Summary of 'iWARP Forum'
held 12 September 1989, in Washington DC

Bradley C. Kuszmaul

Abstract: iWarp is Intel's implementation of CMU's WARP (systolic) architecture. Apparently, Intel decided to add some architectural features to make iWarp into a real computer :-). This paper is a summary of the 'First Annual iWarp Forum: A Direct Dialog with Technical Developers'.

Introduction

The iWarp (which stands for 'integrated Warp' or 'Intel Warp') parallel computing system is being developed by Carnegie Mellon Unisversity and Intel Corporation. Intel is responsible for implementing hardware and CMU is responsible for implementing software. The funding is provided by SPAWAR (Space and Naval Warfare Systems Command) through DARPA. Apparently the original iWarp contract called for a straightforward integration and speedup of the WARP architecture, but Intel has done some things, beyond the specifications of the contract, to extend the architecture.

Intel went to great pains to state that this meeting not a product announcement. This meeting is a status report. The implementation status of the project is that the chip is going to tape-out this week.

Technical Overview

The iWarp component is a 600,000 transisistor, 0.9 micron CMOS, custom VLSI chip, packaged in a 271 pin PGA. The die size is 551 mils square (14.0mm square). The hardware clock is 40Mhz.

It is expected that the chip can be mil-spec'ed (e.g., it uses the same process as the 80386, which is on the list of 'approved' parts for the space station - that means that it is pretty rad-hard).

Everything in the chip is static logic (except for the register read lines, which are precharged and then pulled down by the presense of a '0'.)

The local memory bus has 64 bit wide data bus and a 23 bit wide address bus. (The bus addresses are to 64-bit values, giving a 67 megabyte local address space)

The local memory is implemented with 25ns SRAM.

The communications network can do systolic communication with arbitrary interconnection patterns, and it is possible to do message passing pretty well. All communications is done through registers (e.g., to send a word to a neighbor, one writes that word to a register).

The floating point units operate at 10MHZ for 32 bit operations and 5MHZ for 64 bit operations.

The iWarp system's performance is about 1 GFLOPS per cubic foot.

Processor Architecture

Instruction set: (See Figure 1)
- RISC-like instructions (32 bit instructions)
  (running at 20Mhz for integer operations, 10Mhz for single precesion floating point, 5Mhz for double precision floating point.)

plus
Long instruction word (96 bit instructions) capable of
1 floating point add
1 floating point multiply
loop decrement, test, and branch
and either
2 memory address calculations (offset+base)
1 memory read (of a 64 bit value)
1 (additional) memory read or write (of a 64 bit value)
or
an ILU operation (including branch)

The long instruction runs at 10MHz if it is the fadd an fmul are single precision,
5MHz if either the fadd or the fmul are double precision.

The floating point units are not pipelined, and the result of the floating point operation can be used immediately in the next instruction. On the other hand, the floating point units slow the chip down.

They implemented floating point DIVIDE, REMAINDER, and SQRT directly in hardware because it was easier to do it right in hardware than to provide the hooks for software. (E.g., to do IEEE rounding and denormalized numbers correctly is tricky, so they just did it in hardware.) By 'easier' they said it actually took fewer gates just to implement the logic to do these operations in hardware than to provide the control lines for software. This is easy to understand, since the control lines would have just haired up the ILU's design.

Event support: One can explicitly test events e.g., with a busy-wait (polling the event flags), or one can set up an interupt vector for implicit event testing.

Registers: The register file is 128 32-bit words, accessable as 64 bit, 32 bit, 16 bit or 8 bit registers. The register file has many ports. Most of the registers (numbered 0 through 119) are just data registers that do not interact with the communications network; Those registers have 6 read ports and 3 write ports.

There are some registers which are 'special' (numbered 120 through 127) They are the interface to the communcations network and have 9 read ports and 6 write ports (for those registers there are presence bits or something; if a processor reads a register associated with a communications channel, and there is no data, the processor stalls. There is a timeout mechanism (configurable by the user)).

It appears that the register file serves partly as part of the memory heirarchy (i.e., things from registers are faster than things from memory), and partly as the synchronization namespace (e.g., all logical connections are named (at the source and destination processors) by registers. Synchronization is done on a word-by-word basis on the data in a logical connection. This seems to greatly increase the difficulty of defining the 'context' that might need to be swapped out in some sort of multi-threaded programming model (or even worse, for a multi-user operating system, since there are no protection mechanisms on memory, the communications network, and there is no automatic address translation (except for doing in the user software).

I understand that the 9 read-port, 6 write-port register takes about three times the area of a single-ported register. They were clever about the use of the bit-lines. To write a register, they use double-railed logic, but to read a register they use a single precharged bit line (and then they can use the BIT and BIT-BAR drivers to drive different bit lines).

Memory architecture:

The local memory is implemented using static RAM. They can put 6 Mbytes/board now using 256K SRAM (64K by 4) They expect to use 1M SRAM in 1990, and 4M SRAM in 1994. With 4 cells/board that is 1.5 megabytes/cell. With 1 cell/board that is 6 megabytes/cell. The SRAM is 25ns access time, allowing it to keep up with the 20Mhz processor with no wait states or interleaving. The local memory bandwidth is 160Mbytes/second. There is no ECC (it is not really needed for SRAM), but there is parity (parity exceptions can be handled by software in the on-chip ROM).

For power-consumption reasons, they run each of the four processors on the board out of phase, so that only one of the memory banks will be active at any given time. This reduces the peak instantaneous power requirements substantially.

Instruction cache:

There is a 1Kbyte I-cache. It is organized as 4 sections of 4 blocks of 16 words. I think this means each memory address is direct-mapped to one of 4 sections, and within a section, the cache is fully associative. The cache line size is 16 words. The cache does 'next instruction' prefetch (even across block boundaries). The cache also automagically aranges for the long instructions (3 words each) to be aligned correctly (again, even across block boundaries).

The communications network

Overview of communications protocol

When two computation agents want to communicate, there are three phases:

* Send a 'connection header'
  - which reserves unidirectional routing resources from one cell to some other cell in the array (like "dialing the telephone")
  - Contains the path for source to destination (expressed as 'streetsign routing'.
  - hardware allocates routing resources here.
* send messages. Each message has a head and a tail which are interpreted by the computation agent rather than by the communications agent. (I don't understand what this is used for) The header may contain some sort of 'virtual processor' address.
  A message is like 'a sentence spoken over the open line'.
* Send a 'connection tail' frees the resources and terminates the connection ('like hanging up the telephone).

It appears that, for a 2D interconnect topology, there is a 96 bit (three word) overhead for establishing and closing a connection.

It appears that there is a 64 bit (two word) overhead for the message header and tail.

In typical systolic computing, the connections are established once at the begining of the computation, and then a bunch of messages are sent during the computation, and when the task is done, the connections are terminated.

For message passing, the connections contain exactly one message, and there appears to be a 5 word overhead for sending a message. All of the examples that were 'shown' indicate that they believe that for message passing there are at least 5 or 10 words of data per message.

Streetsign routing:

Every node in the communications network has a collection of 20 bit 'streetsigns' (I don't know how many streetsigns can be stored at a given node). From a node, there are five directions a message can

go to and come from: Left, right, up, down, and to the communications agent.

Every word of the connection-header contains a 20-bit streetsign and an 'action' which is interpreted as "go straight until you find a matching streetsign and then take the action". The implementation of this is that when a connection-header arrives at a node, the node checks to see if the streetsign mentioned in the connection-header matches any of the streetsigns held by the node. If not, the connection goes straight (e.g., left->right, top->bottom, right->left, bottom->top). If the streetsigns match, then the action is taken (the action can say 'go left', 'go right', 'go up', 'go down', 'go to the computation agent'.)

Logical Pathways:

There are 20 logical pathways available in each node of the network. The 20 logical pathways are dynamically allocated to the 5 'directions'. Each of the five physical channels are multiplexed among the logical pathways allocated to that channel (they are multiplexed smartly enough that if only one of the logical pathways actually has any data to send, then that logical pathway gets the full bandwidth of the physical channel) Every word transmitted across a communications channel (from a intermediate-source to an intermediate-destination) is sent along with the logical pathway number (in the intermediate-source).

When a connection is established, a logical pathway is allocated to the connection. If they run out of logical pathways that is like 'running out of registers' or 'running out of memory'. It all sounds pretty dangerous to me.

spooling, streaming, and systolic communication

Spooling: Incoming(outgoing) data can be received(sent) directly to(from) memory to the communications network without processor intervention; There is a finite state machine to interface the memory with the router. The memory is accessed via 'cycle stealing', but I do not know whether the router has higher priority than the processor. There are 8 spool registers.

Streaming (Systolic): Incoming(outgoing) data can be received(sent) directly to(from) processor registers via registers reads(writes). When a processor reads a stream register, the processor stalls until a single word of data becomes available from the connection. If the processor reads the stream register, then the flow control mechanism backs up the message into the router. (Corespondingly for writes, if the processor tries to write a message and the flow control mechanism is saying "stop", then the processor stalls.) There is a user configurable timeout mechanism for the stalls (so that the processor can recover from the case that a message 'never' comes.). This mechanism is the 'systolic' computation mechanism, because the processors produce and consume data out of 'infinitely long' streams. Note that in streaming mode, the arrival of a word of data can also be implicitly or explicitly tested by setting up an event handler (i.e., an interupt) rather than stalling on read. There are 4 stream registers.

Note: The processor can combine spooling and streaming, effectively using memory as a buffer for a stream, by spooling data from the router into memory, and then streaming it from memory into the processor. (I.e., there is some mechanism to stream from memory as well as directly from the router). I asked about how they keep the consumer from overtaking the producer in that scheme, and it turns out that they use a single word of data (32 or 64 bits?) to represent the 'fifo' pointers, and there is a special condition code which checks to see if the 'read' pointer has overtaken the 'write' pointer. This condition code can be tested explicitly (polling) or implicitly (interupts).

Note: If my understanding of the router is correct, I know how to deadlock the router. They are providing an iPSC compatability package, and I know what program to write to deadlock the router.

Physical Channel:

A physical channel between two iWarp chips is implemented as follows: (I use many terms which the iWarp people don't use, e.g., 'flit' is the smallest unit of data physically transmitted.)

Physical Channel Data Format:
There are (See Figure 2.)
- 8 data signals
- 2 enqueue signals
- 1 parity signal
- 1 data-clock signal
- 2 dequeue signals (going back to the source)

for a total of 14 signals.

The handshaking is done on a 32-bit basis; and the 2 enqueue bits actually provide a total of 8 bits of data per handshaking. It takes four flit-times to send a word.

The enqueue bits say which logical path the word is associated with. The dequeue bits are used to say when a word from a logical path has been consumed.

The flit time is 25ns. (40 Mhz)

There is a 200ns latency through the chip for straight-through routing, 250ns latency to turn a corner (this appears to be true for every word of the message: It seems as though the connection-header is even slower, e.g., by at least another 100ns for the turns because the header is consumed at the turn, so it takes at least another 100ns before the next word of the connection-header can be sent to the next chip.

The sender keeps track of how many free spots are in the receiver (the FIFO is four words deep, but there is apparently one FIFO for each logical path: If I understand this correctly, that means there are 80 FIFO's (because each of the four physical inter-chip channels may have 20 logical paths on it). Note that this protocol means that even if the time delay between two chips becomes huge, the protocol is correct (with reduced effective bandwidth): The protocol would allow for four words to be sent from source to dest, and then it might take a while for the words to be consumed at the destination and for the 'dequeue' signal to come back. Then more data could be sent, but in the meanwhile, nothing can be sent.) They believe that they can build systems with 15 foot wires (50 ns propogation delays) before suffering any degradation with this problem. The reason they didn't build the FIFO's deeper may be related to the issue of there being 80 FIFO's (remember that the 80 FIFO's is guesswork on my part.)

Physical Channel Electrical Characteristics:
The wires are connected point-to-point (i.e. there is only one writer and one reader on each wire). Each wire only sends signal in one direction. Within a card-cage, the wires are single-ended (not differential pairs). If a signal leaves the backplane it goes through a converter and is transmitted as a differential signal on twisted pair. The converter sits on a card about 2 by 3 inches in size. The converter does its conversion in about 7ns. The converter chip, which requires only a 5V power supply, is made by AT&T. Thus, at the chip level, the number of signals equals the number of pins, however, they need a tremendous number of ground pins to avoid ground-bounce (the phenomena where the ground voltage locally (in space and time) pulled up (e.g., by as much as one volt) because of all current it is

sinking; this phenomena is much less of a problem for differential pairs because the current always goes right back where it came from.)

They send a data-clock along with the data. They did not understand my question about synchronization failure with the asynchronous clocks writing and reading from the 4 word FIFO. (Review: There is always the possiblity that, given asynchronous reads and writes out of a FIFO, that the system will enter a meta-stable state, causing the machine to fail. The standard solution is to somehow reduce the probability of such failure to an acceptable level. However, in the iWarp scheme, the read and write events are not asynchronous; they are just out of phase. For a naive implementation of this FIFO, there is some phase of skew that always produces a metastable state. The question is: How did the designers avoid the case where the write always happens at 'just the wrong time'?

speed and voltage

The iWarp channels run at 40Mhz (25ns), but they allow the wires to be up to 50ns long. They stack bits on the wires to make this work. The reason it can't be longer is because their FIFO's are shallow, and if the channel was longer, they would not achieve receive the "flow" acknowledgement soon enough to keep the channel busy. (Note: Their flow protocol will still work correctly, since it is a "consumed-the-word" signal rather than a "flow")

The channels run with 3.8 volt swings (according to everything I have seen). (It looks like high-out is $V_{cc}-0.8$ volts and low-out is $0.4 $ volts. $V_{cc}=5$ volts.

However this voltage swing seems inconsistent with the chip power budget (5 to 7 watts) and the board-level power budget (50 watts). They are using 50 ohm parallel terminated (terminated at the driver) transmission lines.

The best guess that I can come up with to make this work is that the 50 ohm termination resistor built into the driver must sink no power at 2.3 volts, and when driving a logic 1 (at 4.2 volts) or a logic 0 (at 0.4 volts) the termination resistor sees a 1.9 volt drop. The power disipated by that resistor is then 72.2mW. There are 56 driven pins for the communications networks, giving 4.04W. However, now I still have not accounted for the power disipated by the driver itself. There is a 0.8V drop from the power supply to high-out, so each driver is disipating (via resistive heating) at least 12.8mW, for a total of 0.71W, and during switching the resistive load through the driver is somewhat higher. I don't neccessarily understand where the power is disipated in these systems, but at the minimum I can count up 4.75W just to drive the pins to the communications network. If the memory pins are also using some sort of tranmission line, then there are another 100 pins worth of stuff to drive, and if the memory wires are capacitors, the power is like CV^2f, and since f is so high there must be significant power disipated there.

They have invented a 50 Ohm driver which automatically compensates for voltage, tempurature, and process variation. They claim to have a patent-pending on this driver. They said something about a charge-pump type circuit to control the Vref (the reference bias voltage) on the output driver. (Note: This sounds very similar to Tom Knight's low-voltage self-terminating output-driver. The reference bias is being used to control the resistance of the termination resistor).

Clock Skew (see Figure)

2ns on a board
4.4ns between components inside a card cage
28.4ns between anything (this is done with the differential signals etc.) Most of this clock skew seems to come from unmatched cable

length. The main constraint they have is to keep the hold-time in good shape. Apparently they are right on the edge in the worst case. (Note, hold time is hurt when the clock skew makes the sender change state before the receiver has sampled the input. Slowing down the clock does not fix this problem (slowing down the clock can improve the setup-time), so it is really important to get this right. This is a symptom of using edge-triggered clocks instead of level-sensitive logic. Maybe it never occured to these guys to use level-sensitive logic.

Packaging: The packaging was very impresive. The whole system seemed rugged and clean.

The chip is a 271 pin PGA (die facing downward, huge heat sink on top). ~100 pins are memory, ~100 pins are communication, ~70 pins are power and ground. The power budget for the chip is 7W.

Each card is 9" by 11" and can hold either one or four processors and a total of 6 Megabytes of memory (divided among the one or four processors). They indicated that they plan to mostly use the four-processor cards.

There is a daughter board to hold extra surface mounted memory. Intel has developed a 132 pin surface mount connector to connect the daughter board to the mother board. (They have 3.5 signals for every power or ground connection).

The power budget for the card is 50W.

The cardcage holds 16 cards (16 to 64 processors), and has its own fans and power supply. The cardcage can sit in any 19" rack or on a tabletop, and it will run fine. The cardcage is UL approved (but not FCC certified).

The power supply is mounted behind the backplane on a roll-out chasis to provide room to get into the space behind the backplane and reconfigure it. The power supply uses a 220 volt 3-phase supply (to keep the AC current down for UL approval), and produces 5 Volts at 300 Amps on the board size (you could do some serious arc welding if you pulled a board out with the power on). They said something about using a ''power factored'' power supply instead of a ''linear'' power supply (I don't know what that means).

For signals leaving a cardcage, there is a specal differential converter board (mentioned above) mounted behind the card cage.

They use a 50 Ohm controlled impedence ribbon cable to rewire the backplane. (Apparently the backplane can be reconfigured away from the 'standard toroidal configuration'.

The connectors for plugging the card into the back plane are 'through connectors', so one can plug things directly into the backplane and they end up connected to the cards. They plug the 50 ohm ribbon cable here, and they plug the differential convert card here. The back-plane connector has 480 pins.

The back-plane is 12 layer 50 Ohm strip-line (controlled impedence). There are 3 ground planes and 2 VCC planes. They have measured less than 30mV drop across the backplane (under some test to measure ground bounce and so forth.) They use a bus-bar to stiffen the back-plane (in the

They are extremely worried about ground-bounce (which they wouldn't have had as much of if they had used differential signals everywhere).

The cards are mounted vertically and air flow is from bottom-to-top in the card cage. The air (for cooling) is passed through the bottom card cage then the next card cage and so forth (so the last card cage has a warmer air source). They move the air at 300 linear feet per minute.

The container holds 4 cardcages. The container is all metal (making it easy to satisfiy FCC requirements and simplify the problems associated with electro-static-discharge (ESD, aka lightning). The front door has an LED panel on it. There is a serial processor (an 8251?) somewhere in the machine that sequentially polls the status line of every iWarp chip and updates the LED display at 10Khz. There is a yellow and green LED for every iWarp chip, and a red 'error' LED for every card cage. They have been careful to leave space to route the cables cleanly; One will not see masses of cables hanging out anywhere (conversely, they did not really push the cabling density). The container is the same one used for the iPSC (except the iPSC is grey and the iWarp is black).
The power budget for the container is 5KW.

The biggest system that they are advertising is a 4-container system (i.e. 256 to 1024 processors). This size limit is really a result of the clock distribution board rather than anything else, so they could probably easily make it bigger.
Clock distribution: Amazing chips (see Figure) to keep clock skew below a few nanoseconds inside the same cardcage. I don't understand how they lose so badly between cardcages (28 ns)
Sun interface board (See Figure 5) This is a VME master or slave board. In master mode the iWarp cell is the bus master (for an array-centric configuration). The board can also be a bus slave (for a host-centric configuratin) It is a big board (the standard 'sun' size rather than the shorter 'VME' size). This holds one processor which can be hooked into an array via its communications paths.

To interface a new device (e.g., a disk drive) to their machine, one uses the local memory bus. The memory bus is very simple (one Intel engineer said it was 'embarassing' . One can build a memory-mapped I/O device (they showed a typical SCSI interface). One could also use a dual-ported memory to interface to their device. They do not recomend trying to interface directly to the communications network; use the iWarp component to go from the communications network to the memory bus format.

software: I did not attend the software session (because it conflicted with the hardware session - what a lose). The software guys from CMU seemed to be not nearly as excited about their work as the hardware guys from Intel, so I went with the hardware session (to learn about the packaging and electrical characteristics). The software looks pretty bad - they claim to be compiler driven, but the general impression I got from the CMU guys was that the software could be made a lot better - they believe in program generators, and if their compiler can't analyse your program, I guess you can't run it... Basically they are talking about AL and APPLY and all the standard iWarp software, which to me looks hard to program in. The model of all that software is that you build a special purpose machine 'in the shape of your problem'. The hardware supports that model by allowing the logical connections to be interesting.

They claim to have a C compiler (you can run C in any cell of the iWARP), a FORTRAN 77 compiler (again, this is a serial compiler). The compilers do the long instruction word optimization, you can call C from FORTRAN, you can call assembly language from C, you can access the communications primitives from C and FORTRAN. (E.g., communications code is just as fast in C as in assembly language).

They are providing an iPSC compatability package (to allow iPSC programs to run on the iWarp). Except for the message deadlocking issue, this looks like a very fast implementation of iPSC.

Future trends

Intel and CMU both talked about future trends. The intel stuff was interesting; they had specific goals and issues. The CMU stuff, with a few exceptions, was generally more abstract, like ''we have got to work on software''.

HTK said they will hook up iWarp into the Nectar network (100Mbyte/sec fiber optimic with 16x16 crossbars). They will also develop an HSC interface (800 to 1600 Mbits/sec). (is 800 megabits the same as 100Mbytes? I may have this wrong...)

GWC said intel would probably

- implement quad-flat-pack & mil-specify the part (maybe in about 18 months)
- iWarp is the lead project in the Intel Multi-chip-module research. They hope to get the 1.5 Megabytes of memory plus the iwarp chip all onto a 2-inch square footprint (maybe 18 months to 2 years)
- Component shrink (iWarp 1.5)
    - The next process is a 3-layer CMOS process, with a factor of 2.5 to 4 improvement in density.
    - Same instruction set
    - 64 bit floating point directly supported (at the same speed as 32 bit floating point)
    - FP pipelining
    - big data and instruction cache
    - bigger busses (e.g. to the cache-line width to the local memory)
    - 50 to 80 Mhz clock
    - 100 to 160 MFLOPS per cell (maybe in 24 to 30 months)
- iWarp 2
    - More unification of communications models (e.g., currently they do message passing and systolic communication 'well'. They hope to do small-talk style things too (ala Dally), and global-shared-memory things too.
    - heterogeneious nodes
    - locally shared memory (e.g., several nodes with physically shared memory and then outside of that they do message passing)
    - 3d packaging

They hope to make iWarp into an industry standard 'backplane' for connecting computers and processors together.

Names of speakers

George W. Cox (GCW) (hands on manager)
Craig Peterson (chip guy?)
David Riss (software)
Dick Hofscheier (packaging engineer)
Les Furnanz (don't know: more of a marketing attitude than engineering attitude.)

CMU
H. T. Kung (HTK)
Thomas R. Gross

Darpa
Stephen Squires

Bibliography (I have these as part of the handouts)

OSTA, 'The Federal High Performance Computing Program', 9/8/89 (available from DARPA, Call Steve Larson (Squires's secretary) at 202-694-5800) (Not in my notebook)

Intel, 'Introduction to iWarp', preliminary manual.

Borkar, et al., 'iWARP: An Integrated Solution to High Speed Parallel Computing', proc IEEE supercomp. conf, orlando FL, Nov 1988

Cohn et al, 'Architecture and Compiler Tradeoffs for a Long Instruction Word Microprocessor', 3rd Int. Conf on Arch. Support for Prog. Lang. and Op. Sys. (ASPLOSS III), Boston, MA, Apr, 1989.

Hamey et al, 'APPLY, A Programming Language for Low-Level Vision on Diverse Parallel Architectures', to appear in Parallel Computation and Computers for AI, Janusz Kowalik (ed), Kluwer Academic Publisehrs, 1987.

Kung, 'Network-based multicomputers: redefining high preformance computing for the 1990's', Decennial Caltech Conf on VLSI, Pasadena, CA, March, 1989.

Compute and Access "*Long*" Instruction

96-bit Instruction Format

Word-1

31	30—29	28—(4)—25	24—(4)—21	20—(7)—14	13—(7)—07	06—(7)—00
J	1 1	Data Mode	FADD	B operand Reg	A operand Reg	K operand Reg

Word-2

31—(9)—23	22—21	20—(7)—14	13—(7)—07	06—(7)—00
Memory Control	FMUL	M operand Reg	N operand Reg	R operand Reg

Word-3

31 30	29—(7)—23	22—(7)—16	15 14	13—(7)—07	06—(7)—00
OP 1	Offset 1	Base 1	OP 2	Offset 2	Base 2

Operand for 1st Read Access **Operand for 2nd Read / Write Access**

—OR—

Word-3

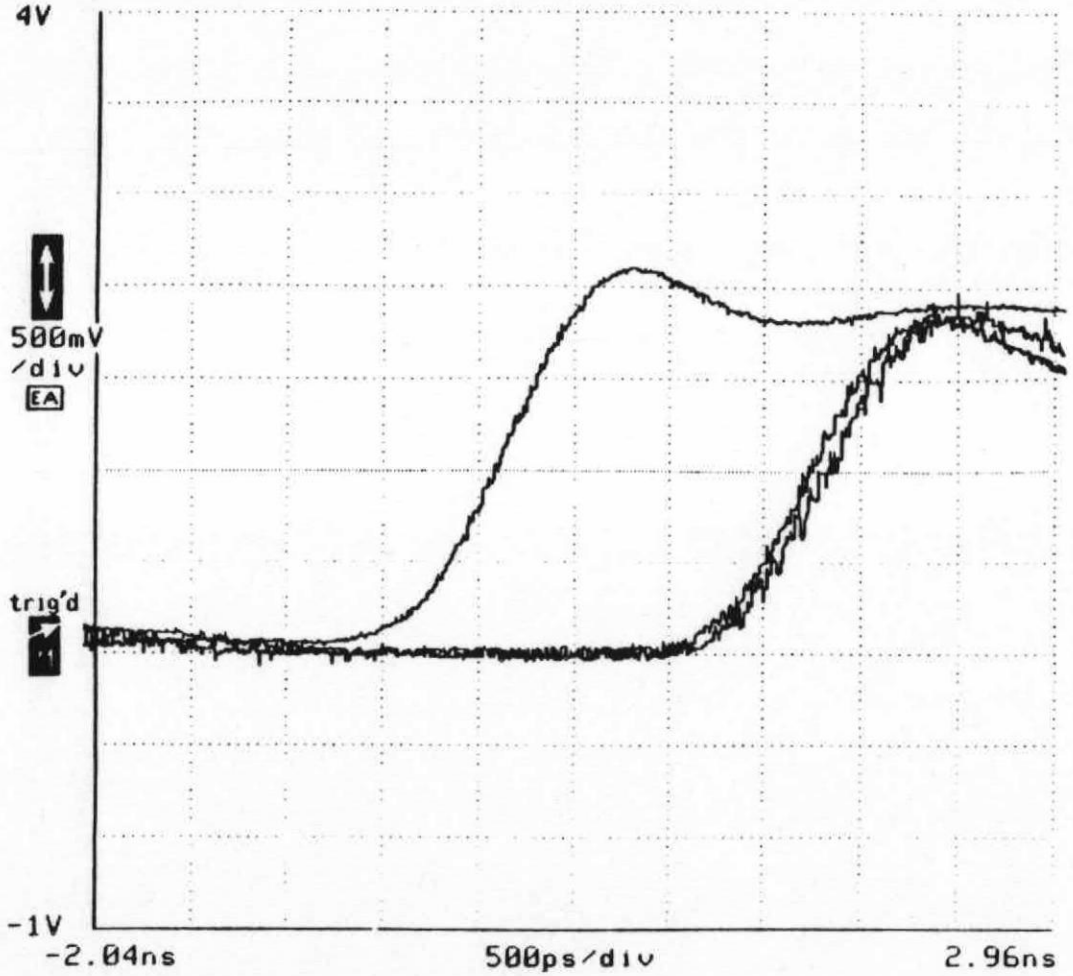
31—(32)—00
Full ILU (Integer Logical Unit) Instruction or general Branch operation

General Purpose "*RISC-like*" Instruction Summary

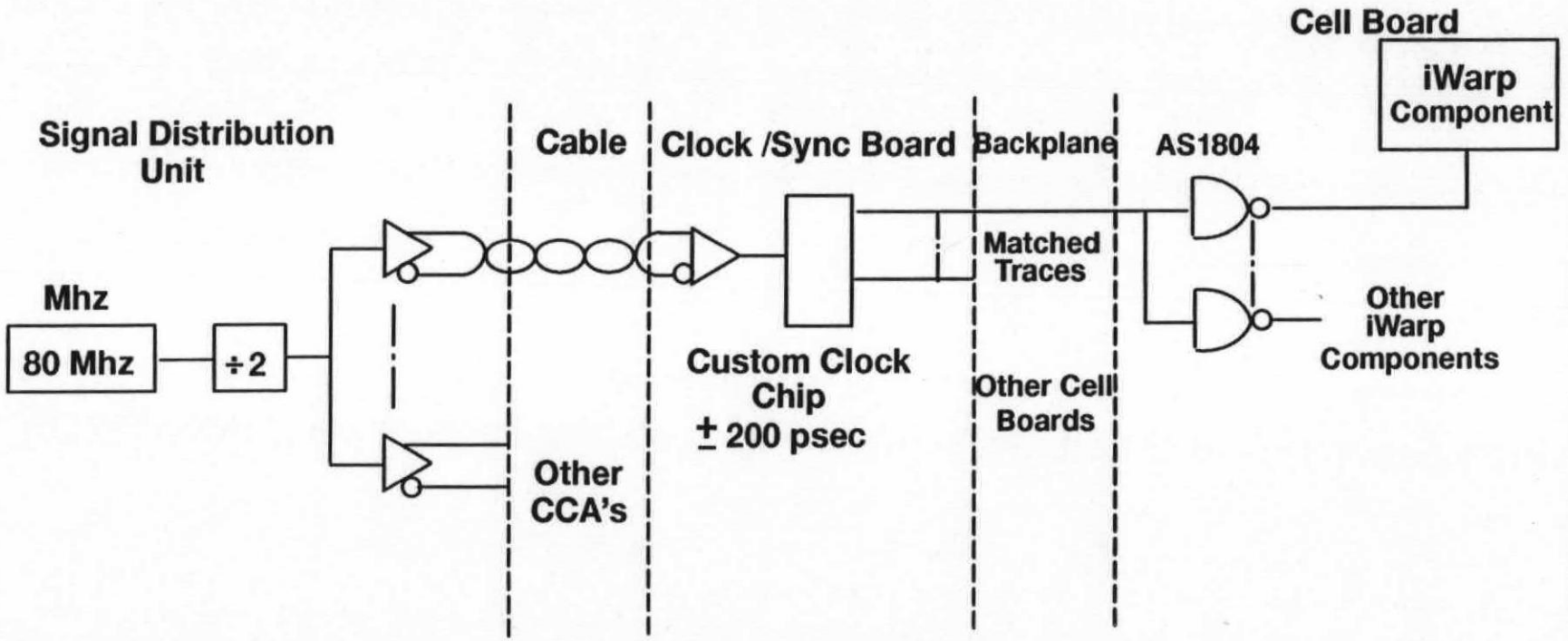
32-bit Instruction Format

Integer/Logical Operations	**Logical ops, Arithmetic ops, Bit ops, Shift & Rotate, Find MSB**
Floating-point Operations	**Add, Sub, Compare, Max, Min, Logb,Scale Mult, Div, Sq Root, Remainder**
Data Conversion Operations	**Integer to Floating-point, Floating-point to Integer**
Memory Access Operations	**Byte, Half-word, Word, Double-word**
Flow Control	**Call, Return, Branch, Push, Pop, Break, Enter loop (Implicit Loops), Stack control**
Extended Flow Control	**Absolute call/branch, Indirect call/branch**
Literal Loads	**Load literal**
Communication Support	**Pathway control, Spool control**
General Control	**Event control, Timer op, Pointer cntrl**

Clock System Performance



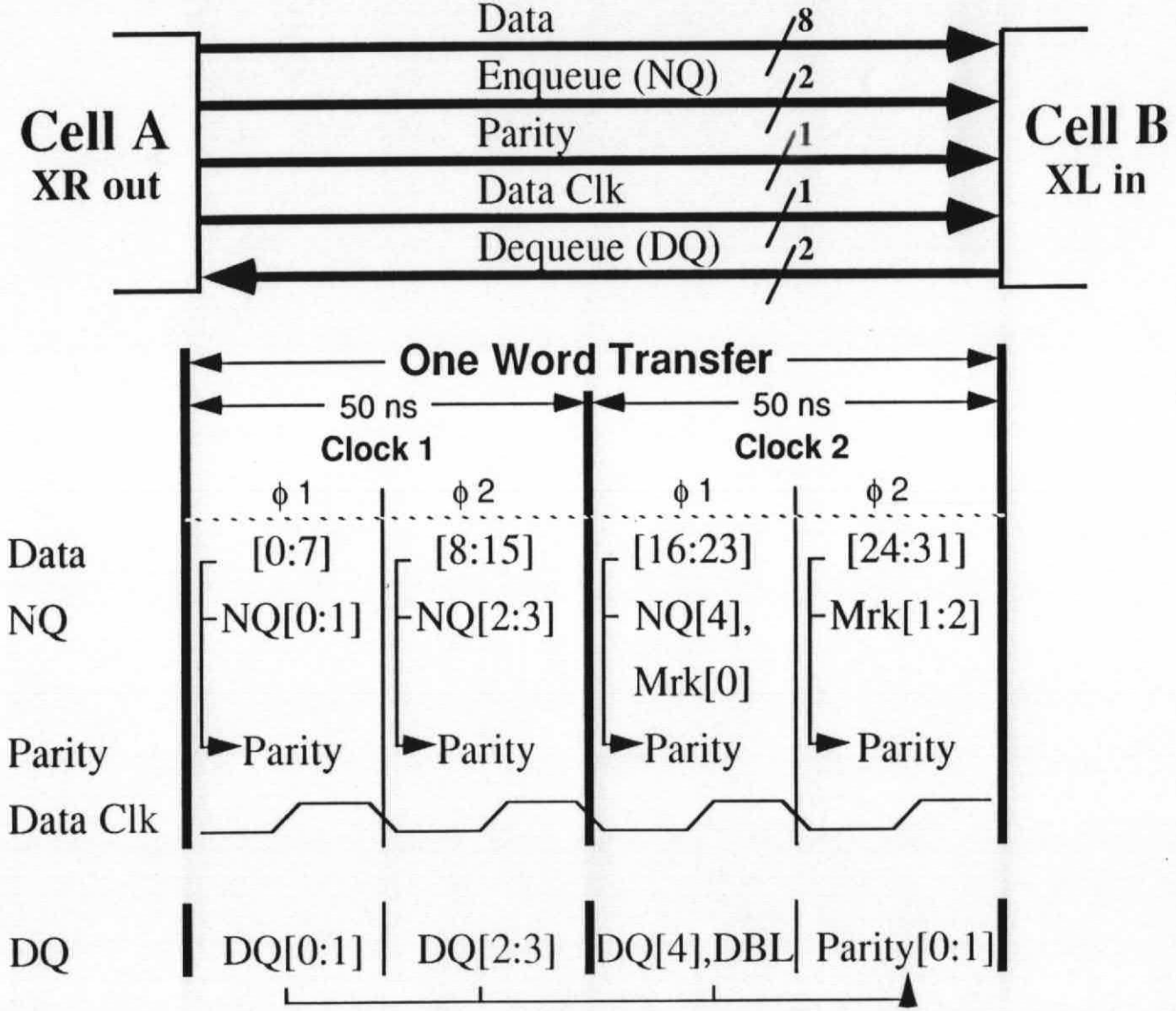
System Clock Distribution



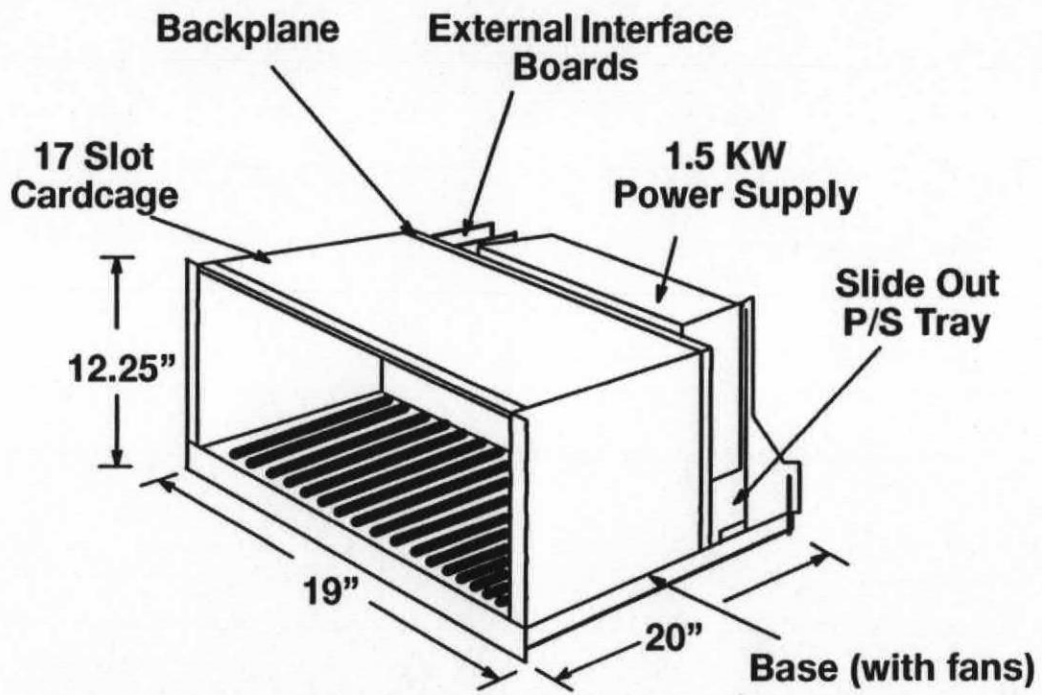
iWarp Component to Component Clock Skew

Same Board < 2ns'
Different Boards, Same Cardcage Assembly < 4.4ns'
Different Cardcage Assemblies < 28.4ns'

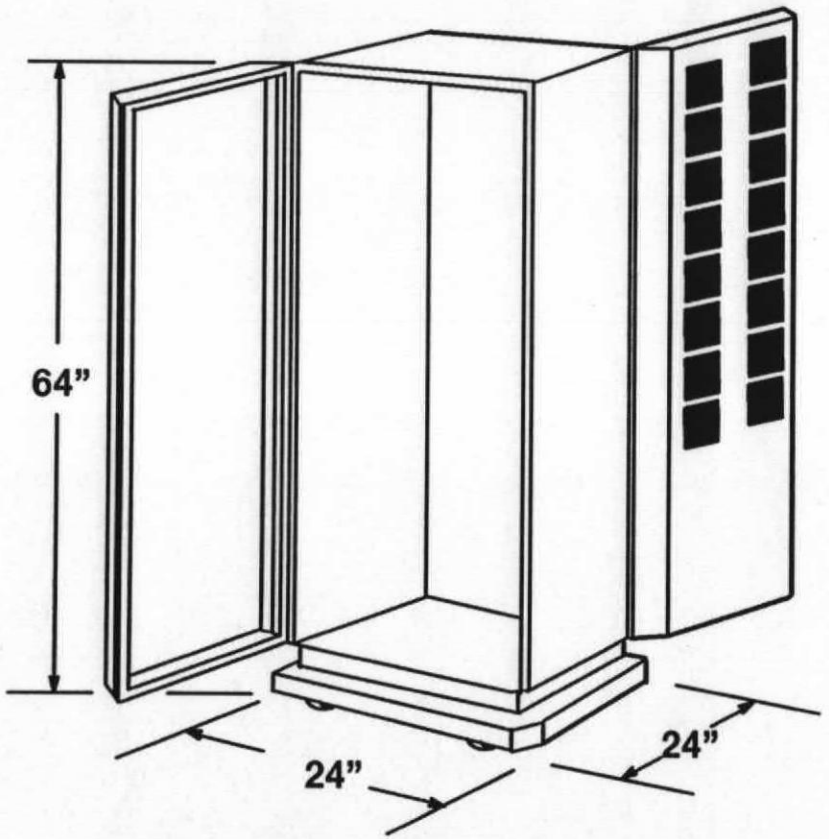
External Pathway Pins and Handshake



Cardcage Assembly

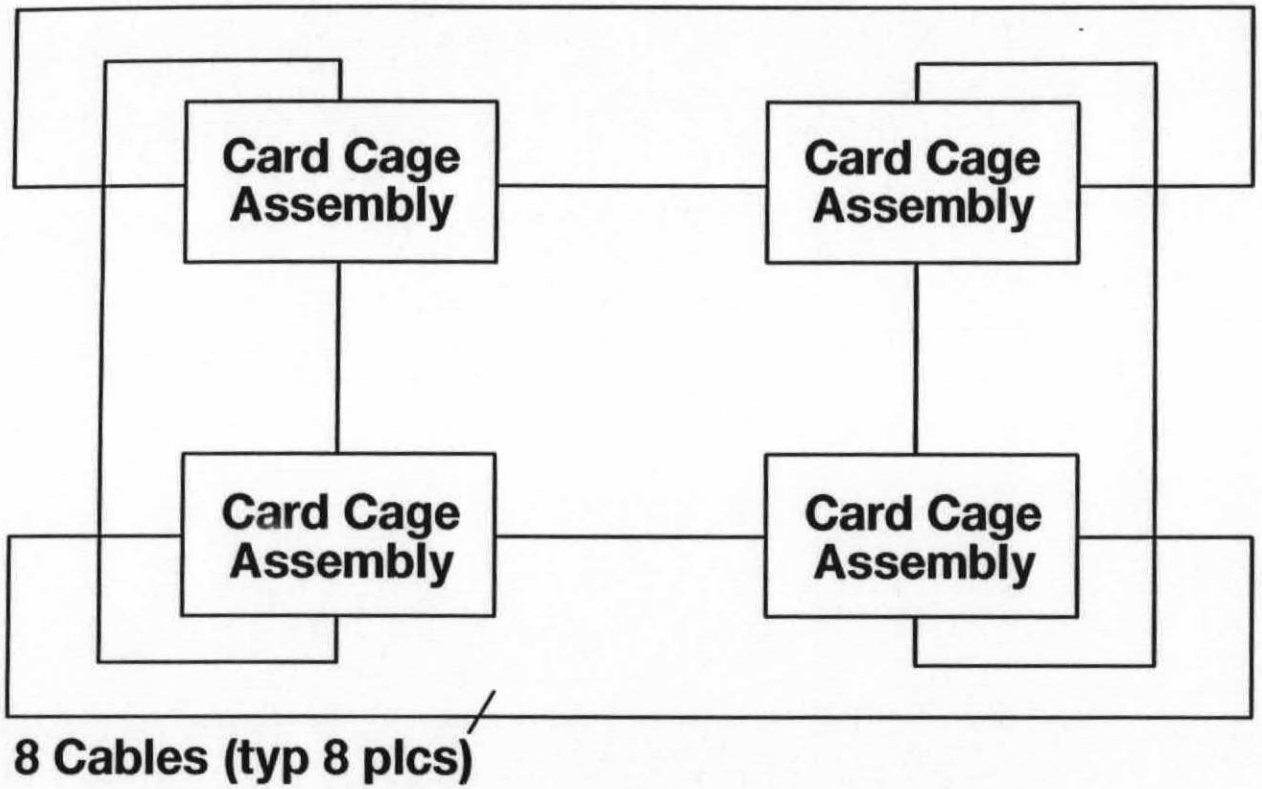


19" Container



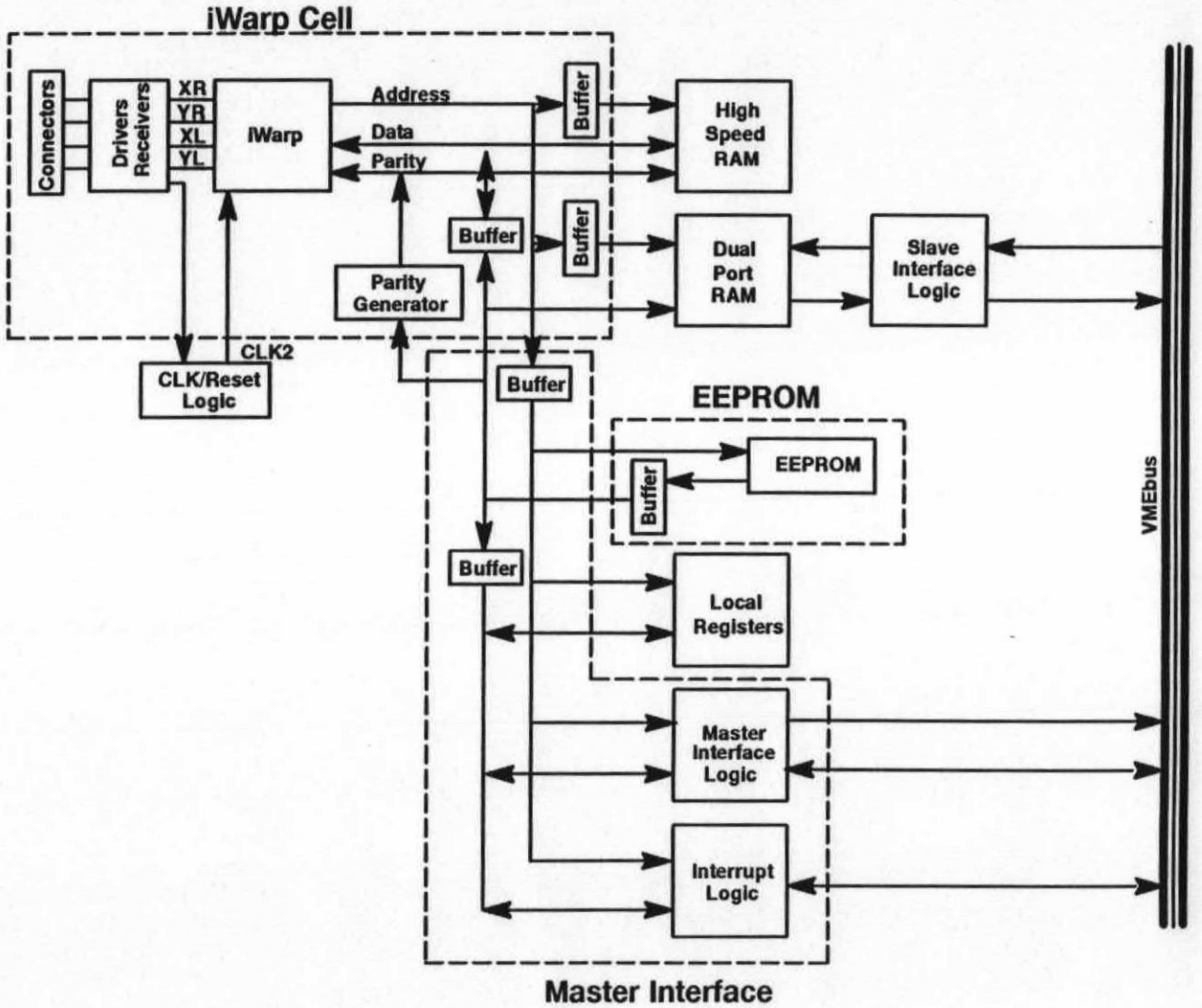
29 Rack Units (1 Rack Unit = 1.75")
5K WATTS

Max Single Container Array



16 x 16 Quad Cell Array (5.12GFlops)
8 x 8 Single Cell Array (1.28GFlops)

Sun Interface Board Block Diagram



The external connection to the iWarp array can be made at any of the loops in either the X or the Y direction.

The iWarp System

Flexibility is the key characteristic of the iWarp system. From one to four iWarp Cardcage Assemblies reside in a single System Cabinet, and up to four cabinets can be connected to form even larger arrays. With a system of four cabinets, an iWarp system can be extended to a 32 by 32 array of 1024 iWarp cells. Figure 2-21 shows the iWarp System Cabinet, which contains up to four Cardcage Assemblies.

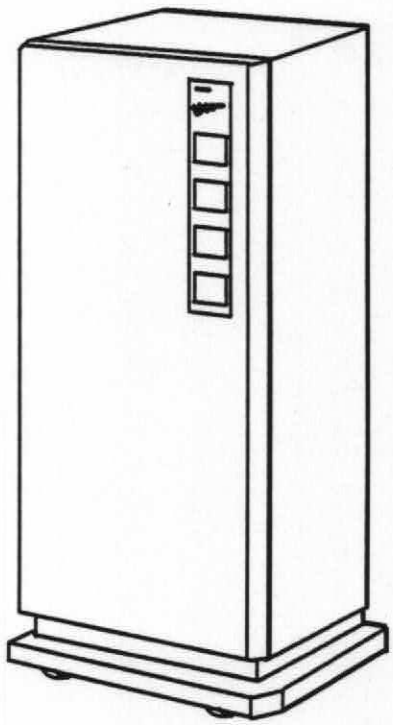


Figure 2-21: iWarp System Cabinet

The front door of the System Cabinet contains an LED display that shows status conditions for each iWarp cell housed in the cabinet. The LED display consists of four 8 by 8 LED arrays, with each array corresponding to one of the cardcages in the cabinet. Each pair of LEDs in the array corresponds to the status of a specific cell. There is also an error LED and a power LED for each array.