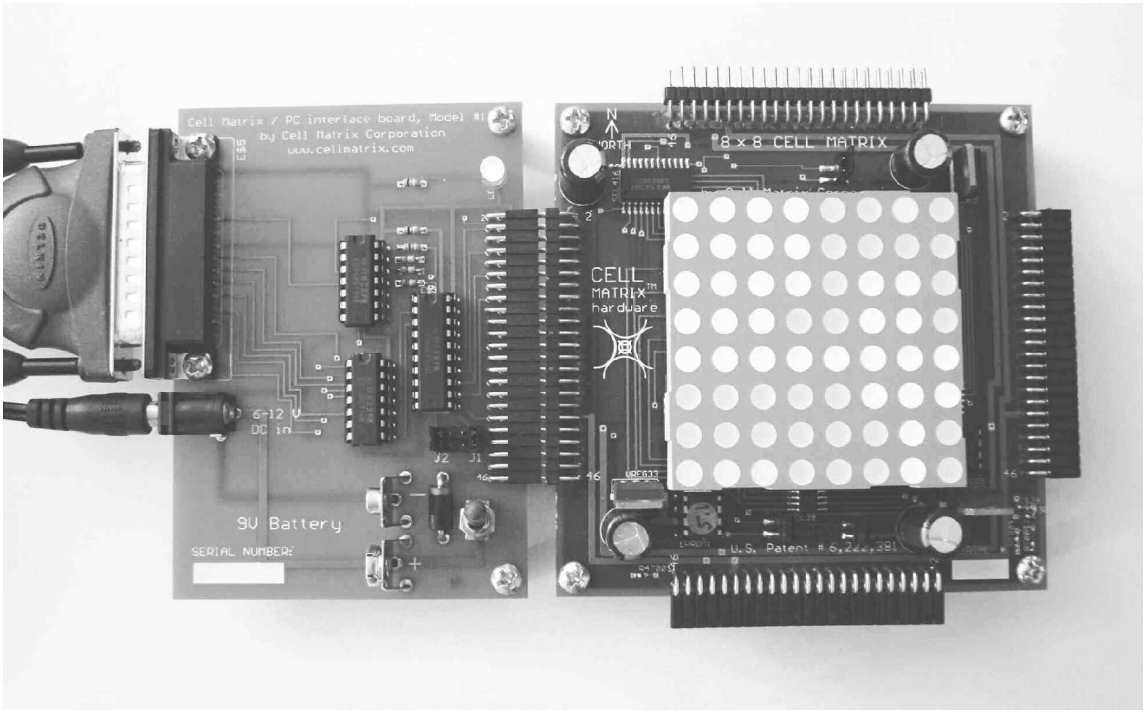
USER MANUAL and TUTORIALS

for the

CELL MATRIX
MOD 88™



Cell Matrix Corporation, www.cellmatrix.com

Table of Contents

1.INTRODUCTION

The Cell Matrix™ is a novel type of reconfigurable hardware device. Like other reconfigurable devices, the Cell Matrix consists of a number of elements (called cells) that can be configured to perform various logic operations. These cells then read inputs, and produce outputs according to their configuration.

The Cell Matrix is distinguished from other reconfigurable devices in a number of ways: it is extremely scalable; it is very *fine grained*, meaning that its atomic unit is very small; and, in additional to being reconfigurable, the Cell Matrix is *self-configurable*. Tutorials 7-10 (Chapter 6) introduce the use of this self-configurability.

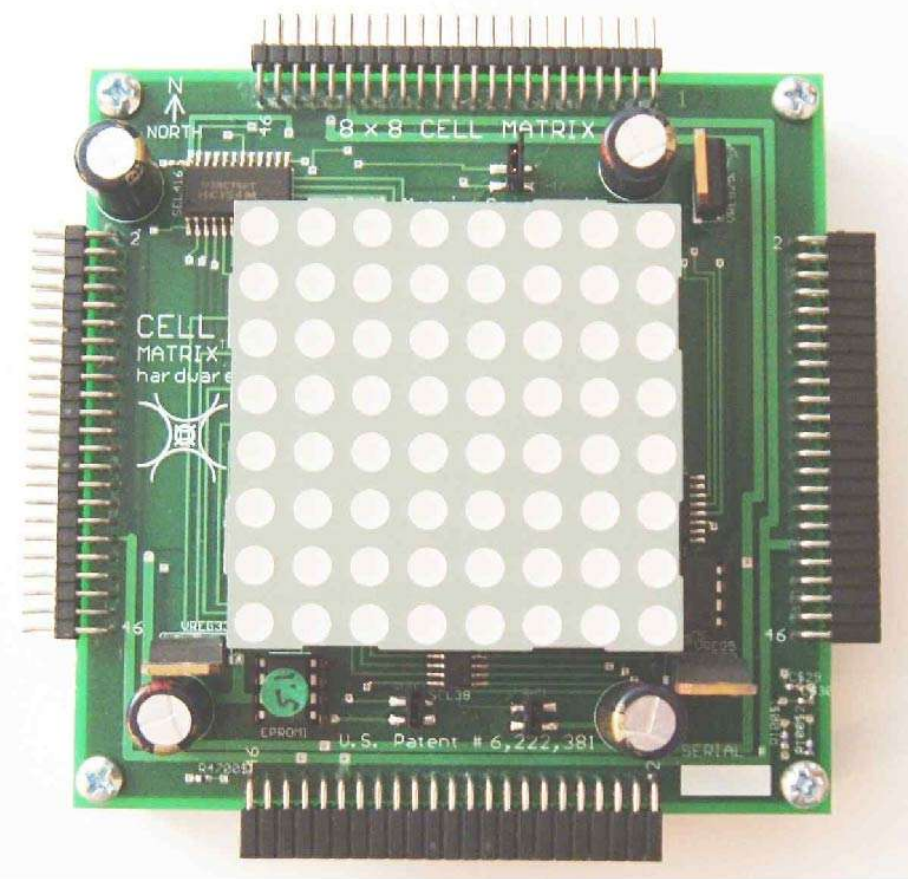


FIGURE 1 - THE CELL MATRIX MOD 88™

The *Cell Matrix Mod 88™* (Figure 1) is a hardware implementation of a two-dimensional, 8x8 Cell Matrix. The board has connectors on each

edge, that allow access to the C and D inputs and outputs of each edge cell. Additionally, the board can be attached to a computer's parallel port via an optional *PC Interface Board* (Figure 2).

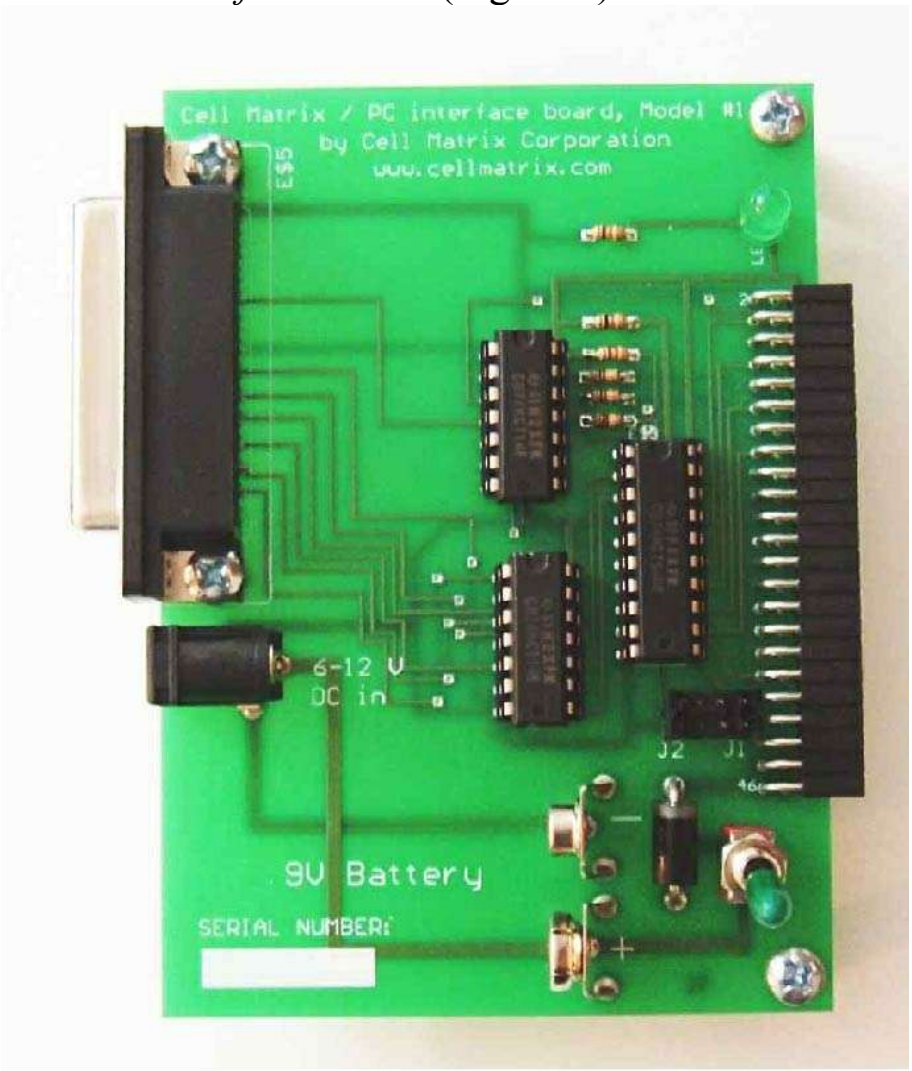


FIGURE 2 - PC INTERFACE BOARD

The *Cell Matrix Mod 88* provides a hands-on introduction to the Cell Matrix architecture. It can be used to study the basic behavior of Cell Matrix cells, including their two dual modes of operating (C-mode, where cells process code, and D-mode, where cells process data). It can be used to experiment with reconfigurable devices in general, e.g., for performing experiments in Evolvable Hardware. It can also be used as a general digital logic platform, for working with logic circuits at the hardware level. This document does not discuss the Cell Matrix architecture *per se*. It assumes a basic familiarity with the architecture,

but not with the detailed use of the Cell Matrix. Please visit www.cellmatrix.com for more information about the Cell Matrix. The page www.cellmatrix.com/entryway/products/pub/publications.html contains publications that describe both the basic and the more advanced features of the Cell Matrix architecture.

Document Organization

Chapter 1 (this introduction) gives a basic overview of the *Cell Matrix Mod 88*. It also discusses the hardware and software requirements of the system.

Chapter 2 describes the *Cell Matrix Mod 88* itself, independent of how you will be interacting with it. This chapter also includes installation instructions.

Chapters 3-6 discuss the *Cell Matrix Mod 88*'s use with an external computer (PC compatible). Chapter 3 covers the *PC Interface Board*, which connects the *Cell Matrix Mod 88* to a standard PC via the parallel port. Chapter 4 describes the Command Line Interface, a program that allows one to control the *Cell Matrix Mod 88* via interactive commands. This is a good chapter to read, even if you're planning to use only the programming interface (API), or to not use the Interface Board at all, as it covers some basic operations of the *Cell Matrix Mod 88*. Chapter 5 discusses an API for communicating with the *Cell Matrix Mod 88* via the Interface Board. Chapter 6 suggests some example tutorial exercises using the *PC Interface Board* and the Command Line Interface.

Chapter 7 provides details on using the *Cell Matrix Mod 88* without the *PC Interface Boar*d.

Chapter 8 discusses how multiple *Cell Matrix Mod 88s* can be used together.

Chapter 9 discusses how to find more information.

The following are the hardware and software requirements if you plan to use the *Cell Matrix Mod 88* with a PC (this is the most convenient way to interact with the *Cell Matrix Mod 88*):

Hardware Requirements:

- PC (Windows or DOS) or Linux machine
- Parallel port cable (25-pin, straight-through, M-M (male end connectors))
- One available parallel port (LPT1: or LPT2:)
- 9V battery or external power source (6-12V DC)
- CD reader or network connection for loading software

Software Requirements:

- Linux , DOS or Windows operating system (see **INSTALLATION** chapter for specific requirements)
- Depending on your setup, you may need Admin or Root access on your system
- JAVA 2 runtime environment, if you wish to use the *Cell Matrix Layout Editor*^{TM} to design cells and Cell Matrix configurations graphically (works only under Windows or Linux operating systems)
- Appropriate compiler/linker environment if you wish to write programs to interact with the board (see **INSTALLATION** chapter for specific requirements)

Cell Matrix Mod 88 Overview

The *Cell Matrix Mod 88* implements an 8x8 Cell Matrix in hardware. The matrix is composed of two-dimensional, four-sided cells, connected in a nearest-neighbor topology, as shown in Figures 3 and 4.

On top of the board is an 8x8 LED array, comprised of 64 bi-color (red and green) LEDs. This array is used to display activity within the matrix. The display is constantly operating, and thus continually shows the state of the cells within the matrix.

NOTE: It may be difficult to see the LEDs in bright indoor light, or direct sunlight. If it is difficult to see, try lowering the ambient light level, or shielding the array from direct light.

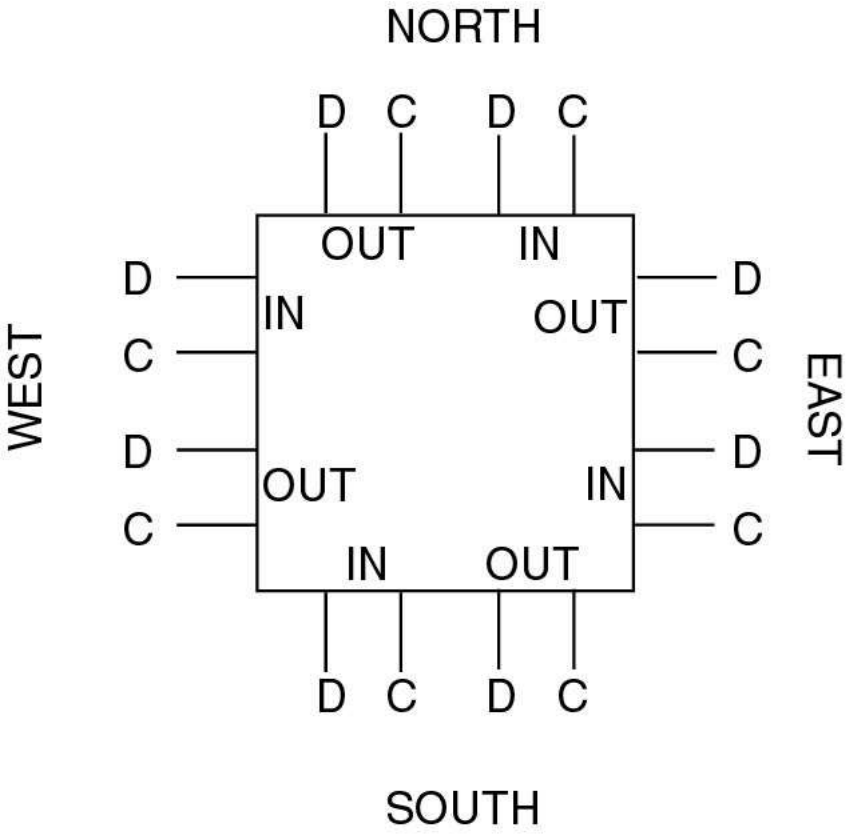


FIGURE 3 - A SINGLE CELL MATRIX CELL

Each of the LEDs corresponds to one cell within the matrix. An LED can be in one of three states:

- LED is red – the cell is in C-mode, i.e., its truth table is being modified;
- LED is green – the cell is in D-mode, and at least one of its D-outputs is non-zero
- LED is dark – the cell is in D-mode, and all of its D outputs are zero.

There are also more subtle output states:

- a half-intensity green LED means a cell's D-output is unstable (e.g., inverter feeding itself), and is alternately outputting 1s and 0s;
- a half-intensity red LED means a neighboring cell's C output is unstable, and is alternately outputting 1s and 0s; and
- an orange LED means the cell's mode is unstable, and it is alternating between D-mode with asserted D outputs, and C-mode.

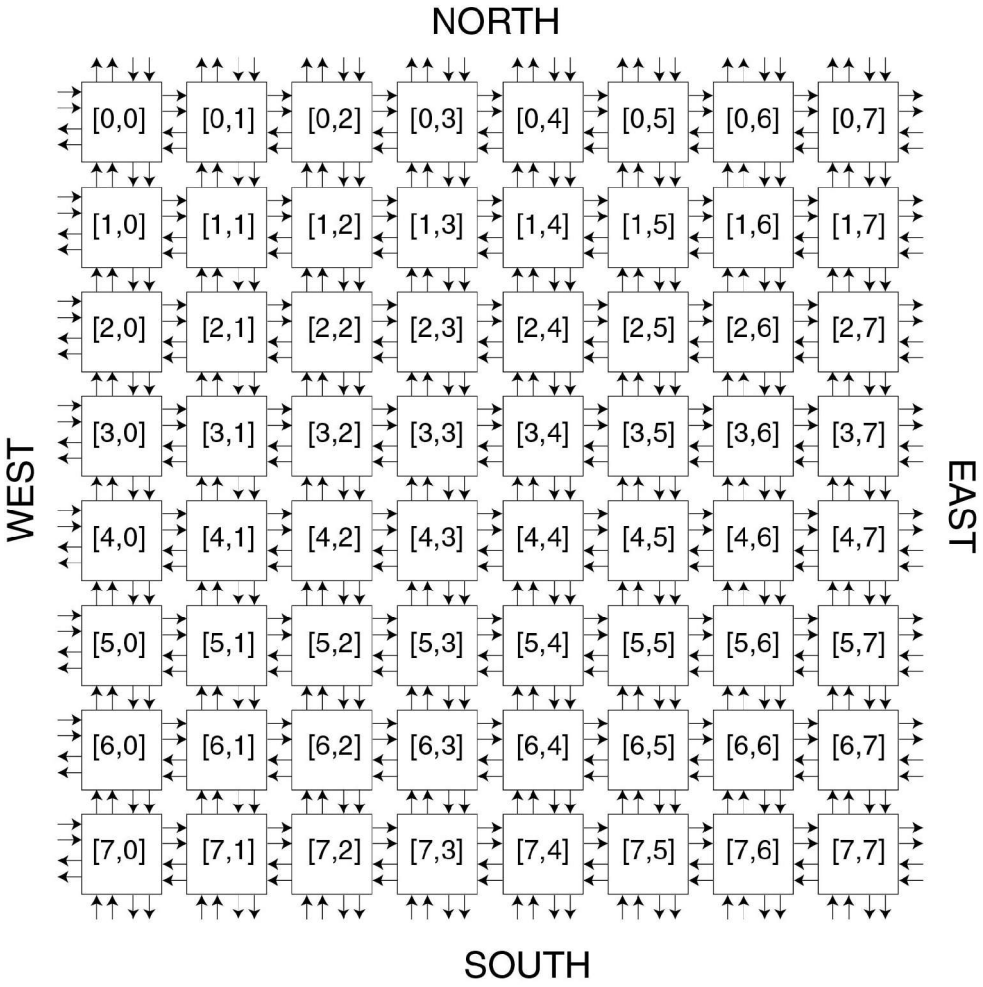


FIGURE 4 - AN 8x8 CELL MATRIX
Cells are labeled with [row,col] designation

With experience, this simple output mechanism can be extremely useful for understanding what is happening inside the matrix. For example, signal transmission tends to look like a line of green LEDs; cell configuration at the end of a line of cells looks like a line of green LEDs, followed by a red LED; and so on.

Edge connectors along the sides of the *Cell Matrix Mod 88* provide access to edge cells, system signals, and power. They are arranged so that two or more *Cell Matrix Mod 88s* can be attached to each other, to create a larger matrix. Figure 5 shows 4 connected boards, implementing a 16x16 Cell Matrix. See Chapter 8 for more information on using multiple boards.

Unconnected D and C inputs are pulled low to logic level 0 on the board itself. There is no need to set these values to 0 yourself. **This is not so for the clock and reset lines though**. These lines must be given definite values from outside the *Cell Matrix Mod 88*. The *PC Interface Board* takes care of this for you.

In working with the Cell Matrix, a relatively small number of possible interactions are combined in a variety of ways to induce extremely complex behavior. The following is a complete list of how one may interact with the Cell Matrix:

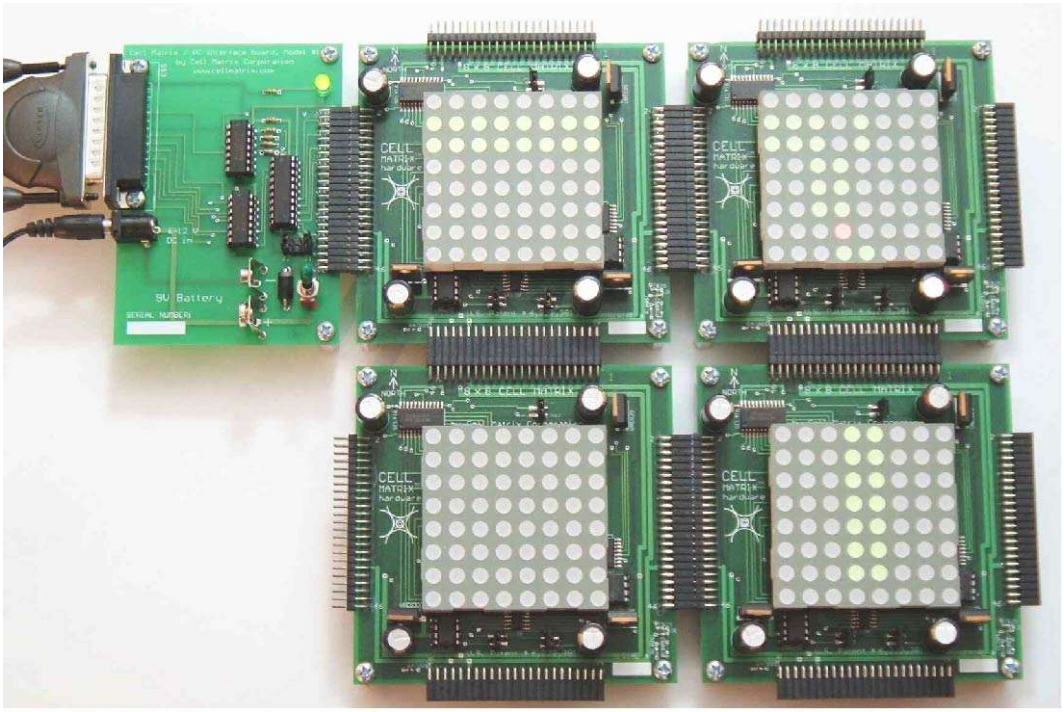


FIGURE 5 - PC INTERFACE BOARD CONNECTED TO
FOUR CELL MATRIX MOD 88 BOARDS
This implements a 16x16 Cell Matrix

- C and D edge inputs of edge cells can be set;
- C and D edge outputs of edge cells can be read;
- the system clocks (Phi 1 and Phi 2) can be driven high and low; and
- the system RESET line can be driven high and low

Despite this seemingly limited set of interactions, one can get the *Cell Matrix Mod 88* to do many things, because the Cell Matrix is a *self-configurable* system. By controlling C and D inputs and the system clocks, one can configure the truth tables of edge cells. By properly controlling such cells, one can gain control over internal (non-edge) cells. In this way, one can manipulate the entire matrix using only a few edge inputs and outputs.

Jumper Settings

There are three jumpers on the *Cell Matrix Mod 88*: all three jumpers must be in place (so each pair of pins is shorted by the jumper) in order for the board to work properly. If any jumpers are missing, unpredictable results may occur. See Figure 6 for the location of these three jumpers.

Precautions

Unlike some other reconfigurable devices, it is impossible to damage the Cell Matrix board via "improper programming."Outputs cannot be wired together via configuration, and thus output contention is impossible. However, it is still possible to damage the board. Some situations to avoid include the following:

- Supplying incorrect power (wrong polarity or too high, or too much current);
- Supplying an incorrect voltage to an input;
- Supplying too much current to an input;
- Electrically shorting outputs together; and
- Attempting to drive an output that the board is already driving.

The above situations can be avoided by using a properly-connected, properly-powered *PC Interface Board*.

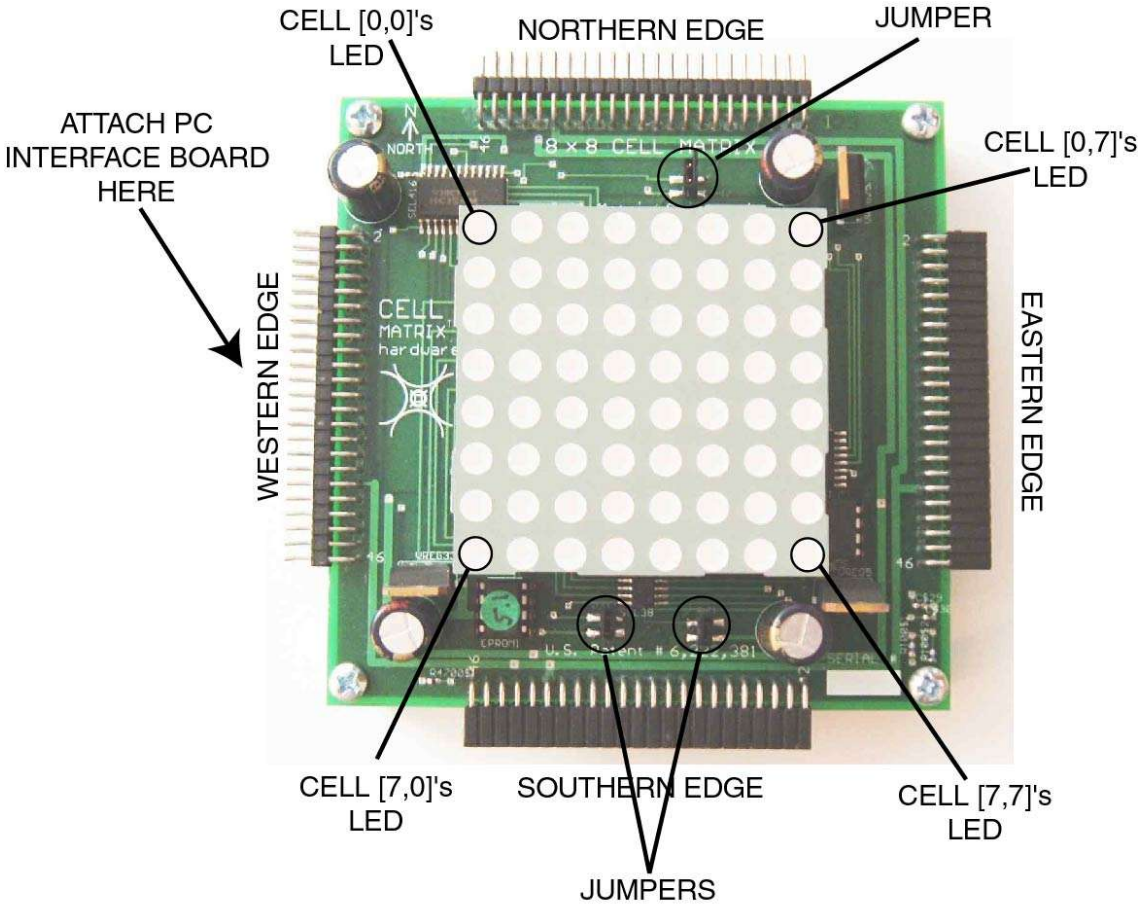


FIGURE 6 - CELL MATRIX MOD 88

The following precautions apply even if the *PC Interface Board* is used. You should avoid:

- Attaching the *Cell Matrix Mod 88* incorrectly to the *PC Interface Board*;
- Attaching two *Cell Matrix Mod 88s* to each other incorrectly;
- Physically damaging the board; and
- Applying static to the board.

There are probably other ways to damage the board. You should use common sense and treat the *Cell Matrix Mod 88* and *PC Interface Board* the way you would any other piece of sensitive electronic equipment.

Returning the Cell Matrix Mod88 or PC Interface Board

While Cell Matrix Corporation cannot be help liable for damage caused by mishandling the boards (for example, dropping a board, breaking off pins, or applying too much voltage to a connection), we may be able to repair repairable damage for a fee (for example, replacing an I/O connector). Please contact us at

mod88@cellmatrix.com

if you have a damaged board.

If you decide you no longer wish to use your boards, please do not discard them. We may be able to buy back used boards from you. Please contact us for more details.

The next chapter describes the hardware and software installation of the *Cell Matrix Mod 88* and optional *PC Interface Board*.

2.INSTALLATION

Hardware Installation

If you plan to use the *Cell Matrix Mod 88* as a standalone device, i.e., you plan to supply power, set inputs, read outputs, and so on using hardware of your own, then you can skip to **Chapter 7**. However, you may still find it useful to read the **TUTORIALS** chapter for ideas on how to interact with the *Cell Matrix Mod 88*.

If you plan to use the *PC Interface Board* to connect the *Cell Matrix Mod 88* to a PC, you will first need to perform the following Hardware Installation steps:

- Identify the Western edge of the *Cell Matrix Mod 88*, as shown in Figure 6;
- Connect the 46-pin header on the Western edge to the 46-pin socket on the *PC Interface Board*;
- Connect a parallel port cable between your PC and the *PC Interface Board*;
- Apply power to the *PC Interface Board*, either by attaching a 9V battery or by using an appropriate external power supply (6-12 V DC, center positive, 6.5mm/2mm plug)
- Make sure the Power Switch on the *PC Interface Board* is ON (flipped towards the top of the board). The Green Power LED should be illuminated (see Figure 7).

Software Installation

Be sure to read Appendix 4, LICENSE TERMS AND CONDITIONS, before installing or using any of the software described below.

See Appendix 2 for information on installing the graphical *Cell Matrix Layout Editor*^{TM}, which you can use to examine and create Cell Matrix configurations usable by the *Cell Matrix Mod 88*.

Depending on the platform/programming environment in which you wish to use the *Cell Matrix Mod 88*, you should skip to one of the following sections:

- if you are working in a Linux environment, skip to section 2.1;
- if you are working in a DOS environment, skip to section 2.2;
- if you are working in a Windows environment, skip to section 2.3.

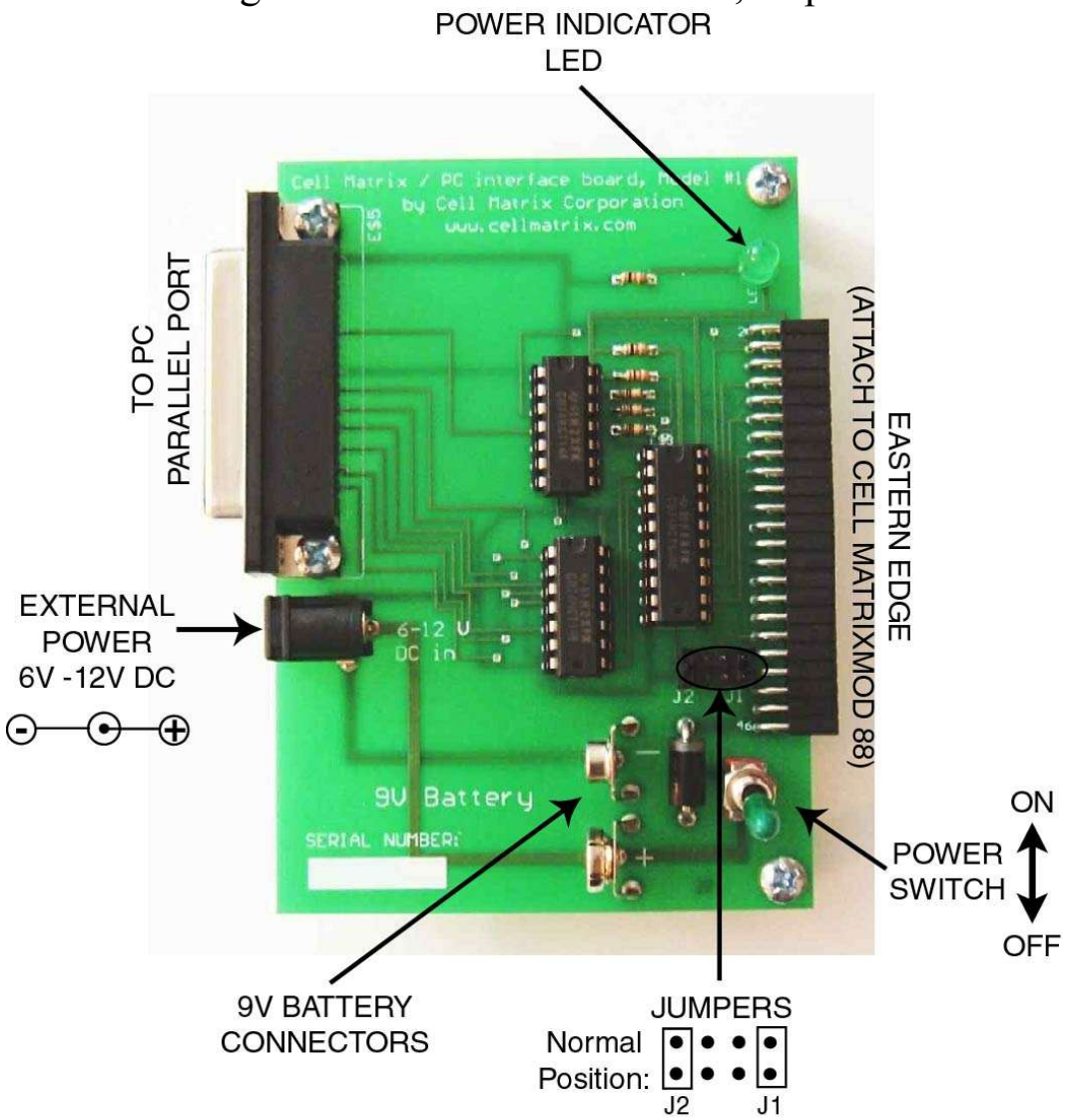


FIGURE 7 - PC INTERFACE BOARD
Jumpers J1 and J2 must be inserted as shown for normal operation.

2.1 LINUX INSTALLATION

There are two ways to use the *Cell Matrix Mod 88* in a Linux

environment:

- You can use the included Command Line Interface (CLI) to configure the Cell Matrix, send values to inputs, examine outputs, change clock states, and so on. This is a useful way to gain familiarity with both the *Cell Matrix Mod 88* the Cell Matrix architecture itself.
- If you wish to perform more complex operations with the *Cell Matrix Mod 88*, you can use the Application Programming Interface (API), along with a compiler/linker (gcc), to create C programs that configure the matrix, set inputs, read outputs, change clock states, and so on.

In either case, you should choose a work directory on your system, and copy the files from the CD:/EXAMPLES directory.

Command Line Interface

The CLI can be copied from the CD:/SOFTWARE/LINUX/main to whatever directory you wish to work in. Note that this program must either be run as user=root, or be setup to SETUID to root. This is necessary in order to execute the ioperm() command, which will grant R/W access to port 0x378, 0x379, 0x278 and 0x279, which are then accessed using outb() and inb().

Chapter 4 covers details of using the CLI. See Chapter 6, **TUTORIALS**, for examples of using the CLI.

Application Programming Interface

By using the Application Programming Interface (API), you can perform various operations on the *Cell Matrix Mod 88* via subroutine calls from a C program. You should copy the files api.h and api.o from the CD:/SOFTWARE/LINUX directory to your chosen work directory.

When writing a C program to interact with the *Cell Matrix Mod 88*, you should have a statement:

```
#include “api.h”
```

near the beginning of your program.

Your program should be linked to the api.o library. A typical command

for compiling and linking *yourprogram.c* would be:

```
gcc -o yourprogram yourprogram.c api.o
```

Chapter 5 covers programming details of using the API.

2.2 DOS INSTALLATION

There are two ways to use the *Cell Matrix Mod 88* in a DOS environment:

- You can use the included Command Line Interface (CLI) to configure the Cell Matrix, send values to inputs, examine outputs, change clock states, and so on. This is a useful way to gain familiarity with both the *Cell Matrix Mod 88* the Cell Matrix architecture itself.
- If you wish to perform more complex operations with the *Cell Matrix Mod 88*, you can use the Application Programming Interface (API), along with an appropriate compiler/linker, to create C programs that configure the matrix, set inputs, read outputs, change clock states, and so on. Currently only the Turbo C 2.01 C compiler (freely available from Borland) has been tested with the API.

In either case, you should choose a work directory on your system, and copy the files from the CD:/EXAMPLES directory.

Command Line Interface

The CLI can be copied from the CD:/SOFTWARE/DOS/main.exe to whatever directory you wish to work in. Chapter 4 covers details of using the CLI. See Chapter 6, **TUTORIALS**, for examples of using the CLI.

Application Programming Interface

By using the Application Programming Interface (API), you can perform various operations on the *Cell Matrix Mod 88* via subroutine calls from a C program. You should copy the files api.h and api.obj from the CD:/SOFTWARE/DOS directory to your chosen work directory.

When writing a C program to interact with the *Cell Matrix Mod 88*, you should have a statement:

#include “api.h”

near the beginning of your program.

Your program should be linked to the api.obj library. A typical command for compiling *yourprogram.c* would be:

tcc -c yourprogram.c

Uou would link this as follows:

tcc -eyourprogram yourprogram.obj api.obj

See the Borland Turbo C 2.01 tcc documentation for more details.

Chapter 5 covers programming details of using the API.

2.3 WINDOWS INSTALLATION

There are two ways to use the *Cell Matrix Mod 88* in a Windows environment:

- You can use the included Command Line Interface (CLI) to configure the Cell Matrix, send values to inputs, examine outputs, change clock states, and so on. This is a useful way to gain familiarity with both the *Cell Matrix Mod 88* the Cell Matrix architecture itself.
- If you wish to perform more complex operations with the *Cell Matrix Mod 88*, you can use the Application Programming Interface (API), along with an appropriate compiler/linker, to create C programs that configure the matrix, set inputs, read outputs, change clock states, and so on. Currently the API has been tested under a number of development environments: Cygwin/gcc; Borland Turbo C++5.5.1; and Visual Basic inside OpenOffice Calc.

In any case, you should choose a work directory on your system, and

copy the files from the CD:/EXAMPLES directory.

Port95NT Installation

Because many versions of Windows operating systems prevent direct access to the parallel port by user software, you must install a separate Parallel Port Driver called PORT95NT. You can install this by executing the program on the CD:/EXTRAS/PORT95NT/port95nt.exe and following the installation instructions given by that program. This is a one-time installation, and must be performed before any CLI or API operations will succeed. Note that PORT95NT is owned by Scientific Software Tools, Inc.

Windows XP Note

If you are running Windows XP, you may need to perform an additional setup step, depending on your system's specific configuration. If you notice erratic behavior by the *Cell Matrix Mod 88*, such as wrong outcomes when running the tutorials, it may be that your operating system is interfering with parallel port operations, mistakenly seeing port activity as indicating the presence of a printer. In this case, please see **APPENDIX 3** for instructions on how to disable this behavior.

Command Line Interface

The CLI can be copied from the CD:/SOFTWARE/WIN32/main.exe to whatever directory you wish to work in. Chapter 4 covers details of using the CLI. See Chapter 6, **TUTORIALS**, for examples of using the CLI.

Application Programming Interface

By using the Application Programming Interface (API), you can perform various operations on the *Cell Matrix Mod 88* via subroutine calls from your own program (written in C, Visual Basic, or other languages). There are three different environments in which you can interact with the API:

- If you work in the **CYGWIN** environment (a freely available UNIX environment that runs on top of Windows), you should copy the files api.h and api.o from the CD:/SOFTWARE/CYGWIN directory to your chosen work directory. You can then compile and link C programs

using the *gcc* command. For example, to compile and link a program called *yourprogram.c* you could use the command

gcc -o yourprogram yourprogram.c api.o

You should also have a line near the beginning of your program that says:

#include “api.h”

- If you work in the **Borland C++ 5.5.1** environment, you should copy the files api.h and api.obj from the CD:/SOFTWARE/WIN32 directory to your chosen work directory. You can then compile and link C programs using the *bcc32* command. For example, to compile and link a program called *yourprogram.c* you could use the command

bcc32 yourprogram.c api.obj

You should also have a line near the beginning of your program that says:

#include “api.h”

This may also work under other 32-bit compilers, but this has not been tested.

- If you are working under **any other 32-bit environment**, you will probably want to access the API using the Windows DLL calling protocol. To use the DLL, you should copy the files api.dll and api.h from the CD:/SOFTWARE/DLL32 directory to your chosen work directory. You then compile and link your programs in the usual way (depending on what programming language/environment/etc. you are using). Your calls to API routines are embedded inside your code using Windows routines to access the DLL. For example, in a C program, you would have the following commands to access the init_io() and reset() routines in the API:

```
#include <windows.h>
#include "api.h"
/* Declare variables for handling the DLL */
static HANDLE dllhandle; /* Pointer to DLL */
static FARPROC init_io,reset; /* Pointers to API routines*/

/* Load the DLL */
dllhandle=LoadLibrary("api.dll"); /* Load the DLL */
if (dllhandle == NULL) return(1); /* ERROR-Failed to load */

/* Get address pointers to DLL routines */
init_io=GetProcAddress(dllhandle,"init_io");
reset=GetProcAddress(dllhandle,"reset");

/* Call the init_io() routine */
(*init_io)(1); /* Initialize IO on LPT1: */

/* Call the reset() routine */
(*reset) ();
```

The calling protocol may be different for other languages or environments. For example, if you are working in Visual Basic, you would use a series of *DECLARE* statements to define the entry points into the DLL. Example 3, described at the end of Chapter 5, is a Visual Basic program that accesses the API via the DLL.

Chapter 5 covers programming details of using the API.

3.PC INTERFACE BOARD OVERVIEW

The *PC Interface Board* is used to attach the *Cell Matrix Mod 88* to a PC compatible computer, via a parallel port. The *PC Interface Board* also provides a handy way to supply power to the *Cell Matrix Mod 88*.

Normally, with a connection to the Western edge of a Cell Matrix, one could only interact with cells on that Western edge: cells on the North, South or East would be inaccessible. However, as a convenience, the *Cell Matrix Mod 88* contains additional (non-Cell Matrix) circuitry that, combined with the *PC Interface Board* and the appropriate software, allow you to write inputs and read outputs on all four sides of the Cell Matrix.

Figure 7 illustrates the various pieces of the *PC Interface Board*. Power can be supplied either through a 9 volt battery, or an external power jack (if a battery is connected and a power jack is then inserted, the battery will be automatically disconnected). External power can be any voltage, from 6V DC to 12 V DC. It does not need to be highly regulated, as the *Cell Matrix Mod 88* has its own voltage regulators. We offer a basic 6V power transformer, though most any from 6-12V will work (but be sure the polarity is correct). The plug is 2mm inside, 6.5 mm outside, center positive. **It is important that the proper sized plug is used.** If a plug with too small a diameter is used, the battery will not be disconnected when external power is applied. **This could cause the battery to explode!**

The current requirements of the boards are difficult to predict, since it depends on what configuration is loaded into the matrix. Typically, the *Cell Matrix Mod 88*/*PC Interface Board* combination requires a minimum of 100 mA while running, but may sometimes draw 2-3 times that amount (or more). Additional *Cell Matrix Mod 88s* seem to draw a minimum of 50 mA each. **NOTE** that the LED array is scanned, i.e., only one LED is illuminated at a time. Thus, the amount of current drawn to illuminate all 64 LEDs is roughly the same as that to illuminate a single

LED.

The 25-pin socket accepts a standard parallel port cable (25-pin male connector). Generally, the shorter the cable, the better the performance you'll get. A 3-6 foot (1-2 meter) cable should present no problems.

The switch controls the flow of power to the *Cell Matrix Mod 88*. When the switch is facing the middle of the board (away from the bottom edge), current is flowing. If a *Cell Matrix Mod 88* is attached, the *Cell Matrix Mod 88* will supply +5V back to the *PC Interface Board*, causing the green Power LED to light up. This indicates that power is being supplied to the *Cell Matrix Mod 88*. **NOTE** that attaching the parallel port to a PC may also cause the Power LED to light up.

The *PC Interface Board* Eastern Connector attaches to the WESTERN edge of the *Cell Matrix Mod 88*, as shown in Figure 8. The Western edge can be identified by the words “CELL MATRIX hardware” along the edge.

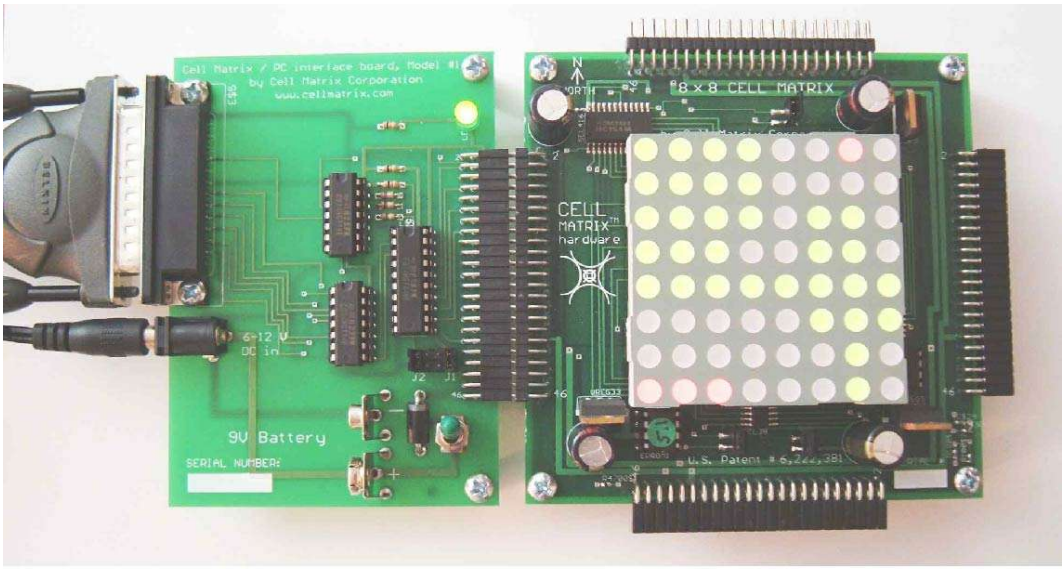


FIGURE 8 - CELL MATRIX MOD 88 AND PC INTERFACE BOARD CONNECTED TOGETHER

WARNING: Incorrectly hooking the *PC Interface Board* to the *Cell Matrix Mod 88* will result in permanent damage to the *Cell Matrix Mod 88*. Make sure the boards are connected properly before supplying power

to the *PC Interface Board* or attaching the parallel port to a PC.

If multiple *Cell Matrix Mod 88s* are hooked together, the *PC Interface Board* can be attached to the Western edge of any board. You should never have more than one *PC Interface Board* attached to a collection of *Cell Matrix Mod 88s*.

Standalone Operation Following Bootstrap

You may wish to use the *PC Interface Board* to bootstrap the *Cell Matrix Mod 88*, but then detach the *PC Interface Board* from an external PC. To do so, follow this procedure:

1. with the parallel port connected to the PC and the *PC Interface Board*, follow whatever bootstrap procedure you desire, to configure the *Cell Matrix Mod 88* as you like;
2. remove Jumper J1 from the *PC Interface Board*;
3. remove Jumper J2 from the *PC Interface Board*; and
4. remove the parallel port connector from the *PC Interface Board*.

Note that there are two unused pairs of pins in between the pins labeled J1 and J2. These unused pins can be used to park the jumpers once you have removed them.

To re-attach the *PC Interface Board* to a PC, simply reverse the above steps.

Separating Boards

It is important that you use care when separating two boards from each other, to avoid bending the pins. A wide-bladed screwdriver can be used, as shown in Figure 9. Place the blade between the boards, and twist a small amount, to slightly separate the connectors on that edge. Then do this on the other side of the connector. Repeat, alternating side to side, until the connectors can be easily separated by pulling the boards apart.

Be careful not to disturb other components on the boards while doing this. Note also that for the boards to work properly, they do not necessarily need to be connected as tightly as possible. If you leave them partially separated (but still making a good electrical connection), they will be easier to take apart later.

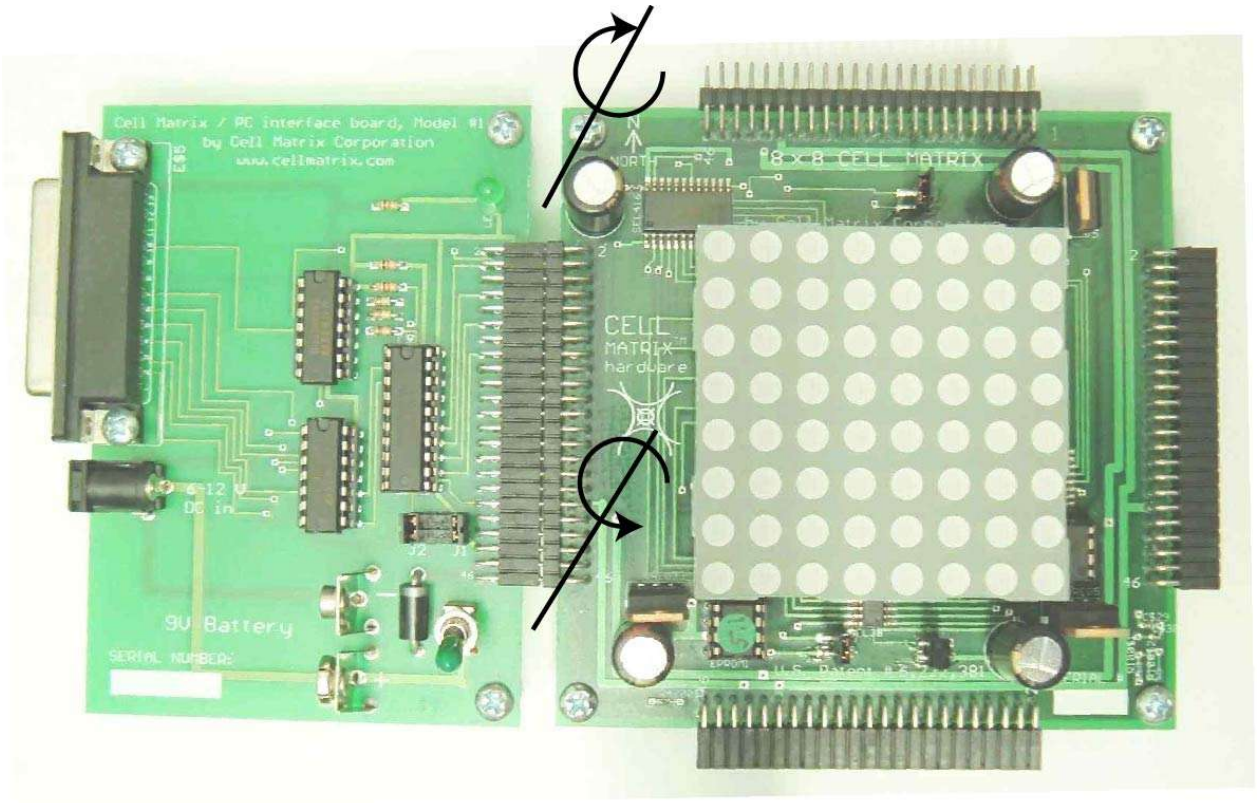


FIGURE 9 - SEPARATING THE BOARDS
Use a wide-bladed screwdriver or similar object, and a twisting action in the locations shown. By alternately working in the two spots shown, the boards can be separated without using extreme force. Be careful not to damage or disturb other components on the boards.

4.COMMAND LINE INTERFACE

An executable program ("main" or "main.exe") is supplied for interacting directly with the *Cell Matrix Mod 88 + PC Interface Board*, via a simple command-line interface. See Chapter 2 for instructions on installing the Command Line Interface for your specific system.

Before using the program, the *PC Interface Board* should be attached to the *Cell Matrix Mod 88*, the parallel port of the *PC Interface Board* should be attached to the PC's parallel port, and power should be supplied to the *PC Interface Board*. The switch should be flipped on (towards the middle of the board), and the green LED should be lit.

To load the program, copy main or main.exe from the included CD to your hard drive. To start the program simply double click on the file.

If you execute the main program from a command line (DOS, Cygwin, etc.) you can specify an optional argument, which is the name of a command file. Lines from this file are executed as commands, after which commands are taken from the keyboard.

NOTE: make sure you have reviewed all the instructions in Chapter 2 (Installation) before attempting to run this program, including the chapters labeled **Port95NT Installation** and **Windows XP Note**.

The program begins with a prompt "> " This indicates that it is waiting for you to enter a command. You will most likely want to begin by issuing the command "r" to reset the *Cell Matrix Mod 88* to its power-up state (all cell truth tables filled with zeros, so cells simply perform NoOps and output only 0s).

The following describes the available commands for the interface program. **NOTE that all commands are case-sensitive.**

Initialization/Exiting:

- **r** – Reset the Cell Matrix. All inputs are set to 0. All truth tables are cleared (loaded with all zeros). This returns the matrix to its initial power-up state.
- **Q** – Exit program immediately
- **I {1 or 2}** – (this command is an uppercase "i") Selects parallel port to use for communicating with the *PC Interface Board*. "I 1" selects LPT1: and "I 2" selects LPT2:
  LPT1: is the default.

CELL INPUT AND OUTPUT:

- **s row col side value** – Set an input to the Cell Matrix. [row,col] identifies the cell whose input is being set. [row,col] refers to cells in the matrix of the board directly connected to the *PC Interface Board*. See Figure 4 for the ordering of [row,col]. "side" is one of DN, DS, DW, DE, CN, CS, CW or CE for (respectively) Data North, South, West, East, or Control North, South, West or East inputs. "value" is 0 or 1
- **x row col side** – Display an edge cells' output value. [row,col] identifies the cell being examined, and "side" is as described above in the "s" command. Only edge outputs may be displayed, i.e., for cell [0,2], only the Northern outputs may be displayed. As with the "s" command, [row,col] refers to cells in the matrix of the board directly connected to the *PC Interface Board*.
- **x** – Display all D outputs of all edge cells

Clock Control:

- **t** – Tick the system clock (Raise Phi 1, drop Phi 1, raise Phi 2, drop Phi 2)
- **t num** – Tick the system clock "num" times
- **p num** – Set a delay between all commands sent to the *PC Interface Board*. This is useful for effectively slowing down the rate at which the LEDs change. For this command, the scale is machine dependent, but 0 is no delay, 10000 is typically large.

Command Files:

It is possible to store a number of commands in a file, and then execute those commands by referring to the file. Three commands are useful for interacting with such command files:

- **@filename** – execute the commands contained inside the file called "filename"
- **P** – pause, waiting for user input. Typically this command is used inside a command file. It allows one to study the state of the output LEDs (for example) before the command file continues. Hitting ENTER will allow the command file execution to continue.
- **q** – If the pause command ("P") is used inside a command file, hitting q followed by ENTER will terminate execution of the command file.

Note that command files may be nested, up to 128 deep. In other words, one command file may include "@" commands to execute other command files.

The command "@con:" allows one to enter commands during execution of a command file. This is useful, for example, after loading a particular configuration via command file instructions: "@con:" would then allow the user to set inputs, examine outputs, and generally interact with the current configuration. ^Z would terminate the @con: input, allowing the command file to continue executing.

@con: is Windows-specific. For CYGWIN, the equivalent command would be "@/dev/tty"

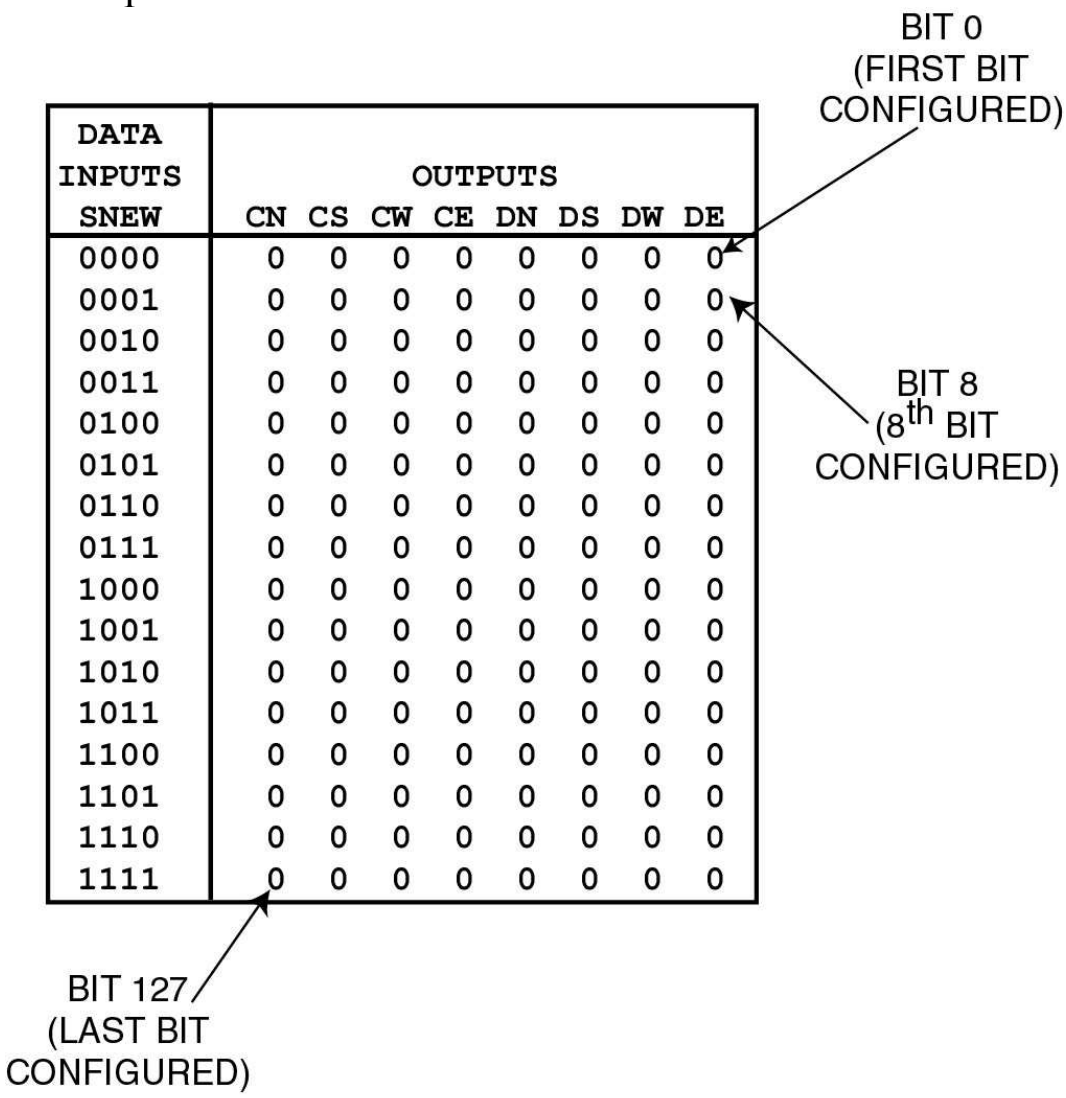
An initial command file can be executed when the Command Line Interpreter is invoked by specifying the command file as an argument, i.e.

```
main.exe comfile
```

Configuration Commands:

Figure 10 shows a sample Cell Matrix truth table. On a Cell Matrix, there

is no way to directly load a cell's truth table. Since all that is accessible from outside the matrix is the C and D inputs of edge cells, these must be used to manipulate a cell's truth table.



DATA INPUTS SNEW	OUTPUTS CN	CS	CW	CE	DN	DS	DW	DE
0000	0	0	0	0	0	0	0	0
0001	0	0	0	0	0	0	0	0
0010	0	0	0	0	0	0	0	0
0011	0	0	0	0	0	0	0	0
0100	0	0	0	0	0	0	0	0
0101	0	0	0	0	0	0	0	0
0110	0	0	0	0	0	0	0	0
0111	0	0	0	0	0	0	0	0
1000	0	0	0	0	0	0	0	0
1001	0	0	0	0	0	0	0	0
1010	0	0	0	0	0	0	0	0
1011	0	0	0	0	0	0	0	0
1100	0	0	0	0	0	0	0	0
1101	0	0	0	0	0	0	0	0
1110	0	0	0	0	0	0	0	0
1111	0	0	0	0	0	0	0	0

FIGURE 10 - ONE CELL'S TRUTH TABLE
Shown in initial (all zero) configuration.
Note that all inputs are D inputs from North, South, West or East.
Outputs are both C and D to North,South, West or East.

Cell configuration proceeds as follows:

- First, a cell's C input is set to 1. This places the cell into “C-mode.” The side on which the C input is high is called the “active side.”
- Once a cell is in C-mode, the D output on the active edge is set to the

current value of bit 0 in the cell's truth table. Other D outputs are set to 0. All C outputs are also set to 0. It is at this point that the D input on the active side should be set to the desired value of bit 0.

- When Phi 1 is raised, the D input on the active side is loaded into the cell's truth table at bit 0.
- When Phi 1 drops, the cell's internal truth table address is incremented to bit 1.
- Phi 2 is raised.
- When Phi 2 is dropped, the D output on the active side is set to the current value of bit 1 in the cell's truth table.

These last four steps repeat on each set of clock transitions, with the bit number incrementing each iteration (from 0 to 127).

This mechanism thus allows one to write a cell's truth table by meticulously controlling the C and D inputs, as well as the system clock lines. This is a simple but cumbersome procedure to perform by hand.

To simplify this often-used set of operations, the following commands are available:

- **c** – compile a set of **Boolean equations** into a truth table. The user is prompted to enter the equations, ending with a line containing a single character (“.”) Equations have the form
  side=expression
  where side is one of DN, DS, DW, DE, CN, CS, DW or CE, specifying which output side is being defined; and
  expression is a combination of operands (N, S, W, E, 1, 0) and operators (+ for OR, & for AND, ~ for INVERSION). Each equation is entered on a separate line. See Tutorials 6-9 for examples of the **c** command.

Expressions are entered in Reverse Polish Notation (RPN). RPN is also called *postfix* notation, because the operator appears after the operand(s). For example, whereas one would normally write “A+B” the RPN expression would be “AB+” i.e., the operator “+” appears after both operands “A” and “B.” Therefore, to write the equation meaning

“Data North Out is assigned the logical AND of Data South In and Data East In”one would write “DN=SE&” Other examples are shown below. Note that while outputs can be D or C outputs, the operands are always DATA (D) inputs. The notion of, say, ANDing two C inputs makes no sense in the Cell Matrix architecture.

EXAMPLES OF EQUATIONS

- DN=1 means DNout is 1 regardless of the values of any D inputs
- DW=E means DWout is the same as the value of DEin. This is a simple wire from East->West
- DW=E~ means DWout is the inverse of DEin
- DN=SE+ SW+ & means (S.OR.E) .AND. (S.OR.W) --> DN
- CN=NW~& means N.AND(.NOT. W) --> CN
- DE=WESN+++ means W.OR.E.OR.S.OR.N --> DE
- DE=WE+S+N+ also means W.OR.E.OR.S.OR.N-->DE

and so on.

The compiled truth table is stored in a temporary location inside the program.

- **d** – Dump a copy of the stored truth table to the display. This allows you to examine the truth table that results from the equations entered following a “c” command
- **l row col side** – (this command is a lowercase “L”) Load the truth table into a cell. [row,col] identify the position in the Cell Matrix of the cell being loaded. “side” is DN, DS, DW or DE, indicating which D input should receive the configuration bits. The command causes the following operations to occur:
    - the cell's D input is set to the value of bit 0 of the stored truth table;
    - the system clocks are ticked;
    - the cell's D input is set to the value of bit 1 of the stored truth table;
    - the system clocks are ticked;

and so on, until all 128 bits of the stored truth table have been sent to the

cell's D input. The command returns after 128 ticks of the system clocks.

See the Tutorial chapter below for examples that use these commands to configure a cell's truth table. Note that this is only useful for configuring edge cells. To configure cells internal to the matrix, one must use edge cells to configure non-edge cells.

- **b filename** – read a .bin file (as produced by the *Cell Matrix Layout Editor*^{TM}), and generate bootstrap sequences for configuring the *Cell Matrix Mod 88* as specified by the .bin file. The smallest file that can be bootstrapped with this command is 3x3.

  The behavior of the **b** command depends critically on the configuration you are attempting to bootstrap. In particular, note that some configurations **cannot** be bootstrapped with this command: it does no analysis on the configuration being bootstrapped, and thus is susceptible to interference from the circuit being bootstrapped, in the case where that circuit is asserting C outputs. Note also that some circuits cannot be bootstrapped by **any** method. For example, if a cell is outputting 1 on its Western C output, then that cell *cannot* be configured from the East, since any cell on the East will be placed in C-mode, and C-mode cells can never assert their C outputs. Of course, the cell might still be configurable from the North, South or East. If, however, a cell asserts its C outputs to all four sides, then it is impossible to reconfigure that cell from within the matrix. Only a system-wide reset will clear the truth table of a cell in such a state.

Miscellaneous Commands:

- **!** - Repeat the last interactive command. This is a one-command history recall. Commands inside command files do not affect the history, i.e., if you execute the command "@filename" then "!" will re-execute the command "@filename"
- # - comment. Entire line is ignored
- **F** – Force full command transmission/disable command cache. This will slow down the interaction with the *PC Interface Board.* Under

normal circumstances, you should not need to use this command, unless you are using two programs to interact with the *PC Interface Board* or power is interrupted to the boards while a program is running.

- **?** - show brief Help screen. Same as entering an unknown command
- **vd name row col side inc len** – define a "vector" of bits. A vector can later be set with the "v" command, to cause a number of bits to be set simultaneously. This is useful, for example, for inputting or examining integers. The arguments are as follows:

"name" is the user-defined name for the vector.
[row,col] specifies the cell corresponding to the Least Significant Bit (LSB) of the vector (row and col are each integers between 0 and 7, inclusive).
"side" specifies the side of cell [row,col] from which bits will be read/written (usually one of "DN" "DS" "DW" or "DE")
"inc" specifies how many cells to move from bit to bit. A value of 1 means consecutive cells correspond to consecutive bits in the vector, a value of 2 means alternate cells correspond to consecutive bits in the vector, and so on.
"len" specifies the number of bits in the vector.

For example, suppose you wish to treat the following four D inputs as a four-bit integer:

[7,7]DS=Bit 0 (LSB)
[7,6]DS=Bit 1
[7,5]DS=Bit 2
[7,4]DS=Bit 3 (MSB)

The following vector command would define a vector called "X" accordingly:

vd X 7 7 DS -1 4

If instead you wanted alternating cells, i.e.,

[7,7]DS=Bit 0 (LSB)
[7,5]DS=Bit 1
[7,3]DS=Bit 2
[7,1]DS=Bit 3 (MSB)

you would use the command:

vd X 7 7 DS -2 4

and so on. Again, see the Tutorials chapter for more examples.

- **v name** – Display the value of an output vector called “name” In the above example, the command “v X” would display the integer value formed from the four D outputs specified in the “vd” command
- **v name=value** – Set the value of an input vector to the indicated integer value. In the above example (“vd X 7 7 DS -1 4”), the command “v X=13” would assign D inputs as follows (note that 13=1101 in binary):

  [7,7]DS=1
  [7,6]DS=0
  [7,5]DS=1
  [7,4]DS=1

This is a convenient way to set a number of bits simultaneously in order to define, say, an integer input value.

5.API INTERFACE

If you wish to interact with the *PC Interface Board*/*Cell Matrix Mod 88* in a more programmed fashion that with the CLI, you can do so using one of the API interfaces supplied with the board. An API is a set of callable routines that allow you to send commands to the *Cell Matrix Mod 88* via your own program. As with the CLI, the API allows you to set inputs, read output, affect the system clock, and so on. However, unlike the CLI, you can write programs with the API to act conditionally based on detected output values, to perform complex calculations and send the results to the matrix, and so on. You can also use the API to load cell configurations into the PC's memory and then bootstrap the matrix using those values.

API Files

The supplied API files can be used to call routines in a variety of ways:

- api.dll is a Windows DLL of routines, which can be called using standard DLL calling procedures.
- VB Header.txt is a set of Visual Basic DECLARE statements for api.dll **Note** that VB Header.txt does more than just define DLL entry points: it also defines the routines raw_comp() and boot_mem() themselves. See the examples at the end of this chapter for calling raw_comp() and boot_mem() from VB.
- api.h includes descriptions of the callable routines in api.obj and api.o. you can #include “api.h” to define prototypes for these routines
- api.obj can be linked to programs compiled with Borland's Turbo C compilers, using a command such as:

  bcc32 yourprogram.c api.obj (Turbo C++5.5.1) or
  tcc yourpro.c api.obj (Turbo C 2.01)
- api.o can be linked to programs compiled under gcc (Cygwin or Linux), using a command such as:

  gcc -o yourprogram yourprogram.c api.o

See the **INSTALLATION** chapter for details on choosing which API

files are appropriate for your environment.

API Routines

There are 5 categories of API calls: Initialization, Cell I/O, System Clocking, Truth Table Manipulation, and Bootstrapping.

Initialization:

int init_io(int port) – This routine must be called before any other API calls. It serves three main functions:

1. it defines the parallel port on which I/O to the *PC Interface Board* will take place;
2. it initializes the routines used to communicate with the parallel port (dllportio); and
3. it initializes various internal structures.

The argument *port* should be 1 (for lpt1:) or 2 (for lpt2:). Any other arguments are illegal.

The return value is 0 for success, 1 for error (meaning an illegal argument, or a problem loading dllportio)

int reset(void) – This call causes a RESET signal to be sent to the Cell Matrix, followed by 256 clock ticks. It also invalidates all address cache entries inside the interface software. You can call this routine whenever you want to reset the hardware. The return value is always 0.

Cell Input/Output*:*

These routines are used to write and read C and D lines on edge cells. It should be noted that on a standard Cell Matrix, one could only directly address I/O lines to which you are directly connected. However, the *PC Interface Board*/*Cell Matrix Mod 88* are designed to work together to

provide access to **all** perimeter I/O lines. Thus, for example, you do not need to physically connect to the Northern edge of the *Cell Matrix Mod 88* in order to read or write C and D lines of cells on the Northern edge of the matrix. This is one big advantage of using the *PC Interface Board*.

int set_input(int row, int col, char *side, int value) – This routine sets the value of a C or D input line. *row* and *col* specify the cell being addressed. *side* must be one of the following two-character mnemonics:

dn – Data North
ds – Data South
dw – Data West
de – Data East
cn – Control North
cs – Control South
cw – Control West
ce – Control East

Note that the value of *side* restricts the value of *row* or *col*:
Northern inputs are set only on cells where *row*=0;
Southern inputs are set only on cells where *row*=7;
Western inputs are set only on cells where *col*=0; and
Eastern inputs are set only on cells where *col*=7.

value is 1 or 0, and specifies the value to which the input is to be set.

The return value is 1 if an illegal *side* or an illegal combination of *side*, *row* and *col* is specified. Otherwise the return value is 0. Note that the software has no way of knowing if the command succeeded, only that the command was sent.

int get_output(int row,int col,char *side) – This routine is used to read a single output. *row, col* and *side* are exactly the same as defined above. The return value of the function is -1 if an illegal *side* or an illegal *side/row/col* combination is specified. Otherwise the return value is 1 or 0, and is the value of the specified output.

int enable_cache(int state) – In order to maintain a high I/O rate, the software will eliminate what it considers to be unnecessary I/O

operations. If you are interacting with the *PC Interface Board* using more than one program, or if you power-cycle the *PC Interface Board* or *Cell Matrix Mod 88* without subsequently calling reset(), future I/O operations may fail. This is due to internal caching of the system's state. You can disable this caching by calling enable_cache(0). However, disabling the cache may significantly slow down I/O performance.

If you wish to simply flush the cache, call enable_cache(0) followed by enable_cache(1). This can be used whenever you power-cycle the system, or another piece of software accesses the *PC Interface Board*.

System Clocking:

These routines are used to manipulate the Cell Matrix's system clock. Note that when a cell is in D-mode, the clock has no effect on that cell *per se*: the system clock only affects cells that are in C-mode.

int clock_tick(void) – This routine takes no arguments, and always returns a 0. It causes the generation of a single clock cycle:

1->phi 1, 0->phi 1, 1->phi 2, 0->phi 2

This will cause all C-mode cells to sample their inputs, load sampled inputs into their internal truth tables, move the the next bit in their truth tables, and send the newly selected truth table bit to their D outputs.

int setpause(int delay) – This routine can be used to slow down execution of *PC Interface Board* commands. The larger the value of *delay*, the longer the commands will take. If *delay*=0, no delay is imposed. Typical values are around 10000-100000 for a 1 GHz CPU. This routine is mainly useful for observing the behavior of the board as various commands are executed – particularly Truth Table Manipulation and Bootstrapping commands.

Note that if *delay* is set to too large a value, commands may take an extremely long time to execute. There is no clean way to interrupt such a command. You may interrupt your program in the usual way (^C, etc.), but the *Cell Matrix Mod 88* will likely be left in an indeterminate state,

requiring a subsequent *reset()* command.

Truth Table Manipulation:

These routines allow one to send sequences of bits to a single input, while ticking the system clock in between the setting of each bit value. This provides a convenient means for configuring edge cells.

int comp(char *eqn) – This routine compiles one or more Boolean equations into an internal truth table format. Each equation is specified in Reverse Polish Notation, as described above in the description of the Command Line Interface's "c" command (Chapter 4, "Command Files" section). Multiple equations are separated by semicolons or commas.

After successful execution, the truth table corresponding to the given equations will be stored in an internal location, until a subsequent call to comp().

The return value is 0 for success, 1 for a syntax error in the equation(s).

int tt_send(int row,int col,char *side) – This command sends the internally-stored truth table (generated by a previous call to *comp()*) to the indicated input, and ticks the system clock in between each of the 128 bits. *row, col* and *side* are as defined above in the *set_input()* call, and the same restrictions on their values apply.

A typical programming sequence would be as follows:

- Set an edge cell's C input to 1 (say the cell's address is [R,C]);
- call comp() to compile a desired truth table;
- call tt_send(R,C,*side*) where *side* is the D input corresponding to the C input previously set to 1; and
- set the C input back to 0.

Following these steps, the given cell will generally be configured with the compiled truth table (unless, for exmple, another cell is simultaneously configuring cell [R,C]). The use of these API routines is

similar to the use of the corresponding CLI commands in Tutorials 6-9 (Chapter 6).

The return value is 0 for success, 1 for any detectable error (generally an illegal combination of *row*, *col* and *side*, e.g., a non-edge cell).

Bootstrapping:

These routines are used to configure a Cell Matrix with a set of truth tables. It should be noted that bootstrapping is a far-reaching concept, having a wide degree of flexibility and a wide range of possible complications. Some configurations simply cannot be bootstrapped, for example, a collection of cells each outputting ones on all their C outputs. Note also that for these bootstrap routines, the smallest file that can be bootstrapped is 3x3.

There are numerous ways to bootstrap Cell Matrices, since bootstrapping is essentially a user-defined, software-level construct. The bootstrap algorithm used in this API is good for most simple circuits. Circuits that are very dynamic in nature, i.e., those that are configuring other cells, may require custom bootstrap sequences.

int boot_bin(char *fname*) - This is the simplest way to bootstrap a Cell Matrix. *fname* is the name of a *binary grid file*, as created by the "Write Binary Grid" command in the *Cell Matrix Layout Editor*^{TM}. *boot_bin* will read the binary description, and send a series of commands to the *PC Interface Board* to cause the *Cell Matrix Mod 88* to be configured as specified in the binary file *fname*.

Binary files have a size associated with them, based on the size of the grid in use when the binary file was written from the *Cell Matrix Layout Editor*^{TM}. Note the following:

- bootstrapping begins at location [0,0], i.e., the upper-left cell. A 4x4 binary file, for example, would be loaded between cells [0,0] and [3,3];
- the smallest file that should be loaded is 3x3. This is not a hard restriction, but it's a special case that requires a different bootstrap

sequence;
- files larger than 8x8 may be bootstrapped exactly like smaller ones. If you have multiple *Cell Matrix Mod 88s* connected together, you can bootstrap larger binary files;
- generally, if you try to bootstrap a file larger than the Cell Matrix you are working with, part of the file will be loaded, and the rest will be ignored. If you send this command to an irregularly-shaped collection of multiple boards connected to each other, you may also be able to bootstrap those boards that are present, depending on the exact shape;
- If *fname* is a binary file for a matrix smaller than the Cell Matrix you are working with, the bootstrap will generally only affect those cells within the bounds of *fname*. In other words, if *fname* specifies a 4x4 configuration, only cells [0,0]-[3,3] will be affected by this call. Cells beyond row 3 or column 3 will be unaffected;
- you can use the *setpause()* call to slow down I/O with the *PC Interface Board*. This will let you observe the bootstrapping behavior.

The command returns 0 for success, 1 for error (generally “File Not Found”).

int raw_comp(char *eqn,int *tt) – This command is used to compile the given equations *eqn* into a 16-element integer array *tt*. *tt*'s elements can then be used in a subsequent call to *boot_mem()* (see below).

int boot_mem(int *tts,int rows,int cols) – This command is similar to *boot_bin()*, but instead of reading the configuration truth tables from a file, they are read from memory. The specific configuration is supplied in the *tts* parameter, which is an array of integers. Each cell's truth table occupies 16 locations on the *tts* array. Successive cells' truth tables are stored, one after the other, in row-major order. Thus, for a 4x4 configuration, *tts* is composed as follows:

tts[0]-tts[15] = Cell [0,0] (row 0, column 0)
tts[16]-tts[31] = Cell [0,1] (row 0, column 1)
tts[32]-tts[47] = Cell [0,2] (row 0, column 2)
tts[48]-tts[63] = Cell [0,3] (row 0, column 3)

tts[64]-tts[79] = Cell [1,0] (row 1, column 0)
tts[80]-tts[95] = Cell [1,1] (row 1, column 1)
tts[96]-tts[111] = Cell [1,2] (row 1, column 2)
tts[112]-tts[127] = Cell [1,3] (row 1, column 3)

tts[128]-tts[143] = Cell [2,0] (row 2, column 0)
tts[144]-tts[159] = Cell [2,1] (row 2, column 1)
tts[160]-tts[175] = Cell [2,2] (row 2, column 2)
tts[176]-tts[191] = Cell [2,3] (row 2, column 3)

tts[192]-tts[207] = Cell [3,0] (row 3, column 0)
tts[208]-tts[223] = Cell [3,1] (row 3, column 1)
tts[224]-tts[239] = Cell [3,2] (row 3, column 2)
tts[240]-tts[255] = Cell [3,3] (row 3, column 3)

rows and *cols* specify the dimensions of the configuration to be bootstrapped. The same comments from *boot_bin()* generally apply to *boot_mem()*, as far as dimensions of the configuration, etc. The return value of *boot_mem()* is generally 0, since it will bootstrap whatever it finds in memory, even if it is not what you intended it to bootstrap. Note that very little checking is done on these arguments. While you cannot damage the *Cell Matrix Mod 88* by loading an unintended configuration, you may need to *reset()* the board afterwards to clear out what was loaded.

API Examples

The CD:/EXAMPLES directory on the CD contains source code for four examples of using these API calls. Example 1 shows basic initialization calls, loading of truth tables into cells, setting of inputs, and examination of outputs. There are two versions of this example: simple1.c calls routines inside api.obj, while simple2.c uses the standard C DLL calling methods.

Example 2(adder.c) shows how the Cell Matrix can be used to add two integers.

Example 3 (bootdemo.txt) is a Basic program that demonstrates calling routines in the API's DLL from Basic. This example loads cell configurations into memory and uses them to bootstrap the Cell Matrix.

Example 4 (evolve.c) demonstrates the concept of Evolvable Hardware, configuring the Cell Matrix with random configurations, examining their behavior on a given task (5-bit odd parity generation), combining the best configurations to produce new ones, and eventually developing a circuit that performs the given task perfectly. This example can be modified to perform other evolutionary experiments, i.e., evolving different functions, applying different fitness measures, and so on.

6.TUTORIALS

The following tutorials are intended to provide a good introduction to the *Cell Matrix Mod 88* and the Command Line Interface. These tutorials assume you have already installed the software, including any necessary drivers, etc. These tutorials also assume you are using LPT1: as the parallel port for communicating with the *PC Interface Board*. If you are using LPT2:, you need to issue the command

I 2

once you begin running the Command Line Interface.

NOTE that the Command Line Interface is case sensitive.

You do not need to follow these tutorials exactly, or even approximately. They are intended only as examples of how to use various interface commands. You should feel free to experiment with these commands however you like. Remember, **you cannot physically damage the *Cell Matrix Mod 88* by "incorrectly" programming it.** While you can configure the board in ways that cannot be changed via C- and D-line manipulations, you can always issue the Reset ("r") command to restore the Cell Matrix to its power-up state. You should therefore feel free to experiment, try different things, or try random things. If you wonder what a certain set of commands might do, give them a try!

The following summary lists the basic commands available in the Command Line Interface. See Chapter 4, **COMMAND LINE INTERFACE** for more details.

QUICK COMMAND SUMMARY

r=Reset cell matrix
s row col [c|d][n|s|w|e] [0|1]=Set input
x row col [c|d][n|s|w|e]=Read output
x=read all
t (or t #)=Tick clock (# of cycles)
c=Compile truth table
d=Dump latest truth table

p n=set inter-command pause to n
P=Pause/optional abort of command file
vd name row col side inc len=Define vector
v name [val]=show [set] vector
l row col [c|d][n|s|w|e]=Step clock 128 cycles, loading latest TT
f toggle auto-freerun flag
S fname=Run a sequence file
I num = Select LPT{num}: for I/O to *PC Interface Board*
b fname=bootstrap Matrix from a .bin file
F = force full address definition (cache dsable)
@filename executes commands from filename
q=abort command file execution
! repeat last interactive command
Q=Exit driver

TUTORIAL 1 – Basic connection and powerup

First, make sure the power switch on the *PC Interface Board* is OFF (positioned towards the bottom of the *PC Interface Board*). Connect the *PC Interface Board's* 46-position edge connector to the WESTERN edge of the *Cell Matrix Mod 88*, as shown in Figure 8. Connect the PC's parallel port cable to the *PC Interface Board's* 25-position socket and o the PC's parallel port. Finally, apply power to the *PC Interface Board*, either through the DC power plug or by using a 9V battery.

Once everything is connected, flip the power switch ON (towards the middle of the *PC Interface Board*). Two things should happen:
(1) all the LEDs on the *Cell Matrix Mod 88* should momentarily turn orange, and then go dark; and
(2)the green LED on the *PC Interface Board* should light up and stay lit. If these occurred, then you have successfully connected the boards to the PC. If these did not occur, remove power immediately, and re-check all your connections.

NOTE: The green LED on the *PC Interface Board* will not illuminate

unless the *Cell Matrix Mod 88* is correctly connected to it.

TUTORIAL 2 – Basic Command Line Operations

Begin the Command Line Interface ("main" or "main.exe"). See Chapter 2, **INSTALLATION** for details on where to find the Command Line Interface for your particular environment. **NOTE:** You should copy the .BIN and .GRD files from the \EXAMPLES directory on the CD into whatever directory you are running the Command Line Interface from. This will prevent your needing to specify full pathnames for these files.

Once the program starts, you will see a header message, followed by the command line prompt ">"

Enter the command

s 0 0 cw 1

As with most commands, the first token ("s") indicates the type of command: "Set an input" in this case. The "0 0" specifies a cell, by Row (# of cells from the top) and Column (# of cells from the Left). "cw" specifies a specific input in the cell: the C West input. "1" is the value to which the input is being set.
The upper left LED should turn red, indicating that cell [0,0] is now in C-mode. Enter the command

s 7 2 cs 1

and now the LED corresponding to cell [7,2] is red, indicating that cell [7,2] is also in C-mode. Enter the command

s 0 0 cw 0

to return cell [0,0] to D-mode, as indicate by the LED no longer being red. Enter the command

r

to reset the Cell Matrix. Note that the "r" command also resets all inputs (D and C) to 0, so now cell [7,2] is again in D-mode.

You can also set D inputs to 1, by issuing a command such as:

s 0 0 dw 1

However, there is no visible change in the LED display. Cells are

initially programmed with all 0s in their truth tables, so even if you set a D input to 1, all the cell's D outputs are still 0, and thus the LED remains dark.

Enter the command

s 4 4 dw 1

and you'll see the error message:

"No access to [4,4]dw"

indicating that you cannot set the specified input. Only edge inputs on edge cells can be set with the "s" command.

TUTORIAL 3 – Loading a binary file

NOTE REGARDING .GRD and .BIN FILES

The EXAMPLES directory on the CD includes a set of .BIN and .GRD files. .GRD files are used with the *Cell Matrix Layout Editor™* (included on the CD under the LAYOUT\WINDOWS or LAYOUT\LINUX directory) to graphically define and edit Cell Matrix configurations. Once a layout has been defined, the *Cell Matrix Layout Editor™* can create a .BIN file describing the layout. Those .BIN files can be used by the *Cell Matrix Mod 88* API and the Command Line Interface. The following tutorials utilize these .BIN files. See Appendix 2 for details on how to install and setup the *Cell Matrix Layout Editor™*.

Enter the "r" command to reset the matrix. Now issue the command

b cross.bin

to load the binary file "cross.bin" into the matrix (if cross.bin is not in your current directory, you must specify the full or relative path to it). You can look at cross.grd in the *Cell Matrix Layout Editor™* to see how the individual cells are configured.

You will see a pattern of illuminated LEDs sweep across the display, as the truth tables contained in cross.bin are loaded into the Cell Matrix. After a few seconds, the display will be dark again. The Cell Matrix is now configured as 32 crossed wires: 8 from West->East, 8 from East-

>West, 8 from North->South, and 8 from South->North.

NOTE: If you want to see what the “b” command is doing, you can slow down the communication rate between the PC and the *PC Interface Board* by using the “p” command. Enter the “r” command and then try the command

p 5000

and then

b cross.bin

The bootstrap should run slower now, allowing you to better observe the bootstrap sequence. If it is still too fast to observe, try a number larger than 5000 (larger numbers cause a longer delay). To remove the delay, enter the command

p 0

To experiment with this circuit, try the command

s 2 0 dw 1

and you will see 8 LEDs illuminate, as the input propagates across the matrix. Enter the command

x

to see the D outputs of the edge cells. Note that cell [2,7] has a DE output of 1. This is the Eastern end of the wire.

Enter the command

s 0 4 dn 1

and you'll see a column of illuminated LEDs from North to South. If you again enter the “x” command, you'll see that cell [7,4]'s DS output is now **asserted** (set to 1).

Enter the command

s 7 4 ds 1

to send a 1 into cell [7,4]'s DS input. Though the LED array does not change, the “x” command will show that cell [0,4]'s DN output is now asserted.

Enter the command

s 2 0 cw 1

You will notice two things: first, the row of LEDs has gone dark. According to the architectural specification of the Cell Matrix, when a cell is in C-mode, its D outputs are set to 0 (except on any side where Cin=1). Since cell [2,0]'s DEoutput is thus 0, the row of cells are now all outputting 0 (except cell [2,4]), and thus the row is dark. And, secondly, cell [2,0]'s LED is red, indicating that the cell is in C-mode.

TUTORIAL 4 – Using Vector Commands

Enter the "r" command to reset the matrix. Now enter the command

b adder.bin

to bootstrap the matrix with the 8-bit adder circuit stored in the file "adder.bin" You can look at the corresponding Layout File adder.grd using the *Cell Matrix Layout Editor™*.

adder.bin is an 8-bit adder circuit, which reads two 8-bit numbers and produces their sum. The first number is supplied to the North, on inputs [0,0]DN=Most Significant Bit; [0,1]DN; [0,2]DN; ...; [0,7]DN=Least Significant Bit. The second number is supplied to the South, in inputs [7,0]DS=Most Significant Bit; [7,1]DS; [7,2]DS; ...; [7,7]DS=Least Significant Bit. The sum is sent to the Southern outputs, in the same order as the Southern input.

You can perform simple additions by using the "s" command. For example, to add 5 (binary 00000101) and 6 (binary 00000110), enter the following commands:

```
s 0 7 dn 1
s 0 5 dn 1
s 7 6 ds 1
s 7 5 ds 1
```

Now enter the "x" command and look at the southern outputs. They should have the values:

0 0 0 0 1 0 1 1

which corresponds to the number 11, the sum of 5 and 6.

NOTE: Interpreting the LED array's state can be confusing. Remember, an LED turns green if any of a cell's D outputs are asserted (logic level 1). In this circuit, setting an input bit on the South will cause then entire column of LEDs to be lit, because the cells are transmitting that 1 from the South to the top row. Similarly, if any output bit is 1, the corresponding column will again be illuminated, because the cells are transmitting each bit of the sum from the top tow to the South. Therefore, in this circuit, each column of LEDs shows the logical OR of the sum with the number entered in the South. Since the sum is now 00001001 and the number from the South is 00000110, all four of the rightmost columns of LEDs are illuminated.

Once you have followed through the example and are familiar with its operation, setting and reading integers in this way quickly becomes tedious. The Command Line Interface makes it possible to define a *vector*, which is a series of inputs whose values are set based on the binary value of an integer. For example, the command

```
vd a 0 7 dn -1 8
```

means “Define a vector named 'a' with the Least Significant Bit at cell [0,7]'s DN input. Consecutive bits of 'a' correspond to cells in the negative direction (-1), i.e., cells [0,6], [0,5], [0,4], ..., for a total of 8 bits.” Enter the above “vd” command. Now you can set the value of all 8 northern D inputs by specifying the value of the vector a. Next, define a vector b for the southern inputs:

```
vd b 7 7 ds -1 8
```

You can now specify the inputs to the adder numerically. Enter the commands

```
v b 0
v a 0
```

and the LEDs should all go dark. Now enter the commands

```
v a 17
x
```

and you'll see the value of 17 (0 0 0 1 0 0 0 1) on the Southern outputs. Now enter the commands

```
v b 25
x
```

and the southern outputs will show the number 0 0 1 0 1 0 1 0 which is

42 (17+25).

You can experiment with other values for a and b and observe their sum on the southern outputs.

Vectors can also be used to examine output values as integers. For example, we can define a vector 'x' to display the output of the adder. Enter the command

vd x 7 7 ds -1 8

and now the command

v x

will display the integer value corresponding to the eight southern outputs. For example, after entering “v a 17” and “v b 25” then command “v x” should display:

“x=42”

If you treat these 8 bit numbers as twos complement integers, you can perform subtraction with this same circuit. For example, in twos complement, the binary number 1 1 1 1 1 1 1 1 (255) is actually -1. If you use the command

v a 255

and then set b to a number less than 128, you'll find that x is b-1.

TUTORIAL 5 – Basic Truth Table Manipulation

You can directly manipulate an edge cell's truth table by controlling the C and D inputs on any accessible side of the cell. For example, begin with the commands:

r

s 0 0 cw 1

to place cell [0,0] into C-mode. Now enter the command

s 0 0 dw 1

to assert the D input on the same side on which the C input is asserted. Now enter the command

t

to tick the system clock a single tick. Now enter the commands

s 0 0 dw 0
s 0 0 cw 0

to return the cell to D-mode. The cell's LED is now asserted. What has happened? You have loaded a truth table containing a single 1 into cell [0,0]. This truth table is:

TRUTH TABLE:

SNEW	CN	CS	CW	CE	DN	DS	DW	DE
0000	0	0	0	0	0	0	0	1
0001	0	0	0	0	0	0	0	0
0010	0	0	0	0	0	0	0	0
0011	0	0	0	0	0	0	0	0
0100	0	0	0	0	0	0	0	0
0101	0	0	0	0	0	0	0	0
0110	0	0	0	0	0	0	0	0
0111	0	0	0	0	0	0	0	0
1000	0	0	0	0	0	0	0	0
1001	0	0	0	0	0	0	0	0
1010	0	0	0	0	0	0	0	0
1011	0	0	0	0	0	0	0	0
1100	0	0	0	0	0	0	0	0
1101	0	0	0	0	0	0	0	0
1110	0	0	0	0	0	0	0	0
1111	0	0	0	0	0	0	0	0

Recall that when a cell is in C-mode, D inputs are shifted into the cell's truth table each time the system clock ticks. The shift begins in the upper right corner of the above truth table, proceeds to the left for 8 ticks, then drops to the second row's rightmost bit, proceeds to the left for 8 ticks, and so on, until the 128th tick, when the lower leftmost bit is loaded. Since you set Din to 1 for a single clock tick, only one bit was loaded into the truth table, resulting in the truth table shown above.

This truth table says that if all D inputs are 0, then DEout should be 1, which is why cell [0,0]'s LED is illuminated. If you now set any input (N or W) on cell [0,0] to 1, you'll note the LED goes dark. This cell is thus acting as a two-input NOR gate: the output is 1 only if DNin is 0 and DWin is 0.

Enter the “x” command, and you'll note that all edge D outputs are 0. Now enter the command

s 0 0 cw 1

to return cell [0,0] to C-mode. If you now re-enter the “x” command, you'll see that cell [0,0]'s DW output is 1. This indicates the current value of bit 0 in the cell's truth table. This is how a cell's truth table can be examined by placing the cell into C-mode.

Enter the “r” command to reset the matrix. Now enter the commands:

s 0 0 cw 1
t 8
s 0 0 dw 1
t
s 0 0 dw 0
s 0 0 cw 0

Now you've configured cell [0,0] as a simple wire. If you set [0,0]DWin to 1, then the cell's DE output is also set to 1. If DWin is 0, then DEout is 0. The corresponding truth table is:

TRUTH TABLE:

SNEW	CN	CS	CW	CE	DN	DS	DW	DE
0000	0	0	0	0	0	0	0	0
0001	0	0	0	0	0	0	0	1
0010	0	0	0	0	0	0	0	0
0011	0	0	0	0	0	0	0	0
0100	0	0	0	0	0	0	0	0
0101	0	0	0	0	0	0	0	0
0110	0	0	0	0	0	0	0	0
0111	0	0	0	0	0	0	0	0
1000	0	0	0	0	0	0	0	0
1001	0	0	0	0	0	0	0	0
1010	0	0	0	0	0	0	0	0
1011	0	0	0	0	0	0	0	0
1100	0	0	0	0	0	0	0	0
1101	0	0	0	0	0	0	0	0
1110	0	0	0	0	0	0	0	0
1111	0	0	0	0	0	0	0	0

If you had said “t 9” instead of t8, then you would have built a cell whose DW output matches its DW input. You could then verify this by using the “x” command.

TUTORIAL 6 – Advanced Truth Table Manipulation

While truth table manipulation as just described is the only way to load a cell's truth table, it can be cumbersome to manually manipulate C and D inputs in this way. There are two commands, “c” and “l” that simplify such truth table manipulations.

First, enter the “r” command to reset the matrix. Now enter the commands:

```
c
de=w
.
```

“c” tells the program that you are compiling one or more Boolean equations. “de=w” is one such equation. The third line above is a single period, which indicates the end of inputting Boolean equations. See “Configuration Commands” in the **COMMAND LINE INTERFACE** chapter above for more details.

This set of commands compiles a truth table corresponding to the Boolean equation “de=w” meaning “DE out=DWin” (the right hand side terms are always D **inputs** and thus are represented simply as “n” “s” “w” or “e”). The compiled truth table is stored inside the Command Line Interface program for future use. You can now use the command “d” to see the truth table you've just created.

If you wish to load this truth table into an edge cell, you can do so as follows. First, put a cell into C-mode with the usual command:

```
s 0 0 cw 1
```

Now use the “l” (lower-case L) command as follows:

```
l 0 0 dw
```

This causes the compiled truth table bits to be sent to [0,0]'s DW input, while automatically issuing a “t” (clock tick) command following the setting of each bit. This sequence of commands will cause the compiled truth table to be loaded into cell [0,0]. Finally, issue the command:

```
s 0 0 cw 0
```

to return the cell to D=mode. Now cell [0,0] is configured as a wire from West to East. By setting or clearing [0,0]DWin you can affect its DEoutput (and hence the color of its LED). This sequence of commands is easier than those in the previous tutorial, especially for more complicated truth tables.

You can repeat the above steps for whatever truth table you like. For example, if you repeat this example but compile a pair of equations as follows:

```
c
ce=w
cs=w~
.
```

you'll find you can illuminate one of two red LEDs depending on the value of [0,0]DWin: if DWin=0, then cell [1,0] is placed in C-mode; if DWin=1, then [0,1] is placed in C-mode.

TUTORIAL 7 – Manipulating Internal Cells

You can use the “c” and “l” commands to perform more complicated truth table manipulations, including causing cells to manipulate other cells. Try the following commands:

```
r
c
ce=1
de=w
.
s 2 0 cw 1
l 2 0 dw
s 2 0 cw 0
```

This configures cell [2,0] to do two things:
1. it places cell [2,1] into C-mode via its CW input; and
2. it copies its own DW input to its DE output, and thus to [2,1]'s DW input.

Therefore, you can control the D input on cell [2,1]'s active side by manipulating cell [2,0]'s DW input. Now issue the commands:

```
c
ce=w
.
l 2 0 dw
```

This copies the above truth table ("ce=w") into cell [2,0]'s DW input, and thus into cell [2,1]'s DW input, and thus into cell [2,1]'s truth table. You've now configured cell [2,1] without having any direct access to its inputs!

To test this cell's behavior, try the following commands:

```
c
de=w
.
s 2 0 cw 1
l 2 0 dw
s 2 0 cw 0
```

This reconfigures cell [2,0] with the single equation "de=w" You now have two cells configured as follows:
Cell [2,0] copies DWin to DEout; and
Cell [2,1] copies DWin to CEout.

If you now set cell [2,0]DWin to 1, you'll see that cell [2,2] is placed in C-mode (LED turns red). If [2,0]DWin=0, then [2,2] returns to D-mode. You can now control the mode of a cell that is two cells away from the edge of the matrix!

These types of sequences, where one cell controls a second cell, and that second cell might control a third, and so on, are a key to performing complex manipulations on the Cell Matrix.

TUTORIAL 8 – An Example of Cell Replication

One of the more unique features of the Cell Matrix is that cells themselves can read and write other cell's configurations. In this tutorial, you will configure a cell to replicate a second (source) cell into an empty (target) location.

Begin by resetting the board ("r"). Now bootstrap the board to a crossed-wire configuration by using the boot command:

```
b cross.bin
```

If you now say:

```
s 2 0 dw 1
```

the entire row of LEDs turns green. If you say

```
s 2 0 dw 0
```

the row turns dark. This illustrates that the row of cells is acting as a wire.

Now compile cell [2,0] to act as an inverter:

```
c
de=w~
.
s 2 0 cw 1
l 2 0 dw
s 2 0 cw 0
```

You'll note that even though [2,0]DWin=0, the entire row of LEDs is now illuminated. This is because cell [2,0] sends the inverse of its Western input to its Eastern output. Since [2,0]DWin is 0, its output is 1, and the cross.bin configuration propagates that output across the entire row. If you set [2,0] dw to 1, the row will go dark.

Note that cell [4,0] behaves oppositely: if you set [4,0]dw to 1, the row illuminates, and if you set it to 0, it goes dark.

Now compile cell [3,0] as a *Cell Replicator*:

```
c
```

```
cn=w
cs=w
dn=n
ds=n
.
s 3 0 cw 1
l 3 0 dw
s 3 0 cw 0
s 3 0 dw 0
```

So far you haven't changed cell [4,0]: you can confirm this by setting its DWin and observing the behavior.

The cell you configured in [3,0] is set up to act as follows. If you set [3,0]DW to 1, the cell will:

- place cells [2,0] and [4,0] into C-mode;
- copy Truth Table bits from [2,0] back into [2,0]; and
- also copy Truth Table bits from [2,0] into [4,0].

This will replicate cell [2,0] in [4,0]. To perform this function, execute the following commands:

```
s 3 0 dw 1
t 128
s 3 0 dw 0
```

The first command tells [3,0] to begin acting as a cell replicator. “t 128” ticks the system clock 128 bits (once for each bit of the truth table that is to be copied). The third command tells [3,0] to stop replicating.

The 5th row of LEDs should now be illuminated. If you say

```
s 4 0 dw 1
```

the row will go dark, and if you say

```
s 4 0 dw 0
```

it will illuminate again. This is because cell [4,0] is now an exact copy of cell [2,0].

TUTORIAL 9 – Cell swapping

You can repeat the above Tutorial 8 from the beginning (starting with the

reset comand), but when you compile cell [3,0], use the commands:

```
c
cn=w
cs=w
dn=s
ds=n
.
s 3 0 cw 1
l 3 0 dw
s 3 0 cw 0
s 3 0 dw 0
```

This is almost the same Cell Replicator as above, but it copies outgoing Truth Table bits from the North to the South, and from the South to the North. Now when you run this:

```
s 3 0 dw 1
t 128
s 3 0 dw 0
```

you will find that cells [2,0] and [4,0] have swapped their truth tables with each other.

You can repeat these last three commands to execute the swap again, and return cells [2,0] and [4,0] to their original configurations.

TUTORIAL 10 – A Self-Clocking Counter

Look at the circuit "count6.grd" in the *Cell Matrix Layout Editor™*. It is a series of 21 Toggle Flip Flops, wired together to act as a long ripple counter. The top row of the matrix is an inverter whose output is feeding its input. This forms an oscillator, whose output value toggles rapidly between 0 and 1. The signal is forced to pass through a number of cells to slow down the oscillation rate.

Oscillation will only begin if cell [0,1]'s DN input is asserted. This is ANDed with the inverter's output before passing back to the input. Finally, the output of the inverter/oscillator is fed into the first flip flop's Toggle input, thus driving the entire chain's counting operation.

To see this circuit work, enter the commands:

```
r
b count6.bin
```

to load the circuit, and then activate the oscillator with the command:

```
s 0 1 dn 1
```

The LEDs should now be changing rapidly, but the lower left ones change slowly enough to see their two states (1 and 0). You may detect slowly-changing patterns in the other LEDs. These are interference patterns between the actual changing of the cell's states and the refresh of the LED display – similar to looking at the spokes of a spinning bicycle wheel with a TV behind it.

You can stop the counter with the command

```
s 0 1 dn 0
```

and then restart it by again setting the input to 1.

If you follow the instructions above under **Standalone Operation Following Bootstrap**, you can undock the *PC Interface Board* from the PC while the counter is counting, and the counter will continue to run.

7.STAND-ALONE OPERATION

The *Cell Matrix Mod 88* can be used in a standalone manner, without the *PC Interface Board*, and without any external PC. This chapter describes the requirements and pin-out specification for using the *Cell Matrix Mod 88* in this way.

WARNING: It is extremely easy to damage the *Cell Matrix Mod 88* by incorrectly connecting it to other devices or incorrectly powering it. Please be extremely careful when making such connections, as the board itself has little protection against such mistakes. See the "Board Overview" and "Precautions" sections of Chapter 1, **INTRODUCTION**, for more details.

General Comments

The board is TTL compatible, though it really uses a Low Voltage TTL standard (0-3.3V). It will tolerate 5V TTL inputs, and its logic level 1 output will be interpreted by standard TTL as High. Again, there is no protection against mistakenly driving an output pin from outside, or from drawing too much current from an output or supplying to much current to an input. See Xilinx's documentation on the XC2S200 FPGA for precise details on the LVTTL I/O standard.

You can pull +5V from the WEST side of the board, on pin 35. There is enough current to drive at least a few HCT-family chips. This is where the *PC Interface Board* gets its regulated power.

The pin location on each side is as shown below. These are FRONT VIEWS, looking directly at each edge. Note also that Pins 2 and 46 are labeled on the *Cell Matrix Mod 88* itself. You can use those markings to doublecheck the pinout on each side.

WEST:	2 4 6 8 10 ... 42 44 46	
(LEFTSIDE) (PINS)	1 3 5 7 9 ... 41 43 **45**	(RIGHTSIDE)
EAST:	46 44 42 40 ... 8 6 4 2	
(LEFTSIDE) (HOLES)	**45** 43 41 39 7 5 3 1	(RIGHTSIDE)
NORTH	2 4 6 8 10 ... 42 44 **46**	
(LEFTSIDE) (PINS)	1 3 5 7 9 ... 41 43 45	(RIGHTSIDE)
SOUTH	**46** 44 42 40 ... 8 6 4 2	
(LEFTSIDE) (HOLES)	45 43 41 39 ... 7 5 3 1	(RIGHTSIDE)

Boldfaced pins/holes are missing/blocked for polarization.

Power Requirements

The *Cell Matrix Mod 88* is powered via two sets of pins: Pins 1 and 2 (on each side of the board) are for GROUND, and pins 43-46 are for POWER. POWER can be any voltage from 6V-15V, and does not need to be regulated (there are voltage regulators on the board). Power can be supplied to any edge of the board, or from different edges. All pins 1 & 2 on all sides are electrically connected to each other, and all pins 43-46 on all sides are electrically connected to each other. **There is no protection against reversing the power.** If power is accidently reversed, the board will likely be permanently damaged.

Clocking

There are two system-wide clocks, Phi1 and Phi2. These are delivered to the board on pins 37 (Phi 1) and 38 (Phi 2), on all sides of the board. Again, all pins 37 are electrically connected to each other, and all pins 38

are electrically connected to each other. The normal clocking scheme is: Raise Phi 1, Drop Phi 1, Raise Phi 2, Drop Phi2.

Reset

There is a system-wide RESET line, available on Pin 38 (all sides). Once asserted, 256 Phi1/Phi2 clock cycles should be run, using the normal clocking scheme described above. Following this the RESET line should be returned to ground. This reset operation should leave all cell's truth tables cleared (all zeros), and all cell outputs should now be 0.

Neighbor Signal

Normally, a cell's inputs are supplied by internal flip flops, whose values can be controlled by an external PC interface. If you wish to communicate directly with a side of the *Cell Matrix Mod 88*, you must disable these flip flops by asserting a NEIGHBOR-BAR pin. See the Pinout Table below for the location of the NEIGHBOR-BAR pin on each side. This is an inverted signal, so you must pull it to Ground to assert it (to indicate that you wish to control a side's cells directly).

Cell Input/Output

Each side allows direct access to the edge C and D inputs and outputs of each cell along that side. For example, along the Northern side, you can access DNin, DNout, CNin and CNout of cells [0,0]-[0,7] See the Pinout Table below for the exact location of each I/O line on each side.

PINOUT TABLE

PIN	***NORTH***	***SOUTH***	***WEST***	***EAST***
1,2	Ground	Ground	Ground	Ground
3	[0,7] Din	[7,7] Dout	[0,0] Dout	[0,7] Din
4	[0,7] Dout	[7,7] Din	[0,0] Din	[0,7] Dout
5	[0,7] Cin	[7,7] Cout	[0,0] Cout	[0,7] Cin
6	[0,7] Cout	[7,7] Cin	[0,0] Cin	[0,7] Cout
7	[0,6] Din	[7,6] Dout	[1,0] Dout	[1,7] Din
8	[0,6] Dout	[7,6] Din	[1,0] Din	[1,7] Dout
9	[0,6] Cin	[7,6] Cout	[1,0] Cout	[1,7] Cin
10	[0,6] Cout	[7,6] Cin	[1,0] Cin	[1,7] Cout
11	[0,5] Din	[7,5] Dout	[2,0] Dout	[2,7] Din
12	[0,5] Dout	[7,5] Din	[2,0] Din	[2,7] Dout
13	[0,5] Cin	[7,5] Cout	[2,0] Cout	[2,7] Cin
14	[0,5] Cout	[7,5] Cin	[2,0] Cin	[2,7] Cout
15	[0,4] Din	[7,4] Dout	[3,0] Dout	[3,7] Din
16	[0,4] Dout	[7,4] Din	[3,0] Din	[3,7] Dout
17	[0,4] Cin	[7,4] Cout	[3,0] Cout	[3,7] Cin
18	[0,4] Cout	[7,4] Cin	[3,0] Cin	[3,7] Cout
19	[0,3] Din	[7,3] Dout	[4,0] Dout	[4,7] Din
20	[0,3] Dout	[7,3] Din	[4,0] Din	[4,7] Dout
21	[0,3] Cin	[7,3] Cout	[4,0] Cout	[4,7] Cin
22	[0,3] Cout	[7,3] Cin	[4,0] Cin	[4,7] Cout
23	[0,2] Din	[7,2] Dout	[5,0] Dout	[5,7] Din
24	[0,2] Dout	[7,2] Din	[5,0] Din	[5,7] Dout
25	[0,2] Cin	[7,2] Cout	[5,0] Cout	[5,7] Cin
26	[0,2] Cout	[7,2] Cin	[5,0] Cin	[5,7] Cout

PIN	*NORTH*	*SOUTH*	*WEST*	*EAST*
27	[0,1] Din	[7,1] Dout	[6,0] Dout	[6,7] Din
28	[0,1] Dout	[7,1] Din	[6,0] Din	[6,7] Dout
29	[0,1] Cin	[7,1] Cout	[6,0] Cout	[6,7] Cin
30	[0,1] Cout	[7,1] Cin	[6,0] Cin	[6,7] Cout
31	[0,0] Din	[7,0] Dout	[7,0] Dout	[7,7] Din
32	[0,0] Dout	[7,0] Din	[7,0] Din	[7,7] Dout
33	[0,0] Cin	[7,0] Cout	[7,0] Cout	[7,7] Cin
34	[0,0] Cout	[7,0] Cin	[7,0] Cin	[7,7] Cout
35	RESERVED	RESERVED	+5 Volts From Board	RESERVED
36	RESET	RESET	RESET	RESET
37	PHI 1	PHI 1	PHI 1	PHI 1
38	PHI 2	PHI 2	PHI 2	PHI 2
39	NBR BAR	RESERVED - GND	RESERVED - GND	NBR BAR
40	RESERVED - GND	NBR BAR	NBR BAR	RESERVED - GND
41	RESERVED	RESERVED	RESERVED	RESERVED
42	RESERVED	RESERVED	RESERVED – PC BAR	RESERVED
43	Power	Power	Power	Power
44	Power	Power	Power	Power
45	Power	Power	MISSING (Polarizing)	MISSING (Polarizing)
46	MISSING (Polarizing)	MISSING (Polarizing)	Power	Power

8.USING MULTIPLE BOARDS

You can use two or more *Cell Matrix Mod 88s* together, by connecting them to each other with their edge connectors. We have worked with as many as 8 boards at once without difficulty. Boards can be assembled in any configuration desired, as long as connections are made only between North/South and East/West (the polarizing of pins 45 and 46 should prevent, for example, connecting West to South or East to North).

The main difficulty in driving multiple boards is generating clean clock and reset signals. If you wish to experiment with this beyond a few boards, please contact Cell Matrix Corporation for further technical details.

If you interface a collection of boards to a PC, you will want a single *PC Interface Board* connected to the Western edge of one *Cell Matrix Mod 88*. It does not matter to which board you make this connection, but you will only have direct I/O access to the board to which the interface is connected. Obviously, your position in the larger (multi-board) matrix will be determined by where you connect the *PC Interface Board*.

It can be difficult mechanically to connect or disconnect a number of boards. The tolerances in the connectors are small, and the friction can be quite high, so it is easy to bend or break connector pins. Please use caution when assembling or disassembling boards. See the “Separating Boards” section in Chapter 3, **INTERFACE BOARD OVERVIEW** for details on how to safely separate boards.

9.FURTHER INFORMATION AND TECHNICAL SUPPORT

For more information, technical help or general feedback, please contact Cell Matrix Corporation by sending email to:

mod88@cellmatrix.com

Please contact us if you have any difficulties using your boards, or if you have questions or comments. We are eager to help you solve any problems in getting the boards working. We are also very interested in knowing what you are doing with these boards, what ideas you have, any suggestions, and so on.

Please visit http://www.cellmatrix.com for further information about the Cell Matrix Architecture and Cell Matrix Corporation.

10.APPENDICES

APPENDIX 1 - CD CONTENTS

The included CD contains documentation, various software components for different environments, and examples/sample code. You should copy whatever pieces you need from the CD to a work directory on your harddrive. See the chapter "**INSTALLATION**" for an overview of which software pieces you might need.

The CD contains the following directories and files:

DOC\
    Contains this document (MOD88.PDF)
EXAMPLES\
    Has .GRD (inputs to the *Cell Matrix Layout Editor™*) and .BIN (outputs from the *Cell Matrix Layout Editor™*) files for sample programs. Also contains .C and .TXT sample code for using the API.
EXTRAS\
    PORT95NT\
        has driver for parallel port access on protected operating systems
    XP\
        has .reg code for disabling/enabling parallel port polling by operating system
LAYOUT\
    has documentation, executables, and sample grids and libraries for the *Cell Matrix Layout Editor™*.
    WINDOWS\
        Contains the Windows-specific version of the layout editor. Also contains an installation package for the Java Run-Time Environment. See Appendix 2 for detail on installing the layout editor.
    LINUX\
        Contains the Linux version of the layout editor.

See Appendix 2 for detail on installing the layout editor.

SOFTWARE\

CYGWIN\

Contains main program for executing Cygwin version of Command Line Interface. Includes cygwin1.dll (required). Also has api.h and api.o for linking to API via Cygwin gcc compiler. Requires PORT95NT to be installed.

DLL32\

Contains api.dll and api.h for executing API commands via Windows DLL. Requires PORT95NT to be installed.

DOS\

Contains main.exe (16-bit executable), api.h and api.obj (16-bit API, linkable under Borland Turbo C 2.01). Will not work on protected operating systems such as Windows XP. Does not appear to work under Windows 3.1 either.

LINUX\

Contains main (Linux executable Command Line Interface), api.h and api.o for linking under Linux's gcc compiler

VB32\

Includes VB Header.txt which defines the DECLARE statements for DLL entry points. Also defines VB-specific versions of raw_comp() and boot_mem() routines.

WIN32\

Includes main.exe (32-bit Windows Command Line Interface), api.h and api.obj (32-bit API, linkable under Borland Turbo C++5.5.1) Requires PORT95NT to be installed.

APPENDIX 2 - Cell Matrix Layout Editor™

Windows Installation Instructions

If you wish to design Cell Matrix configurations graphically, you will need to install the *Cell Matrix Layout Editor™* software. This is included on the CD under the LAYOUT\WINDOWS directory. The file "Cell Matrix Layout Editor.pdf" describes the installation and use of this tool. **Note** that the instructions tell you to execute the "Portable Loader.exe" self-extracting file, but this step is not necessary. You can simply copy the files (all of them!) from the CD's LAYOUT\WINDOWS directory (and all subdirectories) to your desired installation directory.

Next, **YOU SHOULD RUN SETUP.BAT** after copying the files. Some operating systems make files Read Only if they are copied from a CD. SETUP.BAT will change files beneath the current directory to be Read/Write. You can execute SETUP.BAT by double-clicking on it in your work directory, or by using the command "setup.bat" from a command prompt (such as DOS or Cygwin).

If you do not currently have a Java runtime environment on your system (at least V2), you will need to install that as well. An installation file is included under the LAYOUT directory, and instructions are provided in the document Cell Matrix Layout Editor.pdf

Once the files have been copied, setup.bat has been run, and you have a Java runtime environment set up, you should be able to run the *Cell Matrix Layout Editor™* by simply double-clicking "loader.bat" However, depending on your system's configuration, it is possible this may not work. If you encounter difficulties, try installing loader.bat as described in the Cell Matrix Layout Editor.pdf document.

Linux Installation Instructions

If you wish to design Cell Matrix configurations graphically, you will need to install the *Cell Matrix Layout Editor™* software. This is included on the CD under the LAYOUT\LINUX directory. The file "Cell Matrix

Layout Editor.pdf" describes the installation and use of this tool. **Note** that the instructions tell you to execute the "Portable Loader.exe" self-extracting file, but this step is not necessary. You can simply copy the files (all of them!) from the CD's LAYOUT\LINUX directory (and all subdirectories) to your desired installation directory.

Next, you should use the command "source setup" or "./setup" to execute the commands in the setup script. This will ensure that installed files are writable.

You will need the Java runtime environment on your system (at least V2).

Finally, **you need to edit the loader script**. Currently, the script contains the command:

```
java -DLIBROOT=/
local/home/nick/linuxsrc/LIBRARIES/ -classpath
"./loader.jar:./Acme.jar" loader
```

(all on one line). You need to change the -DLIBROOT argument to whatever directory you have copied the layout editor files to (in this case, "/local/home/nick/linuxsrc"), **followed by /LIBRARIES/**

It is critical that you have the trailing "/" after "LIBRARIES"

Note also that all parts of this command are case sensitive. The path (/local/home/nick/linuxsrc) depends on where you have copied the files to; all other parts of the command should be exactly as shown above (but again, on a single line).

Once the files have been copied, setup has been run, you have a Java runtime environment set up, and you've edited the "loader" script, you should be able to run the *Cell Matrix Layout Editor*™ by executing the command "./loader"

APPENDIX 3 - *WINDOWS XP NOTE*

Windows XP expects to be the only software accessing the parallel port, and thus makes free use of that port. Depending on your PC's configuration, Windows may send commands to the parallel port at unexpected times, causing a (non-permanent) malfunction of the *Cell Matrix Mod 88*. There are two ways to remedy this:

(1)in some cases, once you begin the main program and execute an "x" command, Windows will attempt to use the port for approximately one minute, but appears to then give up and relinquish control. Therefore, waiting one minute may cause the problem to go away;

(2)you can modify the Windows registry to cause it to cease this behavior. **WARNING: Modifying the registry can cause permanent damage to your operating system, possibly resulting in loss of usability and loss of data. Make sure you backup and checkpoint your system before proceeding**. Note that you may need to undo this change in order to get certain other parallel port devices to work. The easiest way to make this change is to go to the EXTRAS\XP directory on the CD, and double-click "disablepolling.reg" This will make the changes described below. You must reboot for the changes to take effect. If you want to restore parallel port polling, double click "enablepolling.reg" and reboot. Further details are described below:

```
The following registry setting disables
the port writes:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlS
et\Services\Parport\Parameters]
"DisableWarmPoll"=dword:00000001

The following registry setting enables the
port writes:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlS
et\Services\Parport\Parameters]
"DisableWarmPoll"=dword:00000000

You can make these changes in Windows'
```

regedit utility. Or to make the changes automatically, create and save the following text files:

DisablePolling.reg contains the following text:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Parport\Parameters]
"DisableWarmPoll"=dword:00000001
```

EnablePolling.reg contains the following text:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Parport\Parameters]
"DisableWarmPoll"=dword:00000000
```

To change a registry key, run the program.

APPENDIX 4 – PURCHASE TERMS AND CONDITIONS

PURCHASE TERMS AND CONDITIONS

THE LAWS OF THE STATE OF VIRGINIA SHALL GOVERN THESE PURCHASE TERMS AND CONDITIONS.

LIMITED WARRANTY

LIMITED WARRANTY. CELL MATRIX CORPORATION (THE "COMPANY") WARRANTS THAT (A) ITS SOFTWARE AND HARDWARE (THE "PRODUCT") WILL PERFORM SUBSTANTIALLY IN ACCORDANCE WITH THE ACCOMPANYING WRITTEN MATERIALS FOR A PERIOD OF NINETY (90) DAYS FROM THE DATE OF RECEIPT AND (B) THAT THE MEDIUM ON WHICH THE SOFTWARE IS CONTAINED WILL BE FREE FROM DEFECTS IN MATERIALS AND WORKMANSHIP UNDER NORMAL USE AND SERVICE FOR A PERIOD OF ONE (1) YEAR. IN THE EVENT APPLICABLE LAW IMPOSES ANY IMPLIED WARRANTIES, THE IMPLIED WARRANTY PERIOD IS LIMITED TO NINETY (90) DAYS FROM THE DATE OF RECEIPT. SOME JURISDICTIONS DO NOT ALLOW SUCH LIMITATIONS ON DURATION OF AN IMPLIED WARRANTY, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

CUSTOMER REMEDIES. THE COMPANY'S AND ITS SUPPLIERS' ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT THE COMPANY'S OPTION, EITHER (A) RETURN OF THE PRICE PAID FOR THE PRODUCT, OR (B) REPAIR OR REPLACEMENT OF THE PRODUCT THAT DOES NOT MEET THIS LIMITED WARRANTY AND WHICH IS RETURNED TO THE COMPANY WITH A PROOF OF PURCHASE. ANY REPLACEMENT PRODUCT WILL BE WARRANTED FOR THE REMAINDER OF THE ORIGINAL WARRANTY PERIOD OR THIRTY (30) DAYS, WHICHEVER IS LONGER. YOU MUST CONTACT THE COMPANY OR ITS SUPPLIER WHERE YOU PURCHASED THE PRODUCT BEFORE RETURNING THE PRODUCT FOR A REFUND OR REPLACEMENT. THIS LIMITED WARRANTY IS VOID IF FAILURE OF THE PRODUCT HAS RESULTED FROM ACCIDENT, ABUSE, OR MISAPPLICATION. UNDER NO CIRCUMSTANCES WILL THE COMPANY OR THE SUPPLIER BE LIABLE FOR ANY DAMAGES RESULTING FROM ANY OF (BUT NOT LIMITED TO) THE FOLLOWING: MISAPPLICATION OF POWER/VOLTAGE LEVELS; DAMAGE DUE TO INCORRECT CONNECTION OF ANY PART OF THE PRODUCT TO ANY OTHER PART OF THE PRODUCT OR TO ANY EXTERNAL EQUIPMENT; OR BROKEN CONNECTOR PINS OR OTHER PHYSICAL DAMAGE DUE TO MISHANDLING OF THE PRODUCT.

NO OTHER WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE COMPANY AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH REGARD TO THE PRODUCT AND ANY RELATED OR ACCOMPANYING WRITTEN MATERIALS. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHER RIGHTS WHICH VARY FROM JURISDICTION TO JURISDICTION.

NO LIABILITY FOR DAMAGES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL THE COMPANY OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING WITHOUT LIMITATION, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES FOR PERSONAL INJURY, LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PRODUCT, EVEN IF THE COMPANY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS INCLUDES PHYSICAL DAMAGE TO ANY DEVICE TO WHICH YOU CONNECT THE PRODUCT, AS WELL AS DAMAGE TO ANY SOFTWARE ON ANY SYSTEM TO WHICH YOU CONNECT THE PRODUCT, WHETHER SUCH DAMAGE IS CAUSED BY INCORRECT HANDLING OR USE OF THE PRODUCT, A DEFECT IN THE PRODUCT, OR ANY OTHER REASON WHATSOEVER. IN ANY CASE, THE COMPANY'S AND ITS SUPPLIERS' ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE PRODUCT. BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

Licensing Agreement

The Cell Matrix Mod 88 and included software (collectively, "The Product") utilize intellectual property (IP) belonging to Cell Matrix Corporation and its directors, and is protected under US Patent Law, including US Patents #6,222,381 and #6,577,159. In using The Product, you are granted a non-exclusive, non-transferable license to use the embodied IP, only in the form provided with The Product. You may not redistribute, prepare derivative works, or use the embodied IP except in the form provided with and in conjunction with the use of The Product. Nothing contained herein shall be construed as granting license to any other IP, or as granting any other rights to the IP contained in The Product.

Note Regarding Exports:

You agree not to allow the Cell Matrix Mod 88 hardware to be sent to or used in any country other than the United States except in compliance with this license and applicable U.S. laws and regulations.

If you do not agree with these terms, you must not install or use The Product, or must uninstall and cease using The Product immediately.

Trademark Notice

Cell Matrix is a trademark of Cell Matrix Corporation. All other trademarks used within this document are the property of their respective owner.

NOTES

NOTES

NOTES