FORTH-ICS / TR-143 November 1995

Design, Implementation, and Testing of a 25 Gb/s Pipelined Memory Switch Buffer in Full-Custom CMOS

Aristides Efthymiou

ABSTRACT: Among the switch buffer architectures, shared buffering achieves the best memory utilization and optimum link throughput, but requires the use of high-throughput memories. Pipelined memory is a novel organization for building such high-throughput memories, featuring simplified control and very efficient VLSI implementation. In this work we designed a pipelined memory with a throughput of 25 Gbits/s (16 Gbits/s in the worst case), enough for 8 incoming and 8 outgoing links at gigabit per second rates. Full-custom design techniques were employed to achieve small area and high speed. The full-custom implementation of the pipelined memory pays off the long development time, as our implementation is 4 times smaller and 3 times faster than the pipelined memory of a semi-custom switch chip that was recently designed by our research group. Besides the pipelined memory that was implemented, three other floorplan organizations were investigated which can be used for pipelined memories of different specifications. A test chip containing the pipelined memory and semi-custom control and interface circuits was fabricated. This chip, although it revealed some minor design errors, operates correctly for clock frequencies of up to 80MHz, consuming 2.75 Watts.

Design, Implementation, and Testing of a 25 Gb/s Pipelined Memory Switch Buffer in Full-Custom CMOS

Aristides Efthymiou

Institute of Computer Science (ICS)
Foundation of Research and Technology — Hellas (FORTH)
Science and Technology Park, Heraklio, Crete
POBox 1385, GR-711-10 Greece
E-mail: efthym@ics.forth.gr, Tel: +30 (81) 391660, Fax: +30 (81) 391661

Technical Report FORTH-ICS / TR-143 — November 1995

Work supported by ESPRIT P.6253 “SHIPS” and ACTS P.060 “ASSICOM”

ABSTRACT: Among the switch buffer architectures, shared buffering achieves the best memory utilization and optimum link throughput, but requires the use of high-throughput memories. Pipelined memory is a novel organization for building such high-throughput memories, featuring simplified control and very efficient VLSI implementation. In this work we designed a pipelined memory with a throughput of 25 Gbits/s (16 Gbits/s in the worst case), enough for 8 incoming and 8 outgoing links at gigabit per second rates. Full-custom design techniques were employed to achieve small area and high speed. The full-custom implementation of the pipelined memory pays off the long development time, as our implementation is 4 times smaller and 3 times faster than the pipelined memory of a semi-custom switch chip that was recently designed by our research group. Besides the pipelined memory that was implemented, three other floorplan organizations were investigated which can be used for pipelined memories of different specifications. A test chip containing the pipelined memory and semi-custom control and interface circuits was fabricated. This chip, although it revealed some minor design errors, operates correctly for clock frequencies of up to 80MHz, consuming 2.75 Watts.

Contents

Chapter 1

Introduction

Throughput, scalability, and low latency are the most important features of present and future computer and communication networks. These are achieved by employing point-to-point links and switches (routers), as the building blocks of the networks. A switch is a device connected to a number of incoming and outgoing links whose function is to route incoming packets to the appropriate outgoing link(s).

Frequently, two or more incoming packets that arrive simultaneously need to be routed to the same outgoing link. In such occasions, one of the packets is forwarded to the outgoing link, while the others are *buffered* inside the switch and transmitted at a later time.

Many architectural alternatives for organizing these buffers exist, of which the most important are input queueing, output queueing, crosspoint queueing and shared buffering. Shared buffering performs best among the alternatives, but it needs to use high throughput buffers. Such buffers are usually implemented as multiple interleaved banks of memory or as a single memory with a large word size (wide memory). A new, better method for building high throughput memories, called pipelined memory, was recently introduced by Katevenis (see [KaVE95]). This thesis describes the full-custom CMOS implementation of a pipelined memory for an 8×8 switch and its post-fabrication testing. Earlier, two other implementations of the pipelined memory were realized using discrete components and standard cells as the buffer memory of the Telegraphos [Kate94] I and II switches.

1.1 Switch Queueing Architectures

Instead of buffering, some switches use *deflection routing*. That is, the packets that can not be forwarded to the required outgoing link are transmitted through other links that are idle at the time. Depending on the network topology and on the routing algorithm, the deflected packets either use another path to reach their destination, or they return back to the switch that "deflected" them, hoping that they will succeed to pass through this time.

In effect, deflection routing uses the links to delay the packets until the requested port

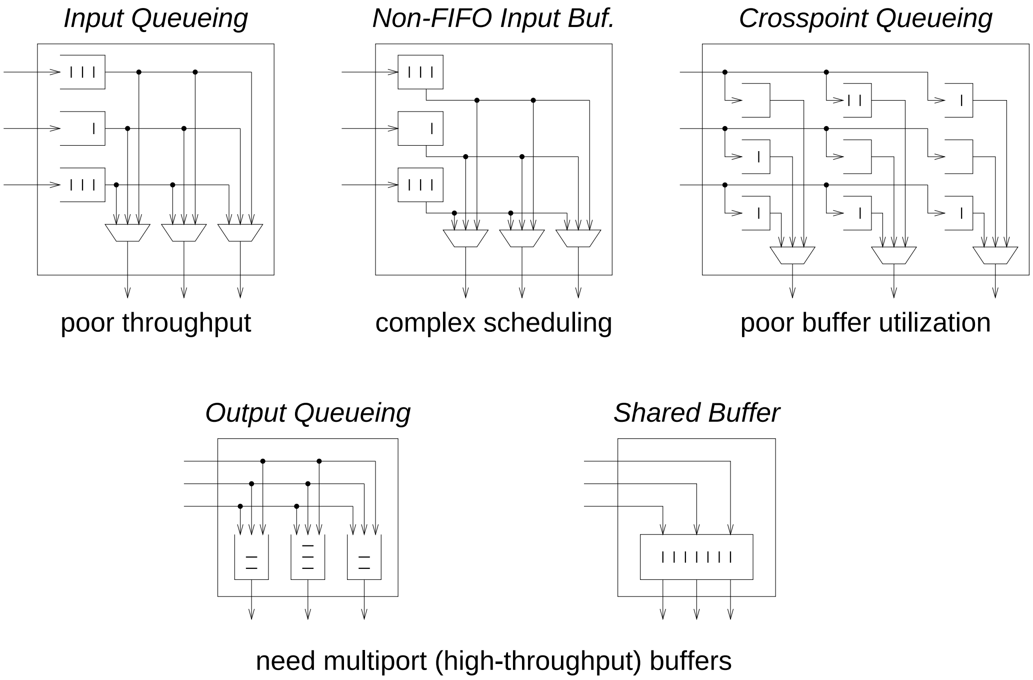


Figure 1.1: Switch queueing architectures.

is free, whereas buffering uses on chip memory for the same purpose. In VLSI technology, inter-chip communication is more expensive than on-chip memory, as it is well known that adding more pins to a chip is costlier than placing more memory in it. In addition, the minimum delay for transmitting and then receiving back a deflected packet is probably longer, and the operation more complicated, than an on-chip memory write and read. For these reasons, this work considers only switches that use buffering.

With regard to buffering, four basic alternatives are available to switch architects: input, output, cross-point, and shared buffering, with FIFO or non-FIFO management (see figure 1.1).

Input FIFO buffering, referred to as *input queueing*, is the simplest architecture; each incoming link feeds one queue dedicated to it. Unfortunately, it is also the worst performing one, due to *head-of-line blocking*: if the first packet in the queue is blocked as a result of an output conflict, the packets behind it are also blocked, even if the outgoing links they are destined to are currently idle. Input buffering without FIFO management yields better performance than input queueing [AOST93] [TaCh93] [LaSe94], but even then output throughput is usually less than optimal. In addition, these non-FIFO input buffers and their scheduling are more complicated than those of input queueing [AOST93] [TaCh93] [LaSe94] [TaFr88].

In *cross-point queueing* there is one FIFO buffer for each pair of input and output ports. In this scheme, each output can select a packet for transmission, among the queues that correspond to it, independent of the other outputs. Thus, the utilization of output throughput is optimal in this architecture. The problem with cross-point queueing is that many buffer memories are needed, yielding more circuit overhead and a larger total memory capacity than that required by other, similar-performing architectures.

The buffer organization where every output port is associated with only one queue is called *output queueing*. In this architecture, each output buffer must have enough throughput to be able to receive packets from all inputs simultaneously, in the worst case. This organization achieves optimal output throughput and better memory utilization than cross-point queueing, but the required buffer throughput is much higher than that of the aforementioned architectures.

Finally, *shared buffering* uses one central buffer for all of the input and output ports. This buffer must have enough throughput to receive packets from all inputs and to provide packets to all outputs, simultaneously. Shared buffering not only achieves optimal output throughput, like cross-point and output queueing, but it also achieves the best memory utilization [HlKa88]. Thus, shared buffering is the best switch buffer architecture for high performance switches. The cost of implementing this buffer organization is not too high, as the existing techniques and the recent pipelined memory architecture are not more expensive to build than the implementations of the other organizations (see [KaVE95], section 5.1).

1.2 This Thesis

In this work we present the design of a pipelined memory for an 8×8 switch, with a packet size of 256 bits, and a total storage of 256 packets. It is designed using full-custom VLSI techniques and a digital CMOS $1\mu m$ technology. Our implementation is simulated to work at 100MHz in the typical conditions, yielding a total buffer throughput of 25Gbits/sec. Four basic alternative organizations were considered for the floorplan of this pipelined memory, and although only one of them is implemented, the others may be preferable for different buffer specifications. The implemented pipelined memory was included in a test chip which also contains interface and control circuits that were designed with standard cells. This chip tests all the operations of the pipelined memory at full speed, without requiring expensive external circuitry.

In chapter 2 we summarize the existing techniques for implementing high throughput memories and describe the new pipelined memory organization. Chapter 3 covers the design, floorplan organization, and other implementation issues of a pipelined memory for an 8×8 switch, and chapter 4 describes the test chip, where this pipelined memory is included. In chapter 5 we present the results of post-fabrication testing, and we conclude in chapter 6, where we also give ideas for extending this work.

Chapter 2

High Throughput (Multiport) Memories

True multiport memory is overly expensive in silicon area, because of the extra memory cell transistors, bit-lines and word-lines that have to be added. Thus, true multiport memories are only used for building small memory blocks like processor register files, which is not the case for switch buffers. Therefore, switch architects use pseudo-multiport memories: interleaved memory and wide memory.

This chapter describes the existing multiport memory organizations and the pipelined memory organization proposed in [KaVE95].

2.1 Pseudo-Multiport Memories

We call *interleaved memory* the organization of a memory into multiple interleaved single ported banks. Interleaved means that the address space is dispersed among the banks, with the most usual case being that consecutive addresses correspond to different memory banks. In this way, the number of simultaneous memory accesses is equal to the number of the banks, if these memory accesses request data residing in different banks. Many switches use interleaved memory, either as a shared buffer, or for building high throughput output queues: “Knockout” [YeHa87], “Prelude” [DeCS88] (for reads only), [Turn93], [Mirf95], and “PRIZMA” [DeEI95].

Instead of achieving high-throughput using multiport memory built from several single-ported memories, a single memory with a large word size can be used (*wide memory*). One can view this as an interleaved memory in which the same address register is used for all of the banks. This works best when the size of the packet is a multiple of the wide memory word size, so that all accesses are aligned on wide-word boundaries. Wide memory is very attractive for single chip switches, because, in VLSI technology, memories are naturally built wide. The wide memory organization is quite popular: the IBM “Vulcan” [Stun94], the “Prelude” [DeCS88] (when storing packets), and the VLSI ATM switches [Koza91] and [Shob91], all use this organization.

A shared buffer ATM switch employing wide memory [KaSC91], was designed by our

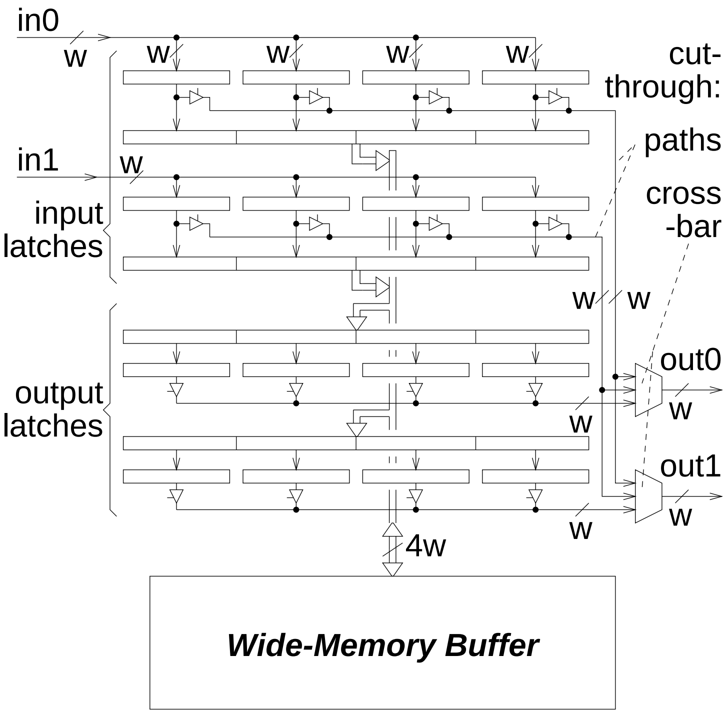


Figure 2.1: The wide memory buffer of [KaSC91].

research group. Figure 2.1 shows its buffer, the incoming and outgoing links, and their associated circuits, shortened to a 2 × 2 switch for simplicity; [KaSC91] is a 4 × 4 switch. We assume that packets are transferred on the links in the form of multiple words of w bits each, one word per cycle time, and that the link cycle time is equal to the memory cycle time. In the simple example of figure 2.1, the link throughput is 1 word/cycle, the packet size is 4 words (or a multiple thereof), and the width of the buffer memory is 4 words. Hence, the memory throughput is 4 words/cycle, which is equal to the aggregate incoming and outgoing link throughput. Since in the wide memory organization a packet can be stored or retrieved as a whole, it must first be fully assembled and then stored in memory or, for transmission, it must first be read into a latch and then transmitted, one word at a time. This assembly and disassembly is performed in the input and output latches. Double buffering is essential for the incoming packets, because at the time the last word of a packet has been received, it is not guaranteed that the wide memory will be available for storing the packet, while a new packet may arrive from the same link, overwriting the previous one. This is possible, since we assume that packet arrivals are not synchronized. As can be seen in figure 2.1, double buffering is also used in [KaSC91] for the outgoing link circuits in order to allow transmissions of packets on the different links to start in the same cycle, if desired; however this is a feature of the switch, not a requirement.

Care was taken to include *cut-through* in [KaSC91]. Cut-through is the capability of a switch to start transmitting a packet just after its header has been received and translated, and before the whole packet has been received. Cut-through is very important for low latency networks, especially those of parallel computers. In order to include cut-through capability to a wide memory shared buffer switch, an incoming packet must be able to be transferred to the appropriate outgoing link from the latches where it is being assembled, through paths other than the wide memory or its data bus. This is because a packet cannot be stored in the buffer before all of it has been received. Thus, tri-state drivers, bus wires and an output crossbar have to be added, as shown in the figure (2.1).

2.2 The Pipelined Memory Organization

The pipelined memory is a simplification of the interleaved memory, resulting when all accesses are sequential. In the pipelined memory, the address used in a bank in the current cycle is the address of the previous bank in the previous cycle. This organization simplifies control compared to interleaved memory and leads to a faster VLSI implementation, with fewer buffer peripheral circuits than wide memory, as will become apparent below.

Figure 2.2 is the pipelined memory equivalent of the wide memory shared buffer of figure 2.1. The pipelined memory version does not have double latches for the incoming links, or any of the outgoing link latches, and, most importantly, it does not need special cut-through circuits. On the other hand, an address register and a memory address decoder are added to each pipeline stage of the new buffer organization.

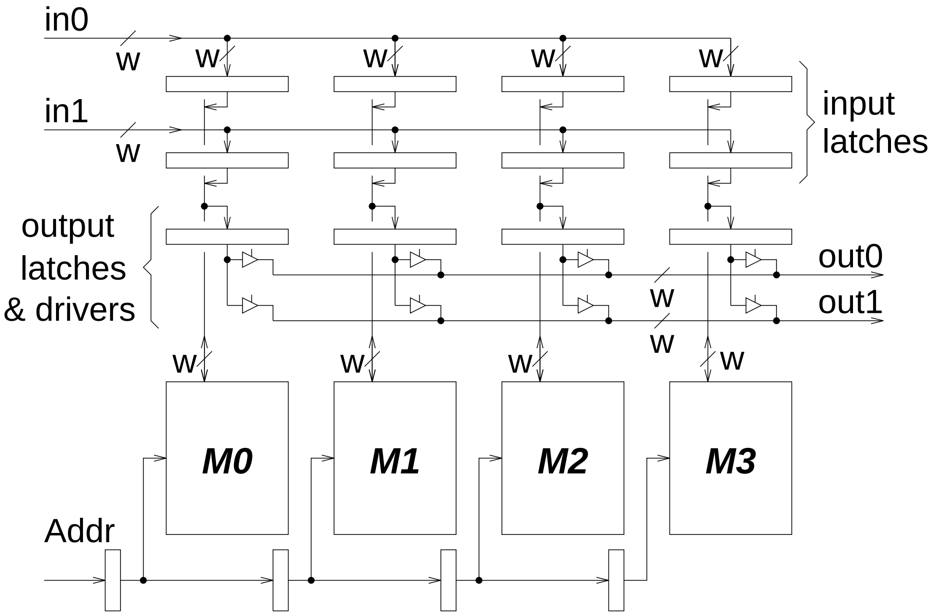


Figure 2.2: Pipelined memory buffer of a simple 2 × 2 switch.

As we have mentioned already, the shared buffer of a switch must have enough throughput for reading and writing as many packets as the total number of its ports (input and output), during a packet time. Thus, assuming a switch with N incoming and N outgoing links, and a link cycle time equal to the memory bank cycle time, a pipelined memory implementing the switch's shared buffer must have at least $2N$ pipeline stages, and the packet size is equal to $2N$ times the number of parallel data bits transferred by a link, or a multiple of this size.

As a packet arrives at a switch from an incoming link, at the rate of one word per cycle, it is stored in the latches below the link, with the first packet word occupying the leftmost latch and the following words occupying the rest of them from left to right, like a wave. At some cycle after the first word of the packet has been stored in the leftmost latch, and before the arrival of a new packet[1] from the same link, the first packet word will be copied from its latch into the first memory bank M0. In the subsequent cycles, the other words will be written into banks M1, M2, etc., again at the rate of one word per cycle. The words of the same packet are stored at the same address in all the memory banks. Note that a new packet from the same link can start being stored in the link latches or even in the buffer, before the tail of the previous packet has been stored in the buffer. This is perfectly legal and will occur almost invariably if the links are highly utilized. This will not result in any loss of data, because the write of the first packet is initiated before that of the second, and the two "waves" of writes proceed rightward at the same rate. Thus, unlike the wide memory scheme, pipelined memory does not need double buffering. In the same manner as with writes, when a packet is to be transmitted, a read "wave" is initiated at the first memory bank and as soon as the data are read out, they are fed to the bus wires of the selected outgoing link.

The exact cycle when a packet write or read is initiated depends on the outcome of the resolution of contention between the incoming and outgoing links. Normally, the outgoing links are serviced first, so as to maximize link utilization.

In the pipelined memory organization only one latch (with one word per stage) is used for the outgoing links, compared to the one latch per link of wide memory organization. This imposes the restriction that only one packet transmission can be initiated during any cycle; however, the additional packet latency caused by this restriction is negligible as explained in [KaVE95], section 3.4.

Cut-through is automatic in the pipelined memory architecture; if a read happens to be initiated before the packet write wave has reached the last stage, it constitutes cut-through. In the implementation described in chapter 3, the two waves of write and read can even occur simultaneously and as each memory stage is being written to, the new data are also made available at the memory output[2]. Therefore, cut-through is performed through the pipelined memory, saving the considerable silicon area of tri-state drivers, wires and crossbar needed by the wide memory organization.

---

[1] Assuming a packet size equal to the total word-width of the pipelined memory, not a multiple.

[2] In our implementations the memory banks have two separate unidirectional memory data buses.

2.2.1 Control of the Pipelined Memory

As we have mentioned, the control of the pipelined memory is simpler than that of the other shared buffer organizations. At every cycle, a new operation can start in the first stage M0 of the pipelined memory; we can start reading a new packet for transmission, or we can start writing a packet from the incoming link latches. Loading a new packet into the incoming link's latches is independent from the rest of the pipelined memory operations; many incoming link latches of the same stage can be loaded simultaneously.

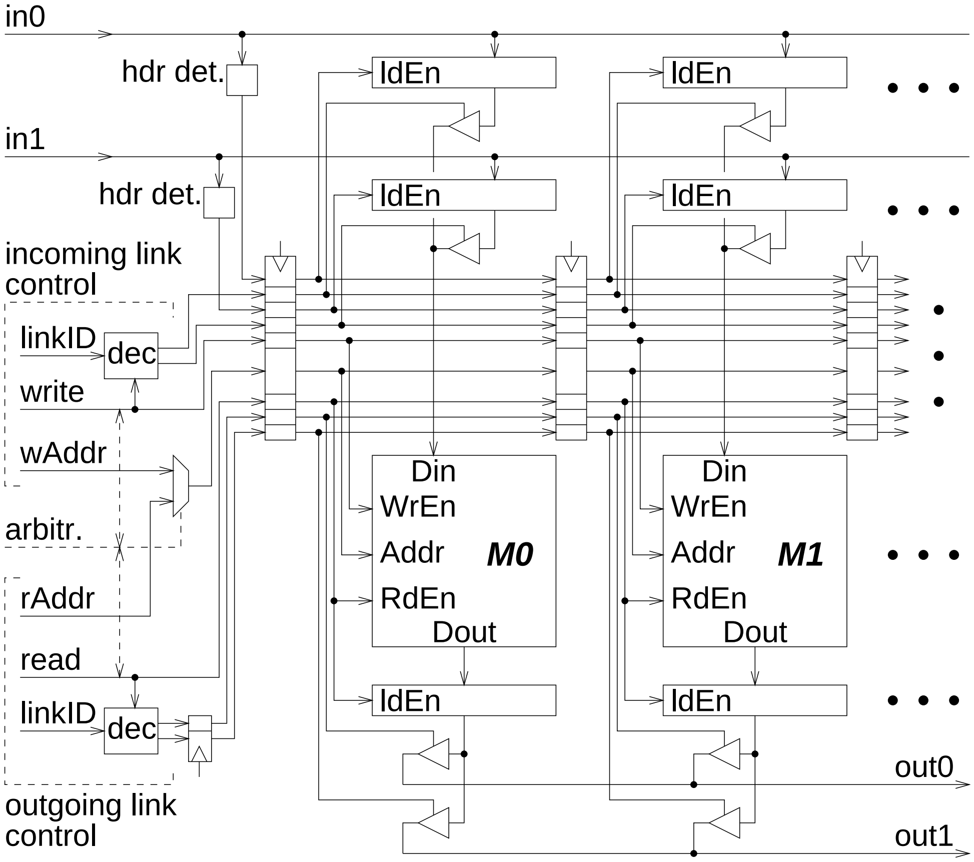
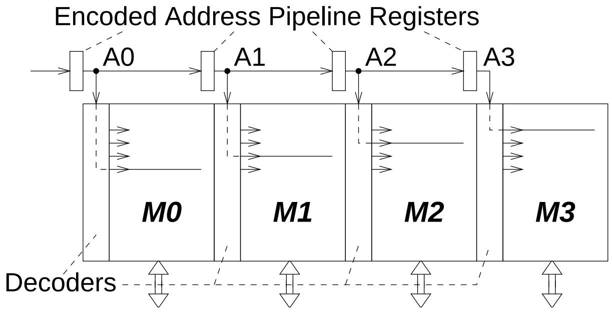


Figure 2.3: Control circuits of the pipelined memory buffer.

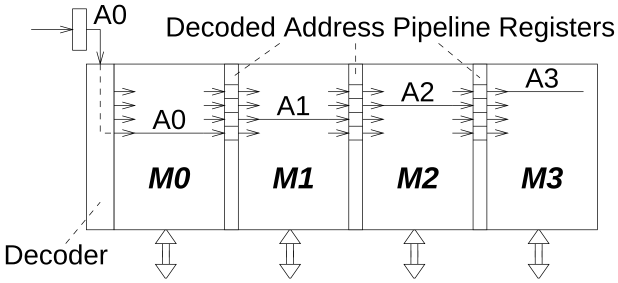
The control signals for the pipelined memory, including the incoming link latches' load enables, are generated for the first stage only. The control signals of the subsequent stages are delayed versions of those of the first stage. Figure 2.3 shows the control signals for the pipelined memory of figure 2.2. Whenever a new packet arrives from a link, it is detected by a link header detector (hdr det.) and the detector's output is used as the load enable signal for the link's latches of the first stage. This signal is then used as the load enable signal for the latches of the subsequent stages, delayed by the appropriate number of cycles. For each read or write initiation, an outgoing or incoming link (linkID) is specified respectively, besides the memory address.

The circuits that provide these signals are independent of the pipelined memory and their choice is orthogonal to this shared buffer organization.

2.2.2 Pipelined Memory Optimizations in VLSI



(a) Traditional



(b) Novel organization and floorplan

Figure 2.4: Pipelined memory in VLSI.

A pipelined memory consisting of traditional RAMs can be arranged as in figure 2.4a. This arrangement has more address decoders than wide memory approach, but it has shorter, and thus faster, wordlines. Actually, the RC delay of the very long wordlines of wide memories makes the memory access cycle long, and so most wide memories are split into many narrower ones, in order to achieve a short memory access time. Thus, for comparable memory access cycles, the area overhead of the additional decoders of the pipelined memory is not very large.

In fact, the extra pipelined memory decoders can be removed if we make the following observation: since the memory stages are placed next to each other, we can transfer the decoded address (the wordlines actually) directly from one stage to the next without redecoding the address at every stage. This is shown in figure 2.4b. With this optimization, each decoder is replaced by a column of wordline pipeline flip/flops, which is often smaller, faster, and may also be less power-consuming, than the decoder.

Figure 2.5 shows a variation of the VLSI pipelined memory that will be appropriate in future very-high-speed IC technologies, when on-chip clock skew will be considerable relative to clock period, and when the buses around the shared buffer and the bit-lines will have high RC delays. Even though these incoming and outgoing link buses have a limited capacitive load (compared e.g. to RAM word-lines), they will also suffer from RC delays in future high-speed technologies, because of the continued technology trend for shrinking geometries and increasing clock frequencies. At that point, the solution will be to pipeline these buses, as shown in figure 2.5, "extra pipeline stage (a)". In this figure, only one of the input and one of the output buses that will be present in a shared buffer are shown. Each of these buses is divided into two pipeline stages in figure 2.5; more stages are also possible. The number of stages in all input buses must be the same, and it must also equal the number of stages in all output buses. At the point where each I/O bus is split by a pipeline register, one extra decoded-address pipeline register must be inserted into the pipelined RAM. Any word of a packet that is stored in memory stages M0 through M3, in figure 2.5, passes through 2 pipeline registers on its way in, and passes through 3 pipeline registers on its way out. On the other hand, words stored in memory stages M4 through M7 pass through 3 stages on their way in, and 2 registers on their way out. Consequently, the total number of clock cycles spent in going through the I/O pipeline is the same (5 cycles) for all words of every packet; this observation is especially useful in understanding the timing of cut-through operation.

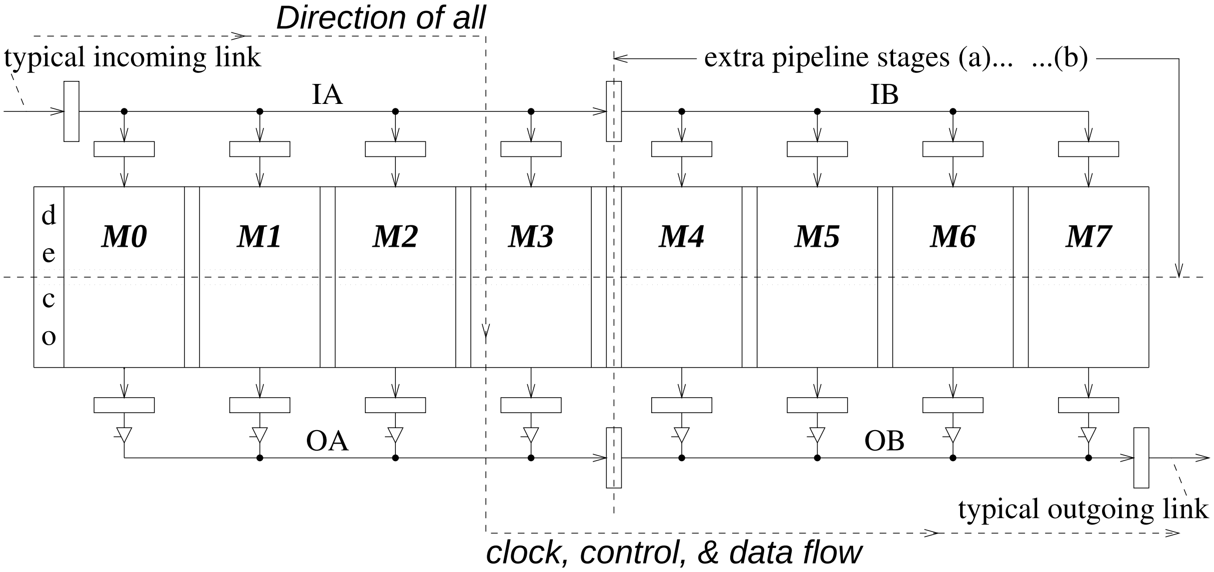


Figure 2.5: Pipelining the input and output buses and the bit-lines of pipelined memory.

Another feature of the VLSI floorplan of figure 2.5 is that data *always flow in a single direction*, and hence control and clock signals can also be made to always flow in the same direction. To achieve this, the outgoing and incoming links are placed on opposite sides of the memory arrays. By making all signals flow in the same direction and along paths of similar length, it is possible to match (to some degree) the RC delays of the wires that carry these signals, so that their *relative skew* is minimized, which will be crucial in the high-speed technologies of the future.

Finally, if the memory arrays have many rows and the bit-line drive time becomes too long, bit-lines can also be split into multiple pipeline stages, as shown in figure 2.5, “extra pipeline stage (b)”, provided that all memory stages as well as the address decoder are also split in the same manner. This arrangement adds extra pipeline registers in the vertical flow; all packet words passing through the switch do so through an equal number of stages, thus preserving their relative timing. During each memory write or read access, only one of the bit-line stages performs the access, while the others merely pass the information from top to bottom.

Chapter 3

VLSI Implementation of a Pipelined Shared Buffer

The pipelined memory organization is used as the shared buffer of the two Telegraphos [Kate94] switches that were designed by our group. The first is a printed circuit board implemented using FPGAs and commercially available chips, and the second is a single chip design using a 0.7 micron, semi-custom VLSI technology. Since pipelined memory is a regular structure, it can benefit greatly by full-custom design and because in neither of the switch designs, could the VLSI optimizations described earlier be applied, we decided to design another version of the pipelined memory using full-custom techniques.

This chapter presents the specifications of the pipelined memory that we implemented, its microarchitecture and circuits, and the most important decisions that we made at various steps of the design.

3.1 Specifications

We chose to design a pipelined memory for an 8×8 switch, with 16 bits per port, 256 packets of storage, and 256 bits per packet. The first candidate for the packet size was the ATM cell size but that would make our implementation too wide for our $1mum$ technology, without folding the pipelined memory. There is no interest in folding for our purposes though, as it is a simple extension of our implementation and our target in this work was the full-custom implementation of an extendable pipelined memory, not the design of a full ATM switch. Thus, we chose a big enough packet size, leading to a pipelined memory area which is realizable with the technology available to us. Folding is one of the floorplan organizations that were investigated and it is presented in section 3.2.

We aimed at a clock frequency of 100MHz, which we achieved for the typical operating conditions; the worst case clock frequency is 62.5MHz. Thus the aggregate throughput of this memory is 25.6 Gbits/sec for the typical case and 16 Gbits/sec for the worst. For an ATM cell size and the same clock frequency, these would be 42 Gbits/sec and 26 Gbits/sec, respectively.

According to the targeted port number and bits per port, the pipelined memory that we implemented has 16 stages with 256×16 bits of memory in each stage and the packet size is 256 bits. Each of the 8 incoming links consists of a bus of 16 bits of data which run above all stages, plus 2 bits of control that are latched and re-generated at every stage. Of these two control bits, one is the load enable of the incoming link latches and the other enables the corresponding link register to drive the memory input bus. The outgoing links need a single bit of control, which is latched and re-generated at every stage, and, when asserted, drives the 16 outgoing link data wires from the memory output of the appropriate stage. Besides the incoming link data wires and the control signals of the first stage, the other inputs of the pipelined memory are an 8 bit address, a memory write signal, and a clock input. There is no read signal because the memory always performs reads, that is, amplifies the voltage difference on the bitlines and drives the memory output bus, even when a write is performed. In that case, the newly written data are read out. The outputs are the 8 outgoing link buses, which are each 16 bits wide.

Static memory is used instead of the more area efficient dynamic memory for simplicity of our design. Pipelined memory can equally well be implemented with dynamic RAM. Its throughput would have to be somewhat increased so that refreshing will not interfere with its normal operation, but this may not be necessary, as the time a packet waits in a switch buffer will normally be less than the charge decay time of dynamic memory cells.

The target technology is the European Silicon Structures (ES2) $1.0\mu m$ CMOS technology with two metal layers and one polysilicon layer, operating at a supply voltage of 5 volts. The SPICE model parameters we had for this technology were old and they had significant differences in some parameters between consecutive versions, which made us suspicious about their accuracy. So we were very conservative designing our circuits, to be assured that they work under all conditions. Ironically, we received a new version of these parameters just as we were submitting the test chip for fabrication. For accuracy we will use the new parameters for the circuit simulations that will be presented, but the reader should keep in mind that all of the circuits were designed and optimized by simulating them with the previous model version.

3.2 Floorplan Organization

We found many ways to physically organize the pipelined memory. The most important of them are described here, using the specifications stated earlier (port number, bits per port, etc.). In these organizations, we assume that the incoming link latches and the memory input drivers are physically placed under the incoming link wires and the same holds for the outgoing link drivers. This assumption is realistic, and it has been easily achieved in our design.

In the following diagrams describing these organizations, the circles are either incoming link latches and memory input drivers, or outgoing link drivers, while the small rectangles are pipeline latches. Moreover, solid lines represent data wires and dashed lines represent control wires.

3.2.1 Alternatives Considered

First Alternative Organization

The first organization is shown in figure 3.1. This is the organization that was finally chosen for implementation. The memory banks are arranged as 256 bits high by 16 bits wide each, and they are placed side by side in a row, with wordline pipeline latches between them. The bitline drivers, the incoming links, and their associated circuits are placed above the memory banks, whereas the sense amplifiers and the outgoing links are placed below the memory banks.

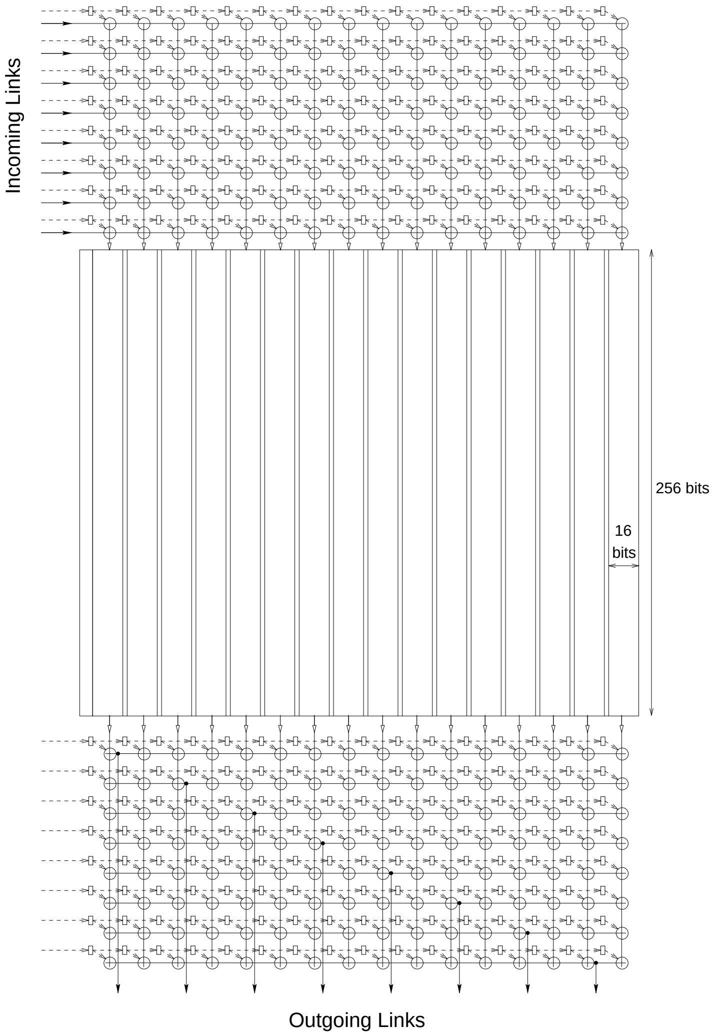


Figure 3.1: First alternative organization of the pipelined memory.

Although it may seem natural for the outgoing links to have their outputs on the right and bottom end of the block, this is not the case in our implementation. These link wires are long and carry large capacitance, so they need buffers (inverters) for amplification and effective driving of the external circuits. These buffers are hard to place horizontally next to the link buses, because the latter have a very small vertical pitch as they are placed with the minimum allowed distance between them. Our solution is to add a vertical wire as an extension to each link wire and route this extension through the outgoing link driver area to the bottom side of the block. In this way, the buffers are placed below the links, making their outputs available for connection at the bottom end of the pipelined memory block. The extra wire added does not increase the outgoing link wire capacitance too much because it is very short compared to the link wire's horizontal length.

Using a square memory cell layout, this microarchitecture leads to an almost square shared buffer block: there are 256 bits of memory in each dimension, and the total width of the 15 wordline pipeline latches plus the decoder is comparable to the total height of the incoming and outgoing link area.

Second Alternative Organization

An alternative way to organize the pipelined memory (see figure 3.2) is to split all of the memory banks in two parts, so that half of the packets are stored in the top row of bank segments and the rest in the bottom row. The links run horizontally between the two memory rows.

This architecture has the advantage of having half-height memory blocks with the bitlines carrying half the capacitance relative to the previous organization, which leads to a faster memory access time. There are some disadvantages though: the incoming and outgoing links are on the same side of the memory banks and consequently both the memory input and output buses cross them, making their cell's layouts wider than those of the first microarchitecture. In addition, the memory inputs and outputs are on the same side, as well, complicating the layout of the bitline drivers and sense amplifiers. The outgoing link buffers must be placed horizontally, as the vertical direction is now blocked by memory. Finally, the bitline drivers and the sense amplifiers are duplicated, since each memory bank is split in two.

Third Alternative Organization

The third organization (figure 3.3) is formed by placing half of the pipeline *stages* in one row and the other half in another, with the incoming and outgoing links placed in between. This *folded* organization is of interest in case where the width of the pipelined memory is much larger than its height. This can happen when the packet is large and/or when there are many pipeline stages.

The layout of this organization would be too tall, if we used a memory cell with an aspect ratio of 1, since there are four times more memory cells in the vertical dimension than in the

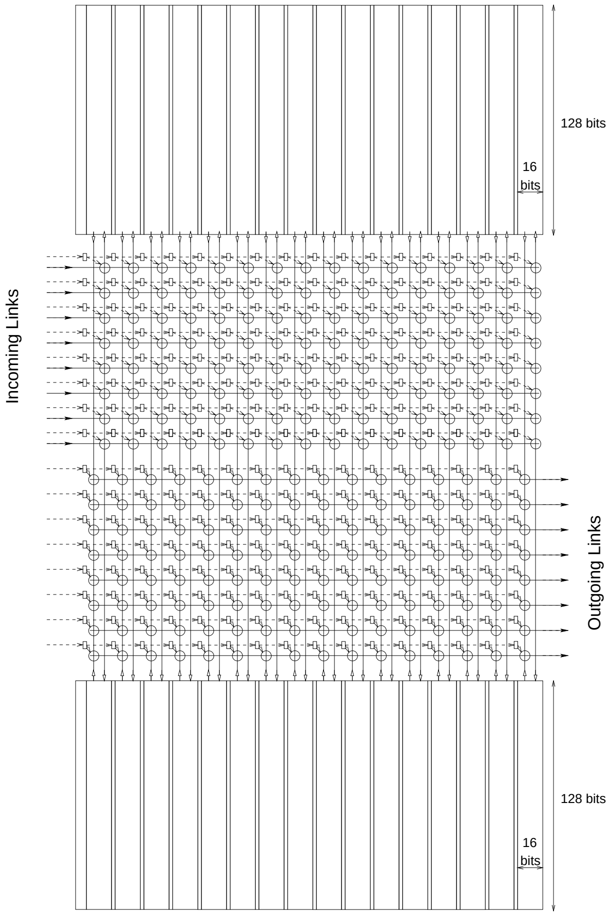


Figure 3.2: Second alternative organization of the pipelined memory.

horizontal, for a total memory size equal to that of the other organizations. Thus, the memory cell layout must be wide and short. Alternatively, column decoding/multiplexing can be used to achieve an aspect ratio close to 1 for the whole pipelined memory block. The disadvantages of the second organization concerning the layout of the peripheral circuits, are also present in this organization, but they are not important any more, as the width of the memory cell, and thus the width of the link associated cells, is inherently large. This organization also needs an extra decoder for the second row of pipeline stages.

Fourth Alternative Organization

Finally, the fourth organization (figure 3.4) is a variation of the first, but the basic idea behind it can be applied to the other microarchitectures as well. This *bit-sliced* organization divides each memory bank *word* and each link data bus in two (or more) parts. In this way, two pipelined memories of the first organization are formed, with half the word size and the link

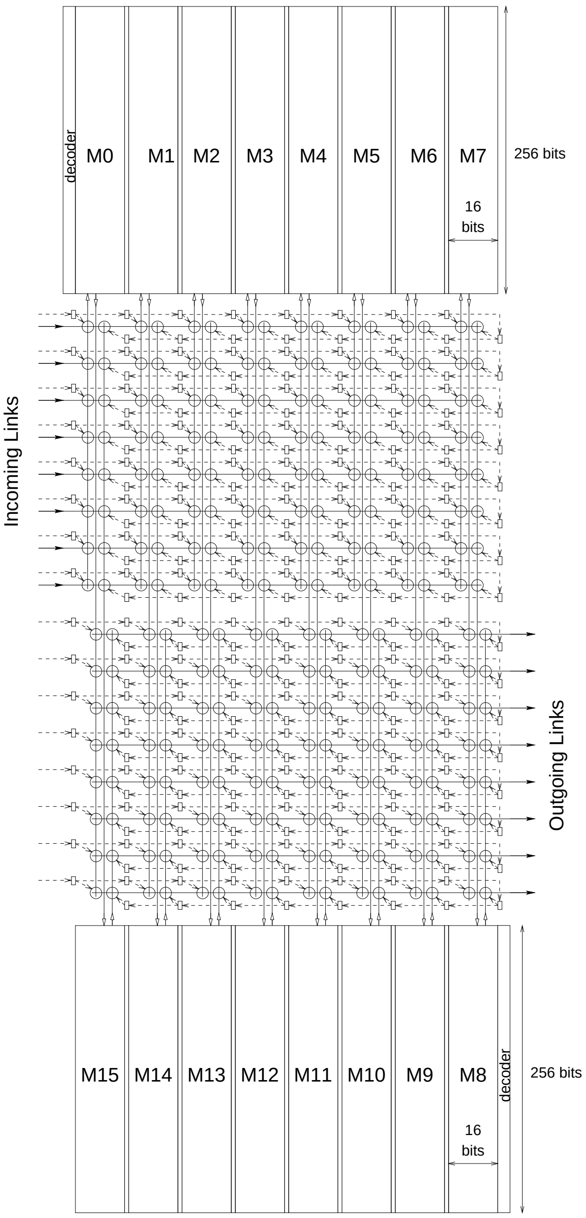


Figure 3.3: Third alternative organization of the pipelined memory.

data-bus width for each. The memory arrays of the two banks can be rearranged to use column decoding/multiplexing in order to keep the aspect ratio of the whole block close to 1.

Bit-slicing the pipelined memory reduces the link area by the factor of slicing. Let N be the number of incoming or outgoing links, w the width of the links' data-bus (and the word size of each memory stage), and $2Nw$ the total width, in bits, of the pipelined memory. Then, for the first organization, the area of the incoming (or the outgoing) links would grossly be Nw

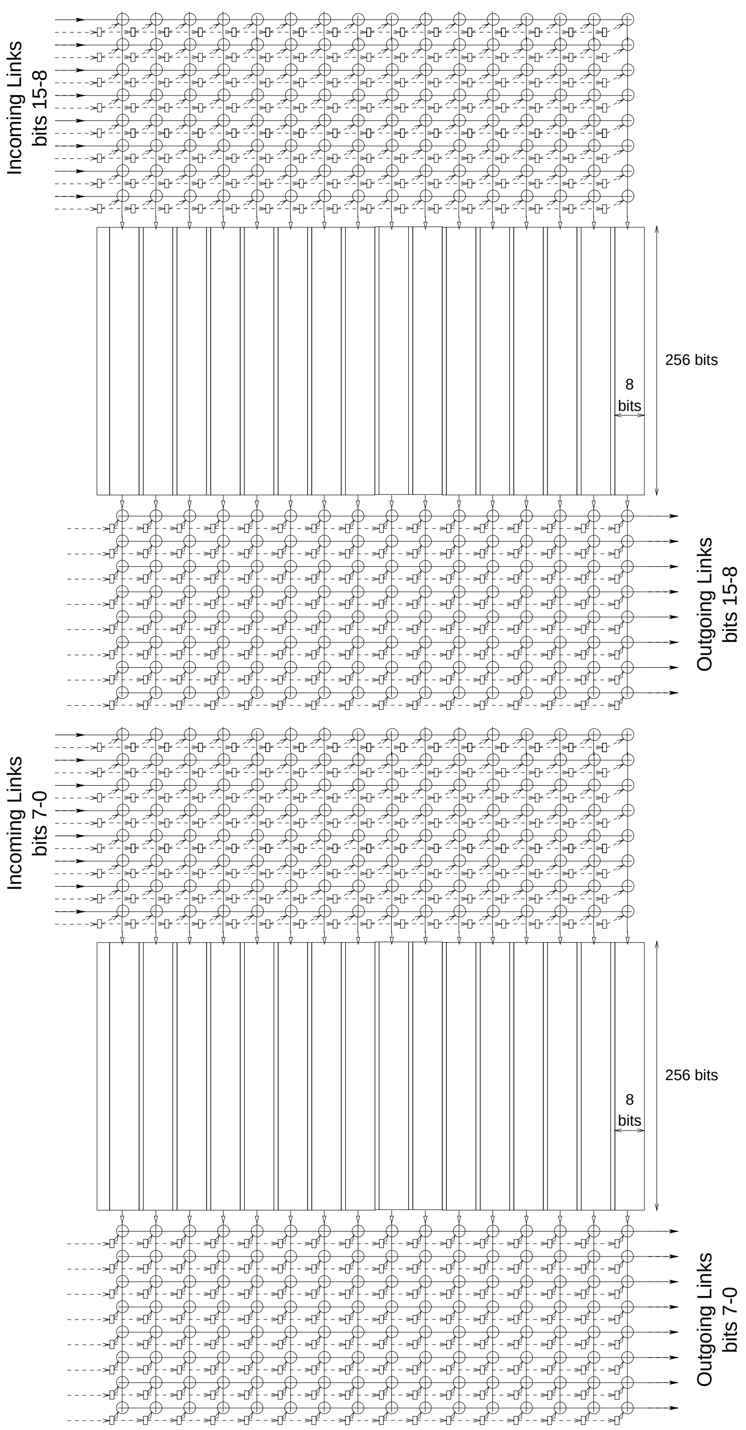


Figure 3.4: Fourth alternative organization of the pipelined memory.

wires vertically times $2Nw$ memory cells horizontally, that is a total of $2N^2w^2$. If this pipelined memory is sliced in s parts ($1 \leq s \leq w$), then each of these slices has an incoming (or outgoing) link area of $N(w/s)$ wires vertically times $2N(w/s)$ memory cells. The total link area for all the slices would be $s \cdot N(w/s) \cdot 2N(w/s) = (2/s)N^2w^2$.

Although the above results show that the link area is minimized when the pipelined memory is split in the maximum number (w) of slices, this is not true, because the above area estimates do not take into account the control wires needed for each link (load enable, memory input bus drive, and outgoing link bus drive) which contribute a factor of N times the horizontal width, and the area of the sense amplifiers and bit-line drivers (a big constant), which must be added to each sliced memory bank. Thus, bit-slicing is an interesting idea when the link data buses and memory bank words are wide, and saves considerable link area in many cases, but it cannot be applied to its extreme. In addition to the area savings, the small memory banks of this organization will have a shorter memory access cycle relative to those of the first.

We chose to implement the first organization, because it is the most straightforward, the easiest to lay out, and the other organizations are simple extensions of it.

3.2.2 Floorplan of the Implemented Memory

The detailed floorplan of the pipelined memory that we implemented is shown in figure 3.5. The width of each decoded address pipeline register is 2.2 times smaller than that of a normal address decoder, justifying the use of the first instead of the latter.

	Semi-custom	Full-custom	times better
Core Area	47mm^2	31mm^2	1.5
Periph. Ckt Area	128mm^2	12mm^2	10
Total Area	175mm^2	43mm^2	4
Cycle Time (worst case)	40ns	14ns	3
Aggregate Throughput	10Gbps	30Gbps	3

Table 3.1: Full-custom versus semi-custom comparison: 256-cell pipelined memory for a 16×16 ATM switch.

Table 3.1 is a comparison of a semi-custom and a full-custom implementation of the pipelined memory for a 16×16 ATM switch, with 256 cells storage, using the ES2 0.7 micron CMOS technology. The areas are calculated by transforming our 0.7 micron semi-custom Telegraphos II switch and this 1 micron full-custom design to the ATM cell size and the 0.7 micron technology. The full-custom implementation would be 3 times faster and 4 times smaller than the semi-custom, while the full-custom peripheral area would be 10 times smaller relative to that of the semi-custom. These area savings relative to the standard-cell version stem from the use of dynamic latches for storing the incoming packets, the use of precharging (actually, domino logic [KrLL82]) for all long wires that have multiple drivers (e.g. outgoing link buses), and the overlap of the peripheral circuitry with the link wire routing.

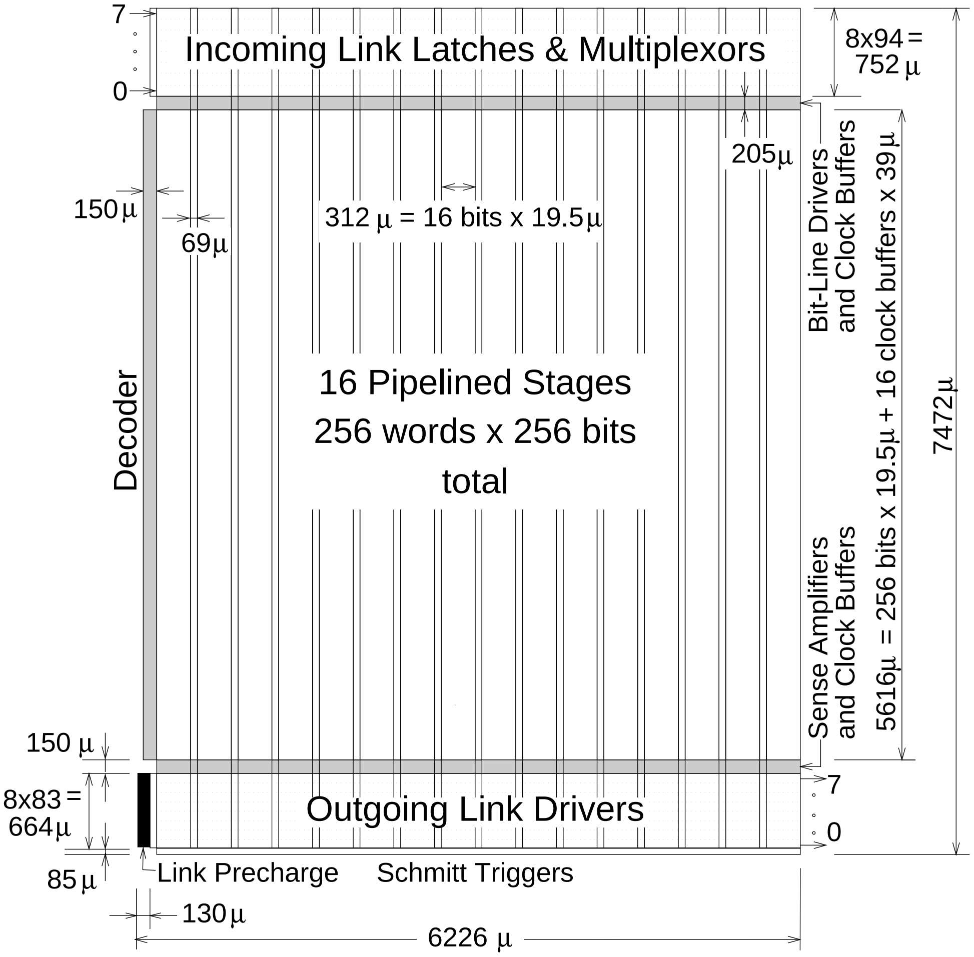


Figure 3.5: Floorplan of the pipelined memory that we implemented.

3.3 Detailed Circuit Description and Simulations

Figure 3.6 shows the circuits for one bit of the pipelined memory, the control signals and the clock phase when they are activated.

The clocking scheme that we use is pseudo-two-phase clocking, where only one phase is distributed in the full-custom block and the other is generated locally, wherever it is needed. Pseudo-two-phase is the simplest way to implement a two-phase clock, while the tri-state dynamic latches and the edge triggered latches that we use are both insensitive to the short overlap of the two phases.

Tables 3.2 and 3.3 summarize the clock phase when the basic operations are performed, and the polarity of the control signals and the clock phase when they are asserted respectively,

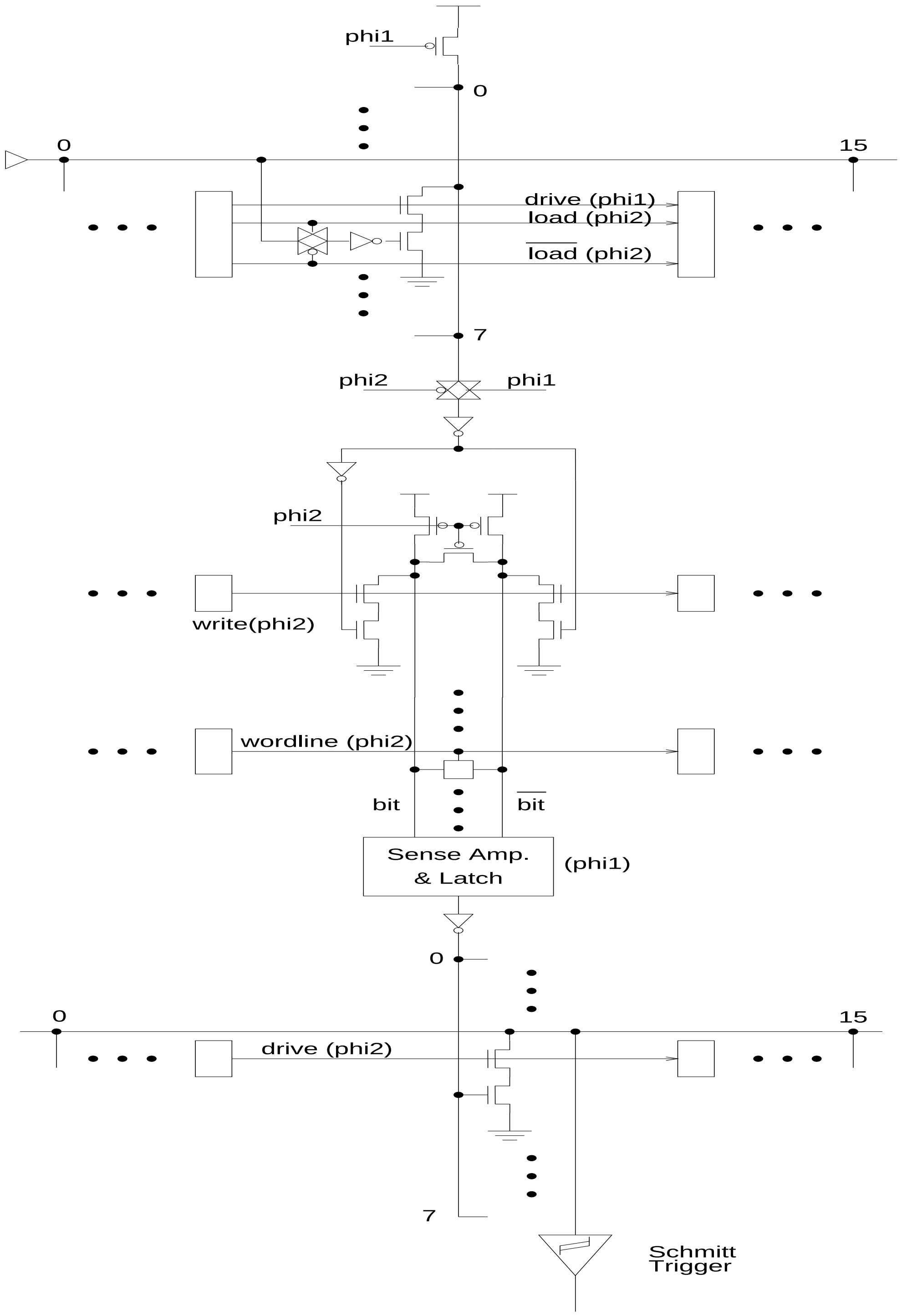


Figure 3.6: General circuit diagram.

for the reader's convenience.

Circuit	Activation phase
incoming link latch load	phi2
memory input bus: precharge	phi2
memory input bus: drive	phi1
bitline precharge and equalization	phi1
memory access, wordline drivers	phi2
sense amplifiers	phi1
outgoing link: precharge	phi1
outgoing link: drive	phi2

Table 3.2: Clock phases when the basic operations are performed.

Signal	Phase	Active
incoming link **load**	phi2	high
incoming link **loadBar**	phi2	low
memory input **drive**	phi1	high
write	phi2	high
outgoing link **drive**	phi2	high

Table 3.3: Clock phase and polarity of clock qualified control signals.

3.3.1 Memory circuits

The circuits that make up the memory blocks are SRAM cells, sense amplifiers, bitline drivers, the decoder, and the wordline pipeline latches.

SRAM Cell

We use the 6-transistor static memory cell, whose layout is shown in figure 3.7, as the building block of the memory arrays. Its size is $19.5\mu m \times 19.5\mu m$ and it has its contacts and wires along its perimeter shared with the adjacent cells. In the memory array, the ground lines are vertical, since they should always run in the bitline direction, while the supply lines are horizontal; they are connected at both of their ends to larger supply lines. We chose to run the bitlines in the second metal layer, because this layer, being farther from the substrate, has less capacitance than metal 1, and bitline capacitance is the most significant contributor to the memory access time. As a result of this decision, the ground lines, which also have a vertical direction use the second metal layer too, and the horizontal wordlines and VDD supply lines use the first metal layer. We included substrate and well contacts in each memory cell as an extra safeguard from latchup. Finally, we gained a reduction in bitline capacitance by sharing the drain area of each

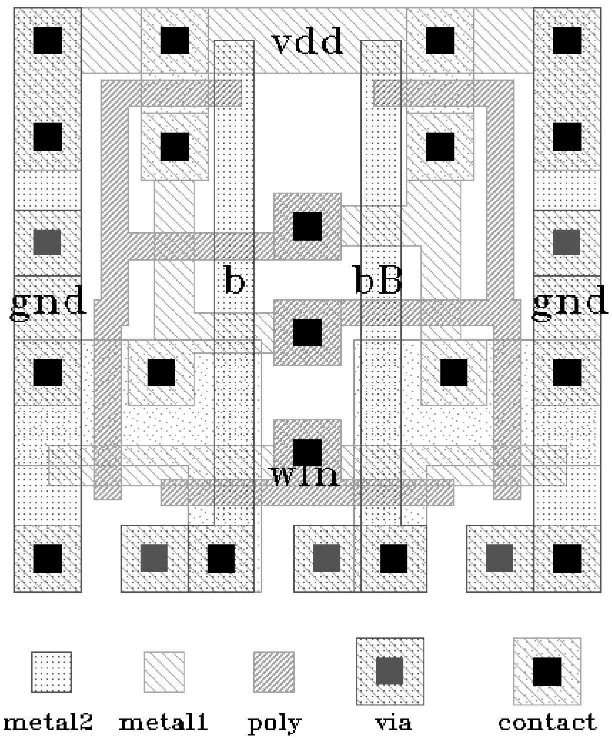


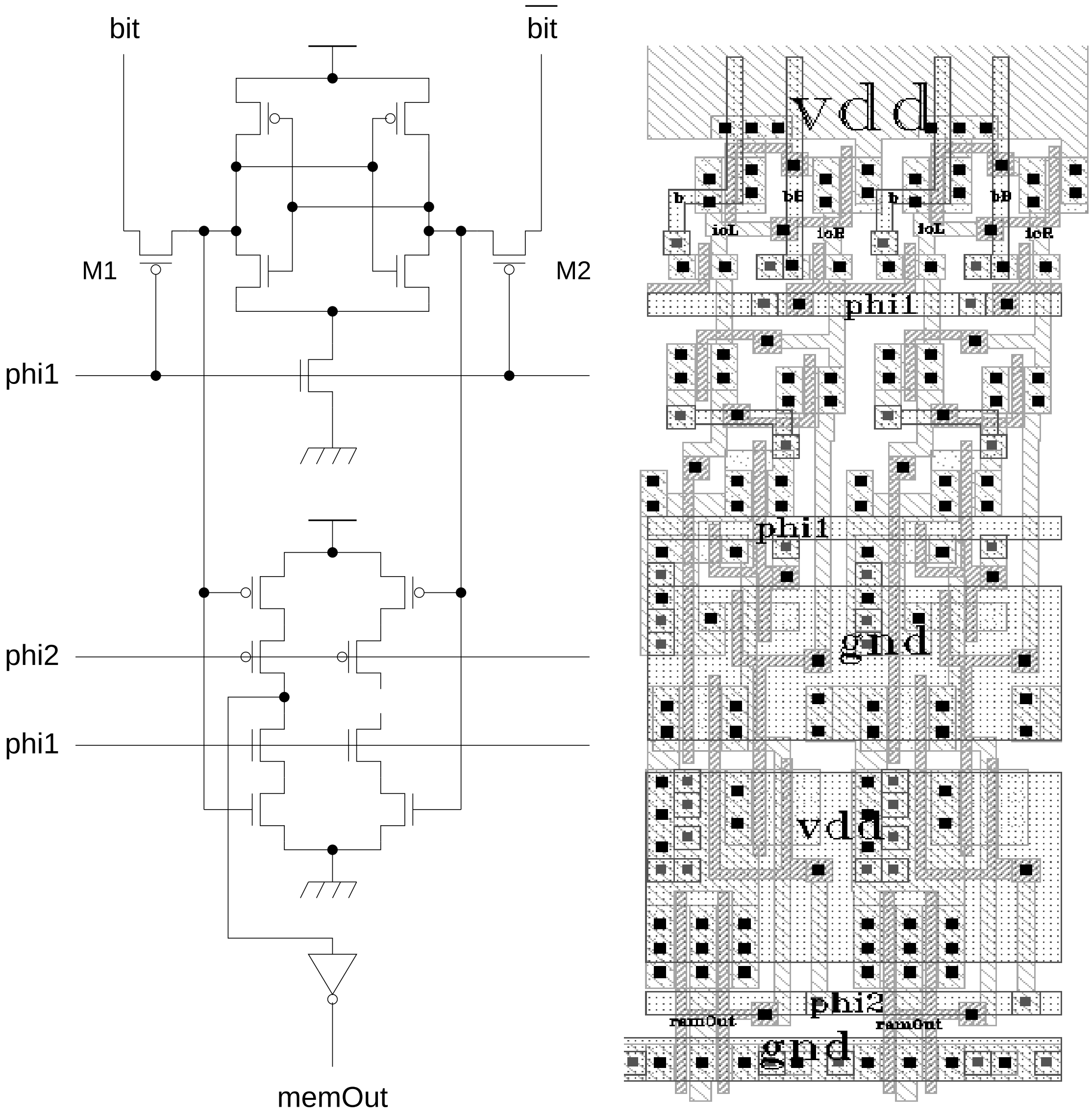
Figure 3.7: Layout of the SRAM memory cell.

pair of pass transistors that belong to vertically adjacent memory cells.

Sense Amplifier

Figure 3.8 shows the sense amplifier and latch schematic and layout. It consists of the differential sense amplifier itself, a latch, and an inverter driver for the memory output data line. This classical amplifier ([GrHo86] and [WeEs93]) consists of two inverters connected back-to-back. It employs positive feedback to achieve a very high gain, which makes the amplifier outputs reach the supply voltages very fast. When this happens, the amplifier stops consuming power, and behaves as a static latch.

During the phi2 phase of each cycle, the two PMOS pass transistors M1 and M2 are on and connect the sense amplifier terminals to the bitlines. Since the bitline capacitances are much larger than those of the sense amplifier terminals, charge sharing does not significantly affect the voltage of the bitlines, which in this phase are slewed by a memory cell, for read cycles, or the bitline drivers, for writes. At the same time the cross-coupled inverters' path to ground is cut by an NMOS transistor, disabling the amplifier. When the next phase starts, the PMOS pass transistors disconnect the amplifier's terminals from the bitlines and the amplifier's path



(a) Schematic

(b) Layout of two cells

Figure 3.8: Sense amplifier, latch and memory output driver.

to the ground is re-established. The voltage difference at the amplifier terminals grows quickly due the positive feedback loop and within a few nanoseconds the terminals reach the supply and ground potentials. From this time on, the circuit behaves as a static latch, until it is reset at the next phi2 phase, by cutting its path to the ground and shorting its terminals to the bitlines.

P-type pass MOSFETs were chosen to connect the bitlines to the amplifier terminals instead of transmission gates or n-type MOSFETs, because the bitlines are usually at high potential (close to the 5 Volts supply voltage) which does not pass efficiently through n-type transistors. Thus, the NMOS transistors of the transmission gates would be useless, while the amplifier would not function for reads if only n-type pass transistors were used. During write accesses, when one of the bitlines is pulled down close to the ground potential, the p-type MOSFET connected to this bitlines cannot pull the corresponding amplifier terminal to the same low voltage, but this is not important as the voltage difference between the amplifier's terminals will be high enough to ensure its correct operation.

The layout of the sense amplifier was drawn very carefully, so that the transistor sources of the two cross-coupled inverters are on the same side of the gate, and similarly for the two drains. This ensures good matching even in case of mask misalignments during chip fabrication. Care was taken to match the capacitances of the amplifier outputs, so, although only one of the outputs is needed, we also feed the other amplifier output to a "dummy latch". Due to the limited space of the cell layout, this "dummy latch" was left with two chains of NMOS and PMOS transistors unconnected, as can be seen in figure 3.8(a). With this change, the two clock-controlled transistors do not serve any purpose and could have been omitted to decrease the capacitance on the clock wires. Moreover, the capacitance matching of the two sense amplifier terminals is not perfect due to the use of different latches, as the Miller capacitance is not present in the "dummy latch". This mismatch is insignificant though, because the capacitance due to the Miller effect is much less pronounced for a two (or more) transistor chain.

Many other alternatives were investigated for the sense amplifier circuit but we rejected them due to static power consumption (CMOS differential amplifiers e.g. that of [WeEs93]) or due to their complexity, e.g. [Seki93].

Bit-Line Drivers

The bitline driver cell is shown in figure 3.9. It consists of two parts: the bitline precharge circuits and the write circuits. Two PMOS transistors (M1 and M2) precharge the bitlines at phi1 towards the 5 volts supply voltage. In the same clock phase, another PMOS (M3) shorts the two bitlines together to equalize their voltage, which is essential for the correct operation of the sense amplifier. This last transistor shortens the bitline precharging time, especially after write accesses, when one of the bitlines had been discharged. The write circuits are two NMOS transistors in series, for each bitline. One end of this two transistor chain is connected to a bitline and the other end to the ground. The gates of M5 and M7 are controlled by the write signal which is a qualified clock active in phi2, when a write access is taking place. The other two NMOS transistors M4 and M6 are controlled by the memory data input bus, with the correct polarity for each of the two transistor gates. Most of the transistors in this cell are very large so as to achieve fast memory accesses; in the layout most of them are broken up to two or more parallel transistors, to make the layout more compact.

At an earlier stage of the design of this cell, an NMOS transistor was also used for shorting

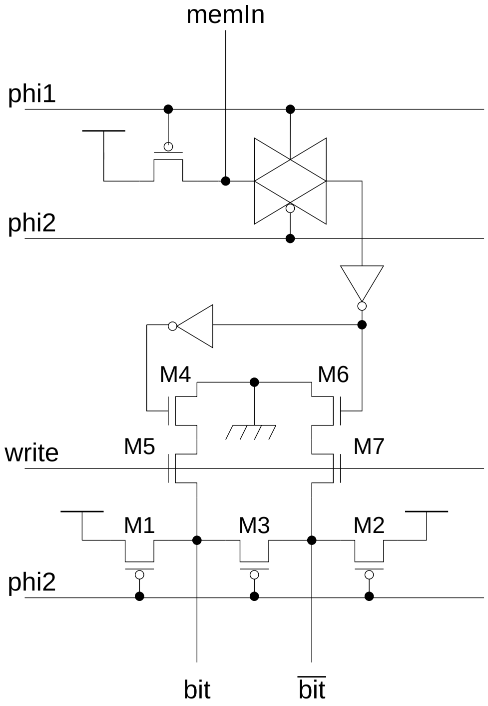


Figure 3.9: Bitline driver, memory input latch and precharge PMOSFETs.

the bitlines during the precharge phase in parallel with the PMOS one. This transistor speeds up the precharge operation but requires a different clock polarity from its PMOS counterpart, and we couldn't fit this extra clock driver in the very limited clock buffer cell area.

Wordline Pipeline Registers

The wordline or decoded address pipeline registers latch the wordlines of the previous memory stage and drive the wordlines of the current stage in phase phi2 of the next cycle. The schematic and the layout of one such cell can be seen in figures 3.10 and 3.11. We chose an edge-triggered register [YuKS87] (rising edge of phi1), instead of the usual two-phase tri-state dynamic latch, in order to only have to distribute one clock signal in the limited area of the wordline pipeline latches. Note that the two series PMOS transistors driving the wordline are adequate for the 16-bit long wordline of this implementation, but one may have to modify this driver if a larger memory word size is required. One interesting feature of the wordline driver is that it uses, in addition to the *phi1* controlled NMOS, another NMOS, controlled by the latched data, so that the wordline will not be left floating during *phi2* if it is not asserted.

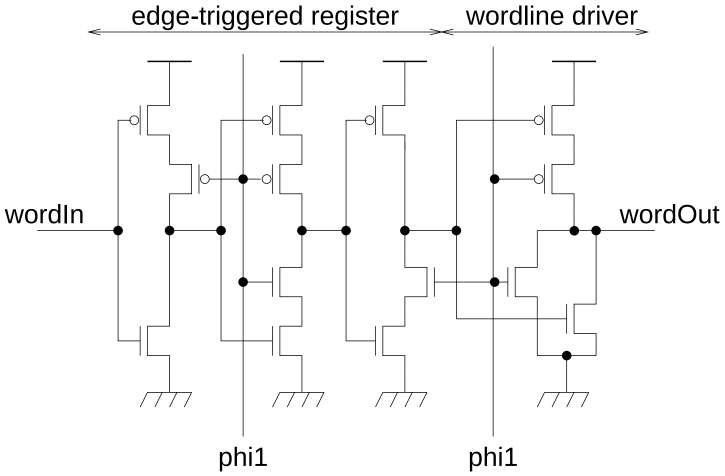


Figure 3.10: Wordline register and driver schematic.

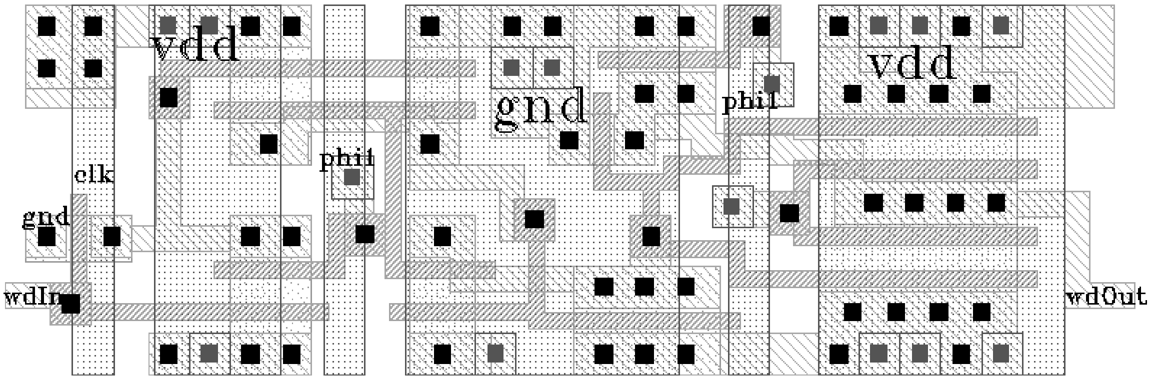


Figure 3.11: Wordline register and driver layout.

Decoder

The address decoder is a simple, 2-bit predecoded, 8 to 256 decoder. The decoder and predecoder basic circuits, along with the address latches, are shown in figures 3.12 and 3.13.

In the decoder circuit, although the basic *nand* gate is precharged, the classic *nand* PMOS transistors are also present. If these PMOS transistors weren't present, then the precharged *nand* would suffer from charge sharing between the *nand* output node and the small capacitors formed at the drains and sources of the NMOS transistor chain. We took care to make the redundant PMOS transistors of this *nand* small enough, so as not to load the address predecoder circuits too much.

Simulation

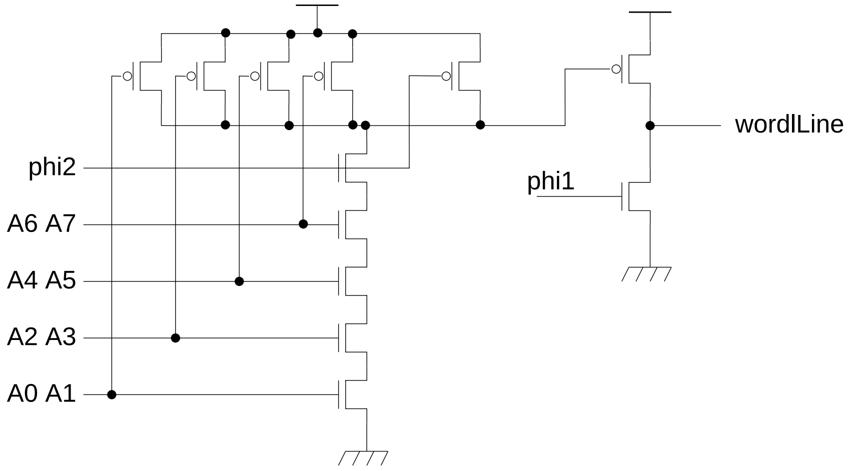


Figure 3.12: Decoder.

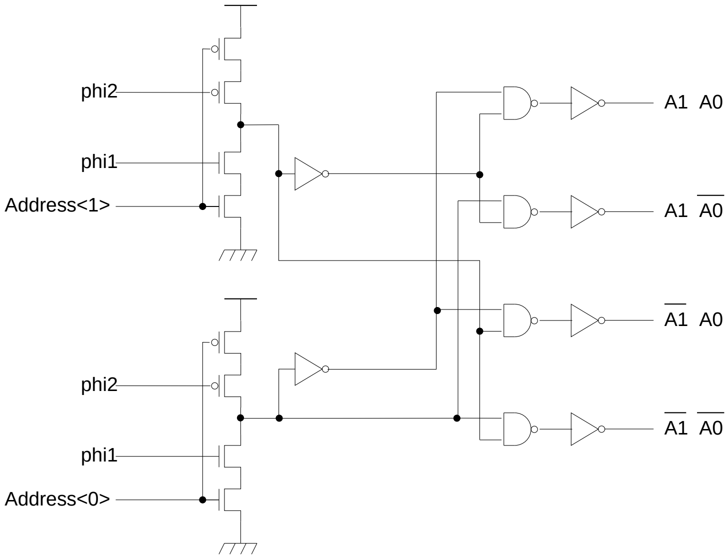


Figure 3.13: Predecoder and address latches.

Figure 3.14 shows the simulated operation of one stage of pipelined memory, in the worst operating conditions, i.e. 4.5 volts supply voltage, slow transistor models, high parasitics, and 125 degrees centigrade junction temperature. We present the waveforms of one pair of bitlines, the wordlines for two different words of memory, and the terminals of the sense amplifier that are connected to these bitlines. A write is performed, followed by a read of a cell with opposite data polarity which is connected to the same bitlines (**B** and **BBAR**). This read is then followed by another read of the previously written bit. The wordline waveforms (**WLN1, WLN2**) are produced by wordline pipeline latches and not from behavioural voltage sources. Of the sense amplifier terminals, **XSENSE.IOR** corresponds to **B** and **XSENSE.IOL** to **BBAR**. It can be clearly

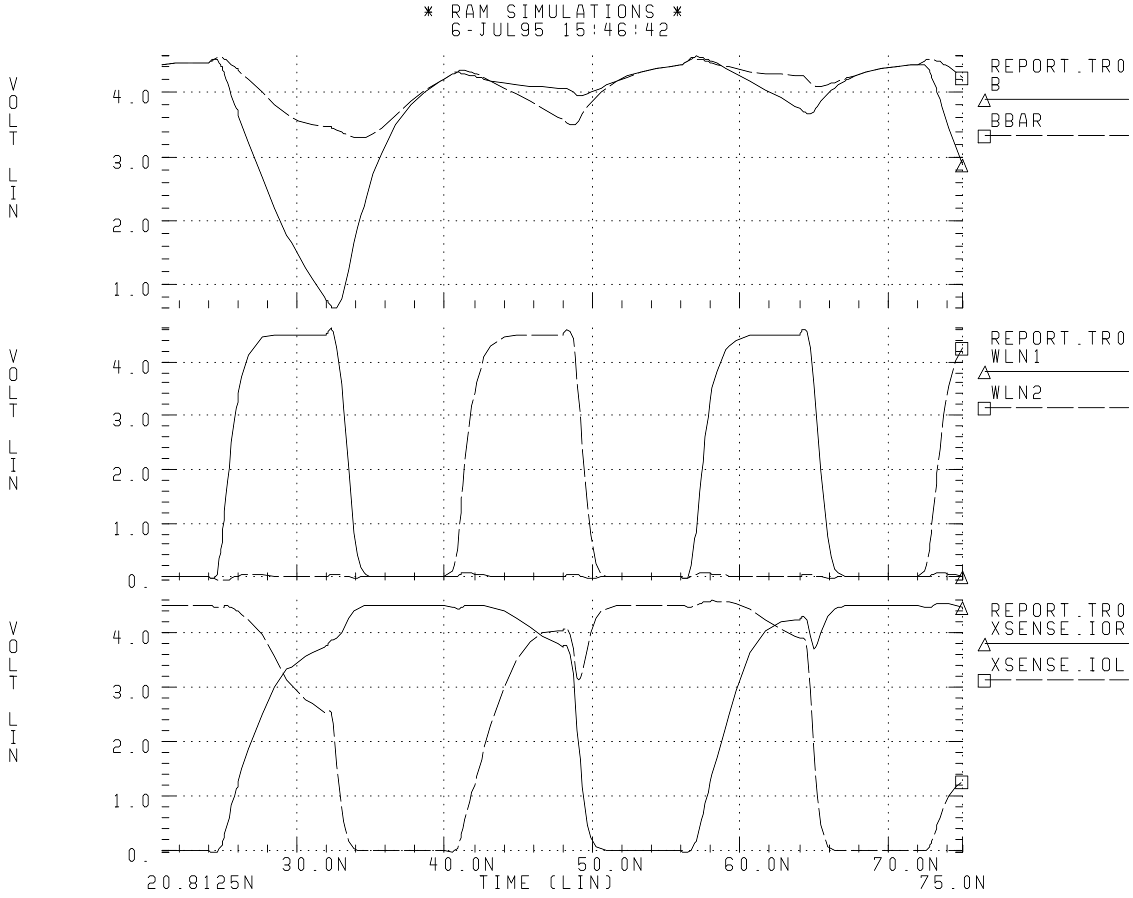


Figure 3.14: SRAM simulation.

seen at the time intervals 42ns to 48ns and 58ns to 64ns, when the memory cells drive the bitlines (at phi2), that the amplifier terminals have the same voltage with their respective bitlines. Within 2ns the sense amplifier has amplified its input and then behaves like a latch for the rest of the phi1 phase.

3.3.2 Incoming and Outgoing Links and Circuits

Incoming Links, Latches and Memory Input Drivers

The circuits associated with an incoming link consist of a tri-state dynamic latch and a memory input driver for each of the 16 bits of the link data bus, for each stage. The latch and driver for one bit are presented in figure 3.15. Packet data are stored in these latches for a short time, at most for as long as the reception of all the words of the packet, so dynamic circuits can be safely used. The two complementary load enable signals control all of the latches of the same

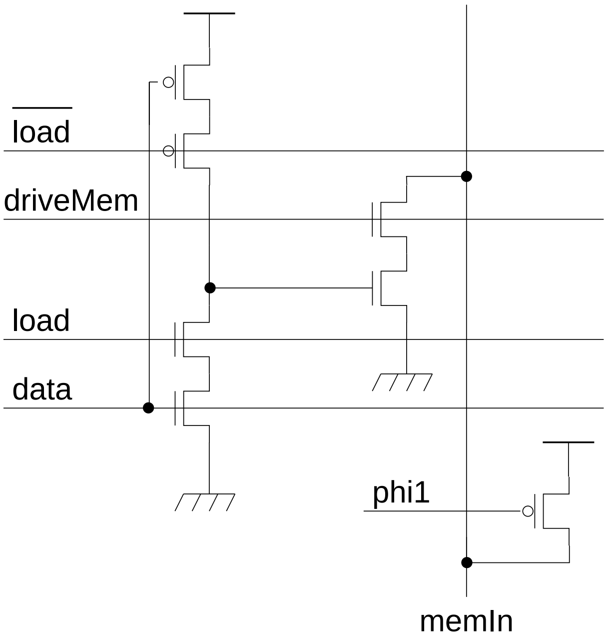


Figure 3.15: Incoming link latch and memory input driver.

link and stage, and are generated by the incoming link control signal pipeline latches. These load enables are qualified clocks, that is, they are active when a load enable is to take place and it is the right clock phase (phi2) to load the data. The memory input driver is just a chain of two N-type MOSFETs, which pull the precharged memory data input wire to the ground, when the corresponding latch holds a one and the qualified clock **driveMem** is high.

Physically, these circuits are placed under the incoming link wires, as can be seen in figure 3.16. With this layout, the incoming link circuit height[1] is 47% larger than that needed by the link data wires only, and 3.6% larger than that needed by data, control, and power wires. Note that due to the availability of only two metal layers in our technology and to the limited area for vertical wires, the power lines had to be run horizontally. The long link data wires, and thus all the horizontal wires of the incoming link area, are drawn in the second metal layer so as they have less capacitance and resistance than in metal 1. Care was also taken to share the drain of the memory input bus drivers between adjacent links in order to minimize the capacitance of the memory input wires.

Although there is some coupling between the incoming link data wires and the node where the latch stores the data, it is not possible for the latched data to be destroyed by this coupling. The maximum induced voltage when all of the link wires rise or fall within 1ns — which is almost impossible to achieve as they have large capacitance — is about 0.15 Volts. This voltage does not accumulate with time, as we show here. Imagine a rise in all of the link data wires which increases the potential of the storage node by 0.15 volts, due to the coupling effect.

---

[1] The width is pitch-matched to the memory.

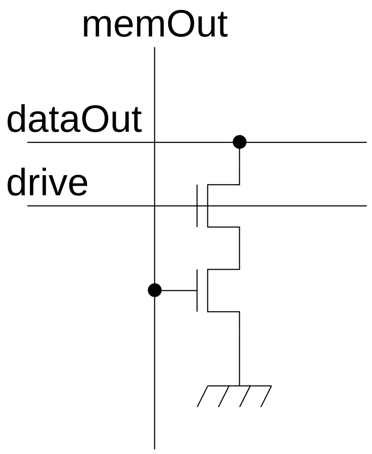


Figure 3.17: Outgoing link driver.

This dense placement of the data wires causes some coupling problems between them. For example, consider the case where one data wire is to be left charged at 5 volts, while its two neighbouring wires are pulled down to zero volts. Due to the capacitive coupling between the wires, which is large as they run in parallel for several millimeters, the middle wire will loose some of its charge and consequently, its voltage will drop. This drop was simulated to be about 1.6 volts in the worst case, and forced us to add Schmitt trigger inverters at the outputs, because plain inverters would not be able to keep their outputs within the appropriate CMOS signal levels when fed with such inputs. We use the CMOS Schmitt triggers of [Uyem88]. Alternatively, one could modify the link drivers to include a pull-up part, while keeping the precharge transistor as the main pull-up device, at the cost of increasing the capacitance of the outgoing link lines. Another solution is to use active pull-ups and let them consume current when the link drivers pull the wires down, or generate a reference threshold voltage and use analog comparators in the place of the Schmitt triggers.

As Schmitt triggers were found to be too slow to work at the same clock phase with the outgoing link drivers, we added one pipeline stage by means of a transmission gate, to act as a "sample and hold" of the outgoing link wire voltage, before it is applied to the Schmitt trigger. Finally, an inverter has been added after the Schmitt trigger to drive the external circuits.

Simulations

Figure 3.18 shows the simulated operation of one incoming link latch and its respective memory input line, in the worst operating conditions. For this simulation, the latch is loaded at the phi2 phase of each cycle, and it drives the memory input line at the next phi1 phase. The small drop on the memory input line voltage, **MEMIN**, in the interval 50ns to 58ns, is due to charge sharing between the memory input line and the diffusion area of the open NMOS transistor of the enabled driver, but it does no harm as it is only about 0.3 Volts, that is less than half the

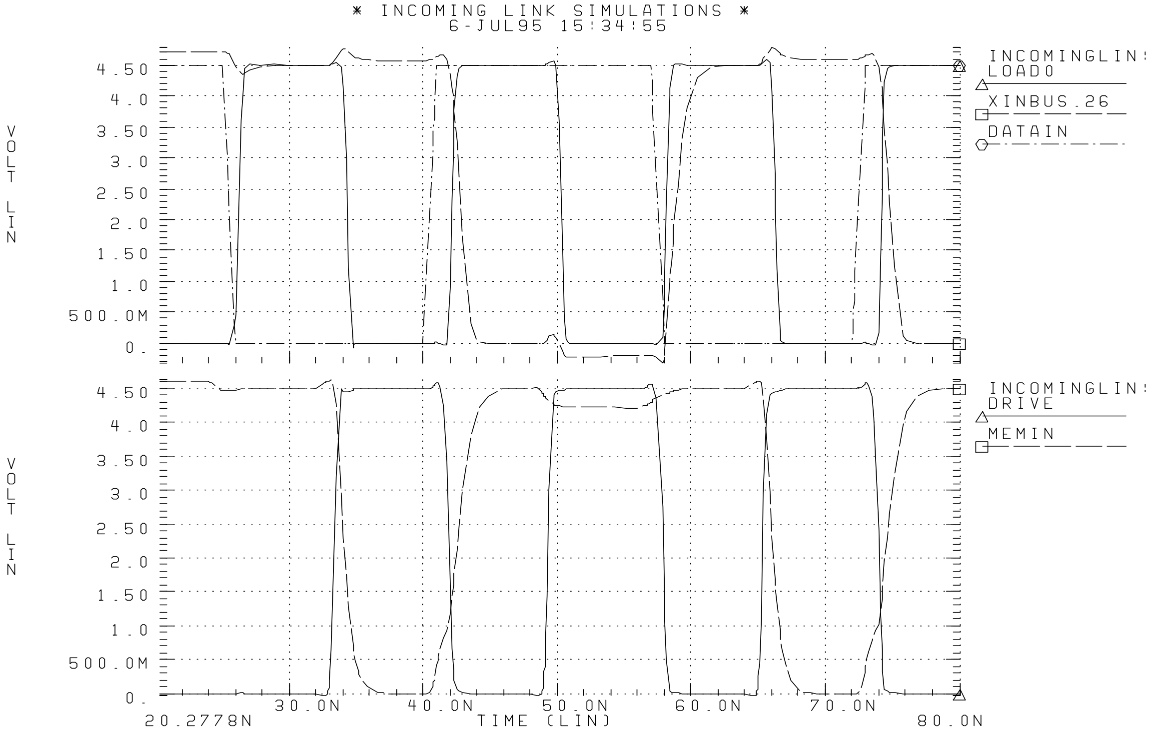


Figure 3.18: Simulation of the incoming link latch and memory input driver.

threshold voltage of a PMOS transistor.

Figure 3.19 shows the simulated operation of the outgoing link drivers and the Schmitt triggers. The results of three memory reads drive the memory output line **MEMOUT** and the outgoing link is driven after all of these reads[2]. The coupling problem identified earlier occurs in the interval 24 to 32ns: the line should stay at the 4.5 volt, for the worst case supply voltage, but coupling from the neighbouring lines drop it to about 2.75 volts.

3.3.3 Pipeline Latches

Besides the wordline pipeline latches, that were described in the memory circuits' section, pipeline latches are also needed for the control signals of the incoming and outgoing links. These circuits (figures 3.20 and 3.21) latch the control signal of the previous stage and apply it to the circuits of their own stage, at the appropriate phase of the next clock cycle. These latches are the classic dynamic tri-state latches, where one of the clock controlled transistors of the first stage is removed, since the signal is already qualified (ANDed) with that clock. Only

---

[2] This, of course, would never happen under normal operation, but it behaves identically to the normal situation where subsequent stage drivers drive the outgoing link.

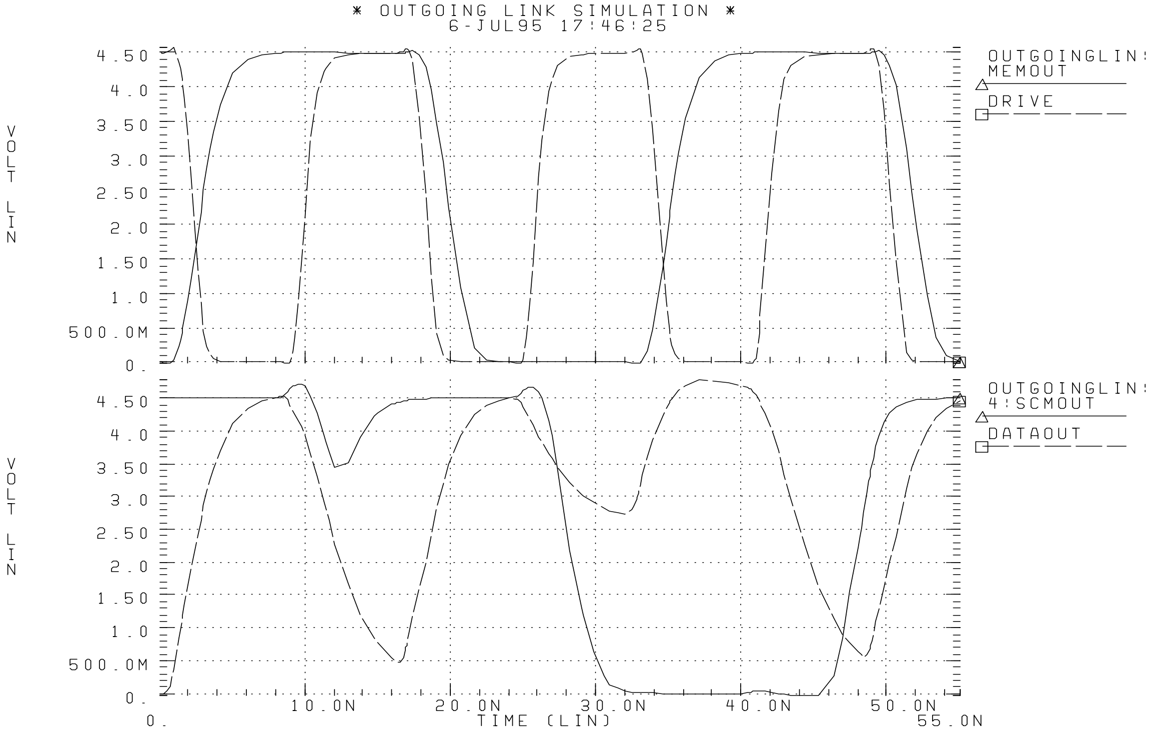


Figure 3.19: Simulation of the outgoing link driver and Schmitt trigger.

the pipeline latches of the first stage have this transistor, so that the external circuits will not have to qualify the input control signal with the appropriate clock phase, which is difficult to generate outside the full-custom block. Table 3.3 shows the polarity and the clock phase when these signals are asserted and figure 3.22 shows the layout of the incoming link control signal pipeline latch and driver.

3.3.4 Clock Buffers

The design and specifically the layout of the clock buffers were two of the most difficult tasks of this implementation, due to the large number of latches in the design and the high clock frequency. Our intention was to route the clock in the same direction as the data flow, from left to right and from top to bottom, so as to minimize the relative skew between the clock signals and data. The problem that we encountered and made us change the clock routing scheme, is the very high capacitance of the 256 wordline pipeline latches, which was impossible to drive from only one side of the latch array. Thus, we distributed the clock using two horizontal main lines, one above the memory array and one below it, both driven at their left ends, and we constructed a tree of clock buffers for each column of wordline pipeline latches. The final topology of the clock tree is shown in figure 3.23.

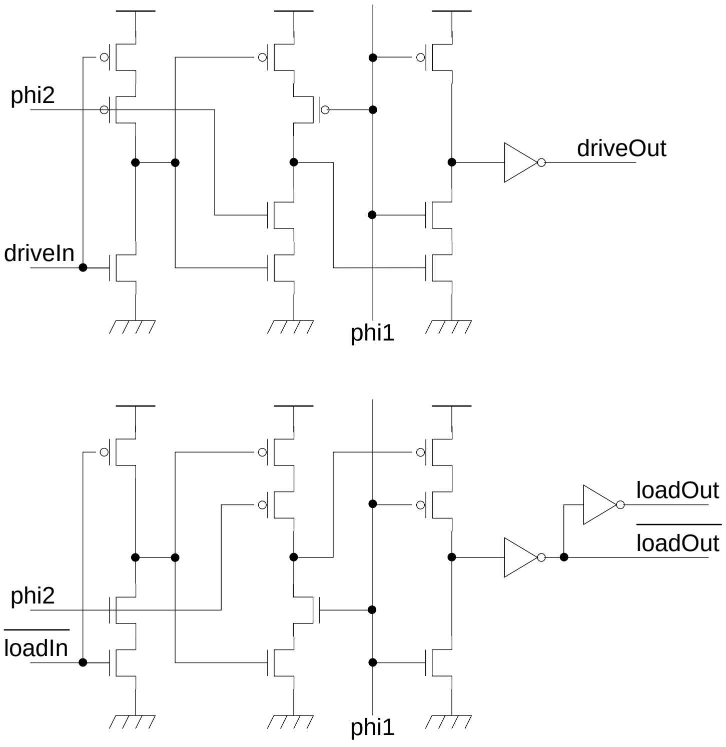


Figure 3.20: Incoming link control pipeline latches and drivers.

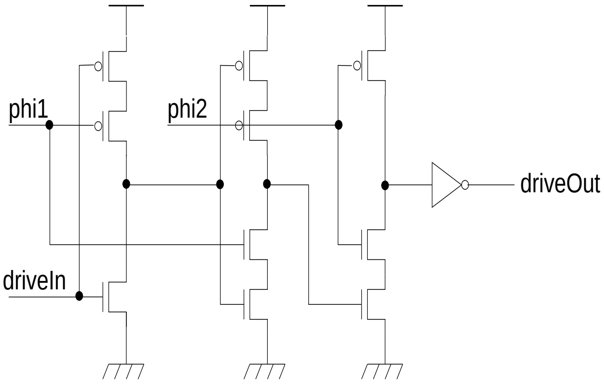


Figure 3.21: Outgoing link control pipeline latch and driver.

The 16 clock buffers of the two clock subtrees of the wordline pipeline latches area take up 12.5% of the total height of the memory array. Alternatively, instead of breaking the memory array vertically, the clock buffers can be placed at the left of the wordline pipeline latches increasing their width, but the area cost would be grossly the same. The other clock buffers are placed in the limited area between the sense amplifiers, the wordline latches and the control signals' latches. This is why their layout was so difficult to draw.

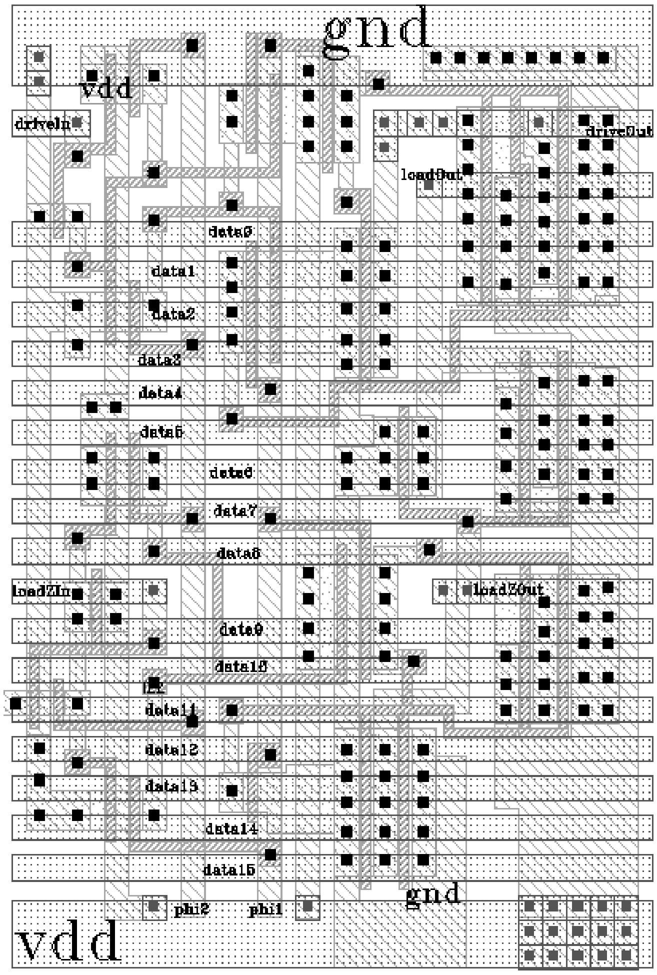


Figure 3.22: Layout of the incoming link control pipeline latches and drivers.

The worst skew is observed between the clock signals of the first stage and those of the last and it was found by simulations to be about 1ns. This means that the clock skew between two subsequent stages would be about 63ps, which is negligible. We are more interested in the clock skew among subsequent stages, because the pipeline latches of these stages are in effect the neighbouring latches of a shift register, which is the most sensitive circuit to clock skew.

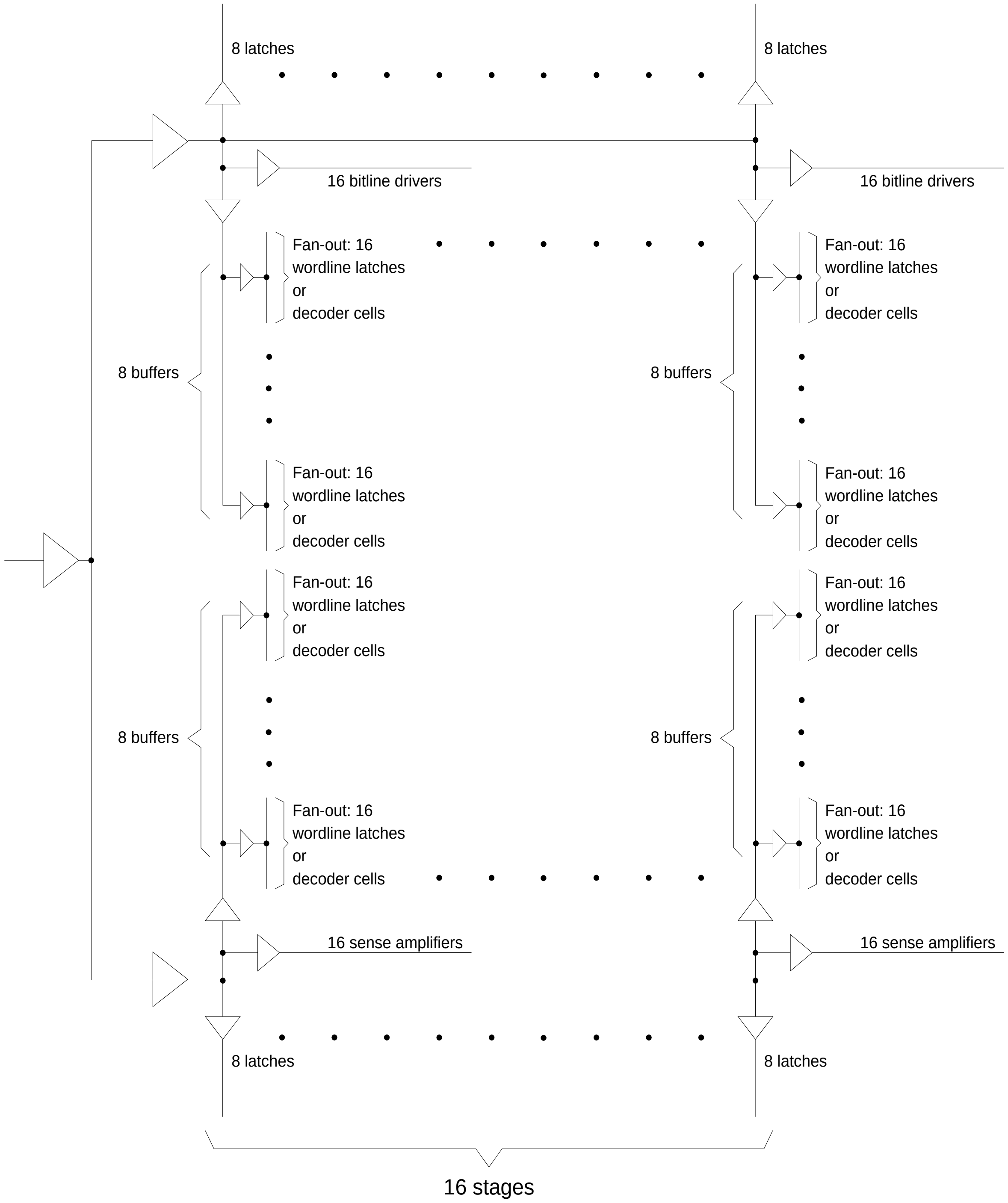


Figure 3.23: Clock tree topology.

Chapter 4

Pipelined Memory Test Chip

The pipelined memory described in the previous chapter was placed inside a test chip for fabrication and testing. The control and interface circuits of this chip are built using the standard cells provided by ES2 for their 1 micron technology. We did not design a full switch chip with this memory, because the architecture of the other parts of the next generation of Telegraphos switches was not finalized early enough to be included in this chip, and because the full switch would be more difficult to test. We submitted the $10.58 \times 10.18 mm^2$ test chip for fabrication through EUROCHIP in June 1995.

In this chapter, first we briefly examine the operations that should be supported by the test chip so that the pipelined memory can be effectively tested, and then we give a functional description of the test chip followed by details of its most important parts. We conclude this chapter with information about the design flow and the tools we used to implement the pipelined memory and the rest of the chip.

4.1 Test Operations Required

Evidently, the basic operations required to test a memory, are reads and writes, and actually, most memory testing schemes involve reading and writing various patterns to all the memory words in a pseudorandom sequence for many clock cycles.

However, in the pipelined memory, the banks are not directly controllable: they are read from or written to, through the link circuits. Thus, the basic operations available become **write packet** and **read packet**, which also check the incoming or the outgoing link circuits, at the same time. Allowing each basic operation to test two circuits at a time, has the drawback that in case of a fault, it is impossible to find which of the circuits is faulty, so as to redesign it in a later version of the chip.

The pipelined memory banks could be directly controllable, if we allowed each memory input wire to have one more driver besides the 8 incoming link drivers and if we made each sense amplifier output accessible to the test circuitry. This extra driver would be controlled by test circuits outside the pipelined memory block and the sense amplifier outputs would be latched and compared to the memory inputs. Unfortunately, we did not think of this early enough during our design.

Apart from the tests that check the memory of the shared buffer, we need to test the peripheral circuits, that is, the link circuits, at high speeds. This can be done by issuing consecutive packet reads and writes at high rates. Read packet and write packet operations suffice and it is their order and the idle time between them, that is important for these tests.

An additional operation needed is the simultaneous read and write of the shared buffer memory as an extreme case of cut-through which is supported by our design. A separate test operation is required for this check which actually performs a packet read and write simultaneously.

We decided not to use the scan-path testing method [IEEE90] in our full-custom block as the multiplexors and the control lines needed would offset much of the speed and area gained by designing the pipelined memory in full-custom. We did not use scan-paths in the semi-custom test circuits either, because they contain registers mostly, and if we added a multiplexor for each register, as is required for this testing scheme, the test chip would have been about double in size. For these reasons we content ourselves with functional testing only.

4.2 Functional Description of the Test Chip

Testing a chip with our pipelined shared buffer in it has two practical difficulties: the limited number of pins of the chip packages available to us and the high speed of the links. Due to the first, we cannot have all the links connected to chip pins, as we would need at least 272 pins: 16 for data, plus 1 to signal the arrival of a new packet, for each of the 8 incoming and 8 outgoing links. The high speed links make it very difficult to control and feed with data the test chip, because the supporting chips of a test board would have to work at the same speed and be synchronized with the test chip. This would make the test board overly expensive and its design time would be very long.

To overcome these problems, we include two asynchronous bit serial interfaces, in the test chip, one for input (serial to parallel) and one for output (parallel to serial), and we make only two incoming and two outgoing links directly accessible through the chip pins as figure 4.1 shows. Thus, we allow the external support circuits to run at a low speed, communicating information to the test chip through the asynchronous interfaces, and at the same time, we can observe the 2 external links, or connect more than one test chips together using these links. In order not to leave the other five[1] links idle, we connected them internally with the outgoing links in a loopback way.

To make the control easier, we make the convention that the most significant byte of the first word of a packet entering the test chip, either through the serial interface or through a link,

---

[1] The incoming and outgoing links 0 are hard wired to the input and output serial interfaces, respectively.

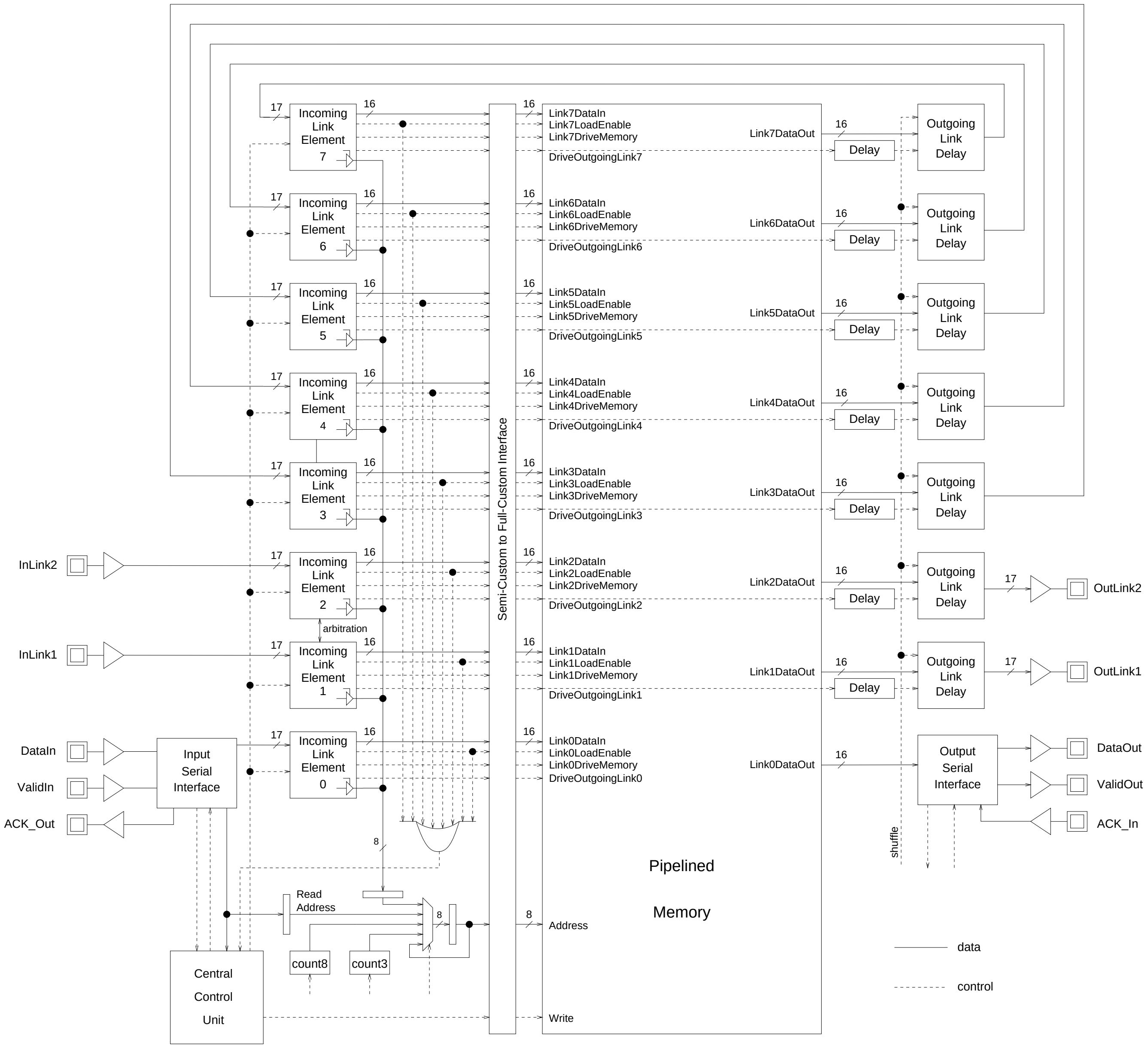


Figure 4.1: Block diagram of the test chip.

is the memory address where the packet is stored. This convention alleviates the test chip from having to keep track of the empty memory locations or the support chips from providing the address, and allows more flexible control of the pipelined memory. For example, we can write twice at a memory location to test for potential write faults, which would have been impossible, or at least much more complicated if there were an on-chip memory control unit providing free addresses for each packet write.

Command	Op-code	Arguments
Write packet	01100000	packet data (256 bits)
Read packet	00100000	address (8 bits)
Cut-through	01000000	packet data (256 bits)
Read&transmit	00000000	*none*
Receive&transmit	10100000	*none*
Receive&store	10000000	*none*
Selftest	11000000	*none*
No-op	11100000	*none*

Table 4.1: Test chip commands.

The test chip is controlled by executing simple commands received from its input serial interface. The first 8 bits of each bit-stream received form a command code and the following bits are regarded either as its arguments or as another command, depending on the command's semantics. During a command execution, the input serial interface ignores its input, with the exception of two commands as will be stated below.

The test chip recognizes the following commands, which are also summarized in table 4.1 for convenience:

Write packet This command writes a packet into the pipelined memory. The 256 bits following the command code are the packet data. As stated earlier, the first byte of the packet is the memory address where the packet will be stored. Immediately after all of the packet data is received in the serial input interface, it will be passed, at the rate of 16 bits per cycle, to the incoming link 0 of the pipelined memory. Half a clock cycle after each packet word is stored in an incoming link latch, it is written to the shared buffer. The test chip will not accept a new command before the write is complete.

Read packet Reads a packet from the pipelined memory. It takes an 8-bit argument which is the address of the packet to be read. The packet is read through outgoing link 0 and is stored in the output serial interface, at the link's rate of 16 bits per cycle. It is then scanned out, one bit at a time, until all 256 bits of the packets are out. Only after all of the packet has been scanned out does the test chip accept a new command.

Cut-through Writes and simultaneously reads a packet, using the simultaneous write and read capability of the memory banks. This command takes the same argument as write, and uses the 0 incoming and outgoing links. The serial input interface accepts a new command after the packet has been send through the output interface.

Read&transmit This command sequentially reads all of the pipelined memory starting from address 0, and transmits them through outgoing links 1 and 2 that are connected to chip pins. It initiates the transmission of two packets in two consecutive clock cycles, then pauses for some cycles to allow the packets to leave the chip, and repeats the operation until all of the buffer memory is scanned. During this command's execution the other links are idle and the serial input interface ignores its input.

Receive&transmit This command puts the test chip in a state where it waits for packets to arrive from incoming links 1 and 2 and then transmits them through outgoing links 1 and 2, respectively. Whenever a new packet arrives, a memory write is initiated for it and in the next cycle a read and a transmission are initiated through the outgoing link corresponding to the incoming link from where the packet arrived. Arbitration circuits resolve simultaneous arrivals or coincidence of an arrival with the initiation of a transmission. If new data is received by the input serial interface, it is treated as a new command and the chip exits from the receive&transmit state, terminating the endless loop.

Receive&store This command is similar to receive&transmit, but with this command the received packets are not transmitted. Arbitration is again used to resolve coincident arrivals. This command also ends when new data is received by the serial interface.

SelfTest This is the hardest test because it uses the pipelined memory for many cycles and mixes read and write accesses. It assumes that outgoing links 1 and 2 are connected to their corresponding incoming links. It initiates 7 reads/transmissions, one every two clock cycles, using links 1 through 7, from addresses 0,1,126,127,128,129,254[2] respectively. The transmitted packets are regarded as new packets at the incoming links, with their first byte treated as their address. As these packets are received, they are stored and then immediately retransmitted. Since there is one free cycle between two consecutive packet arrivals at this phase of the command, this read initiation immediately after the packet write will not coincide with another memory access. When these packets re-appear at the inputs, they are stored again in the shared buffer and their journey finally ends. The packets can later be examined using **read packet** commands.

In order for the packets to use different memory locations each time they are received, we change the order of the first words of each packet every time the packets are read out of the memory[3]. This change of address, each time a packet is received, is essential because else we would not be able to know if the packet was actually moved in the way **selfTest** intended or if the command failed somehow, but the packet is read correctly at the address where it was stored before **selfTest**.

Noop This command, as its name signifies, does nothing. It is intended to stop **receive&transmit** or **receive&store** without initiating another command.

---

[2] These addresses were selected as they use the top, middle and bottom words of the memory array.

[3] The details of this rearrangement of the packet words will be given in the implementation section.

Write packet and **read packet** test the memory of the shared buffer and they correspond to the write packet and read packet operations of the previous section. These commands cannot test the links at high speeds, because between two consecutive command executions the long time of a command reception intervenes. All of the other commands, except **noop**, test the links at high speed by issuing many packet reads and writes without the delay of receiving a new command, decoding it and then executing it, between subsequent operations.

A small network can be constructed using three or more test chips and packets can be transmitted and received between them using the **read&transmit**, **receive&transmit** and **receive&store** commands.

4.2.1 Arbitration Between the Incoming Links

When the links are connected in such a way that outgoing links drive their respective incoming links, no arbitration for memory access between the (incoming) links is required: as stated in chapter 2, only one link can start transmitting at a time, so, for this link configuration, only one packet will arrive at any time. In addition, when **write packet**, **read packet** and **cut-through** commands are executed, only the links number 0 are used, so again arbitration is not needed. The only case where link arbitration is needed is when the test chip is in the **receive&transmit** or the **receive&store** mode, and in this case the arbitration is required only between links 1 and 2, as the others are idle during these commands. Conflicts between these links are resolved in favor of link 2. A interesting case may arise during **receive&transmit**. In this chip mode, one cycle after a packet's reception, it must be transmitted again. If at this cycle another packet arrives, it is delayed by the arbitration circuits, even if it comes from the high priority link 2.

4.3 Implementation

The control and interface circuits of the test chip are built using the standard cells provided by ES2 for their 1 micron technology. Since these circuits have to function at the 16ns (for the worst case) clock period of the pipelined memory circuits, which is enough for only 3 to 5 stages of combinational standard cells, a lot of pipeline stages and thus many registers were used for the implementation of the interface and control circuits.

Having both full-custom and standard-cell circuits on one chip poses the problem of clock synchronization between them. The delay through the clock tree of the full-custom block cannot be matched with that through the semi-custom clock tree, because the size of the standard cell inverters is fixed and the routing of the cells' inputs and outputs is not fully controlled. For these reasons, we decided to use separate clock inputs for the two parts of the chip and to have leaf clock signals of the blocks drive chip pins, so that an external circuit can observe their phase difference and make the appropriate adjustments to eliminate it.

Figure 4.2 is a picture of the test chip. The pipelined memory is the large block in the

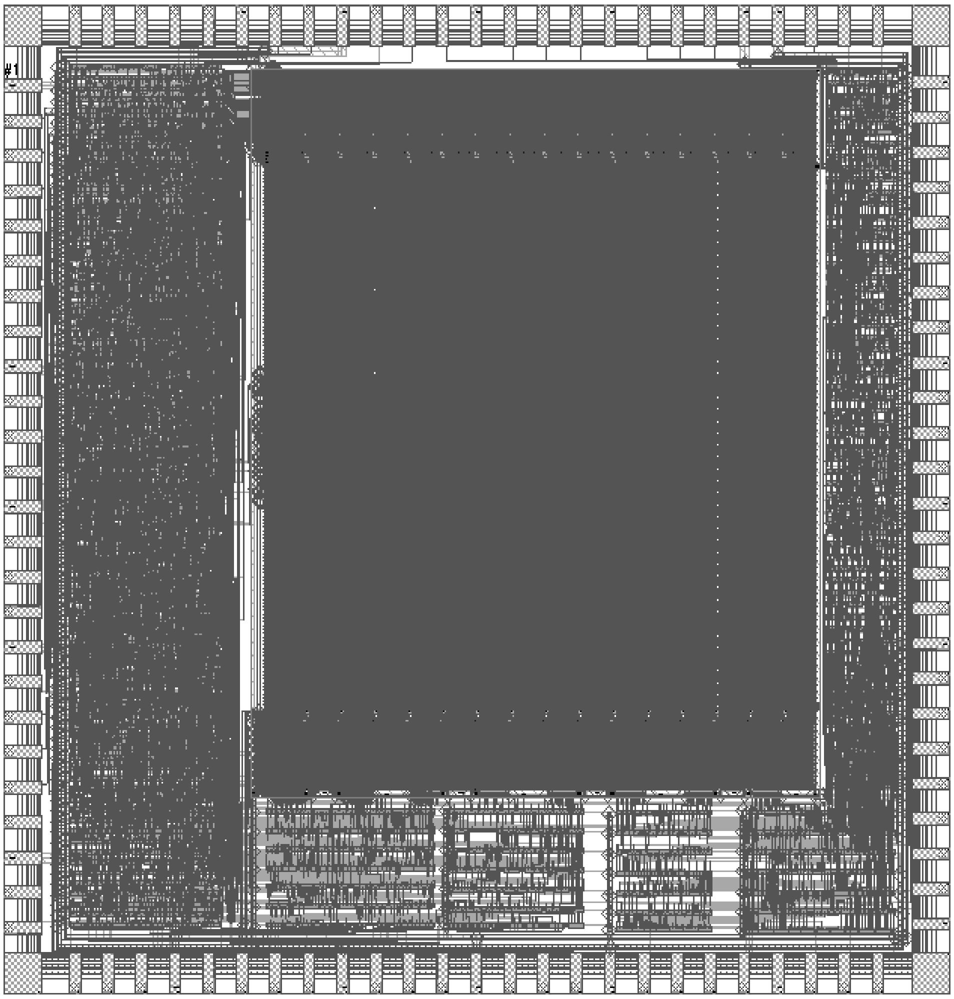


Figure 4.2: The test chip.

middle and top of the chip. The output serial interface circuits are on the right side, together with the outgoing link delay circuits for links 1 and 2. On the left, the input serial interface and the incoming link elements are placed, along with most of the outgoing link delay circuits of links 3 to 7. The area below the pipelined memory is taken up by the central control unit and the rest of the outgoing link delay circuits.

The chip is placed in a 120-pin PGA package of which 100 pins are used and the rest are left unconnected. The location of the chip pins on the package is given in table 4.2.

Pad	Pin	Signal	Pad	Pin	Signal	Pad	Pin	Signal
1	A1	UNCONNECTED	41	L6	inLnk2<14>	81	E12	UNCONNECTED
2	C2	5V	42	M6	UNCONNECTED	82	D13	outLnk1<15>
3	D3	inLnk1<9>	43	N6	inLnk2<15>	83	E11	outLnk1<14>
4	B1	inLnk1<10>	44	N7	fcClkOut	84	D12	outLnk1<13>
5	C1	inLnk1<11>	45	L7	UNCONNECTED	85	C13	UNCONNECTED
6	D2	inLnk1<12>	46	M7	5V	86	C12	outLnk1<12>
7	E3	UNCONNECTED	47	N8	semiClkOut	87	D11	outLnk1<11>
8	D1	inLnk1<13>	48	M8	reset	88	B13	outLnk1<10>
9	E2	inLnk1<14>	49	L8	semiClockIn	89	B12	5V
10	E1	inLnk1<15>	50	N9	0V	90	C11	outLnk1<9>
11	F3	UNCONNECTED	51	M9	outLnk2<15>	91	A13	outLnk1<8>
12	F2	0V	52	N10	UNCONNECTED	92	B11	outLnk1<7>
13	F1	pckHeadIn1	53	L9	outLnk2<14>	93	C10	5V
14	G1	fullcClock	54	M10	outLnk2<13>	94	A12	0V
15	G3	serDataIn	55	N11	outLnk2<12>	95	A11	UNCONNECTED
16	G2	5V	56	M11	5V	96	B10	outLnk1<6>
17	H1	ACKout	57	L10	outLnk2<11>	97	C9	outLnk1<5>
18	H2	serValidIn	58	N12	UNCONNECTED	98	A10	outLnk1<4>
19	H3	pckHeadIn2	59	M12	outLnk2<10>	99	B9	5V
20	J1	5V	60	L11	outLnk2<9>	100	A9	outLnk1<3>
21	J2	UNCONNECTED	61	N13	0V	101	C8	UNCONNECTED
22	K1	inLnk2<0>	62	L12	outLnk2<8>	102	B8	outLnk1<2>
23	J3	inLnk2<1>	63	K11	outLnk2<7>	103	A8	outLnk1<1>
24	K2	UNCONNECTED	64	M13	UNCONNECTED	104	A7	UNCONNECTED
25	L1	inLnk2<2>	65	L13	outLnk2<6>	105	C7	0V
26	L2	inLnk2<3>	66	K12	outLnk2<5>	106	B7	outLnk1<0>
27	K3	inLnk2<4>	67	J11	outLnk2<4>	107	A6	inLnk1<0>
28	M1	0V	68	K13	outLnk2<3>	108	B6	inLnk1<1>
29	M2	inLnk2<5>	69	J12	outLnk2<2>	109	C6	0V
30	L3	inLnk2<6>	70	J13	0V	110	A5	inLnk1<2>
31	N1	inLnk2<7>	71	H11	outLnk2<1>	111	B5	UNCONNECTED
32	M3	inLnk2<8>	72	H12	UNCONNECTED	112	A4	inLnk1<3>
33	L4	inLnk2<9>	73	H13	outLnk2<0>	113	C5	5V
34	N2	UNCONNECTED	74	G13	pckHeadOut2	114	B4	inLnk1<4>
35	N3	5V	75	G11	5V	115	A3	inLnk1<5>
36	M4	inLnk2<10>	76	G12	serDataOut	116	B3	inLnk1<6>
37	L5	inLnk2<11>	77	F13	AckIn	117	C4	UNCONNECTED
38	N4	inLnk2<12>	78	F12	serValidOut	118	A2	inLnk1<7>
39	M5	inLnk2<13>	79	F11	0V	119	B2	inLnk1<8>
40	N5	0V	80	E13	pckHeadOut1			

Table 4.2: Test chip package connections.

4.3.1 Serial Interfaces

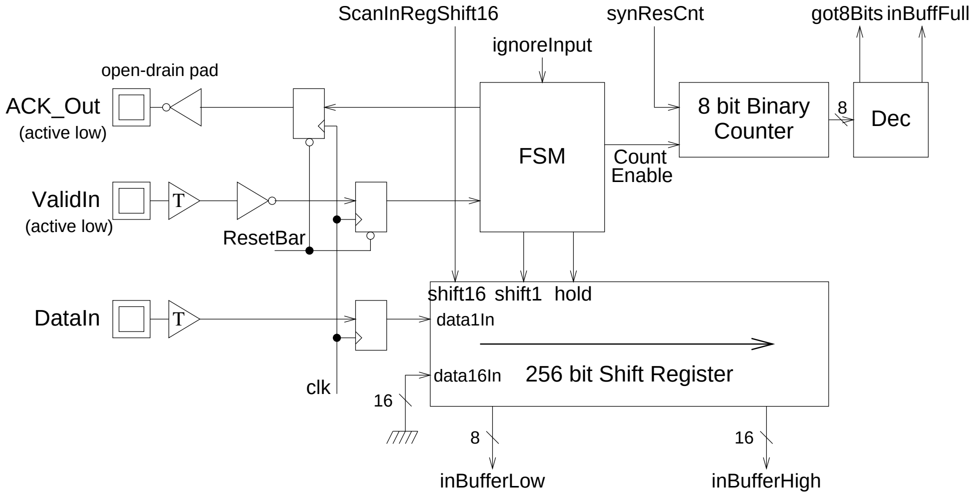


Figure 4.3: Serial Input Interface.

The two serial interfaces are similar to each other, so they are described together. Their block diagrams are shown in figures 4.3 and 4.4, for the input and output interfaces, respectively.

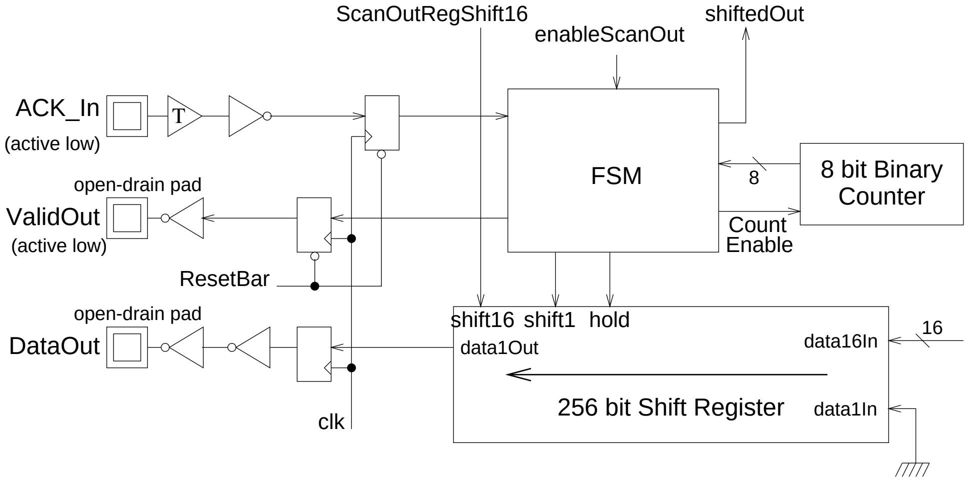


Figure 4.4: Serial Output Interface.

The central part of each interface is a 256 bit (equal to the packet size) register, which can shift either by 16 bits or by 1 bit, in the same direction. Its function is to convert one-bit serial streams to sixteen-bit serial streams and vice versa. For the input interface the shift by one is used for receiving a packet from the external circuits and the 16 bit shift is used to pass the packet to incoming link 0. For the output interface this register shifts by one when the packet is transmitted to the external world and shifts by 16 when a packet from the outgoing link 0 is read out of the pipelined memory. These shift registers are controlled by both the serial interface control FSM and the central control unit. The 8 least significant bits of the input interface's shift register, **InbufferLow**, are made available outside the interface block, because they contain the command code when a command is received and the packet address for the read command.

We implemented these shift registers in our standard-cell technology using 4 bit register cells, which are slightly smaller than 3 single D flip/flop cells of the same technology, and we implemented each register's multiplexor using three 4-ply tri-state drivers, instead of four 4-to-1 (using actually only three of their inputs) multiplexors, because the latter are much slower. The two shift registers are quite large, as they consume about 20% of the total test chip area.

We now describe the bit-serial communication protocol, the signals, and their timing, from the point of view of the input interface. The output interface signals and timing are the same as those of the sender that communicates with the input interface. When a new bit is transmitted, the **ValidIn** signal goes low and the **DataIn** signal carries the data. The input interface stores the data and responds with an **ACK_Out** signal, which is active low. The sender "sees" the acknowledgment and withdraws (makes high) its **ValidIn** signal. The receiver then withdraws the **ACK_Out** and is ready to receive the next bit. On the other side the sender waits for the **ACK_Out** signal to go high and then initiates the transmission of the next bit, if it has more data to send.

Simple FSMs which execute the asynchronous handshake protocol for bit-serial communication with the external world govern the serial interfaces. The central control unit enables or disables the reception or the transmission, according to the chip's current state.

An 8-bit binary counter is included in each interface, to count the number of bits received or transmitted. For the output interface its sole use is to determine when the packet transmission has finished. The input interface counter, besides detecting when a packet is completely received, notifies the central control unit (CCU) when the first 8 bits of a bit-stream, which correspond to a command, have been received so that the CCU can decode it and decide whether to accept more input from the serial interface, or to stop receiving and execute the command. The counter is then reset by the CCU, to become ready to count either the data bits following the command code, or the next bit-stream. When the command is followed by data, the previously reset counter will signal again the CCU at the reception of the first 8 bits of data, but at this time the CCU will be in a different state, executing a command rather than expecting one, and it will treat this signal appropriately.

4.3.2 Incoming Link Elements

The incoming link elements extract the addresses of the incoming packets, generate control signals for the incoming links and, in some cases, for the outgoing links too. In addition, the incoming link elements of links 1 and 2 also contain the arbitration circuits. Specifically, when a new packet enters an incoming link element, its most significant byte of the first packet word is copied into an internal register as the address where this packet will be stored, and the incoming link control signals, load enable and memory input drive enable, are generated. When the **selfTest** command is executed, the outgoing link drive signal is also generated.

As mentioned earlier, the only links from where packets can arrive simultaneously are links 1 and 2 and this can happen only when **receive&transmit** or **receive&store** are executed. If a simultaneous arrival occurs, priority is given to the packet from link 2. In the case of **receive&transmit**, where one cycle after a packet is stored, it is read out and transmitted, if the packets from links 1 and 2 arrive with one cycle delay between them, then the write of the second packet, even if it comes from the higher priority link 2, is delayed one cycle so that the other packet's read will be initiated immediately after its write.

4.3.3 Outgoing Link Delay Circuits

These circuits are included to delay the transmitted packets for 8 cycles, before they are fed back into the incoming links. This delay is essential for the **selfTest** command, when the packets are read out, received back and then immediately retransmitted. Without this delay, the second transmission of the packets would start before their first transmission had completed, which means that the outgoing links would be driven simultaneously by two memory stages of the pipelined memory.

These circuits, also perform the re-ordering of the packet words, also required by **selfTest**, so that the packets are stored in different locations each time they are received back in the test chip. When the control signal **shuffle** is asserted, the first word of the packet becomes fourth and the second, third, and fourth words move one position ahead. This happens at every packet transmission during the **selfTest** command, so when the command finishes, the packet words have the following order: the third and fourth words are now first and second, and the first and second are now third and fourth respectively.

4.3.4 Semi-Custom to Full-Custom Interface

The function of this block is to prepare the inputs of the pipelined memory so that they are fed to the full-custom block with the correct timing. For example, when a packet is received, the address, the control signals and its data, enter the semi-custom to full-custom interface at the edge of the same clock cycle. But the pipelined memory expects these signals at different phases of the clock, or even at different clock cycles, as is the case for the first word of a packet and the memory address and write signals; this is done by the semi-custom to full-custom interface.

4.3.5 Central Control Unit

The Central Control Unit (CCU) is where the test chip decodes and executes the test commands. In the CCU's FSM for most of the commands there are states whose sole purpose is to count a number of clock cycles to determine the finish of an operation in the pipeline memory. In order to reduce the number of such states in the FSM, we included a sequence counter in the CCU that is used for counting these cycles for all commands.

The state encoding of the FSM follows the "one-hot" style, that is, the bit length of the state representation is equal to the number of states in the FSM. This encoding proved very helpful, because in order to determine the machine's state only one bit in the state vector needs to be tested, reducing the number of combinational circuit stages needed to generate time critical control signals. The sequence counter, also follows the "one-hot" encoding style, greatly simplifying its implementation to that of a simple shift register.

The CCU and the other circuits are initialized by means of a global reset signal, that is connected to one of the chip input pins.

4.4 Design Flow and Design Environment

Since the pipelined memory does not have complicated timing or control circuits, we did not write a behavioural model of it before starting to design its circuits. We designed the circuits on paper and simulated them with the HSPICE circuit simulator to validate their operation and to estimate the sizes of the transistors. Then, we laid out the circuits using the Virtuoso Layout Editor of Cadence Design Framework II, using the technology parameters from the ES2 design kit. Each circuit was extracted using the Diva extraction and verification tool of the Cadence framework and simulated again, with the parasitic capacitors included this time. In order to include the coupling effects in our simulations, the parasitic capacitors formed between adjacent wires were also extracted at that time. Finally the whole pipelined memory block was assembled, its input and output terminals were defined and an abstract representation of it was generated by the Abstract Editor Cadence tool to be used by the place and route tools.

It must be noted that the design flow just described is not fully automatic and many actions had to be done manually. For example, the Diva technology file included in the ES2 design kit does not extract the area of drain, and the perimeter of drain of the transistors, but instead it extracted them as parameters of diodes. On the other hand, the SPICE models of the design kit did not contain the diode models. As a result, in order to include the effects (mostly the capacitance) of the diffusion areas in the simulation, we had to manually edit the extracted spice file and convert the diode parameters to the appropriate transistor parameters.

Another problem that we encountered when we tried to extract the full-custom block and simulate it together with the semi-custom circuits was that the complete full-custom block was too big to be extracted as a flat design and no hierarchical extraction was supported by the ES2 design kit. Thus, we wrote a behavioural model of the full-custom part, which we tried to build as close to the actual circuit as possible, and used this model for our mixed semi-custom/full-custom simulations.

The design environment posed fewer difficulties during the design of the standard-cell circuits, mostly due to the use of the Synopsys synthesis tool, which is very robust and easy to use. We modeled these circuits in the Verilog hardware definition language (HDL) following the Synopsys conventions, and simulated them extensively using the Verilog-XL simulator, which is part of the Cadence framework, to validate their correctness. We then used the Synopsys Design Compiler to translate and optimize the high-level Verilog code into a gate-level netlist using cells from the ES2 standard cell library. Next, the gate-level netlist was fed to the Cadence framework for the placement and routing phases.

The place and route tools perform the placement of the standard cells in the chip and route the wires (called nets) that connect the cells, using a rough floorplan and the net information provided by the designer. This floorplanning involves placing manually the large circuit blocks like the pipelined memory and the I/O pads, defining placement regions for standard cells and the number of cell rows in them, partitioning the cells into groups with high intra-group and low inter-group connectivity and assigning groups to the placement regions. The net information consist of a priority number and the width of each wire. The net priority number is very important as it is the parameter that determines the exact placement of the standard cells in their assigned region. Unfortunately the number of nets is huge and it is very easy to forget to assign priorities to some important nets, especially when they are not labeled with a name, as is the case with intermediate nodes of combinational circuits. One such mistake can lead to a one or two weeks of delay, because it can only be discovered after simulating the completely placed and routed chip using the extracted parasitic capacitances of the nets.

We used the Cell Ensemble Cadence tool (the one supported by the ES2 design kit) to perform the actual placement of the cells using the floorplan and the net priorities that we defined. The placement is followed by two routing phases, called global routing and detailed routing. The global routing phase decides which channels, that is, free areas between the standard cell rows, will be used to route each net, and detailed routing draws the actual layout of the nets. In our design, the global router did not perform satisfactory, so all of the clock nets and most of the critical nets were manually global routed. Moreover, the detail router introduced some design rule violation (called DRC errors), of which some were corrected manually using the layout editor, and the others using a Skill (the Cadence Design Framework's programming language) program that we developed[4].

The parasitic capacitances of the nets were extracted from the final routed chip and back-annotated simulation was performed to validate the circuit's operation and to reveal net overloading problems, that is, nets with more capacitance than that handled by their drivers. Such overloading problems can be corrected either by raising the priority of the net or, for severe overloading, by inserting an inverter/buffer to drive the net. Following these changes the chip has

---

[4] This program is available through the author's WWW home page: http://www.ics.forth.gr/ efthym.

to be placed and routed again. Due to our careful design, we did not encounter any overloading problems at that stage.

Finally, the placed and routed chip was converted to the GDSII format and submitted for fabrication through EUROCHIP.

Chapter 5

Post Fabrication Testing

In this chapter we present the testing environment and the test results.

5.1 Testing Environment

The testing environment (figure 5.1) consists of a a simple matrix board containing the test chip, a general purpose, programmable, prototype PCB-board containing an intel 80C32 microcontroller, and a personal computer (PC). The emulator module present in the figure contains the BIOS of the microcontroller, which is downloaded from the PC through an interface card on the I/O bus of the PC. These BIOS routines are written in C and implement the asynchronous protocol for communicating with the test chip serial interfaces, and the routines for receiving and transmitting through the RS-232 serial port of the PC.

The PC is used as a simple terminal running the Procomm Plus software. Whatever is typed to the keyboard is transferred to the microcontroller through the RS-232 interface, echoed back to the PC's monitor, and then sent to the test chip through its serial input interface and according to its protocol. On the other direction, when data is transmitted by the serial output interface of the test chip, the microcontroller is interrupted and when a full byte is assembled, the data is sent to the PC through the RS-232 interface. Since the chip is designed to respond to commands, it never transmits data while it is receiving a command, which simplifies the microcontroller's task.

As stated in the previous chapter, the two parts of the test chip, the semi-custom interface and control circuits, and the full-custom pipelined memory use separate clock distribution trees and thus separate clock input pins. Since these clocks must be synchronized, two leaf level signals of these clocks are also output for external observation and correction of their phase difference. The delay line on the test board serves the correction purpose. One of the clock inputs is directly connected to the oscillator, while the other is connected to an output of the delay line chip through jumper switches. The particular delay line chips that we used have 10 output pins, each carrying a one to ten nanoseconds delayed version of the input signal. The two

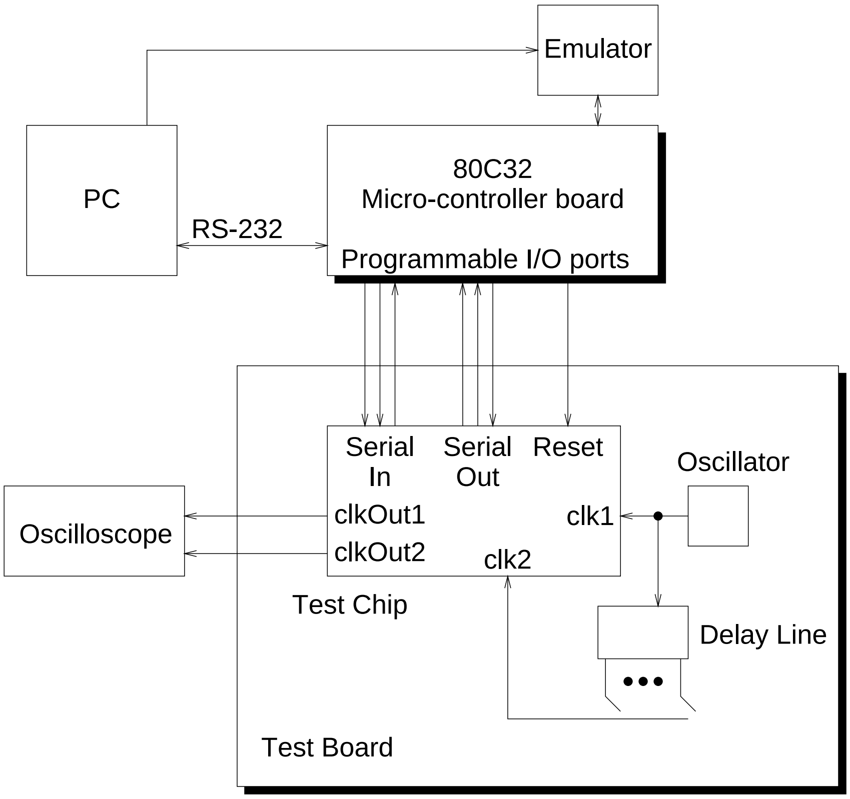


Figure 5.1: Testing Environment.

clock outputs are displayed on an oscilloscope, so that their phase difference can be measured, and the appropriate jumper can be set to minimize this difference.

We encountered noise problems on the clock wires due to the simple matrix board where the test chip, the oscillator, and the delay line were placed. We found that the problems were much less severe when we removed the delay line and connected both of the clock input pins directly to the oscillator. The clock skew between the two output clock signals is in this case 0.8ns. Thus, our prototype board does not currently use the delay line.

The testing is simply performed by typing the code number of a test chip command, followed by the address for reads or by the packet data for write and cut-through, and observing the data that is received back. Everything that is displayed on the monitor can also be logged in a file and, in addition, a file can be directed to the serial interface as if its contents were typed to the keyboard. We automated the test procedure by using these features. We wrote the test commands in a file, and produced another containing the expected results. For each chip under test, we feed it with the command file and log the results in a new file. Then this log file is compared to the one with the expected results and the chip is characterized as faulty or not.

5.2 Test Results

The testing revealed two design errors in the full-custom part of the chip. In three out of the four predecoder circuit instances, the output of a nand gate is shorted to the ground line. As a result, the decoder behaves as if the six most significant bits of the address are all zeroes. This design error occurred when the full-custom design was completed and some minor changes to the widths of ground and supply wires were performed, on the topmost cell of the full-custom design. This is why the short is found only in three and not in all of the predecoder cell instances. Since the decoder is too large to be extracted from its layout and then simulated (see section 4.4), we did not discover this error.

The other design error is found in the clock buffer area below the decoder. The two clock signals that connect to the latches of the control signals of the outgoing links have the opposite polarity of the intended one. This clock buffer cell has minor differences compared to those of the other stages, and was never used in simulations. Although this is a serious error, the control signals of the outgoing links operate correctly for the other stages. This is because the control signal latches and drivers of these stages assume that their input is qualified with the correct clock phase, and thus they are slightly different from the classic tri-state latches (see figure 3.21). Although the control signal output of the first stage is not correctly qualified, it passes the first “tri-state” gate of the next latch and also the second, since it is asserted in the phase when this gate is transparent. Thus the signal gets qualified with the correct clock signal starting from the second stage. As a result of this error, the first 16 bits of the data are always read as zero, but unfortunately the data of the second stage are affected too. The data wires do not have enough time to be fully precharged, and therefore the data that is seen on the link wires at the second stage depend on the data output of the first stage, the coupling between the data wires (see section3.3.2), and the true data output of the second stage. The coupling of the data wires is a problem in this case, because the Schmitt trigger does not expect that the data wires are not fully precharged. Also, because of this error, the clock output pin of the full-custom circuitry has the inverse polarity than the semi-custom clock output.

It must be noted again, that these errors, which we believe can happen or have happened to many designers, are easily identified by extracting the circuit from the layout and then simulating. Unfortunately, the combination of Cadence Design Framework II and the ES2 Design Kit that we used was unable to extract large designs and consequently these mistakes have passed unattended. Moreover, the errors are found in the cells that were designed last, under time pressure.

Besides these errors, the test chip was tested and found operational for clock frequencies up to 80MHz. The power consumption at this frequency is 2.75 Watts. This frequency corresponds to the minimum clock cycle of the semi-custom circuits when simulated for typical operating conditions. We did not test it at higher frequencies because we do not presently have a higher frequency oscillator.

Most of the semi-custom part is easily tested by observing if the commands that are given are received by the test chip, and if the chip responds to commands such as **read** and **cut-through**. Thus a command, that passes data from the serial input to the serial output of the test chip, without using any of the full-custom circuits, so as to test the semi-custom part only, is not necessary and thus was not included. The commands that were tested are **write**, **read**, **cut-through**, **noop**, and, partially, **read&transmit**. We could not fully test **read&transmit** and the other commands, as the the first byte of each packet that is regarded by convention as its address would always be zero. These is also some difficulty in using the **selfTest** command since we can only use the addresses from 0 to 4, while **selfTest** accesses packets in a wide range of addresses.

Concerning the yield, of the 30 sample chips that we received from EUROCHIP, 6 had a short circuit, three did not respond to any of the test commands, while the rest were operational, at least to the commands that we could issue. Thus the yield is 70%.

Chapter 6

Extensions and Conclusions

In this chapter we present some extensions to the pipelined memory and our conclusions.

6.1 Extensions

As some of the parameters of the pipelined memory change, other VLSI organizations may achieve better results than the one we implemented. For example, if the packet size were doubled, our pipelined memory would be too wide to be placed in a single switch chip, although its total area would not be very large. The organizations presented in chapter 3 could be employed if more packet storage, larger word size per stage, or more pipeline stages were required. Specifically, the second and fourth microarchitecture can be used when more packet storage is required, or if the word of each memory stage is larger. The third microarchitecture can prove very helpful for a large number of pipeline stages or for large packet size.

If the link bus wires become very long and thus slow to drive, an interesting extension to the first organization is to split these wires in two or more pipeline stages. At each splitting point an additional pipeline stage is also inserted into the wordlines and the link control signals. The net effect is that all packet data are delayed by an equal number of cycles on their way from an input to an output link.

Furthermore, one can also split the bitlines into multiple pipeline stages, to speed up the memory access time; the address decoder also has to be split in the same manner for this extension. Figure 6.1 presents how one of the original memory stages is extended to two vertical stages. At the bitline splitting point a combined sense amplifier and bitline driver is inserted. When a read is performed and the selected word is in the top memory block above the splitting point, this circuit would sense the voltage difference on the bitlines “above” it and use the read output to drive the bitlines of the following stage in the next cycle as if a write were performed at that stage. Since the word to be read belongs to the top memory block, the decoder of the second stage will not assert any of the wordlines and thus this write operation on the second (vertical) memory stage will only pass the result of the previous read to the bitlines

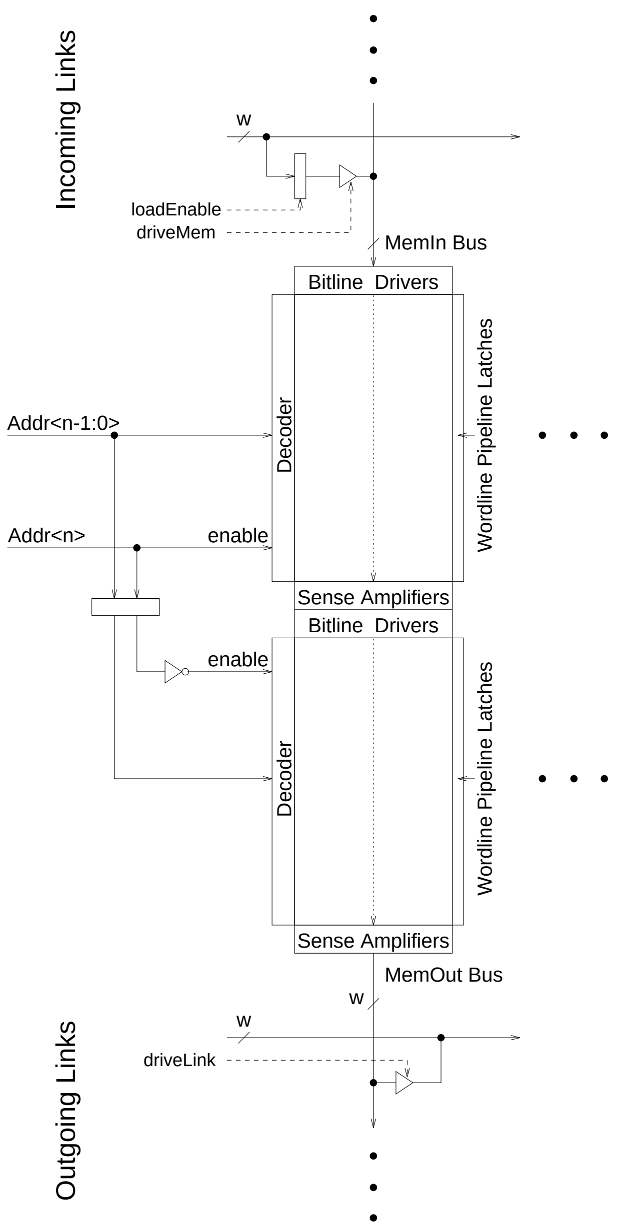


Figure 6.1: Pipelined bitlines extension.

without destroying the data of any memory word. Thus, the basic idea behind this scheme is that each vertical memory stage just passes the data given to it from above, when the access does not involve one of its words. Note that the wordline pipeline latches need no changes like those of the decoders, proving for one more time the ease of control of the pipelined memory. In order to save on power consumption, the "dummy writes" can be performed using circuits similar to the write circuits of the bitline drivers (section 3.3.1) but with smaller transistor widths, so as to decrease the voltage swing[1] of the bitlines during these “writes”.

Finally, the speed of the pipelined memory can be improved if it is designed in a BiCMOS technology, as there are many long and highly capacitive lines that can benefit from the high current capability of bipolar transistor drivers. The most appropriate candidates would be the link data lines, the clock lines, the memory input and output busses and finally the bitlines. Also the high gain of amplifiers built of bipolar transistors could be put in effect for implementing the sense amplifiers.

6.2 Conclusions

Among the switch buffer architectures, shared buffering achieves the best memory utilization and optimum link throughput, but needs to use high-throughput memories. Pipelined memory is a novel organization for building such high-throughput memories, featuring simplified control and very efficient VLSI implementation.

In this work we designed a pipelined memory with a throughput of 25 Gbits/s (16 Gbits/s in the worst case), enough for 8 incoming and 8 outgoing links at gigabit per second rates. Full-custom design techniques were employed to achieve small area and high speed. The full-custom implementation of the pipelined memory pays off the long development time, as our implementation is 4 times smaller and 3 times faster than the pipelined memory of a semi-custom switch chip that was recently designed by our group. Moreover, most of the development time of our implementation is attributed to the numerous difficulties that we faced with the design environment and not to the complexity of the circuits or of the pipelined memory organization itself.

A test chip containing the pipelined memory and semi-custom control and interface circuits was fabricated through the European Community multi-project fabrication service, EUROCHIP. This chip, although it revealed some minor design errors, operates correctly for clock frequencies of up to 80MHz, consuming 2.75 Watts.

---

[1] About 0.5 Volts are sufficient for the correct operation of the sense amplifiers.

Bibliography

[AOST93] T. Anderson, S. Owicki, J. Saxe, C. Thacker: “High-Speed Switch Scheduling for Local-Area Networks”, *ACM Trans. on Computer Systems*, vol. 11, no.4, November 1993, pp. 319-352.

[DeCS88] M. Devault, J-Y. Cochennec, M. Servel: “The “Prelude” ATD Experiment: Assesments and Future Prospects”, *IEEE Journal on Sel. Areas in Communications*, vol. 6, no. 9, December 1988, pp. 1528-1537.

[DeEI95] W. Denzel, A. Engbersen, I. Iliadis: “A Flexible Shared-Buffer Switch for ATM at Gb/s Rates”, *Computer Networks & ISDN Systems* 27 (1995), pp. 611-624, Elsevier Science B.V.

[GrHo86] T. Grotjohn, B. Hoefflinger: “Sample-Set Differential Logic (SSDL) for Complex High-Speed VLSI”, *IEEE Journal of Solid-State Circuits*, vol. 21, no. 2, April 1986, pp. 367-369.

[HlKa88] M. Hluchyj, M. Karol: “Queueing in High-Performance Packet Switching”, *IEEE Journal on Sel. Areas in Communications*, vol. 6, no. 9, December 1988, pp. 1587-1597.

[IEEE90] IEEE Standard 1149.1, *Test access port and boundary scan architecture*, 1990

[Kate94] M. Katevenis: “Telegraphos: High-Speed Communication Architecture for Parallel and Distributed Computer Systems”, Technical Report FORTH-ICS/TR-123, *ICS, FORTH, Heraklio, Crete, GR*, May 1994, 38 pages; on-line: ftp.ics.forth.gr: techreports/1994/94.TR123.Telegraphos.ps.Z

[KaSC91] M. Katevenis, S. Sidiropoulos, C. Courcoubetis: “Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip”, *IEEE Journal on Sel. Areas in Communications*, vol. 9, no. 8, October 1991, pp. 1265-1279.

[KaVE95] M. Katevenis, P. Vatsolaki, A. Efthymiou: “Pipelined Memory Shared Buffer for VLSI Switches”, *Proceedings of the ACM SIGCOMM’95 Conference*, Cambridge, Ma., USA, Aug. 1995.

[Koza91] T. Kozaki, N. Endo, Y. Sakurai, O. Matsubara, M. Mizukami, K. Asano: "32 × 32 Shared Buffer Type ATM Switch VLSI's for B-ISDN's", *IEEE Journal on Sel. Areas in Communications*, vol. 9, no. 8, October 1991, pp. 1239-1247.

[KrLL82] R. Krambeck, C. Lee, H. Law: "High Speed Compact Circuits with CMOS", *IEEE Journal of Solid-State Circuits*, vol. SC-17, June 1982, pp. 614-619.

[LaSe94] R. Lamaire, D. Serpanos: "Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues", *IEEE/ACM Trans. on Networking*, vol. 2, no. 5, October 1994, pp 471-482.

[Mirf95] N. Mirfakhraei: "Design of a CMOS Buffered Switch for a Gigabit ATM Switching Network", *IEEE Journal of Solid-State Circuits*, vol. 30, no. 1, January 1995.

[Seki93] T. Seki, E. Itoh, C. Furukawa, I. Maeno, T. Ozawa, H. Sano, N. Suzuki: "A 6-ns 1-Mb CMOS SRAM with Latched Sense Amplifier", *IEEE Journal of Solid-State Circuits*, vol. 28, no. 4, April 1993, pp. 478-483.

[Shob91] Y. Shobakate, M. Motoyama, E. Shabakate, T. Kamitake, S. Shimizu, M. Noda, K. Sakaue: "A One-Chip Scalable 8*8 ATM Switch LSI Employing Shared Buffer Architecture", *IEEE Journal on Sel. Areas in Communications*, vol. 9, no. 8, October 1991, pp. 1248-1254.

[Stun94] C. Stunkel, D. Shea, B. Abali, M. Denneau, P. Hochschild, D. Joseph, B. Nathanson, M. Tsao, R. Varker: "Architecture and Implementation of Vulcan", *Int. Parallel Processing Symposium*, April 1994.

[TaCh93] Y. Tamir, H-C. Chi: "Symmetric Crossbar Arbiters for VLSI Communication Switches", *IEEE Trans. on Parallel & Distributed Systems*, vol. 4, no. 1, January 1993, pp. 13-27.

[TaFr88] Y. Tamir, G. Frazier: "High-Performance Multi-Queue Buffers for VLSI Communication Switches", *Proc. of the 15th Int. Symp. on Computer Architecture*, ACM SIGARCH vol. 16, no. 2, May 1988, pp. 343-354.

[Turn93] J. Turner: "An Optimal Nonblocking Multicast Virtual Circuit Switch", Technical Report WUCS-93-30. *Washington Univ. St. Louis, MO, USA*, June 1993, 22 pages.

[Uyem88] John P. Uyemura: "*Fundamentals of MOS Digital Integrated Circuits*", Addison-Wesley, ISBN XXXXXXXXXX, 1988.

[WeEs93] N. Weste, K. Eshraghian: "*Principles of CMOS VLSI Design — a Systems Perspective*", second edition, Addison-Wesley, ISBN 0-201-53376-6, 1993.

[YeHa87] Y. Yeh, M. Hluchyj, A. Acampora: “The Knockout Switch: A Simple, Modular, Architecture for High-Performance Packet Switching”, *IEEE Journal on Sel. Areas in Communications*, vol. SAC-5, no. 8, October 1987, pp. 1274-1283.

[YuKS87] J-R. Yuan, I. Karlsson, C. Svensson: “A True Single-Phase-Clock Dynamic CMOS Circuit Technique”, *IEEE Journal of Solid-State Circuits*, vol. 22, no. 5, October 1987.