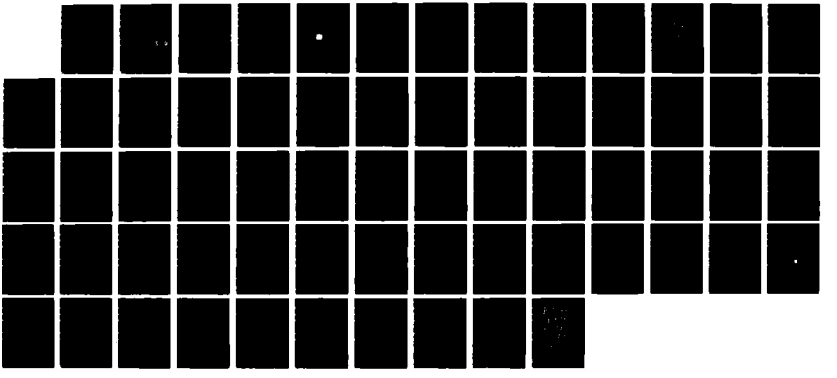
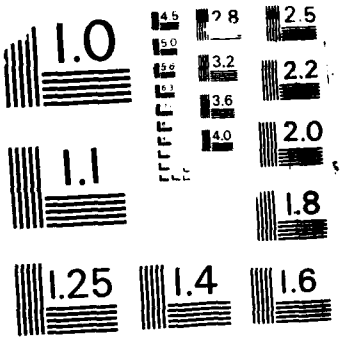
AD-A190 631 POKER 31: A PROGRAMMER'S REFERENCE GUIDE(U) WASHINGTON 1/1
UNIV SEATTLE DEPT OF COMPUTER SCIENCE L SNYDER SEP 85
TR-85-09-03 N00014-85-K-0328

UNCLASSIFIED F/G 12/5 NL





SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1 REPORT NUMBER: none	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4 TITLE (and Subtitle): Poker 3.1: A Programmer's Reference Guide		5. TYPE OF REPORT & PERIOD COVERED: Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Lawrence Snyder		8. CONTRACT OR GRANT NUMBER(s): N00014-85-K-0328
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Washington Department of Computer Science, FR-35 Seattle, Washington 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11 CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research Information Systems Program Arlington, VA 22217		12. REPORT DATE: September 1985 13. NUMBER OF PAGES: 61
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

Poker parallel programming environment

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This manual defines all facilities available within the Poker Parallel Programming Environment. In addition, there are two appendices, one describing the structure of the distributed library and the other describing systems related to Poker.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Poker 3.1: A Programmer's Reference Guide

Lawrence Snyder
Department of Computer Science, FR-35
niversity of Washington
Seattle, Washington 98195

TR 85-09-03

Abstract

This manual defines all facilities available within the Poker Parallel Programming Environment. In addition, there are two appendices, one describing the structure of the distributed library and the other describing systems related to Poker.

This document has been funded in part by the Office of Naval Research Contract No. N00014-85-K-0328 and the National Science Foundation Grant No. DCR-8416878.

Poker (3.1) Programmer's Reference Guide

Lawrence Snyder

This document gives a succinct description of the facilities available with the Poker Programming Environment. The emphasis is on "what is available" rather than "how to achieve particular results." Although the sections are self-contained, so that they may be referred to independently, there are a few things you should know:

- Poker uses interactive graphics. The graphics are described in Section 2; the interaction is described in Section 3.
- The usual programming language notion of a "source program" as a monolithic piece of symbolic text has been replaced in Poker by a database. The way to create, view, and change the database is described in Section 4.
- Object programs (the "compiled database") are emulated by Poker and snapshots of the execution can be continuously displayed. (See Section 14).
- Poker supports a variety of CHiP architectures; the current one can be displayed or changed using the CHiP Parameters facility, Section 7.
- The back page of this document gives a summary of the commands.
- Other versions of Poker exist; consult Appendix B for your particular system.

Figure 1 lists the contents of this document.

Figure 1. Table of Contents

1 First time access

To access Poker the user should include the directory "/usr/poker/bin" in his search path. This requires a (one-time) change to the PATH line of your .profile or .login file. The required modification is to append the text ":/usr/poker/bin" to the PATH line. [CAUTION: Your installation's protocol may differ.] Users of the Teletype 5620 display must type "setpoker", and see Appendix B.3 for device specific information.

First time users who would like a short computer-assisted overview of the Poker system should type "playpoker." This demonstration is intended to be a brief (less than ten minutes) and entertaining introduction to the major components of Poker.

To use Poker, type 'poker' from the bit-mapped display (see Section 2). The system will begin at the point where execution was previously terminated, or to the CHiP Parameters display (Section 7) if there was no previous session.

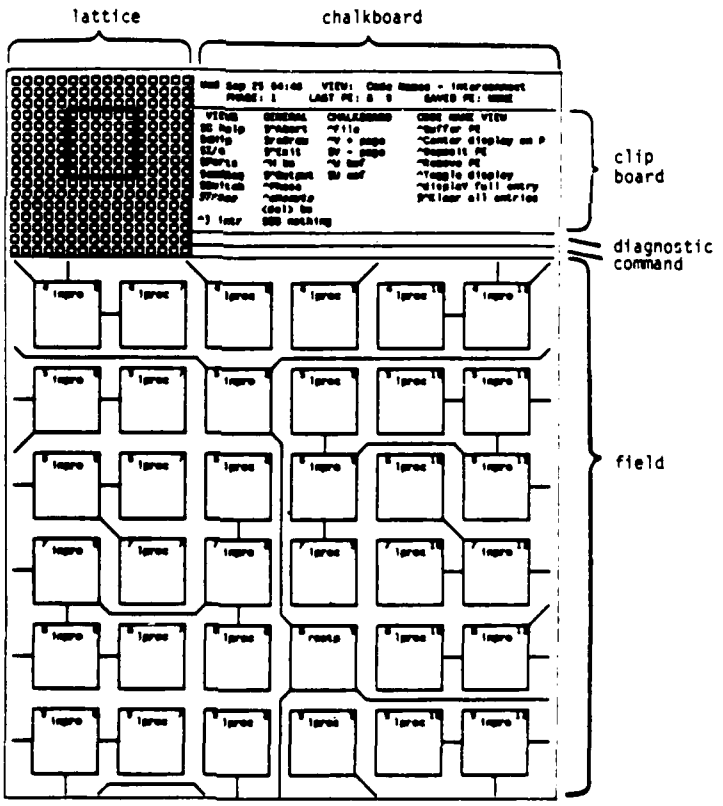


Figure 2. A typical Poker display.

2 The display

The Poker System uses two displays: a bitmapped display and a secondary terminal. [CAUTION: Your system may differ.] It is possible, though inconvenient, to use just the bitmapped display. The user should be logged into both terminals and should have both referring to a common directory. To avoid name conflicts, it is advised that the directory be empty (initially). The command 'poker' from the bitmapped display terminal causes the system to be entered (assuming the modifications of Section 1).

The display will have a form of the type shown in Figure 2 (see previous page). The regions of the display are as follows:

field — a region showing a schematic picture of the CHiP Computer's lattice; this is the region where most programming activity takes place,

lattice — a schematic diagram of the processing elements (PEs) with a box enclosing those PEs currently shown in the field; no direct user activity is available in the region,

chalkboard — the upper righthand region of the display giving status information,

command line — the area where textual commands are given; last line of the chalkboard,

diagnostic line — the area where error indications are given; the next-to-last line of the chalkboard,

clipboard — a ten line region of the chalkboard used for the display of transient information and available for displaying files,

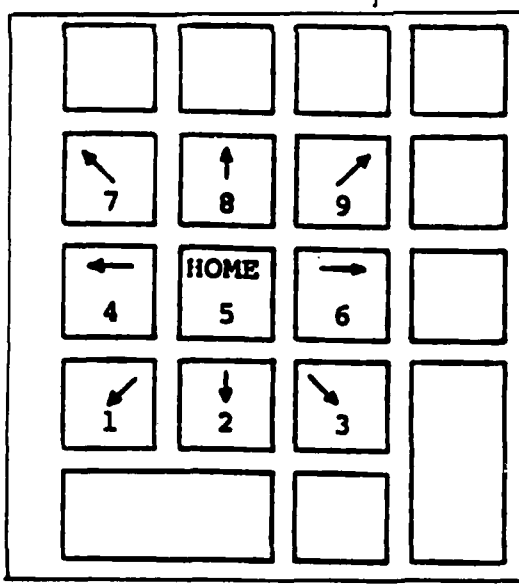
status — area displaying current state information; top two lines of the chalkboard.

The information shown in the status area is updated as follows. The time of day is current to the last write to the screen. The view and phase number are updated with the key stroke that changes them. Last PE is updated when the cursor visits a new PE; visiting switches does not change Last PE. Saved PE is set in Code Names and Port Names when the PE is buffered, and it is cleared when that PE is modified or the view changes. (See Sections 9 and 10.) Num Ticks (Command Request and Trace) is updated each time the emulator stops. (See Sections 13 and 14.)

3 Keys and Cursor Motions

The Poker system is interactive: *virtually all key strokes cause an immediate action.* Most actions are given by composite key strokes formed either by *depressing* the control key while striking a letter key (e.g., we write ^h to denote depressing the control key while striking a letter h (which causes the cursor to backspace)), or by first striking the escape key (written **$**) followed by another (possibly composite) key (e.g., **$**^e is the command to exit and return to UNIX). Should escape be inadvertently struck, it can be cleared by striking it twice more, i.e. **$$$** is a "no operation".

Movement around the display is controlled by the numeric keys of the key pad (located on the right side of the keyboard and illustrated in Figure 3). Two kinds of motions are provided: gross cursor motions and fine cursor motions. The gross cursor motions, which are two-key operations composed of an escape followed by a directional key, usually move to the next PE in the indicated direction. Fine motions, which are given just by a directional key, vary in meaning with the view being displayed. The "home" key is used to move back and forth between the command line and various positions in the field. [CAUTION: This paragraph is device-specific and your terminal may vary; see Appendix B for an explanation of differences, especially the mouse usage for the Teletype 5620 and the keypad limitations for the Sun I.]



Fine		Gross
%4	West	$% 4
%7	Northwest	$% 7
%8	North	$% 8
%9	Northeast	$% 9
%6	East	$% 6
%3	Southeast	$% 3
%2	South	$% 2
%1	Southwest	$% 1
%5	Home	$% 5

Figure 3. Cursor key bindings.

4 Views

The current state of the Poker system is stored as a database that can be queried, created and changed by using one of seven *views*. (A checklist of the information required to produce and run a Poker program is given in Figure 4.) The system begins with the view active at the end of the previous session, or if there was none, the CHiP Parameters view. The available views are:

$h cHip parameters: Specifies the architectural characteristics (e.g. number of processor elements) of the CHiP machine being programmed. (See Section 7.)

$s Switch settings: Specifies the graph of the communication structure to be used by the processor elements. (See Section 8.)

$c Code names: Specifies the name of and actual parameters of the process assigned to each processor element. (See Section 9.)

$p Port names: Specifies the symbolic names of a processor's communication channels. (See Section 10)

$i Input/output: Specifies the input and output data streams. (See Section 11.)

$r command Request: Converts the source information into object form for execution; the database is not actually displayed in this view. (See Section 13.)

$t Trace: Displays the current state of the traced variables of the executing program. (See Section 14.)

In addition, the processes themselves are defined using a standard editor on the secondary display. The processes are written in the XX programming language. (See Section 12.)

As noted before the database is the Poker "source program." It is composed of a sequence of phases; there is a switch settings, code names, port names and IO names specification for each phase. A phase corresponds roughly to a single algorithm that uses a single processor interconnection structure (switch settings). For example, an algorithm to multiply two matrices would likely be a phase. Phases are numbered, but they can also be given symbolic names. (See Section 7.)

The above composite keys are always recognized. An attempt to change to the current view (e.g. the use of **$s** from switch settings) results in the display of the legal commands for that view being given in the clipboard area of the chalkboard.

Programming Checklist

To write and run a Poker program one must:

1. Comprehend the algorithm
2. Specify the communication structure - use Switch Settings view
3. Specify processes (PE codes) - use XX language and standard editor
4. Assign processes to PEs with parameters - use Code Names view
5. Name communication ports - use Port Names view
6. Name data streams - use IO Names view
7. Create and format data files - use standard editor and, if needed, packIO
8. Bind stream names to file names - use Bind, an execute command
9. Convert source "program" to object form - use Command Request view
10. Run program and watch results - use Trace view

Notice that the order of steps 2 - 7 is arbitrary.

Figure 4. Programming Checklist

5 Global Commands

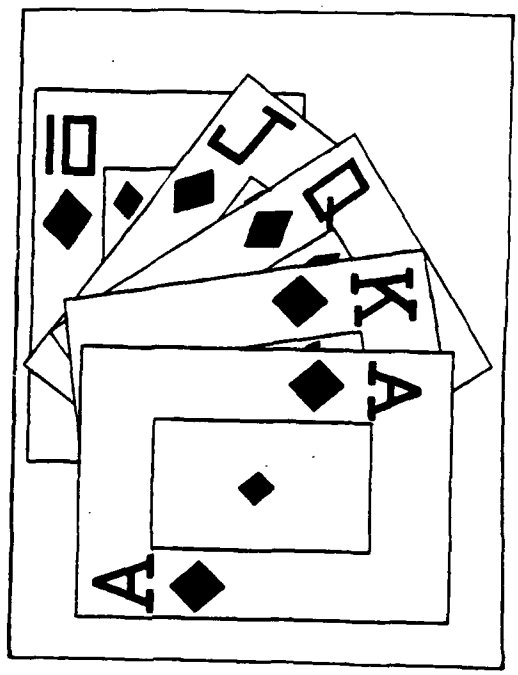
In addition to the view commands previously described (Section 4), the following commands are always recognized:

$^a	Abort	Return to UNIX without saving state. (See Interrupt key below.)
$^e	Exit	Return to UNIX and save the current source specifications (i.e. CHiPParams, SwitchSet, CodeNames, PortNames, IONames).(See Interrupt key below.)
$^o	Output	The bitmapped display's raster memory is dumped to a file named BG*XXXXXX* in the current directory, where *XXXXXX* is a random number. [CAUTION: Your system may vary.] As the bitmap is being uploaded, the screen is complemented; when the upload is finished, the screen is recomplimented and a message indicating completion is given. The file must be reformatted using a program called ulbg (See Appendix B) so as to be printable on the laser printer.
$d	reDraw	The screen is redisplayed.
^p	Phase	The symbolic phase name or phase number given on the command line becomes the new phase; if none is given, the next phase is the current phase plus one, cyclically.
^x	eXecute	The command, given on the command line, is executed; if the command line is blank, a list of the currently available commands is given in the clipboard area. (See Sections 6 and 15.) The 'return' key has the same effect as ^x.

Commands specialized to the clipboard:

^f	File	The file whose name is given on the command line is made the "currently displayed file" and its first ten lines are displayed. If no file is given, the current contents of the Clipboard file, a file used for diagnostics and auxiliary functions, is given.
^v	+page	Advance the file display a page (eight lines).
$v	-page	Backup the file display by a page (eight lines).
^u	eof	Advance the file display to the end of file.
$u	bof	Backup the file display to the beginning of the file.

Additionally, the (UNIX) interrupt key is bound to ^]. If Poker terminates with Abort or Exit and the user wishes to preempt drawing of the final diplay, the user's "normal" UNIX interrupt key can be used.



Final Display

6 File Management

As a programming environment, Poker makes extensive use of many types of files; Figure 5 gives a summary of the types. This section gives a description of the contents of the files and *a description of the copy command.*

Source Database Files

CHiPParams	Information entered in CHiP Parameters view (Poker)
SwitchSet	Information entered in Switch Settings view (Poker)
CodeNames	Information entered in Code Names view (Poker)
PortNames	Information entered in Port Names view (Poker)
IONames	Information entered in IO Names view (Poker)

Process Files

<proc>.x	XX source text for process <proc> (User)
<proc>.s	Assembly code for process <proc> (Poker)
<proc>.o	Object code for process <proc> (Poker)
<proc>.err	Diagnostics file for process<proc> (Poker)
.pe<i>,<j>.o	Object Code for processor element <i>, <j> (Poker)
pe<i>,<j>.err	Diagnostics for processor element <i>, <j> (Poker)

Communications File

connections	Symbolic list of source, target pairs of communication graph (Poker)
connections.o	Object form of communication graph (Poker)

Object File

SnapShot Emulator state (Poker)

Data Files

<name> Streams laid "side-by-side" (User/Poker)

System File

Clipboard Save area for Clipboard displayed information (Poker)

Figure 5. Summary of Files, contents and (creators).

Source Database Files: Poker replaces the usual notion of a source program – a monolithic piece of symbolic text – by a database that is displayed using the views (Section 4). Although the views generally show a composite picture of a portion of the database built from information of several database entities, all the information entered in one view is regarded as one entity and stored externally as one file:

View	Entity Name	FileName
CHiP Parameters	CP	CHiPParams
Switch Settings	SS	SwitchSet
Code Names	CN	CodeNames
Port Names	PN	PortNames
IO Names	IO	IONames

Note that although these files are regular UNIX files they are treated as typed information and are safely manipulated with the **copy** command, (see below). Each entity file contains a description of the CHiP architecture in force at the time of its definition.

Copy Command: The **copy** is an execute command (Section 15) given on the command line and followed by ^x. The activity, as with any copy command, is to move the information stored in the *from structure* to a new location in the *to structure*. The *from* and *to structures* can refer to the current internal state of the Poker system or to external files. As examples we have:

copy PreviousSS, SS 3	loads phase 3 switch settings from a file
copy *, .	saves all source entities (* = SS, CN, PN, IO) in the current directory
copy Proglib, *	loads all source entities from a stored program into Poker; relevant ".x" files are also transferred.

Notice in the third example, Proglib is a directory.

The syntax is:

copy <from structure>, <to structure> ^x

where both operands have the form

<structure> [<phase>]

where the optional phase is given either symbolically or numerically and the <structure> is selected from

SS|CN|PN|IO|*|.|<name>

where

SS, CN, PN and IO refer to the current internal entities of that name,
* abbreviates the set [SS, CN, PN, IO],
. refers to the current directory,
<name> is a file name or directory name,

and the semantics are to copy the specified <from structure> entities to the specified <to structure> entities subject to the following conditions:

- A <name> given as a <from structure> must exist as an entity file (or, if appropriate to the command, a directory).
- A <name> given as a <to structure> with a nonempty phase specification must exist as a entity file.
- A <name> can be an absolute path name.
- A transfer involving * or . has the side effect of transferring those <proc>.x files whose name <proc> is mentioned in the Code Names entity.
- The characteristics of the <from structure> must "conform" to the current CHiP Parameters specifications whenever an entity is read in. If they do not conform the user is given the option to have them transformed as described under the "change" paragraphs of CHiP Parameters (Section 7).

Process Files: The sequential code executed by a single PE is called a "process" and it is specified using a restricted sequential language called XX, see Section 12. The symbolic text for the process named <proc> is stored in a (user defined) file called <proc>.x in the current directory. The XX compiler when executed in the Command Request view, produces a file of assembly code called <proc>.s. The assembler, when executed in the Command Request view produces an object file called <proc>.o in the current directory. Any errors in the compilation or assembly of process <proc> are stored in the file <proc>.err.

The link editor binds a <proc>.o file to a particular PE <i>,<j> and stores the result in .pe<i>,<j>.o; errors in the process are reported in pe<i>,<j>.err.

Communications Files: The communication graph given in the SS entity is compiled into an adjacency list form, i.e. a set of pairs giving the source and target of every edge. The symbolic (user readable) form of this compilation is called "connections", the object form is called "connections.o" and the diagnostics from the compilation are listed in the Clipboard file.

Object State: The state of the emulator can be saved whenever it is stopped. The file is called SnapShot when no other name is specified.

Data Files: External input/output in Poker is based on the concept of a stream, a sequence of values. Stream names are defined using the IO Names view (Section 11). Stream names are bound to file names using the bind execute command (Section 15). Streams are related to normal sequential files as follows.

A stream of *n values is a file of n* records, each record having a single field. Two streams of length *n* and *m*, respectively, form a file of max (*n*,*m*) records, each record having a pair of fields. Larger numbers of streams are treated similarly. If the file is named <filename>, then its streams – say there are three of them – are referred to as

<filename> 1
<filename> 2

<filename> 3

For example, let the file *nums* be the four records:

```
136.25569102,  666.6666666,  -111.1010101,
      3.1415,      0.365590,      -219.333,
      -22.01,      444.0444,      515.6161,
       20.02,        -11421,        212.33,
```

This file can be treated as three streams of length four values each.

Since a Poker program will use a specific number of stream names which we would like to associate with different files when, for example we want to run the program on successive data sets, we give the association with a bind command

bind <streamname> <filename> ^x,

which says that references to streams named <streamname> refer to the streams of the file <filename>. (See Section 15.) For example, if the program's three streams are named *data 1, data 2, and data 3*, then they can be associated with the streams of the *nums* file by the command

bind data, nums

which associates stream *data 1* with stream *nums 1*, etc.

Data File Formats

Records for data files have a fixed number of fields which is the number of streams. Fields are of fixed size: twelve characters followed by a comma. Real data uses standard floating point format (%F), characters are preceded by an apostrophe ('), Booleans are given by TRUE or FALSE, and integers use normal signed representation and are taken to be sints when they are small and positive. Files not in this format can be converted to this format by using the utility program, packIO, described in Appendix B.

Streams can either be unterminated or terminated by **EOS**, mnemonic for end of stream. Reading passed the end of an

unterminated stream yields zeros which are coerced to the appropriate data type (See Section 12). Reading passed a terminated stream produces an internal PE error. To terminate a stream, place the symbols **EOS** at the end of a stream.

7 CHiP Parameters

Purpose: To specify the characteristics of the CHiP machine being programmed and to define symbolic phase names.

Display: The current values of the CHiP computer's parameters are given in the command line; their meaning is described in Table 1. In the field are the symbolic names for the phases.

Activity: The cursor is moved right and left along the command line using (gross or fine) east and west cursor motions. Numbers entered replace the symbol pointed to by the cursor. The new values take effect when the view is changed provided they are in range and satisfy the constraints; no changes take place if *any* parameter is illegal.

Limitations: Certain specifications (e.g. n=64) are not possible on some systems due to inadequate page table space in the UNIX kernel.

Parameter	Range	Constraints	Default
n - size, number of PEs on the side of the lattice.	$2 \leq n \leq 64$	$n = 2^k$	8
w - internal corridor width, the number of switches separating two adjacent PEs.	$1 \leq w \leq 4$		1
u - external corridor width, the number of switches between the perimeter and the edge PEs.	$1 \leq u \leq 4$	$u \leq w$	1
d - degree, number of datapaths incident to PEs and switches.	8	fixed	8
c - crossover level, number of distinct data paths through a switch.	$1 \leq c \leq 4$		4
p - number of phases, the size of of the switch memory.	$1 \leq p \leq 16$		1

Table 1. Description of the CHiP Parameters.

Change *n*: If the value of *n* is increased, the old lattice becomes the upper left-hand corner of the new lattice; if *n* decreases, the new lattice retains the values of the upper left-hand corner of the old lattice.

Change *w*: A change in *w* causes switch columns (rows) to be added or removed from the right (bottom) of vertical (horizontal) switch corridors. Existing switches retain their settings; new switches are unset.

Change *u*: A change in *u* causes switches to be added or removed at the perimeter. Existing switches retain their settings; new switches are unset.

Change *c*: A change in *c* permits the number of distinct data paths through a switch that can be set to be either increased or decreased.

Change *p*: If *p* is increased, phases with consecutive higher numbers are added; if *p* is decreased, phases with high indexes are removed. Added phases are clear.

Phase Names: Symbolic names for the phases can be entered in windows displayed in the field. "Home" from the command line moves to the phase name windows, and from the windows back to the command line. *Motion between windows uses (gross or fine) north* and south cursor keys. Phase names are (a maximum of) 16 alphanumeric characters beginning with a letter.

Global Commands

$h	help	**$^a**	Abort
$s	Switch	**$^e**	Exit
$c	Code	**$^o**	Output screen
$p	Port	**$d**	reDraw screen
$i	Input/output	**^p**	Phase
$r	Request	**^h**	backspace
$t	Trace	**$$$**	noop

8 Switch Settings

Purpose: To specify or modify a processor interconnection structure for the lattice.

Display: The current processor interconnection structure of (a portion of) the lattice for this phase is shown in the field; boxes represent processors, circles represent switches, and lines represent bidirectional data paths.

Cursor motion: Gross cursor motions advance the cursor to the next PE in the indicated direction; fine cursor motions advance the cursor to the next entity (PE or switch) in the indicated direction. "Home", from a switch causes the cursor to return to Last PE, from a PE causes it to go to the command line, and from the command line, to go to Last PE.

Activity: The cursor is moved around the lattice. If the draw mode is set, a wire is "pulled along" from the current position to the cursor's new position. If the remove mode is set, wires traced by the cursor are removed. At a switch all wires common to a level can be highlighted. If the chase mode is set, the cursor follows the wire in the direction indicated until it reaches a PE, or terminates, or reaches a switch that fans out, or cycles.

Generalization: The Pringle and Pringle emulator only support point-to-point communication, but in the Switch Settings view it is possible to define paths that fan out. (See Figure 6.) Communication structures with fanout cannot be emulated.

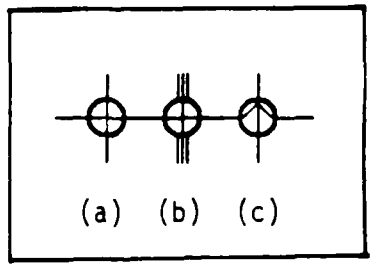


Figure 6. Switch (a) shows two distinct data paths crossing a switch, these paths are on separate levels; the north-south level of the same structure is highlighted in switch (b); switch (c) shows a data path fanning out, i.e. all paths are on the same level.

Recognised Keys:

^c	Center	The cursor is moved to the PE whose index is given on the command line or if this is not visible, the display is changed so it is as close to the center of the field as possible, consistent with the requirement that the field remain fully utilized; if the command line is blank, the Last PE is used for centering.
^d	Draw	The mode is set to "draw" so that subsequent cursor motions cause a line to be drawn.
^r	Remove	The mode is set to "remove", so that subsequent cursor motions that trace a line cause it to be removed.
^g	Go for	Set chase mode, so that (only) the next cursor motion will follow the line in the indicated direction until it terminates, reaches a PE, reaches a switch that fans out or cycles.
^n	Null	End the current mode, i.e. cancel draw, remove or chase.
^l	Level	The level of the switch pointed to by the cursor is changed to the next level. Repeated use of this key cycles through all assigned levels and one unassigned level. The current level is highlighted.
$^k	Klear	Remove all switch settings for the current phase.
<key>		Keys, i.e. alphanumeric text, are placed on the command line.

Commands: The functionality of the above operations is augmented through the use of execute commands. The following are of particular interest for the Switch Settings view:

copy <from structure>, <to structure>
test paths

Refer to Sections 6 (for *copy*) and 15 for details.

Global Commands

$h	cHip	$^a	Abort	^f	File
$s	help	$^e	Exit	^v	+ page
$c	Code	$^o	Output screen	$v	- page
$p	Port	$d	reDraw screen	^u	eof
$i	Input/output	^p	Phase	$u	bof
$r	Request	^x	eXecute	$$$	noop
$t	Trace	^h	backspace		

Interpretation: The line segments specified in Switch Settings mode are provided to create connections between PEs or between pads and PEs. However, some line segments may not participate in establishing a well formed connection; for example, two segments could meet at a switch without crossing it, or a sequence of segments might not connect to a PE. In the other views only legal connections are displayed; the computation of these connections causes the pause when leaving Switch Settings for another view.

9 Code Names

Purpose: To specify or modify the assignment of XX processes to the PEs or to specify actual parameters to the processes.

Display: The current code names and parameter assignments of (a portion of) the lattice for this phase are given in the field. One display format shows boxes representing the PEs; the other display format shows boxes representing the PEs and lines representing the interconnection structure; a key (^t) toggles between these two. A name of up to 16 characters, clipped to five characters, is shown for the program name, and four symbol strings of up to 16 characters, clipped to ten characters, are shown for the parameters:

Cursor motions: Gross cursor motions advance the cursor to the home position of the next PE in the indicated direction; fine cursor motions (north and south) move to the first position of the windows for the code name and the parameters. "Fine home," from a window moves the cursor to the home position of the PE; from the home position in a PE, "fine home" moves back to the last line referenced in a PE. "Gross home" from a PE moves the cursor to the command line, and from the command line, "gross home" moves to the home position of Last PE.

Activity: The names used in the XX process codes (following the word *code*) and (actual) parameter values are entered into the appropriate positions. Code names can be any legal identifier of the XX programming language not containing blanks, and parameters can be any legal constant of the XX programming language.

Buffering: The code name and parameters of a PE can be saved in a buffer (using ^b) that is then displayed in the chalkboard. The PE to be saved is the

PE containing the cursor, or if the cursor is on the command line, the <i><j> given on the command line, or if it is blank, the Last PE. The saved values are deposited into one or more PEs by specifying recipient PEs followed by a deposit (^d) command. Recipient PEs are specified in one of two ways, either explicitly, by giving an index pair (i j), or implicitly. The implicit specifications uses an expression where each index position is either an index, a relation (<, <=, >, >=) followed by an index, meaning all indices standing in that relationship to the index, or a period (.), meaning all index values. Thus, a command

. 2

followed by ^d causes all PEs in the second column to receive the buffered values.

Recognized keys:

^b	Buffer	The code name and parameters of the PE containing the cursor are saved and displayed in the chalkboard. Modification to any of the entries of the buffered PE cause it to be removed from the buffer. The buffered PE remains buffered even if the chalkboard is overwritten.
^d	Deposit	Insert the buffered names into the recipient PE(s). If the command line is blank, the recipient is the PE containing the cursor; if the command line is nonblank the recipient is given by the command line expression as described in Buffering above.
^r	Remove	Delete the code names and parameters in the indicated PEs. If the command line is blank clear the PE containing the cursor; if the command line is nonblank the cleared PEs are given by the command line expression as described in Buffering above.
^c	Center	The cursor is moved to the PE whose index is given on the command line or if this is not visible, the display is changed so that the PE is as close to the center of the field as possible consistent with the requirement that the field be fully utilized; if the command line is blank use the Last PE for centering.

^t	Toggle	The display is changed to the "other" format as described in Display above.
^y	displaY	The full (unclipped) entry of the window containing the cursor is shown in the chalkboard.
$^k	Klear	Remove all code names and parameters entries for the current phase.
<key>		If the cursor is in the window, the symbol replaces the symbol pointed to by the cursor; if the cursor is at the home position of a PE or on the command line, the symbol appears on the command line.

Commands: The functionality of the above operations is augmented through the use of execute commands. The following are of particular interest for the Code Names view:

copy <from structure>, <to structure>
replace <old> <new>

Refer to Sections 6 (for *copy*) and 15 for details.

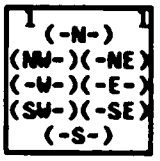
Global Commands

$h	cHip	$^a	Abort	^f	File
$s	Switch	$^e	Exit	^v	+ page
$c	help	$^o	Output screen	$v	- page
$p	Port	$d	reDraw screen	^u	eof
$i	Input/output	^p	Phase	$u	bof
$r	Request	^x	eXecute	$$$	noop
$t	Trace	^h	backspace		

10 Port Names

Purpose: To specify or modify the names assigned to the eight input/output ports of a PE.

Display: The current port names of (a portion of) the lattice for this phase are shown in the field. The display format shows an array of boxes representing the PEs; the other display format shows boxes representing the PEs and lines representing the interconnection structure; a key (^t) toggles between the two. Names of up to 16 characters, clipped to the first five characters, are shown in the PE boxes:



Cursor Motion: Gross cursor motions advance the cursor to the home position of the next PE in the indicated direction; fine cursor motions move the cursor to the first position in the window for the port name for that direction. A "fine home", from a port window moves to the home position of this PE; from the home position in a PE "fine home" moves back to the last window referenced in a PE. "Gross home" from a PE moves to the command line; from the command line "gross home" moves to the home position of Last PE.

Activity: Port names are entered into the appropriate windows to name the ports connecting to the incident data paths. Port names can be any legal identifier of the XX programming language not containing blanks.

Buffering: The port names of any PE can be saved in a buffer (using ^b) that is then displayed in the chalkboard. The PE to be saved is the PE containing the cursor, or if the cursor is on the command line, the <i><j> given the command line, or if it is blank, the Last PE. The saved port names can be deposited into one or more PEs by specifying recipient PE(s) on

the command line followed by a deposit (^d) command. Recipient PE(s) are specified in one of two ways, either explicitly, by an index pair (i j), or implicitly. The implicit specification uses an expression where each index position is either an index, a relation (<, <=, >,>=) followed by an index, meaning all indices standing in that relation to the index, or a period (.), meaning all index values. Thus a command

```
. <= 4
```

followed by ^d causes the first four columns to receive the saved port names.

Recognized keys:

^b	Buffer	The port names of the PE containing the cursor are saved and displayed in the chalkboard. Modification of the port names of a buffered PE cause it to be removed from the buffer. A buffered PE remains buffered even if the chalkboard is overwritten.
^d	Deposit	The buffered names are placed into the recipient PE(s). If the command line is blank, the recipient is the PE containing the cursor; if the command line is nonblank the recipient(s) are given by a command line expression as described in Buffering above.
^r	Remove	Clear all port names in the specified PE(s). If the command line is blank the cleared PE is the PE containing the cursor, if the command line is nonblank, the cleared PE(s) are given by the command line expression as described in Buffering above.
^c	Center	The cursor is moved to the PE whose index is given on the command line or if this is not visable, the display is changed so that the PE is as close to the center of the field as possible, consistent with the requirement that the field remain fully utilized; if the command line is blank, the Last PE is used for centering.
^t	Toggle	The display is changed to be "in the other" format; see Display above.
^y	displaY	The full (unclipped) entry of the window containing the cursor is given in the chalkboard.
$^k	Klear	Remove all port name entries for the current phase.
<key>		If the cursor is in a window, the symbol replaces the symbol pointed to by the cursor; if the cursor is at the home position of a PE or on the command line, the symbol appears on the command line.

Commands: The functionality of the above operations is augmented through the use of execute commands. The following are of particular interest for the Port Names view:

copy <from structure>, <to structure>
replace <old> <new>
test ports

Refer to Sections 6 (for *copy*) and 15 for details.

Global Commands

$h	cHip	**$^a**	Abort	^f	File
$s	Switch	**$^e**	Exit	^v	+ page
$c	Code	**$^o**	Output screen	**$v**	- page
$p	help	**$d**	reDraw screen	^u	eof
$i	Input/output	^p	Phase	**$u**	bof
$r	Request	^x	eXecute	**$$$**	noop
$t	Trace	^h	backspace		

11 IO Names

Purpose: To specify or modify the stream names assigned to the lattice's input/output pads. The "input/output pads" are the points at which wires, given in the switch settings view, extend "off" the edge of the lattice. *Data streams*, sequences of data values, can be read and written through the input/output pads. A file, named <name>, composed of records with k fields is interpreted as a collection of k streams laid "side-by-side". The individual streams are referred to by their <name> and an index which is the field number. For example, <name> 3 is a stream whose first element is the third field of the first record of file <name>. Note, streams are unidirectional.

Display: The field is divided in half. The upper half lists the input/output pads, clockwise beginning from the northwest corner, with the associated stream name, stream index, direction (input or output), as well as the port direction, port name, code name and index of the connected PE. The lower half gives a schematic diagram of the lattice with an arrow pointing to that pad whose window, in the upper half, contains the cursor.

Cursor motions: North/south cursor motions move between pad windows: Fine motions move a single window; gross motions move six windows. East/west (gross or fine) cursor motions move between the name and the index panes. "Home", from a pad window moves to the command line, and from the command line returns to the last pad window.

Activity: The names, indexes and directions (input or output) are entered into the appropriate windows.

Buffering: The stream name, index and direction of a pad can be saved in a buffer (using ^b) and displayed in the chalkboard. The saved values, appropriately modified, are deposited into one or more pad windows by specifying the recipient pads followed by a deposit (^d) command. Saved values are modified by incrementing the saved index value by one prior to each deposit. The recipient pad is specified explicitly by giving one, two or three values on the command line. A single value specifies the pad number of the recipient, two values specify the start and end of a range of recipient pads; and three values specify the start,

end (inclusive) and step size of a subrange of recipient pads. Thus the command

4 1 -1

followed by ^d assigns the first four pads the buffered stream, index and direction information such that the indexes are decreasing, i.e. the lowest index is assigned to pad 4, the next lowest to pad 3, etc.

Recognized keys:

^b	Buffer	The stream name, stream index and direction of the window containing the cursor are saved and displayed in the chalkboard.
^c	Center	The pad window whose number is specified on the command line is placed at the center of the upper half of the field, if the command line is blank, the window containing the cursor is centered.
^d	Deposit	If the command line is blank, the pad window containing the cursor receives the buffered information after the stream index has been incremented, a nonblank command line is treated as a range specification (as described in Buffering above) and the saved information is deposited into each of the specified pads with an incremented stream index.
^i	Input	The pad whose window contains the cursor is designated as an input pad.
^o	Output	The pad whose window contains the cursor is designated as an output pad.
^r	Remove	If the command line is blank the pad window containing the cursor is cleared, a nonblank command line is treated as a range specification and the specified pad windows are cleared.
$^k	Klear	Remove all stream names, indices and directions for the current phase.
<key>		The key replaces the symbol pointed to by the cursor.

Commands: The functionality of the above operations is augmented through the use of execute commands. The following are of particular interest for the IO Names view:

bind <stream> <file>
copy <from structure>, <to structure>
display bind
replace <old> <new>
test pads
unbind <stream>

Refer to Sections 6 (for *copy*) and 15 for details. In addition, facilities for formatting files, such as packIO, are described in Appendix B.

Global Commands

$h	cHip	**$^a**	Abort	^f	File
$s	Switch	**$^e**	Exit	^v	+ page
$c	Code	**$^o**	Output screen	**$v**	- page
$p	Port	**$d**	reDraw screen	^u	eof
$i	help	^p	Phase	**$u**	bof
$r	Request	^x	eXecute	**$$$**	noop
$t	Trace	^h	backspace		

12 The XX Programming Language[1]

Purpose: The XX (dos equis) programming language is a simplified sequential programming language for defining the codes to be executed by the processing elements of the CHiP computer.

Activity: Files are created or modified using a conventional UNIX editor. The files are named <proc>.x where <proc> is the name of the process referred to in the Code Names entries. For convenience in referring to Poker state information on the bitmapped display, it is recommended that XX program files be developed on the secondary Poker display.

Programs: XX programs begin with a preamble that gives the program name, the formal parameters, trace variables and the port names. The preamble is followed by the program body block:

<program> ::= **code** <id> <parmlist>; <tracelist> <port list> <body>

<parmlist> ::= (<idlist>) | λ
<tracelist> ::= **trace** <idlist>; | λ
<portlist> ::= **ports** <idlist>; | λ
<idlist> ::= <id>, <idlist> | <id>
<body> ::= **begin** <declarations> <statlist> **end.**

where the parameters identifier list and **trace** identifiers list are limited to at most four identifiers each separated by commas and the **ports** identifier list is limited to 8 identifiers separated by commas. The identifier following **code** names the program and should match the <name> of the file and the <name> used in the Code Names entries. The parameters are formal parameters that correspond one-to-one to the actual parameters stored in the Code Names/Parameters entries of the PEs; each formal parameter must be declared in the <declarations> section of the <body>. The trace list identifiers have their values displayed during tracing and they must be declared in the <declarations> section of the <body>. They are traced in the order given in the *trace* declaration. The port list identifiers are the symbolic port names that are assigned physical positions in the Port Names entries. Values sent and received

[1] Developed with J. E. Cuny and D. B. Gannon.

through ports are typed and coerced at the destination as described in Table 2.

Declarations: There are five data types: Booleans (1 bit), characters (8 bits), short natural numbers (8 bits), signed integers (32 bits), and signed floating point numbers (32 bits). All identifiers, except statement labels and identifiers declared in **port** or **ports** declarations, must be declared to have a data type; notice this includes the formal parameters. Simple identifiers are scalar values of the indicated type. Identifiers followed by square brackets specify arrays of one or two dimensions whose indices in a dimension run from 0 to <unsignint> in values:

<declarations> ::= <decl>; <declarations> | λ
<decl> ::= <type> <varlist>
<type> ::= **real** | **int** | **bool** | **char** | **sint** | **port**
<varlist> ::= <varid>, <varlist> | <varid>
<varid> ::= <id> | <id> [<unsignint>]
| <id> [<unsignint>, <unsignint>]
where no <id> appears more than once.

The **port** declaration specifies that the variable can be assigned port name constants (declared in the **ports** declaration of the preamble) and be used in functions such as dataavail (see Built-in functions, below.) Variables of type **port** can be used with assignment statements, I/O assignments, and the relational operators = and < >; otherwise, **port** variables cannot be used in expressions; **port** variables cannot be traced.

Statements: The statements are:

<statlist> ::= <lstatement>; <statlist> | <lstatement>
<lstatement> ::= <id>: <statement> | <statement>
<statement> ::= <assignment> | <conditional> |
<while> | <break> | <for> | <compound> | <io> |
<gosub> | <return> | <exit> | λ

where <id> is used for the target of a **gosub**.

Assignment: The Assignment statement is:

<assignment> ::= <varid> := <expression>

where the coercion to the left-hand side identifier type is provided as described in Table 2.

Conditional: In the Conditional statement

<conditional> ::= **if** <expression> **then** <lstatement>
else <lstatement> | **if** <expression> **then** <lstatement>

the <expression> must evaluate to a Boolean value and an **else** is associated with the immediately preceding **then.**

While: In the While statement

<while> ::= **while** <expression> **do** <lstatement>

the expression must evaluate to a Boolean value. To assist in synchronization the compiler recognizes the construction **while true do** <lstatement> as a special case and does not generate the conditional branch code.

Break: The Break statement

<break> ::= **break**

has meaning only within the <lstatement> of a While statement, and causes control to skip to the statement following the immediately surrounding While statement.

For: In the For statement

<for> ::= **for** <id> := <expression> **to** <expression> **do** <lstatement>

the two expressions, the lower and upper limits of the iteration, respectively, are evaluted once prior to beginning the loop. If the lower and upper limits are not integers, they are coerced to integers as described in Table 2.

Compound: Notice that the Compound statement

<compound> ::= **begin** <statlist> **end**

is not a block and may not contain declarations.

I/O: The I/O statements

<io> ::= <varid> <- <expression> |
<varid> <- <expression>, <varid> <- <expression>

require that either the lefthand side or the righthand side of the operator be an identifier declared in the **ports** or **port** declarations. If the port name is on the left then the operation is write or send to the indicated port; if the port name is on the right the operation is a read or receive

to the indicated port. If two I/O statements are specified, one must be a read and one a write. (The semantics are "simultaneous I/O" but since the current hardware cannot support that, the statement is essentially serially executed.)

When a PE performs a read and the data has not been received, the PE waits until the data arrives. If the type of data sent differs from the type of the variable receiving it, type coercion is performed at the destination as described in Table 2. (For the purposes of synchronization, a keyword port named **null** is recognized. It does not actually do I/O, but it uses the same amount of time.)

Subroutines: The statement

<gosub> ::= **gosub** <id>

provides a parameterless subroutine branch to the statement with label <id>. The <id> cannot be located on a statement in the body of a **for** loop. All variables are global. Recursive calls are permitted. Execution of a **return** from the subroutine will cause execution to resume with the statement following the **gosub**.

bool → char: The Boolean bit becomes the least significant bit; others are 0.
char → bool: The least significant bit forms the Boolean.
char = sint: The bit sequences are the same.
sint → int: The 8 character bits become least significant bits; others are 0.
int → sint: The eight least significant bits form the short integer.
int → real: Converted to floating point notation.
real → int: The floating point value is truncated and converted to integer form.

Table 2. Semantics of representation conversion; conversions not listed are performed transitively: type1 → type2 → type3, etc.

Returns: The Return statement

<return> ::= **return**

causes execution of a subroutine to terminate and for control to resume

at the instruction following the **gosub** call. Execution of a **return** when no subroutine is pending produces unpredictable results.

Exit: The statement

<exit> ::= **exit**

causes all execution of the PE code to terminate and for the PE to enter the quiescent state.

Assembler: Allows the direct inclusion of assembler statements in XX code.

Expressions: The expressions

<expression> ::= <expression> <binary> <expression> |
<unary> <expression> |
<expression> <relational> <expression> | <builtin> () |
<builtin> (<expression>) |
<builtin> (<expression>, <expression>)
(<expression>) | <varid>
<unsignint> | <unsignreal> | <character> |
<boolean> | PEi | PEj | PEID | PEn | EOS

have precedence and association as in the C programming language. Expressions of mixed type are coerced to the higher type, where types are ranked **bool < char = sint < int < real,** as described in Table 2. The operators are given in Table 3.

Constants: The constants are unsigned integers for <unsignint>, reals in standard formats, for <unsignreal>, quoted (') characters for <character>, and TRUE and FALSE for <boolean>. A special constant, EOS, terminates streams and can be used as the right-hand-side of an I/O assignment only.

Identifiers: All identifiers begin with a letter and are followed by any combination of letters and numerals. The maximum length of an identifier is 10 symbols. Two keyword identifiers, **PEi** and **PEj** of type sint, are available giving the row and column index, respectively, of the PE on which the code is executing. **PEn** is a type sint and gives the number of PEs in a row or column for the current machine. Additionally, a keyword identifier, **PEID**, is recognized by the compiler but not otherwise supported.

<unary>		<binary>	
+ <real>	no op	<real> + <real>	addition
- <real>	negation	<real> - <real>	subtraction
~ <char>	not	<real> * <real>	multiplication
		<real> / <real>	division
		<real> **mod** <real>	modulus
		<real> >= <real>	greater than or equal
		<real> > <real>	greater than
		<real> <> <real>	not equal
		<real> < <real>	less than
		<real> <= <real>	less than or equal
		<real> = <real>	equal
		<char> & <char>	and
		<char> | <char>	or
		<char> || <char>	exclusive or

Table 3. XX operators. The type indicates the highest type for which the operation is defined; the operation is defined for all lower types.

Arrays: Arrays can only be subscripted by character, sint or integer types.

Built-in functions: The built-in functions taking real arguments and giving real results are: pi ( ), asin(<real>), acos(<real>), atan(<real>), cos(<real>, exp(<real>), log(<real>), ln(<real>), sin(<real>), sqr(<real>), sqrt (<real>), tan(<real>), pwr(<real>,<real>), where the arguments to the last function are base and exponent, respectively.

A built-in function, dataavail (<port>), is a predicate that returns TRUE if data has arrived on the <port> and returns FALSE otherwise. The argument <port> can be either a constant (from the **ports** declaration) or a variable of type **port**. (Two other functions, bitmap and checkports, are recognized by the compiler but not otherwise supported.)

A built-in function, isEOS (<port>), tests a port (either constant or **port** variable) to be equal to the end-of-stream delimiter. EOS returns TRUE if so and FALSE otherwise.

The relationship between dataavail and isEOS is as follows: If isEOS(port) is true then dataavail(port) is false. If isEOS(port) is false then dataavail (port) may be true - meaning that the stream has not ended and the next data item is available - or it may be false meaning that neither data nor an **EOS** token has arrived. An acceptable program sequence for reading a terminated stream is:

```
while ~ isEOS(port) do
  if dataavail(port) then begin ... end;
```

This will busy wait on data, process it in the compound statement when it arrives, and exit the loop when the stream ends.

Multiple terminated streams, may be passed between PEs. To do so, the EOS of one stream must be purged by executing the statement clearEOS (<port>) in order to permit the next stream to enter.

Comments: Comments begin with the characters /* and end with the characters */.

13 Command Request

Purpose: To convert the source form of the program, as specified by the switch settings, code names, port names, IO names and the XX programs, into object form for execution.

Display: The field is cleared and status information is given.

Activity: Commands are invoked which cause the database to be compiled, assembled, coordinated and loaded into the Pringle or the Pringle emulator. In addition the communication structure is compiled and loaded into the emulator. Diagnostics are given in the appropriate ".err" file and in the file Clipboard, which may be displayed using ^f with a blank command line.

Limitation: Although the status information indicates that Poker "coordinates" programs, it does not. Coordination is presently a noop, and consequently all I/O is asynchronous.

Recognized keys:

$m	Make	For each process name <proc> given in code names, a file named <proc>.x is sought, compiled into <proc>.s, assembled into <proc>.o and link edited into .pe<i>,<j>.o; the switch settings are compiled into connections.o, and the whole object specification is loaded into the Pringle or the Pringle emulator.
^c	Compile	If the command line is blank, then for every <proc> given in code names, the file <proc>.x is sought, compiled into assembly code, and stored as <proc>.s. If the command line contains the name of a process <proc> only <proc>.x is compiled to <proc>.s. Diagnostics are given in <proc>.err.
^a	Assemble	If the command line is blank, then for every <proc> given in code names, the file <proc>.s is sought, assembled into object code, and stored as <proc>.o. If the command line contains the name of a process <proc> only <proc>.s is assembled to <proc>.o. Diagnostics are given in <proc>.err.

^k	linK	The object code for the process for each PE<i>,<j> is link edited and stored in the file.pe<i>,<j>.o. [Note .pe<i>,<j>.o is not listed by the UNIX ls command without the -a option.] Diagnostics are given in pe<i>,<j>.err.
^n	coNnections	The communications graph given in switch settings is "compiled" into an adjacency list form and stored in connections.o. A symbolic version is given in "connections" and diagnostics are given in the Clipboard file.
^l	Load	The Pringle emulator is loaded with the link edited object codes, the connection files and the file-to-stream bindings in preparation for emulation.
^g	Go	If the command line is blank, the current phase of the (loaded) program is emulated for 1000 time units or *ticks* and the progress of the emulation is reported each 100 ticks. A non-blank command line is interpreted as the number of ticks to be emulated; progress is reported every 100 ticks.
^d	Dump	The current state of the emulator is saved in the file named on the command line; if the command line is blank the file is called SnapShot. These files serve as checkpoints with which Poker can be restarted.
^r	Reload	The saved emulator state stored in the file named on the command line is reloaded into the emulator; if the command line is blank, the file name SnapShot is used.
^\	interrupt	The Pringle emulator or the Pringle, whichever is running, is interrupted.
<key>		Key input is directed to the command line.

Commands: The functionality of the above operations is augmented through the use of execute commands. The following are of particular interest for Command Request

bind <stream> <file>
continue [<condition>] [trace | notrace]
display bind
flushbuffers
pringle
run [<phase>] [<condition>] [trace | notrace]

script <file>

Refer to Section 15 for details.

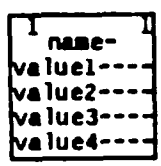
Global Commands

$h	cHip	**$^a**	Abort	^f	File
$s	Switch	**$^e**	Exit	^v	+ page
$c	Code	**$^o**	Output screen	**$v**	- page
$p	Port	**$d**	reDraw screen	^u	eof
$i	Input/Output	^p	Phase	**$u**	bof
$r	help	^x	eXecute	**$$$**	noop
$t	Trace	^h	backspace		

14 Trace

Purpose: To display the current values of the traced variables and to control the Pringle emulator or Pringle execution.

Display: The code name and the current values assigned to the trace variables of PEs in (a portion of) the lattice for this phase are given in the field. One display format shows boxes representing PEs; the other display format shows boxes representing PEs and lines representing the interconnection structure; a key (^t) toggles between the two. The code name is clipped to five characters (and cannot be changed) and values are shown clipped to the first 10 symbols:



If the value of a traced variable changes between two consecutive displays, the value is highlighted.

An equal sign (=) appearing in the home position of a PE indicates that when emulation was suspended, the PE was executing (as opposed to being quiescent). An asterisk (*) in the home position of a PE indicates either that the PE has not begun executing or that it has "crashed" due to a fatal internal error.

Cursor motions: Gross cursor motions advance the cursor to the home position of the next PE in the indicated direction; fine cursor motions (north, south) move to the first position of the windows for the code name and trace variables. "Fine home", from a window moves the cursor to the "home" position of the PE; from the "home" position in a PE, "fine home" moves back to the last referenced line. "Gross home" moves the cursor between the command line and the "home" position of Last PE.

Activity: The execution of a loaded program is controlled and the values of the traced variables are displayed. Execution can be effected in single step

units, multiple steps or until any displayed trace variable changes value.

Limitations: This view cannot be entered unless a program is loaded. Ironically, "poke" - the feature that gives Poker its name - is not implemented.

Interpretation: When Poker runs programs in the Command Request and Trace views, it is emulating the Pringle parallel computer, a 64 processor device based on the Intel 8031 with a floating point coprocessor. The Pringle differs from a CHiP architecture in a variety of ways, but the most important difference is that the logical point-to-point communication of Poker is implemented on the Pringle by a polled bus. A CHiP machine would implement the point-to-point communication directly. The Pringle's bus polling not only yields poorer performance than would be achieved on a true CHiP architecture, but it can also lead to performance differences in successive runs of the emulated programs because PE's I/O requests can be serviced at different times for different runs. It is advisable to emulate the program a few times to get a measure of the possible performance range.

Recognized keys:

^c	Center	The cursor is moved to the PE whose index is given on the command line or if the PE is not visible the display is changed so the PE is as close to the center of the field as possible consistent with the requirement that the field be fully utilized; if the command line is blank, the Last PE is used for centering.
^l	Load	The Pringle emulator is loaded with the link edited object codes, the connection files and the file-to-stream bindings in preparation for emulation.
^g	Go	The command line is interpreted as the (integer) number of steps the emulator is to execute of the current phase; if the command line is blank 1000 steps are executed. The new values of the trace variables are displayed at completion of the execution and a report of progress is given every 100 steps.
^t	Toggle	The display is changed to the "other" format; see Display above.
^y	displaY	The full (unclipped) entry of the traced value window containing the cursor is given in the chalkboard. Notice that the command is needed because certain number representations require more than the available ten character field size.

^e	Event	The command line is interpreted as a pair of positive integers <e <t> specifying that the emulator is to execute the current phase f either <e> events or <t> ticks, whichever comes first. An *eve* is defined to be a change in one or more of the currently displaye trace variables. The traced value causing the event is highlighte on the display. A summary of the number of ticks remaining befo the emulator stops is given on the diagnostic line. A command li containing only one integer <e> is equivalent to a request of <e 1000, and a blank command line is equivalent to a request of 1 100
^d	Dump	The current state of the emulator is saved in the file named on th command line; if the command line is blank the file is called SnapSho These files serve as checkpoints with which Poker can be restarted.
^r	Reload	The saved emulator state stored in the file named on the comman line is reloaded into the emulator; if the command line is blank, th file name SnapShot is used.
^\	interrupt	The execution of the Pringle or the Pringle emulator is interrupte and the current values of the traced variables are displayed.
<key>		The text is directed to the command line.

Commands: The functionality of the above operations is augmented through the use of execute commands. The following are of particular interest for Trace view:

continue [<condition>] [trace | notrace]
flushbuffers
log [<file>]
run [<phase>] [<condition>] [trace | notrace]
script <file>
test PEerr

Refer to Section 15 for details.

Global Commands

$h	cHip	**$^a**	Abort	^f	File
$s	Switch	**$^e**	Exit	^v	+ page
$c	Code	**$^o**	Output screen	**$v**	- page
$p	Port	**$d**	reDraw screen	^u	eof
$i	Input/Output	^p	Phase	**$u**	bof
$r	Request	^x	eXecute	**$$$**	noop
$t	help	^h	backspace		

15 Execute Commands

Purpose: To perform transformations on the source or object states of the system.

Activity: Textual commands and their associated parameters are given on the command line in any view (except CHiP Parameters), and followed by ^x. A blank command line gives a list of the available commands.

Commands: The recognized commands are given below. In general, their syntax is prefix notation with operands separated by one or more spaces; the exception is *copy*. Note that commands require only as many letters as are necessary to disambiguate them from other commands; usually a one or two letter prefix suffices.

bind <stream> <file>
The streams named <stream> of the current phase are associated with the file named <file> as described in "External Data Files" of the file management section (Section 6).

continue [<condition>] [trace | notrace]
The command resumes emulation in Command Request and Trace views only. If no parameters are given then the parameters of the last run or continue command remain in force. The <condition> is a termination condition:

all	Run the phase until all PEs halt.
any	Run the phase until some PE halts.
<i> <j>	Run the phase until PE<i><j> halts.

The trace modifier causes all traced value changes to be displayed; with notrace no changes will be given.

copy <from structure>, <to structure>
Database entities are transferred as described in Section 6.

display <infotype>
Information about the current settings of system or program variables is presented to the user in the Clipboard. The options are:

<infotype>	*meaning*
bind	Lists the stream names and file names bound to them for the current phase (see bind command).
set	Lists the current values of the system parameters that can be set (see set command).

Notice that the <infotype> specification must be given in full.

flushbuffers The contents of the output stream buffers are written to the appropriate files.

help Displays the recognized commands for the current view in the chalkboard; it is equivalent to attempting to change to the current view.

log [<file>]
Append to the <file> the current values of all of the traced variables. If the <file> is not specified, the default "Log" is used.

pringle <keyword>
A command to assist interfacing to the Pringle hardware. The legal options for <keyword> are:

on	Lock the pringle and set it as the recipient of loader output.
off	Unlock Pringle and set the standard emulator as the recipient of loader output.
emulate	Reduce memory limit for standard emulator to 2KB.
lock	Lock Pringle.
unlock	Unlock Pringle.

The command is not applicable for installations without a Pringle.

replace <old string> <new string>

In the Code Names, Port Names and IO Names editing views the <old string> is uniformly replaced by the <new string> whenever it occurs in the current phase of the current view. Notice that this is literal textual substitution.

run [<phase>] [<condition>][trace | notrace]

In Command Request and Trace views only the <phase>, given by its number or symbolic name, is emulated until the given terminating <condition> occurs:

all	Run the phase until every PE halts.
any	Run the phase until some PE halts.
<i> <j>	Run the phase until the PE whose index is i j halts.

The trace modifier causes all traced value changes to be displayed; notrace means no changes will be given. The default are: the current phase for <phase>, all for <condition> and notrace.

script <file>

The keyboard is replaced by the <file> as the primary input source. All commands (except script) are legal. The following conventions are used:

Keyboard	Script file
control <letter>	^<letter>
escape	$
<fine cursor>	% <digit>
<gross cursor>	$% <digit>

where ^ is the caret symbol and <digit> is the numeral corresponding to the direction, e.g. % 2 is a fine south cursor move; see Figure 3. Additionally, ! is recognized as "wait for keyboard input, ignore it, and proceed". IF POKER IS WAITING FOR KEYBOARD INPUT, A PERIOD (.) IS DISPLAYED TO THE LEFT OF THE PHASE INDICATOR IN THE STATUS AREA. The two character pairs, /*, */, are recognized as begin comment and end comment, respectively. All spaces are compressed to a single space.

set <parameter> <value>

The global system parameter is set to the specified value. The parameter name must be given in full. The options are:

<parameter>	*meaning*
checkpoint	Define the frequency of automatic checkpointing of the emulator state; checkpoints will occur whenever <value> $\neq 0$ and (numticks) *mod* <value> $\equiv 0$; the checkpoint is stored in the file SnapShot and replaces any previous checkpoint. The default value is 0.
logpoint	Define the frequency of automatic logging of traced values; logs will occur whenever <value> $\neq 0$ and (numticks) *mod* <value> $\equiv 0$; the log of the traced values is appended to the file Log; the default value is 0.
buffersize	Define the number of bytes of buffer space allocated to each port; the number must be in the range [8, 256] and will be rounded up to a power of two; the default value is 32.

Users who expect to use other than the default values of these parameters as a routine operating procedure may wish to alias the "poker" command to a "batch poker" command (See Appendix B) that uses a short script to set these variables.

shell <command>

The argument is treated as a text string and is sent to the UNIX shell to be interpreted as a UNIX command. A response from UNIX, if any, is sent to the Clipboard and is displayed with ^f.

test <predicate>

The auxiliary function <predicate> is applied to the source database and the resulting diagnostics are displayed in the Clipboard. Auxiliary functions perform diagnostic analysis on Poker programs to insure correctness. The choices for <predicate> are:

paths	Test that all edges given in switch settings are connected paths between two PEs or between a PE and a pad.
pads	Test that the stream specifications are consistent, e.g. that streams with a common name are all either input or output, etc.
ports	Test that for each named port of a PE there is a data path defined, and for each data path connected to a PE there is a port name.
PEerr	Test the PEs for internal errors such as reading from a terminated stream, divide by zero, etc.

If <predicate> is null a list of the available auxiliary functions is given on the Diagnostic line.

unbind <stream>

The indicated <stream> and associated file of the current phase, if any, is purged from the bind table.

16 Acknowledgments

Poker is the product of the ideas and efforts of many people. Janice E. Cuny and Dennis B. Gannon, in addition to contributing to the definition of the XX programming language, were a continual source of ideas, judgement and constructive criticism. Version 1.0 of Poker was written during the summer of 1982 by a delightful and committed group of gentlemen, the "Poker players": Steven S. Albert, Carl W. Amport, Brian G. Beuning, Alan J. Chester, John P. Guaragno, Christopher A. Kent, John Thomas Love, Eugene J. Shekita and Carleton A. Smith. Primary contributions to Version 1.1 were made by Steven J. Holmes and Ko-Yang Wang; their work steadily enhanced the system. The 1984 "Poker players", another congenial group, completed Versions 2.0 and 3.0: Kathleen E. Crowley, S. Morris Rose, James L. Schaad, and Akhilesh Tyagi. Further contributions to the completion of Version 3.0 were made by Philip A. Nelson and David G. Socha. Crowley, Nelson, Schaad and Socha, who completed Version 3.1 during the summer of 1985, made valuable comments that improved the document. Finally, Julie K. Hanover and Eriko De La Mare managed to retain their patience and good cheer throughout the preparation of this complex document and its many predecessors. It is a pleasure to work with such fine people and to acknowledge these valuable contributions.

The design, implementation, testing, documentation and preparation for distribution of the Poker parallel programming environment have been activities of the Blue CHiP Project which has received funding from the following sources: The Office of Naval Research contracts: N00014-81-K-0360, SRO-100, N00014-84-K-0143 and N00014-85-K-0328, National Science Foundation Grant DCR-8416878, and a Department of Defense Instrumentation Award.

A. Library

A library of Poker programs and Poker program schemas is located in /usr/poker/lib. [CAUTION: Your implementation may differ.] A Poker program schema, which encapsulates the parallel aspects of the program, is a Poker program with one phase, "standard" names and trivial process code files; use of a schema saves programming time for routine interconnection structures, since they are easily modified to become a computational phase.

A Poker program is stored as a UNIX directory containing the five Poker database entities - CHiPParams, SwitchSet, CodeNames, PortNames and IONames - and the associated text (".*x*") files. A schema is like a program except that the text files are trivial.

A program or schema can be read into Poker using the copy command (see Section 6). Notice that the stored program is "typed" information, that is it contains a description of the particular architecture for which it was written, e.g. number of PEs, corridor width, etc. When reading in the stored program or schema the CHiP Parameters of the current Poker system should match those of the program. Program names are capitalized and schema names are not. Furthermore, as a heuristic schemas use a naming convention of the form

<short graph name><n>.<w>

where the <short graph name> briefly describes the interconnection structure, <n> is the number of PEs on the side of the lattice, and <w> is the corridor width (see Section 7.)

The structure and contents of the Poker library are shown in Figure A1, where a trailing slash (/) indicates a directory. Within each directory are the files:

CHiPParams
SwitchSet
CodeNames
PortNames
IONames

and one or more text files (e.g. proc.x), where the names depend on the logical process types used in the CodeNames entity.

Notice that for most of the above schemas, smaller instances can be created by first loading the schema into a one phase Poker system, changing the CHiP Parameters to the smaller size, (possible) fixing a few dangling edges, and saving the result. Larger instances can be constructed using the ssc program; see Appendix B.

cardcat	listing of the library contents
PlayingPoker/	sample program from Playing Poker document
BatcherSort/	sort program based on Batcher's algorithm
Load/	Program for 16 × 16, corridor width 1 lattice to load an array stored in row major order, one record per row.
hex16.1/	16 × 16, corridor width 1, hexagonal mesh schema
hex16.4/	16 × 16, corridor width 4, hexagonal mesh schema
mesh16.1/	16 × 16, corridor width 1, 4 mesh schema
mesh16.4/	16 × 16, corridor width 4, 4 mesh schema
shuff8.1/	64 node shuffle exchange graph schema, no pads
torus16.1/	16 × 16, corridor width 1 interleaved torus schema
tree16.1/	255 node tree schema, pad to root on East side

Figure A1. Structure of library as distributed.

B. Related Systems

Poker as described in the foregoing sections represents the production form as it exists at the University of Washington. Other users may prefer other systems because of different hardware or different scientific objectives. The known options are described below; users who port or substantially modify Poker are encouraged to contact the Blue CHiP Project in order that their contribution may be entered into the list.

Figure B.1 Appendix table of contents

B.1 The Pringle Emulator

The execution displayed in the Trace is a literal emulation of the physical Pringle hardware. Any n=8 program run using this emulator can be run on the Pringle hardware with identical behavior and thus it serves as a means of demonstrating physically achievable performance. However, the Pringle will provide an overly pessimistic performance estimate because of its antiquated technology and the fact that it is not literally implementing the CHiP architecture. It is not possible to give a simple statement describing how much improvement could be realized with a true CHiP architecture implementation because computation and communication can be sped up by different amounts. (An overall improvement of at least a factor of 10 may be assumed, but much greater improvements are regarded as very likely.) A revised emulator reflecting more realistic timings is anticipated.

B.2 Sun Implementation

Since most users do not have a BBN BitGraph coupled to a VAX 11/780 as a workstation, the Poker system has been ported to the Sun workstations. The chief differences between the two systems are the format of the display (see Figure 7) and the performance of the system - Poker runs much slower on a Sun. The only other difference is that for the Sun I, the key pad cannot be distinguished from the numeric keys, so all gross cursor motions must be preceded by two escapes (**$$**) and all fine cursor motions must be preceded by one escape (**$**).

Users who run Poker in batch mode (see Section B.5) and redirect the screen output to a file, should display the file using the program "pokercat" in /usr/poker/bin.

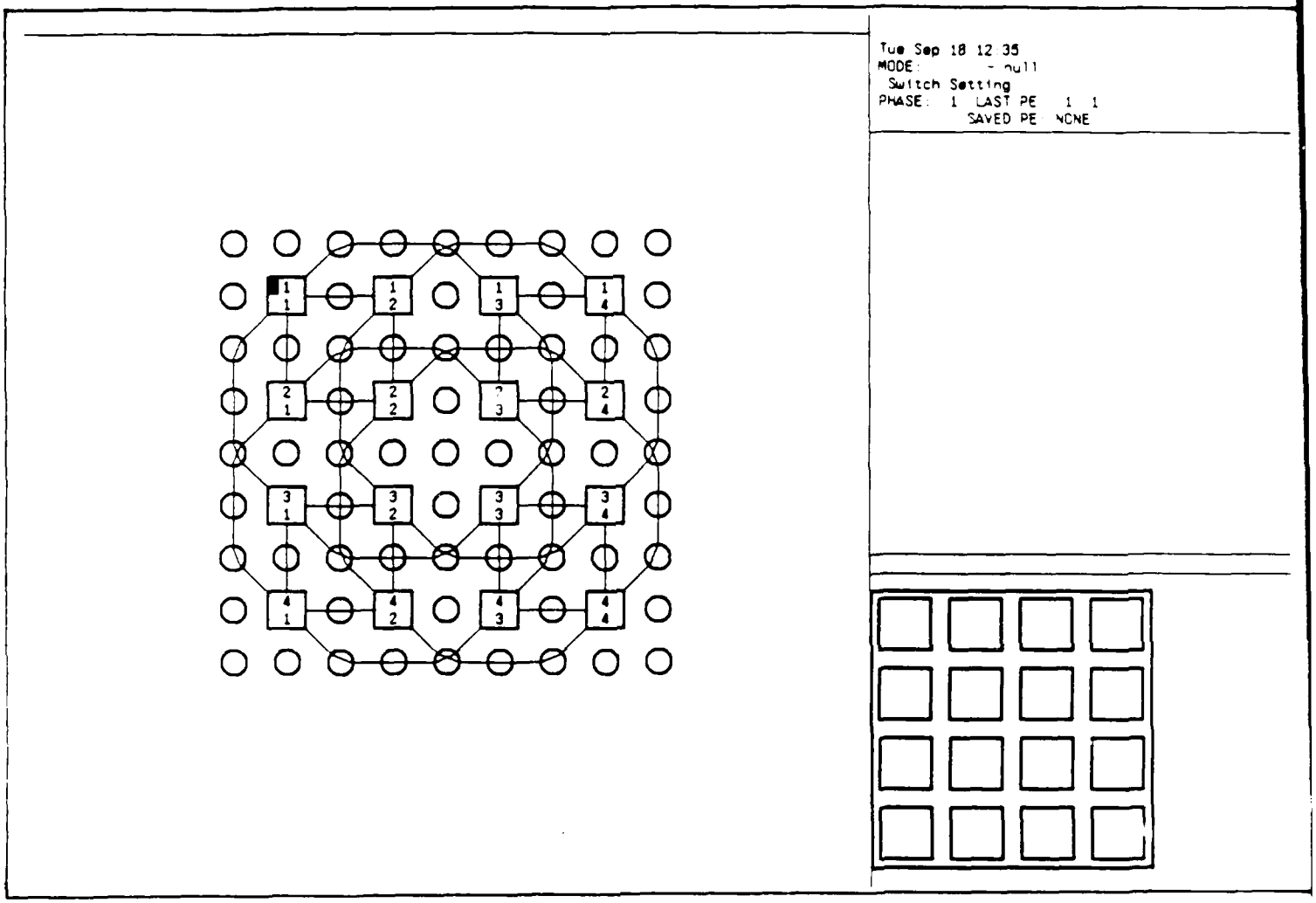
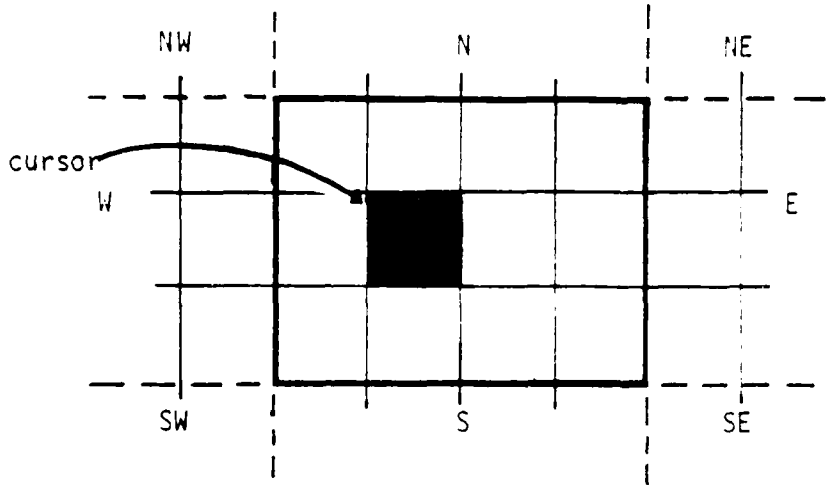


Figure B2. A typical Poker display for a Sun workstation.

B.3 Teletype 5620 Support

The Teletype 5620 Dotmapped Display (or "BLT") is supported as an output device for the VAX 11/780 running UNIX 4.2, or other UNIX server. The screen format is the same as that of the BitGraph display. To run Poker using a BLT, it is necessary to download software to the terminal using an alias, called "setPoker", stored in /usr/poker/bin/setPoker Alias. [CAUTION: Your implementation may vary.] The software can remain resident throughout the session, i.e. it need not be reloaded each time Poker is run.

Cursor motions for the BLT: The keypad is not used for cursor motions, because it is not accessible via the available software. Rather, the mouse is used to indicate direction. A direction is specified by the relationship of the mouse pointer to the current position of the cursor when button 1 (leftmost button on the mouse) is pressed. For example, if the mouse pointer is below the cursor when button 1 is pressed the direction specified is south, and therefore, equivalent to pressing 2 on the keypad of the BitGraph. Specifically, to determine direction, visualize the cursor as if it is leftcenter in a 3 × 4 character box and assign directions with respect to the box:



When the pointer is within the box the "home" specification is implied. To perform gross cursor motions, press an escape (**$**) followed by a mouse click to specify direction. Notice that rapidly entering cursor motions can sometimes result in anomalous behavior because the cursor position changes during screen updating, it is advisable to press button 1 only when the mouse pointer is visible.

Button 2 (center) performs the same way button 1 does except that the direction specification is preceded by the null command (**$**^n). This is useful in Switch Setting view to clear the mode specification.

Button 3 (right most) presents a "pop-up" menu from the terminal support software.

The options are as follows:

continue	no operation
^Q/^S	enable/disable terminal flow control
nullterm	ignore mouse, except this button
poker	emit symbols for Poker
emacs	emit symbols for EMACS
spread	enable/disable character expansion
clear screen	clear screen
exit	leave terminal emulator

Notice that the options are "commands to change state", so their sense is opposite of the current state. It is advisable to disable *spread* because the Poker display format is based on standard size characters.

The BLTs also have an emacs mode permitting the use of the mouse with emacs. The first thing that should be done is to bind an emacs mouse handler to the key sequence <esc>M such as

```
(defun (mouse-set-dot x y b
        (setq b (get-tty-character))
        (setq x (get-tty-character))
        (setq y (get-tty-character))
        (move-dot-to-x-y x y)
    )
)

(bind-to-key "mouse-set-dot" "\eM")
```

This routine will cause both the left and center buttons to move the cursor to the position of the mouse. The right button will cause the BLT menu to appear.

When emacs mode is selected from the BLT menu, 28 boxes are displayed on the bottom of the screen. These may be set to echo one of two different strings when either the left or middle button on the mouse is pushed and the mouse pointer is within the box. The character sequence to load the boxes is

<esc>[?<box number>;<display length>;<left length>;<mid length>S<display string><lef
string><mid string>

and an example of such a set of sequences is:
\033[?0;2;1;1Sup\020\016

```
\033[?1;5;1;1Sright\002R\006
\033[?2;4;2;1Spage\026
\033[?3;4;2;2Snext\030n\030p
\033[?4;5;2;2Ssplit\0302\030d
\033[?5;5;2;3Spause\030\032fg
```

The above many be put into a file and then "cat'ed" to the terminal.

B.4 PackIO

PackIO puts an input (stream) file into the format expected by Poker. Each field is packed into 12 characters, stripping off any characters passed the 12th character or padding the rest of the field with spaces, if needed, and then terminated with a comma in the 13th position. PackIO is a "fixed point operator": repeated invocations on a packed file will not change the file.

The input file must have each entry TERMINATED by a comma. Entries may not have spaces within them, but spaces and tabs may separate entries, commas, and newlines from each other. Make sure to mark "empty" entries with a terminal comma, otherwise packIO will pack your file in a way you might not expect. To quote a comma, i.e. to use a comma as character input, precede it with a single quote ('). The rest of the characters in a field after a quoted character are ignored.

The usage of packIO is:

packIO [<input-file> [<output-file>]]

If no file names are given, packIO acts as a filter, using stdin for the input and sending output to stdout. If only one file name is given, that name is used for the input file, and the output is directed to stdout. If two files are named, the first is the input file, and the second is the output file. The input and output file names must be distinct; if they are not distinct, packIO prints the error message:

Error: input and output file names must be distinct

and then aborts with an error code of 1. If more than three arguments appear after packIO, packIO prints a usage message and aborts with an error code of 1.

If the entry is longer than 12 characters, the extra characters are dropped, and packIO prints a warning message such as:

Warning, line 5, field 2: entry too long; dropping extra characters.

to the stderr device. A warning also is printed for empty lines:

Warning, line 8: empty line in file.

Notice that the files *produced* by Poker are in the proper format as produced and need not be processed by packIO.

B.5 Batch Poker

Poker can be run in a "batch" mode by invoking the system with a file name as a parameter,

poker <script>

where the <script> file uses the conventions described for the script execute command; see Section 15. The result is to run Poker so that the input that would normally come from the keyboard comes from the file <script>. When the <script> file has been processed, the keyboard again becomes the input source. The output is still directed to the screen. It is possible to run Poker in the foreground without using the screen by redirecting the output to a file. The command

poker <script> > <outfile>

stores into <outfile> all of the information that would normally be displayed on the screen; this output can be later "played back" by using the UNIX "cat" command. The output files so produced become enormous (many megabytes) for any reasonable Poker program, so it is suggested that the output be redirected to the bitbucket device, /dev/null. In this case, Poker will only produce results, i.e. output streams stored in files, and there is no graphic record of the run. It is, therefore, recommended that when the screen output is thrown away that log commands (see Section 15) be included in the script. If with redirected output an error occurs while executing the script, Poker will abort (**$**^A); if the script comes to an end, Poker will exit (**$**^E). Finally, to run Poker in the background, one must use ^z, or append an & to the command.

Notice that because batch Poker enables the keyboard when the execution of the script is finished, it is possible to create a script that performs the routine activities normally performed at the start of a Poker session, such as *set*ting state variables, listing .x files in the Clipboard, etc. in order to save having to perform them manually.

B.6 Printing Screen Images

Screen images (created by **$**^o) are prepared for printing on the laser printer using a program, ulbg, as follows:

ulbg -f -r BG.XXXXX

(Use SunUpload rather than ulbg for Suns; the options are the same.) The result is a file, #rastYYYYY, that can be printed on the laser printer. Notice that ulbg produces the file in the background (-f) and so attempts to print the #rastYYYYY file before it is finished will produce unpredictable results. Because of the special symbol (#), UNIX will automatically remove these files a few days after their creation. Other options for ulbg are given in the "man page." [CAUTION: Your system may vary.]

B.7 Switch Set Copy Utility - SSC

Overview

The Switch Set Copy Utility program is a separate utility for manipulating the switch setting entity of the Poker database. This was written as a quick tool for saving time for one Poker user. With a little more work, it was made a little more usable by "any" Poker user.

Switch Set Copy (ssc) manipulates arbitrary rectangles in the switch/pe lattice. The basic operation is to copy one rectangle of switch settings to another rectangle. This includes reflections of the rectangles. The standard copy is a complete copy, but the user can select an "or" mode where previous settings of a switch are not touched. Ssc can handle multiple "SwitchSet" files and multiple phases in a single file. Other commands include turning a square block of switches in increments of 90 degrees and plotting an arbitrary rectangle of the lattice.

Lattice specifications and command formats

In ssc, the user must be able to specify a specific switch, not just the PEs. To avoid counting the number of lattice rows and columns, ssc uses a PE relative addressing scheme for lattice rows and columns. The row with PE 1,1 is addressed as 1. The row of switches above PE 1,1 is addressed as 1-1. The row of switches below PE 1,1 is addresed as 1+1. Also, the row of switches below PE 1,1 can be addressed relative to PE 2,1. For example, with a corridor width of 3, the row of switches below PE 1,1 can be addressed as 2-3.

To specify a particular lattice element, both a row and a column must be given. This is done with the format '<row>;<column>'. For example, the address '1;1' specifies PE 1,1. The address '3+2;4-1' specifies the lattice element that is two lattice rows below and one lattice column to the left of PE 3;4.

Commands are given to ssc in response to the prompt 'ssc>'. Because the first letters of the commands are unique, all that is needed is the first letter. Full command names can be used but all characters between the first character and the first space are ignored. The space is required to separate the command from the parameters. Parameters are separated by commas, and are specified by position; thus, optional parameters must be preceded by the proper number of commas, and commas following the last parameter may be elided.

The commands

1. read [file_name],[logical_name]
   write [logical_name],[file_name]

These commands read and write complete SwitchSet files. The files are always referred to using their logical name. This allows for two copies of the same physical file under two different logical names. The logical names must be unique. File names may be complete paths. (Warning: Since this program is written in PASCAL, no attempt is made to verify that the file exists.)

Defaults:

read - file_name = 'SwitchSet'
logical_name = 'ss'
write - logical_name = 'ss'
file_name = original name for file

Examples:

read	reads 'SwitchSet' with logical name 'ss'.
read ,ss1	reads 'SwitchSet' with logical name 'ss1'.
read /u1/phil/prog/SwitchSet,philss	reads the indicated file and calls it 'philss'
write	writes 'ss' to a file called 'SwitchSet'.
write ss1	writes 'ss1' back into the original file.
write ,newss	writes 'ss' into a file called 'newss'.

2. files

This command lists all the files that are in memory. The listing includes the logical name, the file name, the size of the processor array, the internal and external corridor widths and the number of phases. The "current" file and phase are identified in this list. (See visit command.)

3. visit logical_name,[phase_number]

This command specifies the "current" file and phase. Unless specifically noted otherwise, commands operate on the current file and phase. The read command sets the current file and phase to phase 1 of the last file read in. The logical_name is required.

Default: phase_number = 1

Examples:

visit ss sets 'ss' to current file, phase 1
visit ss,3 sets 'ss' to current file, phase 3

4. copy ul,lr,dest,[from_phase],[to_phase],[from_file],[to_file],[OR]

This command is the reason for this utility. It copies a rectangle of the switch/pe lattice to another identical sized rectangle. The parameters "ul", "lr", and "dest" are what specify the from rectangle and the place to copy it to. "ul" is the lattice element that

is the upper left of the rectangle to be copied. "lr" is the lower right lattice element to be copied. "dest" is the lattice element that the "ul" element will be copied to. The order of copying is determined by the row-major-order sequence of the destination rectangle, and proceeds one switch at a time. Thus, for example, the top row of an array can be duplicated throughout the array by copying rows 1 to n-1 into rows 2 through n. If "lr" is above "ul", the copy goes back row by row from "ul", but the destination is always forward row by row. The switches are correctly reflected about the east-west axis. A similar thing happens for "lr" above and left of "ul", and for "lr" below and left of "ul". The switches are correctly reflected for all copies. The last parameter turns on the "or" mode. Normally, a switch is copied into the new position, over writing the previous setting. A PE copied to a switch leaves an empty switch. A switch copied to a PE does nothing. In "or" mode, current settings of the switch are not disturbed. The copy from a switch sets only unset positions of the destination switch.

Defaults:

from_phase - current phase
to_phase - current phase
from_file - current file
to_file - current file

Examples:

copy 1;1,4;4,1;5	copies PE 1,1 thru 4,4 to 1,5 thru 4,8.
copy 1-2;1-2,2+1;4+2,3-1;1-2	reflects top half of a 4 × 4 to bottom half.
copy 1;1,4;4,1;1,1,2,ss,ss1	copies PE 1,1 thru 4,4 of file ss, phase 1 to PE 1,1 thru 4,4 of file ss1, phase 2.
copy 1;1,2;2,3;3,,,,,OR	copies PE 1,1 thru 2,2 to 3,3 thru 4,4 with the "or" mode.

5. turn upper_left,lower_right,number_of_90s

This command takes a square piece of the lattice and turns it by the specified number of clockwise 90s. The PEs must be placed so that by turning, the PEs end up in the same location relative to the square as before the turn. For example

turn 1;1,2;2,1

is legal, but

turn 1;1,2-1;2-1,1

is not legal. The number of 90s should be an integer in the range 1 - 3. A turn of 4 is a "no operation".

6. plot upper_left,lower_right,[file_name]

This command takes the specified rectangle of the lattice and produces a file for the troff preprocessor "pic". This file when run through pic produces a picture of the rectangle. The user may need to add scaling information to the pic file. The units are in inches. Each lattice element is placed at an integral unit. Seven elements then require seven inches. Scaling and number size are independent. To size down the numbers use the ".ps" troff command for setting the point size. (See the pic users manual for details.) If no numbers are wanted in the PEs, the macro can be changed by putting an 'X' at the end of the first macro line and deleting the second two lines.

Default: file_name = "picfile"

7. quit

This command does the obvious. No write is performed. Also, no warning is given if a file is modified but not written out.

8. help

This command prints a list of the available commands with their syntax.

Cautions

This program changes part of the Poker database without regard to the other parts of the database. Most notably, information in the IONames view may be lost. It is recommended that this utility not be used to write files after information in IONames view has been entered.

B.8 Pen Plotting - bbplot

A program, called "bbplot," is available for plotting lattices on the HP7585 (Big Bertha) and compatible plotters. This program takes as input a "pic" file as generated by ssc. The program assumes the plotter is inline with a terminal. The plotter commands are written to standard output. The command line is as follows:

bbplot [-pen_number] ssc_pic_file

The pen number is optional and must be a number in the range 1-8. The plot is done relative to p1 and p2, (see HP7585 user manual). Make sure that p1 and p2 are set for the correct size rectangle before issuing the command.

B.9 Poker Coordination

A system is under development by Janice E. Cuny and her students at the University of Massachusetts which "coordinates" Poker programs. A coordinated Poker program is one which has been automatically converted to be run synchronously without the expensive handshaking protocols used for asynchronous I/O [1, 2]. Substantial performance improvements can be achieved with coordination. Inquiries should be directed to:

Professor Janice E. Cuny
Computer and Information Sciences Department
University of Massachusetts
Amherst, Massachusetts 01002

[1] Janice E. Cuny and Lawrence Snyder
Compilation of Data-driven Programs for Synchronous Execution
Proceedings 10th ACM Symposium on Principles of Programming Languages pp. 197-202, 1983.

[2] Duane A. Bailey, Janice E. Cuny and Bruce B. MacLeod
Coordination in the Poker Parallel Programming Environment: Parallel Code Optimization
Technical Report, 85-21, COINS Department, University of Massachusetts, 1985.

B.10 Prep-P

A system, called Prep-P, is under development by Francine D. Berman and her students at the University of California at San Diego which maps large Poker programs down to run on a smaller number of processors [1]. This experimental system accepts a Poker program which has as interconnection structure any undirected graph, and "contracts" the graph to an intermediate graph with a fewer number of nodes. The intermediate graph is then laid out on a CHiP lattice whose size may be specified by the user (at most 8×8). The result is that several processes will be mapped to one PE, so the Prep-P system multiplexes the execution of these several processes and provides for the proper communication. Inquiries should be directed to:

Professor Francine D. Berman
EECS C-014
Applied Physics and Math Building
University of California at San Diego
La Jolla, CA 92093

[1] Francine Berman, Michael Goodrich, Charles Koelbel, W. J. Robison III, Karen Showell
Prep-P: A Mapping Preprocessor for CHiP Computers
Proceedings of the International Conference on Parallel Processing (Douglas DeGroot, editor) IEEE, 1985, 731-733.

Poker Command Summary

The symbol **$** abbreviates "escape key" and ^<char> indicates striking the <char> key while holding down the "control key."

Global Commands

$h	cHip parameters	**$^a**	Abort, no state save	^f	display File
$s	Switch settings	**$^e**	End, save state	^v	+page
$c	Code names	**$^o**	Output screen	**$v**	-page
$p	Port names	**$d**	reDraw screen	^u	eof
$i	Input/output names	^p	phase change	**$u**	bof
$r	command Request	^x	eXecute command line	**$$$**	noop
$t	Trace values	^h	backspace		

Specific Commands

Switch Set	Code Names	Port Names	I/O Names	Cmd Request	Trace Values
^n Null	^b Buffer	^b Buffer	^b Buffer	**$m** Make	^e Event
^c Center	^c Center	^c Center	^c Center	^c Compile	^c Center
^d Draw	^d Deposit	^d Deposit	^d Deposit	^d Dump	^d Dump
^r Remove	^r Remove	^r Remove	^r Remove	^r Reload	^r Reload
^g Go for	^t Toggle	^t Toggle	^i Input	^a Assemble	^t Toggle
$^k Klear	**$^k** Klear	**$^k** Klear	^o Output	^k linK	^y displaY
	^y displaY	^y displaY		^l Load	^l Load
				^g Go	^g Go
				^n coNnections	^\ interrupt
				^\ interrupt	

Execute Commands

bind <stream> <file>
continue [<condition>] [trace|notrace]
copy <from structure>, <to structure>
display<type>
flushbuffers
help
log <file>
replace <old string> <new string>
run[<phase>] [<condition>] [trace|notrace]
script <file>
set <param><value>
shell <UNIX cmd>
test <predicate>
unbind

END

DATE

FILMED

DTIC

4/88