

CHAPTER 4 USING MiniMINC WITH SERIAL ASCII EQUIPMENT

This chapter gives a general description of how MiniMINC communicates with serial ASCII equipment and describes how to use LISSJU.BAS, a demonstration program that creates Lissajous figures on an interactive plotter.

MiniMINC can communicate with instruments that transmit characters according to EIA standard RS-232C. This standard sets the protocol for serial transfer by which most terminals (including the MiniMINC terminal) and some other instruments, such as interactive plotters and analytical balances, transmit and receive ASCII characters (see Appendix A in *Book 2: MINC Programming Fundamentals* for a table of ASCII codes). *Serial transfer* is a method of data transmission in which the components of a data word are transmitted one after another (serially) along a single pair of lines from a sending to a receiving device.

SERIAL ASCII EQUIPMENT

You connect devices to MiniMINC via the six SLU (Serial Line Unit) ports located on the back panel of the chassis (see Figure 16).

MiniMINC's SLU PORTS

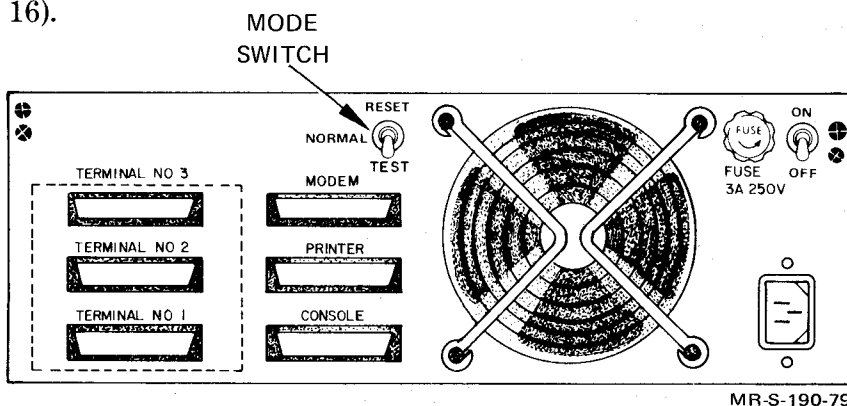


Figure 16. Back of the MiniMINC Chassis

The ports and their uses are:

MODEM	for sending and receiving data with the MINC NFT option via modems (devices used in long distance transmission of digital data), acoustic couplers, or direct cable
PRINTER	for sending data to a printing device (printer)
CONSOLE	for communication with the MiniMINC terminal
TERMINAL #3	for communication with most terminals, printers, and other serial ASCII equipment conforming to EIA standard RS-232C
TERMINAL #2	
TERMINAL #1	

The MODEM, PRINTER, and CONSOLE ports perform only their designated functions; the TERMINAL #3, TERMINAL #2, and TERMINAL #1 ports are identical, general-purpose ports that function with a variety of devices. Chapter 2 of the *MiniMINC Supplement* provides information and illustrations regarding the connection and use of serial ASCII equipment with MiniMINC.

SERIAL ASCII ROUTINES

Communication between MiniMINC and a device depends on programs that include a set of special MiniMINC routines. These routines (SET_SERIAL, CIN, and COUT) transfer characters between MiniMINC and a connected serial ASCII device and determine the speed at which the characters travel, how they are represented, and the method to be used for checking their integrity. Chapter 2 of the *MiniMINC Supplement* describes the routines in detail.

THE SERIAL ASCII DEMONSTRATION PROGRAM

Description of LISSJU

LISSJU, a serial ASCII program, creates Lissajous figures on a Tektronix 4662 interactive digital plotter.

When the program starts, you supply five values that determine what kind of Lissajous figure LISSJU will create. The program computes the graph coordinates for your Lissajous figure and then chains to PLOT, a program that controls the plotter. PLOT causes the plotter to draw a border, the labels, and then the Lissajous figure.

Lissajous figures are frequency patterns commonly generated on an oscilloscope by connecting a source of known frequencies to one set of deflection plates and an unknown frequency to the other set. A recognizable pattern results when one frequency is a harmonic or integral multiple of the other, thus revealing the unknown frequency.

The following line in LISSJU contains the parametric equations used in computing the graph coordinates (REM is the REMARK statement. It distinguishes a comment from the executable portion of a program. See *Book 3: MINC Programming Reference* for a description):

```
X=SIN(X1*M*I+X2)\Y=SIN(Y1*M*I+Y2)\REM THE LISSAJOUS EQUATIONS
```

where:

X1	is the horizontal multiplier
Y1	is the vertical multiplier
X2	is the horizontal phase in radians
Y2	is the vertical phase in radians
M	is 2π divided by one less than the number of points N
I	is an integer that progresses from 1 to N

The ratio of X1 to Y1 determines the actual shape of the Lissajous figure. These multipliers are the frequencies of the X and Y sine waves that combine together to form the Lissajous figure.

The phase shift resulting from the combined effects of X2 and Y2 causes the figure to “rotate” so that if you created a series of Lissajous drawings with identical multipliers (X1 and Y1), each drawing would be a different view of the same figure. Figures 17 through 19 show the rotation of a Lissajous figure with a 1/3 frequency ratio.

The SET_SERIAL routine in the PLOT program sets the attributes of the serial line connecting MiniMINC with the plotter. This routine adjusts the serial line each time you run the programs to create a Lissajous figure. Chapter 2 of the *MiniMINC Supplement* explains the SET_SERIAL routine in detail.

PLOT's Serial ASCII Routines

The COUT serial ASCII routines in PLOT signal the plotter when to print text and when to plot points. They then direct the

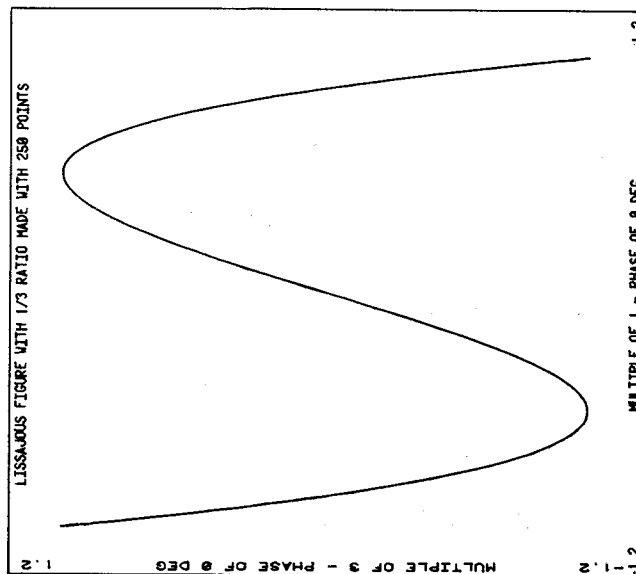


Figure 17.
Lissajous Figure
0°/0° Phase Shift

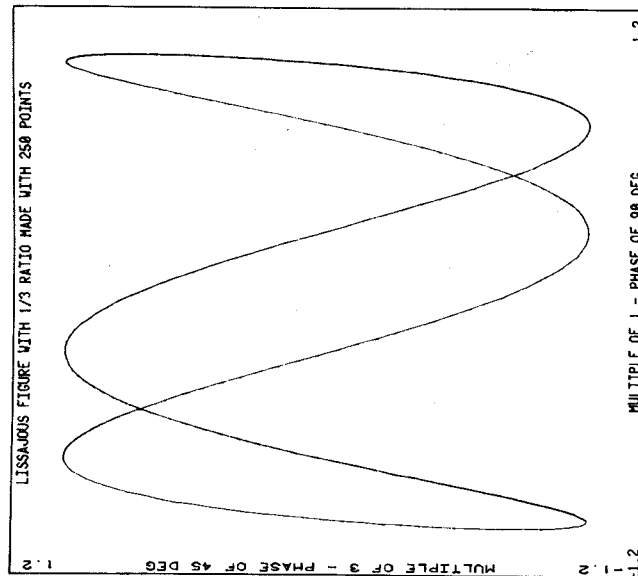


Figure 18.
Lissajous Figure
90°/45° Phase Shift

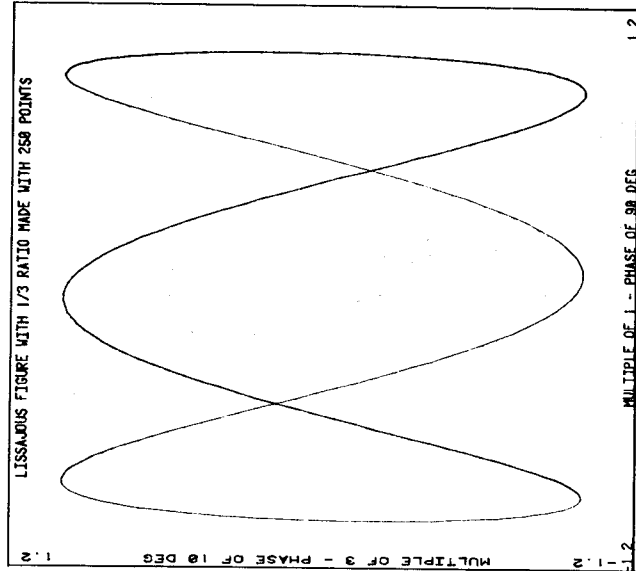


Figure 19.
Lissajous Figure
90°/10° Phase Shift

MR-S-192-79

plotter's pen to the proper coordinates, creating the final figure. Chapter 2 of the *MiniMINC Supplement* also describes this routine and its use.

Power up MiniMINC and the plotter (you can find the plotter's switch settings documented inside the PLOT and LISSJU programs). Next, insert the demonstration diskette into MiniMINC's drive unit 0 and engage the system. Then type:

The Procedure

RUN LISSJU (RET)

LISSJU responds:

LISSJU creates Lissajous figures on an interactive digital plotter (see Chap. 4, Introduction to MiniMINC).

Note that LISSJU produces a figure of almost any frequency ratio and phase shift. However, to produce high-quality drawings, please confine your responses to the following ranges:

1. Ratio between horz/vert multipliers not to exceed 40:1
2. Phases of 0 to 360
3. No. of points no greater than 295

HORIZONTAL MULTIPLIER?

For the purpose of this demonstration, type the following values:

HORIZONTAL MULTIPLIER? 5 (RET)

VERTICAL MULTIPLIER? 6 (RET)

HORIZONTAL PHASE ANGLE IN DEGREES? 18 (RET)

VERTICAL PHASE ANGLE IN DEGREES? 18 (RET)

NO. OF POINTS TO USE (0-295)? 250 (RET)

LISSJU then verifies your answers with the following messages:

LISSAJOUS FIGURE WITH 5/6 RATIO MADE WITH 250 POINTS

MULTIPLE OF 5 - PHASE OF 18 DEG

MULTIPLE OF 6 - PHASE OF 18 DEG

Approximately 50 to 60 seconds will pass before the plotter be-

gins to draw. If you entered the values recommended for this example, the plotter will produce a figure like the one in Figure 20.

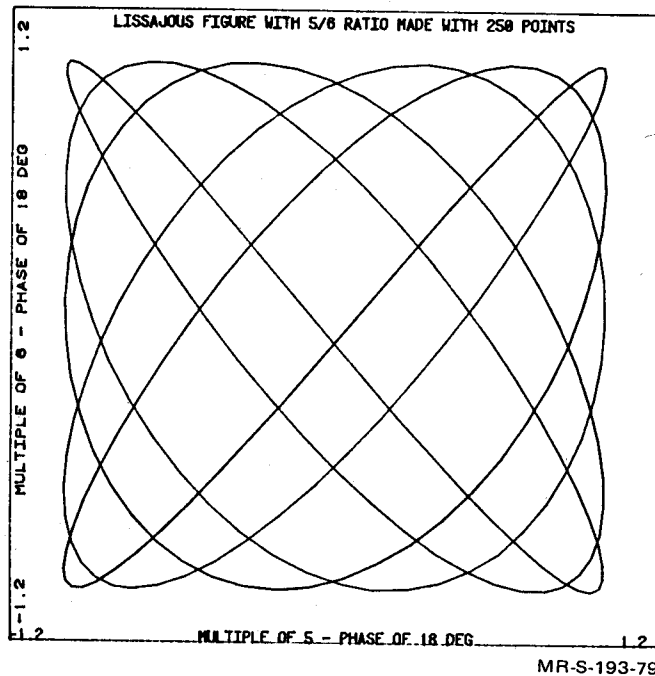


Figure 20. Lissajous Figure — Plotter Sample

You can examine the LISSJU and PLOT programs with the OLD and LIST commands. Type:

OLD LISSJU or OLD PLOT

When MiniMINC responds with READY, type:

LIST

If you have a printer connected to the MiniMINC, type:

COPY LISSJU LP: or COPY PLOT LP:

CHAPTER 5

A MiniMINC PROGRAMMING PRIMER

This chapter describes how programs are used in MiniMINC and suggests a method you can use for developing programs. *Book 2: MINC Programming Fundamentals* discusses the MINC BASIC language and how to program in it.

A program is a set of computer instructions or symbolic statements combined to perform some task.

MiniMINC has two types of programs: system programs that make up the programming system and user programs (also known as application programs) that perform the special tasks required by the user. There are programs of both types supplied with MiniMINC — system programs and some user programs that demonstrate MiniMINC's capabilities.

MiniMINC system programs are machine language programs that are immediately intelligible to the computer without the intervention of a translating facility (the high-level MINC BASIC language requires interpretation before execution). System programs control and direct the computer's activities.

MiniMINC user programs are series of MINC BASIC statements combined to perform tasks required by users. Such tasks include data processing, graphic displays, and serial ASCII data transfers.

Writing a program to run on MiniMINC requires a programming method and knowledge of the MINC BASIC language and program structure. *Book 2: MINC Programming Fundamentals*

PROGRAMS — A DESCRIPTION

CREATING USER PROGRAMS

explains MINC BASIC and this chapter explains a programming method.

There are probably as many ways to write a program as there are programmers, but not every method results in an efficient and effective program. This chapter describes one method that works. Note that the following pages describe steps to take before finally writing the program in the MINC BASIC language.

Defining a Program's Objective

First determine what you want the program to do.

Suppose you want a program that arranges names in a list. There are some basic questions you must answer before you can write the program. For instance:

- How should the names be ordered (alphabetically, randomly, order of importance)?
- In what form should the names appear (full name, last name only, last name and first initial)?
- How many names will the list contain (fixed or variable number)?
- How will the computer receive the names (typed at the terminal or from a file on diskette)?
- In what form should the results appear (on the terminal screen, in a file, or on paper from a printer)?

The answers to questions like these help to clarify the program's objective.

A definition of the program's objective can help you to judge the program's feasibility in terms of available time and resources. For instance, your potential program may require a printer when your system lacks one or you may not have the time necessary to type long lists of names should they be required. Whatever the limiting factor or factors (computer memory size, limited time, unavailable equipment, or insufficient data), you can adjust your program's objective accordingly. Thus a comprehensive definition at the outset of your project can save you time and frustration later on while you're programming.

Pretend, for the sake of illustration, that you need a program to sort a list of names. Your requirements are for a program that will:

- accept up to 1000 names from the terminal
- use names in the form of last name and first initial
- sort names alphabetically
- display the names in correct order on the terminal screen

This program already exists as the demonstration program ALPHA. The remainder of this chapter traces ALPHA's development up to the final programming effort.

The next step in building a program is to establish your programming strategy.

Outlining a Solution

Create an outline for your program by describing the program's method of operation in general terms. Your outline should include every process required by your program to achieve its objective. The processes should account for the input of data, data processing, and the output of data. When creating the outline, list the processes according to their sequence in the program.

For example, you can outline ALPHA in the following way:

1. Input names
2. Reorder names
3. Output reordered names

This outline represents everything the program must do. Step 1 is data entry, Step 2 is data processing, and Step 3 is the result.

Outlining a solution increases the likelihood that you will develop an effective program. If you are satisfied that the outline contains all the processes required by the program's objective, then proceed to the next step. The ability to define a process will develop as you learn the MINC BASIC program structure described in *Book 2: MINC Programming Fundamentals*.

An *algorithm* is a fixed series of well-defined procedures that solves a problem in a finite amount of time. It is the essence of a program.

Constructing an Algorithm

An algorithm summary is the written representation of an algorithm. The summary often takes the form of brief English sentences or statements that succinctly describe each procedure

within the algorithm. These sentences or statements appear in the summary in the same order as the procedures they describe.

Consider the outline for ALPHA. The first process, inputting the names, uses an algorithm described in the following summary:

1. Create array for storing unordered names
2. Receive name from terminal
3. If no more names, then go to Step 6
4. Store name in unordered array
5. Go to Step 2
6. Stop

The procedures are performed in numerical order. However, some procedures override this order by directing control to a specific procedure as in steps 3 and 5. Step 3 switches control to Step 6 when you are finished typing names. The switching procedure in Step 3 reflects a MINC BASIC programming technique called a *conditional branch*. Step 5 always passes control to Step 2. This switching procedure reflects an *unconditional branch*. You should always take care to limit the number of branches of either kind. Algorithm summaries (and programs) can become confusing unless most of the activity is performed in simple numerical order.

You may wonder why there are so many steps in this algorithm summary. This is because each procedure corresponds to a MINC BASIC statement or set of statements that you will learn about in Book 2.

For the moment, concentrate on the form of the algorithm summary. Mentally test the algorithm. The more you remember of its pattern, the better you'll understand the discussions on program structure in Book 2. This is true because a program is an algorithm translated into action by the use of a programming language.

The following algorithm summaries show how ALPHA reorders and then outputs names.

REORDER NAMES

1. Create array to store ordered names
2. Take next name from unordered array
3. Compare name with next name in unordered array
4. Take name with lowest alphabetical position, switching position of names if necessary
5. If name compared with last element of unordered array, then go to Step 7
6. Go to Step 3
7. Insert name in first open position of ordered array
8. If every name in unordered array tested for alphabetical order, then go to Step 10
9. Go to Step 2
10. Stop

OUTPUT ORDERED NAMES

1. Display next name in ordered array
2. If end of ordered array, then go to Step 4
3. Go to Step 1
4. Stop

NOTE

The word "next" in the algorithm summaries can also mean "first." This practice reflects how the BASIC language handles arrays.

Examine these algorithm summaries carefully. Notice how explicit they are. This exactness is required because programs direct the computer's operation, and computers perform only as directed. If you write a program based on an ambiguous algorithm, you must compensate for the ambiguity while program-

ming. However, because programming involves much more detail than constructing algorithms, allowing ambiguity into your algorithm summary greatly increases your work load.

The three algorithm summaries given in this chapter should be combined into one summary to represent the entire ALPHA program. The resulting algorithm summary for ALPHA is:

1. Create array for storing unordered names
2. Receive name from terminal
3. If no more names, then go to Step 6
4. Store name in unordered array
5. Go to Step 2
6. Create array to store ordered names
7. Take next name from unordered array
8. Compare name with next name in unordered array
9. Take name with lowest alphabetical position, switching position of names if necessary
10. If name compared with last element of unordered array, then go to Step 12
11. Go to Step 8
12. Insert name in first open position of ordered array
13. If every name in unordered array tested for alphabetical order, then go to Step 15
14. Go to Step 7
15. Display next name in ordered array
16. If end of ordered array, then go to Step 18
17. Go to Step 15
18. Stop

Notice that the “stops” formerly at the end of each process algorithm are missing. They were needed because a *loop* (a repeating sequence of events) inside each process terminated itself and the process by sending control to a stop. Now that the processes are combined, control passes to the next process instead of terminating at a stop. The stop at Step 18 remains to terminate the ALPHA algorithm.

The algorithm described for ALPHA is only one of several algorithms capable of the same function. In fact, there is almost always more than one solution to any given programming problem. Keeping this in mind, you should evaluate your program objectives carefully in order to construct the most efficient and effective algorithms possible. A useful technique that can improve your understanding of a process, as well as the quality of the final algorithm, is to develop several algorithms for the same program.

After constructing an algorithm, and in its early stages of development, you should test the algorithm for integrity of design or soundness.

Testing an Algorithm

To be sound, an algorithm must solve a problem in a finite amount of time with a fixed series of well-defined processes. Every process required to solve the problem must therefore be included within the algorithm and occur in proper sequence. There must also be a point in time when all the processes are finished and the problem is solved.

A method for appraising an algorithm's design integrity is to mentally trace the path of several data items through the algorithm, one item at a time. You can detect flaws with this method that you would probably overlook in a general inspection.

Try this method on ALPHA's algorithm. Perform each step in the algorithm separately and in the indicated order. For now, and whenever you test an algorithm in this manner, suspend your intuition and perform each step exactly as described.

Using a piece of paper as the unordered array, write down a name each time you encounter Step 4, until you've entered four or five names (any names will do). When you've entered all the names, Step 3 directs you to Step 6. Continue with the algorithm until you reach the stop in Step 18. Use additional paper to represent the ordered array and ALPHA's output.

This simple test method should work for all programming algo-

rithms. Proceed to the next stage in program development, implementing the algorithm, when you are convinced of the algorithm's soundness.

Implementing an Algorithm

An implemented algorithm is a program. The writing of a program requires knowledge of a programming language and program structure.

You should next read *Book 2: MINC Programming Fundamentals*. This book explains what you need to know to begin programming in MINC BASIC. The other MiniMINC manuals (*Book 3: MINC Programming Reference*, *Book 4: MINC Graphic Programming*, and the *MiniMINC Supplement*) describe the MINC BASIC commands, statements, and routines.

SUMMARY

The MiniMINC system uses two types of programs: system and user programs. System programs control and direct the computer's operations and thus form part of the MiniMINC system. User programs are created by MiniMINC users to perform tasks with the system. The system programs are machine language programs and the user programs are high-level language programs.

There are many ways to create user programs. This manual demonstrates one of them. The steps involved are:

Define a program's objective

Outline a solution

Construct an algorithm

Test the algorithm

Implement the algorithm

Book 2: MINC Programming Fundamentals explains how to implement an algorithm with the MINC BASIC language.

CHAPTER 6

DIRECTIONS FOR READING THE MiniMINC MANUALS

If you are an experienced programmer in a language other than BASIC, we recommend that you follow the reading order suggested below for novice users.

EXPERIENCED PROGRAMMERS

If you are familiar with BASIC, but not MINC BASIC, you should at least skim Books 2 and 3 for new commands and the MiniMINC approach to standard commands.

Book 2: MINC Programming Fundamentals continues your education in MiniMINC programming, but concentrates on more practical matters such as file creation and editing. It teaches MINC BASIC fundamentals in a thorough, uncomplicated manner using examples. You should find Book 2 and Chapter 5 of this manual sufficient preparation for programming on your own.

NOVICES

You will probably need to review sections of Book 2 before you fully understand MINC BASIC as a "language." Review and practice are the keys to a functional understanding of MINC BASIC or any other programming language. Only by gaining experience will you achieve programming proficiency.

You should find *Book 3: MINC Programming Reference* useful once you start programming. Book 3 and the *MiniMINC Supplement* together describe every command, statement, function, and routine in MINC BASIC.

Book 4: MINC Graphic Programming teaches the concepts underlying graphics and conducts you through some demonstrations. Its second half serves as a reference manual for graphic

INTRODUCTION TO MiniMINC

routines once you begin graphic programming. We recommend that you read it thoroughly beforehand.

Reserve the *MiniMINC Supplement* for last. This book contains technical information for connecting MiniMINC to printers and other devices, testing your equipment, and correcting problems that may arise. The Supplement also describes new features that Books 1 through 4 don't point out.