

Power was gradually applied. It didn't smoke.

Current Draw

Power was increased to 5 volts. Static current was under 100mA. No signs of latchup on repeated power cycling.

JTAG Instruction Registers

Using code on the PC, the JTAG instruction was successfully written and read from. This was converted into a test loop, which ran 1,000 write/read cycles successfully.

JTAG Data Registers

The on-chip registers can be written to, but not read from. Overriding an I/O pad connected to an LED allowed us to see that register writes were working. See Section 9.1 for a description of the bug.

Diagnostic Port to Processor Out

Using the JTAG functionality, a head flit was loaded into the diagnostic input and then sent to the processor output. We were able to verify `pr_phase_out`, `pr_validout`, and one of the data bits. Many, many things on the chip had to be operational for this to work. Compute routing problem, FIFO, flit Munger, arbiter, disable of other ports, virtual channel slice, crossbar wiring, processor output controller including plesiochronous retiming, xbar to processor ok, clocks, reset, busy/free logic, common control, flow control. In short, nearly everything had to work.

Processor clock in to phase out is about 12ns.

The computeRoutingProblem does seem to compare the address in the head flit with the chip address.

Proper operation was observed at 32MHz.

9.1 Errata

1. The final JTAG latch in the IR should be clocked on tclk. It is currently attached to clockIR.
2. clockIR and clockDR are queiencent high, not low. The clock enable logic should reflect this.
3. Add back in the JTAG bypass register.
4. In the pr_frontend flow control slice, add a latch before the R and S inputs. Works because the initial polynomial states all have cover terms, but it is just not a clean design.
5. CTS in the diag output needs more buffering.

9.2 Desirata

1. Remove unneed latches from JTAG subregisters.
2. Make more state readable.
3. Make the FIFO writable and readable under JTAG control.
4. Drop CTS to the processor a bit earlier. The pipelining and synchronizer delays make it a bit close.
5. Add individual blocks on the diagnostic input, to allow the diagnostic routine to load an entire message and then send it.
6. CTS in the diagnostic port could be moved up the pipeline by looking for any bids in the previous `clk50` cycle.
7. Add the ability to override the internal port_status signals under JTAG control. This would allow the sending of a flit to an unconnected port for test purposes.

Chapter 10

Conclusion

The research behind this thesis began by looking for better ways to build large, fault-tolerant, parallel computers. It was observed that large machines tended to be less reliable than small ones and that parallel computers also tended to be large machines. They could stand to benefit from some fault-tolerant capabilities.

Parallel computers are generally built from processing units which are interconnected using a network. The processing units are small and often multiple units are placed on the same circuit board or circuit module. The technology used to construct these modules is fairly reliable. What tends to be less reliable is the interconnection network, as it includes things like connectors and cables. These connections will be made and unmade over the life of a machine, further reducing their reliability. Thus, a good place to add fault-tolerant capabilities is the interconnection network.

Like the interconnection network, the other globally distributed resources within a parallel computer can lead to reduced reliability. These resources are power and clock. Any techniques developed should try to eliminate the need for either a system-wide power source or a system-wide clock source.

The performance of the system is important. An increase in reliability must *not* be offset by a corresponding decrease in performance.

Contributions

To address these concerns, the Reliable Router project set about to build a better interconnection fabric. The following are the results:

- The Unique Token Protocol was developed to provide fault-tolerance in the network. This protocol is not an end-to-end protocol. The protocol obviates the need for the the sender to keep a copy of a message until an acknowledge is received. Instead, the sender holds onto a copy until the network has two copies. No final acknowledgement from the message recipient is needed. Further, when a fault does not disrupt the message, the network is able to certify to the message recipient that the receiver has gotten exactly one copy of the message. The protocol has been modified for worm-hole routing.

This protocol allows the construction of a high-performance, low-overhead, fault-tolerant communication network.

- Simultaneous Bidirectional Signalling is used for high-performance interchip communication, 200Mbit/s each way per wire. A single wire carries full duplex communication concurrently through the use of waveform superposition.

  This signalling technique improves the performance of the communication network. Further, the current steering technique employed needs only a single voltage reference between routers, VT . Power and ground can be partitioned for each processing unit.

- Low-Latency Plesiochronous Data Retiming is used for interchip communication. Each router operates in a separate clock domain. The basic retiming method is fully described, along with circuits and constraints for correct operation. Average latency is one-half a clock period. Extensions for integral subrates and cascaded timing circuits are also developed.

  This retiming technique eliminates the need for any global clock distribution system, yet does so with a minimal performance penalty.

- The Reliable Router is a high-performance device. It uses wormhole routing, 5 virtual channels per physical channel, 2 priority levels, minimally adaptive and one-fault-tolerant routing. Bandwidth between routers is 3.2Gbit/s, each way. Router latency is about 70ns. The router has been fabricated on a 13.5mm by 15mm die and packaged in a 463-pin PGA.

  The router has been implemented and partially tested. The techniques developed have clean implementations and do not have significant area or performance penalties.

Future Work

These are a few of the things still to do. The first set of tasks is to leave the Reliable Router in a good state (a critical part of the methodology).

- There is additional testing to be done on the Reliable Router, to bring up a link and verify the signalling, retiming, and error recovery circuits.
- The Reliable Router schematics should be cleaned up.
- Now that one knows more the desired design methodology, the home-brew tools should be cleaned up and documented.

Afterwards, research should be done on the following areas:

- The Unique Token Protocol's token-passing should be proven correct.

- Establishing and verifying packaging and interconnect models. How does one know when to use a first-level model and when to use a second-level model?
- The bidirectional pads should be pushed to the limit. Are there tricks to help the on-chip reference look more like the line?[1]
- Cascading plesiochronous retiming was based on a unidirectional system. How would it work if a node requested non-data items?

Of course, the author would like to see the Reliable Router made fully operational and in use by the research community.

[1]Gill Pratt suggests compensating capacitors.

Bibliography

[1] James D. Allen, Patrick T. Gaughan, David E. Schimmel, and Sudhakar Yalamanchili. Ariadne – an adaptive router for fault-tolerant multicomputers. In *Proceedings of the 21st International Symposium on Computer Architecture*, pages 278–288, 1994.

[2] Kevin Bolding and William Yost. Design of a router for fault-tolerant networks. In *Proceeding of the Parallel Computer Routing and Communication Workshop*, pages 226–240, 1994.

[3] Andrew A. Chien and Jae H. Kim. Planar adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 268–277, 1992.

[4] R.R. Cordell. A 45-mbit/s cmos vlsi digital phase aligner. *IEEE Journal of Solid State Circuits*, 23(2):323–328, April 1988.

[5] William J. Dally. *A VLSI Architecture for Concurrent Data Structures.* Kluwer Academic Publishers, 1987.

[6] William J. Dally. Network and processor architecture for message-driven computers. In Suaya and Birtwhistle, editors, *VLSI and Parallel Computation.* Morgan Kaufmann, 1990.

[7] William J. Dally and Hiromichi Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[8] William J. Dally, John S. Keen, and Michael D. Noakes. The j-machine multicomputer: Architecture and evaluation. In *Compcon Spring 93*, pages 183–188. IEEE Computer Society Press, February 1993.

[9] William J. Dally and Charles L. Seitz. The torus routing chip. *Distributed Computing*, 1:187–196, 1986.

[10] William J. Dally and Charles L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–53, May 1987.

[11] Larry R. Dennison. Reliable interconnection networks for parallel computers. Technical Report AI-TR 1294, MIT Artificial Intelligence Laboratory, 545 Tech. Sq., Cambridge MA 02139, October 1991.

[12] Larry R. Dennison, Whay S. Lee, and William J. Dally. High-performance bidirectional signalling in VLSI systems. In *Proceedings of the Symposium on Research on Integrated Systems*, pages 300–319. MIT Press, March 1993.

[13] José Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers: Design methodologies. In *Proceedings of PARLE '91*, pages 390–405. Springer-Verlag, June 1991.

[14] José Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. In *Proceedings of the International Conference on Parallel Processing*, August 1994.

[15] Ian Eslick, André DeHon, and Thomas F. Knight, Jr. Guaranteeing idempotence for tightly-coupled, fault-tolerant networks. In *Proceeding of the Parallel Computer Routing and Communication Workshop*, pages 215–225, 1994.

[16] P.T. Gaughan and Y. Sudhakar. A family of fault-tolerant routing protocols for direct multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(5):482–497, May 1995.

[17] Cristopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. In *Proceedings of the 19th International Symposium on Computer Architecture*, pages 278–287, 1992.

[18] Cristopher J. Glass and Lionel M. Ni. Fault-tolerant wormhole routing in meshes. 1993.

[19] Thomas F. Knight and A. Krymm. A self-terminating low-voltage swing cmos output driver. *IEEE Journal of Solid-State Circuits*, 23(2):457–464, April 1988.

[20] Kevin Lam, Larry Dennison, and William Dally. Simultaneous bidirectional signaling for ic systems. In *Proceedings of the 1990 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 430–433. IEEE, September 1990. Presented at same conference.

[21] K. Lee, S. Kim, G. Ahn, and D. Jeong. A cmos serial link for fully duplexed data communication. *IEEE Journal of Solid-State Circuits*, 30(4):353–364, April 1995.

[22] Whay S. Lee, Larry R. Dennison, and William J. Dally. Implementation of a simultaneous bidirectional i/o pad for inter-chip communication. November 1991.

[23] Daniel H. Linder and Jim C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, 40(1), January 1991.

[24] D. G. Messerschmitt. Synchronization in digital system design. *IEEE Journal on Selected Areas in Communications*, 8(8), October 1990.

[25] R. Mooney, C. Dike, and S. Borkar. High speed bidirectional signalling scheme. In *Proceedings of the 1995 Integrated Solid State Circuits Conference*, pages 38–39, 1995.

[26] Michael D. Noakes, Deborah A. Wallach, and William J. Dally. The J-Machine multicomputer: An architectural evaluation. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 224–235, San Diego, California, May 1993. IEEE.

[27] Peter R. Nuth and William J. Dally. The j-machine network. In *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*, pages 420–423. IEEE, October 1992.

[28] Gustavo D. Pifarré, Luis Gravano, Sergio A. Felperin, and Jorge L. C. Sanz. Fully adaptive minimal deadlock-free packet routing in hypercubes, meshes, and other networks: Algorithms and simulations. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):247–263, March 1994.

[29] Gill A. Pratt and Steven A. Ward. Rationally clocked communication. Manuscript obtained from author.

[30] Randall D. Rettberg and Lance A. Glasser. Digital phase adjustment. United States Patent No. 4,700,347, Oct 13., 1987.

[31] Steve Scott and Greg Thorson. Optimized routing in the cray t3d. In *Proceeding of the Parallel Computer Routing and Communication Workshop*, pages 281–294, 1994.

[32] Charles R. Seitz and Wen-King Su. A family of routing and communication chips based on the mosaic. In *Proceedings of the Symposium on Research on Integrated Systems*, pages 320–337. MIT Press, March 1993.

[33] D.P Sierwiorek and R.S. Swarz. *Reliable Computer Systems: Design and Evaluation, Second Edition.* Digital Press, Bedford, MA, second edition, 1992.

[34] W. K. Stewart and Steven Ward. A solution to a special case of the synchronization problem. *IEEE Transactions on Computers*, 37(1), January 1988.

[35] T. Takahasi et al. A cmos gate array with 600mb/s simultaneous bidirectional i/o circuits. In *Proceedings of the 1995 Integrated Solid State Circuits Conference*, pages 40–41, 1995.

[36] A. S. Tanenbaum. *Computer Networks, Second Edition.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[37] Thucydides Xanthopoulos. Fault tolerant adaptive routing in multicomputer networks. Master's thesis, MIT, 545 Technology Sq., Cambridge, MA, May 1992.

Appendix A

Reliable Router Schematics

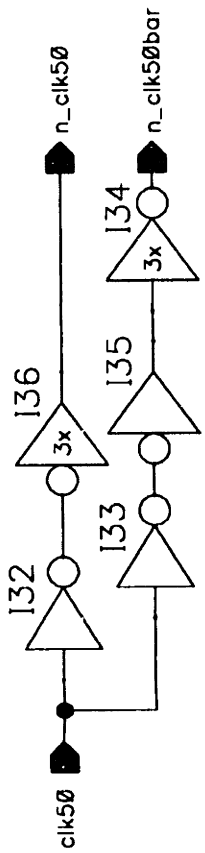


Figure A-1: Library arbiter, cell clkbuf

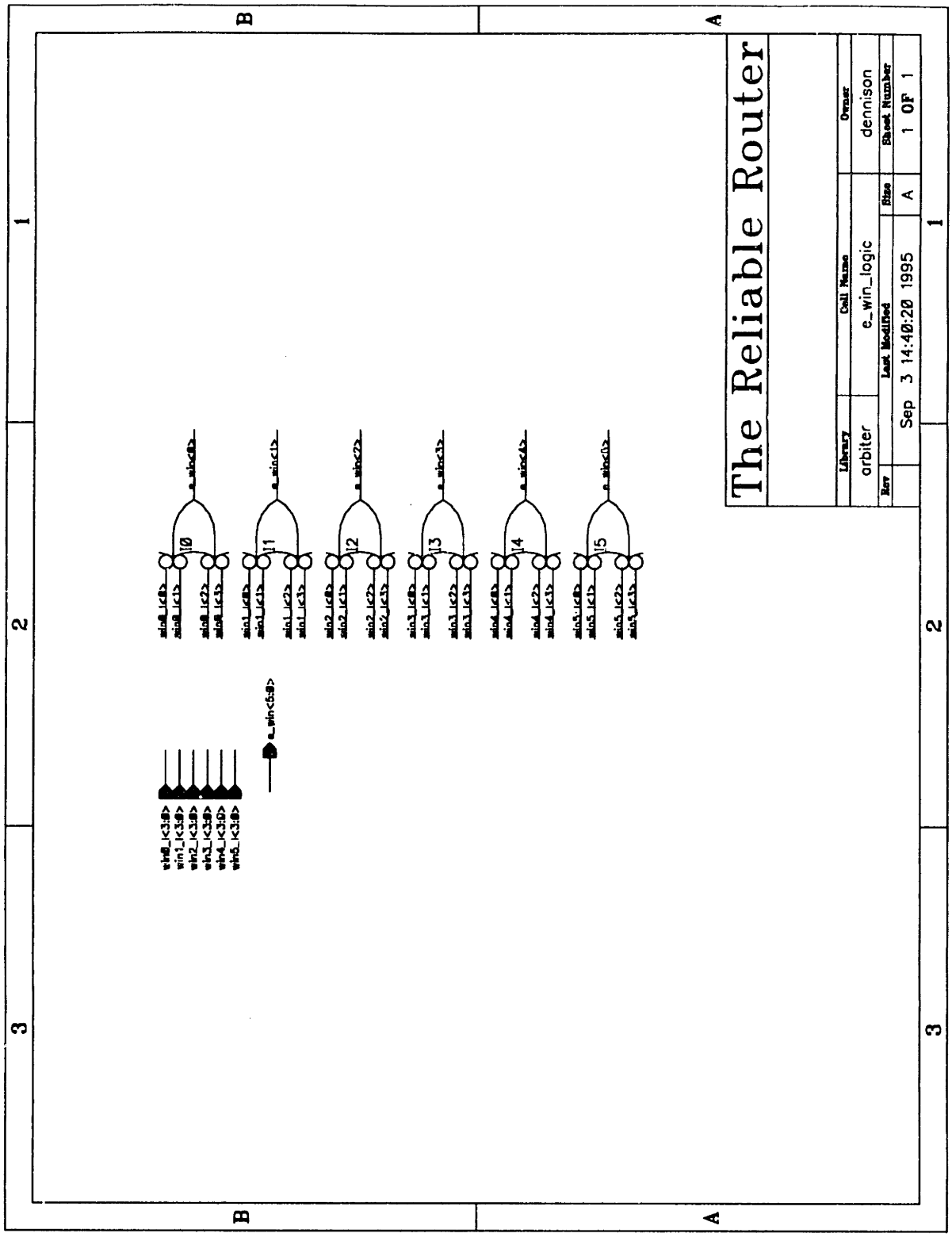


Figure A-2: Library arbiter, cell e_win_logic

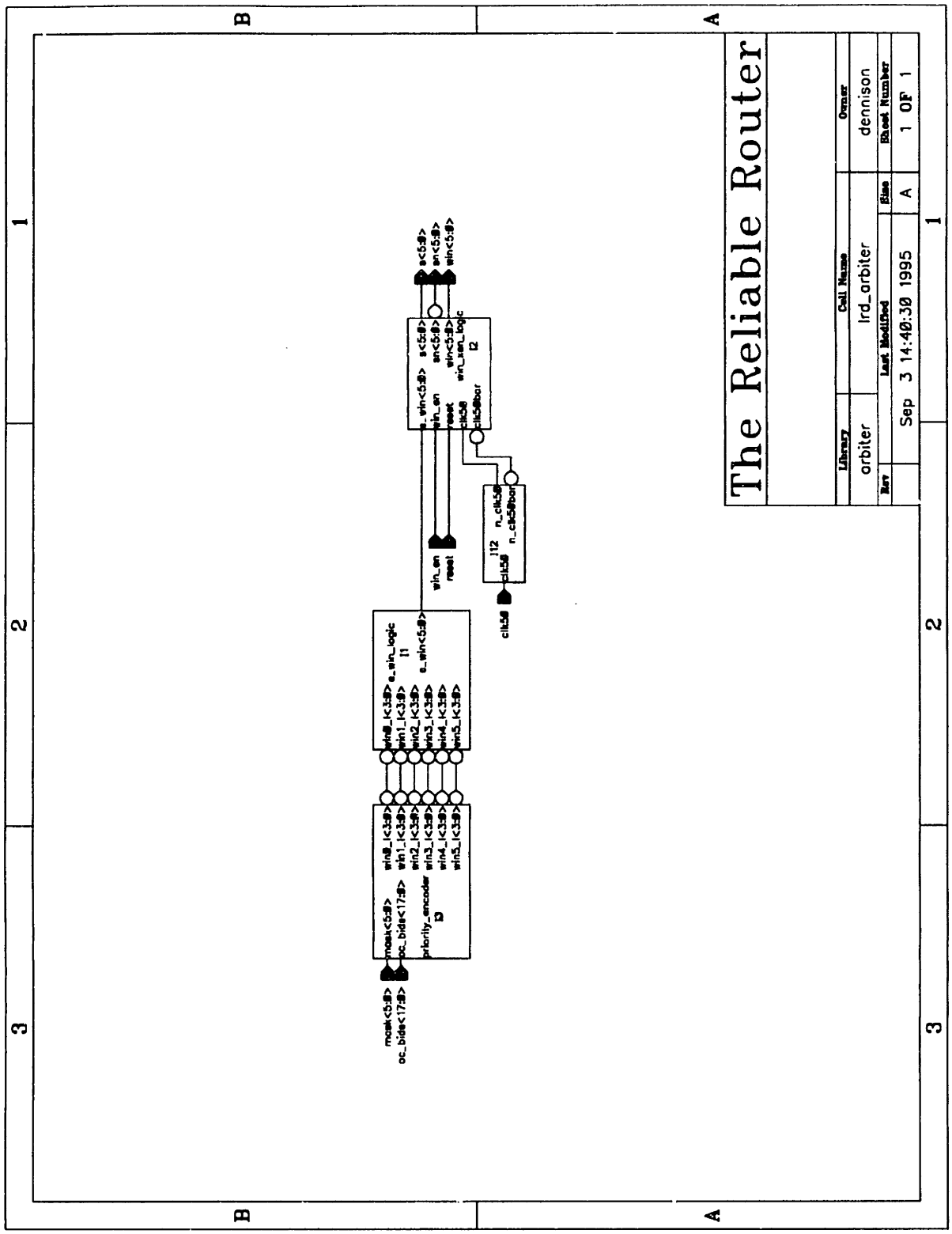


Figure A-3: Library arbiter, cell lrd_arbiter

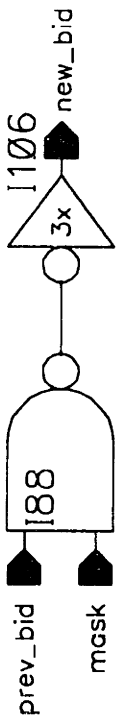


Figure A-4: Library arbiter, cell new_bid_and2

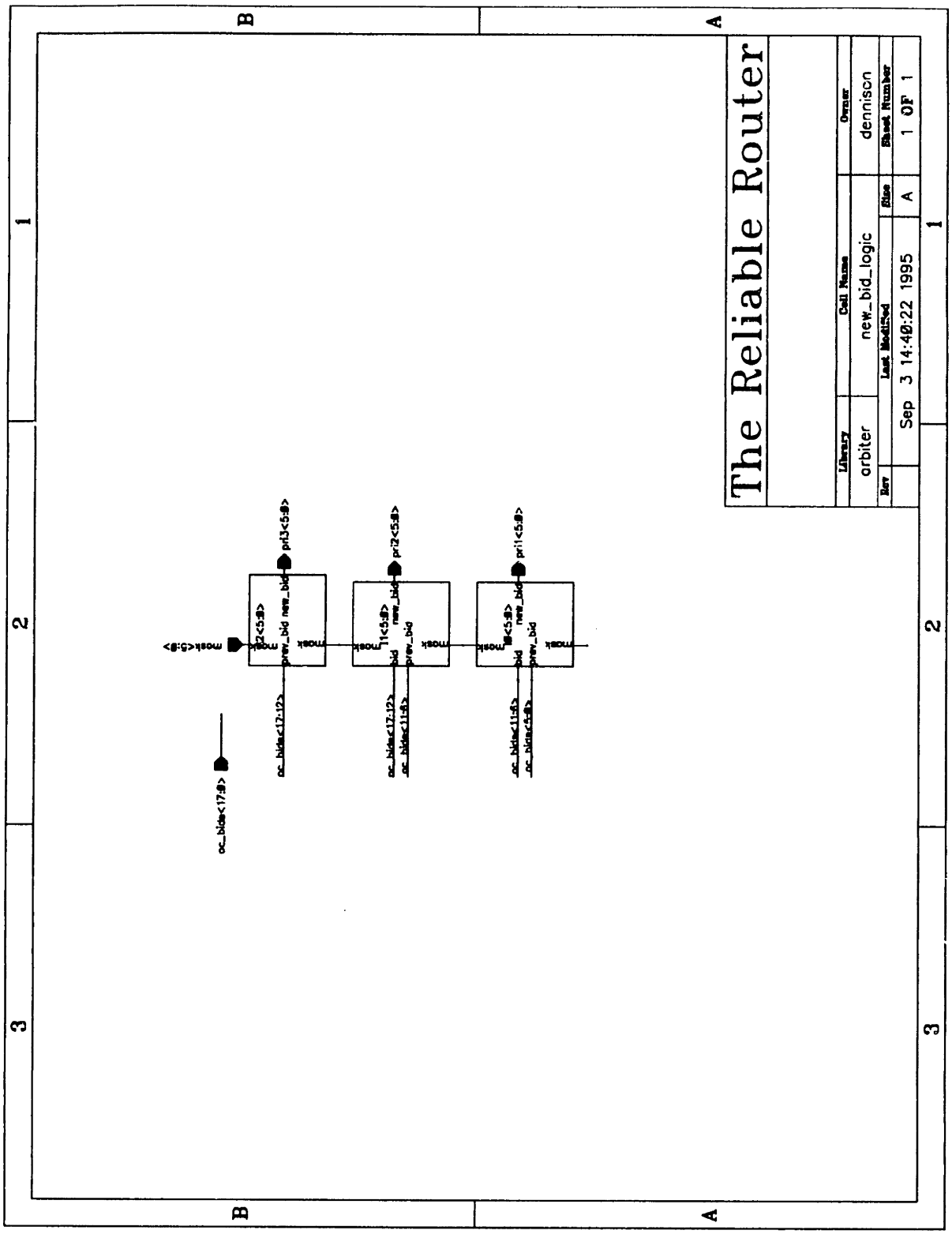


Figure A-5: Library arbiter, cell new_bid_logic

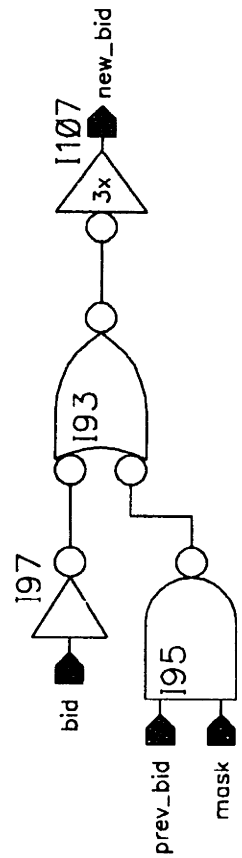


Figure A-6: Library arbiter, cell new_bid_nand2_invert

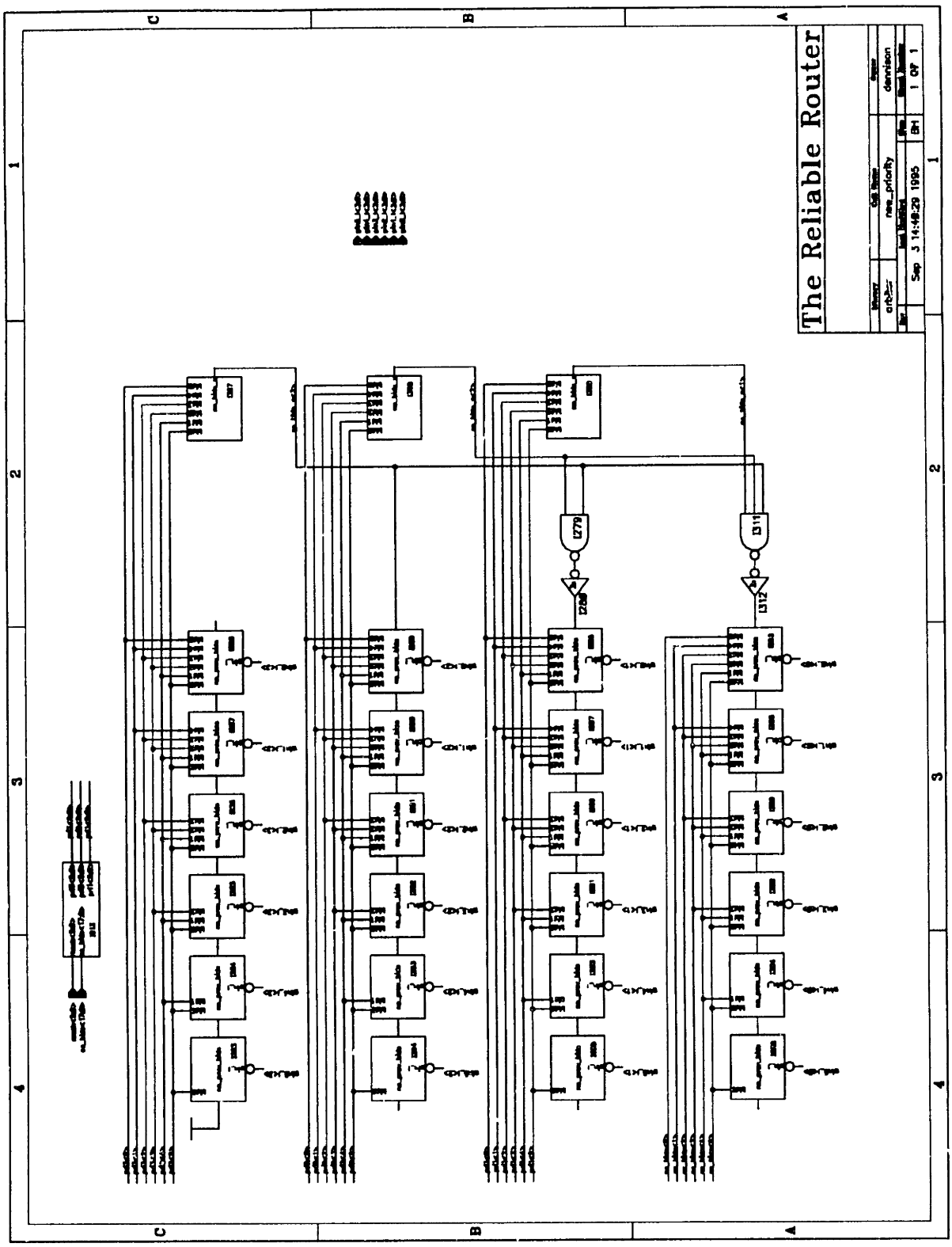


Figure A-7: Library arbiter, cell new_priority

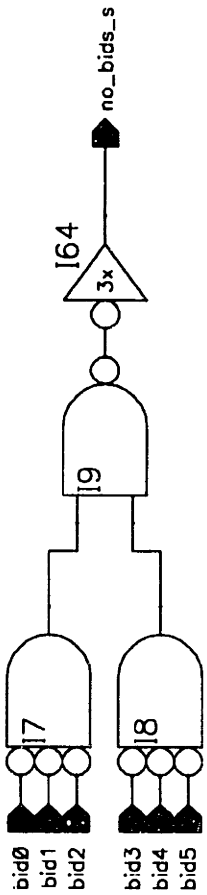


Figure A-8: Library arbiter, cell no_bids

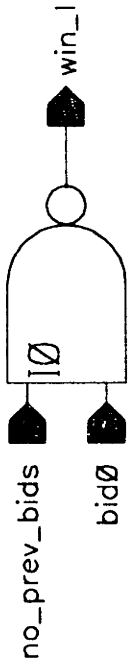


Figure A-9: Library arbiter, cell win0

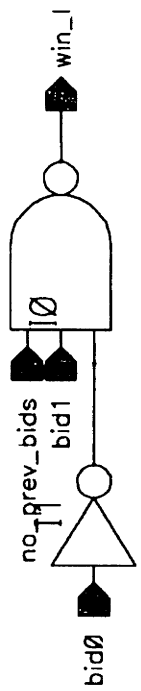


Figure A-10: Library arbiter, cell win1

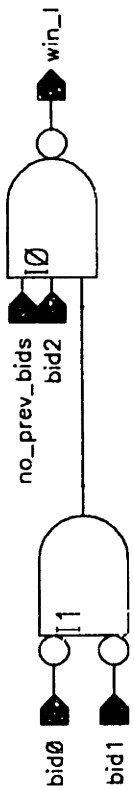


Figure A-11: Library arbiter, cell win2

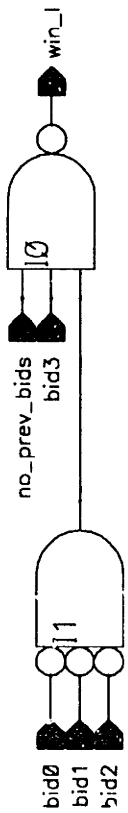


Figure A-12: Library arbiter, cell win3

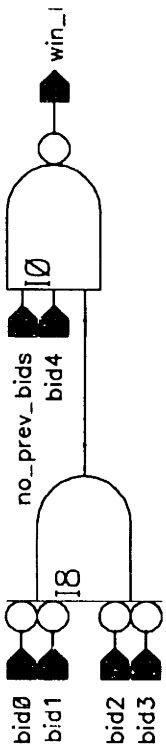


Figure A-13: Library arbiter, cell win4

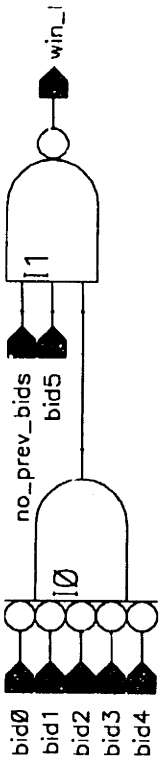


Figure A-14: Library arbiter, cell win5

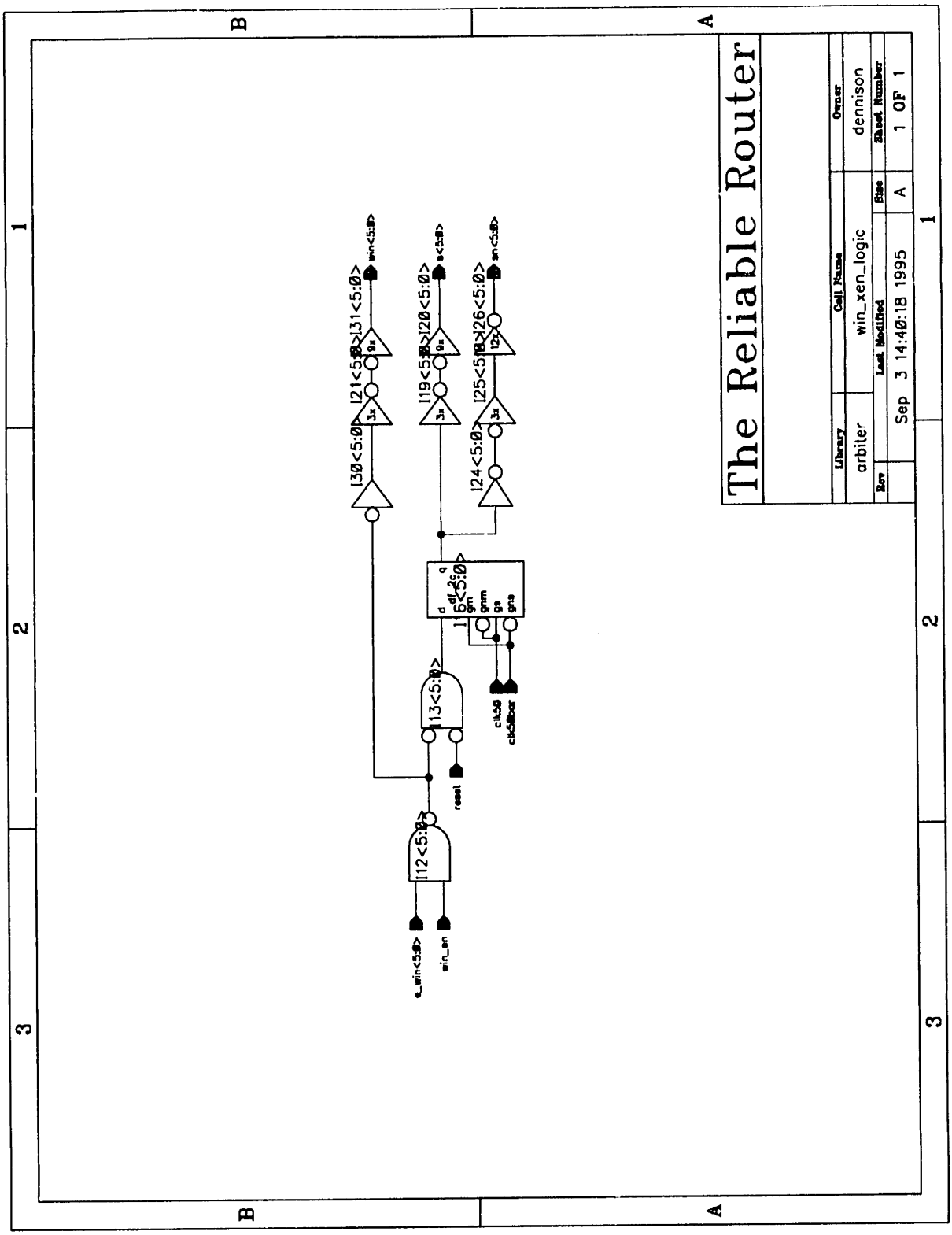


Figure A-15: Library arbiter, cell win_xen_logic

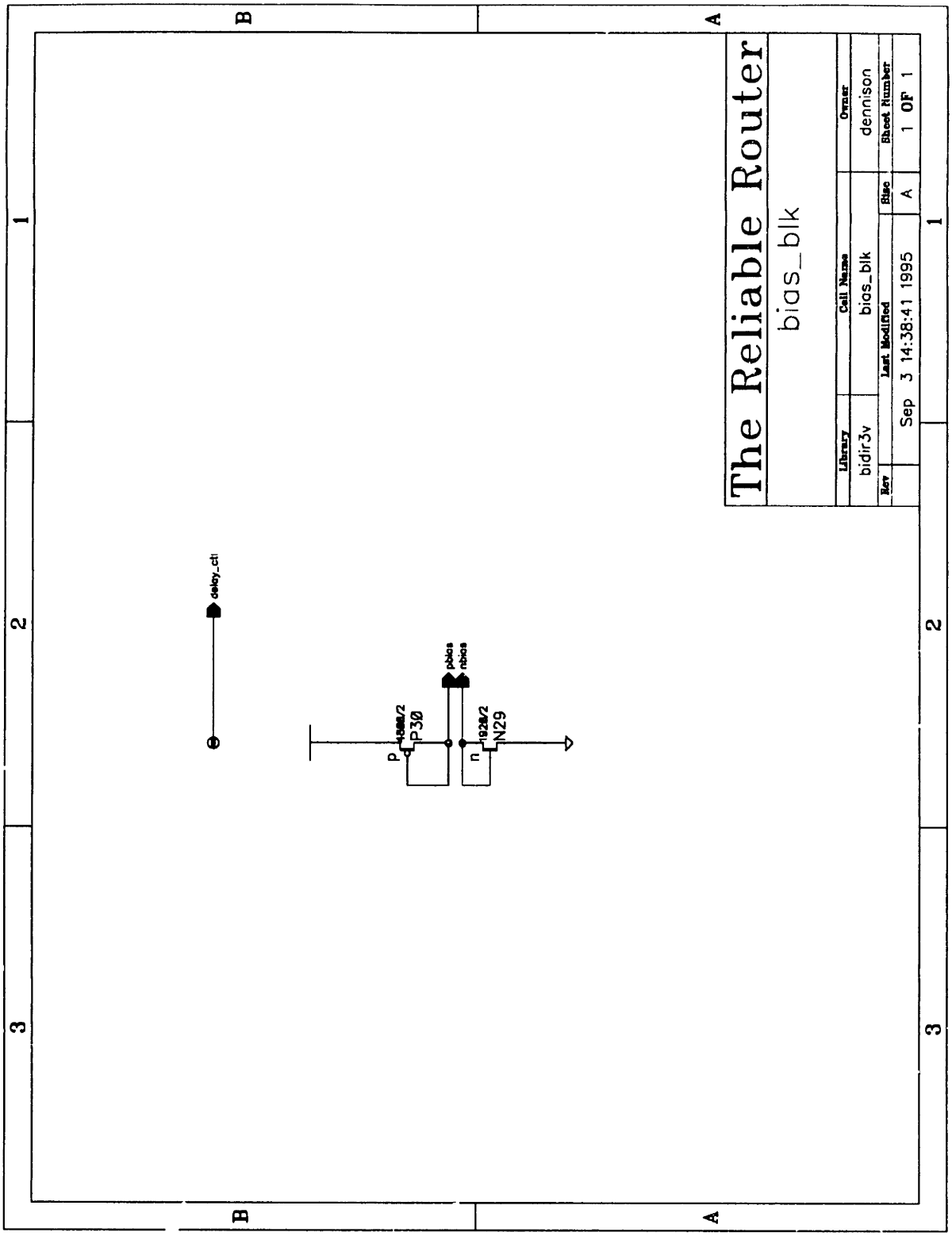


Figure A-16: Library bidir3v, cell bias_blk

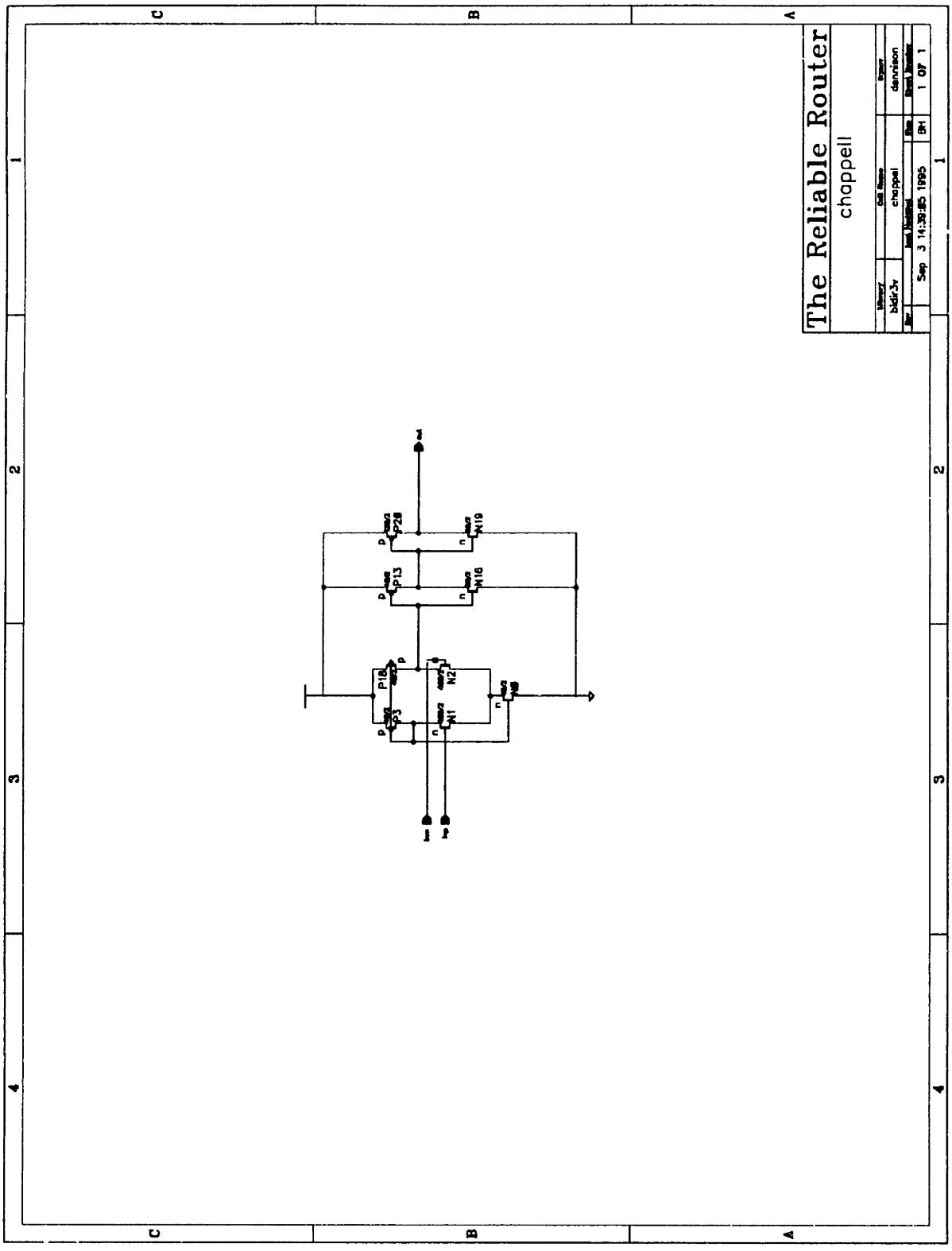


Figure A-17: Library bidir3v, cell chappel

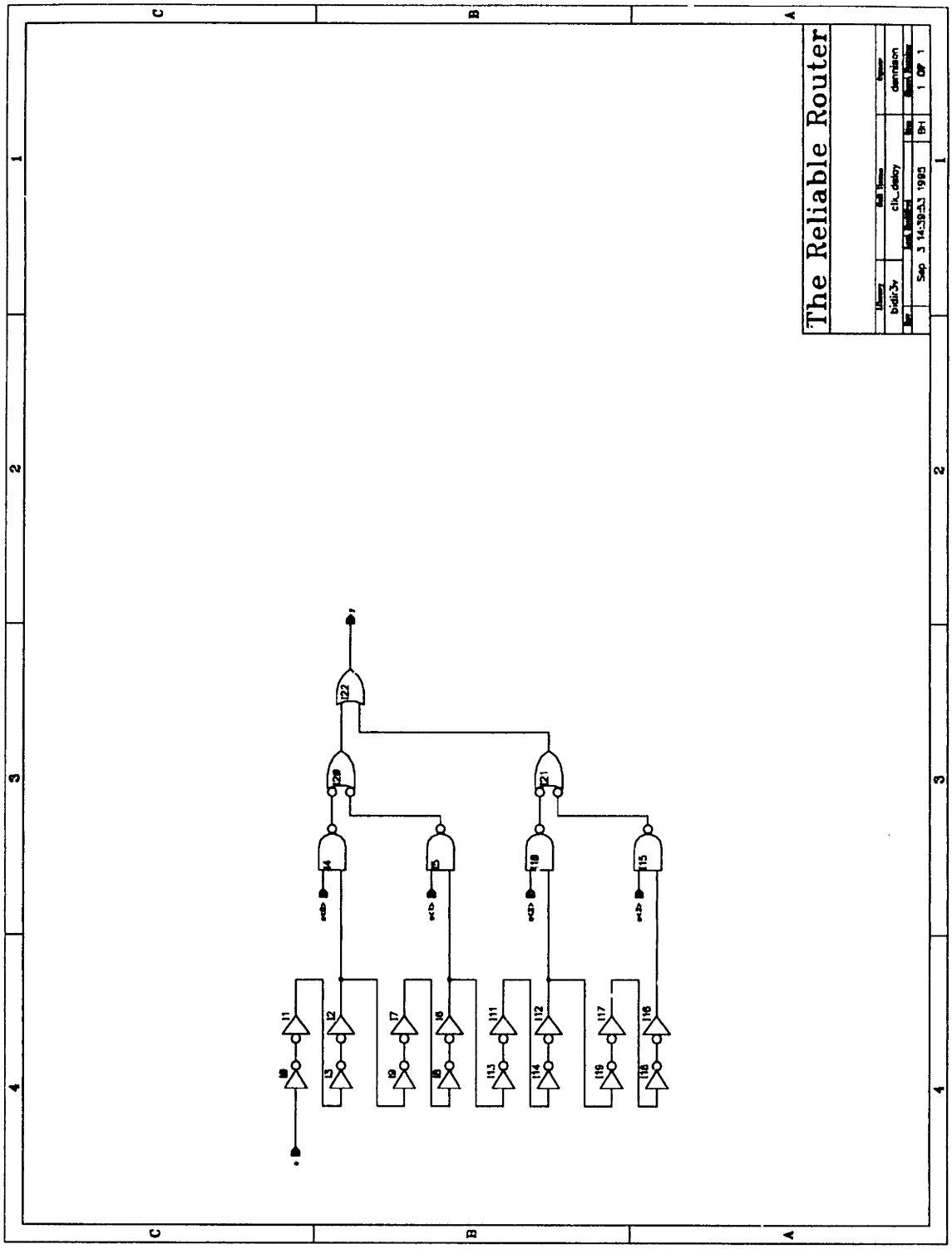


Figure A-18: Library bidir3v, cell clk_delay

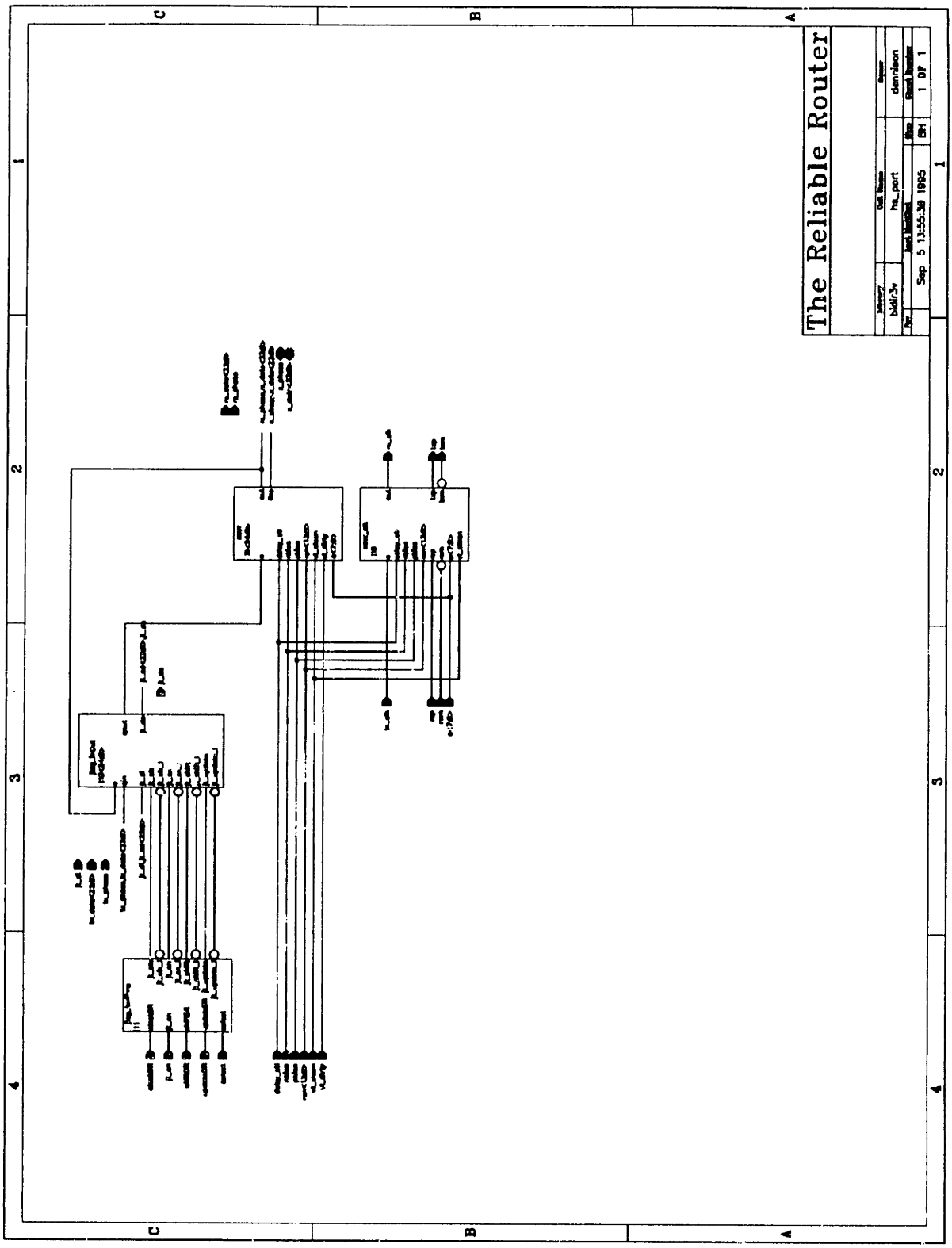


Figure A-19: Library bidir3v, cell hs_port

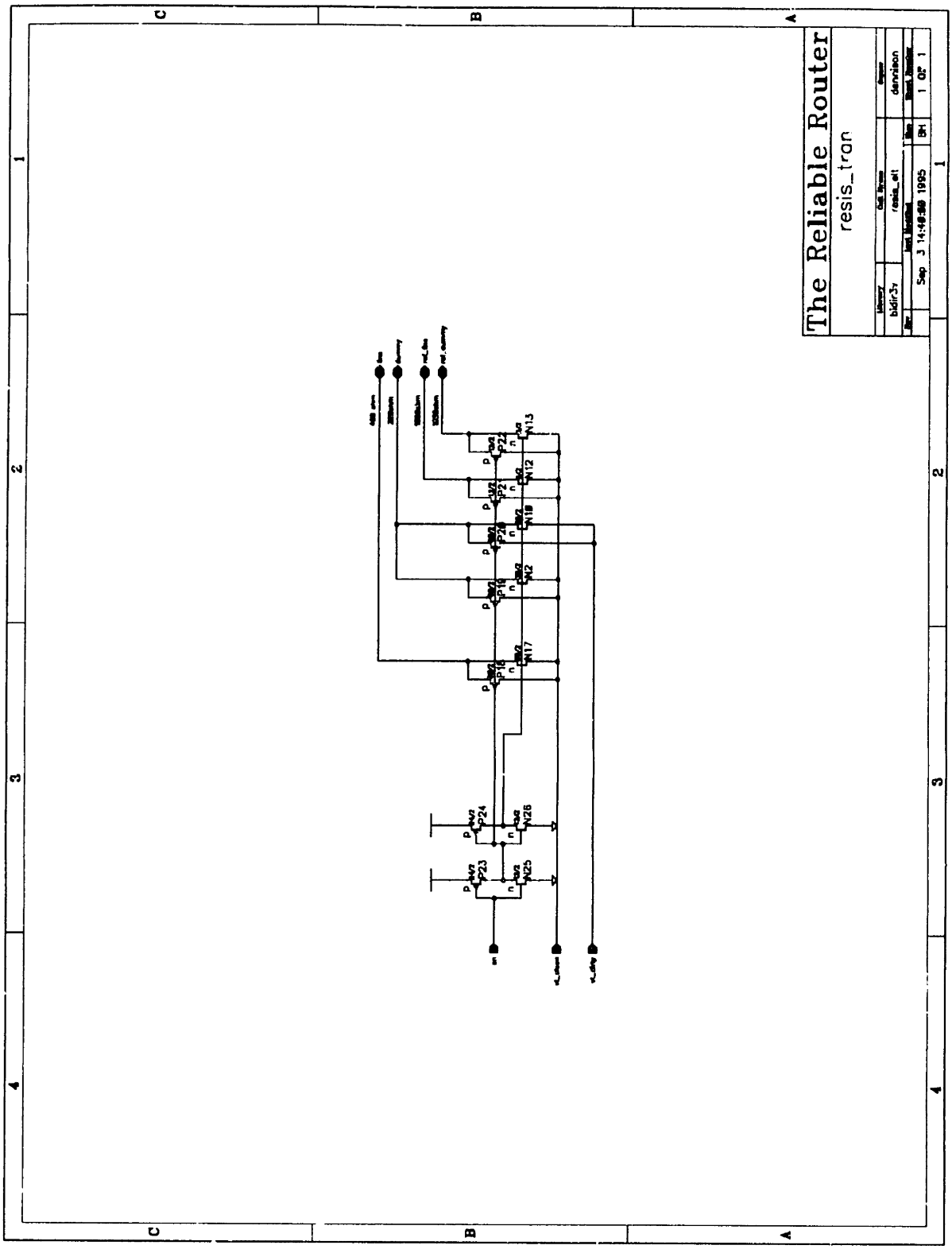


Figure A-20: Library bidir3v, cell resis_elt

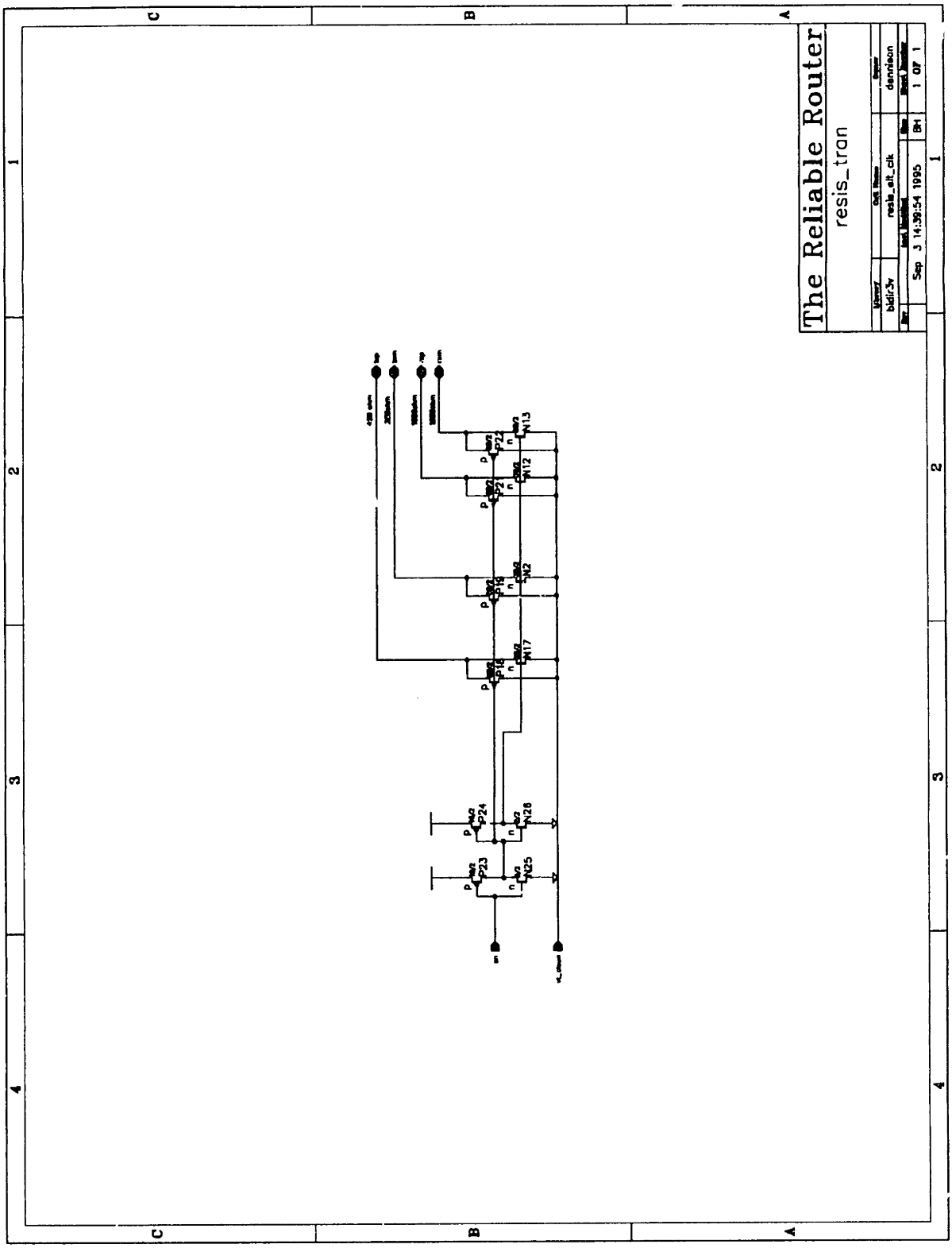


Figure A-21: Library bidir3v, cell resis_elt_clk

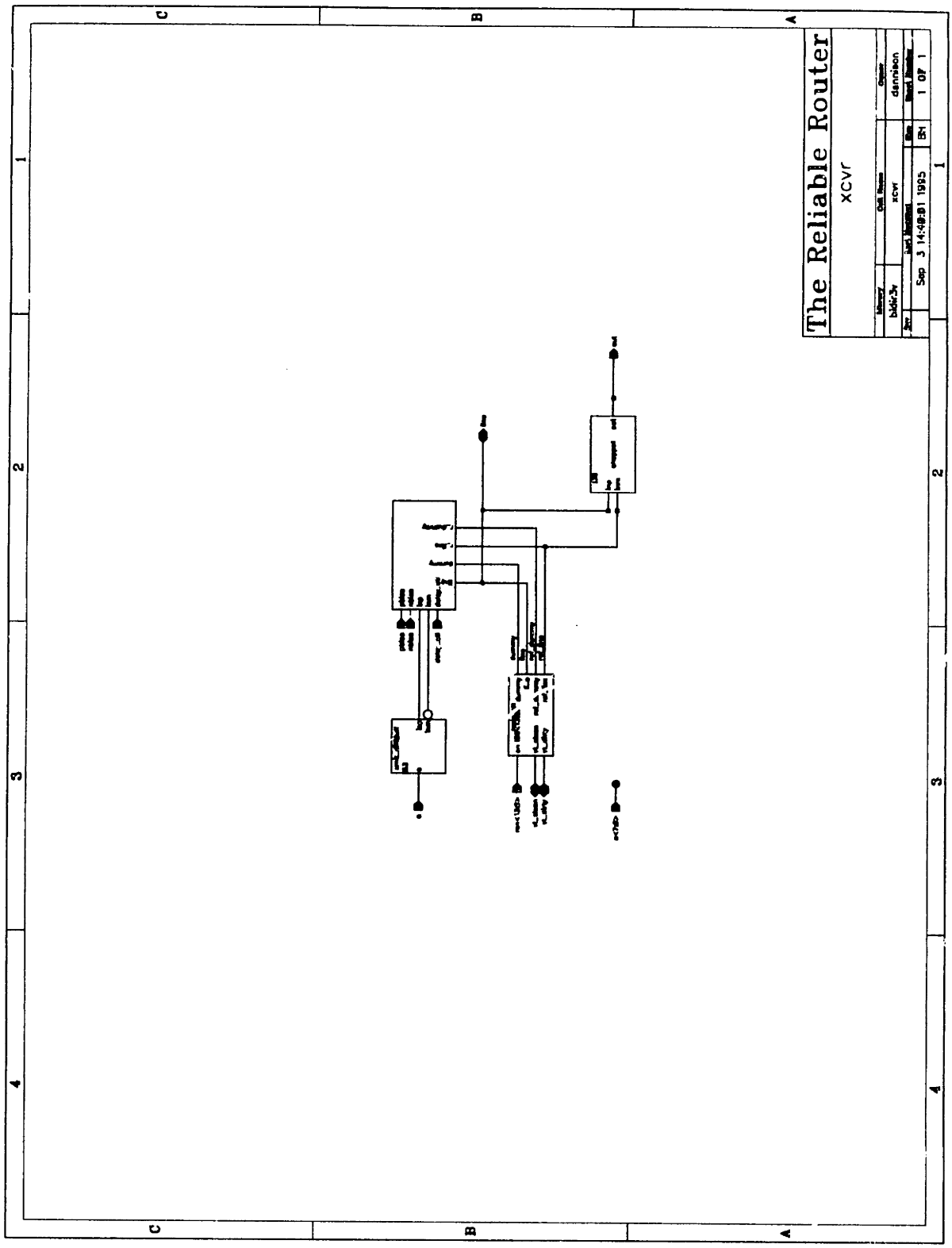


Figure A-22. Library bidir3v, cell xcvr

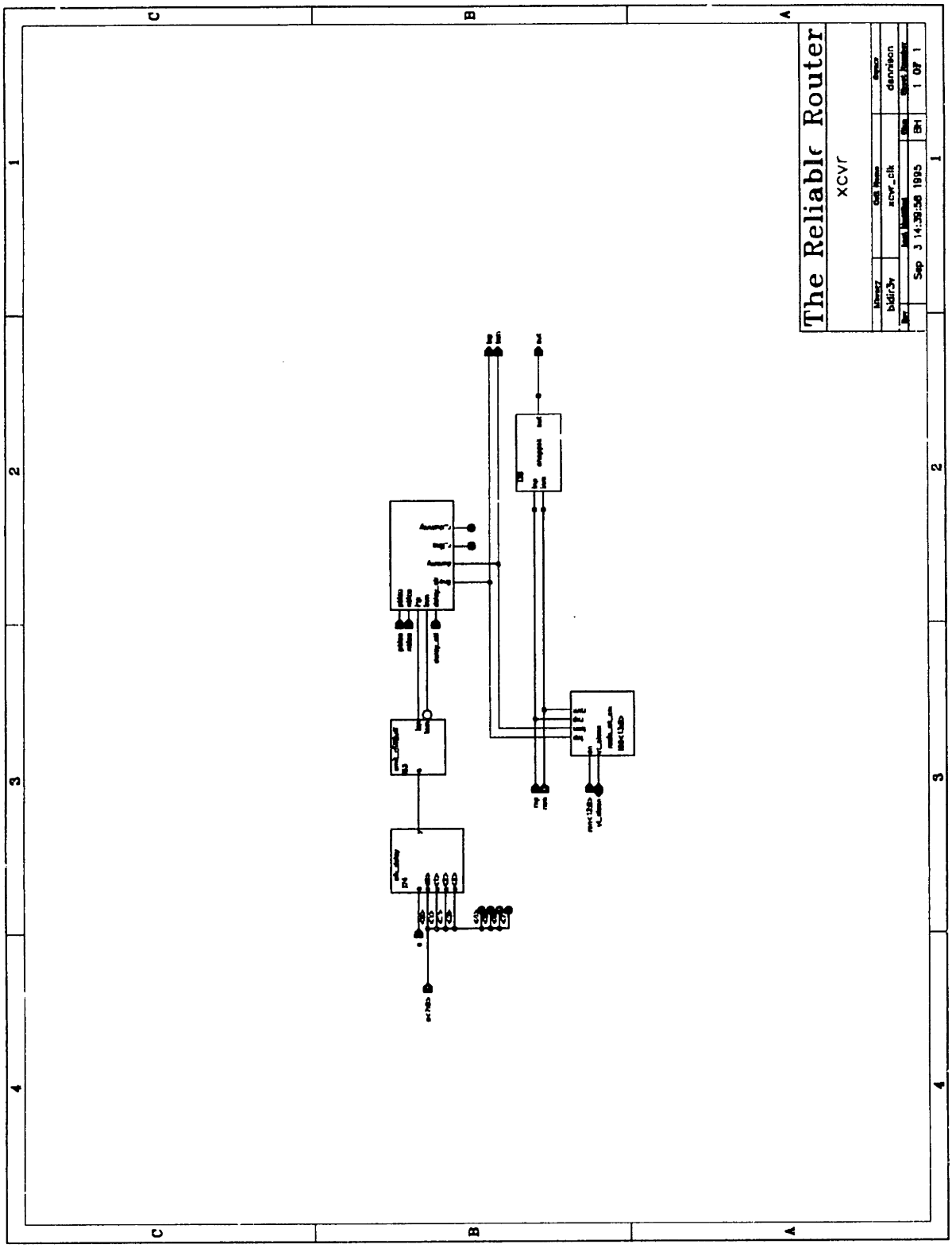


Figure A-23: Library bidir3v, cell xcvr_clk

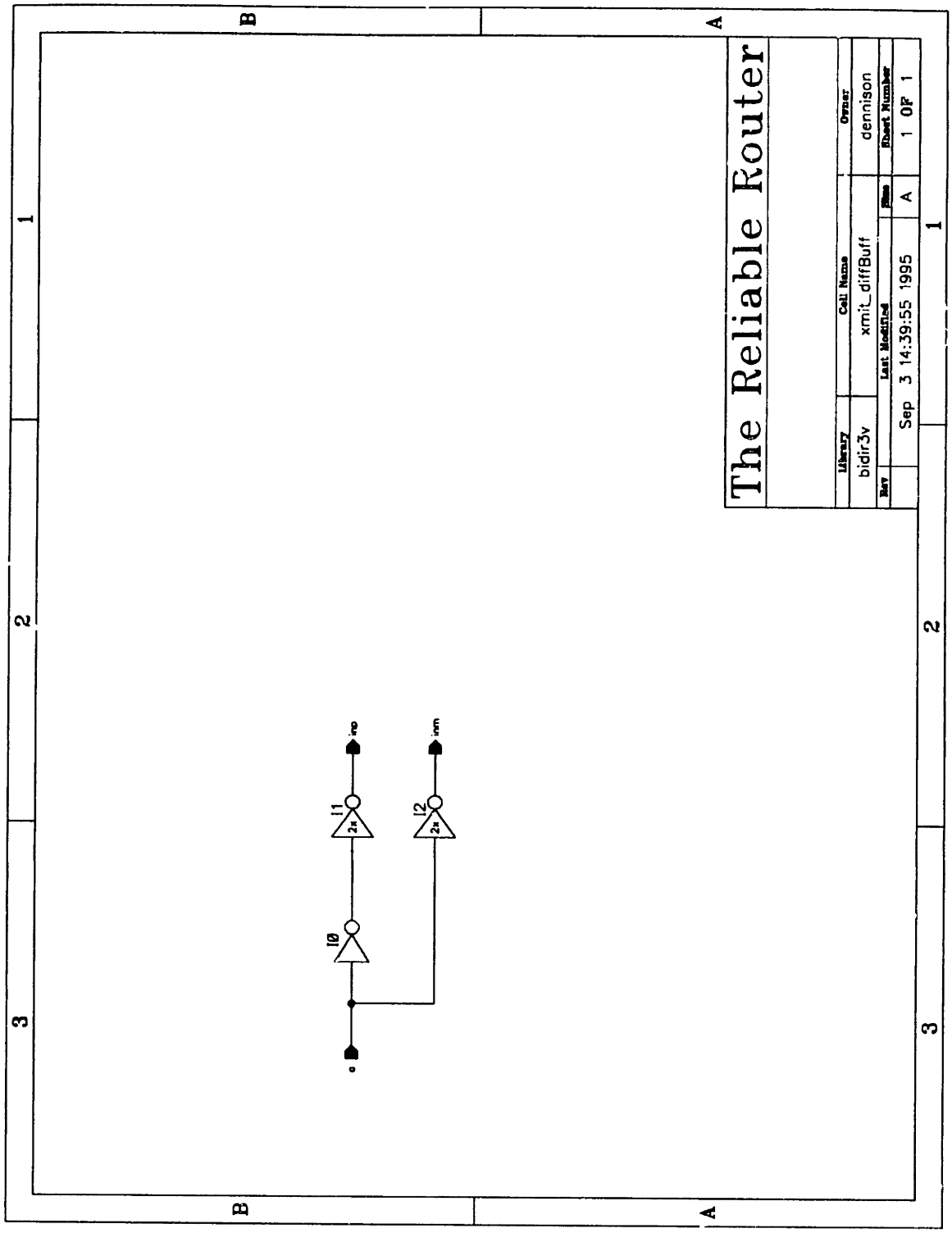


Figure A-24: Library bidir3v, cell xmit_diffBuff

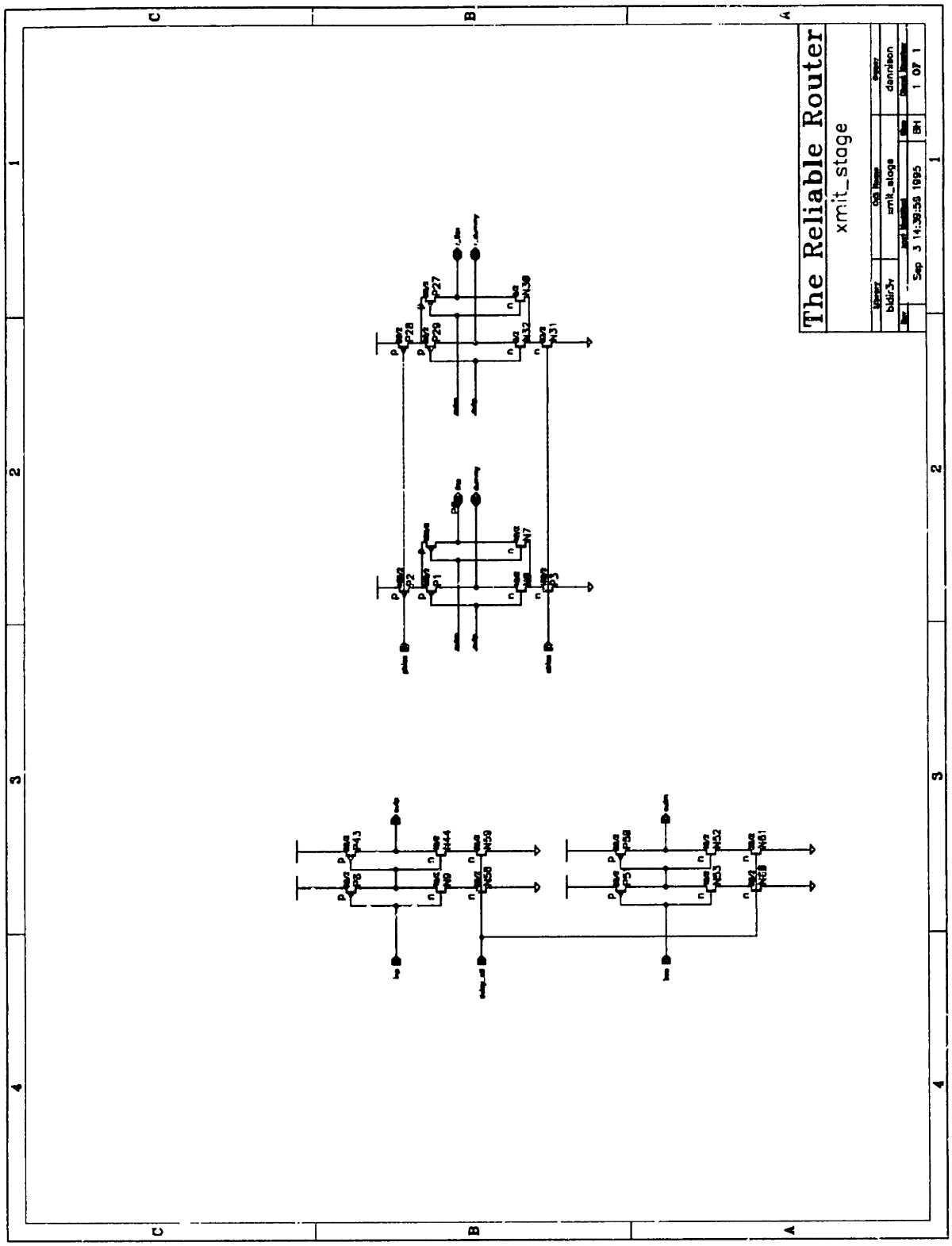


Figure A-25: Library bidir3v, cell xmit_stage

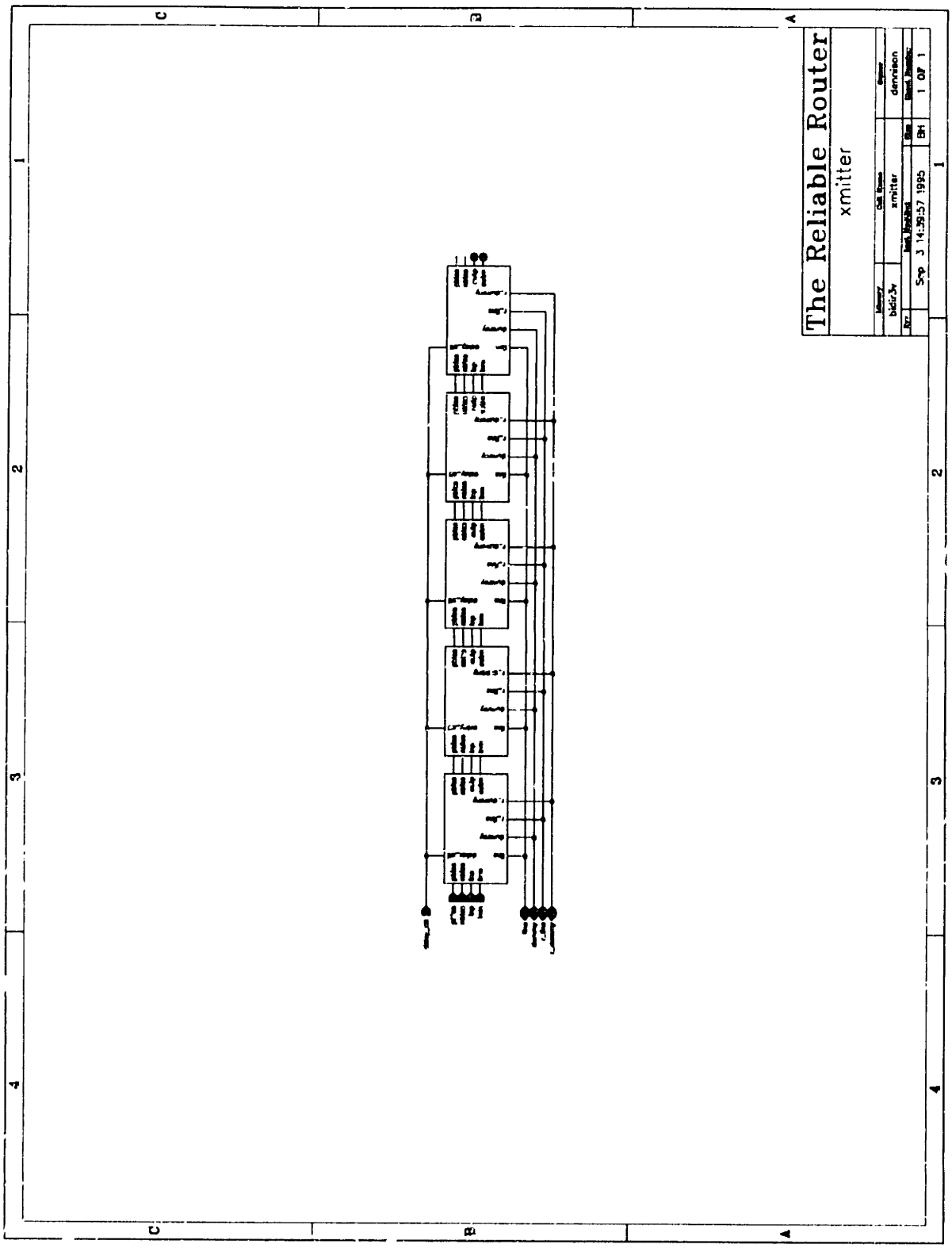


Figure A-26: Library bidir3v, cell xmitter

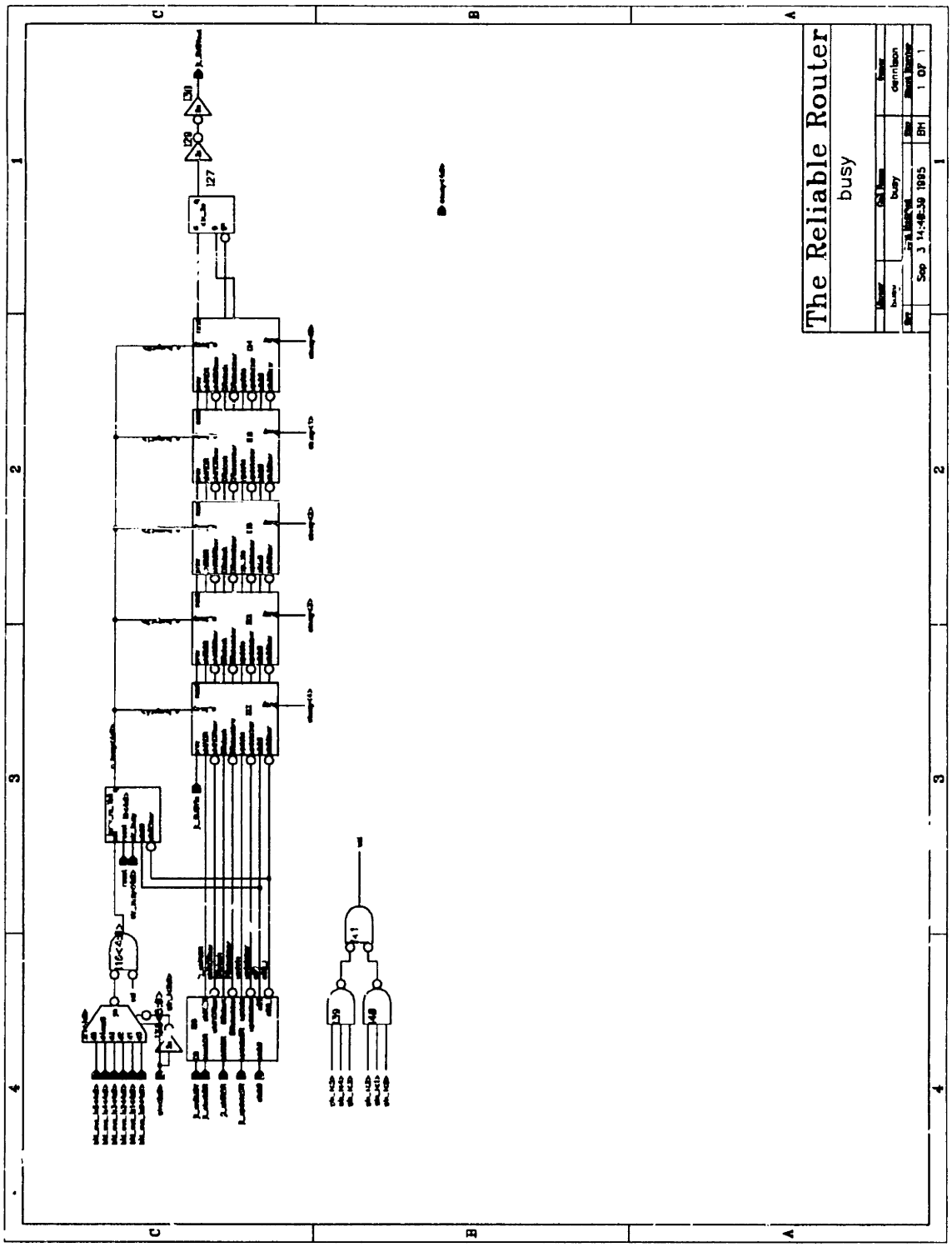


Figure A-27: Library busy, cell busy

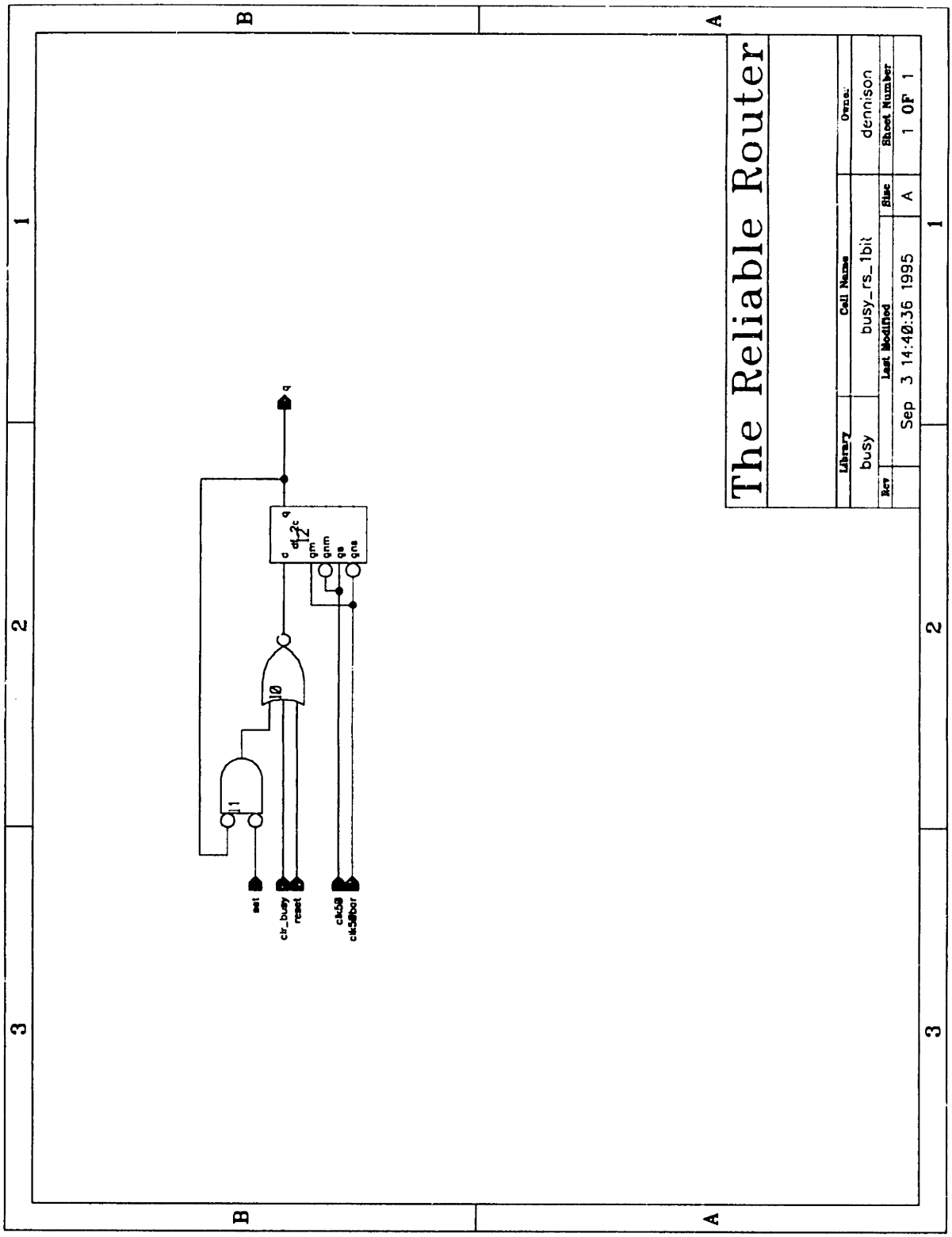


Figure A-28: Library busy, cell busy_rs_1bit

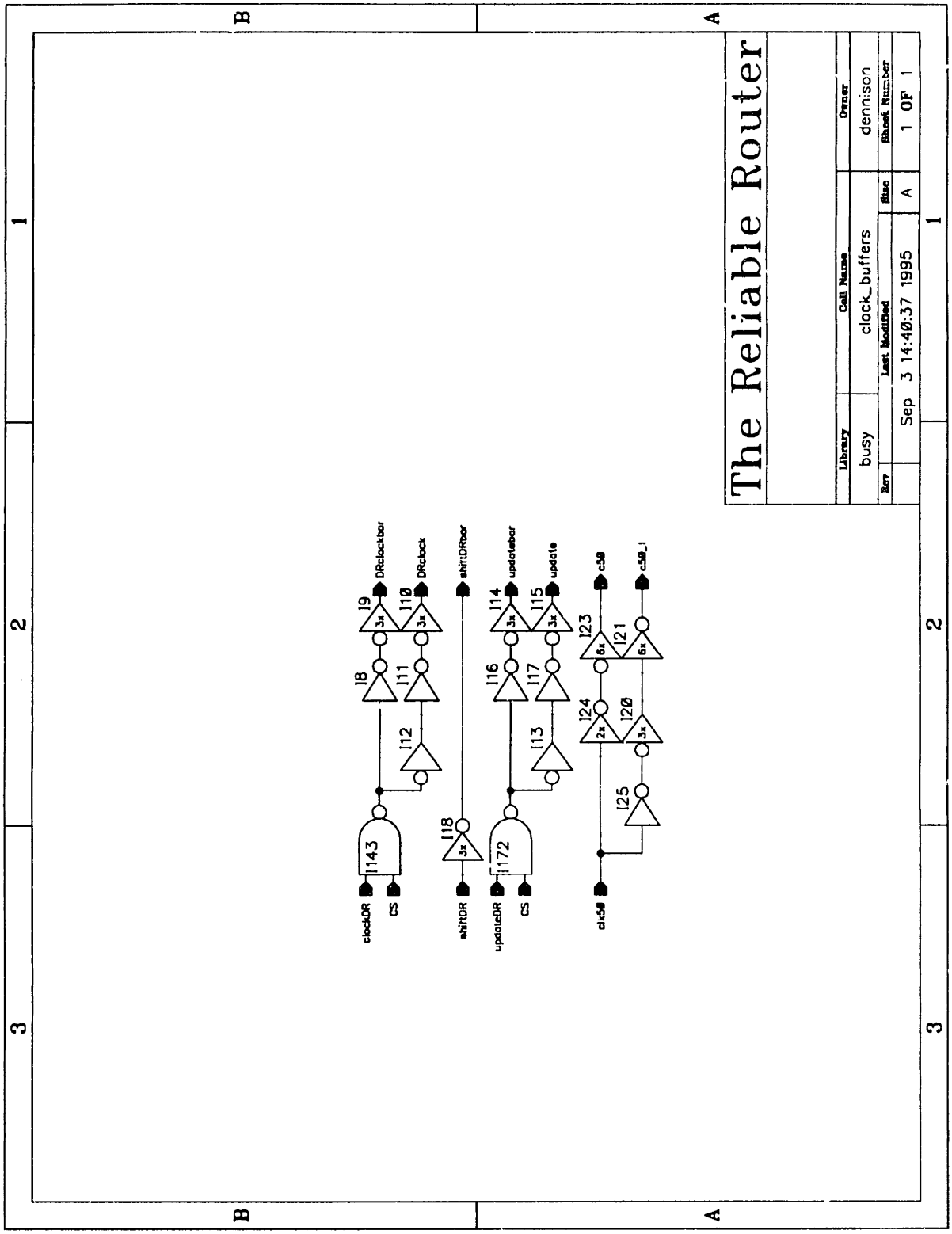


Figure A-29: Library busy, cell clock_buffers

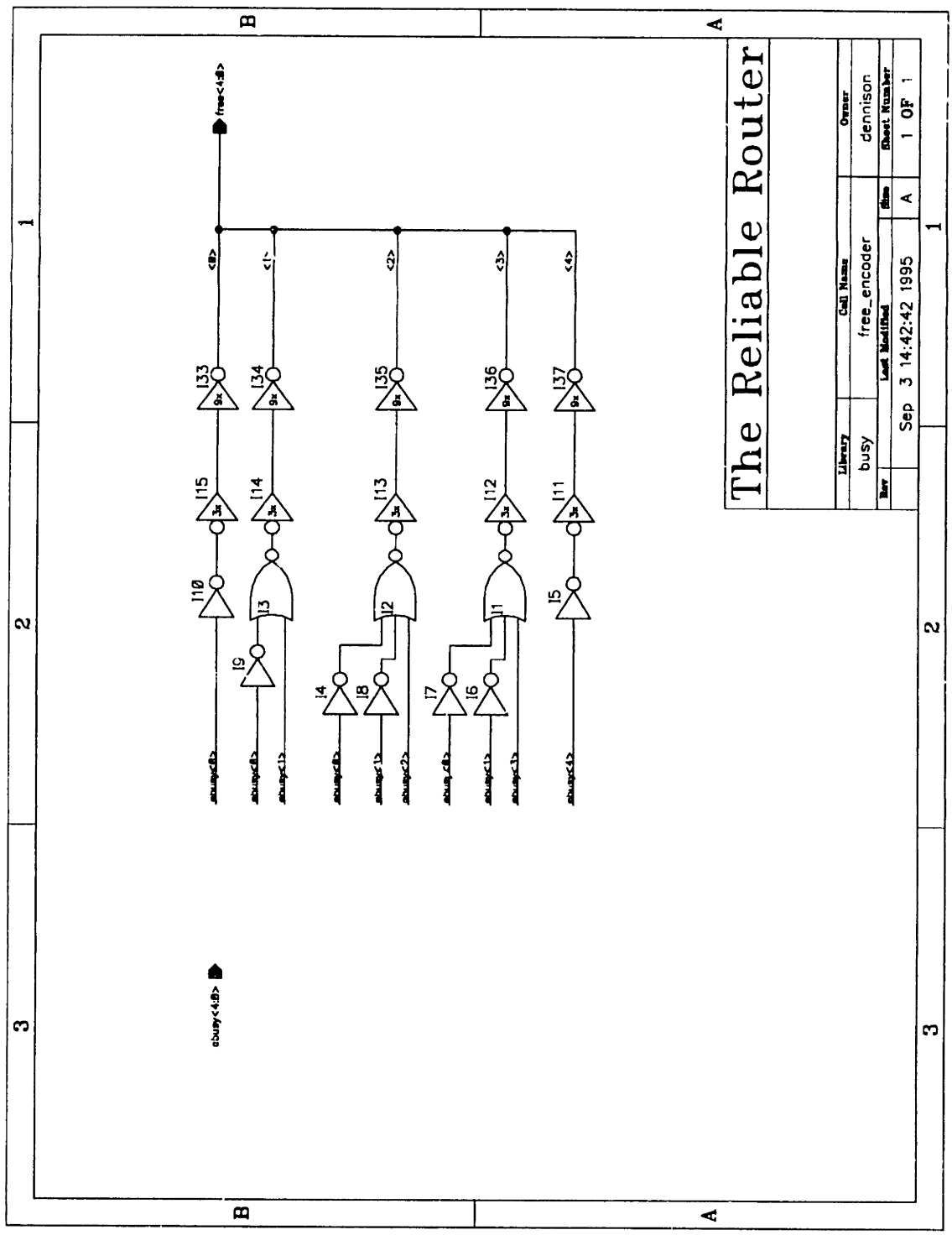


Figure A-30: Library busy, cell free_encoder

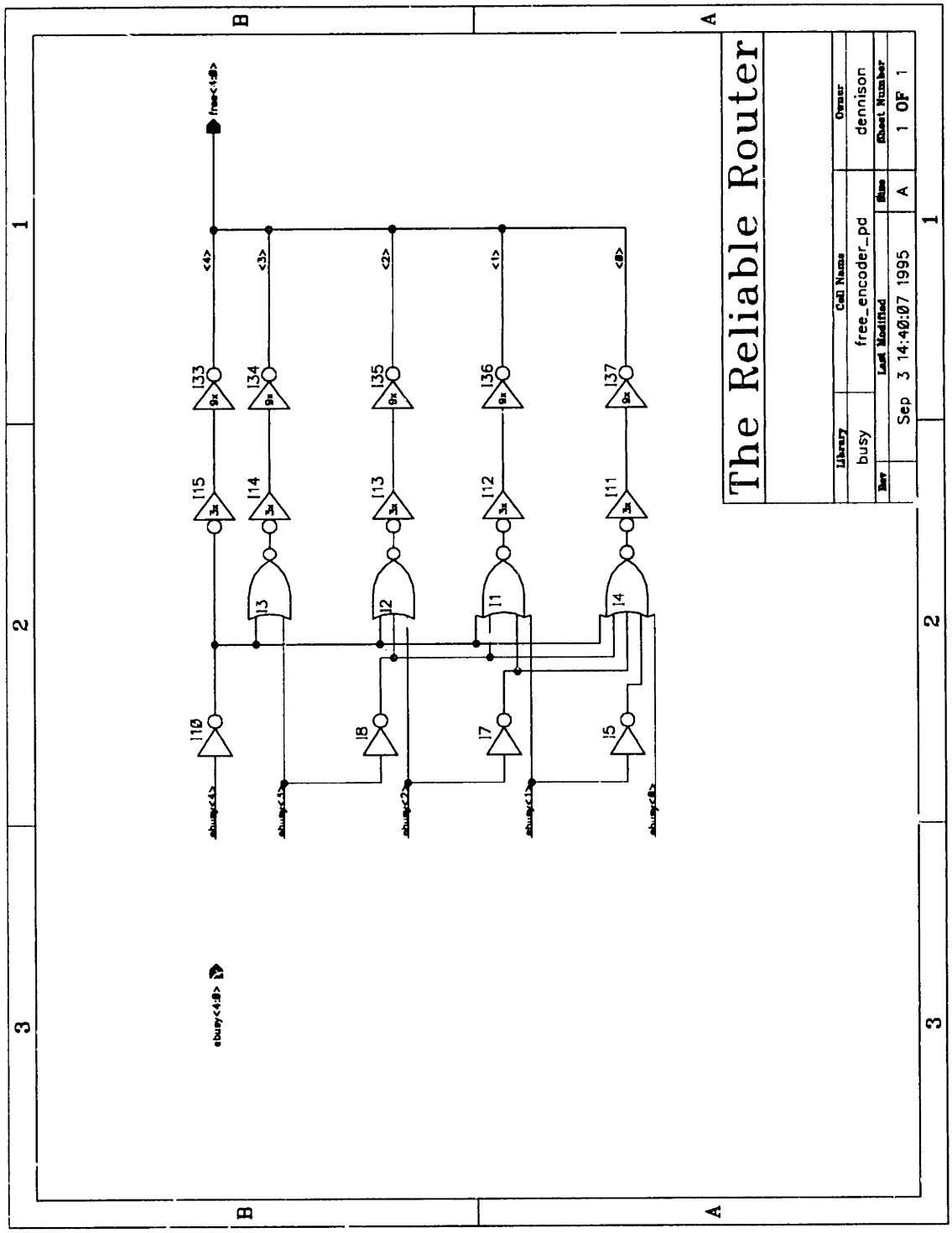


Figure A-31: Library busy, cell free_encoder_pd

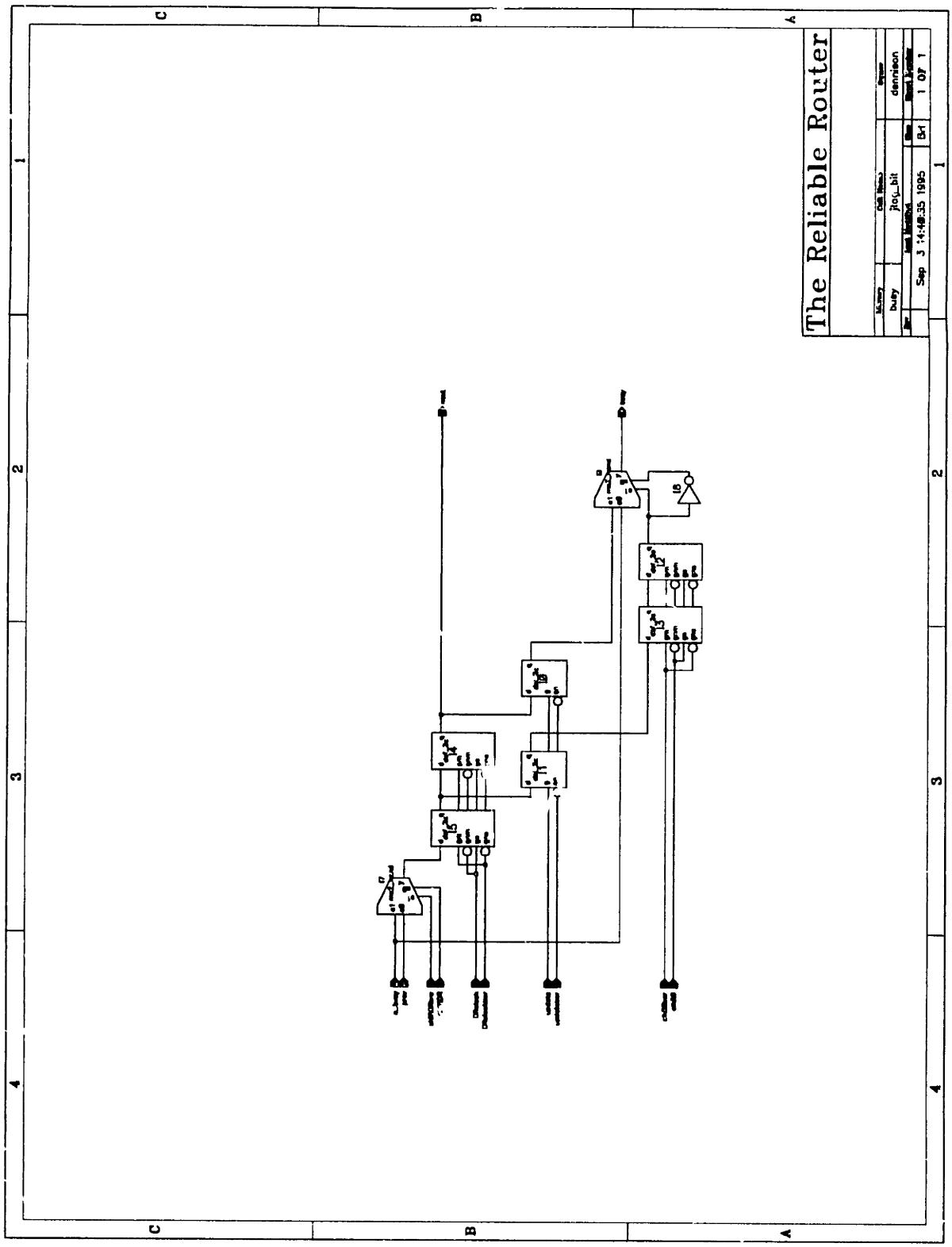


Figure A-32: Library busy, cell jtag_bit

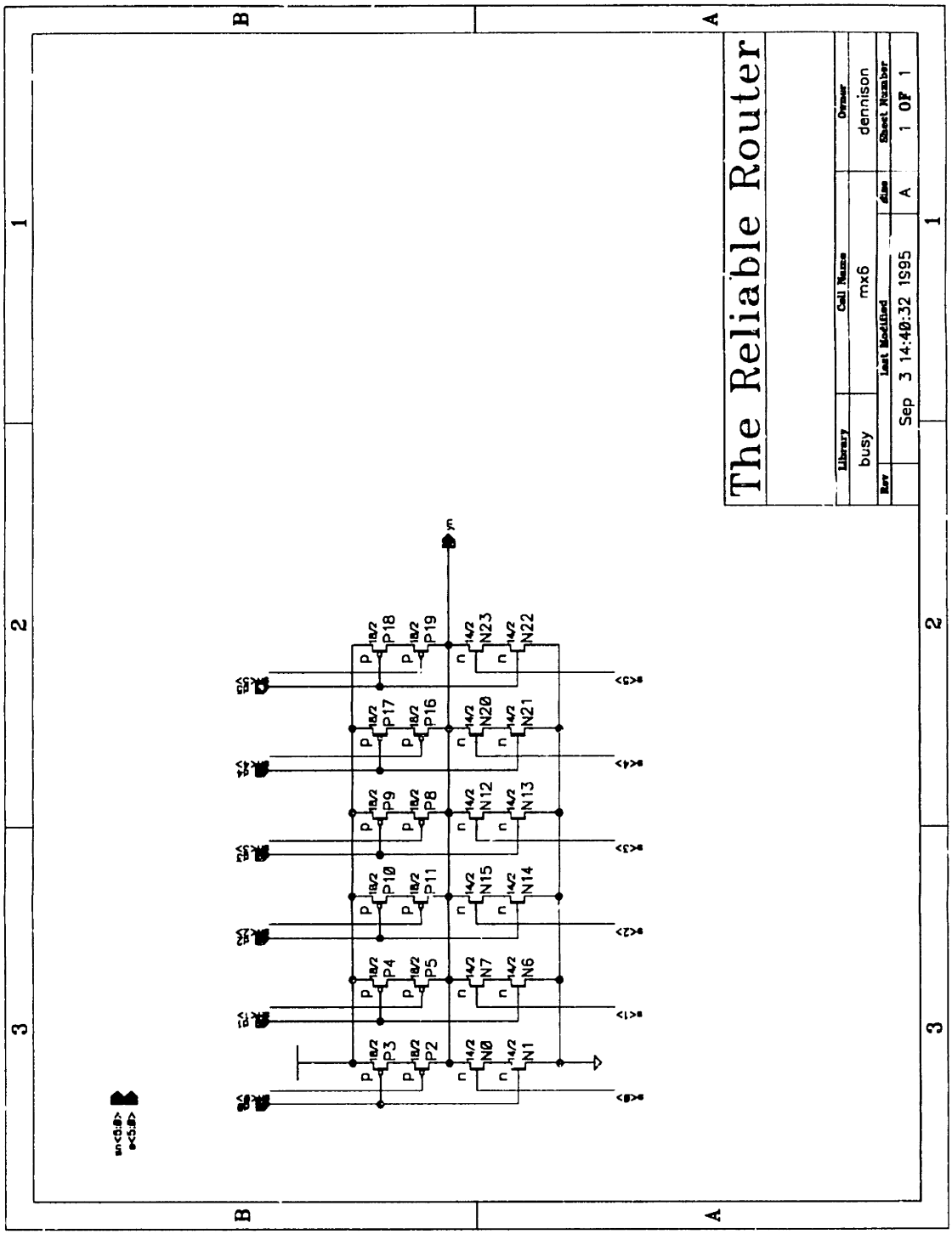


Figure A-33: Library busy, cell mx6

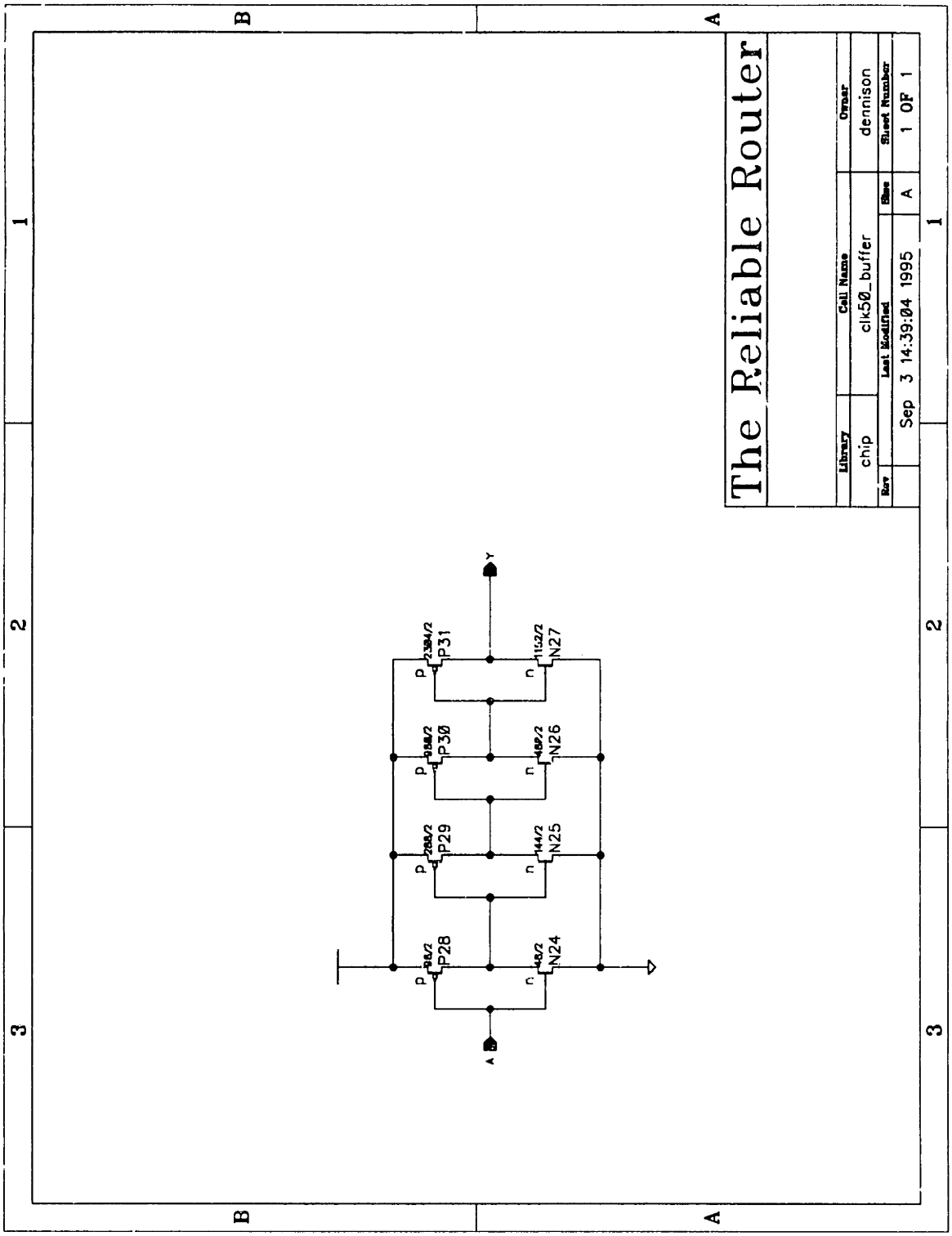


Figure A-34: Library chip, cell clk50_buffer

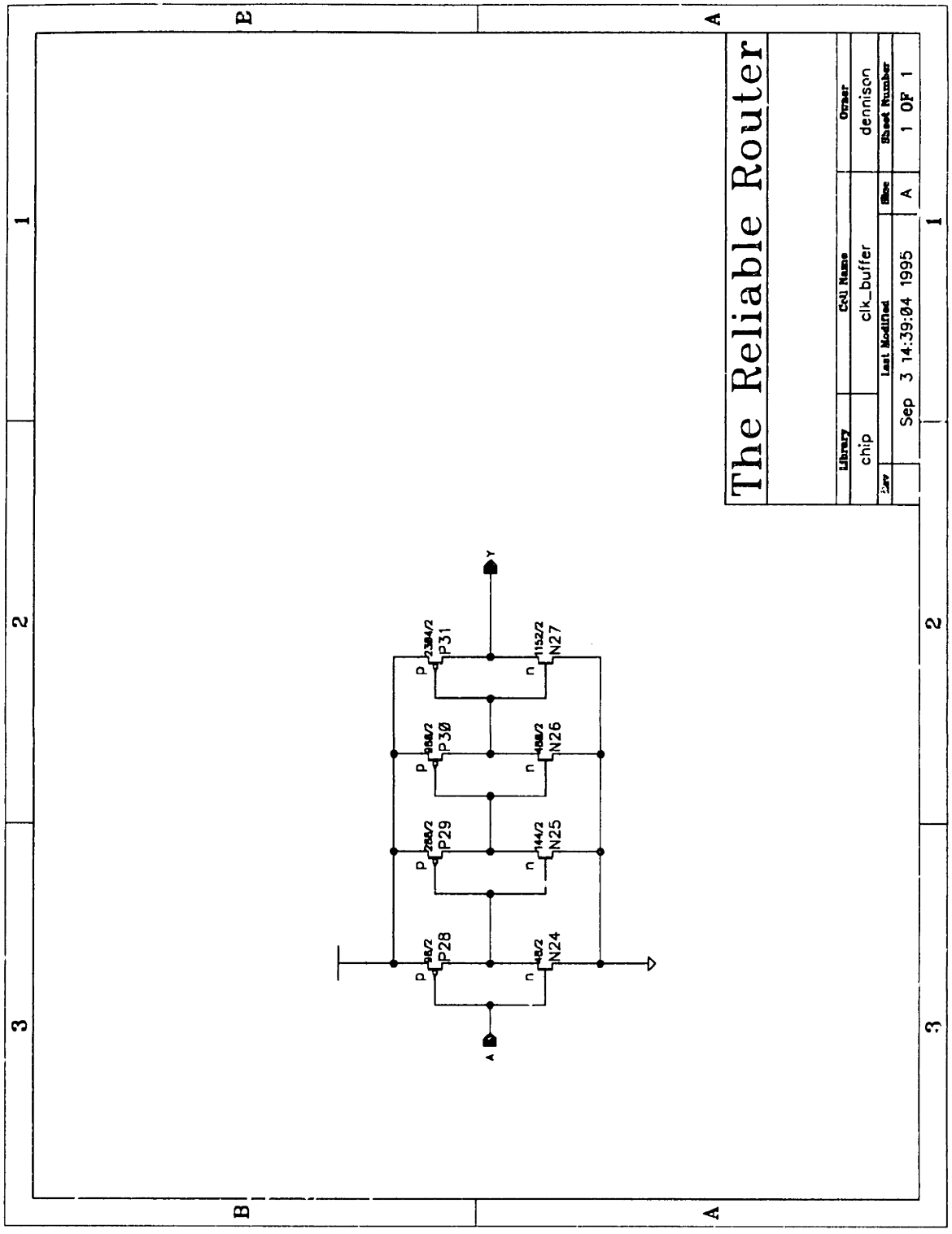


Figure A-35: Library chip, cell clk_buffer

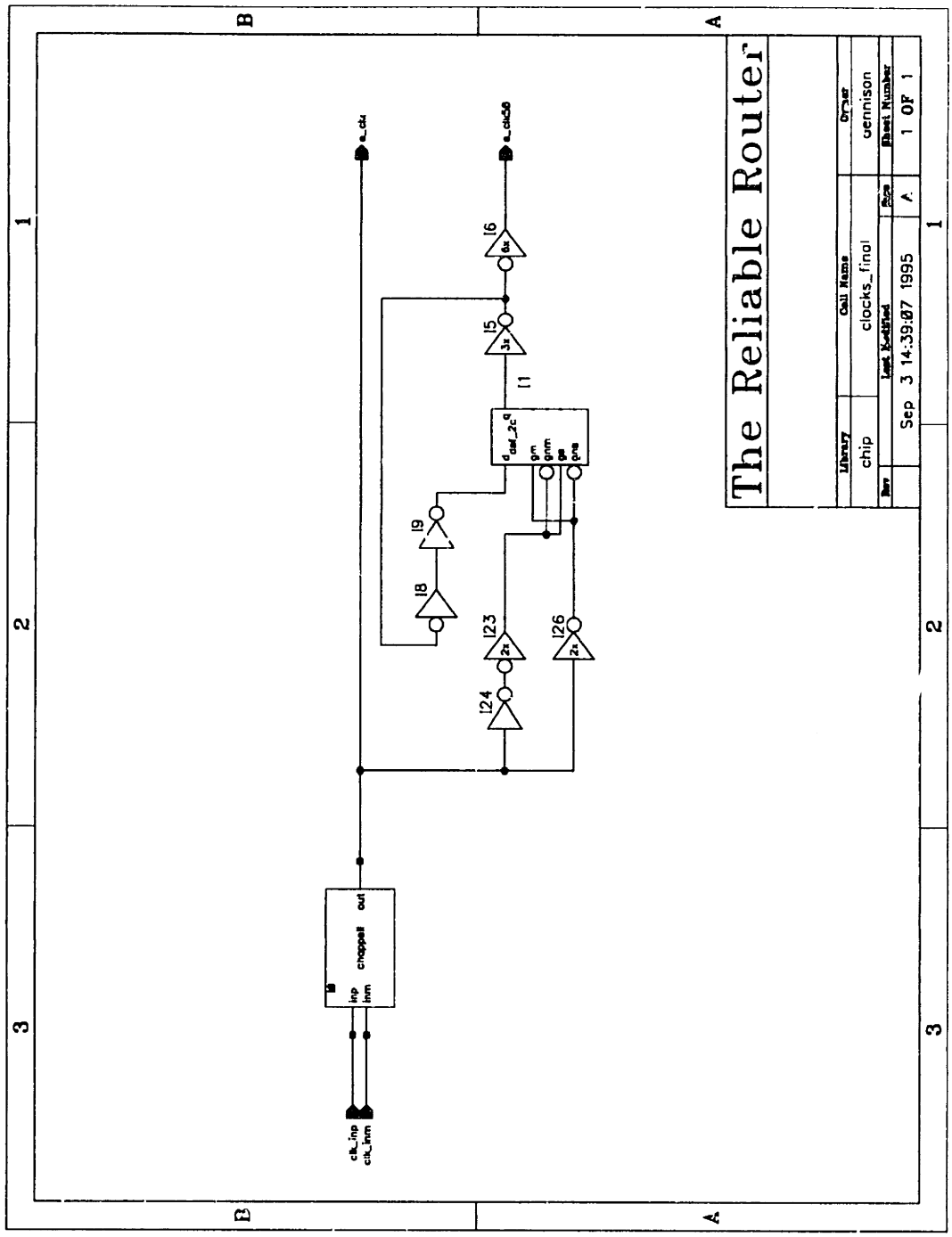


Figure A-36: Library chip, cell clocks_final

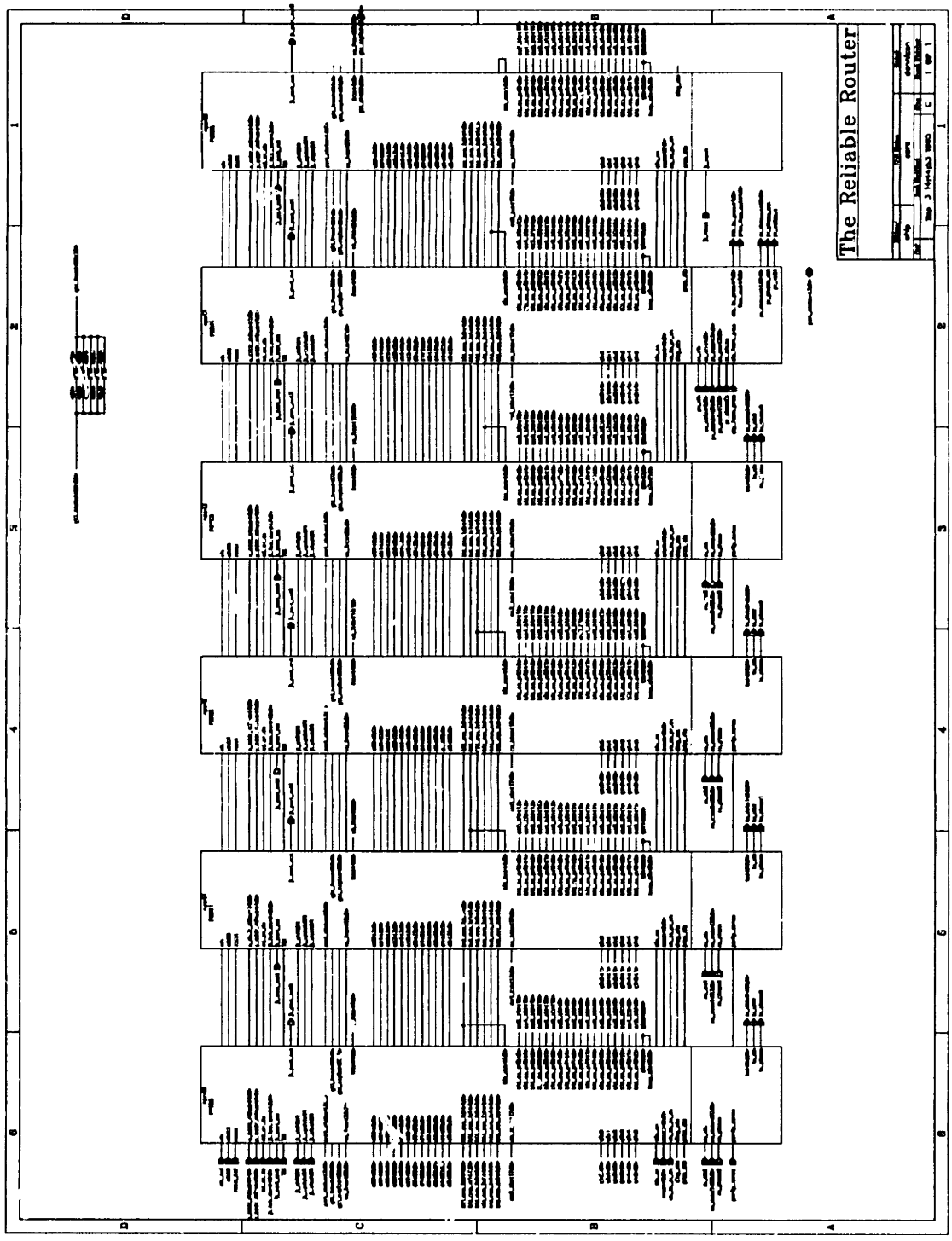


Figure A-37: Library chip, cell core

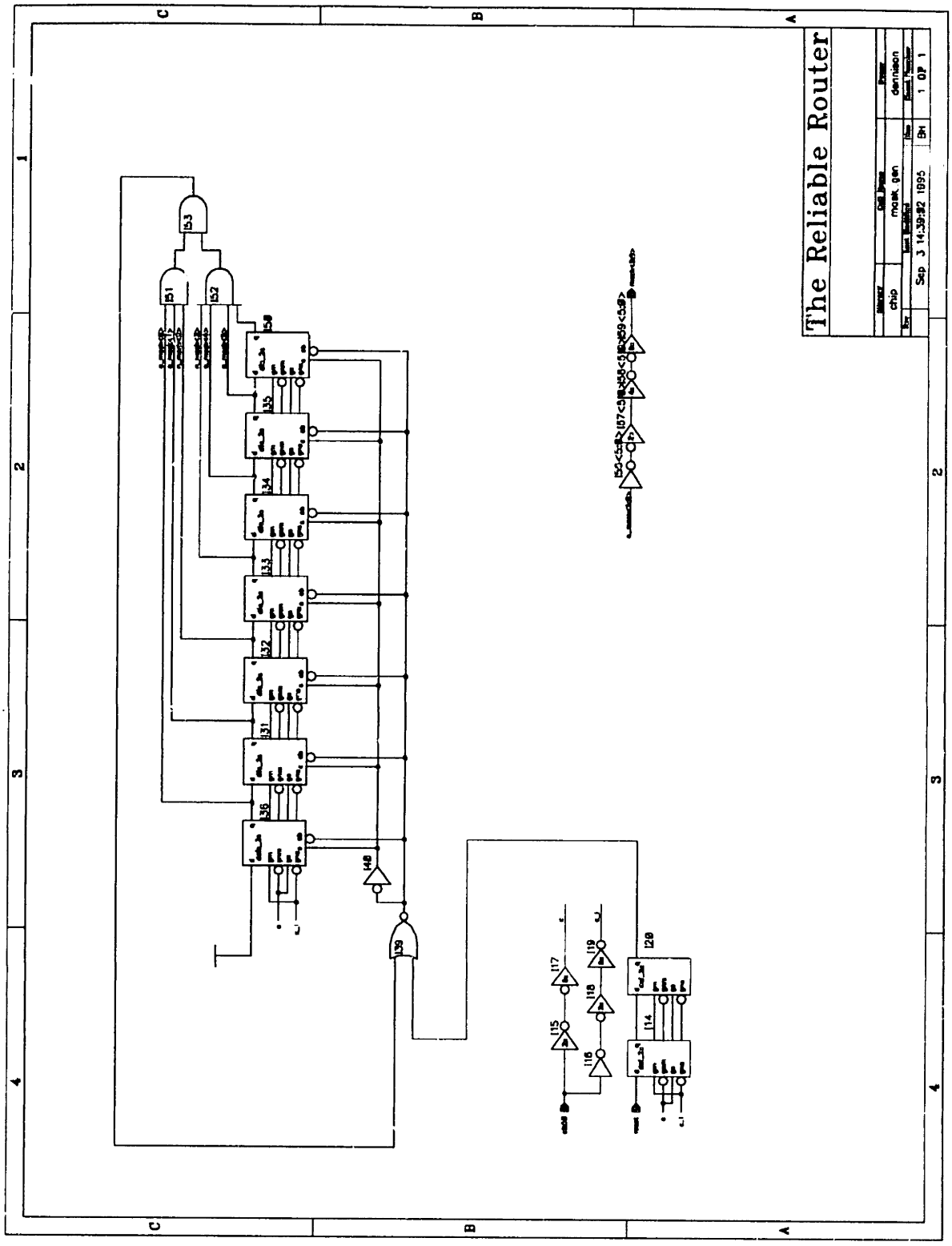


Figure A-38: Library chip, cell mask_gen

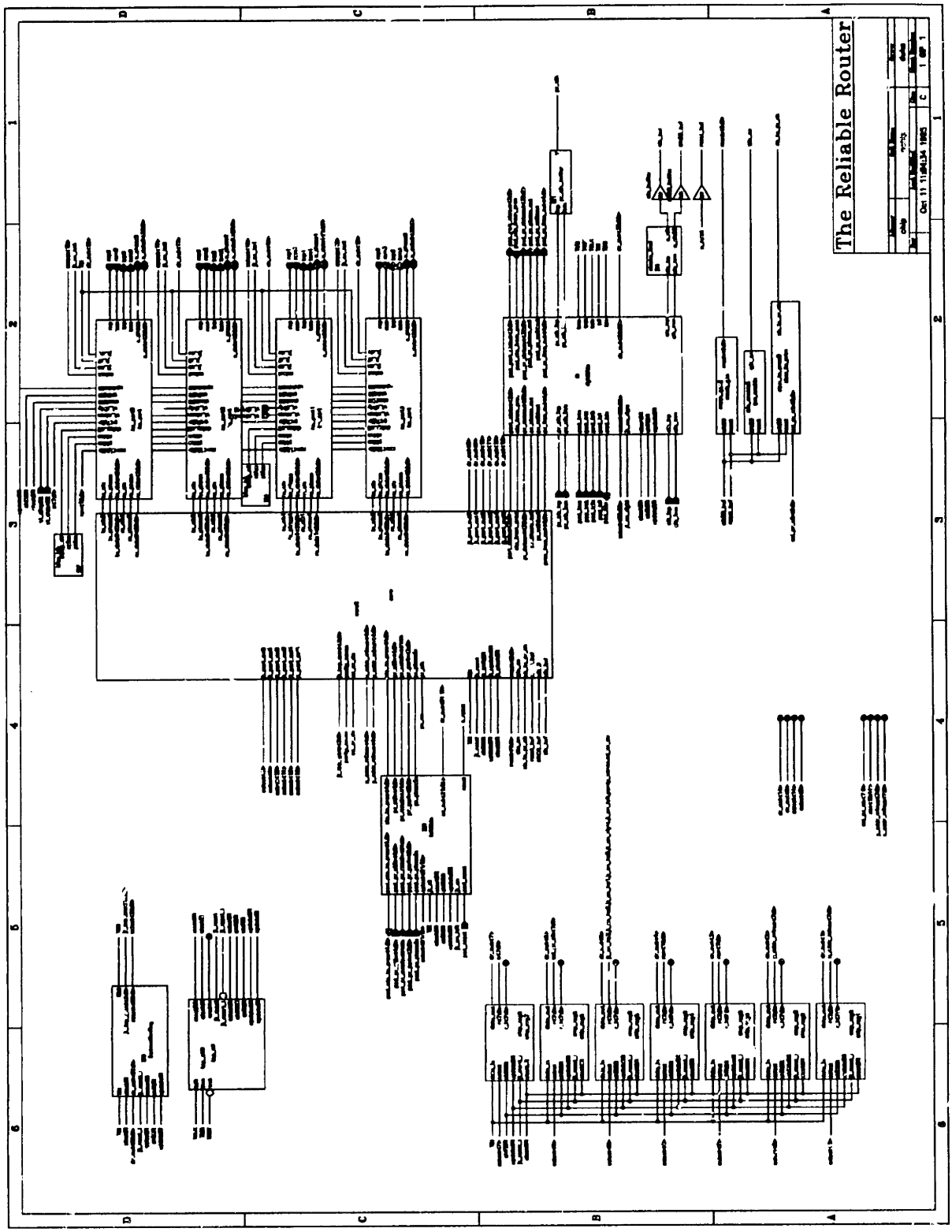


Figure A-39: Library chip, cell nchip

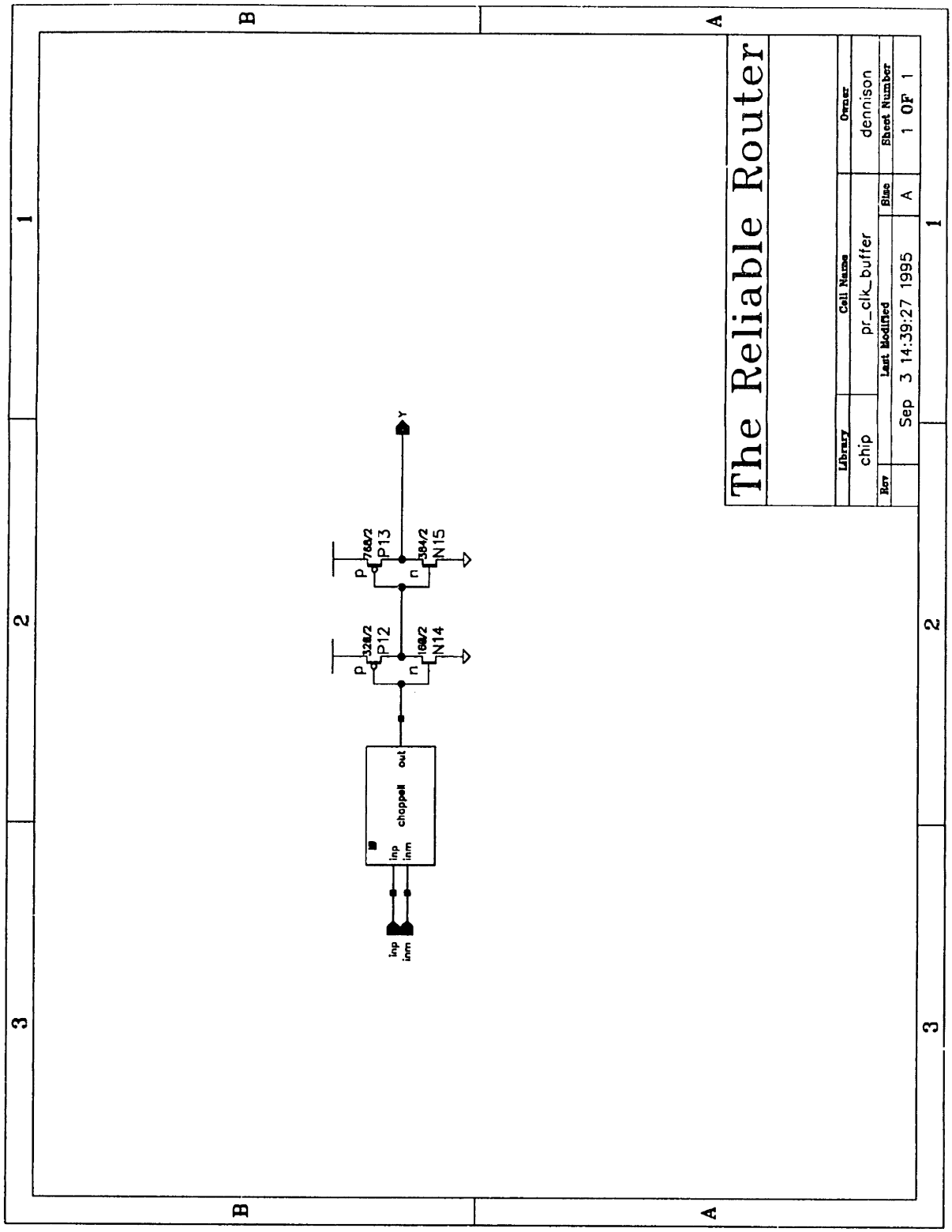


Figure A-40: Library chip, cell pr_clk_buffer

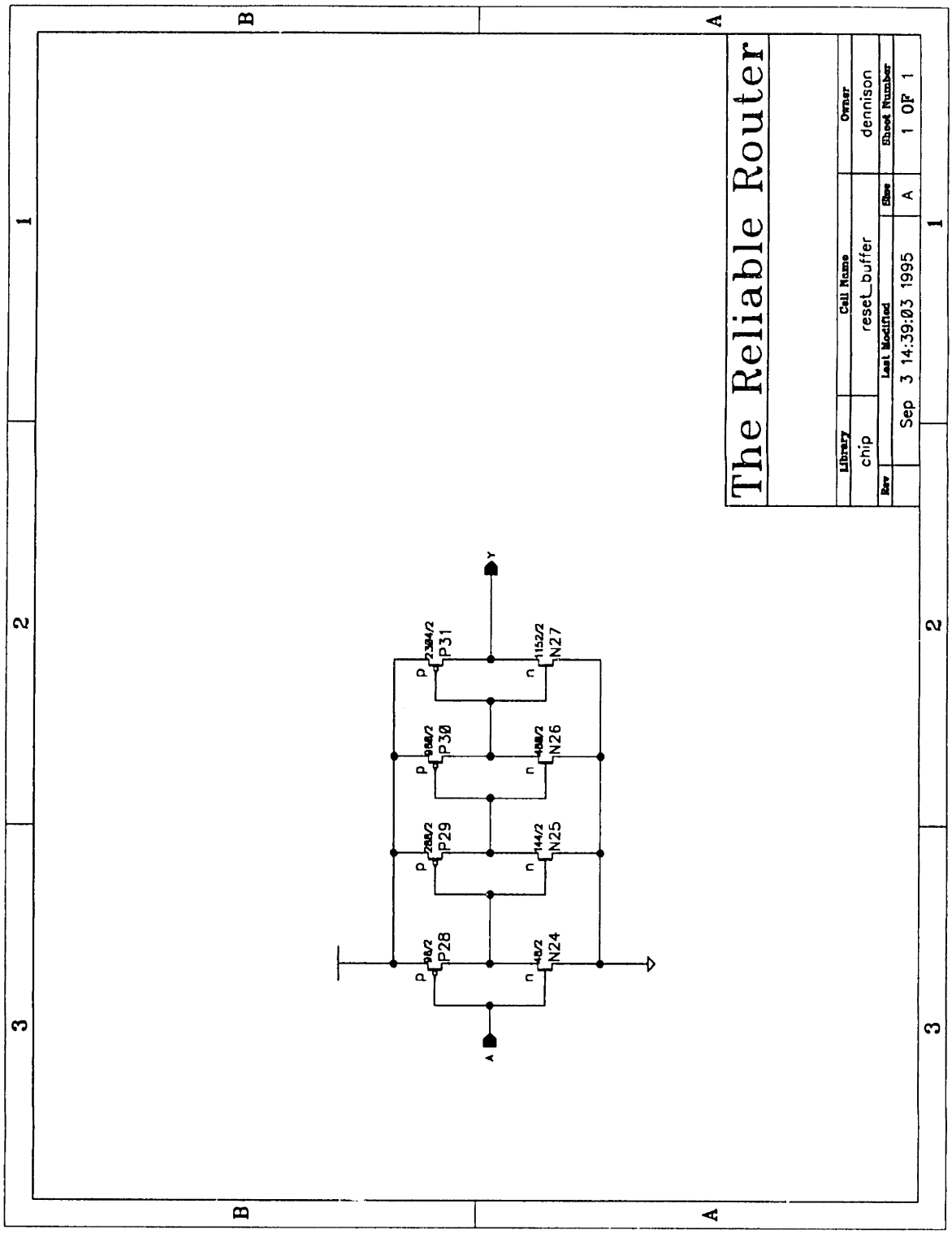


Figure A-41: Library chip, cell reset_buffer

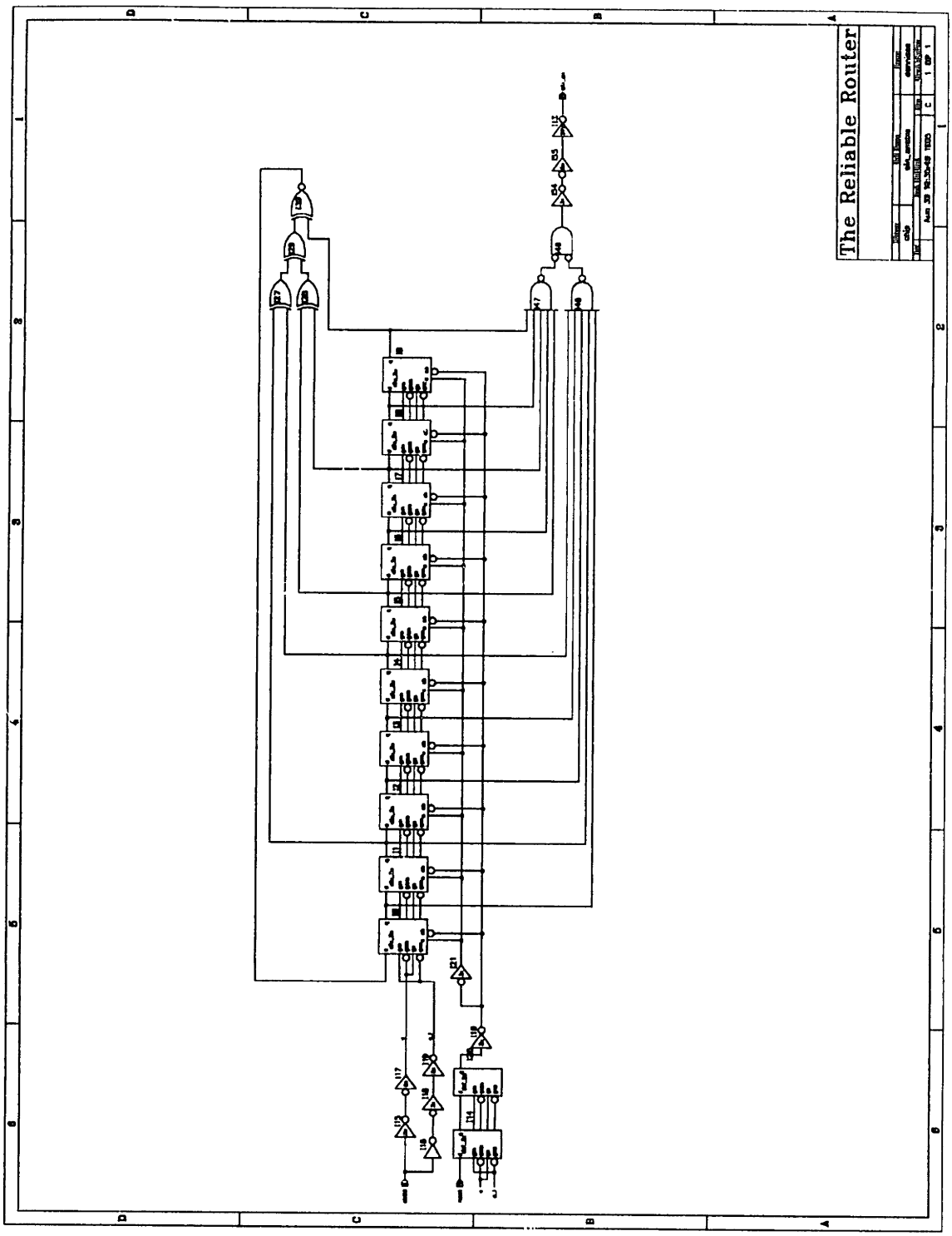


Figure A-42: Library chip, cell win_enable

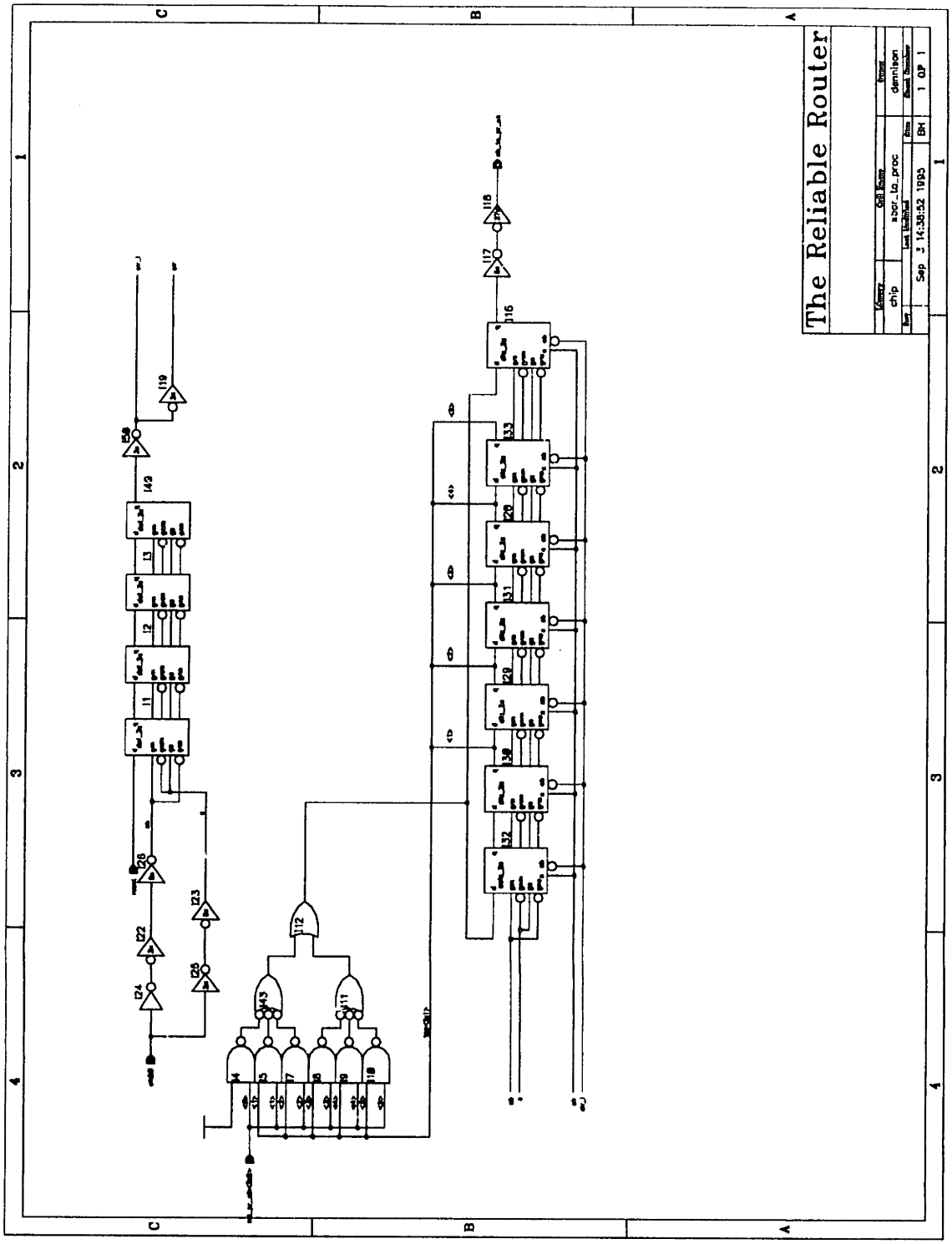


Figure A-43: Library chip, cell xbar_to_proc

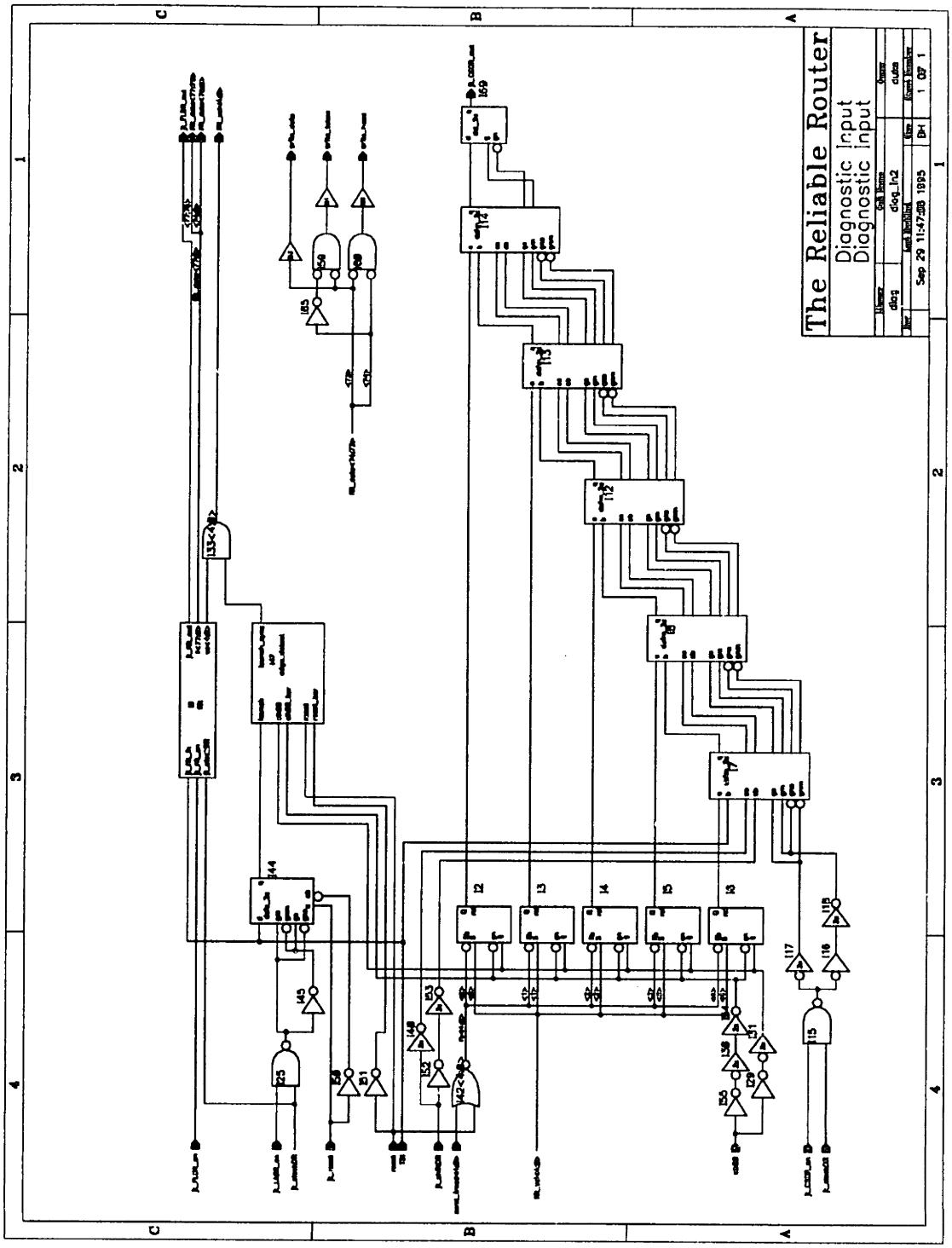


Figure A-44: Library diag, cell diag_in2

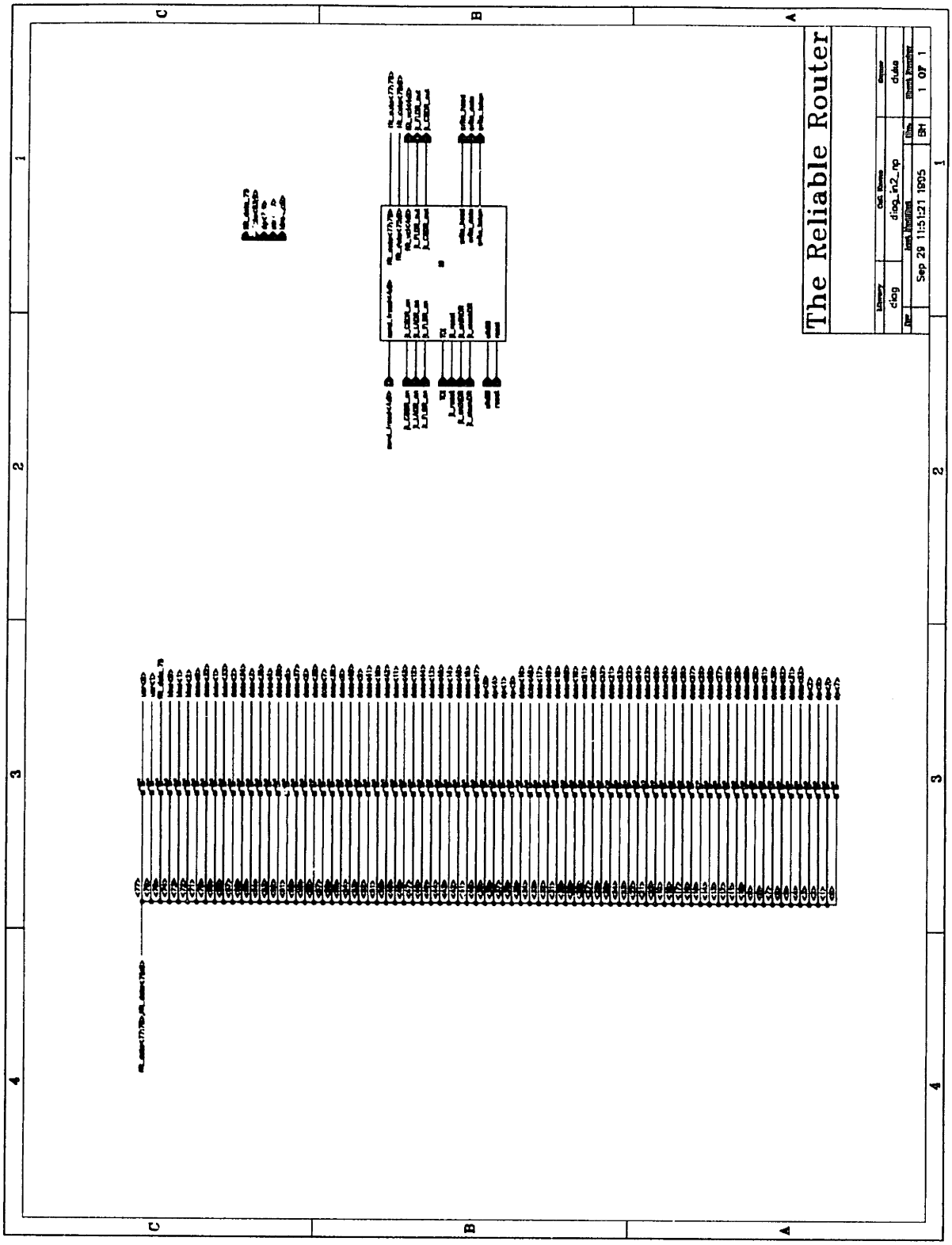


Figure A-45: Library diag, cell diag_in2_np

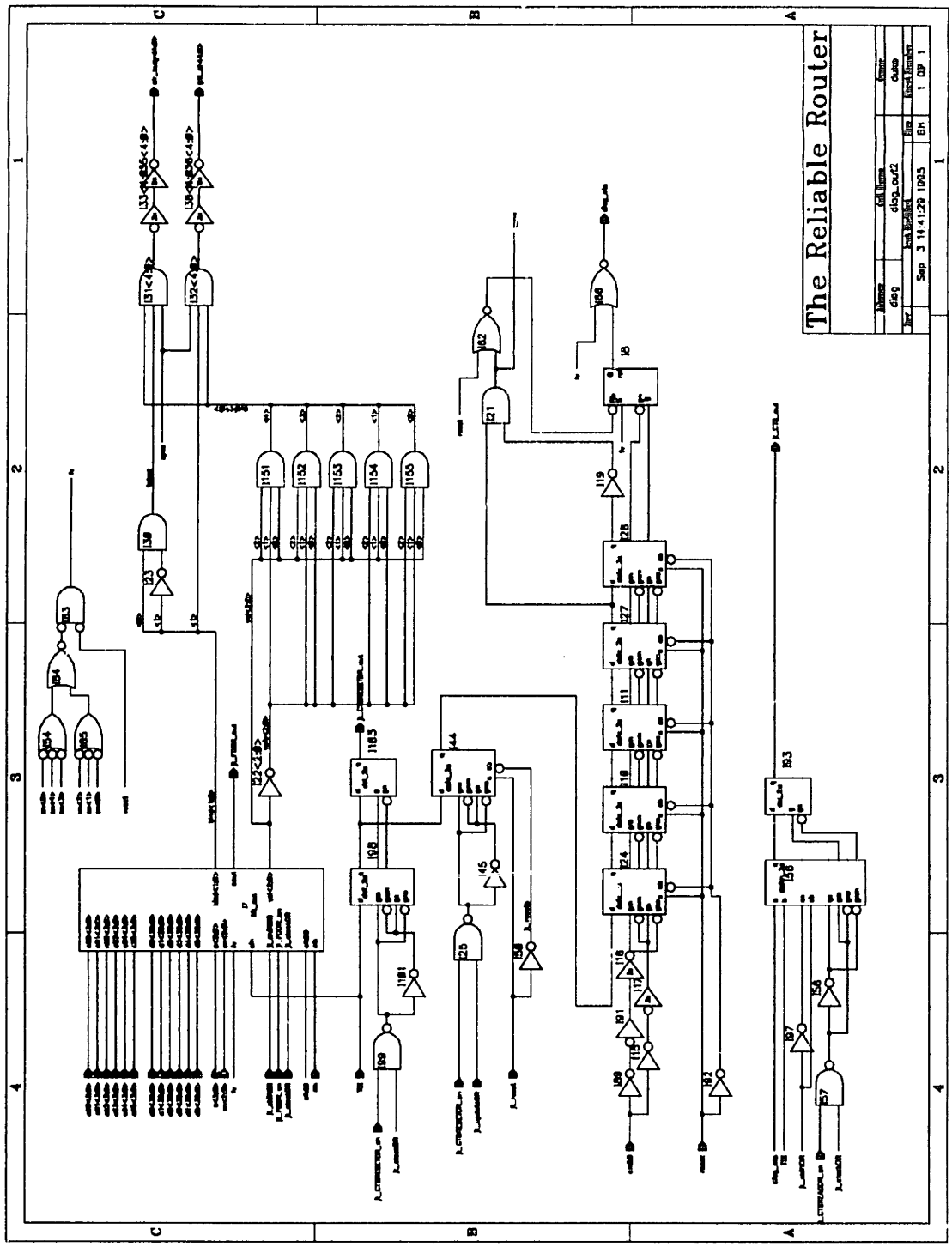


Figure A-46: Library diag, cell diag_out2

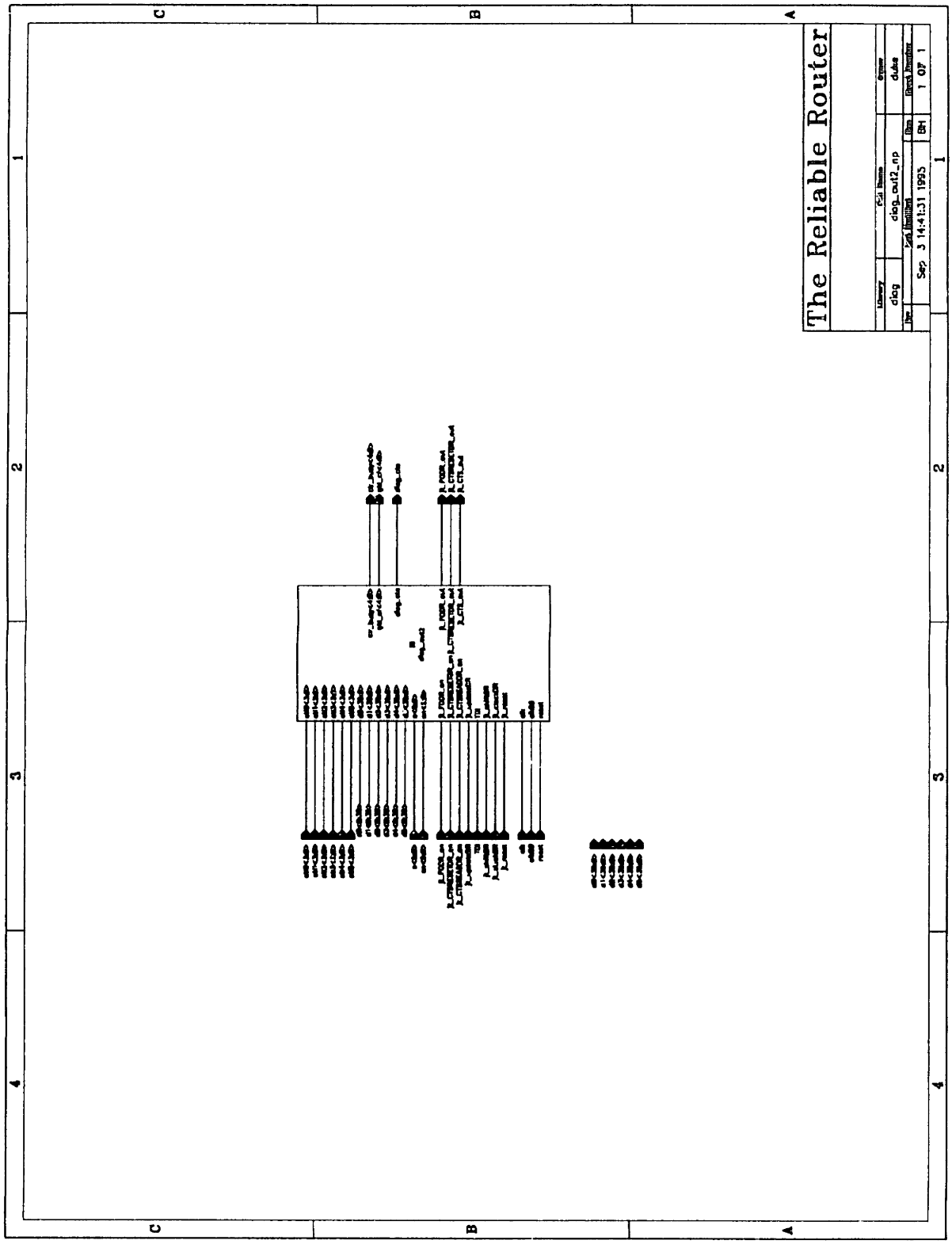


Figure A-47: Library diag, cell diag_out2_np

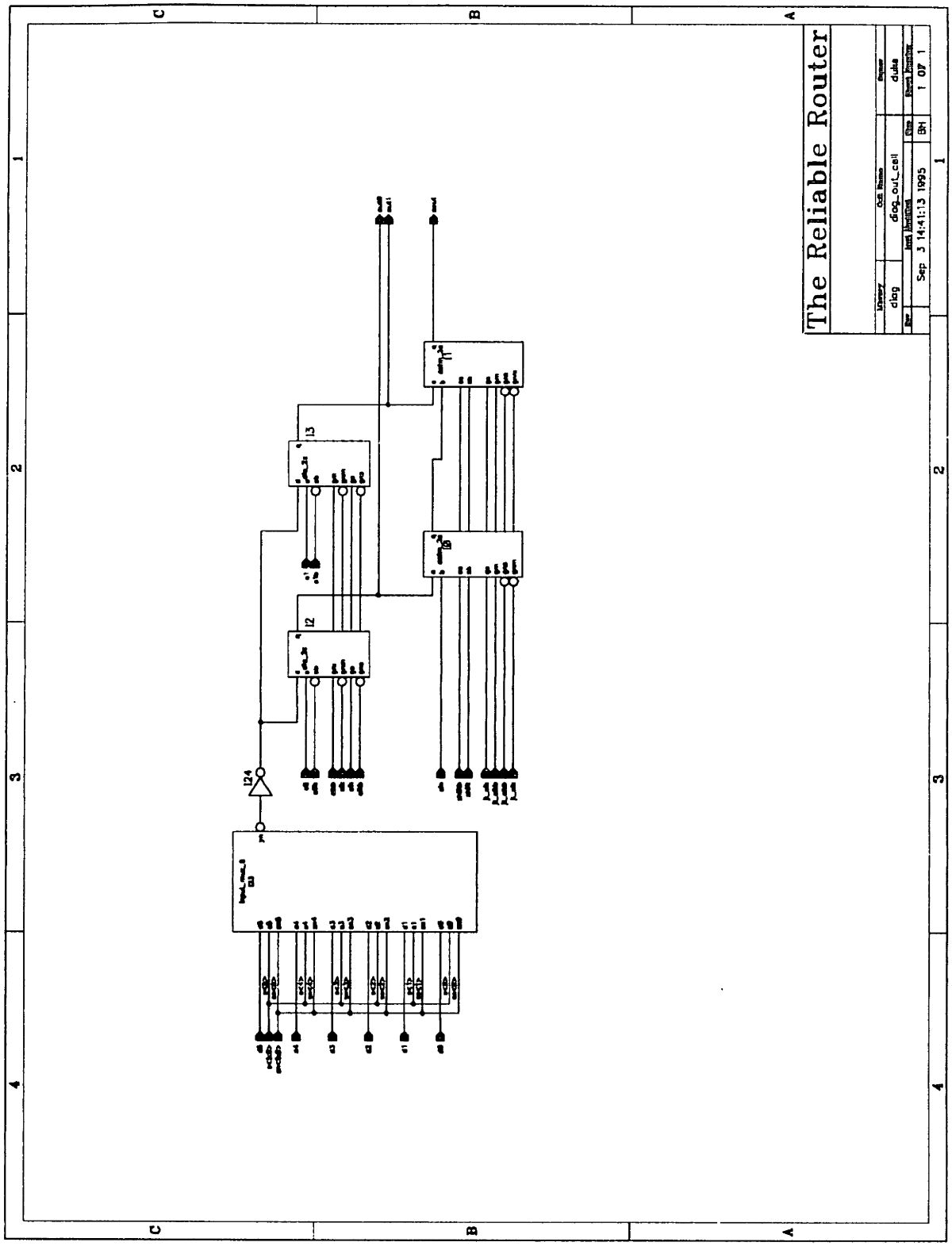


Figure A-48: Library diag, cell diag_out_cell

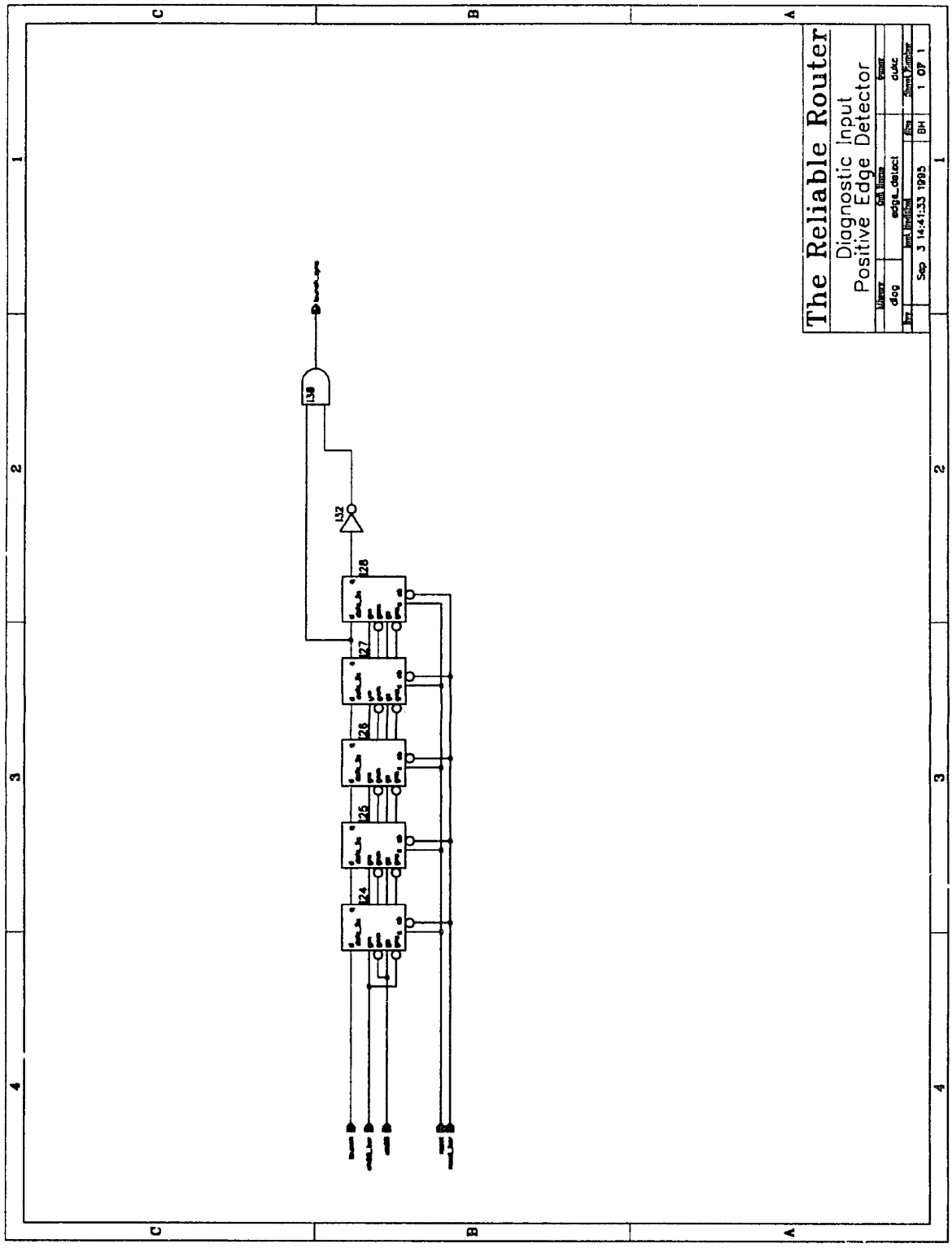


Figure A-49: Library diag, cell edge_detect

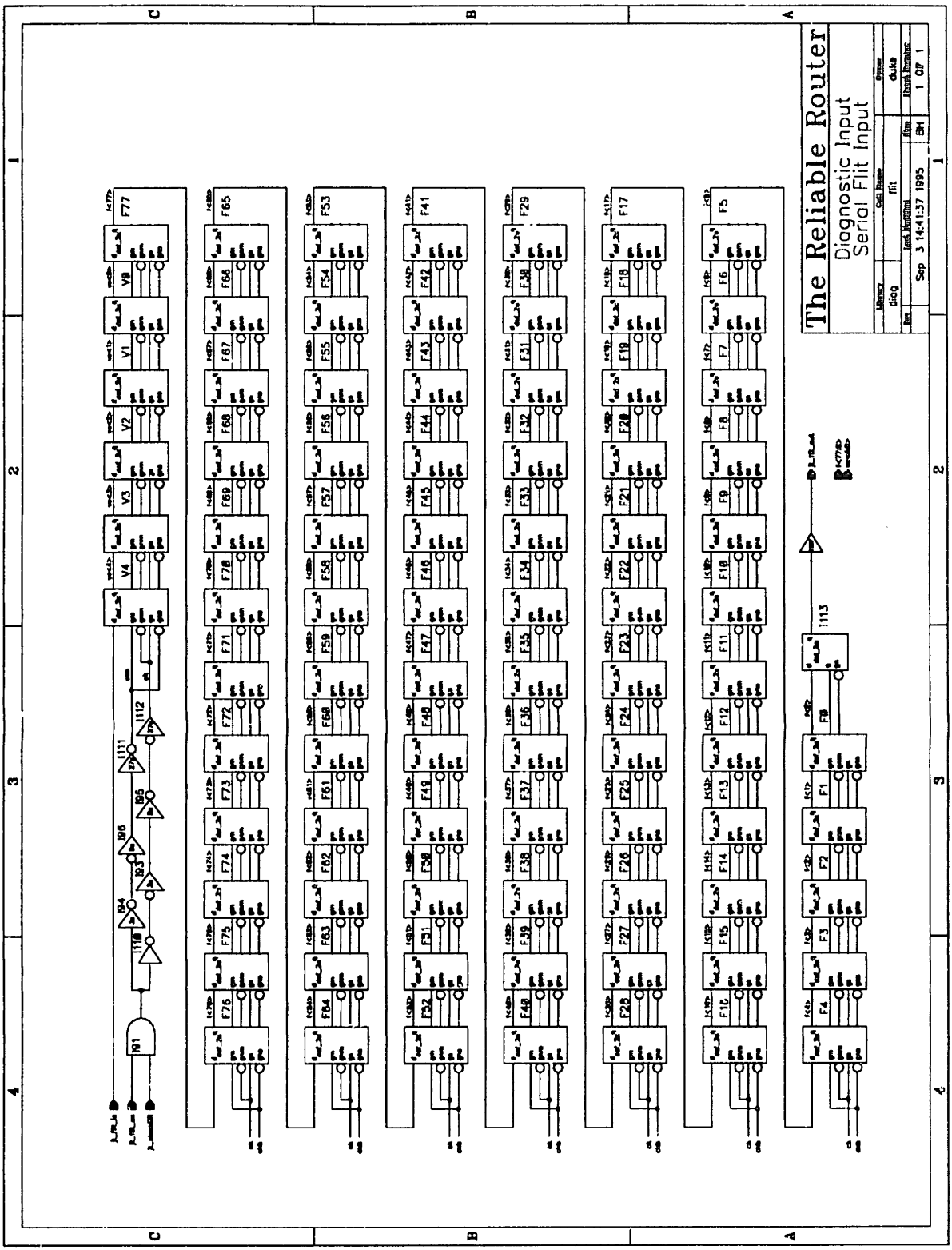


Figure A-50: Library diag, cell flit

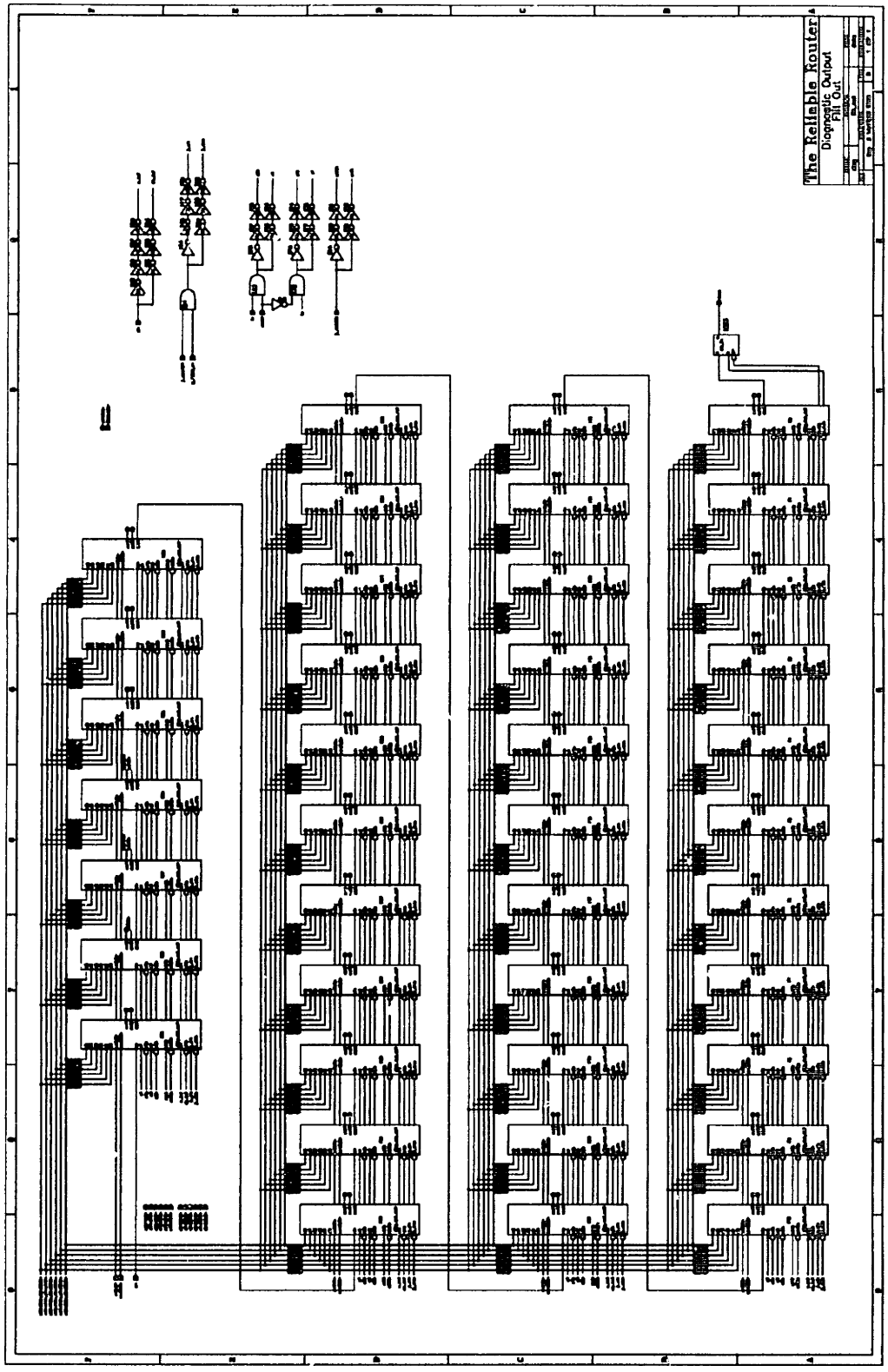


Figure A-51: Library diag, cell flit_out

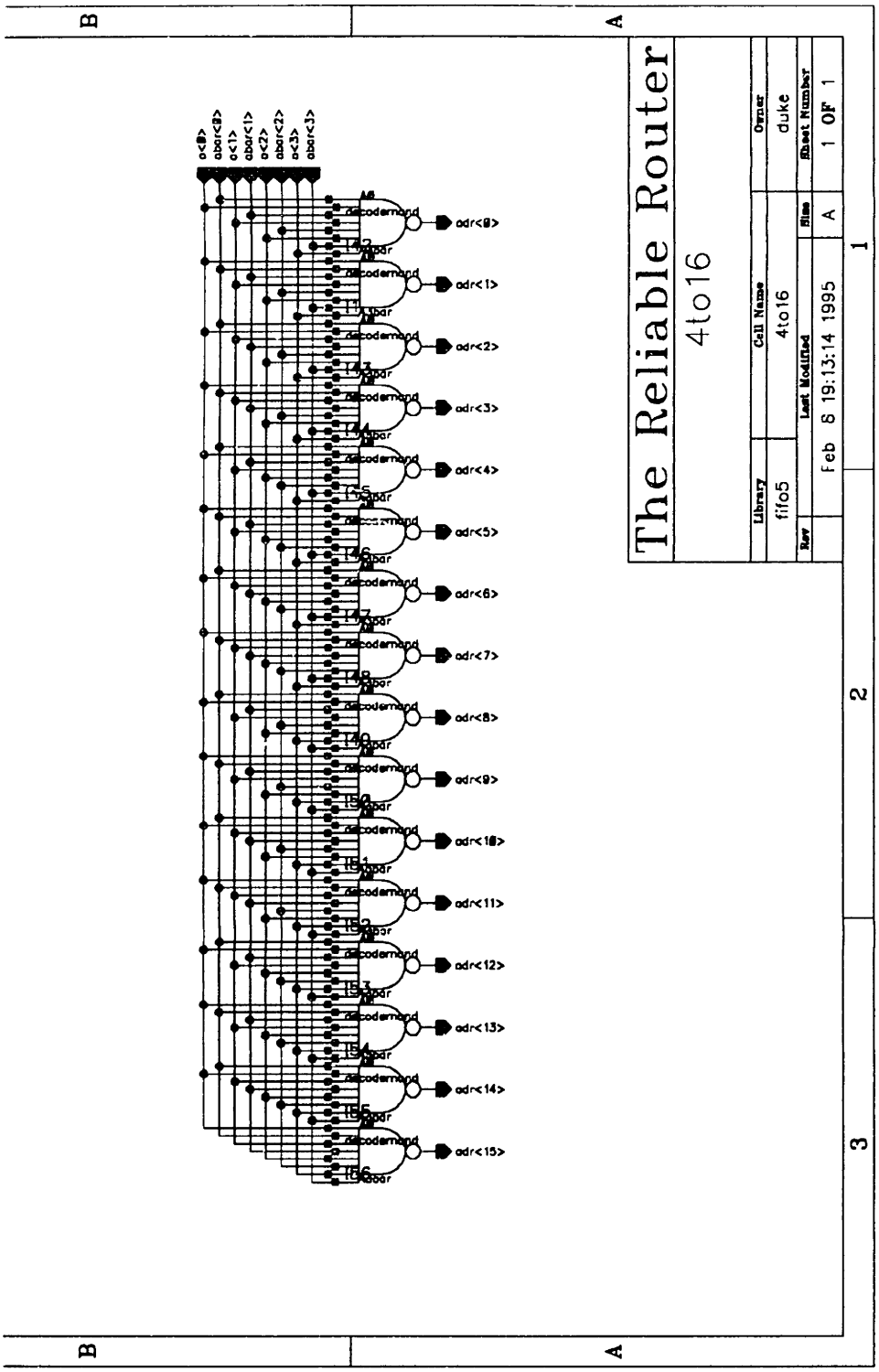


Figure A-52: Library fifo5, cell 4to16

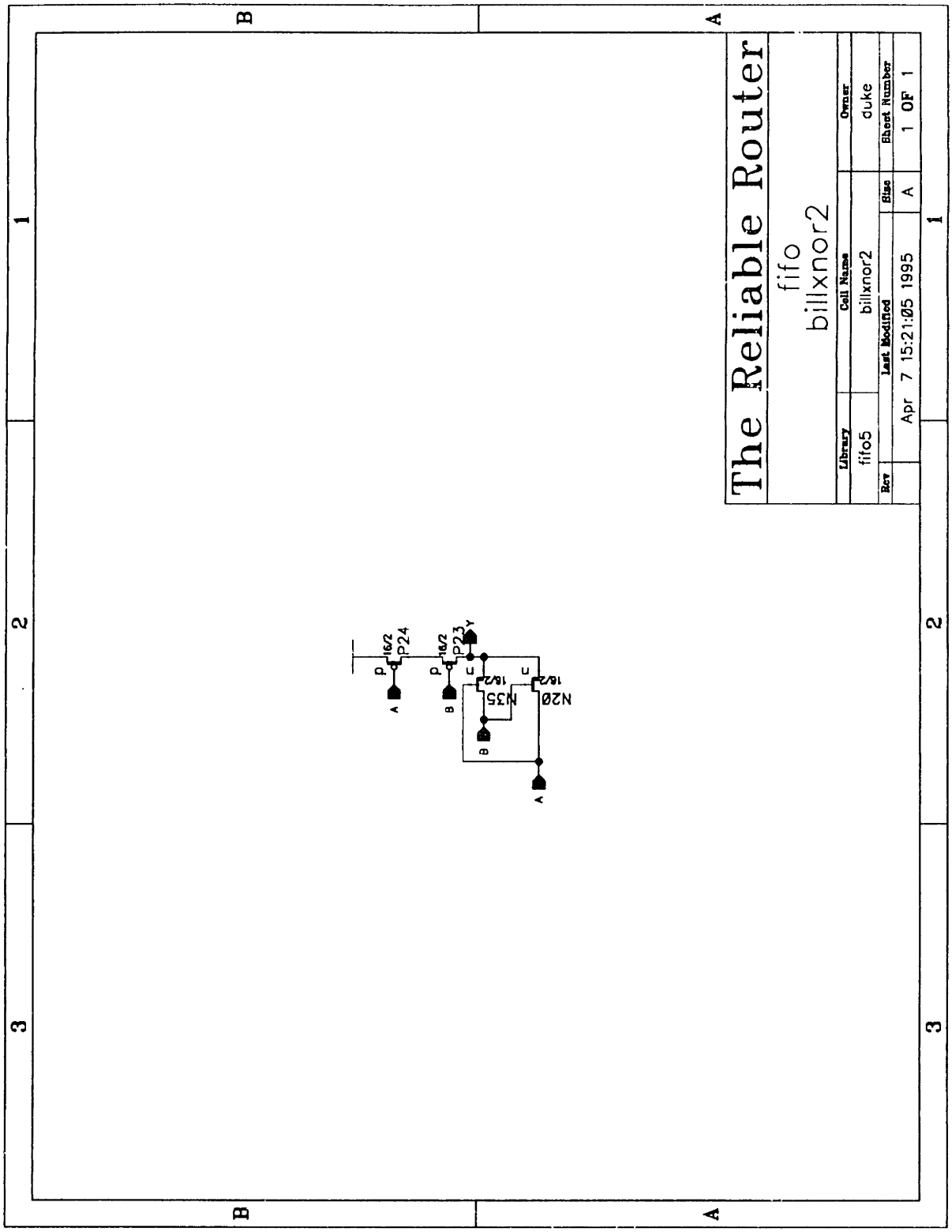


Figure A-53: Library fifo5, cell billxnor2

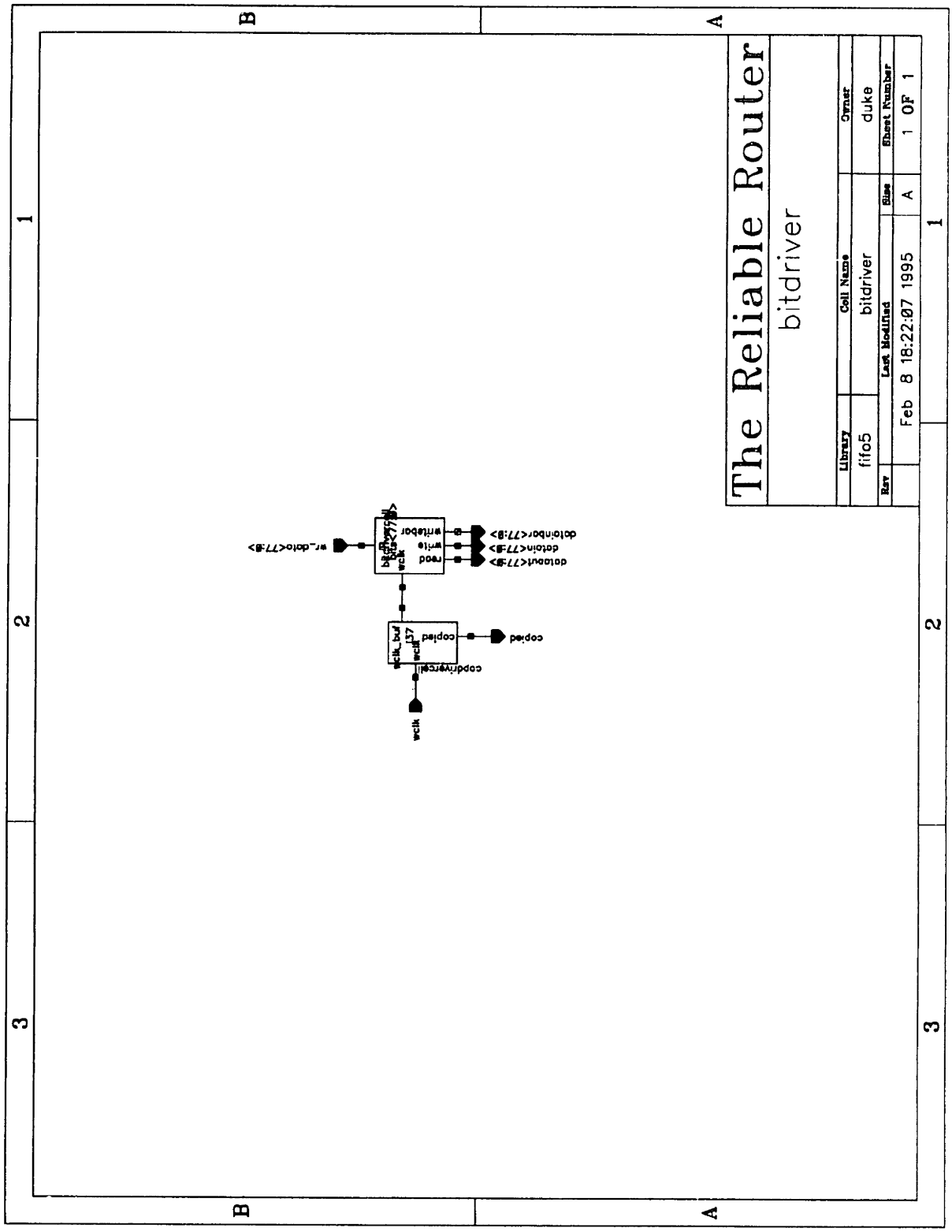


Figure A-54: Library fifo5, cell bitdriver

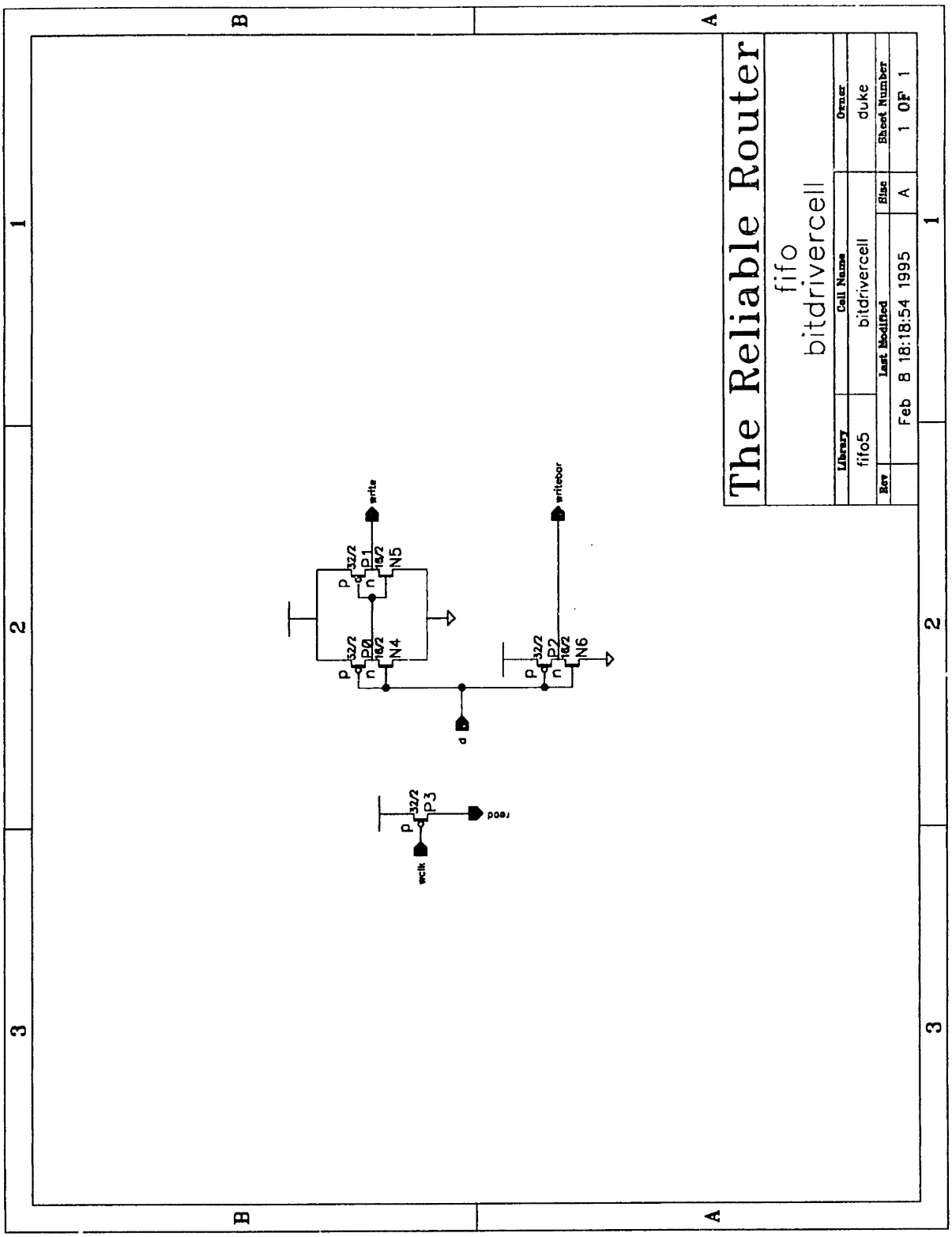


Figure A-55: Library fifo5, cell bitdrivercell

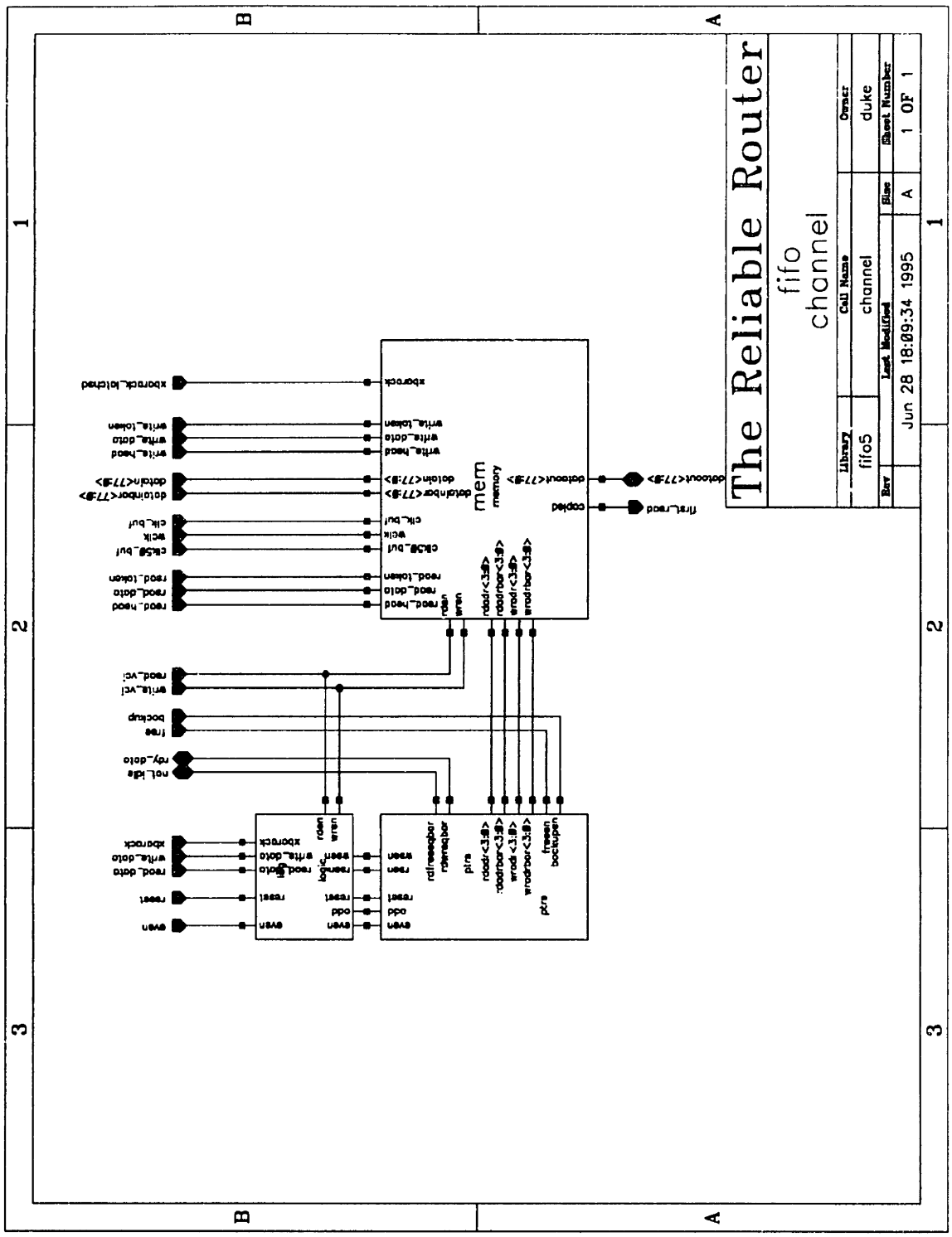


Figure A-56: Library fifo5, cell channel

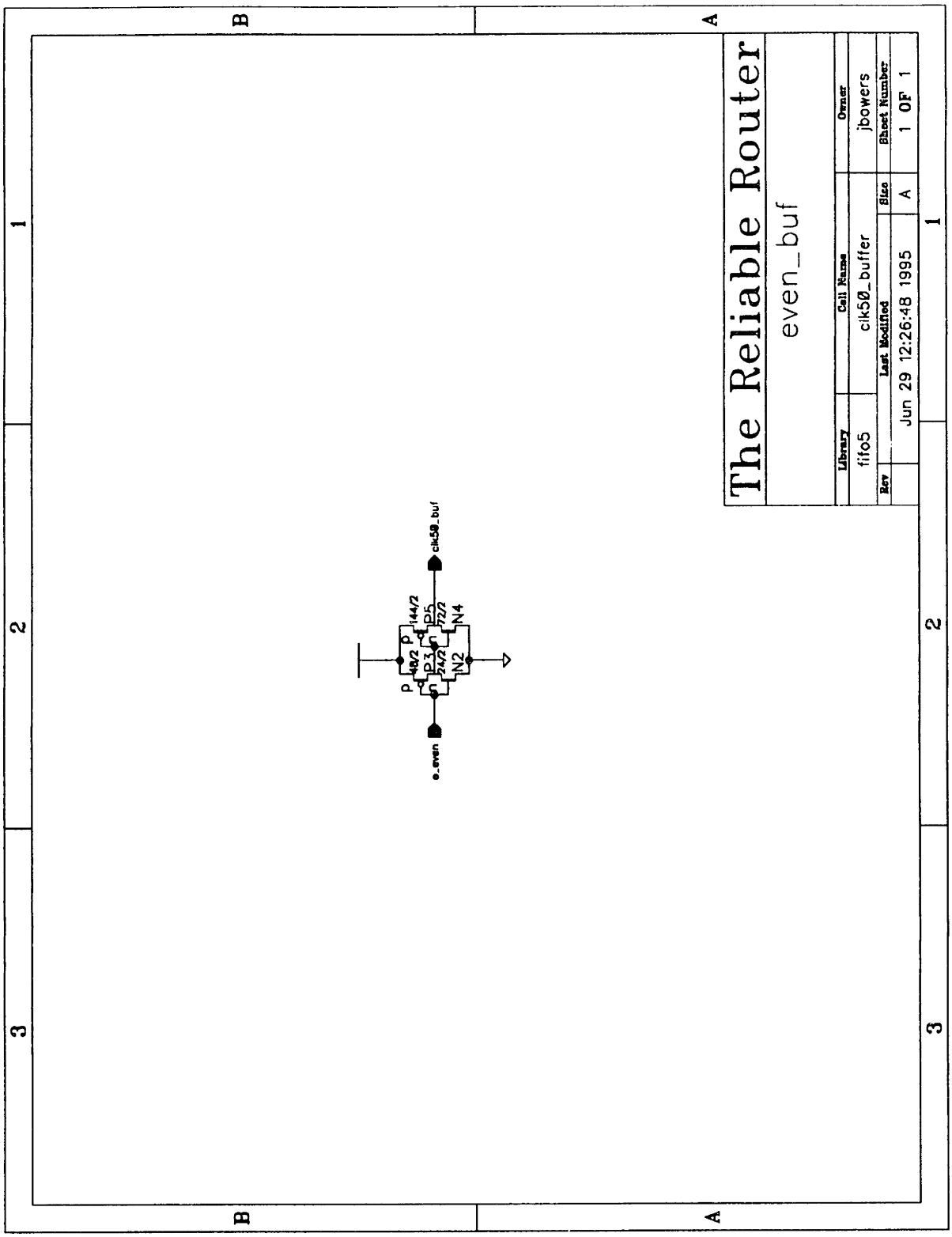


Figure A-57: Library fifo5, cell clk50_buffer

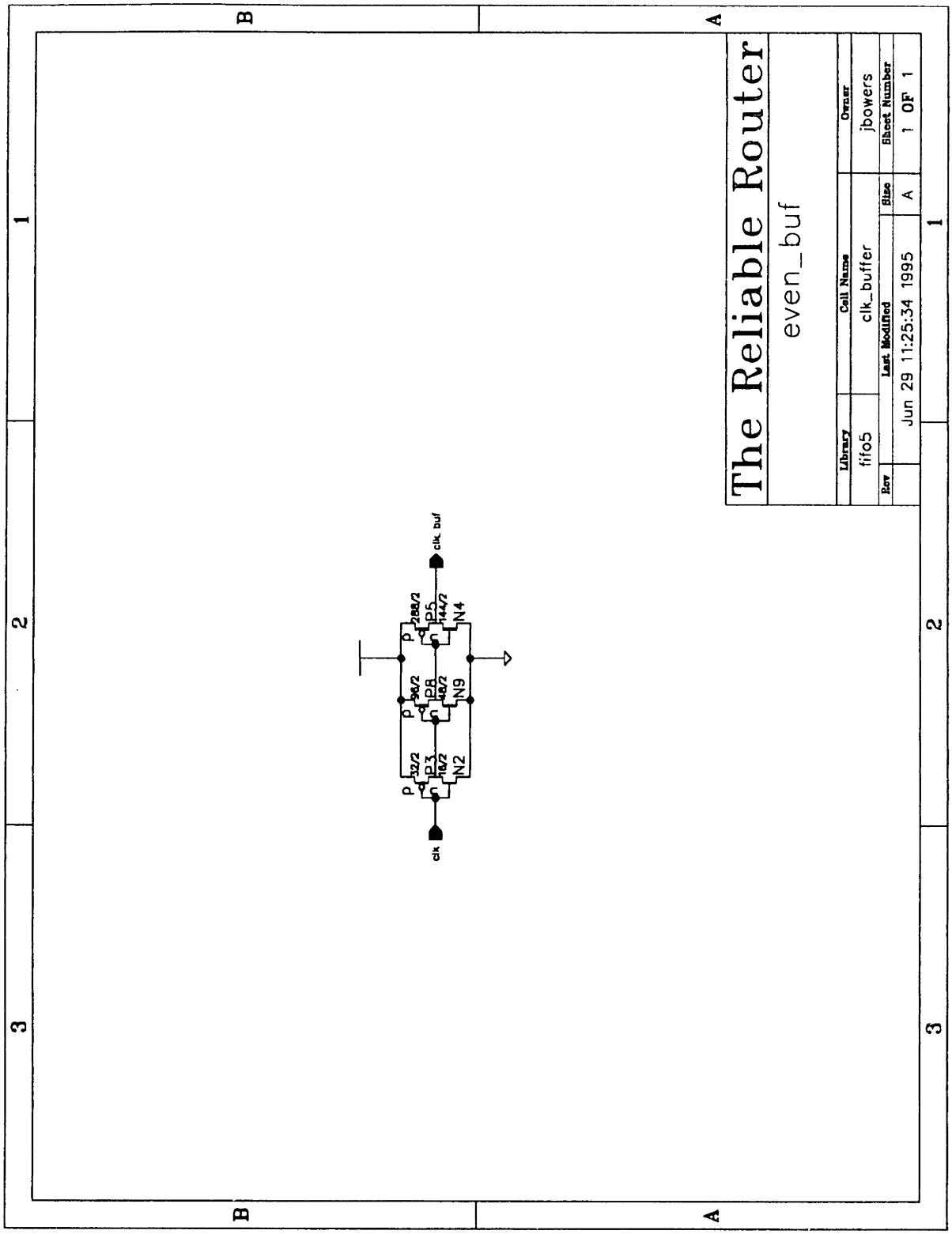


Figure A-58: Library fifo5, cell clk_buffer

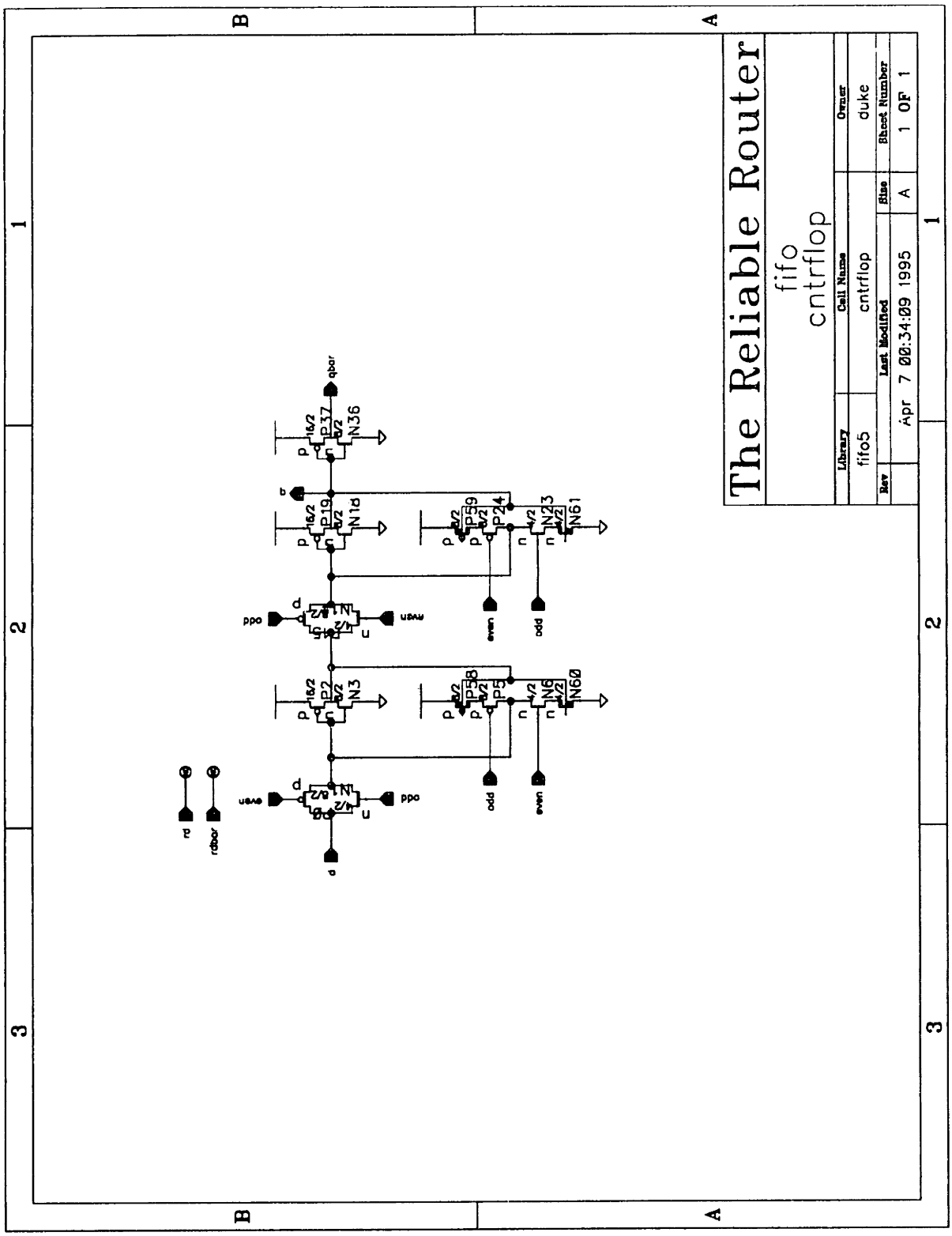


Figure A-59: Library fifo5, cell cntrflop

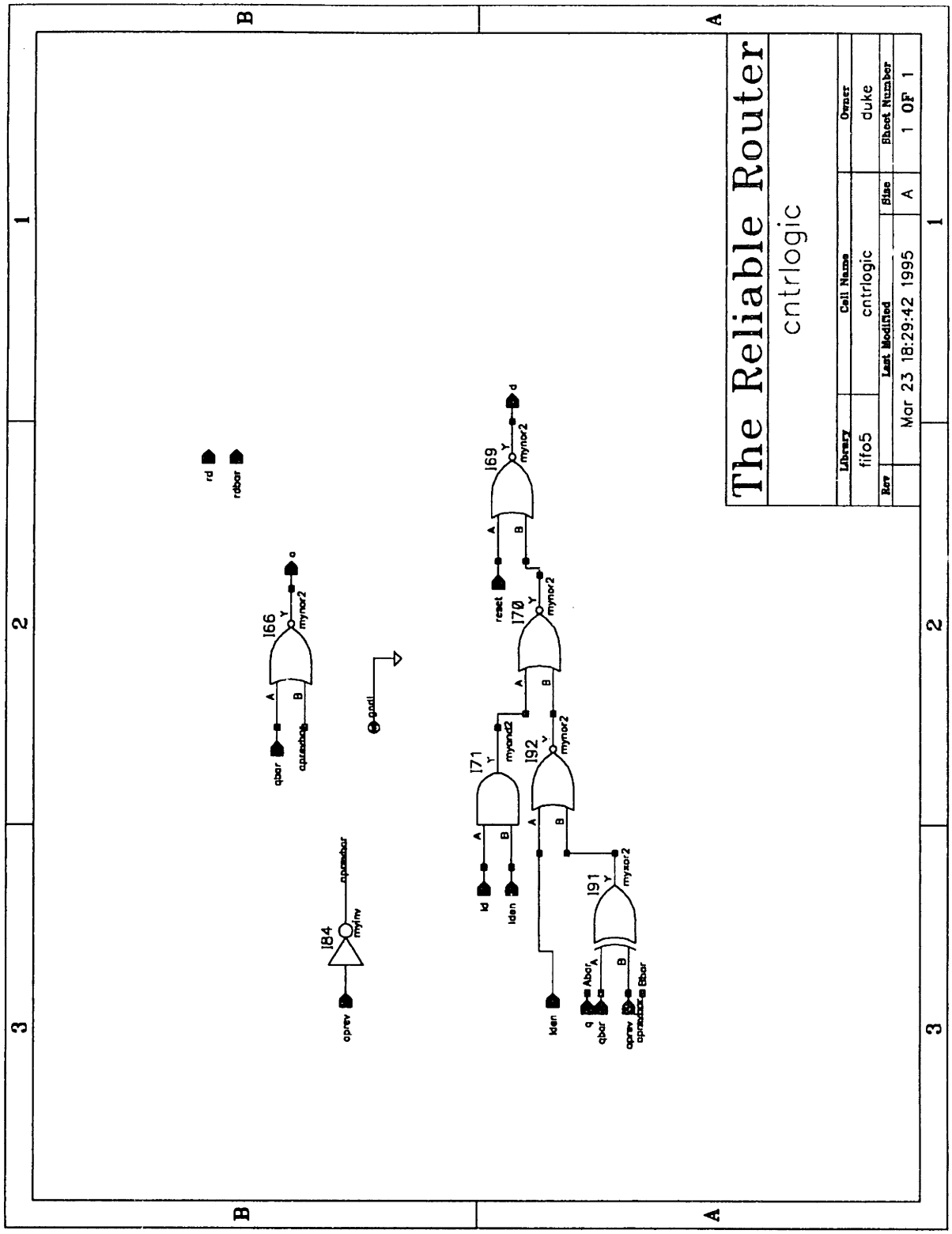


Figure A-60: Library fifo5, cell cntrlogic

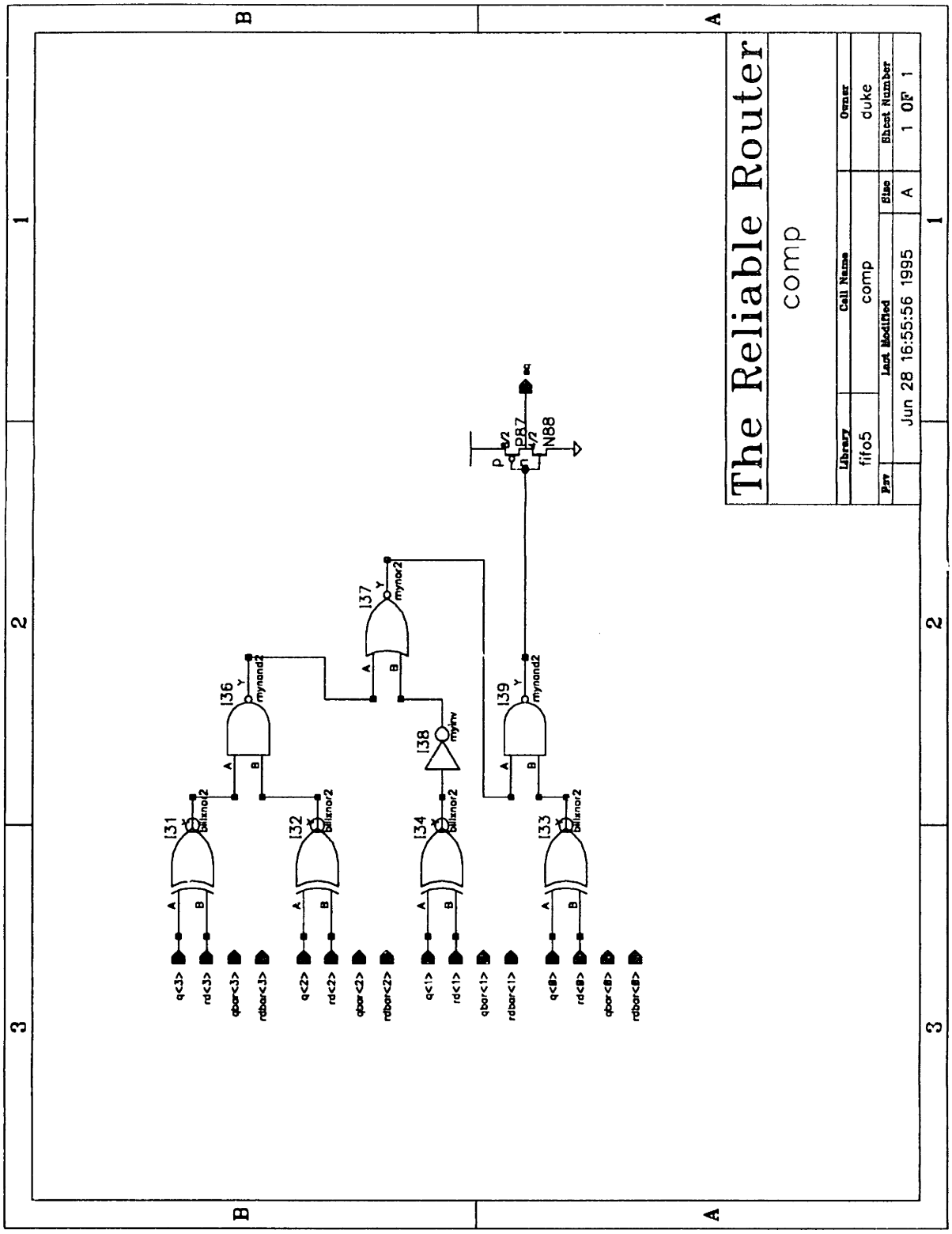


Figure A-61: Library fifo5, cell comp

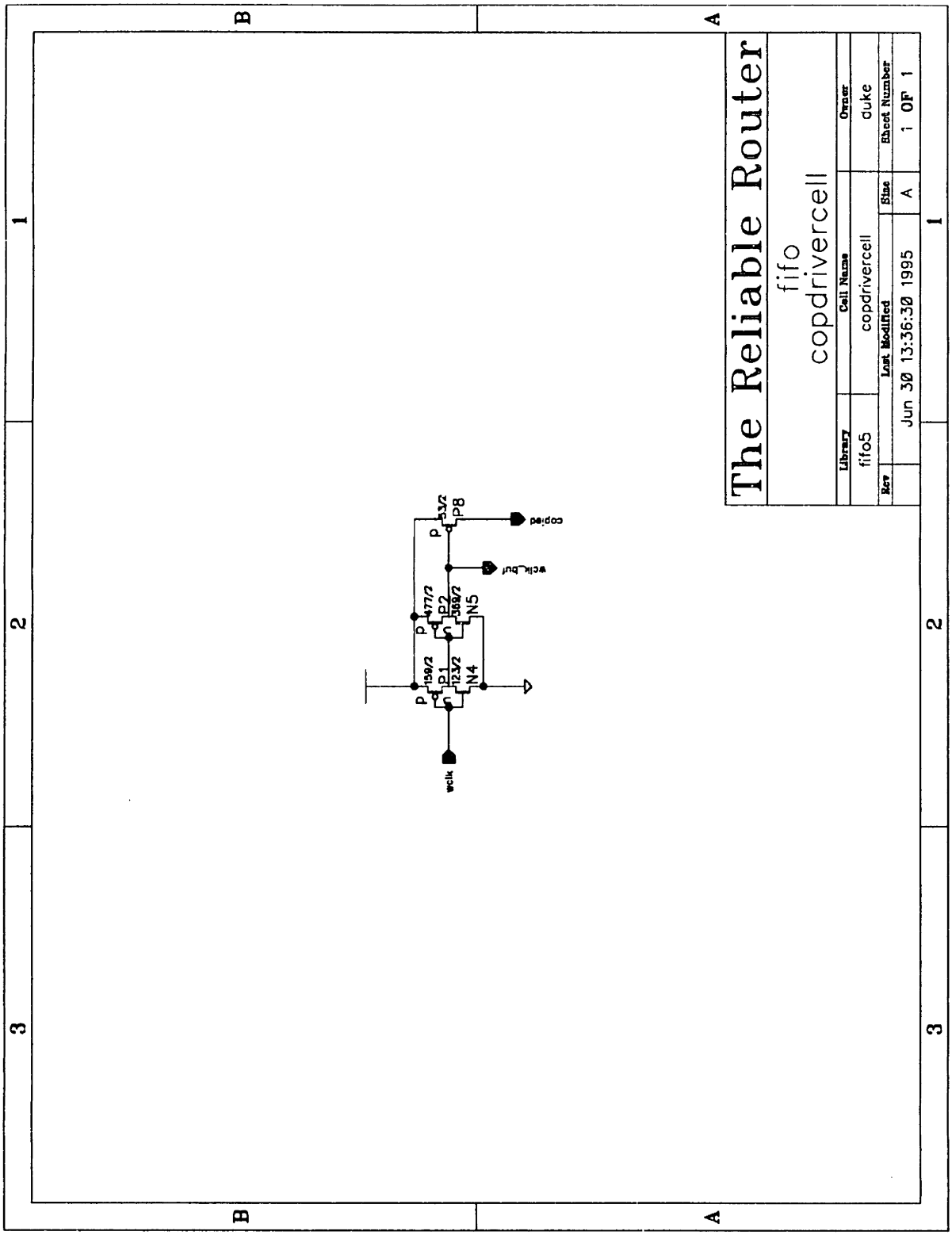


Figure A-62: Library fifo5, cell copdrivercell

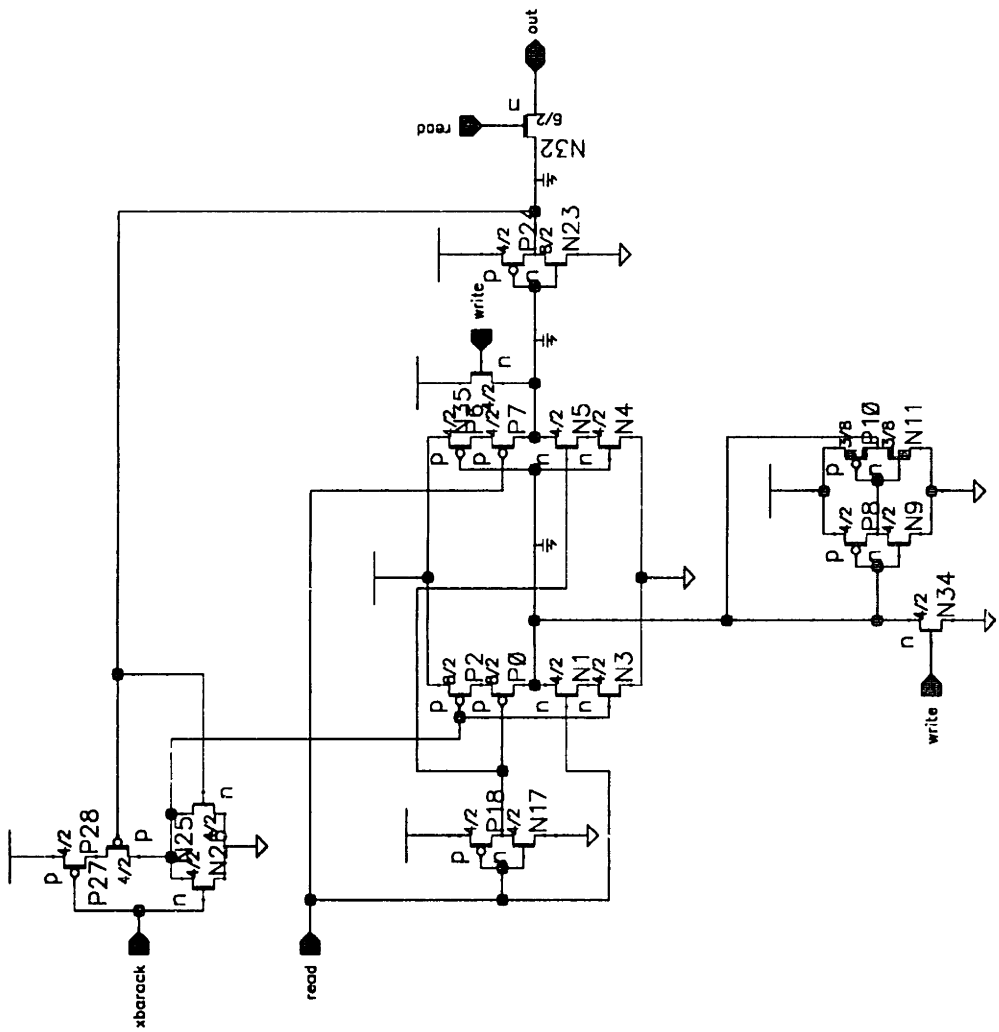


Figure A-63: Library fifo5, cell copied

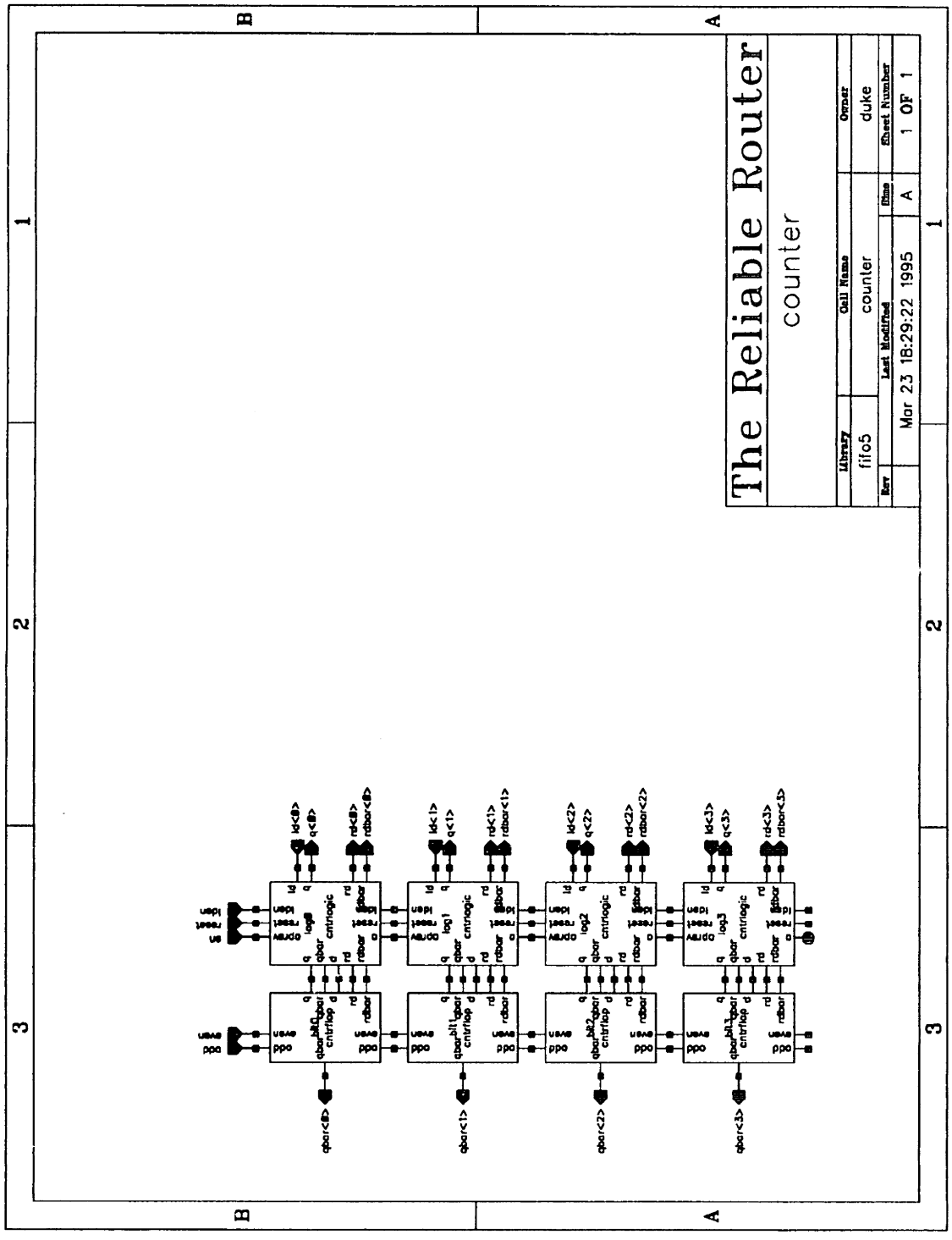


Figure A-64: Library fifo5, cell counter

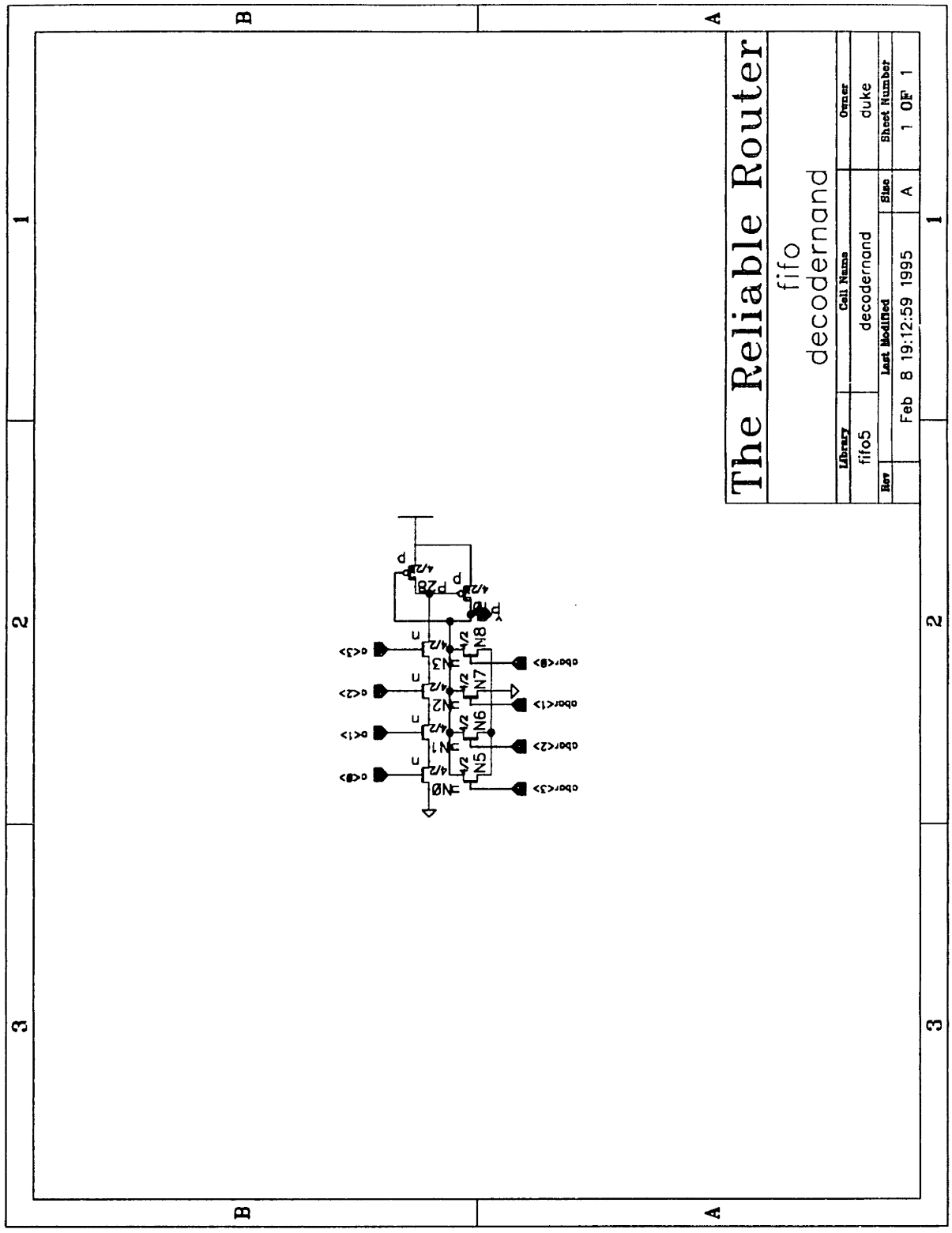


Figure A-65: Library fifo5, cell decodernand

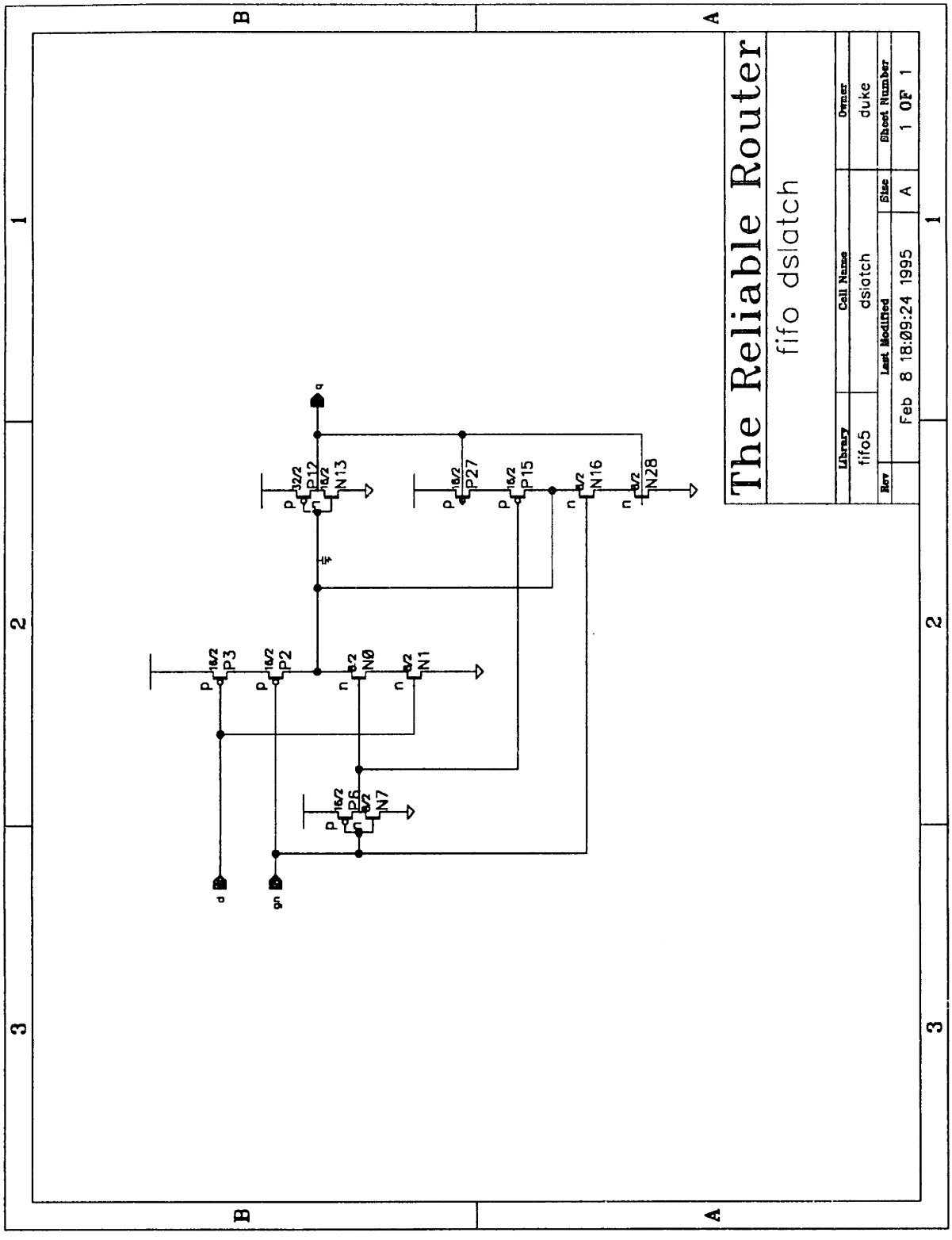


Figure A-66: Library fifo5, cell dslatch

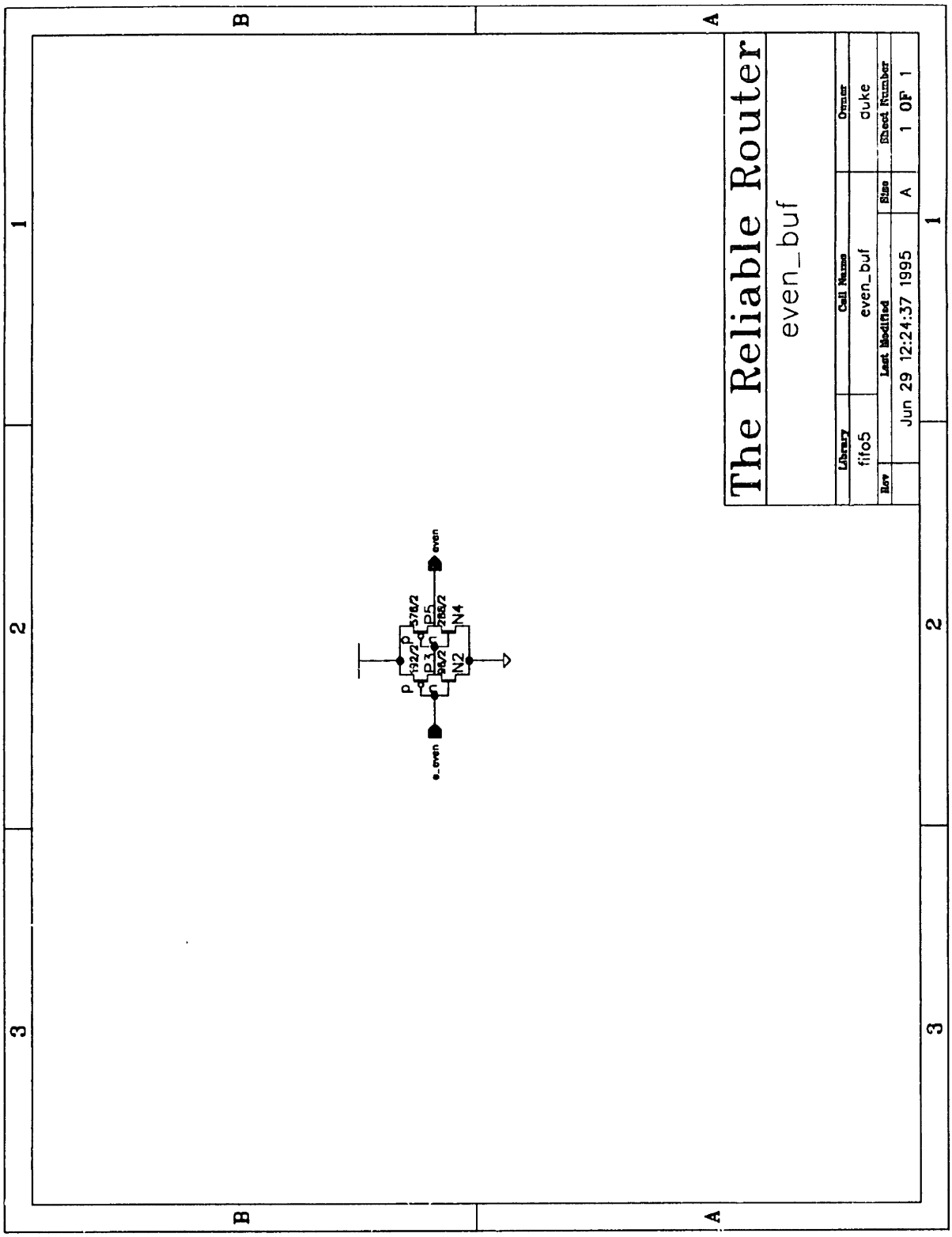


Figure A-67: Library fifo5, cell even_buf

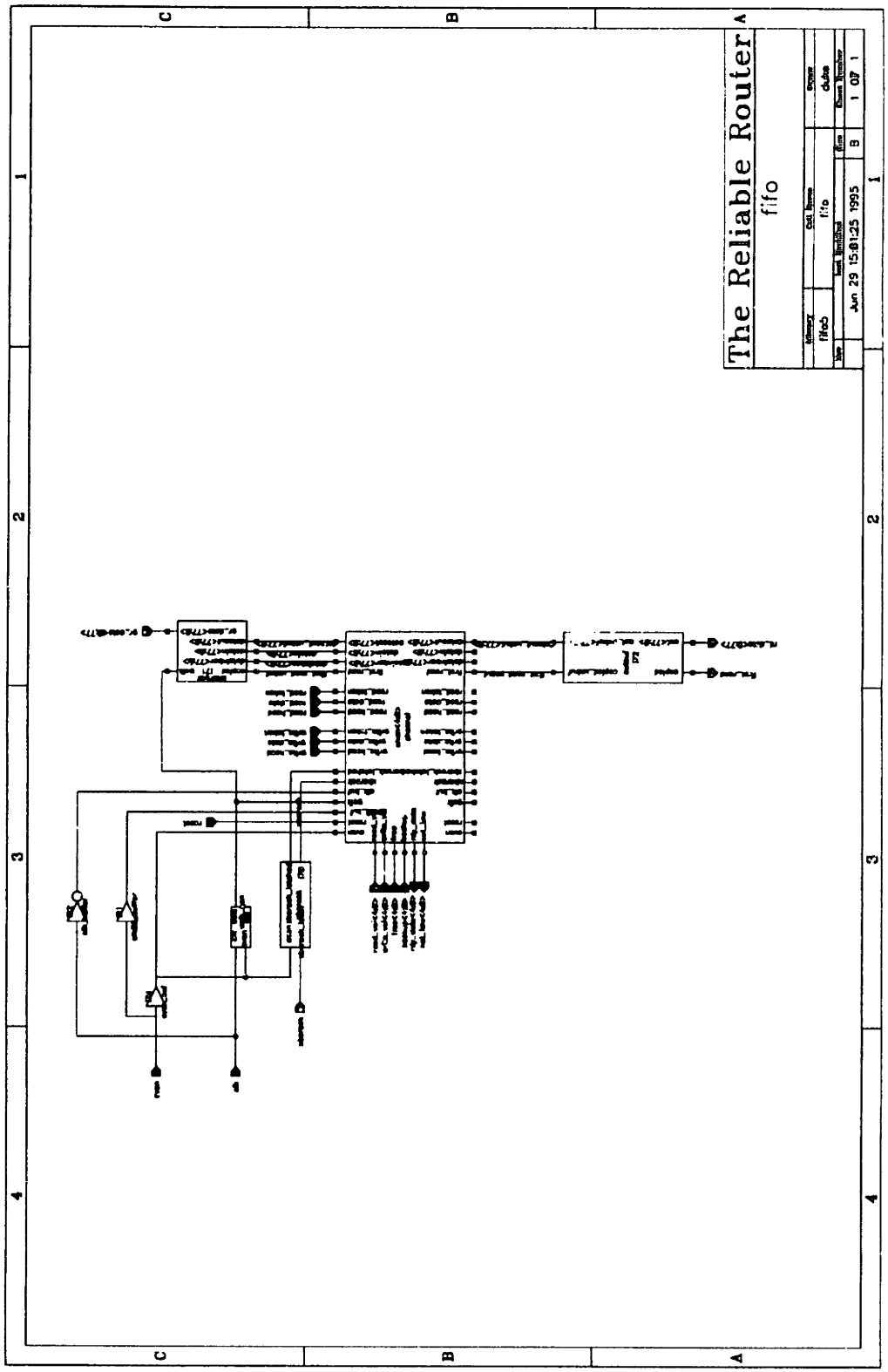


Figure A-68: Library fifo5, cell fifo

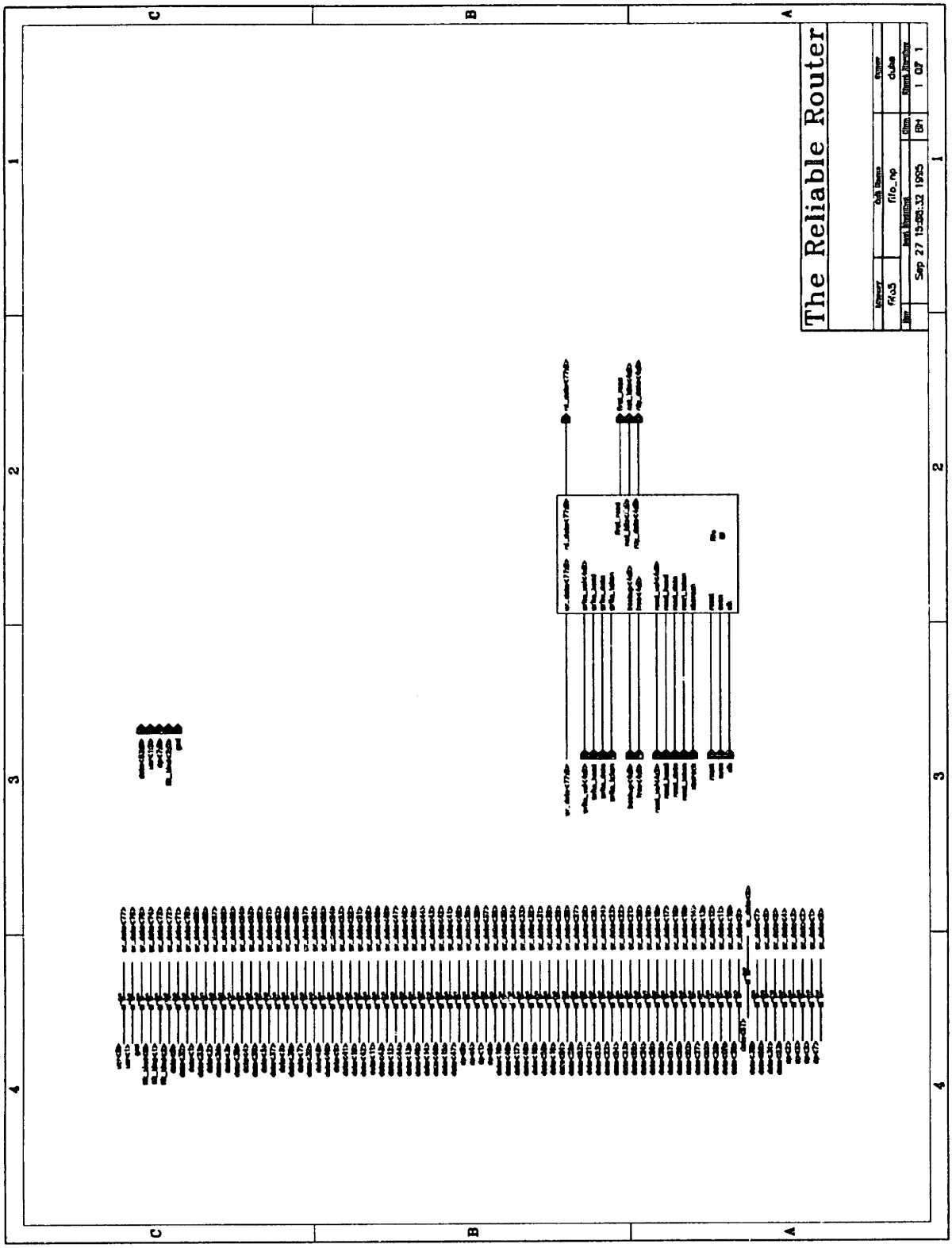


Figure A-69: Library fifo5, cell fifo_np

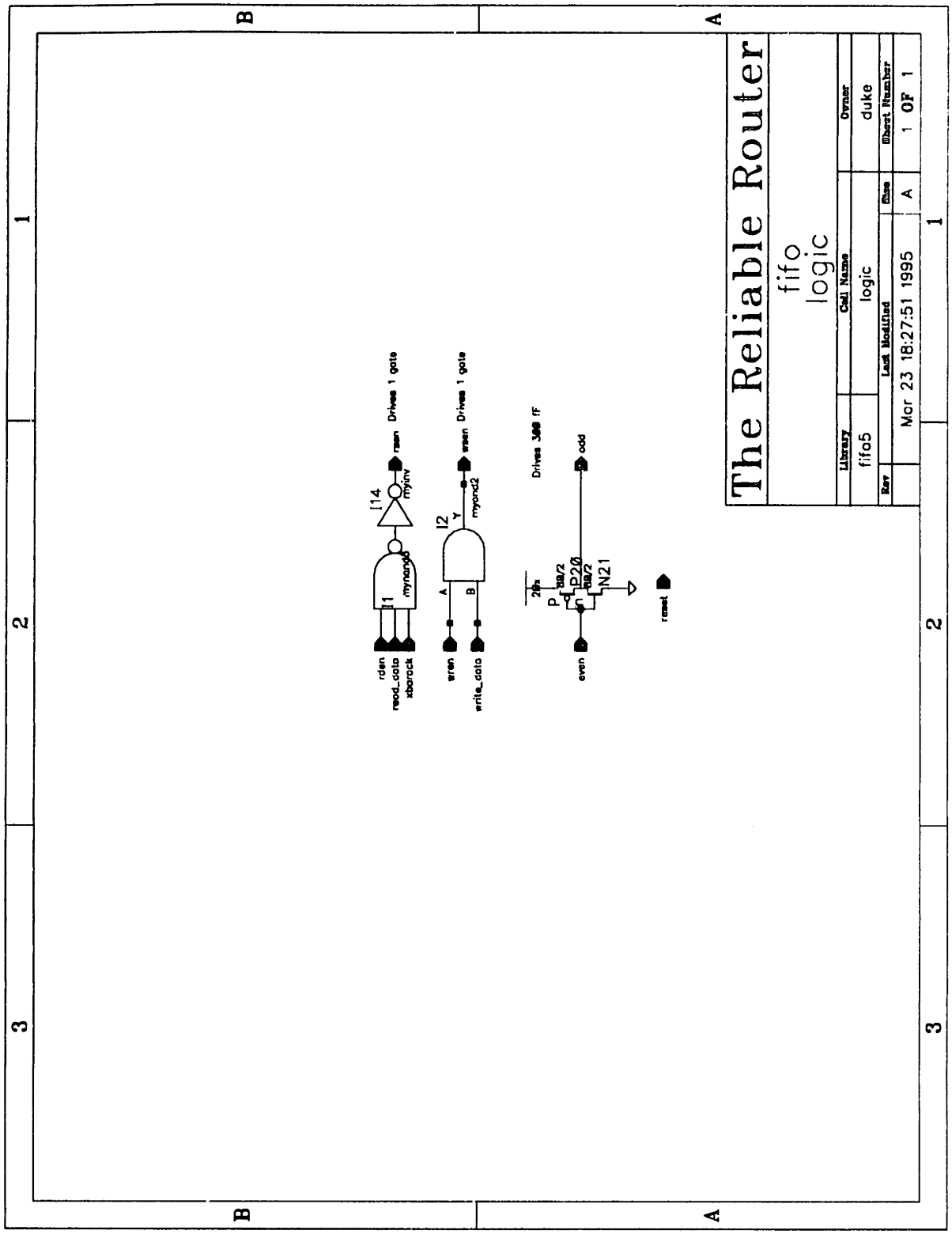


Figure A-70: Library fifo5, cell logic

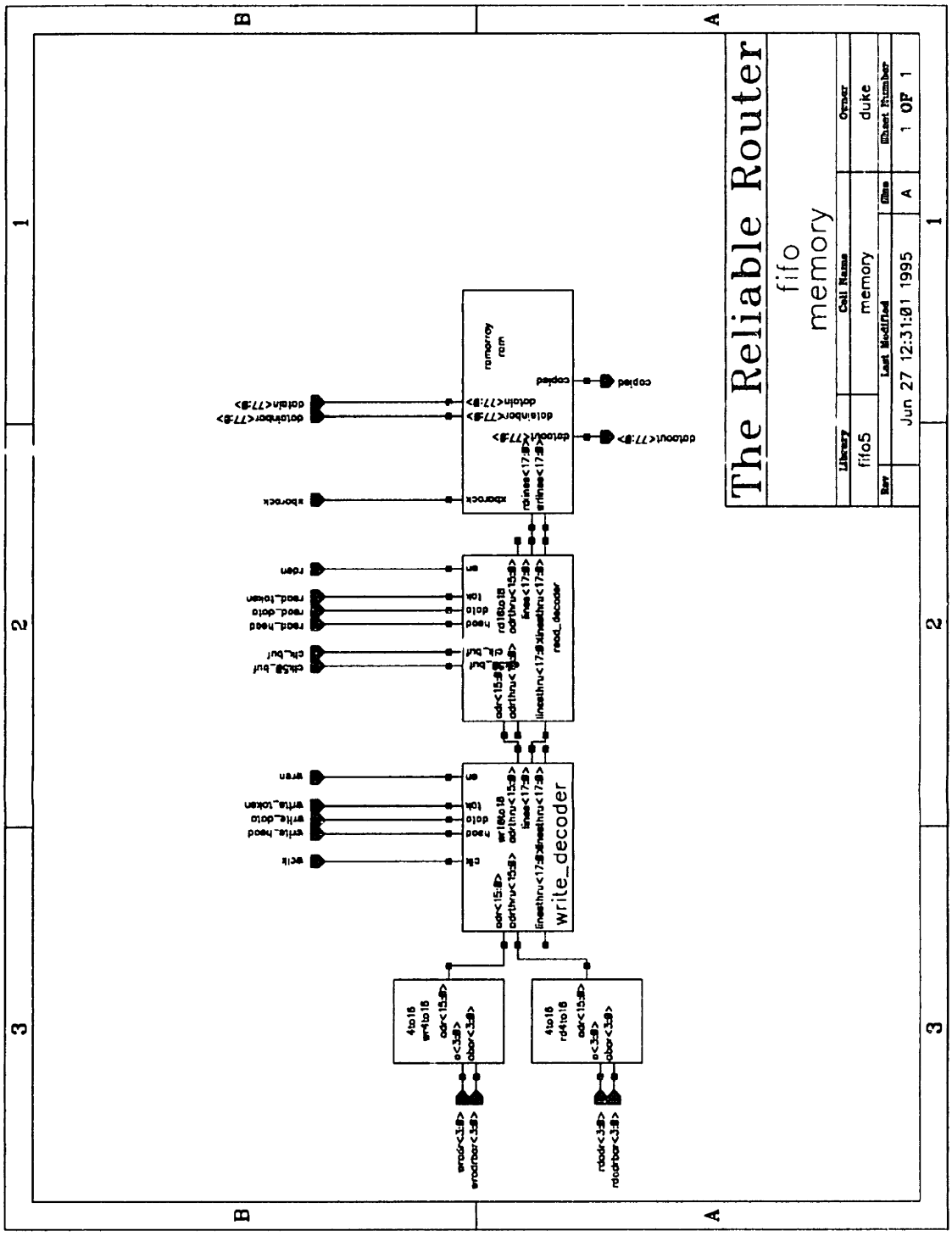


Figure A-71: Library fifo5, cell memory

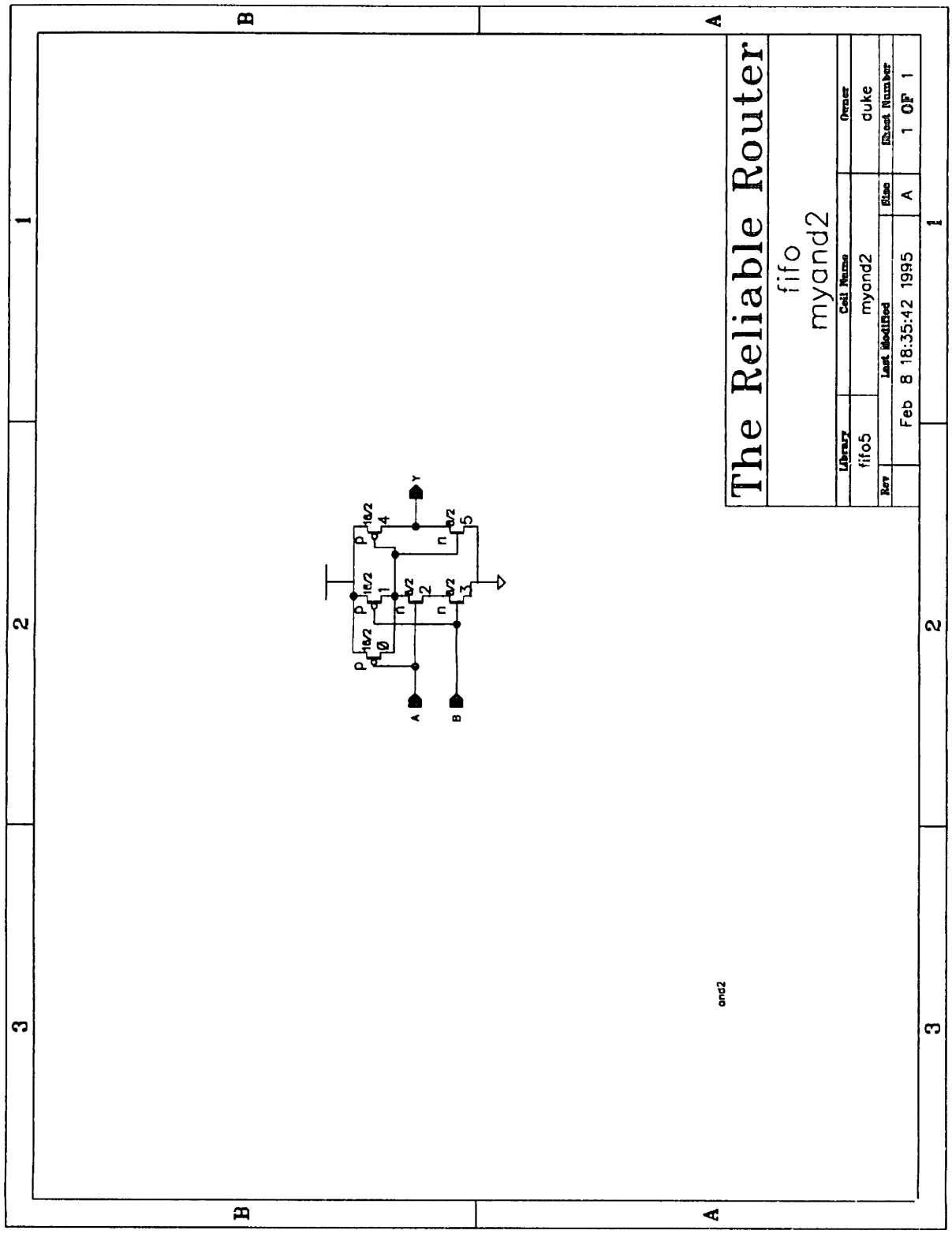


Figure A-72: Library fifo5, cell myand2

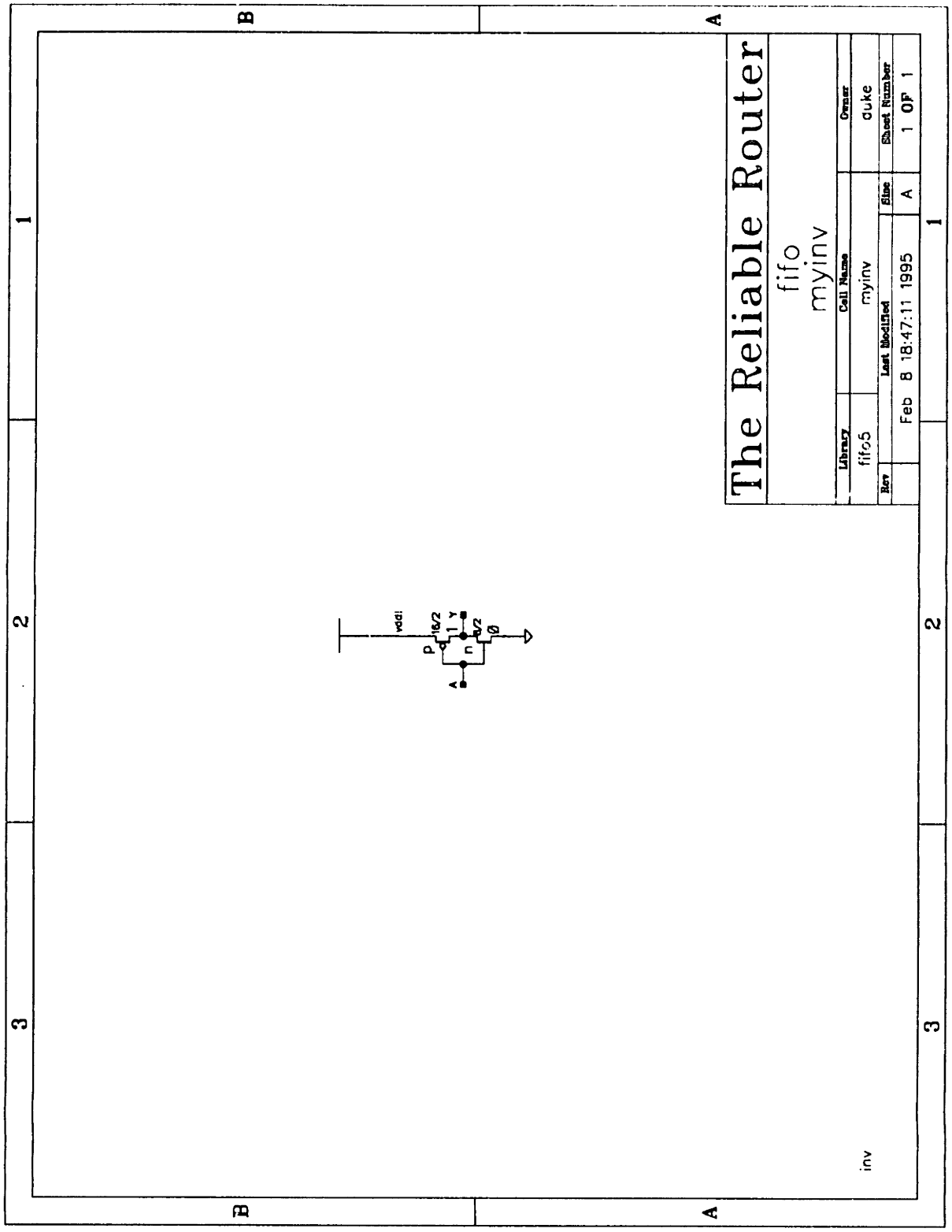


Figure A-73: Library fifo5, cell myinv

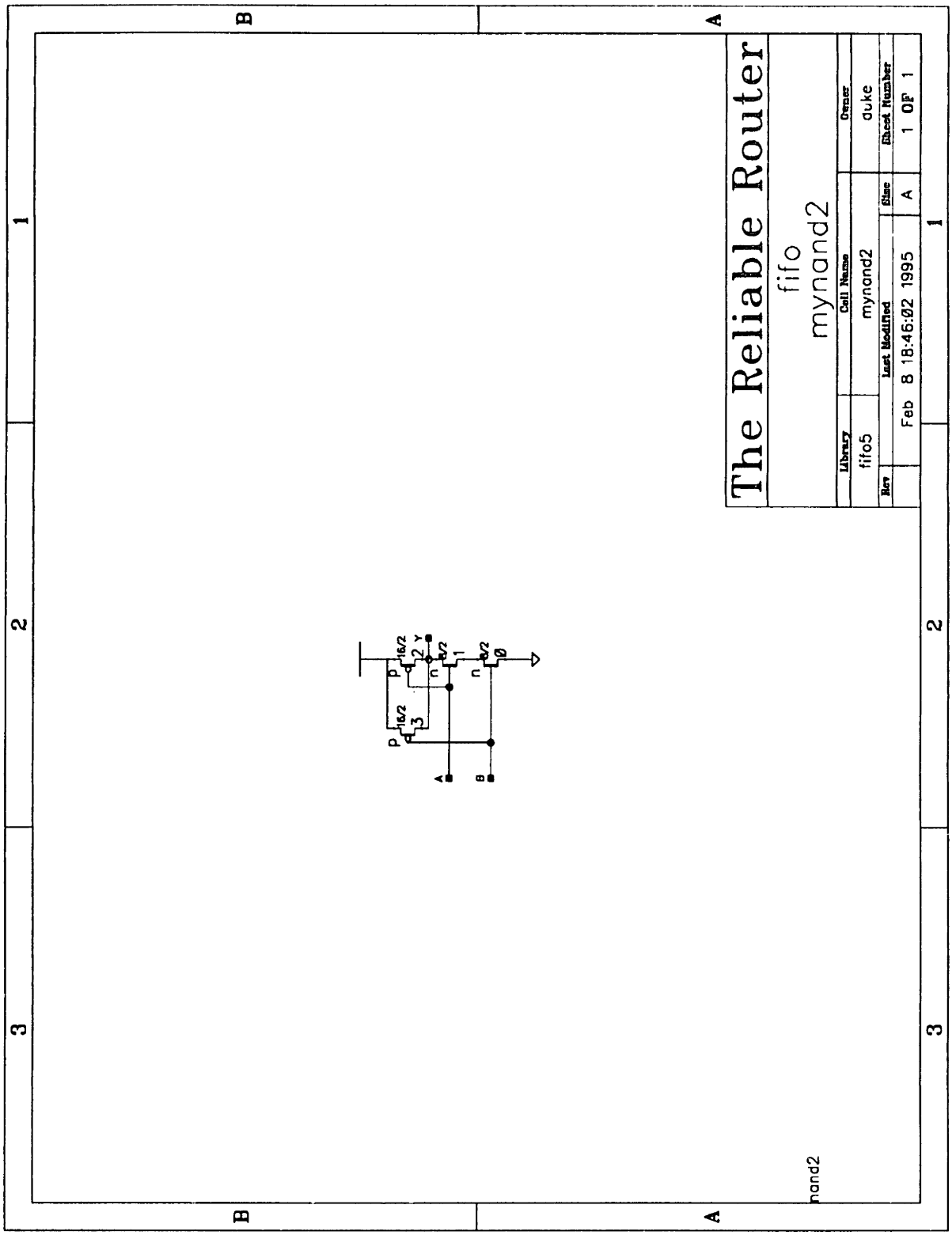


Figure A-74: Library fifo5, cell mynand2

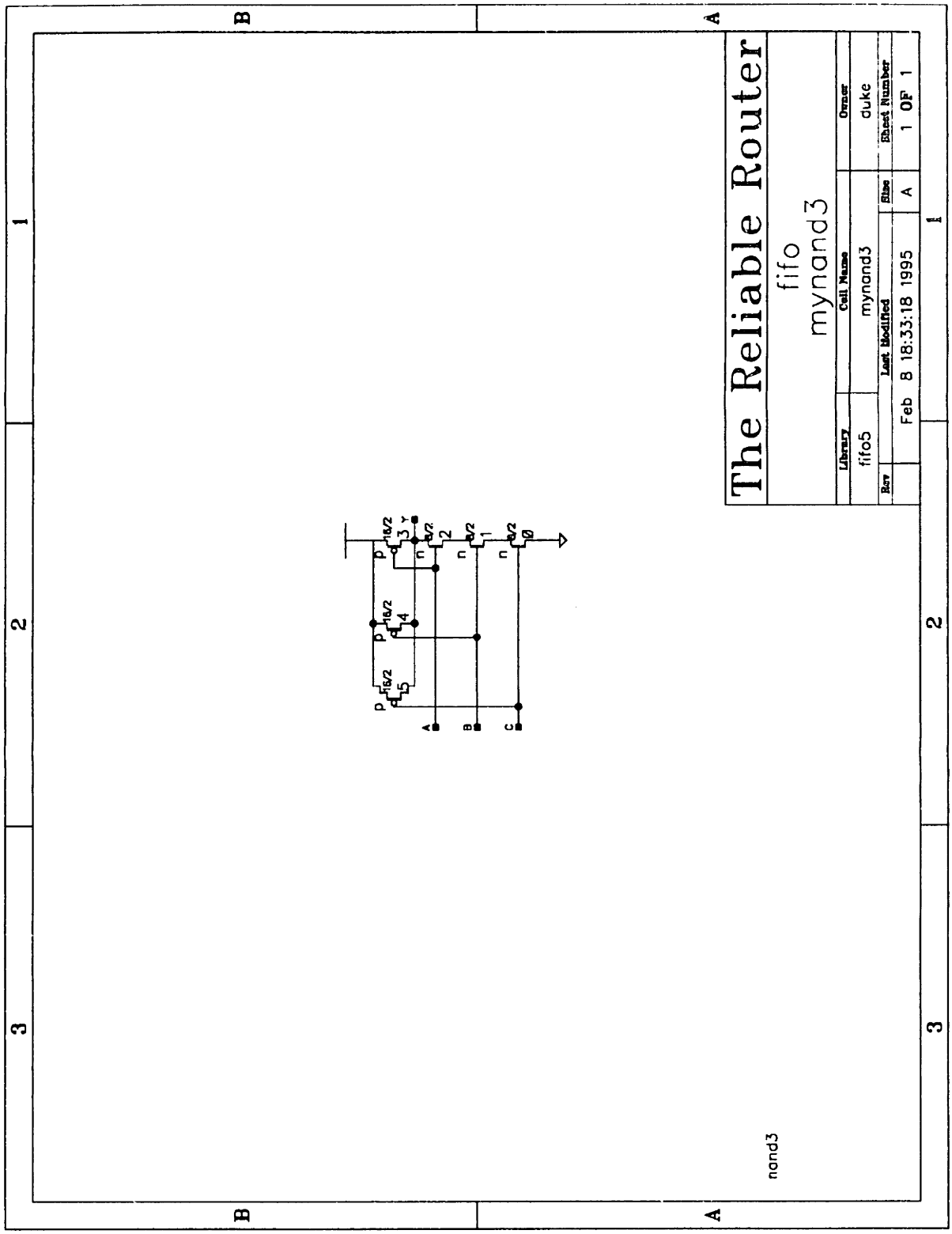


Figure A-75: Library fifo5, cell mynand3

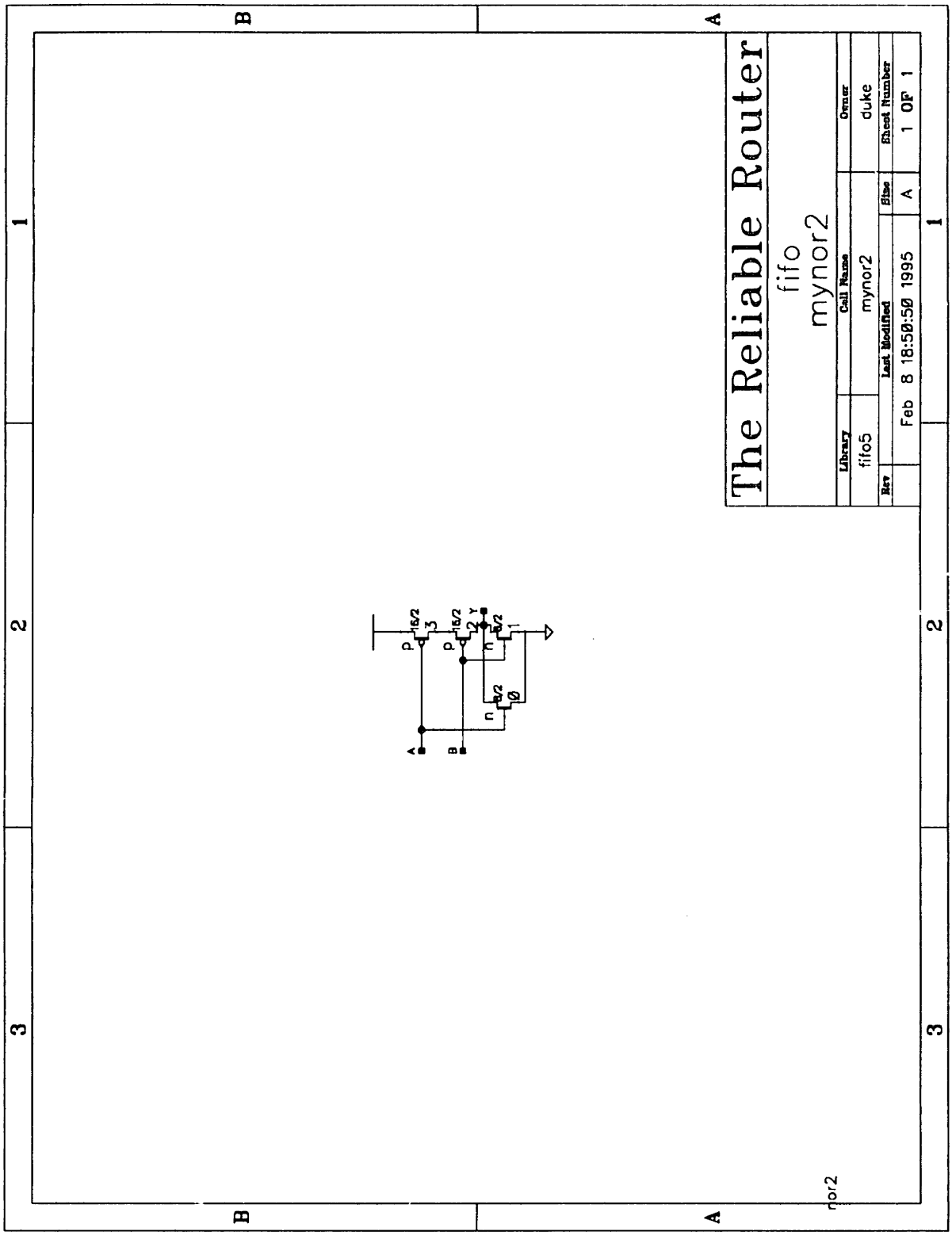


Figure A-76: Library fifo5, cell mynor2

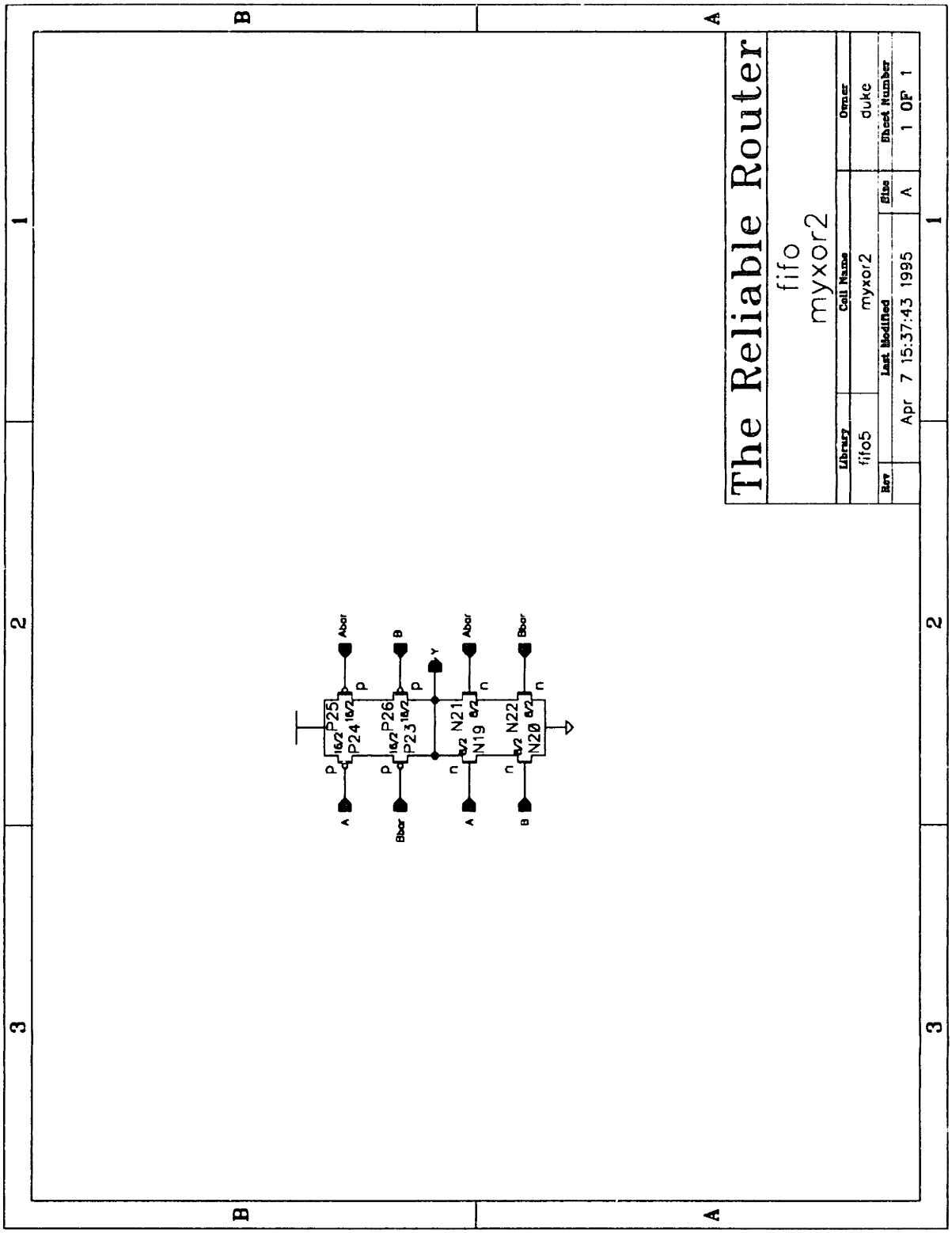


Figure A-77: Library fifo5, cell myxor2

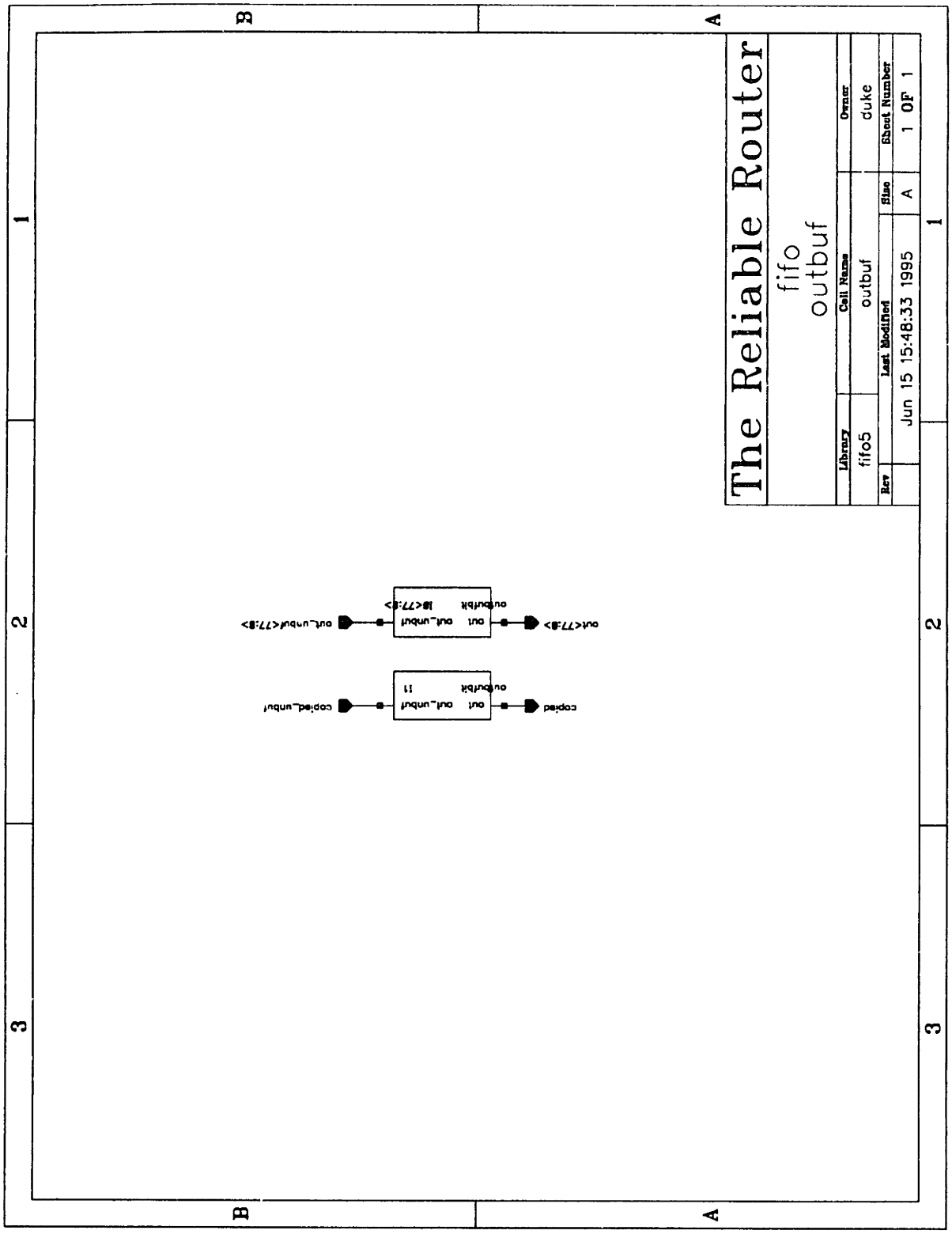


Figure A-78: Library fifo5, cell outbuf

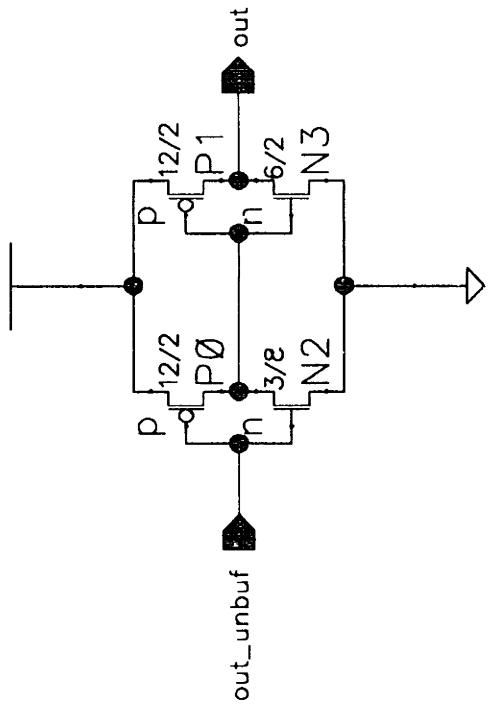


Figure A-79: Library fifo5, cell outbufbit

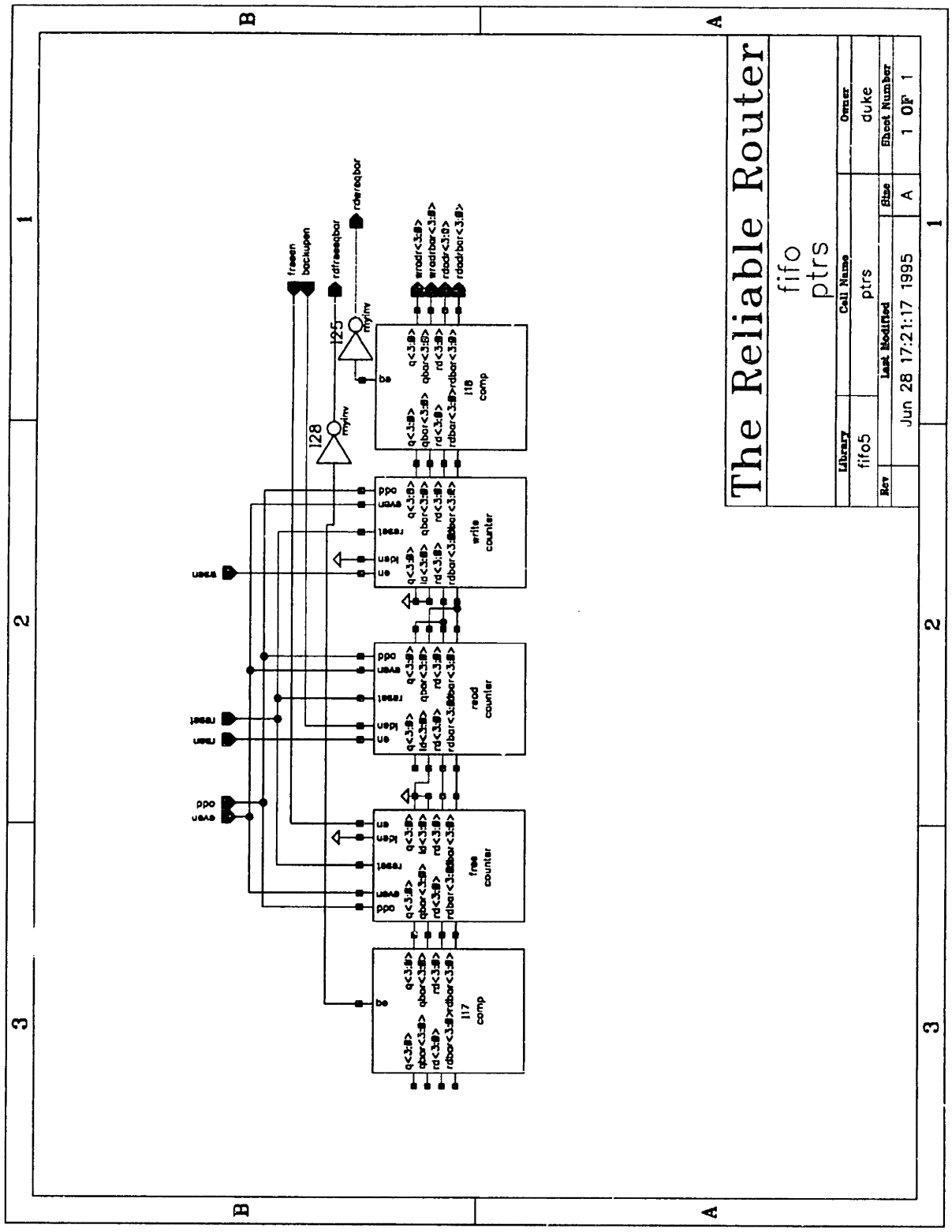


Figure A-80: Library fifo5, cell ptrs

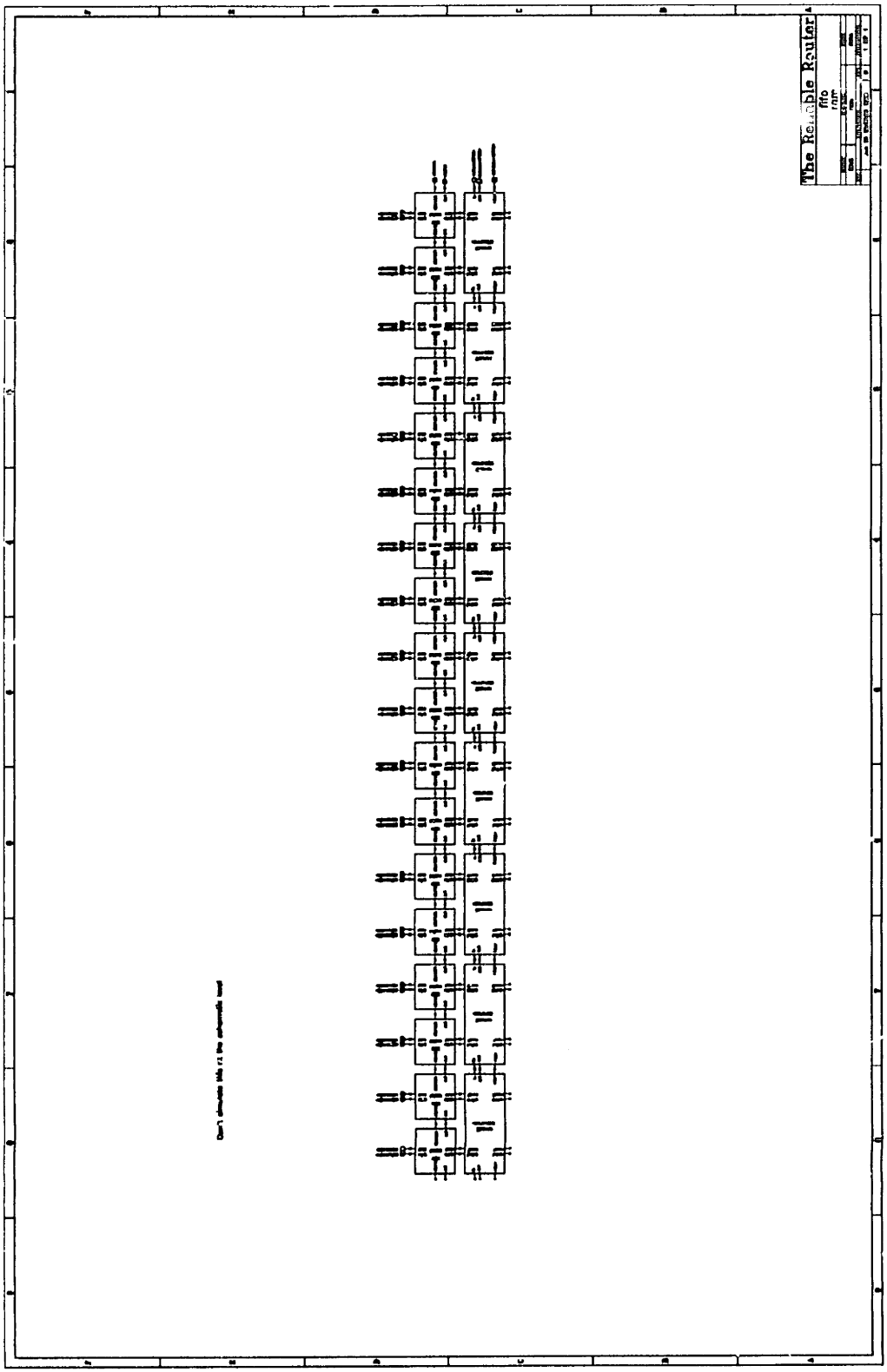


Figure A-81: Library fifo5, cell ram

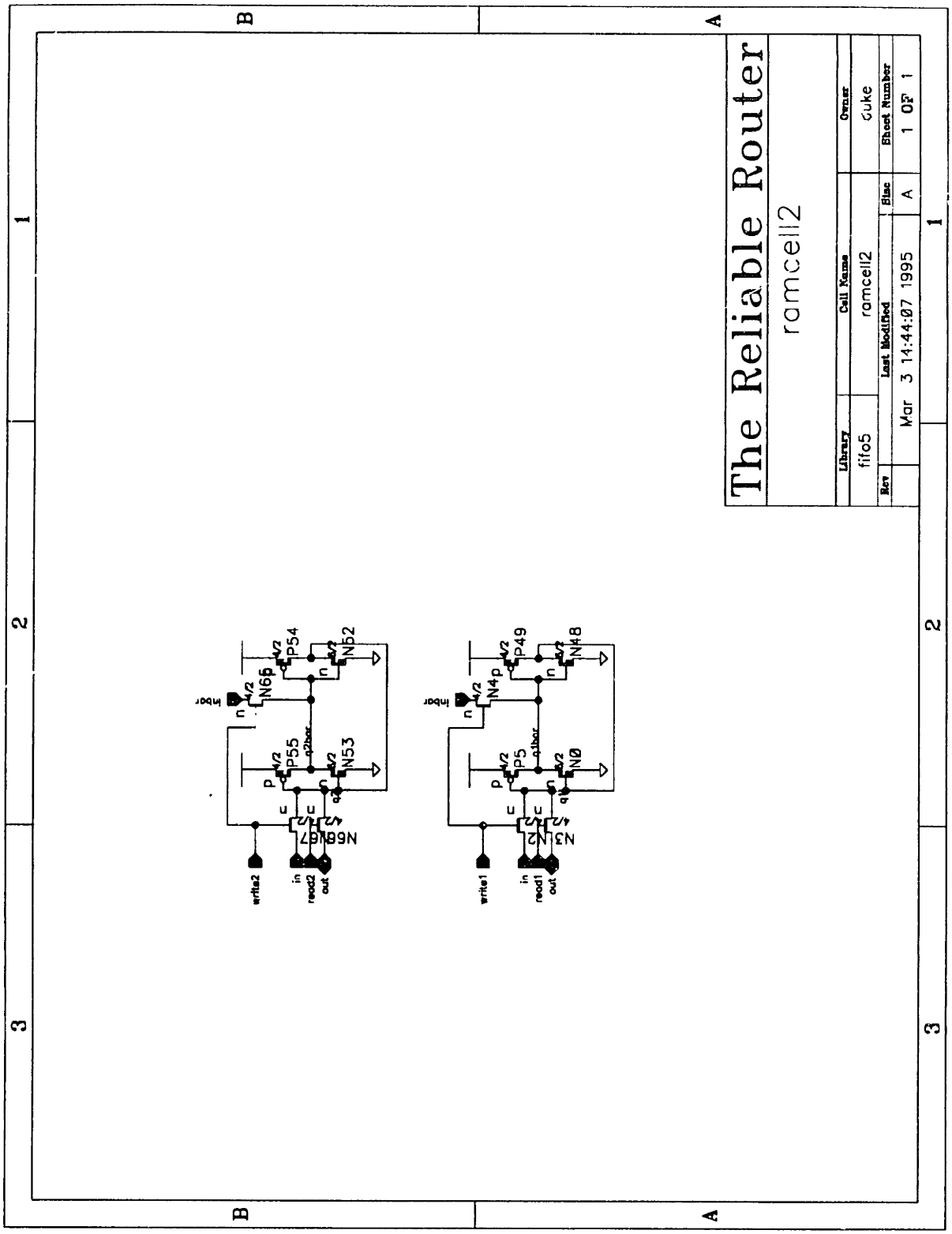


Figure A-82: Library fifo5, cell ramcell2

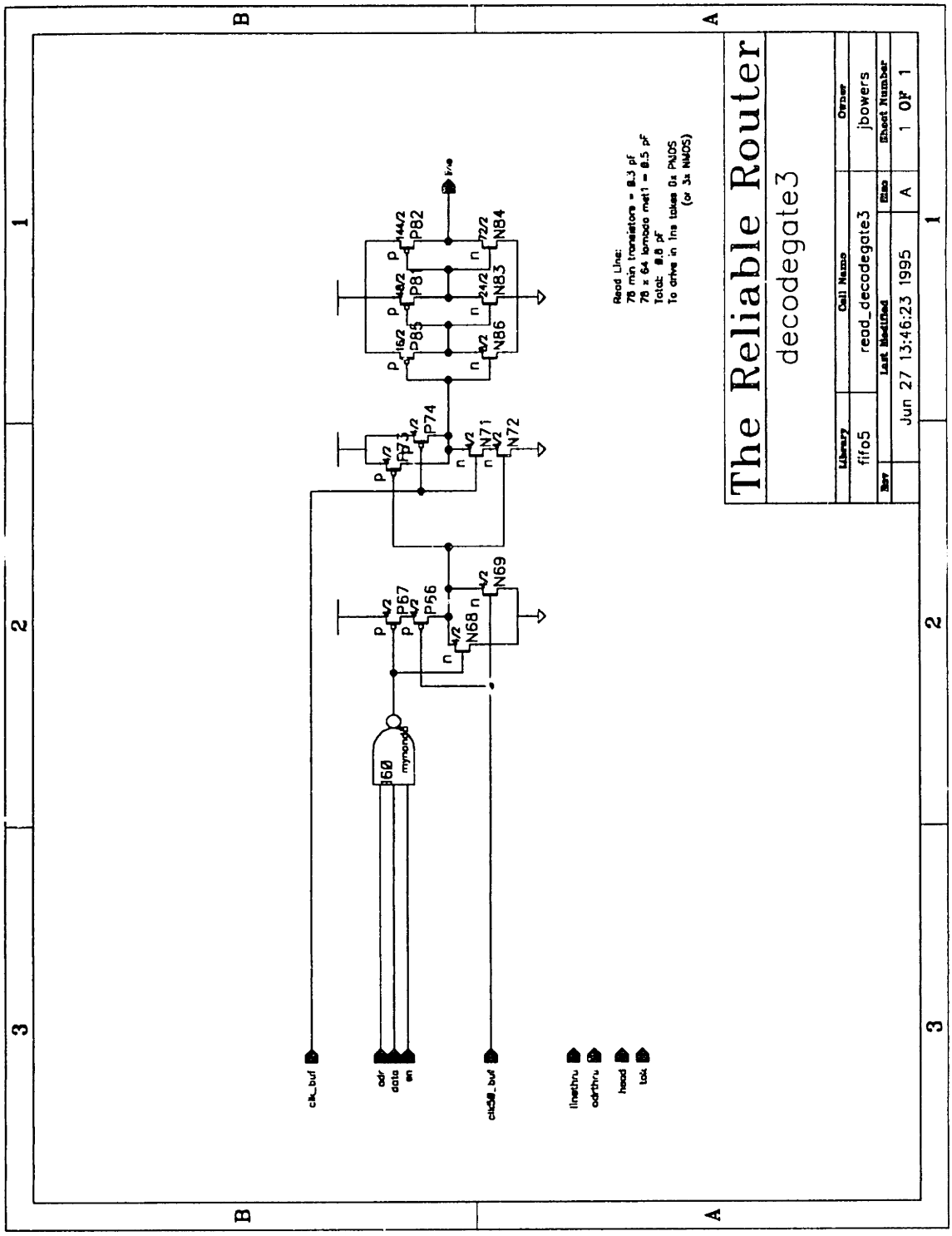


Figure A-83: Library fifo5, cell read_decodegate3

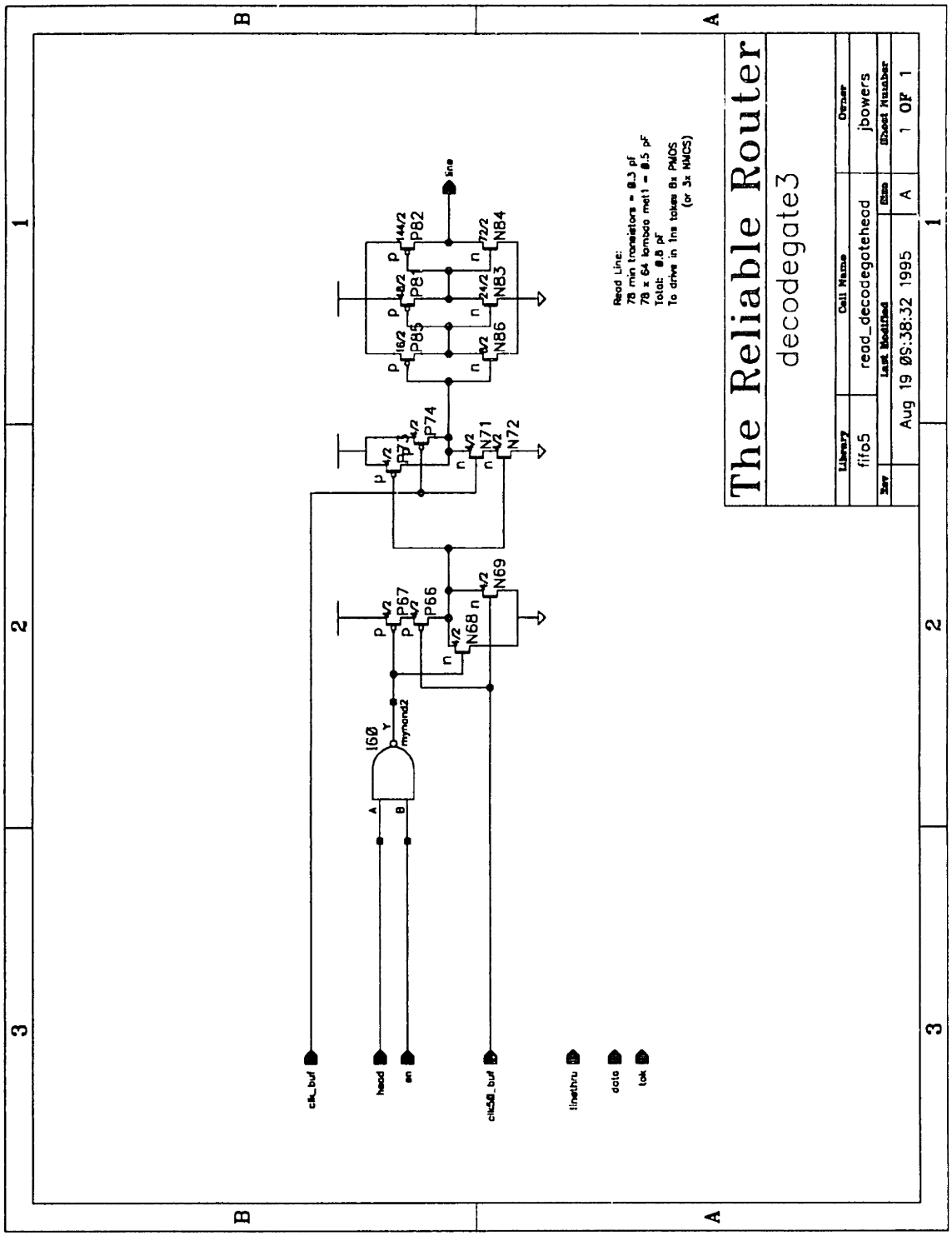


Figure A-84: Library fifo5, cell read_decodegatehead

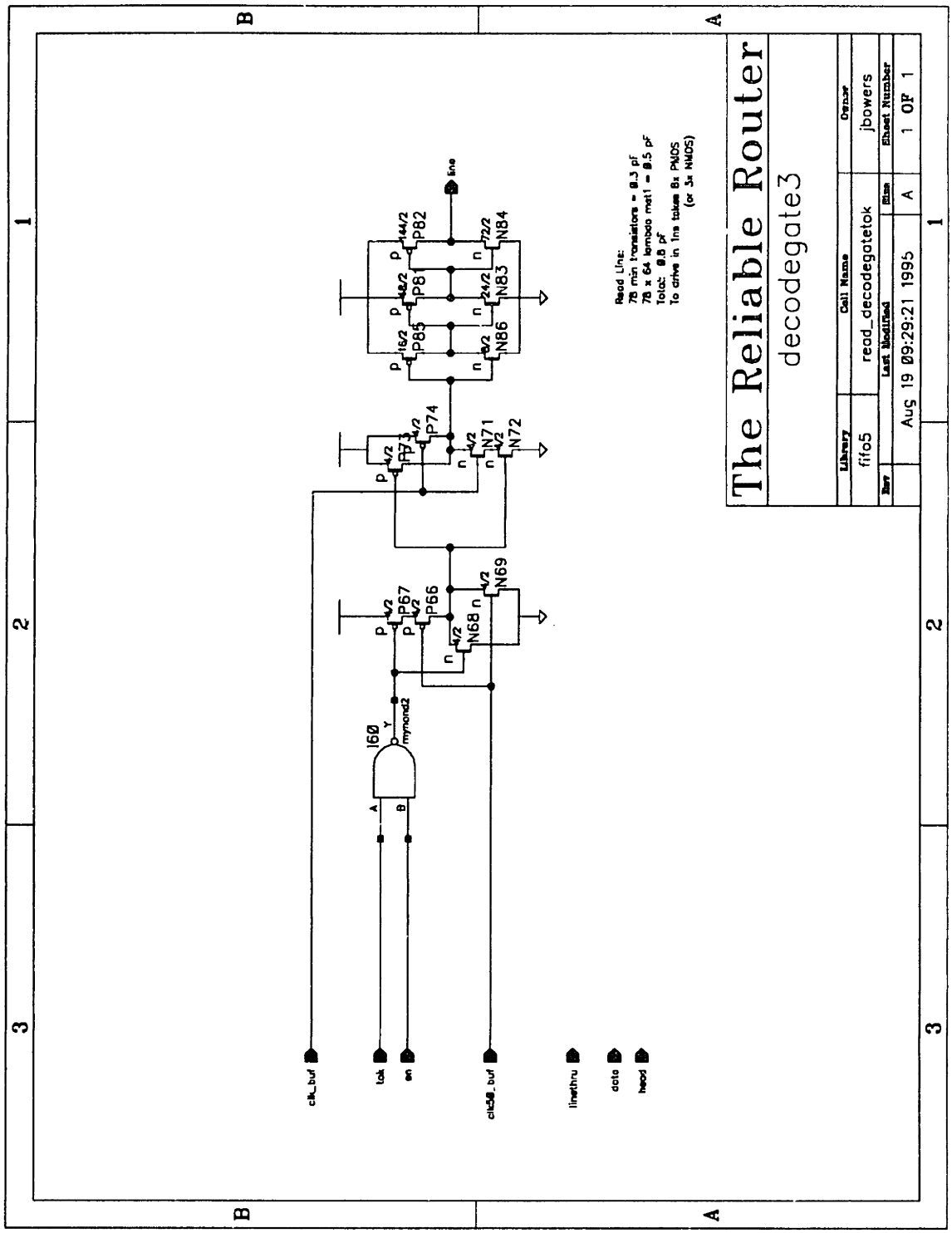


Figure A-85: Library fifo5, cell read_decodegatetok

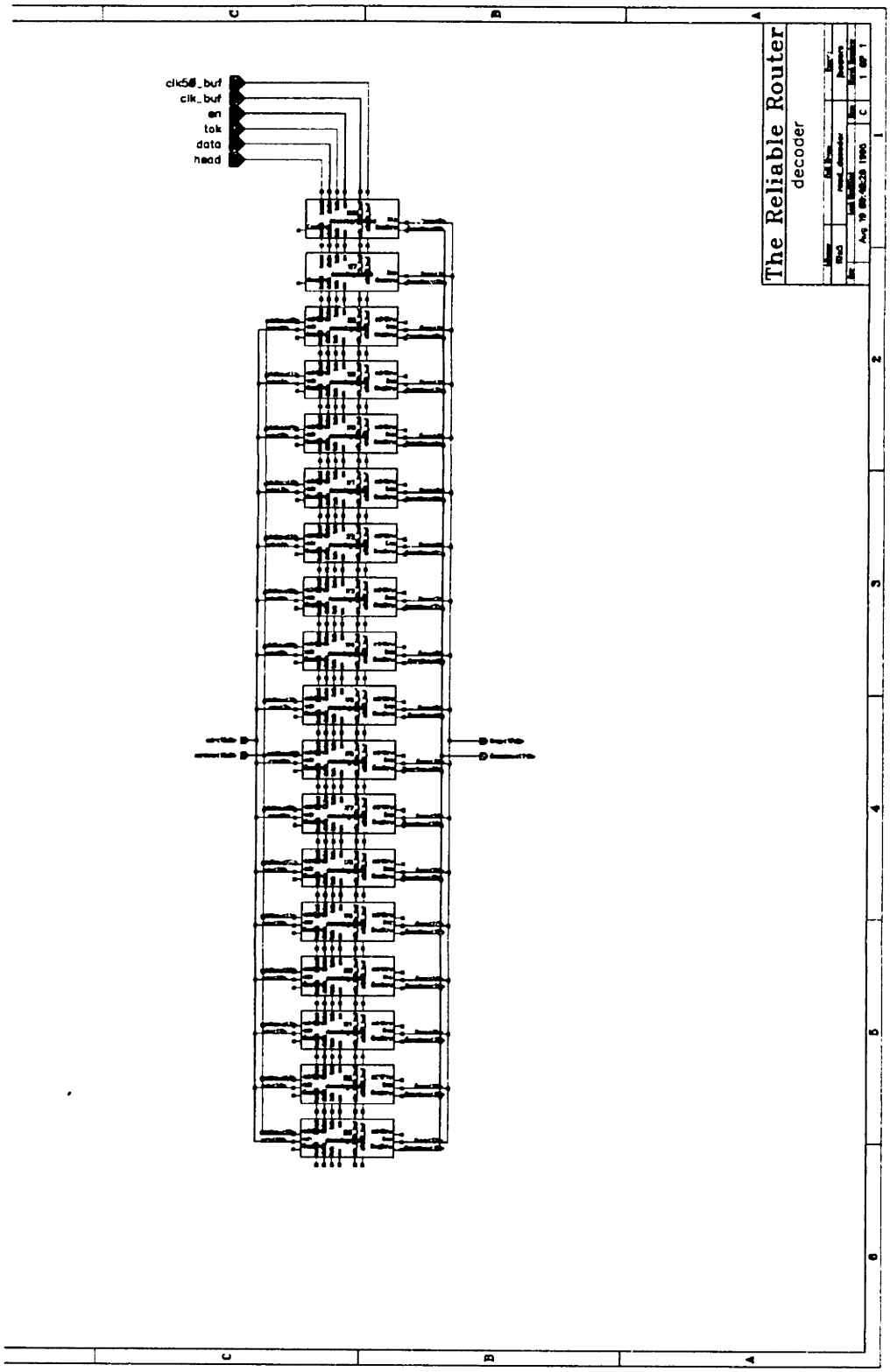


Figure A-86: Library fifo5, cell read_decoder

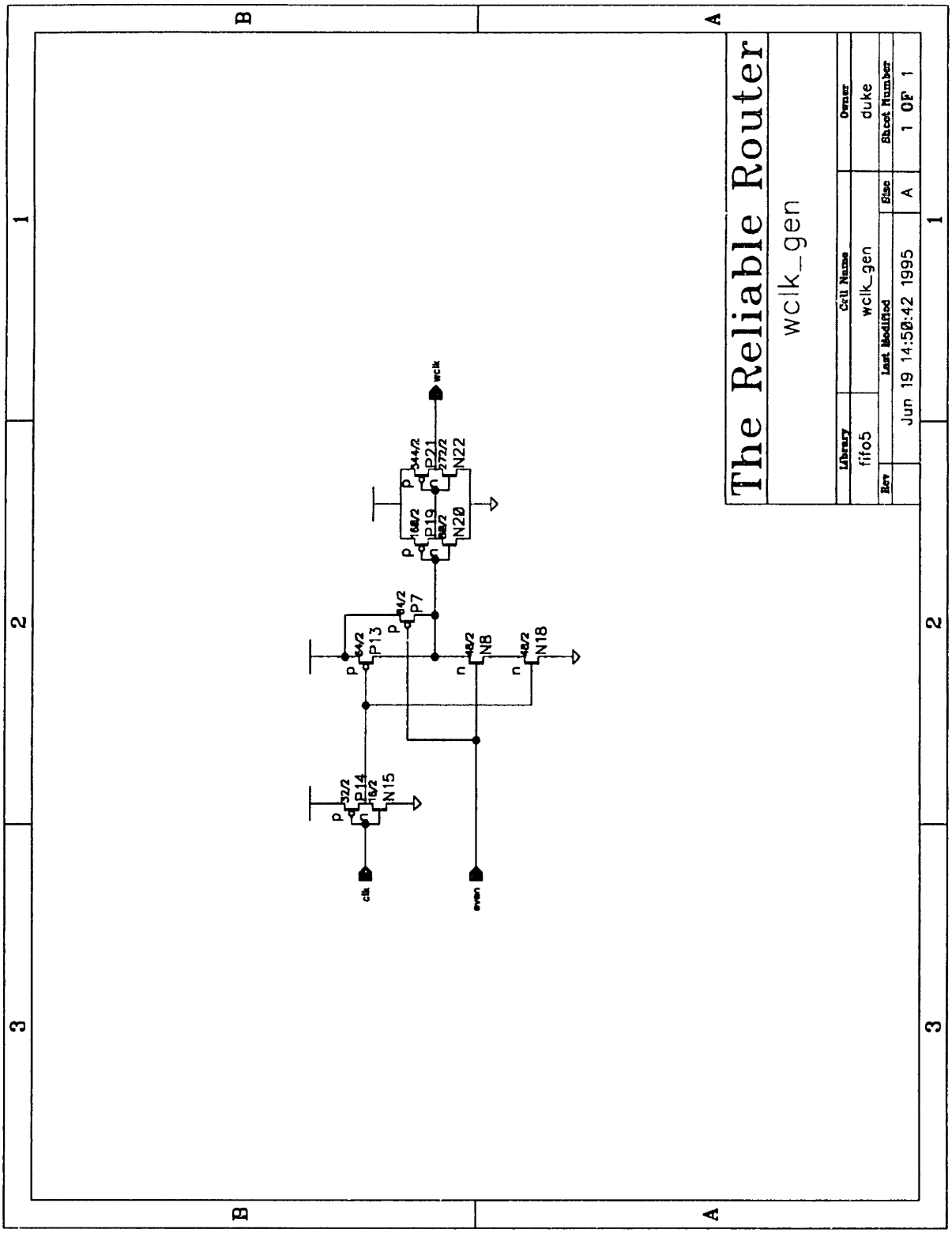


Figure A-87: Library fifo5, cell wclk_gen

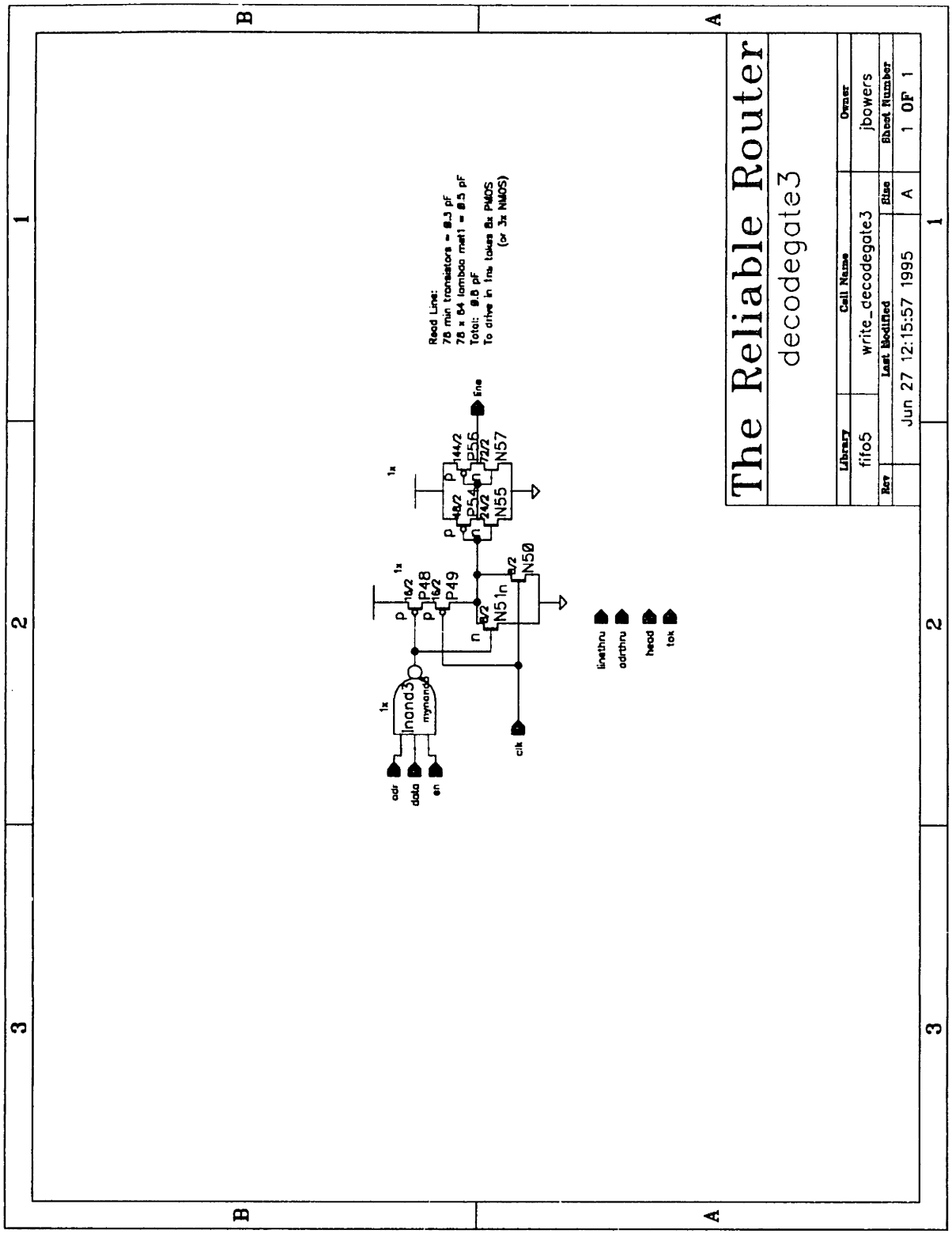


Figure A-88: Library fifo5, cell write_decodegate3

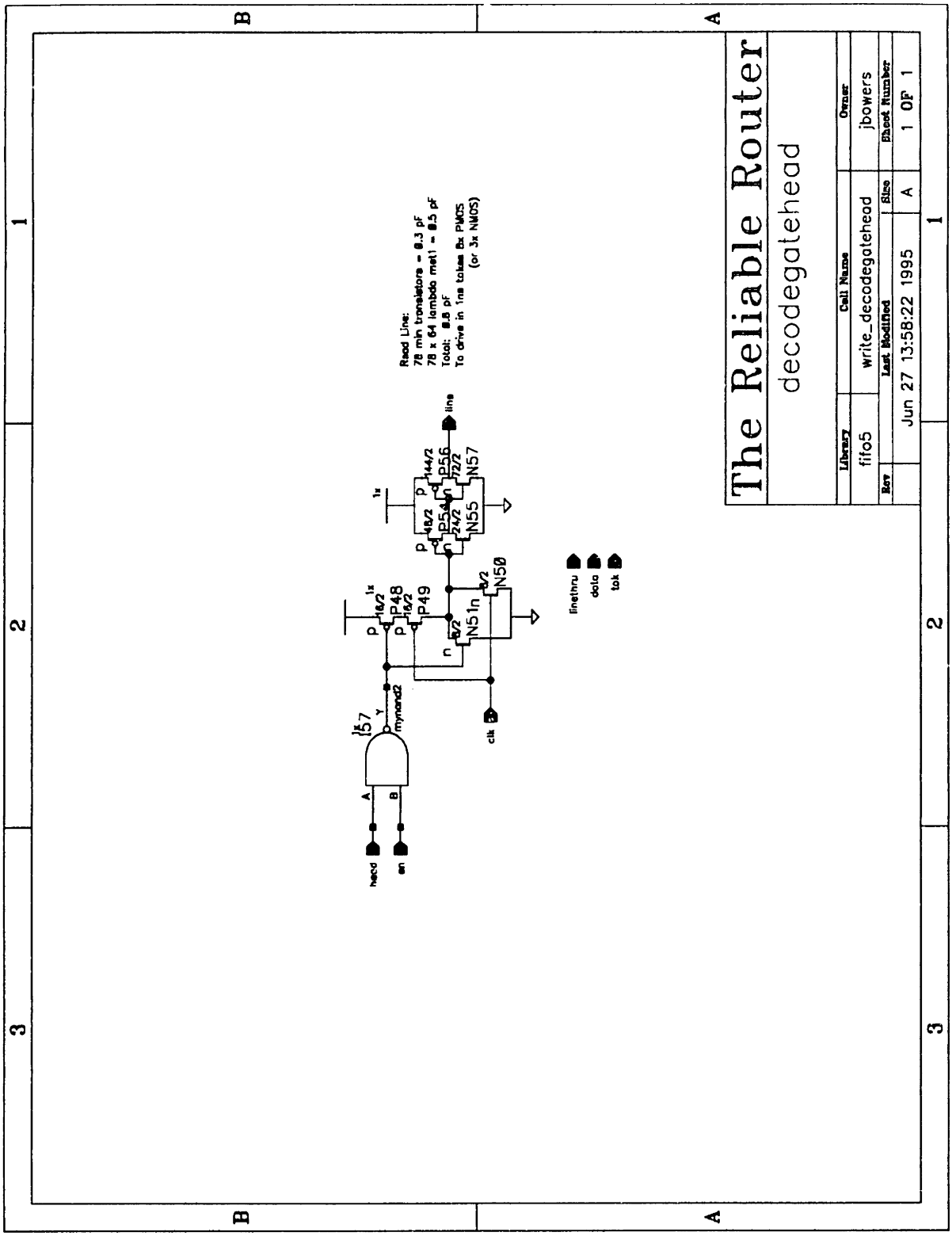


Figure A-89: Library fifo5, cell write_decodegatehead

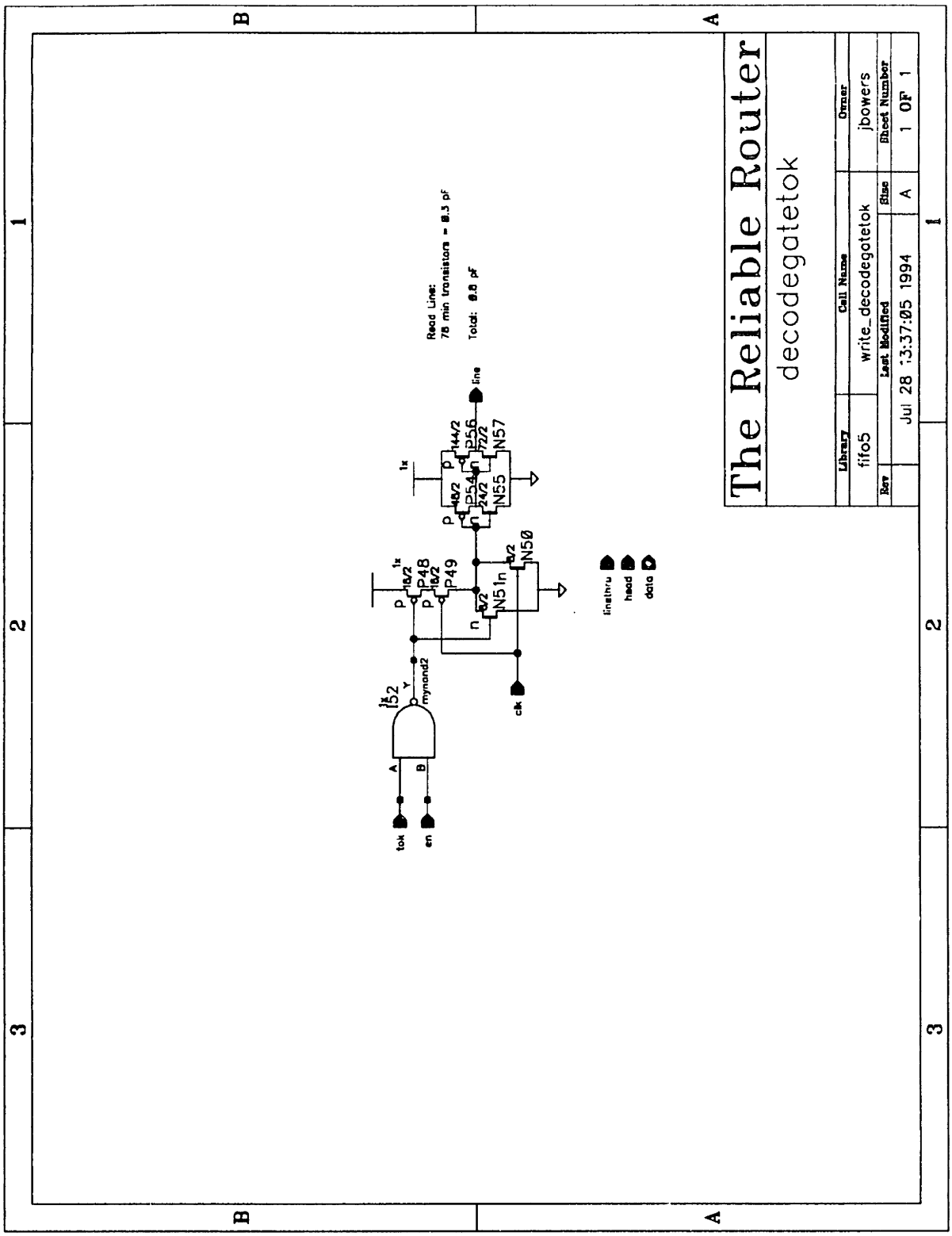


Figure A-90: Library fifo5, cell write_decodegatetok

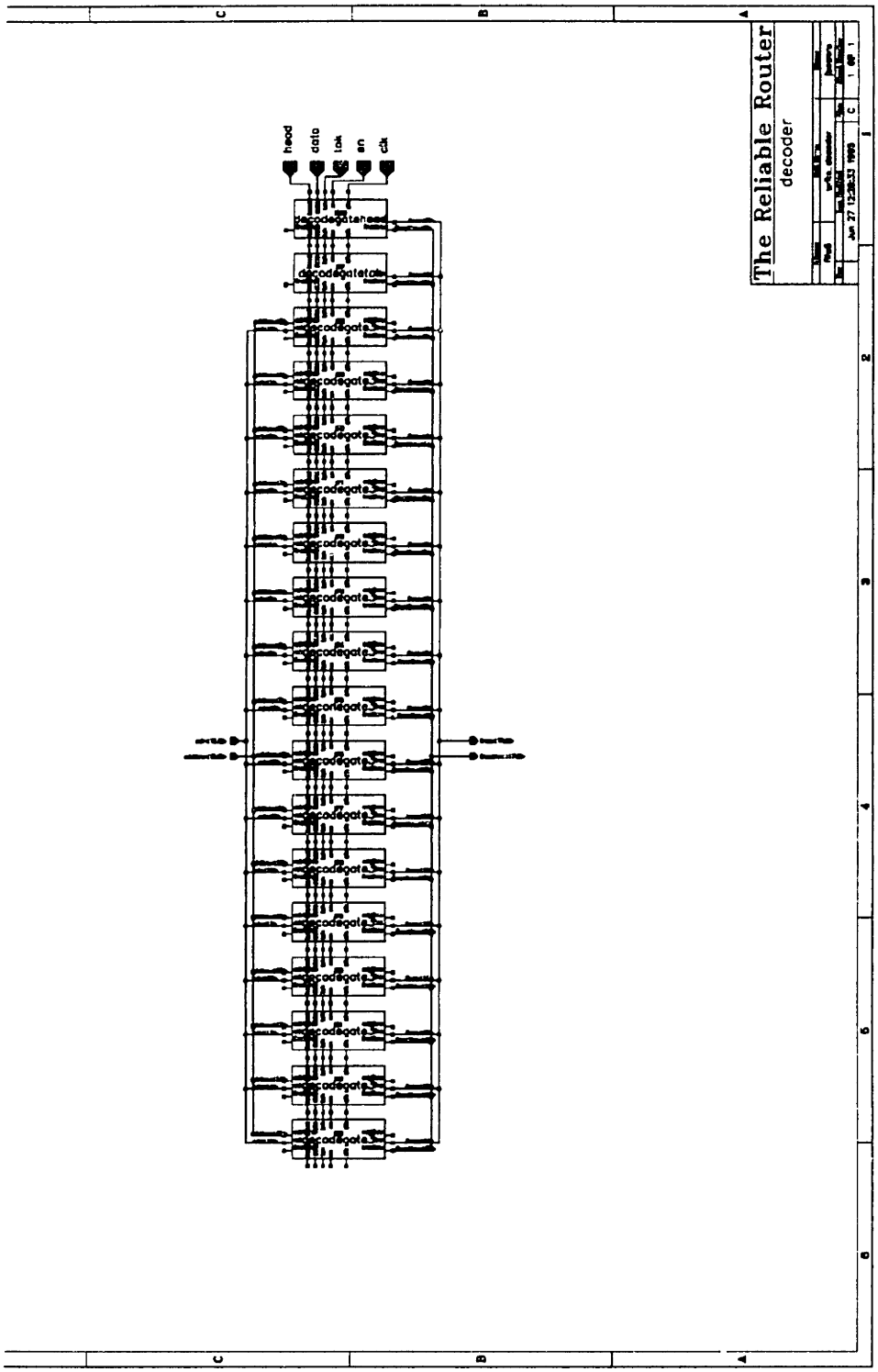


Figure A-91: Library fifo5, cell write_decoder

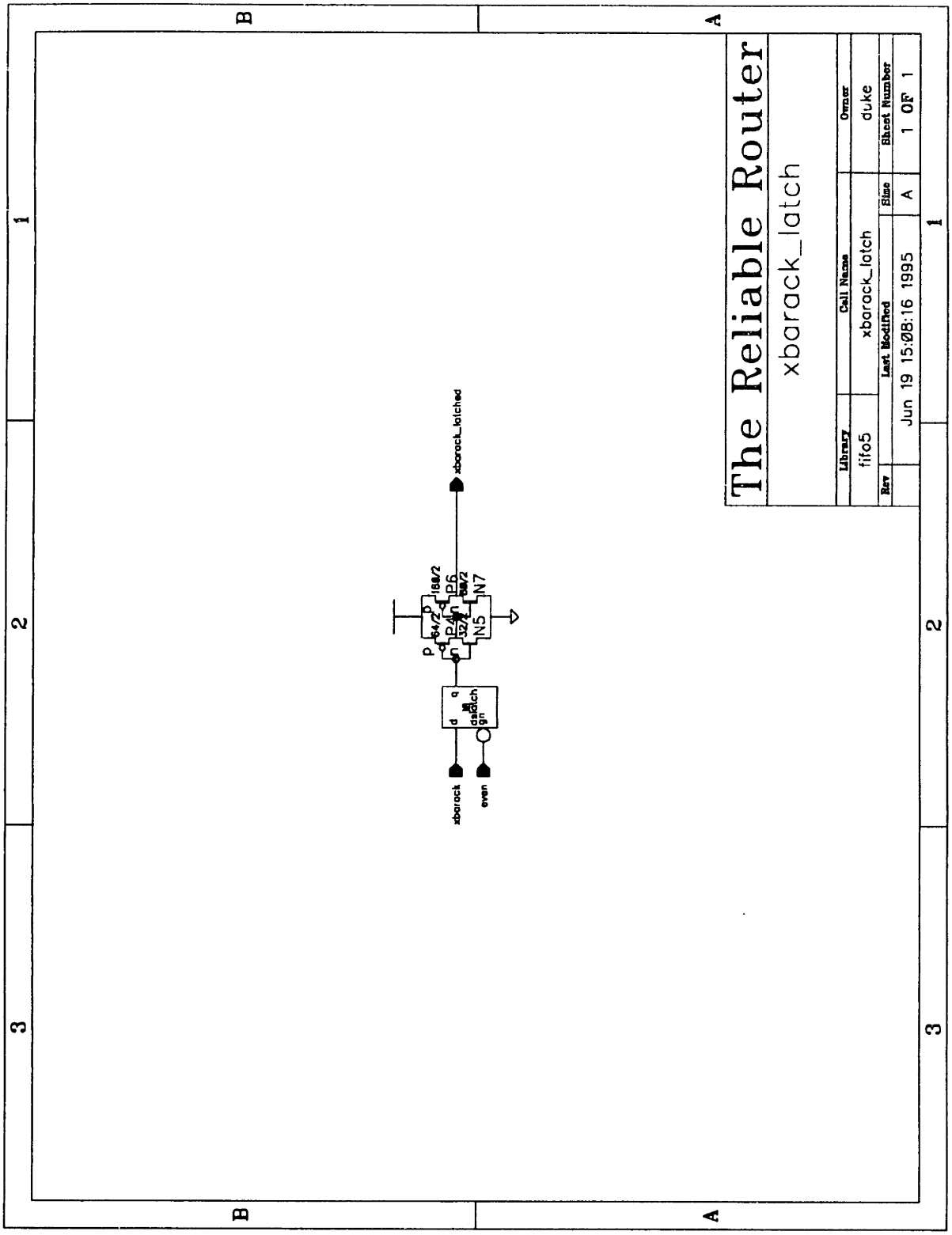


Figure A-92: Library fifo5, cell xbarack_latch

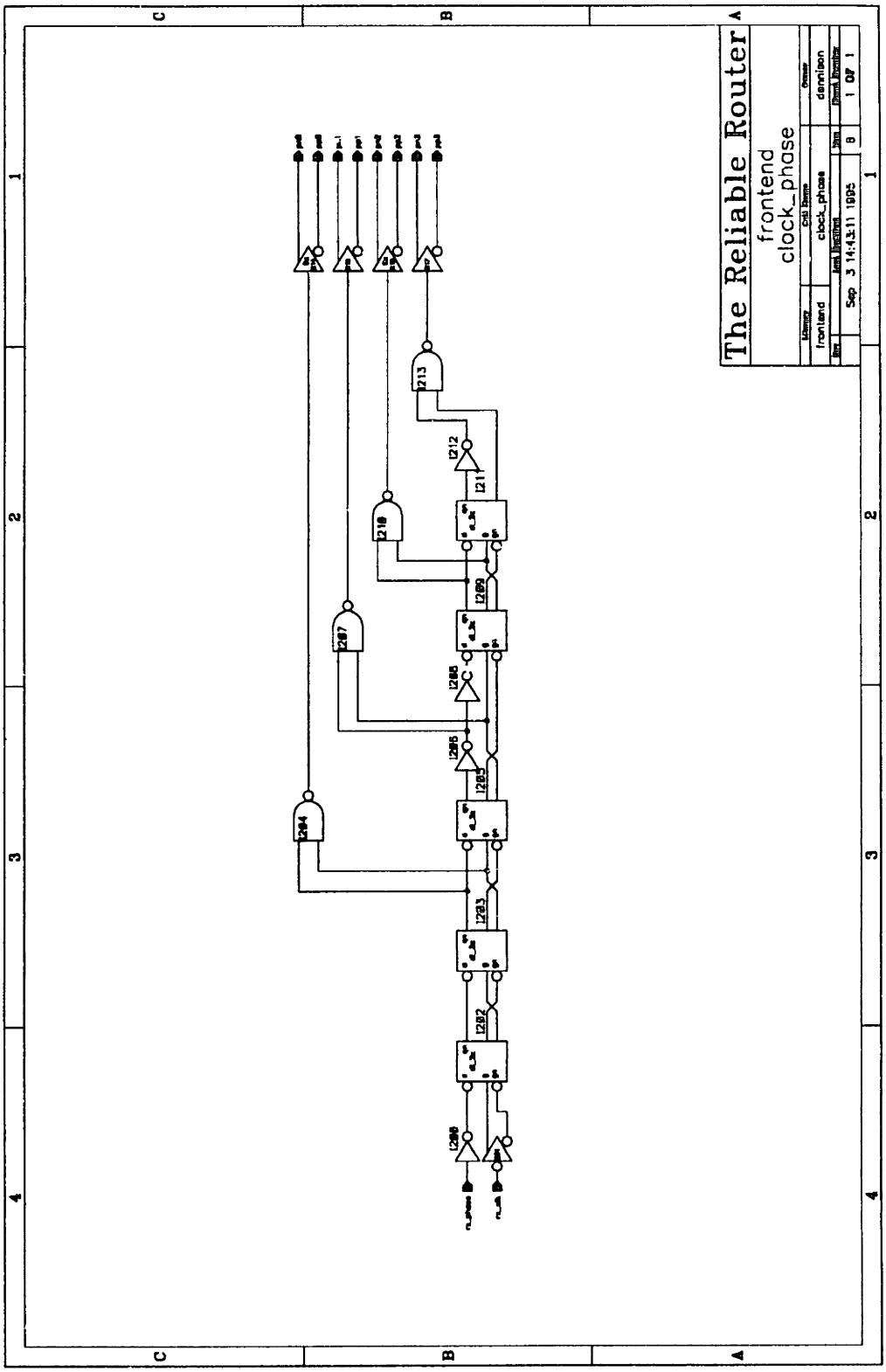


Figure A-93: Library frontend, cell clock_phase

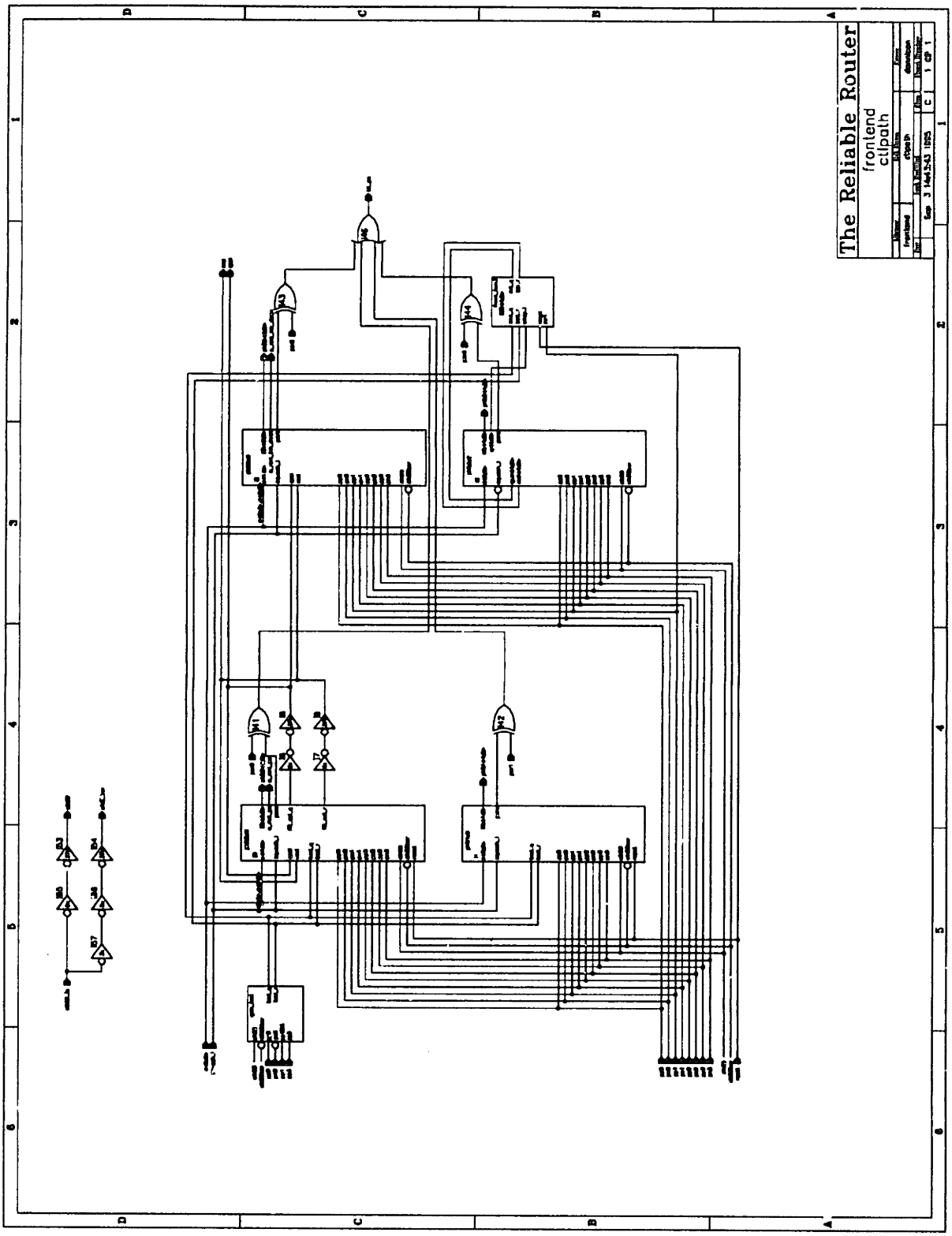


Figure A-94: Library frontend, cell ctlpath

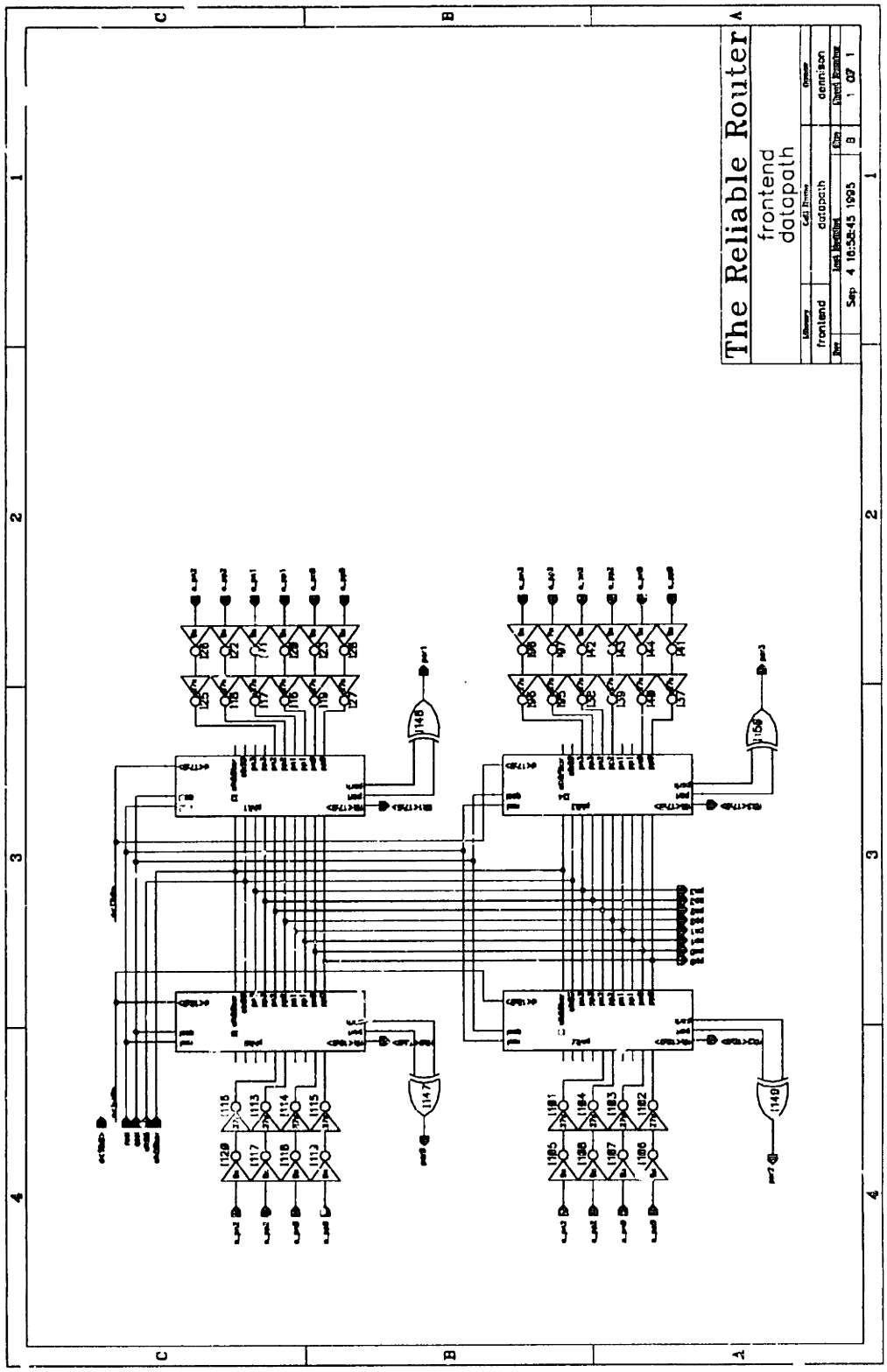


Figure A-95: Library frontend, cell datapath

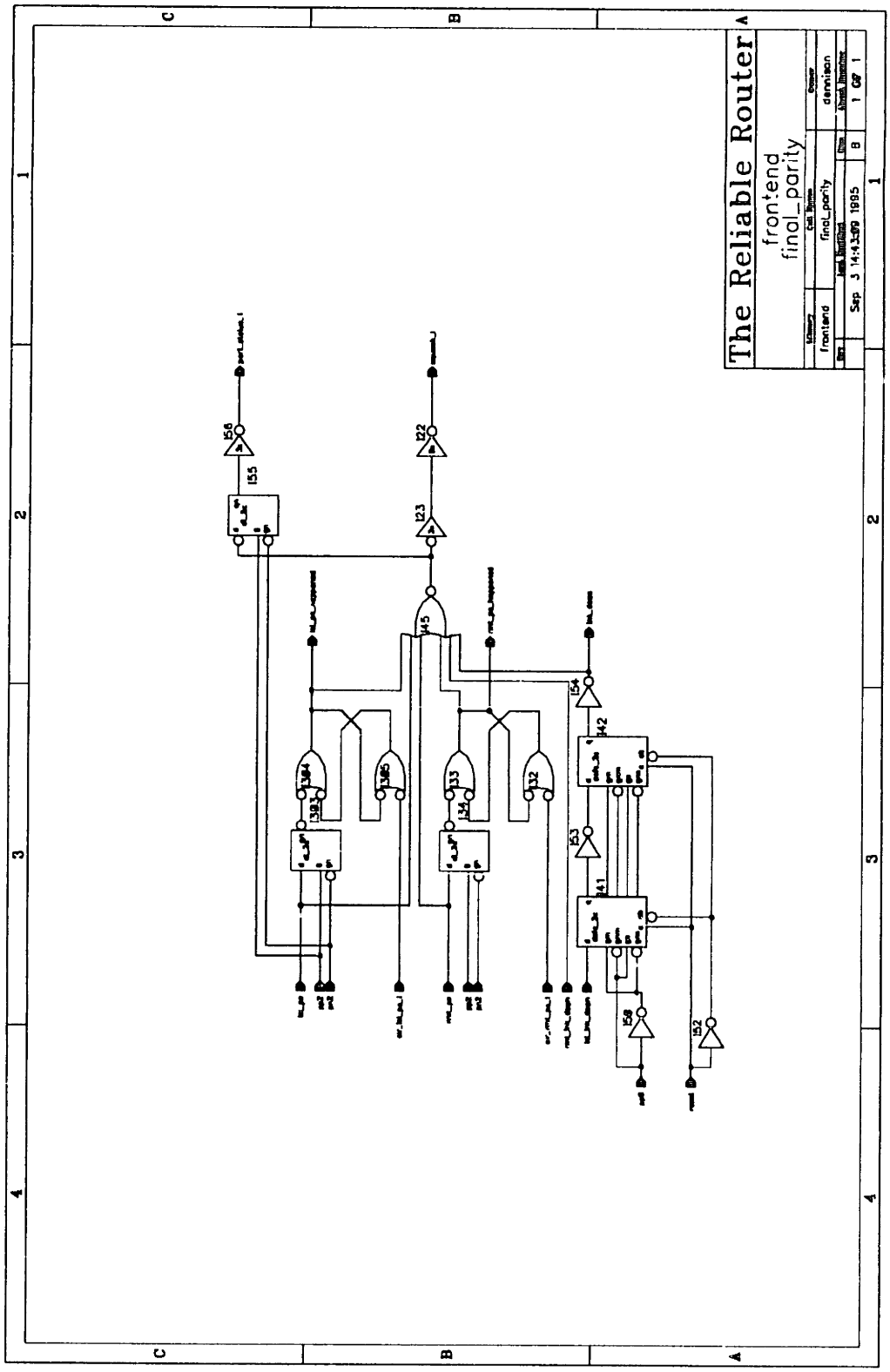


Figure A-96: Library frontend, cell final_parity

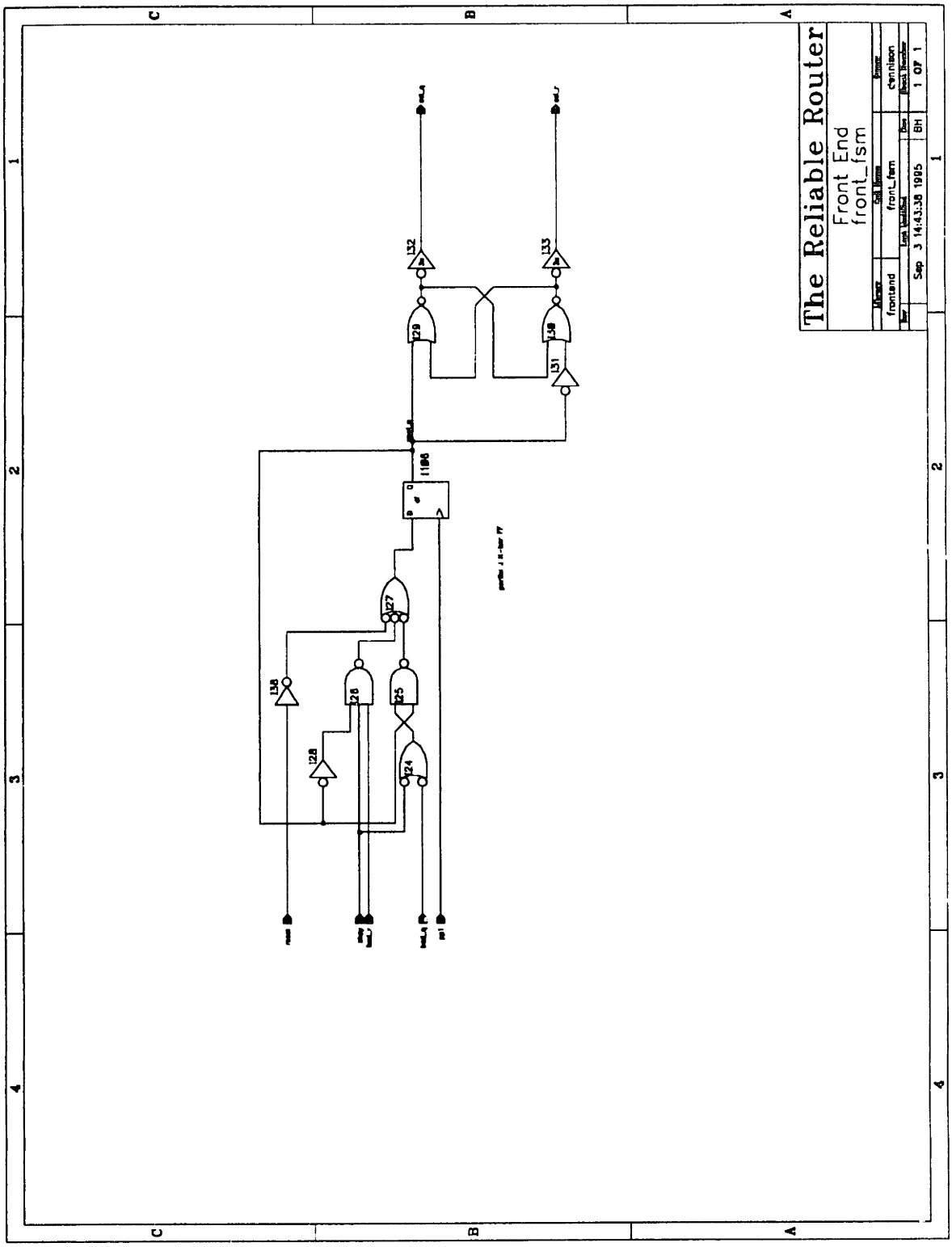


Figure A-97: Library frontend, cell front_fsm

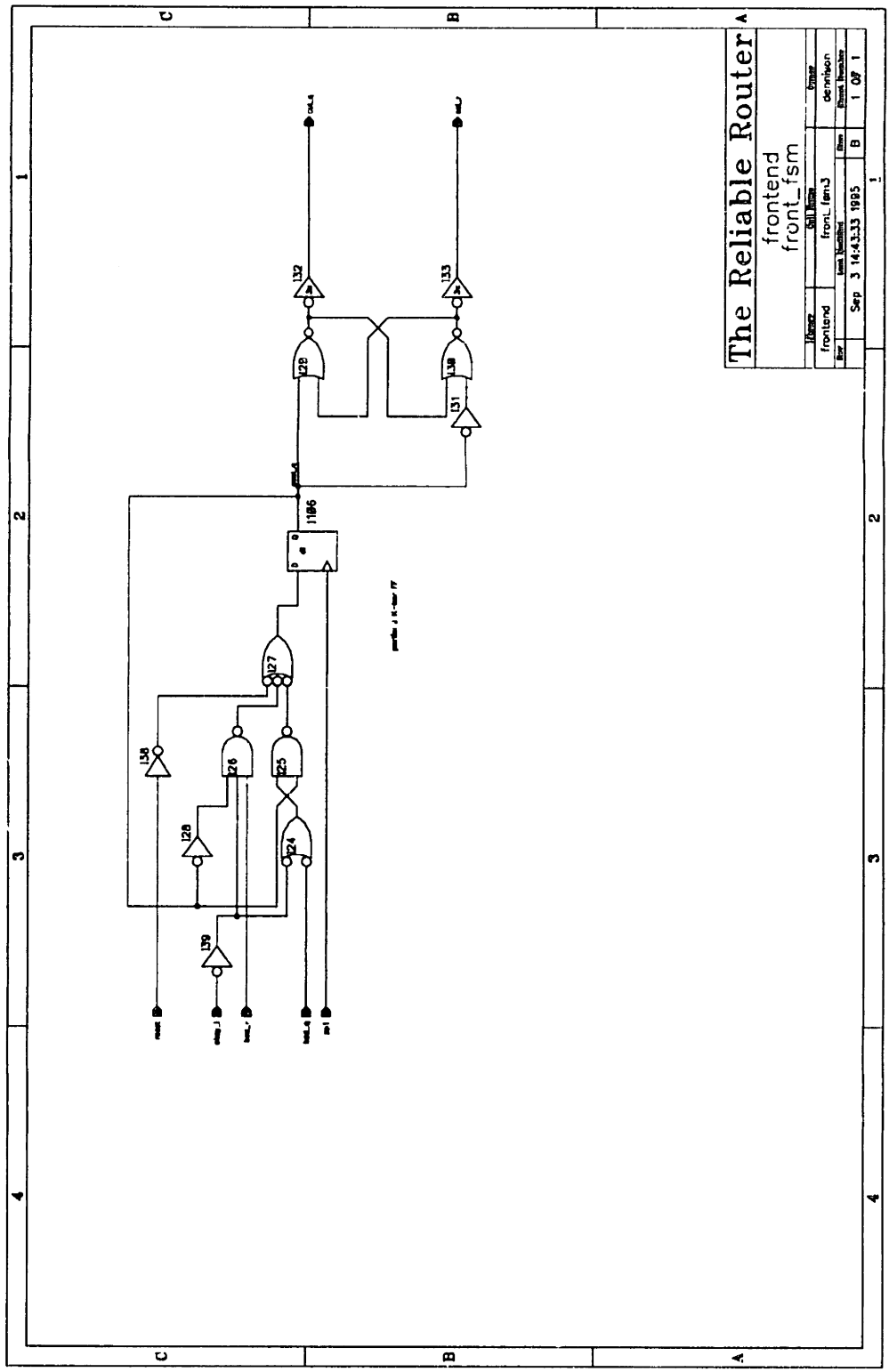


Figure A-98: Library frontend, cell front_fsm3

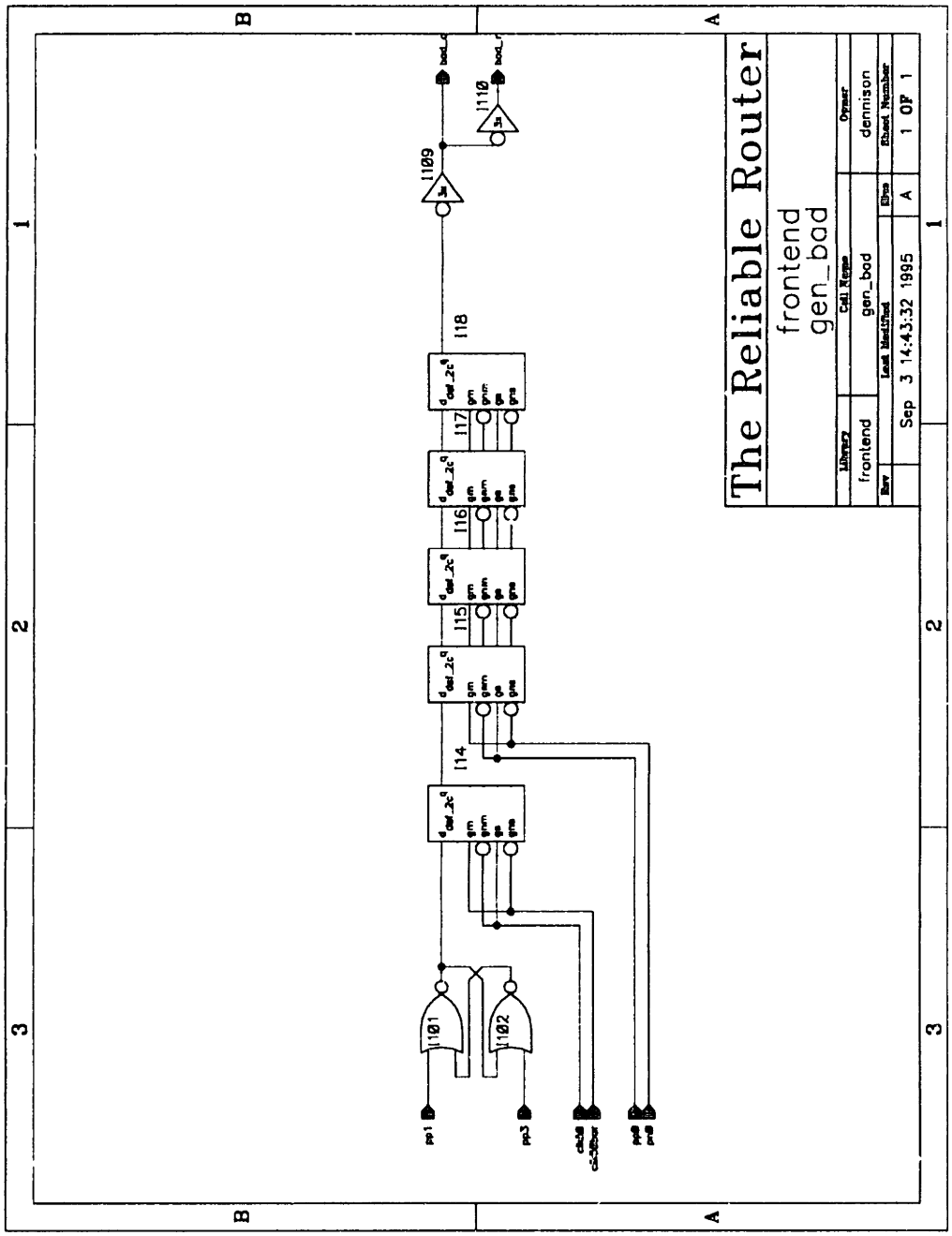


Figure A-99: Library frontend, cell gen_bad

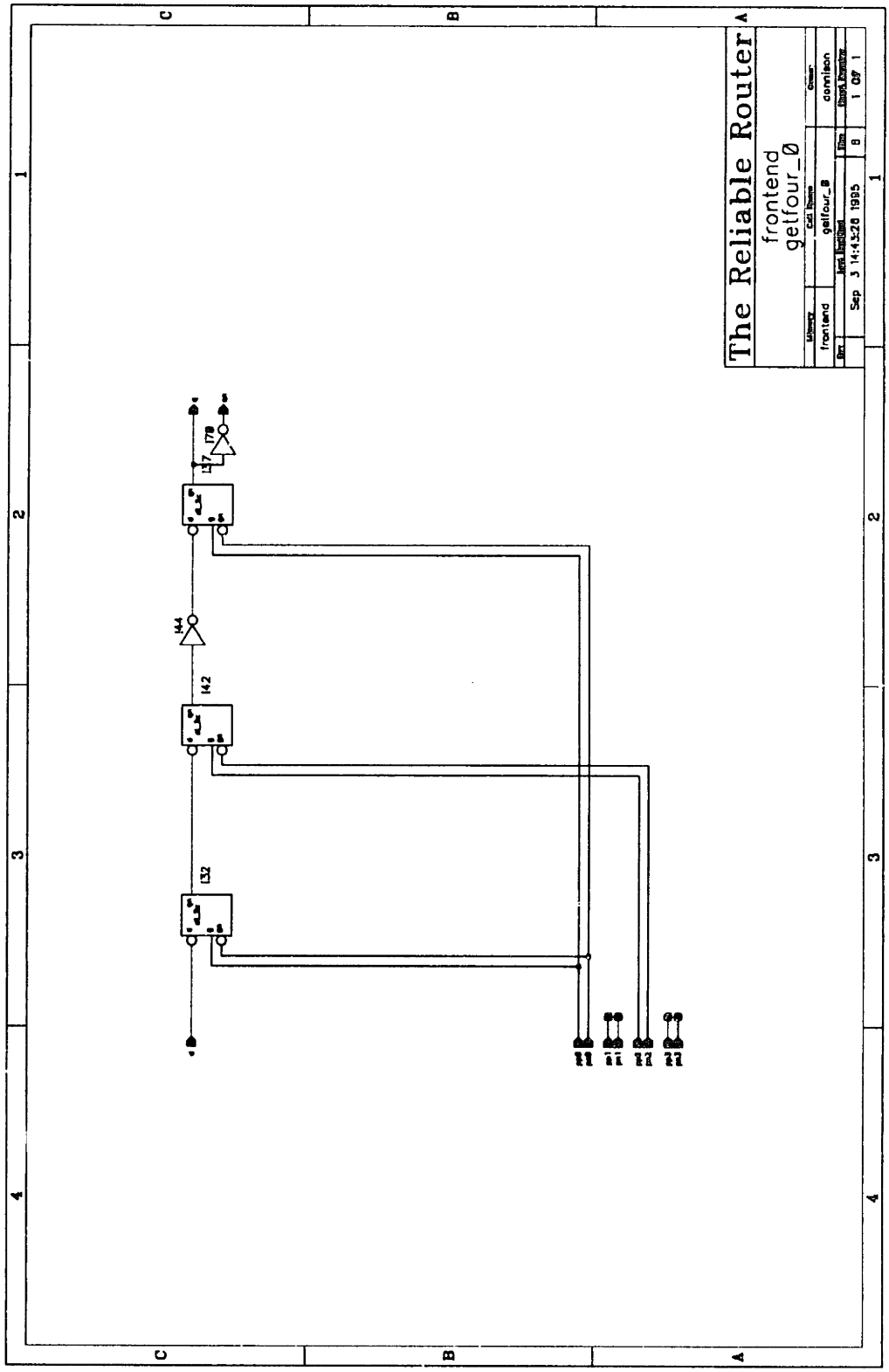


Figure A-100: Library frontend, cell getfour_0

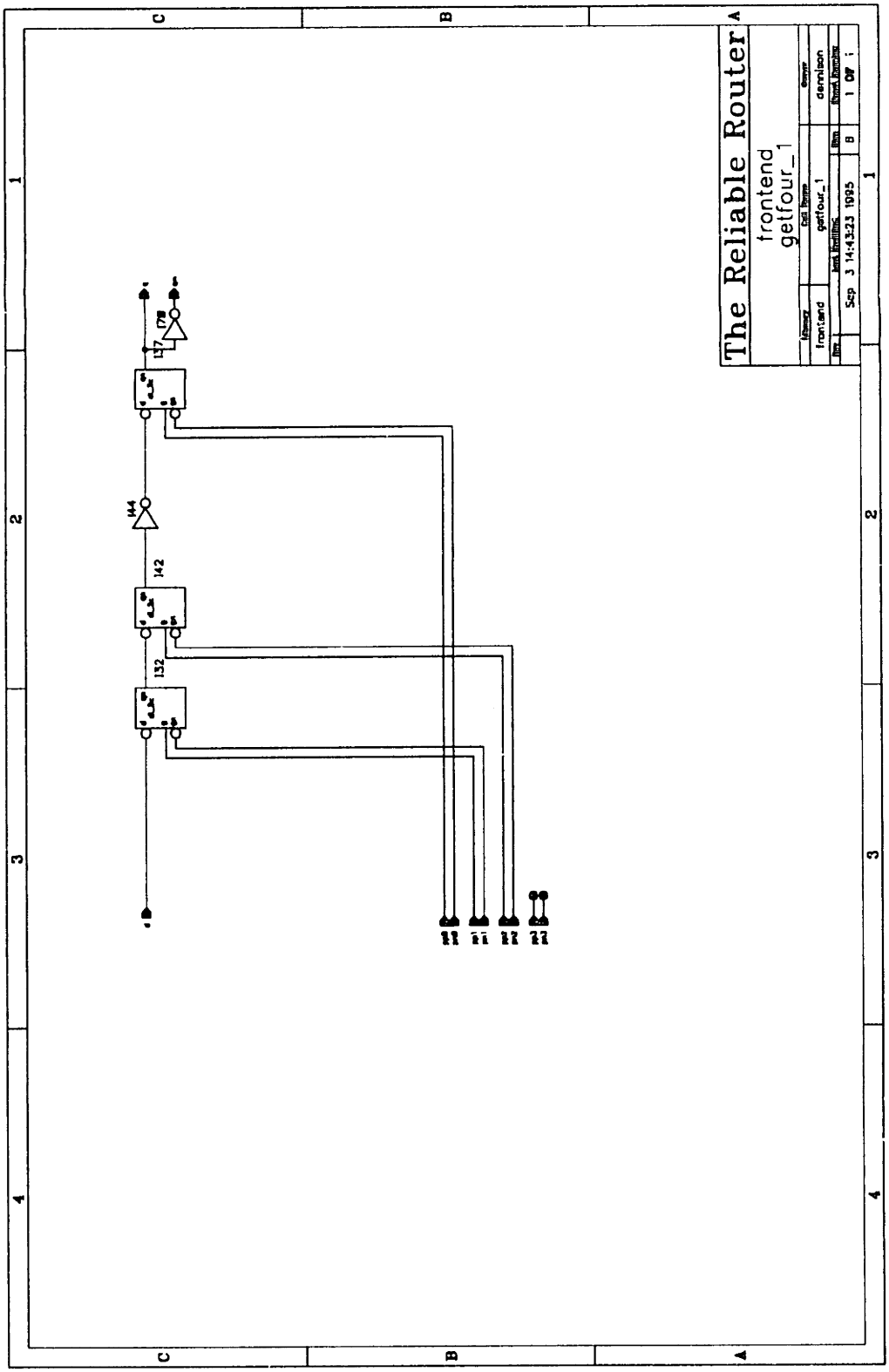


Figure A-101: Library frontend, cell getfour_1

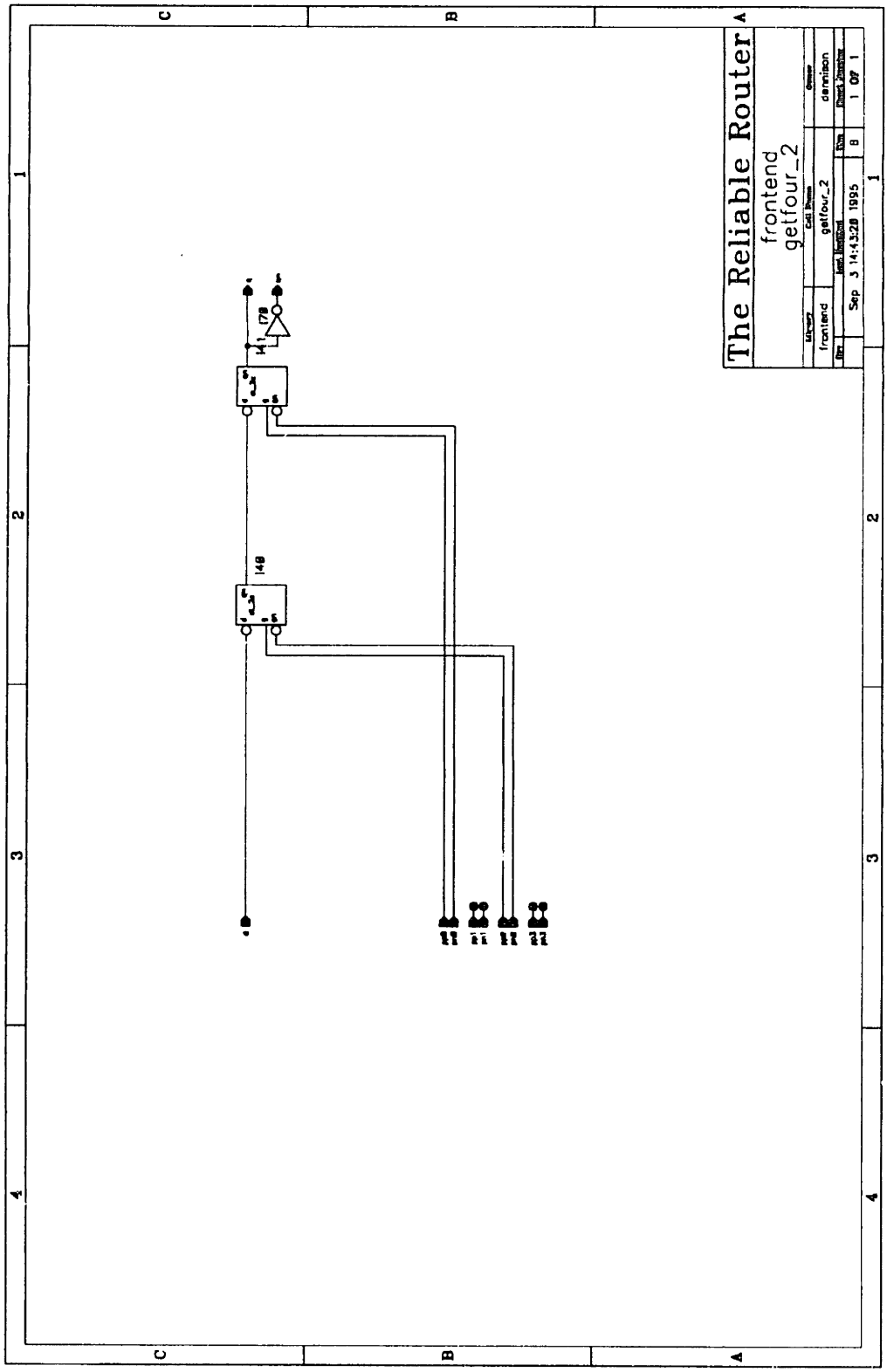


Figure A-102: Library frontend, cell getfour_2

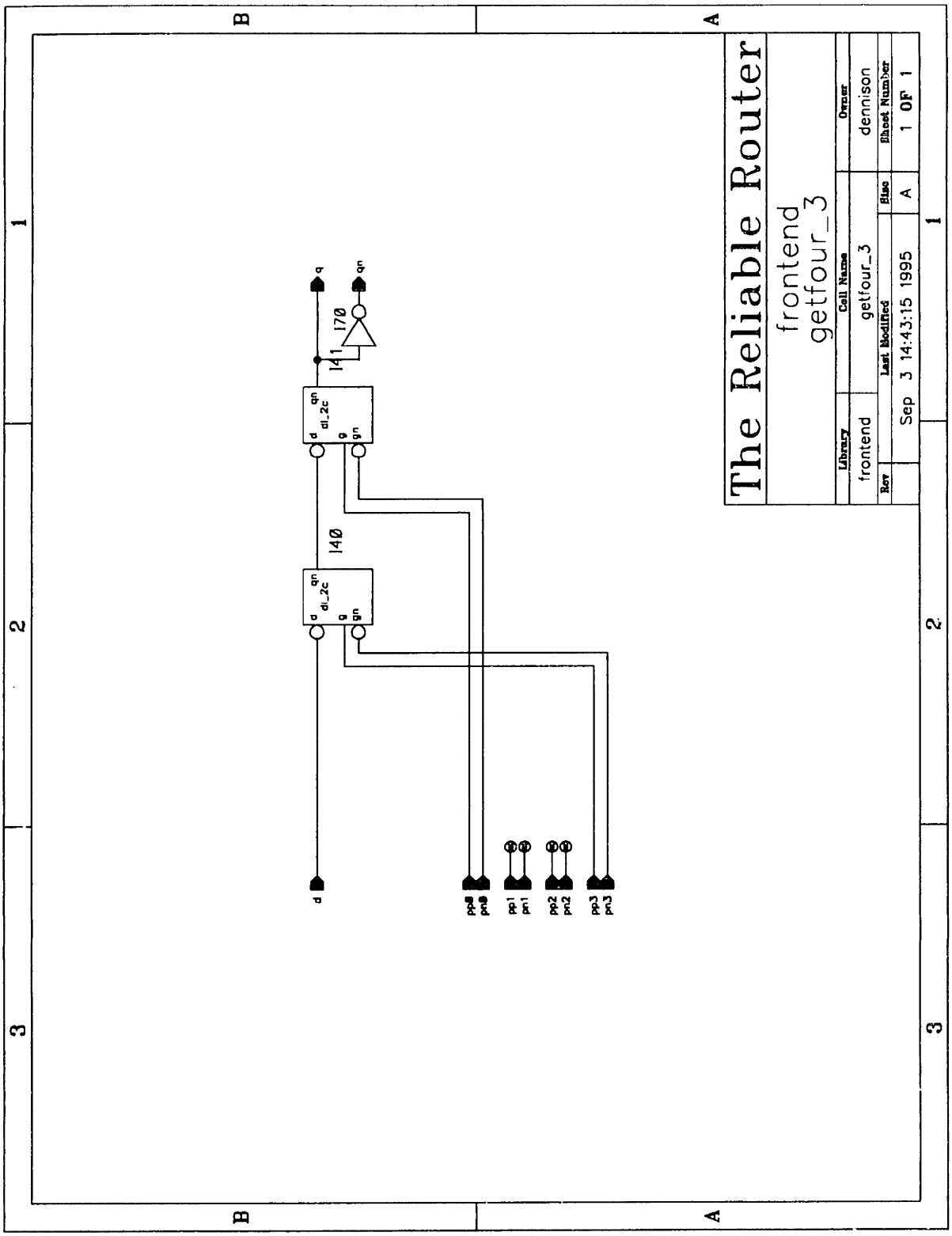


Figure A-103: Library frontend, cell getfour_3

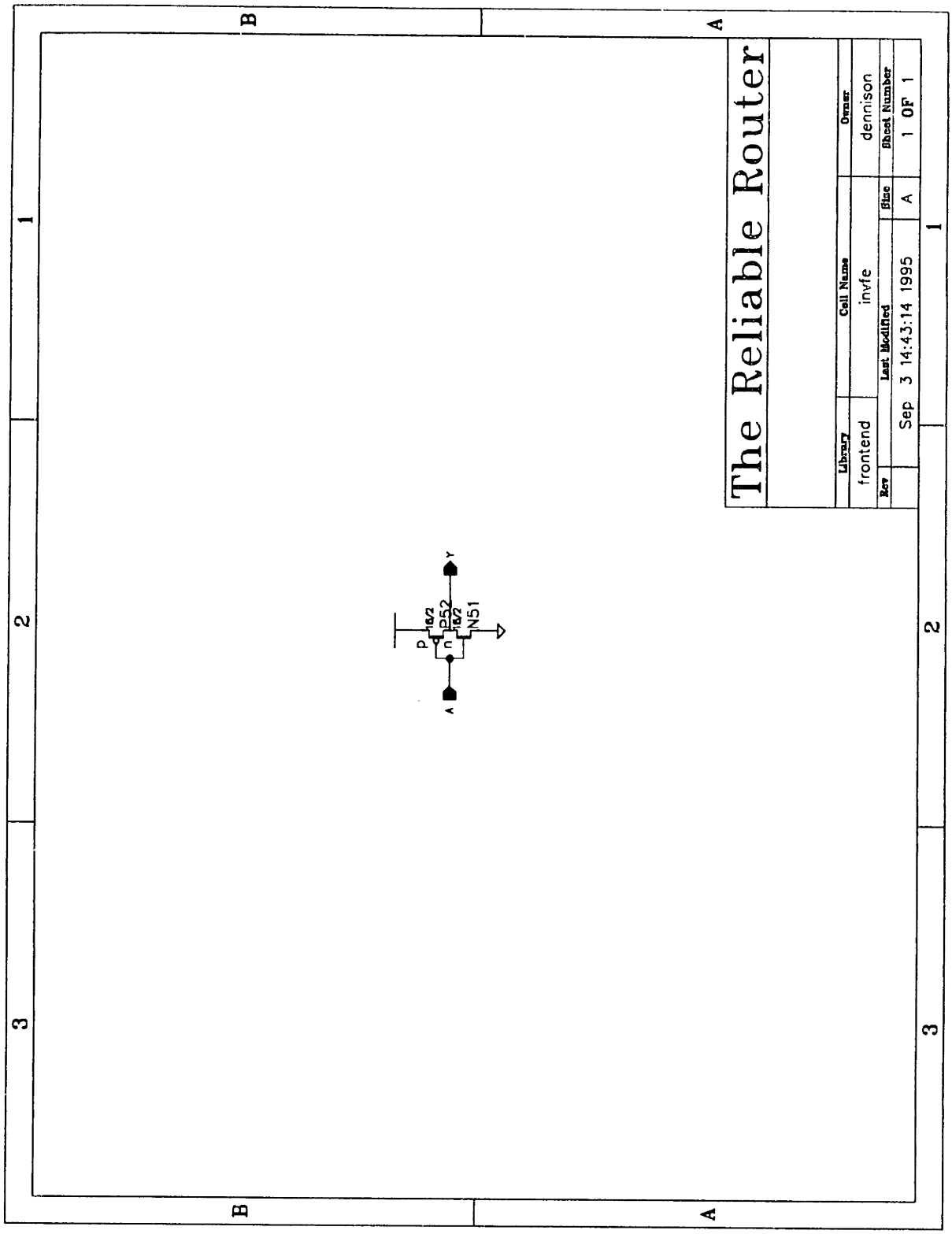


Figure A-104: Library frontend, cell invfe

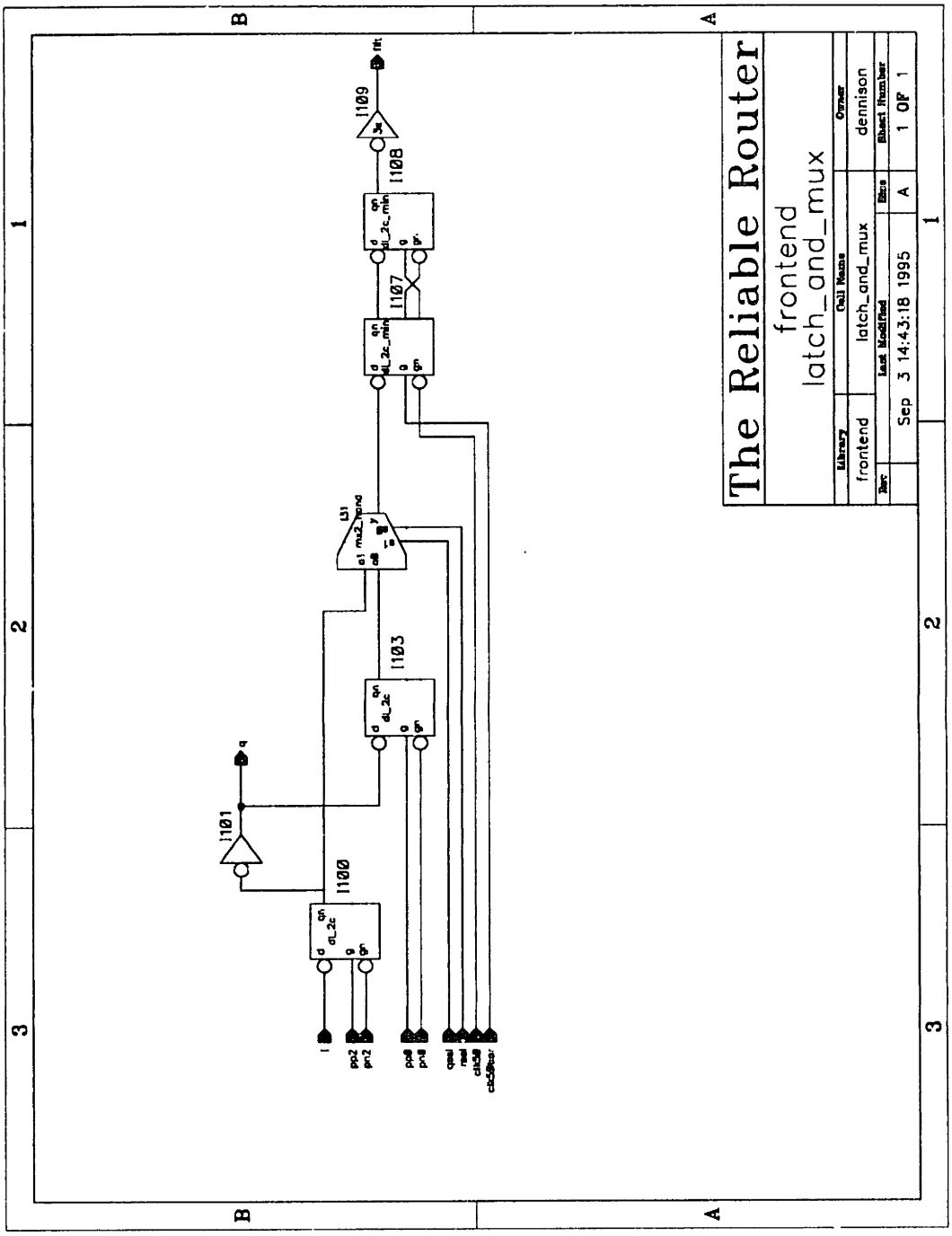


Figure A-105: Library frontend, cell latch_and_mux

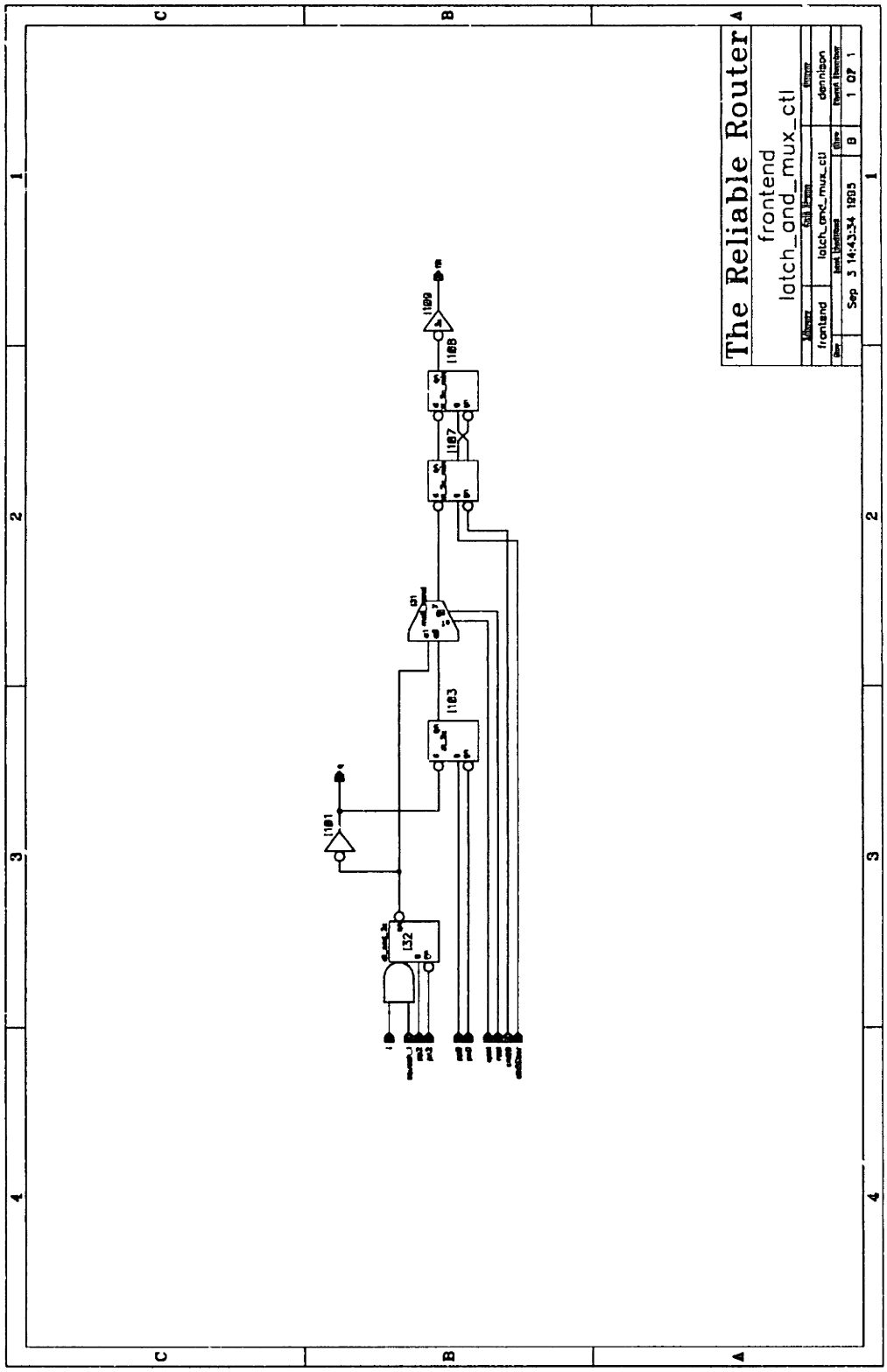


Figure A-106: Library frontend, cell latch_and_mux_ctl

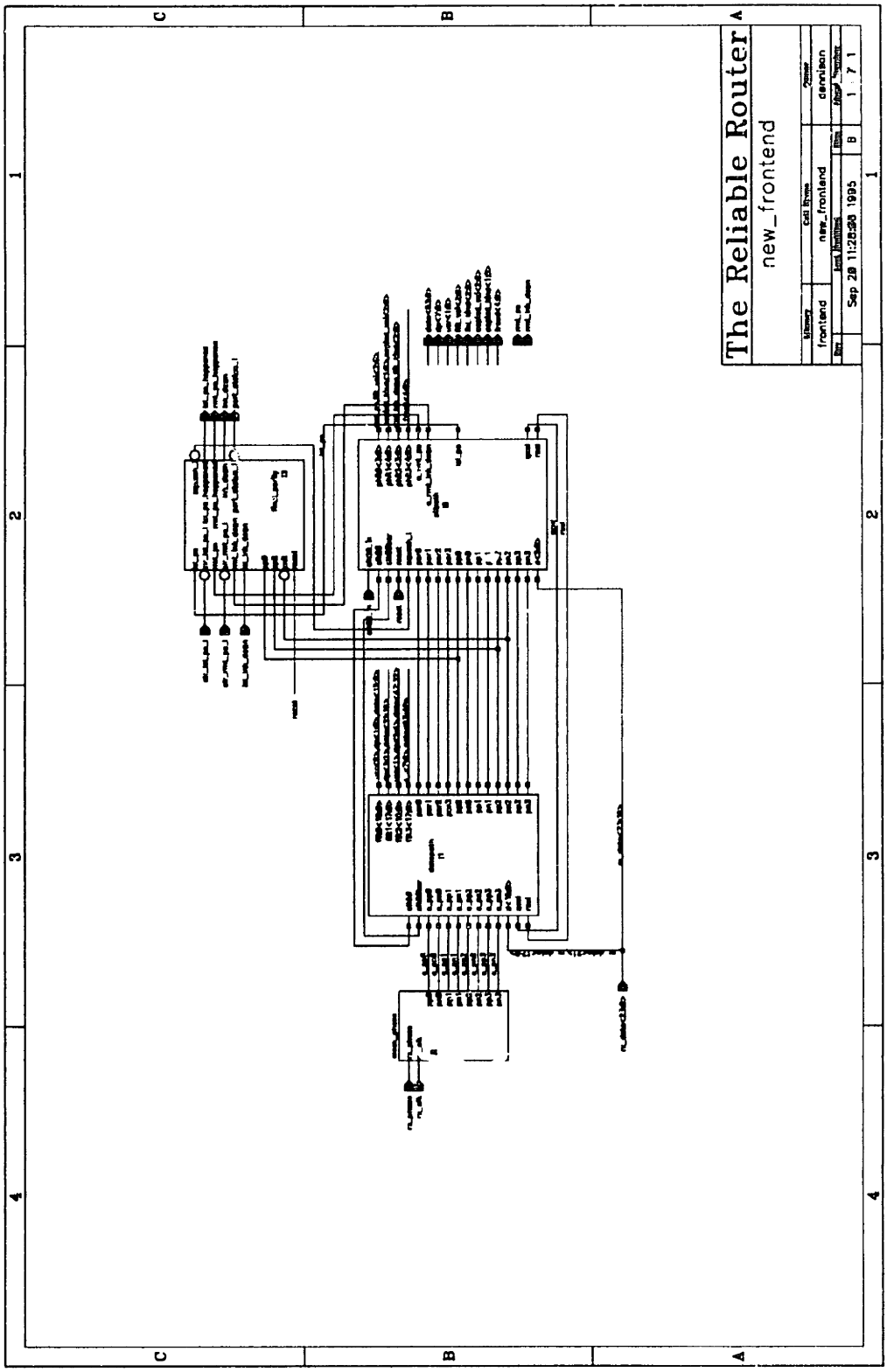


Figure A-107: Library frontend, cell new_frontend

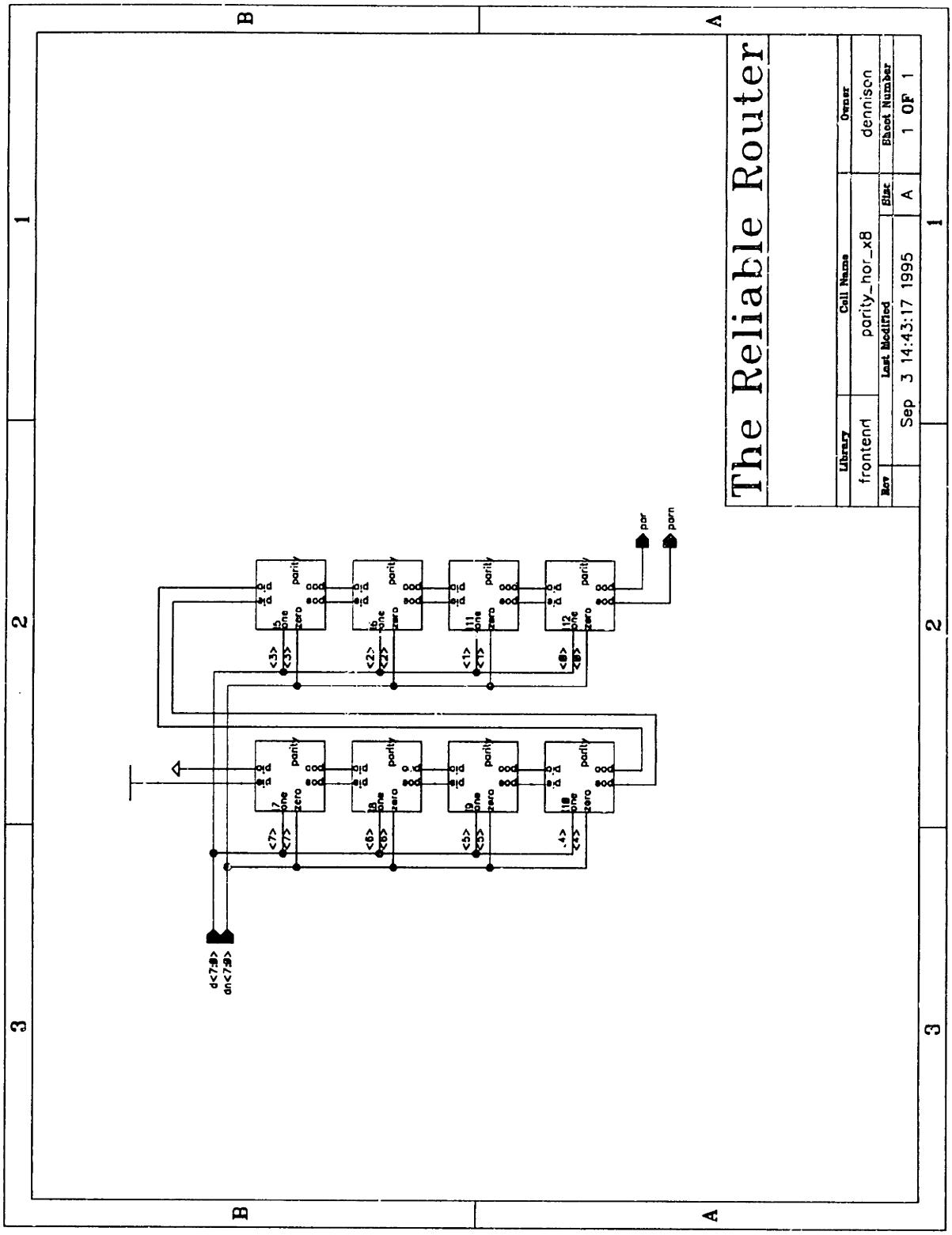


Figure A-108: Library frontend, cell parity_hor_x8

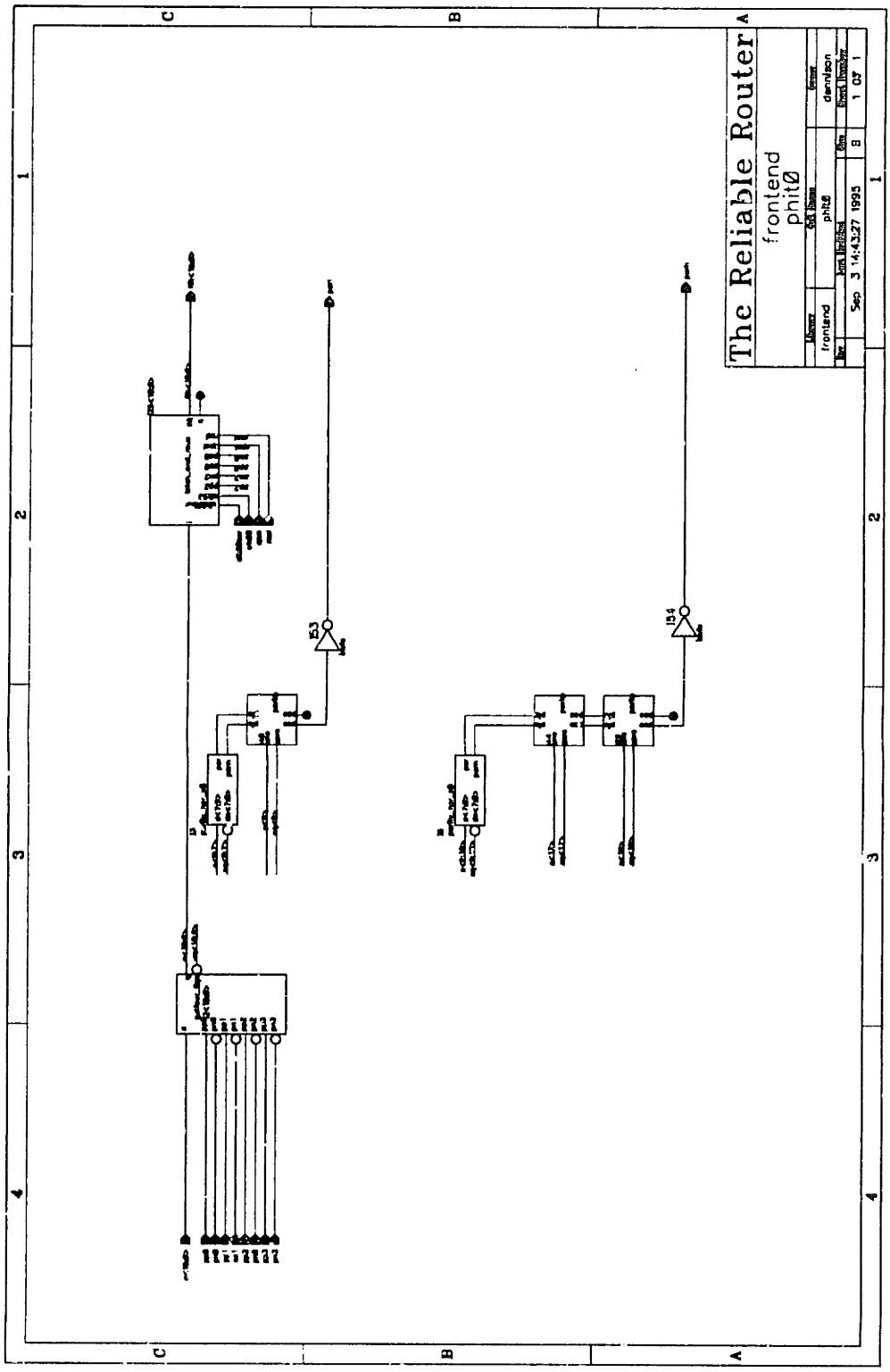


Figure A-109: Library frontend, cell phit0

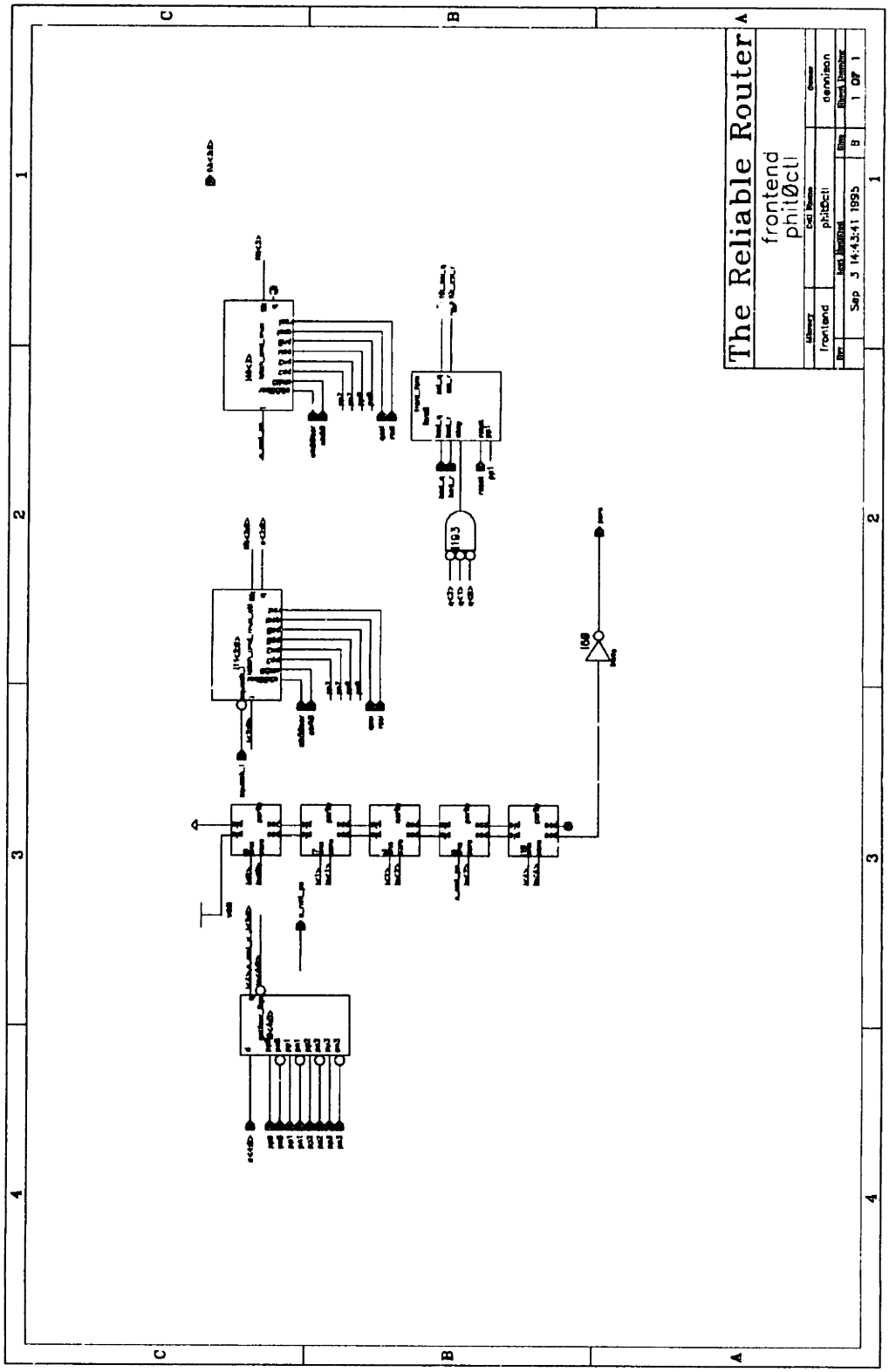


Figure A-110: Library frontend, cell phit0ctl

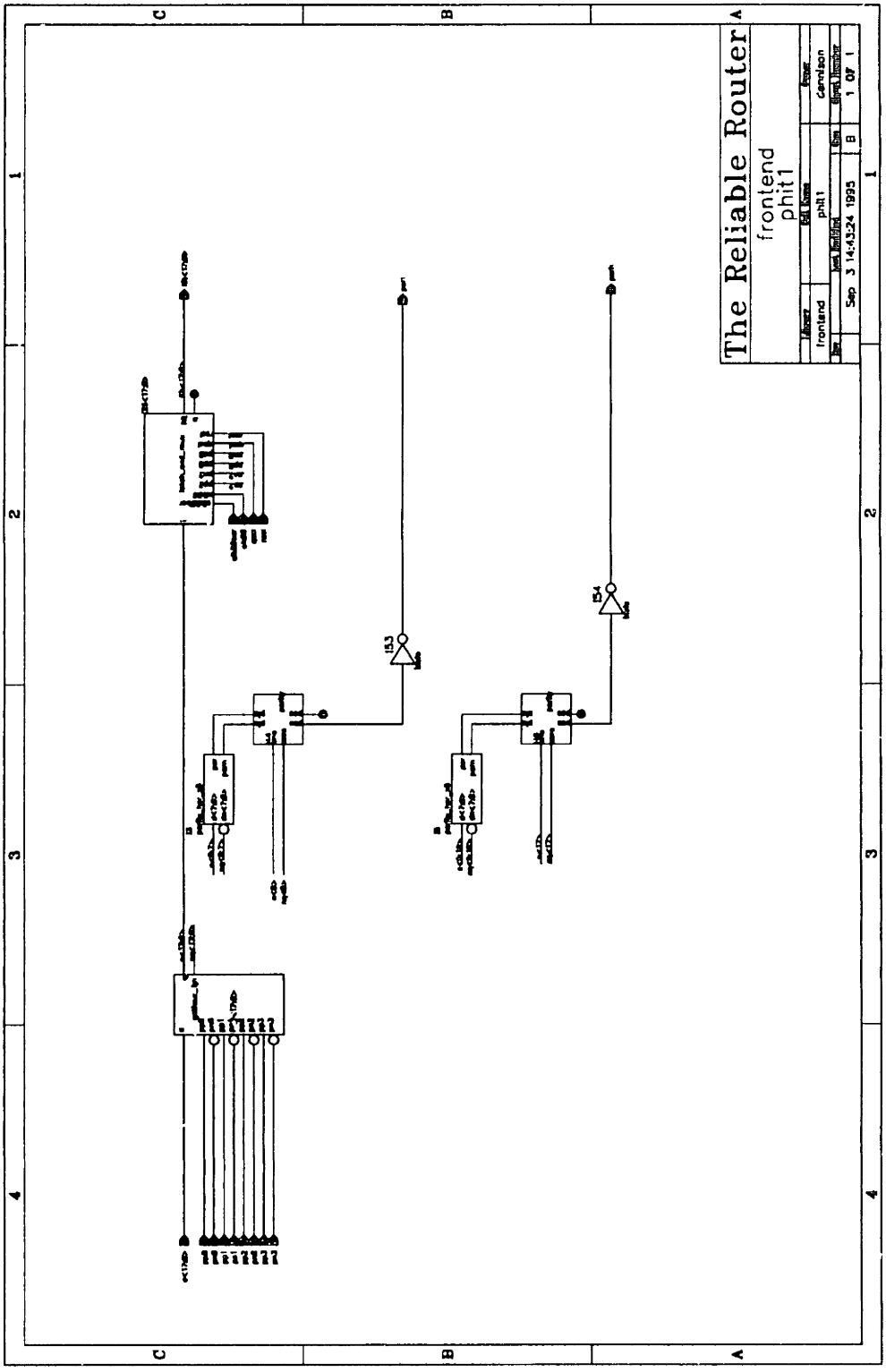


Figure A-111: Library frontend, cell phit1

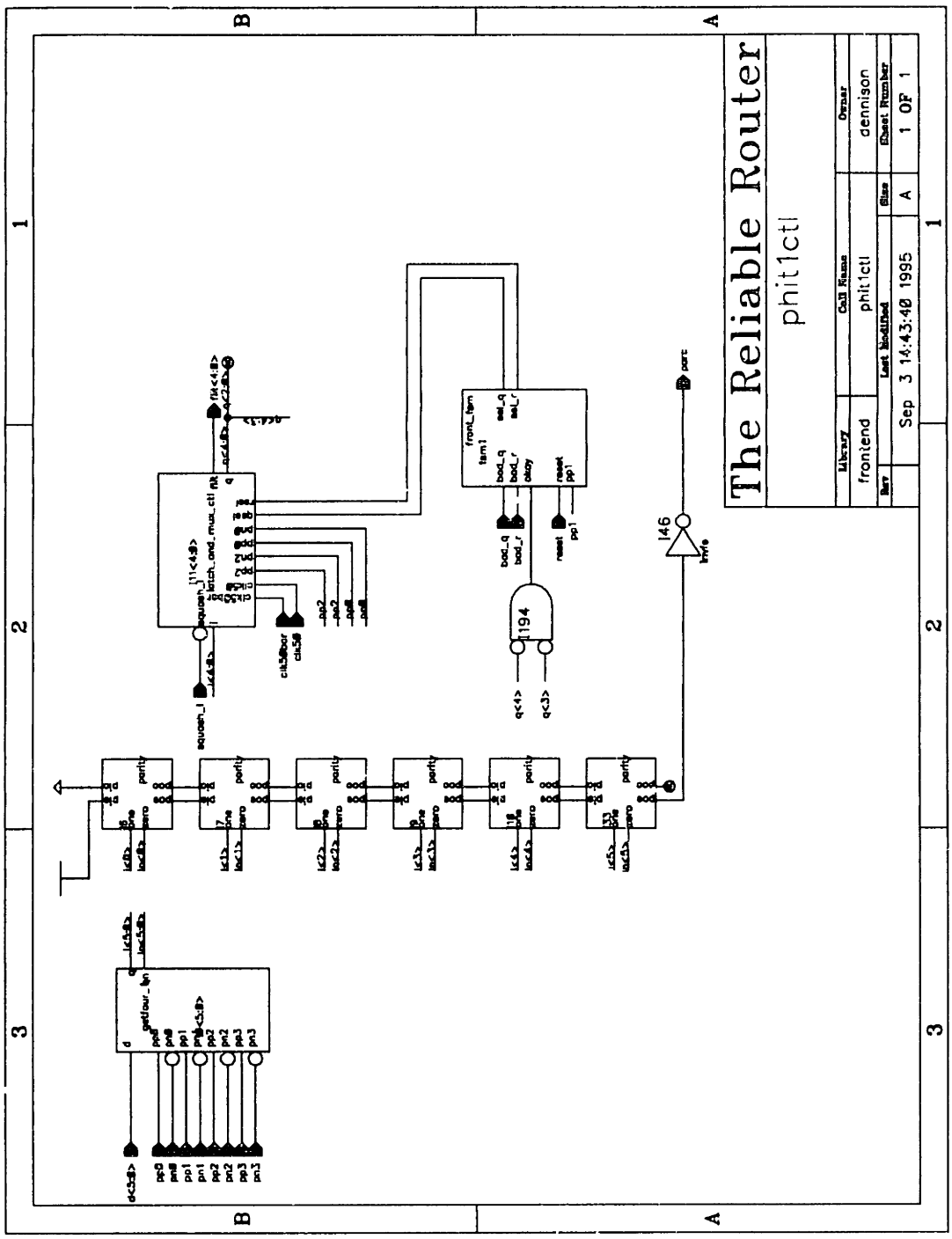


Figure A-112: Library frontend, cell phit1ctl

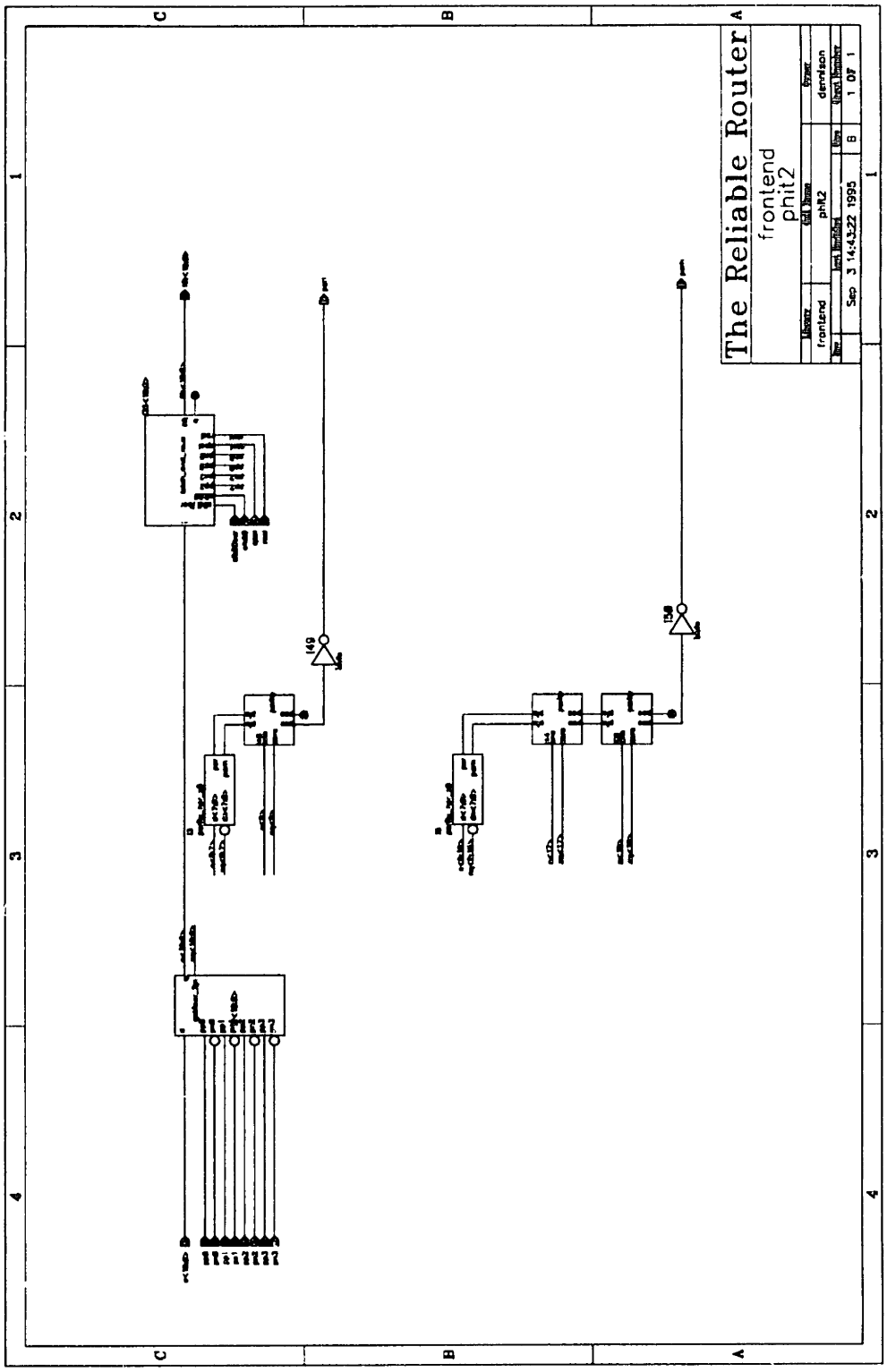


Figure A-113: Library frontend, cell phit2

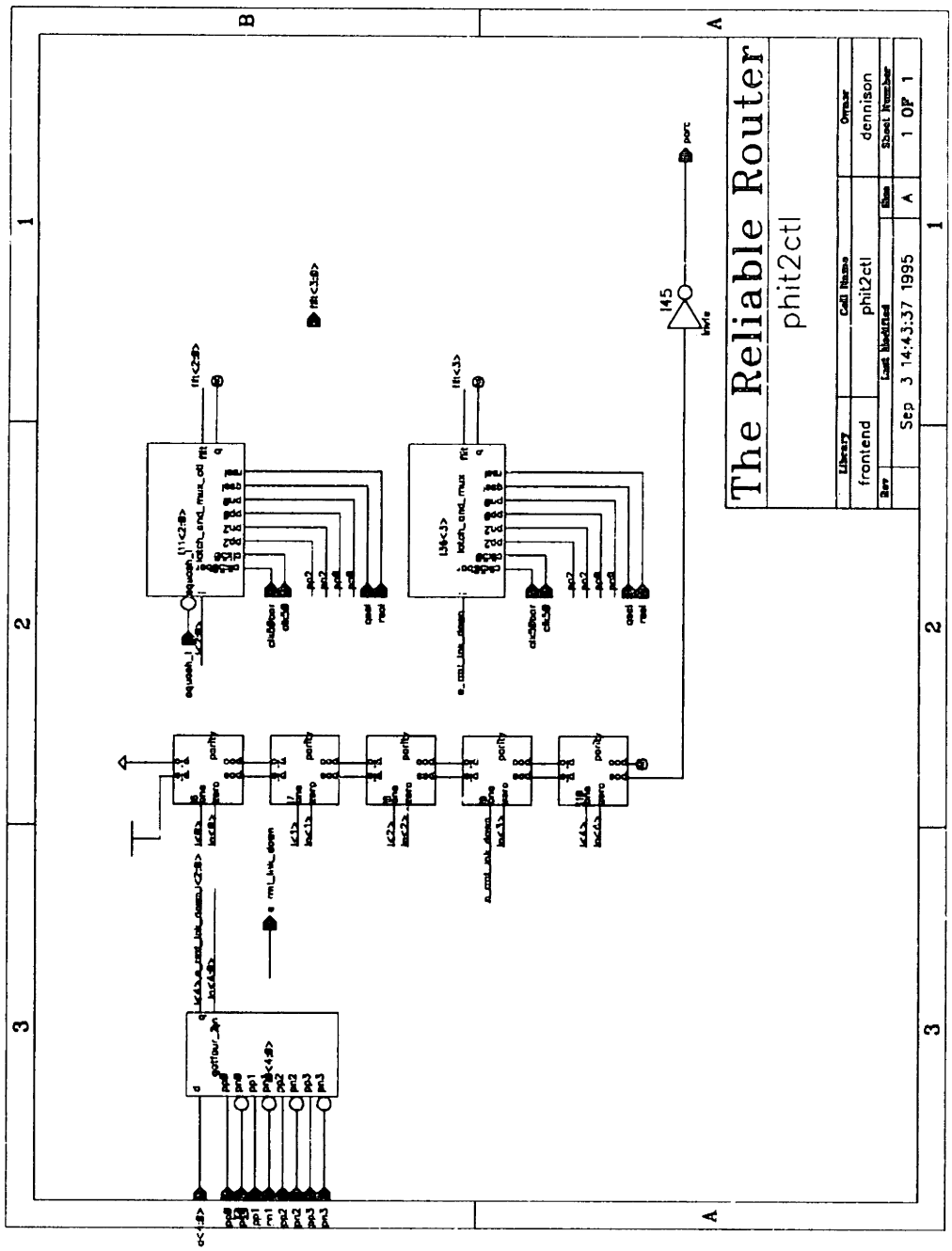


Figure A-114: Library frontend, cell phit2ctl

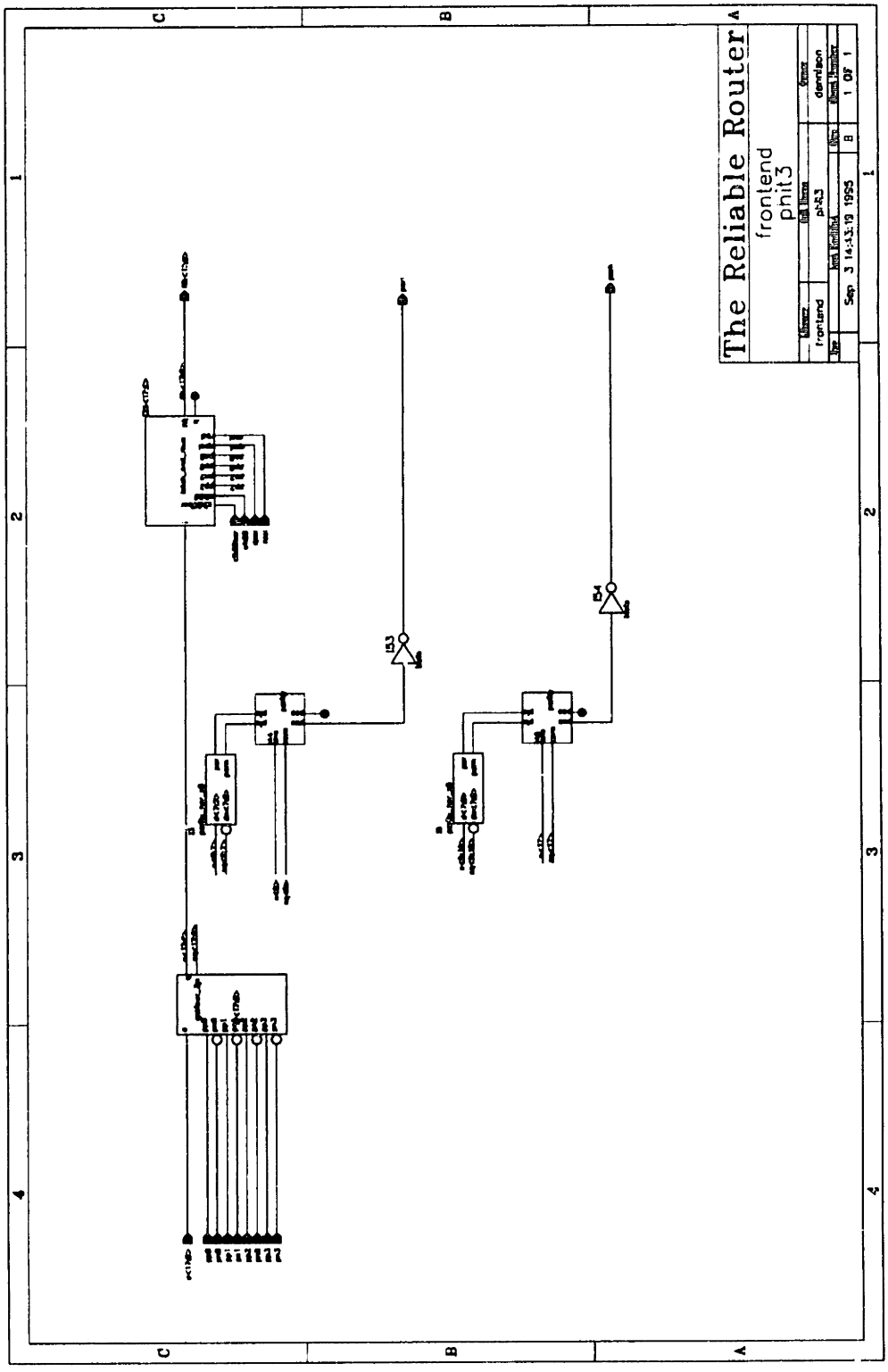


Figure A-115: Library frontend, cell phit3

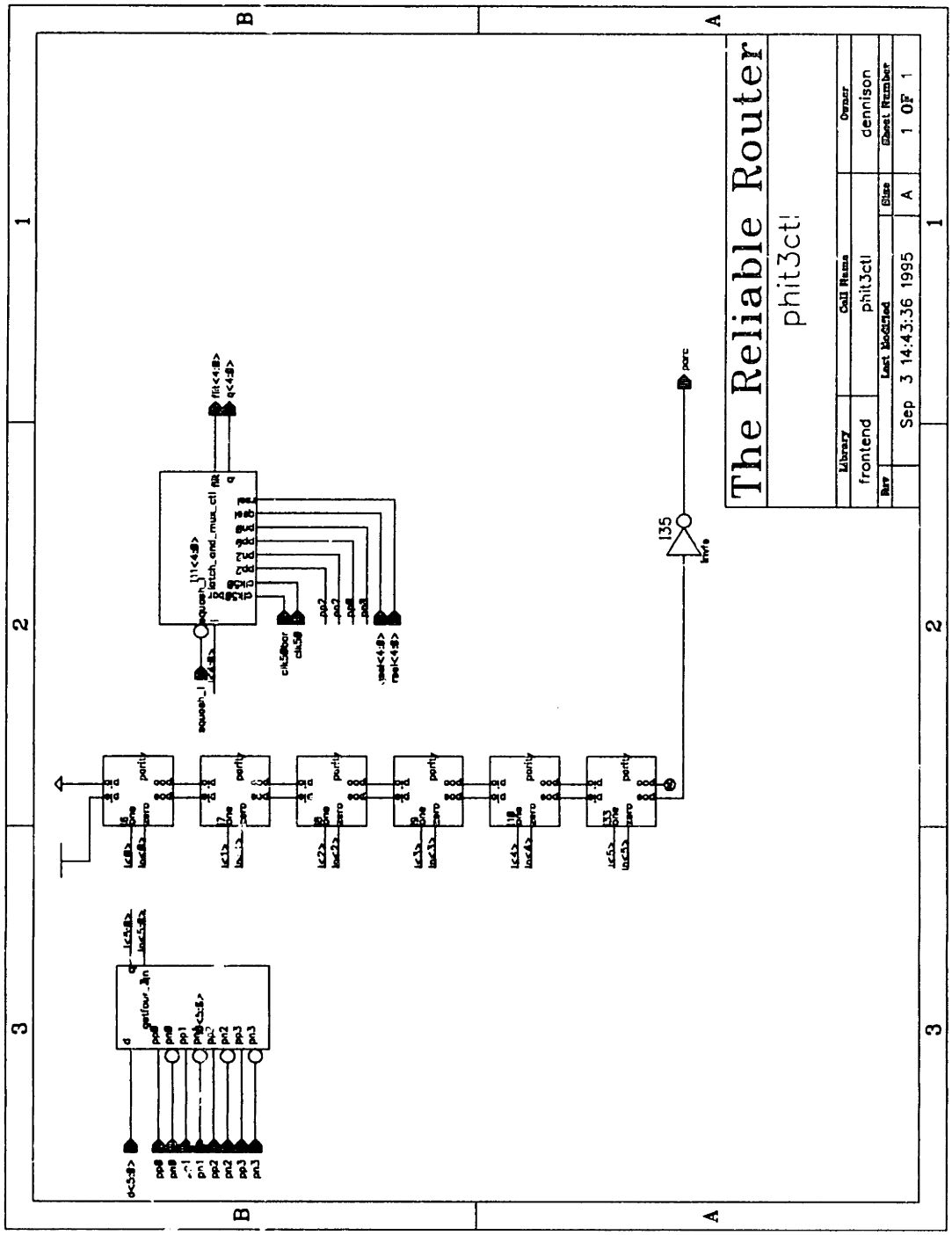


Figure A-116: Library frontend, cell phit3ctl

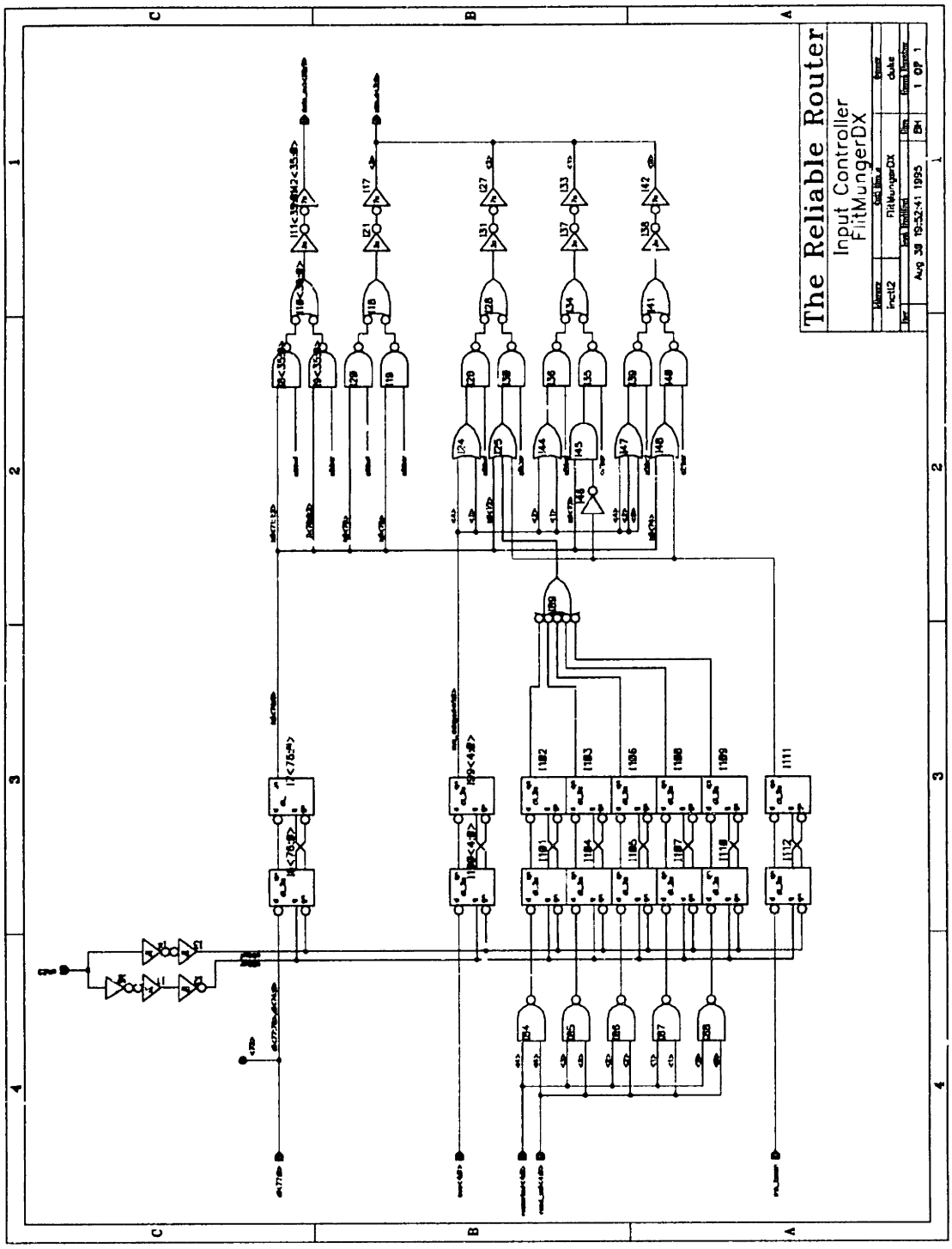


Figure A-117: Library inctl2, cell FlitMungerDX

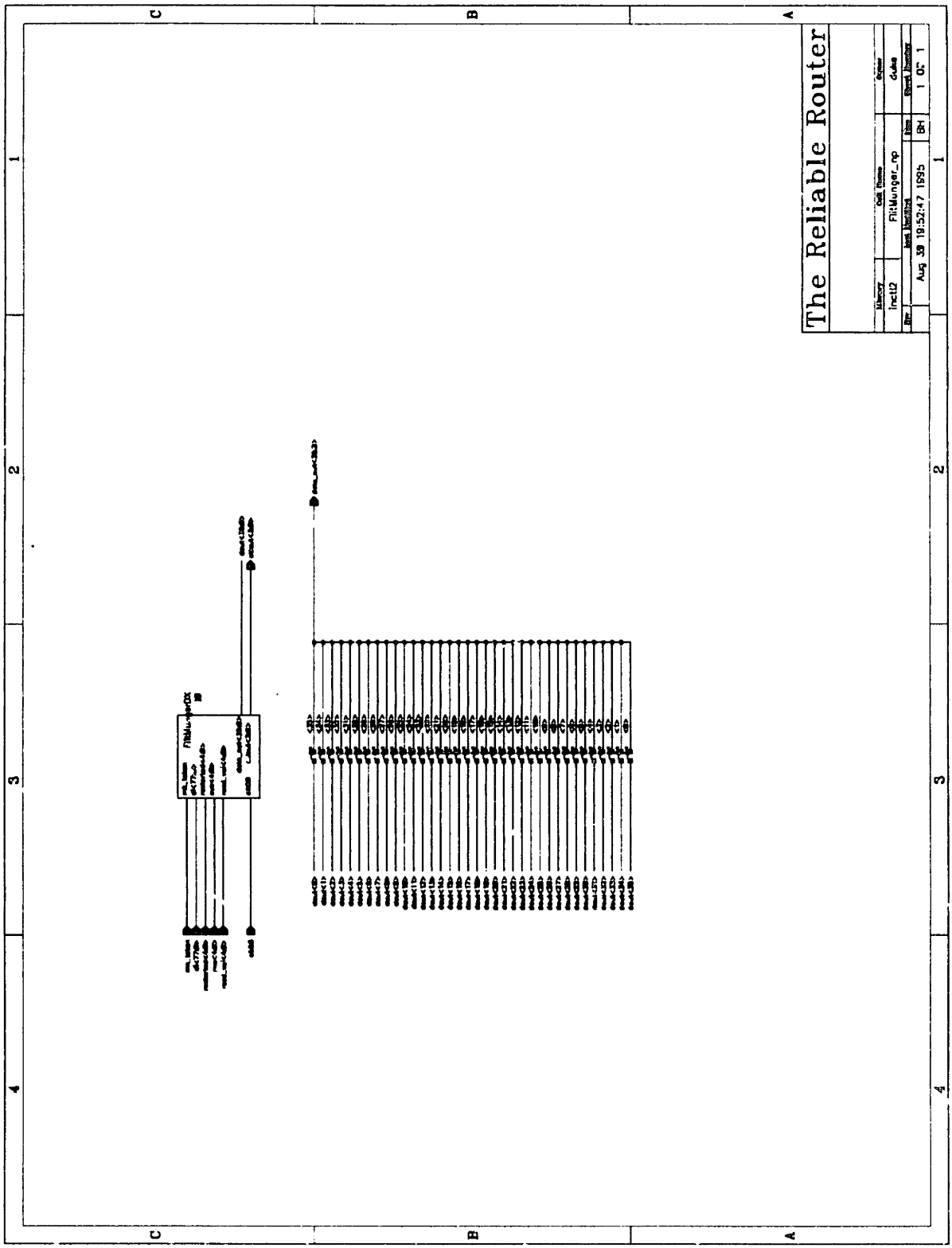


Figure A-118: Library inctl2, cell FlitMunger_np

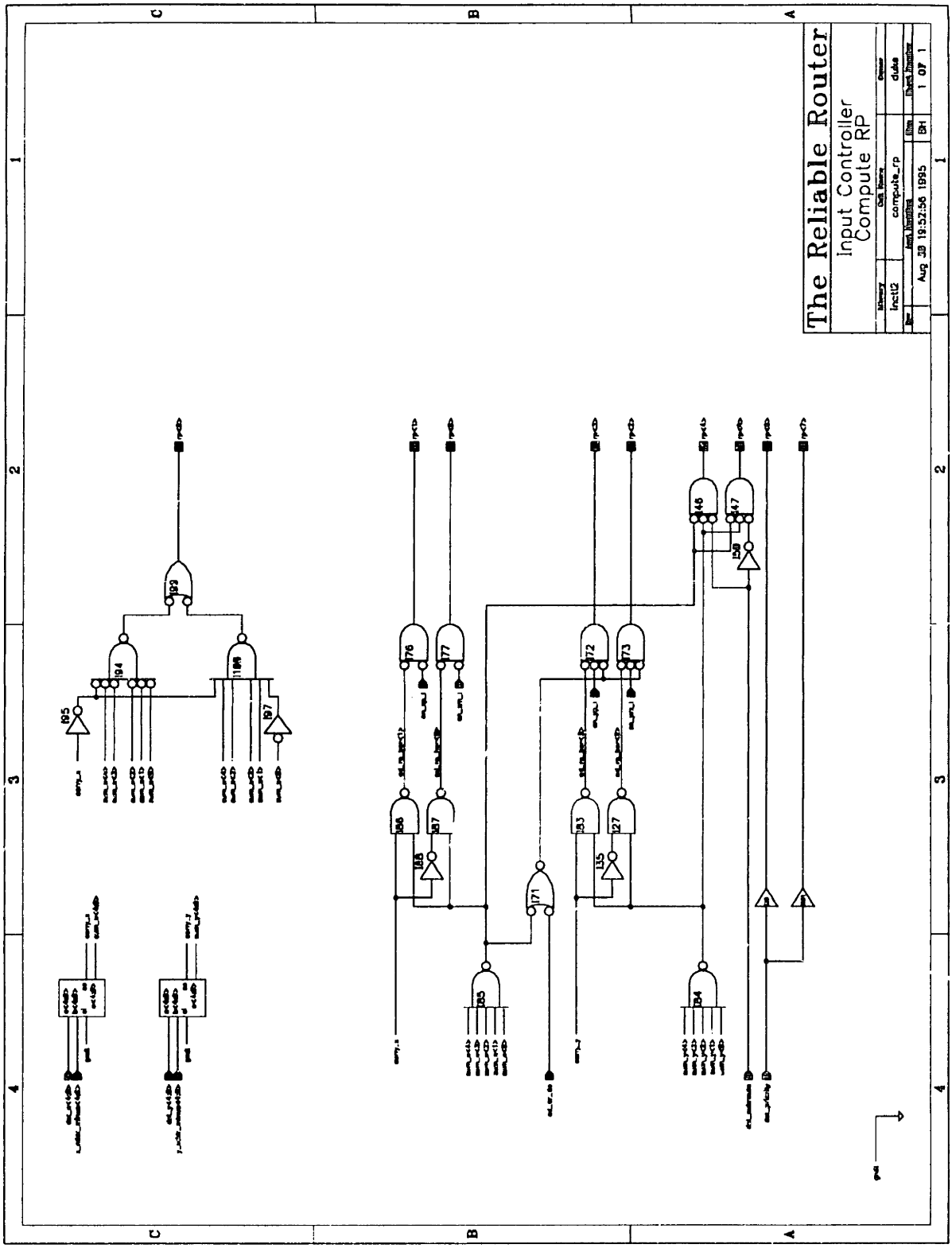


Figure A-119: Library inctl2, cell compute_rp

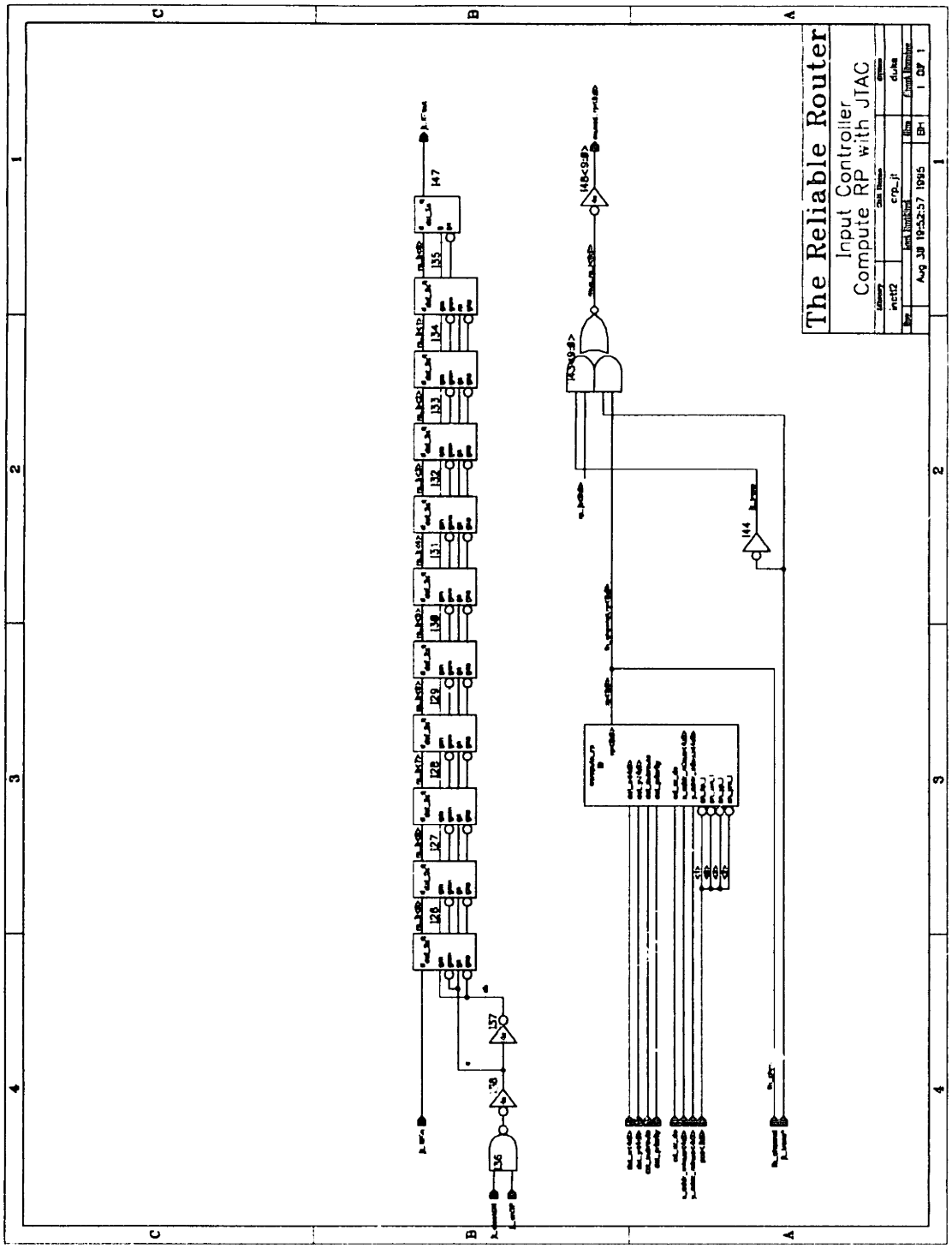


Figure A-120: Library inctl2, cell crp_jt

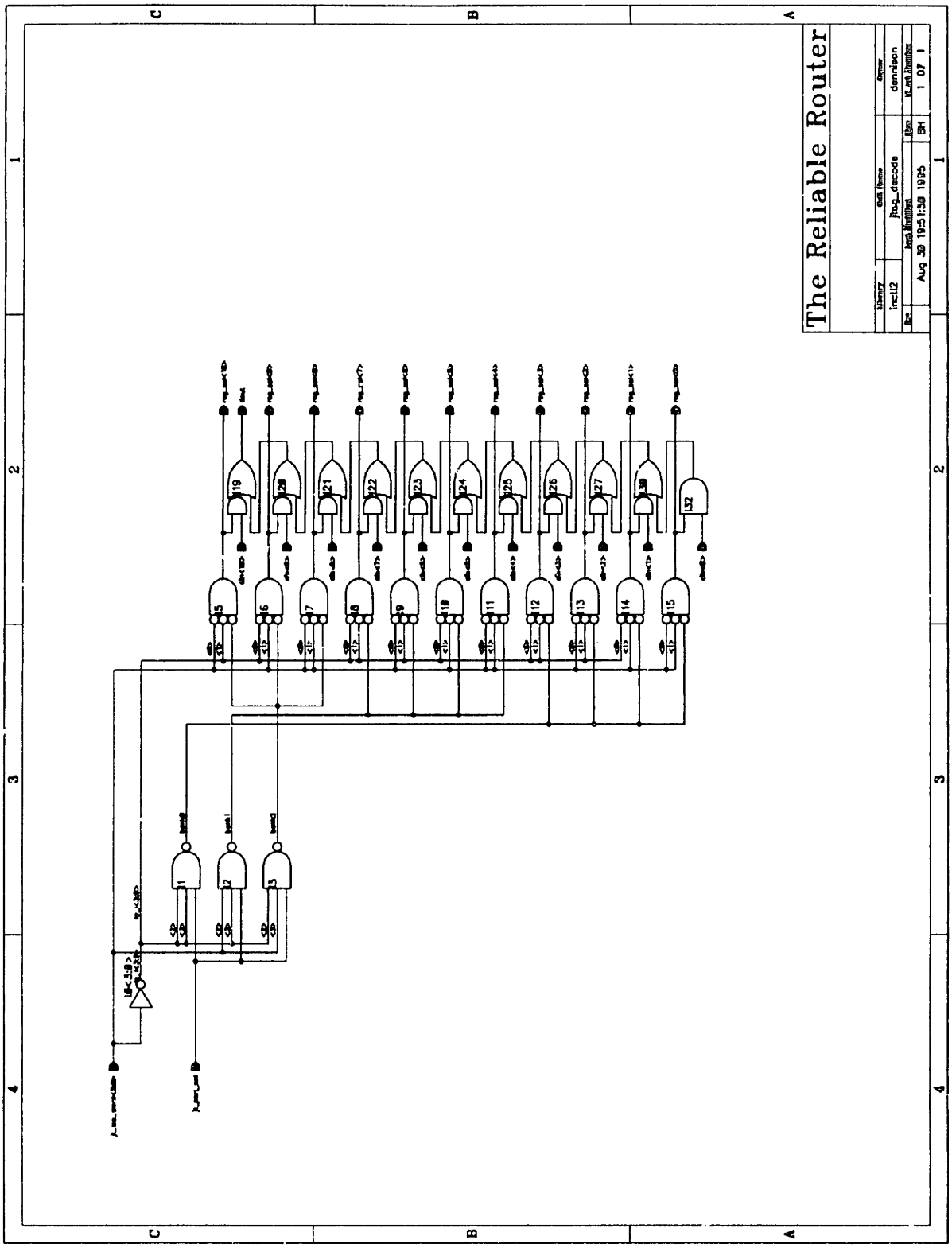


Figure A-121: Library inctl2, cell jtag_decode

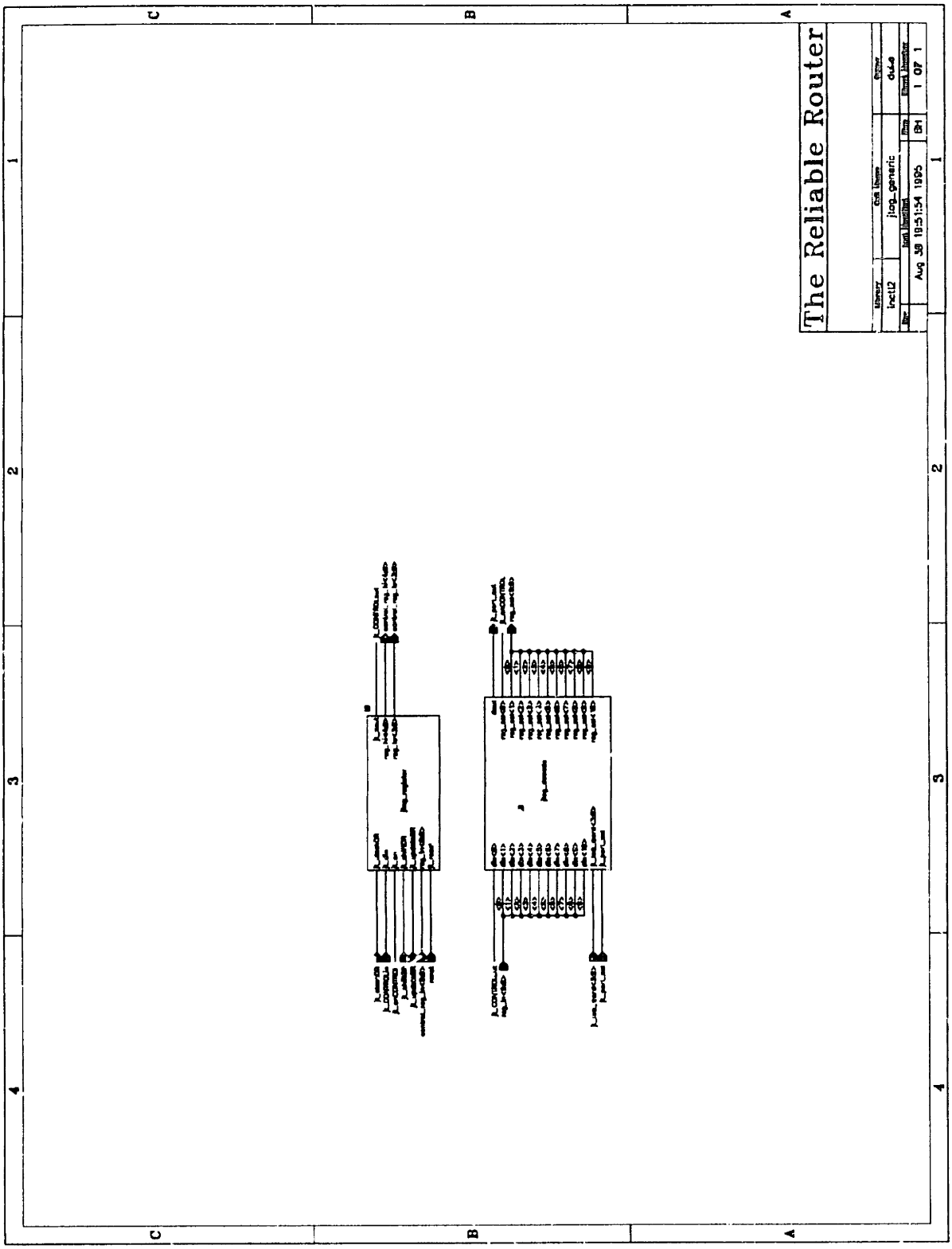


Figure A-122: Library inctl2, cell jtag_generic

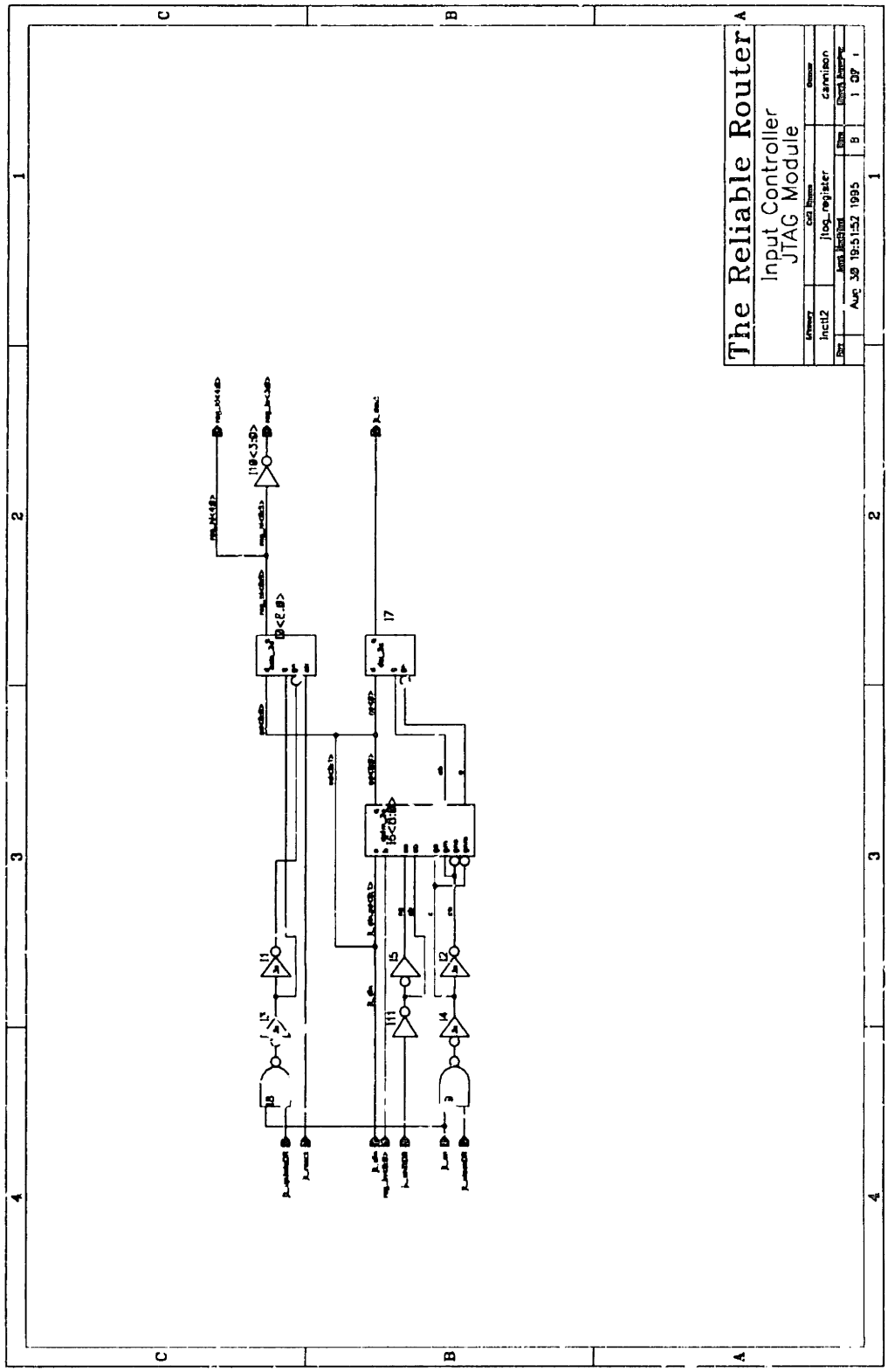


Figure A-123: Library inctl2, cell jtag_register

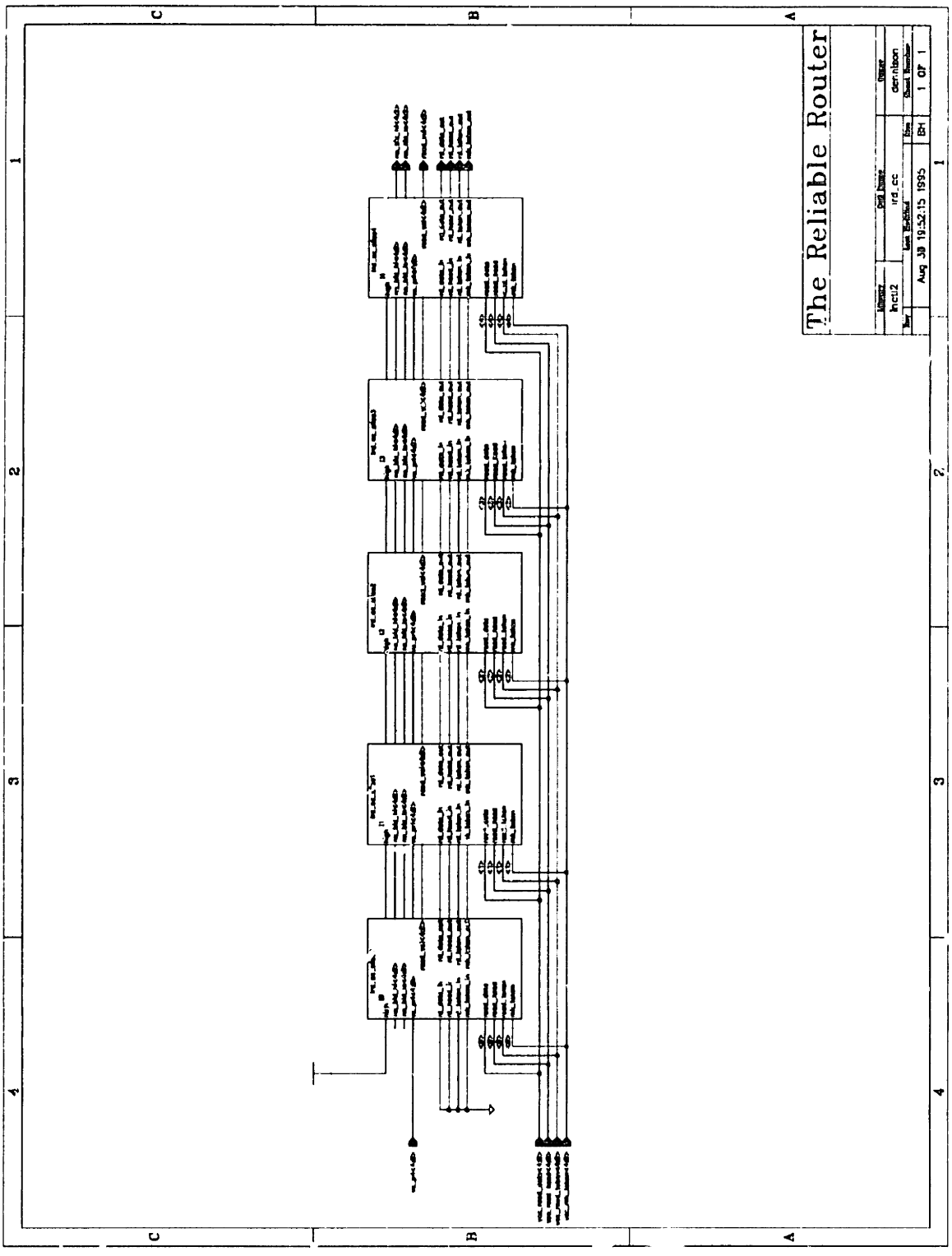


Figure A-124: Library inctl2, cell lrd_cc

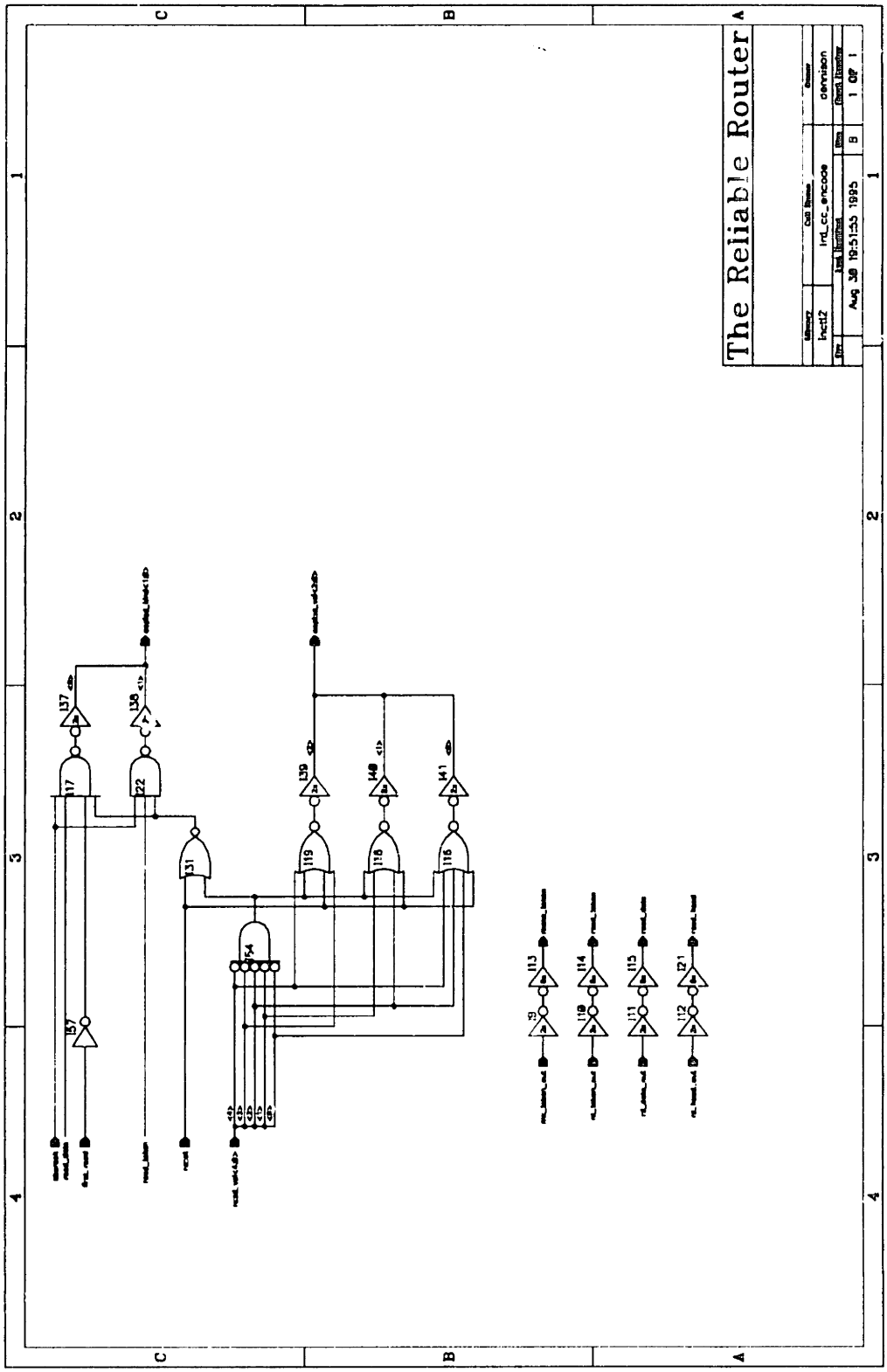


Figure A-125: Library inctl2, cell lrd_cc_encode

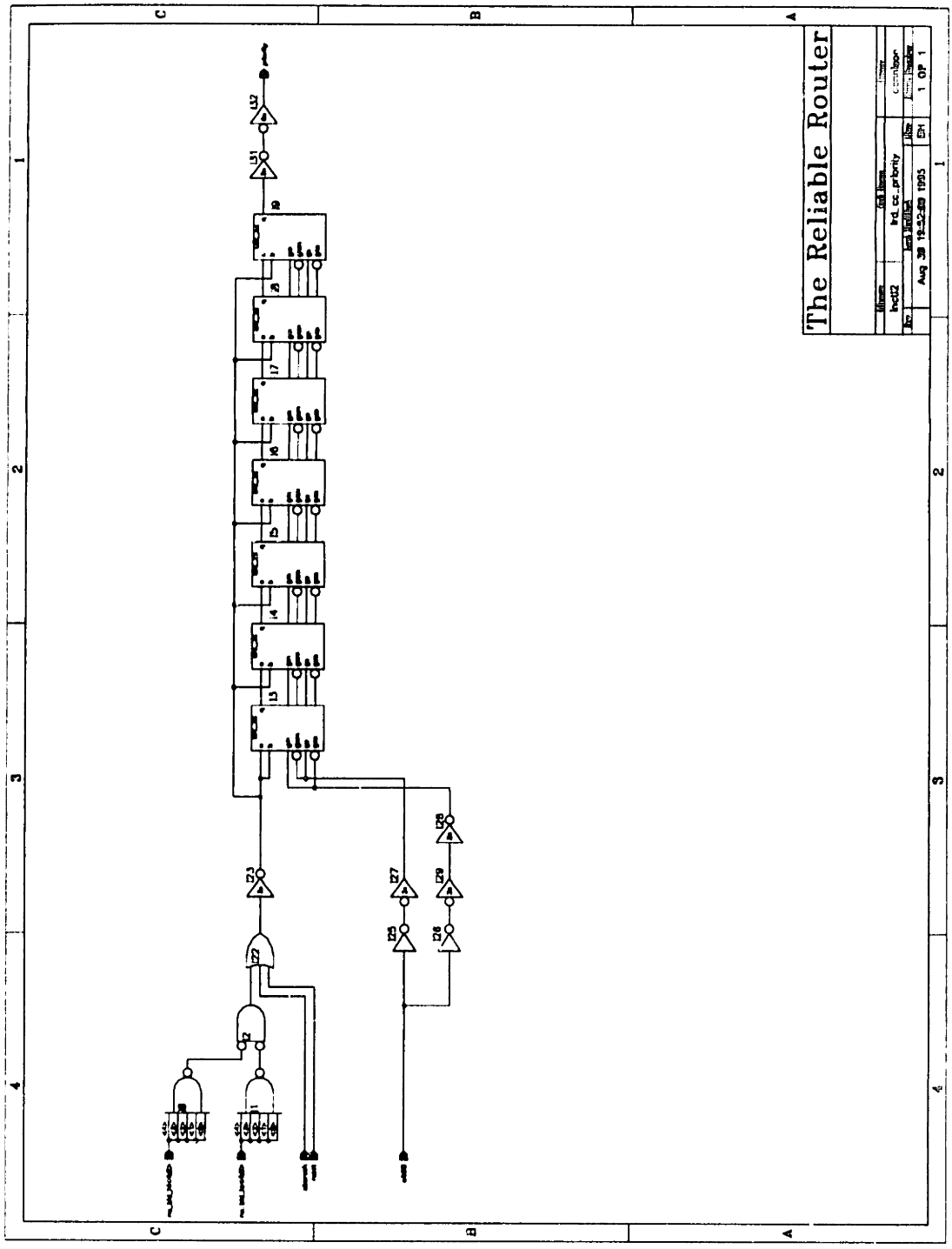


Figure A-126: Library inctl2, cell lrd_cc_priority

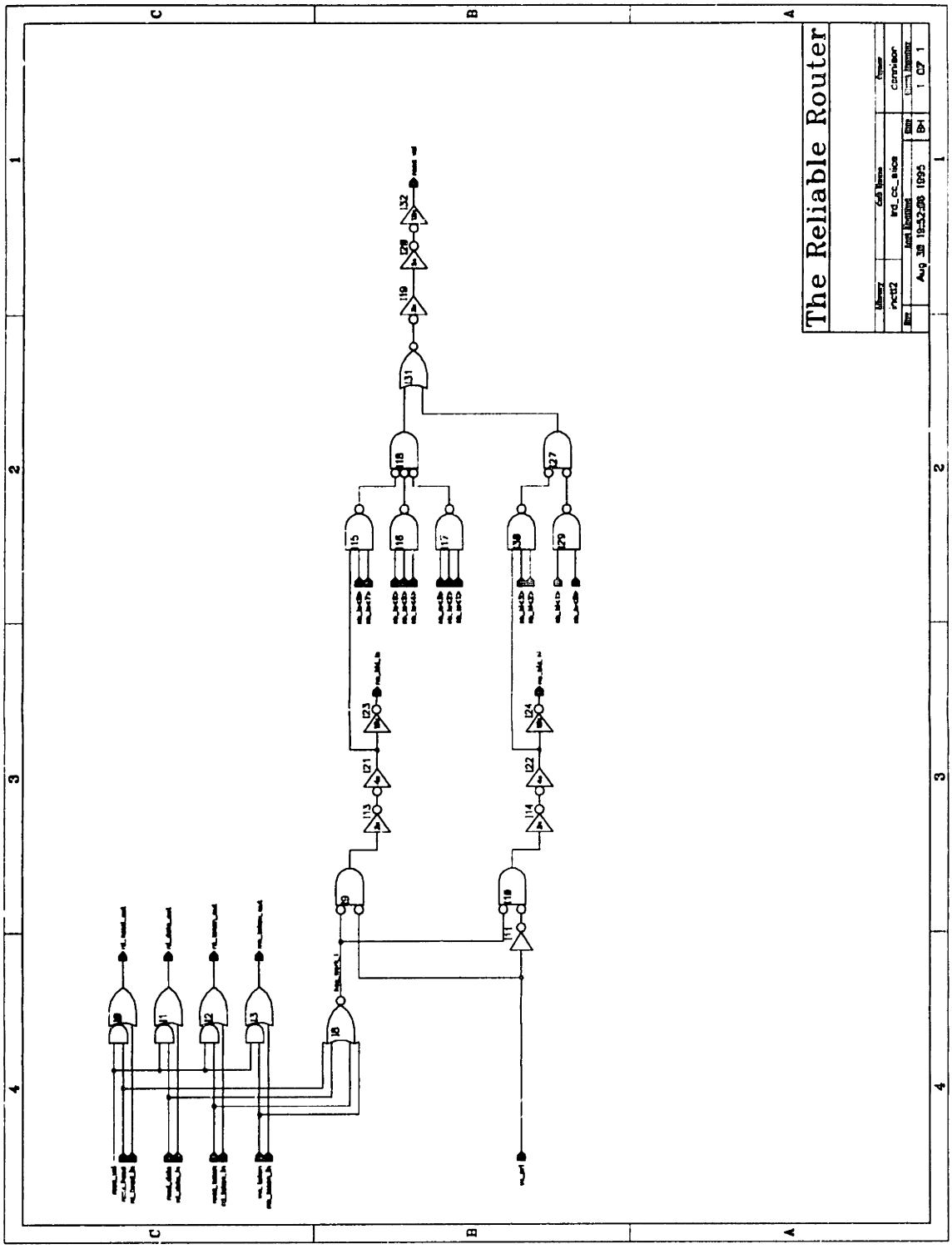


Figure A-127: Library inctl2, cell lrd_cc_slice

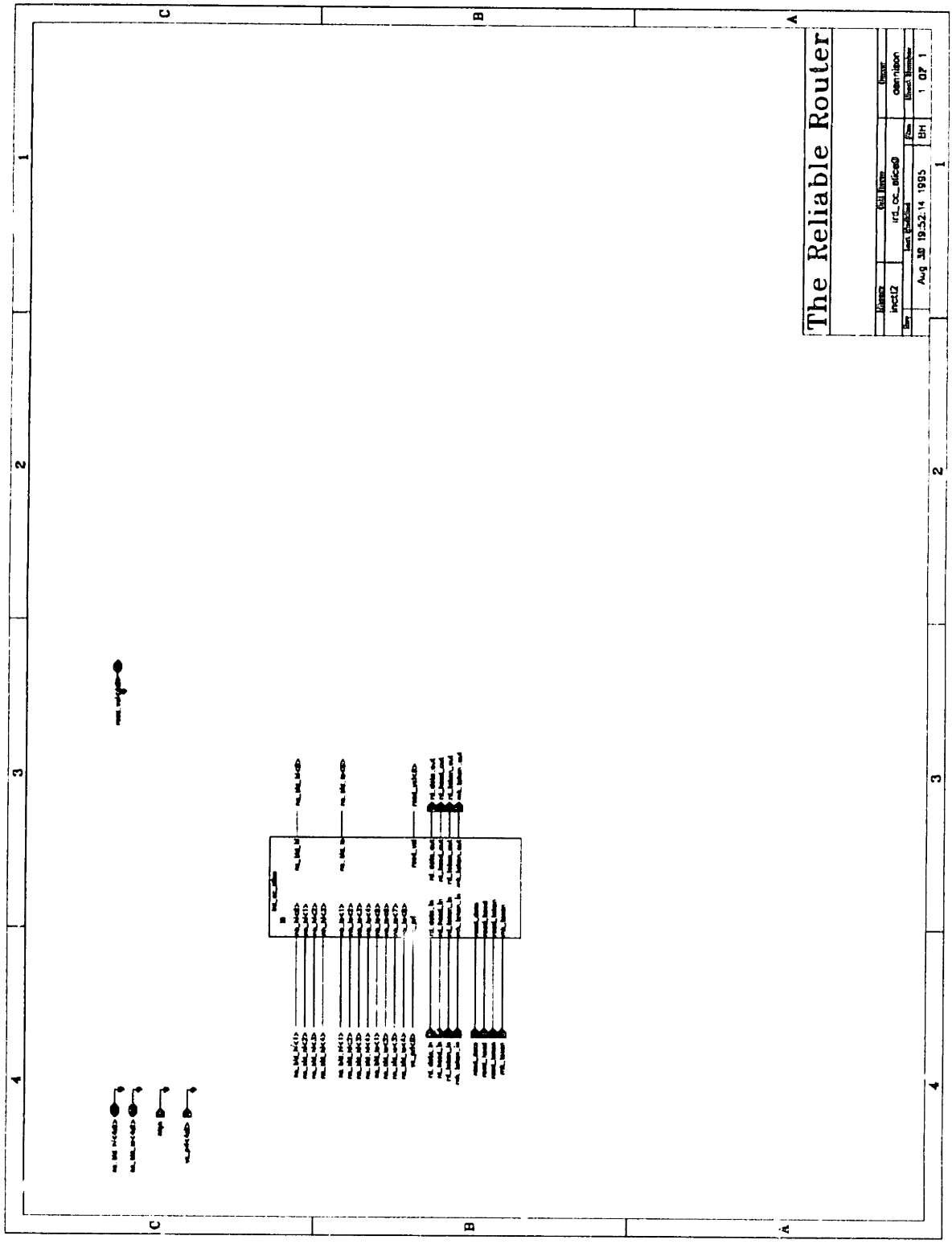


Figure A-128: Library inctl2, cell lrd_cc_slice0

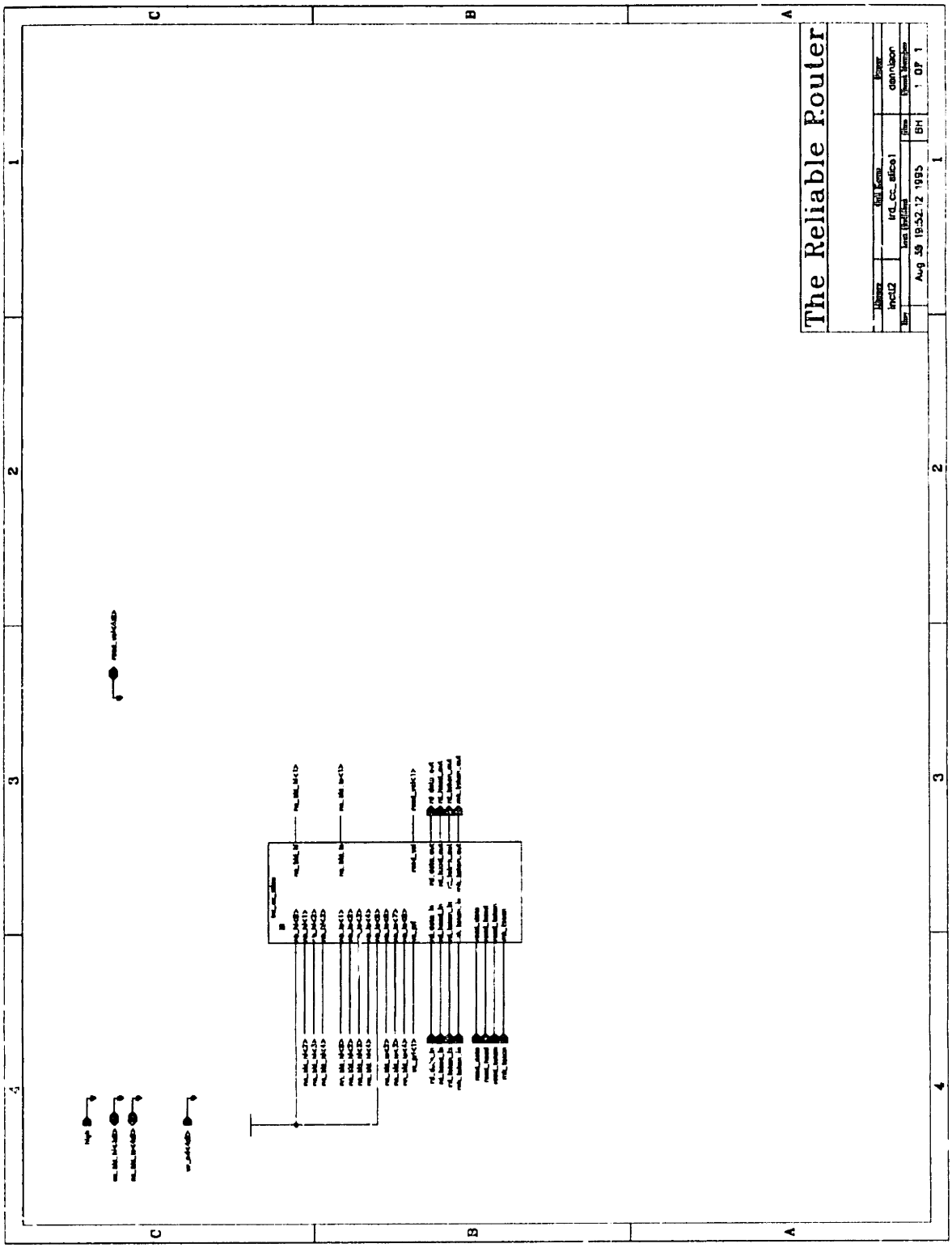


Figure A-129: Library inctl2, cell lrd_cc_slice1

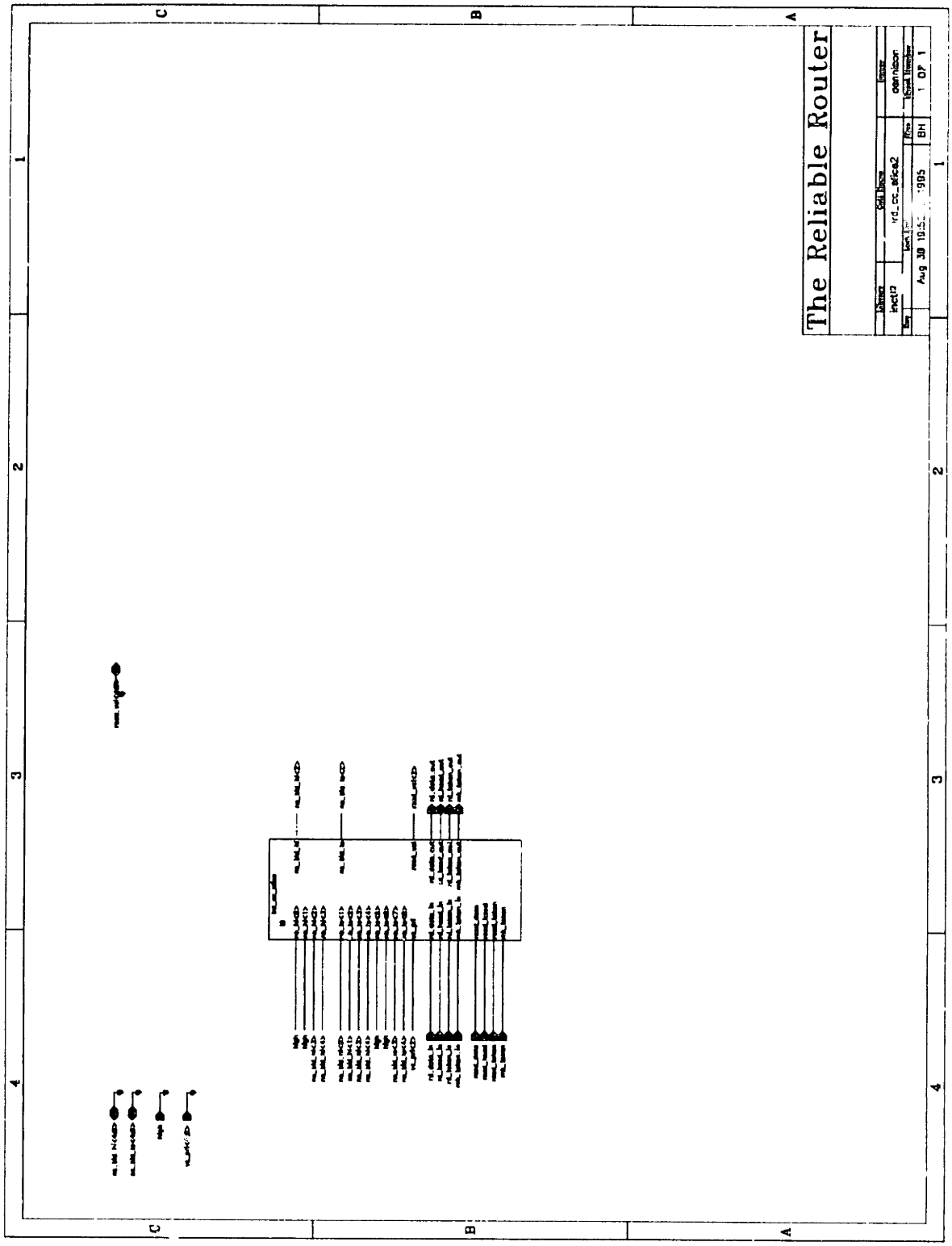


Figure A-130: Library inctl2, cell lrd_cc_slice2

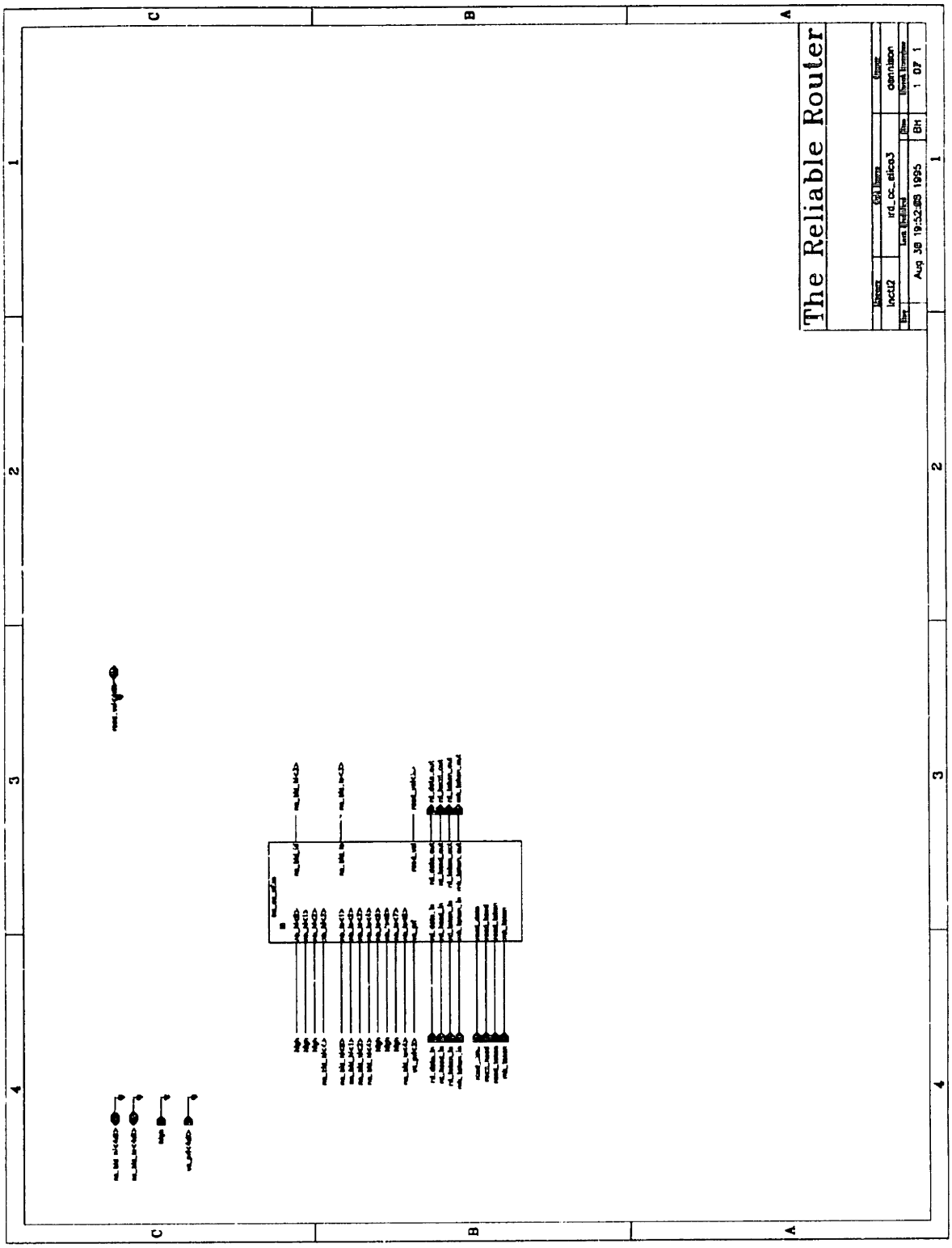


Figure A-131: Library inctl2, cell lrd_cc_slice3

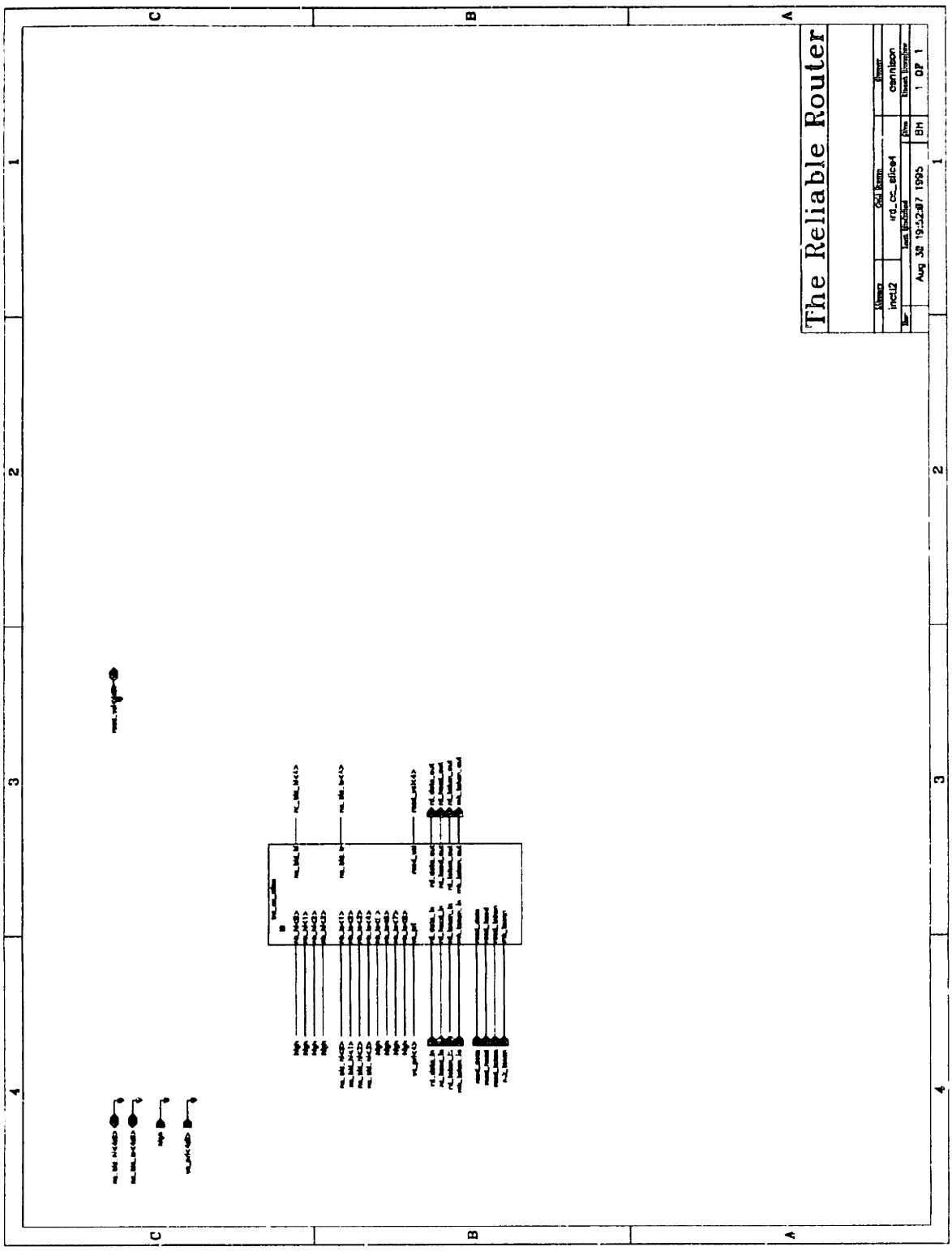


Figure A-132: Library inctl2, cell lrd_cc_slice4

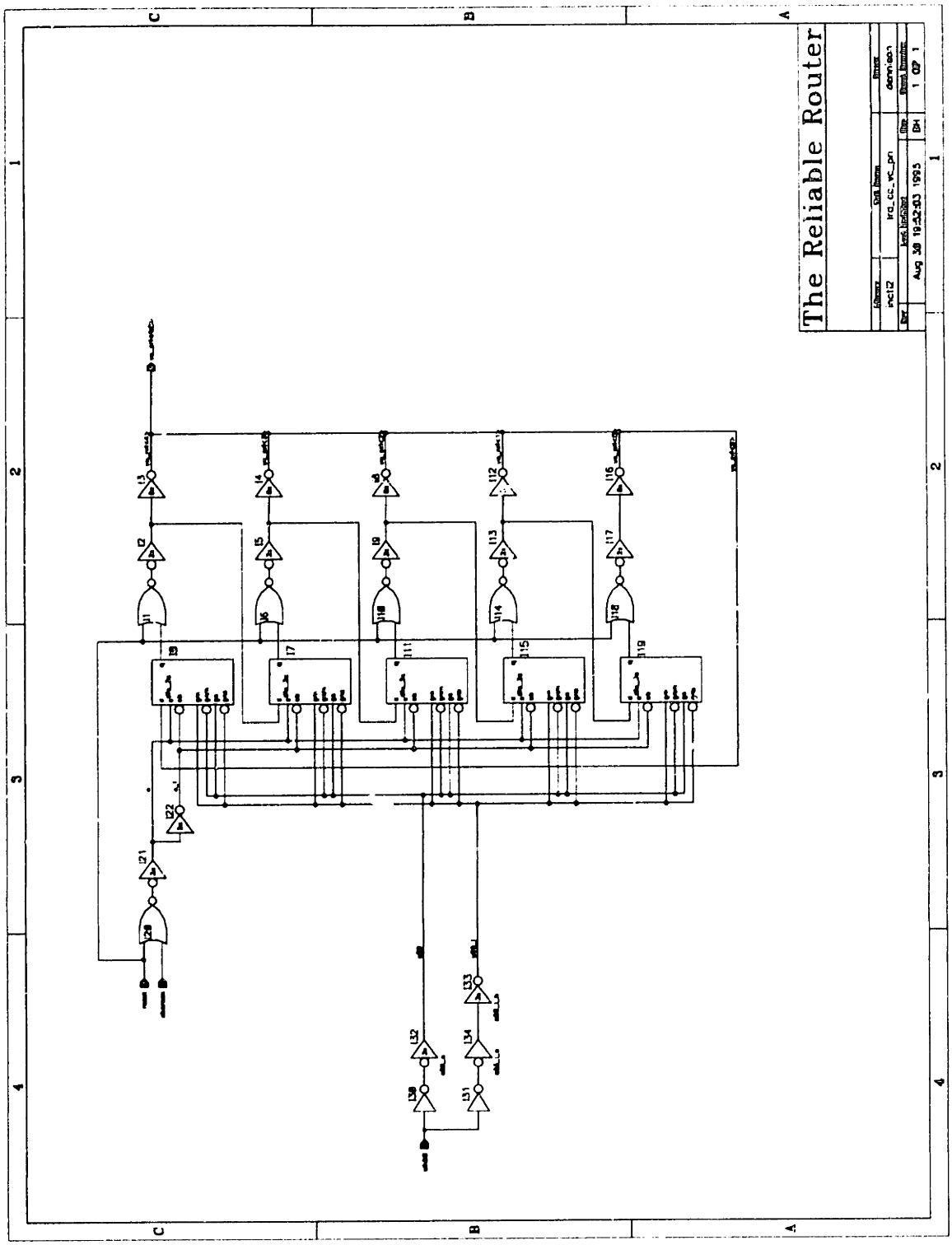


Figure A-133: Library inctl2, cell lrd_cc_vc_pri

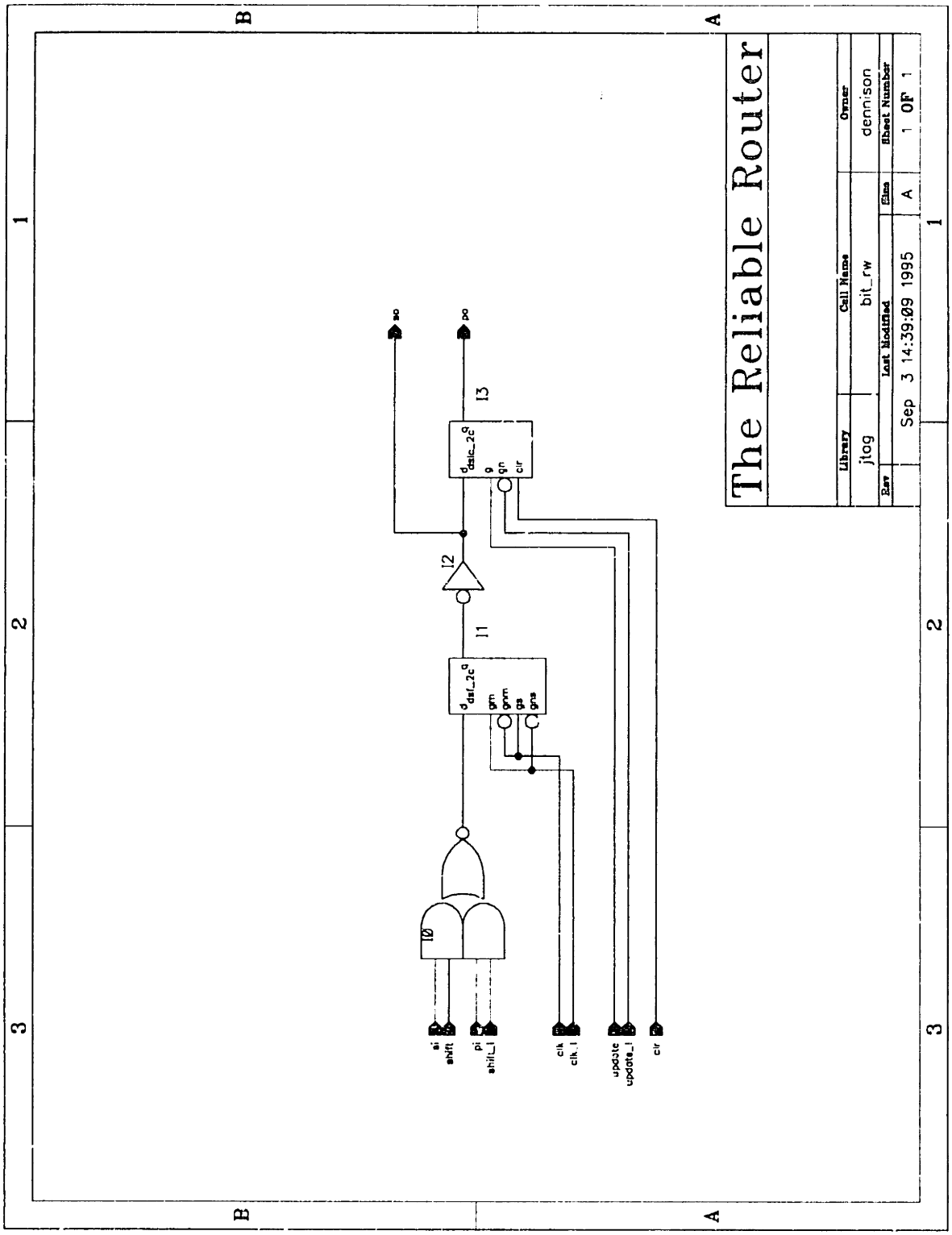


Figure A-134: Library jtag, cell bit_rw

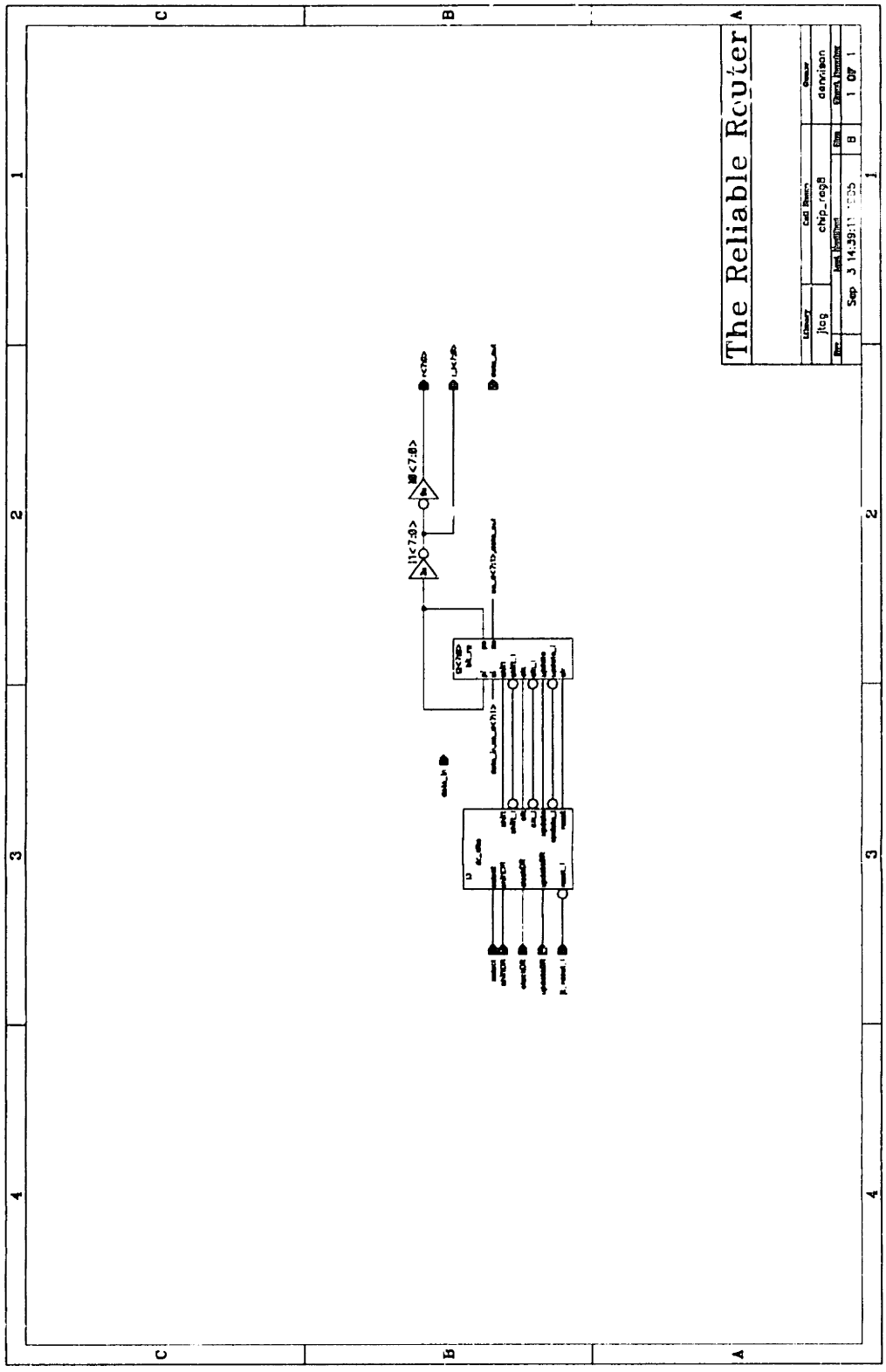


Figure A-135: Library jtag, cell chip_reg8

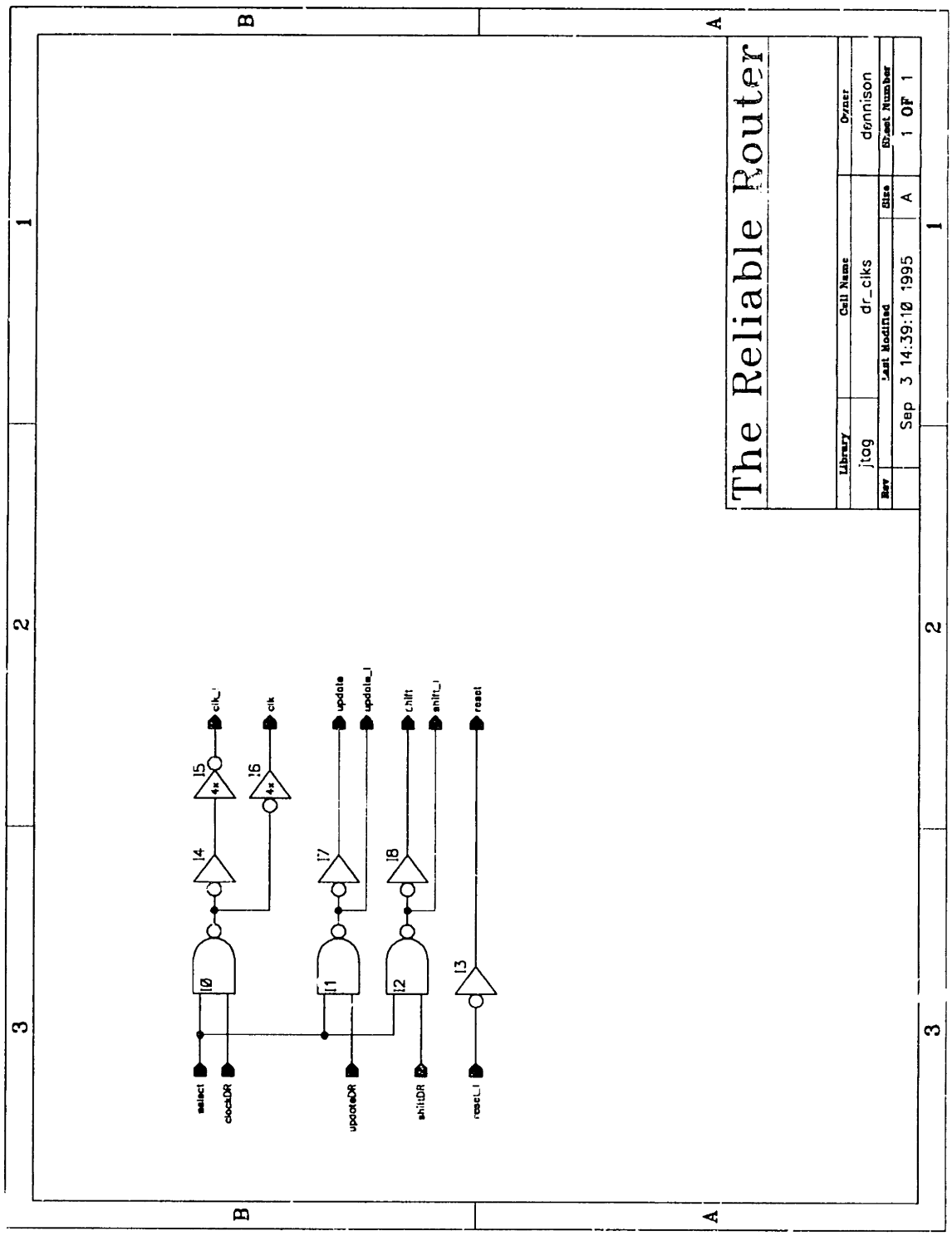


Figure A-136: Library jtag, cell dr_clks

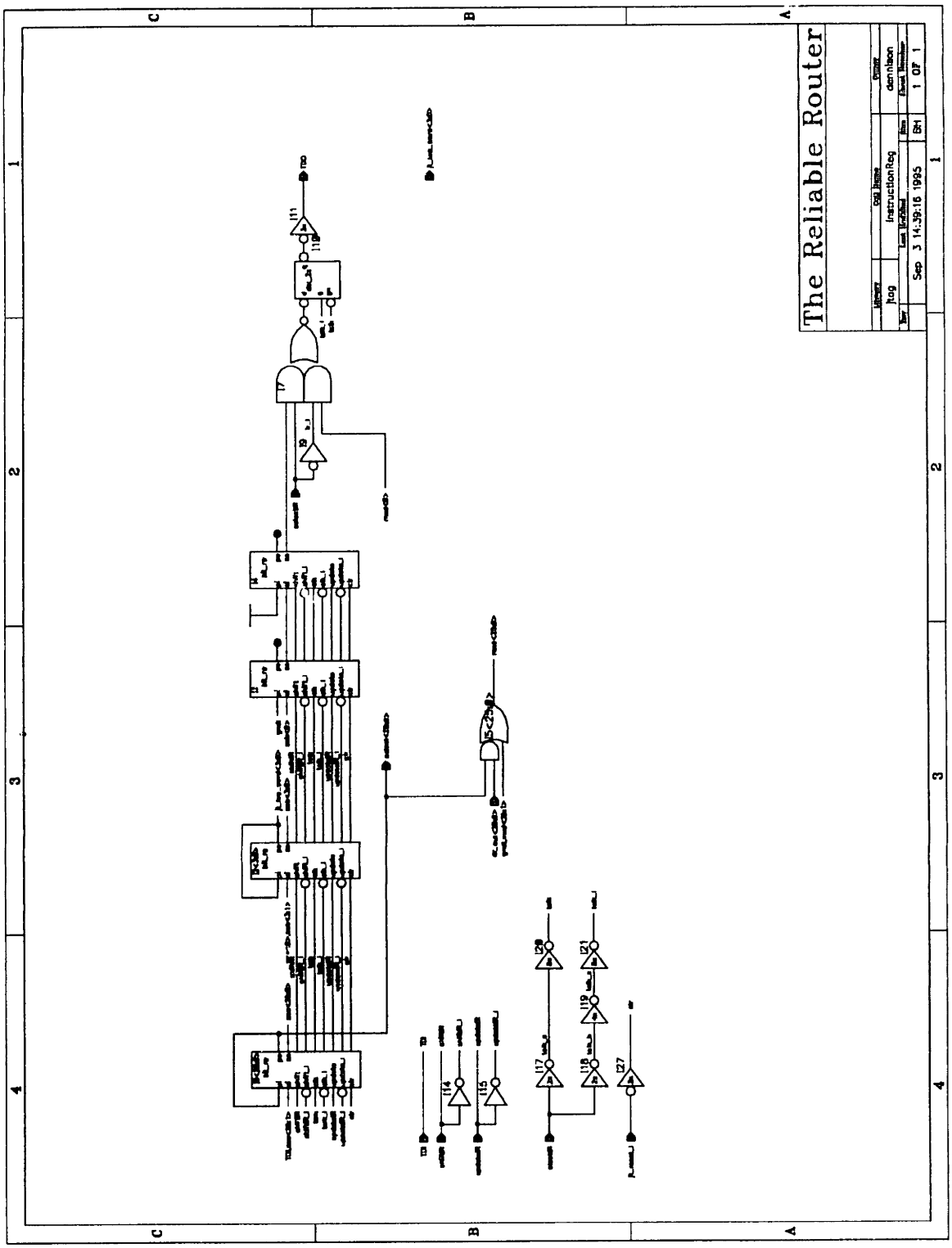


Figure A-137: Library jtag, cell instructionReg

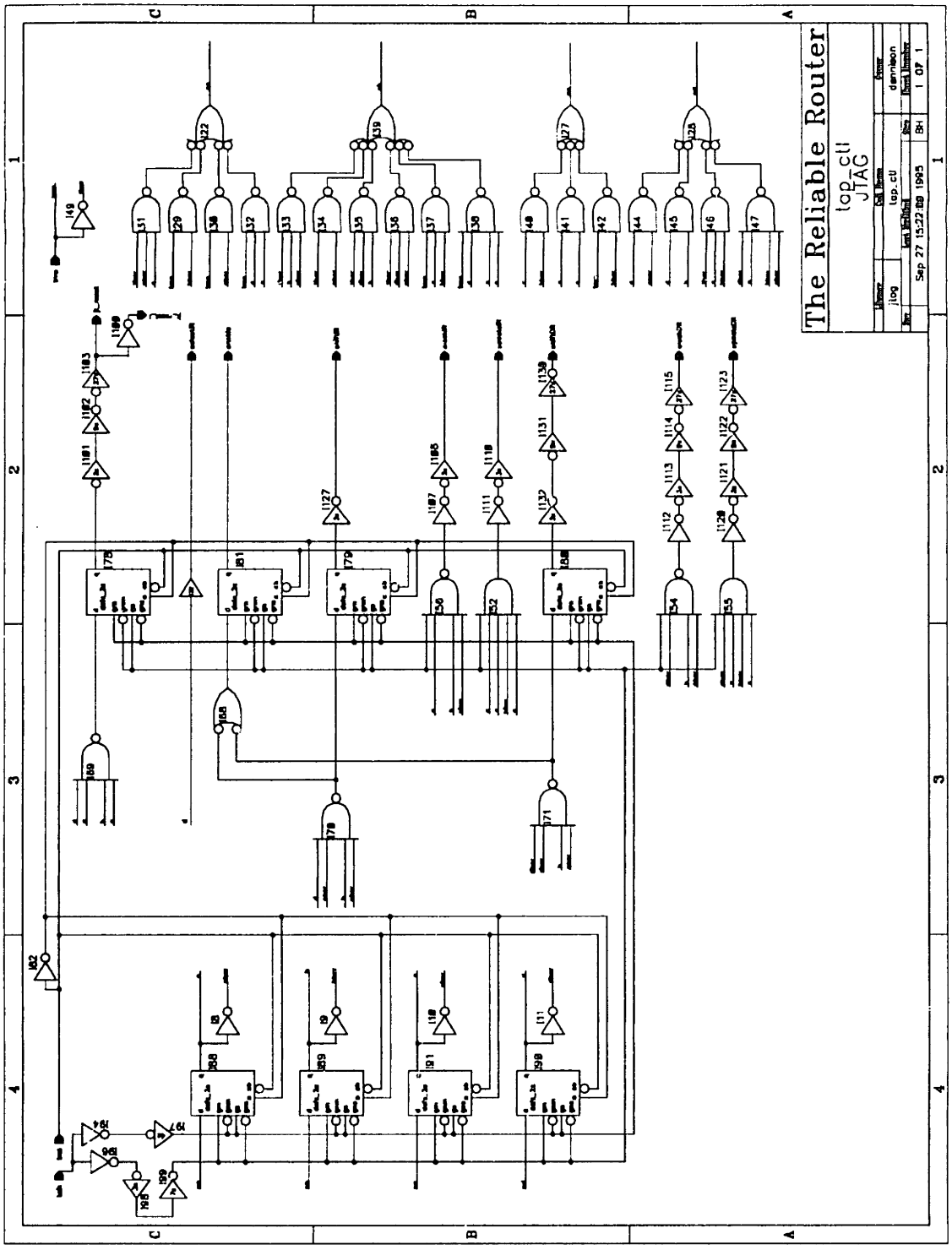


Figure A-138: Library jtag, cell tap_ctl

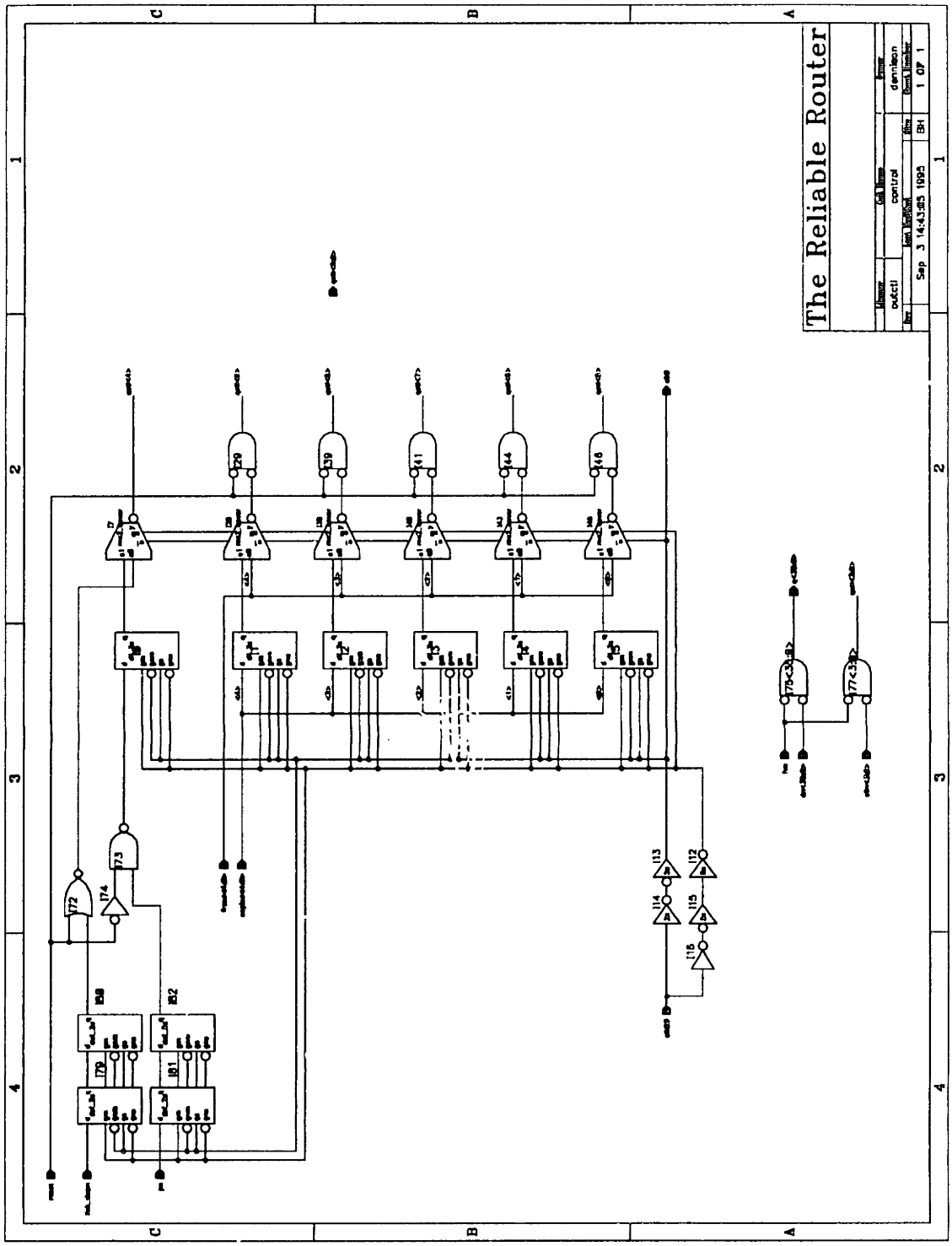


Figure A-139: Library outctl, cell control

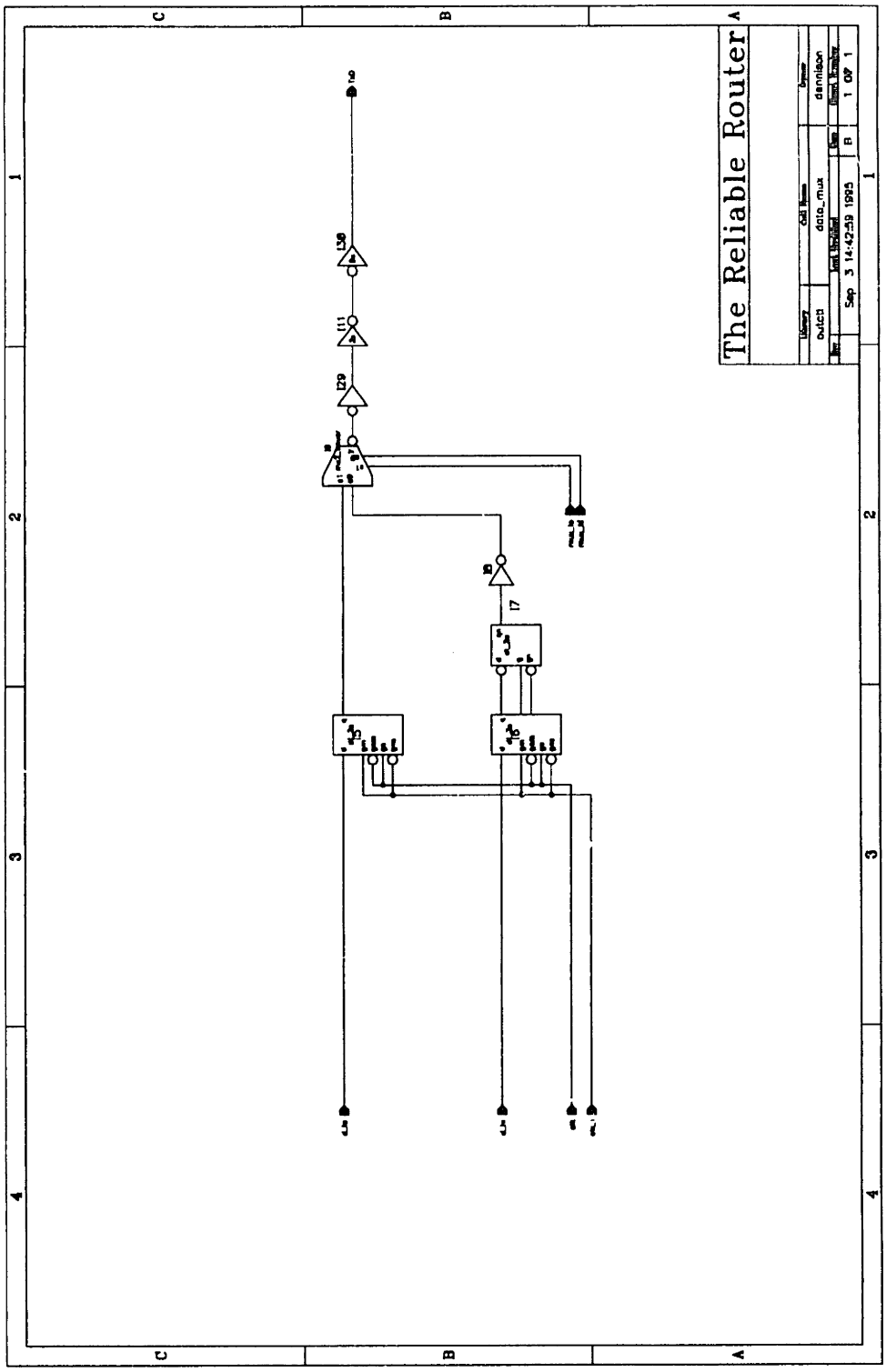


Figure A-140: Library outctl, cell data_mux

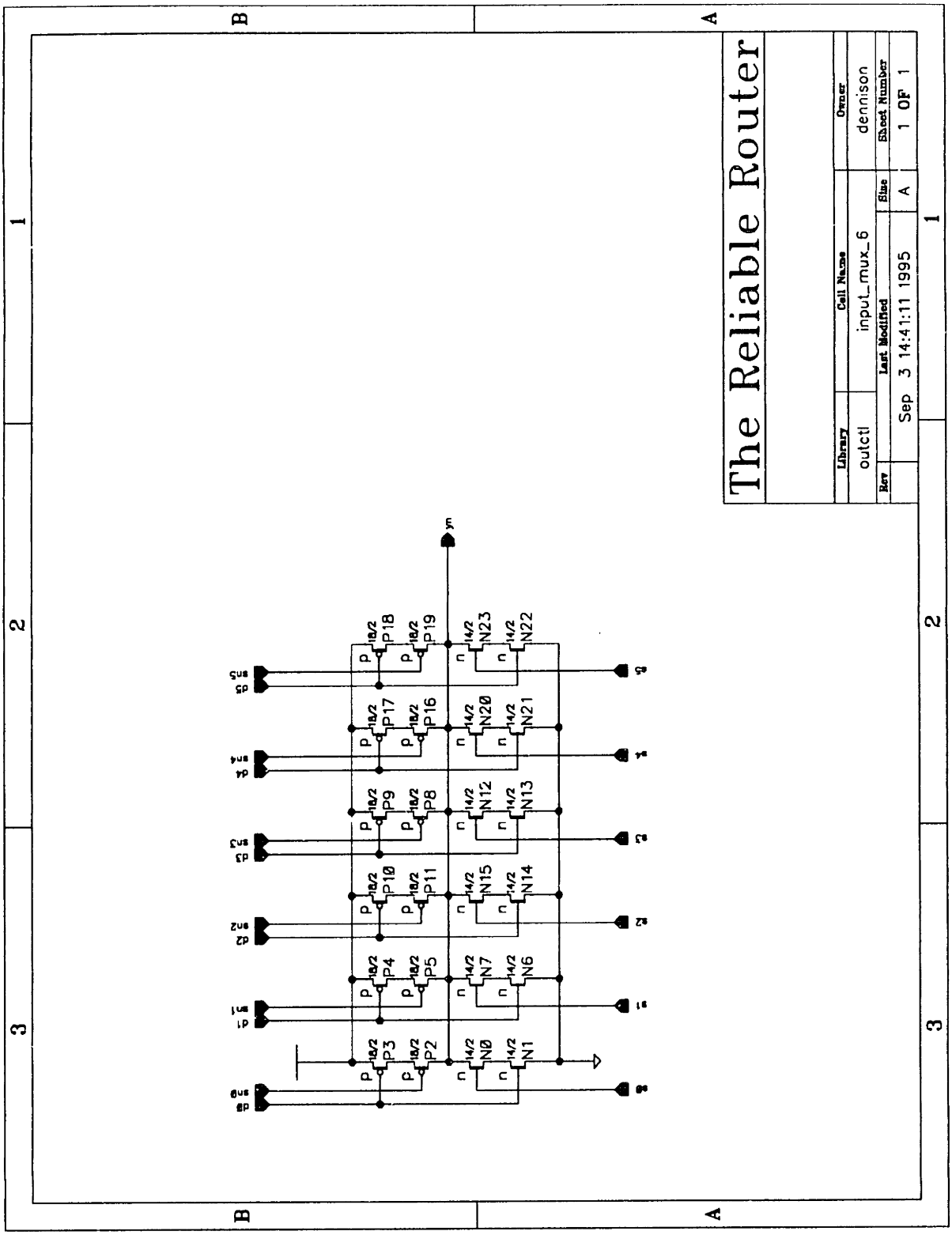


Figure A-141: Library outctl, cell input_mux_6

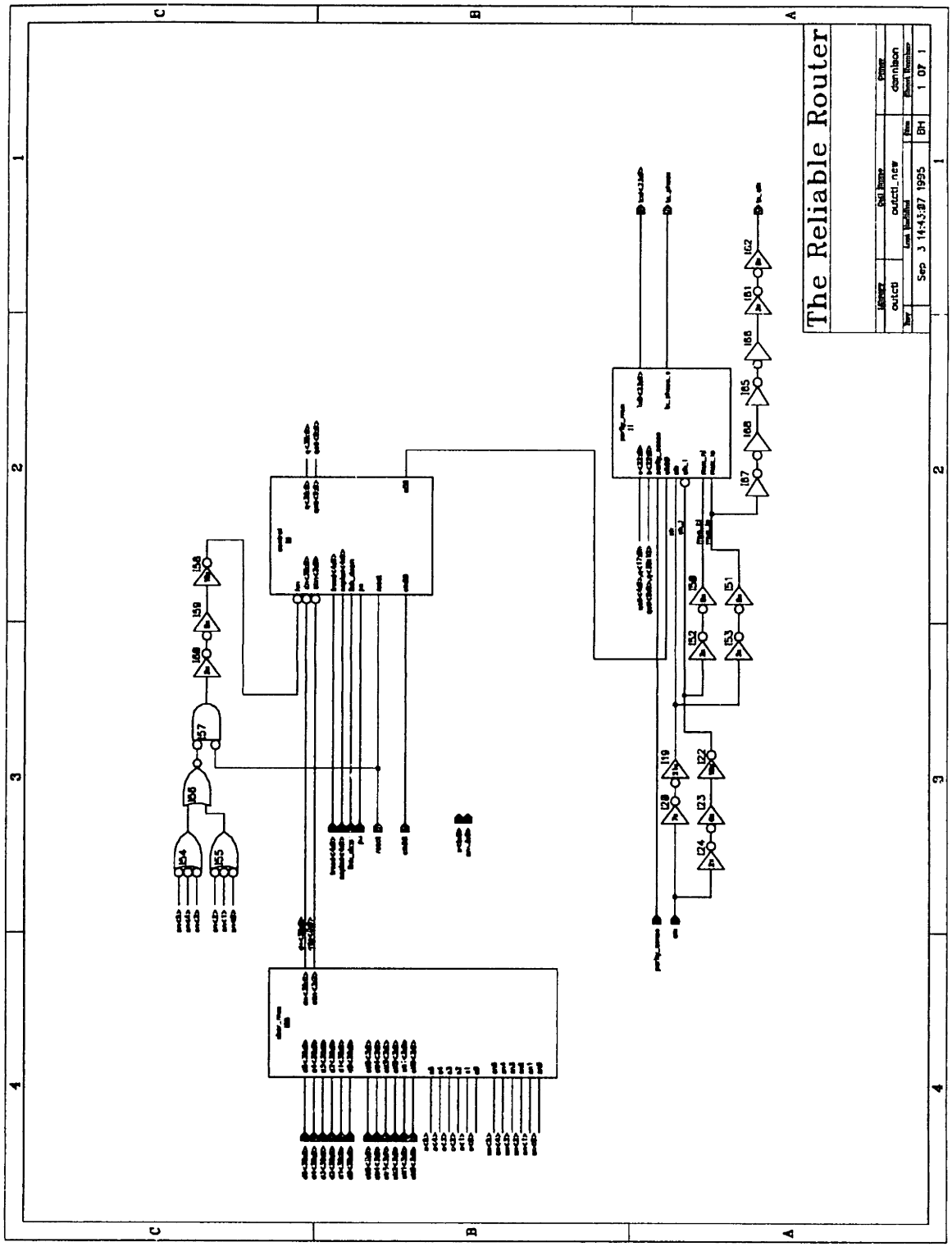


Figure A-142: Library outctl, cell outctl_new

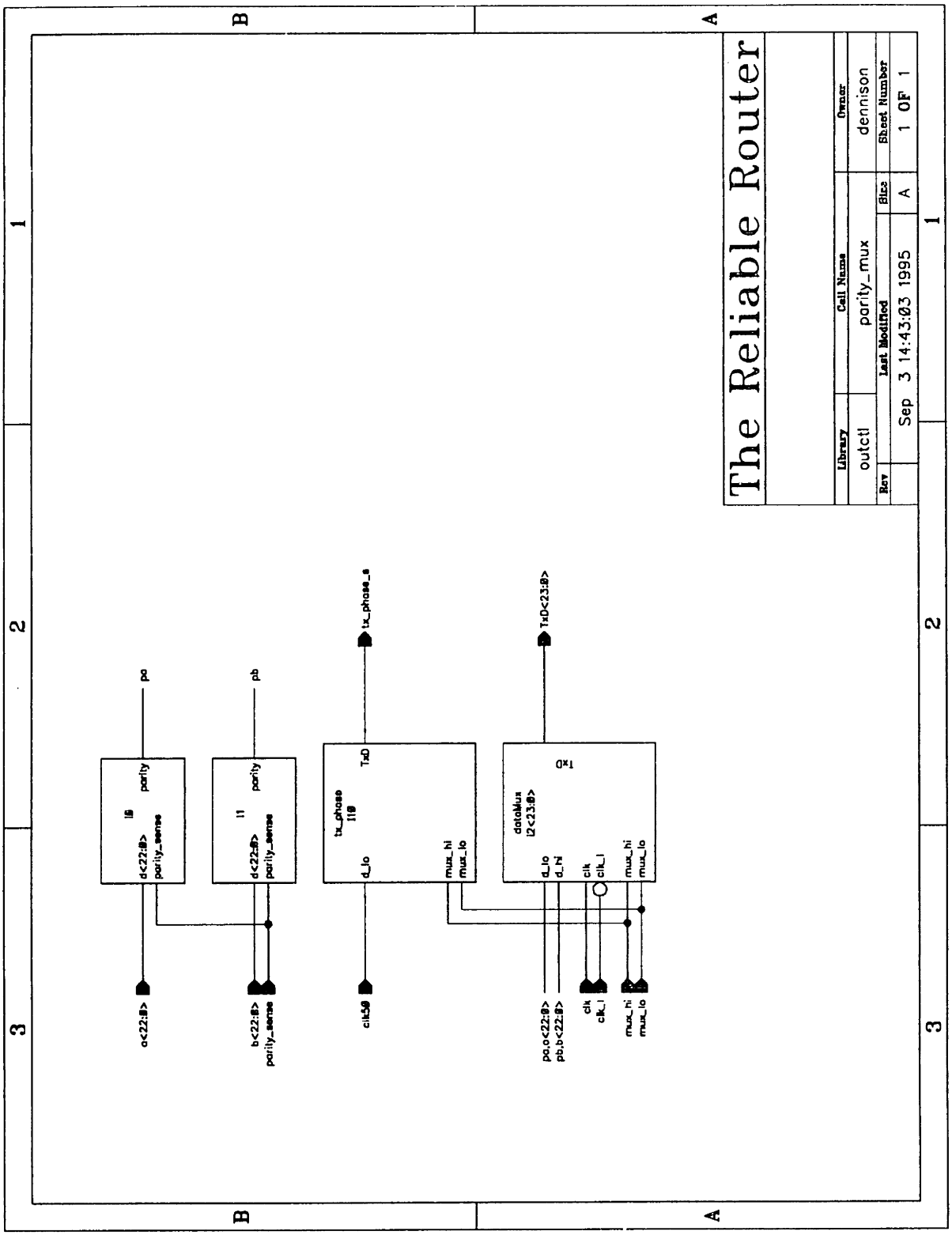


Figure A-143: Library outctl, cell parity_mux

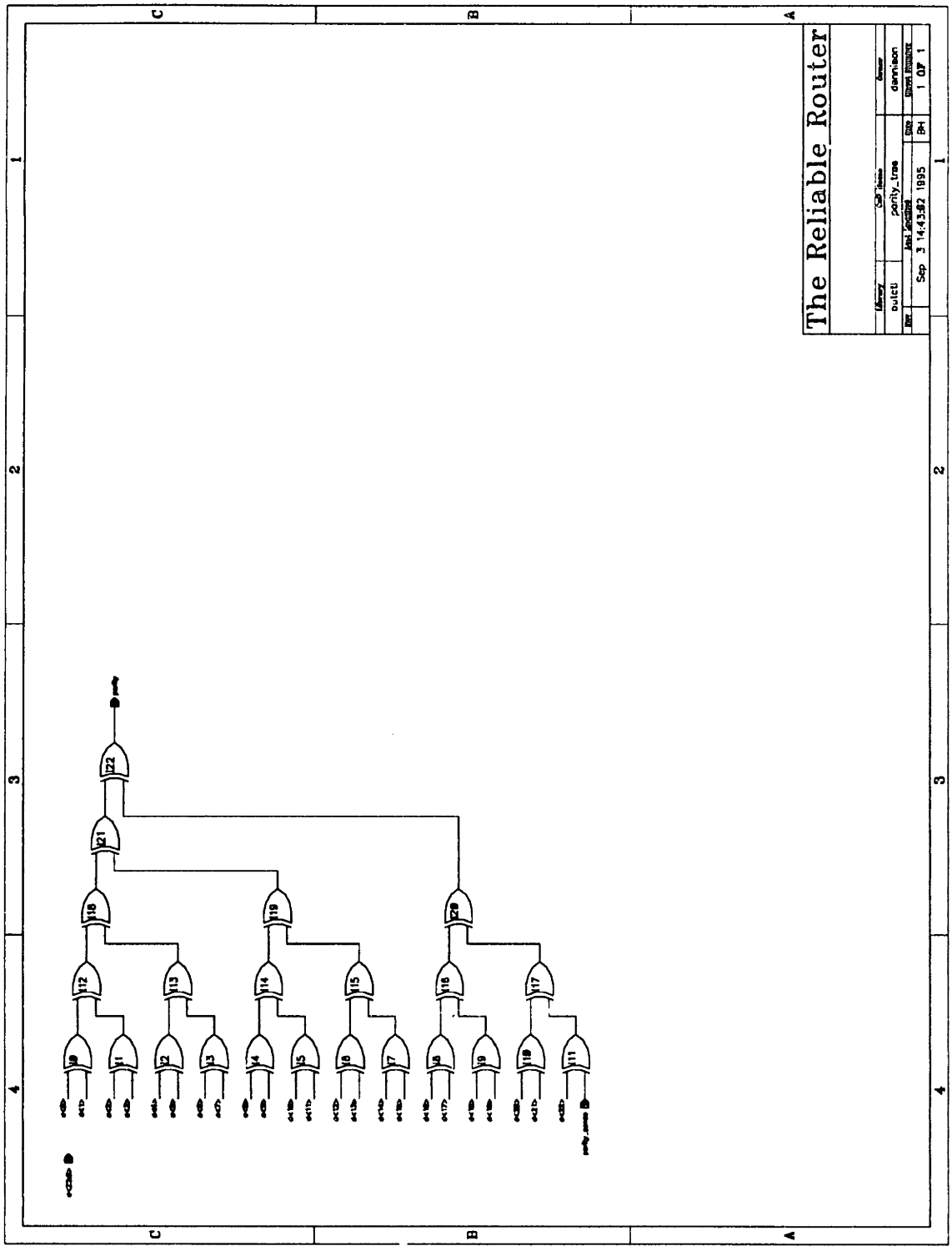


Figure A-144: Library outctl, cell parity_tree

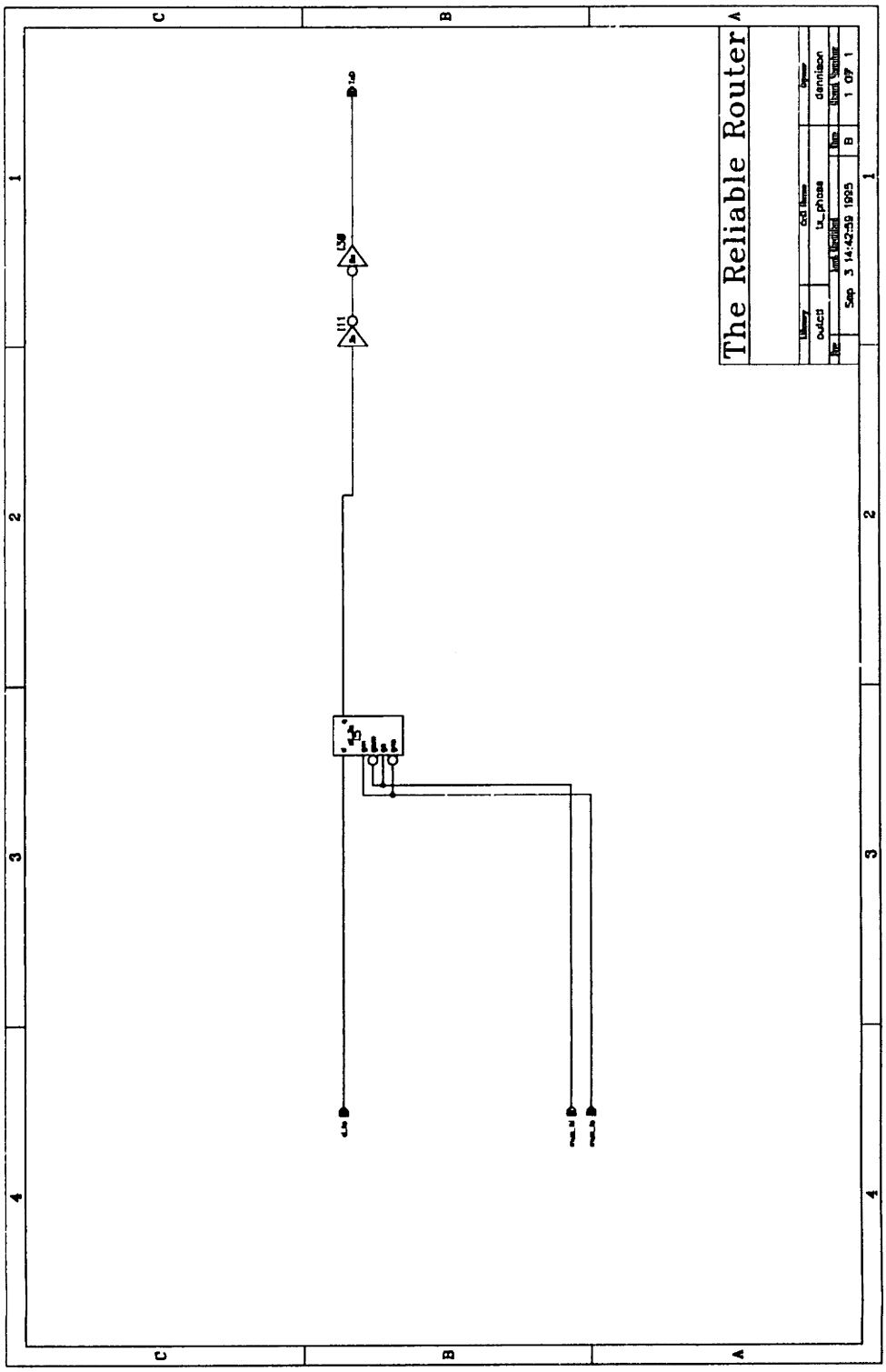


Figure A-145: Library outctl, cell tx_phase

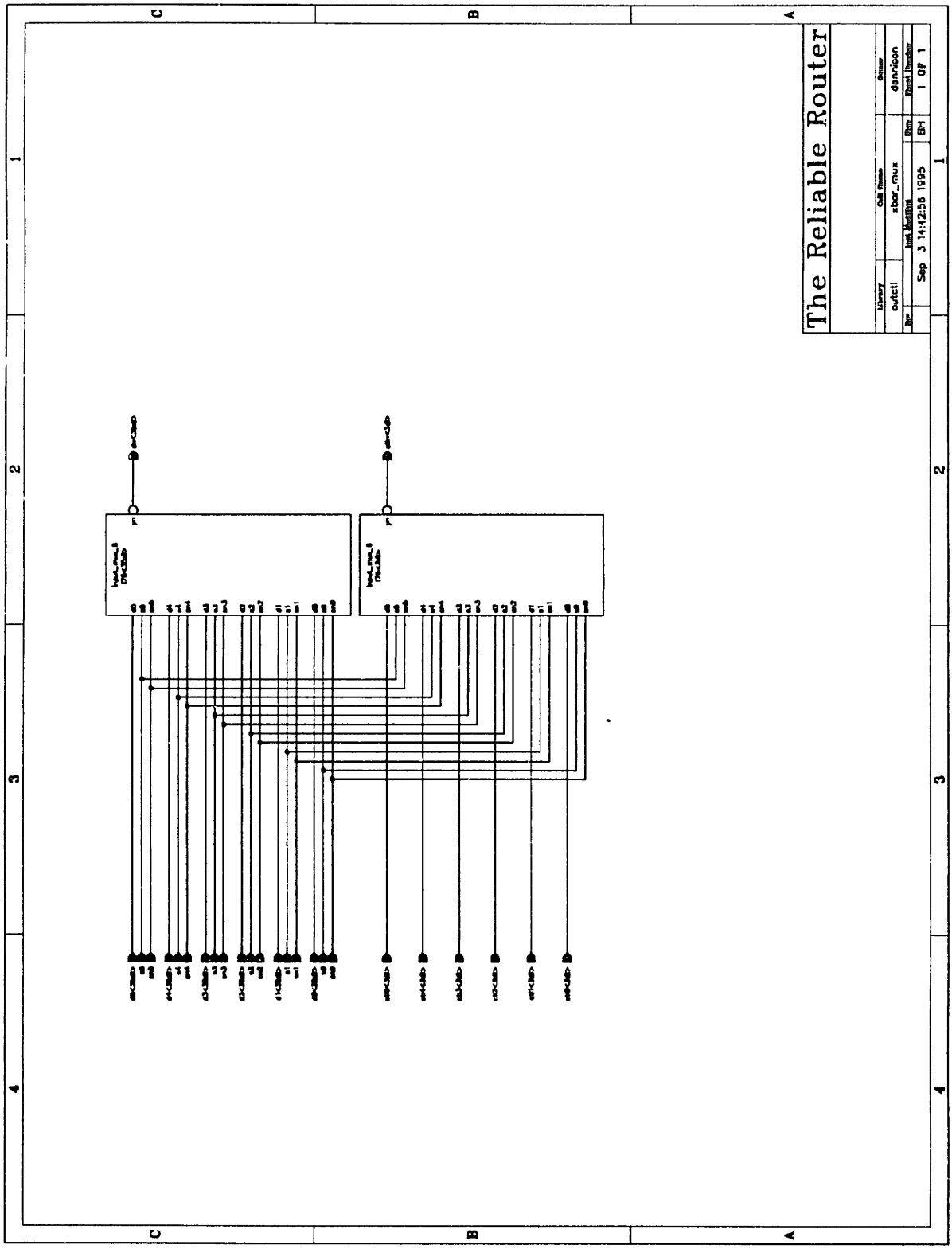


Figure A-146: Library outctl, cell xbar_mux

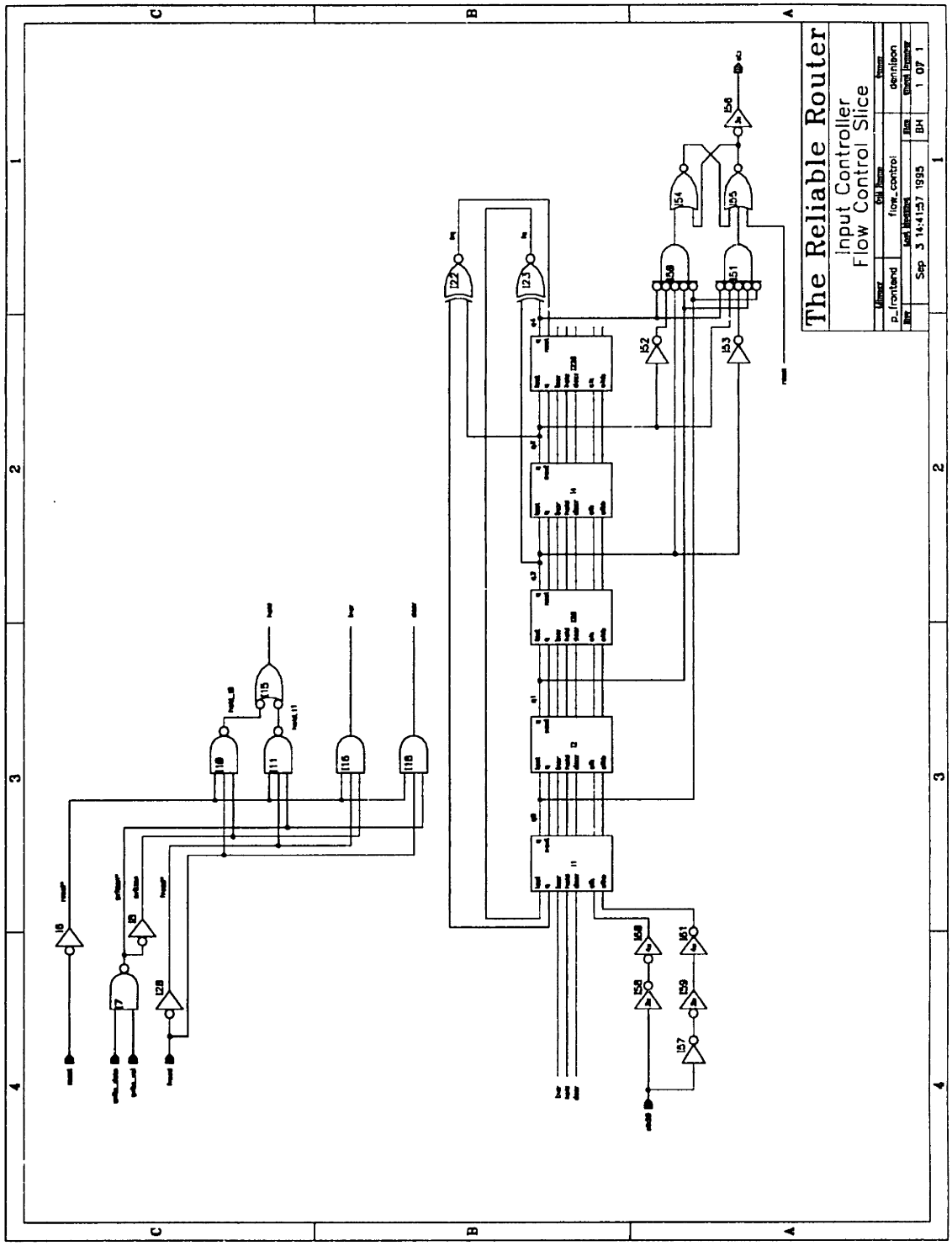


Figure A-147: Library p_frontend, cell flow_control

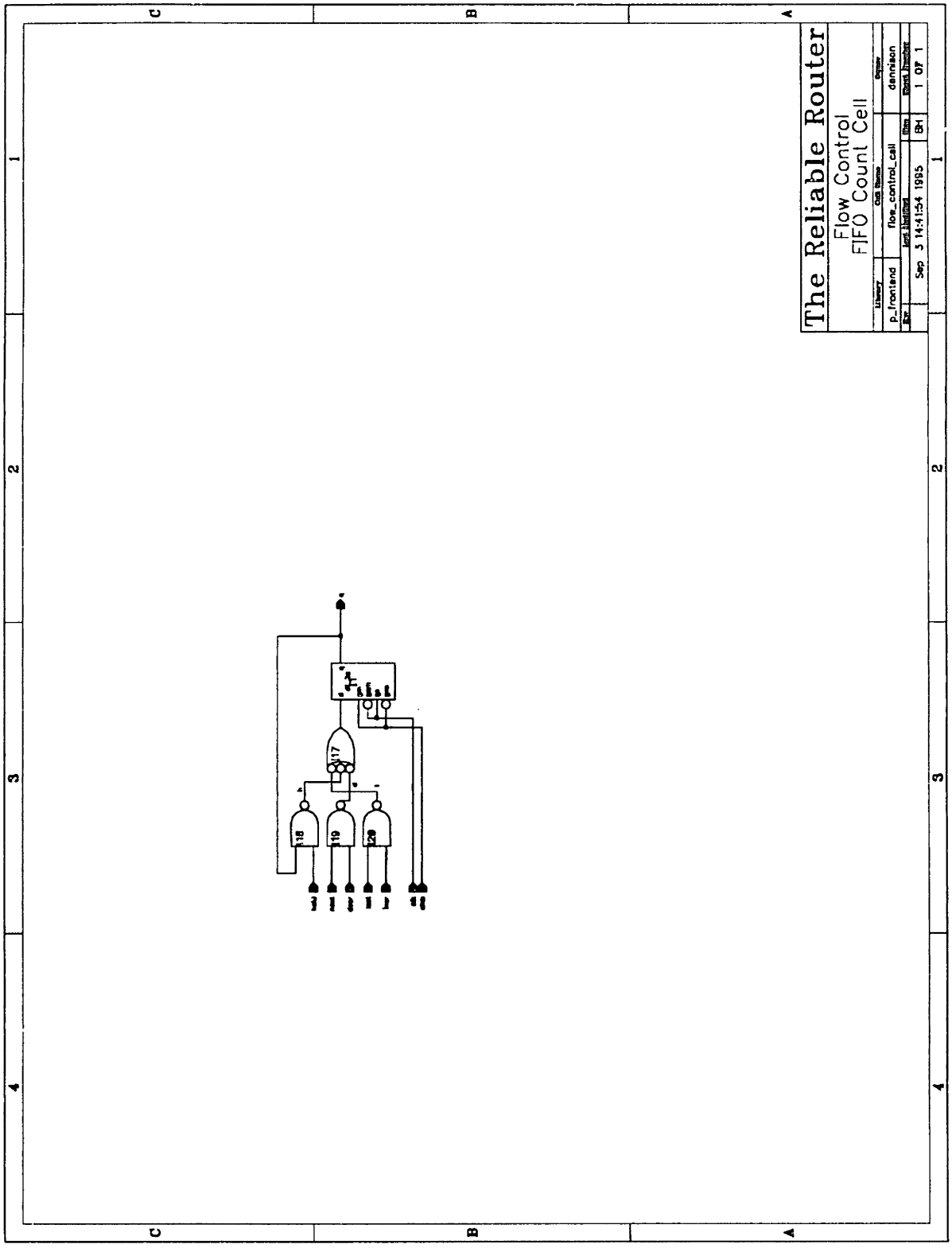


Figure A-148: Library p_frontend, cell flow_control_cell

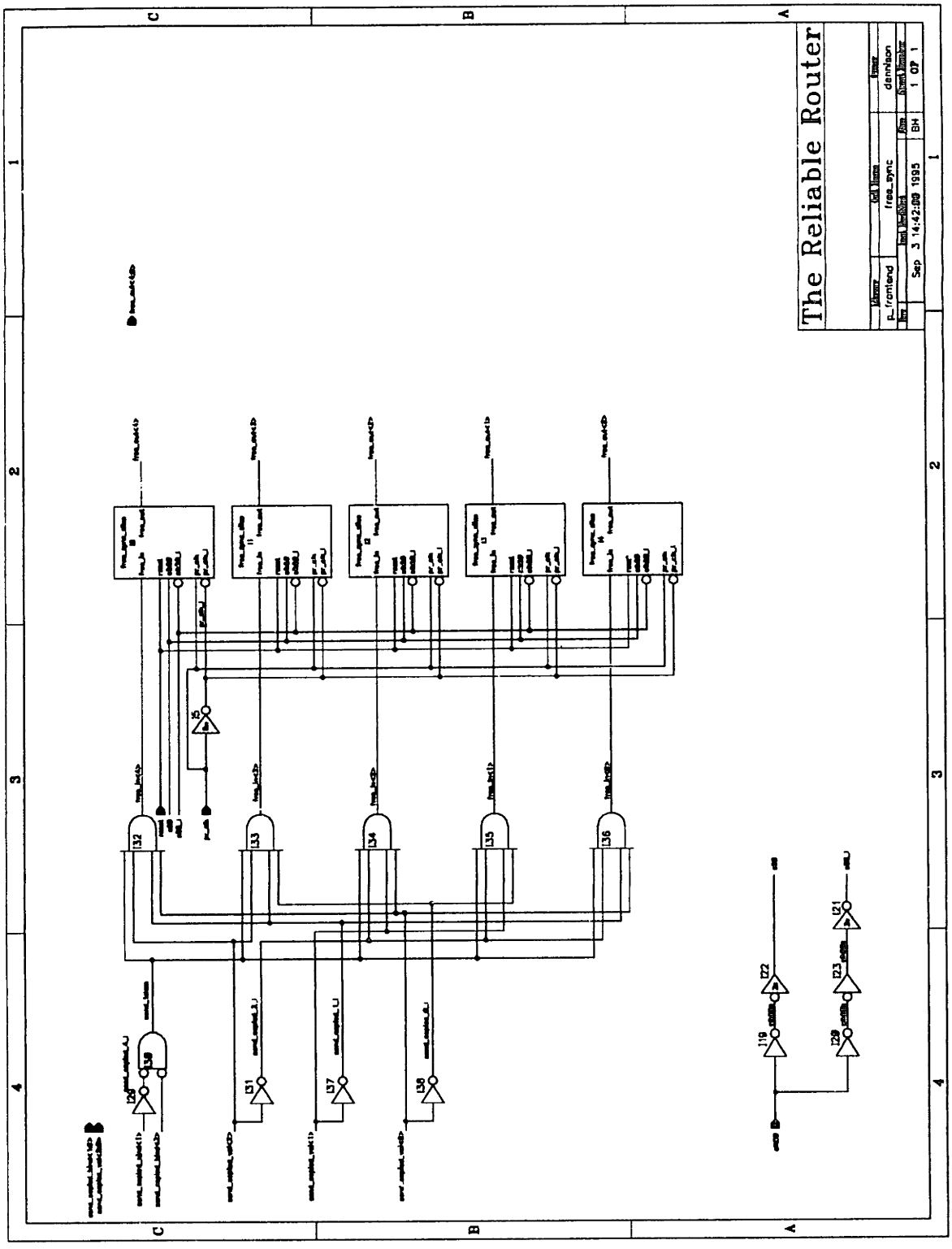


Figure A-149: Library p_frontend, cell free_sync

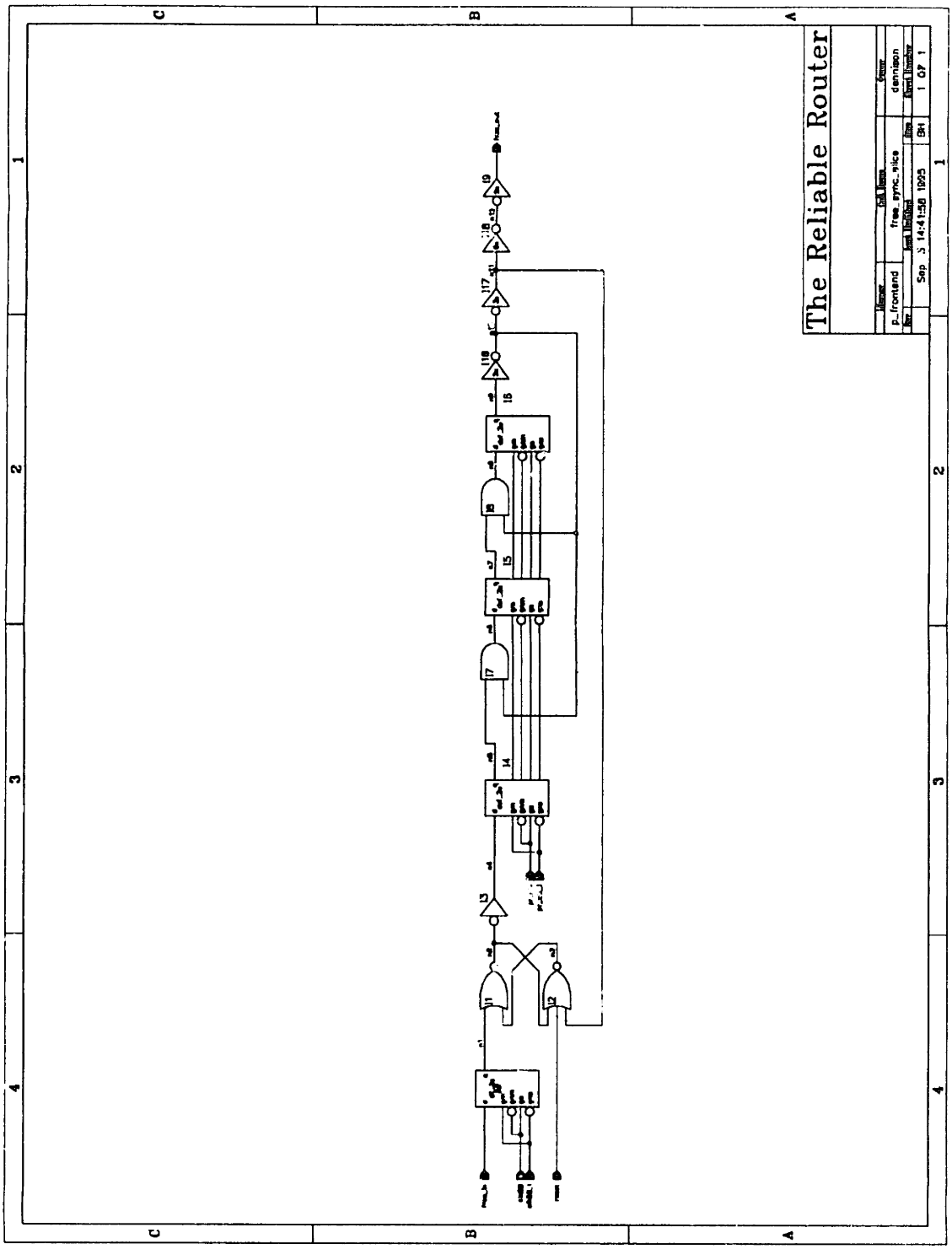


Figure A-150: Library p_frontend, cell free_sync_slice

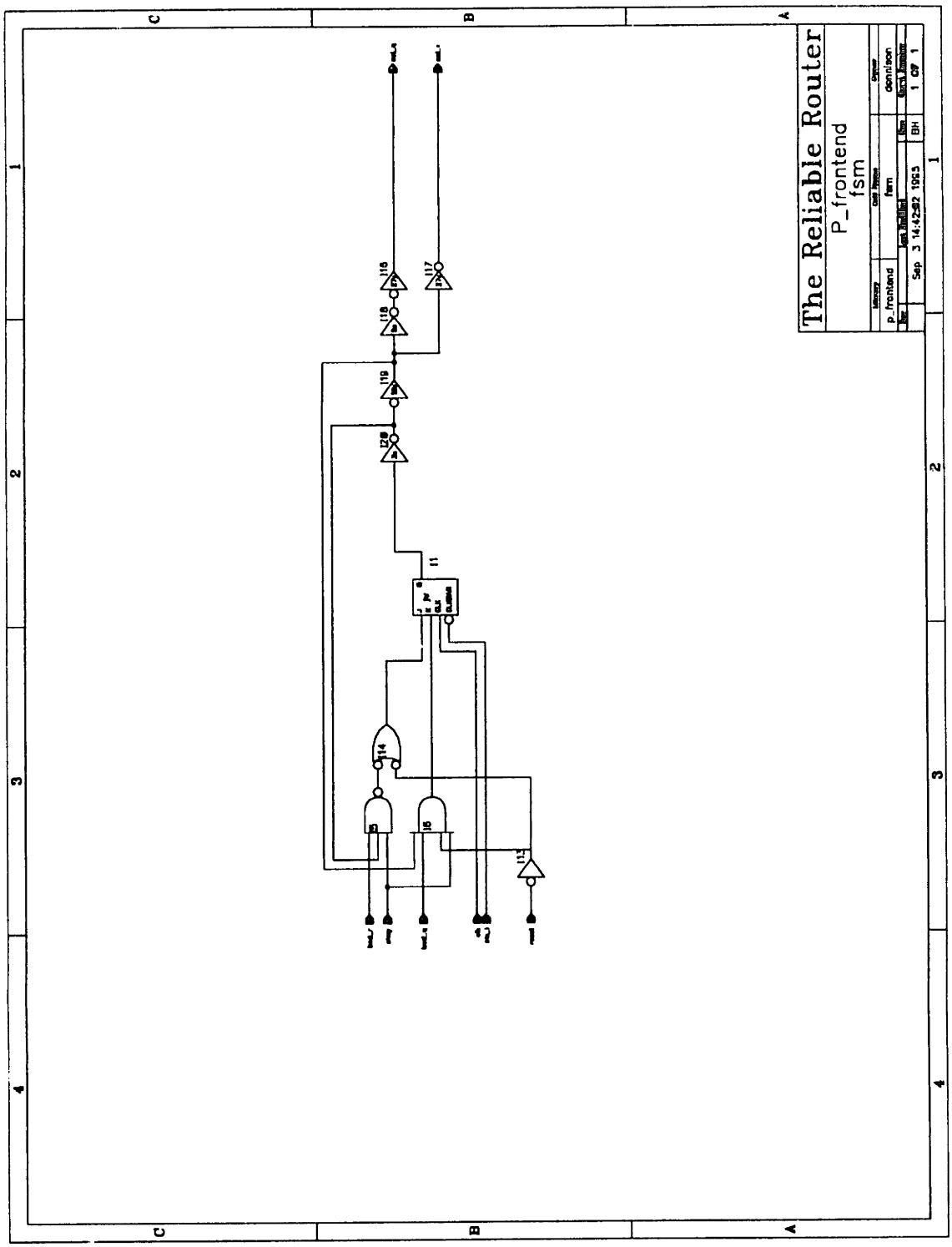


Figure A-151: Library p_frontend, cell fsm

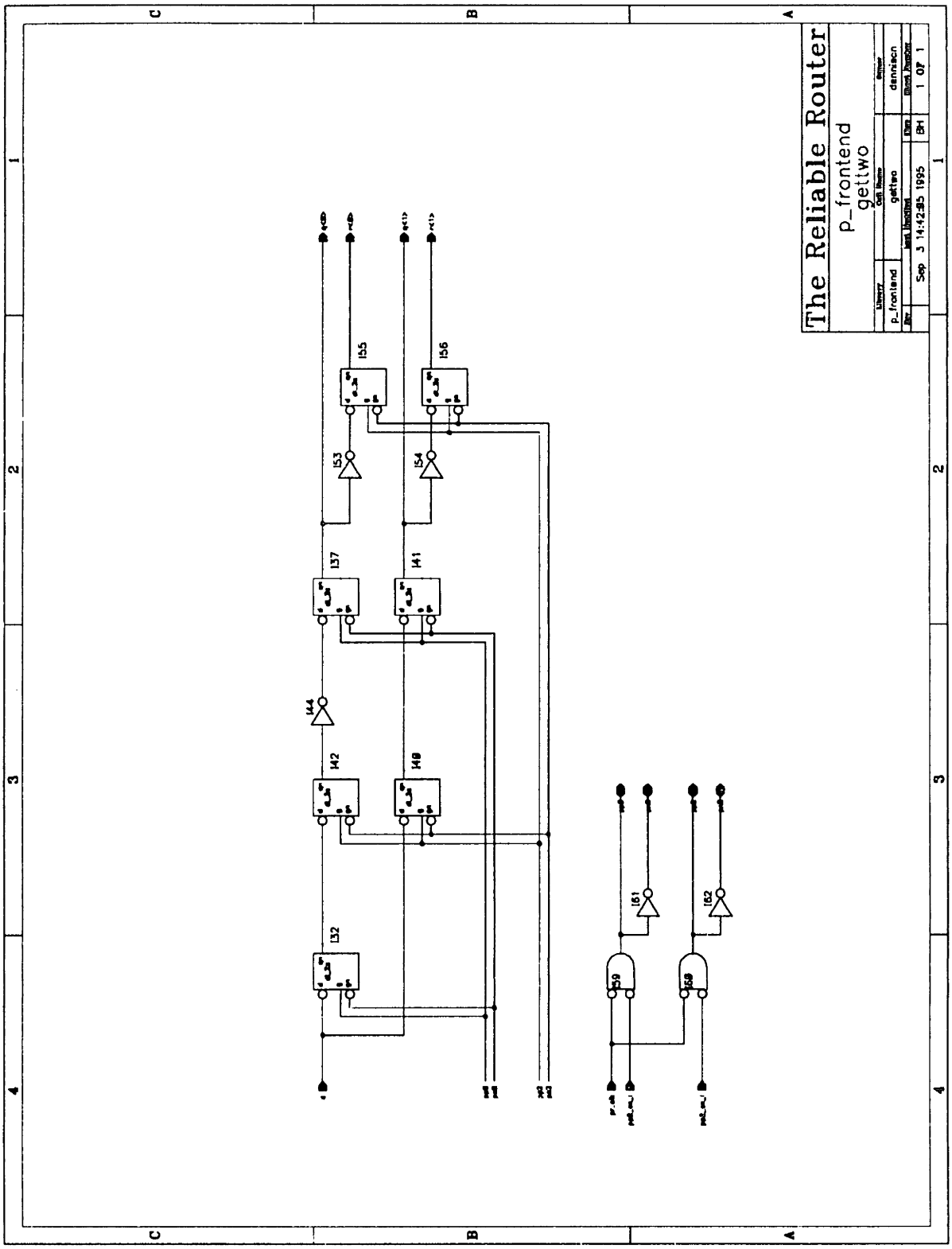


Figure A-152: Library p_frontend, cell gettwo

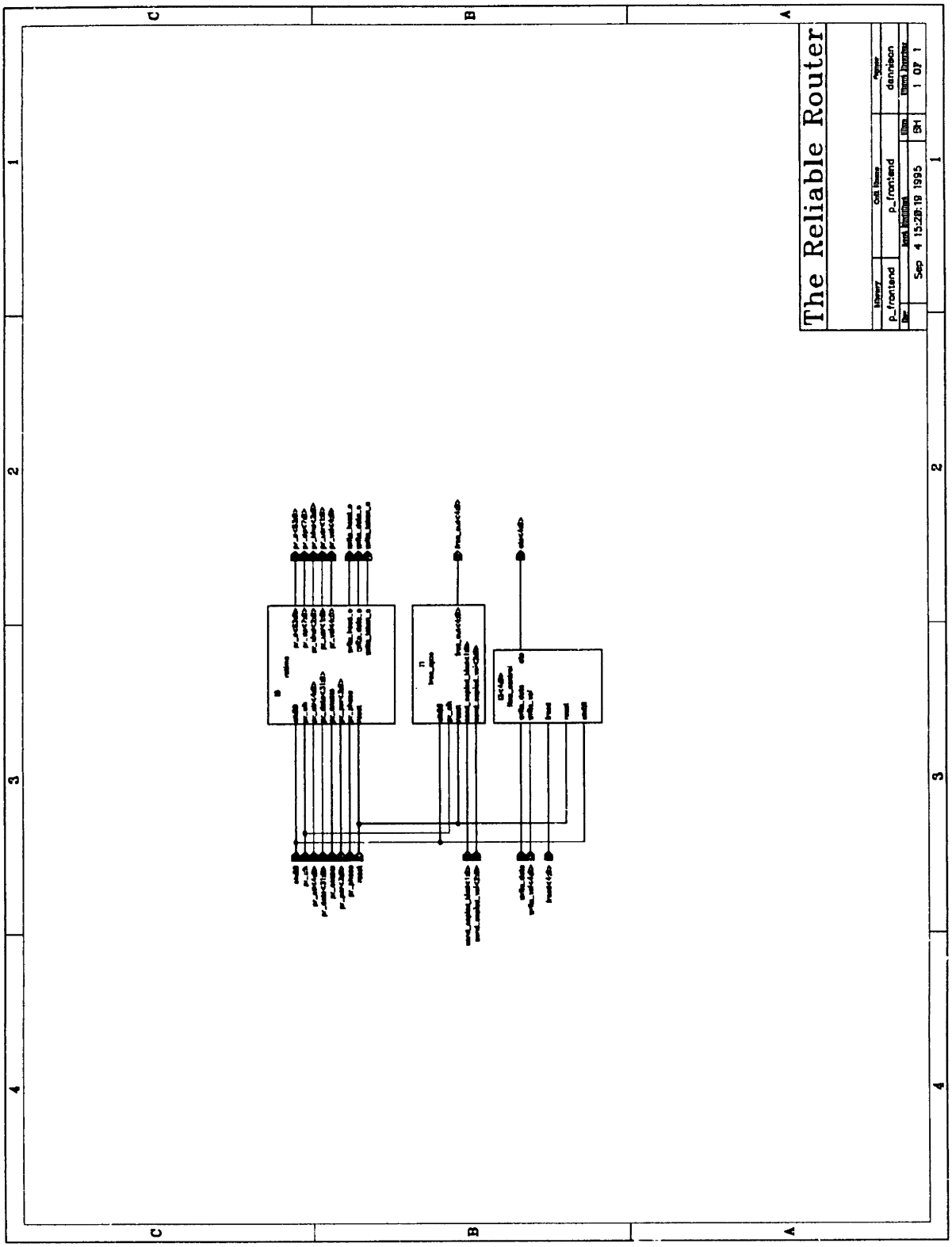


Figure A-153: Library p_frontend, cell p_frontend

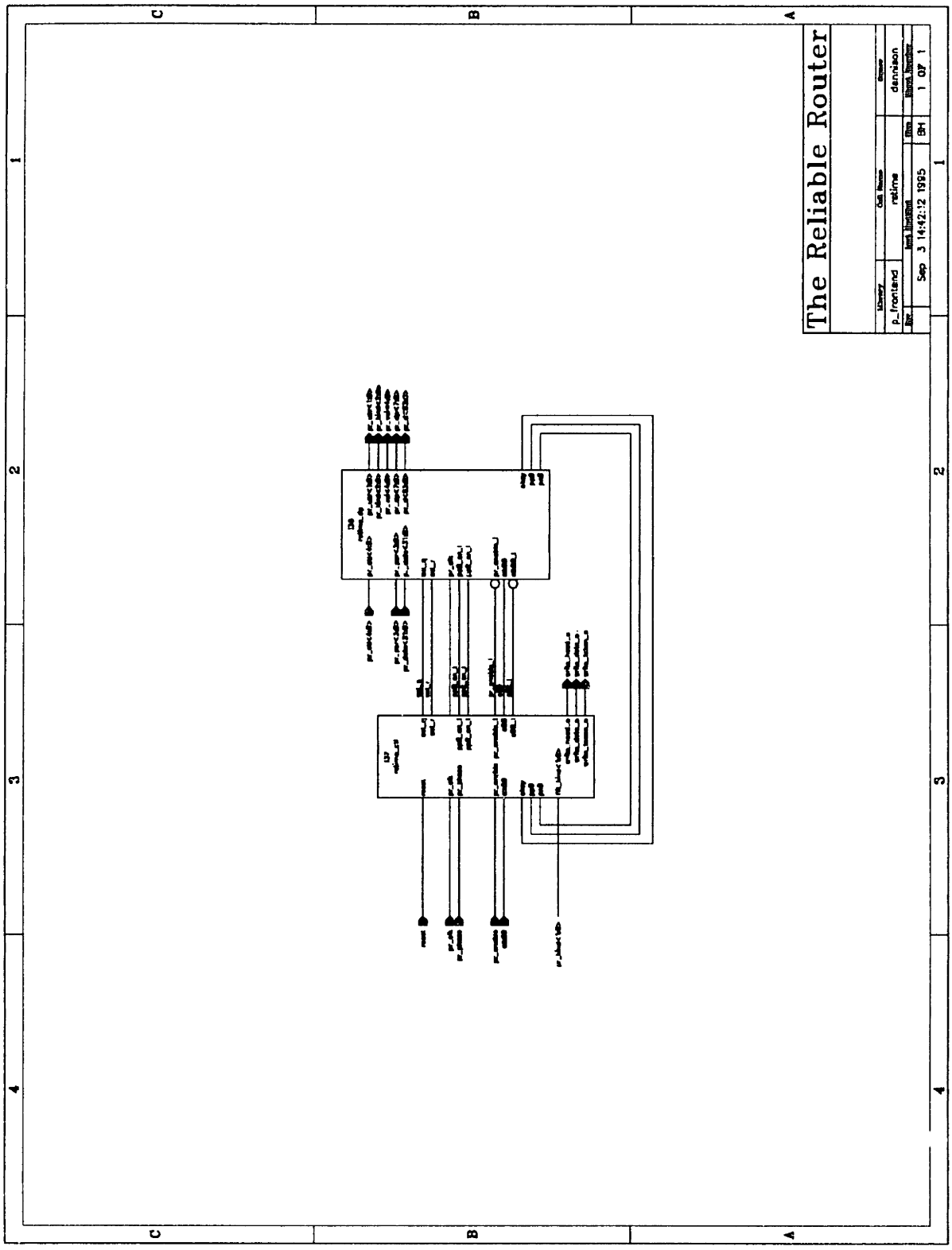


Figure A-154: Library p_frontend, cell retime

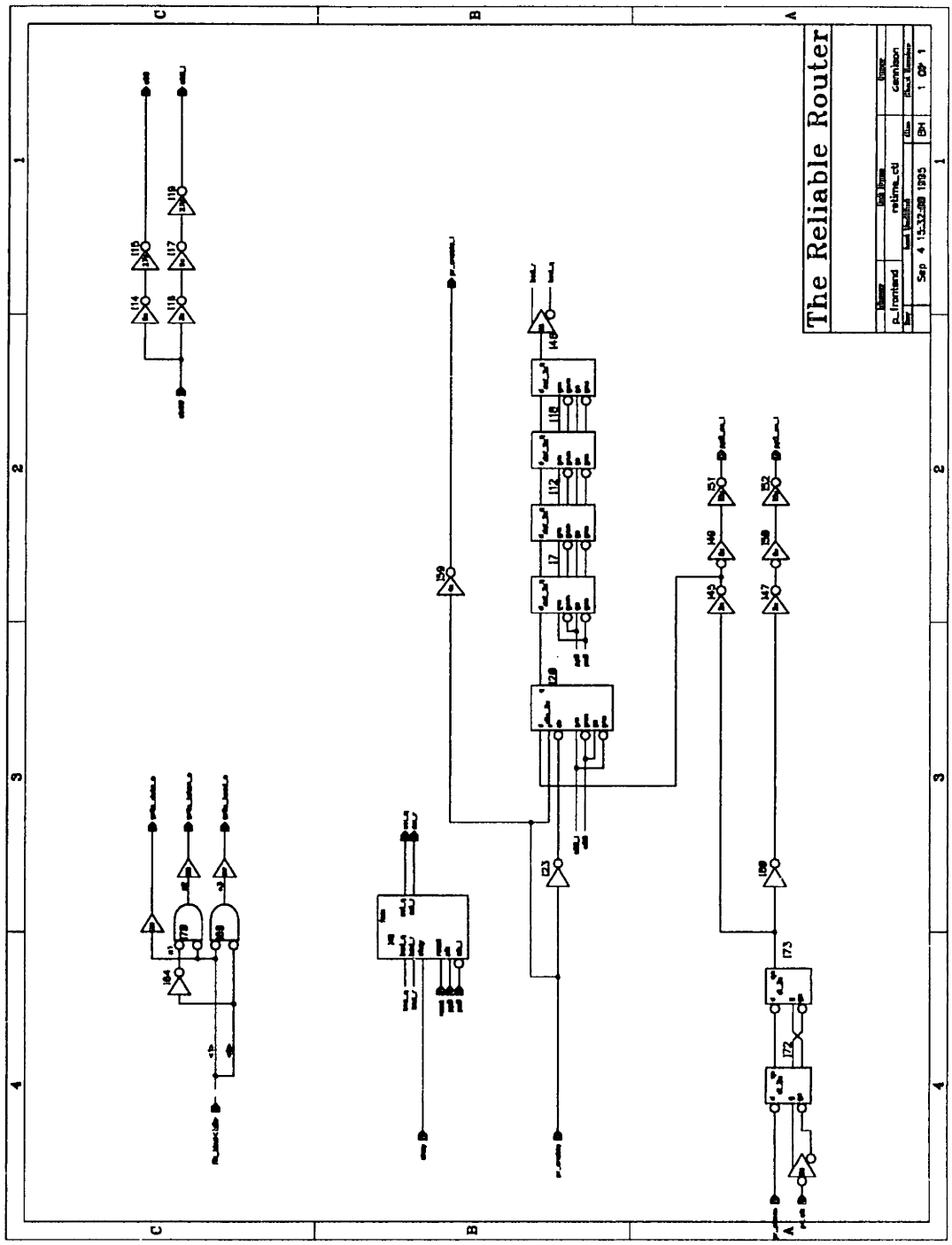


Figure A-155: Library p_frontend, cell retime_ctl

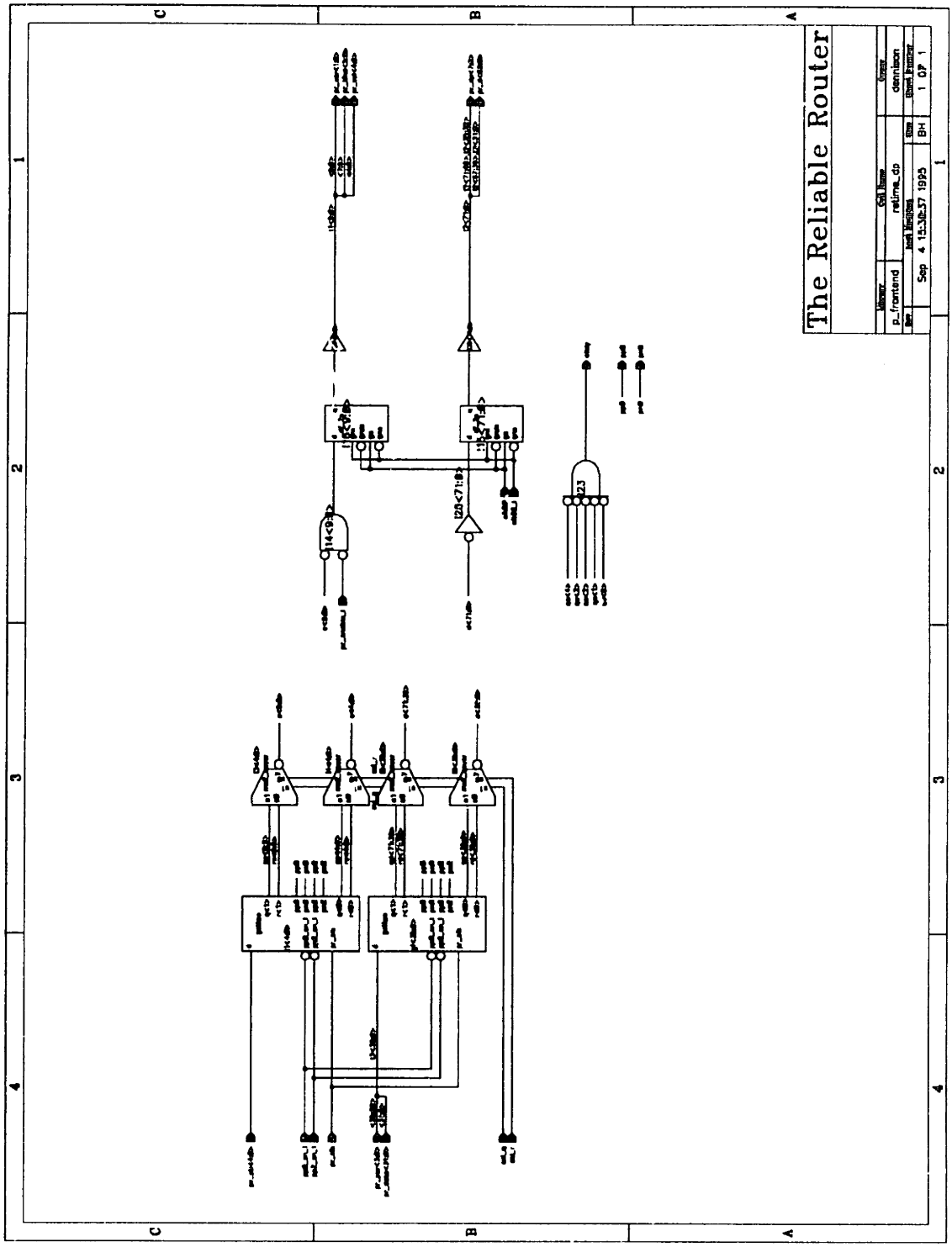


Figure A-156: Library p_frontend, cell retime_dp

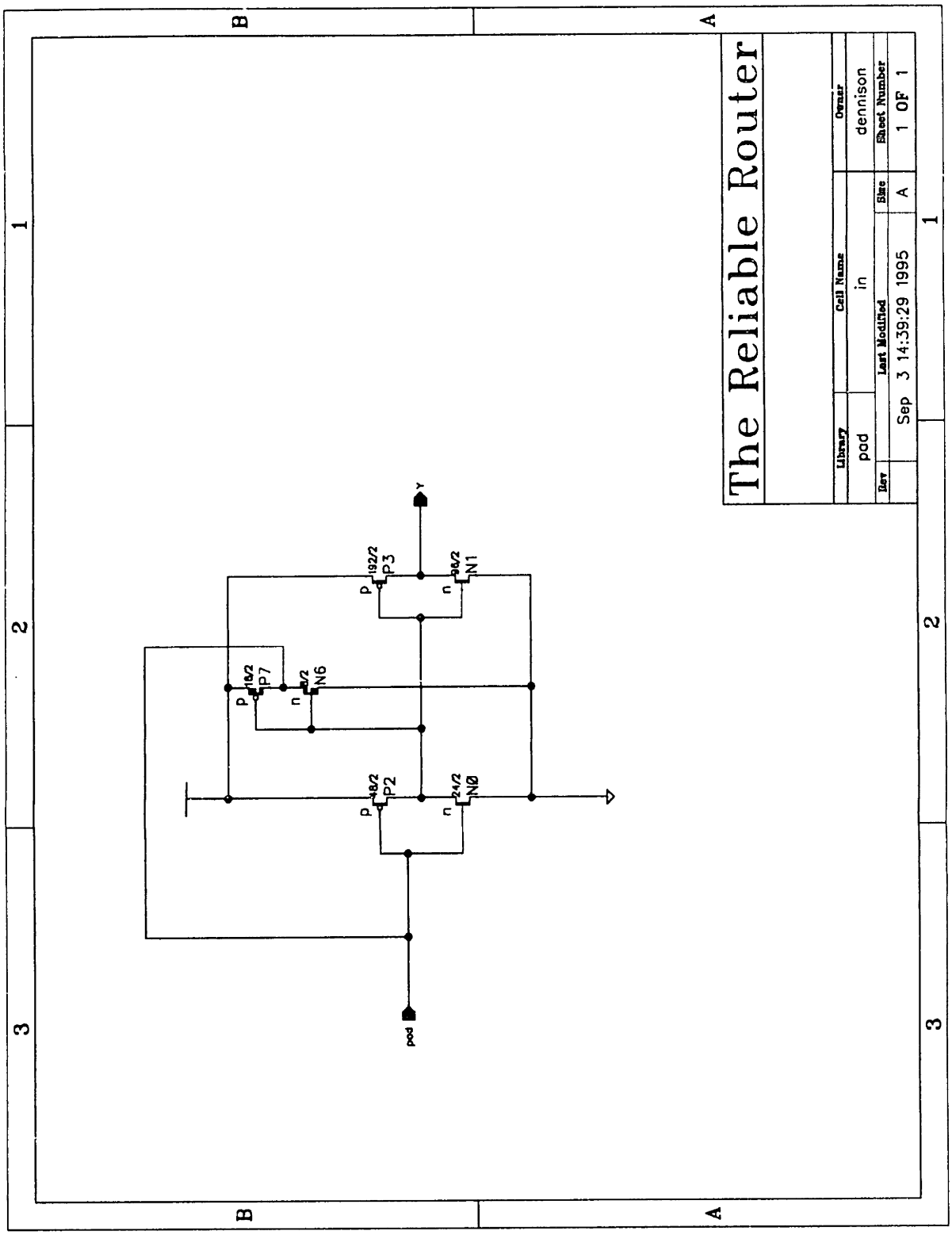


Figure A-157: Library pad, cell in

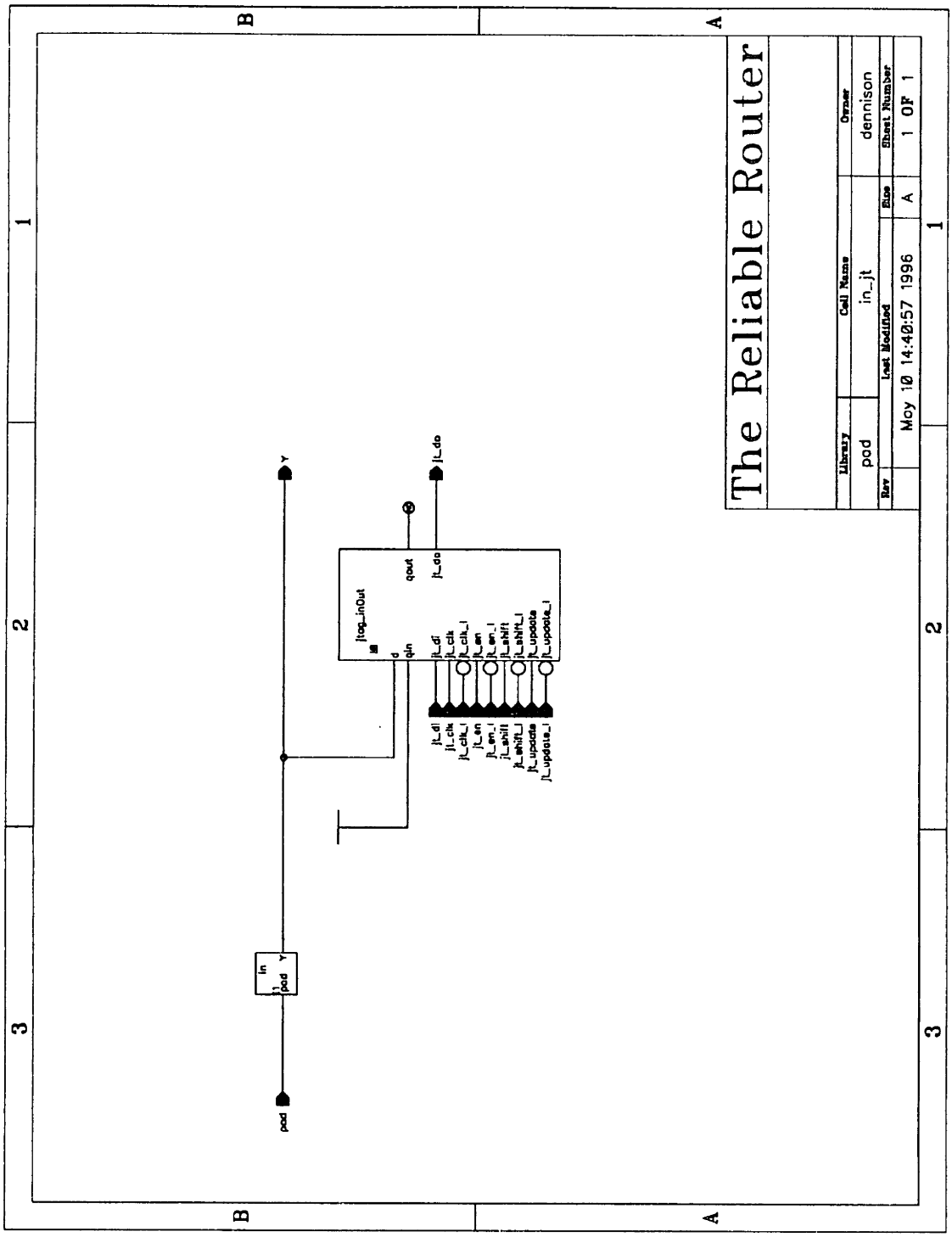


Figure A-158: Library pad, cell in_jt

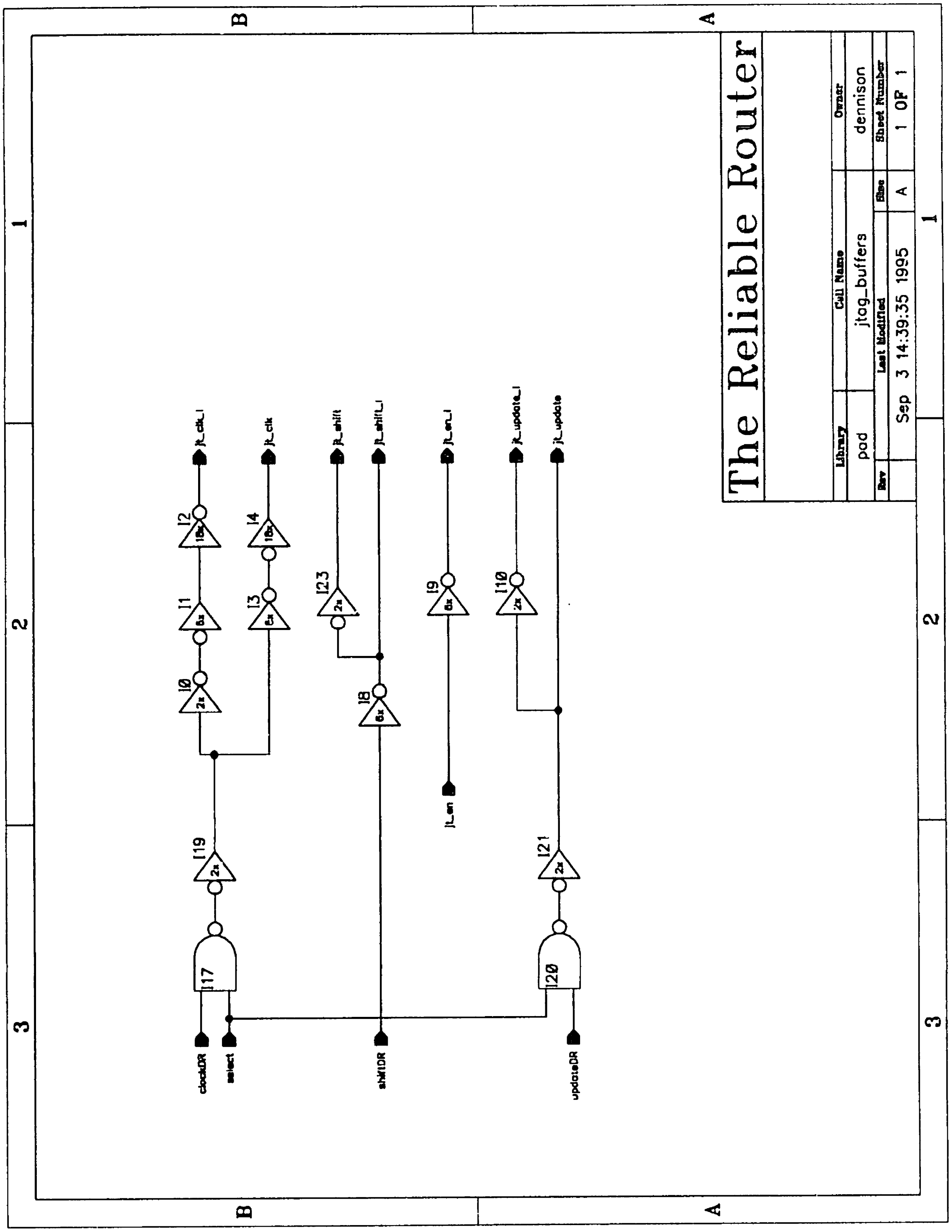


Figure A-159: Library pad, cell jtag_buffers

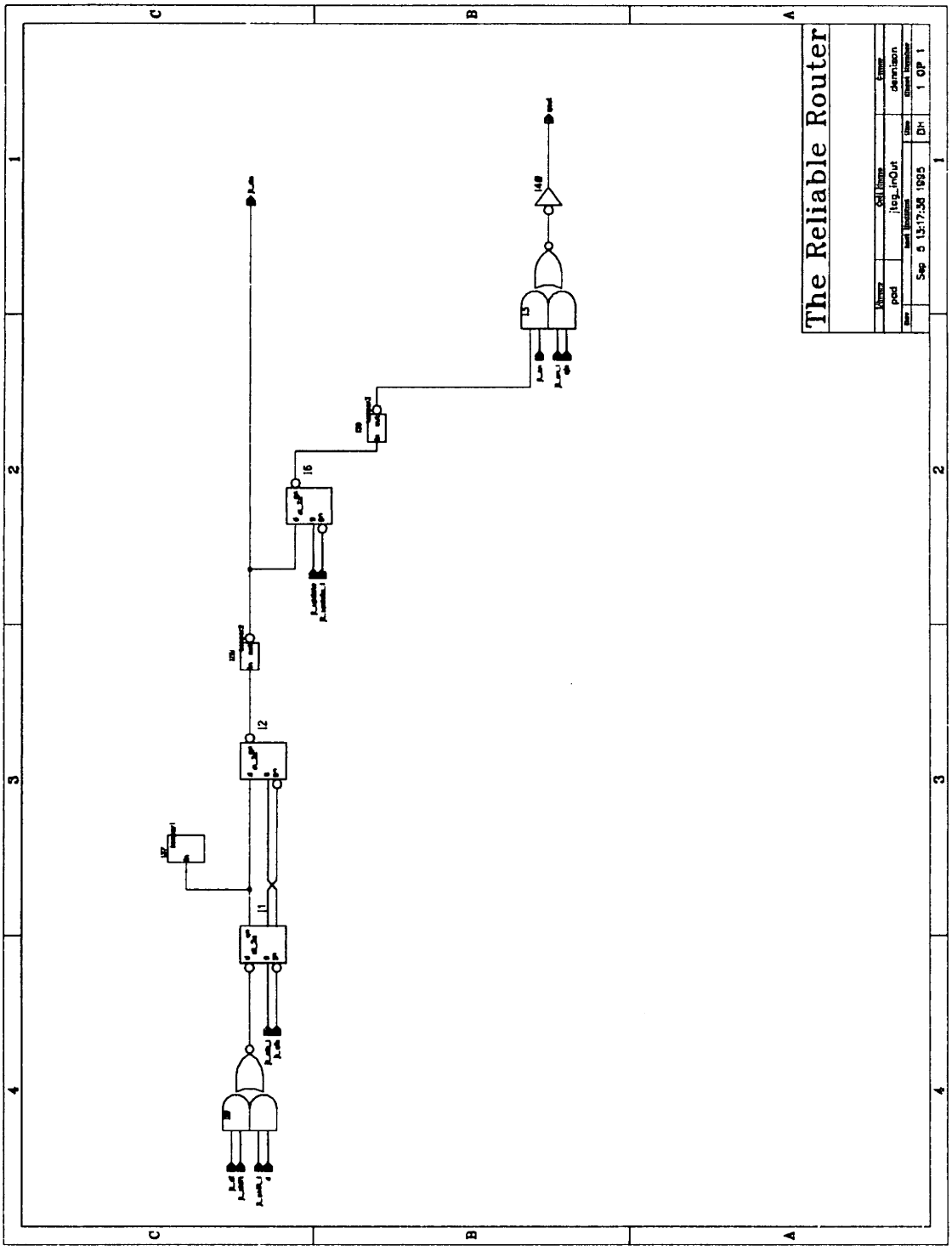


Figure A-160: Library pad, cell jtag_inOut

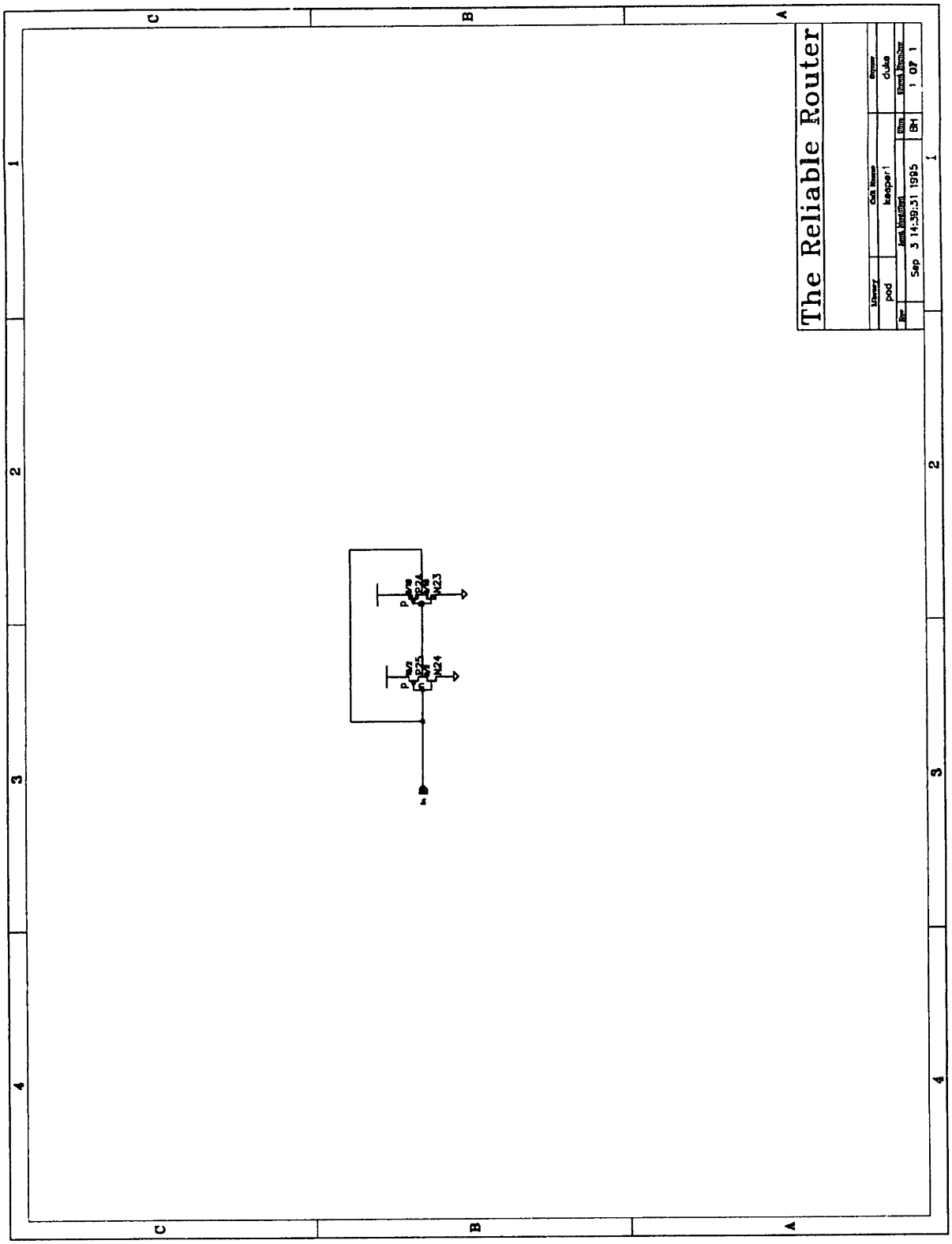


Figure A-161: Library pad, cell keeper1

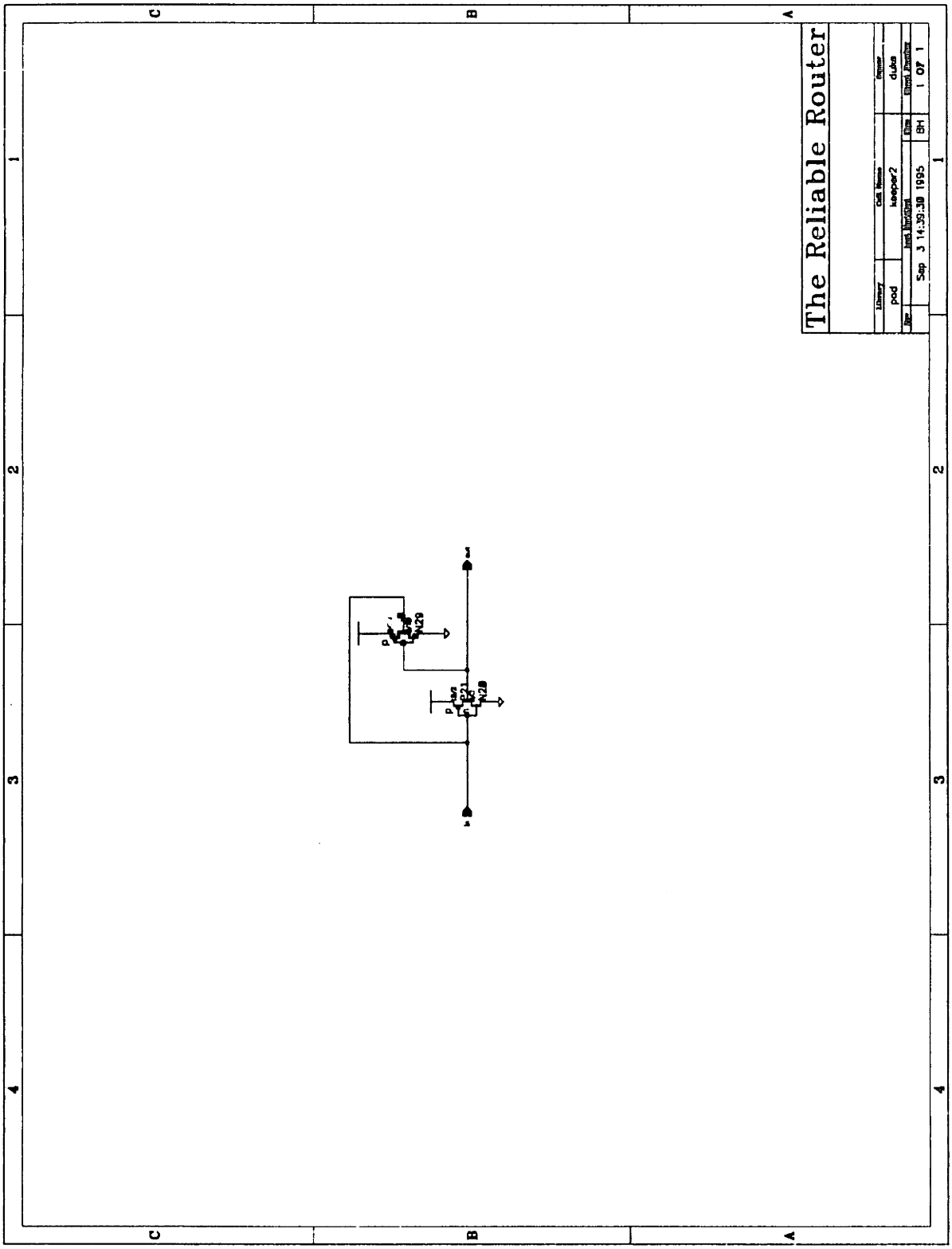


Figure A-162: Library pad, cell keeper2

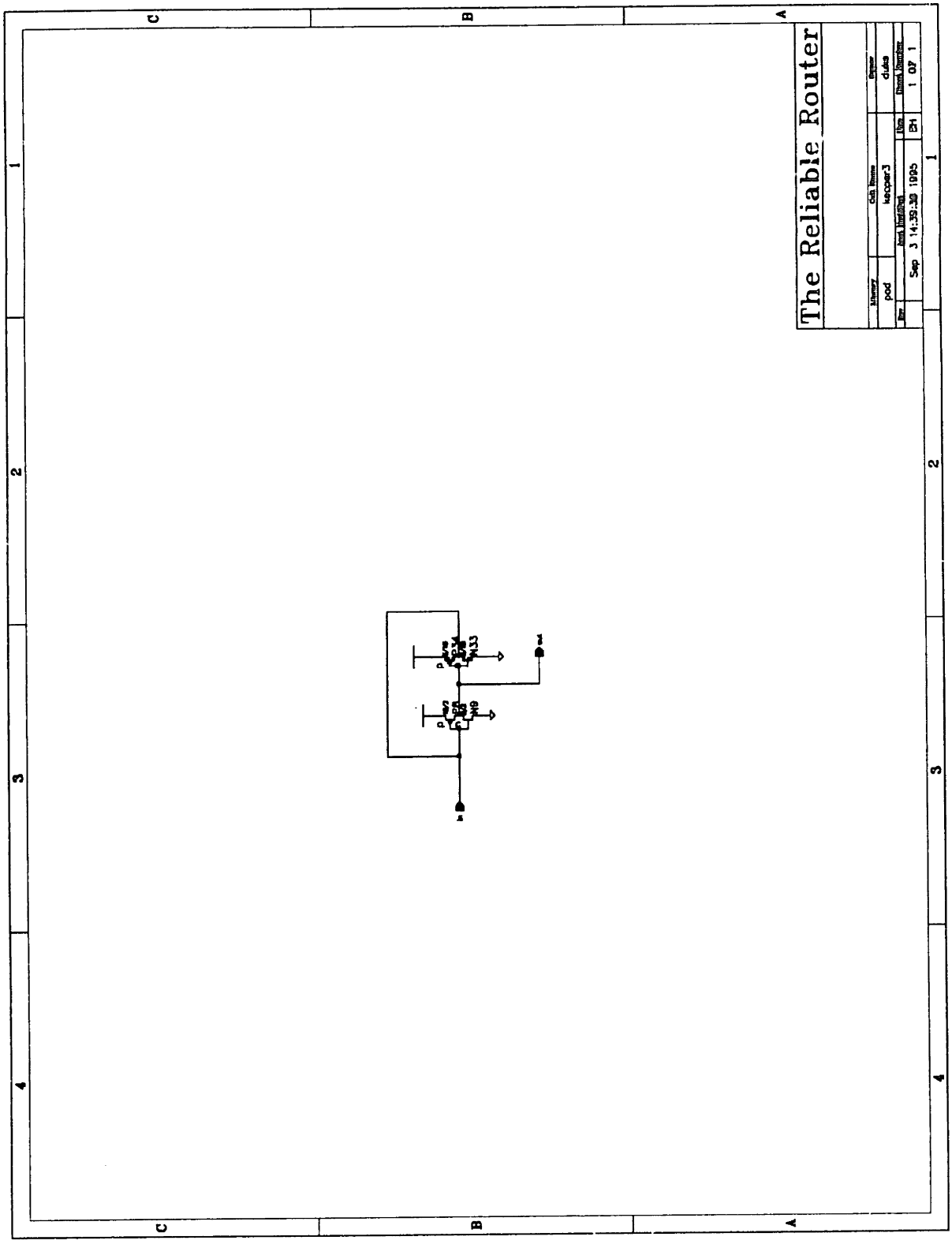


Figure A-163: Library pad, cell keeper3

Figure A-164: Library pad, cell leftSide

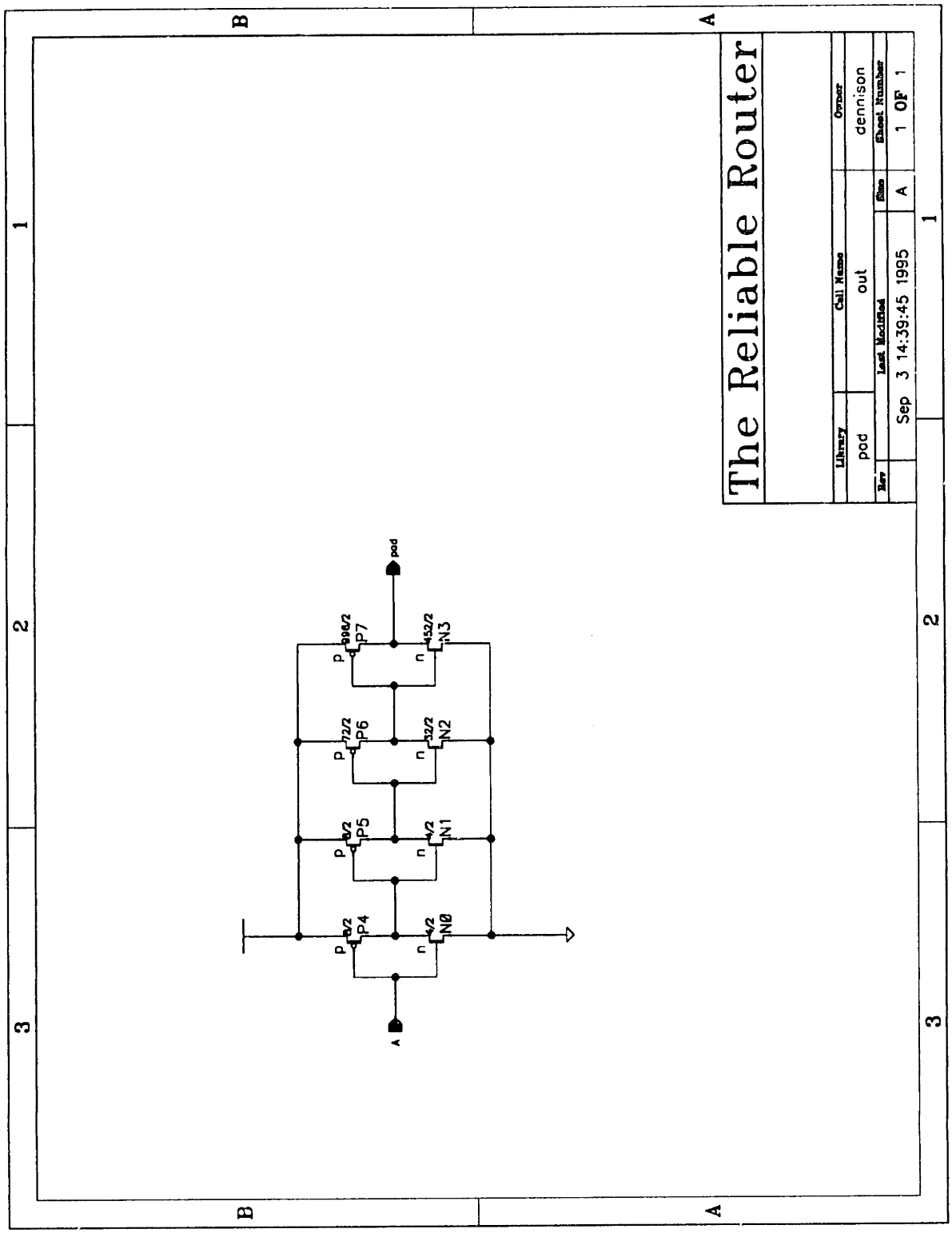


Figure A-165: Library pad, cell out

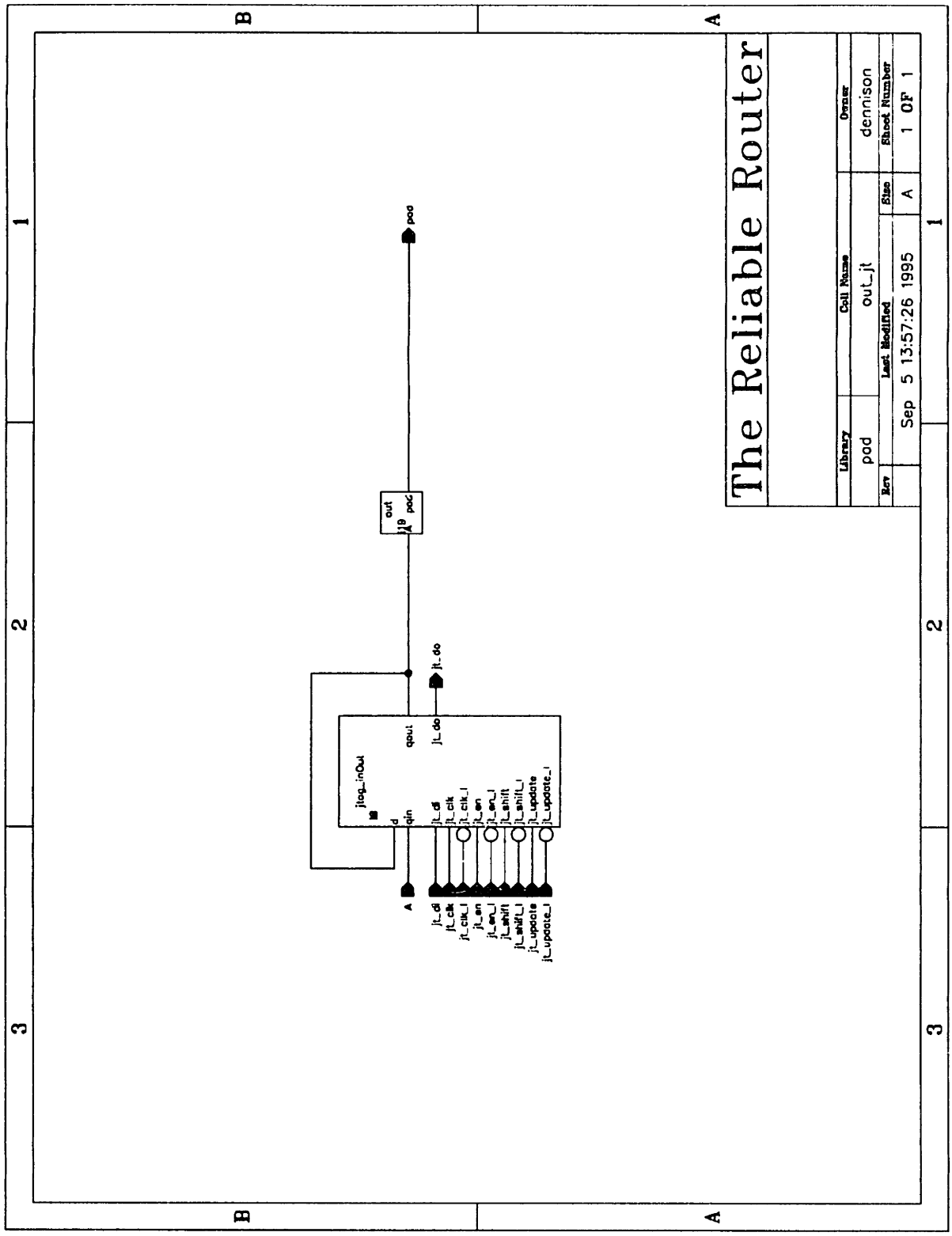


Figure A-166: Library pad, cell out_jt

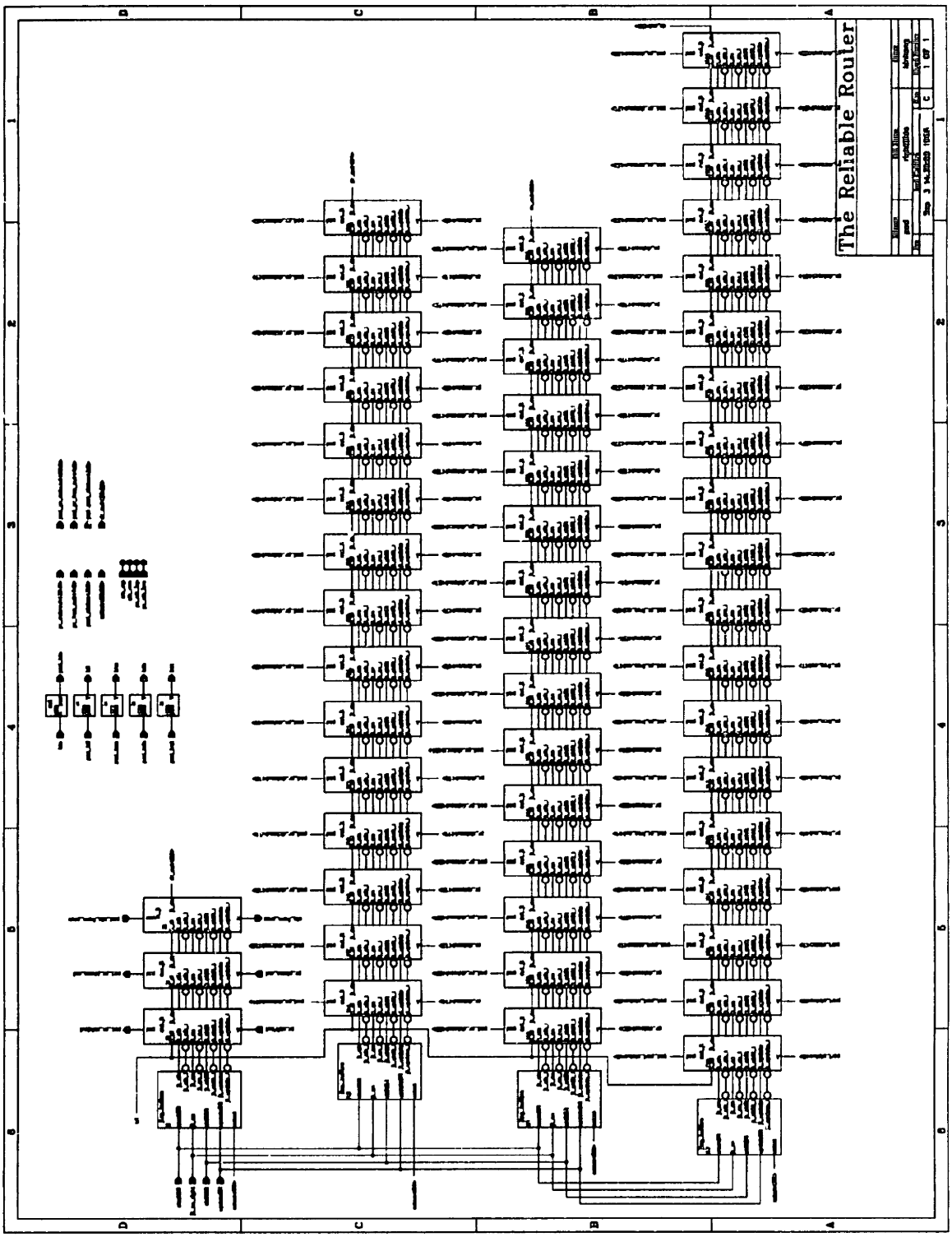


Figure A-167: Library pad, cell rightSide

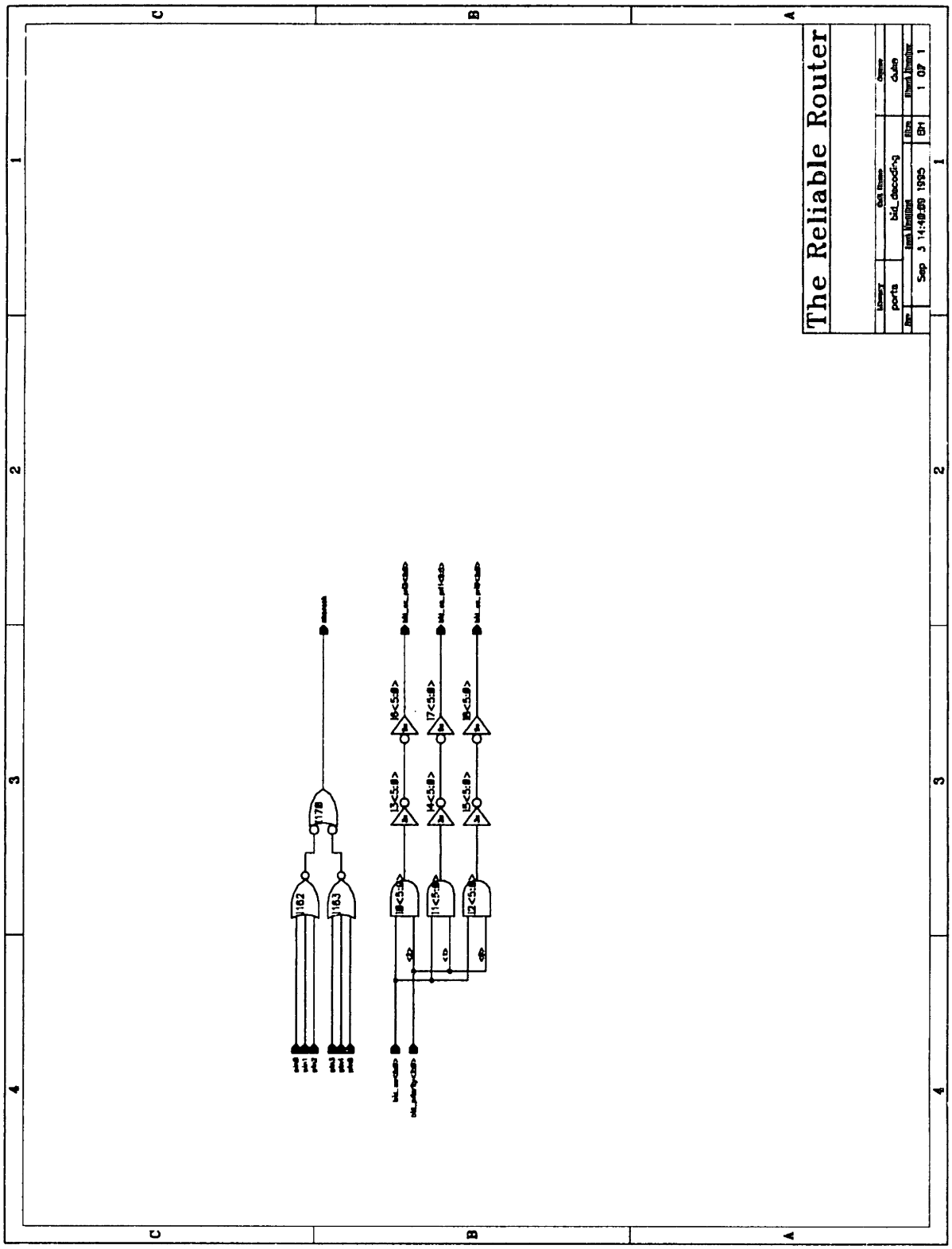


Figure A-168: Library ports, cell bid_decoding

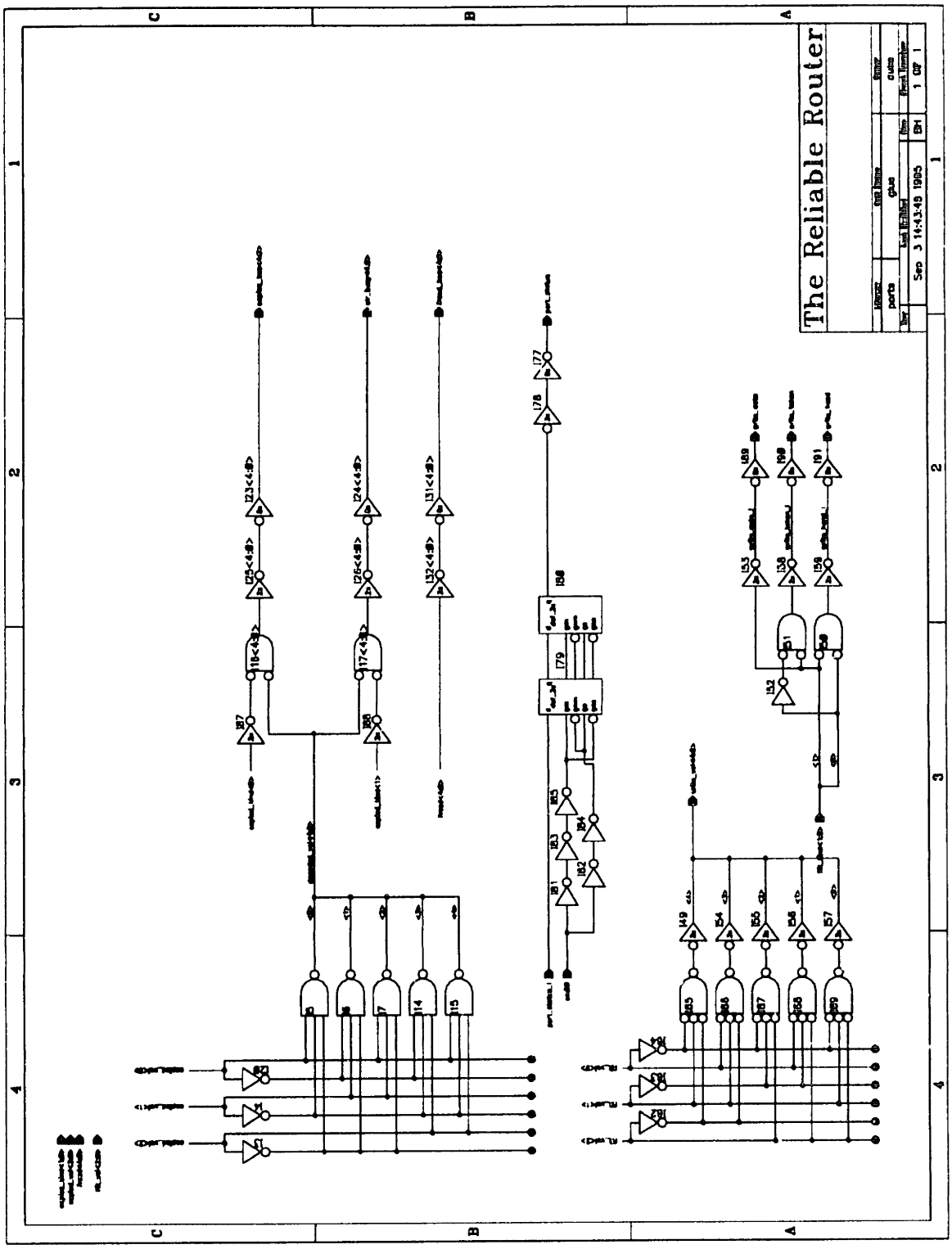


Figure A-169: Library ports, cell glue

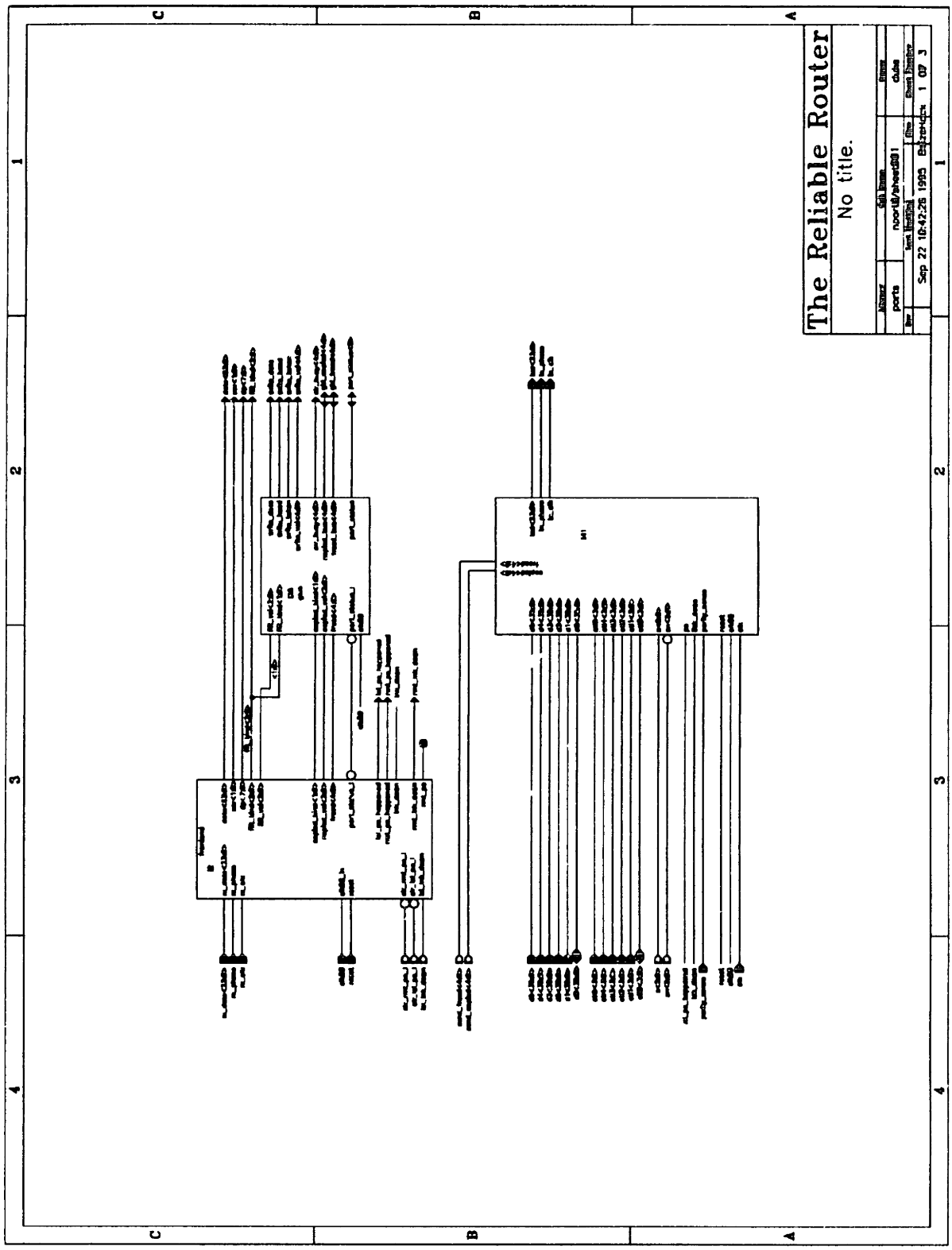


Figure A-170: Library ports, cell nport0 (sheet 1)

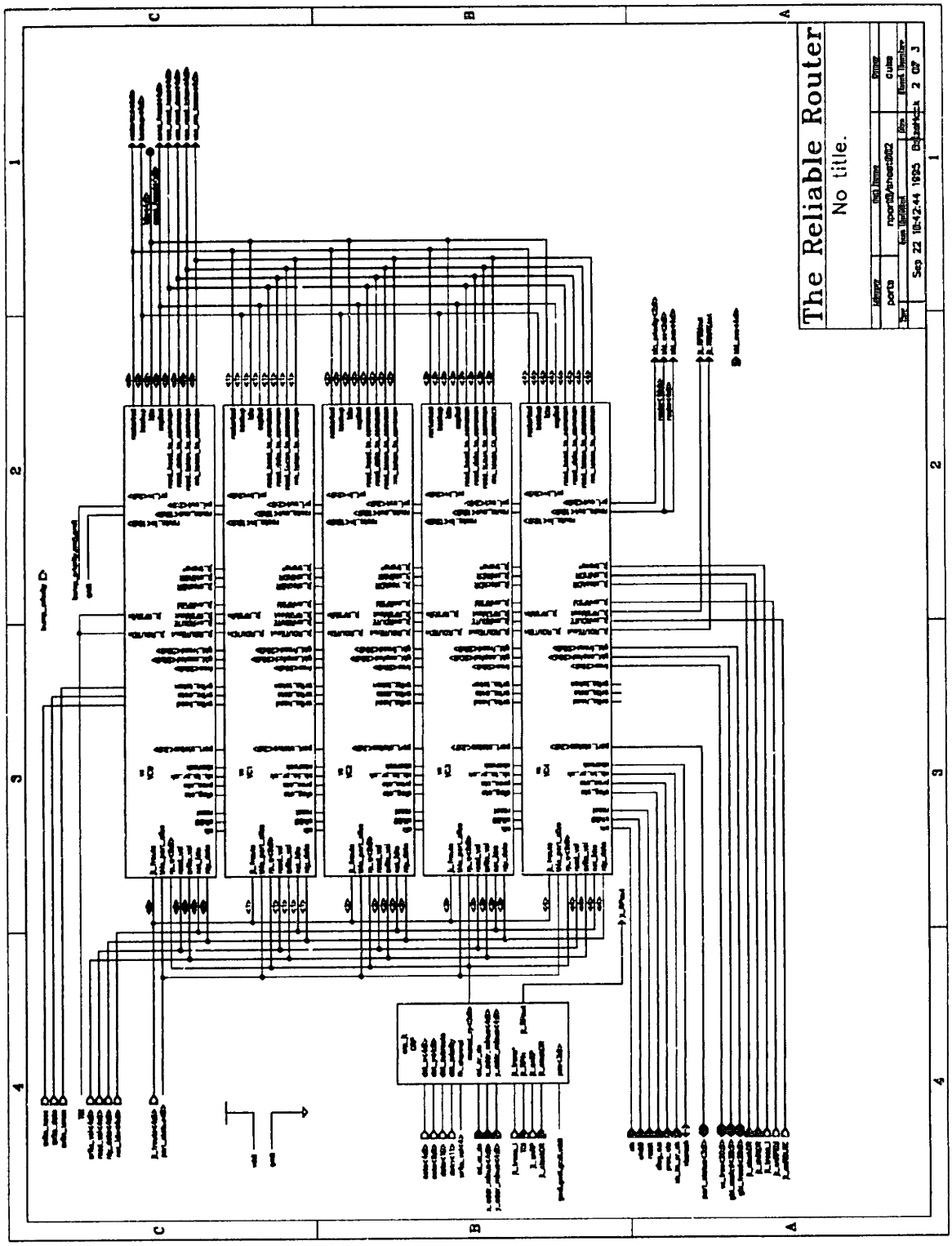


Figure A-171: Library ports, cell nport0 (sheet 2)

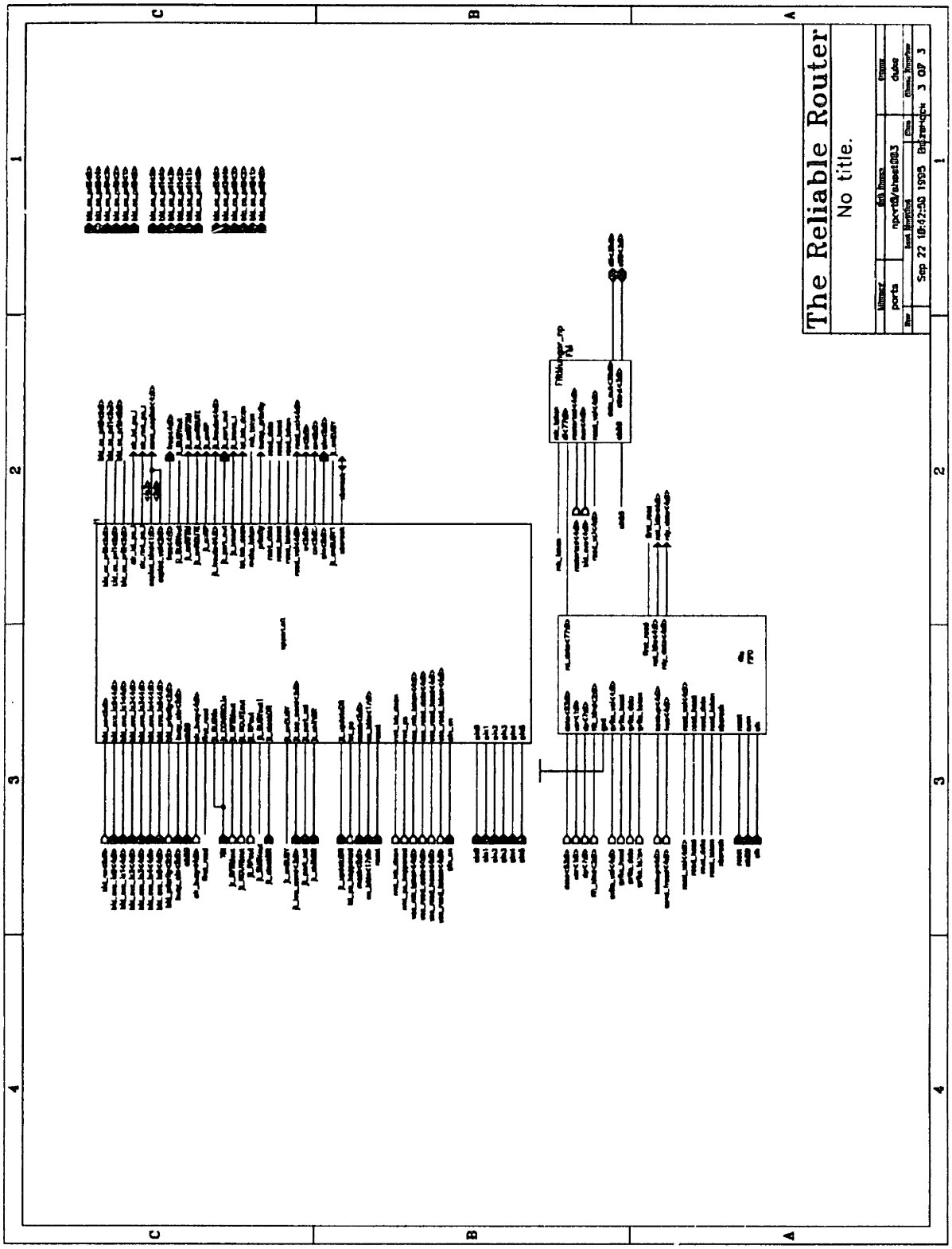


Figure A-172: Library ports, cell nport0 (sheet 3)

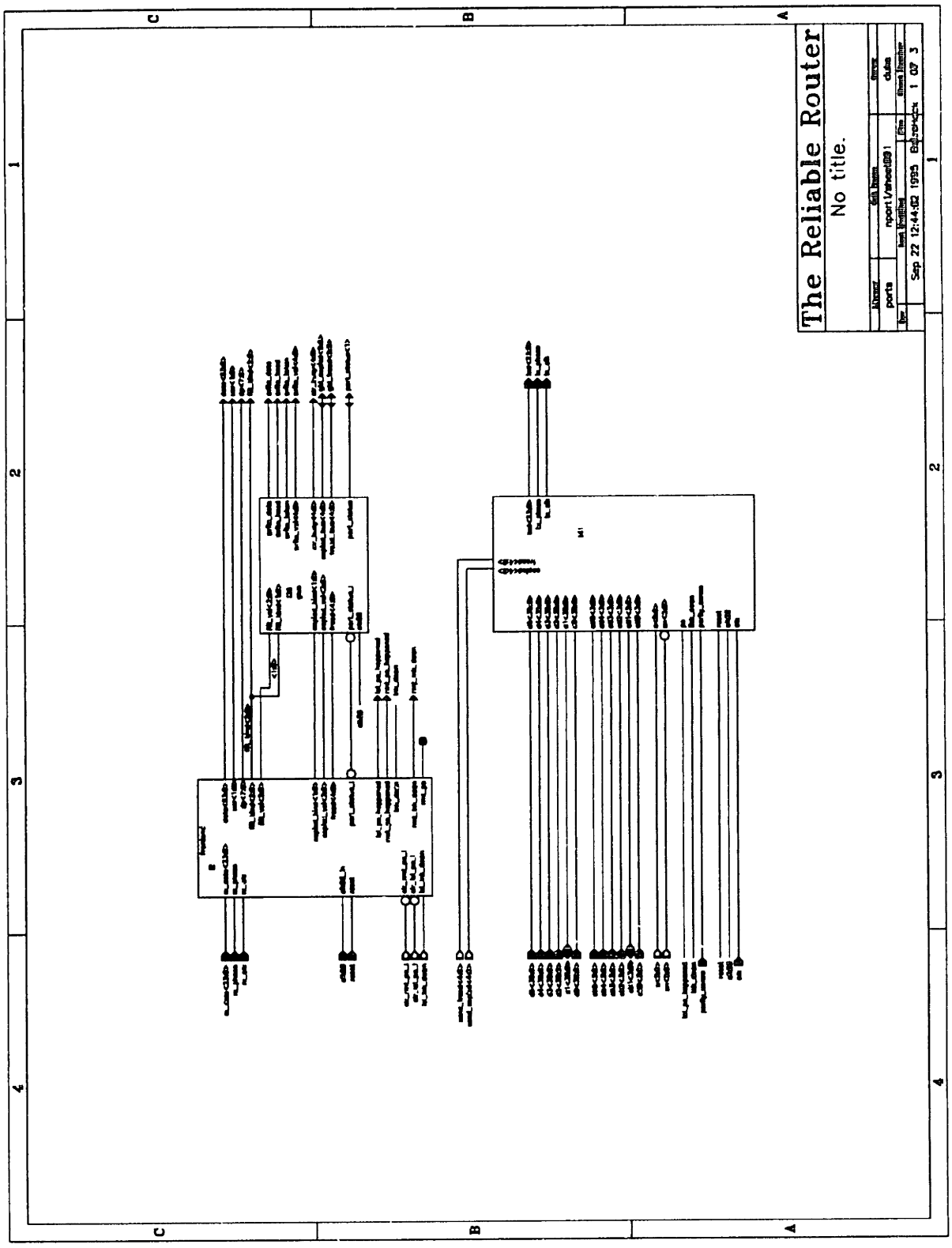


Figure A-173: Library ports, cell nport1 (sheet 1)

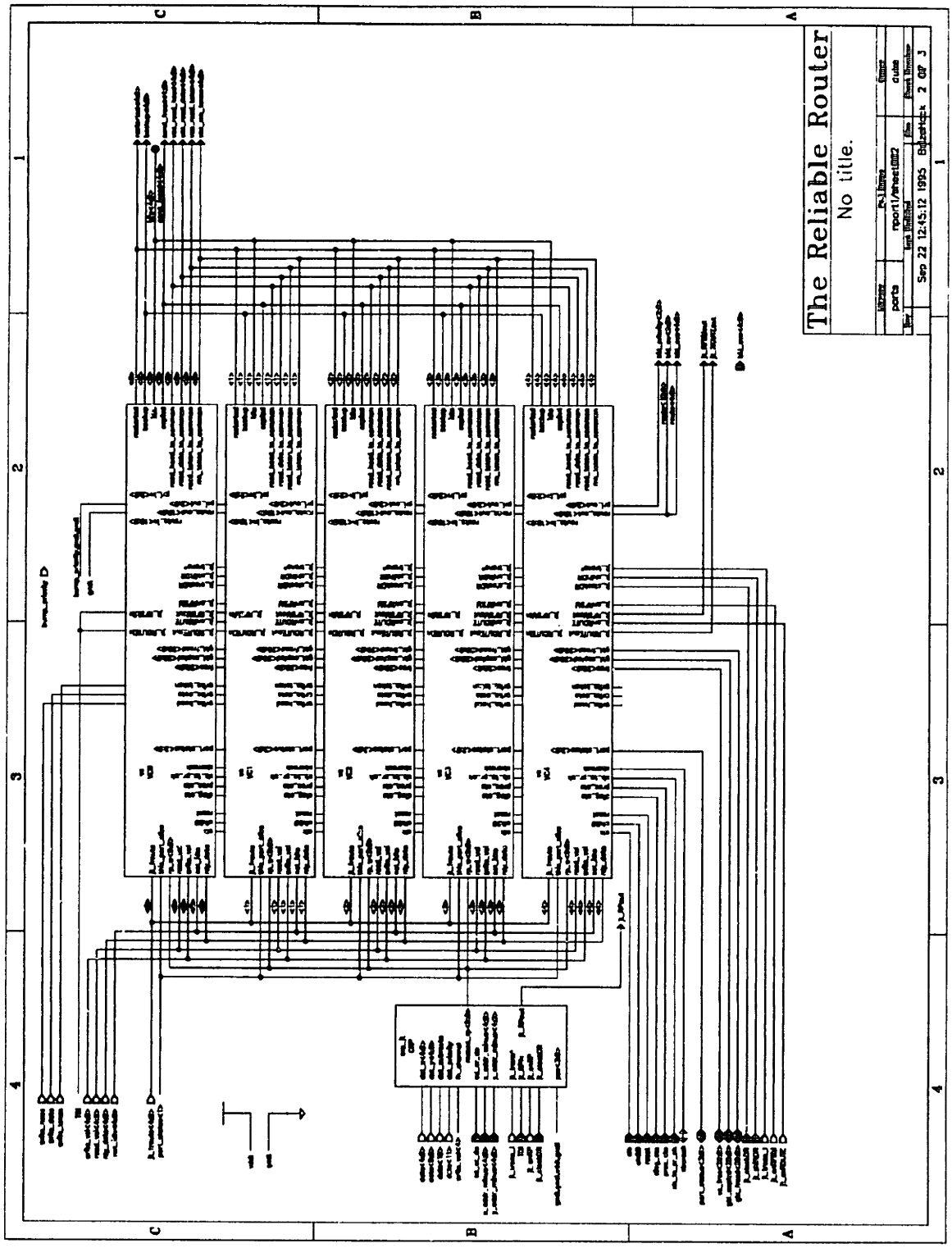


Figure A-174: Library ports, cell nport1 (sheet 2)

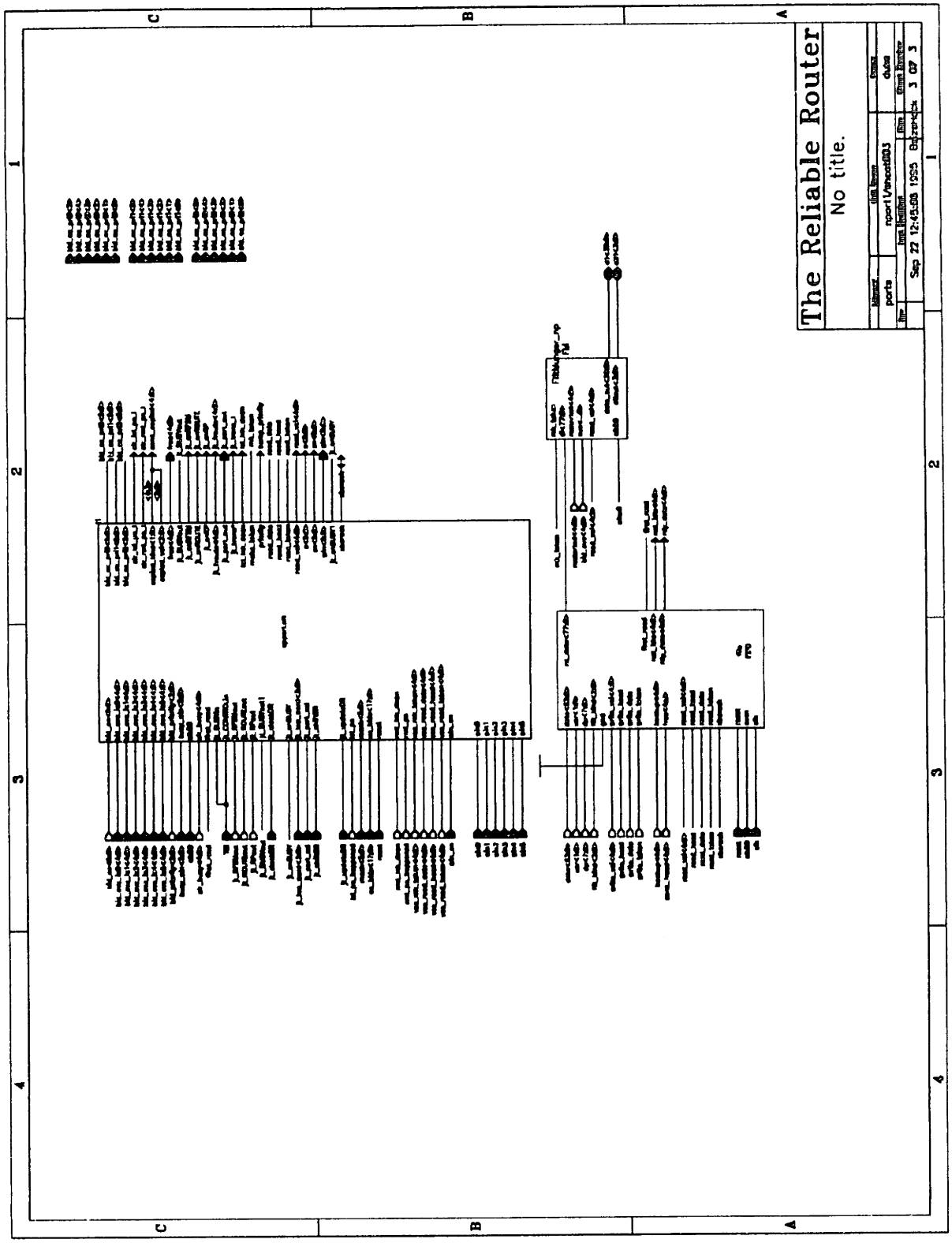


Figure A-175: Library ports, cell nport1 (sheet 3)

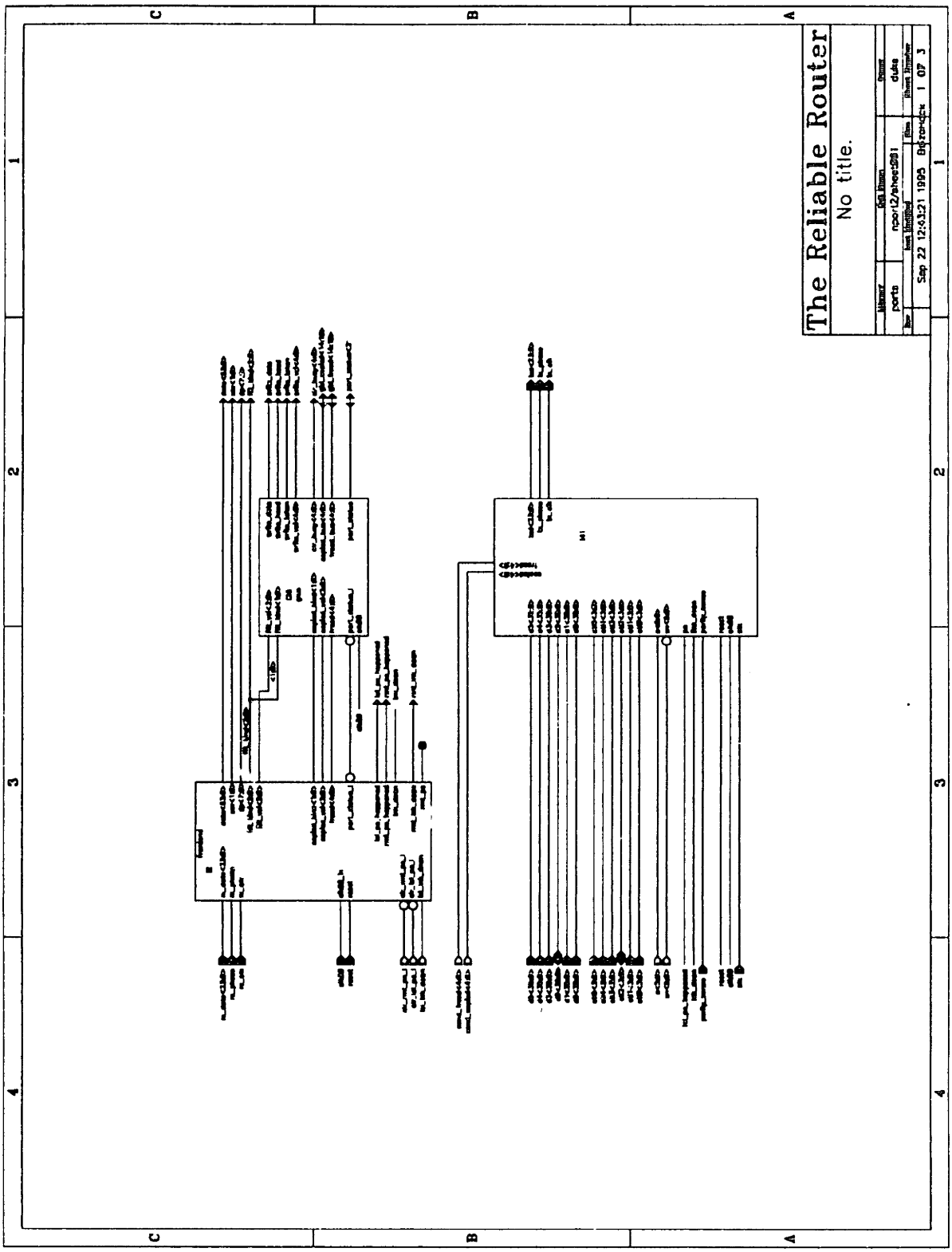


Figure A-176: Library ports, cell nport2 (sheet 1)

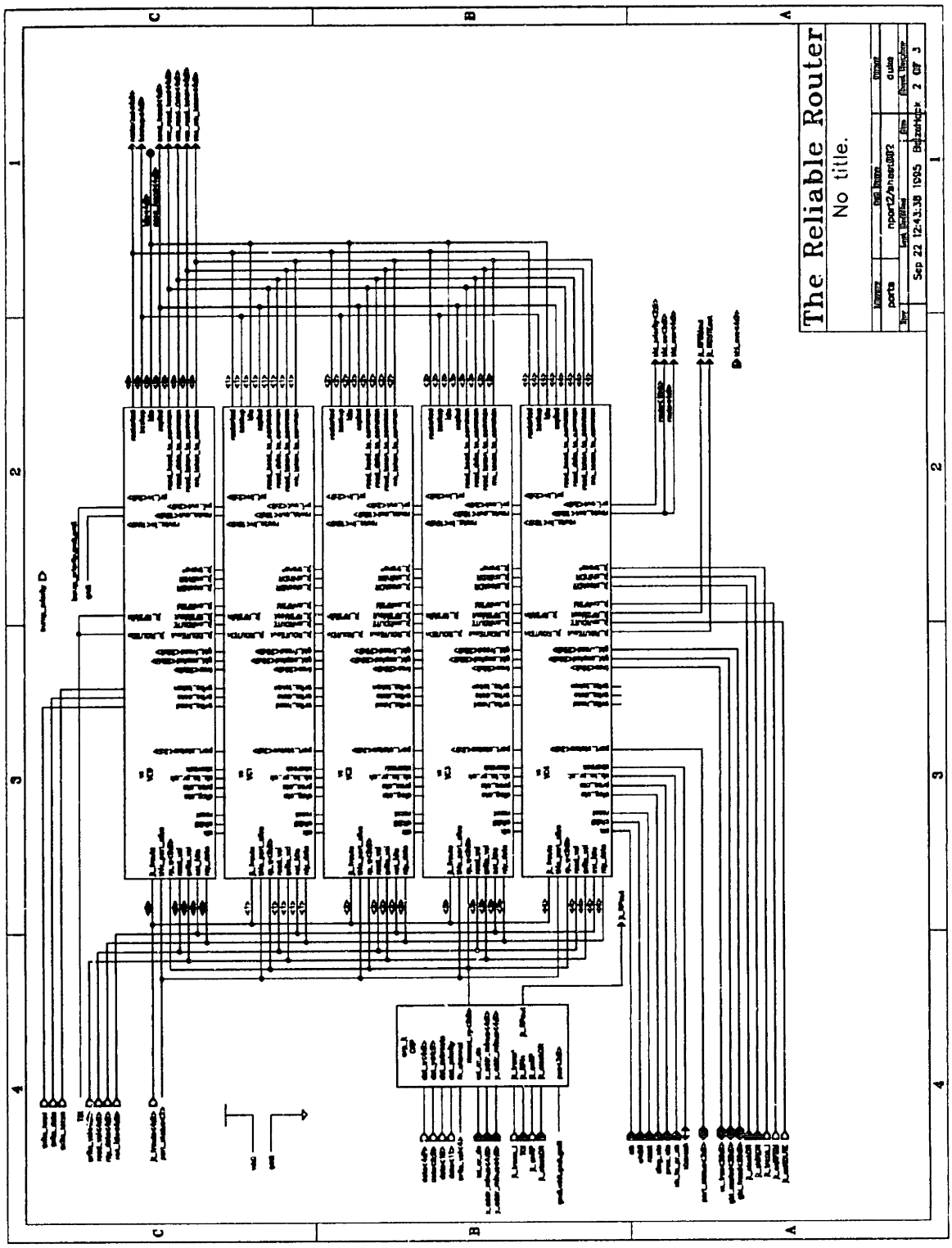


Figure A-177: Library ports, cell nport2 (sheet 2)

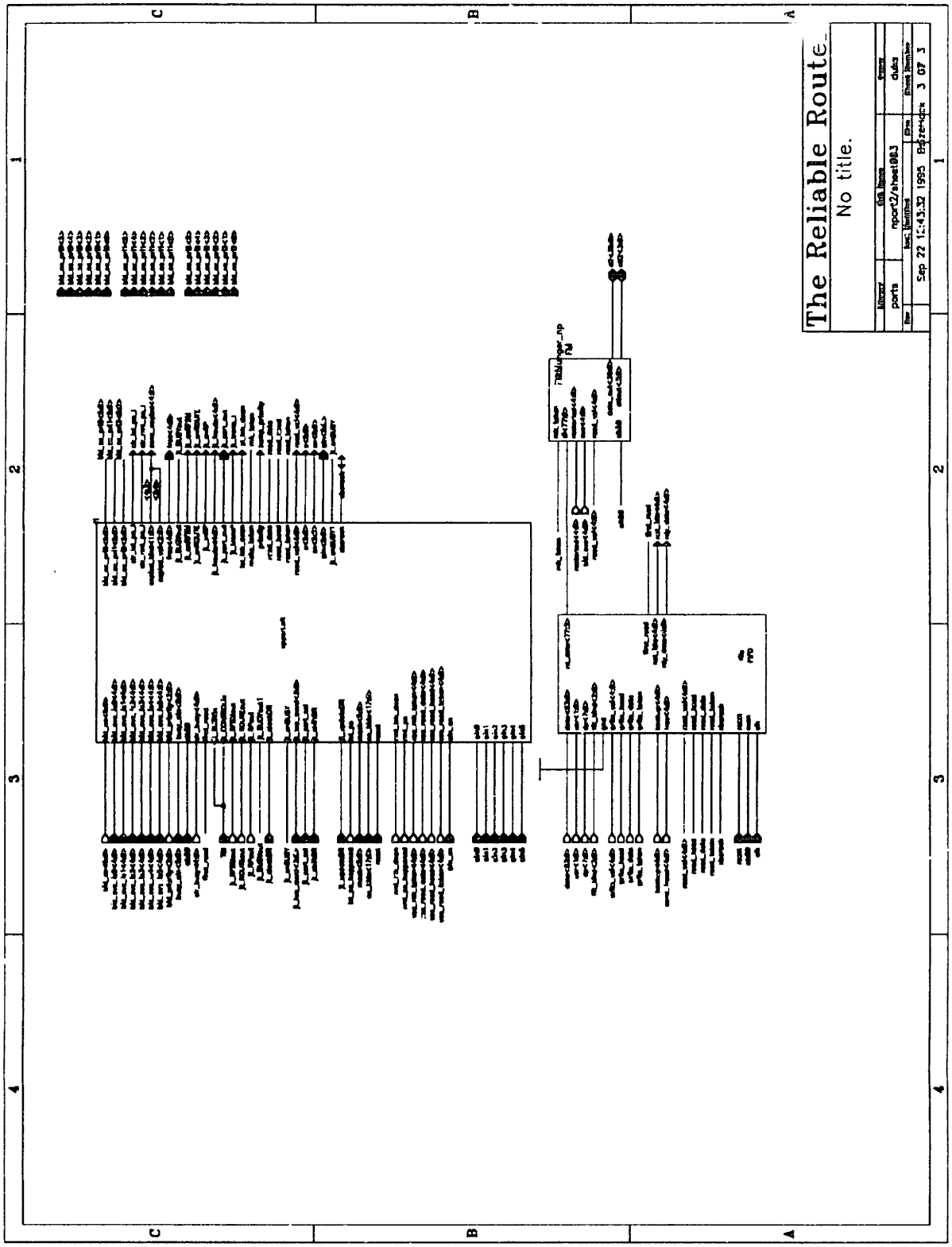


Figure A-178: Library ports, cell nport2 (sheet 3)

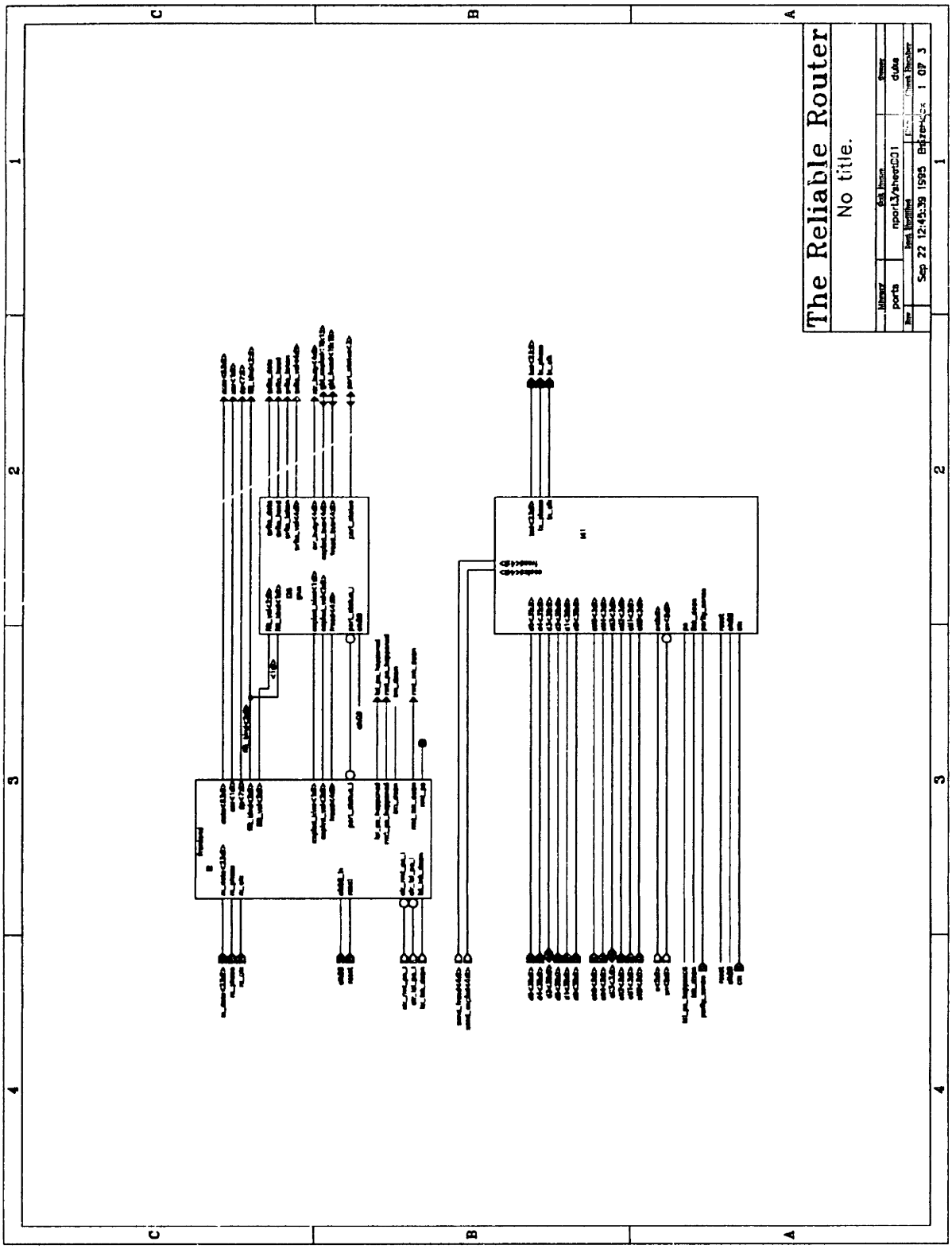


Figure A-179: Library ports, cell nport3 (sheet 1)

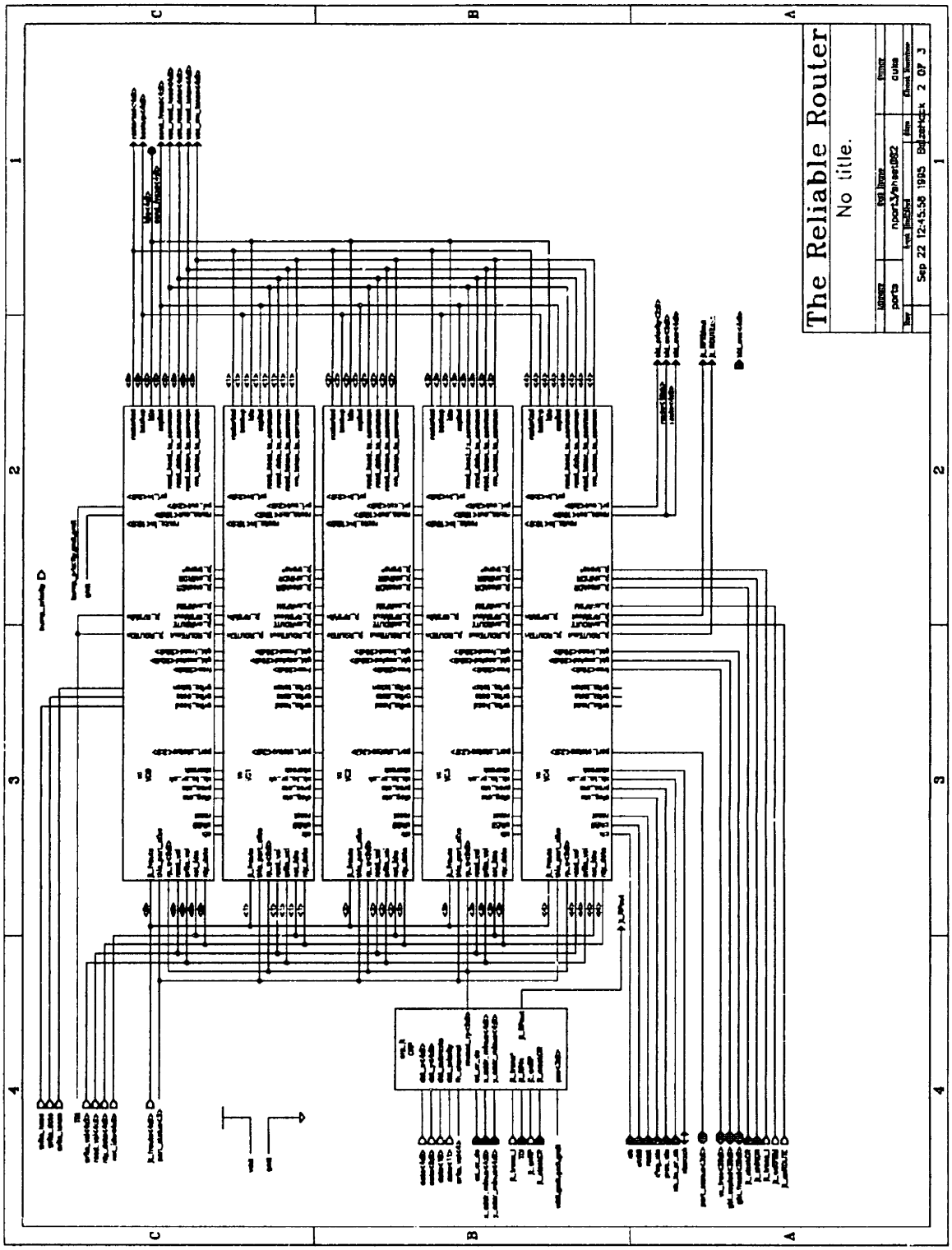


Figure A-180: Library ports, cell nport3 (sheet 2)

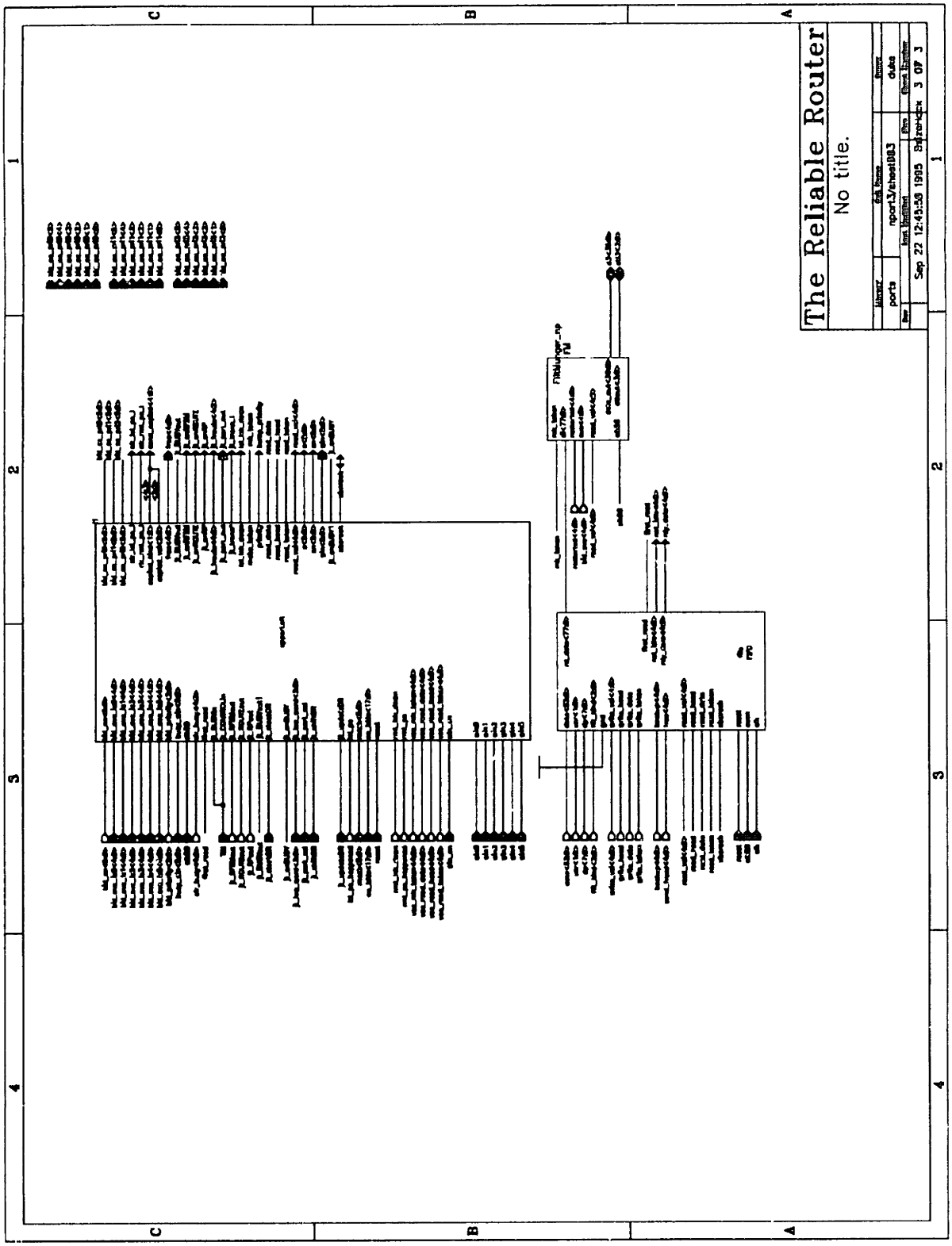


Figure A-181: Library ports, cell nport3 (sheet 3)

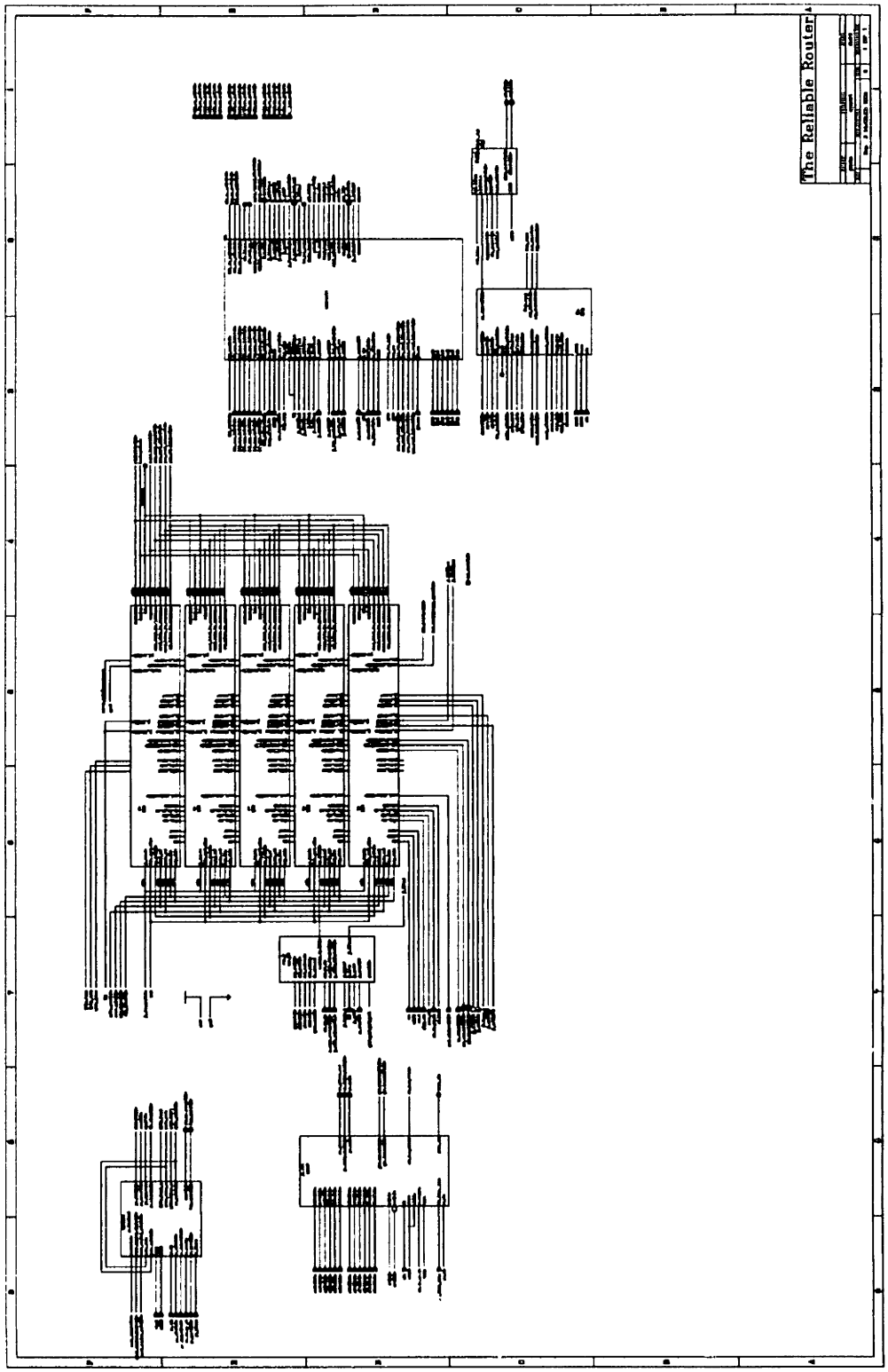


Figure A-182: Library ports, cell nport4

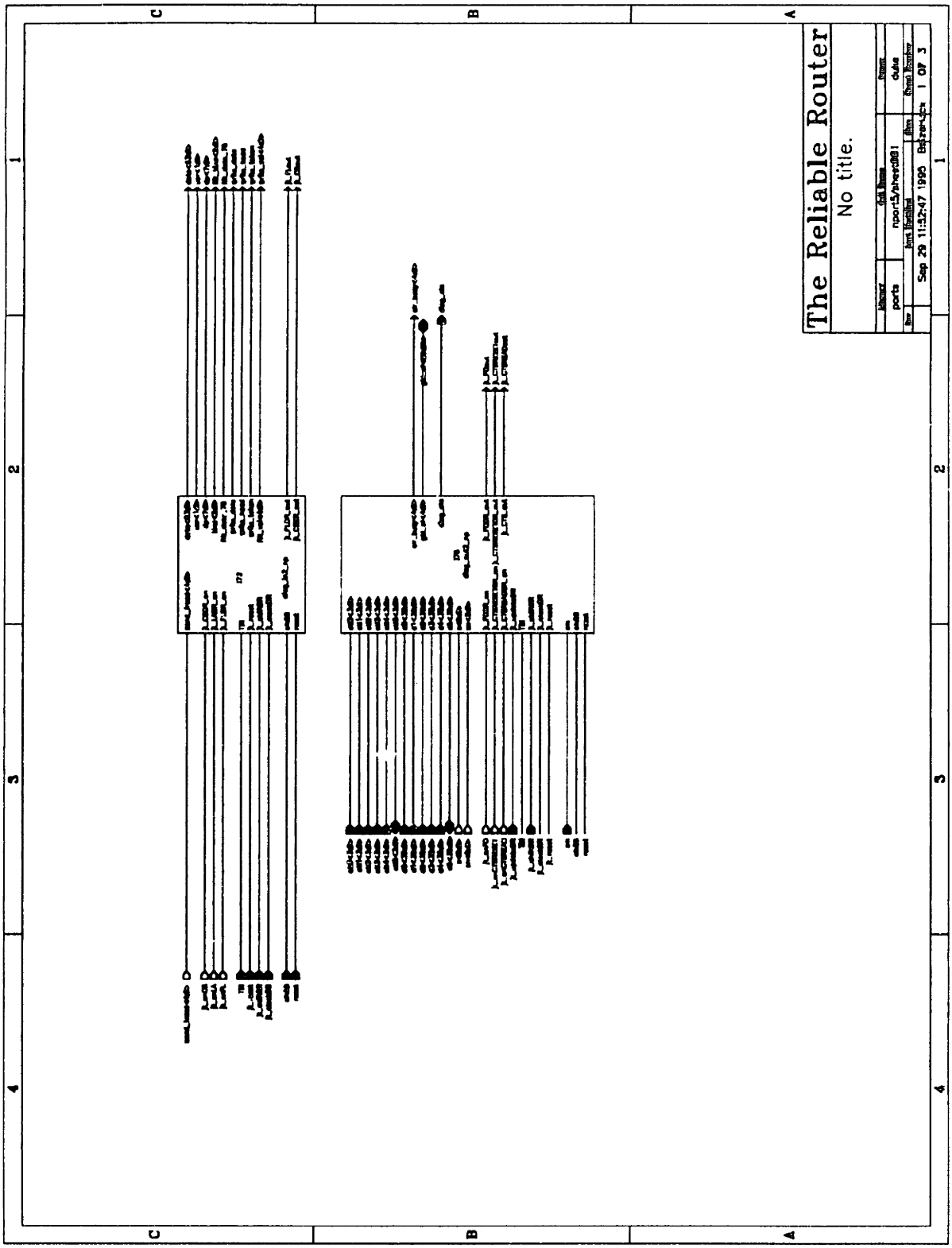


Figure A-183: Library ports, cell nport5 (sheet 1)

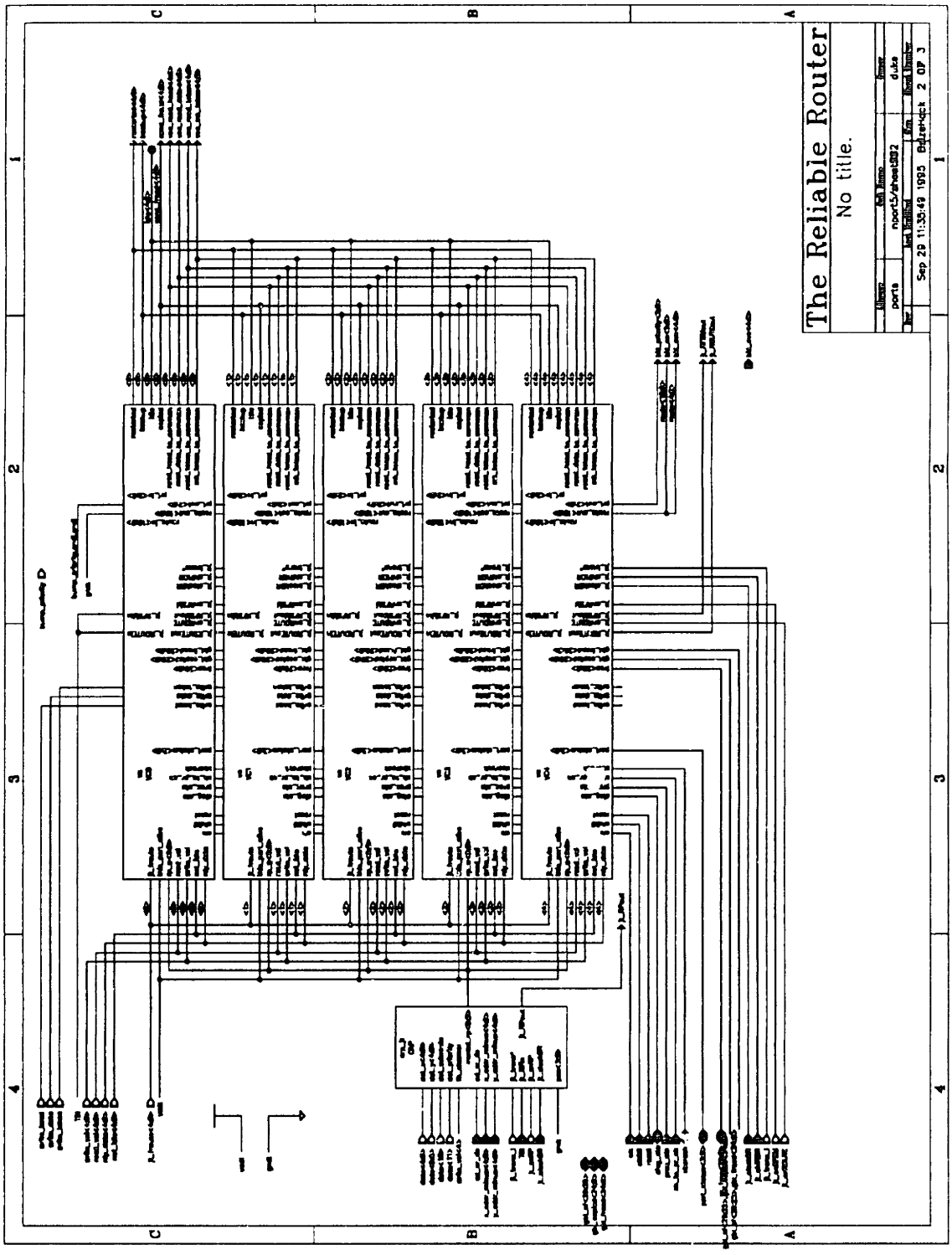


Figure A-184: Library ports, cell nport5 (sheet 2)

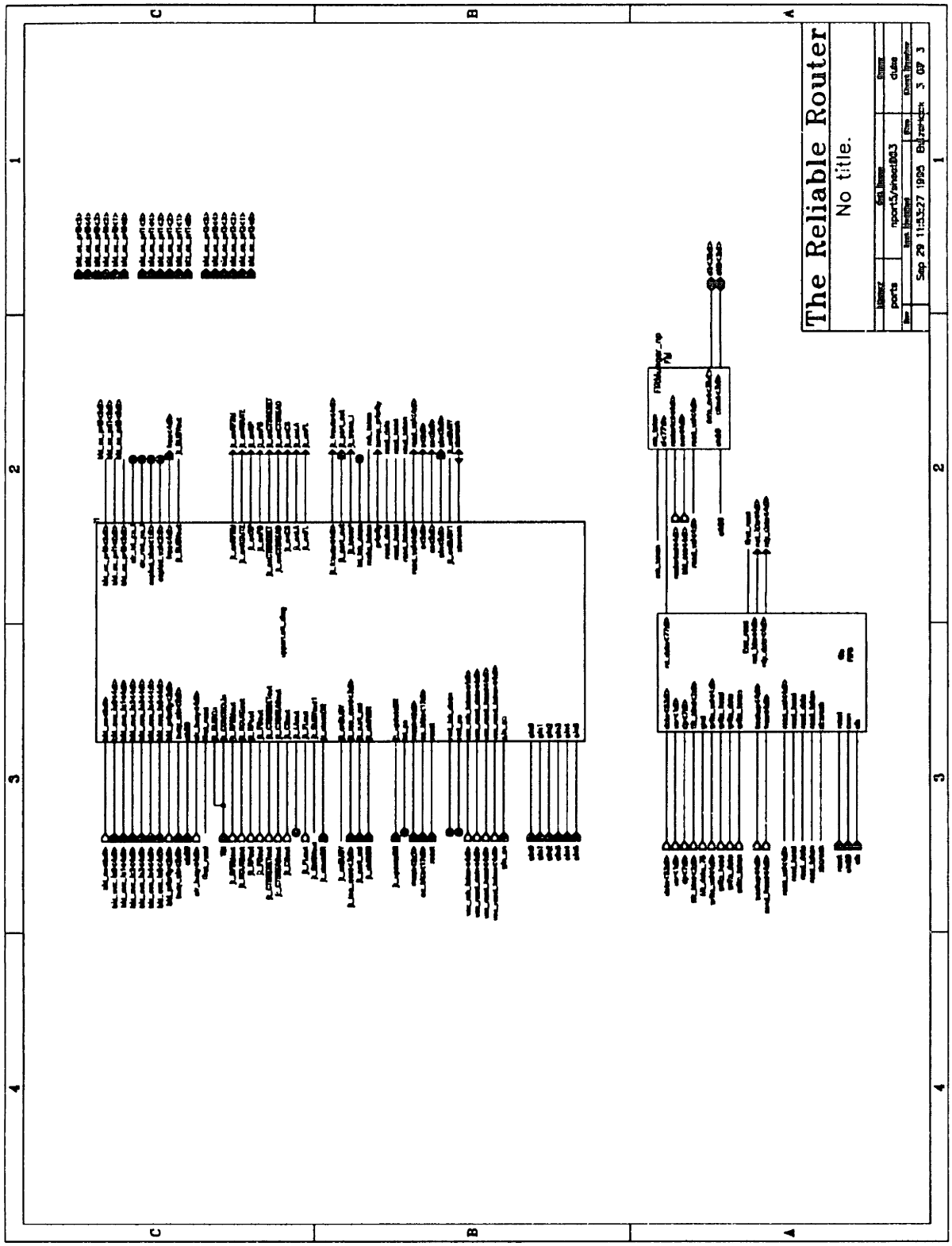


Figure A-185: Library ports, cell nport5 (sheet 3)

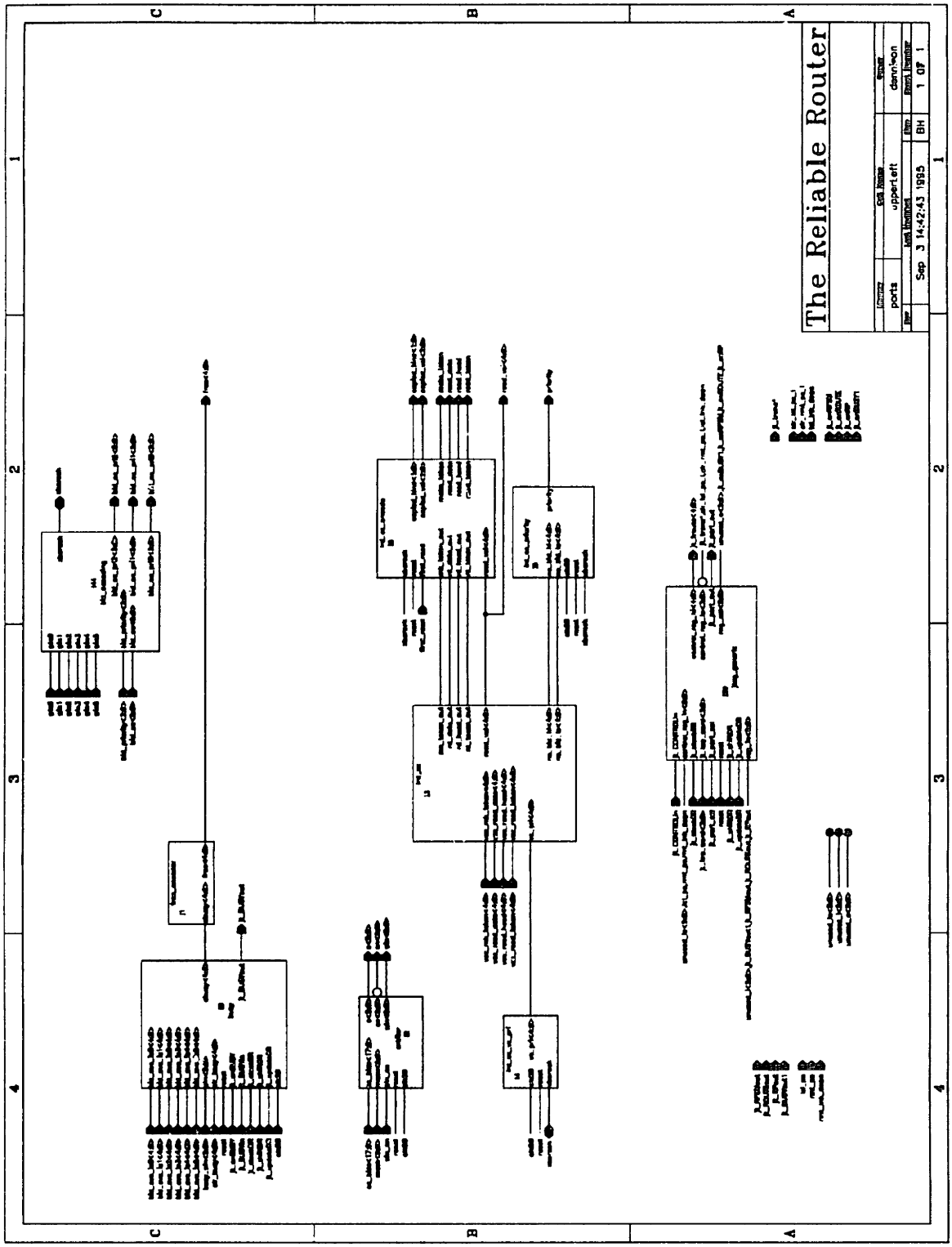


Figure A-186: Library ports, cell upperLeft

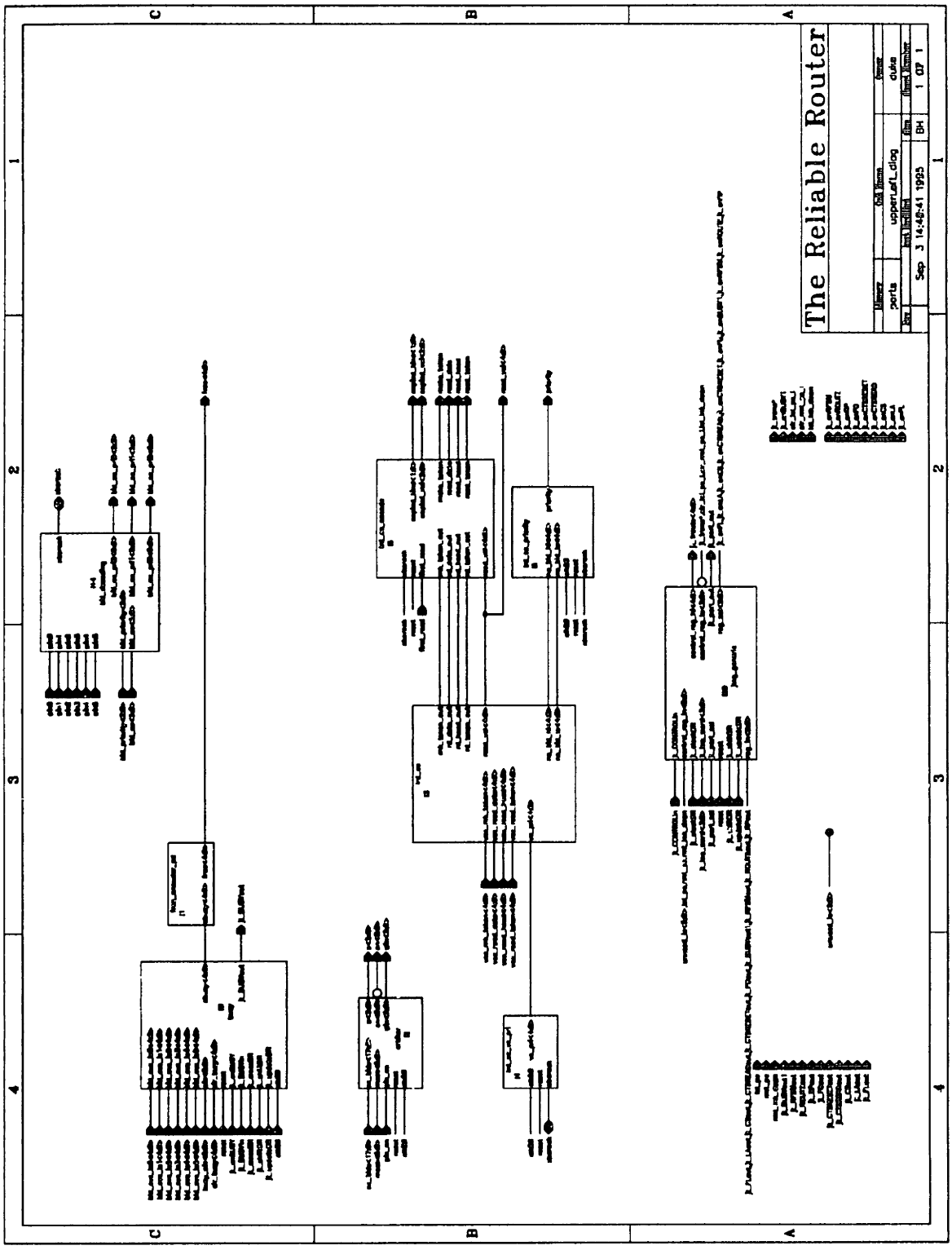


Figure A-187: Library ports, cell upperLeft_diag

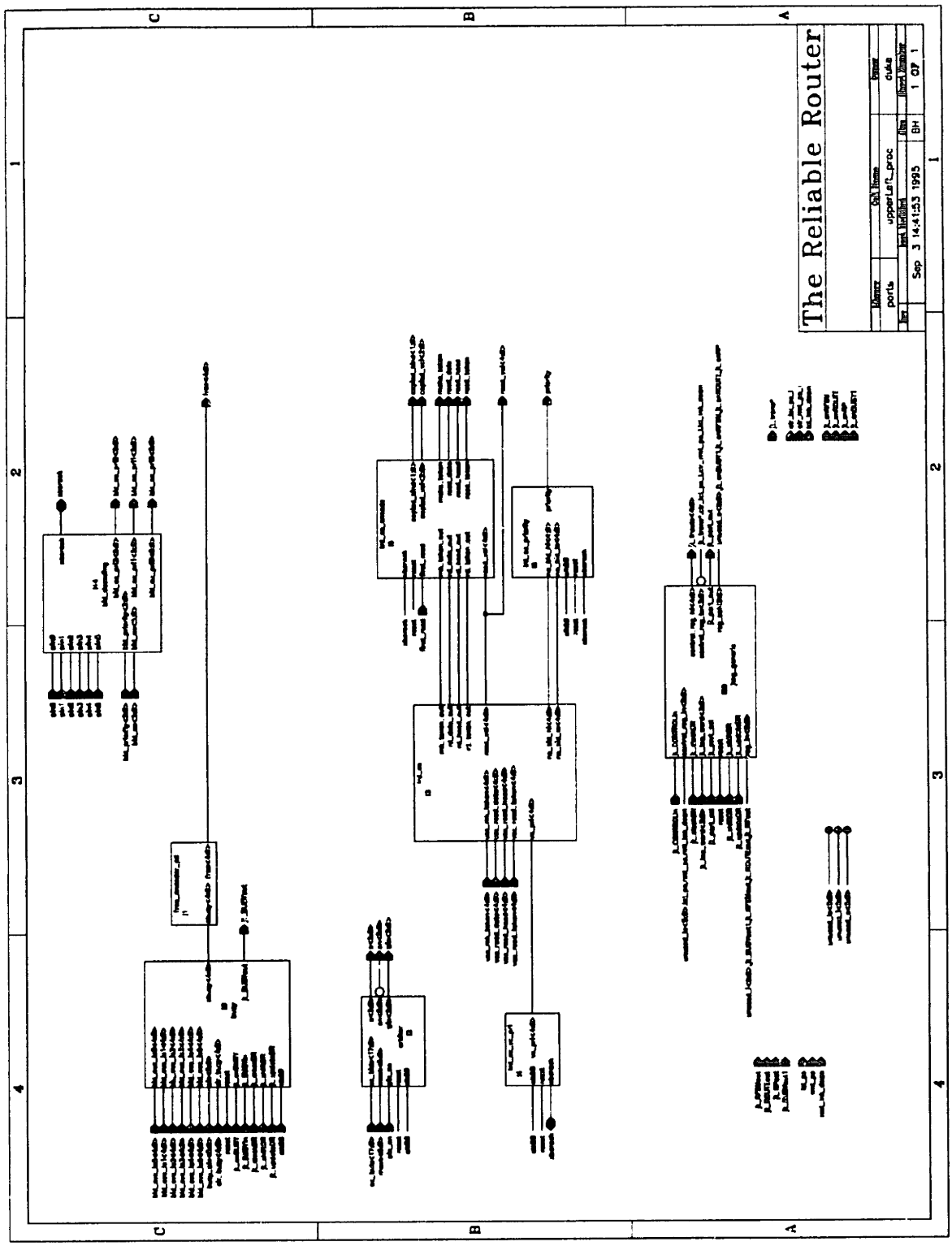


Figure A-188: Library ports, cell upperLeft_proc

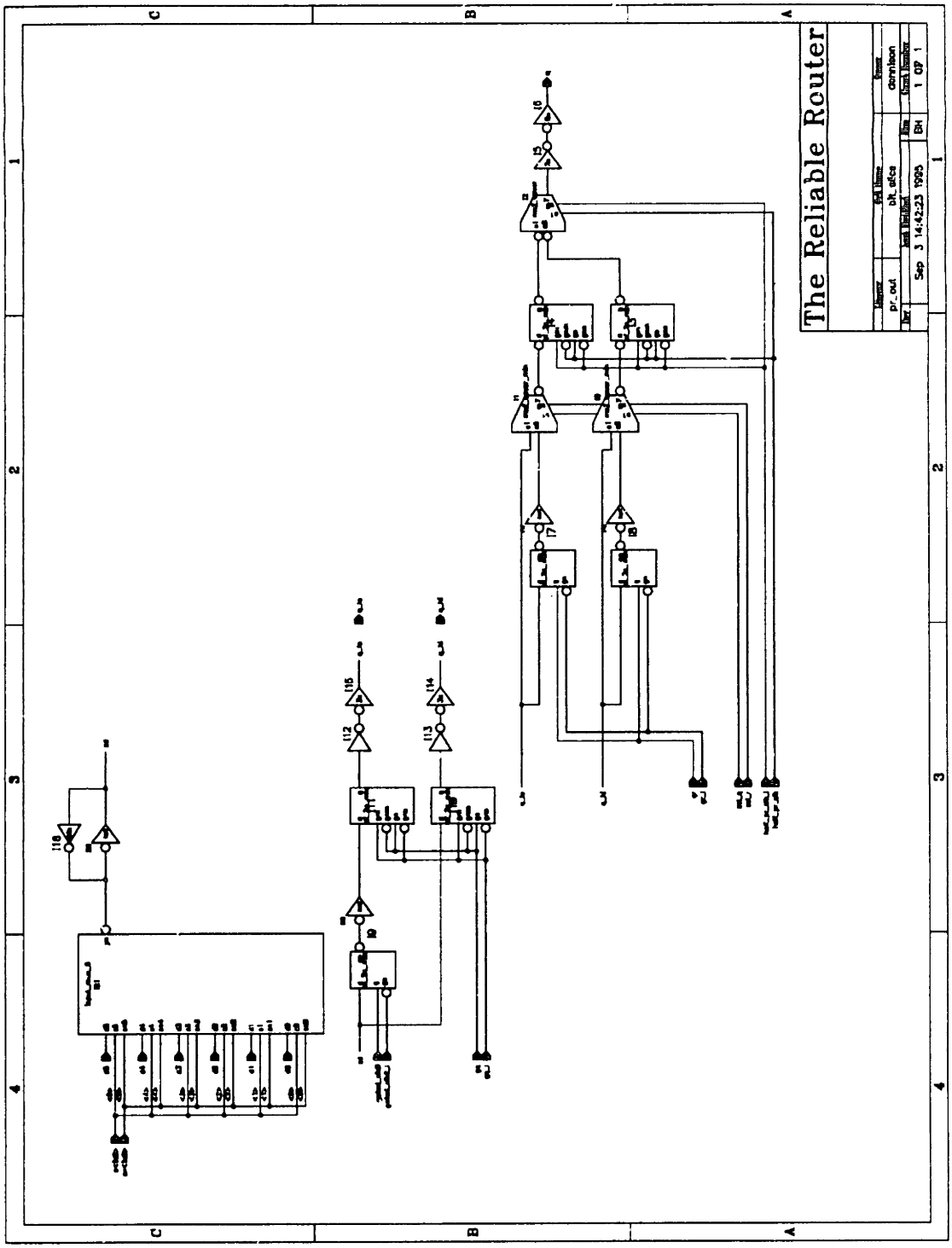


Figure A-189: Library pr_out, cell bit_slice

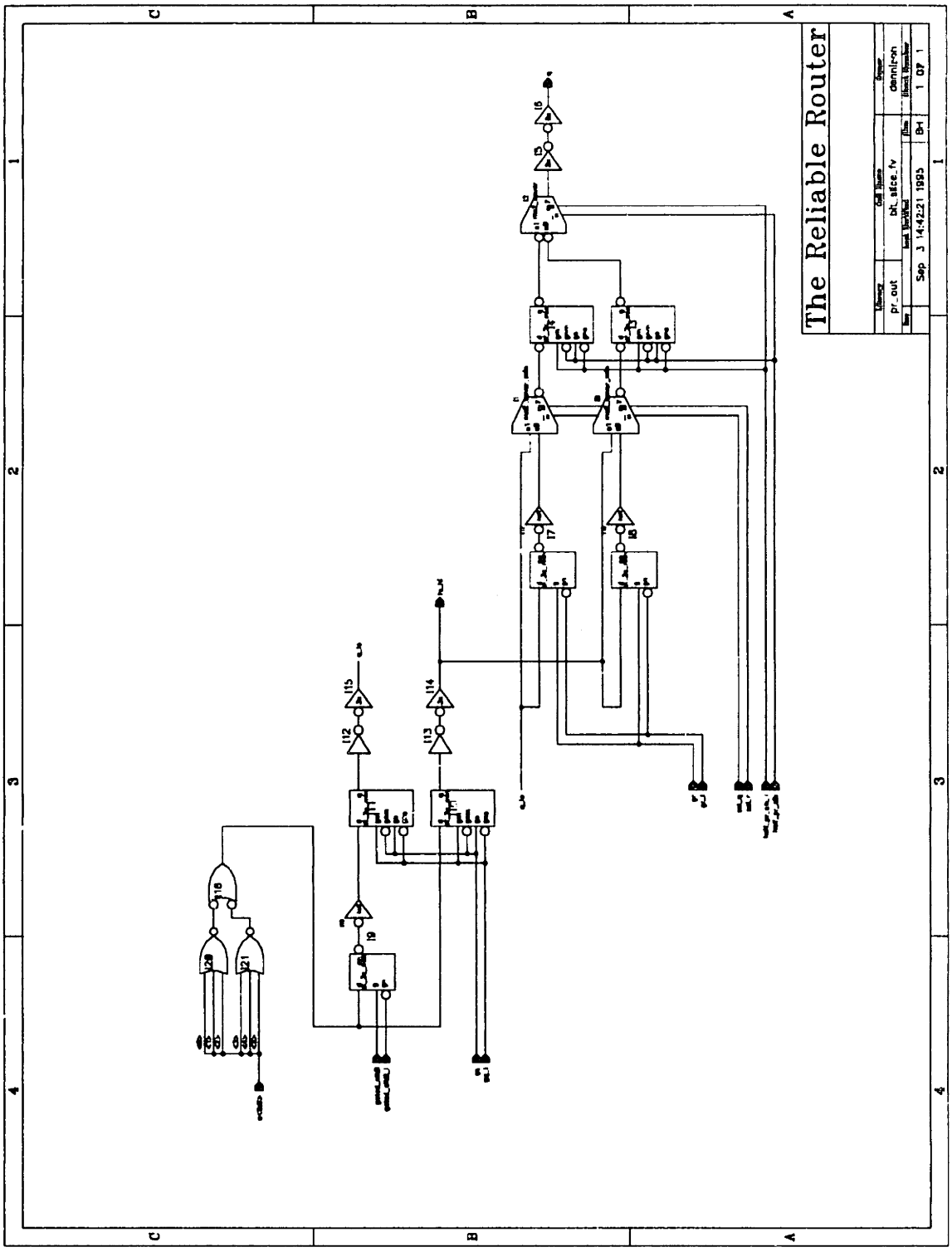


Figure A-190: Library pr_out, cell bit_slice_fv

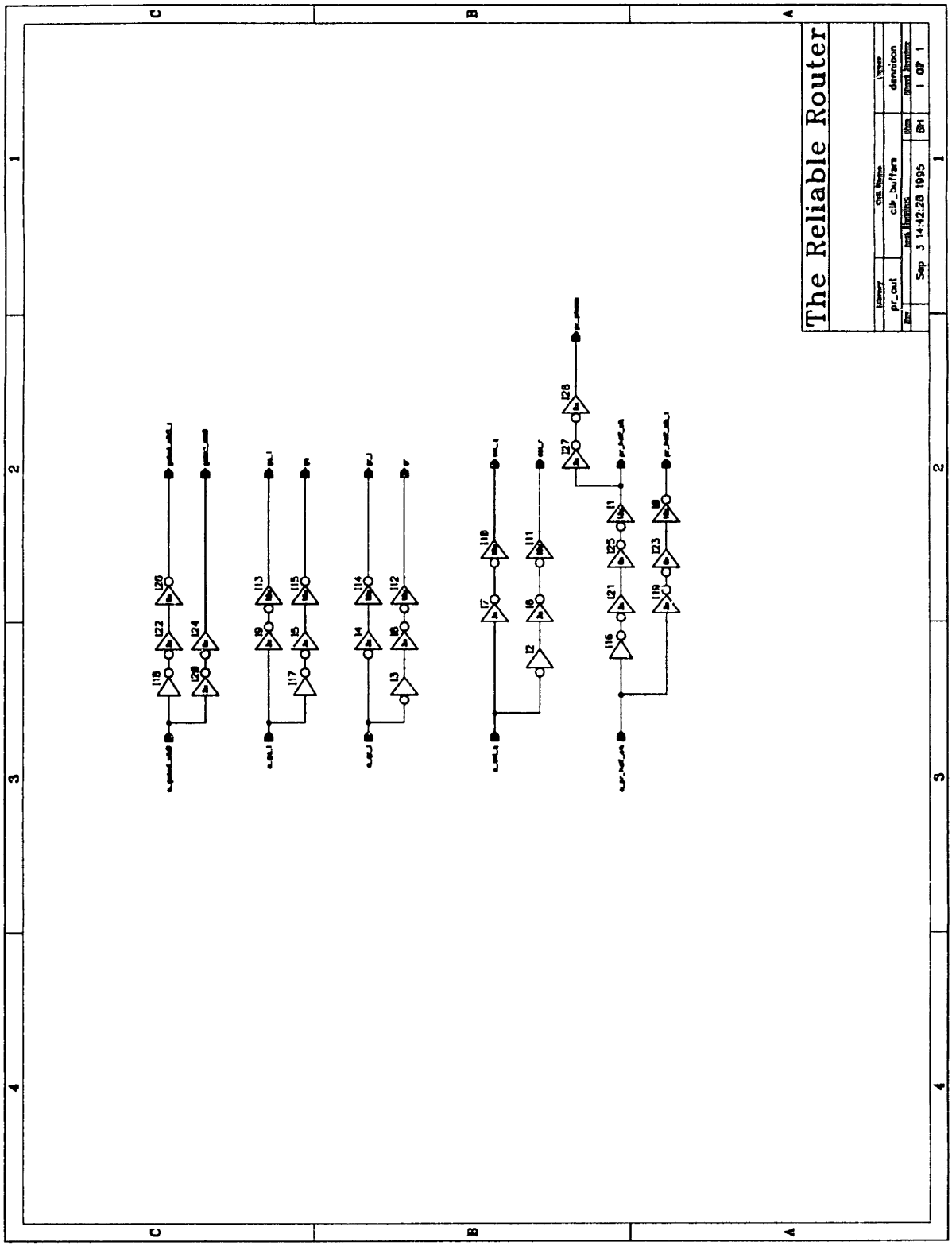


Figure A-191: Library pr_out, cell clk_buffers

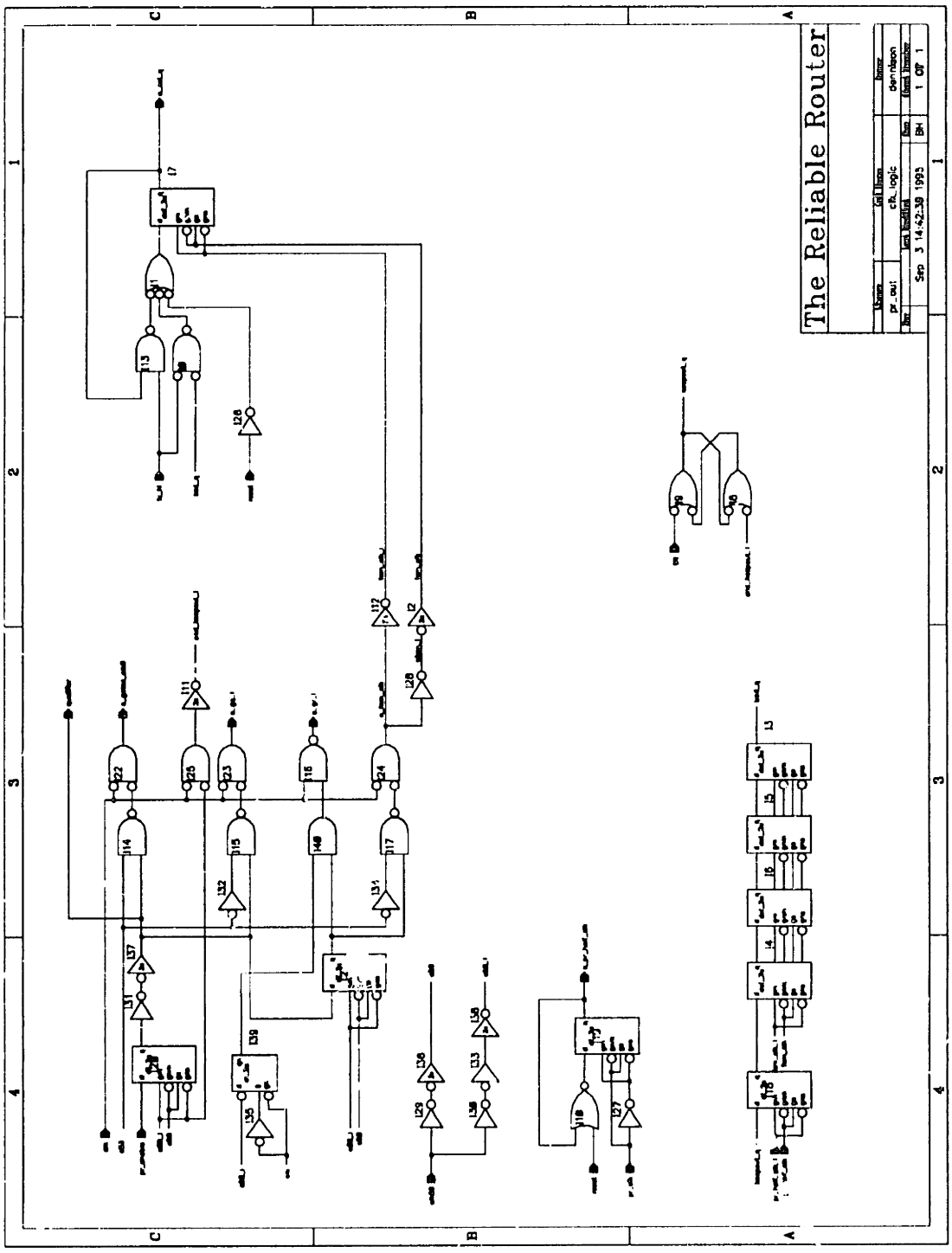


Figure A-192: Library pr_out, cell clk_logic

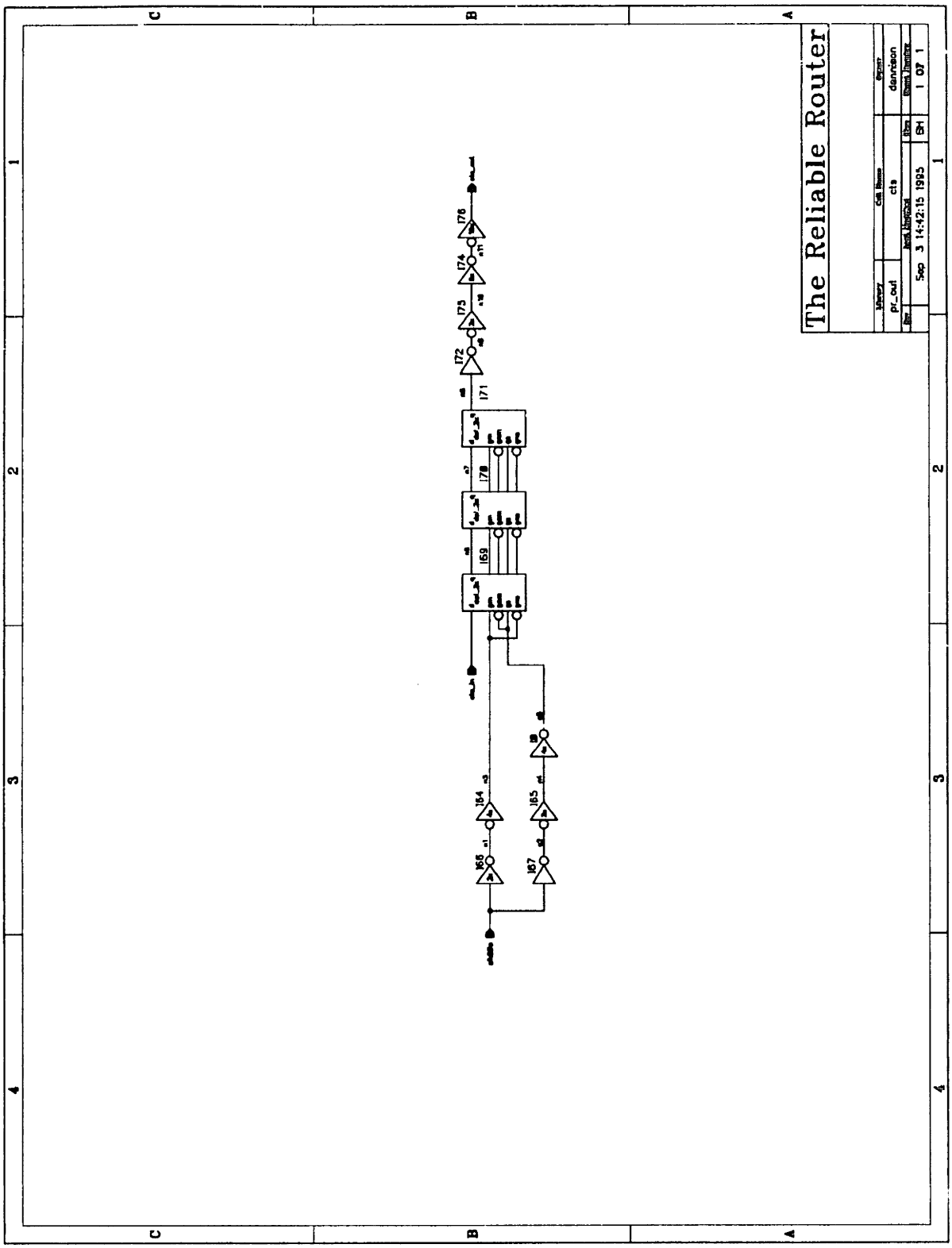


Figure A-193: Library pr_out, cell cts

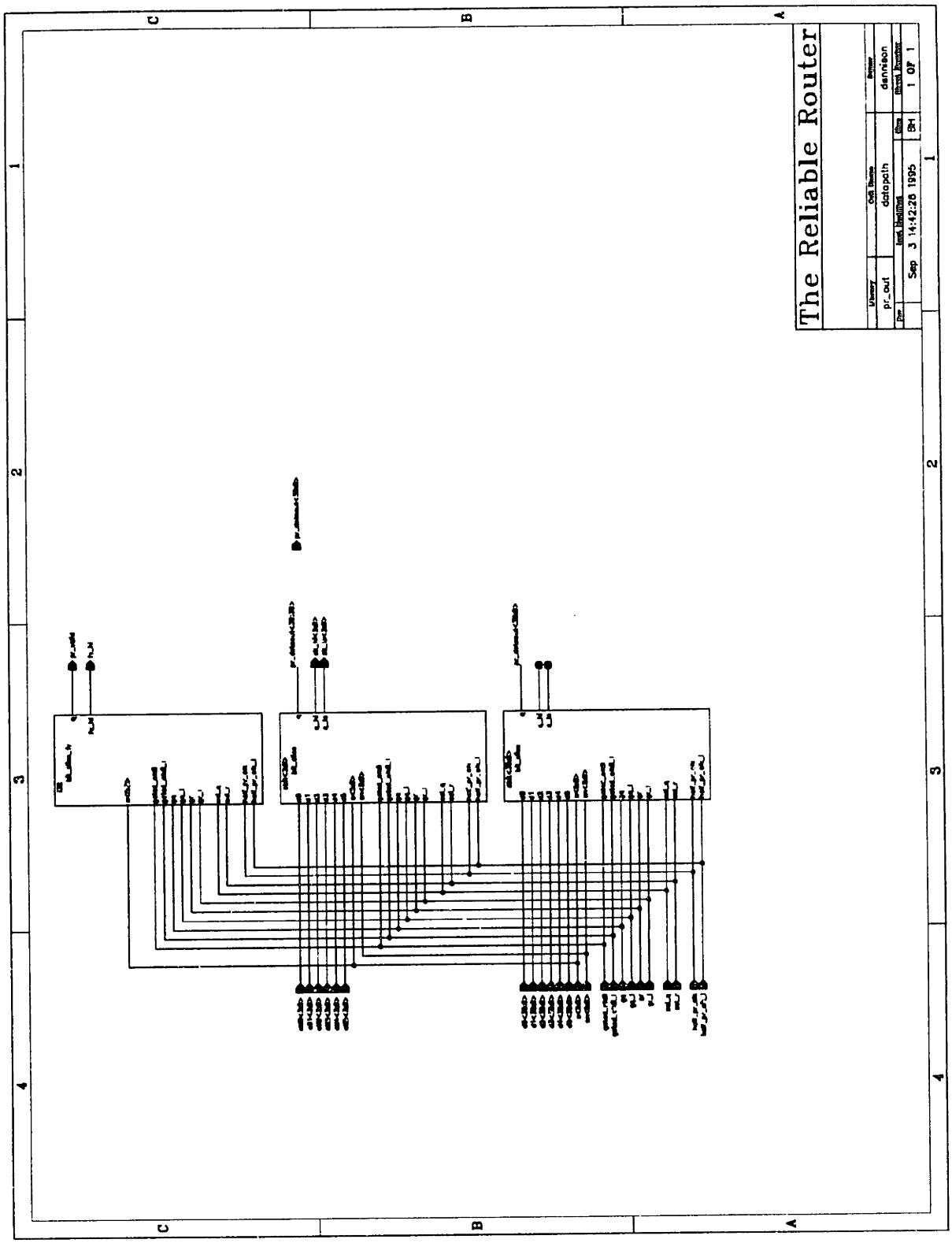


Figure A-194: Library pr_out, cell datapath

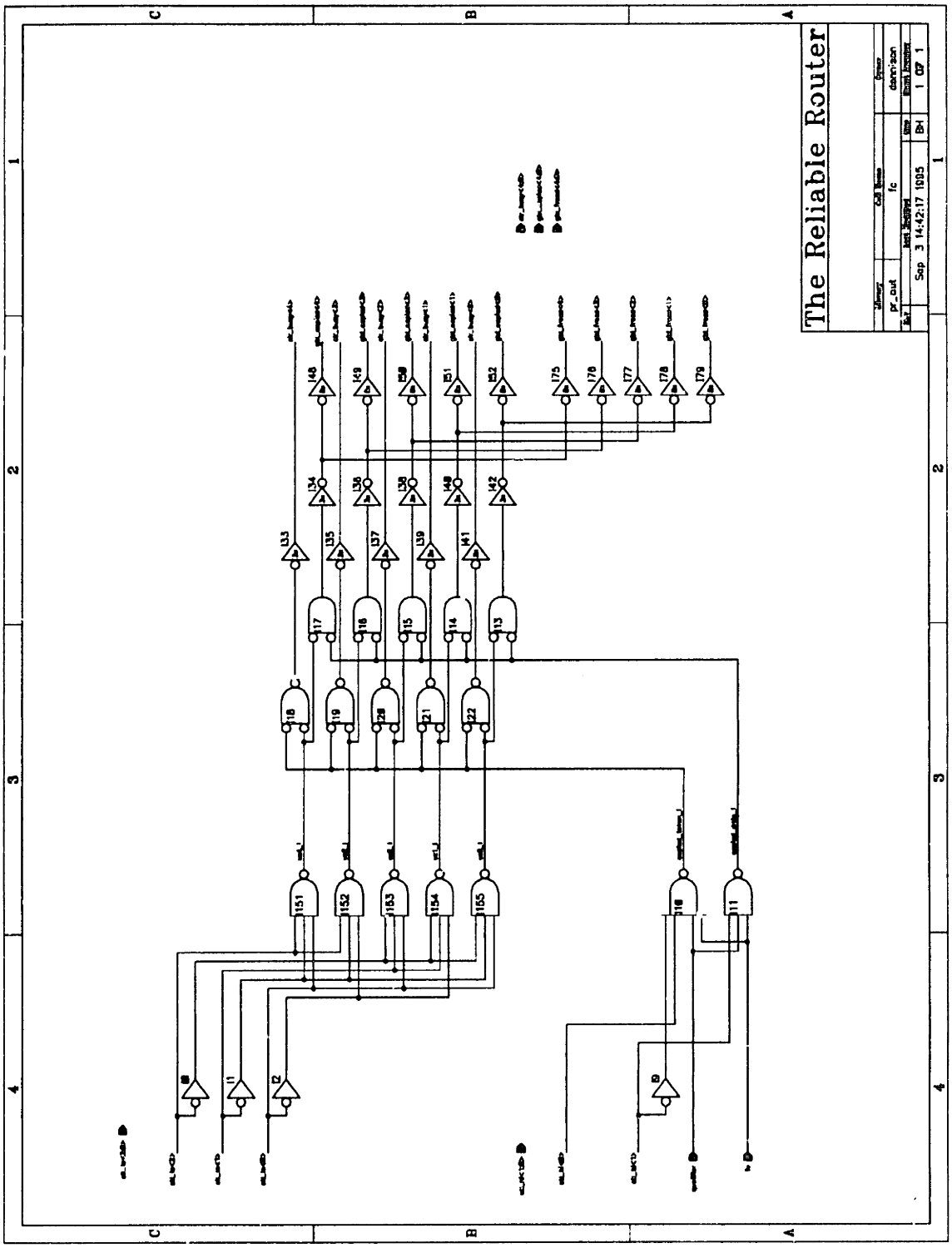


Figure A-195: Library pr_out, cell fc

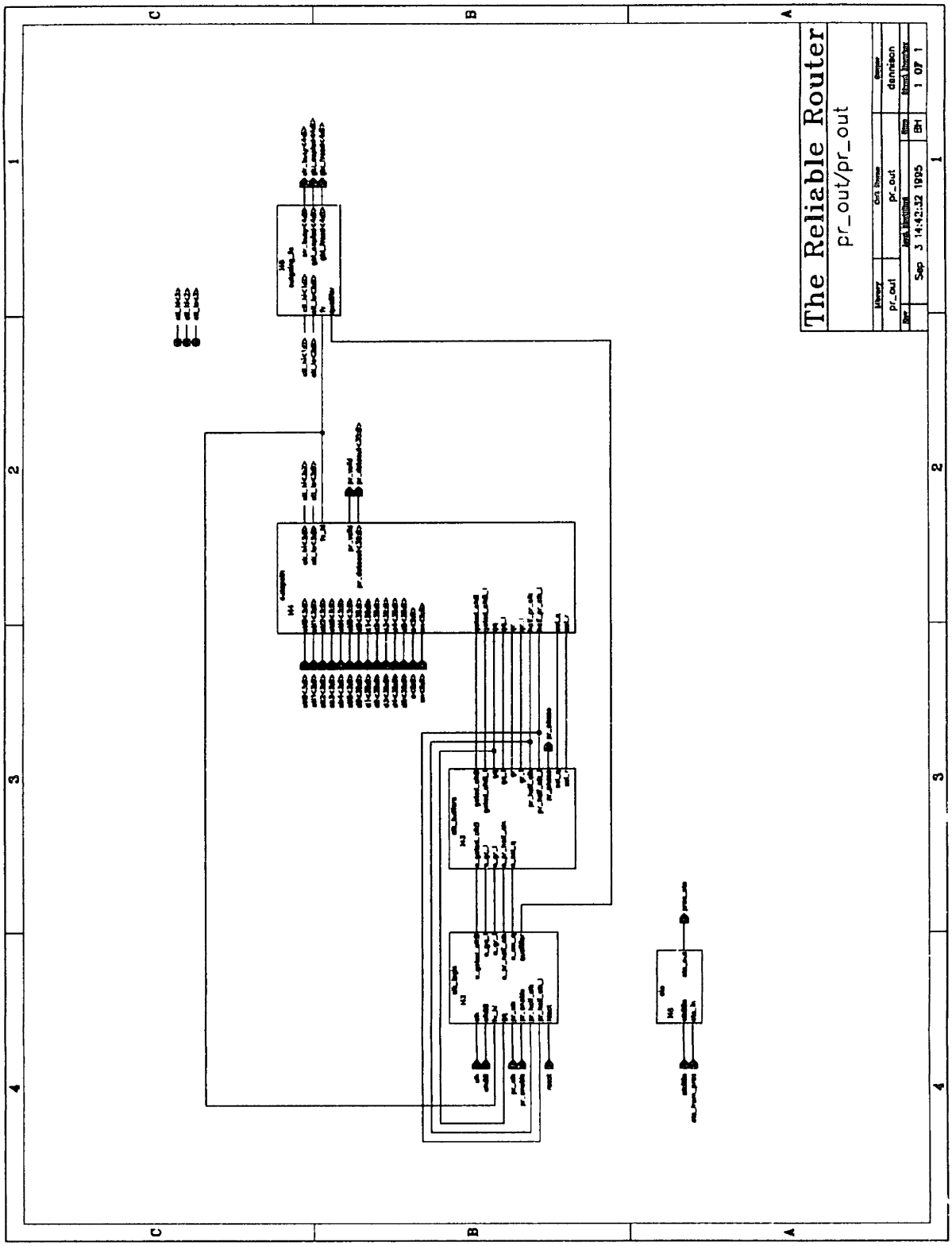


Figure A-196: Library pr_out, cell pr_out

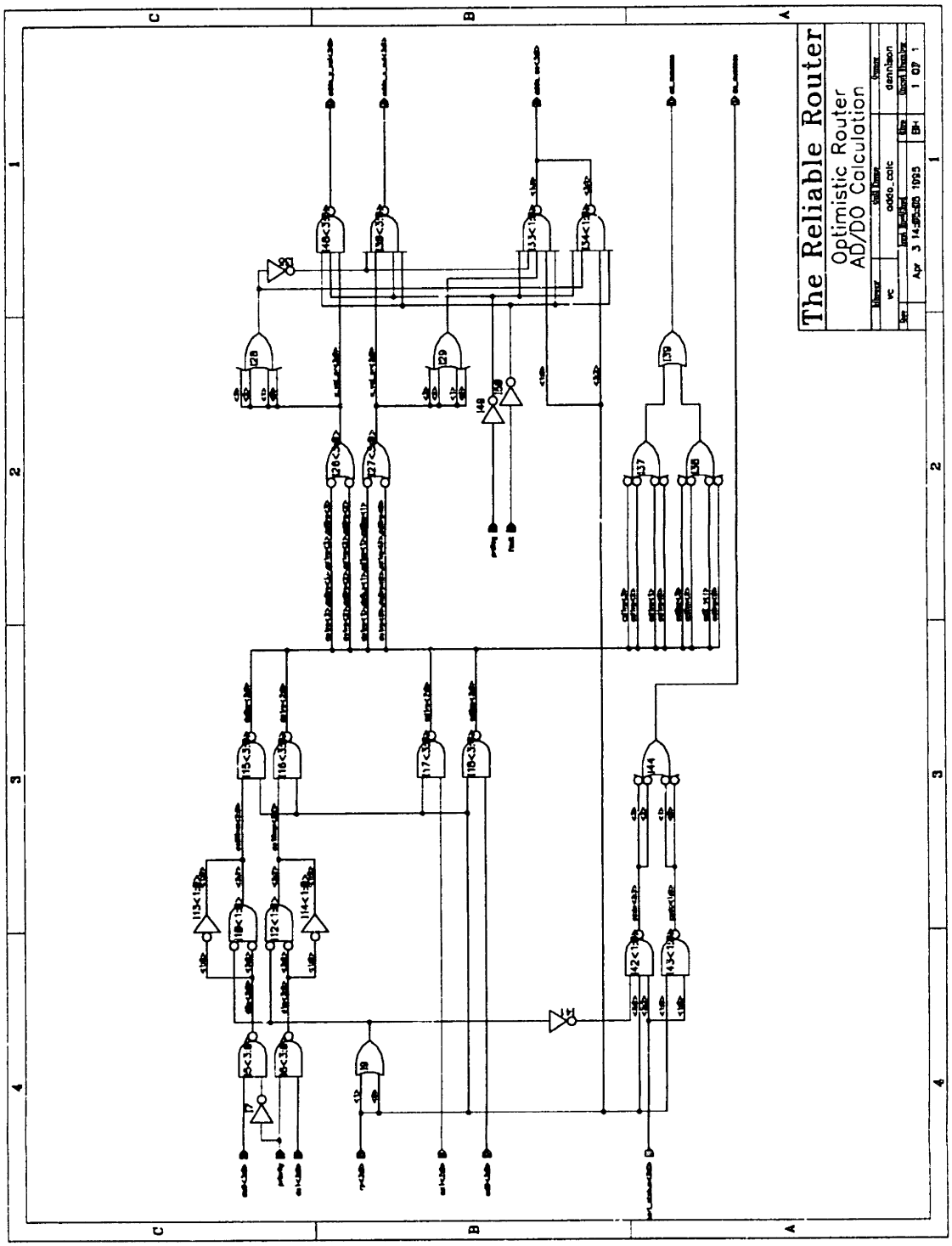


Figure A-197: Library vc, cell addo_calc

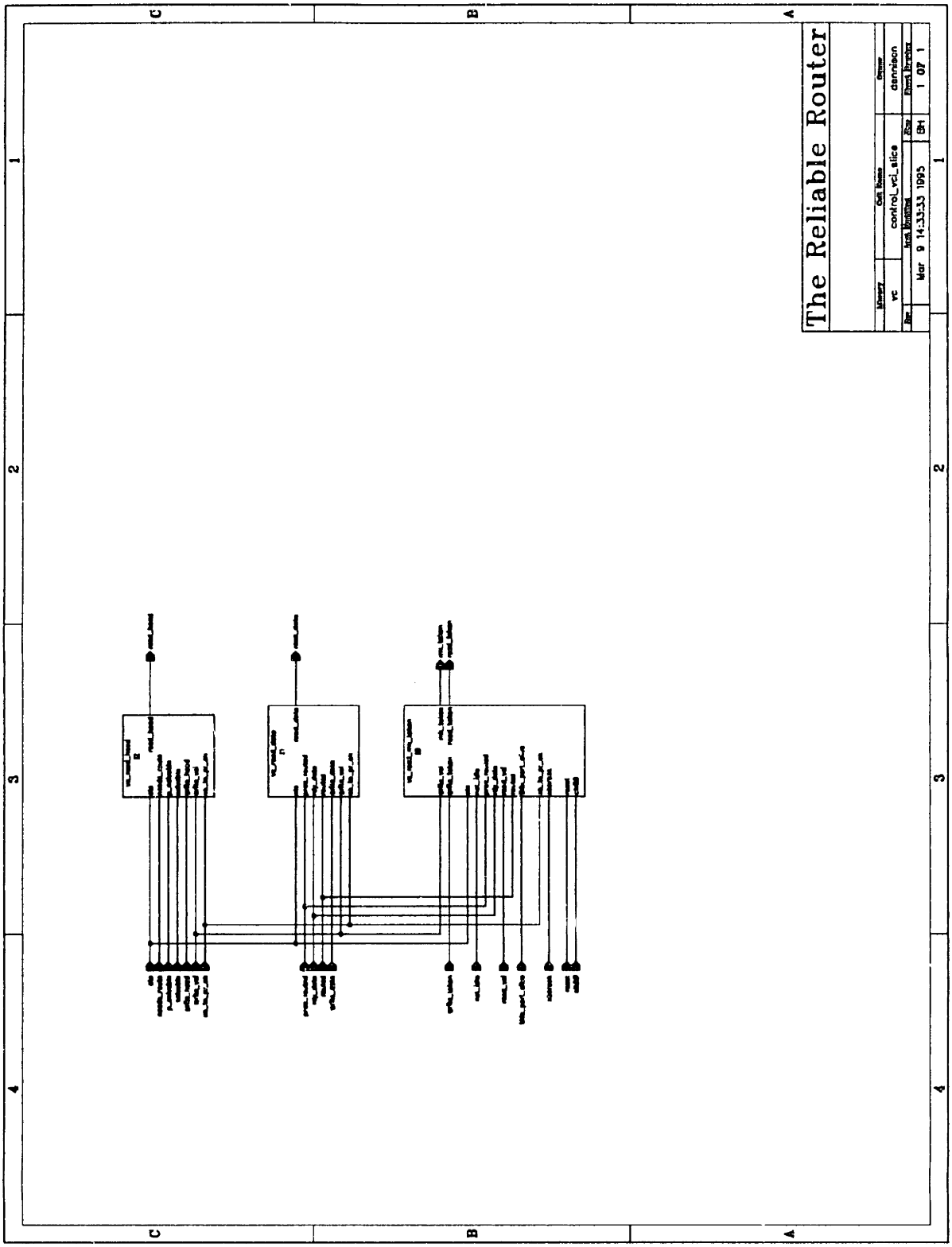


Figure A-198: Library vc, cell control_vci_slice

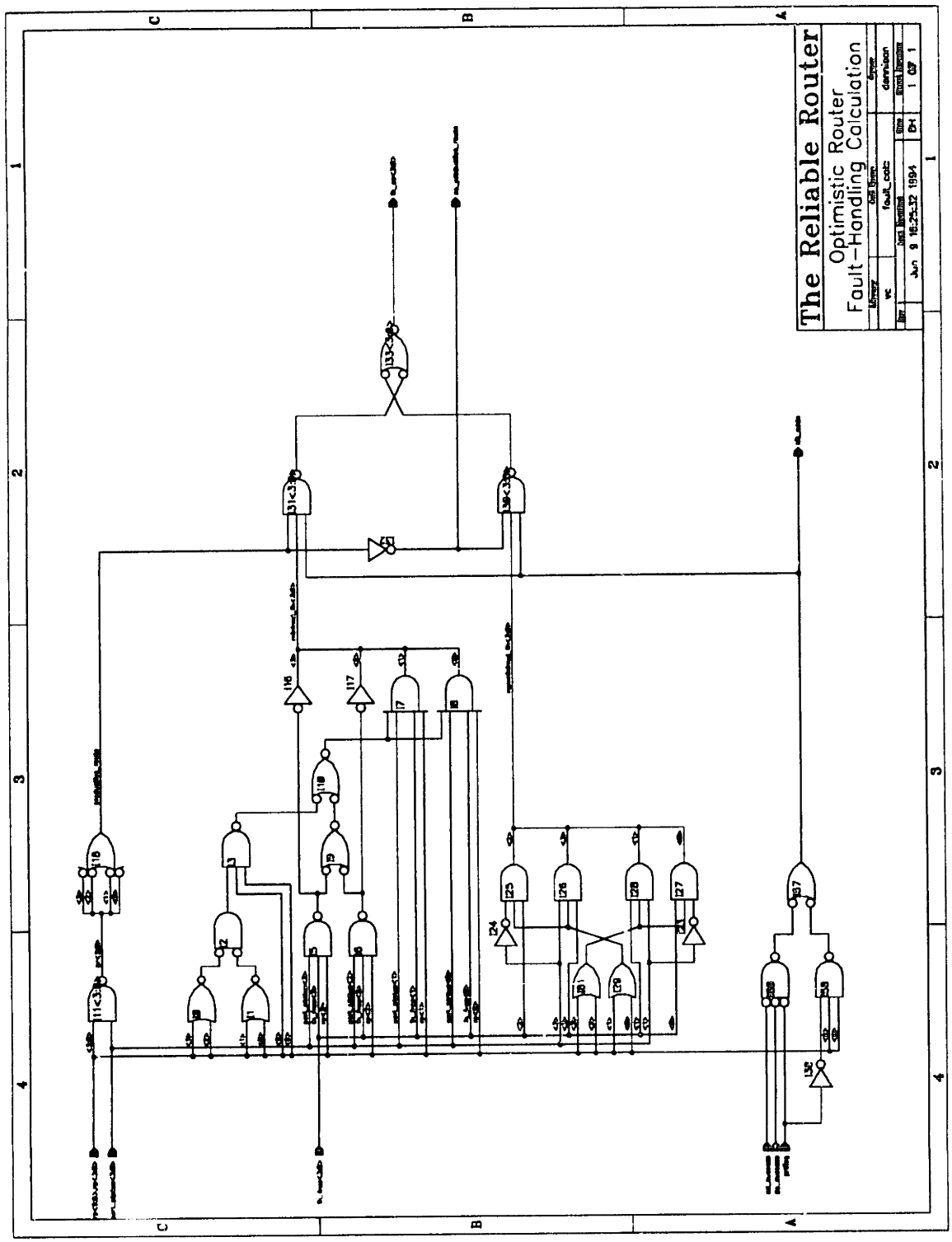


Figure A-199: Library vc, cell fault_calc

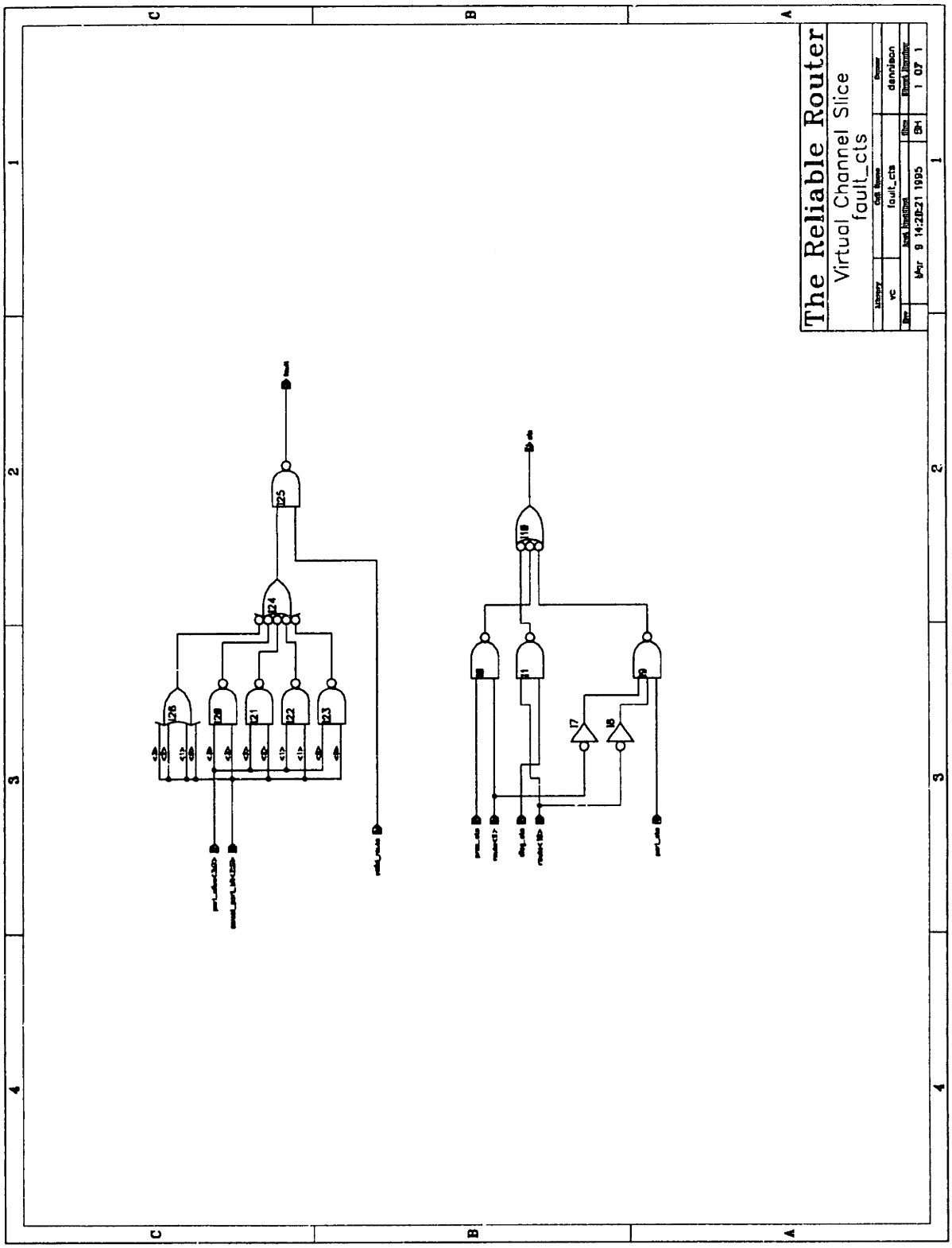


Figure A-200: Library vc, cell fault_cts

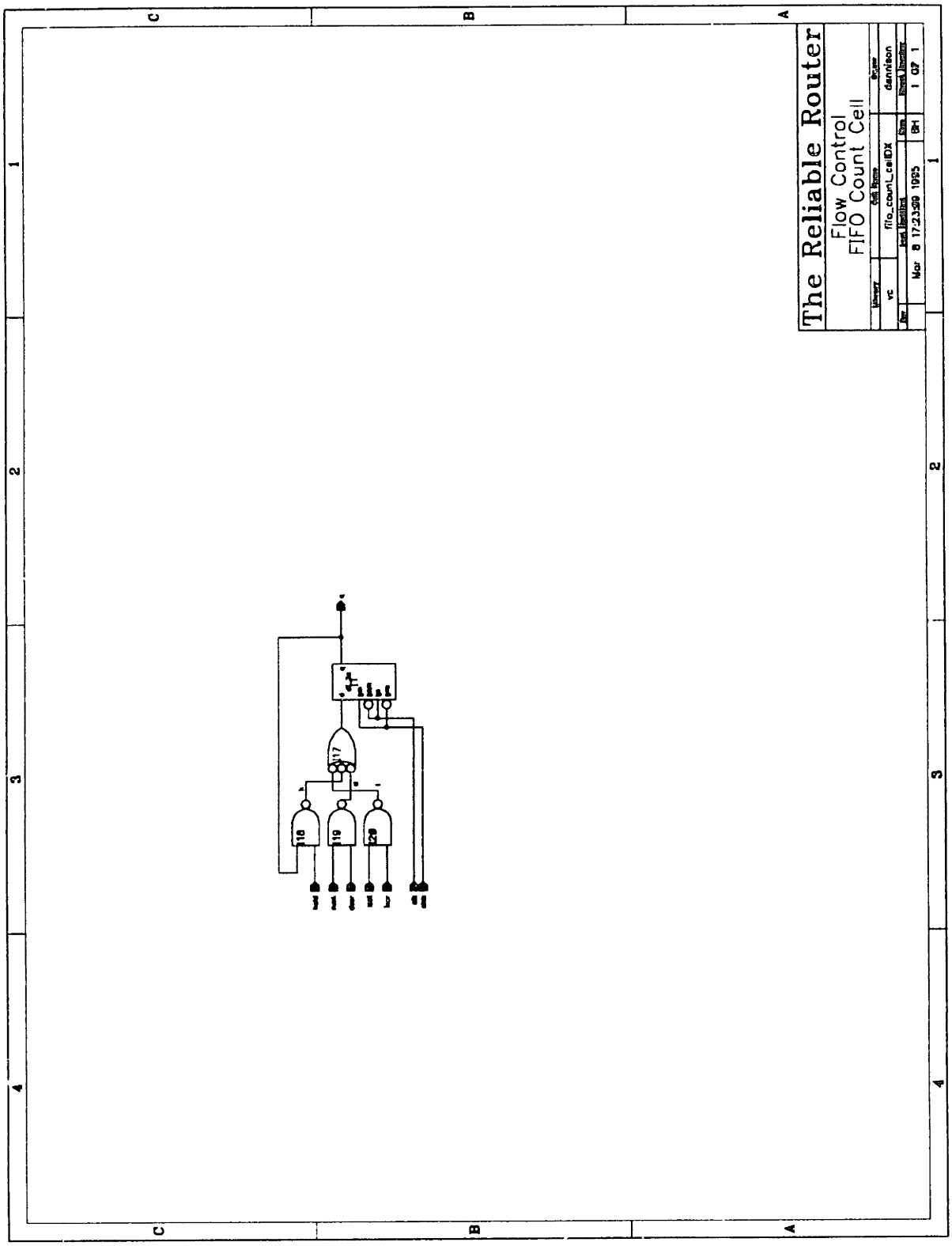


Figure A-201: Library vc, cell fifo_count_cellDX

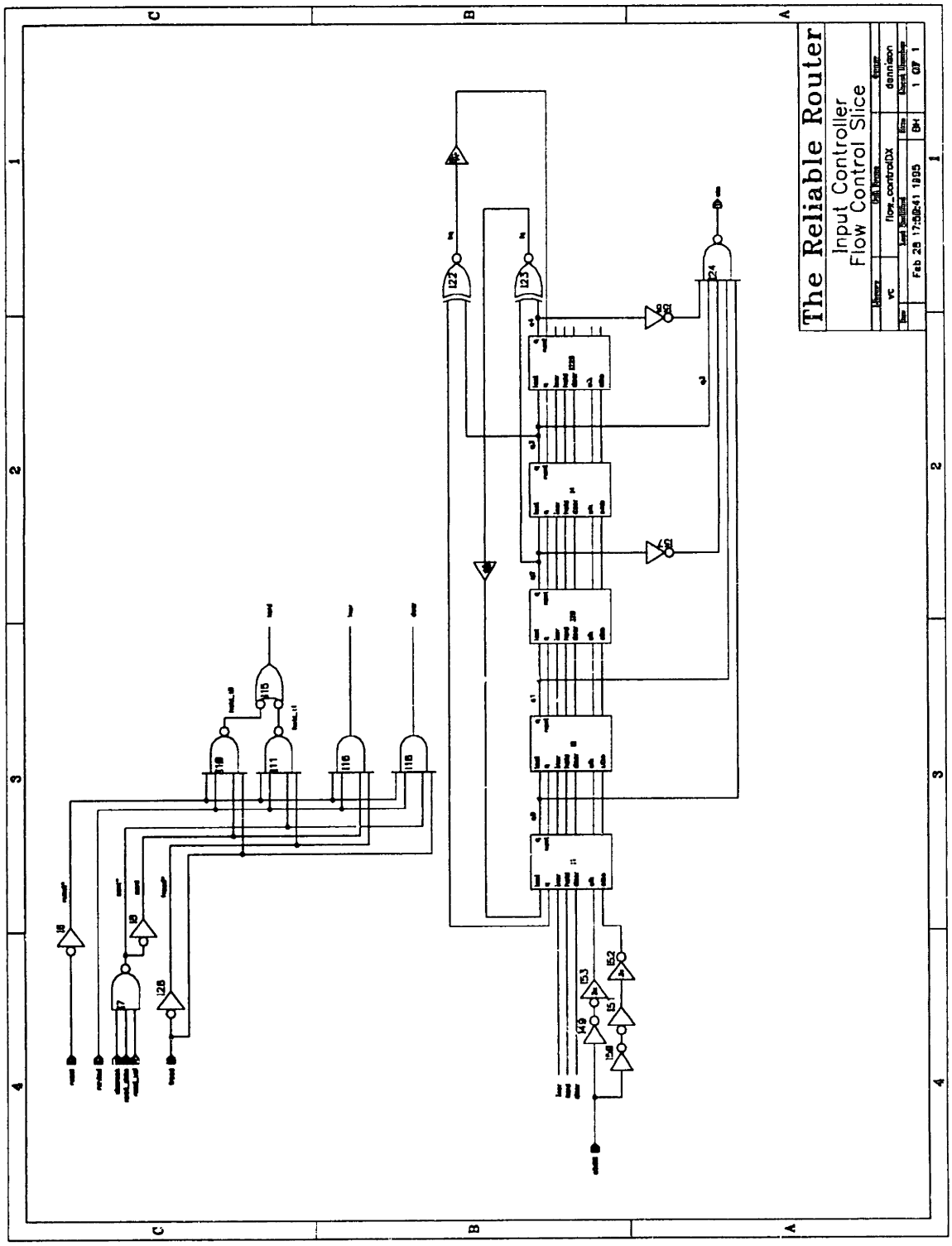


Figure A-202: Library vc, cell flow_controlDX

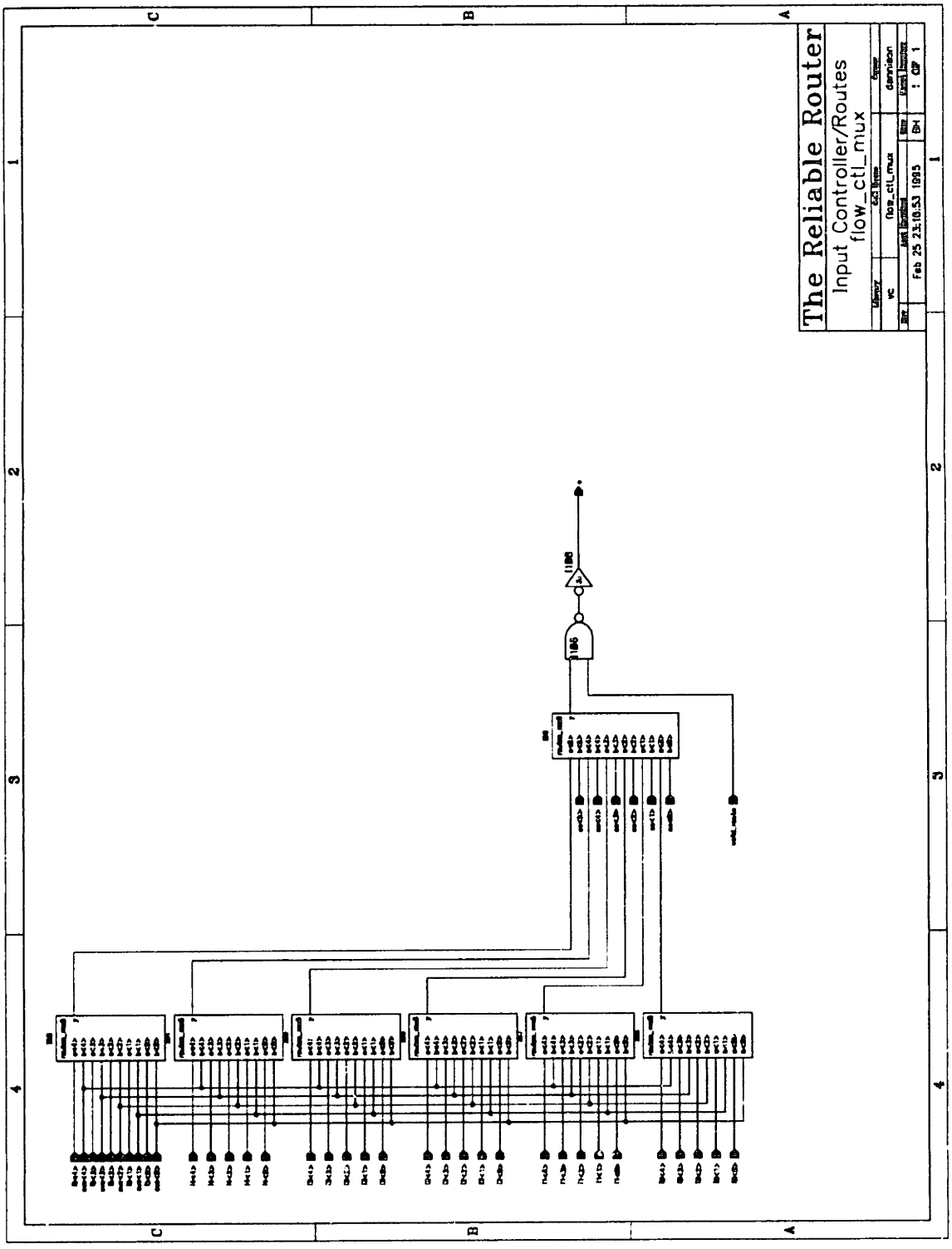


Figure A-203: Library vc, cell flow_ctl_mux

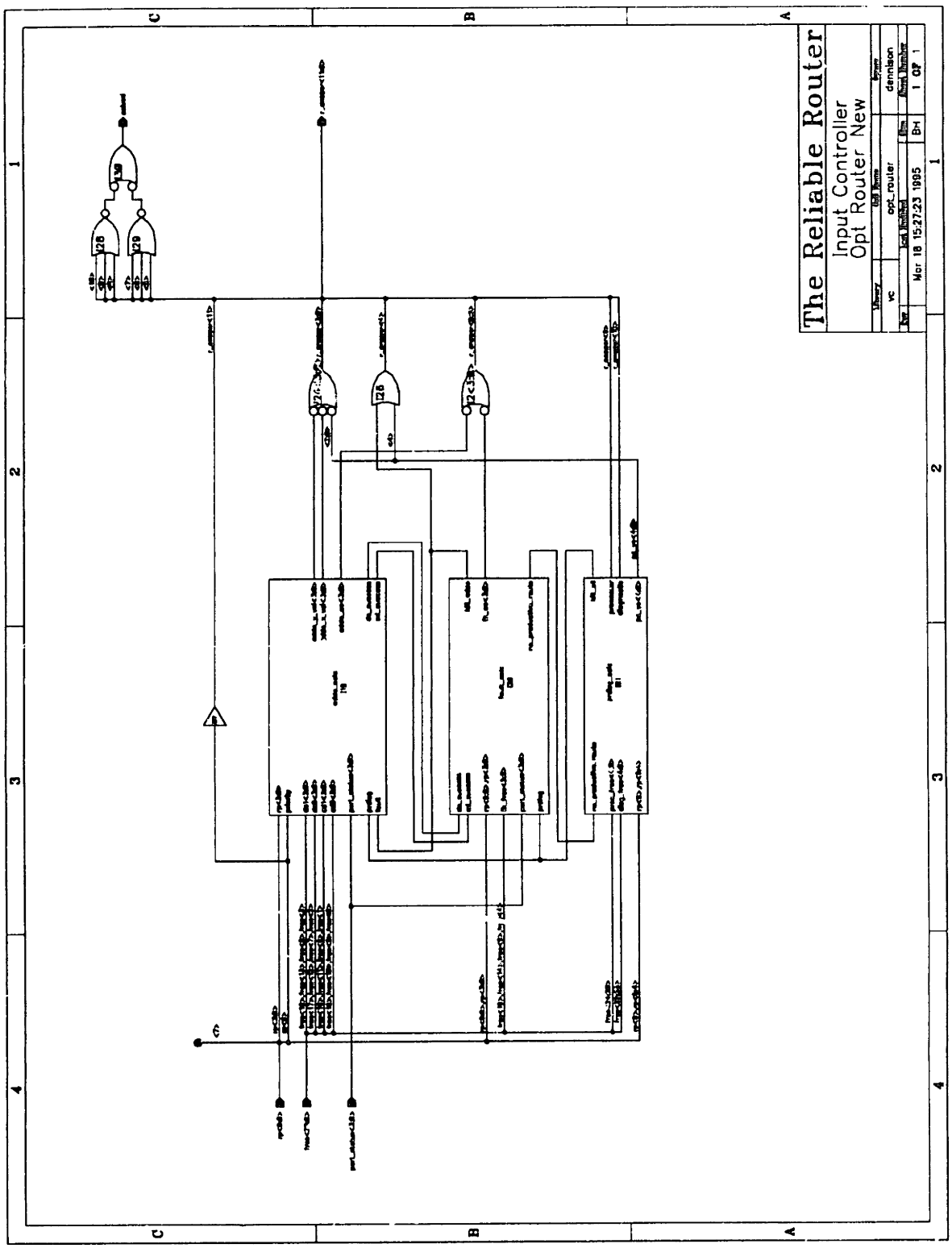


Figure A-204: Library vc, cell opt_router

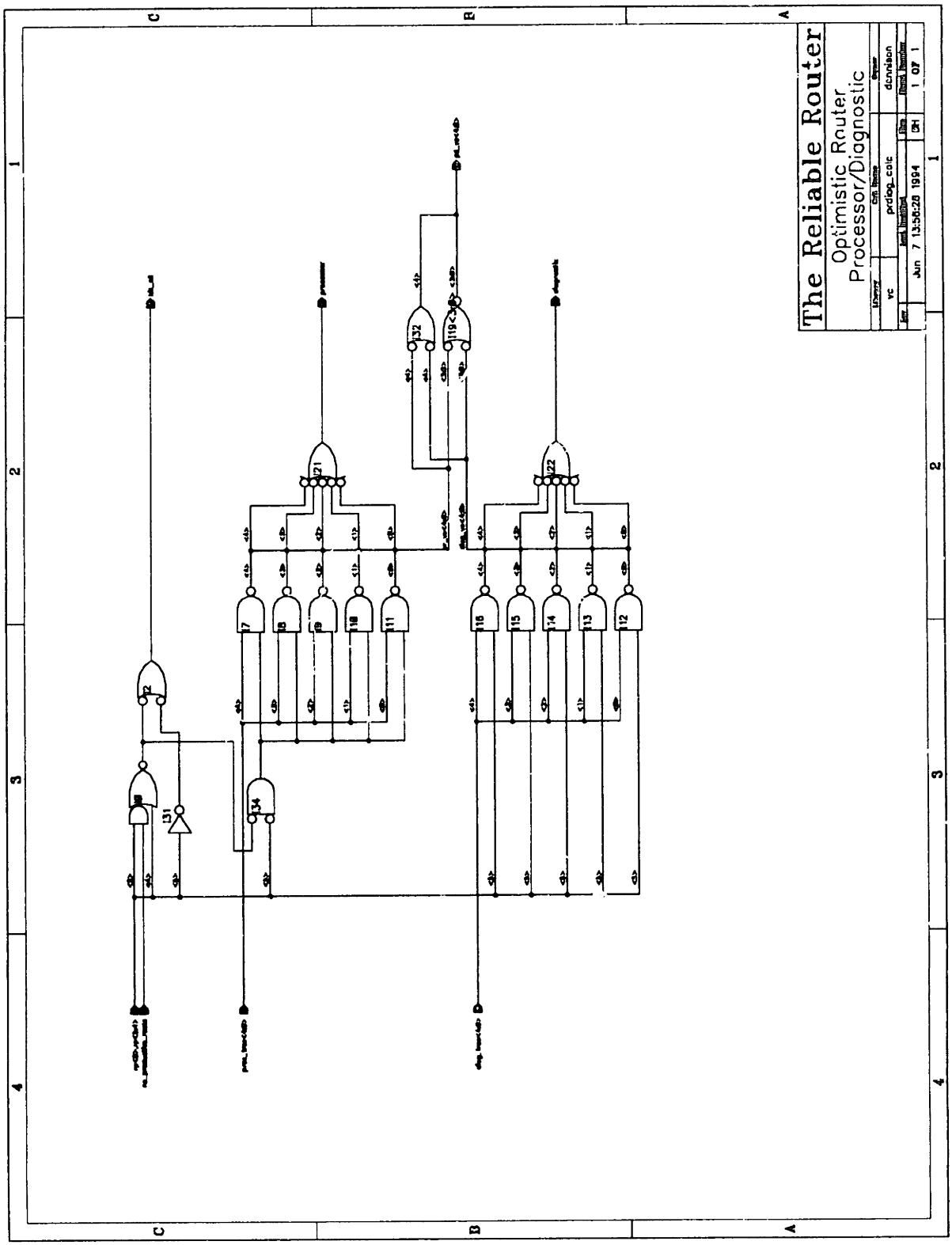


Figure A-205: Library vc, cell prdiag_calc

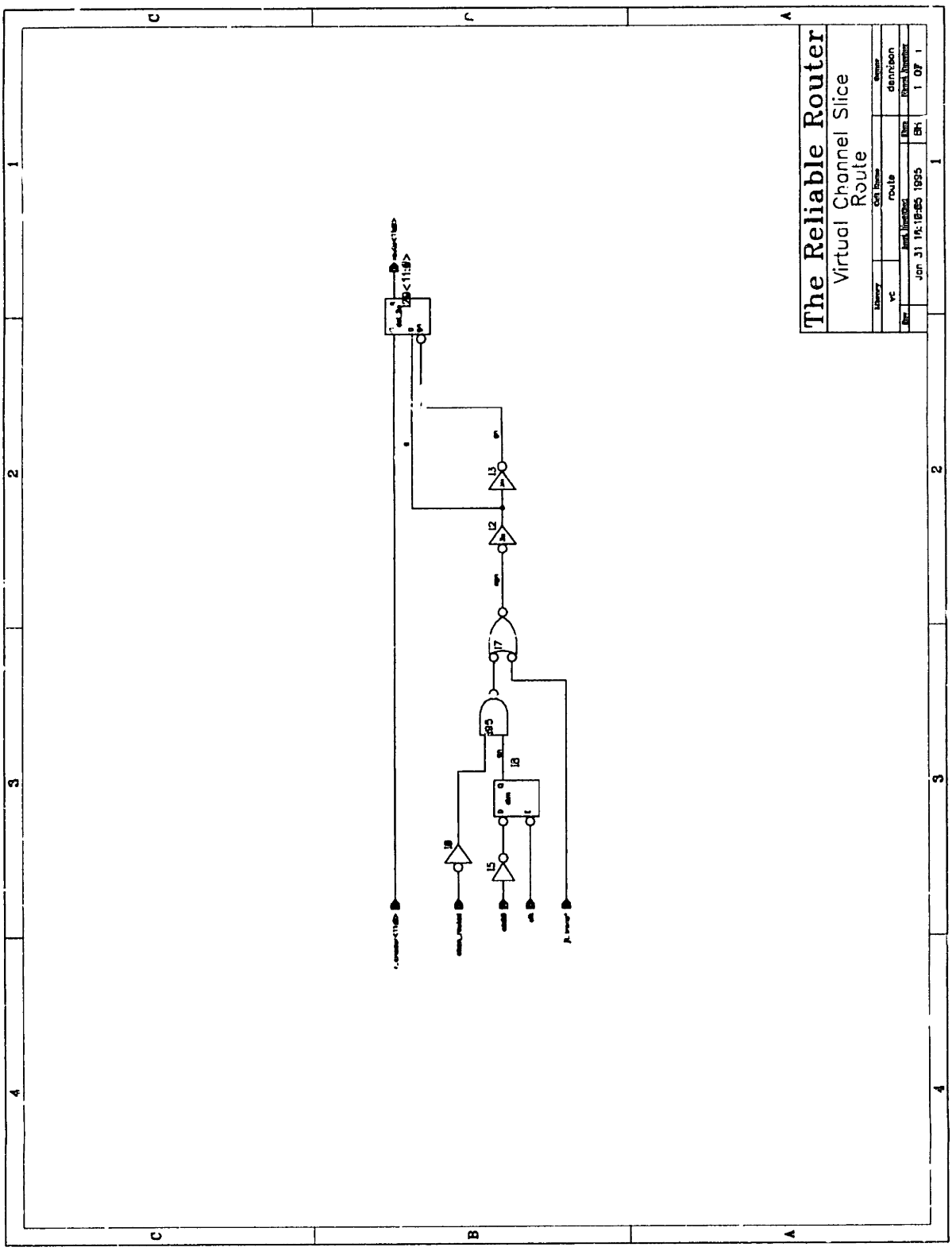


Figure A-206. Library vc, cell route

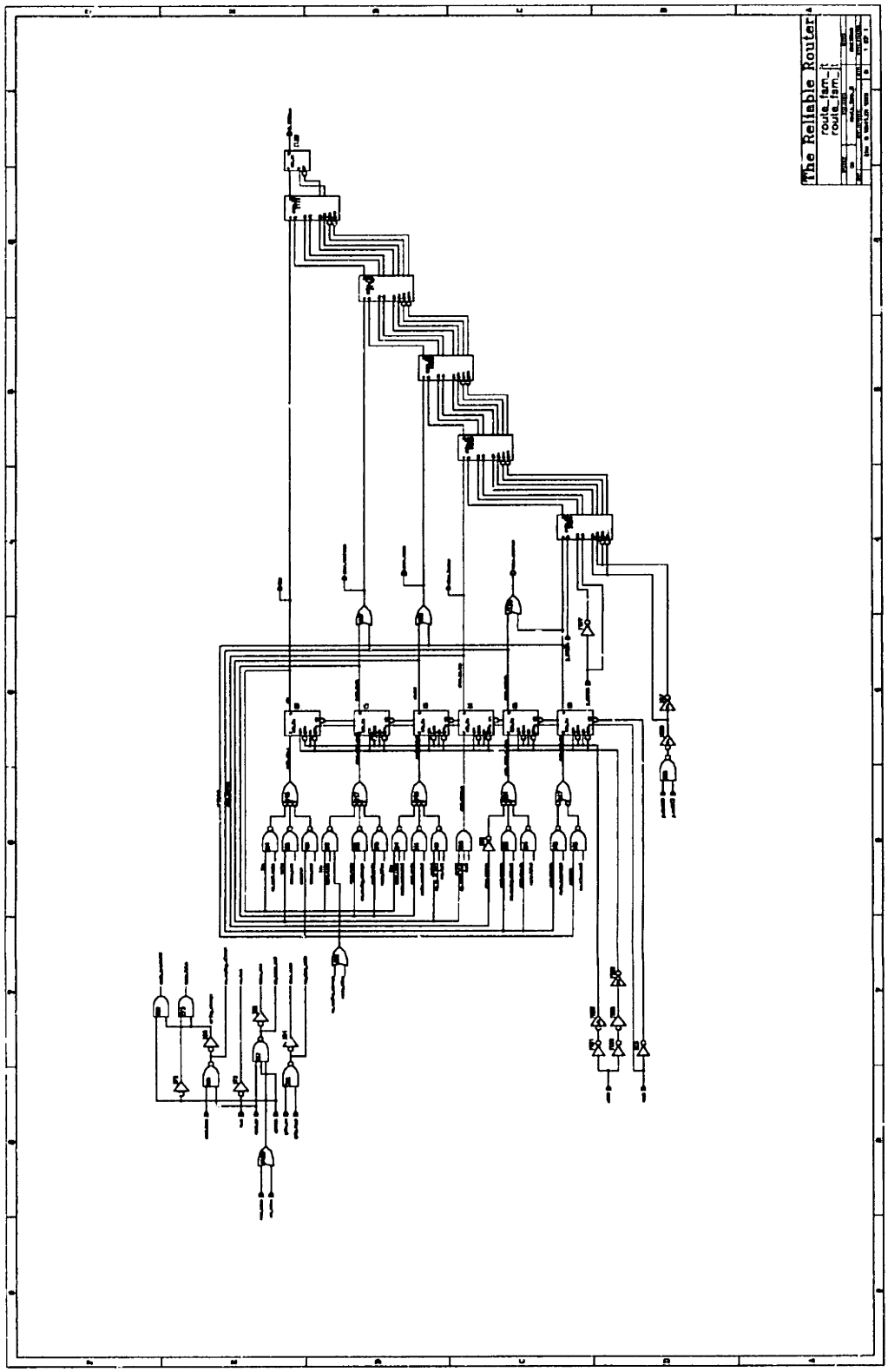


Figure A-207: Library vc, cell route_fsm_jt

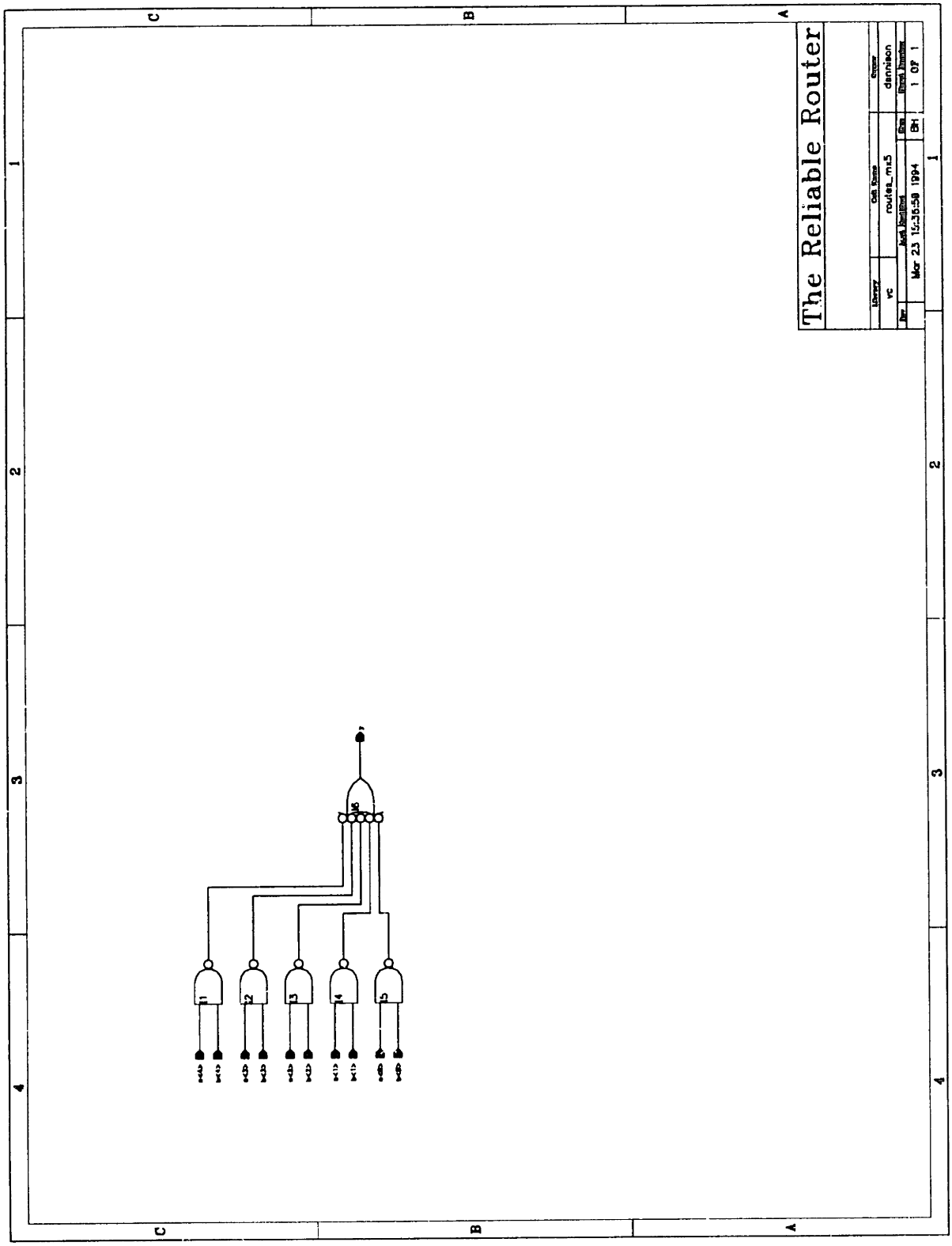


Figure A-208: Library vc, cell routes_mx5

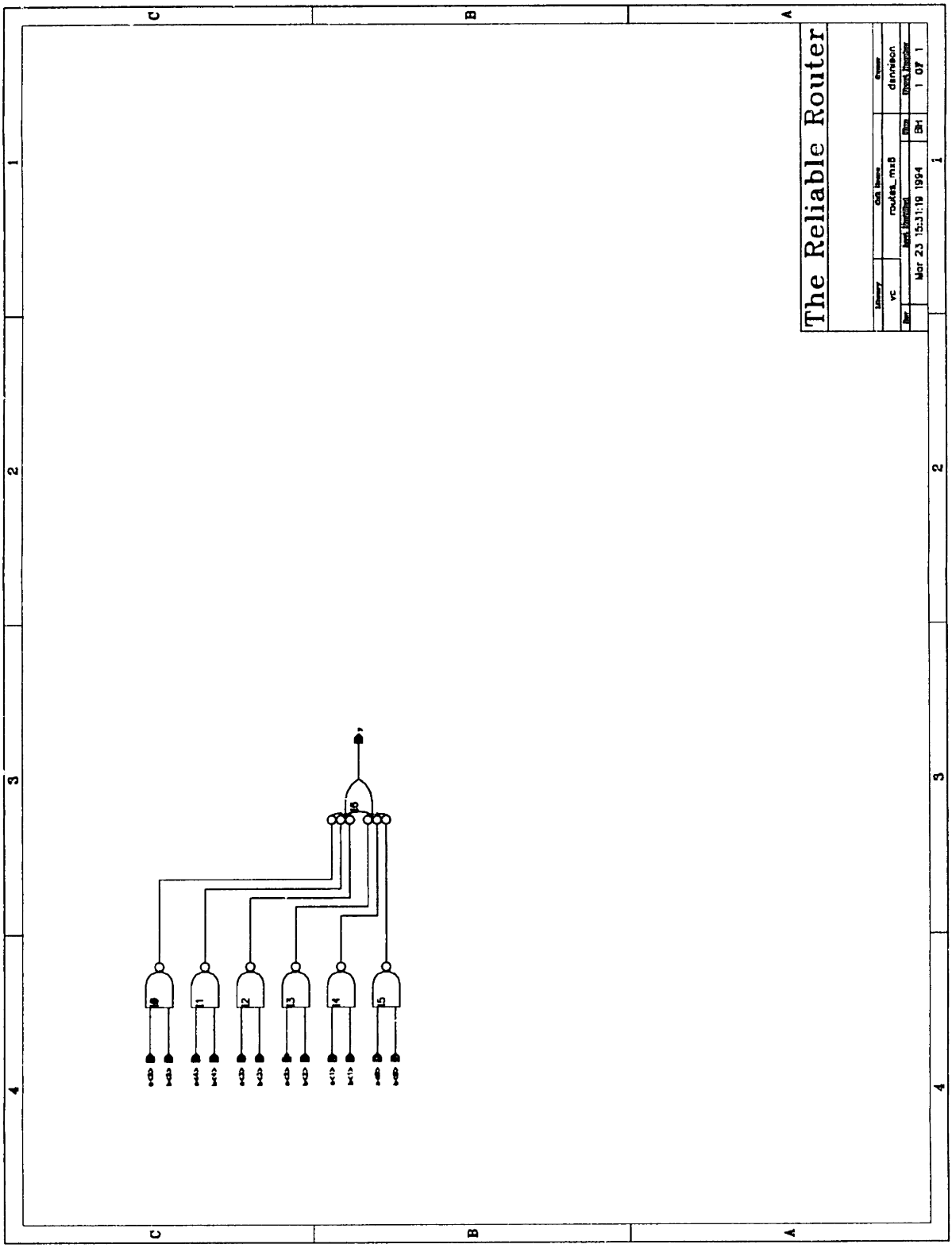


Figure A-209: Library vc, cell routes_mx6

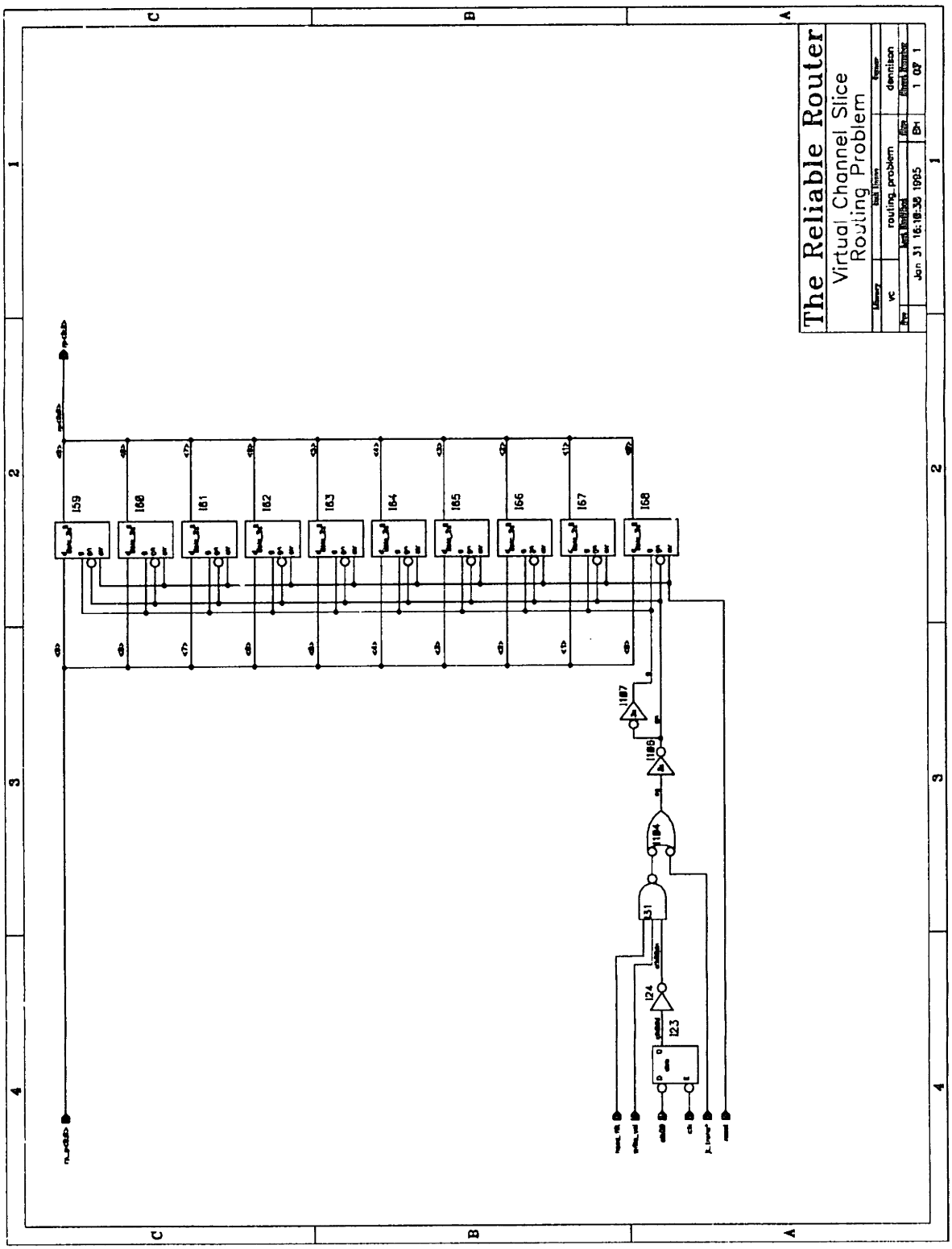


Figure A-210: Library vc, cell routing_problem

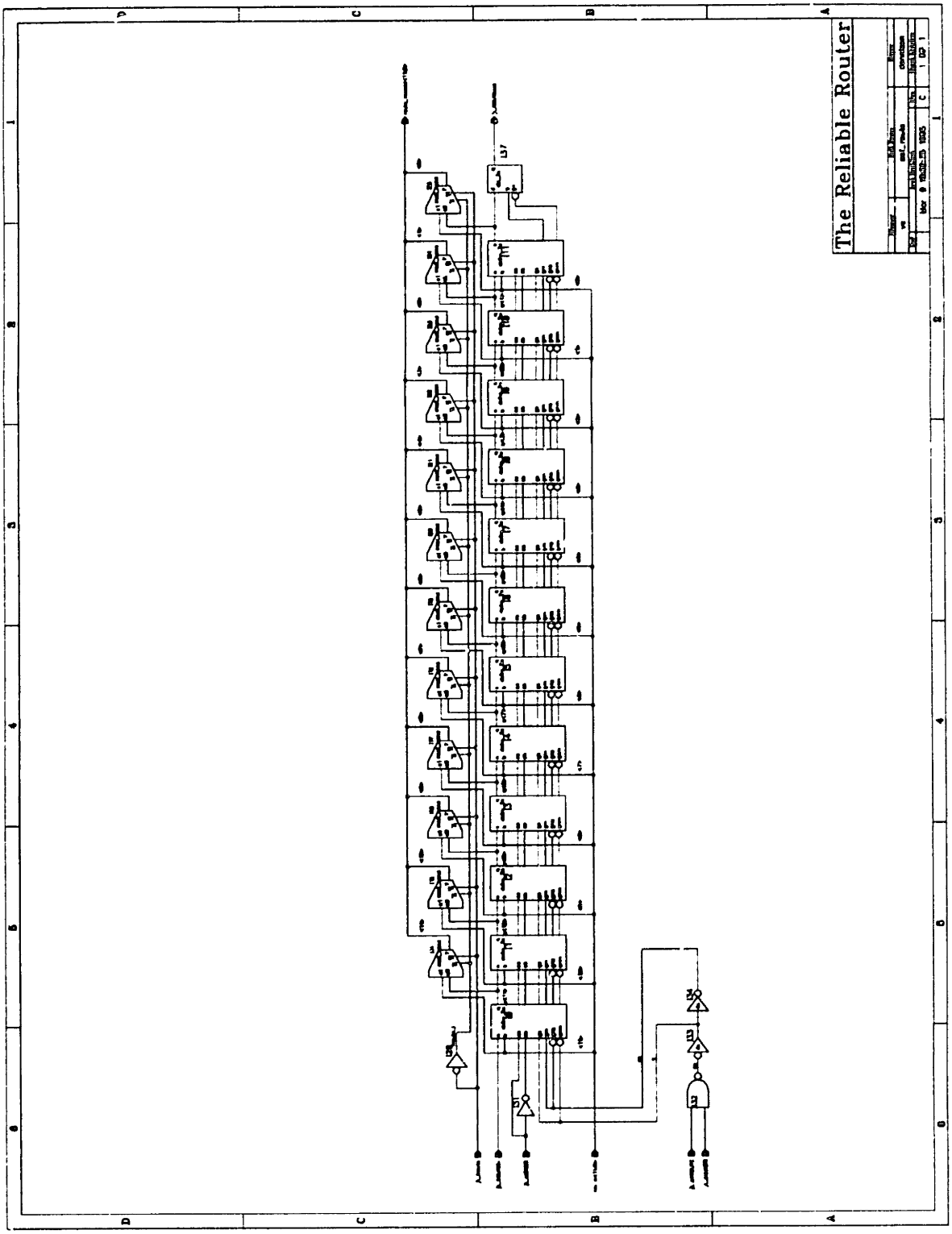


Figure A-21[1]: Library vc, cell saf_route

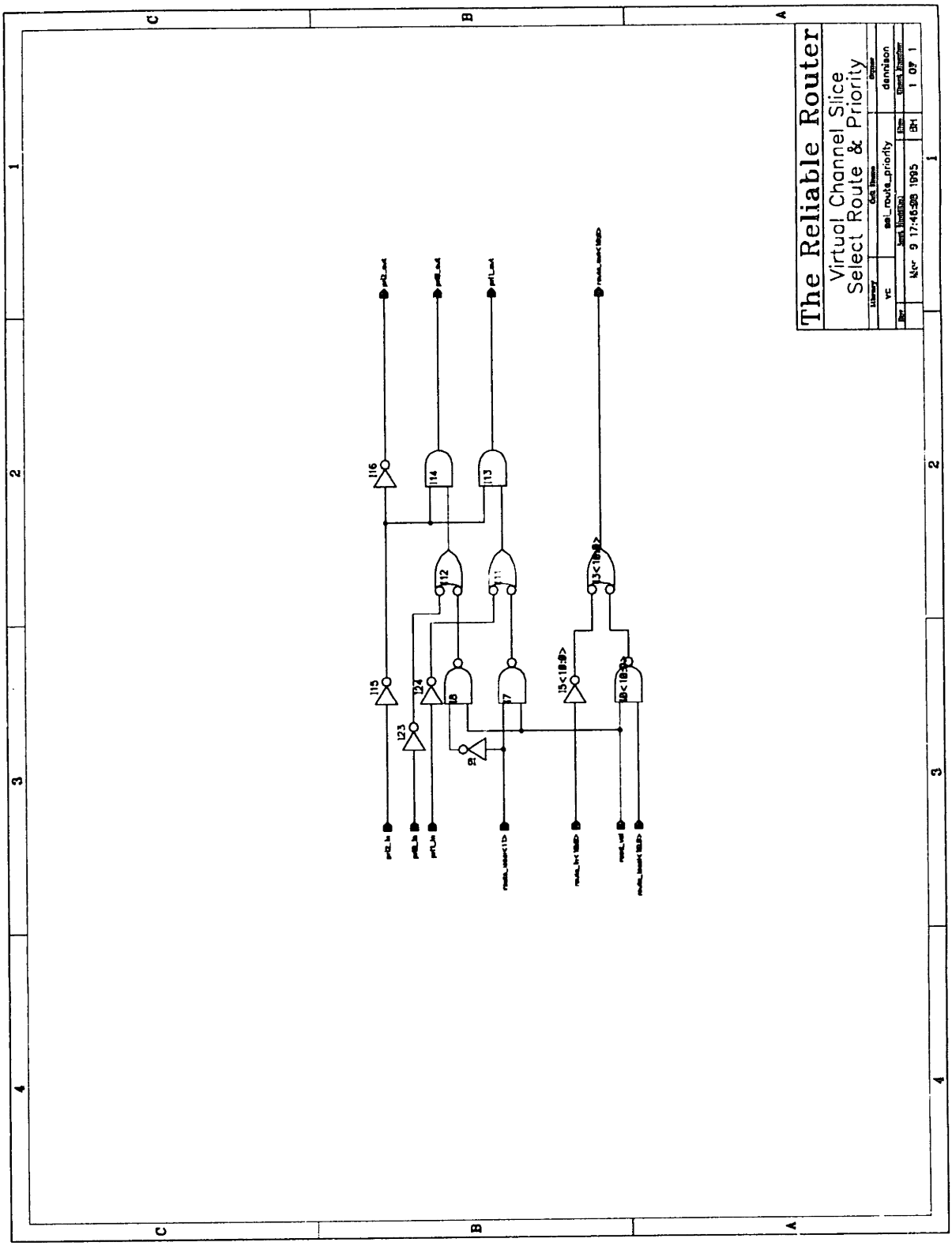


Figure A-212: Library vc, cell sel_route_priority

Figure A-213: Library vc, cell vc

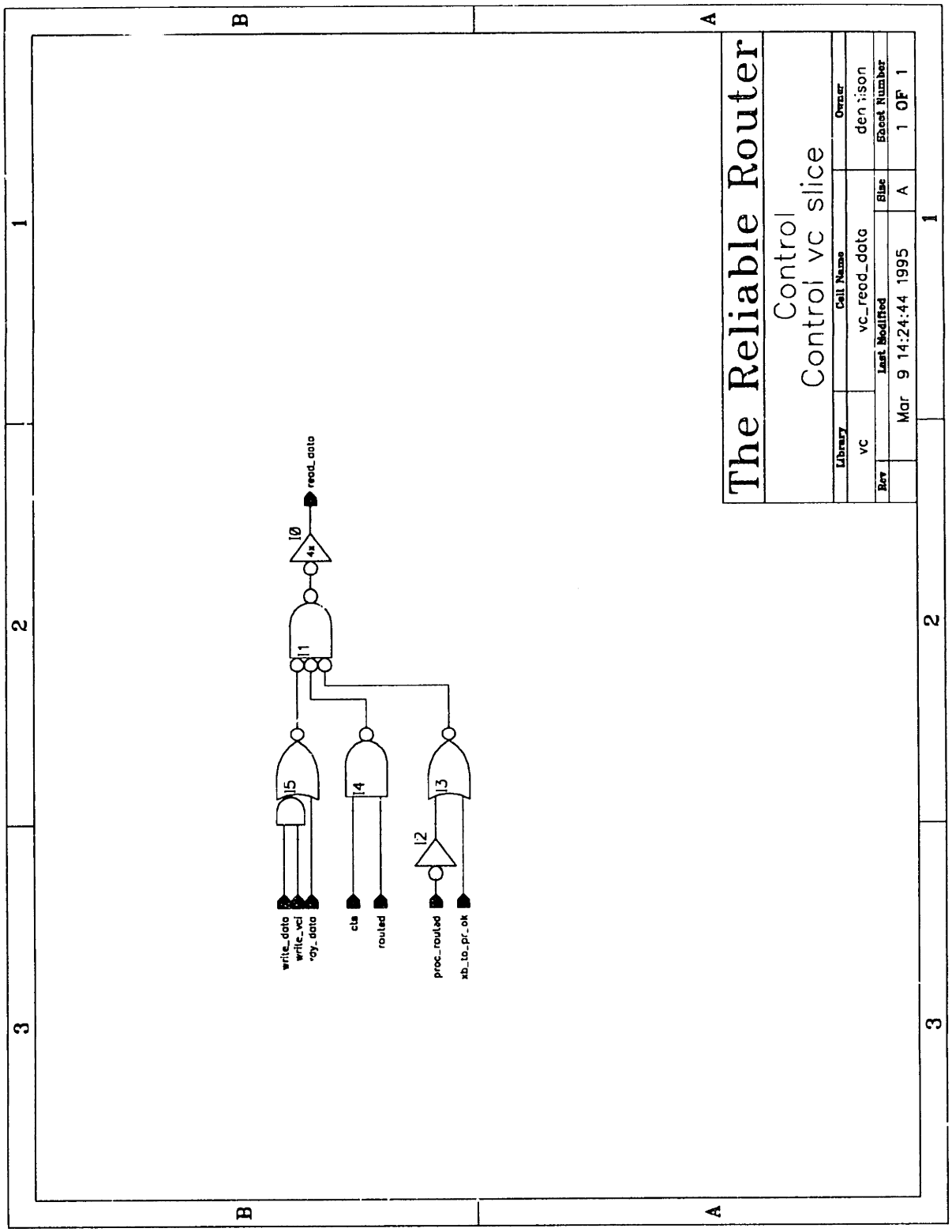


Figure A-214: Library vc, cell vc_read_data

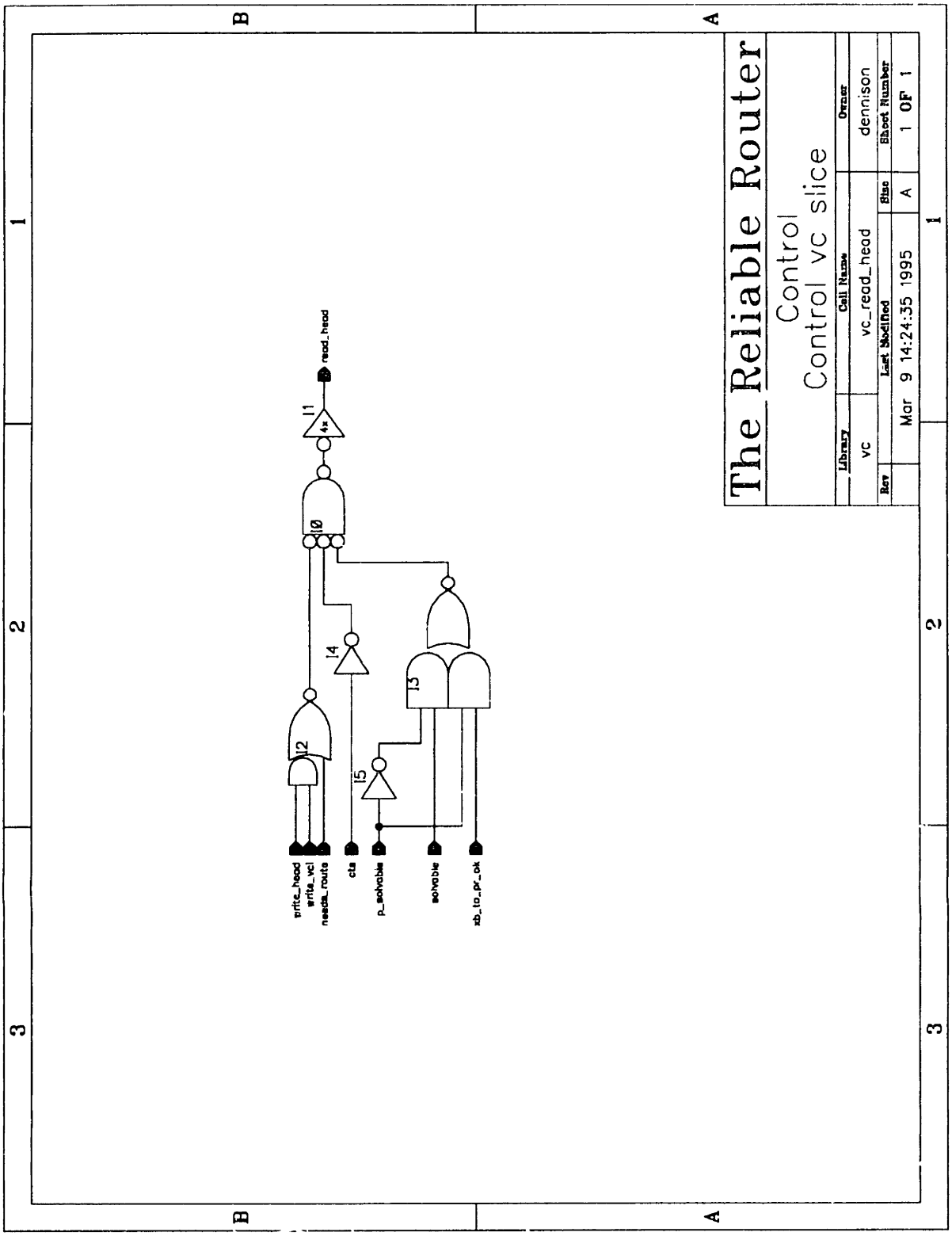


Figure A-215: Library vc, cell vc_read_head

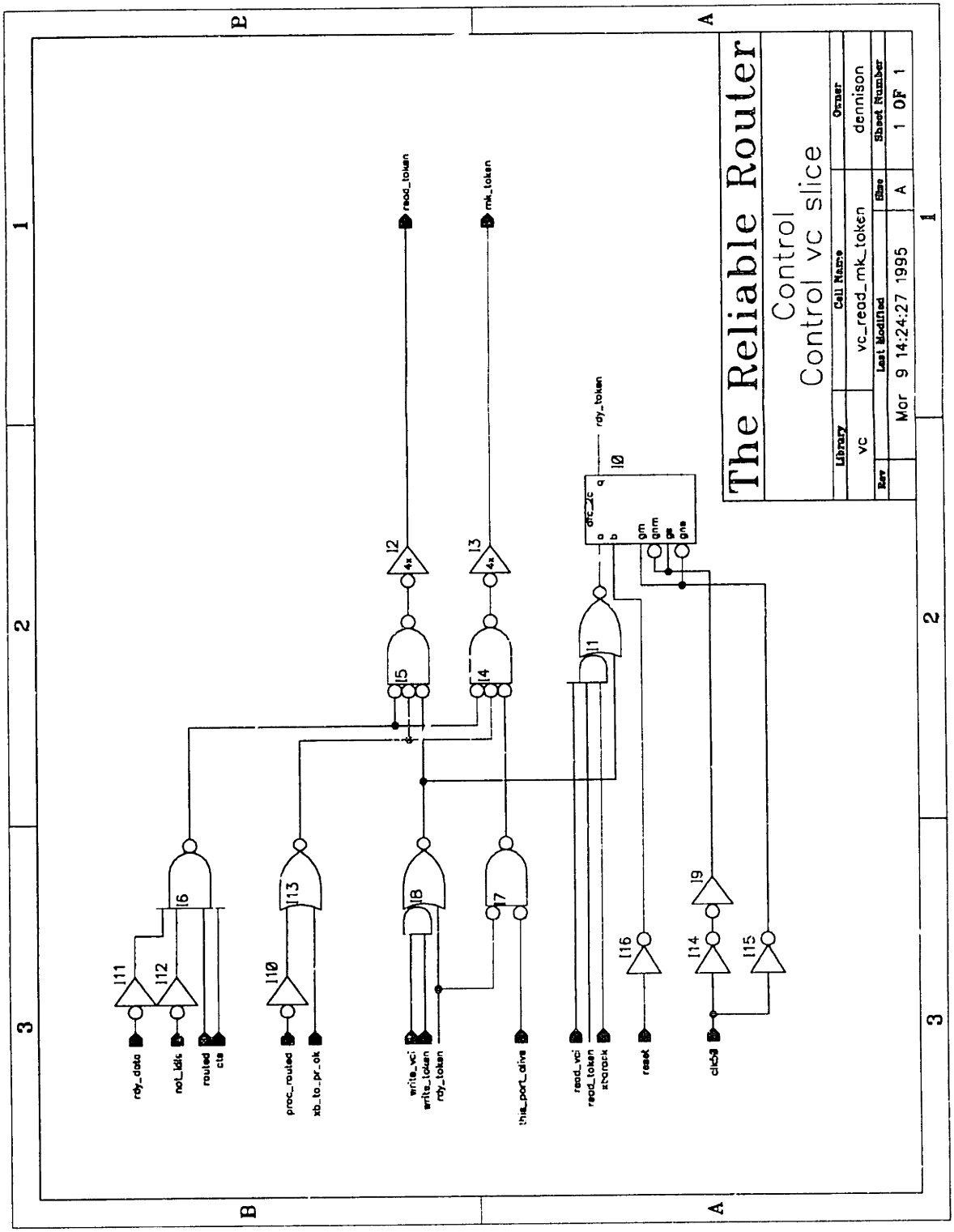


Figure A-216: Library vc, cell vc_read_mk_token

Appendix B

Reliable Router Standard Cell Schematics

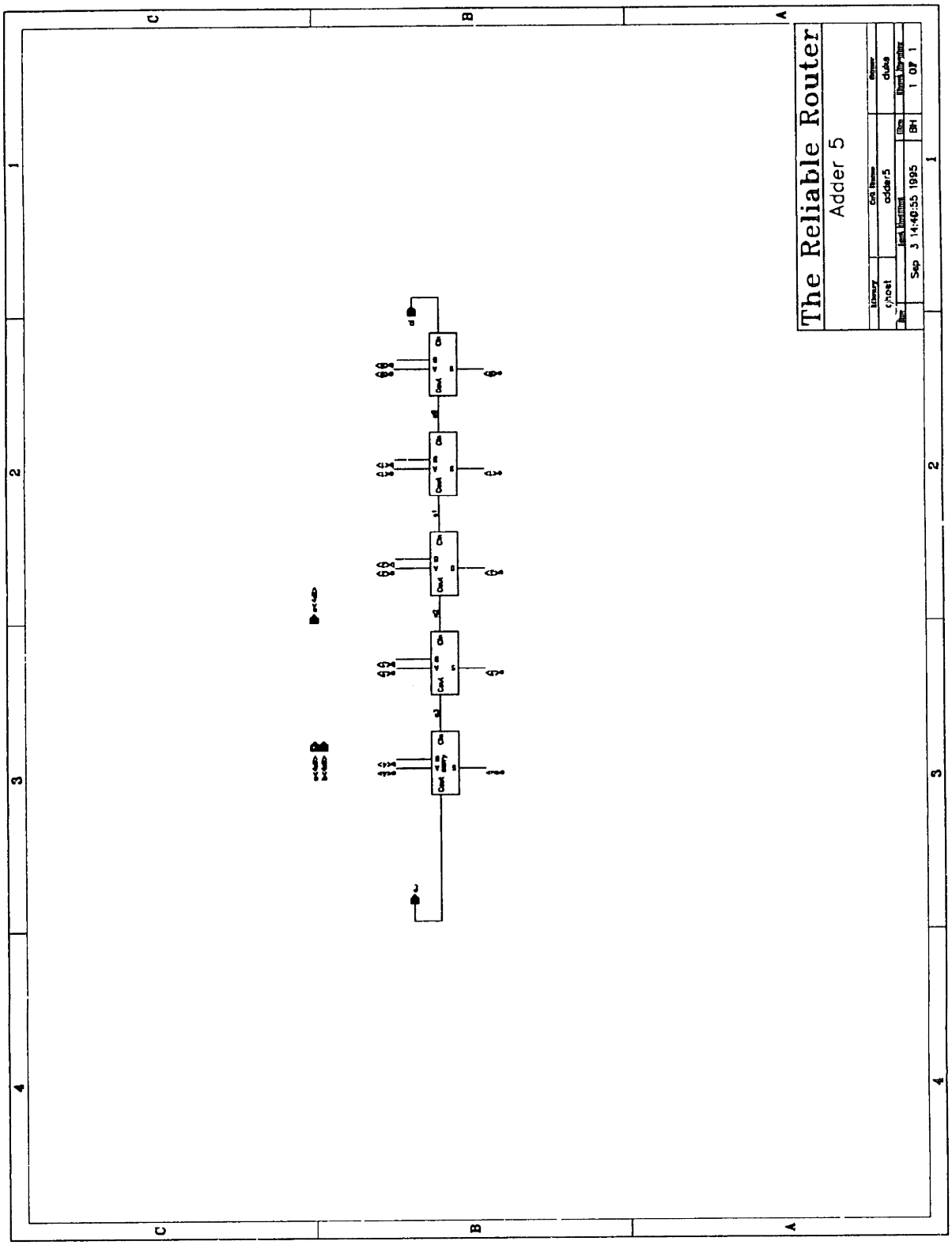


Figure B-1: Library ghost, cell adder5

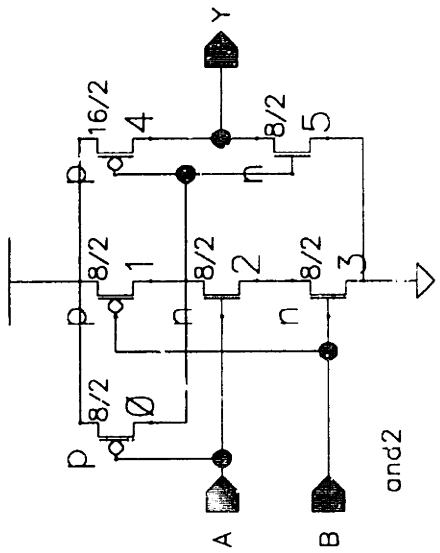


Figure B-2: Library ghost, cell and2

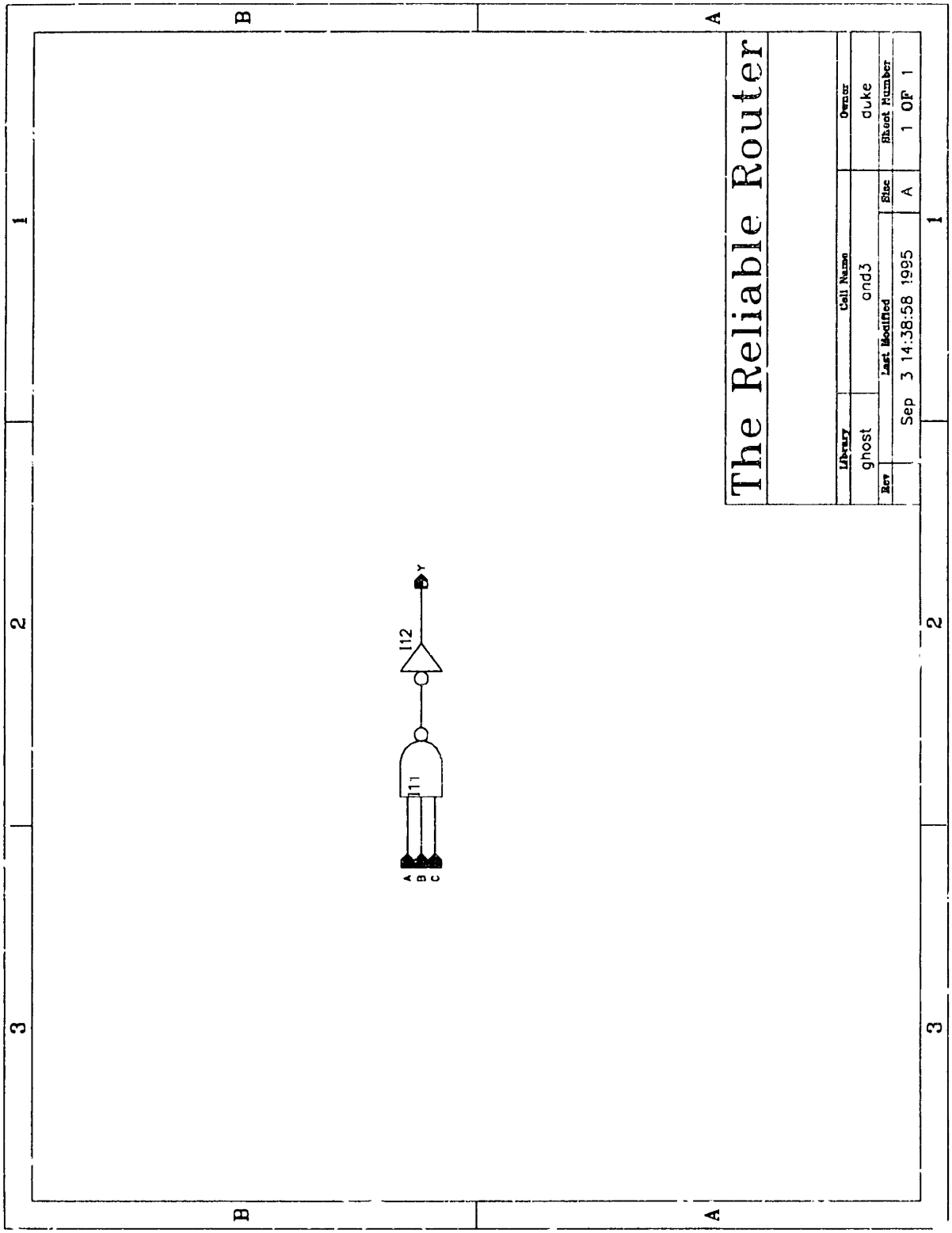


Figure B.3: Library ghost, cell and3

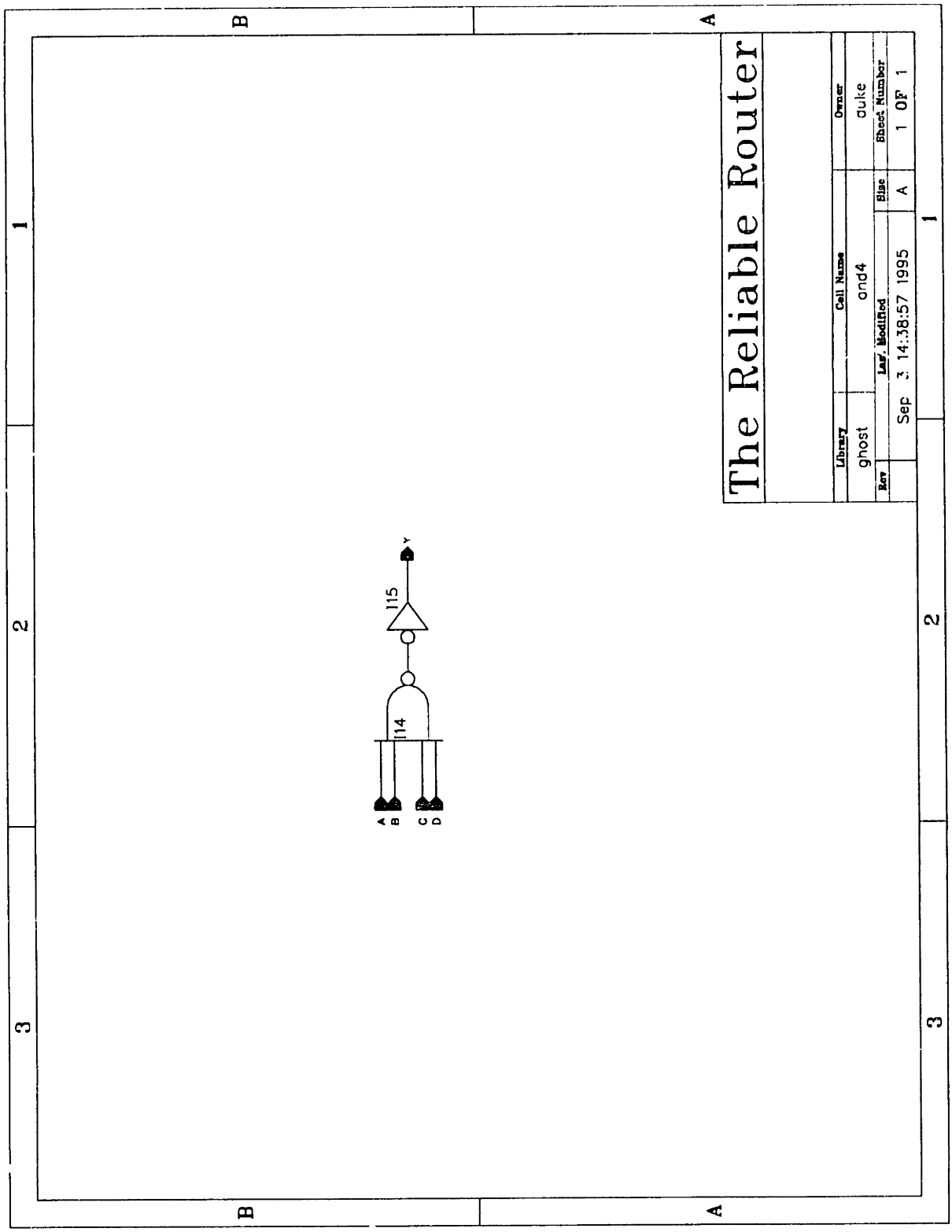


Figure B-4: Library ghost, cell and4

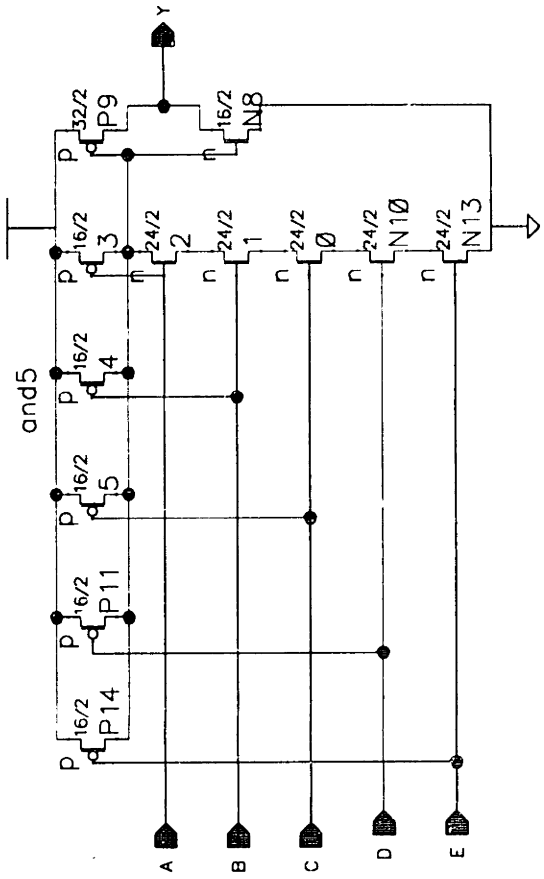


Figure B-5: Library ghost, cell and5

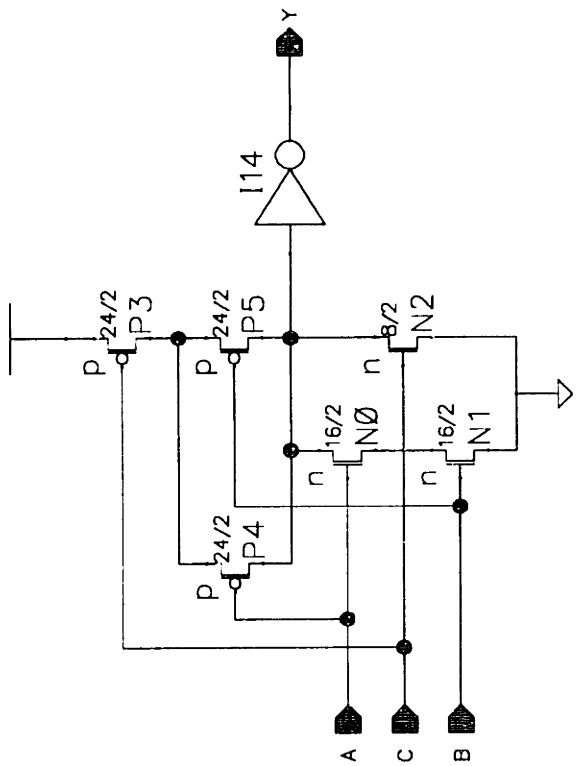


Figure B-6: Library ghost, cell ao12

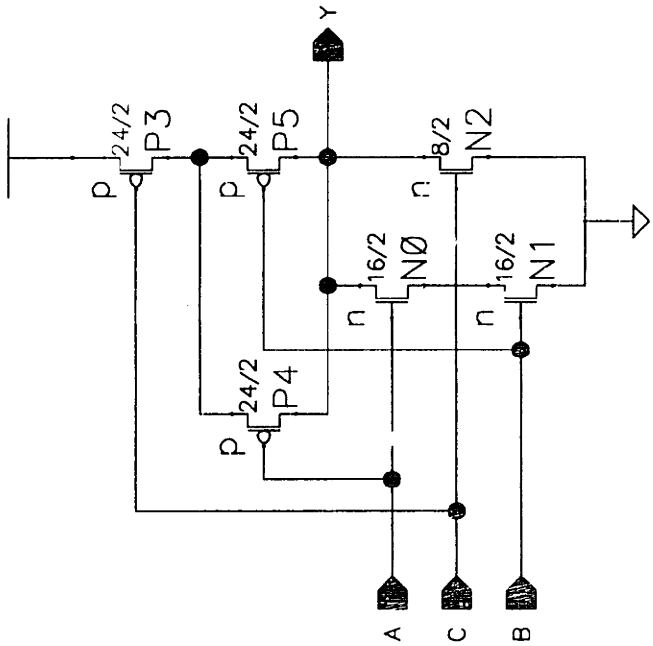


Figure B-7: Library ghost, cell aoi12

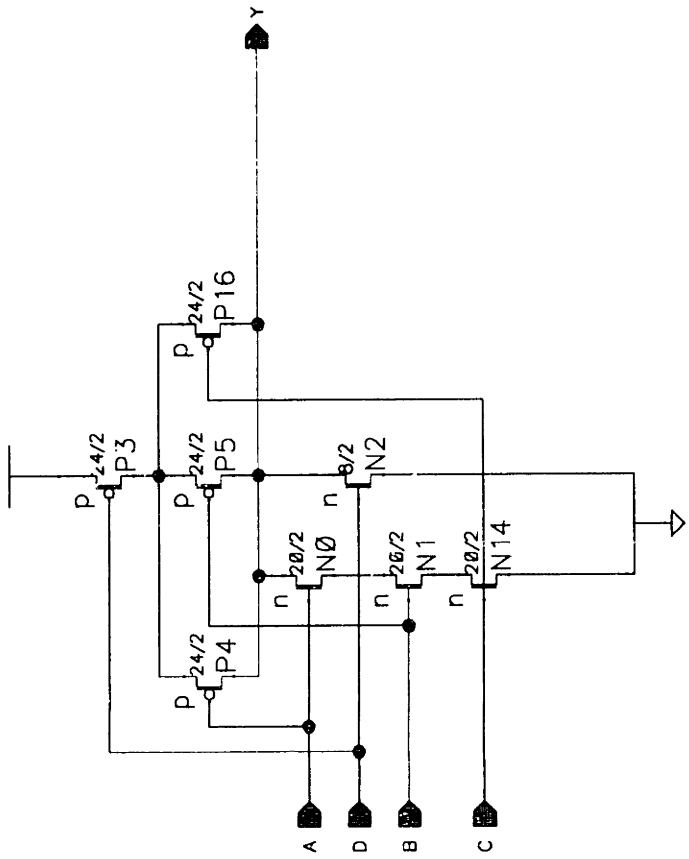


Figure B-8: Library ghost, cell aoi13

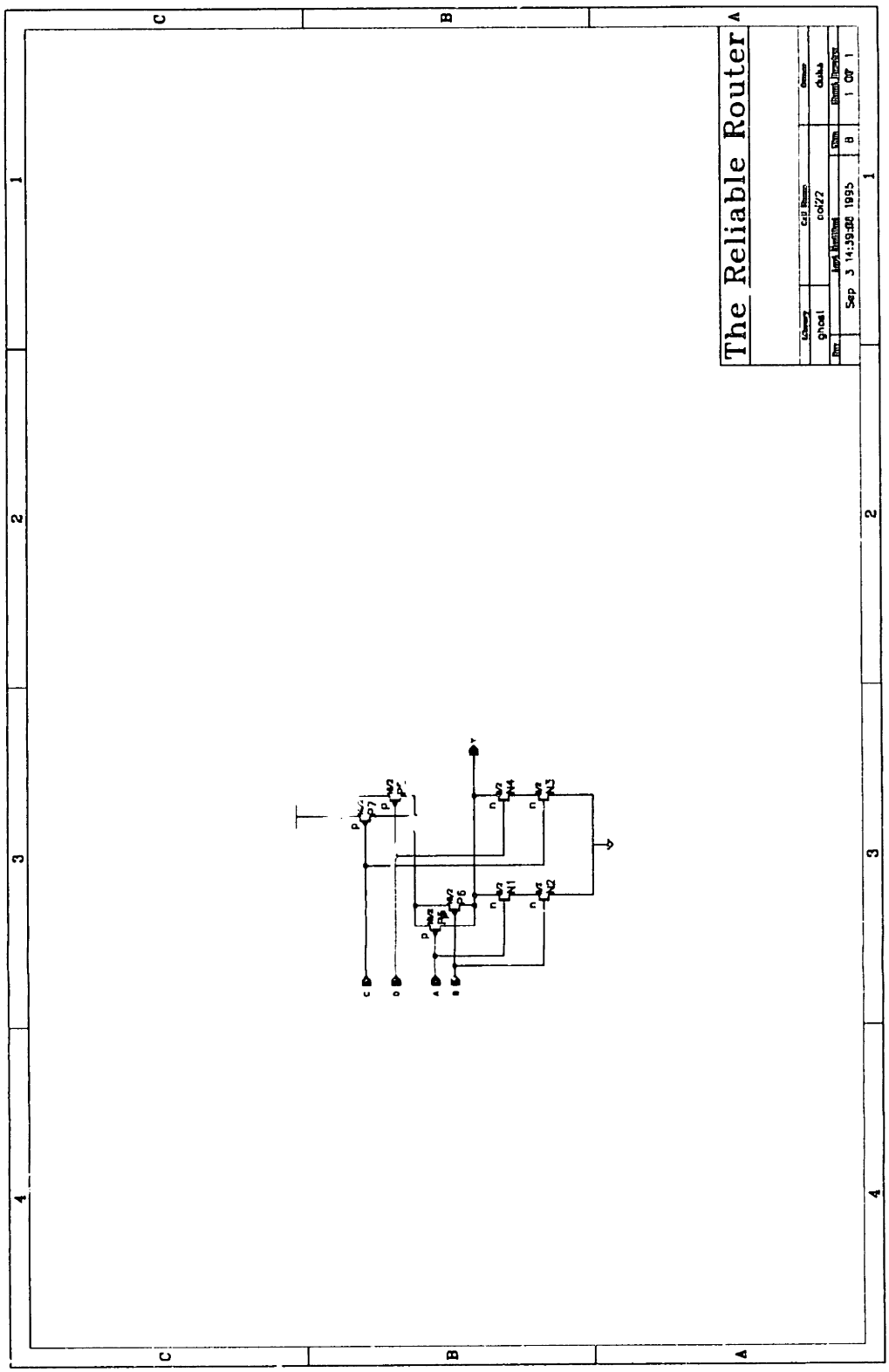


Figure B-9: Library ghost, cell aoi22

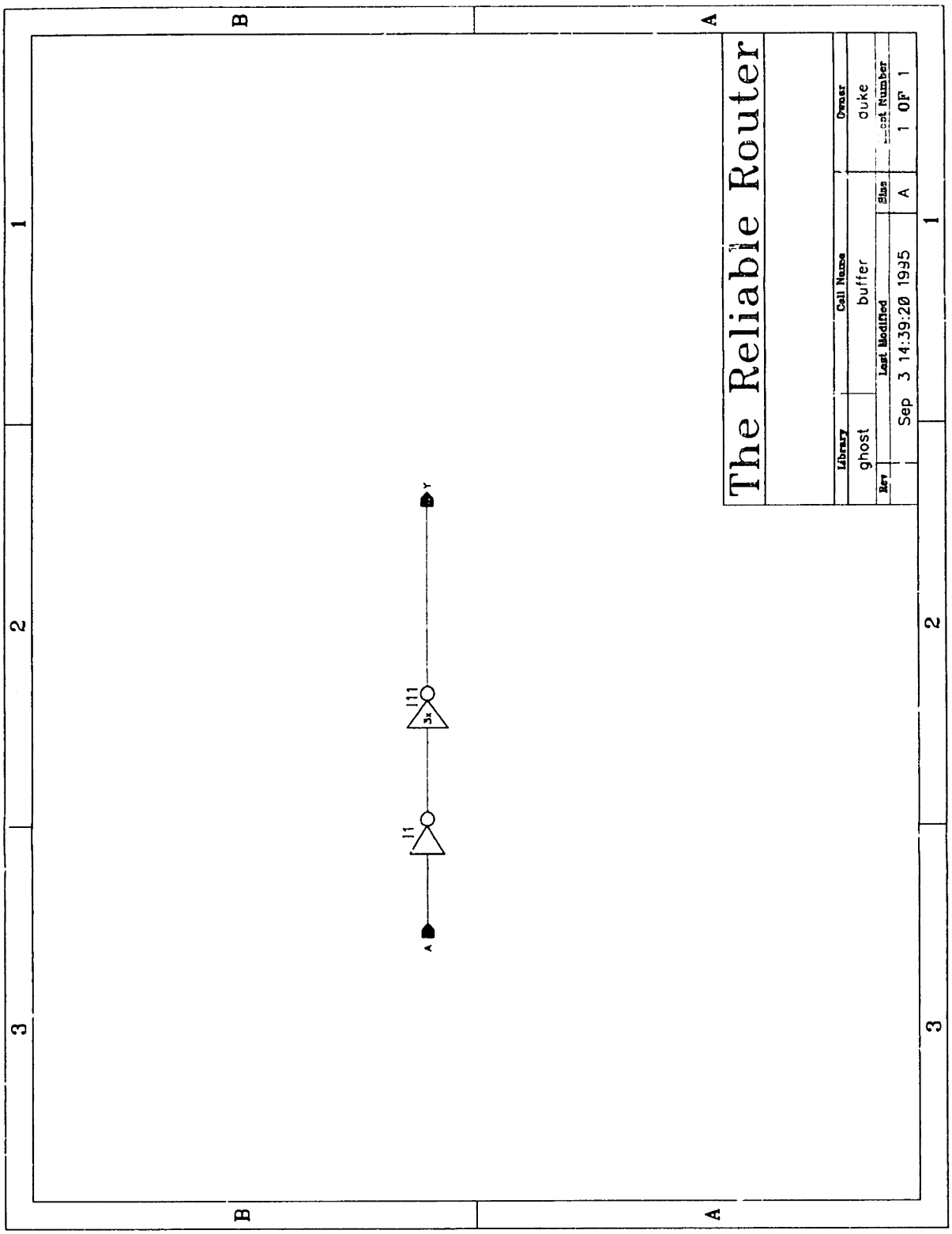


Figure B-10: Library ghost, cell buffer

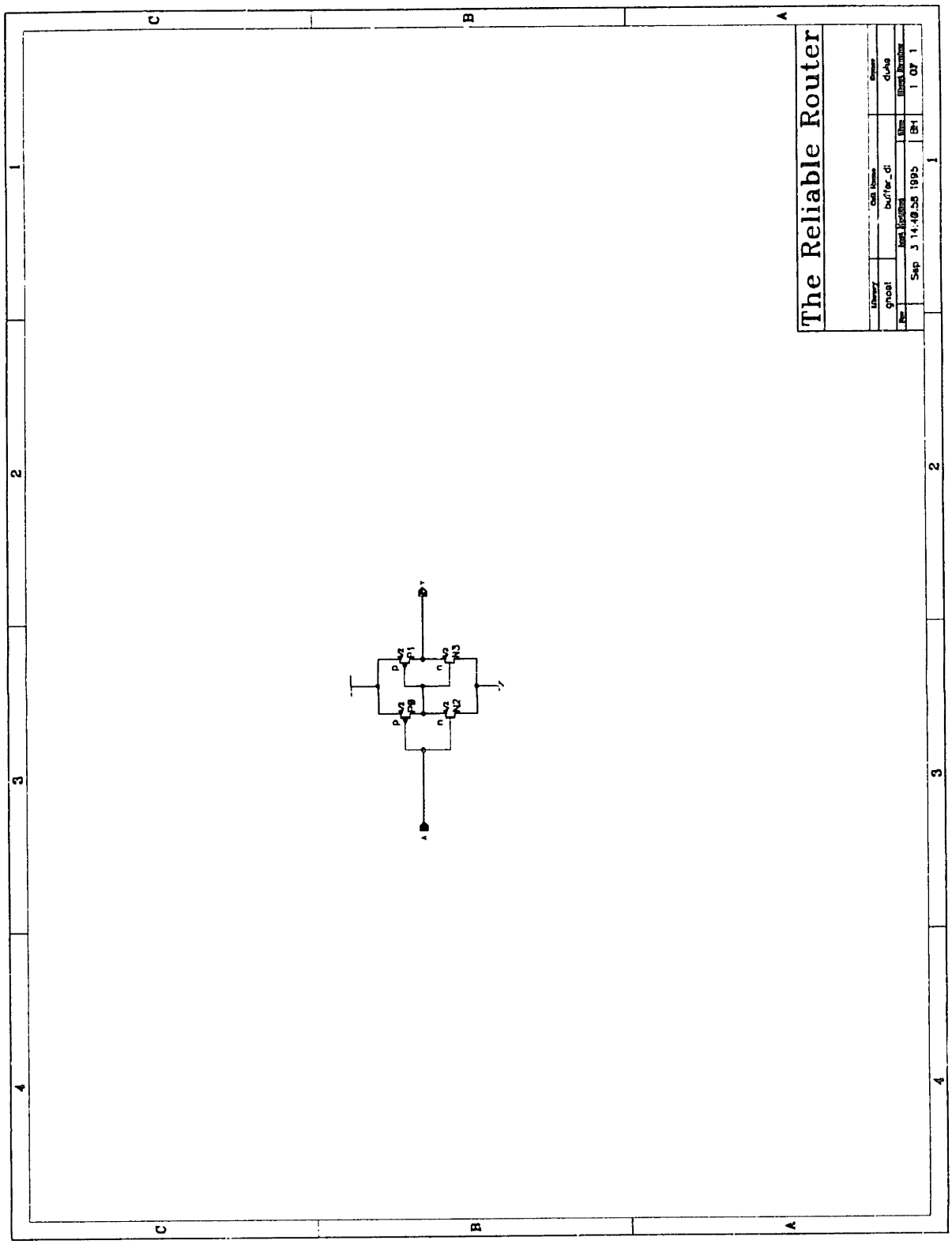


Figure B-11: Library ghost, cell buffer_dl

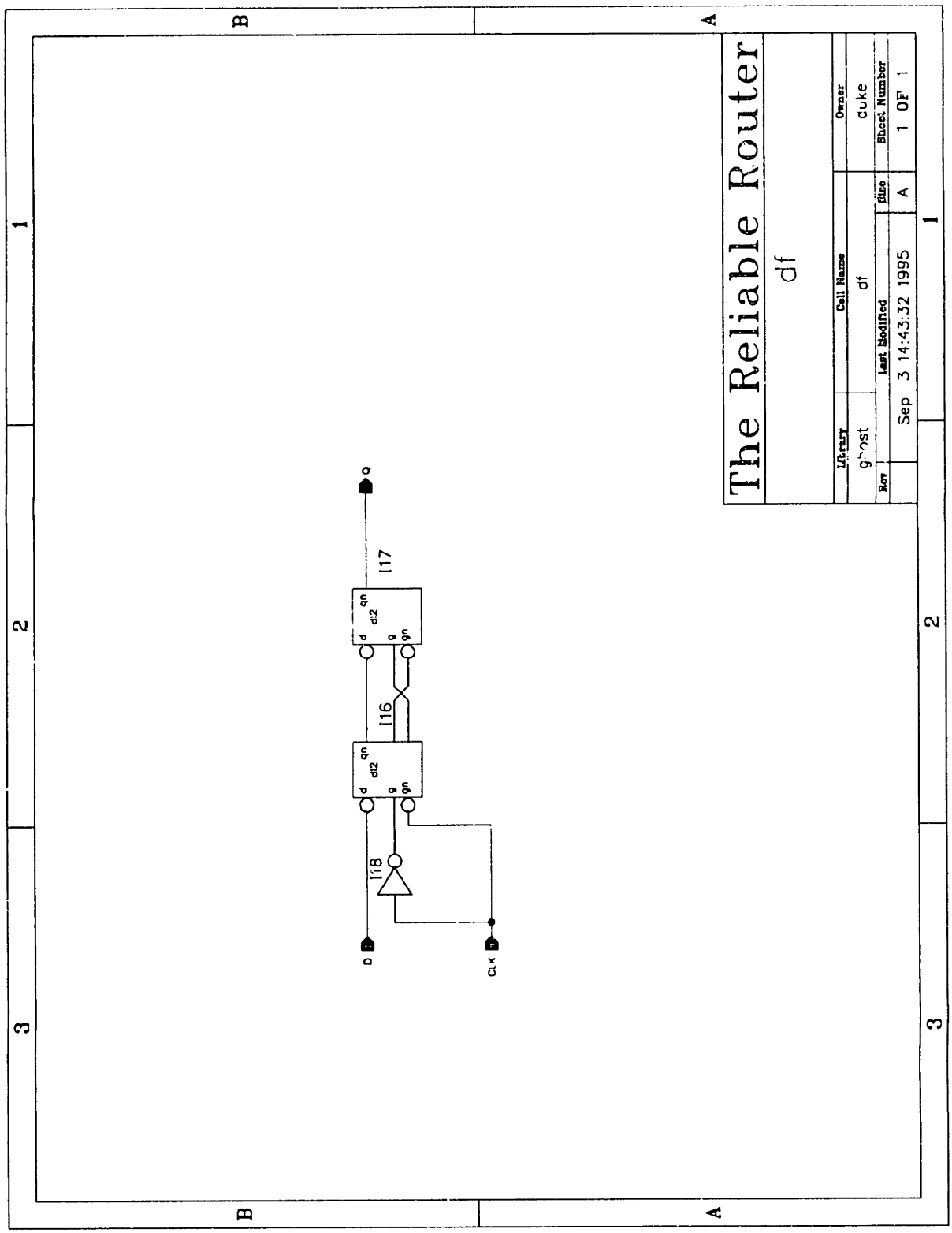


Figure B-12: Library ghost, cell df

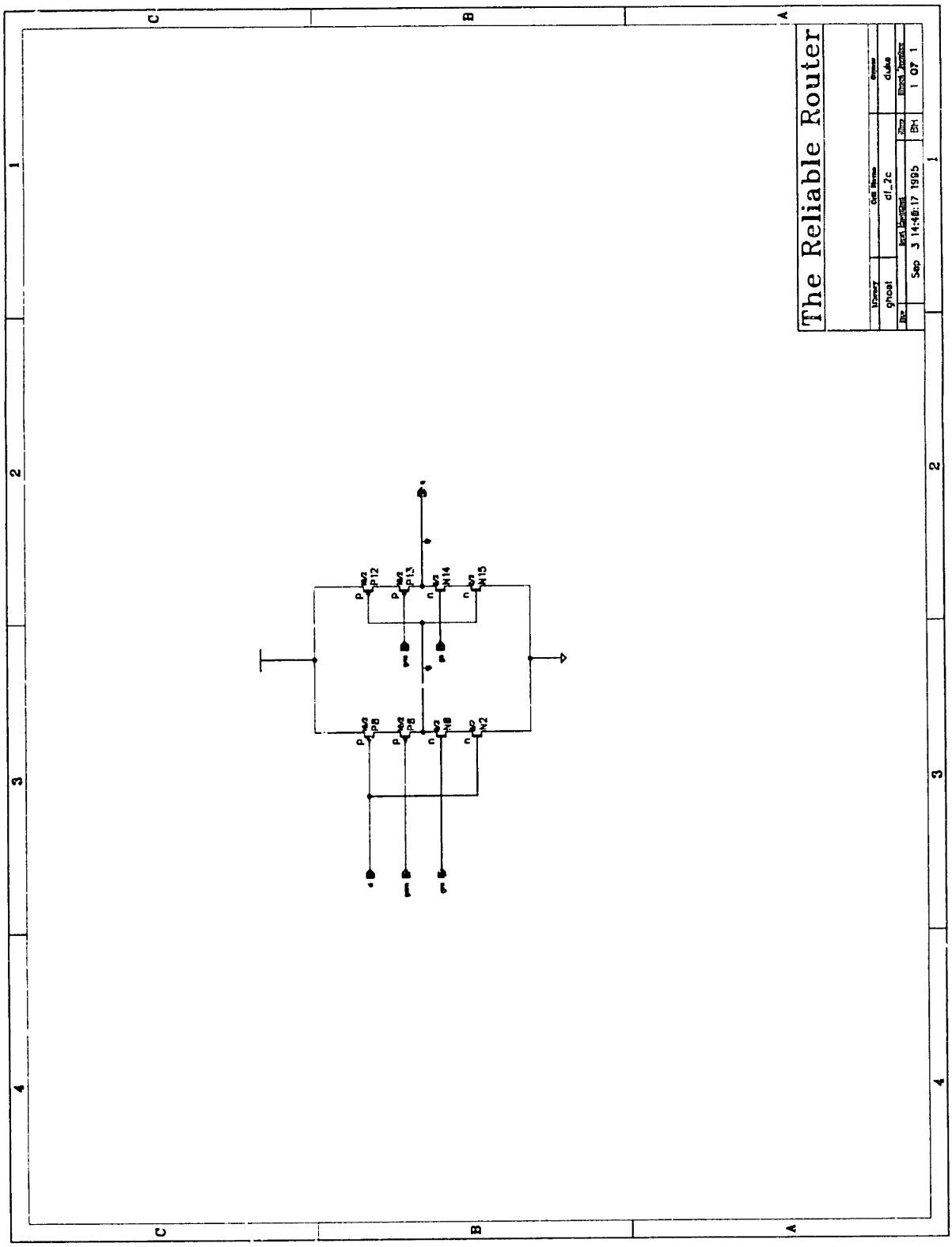


Figure B-13: Library ghost, cell df_2c

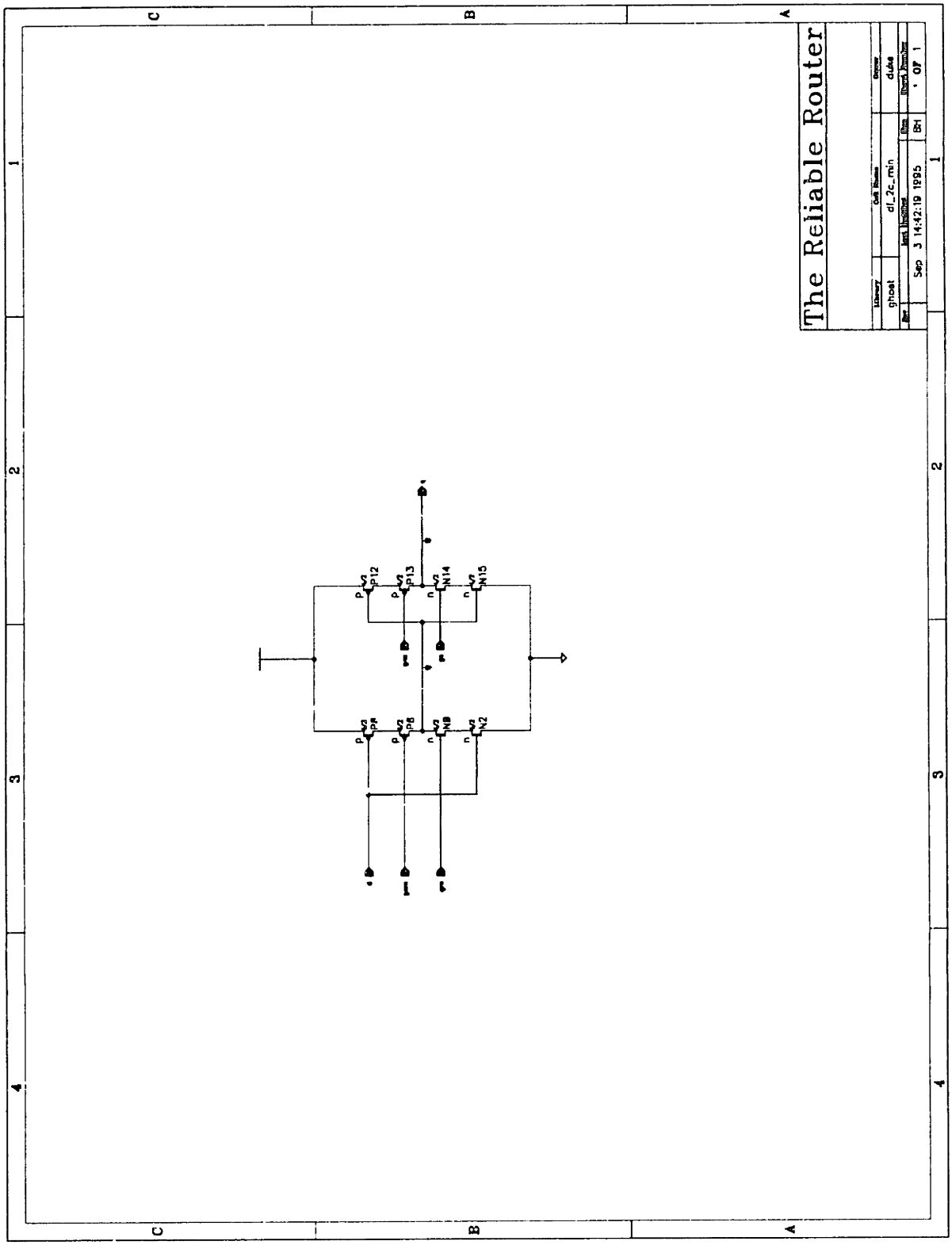


Figure B-14: Library ghost, cell df_2c_min

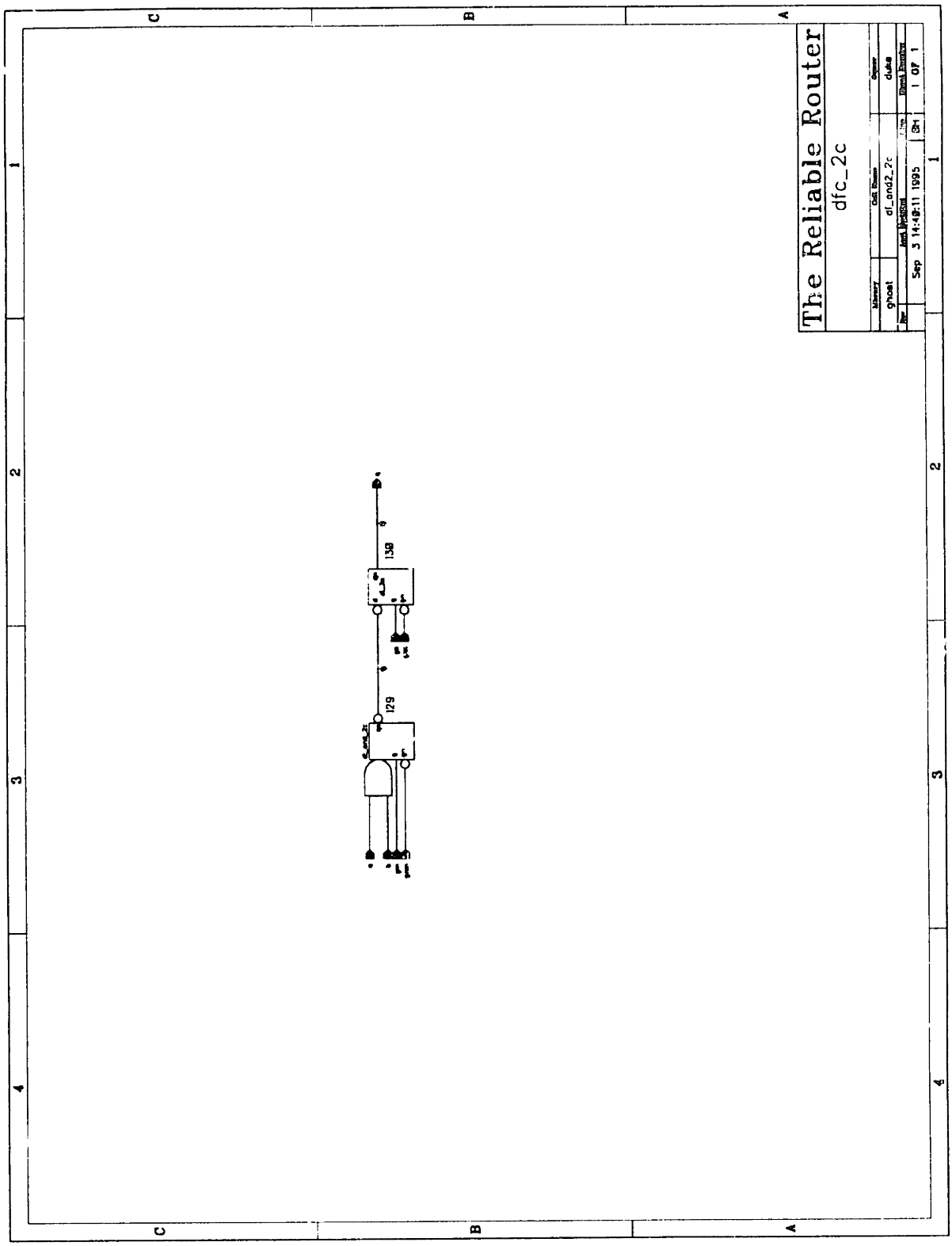


Figure B-15: Library ghost, cell df_and2_2c

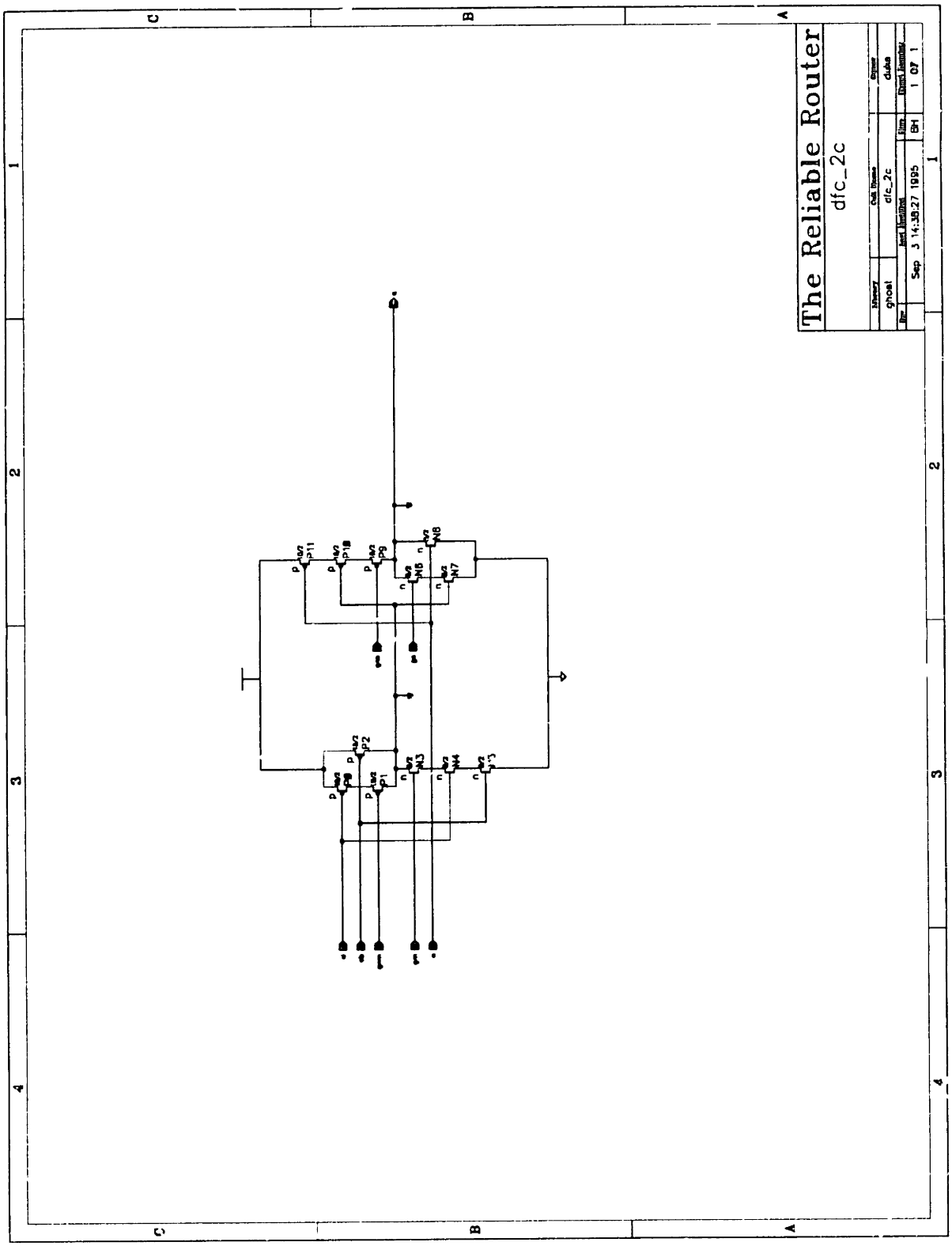


Figure B-16: Library ghost, cell dfc_2c

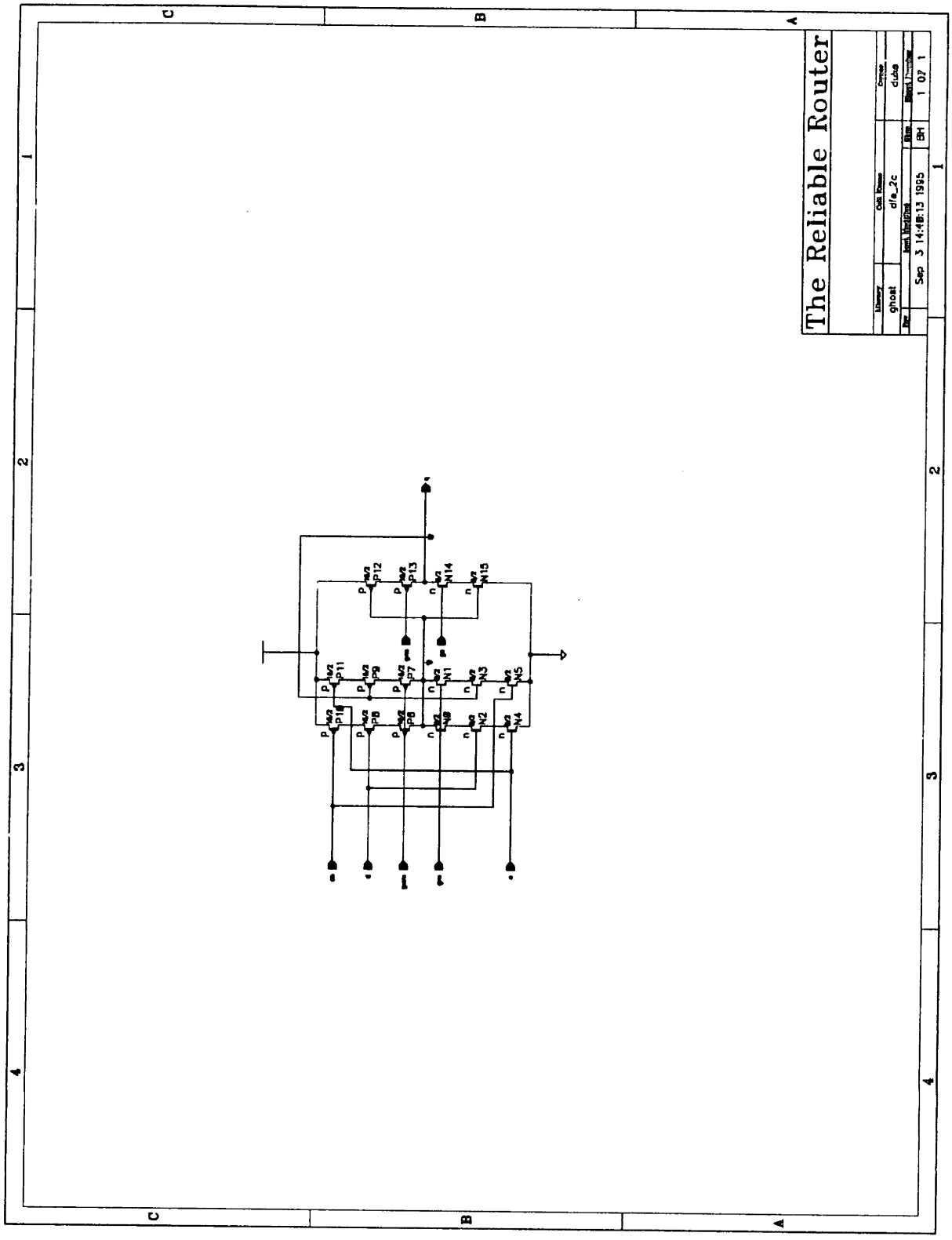


Figure B-17: Library ghost, cell dfe_2c

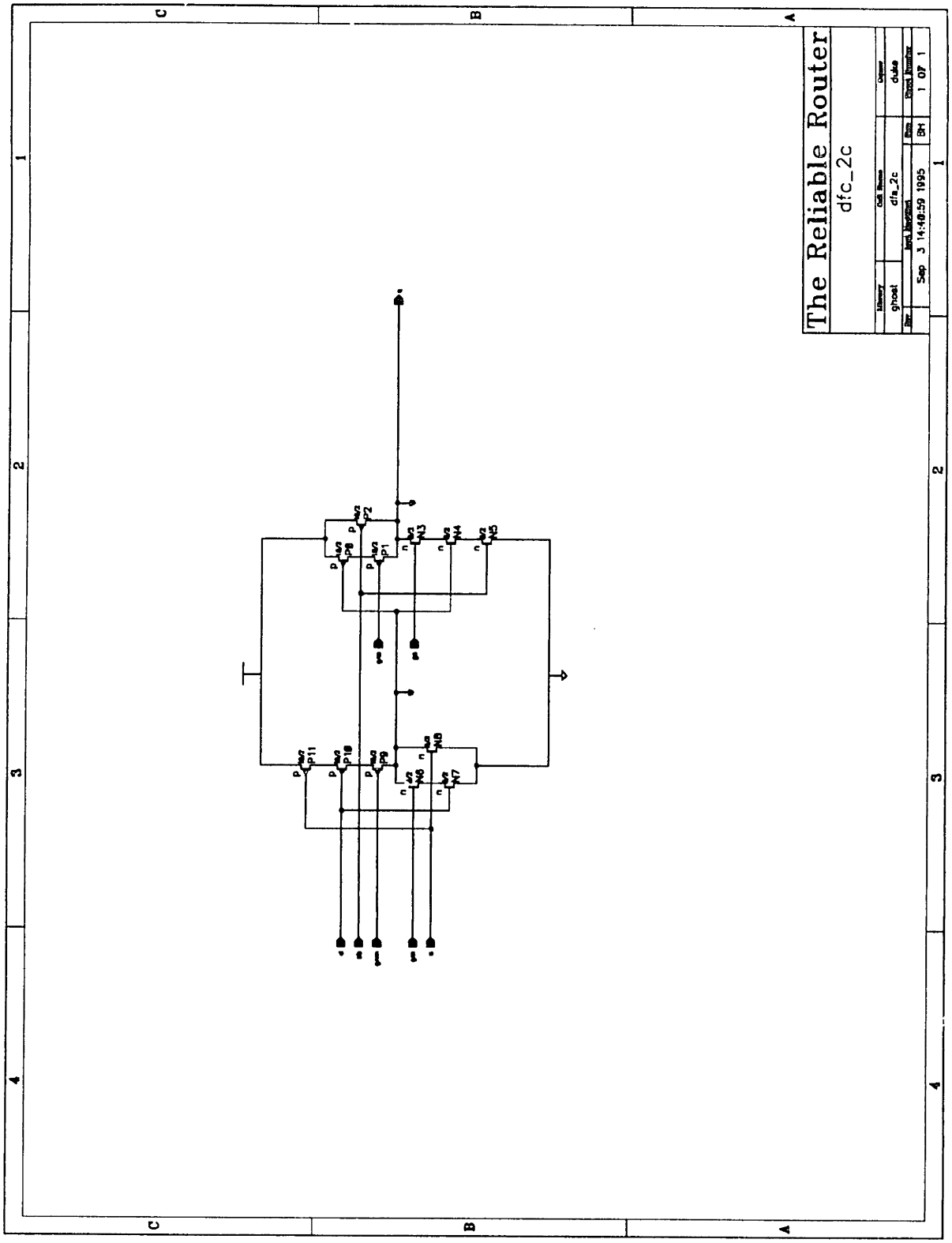


Figure B-18: Library ghost, cell dfs_2c

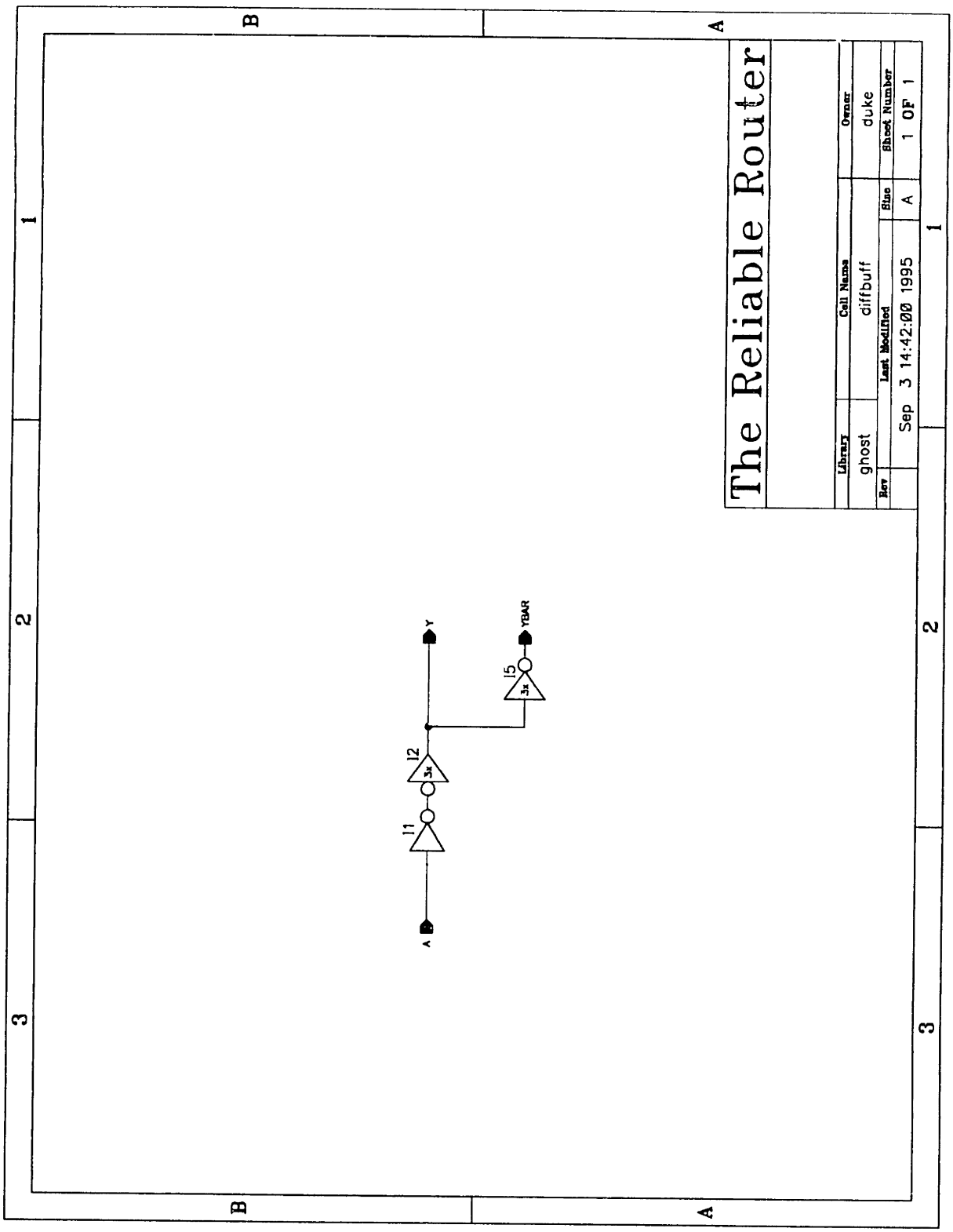


Figure B-19: Library ghost, cell diffbuff

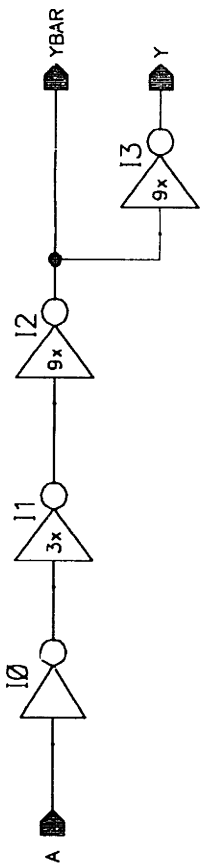


Figure B-20: Library ghost, cell diffbuff9x

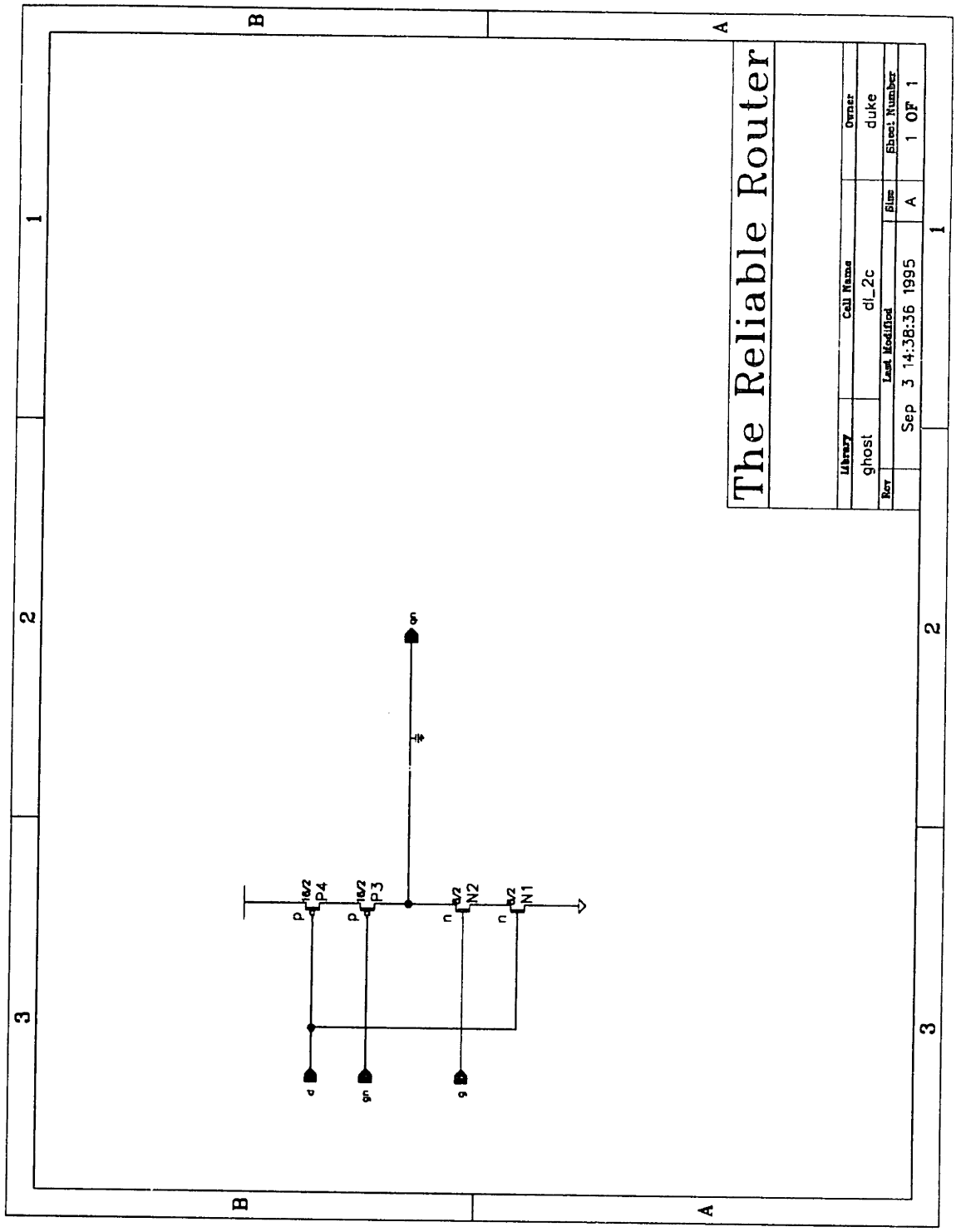


Figure B-21: Library ghost, cell dl_2c

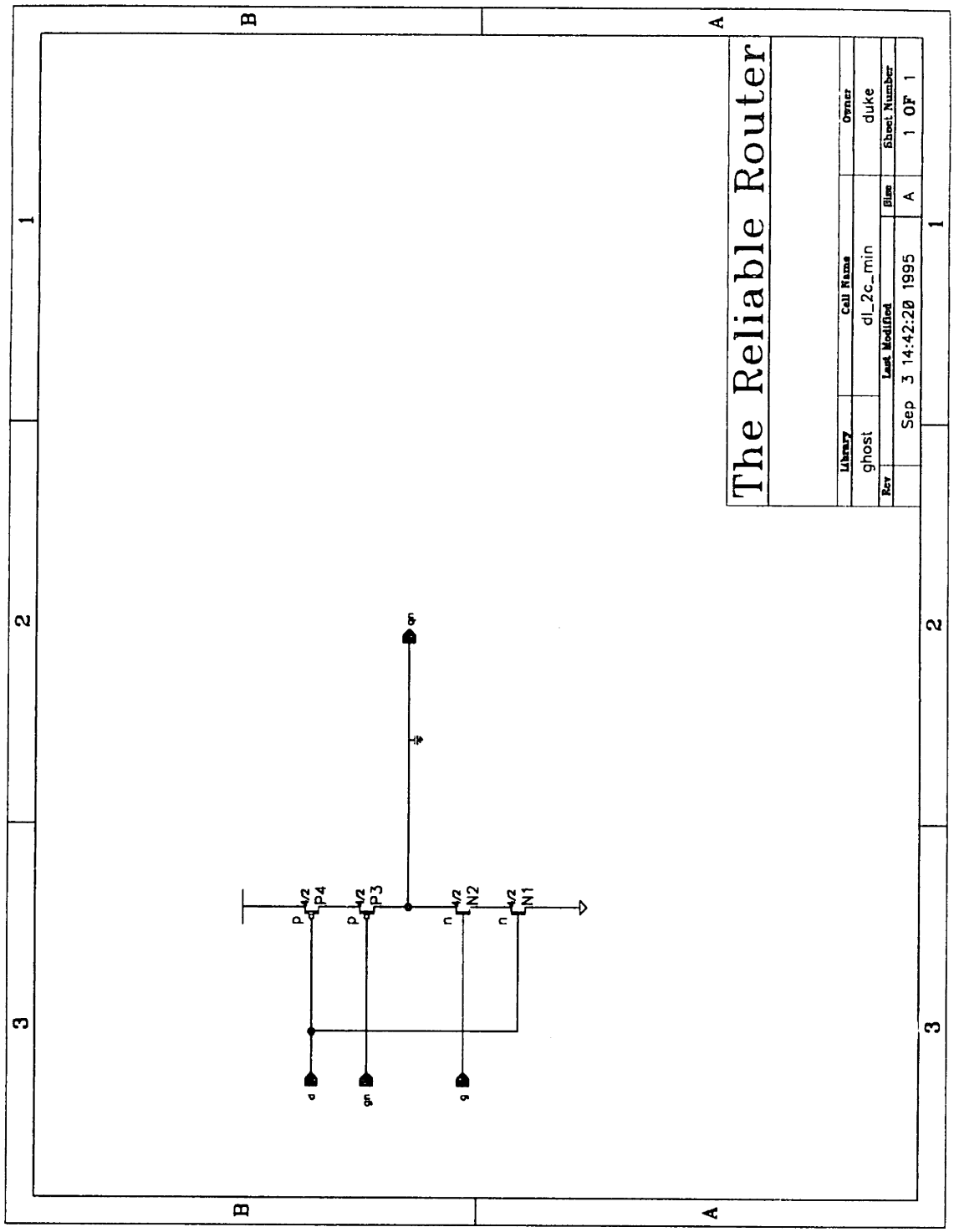


Figure B-22: Library ghost, cell dl_2c_min

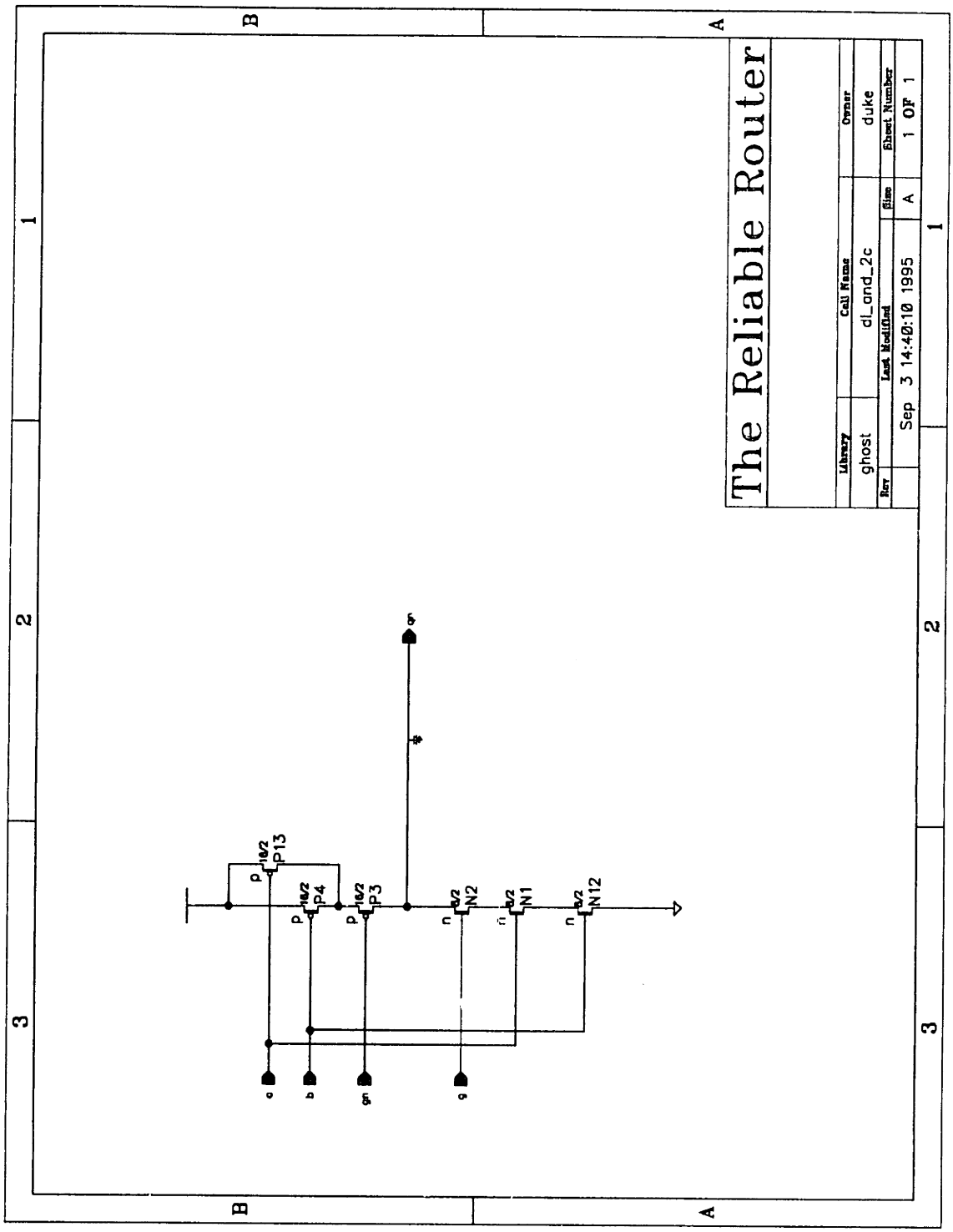


Figure B-23: Library ghost, cell dl_and_2c

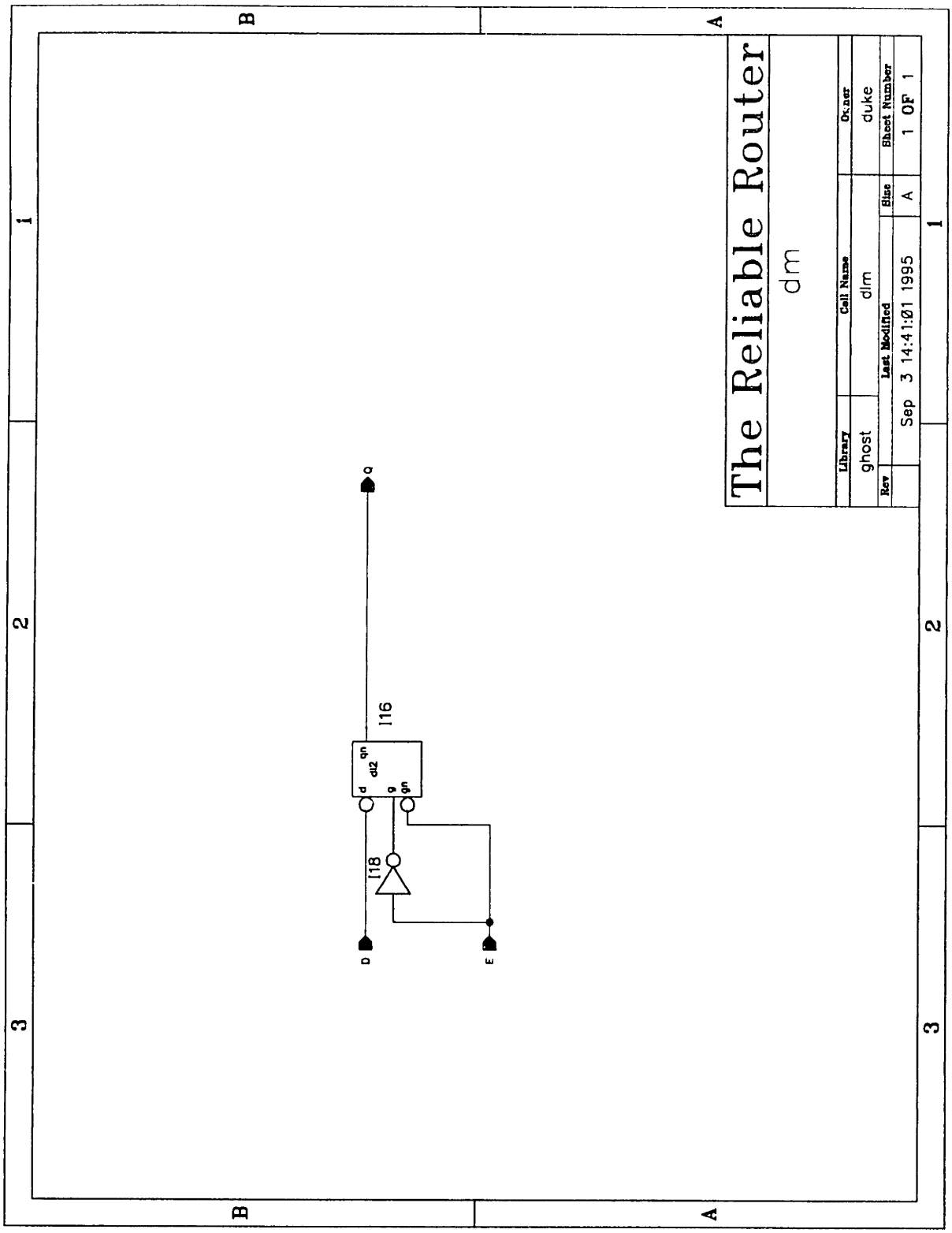


Figure B-24: Library ghost, cell dlm

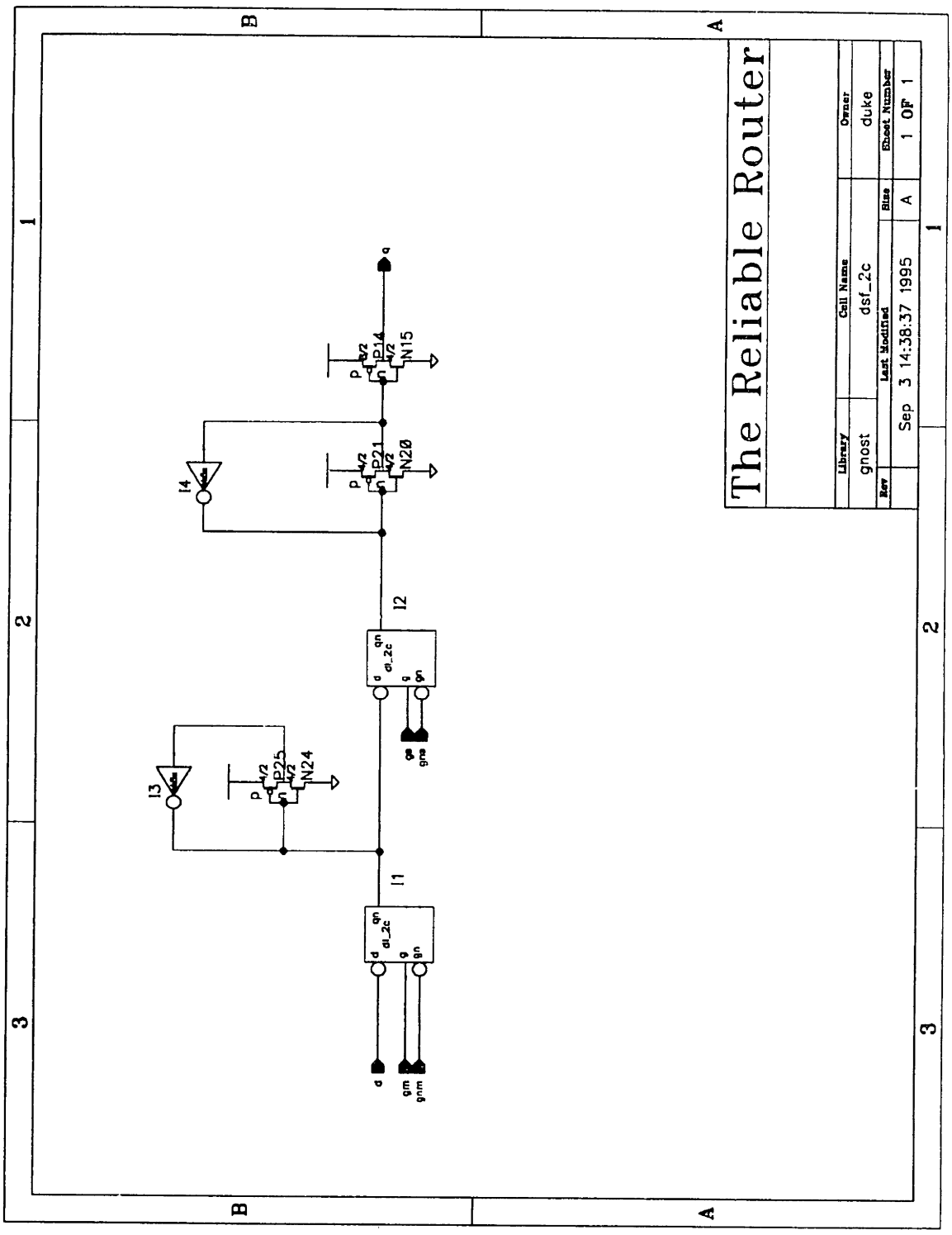


Figure B-25: Library ghost, cell dsf_2c

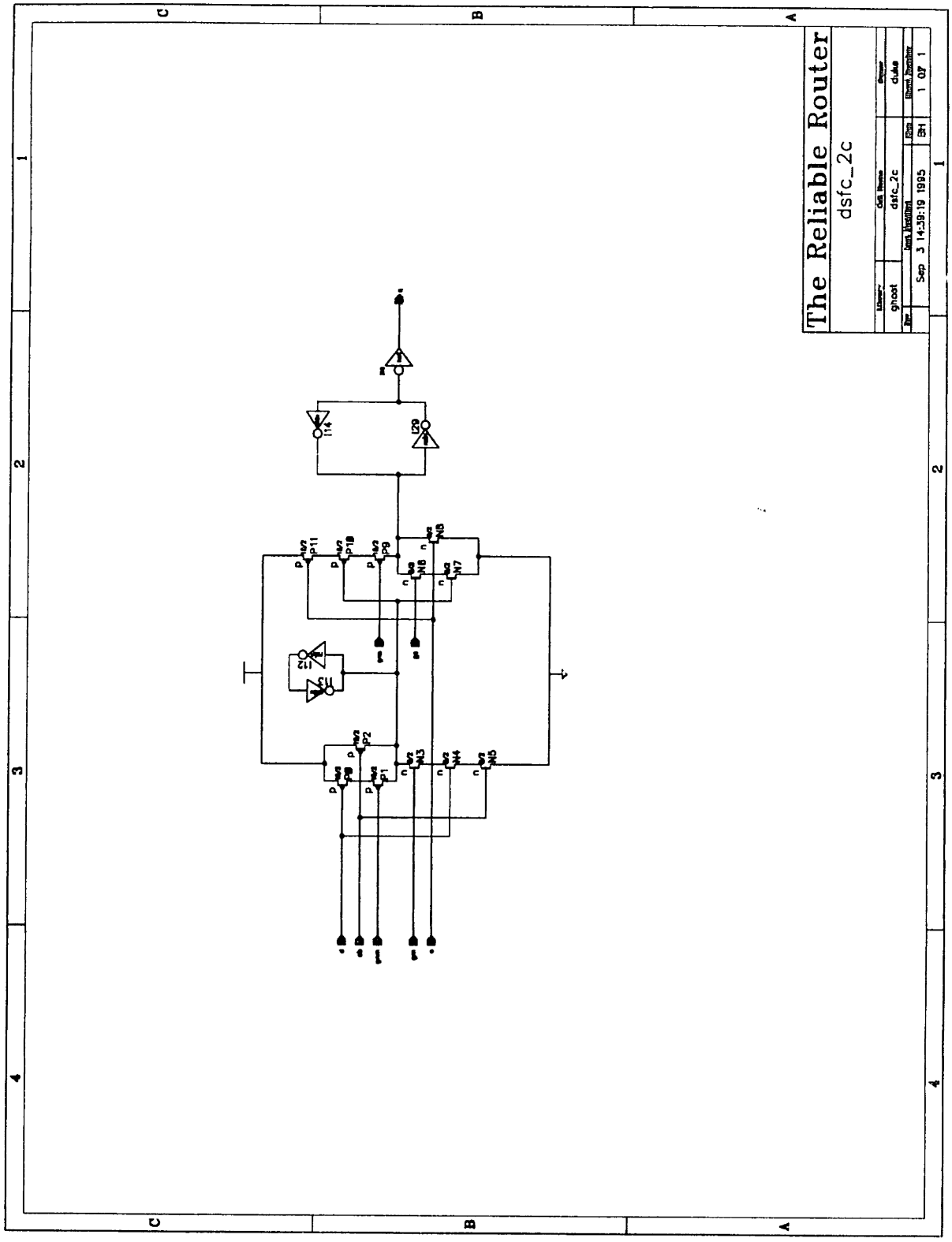


Figure B-26: Library ghost, cell dsfc_2c

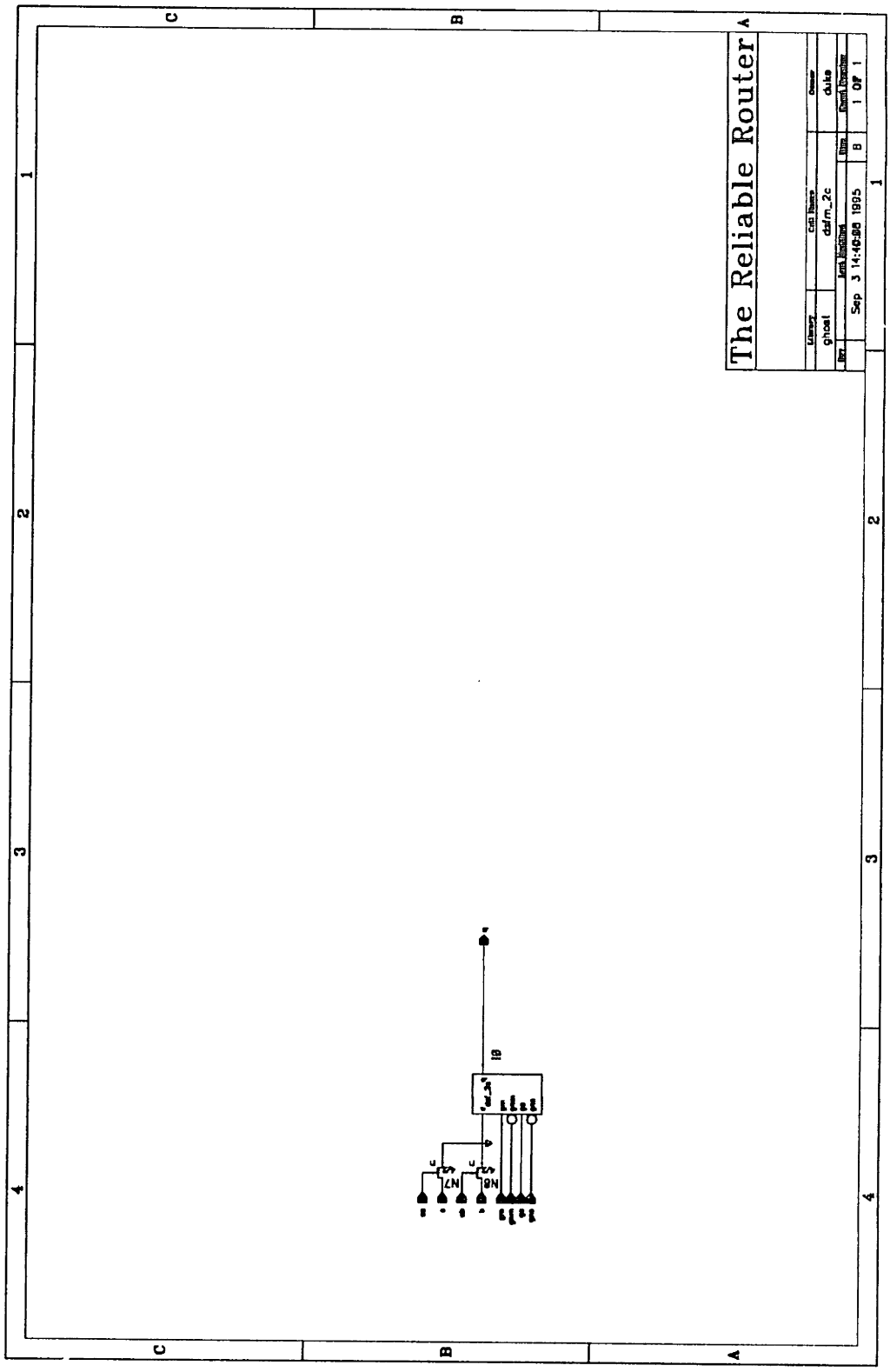


Figure B-27: Library ghost, cell dsfm_2c

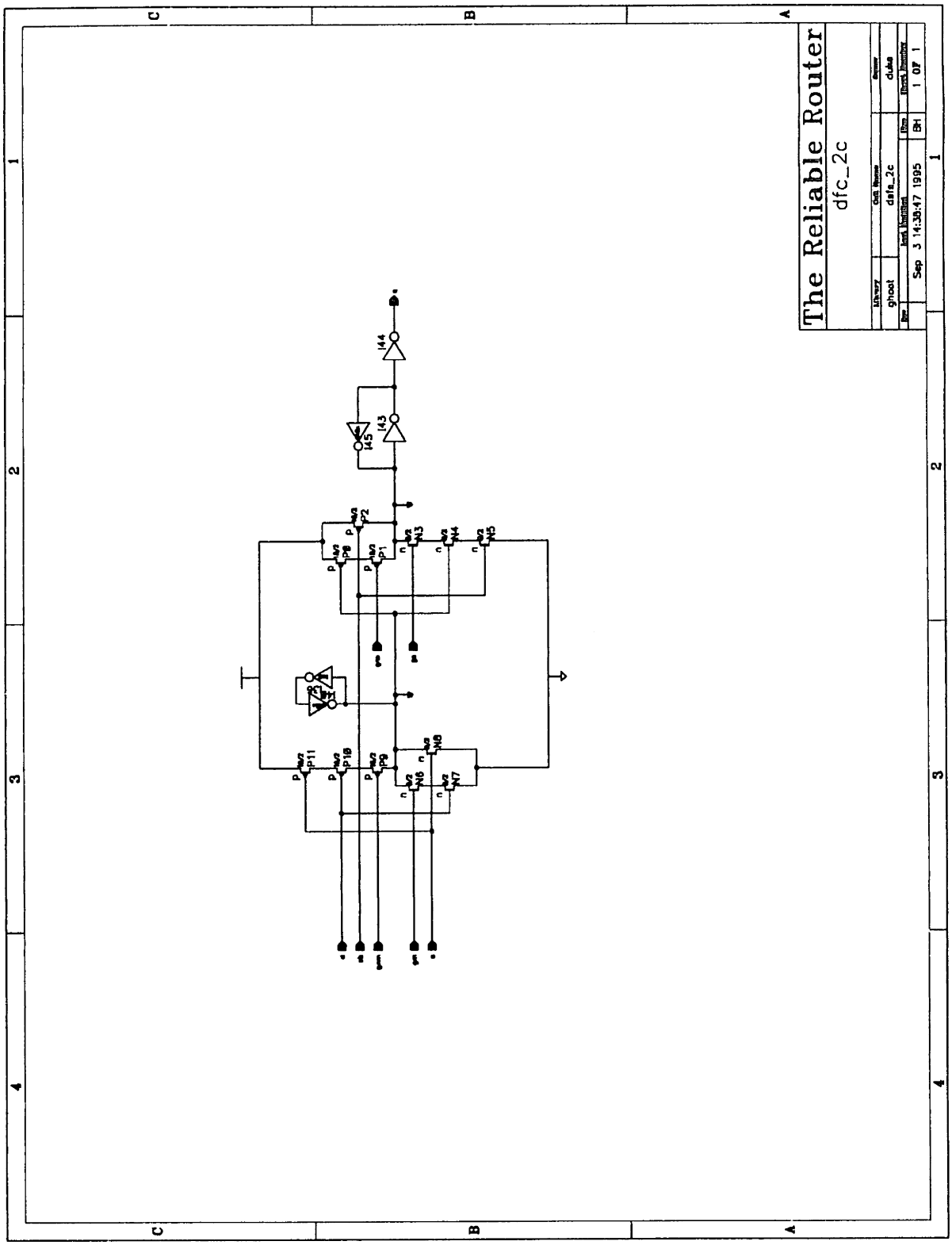


Figure B-28: Library ghost, cell dsfs_2c

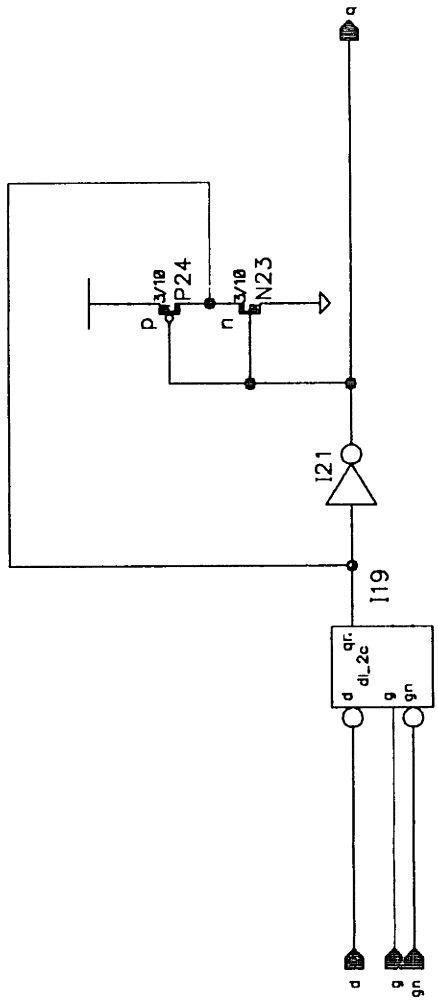


Figure B-29: Library ghost, cell dsl_2c

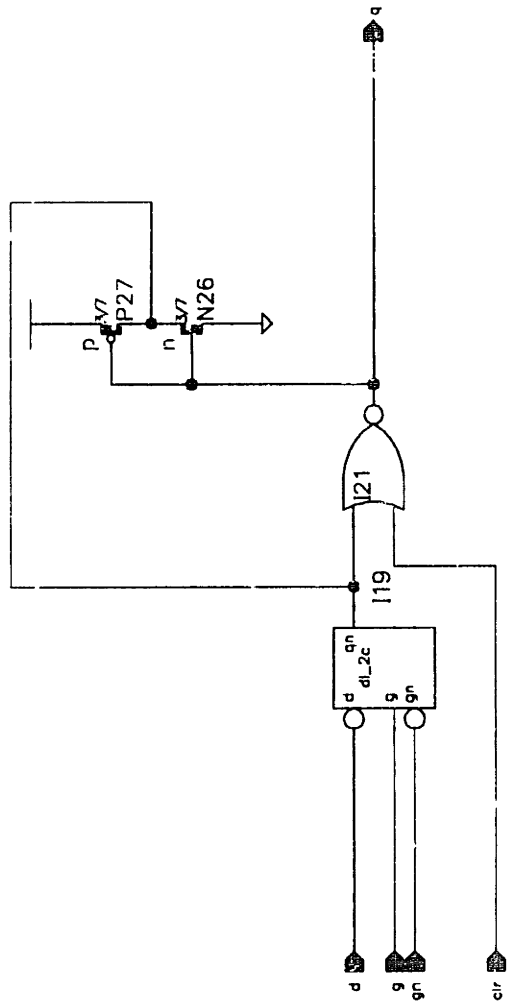


Figure B-30: Library ghost, cell dslc_2c

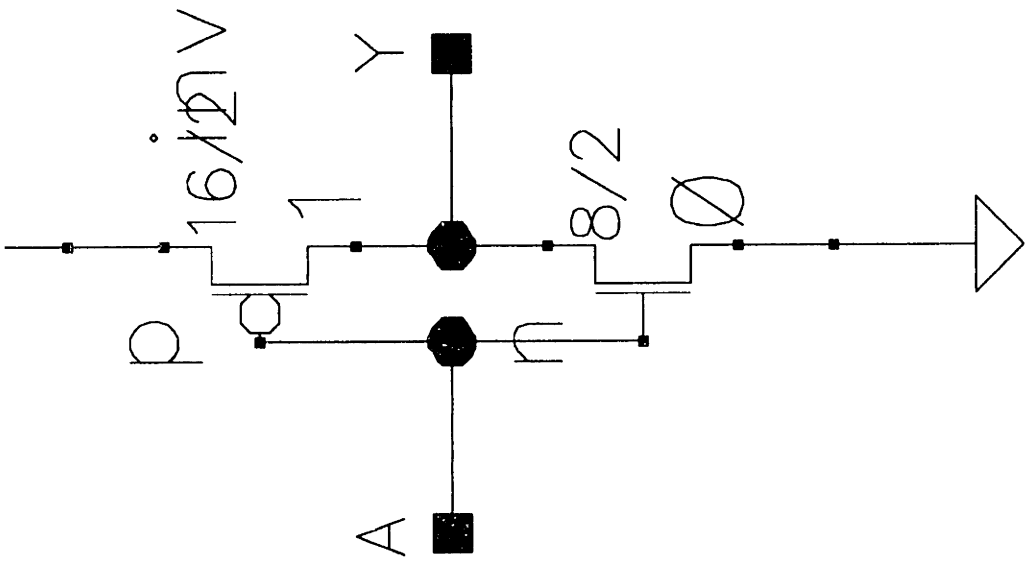


Figure B-31: Library ghost, cell inv

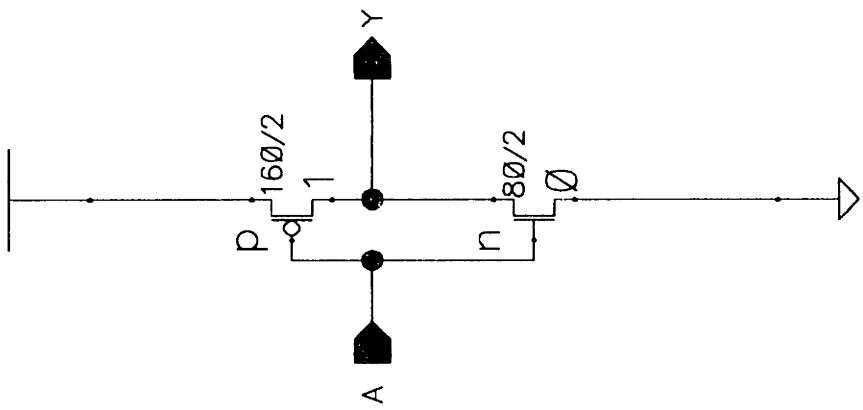


Figure B-32: Library ghost, cell inv10x

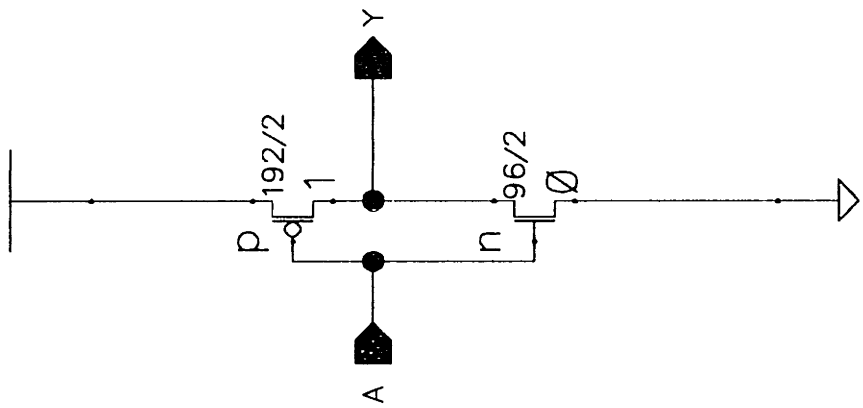


Figure B-33: Library ghost, cell inv12x

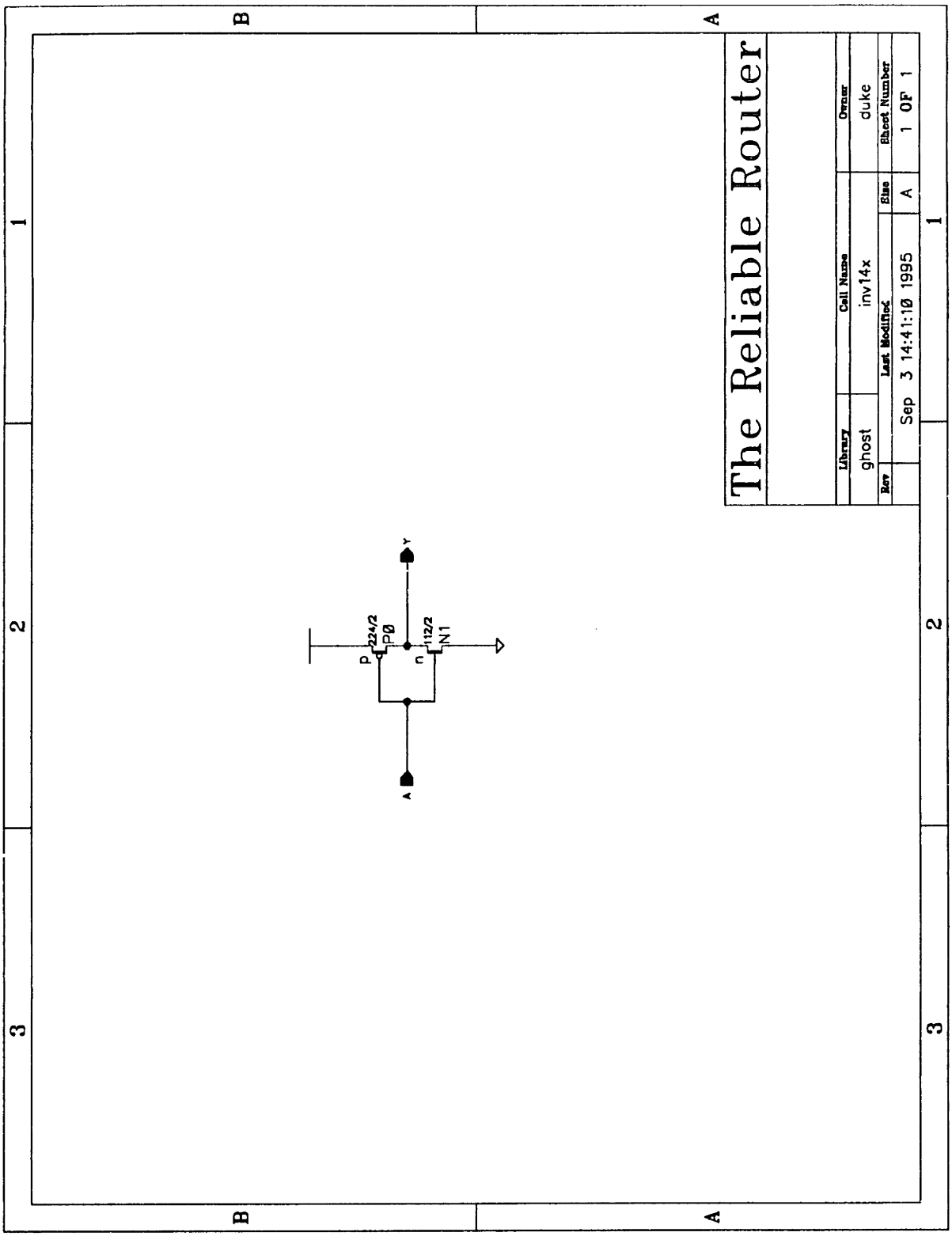


Figure B-34: Library ghost, cell inv14x

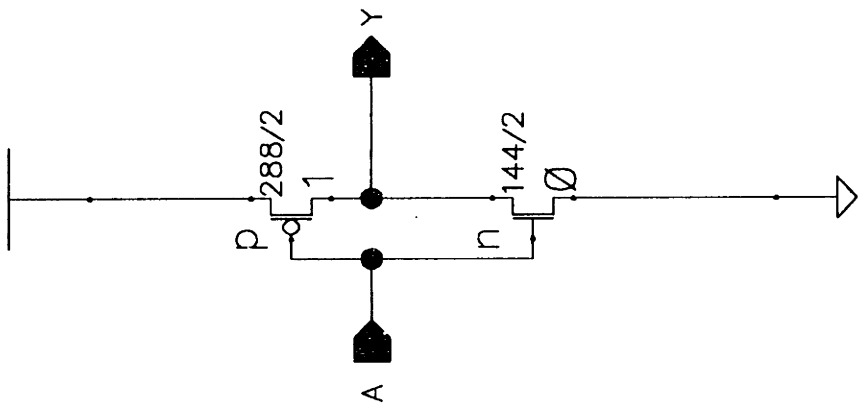


Figure B-35: Library ghost, cell inv18x

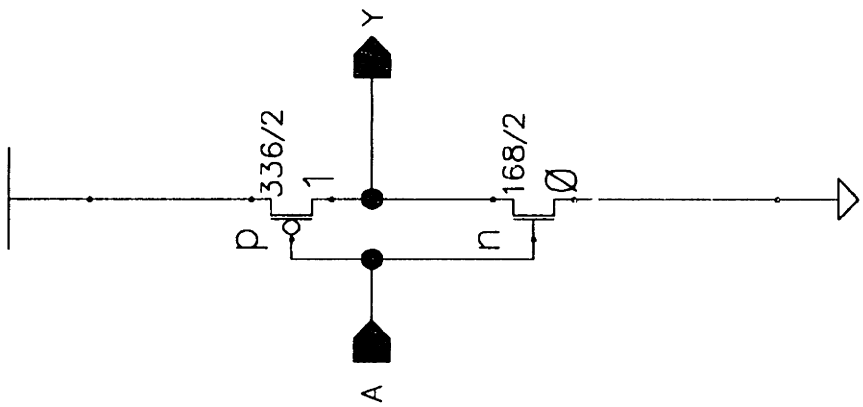


Figure B-36: Library ghost, cell inv21x

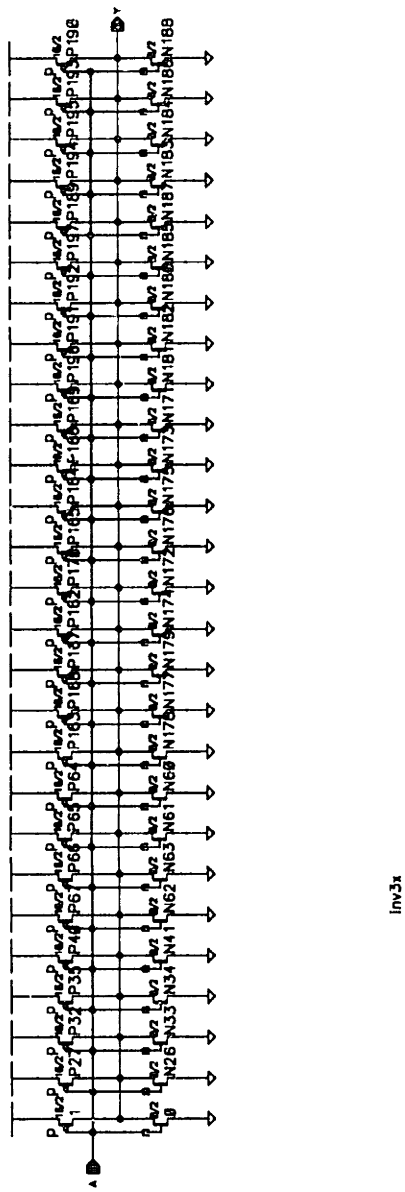


Figure B-37: Library ghost, cell inv27x

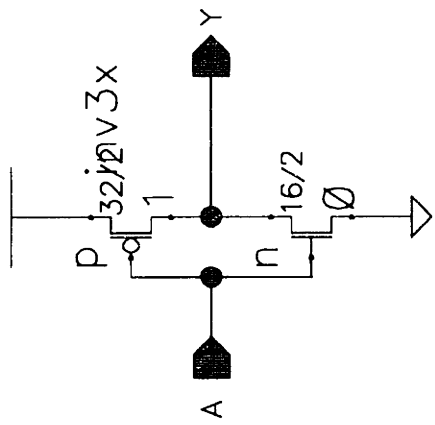


Figure B-38: Library ghost, cell inv2x

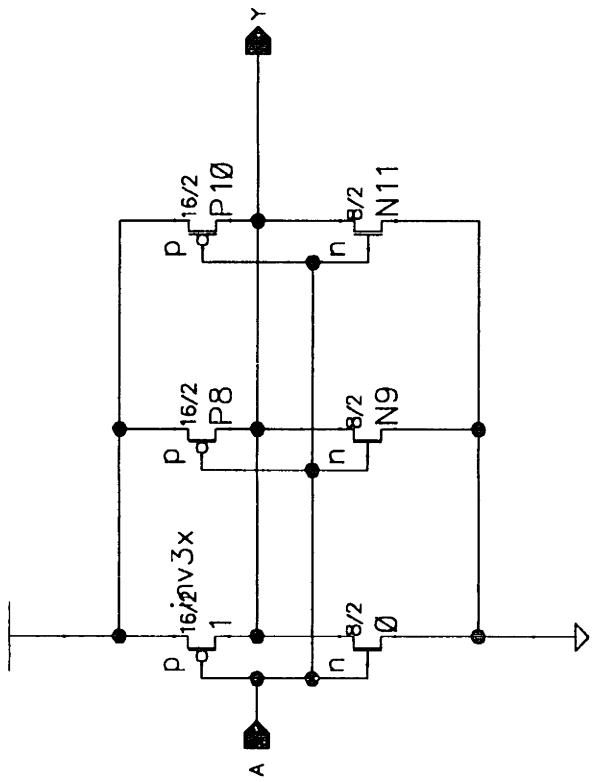


Figure B-39: Library ghost, cell inv3x

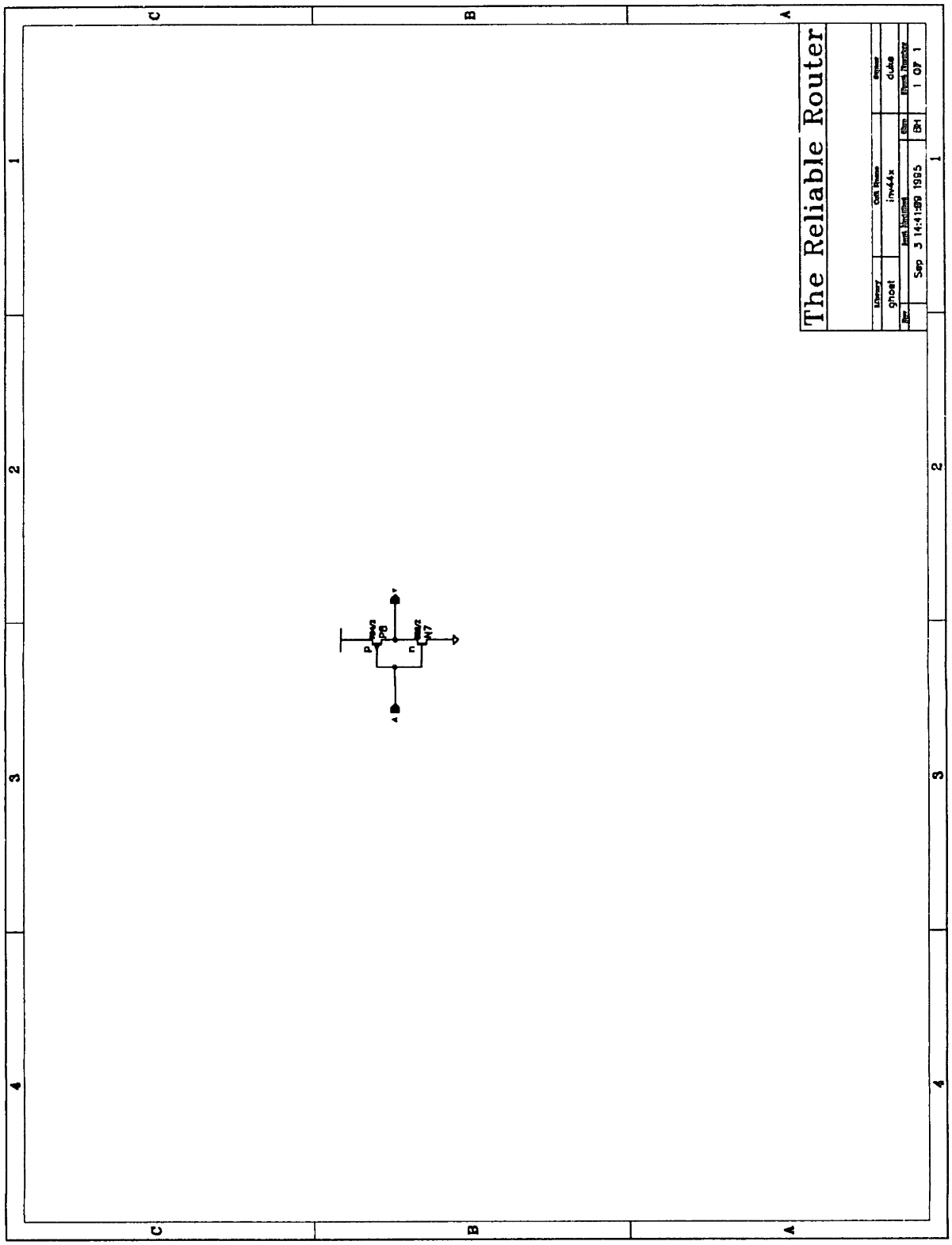


Figure B-40: Library ghost, cell inv44x

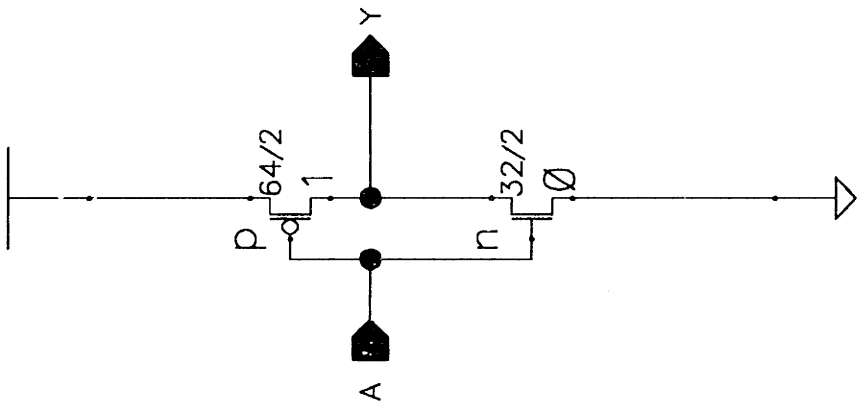


Figure B-41: Library ghost, cell inv4x

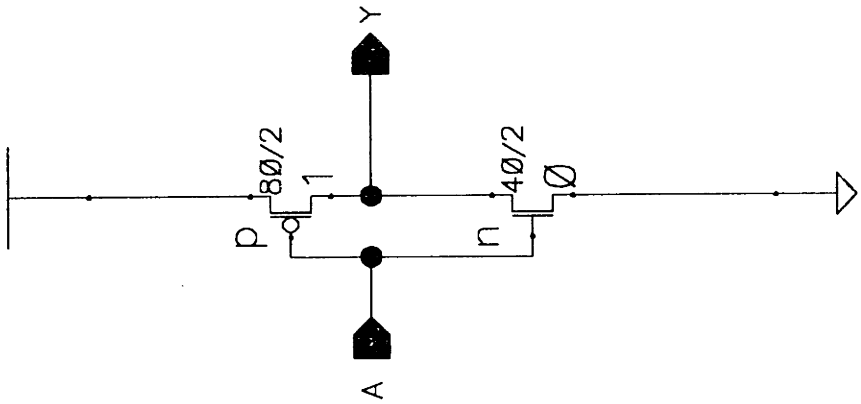


Figure B-42: Library ghost, cell inv5x

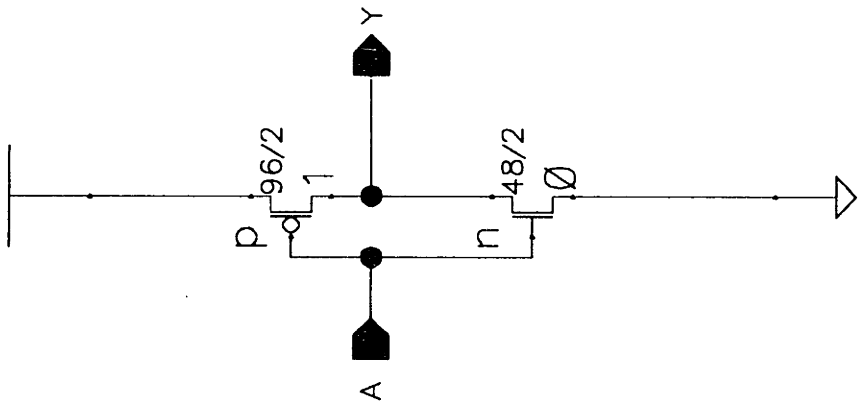
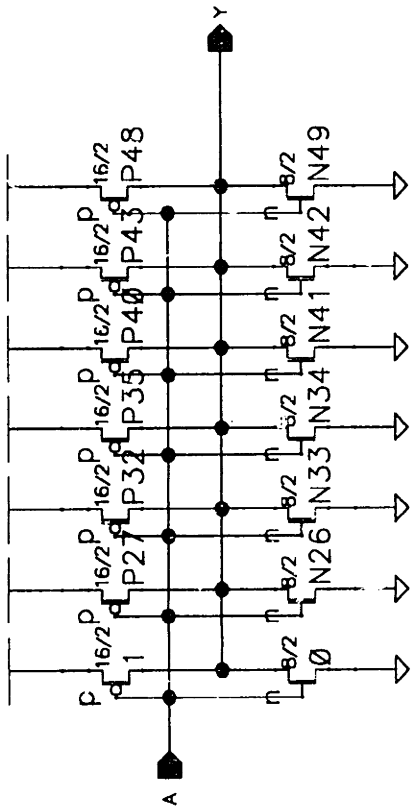


Figure B-43: Library ghost, cell inv6x



inv3x

Figure B-44: Library ghost, cell inv7x

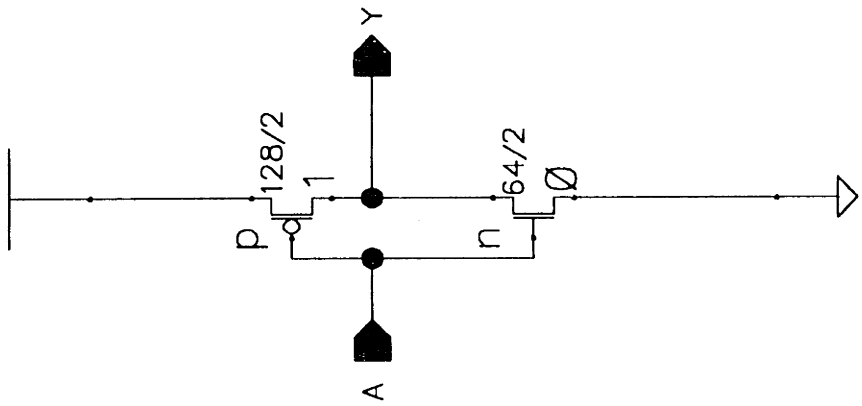


Figure B-45: Library ghost, cell inv8x

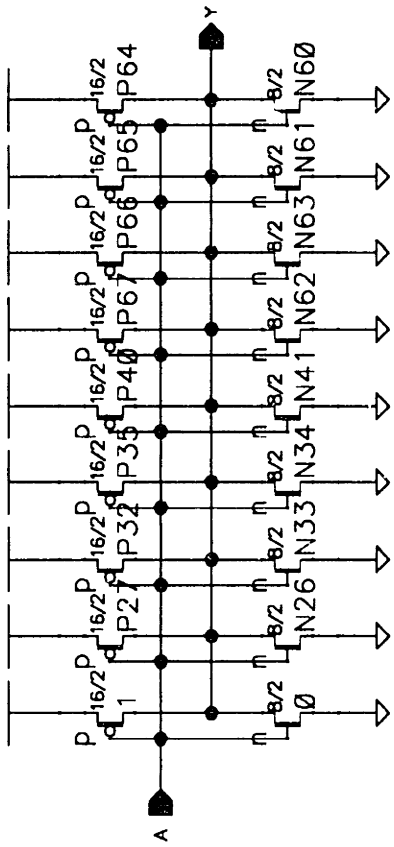


Figure B-46: Library ghost, cell inv9x

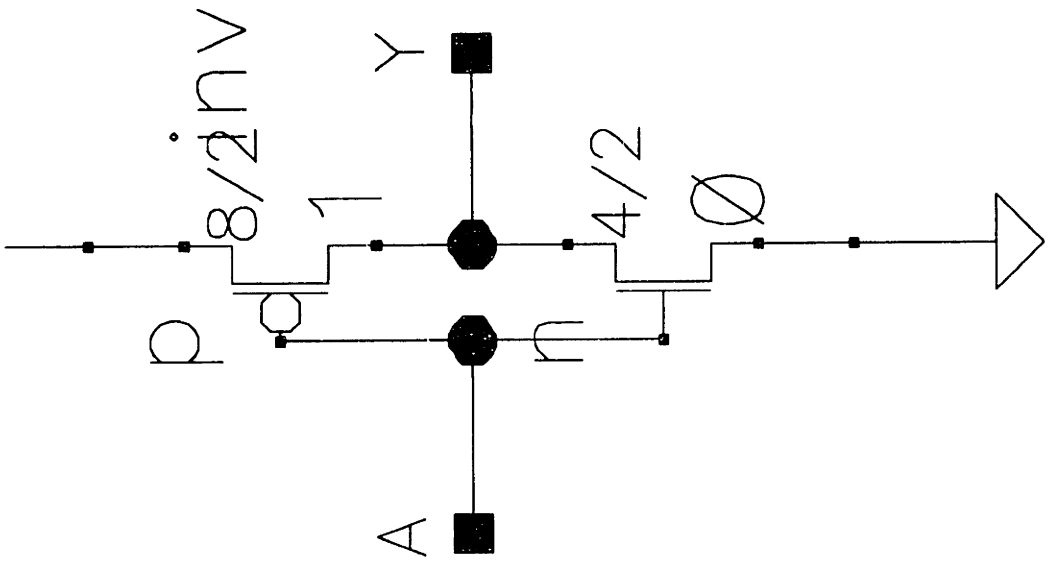


Figure B-47: Library ghost, cell inv_half

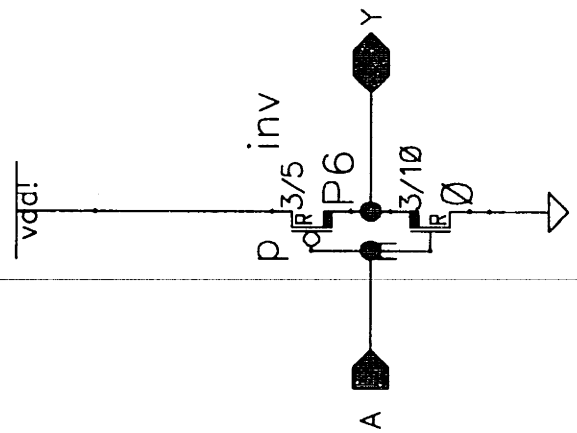


Figure B-48: Library ghost, cell invweak_5x

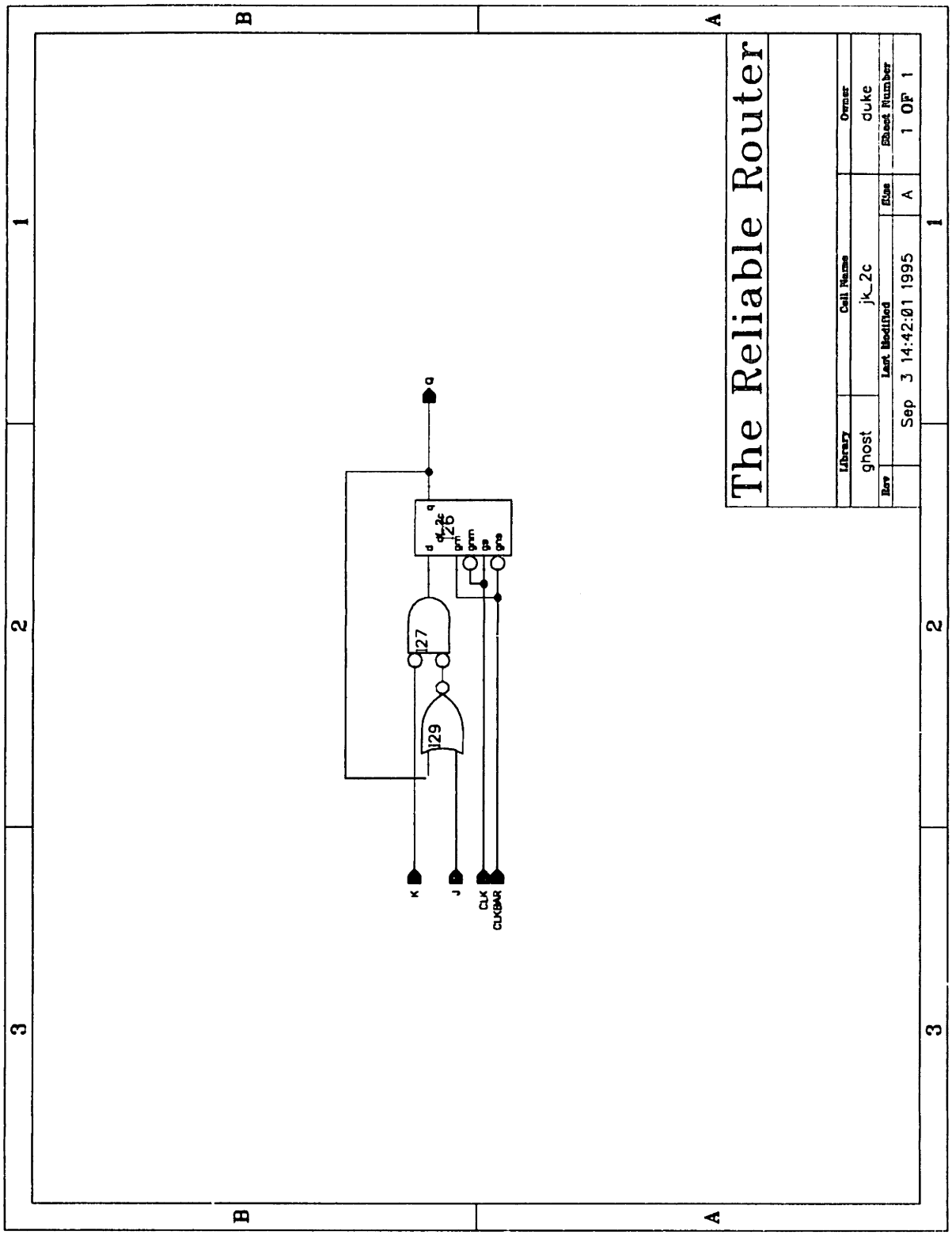


Figure B-49: Library ghost, cell jk_2c

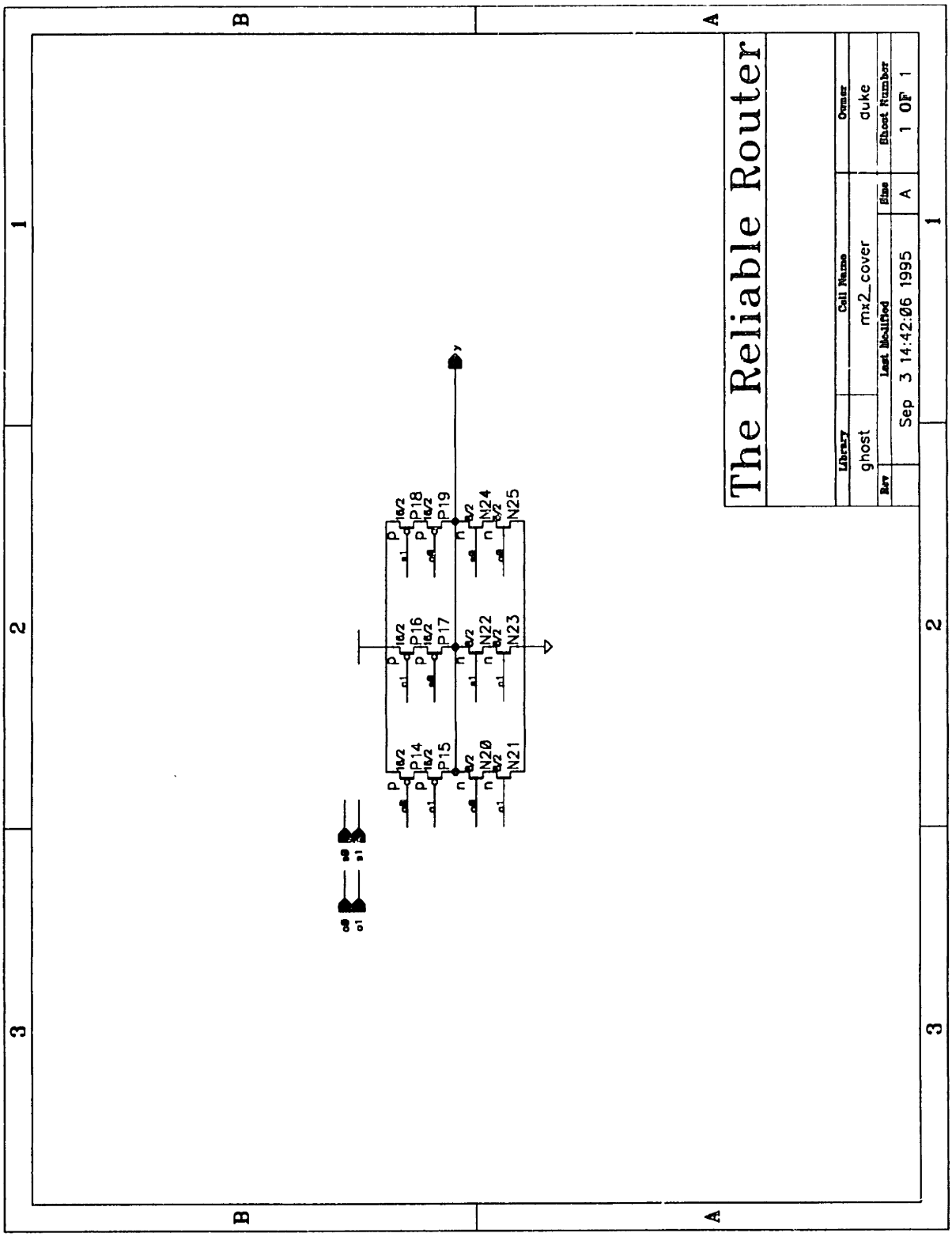


Figure B-50: Library ghost, cell mx2_cover

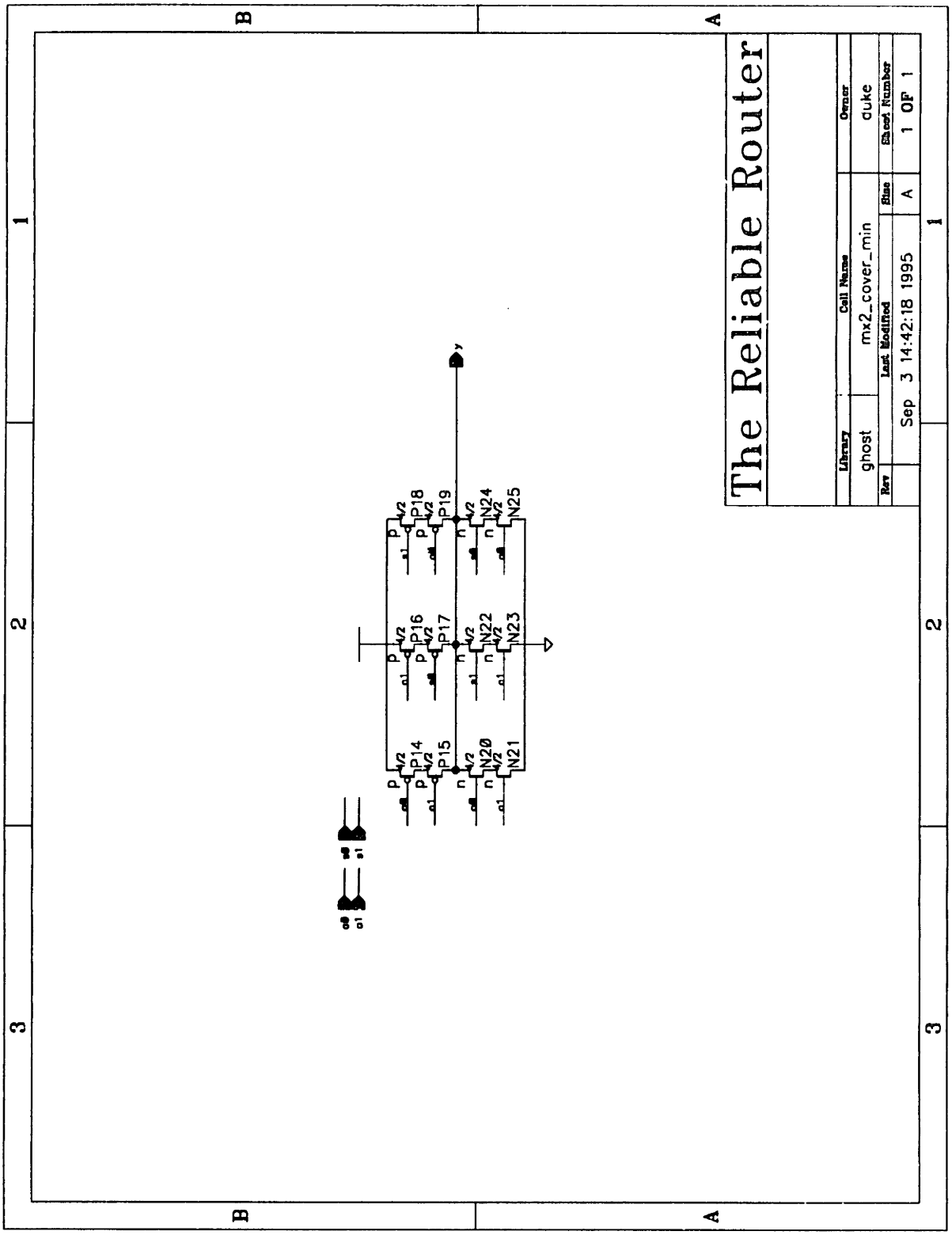


Figure B-51: Library ghost, cell mx2_cover_min

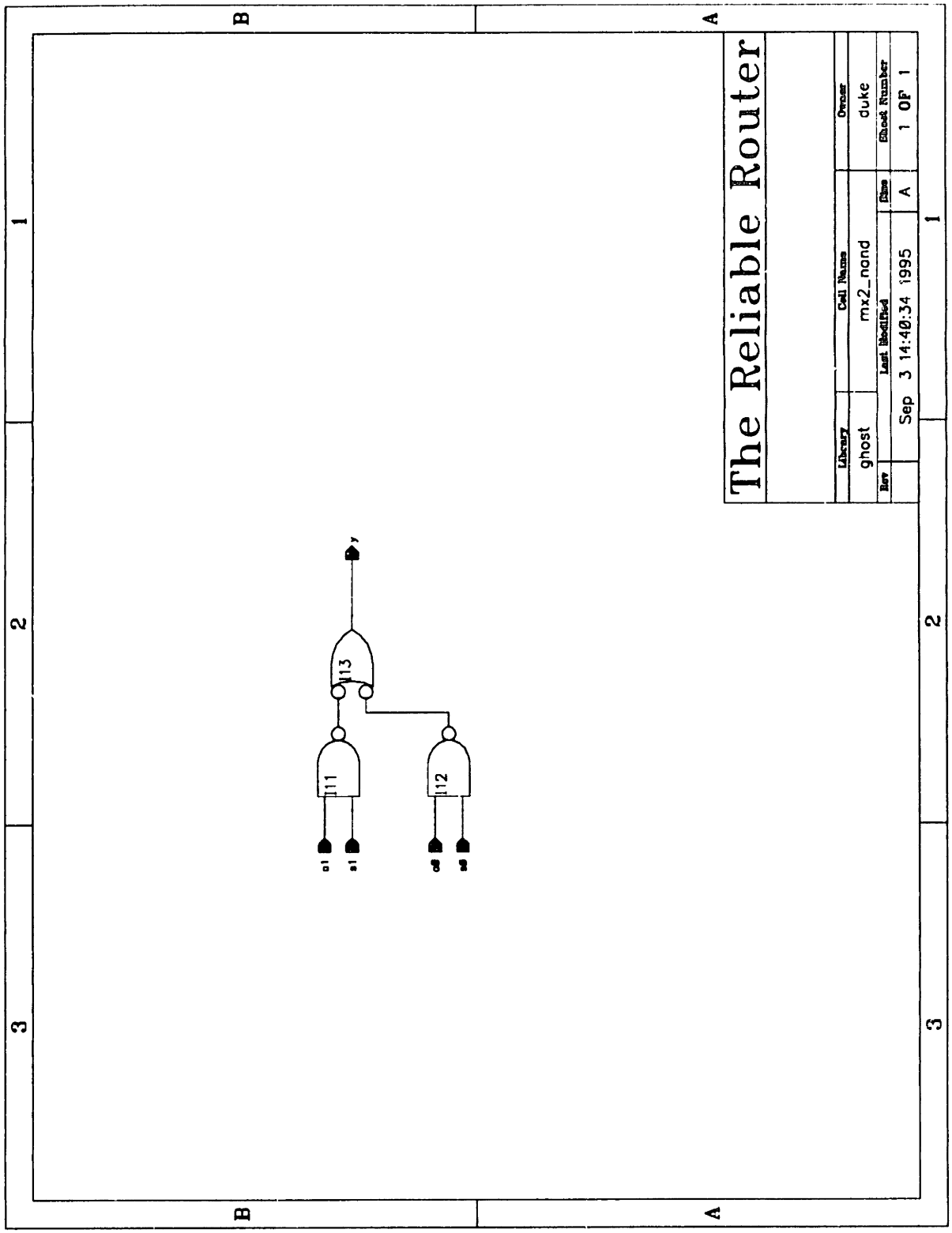


Figure B-52: Library ghost, cell mx2_nand

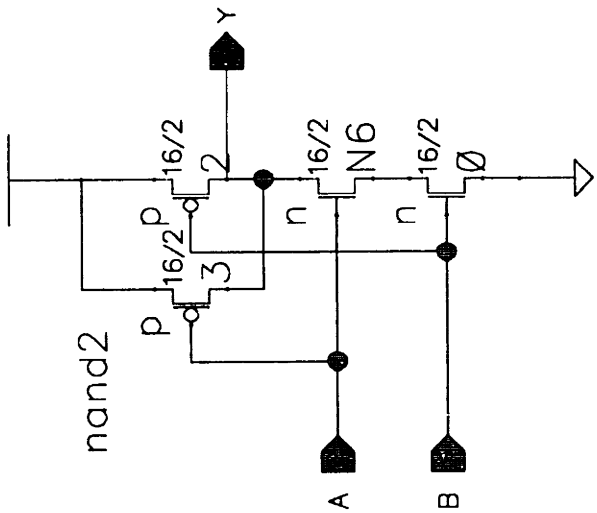


Figure B-53: Library ghost, cell nand2

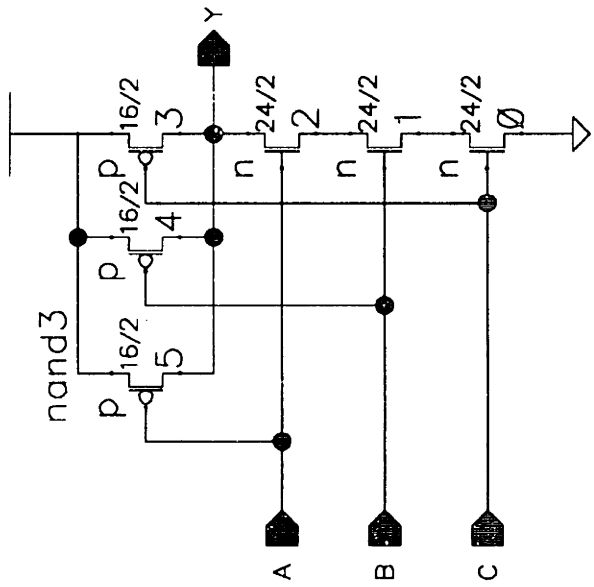


Figure B-54: Library ghost, cell nand3

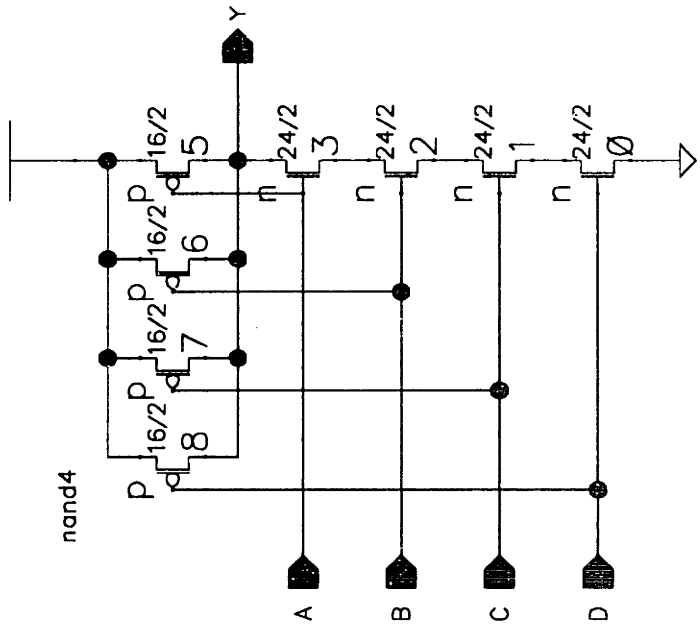


Figure B-55: Library ghost, cell nand4

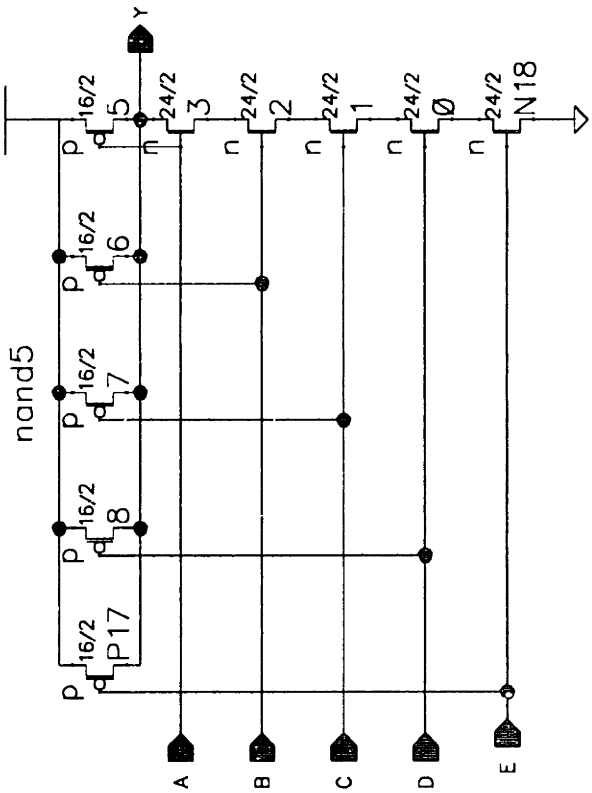


Figure B-56: Library ghost, cell nand5

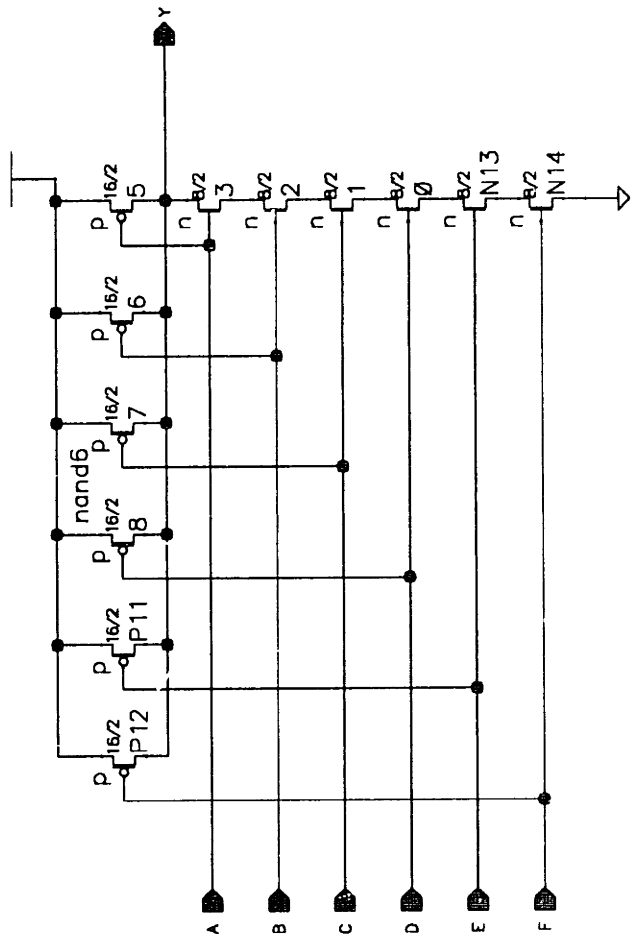


Figure B-57: Library ghost, cell nand6

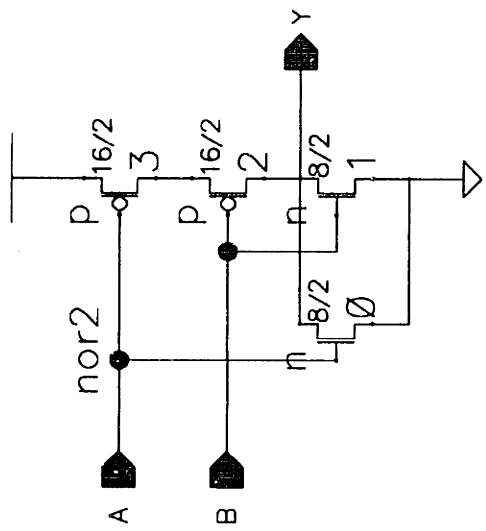


Figure B-58: Library ghost, cell nor2

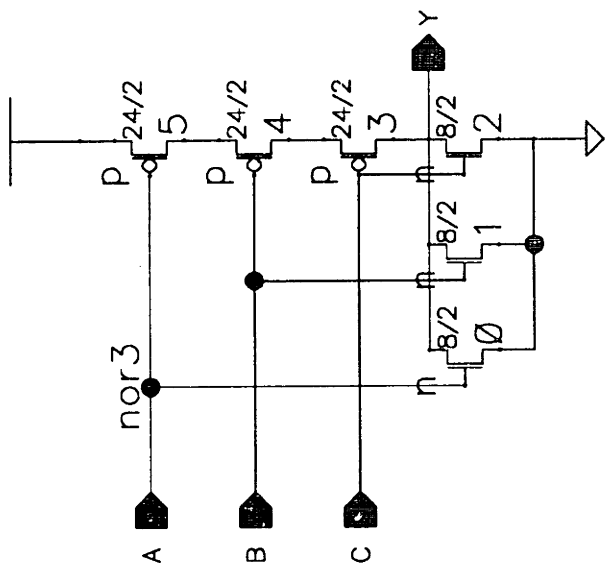


Figure B-59: Library ghost, cell nor3

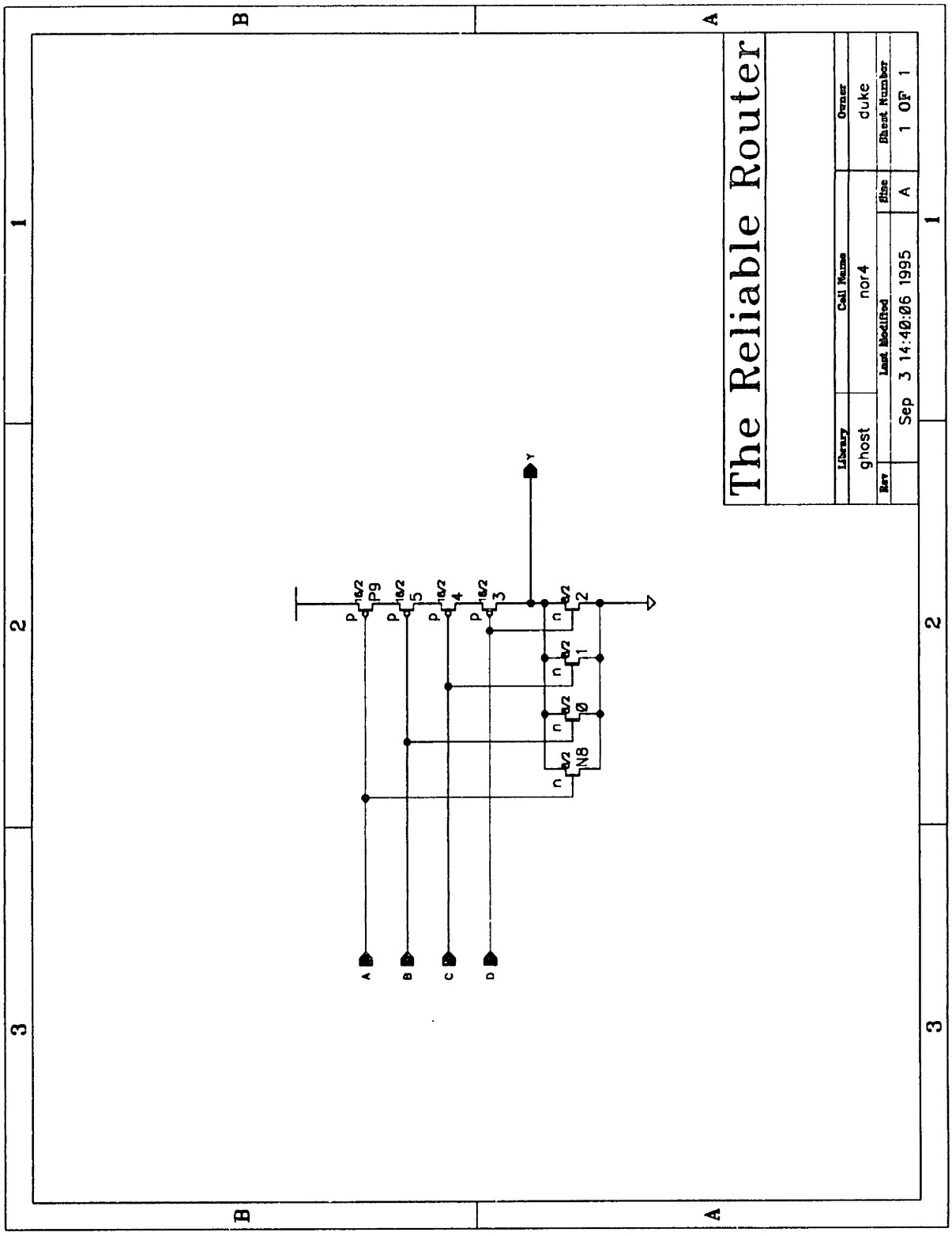


Figure B-60: Library ghost, cell nor4

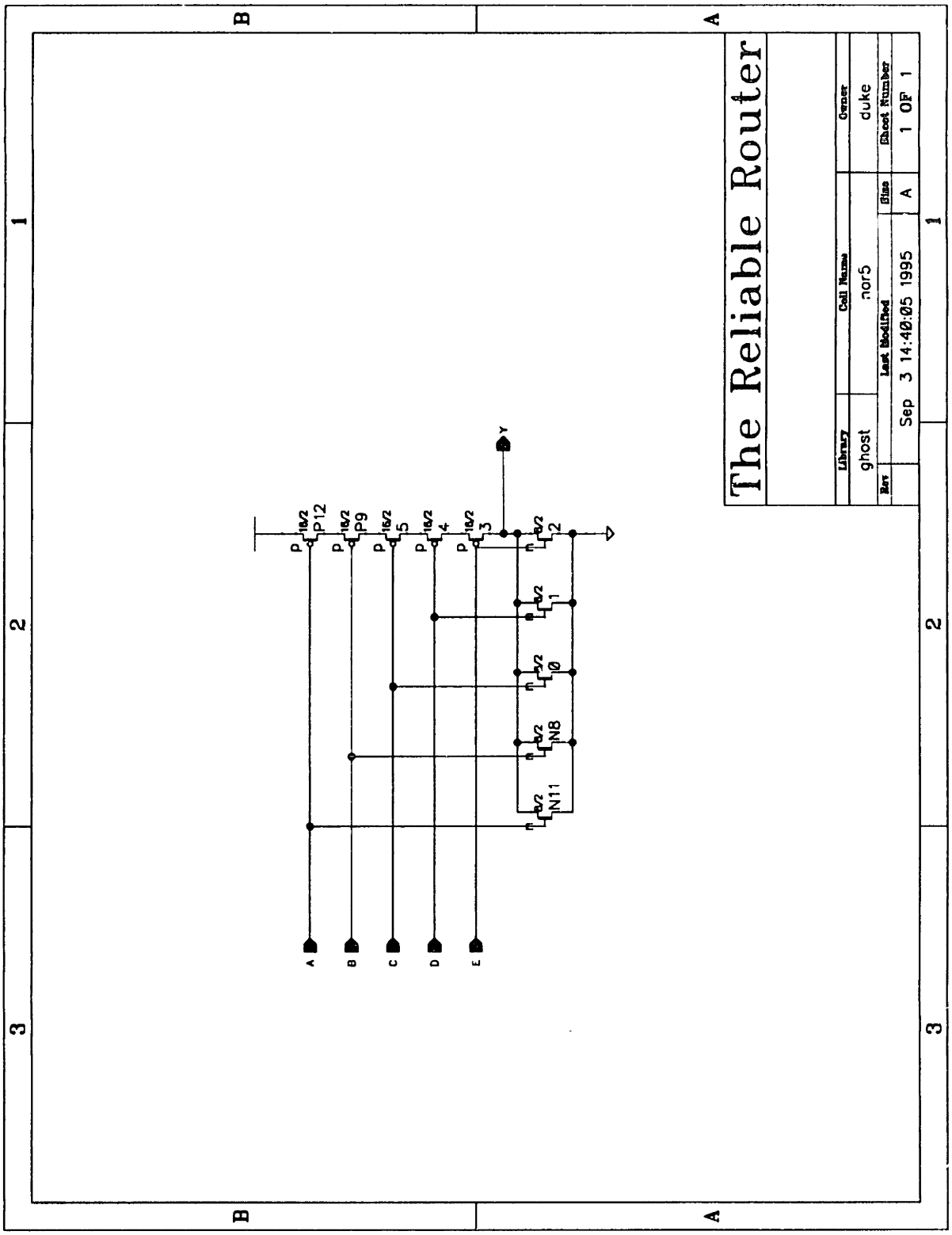


Figure B-61: Library ghost, cell nor5

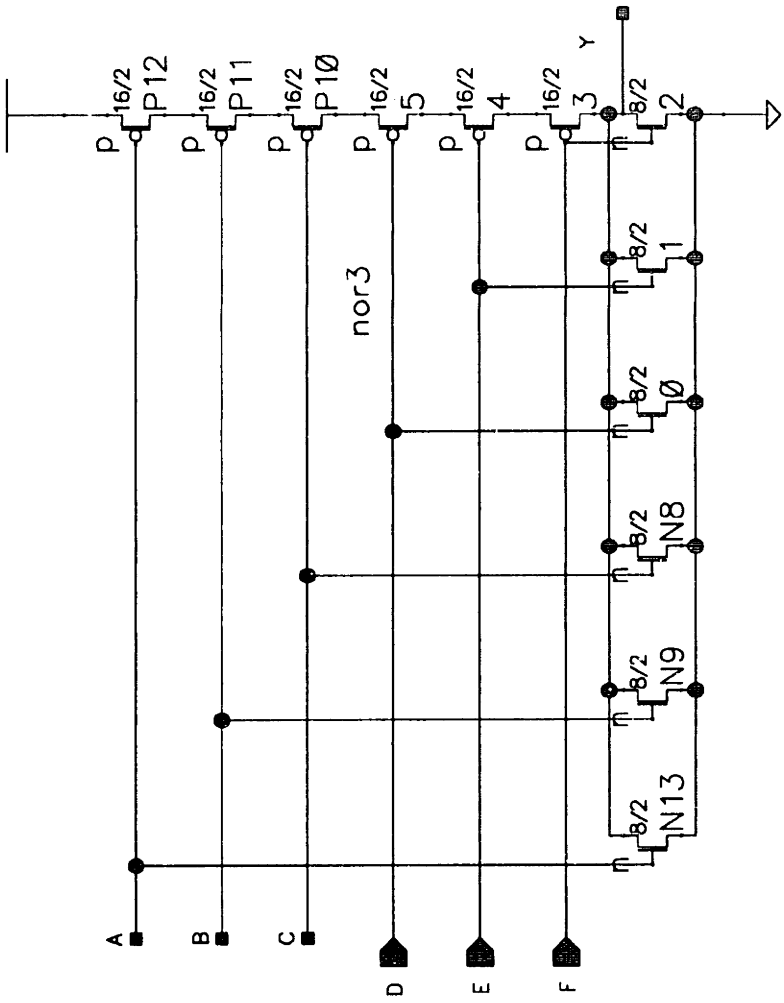


Figure B-62: Library ghost, cell nor6

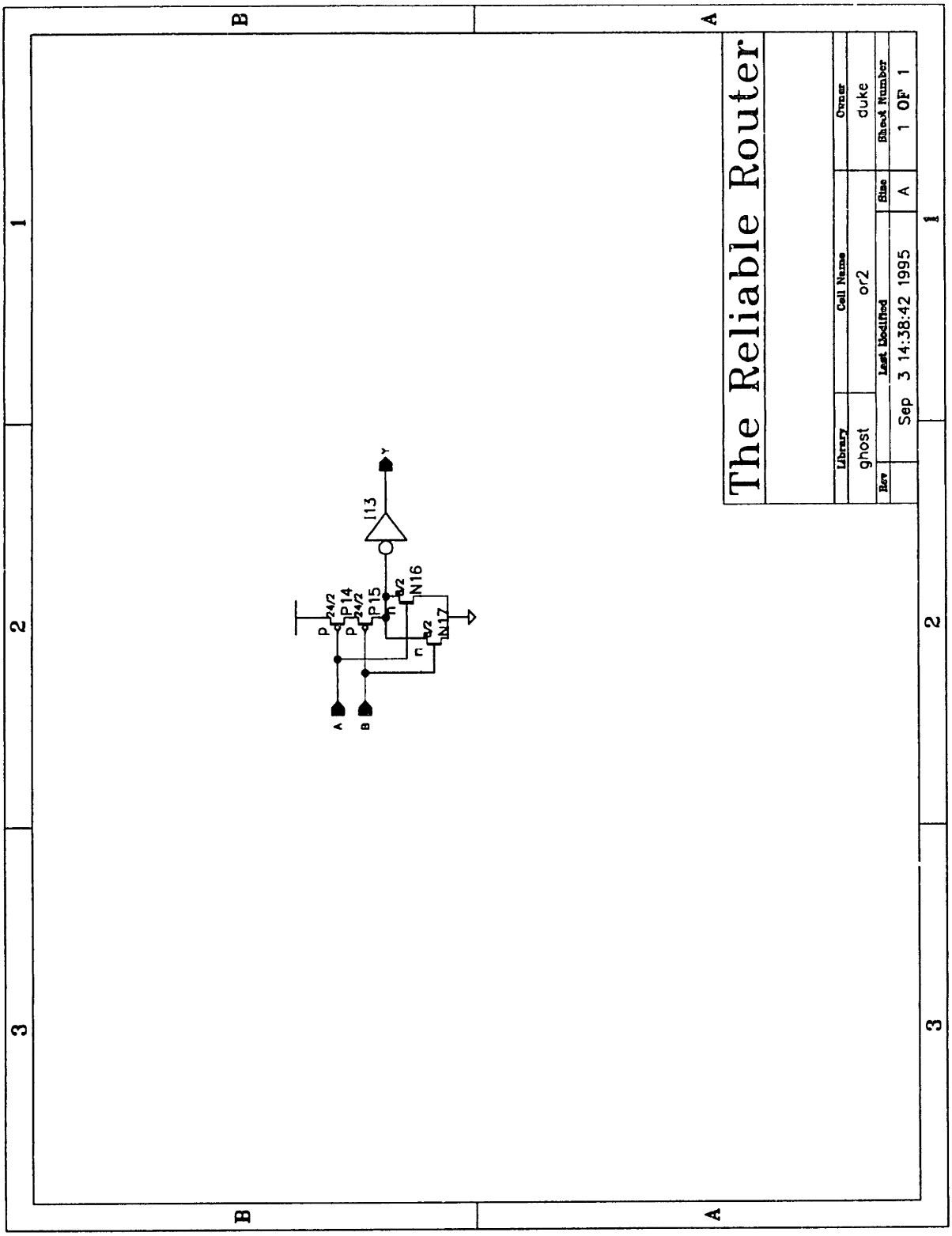


Figure B-63: Library ghost, cell or2

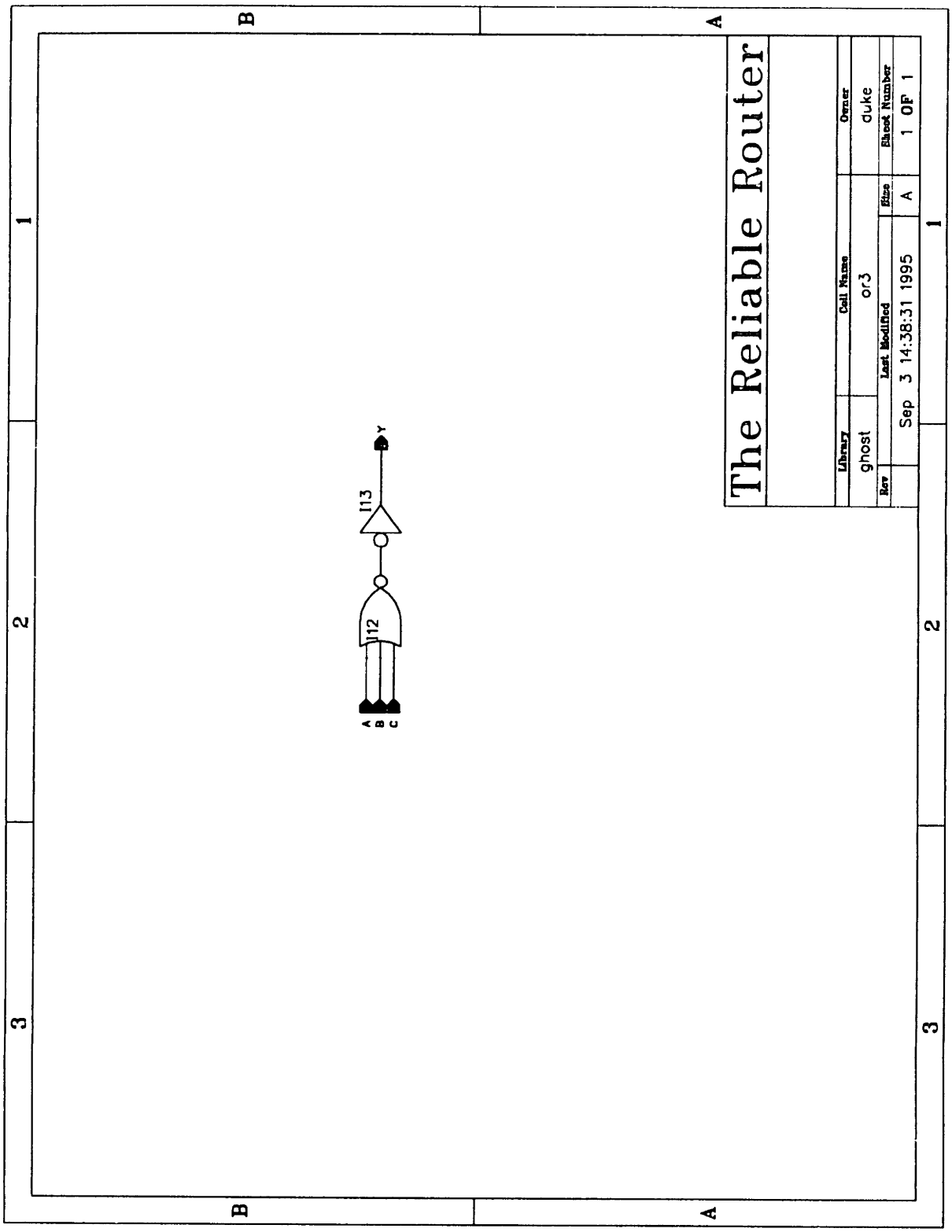


Figure B-64: Library ghost, cell or3

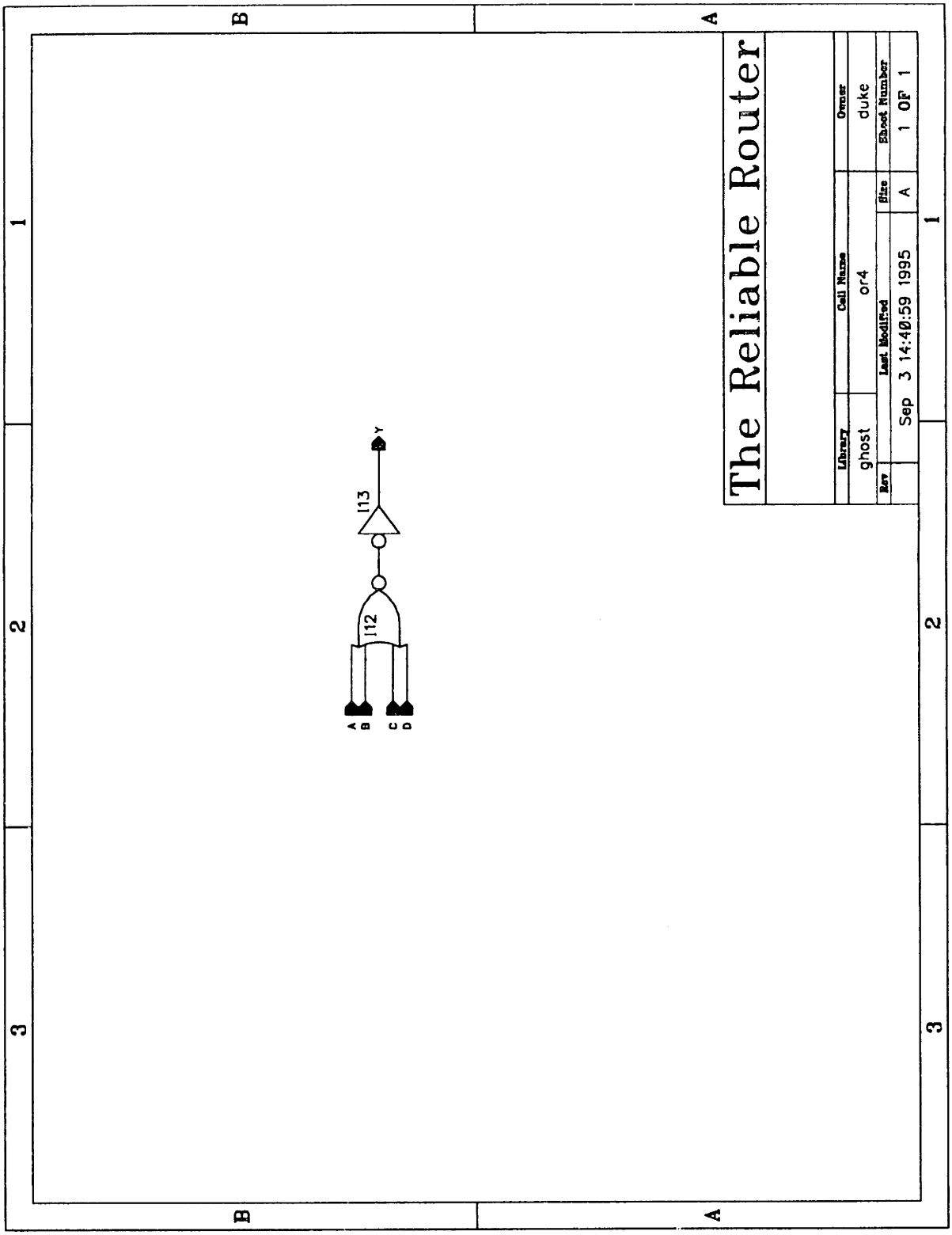


Figure B-65: Library ghost, cell or4

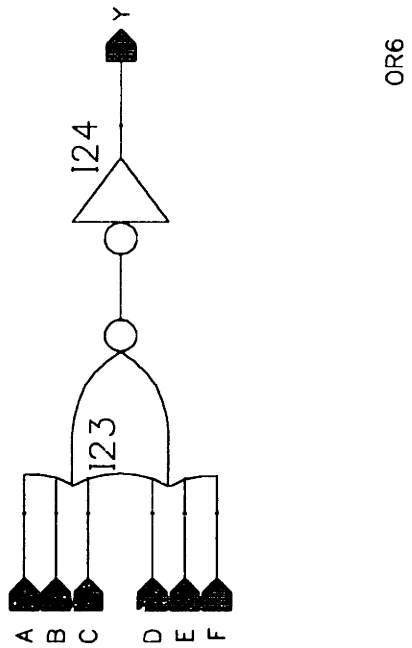


Figure B-66: Library ghost, cell or6

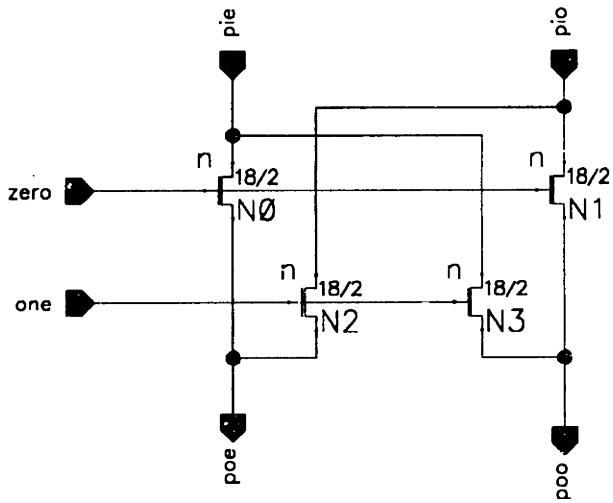


Figure B-67: Library ghost, cell parity_hor

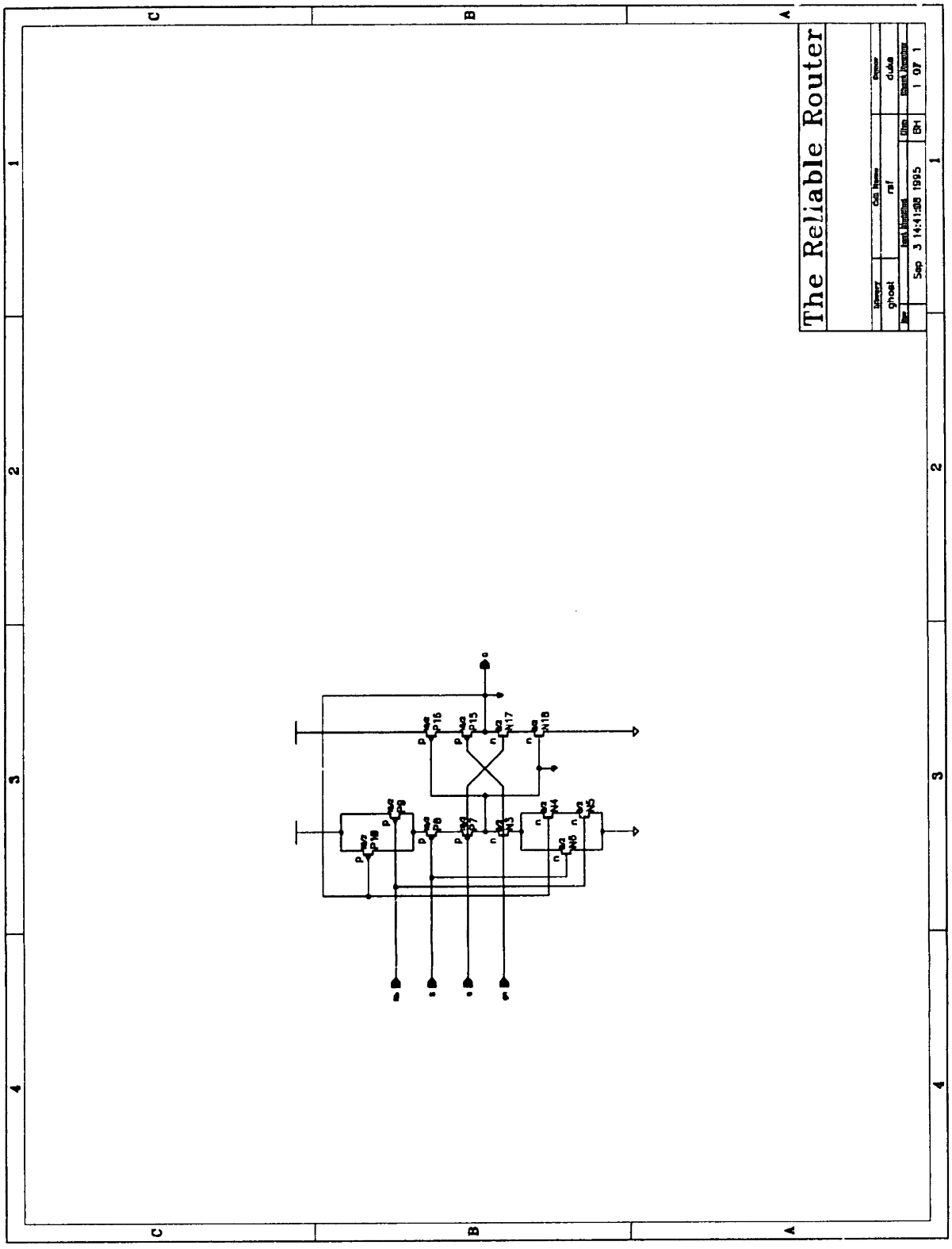


Figure B-68: Library ghost, cell rsf

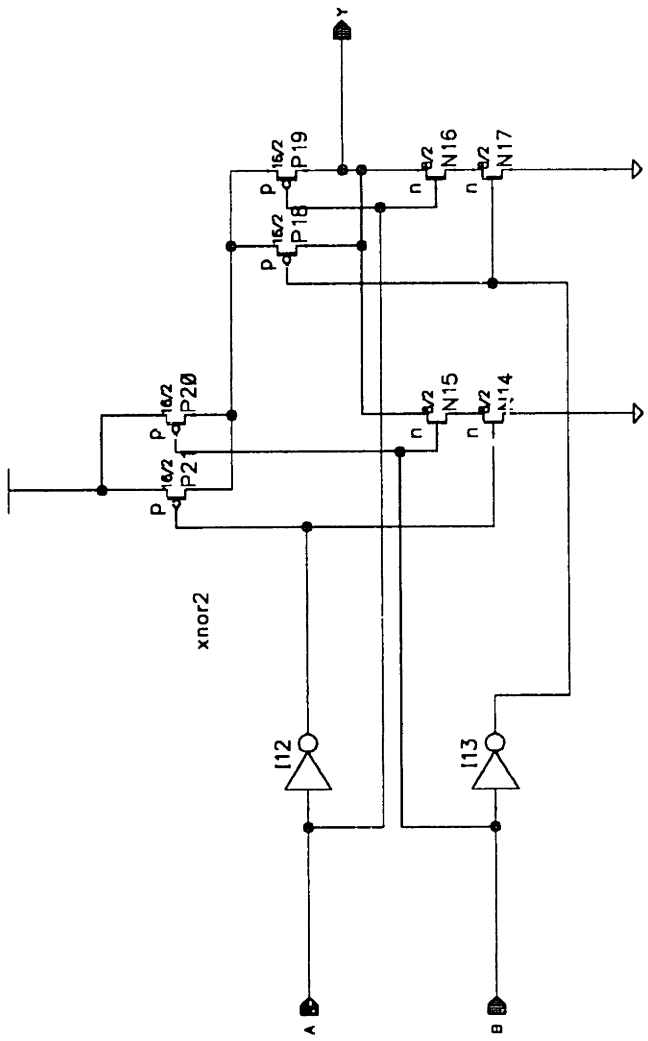


Figure B-69: Library ghost, cell xnor2

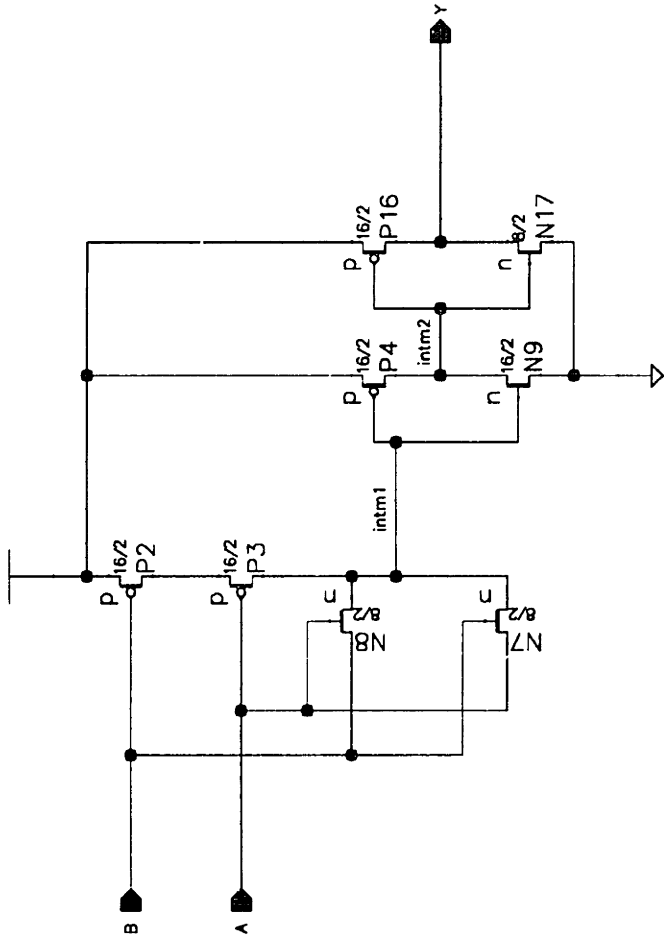


Figure B-70: Library ghost, cell xnor2_new

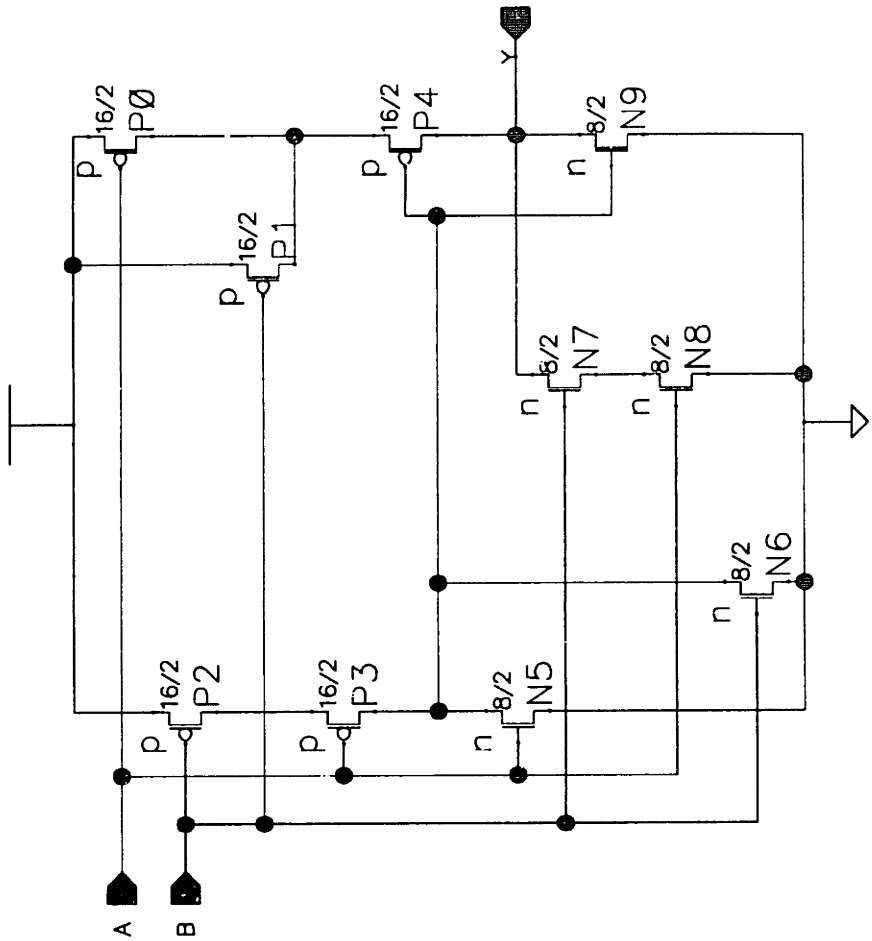


Figure B-71: Library ghost, cell xor2

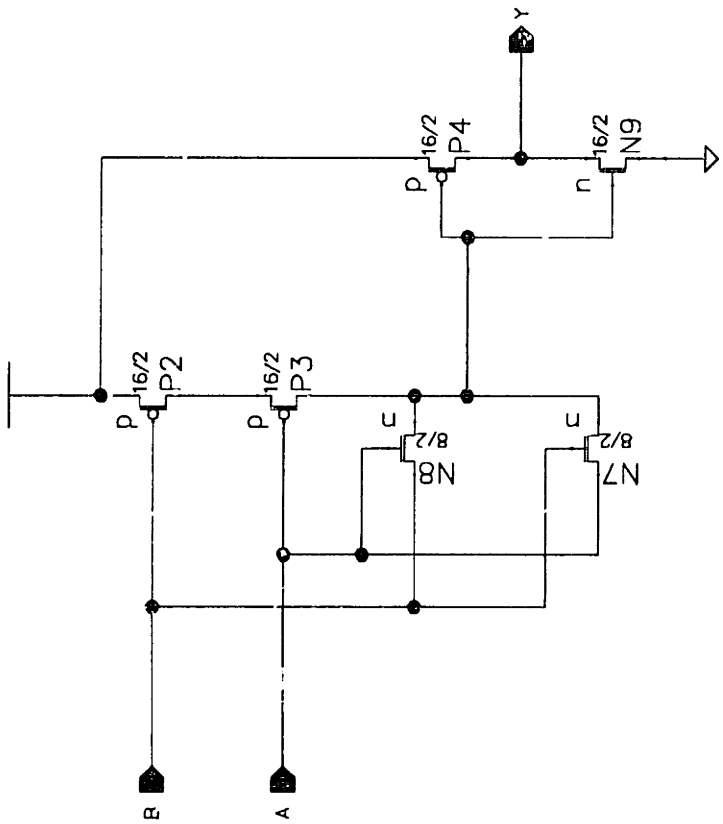


Figure B-72: Library ghost, cell xor2_new