side4linux

MACHINE CONTROLLER OPERATING MANUAL

INTRODUCTION

Brief description of this manual.

This manual describes a new beginning for us. As a metals manufacturing company we have need for a more integrated manufacturing environment embodying the benefits of a modern computer operating system along with the adoption of an Open Source methodology.

By using Open Source software we are able to develop new programs that we are in need of using modern tools such as the GCC 'C' compiler in the friendly graphical environment of GNOME. This will not be a one way street as we are donating back to the Open Source Community several programs of our own that have already seen many years of development, check out the 'side4linux' Project on www.sourceforge.net as a starter. We are also adopting other Open Source programs such as the AVRA assembler and will contribute to their continual improvement.

Our machines now have a new Machine Controller installed based on an Atmel AVR micro controller chip. The Machine Controller is a design of our own making produced entirely in-house. This Manual will now set about describing its function, but first a little history.

The programs that we have been developing since 1992 called "BCAM" and "RUNMILL" for Numerically Controlled (NC) metals manufacture are now being converted over to **'side4linux'**, our new Integrated Development Environment. This new IDE is being developed to provide us with a complete Computer Integrated Manufacture system (CIM) on Linux. The program "BCAM" originally took an ASCII text file written in a subset of ANCA type G-Code code and translated it into the code understood by various OEM machine controllers. Then "RUNMILL" which was a real time program was employed to actually run the machinery from a controlling computer via a serial link. The programs "BCAM" and "RUNMILL" continue on as part of the **side4linux** Project although somewhat split up and renamed. They have been converted from Turbo Pascal/Delphi on Dos/Windows to 'C' and GNOME/GTK+ on Linux. This took quite a while and was only possible due to the availability of 'Gedit' a very good text editor, the 'p2c' conversion program, and the extensive Help Documentation available for GTK/GNOME/Linux, all of which are Open Source. This manual is being written using Open Office 2, also Open Source.

You will notice as you read on that many programs start with the letter 'B'. This is an historical connection to our first NC machine a Bridgeport 2 vertical axis milling machine and has no connection either with the Bridgeport Company or the **side4linux** Copyright holder. This convention continues on simply for historical reasons so that we know the origins and purpose of that part of our extensive code base.

This manual is, and will continue to be, a work in progress. This is a deliberate and calculated action that is part of our policy of continual improvement. At some stage the circuit diagrams will become developed enough to be included and so this manual will take on a maintenance role as well as being the operator/programmer manual.

I guess the good part for you the reader is that everything to make your own machine controller will be in the **side4linux** project. There will be nothing to stop you building your own, either a direct copy, or incorporating your own ideas as well. This work is released under the GPL, you may obtain a copy of the license as follows,

David Burke, Adelaide Aeromotive Pty. Ltd, Adelaide, South Australia, 2007.

Table of Contents

1 Machine Controller Overview

The machine will be controlled by an Atmel AVR based Machine Controller from a control computer connected via a serial link using a three step process as described in Sections 1.2, 1.3 & 1.4 below. It changes the automatic mode of operation in the Company from tape NC to computer CNC as is the modern practice. The Machine Controller can also be accessed in debugging mode by the 'Bterm' program via the same serial link from the controlling computer. The 'Debug' mode provides a low level interface to the control code on-board the Machine Controller to probe or alter memory or register locations. The Debug commands are described in Section 1.5 below. To carry on with the overview the actual files produced by side4linux in each G-code Project will be discussed in Section 2.

1.1 Machine controller language summary

By using a three step process high level commands are translated firstly from high level G-code files into an 'Intermediate' set of files and then into a single machine type specific 'Instruction' file. This provides full abstraction from the high level design down to the real world machining of a metal component. Further a debuging program is provided on-board the Machine Controller for diagnostic purposes.

Commands recognised by the machine controller therefore fall into two groups,

Machine Control.	(Described by Sections 1.2, 1.3 and 1.4)
Debug Monitor.	(Described by Section 1.5)

1.2 ANCA style G-code control emulation

ANCA style G-code is a machine control code by which the sequence of machining operations on a part may be controlled from a human understandable level. The side4linux G-code compiler 'Compileanca' will emulate the G-code command set as follows (see the Help system of the IDE for a more detailed description),

G CODES		
	G0	= Rapid traverse positioning eg G0 X125.5 Y27.0 Z50.0
	G1	= Linear interpolation at the G94 feed rate.
	G2	= Circular interpolation clockwise when looking down on a plane.
	G3	= Circular interpolation anti-clockwise when looking down on a plane. e.g. (G2/G3)XnnYnnZnnInnJnn
	G17	= Set to circular interpolation in the XY plane.
	G18	= Set to circular interpolation in the XZ plane.
	G19	= Set to circular interpolation in the YZ plane.
	G40	= Cutter compensation cancel.
	G41	= Cutter compensation left.
	G42	= Cutter compensation right.
	G57	= Start of subroutine (G57Rn)
	G58	= End of subroutine (G58)
	G59	= Call subroutine Rn, Cn times (G59R5C6)
	G90	= Absolute dimensions.
	G91	= Relative dimensions.
	G94	= Feed in millimeters per minute. (G94nnn)
M CODES		
	M2	= End of program : indicates end of program and rewinds program to start.
OTHER		
	(	= Comment follows, e.g. G0X7.6 (THIS IS TO BE IGNORED BY THE COMPILER

Essentially machining breaks down into two types of basic operations they are Linear interpolation (machining in a straight line) and Circular interpolation (machining arcs which are parts of a circle).

<u>Linear interpolation.</u>

Say you want to machine from the origin ahead to a position of X = 10, Y = 5,

You can go flat out to get there with the G0 command e.g. G0X10Y5
or go at the pre-set machining rate with the G1 command e.g. G1X10Y5

<u>Circular interpolation.</u>

When programming circular interpolation it pays to remember these steps,

1. Determine arc direction (G2,G3)
2. Establish the co-ordinate points at the end of the circular motion. (X/Y values)
1. Calculate arc centre offset using the I,J,K letter codes as follows, (I/J values)

I Is the distance parallel to the x axis from the start position of the arc to the centre point of rotation.
J Is the distance parallel to the y axis from the starting position of the arc to the centre point of rotation.
K Is the distance parallel to the z axis from the starting position of the arc to the centre point of rotation.

For example a clockwise arc in the XY plane which has the following,

Center	= X0,Y0	
Start	= X5,Y3	(will calculate to I/J values)
End	= X-2,Y-5	(will calculate to X/Y values)

Would be G2X2Y5I-5J-3 (Note Y = 5 points UP to the center and I = 5 points BACKWARD to the center.)
So if FORWARD is to the right hand when facing the XY plane , UP/FORWARD is positive and BACKWARD/DOWN is negative.

(Or just use the side4linux IDE, Click on Tools>ANCA>G2 G3 Arc Calculator).

Distance can be relative or absolute depending on the G90/G91 code setting so beware of possible problems which can arise. Note that G2 or G3 only cut at the machining rate as if it was a G1 command. There is no equivalent G0 command!

1.3 Machine control intermediate commands

Machine control routines in G-code are read into the 'Compileanca' program where they are compiled into 'Intermediate' command files. The Intermediate command files are then processed by the 'B2raw' program to create the Machine Controller 'Instruction' file. This single file can then be sent on to either of the 'Bmach' programs (Bmach2d or Bmach3d) for a visual check run or be sent by the 'Milldrive' program to a Machine Controller via a serial link between the Mill Controller and the controlling computer. The 'Intermediate' commands recognised are,

VX	Change X axis Vector register (relative co-ordinates)
VY	Change Y axis Vector register (relative co- ordinates)
VZ	Change Z axis Vector register (relative co-ordinates)
AX	Change X axis Vector register (absolute co-ordinates)
AY	Change Y axis Vector register (absolute co-ordinates)
AZ	Change Z axis Vector register (absolute co-ordinates)
VL	Load step Values into Move Axis Registers
VM	Interpolated Move to Current Position of Vector Registers X,Y,Z
M00	Program END
M01	Set Feed Rate in 1/10 inches per minute. e.g. M01 22
M02	Rewind PROGRAM, Spindle OFF, Coolant OFF
M03	Display Message on front panel display
M04	Spindle ON
M05	Spindle OFF
M06	Program STOP, Spindle STOP, Coolant OFF (press Start to continue)
M07	Coolant ON
M08	Reserved.
M09	Coolant OFF
M20	Annul Axis REVERSALS
M21	Reverse X Axis
M22	Reverse Y Axis
M23	Reverse Z Axis
M50	Drill Cycle OFF, Quill UP
M51	Cycle Drill
M52	AUX. Function 'A' ON
M53	AUX. Function 'A' OFF
M54	Advance turret stop one position
M56	AUX. Function 'B' PULSE
M57	CYCLE/MILL,QUILL DOWN
M58	Quill Up
M59	Quill Down
M90	Set Scale for one Axis
M91	Assign Variable a Value
M92	Accept calculation block VAR3 = VAR1 +%*/ VAR2
M93	Rotate Axis
M94	Set new origin
M95	Reset to original origin
M98	Reply to sender with the Interpolator Flag Register
M99	Reply to sender with the Axis Absolute count for Z/Y/X/R/W

1.4 Machine controller instruction language

Machine control routines are accessed by the 'drive' series of programs (milldrive, lathedrive,robodrive etc.) via the serial link between the Mill Controller and the controlling computer or by one of the 'Bmach' programs for visual checking purposes. The Machine Controller 'Instruction' commands recognised are

N	Next steps forshadowed	e.g. 'N Xd Yd Zd Rd Wd 345 267 89 0 0' (direction d = P or N)
T	Tool change command.	e.g. 'T#1 3.0 0'
F	Feedrate change command,	e.g. 'F 50'
X	X axis positive step.	e.g. 'X'
x	X axis negative step	e.g. 'x'
Y	Y axis positive step	e.g. 'Y'
y	Y axis negative step	e.g. 'y'
Z	Z axis positive step	e.g. 'Z'
z	Z axis negative step	e.g. 'z'
R	R axis positive step	e.g. 'R'
r	R axis negative step	e.g. 'r'
W	W axis positive step	e.g. 'W'
w	W axis negative step	e.g. 'w'
Q	Quit command	e.g. 'Q'

1.5 Machine controller monitor routines

Machine Controller 'Monitor' routines used for debugging purposes are stored in the monitor jump table on board the Machine Controller and are accessed from the controlling computer by setting the following front panel switches as indicated,

MODE SELECT SWITCH TO --------> MANUAL
DATA SELECT SWITCH TO ----------> C
PRESSING ----------------------------------> START

To communicate with the MONITOR you must have connected the serial cable between the computer and the Machine Controller and called up the 'Bterm' program from the side4linux IDE.

Upon pressing the start button the following (or similar) will appear on the controlling computer screen,

```
'MCV6.1 © D.BURKE 2006'
```

The following commands are then available,

QDnnnn	Display next sixteen bytes of program memory.
QFxxxxyyyyzz	Fill data RAM memory from xxxx to yyyy with data zz.

<u>NOTE</u>

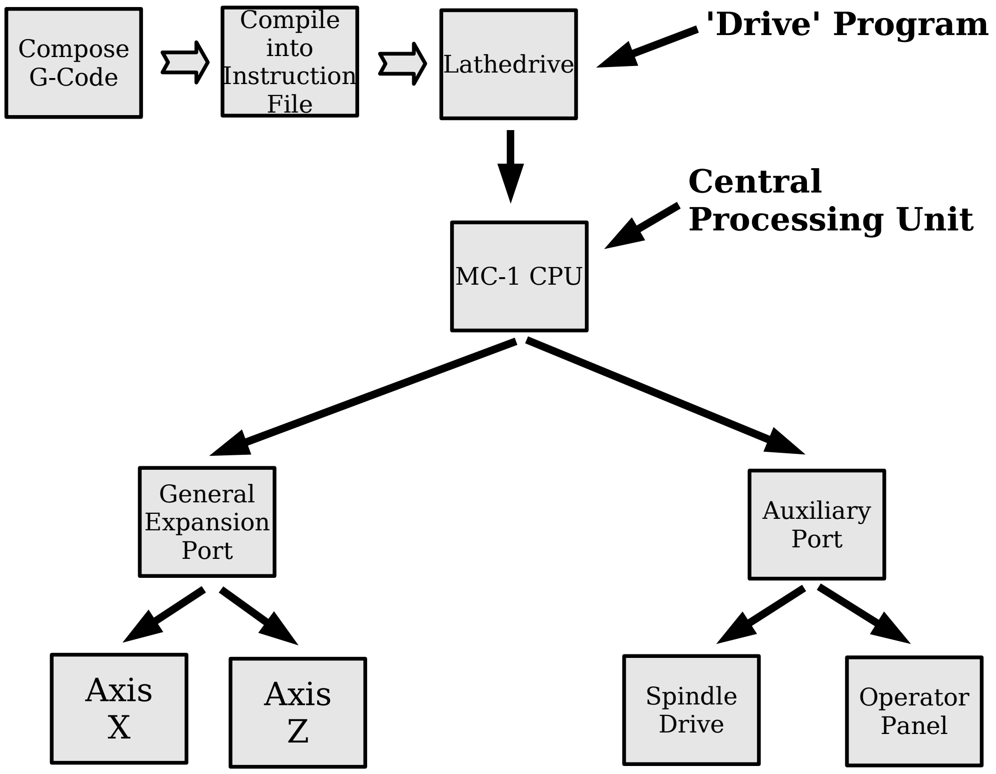
1/ All characters and hex letters are to be in upper case only.
2/ All numbers are in hex ascii.

2 Machine Controller Files and Programs

Introduction

Compiling from the high level G-code down to machine step commands is a three step process as described in Section 1. Here we look further at the files produced during this process and the programs that create them but firstly a look at the overall process using a metal working lathe as an example.

General flow of G-Code through the Machine Controller to the Axis Drives of a two Axis lathe.



The side4linux IDE takes text files (in UTF8 character format) written in a subset of G-code and compiles it to the 'Intermediate' M-code outlined on the previous pages. The 'Intermediate' files are further processed by the B2raw program to produce the raw step command 'Instruction' file understood by the new Machine Controller and also by visual emulator programs such as Bmach2d and Bmach3d. It should be pointed out here that the G-code is normally written using metric dimensions but the raw code is converted to imperial since most of the machine ball screws we have are imperial. This conversion is transparent to the user as it is set by a Project variable that once set need not be changed for the target machine.

2.1 Step 1, Generate the G-code and Overlay code

Using the side4linux text editor or one of the Tools provided (such as the PCBS Gerber 'Isolation' Tool) write the text for the G-Code program and include any subroutines. Using the text editor or one of the 'Design' programs provided (such as 'Bdesign3d') write up the Overlay file. Once the G-Code file is ready and current in the text editor click on the 'Build' button to compile the code. You may then simulate the raw code using a simulator provided (such as Bmach2d). The structure for a typical G-Code file is shown below,

```
( G-code file for 'isolation' cutting of a PCB's tracks.
G90 ( absolute positioning from bottom left hand corner
G94 ( use units per minute movements
G70 ( we are using inch measurements!
F2.0 ( initial feedrate at 2 inches per minute, alter to suit!
( ========================= SUBROUTINES =========================
G57R1 ( drill hole subroutine
G0Z-0.01 ( race down to surface
G1Z0.08 ( drill hole at feed rate
G0Z-0.02 ( clear surface
G58 ( end hole drilling subroutine
(
G57R2 ( lower isolation cutter subroutine
G0Z-0.001 ( race down to surface
G1Z0.008 ( cut to bottom of isolation trench at feed rate
G58 ( end lower isolation cutter subroutine
( ============================ MAIN ============================
( Start of program ...
( Start of Drill all holes ....
T#2 0.02 0 ( select 20 mil diameter twist drill for holes
(
G0X0.290000Y0.230000 ( go to start of hole
G59R1C1 ( drill a hole here
(
T#1 0.008 0 ( replace the hole drill with the isolation Tool
( begin track isolation cutting ....
(
G59R3C1 ( raise isolation cutter
G0X0.274000Y0.097000 ( race to start of track
G59R2C1 ( lower isolation cutter
G1X0.275000Y0.096000 ( isolation cut track
G1X0.276000Y0.096000 ( isolation cut track
G1X0.308000Y0.096000 ( isolation cut track
( End of isolation cutting so back to start condition!
(
G0Z-0.02 ( clear the surface
G0X0Y0 ( home spindle back to bottom left corner
G0Z0 ( touch the surface
M2 ( that is all!
```

From the file above we can see the CGODE instructions, one per line, the structure is,

- Preliminary comments preceded with '(' comment out rest of line instruction
- Setup instructions such as setting the feedrate, selecting the first tool a 20mil twist drill etc.
- The Subroutine section which contains routines for hole drilling and raising/lowering the isolation engraving tool.
- The MAIN section of the program,
    - where one hole is drilled,
    - followed by a tool change to the isolation engraving tool,
    - Isolation cutting of one track,
    - returning to the start position,
    - and finally the all important M2 end command.

You can open the Help Browser and check out the ANCA Coding Help to understand what each instruction actually does.

2.2 Step 2, G-code to Intermediate code

So the command files required are,

1. The 'ANC' G-code master control file,
2. Any subroutine 'ANC' files to be included and
3. The 'OVL' overlay file to act as a visual template.

When compiled the following files are produced in the 'src' directory of the Project,

.RUNMILL.1	Master control file
.SRUNMILL.1	First control subroutine segment
.SRUNMILL.999	Up to the last subroutine segment
NAME.OVL	The visual overlay template.

2.3 Step-3, Intermediate code to Instruction code

When compiled the following files are produced in the 'src' directory of the Project,

.MILLCMDS.CMD

Take note of the 'dot' in front of some files, you need to set the Linux file browser Nautilus to,

View>Show Hidden Files

to be able to see them. They are hidden because they are not to be altered by the Operator, to do so could create a dangerous incident as the machine may be damaged or the Operator could be put at risk.

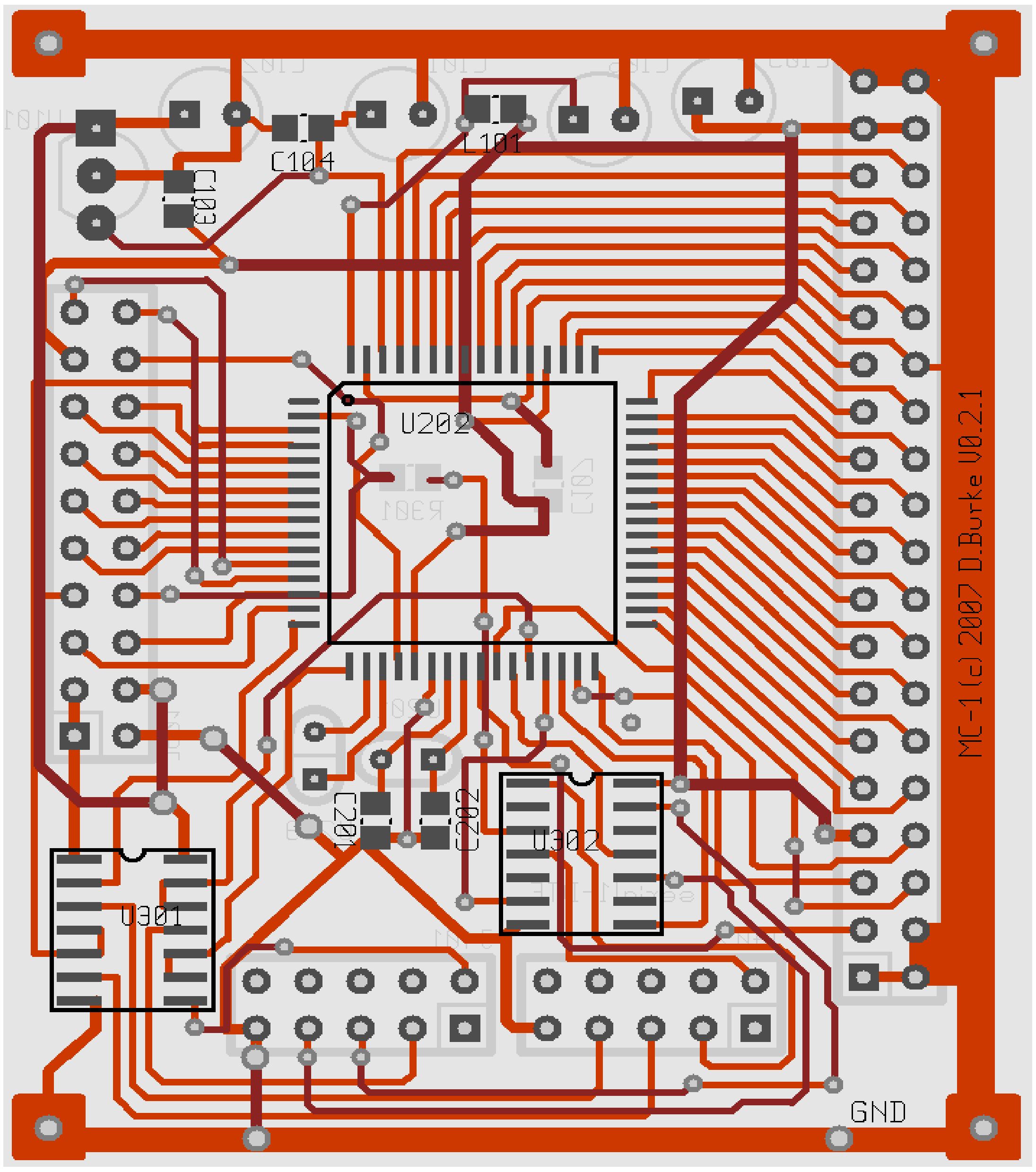
2.4 Sending the Instruction Code to the machine

Having simulated the raw '.MILLCMDS.CMD' command file the user could then use one of the provided 'Drive' programs such as 'Lathedrive' to send the step commands to the MC-1 Machine Controller on the target machine via a serial RS232C communication line. Notice that the commands are the individual step commands for that particular machine. This command file is much too large to fit into the Machine Controller all in one go so there is a dialogue set up between the MC-1 and the PC. A cue is established on board the MC-1 and one thousand steps are sent. Note here that these steps are not sent in real time as Linux is not made that way (there is 'Real Time Linux' but we are presuming just the standard version here). The Machine Controller then activates the Axis Drives in real time incorporating acceleration and de-acceleration ramping. At some point the step cue is smaller so more steps are then ordered from the PC until all of the '.MILLCMDS.CMD' command file has been sent. If there is a fault during this dialogue an emergency stop command is sent to the PC to warn the Operator of the fault condition.

3 MC-1 Machine Controller Hardware Description

3.1 MC-1 Machine Controller Central Processing Unit

The 'heart' of the Machine Controller is the central processing unit MC-1 which comprises the Atmel AVR microcomputer chip, power conditioning components, two serial ports, a twenty pin Auxiliary port and a forty pin General Expansion port. To make a complete Machine Controller there will be the need to add the Axis Controller Board and the Spindle Drive Board. To continue we will start with describing Ports available on the MC-1.



Screen shot of the MC-1 board from the 'pcb' program.

3.1.1 MC-1 Auxiliary Input Port

The 20 pin Auxiliary Port serves two purposes. It acts as the on-board programming connector to allow the transfer of the control code to the AVR's Flash memory and doubles as an input/output port when in use as a controller.

3.1.2 MC-1 General Expansion Port

The General Expansion Port is connected via a 40 pin IDC ribbon cable to the Axis Controller Board and carries the commands necessary to control axis movement.

3.1.3 Programming the MC-1

A simple interface board is provided which when connected to the PC and Auxiliary Port provides two way communication between the two. Refer to the Practical 'Demo-8' provided in the side4linux PCBS Help section for detailed instructions on programming the MC-1. Graphical programming software 'dbavrprog' is provided for programming the MC-1 from the side4linux IDE as an embedded 'tool'. Also you may download and install 'avrdude' as the programming adapter emulates the **'dapa'** (**'D**irect **A**VR **P**arallel **A**ccess cable**'**). The programming adapter software is designed to use the first parallel printer port also known as 'LPT:1'.

3.1.4 MC-1 Serial Ports

The MC-1 board provides two serial ports, serial0 and serial1 for communication between the controlling Linux PC and the Machine being controlled. Note that the serial0 port which is a Data Communications Equipment port (DCE) connects to the control PC and Serial1 which is a Data Terminal Equipment port (DTE) connects to a Slave serial port.

DB9 Female Serial Port-0 (DCE)			***DB9 Male PC Serial Port (DTE)***		
Pins	**Name**	**Function**	**Name**	**Function**	**Direction**
1	N/C	Not Connected	CD	Carrier Detect	In
2	TX-1	Transmitted Data	RD	Received Data	In
3	RX-1	Received Data	TD	Transmitted Data	Out
4	DTR-IN	Data Terminal Ready In	DTR	Data Terminal Ready	Out
5	GND	Signal Ground	GND	Signal Ground	<-->
6	DSR-OUT	Data Set Ready Out	DSR	Data Set Ready	In
7	N/C	Not Connected	RTS	Request To Send	Out
8	N/C	Not Connected	CTS	Clear To Send	In
9	N/C	Not Connected	RI	Ring Indicator	In

DB9 Male Serial Port-1 (DTE)			***DB9 Female External Slave Port (DCE)***		
Pins	**Name**	**Function**	**Name**	**Function**	**Direction**
1	N/C	Not Connected	N/C	Not Connected	
2	RX-2	Received Data	TD	Transmitted Data	Out
3	TX-2	Transmitted Data	RD	Received Data	In
4	DTR-OUT	Data Terminal Ready OUT	DTR-IN	Data Terminal Ready In	In
5	GND	Signal Ground	GND	Signal Ground	<-->
6	DSR-IN	Data Set Ready In	DSR-OUT	Data Set Ready Out	Out
7	N/C	Not Connected	N/C	Not Connected	
8	N/C	Not Connected	N/C	Not Connected	
9	N/C	Not Connected	N/C	Not Connected	

3.1.5 Communicating with the MC-1 from a PC

A 'monitor' program is provided in the side4linux package to allow for diagnostic operations on the MC-1 so refer to the Practicals provided in the side4linux Help.

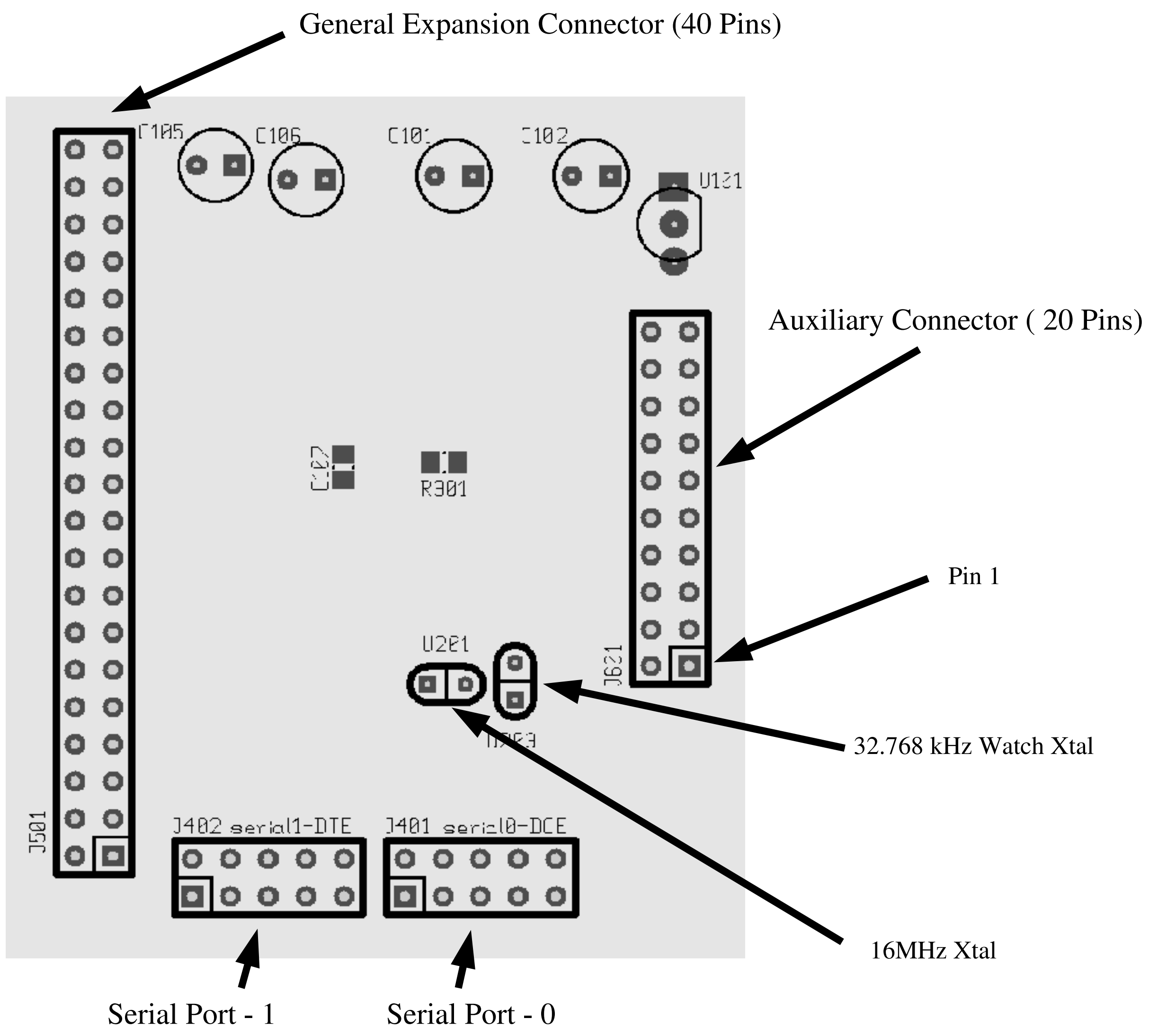
The controlling computer is an x86 PC running the Linux Operating System with two serial ports which we will call 'COM1' and 'COM2'. Both of the PC's serial ports are configured as DTE which means on a modern PC that they are male DB9 (9 pin) sockets. The Machine Controller has also got two serial ports which we will call 'serial0' and 'serial1'. 'serial0' is configured as DCE and 'serial1' as DTE. Points to be considered are,

- The ports on the Machine Controller will need to be a female DB9 outlet socket (DCE) for serial0 and a male DB9 outlet socket (DTE) for serial1.
- Communication between the two devices will be at a rate of 9600 Baud, one start bit and two stop bits with no Parity checking using 7 bit ASCII code.
- IDC (Insulation Displacement Connection) cables need to be made which will connect the 10 pin male headers on the Machine Controller to the DB9 outlet sockets.
- A communication cable (available commercially as a serial modem cable) will be needed to connect the Machine Controller DCE port 'serial0' to one of the DTE ports on the PC (usually COM2).
- Software will be required on the PC to take the raw 'Instruction Code' or 'Monitor' commands generated and transfer them to/from the Machine Controller (provided in the side4linux package).
- There needs to be a 'Monitor' program on board the Machine Controller to respond to 'Machine Instructions' or 'Monitor Commands' sent from the PC (provided in the side4linux package).
- There needs to be diagnostic routines on board the Machine Controller to read/alter register and memory locations (provided in the side4linux package).
- There needs to be a handshake line from the Machine Controller to the controlling Linux PC to indicate that the Machine Controller is ready to receive data or commands.
- There needs to be a DTE port available on the Machine Controller ('serial1') to communicate with either a machine to be controlled or another PC for operational and diagnostic purposes.
- There needs to be a handshake line from either a machine to be controlled or another PC for operational and diagnostic purposes to the Machine Controller to indicate to the Machine Controller when the machine or other PC is ready to receive instructions.

The ten pin male headers of the Machine Controller's two serial ports ('serial0' and 'serial1') are connected to their male and female sockets respectively via Insulation Displacement Ribbon Cables. Note that pin-1 is redundant and may be removed and pressed into the ribbon cable connector to provide for connector polarisation. Connect with the recommended flat ribbon IDC cable according to the following pin out table,

10 Pin Headers viewed from top			***IDC DB9 Socket pin out***	
Pin Number	SERIAL0 (DCE)	SERIAL1 (DTE)	Pin Number	RS232C Signal Name
1	n/c	n/c	1	no connect
2	TX1	RX2	2	TD/RD
3	RX1	TX2	3	RD/TD
4	DTR-IN	DTR-OUT	4	DTR
5	GND	GND	5	Signal Ground
6	n/u	n/u	not used	not used
7	n/c	n/c	9	no connect
8	n/c	n/c	8	no connect
9	n/c	n/c	7	no connect
10	DSR-OUT	DSR-IN	6	DSR

3.1.6 Layout of the MC-1 CPU board



Notice that Pin-1 on all headers is the square pin!

In the case of electrolytic capacitors positive is the square pin!

3.1.7 MC-1 CPU Auxiliary Port Pin-outs

Note that pin-1 is redundant and may be removed and pressed into the ribbon cable connector to provide for connector polarisation.

MC-1 Auxiliary Port Pin-outs

Pin Number	Net	Pin Number	Net
1	-9V	11	PE7
2	Gnd	12	PE6
3	-9V	13	PE5
4	+9V	14	PE4
5	PB5	15	PE3
6	PB4	16	PE2
7	PE1	17	+5V
8	PE0	18	RESET
9	SCK	19	MOSI
10	PB0	20	MISO

3.1.8 MC-1 CPU Expansion Port Pin-outs

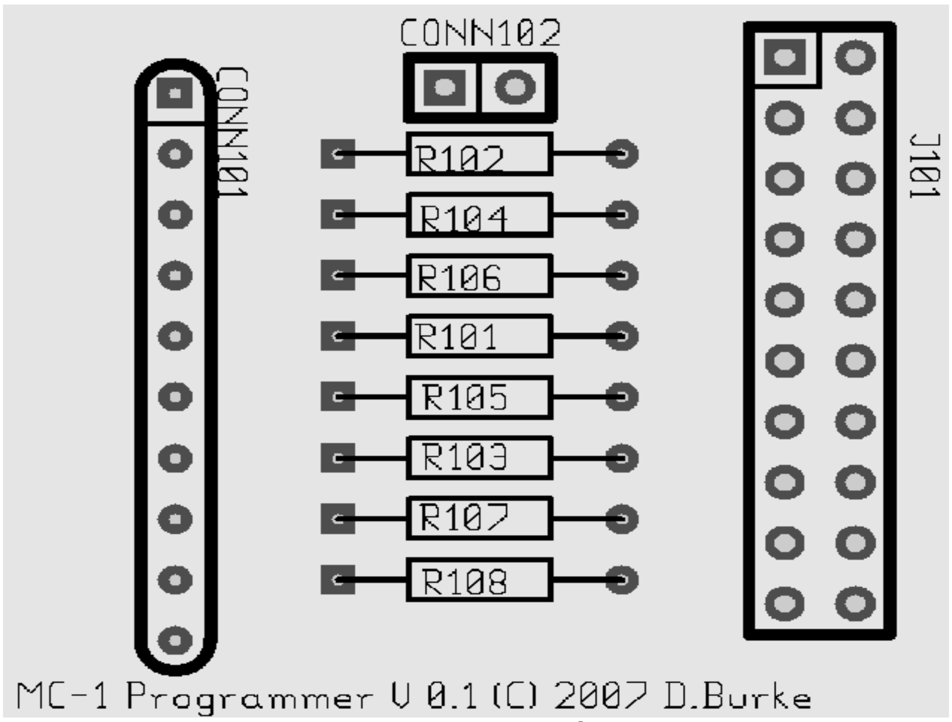
Expansion Port connection is made through a 40 pin IDC ribbon cable (available commercially as an IDE hard disk drive 40 pin PATA cable) to the Axis Control Board. Note that pin-1 is redundant and may be removed and pressed into the ribbon cable connector to provide for connector polarisation.

MC-1 Expansion Port Pin-outs

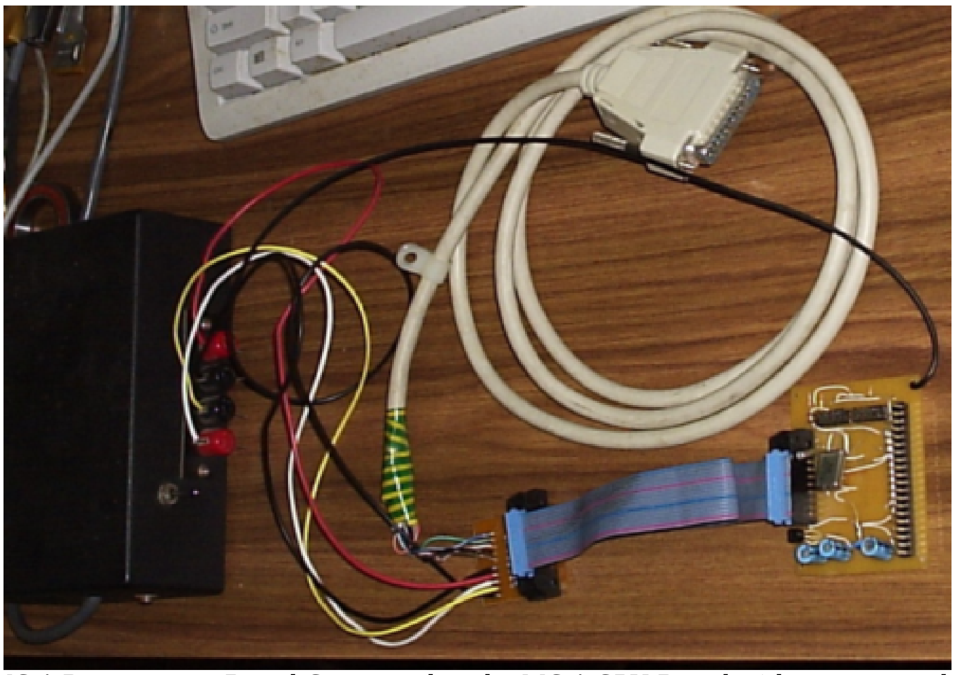
Pin Number	**Net**	**Pin Number**	**Net**
1	GND	21	PA6
2	GND	22	PA5
3	PD0	23	PA4
4	GND	24	PA3
5	PD7	25	PA2
6	PD1	26	PA1
7	+5V	27	PA0
8	PD6	28	GND
9	PG1	29	PF7
10	PG0	30	PF6
11	PC0	31	PF5
12	PC1	32	PF4
13	PC2	33	PE3
14	PC3	34	PE2
15	PC4	35	PE1
16	PC5	36	PE0
17	PC6	37	+5V
18	PC7	38	+5V
19	PG2	39	GND
20	PA7	40	GND

3.2 MC-1 Machine Controller Programmer Board

The MC-1 Programmer board is basically just a parallel printer cable that has had the printer plug cut off and then wired into an interface printed circuit board that provides cleaned up signals. These programming signals are presented to the MC-1 CPU board via a short ribbon cable connected to the Auxiliary Port. The basic layout of the programmer board is shown below followed by a photo which shows it connected to a MC-1 CPU.



MC-1 Programmer Board Layout



MC-1 Programmer Board Connected to the MC-1 CPU Board with power supply.