# HIPPI–6400 Overview

***Why HIPPI–6400?***

***What is it?***

***Hardware Components***

***Software Techniques***

**Greg Chesson**
**greg@sgi.com**

# Why HIPPI–6400?

***NEED***

**High–bandwidth, low–latency networking is critical for clusters, arrays, NOWs, and all forms of distributed computing. The need is independent of the programming model.**

**Requirements for next generation systems:**

**1 GByte/sec *reliable* transfers (=> flow+error control),**
**short message latency (1–128 byte) < 1 microsecond,**
**carry a wide range of traffic (bursty, asynch, bulk, IP, continuous).**

***STATUS***

**HIPPI doesn’t provide this level of performance, and has only flow control, no error control.**

**SCI has bandwidth, but no error control, limited flow control.**

**ATM has neither, is complex and prone to congestion (!flow control).**
**OC96c or OC192c links, switches and adaptors are unlikely.**
**ATM may not evolve to handle dynamic bursty async loads.**

**Fibre channel has both flow and error control, but not for all classes of service, plus is not quite fast enough (could be fixed) and is complex, although not in the same class as ATM.**

# HIPPI–6400 defined

1. HIPPI–6400 (SuperHIPPI) is a proposal to extend the definition of HIPPI (HIPPI–800 and HIPPI–1600) to include an additional PHY (6400).

2. Existing HIPPI frames are defined for use over the new PHY. This imples that HIPPI–to–SuperHIPPI translation through a switch or other device is straightforward: addressing and semantics are equivalent.

3. Existing 32 or 64–bit copper HIPPI links at 25 Mhz are replaced by (16+4)–bit copper or fibre bidirectional parallel links at (effective) 400 Mhz. Peak bandwidth is 2*800 MBytes/sec.

4. HIPPI–6400 uses credit–based flow control and go–back–N retransmission on a hop–by–hop basis. With proper crc coverage and interface design, this can eliminate the need for a software protocol layer (i.e. OS Bypass) whenever HIPPI–6400 is employed end–to–end.

5. Using parallel fiber, a HIPPI–6400 link could eventually extend to 1 km as technology improves. Microcoax links would be limited to the 50–75m range.

6. The new PHY defines a small number (4) of virtual channels to avoid the "destination–fax–machine–is–busy" problem of HIPPI. The VCs are assigned to cmd, response, IP, and bulk traffic.

# What is needed?

*A new LAN needs:*

| | |
|---|---|
| 1. working hardware for proof of concept. | ✓ |
| 2. standards–quality specifications. | TBD |
| 3. no legal encumbrances (e.g. patents) | ✓ |
| 4. commercial MAC device(s). | SGI, license, asic cell |
| 5. interworking with HIPPI–800 and HIPPI–1600. | ✓ |
| 6. readily available HIPPI–6400 switches | ✓ |

# What exists today?

dc–coupled links are operational today

length: designed for 3 meters, tested ok at 5 meters.

frequency: designed for 400 Mhz, cables tested at 440 Mhz.

signaling: 1 volt differential drivers.

cable/connector: 50–pair  micro–coax.

circuit technology: LSI Logic 500K or equivalent.

utility: chip–to–chip, board–to–board, box–to–nearby–box.

# Interconnect Test Fixtures

test board with two test chips

test stand with logic analyzer

# Extending the link for SuperHIPPI

1. code the bit lines using a 3/4 code (simple, small, fast).

2. employ ac–coupling instead of dc–coupling.

3. define both copper and fiber transmission.

4. choose fiber format:

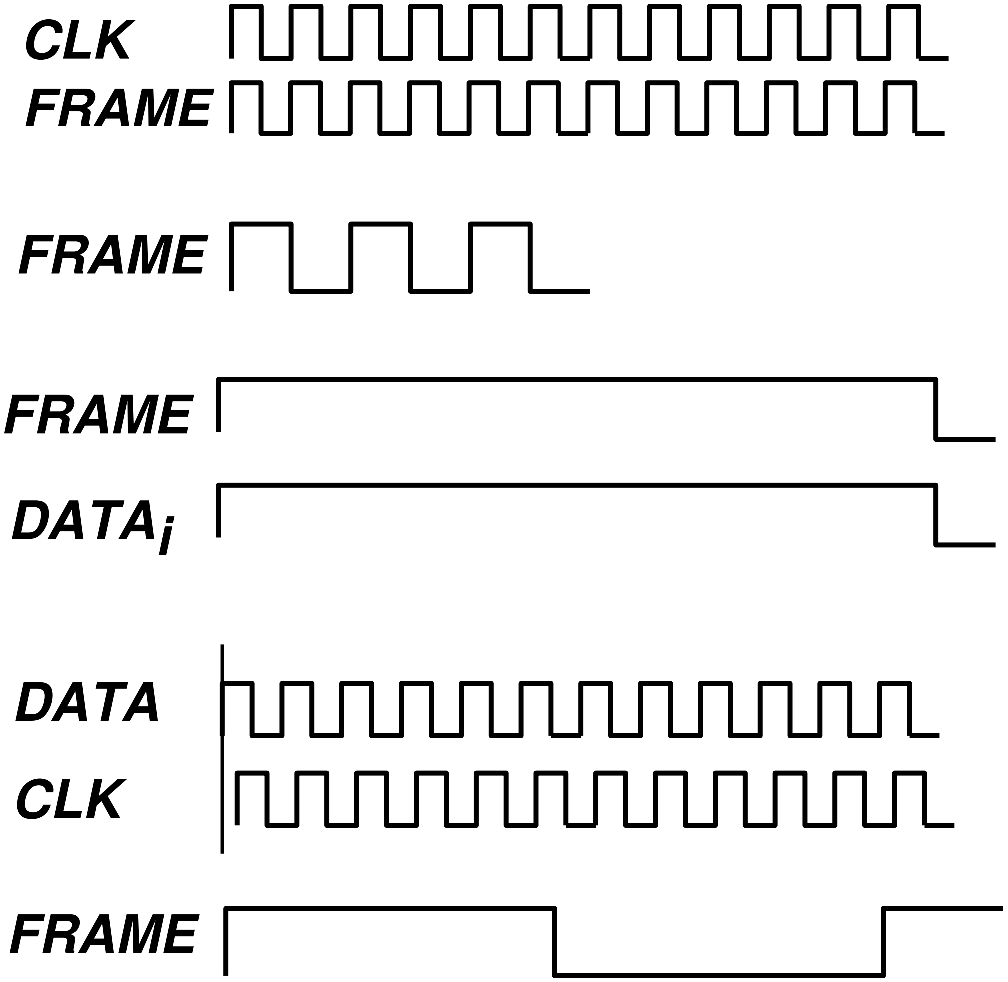
   use commercial/telco standard 12–channel format.
   link becomes 24 channels at 533 Mhz, or
   12–channels at 1066 Mhz.

5. choose maximum link distance (1 km).
   note: with (expensive) low–skew fiber (1ps/m) the skew buildup over 1km could be 1 ns which is a reasonable limit for skew compensation circuitry. Using available fiber and lowest–cost components, a HIPPI–6400 fiber link would probably be limited to 100m in 1998.

6. add skew compensation circuit on bit lines.

7. signal at 3v instead of 1v (for 3X greater distance in microcoax).

# Clocking and Initialization




1. transmit CLK on FRAME
   1 sec to charge cable
   align FRAME to CLK edge

2. acknowledge step 1
   proceed to alignment

3. align each DATA bit to
   the FRAME bit

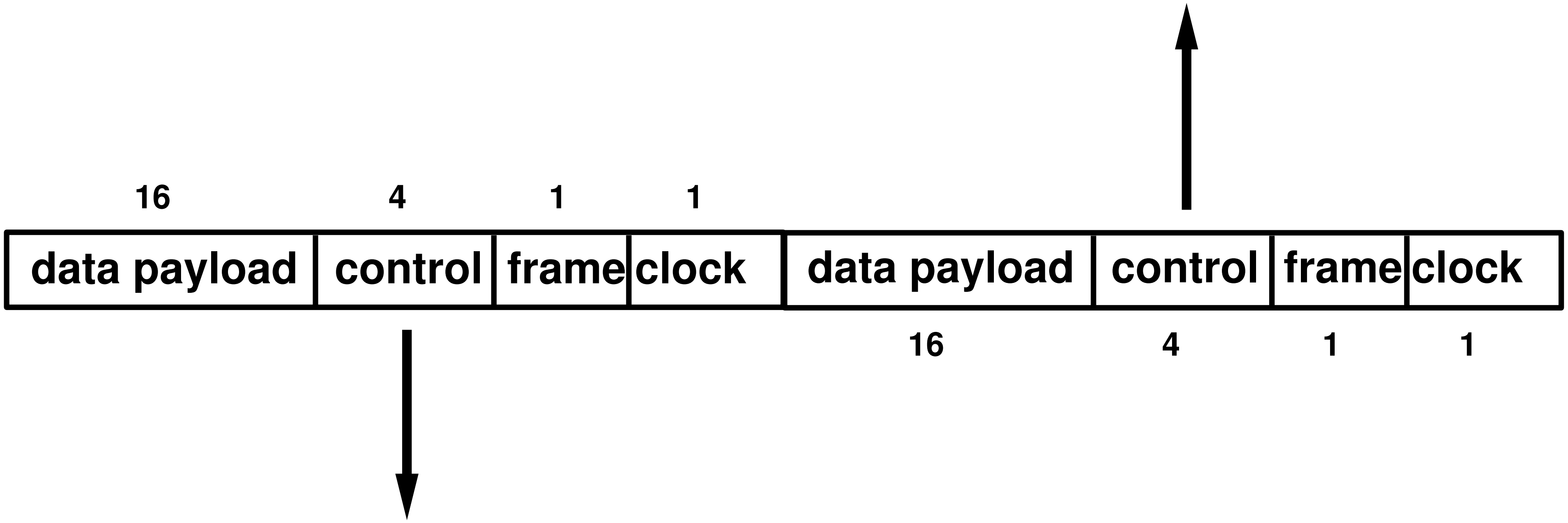
4. center CLK on DATA

5. normal FRAME pattern
   link is up

6. drift adjustments are
   made during IDLE packets

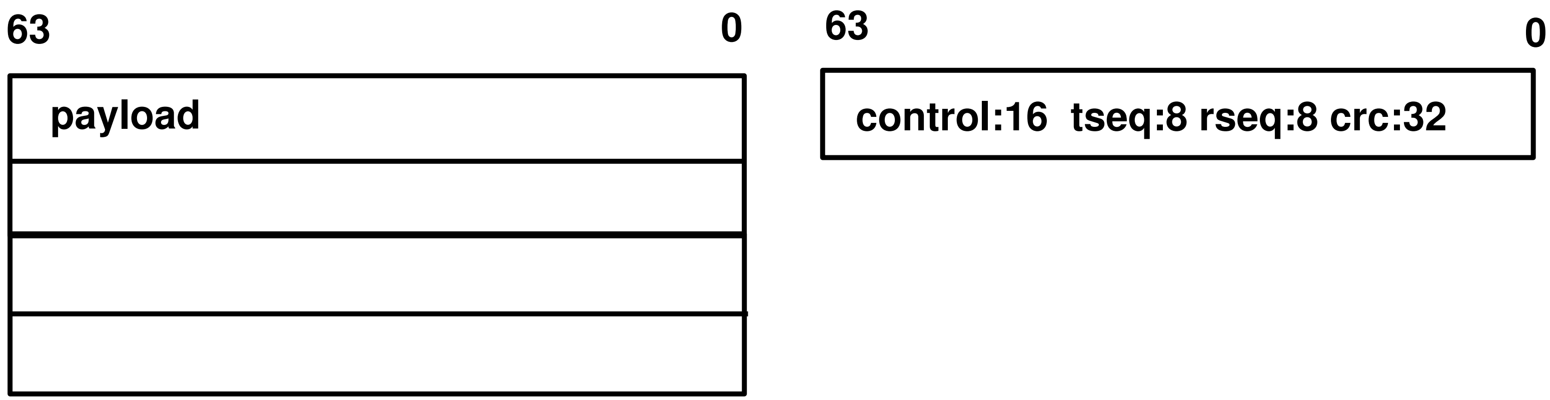
State progression is encoded by the FRAME signal.

# Signal Design




Source synchronous clocks in each direction.
Both clock edges are used to latch data.
Differential clock receiver generates complementary clocks.
There are 20 data bits (16+4) plus one frame bit.
Control bits are in parallel for lower latency.
The frame bit denotes start–of–micropacket.

# Micropacket Design




16 payload + 4 control bits per clock

fixed size micropackets:
16 clock ticks == 1 micropacket == 32 payload bytes == 40 ns
== 8 control bytes

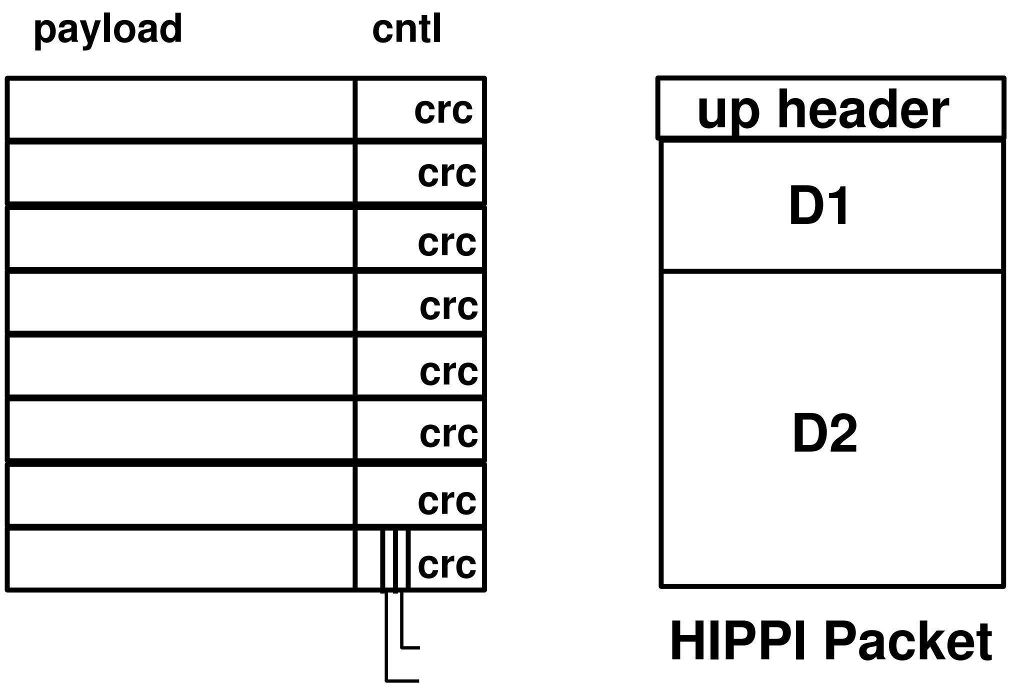
micropackets are acked and retransmitted on nak or timeout

32 µpackets == 1 KByte == 1.28 µseconds == 256 meters (fiber @ 5 ns/m)

∆ 8–bit sequence and 8 KB buffers cover 1 km of delay*bandwidth

# SuperHIPPI Packet

an arbitrary length sequence of micropackets




HIPPI Packet

32–bit crc is accumulated over entire message.
Partial sums are sent with each micropacket.
Flow control and retransmission are on micropackets.
M icropackets for different VCs may be interleaved on the link.
Final micropacket indicated by TAIL bit.  CRC error indicated by ERROR.
Entire packet need not be wholly contained in any transit buffer.

# SuperHIPPI control bits

*sent with every micropacket*

| *tseq* | *rseq* | *crc* | *vc* | *cr* | *vcr* | *type* | *error* | *tail* | *total* | *reserved* |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 8 | 32 | 2 | 6 | 2 | 4 | 1 | 1 | 64 | 0 |

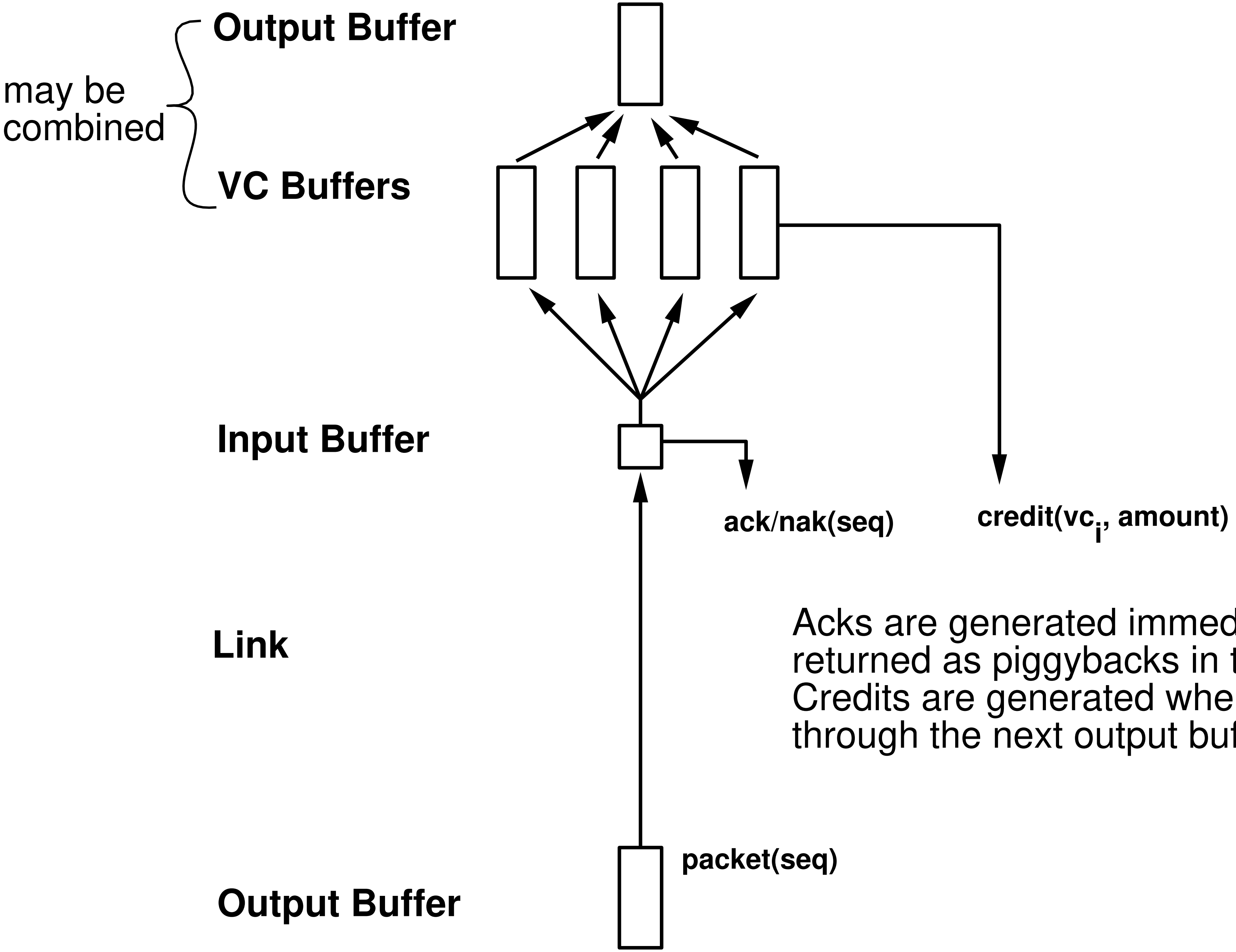
vc = virtual channel of this μpacket
vcr = virtual channel to assign credit
cr = amount of credit
type = {null, reset, credit, no–credit, credit–only}
error = flush μpacket and message
tail = end of message

# Message Header

HIPPI Ifield:32 | mseq:32 | reserved:192

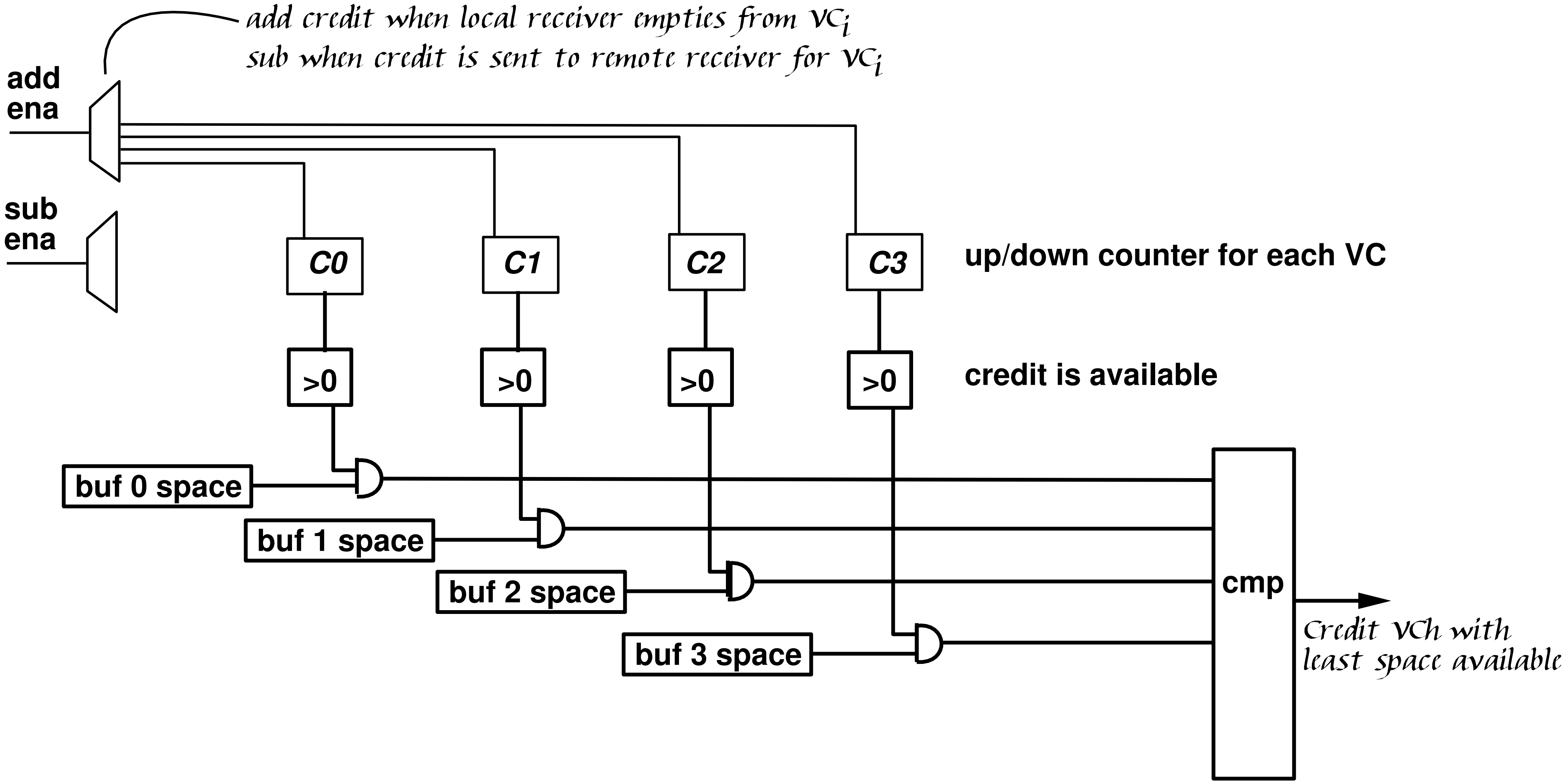
initial micropacket

# Credits

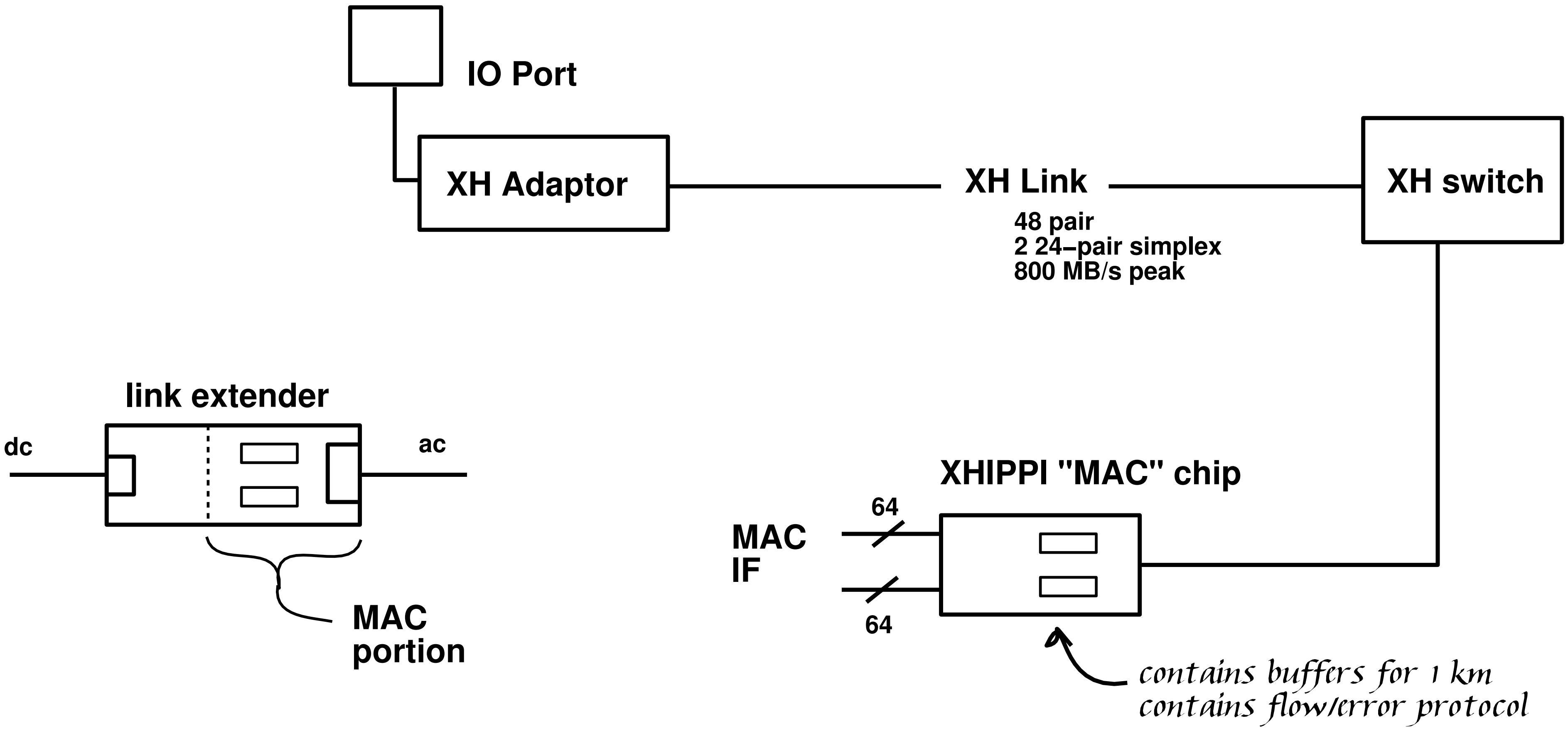



Acks are generated immediately and returned as piggybacks in the stream. Credits are generated when data exits through the next output buffer.

# Credit Control

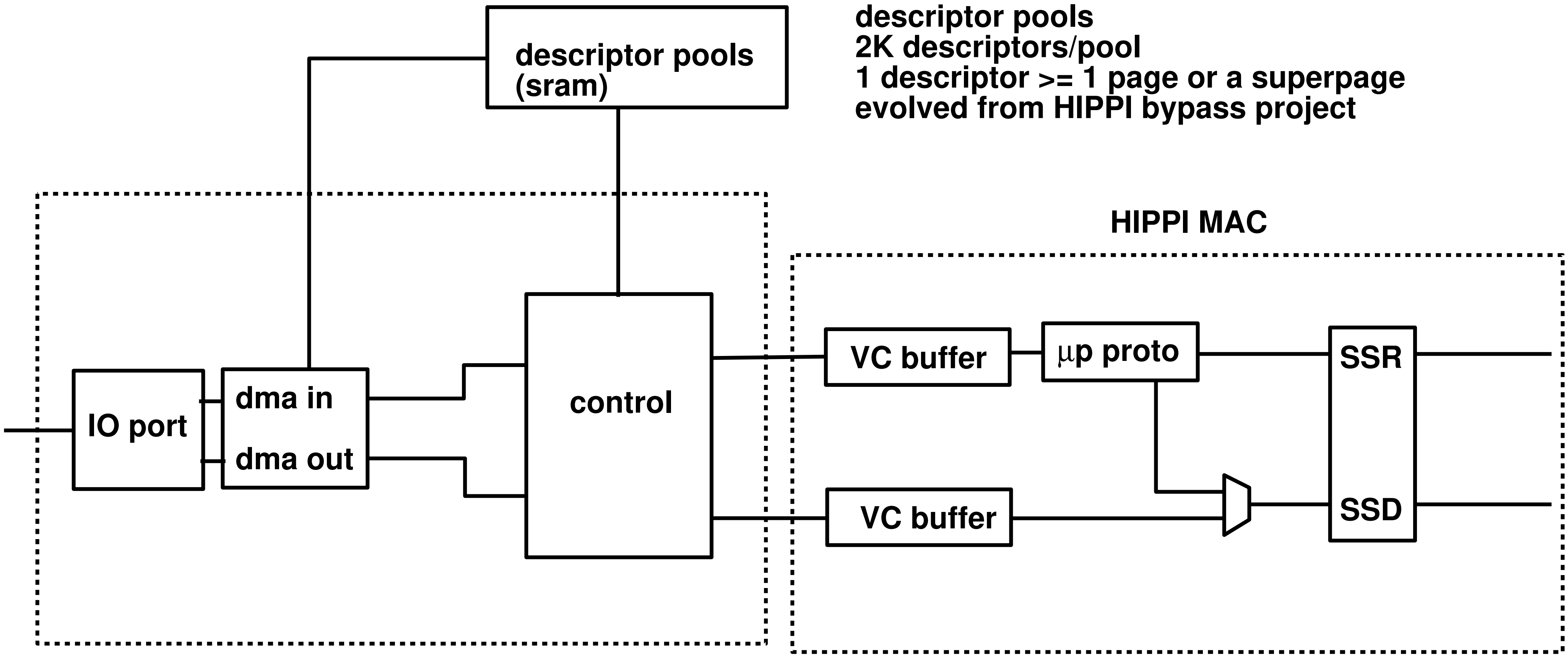


# System Components



# SuperHIPPI Adaptor

descriptor pools
2K descriptors/pool
1 descriptor >= 1 page or a superpage
evolved from HIPPI bypass project




All high-speed data buffers internal to vlsi circuits.

# OS Bypass

*Binding physical flow control to software*

1. Prototype on HIPPI.  demo at SGI booth at SC ’95.
   Details to be published.
   provides model for SuperHIPPI.

2. Application maps in buffer and descriptor pools.
   Structures are shared in HIPPI controller memory.
   Send a message == write a 128–bit descriptor.

3. No interrupts or kernel stack or context–switching
   once initialized.

4. Handles 8 distributed "jobs" on today’s hardware.
   limit is imposed by memory on the adaptor.
   new Serial–HIPPI adaptor not so limited.

5. Less than 100 us latency for short messages today.
   About 50 us on newer HIPPI hardware.
   HIPPI–6400 could achieve 1–10 us depending on the
   software model and host interface.

# NFS Bulk Data Server

1. current version demonstrates 3*60 MBytes/sec
   NFS streams (over 3 HIPPI boards) on a small Challenge
   (with sufficient disk arrays). Could do more.
   With SuperHIPPI, 500 MBytes/sec per stream is likely.

2. use: open("name", mode+O_DIRECT);
   library forks off pair of tcp streams to bds server.
   tcp is not needed if SuperHIPPI is end–to–end.

4. demonstrated as a user space hack. no security.

5. under development as standard product:
   kernel implementation
   public Linux port
   public RFC
   useful for many high–bandwidth bulk data applications

# HIPPI–6400 Summary

1. working hardware.
2. ready for standards–quality documentation.
3. no legal encumbrances.
4. commercial MAC device(s).
5. interworking with HIPPI–800 and HIPPI–1600.
6. readily available HIPPI–6400 switches.
7. OS integration strategy.