NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

LINC; THE LINK AND INTERCONNECTION CHIP

by

F.H. Hsu, H.T. Kung, T. Nishizawa, A. Sussman

DRC-15-23-84

December, 1984

LINC: The Link and Interconnection Chip

F.H.Hsu
H.T.Kung
T. Nishizawa
A. Sussman

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

May 1984
(Last revised October 1984)

Copyright ® 1984 F i i Hsu, RT. Kung, T. Nishizawa, and A. Sussman

The research was supported in part by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539 and in part by the Office of Naval Research under Contracts N00014-76-C-0370. NR 044-422 and N00014-80-C-0236, NR 048-659. It T. Kung was supported in part by a Guggenheim Fellowship.

620.0042
C28d
DRC-15-23-84

Abstract

The link and interconnection chip (LINC) is a custom chip whose function is to serve as an efficient link between system functional modules, such as arithmetic units, register files, and I/O ports.

LINC has 4-bit datapaths consisting of an 8x8 crossbar interconnection, a FIFO or programmable delay for each of its inputs, and a pipeline register file for each of its outputs. Using pre-stored control patterns LINC can configure its interconnection and delays on-the-fly, while running. Therefore the usual functions of buses and register files can be realized with this single chip.

LINC can be used in a bit-sliced fashion to form interconnections with datapaths wider than 4 bits. Moreover, by tri-stating the proper data output pins, multiple copies of LINC can form crossbar interconnections larger than 8x8.

Operating at the target cycle time of 100 ns, LINC makes it possible to implement a variety of high-performance processing elements with much reduced package counts. This reduction of chip counts is especially significant for cost-effective implementations of those multiprocessors such as systolic arrays which call for large numbers of processing elements.

This paper gives the architectural specification of UNC, and justifies the specification by some application examples.

1. INTRODUCTION

Many high-speed, high-density building-block chips arc rapidly becoming commercially available. Notable examples are 32-bit floating-point chips that can perform an arithmetic operation in less than a microsecond. If there is efficient hardware support to link these chips together, then very powerful systems can be built at low cost

LINC is a super "glue" chip for system construction. As depicted in Figure 1 the chip provides physical communications and data buffering between functional units of a system. It can also efficiently implement some complicated data shuffling operations such as the corner turning used in packing bytes into words and unpacking words into bytes.

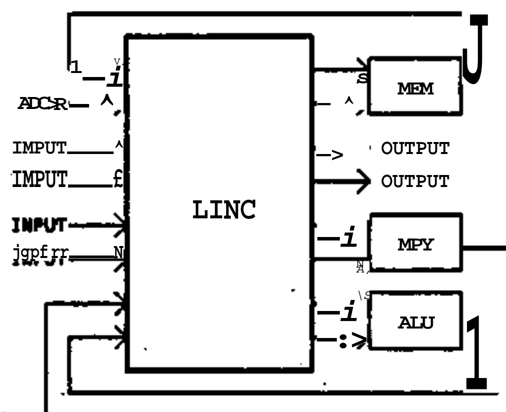


Figure 1. System components linked" together by LINC

LINC can significantly reduce chip counts in many systems, especially for highly pipelined processors such as the CMU Warp processor [5,6]. In general, LINC can efficiently link high-speed, off-the-shelf arithmetic chips together to form powerful multiprocessor systems for a variety of applications, such as 3-D computer graphics and robot arm control

The LINC architecture is highly regular. This makes the chip most suitable for custom VLSI implementation. As of October 1984 the logic design of UNC has been successfully simulated on a Daisy design station at CMU, and circuit and layout designs in CMOS technology are being carried out in cooperation with the General Electric Company in Schenectady, New York.

UNIVERSITY LIBRARIES
CARNEGi'E-MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA 15213

2. SYSTEM OVERVIEW

This section gives an overview of LINC from the perspective of its function within a system and its interface to the outside world.

The signal I/O pins of LINC, classified into four groups, are listed in the table below.

INPUT SIGNALS PIN NAME	ABBREVIATION	OUTPUT SIGNALS PIN NAME	ABBREVIATION
DATA I/O:			
A input	AI[0-3]	A output	AO[0-3]
B input	BI[0-3]	B output	BO[0-3]
C input	CI[0-3]	C output	CO[0-3]
D input	DI[0-3]	D output	DO[0-3]
E input	EI[0-3]	E output	EO[0-3]
F input	FI[0-3]	F output	FO[0-3]
G input	GI[0-3]	G output	GO[0-3]
H input	HI[0-3]	H output	HO[0-3]
FIFO CONTROL AND STATUS:			
Write A-FIFO	WAF	A-FIFO (almost) full	AFF
Write B-FIFO	WBF	B-FIFO (almost) full	BFF
Read A-FIFO	RAF	A-FIFO (almost) empty	AFE
Read B-FIFO	RBF	B-FIFO (almost) empty	BFE
CONTROL PATTERN MEMORY ADDRESS:			
Control address	CA[0-4]		
LOADING AND TESTING:			
Chip select	CS		
Mode control	MC[0-3]		
Run/~Halt	R/~H		
Reset	RESET		
Ctrl pattern in	CC[0-7]	Ctrl pattern out	CC[0-7]

Note that pins CC[0-7] are bidirectional.

In addition to signal pins there are two clock pins (PHI1 and PHI2), two power pins (VDD1 and VDD2), and two ground pins (GND1 and GND2). Thus LINC has a total of 98 pins. Using a standard 100 pin grid array package, LINC has two pins reserved for possible future needs.

A system overview of LINC, omitting features related to loading and testing, is depicted in Figure 2. Between each data input port and the crossbar is a FIFO or programmable delay (FPD), and between the crossbar and each data output port is a pipeline register file (PRF). A PRF is a set of registers that shifts in its

input in a pipelined manner, but allows random access into the pipeline for its output. The AMD AM29520 is an example of a PRF.

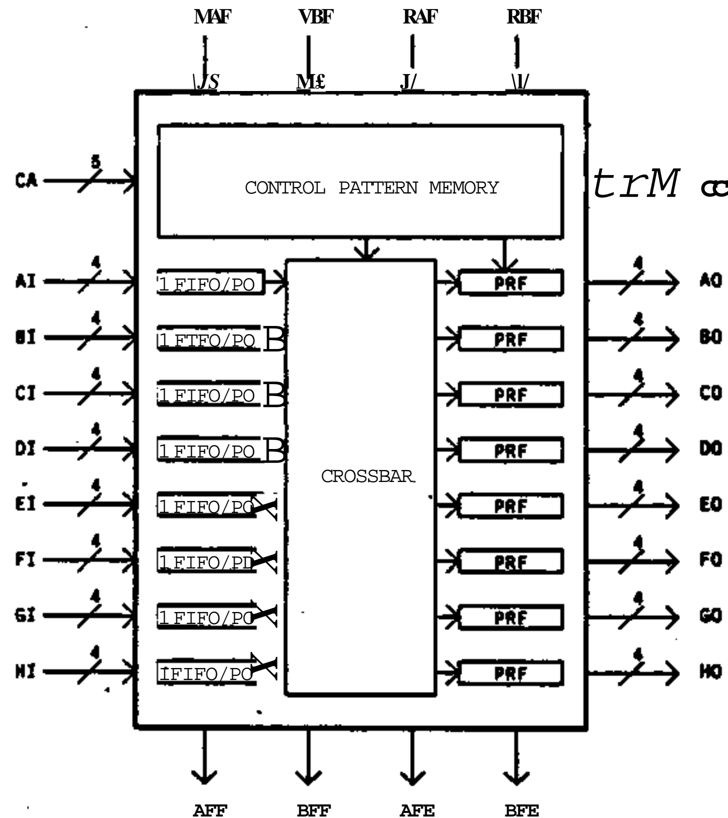


Figure 1 System overview of UNC

In the following we briefly describe the functions of the signal I/O pins in each of the four groups listed in the table above. Sections 3 and 4 will give detailed descriptions of these functions.

DATA I/O:

Through the data I/O ports, LINC can input as well as output eight 4-bit data items simultaneously every cycle. With a cycle time of 100 ns, this means a total data I/O bandwidth of 80 Mbytes per second. It is possible to increase the I/O bandwidth further by using multiple copies of LINC in parallel. Figure 3 (a) illustrates that UNC can be used in a bit-sliced fashion to form interconnections with data paths wider than 4 bits. Figure 3 (b) illustrates that by tri-stating the proper data output pins, multiple copies of UNC can fbnn crossbar interconnections larger than 8x8.

Suppose that LINC inputs data from its top boundary and outputs data along its right boundary. Then the 16x16 crossbar interconnection of Rgure 3 (b) can be laid out in a regular and compact manner, as shown in Figure 4. It is straightforward to generalize this layout scheme to implement larger crossbar intcrcon-

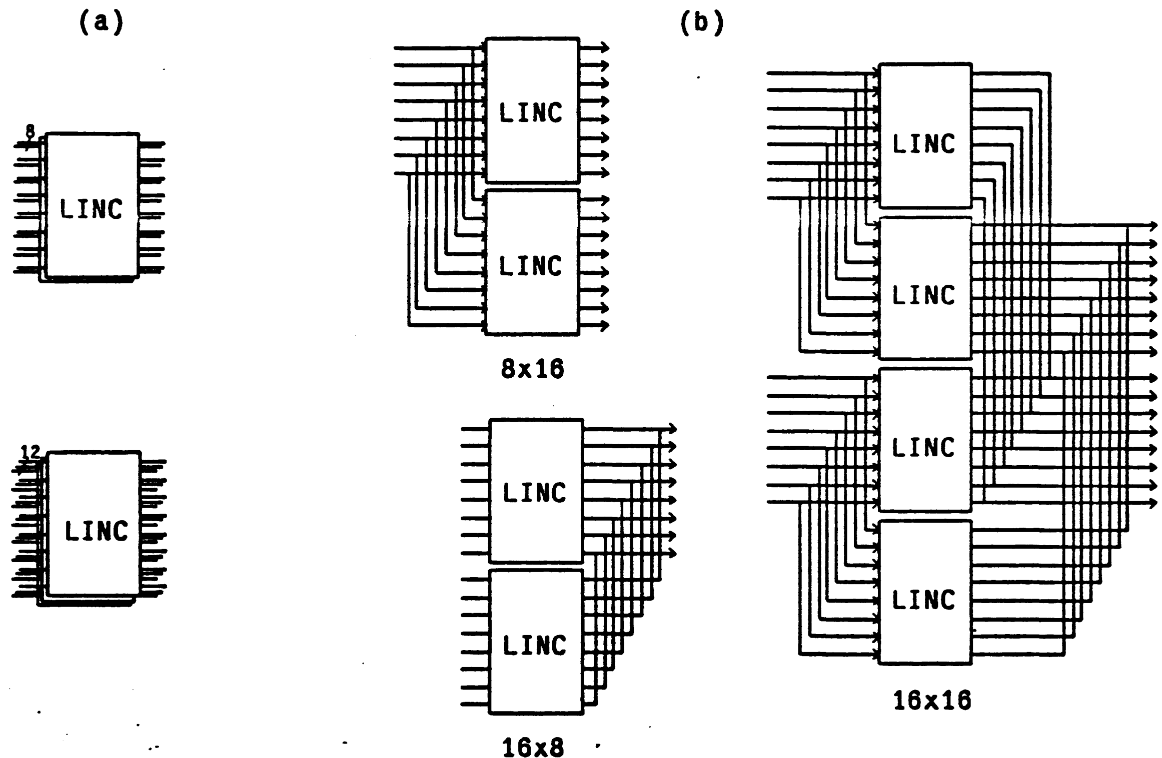


Figure 3. Multiple copies of LINC *to form (a) interconnections with data paths wider than* 4 *bits, and (b) interconnections larger than* 8×8

nections.

FIFO CONTROL AND STATUS:

LINC can be configured to have up to two FIFOs, an A-FIFO and a B-FIFO. The FIFOs' widths can be set by programmers in 4-bit increments, but the total width of the two FIFOs of course cannot be more than 32-bits (the total width of the eight input data ports). Input data ports not used by the FIFOs are configured as programmable delays. Typically, the programmable delays are used to equalize the lengths of various pipelines for different arithmetic units in the same system.

The FIFOs of LINC can be used to buffer data coming from other systems at varying rates. Figure 5 depicts a simple scenario of a collection of cooperating systems—each system receives data from the system to the left. (For an instance of such cooperating systems, see the geometry system application of LINC in Section 6.5.) The controller of each system sends FIFO read requests (RAF, RBF) to the LINC of the same system, but sends FIFO write requests (WAF, WBF) to the LINC of the system to the right. The LINC of each system sends its FIFO status signals AFE and BFE (empty or almost empty) to the controller of the same system, but sends status signals AFF and BFF (full or almost full) to the controller of the system to the left. A status signal may be sent before a FIFO becomes *completely* full or empty, to give sufficient time for the signal to reach the controller.

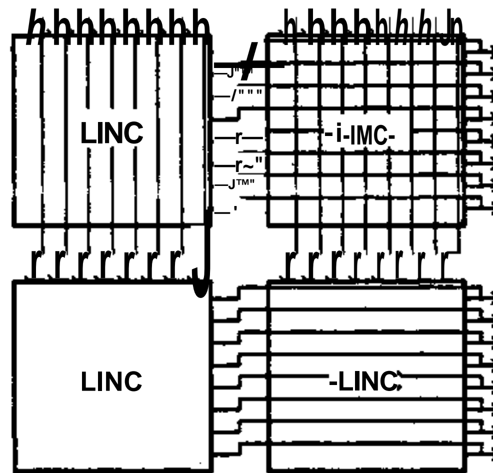


Figure 4. Regular and compact layout of the 16x16 *crossbar interconnection of Figure 3 (b)*

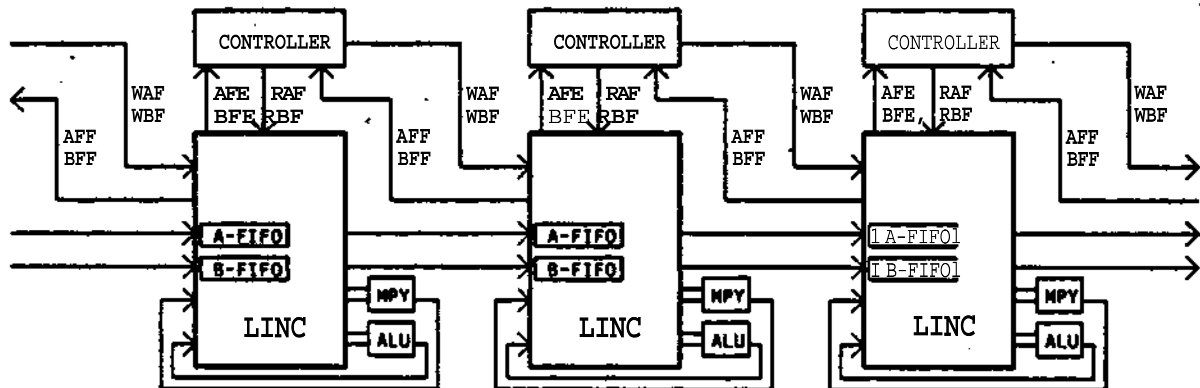


Figure 5. Use of FIFO *control and status signals in cooperating multiple systems*

CONTROL PATTERN ADDRESS:

The crossbar and pipeline register file of UNC may use a new control pattern every cycle. Since a control pattern has 64 bits, it is infeasible to input all these bits to UNC every cycle for a chip of 100 pins. Instead, a 5-bit address, CAtfM], is sent to the chip every cycle, to fetch one of the 32 control patterns pre-stored in one of the two banks of the control pattern memory of the chip.

LOADING AND TESTING:

The control pattern memory has two banks, with 32 words each, so that while one bank is in use the other bank can be loaded with new patterns via pins CCtO-7]. The control register for the FIFOs/programmable

delays also can be loaded with new contents via pins CC[0-7]. The control pattern memory and the control register should be loaded before LINC starts running. Through the mode control pins, LINC can be configured to test the control pattern memory, the FIFO/programmable delay controller, and the datapath.

3. DATAPATH AND CONTROL

A functional block diagram of LINC is shown in Figure 6. In the following we discuss the main functional features.

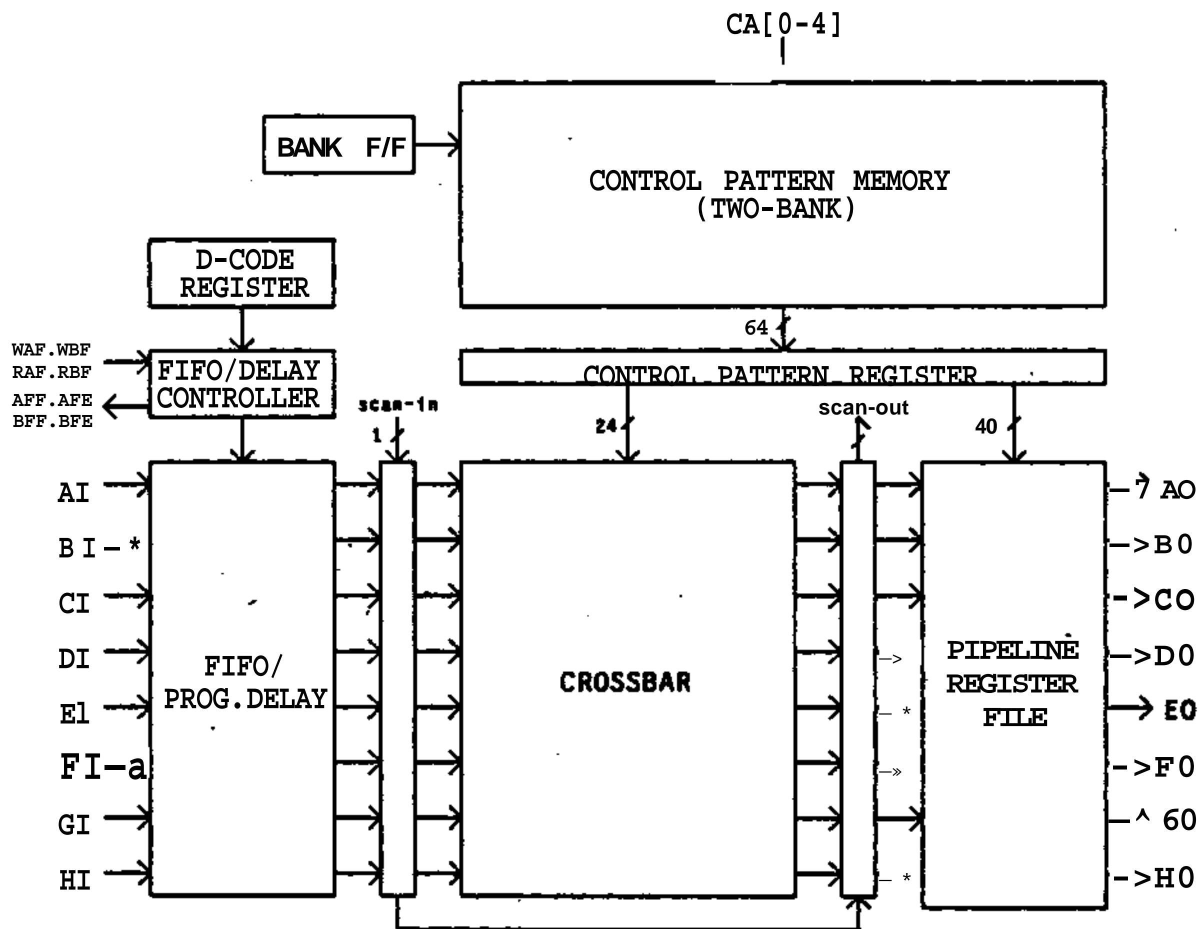


Figure 6. LINC *datapath block diagram*

3.1. Datapath

The datapath operates with a 100 ns minimum cycle time. Every I/O port is capable of performing one data transfer per cycle. This gives a 40 Mbytes/sec input transfer rate with all of the eight 4-bit input ports active, and a 40 Mbytes/sec total output transfer rate for the eight 4-bit output ports. The datapath consists of eight FPDs (FIFO and/or programmable delay), an 8x8 4-bit wide crossbar, and eight pipeline register files. There is a minimum delay of 2 clock-cycles before an input can appear at an output port The control flow from the control pattern register is matched to the data flow so that the entire transfer of each data item is controlled by the pattern in the control pattern register when the transfer starts, even though the transfer

actually takes 2 clock cycles. The net effect of this is that the programmer can view the chip as having zero internal delay across the FPDs, the crossbar, and the pipeline register files, but having a 2-cyclc delay at the output ports.

FIFOs and/or programmable delays

The eight FPDs can be configured into 0, 1, or 2 FIFOs and/or 0 to 8 4-bit wide programmable delays. The FIFOs' widths can be set by the programmer in 4-bit increments, but the total width of the FIFOs cannot be more than 32-bits. Each FPD can be selected to be part of one FIFO or as a programmable delay. For example, LINC can be configured to have two 8-bit wide FIFOs and four 4-bit wide programmable delays. The programmable delay time can be from 0 to 31 cycles. The two FIFOs, each 31 deep, are controlled by the off-chip signals WAF, WBF, RAF, and RBF. Notice that the outputs of the FPDs, even when configured as FIFOs, go to the on-chip crossbar rather than off-chip directly. Because there are logic delays inside LINC in sending out FIFO status signals (AFF/AFE and BFF/BFE), these signals refer to an "almost full" or "almost empty" status, "Almost full" means that the FIFO has at most two empty slots left, and "almost empty" means that the FIFO holds at most two valid items.* The configuration of the FPDs is determined by a 64-bit wide control register, called the d-code register (delay code register), which is loaded before system execution begins and normally does not change very often thereafter.

Crossbar

An 8x8 4-bit wide crossbar connects the FPDs to die pipeline register files. The crossbar is uni-directionaL The inputs come from the FPDs and the outputs go to the pipeline register files. The control for the crossbar comes from a 24-bit field in the control pattern register. Each output port of the crossbar, controlled by 3 bits, can accept any of the eight input values.

Pipeline register files

At the output of the crossbar are eight 4-bit wide pipeline register files (PRFs). Each PRF has 14 stages, and uses one bit in the control pattern register to decide whether to shift in the current crossbar output The output of each PRF, which also serves as one of the output ports, is specified by a 4-bit field in the control pattern register to be either one of the 14 stages, the crossbar output, or high-impedance.

32. Control

The functional description of the control circuit in this subsection refers only to the control of the operational states. For description of the loading/testing control, see the next section.

Control pattern memory and control pattern register

---

*A caution to the programmer when the controller of a system receives the almost empty status signal of a FIFO, the FIFO could still be holding up to two valid data items. At the termination of a computation, one must make sure that no valid data are still left in a FIFO. One method to achieve this is to write two dummy data items to the FIFO while the rest of UNC is halted. The idea is that these two dummy data will "push out" any valid data that might still be left in the FIFO. Application independent code can be written to deal with this "termination problem.**

The control pattern memory and the control pattern register determine the operation of the crossbar and the pipeline register files. The control pattern memory, a 64×64 static RAM, is configured as two 32-word banks. One bank can be loaded with new control patterns while the other is controlling the data flow. A 5-bit off-chip address, CA[0-4], selects the control pattern from the bank currently controlling the chip. The control pattern register holds the pattern addressed in the last cycle and controls the circuit behavior at the current cycle. (Actually, this is not quite true. Because of the requirement to match the control flow with the data flow, part of the control pattern that controls PRF will still be active on the next cycle.)

FPD control

Each FPD is controlled by an 8-bit field in the d-code register. Two bits are used to decide whether the FPD is a programmable delay, or part of the A-FIFO or B-FIFO. The other five bits are a count field. For a programmable delay, the count specifies the fixed length of the delay, and is set when loading the d-code register. For a FIFO, the count contains the current length of the FIFO. When loading the d-code register, it is necessary to set the count field of the register to zero, effectively creating an empty FIFO. A new d-code can be loaded into a shift register, whose contents can then be transferred to the d-code register in one cycle. The shift register is byte-wide and can be loaded through the CC bus in 8 cycles. The loading of the shift register can be performed when the chip is running or is halted.

External control

The run/~halt pin controls the running state of the chip. When the run pin becomes inactive, all the circuits, except the FIFO circuits and loading/testing circuits, become inactive. After the reset pin is active for more than one cycle, the eight data output ports become high impedance. This is useful in avoiding spurious outputs when loading the control codes into a LINC chip for the first time.

4. LOADING AND TESTING

Warning: First time readers should skip this section. In this section, *c-code* means *controlpattern, d-code* means *delay code, c-addr* means *controlpattern address* and *s-data* means *scan data.*

The mode control table shown below defines the various loading and testing modes. Only when the chip select pin is active do the CC pins and internal loading and testing logic become active. In the following we discuss the various loading and testing processes.

MODE	C-CODE	D-CODE	S-DATA	C-ADDR Counter	Bank F/F	comment
0000	shift	-	-	-	•	c-code in
0001	shift	-	-	-	-	c-code out
0010	load	-	-	-	-	to c-code REG
0011	unload	-	-	-	-	from c-code REG
0100	store	-	-	post inc	-	to c-code MEM
0101	read	-	-	post inc	-	from c-code MEM
0110	-	-	-	-	toggle	swap memory banks
0111	-	-	-	reset	-	addr cntr
1000	-	shift	-	-	-	delay code in
1001	-	shift	-	-	-	delay code out
1010	-	load	-	-	-	to d-code REG
1011	-	unload	-	-	-	from d-code REG
1100	-	-	shift	-	-	data scan in
1101	-	-	shift	-	-	data scan out
1110	-	load	-	-	toggle	load and swap

4.1. Control Pattern Loading

Conceptually, the control pattern memory can be considered as two swappable memory banks, one working bank and one loading bank. New control patterns arc always loaded into the loading bank. The loading process can be performed either while the chip is running or when the chip is halted.

Typically, a control pattern loading process starts by resetting the loading address counter to zero (mode 0111). The pattern words are then written sequentially into the loading bank. It is not necessary to fill the entire bank, thus saving time in loading a new set of control patterns. Users are allowed to skip some words by using the rcad-and-increment mode (0101). Each 64-bit pattern word is written by shifting in the new pattern in eight cycles through the 8-bit CC bus, using mode 0000, and then "executing" a store-and-increment mode (0100). The swap mode (0110) is used to swap the loading bank and the working bank.

4.2. FIFO/Delay Control Loading

The d-code loading shift register can be loaded with a new d-code through the CC bus in 8 cycles using mode 1000. Mode 1010 is then used to load the value in the shift register into the d-code register in one cycle. It is possible to simultaneously load the d-code register and swap the control memory banks with mode 1110 (load and swap).

The internal pipelining of the chip implies thaL the swap mode (0110) takes one more cycle than the

d-code loading mode (1010) to become effective after the modes are executed. Therefore the swap mode should be issued a cycle earlier than the d-code loading mode if the programmer wants them to be effective on the same data.

4.3. Control Pattern Memory Logic Testing

The testing of the control store normally starts with the control store loading shift register. A test code pattern is shifted into the shift register with mode 0000. Then the test code pattern is shifted out of the shift register with mode 0001 to verify the functionality of the shift register. The c-code memory (the current loading bank only) can be tested by first loading in test patterns and then resetting the counter to 0 (mode 0111) and using read mode (0101) and shiftout mode (0001) to read the patterns. To test the other bank, mode 0110, swap-memory-banks, can be used to swap the banks. To test the functionality of pattern addressing logic, a control address can be set with the CA pins and the contents of the corresponding memory location are read into the control pattern register, which can then be unloaded in the next cycle into the shift register (mode 0011) and shifted out (mode 0001) for verification. The functionality of the control pattern register itself can be tested by first loading from the shift register using mode 0010 and then unloading with mode 0011.

4.4. FIFO/Delay Control Testing

The delay code loading logic uses a different shift register from the one used by the c-code loading logic. The delay code shift register can be tested by writing with mode 1000 and then reading out with mode 1001. The current d-code register can be read by using mode 1011 to unload the d-code into the shift register and then reading the shift register with mode 1001.

4.5. Datapath Testing

In addition to the input and output data ports for indirect observation, the internal datapath can be examined with the scan path built around the crossbar. The 32-bit input into the crossbar and the 32-bit output from the crossbar are placed in a scan-in-scan-out path. The scan path can be set by shifting data in using mode 1100. Mode 1101 can be used to read out the current content of the scan path. Notice the scan out process is a destructive read and should NOT be performed while the chip is running. Also, the scan in process should not be performed while the chip is running. The scan path is only one bit wide, and only pin CC[0] is used for scan data I/O. Thus, loading or unloading the scan path will take 64 cycles to complete.

It is important to point out that when data are scanned out via pin CC[0], the scan path shifter behaves as a rotator. That is, after 64 cycles all the data will return to their original positions, although a copy of the data has been read out. At this point, the chip has returned to its original state, and is ready to resume operation.

5. TIMING SPECIFICATIONS

The first implementation of LINC uses an external two-phase clock to provide clocking flexibility after fabrication. For future implementations it would be possible to use an external single-phase clock and put a two-phase clock generator on the chip.

Figure 7 shows the relationship between the two-phase clock and the input/output signals. Since LINC uses dynamic circuits internally, there exists a minimum clock frequency. The target maximum clock rate is at least 10 MHz.

PHI1
PHT2
R/~H
CS.MC
CA
AI-HI
WAF.WBF
RAF.RBF
.CCfwrite)
tc- 1LOOns
ts
tH
t m
tflO2 NI
toTD
[AQ-HQ
iCC(pead)

Figure 7. Input/output timing diagram

The timing diagram for the running state is depicted in Figure 8. To read a control pattern from the control pattern memory, the control pattern address (CA[0-4]) must be supplied one clock cycle earlier. It takes two cycles for input data to go through the internal circuit being controlled by the control pattern and come out at the output ports. The input timing for die FIFO control signals is the same as for the data input, and FIFO status signals appear *two* clock cycles later.

The timing for loading/testing the control pattern, FIFO/delay control code, and internal data is shown in Figure 9. The control pattern and FIFO/delay control codes are written/read eight bits (CQ0-7D at a time. Therefore it takes eight cycles to read/write one 64-bit word. To access a 64-bit internal data item through the bit-serial scan line, on the other hand, takes 64 cycles, since only one bit is shifted in/out through the CQO] pin every cycle. Pins CCJO-7] output control patterns, FIFO/delay control code or internal data only when the CS pin is active and the MC pins indicate reading. Otherwise, CQO-7] become high-impedance.

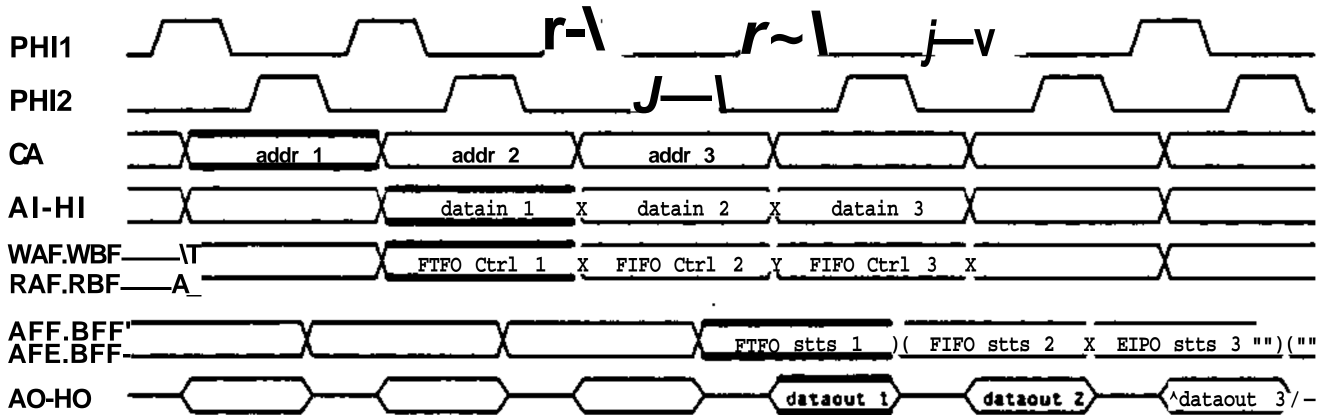


Figure 8. Timing diagram for running slate

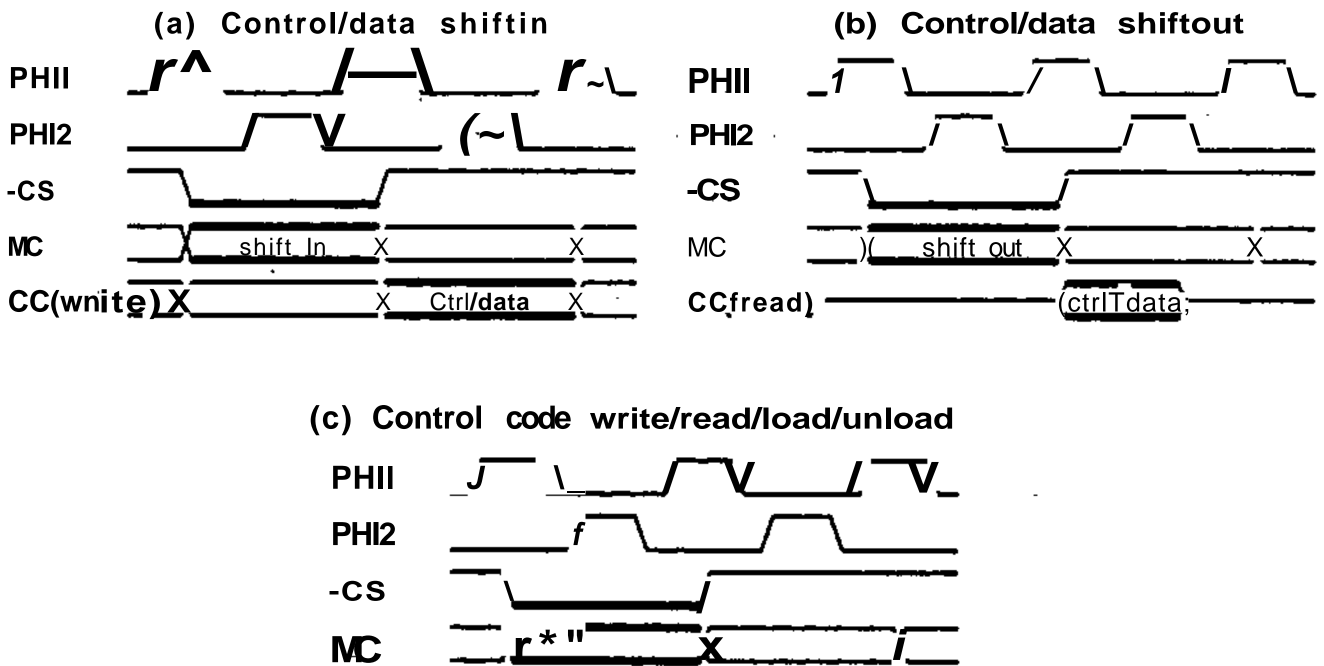
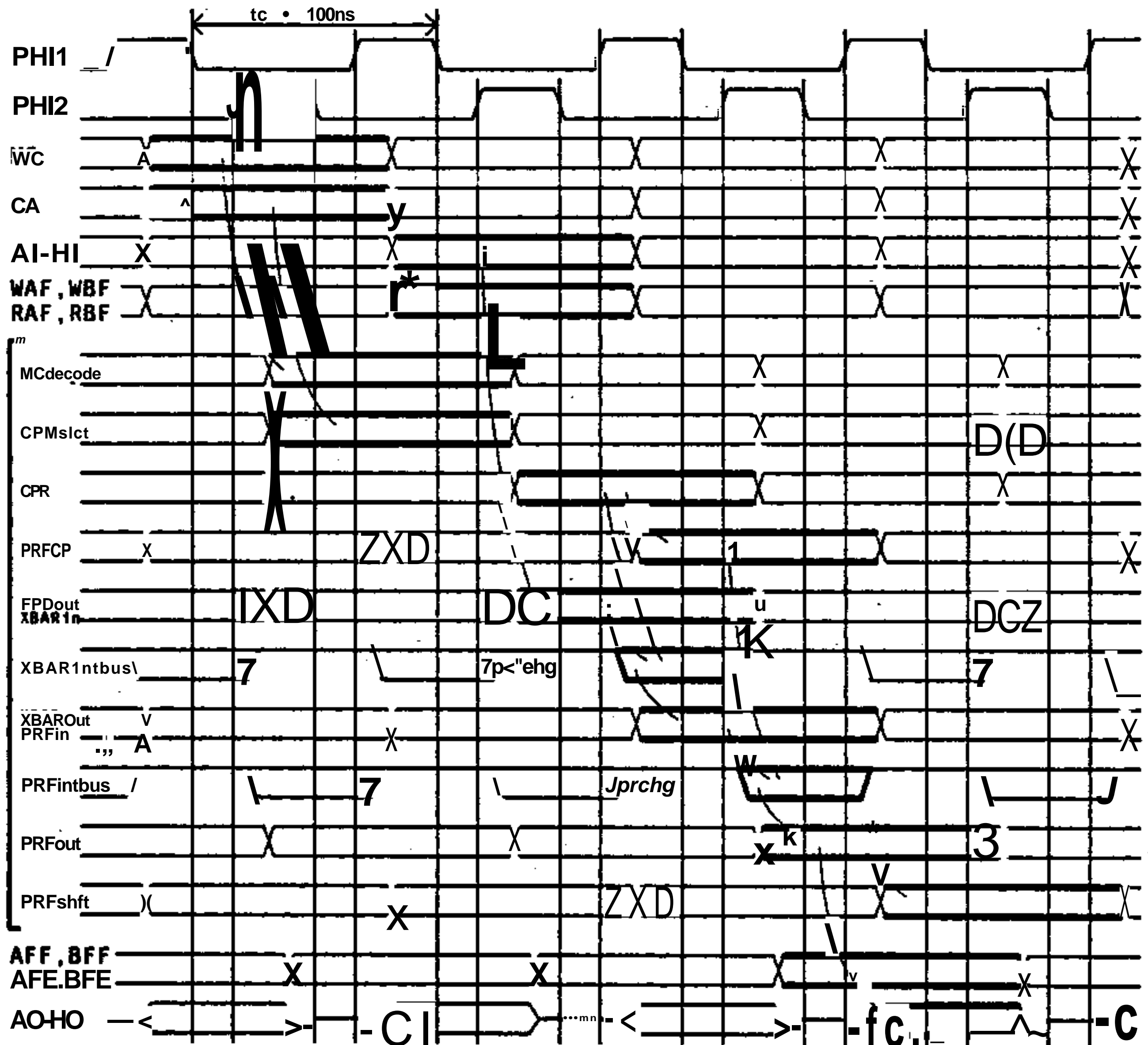


Figure 9. Timing diagram for loading and testing

Figure 10 depicts the internal timing diagram for the running state. LINC contains four (approximately 50 ns) stages, each requiring one half clock cycle to execute, which accounts for the two cycle minimum input to output delay stated in the datapath description.



MCdtcodt - Nodt Control dtcodt
CPUslet * Control Patttrn Mcoory stitct
CPR - Control Patttrn Rtyistcr
PRFCP - Plpelln« Register Fllt Control Patttrn
FPOout/XBARIn - FIFO/Prog. Dtlay out, XBAR In

XBARIntbus - XBAR Internal bus
XBARout/PRFln - XBAR out. Piptiint Register Flit ln
PRFintbus - PRF Internal bus
PRFout - PRF output
PRFshft - PRF shift

Figure 10. Internal timing diagram

6. APPLICATION EXAMPLES

6.1. Corner Turning

Functional units of a system often have different data input and output formats. As a result, outputs of a unit may have to be reassembled before they can be used as inputs to another unit. A reassembling operation that occurs frequently is known as "corner turning." We show how corner turning can be efficiently carried out with LINC.

The corner turning operation is like matrix transposition. That is, given an input matrix, say in column-major ordering, we want to transpose it so that the output will be in row-major ordering. This definition is illustrated by Figure 11 (a). For example, corner turning operation is needed in preparing input data for some systolic arrays [1], and in packing bytes into words and unpacking words into bytes.

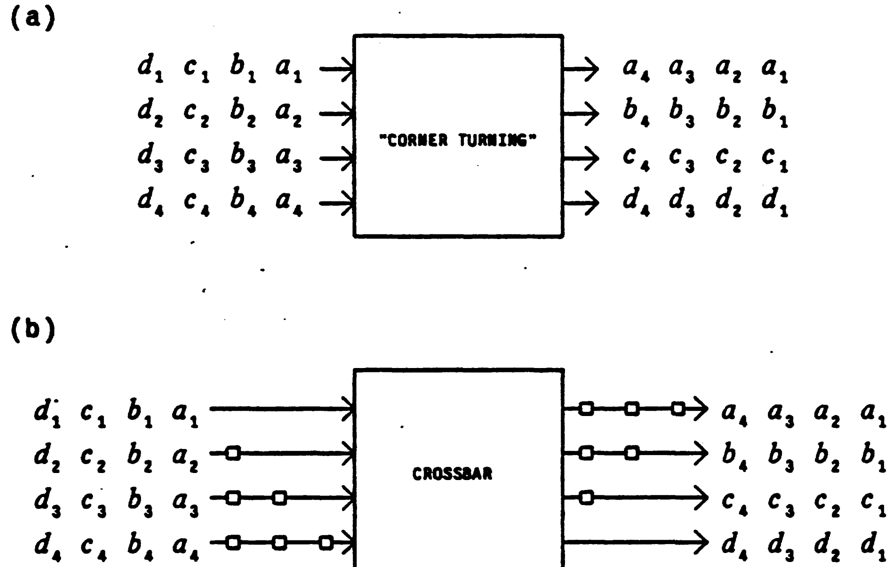


Figure 11. (a) "Corner turning", and (b) its implementation

Referring to Figure 11 (a) we see for example that inputs a_1, a_2, a_3 and a_4 all arrive at the same time, but they must be buffered so that they can be output from the same output port serially. It is easy to see from Figure 11 (b) that by providing buffer delays both before and after a crossbar, the corner turning can be accomplished. Since LINC does provide these buffering facilities and the crossbar, it can implement the corner turning operation.

For transposing large matrices, we can use multiple copies of LINC, as depicted by Figure 3 (b) and Figure 4. Alternatively, we can multiplex in time a single copy of LINC. For example, Figure 12 shows that we can transpose an 8×4 matrix by transposing the first four rows of the matrix followed by the remaining four rows. Of course, if the size of the matrix exceeds that of the programmable delays and pipeline registers

of LINC, additional buffers outside LINC must be used.

Figure 12. Transposing a large matrix by multiplexing LINC

As its datapath indicates, LINC can buffer inputs at its input ports, and send them through the crossbar to any of the output ports, at which data can again be buffered by the pipeline register files. These features seem to be general and powerful. Corner turning is just one example of many data shuffling operations that LINC can efficiently implement

6.2. Systolic Array Implementation

CMU is currently building a programmable systolic array processor that can efficiently perform many essential computations in signal processing, such as the FFT and convolution. As depicted in Figure 13 this is a one-dimensional systolic array that takes inputs from one end cell and produces outputs at the other end, with data and control all flowing in one direction. We call this particular systolic array a Warp processor, suggesting that it can perform various transformations at very high speed [5,6].

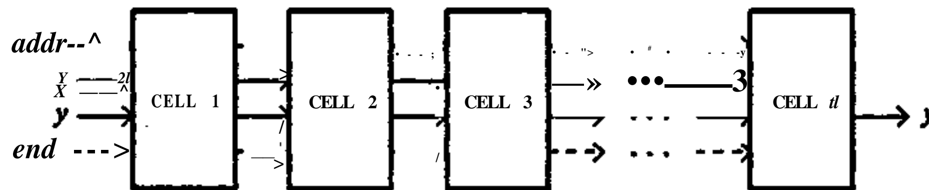


Figure 13. The Warp *processor*

Each cell of the Warp processor uses a pipelined 32-bit floating-point chip set from Weitek [10] that can perform 10 million floating-point operations per second (MFLOPS). A 10-cell Warp processor can process 1024-point complex FFTs at a rate of one FFT every 600 *fis*. Under program control, the same processor can perform many other primitive computations in signal, image and vision processing, including two-dimensional convolution and complex matrix multiplication, at a rate of 100 MFLOPS. Together with another processor capable of performing divisions and square roots, the Warp processor can also efficiently carry out a number of difficult matrix operations such as solving covariant linear systems, a crucial computation in real-time adaptive signal processing.

Figure 14 summarizes the datapath of each cell of the Warp processor. For the CMU prototype that is being built, we use only off-the-shelf components, and each cell is implemented on one board. We note that

all the components inside the region surrounded by the dotted lines are "glue chips" and they can be implemented efficiently with LINC. In particular if LINC is used, a board of the same size will be able to host three or more Warp cells.

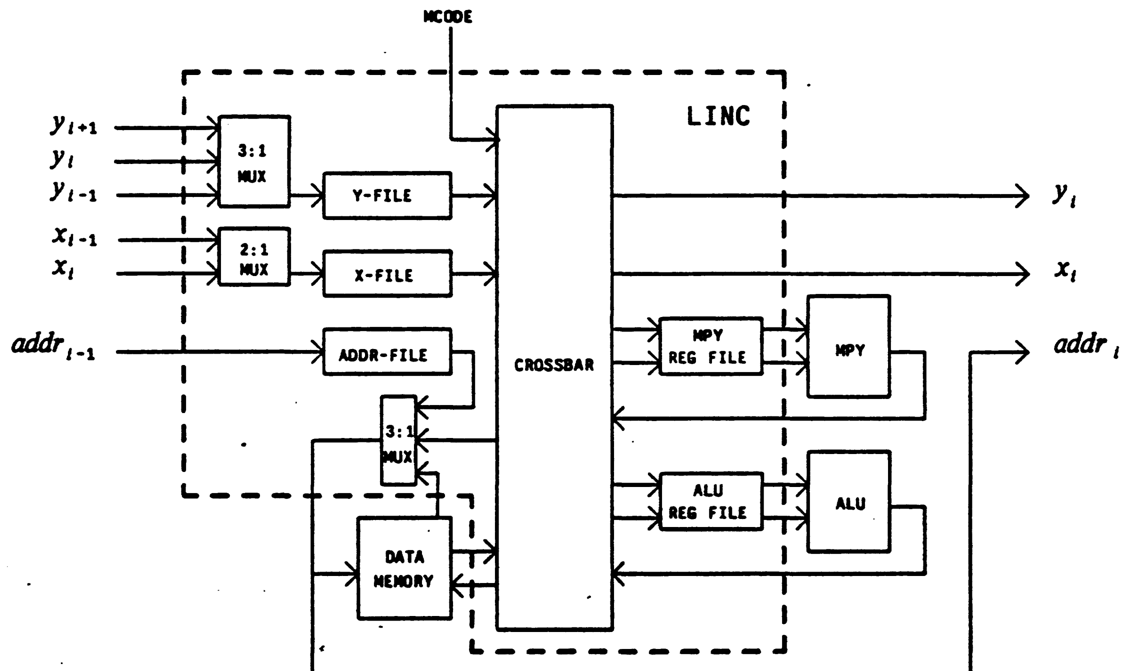


Figure 14. Warp *cell implemented with* LINC

In general, through LINC, processors and memories can be linked together to form various processor arrays, as illustrated in Figure 15.

6.3. Fast Fourier Transform

We describe how an n-point fast Fourier transform (FFT) can be carried out on a processor implemented with LINC, based on a scheme originally proposed for the Warp processor [5].

The FFT uses $\log_2 n$ stages of $n/2$ butterfly operations, with data shuffling between stages. The so-called constant geometry version of the FFT allows the same pattern of data shuffling between all stages [9].

In the Warp processor array, the butterfly operations for the i-th stage of the FFT are executed by cell i, and the results are stored to the data memory of cell $i+1$. The data memory of each cell is double buffered, so that cell i can write into the data memory of cell $i+1$ while cell $i+1$ is working on stage $i+1$ of another FFT problem. In this way, if many FFTs are to be performed, all cells in the array can be kept occupied at all times.

Figure 16 shows the cell block diagram, with LINC controlling all data flow to the processing units.

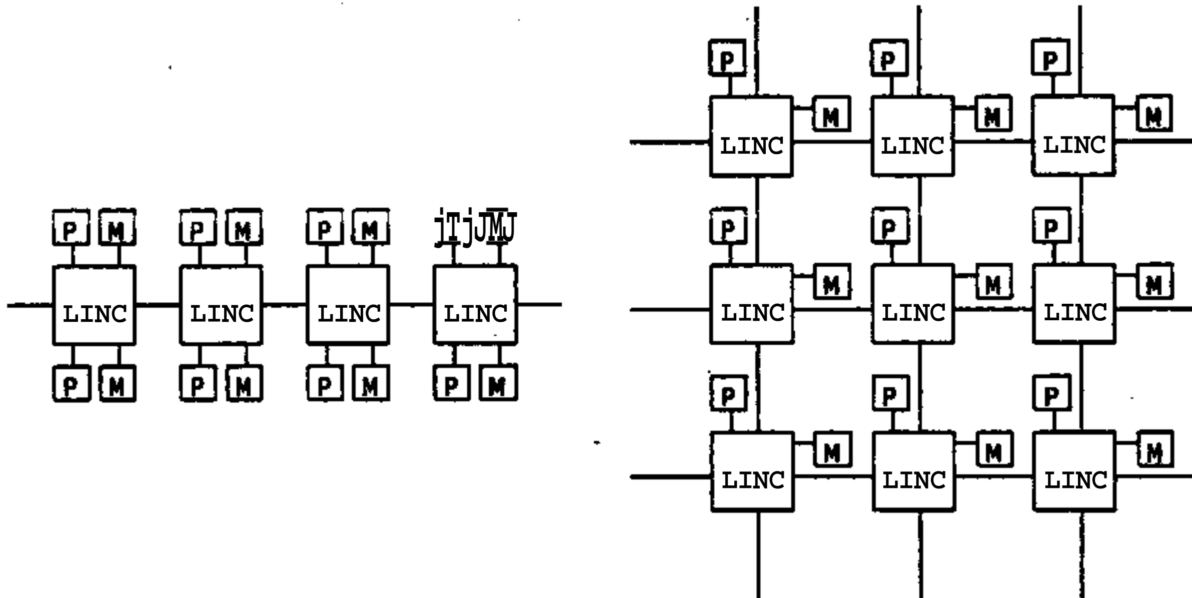


Figure 15. Processor arrays implemented with LJNC

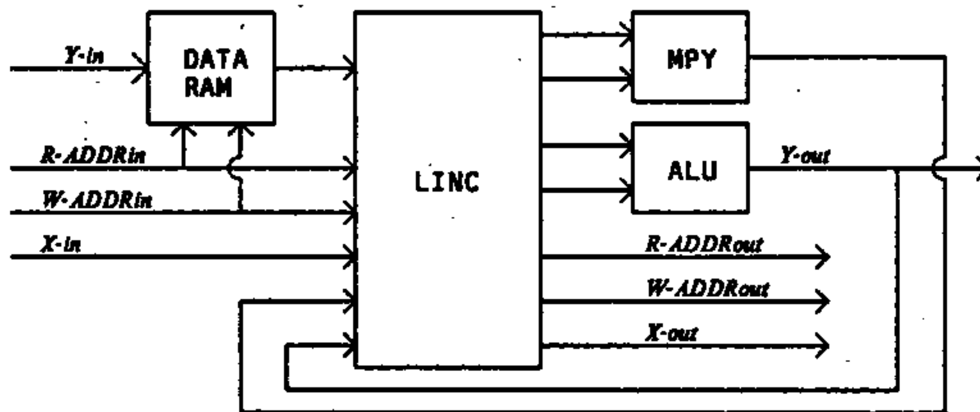


Figure 16. Cell of constant geometry FFT *processor using* LJNC

A butterfly operation is defined as:

$$(a_r +$$
$$= [or \pm (b_r - \text{HV} - b_v\, wfl + j[a_t \pm \{b_f\, w, + b_i \cdot w_r)\}]$$

requiring four real multiplications and six real additions. Using LJNC to control the data flow, it is possible

to fully occupy the ALU so that it actually takes the minimum of six cycles to do a butterfly. This is accomplished by interleaving up to four different butterfly operations in all cells at all times. L1NC solves the problem of reusing the inputs to each butterfly twice, by storing such inputs inside LINC and reading them out at the appropriate times.

We now define the data streams for LINC and for the cell. The cell >>*data stream provides the input for the data memory and is written into the address provided by the *w-addr* stream. The *r-addr* stream provides the read address for the data memory, and the data memory provides the butterfly inputs $(a_{r\%}\ a_{l\%}\ b_r,\ bj)$ for LINC. The weight inputs for the butterfly operations (w_r, w_t) arc provided by the x-data stream. There are also feedback loops in the cell from the outputs of the ALU and multiplier back to the inputs of LINC. This allows LINC to control the correct accumulation of the results of the butterfly. The cell ^output is the output from the ALU, while the rest of the output streams are simply the corresponding input streams, delayed by two cycles (the pipeline delay of LINC). All data streams pass systolically from cell to cell, including addresses for the data memory. This is a good reason to implement the FFT in the constant geometry version, because the method uses the same data shuffling between *all* cells.

Programming LINC for this application is not a difficult task. As we have already noted, data shuffling is the same between all cells, which means that all cells require the same LINC control. A total of eight different control patterns are used, six for the main loop of the FFT and two more to allow for the case of shifting in new weights for the current butterfly operation. The decision on whether to shift in new weights is made by the cell controller, based on the method described in [5]. The general method by which LINC controls the data for a butterfly operation is quite simple. First it buffers $6_r$and b_t in one pipeline register file connected to the multiplier, and the corresponding weights in the other pipeline register file connected to the multiplier. Then, as the multiplier computes the products they are sent back through LINC and buffered in the pipeline register files connected to the ALU. As soon as both operands are available, they are sent to the ALU to continue the butterfly operation. Once the ALU finishes this set of computations, the results are sent back through LINC once again, buffered as inputs to the ALU, and used as operands with a_r and a_t (which are also buffered), to produce the final cell outputs. The important point to notice is that LINC provides the flexibility necessary to regulate several data streams concurrently, without inserting unnecessary empty cycles into the pipelines of the processing units. It is also important to note that up to four different butterfly operations are going on simultaneously in one cell, but that LINC can handle all the necessary control with only *eight* distinct control patterns.

6.4. Robot Ann Control

The problem of controlling a robot manipulator can best be described as a problem in transforming the easily specified desired Cartesian (world) coordinates into the arm's joint coordinates. These transformations are defined by a set of homogeneous transformations, each of which is a 4x4 matrix which when applied to a coordinate's 4-vector (jy,z, and scale) transforms it into another 4-vector [7]- The standard transformations include translation, rotation, stretching and scaling. Since the major computational problem involved is that of matrix multiplication (composing the necessary homogeneous transformations to tranfonn Cartesian to joint coordinates), a systolic array to perform matrix multiplication could provide the necessary computational power for robot arm control

A method for doing matrix multiplication with linear systolic arrays using pipelined arithmetic and multiplier units is described in [5]. The systolic cell requires buffering for both data streams associated with the matrices being multiplied, and also for the result data stream. The cell would look exactly like Figure 16, with the only difference between such a cell and an FFT cell being the programming of LINC and the cell controller. The matrices we are discussing are 4x4, so a four-cell systolic array could process a single matrix

multiplication in exactly sixteen cycles, since we know that by interleaving independent matrix multiplications a new task can enter each adder every cycle. This implies that, with a 200 ns arithmetic unit cycle time, it would take 3.2 /is to do a single matrix multiply.

Two robot arm problems that illustrate the necessity for the powerful processing elements we describe above arc that of generating the robot arm joint position error and joint velocity set points. The equations for solving these problems involve composing many homogeneous transformations and computing inverse homogeneous transformations, both of which require many scalar multiplications (to compute matrix products and inner products, respectively). The equations needed to solve these problems [8] are quite complex, but what is more important is calculating the necessary requirements on computing the solutions. One part of the solution derivation requires a computation of approximately 2000 multiplications (scalar, not matrix) to do the necessary matrix multiplications. The time delay in computing the joint servo of a typical robot arm is important in determining the computation rate, and requires a delay of less than 250 /is. This is well within the capability of our systolic array, which can do the 2000 multiplications in (1/4)-(2000>200ns(=100/is). Then the evaluations of the two set points can be done, requiring approximately another 1000 multiplications within the same 2S0 /is delay, which is also easily met by the systolic array's capabilities.

Robot arm control is clearly amenable to applying the computational power of a systolic array based on LINC. It seems that die task requires too much computation in too short a period of time for a conventional architecture to be able to handle it effectively, and any other method of solving the problem requires approximation algorithms that are not completely adequate for the task. As a measure of how cost-effective we expect LINC to be in doing this task, we know that a Warp processor board can handle the computations described above, and such a board would require only 45 chips using LINC. The flexibility available in programming LINC to control data flow and the computational power of a systolic array can be a major help in alleviating the computational difficulties of robot manipulator control.

6.5.3-D Computer Graphics

Hardware for high performance three-dimensional computer graphics can be viewed as consisting of two main parts: a geometry system and a display system. The geometry system transforms object descriptions in world coordinates into descriptions in normalized device coordinates. The display system eliminates hidden surfaces and outputs each pixel to a raster-scan display. There have been several attempts to apply VLSI technology to both systems [3,4]. In comparing the two systems, the geometry system requires many floating-point operations and is appropriate for implementation with LINC and floating-point processors, whereas the display system needs sorting operations of fixed-point values. Although LINC could be useful in implementing the display system we will not explore that idea at the present time. We will now discuss further die architecture of the geometry system.

There are three tasks in the geometry system:

- *Matrix multiplication*
  For the transformation from world to normalized device coordinates, we need to perform homogeneous transformations defined by 4x4 matrices. The computation involves many matrix-vector multiplications.

- *Clipping*
  The transformed data in normalized device coordinates are clipped into the space which a viewer

can see through a virtual window. In world coordinates, this space corresponds to a truncated viewing pyramid with "front" and "back" planes.

•> *Scaling*
This step transforms the data in a homogeneous coordinate system *[x, y,* z, *w* (where *w* is a scaling factor) resulting from clipping, into another coordinate system *[x'ₜy'%* z⁷,1].

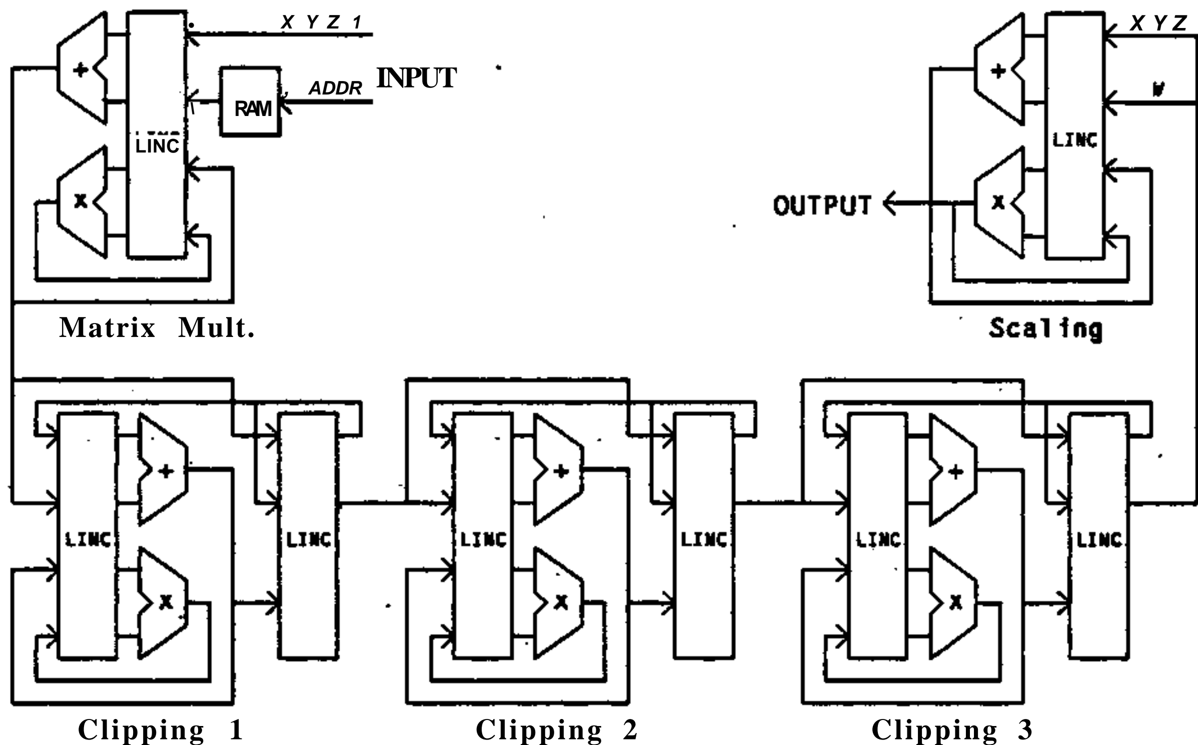


Figure 17. Implementation of the geometry system

Figure 17 shows the implementation of the geometry system. The first cell performs the matrix multiplications, the next three cells do dipping, and the last cell does the scaling operation. Each eel! is composed of copies of LINC, a floating-point multiplier and a floating-point ALU. (In the figure, one LINC box represents two copies of LINC).

The matrix multiplication cell has a random-access memory which stores the coefficients of the homogeneous transformations. The cell inputs memory addresses and vertex vectors in world coordinates [JC, *y,* z, 1]. The transformed coordinates of a vertex vector are output every 32 LINC cycles.

Given an input point, each of the clipping cells clips every edge of a polygon by two planes that are parallel in normalized device coordinates, such as *x*— w=0 and *x*+ w=0. The first part of a clipping cell computes the two intersections of an infinite line with the two planes. The second part selects at most two

points from the input point and the two intersection points. These selected points correspond to the endpoints of the edge of the polygon between the planes. Notice that data may be output at different rates. That is, even though the data output rate from the matrix multiplier cell is constant (1 vector per 32 LING cycles), each clipping cell has the possibility of outputting no points, 1 point or 2 points. The FIFOs, in LJNC are useful for smoothing this variance in I/O rates between cells. In particular the LINC at the inpiit of each clipping cell provides a FIFO to buffer inputs which may arrive in bursts.

The scaling cell performs divisions to compute *[x/w, y/w, z/w[.* This can be efficiently implemented, for example, by a custom chip being designed at CMU that is capable of computing reciprocals of 32-bit floating-point numbers at the rate of one every 200 ns. However, if no special hardware for divisions is assumed, we can still use existing schemes that can replace a quotient computation (no remainder) with a few multiplications [2]. In particular, for 32-bit floating point numbers with a 24-bit mantissa, computing the inverse of *w* takes no more than eighteen LINC cycles. With the addition of the six LINC cycles required for multiplying x_t *y* and *z* by the inverse of w, we need no more than a total of 24 LINC cycles to scale one vertex vector.

Each cell in Figure 17 is capable of computing one result vector in 32 LINC cycles *(32* /is). Therefore the maximum throughput of this system is approximately 9K vertices in one frame period (30 ms).

REFERENCES

[1] Bojanczyk, A., Brent, R.P. and Kung, H.T.
Numerically Stable Solution of Dense Systems of Linear Equations Using Mesh-Connected Processors.
SIAM Journal on Scientific and Statistical Computing 5(1):95-104, March, 1984.

[2] Cavanagh, J.J.F.
Digital Computer Arithmetic: Design and Implementation.
McGraw-Hill, New York, 1984.

[3] Clark, J.H.
The Geometry Engine: A VLSI Geometry System for Graphics.
Computer Graphics 16(3):127-133, July, 1982.

[4] Fuchs, H. and Poulton, J.
Pixel-Planes: A VLSI-Oriented Design for a Raster Graphics Engine.
VLSI Design 11(3):20-28, 1981.

[5] Kung, H.T.
Systolic Algorithms for the CMU Warp Processor.
In *Proceedings of the Seventh International Conference on Pattern Recognition*, pages 570-577. International Association for Pattern Recognition, 1984.

[6] Kung, H.T. and Menzilcioglu, O.
Warp: A Programmable Systolic Array Processor.
In *Proceedings of SPIE Symposium, Vol. 495, Real-Time Signal Processing VII.* Society of Photo-Optical Instrumentation Engineers, August, 1984.

[7] Paul, R.
Robot Manipulators: Mathematics, Programming, and Control.
The MIT Press, Cambridge, Massachusetts, 1981.

[8] Paul, R.P., Shimano, B. and Mayer, G.
Kinematic Control Equations for Simple Manipulators.
IEEE Transactions on Systems, Men and Cybernetics SMC-11:449-455, 1981.

[9] Rabiner, L.R. and Gold, B.
Theory and Application of Digital Signal Processing.
Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[10] Woo, B., Lin, L. and Ware, F.
A High-Speed 32 Bit IEEE Floating-Point Chip Set for Digital Signal Processing.
In *Proceedings of ICASSP 84*, pages 16.6.1-16.6.4. IEEE, 1984.