

---

# SLCD5 / SLCD5+ Reference Manual

## Version 1.6

Firmware Version 1.4.0 and above; BMPload 2.3.1 and above

August 4, 2011  
Hardware Revisions A thru D

---

© Copyright Reach Technology Inc. 2003-2011  
All Rights Reserved

*Note: The software included with this product is subject to a license agreement as described in this Manual.*

[www.reachtech.com](http://www.reachtech.com)

Sales: 510-770-1417  
[sales@reachtech.com](mailto:sales@reachtech.com)

Technical Support: 503-675-6464  
[techsupport@reachtech.com](mailto:techsupport@reachtech.com)

# Table of Contents

<b>0.</b>	<b>HARDWARE LIMITED WARRANTY AND SOFTWARE LICENSE AGREEMENT .....</b>	<b>9</b>
0.1.	HARDWARE LIMITED WARRANTY .....	9
0.2.	RETURNS AND REPAIR POLICY .....	9
0.3.	SOFTWARE LICENSE AGREEMENT .....	9
<b>1.</b>	<b>INTRODUCTION .....</b>	<b>12</b>
1.1.	OVERVIEW .....	12
1.2.	FEATURES .....	13
1.3.	ELECTRICAL CHARACTERISTICS .....	13
1.4.	PANEL SUPPORT .....	14
1.5.	SD CARD SUPPORT .....	14
1.6.	DIMENSIONS .....	15
<b>2.</b>	<b>CONNECTORS AND JUMPERS .....</b>	<b>16</b>
2.1.	OVERVIEW .....	17
2.2.	J6 - POWER AND COMMUNICATION COM0 (RS232 MODE) .....	18
2.3.	J6 - POWER AND COMMUNICATION COM0 (3.3V CMOS MODE) .....	19
2.4.	J7 - COMMUNICATION COM1 AND RESET .....	20
2.5.	J4 - POWER .....	22
2.6.	J3 - 4 WIRE TOUCH (HARDWARE VERSION DEPENDENT) .....	22
2.7.	J11 - 4 WIRE TOUCH (HARDWARE VERSION DEPENDENT) .....	22
2.8.	J12 - 4 WIRE TOUCH (HARDWARE VERSION DEPENDENT) .....	23
2.9.	J9 - 33 PIN 0.5MM FLAT FLEX LCD CONNECTOR .....	23
2.10.	J8 - 31 PIN 1MM LCD CONNECTOR .....	24
2.11.	J10 - BACKLIGHT / INVERTER CONTROL .....	25
2.12.	J13 - BACKLIGHT / INVERTER CONTROL .....	25
2.13.	J5 - EXTERNAL AUDIO OUTPUT (NOT TYPICALLY USED) .....	26
2.14.	J15 SD CARD CONNECTOR .....	26
2.15.	J14 - FOR EXTERNAL SD CARD .....	27
2.16.	JP5 - SD CARD POWER .....	27
2.17.	JP2 - DEPRECATED FOR REV B AND ABOVE .....	27
<b>3.</b>	<b>CONFIGURATION GUIDE .....</b>	<b>28</b>
3.1.	POWER CONNECTIONS .....	28
3.2.	SERIAL .....	28
3.3.	TFT HARDWARE PANEL ORIENTATION .....	29
3.4.	SYSTEM CONFIGURATION .....	29
<b>4.</b>	<b>SYSTEM OVERVIEW .....</b>	<b>33</b>
4.1.	GENERAL SLCD5 CONTROLLER INFORMATION .....	33
4.2.	COMPATIBILITY WITH SLCD CONTROLLER .....	33
4.3.	BITMAPS AND MACROS .....	33
4.4.	OVERVIEW - SLCD5 EVALUATION KITS .....	34
4.5.	GETTING STARTED .....	34
4.6.	CONNECTING THE KIT TO A PC .....	35
4.7.	SWITCHING BETWEEN PC AND EMBEDDED CONTROLLER .....	35
4.8.	POWERCOM4 SCHEMATIC .....	37

4.9.	POWERCOM4 OPERATIONAL NOTES .....	38
	<i>Operational Notes</i> .....	38
4.10.	COMMUNICATIONS INTERFACE .....	39
	<i>General</i> .....	39
4.11.	SLCD5 INPUT BUFFER PROCESSING.....	39
	<i>Input Buffer</i> .....	39
	<i>Flow Control</i> .....	40
	<i>Buffer Limit Discussion</i> .....	40
	<i>Prompt Summary</i> .....	41
4.12.	TOUCH INTERFACE.....	41
	<i>Hotspot</i> .....	41
	<i>Button</i> .....	41
	<i>Host Notification</i> .....	42
4.13.	HOST INPUT PROCESSING .....	42
4.14.	SYSTEM FIRMWARE BOOT PROCESS .....	42
	<i>Algorithm</i> .....	43
<b>5.</b>	<b>SOFTWARE COMMAND REFERENCE.....</b>	<b>45</b>
5.1.	COMMANDS AFFECTING NON-VOLATILE SETTINGS (EEPROM) .....	45
5.2.	COMMAND BASICS .....	45
5.3.	COMPRESSED SYNTAX .....	45
5.4.	ASSUMED ORIENTATION .....	45
5.5.	TOUCH PRIORITY .....	46
5.6.	COMMANDS.....	46
	<i>SET PEN WIDTH</i> .....	46
	<i>SET DRAW MODE</i> .....	46
	<i>SET CURSOR</i> .....	46
	<i>SET TEXT ALIGNMENT</i> .....	47
	<i>SET TEXT MODE</i> .....	47
	<i>SET ORIGIN</i> .....	48
	<i>SET COLOR (BASIC)</i> .....	49
	<i>SET COLOR (DETAILED)</i> .....	50
	<i>SET FONT</i> .....	50
	<i>CLEAR SCREEN</i> .....	51
	<i>SET UTF8 ENCODING</i> .....	51
	<i>DISPLAY BITMAP IMAGE</i> .....	51
	<i>DISPLAY BITMAP IMAGE CENTERED</i> .....	52
	<i>DISPLAY CLIPPED BITMAP IMAGE</i> .....	52
	<i>DISPLAY WINDOWED BITMAP IMAGE</i> .....	53
	<i>DISPLAY IMAGE FILE</i> .....	53
	<i>TEXT DISPLAY</i> .....	54
	<i>GET TEXT DISPLAY WIDTH IN PIXELS</i> .....	56
	<i>TEXT FLASHING DISPLAY</i> .....	56
	<i>TEXT FLASHING DISABLE</i> .....	57
	<i>TEXT FLASHING ENABLE</i> .....	57
	<i>TEXT FLASHING DELETE</i> .....	57
	<i>TEXT FLASHING SYNCHRONIZATION</i> .....	58
	<i>TEXT FLASH ANIMATION ENABLE</i> .....	58
	<i>SAVE DRAWING ENVIRONMENT (STATE SAVE)</i> .....	58
	<i>RESTORE DRAWING ENVIRONMENT (STATE RESTORE)</i> .....	58
	<i>PIXEL READ / WRITE</i> .....	59

<b>DRAW LINE</b> .....	59
<b>DRAW RECTANGLE</b> .....	60
<b>DRAW CIRCLE</b> .....	60
<b>DRAW TRIANGLE</b> .....	61
<b>DRAW OUTLINE POLYGON</b> .....	61
<b>DRAW FILLED POLYGON</b> .....	61
<b>DRAW ROTATED POLYGON</b> .....	62
<b>DRAW ROTATED FILLED POLYGON</b> .....	62
<b>REDRAW ROTATED POLYGON</b> .....	62
<b>DRAW POLYLINE</b> .....	62
<b>DRAW ROTATED POLYLINE</b> .....	63
<b>DRAW FILLED ELLIPSE</b> .....	63
<b>DRAW ELLIPSE</b> .....	63
<b>DRAW ARC SEGMENT</b> .....	64
<b>SCROLL SCREEN AREA</b> .....	64
<b>BUTTON DEFINE - MOMENTARY</b> .....	65
<b>BUTTON DEFINE – MOMENTARY (continued)</b> .....	66
<b>BUTTON DEFINE – LATCHING STATE</b> .....	67
<b>BUTTON DEFINE CENTER TEXT</b> .....	68
<b>DEFINE HOTSPOT (VISIBLE TOUCH AREA)</b> .....	69
<b>DEFINE SPECIAL HOTSPOT (INVISIBLE TOUCH AREA)</b> .....	69
<b>DEFINE SPECIAL HOTSPOT (INVISIBLE, NO BEEP)</b> .....	69
<b>DEFINE TYPOMATIC TOUCH AREA</b> .....	70
<b>DEFINE SPECIAL TYPOMATIC TOUCH AREA</b> .....	70
<b>DEFINE X-Y HOTSPOT</b> .....	70
<b>DEFINE X-Y HOTSPOT (SILENT – NO BEEP)</b> .....	70
<b>CHANGE (LATCHING) STATE BUTTON</b> .....	71
<b>BUTTON CLEAR</b> .....	71
<b>TOUCHSCREEN ON/OFF</b> .....	71
<b>DISABLE TOUCH (HOTSPOT / BUTTON)</b> .....	72
<b>ENABLE TOUCH (HOTSPOT / BUTTON)</b> .....	72
<b>SET TOUCH CHARACTERISTICS</b> .....	73
<b>CLEAR HOTSPOT</b> .....	73
<b>CLEAR ALL TOUCH</b> .....	73
<b>SLIDER DEFINE</b> .....	74
<b>SLIDER VALUE</b> .....	74
<b>METER DEFINE</b> .....	75
<b>METER VALUE</b> .....	76
<b>CHART BITMAP DEFINE</b> .....	76
<b>CHART DEFINE</b> .....	77
<b>CHART REDEFINE PEN</b> .....	78
<b>CHART REDEFINE BACKGROUND COLOR</b> .....	78
<b>CHART REDEFINE BACKGROUND BITMAP</b> .....	78
<b>CHART REDEFINE RETRACE MODE</b> .....	78
<b>CHART REDEFINE CLEAR-BEFORE-DRAW MODE</b> .....	79
<b>CHART VALUES</b> .....	79
<b>BINARY CHART VALUES</b> .....	80
<b>CHART RESET PENS TO START</b> .....	80
<b>CHART CLEAR DISPLAY AREA</b> .....	81
<b>LEVELBAR DEFINE</b> .....	81
<b>LEVELBAR VALUE</b> .....	82

<b>MACRO EXECUTE</b> .....	82
<b>MACRO ABORT</b> .....	83
<b>TOUCH MACRO ASSIGN</b> .....	83
<b>TOUCH MACRO ASSIGN QUIET</b> .....	84
<b>TOUCH MACRO ASSIGN WITH PARAMETERS</b> .....	85
<b>TOUCH MACRO ASSIGN WITH PARAMETERS (continued)</b> .....	86
<b>ANIMATION DEFINE</b> .....	86
<b>ANIMATION LIST</b> .....	89
<b>ANIMATION YIELD</b> .....	90
<b>ANIMATION DISABLE</b> .....	90
<b>ANIMATION ENABLE</b> .....	90
<b>ANIMATION CLEAR</b> .....	90
<b>ANIMATION DELETE</b> .....	91
<b>ANIMATION SYNCH</b> .....	91
<b>WAIT VERTICAL RETRACE</b> .....	91
<b>WAIT FOR REFRESH</b> .....	92
<b>OUTPUT STRING</b> .....	92
<b>WRITE TO AUX PORT</b> .....	93
<b>READ FROM AUX PORT</b> .....	93
<b>SPEAKER ON / OFF (Rev D and later boards only)</b> .....	93
<b>BEEP ONCE</b> .....	94
<b>BEEP WAIT</b> .....	94
<b>BEEP VOLUME, BEEP VOLUME SPECIAL</b> .....	94
<b>BEEP FREQUENCY, BEEP FREQUENCY SPECIAL</b> .....	95
<b>BEEP REPEAT</b> .....	95
<b>BEEP TOUCH</b> .....	96
<b>ALARM</b> .....	96
<b>WAIT</b> .....	96
<b>DISPLAY ON/OFF</b> .....	96
<b>EXTERNAL BACKLIGHT ON/OFF</b> .....	97
<b>EXTERNAL BACKLIGHT BRIGHTNESS CONTROL</b> .....	97
<b>SET BAUD RATE OF COM0 OR COM1 PORT</b> .....	97
<b>SET BAUD RATE OF COM0 OR COM1 PORT (but output prompt first)</b> .....	98
<b>SET BAUD RATE OF COM0 OR COM1 PORT(sticky and prompt first)</b> .....	98
<b>TOUCH CALIBRATE</b> .....	98
<b>RESET TOUCH CALIBRATION</b> .....	99
<b>VERSION</b> .....	99
<b>SET LED</b> .....	99
<b>READ FRAME BUFFER LINE</b> .....	99
<b>CRC SCREEN</b> .....	99
<b>CRC DATA FLASH</b> .....	100
<b>CRC PROCESSOR FIRMWARE CODE</b> .....	100
<b>CRC PROCESSOR BOOTLOAD CODE</b> .....	100
<b>LIST FLASH DATA RECORDS</b> .....	100
<b>LIST BITMAPS DETAIL</b> .....	100
<b>LIST MACROS DETAIL</b> .....	101
<b>LAST FIRMWARE FILE LOADED</b> .....	101
<b>READ TEMPERATURE</b> .....	101
<b>RESET SOFTWARE</b> .....	101
<b>RESET BOARD TO MANUFACTURED STATE</b> .....	102
<b>LOAD SYSTEM FILE FROM SD CARD (CURRENT DIRECTORY)</b> .....	102

	<i>LOAD SYSTEM FILE FROM SD CARD (SPECIFIED DIRECTORY)</i> .....	102
	<i>CHANGE SD CARD DIRECTORY</i> .....	102
	<i>LIST SD CARD DIRECTORY</i> .....	103
	<i>SAVE SCREEN SHOT TO SD CARD</i> .....	103
	<i>DEBUG TOUCH</i> .....	103
	<i>DEBUG MACRO</i> .....	103
	<i>MACRO NOTIFY</i> .....	104
	<i>SPLASH SCREEN</i> .....	104
	<i>POWER-ON MACRO</i> .....	105
	<i>SET VARIABLE</i> .....	106
	<i>GET VARIABLE</i> .....	106
	<i>BINARY NOTIFICATION MODE</i> .....	107
	<i>GET PANEL TYPE</i> .....	108
	<i>CONTROL PORT AUTO SWITCH</i> .....	108
	<i>SET CONTROL PORT</i> .....	108
	<i>SET PREVIOUS CONTROL PORT</i> .....	108
	<i>SET TYPEMATIC PARAMETERS</i> .....	109
	<i>SET TOUCH DEBOUNCE</i> .....	109
	<i>DEFINE PANEL ORIENTATION</i> .....	110
	<i>DEFINE INVERTER CONTROL POLARITY</i> .....	110
	<i>DEFINE TOUCH SIGNAL ORIENTATION</i> .....	110
	<i>DEFINE INVERTER PWM CONTROL</i> .....	111
	<i>DEFINE MAX, MIN BRITE</i> .....	111
	<i>DEFINE TOUCH ACTION</i> .....	112
	<i>DEFINE TOUCH PARAMETERS</i> .....	112
	<i>DEFINE TOUCH CALIBRATION TIMEOUT</i> .....	112
	<i>DEFINE AUX ESCAPE</i> .....	113
	<i>DISPLAY CONFIG STRING</i> .....	113
	<i>PANEL TIMING ADJUST</i> .....	113
	<i>EEPROM READ, WRITE</i> .....	114
	<i>SPEED TEST</i> .....	114
	<i>COPY FLASH to DRAM (SLCD5 PLUS only)</i> .....	114
	<i>LOAD SYSTEM FILE FROM SD CARD INTO FLASH (SLCD5 PLUS only)</i> .	115
	<i>CONTROLLER TYPE</i> .....	115
<b>6.</b>	<b>FONTS</b> .....	<b>116</b>
6.1.	EXTERNAL FONTS .....	116
6.2.	CHARACTER SET - ISO 8859-1 .....	116
<b>7.</b>	<b>USING CRC'D COMMANDS</b> .....	<b>118</b>
7.1.	OVERVIEW .....	118
7.2.	COMMAND PROTOCOL.....	118
7.3.	EXAMPLE CRC GENERATION CODE .....	118
<b>8.</b>	<b>COMMUNICATIONS WATCHDOG TIMER</b> .....	<b>121</b>
8.1.	OVERVIEW .....	121
8.2.	USING THE ‘*COMWDT’ COMMAND .....	121
8.3.	EXAMPLE .....	122
<b>9.</b>	<b>WORKING WITH VARIABLES</b> .....	<b>123</b>
9.1.	OVERVIEW .....	123
9.2.	USER VARIABLES.....	123

9.3.	SYSTEM VARIABLES .....	124
9.4.	FORMATTING VARIABLES.....	125
<b>10.</b>	<b>USING SIMPLE MATH EXPRESSIONS .....</b>	<b>127</b>
10.1.	OVERVIEW .....	127
10.2.	LIMITATIONS AND REQUIREMENTS .....	127
10.3.	EXAMPLES.....	128
<b>APPENDIX A - LCD AND TOUCH PANELS COMPATIBLE WITH THE SLCD5</b>		
	<b>CONTROLLER.....</b>	<b>129</b>
A.1	LCD PANELS.....	129
A.2	TOUCH PANEL.....	129
<b>APPENDIX B - PARTS AND SUPPLIERS FOR SLCD5 CONTROLLER</b>		
	<b>CONNECTIONS.....</b>	<b>130</b>
B.1	CONNECTORS AND CABLES FOR J6, J7, J10, J13 .....	130
B.2	CABLES FOR J8.....	130
B.3	DISCRETE WIRE CABLE VENDORS .....	130
<b>APPENDIX C - ORDERING INFORMATION.....</b>		<b>131</b>
C.1	GENERAL INFORMATION.....	131
C.2	CONTACT REACH DIRECTLY FOR SPECIFIC ORDERING INFORMATION.....	131
<b>APPENDIX D - BMPLOAD PROGRAM.....</b>		<b>132</b>
D.1	OVERVIEW .....	132
D.2	BITMAP FORMAT.....	132
D.3	PROGRAM OPERATION.....	133
D.4	CONNECTING VIA SERIAL PORT.....	136
D.5	BMPLOAD SPEED ISSUES.....	137
<b>APPENDIX E – MACRO COMMANDS AND FILE FORMAT .....</b>		<b>138</b>
E.1	INTRODUCTION AND LIMITATIONS.....	138
E.2	MACRO FILE FORMAT.....	138
E.3	MACRO PARAMETERS (ARGUMENTS).....	140
E.4	ASSIGNING MACROS TO BUTTONS.....	140
E.5	SPECIAL MACRO COMMANDS, AND MACRO LABELS .....	140
	<i>Memory commands</i> .....	<b>140</b>
	<i>Special Arguments</i> .....	<b>141</b>
	<i>Repeat command</i> .....	<b>141</b>
	<i>Labels</i> .....	<b>142</b>
E.5	CHANGING THE FIRMWARE POWER-ON BAUD RATE .....	143
E.6	CHANGING THE BOOTLOADER AND FIRMWARE POWER-ON BAUD RATE .....	143
<b>APPENDIX F – TROUBLESHOOTING.....</b>		<b>145</b>
F.1	TOUCH UNRELIABLE OR NON-OPERATIVE.....	145
<b>APPENDIX G – EVALUATION / DEMO KIT TUTORIAL .....</b>		<b>146</b>
G.1	SELF-RUNNING DEMONSTRATION .....	146
G.2	CONNECTION AND CONTROL VIA PC .....	146
G.3	SIMPLE COMMANDS .....	146
G.4	BITMAPS.....	147
G.5	MACROS.....	148
G.6	FONTS .....	148

G.7	ATTACHING TO THE EMBEDDED CONTROLLER .....	149
<b>APPENDIX H – WORKING WITH BITMAPS .....</b>		<b>150</b>
H.1	CREATING BITMAPS .....	150
H.2	HIGH COLOR .....	150
H.3	GRADIENTS .....	150
H.4	TRANSPARENCY .....	151
<b>APPENDIX J – SD CARD SUPPORT FEATURES .....</b>		<b>152</b>
J.1	OVERVIEW .....	152
J.2	FIRMWARE UPGRADE .....	152
J.3	HOLDING BINARY WORKING SET FILE .....	153
J.4	UPDATE BINARY FILE IN FLASH .....	153
J.5	SCREEN SHOT SAVING .....	153
	<i>LIST SD CARD DIRECTORY .....</i>	<i>Error! Bookmark not defined.</i>
J.6	RUN DEMO MACRO .....	154
J.7	MULTIPLE BINARY FILES .....	154
J.8	SEPARATE IMAGE FILES .....	154
<b>APPENDIX K – SD CARD REMOTE .....</b>		<b>156</b>
K.1	OVERVIEW .....	156
K.2	SCHEMATIC .....	157
L.1	REV A VS. REV B BOARD FAB REV A VS. REV B BOARD FAB .....	158
L.2	REV B VS. REV D BOARD FAB .....	158
<b>APPENDIX M - SOFTWARE MANUAL CHANGE HISTORY .....</b>		<b>159</b>

## Figures

FIGURE 1: SLCD5 PLUS (REV D) CONTROLLER BOARD .....	12
FIGURE 2: SLCD5 DIMENSIONS AND CONNECTOR LOCATIONS (INCHES) .....	15
FIGURE 3: CONNECTORS AND JUMPERS (SLCD5 PLUS REV D SHOWN) .....	16
FIGURE 4: TYPICAL (RS-232) CONNECTION FROM A PC .....	19
FIGURE 5: POWERCOM4 BOARD SUPPLIED WITH EVALUATION KIT .....	36
FIGURE 6: POWERCOM4 SCHEMATIC .....	37
FIGURE 7: SYSTEM SOFTWARE BOOT ALGORITHM .....	44
FIGURE 8: SD CARD SCHEMATIC .....	157



## **0. Hardware Limited Warranty and Software License Agreement**

### **0.1. Hardware Limited Warranty**

REACH TECHNOLOGY, Inc. warrants its hardware products to be free from manufacturing defects in materials and workmanship under normal use for a period of one (1) year from the date of purchase from REACH. This warranty extends to products purchased directly from REACH or an authorized REACH distributor. Purchasers should inquire of the distributor regarding the nature and extent of the distributor's warranty, if any. REACH shall not be liable to honor the terms of this warranty if the product has been used in any application other than that for which it was intended, or if it has been subjected to misuse, accidental damage, modification, or improper installation procedures. Furthermore, this warranty does not cover any product that has had the serial number altered, defaced, or removed. This warranty shall be the sole and exclusive remedy to the original purchaser. In no event shall REACH be liable for incidental or consequential damages of any kind (property or economic damages inclusive) arising from the sale or use of this equipment. REACH is not liable for any claim made by a third party or made by the purchaser for a third party. REACH shall, at its option, repair or replace any product found defective, without charge for parts or labor. Repaired or replaced equipment and parts supplied under this warranty shall be covered only by the unexpired portion of the warranty. Except as expressly set forth in this warranty, REACH makes no other warranties, expressed or implied, nor authorizes any other party to offer any warranty, including any implied warranties of merchantability or fitness for a particular purpose. Any implied warranties that may be imposed by law are limited to the terms of this limited warranty. This warranty statement supercedes all previous warranties, and covers only the Reach hardware. The unit's software is covered by a separate license agreement.

### **0.2. Returns and Repair Policy**

No merchandise may be returned for credit, exchange, or service without prior authorization from REACH. To obtain warranty service, contact the factory and request an RMA (Return Merchandise Authorization) number. Enclose a note specifying the nature of the problem, name and phone number of contact person, RMA number, and return address.

Authorized returns must be shipped freight prepaid to Reach Technology Inc. 4575 Cushing Parkway, Fremont, California 94538 with the RMA number clearly marked on the outside of all cartons. Shipments arriving freight collect or without an RMA number shall be subject to refusal. REACH reserves the right in its sole and absolute discretion to charge a 15% restocking fee, plus shipping costs, on any products returned with an RMA.

Return freight charges following repair of items under warranty shall be paid by REACH, shipping by standard ground carrier. In the event repairs are found to be non-warranty, return freight costs shall be paid by the purchaser.

### **0.3. Software License Agreement**

**PLEASE READ THIS SOFTWARE LICENSE AGREEMENT CAREFULLY BEFORE USING THE SOFTWARE.**

This License Agreement (“Agreement”) is a legal contract between you (either an individual or a single business entity) and Reach Technology Inc. (“Reach”) for software referenced in this

manual, which includes software embedded in the hardware product, and, as applicable, associated media, printed materials, and “online” or electronic documentation (the “Software”).

BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT INSTALL OR USE THE SOFTWARE. IF YOU HAVE PAID A FEE FOR THIS LICENSE AND DO NOT ACCEPT THE TERMS OF THIS AGREEMENT, REACH WILL REFUND THE FEE TO YOU PROVIDED YOU (1) DO NOT INSTALL THE SOFTWARE AND (2) RETURN ALL SOFTWARE, MEDIA AND OTHER DOCUMENTATION AND MATERIALS PROVIDED WITH THE SOFTWARE TO REACH TECHNOLOGY INC AT: REACH TECHNOLOGY INC., 4575 Cushing Parkway, Fremont, California 94538.

Reach Technology Inc. ("Reach") and its suppliers grant to Customer ("Customer") a nonexclusive and nontransferable license to use the Reach software ("Software") in object code form on one or more central processing units owned or leased by Customer or otherwise embedded in equipment provided by Reach.

EXCEPT AS EXPRESSLY AUTHORIZED ABOVE, CUSTOMER SHALL NOT: COPY, IN WHOLE OR IN PART, SOFTWARE OR DOCUMENTATION; MODIFY THE SOFTWARE; REVERSE COMPILE OR REVERSE ASSEMBLE ALL OR ANY PORTION OF THE SOFTWARE; OR RENT, LEASE, DISTRIBUTE, SELL, OR CREATE DERIVATIVE WORKS OF THE SOFTWARE.

Customer agrees that aspects of the licensed materials, including the specific design and structure of individual programs, constitute trade secrets and/or copyrighted material of Reach. Customer agrees not to disclose, provide, or otherwise make available such trade secrets or copyrighted material in any form to any third party without the prior written consent of Reach. Customer agrees to implement reasonable security measures to protect such trade secrets and copyrighted material. Title to Software and documentation shall remain solely with Reach.

**SOFTWARE LIMITED WARRANTY.** Reach warrants that for a period of ninety (90) days from the date of shipment from Reach: (i) the media on which the Software is furnished will be free of defects in materials and workmanship under normal use; and (ii) the Software substantially conforms to its published specifications. Except for the foregoing, the Software is provided AS IS. This limited warranty extends only to Customer as the original licensee. Customer's exclusive remedy and the entire liability of Reach and its suppliers under this limited warranty will be, at Reach's option, repair, replacement, or refund of the Software. In no event does Reach warrant that the Software is error free or that Customer will be able to operate the Software without problems or interruptions.

This warranty does not apply if the software (a) has been altered, except by Reach, (b) has not been installed, operated, repaired, or maintained in accordance with instructions supplied by Reach, (c) has been subjected to abnormal physical or electrical stress, misuse, negligence, or accident, or (d) is used in High Risk Activities.

**DISCLAIMER.** EXCEPT AS SPECIFIED IN THIS WARRANTY, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS, AND WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE, ARE HEREBY EXCLUDED TO THE EXTENT ALLOWED BY APPLICABLE LAW.

IN NO EVENT WILL REACH OR ITS SUPPLIERS BE LIABLE FOR ANY LOST REVENUE, PROFIT, OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL, OR PUNITIVE DAMAGES HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE EVEN IF REACH OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**HIGH RISK ACTIVITIES.** The Software Product is not fault-tolerant and is not designed, manufactured, or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software Product, or any software, tool, process, or service that was developed using the Software Product, could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). Accordingly, Reach and its suppliers and licensors specifically disclaim any express or implied warranty of fitness for High Risk Activities. You agree that Reach and its suppliers and licensors will not be liable for any claims or damages arising from the use of the Software Product, or any software, tool, process, or service that was developed using the Software Product, in such applications.

In no event shall Reach's or its suppliers' liability to Customer, whether in contract, tort (including negligence), or otherwise, exceed the price paid by Customer. The foregoing limitations shall apply even if the above-stated warranty fails of its essential purpose. **SOME STATES DO NOT ALLOW LIMITATION OR EXCLUSION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES.**

The above warranty DOES NOT apply to any beta software, any software made available for testing or demonstration purposes, any temporary software modules or any software for which Reach does not receive a license fee. All such software products are provided AS IS without any warranty whatsoever.

This License is effective until terminated. Customer may terminate this License at any time by destroying all copies of Software including any documentation. This License will terminate immediately without notice from Reach if Customer fails to comply with any provision of this License. Upon termination, Customer must destroy all copies of Software.

Software, including technical data, is subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Customer agrees to comply strictly with all such regulations and acknowledges that it has the responsibility to obtain licenses to export, re-export, or import Software.

This License shall be governed by and construed in accordance with the laws of the State of California, United States of America, as if performed wholly within the state and without giving effect to the principles of conflict of law. If any portion hereof is found to be void or unenforceable, the remaining provisions of this License shall remain in full force and effect. This License constitutes the entire License between the parties with respect to the use of the Software.

# 1. Introduction

## 1.1. Overview

The SLCD5 controller family includes the original SLCD5 and the current SLCD5+. This manual will refer to both as the SLCD5 except where their features and capabilities differ. The SLCD5 and SLCD5+ have the same base board (large board) but different attached CPU modules. The SLCD5+ module has more storage, a faster processor, and is recommended for new designs.

The SLCD5 controllers provide complete Graphical User Interface for embedded systems **using VGA, WVGA, and (SLCD5+ only) SVGA panels. Using the SLCD5 is simply the** quickest way to generate a user interface without a lot of graphical programming. It has a small size to fit in space-constrained applications.

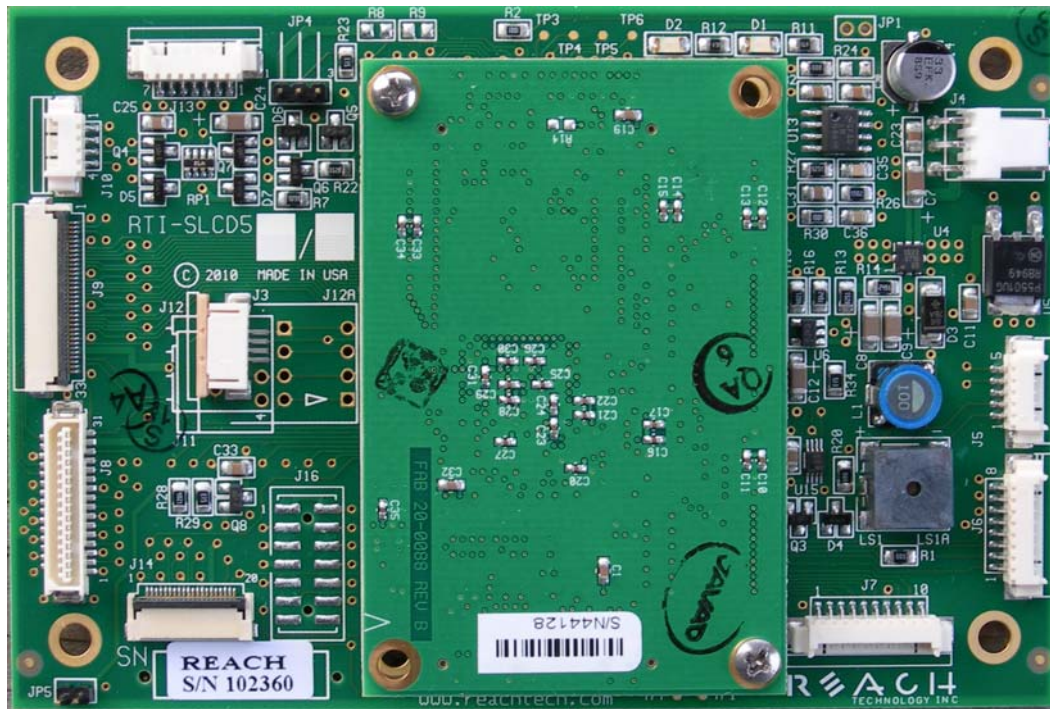


Figure 1: SLCD5+ (Rev D) controller board

## 1.2. Features

- Drives digital TFT displays at VGA, WVGA, and SVGA resolution
- 16-bit color (565 mapping - same as PC / Mac)
- Touch controller (4 wire resistive) on board
- Beeper for audible touch feedback and alarms
- 3" by 4.5" size, 0.55" thick
- High speed ARM9 processor (200+MHz)
- On-board RS232, RS485, CMOS level interfaces up to 230,400 baud
- Up to 28MB highly reliable NOR data flash for user downloadable bitmaps with optional RLE compression
- Backlight enable and brightness control
- SD card slot for firmware upgrades and bitmap / macro storage
- Can be modified for specific OEM requirements

## 1.3. Electrical Characteristics

In General, the SLCD5/5+ can be powered with 5-12V DC +/- 10%. It has an on-board switching regulator that generates the 3.3V required by the LCD panel and the SLCD5/5+ processor.

**NOTE: refer to the appropriate Data Sheet for the Reach Display Module you are using. If you are integrating the SLCD5/5+ Controller Board with your own display panel, you must determine the appropriate power for your display.**

LCD panels that require 3.3V power and have a 3.3V "CMOS" level compatible data interface are supported. A list of compatible panels is provided in [Appendix A](#).

The power requirements shown below are typical values. The theoretical maximum when all parts are worst case is up to 2 times these values, but it is extremely unlikely that this will occur on a specific board. We suggest a 150% of the below values is a reasonable value.

**The following table gives the SLCD5 Board Only Power Supply requirement; for a total power budget you must include the input-referenced panel power plus backlight power.**

Supply Voltage:	Typical Current (beeper off):	Typical Current (beeper on full):	Typical Current (Vin -10% , speaker option, max volume):
5V nominal	260mA (typ.)	330mA (typ.)	530mA (typ.)
12V nominal	160mA (typ.)	210mA (typ.)	270mA (typ.)

#### 1.4. **Panel support**

See [Appendix A](#).

#### 1.5. **SD card support**

The SD card slot supports cards that are FAT 16 formatted, maximum 2GB. We have tested, and recommend SanDisk brand cards. You may need to format them on the PC before use with FAT (also known as FAT16) format selected (DO NOT USE “FAT32”).

Long filenames are not supported; only 8.3 format names can be used.

## 1.6. Dimensions

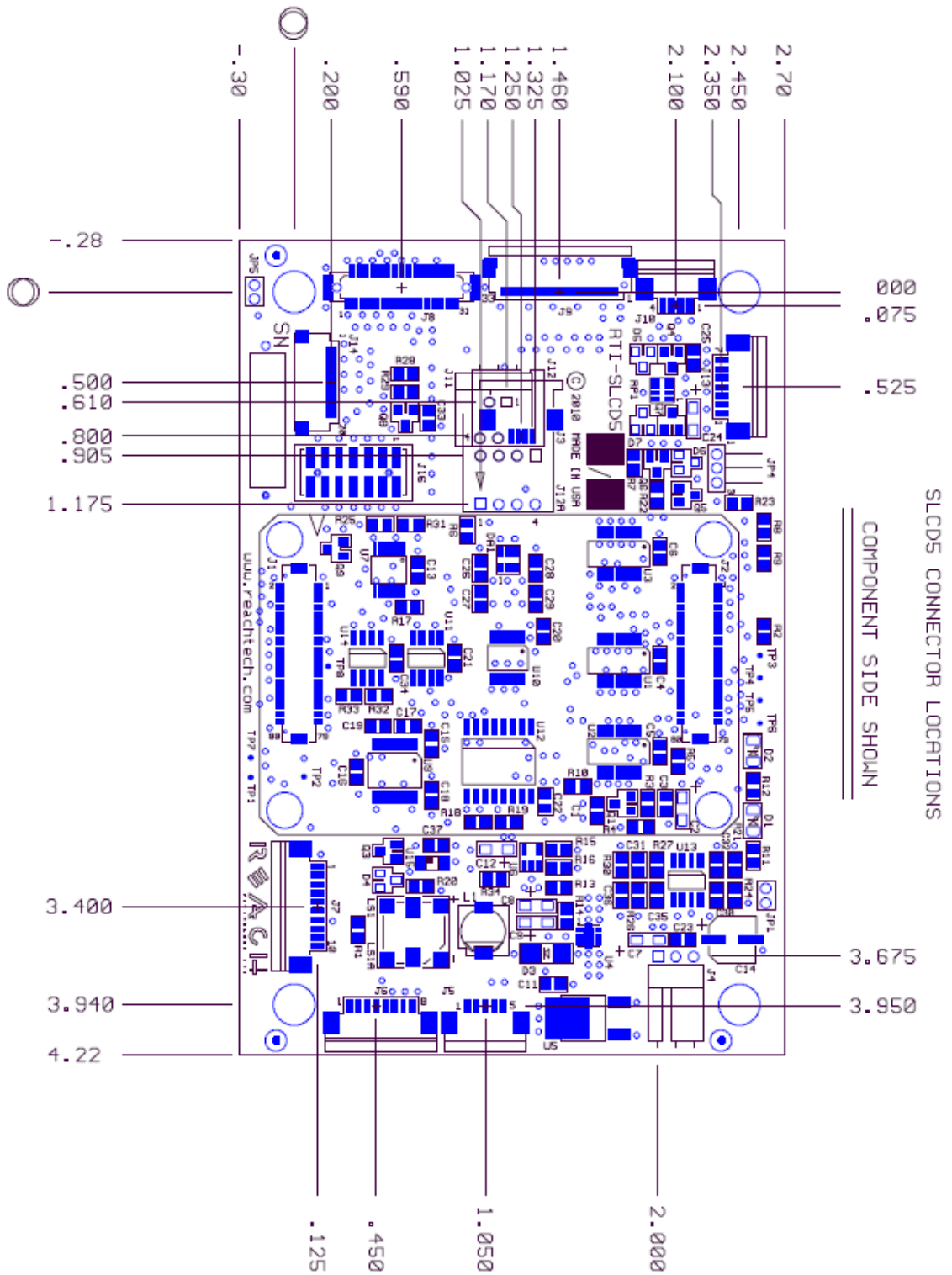


Figure 2: SLCD5 dimensions and connector locations (inches)

## 2. Connectors and Jumpers

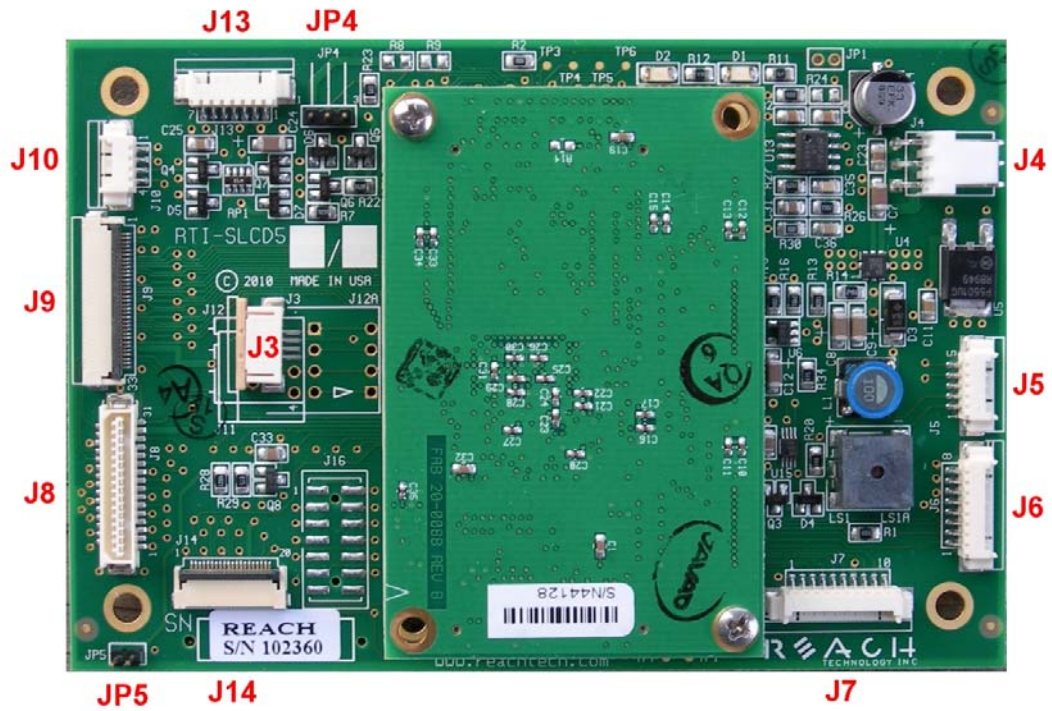


Figure 3: Connectors and Jumpers (SLCD5+ Rev D shown)



## 2.1. Overview

The SLCD5 requires the following major connections for operation: **Power, Communications, Panel, and Inverter.** The following table summarizes these connections and options.

Purpose	Use	Notes
<b>POWER</b>	J6	J6 has both power and communications signals.
	- or - J4	Alternate power connector. Can also be used as power out if J6 is providing power. Required if inverter (see below) takes more than 1A current.
<b>COMMUNICATIONS</b>	J6	COM0 port (typically RS-232).
	J7	COM1 port (RS-232, RS485/RS422 or CMOS 3.3V).
<b>PANEL</b>	J8	VGA panels, SVGA panels using DF9-31 or DF9-41 connector.
	J9	All other panels.
<b>INVERTER</b>	J13	High power inverters, two types of dimming supported.
	J10	Compatible with SLCD inverter cabling.

## 2.2. J6 - Power and Communication COM0 (RS232 Mode)

**J6 8 Pin** Molex 53261-0890 for Power and Main Communications

Pin	RS232 Mode – R1 installed on board
1	Do not connect
2	Do not connect
3	RS232 input (ESD protected 15KV)*
4	RS232 output (ESD protected 15KV)*
5	Backlight power input, feeds directly to J10, J13, max. 1A (Note 2, 4)
6	Ground
7	5 - 12V main power Input (Note 1, 3, 4)
8	Ground

*Note 1: The board can be powered either through J6 OR J4. If it is powered through J4, pins J6-5 and J6-7 do not need to be connected.*

*Note 2: Rev B board: if pin 5 is powered with 5V for a 5V backlight driver / inverter, resistor R8 (zero ohms) needs to be installed for full speaker volume.*

*Note 3: Rev A board: if pin 7 is powered with 5V, Jumper JP2 needs to be installed for full speaker volume.*

*Note 4: Consult appropriate Reach Display Module Data Sheet; some modules work only with 5V, some only with 12V.*

*\*ESD protection via ICL3222E ESD protected RS-232 transceiver or equivalent.*

Typical connection from a PC (RS-232) is as follows:

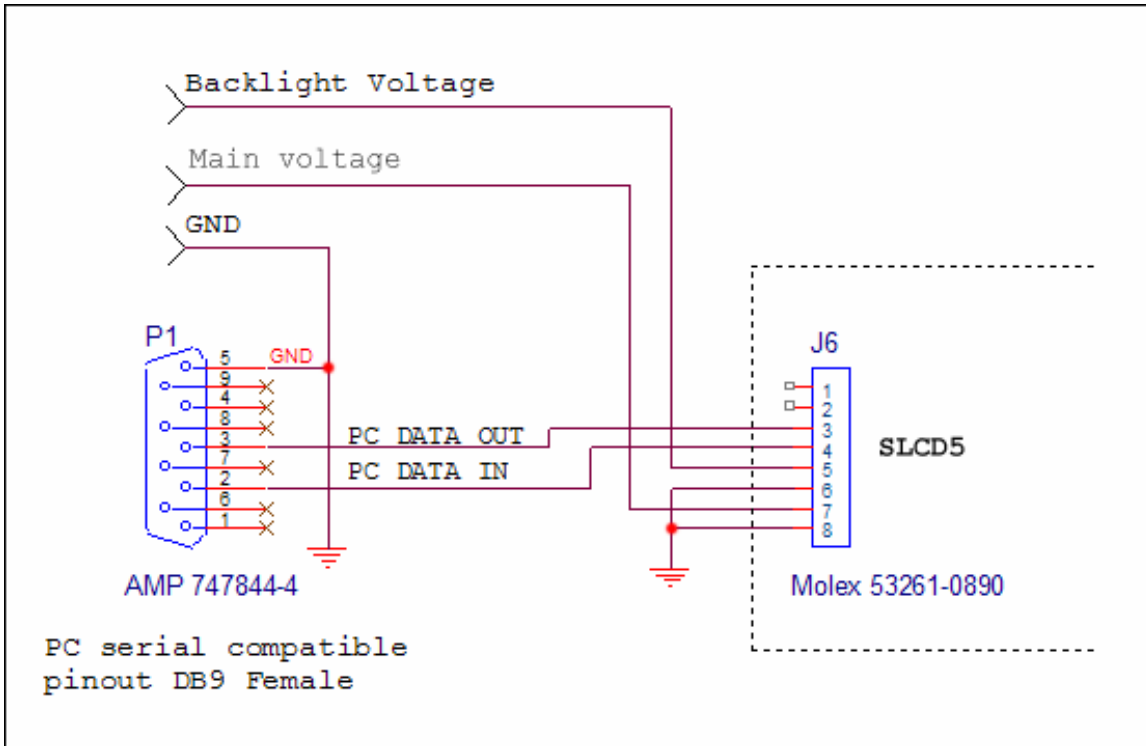


Figure 4: Typical (RS-232) Connection from a PC

### 2.3. J6 - Power and Communication COM0 (3.3V CMOS Mode)

To use this mode, REMOVE resistor R1 from the board to disconnect the RS232 receiver from pin 2.

**J6 8 Pin** Molex 53261-0890 for Power and Main Communications

Pin	CMOS I/F Mode – R1 removed from board
1	UART TxD (out from SLCD5)
2	UART RxD (in to SLCD5)
3	Do not connect
4	Do not connect
5	Backlight power input (Typically. 12V) input, max. 1A (Note 1, 2, 4)
6	Ground
7	5 - 12V main power Input (Note 1, 3, 4)
8	Ground

Notes: See Section 2.2 above

## 2.4. J7 - Communication COM1 and Reset

On the standard SLCD5 controller, J7 supports either RS-232 levels OR 3.3V CMOS levels as a serial port interface. This is selected via a jumper on the connector. It can also support RS485/RS422 as a manufacturing option.

### J7 10 Pin Molex 53261-1090 for RS232 COM1 Communications

Pin	Connections for RS232 Mode
1	3.4V output (for powering external transceiver)
2	Not used
3	No connect
4	Connect to pin 9
5	RS-232 out (ESD protected 15KV*)
6	RS-232 in (ESD protected 15KV*)
7	Ground
8	RESET- input (active low with on-board pullup)
9	Connect to pin 4 to select RS232 operation
10	Not used

\*ESD protection via ICL3222E ESD protected RS-232 transceiver or equivalent.

### Example J7 Cable for connecting to PC serial port via DB9-Female

J7 Pin	DB9 (Female) pin	Signal
5	2	RS-232 from SLCD5 to PC
6	3	RS-232 from PC to SLCD5
7	5	Ground
4	n/a	Connect to pin 9
9	n/a	Connect to pin 4

**J7 10 Pin** Molex 53261-1090 for **3.3V CMOS** COM1 Communications

Pin	Connections for 3.3V CMOS Mode
1	3.4V output (for powering external transceiver)
2	RTS (can be used to enable external driver)
3	No connect
4	RxD- (standard UART receive data input), 3.3V CMOS levels
5	Do not use
6	Do not use
7	Ground
8	RESET input SLCD5 PLUS: active high with on-board SLCD5: active low with on-board pullup
9	Do not use
10	*TxD- (standard UART transmit data output)

**NOTE: THIS ONLY AVAILABLE AS A MANUFACTURING SUFF OPTION (U14 SN65HVD32D, R25=0 OHM)**

**J7 10 Pin** Molex 53261-1090 for **RS485/RS422** COM1 Communications

Pin	Connections for RS485 / RS422 Mode
1	3.4V output (for powering external transceiver)
2	Inverting driver output*
3	Inverting input**
4	Non-inverting input**
5	Do not use
6	Do not use
7	Ground
8	RESET- input (active low with on-board pullup)
9	Do not use
10	Non-inverting driver output*

\*On-board optional output termination resistor is R33.

\*\* On-board optional input termination resistor is R32.

## 2.5. J4 - Power

***(Used if separate power input is desired; can also be used as pass-through output for power supplied via J6)***

**J4 3 Pin** Molex 22-05-3031 0.1" polarized or equivalent

Pin	Signal
1	Backlight and Speaker power. Connects directly to J6-5. J10-1, J13-1, J13-2. Typically 5V or 12V (Note 1, 4)
2	Main power, connects directly to J6-7. Typically 5V or 12V (Note 1, 4)
3	Ground (common to main and inverter power)

*Note 1: Notes for J6 power levels should be applied.*

## 2.6. J3 - 4 Wire Touch (hardware version dependent)

**J3 4 Pin** Molex 52271-0429 or equivalent bottom contact 1mm pitch bottom contact Zero-Insertion-Force

Pin	Signal*
1	X Right / X+
2	Y Down / Y+
3	X Left / X-
4	Y Up / Y-

*Connector compatible with Fujitsu N010-0554-T511 8.4" 4-wire touch or similar.*

*All signals are ESD protected via California Micro Devices PACDN044Y5.*

## 2.7. J11 - 4 Wire Touch (hardware version dependent)

**J11 4 Pin** Molex 39-51-3043 or equivalent 1.25mm pitch top contact Zero-Insertion-Force Connector

Pin	Signal
1	Y Up
2	X Left
3	Y Down
4	X Right

*For Kyocera touch.*

## 2.8. J12 - 4 Wire Touch (hardware version dependent)

**J12 4 Pin** Molex 22-05-3041 or equivalent 0.1" pitch 0.025" square post right angle friction latch Connector.

Pin	Signal
1	X Right
2	Y Up
3	X Left
4	Y Down

*Compatible with Kyocera top contact 1.25 mm pitch touch panels.*

## 2.9. J9 - 33 pin 0.5mm Flat Flex LCD Connector

**J9 33 Pin** Omron XF2M-3315-1 0.5mm pitch Zero-Insertion-Force Connector

Pin	Signal	Pin	Signal
1	GND	18	Green 5
2	LCD Clock	19	GND
3	LCD Line Pulse (HSYNC)	20	Blue 0 (= Blue 5)
4	LCD Frame Pulse (VSYNC)	21	Blue 1
5	GND	22	Blue 2
6	Red 0 (= Red 5)	23	Blue 3
7	Red 1	24	Blue 4
8	Red 2	25	Blue 5
9	Red 3	26	GND
10	Red 4	27	LCD DE (Data Enable)
11	Red 5	28	LCD VCC
12	GND	29	LCD VCC
13	Green 0	30	R/L *
14	Green 1	31	U/D *
15	Green 2	32	V_Q**
16	Green 3	33	GND
17	Green 4		

\* These signals can be set via SLCD5 *"\*orient"* command.

\*\* This signal is typically set by the board's firmware and is panel-dependent.

*Note: The SLCD5 supports 565 16-bit color, so the least significant color bits are set to the most significant bit. This preserves dynamic range at the expense of middle level steps.*

## 2.10. J8 - 31 pin 1mm LCD Connector

### J8 31 Pin Hirose DF9-31P Connector

Pin	Signal	Pin	Signal
1	GND	17	Green 4
2	LCD Clock	18	Green 5
3	LCD Line Pulse (HSYNC)	19	GND
4	LCD Frame Pulse (VSYNC)	20	Blue 0 (= Blue 5)
5	GND	21	Blue 1
6	Red 0 (= Red 5)	22	Blue 2
7	Red 1	23	Blue 3
8	Red 2	24	Blue 4
9	Red 3	25	Blue 5
10	Red 4	26	GND
11	Red 5	27	LCD DE (Data Enable)
12	GND	28	LCD VCC
13	Green 0	29	LCD VCC
14	Green 1	30	R/L *
15	Green 2	31	U/D *
16	Green 3		

\* These signals can be set via SLCD5 *"\*orient"* command



## 2.11. J10 - Backlight / Inverter Control

**J10 4 Pin** Molex 53261-0471

Pin	Signal
1	Backlight power (connects directly to J6-5. J4-1, J13-1, J13-2)
2	Ground
3	Backlight on/off control
4	Backlight brightness control (analog voltage)

*This connector is used to power and control the panel backlight. The active sense of the on/off control (active high or low) is set in the firmware. The sense and range of the brightness voltage output is also set in the firmware.*

## 2.12. J13 - Backlight / Inverter Control

**J13 7 Pin** Molex 53261-0771

Pin	Signal
1	Backlight power (connects directly to J6-5. J4-1, J10-1)
2	Backlight power (connects directly to J6-5. J4-1, J10-1)
3	Ground
4	Ground
5	Backlight on/off control
6	Backlight brightness control (analog voltage)
7	ERG Backlight on/off + PWM (high = on)

*This connector is used to power and control the panel backlight. The active sense of the on/off control (active high or low) is set by the in. The sense and range of the brightness voltage output is also set in the firmware.*

### 2.13. J5 - External Audio Output (not typically used)

J5 5 Pin Molex 53261-0571 or equivalent

Pin	Signal
1	Audio out (-)
2	Audio out (+)
3	N/C
4	Speaker out 1
5	Speaker out 2

*The Audio out provides a square wave at audio frequency and is the same signal as connected to the on-board beeper. The Speaker output is designed to connect to an 8 ohm speaker, typically 0.5W - 1W size. The speaker vs. beeper selection is made by firmware command (see \*spkr command).*

### 2.14. J15 SD card connector

The SD card connector on the back of the SLCD5 is compatible with standard SD cards.

J15 ALPS P/N SCDA\$A0301

Pin	Signal
1	SD_DAT3
2	SD_CMD
3	GND
4	V_CARD (approx 3.15V)
5	SD_CLK
6	GND
7	SD_DAT0
8	SD_DAT1
9	SD_DAT2
10	GND (Tab)
11	GND (Tab)
12	GND (Tab)
13	n/c
14	SD_DETECT
15	SD_WP
16	GND

### 2.15. J14 - for external SD card

The SLCD5 has an SD card slot on the back of the board. The SD card signals are also routed to J14 so that the SD socket can be placed on a small board and located in a more convenient location.

**J14 20 Pin Omron XF2M-2015-1 0.5mm pitch Zero-Insertion-Force Connector**

Pin	Signal	Pin	Signal
1	GND	11	GND
2	SD_DAT2	12	SD_DAT1
3	GND	13	GND
4	SD_DAT3	14	n/c
5	GND	15	V_CARD (approx 3.15V)
6	SD_CMD	16	n/c
7	GND	17	n/c
8	SD_CLK	18	SD_LED* for activity indication
9	GND	19	SD_DETECT
10	SD_DAT0	20	SD_WP

*\* High true LED drive, max 3.4V, must provide external series resistor.*

### 2.16. JP5 - SD card power

When installed, JP5 forces the SD card power to be on when power is applied. This is for compatibility with older firmware that did not control the SD card power, and previous hardware versions.

### 2.17. JP2 - deprecated for Rev B and above

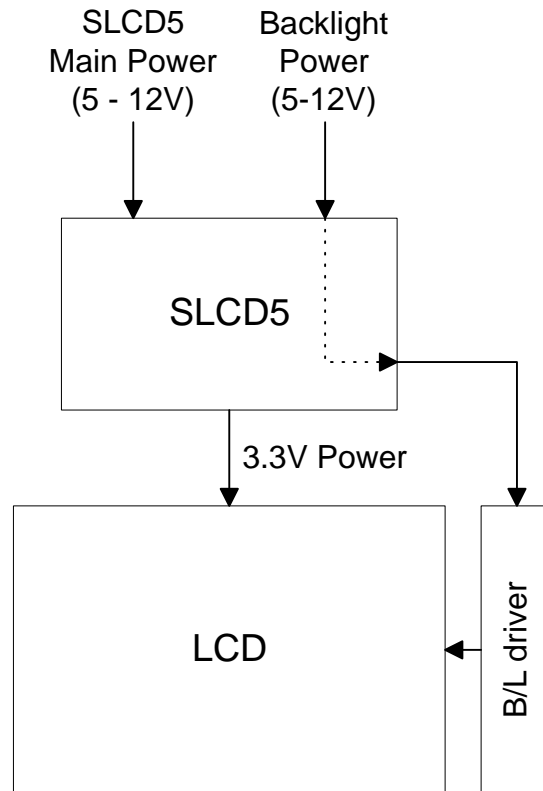
The SLCD5 is typically powered by 12VDC, and generates 5V for its on-board speaker and inverter interface via a linear regulator. If the board is powered with 5V (JP6-7 or J4-2) the linear regulator needs to be bypassed for maximum beeper volume and inverter control. Insert JP2 in this case.

**CAUTION: WITH JP2 INSTALLED, APPLYING 12VDC to JP6-7 / J4-2 WILL DAMAGE THE BEEPER AND POSSIBLY THE ATTACHED INVERTER.**

### 3. Configuration Guide

#### 3.1. Power Connections

The diagram below illustrates the power flow in an SLCD5 configuration.



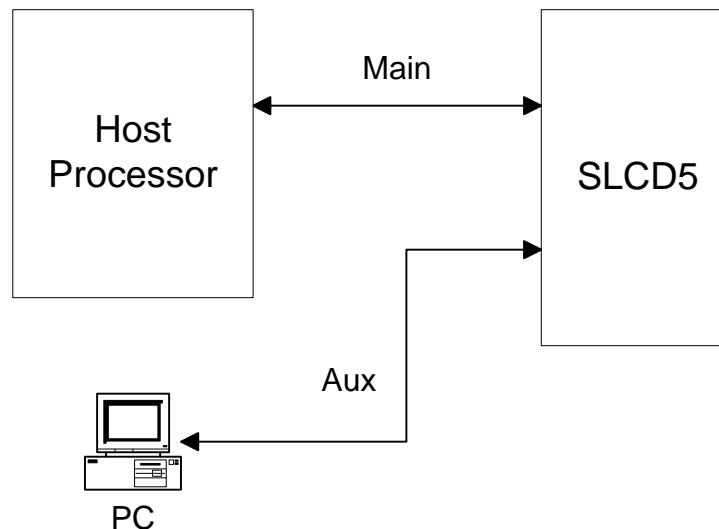
#### 3.2. Serial

The SLCD5 can use either RS-232 levels or CMOS logic levels for serial communication. As shipped in the standard configuration, the COM0 serial port is configured for RS-232 and the COM1 serial port is either RS-232 or CMOS cable jumper selectable. The COM0 can be set for CMOS levels by removing resistor R1. This can be provided as a factory option when production board quantities are ordered - contact the factory for ordering information. COM1 has the option to be RS485/RS422 as a manufacturing option - consult the factory.

In the SLCD5 development kits, the COM0 and COM1 ports are mapped to "Main" and "Aux" ports. The Main port is connected to the embedded processor and controls the display. The Aux port is typically connected to a PC and is used to update the bitmaps and macros stored on the board.

By default, the serial communication protocol is 115200 baud, 8 data bits, no parity, with 1 stop bit, and software (XON/XOFF) flow control. The baud rate can be changed by using a power-on macro (see Appendix E), or by using the "bauds" command.

The following diagram illustrates this communications setup.



### 3.3. TFT Hardware Panel Orientation

Some TFT LCD panels have orientation signals such as "UP / DOWN", and "LEFT / RIGHT" that allow the display orientation to be flipped in the horizontal and / or vertical orientation. The SLCD5 provides these signals on the LCD connector, and they can be set under software control using the orient command / config.ini setting.

### 3.4. System Configuration

The SLCD5 maintains touch calibration and option parameters in non-volatile memory (EEPROM). These parameters include LCD timing, inverter interface details, startup command, and so on. These can be set via the serial command interface, but in a production environment, it is recommended to set them via a file on an SD card. In this way, the SLCD5 can be configured for production simply by inserting the appropriate SD card and cycling power. When the controller starts, it reads the configuration file and stores the specified values into EEPROM. This is done only once, as typically the SD card is not shipped installed in the unit.

***Warning: the EEPROM has a limited life of 1,000,000 writes. The CONFIG.INI file causes the EEPROM to be written, and so it should not be present on an SD card installed in a production system.***

The config.ini file must be in the root directory of the SD card. It is a plain text file, with one or more lines in the following format:

```
<variable name> = <value>
```

Comments are indicated by a # character at the beginning of a line. Empty lines are allowed and ignored.

Note that unlike a PC Windows system, the config.ini file only needs to be present once on power-on to set the board optional parameters. It should not be present on an SD card that is present on power-on since it will re-write the EEPROM each time the board is powered on.

CONFIG.INI Example:

```
#---- start of file-----  
#this is an example  
  
#display active config.ini statements on COM0 port at  
power-on  
verbose = 1  
  
#set power on macro to 1  
PONMAC = 1  
#PONMAC = 2 - example of how to comment out an  
alternate value  
  
#set test string response to *config command  
config = "test config file"  
  
#---- end of file ----
```

CONFIG.INI options: (\* indicates default)

Variable Name	Values	Action
verbose	0* or 1	If 1, the valid config variables are displayed on the console port at power-on. This helps in debugging or validating a config,ini file.
mainPort	0* or 1	Sets the COM port that is active on power-on. Note that boot and startup messages will always be displayed on COM0.
orient	0..3	<a href="#">Equivalent to *orient command.</a>
invEnaHiTrue	0 or 1	<a href="#">Same as *invEnaHiTrue command.</a>
pwmlsEnable	0 or 1	<a href="#">Same as *pwmlsEnable command</a>
maxBrite	0..255	<a href="#">Same as *maxBrite command</a>
minBrite	0..255	<a href="#">Same as *minBrite command</a>
brite	0..255	<a href="#">Same as xbb command</a>
touchSwap	0 or 1	<a href="#">Same as *touchSwap command</a> This is no longer needed due to 5 point calibration.
tsamp	0-20	Default value is panel-dependent <a href="#">See *touchParm command.</a>
tspan	0-20	Default value is panel-dependent <a href="#">See *touchParm command.</a>
touchMode	0*..7	bit 1 => lockout bit 2 => validate touch pressure bit 3 => touch wander acts as release <a href="#">See *touchMode command.</a>
tplo	0-65535	Default value is panel-dependent <a href="#">See *touchMode command.</a>
tphi	0-65535	Default value is panel-dependent <a href="#">See *touchMode command.</a>
tcTimeout	0-99	<a href="#">Same as *tcTimeout command</a>
auxEsc		<a href="#">Same as *auxEsc command.</a>
beepFreq	0..3000	<a href="#">Same as bf command</a>
beepVol	0..255	<a href="#">Same as bv command</a>
debounce	50 - 200	<a href="#">Same as *debounce command</a>
typematicDelay		<a href="#">Same as first argument of typematic command</a>
typematicRepeat		<a href="#">Same as second argument of typematic command</a>
PONMAC		<a href="#">Same as *PONMAC command</a>
SPL		<a href="#">Same as *SPL command</a>
hfp		Set LCD horizontal "front porch" timing in clocks

hpw		Set LCD horizontal sync width timing in clocks
hbp		Set LCD horizontal "back porch" timing in clocks
vfp		Set LCD vertical "front porch" timing in lines
vpw		Set LCD vertical sync width timing in lines
vbp		Set LCD vertical "back porch" timing in lines
config	"text string"	<a href="#">Sets response for *config command</a>
LCDshiftclock	0 or 1	0 = Active on positive edge of LCD Shift Clock. 1 = Active on negative edge of LCD Shift Clock.
baud0 baud1	1200 4800 9600 19200 38400 57600 115200 230400	Set COM0 baud rate Set COM1 baud rate The new rate becomes active at the next power cycle.
externalSpeaker	0* or 1	0 = External Speaker OFF, on-board Beeper ON 1 = External Speaker ON, on-board Beeper OFF
autoCal	0* or 1	0 = autoCal disabled 1 = autoCal enabled: if the touch screen is touched while the unit powers on, it will perform a touch calibration and then continue normal startup.
macdebug	0* or 1	Same as <a href="#">*macdebug</a> command. Use to debug power on macro.



## 4. System Overview

### 4.1. General SLCD5 Controller Information

The SCLD5 acts as a "smart terminal" and is meant to be connected to a host processor that implements the desired Graphical User Interface (GUI) by issuing commands to the SLCD5 and processing button press responses from the SLCD5. In this manual, the term "host" is used to describe the device connected to the SLCD5.

For use in systems where the connection between the SLCD5 and the host needs to be checked for validity on a regular basis, a Communications Watchdog feature can be enabled to cause special messages to be sent to the "host" if the SLCD5 does not receive input from the "host" for an interval determined by the host (see [COMMUNICATIONS WATCHDOG](#)).

### 4.2. Compatibility with SLCD, SLCD6, SLCD43, SLCD+ controllers

The SCLD5 is generally software-compatible with the smaller screen SLCD family controller. The significant differences are:

- The built-in fonts are different changed. The SLCD fonts were specifically designed for a small screen. The SLCD5 fonts are Microsoft Windows compatible to ease graphic bitmap development. The SLCD font names are still supported so that macros will not fail, but the fonts are different.
- The latching button response has changed; there is now a space between the button index and the state:
- Response format:       s<index> <state>  
Example:               s128 1
- The SLCD5 is much faster; this may impact host timing.

### 4.3. Bitmaps and macros

To implement a GUI, a set of graphic images are needed for backgrounds, logos, buttons, switches, and so forth. These are created on a PC as bitmaps (.bmp files) in 24-bit color. Photoshop and Gimp can be used as bitmap editors. The bitmaps are compressed into a binary file which is then either downloaded directly into the SLCD5 flash memory via the serial port, or stored on an SD card that is then inserted into the SLCD5.

This binary file can also contain macros and extra fonts. A macro is a set of commands that can be invoked with a single command, and this is detailed in [Appendix E](#).

In a development environment, the SD card is a convenient way to load bitmaps stored macros. Once development is complete, the SD card is also used to update the on-board flash memory. Once updated, an SD card is no longer needed to be shipped with the controller.

See [Appendix D](#) regarding the *BMPload* program that is provided to store bitmaps and other data into the SLCD5.

#### 4.4. Overview - SLCD5 Evaluation Kits

The SCLD5 is available in an evaluation kit form. It comes pre-loaded with bitmaps and macros that implement a demo if the unit is powered on with an SD card installed that contains a file "rundemo.ini". This file is a standard text file that contains the line:

- demomac = <macro number>

example:

- demomac = 1

Where the <macro number> is the macro to run at power-on, typically 1. The SLCD5 evaluation kit comes with a two-port DB9 interface board that makes it easier to develop applications. One port connects to the host processor and the other connects to a PC which is used to interactively develop screen content and test commands and also to download bitmap images.

#### 4.5. Getting Started

The SLCD5 kit as shipped contains a demo that allows you to verify its functionality and see various interface possibilities. Just plug the supplied power supply into the barrel connector on the triangular PowerCom 4 board. The display should light up and lead you through various touch-activated screens. If for some reason the touch is not working, you may need to run the "touch calibrate" command.

Note that in a kit, the demo is preloaded into the unit's flash memory, and includes both bitmap files and a macro file. To best learn how the SLCD5 board and this kit works, start with simple commands using the serial interface and leave the creation and use of macros for later. [Appendix G](#) provides a brief tutorial.

#### 4.6. **Connecting the kit to a PC**

The kit should be connected to a PC so that the serial command interface can be experimented with. This is a preliminary step before the unit is connected to the embedded system that will control the kit.

As shipped, the serial port is set to 115,200 baud, 8-bit, 1 start, 1 stop, no parity. There are two DB9 connectors on the triangular "PowerCom 4" board. Connect the PC using a straight through serial cable to the DB9 marked "MAIN" (P1). A USB-to-serial adapter cable can also be used and plugged directly into this connector. Reach recommends the FTDI series of serial adapters, see [www.ftdichip.com](http://www.ftdichip.com). These can be purchased from [www.digikey.com](http://www.digikey.com). **Warning: the Belkin USB-serial adapter has software compatibility issues and is not recommended.**

Once connected, use a terminal emulator such as ReaTerm (<http://realterm.sourceforge.net/>) to send and receive commands from the unit. **Appendix G** provides a brief tutorial.

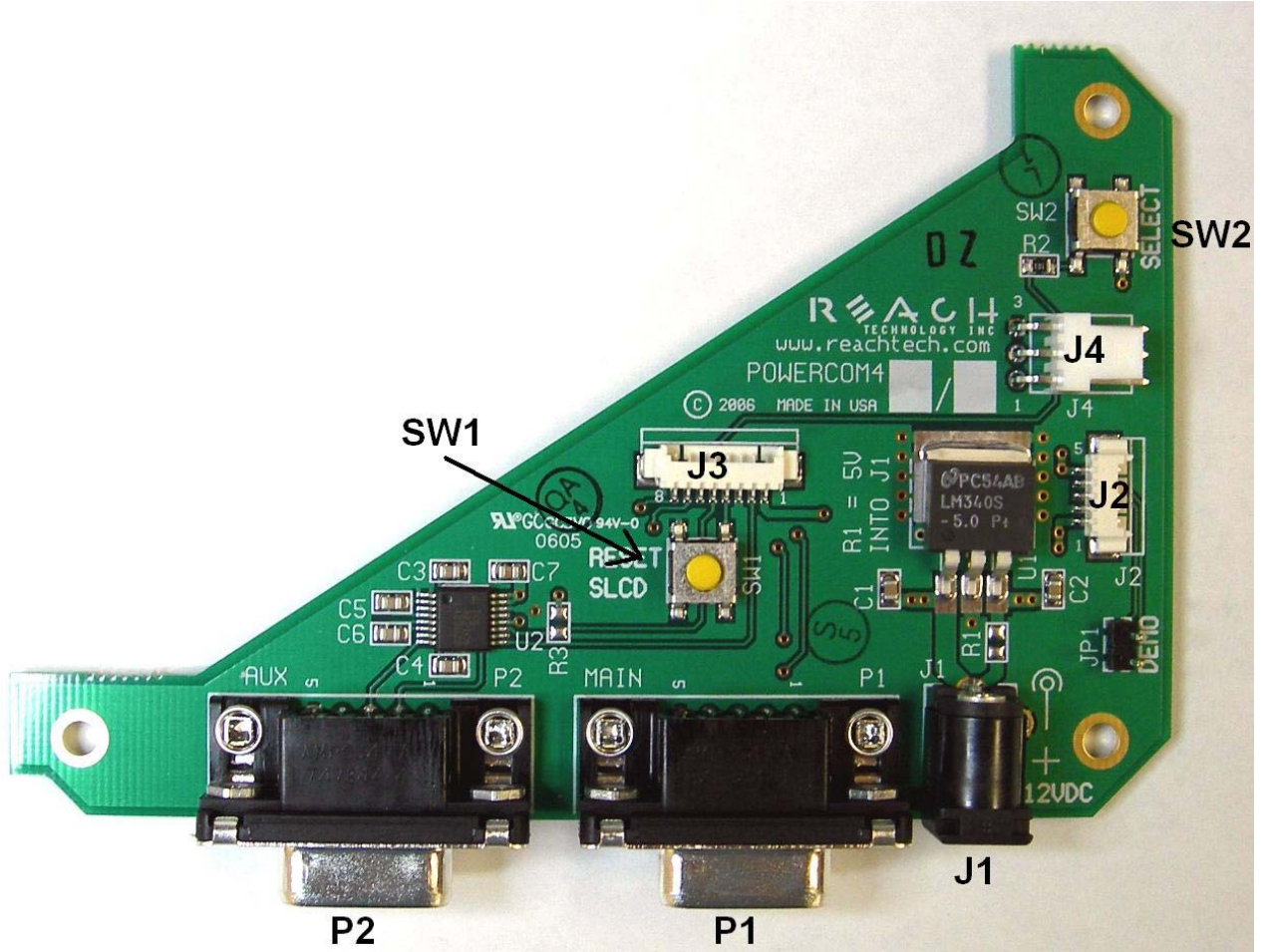
#### 4.7. **Switching between PC and embedded controller**

The SLCD5 firmware has the concept of a "Main" control port and an "Aux" port. The main port is used to send commands to the unit. During development, it is useful to be able to switch which physical port is considered Main. This is useful so that the PC can take control and download new images, or be used to test a command via a serial terminal emulator.

Switching happens when the aux port receives three <return> characters in a row. After that, it switches to become the main port. The BMPload program uses this method to take control and download new bitmaps. It also uses the \*prevCons command to switch back so that the embedded host has control.

The embedded host code should be written such that when it starts and tries to connect to the SLCD5, it issues a <return>, waits for the SLCD5 prompt (2 characters, '>' followed by <return>), and then tries again. Doing this, it will end up sending the three <return> characters in a row and auto switch the port it is connected to the main port.

NOTE: only 1 port is in control at any given time.



**Figure 5: PowerCom4 board supplied with Evaluation Kit**

## 4.8. PowerCom4 Schematic

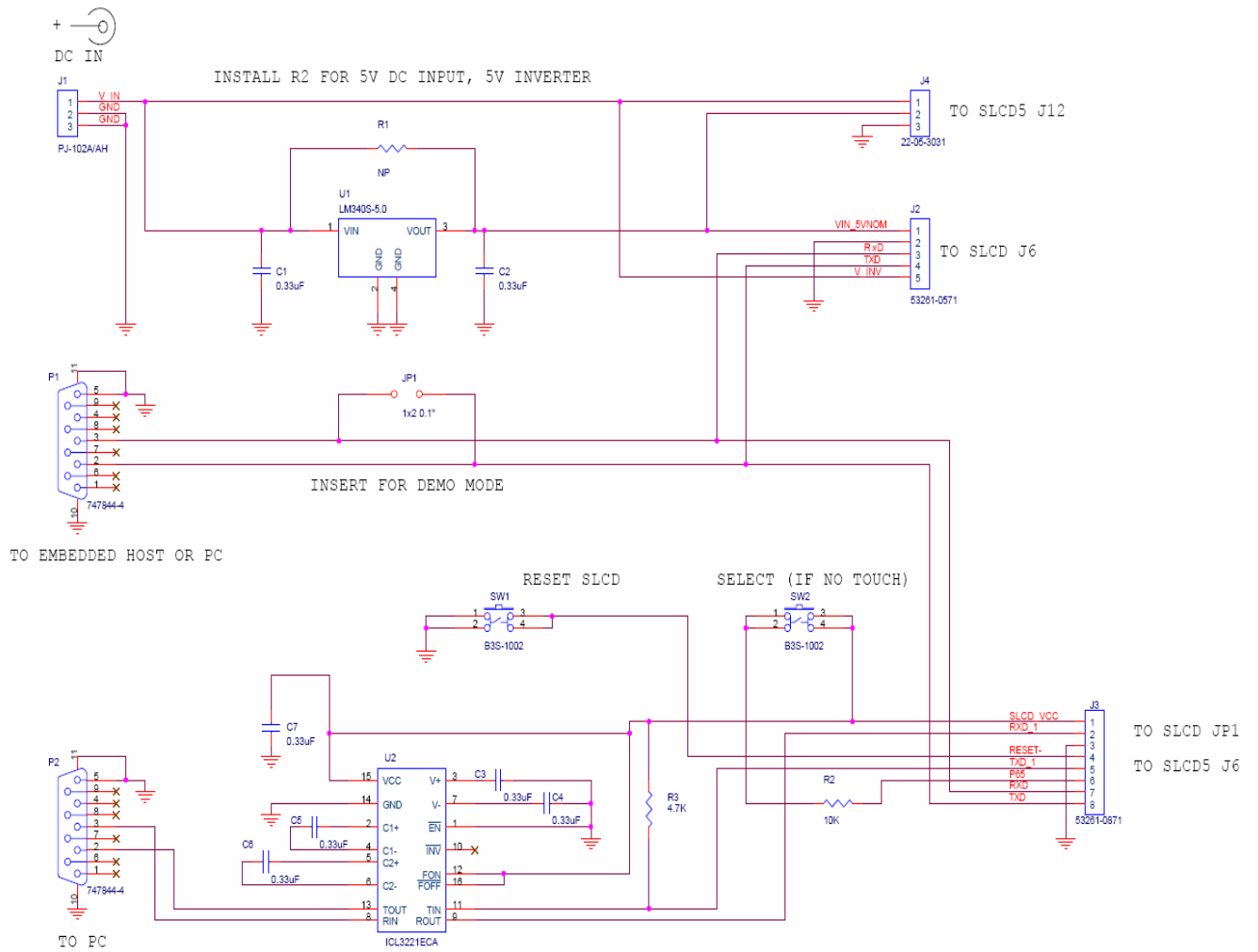


Figure 6: PowerCom4 Schematic

## 4.9. PowerCom4 Operational Notes

### Operational Notes

1. The unit default baud rate is 115200. The unit does not echo characters (for communications efficiency), so you must select "echo characters locally" or set "half duplex" in your PC communications program. Also, all return strings are terminated by a <return> only, so you need to specify "add line feed to line return" as well.
2. The internal demo starts with an optional touch calibration. In order for the touch screen to work reliably, ensure the LCD frame is grounded to the SLCD5 mounting holes.
3. The demo requires a certain set of bitmaps to be loaded. If these are not present, it will not run correctly. Copies of these are provided in the "BMPs and Macros" directory on the CD provided. Copy the SLCDDEMO.BIN file to the SD card and install to run the demo.
4. The SW1 "RESET" button on the PowerCom4 board resets the SLCD5 processor and performs the equivalent of a power-on reset.
5. The SW2 "SELECT" button on the PowerCom 4 board is intended for use with kits that don't have a touch screen, and is not implemented on a standard kit.
6. Jumper JP1 marked "DEMO" should not be used with the SLCD5 controller.
7. The J1 barrel connector is the external power supply connector for the development kit. For kits that use 12V input, it is 2.1mm, center pin positive. For kits with 5V input, it is 2.5mm, center pin positive. This prevents a 12V supply from being used with a 5V kit.
8. Connector J2 provides the communications path for the P1 "MAIN" RS232 serial port. It connects to J6 of the SLCD5 controller. Connector J2 also provides 5VDC power to the SLCD5 controller.
9. Connector J3 of the PowerCom4 board is the communications path for the P2 "AUX" RS232 serial port. It connects to JP1 of the SLCD5 controller. This provides the path for the "RESET", and "SELECT" signal buttons. As well as the communications path for downloading of bitmaps and macros to the SLCD5 controller.
10. Connector J4 is reserved for future use.

## 4.10. Communications Interface

### General

- Default communication is at a baud rate of 115200 with no parity, software (XON/XOFF) flow control, 8 bits of data, and 1 stop bit. The baud rate can be set to a different initial value on power-on by using the bauds or POWER-ON MACRO feature.
- Only 1 com port may be in control of the SLCD5 at any time; control may be switched to the other port under software control. (see [Switching between PC and embedded controller](#)).
- ASCII commands consist of a command (one or more ASCII characters) followed by the data associated with that command, followed by a carriage return. In this manual, the return character (value 0x0D, decimal 13) is signified by <return>.
- Screen pixel values start at the upper left-hand corner. This is point x=0, y=0. The lower right corner is point x=(width-1), y=(height-1); for VGA, width=640 and height =480.
- The maximum length of any command including the termination character is 127 characters.

## 4.11. SLCD5 Input Buffer Processing

### Input Buffer

The SLCD5 has a nominal 512 byte input circular buffer. As commands are received, they are queued in the buffer and executed first come first served. After a command has been processed, the SLCD5 issues a "prompt" character followed by a <return> indicating the success or failure of the command. The '>' prompt indicates success and the '!' prompt indicates failure. Failure can be due to either a syntax error or an out-of-bounds parameter. Depending on how long a command takes to execute, one or more commands may be stacked in the input buffer. The SLCD5 will issue a prompt for each command after it executes. These prompts may be issued while the host is sending a command to the SLCD5 (full duplex operation).

The purpose of the circular buffer is to provide overlapped command issue and execution with full duplex communication. If this is not needed, the host can wait for the prompt before sending another command.

The SLCD5 controller issues a prompt when it has finished processing a command. This includes the null command which is just a <return>.

There is no special "power-on" prompt supplied when the unit first powers on. To detect that the board is available for commands, the host should send a null command (single <return> character) and wait at least 10ms for a success prompt back. Alternatively, the

POWER-ON MACRO command / feature can be used together with the OUTPUT command to send a unique message indicating that the unit is up and running.

## Flow Control

The SLCD5 implements software flow control using the XON (decimal 17) and XOFF (decimal 19) characters. When the circular buffer is approximately  $\frac{3}{4}$  full, an XOFF is issued to the host. An XON is then issued when the buffer is approximately  $\frac{1}{4}$  full. If the host cannot or does not want to accommodate software flow control, the host can make sure that no more than 2 commands are outstanding at any time. Given that the maximum length of any command is 127 bytes, this guarantees that the host will not be sent an XOFF character.

## Buffer Limit Discussion

The input buffer can become full and unable to accept more data in two scenarios, both of which should never happen in normal operation. This discussion is presented because buffer overflow issues have presented security and reliability problems in PC and internet devices. The two scenarios are as follows. In both cases, the buffer limit event happens when the buffer is full and one more character is received and has to be thrown away.

**Scenario #1:** The host sends data that a) does not conform to the command specification, and b) keeps doing so until the buffer size limit is reached, and c) ignores the XOFF request from the SLCD5. ASCII commands are limited to a total of 127 characters including the <return>. Input buffer limit will occur when enough data is sent without a <return> to fill the buffer. This indicates a flaw in the host protocol or a hardware failure (for example, the communication line is chattering).

**Scenario #2:** The host sends valid commands that take a long time to execute and ignores the XOFF request from the SLCD5. The limit event can occur when the buffer is full of unexecuted commands.

In both of the previous cases, when the SLCD5 detects a buffer limit it does the following:

- Discards the received character that caused the limit event, and resets (flushes) the entire input buffer. This is done in an attempt to make the error obvious to the GUI user. If a buffer overflow occurs it is a serious system error.
- Sends an overflow prompt to the host. The overflow prompt is '^' <return>. That is, shift-6 or caret followed by a return.
- Sends an XON character to the host (matches the XOFF that was previously sent)



## Prompt Summary

The SLCD5 can issue the following prompts. These are in addition to any result of a command or button press event.

- `>'<return> Indicates that a command has been executed successfully.
- `!'<return> Indicates that the command had a syntax or parameter error.
- `^'<return> Indicates that an input buffer full event occurred.
- `?'<return> Indicates that a transmission line error occurred. This includes parity, framing, and receive overrun errors.

### 4.12. Touch interface

The SLCD5 contains a touch controller that interfaces to a four wire resistive touchscreen. Touch sensitive areas of the display are defined as either "hotspots" or "buttons". When either of these is pressed or released, the SLCD5 can either notify the host directly or execute a "macro", or both. A macro is a predefined sequence of SLCD5 commands.

#### Hotspot

A hotspot is an area of the display that is touch sensitive. There are two types of hotspots – visible and invisible. A visible hotspot is the standard type and when touched, the display area of the hotspot is color inverted (technically XOR'd with the foreground color) to provide a visual indication that a hotspot has been activated. An invisible hotspot does not provide any visual indication when touched.

The invisible hotspot is useful where a touch control is used to switch display screens. If a visible hotspot is used, and the host redraws the screen when the hotspot is pressed, the hotspot area can become inverted when the user removes their finger from the screen.

#### Button

A button is a touch sensitive area that has two bitmaps associated with it. These bitmaps correspond to the two states of the button – 1) normal /not pressed and 2) active / pressed. This allows a button to look like any GUI object including pushbuttons, toggle switches, radio buttons, check boxes, and so forth.

There are two major types of buttons: normal (momentary) and latching. A momentary button changes visual state only when pressed. This is like a momentary pushbutton or a keyboard key. A latching button is like a checkbox – press and release it once and the checkbox is filled, press and release again to clear it.

## **Host Notification**

When a touch sensitive area is pressed or released, the SLCD5 can either notify the host, execute a macro or both. See the `BUTTON DEFINE` and `TOUCH MACRO ASSIGN` commands for details.

### **4.13. *Host input processing***

When integrated into a host environment, the SLCD5 sends prompts, touch activity notifications, and user-defined text to the host it is connected to. In general, all SLCD5 messages are terminated with a `<return>`.

There can be no guarantee as to the order of arrival for prompts, touch notifications, etc. It is guaranteed that the messages arrive complete and do not overwrite each other. The debounce algorithm for touch processing ensures that the host is not overwhelmed by touch notifications.

### **4.14. *System Firmware Boot Process***

The sequence of operations that occur after power is applied to the SLCD5 is important for system configuring, macro application programming (via bitmaps and macros), demo execution, and updating firmware. These operations are indicated by output activity on the serial connection COM0 and LED (D2). Users can observe text messages of the Boot Process with a terminal emulator running on a PC when connected to the MAIN RS232 port of the POWERCOM4 board.

The System Software consists of two major components: the Bootloader and the main Firmware. The Bootloader is responsible for hardware component initialization, and Firmware updating. The Firmware is responsible for the standard board application functionality as outlined in this document. The diagram below explains the high level sequence of major operations between these major software components. See sections on System Configuration and Application Programming (Bitmaps and macros).

## Algorithm

The following pseudo code is an explanation of the System Software Boot Process. Note that files that are just checked for presence (e.g. FLASH?.INI) should be non-zero length.

### **Boot Loader begins execution**

Hardware Component Initialization

IF “bauds” command as set a “sticky” baud rate, use that instead of default 115200 ENDIF

Display version and copyright

Look in SD Card for “\*.ELF” firmware update file

Read last saved firmware file name from EEPROM

IF the two file names do not match (SD card has different firmware file) THEN

Update firmware in Flash (erase / program)

Display status

ENDIF

Display “Starting firmware”

Jump to Firmware code

**Boot Loader completed**

### **Firmware begins execution**

Read EEPROM

IF EEPROM contents not valid THEN

Set Default EEPROM Values

ENDIF

IF SD Card present THEN

IF CONFIG.INI file present THEN

Parse option keywords, set option variables

IF verbose = 1 THEN

Display (option keyword and value)

ENDIF

ENDIF

IF SPLASH.BMP file present THEN

Display SPLASH.BMP

ENDIF

IF SLCD\*.BIN present THEN

----- *SLCD only* -----

Copy .BIN file to DRAM, use as working set BMP/macro/fonts

IF FLASH1.INI present THEN

Copy DRAM to base board data flash (4MB)

ENDIF

----- *SLCD5 PLUS only* -----

IF FLASH2.INI file present THEN

Copy .BIN file to CPU board flash (28MB)

Use flash for working set BMP/macro/fonts

ELSE

Copy .BIN file to DRAM, use as working set BMP/macro/fonts

ENDIF

```

----- end of model-specific code -----
ELSE
  IF Flash memory contains a .BIN file THEN
    ----- SLCD only -----
    Copy flash to DRAM, use as working set BMP/macro/fonts
    ----- SLCD5 PLUS only -----
    Use flash for working set BMP/macro/fonts (no copy for speed)
    ----- end of model-specific code -----
  ENDIF
ENDIF

IF RUNDEMO.INI present THEN
  IF "demomac = <macro number>" found THEN
    Run <macro number>
    Display message on main port if "verbose = 1" previously found
  ENDIF
ENDIF

ELSE //SD Card NOT Present
  IF Flash memory contains a .BIN file THEN
    ----- SLCD only -----
    Copy flash to DRAM, use as working set BMP/macro/fonts
    ----- SLCD5 PLUS only -----
    Use flash for working set BMP/macro/fonts (no copy for speed)
    ----- end of model-specific code -----
  ENDIF
ENDIF

Start main firmware application

```

**Figure 7: System Software Boot Algorithm**

## 5. Software Command Reference

### 5.1. COMMANDS AFFECTING NON-VOLATILE SETTINGS (EEPROM)

Some commands affect settings that are stored in non-volatile memory; this memory is implemented using an I2C Serial EEPROM with a limit of 1 million write cycles.

*Warning: At 1 write per second, 24 hours a day, the EEPROM will exceed 1 million writes and become unreliable in only 11 days. Take care that commands marked as affecting EEPROM non-volatile settings are not continuously executed.*

*IT IS RECOMMENDED that commands writing to EEPROM only be used in development, and that the config.ini file (Section 3.4) method of setting these values be in production.*

### 5.2. COMMAND BASICS

All commands end with a <return> character, which is hex 0x0D, decimal 13. All responses from the SLCD5 end with a <return> character. Linefeeds are not provided in order to minimize communications overhead.

Arguments in angle brackets <..> are to be replaced by the characters described in the command arguments. The angle brackets themselves are not to be used. Square brackets with a vertical spacer '|' indicates optional arguments.

All commands that take x and y position arguments are affected by the SET ORIGIN command. This allows a set of graphic commands to draw the same object in different locations.

### 5.3. COMPRESSED SYNTAX

All ASCII commands are shown with a space after the command mnemonic, for example:

```
p <pixels>
```

This command sets the line drawing width. This space is optional in all commands where the first argument is numeric (e.g. not text display) and can be removed to reduce code space and transmission overhead. For example:

```
p2<return>
```

sets the line width to 2.

### 5.4. ASSUMED ORIENTATION

Note: All command descriptions assume the display is running in landscape mode. X and Y parameter limits need to be swapped for portrait mode.

## 5.5. TOUCH PRIORITY

If touch areas overlap, the one with the lowest index value (lowest button or hotspot number) has priority.

## 5.6. COMMANDS

### SET PEN WIDTH

Description: Sets the pen width for line drawing commands including line, rectangle but not circle. Default is width of 1 at power up, width of 2 after ‘[TOUCH CALIBRATE](#)’ command is executed. If given no argument, returns the current setting.

Command: `p <pixels>`

Arguments: `<pixels>` is a number from 1 to 200.

Example: `p 1`

This sets the pen width to 1 pixel wide.

### SET DRAW MODE

Description: Sets the drawing mode for all line draw commands including draw line, rectangle, and circle. Note that for color displays the XOR mode produces the bit-inverted RGB color. If given no argument, returns the current setting.

Command: `d [<n|x>]`

Arguments: `n`: Normal drawing mode; draws with the colors from Set Color.  
`x`: XOR drawing mode; inverts the existing pixel to draw lines.

Example: `d n`

This sets the drawing mode to normal.

Caution: In general, the XOR mode will not work as expected with pen width of more than 1. This is due to multiple pixel writes when rounded ends are drawn. The same issue arises with circles.

### SET CURSOR

Description: Sets the location where text will be displayed by default. This is used with the TEXT DISPLAY command where only the text to be displayed is the argument. This is useful when text is generated by a macro and the location is specified before the macro is invoked. Reset to (0,0) by “z” command. . If given no argument, returns the current setting.

Command: `sc [<x> <y>]`

Example: `sc 10 20`  
`t "hello"`

The above is equivalent to:

```
t "hello" 10 20
```

## SET TEXT ALIGNMENT

- Description:** Sets the alignment of the text with respect to the insert point specified. Used with the text display command. If given no argument, returns the current setting. *Note: The horizontal alignment is reset to "L" after text is written.*
- Command:** `ta [<L|C|R><T|C|B>]`
- Arguments:** The first letter argument is horizontal (Left, Center, Right), second is vertical (Top, Center, Bottom) alignment.
- Example 1:**
- ```
ta CC
t "hello" 100 100
```
- This draws the text "hello" centered horizontally and vertically around x=100, y = 100.
- Example 2** (assume VGA LCD):
- ```
ta RB
t "bottom right corner" 639 479
```
- This draws the text "hello" in the bottom right hand corner of the LCD screen.

## SET TEXT MODE

- Description:** Sets the text draw mode for subsequent TEXT DISPLAY commands. With no argument, the command returns the current mode.
- Command:** `tm [R|T|X|TR|N]`
- Arguments:** Same as TEXT DISPLAY, with N for "normal".
- Note:** When used within an animation, only guaranteed to persist until a yield occurs; likewise, within a macro, only guaranteed to persist until an animation interrupts the macro, or another macro starts (via a touch press or release event).

## SET ORIGIN

Description: Sets the origin, relative to the upper left corner of the display, for all subsequent operations including lines, text, bitmaps, buttons and so forth (but not another 'o' command - origin cannot be nested). This is useful for macros that draw compound objects. If the macro draws everything relative to (0,0), by setting the origin before calling the macro, the compound object can be placed anywhere on the screen. If given no argument, returns the current setting. *Note that the SET CURSOR command location is relative to this global origin, AND the "z" command will reset the origin to the upper left corner.*

Command: o [`<x>` `<y>`]

Arguments: `<x>` X axis value between 0 and 319 for QVGA or 479 for VGA.  
`<y>` Y axis value between 0 and 239 for QVGA or 639 for VGA.

Example: o 10 20  
t "hello" 0 0

This sets the origin to x=10, y=20, and then displays the text "hello" at absolute location 10, 20.



## SET COLOR (BASIC)

Description: Sets the background and foreground color for all commands using a basic color palette.

Command: `s <fore> <back>`

Arguments: `<fore>` = foreground color value per the table below  
`<back>` = background color value per the table below

Color value	Color	Color value	Color
<b>0</b>	Black	<b>10</b>	Light Grey
<b>1</b>	White	<b>11</b>	Light Blue
<b>2</b>	Blue	<b>12</b>	Light Green
<b>3</b>	Green	<b>13</b>	Light Cyan
<b>4</b>	Cyan	<b>14</b>	Light Red
<b>5</b>	Red	<b>15</b>	Light Magenta
<b>6</b>	Magenta	<b>16</b>	Yellow
<b>7</b>	Dark Brown		
<b>8</b>	Dark Grey		
<b>9</b>	Grey		

Example: `s 0 1`

From this point on, all objects will be drawn in black with a white background if applicable.

Note: To reset the background after changing the color, the screen can be cleared using the command, 'z'. The screen is cleared to the background color!

## SET COLOR (DETAILED)

- Description: Sets the background and foreground color for all commands using arbitrary RGB values. If given no argument, returns the current setting.
- Command: S [<fore\_detail> <back\_detail>]
- Arguments: <fore\_detail> = foreground color value in RGB format  
<back\_detail> = foreground color value in RGB format  
RGBformat = RGB where R, G, B are each a single character from 0 to f, or  
RRGGBB where RR, GG, BB is a value from 00 to FF
- Note: The recommended argument format is with the 24-bit color value RRGGBB. The 12-bit argument is preserved for SLCD compatibility.
- Example: S 00FF00 112233  
Foreground = maximum green, background = 0x11 red, 0x22 green, 0x33 blue.
- Usage note: To visibly change the screen to the background color after using this command, the screen must be cleared using the command, 'z'.
- Usage note: The SLCD5 uses 565-color encoding - that is, 5 bits for red, 6 bits for green and 5 bits for blue. Therefore, the full 24-bit value specified will be mapped to the 565 space.

## SET FONT

- Description: Sets the font to be used in subsequent TEXT DISPLAY, and BUTTON DEFINE commands. The "f?" command will list available fonts (built-in fonts and externally downloaded fonts, if any).
- Commands: f <font>  
f?
- Arguments: <font> is one of the following.  
**Proportional fonts**; equivalent to Windows Arial at point size shown; trailing 's' signifies not antialiased (s = "standard"):  
8s, 8Bs, 8, 8B, 10s, 10Bs, 10, 10B, 12, 12B, 14, 14B, 16, 16B, 20, 20B, 24, 24B, 32, 32B, 48, 64  
**Monospace fonts**; equivalent to Windows Monospac821 BT at point size shown. Fonts m48 and m64 are NOT antialiased.  
m8, m8B, m10, m10B, m12, m12B, m14, m14B, m16, m16B, m20, m20B, m24, m24B, m32, m32B, m48, m64
- Example: f 16B Sets the current font to 16 point Arial bold.
- Example: f? Displays a list of loaded fonts
- Example: f Displays the current font

## CLEAR SCREEN

Description: Clears the screen to the background color and removes any buttons and hotspots.

Command: `z`

## SET UTF8 ENCODING

Description: Enables or disables UTF8 encoding in text strings. When UTF8 is disabled, the 256 character extended ASCII character set per ISO 8859-1 is accessible. This is all that is needed for the basic font set. For downloaded fonts with Unicode sets larger than 256 characters, UTF8 encoding is used.

Command: `utf8 [on|off]`

Example: `utf8 on`  
`t "\xe4\xb8\x81"`

This displays the Unicode character 0x4e01 whose UTF8 equivalent is hex E4 B8 81. Note that an embedded host can send the UTF8 characters as three bytes - the text escape is not necessary unless the host can only send 7-bit ASCII.

## DISPLAY BITMAP IMAGE

Description: Copies stored bitmap onto the screen at x y (top left corner of bitmap target).

The Windows program BMPload.exe is used to download bitmaps into the SLCDx flash memory. These are accessed by index number.

Command: `xi <index> x y`

Arguments: `<index>` - bitmap index.

`x y` - location of top left corner of bitmap.

Example: `xi 4 10 20`

This displays the 4<sup>th</sup> bitmap at location (10, 20).

## DISPLAY BITMAP IMAGE CENTERED

Description: Same as xi command above, except the bitmap is centered at (x, y).

Command: `xim <index> x y`

Arguments: `<index>` - bitmap index.

`x y` - location of center of bitmap.

Example: `xim 4 100 120`

This displays the 4<sup>th</sup> bitmap centered at location (100, 120).

## DISPLAY CLIPPED BITMAP IMAGE

Description: Same as xi command above, except that a clipping area is applied so only part of the bitmap is displayed. This is useful to restore a part of a large graphic that has had text or graphics overlaid on it, for example when a graphic cursor is drawn on a map. When the cursor moves, the map area previously obscured by the cursor needs to be restored. The clip area is defined *relative to the top left of the bitmap* (i.e.: `x0 y0 x1 y1` are offsets from `x y`).

Command: `xic <index> x y x0 y0 x1 y1`

Arguments: `<index>` - stored bitmap index. (bitmap should not be compressed by BMPLoad)

`x y` - location of top left corner of bitmap

`x0 y0 x1 y1`

- rectangle within the bitmap to be displayed

Note: `0 <= x0 < bitmap width, 0 <= y0 < bitmap height,`  
`x0 <= x1 < bitmap width, y0 <= y1 < bitmap height`

Example: `xi 1 10 20 // draw main bitmap`  
`p 1 // set pen width to 1`  
`r 30 40 35 45 // draw rectangle`  
`// calc rectangle offsets (x0 y0 x1 y1):`  
`// (30-10) (40-20) (35-10) (45-10)`  
`xic 1 10 20 20 20 25 25`

This example draws a main bitmap #1. Then it places a rectangle on top of it. Then, instead of redrawing the entire bitmap to erase the rectangle, the xic command is used to only redraw a part of it.

## DISPLAY WINDOWED BITMAP IMAGE

**Description:** Displays a section (window) of a stored bitmap with an offset. This is used to implement a sliding window into a larger bitmap, for example, a section of a compass or ruler. It can also be used to simulate a rotating dial. **Only uncompressed bitmaps are supported by this command.**

*Note that the window is clipped and offset only in the specified direction. For example, with a horizontal compass bitmap, the visible rectangle width is specified with the length parameter, but the height is always the full vertical height of the bitmap.*

**Command:** `xio <index> <x> <y> <0|1> <length> <offset>`

**Arguments:** `<index>` - stored bitmap index. (bitmap should not be compressed by BMPLoad)

`<x> <y>` - screen coordinates for drawing location

`<0|1>` - 0: Vertical window  
- 1: Horizontal window

`<length>` - number of pixels to display along orientation

`<offset>` - offset into

**Note:** Highcolor firmware will only support Highcolor bitmaps or the xio command will return an error.

**Example:** `xio 4 10 10 1 100 50`

Bitmap #4 can be wider than the LCD screen; assume it is N pixels long and 45 pixels high. This command draws a rectangular screen area (10, 10) to (109, 54) with the source being bitmap #4 with a horizontal offset into the bitmap of 50 pixels.

## DISPLAY IMAGE FILE

**Description:** Displays an image file from the current directory on the SD Card (see [CHANGE SD CARD DIRECTORY](#)) onto the screen at x y (relative to top left corner of screen, which is location 0, 0)

**Command:** `xif <filename>.<ext> x y`

**Arguments:** `<filename>` is the name of the image file (max 8 chars)

`<ext>` is a 3 character extension identifying the type of image file:

- "bmp" bitmap
- "gif" GIF
- "jpg" JPEG

**Example:** `xif button.bmp 10 20`

This displays the bitmap image in file "button.bmp", found in the current directory of the SD Card, at location (10, 20).

## TEXT DISPLAY

**Description:** Displays text string starting at a specified point using the currently set font. Draws text in foreground color inside a background color box unless options are specified. The backslash ("\") is the escape character, used to create double quotes ("\""), newline characters ("\n"), backslashes ("\\"), or arbitrary characters ("\xhh). A newline will move the next character down one character block height in the implied box starting at the x pixel location.

**Command:** t "text string" x y [mode]  
or  
t "text string" x0 y0 x1 y1 [[mode][wrap/rotate]]  
or  
t "text string"

**Arguments:** x is the left edge of the first character areas.  
y is the top edge of the first character area.  
x0 y0 is top left corner of rectangle  
x1 y1 is bottom right corner of rectangle

[mode] is one of:

R – Reverse: foreground / background colors are reversed.  
T – Transparent: text written on top of current display with no "background box".  
X – XOR  
TR – Transparent reversed  
N – Normal: foreground / background colors are used.

[wrap/rotate] is one of:

WW – wrap text on word boundary  
WC – wrap text on char boundary  
CW – rotate text 90 degrees clockwise  
CCW – rotate text 90 degrees counter-clockwise  
I – rotate text 180 degrees (invert)

Notes:

Quotes are required around the text string. *The entire command including <return> must be less than 120 characters.*

In the first 2 forms of the command (see above) if the text mode (N, R, T, X, TR) is not specified, it will be reset to Normal; if it is specified, it will be used and will persist until another mode is specified (which may occur within an animation or a macro attached to a touch press/release event). Initially, Normal mode is used, and will remain in effect until changed. To prevent confusion about which mode the command will use, always explicitly declare it within the command. This is especially important within macros and animations.

In the second form of the command, there must be no space between the [mode] and the [wrap/rotate] if both are present.

Examples:

```
t "Press \"next\" \"nto continue" 10 0 N
```

This displays the text

```
Press "next"  
to continue
```

With the top left corner of the 'P' at location x=10, y=0, in Normal mode

```
t "\xa9Copyright" 0 0 R  
t "\n 1999-2009"
```

displays the text

```
©Copyright  
1999-2009
```

at the top left corner of the screen, in Reverse mode

```
f13B  
r 100 100 160 200  
ta CC  
t "This is in a box" 100 100 160 200 NWW
```

displays the text (in Normal Text Mode)

```
This is in a box
```

centered in a rectangle with word wrap enabled; "This is in" is the 1st line; "a box" is the 2nd line; the rectangle is at 100, 100 is 61 wide and 101 tall.

## GET TEXT DISPLAY WIDTH IN PIXELS

Description: Returns the width of the space required to display the given text using the current font, in pixels.

Command: `t? "text string"`

Examples: `f 20B`  
>  
`t? "string one"`  
134  
>  
`t? "string TWO"`  
149  
>

## TEXT FLASHING DISPLAY

Description: Creates an animation to display a flashing text string starting at the current or specified point using the currently set font. The string is alternately drawn in the current foreground color, then erased (by drawing in the background color) at a rate specified by the parameter <t> (delay time). Each alternate view is displayed for <t> mS. The total time to cycle is 2X the selected delay time. See TEXT DISPLAY for text string escapes and other details.

Command: `tf index t "text string" x y [R|N]`

All parameters except index and "text string" are optional. If unspecified, <t> will default to 500 mS, [x y] will be the current character position, and [R | N] will be N (Normal). A null text string "" is acceptable.

Arguments: <index> is an identifier for this animation; valid range is 0-9.

t is the number of milliseconds between flashes.

x is the left edge of the first character areas.

y is the top edge of the first character area.

R – Reverse flashing text: the string is shown in reverse text mode, and then erased.

N – Normal flashing text: the string is shown in normal text mode, then erased. This is the default mode of operation, and need not be specified.

Note: Quotes are required around the text string. ***The entire command including <return> must be less than 120 characters.***

Example: `tf 0 300 "FLASHING TEXT" 10 0`

This puts the text FLASHING TEXT with the top left corner of the 'F' at location x=10, y=0 with a delay of 300 Milliseconds between displayed and non-displayed text.

Note: The clear screen command 'z' clears all flashing text instances (and all other types of animations).



## TEXT FLASHING DISABLE

- Description:** Disables a flashing text animation as specified by the index (see tf command). The stopping point state can be specified.
- Command:** `tfd <index> <state>`
- Arguments:** `<index>` is the identifier for the animation; valid range is 0-9.  
`<state>` specifies the state to stop the animation, for text flash, this is 0 or 1.
- Examples:** `tfd 0 0`  
This stops the text flash animation at the first state with text in selected foreground color.
- `tfd 0 1`  
This stops the test flash animation at the second state with text in the selected background color.
- Note: To delete and re-use a text flash's animation index, use the "tfd <index> <state>" command to stop the animation at the selected state, and then use the "tfx <index>" command to delete the animation.

## TEXT FLASHING ENABLE

- Description:** Enables text flashing for individual strings as specified by the identifier. If the text flash animation is currently running for that identifier, no action is performed.
- Command:** `tfe <index>`
- Arguments:** `<index>` is the identifier for the animation; valid range is 0-9.
- Examples:** `tfe 0`  
This resumes the text animation from a previously stopped state.

## TEXT FLASHING DELETE

- Description:** Deletes the specified text flash animation.
- Command:** `tfx <index>`
- Arguments:** `<index>` is the identifier for the text; accepted identifiers are 0 through 9.
- Examples:** `tfx 0`  
This stops and deletes from memory the specified (0) text animation.

## TEXT FLASHING SYNCHRONIZATION

Description: Synchronizes all animations.

Command: `tfs`

Arguments: None

Examples: `tfs`

Resets all animations and flashing text to initial state. If animations or flashing text are using integral delays, the animation and text flashing will be performed in synchronization.

## TEXT FLASH ANIMATION ENABLE

Description: Re-Enables a currently stopped text flash animation.

Command: `tfe <index>`

Arguments: `<index>` is the identifier for the animation; valid range is 0-9.

Examples: `tfe 0`

Stopped test flash animation is re-enabled and executes.

## SAVE DRAWING ENVIRONMENT (STATE SAVE)

Description: Saves the current drawing state including color, pen and line style, cursor position and origin (margins), font, text alignment and drawing mode.

Note: each Macro call-level has its own memory for state save/restore, including call-level 0 (no macro running); also, the animation engine uses its own memory to save the existing state before animations run, and restoring it after.

Command: `ss`

Arguments: None

Examples: `ss`

## RESTORE DRAWING ENVIRONMENT (STATE RESTORE)

Description: Restores the drawing state. If the save state (`ss`) command has not been executed since power up or reset, the power up state is used. See Note above regarding state save/restore memory.

Command: `sr`

Arguments: None

Examples: `sr`

## PIXEL READ / WRITE

Description:	The "pw" commands write a pixel and the "pr" commands read a pixel at location (x, y).
Commands:	<code>pw x y [565 color value]</code> <code>pw x y [RRGGBB 6 character color value]</code>
Arguments:	if no color value is given, the pixel is written in the foreground color. 565 color = RRRRRGGGGGBBBBB (binary). RRGGBB = 24-bit color value as 6-character ASCII string where each color is 00 thru FF.
Example:	<code>pw 10 20 F801</code> This sets the pixel at (10, 20) to R=11111XXX (8 bit), G = 0, B = 00001XX.
Command:	<code>pr x y</code>
Returns:	565 color value as 4 ASCII hex digits, e.g. F801
Command:	<code>prx x y</code>
Returns:	24-bit color value as 6 ASCII hex digits, e.g. FF0008
Example:	<code>prx 10 20</code> (assume pw command above has been sent)
Returns:	F80008

## DRAW LINE

Description:	Draws a line from (x0, y0) to (x1, y1) using the foreground color and current pen width.
Command:	<code>l x0 y0 x1 y1</code>
Example:	<code>l 0 0 639 479</code> This will draw a line from the upper left-hand corner of the screen to the lower right hand corner of a VGA display (640x480).

## DRAW RECTANGLE

- Description:** Draws a rectangle using the current foreground color and pen width, or an alternate style and color.
- Command:** `r x0 y0 x1 y1`  
`r x0 y0 x1 y1 <style>`  
`r x0 y0 x1 y1 1 [color]`
- Arguments:** Upper left corner is (x0,y0) and lower right corner at (x1,y1).  
<style>: 1=filled, 2= one pixel wide dotted line.  
[color]: fill color in RGB format (see SET COLOR detailed); if not present, uses foreground color.
- Example:** `r 100 100 179 119`  
Draws a rectangle positioned at 100,100 with a width of 80 and a height of 20.  
`r 100 100 179 119 1`  
Draws a rectangle filled with the foreground color positioned at 100,100 with a width of 80 and a height of 20.  
`r 50 100 179 119 1 C03`  
Draws a rectangle filled with the color R=C,G=0,B=3 positioned at 50,100 with a width of 130 and a height of 20

## DRAW CIRCLE

- Description:** Draws a single pixel width circle using the foreground color. If the optional fill argument is supplied, the entire circle is filled with the foreground color.
- Command:** `c x0 y0 r [f]`
- Arguments:** Center is (x0,y0) with radius r. The circle is not filled if f is omitted, and filled if f=1.
- Example:** `c 100 100 50`  
Draws a circle centered at 100,100 with a radius of 50.  
`c 100 100 50 1`  
Draws a circle filled with the foreground color centered at 100, 100 with a radius of 50.

## DRAW TRIANGLE

Description: Draws a triangle using the current pen width and foreground color for the line. If the optional fill argument is supplied, the triangle is also filled with the specified color. Note: To fill without an outline border, set the pen width to 1.

Command: `tr x0 y0 x1 y1 x2 y2 [RGB | RRGGBB]`

Arguments: The three x, y sets are the triangle vertices. The optional color fill argument is three or 6 hex characters; see SET COLOR DETAILED command.

Example: `tr 10 10 10 100 200 200`

Draws a triangle with points (10,10), (10,100), (100,200).

`tr 10 10 10 100 200 200 0CC`

Same as above, but the triangle is filled with light blue.

## DRAW OUTLINE POLYGON

Description: Draws a polygon at specified origin, using current foreground color and pen width.

Command: `pg <X Orig> <Y Orig> <X/Y vertices....>`

Arguments: `<X Orig> <Y Orig>` is X/Y location for polygon.  
`<X/Y vertices ...>` are vertex end-points (MAX=11).

Example: `pg 100 100 0 0 15 15 15 10 40 10 40 -10 15 -10 15 -15 0 0`

Draws polygon at offset 100 100.

## DRAW FILLED POLYGON

Description: Draws a filled polygon at specified origin, using current foreground color.

Command: `pf <X Orig> <Y Orig> <X/Y vertices....>`

Arguments: `<X Orig> <Y Orig>` is X/Y location for polygon.  
`<X/Y vertices ...>` are vertex end-points (MAX=11)

Example: `pf 100 100 0 0 15 15 15 10 40 10 40 -10 15 -10 15 -15 0 0`

Draws filled polygon at offset 100 100.

## DRAW ROTATED POLYGON

Description: Draws a rotated polygon at specified origin.

Command: `pgr <Angle> <X Orig> <Y Orig> <X/Y vertices...>`

Arguments: `<angle>` Number of degrees to rotate polygon  
`<X Orig> <Y Orig>` is X/Y location for polygon.  
`<X/Y vertices ...>` are vertex end-points (MAX=11).

Example: `pgr 45 100 100 0 0 15 15 15 10 40 10 40 -10 15  
-10 15 -15 0 0`  
Draws polygon rotated 45 Deg CW at offset 100 100.

## DRAW ROTATED FILLED POLYGON

Description: Draws a rotated, filled, polygon at specified origin.

Command: `pfr <Angle> <X Orig> <Y Orig> <X/Y vertices....>`

Arguments: `<angle>` Number of degrees to rotate polygon  
`<X Orig> <Y Orig>` is X/Y location for polygon.  
`<X/Y vertices ...>` are vertex end-points (MAX=11).

Example: `pfr 45 100 100 0 0 15 15 15 10 40 10 40 -10 15  
-10 15 -15 0 0`  
Draws filled polygon rotated 45 Deg CCW at offset 100 100.

## REDRAW ROTATED POLYGON

Description: Draws a rotated, filled, polygon at specified origin.

Command: `ppgr <Angle> <X Orig> <Y Orig>`

Arguments: `<angle>` Number of degrees to rotate polygon  
`<X Orig> <Y Orig>` is X/Y location for polygon.

Example: `ppgr 45 100 100`  
Draws previously defined polygon rotated 45 Deg CW at offset 100 100.  
Defined polygon persists until overwritten by a new polygon definition.

## DRAW POLYLINE

Description: Draws a polyline at specified origin.

Command: `pl <X Orig> <Y Orig> <X/Y vertices...>`

Arguments: `<X Orig> <Y Orig>` is X/Y location for polygon.  
`<X/Y vertices ...>` are vertex end-points (MAX=11).

Example: `pl 100 100 35 -15 0 0 35 15`

## DRAW ROTATED POLYLINE

Description: Draws a rotated polyline at specified origin.

Command: `plr <Angle> <X Orig> <Y Orig> <X/Y vertices...>`

Arguments: `<angle>` Number of degrees to rotate polygon  
`<X Orig> <Y Orig>` is X/Y location for polygon.  
`<X/Y vertices ...>` are vertex end-points (MAX=11).

Example: `plr 45 100 100 35 -15 0 0 35 15`  
Draws polyline rotated 45 Deg **CCW** at offset 100 100.

## DRAW FILLED ELLIPSE

Description: Draws a filled ellipse.

Command: `ef x y <x radius> <y radius>`

Arguments: `x y` Specifies the center point of the ellipse.  
`<x radius>` Specifies the X radius of the ellipse.  
`<y radius>` Specifies the Y radius of the ellipse.

Notes:

1. Ellipse radii are limited to values of 180.
2. Ellipses are limited to horizontal and vertical orientation.

Example: `ef 150 150 30 50`  
Draws an ellipse centered at 150X150, Ellipse is vertically orientated.

## DRAW ELLIPSE

Description: Draws an ellipse.

Command: `e x y <x radius> <y radius>`

Arguments: `x y` Specifies the center point of the ellipse.  
`<x radius>` Specifies the X radius of the ellipse.  
`<y radius>` Specifies the Y radius of the ellipse.

Notes:

1. Ellipse radii are limited to values of 180.
2. Ellipses are limited to horizontal and vertical orientation.

Example: `e 150 150 30 50`  
Draws an ellipse centered at 150X150. Ellipse is vertically orientated.

## DRAW ARC SEGMENT

Description: Draws an ARC segment.

Command: a <X0> <Y0> <Radius> <Start Angle> <End Angle>

Arguments: <X0> <Y0> Center point of the ARC segment.  
<Radius> Radius of arc (Pixels)  
<Start Angle> Starting angle (Degrees)  
<End Angle> Ending angle (Degrees)

**Angles are CCW from 0=horizontal right (on compass, East).**

Example: a 100 100 40 20 110  
Draws a semi-circle centered at 100X100.

## SCROLL SCREEN AREA

Description: Scrolls a screen area up, down, left, or right. The background color is used to fill in the moved pixels. Can also rotate left or right by one pixel.

Command: k x0 y0 x1 y1 <numlines>[l|r|u|d|L|R]

Arguments: x0 y0 x1 y1 – defines the rectangle area for the scroll.  
<numlines> - number of lines to scroll. Must be 1 for 'L' or 'R' action.  
l = left scroll; r = right scroll; u = up scroll; d = down scroll  
L = left rotate 1 pixel (<numlines> must be 1)  
R = right rotate 1 pixel (<numlines> must be 1)

Note: 1. <numlines> is limited to the number of pixels in the axis of the scroll.  
2. E.g. If the rectangle is 10x X 20y pixels, the maximum X <numlines> is 10 and the maximum Y <numlines> is 20.

Example: f13B  
t "line 1\nline 2" 100 120  
k 100 120 140 146 13u  
This displays two lines of text and then scrolls up the text area such that the lower line replaces the upper line.



## BUTTON DEFINE - MOMENTARY

**Description:** Defines a momentary touch button on the screen. When touched, the host is notified, and optionally a macro can be invoked – see TOUCH MACRO ASSIGN.

**Note:** When a button is number is redefined, all macro assignments are cleared.

**Command:** bd <n> <x> <y> <type> "text" <dx> <dy> <bmp0> <bmp1>

**Arguments:**

<n> Button number, must be in the range of 0 to 127.

<x> <y> Upper left hand corner of the button

<type> Button type:

- 1 Standard. Displays <bmp0> normally, and <bmp1> when pressed. Host is notified when button is pressed, but not when it is released.
- 3 Typematic. Same as regular but with typematic functionality; that is, host notification repeats after the button is held down. See SET TYPEMATIC PARAMETERS command.
- 30 Typematic; same as type 3 above, except that subsequent host notifications do not generate a beep.
- 4 Standard except host is notified only when the button is released.
- 5 Standard with both press and release notification.

"text" Text string to be displayed on the button. The current foreground color will be used for the text. For multi-line text, use the newline ('\n') character decimal 10.

<dx> Text offset in the x direction from the upper left-hand corner of the button.

<dy> Text offset in the y direction from the upper left-hand corner of the button.

<bmp0> Index of bitmap displayed in the unpressed state.

<bmp1> Index of bitmap displayed in the pressed state.

**Note:** Both bitmaps must be the same size.

Host notification, type 1, 3, or 5 when button pressed:

x<n><return>

Host notification, type 4, 5 when button released:

r<n><return>

## BUTTON DEFINE – MOMENTARY (continued)

Example: `bd 23 150 100 1 "Test" 10 12 2 3`

Defines button number 23 displayed at x=150, y=100. The "un-pressed" image uses bitmap 2 with the text "Test" drawn on the bitmap in the current font at offset x=10, y=12 from the top left corner of the bitmap. The "pressed" image is the same except bitmap 3 is used. Bitmaps 2, 3 must be loaded and have the same size. When pressed, the host is sent:  
`x23<return>`

Example: `bd 0 10 20 5 "" 0 0 5 6`

Defines button 0 displayed at x=10, y=20. The "un-pressed" image uses bitmap 5, and the "pressed" image uses bitmap 6. No text is supplied so the bitmaps themselves must contain the description. For example, the bitmap 5 could show a toggle switch in the "up" position, and bitmap 6 could show a toggle switch in the "down" position. Bitmaps 5, 6 must be loaded and have the same size. When pressed, the host is sent:  
`x0<return>`

When released, the host is sent:

`r0<return>`

## BUTTON DEFINE – LATCHING STATE

- Description:** Defines a touch button on the screen with two distinct states. This is the equivalent of a retractable pen actuator – push it down, it clicks and stays down; push it again and it comes back up. When touched, the host is notified. A macro can also be invoked from a button press – see [TOUCH MACRO ASSIGN](#).
- Command:** `bd <n> <x> <y> <type> "text0" "text1" <dx0> <dy0> <dx1> <dy1> <bmp0> <bmp1>`
- Arguments:**
- `<n>` Button number, must be in the range of 0 to 127.
  - `<x> <y>` Upper left hand corner of the button
  - `<type>` Button type:
    - 2 Latching. Displays `<bmp0>` in state 0 and `<bmp1>` in state 1
    - 20 Latching. Same as above. (Initial state is set to state 0)
    - 21 Latching. Same as above, with initial state set to state 1
  - `"text0"` Text string to be displayed on the button in state 0. The current foreground color will be used for the text. For multi-line text, use the newline ('\n') character decimal 10.
  - `"text1"` Text string to be displayed on the button in state 1. The current foreground color will be used for the text..
  - `<dx0>` Text offset in the x direction from the upper left-hand corner of the button for `"text0"`.
  - `<dy0>` Text offset in the y direction from the upper left-hand corner of the button for `"text0"`.
  - `<dx1>` Same as above for `"text1"`.
  - `<dy1>` Same as above for `"text1"`.
  - `<bmp0>` Index of bitmap displayed in state 0.
  - `<bmp1>` Index of bitmap displayed in the state 1.
- Note: Both bitmaps must be the same size.

**Host notification:** `s<n> <s><return>` where `<s>` is 0 or 1 for the new state. Note the space between the button index and the state value.

**Example:** `bd 3 20 30 2 "GO" "STOP" 10 5 3 5 7 8`  
Define a latching button #3 at x=20, y=30 using bitmaps 7 and 8 with the text "GO" displayed in state 0 at offset (10,5) and "STOP" in state 1 at offset (3,5).

`bd 3 20 30 2 " " " 0 0 0 0 2 3`

Define a button as above, but use bitmaps that have the GO and STOP text as part of the bitmaps so no text is needed.

## BUTTON DEFINE CENTER TEXT

**Description:** Defines a momentary or latching state touch button on the screen. The text for the button(s) is automatically centered vertically and horizontally. The difference between this command and other BUTTON DEFINE commands syntactically, is that the text offsets are not needed.

**Command:** `bdc <n> x y type "text0" ["text1"] bmp0 bmp1`

**Arguments:** `<n>` Button number, must be in the range of 0 to 127.

`x y` Upper left hand corner of the button bitmap.

`type` Button type:

All types supported in `BUTTON DEFINE – LATCHING STATE` and `BUTTON DEFINE – MOMENTARY` commands.

`"text0"` Text string to be displayed on the button. Quotes are required. The current foreground color will be used for the text. For multi-line text, use the newline ('\n') character decimal 10 in the string.

`"text1"` Additional argument for `BUTTON DEFINE – LATCHING STATE` types; text displayed on button in pressed state.

`bmp0` Index of bitmap displayed in the unpressed state.

`bmp1` Index of bitmap displayed in the pressed state.

**Note:** Both bitmaps must be the same size.

**Example 1:** `bdc 23 150 100 1 "Test" 2 3`

Defines button number 23 displayed at x=150, y=100. The "un-pressed" image uses bitmap 2 with the text "Test" drawn on the bitmap in the vertical and horizontal center of the bitmap. The "pressed" image is the same except bitmap 3 is used.

**Host notification:** See [BUTTON DEFINE– MOMENTARY](#) command.

**Example 2:** `bdc 24 150 200 2 "ON" "off" 2 3`

Defines button number 24 displayed at x=150, y=200. The "un-pressed" image uses bitmap 2 with the text "ON" centered on it. The "pressed" image uses bitmap 3 with the text "off" centered on it.

**Host notification:** See [BUTTON DEFINE – LATCHING STATE](#) command.

### **DEFINE HOTSPOT (VISIBLE TOUCH AREA)**

Description: Defines a touch area on the screen. When touched, this area's number will be returned on the serial control line. The area defined will be set to reverse video while touched.

Command: `x <n> x0 y0 x1 y1`

Arguments: `<n>` touch button number. Must be in the range of 128 to 255. `(x0,y0)`, and `(x1,y1)` specify the touch area for this hotspot.

Returns: `x<n><return>`  
when the corresponding button is pushed. Note that once a hotspot is defined, the return string can be transmitted at any time including during a command transmission to the unit (full duplex).

Example: `x 135 100 100 180 140`  
Creates a rectangular hotspot with width of 80 and height of 40.

### **DEFINE SPECIAL HOTSPOT (INVISIBLE TOUCH AREA)**

Description: Same as DEFINE HOTSPOT except that the touch area is not reverse video highlighted when touched. This allows a "hidden" touch area to be placed on the screen.

Command: `xs <n> x0 y0 x1 y1`

Arguments,  
Returns: Same as DEFINE HOTSPOT command.

Example: `xs 135 100 100 180 140`  
Draws a rectangular hotspot with width of 80 and height of 40.

### **DEFINE SPECIAL HOTSPOT (INVISIBLE, NO BEEP)**

Description: Same as DEFINE HOTSPOT except that the touch area is not reverse video highlighted and there is no audible beep when touched. This allows a silent, hidden touch area to be placed on the screen.

Command: `xsnb <n> x0 y0 x1 y1`

Arguments,  
Returns: Same as DEFINE HOTSPOT command.

## DEFINE TYPEMATIC TOUCH AREA

Description: Same as DEFINE HOTSPOT except that the touch area is typematic and will repeatedly send the return code if the area is pressed continuously.

Command: `xt <n> x0 y0 x1 y1`

Arguments,  
Returns: Same as DEFINE HOTSPOT command.

## DEFINE SPECIAL TYPEMATIC TOUCH AREA

Description: Same as DEFINE TYPEMATIC TOUCH AREA except that the touch area is not reverse video highlighted when touched and beeps only once when pressed continuously (use “xset -T” to enable repeated beeps).

Command: `xst <n> x0 y0 x1 y1`

Arguments,  
Returns: Same as DEFINE HOTSPOT command.

## DEFINE X-Y HOTSPOT

Description: Defines a touch area on the screen. When touched, this area’s number and the relative X and Y position of the touch is returned. When the area is touched, a beep is generated.

Command: `xyx <n> x0 y0 x1 y1`

Arguments: `<n>` touch button number. Must be in the range of 128 to 255. `(x0,y0)`, and `(x1,y1)` specify the touch area for this hotspot.

Returns: `x<n> <x> <y><return>`  
when the screen is touched in the hotspot area. Note that once a hotspot is defined, the return string can be transmitted at any time including during a command transmission to the unit (full duplex).

Example: `x 140 100 120 200 220`  
Enables a rectangular hotspot. When the screen location `x=110, y=140` is touched the following string is sent to the host:  
`x140 10 20<return>`

## DEFINE X-Y HOTSPOT (SILENT – NO BEEP)

Description: Same as DEFINE X-Y HOTSPOT without the beep.

Command: `xyynb <n> x0 y0 x1 y1`

## CHANGE (LATCHING) STATE BUTTON

Description: Changes the latching state button to a specified state. This can be used to implement a set of selection buttons where pushing one down causes the others to pop up.

Command: `ssb <n> <state>`

Arguments: `<n>` - latching button number (0-127)  
`<state>` - specifies the desired state (0 or 1).

Example: `ssb 5 1`

This command would force a button defined with `DEFINE BUTTON (type=2)` into state 1.

## BUTTON CLEAR

Description: Clears the definition for the specified button. *Note: This DOES NOT CHANGE THE SCREEN IMAGE.*

Command: `bc <n>`

Arguments: `<n>` - previously defined button number (0-127)

Example: `bc 3`

This command clears the definition of the previously defined button 3.

*Note: To clear all buttons, see the CLEAR ALL TOUCH "xc all" command*

## TOUCHSCREEN ON/OFF

Description: Enables or Disables the whole touch screen, or reports its status.

Command: `touch [on/off]`

Arguments: `[on/off]` - turns the touch screen on/off; if not given, reports the current setting: "touch on" or "touch off".

Example: `touch off`  
`bdc 1 10 10 1 "1"`  
`bdc 2 80 10 1 "2"`  
`bdc 3 150 10 1 "3"`  
`xmq 1 b1_macro`  
`xmq 2 b2_macro`  
`xmq 3 b3_macro`  
`touch on`

This example causes all touches to be ignored until after the buttons are defined and attached to macros.

## DISABLE TOUCH (HOTSPOT / BUTTON)

**Description:** Temporarily disables touch area or button. Once disabled, the button graphic may be overwritten by a pop-up or other element. Disabled touch areas / buttons can be re-enabled.

**Command:** `xd <n | "all"> [<n last>]`

**Arguments:** <n> touch area or button number. Must be in the range of 0 to 255, and must have been previously defined.

“all” disables all touch area or buttons (entire screen).

<n last> optional parameter that is the highest number of touch area or button; if present, all touch areas and buttons are disabled incrementally from <n> thru <n last>.

**Example:** `xd 1`

Disables previously defined button 1.

`xd 2 10`

Disables touch area or buttons numbered 2-10.

## ENABLE TOUCH (HOTSPOT / BUTTON)

**Description:** Re-enables touch area or button. For buttons, the state of the button is remembered and the correct graphic is displayed.

**Command:** `xe <n | "all"> [<n last>]`

**Arguments:** <n> touch area or button number. Must be in the range of 0 to 255, and must have been previously defined.

“all” re-enables all touch area or buttons (entire screen).

<n last> optional parameter that is the highest number of touch area or button; if present, all touch areas and buttons are re-enabled incrementally from <n> thru <n last>.

**Example:** `xe 1`

Enables previously disabled button 1.

`xe 2 10`

Enables touch area or buttons numbered 2-10.



## SET TOUCH CHARACTERISTICS

Description: Allows a button or hotspot characteristics to be modified after being defined. Useful to make a typematic hotspot.

Command: `xset <n> [+|-][p|r|t|T|x]`

Arguments: + (optional) add the following option  
- remove the following option

|                |  |
|----------------|--|
| <code>p</code> | notify on press  |
| <code>r</code> | notify on release  |
| <code>b</code> | beep on notify (normal, use -b to disable beep)          |
| <code>t</code> | typematic  |
| <code>T</code> | typematic special (beep on first press only)             |
| <code>x</code> | include relative x and y in notification (hotspots only) |

## CLEAR HOTSPOT

Description: Clears the previously defined hotspot (touch area).

Command: `xc <n>`

Arguments: `<n>` Hotspot (touch button) number. Value must be in the range of 128 to 255.

## CLEAR ALL TOUCH

Description: Clears all previously defined touch areas including the button touch areas.

Command: `xc all`

## SLIDER DEFINE

Description: Creates a slider object using background and slider control bitmaps.

Command: `sl idx bg x y slider off ornt inv cont hi lo`

Arguments: `idx` - slider index. Must be in the range 128 to 136 (maximum 8 sliders). Note that slider indices are shared with hotspot indices; that is if a slider is defined with index 128, hotspot index 128 cannot be used.

`bg` - background bitmap index

`x, y` - top left corner to place the background bitmap

`slider` - slider control (e.g. knob / button) bitmap index

`off` - slider offset from the edge of the background bitmap

`ornt` - orientation: 0 = vertical; 1 = horizontal

`inv` - invert: 0 = top / left is low; 1 = bottom / right is low

`cont` - Always one. Value has no impact.

`hi` - maximum slider value

`lo` - minimum slider value

Host notification when slider value is changed:

`l<idx>:<value>`

Example: `sl 128 44 100 30 45 5 0 1 1 100 0`

This example assumes that the demo bitmaps are loaded with 44 and 45 being the slider background and control respectively. A slider is created in the middle of the screen (left corner = 100, 30) in a vertical orientation with the control bitmap offset 5 pixels from the left edge of the background bitmap. The slider values range from 100 at the top to 0 at the bottom.

Example notification: `1128:50`

## SLIDER VALUE

Description: Sets the value of a previously defined slider object.

Command: `sv idx val`

Arguments: `idx` - slider index

`val` - value for the slider

Example: `sv 128 50`

Sets slider index 128 to value 50.

## METER DEFINE

- Description:** Creates a “Meter” object that resembles an analog meter (with an indicator). The meter object uses a background bitmap that visually represents the meter, and a polygon for the indicator needle.
- Command:** `md <idx> <bitmap> <x> <y> <type> <minVal>  
<maxVal> <init_val> <minAngle> <maxAngle> <x0 y0>  
<x1 y1> . . . [x10 y10]>`
- Arguments:**
- `idx` - meter index. The meter index must be in the range 0 to 7 (maximum 8 meters).
  - `Bitmap` - background bitmap index (bitmap should not be compressed by `BMPLoad`)
  - `x, y` - top left corner to place the background bitmap
  - `type` - always 1.
  - `minVal` - minimum numerical value for indicator
  - `maxVal` - maximum numerical value for indicator
  - `init_val` - initial numerical value for indicator
  - `minAngle` - minimum angle for minimum numerical value for indicator.
  - `maxAngle` - maximum angle for maximum numerical value for indicator
  - `x0 y0` - pivot point for indicator relative to 0,0 top left of bitmap
  - `x1 y1 . . . [x10, y10]`
    - polygon points for indicator relative to pivot point. Max 10 points
- Notes:** The angle values are with respect to the indicator as specified by the polygon points where 0 degrees is as drawn and degrees (only positive) move indicator clockwise.
- See [Draw Rotated Filled Polygon](#) for details on the operation of the indicator needle parameters.
- Example:** `md 1 48 0 0 1 475 515 500 270 90 126 120 -4 0 0 -  
78 4 0`
- This example defines a meter with index number 1, using bitmap index 48 as the background image. The type is always 1. The minimum value of 475 for the indicator is at angle 270 degrees (90 degrees to left of vertical), and the maximum value of 515 is at angle 90 degrees. The indicator will point to initial value 500. The indicator pivot point is 126 120 and the indicator is drawn as a vertical triangle with polygon points -4 0 0 -78 4 0.

## METER VALUE

- Description: Sets the value of indicator for a specified meter. The meter must have been previously created by the Meter Define command.
- Command: `mv id value`
- Arguments: `id` - meter index value previously defined.  
`value` - value to set indicator. Must be in the range of values as defined by the Meter Define command.

## CHART BITMAP DEFINE

- Description: Creates a chart to which data can be added. See CHART VALUE command to add data to a chart. The background of the chart is a bitmap. If more data points are added than can fit on the graph, the data starts again on the left in "Oscilloscope" style.
- Command: `cdb n x y dw bv tv bitmap <pens>`
- Arguments: `n` - chart index from 0 to 9 (maximum 10 charts).  
`x` and `y` are the top left corner coordinates. The bottom right corner coordinate is defined by the width (`x` axis length) and height (`y` axis length) of the chart area.  
`dw` - data width, number of pixels horizontally between chart data points  
`bv` - bottom data value (lowest `y` value)  
`tv` - top data value (highest `y` value)  
`bitmap` - bitmap index  
`<pens>` - one or more sets of two values: pen width and pen color; up to 8 pens can be defined. Width can be 1 -- 5, color is same format as "bc" parameter.
- Example: `cdb 0 10 20 4 0 99 72 2 0FF 1 F00`
- Defines a chart in the rectangular area defined by bitmap index 72, starting in the upper left (10,20). The lower right is defined by the bitmap's width and height. Each data value will be 4 horizontal pixels wide. The chart ('Y') values are scaled from 0 to 99. The background bitmap is index 72. Two pens are defined: the first is pen width 2, color teal (0FF), the second is pen width 1, color red (F00).

## CHART DEFINE

|              |  |
|--------------|--|
| Description: | <p>Creates a chart to which data can be added. See CHART VALUE command to add data to a chart. If more data points are added than can fit on the graph, behavior is determined by chart type:</p> <ul style="list-style-type: none"><li>• 0 (STRIP): initially, data is added at the left edge of the chart until the right edge is reached; then, current data is shifted left and new data is added at the right hand edge of the chart, like a strip chart recorder.</li><li>• 1 (OSCILLOSCOPE): the new data is added at the left edge of the chart, overwriting the oldest data, like an oscilloscope.</li><li>• 3 (STRIP starting at RIGHT EDGE): data is always added at the right edge of the chart after shifting the current data left.</li><li>• <i>NOTE: Type 2 is reserved for internal use by the "cdb" command (see CHART DEFINE BITMAP, above)</i></li></ul> |
| Command:     | <code>cd n x0 y0 x1 y1 t dw bv tv bc &lt;pens&gt;</code>   |
| Arguments:   | <p>n - chart index from 0 to 9 (maximum 10 charts).</p> <p>x0 , y0 and x1 , y1 are the top left corner and bottom right corners of the chart area</p> <p>t - chart type; must be 0, 1, or 3 (see Description, above)</p> <p>dw - data width, number of pixels horizontally between chart data points</p> <p>bv - bottom data value (lowest y value)</p> <p>tv - top data value (highest y value)</p> <p>bc - background color in RGB format (3 ASCII hex characters – see SET COLOR DETAILED)</p> <p>&lt;pens&gt; - one or more sets of two values: pen width and pen color; up to 8 pens can be defined. Width can be 1 -- 5, color is same format as "bc" parameter.</p>   |
| Example:     | <pre>cd 0 10 20 110 120 1 4 0 99 333 2 0FF 1 F00</pre> <p>Defines a chart in the rectangular area (10,20), (110,120). Each data value will be 4 horizontal pixels wide. The chart ('Y') values are scaled from 0 to 99. The background color is dark gray (333). Two pens are defined: the first is pen width 2, color teal (0FF), the second is pen width 1, color red (F00).</p>   |

### CHART REDEFINE PEN

Description: Redefines a given chart's pen.

Command: `cdp <n> <pen> <width> <color>`

Arguments: `n` - object index from 0 to 9 (maximum 10 charts).

`pen` - pen number, 1-8

`width` - pen width

`color` - 3 color in RGB format (3 ASCII hex characters – see SET COLOR DETAILED)

### CHART REDEFINE BACKGROUND COLOR

Description: Redefines a given chart's pen; useful to highlight a portion of a trace.

Command: `cdbc <n> <color>`

Arguments: `n` - object index from 0 to 9 (maximum 10 charts).

`color` - 3 color in RGB format (3 ASCII hex characters – see SET COLOR DETAILED)

### CHART REDEFINE BACKGROUND BITMAP

Description: Redefines a given chart's background bitmap; useful to highlight a portion of a trace on a chart with a bitmap background.

Command: `cdp <n> <bitmap>`

Arguments: `n` - object index from 0 to 9 (maximum 10 charts).

`bitmap` - bitmap index

`width` - pen width

`color` - 3 color in RGB format (3 ASCII hex characters – see SET COLOR DETAILED)

### CHART REDEFINE RETRACE MODE

Description: Redefines a given OSCILLOSCOPE type (with or without a bitmap background) chart's Retrace Mode; default mode is on.

Command: `cdrm <n> <on|off>`

Arguments: `n` - object index from 0 to 9 (maximum 10 charts).

## CHART REDEFINE CLEAR-BEFORE-DRAW MODE

Description: Redefines a given OSCILLOSCOPE type (with or without a bitmap background) chart's Draw Mode; default mode is on.

Command: `cddm <n> <on|off>`

Arguments: `n` - object index from 0 to 9 (maximum 10 charts).

## CHART VALUES

Description: Adds data points to previously defined chart. Note: If multiple pens are defined, they are drawn in order first to last – if multiple pens have the same value only the last pen color will be visible. If a minus sign character, '-', is present instead of a `pen_value`, the associated pen will not draw on the chart, leaving a gap in the line for that pen.

Command: `cv n pen0_value [pen1_value ..]`

Arguments: `n` - chart index from 0 to 9 (maximum 10 charts).

`pen0_value` - value to be added for pen 0. Must be in the range previously defined for chart 'n'.

`pen1_value` - additional values for each pen defined for chart 'n'. Must be in the range previously defined for chart 'n'.

Example:

```
cd 0 10 20 110 120 1 4 0 99 333 2 0FF 1 F00
cv 0 30 50
cv 0 40 60
cv 0 - 60
cv 0 40 -
```

Defines a chart (see CHART DEFINE) and enters a value of 30 for the teal pen and 50 for the red pen. The lines will be 4 horizontal pixels long for each. The second `cv` command extends the teal pen another 4 pixels in the X+ (left to right) direction and to 50 in the Y axis. The red pen moves 4 pixels in the X+ direction and to 60 in the Y axis. The third `cv` command does not draw the teal pen, but does extend the red pen another 4 pixels. The fourth `cv` command draws a 4 pixel line segment at value 40 with the teal pen, but does not draw the red pen.

## BINARY CHART VALUES

- Description:** Same as CHART VALUES except that multiple values are sent using binary data format for faster drawing speed. Use this command if the standard chart values command is too slow. Note that the pen value is limited to maximum 65535. There is a timeout of 5 seconds if the amount of data specified in the command is not received. If a timeout occurs, the error prompt "! \r" is returned.
- Command:** `bcv n number_of_pen_values_to_follow  
<lo byte pen0 value><high byte pen0 value>  
<lo byte pen1 value><high byte pen1 value>  
.  
.  
<lo byte penN value><high byte penN value>  
.  
.  
.`
- Arguments:** `n` - chart index from 0 to 9 (maximum 10 charts).  
`number_of_pen_values_to_follow` - this is equivalent to the number of bytes to follow divided by two.  
`pen0_value` - value to be added for pen 0. Must be in the range previously defined for chart 'n'.  
`penN_value` - additional values for each pen defined for chart 'n'. Must be in the range previously defined for chart 'n'.
- Example:** `Send: "cd 0 10 20 110 120 1 4 0 99 333 2 0FF 1  
F00\r"  
Get: ">\r"  
Send: "bcv 0 5\r"  
Get: ">\r"  
Send (10 bytes): 0x00, 0x05, 0x05, 0x0A, 0x0A, 0x14,  
0x0F, 0x1E, 0x14, 0x28  
Get: ">\r"`  
The example defines a chart (see CHART DEFINE) and adds values of 0,5,10,15,20 for the teal pen and 0,10,20,30,40 for the red pen. Each data point is 4 pixels to the right of the last one.

## CHART RESET PENS TO START

- Description:** Resets a given chart's pens to their starting position, without changing what's already displayed.
- Command:** `cr <n>`
- Arguments:** `n` - object index from 0 to 9 (maximum 10 charts).



## CHART CLEAR DISPLAY AREA

Description: Clears a given chart's display area and resets it's pens to their starting position.

Command: `cc <n>`

Arguments: `n` - object index from 0 to 9 (maximum 10 charts).

## LEVELBAR DEFINE

Description: Defines a "levelbar" object. The object provides scaling and different colors for different levels, similar to a sound level meter. Note that the object is not visible until a value is assigned – see the LEVELBAR VALUE command.

Command: `ld n x0 y0 x1 y1 or inv bv bc <levels>`

Arguments: `n` - object index from 0 to 9 (maximum 10 charts).

`x0` , `y0` and `x1` , `y1` are the top left corner and bottom right corners of the object's area

`or` - orientation: 0 = vertical, 1 = horizontal

`inv` - invert: 0 = no (low value at bottom / left); 1 = yes (low value at top / right)

`bv` - bottom data value; should be 1 if value 0 means no level displayed

`bc` - background color in RGB or RRGGBB format (3 or 6 ASCII hex characters – see SET COLOR DETAILED)

`<levels>` - one or more sets of two values: value and associated color. These start with the maximum and go down. At most 3 sets are possible. Color is the same format as the `bc` parameter.

Example: `ld 0 10 10 30 200 0 0 1 333 99 F00 50 FF0 40 0F0`

Defines a levelbar in the rectangular area (10,10), (30,200). Levelbar is vertical with the lowest value at the bottom; minimum visible value of 1, with background color dark gray (333). Three color bands are defined: red (F00) from 99 to 51, yellow (FF0) from 50 to 41, and green (0F0) from 40 to 1.

## LEVELBAR VALUE

Description: Sets the value of a previously defined "levelbar" object.

Command: `lv n val`

Arguments: `n` - object index  
`val` - value for the levelbar.

Example: `lv 0 50`  
Sets levelbar 0 to value 50.

## MACRO EXECUTE

Description: Runs a macro (list of commands) previously stored in flash memory. The BMPload.exe program is used to store both macros and bitmaps into the flash; see [Appendix D](#). See [Appendix E](#) for the macro file format.

The stored macros can be defined to take arguments when called. In this case, the arguments are specified by this command. For more details on parameterized macros, see [Appendix E](#).

*NOTE: The maximum number of arguments is 10, and the maximum size of arguments is only limited by total command line length (max 128). See [Appendix E](#).*

Command: `m <n|name> [macro parameters . . .]`

Arguments: `<n|name>` is the macro index number (1 to 254), or the macro name. If the macro takes arguments, the values are supplied in order after the macro number. They are delimited by spaces. If a space is to be included in an argument, the argument must be enclosed with double quotes.

Example: `m2`  
This causes macro #2 to execute.

Example: `m test3 " " 2`  
This causes macro "test3" to execute with a value for the first parameter of a space character, and the value of the second parameter the number 2.

Command: `m <n|name>:<label> [macro parameters . . .]`

Arguments: Same as above, with macro label. See [Appendix E](#) for a full description.

Example: `m test3:lbl_4 arg1 arg2`

## MACRO ABORT

- Description:** This command stops execution of the current running macro. In addition, the command flushes (resets) the incoming command buffer. It is useful for instance if the host has sent a long sequence of commands to update data on a screen. While those commands are being processed by the SLCD5, a button is pushed which means "draw different screen". The host then sends a MACRO ABORT command and waits for the response. The SLCD5 can start drawing the new screen immediately, instead of having to wait until the previously buffered commands were finished.
- Command:** \*abt
- Host Notification:** :aborting on \*abt<return> ... 1 line per macro call level  
'<return> ... the success prompt, indicates successful abort of executing macro(s).

## TOUCH MACRO ASSIGN

- Description:** Links a button or hotspot to a macro. When the button or hotspot is touched, the associated macro is executed. See the [MACRO NOTIFY](#) command for host notification of macro execution options.
- Command:** xm <touch index><macro index | name> [<macro2 index | name>]
- Arguments:** <touch index> is the index of the button or hotspot.  
<macro index | name > is the index (or name) of the macro to be executed when the button or hotspot is pressed, or in the case of latching buttons, when the button is pressed to change from state 0 to state 1.  
<macro2 index | name> is an optional parameter. In the case of button or hotspot, this specifies a macro to be executed when the touch area is released. For latching buttons, this macro is executed when the button changes state from state 1 to 0.  
Within the <name> an optional system-constructed, predefined label (based upon the type of button or hotspot) may be concatenated. The format for these labels only applies to this command. The specific format of the predefined label is related to the host response for the type of hotspot or button. The format for the predefined label is:  
'<response character: 'x' | 'r' | 's'><touch index>[\_<state: '0' | '1'>]  
For a latching state button with index 13, the labels would be ":s13\_0" (button 13 in state 0) or ":s13\_1" (button 13 in state 1).  
For a momentary button with index 14 defined to notify on press only, release only, or both, the labels would be ":x14" (button 14 pressed) or ":r14" (button 14 released).  
Similarly, for a hotspot with index 150, the label would be ":x150" (hotspot 150 touched or released).  
NOTE: when using predefined labels, both macros (pressed and released) must include labels.

Examples:

```
xm 128 2
```

This will run macro #2 when hotspot 128 is pressed.

```
xm 128 2 3
```

This will run macro #2 when hotspot 128 is pressed, and #3 when it is released.

```
bd 2 150 100 2 "OFF" "ON" 30 10 30 10
```

```
xm 2 5 3
```

This creates a latching button and executes macro 5 when the button is switched to "ON" and macro 3 when the button is switched to "OFF"

```
bdc 1 100 100 20 "TURN ON" "TURN OFF"
```

```
xm 1 button_laction:s1_1 button_laction:s1_0
```

This creates a latching button with centered text. The initial state is 0. When the button is pressed and the state becomes 1, macro "button\_laction" is executed, including the macro statements following the label "s1\_1".

## TOUCH MACRO ASSIGN QUIET

Description: This has the same functionality as [TOUCH MACRO ASSIGN](#) except that the standard button response to the host is disabled AND pushing the button does not cause a beep. This is useful when the macro contains an OUT command to generate arbitrary button responses.

Command: `xmq <touch index><macro index | name> [<macro2 index | name>]`

Arguments: See TOUCH MACRO ASSIGN.

Example: `xmq 5 2`

This will run macro #2 whenever button 5 is touched, and the standard button press response will not be given to the host.

## TOUCH MACRO ASSIGN WITH PARAMETERS

**Description:** Links a button or hotspot to a parameterized macro. When the button or hotspot is touched, the associated macro is executed with the specified arguments. If given no arguments, returns the amount of free argument storage space.

*NOTE: The maximum number of arguments, and maximum size of each argument is version-dependent (see [Appendix E](#)).*

**Command:** xa[q] <t><action><m>[<args>]

**Arguments:** <t> the index of the button or hotspot.

<action> is one of:

- p - execute the macro and arguments when the button is pressed (momentary) or when it changes from state 0 to state 1 (latching).
- l - same as above.
- r - execute the macro and arguments when the button is released (momentary) or when it changes from state 1 to state 0 (latching).
- 0 - same as above.

<m> the index of the macro to be executed when the button or hotspot is pressed.

[<args>] optional arguments for the macro. These are delimited by spaces. Double quotes can be used to surround an argument if it contains spaces..

**Example 1:**  
bd 1 100 100 1 "test" 10 15  
xa 1 p 17 Check

This defines button 1 and assigns macro 17 to run with the first argument = Check when button 1 is pushed. The corresponding macro definition could look as follows:

```
#define test 17  
t 0 0 `0`  
#end
```

When button 1 is pushed, macro 17 is invoked and the following command will be executed:

```
t 0 0 Check
```

## TOUCH MACRO ASSIGN WITH PARAMETERS (continued)

Example 2:        `xa 1 p 17 Check`

Assuming button 1 has been defined in a previous command, this assigns macro 17 to run with the first argument Check when button 1 is pushed. The corresponding macro definition could look as follows:

```
#define test 17
t 0 0 `0`
#end
```

When button 1 is pushed, macro 17 is invoked and the following command will be executed:

```
t 0 0 Check
```

## ANIMATION DEFINE

Description:       Defines a sequence of commands to be played back continuously or on demand, concurrently and independent of commands received from the communications port, macros and buttons. A delay function (see “Yield”) is used to suspend the animation for a specified period of milliseconds. The animation may be suspended at any “Yield” point (see “anid” ).

Animations are disabled when defined and must be activated using the “anie” (animate enable) command. An animation without a yield is executed once and suspended at the end of the animation. Animations cycle continuously unless a “yield stop” is contained in the animation script, the animation lacks a yield, or the “anid” command is issued to stop the animation.

An animation executes in a temporary state (see [State Save](#)), which exists only until the animation yields. Properties like foreground color must be set within the yield point they are to be used.

Command:           `ani <n> <text string>`

Arguments:         `<n>`                The index of the animation, 0 through 9

`<text string>` Any valid command ***Except:***

- An animation Define / Flashing Text Define command.
- A wait command (use Yield instead)
- A State Save/Restore command

Note: To control an animation using an animation script, the controlled animation or flashing text must be defined before defining the controlling animation. Only one animation may be defined at a time. Each “ani” command is used to define one graphics command; multiple commands may be incorporated into a single animation by breaking the display list into multiple lines, one command per line. Defining an animation with a different index closes the previous animation. Use the “anie” and “anid” commands to activate and deactivate defined animations. Use the “anic” and “anix” commands to delete animations.

Example:

The list of commands below implements the “New Features” demo included with the kit. Comments were added to assist the reader and are stripped when received by the display.

```
xi 7 0 0          // Background bitmap
anic             // Clear animation
// setup font and color for TF command
f 24B
S 0f0 fff       // Green text
tf 0 "FLASHING TEXT" 45 110 T

// Define animation #1 - Flashing LEDs
ani 1 xi 27 150 130 // Left LED On
ani 1 y 50          // wait 50 MS
ani 1 xi 26 150 130 // Left LED Off
ani 1 xi 27 210 130 // Middle LED On
ani 1 y 50          // Wait 50 MS
ani 1 xi 26 210 130 // Middle LED Off
ani 1 xi 27 270 130 // Right LED On
ani 1 y 50          // Wait 50 MS
ani 1 xi 26 270 130 // Right LED Off
ani 1 y 50          // Wait 50 MS
// End of animation #1
anie 1            // Start animation 1

// Define animation #2 - "ROTATE" left
continuously
ani 2 k 226 70 300 100 1 L// "ROTATE" left
ani 2 y 50        // Wait 50 MS
// End of animation #2
anie 2            // Start animation 2
k 100 60 180 110 10 u // Move "scroll" up

// Define animation #3 - "SCROLL" moves Down
ani 3 s 0 1       // Scroll fill color
ani 3 k 100 60 180 110 1 d// Scroll Down
ani 3 y 50        // Wait 50 MS
// End of animation #3

// Define animation #4 - "SCROLL" moves Up
ani 4 s 0 1       // Scroll fill color
ani 4 k 100 60 180 110 1 u// Scroll Up
ani 4 y 50        // Wait 50 MS
// End of animation #4

// Define Controlling animation #5, This
```

```
animation
// selectively enables and disables animation
scripts
// 3 and 4 successively for a period of ½ second.
// The effect is "SCROLL" scrolls up for a period
// ½ second, down for ½ second and repeats.

ani 5 anie 3          // Enable Scroll Down
ani 5 y 500           // wait ½ Sec.
ani 5 anid 3 0        // Stop at first yield
ani 5 anie 4          // Enable Scroll Up
ani 5 y 500           // Wait for ½ Sec.
ani 5 anid 4 0        // Stop at first yield
// end of animation #5
anie 5                // Start animation #5
S 000 fff
```



## ANIMATION LIST

**Description:** Lists the animation to the serial port for editing or incorporation into a macro, button or script. The animation is listed in a form suitable for cut and paste into other scripts. This method can be used to develop and tune animations, then incorporate the completed animation into a script.

**Command:** `ani? <n>`

**Arguments:** `<n>` The index of the animation, 0 through 9. If no parameter is given, the amount of free animation space in bytes is returned.

**Example:** When the command `tf 0 "hello world"` is entered to create a text flash animation, after issuing ``f12``, ``s 0 1``, and ``z`` commands:

```
f12
>
s 0 1
>
z
>
tf 0 "hello world"
>
ani? 0 Returns:

// ani:0 count: 500 Size: 134
ani 0 f 12
ani 0 S 000000 fffffff
ani 0 ta LT
ani 0 o 0 0
ani 0 sc 0 0
ani 0 tm N
ani 0 T "hello world"
ani 0 y 500
ani 0 f 12
ani 0 S fffffff fffffff
ani 0 ta LT
ani 0 o 0 0
ani 0 sc 0 0
ani 0 tm N
ani 0 T "hello world"
ani 0 y 500
>
```

## ANIMATION YIELD

Description: Suspends (sleeps) an animation for <Milliseconds> or stops the animation.

Note: The Yield command is only valid when executed in an animation script.

Command: `y [<Milliseconds> | stop]`

Arguments: <Milliseconds> Number of milliseconds to sleep this animation.  
`stop` Halt this animation until ANIE command issued.

## ANIMATION DISABLE

Description: Stops the animation specified by animation index and yield #.

Command: `anid <n> <Yield #>`

Arguments: <n> The index of the animation, 0 through 9.  
<Yield #> A reference to one of a animation's yield commands.

Yields are numbered for each animation starting at 0 (zero) and continues up to the number of yields-1 contained in that animation.

Note: Animations are "stopped" by advancing and executing those commands between the previous and selected yield.

Example: `anid 0 0`  
Stops animation 0 at the first yield command.

## ANIMATION ENABLE

Description: Enables animation execution for a specified animation.

Command: `anie <n>`

Arguments: <n> The index of the animation, 0-9.

Example: `anie 0`  
Enables animation 0.

## ANIMATION CLEAR

Description: Clear the animation and flashing text definitions and disables the animation engine.

Command: `anic`

Arguments: None

Example: `anic`  
Clears the animation buffers and stops the animation engine.

## ANIMATION DELETE

Description: Deletes the selected animation script.  
Command: `anix <N>`  
Arguments: `<n>` The index of the animation, 0 through 9.  
Example: `anix 0` Removes animation 0, reclaims animation memory.

## ANIMATION SYNCH

Description: Restarts all animations at beginning of scripts.  
Command: `anis`  
Arguments: None  
Example: `anis`

Any running animations are restarted.

Note: For best results, stop the animations using the `anid` command with the proper yield points to prevent graphics residue. Possible uses of this command are synchronizing flashing text or lamps.

## WAIT VERTICAL RETRACE

Description: Returns when a vertical display retrace occurs with an optional offset. Used to avoid "tearing" in animations.  
Command: `wvr [<line>]`  
Arguments: `<line>` Optional number of vertical lines to wait after retrace. Used to wait until the display trace has passed a specified vertical display line.  
Example: `wvr 135`

This waits until vertical refresh plus 135 vertical lines. Useful to put in an animation script before displaying a bitmap a x,100 with height 35. Note that for best results, bitmaps for animations should be stored uncompressed in flash memory.

## WAIT FOR REFRESH

**Description:** Imported from SLCD43/6/+ for compatibility. Similar to WAIT VERTICAL RETRACE, works only in Landscape orientation. Returns when a vertical display retrace occurs. This command is used to avoid "tearing" or "flashing" in animations.

**Command:** `wrf <x> <y>`

**Arguments:** `<x>` Ignored in SLC5; present for compatibility only.  
`<y>` This is the y coordinate. This is also the number of horizontal scan lines to wait after retrace. Note that the horizontal scan lines go from the top to bottom of screen.

**Note:** For best results, bitmaps for animations should be stored uncompressed in flash memory.

**Example:** `wrf 100 135`

The 100 is ignored (no portrait mode on SLCD5); same as `wvr 135`.

## OUTPUT STRING

**Description:** This outputs a text string to the serial port. This is typically used in macros that are assigned to buttons using the quiet feature above. This enables a button press to output arbitrary text to the serial port.

**Command:** `out "<text string>"`

**Arguments:** The text string can contain back tick enclosed variables or the following escapes:

`\\ = \`

`\\" = "`

`\` = ``

`\n = line feed`

`\r = return`

`\xhh = arbitrary character with hex value hh`

**Example:** `out "\x48ello \"world\"\\r"`

This will send the following string out on the serial port:

Hello "world"<return>

## WRITE TO AUX PORT

- Description:** Similar to OUTPUT STRING, but writes to the AUX communications port. Note that whatever port is acting as the main port cannot be written to this way; use the OUTPUT STRING command. Standard hex escapes are supported. A null byte must be sent as “\x00”.
- Command:** `about "<your message>"`
- Argument:** Quoted string to send “<your message>”
- Example:** `about "hello world\x01\xff"`  
Sends “hello world” followed by two bytes hex 01 and hex FF to the Auxillary serial port.
- Example:** `about "\x00"`  
Sends a single byte 0x00 to the Auxillary port.

## READ FROM AUX PORT

- Description:** Reads serial data from the AUX port. The AUX port receive buffer is 80 bytes long (79 characters plus a NULL char). If the buffer becomes full, any further data is thrown away. Outputs a ‘:’, followed by the received data in ASCII printable range 0x20-0x7E, followed by a standard prompt 0x3e 0x0d (“><return>”). If optional ‘b’ is present, outputs in binary, 1st byte output is number of bytes received, followed by the received bytes, then a standard prompt.
- Command:** `ain[b]`
- Example:** `ain`
- Returns:** `:<received data from COM 3><prompt>`

## SPEAKER ON / OFF (Rev D and later boards only)

- Description:** Turns the external speaker on/off and stores the setting in non-volatile memory ([see warning](#)) to be restored on power-on. Normally, the external speaker is off and the onboard beeper is on; when the speaker is turned on, the beeper is turned off. If no setting is given, returns the current setting.
- Command:** `*spkr [on|off]`
- Arguments:** `[on| off]` external speaker setting.

## BEEP ONCE

- Description: Beeps the beeper for <count> ms. This will temporarily interrupt any running repeating beep. A prompt is returned immediately even if the beep continues.
- Command: `beep <count>`
- Arguments: <count> is number of ms to sound the beeper.

## BEEP WAIT

- Description: Beeps the beeper for <count> ms. This will temporarily interrupt any running repeating beep. The system issues a command prompt only after the beep has stopped.
- Command: `beepw <count>`
- Arguments: <count> is number of ms to sound the beeper.

## BEEP VOLUME, BEEP VOLUME SPECIAL

- Description: Sets the volume level of the beeper; 'bv' stores the setting in non-volatile memory ([see warning](#)), whereas 'bvs' does not. When given no arguments, returns current setting.
- Command: `bv[s] [+|-]<level>`
- Arguments: <level> is number from 0 through 255. Default is 200. Loudest is 255. The optional '+' or '-' prefix changes the <level> into an increment up or down.
- Example: `bvs 120`
- Sets the beep volume setting to 120, but does not store the setting; on powerup, the value stored in non-volatile memory will be used.
- `bv (or bvs)`
- Returns 120 after the above command was issued.

## BEEP FREQUENCY, BEEP FREQUENCY SPECIAL

*NOTE: The beep frequency is set at factory to generate maximum loudness level.*

**Description:** Sets the frequency of the beeper; 'bf' stores the setting in non-volatile memory ([see warning](#)), whereas 'bfs' does not. When given no arguments, returns current setting.

This command is used during factory calibration to set the sound level as the sounders used resonate at slightly different frequencies. If you use it to change the frequency, the factory test results are invalid. Please do not use without premeditation! The \*MFGRESET command cannot restore the original value of this setting.

**Command:** bf[s] [<hertz>]

**Arguments:** <hertz> is number from 1 through 4000. Default is 2650, but may be slightly different due to volume calibration at the factory. If no argument is supplied, the current frequency is returned as a variable length decimal number.

**Example:** bfs 2500

Sets the beep frequency to 2500 Hertz, but does not store the setting; on powerup, the value stored in non-volatile memory will be used.

bf (or bfs)

Returns 2500 after the above command was issued.

## BEEP REPEAT

**Description:** Beeps the beeper for <on> ms, stays silent for <off> ms, and then repeats until the values are changed with another "rb" command. Can be temporarily overridden by a regular "beep" command. If <on> and <off> are both 0 the repeat stops.

**Command:** rb <on> <off> [alarm]

**Arguments:** <on> is number of ms to sound the beeper.

<off> is number of ms to stay silent before beeping again.

[alarm] is an optional parameter to use the alarm sound instead of a steady tone. See alarm command for valid alarm numbers.

**Example:** rb 100 400

Repeatedly beeps for 100 ms then goes silent for 400 ms during each 500 ms cycle.

## BEEP TOUCH

Description: Sets the duration of the audible feedback beep when a hotspot or button is pressed. Not stored in non-volatile memory. Default is 10 which equals 100ms beep.

Command: `bb <number>`

Arguments: `<number>` is tens of milliseconds to sound the beeper.

Example: `bb 10`

Sets the beep feedback to power-on value.

## ALARM

Description: Sounds an alarm sound using the beeper.

Command: `al <alarm> <count>`

Arguments: `<alarm>` is the alarm sound:

1 = whoop

2 = annoy

3 = dee-dah

`<count>` is number of ms to sound the beeper.

Example: `al 2 1500`

Sounds the "annoy" alarm for 1.5 seconds.

## WAIT

Description: Returns command prompt after a specified number of milliseconds. This is useful in macros that implement self-paced demonstrations as it delays execution of the next line.

Command: `w <number of milliseconds>`

Arguments: `<number of milliseconds>` is the number of milliseconds to delay, maximum is 65535.

Example: `w 1000`

This will return the command prompt in 1 second or in the case of a macro, delays execution by 1 second.

## DISPLAY ON/OFF

Description: Turns power to the display (and backlight) on or off.

Command: `v <on|off>`



## EXTERNAL BACKLIGHT ON/OFF

Description: Turns the external backlight control on or off via J10.  
Command: `xbl <on|off>`

## EXTERNAL BACKLIGHT BRIGHTNESS CONTROL

Description: Sets the brightness of the external backlight if the external unit supports this feature; 'xbb' stores the setting in non-volatile memory ([see warning](#)), whereas 'xbbs' does not. When given no arguments, returns current setting.

Command: `xbb[s] [+|-]<level>`

Arguments: `<level>` is number from 0 through 255. The 0 is the dimmest and 255 is brightest. The optional '+' or '-' prefix makes the level value an increment up or down rather than an absolute value. The value saturates at 0 and 255 without error; in other words if the level is at 255 and an "xbb +10" is issued, the level stays at 255 and no error prompt is issued.

Example: `xbb -10`  
This will reduce the brightness by 10 units but no lower than 0.

Example: `xbbs 255`  
This set the brightness to maximum and no save in EEPROM (executes quickly).

## SET BAUD RATE OF COM0 OR COM1 PORT

Description: Sets a new baud rate for COM0 or COM1 (baud0, baud1); if the port is not specified, the port currently in control will be set. This is temporary and the unit will revert to the default setting the next time power is cycled. Use a "power-on" macro to set the baud rate on power-up. Unless the command fails, the system prompt will be output at the new rate.

Command: `baud[0|1] <rate>`

Argument: `<rate>`: valid values are 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800, and 1200.

Example: `baud0 57600`  
This sets COM0 to 57600 baud and then outputs the system prompt.

Example: `baud 19200`  
This sets COM port currently configured as the Control/Main port to 19200 baud and then outputs the system prompt.

## SET BAUD RATE OF COM0 OR COM1 PORT (but output prompt first)

Description: Sets a new baud rate for COM0 or COM1 (baudp0, baudp1); if the port is not specified, the port currently in control will be set. This is temporary and the unit will revert to the default setting the next time power is cycled. Use a "power-on" macro to set the baud rate on power-up. The system prompt will be output at the old rate, before the rate is changed.

Command: `baudp[0|1] <rate>`

Argument: `<rate>`: valid values are 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800, and 1200.

Example: `baudp0 57600`

This outputs the system prompt and then sets COM0 to 57600.

Example: `baudp 19200`

This outputs the system prompt and then sets the COM port currently configured as the Control/Main port to 19200 baud.

## SET BAUD RATE OF COM0 OR COM1 PORT(sticky and prompt first)

Description: Sets a new baud rate for COM0 or COM1 (baud0, baud1) and stores it in non-volatile memory ([see warning](#)); if the port is not specified, the port currently in control will be set. The system prompt will be output at the old rate, before the rate is changed.

Command: `bauds[0|1] <rate>`

Argument: `<rate>`: valid values are 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800, and 1200.

Example: `bauds0 57600`

This outputs the system prompt, sets COM0 to 57600 baud, and then stores the setting in non-volatile memory, so it will be used again after a power cycle or reset.

Example: `bauds 19200`

This outputs the system prompt, sets the COM port currently configured as the Control/Main port to 19200 baud, and then stores the setting in non-volatile memory, so it will be used again after a power cycle or reset.

## TOUCH CALIBRATE

Description: Runs the touch calibration procedure. This displays calibration points on the screen and asks the user to touch them to calibrate the screen. Note that a command prompt is not given until the procedure has been completed. Calibration values are stored in non-volatile memory and restored on power-on. **NOTE: pen width gets changed to 2.**

Command: `tc`

## RESET TOUCH CALIBRATION

Description: Resets the touch calibration to a default value. Doing this before setting the entire screen to be a touch sensitive area guarantees that a touch will be seen independent of the current touch calibration. *NOTE: Only use this if it is to be followed by a touch calibrate command.*

Command: \*RT

## VERSION

Description: Displays the version of the software.

Command: vers

Returns: SLCD5 V<maj>.<min>.<bld> <resolution> <panel string>

Parameters: resolution - this is the resolution standard of the display (VGA, WVGA, SVGA).  
Panel string - Contact REACH Technology with Panel String to determine display panel information.

Example: SLCD5-PLUS V1.2.63 WVGA "VGG8048\_PVLED"

## SET LED

Description: The LED D2 on the board usually is controlled by the system and is on when the software is not idle. For debug and other purposes, the LED can be controller by the host using this command.

Command: led [on|off|sys]

Arguments: sys is the default state.

## READ FRAME BUFFER LINE

Description: Displays 640 (VGA) or 800 (WVGA, SVGA) comma separated frame buffer hex words for a given display line. Each word is 4 ASCII characters representing a 16-bit 565 value.

Command: \*FB <line>

Arguments: <line> is the display line buffer from 0 to 479 (VGA, WVGA) or 599 (SVGA).

## CRC SCREEN

Description: Returns the 16-bit CRC of the display buffer. This can be used to generate automated tests to verify correct user interface operation across a user's system software version changes.

Command: \*CRC

Returns: 0XXXXX<return> where XXXX is a hex number.

## CRC DATA FLASH

Description: Returns the 16-bit CRC of the data flash used to store macros and bitmaps. This can be used in production code to verify that the correct bitmaps are loaded in the board.

Command: \*CEXT

Returns: 0XXXXX<return> where XXXX is a hex number.

## CRC PROCESSOR FIRMWARE CODE

Description: Returns a 16-bit CRC of the entire processor code space. The purpose is to give an indication of the firmware version loaded.

Command: \*CSUM

Returns: 0xHHHH<return> where H is a single hex digit.

## CRC PROCESSOR BOOTLOAD CODE

Description: Returns a 16-bit CRC of the processor bootload code space (first 64KB of program flash).

Command: \*CSUMB

Returns: 0xHHHH <return> where H is a single hex digit.

## LIST FLASH DATA RECORDS

Description: Returns a summary of the records in data flash memory. This is for human debugging and the format is subject to change.

Command: ls

## LIST BITMAPS DETAIL

Description: Returns extended details of the bitmaps stored in downloadable flash memory. This is for human debugging and the format is subject to change.

Command: lsbmp [index]

Arguments: index - optional bitmap index used to display a single bitmap.

## LIST MACROS DETAIL

- Description: Returns extended details of the macros stored in downloadable flash memory. This is for human debugging and the format is subject to change.
- This command also lists the current button to macro assignments.
- Command: `lsmac [index]`
- Arguments: `index` - optional macro index used to display a single macro.

## LAST FIRMWARE FILE LOADED

- Description: When the bootloader finds a new firmware file on the SD card, it stores the name of the file in EEPROM. This command displays the name of the most recent firmware file loaded. Fresh from the factory, or after using the \*MFGRESET command ([see below](#)), the response will be "????????".
- Command: `*firmware`
- Returns: "`<eight ASCII characters>`"
- This is the 8 character DOS filename. The firmware file is this name with extension ".elf".

## READ TEMPERATURE

- Description: Displays temperature measured by sensor at location U10 (center of the board, under the processor module) in degrees Centigrade. The accuracy, and the minimum step size is approximately 1.5 degrees C.
- Command: `temp`
- Returns: `<degrees>.<tenths><return>`

Where `<degrees>` is a three character number with leading zeros as spaces and `tenths` is a single numeric character. The 'C' equivalent to produce this is

```
printf("%3d.%1d",degrees, tenths);
```

## RESET SOFTWARE

- Description: Issues a software reset to the processor. Used to simulate a power-on condition for testing. This command can take a second or so to execute.
- Command: `*RESET`
- Returns: "Power on" prompt.

## RESET BOARD TO MANUFACTURED STATE

Description: Clears the on-board EEPROM memory ([see warning](#)) and issues a software reset (see above). This restores the board to factory manufactured state with the exception that the contents of the data flash memory (bitmap and macro storage) is not affected.

***Note: After issuing this command, the touch screen MUST BE recalibrated using the "tc" command.***

Command: \*MFGRESET

Returns: "Power on" prompt.

## LOAD SYSTEM FILE FROM SD CARD (CURRENT DIRECTORY)

Description: Causes the file "SLCD????.BIN" to be loaded from the SD card current directory into memory; if the optional "SPLASH.BMP" file is present, the image in it will be displayed. For a complete description refer to the Firmware portion of "Figure 7: System Software Boot Algorithm". Note that file "SLCD????.BIN" is generated by the BMPload.exe program and contains bitmaps, macros, and external fonts.

Command: \*load (or \*LOAD)

## LOAD SYSTEM FILE FROM SD CARD (SPECIFIED DIRECTORY)

Description: Causes only the bitmap file "SLCD????.BIN" to be loaded from the SD card directory into memory. This file is generated by the BMPload.exe program and contains both bitmaps and macros.

Command: \*sdload <directory name>

Note: <directory name> follows the DOS 8.3 naming convention. The <directory name> argument limit is 16 characters so that directories can be specified using the . . and / (Linux)conventions.

Example: \*sdload /French

The file "SLCD????.BIN" (if present) is loaded into memory from the directory "/French" on the SD Card.

## CHANGE SD CARD DIRECTORY

Description: Changes the current working directory of the installed SD card. This can be used in conjunction with the xif and \*LOAD commands.

Command: \*cd <directory name>

Note: <directory name> follows the DOS 8.3 naming convention, so the directory name is limited to 8 characters.

Example: \*cd /Images

## LIST SD CARD DIRECTORY

Description: Lists the contents of the current working directory of the installed SD card. Directories are listed with the "/" char following their name.

Command: `*ls`

Note: Listed files and directories will follow the DOS 8.3 naming convention.

Example: `*ls`

```
RUNDEMO.INI
RT2_DEMO/
IKB_DEMO/
GFX_DEMO/
SLCD5PWV.BIN
SPLASH.BMP
```

## SAVE SCREEN SHOT TO SD CARD

Description: Causes an image of the entire screen ("screen shot") to be saved to the SD Card. This image generated is saved in the form of a BMP file format. The name of the file is SLCD###.BMP, where "####" is the decimal value indicating a count of presently saved files minus one, up to 1,000 (ie: first file is SLCD0000.BMP).

Command: `*getScreenAsBMPFile`

Note: The write performance of SD Cards can vary greatly. Saving an entire screen shot can take as long as 40 seconds.

Returns: `:writing SD File SLCD####.BMP....` (1 dot added each half-second or so to indicate progress; at completion, returns a prompt)

## DEBUG TOUCH

Description: Used for touch screen debugging. When set to 1, an "X" is written on the screen when a valid touch is detected. Also, additional information is displayed during touch calibration.

Command: `*debug <0|1>`

Returns: `[on|off]<return>`

## DEBUG MACRO

Description: Used to enable macro debug. When set, the commands in the macro are displayed as they are executed. This is useful when a macro stops due to a command error.

Command: `*macdebug <0|1>`

Returns: `[on|off]<return>`

## MACRO NOTIFY

**Description:** This command sets the desired macro execution notification. This is used when a button or hotspot is assigned to a macro (see [TOUCH MACRO ASSIGN](#)). By default when an assigned macro executes there is no notification to the host other than the button response. For debugging and software interface verification purposes, the host can be notified when a touch-invoked macro is executed, when it finishes, or both.

Note that the notification is sent after the button press response.

**Command:** \*macnote <0 | 1 | 2 | 3>

**Arguments:**

- 0 – turn notification off.
- 1 – send notification "m<index><return>" when macro starts.
- 2 – send notification "e<index><return>" when macro ends.
- 3 – send start and end notifications per 1 and 2 above.

**Returns:** off<return> or  
start<return> or  
end<return> or  
both<return>

## SPLASH SCREEN

**Description:** Selects a downloaded bitmap as the power-on "splash screen", which will be shown at power-on instead of the copyright notice and firmware version text string; the setting is stored in non-volatile memory (see [warning](#)). When given no arguments, returns the current setting (0 is the default, meaning 'off').

The Windows program BMPload.exe is used to download bitmaps into the SLCD5 data flash memory. See Appendix D for details.

**Command:** \*SPL <number>

**Arguments:** <number> is bitmap number as listed in the "ls" command. If 0 is used, no bitmap is selected and the standard product text string is displayed.

**Example** \*SPL 5

This displays the 5<sup>th</sup> memory record at location (0, 0) on power-on reset.



## POWER-ON MACRO

**Description:** Used to specify a macro to be executed when the unit is first powered on; stores the setting in non-volatile memory ([see warning](#)); when given no arguments, returns the current setting (0 is the default, meaning no macro will run at power-up). A common usage for a power-on macro is to set the baud rate to a value other than the default of 115200.

**Note:** The internally generated power-on copyright notice is displayed **AFTER** the power-on macro executes. This is done so the baud rate can be displayed. This can be disabled via the optional second parameter.

Note that the power-on copyright can also be suppressed by the splash screen option. If a splash screen is specified, the copyright notice is not displayed. The splash screen can be any bitmap, even a very small one that is the same color as the screen background.

**Command:** \*PONMAC <index | name> [<option>]

**Arguments:** (none) = display the current power-on macro index, or 0 for none.  
<index> = 0 or 255 disables the power-on macro feature  
<index | name> = 1 through 254 (or name) sets the power-on macro to the specified macro.  
<option> = optional argument; 0 means display the power-on copyright, and 1 means do not display it.

**Example:** \*PONMAC 2

## SET VARIABLE

- Description:** Used to set a value to an internal variable. If an internal variable (Integer, String, and Point Coordinate) is used, it should be after this command to have a meaningful value. When setting Eeprom values, note the EEPROM chip has limited total write cycles ([see warning](#)).
- Command:** `set <internal variable name> <value>`
- Arguments:** `<internal variable name>`  
Integer - i0 thru i9  
Eeprom val - e0 thru eF  
String - s0 thru s9  
Point Coordinate - p0 thru p9
- Examples:** `set i9 -200`  
Set internal integer variable i9 to negative 200.  
`set s5 "Hello World"`  
Set internal string variable s5 to the string value, "Hello World".

## GET VARIABLE

- Description:** Used to return the value of an internal variable (Integer, String, and Point Coordinate).
- Command:** `get <internal variable name>`
- Arguments:** `<internal variable name>`  
Integer - i0 thru i9  
Eeprom val - e0 thru eF  
String - s0 thru s9  
Point Coordinate - p0 thru p9
- Returns:** `<value><return>`
- Example:** `get i9`  
-200  
Internal Integer variable i9 returns its value, negative 200.  
`get p5`  
20 200  
Internal Point Coordinate variable p5 returns its value, x coordinate, 20 and y coordinate, 200.

## BINARY NOTIFICATION MODE

- Description:** Used to set SLCD5 notification mode to binary or ASCII.
- Due to parsing constraints, it is sometimes useful to have the SLCD5 provide notifications in fixed length binary format instead of variable length ASCII. This command provides the binary option.
- Command:** \*binr <0|1>
- Arguments:**
- 0 Button / Hotspot / Macro notification is standard ASCII as specified in the button, and hotspot, and macro notify commands.
  - 1 Button / Hotspot / Macro notification is in binary format as follows:
- Functionality:**

Standard (ASCII) notification	Binary notification
x<index><return>	X<binary index>
x<index> <Xr> <Yr><return>	Y<index_Byte><Xr_LSByte><Xr_MSBByte> <Yr_LSByte><Yr_MSBByte>
r<index><return>	R<binary index>
s<index> <state><return>	S<binary index><binary state>
m<index><return>	M<binary index>
e<index><return>	E<binary index>
<index> is 1-3 ASCII digits	<binary index> is a single byte

- Returns:**
- on<return>
  - or
  - off<return>

## GET PANEL TYPE

- Description: The unit's firmware is different depending on the panel and inverter it supports, even if the software version is the same. This command displays a human readable string that shows the panel definition used to create the firmware. Note that if the CONFIG.INI specifies a config string, this string will be returned by the \*panel command.
- Command: `*panel`

## CONTROL PORT AUTO SWITCH

- Description: The SLCD5 unit has two serial ports. COM0 is connected to J6 while COM1 is connected to J10. Only one port is active at a time as the unit's control port. However in certain circumstances it is useful to be able to switch ports temporarily.
- Command: This can be done by sending three consecutive special characters to the inactive port. Once this is done, the inactive port will become the active port temporarily.
- The special character is <return> by default. The \*auxEsc command can be used to change the special character.

## SET CONTROL PORT

- Description: Used to set the port used to control the unit. This is stored in non-volatile memory ([see warning](#)) and will be used on power-up. Note that this switches between the MAIN and AUX ports of the PowerCom4 board.
- Command: `*com0main`
- This sets the main port to COM0. When the unit is powered up, COM0 will be sent the '>' prompt.
- Command: `*com1main`
- This sets the main port to COM1. When the unit is powered up, COM1 will be sent the '>' prompt.

## SET PREVIOUS CONTROL PORT

- Description: Used to revert to the previous port after a port autoswitch (three <return> characters on the inactive port).
- Command: `*prevCons`

## SET TYPEMATIC PARAMETERS

Description: Sets the delay and repeat rate for typematic buttons and stores the values in non-volatile memory ([see warning](#)); when given no arguments, returns the current settings. If optional 's' is present, does NOT store values.

Command: `typematic[s] <delay> <repeat>`

Arguments: <delay> is the number of 10's of milliseconds a typematic button must be held down before it starts to repeat. <repeat> is the repeat interval in 10s of milliseconds. Both <delay> and <repeat> must be numbers between 0 and 255 (inclusive).

Example: `typematics 200 50`

This sets the delay to 2 seconds and the repeat rate at 500ms = 2 per second, but does not store the values, making the change temporary.

Example return: `Delay 2000ms, Repeat 500ms<return>`

## SET TOUCH DEBOUNCE

Description: Sets the delay between touch button responses and stores the value in non-volatile memory ([see warning](#)); when given no arguments, returns the current setting. Manufacturing default is 100ms.

Command: `*debounce <delay>`

Return: `Debounce = ###ms<return>`

Arguments: <delay> is the number of milliseconds after a touch is recognized that another touch can be recognized. If no argument is given, the current value is returned.

Example: `*debounce 50`

This sets the delay to 50 milliseconds.

## DEFINE PANEL ORIENTATION

Description: Some LCD panels have hardware signals to flip the display horizontally, vertically, or both. This command allows these signals to be set and stores the value in non-volatile memory ([see warning](#)); when given no arguments, returns the current setting.

Command: `*orient [0|1|2|3]`

Argument: (none) - display current state  
0 - pin 31 low, pin 30 low (pins on DF9-31 connector)  
1 - pin 31 low, pin 30 high (pins on DF9-31 connector)  
2 - pin 31 high, pin 30 low (pins on DF9-31 connector)  
3 - pin 31 high, pin 30 high (pins on DF9-31 connector)

Example: `*orient 2`

This flips an NEC panel 180 degrees compared to the "`*orient 0`" position.

## DEFINE INVERTER CONTROL POLARITY

Description: The inverter power connectors J10, J13 have an "inverter enable" signal. On some inverters, this is high true (high = typically 5V), and on others it is low true. This command sets the enable polarity and stores the value in non-volatile memory ([see warning](#)); when given no arguments, returns the current setting.

Command: `*invEnaHiTrue [0|1]`

Argument: (none) - display current state.  
0 - inverter enable signal will be low for inverter on.  
1 - inverter enable signal will be high for inverter on.

## DEFINE TOUCH SIGNAL ORIENTATION

Description: The touch panel connectors J3, J11, J12 are defined in terms of X/left/right and Y/up/down. A touch panel that has these X and Y reversed (swapped) can be accommodated by using this command, which stores the value in non-volatile memory ([see warning](#)); when given no arguments, returns the current setting.

Command: `*touchSwap [0|1]`

Arguments: (none) - display current state.  
0 - touch signal names are given per J3 / J11 / J12 descriptions.  
1 - touch signals on J3 / J11 / J12 are swapped X and Y.

## DEFINE INVERTER PWM CONTROL

- Description: The backlight inverter is typically either voltage or enable controlled. This corresponds to high or low PWM frequency respectively. If the inverter is voltage controlled the PWM must be high frequency so the analog filter will work. This command sets the value stores it in non-volatile memory ([see warning](#)); when given no arguments, returns the current setting.
- Command: `*pwmIsEnable [0|1]`
- Arguments: (none) - display current state.  
0 - pwm is high frequency for voltage controlled inverters.  
1 - pwm is low frequency for "enable" controlled inverters.

## DEFINE MAX, MIN BRITE

- Description: These commands set the range and polarity for inverter brightness control via J10 / J13, stores the values in non-volatile memory ([see warning](#)); when given no arguments, they return the current setting.
- Command: `*maxBrite <value from 0 to 255>`  
`*minBrite <value from 0 to 255>`
- Arguments: if `maxBrite > minBrite`, the polarity of the brightness signal is positive: higher brightness = higher level.  
  
if `maxBrite < minBrite`, the polarity of the brightness signal is negative: higher brightness = lower level.
- Example: `maxBrite 0`  
`minBrite 230`  
When the backlight brightness command "`xbb 255`" is issued, the brightness control level will be zero. When "`xbb 0`" is issued, the brightness control level will be  $(230/255 * \text{maximum value})$ .

## DEFINE TOUCH ACTION

- Description: The touch screen can have different operating characteristics defined using this command. It stores the value in non-volatile memory ([see warning](#)); when given no arguments, returns the current setting. If optional 's' is present, does NOT store the value.
- Command: `*touchMode[s] [S|L|P|W]`
- Arguments: (none) - display current options.  
S - standard (non of the others).  
L - lockout other touch areas until a touch release (turn off "n-key rollover) .  
P - use touch pressure to validate touch (can eliminate multiple touch aliasing) .  
W - wandering from the initial press will cause the touch to release.
- Example: `*touchModes LPW`
- This sets the touch action to be the most guarded against unintentional presses at the cost of less responsive feel.

## DEFINE TOUCH PARAMETERS

- Description: The touch screen can have different sensitivities defined using this command. These values are dependent on the touch panel used. The command stores the values in non-volatile memory ([see warning](#)); when given no arguments, returns the current settings.
- Command: `*touchParm [<samples> <span>]`
- Arguments: (none) - display current options.  
<samples> - number of touch samples required for a valid touch; the larger the number the less sensitive.  
<span> - allowable range for sample location measurement; the smaller the number the less sensitive.
- Example: `*touchParm 8 12`

## DEFINE TOUCH CALIBRATION TIMEOUT

- Description: The touch screen calibrate command has a timeout when waiting for a touch input. This sets the timeout value and stores it in non-volatile memory ([see warning](#)); when given no arguments, returns the current setting.
- Command: `*tcTimeout [<timeout in seconds>]`
- Example: `*tcTimeout 5`
- This sets the timeout to five seconds.



## DEFINE AUX ESCAPE

- Description: The control port can be selected by sending three consecutive special characters on the inactive port. For example, if COM0 is the active control port, sending three special characters on COM1 will cause COM1 to become the control port. This command sets the special character and stores the value in non-volatile memory ([see warning](#)); when given no arguments, returns the current setting.
- Command: `*auxEsc <hex value of ASCII character>`
- Arguments: (none) - display current escape character.  
0x01 through 0x7e - set escape character.
- Example: `*auxEsc 0x0d`
- This sets the escape character to <return> which is the standard default value.

## DISPLAY CONFIG STRING

- Description: The SLCD5 configuration can be set by a CONFIG.INI file. This file can also define the response of this command. In this way, the host software can verify the SLCD5 configuration.
- Command: `*config`
- Returns: text string that was set by the line:  
`config = "text string"`  
in the [CONFIG.INI](#) file.
- Note: The text string displayed by this command does not have double quotes around it.

## PANEL TIMING ADJUST

- Description: The SLCD5 firmware sets its LCD panel timings to certain default values. These may not work correctly with certain panels. This command can be used to adjust the timings.
- Command: `*paneladj`
- Use: Follow the directions displayed on the terminal emulator. Once the proper timing has been determined, these values can be put in the [CONFIG.INI](#) file for production setting, or stored in non-volatile memory ([see warning](#)).

## EEPROM READ, WRITE

Description: The EEPROM can be read from and written to, at addresses 0x00-0x0F (16 bytes total). When writing values, be aware the EEPROM chip has limited total write cycles ([see warning](#)).

Commands: `*eer <hex address byte>`  
`*eew <hex address byte> <hex value byte>`

Arguments: all arguments are ASCII Hex characters, representing byte values.

Examples: `*eew 0f a5`  
`*eer 0f`

Returns: 0F = A5

## SPEED TEST

Description: Command to determine the execution speed of a given command.

Command: `*speedtest N <command line>`

Arguments: N - number of times to run the command.  
`<command line>` - Command line enclosed in angle brackets. This is done so that quotes (") can be used in the command.

Examples: `*speedtest 100 <t "Hello World" 0 0>`

Returns: `:time/command = 4 ms`

## COPY FLASH to DRAM (SLCD5 PLUS only)

Description: If the flash memory is being used directly as the working set for Bitmaps/macros/fonts (no SD card present), the image load time can be slower than if the working set were in DRAM. The flash is used directly so that boot time from power-on is fast. If the extra speed is needed, this command copies the working set to DRAM. This command may take several seconds to execute depending on the size of the .bin file loaded in flash.

Command: `*flash2ram`

Returns: Standard prompt; error prompt means the flash contents were not valid.

## **LOAD SYSTEM FILE FROM SD CARD INTO FLASH (SLCD5 PLUS only)**

Description: Same as the \*load command except that instead of loading into memory, the file is loaded into flash, and used from there. This can take some time to execute. Status is provided on the board LEDs.

Command: \*loadf or \*LOADF

Returns: status messages as progress, standard or error prompt if command succeeds or fails.

## **CONTROLLER TYPE**

Description: Returns type of controller – SLCD5 or SLCD5 PLUS.

Command: \*ctype

Returns: 0 – SLCD5  
1 – SLCD5 PLUS  
! – error prompt for older firmware; controller type is SLCD5

## 6. Fonts

### 6.1. External Fonts

External fonts allow users to use fonts which are available in MS Windows. These programmable fonts are used often used to get a particular look-and-feel for an application, or to support a non-Latin language for product localization with. These fonts are loaded into the on-board flash memory along with bitmaps and macros. A font is contained in a .SIF extension file. A separate .SIF file is needed for each unique combination of font, size, and attribute (e.g. bold).

The basic steps to creating and using External fonts are:

1. Request a System Independent Font (.SIF) file from REACH Technical Support: Specify Windows font name, size (height in pixels or points), and attributes (bold, italic, both), code set ("ASCII + ISO 8859" (256 characters) or Unicode Character set), or individual characters\*.
2. REACH will generate a .SIF from your specifications
3. Create a Font List (.RFL) file: Create a text file with a ".RFL" file extension. The contents should contain a line for each font in the format:
  - a. "<font\_alias> <filename>.sif <CR>"
  - b. There must be a space character between font alias and the filename.
  - c. The limit for the font alias is eight characters.
  - d. The filename must include "aa2" or "aa4" (case insensitive) if the font uses 2bpp or 4bpp anti-aliasing.
4. Download Font List file: Use the Windows BMPload application under Windows.
5. Use the External font: Use the SET FONT command ("f <font\_alias>") before any text related command.
6. If the character set is non-Latin Unicode, you will need to use the SET UTF8 ENCODING command before displaying any characters.

\*Users which would like a .SIF file with individual characters (rather than a range of Unicode values) should submit a MS Notepad generated text file known as a "Pattern File". To generate this file, copy desired characters into MS Notepad. When complete, use menu option "File->Save As", with Encoding set to Unicode.

### 6.2. Character Set - ISO 8859-1

The ISO 8859-1 character set used by all fonts is as follows. Note that the ASCII character set is the same as the ISO up to Code 127. The ISO set does not define characters 0-31, or 127-159.

The following tables can be used to determine the character code for non-ASCII characters. The standard ASCII set is 0x20 through 0x7E. The extended set is from 0xA0 through 0xFF.

For example, to display the copyright symbol, note that it is hex A9, so the command to display the character is:

t "\xa9"

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0020		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0070	p	q	r	s	t	u	v	w	x	y	z	{		}	~	□

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00A0		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
00B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
00C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
00D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
00E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
00F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

## 7. USING CRC'D COMMANDS

### 7.1. Overview

The SLCD5 can accept a command with a CRC prefix and use it to verify the command was not corrupted in transmission from the host. Once verified, the command is processed in the normal manner, and the SLCD5 responds as expected. If, however, the CRC check fails, the SLCD5 ignores the command and returns a invalid CRC response ('#<return>').

### 7.2. Command Protocol

The format for a CRC'd command is:

```
~<CRC><Command><return>
```

A '~' (tilde) character at the start of the command string signals the SLCD5 that an embedded CRC (4 ASCII-Hex chars, [0-9,a-f,A-F]) will follow the '~' and then the actual SLCD5 command will begin. The CRC is calculated for the SLCD5 command and its <return>, which means a NULL Command (just a <return>) will still have a CRC to validate the <return>.

For example: to send the "s 0 1" command with a CRC, calculate the CRC for the 'C' string "s 0 1\r", which is 0x4050. Send:

```
~4050s 0 1<return>
```

and the SLCD5 will validate the command, execute it, and respond with the '><return>' prompt, indicating success. If the CRC value does not match the string's computed CRC, the '#<return>' prompt is given. If the CRC is correct, but the command has a syntax error, the standard error prompt '!<return>' is given.

### 7.3. Example CRC generation code

Included below is 'C' code for a program that accepts a standard SLCD5 command as input and generates the CRC'd version of the command as output. It includes a CRC generator function that produces CRC's compatible with the SLCD5. The CRC polynomial is CRC-16-IBM.

```

//=====
// Functions for calculating CRC-16-IBM
// (Bisync, Modbus, USB, ANSI X3.28, CRC-16, or CRC-16-ANSI)
//=====

// CRC tables for the CRC16. The polynomial is x^16 + x^15 + x^2 + 1

unsigned char CRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,
0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
} ;

unsigned char CRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
} ;

/*-----< CalcCRC >-----*/
unsigned short CalcCRC(unsigned char *ptr, unsigned int length)
{
    register unsigned char k, crclo, crchi;

    crclo = 0xff;
    crchi = 0xff;

    while (length--)
    {
        k = crclo ^ *ptr++;
        crclo = crchi ^ CRCHi[k];
        crchi = CRCLo[k];
    }
    return (unsigned short)crclo + ((unsigned short)crchi << 8);
}
//=====

```

```

// main()
//=====

static char cmdStr[ 129 ];

// syntax: CmdCrc "SLCD Command Line"
// output: "~HHHSLCD Command Line<CR>"
int main(int argc, char* argv[])
{
    unsigned short crc;

    // must be 1 and only 1 arg:
    if( argc == 2 )
    {
        // copy slcd command to our buffer:
        strcpy( cmdStr, argv[1] );
        // append a <CR>:
        strcat( cmdStr, "\r" );
        // calc the CRC:
        crc = CalcCRC( (unsigned char *)cmdStr, strlen(cmdStr) );
        // show results:
        printf( "    Input: [%s\\r]\\r\\n", argv[1] );
        printf( "    Output: [~%04X%s\\r]\\r\\n", crc, argv[1] );
    }
    else
    {
        printf( "    ERROR: syntax is 'CmdCrc \"slcd cmd\"' (quotes req'd)\\r\\n" );
        return( -1 );
    }
    return 0;
}

```

**Example usages:**

```

C:\CRC>cmdcrc "s 0 1"
Input: [s 0 1\r]
Output: [~4050s 0 1\r]

C:\CRC>cmdcrc "t \"Hello\""
Input: [t "Hello"\r]
Output: [~298Ct "Hello"\r]

```



## 8. COMMUNICATIONS WATCHDOG TIMER

### 8.1. Overview

As of version 1.3.0 and later, the SLCD5 firmware contains a Communications Watchdog Timer feature that can be used to execute a macro with one of four predefined labels when a corresponding communications events is detected:

1. Short Term loss of communication
  - No characters received for X seconds
  - Specified macro executes with label “:st\_down”
2. Communication restored after Short Term loss
  - A character is received after a Short Term loss occurred
  - Specified macro executes with label “:st\_up”
3. Long Term loss of communication
  - No characters received for Y seconds
  - Specified macro executes with label “:lt\_down”
4. Communication restored after Long Term loss
  - A character is received after a Long Term loss occurred
  - Specified macro executes with label “:lt\_up”

The macro to execute and the definitions of Short Term and Long Term communications loss are specified using the ‘\*comwdt’ command, which also enables the feature. Once enabled, it remains active until power is cycled or the SLCD5 is reset.

### 8.2. Using the ‘\*comwdt’ command

Command: \*comwdt <macro> <short> <long>

Arguments:

- |         |   |                                   |
|---------|---|-----------------------------------|
| <macro> | - | name or index of macro to execute |
| <short> | - | Short Term timeout, in seconds    |
| <long>  | - | Long Term timeout, in seconds     |

### 8.3. Example

Below are example macro definitions demonstrating a simple usage of the Communications Watchdog:

```
// macro to enable the watchdog
// short term timeout is 5 seconds, long term is 10 seconds
//
#define enable_wdt
z
f32B
ta CC
t "Communications Timeout Demo\n" 240 25
*comwdt com_wdt 5 10
#end
// macro that gets executed when comwdt times out
// predefined labels are used for the timeout actions:
//
#define com_wdt
//
// the following is a required label, do not change it
:st_down
// short term timeout expired; animate backlight blinking
ani 0 xbbs 127
ani 0 y 250
ani 0 xbbs 255
ani 0 y 250
anie 0
//
// the following is a required label, do not change it
:st_up
// recover from short term timeout; kill the animation and
// restore to full brightness
anix 0
xbb 255
//
// the following is a required label, do not change it
:lt_down
// long term timeout expired; clear screen and warn user
anix 0
xbb 255
z
f32B
ta CC
t "Communications Timeout\n" 240 75
ta CC
t "Please Restart System"
//
// the following is a required label, do not change it
:lt_up
// recover from long term timeout; in a real system the power-on macro or some other
startup
// macro would be run
ta CC
t "Communications Restored" 240 200
#end
```

## 9. WORKING WITH VARIABLES

### 9.1. Overview

In versions prior to 1.2.0, the SLCD5 has supported a limited set of variables with primitive access capabilities. As of version 1.3.0, simple printf() like formatting is supported; this allows using variables in formatted output on the SLCD5 display or in a line of text sent to one of the serial ports.

### 9.2. User Variables

User Variables are available with predefined names which specify their data type, as shown in the table below. They are assigned values using the ["set" command](#), and the host can query their values using the ["get" command](#). Their values can also be used as arguments to commands or macros by enclosing their names inside back-tick characters.

<i>Variable Name</i>	<i>Format Type</i>	<i>Data Type</i>
i0 -- i9	Numeric	32-Bit integer
e0 -- e9	Numeric	8-bit unsigned integer
s0 -- s9	String	Text string, max of 80 chars
p0 -- p9	Numeric	Pair of X and Y Coordinates (16-Bit integers)
p#.x, p#.y	Numeric	X and Y coordinates of point #

### 9.3. System Variables

*System Variables have values that are set by the system and cannot be accessed via the "get" and "set" commands. Their values can be used as arguments to commands or macros by enclosing their names inside back-tick characters.*

<b>Variable Name</b>	<b>Format Type</b>	<b>Data Type</b>
H#	numeric	Height of bitmap # (in pixels)
W#	numeric	Width of bitmap # (in pixels)
V	numeric	Volume setting (0--255)
B	numeric	Brightness setting (0--255)
M	string	Default "mpush" string, max of 80 chars
T	string	5 char string containing SLCDx's temp in degrees C; formatted "%+05.1d", the last 2 chars will always be the decimal point and the tenths digit.
L128 -- L255	numeric	Slider value (16-bit integer)
R#:#	numeric	Random integer in the range #:#
Xa, Ya, Xr, Yr	numeric	Last Touch Coordinates, absolute and relative to hotspot Top Left Corner (16-bit integers)
Xc, Yc, Xm, Ym, Xs, Ys, Xo, Yo	numeric	Absolute X and Y coordinates for center of display, bottom right corner of display, screen cursor as set by the 'sc' command, and origin.
\$BAUD0	numeric	String containing Main Comm port Baud Rate

## 9.4. Formatting Variables

A simple printf()-like output format may be applied to a variable by inserting a format specifier between the 1st back-tick and the variable's name, with a space between them, as shown in the examples below. Two different format specifier syntaxes are supported; one for string variables, and one for numeric variables (see Format Types in preceding tables).

NOTE: macro arguments '0'-'9' must be formatted using the string syntax.

**String syntax:** ``%[-]<width>[.<max>] <varName>``

`[-]` Left justify output text by adding trailing spaces; default is right justification by adding leading spaces

`<width>` Minimum output field width

`<max>` Maximum number of characters from string variable to output

Examples: `set s0 "abcdef"`

Sets string variable s0 to "abcdef"

`get s0`

abcdef

Outputs string variable s0

`out "[%-4.3 s0`]\r"`

[abc ]

Outputs "[", upto 3 chars from s0 in a left justified field of 4 chars, "]", and a return.

`out "[%4.3 s0`]\r"`

[ abc]

Outputs "[", upto 3 chars from s0 in a right justified field of 4 chars, "]", and a return.

`out "[%6 T`]\r"`

[ +33.1]

Outputs "[", the temperature string in a right justified field of 6 chars, "]", and a return.

`out "[%6.3 T`]\r"`

[ +33]

Outputs "[", 3 chars from the temperature string in a right justified field of 6 chars, "]", and a return.

**Numeric syntax: ``%[+|-|0|+0]<width> <varName>``**

- [+] Always include sign of value.
- [-] Left justify output text by adding trailing spaces; default is right justification by adding leading spaces.
- [0] Right justify by adding leading 0's.
- [+0] Always include sign of value AND right justify output by adding leading 0's after the sign.

`<width>` Minimum output field width

Examples: `set i0 123`

Sets integer variable i0 to positive 123

```
get i0
```

```
123
```

Outputs integer variable i0

```
out "[%+06 i0`]\r"
```

```
[+00123]
```

Outputs "[", the value of i0 in a right justified field of 6 chars with the sign of i0 followed by enough leading 0's to fill out the field, "]", and a return.

```
out "[%+6 i0`]\r"
```

```
[ +123]
```

Outputs "[", the value of i0 in a right justified field of 6 chars with the sign of i0 preceded by enough leading spaces to fill out the field, "]", and a return.

```
out "[%-6 i0`]\r"
```

```
[123  ]
```

Outputs "[", the value of i0 in a left justified field of 6 chars with enough trailing spaces to fill out the field, "]", and a return.

```
set p0 12 345
```

```
out "[%06 p0`]\r"
```

```
[000012 000345]
```

Outputs "[", the X and Y values of p0 in right justified fields of 6 chars with enough leading 0's to fill out the fields and a space between fields, "]", and a return.

## 10. USING SIMPLE MATH EXPRESSIONS

### 10.1. Overview

Since version 1.2.0, the SLCD5 has supported a limited simple math capability, using the backtick-escaped format:

``(<argNum>`<op><value>`)`, where:

- `<argNum>` can be '0'-'9', representing macro arg ``0`-`9``
- `<op>` can be '+' or '-' for addition, or subtraction
- `<value>` can be 0-65536

Such expressions could be used only within a macro, and were thus of limited value. Now, in version 1.3.0 and later, simple math expressions can be used with any command that accepts integer values as arguments. When the command executes, the expression will be evaluated and its numeric value will be used. The new format is:

``(<value><op><value>`)`, where:

- `<value>` can be:
  - A positive number between 0 and 6536
  - A backtick-escaped macro argument (``0`-`9``) or the backtick-escaped name of any variable with a numeric format type, as listed in [WORKING WITH VARIABLES](#) (above).
- `<op>` can be one of:
  - '+' for addition
  - '-' for subtraction
  - '\*' for multiplication
  - '/' for division (integer, truncating)
  - '%' for integer modulo

### 10.2. Limitations and requirements

- No nesting of expressions is allowed (to avoid problems with recursion).
- BMPload for the SLCD5, version 2.2.2 or later is required.
- Expressions used in an "ani" command or a "tf" command will be evaluated before the resulting command is stored in the animation buffer.

### 10.3. Examples

Below are some example macro definitions demonstrating valid variants of the new format:

```
// Display a given bitmap centered on the screen:
// -- arg 0 is index of bitmap
// -- arg 1 is its width
// -- arg 2 is its height
//
#define showBitmapCentered
set i0 `(`1`/2)` // divide width by 2
set i0 `(`Xc`-`i0`)` // subtract from Screen Center X coord
set i1 `(`2`/2)` // divide height by 2
set i1 `(`Yc`-`i1`)` // subtract from Screen Center Y coord
xi `0` `i0` `i1` // display the bitmap
#end

// show bitmap 1 in center of screen
//
#define test_sbc
m showBitmapCentered 1 `W1` `H1`
#end

// increment variable i0 each time called:
//
#define inc_i0
set i0 `(`i0`+1)`
#end
```



## **Appendix A - LCD and touch panels compatible with the SLCD5 controller**

### ***A.1 LCD Panels***

The SLCD5 can support a variety of LCD panels. The main issue is cabling and external backlight driver interface. Reach manufactures custom flat flex cabling for interfacing the SLCD5 to Hitachi and other panels that use 40 pin FFC connectors. Reach also supports most panels via ERG inverters and backlight drivers; see [www.ergpower.com](http://www.ergpower.com). Please contact Reach sales for specific requirements.

### ***A.2 Touch Panel***

The on-board touch controller supports 4 wire resistive panels with either 1mm pigtail contact spacing (Fujitsu 4 wire standard), 1.25mm spacing (Kyocera standard), or 0.1” spacing (3M standard).

## Appendix B - Parts and suppliers for SLCD5 controller connections

### **B.1 Connectors and cables for J6, J7, J10, J13**

The board connector is Molex type 53261-XX90 where XX is the number of pins. The mating connector is made of two parts: a receptacle housing and crimp pins. A special tool is needed to make the crimps. Alternatively, custom cables can be purchased. See B.3 for cable vendors.

J6 Receptacle housing Molex P/N 51021-0800

J7 Receptacle housing Molex P/N 51021-1000

J10 Receptacle housing Molex P/N 51021-0400

J13 Receptacle housing Molex P/N 51021-0700

Crimp pins Molex 50079 or 50058

Prototype (small qty) crimp tool Molex 63811-0200

Production crimp tool Molex 63811-0000

All of the above are available from [www.digikey.com](http://www.digikey.com)

### **B.2 Cables for J8**

A DF9-31 to DF9-31 cable is used to connect J8 to panels using this type of connector. This cable is available from:

[www.axoncable.com](http://www.axoncable.com) e.g. P/N FDC31/254AFF03  
DF9-31 to DF9-31 cable 10" female - female same side 1-1 pinout

[www.digikey.com](http://www.digikey.com) e.g. P/N PS563AB0254S-ND  
DF9-31 to DF9-31 cable 10" female - female same side 1-1 pinout

### **B.3 Discrete wire cable vendors**

The cables needed for J6 through J13 can be specified and supplied as assembled cables by: [www.intcomptech.com](http://www.intcomptech.com).

## Appendix C - Ordering information

### **C.1 General information**

See the Reach website at [www.reachtech.com](http://www.reachtech.com) for SLCD5 controller and display model ordering numbers. When ordering either a board or a module, you must also specify a specific two-part revision number. The first number indicates the hardware version and accounts for options such as touch connector configuration. The second number designates the firmware that comes pre-loaded on the board. The firmware revision numbering ensures that production boards have the same firmware as was tested and approved.

### **C.2 Contact Reach directly for specific ordering information.**

Reach Technology, Inc.  
4575 Cushing Parkway  
Fremont, California 94538  
(510) 770-1417

## Appendix D - BMPload program

### D.1 Overview

The BMPload program is used to load bitmaps, optional macros, and optional fonts into the SLCD5 controller. It creates a compressed binary file containing all these elements. This file is either loaded directly into the SLCD5 flash via a serial port, or stored on an SD card that is inserted into the controller. There are different use cases depending on whether the controller is being used for development or as part of the end equipment production. See Section 4.14 for a discussion of the exact sequence power-on loading of the binary file.

#### Small Image Development

If the bitmaps and other files are relatively small, BMPload can be connected to the SLCD5 controller via a serial port, and the binary file directly loaded into the controller flash memory. This is useful for rapid testing of different images and macros.

#### Large Image Development

When the binary file is too large for the serial programming method (loading is too slow), BMPload can store the file on an SD card. This card is then inserted into the SLCD5 controller, and it will be read from the card into DRAM on power-on. This loading process can delay the controller power-on but in most cases this is acceptable for development.

#### Alpha test through Production

In production, the file used in Large Image Development is placed on an SD card along with a file named "FLASH2.INI" whose presence directs the controller to store the binary file into the on-board flash memory. Once written to flash, the SD card does not need to be used, and the power-on to screen active time is short.

See [Appendix J](#) for more details about using the SD card.

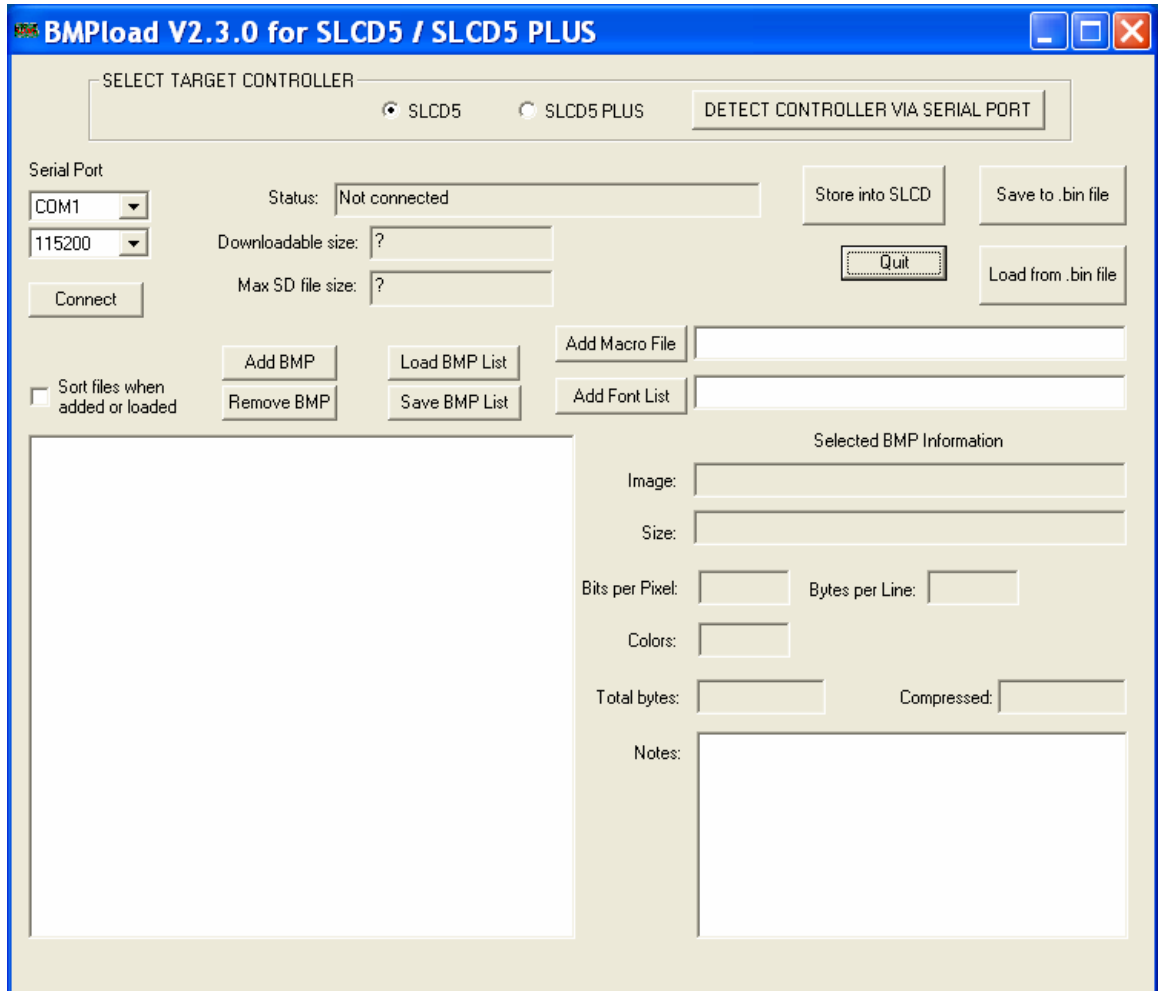
### D.2 Bitmap Format

The BMPload program requires bitmaps to be in 24-bit RGB color. The SLCD5 uses 16-bit color in RGB565 format; the BMPload program does the required translation.

BMPload also compresses images using RLE16 (run length encoding). This is very efficient for control surface images that have horizontally repeated pixels of the same color.

### D.3 Program Operation

BMPLoad runs under Windows XP and Vista. The computer running BMPLoad must have a serial port connected to the SLCD5 board using either the Main or Aux ports. The serial port must not be in use by another program. When it is first run, you will see the following:



You must first set the Target Controller to the appropriate type for your SLCD5 controller board. If you are already connected to the serial port, the type can be automatically detected and set by clicking the "DETECT CONTROLLER VIA SERIAL PORT" button.

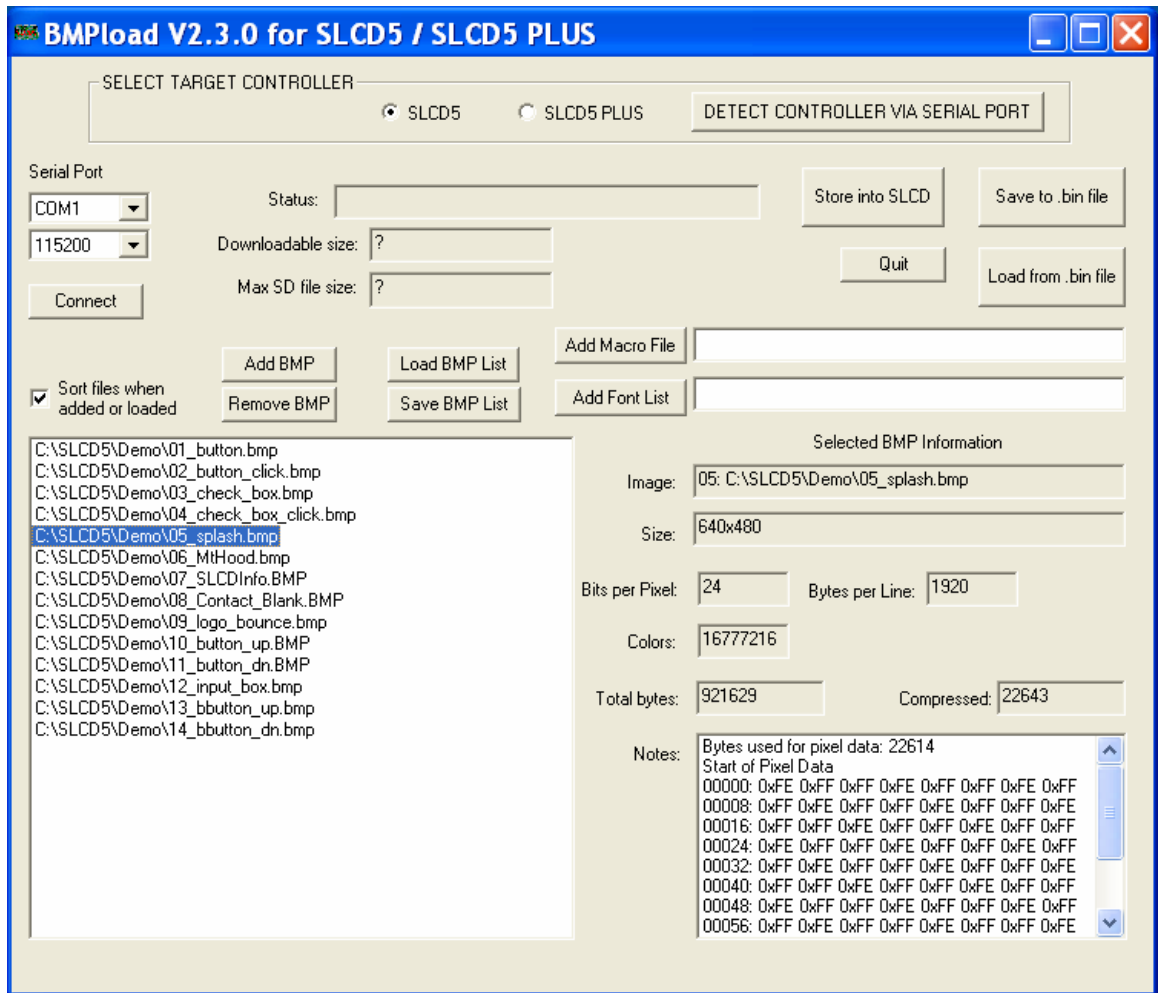
You can now use the "Add BMP" button to add BMP files to the list. Note that the order is important because you use the index number in the DISPLAY BITMAP IMAGE command. The best way to keep this clear is to start all bmp file names with their index number, for example "01\_first\_bitmap.bmp" and then enable the "Sort files when added or loaded" checkbox.

A list file (.lst) is a better way to load all the image files used for the interface. It also keeps the file names in the correct order. The list file is a standard text file; the following list was

used for the screen shot that follows. Note that since the full path name is not specified, the files are assumed to be in the same directory as the list file.

```
01_button.bmp
02_button_click.bmp
03_check_box.bmp
04_check_box_click.bmp
05_splash.bmp
06_MtHood.bmp
07_SLCDInfo.BMP
08_Contact_Blank.BMP
09_logo_bounce.bmp
10_button_up.BMP
11_button_dn.BMP
12_input_box.bmp
13_bbutton_up.bmp
14_bbutton_dn.bmp
```

Note that the full path name is displayed in the file list, but does not need to be in the list file. After the list has been loaded, this is what the program looks like. Note that when a bitmap is selected, the compression and size information is shown on the right.



Use the "Add Macro File" button to add a macro file to the binary file in addition to bitmaps.

Once you have added the bitmaps, use the "Save to .bin File" button to save them to the SD card. The file name must be 8 characters, start with "slcd" and have the extension ".bin".

Alternatively, you can use the "Store into SLCD" to write the binary file directly into the SLCD5 flash memory via the serial port. This can be much slower than using the SD card.

#### D.4 Connecting via Serial Port

To connect to the SLCD5 via a serial port, select the port and / or baud rate. If the specified port is in use by another program, you will see something like this (COM1 will be replaced by the COM port you are trying to open)



This means that the program was unable to open the specified port. This is due to another program such as Teraterm being open and connected to it. Shut down or disconnect any other serial programs, and click OK.

You may see the following message instead:



This means that the serial connection between the PC and the SLCD5 is not working. Check the cables.



## **D.5 BMPload Speed Issues**

For fast serial loading, speeds of up to 460800 baud are supported. For reliable communication at this speed, use a USB-serial adapter with the electronics in the DB9 connector.

The BMPload program will work with any type of serial port. However, some USB-to-serial converters have high overhead for the small transaction sizes used by the BMPload program. If you are seeing slow programming times with a USB converter, try using a standard serial port, or use the “Save to .bin File” method.

If you are using an FTDI adapter ([www.ftdichip.com](http://www.ftdichip.com)), you can set the latency timer to a smaller than default value to speed up transfers. To do this on an XP machine, open the Control Panel, open System, select Hardware, Device Manager, Ports, Serial port (select the USB adapter), Properties, Port Settings, Advanced, Latency Timer.

## Appendix E – Macro commands and file format

### E.1 Introduction and limitations

Macros have two main purposes.

1. They allow a series of commands to be invoked by a single command. This can speed up the display by reducing communication overhead. It also reduces the space needed to store commands on the host processor.
2. They can be linked to buttons and other graphical objects so that by pushing a button, a macro can be invoked for example to draw a new screen. This is useful to keep the overhead on the processor low and provide fast response for users.

Macros can have parameters associated with them. This allows a general purpose macro to be used in different ways. For example, a macro can create a numeric keypad and the parameters can specify where to draw the keypad on the screen. This reduces hard coding of graphical elements and promotes reuse of screen elements.

These are the limits on the macro commands and their arguments:

- MAXIMUM NUMBER OF MACROS = 254  
A macro can use an unlimited number of labels, which is an effective way to work around this limit.
- MAXIMUM MACRO NAME LENGTH = 64 chars
- MAXIMUM CALL DEPTH = 4  
A macro can call another macro, but only to a depth of 4.
- MAXIMUM ARGUMENTS PER MACRO = 10
- MAXIMUM BYTES PER ARGUMENT is only limited by the total command line length (max 128)
- MAXIMUM TOTAL STORED ARGUMENTS = 100

### E.2 Macro File Format

The macro file is an ASCII text file and can be generated by Windows applications such as Notepad. There are two macro definition versions to choose from when creating a macro file. The original version, version 1, takes two arguments <text\_name> and <number>. This version requires that all macros be listed in numerical order starting at 1 and incrementing by 1. It has the disadvantage that editing a macro file can be cumbersome because you have to keep track of macro numbers.

Version 2 takes only the <text\_name> argument. When using version 2 each macro definition is assigned a number based on the order in which it appears, starting with 1. This

way, when using functions that refer to macros, the <text\_name> can be used to reference them.

The BMPload program generates a header file in the same folder with the same name as the macro file but with extension '.h'. These header files list all the macro defines and display every macro name with its assigned number. This header file can be used as a 'C' include file in the user's microcontroller program.

The format for each macro in version 1 is as follows:

```
#define <text_name> <number>
(one or more command lines)
#end
```

The format for each macro in version 2 is as follows:

```
#define <text_name>
(one or more command lines)
#end
```

The <text\_name> is an identifier that follows 'C' language conventions. In version 2 the name can also be used instead of the macro number when using a command that references a macro, such as the command that assigns a macro to a button.

All macro names must start with an alphabetical letter or an underscore but thereafter can also contain numbers.

In version 1 the <number> argument must be 1 for the first macro, 2 for the second, and so on. The macros must be listed in increasing contiguous index order.

Comments are ignored. Comments are text starting with the "//" string. All lines outside of a "#define...#end" pair are treated as comments.

Version 2 referencing examples:

```
#define example_a
m example_b
#end

#define example_b
*PONMAC example_a
#end
```

Also with BMPload version 2.2 or higher it is acceptable to indent lines.

### E.3 Macro Parameters (Arguments)

Macros can be parameterized by using the special escape sequences ``0`` thru ``9`` in the command lines. These are replaced at execution time by the arguments supplied by the command that invoked the macro. The combined total length of all macro arguments for a macro call is 128 characters (command line length) minus the character length of the macro name or number plus spaces, and delimiters (ex. double quotes). Note the special escape sequence delimiter character ```` has the ASCII value 96 decimal, 60 hexadecimal.

Parameterized macro example:

```
#define argExample
t "`0`" `1` `2`
#end
```

The following command uses this macro to display the text "Hello" at location x=10, y=20:

```
>m argExample Hello 10 20
```

### E.4 Assigning macros to buttons

The [Touch Macro Assign](#) command and variants can be used to automatically run a macro when a button is pressed or released. Note that macros that execute this way (as a result of touching a graphic object with an associated macro) will cause any currently executing macro to quit running. This also halts any macros that called the currently running macro.

### E.5 Special macro commands, and macro labels

#### Memory commands

Memory commands were added to implement the keyboard in the demo macros that come with the SLCD5 kits. These allow a character string to be saved and manipulated. The character string is accessed as a special macro parameter.

The commands are:

```
mpush [index] "<string>" [max]
```

If [index] is 0-9, <string> will be appended to the corresponding string var, s0-s9; otherwise, <string> will be appended to memory variable ``M``; length is limited to 80, or [max] if given (must be between 1 and 80).

Example: `mpush 1 "0" 3` appends "0" to ``s1`` variable, unless length is already 3.

```
mpop [index] <number>
```

If [index] is 0-9, removes <number> of characters specified from the end of the corresponding string var, s0-s9; otherwise, removes from the end of memory variable ``M``. If <number> is -1, all characters are removed.

Example: `mpop 1 3` removes 3 chars from end of ``s1``.

## Special Arguments

The macro system recognizes other symbols in the parameter escape format – enclosed in back tick marks. These are as follows.

### Simple Math on integer arguments

``(`0`+ 1)``

This is replaced by the value of the macro's first argument plus 1. Simple math supports addition or subtraction, *but only on arguments with an integer value; ie: "abc" or "1.2" would cause an error.*

### Memory variable

``M``

This is replaced by the string stored by the mpush command.

### Integer variable

``i0` thru `i9``

This is replaced by a 32-bit signed integer. See the [SET VARIABLE](#) command.

### String variable

``s0` thru `s9``

This is replaced by a character string with a maximum length of 80 characters. See the [SET VARIABLE](#) command.

### Point Coordinate variable

``p0` thru `p9``

This is replaced by the string representing a coordinate of a point on the screen. The format is “<x coordinate value><space> <y coordinate value>”. See the [SET VARIABLE](#) command.

### Slider value

``L<index>``

This is replaced by the value of the slider defined by <index>.

### Random number

``R<lo>:<hi>``

This is replaced by a random number in the range <lo> to <hi>.

## Repeat command

A special macro directive allows a macro to repeat execution. The directive must be at the start of a line and is:

```
:repeat
```

When the macro processor reads this line, the macro will begin execution again at the first line of the macro. NOTE: An escape character (hex 1B) followed by a <return> received from the serial port will halt a looping macro.

## Labels

A special directive can be used to identify a location within a macro in order to selectively execute specific command lines when the macro is called with the optional label identifier. A label consists of a colon (':') followed by a maximum of 32 alphanumeric characters (label name). The first character of a label name must not be numeric. The line placement of the label must begin in column 1 of the line (the colon in column 1).

Example:

```
:beep
```

A special label, “:default”, can be used to execute commands when the given label identifier is missing or does not match any other labels within the macro. This label must always be the last label in a macro.

Labels are invoked by calling the macro in the normal fashion, but including the colon and label name directly after the macro name/number in a macro call. For example say when want to invoke label “attach” in macro number 8. The command would be:

```
m 8:attach (invoking macro by number)
or, if its name is "button_action":
m button_action:attach (invoking macro by name)
```

The format of a macro label invocation is: “m <macro name/number>:<label name>”.

The execution flow of a macro invoked with an optional label starts with the “common code area”. The common code area is a new feature with labels. The common code area is all command lines in a macro after the macro definition line (#define) up to the first label. So, first the common code area command lines are executed, then execution starts with the line after the matching label, and ends with the next label. Below are examples of a macro that uses labels. Command lines executed are in **bold**.

### Example of Macro call with label

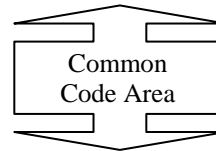
```
m 8:attach (user calls macro number 8 with label
":attach")
```

```
#define create_button /* (command lines below are
always executed)
```

```
s 333 CCC
```

```
f 24
```

```
t "Calling create_button" 200 10
```



```
:attach (command lines below this matching
label are processed)
```

```
t "Attaching button 1 to macro" 200 30
```

```
xa 1 p 3 0
```

```
:define (execution stops here)
```

```
t "Defining button" 200 50
```

```
bd 1 0 32 1 "INCREASE" 9 5 10 11
```

```
#end */
```

### Example of Macro call without a label

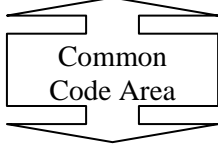
```
m 8 (user calls macro number 8 without any label)

#define create_button /* (command lines below are
always executed)
s 333 CCC
f 24
t "Calling create_button" 200 10

:attach (command lines below are not executed
since there is no matching label)
t "Attaching button 1 to macro" 200 30
xa 1 p 3 0

:define (command lines below are not executed
since there is no matching label)
t "Defining button" 200 50
bd 1 0 32 1 "INCREASE" 9 5 10 11

#end */
```



Error conditions for a label include:

- Label name is not found.
- Use of a label that results in no commands line being executed.

In both these cases, an error message is transmitted to the user.

### E.5 Changing the firmware power-on baud rate

The following is an example of how to set the power-on baud rate. Create and load the following macro file:

```
#define pon_mac 1
/* (start comment out contents)...
// set baud rate
baud 9600
#end */
```

Now, connect to the SLCD5 and run the command:

```
*PONMAC 1
```

Now cycle power to the SLCD5, and the initial baud rate will be 9600 baud.

### E.6 Changing the bootloader and firmware power-on baud rate

The power on macro can change the firmware baud rate, but the bootloader will start at 115200

baud. To set the baud rate for the bootloader as well, use the "bauds" command.



## Appendix F – Troubleshooting

### ***F.1 Touch unreliable or non-operative***

If the touch screen is unreliable or non-operative, do the following:

1. Make sure the metal shell of the display is connected to one of the SLCD5 mounting holes. This is the same as saying that the display case should be grounded to SLCD5 ground. Note that this only applies to displays with CCFL backlights, not to LED backlights.
2. Run the TOUCH CALIBRATE command, "tc". This will reset the calibration values and allow you to recalibrate the touch screen.

If after doing this the touch is still non-operative, check the touch connection into the SLCD5 board. Some touch panels use conductive ink that can be easily scraped off by too many or incorrect connector insertion cycles. If there are holes you can see through on the touch connector end where it plugs into the SLCD5 connector this is the problem. To determine the accuracy and sensitivity of the touch, you can use the "touch debug" command as follows:

```
*debug 1
```

This puts an "X" on the screen whenever a valid touch is recognized. To turn off, use:

```
*debug 0
```

## Appendix G – Evaluation / Demo Kit Tutorial

### G.1 Self-running demonstration

The kit comes shipped with a demo installed. This tutorial will uninstall the demo. To run the demo afterwards, place the supplied SLCD.BIN and RUNDEMO.INI files in the root directory of an empty SD card, install the card, and reset the board or cycle power. This will run the demo. Note that it may be slow to load because it is big, but in production on the SLCD5 PLUS, it will be loaded into flash for fast startup.

### G.2 Connection and control via PC

This section describes how to connect to and control the SLCD5 from a PC type computer. Any other computer (Mac / Unix / Linux) can be used instead with analogous procedures.

The two DB9 serial ports (Main and Aux) on the PowerCom4 board are wired to be compatible with the PC 9 pin serial standard. You need a DB9 female to DB9 male 1-1 serial cable to connect to the PC serial port. Alternatively if you have a USB-serial adapter you can plug the adapter directly onto the PowerCom4 board. The Main port should be used for initial communications with the host PC.

The PC needs to run a serial terminal emulator. Options are Realterm or Teraterm, both are open source. Assuming Teraterm, once installed do the following:

Setup -> Serial Port -> Baud Rate 115200, Flow control Xon/Xoff

Setup -> Terminal -> Local Echo checked, Newline Receive: CR+LF

Setup -> Save Setup -> Save (replaces default setup file)

In the case of a USB-serial adapter, you may have to open the Control Panel -> System -> Hardware -> Device Manager -> Ports to determine which COM port is mapped to the USB adapter. With FTDI adapters, the device Properties -> Port Settings -> Advanced allows you to assign a specific COM port to the adapter.

Once this is done, open the terminal emulator, connect to the Main DB9 port, and hit enter. You should see a '>' prompt come back. Now, type 'z' followed by the enter key. The display should clear as a result. You should also see the 'z' letter you typed and the '>' prompt. You are successfully controlling the SLCD5 now.

### G.3 Simple commands

This section presents some simple commands that illustrate some of the SLCD5 capabilities.

Type in the line(s) as shown in `courier` typeface followed by the enter key. [Note: to minimize typing, you can use the "Text select tool" in Acrobat Reader to select each line, right click to copy, then paste into the terminal screen]

Clear the screen:

```
z
```

Type Hello World in a 24 point bold font starting at pixel location x=100, y=110:

```
f 24B // set font command
t "Hello World" 100 110 // draw text command
```

Same as above, but with yellow text on a blue background:

```
z
f 24B
s 16 2 // set foreground and background color
t "Hello World" 100 110
```

Restore default colors (black text, white background) and clear screen"

```
s 0 1
z
```

Create a solid pure blue rectangle with corners (140, 200) and (160, 250)

```
r 140 100 160 250 1 0000FF
```

Define momentary pushbutton #1 named "Test" at location (100,50) which is the top left corner of the button, use default (internal) bitmaps for the button, and send notification when both pressed and released:

```
z
f 16B
bdc 1 150 110 5 "Test" // button define command
//press button on screen to see notification messages)
```

Create types of hotspot

```
z
p 5 // set pixel width 5 for rectangle command
r 100 200 150 250
x 128 100 200 150 250 // standard type
//press rectangle on screen to see notification messages
r 200 250 300 300
xst 129 200 250 300 300 // invisible, typematic, only first touch beeps
```

## G.4 Bitmaps

Note: If you have another serial port available, hook it up to the AUX DB9 so that bitmaps and macros can be downloaded without having to disconnect the terminal emulator. Otherwise, when using BMPload, make sure the terminal emulator is disconnected.

Start the BMPload program. Select the controller type or use the "DETECT CONTROLLER" button. Click "Load BMP list", and look in the supplied CD for the Demo directory containing bitmaps (.BMP) and list files (.lst) for the screen size you are using. For the 7" screen, that is WVGA. Click on the demo.lst and a set of bitmap files will

appear in the main window. Note that the files are numbered; this is useful because bitmaps are referred to by load order, so the number helps identify the bitmaps.

Clicking on any bitmap file name will display statistics for the bitmap. You may have to scroll horizontally to see the bitmap as full path names are shown. For this next tutorial section, it is assumed that image #25 is 25\_LED OFF.unc.bmp and 26 is 26\_LED ON.unc.bmp.

Make sure the Serial port is configured to the correct speed and port. Click on "Connect" to test the connection. If connected, click on "Store into SLCD" and it should download the bitmaps to the controller. If the download is less than 8KB/sec, and you are using a USB-serial adapter, you may have to reduce the latency; see Appendix D, BMPload Speed Issues. Note that serial download is suitable for small bitmaps and for larger file collections you need to use the SD card method described in the Fonts Section below.

Once loaded, use Teraterm to connect to the controller. Do the following:

Display the bitmaps loaded in the controller:

```
ls
```

Display a bitmap by number:

```
xi 5 0 0 // display bitmap #6 with top right corner at 0,0
```

Use "LED" bitmaps for the two states of a latching button with index #2:

```
z
```

```
bdc 2 100 200 2 " " 25 26
```

```
// touch red dot to see touch notification with latched state
```

## **G.5 Macros**

The easiest way to develop and investigate the uses of macros is to have a small set of bitmap images and a small macro file. See the demo macro file macros.txt for examples of macros.

## **G.6 Fonts**

See Section 6 for a discussion of font lists and files. This tutorial will also demonstrate how to use an SD card with BMPload. For conversion between Unicode and UTF-8, see <http://www.utf8-chartable.de/>

Restart BMPload, and select "Add Font List", and open the fonts.rfl file from the same directory as the bitmaps used above. Then insert a blank SD card into the PC and hit the "Save to .bin file" button and save the file as slcd.bin on the SD card root directory. Then exit the BMPload program and eject the SD card via the File Explorer right click menu. Install the card into the SLCD5 and reset the board. Then do the following:

List available fonts:

```
f?
```

```
// font names are listed; last one is CH48 which was just downloaded
```

Set font and display

```
z
```

```
utf8 on           // enable UTF-8 to address more than 256 characters
```

```
f 16B
```

```
o 100 50         // use an offset for the text
```

```
t "UNICODE character 4E1C in 48 point is:\n"
```

```
f CH48           // select the downloaded font (alias specified in fonts.rfl)
```

```
t "\xe4\xb8\x9c" // use escapes to send hex in ASCII
```

(the above can be sent with the UTF-8 sent as 8 bit bytes instead of escape codes)

## **G.7 Attaching to the embedded controller**

The PC connection method described above is useful for trying out the development kit. After that, the kit should be attached to the OEM's embedded microcontroller for actual GUI development and testing. Attaching one port (MAIN) to the microcontroller, and the other (AUX) to the micro allows either the terminal emulator or the BMPload program to share control with the micro. This is useful when a series of commands is to be tested and it is faster to test using a terminal than writing and compiling microcontroller code. To switch between the micro being the active host and the PC being the active host, either one sends three <return> characters in a row. BMPload does this and then switches the port back so the microcontroller does not have to reestablish its port as the control port. It can also be done at the terminal via the \*prevCons command.

This tutorial assumes the host microcontroller has an RS-232 port. This is often the case with microcontroller development kits.

Attach the micro's 232 port to the Main SLCD5 port; typically via a serial crossover cable. To verify, look at the voltages on pins 2 and 3 of each DB9; connect one side's pin X at a negative voltage to the other side's pin Y of no voltage and vice versa.

Reset the SLCD5 via the yellow button on the PowerCom4 triangle board. Write code on the micro to poll the serial port waiting for the '>' <return> prompt. when it gets this, have the micro send commands such as described above and verify that they work as expected.

## Appendix H – Working with bitmaps

### H.1 Creating bitmaps

Bitmaps are used to create the visual elements of the user interface. These include buttons, tabbed folders, and data entry and display areas. Most interface styles implemented on Microsoft Windows can be duplicated on the SLCD5. The way to do this is to create or capture the visual element and create the desired layout.

The popular programs used to create bitmaps (.BMP files) are Adobe Photoshop, and the open source program, GIMP.

To capture any visual element on the PC screen, hit the "PrintScreen" button on the PC's keyboard. This captures the screen to the clipboard. Then open the image editing program, open a new window, and paste the screen to that window. The desired elements can then be copied and saved.

Note that bitmaps must be saved in 24-bit RGB color mode.

The basic algorithm for converting between Truecolor (24-bit BMP file) to Highcolor (16-bit LCD screen) is below:

1. Divide [Blue and Red value] by 8 (ignore remainder)
2. Divide [Green value] by 4 (ignore remainder)
3. Add [Red value times 2048] plus [Green value times 32] plus [Blue value] (Highcolor value)

### H.2 High Color

The SLCD5 supports high color depth as standard. The BMPload program converts 24-bit BMPs into the RGB565 physical format used by the controller. In this format, 5 bits of color are used for RED and BLUE and 6 bits for GREEN.

The basic algorithm for converting between Truecolor (24-bit BMP file) to Highcolor (16-bit LCD screen) is below:

1. Divide [Blue and Red value] by 8 (ignore remainder)
2. Divide [Green value] by 4 (ignore remainder)
3. Add [Red value times 2048] plus [Green value times 32] plus [Blue value] (Highcolor value)

Users can read the stored Highcolor value by using the [PIXEL READ](#) command.

### H.3 Gradients

Many modern-looking graphic designs use 24-bit color gradients. These can end up with a "banded" look when converted to 16-bit color. To eliminate this effect, use a spatial diffusion converter (dither) to limit the number of colors to 16-bit while diffusing the step

changes spatially. This should be done after the image is rendered for use. A Photoshop plugin called DepthDither can be used to perform this function.

#### **H.4 Transparency**

High Color depth transparent bitmaps are supported. To make a bitmap transparent, select a transparency color. It must have an exact RGB 565 equivalent, e.g. solid red 0xF800 (top 5 bits), solid green 0x07E0 (middle 6 bits), or solid blue 0x001F (lower 5 bits). Then have the bitmap file name include the string ".unc.trXXXX ", where XXXX is the 16-bit RGB transparent color value. For example, if solid red is transparent, have the filename include the form above, such as "01\_my\_bitmap.unc.trf800.bmp". Note, the ".unc" indicates that the bitmap is uncompressed, which is required for transparency. When this bitmap is displayed, the transparent color will not be displayed.

## Appendix J – SD card support features

### J.1 Overview

The SLCD5 board contains an SD card slot. This slot can be replicated in a more convenient location using connector J14. See Appendix K for details.

The SD card be used to do the following:

- Initialize non-volatile (EEPROM) system variables via config.ini file (see Section 2, System Configuration).
- Load new firmware into the SLCD5 controller.
- Hold a BMPload binary file (bitmaps, macros, fonts) to be used as the working set of images, macros, and fonts
- Update the on-board flash with the above binary file
- Save screen shots.
- Run demo macro on startup.
- Hold multiple BMPload binary files (working sets) to be loaded as needed
- Hold individual image files of type .jpg, .gif, and .bmp

NOTE: The SD card must be 2GB or less in size and formatted as FAT (also known as FAT16) to work properly with the SLCD5. **Do not use FAT32.** Some SD cards must be reformatted before use. Reach recommends the SanDisk brand of SD card.

### J.2 Firmware Upgrade

On power-on, the SLCD5 boot loader program checks if an SD card is present, and if so, looks in the root directory for a file with the name:

SLCD????.ELF

where '?' is any character. If it finds this file, it compares the file name with the name of the last loaded firmware stored in non-volatile memory (EEPROM). If they are different, it initiates a firmware update. This process may take a minute or two, during which time the LED D2 flashes, and progress messages appear on COM0 (115200 baud). Once the firmware has been upgraded it starts running.

To force an update (ignore the stored name), place a non-null text file called "LOADFW.INI" in the root directory.



### **J.3 Holding binary working set file**

When the SLCD5 controller powers on, it looks to load a binary file (working set of images, macros, and fonts). If an SD card is present, and has a file named SLCD\*.BIN, this file is used. It is loaded into DRAM because loading images on the fly from SD is too slow. Loading into DRAM provides the fastest display time for images, but this loading can delay the power-on of the display. This can be eliminated by writing the binary file to on-board flash.

### **J.4 Update binary file in Flash**

In order to speed up the power-on display time, the binary file can be loaded into on-board flash. To do this, include a non-null text file called "FLASH2.INI" in the SD card root directory. When the SLCD5 controller powers on, if an SD card is present, and has a file named SLCD\*.BIN, and the FLASH2.INI is present, the binary file is directly loaded into flash memory and used from there. This may take some time; the main serial port will display progress dots and the on-board LED will flash.

### **J.5 Screen Shot Saving**

Users can save the current contents of the entire screen on a file on an inserted SD Card.

Refer to the "

#### **CHANGE SD CARD DIRECTORY**

Description: Changes the current working directory of the installed SD card. This can be used in conjunction with the xif and \*LOAD commands.

Command: \*cd <directory name>

Note: <directory name> follows the DOS 8.3 naming convention, so the directory name is limited to 8 characters.

Example: \*cd /Images

## LIST SD CARD DIRECTORY

Description: Lists the contents of the current working directory of the installed SD card. Directories are listed with the "/" char following their name.

Command: `*ls`

Note: Listed files and directories will follow the DOS 8.3 naming convention.

Example: `*ls`

```
RUNDEMO . INI
RT2_DEMO/
IKB_DEMO/
GFX_DEMO/
SLCD5PWV . BIN
SPLASH . BMP
```

SAVE SCREEN SHOT TO SD CARD” command for details.

### J.6 Run demo macro

A special file, rundemo.ini can be used to launch a macro on power-on. If an SD card contains both a bitmap/macro binary file (e.g. SLCD.BIN) and this special file, a demo can be run without disturbing the binary file stored in flash.

The rundemo.ini file only has two options: verbose mode and the number of the macro to run. Verbose mode is the same as for the config.ini file (See Section 2). To specify the macro to run, use the syntax below.

```
demomac = <macro number>
```

example:

```
demomac = 1
```

### J.7 Multiple Binary Files

In some cases it is useful to be able to switch the “working set” of images, macros, and fonts. This could be done to support different GUI styles (skins) or languages. This can be done by storing the binary files in different root directories on the SD card. The command “\*sdload <directory name>” can be used to load a different binary file. See Section 5.

### J.8 Separate image files

The SD card can also be used to hold individual images as .bmp, .jpg, or .gif formatted files. This may be useful if for instance a users manual was to be stored and displayed page by page. The “xif” command is used to display these files. The “\*cd” command changes

the current directory on the SD card so that different groups of images can be used. See Section 5.

## Appendix K – SD CARD REMOTE

### **K.1 Overview**

The SLCD5 has an on-board SD card slot. It may be more convenient to place this SD card connector elsewhere; in this case, connector J14 can be used to cable to a small application-specific board containing a remote SD card slot.

To do this, refer to the schematic in Section K.2.

The LED output is used for external status and mirrors the SD card activity. A maximum drive of 10mA is provided, at Max 3.4V. So for a green LED with Vf of 2.1V, the resistor value would be

$$(3.4-2.1) / 0.01 = 130 \text{ ohms.}$$

## K.2 Schematic

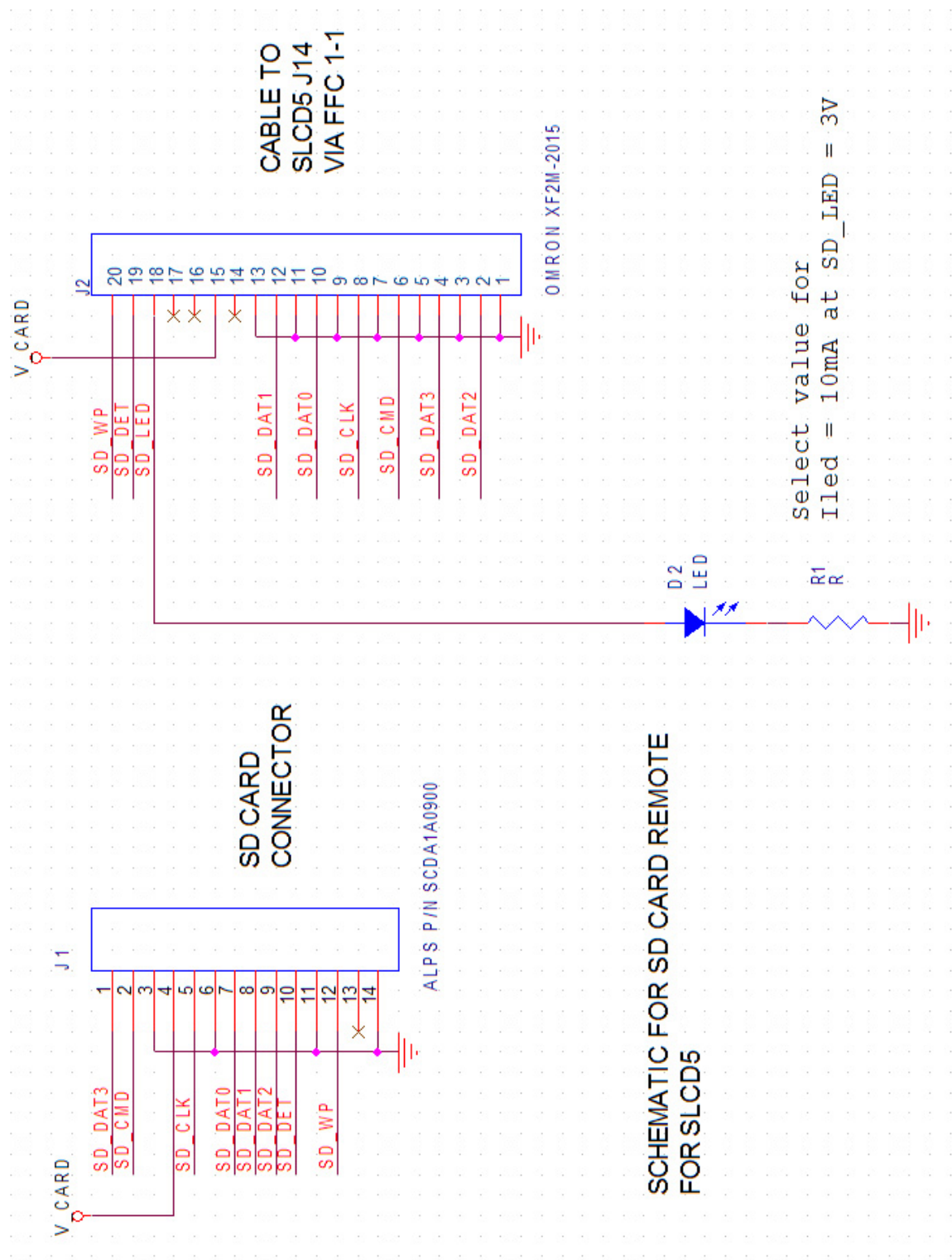


Figure 8: SD Card Schematic

## Appendix L – Previous Versions

This Appendix details the changes made from previous versions to the current version of the Base Board for both the SLCD and SLCD Plus.

### L.1 Rev A vs. Rev B board Fab Rev A vs. Rev B board Fab

The board Fab is located on the back side of the board in the metal layer. The Base SLCD5 board Rev A and Rev B have the following differences. From the component side, the Rev A board can be identified by Jumper JP2 located in one corner that is not present on Rev B. From the back side, the board revision is etched in the trace layer.

Rev A	Rev B	Explanation
JP2 present	JP2 not present	On Rev A, when the board is powered with 5V, the jumper should be installed. When powered with 12V, the jumper must be removed or the speaker will be damaged.
R8 not present	R8 present	On Rev B, the jumper function is not needed as long as the backlight inverter is powered from >7V. Otherwise, R8 must be installed.
	R6 option	R6 normally is a zero ohm resistor and allows the *orient command to control the LCD connectors J8, J9 pin 30 (R/L). Some panels such as the NEC 6.5" need this to be a true no connect, and for these panels R6 is removed.
	R7	Resistor added to keep backlight off during board power-on. Only applicable for inverters using pin 7 of J13.
	C26-C29	Capacitors added for noise reduction on touch screen.

### L.2 Rev B vs. Rev D board Fab

Rev D incorporates the following changes from Rev B.

Rev B	Rev D	Explanation
J5	J5	J5 was I2C expansion on Rev B, and is external beeper / speaker on Rev D.
J3	J3	J3 was moved to be closer to the board edge
	JP1	JP1 added; if inserted, the SD card power is always on like Rev B. If not, the processor controls the SD card power.

## Appendix M - Software Manual Change History

Date	Changes
10/17/08	Modified to clarify use of RUNDEMO.INI file and associated keywords.
10/24/08	Added "System Software Boot Process" section and updated other flowcharts.
11/04/08	Added macro label description in Section 5 and <a href="#">Appendix E</a> .
11/24/08	Added "SAVE SCREEN SHOT TO SD CARD" command.
11/25/08	Added System Software Boot Algorithm, and removed existing related flowcharts for clarity.
12/15/08	Clarified binary filename for "LOAD BITMAP / MACRO FILE" command.
03/09/09	Removed extra verbiage for "J7 10 Pin Molex 53261-1090 for 3.3V CMOS COM1 Communications" table asterisk note.
3/24/2009	Section 2.5 clarified ("J4 - Power (if separate power input is desired)": Noted that Pin 1 on J4 also powers speaker for customer clarity.
3/26/2009	Added Feature 82 (Load binary file from SD Card Directory).
4/7/2009	Added comments on 24- to 16-bit conversion in "Appendix H – Working with bitmaps".
7/23/2009	Added METER DEFINE and METER VALUE. Appendix section on Screen Shot information had some formatting errors, and was made brief.
8/12/2009	Changed corporate address.
8/17/2009	Added "LCDshiftclock" to CONFIG.INI options.
8/18/2009	Added improvements to DISABLE and ENABLE TOUCH.
1/07/2010	<ul style="list-style-type: none"> <li>• Feature 87 (Bitmap translation needed (2-bit to High Color))</li> <li>• Feature 142 (UTF8 support for UNICODE fonts)</li> <li>• Feature 133 (Macros support up to ten parameters)</li> <li>• Feature 132 (Macro call by name). Macros can also use a name.</li> <li>• Feature 151 (Incoming CRC). Commands sent to the SLCD5 can use a CRC.</li> <li>• Feature 128 (Add index parameter to listing commands).</li> <li>• Feature 127 (Macro Save State and Restore State commands macro depth aware). These commands were added to the manual.</li> <li>• Feature 109 (Version string needs additional build information).</li> <li>• Feature 55 (Ability to remember serial baud rates changes).</li> <li>• Defect 115 (SLIDER DEFINE continuous touch has no impact)</li> <li>• Defect 187 (Transparent bitmaps not documented and look icky....)</li> <li>• Defect 219 (DISABLE/ENABLE TOUCH does not document range optional parameters)</li> <li>• Defect 231 (Section missing on how to use external fonts).</li> <li>• Defect 241 (Internal Arguments description valid only within macro.</li> <li>• Defect 242 (DRAW RECTANGLE doesn't do what manual says).</li> </ul>
1/28/2010	Feature 152 (Text Box commands)

2/25/2010	Defect 321 (CRC generation example code uses wrong CRC polynomial)
3/09/2010	Defect 315 (Need command and config variable to support selecting sound output)
5/10/2010	New Release, version 1.2.9
1/20/2011	New Release, firmware version 1.3.0
2/16/2011	Fixed a few command description errors, examples, and pagination
4/18/2011	Added Chart commands 'cc', 'cr', 'cdp', 'cdbc', 'cdbb', 'cdrm', 'cddm'
8/4/2011	Added missing config.ini options, added ainb command
8/4/2011	New Release, firmware version 1.4.0